IBM DB2 10.1
for Linux, UNIX and Windows

pureXML - Handbuch

SC12-4684-00

IBM DB2 10.1
for Linux, UNIX and Windows

pureXML - Handbuch

SC12-4684-00

Hinweis

Vor Verwendung dieser Informationen und des darin beschriebenen Produkts sollten die allgemeinen Informationen in Anhang E, „Bemerkungen", auf Seite 543 gelesen werden.

Diese Veröffentlichung ist eine Übersetzung des Handbuchs
IBM DB2 10.1 for Linux, UNIX, and Windows, pureXML Guide,
IBM Form SC27-3892-00,
herausgegeben von International Business Machines Corporation, USA

Informationen, die nur für bestimmte Länder Gültigkeit haben und für Deutschland, Österreich und die Schweiz nicht zutreffen, wurden in dieser Veröffentlichung im Originaltext übernommen.

Möglicherweise sind nicht alle in dieser Übersetzung aufgeführten Produkte in Deutschland angekündigt und verfügbar; vor Entscheidungen empfiehlt sich der Kontakt mit der zuständigen IBM Geschäftsstelle.

Änderung des Textes bleibt vorbehalten.

Herausgegeben von:
TSC Germany
Kst. 2877
Mai 2012

Inhaltsverzeichnis

Zu diesem Handbuch

Das pureXML-Handbuch enthält eine Beschreibung zur Arbeit mit XML-Daten in DB2-Datenbanken. Es bietet Informationen zu den XML-Datentypen und zur XML-Speicherung, zur Arbeit mit XML-Daten unter Verwendung von SQL und XQuery sowie zur Indexierung von XML-Daten unter Berücksichtigung von Leistungsaspekten. In weiteren Themen werden die pureXML-Anwendungsentwicklung, das Versetzen von Daten und die Dekomposition von XML-Daten in relationale Formate behandelt.

Kapitel 1. Übersicht über pureXML - DB2 als XML-Datenbank

Mithilfe der Komponente pureXML können korrekt formatierte XML-Dokumente in Datenbanktabellenspalten, die den Datentyp XML aufweisen, gespeichert werden. Durch die Speicherung von XML-Daten in XML-Spalten werden die Daten in ihrem nativen hierarchischen Format beibehalten und nicht im Textformat abgelegt oder einem anderen Datenmodell zugeordnet.

Da der pureXML-Datenspeicher vollständig integriert ist, können Sie auf die gespeicherten XML-Daten zugreifen und diese verwalten, indem Sie die vorhandene DB2-Datenbankserverfunktionalität nutzen.

Die Speicherung von XML-Daten in deren nativen hierarchischen Format ermöglicht die effiziente Ausführung von Such-, Abruf- und Aktualisierungsoperationen für XML-Daten. Zum Abfragen und Aktualisieren von XML-Daten kann XQuery, SQL oder eine Kombination dieser beiden Sprachen verwendet werden. SQL-Funktionen, die XML-Daten zurückgeben oder mit XML-Argumenten arbeiten (und deswegen als SQL/XML-Funktionen bezeichnet werden), ermöglichen außerdem die Erstellung und Veröffentlichung von XML-Daten auf der Basis von Werten, die aus der Datenbank abgerufen werden.

Abfragen und Aktualisieren

XML-Dokumente, die in XML-Spalten gespeichert sind, können mithilfe der folgenden Methoden abgefragt und aktualisiert werden:

XQuery
Bei XQuery handelt es sich um eine allgemeine Sprache zum Interpretieren, Abrufen und Ändern von XML-Daten. Mithilfe des DB2-Datenbankservers kann XQuery entweder direkt oder über SQL aufgerufen werden. Da die XML-Daten in DB2-Tabellen und -Sichten gespeichert sind, werden Funktionen zum Extrahieren der XML-Daten aus angegebenen Tabellen und Sichten bereitgestellt. Hierzu wird die gewünschte Tabelle oder Sicht entweder direkt angegeben, oder es wird eine entsprechende SQL-Abfrage definiert. XQuery unterstützt verschiedene Ausdrücke für das Verarbeiten von XML-Daten, für das Aktualisieren von vorhandenen XML-Objekten (wie beispielsweise Elementen und Attributen) und für das Erstellen neuer XML-Objekte. Die Programmierschnittstelle für XQuery bietet Funktionen, die Ähnlichkeiten mit den entsprechenden SQL-Funktionen aufweisen und zur Ausführung von Abfragen und zum Abrufen von Ergebnissen verwendet werden können.

SQL-Anweisungen und SQL/XML-Funktionen
Zahlreiche SQL-Anweisungen unterstützen den Datentyp XML. Auf diese Weise können Sie viele allgemeine Datenbankoperationen mit XML-Daten ausführen. Hierzu gehören z. B. das Erstellen von Tabellen mit XML-Spalten, das Hinzufügen von XML-Spalten zu vorhandenen Tabellen, das Erstellen von Indizes zu XML-Spalten und von Triggern für Tabellen mit XML-Spalten sowie das Einfügen, Aktualisieren oder Löschen von XML-Dokumenten. Die Gruppe der SQL/XML-Funktionen, -Ausdrücke und -Spezifikationen, die vom DB2-Datenbankserver unterstützt werden, wurde erweitert, um alle Vorteile des Datentyps XML voll nutzen zu können.

XQuery kann über eine SQL-Abfrage aufgerufen werden. In diesem Fall kann die SQL-Abfrage Daten in Form gebundener Variablen an XQuery übergeben.

Anwendungsentwicklung

Unterstützung für die Anwendungsentwicklung wird von verschiedenen Programmiersprachen sowie durch SQL-Prozeduren und externe Prozeduren bereitgestellt:

Unterstützung für Programmiersprachen
Die Anwendungsentwicklungsunterstützung der neuen Komponente pureXML ermöglicht es Anwendungen, die Funktionen für den Zugriff auf XML-Daten und relationale Daten und deren Speicherung zu kombinieren. Die folgenden Programmiersprachen unterstützen den Datentyp XML:

- C oder C++ (eingebettetes SQL oder CLI)
- COBOL
- Java (JDBC oder SQLJ)
- C# und Visual Basic (IBM® Data Server Provider für .NET)
- PHP
- Perl

SQL-Prozeduren und externe Prozeduren
XML-Daten können an SQL-Prozeduren und externe Prozeduren übergeben werden, indem Parameter vom Datentyp XML in die Parametersignaturen von CREATE PROCEDURE eingeschlossen werden. Vorhandene Prozedurenfunktionen unterstützen die Implementierung des prozeduralen Logikablaufs für SQL-Anweisungen, die XML-Werte generieren oder diese verwenden, sowie die temporäre Speicherung von XML-Datenwerten in Variablen.

Verwaltung

Die Komponente pureXML stellt ein Repository für die Verwaltung der URI-Abhängigkeiten von XML-Dokumenten bereit und ermöglicht das Versetzen von XML-Daten zwecks Datenbankverwaltung:

XML-Schema-Repository (XSR)
Das XML-Schema-Repository (XSR) ist ein Repository für alle XML-Artefakte, die zur Verarbeitung von XML-Instanzdokumenten benötigt werden, die in XML-Spalten gespeichert sind. Es dient zur Speicherung von XML-Schemata, Dokumenttypdeklarationen (DTDs) sowie externer Entitäten, auf die in XML-Dokumenten verwiesen wird.

Dienstprogramme zum Importieren (IMPORT), zum Exportieren (EXPORT) und zum Laden (LOAD)
Die Dienstprogramme IMPORT, EXPORT und LOAD wurden aktualisiert und unterstützen nun den nativen XML-Datentyp. Diese Dienstprogramme verarbeiten XML-Daten wie LOB-Daten, d. h., beide Datentypen werden außerhalb der eigentlichen Tabelle gespeichert. Darüber hinaus steht Anwendungsentwicklungsunterstützung für das Importieren, Exportieren und Laden von XML-Daten zur Verfügung. Diese Unterstützung wird über die aktualisierten Anwendungsprogrammierungsschnittstellen (APIs) db2Import, db2Export und db2Load bereitgestellt. Diese aktualisierten Dienstprogramme ermöglichen Ihnen das Versetzen von Daten für in XML-Spalten gespeicherte XML-Dokumente, das in ähnlicher Weise ausgeführt wird wie die Datenversetzung bei relationalen Daten.

Leistung

Beim Arbeiten mit XML-Dokumenten, die in XML-Spalten gespeichert sind, stehen Ihnen zahlreiche leistungsorientierte Funktionen zur Verfügung:

Indizes zu XML-Daten
Die Indexierungsunterstützung steht für Daten zur Verfügung, die in XML-Spalten gespeichert sind. Die Verwendung eines Index zu XML-Daten kann zur Verbesserung der Effizienz von Abfragen beitragen, die für XML-Dokumente abgesetzt werden. Ähnlich wie ein relationaler Index führt auch ein Index für XML-Daten eine Indexierung für Spalten durch. Der Unterschied besteht jedoch darin, dass ein relationaler Index die gesamte Spalte indexiert, wohingegen ein Index für XML-Daten bei der Indexierung nur einen Teil der Spalte berücksichtigt. Sie können angeben, welche Teile einer XML-Spalte indexiert werden sollen, indem Sie ein XML-Muster angeben, bei dem es sich um einen eingeschränkten XPath-Ausdruck handelt.

Optimierungsprogramm
Das Optimierungsprogramm wurde aktualisiert und unterstützt jetzt die Auswertung von SQL-, XQuery- und SQL/XML-Funktionen, die XQuery-Komponenten einbetten. Die Funktionen werden hierbei auf der Basis von XML-Daten und relationalen Daten ausgewertet. Das Optimierungsprogramm verwendet Statistikdaten, die zu den XML-Daten erfasst wurden, sowie Daten aus Indizes zu XML-Daten, um auf der Basis dieser Informationen effiziente Abfrageausführungspläne zu erstellen.

EXPLAIN-Einrichtung
Die EXPLAIN-Funktion wurde aktualisiert und unterstützt nun die funktionalen SQL-Erweiterungen zur Abfrage von XML-Daten und zur Unterstützung von XQuery-Ausdrücken. Diese Aktualisierungen an der EXPLAIN-Funktion ermöglichen es Ihnen, den Status der vom DB2-Datenbankserver durchgeführten Auswertung von Abfrageanweisungen für XML-Daten schnell und einfach anzuzeigen.

Tools

Der XML-Datentyp wird in verschiedenen Tools wie beispielsweise im Befehlszeilenprozessor (CLP), in IBM Data Studio sowie in IBM Database Add-Ins for Microsoft Visual Studio unterstützt.

Dekomposition mithilfe eines mit Annotationen versehenen XML-Schemas

Die Komponente pureXML ermöglicht Ihnen die Speicherung von und den Zugriff auf XML-Daten im XML-Format, d. h. in deren hierarchischen Format; es kann jedoch in bestimmten Fällen erforderlich sein, auf XML-Daten in Form relationaler Daten zuzugreifen. Bei der Dekomposition mithilfe eines mit Annotationen versehenen XML-Schemas werden Dokumente auf der Basis von Annotationen zerlegt, die in einem XML-Schema angegeben sind.

XML-Datentyp

Mithilfe des XML-Datentyps können Sie Spalten einer Tabelle definieren und XML-Werte speichern. Bei allen XML-Werten muss es sich um korrekt formatierte XML-Dokumente handeln. Sie können diesen nativen Datentyp dazu verwenden, korrekt formatierte XML-Dokumente in ihrem nativen hierarchischen Format zusammen mit anderen relationalen Daten in der Datenbank zu speichern.

XML-Werte werden in einer internen Darstellung verarbeitet, bei der es sich nicht um ein Zeichenfolgeformat handelt und die nicht direkt mit Zeichenfolgewerten verglichen werden kann. Ein XML-Wert kann in einen serialisierten Zeichenfolgewert umgesetzt werden, der das XML-Dokument darstellt. Hierzu wird die Funktion XMLSERIALIZE verwendet oder der Wert an eine Anwendungsvariable eines XML-, Zeichenfolge- oder Binärtyps gebunden. Ein Zeichenfolgewert, der ein XML-Dokument darstellt, kann in ähnlicher Weise in einen XML-Wert umgesetzt werden. Hierzu wird die Funktion XMLPARSE verwendet oder ein Anwendungszeichenfolge-, Binär- oder XML-Anwendungstyp an einen XML-Wert gebunden. Bei SQL-Datenänderungsanweisungen (z. B. INSERT), die auch XML-Spalten betreffen, wird ein Zeichenfolge- oder ein Binärwert, der für ein XML-Dokument steht, in einen XML-Wert umgesetzt. Hierzu wird die Funktion XMLPARSE eingefügt. Ein XML-Wert kann implizit syntaktisch analysiert oder serialisiert werden, wenn ein Austausch mit Anwendungszeichenfolge- oder Binärdatentypen durchgeführt wird.

Es bestehen keine architekturbedingten Einschränkungen für die Höhe eines XML-Werts in einer Datenbank. Hierbei ist jedoch zu beachten, dass für serialisierte XML-Daten, die mit einem DB2-Datenbankserver ausgetauscht werden, eine effektive Beschränkung auf 2 GB gilt.

XML-Dokumente können mithilfe der SQL-Datenbearbeitungsanweisungen eingefügt, aktualisiert und gelöscht werden. Die Gültigkeitsprüfung für ein XML-Dokument auf der Basis eines XML-Schemas, die normalerweise während einer Einfüge- oder Aktualisierungsoperation ausgeführt wird, wird vom XML-Schema-Repository (XSR) unterstützt. Das DB2-Datenbanksystem stellt darüber hinaus Mechanismen zur Verfügung, mit denen XML-Werte erstellt und abgefragt sowie XML-Daten exportiert und importiert werden können. Für eine XML-Spalte kann ein Index für XML-Daten definiert werden, der eine verbesserte Leistung bei Suchoperationen in XML-Daten bietet. Die XML-Daten in Tabellen- oder Sichtspalten können als serialisierte Zeichenfolgedaten über verschiedene Anwendungsschnittstellen abgerufen werden.

Übersicht zur XML-Eingabe und -Ausgabe

Der DB2-Datenbankserver, der sowohl zur Verwaltung relationaler als auch zur Verwaltung von XML-Daten eingesetzt werden kann, bietet verschiedene Methoden zur Ein- und Ausgabe von XML-Dokumenten.

XML-Dokumente werden in Spalten gespeichert, die mit dem Datentyp XML definiert sind. Jede Zeile einer XML-Spalte enthält hierbei ein einziges, korrekt formatiertes XML-Dokument. Das gespeicherte Dokument wird in seinem hierarchischen Format gespeichert, wobei das XML-Datenmodell beibehalten wird. Das Dokument wird nicht als Text gespeichert und keinem anderen Datenmodell zugeordnet.

XML-Spalten können in Tabellen definiert werden, die auch Spalten anderer Typen enthalten, in denen relationale Daten gespeichert werden. In einer Tabelle können auch mehrere XML-Spalten definiert werden.

Eingabe

Abb. 1 auf Seite 5 zeigt die verschiedenen Möglichkeiten zur Integration von XML-Daten in das Datenbanksystem.

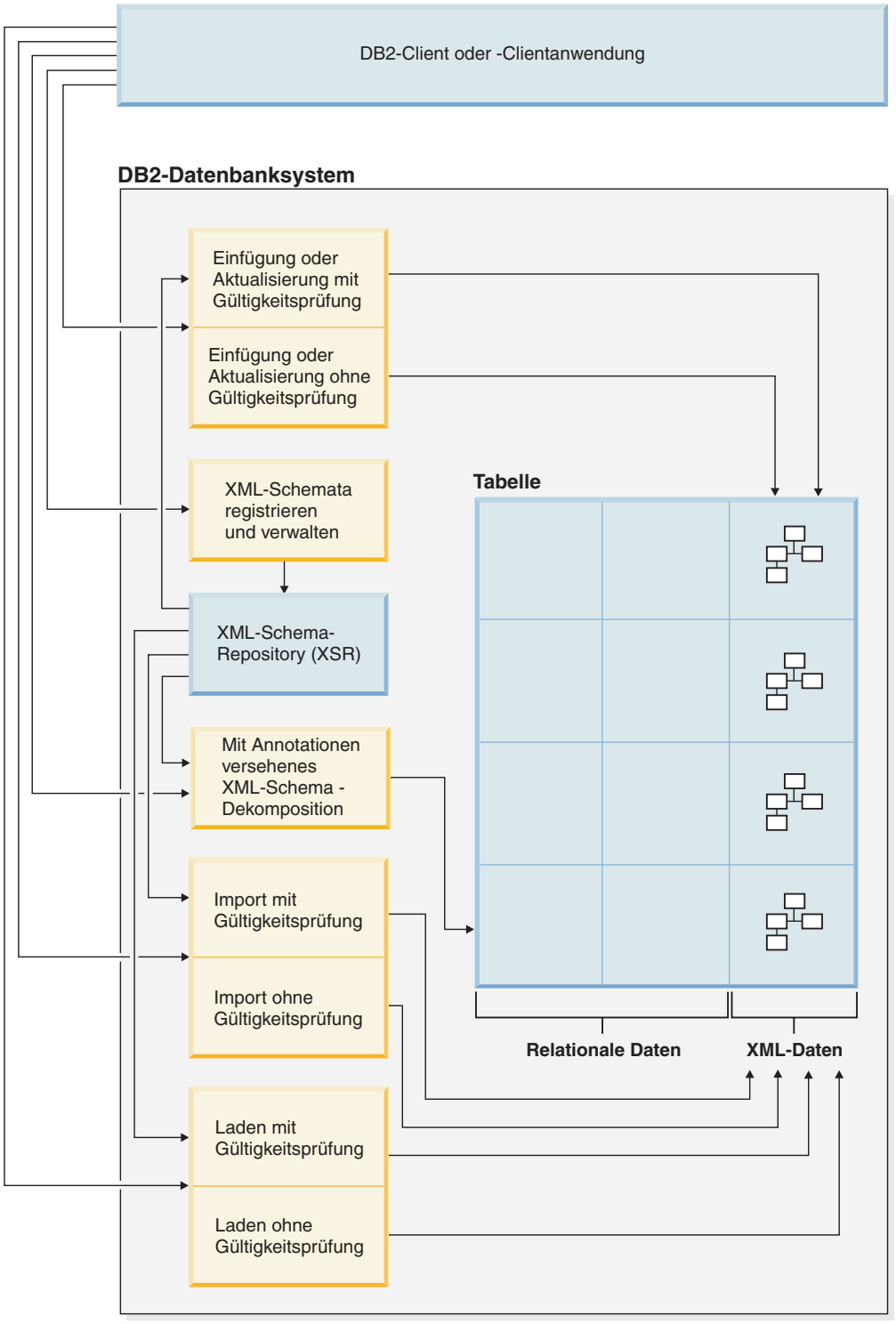


Abbildung 1. Methoden zur Eingabe von XML-Daten

Die verwendete Eingabemethode hängt von der zu erledigenden Task ab:

Einfügung oder Aktualisierung
Korrekt formatierte Dokumente werden in XML-Spalten mithilfe der SQL-Anweisung INSERT eingefügt. Ein Dokument ist dann korrekt formatiert, wenn es erfolgreich syntaktisch analysiert werden kann. Die Gültigkeitsprüfung der XML-Dokumente während einer Einfüge- oder Aktualisie-

rungsoperation ist optional. Wenn die Gültigkeitsprüfung durchgeführt wird, muss das XML-Schema zuerst im XML-Schema-Repository (XSR) registriert werden. Die Aktualisierung von Dokumenten erfolgt mithilfe der SQL-Anweisung UPDATE oder mithilfe von XQuery-Aktualisierungsausdrücken.

Dekomposition mithilfe eines mit Annotationen versehenen XML-Schemas
Daten aus XML-Dokumenten können zerlegt oder in relationalen Spalten und XML-Spalten gespeichert werden. Hierzu wird die Dekomposition mithilfe eines mit Annotationen versehenen XML-Schemas verwendet. Bei der Dekomposition werden Daten anhand von Annotationen, die zu XML-Schemadokumenten hinzugefügt werden, in Spalten gespeichert. Diese Annotationen stellen eine Zuordnung zwischen den Daten in XML-Dokumenten und den Spalten von Tabellen her.

XML-Schemadokumente, auf die durch die Dekompositionsfunktion verwiesen wird, werden im XML-Schema-Repository (XSR) gespeichert.

Import
XML-Dokumente können mit dem Dienstprogramm IMPORT in XML-Spalten importiert werden. Optional kann für die zu importierenden XML-Dokumente eine Gültigkeitsprüfung durchgeführt werden. Wenn die Gültigkeitsprüfung durchgeführt wird, muss das XML-Schema, anhand dessen die Dokumente geprüft werden, zuerst im XML-Schema-Repository (XSR) registriert werden.

Registrierung des XML-Schema-Repositorys (XSR)
Das XML-Schema-Repository dient zur Speicherung von XML-Schemata, die für die Gültigkeitsprüfung oder Dekomposition von XML-Dokumenten benutzt werden. Die Registrierung von XML-Schemata stellt normalerweise eine Voraussetzung für andere Tasks dar, die für XML-Dokumente ausgeführt werden sollen, die von diesen Schemata abhängen. XML-Schemata werden im XSR mithilfe von gespeicherten Prozeduren oder Befehlen registriert.

Ausgabe

Abb. 2 auf Seite 7 zeigt die verschiedenen Möglichkeiten zum Abrufen von XML-Daten aus dem Datenbanksystem.

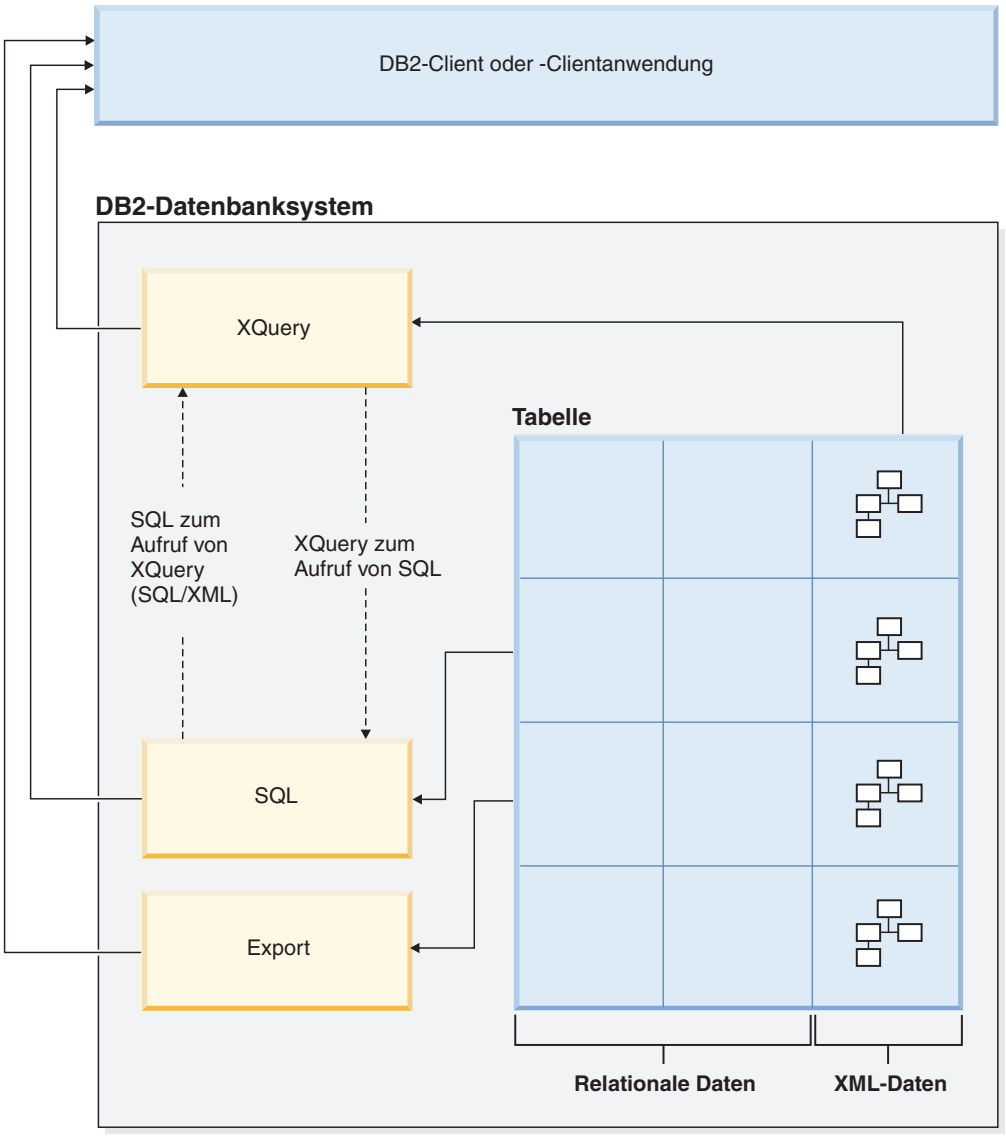


Abbildung 2. Methoden zur Ausgabe von XML-Daten

Die verwendete Ausgabemethode hängt von der zu erledigenden Task ab:

XQuery

Bei XQuery handelt es sich um eine Sprache, mit der Sie Daten in XML-Dokumenten abfragen können. Sie wurde für spezielle Anforderungen entwickelt, die für die Abfrage von XML-Daten gelten. Deren Struktur ist im Gegensatz zur Struktur relationaler Daten sehr variabel und kann nur schwierig vorhergesagt werden.

XQuery kann alleine aufgerufen werden oder SQL aufrufen, um über die XQuery-Funktionen db2-fn:xmlcolumn und db2-fn:sqlquery XML-Daten abzufragen, die in einer DB2-Datenbank gespeichert sind. db2-fn:xmlcolumn ruft eine vollständige XML-Spalte ab, wohingegen db2-fn:sqlquery XML-Werte abruft, die auf einer SQL-Fullselect-Operation basieren.

SQL Bei der Abfrage von XML-Daten mit einer SQL-Fullselect-Operation wird die Abfrage auf Spaltenebene durchgeführt. Aus diesem Grund können durch eine solche Abfrage nur vollständige XML-Dokumente zurückgegeben werden. Fragmente eines XML-Dokuments können nicht zurückgege-

ben werden, wenn nur SQL verwendet wird. Zur Abfrage von Daten in XML-Dokumenten muss XQuery verwendet werden. XQuery kann über SQL mithilfe der SQL/XML-Funktion XMLQUERY oder XMLTABLE aufgerufen werden. Alternativ hierzu kann auch das Vergleichselement XMLEXISTS benutzt werden. Die Funktion XMLQUERY gibt das Ergebnis eines XQuery-Ausdrucks als XML-Sequenz zurück. Die Funktion XMLTABLE gibt das Ergebnis eines XQuery-Ausdrucks als Tabelle zurück. Das SQL-Vergleichselement XMLEXISTS legt fest, ob ein XQuery-Ausdruck eine nicht leere Sequenz zurückgibt.

Es steht auch eine Reihe von Veröffentlichungsfunktionen zur Verfügung, um XML-Werte aus XML-Daten zu erstellen, die im DB2-Datenbankserver gespeichert sind. XML-Werte, die mit diesen Veröffentlichungsfunktionen erstellt werden, müssen keine korrekt formatierten XML-Dokumente sein.

Export XML-Dokumente können aus XML-Spalten mithilfe des Dienstprogramms EXPORT exportiert werden. Exportierte XML-Daten werden separat von den exportierten relationalen Daten in der Hauptdatendatei gespeichert. Details zu den einzelnen exportierten XML-Dokumenten werden nicht direkt in der Hauptdatei der exportierten Daten gespeichert. Diese Details werden in der Hauptdatendatei stattdessen durch eine XML-Datenkennung (XDS) dargestellt.

Vergleich zwischen dem XML-Modell und dem relationalen Modell

Beim Entwurf Ihrer Datenbanken müssen Sie entscheiden, ob sich Ihre Daten besser für das XML-Modell oder für das relationale Modell eignen. Nutzen Sie den Hybridcharakter von DB2-Datenbanken, d. h. die Möglichkeit zur Unterstützung sowohl von relationalen Daten als auch von XML-Daten in einer einzigen Datenbank.

In dieser Beschreibung werden zwar einige der Hauptunterschiede zwischen den Modellen und die jeweils zu berücksichtigenden Aspekte behandelt, doch spielen bei der Festlegung des am besten geeigneten Modells für eine bestimmte Implementierung zahlreiche Faktoren eine Rolle. Betrachten Sie die folgende Beschreibung lediglich als Richtlinie, um die Faktoren zu beurteilen, die sich auf Ihre spezifische Implementierung auswirken können.

Hauptunterschiede zwischen XML-Daten und relationalen Daten

XML-Daten sind hierarchisch aufgebaut, während relationale Daten in einem Modell logischer Beziehungen dargestellt werden.
Ein XML-Dokument enthält Informationen zu den Beziehungen von Datenelementen untereinander in Form einer Hierarchie. Beim relationalen Modell lassen sich lediglich solche Typen von Beziehungen angeben, die durch die Beziehungen übergeordneter und abhängiger Tabellen definierbar sind.

XML-Daten sind selbst-beschreibend, relationale Daten sind dies hingegen nicht. Ein XML-Dokument enthält nicht nur die Daten, sondern auch eine Kennzeichnung, die erläutert, worum es sich handelt. Ein einziges Dokument kann verschiedene Typen von Daten enthalten. Beim relationalen Modell wird der Inhalt der Daten durch die zugehörige Spaltendefinition festgelegt. Alle Daten einer Spalte müssen denselben Datentyp besitzen.

XML-Daten besitzen eine inhärente Anordnung, relationale Daten hingegen nicht. Bei einem XML-Dokument gilt die Annahme, dass die Reihenfolge, in der Datenelemente angegeben werden, die Reihenfolge der Daten im Dokument ist. Häufig gibt es keine andere Möglichkeit, die Reihenfolge inner-

halb eines Dokuments anzugeben. Bei relationalen Daten ist die Reihenfolge der Zeilen nicht garantiert, sofern nicht eine Klausel ORDER BY für eine oder mehrere Spalten angegeben wird.

Einflussfaktoren bei der Auswahl eines geeigneten Datenmodells

Die Art der zu speichernden Daten kann dabei helfen zu bestimmen, wie diese Daten gespeichert werden sollen. Wenn die Daten zum Beispiel eine natürliche Hierarchie aufweisen und selbst-beschreibend sind, können sie als XML-Daten gespeichert werden. Allerdings können auch noch andere Faktoren die Wahl eines geeigneten Modells beeinflussen:

Bei Bedarf an hoher Flexibilität
Relationale Tabellen basieren auf einem recht starren Modell. Zum Beispiel kann sich eine Normalisierung einer Tabelle in mehrere Tabellen oder auch eine Denormalisierung mehrerer Tabellen zu einer Tabelle sehr schwierig gestalten. Wenn sich die Datenstruktur häufig ändert, ist ihre Darstellung in Form von XML-Daten die geeignetere Wahl. XML-Schemata können beispielsweise im Laufe der Zeit weiterentwickelt werden.

Bei Bedarf an hoher Leistung beim Datenabruf
Mit der Serialisierung und Interpretation von XML-Daten ist einiger Aufwand verbunden. Wenn die Leistung eine wichtigere Rolle spielt als die Flexibilität, könnten sich relationale Daten als die bessere Wahl erweisen.

Bei einer späteren Verarbeitung der Daten als relationale Daten
Wenn die nachfolgende Verarbeitung der Daten davon abhängig ist, dass die Daten in einer relationalen Datenbank gespeichert werden, kann ein geeignetes Verfahren darin bestehen, Teile der Daten durch Dekomposition (XML-Zerlegung) als relationale Daten zu speichern. Ein Beispiel für einen solchen Fall ist eine OLAP-Verarbeitung (Online Analytical Processing), die auf Daten in einem Data-Warehouse angewendet wird. Wenn weitere Verarbeitungsschritte für das XML-Dokument als Ganzes erforderlich sind, kann in diesem Fall darüber hinaus eine geeignete Lösung darin bestehen, einige Daten als relationale Daten und zudem das gesamte XML-Dokument zu speichern.

Bei einer etwaigen Bedeutung der Datenkomponenten außerhalb einer Hierarchie
Daten können inhärent eine hierarchische Spezifik aufweisen, ohne dass die untergeordneten Komponenten notwendigerweise die übergeordneten Komponenten zur Angabe von Werten benötigen. Zum Beispiel kann eine Bestellung Teilenummern enthalten. Die Bestellungen mit den Teilenummern werden möglicherweise am besten in Form von XML-Dokumenten dargestellt. Die einzelnen Teilnummern besitzen jedoch jeweils eine zugeordnete Teilebeschreibung. Es kann vorteilhafter sein, die Teilebeschreibungen in einer relationalen Tabelle abzulegen, weil die Beziehung zwischen den Teilenummern und den Teilebeschreibungen von den Bestellungen, in denen die Teilenummern verwendet werden, logisch unabhängig ist.

Bei einem Geltungsbereich von Datenattributen für alle Daten oder nur für eine kleine Untergruppe der Daten
Einige Datengruppen besitzen eine große Anzahl möglicher Attribute, wobei jedoch nur eine kleine Anzahl dieser Attribute jeweils für einen bestimmten Datenwert gilt. In einem Einzelhandelskatalog können beispielsweise zahlreiche mögliche Datenattribute vorkommen, wie zum Beispiel Größe, Farbe, Gewicht, Material, Stil, Gewebe, Anschlusswerte oder Kraftstoffbedarf. Für jedes bestimmte Element im Katalog ist nur eine Untermenge dieser Attribute relevant: Anschlusswerte sind für eine Elektrotisch-

säge, jedoch nicht für einen Mantel von Bedeutung. In einem relationalen Modell lässt sich diese Art von Daten nur schwer darstellen und in einer für Suchen geeigneten Form speichern, während dies in einem XML-Modell relativ einfach ist.

Bei einem hohen Verhältnis von Datenkomplexität zu Datenvolumen
In vielen Situationen werden hoch strukturierte Informationen in kleinen Mengen benötigt. Die Darstellung solcher Daten in einem relationalen Modell kann komplexe Sternschemata erfordern, in denen jede Dimensionstabelle mit vielen weiteren Dimensionstabellen verknüpft wird und die meisten Tabellen nur wenige Zeilen enthalten. Eine bessere Methode zur Darstellung dieser Daten besteht darin, eine einzige Tabelle mit einer XML-Spalte zu verwenden und Sichten für diese Tabelle zu erstellen, die jeweils eine Dimension darstellen.

Bei erforderlicher referenzieller Integrität
XML-Spalten können nicht als Teil von referenziellen Integritätsbedingungen definiert werden. Wenn Werte in XML-Dokumenten in referenzielle Integritätsbedingungen einzubinden sind, sollten Sie die Daten daher als relationale Daten speichern.

Bei häufig erforderlichen Datenaktualisierungen
XML-Daten in einer XML-Spalte können nur in der Art aktualisiert werden, dass vollständige Dokumente ersetzt werden. Wenn häufig kleine Fragmente sehr großer Dokumente für eine große Anzahl von Zeilen aktualisiert werden müssen, kann es effizienter sein, die Daten in Nicht-XML-Spalten zu speichern. Wenn hingegen kleine Dokumente und nur wenige Dokumente gleichzeitig zu aktualisieren sind, kann sich die Speicherung in XML-Spalten ebenfalls als effizient erweisen.

XQuery- und XPath-Datenmodell

XQuery-Ausdrücke arbeiten mit Instanzen des XQuery- und des XPath-Datenmodells (XDM) und geben Instanzen des Datenmodells zurück.

Das XQuery- und XPath-Datenmodell stellt eine abstrakte Darstellung eines oder mehrerer XML-Dokumente oder -Fragmente zur Verfügung. Das Datenmodell definiert alle gültigen Werte von Ausdrücken in XQuery, einschließlich der Werte, die während der Zwischenberechnungen verwendet werden.

Das Parsing (Syntaxanalyse) von XML-Daten in das XQuery- und XPath-Datenmodell und die Gültigkeitsprüfung der Daten auf der Basis eines Schemas werden vor der Verarbeitung der Daten durch XQuery ausgeführt. Während der Datenmodellgenerierung wird das XML-Eingabedokument syntaktisch analysiert und in eine Instanz des XQuery- und XPath-Datenmodells konvertiert. Das Dokument kann mit oder ohne Gültigkeitsprüfung syntaktisch analysiert werden.

Das XQuery- und XPath-Datenmodell wird anhand von Sequenzen aus atomaren Werten und Knoten beschrieben.

Sequenzen und Elemente

Eine Instanz des XQuery- und XPath-Datenmodells (XDM) ist eine Sequenz. Eine *Sequenz* ist eine sortierte Gruppe aus null oder mehr Elementen. Ein *Element* ist entweder ein atomarer Wert oder ein Knoten.

Eine Sequenz kann Knoten, atomare Werte oder eine Kombination aus Knoten und atomaren Werten enthalten. Jeder Eintrag in der folgenden Liste beispielsweise ist eine Sequenz:

- `36`
- `<dog/>`
- `(2, 3, 4)`
- `(36, <dog/>, "cat")`
- `()`

Zusätzlich zu den Einträgen in der Liste, ist auch ein XML-Dokument, das in einer XML-Spalte in einer DB2-Datenbank gespeichert ist, eine Sequenz.

In den Beispielen wird zur Darstellung von Sequenzen eine Notation verwendet, die der Syntax entspricht, die zum Erstellen von Sequenzen in XQuery verwendet wird:

- Jedes Element in der Sequenz wird durch ein Komma von den restlichen Elementen getrennt.
- Eine ganze Sequenz wird in runde Klammern eingeschlossen.
- Zur Darstellung einer leeren Sequenz wird ein Paar leerer runder Klammern verwendet.
- Ein einzelnes Element, das alleine steht, entspricht einer Sequenz, die ein Element enthält.

  Es gibt z. B. keinen Unterschied zwischen der Sequenz (36) und dem atomaren Wert 36.

Sequenzen dürfen nicht verschachtelt sein. Wenn zwei Sequenzen kombiniert werden, dann wird als Ergebnis immer eine abgewickelte Sequenz von Knoten und atomaren Werten generiert. Wenn Sie z. B. die Sequenz (2, 3) an die Sequenz (3, 5, 6) anfügen, dann wird eine Sequenz generiert, die (3, 5, 6, 2, 3) lautet. Durch die Kombination dieser Sequenzen kann nicht die Sequenz (3, 5, 6, (2, 3)) generiert werden, da verschachtelte Sequenzen niemals auftreten.

Eine Sequenz, die null Elemente enthält, wird als *leere Sequenz* bezeichnet. Leere Sequenzen können zur Darstellung fehlender oder unbekannter Informationen verwendet werden.

Atomare Werte

Als *atomarer Wert* wird eine Instanz eines der integrierten atomaren Datentypen bezeichnet, die mit XML-Schemata definiert werden können. Diese Datentypen umfassen Zeichenfolgen, ganze Zahlen (Integer), Dezimalzahlen, Datumsangaben und weitere atomare Typen. Diese Typen werden als atomar bezeichnet, weil sie nicht weiter in kleinere Einheiten unterteilt werden können.

Im Gegensatz zu Knoten verfügen atomare Werte nicht über eine Identität. Jede Instanz eines atomaren Wertes (z. B. die ganze Zahl 7) ist identisch mit allen anderen Instanzen dieses Wertes.

Die folgenden Beispiele zeigen verschiedene Möglichkeiten zur Erstellung von atomaren Werten:

- Extraktion aus Knoten über den so genannten Atomisierungsprozess. Die Atomisierung wird von Ausdrücken immer dann verwendet, wenn eine Sequenz atomarer Werte erforderlich ist.

- Angabe als numerischer oder Zeichenfolgeliteralwert. Literalwerte werden von XQuery als atomare Werte interpretiert. Die folgenden Literalwerte werden z. B. als atomare Werte interpretiert:
  - "dies ist eine zeichenfolge" (Der Typ lautet xs:string)
  - 45 (Der Typ lautet xs:integer)
  - 1,44 (Der Typ lautet xs:decimal)
- Berechnung mit Konstruktorfunktionen. Die folgende Konstruktorfunktion beispielsweise kann zur Erstellung eines Wertes des Typs 'xs:date' auf der Basis der Zeichenfolge "2005-01-01" verwendet werden:

  ```
  xs:date("2005-01-01")
  ```
- Rückgabe durch die integrierten Funktionen fn:true() und fn:false(). Diese Funktionen geben die Booleschen Werte `true` (wahr) und `false` (falsch) zurück. Diese Werte können nicht als Literalwerte dargestellt werden.
- Rückgabe durch viele Arten von Ausdrücken wie z. B. arithmetischen Ausdrücken und logischen Ausdrücken.

Knotenhierarchien

Die Knoten einer Sequenz bilden mindestens eine *Hierarchie* oder *Baumstruktur*, die aus einem Rootknoten und allen anderen Knoten besteht, die über den Rootknoten direkt oder indirekt aufgerufen werden können.

Jeder Knoten gehört zu genau einer Hierarchie, und jede Hierarchie verfügt über genau einen Rootknoten. DB2 unterstützt sechs verschiedene Knotensorten: Dokument-, Element-, Attribut-, Text-, Verarbeitungsanweisungs- und Kommentarknoten.

Im Folgenden ist das XML-Dokument `products.xml` aufgeführt, das ein Rootelement mit dem Namen `products` (Produkte) enthält, das verschiedene Produktelemente (`product`) umfasst. Jedes dieser `product`-Elemente verfügt über ein Attribut mit dem Namen `pid` (Produkt-ID) und über ein untergeordnetes Element (Kind) mit dem Namen `description` (Beschreibung). Das Element `description` enthält untergeordnete Elemente mit dem Namen `name` und `price` (Preis).

```
<products>
  <product pid="10">
    <description>
      <name>Fleecejacke</name>
      <price>19,99</price>
    </description>
  </product>
  <product pid="11">
    <description>
      <name>Nylonstrumpfhosen</name>
      <price>9,99</price>
    </description>
  </product>
</products>
```

Abb. 3 auf Seite 13 zeigt eine vereinfachte Darstellung des Datenmodells für `products.xml`. Die Abbildung umfasst einen Dokumentknoten (D), Elementknoten (E), Attributknoten (A) sowie Textknoten (T).

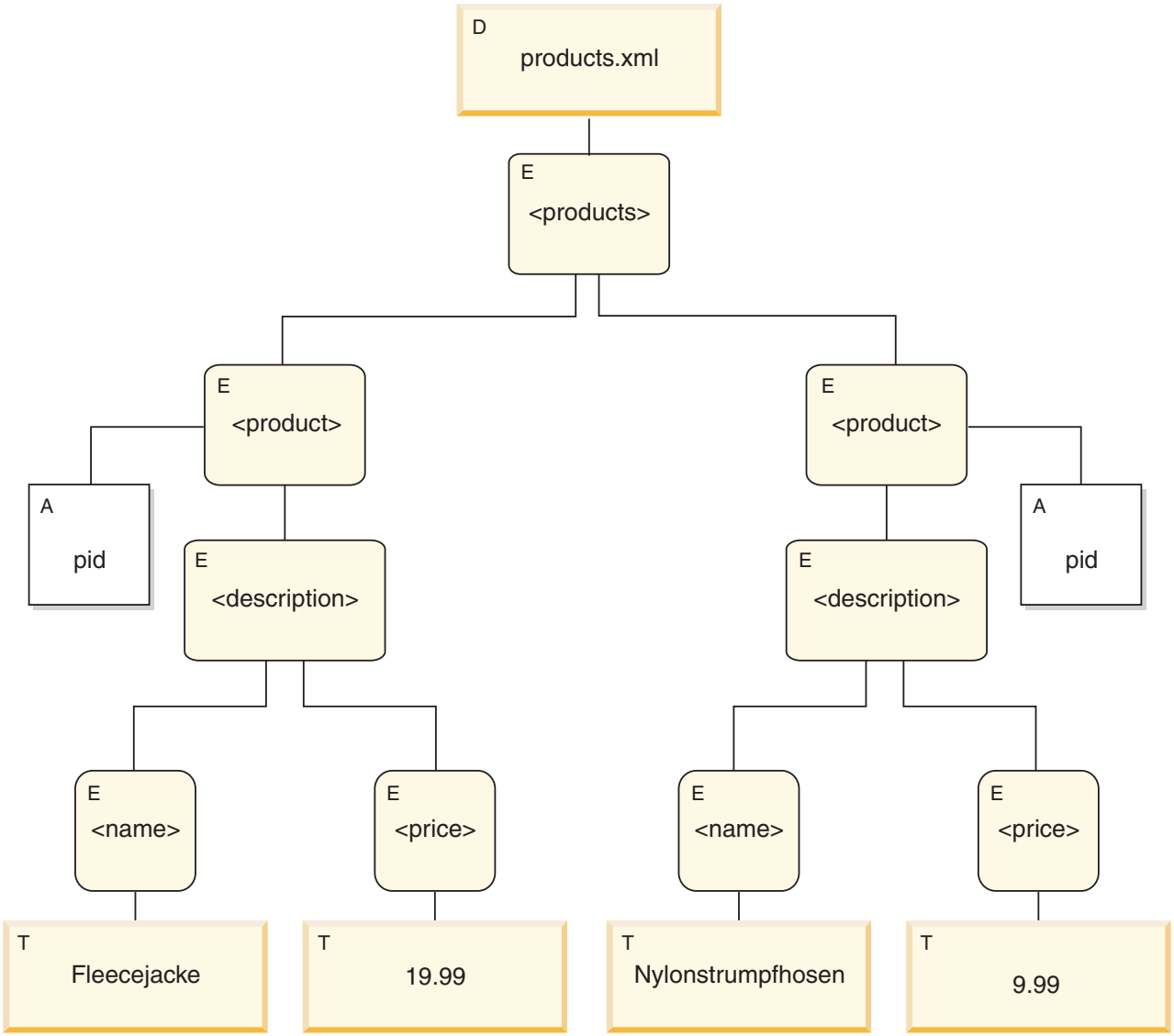


Abbildung 3. Datenmodelldiagramm für das Dokument products.xml

Wie im Beispiel dargestellt, kann ein Knoten weitere Knoten als untergeordnete Elemente umfassen und so eine oder mehrere *Knotenhierarchien* bilden. Im vorliegenden Beispiel stellt das Element `product` ein untergeordnetes Element von `products` dar. Das Element `description` ist ein untergeordnetes Element von `product`. Die Elemente `name` und `price` sind untergeordnete Elemente des Elements `description`. Der Textknoten mit dem Wert `Fleecejacke` ist ein untergeordnetes Element des Elementknotens `name`, und der Textknoten `19,99` ist ein untergeordnetes Element von `price`.

Knoteneigenschaften

Jeder Knoten verfügt über bestimmte *Eigenschaften*, in denen die Merkmale dieses Knotens beschrieben sind. In den Knoteneigenschaften können z. B. der Knotenname, seine untergeordneten und übergeordneten Elemente, Attribute und weitere Informationen festgelegt sein, die den Knoten beschreiben. Die Knotensorte legt hierbei fest, welche Eigenschaften für bestimmte Knoten definiert sind.

Ein Knoten kann über eine oder mehrere der nachfolgend aufgeführten Eigenschaften verfügen:

Knotenname (node-name)
Der Name des Knotens in Form eines QName.

Übergeordnetes Element (parent)
Der Knoten, der als übergeordnetes Element des aktuellen Knotens definiert ist.

Typname (type-name)
Der dynamische Typ (Laufzeittyp) des Knotens (wird auch als *Typenannotation*) bezeichnet.

Untergeordnete Elemente (children)
Die Knotensequenz, die untergeordnete Elemente des aktuellen Knotens darstellen.

Attribute
Die Gruppe der Attributknoten, die zum aktuellen Knoten gehören.

Zeichenfolgewert (string-value)
Ein Zeichenfolgewert, der aus dem Knoten extrahiert werden kann.

Typisierter Wert (typed-value)
Eine Sequenz von null oder mehr atomaren Werten, die aus dem Knoten extrahiert werden können.

Gültige Namensbereiche (in-scope namespaces)
Die gültigen Namensbereiche, die dem Knoten zugeordnet sind.

Inhalt (content)
Der Inhalt des Knotens.

Knotensorten

DB2 unterstützt sechs verschiedene Knotensorten: Dokument-, Element-, Attribut-, Text-, Verarbeitungsanweisungs- und Kommentarknoten.

Dokumentknoten

Ein Dokumentknoten dient zur Einbindung eines XML-Dokuments.

Ein Dokumentknoten kann null oder mehr untergeordnete Elemente (Kinder) aufweisen. Diese untergeordneten Elemente können Elementknoten, Verarbeitungsanweisungsknoten, Kommentar- und Textknoten umfassen.

Der Zeichenfolgewert eines Dokumentknotens entspricht dem verknüpften Inhalt aller untergeordneten Textknoten in der Reihenfolge der Dokumente. Der Typ des Zeichenfolgewertes lautet xs:string. Der typisierte Wert eines Dokumentknotens stimmt mit seinem Zeichenfolgewert überein. Eine Ausnahme bildet hierbei allerdings die Tatsache, dass der Typ des typisierten Wertes xdt:untypedAtomic lautet.

Ein Dokumentknoten verfügt über die folgenden Knoteneigenschaften:

- Untergeordnete Elemente (children), möglicherweise nicht angegeben
- Zeichenfolgewert (string-value)
- Typisierter Wert (typed-value)

Dokumentknoten können in XQuery-Ausdrücken mithilfe berechneter Konstruktoren erstellt werden. Eine Sequenz aus Dokumentknoten kann auch von der Funktion db2-fn:xmlcolumn zurückgegeben werden.

Elementknoten

Ein Elementknoten dient zur Einbindung eines XML-Elements.

Ein Elementknoten verfügt über null übergeordnete Elemente bzw. ein übergeordnetes Element sowie über null oder mehr untergeordnete Elemente. Diese untergeordneten Elemente können Elementknoten, Verarbeitungsanweisungsknoten, Kommentar- und Textknoten umfassen. Dokument- und Attributknoten werden nie als untergeordnete Elemente von Elementknoten verwendet. Allerdings wird ein Elementknoten als übergeordnetes Element seiner Attribute interpretiert. Die Attribute eines Elementknotens müssen über eindeutige QNames verfügen.

Ein Elementknoten weist die folgenden Knoteneigenschaften auf:

- Knotenname (node-name)
- Übergeordnetes Element (parent), möglicherweise nicht angegeben
- Typname (type-name)
- Untergeordnete Elemente (children), möglicherweise nicht angegeben
- Attribute (attributes), möglicherweise nicht angegeben
- Zeichenfolgewert (string-value)
- Typisierter Wert (typed-value)
- Bereichsinterne Namensbereiche (in-scope-namespaces)

Elementknoten können in XQuery-Ausdrücken mithilfe direkter oder berechneter Konstruktoren erstellt werden.

Die Eigenschaft 'type-name' eines Elementknotens gibt die Beziehung zwischen seinem typisierten Wert und seinem Zeichenfolgewert an. Wenn ein Elementknoten z. B. die type-name-Eigenschaft xs:decimal und den Zeichenfolgewert '47.5' aufweist, dann wird als typisierter Wert der Dezimalwert 47.5 verwendet. Wenn die type-name-Eigenschaft eines Elementknotens xdt:untyped lautet, dann stimmt der typisierte Wert des Elements mit seinem Zeichenfolgewert überein und weist den Typ xdt:untypedAtomic auf.

Attributknoten

Ein Attributknoten stellt ein XML-Attribut dar.

Ein Attributknoten verfügt entweder über kein oder ein übergeordnetes Element. Der Elementknoten, der als Eigner eines Attributs definiert ist, wird als übergeordnetes Element behandelt, obwohl ein Attributknoten kein untergeordnetes Element dieses Elements ist.

Ein Attributknoten weist die folgenden Knoteneigenschaften auf:

- Knotenname (node-name)
- Übergeordnetes Element (parent), möglicherweise nicht angegeben
- Typname (type-name)
- Zeichenfolgewert (string-value)
- Typisierter Wert (typed-value)

Attributknoten können in XQuery-Ausdrücken mithilfe direkter oder berechneter Konstruktoren erstellt werden.

Die Eigenschaft 'type-name' eines Attributknotens gibt die Beziehung zwischen seinem typisierten Wert und seinem Zeichenfolgewert an. Wenn ein Attributknoten

z. B. die type-name-Eigenschaft xs:decimal und den Zeichenfolgewert "47,5" aufweist, dann wird als typisierter Wert der Dezimalwert 47,5 verwendet.

Textknoten

Ein Textknoten dient zur Einbindung von Inhalten mit XML-Zeichen.

Ein Textknoten kann null oder ein übergeordnetes Element aufweisen. Textknoten, bei denen es sich um untergeordnete Elemente eines Dokument- oder Elementknotens handelt, treten niemals als benachbarte gleichgeordnete Elemente auf. Wenn ein Dokument- oder Elementknoten erstellt wird, werden alle benachbarten, gleichgeordneten Textknoten zu einem einzigen Textknoten kombiniert. Wenn der resultierende Textknoten leer ist, wird er gelöscht.

Textknoten weisen die folgenden Knoteneigenschaften auf:

- Inhalt (content), möglicherweise nicht angegeben
- Übergeordnetes Element (parent), möglicherweise nicht angegeben

Textknoten können in XQuery-Ausdrücken mithilfe berechneter Konstruktoren oder mit der Aktion eines direkten Elementkonstruktors erstellt werden.

Verarbeitungsanweisungsknoten

Ein Verarbeitungsanweisungsknoten dient zum Einbinden einer XML-Verarbeitungsanweisung.

Ein Verarbeitungsanweisungsknoten kann null übergeordnete Elemente oder ein übergeordnetes Element aufweisen. Die Verwendung der Zeichenfolge ?> ist in einer Verarbeitungsanweisung nicht zulässig. Als Zieleinheit einer Verarbeitungsanweisung muss ein Name ohne Doppelpunkte (NCName) angegeben sein. Die Zieleinheit wird verwendet, um die Anwendung zu identifizieren, für die die Anweisung bestimmt ist.

Ein Verarbeitungsanweisungsknoten verfügt über die folgenden Knoteneigenschaften:

- Ziel (target)
- Inhalt (content)
- Übergeordnetes Element (parent), möglicherweise nicht angegeben

Verarbeitungsanweisungsknoten können in XQuery-Ausdrücken mithilfe direkter oder berechneter Konstruktoren erstellt werden.

Kommentarknoten

Ein Kommentarknoten dient zur Einbindung eines XML-Kommentars.

Ein Kommentarknoten kann null übergeordnete Elemente oder ein übergeordnetes Element aufweisen. Der Inhalt eines Kommentarknotens darf weder die Zeichenfolge "--" (zwei Silbentrennstriche) noch einen Silbentrennstrich (-) als letztes Zeichen enthalten.

Ein Kommentarknoten verfügt über die folgenden Knoteneigenschaften:

- Inhalt (content)
- Übergeordnetes Element (parent), möglicherweise leer

Kommentarknoten können in XQuery-Ausdrücken mithilfe direkter oder berechneter Konstruktoren erstellt werden.

Dokumentreihenfolge der Knoten

Alle Knoten in einer Hierarchie entsprechen einer Reihenfolge, die als *Dokumentreihenfolge* bezeichnet wird und in der alle Knoten vor den zugehörigen untergeordneten Elementen dargestellt werden. Die Dokumentreihenfolge entspricht der Reihenfolge, in der die Knoten angezeigt werden, wenn die Knotenhierarchie im serialisierten XML-Format dargestellt wird.

Knoten in einer Hierarchie werden in der folgenden Reihenfolge dargestellt:

- Der Rootknoten ist der erste Knoten.
- Elementknoten werden vor den zugehörigen untergeordneten Elementen dargestellt.
- Attributknoten werden direkt nach dem Elementknoten dargestellt, dem sie zugeordnet sind. Die relative Reihenfolge der Attributknoten ist beliebig, diese Reihenfolge wird jedoch während der Verarbeitung einer Abfrage nicht geändert.
- Die relative Reihenfolge der gleichgeordneten Elemente wird von ihrer Reihenfolge in der Knotenhierarchie bestimmt.
- Untergeordnete Elemente und Nachkommen eines Knotens werden vor den gleichgeordneten Elementen dargestellt, die auf den Knoten folgen.

Knotenidentität

Jeder Knoten hat eine eindeutige Identität. Zwei Knoten können anhand dieser Identität auch dann unterschieden werden, wenn sie identische Namen und Werte aufweisen. Im Gegensatz hierzu verfügen atomare Werte nicht über eine Identität.

Die Knotenidentität ist nicht gleichbedeutend mit einem Attribut vom Typ ID. Einem Element in einem XML-Dokument kann vom Dokumentautor ein Attribut vom Typ ID zugeordnet werden. Eine Knotenidentität wird hingegen jedem Knoten vom System automatisch zugeordnet, kann vom Benutzer jedoch nicht direkt angezeigt werden.

Die Knotenidentität wird zur Verarbeitung folgender Ausdruckstypen verwendet:

- Knotenvergleiche. Die Knotenidentität wird vom Operator **`is`** verwendet, um festzustellen, ob zwei Knoten über dieselbe Identität verfügen.
- Pfadausdrücke. Die Knotenidentität wird von Pfadausdrücken verwendet, um doppelte Knoten zu eliminieren.
- Sequenzausdrücke. Die Knotenidentität wird vom Operator **`union`**, **`intersect`** oder **`except`** verwendet, um doppelte Knoten zu eliminieren.

Typisierte Werte und Zeichenfolgewerte von Knoten

Jeder Knoten verfügt sowohl über einen *typisierten Wert* als auch über einen *Zeichenfolgewert*. Diese beiden Knoteneigenschaften werden in den Definitionen bestimmter XQuery-Operationen (beispielsweise bei der Atomisierung) und -Funktionen (beispielsweise fn:data, fn:string und fn:deep-equal) verwendet.

Tabelle 1. Zeichenfolgewerte und typisierte Werte von Knoten

Knotensorte	Zeichenfolgewert	Typisierter Wert
Dokument	Eine Instanz des Datentyps xs:string, die den verknüpften Inhalt aller untergeordneten Textknoten in Dokumentreihenfolge enthält.	Eine Instanz des Datentyps xdt:untypedAtomic, die den verknüpften Inhalt aller untergeordneten Textknoten in Dokumentreihenfolge enthält.
Element in einem XML-Dokument	Eine Instanz des Datentyps xs:string, die den verknüpften Inhalt aller untergeordneten Textknoten in Dokumentreihenfolge enthält.	Eine Instanz des Datentyps xdt:untypedAtomic, die den verknüpften Inhalt aller untergeordneten Textknoten in Dokumentreihenfolge enthält.

Tabelle 1. Zeichenfolgewerte und typisierte Werte von Knoten (Forts.)

Knotensorte	Zeichenfolgewert	Typisierter Wert
Attribut in einem XML-Dokument	Eine Instanz des Datentyps xs:string, die den Attributwert im ursprünglichen XML-Dokument darstellt.	Eine Instanz des Datentyps xdt:untypedAtomic, die den Attributwert im ursprünglichen XML-Dokument darstellt.
Text	Der Inhalt als Instanz des Datentyps xs:string.	Der Inhalt als Instanz des Datentyps xdt:untypedAtomic.
Kommentar	Der Inhalt als Instanz des Datentyps xs:string.	Der Inhalt als Instanz des Datentyps xs:string.
Verarbeitungs-anweisung	Der Inhalt als Instanz des Datentyps xs:string.	Der Inhalt als Instanz des Datentyps xs:string.

Tools mit XML-Unterstützung

Sowohl IBM Tools als auch Tools von anderen Anbietern bieten Unterstützung für das Arbeiten mit der Komponente pureXML. Die folgenden Tools, die entweder mit einem DB2-Datenbankserver geliefert werden oder separat als Download erhältlich sind, sind über IBM verfügbar:

IBM Data Studio

Die XML-Unterstützung umfasst Folgendes:

- **Gespeicherte Prozeduren:** Sie können gespeicherte Prozeduren erstellen und ausführen, die XML-Datentypen als Ein- oder Ausgabeparameter enthalten.
- **Datenausgabe:** Sie können Dokumente, die in XML-Spalten enthalten sind, im Baumstruktur- oder Textformat anzeigen.
- **SQL-Editor:** Sie können SQL-Anweisungen und XQuery-Ausdrücke erstellen, die sowohl zusammen mit relationalen Daten als auch zusammen mit XML-Daten verwendet werden können.
- **XML-Schemata:** Sie können Schemadokumente im XML-Schema-Repository (XSR) verwalten, Schemata registrieren und löschen sowie Schemadokumente bearbeiten.
- **Gültigkeitsprüfung von XML-Dokumenten:** Sie können XML-Dokumente auf der Basis von Schemata prüfen, die im XML-Schema-Repository (XSR) registriert wurden.
- **Benutzerdefinierte SQL-Funktionen:** Sie können benutzerdefinierte SQL-Funktionen erstellen und ausführen, die XML-Parameter verwenden.

Befehlszeilenprozessor

Eine Reihe von DB2-Befehlen unterstützt die native Speicherung von XML-Daten. XML-Daten können über den DB2-Befehlszeilenprozessor (CLP = Command Line Processor) zusammen mit relationalen Daten verarbeitet werden. Über den Befehlszeilenprozessor können z. B. die folgenden Tasks ausgeführt werden:

- Absetzen von XQuery-Anweisungen durch Angabe des Schlüsselwortes XQUERY als Präfix.
- Importieren und Exportieren von XML-Daten.
- Erfassen von Statistikdaten für XML-Spalten.
- Aufrufen von gespeicherten Prozeduren mit dem Parameter IN, OUT oder INOUT des Datentyps XML.
- Arbeiten mit XML-Schemata, DTDs und externen Entitäten, die zum Verarbeiten von XML-Dokumenten erforderlich sind.

- Reorganisieren von Indizes zu XML-Daten und -Tabellen, die XML-Spalten enthalten.
- Zerlegen von XML-Dokumenten.

IBM Database Add-Ins für Microsoft Visual Studio

Sie können IBM Database Add-Ins for Microsoft Visual Studio zum Erstellen von Tabellen mit XML-Spalten und Indizes zu XML-Daten verwenden. Mit diesem Tool können Sie eine XML-Spalte in derselben Weise wie eine beliebige andere Spalte erstellen. Geben Sie hierbei einfach als Datentyp 'XML' an. Sie können in diesem Tool einen Index erstellen, indem Sie den XML Index Designer verwenden. Der XML-Musterausdruck muss nicht manuell angegeben werden, wie dies in der Syntax von CREATE INDEX bei einem Index zu XML-Daten der Fall ist. Stattdessen können Sie die XML-Knoten, die indexiert werden sollen, in einer Baumstrukturdarstellung eines registrierten XML-Schemas, eines Dokuments aus der XML-Spalte oder eines XML-Schemas in einer lokalen Datei auswählen. Das Tool generiert dann Ihren XML-Musterausdruck. Alternativ hierzu können Sie den XML-Musterausdruck auch manuell angeben. Nachdem Sie alle anderen Indexattribute angegeben haben, generiert das Tool den Index für Sie.

EXPLAIN

Sie können die Anweisung EXPLAIN für XQuery-Anweisungen und SQL/XML-Anweisungen absetzen, um den Zugriffsplan für diese Anweisungen aufzurufen und festzustellen, ob der DB2-Datenbankserver mit Indizes arbeitet. Um die Anweisung EXPLAIN für eine XQuery-Anweisung abzusetzen, müssen Sie das Schlüsselwort XQuery und anschließend eine XQuery-Anweisung verwenden, die in einfache oder doppelte Anführungszeichen eingeschlossen ist. Beispiel:

```
EXPLAIN PLAN SELECTION FOR XQUERY 'for $c in
db2-fn:xmlcolumn("XISCANTABLE.XMLCOL" )/a[@x="1"]/b[@y="2"] return $c'
```

DB2 erfasst die Daten des Zugriffsplans in den EXPLAIN-Tabellen. Die erwartete Sequenzgröße für XML-Spalten wird in der Spalte SEQUENCE_SIZES der Tabelle EXPLAIN_STREAM gespeichert. Darüber hinaus werden Sie in der Tabelle EXPLAIN_PREDICATE möglicherweise auch Daten für verschiedene Vergleichselemente feststellen, die Sie nicht identifizieren können. Diese Vergleichselemente werden vom DB2-Datenbankserver während der Ausführung von EXPLAIN generiert, um XPath-Ausdrücke auszuwerten, die bei einer Indexsuche verwendet werden. Diese Vergleichselementdaten müssen nicht ausgewertet werden. Die entsprechenden Vergleichselemente gehören nicht zum Optimierungsprogrammplan und weisen deshalb in den Spalten PREDICATE_ID und FILTER_FACTOR den Wert -1 auf.

Alternativ hierzu können Sie das manuelle Interpretieren der EXPLAIN-Tabellen auch umgehen, indem Sie IBM Data Studio zur Anzeige einer grafischen Darstellung dieser Zugriffspläne einsetzen. Einzelheiten hierzu finden Sie in Erstellen von Diagrammen für Zugriffspläne mit Visual Explain.

Die folgenden Knoten werden in den Diagrammen angezeigt, um die XML-Operationen darzustellen:

IXAND
Gibt an, dass der DB2-Datenbankserver das Vergleichselement AND auf die Ergebnisse mehrerer Indexsuchoperationen angewendet hat.

XISCAN
Gibt an, dass der DB2-Datenbankserver für den Datenzugriff einen Index für die XML-Daten verwendet hat.

XSCAN
Gibt an, das der DB2-Datenbankserver die XPath-Ausdrücke ausgewertet und XML-Dokumentfragmente aus den XML-Dokumenten extrahiert hat.

XANDOR
Gibt an, dass der DB2-Datenbankserver die Vergleichselemente AND und OR auf die Ergebnisse mehrerer Indexsuchoperationen angewendet hat.

XTQ
Gibt an, dass der DB2-Datenbankserver eine spezielle Tabellenwarteschlange (TQ) verwendet hat, um die einzelnen Elemente einer globalen XML-Sequenz an ihre ursprüngliche Datenbankpartition weiterzuleiten, XML-Daten aus XML-Dokumenten entsprechend der Auswertung der Elemente abzurufen und die XML-Daten zu einer Ausgabesequenz zusammenzufassen.

Föderationsunterstützung für pureXML

In einer föderierten Umgebung können Sie mit fernen Datenquellen arbeiten, die XML-Dokumente in XML-Spalten enthalten. Sie können ferne XML-Daten sowohl abfragen als auch bearbeiten. Dies umfasst auch die Dekomposition von XML-Dokumenten für ferne Tabellen.

Bevor Sie mit fernen XML-Daten arbeiten können, müssen Sie einen Kurznamen für die ferne Tabelle erstellen, die die XML-Spalte enthält, in der die gewünschten Dokumente gespeichert sind.

Weitere Informationen zum Konfigurieren eines föderierten Systems mit XML-Datenquellen finden Sie im Abschnitt zum Arbeiten mit fernen XML-Daten in der Dokumentation von WebSphere Federation Server.

Replikations- und Event-Publishing-Unterstützung für pureXML

Die Unterstützung von WebSphere® Replication Server und WebSphere Data Event Publisher für den XML-Datentyp ermöglicht Ihnen das Replizieren und Veröffentlichen von XML-Dokumenten, die in XML-Spalten gespeichert sind.

Sie können Q Replication verwenden, um XML-Dokumente zwischen Datenbanken zu replizieren, oder Sie können Event-Publishing verwenden, um Dokumente für Anwendungen zu veröffentlichen.

Weitere Informationen zum Konfigurieren von Q Replication und Event-Publishing für Datenbanken mit XML-Dokumenten in XML-Spalten finden Sie unter dem Thema "XML-Datentyp" sowie unter übergeordneten Themen in der Dokumentation von WebSphere Replication Server und WebSphere Data Event Publisher.

Artikel zur XML-Unterstützung

Über das developerWorks-Informationsmanagement stehen weitere Artikel zur Nutzung der XML-Unterstützung zur Verfügung. Diese Artikel behandeln eine breite Palette von Themen. Hierzu gehören neben der Migration und der Datenversetzung allgemeine Übersichten, Lernprogramme mit schrittweisen Anweisungen und Best Practices zum Arbeiten mit XML-Daten.

Die Artikel können unter der Adresse www.ibm.com/developerworks/data/zones/xml/ abgerufen werden.

Anmerkung: developerWorks gehört nicht zum DB2 Information Center. Dieser Link wird außerhalb des DB2 Information Center geöffnet.

Kapitel 2. Lernprogramm für pureXML

Mit dem XML-Datentyp pureXML können Sie Tabellenspalten so definieren, dass in jeder Zeile ein separates, korrekt formatiertes XML-Dokument gespeichert werden kann. Dieses Lernprogramm zeigt, wie Sie eine DB2-Datenbank zum Speichern von XML-Daten einrichten und grundlegende Operationen mit der Funktion pureXML ausführen.

Nach dem Durcharbeiten dieses Lernprogramms können Sie die folgenden Tasks ausführen:

- Erstellen einer DB2-Datenbank und einer Tabelle zum Speichern von XML-Daten
- Erstellen von Indizes zu XML-Daten
- Einfügen von XML-Dokumenten in Spalten mit dem Datentyp XML
- Aktualisieren von XML-Dokumenten, die in einer XML-Spalte gespeichert sind
- Löschen von Zeilen auf der Basis des Inhalts von XML-Dokumenten
- Abfragen von XML-Daten
- Prüfen von XML-Dokumenten anhand von XML-Schemata
- Umsetzung mit XSLT-Style-Sheets

Anwendungsprogrammierungssprachen wie C++, Java und PHP unterstützen den XML-Datentyp. Sie können Anwendungen zum Speichern von XML-Daten in DB2-Datenbanktabellen, zum Abrufen von Daten aus Tabellen oder zum Aufrufen gespeicherter Prozeduren bzw. benutzerdefinierter Funktionen mit XML-Parametern schreiben.

Dieses Lernprogramm wurde für eine Umgebung mit Einzelpartitionsdatenbank verfasst. Sie können die Funktion pureXML jedoch auch in einer Umgebung mit partitionierten Datenbanken verwenden.

Voraussetzungen

Starten Sie in einem DB2-Befehlsfenster den DB2-Befehlszeilenprozessor, indem Sie den Befehl `db2 -td~` absetzen, also den Befehl **db2** mit der Option **-td~**.[1]

Mit der Option **-td** wird die Tilde (~) als Anweisungsabschlusszeichen festgelegt. Durch die Angabe eines anderen Abschlusszeichens als dem standardmäßigen Semikolon (Option **-t**) lässt sich sicherstellen, dass Anweisungen oder Abfragen, die Namensbereichsdeklarationen verwenden, nicht fehlinterpretiert werden, da Namensbereichsdeklarationen auch durch ein Semikolon abgeschlossen werden. In den Beispielen dieses Lernprogramms wird durchgehend das Abschlusszeichen Tilde (~) verwendet.

Sie können die Beispiele in den Lerneinheiten im interaktiven Modus in den DB2-Befehlszeilenprozessor eingeben oder sie kopieren und einfügen.

Namensbereiche: Die im Lernprogramm verwendeten XML-Dokumente enthalten Namensbereiche. Bei der Verwendung von XML-Dokumenten, die Namensbereiche enthalten, müssen alle Abfragen und zugehörigen Operationen, die einen Namensbereich angeben, beispielsweise die Erstellung eines Index zu XML-Daten mit der

1. Starten Sie unter Windows mit dem Befehl **db2cmd** ein DB2-Befehlsfenster, bevor Sie den Befehl **db2 -td~** absetzen.

Anweisung CREATE INDEX oder die Abfrage von XML-Daten mit einem XQuery-Ausdruck, denselben Namensbereich deklarieren, um die erwarteten Ergebnisse liefern zu können. Das Deklarieren von Namensbereichen bei der Verwendung von XML-Dokumenten, die Namensbereiche enthalten, ist das Standardverhalten von Namensbereichen.

Lerneinheit 1: Erstellen einer DB2-Datenbank und einer Tabelle, in der XML-Daten gespeichert werden können

Diese Lerneinheit zeigt, wie eine Datenbank mit einer Tabelle erstellt wird, die eine XML-Spalte enthält.

Im Lernprogramm speichern Sie XML-Daten in einer Tabelle, die eine Tabelle mit einer Spalte des Typs XML enthält. Führen Sie folgende Schritte aus, um die im Lernprogramm verwendete Datenbank und Tabelle zu erstellen:

1. Erstellen Sie eine Datenbank namens XMLTUT, indem Sie den folgenden Befehl absetzen:

   ```
   CREATE DATABASE xmltut~
   ```

   Datenbanken verwenden standardmäßig den codierten Zeichensatz UTF-8 (Unicode). Wenn Sie XML-Daten in einer Datenbank mit einem anderen codierten Zeichensatz als UTF-8 speichern wollen, werden diese Daten am besten so eingefügt, dass keine Codepagekonvertierung stattfindet, wie beispielsweise bei BIT DATA, BLOB oder XML. Setzen Sie den Konfigurationsparameter **ENABLE_XMLCHAR** auf **NO**, um die Verwendung von Zeichendatentypen während des XML-Parsings zu blockieren und somit zu verhindern, dass möglicherweise eine Zeichensubstitution stattfindet.
2. Stellen Sie eine Verbindung zur Datenbank her:

   ```
   CONNECT TO xmltut~
   ```
3. Erstellen Sie eine Tabelle mit dem Namen 'Customer', die die XML-Spalte 'INFO' enthält:

   ```
   CREATE TABLE Customer (Cid BIGINT NOT NULL PRIMARY KEY, Info XML)~
   ```

   Es ist kein Primärschlüssel zum Speichern oder Indexieren von XML-Daten erforderlich.

   Mit der SQL-Anweisung ALTER TABLE können Sie auch eine oder mehrere XML-Spalten zu einer Tabelle hinzufügen.

Zum Lernprogramm zurückkehren

Lerneinheit 2: Erstellen von Indizes für XML-Daten

In dieser Lerneinheit wird gezeigt, wie Sie einen Index für XML-Daten erstellen. Indizes für XML-Daten können die Leistung von Abfragen auf XML-Spalten verbessern. Indexieren Sie XML-Elemente oder XML-Attribute, die Sie häufig in Vergleichselementen und dokumentübergreifenden Joins verwenden.

Ein relationaler Index führt ebenso wie ein Index für XML-Daten eine Indexierung für Spalten durch. Während jedoch ein relationaler Index die gesamte Spalte indexiert, berücksichtigt ein Index für XML-Daten bei der Indexierung nur einen Teil der Spalte. Sie können angeben, welche Teile einer XML-Spalte indexiert werden sollen, indem Sie ein XML-Muster angeben, bei dem es sich um einen eingeschränkten XPath-Ausdruck handelt. Sie müssen außerdem den Datentyp angeben, mit dem die indexierten Werte gespeichert werden. Im Allgemeinen sollte der von Ihnen ausgewählte Typ derselbe Typ sein, der auch in Abfragen verwendet wird.

Wie bei relationalen Indizes ist zu empfehlen, solche XML-Elemente bzw. XML-Attribute zu indexieren, die häufig in Vergleichselementen und dokumentübergreifenden Joins verwendet werden.

Sie können nur einzelne XML-Spalten indexieren. Zusammengesetzte Indizes werden nicht unterstützt. Sie können jedoch mehrere Indizes für eine XML-Spalte haben.

Nicht alle Klauseln der Anweisung CREATE INDEX lassen sich auf Indizes für XML-Daten anwenden. Detaillierte Informationen finden Sie in der Dokumentation zur Anweisung CREATE INDEX.

Setzen Sie die folgende Anweisung ab, um einen Index für XML-Daten zu erstellen:

```
CREATE INDEX cust_cid_xmlidx ON Customer(Info)
   GENERATE KEY USING XMLPATTERN
  'declare default element namespace "http://posample.org"; /customerinfo/@Cid'
AS SQL DOUBLE~
```

Die Anweisung indexiert die Werte des Attributs 'Cid' des Elements <customerinfo> der Spalte 'INFO' der Tabelle 'CUSTOMER'. Wenn sich indexierte XML-Daten nicht in den angegebenen Datentyp SQL DOUBLE umsetzen lassen, wird standardmäßig kein Indexeintrag erstellt und kein Fehler zurückgegeben.

Bei der Angabe des XML-Musters muss die Groß-/Kleinschreibung beachtet werden. Wenn die XML-Dokumente zum Beispiel nicht im Attribut "Cid", sondern im Attribut "cid" enthalten sind, entsprechen diese Dokumente nicht dem zu erstellenden Index.

Zum Lernprogramm zurückkehren

Lerneinheit 3: Einfügen von XML-Dokumenten in Spalten mit einem XML-Datentyp

Korrekt formatierte XML-Dokumente werden in Spalten mit einem XML-Datentyp mithilfe der SQL-Anweisung INSERT eingefügt. In dieser Lerneinheit wird gezeigt, wie Sie korrekt formatierte XML-Dokumente mithilfe der SQL-Anweisung INSERT in XML-Spalten einfügen.

In dieser Lerneinheit wird gezeigt, wie Sie XML-Dokumente mithilfe des Befehlszeilenprozessors manuell in XML-Spalten einfügen. In der Regel werden XML-Dokumente jedoch über Anwendungsprogramme eingefügt.

Sie können XML-Daten zwar mithilfe von XML-, Binär- oder Zeichendatentypen einfügen, sollten dazu jedoch XML- oder Binärdatentypen verwenden, um Probleme durch Codepagekonvertierungen zu vermeiden. Die XML-Dokumente in dieser Lerneinheit sind Zeichenliterale. In den meisten Fällen können Sie Zeichenfolgedaten nicht direkt einem Ziel mit einem XML-Datentyp zuordnen, sondern müssen sie zunächst mithilfe der Funktion XMLPARSE explizit syntaktisch analysieren. In INSERT-, UPDATE- oder DELETE-Operationen können Zeichenfolgedaten jedoch direkt XML-Spalten zugeordnet werden, ohne die Funktion XMLPARSE explizit aufzurufen. In diesen drei Fällen erfolgt eine implizite Syntaxanalyse der Zeichenfolgedaten. Weitere Informationen zu diesem Thema finden Sie in der Dokumentation zur Syntaxanalyse (Parsing) von XML-Daten.

Setzen Sie die folgenden Anweisungen ab, um drei XML-Dokumente in die Tabelle 'Customer' einzufügen, die Sie in Lerneinheit 1 erstellt haben:

```
INSERT INTO Customer (Cid, Info) VALUES (1000,
'<customerinfo xmlns="http://posample.org" Cid="1000">
    <name>Kathy Smith</name>
  <addr country="Canada">
        <street>5 Rosewood</street>
        <city>Toronto</city>
    <prov-state>Ontario</prov-state>
        <pcode-zip>M6W 1E6</pcode-zip>
  </addr>
    <phone type="work">416-555-1358</phone>
</customerinfo>')~
INSERT INTO Customer (Cid, Info) VALUES (1002,
'<customerinfo xmlns="http://posample.org" Cid="1002">
  <name>Jim Noodle</name>
  <addr country="Canada">
    <street>25 EastCreek</street>
        <city>Markham</city>
    <prov-state>Ontario</prov-state>
        <pcode-zip>N9C 3T6</pcode-zip>
  </addr>
    <phone type="work">905-555-7258</phone>
</customerinfo>')~
INSERT INTO Customer (Cid, Info) VALUES (1003,
'<customerinfo xmlns="http://posample.org" Cid="1003">
    <name>Robert Shoemaker</name>
  <addr country="Canada">
        <street>1596 Baseline</street>
        <city>Aurora</city>
    <prov-state>Ontario</prov-state>
        <pcode-zip>N8X 7F8</pcode-zip>
  </addr>
    <phone type="work">905-555-2937</phone>
</customerinfo>')~
```

Mit der folgenden Anweisung können Sie sich vergewissern, ob die Datensätze erfolgreich eingefügt wurden:

```
SELECT * from Customer~
```

Zum Lernprogramm zurückkehren

Lerneinheit 4: Aktualisieren von XML-Dokumenten, die in XML-Spalten gespeichert sind

Diese Lerneinheit zeigt, wie XML-Dokumente mithilfe einer SQL-Anweisung UPDATE mit oder ohne XQuery-Aktualisierungsausdruck aktualisiert werden.

Aktualisierung ohne XQuery-Aktualisierungsausdruck

Wenn Sie die Anweisung UPDATE ohne XQuery-Aktualisierungsausdruck verwenden, müssen Sie eine Aktualisierung des gesamten Dokuments durchführen.

Führen Sie zum Aktualisieren der Werte der Elemente <street>, <city> und <pcode-zip> für eines der Dokumente, die Sie in Lerneinheit 3 eingefügt haben, die folgende Anweisung aus:

```
UPDATE customer SET info =
'<customerinfo xmlns="http://posample.org" Cid="1002">
  <name>Jim Noodle</name>
  <addr country="Canada">
        <street>1150 Maple Drive</street>
```

```
        <city>Newtown</city>
    <prov-state>Ontario</prov-state>
        <pcode-zip>Z9Z 2P2</pcode-zip>
  </addr>
    <phone type="work">905-555-7258</phone>
</customerinfo>'
WHERE XMLEXISTS (
   'declare default element namespace "http://posample.org";
  $doc/customerinfo[@Cid = 1002]'
  passing INFO as "doc")~
```

Das XMLEXISTS-Vergleichselement stellt sicher, dass nur das Dokument mit dem Attribut Cid="1002" ersetzt wird. Beachten Sie, dass der Vergleichsausdruck [@Cid = 1002] in XMLEXISTS nicht als Zeichenfolgevergleich [@Cid = "1002"] angegeben wird. Das liegt daran, dass Sie in Übung 2 beim Erstellen des Index für das Attribut "Cid" den Datentyp DOUBLE verwendet haben. Wenn der Index dieser Abfrage entsprechen soll, darf "Cid" nicht als Zeichenfolge im Vergleichsausdruck angegeben werden.

Mit der folgenden Anweisung können Sie sich vergewissern, ob das XML-Dokument aktualisiert wurde:

```
SELECT * from Customer~
```

Der Datensatz mit 'Cid="1002"' enthält jetzt die geänderten Werte für <street>, <city> und <pcode-zip>.

Wenn Sie XML-Dokumente mithilfe von Werten in den Nicht-XML-Spalten der gleichen Tabelle identifizieren können, können Sie SQL-Vergleichselemente zur Ermittlung der zu aktualisierenden Zeilen verwenden. Im vorigen Beispiel, in dem der Wert 'Cid' aus dem XML-Dokument auch in der Spalte 'CID' der Tabelle 'CUSTOMER' gespeichert wird, hätten Sie auch ein SQL-Vergleichselement für die Spalte 'CID' zur Ermittlung der Zeile verwenden könnten. Sie können die Klausel WHERE im vorigen Beispiel durch die folgende Klausel ersetzen:

```
WHERE Cid=1002
```

Aktualisierung mit XQuery-Aktualisierungsausdruck

Wenn Sie die Anweisung UPDATE mit einem XQuery-Aktualisierungsausdruck verwenden, können Sie Teile eines vorhandenen XML-Dokuments aktualisieren.

Führen Sie zum Aktualisieren der Kundenadresse in einem vorhandenen XML-Dokument die folgende SQL-Anweisung aus, die einen XQuery-Umsetzungsausdruck verwendet:

```
UPDATE Customer set Info =
  XMLQUERY( 'declare default element namespace "http://posample.org";
  transform
  copy $mycust := $cust
  modify
    do replace $mycust/customerinfo/addr with
      <addr country="Canada">
        <street>25 EastCreek</street>
            <city>Markham</city>
        <prov-state>Ontario</prov-state>
            <pcode-zip>N9C 3T6</pcode-zip>
     </addr>
  return $mycust'
  passing INFO as "cust")
WHERE CID = 1002~
```

Um die Kundenadresse zu aktualisieren, führt die Funktion XMLQUERY einen XQuery-Umsetzungsausdruck aus, der einen Ersetzungsausdruck verwendet und anschließend die aktualisierten Informationen wie folgt an die Anweisung UPDATE zurückgibt:

- Die XMLQUERY-Übergabeklausel verwendet die Kennung `cust`, um die Kundeninformationen aus der XML-Spalte INFO an den XQuery-Ausdruck zu übergeben.
- In der Klausel **`COPY`** des Umsetzungsausdrucks wird eine logische Momentaufnahme der Kundeninformationen erstellt und der Variablen `$mycust` zugeordnet.
- In der Klausel **`MODIFY`** des Umsetzungsausdrucks ersetzt der Ersetzungsausdruck die Adressinformationen in der Kopie der Kundeninformationen.
- XMLQUERY gibt das aktualisierte Kundendokument in der Variablen `$mycust` zurück.

Mit der folgenden Anweisung können Sie sich vergewissern, ob das XML-Dokument die aktualisierte Kundenadresse enthält:

```
SELECT Info FROM Customer WHERE Cid = 1002~
```

Zum Lernprogramm zurückkehren

Lerneinheit 5: Löschen von XML-Daten

Diese Lerneinheit zeigt, wie ganze XML-Dokumente oder Abschnitte aus XML-Dokumenten mithilfe von SQL-Anweisungen gelöscht werden.

Löschen ganzer XML-Dokumente

Verwenden Sie zum Löschen von ganzen XML-Dokumenten die SQL-Anweisung DELETE. Zur Angabe von bestimmten zu löschenden Dokumenten verwenden Sie das XMLEXISTS-Vergleichselement.

Wenn Sie nur XML-Dokumente löschen wollen, die ein Element <customerinfo> mit einem Attribut "Cid=1003" aus der Spalte "Info" enthalten, setzen Sie die folgende Anweisung ab:

```
DELETE FROM Customer
WHERE XMLEXISTS (
   'declare default element namespace "http://posample.org";
  $doc/customerinfo[@Cid = 1003]'
  passing INFO as "doc")~
```

Wenn Sie XML-Dokumente mithilfe von Werten in den Nicht-XML-Spalten der gleichen Tabelle identifizieren können, können Sie SQL-Vergleichselemente zur Ermittlung der zu löschenden Zeilen verwenden. Im vorigen Beispiel, in dem der Wert 'Cid' aus dem XML-Dokument auch in der Spalte 'CID' der Tabelle 'CUSTOMER' gespeichert wird, könnten Sie die gleiche Operation mit der folgenden DELETE-Anweisung ausführen, in der ein SQL-Vergleichselement auf die Spalte 'CID' zur Ermittlung der Zeile angewendet wird:

```
DELETE FROM Customer WHERE Cid=1003~
```

Mit der folgenden SQL-Anweisung können Sie sich vergewissern, ob die XML-Dokumente gelöscht wurden:

```
SELECT * FROM Customer~
```

Es werden zwei Datensätze zurückgegeben.

Löschen von Abschnitten aus XML-Dokumenten

Wenn Sie nur bestimmte Abschnitte aus einem XML-Dokument und nicht das gesamte Dokument löschen wollen, verwenden Sie eine SQL-Anweisung UPDATE, die einen XQuery-Aktualisierungsausdruck enthält.

Führen Sie zum Löschen aller Telefonnummern aus einem Kundeneintrag mit dem Cid-Wert 1002 die nachstehende SQL-Anweisung aus, die die Funktion XMLQUERY verwendet:

```
UPDATE Customer
SET info = XMLQUERY(
     'declare default element namespace "http://posample.org";
    transform
    copy $newinfo := $info
    modify do delete ($newinfo/customerinfo/phone)
    return $newinfo' passing info as "info")
WHERE cid = 1002~
```

Um das Element <phone> zu entfernen, führt die Funktion XMLQUERY einen XQuery-Umsetzungsausdruck aus, der einen Löschungsausdruck verwendet und anschließend die aktualisierten Informationen wie folgt an die Anweisung UPDATE zurückgibt:

- Die XMLQUERY-Übergabeklausel verwendet die Kennung `info`, um die Kundeninformationen aus der XML-Spalte INFO an den XQuery-Ausdruck zu übergeben.
- In der Klausel **`COPY`** des Umsetzungsausdrucks wird eine logische Momentaufnahme der Kundeninformationen erstellt und der Variablen `$newinfo` zugeordnet.
- In der Klausel **`MODIFY`** des Umsetzungsausdrucks löscht der Löschungsausdruck das Element <phone> in der Kopie der Kundeninformationen.
- XMLQUERY gibt das aktualisierte Kundendokument in der Variablen `$newinfo` zurück.

Mithilfe der folgenden Anweisung können Sie prüfen, ob der Kundeneintrag das Element <phone> nicht mehr enthält:

```
SELECT * FROM Customer WHERE Cid=1002~
```

Zum Lernprogramm zurückkehren

Lerneinheit 6: Abfragen von XML-Daten

In dieser Lerneinheit wird gezeigt, wie XML-Dokumente mithilfe von SQL, XQuery (mit XQuery-Ausdrücken) oder mit einer Kombination dieser beiden Sprachen abgefragt werden.

Wenn Sie nur SQL verwenden, können Sie lediglich Abfragen auf Spaltenebene durchführen. Das heißt, Sie können das gesamte in einer Spalte gespeicherte XML-Dokument abrufen, jedoch keine Werte innerhalb des Dokuments abfragen oder Fragmente des Dokuments abrufen. Zur Abfrage von Werten innerhalb eines XML-Dokuments oder zum Abrufen von Fragmenten eines Dokuments müssen Sie XQuery verwenden.

Bei den Abfragen in dieser Lerneinheit wird XQuery im SQL-Kontext und SQL im XQuery-Kontext verwendet.

Wichtig: Bei XQuery ist die Groß-/Kleinschreibung zu beachten, während dies bei SQL nicht erforderlich ist. Gehen Sie daher bei der Verwendung von XQuery besonders sorgfältig vor, wenn Sie Namen wie Tabellen- oder SQL-Schemanamen angeben, die standardmäßig in Großbuchstaben vorliegen. Selbst in einem SQL-Kontext ist in XQuery-Ausdrücken die Groß-/Kleinschreibung zu beachten.

Abfragen im SQL-Kontext

Abrufen gesamter XML-Dokumente

Führen Sie zum Abrufen aller XML-Dokumente in der Spalte 'INFO' und aller Werte aus der Primärschlüsselspalte 'CID' die folgende SELECT-Anweisung aus:

```
SELECT cid, info FROM customer~
```

Diese Abfrage gibt die beiden gespeicherten XML-Dokumente zurück.

Abrufen und Filtern von XML-Werten

Führen Sie für Abfragen innerhalb der XML-Dokumente in der Spalte 'INFO' die folgende Anweisung SELECT aus, die mithilfe der Funktion XMLQUERY einen XQuery-Ausdruck aufruft:

```
SELECT XMLQUERY (
  'declare default element namespace "http://posample.org";
      for $d in $doc/customerinfo
   return <out>{$d/name}</out>'
      passing INFO as "doc")
FROM Customer as c
       WHERE XMLEXISTS ('declare default element namespace "http://posample.org";
   $i/customerinfo/addr[city="Toronto"]' passing c.INFO as "i")~
```

In der Funktion XMLQUERY wird zunächst ein Standardnamensbereich ('namespace') angegeben. Dieser Namensbereich stimmt mit dem Namensbereich der zuvor eingefügten Dokumente überein. Die Klausel **for** gibt eine Iteration durch die Elemente <customerinfo> aller Dokumente in der Spalte 'Info' an. Die Spalte 'INFO' wird mithilfe der Klausel **passing** angegeben, die die Spalte 'INFO' an die Variable `doc` bindet, die in der Klausel **for** angegeben ist. Die Klausel **return** konstruiert ein Element <out>, in dem aus jeder Iteration der Klausel **for** ein Element <name> enthalten ist.

In der Klausel WHERE wird das Vergleichselement XMLEXISTS verwendet, um die Abfrage auf eine Teilmenge der Dokumente in der Spalte 'Info' zu begrenzen. Diese Filterung liefert nur diejenigen Dokumente, in denen das Element <city> (im angegebenen Pfad) den Wert "Toronto" besitzt.

Die SELECT-Anweisung gibt das folgende konstruierte Element zurück:

```
<out xmlns="http://posample.org"><name>Kathy Smith</name></out>
```

Verwendung von db2-fn:sqlquery mit Parametern

Führen Sie die folgende Abfrage aus, um einen Wert an eine SQL-Fullselect-Operation in der Funktion db2-fn:sqlquery zu übergeben:

```
VALUES XMLQUERY (
  'declare default element namespace "http://posample.org";
  for $d in db2-fn:sqlquery(
    ''SELECT INFO FROM CUSTOMER WHERE Cid = parameter(1)'',
    $testval)/customerinfo
  return <out>{$d/name}</out>'
  passing 1000 as "testval" )~
```

Die Funktion XMLQUERY übergibt den Wert "1000" unter Verwendung der Kennung `testval` an den XQuery-Ausdruck. Anschließend übergibt der

XQuery-Ausdruck unter Verwendung der Skalarfunktion PARAMETER den Wert an die Funktion db2-fn:sqlquery.

Der XQuery-Ausdruck gibt das folgende erstellte Element zurück:

```
<out xmlns="http://posample.org">
     <name>Kathy Smith</name>
</out>
```

Abfragen im XQuery-Kontext

DB2 XQuery bietet zwei integrierte Funktionen speziell zur Verwendung mit DB2-Datenbanken an: db2-fn:sqlquery und db2-fn:xmlcolumn. Die Funktion db2-fn:sqlquery ruft eine Sequenz ab, welche die Ergebnistabelle einer SQL-Fullselect-Anweisung ist. Die Funktion db2-fn:xmlcolumn ruft eine Sequenz aus einer XML-Spalte ab.

Wenn bei Ihrer Abfrage ein XQuery-Ausdruck direkt aufgerufen wird, müssen Sie ihm das Schlüsselwort "XQUERY" voranstellen (wobei die Groß-/Kleinschreibung nicht beachtet werden muss).

Anmerkung: Es stehen verschiedene Optionen zur Verfügung, die Sie zur Optimierung der Umgebung Ihres Befehlszeilenprozessors und speziell für die Anzeige der Ergebnisse eines XQuery-Ausdrucks angeben können. Geben Sie beispielsweise die Option **-i** an, um die Ergebnisse von XQuery-Ausdrücken lesbarer zu machen:

```
UPDATE COMMAND OPTIONS USING i ON~
```

Abrufen gesamter XML-Dokumente

Zum Abrufen aller XML-Dokumente, die zuvor in die Spalte 'INFO' eingefügt wurden, können Sie XQuery mit db2-fn:xmlcolumn oder db2-fn:sqlquery verwenden.

Verwendung von db2-fn:xmlcolumn

Zum Abrufen aller XML-Dokumente in der Spalte 'INFO' führen Sie die folgende Abfrage aus:

```
XQUERY db2-fn:xmlcolumn ('CUSTOMER.INFO')~
```

Namen in SQL-Anweisungen werden standardmäßig automatisch in Großbuchstaben umgesetzt. Wenn die Tabelle 'CUSTOMER' mit der SQL-Anweisung CREATE TABLE erstellt wurde, wurden daher die Namen der Tabelle und der Spalten in Großbuchstaben umgewandelt. Da bei XQuery die Groß-/Kleinschreibung unterschieden wird, müssen Sie sorgfältig darauf achten, die Tabellen- und Spaltennamen bei Verwendung der Funktion db2-fn:xmlcolumn in der korrekten Groß-/Kleinschreibung anzugeben.

Diese Abfrage entspricht der SQL-Abfrage `SELECT Info FROM Customer`.

Verwendung von db2-fn:sqlquery

Zum Abrufen aller XML-Dokumente in der Spalte 'INFO' führen Sie die folgende Abfrage aus:

```
XQUERY db2-fn:sqlquery ('SELECT Info FROM Customer')~
```

Sie müssen die Namen 'INFO' und 'CUSTOMER' nicht in Großbuchstaben angeben, weil die SELECT-Anweisung in einem SQL-Kontext verarbeitet wird, in dem die Groß-/Kleinschreibung nicht unterschieden wird.

Abrufen partieller XML-Dokumente

Anstatt ein vollständiges XML-Dokument abzurufen, können Sie auch Fragmente eines Dokuments abrufen und nach Werten filtern, die in dem Dokument enthalten sind, indem Sie XQuery mit db2-fn:xmlcolumn oder db2-fn:sqlquery verwenden.

Verwendung von db2-fn:xmlcolumn

Führen Sie die folgende Abfrage aus, um die Elemente, die Knoten <name> enthalten, für alle Dokumente in der Spalte 'Info' zurückzugeben, deren Elemente <city> (im angegebenen Pfad) den Wert "Toronto" haben.

```
XQUERY declare default element namespace "http://posample.org";
  for $d in db2-fn:xmlcolumn('CUSTOMER.INFO')/customerinfo
  where $d/addr/city="Toronto"
  return <out>{$d/name}</out>~
```

Die Funktion db2-fn:xmlcolumn ruft eine Sequenz aus der Spalte 'INFO' der Tabelle 'CUSTOMER' ab. Die Klausel **for** bindet die Variable $d an jedes Element <customerinfo> in der Spalte CUSTOMER.INFO. Die Klausel **where** grenzt die Datenelemente auf diejenigen ein, deren Element <city> (im angegebenen Pfad) den Wert "Toronto" hat. Die Klausel **return** konstruiert den zurückzugebenden XML-Wert. Dieser Wert ist ein Element <out>, in dem das Element <name> für alle Dokumente enthalten ist, welche die Bedingung der Klausel **where** erfüllen:

```
<out xmlns="http://posample.org">
<name>
                              Kathy Smith
</name>
</out>
```

Verwendung von db2-fn:sqlquery

Führen Sie die folgende Abfrage aus, um in einem XQuery-Ausdruck eine Fullselect-Anweisung abzusetzen:

```
XQUERY declare default element namespace "http://posample.org";
  for $d in db2-fn:sqlquery(
        'SELECT INFO
     FROM CUSTOMER
          WHERE Cid < 2000')/customerinfo
  where $d/addr/city="Toronto"
  return <out>{$d/name}</out>~
```

In diesem Beispiel wird die Gruppe von XML-Dokumenten, die abgefragt wird, zunächst in der Fullselect-Anweisung durch bestimmte Werte in der Nicht-XML-Spalte 'CID' begrenzt. Dieses Beispiel veranschaulicht den Vorteil der Funktion db2-fn:sqlquery: Sie ermöglicht die Anwendung von SQL-Vergleichselementen innerhalb eines XQuery-Ausdrucks. Die aus der SQL-Abfrage resultierenden Dokumente werden anschließend in der Klausel **where** des XQuery-Ausdrucks weiter auf diejenigen Dokumente eingegrenzt, die ein Element <city> (im angegebenen Pfad) mit dem Wert "Toronto" haben.

Die Abfrage liefert dasselbe Ergebnis wie die Abfrage im vorigen Beispiel, in dem die Funktion db2-fn:xmlcolumn verwendet wird:

```
<out xmlns="http://posample.org">
<name>
                              Kathy Smith
</name>
</out>
```

Verwendung von db2-fn:sqlquery mit Parametern

Führen Sie die folgende Abfrage aus, um einen Wert an eine SQL-Fullselect-Operation in der Funktion db2-fn:sqlquery zu übergeben:

```
XQUERY declare default element namespace "http://posample.org";
  let $testval := 1000
  for $d in db2-fn:sqlquery(
    'SELECT INFO FROM CUSTOMER WHERE Cid = parameter(1)',
    $testval)/customerinfo
  return <out>{$d/name}</out>~
```

Im XQuery-Ausdruck stellt die Klausel **LET** den Wert für `$testval` auf "1000" ein. Anschließend übergibt der Ausdruck in der Klausel **for** unter Verwendung der Skalarfunktion PARAMETER den Wert an die Funktion db2-fn:sqlquery.

Der XQuery-Ausdruck gibt das folgende erstellte Element zurück:

```
<out xmlns="http://posample.org">
     <name>Kathy Smith</name>
</out>
```

Zum Lernprogramm zurückkehren

Lerneinheit 7: Prüfen von XML-Dokumenten anhand von XML-Schemata

In dieser Lerneinheit wird gezeigt, wie Sie XML-Dokument auswerten. Sie können Ihre XML-Dokumente nur an XML-Schemata prüfen. Eine Prüfung an DTDs wird nicht unterstützt. Obwohl keine Prüfung an DTDs möglich ist, können Sie dennoch Dokumente einfügen, die einen DOCTYPE enthalten oder sich auf DTDs beziehen.

Sie können Tools wie die im Lieferumfang von IBM Rational Application Developer enthaltenen verwenden, die Ihnen die Generierung von XML-Schemata aus verschiedenen Quellen wie DTDs, Tabellen und XML-Dokumenten erleichtern.

Bevor Sie eine Prüfung durchführen können, müssen Sie Ihr XML-Schema im integrierten XML-Schema-Repository (XSR) registrieren. Dieser Prozess umfasst das Registrieren aller XML-Schemadokumente, die das XML-Schema bilden, und den Abschluss der Registrierung. Eine Methode zur Registrierung eines XML-Schemas besteht in der Ausführung von Befehlen.

Führen Sie den folgenden Befehl aus, um das Schemadokument zu registrieren und die Registrierung des XML-Schemas 'posample.customer' abzuschließen. (Da dieses XML-Schema lediglich aus einem einzigen Schemadokument besteht, können Sie mit einem einzigen Befehl das Dokument registrieren und die Registrierung abschließen.) Der Befehl gibt den absoluten Pfad zum Verzeichnis `sqllib/samples/xml` an. Beginnt der Pfad auf Ihrem System nicht mit `c:/sqllib/`, müssen Sie den Dateipfad in dem Befehl entsprechend ändern.

```
REGISTER XMLSCHEMA 'http://posample.org'
FROM 'file:///c:/sqllib/samples/xml/customer.xsd' AS posample.customer COMPLETE~
```

Sie können überprüfen, ob das XML-Schema erfolgreich registriert wurde. Fragen Sie hierzu die Katalogsicht SYSCAT.XSROBJECTS ab, die Informationen zu den im XML-Schema-Repository (XSR) gespeicherten Objekten enthält. Diese Abfrage und die zugehörigen Ergebnisse, die zur besseren Lesbarkeit formatiert wurden, lauten wie folgt:

```
SELECT OBJECTSCHEMA, OBJECTNAME FROM SYSCAT.XSROBJECTS~

OBJECTSCHEMA           OBJECTNAME
--------------------   --------------------
POSAMPLE               CUSTOMER
```

Nun können Sie das XML-Schema für die Prüfung verwenden. In der Regel wird die Prüfung während einer INSERT- oder UPDATE-Operation mit der Funktion XMLVALIDATE ausgeführt. Die INSERT- oder UPDATE-Operation, für die Sie die Funktion XMLVALIDATE angeben, wird nur dann ausgeführt, wenn die Prüfung erfolgreich ist.

Setzen Sie die folgende Anweisung ab, um ein XML-Dokument in die Spalte 'INFO' der Tabelle 'CUSTOMER' einzufügen, wenn das Dokument gemäß dem XML-Schema 'posample.customer' gültig ist:

```
INSERT INTO Customer(Cid, Info) VALUES (1003, XMLVALIDATE (XMLPARSE (DOCUMENT
'<customerinfo xmlns="http://posample.org" Cid="1003">
    <name>Robert Shoemaker</name>
  <addr country="Canada">
        <street>1596 Baseline</street>
        <city>Aurora</city>
    <prov-state>Ontario</prov-state>
        <pcode-zip>N8X 7F8</pcode-zip>
  </addr>
    <phone type="work">905-555-7258</phone>
    <phone type="home">416-555-2937</phone>
    <phone type="cell">905-555-8743</phone>
    <phone type="cottage">613-555-3278</phone>
</customerinfo>' PRESERVE WHITESPACE )
ACCORDING TO XMLSCHEMA ID posample.customer ))~
```

Das XML-Dokument im vorliegenden Beispiel wird als Zeichendaten übergeben. Die Funktion XMLVALIDATE operiert jedoch nur auf XML-Daten. Da das XML-Dokument als Zeichendaten übergeben wird, müssen Sie die Daten explizit mit der Funktion XMLPARSE syntaktisch analysieren. Die Funktion XMLPARSE analysiert das angegebene Argument als XML-Dokument und gibt einen XML-Wert zurück.

Der DB2-Datenbankserver führt bei bestimmten Operationen implizites Parsing aus, beispielsweise wenn Sie in einer Anweisung INSERT, UPDATE, DELETE oder MERGE eine Hostvariable, Parametermarke oder einen SQL-Ausdruck mit einem Zeichenfolgedatentyp (Zeichen, Grafik oder Binär) einer XML-Spalte zuordnen.

Wenn Sie sich vergewissern möchten, ob die Prüfung und die Einfügung erfolgreich ausgeführt wurden, fragen Sie die Spalte 'INFO' ab:

```
SELECT Info FROM Customer~
```

Diese Abfrage sollte drei XML-Dokumente zurückgeben, von denen eines das soeben eingefügte Dokument ist.

Zum Lernprogramm zurückkehren

Lerneinheit 8: Umsetzung mit XSLT-Formatvorlagen

In dieser Lerneinheit wird gezeigt, wie Sie XML-Daten in einer Datenbank mithilfe einer XSLT-Formatvorlage (XSLT - Extensible Stylesheet Language Transformation) und der integrierten Funktion XSLTRANSFORM in andere Formate konvertieren.

Beispiel: Ein XML-Dokument enthält eine beliebige Anzahl an Einträgen zu Studenten. Jedes Element vom Typ 'student' enthält eine Studenten-ID, einen Vornamen, einen Nachnamen, das Alter des Studenten sowie den Namen der von ihm besuchten Universität. Das folgende Dokument enthält zwei Studenten:

```
<?xml version="1.0"?>
<students xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
     <student studentID="1" givenName="Steffen" familyName="Siegmund"
        age="21" university="Rostock"/>
     <student studentID="2" givenName="Helena" familyName="Schmidt"
   age="23" university="Rostock"/>
</students>
```

Sie wollen nun die Informationen in den XML-Einträgen extrahieren und eine HTML-Webseite erstellen, die in einem Browser angezeigt werden kann. Zum Umsetzen der Informationen benötigen Sie die folgende XSLT-Formatvorlage:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:param name="headline"/>
<xsl:param name="showUniversity"/>
<xsl:template match="students">
     <html>
     <head/>
     <body>
       <h1><xsl:value-of select="$headline"/></h1>
     <table border="1">
     <th>
     <tr>
     <td width="80">StudentID</td>
     <td width="200">Given Name</td>
     <td width="200">Family Name</td>
     <td width="50">Age</td>
     <xsl:choose>
          <xsl:when test="$showUniversity =''true''">
               <td width="200">University</td>
          </xsl:when>
     </xsl:choose>
     </tr>
     </th>
     <xsl:apply-templates/>
     </table>
     </body>
     </html>
     </xsl:template>
            <xsl:template match="student">
               <tr>
               <td><xsl:value-of select="@studentID"/></td>
               <td><xsl:value-of select="@givenName"/></td>
               <td><xsl:value-of select="@familyName"/></td>
               <td><xsl:value-of select="@age"/></td>
               <xsl:choose>
                    <xsl:when test="$showUniversity = ''true'' ">
                         <td><xsl:value-of select="@university"/></td>
                    </xsl:when>
               </xsl:choose>
               </tr>
     </xsl:template>
</xsl:stylesheet>
```

Gehen Sie wie folgt vor, um die Daten umzusetzen:

1. Erstellen Sie mithilfe des folgenden Befehls zwei Tabellen, in denen Sie das XML-Dokument und das Formatvorlagendokument speichern:

```
CREATE TABLE XML_DATA (DOCID INTEGER, XML_DOC XML )~
CREATE TABLE XML_TRANS (XSLID INTEGER, XSLT_DOC CLOB(1M))~
```

2. Fügen Sie das XML-Dokument und die gesamte XSLT-Formatvorlage mithilfe der folgenden INSERT-Anweisungen in die Tabellen ein.

   Zur besseren Übersicht wird in diesem Schritt eine abgeschnittene Version der XSLT-Formatvorlage in der zweiten Anweisung INSERT angezeigt. Ersetzen Sie vor der Verwendung der Anweisung INSERT die abgeschnittene Formatvorlage durch die oben angeführte XSLT-Formatvorlage.

```
INSERT INTO XML_DATA VALUES
     (1,
      '<?xml version="1.0"?>
       <students xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
          <student studentID="1" givenName="Steffen" familyName="Siegmund"
        age="21" university="Rostock"/>
          <student studentID="2" givenName="Helena" familyName="Schmidt"
   age="23" university="Rostock"/>
      </students>'
 )~

INSERT INTO XML_TRANS VALUES
     (1,
     '<?xml version="1.0" encoding="UTF-8"?>
     <xsl:stylesheet version="1.0"
          ...
     </xsl:stylesheet>'
)~
```

3. Setzen Sie das XML-Dokument um, indem Sie die Funktion XSLTRANSFORM aufrufen:

```
SELECT XSLTRANSFORM (XML_DOC USING XSLT_DOC AS CLOB(1M))
   FROM XML_DATA, XML_TRANS WHERE DOCID = 1 and XSLID = 1 ~
```

Die Ausgabe der Umsetzung ist die folgende HTML-Datei:

```
<html>
<head>
<META http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<h1></h1>
<table border="1">
<th>
<tr>
<td width="80">StudentID</td>
<td width="200">Given Name</td>
<td width="200">Family Name</td>
<td width="50">Age</td>
</tr>
</th>
<tr>
<td>1</td>
<td>Steffen</td><td>Siegmund</td>
<td>21</td>
</tr>
<tr>
<td>2</td><td>Helena</td><td>Schmidt</td>
<td>23</td>
</tr>
</table>
</body>
</html>
```

Unter Umständen ist es sinnvoll, das Verhalten der XSLT-Formatvorlage während der Laufzeit zu ändern, um Informationen hinzuzufügen, die in den XML-Einträgen nicht enthalten sind, oder um die Art der Ausgabe selbst zu ändern (beispiels-

weise von der HTML-Standardausgabe in XHTML). Zur Änderung des Verhaltens können Sie die Parameter mithilfe einer Parameterdatei an den XSLT-Prozess übergeben. Die Parameterdatei ist selbst ein XML-Dokument und enthält Anweisungen vom Typ `param`, die ähnlichen Anweisungen in der XSLT-Formatvorlagendatei entsprechen.

Beispiel: In der Formatvorlage wurden die beiden folgenden Parameter definiert, jedoch bei der vorigen Umsetzung nicht verwendet:

```
<xsl:param name="showUniversity"/>
<xsl:param name="headline"/>
```

Um das XML-Dokument mit diesen Parametern umzusetzen, speichern Sie eine Parameterdatei in einer Tabelle und verwenden Sie diese Datei mit der Funktion XSLTRANSFORM.

1. Erstellen Sie die Tabelle PARAM_TAB, um die Parameterdatei zu speichern:

   ```
   CREATE TABLE PARAM_TAB (DOCID INTEGER, PARAM VARCHAR(1000))~
   ```

2. Erstellen Sie die Parameterdatei wie folgt:

   ```
   INSERT INTO PARAM_TAB VALUES
        (1,
        '<?xml version="1.0"?>
        <params xmlns="http://www.ibm.com/XSLTransformParameters">
             <param name="showUniversity" value="true"/>
             <param name="headline">The student list</param>
        </params>'
        )~
   ```

3. Setzen Sie das XML-Dokument um, indem Sie die Funktion XSLTRANSFORM aufrufen:

   ```
   SELECT XSLTRANSFORM (XML_DOC USING XSLT_DOC WITH PARAM AS CLOB(1M) )
      FROM XML_DATA X , PARAM_TAB P, XML_TRANS
      WHERE X.DOCID=P.DOCID and XSLID = 1 ~
   ```

Die Ausgabe dieser Verarbeitung ist die folgende HTML-Datei:

```
<html>
<head>
<META http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<h1>The student list</h1>
<table border="1">
<th>
<tr>
<td width="80">StudentID</td>
<td width="200">Given Name</td>
<td width="200">Family Name</td>
<td width="50">Age</td>
<td width="200">University</td>
</tr>
</th>
<tr>
<td>1</td>
<td>Steffen</td><td>Siegmund</td>
<td>21</td>
<td>Rostock</td>
</tr>
<tr>
<td>2</td>
<td>Helena</td><td>Schmidt</td>
<td>23</td>
<td>Rostock</td>
```

```
</tr>
</table>
</body>
</html>
```

Zum Lernprogramm zurückkehren

Kapitel 3. XML-Speicher

XML-Dokumente, die Sie in Spalten vom Typ XML einfügen, können entweder im Standardspeicherobjekt oder direkt in der Basistabellenzeile gespeichert werden. Das Speichern in der Basistabellenzeile wird von Ihnen gesteuert und ist nur für kleine Objekte verfügbar. Größere Objekte werden stets im Standardspeicherobjekt gespeichert.

Die Entscheidung, Dokumente in der Basistabellenzeile zu speichern, hängt von Ihren Speicher- und Leistungsanforderungen sowie den entsprechenden Vor- und Nachteilen ab, die Sie zu akzeptieren bereit sind.

Speichern im XML-Speicherobjekt
Dies ist die Standardmethode zum Speichern von XML-Dokumenten. Dokumente mit einem Speicherbedarf von mehr als 32 KB oder einer Größe, die die Seitengröße überschreitet, werden stets im Standardspeicherobjekt gespeichert, unabhängig davon, welche Speicheroption Sie auswählen. Das Speichern im Standardspeicherobjekt ermöglicht Ihnen das Einfügen und Abrufen von XML-Dokumenten mit einer Größe von bis zu 2 Gigabyte.

Speichern in der Basistabellenzeile
Bei XML-Dokumenten mit einem Speicherbedarf bis 32 KB können Sie wählen, ob Sie die Dokumente direkt in der Basistabellenzeile speichern möchten. Diese Option kann die Leistung aller Operationen zum Abfragen, Einfügen, Aktualisieren oder Löschen von XML-Dokumenten verbessern, da weniger E/A-Operationen erforderlich sind.

Wenn Sie für die Tabelle die Datenzeilenkomprimierung aktivieren, werden XML-Dokumente bei der Speicherung im standardmäßigen XML-Speicherobjekt und in der Basistabellenzeile komprimiert. Mit der Komprimierung lassen sich die Speicherplatzanforderungen reduzieren und die Ein-/Ausgabeleistung von Operationen für XML-Dokumente verbessern.

XML-Speicherobjekt

Ähnlich wie LOB-Daten getrennt vom übrigen Inhalt einer Tabelle gespeichert werden, speichert der DB2-Datenbankserver XML-Daten, die in Tabellenspalten des Typs 'XML' enthalten sind, standardmäßig in XML-Speicherobjekten.

XML-Speicherobjekte sind von ihren übergeordneten Tabellenobjekten zwar getrennt, jedoch von ihnen abhängig. Für jeden XML-Wert, der in einer Zeile einer XML-Tabellenspalte gespeichert ist, verwaltet DB2 einen Datensatz, der als XML-Datenkennung (XDS = XML Data Specifier) bezeichnet wird und angibt, von welcher Position die auf der Platte gespeicherten XML-Daten aus dem zugehörigen XML-Speicherobjekt abgerufen werden können. Erfolgt die Speicherung im systemverwalteten Speicherbereich, weisen die Dateien, die den XML-Speicherobjekten zugeordnet sind, die Dateityperweiterung .xda auf. Ein XML-Speicherobjekt wird manchmal als XML-Datenbereich (XDA) bezeichnet.

Sie können XML-Dokumente mit einer Größe von bis zwei GB in einer Datenbank speichern. Da XML-Daten enorm groß sein können, ist es vielleicht sinnvoll, die Pufferaktivitäten für XML-Daten separat von den Pufferaktivitäten für andere Daten zu überwachen. Ein Reihe von Monitorelementen steht zur Verfügung, um Ihnen bei der Messung der Pufferpoolaktivitäten für XML-Speicherobjekte zu helfen.

Weitere Informationen zum Platzbedarf für XML-Spalten mit XML-Speicherobjekten finden Sie unter "Bytezähler" für XML-Spalten ohne Angabe des Parameters INLINE LENGTH in der Anweisung CREATE TABLE.

Speichern von XML in Basistabellenzeilen

Kleine und mittelgroße XML-Dokumente müssen nicht im standardmäßigen XML-Speicherobjekt gespeichert werden, sondern können wahlweise auch in der Zeile der Basistabelle gespeichert werden. Das Speichern von XML-Dokumenten in Zeilen ähnelt dem integrierten Speichern (INLINE-Methode) einer Instanz eines strukturierten Typs in der Zeile einer Tabelle.

Bevor Sie das Speichern in Basistabellenzeilen aktivieren, müssen Sie festlegen, wie viel Speicherplatz für das Speichern in Zeilen pro XML-Spalte zugeordnet werden soll. Die Größe des Speicherplatzes, der zugeordnet werden kann, hängt von der maximal verfügbaren Zeilengröße ab, die wiederum von der Seitengröße des Tabellenbereichs abhängt, in dem die Tabelle erstellt wird, sowie von den anderen Spalten, die als Teil der Tabelle angegeben werden. Informationen zum Berechnen des verfügbaren Zeilenspeicherplatzes finden Sie unter "Zeilengröße" und "Bytezähler" für XML-Spalten mit Angabe des Parameters INLINE LENGTH in der Anweisung CREATE TABLE.

Speichern in Basistabellenzeile aktivieren

Wenn Sie eine Tabelle mit einer XML-Spalte erstellen oder eine bestehende Tabelle mit einer XML-Spalte ändern, können Sie angeben, dass XML-Dokumente nicht im standardmäßigen XML-Speicherobjekt, sondern in Basistabellenzeilen gespeichert werden sollen. Um das Speichern in Basistabellen zu aktivieren, müssen Sie die Schlüsselwörter INLINE LENGTH in die Anweisung CREATE TABLE bzw. ALTER TABLE für jede XML-Spalte einschließen, für die das Speichern in Zeilen verwendet werden soll. Nach den Schlüsselwörtern INLINE LENGTH müssen Sie die maximale Größe in Byte für die XML-Dokumente angeben, die in einer Basistabellenzeile gespeichert werden sollen.

Bitte beachten Sie Folgendes: Wenn Sie die XML-Spalte einer bestehenden Tabelle ändern, werden die XML-Dokumente, die bereits in der betreffenden Zeile gespeichert sind, nicht automatisch in Basistabellenzeilen versetzt. Um diese XML-Dokumente zu versetzen, müssen Sie alle XML-Dokumente mit der Anweisung UPDATE aktualisieren.

Einschränkungen

Das Speichern in Basistabellenzeilen steht nur für XML-Dokumente zur Verfügung, die eine interne Darstellung von maximal 32 KB (oder weniger, wenn Ihre Zeilengröße geringer ist) aufweisen, abzüglich der erforderlichen Bytezahl für eine XML-Spalte mit Angabe der Option INLINE LENGTH. Bei einer Größe von 32 KB wird angenommen, dass der Tabellenbereich eine Seitengröße von 32 KB aufweist. Wenn Sie XML-Dokumente speichern, die die integrierte Länge (INLINE LENGTH) überschreiten, werden diese Dokumente automatisch im standardmäßigen XML-Speicherobjekt gespeichert.

Nachdem Sie eine integrierte Länge für eine XML-Spalte angegeben haben, können Sie diese Länge für das Speichern von XML-Dokumenten in Zeilen lediglich erhöhen, nicht jedoch reduzieren.

Beispiele

In nachstehendem Beispiel wird das Speichern von XML-Dokumenten in Basistabellenzeilen für die XML-Spalte DESCRIPTION der Tabelle PRODUCT in der Beispieldatenbank SAMPLE aktiviert. In diesem Beispiel wird die maximal zulässige integrierte Länge für das Speichern von XML-Dokumenten in der Basistabellenzeile auf 32000 Byte festgelegt, um dem für den Systemaufwand zusätzlich erforderlichen Speicherplatz Rechnung zu tragen. Bei Verwendung des Werts 32000 Byte wird angenommen, dass der Tabellenbereich eine Seitengröße von 32 KB aufweist. Nach dem Ändern der XML-Spalte wird eine Anweisung UPDATE verwendet, um die XML-Dokumente in die Basistabellenzeile zu versetzen:

```
ALTER TABLE PRODUCT
   ALTER COLUMN DESCRIPTION
   SET INLINE LENGTH 32000

UPDATE PRODUCT SET DESCRIPTION = DESCRIPTION
```

Im nachstehenden Beispiel wird eine Tabelle namens MYCUSTOMER erstellt, die der Tabelle CUSTOMER in der Beispieldatenbank SAMPLE ähnlich ist, mit der Ausnahme, dass für die XML-Spalte 'Info' das Speichern in Basistabellenzeilen aktiviert ist. Dokumente mit einer internen Darstellung von maximal 2000 Byte werden beim Einfügen in die Spalte 'Info' in der Basistabellenzeile gespeichert:

```
CREATE TABLE MYCUSTOMER (Cid BIGINT NOT NULL,
   Info XML INLINE LENGTH 2000,
   History XML,
   CONSTRAINT PK_CUSTOMER PRIMARY KEY (Cid)) in IBMDB2SAMPLEXML
```

Speicherbedarf für XML-Dokumente

Die Größe des Speichers, der von einem XML-Dokument in einer DB2-Datenbank belegt wird, wird durch die Anfangsgröße des Dokuments in unaufbereiteter Form sowie durch eine Reihe anderer Faktoren bestimmt.

In der folgenden Liste sind die wichtigsten dieser Faktoren aufgeführt:

Dokumentstruktur
: XML-Dokumente, die komplexe Markup-Befehle enthalten, benötigen mehr Speicherplatz als Dokumente mit einem einfachen Markup. Zum Beispiel belegt ein XML-Dokument mit zahlreichen verschachtelten Elementen, die jeweils eine kleine Menge Text enthalten oder kurze Attributwerte haben, mehr Speicher als ein XML-Dokument, dessen Hauptinhalt aus Text besteht.

Knotennamen
: Die Längen von Elementnamen, Attributnamen, Namensbereichspräfixen und ähnlichen, nicht inhaltlichen Daten wirken sich ebenfalls auf den Speicherbedarf aus. Jede Informationseinheit dieses Typs, die 4 Byte in unaufbereiteter Form überschreitet, wird zum Speichern komprimiert, sodass sich eine vergleichsweise höhere Speichereffizienz für längere Knotennamen ergibt.

Verhältnis von Attributen zu Elementen
: In der Regel gilt, dass die Größe des für ein XML-Dokument erforderlichen Speicherbereichs umso geringer ist, je mehr Attribute pro Element verwendet werden.

Codepage des Dokuments
XML-Dokumente mit einer Codierung, die mehr als ein Byte pro Zeichen verwendet, belegen mehr Speicher als Dokumente, die mit einem Einzelbytezeichensatz arbeiten.

Komprimierung
Wenn Sie für eine Tabelle, die XML-Spalten enthält, die Datenzeilenkomprimierung aktivieren, benötigen XML-Dokumente weniger Speicherplatz.

Die Datenkomprimierung im XML-Speicherobjekt einer Tabelle wird nicht unterstützt, wenn die Tabelle XML-Spalten enthält, die das XML-Satzformat von DB2 Version 9.5 oder älter verwenden. Wenn Sie eine solche Tabelle für die Datenzeilenkomprimierung aktivieren, werden nur die Tabellenzeilendaten im Tabellenobjekt komprimiert. Damit die Daten im XML-Speicherobjekt der Tabelle für die Komprimierung infrage kommen, müssen Sie die Tabelle mit der gespeicherten Prozedur ADMIN_MOVE_TABLE in das neue Format konvertieren und anschließend die Datenzeilenkomprimierung aktivieren.

Datentypen für die Archivierung von XML-Dokumenten

Serialisierte XML-Zeichenfolgedaten können zwar in einer Spalte mit beliebigem binären oder Zeichendatentyp gespeichert werden, Nicht-XML-Spalten sollten jedoch nur zur Archivierung von XML-Daten verwendet werden. Der am besten geeignete Spaltendatentyp für die Archivierung von XML-Daten ist ein Binärdatentyp, wie zum Beispiel BLOB.

Die Verwendung einer Zeichendatenspalte für die Archivierung macht eine Codepagekonvertierung erforderlich, wodurch das Dokument möglicherweise die Konsistenz mit seinem ursprünglichen Format verliert.

Kapitel 4. Einfügen von XML-Daten

Bevor Sie XML-Dokumente einfügen können, müssen Sie entweder eine Tabelle erstellen, die eine XML-Spalte enthält, oder einer bestehenden Tabelle eine XML-Spalte hinzufügen.

Erstellen von Tabellen mit XML-Spalten

Zum Erstellen von Tabellen mit XML-Spalten müssen Sie in der Anweisung CREATE TABLE Spalten mit dem Datentyp XML angeben. Eine Tabelle kann eine oder auch mehrere XML-Spalten enthalten.

Beim Definieren einer XML-Spalte wird keine Länge angegeben. Für serialisierte XML-Daten, die mit einer DB2-Datenbank ausgetauscht werden, gilt jedoch eine Beschränkung auf 2 GB pro Wert des Typs XML, sodass sich eine effektive Begrenzung von 2 GB für ein XML-Dokument ergibt.

Wie LOB-Spalten enthalten auch XML-Spalten nur einen Deskriptor für die Spalte. Die Daten werden separat gespeichert.

Anmerkung:

- Wenn Sie für die Tabelle die Datenzeilenkomprimierung aktivieren, benötigen XML-Dokumente weniger Speicherplatz.
- Kleine und mittelgroße XML-Dokumente müssen nicht im standardmäßigen XML-Speicherobjekt gespeichert werden, sondern können wahlweise auch in der Zeile der Basistabelle gespeichert werden.

Beispiel: Die Beispieldatenbank enthält eine Tabelle für Kundendaten, in der zwei XML-Spalten definiert sind. Die Definition hat folgendes Format:

```
CREATE TABLE Customer (Cid BIGINT NOT NULL PRIMARY KEY,
                       Info XML,
                       History XML)
```

Beispiel: Mit dem Vergleichselement VALIDATED wird festgestellt, ob der Wert in der angegebenen XML-Spalte auf seine Gültigkeit überprüft wurde. Mit dem Vergleichselement VALIDATED können Sie eine Prüfung auf Integritätsbedingungen für XML-Spalten in Tabellen definieren, um sicherzustellen, dass alle eingefügten bzw. aktualisierten Dokumente in einer Tabelle gültig sind.

```
CREATE TABLE TableValid (id BIGINT,
                         xmlcol XML,
                         CONSTRAINT valid_check CHECK (xmlcol IS VALIDATED))
```

Beispiel: Wenn Sie das Attribut COMPRESS auf YES einstellen, wird die Datenzeilenkomprimierung aktiviert. Die Zeilenkomprimierung wird für XML-Dokumente ausgeführt, die in XML-Spalten gespeichert sind. Dank der Datenkomprimierung auf Zeilenebene können wiederkehrende Muster durch kürzere Symbolzeichenfolgen ersetzt werden.

```
CREATE TABLE TableXmlCol (id BIGINT,
                          xmlcol XML) COMPRESS YES
```

Beispiel: Mit der folgenden Anweisung CREATE TABLE wird eine nach dem Besuchsdatum partitionierte Patiententabelle erstellt. Alle Einträge zwischen dem 1.

Januar 2000 und dem 31. Dezember 2006 befinden sich auf der ersten Partition. Die neueren Daten werden alle sechs Monate partitioniert.

```
CREATE TABLE Patients ( patientID BIGINT, visit_date DATE, diagInfo XML,
    prescription XML )
    INDEX IN indexTbsp LONG IN ltbsp
    PARTITION BY ( visit_date )
       ( STARTING '1/1/2000' ENDING '12/31/2006',
         STARTING '1/1/2007' ENDING '6/30/2007',
            ENDING '12/31/2007',
            ENDING '6/30/2008',
            ENDING '12/31/2008',
            ENDING '6/30/2009' );
```

Hinzufügen von XML-Spalten zu vorhandenen Tabellen

Um XML-Spalten zu einer vorhandenen Tabelle hinzuzufügen, müssen Sie die gewünschten Spalten einschließlich des Datentyps XML in der Anweisung ALTER TABLE mit der Klausel ADD angeben. Eine Tabelle kann eine oder auch mehrere XML-Spalten enthalten.

Beispiel: Die Beispieldatenbank enthält eine Tabelle für Kundendaten, in der zwei XML-Spalten definiert sind. Die Definition hat folgendes Format:

```
CREATE TABLE Customer (Cid BIGINT NOT NULL PRIMARY KEY,
                                            Info XML,
                                            History XML)
```

Erstellen Sie eine Tabelle mit dem Namen 'MyCustomer', bei der es sich um eine Kopie der Tabelle 'Customer' handelt. Fügen Sie anschließend eine XML-Spalte hinzu, um die Kundenpräferenzen zu beschreiben:

```
CREATE TABLE MyCustomer LIKE Customer;
ALTER TABLE MyCustomer ADD COLUMN Preferences XML;
```

Beispiel: Wenn Sie das Attribut COMPRESS auf YES einstellen, wird die Datenzeilenkomprimierung aktiviert. Die Zeilenkomprimierung wird für XML-Dokumente ausgeführt, die in XML-Spalten gespeichert sind. Dank der Datenkomprimierung auf Zeilenebene können wiederkehrende Muster durch kürzere Symbolzeichenfolgen ersetzt werden.

```
ALTER TABLE MyCustomer ADD COLUMN Preferences XML COMPRESS YES;
```

Beispiel: Mit der folgenden Anweisung CREATE TABLE wird eine nach dem Besuchsdatum partitionierte Patiententabelle erstellt. Alle Einträge zwischen dem 1. Januar 2000 und dem 31. Dezember 2006 befinden sich auf der ersten Partition. Die neueren Daten werden alle sechs Monate partitioniert.

```
CREATE TABLE Patients ( patientID INT, Name Varchar(20), visit_date DATE,
      diagInfo XML )
    PARTITION BY ( visit_date )
       ( STARTING '1/1/2000' ENDING '12/31/2006',
         STARTING '1/1/2007' ENDING '6/30/2007',
            ENDING '12/31/2007',
            ENDING '6/30/2008',
            ENDING '12/31/2008',
            ENDING '6/30/2009' );
```

Mit der folgenden Anweisung ALTER TABLE wird eine weitere XML-Spalte für Angaben zu den Verschreibungen für die Patienten hinzugefügt:

```
ALTER TABLE Patients ADD COLUMN prescription XML ;
```

Einfügen in XML-Spalten

Zum Einfügen von Daten in eine XML-Spalte können Sie die SQL-Anweisung INSERT benutzen. Als Eingabe für die XML-Spalte muss ein korrekt formatiertes XML-Dokument angegeben werden, das den Angaben in der XML 1.0-Spezifikation entspricht. Als Anwendungsdatentyp kann ein XML-, Zeichen- oder Binärdatentyp verwendet werden.

Es wird empfohlen, XML-Daten aus Hostvariablen anstatt aus Literalelementen einzufügen, sodass der DB2-Datenbankserver den Datentyp der Hostvariablen verwenden kann, um bestimmte Codierungsinformationen zu ermitteln.

XML-Daten werden in einer Anwendung im serialisierten Zeichenfolgeformat angegeben. Wenn Sie Daten in eine XML-Spalte einfügen, müssen die Daten in das entsprechende hierarchische XML-Format umgewandelt werden. Wenn als Anwendungsdatentyp ein XML-Datentyp verwendet wird, führt der DB2-Datenbankserver diese Operation implizit aus. Wenn als Anwendungsdatentyp kein XML-Typ verwendet wird, können Sie die Funktion XMLPARSE explizit aufrufen, wenn Sie die Einfügeoperation ausführen, um die Daten aus dem serialisierten Zeichenfolgeformat ins hierarchische XML-Format umzusetzen.

Während der Dokumenteinfügung können Sie das XML-Dokument auch auf der Basis eines registrierten XML-Schemas prüfen. Dieser Arbeitsschritt kann mit der Funktion XMLVALIDATE ausgeführt werden.

In den folgenden Beispielen wird dargestellt, wie XML-Daten in XML-Spalten eingefügt werden können. In den Beispielen wird mit der Tabelle 'MyCustomer' gearbeitet, bei der es sich um eine Kopie der Beispieltabelle 'Customer' handelt. Die XML-Daten, die eingefügt werden sollen, sind in der Datei 'c6.xml' gespeichert und haben folgendes Format:

```
<customerinfo Cid="1015">
  <name>Christine Haas</name>
  <addr country="Canada">
    <street>12 Topgrove</street>
        <city>Toronto</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>N8X-7F8</pcode-zip>
  </addr>
  <phone type="work">905-555-5238</phone>
  <phone type="home">416-555-2934</phone>
</customerinfo>
```

Beispiel: In einer JDBC-Anwendung müssen Sie die XML-Daten aus der Datei 'c6.xml' als Binärdaten einlesen und diese dann in eine XML-Spalte einfügen.

```
PreparedStatement insertStmt = null;
String sqls = null;
int cid = 1015;
sqls = "INSERT INTO MyCustomer (Cid, Info) VALUES (?, ?)";
insertStmt = conn.prepareStatement(sqls);
insertStmt.setInt(1, cid);
File file = new File("c6.xml");
insertStmt.setBinaryStream(2, new FileInputStream(file), (int)file.length());
insertStmt.executeUpdate();
```

Beispiel: In einer eingebetteten statischen C-Anwendung müssen Sie die Daten aus einer binären XML-Hostvariablen in eine XML-Spalte einfügen:

```
EXEC SQL BEGIN DECLARE SECTION;
      sqlint64 cid;
      SQL TYPE IS XML AS BLOB (10K) xml_hostvar;
```

```
EXEC SQL END DECLARE SECTION;
...
cid=1015;
/* Read data from file c6.xml into xml_hostvar */
...
EXEC SQL INSERT INTO MyCustomer (Cid,Info) VALUES (:cid, :xml_hostvar);
```

XML-Parsing

Als XML-Parsing wird der Prozess bezeichnet, bei dem XML-Daten aus dem serialisierten Zeichenfolgeformat in das entsprechende hierarchische Format umgewandelt werden.

Das Parsing kann vom DB2-Datenbankserver implizit ausgeführt werden. Alternativ hierzu können Sie das XML-Parsing auch explizit ausführen.

Das *implizite XML-Parsing* wird in den folgenden Fällen verwendet:

- Wenn Sie Daten unter Verwendung einer Hostvariablen vom Typ 'XML' an den Datenbankserver übergeben oder eine Parametermarke vom Typ 'XML' verwenden.

  Der Datenbankserver führt das Parsing aus, wenn der Wert für die Hostvariable oder die Parametermarke zur Verwendung bei der Anweisungsverarbeitung gebunden wird.

  In diesem Fall muss das Parsing implizit ausgeführt werden.
- Wenn Sie in einer Anweisung INSERT, UPDATE, DELETE oder MERGE eine Hostvariable, Parametermarke oder einen SQL-Ausdruck mit einem Zeichenfolgedatentyp (Zeichen, Grafik oder Binär) einer XML-Spalte zuordnen. Das Parsing wird ausgeführt, wenn der SQL-Compiler die Funktion XMLPARSE implizit zur Anweisung hinzufügt.

Das *explizite XML-Parsing* wird ausgeführt, wenn Sie die Funktion XMLPARSE für die XML-Eingabedaten aufrufen. Das Ergebnis von XMLPARSE kann in einem beliebigen Kontext verwendet werden, der einen XML-Datentyp akzeptiert. Beispiel: Sie können das Ergebnis einer XML-Spalte zuordnen oder als Parameter vom Typ 'XML' für eine gespeicherte Prozedur verwenden.

Die Funktion XMLPARSE verwendet als Eingabe einen Nicht-XML-Datentyp, einen Zeichendatentyp oder einen binären Datentyp. Für Anwendungen mit eingebettetem dynamischem SQL müssen Sie die Parametermarke, die das Eingabedokument für XMLPARSE darstellt, in den geeigneten Datentyp umsetzen. Beispiel:

```
INSERT INTO MyCustomer (Cid, Info)
  VALUES (?, xmlparse(document cast(? as clob(1k)) preserve whitespace))
```

Bei Anwendungen mit statischem eingebettetem SQL darf ein Hostvariablenargument der Funktion XMLPARSE nicht als XML-Typ (Typ XML AS BLOB, XML AS CLOB oder XML AS DBCLOB) deklariert werden.

XML-Parsing und Verarbeitung von Leerzeichen

Während des impliziten oder expliziten XML-Parsings können Sie die Beibehaltung oder Entfernung von Leerzeichen am Anfang oder Ende steuern, wenn die Daten in der Datenbank gespeichert werden.

Gemäß dem XML-Standard werden als Leerzeichen alle Zeichen betrachtet, die zur besseren Lesbarkeit in ein Dokument eingefügt werden. Hierzu gehören neben den Leerzeichen selbst (U+0020) auch Zeilenschaltungszeichen (U+000D), Zeilenvor-

schubzeichen (U+000A) und Tabulatorzeichen (U+0009). Wenn eines dieser Zeichen in einer Textzeichenfolge enthalten ist, werden sie nicht als Leerzeichen betrachtet.

Als *Begrenzungsleerzeichen* werden Leerzeichen bezeichnet, die zwischen Elementen eingefügt sind. Im folgenden Dokument sind z. B. die Leerzeichen zwischen <a> und sowie zwischen und Begrenzungsleerzeichen.

```
<a> <b> und zwischen </b> </a>
```

Beim expliziten Aufrufen von XMLPARSE verwenden Sie die Option STRIP WHITESPACE oder PRESERVE WHITESPACE, um die Beibehaltung von Begrenzungsleerzeichen zu steuern. Standardmäßig werden die Begrenzungsleerzeichen entfernt.

Beim impliziten XML-Parsing gilt Folgendes:

- Wenn als Eingabedatentyp kein XML-Typ verwendet wird oder die Daten nicht in einen XML-Datentyp umgesetzt werden, dann entfernt der DB2-Datenbankserver die Leerzeichen in jedem Fall.
- Wenn als Eingabedatentyp ein XML-Datentyp verwendet wird, können Sie das Sonderregister CURRENT IMPLICIT XMLPARSE OPTION verwenden, um die Beibehaltung von Begrenzungsleerzeichen zu steuern. Dieses Sonderregister kann auf den Wert STRIP WHITESPACE oder PRESERVE WHITESPACE eingestellt werden. Standardmäßig werden die Begrenzungsleerzeichen entfernt.

Wenn Sie mit der XML-Gültigkeitsprüfung arbeiten, ignoriert der DB2-Datenbankserver das Sonderregister CURRENT IMPLICIT XMLPARSE OPTION und verwendet ausschließlich die Prüfregeln, um in den folgenden Fällen festzustellen, ob die Leerzeichen beibehalten oder entfernt werden sollen:

```
xmlvalidate(? ACCORDING TO XMLSCHEMA ID schemaname)
xmlvalidate(?)
xmlvalidate(:hvxml ACCORDING TO XMLSCHEMA ID schemaname)
xmlvalidate(:hvxml)
xmlvalidate(cast(? as xml) ACCORDING TO XMLSCHEMA ID schemaname)
xmlvalidate(cast(? as xml))
```

In diesen Fällen steht das Fragezeichen (?) für XML-Daten und `:hvxml` für eine XML-Hostvariable.

Informationen zu den Auswirkungen der XML-Gültigkeitsprüfung auf die Verarbeitung von Leerzeichen finden Sie unter XML-Gültigkeitsprüfung.

Der XML-Standard gibt das Attribut `xml:space` an, mit dem die Entfernung oder Beibehaltung von Leerzeichen in XML-Daten gesteuert werden kann. Die Attribute `xml:space` überschreiben alle Leerzeicheneinstellungen für das implizite oder explizite XML-Parsing.

Im folgenden Dokument beispielsweise werden die Leerzeichen direkt vor und nach stets beibehalten. Hierbei spielen eventuell definierte Optionen für das XML-Parsing keine Rolle, weil die Leerzeichen sich innerhalb eines Knotens mit dem Attribut `xml:space="preserve"` befinden:

```
<a xml:space="preserve"> <b> <c>c</c>b </b></a>
```

Im folgenden Dokument hingegen können die Leerzeichen direkt vor und nach mit den Optionen für das XML-Parsing gesteuert werden, weil die Leerzeichen sich in einem Knoten mit dem Attribut `xml:space="default"` befinden:

```
<a xml:space="default"> <b> <c>c</c>b </b></a>
```

XML-Parsing in Nicht-Unicode-Datenbanken

Wird ein XML-Dokument in eine Nicht-Unicode-Datenbank übergeben, kann die Codepagekonvertierung erstmals bei der Übergabe des Dokuments vom Client an den Zieldatenbankserver und anschließend bei der Übergabe des Dokuments an den DB2-XML-Parser stattfinden. Bei der Übergabe eines XML-Dokuments mit einer Hostvariablen oder einer Parametermarke vom Typ XML wird eine Codepagekonvertierung verhindert. Bei der Übergabe eines XML-Dokuments mit einem Zeichendatentyp (CHAR, VARCHAR, CLOB oder LONG VARCHAR) kann die Codepagekonvertierung zur Verwendung von Substitutionszeichen führen, die alle Zeichen in den XML-Daten ersetzen, die nicht zur Codepage der Zieldatenbank gehören.

Um die Verwendung von Substitutionszeichen und damit eine mögliche Beeinträchtigung der eingefügten XML-Daten zu verhindern, müssen Sie sicherstellen, dass beim Parsen von XML-Daten mit einem Zeichendatentyp alle Codepunkte im Quellendokument Teil der Codepage der Zieldatenbank sind. Für alle Zeichen, die nicht zu dieser Codepage gehören, können Sie eine Dezimal- oder Hexadezimalzeicheneinheit als Referenz zur Angabe des korrekten Unicode-Codepunkts verwenden. So kann beispielsweise entweder &#x003E oder &#0062 verwendet werden, um das Operatorzeichen > (größer als) anzugeben.

Sie können auch den Konfigurationsparameter ENABLE_XMLCHAR verwenden, um zu steuern, ob das XML-Parsing für Zeichendatentypen aktiviert sein soll oder nicht. Wird der Parameter ENABLE_XMLCHAR auf "NO" gesetzt, wird sowohl das explizite als auch das implizite XML-Parsing bei Verwendung von Zeichendatentypen blockiert.

XML-Parsing und Dokumenttypdeklarationen

Wenn die Eingabedaten eine interne Dokumenttypdeklaration (DTD) oder Verweise auf eine externe DTD enthalten, überprüft der XML-Parsingprozess auch die Syntax dieser DTDs. Darüber hinaus werden beim Parsingprozess die folgenden Operationen ausgeführt:

- Anwenden der Standardwerte, die von den internen und externen DTDs definiert wurden.
- Erweitern der Entitätsverweise und Parameterentitäten.

Beispiele

In den folgenden Beispielen wird dargestellt, wie Leerzeichen in einem XML-Dokument in verschiedenen Situationen verarbeitet werden.

Beispiel: Die Datei 'c8.xml' enthält das folgende Dokument:

```
<customerinfo xml:space="preserve" Cid='1008'>
    <name>Kathy Smith</name>
  <addr country='Canada'>
    <street>14 Rosewood</street>
        <city>Toronto</city>
    <prov-state>Ontario</prov-state>
        <pcode-zip>M6W 1E6</pcode-zip>
  </addr>
  <phone type='work'>416-555-3333</phone>
</customerinfo>
```

In einer JDBC-Anwendung müssen Sie das XML-Dokument aus der Datei lesen und die Daten in die XML-Spalte 'INFO' der Tabelle 'MYCUSTOMER' einfügen, bei

der es sich um eine Kopie der Beispieltabelle 'Customer' handelt. Weisen Sie den DB2-Datenbankserver zur Ausführung einer impliziten XML-Parsingoperation an.

```
PreparedStatement insertStmt = null;
String sqls = null;
int cid = 1008;
sqls = "INSERT INTO MyCustomer (Cid, Info) VALUES (?, ?)";
insertStmt = conn.prepareStatement(sqls);
insertStmt.setInt(1, cid);
File file = new File("c8.xml");
insertStmt.setBinaryStream(2, new FileInputStream(file), (int)file.length());
insertStmt.executeUpdate();
```

Da für die Behandlung von Leerzeichen keine Angaben gemacht wurden, wird davon ausgegangen, dass die Leerzeichen entsprechend der Standardeinstellung entfernt werden sollen. Das Dokument enthält jedoch das Attribut `xml:space="preserve"`, das angibt, dass die Leerzeichen beibehalten werden sollen. Dies bedeutet, dass die Zeilenschaltungs-, Zeilenvorschub- und Leerzeichen zwischen den Elementen des Dokuments nicht entfernt werden.

Wenn Sie die gespeicherten Daten abrufen, dann wird folgender Inhalt aufgelistet:

```
<customerinfo xml:space="preserve" Cid='1008'>
    <name>Kathy Smith</name>
  <addr country='Canada'>
    <street>14 Rosewood</street>
        <city>Toronto</city>
    <prov-state>Ontario</prov-state>
        <pcode-zip>M6W 1E6</pcode-zip>
  </addr>
  <phone type='work'>416-555-3333</phone>
</customerinfo>
```

Beispiel: Das folgende Dokument ist in der BLOB-Hostvariablen blob_hostvar enthalten.

```
<customerinfo xml:space="default" Cid='1009'>
    <name>Kathy Smith</name>
  <addr country='Canada'>
    <street>15 Rosewood</street>
        <city>Toronto</city>
    <prov-state>Ontario</prov-state>
        <pcode-zip>M6W 1E6</pcode-zip>
  </addr>
  <phone type='work'>416-555-4444</phone>
</customerinfo>
```

In einer statischen eingebetteten C-Anwendung müssen Sie das Dokument aus der Hostvariablen in die XML-Spalte 'Info' der Tabelle 'MyCustomer' einfügen. Die Hostvariable weist keinen XML-Typ auf, sodass XMLPARSE explizit ausgeführt werden muss. Geben Sie STRIP WHITESPACE an, um Begrenzungsleerzeichen zu entfernen.

```
EXEC SQL BEGIN DECLARE SECTION;
      SQL TYPE BLOB (10K) blob_hostvar;
EXEC SQL END DECLARE SECTION;
...
EXEC SQL INSERT INTO MyCustomer (Cid, Info)
    VALUES (1009,
    XMLPARSE(DOCUMENT :blob_hostvar STRIP WHITESPACE));
```

Das Dokument enthält das Attribut `xml:space="default"`, sodass die XMLPARSE-Spezifikation STRIP WHITESPACE die Verarbeitung von Leerzeichen steuert. Dies bedeutet, dass die Zeilenschaltungs-, Zeilenvorschub- und Leerzeichen zwischen den Elementen des Dokuments entfernt werden.

Wenn Sie die gespeicherten Daten abrufen, erhalten Sie eine einzige Zeile mit dem folgenden Inhalt:

```
<customerinfo xml:space="default" Cid='1009'>
<name>Kathy Smith</name><addr country='Canada'><street>15 Rosewood</street>
<city>Toronto</city><prov-state>Ontario</prov-state><pcode-zip>M6W 1E6</pcode-zip>
</addr><phone type='work'>416-555-4444</phone></customerinfo>
```

Beispiel: In einer C-Anwendung enthält die Hostvariable clob_hostvar das folgende Dokument, das eine interne DTD umfasst:

```
<!DOCTYPE prod [<!ELEMENT description (name,details,price,weight)>
   <!ELEMENT name (#PCDATA)>
   <!ELEMENT details (#PCDATA)>
   <!ELEMENT price (#PCDATA)>
   <!ELEMENT weight (#PCDATA)>
   <!ENTITY desc  "Anvil">
]>
<product pid=''110-100-01'' >
 <description>
 <name>&desc;</name>
 <details>Very heavy</details>
 <price>         9.99            </price>
 <weight>1 kg</weight>
 </description>
</product>'
```

Fügen Sie die Daten in die Tabelle 'MYPRODUCT' ein, die eine Kopie der Beispieltabelle 'PRODUCT' darstellt:

```
EXEC SQL BEGIN DECLARE SECTION;
      SQL TYPE CLOB (10K) clob_hostvar;
EXEC SQL END DECLARE SECTION;
...
EXEC SQL insert into
    Product ( pid, name, Price, PromoPrice, PromoStart, PromoEnd, description )
   values ( '110-100-01','Anvil', 9.99, 7.99, '11-02-2004','12-02-2004',
    XMLPARSE ( DOCUMENT :clob_hostvar STRIP WHITESPACE ));
```

XMLPARSE gibt an, dass die Leerzeichen entfernt werden sollen. Aus diesem Grund werden Begrenzungsleerzeichen im Dokument entfernt. Darüber hinaus wird bei der Ausführung von XMLPARSE durch den Datenbankserver der Entitätsverweis `&desc;` durch den zugehörigen Wert ersetzt.

Wenn Sie die gespeicherten Daten abrufen, erhalten Sie eine einzige Zeile mit dem folgenden Inhalt:

```
<product pid="110-100-01"><description><name>Anvil
</name><details>Very heavy</details><price>         9.99            </price>
<weight>1 kg</weight></description></product>
```

XML-Datenintegrität

Wenn Sie sicherstellen müssen, dass Ihre XML-Dokumente bestimmten Regeln entsprechen oder bestimmte Verarbeitungsvoraussetzungen erfüllen, können Sie zusätzliche XML-Datenintegritätsprüfungen durchführen oder zusätzliche Bedingungen angeben, die erfüllt sein müssen, bevor eine Aktion ausgeführt wird.

Zur Sicherstellung der XML-Datenintegrität steht eine Reihe von Methoden zur Verfügung. Welche Methode Sie auswählen, hängt von Ihren jeweiligen Anforderungen an die Datenintegrität und Verarbeitung ab.

Beim Indexieren von XML-Dokumenten können Sie die Eindeutigkeit innerhalb einer XML-Spalte für alle diejenigen Dokumente erzwingen, deren Knoten durch das

XML-Muster qualifiziert werden, anhand dessen die Indexierung vorgenommen wird. Der Abschnitt "Semantik des Schlüsselworts UNIQUE" enthält weitere Informationen hierzu.

Prüfungen auf Integritätsbedingungen für XML-Spalten

Mithilfe einer Prüfung auf Integritätsbedingungen können bestimmte Einschränkungen für XML-Spalten festgelegt werden. Integritätsbedingungen werden umgesetzt, sobald versucht wird, Daten in der betreffenden XML-Spalte einzufügen oder zu aktualisieren. Hierbei wird die entsprechende Operation nur dann ausgeführt, wenn die von der Integritätsbedingung angegebenen Kriterien zutreffen, d. h. wahr (true) sind.

Beim Arbeiten mit XML-Dokumenten ist es wichtig zu wissen, ob die betreffenden Dokumente bereits anhand von XML-Schemata auf Gültigkeit geprüft worden sind oder nicht. Wenn Sie sicherstellen müssen, dass Sie lediglich diejenigen Dokumente abfragen, einfügen, aktualisieren oder löschen, die bestimmte Prüfkriterien erfüllen, müssen Sie zum Angeben der entsprechenden Kriterien das Vergleichselement VALIDATED verwenden. Bitte beachten Sie, dass mit einer Prüfbedingung in keinem Fall XML-Dokumente selbst geprüft werden, sondern lediglich getestet wird, ob XML-Dokumente bereits auf Gültigkeit geprüft worden sind oder nicht.[2]

Das Vergleichselement VALIDATED wertet den Status der Gültigkeitsprüfung des Wertes aus, der von *XML-Ausdruck* angegeben ist und der einen XML-Datentyp aufweisen muss. Wird die optionale Entsprechungsklausel (ACCORDING TO) nicht angegeben, hat das für die Gültigkeitsprüfung verwendete XML-Schema keinen Einfluss auf das Ergebnis. Prüfbedingungen führen keine Prüfung der XML-Dokumente selbst aus. Es wird lediglich der aktuelle Status der Gültigkeitsprüfung des Dokuments getestet (IS VALIDATED oder IS NOT VALIDATED). Wird die Entsprechungsklausel (ACCORDING TO) angegeben, muss das XML-Schema, das zur Prüfung des von *XML-Ausdruck* angegebenen Wertes verwendet wird, ein von der Entsprechungsklausel identifiziertes XML-Schema sein. XML-Schemata müssen für das XML-Schema-Repository registriert werden, bevor in einem Vergleichselement vom Typ VALIDATED auf sie verwiesen werden kann.

Anmerkungen:

- Prüfbedingungen hängen von den XML-Schemata ab, auf die sie verweisen. Wenn das XSR-Objekt eines XML-Schemas gelöscht wird, werden auch sämtliche Integritätsbedingungen gelöscht, die auf das Schema verweisen.
- XML-Spalten unterstützen Integritätsbedingungen vom Typ NOT NULL.
- XML-Spalten unterstützen informative Integrationsbedingungen, die für die XML-Gültigkeitsprüfung definiert wurden.

Auswertung von Prüfungen auf Integritätsbedingungen

Prüfungen auf Integritätsbedingungen testen den Prüfungsstatus von Dokumenten aufgrundlage des Ergebnisses des Vergleichselements IS VALIDATED. Wenn die angegebene Bedingung erfüllt ist, ergibt die Auswertung der Integritätsbedingung den Wert 'true' (wahr), andernfalls ergibt die Auswertung den Wert 'false' (falsch). Ist der von *XML-Ausdruck* angegebene Wert null, ist das Ergebnis des Vergleichselements unbekannt.

2. Wenn Sie XML-Dokumente automatisch prüfen müssen, bevor sie in einer XML-Spalte gespeichert werden, können Sie einen Trigger vom Typ BEFORE verwenden.

Das Ergebnis des Vergleichselements VALIDATED ist **true** (wahr), wenn der von *XML-Ausdruck* angegebene Wert ungleich null ist UND Folgendes gilt:

- Es wurde keine Entsprechungsklausel (ACCORDING TO) angegeben, und der von *XML-Ausdruck* angegebene Wert wurde geprüft, ODER
- Es wurde eine Entsprechungsklausel (ACCORDING TO) angegeben, und der von *XML-Ausdruck* angegebene Wert wurde anhand eines der von der Entsprechungsklausel angegebenen XML-Schemata geprüft.

Das Ergebnis des Vergleichselements ist **false** (falsch), wenn der von *XML-Ausdruck* angegebene Wert ungleich null ist UND Folgendes gilt:

- Es wurde keine Entsprechungsklausel (ACCORDING TO) angegeben, und der von *XML-Ausdruck* angegebene Wert wurde nicht geprüft, ODER
- Es wurde eine Entsprechungsklausel (ACCORDING TO) angegeben, und der von *XML-Ausdruck* angegebene Wert wurde nicht anhand eines der von der Entsprechungsklausel angegebenen XML-Schemata geprüft.

In Fällen, in denen die optionale Entsprechungsklausel (ACCORDING TO) angegeben wurde, gibt das Vergleichselement IS NOT VALIDATED den Wert 'true' (wahr) zurück, wenn der von *XML-Ausdruck* angegebene Wert nicht geprüft wurde oder wenn der von *XML-Ausdruck* angegebene Wert zwar geprüft wurde, nicht jedoch entsprechend einem der angegebenen XML-Schemata.

Äquivalenz von Ausdrücken

Das Vergleichselement VALIDATED

```
  Wert1
IS NOT VALIDATED optionale_Klausel
```

ist äquivalent zur Suchbedingung

```
  NOT(Wert1 IS VALIDATED
optionale_Klausel)
```

Beispiele

Beispiel: Wählen Sie nur geprüfte XML-Dokumente aus. Nehmen Sie an, dass die Spalte XMLCOL in der Tabelle T1 definiert ist. Rufen Sie nur diejenigen XML-Werte ab, die anhand eines XML-Schemas geprüft worden sind:

```
SELECT XMLCOL FROM T1
  WHERE XMLCOL IS VALIDATED
```

Beispiel: Erzwingen Sie die Regel, dass Werte erst nach erfolgter Prüfung eingefügt oder aktualisiert werden können. Nehmen Sie an, dass die Spalte XMLCOL in der Tabelle T1 definiert ist, und fügen Sie der Spalte XMLCOL eine Prüfung auf Integritätsbedingungen hinzu:

```
ALTER TABLE T1 ADD CONSTRAINT CK_VALIDATED
  CHECK (XMLCOL IS VALIDATED)
```

Beispiel: Die Integritätsbedingung INFO_CONSTRAINT ist eine informative Integrationsbedingung. Die Regel, dass Werte erst nach erfolgter Prüfung eingefügt oder aktualisiert werden können, wird nicht erzwungen.

```
CREATE TABLE xmltab (ID INT,
     DOC XML, CONSTRAINT INFO_CONSTRAINT CHECK (DOC IS VALIDATED) NOT ENFORCED)
```

Informative Integrationsbedingungen werden zur Verbesserung der Abfrageleistung verwendet.

Triggerverarbeitung von XML-Daten

Wenn Sie Trigger erstellen möchten, die auf Einfüge-, Aktualisierungs- oder Löschoperationen für XML-Daten reagieren, verwenden Sie die Anweisung CREATE TRIGGER, um Trigger vom Typ BEFORE oder AFTER mit der Option INSERT, UPDATE oder DELETE für XML-Spalten zu erstellen.

Im Triggerhauptteil können Übergangsvariablen, die auf Spalten vom Typ XML in den betreffenden Zeilen verweisen, nur für die Überprüfung durch die Funktion XMLVALIDATE verwendet werden, um XML-Spaltenwerte auf NULL zu setzen oder um XML-Spaltenwerte unverändert zu lassen.

Ein Trigger vom Typ BEFORE kann mit den Anweisungen INSERT oder UPDATE verwendet werden, um XML-Dokumente automatisch zu prüfen, bevor sie in einer XML-Spalte gespeichert werden. Die Gültigkeitsprüfung von XML-Dokumenten aufgrundlage von registrierten XML-Schemata ist optional, wird jedoch dringend empfohlen, wenn Zweifel an der Datenintegrität bestehen, da durch die Prüfung sichergestellt wird, dass ausschließlich gültige XML-Dokumente eingefügt oder aktualisiert werden.

Der Trigger wird aktiviert, wenn die für ihn definierte Bedingung eintritt. Wird keine Bedingung angegeben, wird der Trigger immer aktiviert. Wenn die Gültigkeitsprüfung von XML-Dokumenten aufgrundlage von XML-Schemata nur bei Bedarf ausgelöst werden soll, können Sie mit der Klausel WHEN des Triggers BEFORE eine Bedingung für die betreffende XML-Spalte angeben. In die Klausel WHEN wird der erforderliche Prüfstatus für die XML-Dokumente eingeschlossen, der angibt, dass die Dokumente zum Aktivieren des Triggers bereits geprüft sein müssen (IS VALIDATED) oder nicht (IS NOT VALIDATED). Optional ist es möglich, mit der Klausel ACCORDING TO XMLSCHEMA ein XML-Schema oder mehrere XML-Schemata anzugeben, das bzw. die der Trigger bei der Auswertung der Integritätsbedingung berücksichtigen soll.

Anmerkung: Ein Trigger mit der Klausel WHEN verursacht einen größeren Systemaufwand. Wenn vor dem Einfügen von XML-Dokumenten stets eine Gültigkeitsprüfung erfolgen soll, kann die Klausel WHEN ausgelassen werden.

Jeder Trigger, der auf ein XML-Schema verweist, ist von dem betreffenden Schema abhängig. Bevor auf ein XML-Schema verwiesen werden kann, muss es im XML-Schema-Repository registriert worden sein. Wenn das XML-Schema, von dem der Trigger abhängt, zu einem späteren Zeitpunkt aus dem XML-Schema-Repository gelöscht wird, wird der Trigger als funktionsunfähig markiert.

Beispiel 1: Erstellen Sie einen Trigger vom Typ BEFORE, der automatisch XML-Dokumente mit neuen Produktbeschreibungen auf Gültigkeit prüft, bevor diese Dokumente in die Tabelle PRODUCT der Beispieldatenbank SAMPLE eingefügt werden. Dieser Trigger wird stets aktiviert, bevor XML-Dokumente aktualisiert werden:

```
CREATE TRIGGER NEWPROD NO CASCADE BEFORE INSERT ON PRODUCT
    REFERENCING NEW AS N
  FOR EACH ROW MODE DB2SQL
  BEGIN ATOMIC
     SET (N.DESCRIPTION) = XMLVALIDATE(N.DESCRIPTION
     ACCORDING TO XMLSCHEMA URI 'http://posample.org/product.xsd');
  END
```

Beispiel 2: Nach der Weiterentwicklung des XML-Schemas 'product2.xsd' sind XML-Dokumente, die bereits gemäß dem ursprünglichen XML-Schema 'produc-

t.xsd' gültig waren, garantiert auch gemäß dem weiterentwickelten Schema gültig. Es kann jedoch sinnvoll sein sicherzustellen, dass auch alle Aktualisierungen dieser XML-Dokumente gemäß dem weiterentwickelten Schema 'product2.xsd' gültig sind. Nach der Registrierung des Schemas 'product2.xsd' beim XML-Schema-Repository können XML-Dokumente mithilfe eines Triggers vom Typ BEFORE UPDATE geprüft werden, bevor Aktualisierungen vorgenommen werden:

```
CREATE TRIGGER UPDPROD NO CASCADE BEFORE UPDATE ON PRODUCT
  REFERENCING NEW AS N
  FOR EACH ROW MODE DB2SQL
  BEGIN ATOMIC
    SET (N.DESCRIPTION) = XMLVALIDATE(N.DESCRIPTION
    ACCORDING TO XMLSCHEMA ID product2);
  END
```

Beispiel 3: Sie wollen eingefügte bzw. aktualisierte Kundenstammdaten in einer anderen Tabelle protokollieren. Dazu müssen Sie zwei Trigger erstellen: einen vom Typ AFTER INSERT für neu eingefügte Einträge und einen vom Typ AFTER UPDATE für aktualisierte Einträge. Im folgenden Beispiel werden die Trigger für die XML-Spalte 'Info' der Tabelle 'MyCustomer' erstellt. Bei dieser Tabelle handelt es sich um eine Kopie der Beispieltabelle 'Customer'. Die Trigger bewirken, dass ein Eintrag mit der Zeitmarke und der Kunden-ID in eine Tabelle mit dem Namen 'CustLog' geschrieben wird, sobald ein Eintrag in die Tabelle 'MyCustomer' eingefügt oder dort aktualisiert wird. In Beispiel 4 ist dargestellt, wie eine Kopie der eigentlichen Daten auch in der Tabelle 'CustLog' beibehalten wird.

Erstellen Sie zunächst den Trigger AFTER INSERT für die Tabelle 'MyCustomer':

```
CREATE TRIGGER INSAFTR
  AFTER INSERT ON MyCustomer
  REFERENCING NEW AS N
  FOR EACH ROW
  BEGIN ATOMIC
    INSERT INTO CustLog VALUES(N.CID, CURRENT TIMESTAMP, 'Insert');
  END
```

Erstellen Sie anschließend den Trigger AFTER UPDATE für die Tabelle 'MyCustomer':

```
CREATE TRIGGER UPDAFTR
  AFTER UPDATE OF Info
  ON MyCustomer
  REFERENCING NEW AS N
  FOR EACH ROW
  BEGIN ATOMIC
    INSERT INTO CustLog VALUES(N.CID, CURRENT TIMESTAMP, 'Update');
  END
```

Beispiel 4: In diesem Beispiel wird veranschaulicht, wie eine Tabelle so eingerichtet wird, dass sie als Prüfprotokoll für eingefügte oder aktualisierte Kundenstammdaten dient. Wie in Beispiel 3 erstellen Sie zwei Trigger, einen vom Typ AFTER INSERT für neu eingefügte Einträge und einen von Typ AFTER UPDATE für aktualisierte Einträge. Die Trigger werden für die XML-Spalte 'Info' der Tabelle 'MyCustomer' erstellt, bei der es sich um eine Kopie der Beispieltabelle 'Customer' handelt. Jedes Mal, wenn ein Eintrag in die Tabelle 'MyCustomer' eingefügt oder in dieser Tabelle aktualisiert wird, bewirken die Trigger, dass ein Eintrag mit der Zeitmarke, der Kunden-ID und dem Inhalt der XML-Spalet 'Info' in eine Tabelle mit dem Namen 'CustLog' geschrieben wird.

Erstellen Sie zunächst den Trigger AFTER INSERT für die Tabelle 'MyCustomer':

```
CREATE TRIGGER INSAFTR
  AFTER INSERT ON MyCustomer
  REFERENCING NEW AS N
  FOR EACH ROW
  BEGIN ATOMIC
    INSERT INTO CustLog VALUES(N.CID, CURRENT TIMESTAMP, 'Insert',
      (SELECT Info FROM MyCustomer WHERE CID = N.CID));
  END
```

Erstellen Sie anschließend den Trigger AFTER UPDATE für die Tabelle 'MyCustomer':

```
CREATE TRIGGER UPDAFTR
  AFTER UPDATE OF Info
  ON MyCustomer
  REFERENCING NEW AS N
  FOR EACH ROW
  BEGIN ATOMIC
    INSERT INTO CustLog VALUES(N.CID, CURRENT TIMESTAMP, 'Update',
      (SELECT Info FROM MyCustomer WHERE CID = N.CID));
  END
```

XML-Gültigkeitsprüfung

Die XML-Gültigkeitsprüfung ist der Prozess, mit dem festgelegt wird, ob die Struktur, der Inhalt und die Datentypen eines XML-Dokuments gültig sind. Außerdem werden bei der XML-Gültigkeitsprüfung *ignorierbare Leerzeichen* aus dem XML-Dokument entfernt.

Die Gültigkeitsprüfung ist optional, wird jedoch dringend empfohlen, wenn Zweifel an der Datenintegrität bestehen, da durch die Prüfung sichergestellt wird, dass XML-Dokumente nicht nur korrekt formatiert sind, sondern auch den von ihren jeweiligen XML-Schemata bereitgestellten Regeln entsprechen.

Bitte beachten Sie, dass Sie Ihre XML-Dokumente nur anhand von XML-Schemata auf Gültigkeit prüfen können. Die Prüfung von XML-Dokumenten anhand einer Dokumenttypdeklaration (DTD) wird nicht unterstützt.

XML-Dokumente werden mithilfe der Funktion XMLVALIDATE auf Gültigkeit geprüft. Sie können XMLVALIDATE mit einer SQL-Anweisung angeben, die XML-Dokumente in eine DB2-Datenbank einfügt oder dort aktualisiert. Zur automatischen Gültigkeitsprüfung von XML-Dokumenten kann die Funktion XMLVALIDATE auch durch einen Trigger vom Typ BEFORE in einer XML-Spalte aufgerufen werden. Um die Gültigkeitsprüfung von XML-Dokumenten zu erzwingen, müssen Sie eine Prüfung auf Integritätsbedingungen erstellen.

Bevor die Funktion XMLVALIDATE aufgerufen werden kann, müssen alle Schemadokumente, aus denen sich ein XML-Schema zusammensetzt, im integrierten XML-Schema-Repository registriert werden. Ein XML-Dokument muss sich selbst nicht in einer Datenbank befinden, um mit XMLVALIDATE geprüft werden zu können.

XML-Gültigkeitsprüfung und nicht relevante Leerzeichen

Gemäß dem XML-Standard werden als *Leerzeichen* alle Zeichen betrachtet, die zur besseren Lesbarkeit in ein Dokument eingefügt werden. Hierzu gehören neben den Leerzeichen selbst (U+0020) auch Zeilenschaltungszeichen (U+000D), Zeilenvorschubzeichen (U+000A) und Tabulatorzeichen (U+0009). Wenn eines dieser Zeichen in einer Textzeichenfolge enthalten ist, werden sie nicht als Leerzeichen betrachtet.

Als *nicht relevante Leerzeichen* gelten Leerzeichen, die aus einem XML-Dokument gelöscht werden können. Das XML-Schemadokument legt fest, welche Leerzeichen als nicht relevante Leerzeichen betrachtet werden. Wenn ein XML-Dokument einen komplexen, reinen Elementtyp (d. h. ein Element, das ausschließlich andere Elemente enthält) definiert, dann sind die Leerzeichen zwischen den Elementen nicht relevant und können ignoriert werden. Wenn das XML-Schema ein einfaches Element definiert, das einen anderen Datentyp als Zeichenfolgedaten (string) enthält, dann sind die Leerzeichen in diesem Element ebenfalls nicht relevant.

Beispiel: Das Element `description` im XML-Beispielschemadokument 'product.xsd' wird wie folgt definiert:

```
<xs:element name="description" minOccurs="0" maxOccurs="unbounded">
              <xs:complexType>
    <xs:sequence>
      <xs:element name="name" type="xs:string" minOccurs="0" />
      <xs:element name="details" type="xs:string" minOccurs="0" />
      <xs:element name="price" type="xs:decimal" minOccurs="0" />
      <xs:element name="weight" type="xs:string" minOccurs="0" />
 ...
 </xs:complexType>
  </xs:element>
```

Das Element `description` weist einen komplexen, reinen Elementtyp auf, da es ausschließlich andere Elemente enthält. Aus diesem Grund sind die Leerzeichen zwischen den Elementen des Elements `description` nicht relevant. Das Element `price` kann ebenfalls nicht relevante Leerzeichen enthalten, da es sich um ein einfaches Element handelt, das einen anderen Datentyp als den Zeichenfolgedatentyp 'string' enthält.

In der Funktion XMLVALIDATE können Sie das XML-Schemadokument, das für die Gültigkeitsprüfung verwendet werden soll, explizit angeben. Wenn Sie kein XML-Schemadokument angeben, sucht der DB2-Datenbankserver im Eingabedokument nach dem Attribut xsi:schemaLocation oder xsi:noNamespaceSchemaLocation, in dem das XML-Schemadokument angegeben wird. Die Attribute xsi:schemaLocation und xsi:noNamespaceSchemaLocation werden in der XML-Schemaspezifikation definiert und als *XML-Schemahinweise* bezeichnet. Das Attribut xsi:schemaLocation enthält mindestens ein Wertepaar, mit dem das XML-Schemadokument gesucht werden kann. Der erste Wert innerhalb des Wertepaares gibt einen Namensbereich an, der zweite Wert enthält einen Hinweis, der angibt, wo das XML-Schema für den Namensbereich abgelegt ist. Ein Wert für xsi:noNamespaceSchemaLocation enthält lediglich einen Hinweis. Wenn in der Funktion XMLVALIDATE ein XML-Schemadokument angegeben ist, überschreibt dieses das Attribut xsi:schemaLocation bzw. xsi:noNamespaceSchemaLocation.

In den folgenden Beispielen wird davon ausgegangen, dass das Schema `product` im XML-Schema-Repository (XSR) registriert wurde. Zur Registrierung können Sie eine Anweisung des Befehlszeilenprozessors wie die im Folgenden aufgeführte verwenden:

```
REGISTER XMLSCHEMA http://posample.org/product.xsd FROM product.xsd \
AS myschema.product
COMPLETE XMLSCHEMA myschema.product
```

Alternativ hierzu können Sie, da das XML-Schema aus nur einem Schemadokument besteht, auch eine einzige Anweisung benutzen, um das XML-Schema zu registrieren und die Registrierung abzuschließen.

```
REGISTER XMLSCHEMA http://posample.org/product.xsd FROM product.xsd \
AS myschema.product COMPLETE
```

Beispiel: Sie erstellen die Tabelle 'MyProduct' und geben hierzu Folgendes an:

```
CREATE TABLE MyProduct LIKE Product
```

Sie wollen das folgende Dokument in die XML-Spalte 'Info' der Tabelle 'MyProduct' einfügen und hierzu eine dynamische SQL-Anwendung benutzen. Sie wollen außerdem die Gültigkeit der XML-Daten auf der Basis des XML-Schemadokuments 'product.xsd' prüfen, das sich im XML-Schema-Repository auf demselben Datenbankserver wie die Tabelle 'MyProduct' befindet.

```
<product xmlns="http://posample.org" pid=''110-100-01'' >
 <description>
 <name>Anvil</name>
 <details>Very heavy</details>
 <price>         9.99            </price>
 <weight>1 kg</weight>
 </description>
</product>'
```

In Ihrer Anweisung INSERT gibt die Funktion XMLVALIDATE das XML-Schema an, das für die Gültigkeitsprüfung verwendet werden soll:

```
Insert into MyProduct
  (pid, name, Price, PromoPrice, PromoStart, PromoEnd, description)
  values ( '110-100-01','Anvil', 9.99, 7.99, '11-02-2004','12-02-2004',
 XMLVALIDATE(? ACCORDING TO XMLSCHEMA ID myschema.product))
```

Wenn Sie die gespeicherten Daten abrufen, dann können Sie erkennen, wo XMLVALIDATE nicht relevante Leerzeichen entfernt. Die abgerufenen Daten befinden sich in einer einzigen Zeile, die Folgendes enthält:

```
<product xmlns="http://posample.org" pid="110-100-01"><description><name>Anvil
</name><details>Very heavy</details><price>9.99</price><weight>1 kg</weight>
</description></product>
```

Das Schema `product` definiert die Leerzeichen, die die Elemente `name`, `details`, `price` und `weight` umgeben. Die Leerzeichen im Element 'price' sind hierbei als nicht relevant festgelegt, sodass sie von XMLVALIDATE entfernt werden.

Wenn Sie sich vergewissern wollen, dass nur solche Dokumente in eine XML-Spalte eingefügt bzw. nur solche Dokumente aus einer XML-Spalte abgerufen werden, deren Gültigkeit erfolgreich überprüft wurde, können Sie das Vergleichselement VALIDATED benutzen.

Um vor dem Einfügen oder Aktualisieren eines XML-Dokuments zu testen, ob das betreffende XML-Dokument auf Gültigkeit geprüft wurde, können Sie eine Prüfung auf Integritätsbedingungen erstellen, die das Vergleichselement VALIDATED in der XML-Spalte enthält. Um nur geprüfte Dokumente aus einer XML-Spalte abzurufen oder um nur Dokumente abzurufen, die ohne Gültigkeitsprüfung eingefügt wurden, können Sie das Vergleichselement VALIDATED in einer WHERE-Klausel verwenden. Wenn Sie prüfen müssen, ob ein XML-Dokument gemäß bestimmten XML-Schemata geprüft wurde, können Sie die betreffenden XML-Schemata mit dem Vergleichselement VALIDATED in die Klausel ACCORDING TO XMLSCHEMA einschließen.

Das Vergleichselement VALIDATED kann auch als Teil eines Triggers verwendet werden. Um die Gültigkeitsprüfung von bisher nicht geprüften XML-Dokumenten auszulösen, bevor diese in eine XML-Spalte eingefügt oder dort aktualisiert werden, können Sie zum Aufrufen der Funktion XMLVALIDATE einen Trigger vom Typ BEFORE erstellen, der das Vergleichselement VALIDATED in der XML-Spalte enthält.

Beispiel: Sie möchten nur XML-Dokumente aus der Spalte 'Info' der Tabelle 'MyCustomer' abrufen, deren Gültigkeit erfolgreich überprüft wurde. Führen Sie hierzu die folgende Anweisung SELECT aus:

```
SELECT Info FROM MyCustomer WHERE Info IS VALIDATED
```

Beispiel: Sie möchten nur XML-Dokumente in die Spalte 'Info' der Tabelle 'MyCustomer' einfügen, deren Gültigkeit erfolgreich überprüft wurde. Sie können eine Prüfung auf Integritätsbedingungen definieren, um die Durchsetzung dieser Bedingung zu erzwingen. Ändern Sie die Tabelle 'MyCustomer' wie folgt ab:

```
ALTER TABLE MyCustomer ADD CONSTRAINT CK_VALIDATED CHECK (Info IS VALIDATED)
```

Durch die Eingabe dieser Anweisung wird die Verwendung des Vergleichselements VALIDATED im vorherigen Beispiel unnötig, da nur gültige Dokumente in die Tabelle eingefügt bzw. aktualisiert werden können.

Beispiel: Sie möchten die Gültigkeit des folgenden Dokuments mit dem Schema 'customer' überprüfen, Sie wollen es jedoch nicht in einer Datenbank speichern.

```
<customerinfo xml:space="default"
    xmlns="http://posample.org"
  Cid='1011'>
    <name>Kathy Smith</name>
  <addr country='Canada'>
  <street>25 Rosewood</street>
      <city>Toronto</city>
  <prov-state>Ontario</prov-state>
      <pcode-zip>M6W 1E6</pcode-zip>
  </addr>
  <phone type='work'>416-555-6676</phone>
</customerinfo>
```

Sie haben das Dokument einer Anwendungsvariablen zugeordnet. Sie können eine Anweisung VALUES wie die folgende verwenden, um die Gültigkeitsprüfung durchzuführen:

```
VALUES XMLVALIDATE(? according to xmlschema id myschema.customer)
```

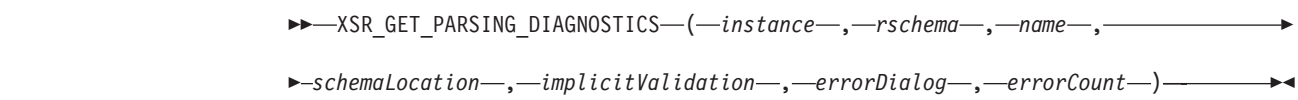
Dieses Dokument wurde auf der Basis des XML-Schemas als gültiges Dokument identifiziert, sodass die Anweisung VALUES eine Ergebnistabelle zurückgibt, die dieses Dokument enthält. Wenn das Dokument nicht gültig ist, wird von VALUES ein SQL-Fehler zurückgegeben.

Gespeicherte Prozedur XSR_GET_PARSING_DIAGNOSTICS

Die gespeicherte Prozedur XSR_GET_PARSING_DIAGNOSTICS generiert detaillierte Informationen zu den bei der Syntaxanalyse oder Gültigkeitsprüfung eines XML-Dokuments auftretenden Fehlern. Mit der gespeicherten Prozedur können sowohl Parsing- als auch Gültigkeitsfehler oder nur Parsing-Fehler dokumentiert werden.

Sie können diese gespeicherte Prozedur über das DB2-Befehlsfenster aufrufen oder sie zu einer Anwendung hinzufügen, wenn bei der Syntaxanalyse oder Gültigkeitsprüfung von XML-Dateien ein Fehler auftritt. Wenn beispielsweise bei der Gültigkeitsprüfung eines XML-Dokuments mit der Skalarfunktion XMLVALIDATE Fehler auftreten, können Sie mit dieser gespeicherten Prozedur detaillierte Fehlerinformationen generieren.

Syntax



Das Schema der gespeicherten Prozedur ist SYSPROC.

Berechtigung

XML-Schemaberechtigung: Das für die Gültigkeitsprüfung verwendete XML-Schema muss vor der Verwendung im XML-Schema-Repository registriert werden. Die Berechtigungs-ID der gespeicherten Prozedur muss zumindest über Folgendes verfügen:

- Berechtigung USAGE für das bei der Prüfung verwendete XML-Schema
- Berechtigung DBADM

Prozedurparameter

instance
: Ein Eingabeargument des Typs BLOB (30M), das den Inhalt des XML-Dokuments enthält. Ein XML-Dokument muss angegeben werden. Der Wert darf nicht NULL sein.

rschema
: Ein Eingabeargument des Typs VARCHAR(128), das den SQL-Schemateil des zweiteiligen XSR-Objektnamens angibt, der im XML-Schema-Repository registriert ist. Der Wert kann NULL sein. Ist dieser Wert NULL, wird davon ausgegangen, dass der SQL-Schemateil der aktuelle Wert des Sonderregisters CURRENT SCHEMA ist.

name
: Ein Eingabeargument des Typs VARCHAR(128), das den Schemanamen des zweiteiligen XSR-Objektnamens angibt, der im XML-Schema-Repository registriert ist. Die vollständige SQL-Kennung für das XML-Schema lautet *rschema.name*. Sie muss unter allen Objekten im XML-Schema-Repository (XSR) eindeutig sein. Der Wert kann NULL sein.

schemaLocation
: Ein Eingabeargument des Typs VARCHAR(1000), das die Schemaposition des primären XML-Schemadokuments angibt. Dieses Argument ist der externe Name des XML-Schemas, d. h., das primäre Dokument kann im XML-Instanzdokument mit dem Attribut `xsi:schemaLocation` angegeben werden. Der Wert kann NULL sein.

implicitValidation
: Ein Eingabeargument des Typs INTEGER, das angibt, ob die Schemaposition aus dem Instanzdokument zum Auffinden des XML-Schemas verwendet werden soll. Der Wert darf nicht NULL sein. Mögliche Werte sind 0 für 'nein' und 1 für 'ja'.

  0 Verwenden Sie nicht die Schemaposition aus dem Instanzdokument. Wenn ein Nullwert übergeben wird, können Sie das Schema mithilfe einer der folgenden Methoden angeben:

  - Durch Angeben des XSR-Objektnamens als Argumente *rschema* und *name* für das im XML-Schema-Repository registrierte Schema
  - Durch Angeben der Schemaposition mit dem Argument *schemaLocation*

Werden sowohl der XSR-Objektname als auch *schemaLocation* angegeben, wird der XSR-Objektname verwendet. Wird nichts angegeben, erfolgt keine Gültigkeitsprüfung. Es wird nur ein XML-Parsing ausgeführt und Fehler beim XML-Parsing werden dokumentiert.

1 Verwenden Sie die Schemaposition aus dem Wert des Attributs `xsi:schemaLocation` im Instanzdokument.

Für das Eingabedokument wird eine Prüfung auf der Basis von XML-Schemadokumenten durchgeführt, die zu einem früheren Zeitpunkt im XML-Schema-Repository registriert wurden.

Das Instanzdokument wird ohne Prüfung syntaktisch analysiert, wenn der Wert des Arguments *implicitValidation* 0 und die Werte der Argumente *rschema*, *name* und *schemaLocation* NULL lauten.

errorDialog
Ein Ausgabeargument des Typs VARCHAR(32000), das ein XML-Dokument im UTF-8-Format enthält, das die Parsing- und Gültigkeitsfehler auflistet. Das Dokument wird nur generiert, wenn mindestens ein Fehler vorliegt.

errorCount
Ein Ausgabeargument des Typs INTEGER, das die Gesamtzahl der XML-Parsing- und XML-Gültigkeitsfehler angibt.

Verwendung

Die Gültigkeitsprüfung für ein XML-Dokument kann an einem registrierten XML-Schema auf drei Arten ausgeführt werden:

- Durch Angeben des XSR-Objektnamens für das XML-Schema mit den Argumenten *rschema* und *name*
- Durch Angeben der Schemaposition mit dem Argument *schemaLocation*
- Durch Einstellen von *implicitValidation* auf 1, wenn das XML-Instanzdokument das Schema als Wert für das Attribut `xsi:schemaLocation` angibt

Wenn bei Verwendung der gespeicherten Prozedur XSR_GET_PARSING_DIAGNOSTICS ein Parsing- oder Gültigkeitsfehler festgestellt wird, werden die Ausgabeparameter *errorDialog* und *errorCount* gesetzt. Der Parameter *errorDialog* enthält ein XML-Dokument, das die Fehler auflistet. Sie können die gespeicherte Prozedur XSR_GET_PARSING_DIAGNOSTICS über Anwendungen aufrufen, die CLI, Java oder C++ verwenden, und die Ausgabe der gespeicherten Prozedur mithilfe einer Parametermarke abrufen.

Anzeige detaillierter XML-Parsing- und XML-Gültigkeitsfehler

Mithilfe der Angaben im Ausgabeparameter **`errorDialog`** der gespeicherten Prozedur XSR_GET_PARSING_DIAGNOSTICS können Sie Sie XML-Parsing- und XML-Gültigkeitsfehler beheben.

Wenn bei der Gültigkeitsprüfung eines XML-Dokuments mit der Skalarfunktion XMLVALIDATE Fehler auftreten, können Sie mit der gespeicherten Prozedur XSR_GET_PARSING_DIAGNOSTICS detaillierte Fehlerinformationen generieren. Im folgenden Beispiel wird die gespeicherte Prozedur XSR_GET_PARSING_DIAGNOSTICS gemeinsam mit einem einfachen XML-Schema und XML-Dokument dazu verwendet, detaillierte Angaben zu Gültigkeitsfehlern zu generieren.

Beispiel für ein XML-Schema

In dem Beispiel wird die folgende XML-Schemadefinition (XSD) verwendet. Das Schema enthält verschiedene Elemente, bei denen ein Attribut vom Typ 'integer' dem Element 'Age' zugeordnet ist.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://my.simpletest"
           xmlns="http://my.simpletest"
           elementFormDefault="qualified">
<xs:element name="Person">
 <xs:complexType>
   <xs:sequence>
     <xs:element name="Name">
      <xs:complexType>
       <xs:sequence>
        <xs:element name="FirstName" type="xs:string"/>
        <xs:element name="LastName" type="xs:string"/>
       </xs:sequence>
      </xs:complexType>
     </xs:element>
    <xs:element name="Age" type="xs:integer"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Beispiel für ein XML-Dokument

Das folgende XML-Dokument wird mit dem XML-Beispielschema geprüft. Das Dokument hält einige der Schemaregeln nicht ein. Das Element 'Age' ist nicht numerisch und das Element Notes ist im Schema nicht als Unterelement des Elements 'Person' definiert.

```
<?xml version="1.0"?>
<Person xmlns="http://my.simpletest"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://my.simpletest http://my.simpletest/simple">
  <Name>
   <FirstName>Thomas</FirstName>
   <LastName>Watson</LastName>
  </Name>
  <Age>30x</Age>
  <Notes/>
<Person>
```

Befehl zum Registrieren des XML-Schemas

Vor der Gültigkeitsprüfung eines XML-Dokuments mit der gespeicherten Prozedur XSR_GET_PARSING_DIAGNOSTICS muss das für die Prüfung verwendete XML-Schema im XML-Schema-Repository von DB2 registriert werden. Bei dem folgenden Befehl REGISTER XMLSCHEMA wird angenommen, dass das Schema sich im Pfad `c:\temp\simpleschema.xsd` befindet und das SQL-Schema den Namen 'USER1' hat:

```
REGISTER XMLSCHEMA 'http://my.simpletest/simple'
   FROM 'file:///c:/temp/simpleschema.xsd'
   AS user1.simple COMPLETE
```

Aufruf zum Generieren detaillierter Angaben zu Gültigkeitsfehlern

Der folgende Aufruf verwendet zum Generieren detaillierter Angaben zu Gültigkeitsfehlern die gespeicherte Prozedur XSR_GET_PARSING_DIAGNOSTICS über den Befehlszeilenprozessor:

```
CALL XSR_GET_PARSING_DIAGNOSTICS(
blob('<?xml version="1.0"?>
<Person xmlns="http://my.simpletest"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://my.simpletest http://my.simpletest/simple">
<Name>
  <FirstName>Thomas</FirstName>
  <LastName>Watson</LastName>
 </Name>
 <Age>30x</Age>
 <Notes/>
</Person>
'),'','','',1,?,?)@
```

Detaillierte Angaben zu Gültigkeitsfehlern

Für den Aufruf der gespeicherten Prozedur XSR_GET_PARSING_DIAGNOSTICS wird die folgende Ausgabe zurückgegeben. Der Wert des Parameters **errorDialog** ist ein XML-Dokument, in dem detaillierte Angaben zu Gültigkeitsfehlern enthalten sind. In dem XML-Dokument werden die konkreten Fehler und deren Position durch den Inhalt der Elemente errText, location, lineNum und colNum angezeigt. Wird der vorhergehende Aufruf über eine CLP-Befehlszeile ausgeführt, wird die folgende Ausgabe in eine Standardausgabe geschrieben:

```
Wert der Ausgabeparameter
-------------------------
Parametername: ERRORDIALOG
Parameterwert: <ErrorLog>
<XML_Error parser="XML4C">
  <errCode>238</errCode>
  <errDomain>http://apache.org/xml/messages/XML4CErrors</errDomain>
  <errText>Datatype error: Type:InvalidDatatypeValueException,
    Message:Value '30x' does not match regular expression facet '[+\-]?[0-9]+' .
</errText>
  <lineNum>1</lineNum>
  <colNum>272</colNum>
  <location>/Person/Age</location>
  <schemaType>http://www.w3.org/2001/XMLSchema:integer</schemaType>
  <tokenCount>2</tokenCount>
  <token1>30x</token1>
  <token2>13</token2>
</XML_Error>
<XML_Error parser="XML4C">
  <errCode>2</errCode>
  <errDomain>http://apache.org/xml/messages/XMLValidity</errDomain>
  <errText>Unknown element 'Notes' </errText>
  <lineNum>1</lineNum>
  <colNum>282</colNum>
  <location>/Person</location>
  <schemaType>http://www.w3.org/2001/XMLSchema:integer</schemaType>
  <tokenCount>2</tokenCount>
  <token1>Notes</token1>
  <token2>37</token2>
</XML_Error>
<XML_Error parser="XML4C">
  <errCode>7</errCode>
  <errDomain>http://apache.org/xml/messages/XMLValidity</errDomain>
  <errText>Element 'Notes' is not valid for content model: '(Name,Age)'
```

```
  </errText>
  <lineNum>1</lineNum>
  <colNum>292</colNum>
  <location>/Person</location>
  <schemaType>http://www.w3.org/2001/XMLSchema:anyType</schemaType>
  <tokenCount>2</tokenCount>
  <token1>Notes</token1>
  <token2>31</token2>
</XML_Error>
<DB2_Error>
  <sqlstate>2200M</sqlstate>
  <sqlcode>-16210</sqlcode>
  <errText>
    [IBM][CLI Driver][DB2/NT] SQL16210N  Das XML-Dokument enthielt einen Wert "30x",
    der gegen eine Fassetteneinschränkung verstößt. Ursachencode = "13".
    SQLSTATE=2200M
    </errText>
</DB2_Error>
</ErrorLog>

Parametername: ERRORCOUNT
Parameterwert: 3
Rückgabestatus = 0
```

XML-Schemadefinition 'ErrorLog' für erweiterte Fehlernachrichtenunterstützung

Die XML-Schemadefinition (XSD) 'ErrorLog' beschreibt das XML-Dokument im UTF-8-Format, das von der gespeicherten Prozedur XSR_GET_PARSING_DIAGNOSTICS aufgrund von XML-Parsing- und XML-Gültigkeitsfehlern generiert wurde. Das XML-Ausgabedokument wird im Argument *errorDialog* gespeichert.

ErrorLogType

Das Stammelement der XML-Schemadefinition heißt 'ErrorLog' und hat den Typ 'ErrorLogType'.

XML-Schemadefinition

```
<xs:complexType name="ErrorLogType">
  <xs:sequence>
    <xs:element name="XML_Error" type="XML_ErrorType" minOccurs="0"
          maxOccurs="unbounded"/>
    <xs:element name=name="XML_FatalError" type="XML_ErrorType" minOccurs="0"
          maxOccurs="unbounded"/>
    <xs:element name="DB2_Error"           type="DB2_ErrorType"/>
  </xs:sequence>
</xs:complexType>
```

Unterelemente

XML_Error
XML_FatalError

Typ: XML_ErrorType

Hinweise:

Ein Element 'XML_Error' oder 'XML_FatalError' enthält eine vom XML-Parser generierte Fehlernachricht. Das Element 'XML_Error' und das Element 'XML_FatalError' haben denselben XML-Schematyp. 'XML_FatalError' ist ein Fehler, der bewirkt, dass der XML-Parser den Parsingprozess abbricht.

```
xs:complexType name="XML_ErrorType">
 <xs:sequence>
   <xs:element name="errCode"    type="xs:int"/>
   <xs:element name="errDomain"  type="xs:string"/>
   <xs:element name="errText"    type="xs:string"/>
   <xs:element name="lineNum"    type="xs:unsignedInt"/>
   <xs:element name="colNum"     type="xs:unsignedInt"/>
   <xs:element name="location"   type="xs:string"/>
   <xs:element name="schemaType" type="xs:string"/>
   <xs:element name="tokens">
     <xs:complexType>
       <xs:sequence minOccurs="0">
         <xs:element name="token" type="xs:string" minOccurs="0"
              maxOccurs="unbounded"/>
       </xs:sequence>
       <xs:attribute name="count" type="xs:unsignedByte" use="required"/>
   </xs:complexType>
  </xs:element>
 </xs:sequence>
 <xs:attribute name="parser" type="xs:string" use="required"/>
</xs:complexType>
```

'XML_ErrorType' enthält die folgenden Unterelemente:

errCode
Ein vom XML-Parser zurückgegebener Fehlercode.

errDomain
Eine vom XML-Parser zurückgegebene Fehlerdomäne.

errText
Eine ursprünglich vom XML-Parser zurückgegebene Fehlernachricht.

lineNum
Die Nummer einer Zeile, in der der Fehler aufgetreten ist.

colNum
Die Nummer einer Spalte, in der der Fehler aufgetreten ist.

location
Das Element 'location' stellt einen XPath-Ausdruck dar, der auf das letzte XML-Element vor dem Auftritt des Fehlers verweist.

schemaType
Ein XML-Schematyp des letzten geparsten XML-Elements.

tokens
Ein numerischer Wert, der angibt, wie viele Token zurückgemeldet werden.

token Ein Token ist ein Zeichenfolgewert, der zur Generierung der DB2-Fehlernachricht verwendet wird.

Attribute

parser (required)
Das Attribut 'parser' gibt den zugrunde liegenden XML-Parser an, der verwendet wurde.

DB2_Error

Typ: DB2_ErrorType

Hinweise:
Ein Element 'DB2_Error' enthält die DB2-Fehlernachricht.

```
<xs:complexType name="DB2_ErrorType">
  <xs:sequence>
    <xs:element name="sqlstate" type="xs:string"/>
    <xs:element name="sqlcode"  type="xs:int"/>
    <xs:element name="errText"  type="xs:string"/>
  </xs:sequence>
 <xs:attribute name="parser" type="xs:string" use="required"/>
</xs:complexType>
```

'DB2_ErrorType' enthält die folgenden Unterelemente:

sqlstate
Ein SQLSTATE-Wert

sqlcode
Ein SQLCCODE-Wert

errText
Eine DB2-Fehlernachricht

XML-Schema für erweiterte Fehlernachrichtenunterstützung

Die gespeicherte Prozedur XSR_GET_PARSING_DIAGNOSTICS generiert detaillierte Informationen zu den bei der Syntaxanalyse und Gültigkeitsprüfung eines XML-Dokuments auftretenden Fehlern. Die Informationen werden in Form eines XML-Dokuments generiert. Das Schema definiert die gültige XML-Ausgabe der gespeicherten Prozedur.

Die folgende Liste ist eine Darstellung des XML-Schemas 'ErrorLog' für das mit der gespeicherten Prozedur XSR_GET_PARSING_DIAGNOSTICS generierte XML-Dokument.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns="http://www.ibm.com/db2/XMLParser/Diagnosticsv10"
     xmlns:xs="http://www.w3.org/2001/XMLSchema"
     targetNamespace="http://www.ibm.com/db2/XMLParser/Diagnosticsv10"
     elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:element name="ErrorLog">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="XML_Error" type="XML_ErrorType" minOccurs="0"
             maxOccurs="unbounded"/>
        <xs:element name="XML_FatalError" type="XML_ErrorType" minOccurs="0"/>
        <xs:element name="DB2_Error"           type="DB2_ErrorType"/>
        <xs:any namespace="##any" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="XML_ErrorType">
   <xs:attribute name="parser" type="xs:string" use="required"/>
    <xs:sequence>
      <xs:element name="errCode"    type="xs:int"/>
      <xs:element name="errDomain"  type="xs:string"/>
      <xs:element name="errText"  type="xs:string"/>
      <xs:element name="lineNum"    type="xs:unsignedInt"/>
      <xs:element name="colNum"     type="xs:unsignedInt"/>
      <xs:element name="location"   type="xs:string"/>
      <xs:element name="schemaType" type="xs:string"/>
      <xs:element name="tokens">
        <xs:complexType>
          <xs:sequence minOccurs="0">
            <xs:element name="token" type="xs:string" minOccurs="0"
                 maxOccurs="unbounded"/>
          </xs:sequence>
          <xs:attribute name="count" type="xs:unsignedByte" use="required"/>
        </xs:complexType>
      </xs:element>
```

```
      <xs:any namespace="##any"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="DB2_ErrorType">
   <xs:attribute name="parser" type="xs:string" use="required"/>
    <xs:sequence>
      <xs:element name="sqlstate" type="xs:string"/>
      <xs:element name="sqlcode"  type="xs:int"/>
      <xs:element name="errText"  type="xs:string"/>
      <xs:any namespace="##any"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

XML in Nicht-Unicode-Datenbanken verwenden

Ab Version 9.5 können XML-Daten in Datenbanken, die nicht die Unicode-Codepage verwenden, gespeichert und aus diesen abgerufen werden.

Intern werden XML-Daten unabhängig von der Datenbankcodepage stets vom DB2-Datenbankserver in einem Unicode-Format verwaltet. Relationale nicht-XML-Daten werden in der Datenbankcodepage verwaltet. Wenn SQL- oder XQuery-Anweisungen sowohl XML-Daten als auch relationale SQL-Daten umfassen, was beispielsweise bei der Umsetzung eines Datentyps in einen anderen Datentyp oder bei Vergleichen, die sowohl den XML-Datentyp als auch SQL-Datentypen umfassen, der Fall sein kann, ist häufig eine Codepagekonvertierung erforderlich. Bei Vergleichen zwischen XML-Daten ist keine Codepagekonvertierung erforderlich, da beide Datengruppen bereits im UTF-8-Format vorliegen. Ebenso ist bei Vergleichen zwischen SQL-Daten keine Codepagekonvertierung erforderlich, da beide Datengruppen bereits im Format der Datenbankcodepage vorliegen.

Bei Operationen, die XML- und SQL-Daten umfassen, ist in Unicode-Datenbanken keine Codepagekonvertierung erforderlich, da diese Datenbanken für alle Datentypen dieselbe Codierung verwenden. In Nicht-Unicode-Datenbanken können Operationen, die eine Codepagekonvertierung umfassen, jedoch möglicherweise zu einer Beschädigung von Daten oder zu Datenverlust führen. Wenn die XML-Daten, die konvertiert werden, Zeichen mit Codepunkten enthalten, die nicht zur Datenbankcodepage gehören, werden die betreffenden Zeichen ersetzt. Daher können Umsetzungs- oder Vergleichsoperationen unerwartete Ergebnisse liefern, und XML-Daten, die aus der Datenbank abgerufen werden, enthalten unter Umständen falsche Werte. Im folgenden Abschnitt werden verschiedene Methoden erläutert, die verwendet werden können, um Probleme bei der Codepagekonvertierung zu vermeiden und so die Integrität von gespeicherten XML-Daten und der sie betreffenden Operationen sicherzustellen.

Einfügen von XML-Dokumenten und Codepagekonvertierung

Sobald XML-Daten über eine Hostvariable oder eine Parametermarke mit einem Zeichendatentyp (Datentyp CHAR, VARCHAR oder CLOB ohne den Typ FOR BIT DATA) in einen DB2-Datenbankserver eingefügt werden, findet eine Codepagekonvertierung statt, wenn sich die Datenbankcodepage von der Codepage des Clients oder der Anwendung, der bzw. die die Anforderung absetzt, unterscheidet. Eine zweite Konvertierung findet statt, wenn die eingefügten Zeichendaten aus der Datenbankcodepage in Unicode (das Format, in dem XML-Daten intern verwaltet werden) umgewandelt werden.

Die nachstehende Tabelle zeigt die verschiedenen zulässigen Codierungskombinationen zwischen einer Datenbank und einer von einem Client oder einer Anwen-

dung eingefügten XML-Dokumentzeichenfolge. Die Codierung des XML-Dokuments entspricht der Client-Codepage, da der Client die XML-Daten über einen Zeichendatentyp einfügt. Für jede Kombination werden die Auswirkungen auf die Codepagekonvertierung und die Möglichkeit von Zeichenersetzungen beim Einfügen von XML-Dokumenten beschrieben.

Tabelle 2. Codierungsszenarios beim Einfügen einer XML-Dokumentzeichenfolge in eine Datenbank

Szenario	XML-Dokumentcodierung	Datenbankcodierung	Stimmen die Codepages überein?
1.	Unicode (UTF-8)	Unicode (UTF-8)	Ja
2.	Nicht-Unicode	Unicode (UTF-8)	Nein
3.	Nicht-Unicode	Nicht-Unicode	Ja
4.	Unicode (UTF-8)	Nicht-Unicode	Nein
5.	Nicht-Unicode	Nicht-Unicode	Nein

1. Im 1. Szenario weisen ein XML-Dokument und eine Datenbank beide eine Unicode-Codierung auf. Beim Einfügen des XML-Dokuments findet keine Zeichenkonvertierung statt. Das Einfügen von XML-Daten auf diese Weise ist stets problemlos.
2. Im 2. Szenario wird ein XML-Dokument, das nicht in Unicode vorliegt, beim Einfügen in eine Unicode-Datenbank in UTF-8 konvertiert. Bei diesem Konvertierungsprozess werden keine Zeichen ersetzt. Das Einfügen von XML-Daten auf diese Weise ist stets problemlos.
3. Im 3. Szenario weisen ein XML-Dokument und eine Datenbank beide dieselbe Nicht-Unicode-Codierung auf. In diesem Fall kann das XML-Dokument lediglich Codepunkte enthalten, die zur Datenbankcodepage gehören, sodass bei der Codepagekonvertierung keine Zeichen ersetzt werden. Das Einfügen von XML-Daten auf diese Weise ist stets problemlos.
4. Im 4. Szenario wird ein Unicode-XML-Dokument in eine Nicht-Unicode-Datenbank eingefügt. Es findet eine Codepagekonvertierung statt, wenn das XML-Dokument von einem UTF-8-Client oder einer UTF-8-Anwendung über eine Hostvariable oder eine Parametermarke mit Zeichendatentyp eingefügt wird. Alle Zeichen im XML-Dokument ohne entsprechende Codepunkte in der Datenbankcodepage werden ersetzt.
5. Im 5. Szenario wird ein XML-Dokument in einen Datenbankserver eingefügt, wobei das Dokument und der Server unterschiedliche Codierungen aufweisen, bei denen es sich in beiden Fällen nicht um UTF-8 handelt. In diesem Szenario werden - ebenso wie in Szenario 4 - Zeichen ersetzt, wenn das XML-Dokument mit einem Zeichendatentyp eingefügt wird und Zeichen enthält, die in der Datenbankcodepage ungültig sind.

XML-Daten ohne Probleme in eine Nicht-Unicode-Datenbank einfügen

Am besten lässt sich die Integrität von XML-Daten durch Verwendung einer Unicode-Datenbank sicherstellen. Ist dies nicht möglich, stehen jedoch auch andere Möglichkeiten zur Verfügung, um zu verhindern, das Zeichen ersetzt werden. In der folgenden Liste werden verschiedene Methoden beschrieben, die verwendet werden können, um XML-Daten problemlos sowohl in Unicode- als auch in Nicht-Unicode-Datenbanken einzufügen:

Verwenden Sie eine Unicode-Datenbank oder stellen Sie sicher, dass die Datenbank und der Client dieselbe Codierung verwenden.

Wie aus Tabelle 2 auf Seite 67 ersichtlich wird, lassen sich Probleme bei der Codepagekonvertierung für XML-Daten stets vermeiden, wenn eine der folgenden Bedingungen erfüllt ist:

- Die Datenbank liegt im Unicode-Format vor.
- Die Datenbank und der Client weisen dieselbe Codierung auf, unabhängig davon, ob es sich um Unicode handelt oder nicht.

Vermeiden Sie die Verwendung einer Hostvariablen oder Parametermarke mit einem Zeichendatentyp.

Kann keine Unicode-Datenbank verwendet werden, lässt sich eine Codepagekonvertierung für XML-Daten auch dadurch vermeiden, dass die XML-Daten mithilfe einer Hostvariablen oder Parametermarke vom Typ XML oder einem beliebigen binären Datentyp gebunden werden. Wird also für die XML-Daten ein anderer Datentyp als CHAR, VARCHAR oder CLOB angegeben, können die XML-Daten direkt von der Client- oder Anwendungscodepage in Unicode konvertiert werden, ohne dass zuvor eine Konvertierung in die Datenbankcodepage erforderlich ist.

Mithilfe des Konfigurationsparameters ENABLE_XMLCHAR können Sie steuern, ob Einfügungen mithilfe von Zeichendatentypen zulässig sind oder nicht. Wenn Sie ENABLE_XMLCHAR auf "NO" setzen, wird die Verwendung von Zeichendatentypen beim Einfügen von XML-Dokumenten blockiert und somit verhindert, dass möglicherweise Zeichen ersetzt werden, wodurch die Integrität der gespeicherten XML-Daten gewahrt bleibt. Die Datentypen BLOB und FOR BIT DATA sind weiterhin zulässig, da bei diesen Datentypen keine Codepagekonvertierung stattfindet. Standardmäßig ist der Konfigurationsparameter ENABLE_XMLCHAR auf "YES" gesetzt, sodass das Einfügen von Zeichendatentypen zulässig ist.

Bei Verwendung einer Unicode-Datenbank ist eine Codepagekonvertierung stets problemlos, sodass der Konfigurationsparameters ENABLE_XMLCHAR in diesem Fall keine Rolle spielt und Zeichendatentypen beim Einfügen von XML-Dokumenten unabhängig von der Einstellung dieses Parameters verwendet werden können.

Verwenden Sie Zeichenentitätsverweise für Zeichen, die nicht zur Datenbankcodepage gehören.

Lässt sich eine Codepagekonvertierung nicht vermeiden und muss für den XML-Datenstrom ein Zeichendatentyp verwendet werden, sollte sichergestellt werden, dass alle Zeichen im XML-Dokument über entsprechende Codepunkte in der Datenbankcodepage verfügen. Für Zeichen in den XML-Daten, die keine entsprechenden Codepunkte in der Zieldatenbank aufweisen, können Sie einen Zeichenentitätsverweis verwenden, um den jeweiligen Unicode-Codepunkt des betreffenden Zeichens anzugeben. Bei der Verwendung von Zeichenentitätsverweisen wird die Codepagekonvertierung stets umgangen, sodass in den XML-Daten das korrekte Zeichen beibehalten wird. Der Zeichenentitätsverweis > bzw. > beispielsweise ist die hexadezimale bzw. dezimale Entsprechung des Größer-als-Zeichens (>").

XML-Daten in einer Nicht-Unicode-Datenbank abfragen

Ebenso wie beim Einfügen von XML-Daten in eine Datenbank lässt sich die Datenintegrität auch beim Abfragen von XML-Daten am besten durch Verwendung einer Unicode-Datenbank sicherstellen. Ist dies nicht möglich, können Sie das Erset-

zen von Zeichen vermeiden, indem Sie sicherstellen, dass alle XML-Daten in der Datenbankcodepage dargestellt werden können, oder indem Sie Zeichenentitätsverweise für Zeichen verwenden, die nicht zur Datenbankcodepage gehören.

Falls eine Abfrage XML-Daten mit Zeichen enthält, die in der Datenbankcodepage nicht dargestellt werden können, können die folgenden beiden Arten von Zeichenersetzungen auftreten, die unter Umständen zu unerwarteten Abfrageergebnissen führen:

Ersetzung durch das standardmäßige Substitutionszeichen
Ein Zeichen in den XML-Daten, das keine Entsprechung aufweist, wird durch das jeweilige standardmäßige Substitutionszeichen für die Codepage ersetzt. Beispiel: Wird ein chinesisches Zeichen an eine mit ASCII codierte Datenbank (ISO-8859-1) übergeben, wird das ursprüngliche Zeichen durch den ASCII-Codepunkt 0x1A ersetzt, bei dem es sich um ein Steuerzeichen handelt, das auf einem Client üblicherweise als Fragezeichen ('?) angezeigt wird. Werden die XML-Daten aus der Datenbankcodepage in Unicode konvertiert, wird das Substitutionszeichen beibehalten.

Ersetzung durch das Zeichen mit der nächstbesten Entsprechung ("Umsetzung")
Das ursprüngliche Eingabezeichen wird durch ein Zeichen in der Zielcodepage ersetzt, das dem ursprünglichen Zeichen ähnlich ist, mit diesem aber nicht unbedingt identisch sein muss. In einige Fällen werden zwei oder mehr Zeichen mit unterschiedlichen Unicode-Codepunkten einem einzigen Codepunkt in einer Datenbankcodepage (der nächstbesten Zeichenentsprechung in der Zielcodepage) zugeordnet, sodass nach dem Einfügen in die Datenbank zwischen den Werten kein Unterschied mehr besteht. Dieses Szenario wird in Beispiel 2 veranschaulicht.

Beispiele

Die folgenden Beispiele zeigen die möglichen Auswirkungen einer Codepagekonvertierung, wenn ein Client oder eine Anwendung mit einer UTF-8-Codierung zum Abfragen von XML-Daten in einer Nicht-Unicode-Datenbank verwendet wird. Bei diesen Beispielen wird davon ausgegangen, dass die Datenbank unter Verwendung der Codepage ISO8859-7 (Griechisch) erstellt wurde. Es werden XQuery-Ausdrücke verwendet, um XML-Daten abzugleichen, die in Tabelle T1 gespeichert sind und die aus den griechischen Unicode-Sigmazeichen (Σ_G) und dem mathematischen Unicode-Sigmazeichen (Σ_M) bestehen. Der Codepunkt 0xD3 identifiziert das Sigmazeichen in der ISO8859-7-Datenbank.

Die Tabelle T1 wird mithilfe der folgenden Befehle erstellt und aufgefüllt:

```
CREATE TABLE T1 (DOCID BIGINT NOT NULL, XMLCOL XML);
INSERT INTO T1 VALUES (1, XMLPARSE(
       document '<?xml version="1.0" encoding="utf-8" ?> <Specialchars>
 <sigma>Σ_G</sigma>
 <summation>Σ_M</summation>
 </Specialchars>'
       preserve whitespace));
```

Beispiel 1: Erfolgreiche Codepagekonvertierung (das Zeichen kann in der Datenbankcodepage dargestellt werden)

```
XQUERY for $test in db2-fn:xmlcolumn("T1.XMLCOL")//*[. = "Σ_G"] return $test
```

Dieser Ausdruck führt zum gewünschten Ergebnis:

```
<sigma>Σ_G</sigma>
```

In diesem Fall beginnt der Ausdruck Σ_G auf der Clientseite als Unicode-Codepunkt für das griechische Sigmazeichen (U+03A3), wird in das Sigmazeichen in der griechischen Datenbankcodepage konvertiert (0xD3) und anschließend in das korrekte Unicode-Zeichen für die XML-Verarbeitung zurück konvertiert. Da das griechische Sigmazeichen in der Datenbankcodepage dargestellt werden kann, wird der Ausdruck korrekt abgeglichen. Diese Zeichenkonvertierung wird in folgender Tabelle dargestellt:

Tabelle 3. Konvertierung von Zeichendaten (Beispiel 1)

	Client (UTF-8)		Datenbank (ISO8859-7)		XML-Parser (UTF-8)
Zeichen	U+03A3 (Griechisches Sigmazeichen)	→	0xD3 (Griechisches Sigmazeichen)	→	U+03A3 (Griechisches Sigmazeichen)

Beispiel 2: Nicht erfolgreiche Codepagekonvertierung (das Zeichen kann in der Datenbankcodepage nicht dargestellt werden)

```
XQUERY for $test in db2-fn:xmlcolumn("T1.XMLCOL")//*[. = "Σ_M"] return $test
```

Dieser Ausdruck führt nicht zum gewünschten Ergebnis:

```
<sigma>Σ_G</sigma>
```

In diesem Fall beginnt der Ausdruck Σ_M auf der Clientseite als Unicode-Codepunkt für das mathematische Sigmazeichen (U+2211), wird in das Sigmazeichen in der griechischen Datenbankcodepage konvertiert (0xD3) und wird beim XML-Vergleich anschließend mit dem Zeichen Σ_G abgeglichen. Im Hinblick auf den zurückgegebenen Ausdruck ist die Verarbeitung mit der in Beispiel 1 identisch: Das Unicode-XML-Zeichen Σ_M wird zunächst in das Sigmazeichen in der griechischen Datenbankcodepage (Σ_A) konvertiert und anschließend in das griechische Sigmazeichen in der UTF-8-Codepage des Clients (Σ_G) zurück konvertiert. Diese Zeichenkonvertierung wird in folgender Tabelle dargestellt:

Tabelle 4. Konvertierung von Zeichendaten (Beispiel 2)

	Client (UTF-8)		Datenbank (ISO8859-7)		XML-Parser (UTF-8)
Zeichen	U+2211 (Mathematisches Sigmazeichen)	→	0xD3 (Griechisches Sigmazeichen)	→	U+03A3 (Griechisches Sigmazeichen)

Beispiel 3: Umgehung der Codepagekonvertierung durch Verwendung eines Zeichenentitätsverweises

```
XQUERY for $test in db2-fn:xmlcolumn("T1.XMLCOL")//*[. = "&#2211;"]
   return $test
```

Dieser Ausdruck führt zum gewünschten Ergebnis:

```
<summation>Σ_M</summation>
```

In diesem Fall beginnt der Ausdruck Σ_M auf der Clientseite als Unicode-Codepunkt für das mathematische Sigmazeichen (U+2211). Da für den Codepunkt ein Zeichenverweis (&#2211) angegeben wird, bleibt der Codepunkt bei der Übergabe an den XML-Parser erhalten, wodurch ein erfolgreicher Abgleich mit dem gespeicherten XML-Wert (Σ_M) möglich ist. Die Umgehung der Zeichenkonvertierung wird in folgender Tabelle dargestellt:

Tabelle 5. Konvertierung von Zeichendaten (Beispiel 3)

	Client (UTF-8)		Datenbank (ISO8859-7)		XML-Parser (UTF-8)
Zeichen	U+2211 (mathematisches Sigmazeichen)	→	"&#2211" (Zeichenverweis für mathematisches Sigmazeichen)	→	U+2211 (mathematisches Sigmazeichen)

Beispiel 4: Nicht erfolgreiche Codepagekonvertierung (das Zeichen kann in der Datenbankcodepage nicht dargestellt werden)

Dieses Beispiel ähnelt Beispiel 1, mit der Ausnahme, dass in diesem Fall eine mit ASCII codierte Datenbank verwendet wird und das standardmäßige Substitutionszeichen für die Codepage in den XML-Ausdruck eingefügt wird:

```
XQUERY for $test in db2-fn:xmlcolumn("T1.XMLCOL")//*[. = "Σ_G"] return $test
```

Bei dieser Abfrage erfolgt der Abgleich nicht mit dem korrekten Wert in Tabelle T1. In diesem Fall verfügt das Unicode-Zeichen U+2211 (griechisches Sigmazeichen) nicht über einen entsprechenden Codepunkt in der ASCII-Codepage, sodass ein standardmäßiges Substitutionszeichen eingefügt wird (hier das Fragezeichen ('?')). Diese Zeichenkonvertierung wird in folgender Tabelle dargestellt:

Tabelle 6. Konvertierung von Zeichendaten (Beispiel 4)

	Client (UTF-8)		Datenbank (ISO8859-1)		XML-Parser (UTF-8)
Zeichen	U+2211 (mathematisches Sigmazeichen)	→	0x003F ('?')	→	0x003F ('?')

Kapitel 5. Abfragen von XML-Daten

XML-Daten in der Datenbank können mithilfe von zwei Hauptabfragesprachen abgefragt bzw. abgerufen werden. Hierbei ist es möglich, entweder jede Sprache einzeln oder eine Kombination aus beiden Sprachen zu verwenden.

Die folgenden Optionen sind verfügbar:

- Nur XQuery-Ausdrücke
- XQuery-Ausdrücke, die SQL-Anweisungen aufrufen
- Nur SQL-Anweisungen
- SQL-Anweisungen, die XQuery-Ausdrücke ausführen

Diese verschiedenen Methoden ermöglichen das Abfragen bzw. Abrufen von XML-Daten sowie von anderen relationalen Daten aus einem SQL- oder XQuery-Kontext heraus.

Mit diesen Methoden können XML-Dokumente teilweise oder vollständig abgefragt und abgerufen werden. Abfragen können Fragmente oder vollständige XML-Dokumente zurückgeben. Abfrageergebnisse können durch Vergleichselemente eingegrenzt werden. Da Abfragen auf XML-Daten XML-Sequenzen zurückgeben, kann auch das Ergebnis einer Abfrage in der Konstruktion von XML-Daten verwendet werden.

Einführung zu XQuery

Bei XQuery handelt es sich um eine funktionsorientierte Programmiersprache, die von W3C (World Wide Web Consortium) entwickelt wurde, um die speziellen Anforderungen zu erfüllen, die beim Abfragen und Ändern von XML-Daten gelten.

Anders als bei relationalen Daten, die vorhersehbar sind und über eine reguläre Struktur verfügen, sind XML-Daten sehr variabel strukturiert. XML-Daten können häufig nicht genau vorhergesehen werden, enthalten nur wenige Informationen und sind selbstbeschreibend.

Da die Struktur von XML-Daten nicht genau vorhersehbar ist, unterscheiden sich auch die Abfragen, die für XML-Daten ausgeführt werden müssen, häufig von normalen Abfragen für relationale Daten. Die XQuery-Sprache bietet die Flexibilität, die zur Ausführung dieser Operationen erforderlich ist. So kann es beispielsweise erforderlich sein, dass Sie die Sprache XQuery zur Ausführung der folgenden Operationen verwenden müssen:

- Durchsuchen von XML-Daten nach Objekten, die sich innerhalb der Hierarchie auf einer nicht bekannten Ebene befinden.
- Ausführen struktureller Konvertierungen für die Daten (z. B. zur Umkehrung einer Hierarchie).
- Zurückgeben von Ergebnissen mit gemischten Typen.
- Aktualisieren vorhandener XML-Daten.

Komponenten einer XQuery-Abfrage

In XQuery werden Abfragen auf der Basis von Ausdrücken erstellt. Diese Ausdrücke können verschachtelt sein und bilden den Hauptteil einer Abfrage. Eine Abfra-

ge kann außerdem über einen Prolog verfügen, der diesem Hauptteil vorangestellt ist. Der *Prolog* enthält eine Reihe von Deklarationen, mit denen die Verarbeitungsumgebung der Abfrage definiert wird. Der *Abfragehauptteil* besteht aus einem Ausdruck, der die Ergebnisse der Abfrage definiert. Dieser Ausdruck kann aus mehreren XQuery-Ausdrücken zusammengesetzt sein, die mithilfe von Operatoren oder Schlüsselwörtern kombiniert werden.

Abb. 4 zeigt die Struktur einer typischen Abfrage. In diesem Beispiel enthält der Prolog zwei Deklarationen: Eine Versionsdeklaration (version), in der die Version der XQuery-Syntax angegeben ist, die zur Verarbeitung der Abfrage verwendet werden soll, und eine Standarddeklaration für den Namensbereich (namespace), in der die Namensbereichs-URI angegeben ist, die für Element- und Typnamen ohne Präfix verwendet werden kann. Der Abfragehauptteil enthält einen Ausdruck, mit dem ein Element `price_list` erstellt werden kann. Der Inhalt des Elements `price_list` umfasst eine Liste mit `product`-Elementen, die in absteigender Reihenfolge nach Preis (price) sortiert sind.

Abbildung 4. Struktur einer typischen XQuery-Abfrage

Abrufen von DB2-Daten mit XQuery-Funktionen

In XQuery kann eine Abfrage eine der folgenden Funktionen aufrufen, um XML-Eingabedaten aus einer DB2-Datenbank abzurufen: 'db2-fn:sqlquery' und 'db2-fn:xmlcolumn'.

Die Funktion 'db2-fn:xmlcolumn' ruft eine vollständige XML-Spalte ab, während die Funktion 'db2-fn:sqlquery' XML-Werte abruft, die auf einer SQL-Fullselect-Operation beruhen.

db2-fn:xmlcolumn

Die Funktion db2-fn:xmlcolumn verwendet ein Zeichenfolgeliteralargument, in dem eine XML-Spalte in einer Tabelle oder einer Sicht angegeben ist, und gibt eine Sequenz von XML-Werten zurück, die sich in dieser Spalte befinden. Beim Argument dieser Funktion muss die Groß-/Kleinschreibung beachtet werden. Das Zeichenfolgeliteralargument muss einen qualifizierten Spaltennamen des Typs XML angeben. Diese Funktion ermöglicht Ihnen das Extrahieren einer vollständigen Spalte von XML-Daten, ohne dass hierzu eine Suchbedingung angewendet werden muss.

Im folgenden Beispiel verwendet die Abfrage die Funktion db2-fn:xmlcolumn, um alle Bestellungen in der Spalte PURCHASE_ORDER der Tabelle BUSINESS.ORDERS abzurufen. Die Abfrage verwendet dann diese Eingabedaten, um die Städte aus der Versandadresse dieser Bestellungen zu extrahieren. Das Ergebnis der Abfrage ist eine Liste aller Städte, in die Bestellungen verschickt wurden.

```
db2-fn:xmlcolumn('BUSINESS.ORDERS.PURCHASE_ORDER')/shipping_address/city
```

db2-fn:sqlquery

Die Funktion db2-fn:sqlquery verwendet ein Zeichenfolgeargument, das eine Fullselect-Operation darstellt, und gibt eine XML-Sequenz zurück, die eine Verknüpfung der XML-Werte darstellt, die von der Fullselect-Operation zurückgegeben werden. Die Fullselect-Operation muss eine aus einer Zeile bestehende Ergebnismenge angeben, wobei diese Spalte den Datentyp XML aufweisen muss. Durch die Angabe einer Fullselect-Operation können Sie die Leistungsfähigkeit von SQL zur Bereitstellung von XML-Daten für XQuery nutzen. Die Funktion unterstützt die Verwendung von Parametern zum Übergeben von Werten an die SQL-Anweisung.

Im folgenden Beispiel enthält die Tabelle BUSINESS.ORDERS eine XML-Spalte mit dem Namen PURCHASE_ORDER. Die Abfrage in diesem Beispiel verwendet die Funktion db2-fn:sqlquery, um SQL aufzurufen und alle Bestellungen abzurufen, die das Versanddatum 15. Juni 2005 aufweisen. Die Abfrage verwendet dann diese Eingabedaten, um die Städte aus den Versandadressen dieser Bestellungen zu extrahieren. Das Ergebnis der Abfrage ist eine Liste aller Städte, in die am 15. Juni Bestellungen verschickt wurden.

```
db2-fn:sqlquery("
SELECT purchase_order FROM business.orders
WHERE ship_date = '2005-06-15' ")/shipping_address/city
```

Wichtig: Eine XML-Sequenz, die von der Funktion db2-fn:sqlquery oder db2-fn:xmlcolumn zurückgegeben wird, kann alle XML-Werte einschließlich atomarer Werte und Knoten enthalten. Diese Funktionen geben nicht immer eine Sequenz von korrekt formatierten Dokumenten zurück. Die Funktion kann z. B. einen einzelnen atomaren Wert (z. B. 36) als Instanz des Datentyps XML zurückgeben.

SQL und XQuery verwenden bei der Beachtung der Groß-/Kleinschreibung unterschiedliche Konventionen. Sie müssen diese Unterschiede berücksichtigen, wenn Sie die Funktionen db2-fn:sqlquery und db2-fn:xmlcolumn verwenden.

Bei SQL muss die Groß-/Kleinschreibung nicht beachtet werden.

Standardmäßig werden alle Standardbezeichner, die in SQL-Anweisungen verwendet werden, automatisch in Großschreibung umgesetzt. Aus diesem Grund werden die Namen von SQL-Tabellen und -Spalten normalerweise in Großbuchstaben angezeigt. In den obigen Beispielen trifft dies z. B. für BUSINESS.ORDERS und PURCHASE_ORDER zu. In einer SQL-Anweisung können Sie mit Kleinbuchstaben auf diese Spalten verweisen und z. B. business.orders und purchase_order verwenden, da diese Angaben automatisch während der Verarbeitung der SQL-Anweisung in Großbuchstaben umgesetzt werden. (Sie können auch einen Namen erstellen, bei dem die Groß-/Kleinschreibung beachtet werden muss und der in SQL als *begrenzter Bezeichner* bezeichnet wird. Setzen Sie hierzu den Namen in doppelte Anführungszeichen.)

Bei XQuery muss die Groß-/Kleinschreibung beachtet werden.

Namen in Kleinbuchstaben werden von XQuery nicht in Großschreibung umgesetzt. Dieser Unterschied kann bei der gemeinsamen Verwendung

von XQuery und SQL zu Problemen führen. Die Zeichenfolge, die an db2-fn:sqlquery übergeben wird, wird als SQL-Abfrage interpretiert und vom SQL-Parser syntaktisch analysiert. Hierbei werden alle Namen in Großbuchstaben umgesetzt. Im Beispiel mit db2-fn:sqlquery können der Tabellenname business.orders und die Spaltennamen purchase_order und ship-_date sowohl in Groß- als auch in Kleinschreibung angegeben werden. Der Operand von db2-fn:xmlcolumn ist allerdings keine SQL-Abfrage. Hierbei handelt es sich vielmehr um ein XQuery-Zeichenfolgeliteral, bei dem die Groß-/Kleinschreibung beachtet werden muss und das für den Namen einer Spalte steht. Da der tatsächliche Name der Spalte BUSINESS.ORDERS.PURCHASE_ORDER lautet, muss dieser Name im Operanden von db2-fn:xmlcolumn in Großbuchstaben angegeben werden.

Einführung in die XML-Datenabfrage mithilfe von SQL

XML-Daten können mithilfe einer SQL-Fullselect-Anweisung oder mit den SQL/XML-Abfragefunktionen XMLQUERY und XMLTABLE abgefragt werden. Das XMLEXISTS-Vergleichselement kann in SQL-Abfragen für XML-Daten verwendet werden.

Eine reine SQL-Abfrage für XML-Daten, d. h. ohne XQuery, kann nur eine Abfrage mit einer Fullselect-Anweisung auf Spaltenebene sein. Aus diesem Grund können nur vollständige XML-Dokumente durch eine solche Abfrage zurückgegeben werden. Reines SQL kann keine Fragmente eines Dokuments zurückgeben.

Für Abfragen auf Daten innerhalb von XML-Dokumenten müssen Sie XQuery verwenden. XQuery kann aus SQL heraus mit einer der folgenden SQL/XML-Funktionen bzw. mit dem folgenden Vergleichselement aufgerufen werden:

XMLQUERY
Eine SQL-Skalarfunktion, die das Ergebnis eines XQuery-Ausdrucks in Form einer XML-Sequenz zurückgibt.

XMLTABLE
Eine SQL-Tabellenfunktion, die das Ergebnis eines XQuery-Ausdrucks in Form einer Tabelle zurückgibt.

XMLEXISTS
Ein SQL-Vergleichselement, das bestimmt, ob ein XQuery-Ausdruck eine nicht leere Sequenz zurückgibt.

Vergleich zwischen XQuery und SQL

DB2-Datenbanken unterstützen das Speichern korrekt formatierter XML-Daten in einer Spalte einer Tabelle und das Abrufen der XML-Daten aus der Datenbank mithilfe von SQL und/oder XQuery. Beide Sprachen werden als primäre Abfragesprachen unterstützt, und beide Sprachen bieten Funktionen zum Aufrufen der jeweils anderen Sprache.

XQuery
Eine Abfrage, die XQuery direkt aufruft, beginnt mit dem Schlüsselwort XQUERY. Dieses Schlüsselwort gibt an, dass XQuery verwendet wird und dass der DB2-Server aus diesem Grund die Regeln zur Beachtung der Groß-/Kleinschreibung verwenden muss, die für die XQuery-Sprache gelten. Die Fehlerbehandlung basiert auf den Schnittstellen, die zur Verarbeitung der XQuery-Ausdrücke verwendet werden. XQuery-Fehler werden in derselben Weise wie SQL-Fehler mit einem SQLCODE-Wert und einem SQLSTATE-Wert gemeldet. Bei der Verarbeitung von XQuery-Ausdrücken

werden keine Warnungen ausgegeben. XQuery fordert Daten durch Aufruf von Funktionen an, mit denen XML-Daten aus DB2-Tabellen und -Sichten extrahiert werden. XQuery kann darüber hinaus auch über eine SQL-Abfrage aufgerufen werden. In diesem Fall kann die SQL-Abfrage XML-Daten in Form gebundener Variablen an XQuery übergeben. XQuery unterstützt verschiedene Ausdrücke für die Verarbeitung von XML-Daten und zum Erstellen neuer XML-Objekte wie z. B. Elemente und Attribute. Die Programmierschnittstelle für XQuery bietet Funktionen, die Ähnlichkeiten mit den entsprechenden SQL-Funktionen aufweisen und zur Vorbereitung von Abfragen und zum Abrufen von Abfrageergebnissen benutzt werden können.

SQL SQL bietet Funktionen, mit denen Werte des Datentyps XML definiert und instanziiert werden können. Zeichenfolgen, die korrekt formatierte XML-Dokumente enthalten, können über eine Syntaxanalyse in XML-Werte umgesetzt werden, optional auf der Basis eines XML-Schemas auf Ihre Gültigkeit überprüft und in Tabellen eingefügt oder in diesen aktualisiert werden. Alternativ hierzu können XML-Werte auch mithilfe einer SQL-Konstruktorfunktion erstellt werden, mit der andere relationale Daten in XML-Werte umgesetzt werden. Das System bietet außerdem Funktionen zum Abfragen von XML-Daten mithilfe von XQuery und zum Konvertieren von XML-Daten in eine relationale Tabelle, die dann in einer SQL-Abfrage verwendet werden kann. Zusätzlich zur Serialisierung von XML-Werten in Zeichenfolgewerte können die Daten zwischen den Datentypen von SQL und XML umgesetzt werden.

SQL/XML bietet die folgenden Funktionen und Vergleichselemente für das Aufrufen von XQuery über SQL:

XMLQUERY
Bei XMLQUERY handelt es sich um eine Skalarfunktion, die einen XQuery-Ausdruck als Argument verwendet und eine XML-Sequenz zurückgibt. Die Funktion umfasst optionale Parameter, die zum Übergeben von SQL-Werten als XQuery-Variablen an den XQuery-Ausdruck verwendet werden können. Die XML-Werte, die von XMLQUERY zurückgegeben werden, können im Kontext der SQL-Abfrage weiter verarbeitet werden.

XMLTABLE
Bei XMLTABLE handelt es sich um eine Tabellenfunktion, die XQuery-Ausdrücke zum Generieren einer SQL-Tabelle auf der Basis von XML-Daten verwendet, die mit SQL weiter verarbeitet werden können.

XMLEXISTS
XMLEXISTS ist ein SQL-Vergleichselement, mit dem festgestellt werden kann, ob ein XQuery-Ausdruck eine Sequenz von einem oder mehreren Elementen (und keine leere Sequenz) zurückgibt.

Vergleich von Methoden zur XML-Datenabfrage

Da XML-Daten auf verschiedene Weisen mithilfe von XQuery, SQL oder einer Kombination dieser Sprachen abgefragt werden können, hängt die zu wählende Methode von der jeweiligen Situation ab. In den folgenden Abschnitten werden Bedingungen beschrieben, die für eine bestimmte Abfragemethode vorteilhaft sind.

Nur XQuery

Reine XQuery-Abfragen können in folgenden Fällen eine geeignete Wahl sein:

- Wenn Anwendungen nur auf XML-Daten zugreifen, ohne relationale Nicht-XML-Daten abfragen zu müssen.
- Wenn zuvor in XQuery geschriebene Abfragen auf DB2 Database for Linux, UNIX and Windows migriert werden.
- Wenn Abfrageergebnisse zurückgegeben werden, die als Werte zur Konstruktion von XML-Dokumenten verwendet werden sollen.
- Wenn der Abfrageverfasser mit XQuery besser vertraut ist als mit SQL.

XQuery mit aufgerufenen SQL-Anweisungen

Abfragen mit XQuery, die SQL aufrufen, können (neben den im vorherigen Abschnitt zur Verwendung von reinem XQuery genannten Szenarios) in folgenden Fällen eine geeignete Wahl sein:

- Abfragen umfassen XML-Daten und relationale Daten, und SQL-Vergleichselemente sowie Indizes, die für die relationalen Spalten definiert sind, können in der Abfrage genutzt werden.
- Sie wollen XQuery-Ausdrücke auf die Ergebnisse folgender Elemente anwenden:
  - UDF-Aufrufe, da diese nicht direkt aus XQuery aufgerufen werden können
  - XML-Werte, die mithilfe von SQL/XML-Veröffentlichungsfunktionen aus relationalen Daten konstruiert werden
  - Abfragen, die DB2Net Search Extender verwenden, ein Produkt, das die Möglichkeit zur Volltextsuche in XML-Dokumenten bietet, jedoch mit SQL verwendet werden muss

Nur SQL

Wenn Sie XML-Daten nur mit SQL (ohne XQuery) abrufen, kann die Abfrage nur auf XML-Spaltenebene erfolgen. Aus diesem Grund können nur vollständige XML-Dokumente durch die Abfrage zurückgegeben werden. Diese Art der Verwendung ist in folgenden Fällen geeignet:

- Wenn Sie vollständige XML-Dokumente abrufen wollen.
- Wenn keine Abfrage auf der Basis von Werten innerhalb der gespeicherten Dokumente erforderlich ist oder wenn die Vergleichselemente der Abfrage auf andere Spalten, d. h. Nicht-XML-Spalten, der Tabelle angewendet werden.

SQL/XML-Funktionen mit Ausführung von XQuery-Ausdrücken

Die SQL/XML-Funktionen XMLQUERY und XMLTABLE sowie das XMLEXISTS-Vergleichselement ermöglichen eine Ausführung von XQuery-Ausdrücken innerhalb des SQL-Kontexts. Die Ausführung von XQuery innerhalb von SQL kann in folgenden Fällen eine geeignete Wahl sein:

- Vorhandene SQL-Anwendungen müssen zum Abfragen von Daten in XML-Dokumenten eingerichtet werden. Zum Abfragen von Daten innerhalb von XML-Dokumenten müssen XQuery-Ausdrücke ausgeführt werden. Dies kann mithilfe von SQL/XML-Funktionen geschehen.
- Anwendungen, die XML-Daten abfragen, müssen Parametermarken an den XQuery-Ausdruck übergeben. (Die Parametermarken werden zunächst an XQuery-Variablen in den Funktionen XMLQUERY oder XMLTABLE gebunden.)
- Der Abfrageverfasser ist mit SQL besser vertraut als mit XQuery.
- Sowohl relationale Daten als auch XML-Daten müssen durch eine einzige Abfrage zurückgegeben werden.
- Sie müssen XML-Daten und relationale Daten verknüpfen (Join).

- Sie wollen XML-Daten gruppieren oder zusammenfassen. Sie können die Klauseln GROUP BY oder ORDER BY eines Subselect auf die XML-Daten anwenden (zum Beispiel nachdem die XML-Daten mit der Funktion XMLTABLE abgerufen und im Tabellenformat erfasst wurden).

Angeben von XML-Namensbereichen

In einem XML-Dokument kann optional ein XML-Namensbereich angegeben werden, der als Präfix für Knotennamen im XML-Dokument verwendet wird. Um auf die Knoten in einem XML-Dokument zugreifen zu können, das einen Namensbereich verwendet, muss in den XQuery-Ausdrücken derselbe Namensbereich als Teil der Knotennamen angegeben werden.

Es ist auch möglich, für ein Dokument einen standardmäßigen XML-Namensbereich und XML-Namensbereiche für bestimmte Elemente in einem Dokument anzugeben.

Bitte beachten Sie, dass Namensbereichsdeklarationen durch ein Semikolon (`;`) abgeschlossen werden müssen. Dies bedeutet, dass ein Semikolon nicht als Abschlusszeichen für Anweisungen verwendet werden kann, wenn Sie auch mit SQL-Anweisungen und XQuery-Ausdrücken arbeiten wollen, die Semikola enthalten, beispielsweise indem Sie den Befehlszeilenprozessor mit `db2 -t` aufrufen. Sie können mit der Option `-td` ein anderes Abschlusszeichen als das Semikolon angeben, um sicherzustellen, dass die Anweisungen mit Namensbereichsdeklarationen nicht falsch interpretiert werden. In den Beispielen im Lernprogramm wird die Tilde (`~`) als Abschlusszeichen verwendet (`-td~`); das Prozentzeichen (`%`) wird jedoch auch häufig eingesetzt (`-td%`).

Das Lernprogramm für pureXML beispielsweise verwendet XML-Dokumente, die einen Standardnamensbereich für Elemente für ein XML-Dokument angeben. Das folgende XML-Dokument gehört zu den im Lernprogramm verwendeten XML-Dokumenten:

```
<customerinfo xmlns="http://posample.org" Cid="1002">
  <name>Jim Noodle</name>
  <addr country="Canada">
    <street>25 EastCreek</street>
        <city>Markham</city>
    <prov-state>Ontario</prov-state>
        <pcode-zip>N9C 3T6</pcode-zip>
  </addr>
    <phone type="work">905-555-7258</phone>
</customerinfo>
```

Der Rootknoten des XML-Dokuments bindet den Standardnamensbereich für Elemente für das Dokument an die URI (Universal Resource Identifier) `http://posample.org`.

```
<customerinfo xmlns="http://posample.org" Cid="1002">
```

Die XQuery-Ausdrücke, die Sie im Lernprogramm ausführen, binden ebenfalls eine URI als Standardnamensbereich für Elemente, indem der Prolog **declare default element namespace** zum Deklarieren eines Standardelementnamensbereichs eingeschlossen wird. Der XQuery-Ausdruck in der nachstehenden Anweisung SELECT beispielsweise deklariert einen Standardnamensbereich für Elemente. Bei Ausführung der Anweisung SELECT für die im Lernprogramm erstellte Tabelle CUSTOMER wird eine Kunden-ID zurückgegeben:

```
SELECT cid FROM customer
   WHERE XMLEXISTS('declare default element namespace "http://posample.org";
   $i/customerinfo/addr/city[ . = "Markham"]' passing INFO as "i")
```

Durch Verwendung derselben URI wie die des Standardnamensbereichs für Elemente im XML-Dokument qualifiziert der Ausdruck die Knotennamen im Ausdruck durch das korrekte Namensbereichspräfix. Ohne die Deklaration eines Standardnamensbereichs für Elemente oder mit einer anderen gebundenen URI als die dieses Standardnamensbereichs qualifiziert der Ausdruck die Knotennamen nicht mit dem korrekten Namensbereich, und es werden keine Daten zurückgegeben. Die nachstehende Anweisung SELECT beispielsweise ähnelt der vorherigen Anweisung, verfügt jedoch nicht über eine Standardnamensbereichsdeklaration. Bei Ausführung dieser Anweisung für die im Lernprogramm erstellte Tabelle CUSTOMER werden keine Daten zurückgegeben.

```
SELECT cid FROM customer
WHERE XMLEXISTS('$i/customerinfo/addr/city[ . = "Markham"]'
   passing INFO as "i")
```

Ein Namensbereichspräfix mit einem Knotennamen verwenden

Um einen Knotennamen mit einem Namensbereich zu qualifizieren, können Sie das Namensbereichspräfix den jeweiligen Knotennamen hinzufügen. Das Präfix und der Knotenname werden durch einen Doppelpunkt (:) getrennt. Beim Knoten `po:addr` beispielsweise wird das Namensbereichspräfix `po` vom lokalen Knotennamen `addr` getrennt. Wenn Sie ein Namensbereichspräfix mit einem Knotennamen qualifzieren, müssen Sie sicherstellen, dass das Präfix an eine URI gebunden ist. Der XQuery-Ausdruck in der nachstehenden Anweisung SELECT beispielsweise bindet das Namensbereichspräfix `po` an die URI `http://posample.org` durch Deklaration des Namensbereichs `po`. Bei der Ausführung der folgenden Anweisung für die im Lernprogramm erstellte Tabelle CUSTOMER wird ein Ergebnis zurückgegeben.

```
SELECT cid FROM customer
   WHERE XMLEXISTS('
    declare namespace po = "http://posample.org";
   $i/po:customerinfo/po:addr/po:city[ . = "Markham"]' passing INFO as "i")
```

Das Namensbereichspräfix `po` könnte auch durch ein beliebiges anderes Präfix ersetzt werden. Wichtig ist nur die an das Präfix gebundene URI. Der XQuery-Ausdruck in der nachstehenden Anweisung SELECT beispielsweise verwendet das Namensbereichspräfix `mytest`, ist aber dennoch äquivalent zum Ausdruck in der vorherigen Anweisung:

```
SELECT cid FROM customer
WHERE XMLEXISTS('declare namespace mytest = "http://posample.org";
   $i/mytest:customerinfo/mytest:addr/mytest:city[ . = "Markham"]'
   passing INFO as "i")
```

Ein Platzhalterzeichen als Namensbereichspräfix verwenden

Sie können in einem XQuery-Ausdruck ein Platzhalterzeichen verwenden, das einem beliebigen in den XML-Daten verwendeten Namensbereich entspricht. Im XQuery-Ausdruck der folgenden Anweisung SELECT wird ein Platzhalterzeichen als Entsprechung mit allen Namensbereichspräfixen verwendet:

```
SELECT cid FROM customer
WHERE XMLEXISTS('$i/*:customerinfo/*:addr/*:city[ . = "Markham"]'
   passing INFO as "i")
```

Bei der Ausführung der Anweisung SELECT für die im Lernprogramm erstellte Tabelle CUSTOMER wird eine Kunden-ID zurückgegeben.

Beispiel: Änderung des Namensbereichspräfix eines Elements

Diese Beispiele veranschaulichen, wie Namensbereichsbindungen zum qualifizierten Namen (QName) eines Elements hinzugefügt und zugeordnete Bindungen entfernt werden.

Der qualifizierte Name (QName) eines Elements besteht aus dem optionalen Namensbereichspräfix und dem lokalen Namen. Das Namensbereichspräfix und der lokale Name werden durch einen Doppelpunkt voneinander getrennt. Das Namensbereichspräfix (sofern vorhanden) ist an eine URI (Universal Resource Identifier) gebunden und stellt eine Kurzform der URI dar. Der erweiterte QName umfasst die Namensbereichs-URI und einen lokalen Namen.

Mit Funktionen wie fn:QName, fn:local-name und fn:namespace-uri können Sie Attribut- und Knotennamen umbenennen und nach der URI suchen, die dem Namensbereichspräfix eines Knotens zugeordnet ist.

XQuery-Namensbereichsdeklarationen werden durch ein Semikolon (;) abgeschlossen. Sie können Anweisungen nicht mit einem Semikolon abschließen. Im Beispiel wird die Tilde (~) als Abschlusszeichen verwendet. Dies entspricht dem im Lernprogramm für pureXML verwendeten Abschlusszeichen.

Hinzufügen eines Namensbereichspräfix zu einem Element

In den Beispielen wird ein XML-Dokument aus einer Tabelle verwendet. Die folgenden Anweisungen dienen zum Erstellen der Tabelle und zum Einfügen eines XML-Dokuments in diese Tabelle:

```
CREATE TABLE XMLTEST (ID BIGINT NOT NULL PRIMARY KEY, XMLDOC XML ) ~
INSERT INTO XMLTEST Values (4,
  '<depts>
    <dept id="A07">
        <emp id="31201" >
           <location region="31" />
        </emp>
        <emp type="new" id = "23322" >
          <moved:location xmlns:moved="http://oldcompany.com" region="43" />
        </emp>
    </dept>
    </depts> ') ~
```

Mit dem XQuery-Ausdruck in der folgenden Anweisung SELECT wird ein Namensbereichspräfix zu einem Element hinzugefügt. Der XQuery-Ausdruck erstellt mithilfe von fn:QName einen qualifizierten Namen (QName) mit einer Namensbereichsbindung.

Mit der XQuery-Klausel **let** wird ein leeres Element mit dem Namen `emp` und dem Namensbereich `http://example.com/new` erstellt. Im Ausdruck **transform** ändert der Ausdruck **rename** den Namen des mit dem XPath-Ausdruck `$newdept/depts/dept/emp[@type="new"]` angegebenen Elements. Das Element hat den Namen `emp`, besitzt jedoch kein Namensbereichspräfix.

```
SELECT XMLQUERY ('
 let $newemp := fn:QName( "http://mycompany.com", "new:emp")
 return
 transform
   copy $newdept := $doc
```

```
    modify
    do rename $newdept/depts/dept/emp[@type="new"] as $newemp
  return
  $newdept  ' passing XMLDOC as "doc" )
from XMLTEST where ID = 4 ~
```

Der XQuery-Ausdruck gibt die folgenden XML-Daten zurück, wobei die Namensbereichsbindung als Teil des Knotens `new:emp` angegeben ist. Das geänderte Dokument ist gültiges XML, das Namensbereichspräfixe umfasst, die an einen Namensbereich gebunden sind. Zu dem gerade umbenannten Element wird eine Namensbereichsdeklaration hinzugefügt.

```
<depts>
 <dept id="A07">
  <emp id="31201">
   <location region="31" />
  </emp>
  <new:emp xmlns:new="http://mycompany.com" type="new" id="23322">
   <moved:location xmlns:moved="http://oldcompany.com" region="43" />
  </new:emp>
 </dept>
</depts>
```

Entfernen eines Namensbereichspräfix von einem Element

Sie können ein Namensbereichspräfix von einem Knotennamen entfernen, indem Sie den Knotennamen in einen qualifizierten Namen (QName) ohne Namensbereichsbindung umbenennen. Der folgende XQuery-Ausdruck in der Anweisung SELECT ermittelt mithilfe einer Klausel 'for' und der Funktion fn:namespace-uri die URI der Elementknoten in den einzelnen Knoten 'emp'. Wenn die URI `http://oldcompany.com` lautet, verwendet der Umbenennungsausdruck die Funktionen fn:QName und fn:local-name, um das Namensbereichspräfix aus dem Elementknoten zu entfernen.

```
 SELECT XMLQUERY ('
  transform
    copy $newdept := $x
    modify
     for $testemp in $newdept/depts/dept/*:emp/*
     return
     if ( fn:namespace-uri( $testemp ) eq "http://oldcompany.com")
     then
       do rename $testemp as fn:QName( "", fn:local-name($testemp) )
     else()
  return
  $newdept
 ' passing XMLDOC as "x" )
from XMLTEST where ID = 4 ~
```

Der XQuery-Ausdruck gibt die folgenden XML-Daten zurück. Der Knotenname `new:location` wird durch `location` ersetzt. Die Namensbereichsbindung wird nicht von dem Knoten entfernt.

```
<depts>
  <dept id="A07">
    <emp id="31201">
      <location region="31" />
    </emp>
    <emp type="new" id="23322">
      <location xmlns:moved="http://oldcompany.com" region="43" />
    </emp>
  </dept>
</depts>
```

XMLQUERY - Funktionsübersicht

Mit der SQL-Skalarfunktion XMLQUERY können Sie einen XQuery-Ausdruck innerhalb eines SQL-Kontexts ausführen. Sie können Variablen an den in XMLQUERY angegebenen XQuery-Ausdruck übergeben. XMLQUERY gibt einen XML-Wert zurück, der eine XML-Sequenz darstellt. Diese Sequenz kann leer sein oder auch ein oder mehrere Elemente enthalten.

Das Ausführen von XQuery-Ausdrücken innerhalb des SQL-Kontexts bietet folgende Möglichkeiten:

- Sie können Operationen nur an Teilen gespeicherter XML-Dokumente anstatt gesamter XML-Dokumente ausführen. (Nur XQuery kann zum Abfragen von Daten innerhalb eines XML-Dokuments verwendet werden. Abfragen, die nur mit SQL arbeiten, können nur auf der Ebene ganzer Dokumente operieren.)
- Sie können XML-Daten in SQL-Abfragen einbinden.
- Sie können auf relationalen Daten und XML-Daten gleichzeitig operieren.
- Sie können weitere SQL-Verarbeitungsschritte auf die zurückgegebenen XML-Werte anwenden (zum Beispiel, um Ergebnisse durch die Klausel ORDER BY eines Subselects zu sortieren).

Weitere detaillierte Informationen finden Sie in der Dokumentation zum Vergleich von Abfragemethoden.

Beachten Sie, dass in XQuery die Groß-/Kleinschreibung unterschieden wird, sodass XQuery-Ausdrücke und Variablen in der Funktion XMLQUERY sorgfältig angegeben werden müssen.

Wenn die vollständige Funktionalität zum Übergeben von SQL-Ausdrücken nicht benötigt wird, steht auch eine vereinfachte Syntax zur Verfügung, mit der Spaltennamen übergeben werden können, ohne die Namen in der Klausel **PASSING** explizit angeben zu müssen. Weitere Informationen hierzu enthält der Abschnitt Vereinfachte Übergabe von Spaltennamen mit XMLEXISTS, XMLQUERY und XMLTABLE.

Von XMLQUERY zurückgegebene nicht leere Sequenzen

Die Funktion XMLQUERY gibt eine nicht leere Sequenz zurück, wenn der in der Funktion angegebene XQuery-Ausdruck in einer nicht leeren Sequenz resultiert.

Beispiel: Betrachten Sie die beiden folgenden XML-Dokumente, die in der XML-Spalte INFO der Tabelle CUSTOMER gespeichert sind:

```
<customerinfo Cid="1002">
  <name>Jim Noodle</name>
  <addr country="Canada">
    <street>25 EastCreek</street>
    <city>Markham</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>N9C 3T6</pcode-zip>
  </addr>
  <phone type="work">905-555-7258</phone>
</customerinfo>

<customerinfo Cid="1003">
  <name>Robert Shoemaker</name>
    <addr country="Canada">
    <street>1596 Baseline</street>
    <city>Aurora</city>
    <prov-state>Ontario</prov-state>
```

```
    <pcode-zip>N8X 7F8</pcode-zip>
  </addr>
  <phone type="work">905-555-7258</phone>
  <phone type="home">416-555-2937</phone>
  <phone type="cell">905-555-8743</phone>
  <phone type="cottage">613-555-3278</phone>
</customerinfo>
```

Nun wird die folgende Abfrage ausgeführt:

```
SELECT XMLQUERY ('$d/customerinfo/phone' passing INFO as "d")
FROM CUSTOMER
```

Die Ergebnismenge enthält nur zwei Zeilen, wie im Folgenden dargestellt (die Ausgabe wurde aus Gründen der Übersichtlichkeit formatiert):

```
1
-----------
    <phone type="work">905-555-7258</phone>
   <phone type="work">905-555-7258</phone><phone type="home">416-555-2937</
   phone><phone type="cell">905-555-8743</phone><phone type="cottage">613-
   555-3278</phone>
 2 Satz/Sätze ausgewählt.
```

Beachten Sie, dass die erste Zeile eine Sequenz aus einem Element <phone> enthält, während die zweite Zeile eine Sequenz aus vier Elementen <phone> enthält. Dieses Ergebnis ist darauf zurückzuführen, dass das zweite XML-Dokument vier Elemente <phone> enthält und die Funktion XMLQUERY eine Sequenz aus allen Elementen zurückgibt, die den XQuery-Ausdruck erfüllen. (Beachten Sie, dass das Ergebnis in der zweiten Zeile kein korrekt formatiertes Dokument darstellt. Es muss infolgedessen sichergestellt werden, dass jede Anwendung, die ein solches Ergebnis empfängt, diese Funktionsweise ordnungsgemäß verarbeiten kann.)

Das obige Beispiel veranschaulicht, wie die Funktion XMLQUERY gewöhnlich verwendet wird: angewendet auf jeweils ein XML-Dokument gleichzeitig, wobei jede Zeile in der Ergebnistabelle das Ergebnis aus einem Dokument darstellt. Die Funktion XMLQUERY kann jedoch auch auf mehrere Dokumente gleichzeitig angewendet werden, wie es zum Beispiel der Fall ist, wenn in einer einzigen Sequenz mehrere Dokumente enthalten sind. In diesem Fall werden die Ergebnisse aus der Anwendung von XMLQUERY auf alle Dokumente in der Sequenz in einer einzigen Zeile zurückgegeben.

Nehmen Sie zum Beispiel an, dass die gleichen Dokumente wie oben in der Spalte INFO der Tabelle CUSTOMER gespeichert sind. Die Funktion 'db2-fn:xmlcolumn' in der folgenden Abfrage gibt eine Sequenz zurück, die die beiden XML-Dokumente in der Spalte INFO enthält.

```
VALUES
 (XMLQUERY
   ('db2-fn:xmlcolumn("CUSTOMER.INFO")/customerinfo/phone'))
```

Anschließend wird die Funktion XMLQUERY auf diese einzelne Sequenz aus XML-Dokumenten angewendet und die Ergebnismenge enthält nur eine Zeile, wie im Folgenden dargestellt:

```
1
-----------
   <phone type="work">905-555-7258</phone><phone type="work">905-555-7258</
   phone><phone type="home">416-555-2937</phone><phone type="cell">905-555-
   8743</phone><phone type="cottage">613-555-3278</phone>
```

```
1 Satz/Sätze ausgewählt.
```

Alle Elemente <phone> aus den XML-Dokumenten in der Spalte INFO werden in einer einzigen Zeile zurückgegeben, weil die Funktion XMLQUERY auf einem einzigen Wert operiert: der Sequenz aus XML-Dokumenten, die aus der Funktion 'db2-fn:xmlcolumn' zurückgegeben wird.

Von XMLQUERY zurückgegebene leere Sequenzen

Die Funktion XMLQUERY gibt eine leere Sequenz zurück, wenn der in der Funktion angegebene XQuery-Ausdruck eine leere Sequenz zurückgibt.

In der folgenden Abfrage gibt XMLQUERY zum Beispiele eine leere Sequenz für jede Zeile der Tabelle CUSTOMER zurück, die kein Element <city> mit dem Wert "Aurora" in der Spalte INFO enthält.

```
SELECT Cid, XMLQUERY ('$d//addr[city="Aurora"]' passing INFO as "d") AS ADDRESS
FROM CUSTOMER
```

Nehmen Sie an, dass drei Zeilen der Tabelle CUSTOMER vorhanden sind, jedoch nur ein XML-Dokument, das ein Element <city> mit dem Wert "Aurora" enthält. Die folgende Tabelle würde aus der oben gezeigten SELECT-Anweisung resultieren (die Ausgabe wurde aus Gründen der Übersichtlichkeit formatiert).

Tabelle 7. Ergebnistabelle

CID	ADDRESS
1001	
1002	
1003	<addr country="Canada"><street>1596 Baseline</street><city>Aurora</city><prov-state>Ontario</prov-state><pcode-zip>N8X-7F8</pcode-zip></addr>

Beachten Sie, dass leere Sequenzen aus serialisiertem XML der Länge null anstelle von NULL-Werten für Zeilen zurückgegeben werden, die kein Element <city> mit dem Wert "Aurora" enthalten. Das Element <addr> wird jedoch in der dritten Zeile zurückgegeben, da es den XQuery-Ausdruck erfüllt. In der dritten Zeile wird eine nicht leere Sequenz zurückgegeben.

Sie können die Rückgabe von Zeilen, die leere Sequenzen enthalten, vermeiden, indem Sie ein Vergleichselement, wie zum Beispiel XMLEXISTS, in der WHERE-Klausel und nicht in der SELECT-Klausel Ihrer Anweisung anwenden. Die vorige Abfrage lässt sich zum Beispiel wie folgt umschreiben, wobei das Filtervergleichselement aus der Funktion XMLQUERY in die WHERE-Klausel verschoben wird:

```
SELECT Cid, XMLQUERY ('$d/customerinfo/addr' passing c.INFO as "d")
FROM Customer as c
WHERE XMLEXISTS ('$d//addr[city="Aurora"]' passing c.INFO as "d")
```

Die aus dieser Abfrage resultierende Tabelle sieht folgendermaßen aus:

Tabelle 8. Ergebnistabelle

CID	ADDRESS
1003	<addr country="Canada"><street>1596 Baseline</street><city>Aurora</city><prov-state>Ontario</prov-state><pcode-zip>N8X-7F8</pcode-zip></addr>

Die Funktion XMLQUERY wird in der Regel in einer SELECT-Klausel verwendet, um Fragmente ausgewählter Dokumente abzurufen. Vergleichselemente, die im XQuery-Ausdruck der Funktion XMLQUERY angegeben werden, filtern keine Zeilen aus der Ergebnismenge, sondern dienen lediglich zur Ermittlung der zurückgegebenen Fragmente. Um Zeilen effektiv aus der Ergebnismenge zu eliminieren, müssen Sie ein Vergleichselement in der WHERE-Klausel anwenden. Das XMLEXISTS-Vergleichselement kann zur Anwendung von Vergleichselementen verwendet werden, die von Werten innerhalb gespeicherter XML-Dokumente abhängig sind.

Explizite Umwandlung von XMLQUERY-Ergebnissen in Nicht-XML-Typen

Wenn Sie die Ergebnisse der Funktion XMLQUERY zur weiteren Verarbeitung, zum Beispiel für Vergleichs- oder Sortieroperationen, an den SQL-Kontext zurückgeben wollen, müssen Sie den zurückgegebenen XML-Wert in einen kompatiblen SQL-Datentyp umsetzen. Sie können die XMLCAST-Spezifikation dazu verwenden, Umsetzungen zwischen XML- und Nicht-XML-Werten durchzuführen.

Anmerkung:

1. Sie können das Ergebnis der Funktion XMLQUERY nur dann in einen SQL-Datentyp umsetzen, wenn der in XMLQUERY angegebene XQuery-Ausdruck eine Sequenz zurückgibt, die nur ein Element enthält, das atomisiert wurde.
2. In einer Nicht-UTF-8-Datenbank führt die Umsetzung von XMLQUERY in einen SQL-Datentyp zu einer Codepagekonvertierung, da der zurückgegebene Wert aus einer internen UTF-8-Codierung in die Datenbankcodepage konvertiert wird. Alle Codepunkte im Rückgabewert, die nicht Teil der Datenbankcodepage sind, werden durch Substitutionszeichen ersetzt. Die Einführung von Substitutionszeichen kann bei Vergleichen zwischen XML-Werten und Nicht-XML-Werte zu unerwarteten Ergebnissen führen. Daher sollte sorgfältig darauf geachtet werden, dass die gespeicherten XML-Daten ausschließlich Codepunkte enthalten, die zur Datenbankcodepage gehören.

Beispiel: Vergleichen von XML-Werten mit Nicht-XML-Werten in einer Abfrage

In der folgenden Abfrage wird die von der Funktion XMLQUERY zurückgegebene Sequenz von ihrem XML-Typ in einen Zeichentyp umgewandelt, sodass sie mit der Spalte NAME der Tabelle PRODUCT verglichen werden kann. (Falls der XML-Wert, der aus der Funktion XMLQUERY resultiert, keine serialisierte Zeichenfolge ist, kann die XMLCAST-Operation fehlschlagen.)

```
SELECT R.Pid
FROM PURCHASEORDER P, PRODUCT R
WHERE R.NAME =
        XMLCAST( XMLQUERY ('$d/PurchaseOrder/item/name'
                           PASSING P.PORDER AS "d") AS VARCHAR(128))
```

Beispiel: Sortieren von XMLQUERY-Ergebnissen (mit ORDER BY)

In der folgenden Abfrage werden Produkt-IDs in einer Reihenfolge zurückgegeben, die nach dem Wert des Elements <name> der als XML-Dokument gespeicherten Beschreibung des Produkts sortiert wird. Da SQL keine Sortierung an XML-Werten ausführen kann, muss die Sequenz in einen Werttyp umgewandelt werden, der von SQL sortiert werden kann. In diesem Fall ist dies ein Zeichentyp.

```
SELECT Pid
FROM PRODUCT
ORDER BY XMLCAST(XMLQUERY ('$d/product/description/name'
                           PASSING DESCRIPTION AS "d") AS VARCHAR(128))
```

Umsetzung zwischen Datentypen

Es gibt viele Situationen, in denen ein Wert mit einem bestimmten Datentyp in einen anderen Datentyp oder in denselben Datentyp mit einer anderen Länge, Genauigkeit und Skala *umgesetzt* werden muss.

Die Datentypumstufung ist ein Beispiel dafür, dass beim Umstufen eines Datentyps in einen anderen der Wert in den neuen Datentyp umgesetzt werden muss. Ein Datentyp, der in einen anderen Datentyp umgesetzt werden kann, ist aus dem Quellendatentyp in den Zieldatentyp *umsetzbar*.

Die Umsetzung bzw. das Casting eines Datentyps in einen anderen kann implizit oder explizit erfolgen. Mit den Umsetzungsfunktionen und der Spezifikation CAST oder XMLCAST können Sie einen Datentyp je nach den einbezogenen Datentypen explizit ändern. Außerdem müssen beim Erstellen einer benutzerdefinierten Funktion mit Quelle die Datentypen der Parameter der Quellenfunktion in die Datentypen der Funktion umsetzbar sein, die erstellt wird.

Die zwischen integrierten Datentypen unterstützten Umsetzungen sind in Tabelle 9 auf Seite 89 aufgeführt. Die erste Spalte stellt den Datentyp des Umsetzungsoperanden (Quellendatentyp) dar und die Datentypen horizontal zur Kopfzeile stellen den Zieldatentyp der Umsetzungsoperation dar. Ein 'J' zeigt an, dass sich mit der Spezifikation CAST der Quellen- und Zieldatentyp kombinieren lassen. Die Fälle, in denen nur die Spezifikation XMLCAST verwendet werden kann, sind mit einer Anmerkung versehen.

Wenn beim Umsetzen eines beliebigen Datentyps in einen Zeichen- oder Grafikdatentyp ein Abschneidevorgang erfolgt, wird eine Warnung zurückgegeben, sobald ein belegtes Zeichen abgeschnitten wird. Dieses Abschneideverhalten unterscheidet sich von der Zuweisung zu einem Zeichen- oder Grafikdatentyp, bei welcher das Abschneiden von belegten Zeichen zu einem Fehler führt.

Die folgenden Umsetzungen mit einzigartigen Datentypen werden unterstützt (mit der Spezifikation CAST, sofern nicht anders vermerkt):

- Umsetzung aus dem einzigartigen Datentyp *DT* in dessen Quellendatentyp *S*
- Umsetzung aus dem Quellendatentyp *S* mit dem einzigartigen Datentyp *DT* in den einzigartigen Datentyp *DT*
- Umsetzung aus dem einzigartigen Datentyp *DT* in denselben einzigartigen Datentyp *DT*
- Umsetzung aus einem Datentyp *A* in den einzigartigen Datentyp *DT*, wobei *A* in den Quellendatentyp *S* mit dem einzigartigen Datentyp *DT* umgestuft werden kann
- Umsetzung aus einem Typ INTEGER in den einzigartigen Datentyp *DT* mit einem Quellendatentyp SMALLINT
- Umsetzung aus einem Typ DOUBLE in den einzigartigen Datentyp *DT* mit einem Quellendatentyp REAL
- Umsetzung aus einem Typ DECFLOAT in den einzigartigen Datentyp DT mit einem Quellendatentyp DECFLOAT
- Umsetzung aus einem Typ VARCHAR in den einzigartigen Datentyp *DT* mit einem Quellendatentyp CHAR

- Umsetzung aus einem Typ VARGRAPHIC in den einzigartigen Datentyp *DT* mit einem Quellendatentyp GRAPHIC
- In Unicode-Datenbanken: Umsetzung aus einem Typ VARCHAR oder VARGRAPHIC in den einzigartigen Datentyp *DT* mit einem Quellendatentyp CHAR oder GRAPHIC
- Umsetzung aus einem einzigartigen Datentyp *DT* mit einem Quellendatentyp *S* in XML (mithilfe der Spezifikation XMLCAST)
- Umsetzung aus XML in einen einzigartigen Datentyp *DT* mit einem beliebigen integrierten Datentyp als Quelle, je nach dem XML-Schemadatentyp des XML-Werts mithilfe der Spezifikation XMLCAST

FOR BIT DATA-Zeichentypen können nicht in den Typ CLOB umgesetzt werden.

Bei Umsetzungen mit einem Arraytyp als Ziel muss der Datentyp der Elemente des Quellenarraywerts in den Datentyp der Elemente der Zielarraydaten umsetzbar sein (SQLSTATE 42846). Handelt es sich beim Zielarraytyp um einen normalen Array, muss der Quellenarraywert ein normaler Array sein (SQLSTATE 42821) und die Kardinalität des Quellenarraywerts muss kleiner als oder gleich wie die maximale Kardinalität des Zielarraydatentyps sein (SQLSTATE 2202F). Ist der Zielarraytyp ein assoziativer Array, muss der Datentyp des Index für den Quellenarraywert in den Datentyp des Index für den Zielarraytyp umsetzbar sein. Ein benutzerdefinierter Wert für einen Arraytyp kann nur in denselben benutzerdefinierten Arraytyp umgesetzt werden (SQLSTATE 42846).

Ein Cursortyp kann weder der Quellendatentyp noch der Zieldatentyp einer CAST-Spezifikation sein, es sei denn, es handelt sich um eine Umsetzung einer Parametermarke in einen Cursortyp.

Bei Umsetzungen mit einem Zeilentyp als Ziel müssen der Grad des Ausdrucks für den Quellenzeilenwert und der Grad des Zielzeilentyps übereinstimmen und jedes Feld im Ausdruck für den Quellenzeilenwert muss in das entsprechende Zielfeld umsetzbar sein. Ein benutzerdefinierter Wert für einen Zeilentyp kann nur in einen anderen benutzerdefinierten Zeilentyp mit demselben Namen umgesetzt werden (SQLSTATE 42846).

Ein Wert mit einem strukturierten Typ kann in keinen anderen Typ umgesetzt werden. Bei einem strukturierten Typ *ST* sollte keine Umsetzung in einen der ihm übergeordneten Typen erforderlich sein, da alle Methoden der übergeordneten Typen von *ST* auf *ST* anwendbar sind. Wenn die erforderliche Operation nur auf einen Subtyp von *ST* anwendbar ist, müssen Sie *ST* mit dem Ausdruck 'subtype-treatment' wie einen seiner Subtypen behandeln.

Wenn in eine Umsetzung ein benutzerdefinierter Datentyp einbezogen ist, der nicht durch einen Schemanamen qualifiziert ist, wird mithilfe des *SQL-Pfads* das erste Schema gesucht, das den benutzerdefinierten Datentyp mit diesem Namen enthält.

Die folgenden Umsetzungen mit Verweistypen werden unterstützt:

- Umsetzung aus dem Verweistyp *RT* in dessen Darstellungsdatentyp *S*
- Umsetzung aus dem Darstellungsdatentyp *S* des Verweistyps *RT* in den Verweistyp *RT*
- Umsetzung aus dem Verweistyp *RT* mit dem Zieltyp *T* in einen Verweistyp *RS* mit dem Zieltyp *S*, wobei *S* ein Supertyp von *T* ist

- Umsetzung aus einem Datentyp *A* in einen Verweistyp *RT*, wobei *A* in den Darstellungsdatentyp *S* des Verweistyps *RT* umgestuft werden kann

Wenn der Zieltyp eines in eine Umsetzung einbezogenen Verweisdatentyps nicht durch einen Schemanamen qualifiziert ist, wird mithilfe des *SQL-Pfads* das erste Schema gesucht, das den benutzerdefinierten Datentyp mit diesem Namen enthält.

Tabelle 9. Unterstützte Umsetzungen zwischen integrierten Datentypen

	Zieldatentyp																				
Quellendatentyp	**SMALLINT**	**INTEGER**	**BIGINT**	**DECIMAL**	**REAL**	**DOUBLE**	**DECFLOAT**	**CHAR**	**CHAR FBD[2]**	**VARCHAR**	**VARCHAR FBD[2]**	**CLOB**	**GRAPHIC**	**VARGRAPHIC**	**DBCLOB**	**BLOB**	**DATE**	**TIME**	**TIMESTAMP**	**XML**	**BOOLEAN**
SMALLINT	J	J	J	J	J	J	J	J	J	J	J	-	Y[1]	Y[1]	-	-	-	-	-	J[3]	J[7]
INTEGER	J	J	J	J	J	J	J	J	J	J	J	-	Y[1]	Y[1]	-	-	-	-	-	J[3]	J[7]
BIGINT	J	J	J	J	J	J	J	J	J	J	J	-	Y[1]	Y[1]	-	-	-	-	-	J[3]	J[7]
DECIMAL	J	J	J	J	J	J	J	J	J	J	J	-	Y[1]	Y[1]	-	-	-	-	-	J[3]	-
REAL	J	J	J	J	J	J	J	J	J	J	J	-	Y[1]	Y[1]	-	-	-	-	-	J[3]	-
DOUBLE	J	J	J	J	J	J	J	J	J	J	J	-	Y[1]	Y[1]	-	-	-	-	-	J[3]	-
DECFLOAT	J	J	J	J	J	J	J	J	J	J	J	-	Y[1]	Y[1]	-	-	-	-	-	-	-
CHAR	J	J	J	J	J	J	J	J	J	J	J	J	J[1]	J[1]	J[1]	J	J	J	J	J[4]	-
CHAR FOR BIT DATA	J	J	J	J	J	J	J	J	J	J	J	-	-	-	-	J	J	J	J	J[3]	-
VARCHAR	J	J	J	J	J	J	J	J	J	J	J	J	J[1]	J[1]	J[1]	J	J	J	J	J[4]	-
VARCHAR FOR BIT DATA	J	J	J	J	J	J	J	J	J	J	J	-	-	-	-	J	J	J	J	J[3]	-
CLOB	-	-	-	-	-	-	-	J	-	J	-	J	J[1]	J[1]	J[1]	J	-	-	-	J[4]	-
GRAPHIC	J[1]	J[1]	J[1]	J[1]	Y[1]	Y[1]	J[1]	J[1]	-	J[1]	-	J[1]	J	J	J	J	J[1]	J[1]	J[1]	J[3]	-
VARGRAPHIC	J[1]	J[1]	J[1]	J[1]	Y[1]	Y[1]	J[1]	J[1]	-	J[1]	-	J[1]	J	J	J	J	J[1]	J[1]	J[1]	J[3]	-
DBCLOB	-	-	-	-	-	-	-	J[1]	-	J[1]	-	J[1]	J	J	J	J	-	-	-	J[3]	-
BLOB	-	-	-	-	-	-	-	-	J	-	J	-	-	-	-	J	-	-	-	J[4]	-
DATE	-	J	J	J	-	-	-	J	J	J	J	-	J[1]	J[1]	-	-	J	-	J	J[3]	-
TIME	-	J	J	J	-	-	-	J	J	J	J	-	J[1]	J[1]	-	-	-	J	-	J[3]	-
TIMESTAMP	-	-	J	J	-	-	-	J	J	J	J	-	J[1]	J[1]	-	-	J	J	J	J[3]	-
XML	J[5]	J[5]	J[5]	J[5]	J[5]	J[5]	J[5]	J[5]	J[5]	J[5]	J[5]	J[5]	J[5]	J[5]	J[5]	J[5]	J[5]	J[5]	J[5]	J	-
BOOLEAN	J[7]	J[7]	J[7]	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	J[7]

Tabelle 9. Unterstützte Umsetzungen zwischen integrierten Datentypen (Forts.)

	Zieldatentyp																				
Quellendatentyp	**SMALLINT**	**INTEGER**	**BIGINT**	**DECIMAL**	**REAL**	**DOUBLE**	**DECFLOAT**	**CHAR**	**CHAR FOR BIT DATA**[2]	**VARCHAR**	**VARCHAR FOR BIT DATA**[2]	**CLOB**	**GRAPHIC**	**VARGRAPHIC**	**DBCLOB**	**BLOB**	**DATE**	**TIME**	**TIMESTAMP**	**XML**	**BOOLEAN**
SMALLINT																					
INTEGER																					
BIGINT																					
DECIMAL																					
REAL																					
DOUBLE																					
DECFLOAT																					
CHAR																					
CHAR FOR BIT DATA[2]																					
VARCHAR																					
VARCHAR FOR BIT DATA[2]																					
CLOB																					
GRAPHIC																					
VARGRAPHIC																					
DBCLOB																					
BLOB																					
DATE																					
TIME																					
TIMESTAMP																					
XML																					
BOOLEAN																					

Anmerkungen

- Informationen zu den unterstützten Umsetzungen von benutzerdefinierten Datentypen und Verweistypen finden Sie in der Beschreibung über der Tabelle.
- Ein Wert mit einem strukturierten Typ kann in keinen anderen Typ umgesetzt werden.
- Die Datentypen LONG VARCHAR und LONG VARGRAPHIC werden zwar weiterhin unterstützt, sind jedoch veraltet, werden nicht empfohlen und könnten in einem zukünftigen Release entfernt werden.

[1] Die Umsetzung wird nur bei Unicode-Datenbanken unterstützt.

[2] FOR BIT DATA

[3] Die Umsetzung ist nur mit XMLCAST möglich.

[4] Eine XMLPARSE-Funktion wird implizit verarbeitet, um eine Zeichenfolge bei deren Zuweisung (INSERT oder UPDATE) zu einer XML-Spalte in XML zu konvertieren. Die Zeichenfolge muss ein korrekt formatiertes XML-Dokument sein, damit die Zuweisung gelingt.

[5] Die Umsetzung ist nur mit XMLCAST möglich und hängt von dem zugrunde liegenden XML-Schemadatentyp des XML-Werts ab. Detaillierte Informationen finden Sie im Abschnitt zu „XMLCAST".

[6] Ein Cursortyp kann weder der Quellendatentyp noch der Zieldatentyp einer CAST-Spezifikation sein, es sei denn, es handelt sich um eine Umsetzung einer Parametermarke in einen Cursortyp.

[7] Wird nur unter Verwendung der CAST-Spezifikation unterstützt. Es ist keine Umsetzungsfunktion vorhanden.

In Tabelle 10 finden Sie Angaben dazu, wo sich die Regeln befinden, die bei der Umsetzung in die angegebenen Zieldatentypen gelten.

Tabelle 10. Regeln bei der Umsetzung in einen Datentyp

Zieldatentyp	Regeln
SMALLINT	Falls der Quellentyp BOOLEAN lautet, dann ist TRUE die Umsetzung von 1 und FALSE die Umsetzung von 0. Für alle anderen Quellentypen finden Sie Angaben unter .„SMALLINT, Skalarfunktion" in *SQL Reference Volume 1*
INTEGER	Falls der Quellentyp BOOLEAN lautet, dann ist TRUE die Umsetzung von 1 und FALSE die Umsetzung von 0. Für alle anderen Quellentypen finden Sie Angaben unter .„INTEGER, Skalarfunktion" in *SQL Reference Volume 1*

Tabelle 10. Regeln bei der Umsetzung in einen Datentyp (Forts.)

Zieldatentyp	**Regeln**
BIGINT	Falls der Quellentyp BOOLEAN lautet, dann ist TRUE die Umsetzung von 1 und FALSE die Umsetzung von 0. Für alle anderen Quellentypen finden Sie Angaben unter .„BIGINT, Skalarfunktion" in *SQL Reference Volume 1*
DECIMAL	Abschnitt „DECIMAL scalar function" in *SQL Reference Volume 1*
NUMERIC	Abschnitt „DECIMAL scalar function" in *SQL Reference Volume 1*
REAL	Abschnitt „REAL scalar function" in *SQL Reference Volume 1*
DOUBLE	Abschnitt „DOUBLE scalar function" in *SQL Reference Volume 1*
DECFLOAT	Abschnitt „DECFLOAT scalar function" in *SQL Reference Volume 1*
CHAR	Abschnitt „CHAR scalar function" in *SQL Reference Volume 1*
VARCHAR	Abschnitt „VARCHAR scalar function" in *SQL Reference Volume 1*
CLOB	Abschnitt „CLOB scalar function" in *SQL Reference Volume 1*
GRAPHIC	Abschnitt „GRAPHIC scalar function" in *SQL Reference Volume 1*
VARGRAPHIC	Abschnitt „VARGRAPHIC scalar function" in *SQL Reference Volume 1*
DBCLOB	Abschnitt „DBCLOB scalar function" in *SQL Reference Volume 1*
BLOB	Abschnitt „BLOB scalar function" in *SQL Reference Volume 1*
DATE	Abschnitt „DATE scalar function" in *SQL Reference Volume 1*
TIME	Abschnitt „TIME scalar function" in *SQL Reference Volume 1*
TIMESTAMP	Wenn der Quellentyp eine Zeichenfolge ist, lesen Sie den Abschnitt „TIMESTAMP scalar function" in *SQL Reference Volume 1* . Dort wird ein Operand angegeben. Wenn der Quellendatentyp DATE ist, besteht die Zeitmarke aus dem angegebenen Datum und der Uhrzeit 00:00:00.
BOOLEAN	Falls der Quellentyp numerisch ist, dann ist 0 die Umsetzung von FALSE und 1 ist die Umsetzung von TRUE. NULL ist die Umsetzung nach NULL.

Umsetzen von Nicht-XML-Werten in XML-Werte

Tabelle 11. Unterstützte Umsetzungen von Nicht-XML-Werten in XML-Werte

	Zieldatentyp	
Quellendatentyp	**XML**	**Resultierender XML-Schematyp**
SMALLINT	J	xs:short
INTEGER	J	xs:int
BIGINT	J	xs:long
DECIMAL oder NUMERIC	J	xs:decimal
REAL	J	xs:float
DOUBLE	J	xs:double
DECFLOAT	N	-
CHAR	J	xs:string
VARCHAR	J	xs:string
CLOB	J	xs:string
GRAPHIC	J	xs:string
VARGRAPHIC	J	xs:string
DBCLOB	J	xs:string
DATE	J	xs:date
TIME	J	xs:time
TIMESTAMP	J	xs:dateTime[1]
BLOB	J	xs:base64Binary
Zeichentyp FOR BIT DATA	J	xs:base64Binary
Einzigartiger Datentyp		Verwenden Sie beim Quellentyp des einzigartigen Datentyps diese Tabelle.

Hinweise

[1] Der Quellendatentyp TIMESTAMP unterstützt eine Zeitmarkengenauigkeit von 0 bis 12. Die maximale Genauigkeit von Sekundenbruchteilen von xs:dateTime ist 6. Wenn die Zeitmarkengenauigkeit eines TIMESTAMP-Quellendatentyps den Wert 6 überschreitet, wird der Wert beim Umsetzen in xs:dateTime abgeschnitten.

Die Datentypen LONG VARCHAR und LONG VARGRAPHIC werden zwar weiterhin unterstützt, sind jedoch veraltet, werden nicht empfohlen und könnten in einem zukünftigen Release entfernt werden.

Beim Umsetzen von Zeichenfolgewerten in XML-Werte darf der sich ergebende atomare Wert xs:string keine unzulässigen XML-Zeichen enthalten (SQLSTATE 0N002). Wenn die Eingabezeichenfolge nicht in Unicode vorliegt, werden die Eingabezeichen in Unicode konvertiert.

Beim Umsetzen von binären SQL-Typen ergeben sich atomare XQuery-Werte des Typs xs:base64Binary.

Umsetzen von XML-Werten in Nicht-XML-Werte

Eine XMLCAST-Umsetzung aus einem XML-Wert in einen Nicht-XML-Wert kann als zwei Umsetzungen beschrieben werden: Zuerst wird der XML-Quellenwert mit

einer XQuery-Umsetzung in einen XQuery-Typ konvertiert, der dem SQL-Zieltyp entspricht, und danach folgt eine Umsetzung aus dem entsprechenden XQuery-Typ in den eigentlichen SQL-Typ.

XMLCAST wird unterstützt, wenn dem Zieltyp ein unterstützter XQuery-Zieltyp entspricht und wenn eine unterstützte XQuery-Umsetzung aus dem Typ des Quellenwerts in den entsprechenden XQuery-Zieltyp vorhanden ist. Der bei der XQuery-Umsetzung verwendete Zieltyp basiert auf dem entsprechenden XQuery-Zieltyp und enthält möglicherweise weitere Einschränkungen.

In der folgenden Tabelle sind die XQuery-Typen aufgeführt, die bei einer solchen Konvertierung entstehen.

Tabelle 12. Unterstützte Umsetzungen von XML-Werten in Nicht-XML-Werte

	Quellendatentyp	
Zieldatentyp	**XML**	**Entsprechender XQuery-Zieltyp**
SMALLINT	J	xs:short
INTEGER	J	xs:int
BIGINT	J	xs:long
DECIMAL oder NUMERIC	J	xs:decimal
REAL	J	xs:float
DOUBLE	J	xs:double
DECFLOAT	J	kein entsprechender Typ[1]
CHAR	J	xs:string
VARCHAR	J	xs:string
CLOB	J	xs:string
GRAPHIC	J	xs:string
VARGRAPHIC	J	xs:string
DBCLOB	J	xs:string
DATE	J	xs:date
TIME (ohne Zeitzone)	J	xs:time
TIMESTAMP (ohne Zeitzone)	J	xs:dateTime[2]
BLOB	J	xs:base64Binary
CHAR FOR BIT DATA	N	nicht umsetzbar
VARCHAR FOR BIT DATA	J	xs:base64Binary
Einzigartiger Datentyp		Verwenden Sie beim Quellentyp des einzigartigen Datentyps diese Tabelle.
Zeile, Verweis, strukturierter oder abstrakter Datentyp, andere	N	nicht umsetzbar

Tabelle 12. Unterstützte Umsetzungen von XML-Werten in Nicht-XML-Werte (Forts.)

	Quellendatentyp	
Zieldatentyp	**XML**	**Entsprechender XQuery-Zieltyp**

Hinweise

[1] DB2 unterstützt XML Schema 1.0, was keinen übereinstimmenden XML-Schematyp für DECFLOAT bereitstellt. Der Schritt zur XQuery-Umsetzung von XMLCAST wird wie folgt verarbeitet:

- Wenn der Quellenwert einen numerischer XML-Schematyp hat, verwenden Sie diesen numerischen Typ.
- Wenn der Quellenwert den XML-Schematyp xs:boolean hat, verwenden Sie xs:double.
- Verwenden Sie andernfalls xs:string, und suchen Sie dabei zusätzlich nach einem gültigen numerischen Format.

[2] Die maximale Genauigkeit von Sekundenbruchteilen von xs:dateTime ist 6. Der Quellendatentyp TIMESTAMP unterstützt eine Zeitmarkengenauigkeit von 0 bis 12. Ist der Wert für die Zeitmarkengenauigkeit eines TIMESTAMP-Zieldatentyps kleiner als 6, wird der Wert beim Umsetzen von xs:dateTime abgeschnitten. Überschreitet die Zeitmarkengenauigkeit eines TIMESTAMP-Zieldatentyps den Wert 6, wird der Wert beim Umsetzen von xs:dateTime mit Nullen aufgefüllt.

In den folgenden Einschränkungsfällen wird ein durch Einschränkung abgeleiteter XML-Schemadatentyp effektiv als Zieldatentyp bei der XQuery-Umsetzung verwendet.

- XML-Werte, die in den Datentyp 'string' konvertiert werden sollen, müssen mit Ausnahme von CHAR und VARCHAR die Längenbegrenzungen solcher DB2-Typen einhalten, ohne irgendein Zeichen der Byte abzuschneiden. Für den abgeleiteten XML-Schematyp wird ein Name verwendet, der aus dem Namen des SQL-Typs in Großbuchstaben, einem Unterstreichungszeichen und der maximalen Länge der Zeichenfolge besteht. Beispiel: CLOB_1048576, wenn XMLCAST den Zieldatentyp CLOB(1M) hat.

  Falls ein XML-Wert in den Typ CHAR oder VARCHAR konvertiert wird und dieser zu klein ist, um alle Daten aufzunehmen, dann werden die Daten abgeschnitten, damit sie in den angegebenen Datentyp passen und es wird kein Fehler zurückgegeben. Falls belegte Zeichen abgeschnitten werden, wird eine Warnung (SQLSTATE 01004) zurückgegeben. Falls das Abschneiden der Werte zu einem Abschneiden von Mehrbytezeichen führt, wird das gesamte Mehrbytezeichen entfernt. Daher erzeugt das Abschneiden in einigen Fällen kürzere Zeichenfolgen als erwartet. Das Zeichen ñ wird zum Beispiel in UTF-8 durch zwei Byte dargestellt: ‘C3 B1'. Wenn dieses Zeichen als VARCHAR(1) umgesetzt wird, dann führt das Abschneiden von ‘C3 B1' um 1 Byte dazu, dass das abgeschnittene Zeichen ‘C3' verbleibt. Dieses abgeschnittene Zeichen 'C3' wird ebenfalls entfernt, so dass das Endergebnis eine leere Zeichenfolge ist.
- XML-Werte, die in DECIMAL-Werte konvertiert werden sollen, dürfen nicht mehr als (*genauigkeit - skala*) Stellen vor dem Dezimalzeichen aufweisen. Überzählige Stellen außerhalb der Skala und nach dem Dezimalzeichen werden abgeschnitten. Für den abgeleiteten XML-Schematyp wird der Name DECIMAL-*_genauigkeit_skala* verwendet. Dabei steht *genauigkeit* für die Genauigkeit des SQL-Zieldatentyps und *skala* für die Skala des SQL-Zieldatentyps. Beispiel: DECIMAL_9_2, wenn XMLCAST den Zieldatentyp DECIMAL(9,2) hat.
- XML-Werte, die in TIME-Werte konvertiert werden sollen, dürfen nach dem Dezimalzeichen keine Komponente für Sekunden mit Stellen ungleich null enthalten. Für den abgeleiteten XML-Schematyp wird der Name TIME verwendet.

Der Name des abgeleiteten XML-Schematyps wird in einer Nachricht nur angezeigt, wenn ein XML-Wert eine dieser Einschränkungen nicht erfüllt. Der Name dieses Typs erleichtert das Verständnis für die Fehlernachricht. Er entspricht keinem definierten XQuery-Typ. Wenn der Eingabewert nicht mit dem Basistyp des abgeleiteten XML-Schematyps (dem entsprechenden XQuery-Zieltyp) übereinstimmt, wird in der Fehlernachricht möglicherweise dieser Typ angezeigt. Da dieses Namensformat des abgeleiteten XML-Schematyps möglicherweise geändert wird, sollte es nicht als Programmierschnittstelle verwendet werden.

Bevor die XQuery-Umsetzung einen XML-Wert verarbeitet, wird jeder Dokumentknoten in der Sequenz entfernt, und jedes direkt untergeordnete Element des entfernten Dokumentknotens wird ein Element in der Sequenz. Wenn der Dokumentknoten mehrere direkt untergeordnete Knoten besitzt, enthält die überarbeitete Sequenz mehr Elemente als die ursprüngliche Sequenz. Anschließend wird der XML-Wert ohne Dokumentknoten mit der XQuery-Funktion fn:data atomisiert und der Wert der entstandenen atomisierten Sequenz in der XQuery-Umsetzung verwendet. Wenn der Wert der atomisierten Sequenz eine leere Sequenz ist, wird von der Umsetzung ohne weitere Verarbeitung ein Nullwert zurückgegeben. Wenn im Wert der atomisierten Sequenz mehrere Elemente vorhanden sind, wird ein Fehler zurückgegeben (SQLSTATE 10507).

Wenn der Zieltyp von XMLCAST der SQL-Datentyp DATE, TIME oder TIMESTAMP ist, wird der aus der XQuery-Umsetzung resultierende XML-Wert ebenfalls in UTC angepasst, und die Zeitzonenkomponente des Werts wird entfernt.

Beim Konvertieren des entsprechenden Werts des XQuery-Zieltyps in den SQL-Zieltyp werden binäre XML-Datentypen wie xs:base64Binary oder xs:hexBinary aus dem Zeichenformat in echte Binärdaten konvertiert.

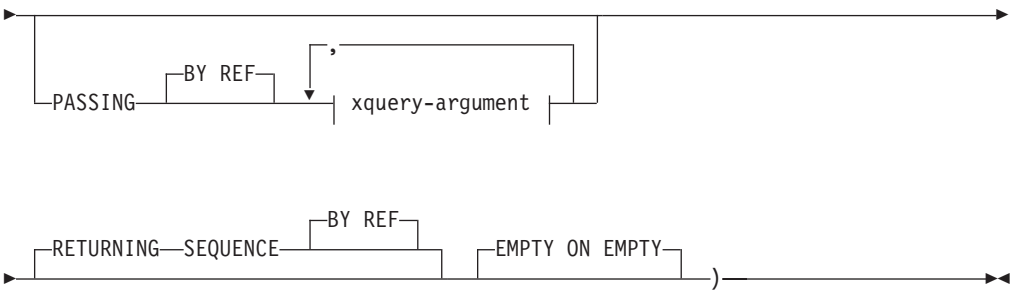
Beim Umsetzen eines xs:double- oder xs:float-Werts INF, -INF oder NaN (mit XMLCAST) in einen SQL-Wert DOUBLE oder REAL wird ein Fehler zurückgegeben (SQLSTATE 22003). Ein Wert von -0 für xs:double oder xs:float wird in +0 konvertiert.

Der Zieltyp kann ein benutzerdefinierter einzigartiger Datentyp sein, wenn der Quellenoperand kein benutzerdefinierter einzigartiger Datentyp ist. In diesem Fall wird der Quellenwert mit der Spezifikation XMLCAST in den Quellentyp des benutzerdefinierten einzigartigen Datentyps (d. h. des Zieltyps) umgesetzt. Anschließend wird dieser Wert mit der Spezifikation CAST in den benutzerdefinierten einzigartigen Datentyp umgesetzt.

In Nicht-Unicode-Datenbanken beinhaltet die Umsetzung aus einem XML-Wert in einen Nicht-XML-Zieltyp die Codepagekonvertierung aus einem internen UTF-8-Format in die Datenbankcodepage. Im Zuge dieser Konvertierung werden Substitutionszeichen eingefügt, wenn irgendein Codepunkt in dem XML-Wert nicht auf der Datenbankcodepage vorhanden ist.

XMLQUERY

Die Funktion XMLQUERY gibt bei der Auswertung eines XQuery-Ausdrucks einen XML-Wert zurück, wobei unter Umständen angegebene Eingabeargumente als XQuery-Variablen verwendet werden.



xquery-argument:

Anmerkungen:

1 Der Datentyp des Ausdrucks darf nicht DECFLOAT sein.

Das Schema heißt SYSIBM. Der Funktionsname kann nicht als qualifizierter Name angegeben werden.

xquery-ausdruckskonstante
Gibt eine SQL-Zeichenfolgekonstante an, die als ein XQuery-Ausdruck interpretiert wird, der die Syntax einer unterstützten XQuery-Sprache verwendet. Die Zeichenfolge der Konstanten wird vor der syntaktischen Analyse als XQuery-Anweisung in UTF-8 konvertiert. Der XQuery-Ausdruck wird mit einer optionalen Gruppe von XML-Eingabewerten ausgeführt und gibt eine Ausgabesequenz zurück, die auch als Wert des Ausdrucks XMLQUERY zurückgegeben wird. Der Wert für *xquery-ausdruckskonstante* darf keine leere Zeichenfolge oder eine Zeichenfolge mit Leerzeichen sein (SQLSTATE 10505).

PASSING
Gibt Eingabewerte und das Verfahren für ihre Übergabe an den mit *xquery-ausdruckskonstante* angegebenen XQuery-Ausdruck an. Standardmäßig wird jeder eindeutige Spaltenname, der sich im Geltungsbereich des Funktionsaufrufs befindet, implizit an den XQuery-Ausdruck übergeben, wobei der Name der Spalte als Variablenname verwendet wird. Wenn eine Kennung (*kennung*) in einem angegebenen XQuery-Argument mit einem gültigen Spaltennamen übereinstimmt, wird das explizite XQuery-Argument an den XQuery-Ausdruck übergeben und überschreibt diese implizite Spalte.

BY REF
Gibt an, dass die Werte bei allen Vorkommen von *xquery-variablenausdruck* des Datentyps XML und bei dem Rückgabewert standardmäßig per Verweis übergeben werden. Bei der Übergabe der XML-Werte per Verweis verwendet die XQuery-Auswertung die Eingabeknotenbaumstrukturen (sofern vorhanden) direkt aus den angegebenen Eingabeausdrücken, wobei alle Eigenschaften beibehalten werden, einschließlich der ursprünglichen Knotenidentitäten und der Dokumentreihenfolge. Wenn zwei Argumente denselben XML-Wert übergeben, verweisen Vergleiche der Knotenidentitäten und der Dokumentreihenfolge, die bestimmte zwischen den beiden Eingabeargumenten enthaltene Knoten einbeziehen, möglicherweise auf Knoten derselben XML-Knotenbaumstruktur.

Diese Klausel hat keinen Einfluss auf das Übergabeverfahren von Nicht-XML-Werten. Nicht-XML-Werte erstellen während der Umsetzung in XML eine neue Kopie des jeweiligen Werts.

xquery-argument
Gibt ein Argument an, das an den mit *xquery-ausdruckskonstante* angegebenen XQuery-Ausdruck übergeben werden soll. Ein Argument gibt einen Wert und das Verfahren für dessen Übergabe an. Das Argument enthält einen SQL-Ausdruck, der ausgewertet wird.

- Wenn der resultierende Wert den Typ XML aufweist, wird er zu einem XML-Eingabewert (*xml-eingabewert*). Ein XML-Nullwert wird in eine leere XML-Sequenz konvertiert.
- Wenn der resultierende Wert nicht den Typ XML aufweist, muss er in den XML-Datentyp umsetzbar sein. Ein Nullwert wird in eine leere XML-Sequenz konvertiert. Der konvertierte Wert wird zu einem XML-Eingabewert (*xml-eingabewert*).

Bei der Auswertung von *xquery-ausdruckskonstante* wird eine XQuery-Variable mit einem Wert, der gleich *xml-eingabewert* ist, sowie mit einem durch die Klausel AS angegebenen Namen dargestellt.

xquery-variablenausdruck
Gibt einen SQL-Ausdruck an, dessen Wert während der Ausführung für den mit *xquery-ausdruckskonstante* angegebenen XQuery-Ausdruck verfügbar ist. Der Ausdruck darf weder einen Sequenzverweis (SQLSTATE 428F9) noch eine OLAP-Funktion (SQLSTATE 42903) enthalten. Der Datentyp des Ausdrucks darf nicht DECFLOAT sein.

AS *kennung*
Gibt an, dass der von *xquery-variablenausdruck* generierte Wert als XQuery-Variable an *xquery-ausdruckskonstante* übergeben wird. Der Variablenname lautet *kennung*. Das führende Dollarzeichen ($), das Variablennamen in der XQuery-Sprache vorangestellt wird, ist in *kennung* nicht enthalten. Die Kennung muss ein gültiger XQuery-Variablenname sein und ist auf einen XML-Namen ohne Doppelpunkte (NCName) beschränkt (SQLSTATE 42634). Die Kennung darf eine Höchstlänge von 128 Byte nicht überschreiten. Zwei Argumente in derselben Klausel PASSING dürfen nicht dieselbe Kennung verwenden (SQLSTATE 42711).

BY REF
Gibt an, dass ein XML-Eingabewert per Verweis übergeben werden soll. Bei der Übergabe der XML-Werte per Verweis verwendet die XQuery-Auswertung die Eingabeknotenbaumstrukturen (sofern vorhanden) direkt aus den angegebenen Eingabeausdrücken, wobei alle Eigenschaften beibehalten werden, einschließlich der ursprünglichen Knotenidentitäten und der Dokumentreihenfolge. Wenn zwei Argumente denselben XML-Wert übergeben, verweisen Vergleiche der Knotenidentitäten und der Dokumentreihenfolge, die bestimmte zwischen den beiden Eingabeargumenten enthaltene Knoten einbeziehen, möglicherweise auf Knoten derselben XML-Knotenbaumstruktur. Wenn BY REF nicht im Anschluss an *xquery-variablenausdruck* angegeben wird, werden die XML-Argumente entsprechend dem Standardübergabeverfahren übergeben, das mit der Syntax nach dem Schlüsselwort PASSING bereitgestellt wird. Diese Option kann für Nicht-XML-Werte nicht angegeben werden. Bei der Übergabe eines Nicht-XML-Werts wird der Wert in XML konvertiert. Dabei wird eine Kopie erstellt.

RETURNING SEQUENCE
Gibt an, dass der Ausdruck XMLQUERY eine Sequenz zurückgibt.

BY REF
Gibt an, dass das Ergebnis des XQuery-Ausdrucks per Verweis zurückgegeben wird. Wenn dieser Wert Knoten enthält, erhalten alle Ausdrücke, die den Rückgabewert des XQuery-Ausdrucks verwenden, die Knotenverweise direkt, wobei alle Knoteneigenschaften beibehalten werden, einschließlich der ursprünglichen Knotenidentitäten und der Dokumentreihenfolge. Die referenzierten Knoten bleiben mit ihrer jeweiligen Knotenbaumstruktur verbunden. Wenn die Klausel BY REF nicht angegeben wird, PASSING jedoch angegeben wird, erfolgt die Übergabe entsprechend dem Standardverfahren. Wenn weder BY REF noch PASSING angegeben wird, ist BY REF das Standardrückgabeverfahren.

EMPTY ON EMPTY
Gibt an, dass ein leeres Sequenzergebnis bei der Verarbeitung des XQuery-Ausdrucks als leere Sequenz zurückgegeben wird.

Der Datentyp des Ergebnisses ist XML. Es darf nicht Null sein.

Wenn die Auswertung des XQuery-Ausdrucks zu einem Fehler führt, gibt die Funktion XMLQUERY den XQuery-Fehler (SQLSTATE-Klasse '10') zurück.

Anmerkungen

- **Einschränkungen der Verwendung von XMLQUERY**: Die Funktion XMLQUERY darf Folgendes nicht sein:
  - Teil der Klausel ON, der einem JOIN-Operator oder einer MERGE-Anweisung zugeordnet ist (SQLSTATE 42972)
  - Teil der Klausel GENERATE KEY USING oder RANGE THROUGH in der Anweisung CREATE INDEX EXTENSION (SQLSTATE 428E3)
  - Teil der Klausel FILTER USING in der Anweisung CREATE FUNCTION (extern, skalar) oder der Klausel FILTER USING in der Anweisung CREATE INDEX EXTENSION (SQLSTATE 428E4)
  - Teil einer Prüfung auf Integritätsbedingung oder eines Ausdrucks zur Spaltengenerierung (SQLSTATE 42621)
  - Teil einer Gruppierung nach Klausel (SQLSTATE 42822)
  - Teil eines Arguments für eine Spaltenfunktion (SQLSTATE 42607)
- **XMLQUERY als Unterabfrage**: Ein XMLQUERY-Ausdruck, der als Unterabfrage dient, kann durch Anweisungen eingeschränkt werden, die Unterabfragen einschränken.

XMLTABLE - Funktionsübersicht

Die SQL-Tabellenfunktion XMLTABLE gibt eine Tabelle aus der Auswertung von XQuery-Ausdrücken zurück. XQuery-Ausdrücke geben Werte normalerweise als Sequenz zurück. Mit der Funktion XMLTABLE haben Sie jedoch die Möglichkeit, XQuery-Ausdrücke auszuführen und Werte stattdessen in Form einer Tabelle zurückzugeben.

Die Tabelle, die zurückgegeben wird, kann Spalten eines beliebigen SQL-Datentyps, einschließlich XML, enthalten.

Ebenso wie bei der Funktion XMLQUERY können Sie an den in der Funktion XMLTABLE angegebenen XQuery-Ausdruck Variablen übergeben. Das Ergebnis des XQuery-Ausdrucks wird zur Generierung der Spaltenwerte der Ergebnistabelle

verwendet. Die Struktur der Ergebnistabelle wird durch die Klausel COLUMNS der Funktion XMLTABLE definiert. In dieser Klausel definieren Sie die Merkmale der Spalte, indem Sie den Spaltennamen, den Datentyp und die Art der Generierung des Spaltenwerts angeben. Eine vereinfachte Syntax steht ebenfalls zur Verfügung, mit der Spaltennamen ohne explizite Angabe des Namens übergeben werden können. Siehe „Einfache Übergabe von Spaltennamen mit XMLEXISTS, XMLQUERY oder XMLTABLE" auf Seite 117.

Der Spaltenwert der Ergebnistabelle kann generiert werden, indem ein XQuery-Ausdruck in der Klausel PATH der Funktion XMLTABLE angegeben wird. Wenn für die PATH-Klausel kein XQuery-Ausdruck angegeben wird, wird der Spaltenname als XQuery-Ausdruck zur Generierung des Spaltenwerts verwendet, und das zuvor in XMLTABLE angegebene Ergebnis des XQuery-Ausdrucks wird bei der Generierung des Spaltenwerts zum externen Kontextelement. Darüber hinaus kann eine optionale DEFAULT-Klausel angegeben werden, um einen Standardwert für die Spalte für den Fall bereitzustellen, dass der XQuery-Ausdruck der PATH-Klausel, der den Spaltenwert generiert, eine leere Sequenz zurückgibt.

Beispiel: Die folgende Anweisung SELECT referenziert die Spalte INFO der Tabelle CUSTOMER im XQuery-Ausdruck in der Funktion XMLTABLE.

```
SELECT X.*
FROM CUSTOMER C, XMLTABLE ('$INFO/customerinfo'
               COLUMNS
               CUSTNAME CHAR(30) PATH 'name',
               PHONENUM XML PATH 'phone')
     as X
WHERE C.CID < 1003
```

Falls der Spaltentyp in der Ergebnistabelle nicht XML ist und das Ergebnis des XQuery-Ausdrucks, der den Wert der Spalte definiert, keine leere Sequenz ist, wird der XML-Wert implizit durch XMLCAST in einen Wert des Zieldatentyps umgewandelt.

Die Funktion XMLTABLE bietet die optionale Möglichkeit, Namensbereiche (namespaces) zu deklarieren. Wenn Sie Namensbereiche mithilfe der Deklaration XMLNAMESPACES angeben, gelten diese Namensbereichsbindungen für alle XQuery-Ausdrücke im Aufruf der Funktion XMLTABLE. Wenn Sie Namensbereichsbindungen ohne die Deklaration XMLNAMESPACES deklarieren, gelten die Bindungen nur für den XQuery-Zeilenausdruck, der auf die Namensbereichsdeklaration folgt.

Vorteile von XMLTABLE

Die Rückgabe einer Tabelle anstelle einer Sequenz ermöglicht die Ausführung der folgenden Operationen innerhalb eines SQL-Abfragekontexts:

- Iterieren durch die Ergebnisse eines XQuery-Ausdrucks in einem SQL-Fullselect-Anweisung

  In der folgenden Abfrage iteriert die SQL-Fullselect-Anweisung zum Beispiel durch die Tabelle, die aus der Ausführung des XQuery-Ausdrucks "db2-fn:xmlcolumn("CUSTOMER.INFO")/customerinfo" in der Funktion XMLTABLE resultiert.

  ```
    SELECT X.*
  FROM XMLTABLE ('db2-fn:xmlcolumn("CUSTOMER.INFO")/customerinfo'
                                 COLUMNS "CUSTNAME" CHAR(30) PATH 'name',
                                                    "PHONENUM" XML PATH 'phone')
            as X
  ```

- Einfügen von Werten aus gespeicherten XML-Dokumenten in Tabellen (siehe Beispiel für XMLTABLE: Einfügen von Werten)
- Sortieren von Werten aus einem XML-Dokument

  Zum Beispiel werden in der folgenden Abfrage Ergebnisse nach den Kundennamen sortiert, die in XML-Dokumenten in der Spalte INFO der Tabelle CUSTOMER gespeichert sind.

```
  SELECT X.*
FROM XMLTABLE ('db2-fn:xmlcolumn("CUSTOMER.INFO")/customerinfo'
                              COLUMNS "CUSTNAME" CHAR(30) PATH 'name',
                                                 "PHONENUM" XML PATH 'phone')
     as X
ORDER BY X.CUSTNAME
```

- Speichern einiger XML-Werte als relationale Daten und einiger XML-Werte als XML-Daten (siehe Beispiel für XMLTABLE: Einfügen von Werten)

Wichtig: Folgendes gilt, wenn der in der PATH-Option der Funktion XMLTABLE angegebene XQuery-Ausdruck eines der folgenden Ergebnisse zurückgibt:

- Eine Sequenz aus mehr als einem Element: In diesem Fall muss der Datentyp der Spalte XML sein. Wenn Sie aus der Funktion XMLTABLE zurückgegebene Werte in XML-Spalten einfügen, müssen Sie sicherstellen, dass die eingefügten Werte korrekt formatierte XML-Dokumente sind. Ein Beispiel für die Behandlung von Sequenzen, die mehr als ein Element zurückgeben, finden Sie im Beispiel für XMLTABLE zum Einfügen von Werten.
- Eine leere Sequenz: In diesem Fall wird ein NULL-Wert für den entsprechenden Spaltenwert zurückgegeben.

Beispiel: Einfügen von Werten, die von XMLTABLE zurückgegeben werden

Die SQL-Tabellenfunktion XMLTABLE kann zum Abrufen von Werten aus gespeicherten XML-Dokumenten verwendet werden, die anschließend in eine Tabelle eingefügt werden können.

Dieses Verfahren ist eine einfache Form der Dekomposition, wobei Dekomposition den Prozess des Speicherns von Fragmenten eines XML-Dokuments in Spalten relationaler Tabellen bezeichnet. (Ein etwas allgemeinerer Typ von Dekomposition wird durch die Funktionalität der Dekomposition mithilfe eines mit Annotationen versehenen XML-Schemas zur Verfügung gestellt. Bei der Dekomposition mithilfe eines mit Annotationen versehenen Schemas können Sie mehrere XML-Dokumente gleichzeitig in mehrere Tabellen zerlegen.)

Nehmen Sie zum Beispiel an, dass die beiden folgenden XML-Dokumente in einer Tabelle mit dem Namen CUSTOMER gespeichert wären:

```
<customerinfo Cid="1001">
    <name>Kathy Smith</name>
  <addr country="Canada">
    <street>25 EastCreek</street>
        <city>Markham</city>
    <prov-state>Ontario</prov-state>
        <pcode-zip>N9C 3T6</pcode-zip>
  </addr>
    <phone type="work">905-555-7258</phone>
</customerinfo>

<customerinfo Cid="1003">
    <name>Robert Shoemaker</name>
    <addr country="Canada">
        <street>1596 Baseline</street>
```

```
        <city>Aurora</city>
    <prov-state>Ontario</prov-state>
        <pcode-zip>N8X 7F8</pcode-zip>
  </addr>
    <phone type="work">905-555-7258</phone>
    <phone type="home">416-555-2937</phone>
    <phone type="cell">905-555-8743</phone>
    <phone type="cottage">613-555-3278</phone>
</customerinfo>
```

Nun möchten Sie Werte aus diesen Dokumenten in eine Tabelle einfügen, die wie folgt definiert ist:

```
CREATE TABLE CUSTPHONE (custname char(30), numbers XML)
```

In diesem Fall würde die folgende INSERT-Anweisung die Tabelle CUSTPHONE unter Verwendung der Funktion XMLTABLE mit Werten aus den XML-Dokumenten füllen:

```
INSERT INTO CUSTPHONE
  SELECT X.*
  FROM XMLTABLE ('db2-fn:xmlcolumn("CUSTOMER.INFO")/customerinfo'
                 COLUMNS
                   "CUSTNAME" CHAR(30) PATH 'name',
                   "PHONENUM" XML PATH 'document{<allphones>{phone}</allphones>}'
                ) as X
```

Beachten Sie, dass die XQuery-Knotenkonstruktorfunktion "document{<allphones>{phone}</allphones>}" im PATH-Ausdruck für die Spalte PHONENUM in der Funktion XMLTABLE angegeben wurde. Der document-Konstruktor ist erforderlich, weil Werte, die in XML-Spalten (in diesem Fall die Spalte NUMBERS) eingefügt werden, korrekt formatierte XML-Dokumente sein müssen. In diesem Beispiel werden alle Elemente <phone> für das Dokument <customerinfo> mit Cid="1003" in einer einzigen Sequenz zurückgegeben, die vier Elemente enthält:

```
{<phone type="work">905-555-7258</phone>,<phone type="home">416-555-2937</phone>,
<phone type="cell">905-555-8743</phone>, <phone type="cottage">613-555-3278</phone>}
```

Diese Sequenz für sich genommen ist kein korrekt formatiertes XML-Dokument, sodass sie nicht in die XML-Spalte NUMBERS eingefügt werden kann. Um sicherzustellen, dass die phone-Werte erfolgreich eingefügt werden, werden alle Elemente der Sequenz zu einem korrekt formatierten Dokument konstruiert.

Die Ergebnistabelle sieht folgendermaßen aus (die Ausgabe wurde aus Gründen der Übersichtlichkeit formatiert):

Tabelle 13. Ergebnistabelle

CUSTNAME	NUMBERS
Kathy Smith	<allphones> <phone type="work">905-555-7258</phone> </allphones>
Robert Shoemaker	<allphones> <phone type="work">905-555-7258</phone> <phone type="home">416-555-2937</phone> <phone type="cell">905-555-8743</phone> <phone type="cottage">613-555-3278</phone> </allphones>

Beispiel: Zurückgeben von jeweils einer Zeile für jedes Vorkommen eines Elements mit XMLTABLE

Wenn Ihre XML-Dokumente mehrere Vorkommen eines Elements enthalten und Sie eine Zeile für jedes Vorkommen dieses Elements generieren möchten, können Sie dieses Ziel mithilfe der Funktion XMLTABLE erreichen.

Nehmen Sie zum Beispiel an, dass die beiden folgenden XML-Dokumente in einer Tabelle mit dem Namen CUSTOMER gespeichert wären:

```
<customerinfo Cid="1001">
    <name>Kathy Smith</name>
  <addr country="Canada">
    <street>25 EastCreek</street>
        <city>Markham</city>
    <prov-state>Ontario</prov-state>
        <pcode-zip>N9C 3T6</pcode-zip>
  </addr>
    <phone type="work">905-555-7258</phone>
</customerinfo>

<customerinfo Cid="1003">
    <name>Robert Shoemaker</name>
    <addr country="Canada">
        <street>1596 Baseline</street>
        <city>Aurora</city>
    <prov-state>Ontario</prov-state>
        <pcode-zip>N8X 7F8</pcode-zip>
  </addr>
    <phone type="work">905-555-7258</phone>
    <phone type="home">416-555-2937</phone>
    <phone type="cell">905-555-8743</phone>
    <phone type="cottage">613-555-3278</phone>
</customerinfo>
```

Zur Erstellung einer Tabelle, in der jeder <phone>-Wert in einer separaten Zeile gespeichert wird, verwenden Sie die Funktion XMLTABLE wie folgt:

```
  SELECT X.*
FROM CUSTOMER C, XMLTABLE ('$cust/customerinfo/phone' PASSING C.INFO as "cust"
                           COLUMNS "CUSTNAME" CHAR(30) PATH '../name',
                           "PHONETYPE" CHAR(30) PATH '@type',
                                                "PHONENUM" CHAR(15) PATH '.'
                                              ) as X
```

Diese Abfrage liefert das folgende Ergebnis für die beiden XML-Dokumente:

Tabelle 14. Ergebnistabelle

CUSTNAME	PHONETYPE	PHONENUM
Kathy Smith	work	905-555-7258
Robert Shoemaker	work	905-555-7258
Robert Shoemaker	home	416-555-2937
Robert Shoemaker	cell	905-555-8743
Robert Shoemaker	cottage	613-555-3278

Beachten Sie, dass hier jedes Element <phone> für das XML-Dokument mit dem Namen "Robert Shoemaker" in einer separaten Zeile zurückgegeben wird.

Für die gleichen Dokumente können Sie die Elemente <phone> auch wie folgt als XML-Daten extrahieren:

```
  SELECT X.*
FROM CUSTOMER C, XMLTABLE ('$cust/customerinfo/phone' PASSING C.INFO as "cust"
                           COLUMNS "CUSTNAME" CHAR(30) PATH '../name',
                           "PHONETYPE" CHAR(30) PATH '@type',
                                                "PHONENUM" XML PATH '.'
                                              ) as X
```

Diese Abfrage liefert das folgende Ergebnis für die beiden XML-Dokumente (die Ausgabe wurde aus Gründen der Übersichtlichkeit formatiert):

Tabelle 15. Ergebnistabelle

CUSTNAME	PHONETYPE	PHONENUM
Kathy Smith	work	<phone type="work">416-555-1358</phone>
Robert Shoemaker	work	<phone type="work">905-555-7258</phone>
Robert Shoemaker	home	<phone type="home">416-555-2937</phone>
Robert Shoemaker	cell	<phone type="cell">905-555-8743</phone>
Robert Shoemaker	cottage	<phone type="cottage">613-555-3278</phone>

Beispiel: Verarbeitung von Elementen aus mehreren Baumstrukturen in einem XML-Dokument mit XMLTABLE

In einem XML-Dokument, das mehrere Hierarchien bzw. Baumstrukturen enthält, kann ein Element in einer Baumstruktur des Dokuments eine Beziehung mit einem Element in einer anderen Baumstruktur des Dokuments haben. Mithilfe von XPath-Ausdrücken können Sie die zusammengehörigen Elemente in XML-Dokumenten verarbeiten.

Im folgenden Beispiel wird auf Basis der in einem XML-Dokument enthaltenen Informationen eine Tabelle generiert, die zum Umzug von Möbeln und Einrichtungsgegenständen in ein neues Gebäude dient. Das XML-Dokument enthält Angaben zum Standort der Gegenstände in dem neuen Gebäude.

Mit der folgenden Anweisung wird die Tabelle MOVE erstellt, in der das XML-Dokument gespeichert wird:

```
CREATE TABLE MOVE (ID BIGINT NOT NULL PRIMARY KEY, MOVEINFO XML )
```

Die folgenden XML-Daten enthalten Angaben zu den Gegenständen des Unternehmens und zu deren Standort. Die XML-Informationen sind in zwei Baumstrukturen aufgeteilt. In der einen Baumstruktur befinden sich die Angaben zu den Gegenständen, beispielsweise deren Abteilung, Markierungsnummer und Beschreibung. Die andere Baumstruktur enthält Angaben zum Transport, beispielsweise den neuen Standort der Abteilungen und das Stockwerk der zugeordneten Büros, den öffentlich zugänglichem Bereich und den Speicherbereich.

Mit der folgenden Anweisung wird das XML-Dokument in die Tabelle MOVE eingefügt:

```
INSERT into MOVE (ID, MOVEINFO) values ( 1, '
<listing>
  <items>
   <item dept="acct" tag="12223">
       <name>laser printer</name>
```

```
        <area>common</area>
      </item>
       <item dept="recptn" tag="23665">
        <name>monitor, CRT</name>
        <area>storage</area>
      </item>
       <item dept="acct" tag="42345">
        <name>CPU, desktop</name>
        <area>office</area>
      </item>
       <item dept="recptn" tag="33301">
        <name>monitor, LCD</name>
        <area>office</area>
      </item>
       <item dept="mfg" tag="10002">
        <name>cabinet, 3 dwr</name>
        <area>office</area>
      </item>
       <item dept="acct" tag="65436">
        <name>table, round 1m</name>
        <area>storage</area>
      </item>
      </items>

      <locations>
       <building dept="recptn" >
          <wing>main</wing>
          <floor area="storage">1</floor>
          <floor area="common">1</floor>
          <floor area="office">2</floor>
       </building>

       <building dept="mfg" >
          <wing>east</wing>
          <floor area="storage">1</floor>
          <floor area="common">2</floor>
          <floor area="office">2</floor>
       </building>

       <building dept="acct" >
          <wing>west</wing>
          <floor area="storage">2</floor>
          <floor area="common">1</floor>
          <floor area="office">2</floor>
       </building>
      </locations>

    </listing>
    ')
```

Die folgende Anweisung SELECT kombiniert die Angaben zu den Gegenständen mit den Informationen zum Standort und erstellt eine Tabelle, in der die Angaben zu den Gegenständen, der Abteilung und dem neuen Standort aufgeführt sind.

Der Schlüssel für die Zuordnung liegt darin, die Angaben zu den Gegenständen mithilfe der relativen XPath-Achse `$x/../../` mit den Informationen zum Standort abzugleichen.

```
SELECT T.*
   from MOVE,
  XMLTABLE( '$doc/listing/items/item' PASSING MOVE.MOVEINFO AS "doc"
  COLUMNS
    ITEM_ID  VARCHAR(10) PATH 'let $x := . return $x/@tag' ,
    DESC     VARCHAR(20) PATH 'let $x := . return $x/name' ,
    DEPT     VARCHAR(15) PATH 'let $x := . return $x/@dept' ,
```

```
  WING     VARCHAR(10) PATH 'let $x := . return $x/../../locations/building[@dept = $x/@dept]/wing',
  FLOOR    VARCHAR(10) PATH 'let $x := . return $x/../../locations/building/floor[@area = $x/area
                            and ../@dept = $x/@dept ]'
)as T
```

Bei der Ausführung für die Beispieldaten gibt die Anweisung folgende Daten zurück:

```
ITEM_ID    DESC                DEPT            WING       FLOOR
---------- ------------------- --------------- ---------- ----------
12223      printer, laser      acct            west       1
23665      monitor, CRT        recptn          main       1
42345      CPU, desktop        acct            west       2
33301      monitor, LCD        recptn          main       2
10002      cabinet, 3 dwr      mfg             east       2
65436      table, round 1m     acct            west       2
```

Beispiel: Verarbeitung hierarchischer Daten mit XMLTABLE

XML-Dokumente können eine Hierarchie von Daten mit einer variablen Anzahl von Verschachtelungsebenen enthalten. Mithilfe von XPath-Ausdrücken können Sie die hierarchischen Daten verarbeiten.

Im folgenden Beispiel wird eine Liste mit Teilen eines Computers und mit deren jeweiligen übergeordneten Komponenten erstellt. Die Informationen basieren auf einem XML-Dokument, das die Komponenten des Computers und die Beziehungen zwischen den Komponenten enthält.

Mit der folgenden Anweisung wird die Tabelle BOMLIST erstellt, in der das XML-Dokument gespeichert wird:

```
CREATE TABLE BOMLIST (Cid BIGINT NOT NULL PRIMARY KEY, ITEMS XML )
```

Das XML-Dokument enthält eine Liste der Komponenten und Unterkomponenten. Die Komponenten in der Liste sind in einer Hierarchie untereinander verbunden, die die Unterkomponenten einer Komponente beschreibt. Wenn eine Komponente Unterkomponenten enthält, werden die einzelnen Unterkomponenten als Unterelemente der Komponente aufgeführt.

Mit der folgenden Anweisung wird das XML-Dokument in die Tabelle BOMLIST eingefügt:

```
CREATE TABLE BOMLIST (Cid BIGINT NOT NULL PRIMARY KEY, ITEMS XML )
insert into BOMLIST (Cid, ITEMS) values ( 1, '
<item desc="computersystem" model="L1234123">
   <part desc="computer" partnum="5423452345">
      <part desc="motherboard" partnum="5423452345">
         <part desc="CPU" partnum="6109486697">
            <part desc="register" partnum="6109486697"/>
        </part>
         <part desc="memory" partnum="545454232">
            <part desc="transistor" partnum="6109486697"/>
        </part>
      </part>

      <part desc="diskdrive" partnum="6345634563456">
         <part desc="spindlemotor" partnum="191986123"/>
      </part>
      <part desc="powersupply" partnum="098765343">
         <part desc="powercord" partnum="191986123"/>
      </part>
    </part>

   <part desc="monitor" partnum="898234234">
```

```
      <part desc="cathoderaytube" partnum="191986123"/>
    </part>

   <part desc="keyboard" partnum="191986123">
      <part desc="keycaps" partnum="191986123"/>
    </part>

   <part desc="mouse" partnum="98798734">
      <part desc="mouseball" partnum="98798734"/>
    </part>
</item>
')
```

Die folgende Anweisung SELECT navigiert durch das Dokument und erstellt eine Tabelle, in der das Teil und das übergeordnete Teil aufgeführt sind.

Das Hauptmerkmal besteht in der Erstellung von Tabelle B mithilfe der Funktion XMLTABLE. Mithilfe von // in der XPath-Achse $doc//part kann durch alle Teilelemente im Elementknoten navigiert werden.

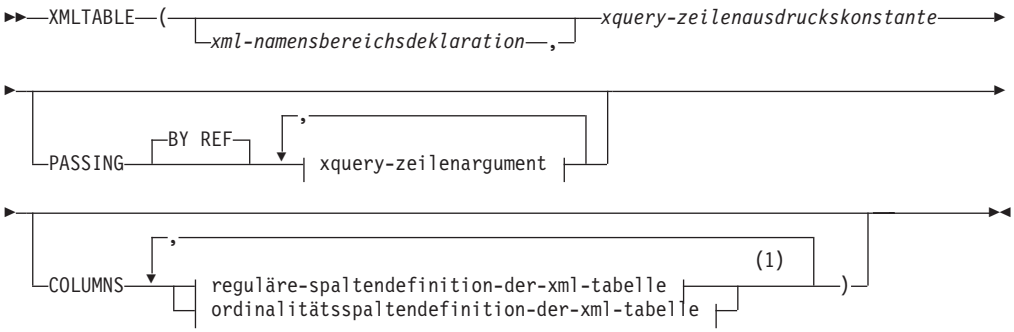
```
SELECT
  A.ITEMNAME,
  B.PART,
  B.PARENT
   FROM BOMLIST ,
   XMLTABLE('$doc/item' PASSING BOMLIST.ITEMS AS "doc"
    COLUMNS
      ITEMNAME VARCHAR(20) PATH './@desc',
      ITEM   XML            PATH '.'
   )AS A,
   XMLTABLE('$doc//part' PASSING A.ITEM AS "doc"
    COLUMNS
     PART  VARCHAR(20) PATH './@desc',
     PARENT VARCHAR(20) PATH '../@desc'
   )AS B
```

Wird die folgende Anweisung für die Daten ausgeführt, wird die folgende Tabelle angezeigt, in der das Teil und das übergeordnete Teil angezeigt werden:

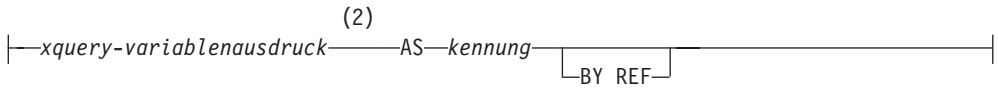
```
ITEMNAME            PART                PARENT
------------------- ------------------- -------------------
computersystem      computer            computersystem
computersystem      motherboard         computer
computersystem      CPU                 motherboard
computersystem      register            CPU
computersystem      memory              motherboard
computersystem      transistor          memory
computersystem      diskdrive           computer
computersystem      spindlemotor        diskdrive
computersystem      powersupply         computer
computersystem      powercord           powersupply
computersystem      monitor             computersystem
computersystem      cathoderaytube      monitor
computersystem      keyboard            computersystem
computersystem      keycaps             keyboard
computersystem      mouse               computersystem
computersystem      mouseball           mouse
```

XMLTABLE

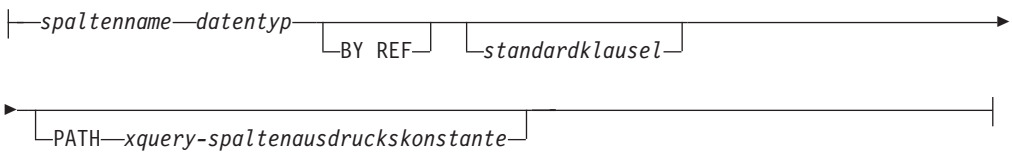
Die Funktion XMLTABLE gibt bei der Auswertung von XQuery-Ausdrücken eine Ergebnistabelle zurück, wobei unter Umständen angegebene Eingabeargumente als XQuery-Variablen verwendet werden. Jedes Element in der Ergebnissequenz des XQuery-Zeilenausdruck stellt eine Zeile der Ergebnistabelle dar.



xquery-zeilenargument:



reguläre-spaltendefinition-der-xml-tabelle:



ordinalitätsspaltendefinition-der-xml-tabelle:

Anmerkungen:

1 Die Klausel ordinalitätsspaltendefinition-der-xml-tabelle darf nicht mehrmals angegeben werden (SQLSTATE 42614).

2 Der Datentyp des Ausdrucks darf nicht DECFLOAT sein.

Das Schema heißt SYSIBM. Der Funktionsname kann nicht als qualifizierter Name angegeben werden.

xml-namensbereichsdeklaration
Gibt einen oder mehrere XML-Namensbereichsdeklarationen an, die in den statischen Kontext von *xquery-zeilenausdruckskonstante* und *xquery-spaltenausdruckskonstante* eingebunden werden. Die Gruppe der statistisch bekannten Namensbereiche für XQuery-Ausdrücke, die Argumente von XMLTABLE sind, ist die Kombination aus den zuvor ermittelten statistisch bekannten Namensbereichen und aus den in dieser Klausel angegebenen Namensbereichsdeklarationen. Der XQuery-Prolog in einem XQuery-Ausdruck überschreibt diese Namensbereiche möglicherweise.

Wenn *xml-namensbereichsdeklaration* nicht angegeben wird, gilt nur die zuvor ermittelte Gruppe der statistisch bekannten Namensbereiche für die XQuery-Ausdrücke.

xquery-zeilenausdruckskonstante
Gibt eine SQL-Zeichenfolgekonstante an, die als ein XQuery-Ausdruck interpretiert wird, der die Syntax einer unterstützten XQuery-Sprache verwendet.

Die Zeichenfolge der Konstanten wird ohne Konvertierung in die Datenbank- oder Abschnittscodepage direkt in UTF-8 konvertiert. Der XQuery-Ausdruck wird mit einer optionalen Gruppe von XML-Eingabewerten ausgeführt und gibt eine XQuery-Ausgabesequenz zurück, in der für jedes Element in der Sequenz eine Zeile generiert wird. Der Wert für *xquery-zeilenausdruckskonstante* darf keine leere Zeichenfolge oder eine Zeichenfolge mit Leerzeichen sein (SQLSTATE 10505).

PASSING
Gibt Eingabewerte und das Verfahren für ihre Übergabe an den mit *xquery-zeilenausdruckskonstante* angegebenen XQuery-Ausdruck an. Standardmäßig wird jeder eindeutige Spaltenname, der sich im Geltungsbereich des Funktionsaufrufs befindet, implizit an den XQuery-Ausdruck übergeben, wobei der Name der Spalte als Variablenname verwendet wird. Wenn eine Kennung (*kennung*) in einem angegebenen XQuery-Zeilenargument mit einem gültigen Spaltennamen übereinstimmt, wird das explizite XQuery-Zeilenargument an den XQuery-Ausdruck übergeben und überschreibt diese implizite Spalte.

BY REF
Gibt an, dass standardmäßig alle XML-Eingabeargumente per Verweis übergeben werden. Bei der Übergabe der XML-Werte per Verweis verwendet die XQuery-Auswertung die Eingabeknotenbaumstrukturen (sofern vorhanden) direkt aus den angegebenen Eingabeausdrücken, wobei alle Eigenschaften beibehalten werden, einschließlich der ursprünglichen Knotenidentitäten und der Dokumentreihenfolge. Wenn zwei Argumente denselben XML-Wert übergeben, verweisen Vergleiche der Knotenidentitäten und der Dokumentreihenfolge, die bestimmte zwischen den beiden Eingabeargumenten enthaltenen Knoten einbeziehen, möglicherweise auf Knoten derselben XML-Knotenbaumstruktur.

Diese Klausel hat keinen Einfluss auf das Übergabeverfahren von Nicht-XML-Werten. Nicht-XML-Werte erstellen während der Umsetzung in XML eine neue Kopie des jeweiligen Werts.

xquery-zeilenargument
Gibt ein Argument an, das an den mit *xquery-zeilenausdruckskonstante* angegebenen XQuery-Ausdruck übergeben werden soll. Ein Argument gibt einen Wert und das Verfahren für dessen Übergabe an. Das Argument enthält einen SQL-Ausdruck, der vor der Übergabe des Ergebnisses an den XQuery-Ausdruck ausgewertet wird.

- Wenn der resultierende Wert den Typ XML aufweist, wird er zu einem XML-Eingabewert (*xml-eingabewert*). Ein XML-Nullwert wird in eine leere XML-Sequenz konvertiert.
- Wenn der resultierende Wert nicht den Typ XML aufweist, muss er in den XML-Datentyp umsetzbar sein. Ein Nullwert wird in eine leere XML-Sequenz konvertiert. Der konvertierte Wert wird zu einem XML-Eingabewert (*xml-eingabewert*).

Bei der Auswertung von *xquery-zeilenausdruckskonstante* wird eine XQuery-Variable mit einem Wert, der gleich *xml-eingabewert* ist, sowie mit einem durch die Klausel AS angegebenen Namen dargestellt.

xquery-variablenausdruck
Gibt einen SQL-Ausdruck an, dessen Wert während der Ausführung für den mit *xquery-zeilenausdruckskonstante* angegebenen XQuery-Ausdruck verfügbar ist. Der Ausdruck darf weder einen Ausdruck NEXT

VALUE oder PREVIOUS VALUE (SQLSTATE 428F9) noch eine OLAP-Funktion (SQLSTATE 42903) enthalten. Der Datentyp des Ausdrucks darf nicht DECFLOAT sein.

AS *kennung*
Gibt an, dass der von *xquery-variablenausdruck* generierte Wert als XQuery-Variable an *xquery-zeilenausdruckskonstante* übergeben wird. Der Variablenname lautet *kennung*. Das führende Dollarzeichen ($), das Variablennamen in der XQuery-Sprache vorangestellt wird, ist in *kennung* nicht enthalten. Die Kennung muss ein gültiger XQuery-Variablenname sein und ist auf einen XML-Namen ohne Doppelpunkte (NCName) beschränkt. Die Kennung darf eine Höchstlänge von 128 Byte nicht überschreiten. Zwei Argumente in derselben Klausel PASSING dürfen nicht dieselbe Kennung verwenden (SQLSTATE 42711).

BY REF
Gibt an, dass ein XML-Eingabewert per Verweis übergeben werden soll. Bei der Übergabe der XML-Werte per Verweis verwendet die XQuery-Auswertung die Eingabeknotenbaumstrukturen (sofern vorhanden) direkt aus den angegebenen Eingabeausdrücken, wobei alle Eigenschaften beibehalten werden, einschließlich der ursprünglichen Knotenidentitäten und der Dokumentreihenfolge. Wenn zwei Argumente denselben XML-Wert übergeben, verweisen Vergleiche der Knotenidentitäten und der Dokumentreihenfolge, die bestimmte zwischen den beiden Eingabeargumenten enthaltenen Knoten einbeziehen, möglicherweise auf Knoten derselben XML-Knotenbaumstruktur. Wenn BY REF nicht im Anschluss an *xquery-variablenausdruck* angegeben wird, werden die XML-Argumente entsprechend dem Standardübergabeverfahren übergeben, das mit der Syntax nach dem Schlüsselwort PASSING bereitgestellt wird. Diese Option kann für Nicht-XML-Werte nicht angegeben werden (SQLSTATE 42636). Bei der Übergabe eines Nicht-XML-Werts wird der Wert in XML konvertiert. Dabei wird eine Kopie erstellt.

COLUMNS
Gibt die Ausgabespalten der Ergebnistabelle an. Wird diese Klausel nicht angegeben, wird per Verweis eine einzelne nicht benannte Spalte des Datentyps XML zurückgegeben, wobei der Wert auf dem Sequenzelement der Auswertung des XQuery-Ausdrucks in *xquery-zeilenausdruckskonstante* basiert (äquivalent zur Angabe von PATH '.'). Zum Verweisen auf die Ergebnisspalte muss in der auf die Funktion folgenden Korrelationsklausel (*korrelationsklausel*) ein Spaltenname (*spaltenname*) angegeben werden.

reguläre-spaltendefinition-der-xml-tabelle
Gibt die Ausgabespalten der Ergebnistabelle einschließlich der Spaltennamen, des Datentyps und des XML-Übergabeverfahrens sowie einen XQuery-Ausdruck an, mit dem der Wert aus dem Sequenzelement der Zeile extrahiert wird.

spaltenname
Gibt den Namen der Spalte in der Ergebnistabelle an. Der Name darf nicht qualifiziert sein und nicht in mehreren Spalten der Tabelle verwendet werden (SQLSTATE 42711).

datentyp
Gibt den Datentyp der Spalte an. Die Syntax und eine Beschreibung der verfügbaren Typen finden Sie unter CREATE TABLE. *datentyp* kann

in XMLTable verwendet werden, wenn eine unterstützte XMLCAST-Umsetzung aus XML in den angegebenen Datentyp (*datentyp*) vorhanden ist.

BY REF
Gibt an, dass XML-Werte für Spalten des Datentyps XML per Verweis zurückgegeben werden. Standardmäßig erfolgt die Rückgabe von XML-Werten BY REF. Bei der Rückgabe der XML-Werte per Verweis schließt der XML-Wert die Eingabeknotenbaumstrukturen (sofern vorhanden) direkt aus den Ergebniswerten ein und behält dabei alle Eigenschaften bei, einschließlich der ursprünglichen Knotenidentitäten und der Dokumentreihenfolge. Diese Option kann für Nicht-XML-Spalten nicht angegeben werden (SQLSTATE 42636). Bei der Verarbeitung einer Nicht-XML-Spalte wird der Wert aus XML konvertiert. Dabei wird eine Kopie erstellt.

standardklausel
Gibt einen Standardwert für die Spalte an. Die Syntax und eine Beschreibung von *standardklausel* finden Sie unter CREATE TABLE. Bei Ergebnisspalten von XMLTABLE wird der Standardwert angewendet, wenn bei der Verarbeitung des in *xquery-spaltenausdruckskonstante* enthaltenen XQuery-Ausdrucks eine leere Sequenz zurückgegeben wird.

PATH *xquery-spaltenausdruckskonstante*
Gibt eine SQL-Zeichenfolgekonstante an, die als ein XQuery-Ausdruck interpretiert wird, der die Syntax einer unterstützten XQuery-Sprache verwendet. Die Zeichenfolge der Konstanten wird ohne Konvertierung in die Datenbank- oder Abschnittscodepage direkt in UTF-8 konvertiert. *xquery-spaltenausdruckskonstante* gibt einen XQuery-Ausdruck an, der den Spaltenwert mit Bezug auf ein Element ermittelt, das aus einer Auswertung des XQuery-Ausdrucks in *xquery-zeilenausdruckskonstante* resultiert. Für jedes Element, das sich bei der Verarbeitung von *xquery-zeilenausdruckskonstante* als externes Kontextelement ergibt, wird *xquery-spaltenausdruckskonstante* ausgewertet und eine Ausgabesequenz zurückgegeben. Der Spaltenwert wird auf der Basis dieser Ausgabesequenz wie folgt ermittelt:

- Wenn die Ausgabesequenz null Elemente enthält, stellt *standardklausel* den Wert der Spalte bereit.
- Wenn eine leere Sequenz zurückgegeben wird, ohne dass *standardklausel* angegeben wurde, wird der Spalte ein Nullwert zugewiesen.
- Wird eine nicht leere Sequenz zurückgegeben, wird der Wert mit XMLCAST in den für die Spalte angegebenen Datentyp (*datentyp*) umgesetzt. Bei der Verarbeitung von XMLCAST könnte ein Fehler zurückgegeben werden.

Der Wert für *xquery-spaltenausdruckskonstante* darf keine leere Zeichenfolge oder eine Zeichenfolge mit Leerzeichen sein (SQLSTATE 10505). Wird diese Klausel nicht angegeben, ist der standardmäßige XQuery-Ausdruck einfach der Spaltenname (*spaltenname*).

ordinalitätsspaltendefinition-der-xml-tabelle
Gibt die Ordinalitätsspalte der Ergebnistabelle an.

spaltenname
Gibt den Namen der Spalte in der Ergebnistabelle an. Der Name darf nicht qualifiziert sein und nicht in mehreren Spalten der Tabelle verwendet werden (SQLSTATE 42711).

FOR ORDINALITY
Gibt an, dass *spaltenname* die Ordinalitätsspalte der Ergebnistabelle ist. Der Datentyp dieser Spalte ist BIGINT. Die Wert dieser Spalte in der Ergebnistabelle ist die fortlaufende Zahl des Elements der Zeile in der Ergebnissequenz aus der Auswertung des XQuery-Ausdrucks in *xquery-zeilenausdruckskonstante*.

Wenn die Auswertung eines XQuery-Ausdrucks zu einem Fehler führt, gibt die Funktion XMLTABLE den XQuery-Fehler (SQLSTATE-Klasse '10') zurück.

Beispiel

Listen Sie als Tabellenergebnis die Bestellpositionen von Bestellungen mit dem Status 'NEW' (neu) auf.

```
SELECT U."PO ID", U."Part #", U."Product Name",
     U."Quantity", U."Price", U."Order Date"
  FROM PURCHASEORDER P,
     XMLTABLE('$po/PurchaseOrder/item' PASSING P.PORDER AS "po"
              COLUMNS "PO ID"         INTEGER       PATH '../@PoNum',
                      "Part #"        CHAR(10)      PATH 'partid',
                      "Product Name"  VARCHAR(50)   PATH 'name',
                      "Quantity"      INTEGER       PATH 'quantity',
                      "Price"         DECIMAL(9,2)  PATH 'price',
                      "Order Date"    DATE          PATH '../@OrderDate'
              ) AS U
   WHERE P.STATUS = 'Unshipped'
```

XMLEXISTS-Vergleichselement beim Abfragen von XML-Daten

Das XMLEXISTS-Vergleichselement bestimmt, ob ein XQuery-Ausdruck eine Sequenz aus einem oder mehreren Elementen zurückgibt. Wenn der in diesem Vergleichselement angegebene XQuery-Ausdruck eine leere Sequenz zurückgibt, liefert XMLEXISTS den Wert 'false'. Ansonsten wird der Wert 'true' zurückgegeben.

Das XMLEXISTS-Vergleichselement kann in den WHERE-Klauseln von SELECT-Anweisungen verwendet werden. Diese Verwendung bietet die Möglichkeit, Werte aus gespeicherten XML-Dokumenten zur Eingrenzung der Menge an Zeilen zu verwenden, auf denen die SELECT-Abfrage operiert.

Die folgende SQL-Abfrage veranschaulicht zum Beispiel, wie die zurückgegebenen Zeilen mithilfe des XMLEXISTS-Vergleichselements nur auf diejenigen eingegrenzt werden können, die ein XML-Dokument mit einem Element <city> enthalten, das den Wert "Toronto" hat. (Berücksichtigen Sie, dass in XQuery-Ausdrücken die Groß-/Kleinschreibung unterschieden wird, während dies in SQL nicht der Fall ist.)

```
SELECT Cid
FROM CUSTOMER
WHERE XMLEXISTS ('$d//addr[city="Toronto"]' passing INFO as "d")
```

Bitte beachten Sie, wie Sie Werte an XQuery-Variablen im XQuery-Ausdruck von XMLEXISTS übergeben können. In diesem Fall wird die XQuery-Variable $d an die Dokumente der Spalte INFO der Tabelle CUSTOMER gebunden. Eine vereinfachte Syntax steht ebenfalls zur Verfügung, mit der Spaltennamen übergeben werden können, ohne die Namen explizit in der Klausel **PASSING** angeben zu müssen. Siehe „Einfache Übergabe von Spaltennamen mit XMLEXISTS, XMLQUERY oder XMLTABLE” auf Seite 117.

Stellen Sie sicher, dass der XQuery-Ausdruck in XMLEXISTS korrekt angegeben wird, um die erwarteten Ergebnisse zu erhalten. Nehmen Sie zum Beispiel an, dass mehrere Dokumente in der XML-Spalte INFO der Tabelle CUSTOMER gespeichert sind, jedoch nur ein Dokument ein Attribut 'Cid' (im angegebenen Pfad) mit dem Wert 1000 enthält:

```
<customerinfo Cid="1000">
    <name>Kathy Smith</name>
  <addr country="Canada">
        <street>5 Rosewood</street>
        <city>Toronto</city>
    <prov-state>Ontario</prov-state>
        <pcode-zip>M6W 1E6</pcode-zip>
  </addr>
    <phone type="work">416-555-1358</phone>
</customerinfo>
```

Die beiden folgenden Abfragen liefern unterschiedliche Ergebnisse, weil sie einen kleinen Unterschied in den XQuery-Ausdrücken aufweisen:

```
    SELECT *
FROM CUSTOMER
WHERE XMLEXISTS ('$d/customerinfo[@Cid=1000]' passing INFO as "d")

    SELECT *
FROM CUSTOMER
WHERE XMLEXISTS ('$d/customerinfo/@Cid=1000' passing INFO as "d")
```

Die erste Abfrage gibt, wie erwartet, die Zeile zurück, die das oben dargestellte XML-Dokument enthält. Die zweite Abfrage gibt jedoch alle Zeilen der Tabelle CUSTOMER zurück, da das XMLEXISTS-Vergleichselement für den angegebenen XQuery-Ausdruck immer den Wert 'true' zurückgibt. Der XQuery-Ausdruck in der zweiten Abfrage liefert eine Sequenz aus Booleschen Elementen, die eine nicht leere Sequenz darstellen, sodass XMLEXISTS immer den Wert 'true' zurückgibt. Infolgedessen wird jede Zeile in der Tabelle CUSTOMER ausgewählt, was jedoch nicht dem gewünschten Ergebnis entspricht.

Verwendung des Vergleichselements XMLEXISTS

Wenn ein XMLEXISTS-Vergleichselement einen XPath-Ausdruck mit einem Wertvergleichselement (*ausdruck*) enthält, setzen Sie das Vergleichselement in eckige Klammern, sodass Sie '[*ausdruck*]' erhalten. Das Setzen des Wertevergleichselements in eckige Klammern stellt sicher, dass die Auswertung von *ausdruck* tatsächlich zu dem semantisch beabsichtigten Ergebnis führt.

Vergleichselement XMLEXISTS, Verhalten

Das folgende Szenario veranschaulicht, wie eine nicht leere Sequenz dafür sorgt, dass XMLEXISTS den Wert *true* liefert, obwohl die nicht leere Sequenz selbst nur aus dem Einzelwert *false* besteht. Es findet kein Indexabgleich statt, und die Abfrage liefert eine wesentlich größere Menge an Ergebnissen als erwartet. Das Problem lässt sich vermeiden, indem Wertvergleichselemente entsprechend in eckige Klammern ([]) gesetzt werden.

Betrachten Sie eine Tabelle, einen Index und zwei Abfragen:

```
CREATE TABLE mytable (id BIGINT, xmlcol XML);
         CREATE INDEX myidx ON mytable(xmlcol)
         GENERATE KEY USING XMLPATTERN '//text()' AS SQL VARCHAR(255);

SELECT xmlcol FROM mytable
         WHERE XMLEXISTS('$doc/CUSTOMER/ORDERS/ORDERKEY/text()="A512" '
         PASSING xmlcol AS "doc")
```

```
SELECT xmlcol FROM mytable
         WHERE XMLEXISTS('$doc/CUSTOMER[ORDERS/ORDERKEY/text()="A512"] '
       PASSING xmlcol AS "doc") ;
```

Der Grund für diese Funktionsweise ist folgender: XMLEXISTS wertet den XQuery-Ausdruck aus und gibt *false (für XMLEXISTS)* zurück, wenn das Ergebnis eine leere Sequenz ist, und *true (für XMLEXISTS)*, wenn das Ergebnis eine nicht leere Sequenz ist. Darauf folgt ein möglicherweise intuitiv nicht gleich durchschaubarer nächster Schritt in der Abfrageauswertung: In der ersten Abfrage gibt der Ausdruck die Anweisung, ORDERKEY mit dem Wert 'A512' zu vergleichen. Das Ergebnis dieses Ausdrucks ist entweder der Wert *false* oder der Wert *true*, je nachdem, welchen Wert ORDERKEY tatsächlich hat. Daher erkennt die Funktion XMLEXISTS in jedem Fall eine Rückkehrsequenz mit einem einzelnen Element, das heißt, ein Element, das entweder *false* oder *true* ist. Da eine Sequenz mit einem Element keine leere Sequenz ist, gibt die Funktion XMLEXISTS **auf jeden Fall** insgesamt den Wert *true (für XMLEXISTS)* zurück. Infolgedessen gibt die Abfrage alle Zeilen zurück. Indizes können nicht genutzt werden, wenn XMLEXISTS in einer Weise verwendet wird, in der sich alle Zeilen als zur Rückgabe qualifiziert erweisen.

Die folgenden fünf Beispiele zeigen nicht leere Sequenzen, wobei drei Sequenzen nur ein Element enthalten:

```
(42, 3,4,78, 1966)
   (true)
   (abd, def)
   (false)
   (5)
```

Jede nicht leere Sequenz dieser Art bewirkt, dass XMLEXISTS den Wert *true (für XMLEXISTS)* zurückgibt, auch wenn die nicht leere Sequenz, die sich aus dem Ausdruck ergibt, selbst den Wert (*false*) hat.

In der zweiten Abfrage stellt der Ausdruck in XMLEXISTS die Anweisung dar, alle Kunden zurückzugeben, die eine Bestellung mit dem Wert 'A512' für ORDERKEY enthalten. Wenn in einem Dokument kein solcher Kunde vorhanden ist, besteht das Ergebnis jedoch diesmal tatsächlich in einer leeren Sequenz. Diese Abfrage nutzt einen Index und liefert die erwarteten Ergebnisse zurück.

Vergleichselement XMLEXISTS, Verwendung

Wenn der gesamte *ausdruck* in eckige Klammern gesetzt wird, wird er als 'Gib die XML-Daten zurück, wenn [*ausdruck*]' interpretiert. Sie sollten daher immer eine leere Sequenz empfangen, wenn die XML-Daten den *ausdruck* nicht erfüllen.

Die Wertevergleiche in allen folgenden Beispielfragmenten für die Verwendung von XMLEXISTS-Vergleichselementen sind jeweils in eckige Klammern gesetzt sind, sodass sie wie erwartet funktionieren:

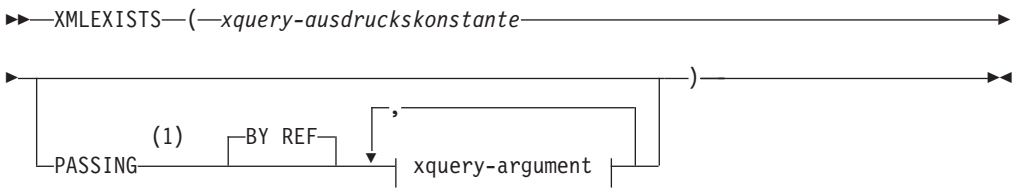
```
... WHERE XMLEXISTS('$doc[CUSTOMER/ORDERS/ORDERKEY/text()="A512"] '
             PASSING xmlcol as "doc") ;
      ... WHERE XMLEXISTS('$doc/CUSTOMER[ORDERS/ORDERKEY/text()="A512"] '
       PASSING xmlcol AS "doc") ;
      ... WHERE XMLEXISTS('$doc/CUSTOMER/ORDERS[ORDERKEY/text()="A512"] '
          PASSING xmlcol AS "doc") ;
```

Diese Richtlinie gilt auch für Abfragen, in denen kein Wertevergleich erfolgt, zum Beispiel wenn Sie die Dokumente für alle Kunden abrufen wollen, die zufällig ein untergeordnetes Element *COMMENT* enthalten:

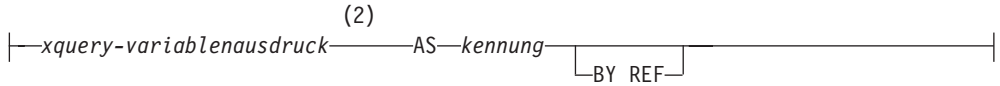
```
... WHERE XMLEXISTS('$doc[CUSTOMER/COMMENT ] '
      PASSING xmlcol AS "doc") ;
```

Vergleichselement XMLEXISTS

Das Vergleichselement XMLEXISTS testet, ob ein XQuery-Ausdruck eine Sequenz aus einem oder mehreren Elementen zurückgibt.



xquery-argument:



Anmerkungen:

1 Der Datentyp darf nicht DECFLOAT sein.

2 Der Datentyp des Ausdrucks darf nicht DECFLOAT sein.

xquery-ausdruckskonstante
Gibt eine SQL-Zeichenfolgekonstante an, die als ein XQuery-Ausdruck interpretiert wird. Die Zeichenfolge der Konstanten wird ohne Konvertierung in die Datenbank- oder Abschnittscodepage direkt in UTF-8 konvertiert. Der XQuery-Ausdruck wird mit einer optionalen Gruppe von XML-Eingabewerten ausgeführt und gibt eine Ausgabesequenz zurück, die getestet wird, um das Ergebnis des Vergleichselements XMLEXISTS zu ermitteln. Der Wert für *xquery-ausdruckskonstante* darf keine leere Zeichenfolge oder eine Zeichenfolge mit Leerzeichen sein (SQLSTATE 10505).

PASSING
Gibt Eingabewerte und das Verfahren für ihre Übergabe an den mit *xquery-ausdruckskonstante* angegebenen XQuery-Ausdruck an. Standardmäßig wird jeder eindeutige Spaltenname, der sich im Geltungsbereich des Funktionsaufrufs befindet, implizit an den XQuery-Ausdruck übergeben, wobei der Name der Spalte als Variablenname verwendet wird. Wenn eine Kennung (*kennung*) in einem angegebenen XQuery-Argument mit einem gültigen Spaltennamen übereinstimmt, wird das explizite XQuery-Argument an den XQuery-Ausdruck übergeben und überschreibt diese implizite Spalte.

BY REF
Gibt an, dass die Werte bei allen Vorkommen von *xquery-variablenausdruck* des Datentyps XML standardmäßig per Verweis übergeben werden. Bei der Übergabe der XML-Werte per Verweis verwendet die XQuery-Auswertung die Eingabeknotenbaumstrukturen (sofern vorhanden) direkt aus den angegebenen Eingabeausdrücken, wobei alle Eigenschaften beibehalten werden, einschließlich der ursprünglichen Knotenidentitäten und der Dokumentreihenfolge. Wenn zwei Argumente denselben XML-Wert übergeben, verweisen Vergleiche der Knotenidentitäten und der Dokumentreihenfolge, die bestimmte zwischen den beiden Eingabeargumenten enthaltene Knoten einbeziehen, möglicherweise auf Knoten derselben XML-Knotenbaumstruktur.

Diese Klausel hat keinen Einfluss auf das Übergabeverfahren von Nicht-XML-Werten. Nicht-XML-Werte erstellen während der Umsetzung in XML eine neue Kopie des jeweiligen Werts.

xquery-argument
Gibt ein Argument an, das an den mit *xquery-ausdruckskonstante* angegebenen XQuery-Ausdruck übergeben werden soll. Ein Argument gibt einen Wert und das Verfahren für dessen Übergabe an. Das Argument enthält einen SQL-Ausdruck, der ausgewertet wird.

- Wenn der resultierende Wert den Typ XML aufweist, wird er zu einem XML-Eingabewert (*xml-eingabewert*). Ein XML-Nullwert wird in eine leere XML-Sequenz konvertiert.
- Wenn der resultierende Wert nicht den Typ XML aufweist, muss er in den XML-Datentyp umsetzbar sein. Ein Nullwert wird in eine leere XML-Sequenz konvertiert. Der konvertierte Wert wird zu einem XML-Eingabewert (*xml-eingabewert*).

Bei der Auswertung von *xquery-ausdruckskonstante* wird eine XQuery-Variable mit einem Wert, der gleich *xml-eingabewert* ist, sowie mit einem durch die Klausel AS angegebenen Namen dargestellt.

xquery-variablenausdruck
Gibt einen SQL-Ausdruck an, dessen Wert während der Ausführung für den mit *xquery-ausdruckskonstante* angegebenen XQuery-Ausdruck verfügbar ist. Der Ausdruck darf weder einen Sequenzverweis (SQLSTATE 428F9) noch eine OLAP-Funktion (SQLSTATE 42903) enthalten. Der Datentyp des Ausdrucks darf nicht DECFLOAT sein.

AS *kennung*
Gibt an, dass der von *xquery-variablenausdruck* generierte Wert als XQuery-Variable an *xquery-ausdruckskonstante* übergeben wird. Der Variablenname lautet *kennung*. Das führende Dollarzeichen ($), das Variablennamen in der XQuery-Sprache vorangestellt wird, ist in *kennung* nicht enthalten. Die Kennung muss ein gültiger XQuery-Variablenname sein und ist auf einen XML-Namen ohne Doppelpunkte (NCName) beschränkt. Die Kennung darf eine Höchstlänge von 128 Byte nicht überschreiten. Zwei Argumente in derselben Klausel PASSING dürfen nicht dieselbe Kennung verwenden (SQLSTATE 42711).

BY REF
Gibt an, dass ein XML-Eingabewert per Verweis übergeben werden soll. Bei der Übergabe der XML-Werte per Verweis verwendet die XQuery-Auswertung die Eingabeknotenbaumstrukturen (sofern vorhanden) direkt aus den angegebenen Eingabeausdrücken, wobei alle Eigenschaften beibehalten werden, einschließlich der ursprünglichen Knotenidentitäten und der Dokumentreihenfolge. Wenn zwei Argumente denselben XML-Wert übergeben, verweisen Vergleiche der Knotenidentitäten und der Dokumentreihenfolge, die bestimmte zwischen den beiden Eingabeargumenten enthaltene Knoten einbeziehen, möglicherweise auf Knoten derselben XML-Knotenbaumstruktur. Wenn BY REF nicht im Anschluss an *xquery-variablenausdruck* angegeben wird, werden die XML-Argumente entsprechend dem Standardübergabeverfahren übergeben, das mit der Syntax nach dem Schlüsselwort PASSING bereitgestellt wird. Diese Option kann für Nicht-XML-Werte nicht angegeben werden. Bei der Übergabe eines Nicht-XML-Werts wird der Wert in XML konvertiert. Dabei wird eine Kopie erstellt.

Anmerkungen

Das Vergleichselement XMLEXISTS darf Folgendes nicht sein:

- Teil der Klausel ON, der einem JOIN-Operator oder einer MERGE-Anweisung zugeordnet ist (SQLSTATE 42972)
- Teil der Klausel GENERATE KEY USING oder RANGE THROUGH in der Anweisung CREATE INDEX EXTENSION (SQLSTATE 428E3)
- Teil der Klausel FILTER USING in der Anweisung CREATE FUNCTION (extern, skalar) oder der Klausel FILTER USING in der Anweisung CREATE INDEX EXTENSION (SQLSTATE 428E4)
- Teil einer Prüfung auf Integritätsbedingung oder eines Ausdrucks zur Spaltengenerierung (SQLSTATE 42621)
- Teil einer Gruppierung nach Klausel (SQLSTATE 42822)
- Teil eines Arguments für eine Spaltenfunktion (SQLSTATE 42607)

Ein Vergleichselement XMLEXISTS, das eine Unterabfrage einbezieht, kann durch Anweisungen eingeschränkt werden, die Unterabfragen einschränken.

Beispiel

```
SELECT c.cid FROM customer c
  WHERE XMLEXISTS('$d/*:customerinfo/*:addr[ *:city = "Aurora" ]'
    PASSING info AS "d")
```

Übergabe von Parametern zwischen SQL-Anweisungen und XQuery-Ausdrücken

Beim Absetzen einer Kombination aus SQL-Anweisungen und XQuery-Ausdrücken können Sie Daten zwischen den Anweisungen und Ausdrücken übergeben, um die Art der Ausführung der Anweisungen und Ausdrücken zu ändern.

Übergeben von Konstanten und Parametermarken an XMLEXISTS und XMLQUERY

Das Vergleichselement XMLEXISTS und die Skalarfunktion XMLQUERY führen XQuery-Ausdrücke innerhalb einer SQL-Anweisung aus. Mithilfe von Konstanten und Parametermarken werden Daten von der SQL-Anweisung an Variablen in einem XQuery-Ausdruck übergeben, der innerhalb der SQL-Anweisung ausgeführt wird.

In einem XQuery-Ausdruck in XMLEXISTS und XMLQUERY können XQuery-Variablen angegeben werden. Werte werden durch die Klausel 'passing' an diese Variablen übergeben. Bei diesen Werten handelt es sich um SQL-Ausdrücke.Da die an den XQuery-Ausdruck übergebenen Werte keine XML-Werte sind, müssen sie entweder implizit oder explizit in Typen umgesetzt werden, die von DB2 XQuery unterstützt werden. Weitere Informationen zu unterstützten Typumsetzungen finden Sie in der Dokumentation zur Umsetzung von Datentypen.

Die Methode der Übergabe von Konstanten und Parametermarken an die Funktion XMLQUERY ist mit der für das XMLEXISTS-Vergleichselement identisch, jedoch ist ihre Verwendung in XMLEXISTS-Vergleichselementen häufiger. Dies liegt daran, dass parametrisierte Vergleichselemente in der Funktion XMLQUERY bei Angabe in SELECT-Klauseln keine Zeilen aus der Ergebnismenge ausschließen. Stattdessen werden die Vergleichselemente zur Bestimmung der zurückzugebenden Fragmente eines Dokuments verwendet. Um Zeilen effektiv aus einer Ergebnismenge zu eli-

minieren, sollte das XMLEXISTS-Vergleichselement in der WHERE-Klausel verwendet werden. Zeilen, die leere Sequenzen enthalten, werden dann nicht als Teil der Ergebnismenge zurückgegeben. Die an dieser Stelle behandelten Beispiele zeigen diese häufigere Verwendung in XMLEXISTS-Vergleichselementen.

Beispiel: Implizite Typumsetzung

In der folgenden Abfrage wird die SQL-Zeichenfolgekonstante 'Aurora', bei der es sich nicht um einen XML-Typ handelt, im XMLEXISTS-Vergleichselement implizit in einen XML-Typ umgewandelt. Im Anschluss an die implizite Typumsetzung besitzt die Konstante den XML-Schemasubtyp 'xs:string' und ist an die Variable '$cityName' gebunden. Diese Konstante kann nun im Vergleichselement des XQuery-Ausdrucks verwendet werden.

```
SELECT XMLQUERY ('$d/customerinfo/addr' passing c.INFO as "d")
FROM Customer as c
WHERE XMLEXISTS('$d//addr[city=$cityName]'
                                  passing c.INFO as "d",
                                  'Aurora' AS "cityName")
```

Beispiel: Explizite Typumsetzung

In der folgenden Abfrage muss die Parametermarke explizit in einen Datentyp umgewandelt werden, weil der Typ der Parametermarke nicht bestimmt werden kann. Die Parametermarke, die explizit in einen SQL-Datentyp VARCHAR umgewandelt wird, wird anschließend implizit in den XML-Schematyp xs:string umgewandelt.

```
SELECT XMLQUERY ('$d/customerinfo/addr' passing c.INFO as "d")
FROM Customer as c
WHERE XMLEXISTS('$d//addr[city=$cityName]'
                                  passing c.INFO as "d",
                CAST (? AS VARCHAR(128)) AS "cityName")
```

Einfache Übergabe von Spaltennamen mit XMLEXISTS, XMLQUERY oder XMLTABLE

Um die Verwendung des Vergleichselements XMLEXISTS, der Skalarfunktion XMLQUERY und der Tabellenfunktion XMLTABLE zu vereinfachen, können Sie einen Spaltennamen als Parameter in einem mit XMLEXISTS, XMLQUERY oder XMLTABLE angegebenen XQuery-Ausdruck verwenden, ohne den betreffenden Namen in der Klausel **PASSING** angeben zu müssen.

Sie müssen die Klausel **PASSING** zur Übergabe des Spaltennamens als Parameter verwenden, wenn der verwendete Parametername nicht mit dem übergebenen Spaltenname übereinstimmt.

Wenn in der Klausel **PASSING** explizit eine Variable angegeben wird, deren Name mit dem Namen einer im XQuery-Ausdruck angegebenen Variablen kollidiert, wird die Variable in der Klausel **PASSING** vorrangig behandelt. Angenommen die Tabelle CUSTOMER enthält zwei XML-Spalten namens INFO und CUST. In folgendem Beispiel werden XML-Daten aus der Spalte INFO abgerufen:

```
SELECT XMLQuery('$CUST/customerinfo/name' PASSING INFO as "CUST") FROM customer
```

Die in der Klausel **PASSING** angegebene Variable CUST wird verwendet, um die Spalte INFO im XQuery-Ausdruck zu ersetzen. Die Spalte CUST aus der Tabelle CUSTOMER wird nicht verwendet.

Beispiele: XMLQUERY und XMLEXISTS

Bitte beachten Sie, dass die Spaltennamen in diesen Beispielen von der Groß-/Kleinschreibung abhängen, da sie in doppelte Anführungszeichen eingeschlossen sind. Sind die Namen nicht in doppelte Anführungszeichen eingeschlossen, gelten die regulären Namensregeln: Spaltennamen sind nicht von der Groß-/Kleinschreibung abhängig und werden in Großbuchstaben gespeichert.

Das nachstehende Beispiel zeigt eine Reihe von SELECT-Anweisungen, die dieselbe Sequenz aus Dokumenten aus der Tabelle PURCHASEORDER zurückgeben. Die XML-Dokumente befinden sich in der Spalte PORDER. Die ersten beiden SELECT-Anweisungen verwenden die Klausel **PASSING**, um den Spaltennamen PORDER an den XQuery-Ausdruck in der Skalarfunktion XMLQUERY zu übergeben. Die dritte SELECT-Klausel verwendet den Spaltennamen PORDER als implizit übergebenen Parameter:

```
SELECT XMLQuery('$PORDER/PurchaseOrder/item/name'  PASSING porder AS "PORDER")
   FROM purchaseorder
SELECT XMLQuery('$PORDER/PurchaseOrder/item/name' PASSING porder AS "porder")
   FROM purchaseorder
SELECT XMLQuery('$PORDER/PurchaseOrder/item/name') FROM purchaseorder
```

Die nachstehenden beiden Beispiele zeigen mehrere Funktionsaufrufe, die sowohl XMLQUERY als auch XMLEXISTS verwenden. Beide Beispiele geben dieselbe Sequenz aus Dokumenten aus der Tabelle CUSTOMER zurück.

Das folgende Beispiel verwendet die Klausel **PASSING**, um den Spaltennamen INFO explizit als Parameter in der Skalarfunktion XMLQUERY und im Vergleichselement XMLEXISTS anzugeben:

```
SELECT XMLQUERY ('$INFO/customerinfo/addr' passing Customer.INFO as "INFO")
FROM CustomerWHERE XMLEXISTS('$INFO//addr[city=$cityName]'
                 passing Customer.INFO as "INFO",
                                  'Aurora' AS "cityName")
```

In nachstehendem Beispiel verwendet die Funktion XMLQUERY keine Klausel PASSING, und die Klausel **PASSING** von XMLEXISTS gibt nicht die Spalte INFO an. Der Spaltenname INFO wird sowohl in der Skalarfunktion XMLQUERY als auch im Vergleichselement XMLEXISTS implizit an den XQuery-Ausdruck übergeben:

```
SELECT XMLQUERY ('$INFO/customerinfo/addr')
FROM CustomerWHERE XMLEXISTS('$INFO//addr[city=$cityName]'
                 passing 'Aurora' AS "cityName")
```

Beispiel: XMLTABLE

Die nachstehenden beiden Beispiele zeigen zwei INSERT-Anweisungen, die die Tabellenfunktion XMLTABLE verwenden. In beiden Beispielen werden dieselben Daten aus der Tabelle T1 in die Tabelle CUSTOMER eingefügt. Die Tabelle T1 enthält eine XML-Spalte namens CUSTLIST. Die Funktion XMLTABLE ruft die Daten aus der Spalte T1.CUSTLIST ab und gibt eine Tabelle mit drei Spalten (Cid, Info und History) zurück. Die Anweisung INSERT fügt Daten aus der Funktion XMLTABLE in drei Spalten der Tabelle CUSTOMER ein.

Das folgende Beispiel verwendet die Klausel **PASSING**, um den Spaltennamen CUSTLIST explizit als Parameter in der Tabellenfunktion XMLTABLE anzugeben:

```
INSERT INTO customer SELECT X.* FROM T1,
    XMLTABLE( '$custlist/customers/customerinfo' passing T1.custlist as "custlist"
                    COLUMNS
    "Cid" BIGINT PATH '@Cid',
    "Info" XML PATH 'document{.}',
    "History" XML PATH 'NULL') as X
```

In nachstehendem Beispiel verwendet die Tabellenfunktion XMLTABLE keine Klausel vom Typ **PASSING**. XMLTABLE verwendet den Spaltennamen CUSTLIST aus der Tabelle T1 als implizit übergebenen Parameter:

```
INSERT INTO customer SELECT X.* FROM T1,
    XMLTABLE( '$custlist/customers/customerinfo'
                    COLUMNS
    "Cid" BIGINT PATH '@Cid',
    "Info" XML PATH 'document{.}',
    "History" XML PATH 'NULL') as X
```

Übergeben von Parametern von XQuery an SQL

Innerhalb eines XQuery-Ausdrucks führt die Funktion 'db2-fn:sqlquery' eine SQL-Fullselect-Anweisung aus, die eine XML-Knotensequenz abruft. Bei Verwendung von 'db2-fn:sqlquery' wird mithilfe der Funktion PARAMETER auf die Daten verwiesen, die vom XQuery-Ausdruck an die in 'db2-fn:sqlquery' angegebene SQL-Fullselect-Anweisung übergeben werden.

Mithilfe der Funktion PARAMETER können Parameter als Teil eines SQL-Fullselect-Ausdrucks in 'db2-fn:sqlquery' angegeben werden. Wenn Sie Funktionen vom Typ PARAMETER beim Aufruf von 'db2-fn:sqlquery' verwenden, müssen Sie auch XQuery-Ausdrücke angeben, die von den PARAMETER-Funktionen verwendet werden. Während der Verarbeitung der SQL-Fullselect-Anweisung wird jeder Aufruf der Funktion PARAMETER durch den Ergebniswert des entsprechenden XQuery-Ausdrucks im Aufruf der Funktion 'db2-fn:sqlquery' ersetzt. Auf den von der Funktion PARAMETER gelieferten Wert kann in derselben SQL-Anweisung mehrmals verwiesen werden.

Die XQuery-Ausdrücke, die zum Aufruf der Funktion 'db2-fn:sqlquery' gehören, geben einen Wert zurück. Da die an die Fullselect-Anweisung übergebenen Werte XML-Werte sind, müssen sie entweder implizit oder explizit in Typen umgesetzt werden, die von DB2 SQL unterstützt werden. Weitere Informationen zu unterstützten Typumsetzungen finden Sie in der Dokumentation zur Funktion 'db2-fn:sqlquery' sowie in der Dokumentation zur Umsetzung von Datentypen.

Beispiel: Übergabe eines Parameters an 'db2-fn:sqlquery'

Das nachstehende Beispiel ist ein XQuery-Ausdruck, der die Funktion 'db2-fn:sqlquery' verwendet. Während der Verarbeitung der Funktion 'db2-fn:sqlquery' geben beide Verweise auf `parameter(1)` den Wert des Attributs `$po/@OrderDate` für das Bestelldatum zurück.

Bei Ausführung für die DB2-Beispieldatenbank SAMPLE gibt der XQuery-Ausdruck die Bestell-ID (PoNum), die Teilekennung (PartID) und das Bestelldatum (PoDate) für alle innerhalb der Werbeaktion verkauften Teile zurück:

```
  xquery
for $po in db2-fn:xmlcolumn('PURCHASEORDER.PORDER')/PurchaseOrder,
  $item in $po/item/partid
for $p in db2-fn:sqlquery(
  "select description
  from product
  where promostart < parameter(1)
```

```
  und
  promoend  >  parameter(1)",
  $po/@OrderDate )
where $p//@pid = $item
return
<RESULT>
  <PoNum>{data($po/@PoNum)}</PoNum>
  <PartID>{data($item)} </PartID>
  <PoDate>{data($po/@OrderDate)}</PoDate>
</RESULT>
```

Datenabruf mit XQuery

XQuery-Ausdrücke können entweder mit XQuery als Primärsprache oder mit SQL (mit der SQL-Funktion XMLQUERY) als Primärsprache ausgeführt werden. Wenn ein XQuery-Ausdruck mit einer dieser Methoden ausgeführt wird, wird eine XML-Sequenz zurückgegeben.

Die XQuery-Spezifikation definiert das Ergebnis eines XQuery-Ausdrucks als Sequenz aus 0, 1 oder mehr Elementen.

Die Art und Weise, wie die Ergebnissequenz in einer Ergebnismenge dargestellt wird, hängt davon ab, ob SQL oder XQuery als Primärsprache verwendet wird:

XQuery als Primärsprache
Wenn ein XQuery-Ausdruck mit XQuery als Primärsprache ausgeführt wird, wird das Ergebnis an eine Clientanwendung in Form einer Ergebnistabelle mit einer Spalte des Typs XML zurückgegeben. Jede Zeile in dieser Ergebnistabelle ist ein Element der Sequenz, die sich aus der Auswertung des XQuery-Ausdrucks ergibt. Wenn eine Anwendung mithilfe eines Cursors Daten aus dieser Ergebnistabelle abruft, liefert jeder Abruf ein serialisiertes Element der Ergebnissequenz.

SQL als Primärsprache (unter Verwendung von XMLQUERY)
XMLQUERY ist eine Skalarfunktion, die einen XML-Wert zurückgibt. Der zurückgegebene Wert ist eine Sequenz aus 0, 1 oder mehreren Elementen. Alle Elemente der Ergebnissequenz werden in Form eines einzigen serialisierten Werts an eine Anwendung zurückgegeben.

Zum Abrufen von Ergebnissen aus Abfragen, die XQuery oder XMLQUERY verwenden, rufen Sie die Ergebnisse aus Ihrer Anwendung ab, wie Sie dies bei jeder anderen Ergebnismenge ebenfalls tun würden. Binden Sie die Anwendungsvariable an die Ergebnismenge und rufen Sie die Daten bis zum Ende der Ergebnismenge ab. Wenn der (direkt oder durch XMLQUERY ausgeführte) XQuery-Ausdruck eine leere Sequenz zurückgegeben hat, ist auch die Zeile in der Ergebnismenge leer.

Verwalten von Abfrageergebnismengen

Wenn Ihre Anwendung verlangt, dass die beim Abfragen mit XQuery zurückgegebenen XML-Werte korrekt formatierte XML-Dokumente sein müssen (zum Beispiel, weil Sie diese Werte in eine Spalte des Typs XML einfügen möchten), können Sie einen Element- oder Dokumentkonstruktor in Ihren XQuery-Ausdruck einschließen, um sicherzustellen, dass die Werte als korrekt formatierte XML-Dokumente zurückgegeben werden.

Beispiel: Unterschied in den Ergebnismengen von XQuery und XMLQUERY

Dieses Beispiel veranschaulicht den Unterschied zwischen den Ergebnismengen der beiden Abfragemethoden.

Wenn die beiden folgenden XML-Dokumente in einer XML-Spalte gespeichert sind, können Sie zum Abrufen aller Elemente <phone> entweder XQuery oder XMLQUERY verwenden. Die von diesen beiden Methoden zurückgegebenen Ergebnismengen unterscheiden sich jedoch und sollten von der jeweiligen Anwendung beim Abrufen der Ergebnismenge entsprechend verarbeitet werden.

```
<customerinfo Cid="1000">
    <name>Kathy Smith</name>
  <addr country="Canada">
        <street>5 Rosewood</street>
        <city>Toronto</city>
    <prov-state>Ontario</prov-state>
        <pcode-zip>M6W 1E6</pcode-zip>
  </addr>
    <phone type="work">416-555-1358</phone>
</customerinfo>

<customerinfo Cid="1003">
    <name>Robert Shoemaker</name>
    <addr country="Canada">
        <street>1596 Baseline</street>
        <city>Aurora</city>
    <prov-state>Ontario</prov-state>
        <pcode-zip>N8X 7F8</pcode-zip>
  </addr>
    <phone type="work">905-555-7258</phone>
    <phone type="home">416-555-2937</phone>
    <phone type="cell">905-555-8743</phone>
    <phone type="cottage">613-555-3278</phone>
</customerinfo>
```

Ausführung eines XQuery-Ausdrucks mit XQuery als Primärsprache wie im folgenden Beispiel:

```
XQUERY
  db2-fn:xmlcolumn ('CUSTOMER.INFO')/customerinfo/phone
```

In der daraus resultierenden Ergebnismenge werden fünf Zeilen wie folgt zurückgegeben:

```
    <phone type="work">416-555-1358</phone>
    <phone type="work">905-555-2937</phone>
    <phone type="home">416-555-2937</phone>
   <phone type="cell">905-555-8743</phone>
   <phone type="cottage">613-555-3278</phone>
```

Ausführung eines XQuery-Ausdrucks über XMLQUERY wie im folgenden Beispiel:

```
SELECT XMLQUERY ('$doc/customerinfo/phone' PASSING INFO AS "doc")
FROM CUSTOMER
```

In der Ergebnismenge werden zwei Zeilen zurückgegeben, wie nachfolgend aufgeführt, wobei alle Elemente <phone> der zweiten Zeile in der Tabelle zu einem einzelnen Skalarwert (einer XML-Sequenz) verkettet werden:

```
   <phone type="work">416-555-1358</phone>
```

```
<phone type="work">905-555-2937</phone><phone type="home">416-555-2937</
phone><phone type="cell">905-555-8743</phone><phone type="cottage">613-
555-3278</phone>
```

Beachten Sie, dass die zweite Zeile dieser Ergebnismenge einen Wert enthält, der kein korrekt formatiertes XML-Dokument darstellt.

Diese Unterschiede in den Ergebnismengen sind darauf zurückzuführen, dass XMLQUERY eine Skalarfunktion ist. Sie wird für jede Zeile der Tabelle und die Ergebnissequenz aus einer Zeile der Tabelle ausgeführt und bildet eine Zeile der Ergebnismenge. XQuery hingegen gibt jedes Element einer Sequenz in einer separaten Zeile der Ergebnismenge zurück.

Beispiel: Verwalten von Abfrageergebnismengen

In diesem Beispiel wird die vorige SQL-Abfrage abgewandelt, indem ein XQuery-Dokumentknotenkonstruktor ('document') eingefügt wird, der sicherstellt, dass sämtliche Ergebniszeilen korrekt formatierte Dokumente enthalten:

```
SELECT XMLQUERY ('document{<phonelist>{$doc/customerinfo/phone}</phonelist>}'
    PASSING INFO AS "doc")
FROM CUSTOMER
```

In der Ergebnismenge dieser Abfrage werden zwei Zeilen zurückgegeben, wobei davon ausgegangen wird, dass dieselben Dokumente wie die zuvor dargestellten in der Datenbank vorhanden sind (die Ausgabe wurde aus Gründen der Übersichtlichkeit formatiert).

```
<phonelist><phone type="work">416-555-1358</phone></phonelist>
```

```
<phonelist><phone type="work">905-555-7258</phone><phone
type="home">416-555-2937</phone><phone type="cell">905-555-8743</
phone><phone type="cottage">613-555-3278</phone></phonelist>
```

Richtlinien zum Abgleich von Indizes mit Abfragen - Übersicht

Dieser Abschnitt enthält einige Richtlinien und Beispiele für den Abgleich von Abfragen mit Indizes zu XML-Daten.

Ob eine Abfrage einen Index nutzen kann, hängt davon ab, ob der Index bzw. die Indizes, die Sie erstellt haben, mit Ihrer Abfrage kompatibel sind (Indexabgleich) und ob das Optimierungsprogramm bei der Abfrageauswertung eine Indexsuche auswählt. Dem Zugriffsplan der EXPLAIN-Funktion ist zu entnehmen, ob bei der Abfrageauswertung eine Indexsuche ausgeführt wird.

Eine Abfrage muss mindestens die folgenden Bedingungen erfüllen, um einen Index für XML-Daten nutzen zu können:

- Die Datentypen in der Abfragesuchbedingung stimmen mit den indexierten Datentypen überein.
- Die Abfragesuchbedingung schließt eine Untergruppe der indexierten Knoten mit ein.

SQL- und XQuery-Optimierungsprogramm

Das Optimierungsprogramm plant die Auswertung von Abfragen und wählt die bei der Auswertung zu verwendenden Indizes aus. Bei der Abfragekompilierung

wird eine Abfrage mit allen Mustern in den XML-Indexdefinitionen abgeglichen, um Kandidatenindizes zu ermitteln, die genügend Informationen enthalten, um einen Teil der Abfrage zu erfüllen.

Das Optimierungsprogramm kann bei der Abfrageauswertung einen der folgenden Schritte ausführen:

- Durchsuchen der Tabelle mit den XML-Dokumenten ohne Verwendung eines Index
- Verwenden eines relationalen Index
- Verwenden logischer Verknüpfungen von Indizes über AND oder OR (Index ANDing oder Index ORing)
- Verwenden eines neuen XML-Indexoperators
- Verwenden eines Index für XML-Daten zur Auswertung eines einzelnen XML-Musters
- Verwenden von ANDing und ORing für einen Index für XML-Daten für die Auswertung komplexer XML-Muster über eine einzelne Abfrage.

EXPLAIN-Funktion

Mithilfe der EXPLAIN-Funktion kann der Zugriffsplan bereitgestellt werden, der zur Auswertung Ihrer Abfrage ausgewählt wurde. Bei der Prüfung des Zugriffsplans geben die folgenden Operatoren an, ob einer oder mehrere Indizes zur Auswertung der Abfrage verwendet wurden:

IXAND
Stellt die Verknüpfung der Satz-IDs (RIDs) aus zwei oder mehr Indexsuchen über logisches AND dar.

XISCAN
Stellt das Durchsuchen eines Index zu XML-Daten dar.

XANDOR
Stellt die mögliche Anwendung von Vergleichselementen, die logisch über AND verknüpft sind, auf mehrere XML-Indizes dar.

Restriktivität von Indexdefinitionen

Ob bei der Auswertung einer Abfrage ein Index verwendet werden kann hängt häufig davon ab, wie restriktiv die Indexdefinition im Vergleich zur betreffenden Abfrage ist. Die folgenden Beispiele zeigen eine Reihe von Abfragen und Indizes, die zusammen verwendet werden können:

Indizes für Abfragen mit einem Vergleichselement für Bereiche
Die folgende Abfrage ruft Unternehmensinformationen für Mitarbeiter mit einem Gehalt von mehr als *35000* aus der Tabelle *companyinfo* mit der XML-Spalte *companydocs* ab:

```
SELECT companydocs FROM companyinfo
   WHERE XMLEXISTS('$x/company/emp[@salary > 35000]'
      PASSING companydocs AS "x")
```

Um kompatibel zu sein, muss ein Index für XML-Daten Gehaltsattributknoten ('salary') für Mitarbeiter in die indexierten Knoten mit einschließen und die Werte mit dem Datentyp DOUBLE oder DECIMAL speichern.

Die Abfrage könnte zum Beispiel die beiden folgenden Indizes zu XML-Daten verwenden:

```
CREATE INDEX empindex on companyinfo(companydocs)
   GENERATE KEY USING XMLPATTERN '//@salary' AS SQL DECIMAL(10,2)

CREATE INDEX empindex on companyinfo(companydocs)
      GENERATE KEY USING XMLPATTERN '/company/emp/@salary'
   AS SQL DECIMAL(10,2)
```

Indizes zur Verwendung durch mehrere Abfragen

Die folgende Abfrage ruft Unternehmensinformationen zu Mitarbeitern mit der Mitarbeiter-ID *31664* ab.

```
SELECT companydocs FROM companyinfo
   WHERE XMLEXISTS('$x/company/emp[@id="31664"]'
      PASSING companydocs AS "x")
```

Eine zweite Abfrage ruft Unternehmensinformationen zu Abteilungen ('dept') mit der ID *K55* ab:

```
SELECT companydocs FROM companyinfo
   WHERE XMLEXISTS('$x/company/emp/dept[@id="K55"]
      PASSING companydocs AS "x")
```

Um mit beiden Abfragen kompatibel zu sein, muss der Index für XML-Daten Attributknoten für die Mitarbeiter-IDs und Abteilungs-IDs in die indexierten Knoten mit einschließen und die Werte mit einem VARCHAR-Datentyp speichern.

Die Abfragen könnten den folgenden Index für XML-Daten nutzen:

```
CREATE INDEX empdeptindex on companyinfo(companydocs)
        GENERATE KEY USING XMLPATTERN '//@id' AS SQL VARCHAR(25)
```

Einschließen von Namensbereichen beim Beschränken von XQuery-Vergleichselementen

Betrachten Sie die folgende Tabelle mit einer XML-Spalte, die Kundeninformationen enthält, und den für die XML-Spalte erstellten Index:

```
CREATE TABLE customer(xmlcol XML) %

CREATE UNIQUE INDEX customer_id_index ON customer(xmlcol)
      GENERATE KEY USING XMLPATTERN
      'DECLARE DEFAULT ELEMENT NAMESPACE
      "http://mynamespace.org/cust";/Customer/@id'
      AS SQL DOUBLE %
```

Anmerkung: In diesem Abschnitt dient das Prozentzeichen (%) als Anweisungsabschlusszeichen, da das Semikolon (;) bereits als Begrenzungszeichen für den Namensbereich ('NAMESPACE') verwendet wird.

Die folgende Abfrage entspricht dem Index nicht:

```
SELECT xmlcol FROM customer
      WHERE XMLEXISTS('$xmlcol/*:Customer[@id=1042]'
      PASSING xmlcol AS "xmlcol") %
```

Damit die Abfrage den Index verwenden kann, muss die Abfrage mindestens ebenso restriktiv wie der Index oder restriktiver sein. Der Index 'customer_id_index' deckt nur Kundenelemente ('customer') in einem bestimmten Namensbereich (*http://mynamespace.org/cust*) ab. Da in der Abfrage mit **:* ein beliebiger Namensbereich angegeben ist, wird der Index nicht verwendet. In diesem Fall könnte sich der Benutzer von der intuitiven Annahme täuschen lassen, dass **:* dem Namensbereich in der Indexdefinition entspricht.

Damit die Abfrage den Index verwenden kann, muss entweder der Index weniger restriktiv oder die Abfrage restriktiver gefasst werden.

Der folgende weniger restriktive Index *customer_id_index2* könnte mit der gleichen Abfrage erfolgreich genutzt werden:

```
CREATE UNIQUE INDEX customer_id_index2 ON customer(xmlcol)
           GENERATE KEY USING XMLPATTERN '/*:Customer/@id' AS SQL DOUBLE %
```

Die folgende restriktivere Abfrage könnte den zuerst definierten Index *customer_id_index* nutzen:

```
SELECT xmlcol FROM customer
      WHERE XMLEXISTS('
      DECLARE NAMESPACE ns = "http://mynamespace.org/cust";
      $xmlcol/ns:Customer[@id=1042]'
      PASSING xmlcol AS "xmlcol") %
```

Wenn der entsprechende Namensbereich in der Abfrage explizit angegeben wird, kann der Index *customer_id_index* verwendet werden, da die Abfrage nun ebenso restriktiv ist wie der Index. Der Index *customer_id_index2* könnte ebenfalls verwendet werden, da er in diesem Beispiel weniger restriktiv ist als die Abfrage.

Aspekte bei der Angabe von Knoten vom Typ 'text()'

Die Angabe von Knoten vom Typ 'text()' in XML-Musterausdrücken kann die Generierung von Indexeinträgen beeinflussen. Verwenden Sie in Indexdefinitionen und Vergleichselementen durchgängig die Angabe '/text()'.

Auswirkungen der Angabe eines Knotens vom Typ 'text()' auf Indexschlüssel

Betrachten Sie das folgende Beispiel eines XML-Dokumentfragments:

```
<company name="Company1">
      <emp id="31201" salary="60000" gender="Female">
     <name><first>Laura</first><last>Brown</last></name>
     <dept id="M25">
   Finance
         </dept>
      </emp>
</company>
```

Wenn der folgende Index unter Angabe von 'text()' am Ende des Musters erstellt wird, werden keine Indexeinträge eingefügt, da die Elemente *name* in den zum Beispiel angenommenen XML-Dokumentfragmenten selbst keinen Text enthalten. Der Text befindet sich in den untergeordneten Elementen *first* und *last*.

```
CREATE INDEX nameindex on company(companydocs)
           GENERATE KEY USING XMLPATTERN '/company/emp/name/text()' AS SQL
           VARCHAR(30)
```

Wenn der nächste Index jedoch unter Angabe des Elements *name* am Ende des Musters erstellt wird, wird der Text aus den untergeordneten Elementen *first* und *last* in den eingefügten Indexeinträgen verkettet.

```
CREATE INDEX nameindex on company(companydocs)
           GENERATE KEY USING XMLPATTERN '/company/emp/name'
        AS SQL VARCHAR(30)
```

Die Angabe bzw. das Weglassen des Knotens 'text()' beeinflusst die Generierung von Indexeinträgen für Nichtblattknotenelemente, jedoch nicht für Blattknotenelemente. Wenn Sie Blattknotenelemente indexieren, wird die Angabe des Knotens 'text()' nicht empfohlen. Wenn Sie 'text()' angeben, müssen auch Abfragen 'text()' verwenden, um erfolgreich mit Indizes abgeglichen werden zu können. Darüber hinaus wird eine Schemaprüfung nur auf Elemente und nicht auf Textknoten angewendet.

Vorsicht ist geboten, wenn Sie XML-Muster angeben, die mit Elementen abgeglichen werden können, die Nichtblattknoten ohne 'text()' sind. Die Verkettung von untergeordneten Elementtextknoten kann zu unerwarteten Ergebnissen führen. Insbesondere führt die Angabe von //* in einem XML-Muster höchstwahrscheinlich zur Indexierung eines Nichtblattknotenelements.

In einigen Fällen kann die Verkettung für Indizes mit VARCHAR-Datentypen nützlich sein. Zum Beispiel kann ein Index für */title* im folgenden Dokumentfragment nützlich sein, um die Fettformatierung innerhalb des Titels zu ignorieren:

```
<title>Dies ist ein <bold>exzellentes</bold> Buch über XML</title>
```

Ein Abfragevergleichselement zur Suche nach einem bestimmten Mitarbeiternamen könnte wie folgt geschrieben werden:

```
db2-fn:xmlcolumn('COMPANY.COMPANYDOCS')/company/emp[name='LauraBrown']
```

Leerzeichen im Vergleichselement und im Dokument sind signifikant. Wenn zwischen 'Laura' und 'Brown' im Vergleichselement ein Leerzeichen eingefügt wird, wird für die folgende Abfrage nichts zurückgegeben, da das XML-Dokumentfragment des Beispiels selbst kein Leerzeichen zwischen dem Vor- und dem Nachnamen enthält:

```
db2-fn:xmlcolumn('COMPANY.COMPANYDOCS')/company/emp[name='Laura Brown']
```

Indizes für Abfragen mit einem zusammengesetzten Gleichheitsvergleichselement

Die folgende Abfrage ruft Unternehmensinformationen zu Mitarbeitern ab, die sowohl in der Abteilung 'Finance' als auch in der Abteilung 'Marketing' tätig sind:

```
SELECT companydocs FROM companyinfo WHERE
   XMLExists('$x/company/emp[dept/text()='Finance'
   or dept/text()='Marketing']'
      PASSING companydocs AS "x")
```

Um kompatibel zu sein, muss der Index für XML-Daten den Textknoten für die Abteilung jedes Mitarbeiters in die indexierten Knoten mit einschließen und Werte als VARCHAR-Datentyp speichern.

Die Abfrage kann den folgenden Index für XML-Daten nutzen:

```
CREATE INDEX empindex on companyinfo(companydocs)
      GENERATE KEY USING XMLPATTERN '/company/emp/dept/text()'
      AS SQL VARCHAR(30)
```

Datentypen von Literalen

Die Datentypen von Literalen müssen dem Datentyp des Index entsprechen, damit die Abfrage den Index nutzen kann.

Abgleich von Datentypen von Literalen

Die folgende Abfrage ruft Unternehmensinformationen zu Mitarbeitern mit der ID *31201* ab:

```
SELECT companydocs FROM companyinfo
   WHERE XMLEXISTS('$x/company/emp[@id="31201"]'
      PASSING companydocs AS "x")
```

Um kompatibel zu sein, muss der Index für XML-Daten die Attributknoten für die Mitarbeiter-ID in die indexierten Knoten mit einschließen und die Werte mit einem VARCHAR-Datentyp speichern.

```
CREATE INDEX empindex on companyinfo(companydocs)
      GENERATE KEY USING XMLPATTERN '/company/emp/@id'
      AS SQL VARCHAR(5)
```

Wenn ein ähnlicher Index mit AS SQL DOUBLE definiert würde, könnte dieser Index von der Abfrage nicht verwendet werden, da das Abfragevergleichselement einen Zeichenfolgevergleich enthält. Die doppelten Anführungszeichen im Vergleichselement *@id="31201"* machen den Ausdruck zu einem Zeichenfolgevergleich, der nur mithilfe eines Zeichenfolgeindex (VARCHAR), jedoch nicht mit einem numerischen Index (DOUBLE) ausgewertet werden kann.

Zur Verdeutlichung des Unterschieds zwischen numerischen Vergleichselementen und Zeichenfolgevergleichselementen betrachten Sie die folgenden Ungleichheitsvergleichselemente:

```
@id > 3
@id > "3"
```

Das numerische Vergleichselement *@id > 3* unterscheidet sich vom Zeichenfolgevergleichselement *@id > "3"*. Das numerische Vergleichselement *@id > 3* würde von einem *@id*-Wert *10* erfüllt, während das Zeichenfolgevergleichselement *@id > "3"* durch diesen Wert nicht erfüllt würde, da bei einem Zeichenfolgevergleich *"3"* größer ist als *"10"*.

Konvertierung von Joinvergleichselementen

Joinvergleichselemente sollten auf beiden Seiten in den richtigen Datentyp konvertiert werden.

Welche Joinvergleichselemente können die Verwendung von Indizes ausschließen?

Betrachten Sie zwei Tabellen mit jeweils einer XML-Spalte für Kundeninformationen bzw. Bestellungen:

```
CREATE TABLE customer(info XML);

CREATE TABLE PurchaseOrder(POrder XML);
```

Die XML-Dokumente, die Kundeninformationen beinhalten, enthalten ein Attribut '@cid' für die numerische Kunden-ID (*cid*). Die XML-Dokumente, die Bestellinformationen enthalten, haben ebenfalls Attribute *@cid*, sodass jede Bestellung einem bestimmten Kunden eindeutig zugeordnet werden kann. Da zu erwarten ist, dass Kunden und Bestellungen häufig über das Attribut *cid* gesucht werden, ist es sinnvoll, Indizes zu definieren:

```
CREATE UNIQUE INDEX idx1 ON customer(info)
   GENERATE KEY USING XMLPATTERN '/customerinfo/@cid' AS SQL INTEGER;

CREATE INDEX idx2 ON PurchaseOrder(POrder)
   GENERATE KEY USING XMLPATTERN '/porder/@cid' AS SQL INTEGER;
```

Nun sollen die Bestellungen für alle Kunden mit einer bestimmten Postleitzahl ('zipcode') ermittelt werden. Intuitiv ließe sich die Abfrage vielleicht wie folgt schreiben:

```
XQUERY
      for $i in db2-fn:xmlcolumn("CUSTOMER.INFO")/customerinfo
      for $j in db2-fn:xmlcolumn("PURCHASEORDER.PORDER")/porder[@cid = $i/@cid]
      where $i/zipcode = "95141"
      return $j;
```

Beachten Sie, dass das Joinvergleichselement @cid = $i/@cid fordert, dass die Kunden-ID *cid* der Bestellung mit der Kunden-ID *cid* des Kunden übereinstimmt.

Diese Abfrage gibt das korrekte Ergebnis zurück, jedoch kann keiner der beiden Indizes verwendet werden. Die Abfrage wird als Join mit Verschachtelungsschleife (Nested Loop Join) mit Tabellensuchen für beide Tabellen ausgeführt. Zur Vermeidung wiederholter Tabellensuchen ist eine einzige Tabellensuche in *customer* zur Ermittlung aller Kunden mit der Postleitzahl ('zipcode') *95141* vorzuziehen, an die sich Indexsuchen für die Tabelle 'PurchaseOrder' mithilfe des Elements '@cid' anschließen. Beachten Sie, dass ein Durchsuchen der Tabelle 'customer' erforderlich ist, da kein Index für *zipcode* erstellt wurde.

Der Index wird nicht verwendet, da dies eine falsche Funktionsweise darstellen würde. Wenn der Index verwendet würde, könnte DB2 einige dem Vergleichselement entsprechende Bestellungen übersehen und ein unvollständiges Ergebnis zurückgeben. Das liegt daran, dass einige Werte im Attribut '@cid' potenziell nicht numerisch sein könnten. Zum Beispiel könnte @cid gleich *YPS* sein und deshalb nicht in den numerischen Index eingefügt worden sein, der mit AS SQL INTEGER definiert wurde.

Anmerkung: Wenn ein Wert eines indexierten Knotens nicht in den angegebenen Indexdatentyp konvertiert werden kann, wird der Indexeintrag für diesen Wert nicht eingefügt und keine Fehlernachricht oder Warnung ausgegeben.

Ermöglichen der Indexverwendung mit Joinvergleichselementen

Es ist möglich, den Index zu verwenden, was dann wünschenswert ist, wenn feststeht, dass alle @cid-Werte numerisch sind. Der Index wird verwendet, wenn das Joinvergleichselement explizit konvertiert wird, sodass es dem Typ des Index entspricht:

```
XQUERY
      for $i in db2-fn:xmlcolumn("CUSTOMER.INFO")/customerinfo
      for $j in db2-fn:xmlcolumn("PURCHASEORDER.PORDER")/porder
   where $i/@cid/xs:int(.) = $j/@cid/xs:int(.)
      and $i/zipcode = "95141"
      return $j;
```

Intuitiv wird deutlich, dass die Konvertierung DB2 anweist, für den Abgleich nur solche Attribute vom Typ '@cid' zu berücksichtigen, die in INTEGER konvertierbar sind. Mit dieser Anweisung lässt sich sicherstellen, dass alle erforderlichen Übereinstimmungen in dem Index, der mit AS SQL INTEGER definiert ist, dargestellt werden, sodass die Verwendung dieses Index sicher ist. Wenn in einem der Dokumente ein nicht numerischer Wert für '@cid' enthalten ist, schlägt die Konvertierung mit einem Laufzeitfehler fehl.

Beachten Sie, dass innerhalb von XQuery die explizite Typumsetzung nur für Singletons (Einzelwerte) funktioniert. Insbesondere wird empfohlen, Elemente (*a*,*b* und *c* im folgenden Beispiel) wie folgt zu konvertieren:

```
/a/b/c/xs:double(.)
```

Wenn Sie die Elemente wie folgt konvertieren würden, träte ein Laufzeitfehler auf, wenn mehrere Elemente *c* unter einem beliebigen Element *b* vorhanden wären:

```
/a/b/xs:double(c)
```

Für Indizes, die mit AS SQL VARCHAR definiert sind, müssen die entsprechenden Joinvergleichselemente den verglichenen Wert mit der Funktion fn:string() in den Datentyp 'xs:string' konvertieren. Dasselbe gilt analog für DATE- und TIMESTAMP-Indizes. Das folgende Beispiel zeigt, wie die Funktion fn:string() in einem Zeichenfolgejoin verwendet wird:

```
XQUERY
      for $i in db2-fn:xmlcolumn("CUSTOMER.INFO")/customerinfo
      for $j in db2-fn:xmlcolumn("PURCHASEORDER.PORDER")/porder
      where $i/zipcode/fn:string(.) = $j/supplier/zip/fn:string(.)
      return <pair>{$i}{$j}</pair>
```

Zusammenfassung der Konvertierungsregeln für Joinvergleichselemente

Die folgende Tabelle enthält eine Zusammenfassung dazu, wie Joinvergleichselemente auf beiden Seiten in den richtigen Datentyp konvertiert werden sollten, um die Verwendung des Index zu ermöglichen.

Tabelle 16. Konvertierungsregeln für Joinvergleichselemente

SQL-Datentyp des Index	XML-Datentyp, in den das Joinvergleichselement zu konvertieren ist
DOUBLE	xs:double
DECIMAL	xs:decimal
INTEGER	xs:int
VARCHAR *integer*, VARCHAR HASHED	xs:string
DATE	xs:date
TIMESTAMP	xs:dateTime

Unbestimmte Abfrageauswertung

Eine Abfrage kann unbestimmt ausgewertet werden und einen Fehler zurückliefern, wenn keine Indexsuche erfolgt. Dieselbe Abfrage liefert möglicherweise entsprechende XML-Daten ohne Fehler, wenn die Abfrageauswertung eine Indexsuche beinhaltet, da nicht umsetzbare XML-Fragmente, die den Fehler verursachen, im Index nicht enthalten sind.

Im Beispiel wird die folgende Anweisung VALUES verwendet, die einen XQuery-Ausdruck enthält. Der Ausdruck gibt mithilfe eines Vergleichs numerischer Werte das XML-Dokument mit der ID 17 zurück:

```
VALUES( XMLQUERY('
  for $i in db2-fn:xmlcolumn("T.DOC")
      where $i/emp/id = 17
   return $i'))
```

Die Tabelle `T` umfasst eine einzelne XML-Spalte `DOC` und enthält zwei XML-Dokumente. Die folgenden Anweisungen dienen zum Erstellen der Tabelle und zum Einfügen der Dokumente:

```
CREATE TABLE t (doc XML) ;
INSERT INTO t VALUES ( '<emp><id>17</id></emp>' );
INSERT INTO t VALUES ( '<emp><id>ABC</id></emp>' );
```

Der XQuery-Ausdruck gibt einen Fehler zurück, sofern das übereinstimmende Dokument in der Tabelle nicht über einen als SQL INTEGER oder SQL DOUBLE definierten Index für XML-Daten gesucht wird. Der Index wird mit der folgenden Anweisung vom Typ CREATE INDEX erstellt:

```
CREATE INDEX EMPDBL ON t (doc) GENERATE KEY USING XMLPATTERN '/emp/id'
    as SQL INTEGER IGNORE INVALID VALUES
```

Wenn der Index nicht in der Tabelle vorhanden ist, wird der Vergleich numerischer Werte in der Klausel **WHERE** sowohl auf das übereinstimmende als auch auf das nicht übereinstimmende Dokument in der Tabelle angewendet. Wird der Vergleich auf das nicht übereinstimmende Dokument angewendet, wird ein Laufzeitfehler verursacht, weil der Wert ABC nicht in einen numerischen Wert konvertiert werden kann. Wenn der Index für XML-Daten, mit dem Mitarbeiter-IDs indexiert werden, in der Tabelle vorhanden ist, verwendet der Ausdruck den Index und gibt das erste XML-Dokument ohne Fehler zurück. Der nicht umsetzbare Wert im zweiten Dokument war nicht im Index für XML-Daten enthalten, weil die Klausel IGNORE INVALID VALUES angibt, dass ungültige Musterwerte bei der Indexerstellung nicht indexiert, sondern ignoriert werden. Standardmäßig werden ungültige Werte ignoriert.

Der von der EXPLAIN-Einrichtung bereitgestellte Zugriffsplan zeigt, ob die Auswertung einer Abfrage mithilfe einer Indexsuche erfolgt.

Verwenden Sie die Klausel REJECT INVALID VALUES, um sicherzustellen, dass alle Musterwerte während der Indexerstellung gültig sind. Treten bei der Erstellung oder Aktualisierung eines Index ungültige Musterwerte auf, wird ein Fehler zurückgegeben.

Volltextsuche in XML-Dokumenten

Die Volltextsuche in nativ gespeicherten XML-Daten wird durch DB2 Net Search Extender bereitgestellt.

DB2 Net Search Extender

DB2 Net Search Extender bietet eine vollständige Unterstützung für den Datentyp XML. Das Produkt ermöglicht eine Volltextindexierung von Dokumenten, die in XML-Spalten gespeichert sind. Durch die Erstellung eines Textindex zu einer XML-Spalte können Sie jeden Text in einem XML-Dokument abfragen und Suchoperationen mit Kriterien wie räumliche Nähe oder Platzhaltersuchen durchführen. DB2 Net Search Extender ist Teil aller DB2-Datenserverprodukte für Linux, UNIX und Windows, muss jedoch separat installiert werden.

Das folgende Beispiel zeigt eine einfache Volltextsuche, die das Wort "marketing" an jeder beliebigen Stelle im Pfad '/dept/description' von XML-Dokumenten findet, die in der Spalte DEPTDOC gespeichert sind:

```
SELECT DEPTDOC
FROM DEPT
WHERE contains (DEPTDOC, SECTIONS("/dept/description") "marketing") = 1
```

Die Funktion 'contains', die von DB2 Net Search Extender bereitgestellt wird, sucht nach der Zeichenfolge "marketing" in jedem Text unter dem Pfad '/dept/description', einschließlich Element- oder Attributnamen und Element- oder Attributwerten.

Zur Nutzung von Volltextsuchen muss SQL verwendet werden. Die Ergebnisse aus der SQL-Abfrage können trotzdem an einen XQuery-Kontext zur weiteren Verarbeitung zurückgegeben werden. Das folgende Beispiel zeigt, wie die Ergebnisse aus einer SQL-Abfrage, die eine Volltextsuche verwendet hat, in einen XQuery-Ausdruck integriert werden können:

```
XQUERY for $i in db2-fn:sqlquery ('SELECT DEPTDOC FROM DEPT
  WHERE contains
    (DEPTDOC, SECTIONS("/dept/description") "marketing") = 1')//employee
  return $i/name
```

In diesem Beispiel werden die Ergebnisse der SQL-Abfrage, die eine Volltextsuche ausgeführt hat, an die Klausel 'for' des XQuery-Ausdrucks FLWOR zurückgegeben. Die Klausel 'for' gibt anschließend alle Elemente <employee> zurück, und die Klausel 'return' gibt die Elemente <name> innerhalb der abgerufenen Elemente <employee> zurück.

Weitere Informationen zu DB2 Net Search Extender finden Sie in der DB2 Net Search Extender-Dokumentation oder im World Wide Web unter www.ibm.com/software/data/db2/extenders/netsearch.

Abrufen von Daten in XML-Spalten für DB2-Clients früherer Versionen

Wenn Sie Daten aus einer XML-Spalte für einen Client abrufen, der ein früheres Release-Level als DB2 Version 9.1 aufweist, kann Ihr Datenbankclient keine XML-Daten verarbeiten.

Wird bei der DRDA-Verarbeitung ein Client erkannt, der keine XML-Daten unterstützt, beschreibt der DB2-Datenbankserver XML-Datenwerte standardmäßig als BLOB-Werte und sendet die Daten als BLOB-Daten an den Client. Die BLOB-Daten sind eine serialisierte Zeichenfolgedarstellung der XML-Daten mit einer vollständigen XML-Deklaration.

Wenn Sie die Daten mit einem anderen Datentyp als BLOB empfangen wollen, verwenden Sie eine der folgenden Methoden:

- Zum Abrufen der Daten als CLOB-Daten, bitten Sie den Administrator des Datenbankservers, die Registrierdatenbankvariable DB2_MAP_XML_AS_CLOB_FOR_DLC auf dem Server mithilfe des Befehls 'db2set' auf den Wert YES zu setzen.

  Wichtig: Wenn Sie die Registrierdatenbankvariable DB2_MAP_XML_AS_CLOB_FOR_DLC auf dem Datenbankserver auf den Wert YES setzen, empfangen **alle** DB2-Clients mit einem früheren Release-Level, die eine Verbindung zu einer Datenbank in der Instanz herstellen, XML-Daten als CLOB-Daten.

  Wichtig: Wenn die Registrierdatenbankvariable DB2_MAP_XML_AS_CLOB_FOR_DLC auf dem Datenbankserver auf den Wert YES gesetzt ist, empfangen Clients CLOB-Daten in Form einer serialisierten Zeichenfolgedarstellung der XML-Daten **ohne** XML-Deklaration.
- Zum Abrufen der Daten als CLOB-, CHAR- oder VARCHAR-Daten rufen Sie die Funktion XMLSERIALIZE für die Spaltendaten auf, um den DB2-Datenbankserver anzuweisen, die Daten in den angegebenen Datentyp umzuwandeln, bevor er sie an den Client sendet.

Wenn Sie die Funktion XMLSERIALIZE zum Abrufen von Daten aus einem Datenbankserver auf einen Client eines früheren Release-Levels nicht aufrufen, verhält sich die Spalte, aus der Sie die Daten abrufen, nicht exakt wie eine BLOB- oder CLOB-Spalte. Zum Beispiel können Sie das Vergleichselement LIKE für eine BLOB-Spalte verwenden, jedoch nicht für eine XML-Spalte, aus der BLOB- oder CLOB-Daten zurückgegeben werden.

SQL/XML-Veröffentlichungsfunktionen für das Erstellen von XML-Werten

Sie können XML-Werte erstellen, die nicht unbedingt korrekt formatierte XML-Dokumente sein müssen, indem Sie diejenigen Veröffentlichungsfunktionen kombinieren, die den Komponenten entsprechen, die Sie im XML-Ergebniswert haben möchten. Die Funktionen müssen in der Reihenfolge angegeben werden, in der die Ergebnisse angeordnet sein sollen.

Werte, die mit den SQL/XML-Veröffentlichungsfunktionen konstruiert werden, werden als XML-Daten zurückgegeben. In Abhängigkeit von der weiteren Verwendung eines XML-Wertes muss der Wert möglicherweise explizit serialisiert werden, um ihn in einen anderen SQL-Datentyp umzuwandeln. Detaillierte Informationen finden Sie in der Dokumentation zur Serialisierung von XML-Daten.

Die folgenden SQL/XML-Veröffentlichungsfunktionen können verwendet werden, um XML-Werte zu erstellen. Eine Beschreibung der Syntax für die jeweilige Funktion finden Sie in Anhang B, „SQL/XML-Veröffentlichungsfunktionen", auf Seite 487:

XMLAGG, Spaltenfunktion
Gibt eine XML-Sequenz zurück, die ein Element für jeden Wert ungleich null in einer Menge von XML-Werten enthält.

XMLATTRIBUTES, Skalarfunktion
Konstruiert XML-Attribute aus den Argumenten. Diese Funktion kann nur als Argument der Funktion XMLELEMENT verwendet werden.

XMLCOMMENT, Skalarfunktion
Gibt einen XML-Wert mit nur einem XQuery-Kommentarknoten und dem Eingabeargument als Inhalt zurück.

Skalarfunktion XMLCONCAT
Gibt eine Sequenz mit der Verkettung einer variablen Anzahl von XML-Eingabeargumenten zurück.

XMLDOCUMENT, Skalarfunktion
Gibt einen XML-Wert mit einem einzigen XQuery-Dokumentknoten mit null oder mehr untergeordneten Knoten zurück. Diese Funktion erstellt einen Dokumentknoten, den jedes XML-Dokument laut Definition haben muss. Ein Dokumentknoten ist in der serialisierten Darstellung von XML nicht sichtbar. Jedoch muss jedes Dokument, das in einer DB2-Tabelle gespeichert werden soll, einen Dokumentknoten enthalten.

XMLELEMENT, Skalarfunktion
Gibt einen XML-Wert zurück, der einen XML-Elementknoten darstellt.Beachten Sie, dass die Funktion XMLELEMENT keinen Dokumentknoten, sondern nur einen Elementknoten erstellt. Beim Erstellen von XML-Dokumenten, die eingefügt werden sollen, reicht es nicht aus, nur einen Elementknoten zu erstellen. Das Dokument muss einen Dokumentknoten enthalten, der mit der Funktion XMLDOCUMENT erstellt wurde.

XMLFOREST, Skalarfunktion
Gibt einen XML-Wert zurück, der eine Sequenz von XML-Elementknoten darstellt.

XMLGROUP, Spaltenfunktion
Gibt ein einzelnes Element auf oberster Ebene zurück, das eine Tabelle oder das Ergebnis einer Abfrage darstellt. Standardmäßig wird jede Zeile

in der Ergebnismenge einem Zeilenunterelement zugeordnet, und jeder Eingabeausdruck wird einem Unterelement des Zeilenunterelements zugeordnet. Optional kann jede Zeile im Ergebnis einem Zeilenunterelement zugeordnet werden, und jeder Eingabeausdruck kann einem Attribut des Zeilenunterelements zugeordnet werden.

XMLNAMESPACES, Deklaration
Konstruiert Namensbereichsdeklarationen aus den Argumenten. Diese Deklaration kann nur als Argument der Funktionen XMLELEMENT, XMLFOREST und XMLTABLE verwendet werden.

XMLPI, Skalarfunktion
Gibt einen XML-Wert mit nur einem XQuery-Verarbeitungsanweisungsknoten zurück.

XMLROW, Skalarfunktion
Gibt eine Sequenz aus Zeilenelementen zurück, die eine Tabelle oder das Ergebnis einer Abfrage darstellt. Standardmäßig wird jeder Eingabeausdruck in ein Unterelement eines Zeilenelements umgesetzt. Optional kann jeder Eingabeausdruck in ein Attribut eines Zeilenelements umgesetzt werden.

XMLTEXT, Skalarfunktion
Gibt einen XML-Wert mit nur einem XQuery-Textknoten mit dem Eingabeargument als Inhalt zurück.

XSLTRANSFORM, Skalarfunktion
Wandelt XML-Daten in andere Formate (einschließlich anderer XML-Schemata) um.

Nullwerte in Elementen

Wenn ein XML-Wert mithilfe der Funktion XMLELEMENT oder XMLFOREST konstruiert wird, ist es möglich, dass bei der Ermittlung des Inhalts des Elements ein Nullwert festgestellt wird. Durch die Optionen EMPTY ON NULL und NULL ON NULL der Funktionen XMLELEMENT und XMLFOREST können Sie angeben, ob ein leeres Element oder kein Element generiert wird, wenn der Inhalt eines Elements null ist. Die Standardbehandlung von Nullwerten für XMLEXISTS ist EMPTY ON NULL. Die Standardbehandlung von Nullwerten für XMLFOREST ist NULL ON NULL.

Beispiele für das Veröffentlichen von XML-Werten

Die folgenden Beispiele zeigen, wie Sie XML-Werte mit SQL/XML-Veröffentlichungsfunktionen (Publishing) und mit XQuery-Ausdrücken erstellen können.

Beispiel: Konstruktion eines XML-Dokuments mit konstanten Werten

Dieses einfache Beispiel zeigt, wie Sie konstante XML-Werte erstellen können, die für das Veröffentlichen mit SQL/XML-Veröffentlichungsfunktionen (Publishing) geeignet sind.

Betrachten Sie das folgende einfache Beispiel eines XML-Elements:

```
<elem1 xmlns="http://posample.org" id="111">
  <!-- example document -->
    <child1>abc</child1>
    <child2>def</child2>
</elem1>
```

Das Dokument setzt sich aus folgenden Bestandteilen zusammen:

- Drei Elementknoten (elem1, child1 und child2)
- Eine Namensbereichsdeklaration
- Ein Attribut "id" im Element <elem1>
- Einen Kommentarknoten

Zur Konstruktion dieses Dokuments führen Sie die folgenden Schritte aus:

1. Erstellen Sie einen Elementknoten mit dem Namen "elem1" mithilfe von XMLELEMENT.
2. Fügen Sie mithilfe von XMLNAMESPACES dem Aufruf der Funktion XMLELEMENT eine Standardnamensbereichsdeklaration für <elem1> hinzu.
3. Erstellen Sie mithilfe von XMLATTRIBUTES ein Attribut mit dem Namen "id", und platzieren Sie es hinter der XMLNAMESPACES-Deklaration.
4. Erstellen Sie innerhalb des Aufrufs der Funktion XMLELEMENT mithilfe von XMLCOMMENT einen Kommentarknoten für <elem1>.
5. Erstellen Sie innerhalb des Aufrufs der Funktion XMLELEMENT mithilfe der Funktion XMLFOREST eine Struktur aus Elementen mit den Namen "child1" und "child2" für <elem1>.

Diese Schritte werden zu folgender Abfrage kombiniert:

```
VALUES XMLELEMENT (NAME "elem1",
       XMLNAMESPACES (DEFAULT 'http://posample.org'),
       XMLATTRIBUTES ('111' AS "id"),
       XMLCOMMENT ('example document'),
       XMLFOREST('abc' as "child1",
                 'def' as "child2"))
```

Beispiel: Konstruktion eines XML-Dokuments mit Werten aus nur einer Tabelle

Dieses Beispiel zeigt, wie Sie konstante XML-Werte erstellen können, die für das Veröffentlichen über eine einzige Tabelle mit SQL/XML-Veröffentlichungsfunktionen (Publishing) geeignet sind.

Dieses Beispiel zeigt, wie sich ein XML-Dokument aus Werten konstruieren lässt, die in einer Tabelle gespeichert sind. In der folgenden Abfrage wird jedes Element <item> mithilfe der Funktion XMLELEMENT aus Werten der Spalte 'name' der Tabelle PRODUCT konstruiert. Alle Elemente <item> werden anschließend durch die Funktion XMLAGG im konstruierten Element <allProducts> zusammengefasst.

```
SELECT XMLELEMENT (NAME "allProducts",
       XMLAGG(XMLELEMENT (NAME "item", p.name)))
FROM Product p
```

```
<allProducts>
  <item>Snow Shovel, Basic 22 inch</item>
  <item>Snow Shovel, Deluxe 24 inch</item>
  <item>Snow Shovel, Super Deluxe 26 inch</item>
  <item>Ice Scraper, Windshield 4 inch</item>
</allProducts>
```

Sie können ein ähnliches XML-Dokument erstellen, das eine Sequenz aus Zeilenelementen enthält, indem Sie die Funktion XMLROW verwenden, anstatt die Elemente mit XMLAGG zusammenzufassen.

```
SELECT XMLELEMENT (NAME "products",
                   XMLROW(NAME as "po:item"))
FROM Product
```

Die resultierende Ausgabe sieht wie folgt aus:

```
<products>
   <4row>
      <po:item>Snow Shovel, Basic 22 inch</po:item>
   </row>
</products>
<products>
   <row>
      <po:item>Snow Shovel, Deluxe 24 inch</po:item>
   </row>
</products>
<products>
      <row><po:item>Snow Shovel, Super Deluxe 26 inch</po:item>
   </row>
</products>
<products>
      <row><po:item>Ice Scraper, Windshield 4 inch</po:item>
   </row>
</products>

  4 Satz/Sätze ausgewählt.
```

Beispiel: Konstruktion eines XML-Dokuments mit Werten aus mehreren Tabellen

Dieses Beispiel zeigt, wie Sie XML-Werte erstellen können, die für das Veröffentlichen über mehrere Tabellen mit SQL/XML-Veröffentlichungsfunktionen (Publishing) geeignet sind.

Dieses Beispiel zeigt, wie sich ein XML-Dokument aus Werten konstruieren lässt, die in mehreren Tabellen gespeichert sind. In der folgenden Abfrage werden mithilfe der Funktion XMLFOREST Elemente vom Typ <prod> aus einer Gesamtstruktur von Elementen mit den Namen 'name' und 'numInStock' erstellt. Diese Struktur wird aus Werten der Tabellen PRODUCT und INVENTORY aufgebaut. Alle Elemente <prod> werden anschließend im konstruierten Element <saleProducts> zusammengefasst.

```
SELECT XMLELEMENT (NAME "saleProducts",
       XMLAGG (XMLELEMENT (NAME "prod",
       XMLATTRIBUTES (p.Pid AS "id"),
       XMLFOREST (p.name as "name",
                  i.quantity as "numInStock"))))
FROM PRODUCT p, INVENTORY i
WHERE p.Pid = i.Pid
```

Die obige Abfrage liefert das folgende XML-Dokument:

```
<saleProducts>
    <prod id="100-100-01">
    <name>Snow Shovel, Basic 22 inch</name>
    <numInStock>5</numInStock>
  </prod>
    <prod id="100-101-01">
    <name>Snow Shovel, Deluxe 24 inch</name>
    <numInStock>25</numInStock>
  </prod>
    <prod id="100-103-01">
    <name>Snow Shovel, Super Deluxe 26 inch</name>
    <numInStock>55</numInStock>
  </prod>
    <prod id="100-201-01">
    <name>Ice Scraper, Windshield 4 inch</name>
    <numInStock>99</numInStock>
  </prod>
</saleProducts>
```

Beispiel: Erstellen eines XML-Dokuments mit Werten aus Tabellenzeilen mit Nullelementen

Dieses Beispiel zeigt, wie Sie XML-Werte erstellen können, die für das Veröffentlichen über Tabellenzeilen, die Nullelemente enthalten, mit SQL/XML-Veröffentlichungsfunktionen (Publishing) geeignet sind.

In diesem Beispiel wird davon ausgegangen, dass die Spalte LOCATION der Tabelle INVENTORY in einer Zeile einen Nullwert enthält. Die folgende Abfrage gibt daher das Element <loc> nicht zurück, weil die Funktion XMLFOREST Nullwerte standardmäßig wie null behandelt:

```
SELECT XMLELEMENT (NAME "newElem",
                                      XMLATTRIBUTES (PID AS "prodID"),
                                      XMLFOREST (QUANTITY as "quantity",
                                                 LOCATION as "loc"))
FROM INVENTORY
```

```
<newElem prodID="100-100-01"><quantity>5</quantity></newElem>
```

Die gleiche Abfrage mit Angabe der Option EMPTY ON NULL gibt ein leeres Element <loc> zurück:

```
SELECT XMLELEMENT (NAME "newElem",
                                      XMLATTRIBUTES (PID AS "prodID"),
                                      XMLFOREST (QUANTITY as "quantity",
                                                 LOCATION as "loc" OPTION EMPTY ON NULL))
FROM INVENTORY
```

```
<newElem prodID="100-100-01"><quantity>5</quantity><loc/></newElem>
```

Beispiel: Veröffentlichen von Daten mit XQuery

Dieses Beispiel zeigt, wie Sie XML-Werte erstellen können, die für das Veröffentlichen sowohl mit SQL/XML-Veröffentlichungsfunktionen (Publishing) als auch mit XQuery-Ausdrücken geeignet sind.

In dem nachstehenden XQuery-Ausdruck wird der Aktualisierungsausdruck für Löschen (DELETE) verwendet, um ein einfaches Kundenverzeichnis zu erstellen. Der Ausdruck entfernt die Kunden-ID, die Kundenadresse, zusätzliche Informationen und die privaten Telefonnummern (nicht die Telefonnummern vom Typ 'work') aus den Kundeninformationen (customerinfo) und versetzt das Länderattribut `country` aus dem Knotenelement für Adresse (`addr`) in das Knotenelement für die Kundeninformationen.

```
  xquery
<phonelist>
  {for $d in db2-fn:xmlcolumn("CUSTOMER.INFO")/customerinfo
  return
    transform
    copy $mycust := $d
    modify (
      do delete ( $mycust/@Cid ,
        $mycust/addr ,
        $mycust/assistant ,
        $mycust/phone[@type!="work"] ),
      do insert attribute country { $mycust/addr/@country } into $mycust )
    return $mycust }
</phonelist>
```

Bitte beachten Sie, dass trotz Löschung des Adresselements `addr` die Adressinformationen innerhalb der Änderungsklausel **`modify`** verfügbar sind und dass das Länderattribut `country` des Adresselements `addr` vom Ausdruck für Einfügen (insert) verwendet wird.

Die Abfrage gibt das folgende Ergebnis zurück:

```
<phonelist>
<customerinfo country ="Canada">
  <name>Kathy Smith</name>
  <phone type="work">416-555-1358</phone>
 </customerinfo>
<customerinfo country ="Canada">
  <name>Kathy Smith</name>
  <phone type="work">905-555-7258</phone>
 </customerinfo>
<customerinfo country ="Canada">
<name>Jim Noodle</name>
  <phone type="work">905-555-7258</phone>
 </customerinfo>
<customerinfo country ="Canada">
  <name>Robert Shoemaker</name>
  <phone type="work">905-555-7258</phone>
 </customerinfo>
<customerinfo country ="Canada">
<name>Matt Foreman</name>
<phone type="work">905-555-4789</phone>
 </customerinfo>
<customerinfo country ="Canada">
<name>Larry Menard</name>
  <phone type="work">905-555-9146</phone>
 </customerinfo>
</phonelist>
```

Behandlung von Sonderzeichen in SQL/XML-Veröffentlichungsfunktionen

SQL/XML-Veröffentlichungsfunktionen weisen bei der Verarbeitung von Sonderzeichen ein Standardverhalten auf.

SQL-Werte in XML-Werte

Bestimmte Zeichen werden in XML-Dokumenten als Sonderzeichen betrachtet und müssen durch das entsprechende vordefinierte Entitätselement in ihrem Escape-Format dargestellt werden. Diese Sonderzeichen sind folgende:

Tabelle 17. Sonderzeichen und ihre darstellenden Entitätselemente

Sonderzeichen	Darstellendes Entitätselement
<	&lt;
>	&gt;
&	&amp;
"	&quot;

Bei der Veröffentlichung von SQL-Werten als XML-Werte mithilfe der SQL/XML-Veröffentlichungsfunktionen werden diese Sonderzeichen in ihr Escape-Format umgesetzt und durch die entsprechenden vordefinierten Entitätselemente ersetzt.

SQL-Kennungen und qualifizierte Namen (QNames)

Bei der Veröffentlichung oder Konstruktion von XML-Werten aus SQL-Werten kann es erforderlich werden, eine SQL-Kennung einem qualifizierten XML-Namen (QName) zuzuordnen. Die Gruppe von Zeichen, die in SQL-Kennungen mit Begrenzern zulässig sind, unterscheidet sich jedoch von den Zeichen, die in einem qualifizierten Namen (QName) zulässig sind. Diese Unterschieden bedeuten, dass einige Zeichen, die in SQL-Kennungen verwendet werden, in qualifizierten Namen

nicht gültig sind. Diese Zeichen werden im qualifizierten Namen infolgedessen durch ihr darstellendes Entitätselement ersetzt.

Betrachten Sie zum Beispiel die mit Begrenzern angegebene SQL-Kennung "phone@work". Da das Zeichen @ in einem QName kein gültiges Zeichen ist, wird das Zeichen in sein Escape-Format umgesetzt, sodass sich der folgende QName ergibt: phone@work.

Beachten Sie, dass diese Standardfunktionsweise der Umsetzung in das Escape-Format nur für Spaltennamen gilt. Für SQL-Kennungen, die als Elementnamen in XMLELEMENT-Funktionen oder als Aliasnamen in der AS-Klausel von XMLFOREST- und XMLATTRIBUTES-Funktionen angegeben werden, sind keine Escape-Standardwerte vorhanden. In diesen Fällen müssen Sie gültige qualifizierte Namen (QNames) angeben. Weitere detaillierte Informationen zu gültigen Namen finden Sie in W3C-XML-Namensbereichsspezifikationen.

XML-Serialisierung

Als XML-Serialisierung wird der Prozess bezeichnet, mit dem XML-Daten aus ihrer Darstellung im XQuery- und XPath-Datenmodell (bei dem es sich um das hierarchische Format handelt, in dem diese Daten in einer DB2-Datenbank gespeichert werden) in das serialisierte Zeichenfolgeformat umgesetzt werden, das in einer Anwendung verwendet wird.

Die Serialisierung kann vom DB2-Datenbankmanager implizit ausgeführt werden. Alternativ hierzu können Sie auch die Funktion XMLSERIALIZE aufrufen, um die XML-Serialisierung explizit anzufordern. Am häufigsten wird die XML-Serialisierung eingesetzt, wenn XML-Daten vom Datenbankserver an den Client gesendet werden sollen.

Die implizite Serialisierung stellt in den meisten Fällen die bevorzugte Methode dar, da sie einfacher zu codieren ist. Darüber hinaus ist beim Senden von XML-Daten an den DB2-Client sichergestellt, dass dieser die Daten korrekt verarbeiten kann. Bei der expliziten Serialisierung sind zusätzliche Verarbeitungsschritte erforderlich. Während der impliziten Serialisierung werden diese Schritte vom Client automatisch ausgeführt.

Im Allgemeinen ist die implizite Serialisierung zu bevorzugen, weil es effizienter ist, die Daten im XML-Format an den Client zu senden. In bestimmten Situationen, die später erläutert werden, ist jedoch eine explizite Serialisierung mit XMLSERIALIZE vorzuziehen.

Am besten sollten XML-Daten in den BLOB-Datentyp umgewandelt werden, weil beim Abrufen binärer Daten ein geringeres Risiko für Codierungsfehler besteht.

Implizite XML-Serialisierung

Bei der impliziten Serialisierung liegen die Daten im XML-Format vor, wenn sie an den Client gesendet werden, sofern dieser den Datentyp XML unterstützt. Bei Anwendungen der CLI und eingebetteten SQL-Anwendungen fügt der DB2-Datenbankserver eine XML-Deklaration mit der entsprechenden Codierungsangabe zu den Daten hinzu. Bei Java- und .NET-Anwendungen fügt der DB2-Datenbankserver keine XML-Deklaration hinzu. Wenn die Daten jedoch für ein `DB2Xml`-Objekt abgerufen und bestimmte Methoden zum Abrufen dieser Daten aus diesem Objekt verwendet werden, fügt IBM Data Server Driver for JDBC and SQLJ eine XML-Deklaration hinzu.

Beispiel: In einem C-Programm müssen Sie das Dokument 'customerinfo' für die Kunden-ID (Cid) '1000' implizit serialisieren und das serialisierte Dokument dann in eine binäre XML-Hostvariable abrufen. Die abgerufenen Daten sind im UTF-8-Codierungsschema angegeben und enthalten eine XML-Deklaration.

```
EXEC SQL BEGIN DECLARE SECTION;
  SQL TYPE IS XML AS BLOB (1M) xmlCustInfo;
EXEC SQL END DECLARE SECTION;
...
EXEC SQL SELECT INFO INTO :xmlCustInfo
  FROM Customer    WHERE Cid=1000;
```

Explizite XML-Serialisierung

Nach einem expliziten XMLSERIALIZE-Aufruf sind die Daten auf dem Datenbankserver in einem Nicht-XML-Format codiert. Sie werden mit diesem Datentyp auch an den Client gesendet.

Bei XMLSERIALIZE können Sie die folgenden Informationen angeben:

- SQL-Datentyp, in den die Daten bei der Serialisierung umgewandelt werden sollen.

  Es kann entweder ein Zeichendatentyp oder ein binärer Datentyp verwendet werden.
- Festlegung dazu, ob die Ausgabedaten die folgende explizite Codierungsangabe (EXCLUDING XMLDECLARATION oder INCLUDING XMLDECLARATION) enthalten sollen:

  ```
  <?xml version="1.0" encoding="UTF-8" ?>
  ```

Bei XMLSERIALIZE werden die Ausgabedaten im Unicode-UTF-8--Format generiert.

Wenn Sie die serialisierten Daten in einem nicht binären Datentyp abrufen, werden diese in die Codierung der Anwendung umgesetzt, die Codierungsangabe wird jedoch nicht geändert. Aus diesem Grund widerspricht die Codierung der Daten höchstwahrscheinlich der Codierungsangabe. Diese Situation führt zur Generierung von XML-Daten, die von den Anwendungsprozessen, die auf der Basis des Codierungsnamens arbeiten, nicht syntaktisch analysiert werden können.

Im Allgemeinen ist die implizite Serialisierung zu bevorzugen, weil es effizienter ist, die Daten im XML-Format an den Client zu senden. In den folgenden Situationen ist jedoch eine explizite Serialisierung mit XMLSERIALIZE vorzuziehen:

- Bei sehr umfangreichen XML-Dokumenten

  Da bei sehr umfangreichen XML-Dokumenten keine XML-Querverweise vorhanden sind, sollten Sie XMLSERIALIZE verwenden, um die Daten in einen LOB-Typ umzuwandeln, sodass LOB-Querverweise eingesetzt werden können.
- Wenn der Client keine XML-Daten unterstützt

  Wenn Sie mit einem Client einer Vorversion arbeiten, der den Datentyp XML nicht unterstützt, und die implizite Serialisierung verwenden, wandelt der DB2-Datenbankserver die Daten in einen der folgenden Datentypen um, bevor diese an den Client gesendet werden:
  - BLOB-Datentyp (Standardeinstellung)
  - CLOB-Datentyp (wenn Sie die Registrierdatenbankvariable DB2_MAP_XML_AS_CLOB_FOR_DLC auf dem Server mit dem Befehl db2set auf die Einstellung YES setzen)

Wenn Sie möchten, dass die abgerufenen Daten einen anderen Datentyp aufweisen, können Sie XMLSERIALIZE ausführen.

Beispiel: Die XML-Spalte 'Info' in der Beispieltabelle 'Customer' enthält ein Dokument, das die hierarchische Entsprechung der folgenden Daten enthält:

```
<customerinfo Cid='1000'>
    <name>Kathy Smith</name>
  <addr country='Canada'>
      <street>5 Rosewood</street>
      <city>Toronto</city>
  <prov-state>Ontario</prov-state>
      <pcode-zip>M6W 1E6</pcode-zip>
  </addr>
  <phone type='work'>416-555-1358</phone>
</customerinfo>
```

Rufen Sie XMLSERIALIZE auf, um die Daten zu serialisieren und in den BLOB-Datentyp umzuwandeln, bevor diese in eine Hostvariable abgerufen werden.

```
SELECT XMLSERIALIZE(Info as BLOB(1M)) from Customer
  WHERE CID=1000
```

Umsetzung mit XSLT-Formatvorlagen

Die Standardmethode zum Umsetzen von XML-Daten in andere Formate ist die Verwendung von XSLT (Extensible Stylesheet Language Transformation). Sie können die integrierte Funktion XSLTRANSFORM verwenden, um XML-Dokumente in HTML, Klartext oder andere XML-Schemata umzusetzen.

XSLT verwendet Formatvorlagen (Style-Sheets), um XML-Daten in andere Datenformate umzusetzen. Sie können ein XML-Dokument entweder ganz oder teilweise umwandeln und die Daten auswählen oder neu anordnen, indem Sie die Abfragesprache XPath und die integrierten Funktionen von XSLT verwenden. XSLT wird normalerweise verwendet, um XML in HTML umzuwandeln. Es ist jedoch auch möglich, XML-Dokumente, die einem bestimmten XML-Schema entsprechen, in Dokumente umzusetzen, die einem anderen Schema entsprechen. Darüber hinaus können XML-Daten mithilfe von XSLT auch in nicht zugehörige Formate wie beispielsweise durch Kommata begrenzten Text oder Formatierungssprachen wie beispielsweise Troff umgewandelt werden. XSLT hat zwei Hauptanwendungsgebiete:

- Formatierung (Umsetzung von XML in HTML oder Formatierungssprachen wie beispielsweise FOP)
- Datenaustausch (Abfragen von Daten, Reorganisieren von Daten und Umsetzen von Daten aus einem XML-Schema in ein anderes XML-Schema oder in ein Datenaustauschformat wie beispielsweise SOAP)

In beiden Fällen kann es erforderlich sein, entweder ein ganzes XML-Dokument oder nur ausgewählte Teile desselben umzusetzen. XSLT umfasst die XPath-Spezifikation, sodass beliebige Daten aus dem XML-Quellendokument abgefragt und abgerufen werden können. Eine XSLT-Schablone kann unter Umständen auch zusätzliche Informationen enthalten oder erstellen. Hierzu gehören beispielsweise Headerdaten für Filter und Anweisungsblöcke, die der Ausgabedatei hinzugefügt werden.

Funktionsweise von XSLT

XSLT-Formatvorlagen sind in XSL (Extensible Stylesheet Language), einem XML-Schema, geschrieben. XSL ist keine algorithmische Sprache (wie beispielsweise C oder Perl), sondern vielmehr eine Schablonensprache. Daher ist die Leistung von

XSL einerseits zwar begrenzt, andererseits ist XSL aber gerade dadurch einzigartig für den beabsichtigten Zweck geeignet. XSL-Formatvorlagen enthalten mindestens ein Element vom Typ `template` (Schablone), das beschreibt, welche Aktion ausgeführt werden soll, wenn ein bestimmtes XML-Element oder eine bestimmte Abfrage in der Zieldatei gefunden wird. Ein XSLT-Schablonenelement beginnt üblicherweise mit der Angabe des Elements, für das es gilt. Beispiel:

```
<xsl:template match="product">
```

Dieses Element deklariert, dass der Inhalt dieser Schablone verwendet wird, um den Inhalt einer beliebigen Markierung vom Typ `<product>` zu ersetzen, die in der XML-Zieldatei gefunden wird. Eine XSLT-Datei besteht aus einer Liste solcher Schablonen in beliebiger Reihenfolge.

Das nachstehende Beispiel zeigt typische Elemente einer XSLT-Schablone. In diesem Fall besteht das Ziel aus XML-Dokumenten, die Inventarinformationen enthalten wie beispielsweise den folgenden Eintrag zu einem Eiskratzer (Ice Scraper):

```
<?xml version="1.0"?>
<product pid="100-201-01">
   <description>
      <name>Ice Scraper, Windshield 4 inch</name>
      <details>Basic Ice Scraper 4 inches wide, foam handle</details>
      <price>3.99</price>
   </description>
</product>
```

Dieser Eintrag enthält Informationen wie beispielsweise die Teilenummer (pid), eine Beschreibung (description) und den Preis (price) eines Eiskratzers für die Windschutzscheibe. Einige dieser Informationen sind in Elementen wie beispielsweise `<name>` enthalten. Andere Informationen wie beispielsweise die Teilenummer sind in Attributen (in diesem Fall dem Attribut `pid` des Elements `<product>`) enthalten. Um diese Informationen als Webseite anzuzeigen, könnten Sie die folgende XSLT-Schablone anwenden:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html"/>
   <xsl:template match="/">
       <html>
            <body>
                <h1><xsl:value-of select="/product/description/name"/></h1>
                <table border="1">
                        <th>
       <xsl:apply-templates select="product"/>
                        </th>
                    </table>
            </body>
       </html>
   </xsl:template>
     <xsl:template match="product">
        <tr>
                    <td width="80">product ID</td>
                    <td><xsl:value-of select="@pid"/></td>
              </tr>
              <tr>
                    <td width="200">product name</td>
                    <td><xsl:value-of select="/product/description/name"/></td>
              </tr>
              <tr>
                    <td width="200">price</td>
                    <td>$<xsl:value-of select="/product/description/price"/></td>
              </tr>
```

```
            <tr>
                 <td width="50">details</td>
                 <td><xsl:value-of select="/product/description/details"/></td>
            </tr>
  </xsl:template>
</xsl:stylesheet>
```

Wenn ein XSLT-Prozessor sowohl die oben dargestellte Schablone als auch die oben dargestellten Zieldokumente als Eingabe erhält, wird das folgende HTML-Dokument ausgegeben:

```
<html>
<body>
<h1>Ice Scraper, Windshield 4 inch</h1>
<table border="1">
<th>
<tr>
<td width="80">product ID</td><td>100-201-01</td>
</tr>
<tr>
<td width="200">product name</td><td>Ice Scraper, Windshield 4 inch</td>
</tr>
<tr>
<td width="200">price</td><td>$3.99</td>
</tr>
<tr>
<td width="50">details</td><td>Basic Ice Scraper 4 inches wide, foam handle</td>
</tr>
</th>
</table>
</body>
</html>
```

Der XSLT-Prozessor prüft das eingehende XML-Dokument auf bestimmte Bedingungen (normalerweise eine Bedingung pro Schablone). Wenn eine Bedingung 'true' (wahr) ist, wird der Schabloneninhalt in die Ausgabe eingefügt. Ist der Wert der Bedingung 'false' (falsch), wird die Schablone vom Prozessor nicht berücksichtigt. Möglicherweise fügt die Formatvorlage der Ausgabe eigene Daten hinzu wie beispielsweise Markierungen und Zeichenfolge (z. B. "product ID") in der HTML-Tabelle.

Mithilfe von XPath ist es möglich, sowohl Schablonenbedingungen zu definieren (wie beispielsweise `<xsl:template match="product">`) als auch Daten an beliebigen Positionen im XML-Datenstrom auszuwählen und dort einzufügen (wie beispielsweise `<h1><xsl:value-of select="/product/description/name"/></h1>`).

Verwendung von XSLTRANSFORM

Mithilfe der Funktion XSLTRANSFORM können Sie XSLT-Formatvorlagen auf XML-Daten anwenden. Wenn Sie für die Funktion den Namen eines XML-Dokuments und eine XSLT-Formatvorlage angeben, wendet die Funktion die Formatvorlage auf das betreffende Dokument an und gibt das entsprechende Ergebnis zurück.

Falls Sie die Dokumentfunktion (document) von XSLT in Ihrer XSLT-Formatvorlage angeben, stellen Sie sicher, dass in der Registry-Variable **DB2_XSLT_ALLOWED_PATH** die Verzeichnisse festgelegt sind, von denen Sie weitere XML-Dokumente herunterladen können.

Übergeben von Parametern an XSLT-Formatvorlagen während der Laufzeit

Mithilfe der integrierten Funktion XSLTRANSFORM zur Umwandlung von XML-Dokumenten können Parameter während der Laufzeit übergeben werden. .

Ein wichtiges Merkmal der Funktion XSLTRANSFORM ist ihre Fähigkeit, XSLT-Parameter während der Laufzeit empfangen zu können. Ohne diese Fähigkeit müssten Sie entweder eine umfangreiche Bibliothek an XSLT-Formatvorlagen verwalten (eine Formatvorlage für jede Variante einer Abfrage für die XML-Daten) oder Ihre Formatvorlagen für jede neue Abfragesorte manuell bearbeiten. Die Parameterübergabe ermöglicht es Ihnen, generische Formatvorlagen zu entwerfen, die nicht weiter bearbeitet werden müssen, während Sie eine Bibliothek an Parameterdateien aufbauen oder entsprechende Dateien möglicherweise während des Betriebs erstellen.

Die XSLT-Parameter werden in einem eigenen XML-Dokument gespeichert. Beispiel:

```
<?xml version="1.0"?>
<params xmlns="http://www.ibm.com/XSLTransformParameters">
        <param name="headline">BIG BAZAAR super market</param>
        <param name="supermarketname" value="true"/>
</params>
```

Jedes Element vom Typ `<param>` benennt einen Parameter und enthält dessen Wert entweder im Attribut `value` oder - bei längeren Werten - im Element selbst. Das vorherige Beispiel zeigt beide Varianten.

Die für die XSLT-Schablonendatei zulässigen Parameter werden mithilfe des Elements `<xsl:param>` wie folgt als Variablen definiert:

```
<xsl:param name="headline"/>
<xsl:param name="supermarketname"/>
```

In diesem Beispiel können Sie die Variablen `$headline` oder `$supermarketname` an einer beliebigen Position innerhalb der Formatvorlage aufrufen, und sie enthalten stets die in der Parameterdatei definierten Daten (in diesem Falls die Zeichenfolge "BIG BAZAAR super market" und den Wert "true" für 'wahr').

Beispiel: Verwendung von XSLT als Formatierungssteuerkomponente

Ein Beispiel, das zeigt, wie die integrierte Funktion XSLTRANSFORM als Formatierungssteuerkomponente verwendet wird.

Dieses Beispiel zeigt, wie XSLT als Formatierungssteuerkomponente verwendet wird. Zur Vorbereitung müssen Sie zunächst die beiden folgenden Beispieldokumente in die Datenbank einfügen:

```
INSERT INTO XML_TAB VALUES
(1,
    '<?xml version="1.0"?>
<students xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation = "/home/steffen/xsd/xslt.xsd">
<student studentID="1" firstName="Steffen" lastName="Siegmund"
    age="23" university="Rostock"/>
</students>',
    '<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
            xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

```
<xsl:param name="headline"/>
<xsl:param name="showUniversity"/>
<xsl:template match="students">
     <html>
              <head/>
                 <body>
                  <h1><xsl:value-of select="$headline"/></h1>
                  <table border="1">
                    <th>
                        <tr>
                                <td width="80">StudentID</td>
                                <td width="200">First Name</td>
                                <td width="200">Last Name</td>
                                <td width="50">Age</td>
                                <xsl:choose>
       <xsl:when test="$showUniversity ='true'">
                                   <td width="200">University</td>
                                    </xsl:when>
       </xsl:choose>
                             </tr>
                           </th>
                           <xsl:apply-templates/>
                  </table>
                 </body>
                </html>
         </xsl:template>
         <xsl:template match="student">
               <tr>
                   <td><xsl:value-of select="@studentID"/></td>
                   <td><xsl:value-of select="@firstName"/></td>
                   <td><xsl:value-of select="@lastName"/></td>
                   <td><xsl:value-of select="@age"/></td>
                   <xsl:choose>
                       <xsl:when test="$showUniversity = 'true' ">
                         <td><xsl:value-of select="@university"/></td>
                       </xsl:when>
                   </xsl:choose>
       </tr>
         </xsl:template>
</xsl:stylesheet>'
);
```

Rufen Sie als Nächstes die Funktion XSLTRANSFORM auf, um die XML-Daten in HTML umzusetzen und anzuzeigen:

```
SELECT XSLTRANSFORM (XML_DOC USING XSL_DOC AS CLOB(1M)) FROM XML_TAB;
```

Als Ergebnis wird folgendes Dokument angezeigt:

```
<html>
<head>
<META http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<h1></h1>
<table border="1">
<th>
<tr>
<td width="80">StudentID</td>
<td width="200">First Name</td>
<td width="200">Last Name</td>
<td width="50">Age</td>
</tr>
</th>
 <tr>
<td>1</td>
<td>Steffen</td><td>Siegmund</td>
```

```
<td>23</td>
</tr>
  </table>
</body>
</html>
```

In diesem Beispiel liegt die Ausgabe in HTML vor, und die Parameter beeinflussen lediglich, welche HTML erstellt wird und welche Daten umgewandelt werden. Somit veranschaulicht das Beispiel die Verwendung von XSLT als Formatierungssteuerkomponente für die Endbenutzerausgabe.

Beispiel: Verwendung von XSLT für den Datenaustausch

Ein Beispiel, das zeigt, wie die integrierte Funktion XSLTRANSFORM verwendet wird, um XML-Dokumente für den Datenaustausch umzuwandeln.

Dieses Beispiel zeigt, wie XSLT für den Datenaustausch eingesetzt wird, indem Parameter mit der Formatvorlage verwendet werden, um unterschiedliche Datenaustauschformate während der Laufzeit zu erstellen.

In diesem Beispiel wird eine Formatvorlage verwendet, die Elemente vom Typ `xsl:param` umfasst, um Daten aus einer Parameterdatei zu erfassen:

```
INSERT INTO Display_productdetails values(1, '<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
                  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:param name="headline"/>
<xsl:param name="supermarketname"/>
<xsl:template match="product">
   <html>
             <head/>
                <body>
                <h1><xsl:value-of select="$headline"/></h1>
                <table border="1">
                        <th>
                        <tr>
                         <td width="80">product ID</td>
                         <td width="200">product name</td>
                         <td width="200">price</td>
                         <td width="50">details</td>
                         <xsl:choose>
   <xsl:when test="$supermarket =''true'' ">
                                  <td width="200">BIG BAZAAR super market</td>
                                 </xsl:when>
    </xsl:choose>
                         </tr>
                        </th>
                        <xsl:apply-templates/>
                </table>
                </body>
                </html>
       </xsl:template>
      <xsl:template match="product">
             <tr>
                   <td><xsl:value-of select="@pid"/></td>
                   <td><xsl:value-of select="/product/description/name"/></td>
                   <td><xsl:value-of select="/product/description/price"/></td>
                   <td><xsl:value-of select="/product/description/details"/></td>
                  </tr>
     </xsl:template>
  </xsl:stylesheet>'
);
```

Die Parameterdatei enthält Parameter, die denen in der XSLT-Schablone entsprechen, samt Inhalt:

```
CREATE TABLE PARAM_TAB (DOCID INTEGER, PARAM VARCHAR (10K));

INSERT INTO PARAM_TAB VALUES
(1,
'<?xml version="1.0"?>
<params xmlns="http://www.ibm.com/XSLTransformParameters">
        <param name="supermarketname" value="true"/>
        <param name="headline">BIG BAZAAR super market</param>
</params>'
);
```

Sie können die Parameterdatei während der Laufzeit anwenden, indem Sie den folgenden Befehl verwenden:

```
SELECT XSLTRANSFORM (XML_DOC USING XSL_DOC WITH PARAM AS CLOB (1M))
FROM product_details X, PARM_TAB P WHERE X.DOCID=P.DOCID;
```

Das Ergebnis liegt in HTML vor, wobei der Inhalt jedoch von der Parameterdatei bestimmt wird und Tests aufgrundlage des Inhalts im XML-Dokument durchgeführt werden:

```
<html>
<head>
<META http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<h1></h1>
<table border="1">
<th>
<tr>
<td width="80">product ID</td>
<td width="200">product Name</td>
<td width="200">price</td>
<td width="50">Details</td>
</tr>
</th>
</table>
</body>
</html>
```

In anderen Anwendungen liegt die Ausgabe von XSLTRANSFORM möglicherweise nicht in HTML vor, sondern in Form eines anderen XML-Dokuments oder einer Datei mit einem anderen Datenformat (beispielsweise in Form einer EDI-Datei).

Bei Anwendungen für Datenaustausch kann die Parameterdatei EDI- oder SOAP-Dateiheaderinformationen wie beispielsweise E-Mail- oder Portadressen bzw. andere kritische Daten, die nur für eine bestimmte Transaktion gelten, enthalten. Da es sich bei den in den vorherigen Beispielen verwendeten XML-Daten um einen Lagerbestandseintrag handelt, kann man sich leicht vorstellen, diesen Eintrag mithilfe von XSLT entsprechend umzuwandeln, um den Datenaustausch mit dem Einkaufsystem eines Kunden zu ermöglichen.

Beispiel: Entfernen von Namensbereichen mit XSLT

In den XML-Dokumenten, die Sie erhalten, sind möglicherweise unnötige oder falsche Informationen zu Namensbereichen enthalten. Mithilfe von XSLT-Formatvorlagen können Sie die Namensbereichsinformationen in den Dokumenten entfernen oder bearbeiten.

In den folgenden Beispielen wird gezeigt, wie Sie Namensbereichsinformationen mithilfe von XSLT aus einem XML-Dokument entfernen. In den Beispielen werden das XML-Dokument und die XSLT-Formatvorlagen in XML-Spalten gespeichert und das XML-Dokument wird mit der Funktion XSLTRANSFORM mithilfe einer der XSLT-Formatvorlagen umgesetzt.

Mit den folgenden CREATE-Anweisungen werden die Tabellen XMLDATA und XMLTRANS erstellt. XMLDATA enthält ein XML-Beispieldokument, XMLTRANS enthält XSLT-Formatvorlagen.

```
CREATE TABLE XMLDATA (ID BIGINT NOT NULL PRIMARY KEY, XMLDOC XML );
CREATE TABLE XMLTRANS (XSLID BIGINT NOT NULL PRIMARY KEY, XSLT XML );
```

Fügen Sie das XML-Beispieldokument mit der folgenden Anweisung INSERT zur Tabelle XMLDATA hinzu.

```
insert into XMLDATA (ID, XMLDOC) values ( 1, '
<newinfo xmlns="http://mycompany.com">
<!-- merged customer information -->
  <customerinfo xmlns="http://oldcompany.com" xmlns:d="http://test" Cid="1004">
     <name>Matt Foreman</name>
  <addr country="Canada">
        <street>1596 Baseline</street>
        <city>Toronto</city>
     <prov-state>Ontario</prov-state>
     <pcode-zip>M3Z 5H9</pcode-zip>
                                      </addr>
  <phone type="work">905-555-4789</phone>
  <h:phone xmlns:h="http://test1" type="home">416-555-3376</h:phone>
  <d:assistant>
      <name>Gopher Runner</name>
      <h:phone xmlns:h="http://test1" type="home">416-555-3426</h:phone>
  </d:assistant>
  </customerinfo>
</newinfo>
');
```

Beispiel einer XSLT-Formatvorlage, mit der alle Namensbereiche entfernt werden

Im folgenden Beispiel werden mithilfe einer XSLT-Formatvorlage alle Namensbereichsinformationen aus dem in der Tabelle XMLDATA gespeicherten XML-Dokument entfernt. In dem Beispiel wird die Formatvorlage in der Tabelle XMLTRANS gespeichert und mit einer Anweisung SELECT auf das XML-Dokument angewendet.

Fügen Sie die Formatvorlage mit der Anweisung INSERT zur Tabelle XMLTRANS hinzu.

```
insert into XMLTRANS (XSLID, XSLT) values ( 1, '
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

 <!-- keep comments -->
 <xsl:template match="comment()">
    <xsl:copy>
      <xsl:apply-templates/>
    </xsl:copy>
  </xsl:template>

  <xsl:template match="*">
    <!-- remove element prefix -->
    <xsl:element name="{local-name()}">
      <!-- process attributes -->
```

```
      <xsl:for-each select="@*">
        <!-- remove attribute prefix -->
        <xsl:attribute name="{local-name()}">
          <xsl:value-of select="."/>
        </xsl:attribute>
      </xsl:for-each>
      <xsl:apply-templates/>
    </xsl:element>
  </xsl:template>
</xsl:stylesheet>
') ;
```

Die folgende Anweisung SELECT konvertiert das XML-Beispieldokument mithilfe der XSLT-Formatvorlage.

```
SELECT XSLTRANSFORM (XMLDOC USING XSLT ) FROM XMLDATA, XMLTRANS
   WHERE ID = 1 and XSLID = 1
```

Der Befehl XSLTRANSFORM konvertiert das XML-Dokument mit der ersten XSLT-Formatvorlage und gibt den folgenden XML-Code zurück, aus dem alle Namensbereichsinformationen entfernt wurden.

```
<?xml version="1.0" encoding="UTF-8"?>
<newinfo>
 <!-- merged customer information -->
<customerinfo Cid="1004">
<name>Matt Foreman</name>
  <addr country="Canada">
    <street>1596 Baseline</street>
    <city>Toronto</city>
   <prov-state>Ontario</prov-state>
<pcode-zip>M3Z 5H9</pcode-zip>
  </addr>
<phone type="work">905-555-4789</phone>
<phone type="home">416-555-3376</phone>
  <assistant>
<name>Gopher Runner</name>
<phone type="home">416-555-3426</phone>
  </assistant>
 </customerinfo>
</newinfo>
```

Beispiel einer XSLT-Formatvorlage, mit der die Namensbereichsbindung für ein Element beibehalten wird

Im folgenden Beispiel wird mithilfe einer XSLT-Formatvorlage nur die Namensbereichsbindung für die Elementknoten 'phone' beibehalten. Der Name des Knotens wird in der XSLT-Variablen `mynode` angegeben. In dem Beispiel wird die Formatvorlage in der Tabelle XMLTRANS gespeichert und mit einer Anweisung SELECT auf das XML-Dokument angewendet.

Fügen Sie die Formatvorlage mit der folgenden Anweisung INSERT zur Tabelle XMLTRANS hinzu.

```
insert into XMLTRANS (XSLID, XSLT) values ( 2, '
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
  <xsl:variable name ="mynode">phone</xsl:variable>

  <!-- keep comments -->
  <xsl:template match="comment()">
      <xsl:copy>
        <xsl:apply-templates/>
      </xsl:copy>
   </xsl:template>
```

```
  <xsl:template xmlns:d="http://test" xmlns:h="http://test1" match="*">
  <xsl:choose>

  <!-- keep namespace prefix for node names $mynode -->
    <xsl:when test="local-name() = $mynode " >
     <xsl:element name="{name()}">
       <!-- process node attributes -->
       <xsl:for-each select="@*">
       <!-- remove attribute prefix  -->
         <xsl:attribute name="{local-name()}">
           <xsl:value-of select="."/>
         </xsl:attribute>
       </xsl:for-each>
      <xsl:apply-templates/>
     </xsl:element>
    </xsl:when>

  <!-- remove namespace prefix from node -->
    <xsl:otherwise>
     <xsl:element name="{local-name()}">
       <!-- process node attributes -->
       <xsl:for-each select="@*">
       <!-- remove attribute prefix  -->
         <xsl:attribute name="{local-name()}">
           <xsl:value-of select="."/>
         </xsl:attribute>
       </xsl:for-each>
      <xsl:apply-templates/>
     </xsl:element>
     </xsl:otherwise>
  </xsl:choose>
  </xsl:template>
</xsl:stylesheet>
');
```

Die folgende Anweisung SELECT konvertiert das XML-Beispieldokument mithilfe der zweiten XSLT-Formatvorlage.

```
SELECT XSLTRANSFORM (XMLDOC USING XSLT) FROM XMLDATA, XMLTRANS
   WHERE ID = 1 and XSLID = 2 ;
```

Der Befehl XSLTRANSFORM konvertiert das XML-Dokument mit der zweiten XSLT-Formatvorlage und gibt den folgenden XML-Code zurück, aus dem nur die Namensbereiche für die Elemente 'phone' entfernt wurden.

```
<?xml version="1.0" encoding="UTF-8"?>
<newinfo>
 <!-- merged customer information -->
<customerinfo Cid="1004">
<name>Matt Foreman</name>
  <addr country="Canada">
    <street>1596 Baseline</street>
    <city>Toronto</city>
   <prov-state>Ontario</prov-state>
<pcode-zip>M3Z 5H9</pcode-zip>
  </addr>
<phone type="work">905-555-4789</phone>
  <h:phone xmlns:h="http://test1" type="home">
   416-555-3376
  </h:phone>
  <assistant>
<name>Gopher Runner</name>
   <h:phone xmlns:h="http://test1" type="home">
    416-555-3426
```

```
   </h:phone>
  </assistant>
 </customerinfo>
</newinfo>
```

Beispiel: Verwendung der Dokumentfunktion von XSLT

Diese Beispiele veranschaulichen alle Schritte, die für die Verwendung der Dokumentfunktion von XSLT erforderlich sind.

Die Verwendung der integrierten Funktionen von XSLT wird in pureXML nicht empfohlen. Wenn Sie dennoch die Dokumentfunktion von XSLT verwenden möchten, müssen Sie die Registry-Variable **DB2_XSLT_ALLOWED_PATH** so definieren, dass Sie die Verzeichnisse auflistet, die Ihre XML-Dokumente enthalten, um die Verweise auf die Liste der URIs einzugrenzen. Die Dokumentfunktion von XSLT kann standardmäßig nicht auf XML-Dokumente zugreifen.

Die folgenden Beispiele veranschaulichen, wie Sie die Dokumentfunktion von XSLT auf sichere Weise verwenden können.

In den Beispielen werden das XML-Dokument und die XSLT-Formatvorlagen in XML-Spalten gespeichert und das XML-Dokument wird mit der Funktion XSLTRANSFORM mithilfe einer der XSLT-Formatvorlagen umgesetzt. Die Beispiele verwenden die Tabellen XMLDATA und XMLTRANS, die für „Beispiel: Entfernen von Namensbereichen mit XSLT” auf Seite 146 erstellt wurden.

Beispiel für Linux- und UNIX-Umgebungen

In diesem Beispiel befinden sich die XML-Dateien im Verzeichnis `/home/user/xml_files`. Legen Sie die Registry-Variable **DB2_XSLT_ALLOWED_PATH** auf dieses Verzeichnis fest.

```
db2set DB2_XSLT_ALLOWED_PATH="/home/user/xml_files"
```

Starten Sie die Instanz erneut, damit die Einstellung dieser Registry-Variablen wirksam wird.

Fügen Sie die Formatvorlage mit der Anweisung INSERT zur Tabelle XMLTRANS hinzu.

```
INSERT INTO XMLTRANS (XSLID, XSLT)
 VALUES ( 3, '<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
               <xsl:template match="/">
                <xsl:value-of select="document(''/home/user/xml_files/n.xml'')/*"/>
               </xsl:template>
              </xsl:stylesheet>') ;
```

Die folgende Anweisung SELECT konvertiert das XML-Beispieldokument mithilfe der XSLT-Formatvorlage mit XSLID = 3.

```
SELECT XSLTRANSFORM (XMLDOC USING XSLT ) FROM XMLDATA, XMLTRANS
   WHERE ID = 1 and XSLID = 3
```

Der Befehl XSLTRANSFORM konvertiert das XML-Dokument mithilfe der XSLT-Formatvorlage und gibt das konvertierte XML-Dokument zurück. Falls der Datenbankmanager das in der Dokumentfunktion von XSLT angegebene Dokument nicht öffnen kann, wird der Aufruf der Dokumentfunktion ignoriert.

Beispiel für Windows-Umgebungen

In diesem Beispiel befinden sich die XML-Dateien im Verzeichnis `C:\Documents and Settings\user\xml_files`. Legen Sie die Registry-Variable **DB2_XSLT_ALLOWED_PATH** auf dieses Verzeichnis fest. Dazu gibt es folgende Möglichkeiten:

- `db2set DB2_XSLT_ALLOWED_PATH="C:\Documents%20and%20Settings\user\xml_files"`
- `db2set DB2_XSLT_ALLOWED_PATH="file:///C:/Documents%20and%20Settings/user/xml_files"`

Verwenden Sie '%20', um ein Leerzeichen darzustellen.

Starten Sie die Instanz erneut, damit die Einstellung dieser Registry-Variablen wirksam wird.

Fügen Sie die Formatvorlage mit der Anweisung INSERT zur Tabelle XMLTRANS hinzu.

```
INSERT INTO XMLTRANS (XSLID, XSLT)
 VALUES ( 4, '<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
            <xsl:template match="/">
              <xsl:value-of select="document(''C:\Documents and Settings\user\xml_files\t.xml'')/*"/>
            </xsl:template>
          </xsl:stylesheet>') ;
```

Die folgende Anweisung SELECT konvertiert das XML-Beispieldokument mithilfe der XSLT-Formatvorlage mit XSLID = 4.

```
SELECT XSLTRANSFORM (XMLDOC USING XSLT ) FROM XMLDATA, XMLTRANS
   WHERE ID = 1 and XSLID = 4
```

Der Befehl XSLTRANSFORM konvertiert das XML-Dokument mithilfe der XSLT-Formatvorlage und gibt das konvertierte XML-Dokument zurück. Falls der Datenbankmanager das in der Dokumentfunktion von XSLT angegebene Dokument nicht öffnen kann, wird der Aufruf der Dokumentfunktion ignoriert.

Wichtige Hinweise zur Umsetzung von XML-Dokumenten

Bei der Verwendung der integrierten Funktion XSLTRANSFORM zum Umwandeln von XML-Dokumenten sind einige wichtige Hinweise und Einschränkungen zu beachten.

Bitte beachten Sie bei der Umsetzung von XML-Dokumenten Folgendes:

- XML-Quellendokumente dürfen nur eine Root haben und müssen korrekt formatiert sein.
- Da bei der XSLT-Umsetzung standardmäßig UTF-8-Zeichen erstellt werden, können im Ausgabedatenstrom möglicherweise Zeichen verloren gehen, wenn der Datenstrom in Spalten eingefügt wird, die mit einem Zeichendatentyp definiert sind.

Einschränkungen

- Es wird ausschließlich 'W3C XSLT Version 1.0 Recommendation' unterstützt.
- Alle Parameter und der Ergebnistyp müssen SQL-Typen sein. Dateinamen sind nicht zulässig.
- Umsetzungen mit mehr als einem Formatvorlagedokument (bei Verwendung der Deklaration xsl:include oder xsl:import) werden nicht unterstützt.

Hinweis

Für die Umsetzung von XML-Dokumenten stehen mehrere Methoden zur Verfügung, z. B. die Funktion XSLTRANSFORM, ein XQuery-Aktualisierungsausdruck sowie die XSLT-Verarbeitung durch einen externen Anwendungsserver. Für Dokumente, die in einer DB2-XML-Spalte gespeichert sind, können viele Umsetzungen effizienter mithilfe eines XQuery-Aktualisierungsausdrucks durchgeführt werden als mit XSLT, da bei XSLT stets eine Syntaxanalyse der umzusetzenden XML-Dokumente erforderlich ist.

Wenn Sie für die Umsetzung von XML-Dokumenten XSLT verwenden möchten, sollten Sie sorgfältig abwägen, ob das Dokument in der Datenbank oder in einem Anwendungsserver umgesetzt werden soll.

Falls Sie die Dokumentfunktion (document) von XSLT in Ihrer XSLT-Formatvorlage angeben, stellen Sie sicher, dass in der Registry-Variable **DB2_XSLT_ALLOWED_PATH** die Verzeichnisse festgelegt sind, von denen Sie weitere XML-Dokumente herunterladen können.

Abweichungen in einem XML-Dokument nach dem Speichern und Abrufen

Nach dem Speichern eines XML-Dokuments in einer DB2-Datenbank ist das abgerufene Dokument möglicherweise nicht mehr identisch mit dem Originaldokument. Dieses Verhalten wird durch den XML- und SQL/XML-Standard definiert und entspricht dem Verhalten des Open-Source-XML-Parsers 'Xerces'.

Einige der Änderungen am Dokument erfolgen, wenn das Dokument gespeichert wird. Diese Änderung sind folgende:

- Wenn Sie XMLVALIDATE ausführen, nimmt der Datenbankserver folgende Änderungen vor:
  - Er fügt dem Eingabedokument Standardwerte aus dem XML-Schema hinzu, das im XMLVALIDATE-Aufruf angegeben wird.
  - Er entfernt ignorierbare Leerzeichen aus dem Eingabedokument.
- Wenn Sie keine XML-Prüfung anfordern, führt der Datenbankserver folgende Aktionen aus:
  - Er entfernt Boundary-Leerzeichen, wenn Sie nicht anfordern, dass diese beibehalten werden sollen.
  - Er führt die in der Spezifikation XML 1.0 festgelegte Zeilenendenormalisierung aus.
  - Er führt die in der Spezifikation XML 1.0 festgelegte Attributwertnormalisierung aus.

    Dieser Prozess bewirkt, dass Zeilenvorschubzeichen (U+000A) in Attributen durch Leerzeichen (U+0020) ersetzt werden.

Weitere Änderungen erfolgen, wenn Sie die Daten aus einer XML-Spalte abrufen. Diese Änderung sind folgende:

- Wenn die Daten eine XML-Deklaration haben, bevor sie an den Datenbankserver gesendet werden, wird die XML-Deklaration nicht beibehalten.

  Bei impliziter Serialisierung fügt der DB2-Datenbankserver für CLI-Anwendungen und Anwendungen mit eingebettetem SQL den Daten eine XML-Deklaration mit der entsprechenden Codierungsspezifikation hinzu. Bei Java- und .NET-Anwendungen fügt der DB2-Datenbankserver keine XML-Deklaration hinzu. Wenn

die Daten jedoch für ein DB2Xml-Objekt abgerufen und bestimmte Methoden zum Abrufen dieser Daten aus diesem Objekt verwendet werden, fügt IBM Data Server Driver for JDBC and SQLJ eine XML-Deklaration hinzu.

Wenn Sie die Funktion XMLSERIALIZE mit Angabe der Option INCLUDING XMLDECLARATION ausführen, fügt der DB2-Datenbankserver eine XML-Deklaration mit einer Codierungsspezifikation für UTF-8-Codierung hinzu.

- Innerhalb des Inhalts eines Dokuments und in Attributwerten werden bestimmte Zeichen durch die entsprechenden vordefinierten XML-Darstellungselemente ersetzt. Diese Zeichen und die zugehörigen vordefinierten Darstellungselemente sind folgende:

Zeichen	Unicode-Wert	Darstellendes Entitätselement
ET-ZEICHEN	U+0026	&
KLEINER-ALS-ZEICHEN	U+003C	<
GRÖSSER-ALS-ZEICHEN	U+003E	>

- In Attribut- oder Textwerten werden bestimmte Zeichen durch ihre numerische Darstellung ersetzt. Diese Zeichen und ihre numerische Darstellung sind folgende:

Zeichen	Unicode-Wert	Darstellendes Entitätselement
TABULATORZEICHEN	U+0009	
ZEILENVORSCHUB	U+000A
WAGENRÜCKLAUF	U+000D
NÄCHSTE ZEILE	U+0085	…
ZEILENTRENNZEICHEN	U+2028	

- In Attributwerten wird das Anführungszeichen (U+0022) durch das entsprechende vordefinierte XML-Darstellungselement " ersetzt.
- Wenn das Eingabedokument eine DTD-Deklaration besitzt, wird die Deklaration nicht beibehalten und keine auf der DTD basierende Markup-Formatierung generiert.
- Wenn das Eingabedokument CDATA-Abschnitte enthält, bleiben diese Abschnitte in der Ausgabe nicht erhalten.

Kapitel 6. XML-Daten indexieren

Ein Index für XML-Daten kann die Effizienz von Abfragen für XML-Dokumente verbessern, die in einer XML-Spalte gespeichert sind.

Im Unterschied zu traditionellen relationalen Indizes, bei denen Indexschlüssel aus mindestens einer von Ihnen angegebenen Tabellenspalte bestehen, verwendet ein Index für XML-Daten einen bestimmten XML-Musterausdruck zum Indexieren von Pfaden und Werten in XML-Dokumenten, die in nur einer XML-Spalte gespeichert sind. Der Datentyp dieser Spalte muss XML sein.

Anstelle des Zugriffs auf den Anfang eines Dokuments ermöglichen die Indexeinträge in einem Index für XML-Daten den Zugriff auf Knoten innerhalb des Dokuments, indem Indexschlüssel auf der Basis von XML-Musterausdrücken erstellt werden. Da ein XML-Muster von verschiedenen Teilen eines XML-Dokuments erfüllt werden kann, können für ein einzelnes Dokument mehrere Indexschlüssel in den Index eingefügt werden.

Ein Index für XML-Daten wird mithilfe der Anweisung CREATE INDEX erstellt und mithilfe der Anweisung DROP INDEX gelöscht. Die Klausel GENERATE KEY USING XMLPATTERN, die Sie in die Anweisung CREATE INDEX einschließen, gibt die zu indexierenden Daten an.

Einige der Schlüsselwörter, die in der Anweisung CREATE INDEX für Indizes von Nicht-XML-Spalten verwendet werden, sind auf Indizes zu XML-Daten nicht anwendbar. Auch das Schlüsselwort UNIQUE hat bei Indizes zu XML-Daten eine andere Bedeutung.

Beispiel: Erstellen eines Index für XML-Daten

Angenommen, die Tabelle 'companyinfo' enthält eine XML-Spalte mit dem Namen 'companydocs', in der XML-Dokumentfragmente wie die folgenden enthalten sind:

Dokument für 'Company1'

```
<company name="Company1">
  <emp id="31201" salary="60000" gender="Female">
    <name>
      <first>Laura</first>
      <last>Brown</last>
    </name>
    <dept id="M25">
      Finance
    </dept>
  </emp>
</company>
```

Dokument für 'Company2'

```
<company name="Company2">
  <emp id="31664" salary="60000" gender="Male">
    <name>
      <first>Chris</first>
      <last>Murphy</last>
    </name>
    <dept id="M55">
      Marketing
    </dept>
  </emp>
  <emp id="42366" salary="50000" gender="Female">
```

```
    <name>
      <first>Nicole</first>
      <last>Murphy</last>
    </name>
    <dept id="K55">
      Sales
    </dept>
  </emp>
</company>
```

Benutzer der Tabelle 'companyinfo' rufen häufig Mitarbeiterinformationen mithilfe der Mitarbeiter-ID ('emp id') ab. In diesem Fall können Sie einen Index wie den folgenden verwenden, um die Effizienz solcher Abrufe zu erhöhen:

```
CREATE INDEX empindex on companyinfo(companydocs)          1
  GENERATE KEY USING XMLPATTERN '/company/emp/@id'         2
   AS SQL DOUBLE                                     3
```

Abbildung 5. Beispiel eines Index für XML-Daten

Anmerkungen zu Abb. 5:

1 Der Index für XML-Daten wird für die Spalte 'companydocs' der Tabelle 'companyinfo' definiert. Die Spalte 'companydocs' muss den Datentyp XML besitzen.

2 Die Klausel GENERATE KEY USING XMLPATTERN gibt Informationen zu den zu indexierenden Daten an. Diese Klausel wird als XML-Indexspezifikation bezeichnet. Die XML-Indexspezifikation enthält eine XML-Musterklausel. Die XML-Musterklausel in diesem Beispiel gibt an, dass die Werte des Attributs 'id' für jedes Mitarbeiterelement ('emp') zu indexieren sind.

3 AS SQL DOUBLE gibt an, dass die indexierten Werte als DOUBLE-Werte gespeichert werden.

XML-Musterausdrücke für Indizes

Es werden nur die Teile eines in einer XML-Spalte gespeicherten XML-Dokuments indexiert, die einem XML-Schemaausdruck entsprechen. Zur Erstellung eines Index, der auf einem XML-Schema basiert, geben Sie eine Anweisung CREATE INDEX mit einer Indexspezifikationsklausel an.

Die Indexspezifikationsklausel beginnt mit der Angabe GENERATE KEY USING XMLPATTERN, an die sich ein XML-Muster und ein Datentyp für den Index für XML-Daten anschließen. Alternativ können Sie auch die Klausel GENERATE KEYS USING XMLPATTERN angeben.

Für jede Anweisung CREATE INDEX ist nur eine Indexspezifikationsklausel zulässig. Für eine XML-Spalte können mehrere XML-Indizes erstellt werden.

XML-Musterausdrücke

Zur Festlegung der zu indexierenden Teile des Dokuments wird mithilfe eines XML-Musters eine Reihe von Knoten innerhalb des XML-Dokuments angegeben. Dieser Musterausdruck ist dem Pfadausdruck ähnlich, der in der XQuery-Sprache definiert ist. Jedoch unterscheidet er sich dadurch, dass für ihn nur eine Untermenge der XQuery-Sprache unterstützt wird.

Die Schritte eines Pfadausdrucks werden durch einen Schrägstrich (/) getrennt. Der doppelte Schrägstrich (//), der eine abgekürzte Syntax für '/descendant-or-

self::node()/' darstellt, kann ebenfalls angegeben werden. In jedem Schritt wird eine Vorwärtsachse (child::, @, attribute::, descendant::, self:: und descendant-or-self::) ausgewählt, gefolgt von einem XML-Namenstest bzw. einem XML-Sortentest. Wenn keine Vorwärtsachse angegeben wird, wird standardmäßig die child-Achse verwendet.

Wenn der XML-Namenstest verwendet wird, wird ein qualifizierter XML-Name oder ein Platzhalter verwendet, um den Knotennamen anzugeben, der für den Schritt im Pfad abzugleichen ist. Anstelle von abzugleichenden Knotennamen kann auch ein XML-Sortentest verwendet werden, um die Art von Knoten anzugeben, die mit dem Muster abzugleichen sind: Textknoten, Kommentarknoten, Verarbeitungsanweisungsknoten oder ein beliebiger anderer Typ von Knoten.

Musterausdrücke können Aufrufe an unterstützte Funktionen beinhalten, um Indizes mit besonderen Eigenschaften zu erstellen, beispielsweise die Nichtbeachtung der Groß- und Kleinschreibung. Für jede Klausel vom Typ XMLPATTERN ist nur ein einziger Funktionsschritt zulässig.

Die folgenden Beispiele zeigen logisch äquivalente Anweisungen mit unterschiedlichen Musterausdrücken.

Die Anweisungen 1 und 2 sind logisch äquivalent. In Anweisung 1 wird die abgekürzte Syntax verwendet.

```
CREATE INDEX empindex on company(companydocs)
  GENERATE KEY USING XMLPATTERN '/company/emp/@id' AS SQL DOUBLE
```

Abbildung 6. Anweisung 1

```
CREATE INDEX empindex on company(companydocs)
           GENERATE KEY USING XMLPATTERN '/child::company/child::emp/attribute::id'
      AS SQL DOUBLE
```

Abbildung 7. Anweisung 2

Die Anweisungen 3 und 4 sind logisch äquivalent. In Anweisung 3 wird die abgekürzte Syntax verwendet.

```
CREATE INDEX idindex on company(companydocs)
          GENERATE KEY USING XMLPATTERN '//@id' AS SQL DOUBLE
```

Abbildung 8. Anweisung 3

```
CREATE INDEX idindex on company(companydocs)
           GENERATE KEY USING XMLPATTERN '/descendant-or-self::node()/attribute::id'
      AS SQL DOUBLE
```

Abbildung 9. Anweisung 4

Anweisung 5 verwendet den XML-Sortentest zum Abgleich von Texttypknoten im angegebenen Muster:

Abbildung 10. Anweisung 5

```
CREATE INDEX empindex on company(companydocs)
           GENERATE KEY USING XMLPATTERN '/company/emp/name/last/text()' AS SQL
           VARCHAR(25)
```

Die Anweisungen 6 und 7 erstellen einen Index, bei dem die Groß-/Kleinschreibung beachtet werden muss. Die Anweisung 7 gibt an, in welche Län-

dereinstellung die im Index gespeicherten Werte konvertiert werden sollen.

```
CREATE INDEX empindex on company(companydocs)
       GENERATE KEY USING XMLPATTERN '/company/emp/name/last/fn:upper-case(.)'
       AS SQL VARCHAR(25)
```

Abbildung 11. Anweisung 6

Abbildung 12. Anweisung 7

```
CREATE INDEX empindex on company(companydocs)
        GENERATE KEY USING XMLPATTERN '/company/emp/name/last/fn:upper-case(.,
         "en_US")' AS SQL VARCHAR(25)
```

Anweisung 8 erstellt einen Index, der das Vorhandensein eines zweiten Vornamens eines Mitarbeiters in der XML-Dokumentstruktur angibt.

Abbildung 13. Anweisung 8

```
CREATE INDEX empindex on company(companydocs)
       GENERATE KEY USING XMLPATTERN
       '/company/emp/name/fn:exists(middle)' AS SQL VARCHAR(1)
```

Anweisung 9 erstellt einen Index vom Typ VARCHAR.

```
CREATE INDEX varcharidx on company(companydocs)
       GENERATE KEY USING XMLPATTERN '/company/emp/name/last'
          AS SQL VARCHAR(30)
```

Abbildung 14. Anweisung 9

Ab DB2 V10.1 kann das Optimierungsprogramm die Verwendung von Indizes des Typs VARCHAR für Abfragen mit Vergleichselementen auswählen, die die Funktion 'fn:starts-with' enthalten. Die Funktion 'fn:starts-with' ermittelt, ob eine Zeichenfolge mit einer bestimmten Unterzeichenfolge beginnt. An vorhandenen Indizes des Typs VARCHAR sind keine Änderungen erforderlich und es muss keine besondere Syntax in der Anweisung CREATE INDEX für neue Indizes verwendet werden.

Den Ausdrücken entsprechende Pfade und Knoten

Betrachten Sie eine Tabelle mit dem Namen "company", die in der XML-Spalte 'companydocs' gespeicherte XML-Dokumente enthält. Die XML-Dokumente besitzen eine Hierarchie mit den beiden Pfaden `'/company/emp/dept/@id'` und `'/company/emp/@id'`. Wenn das XML-Muster einen einzelnen Pfad angibt, kann diesem Pfad eine Gruppe von Knoten im Dokument entsprechen.

Wenn Sie zum Beispiel nach einem bestimmten ID-Attribut für Mitarbeiter (@id) in den Mitarbeiterelementen ('emp') suchen möchten, könnte ein Index mit dem XML-Muster `'/company/emp/@id'` erstellt werden. Anschließend könnten Abfragen mit Vergleichselementen der Form `'/company/emp[@id=42366]'` den Index für eine XML-Spalte nutzen. In diesem Fall gibt der Musterausdruck XMLPATTERN `'/company/emp/@id'` in der Anweisung CREATE INDEX einen einzelnen Pfad an, der sich auf zahlreiche verschiedene Knoten im Dokument bezieht, da jedes Mitarbeiterelement im Dokument ein Mitarbeiter-ID-Attribut haben kann.

```
CREATE INDEX empindex on company(companydocs)
    GENERATE KEY USING XMLPATTERN '/company/emp/@id' AS SQL DOUBLE
```

Wenn das XML-Muster Ausdrücke mit Platzhaltern, die 'descendant'-Achse oder die 'descendant-or-self'-Achse verwendet, kann diesen Ausdrücken eine Gruppe von Pfaden und Knoten entsprechen. Im folgenden Beispiel wird die 'descendant-or-self'-Achse angegeben, sodass sich das XML-Muster `'//@id'` sowohl auf das Attribut 'id' von Abteilungen als auch auf das Attribut 'id' von Mitarbeitern bezieht, da beide '@id' enthalten.

```
CREATE INDEX idindex on company(companydocs)
                  GENERATE KEYS USING XMLPATTERN '//@id' AS SQL DOUBLE
```

Wenn Sie in den Elementen vom Typ 'name' nach dem Nachnamen eines bestimmten Mitarbeiters (<last>) suchen möchten und dabei die Groß-/Kleinschreibung nicht beachtet werden soll, können Sie beispielsweise einen Index mit dem XML-Muster `'/company/emp/name/last/fn:upper-case(.)' erstellen.` . Anschließend könnten Abfragen mit Vergleichselementen der Form `'/company/emp/name/last[fn:upper-case(.)="SMITH"]"` den Index für eine XML-Spalte nutzen. In diesem Fall gibt der Kontextschritt des XML-Musters einen einzigen Pfad an, nämlich `'/company/emp/name/last'`. Für diesen Pfad kommen viele verschiedene Knoten in Frage, da jedes Element vom Typ 'name' im Dokument ein Element <last> haben kann. Die Nachnamen aller Mitarbeiter werden in Großbuchstaben indiziert.

```
CREATE INDEX empindex on company(companydocs)
    GENERATE KEY USING XMLPATTERN '/company/emp/name/last/fn:upper-case(.)'
    AS SQL VARCHAR(25)
```

Wenn das XML-Muster die Funktion 'fn:exists' enthält, wird der Indexwert als einzelnes Zeichen (T oder F) gespeichert, um anzugeben, ob es wahr (true) oder falsch (false) ist, dass sich das zu indizierende Element in der XML-Dokumentstruktur befindet. Im folgenden Beispiel werden die zweiten Vornamen aller Mitarbeiter boolesch indiziert.

```
CREATE INDEX midnameidx on company(companydocs)
         GENERATE KEY USING XMLPATTERN '/company/emp/name/fn:exists(middle)'
         AS SQL VARCHAR(1)
```

Beispiele für die Verwendung von XML-Indizes, bei denen die Groß-/Kleinschreibung nicht beachtet werden muss

Zur Beschleunigung von Abfragen, die nach Zeichenfolgedaten suchen, können Sie die Funktion 'fn:upper-case()' verwenden, um Indizes zu erstellen, die Werte als Einträge speichern, bei denen die Groß-/Kleinschreibung nicht beachtet werden muss.

Erstellen von Indizes, bei denen die Groß-/Kleinschreibung nicht beachtet werden muss

In diesem Beispiel wird veranschaulicht, wie eine Tabelle mit einer XML-Spalte erstellt wird, wie Daten dort eingefügt werden und wie Indizes erstellt werden, bei denen die Groß-/Kleinschreibung nicht beachtet werden muss.

Erstellen Sie zunächst die Tabelle CLIENTS mit der Spalte CONTACTINFO vom Typ XML:

```
CREATE TABLE clients (
  ID            INT PRIMARY KEY NOT NULL,
  NAME          VARCHAR(50),
  STATUS        VARCHAR(10),
  CONTACTINFO   XML
);
```

Fügen Sie die die Tabelle CLIENTS zwei Datensätze ein:

```
INSERT INTO clients VALUES('0092', 'Johny Peterson', 'Standard',
'<Client>
    <address type="permanent">
         <street>8734 Zuze Ave.</street>
         <city>New York</city>
         <state>New York</state>
         <zip>95443</zip>
     </address>
</Client>');

INSERT INTO clients VALUES('0093', 'Rose Locke', 'Golden',
'<Client>
    <address type="PERMANENT">
         <street>1121 Oxford Street</street>
         <city>Albany</city>
         <state>new york</state>
         <zip>19232</zip>
     </address>
</Client>');
```

Sie können im Pfad `/Client/address/state` einen Index erstellen, bei dem die Groß-/Kleinschreibung nicht beachtet werden muss, z. B. mit dem Namen 'clients_state_idx'. Der erste Parameter der Funktion 'fn:upper-case()' muss immer ein Kontextelementausdruck (.) sein.

```
CREATE INDEX clients_state_idx ON clients(contactinfo)
   GENERATE KEYS USING XMLPATTERN '/Client/address/state/fn:upper-case(.)'
   AS SQL VARCHAR(50);
```

Sie können auch Indizes für Attribute erstellen, bei denen die Groß-/Kleinschreibung nicht beachtet werden muss. Sie können beispielsweise den Index mit dem Namen 'client_address_type_idx' für das Adressattribut erstellen; geben Sie den Pfad `/Client/address/@type` ein.

```
CREATE INDEX client_address_type_idx ON clients(contactinfo)
   GENERATE KEYS USING XMLPATTERN '/Client/address/@type/fn:upper-case(.)'
   AS SQL VARCHAR(50);
```

Sowohl für den Index 'clients_state_idx' als auch für den Index 'client_address_type_idx' werden die Indexschlüsselwerte in Großschreibung und im Verschlüsselungssatz für amerikanisches Englisch gespeichert. Beispiel: Für den ersten Datensatz, der bereits weiter oben eingefügt wurde, ist der Wert, der dem Pfad `/Client/address/state` zugeordnet ist, New York; dieser wird jedoch in Großschreibung als NEW YORK gespeichert. Der Wert, der dem Attribut `/Client/address/@type` zugeordnet ist, ist die kleingeschriebene Zeichenfolge 'permanent', die jedoch als PERMANENT gespeichert wird.

Ausführen von Abfragen, die Indizes verwenden, bei denen die Groß-/Kleinschreibung nicht beachtet werden muss

Ein Index, bei dem die Groß-/Kleinschreibung nicht beachtet werden muss, wird vom Optimierungsprogramm nur berücksichtigt, wenn das Indexmuster und das Vergleichselement die folgenden Bedingungen erfüllen:

- Der Pfad des Kontextschritts in der Klausel GENERATE KEYS USING XMLPATTERN der Anweisung CREATE INDEX stimmt mit dem XML-Pfad im Abfragevergleichselement überein.
- Der Name der Ländereinstellung, sofern er in der Anweisung CREATE INDEX angegeben ist, stimmt mit der durch die Funktion 'fn:upper-case()' im Abfragevergleichselement angegebenen Ländereinstellung überein.
- Der erste Parameter von 'fn:upper-case()', der im Abfragevergleichselement verwendet wird, ist ein Kontextelementausdruck (.).

Für die folgende Abfrage wählt das Optimierungsprogramm möglicherweise die Verwendung des Index 'clients_state_idx', bei dem die Groß-/Kleinschreibung nicht beachtet werden muss, aus, falls dieser vorhanden ist. Anstatt eine Tabellensuche durchzuführen, um nach Datensätzen zu suchen, bei denen das Element 'state' den Wert 'New York' aufweist (entweder in Groß- oder in Kleinschreibung), kann das Optimierungsprogramm die Durchsuchung des Index 'clients_state_idx' auswählen, falls dies weniger Arbeit bedeutet.

```
XQUERY db2-fn:xmlcolumn('CLIENTS.CONTACTINFO')
   /Client/address/state[fn:upper-case(.)="NEW YORK"];

--------------------------------
<state>New York</state>
<state>new york</state>

2 Satz/Sätze ausgewählt.
```

Angeben des Ländereinstellungsparameters

Wenn Sie einen Index erstellen, bei dem Groß-/Kleinschreibung nicht beachtet werden muss, können Sie den optionalen Ländereinstellungsparameter der Funktion 'fn:upper-case' verwenden. Die folgende Anweisung beispielsweise erstellt einen Index für das Adressattribut; geben Sie (zusammen mit dem Pfad `/Client/address/@type`) als Ländereinstellung tr_TR ein:

```
CREATE INDEX client_address_type_idx_tr ON clients(contactinfo)
   GENERATE KEYS USING XMLPATTERN '/Client/address/@type/fn:upper-case(., "tr_TR")'
   AS SQL VARCHAR(50);
```

Beachten Sie Folgendes: Wenn Sie die Ländereinstellungszeichenfolge nicht korrekt angeben, wenn Sie z. B. die umschließenden Anführungszeichen vergessen, wird für den Namen der Ländereinstellung ein Standardwert verwendet. Zur Überprüfung der Ländereinstellung während der Indexerstellung können Sie den Befehl **db2look** oder die Anweisung **DESCRIBE** verwenden. Beispiel: `DESCRIBE INDEXES FOR TABLE CLIENTS SHOW DETAIL`.

Das Optimierungsprogramm entscheidet sich möglicherweise für die Verwendung des Index 'client_address_type_idx_tr' nur dann, wenn in der Abfrage ebenfalls die Ländereinstellung tr_TR mit der Funktion 'fn:upper-case()' im Abfragevergleichselement angegeben wird. Beispiel:

```
SELECT id FROM clients client1
  WHERE XMLEXISTS('$XMLDOC/Client/address/@type[fn:upper-case(., "tr_TR")="PERMANENT"]'
  PASSING client1.contactinfo as "XMLDOC")

ID
-----------
         92
         93

  2 Satz/Sätze ausgewählt.
```

Für eine Abfrage wie die im folgenden Beispiel wird der Index 'client_address_type_idx_tr' nicht verwendet, da er eine andere Ländereinstellung angibt:

```
SELECT id FROM clients client1
  WHERE XMLEXISTS('$XMLDOC/Client/address/@type[fn:upper-case(., "en_US")="PERMANENT"]'
  PASSING client1.contactinfo as "XMLDOC")

ID
-----------
         92
         93

  2 Satz/Sätze ausgewählt.
```

Beispiele für die Verwendung von Indizes, die fn:exists angeben

Mithilfe der Funktion 'fn:exists' können Sie einen Index erstellen, der einen booleschen Wert speichert (entweder das Einzelzeichen T oder F), um anzugeben, ob es wahr oder falsch ist, dass ein Element oder ein Attribut einen Datenwert speichert (anstelle der leeren Sequenz).

Erstellen eines Index mit fn:exists

In diesem Beispiel wird veranschaulicht, wie eine Tabelle mit einer XML-Spalte erstellt wird, wie Daten dort eingefügt werden und wie ein Index mithilfe der Funktion fn:exists erstellt wird.

Erstellen Sie zunächst die Tabelle INCOME mit der Spalte INCOMEINFO vom Typ XML:

```
CREATE TABLE income (
  ID           INT PRIMARY KEY NOT NULL,
  INCOMEINFO   XML
);
```

Fügen Sie in die Tabelle INCOME drei Datensätze ein:

```
INSERT INTO income VALUES('1',
'<Employee>
    <salary type="regular">
         <base>5500.00</base>
         <bonus>1000.00</bonus>
    </salary>
</Employee>');

INSERT INTO income VALUES('2',
'<Employee>
    <salary type="contractor">
         <base>7600.00</base>
    </salary>
</Employee>');

INSERT INTO income VALUES('3',
'<Employee>
    <salary>
         <base>2600.00</base>
         <bonus>500.00</bonus>
    </salary>
</Employee>');
```

Sie können beispielsweise einen Index mit dem Namen **exists_bonus_idx** erstellen, um zu überprüfen, welche Mitarbeiter Boni erhalten haben; verwenden Sie hierbei den Pfad `/Employee/salary/fn:exists(bonus)`:

```
CREATE INDEX exists_bonus_idx ON
   income(incomeinfo) GENERATE KEYS USING XMLPATTERN
   '/Employee/salary/fn:exists(bonus)' AS SQL VARCHAR(1);
```

Sie können beispielsweise auch einen Index mit dem Namen **exists_any_attrib_idx** erstellen, um zu überprüfen, ob für das Element 'salary' Attribute vorhanden sind; verwenden Sie hierbei den Pfad `/Employee/salary/fn:exists(@*)`:

```
CREATE INDEX exists_any_attrib_idx ON
   income(incomeinfo) GENERATE KEYS USING XMLPATTERN
   '/Employee/salary/fn:exists(@*)' AS SQL VARCHAR(1);
```

Ausführen von Abfragen, die diese Indizes verwenden

Ein Index, der mit der Funktion 'fn:exists' erstellt wurde, wird vom Optimierungsprogramm nur dann berücksichtigt, wenn die beiden folgenden Anweisungen wahr sind:

- Der Pfad des Indexmusters stimmt mit dem XML-Pfad im Abfragevergleichselement überein.
- Das Abfragevergleichselement führt eine Suche für das Element oder das Attribut aus, das als Parameter von 'fn:exists' in der Anweisung CREATE INDEX angegeben wird.

Für die folgende Abfrage wählt das Optimierungsprogramm möglicherweise die Verwendung des Index **exists_bonus_idx** aus und führt dafür keine Tabellensuche durch, falls dies weniger Arbeit bedeutet:

```
XQUERY db2-fn:xmlcolumn('INCOME.INCOMEINFO')
    /Employee/salary[fn:exists(bonus)];

--------------------------------
<salary type="regular"><base>5500.00</base><bonus>1000.00</bonus></salary>
<salary><base>2600.00</base><bonus>500.00</bonus></salary>

2 Satz/Sätze ausgewählt.
```

Wenn diese Abfrage im folgenden Format erneut geschrieben wird, wird auch der Index **exists_bonus_idx** berücksichtigt:

```
XQUERY db2-fn:xmlcolumn('INCOME.INCOMEINFO')/Employee/salary[bonus];

--------------------------------
<salary type="regular"><base>5500.00</base><bonus>1000.00</bonus></salary>
<salary><base>2600.00</base><bonus>500.00</bonus></salary>

2 Satz/Sätze ausgewählt.
```

Der Index **exists_bonus_idx** wird für die folgenden beiden Abfragen berücksichtigt; dabei wird nach allen Mitarbeitern gesucht, die keinen Bonus erhalten haben:

```
XQUERY db2-fn:xmlcolumn('INCOME.INCOMEINFO')
   /Employee/salary[not(fn:exists(bonus))];

--------------------------------
<salary type="contractor"><base>7600.00</base></salary>

1 Satz/Sätze ausgewählt.

XQUERY db2-fn:xmlcolumn('INCOME.INCOMEINFO')
   /Employee/salary[fn:not(fn:exists(bonus))];
--------------------------------
<salary type="contractor"><base>7600.00</base></salary>

1 Satz/Sätze ausgewählt.
```

Für die folgende Abfrage wählt das Optimierungsprogramm möglicherweise die Verwendung des Index **exists_any_attrib_idx** aus. Dieser Index überprüft das Vorhandensein von Attributen für das Element 'salary'. In Bezug auf unsere Beispieldaten bedeutet dies, dass nur das Attribut 'type' vorhanden ist. Aus diesem Grund ist in diesem Fall das Abfragevergleichselement nur dann wahr, wenn das Attribut `@type` im XML-Pfad `/Employee/salary` vorhanden ist:

```
XQUERY db2-fn:xmlcolumn('INCOME.INCOMEINFO')
   /Employee/salary[fn:exists(@type)];
--------------------------------
```

```
<salary type="regular"><base>5500.00</base><bonus>1000.00</bonus></salary>
<salary type= "contractor"><base>7600.00</base></salary>

2 Satz/Sätze ausgewählt.
```

Der Index **exists_any_attrib_idx** wird vom Optimierungsprogramm auch für Abfragen berücksichtigt, die in den folgenden Formaten geschrieben sind:

```
XQUERY db2-fn:xmlcolumn('INCOME.INCOMEINFO')/Employee/salary[bonus and @type];

 --------------------------------
<salary type="regular"><base>5500.00</base><bonus>1000.00</bonus></salary>

1 Satz/Sätze ausgewählt.


XQUERY db2-fn:xmlcolumn('INCOME.INCOMEINFO')
  /Employee/salary[bonus and base > 3000];

 --------------------------------
<salary type="regular"><base>5500.00</base><bonus>1000.00</bonus></salary>

1 Satz/Sätze ausgewählt.


XQUERY db2-fn:xmlcolumn('INCOME.INCOMEINFO')
   /Employee/salary[bonus and bonus > 600];

 --------------------------------
<salary type="regular"><base>5500.00</base><bonus>1000.00</bonus></salary>

1 Satz/Sätze ausgewählt.


XQUERY db2-fn:xmlcolumn('INCOME.INCOMEINFO')
   /Employee/salary[bonus][bonus > 600];

 --------------------------------
<salary type="regular"><base>5500.00</base><bonus>1000.00</bonus></salary>

1 Satz/Sätze ausgewählt.


XQUERY for $e in db2-fn:xmlcolumn('INCOME.INCOMEINFO')
   /Employee/salary where $e/bonus return $e;

 --------------------------------
<salary type="regular"><base>5500.00</base><bonus>1000.00</bonus></salary>
<salary><base>2600.00</base><bonus>500.00</bonus></salary>

1 Satz/Sätze ausgewählt.
```

Indizes, die mit dem Schlüsselwort UNIQUE erstellt werden

Für einen Index, der mit dem Schlüsselwort UNIQUE erstellt wird, das ebenfalls die Funktion 'fn:exists' in der Klausel XMLPATTERN angibt, beschränkt die eindeutige Semantik die Darstellung von 'xml-wildcard' (und anderer Syntax, z. B. "//") auf lediglich den Kontextschritt des Indexmusters, nicht jedoch auf das Eingabeargument von fn:exists. Die folgende Anweisung ist beispielsweise gültig:

```
CREATE UNIQUE INDEX i2 ON tbx1(x1) GENERATE KEYS USING XMLPATTERN
'/node/node1/fn:exists(*)' AS SQL VARCHAR(1)
```

Die zweite Anweisung hingegen gibt einen Fehler zurück, da der Kontextschritt des Indexmusters nicht eindeutig ist:

```
CREATE UNIQUE INDEX i2 ON tbx1(x1) GENERATE KEYS USING XMLPATTERN
'/node/*/fn:exists(a)' AS SQL VARCHAR(1)
```

Beispiele für die Verwendung von Indizes mit Abfragen, die fn:starts-with angeben

Ab DB2 V10.1 kann das Abfrageprogramm die Verwendung eines Index vom Typ VARCHAR für Abfragen mit Vergleichselementen auswählen, die die Funktion 'fn-:starts-with' enthalten, um die Abfrage zu beschleunigen.

An vorhandenen Indizes des Typs VARCHAR sind keine Änderungen erforderlich und es muss keine besondere Syntax in der Anweisung CREATE INDEX verwendet werden, wenn neue VARCHAR-Indizes erstellt werden.

Die Funktion 'fn:starts-with' ermittelt, ob eine Zeichenfolge mit einer bestimmten Unterzeichenfolge beginnt.

Erstellen eines Index vom Typ VARCHAR

In diesem Beispiel wird veranschaulicht, wie eine Tabelle mit einer XML-Spalte erstellt wird, wie Daten in die Spalte eingefügt werden und wie ein Index des Typs VARCHAR erstellt wird.

Erstellen Sie zunächst die Tabelle FAVORITE_CDS mit der Spalte CDINFO vom Typ XML:

```
CREATE TABLE favorite_cds (
  NAME          CHAR(20) NOT NULL,
  CDID          BIGINT,
  CDINFO        XML
);
```

Fügen Sie in die Tabelle FAVORITE_CDS einen Datensatz ein:

```
INSERT INTO favorite_cds VALUES('John Peterson', 01,
'<FAVORITECDS>
    <CD>
         <TITLE>Top hits</TITLE>
         <ARTIST>Good Singer</ARTIST>
         <COMPANY>Top Records</COMPANY>
         <YEAR>1999</YEAR>
    </CD>
    <CD>
         <TITLE>More top hits</TITLE>
         <ARTIST>Better Singer</ARTIST>
         <COMPANY>Better Music </COMPANY>
         <YEAR>2005</YEAR>
    </CD>
    <CD>
         <TITLE>Even more top hits</TITLE>
         <ARTIST>Best Singer</ARTIST>
         <COMPANY>Best Music</COMPANY>
         <YEAR>2010</YEAR>
    </CD>
</FAVORITECDS>');
```

Sie können im Pfad `/FAVORITECDS/CD/YEAR` einen VARCHAR-Index erstellen, z. B. mit dem Namen **year_idx**.

```
CREATE INDEX year_idx ON favorite_cds (cdinfo)
   GENERATE KEYS USING XMLPATTERN '/FAVORITECDS/CD/YEAR'
   AS SQL VARCHAR(20);
```

Sie können im Pfad /FAVORITECDS/CD/COMPANY auch einen VARCHAR-Index erstellen, z. B. mit dem Namen **company_idx**. Im folgenden Beispiel wird ein Index für den Firmennamen erstellt, bei dem die Groß-/Kleinschreibung nicht beachtet werden muss; hierfür wird die Funktion 'fn:upper-case' verwendet:

```
CREATE INDEX company_idx ON favorite_cds (cdinfo)
   GENERATE KEYS USING XMLPATTERN '/FAVORITECDS/CD/COMPANY/fn:upper-case(.)'
   AS SQL VARCHAR(20);
```

Ausführen von Abfragen mit Vergleichselementen, die fn:starts-with enthalten

Für die folgende Abfrage wählt das Optimierungsprogramm möglicherweise die Verwendung des VARCHAR-Index **year_idx** aus und führt dafür keine Tabellensuche durch, um nach CDs aus den 1990er Jahren zu suchen (in diesem Beispiel wird mit der Funktion 'fn:starts-with' nach CDs gesucht, die mit einem Jahr mit 199 gekennzeichnet sind). Das Optimierungsprogramm kann die Verwendung des Index **year_idx** auswählen, falls dies weniger Arbeit bedeutet.

```
XQUERY for $y in db2-fn:xmlcolumn
   ('FAVORITE_CDS.CDINFO')/FAVORITECDS/CD
    [YEAR/fn:starts-with(., "199")] return $y

--------------------------------
<CD>
   <TITLE>Top hits</TITLE>
   <ARTIST>Good Singer</ARTIST>
   <COMPANY>Top Records</COMPANY>
   <YEAR>1999</YEAR>
</CD>
1 Satz/Sätze ausgewählt.
```

Beachten Sie, dass Abfragen, die in einem Format geschrieben werden, bei dem Sie einen Kontextelementausdruck (.) für den ersten Parameter der Funktion 'fn:starts-with' angegeben haben, das Optimierungsprogramm dazu veranlassen, nur eine einzige Indexsuche auszuwählen; für andere Formate hingegen können zwei Indexsuchen erforderlich sein. Aus diesem Grund wird eine ähnliche Abfrage, die im folgenden Format geschrieben ist, voraussichtlich langsamer ausgeführt:

```
XQUERY for $y in db2-fn:xmlcolumn
   ('FAVORITE_CDS.CDINFO')/FAVORITECDS/CD
    [fn:starts-with(YEAR, "199")] return $y
```

Für die nächste Abfrage wählt das Optimierungsprogramm möglicherweise die Verwendung des VARCHAR-Index **company_idx** aus und führt dafür keine Tabellensuche durch, um nach Datensätzen zu suchen, bei denen der Firmenname mit BES beginnt. Das Optimierungsprogramm kann die Verwendung des Index **company_idx** auswählen, falls dies weniger Arbeit bedeutet.

```
XQUERY for $y in db2-fn:xmlcolumn
   ('FAVORITE_CDS.CDINFO')/FAVORITECDS/CD
    [COMPANY/fn:starts-with(fn:upper-case(.), "BES")] return $y

--------------------------------
    <CD>
         <TITLE>Even more top hits</TITLE>
         <ARTIST>Best Singer</ARTIST>
         <COMPANY>Best Music</COMPANY>
         <YEAR>2010</YEAR>
    </CD>
1 Satz/Sätze ausgewählt.
```

Kontextschritte und Funktionsausdrucksschritte

Der Kontext- und der Funktionsausdrucksschritt sind Teil des XML-Indexmusters, das bei der Erstellung von funktionalen Indizes für XML-Daten angegeben wird.

Der Kontextschritt definiert das XML-Indexmuster einer Reihe von Elementen oder Attributknoten, die indiziert werden müssen. Der Funktionsausdrucksschritt gibt die Funktion an, die die Schlüsselwerte definiert, die im Index gespeichert werden sollen.

Bei der Erstellung von funktionalen Indizes zu XML-Daten, z. B. mithilfe der Funktionen 'fn:exists' und 'fn:upper-case', verfügt der XML-Musterausdruck, den Sie in der Klausel GENERATE KEYS USING XMLPATTERN angeben, im Allgemeinen über zwei Teile.

Der erste Teil wird *Kontextschritt* genannt. Der Kontextschritt gibt den XML-Pfad der Elementknoten oder der Attributknoten an, für die Indexeinträge erstellt werden. Die Syntax des Kontextschritts ist mit der Syntax für Indexmusterausdrücke für XML-Indizes im Allgemeinen identisch; es gibt jedoch eine Reihe von Einschränkungen im Hinblick auf die Verwendung zusammen mit bestimmten Funktionen, z. B. 'fn:upper-case' und 'fn:exists'. Details finden Sie in den Informationen zur Verwendung der Anweisung CREATE INDEX.

Der zweite Teil wird *Funktionsausdrucksschritt* genannt. Der Funktionsausdrucksschritt gibt die Funktion an, z. B. 'fn:exists' oder 'fn:upper-case', sowie deren Parameter. Der Funktionsausdrucksschritt produziert den eigentlichen Schlüsselwert, der im Index für jeden vom Kontextschritt angegebenen Knoten gespeichert werden soll.

So lautet der Kontextschritt für das XML-Indexmuster `/a/b/fn:upper-case(.)` beispielsweise wie folgt:

- `/a/b`
- Der Funktionsausdrucksschritt lautet `fn:upper-case(.)`.

Der Kontextschritt für das XML-Indexmuster `/a/b/fn:exists(c)` lautet beispielsweise wie folgt:

- `/a/b`
- Der Funktionsausdrucksschritt lautet `fn:exists(c)`.

XML-Namensbereichsdeklarationen

Zur Definition der Element- und Attributbefehle in XML-Musterausdrücken werden qualifizierte XML-Namen (QNames) verwendet. Das Qualifikationsmerkmal für einen QName ist ein Namensbereichspräfix, das einer URI für einen Namensbereich zugeordnet wurde.

Das XML-Muster kann mithilfe einer optionalen Namensbereichsdeklaration ('namespace') angegeben werden, um eine Zuordnung zwischen einem Namensbereichspräfix und des Zeichenfolgeliterals einer Namensbereichs-URI herzustellen oder eine Standard-URI für einen Namensbereich für das XML-Muster zu definieren. Das Namensbereichspräfix wird dann zum Qualifizieren von Namen für Elemente und Attribute in dem XML-Muster verwendet, um diese mit dem Namensbereich abzugleichen, der im Dokument verwendet wird.

Im folgenden Beispiel wird das verkürzte Namensbereichspräfix *m* der Adresse *http://www.mycompanyname.com/* zugeordnet.

```
CREATE INDEX empindex on department(deptdocs)
          GENERATE KEYS USING XMLPATTERN
          'declare namespace m="http://www.mycompanyname.com/";
          /m:company/m:emp/m:name/m:last' AS SQL VARCHAR(30)
```

Beachten Sie, dass bei Ausführung dieser Anweisung CREATE INDEX über den Befehlszeilenprozessor (CLP) das eingebettete Semikolon problematisch ist, weil das Semikolon das Standardanweisungsabschlusszeichen ist. Zur Vermeidung dieser Problematik verwenden Sie eine der folgenden Methoden:

- Stellen Sie sicher, dass das Semikolon nicht das letzte Nichtleerzeichen in der Zeile ist (indem Sie zum Beispiel einen leeren XQuery-Kommentar nach dem Semikolon hinzufügen).
- Ändern Sie das Standardanweisungsabschlusszeichen im CLP über die Befehlszeile.

Es können auch mehrere Namensbereichsdeklarationen im gleichen XMLPATTERN-Ausdruck angegeben werden, jedoch muss das Namensbereichspräfix innerhalb der Liste von Namensbereichsdeklarationen eindeutig sein. Darüber hinaus kann der Benutzer optional einen Standardnamensbereich für Elemente deklarieren, die kein Präfix haben. Wenn ein Namensbereich oder ein Namensbereichspräfix für ein Element nicht explizit angegeben ist, wird der Standardnamensbereich verwendet. Deklarationen von Standardnamensbereichen gelten nicht für Attribute. Wenn der Benutzer keinen Standardnamensbereich angibt, heißt der Namensbereich *no namespace*. Es kann nur ein Standardnamensbereich deklariert werden. Diese Funktionsweise für Namensbereichsdeklarationen entspricht den XQuery-Regeln.

Das vorherige Beispiel kann auch wie folgt mit einem Standardnamensbereich geschrieben werden:

```
CREATE INDEX empindex on department(deptdocs)
      GENERATE KEY USING XMLPATTERN
      'declare default element namespace "http://www.mycompany.com/";
        /company/emp/name/last') AS SQL VARCHAR(30)
```

Im nächsten Beispiel hat das Attribut *@id* den Namensbereich *no namespace*, da der Standardnamensbereich *http://www.mycompany.com/* nur für die Elemente *company* und *emp*, jedoch nicht für das Attribut *@id* gilt. Dies entspricht den XQuery-Grundregeln, da in einem XML-Dokument Deklarationen von Standardnamensbereichen nicht für Attribute gelten.

```
CREATE INDEX empindex on department(deptdocs)
      GENERATE KEY USING XMLPATTERN
      'declare default element namespace "http://www.mycompany.com/";
        /company/emp/@id' AS SQL VARCHAR(30)
```

Da das Attribut *@id* den gleichen Namensbereich wie die Elemente *company* und *emp* haben sollte, sollte die Anweisung wie folgt umgeschrieben werden:

```
CREATE INDEX empindex on department(deptdocs)
      GENERATE KEY USING XMLPATTERN
      'declare default element namespace "http://www.mycompany.com/";
        declare namespace m="http://www.mycompanyname.com/";
        /company/emp/@m:id' AS SQL VARCHAR(30)
```

Das Namensbereichspräfix, das zur Erstellung des Index dient, und das Namensbereichspräfix, das in den Instanzdokumenten verwendet wird, müssen nicht übereinstimmen, um indexiert zu werden, jedoch muss der vollständig ausgeschriebene

qualifizierte Name (QName) übereinstimmen. Das heißt, der Wert des Namensbereichs, in den das Präfix umgesetzt wird, ist von Bedeutung, nicht der Name des Präfix an sich. Wenn zum Beispiel das Namensbereichspräfix für den Index als *m="http//www.mycompany.com/"* und das Namensbereichspräfix, das im Instanzdokument verwendet wird, als *c="http//www.mycompany.com/"* definiert sind, wird *c:company/c:emp/@id* in den Instanzdokumenten indexiert, da die beiden verkürzten Namensbereichspräfixe *m* und *c* in denselben Namensbereich umgesetzt werden.

Datentypen in XML-Musterausdrücken für Indizes

Jedem XML-Musterausdruck, der in der Anweisung CREATE INDEX angegeben wird, muss ein Datentyp zugeordnet werden. Es werden die folgenden SQL-basierten Datentypen unterstützt: VARCHAR, DATE, TIMESTAMP, INTEGER, DECIMAL und DOUBLE.

Es ist möglich, das Ergebnis eines Ausdrucks mit unterschiedlichen Datentypen zu interpretieren. Zum Beispiel hat der Wert *123* eine Zeichenfolgedarstellung, kann jedoch auch als numerischer Wert *123* interpretiert werden. Wenn der Pfad */company/emp/@id* sowohl als Zeichendatenfolge als auch als numerischer Wert indexiert werden soll, müssen zwei Indizes erstellt werden: einer für den VARCHAR-Datentyp und ein weiterer für den Datentyp DOUBLE. Die Werte im Dokument werden in den jeweils für den Index angegebenen Datentyp konvertiert.

Die folgenden Beispiele zeigen, wie zwei Indizes mit unterschiedlichen Datentypen für dieselbe XML-Spalte *deptdocs* erstellt werden:

```
CREATE INDEX empindex1 on department(deptdocs)
GENERATE KEY USING XMLPATTERN '/company/emp/@id' AS SQL VARCHAR(10)

CREATE INDEX empindex2 on department(deptdocs)
GENERATE KEY USING XMLPATTERN '/company/emp/@id' AS SQL DOUBLE
```

Beschreibung der unterstützten SQL-Datentypen:

VARCHAR(*ganze_zahl*)
VARCHAR-Daten werden in einem Index zu einer XML-Spalte in der Codepage UTF-8 gespeichert. Wenn der Datentyp VARCHAR mit einer angegebenen Länge *integer* (in Byte) verwendet wird, wird die angegebene Länge als Beschränkung interpretiert. Wenn bei der Erstellung des Index Dokumente gleichzeitig in die Tabelle eingefügt werden oder bereits in der Tabelle vorhanden sind, schlägt das Einfügen des Dokuments oder die Indexerstellung fehl, wenn zu indexierende Knoten vorhanden sind, deren Werte länger als die angegebene Länge sind. Wenn die Einfügung oder Erstellung erfolgreich ausgeführt wird, speichert der Index garantiert alle Zeichenfolgewerte in ihrer Gesamtheit und kann sowohl Bereichssuchen als auch Suchen mit Gleichheitsvergleichselementen unterstützen. Die Länge *integer* ist ein Wert aus dem Bereich von 1 bis zu einem Maximalwert, der von der verwendeten Seitengröße abhängt. In der Dokumentation zur Anweisung CREATE INDEX finden Sie eine Liste der maximal zulässigen Längen. Bei Zeichenfolgevergleichen, für die folgende Leerzeichen von Bedeutung sind, wird die XQuery-Semantik verwendet. Diese unterscheidet sich von der SQL-Semantik, bei der folgende Leerzeichen in Vergleichen keine Bedeutung haben.

```
CREATE INDEX empindex1 on department(deptdocs)
GENERATE KEY USING XMLPATTERN '/company/emp/@id' AS SQL VARCHAR(50)
```

VARCHAR HASHED
Der Datentyp VARCHAR HASHED kann angeben werden, um die Indexierung von Zeichenfolgen beliebiger Längen zu verarbeiten. Wenn Dokumente zu indexierende Zeichenfolgen enthalten, die die maximale Länge *integer* über-

schreiten, die entsprechend dem von der Seitengröße abhängigen Maximalwert zulässig ist, können Sie alternativ den Datentyp VARCHAR HASHED angeben. In diesem Fall generiert das System einen acht Byte langen Hash-Code für die gesamte Zeichenfolge. Die Länge der indexierten Zeichenfolge unterliegt dabei keiner Beschränkung. Bei Angabe des Datentyps VARCHAR HASHED können keine Bereichssuchen ausgeführt werden, da der Index keine wirklichen Zeichendaten, sonder Hash-Codes enthält. Indizes, die solche Hash-Codes für Zeichenfolgen enthalten, können nur für Suchfunktionen mit Gleichheitsvergleichselementen verwendet werden. Bei Zeichenfolgevergleichen mit Gleichheitsvergleichselementen, für die folgende Leerzeichen von Bedeutung sind, wird die XQuery-Semantik verwendet. Diese unterscheidet sich von der SQL-Semantik, bei der folgende Leerzeichen in Vergleichen keine Bedeutung haben. Der Hashwert für die Zeichenfolge ermöglicht bei Suchen mit Gleichheitsvergleichselementen die Verwendung der XQuery-Semantik, jedoch nicht die Verwendung der SQL-Semantik.

```
CREATE INDEX empindex on company(companydocs)
GENERATE KEY USING XMLPATTERN '/company/emp/name/last' AS SQL
VARCHAR HASHED
```

DOUBLE
Alle numerischen Werte werden in den Datentyp DOUBLE konvertiert und im Index gespeichert. Unbegrenzte ('unbounded') Dezimaltypen und 64-Bit-Integerwerte büßen möglicherweise Stellen ein, wenn sie als DOUBLE gespeichert werden. Die Werte für den SQL-Indexdatentyp DOUBLE können die numerischen Sonderwerte *NaN*, *INF*, *-INF*, *+0* und *-0* mit einschließen, obwohl der SQL-Datentyp DOUBLE diese Werte selbst nicht unterstützt.

INTEGER
Alle numerischen Werte, die dem XML-Schematyp xs:int entsprechen, werden in den Datentyp INTEGER konvertiert und im Index gespeichert. Beachten Sie, dass das XML-Schema mehrere Ganzzahldatentypen definiert, einschließlich xs:int und xs:integer. Die Grenzwerte von xs:int entsprechen den Grenzwerten des SQL-Typs INTEGER, xs:integer dagegen definiert keine Grenzwerte. Wenn ein Wert auftritt, der die Grenzwerte von xs:int überschreitet, wird ein Fehler zurückgegeben.

```
CREATE INDEX intidx on favorite_cds(cdinfo)
   GENERATE KEYS USING XMLPATTERN '/favoritecds/cd/year'
   AS SQL INTEGER
```

Das Wort INT kann als Synonym für den SQL-Datentyp INTEGER verwendet werden.

DECIMAL(*integer*, *integer*)
Alle numerischen Werte werden in den Datentyp DECIMAL konvertiert und im Index gespeichert. Die erste ganze Zahl (Integer) ist die Genauigkeit der Zahl, d. h. die Gesamtzahl der Ziffern; diese kann zwischen 1 und 31 liegen. Die zweite ganze Zahl (Integer) ist die Nachkommastelle der Zahl, d. h. die Anzahl der Ziffern rechts neben dem Dezimalzeichen; diese kann zwischen 0 und der Genauigkeit der Zahl liegen. Wenn die Genauigkeit und die Nachkommastelle nicht angegeben werden, wird der Standardwert 5,0 verwendet.

```
CREATE INDEX decidx on favorite_cds(cdinfo) GENERATE KEYS USING XMLPATTERN
   '//price' AS SQL DECIMAL(5,2)
```

Die Wörter DEC, NUMERIC und NUM können als Synonyme für DECIMAL verwendet werden.

DATE
Werte des Datentyps DATE werden vor dem Speichern im Index mit der Welt-

zeit (UTC - Coordinated Universal Time) bzw. Zulu-Zeit normalisiert. Beachten Sie, dass der XML-Schemadatentyp für DATE eine höhere Genauigkeit (mehr Stellen) als der SQL-Datentyp zulässt. Wenn ein Wert außerhalb des gültigen Bereichs angetroffen wird, wird ein Fehler zurückgegeben.

```
CREATE INDEX BirthdateIndex ON personnel(xmlDoc)
   GENERATE KEY USING XMLPATTERN '/Person/Confidential/Birthdate' AS SQL DATE
```

TIMESTAMP

Werte des Datentyps TIMESTAMP werden vor dem Speichern im Index mit der Weltzeit (UTC - Coordinated Universal Time) bzw. Zulu-Zeit normalisiert. Beachten Sie, dass der XML-Schemadatentyp für Zeitmarken eine höhere Genauigkeit (mehr Stellen) als der SQL-Datentyp zulässt. Wenn ein Wert außerhalb des gültigen Bereichs angetroffen wird, wird ein Fehler zurückgegeben.

```
CREATE INDEX LastLogonIndex ON machines(xmlDoc)
   GENERATE KEY USING XMLPATTERN '/Machine/Employee/LastLogon' AS SQL TIMESTAMP
```

Datentypkonvertierung für Indizes für XML-Daten

Bevor Werte in den Index für XML-Daten eingefügt werden können, werden sie zunächst in den XML-Indexdatentyp konvertiert, der dem SQL-Indexdatentyp entspricht.

Für die Datentypen VARCHAR(*integer*) und VARCHAR HASHED wird der Wert mit der XQuery-Funktion 'fn:string' in einen xs:string-Wert umgewandelt. Das Längenattribut von VARCHAR(*integer*) wird als Integritätsbedingung auf den resultierenden xs:string-Wert angewendet. Ein SQL-Indexdatentyp VARCHAR HASHED wendet einen Hashalgorithmus auf den resultierenden xs:string-Wert an, um einen Hash-Code zu generieren, der in den Index eingefügt wird. Daten für VARCHAR-Datentypen werden direkt im Index gespeichert, ohne sie zunächst mit dem Schemadatentyp zu normalisieren.

Für DOUBLE-, INTEGER-, DECIMAL-, DATE- und TIMESTAMP-Indizes wird der Wert mithilfe des XQuery-Umsetzungsausdrucks in den XML-Indextyp konvertiert. Werte der Datentypen DATE und TIMESTAMP werden vor dem Speichern im Index mit der Weltzeit (UTC - Coordinated Universal Time) bzw. Zulu-Zeit normalisiert. XML-Daten, die gemäß den XQuery-Regeln zwar gültig sind, jedoch aufgrund von Systembegrenzungen nicht in den Indexdatentyp umgewandelt werden können, verursachen einen Indexierungsfehler. Die Werte für den SQL-Indexdatentyp DOUBLE können die numerischen Sonderwerte *NaN*, *INF*, *-INF*, *+0* und *-0* mit einschließen, obwohl der SQL-Datentyp DOUBLE selbst diese Werte nicht unterstützt.

Entsprechende Indexdatentypen

Tabelle 18. Entsprechende Indexdatentypen

XML-Datentyp	SQL-Datentyp
xs:string	VARCHAR(*integer*) und VARCHAR HASHED
xs:double	DOUBLE
xs:date	DATE
xs:dateTime	TIMESTAMP
xs:int	INTEGER
xs:decimal	DECIMAL(*integer*, *integer*)

Ungültige XML-Werte

Bei XML-Musterwerten handelt es sich um die indexierten Werte, die von der *XML-Musterklausel* der Anweisung CREATE INDEX generiert werden.

Für Indizes mit den Datentypen DOUBLE, INTEGER, DECIMAL, DATE und TIMESTAMP wird ein XML-Musterwert mithilfe des XQuery-Umsetzungsausdrucks in den XML-Indexdatentyp konvertiert. XML-Werte, die kein gültiges lexikalisches Format für den XML-Datentyp des Zielindex besitzen, gelten als ungültige XML-Werte.

`ABC` beispielsweise ist ein ungültiger XML-Wert für den Datentyp 'xs:double'. Die Art und Weise, in der die ungültigen XML-Werte vom Index verarbeitet werden, ist davon abhängig, ob die Option REJECT INVALID VALUES oder die Option IGNORE INVALID VALUES in der *XML-Typklausel* der Anweisung CREATE INDEX angegeben wird.

REJECT INVALID VALUES

Gibt an, dass alle XML-Musterwerte im Kontext der lexikalischen Definition des XML-Indexdatentyps gültig sein müssen. Außerdem muss der Wert im Wertebereich des XML-Indexdatentyps liegen. Im Abschnitt zu den zugehörigen Referenzen, der später folgt, finden Sie Links zu Details zur lexikalischen Definition und zum Wertebereich für die einzelnen Datentypen. Beispiel: Wenn Sie die Klausel REJECT INVALID VALUES bei der Erstellung eines Index vom Typ INTEGER erstellen, geben XML-Musterwerte wie 3.5, 3.0, 3e0, 'A123' und 'hello' einen Fehler zurück (SQLSTATE 23525). XML-Daten werden nicht in die Tabelle eingefügt oder dort aktualisiert, wenn der Index bereits vorhanden ist (SQLSTATE 23525). Ist der Index nicht vorhanden, wird der Index nicht erstellt (SQLSTATE 23526).

Nehmen Sie zum Beispiel an, ein Benutzer erstellt den Index EMPID, der die numerischen Mitarbeiter-IDs als Datentyp DOUBLE indexiert. Numerische Werte wie beispielsweise `31201` werden indexiert. Wenn eines der Dokumente jedoch versehentlich die Abteilungs-ID `M55` als einen der Werte des Attributs für Mitarbeiter-IDs verwendet, schlägt das Einfügen des Dokuments mit einer Fehlernachricht fehl, da `M55` ein ungültiger Wert für den Typ DOUBLE ist.

```
CREATE INDEX EMPID ON DEPARTMENT(DEPTDOCS)
GENERATE KEY USING XMLPATTERN '//@id' AS SQL DOUBLE   REJECT INVALID VALUES
```

IGNORE INVALID VALUES

Gibt an, dass XML-Musterwerte, die ungültige lexikalische Formate für den XML-Zielindexdatentyp sind, ignoriert werden und dass die entsprechenden Werte in den gespeicherten XML-Dokumenten von der Anweisung CREATE INDEX nicht indexiert werden. Standardmäßig werden ungültige Werte ignoriert. Bei der Durchführung von Einfüge- und Aktualisierungsoperationen werden die ungültigen XML-Musterwerte zwar nicht indexiert, die XML-Dokumente jedoch weiterhin in die Tabelle eingefügt. Es tritt weder ein Fehler noch eine Warnung auf, da die Angabe dieser Datentypen keine Integritätsbedingung für die XML-Musterwerte ist (XQuery-Ausdrücke, die nach bestimmten XML-Indexdatentypen suchen, berücksichtigen diese Werte nicht).

Die Regeln dafür, welche XML-Musterwerte ignoriert werden können, werden vom angegebenen SQL-Datentyp festgelegt.

- Wenn es sich bei dem SQL-Datentyp um einen Zeichenfolgedatentyp handelt, werden XML-Musterwerte in keinem Fall ignoriert, da alle Zeichenfolgen gültig sind.

- Wenn es sich bei dem SQL-Datentyp um einen numerischen Datentyp handelt, wird jeder XML-Musterwert, der nicht dem lexikalischen Format des XML-Datentyps xs:double entspricht, ignoriert. Wenn der XML-Musterwert nicht den spezifischeren lexikalischen Formaten des XML-Datentyps entspricht, der dem numerischen SQL-Datentyp des Index entspricht, wird ein Fehler zurückgegeben. Beispiel: Wenn der SQL-Datentyp INTEGER lautet, entsprechen die XML-Musterwerte 3.5, 3.0 und 3e0 dem lexikalischen Format von xs:double; es wird jedoch ein Fehler zurückgegeben (SQLSTATE 23525), weil sie nicht dem lexikalischen Format von xs:int entsprechen. XML-Musterwerte wie 'A123' oder 'hello' werden für denselben Index ignoriert.
- Wenn es sich bei dem SQL-Datentyp um einen Datentyp für Datum und Uhrzeit handelt, wird jeder XML-Musterwert, der nicht dem lexikalischen Format des entsprechenden XML-Datentyps (xs:date oder xs:dateTime) entspricht, ignoriert.

Wenn ein XML-Musterwert dem entsprechenden lexikalischen Format entspricht, wird ein Fehler zurückgegeben, wenn der Wert außerhalb des Wertebereichs für den Datentyp liegt oder die maximale Länge oder Genauigkeit und Nachkommastelle des angegebenen SQL-Datentyps überschreitet. Ist der Index nicht vorhanden, wird der Index nicht erstellt (SQLSTATE 23526).

Werden ungültige XML-Musterwerte für den Datentyp ignoriert, fungiert der XML-Datentyp des Zielindex als Filter und stellt keine Integritätsbedingung dar, da der Benutzer mehrere Indizes mit unterschiedlichen Datentypen für dieselbe XML-Spalte definieren kann. Nehmen Sie zum Beispiel an, ein Benutzer erstellt zwei Indizes für das gleiche Muster, jedoch mit unterschiedlichen Datentypen. Der Index ALLID arbeitet mit einem VARCHAR-Datentyp und indexiert alle IDs im Dokument (Abteilungs-IDs und Mitarbeiter-IDs). Der Index EMPID indexiert hingegen nur die numerischen Mitarbeiter-IDs und verwendet den Datentyp DOUBLE als Filter:

Verwendung der expliziten Option IGNORE INVALID VALUES:

```
CREATE INDEX ALLID ON DEPARTMENT(DEPTDOCS)
      GENERATE KEY USING XMLPATTERN '//@id' AS SQL VARCHAR(10)
      IGNORE INVALID VALUES

CREATE INDEX EMPID ON DEPARTMENT(DEPTDOCS)
      GENERATE KEY USING XMLPATTERN '//@id' AS SQL DOUBLE
      IGNORE INVALID VALUES
```

Logisch entsprechende Anweisungen mit dem Standardwert:

```
CREATE INDEX ALLID ON DEPARTMENT(DEPTDOCS)
      GENERATE KEY USING XMLPATTERN '//@id' AS SQL VARCHAR(10)

CREATE INDEX EMPID ON DEPARTMENT(DEPTDOCS)
      GENERATE KEY USING XMLPATTERN '//@id' AS SQL DOUBLE
```

Der Wert der Abteilungs-ID `M25` ist ein gültiger Wert für den Datentyp VARCHAR und wird in den Index ALLID eingefügt. Der Wert `M25` kann jedoch nicht in den Datentyp DOUBLE konvertiert werden, sodass der Wert nicht in den Index EMPID eingefügt wird. Dies verursacht keine Fehlernachricht oder Warnung. Der Wert wird für das in der Tabelle gespeicherte Dokument eingefügt.

Obwohl der Wert `M25` im Index EMPID mit dem Datentyp DOUBLE nicht vorhanden ist, wird der Index mit dem Datentyp DOUBLE möglicherweise dennoch von Abfragen zum Abruf aller übereinstimmenden numerischen

Werte verwendet. Dabei treten keine Konvertierungsfehler auf, weil auf das Dokument, das den Wert `M25` enthält, nicht zugegriffen wird.

Wenn die Abfrage jedoch den Index EMPID mit dem Datentyp DOUBLE nicht verwendet und das Dokument unter Verwendung des Vergleichselements `//@id=25` durchsucht, tritt ein Konvertierungsfehler auf, weil der Wert `M25` dem Muster entspricht und weiterhin im Dokument vorhanden ist, jedoch keinen numerischen Wert darstellt.

Beachten Sie, dass alle Werte im Dokument für den Datentyp 'xs:string' (SQL-Datentyp VARCHAR) gültig sind.

Zurückweisung von Dokumenten oder Fehlschlagen von Anweisungen CREATE INDEX

Bei den Indexierungsfehlern SQLSTATE 23525 oder sqlcode -20305 wird ein XML-Dokument für INSERT- oder UPDATE-Anweisungen zurückgewiesen. Bei einer Anweisung CREATE INDEX für eine mit Daten gefüllte Tabelle mit XML-Spalten schlägt die Anweisung CREATE INDEX fehl (SQLSTATE 23526, sqlcode -20306) und das Dokument bleibt in der Tabelle gespeichert.

Fehler bei Integritätsbedingung für Datenlänge des Typs VARCHAR(*ganze_zahl*)
Die Länge eines resultierenden Indexwerts aus einem oder mehreren XML-Musterausdrücken überschreitet die benutzerdefinierte Längenvorgabe für den VARCHAR-Datentyp.

Fehler aufgrund eines nicht unterstützten Listendatentypknotens
Ein oder mehrere XML-Knotenwerte in einem XML-Wert stellen einen Listendatentypknoten dar, der durch den angegebenen Index nicht indexiert werden kann. Listendatentypknoten werden von Indizes für XML-Daten nicht unterstützt.

Konvertierungsfehler
Es wird ein Fehler zurückgegeben, wenn der Quellenwert für den XML-Indexdatentyp ungültig ist und die Option REJECT INVALID VALUES mit der Anweisung CREATE INDEX angegeben wurde. Ein Fehler wird auch dann zurückgegeben, wenn der Quellenwert ein gültiger XML-Wert ist, der jedoch aufgrund interner DB2-Begrenzungen nicht in die DB2-Datenbankserverdarstellung für den Schemadatentyp bzw. für den XML-Indexdatentyp konvertiert werden kann. Der Fehler muss zur Wahrung konsistenter Ergebnisse ausgegeben werden: Wenn eine Abfrage ausgeführt würde, die den Index verwendet, könnte das korrekte Ergebnis der Abfrage einen Wert enthalten, der die unterstützte Begrenzung überschreitet, da der Wert ein gültiger XML-Wert wäre. Um zu verhindern, dass die Abfrage ein unvollständiges Ergebnis zurückgibt, wird ein Fehler zurückgegeben, um konsistente Ergebnisse beizubehalten.

Tabelle 19. Einige Beispiele für interne DB2-Begrenzungen

XML-Datentyp	XML-Schema	DB2-Bereich (min : max)
xs:date	Kein Maximalwert für Jahre Negative Datumswerte werden unterstützt.	0001-01-01: 9999-12-31

Tabelle 19. Einige Beispiele für interne DB2-Begrenzungen (Forts.)

XML-Datentyp	XML-Schema	DB2-Bereich (min : max)
xs:dateTime	Kein Maximalwert für Jahre Negative Datumswerte werden unterstützt. Für Sekundendezimalstellen wird eine beliebige Genauigkeit unterstützt.	0001-01-01T00:00:00.000000Z: 9999-12-31T23:59:59.999999Z
xs:integer	Keine Begrenzung für den Minimal- oder Maximalbereich	-9223372036854775808: 9223372036854775807

Der DB2-Datenbankserver unterstützt nicht den gesamten Bereich von XML-Werten. Zu den nicht unterstützten Wertebereichen gehören:

- Date- oder dateTime-Werte mit Jahr > 9999 oder < 0
- Date- oder dateTime-Werte mit einer Genauigkeit für Sekundendezimalwerte > 6 Stellen
- Außerhalb des gültigen Bereichs liegende numerische Werte

Übersichtstabelle für die Konvertierung in den XML-Indexdatentyp

Zur erfolgreichen Konvertierung von Daten in den XML-Datentyp des Zielindex muss der Quellenwert gemäß dem Schemadatentyp bzw. XML-Indexdatentyp lexikalisch gültig sein und innerhalb der DB2-Begrenzungen für den Schemadatentyp bzw. XML-Indexdatentyp liegen.

In der folgenden Tabelle finden Sie eine Zusammenfassung der Behandlung von Werten bei der Konvertierung in den XML-Indexdatentyp.

Tabelle 20. Übersichtstabelle für die Konvertierung in den XML-Indexdatentyp

Der Wert ist gemäß dem XML-Indexdatentyp gültig (alle Werte sind für den Datentyp 'xs:string' gültig)	Der Wert liegt innerhalb der DB2-Begrenzungen für den XML-Indexdatentyp	Indexierungsergebnis
Nein	Nicht zutreffend	REJECT INVALID VALUES: Fehler IGNORE INVALID VALUES (Standardwert): Der Wert wird ignoriert und nicht indexiert.
Ja	Ja	Der Wert wird indexiert.
Ja	Nein	Fehler: Der Wert liegt außerhalb der DB2-Begrenzungen.

XML-Schemata und Indexschlüsselgenerierung

Eine Untersuchung Ihrer XML-Schemata gibt Aufschluss darüber, wie Sie Ihre Indizes zu XML-Spalten mit Datentypen erstellen können, die den Datentypspezifikationen der XML-Schemata entsprechen. Bei der Entscheidung, welche XML-Muster für Indizes auszuwählen sind, müssen auch die Abfragen berücksichtigt werden, die ausgeführt werden sollen.

Bei Verwendung eines XML-Schemas wird die Struktur von XML-Dokumenten, die in einer XML-Spalte gespeichert werden sollen, überprüft, sodass die Datentypen der Elemente und Attribute in den XML-Dokumenten durch das XML-Schema beschränkt werden. Entspricht ein Dokument den Spezifikationen des Schemas nicht, wird das Dokument vom Parser zurückgewiesen. Wenn zum Beispiel das Schema angibt, dass ein Attribut auf den Datentyp DOUBLE beschränkt ist, und dieses Attribut in einem Dokument den Wert *ABC* hat, wird das Dokument zurückgewiesen. Wenn kein XML-Schema verwendet wird, werden die Dokumentdaten nicht durch den Parser überprüft und als untypisierte Daten betrachtet.

Nehmen Sie zum Beispiel an, dass das folgende XML-Schema verwendet wird:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:element name="product" type="ProdType"/>

<xsd:simpleType name="ColorType">
    <xsd:restriction base="xsd:string">
        <xsd:maxLength value='20'/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="ProdType">
      <xsd:sequence>
        <xsd:element name="name"    type="xsd:string" />
        <xsd:element name="SKU"     type="xsd:string" />
        <xsd:element name="price"   type="xsd:integer" />
        <xsd:element name="comment" type="xsd:string" />
      </xsd:sequence>
    <xsd:attribute name="color"   type="ColorType" />
    <xsd:attribute name="weight"  type="xsd:integer" />
  </xsd:complexType>
</xsd:schema>
```

Bei der Prüfung der auszuführenden Abfragen stellt sich vielleicht heraus, dass Indizes für Preis und Farbe benötigt werden. Eine Analyse der Abfragen hilft Ihnen bei der Entscheidung, welcher XML-Musterausdruck in der Anweisung CREATE INDEX anzugeben ist. Das XML-Schema gibt Aufschluss darüber, welcher Datentyp für den Index zu wählen ist. Sie können feststellen, dass das Preiselement ('price') eine Dezimalzahl ist, sodass der numerische Datentyp DECIMAL für den Index *priceindex* in Betracht kommt. Beim Index *colorindex* kann für das Farbattribut ('color'), das als Zeichenfolge ('string') gekennzeichnet ist, der Datentyp VARCHAR ausgewählt werden.

```
XQUERY for $i in db2-fn:xmlcolumn('COMPANY.PRODUCTDOCS')/product[price > 5.00]
   return $i/name
XQUERY for $i in db2-fn:xmlcolumn('COMPANY.PRODUCTDOCS')/product[@color = 'pink']
   return $i/name
```

```
CREATE INDEX priceindex on company(productdocs)
      GENERATE KEY USING XMLPATTERN '/product/price' AS DECIMAL(7,2)
CREATE INDEX colorindex on company(productdocs)
      GENERATE KEY USING XMLPATTERN '//@color' AS SQL VARCHAR(80)
```

Das Schema kann darüber hinaus weitere Einschränkungen für den Zeichenfolgedatentyp angeben. In obigem Beispiel wird unter *ColorType* die Einschränkung *maxLength* angegeben, die die Zeichenfolge auf 20 Unicode-Zeichen begrenzt. Da die Anweisung CREATE INDEX die Länge des VARCHAR-Datentyps in Byte und nicht in Zeichen angibt, kann die im Schema angegebene Länge mit dem Faktor 4 multipliziert werden, um die maximale Anzahl Byte zu berechnen, die zum Speichern der längsten vom Schema zugelassenen Zeichenfolge im Index erforderlich ist. In diesem Fall müsste für die Berechnung also 4*20 = 80 angesetzt und der Datentyp VARCHAR(80) für den Index *colorindex* ausgewählt werden.

Wenn das Schema keine Längenbeschränkung für einen Zeichenfolgedatentyp angibt und die maximale Zeichenfolgelänge für die Werte in den Dokumenten nicht bekannt ist, können Sie die maximale Länge verwenden, die bei der vom Index verwendeten Seitengröße zulässig ist. Ein Index speichert Zeichenfolgen variierender Längen. Da jedoch nur die tatsächliche Anzahl von Byte für jede Zeichenfolge gespeichert wird, entsteht kein Speicherverlust, wenn eine längere Maximallänge als erforderlich angegeben wird. Allerdings müssen größere Schlüsselpuffer im Hauptspeicher zugeordnet werden, um die maximale Schlüsselgröße bei Indexsuchen verarbeiten zu können. In der Dokumentation zur Anweisung CREATE INDEX finden Sie eine Liste der maximal zulässigen Längen eines Index zu einer XML-Spalte, der mit dem VARCHAR-Datentyp definiert wird.

Wenn die maximale Länge für den VARCHAR-Datentyp nicht ausreicht, um die Dokumentwerte zu indexieren, kann der Datentyp VARCHAR HASHED verwendet werden, der keiner Längenbeschränkung unterliegt. Indizes mit dem Datentyp VARCHAR HASHED können jedoch nur für Suchen mit Gleichheitsvergleichselementen und nicht für Wertebereichssuchen verwendet werden. Beachten Sie, dass Dokumente, die längere Zeichenfolgen enthalten, als der angegebene Datentyp VARCHAR(*integer*) zulässt, zurückgewiesen werden.

Das XML-Schema kann auch Standardattribute und Standardelementwerte angeben. Wenn in einem XML-Dokument keine entsprechenden Werte angegeben sind und das Dokument geprüft wird, werden beim Speichern des Dokuments die Standardwerte aus dem Schema verwendet. Diese Standardwerte werden zusammen mit den anderen, ursprünglich im Eingabedokument vorhandenen Werten indexiert. Wenn das Dokument nicht geprüft wird, werden die Standardwerte dem Dokument nicht hinzugefügt und auch nicht indexiert.

Semantik des Schlüsselworts UNIQUE

Das gleiche Schlüsselwort UNIQUE, das bei Indizes für Nicht-XML-Spalten verwendet wird, kann auch bei Indizes zu XML-Spalten eingesetzt werden, hat jedoch in diesem Fall eine andere Bedeutung.

Bei relationalen Indizes dient das Schlüsselwort UNIQUE in der Anweisung CREATE INDEX dazu, die Eindeutigkeit von Einträgen innerhalb aller Zeilen einer Tabelle zu erzwingen. Bei Indizes zu XML-Daten wird das Schlüsselwort UNIQUE hingegen dazu verwendet, die Eindeutigkeit innerhalb einer XML-Spalte über alle Dokumente hinweg zwingend festzulegen, deren Knoten durch das XML-Muster qualifiziert werden. Die Einfügung eines einzelnen Dokuments kann dazu führen, dass mehrere Werte in einen eindeutigen Index eingefügt werden. Diese Werte müssen in diesem Dokument und in allen anderen Dokumenten in derselben XML-Spalte eindeutig sein. Beachten Sie außerdem, dass die Einfügung einiger Dokumente möglicherweise keine Einfügung von Werten in einen Index zur Folge hat. Für solche Dokumente wird die Eindeutigkeit nicht umgesetzt.

Die Eindeutigkeit wird für den Datentyp des Index, den XML-Pfad zum Knoten und den Wert des Knotens erzwungen, nachdem der XML-Wert in den SQL-Datentyp umgewandelt wurde, der für den Index angegeben ist.

Bei der Verwendung des Schlüsselwortes UNIQUE sollten Sie mit großer Vorsicht vorgehen. Da die Umwandlung in den angegebenen Datentyp für den Index zu Genauigkeits- oder Bereichsverlusten führen kann, oder dazu, dass über den verwendeten Hashalgorithmus unterschiedliche Werte in denselben Schlüsselwert umgesetzt werden, können mehrere Werte, die innerhalb des XML-Dokuments scheinbar eindeutig sind, zu Fehlern aufgrund doppelter Schlüssel führen. Solche Fehler aufgrund doppelter Schlüssel können in folgenden Situationen auftreten:

- Wenn VARCHAR HASHED angegeben wurde, dann werden eindeutige Zeichenfolgen möglicherweise in denselben Hash-Code umgesetzt und führen dann zu Fehlern aufgrund doppelter Schlüssel.
- Bei Indizes des Typs DOUBLE kann es während einer Einfügeoperation durch einen Genauigkeitsverlust oder durch Werte, die sich außerhalb des Bereichs für den Datentyp DOUBLE befinden, zu Fehlern aufgrund doppelter Schlüssel kommen. Bei großen ganzen Zahlen und unbegrenzten Dezimalwerten kann es z. B. zu einem Verlust der Genauigkeit kommen, wenn diese mit dem Datentyp DOUBLE im Index gespeichert werden. Für diese Arten von Daten ist es sinnvoller, einen Index vom Typ INTEGER bzw. DECIMAL zu erstellen.

Wenn VARCHAR(*integer*) angegeben ist, wird die gesamte Zeichenfolge aus dem XML-Dokument im Index gespeichert, sodass keine Fehler aufgrund doppelter Schlüssel auftreten können. Darüber hinaus orientiert sich die Eindeutigkeit von Zeichenfolgen an der XQuery-Semantik, bei der abschließende Leerzeichen berücksichtigt werden. Aus diesem Grund werden Werte, die in SQL als doppelt vorhanden eingestuft werden, sich jedoch in Bezug auf die abschließenden Leerzeichen unterscheiden, in einem Index für XML-Daten als eindeutige Werte interpretiert.

```
CREATE UNIQUE INDEX EMPINDEX ON company(companydocs)
    GENERATE KEY USING XMLPATTERN '/company/emp/name/last' AS SQL
          VARCHAR(100)
```

Bei UNIQUE-Indizes muss im XML-Muster ein einziger, vollständiger Pfad angegeben werden, der keines der folgenden Elemente enthalten darf:

- descendant-Achse
- descendant-or-self-Achse
- /descendant-or-self::node()/ (//)
- Beliebige Platzhalterzeichen für den XML-Namenstest
- node()- oder instruction()-Element für die Verarbeitung beim XML-Sortentest

Zugehörige Datenbankobjekte für die XML-Datenindexierung

Logische und physische Indizes für XML-Daten

Wenn Sie einen Index für XML-Daten erstellen, werden zwei B-Tree-Indizes erstellt: ein logischer und ein physischer Index.

Der logische Index enthält die XML-Musterinformationen, die in der Anweisung CREATE INDEX angegeben wurden. Der physische Index enthält von DB2 generierte Schlüsselspalten zur Unterstützung des logischen Index sowie die indexierten Dokumentwerte, die in den Datentyp umgewandelt wurden, der in der Klausel *xmltype* der Anweisung CREATE INDEX angegeben wurde.

Die Arbeit mit einem Index für XML-Daten findet auf der logischen Ebene statt (zum Beispiel mit den Anweisungen CREATE INDEX und DROP INDEX). Die Verarbeitung des zugrunde liegenden physischen Index durch DB2 erfolgt für Sie transparent. Beachten Sie, dass physische Indizes von keiner Anwendungsprogrammierschnittstelle erkannt werden, die Indexmetadaten zurückgibt.

In der Katalogsicht SYSCAT.INDEXES hat der logische Index den Indexnamen, den Sie in der Anweisung CREATE INDEX angegeben haben, und den Indextyp *XVIL*. Der physische Index hat einen vom System generierten Namen und den Indextyp *XVIP*. Der logische Index wird immer zuerst erstellt und mit einer Index-ID (IID) versehen. Der physische Index wird unmittelbar anschließend erstellt und erhält die nächst folgende Index-ID.

Die Beziehung zwischen logischen und physischen Indizes wird im folgenden Beispiel veranschaulicht. Betrachten Sie die beiden Indizes für XML-Daten: *EMPINDEX* und *IDINDEX*. Bei *EMPINDEX* hat der logische Index den Namen *EMPINDEX*, die Index-ID *3* und den Indextyp *XVIL*. Der entsprechende physische Index besitzt den vom System generierten Namen *SQL060414134408390*, die Index-ID *4* und den Indextyp *XVIP*.

Tabelle 21. Beziehung zwischen logischen und physischen Indizes

Indexname (INDNAME)	Index-ID (IID)	Tabellenname (TABNAME)	Indextyp (INDEXTYPE)
SQL060414133259940	1	COMPANY	XRGN
SQL060414133300150	2	COMPANY	XPTH
EMPINDEX	3	COMPANY	XVIL
SQL060414134408390	4	COMPANY	XVIP
IDINDEX	5	COMPANY	XVIL
SQL060414134408620	6	COMPANY	XVIP

Katalogsichten

Weitere Informationen zu den folgenden Katalogsichten finden Sie im Abschnitt mit den zugehörigen Verweisen.

SYSCAT.INDEXES
Jede Zeile stellt einen Index dar, einschließlich logischer und physischer Indizes für XML-Daten.

SYSCAT.INDEXXMLPATTERNS
Jede Zeile stellt eine Musterklausel in einem Index zu XML-Daten dar.

Indizes in einer Umgebung mit partitionierten Datenbanken

Sie können die Anweisungen CREATE INDEX und ALTER INDEX in einer beliebigen Datenbankpartition absetzen.

Wenn Sie einen Index für eine Tabelle erstellen, die über mehrere Datenpartitionen partitioniert ist, wird der Index ebenfalls über die Datenpartitionen partitioniert.

Prüfung

Indizes für XML-Spalten verwenden den vorhandenen Indexobjekttyp für Prüfungen (Prüffunktion). Nur der logische Index wird geprüft, nicht der physische Index.

Andere zu XML-Spalten zugeordnete Datenbankobjekte

Es gibt zwei intern vom System generierte Indizes, die zu XML-Spalten zugeordnet sind. Die Indizes werden in der Katalogsicht SYSCAT.INDEXES dargestellt. Darüber hinaus werden in der Katalogsicht SYSCAT.INDEXPARTITIONS Indexpartitionen für datenpartitionierte Tabellen mit XML-Daten dargestellt.

XML-Pfadindex und XML-Bereichsindex

Immer wenn Sie eine XML-Spalte erstellen, wird von DB2 automatisch ein XML-Pfadindex zu der XML-Spalte erstellt. Darüber hinaus erstellt DB2 einen einzigen XML-Bereichsindex zu allen XML-Spalten innerhalb einer Tabelle.

Der XML-Pfadindex zeichnet alle eindeutigen Pfade auf, die innerhalb von XML-Dokumenten vorhanden sind, die in einer XML-Spalte gespeichert werden.

Der XML-Bereichsindex erfasst die Bereiche (Regions), in die ein XML-Dokument intern unterteilt ist, wobei Bereiche in diesem Fall Gruppen von Knoten auf jeweils einer Seite sind. Da Bereiche Gruppen von Knoten auf jeweils einer Seite sind, lässt sich die Anzahl von Bereichsindexeinträgen verringern und die Leistung verbessern, wenn eine größere Seitengröße verwendet wird, sodass mehr Knoten auf einer Seite gespeichert werden können.

Bei einer datenpartitionierten Tabelle ist der XML-Bereichsindex stets ein partitionierter Index und die XML-Spaltenpfadindizes sind stets nicht partitionierte Indizes.

Die Tabellenbereichs-ID und die Objekt-ID des XML-Bereichsindex in SYSCAT.INDEXES sind logische Werte, da der Bereichsindex stets partitioniert und der Einzeleintrag in SYSCAT.INDEXES eine logische Darstellung ist. Die Tabellenbereichs-ID und die Objekt-ID sind mit den entsprechenden IDs der partitionierten Tabelle identisch, der sie zugeordnet sind.

Jede Indexpartition, die in einer partitionierten Tabelle einer Datenpartition zugeordnet ist, hat einen Eintrag in der Katalogtabelle SYSIBM.SYSINDEXPARTITIONS, der Informationen und Statistikdaten zu dieser Indexpartition enthält. Auf diese Informationen kann über die Katalogsicht SYSCAT.INDEXPARTITIONS zugegriffen werden. Nicht partitionierte Indizes haben keine Einträge in dieser Katalogtabelle.

Sowohl der XML-Pfadindex als auch der XML-Bereichsindex werden in SYSCAT.INDEXES gespeichert. Beachten Sie, dass diese Indizes von keiner Anwendungsprogrammierschnittstelle erkannt werden, die Indexmetadaten zurückgibt.

Diese internen, den XML-Spalten zugeordneten Indizes unterscheiden sich von den vom Benutzer erstellten Indizes zu XML-Daten. Bei der Indexierung von XML-Daten, wie sie in XML-Spalten gespeichert sind, arbeiten Sie nur mit den logischen Indizes zu XML-Spalten, indem Sie zum Beispiel die Anweisungen CREATE INDEX und DROP INDEX verwenden.

Katalogsichten

Weitere Informationen zu diesen Katalogsichten finden Sie im Abschnitt mit den zugehörigen Verweisen.

SYSCAT.INDEXES
Jede Zeile stellt einen Index, einschließlich XML-Pfadindizes und XML-Bereichsindizes, dar. Der XML-Pfadindex wird als *XPTH* in SYSCAT.INDEXES.INDEXTYPE angezeigt, der XML-Bereichsindex als *XRGN* in SYSCAT.INDEXES.INDEXTYPE.

SYSCAT.INDEXPARTITIONS
Jede Zeile stellt eine Indexpartition dar, die in einer partitionierten Tabelle einer Datenpartition zugeordnet ist.

Erneute Erstellung von Indizes für XML-Daten

Ein Index für XML-Daten wird unter den folgenden Umständen erneut erstellt:

- Bei einem **REORG INDEX**- oder **REORG INDEXES**-Befehl.
- Bei einem **REORG TABLE**-Befehl.
- Wenn ein Befehl **IMPORT** mit der Option REPLACE abgesetzt wird.
- Wenn eine Abfrage-, Einfüge-, Lösch- oder Aktualisierungsoperation versucht, auf eine Tabelle oder einen Index zuzugreifen und feststellt, dass das Indexobjekt als ungültig markiert ist.

Hinweise

- Alle Indizes, die mit der Funktionalität des nativen XML-Datenspeichers verbunden sind, sind in dem gleichen Indexobjekt für eine Tabelle wie relationale Indizes enthalten. Dies schließt alle etwaigen XML-Pfadindizes, XML-Bereichsindizes und Indizes für XML-Daten mit ein, die möglicherweise vorhanden sind. Einzelne Indizes werden nicht allein erneut erstellt. Wenn eine Indexneuerstellung erforderlich wird, werden alle Indizes im Indexobjekt zusammen erneut erstellt.
- Während eines Befehls **REORG TABLE** für eine partitionierte Tabelle wird der XML-Pfadindex nicht erstellt, wenn die Option LOBGLOBDATA nicht angegeben ist.

CREATE INDEX

Mit der Anweisung CREATE INDEX können Sie einen Index für eine DB2-Tabelle definieren. Indizes können für XML-Daten oder relationale Daten definiert werden. Die Anweisung CREATE INDEX wird auch verwendet, um eine Indexspezifikation zu erstellen (Metadaten, die dem Optimierungsprogramm anzeigen, dass eine Datenquelle einen Index besitzt).

Aufruf

Diese Anweisung kann in ein Anwendungsprogramm eingebettet oder mithilfe dynamischer SQL-Anweisungen abgesetzt werden. Es handelt sich um eine ausführbare Anweisung, die nur dann dynamisch vorbereitet werden kann, wenn für das Paket das Ausführungsverhalten DYNAMICRULES wirksam ist (SQLSTATE 42509).

Berechtigung

Die Berechtigungs-ID der Anweisung muss mindestens eine der folgenden Berechtigungen aufweisen:

- Eines der folgenden Zugriffsrechte:

– Zugriffsrecht CONTROL für die Tabelle oder den Kurznamen, für die bzw. den der Index definiert wurde
– Zugriffsrecht INDEX für die Tabelle oder den Kurznamen, für die bzw. den der Index definiert wurde

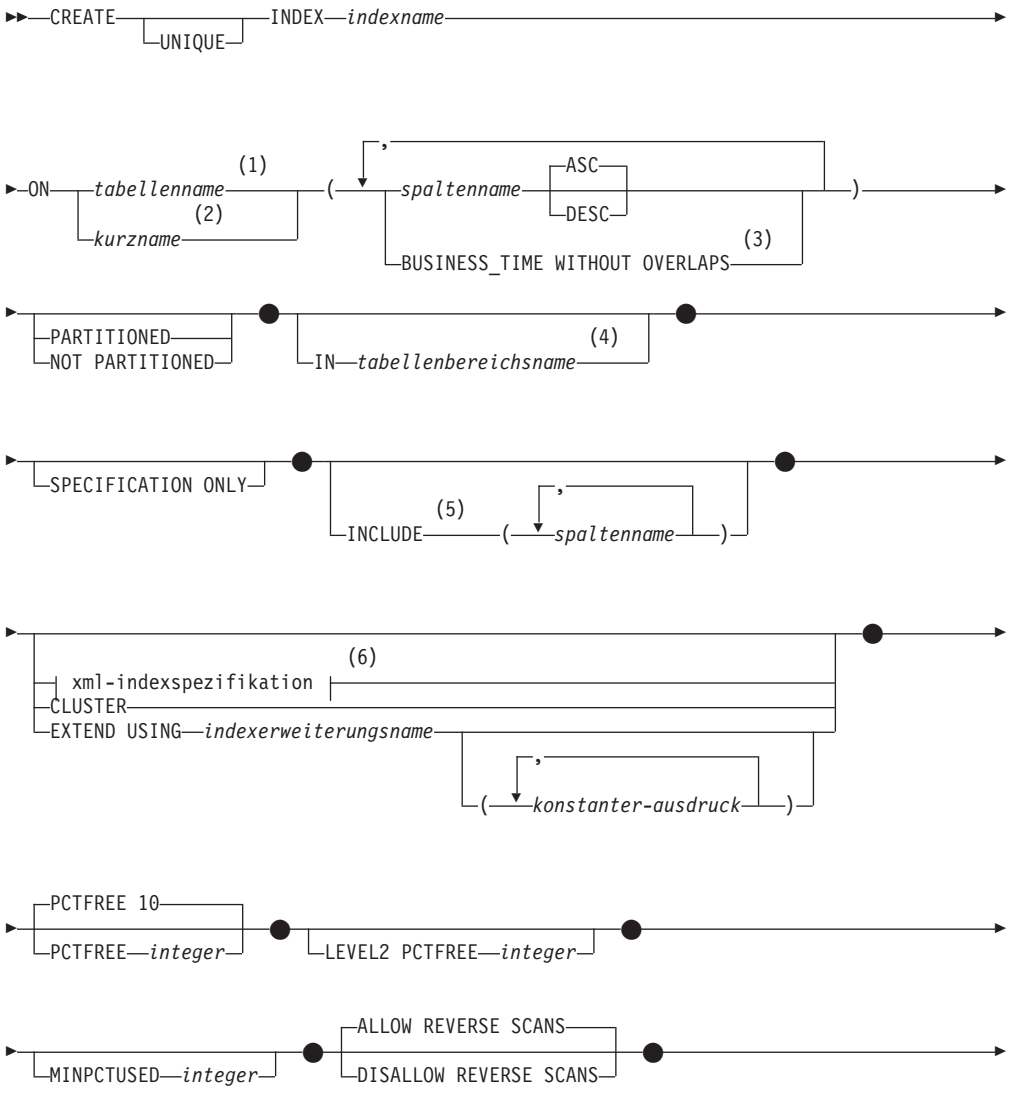
und die folgende Berechtigung oder das folgende Zugriffsrecht:

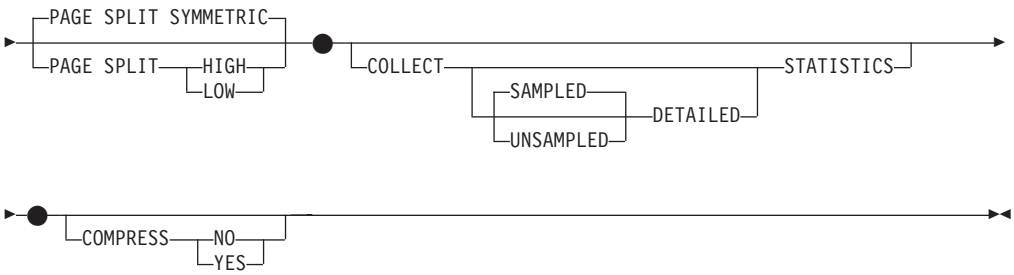
– Berechtigung IMPLICIT_SCHEMA für die Datenbank, wenn der implizite oder der explizite Schemaname des Index nicht vorhanden ist
– Zugriffsrecht CREATEIN für das Schema, wenn der Schemaname des Index auf ein vorhandenes Schema verweist

• Berechtigung DBADM

Zum Erstellen eines Index für eine deklarierte temporäre Tabelle ist kein explizites Zugriffsrecht erforderlich.

Syntax





Anmerkungen:

1 In einem föderierten System muss *tabellenname* eine Tabelle in diesem föderierten System angeben. Es darf keine Tabelle einer Datenquelle angegeben werden.

2 Bei der Angabe von *kurzname* erstellt die Anweisung CREATE INDEX eine Indexspezifikation. In diesem Fall kann INCLUDE, *xml-indexspezifikation*, CLUSTER, EXTEND USING, PCTFREE, MINPCTUSED, DISALLOW REVERSE SCANS, ALLOW REVERSE SCANS, PAGE SPLIT oder COLLECT STATISTICS nicht angegeben werden.

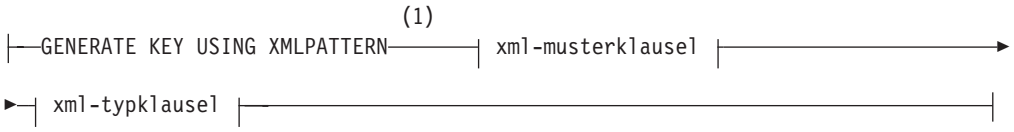
3 Die Klausel BUSINESS_TIME WITHOUT OVERLAPS kann nur angegeben werden, wenn UNIQUE angegeben ist.

4 Die Klausel IN *tabellenbereichsname* kann nur für einen nicht partitionierten Index für eine partitionierte Tabelle angegeben werden.

5 Die Klausel INCLUDE kann nur angegeben werden, wenn UNIQUE angegeben ist.

6 Wenn *xml-indexspezifikation* angegeben wird, kann *spaltenname* DESC, INCLUDE oder CLUSTER nicht angegeben werden.

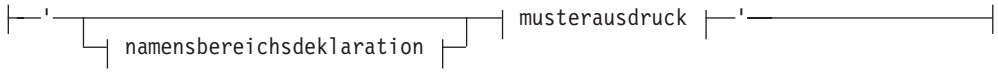
xml-indexspezifikation:



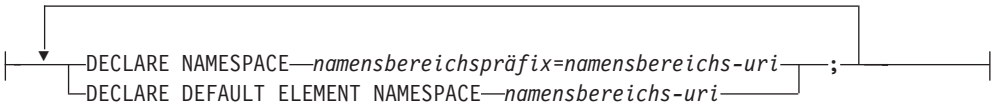
Anmerkungen:

1 Die alternative Syntax GENERATE KEYS USING XMLPATTERN kann verwendet werden.

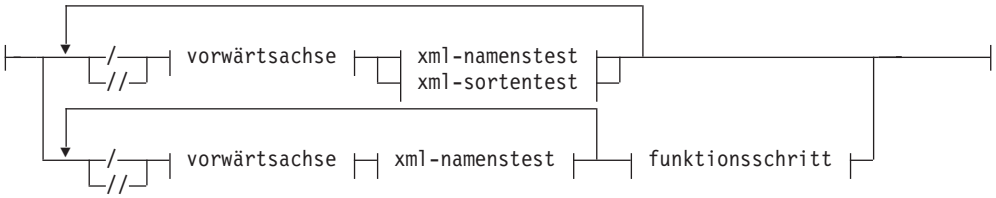
xml-musterklausel:



namensbereichsdeklaration:



musterausdruck:



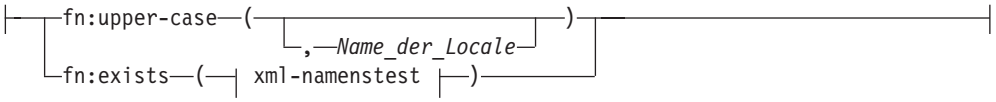
vorwärtsachse:

xml-namenstest:

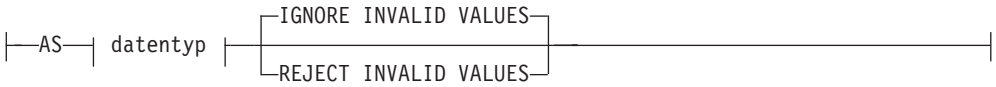
xml-platzhalter:

xml-sortentest:

funktionsschritt:

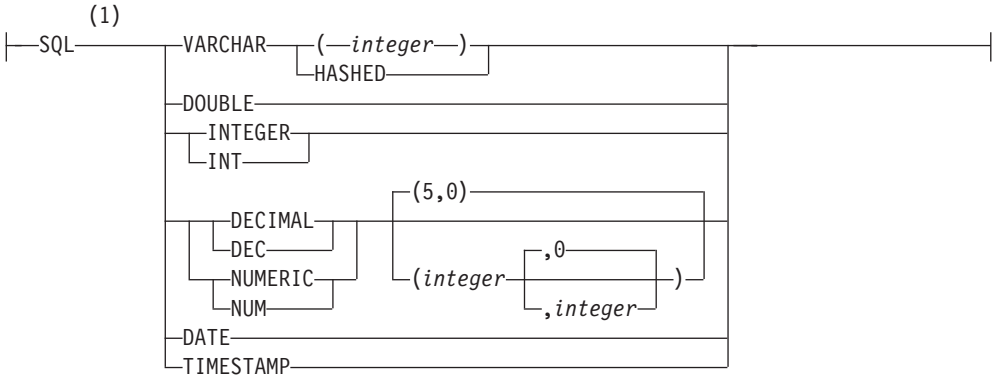


xml-typklausel:



datentyp:

sql-datentyp:



Anmerkungen:

1 Wenn Sie einen Funktionsnamen wie fn:upper-case am Ende des XML-Musters angeben, dann sind die unterstützten Indexdatentypen möglicherweise eine Untergruppe der hier angezeigten Indexdatentypen. Sie können in der Beschreibung für die xmlpattern-Klausel nach gültigen Indexdatentypen suchen.

Beschreibung

UNIQUE

Wenn ON *tabellenname* angegeben wird, verhindert UNIQUE, dass die Tabelle zwei oder mehr Zeilen mit demselben Indexschlüsselwert enthält. Die Eindeutigkeit wird am Ende der SQL-Anweisung erzwungen, mit der Zeilen aktualisiert oder neue Zeilen eingefügt werden.

Die Eindeutigkeit wird auch während der Ausführung der Anweisung CREATE INDEX geprüft. Wenn die Tabelle bereits Zeilen mit doppelten Schlüsselwerten enthält, wird der Index nicht erstellt.

Bei einem Index für eine XML-Spalte (wenn es sich um einen Index zu XML-Daten handelt) gilt die Eindeutigkeit für Werte mit dem angegebenen Musterausdruck (*musterausdruck*) in allen Tabellenzeilen. Die Eindeutigkeit wird für jeden Wert erzwungen, nachdem dieser in den angegebenen SQL-Datentyp (*sql-datentyp*) konvertiert wurde. Da die Konvertierung in den angegebenen SQL-Datentyp (*sql-datentyp*) zu Genauigkeits- oder Bereichsverlusten führen kann oder dazu, dass über den verwendeten Hashalgorithmus unterschiedliche Werte in denselben Schlüsselwert umgesetzt werden, können mehrere Werte, die innerhalb des XML-Dokuments scheinbar eindeutig sind, zu Fehlern aufgrund doppelter Schlüssel führen. Die Eindeutigkeit von Zeichenfolgen hängt von der XQuery-Semantik ab, bei der abschließende Leerzeichen berücksichtigt

werden. Aus diesem Grund werden Werte, die in SQL als doppelt vorhanden eingestuft werden, sich jedoch in Bezug auf die abschließenden Leerzeichen unterscheiden, in einem Index für XML-Daten als eindeutige Werte interpretiert.

Bei Verwendung von UNIQUE werden Nullwerte wie alle anderen Werte behandelt. Wenn der Schlüssel beispielsweise eine einzelne Spalte ist, die Nullwerte enthalten kann, darf diese Spalte höchstens einen einzigen Nullwert enthalten.

Wenn die Option UNIQUE angegeben wird und die Tabelle einen Verteilungsschlüssel besitzt, müssen die Spalten in dem Indexschlüssel eine Obermenge des Verteilungsschlüssels sein. Das bedeutet, dass die für einen eindeutigen Indexschlüssel angegebenen Spalten alle Spalten des Verteilungsschlüssel enthalten müssen (SQLSTATE 42997).

Wenn die Option UNIQUE angegeben wird und die Tabelle einen Tabellenpartitionierungsschlüssel besitzt, müssen die Spalten des Indexschlüssels eine Obermenge des Tabellenpartitionierungsschlüssels sein. Das bedeutet, dass die für den eindeutigen Indexschlüssel angegebenen Spalten alle Spalten des Tabellenpartitionierungsschlüssels enthalten müssen (SQLSTATE 42990).

Primäre oder eindeutige Schlüssel können keine Untermengen von Dimensionen sein (SQLSTATE 429BE).

Wenn ON *kurzname* angegeben wird, darf UNIQUE nur angegeben werden, wenn die Daten für den Indexschlüssel in jeder Zeile der Datenquellentabelle eindeutige Werte enthalten. Die Eindeutigkeit wird nicht geprüft.

Bei einem Index für XML-Daten kann UNIQUE nur eingeschlossen werden, wenn der Kontextschritt des Musterausdrucks (*musterausdruck*) einen einzelnen, vollständigen Pfad angibt und keine descendant- oder descendant-or-self-Achse, "//", keinen XML-Platzhalter (*xml-platzhalter*) und weder *node()* noch *processing-instruction()* enthält (SQLSTATE 429BS).

In einer Umgebung mit partitionierten Datenbanken gelten die folgenden Regeln für eine Tabelle mit mindestens einer XML-Spalte:

- Eine verteilte Tabelle kann keinen eindeutigen Index zu XML-Daten haben.
- Ein eindeutiger Index zu XML-Daten wird nur bei einer Tabelle unterstützt, die keinen Verteilungsschlüssel besitzt und die sich in einer Mehrpartitionsdatenbank mit Einzelnknoten befindet.
- Falls ein eindeutiger Index zu XML-Daten bei einer Tabelle vorhanden ist, kann die Tabelle nicht geändert werden, um einen Verteilungsschlüssel hinzuzufügen.

INDEX *indexname*
Benennt den Index oder die Indexspezifikation. Der Name, einschließlich des impliziten oder expliziten Qualifikationsmerkmals, darf weder einen Index noch eine Indexspezifikation, der bzw. die im Katalog beschrieben ist, noch einen vorhandenen Index für eine deklarierte temporäre Tabelle angeben (SQLSTATE 42704). Das Qualifikationsmerkmal darf nicht SYSIBM, SYSCAT, SYSFUN oder SYSSTAT sein (SQLSTATE 42939).

Das implizite oder explizite Qualifikationsmerkmal für Indizes für deklarierte temporäre Tabellen muss SESSION sein (SQLSTATE 428EK).

ON *tabellenname* **oder** *kurzname*
tabellenname gibt eine Tabelle an, für die ein Index erstellt werden muss. Bei der Tabelle muss es sich um eine Basistabelle (keine Sicht), eine erstellte temporäre Tabelle, eine deklarierte temporäre Tabelle, eine auf dem aktuellen Ser-

ver vorhandene, gespeicherte Abfragetabelle (MQT) oder eine deklarierte temporäre Tabelle handeln. Der Name einer deklarierten temporären Tabelle muss mit SESSION qualifiziert werden. *tabellenname* darf keine Katalogtabelle angeben (SQLSTATE 42832). Wenn UNIQUE angegeben wird und *tabellenname* für eine typisierte Tabelle steht, darf diese keine untergeordnete Tabelle sein (SQLSTATE 429B3).

kurzname ist der Kurzname, für den eine Indexspezifikation zu erstellen ist. Der *kurzname* verweist auf eine Datenquellentabelle, deren Index durch die Indexspezifikation beschrieben wird, oder auf eine Datenquellensicht, die auf einer solchen Tabelle basiert. Der *kurzname* muss im Katalog aufgelistet sein.

spaltenname
Bei einem Index gibt *spaltenname* eine Spalte an, die Teil des Indexschlüssels werden soll. Bei einer Indexspezifikation ist *spaltenname* der Name, mit dem der Server mit föderierten Datenbanken auf eine Spalte einer Datenquellentabelle verweist.

Jeder Spaltenname (*spaltenname*) muss ein nicht qualifizierter Name sein, der eine Spalte in einer Tabelle angibt. Die Anzahl der Spalten addiert mit der doppelten Anzahl der ermittelten Punkte darf 64 nicht überschreiten (SQLSTATE 54008). Wenn *tabellenname* eine typisierte Tabelle ist, kann die Anzahl der Spalten 63 nicht überschreiten (SQLSTATE 54008). Wenn *tabellenname* eine untergeordnete Tabelle ist, muss *spaltenname* mindestens einmal in die untergeordnete Tabelle eingeführt werden, darf also nicht aus einer übergeordneten Tabelle übernommen werden (SQLSTATE 428DS). Es darf kein Vorkommen von *spaltenname* wiederholt werden (SQLSTATE 42711).

Die Summe der gespeicherten Feldlängen der angegebenen Spalten darf die für die Seitengröße geltende Längenbegrenzung für Indexschlüssel nicht überschreiten. Informationen zu Schlüssellängenbegrenzungen finden Sie in der Beschreibung der SQL-Grenzwerte. Wenn *tabellenname* eine typisierte Tabelle ist, reduziert sich die Längenbegrenzung für Indexschlüssel um weitere 4 Byte. Beachten Sie, dass diese Längenbegrenzung durch Systemaufwand weiter verringert werden kann, der je nachdem, welchen Datentyp die Spalte hat und ob sie Nullwerte enthalten kann, variiert. Weitere Informationen zum Einfluss von Systemaufwand auf diese Begrenzung finden Sie im Abschnitt „Bytezähler" unter „CREATE TABLE".

Beachten Sie, dass diese Länge durch Systemaufwand verringert werden kann, der je nachdem, welchen Datentyp die Spalte hat und ob sie Nullwerte enthalten kann, variiert. Weitere Informationen zum Einfluss von Systemaufwand auf diese Begrenzung finden Sie im Abschnitt „Bytezähler" unter „CREATE TABLE".

Es darf keine auf LOB basierende LOB-Spalte oder Spalte mit einem einzigartigen Datentyp in einem Index verwendet werden, selbst wenn das Längenattribut der Spalte so klein ist, dass die Längenbegrenzung für Indexschlüssel für die Seitengröße eingehalten wird (SQLSTATE 54008). Eine Spalte mit einem strukturierten Typ kann nur angegeben werden, wenn auch die Klausel EXTEND USING angegeben wird (SQLSTATE 42962). Bei Angabe der Klausel EXTEND USING kann nur eine Spalte angegeben werden und deren Typ muss ein strukturierter Typ oder einzigartiger Datentyp sein, der nicht auf einem LOB basiert (SQLSTATE 42997).

Wenn ein Index nur eine einzige Spalte hat und deren Datentyp XML ist und wenn außerdem die Klausel GENERATE KEY USING XMLPATTERN angegeben wird, handelt es sich um einen Index für XML-Daten. Eine Spalte mit dem XML-Datentyp kann nur angegeben werden, wenn auch die Klausel GENERA-

TE KEY USING XMLPATTERN angegeben wird (SQLSTATE 42962). Bei Angabe der Klausel GENERATE KEY USING XMLPATTERN kann nur eine einzige Spalte angegeben werden, und deren Typ muss XML sein.

ASC
Gibt an, dass die Indexeinträge in aufsteigender Reihenfolge der Spaltenwerte sortiert werden sollen. Dies ist die Standardeinstellung. ASC kann nicht angegeben werden, wenn Indizes mit EXTEND USING definiert werden (SQLSTATE 42601).

DESC
Gibt an, dass die Indexeinträge in absteigender Reihenfolge der Spaltenwerte sortiert werden sollen. DESC kann nicht angegeben werden, wenn Indizes mit EXTEND USING definiert werden oder es sich um einen Index für XML-Daten handelt (SQLSTATE 42601).

BUSINESS_TIME WITHOUT OVERLAPS
BUSINESS_TIME WITHOUT OVERLAPS kann nur für einen Index angegeben werden, der als UNIQUE (SQLSTATE 428HW) definiert wurde, um anzuzeigen, dass die Werte für den Rest der angegebenen Schlüssel im Hinblick auf alle Zeiträume eindeutig sind. BUSINESS_TIME WITHOUT OVERLAPS kann nur als letztes Element der Liste angegeben werden. Wenn BUSINESS_TIME WITHOUT OVERLAPS angegeben ist, dann werden das Ende und der Anfang der Spalte automatisch in absteigender Reihenfolge zum Indexschlüssel hinzugefügt und erzwingen, dass dort keinerlei Überlappungen von Zeiträumen vorkommen. Wenn BUSINESS_TIME WITHOUT OVERLAPS angegeben ist, dürfen die Spalten des Zeitraums BUSINESS_TIME nicht als Schlüsselspalten, als Spalten im Partitionierungsschlüssel oder als Spalten im Verteilungsschlüssel (SQLSTATE 428HW) angegeben werden.

PARTITIONED
Gibt an, dass ein partitionierter Index erstellt werden soll. Der *tabellenname* muss eine mit Datenpartitionen definierte Tabelle angeben (SQLSTATE 42601).

Wenn die Tabelle partitioniert ist und weder PARTITIONED noch NOT PARTITIONED angegeben ist, wird (mit wenigen Ausnahmen) ein partitionierter Index erstellt. Ein nicht partitionierter Index wird anstelle eines partitionierten Index erstellt, wenn eine der folgenden Situationen eintritt:

- UNIQUE wird angegeben und der Indexschlüssel enthält nicht alle Tabellenpartitionierungsschlüsselspalten.
- Ein räumlicher Index wird erstellt.
- Der Index wird zu XML-Daten definiert.

Ein partitionierter Index mit einer Definition, die die gleichen Werte wie die Definition eines nicht partitionierten Index aufweist, wird nicht als Index mit doppelten Werten betrachtet. Weitere Details hierzu finden Sie im Abschnitt „Regeln” auf Seite 200 dieses Kapitels.

Das Schlüsselwort PARTITIONED kann für die folgenden Indizes nicht angegeben werden:

- Einen Index zu einer nicht partitionierten Tabelle (SQLSTATE 42601)
- Einen Index, der zu XML-Daten definiert wurde (SQLSTATE 42613)
- Einen eindeutigen Index, dessen Indexschlüssel nicht alle Tabellenpartitionierungsschlüsselspalten enthält (SQLSTATE 42990)
- Einen räumlichen Index (SQLSTATE 42997)

Ein partitionierter Index kann nicht für eine partitionierte Tabelle erstellt werden, die abhängige Tabellen mit aufgehobener Zuordnung, z. B. MQTs, besitzt (SQLSTATE 55019).

Die Tabellenbereichsplatzierung für eine Indexpartition des partitionierten Index wird von den folgenden Regeln bestimmt:

- Wenn die zu indexierende Tabelle mithilfe der *partitionstabellenbereich-optionen*-Klausel INDEX IN der Anweisung CREATE TABLE erstellt wurde, wird die Indexpartition in dem Tabellenbereich erstellt, der in dieser INDEX IN-Klausel angegeben ist.
- Wenn in der Anweisung CREATE TABLE für die zu indexierende Tabelle die *partitionstabellenbereich-optionen*-Klausel INDEX IN nicht angegeben ist, wird der partitionierte Index der Indexpartition im selben Tabellenbereich erstellt wie die entsprechende Datenpartition, die er indexiert.

Die Klausel IN der Anweisung CREATE INDEX wird für partitionierte Indizes nicht unterstützt (SQLSTATE 42601). Die *tabellenbereich-klauseln*-Klausel INDEX IN der Anweisung CREATE TABLE wird für partitionierte Indizes ignoriert. Wenn BUSINESS_TIME WITHOUT OVERLAPS als Indexschlüssel angegeben ist, dürfen die Partitionierungsschlüsselspalten keine Anfangs- oder Endespalten des Zeitraums BUSINESS_TIME enthalten (SQLSTATE 428HW).

NOT PARTITIONED
Gibt an, dass ein nicht partitionierter Index erstellt werden soll, der alle für die Tabelle definierten Datenpartitionen umfasst. Der *tabellenname* muss eine mit Datenpartitionen definierte Tabelle angeben (SQLSTATE 42601).

Ein nicht partitionierter Index mit einer Definition, die die gleichen Werte wie die Definition eines partitionierten Index aufweist, wird nicht als Index mit doppelten Werten betrachtet. Weitere Details hierzu finden Sie im Abschnitt „Regeln” auf Seite 200 dieses Kapitels.

Die Tabellenbereichsplatzierung für einen nicht partitionierten Index wird von den folgenden Regeln bestimmt:

- Bei Angabe der Klausel IN der Anweisung CREATE INDEX wird der nicht partitionierte Index in den Tabellenbereich gestellt, der in dieser IN-Klausel angegeben ist.
- Wenn Sie die Klausel IN der Anweisung CREATE INDEX nicht angeben, bestimmen die folgenden Regeln die Tabellenbereichsplatzierung des nicht partitionierten Index:
  - Wenn die zu indexierende Tabelle mithilfe der *tabellenbereich-klauseln*-Klausel INDEX IN der Anweisung CREATE TABLE erstellt wurde, wird der nicht partitionierte Index in den Tabellenbereich gestellt, der in dieser INDEX IN-Klausel angegeben ist.
  - Wenn die zu indexierende Tabelle ohne die *tabellenbereich-klauseln*-Klausel INDEX IN der Anweisung CREATE TABLE erstellt wurde, wird der nicht partitionierte Index im Tabellenbereich der ersten sichtbaren oder zugeordneten Datenpartition der Tabelle erstellt. Die erste sichtbare oder zugeordnete Datenpartition der Tabelle ist die erste Partition in der Liste der Datenpartitionen, die auf der Basis von Bereichsangaben sortiert sind. Die Berechtigungs-ID der Anweisung ist für das Zugriffsrecht USE für den Standardtabellenbereich ebenfalls nicht erforderlich.

IN *tabellenbereichsname*
Gibt den Tabellenbereich an, in dem der Index erstellt wird. Diese Klausel kann nicht für einen partitionierten Index oder einen Index einer nicht partitionierten Tabelle (SQLSTATE 42601) angegeben werden. Die Spezifikation eines

Tabellenbereichs insbesondere für den Index überschreibt eine Spezifikation, die bei der Tabellenerstellung mit der INDEX IN-Klausel erstellt wurde.

Der mit *tabellenbereichsname* angegebene Tabellenbereich muss sich in derselben Datenbankpartitionsgruppe wie die Datentabellenbereiche der Tabelle befinden und Speicherplatz auf dieselbe Art wie die anderen Tabellenbereiche der partitionierten Tabelle verwalten (SQLSTATE 42838). Es muss sich um einen Tabellenbereich handeln, für den die Berechtigungs-ID der Anweisung das Zugriffsrecht USE besitzt.

Wird die Klausel IN nicht angegeben, wird der Index in dem Tabellenbereich erstellt, der mit der Klausel INDEX IN in der Anweisung CREATE TABLE angegeben wurde. Wurde die Klausel INDEX IN nicht angegeben, wird der Tabellenbereich der ersten angezeigten oder zugeordneten Datenpartition der Tabelle verwendet. Dies ist die erste Partition in der Liste der Datenpartitionen, die auf der Basis von Bereichsangaben sortiert sind. Wird die Klausel IN nicht angegeben, braucht die Berechtigungs-ID der Anweisung nicht das Zugriffsrecht USE für den Standardtabellenbereich zu besitzen.

SPECIFICATION ONLY
Gibt an, dass mit dieser Anweisung eine Indexspezifikation erstellt wird, die für die Datenquellentabelle gilt, auf die *kurzname* verweist. Dieses Schlüsselwort muss angegeben werden, wenn *kurzname* angegeben wird (SQLSTATE 42601). Bei der Angabe von *tabellenname* kann SPECIFICATION ONLY nicht angegeben werden (SQLSTATE 42601).

Wenn die Indexspezifikation für einen eindeutigen Index gilt, überprüft DB2 nicht, ob die Spaltenwerte in der fernen Tabelle eindeutig sind. Wenn die fernen Spaltenwerte nicht eindeutig sind, geben Abfragen auf den Kurznamen, die diese Indexspalte enthalten, möglicherweise falsche Daten oder Fehler zurück.

Diese Klausel darf nicht zum Erstellen eines Index für eine erstellte temporäre Tabelle oder eine deklarierte temporäre Tabelle verwendet werden (SQLSTATE 42995).

INCLUDE
Dieses Schlüsselwort führt eine Klausel ein, die zusätzliche Spalten angibt, die an die Gruppe der Indexschlüsselspalten angehängt werden. Alle Spalten, die mit dieser Klausel in den Index eingeschlossen werden, werden nicht zur Umsetzung der Eindeutigkeit verwendet. Solche INCLUDE-Spalten können die Leistung einiger Abfragen durch die Möglichkeit eines reinen Indexzugriffs verbessern. Die Spalten müssen sich von den Spalten unterscheiden, mit denen die Eindeutigkeit umgesetzt wird (SQLSTATE 42711). Bei der Angabe von INCLUDE muss UNIQUE angegeben werden (SQLSTATE 42613). Die Begrenzungen für die Spaltenanzahl und die Summe der Längenattribute gelten für alle Spalten im eindeutigen Schlüssel und im Index.

Diese Klausel darf bei erstellten temporären Tabellen oder deklarierten temporären Tabellen nicht verwendet werden (SQLSTATE 42995).

spaltenname
Gibt eine Spalte an, die im Index enthalten ist, aber nicht zum eindeutigen Indexschlüssel gehört. Es gelten dieselben Regeln, die für Spalten des eindeutigen Indexschlüssels definiert wurden. Die Schlüsselwörter ASC oder DESC können nach dem *spaltennamen* angegeben werden, haben jedoch keinen Einfluss auf die Reihenfolge.

INCLUDE kann nicht angegeben werden, wenn Indizes mit EXTEND USING definiert werden, wenn ein *kurzname* angegeben ist oder wenn der Index in einer XML-Spalte definiert wurde (SQLSTATE 42601).

xml-indexspezifikation
Gibt an, wie Indexschlüssel aus XML-Dokumenten generiert werden, die in einer XML-Spalte gespeichert sind. *xml-indexspezifikation* kann nicht angegeben werden, wenn mehrere Indexspalten vorhanden sind oder wenn die Spalte nicht den XML-Datentyp hat.

Diese Klausel gilt nur für XML-Spalten (SQLSTATE 429BS).

GENERATE KEY USING XMLPATTERN *xml-musterklausel*
Gibt die Abschnitte eines XML-Dokuments an, die indexiert werden sollen. Bei XML-Musterwerten handelt es sich um die indexierten Werte, die mit *xml-musterklausel* generiert werden. Listendatentypknoten werden im Index nicht unterstützt. Wenn ein Knoten durch *xml-musterklausel* qualifiziert wird und ein XML-Schema vorhanden ist, das den Knoten als Listendatentyp ausweist, kann der Listendatentypknoten nicht indexiert werden (SQLSTATE 23526 bei CREATE INDEX-Anweisungen oder SQLSTATE 23525 bei INSERT- und UPDATE-Anweisungen).

xml-musterklausel
Enthält einen Musterausdruck, der die zu indexierenden Knoten angibt. Seine Bestandteile sind: *namensbereichsdeklaration* (optional) und *musterausdruck* (erforderlich).

namensbereichsdeklaration
Wenn im Musterausdruck qualifizierte Namen enthalten sind, muss *namensbereichsdeklaration* angegeben werden, damit Namensbereichspräfixe definiert werden können. Bei nicht qualifizierten Namen kann ein Standardnamensbereich definiert werden.

DECLARE NAMESPACE *namensbereichspräfix=namensbereichs-uri*
Ordnet *namensbereichspräfix*, einen Namen ohne Doppelpunkte (NCName), dem Zeichenfolgeliteral *namensbereichs-uri* zu. *namensbereichsdeklaration* kann mehrere Zuordnungen von *namensbereichspräfix* zu *namensbereichs-uri* enthalten. *namensbereichspräfix* muss in der Liste der Vorkommen von *namensbereichsdeklaration* eindeutig sein (SQLSTATE 10503).

DECLARE DEFAULT ELEMENT NAMESPACE *namensbereichs-uri*
Deklariert die Standardnamensbereichs-URI von nicht qualifizierten Elementnamen oder Typen. Wird kein Standardnamensbereich deklariert, werden nicht qualifizierte Elementnamen und Typen keinem Namensbereich zugeordnet. Es kann nur ein einziger Standardnamensbereich deklariert werden (SQLSTATE 10502).

musterausdruck
Gibt die Knoten in einem XML-Dokument an, die indexiert werden. *musterausdruck* kann Platzhalterzeichen (*) enthalten. Der Begriff ähnelt einem Pfadausdruck in XQuery, unterstützt jedoch eine Untermenge der XQuery-Sprache, die von DB2 unterstützt wird.

/ (Schrägstrich)
Trennt Schrägstriche innerhalb eines Pfadausdrucks.

// (doppelter Schrägstrich)
Die abgekürzte Syntax für '/descendant-or-self::node()/'. Sie

können *// (doppelter Schrägstrich)* nicht verwenden, wenn UNIQUE ebenfalls angegeben wird.

vorwärtsachse

child::
Gibt die (direkt) untergeordneten Elemente des Kontextknotens an. Dies ist die Standardeinstellung, sofern keine andere Vorwärtsachse angegeben wird.

@ Gibt die Attribute des Kontextknotens an. Die abgekürzte Syntax für 'attribute::'.

attribute::
Gibt die Attribute des Kontextknotens an.

descendant::
Gibt die Nachkommen des Kontextknotens an. Sie können *descendant::* nicht verwenden, wenn UNIQUE ebenfalls angegeben wird.

self::
Gibt den Kontextknoten selbst an.

descendant-or-self::
Gibt den Kontextknoten selbst und die Nachkommen des Kontextknotens an. Sie können 'descendant-or-self::' nicht verwenden, wenn UNIQUE ebenfalls angegeben wird.

xml-namenstest
Gibt mit einem qualifizierten XML-Namen (xml-qname) oder mit einem Platzhalter (xml-platzhalter) den Knotennamen für den Schritt im Pfad an.

xml-ncname
Ein von XML 1.0 definierter Name. Er darf keinen Doppelpunkt enthalten.

xml-qname
Gibt einen qualifizierten XML-Namen (auch als QName bezeichnet) an, der zwei Formate haben kann:

- xml-nspräfix:xml-ncname; dabei ist xml-nspräfix ein Wert für xml-ncname, der einen gültigen Namensbereich angibt
- xml-ncname; gibt an, dass der Standardnamensbereich nicht als impliziter Wert für xml-nspräfix angewendet werden soll

xml-platzhalter
Gibt einen Wert für xml-qname als Platzhalter an, der drei Formate haben kann:

- * (einfacher Stern); gibt einen beliebigen Wert für xml-qname an
- *xml-nspräfix*:*; gibt einen beliebigen Wert für xml-ncname innerhalb des angegebenen Namensbereichs an
- *:*xml-ncname*; gibt einen bestimmten XML-Namen in einem beliebigen gültigen Namensbereich an

Sie können *xml-platzhalter* nicht im Kontextschritt des Musterausdrucks verwenden, wenn UNIQUE ebenfalls angegeben wird.

xml-sortentest
Mit diesen Optionen geben Sie an, welchen Knotentypen Ihr Muster entspricht. Die folgenden Optionen sind verfügbar:

node()
Entspricht allen Knoten. Sie können *node()* nicht verwenden, wenn UNIQUE ebenfalls angegeben wird.

text()
Entspricht allen Textknoten.

comment()
Entspricht allen Kommentarknoten.

processing-instruction()
Entspricht allen Verarbeitungsanweisungsknoten. Sie können 'processing-instruction()' nicht verwenden, wenn UNIQUE ebenfalls angegeben wird.

funktionsschritt
Mit diesen Funktionsaufrufen können Sie Indizes mit besonderen Eigenschaften wie beispielsweise die Nichtbeachtung der Groß- und Kleinschreibung angeben. Für jede Klausel vom Typ XMLPATTERN ist nur ein einziger Funktionsschritt zulässig. Funktionsschritte können nur auf Elemente oder Attribute angewendet werden. Keine *xml-sortentest*-Option kann unmittelbar vor dem Funktionsschritt platziert werden. Die Funktion kann nicht in der Mitte von XMLPATTERN verwendet werden und muss nur im letzten Schritt vorkommen. Momentan werden nur die Funktionen fn:upper-case und fn:exists unterstützt.

Beachten Sie, dass anstelle der Angabe des Präfixes fn: für den Funktionsnamen ein anderer gültiger Namensbereich angegeben werden kann. Sie können fn: auch ganz weglassen.

fn:upper-case
Erzwingt, dass die Indexwerte in Großbuchstaben gespeichert werden. Der erste Parameter von fn:upper-case ist obligatorisch und muss ein Kontextelementausdruck sein (' . '); der zweite Parameter ist die Ländereinstellung und diese ist optional. Falls fn:upper-case im Muster vorkommt, sind VARCHAR und VARCHAR HASHED die beiden einzigen unterstützten Indextypen.

fn:exists
Sucht im XML-Dokument nach einem Element oder Attributelement. Falls ein Element vorhanden ist, gibt dieses Vergleichselement den Wert 'true' zurück. Der Parameter fn:exists ist obligatorisch und muss ein Element oder ein Attribut sein. Falls diese Funktion im Indexpfad verwendet wird, muss der Indextyp als VARCHAR(1) definiert sein.

xml-typklausel

AS *datentyp*
Gibt den Datentyp an, in den indexierte Werte vor ihrer Speicherung konvertiert werden. Werte werden in den XML-Indexdatentyp konvertiert, der dem angegebenen SQL-Indexdatentyp entspricht.

Tabelle 22. Entsprechende Indexdatentypen

XML-Indexdatentyp	SQL-Indexdatentyp
xs:string	VARCHAR(*integer*), VARCHAR HASHED
xs:double	DOUBLE
xs:int	INTEGER
xs:decimal	DECIMAL
xs:date	DATE
xs:dateTime	TIMESTAMP

Für die Datentypen VARCHAR(*integer*) und VARCHAR HASHED wird der Wert mit der XQuery-Funktion 'fn:string' in einen xs:string-Wert umgewandelt. Das Längenattribut von VARCHAR(*integer*) wird als Integritätsbedingung auf den resultierenden xs:string-Wert angewendet. Ein SQL-Indexdatentyp VARCHAR HASHED wendet einen Hashalgorithmus auf den resultierenden xs:string-Wert an, um einen Hash-Code zu generieren, der in den Index eingefügt wird.

Für Indizes mit den Datentypen DOUBLE, DATE, INTEGER, DECIMAL und TIMESTAMP wird der Wert mithilfe des XQuery-Umsetzungsausdruck in den XML-Indexdatentyp konvertiert.

Bei einem eindeutigen Index wird die Eindeutigkeit des Wertes nach dessen Konvertierung in den indexierten Typ umgesetzt.

datentyp
Der folgende Datentyp wird unterstützt:

sql-datentyp
Unterstützte SQL-Datentypen:

VARCHAR(*integer***)**
Wenn VARCHAR in diesem Format angegeben wird, verwendet DB2 *integer* als Integritätsbedingung. Wenn Dokumentknoten, die indexiert werden sollen, Werte haben, die länger als *integer* sind, werden die Dokumente nicht in die Tabelle eingefügt, wenn der Index bereits vorhanden ist. Ist der Index nicht vorhanden, wird der Index nicht erstellt. *integer* ist ein Wert zwischen 1 und einem von der Seitengröße abhängigen Maximalwert. Tabelle 23 zeigt den Maximalwert für die jeweilige Seitengröße an.

Tabelle 23. Maximale Länge der Dokumentknoten nach Seitengröße

Seitengröße	Maximale Länge des Dokumentknotens (Byte)
4 KB	817
8 KB	1841
16 KB	3889
32 KB	7985

Bei Zeichenfolgevergleichen, für die folgende Leerzeichen von Bedeutung sind, wird die XQuery-Semantik

verwendet. Diese unterscheidet sich von der SQL-Semantik, bei der folgende Leerzeichen in Vergleichen keine Bedeutung haben.

VARCHAR HASHED
Geben Sie VARCHAR HASHED an, um Zeichenfolgen mit beliebiger Länge zu verarbeiten. Die Länge einer indexierten Zeichenfolge hat keine Begrenzung. DB2 generiert einen 8 Byte großen Hash-Code über die gesamte Zeichenfolge. Indizes, die solche Hash-Codes für Zeichenfolgen enthalten, können nur für Suchfunktionen mit Gleichheitsvergleichselementen verwendet werden. Bei Zeichenfolgevergleichen mit Gleichheitsvergleichselementen, für die folgende Leerzeichen von Bedeutung sind, wird die XQuery-Semantik verwendet. Diese unterscheidet sich von der SQL-Semantik, bei der folgende Leerzeichen in Vergleichen keine Bedeutung haben. Der Hashwert für die Zeichenfolge ermöglicht bei Suchen mit Gleichheitsvergleichselementen die Verwendung der XQuery-Semantik, jedoch nicht die Verwendung der SQL-Semantik.

DOUBLE
Gibt an, dass numerische Werte mit dem Datentyp DOUBLE indexiert werden. Bei unbegrenzten Dezimaltypen und 64-Bit-Ganzzahlen kann es zu einem Verlust der Genauigkeit kommen, wenn sie mit einem Wert des Typs DOUBLE gespeichert werden. Die Werte für DOUBLE können die numerischen Sonderwerte *NaN*, *INF*, *-INF*, *+0* und *-0* mit einschließen, obwohl der SQL-Datentyp DOUBLE diese Werte selbst nicht unterstützt.

INTEGER
Gibt an, dass der Datentyp INTEGER für die XML-Indexwerte verwendet wird. Beachten Sie, dass der XML-Schemadatentyp xs:integer einen größeren Wertebereich zulässt als der SQL-Integerdatentyp. Wenn ein Wert außerhalb des gültigen Bereichs festgestellt wird, wird ein Fehler zurückgegeben. Falls ein Wert mit dem lexikalischen Format xs:double übereinstimmt, aber nicht mit dem lexikalischen Format xs:int, so wie beispielsweise bei 3.5, 3.0 oder 3E1, dann wird ebenfalls ein Fehler zurückgegeben.

DECIMAL(*integer*, *integer*)
Gibt an, dass der Datentyp DECIMAL für die Indexierung von XML-Werten verwendet wird. Der Typ DECIMAL enthält zwei Parameter: *genauigkeit* und *skala*. Der erste Parameter, *genauigkeit* ist eine ganzzahlige Konstante mit einem Wert im Bereich von 1 bis 31, die die Gesamtzahl der Ziffern angibt. Der zweite Parameter-*skala* ist eine ganzzahlige Konstante, die größer-gleich null ist und kleiner-gleich der Genauigkeit. Die Variable *skala* gibt die Anzahl der Ziffern rechts vom Maßstabsfaktor an.

Ziffern werden nicht vom Ende einer Dezimalzahl abgeschnitten. Ein Fehler wird zurückgegeben, falls die Anzahl der Ziffern rechts vom Dezimaltrennzeichen größer als die Skala ist. Ein Fehler wird ferner dann zurückgegeben, wenn die Anzahl der signifikanten Ziffern links vom Dezimalzeichen (der ganzzahlige Teil der Anzahl) größer als die Genauigkeit ist.

DATE
Gibt an, dass XML-Werte mit dem Datentyp DATE indexiert werden. Beachten Sie, dass der XML-Schemadatentyp für xs:date einen größeren Wertebereich erlaubt als der DB2 pureXML-Datentyp für xs:date, der dem SQL-Datentyp entspricht. Wenn ein Wert außerhalb des gültigen Bereichs festgestellt wird, wird ein Fehler zurückgegeben.

TIMESTAMP
Gibt an, dass XML-Werte mit dem Datentyp TIMESTAMP indexiert werden. Beachten Sie, dass der XML-Schemadatentyp für xs:dateTime einen größeren Wertebereich und eine höhere Genauigkeit von Sekundenbruchteilen erlaubt als der DB2 pureXML xs:dateTime-Datentyp, der dem SQl-Datentyp entspricht. Wenn ein Wert außerhalb des gültigen Bereichs angetroffen wird, wird ein Fehler zurückgegeben.

IGNORE INVALID VALUES
Gibt an, dass XML-Musterwerte, die ungültige lexikalische Formate für den XML-Zielindexdatentyp sind, ignoriert werden und dass die entsprechenden Werte in den gespeicherten XML-Dokumenten von der Anweisung CREATE INDEX nicht indexiert werden. Standardmäßig werden ungültige Werte ignoriert. Bei der Durchführung von Einfüge- und Aktualisierungsoperationen werden ungültige XML-Musterwerte zwar nicht indexiert, XML-Dokumente jedoch dennoch in die Tabelle eingefügt. Es wird weder ein Fehler noch eine Warnung ausgegeben, da die Angabe dieser Datentypen keine Integritätsbedingung für die XML-Musterwerte darstellt (XQuery-Ausdrücke, die nach bestimmten XML-Indexdatentypen suchen, berücksichtigen diese Werte nicht).

Die Regeln dafür, welche XML-Musterwerte ignoriert werden können, werden vom angegebenen SQL-Datentyp festgelegt.

- Wenn es sich beim SQL-Datentyp um VARCHAR(*integer*) oder VARCHAR HASHED handelt, werden XML-Musterwerte in keinem Fall ignoriert, da alle Zeichenfolgen gültig sind.
- Wenn es sich bei dem SQL-Datentyp um DOUBLE, DECIMAL oder INTEGER handelt, wird jeder XML-Musterwert, der nicht dem lexikalischen Format des XML-Datentyps xs:double entspricht, ignoriert. Falls der SQL-Datentyp DECIMAL oder INTEGER lautet und der XML-Musterwert mit dem lexikalischen Format des XML-Datentyps xs:double übereinstimmt, aber nicht mit dem lexikalischen Format xs:decimal oder xs:int, so wird ein entsprechender Fehler zurückgegeben. Beispiel: Wenn der SQL-Datentyp INTEGER lautet, entsprechen die XML-Musterwerte 3.5, 3.0 und 3e0 dem lexikalischen Format von xs:double; es wird jedoch ein Fehler zurückgegeben (SQLSTATE 23525), weil sie nicht

dem lexikalischen Format von xs:int entsprechen. XML-Musterwerte wie 'A123' oder 'hello' werden für denselben Index ignoriert.

- Wenn es sich bei dem SQL-Datentyp um einen Datentyp für Datum und Uhrzeit handelt, wird jeder XML-Musterwert, der nicht dem lexikalischen Format des entsprechenden XML-Datentyps (xs:date oder xs:dateTime) entspricht, ignoriert.

Wenn ein XML-Musterwert dem entsprechenden lexikalischen Format entspricht, wird ein Fehler zurückgegeben, wenn der Wert außerhalb des Wertebereichs für den Datentyp liegt oder die maximale Länge oder Genauigkeit und Nachkommastelle des angegebenen SQL-Datentyps überschreitet. Ist der Index nicht vorhanden, wird der Index nicht erstellt (SQLSTATE 23526).

REJECT INVALID VALUES
Alle XML-Musterwerte müssen im Kontext der lexikalischen Definition des XML-Indexdatentyps gültig sein. Außerdem muss der Wert im Wertebereich des XML-Indexdatentyps liegen. Im Abschnitt zu den zugehörigen Referenzen, der später folgt, finden Sie Links zu Details zur lexikalischen Definition und zum Wertebereich für die einzelnen Datentypen. Beispiel: Wenn Sie die Klausel REJECT INVALID VALUES bei der Erstellung eines Index vom Typ INTEGER erstellen, geben XML-Musterwerte wie 3.5, 3.0, 3e0, 'A123' und 'hello' einen Fehler zurück (SQLSTATE 23525). XML-Daten werden nicht in die Tabelle eingefügt oder dort aktualisiert, wenn der Index bereits vorhanden ist (SQLSTATE 23525). Ist der Index nicht vorhanden, wird der Index nicht erstellt (SQLSTATE 23526).

CLUSTER
Gibt an, dass der Index der Clusterindex der Tabelle ist. Der Clusterfaktor eines Clusterindex wird beim Einfügen von Daten in die zugehörige Tabelle beibehalten oder dynamisch verbessert, indem versucht wird, neue Zeilen in der physischen Nähe der Zeilen einzufügen, bei denen sich die Schlüsselwerte dieses Index in demselben Bereich befinden. Da für eine Tabelle nur ein einziger Clusterindex zulässig ist, darf CLUSTER nicht angegeben werden, wenn diese Klausel in einem der vorhandenen Indizes für die Tabelle definiert worden war (SQLSTATE 55012). Ein Clusterindex kann nicht für eine Tabelle erstellt werden, die für die Verwendung des Anfügemodus definiert wurde (SQLSTATE 428D8).

CLUSTER ist nicht zulässig, wenn *kurzname* angegeben wird oder wenn es sich um einen Index für XML-Daten handelt (SQLSTATE 42601). Diese Klausel darf nicht bei erstellten temporären Tabellen oder deklarierten temporären Tabellen (SQLSTATE 42995) oder Bereichsclustertabellen verwendet werden (SQLSTATE 429BG).

EXTEND USING *indexerweiterungsname*
Benennt die Indexerweiterung (*indexerweiterung*), mit der dieser Index verwaltet wird. Bei Angabe dieser Klausel darf *spaltenname* nur ein einziges Mal angegeben werden, und diese Spalte muss einen strukturierten oder einzigartigen Datentyp aufweisen (SQLSTATE 42997). *indexerweiterungsname* muss eine im Katalog beschriebene Indexerweiterung benennen (SQLSTATE 42704). Bei einem einzigartigen Datentyp muss die Spalte genau mit dem Typ des entsprechenden Quellenschlüsselparameters in der Indexerweiterung übereinstimmen. Bei einer Spalte mit einem strukturierten Typ muss der Typ des entsprechen-

den Quellenschlüsselparameters mit dem Spaltentyp oder einem übergeordneten Typ des Spaltentyps übereinstimmen (SQLSTATE 428E0).

Diese Klausel darf bei erstellten temporären Tabellen oder deklarierten temporären Tabellen nicht verwendet werden (SQLSTATE 42995).

Dies Klausel kann nicht in einer DB2 pureScale-Umgebung verwendet werden (SQLSTATE 56038).

konstanter-ausdruck
Gibt Werte für alle für die Indexerweiterung erforderlichen Argumente an. Jeder Ausdruck muss ein konstanter Wert mit einem Datentyp sein, der genau mit dem definierten Datentyp der entsprechenden Indexerweiterungsparameter (Länge oder Genauigkeit und Skala) übereinstimmt (SQLSTATE 428E0). Diese Klausel darf eine Länge von 32.768 Byte in der Datenbankcodepage nicht überschreiten (SQLSTATE 22001).

PCTFREE *integer*
Gibt an, welcher Prozentsatz jeder Indexseite beim Erstellen des Index freier Speicherbereich bleiben soll. Der erste Eintrag auf einer Seite wird ohne Einschränkungen hinzugefügt. Werden weitere Einträge auf einer Indexseite eingefügt, verbleiben auf jeder Seite mindestens *integer* Prozent freier Speicherbereich. Der Wert von *integer* kann zwischen 0 und 99 liegen. Wird ein Wert größer als 10 angegeben, verbleiben auf Nichtblattseiten nur 10 Prozent freier Speicherbereich.

Falls ein expliziter Wert für PCTFREE nicht zur Verfügung gestellt wird und falls **DB2_INDEX_PCTFREE_DEFAULT** nicht definiert wurde, dann weist PCTFREE den Standardwert 10 auf.

PCTFREE ist nicht zulässig, wenn *kurzname* angegeben wird (SQLSTATE 42601). Diese Klausel darf nicht bei erstellten temporären Tabellen oder deklarierten temporären Tabellen verwendet werden (SQLSTATE 42995).

LEVEL2 PCTFREE *integer*
Gibt an, welcher Prozentsatz jeder Indexseite der Stufe 2 beim Erstellen des Index freier Speicherbereich bleiben soll. Der Wert von *integer* kann zwischen 0 und 99 liegen. Wird LEVEL2 PCTFREE nicht festgelegt, verbleiben auf allen Nichtblattseiten mindestens 10 oder PCTFREE Prozent freier Speicherbereich. Wird LEVEL2 PCTFREE festgelegt, verbleiben auf temporären Seiten der Stufe 2 *integer* Prozent freier Speicherbereich und auf temporären Seiten der Stufe 3 und höher mindestens 10 oder *integer* Prozent freier Speicherbereich.

LEVEL2 PCTFREE ist nicht zulässig, wenn *kurzname* angegeben wird (SQLSTATE 42601). Diese Klausel darf nicht bei erstellten temporären Tabellen oder deklarierten temporären Tabellen verwendet werden (SQLSTATE 42995).

MINPCTUSED *integer*
Gibt an, ob Indexseiten online zusammengeführt werden und welcher Schwellenwert für den minimalen Prozentsatz an Speicherplatz auf einer Indexseite genutzt wird. Wenn ein Schlüssel von einer Indexseite entfernt wird und der Prozentsatz an Speicherplatz auf der Seite bei oder unter *integer* Prozent liegt, wird versucht, die verbleibenden Schlüssel auf dieser Seite mit denen einer benachbarten Seite zusammenzuführen. Wenn genügend Platz auf einer dieser Seiten vorhanden ist, wird die Zusammenführung ausgeführt und eine dieser Seiten gelöscht. Der Wert von *integer* kann zwischen 0 und 99 liegen. Unter Leistungsaspekten wird ein Wert von 50 oder weniger empfohlen. Die Angabe dieser Option wirkt sich auf die Leistung beim Aktualisieren und Löschen aus. Die Zusammenführung erfolgt bei Aktualisierungs- und Löschoperationen nur, wenn eine ausschließliche Tabellensperre vorhanden ist. Ist keine ausschließli-

che Tabellensperre vorhanden, werden die Schlüssel bei Aktualisierungs- und Löschoperationen als pseudogelöscht markiert und es findet keine Zusammenführung statt. Ziehen Sie die Verwendung der Option CLEANUP ONLY ALL des Befehls REORG INDEXES zum Zusammenführen von Blattseiten in Betracht, anstatt dazu die Option MINPCTUSED des Befehls CREATE INDEX zu verwenden.

MINPCTUSED ist nicht zulässig, wenn *kurzname* angegeben wird (SQLSTATE 42601). Diese Klausel darf nicht bei erstellten temporären Tabellen oder deklarierten temporären Tabellen verwendet werden (SQLSTATE 42995).

DISALLOW REVERSE SCANS
Gibt an, dass ein Index nur die Vorwärtssuche oder das Durchsuchen des Index in der Reihenfolge unterstützt, die bei der Erstellung des Index definiert wurde.

DISALLOW REVERSE SCANS kann nicht zusammen mit *kurzname* angegeben werden (SQLSTATE 42601).

ALLOW REVERSE SCANS
Gibt an, dass ein Index sowohl die Vorwärts- als auch die Rückwärtssuche unterstützt, d. h. das Durchsuchen des Index in der bei dessen Erstellung definierten Reihenfolge und in der umgekehrten Reihenfolge.

ALLOW REVERSE SCANS kann nicht zusammen mit *kurzname* angegeben werden (SQLSTATE 42601).

PAGE SPLIT
Gibt das Verhalten bei einer Indexteilung an. Der Standardwert ist SYMMETRIC.

SYMMETRIC
Gibt an, dass die Seiten ungefähr in der Mitte geteilt werden sollen.

HIGH
Gibt ein Verhalten bei der Teilung von Indexseiten an, bei dem der Speicherbereich auf Indexseiten effizienter genutzt wird, wenn die Werte der eingefügten Indexschlüssel einem bestimmten Muster folgen. Bei einer Untermenge von Indexschlüsseln müssen die Spalte(n) ganz links im Index denselben Wert und die Spalte(n) ganz rechts im Index Werte enthalten, die sich bei jeder Einfügung erhöhen.

LOW
Gibt ein Verhalten bei der Teilung von Indexseiten an, bei dem der Speicherbereich auf Indexseiten effizienter genutzt wird, wenn die Werte der eingefügten Indexschlüssel einem bestimmten Muster folgen. Bei einer Untermenge von Indexschlüsseln müssen die Spalte(n) ganz links im Index denselben Wert und die Spalte(n) ganz rechts im Index Werte enthalten, die sich bei jeder Einfügung verringern.

COLLECT STATISTICS
Gibt an, dass bei der Indexerstellung grundlegende Statistikdaten zu dem Index erfasst werden sollen.

SAMPLED
Gibt an, dass ein Stichprobenverfahren verwendet werden soll, wenn die Indexeinträge verarbeitet werden, um erweiterte Indexstatistiken zu erfassen. Diese Option wird verwendet, um die Leistungsaspekte mit den Anforderungen nach Genauigkeit der Statistikdaten in Einklang zu bringen. Diese Option ist der Standard, wenn DETAILED unmittelbar nach dem Schlüsselwort COLLECT angegeben wurde.

UNSAMPLED
Gibt an, dass die Stichprobenentnahme nicht bei der Verarbeitung von Indexeinträgen verwendet wird, um erweiterte Indexstatistikdaten zu erfassen. Stattdessen wird jeder Indexeintrag einzeln untersucht. Diese Option kann die CPU-Auslastung und den Speicherbedarf deutlich ansteigen lassen.

DETAILED
Gibt an, dass bei der Indexerstellung auch erweiterte Statistikdaten (CLUSTERFACTOR und PAGE_FETCH_PAIRS) zu dem Index erfasst werden sollen.

COMPRESS
Gibt an, ob die Indexkomprimierung aktiviert ist. Die Indexkomprimierung wird standardmäßig aktiviert, wenn die Datenzeilenkomprimierung aktiviert ist; Ist die Datenzeilenkomprimierung inaktiviert, ist auch die Indexkomprimierung inaktiviert. Diese Option kann zum Überschreiben des Standardverhaltens verwendet werden. COMPRESS ist nicht zulässig, wenn *kurzname* angegeben wird (SQLSTATE 42601).

YES
Gibt an, dass die Indexkomprimierung aktiviert ist. Einfügungs- und Aktualisierungsoperationen beim Index unterliegen der Komprimierung.

NO Gibt an, dass die Indexkomprimierung inaktiviert ist.

Regeln

- Bei dem Versuch, einen Index zu erstellen, der mit einem vorhandenen Index übereinstimmt, schlägt die Anweisung CREATE INDEX fehl (SQLSTATE 01550)

  Um zu ermitteln, ob zwei Indizes miteinander übereinstimmen, werden mehrere Faktoren verwendet. Diese Faktoren werden auf unterschiedliche Arten in den Regeln kombiniert, anhand derer ermittelt wird, ob zwei Indizes miteinander übereinstimmen. Die folgenden Faktoren werden verwendet um zu ermitteln, ob zwei Indizes miteinander übereinstimmen:

  1. Die Gruppen von Indexspalten, einschließlich aller INCLUDE-Spalten, sind in beiden Indizes identisch.
  2. Die Sortierung von Indexschlüsselspalten, einschließlich aller INCLUDE-Spalten, ist in beiden Indizes identisch.
  3. Die Schlüsselspalten des neuen Index sind gleich oder stellen eine Obermenge der Schlüsselspalten im vorhandenen Index dar.
  4. Die Sortierattribute der Spalten sind in beiden Indizes identisch.
  5. Der vorhandene Index ist eindeutig.
  6. Beide Indizes sind nicht eindeutig.

  Die folgenden Kombinationen dieser Faktoren bilden die Regeln, anhand derer ermittelt wird, ob zwei Indizes als Duplikate betrachtet werden:

  – 1 + 2 + 4 + 5
  – 1 + 2 + 4 + 6
  – 1 + 2 + 3 + 5

  Ausnahmen:

  – Wenn es sich bei einem der verglichenen Indizes um einen partitionierten und beim anderen um einen nicht partitionierten Index handelt, gelten die Indizes nicht als Duplikate, wenn sie unterschiedliche Namen aufweisen. Dies gilt auch dann, wenn andere Übereinstimmungsbedingungen für die Indizes erfüllt sind.

  - Bei Indizes zu XML-Daten gelten die Indexbeschreibungen nicht als Duplikate, wenn sich die Indexnamen unterscheiden, selbst wenn die indexierte XML-Spalte, die XML-Muster und der Datentyp einschließlich seiner Optionen identisch sind.
- Eindeutige Indizes für gespeicherte Abfragetabellen (MQTs) werden nicht unterstützt (SQLSTATE 42809).
- Die Optionen von COLLECT STATISTICS werden bei der Angabe eines Kurznamens nicht unterstützt (SQLSTATE 42601).

Hinweise

- Gleichzeitiger Lese-/Schreibzugriff während der Indexerstellung und das Verhalten bei der Standardindexerstellung unterscheiden sich für Indizes bei nicht partitionierten Tabellen, nicht partitionierten Indizes, partitionierten Indizes und Indizes in einer DB2 pureScale-Umgebung:
  - Für nicht partitionierte Indizes ist während der Erstellung eines Index der gleichzeitige Lese-/Schreibzugriff auf die Tabelle zulässig, sofern nicht die EXTEND USING-Klausel angegeben ist. Sobald ein Build des Index erstellt wurde, werden Änderungen, die während der Indexerstellung vorgenommen wurden, in den neuen Index übernommen. Während der Beendigung der Indexerstellung wird der Schreibzugriff kurzzeitig blockiert, und danach steht der neue Index zur Verfügung.
  - Für partitionierte Indizes ist während der Erstellung eines Index der gleichzeitige Lese-/Schreibzugriff auf die Tabelle zulässig, sofern nicht die EXTEND USING-Klausel angegeben ist. Sobald ein Build des Index erstellt wurde, werden Änderungen, die während der Erstellung dieser Indexpartition an der Partition vorgenommen wurden, in die neue Indexpartition übernommen. Während der Beendigung der Indexerstellung für die verbleibenden Datenpartitionen wird der Schreibzugriff auf die Datenpartition blockiert. Sobald ein Build der Indexpartition für die letzte Datenpartition erstellt und die Transaktion festgeschrieben ist, stehen alle Datenpartitionen für Lese- und Schreibvorgänge zur Verfügung.
  - In einer DB2 pureScale-Umgebung ist der gleichzeitige Lesezugriff das Standardverhalten. Der gleichzeitige Schreibzugriff ist während der Indexerstellung nicht zulässig.

  Sie können dieses Standardverhalten umgehen, indem Sie die Tabelle mit der Anweisung LOCK TABLE explizit sperren, bevor Sie eine Anweisung CREATE INDEX absetzen. (Je nachdem, ob Lesezugriff zulässig sein soll, kann die Tabelle im Modus SHARE oder EXCLUSIVE gesperrt werden.)
- Wenn die benannte Tabelle bereits Daten enthält, werden mit CREATE INDEX die zugehörigen Indexeinträge erstellt. Enthält die Tabelle noch keine Daten, erstellt CREATE INDEX eine Beschreibung des Index. Die Indexeinträge werden erstellt, wenn Daten in die Tabelle eingefügt werden.
- b
- Nachdem der Index erstellt und Daten in die Tabelle geladen wurden, empfiehlt es sich, den Befehl RUNSTATS abzusetzen. Der Befehl RUNSTATS aktualisiert die zu den Datenbanktabellen, -spalten und -indizes erfassten Statistikdaten. Mithilfe dieser Statistikdaten wird der optimale Pfad für den Zugriff auf die Tabelle ermittelt. Durch Absetzen des Befehls RUNSTATS kann der Datenbankmanager die Merkmale des neuen Index ermitteln. Wenn vor dem Absetzen der Anweisung CREATE INDEX Daten geladen wurden, empfiehlt es sich, als Alternative zu dem Befehl RUNSTATS die Option COLLECT STATISTICS für die Anweisung CREATE INDEX zu verwenden.

- Die Erstellung eines Index mit einem noch nicht vorhandenen Schemanamen bewirkt die implizite Erstellung dieses Schemas, sofern die Berechtigungs-ID der Anweisung die Berechtigung IMPLICIT_SCHEMA hat. Der Eigner des Schemas ist SYSIBM. Das Zugriffsrecht CREATEIN für das Schema wird PUBLIC zugeteilt.
- Das Optimierungsprogramm kann vor der Erstellung des tatsächlichen Index bestimmte Indizes empfehlen.
- Wenn eine Indexspezifikation für eine Datenquellentabelle definiert wird, die einen Index besitzt, muss der Name der Indexspezifikation nicht mit dem Namen des Index übereinstimmen.
- Das Optimierungsprogramm nutzt Indexspezifikationen zur Verbesserung des Zugriffs auf die Datenquellentabellen, für die diese Spezifikationen gelten.
- ***Indexstatistikdaten erfassen***: Die Option UNSAMPLED DETAILED ist verfügbar, um die Art und Weise zu ändern, mit der Indexstatistikdaten erfasst werden. Sie sollte jedoch nur verwendet werden, wenn deutlich ist, dass mit DETAILED keine genauen Statistikdaten erfasst werden konnten.
- ***Syntaxalternativen***: Die folgende Syntax wird toleriert und ignoriert:
  - CLOSE
  - DEFINE
  - FREEPAGE
  - GBPCACHE
  - PIECESIZE
  - TYPE 2
  - USING-Block

  Die folgende Syntax wird als Standardverhalten akzeptiert:
  - COPY NO
  - DEFER NO

Beispiele

- *Beispiel 1:* Erstellen Sie für die Tabelle PROJECT einen Index mit dem Namen UNIQUE_NAM. Mit dem Index soll sichergestellt werden, dass in der Tabelle nicht mehrere Einträge mit demselben Wert für den Projektnamen (PROJNAME) vorhanden sind. Die Indexeinträge sollen in aufsteigender Reihenfolge sortiert sein.

```
CREATE UNIQUE INDEX UNIQUE_NAM
   ON PROJECT(PROJNAME)
```

- *Beispiel 2:* Erstellen Sie für die Tabelle EMPLOYEE einen Index mit dem Namen JOB_BY_DPT. Ordnen Sie die Indexeinträge in jeder Abteilung (WORKDEPT) in aufsteigender Reihenfolge nach der Jobbezeichnung (JOB).

```
CREATE INDEX JOB_BY_DPT
   ON EMPLOYEE (WORKDEPT, JOB)
```

- *Beispiel 3:* Der Kurzname EMPLOYEE verweist auf eine Datenquellentabelle mit dem Namen CURRENT_EMP. Nach der Erstellung dieses Kurznamens wurde für CURRENT_EMP ein Index definiert. Als Indexschlüssel wurden die Spalten WORKDEBT und JOB gewählt. Erstellen Sie eine Indexspezifikation, die diesen Index beschreibt. Dank dieser Spezifikation weiß das Optimierungsprogramm, dass der Index vorhanden ist und welchen Schlüssel er hat. Auf der Basis dieser Informationen kann das Optimierungsprogramm seine Strategie für den Zugriff auf die Tabelle verbessern.

```
CREATE UNIQUE INDEX JOB_BY_DEPT
  ON EMPLOYEE (WORKDEPT, JOB)
  SPECIFICATION ONLY
```

- *Beispiel 4:* Erstellen Sie einen erweiterten Indextyp mit dem Namen SPATIAL_INDEX für eine Spaltenposition mit einem strukturierten Typ. Mithilfe der Beschreibung in der Indexerweiterung GRID_EXTENSION wird SPATIAL_INDEX verwaltet. Das Literal wird GRID_EXTENSION zur Erstellung der Indexgittergröße gegeben.

  ```
  CREATE INDEX SPATIAL_INDEX ON CUSTOMER (LOCATION)
    EXTEND USING (GRID_EXTENSION (x'000100100010001000400010'))
  ```

- *Beispiel 5:* Erstellen Sie einen Index mit dem Namen IDX1 für eine Tabelle namens TAB1, und erfassen Sie grundlegende Statistikdaten zum Index IDX1.

  ```
  CREATE INDEX IDX1 ON TAB1 (col1) COLLECT STATISTICS
  ```

- *Beispiel 6:* Erstellen Sie einen Index mit dem Namen IDX2 für eine Tabelle namens TAB1, und erfassen Sie detaillierte Statistikdaten zum Index IDX2.

  ```
  CREATE INDEX IDX2 ON TAB1 (col2) COLLECT DETAILED STATISTICS
  ```

- *Beispiel 7:* Erstellen Sie einen Index mit dem Namen IDX3 für eine Tabelle namens TAB1, und erfassen Sie durch die Entnahme von Stichproben detaillierte Statistikdaten zum Index IDX3.

  ```
  CREATE INDEX IDX3 ON TAB1 (col3) COLLECT SAMPLED DETAILED STATISTICS
  ```

- *Beispiel 8:* Erstellen Sie im Tabellenbereich IDX_TBSP einen eindeutigen Index mit dem Namen A_IDX für eine partitionierte Tabelle namens MYNUMBERDATA.

  ```
  CREATE UNIQUE INDEX A_IDX ON MYNUMBERDATA (A) IN IDX_TBSP
  ```

- *Beispiel 9:* Erstellen Sie im Tabellenbereich IDX_TBSP einen nicht eindeutigen Index mit dem Namen B_IDX für eine partitionierte Tabelle namens MYNUMBERDATA.

  ```
  CREATE INDEX B_IDX ON MYNUMBERDATA (B)
    NOT PARTITIONED IN IDX_TBSP
  ```

- *Beispiel 10:* Erstellen Sie einen Index für XML-Daten für eine Tabelle namens COMPANYINFO, der eine XML-Spalte mit dem Namen COMPANYDOCS enthält. Die XML-Spalte COMPANYDOCS enthält eine große Anzahl an XML-Dokumenten, die dem nachstehenden Dokument ähneln:

  ```
  <company name="Company1">
    <emp id="31201" salary="60000" gender="Female">
      <name>
        <first>Laura</first>
        <last>Brown</last>
      </name>
      <dept id="M25">
        Finance
      </dept>
    </emp>
  </company>
  ```

  Benutzer der Tabelle COMPANYINFO müssen häufig Mitarbeiterinformationen mithilfe der Mitarbeiter-ID ('emp id') abrufen. Ein Index wie der folgende kann die Effizienz des Abrufs erhöhen.

  ```
  CREATE INDEX EMPINDEX ON COMPANYINFO(COMPANYDOCS)
    GENERATE KEY USING XMLPATTERN '/company/emp/@id'
      AS SQL DOUBLE
  ```

- *Beispiel 11:* Das folgende Beispiel ist logisch äquivalent zu dem im vorigen Beispiel erstellten Index, außer dass es eine nicht abgekürzte Syntax verwendet.

  ```
  CREATE INDEX EMPINDEX ON COMPANYINFO(COMPANYDOCS)
    GENERATE KEY USING XMLPATTERN '/child::company/child::emp/attribute::id'
      AS SQL DOUBLE
  ```

- *Beispiel 12:* Erstellen Sie einen Index für eine Spalte namens DOC, und indexieren Sie dabei nur den Buchtitel als VARCHAR(100). Da der Buchtitel in allen Büchern eindeutig sein sollte, muss der Index eindeutig sein.

```
CREATE UNIQUE INDEX MYDOCSIDX ON MYDOCS(DOC)
  GENERATE KEY USING XMLPATTERN '/book/title'
    AS SQL VARCHAR(100)
```

- *Beispiel 13:* Erstellen Sie einen Index für eine Spalte namens DOC, und indexieren Sie dabei die Kapitelnummer als DOUBLE. Diese Beispiel umfasst Namensbereichsdeklarationen.

```
CREATE INDEX MYDOCSIDX ON MYDOCS(DOC)
  GENERATE KEY USING XMLPATTERN
    'declare namespace b="http://www.example.com/book/";
     declare namespace c="http://acme.org/chapters";
       /b:book/c:chapter/@number'
    AS SQL DOUBLE
```

- *Beispiel 14:* Erstellen Sie einen eindeutigen Index mit dem Namen IDXPROJEST in der Tabelle PROJECT und beziehen Sie die Spalte PRSTAFF ein, damit allein der Index Zugriff auf die geschätzte durchschnittliche Mitarbeiterinformation erhält.

```
CREATE UNIQUE INDEX IDXPROJEST ON PROJECT (PROJNO) INCLUDE (PRSTAFF)
```

Beispielabfragen auf Indizes für XML-Daten

Indizes zu XML-Daten müssen mit den Abfragen abgeglichen werden, die diese Indizes verwenden sollen. Die folgenden Beispiele enthalten Abfragen, die Indizes zu XML-Daten verwenden bzw. nicht verwenden können.

Beispiele für Abfragen, die einen Index für XML-Daten verwenden können

Abfragen mit einer Vielzahl verschiedener Vergleichselemente können einen Index für XML-Daten nutzen. In diesem Abschnitt werden einige Beispiele von XQuery-Vergleichselementen gezeigt, die entsprechende Indizes verwenden können. Im Anschluss an die Abfragen sind die jeweils nutzbaren Indizes angegeben.

Beispiel 1. Ausführen einer Abfrage mit Gleichheitsvergleichselement: Ermittlung des Mitarbeiters mit der ID *42366*:

```
XQUERY for $i in db2-fn:xmlcolumn('COMPANY.COMPANYDOCS')/company/emp[@id='42366']
          return $i

     CREATE INDEX empindex on company(companydocs)
            GENERATE KEY USING XMLPATTERN '/company/emp/@id' AS SQL VARCHAR(5)
```

Beispiel 2. Abfrage mit einem Bereichsvergleichselement: Ermittlung der Mitarbeiter mit einem Gehalt über *35000*:

```
XQUERY
   for $i in db2-fn:xmlcolumn('COMPANY.COMPANYDOCS')/company/emp[@salary > 35000]
      return $i

     CREATE INDEX empindex on company(companydocs)
      GENERATE KEY USING XMLPATTERN '//@salary' AS SQL DECIMAL(10,2)
```

Beispiel 3. Ausführen einer Abfrage mit Disjunktion (OR): Ermittlung der Mitarbeiter, die zur Abteilung 'Finance' oder 'Marketing' gehören:

```
XQUERY
   for $i in
   db2-fn:xmlcolumn('COMPANY.COMPANYDOCS')/company/emp[dept/text()='Finance'
   or dept/text()='Marketing']
      return $i

     CREATE INDEX empindex on company(companydocs)
            GENERATE KEY USING XMLPATTERN '/company/emp/dept/text()' AS SQL
            VARCHAR(30)
```

Beispiel 4. Verschiedene Abfragen können durch den gleichen Index erfüllt werden:

Ermittlung des Mitarbeiters mit der ID *31201*:

```
XQUERY for $i in db2-fn:xmlcolumn('COMPANY.COMPANYDOCS')/company/emp[@id='31201']
          return $i
```

Ermittlung der Abteilungen mit der ID *K55*

```
XQUERY for $i in db2-fn:xmlcolumn('COMPANY.COMPANYDOCS')/company/emp/dept[@id='K55']
          return $i
```

```
     CREATE INDEX empindex on company(companydocs)
            GENERATE KEY USING XMLPATTERN '//@id' AS SQL VARCHAR(25)
```

Beispiel 5. Abfragevergleichselemente können Pfade enthalten: Ermittlung der Mitarbeiter mit dem Nachnamen *Murphy*, die zur Abteilung 'Sales' gehören:

```
XQUERY
   for $i in db2-fn:xmlcolumn('COMPANY.COMPANYDOCS')/company/emp[name/last='Murphy'
     and dept/text()='Sales']
      return $i
```

```
     CREATE INDEX empindex on company(companydocs)
      GENERATE KEY USING XMLPATTERN '/company/emp/name/last' AS SQL
            VARCHAR(100)
```

```
CREATE INDEX deptindex on company(companydocs)
            GENERATE KEY USING XMLPATTERN '/company/emp/dept/text()' AS SQL
            VARCHAR(30)
```

Beispiel 6. Anwendung einer hierarchischen Eingrenzung in Abfragen: Eine Abfrage kann Indizes nutzen, um eine logische Verknüpfung über AND auf verschiedenen Ebenen der Dokumenthierarchie auszuführen. Eine Abfrage kann die Indizes außerdem zur Bestimmung nutzen, welche untergeordneten Knoten zum gleichen Vorfahren (übergeordneten Knoten) gehören, um eine entsprechende Filterung zu realisieren.

Ermittlung von Unternehmen mit Mitarbeitern, deren Gehalt gleich *60000* ist, und Ermittlung von Unternehmen mit weiblichen Mitarbeitern. In den XML-Beispielfragmenten des Abschnitts 'Indexieren von XML-Daten - Übersicht' würden sowohl Company1 als auch Company2 den Kriterien entsprechen.

```
XQUERY for $i in
   db2-fn:xmlcolumn('COMPANY.COMPANYDOCS')/company[emp/@salary=60000 and
   emp/@gender='Female']
      return $i
```

Ermittlung der Mitarbeiter, die ein Gehalt von *60000* haben und weiblich sind. Diesen Kriterien würde nur Laura Brown von Company1 entsprechen.

```
XQUERY for $i in
   db2-fn:xmlcolumn('COMPANY.COMPANYDOCS')/company/emp[@salary=60000
   and @gender='Female']
      return $i
```

```
     CREATE INDEX empindex on company(companydocs)
      GENERATE KEY USING XMLPATTERN '/company/emp/@salary' AS DECIMAL(10,2)
```

```
CREATE INDEX genderindex on company(companydocs)
      GENERATE KEY USING XMLPATTERN '/company/emp/@gender' AS SQL
            VARCHAR(10)
```

Beispiel 7. Eine Abfrage kann die 'descendant-or-self'-Achse (//) verwenden und Indizes nutzen, vorausgesetzt, dass Abfragevergleichselement ist mindestens ebenso restriktiv oder restriktiver als das Indexmuster.

Ermittlung von Mitarbeitern mit der Abteilungs-ID *K55*:

```
XQUERY
   for $i in
   db2-fn:xmlcolumn('COMPANY.COMPANYDOCS')/company//emp[.//dept//@id='K55' ]
   return $i
```

```
  CREATE INDEX empindex on company(companydocs)
         GENERATE KEY USING XMLPATTERN '//emp//@id' AS SQL VARCHAR(25)
```

Beispiel 8. Bei Tabellen mit mehrdimensionalem Clustering (MDC-Tabellen) kann eine Abfrage MDC-Blockindizes und Indizes zu XML-Daten verwenden. Angenommen, eine MDC-Tabelle mit einer XML-Spalte wird erstellt und verwendet WORKDEPT als Dimension.

```
CREATE TABLE employee (empno char(6), workdept char(3), doc xml) ORGANIZE BY (workdept)
```

Angenommen, mit der folgenden Anweisung vom Typ CREATE INDEX wurde ein Index zu XML-Daten für die Spalte DOC definiert:

```
CREATE INDEX hdate on employee(doc)
      GENERATE KEY USING XMLPATTERN '//hirdate'AS SQL DATE COLLECT STATISTICS
```

Die folgende Abfrage kann in ihrem Zugriffsplan den Blockindex für WORKDEPT und den Index zu XML-Daten (hdate) für DOC verwenden:

```
SELECT COUNT (*) FROM y.employee y
      WHERE workdept='A00'
      AND XMLEXISTS('$p/employee[hiredata > "1964-01-01"]
      PASSING y.doc as "p")
```

Beispiele für Abfragen, die keinen Index für XML-Daten verwenden können

Es gibt einige Bedingungen, unter denen eine Abfrage keinen Index für XML-Daten verwenden kann. Einige Beispiele für XQuery-Vergleichselemente, die die für sie vorgesehenen Indizes nicht verwenden können, sind in diesem Abschnitt aufgeführt.

Beispiel 1. Der von der Abfrage angeforderte Datentyp muss dem indexierten Datentyp entsprechen, bevor die Abfrage den Index nutzen kann. In diesem Beispiel fordert die Abfrage die Mitarbeiter-ID als Zeichenfolge an, während die ID im Index als numerischer Wert (DOUBLE) erfasst ist:

```
XQUERY for $i in db2-fn:xmlcolumn('COMPANY.COMPANYDOCS')/company/emp[@id='31664']
           return $i
```

```
     CREATE INDEX empindex on company(companydocs)
            GENERATE KEY USING XMLPATTERN '/company/emp/@id' AS SQL DOUBLE
```

Beispiel 2. Der zur Erstellung des Index verwendete XML-Musterausdruck ist möglicherweise restriktiver als das Vergleichselement der Abfrage. In diesem Beispiel kann die Abfrage den Index nicht verwenden, weil die Abfrage sowohl die Abteilungs-ID als auch die Mitarbeiter-IDs abruft, der Index jedoch nur die Mitarbeiter-IDs enthält:

```
XQUERY for $i in db2-fn:xmlcolumn('COMPANY.COMPANYDOCS')//@id
           return $i
```

```
     CREATE INDEX empindex on company(companydocs)
            GENERATE KEY USING XMLPATTERN '/company/emp/@id' AS SQL VARCHAR(5)
```

Die folgende Abfrage ruft die Mitarbeiter ab, die die Mitarbeiter-ID *31201* oder die Abteilungs-ID *K55* besitzen. Da die ID entweder eine Mitarbeiter-ID oder eine Ab-

teilungs-ID sein kann, der Index jedoch nur Abteilungs-IDs enthält, kann der Index in seiner erstellten Form nicht verwendet werden.

```
XQUERY
   for $i in
   db2-fn:xmlcolumn('COMPANY.COMPANYDOCS')//emp[.//@id='31201' or .//@id='K55']
   return $i
     CREATE INDEX empindex on company(companydocs)
     GENERATE KEY USING XMLPATTERN '//dept//@id' AS SQL VARCHAR(5)
```

Einschränkungen von Indizes zu XML-Daten

Die Indexierung zu XML-Daten unterliegt Beschränkungen bezüglich der Datentypunterstützung, der Ebenen des gemeinsamen Zugriffs, der XML-Listenelemente sowie der Indexkomprimierung.

Ebenen des gemeinsamen Zugriffs

Die Unterstützung für einige Ebenen des gemeinsamen Zugriffs ist eingeschränkt, wenn XML-Spalten und die zugehörigen Indizes verarbeitet werden. Die folgende Tabelle enthält Informationen zu den unterstützten Ebenen des gemeinsamen Zugriffs.

Tabelle 24. Unterstützte Ebenen des gemeinsamen Zugriffs bei einer nicht partitionierten Tabelle mit mindestens einer XML-Spalte

Befehl	Ebene des gemeinsamen Zugriffs	Unterstützt
REORG INDEXES ALL FOR TABLE	Befehlsklausel: `ALLOW [NO | READ | WRITE] ACCESS`	Ja.
REORG TABLE	Befehlsklausel: `ALLOW [READ | NO] ACCESS`	Ja. Möglicherweise ist ein Index zu XML-Daten vorhanden.
REORG TABLE INPLACE (mindestens ein Index zu XML-Daten in der Tabelle vorhanden)	Befehlsklausel: `ALLOW [READ | WRITE] ACCESS`	Nein.
REORG TABLE RECLAIM EXTENTS	Befehlsklausel: `ALLOW [NO | READ | WRITE] ACCESS`	Ja.

Tabelle 25. Unterstützte Ebenen des gemeinsamen Zugriffs bei einer partitionierten Tabelle mit nicht partitionierten Indizes und mindestens einer XML-Spalte

Befehl	Ebene des gemeinsamen Zugriffs	Unterstützt
REORG INDEX (für einen nicht partitionierten Index zu XML-Daten)	Befehlsklausel: `ALLOW [NO | READ | WRITE] ACCESS`	Ja.
REORG INDEXES ALL FOR TABLE CLEANUP|RECLAIM EXTENTS	Befehlsklausel: `ALLOW [NO | READ | WRITE] ACCESS`	Ja.
REORG INDEXES ALL FOR TABLE REBUILD[1]	Befehlsklausel: `ALLOW [READ | WRITE] ACCESS`	Nein.
REORG TABLE	Befehlsklausel: `ALLOW NO ACCESS` (bei einer partitionierten Tabelle wird nur der Zugriffsmodus `ALLOW NO ACCESS` unterstützt)	Ja.

Tabelle 25. Unterstützte Ebenen des gemeinsamen Zugriffs bei einer partitionierten Tabelle mit nicht partitionierten Indizes und mindestens einer XML-Spalte (Forts.)

Befehl	Ebene des gemeinsamen Zugriffs	Unterstützt
REORG TABLE INPLACE	Befehlsklausel: `ALLOW [NO | READ | WRITE] ACCESS`	Nein. Bei einer partitionierten Tabelle wird der Parameter **INPLACE** nicht unterstützt.
REORG TABLE RECLAIM EXTENTS	Befehlsklausel: `ALLOW [NO | READ | WRITE] ACCESS`	Ja.

Anmerkung:

1. Das Standardverhalten besteht in einem Rebuild des Index. Wenn die Klausel **REBUILD** nicht angegeben wird, wird der Index erneut erstellt.

Tabelle 26. Unterstützte Ebenen des gemeinsamen Zugriffs bei einer partitionierten Tabelle mit partitionierten Indizes und XML-Spalten

Befehl	Ebene des gemeinsamen Zugriffs	Unterstützt
REORG INDEXES ALL FOR TABLE	Befehlsklausel: `ALLOW NO ACCESS`	Ja.
REORG INDEXES ALL FOR TABLE CLEANUP|RECLAIM EXTENTS	Befehlsklausel: `ALLOW [READ | WRITE] ACCESS`	Ja.
REORG INDEXES ALL FOR TABLE REBUILD ON DATA PARTITION[1]	Befehlsklausel: `ALLOW [READ | WRITE] ACCESS`	Ja.
REORG INDEXES ALL FOR TABLE REBUILD[1]	Befehlsklausel: `ALLOW [READ | WRITE] ACCESS`	Nein.
REORG TABLE	Befehlsklausel: `ALLOW NO ACCESS` (bei einer Datenpartitionstabelle wird nur der Zugriffsmodus `ALLOW NO ACCESS` unterstützt)	Ja.
REORG TABLE INPLACE	Befehlsklausel: `ALLOW [NO | READ | WRITE] ACCESS`	Nein. Bei einer Datenpartitionstabelle wird der Parameter **INPLACE** nicht unterstützt.
REORG TABLE RECLAIM EXTENTS	Befehlsklausel: `ALLOW [NO | READ | WRITE] ACCESS`	Ja.

Anmerkung:

1. Das Standardverhalten besteht in einem Rebuild des Index. Wenn die Klausel **REBUILD** nicht angegeben wird, wird der Index erneut erstellt.

Informationen zu den Klauseln und Optionen finden Sie in den Abschnitten zur Anweisung **CREATE INDEX** und zum Befehl **REORG INDEX/TABLE**.

Bei MDC-Tabellen (Multidimensional Clustering, mehrdimensionales Clustering) und ITC-Tabellen (Insert Time Clustering, Clustering anhand der Einfügungszeit) wird die Reorganisation eines Onlineindex (Rebuild) mit `ALLOW WRITE ACCESS` nicht unterstützt.

XML-Listenelemente

Listendatentypknoten können nicht indexiert werden. Wenn ein Knoten durch *xmlpattern* qualifiziert wird und ein XML-Schema vorhanden ist, das den Knoten als Listendatentyp ausweist, kann der Knoten nicht indexiert werden. Die Ausführung einer Anweisung **CREATE INDEX** für einen Listendatentypknoten verursacht einen Fehler (SQLSTATE 23526, SQLCODE -20306). Die Ausführung der Anweisungen **INSERT** und **UPDATE** verursacht ebenfalls einen Fehler (SQLSTATE 23525, SQLCODE -20305).

Eindeutiger Index für XML-Daten (UNIQUE)

In einer Umgebung mit partitionierten Datenbanken gelten die folgenden Regeln für eine Tabelle mit mindestens einer XML-Spalte:

- Eine Tabelle mit einem Verteilungsschlüssel kann keinen eindeutigen Index zu XML-Daten haben.
- Ein eindeutiger Index zu XML-Daten wird nur für eine Tabelle unterstützt, die keinen Verteilungsschlüssel für eine Einzelpartitionsdatenbank aufweist.
- Falls ein eindeutiger Index zu XML-Daten für eine Tabelle vorhanden ist, kann die Tabelle nicht geändert werden, um einen Verteilungsschlüssel hinzuzufügen.

Für eine partitionierte Tabelle wird ein eindeutiger partitionierter Index zu XML-Daten nicht unterstützt. Wenn Sie versuchen, einen solchen Index zu erstellen, erhalten Sie die Fehlernachricht SQL20303N (SQLSTATE=42990).

Die Erstellung von Indizes für XML-Spalten unterliegt außerdem Einschränkungen, die für den nativen XML-Datenspeicher insgesamt gelten.

Allgemeine Aspekte der XML-Indexierung

Wenn Sie beim Indexieren von XML-Daten auf Probleme stoßen, kann dies eine der nachstehenden Ursachen haben.

Fehlerbestimmung für die Fehlernachrichten SQL20305N und SQL20306N

Diese Fehlernachrichten werden ausgegeben, wenn XML-Knotenwerte nicht indexiert werden können. Die Nachricht SQL20305N wird von den Anweisungen INSERT und UPDATE sowie von den Dienstprogrammen IMPORT und LOAD ausgegeben. Die Nachricht SQL20306N wird von der Anweisung CREATE INDEX ausgegeben, wenn sie für aufgefüllte Basistabellen abgesetzt wurde.

Die Nachrichten enthalten jeweils einen Ursachencode für den Fehler. Geben Sie `? SQL20305` oder `? SQL20306` über den Befehlszeilenprozessor ein, um die Erläuterung und Benutzeraktion für den entsprechenden Ursachencode anzuzeigen. Eine generierte XQuery-Anweisung wird als Ausgabe in die Protokolldatei **db2diag** geschrieben, um Sie beim Lokalisieren der fehlgeschlagenen XML-Knotenwerte zu unterstützen.

Wird die Nachricht SQL20305N vom Dienstprogramm LOAD ausgegeben, werden die generierten XQuery-Anweisungen zum Lokalisieren der fehlgeschlagenen Knotenwerte nicht in die Protokolldatei **db2diag** geschrieben. Um diese XQuery-Anweisungen zu generieren, muss das Dienstprogramm IMPORT für die fehlgeschlagenen, nicht geladenen Zeilen ausgeführt werden. Die Abschnitte zum Laden von XML-Daten und zum Beheben von Indexierungsfehlern beim Laden von XML-Daten im DB2-Informationszentrum enthalten zusätzliche Informationen hierzu.

Wird die Fehlernachricht SQL20305N von einer Anweisung INSERT oder UPDATE ausgegeben, lesen Sie den Abschnitt zur Fehlerbehebung bei der von den Anweisungen INSERT und UPDATE ausgegebenen Nachricht SQL20305N. Wird die Fehlernachricht SQL20306N von der Anweisung CREATE INDEX ausgegeben, lesen Sie den Abschnitt zur Fehlerbehebung bei der von der Anweisung CREATE INDEX für aufgefüllte Tabellen ausgegebenen Nachricht SQL20306N.

Fehlerbehebung bei Nachricht SQL20305N von der Anweisung INSERT oder UPDATE

Um die Ursache für eine Fehlernachricht SQL20305N zu ermitteln, lesen Sie den Abschnitt "Fehlerbestimmung für die Fehlernachrichten SQL20305N und SQL20306N" unter , und führen Sie anschließend die folgenden Schritte aus:

Vorgehensweise

1. Ermitteln Sie den Indexnamen und die Indexklausel des XML-Musters.
   a. Rufen Sie den Indexnamen (*indexname*,*indexschema*) aus SYSCAT.INDEXES ab, indem Sie die folgende Abfrage unter Verwendung der *index-id* aus der Fehlernachricht ausgeben:

      ```
      SELECT INDNAME,INDSCHEMA
         FROM SYSCAT.INDEXES
         WHERE IID = index-id AND
         TABSCHEMA ='schema' AND TABNAME ='tabellenname'
      ```

   b. Verwenden Sie *indexname* und *indexschema*, um den Indexdatentyp und das XML-Muster aus SYSCAT.INDEXES abzurufen, indem Sie die folgende Abfrage ausgeben:

      ```
      SELECT DATATYPE, PATTERN
         FROM SYSCAT.INDEXXMLPATTERNS
         WHERE INDSCHEMA = 'indexschema' AND
         INDNAME = 'indexname'
      ```

2. Um die fehlerhaften Knotenwerte im Eingabedokument zu ermitteln, suchen Sie nach der Zeichenfolge SQL20305N in der Protokolldatei **db2diag** und vergleichen Sie den Ursachencode. Im Anschluss an den Ursachencode finden Sie eine Reihe von Anweisungen und danach eine generierte XQuery-Anweisung, die Sie verwenden können, um in dem Dokument denjenigen Wert ausfindig zu machen, der den Fehler verursacht hat. Bei kleinen Knotenwerten wird der vollständige Wert im XQuery-Vergleichselement verwendet. Bei Knotenwerten, die zu lang sind, um in die Protokolldatei **db2diag** geschrieben zu werden, werden im XQuery-Vergleichselement die Startbyte des Wertes mit der Funktion 'fn:starts-with' und die Endbyte des Wertes mit der Funktion 'fn:ends-with' verwendet.
3. Da das Dokument zurückgewiesen wurde und daher in der Tabelle nicht vorhanden ist, kann die XQuery-Anweisung nicht für dieses Dokument ausgeführt werden. Um dieses Problem zu lösen, müssen Sie eine neue Tabelle mit denselben Spalten wie in der Originaltabelle erstellen und das fehlgeschlagene Dokument in die neue Tabelle einfügen. Erstellen Sie für die neue Tabelle keine Indizes.
4. Kopieren Sie die generierte XQuery-Anweisung aus der Protokolldatei **db2diag** und ersetzen Sie den Tabellennamen in der XQuery-Anweisung durch den Namen der neu erstellten Tabelle.
5. Führen Sie die XQuery-Anweisung aus, um das gesamte Dokument und das Dokumentfragment mit dem Wert, der den Fehler verursachte, abzurufen. Um Kontextinformationen zur Verfügung zu stellen, die angeben, wo im Dokument

der Fehler aufgetreten ist, gibt die XQuery-Anweisung das Dokumentfragment aus, das mit dem übergeordneten Element des Knotenwertes beginnt, der den Fehler verursacht hat.

6. Verwenden Sie das XML-Muster des Index, um die Gruppe der übereinstimmenden XML-Knoten zu identifizieren, die überprüft werden sollen. Da die generierte XQuery-Anweisung Platzhalterzeichen für Namensbereiche verwendet, ist es möglich, dass mehrere Problemwerte, die unterschiedliche Namensbereiche aufweisen, übereinstimmen. Dies kommt jedoch nicht häufig vor. Tritt dieser Umstand jedoch ein, müssen Sie die Namensbereichsdeklaration im XML-Muster des Index verwenden, um die korrekte Gruppe der übereinstimmenden XML-Knoten zu ermitteln. Wird im Vergleichselement nicht der vollständige Wert zum Filtern der Ergebnisse verwendet, muss das XML-Muster des Index verwendet werden, um die übereinstimmenden Problemwerte, die von der XQuery-Anweisung zurückgegeben werden, zu überprüfen.
7. Sobald Sie den fehlerhaften Wert im Dokument ermittelt haben, müssen Sie das Eingabedokument entsprechend ändern, um das Problem zu lösen, und die Anweisung INSERT bzw. UPDATE erneut übergeben.

Beispiel: Fehler bei Anweisung INSERT

In dem folgenden Beispiel ist der Wert `hello world` kein gültiger Wert vom Typ DOUBLE, und im generierten XQuery-Vergleichselement wird der gesamte Wert verwendet. Bitte beachten Sie, dass `*N` in der Fehlernachricht als Platzhalter verwendet wird, wenn Schemainformationen nicht anwendbar sind:

```
CREATE TABLE t1 (x XML);

CREATE INDEX ix1 ON t1(x)
   GENERATE KEY USING XMLPATTERN '/root/x/text()'
   AS SQL DOUBLE REJECT INVALID VALUES;

DB20000I  Der SQL-Befehl wurde erfolgreich ausgeführt.

INSERT INTO t1 VALUES (XMLPARSE (DOCUMENT
   'The beginning of the documenthello world'
   STRIP WHITESPACE));

DB21034E  Der Befehl wurde als SQL-Anweisung verarbeitet, da es sich um
keinen gültigen Befehl des Befehlszeilenprozessors handelte.
Während der SQL-Verarbeitung wurde Folgendes ausgegeben:

SQL20305N  Ein XML-Wert kann nicht eingefügt oder aktualisiert werden,
weil beim Einfügen in den Index bzw. beim Aktualisieren des Index,
der durch "IID = 23" in Tabelle "ADUA.T" angegeben ist, ein Fehler
festgestellt wurde.
 Ursachencode = "5".
Bei Ursachencodes für ein XML-Schema ist "*N" die XML-Schemakennung
und "*N" der XML-Schemadatentyp.
SQLSTATE=23525
```

Die Ausgabe in der Protokolldatei **db2diag** sieht wie folgt aus (mit geringfügigen Änderungen an der Formatierung):

```
2007-03-06-12.02.08.116046-480 I4436A1141         LEVEL: Warning
PID     : 1544348              TID  : 1801        PROC : db2sysc
INSTANCE: adua                 NODE : 000         DB   : ADTEST
APPHDL  : 0-18                 APPID: *LOCAL.adua.070306200203
AUTHID  : ADUA
EDUID   : 1801                 EDUNAME: db2agent (ADTEST)
FUNCTION: DB2 UDB, Xml Storage and Index Manager,
          xmlsIkaProcessErrorMsg, probe:651
MESSAGE : ZRC=0x80A50411=-2136669167=XMS_XML_IX_INSERT_UPDATE_ERROR
          "XML node value error during insert or update XML index"
```

```
DATA #1 : String, 36 bytes
SQL Code: SQL20305N ; Reason Code: 5
DATA #2 : String, 321 bytes
To locate the value in the document that caused the error, create
a new table with the same columns as the original table and insert
the failing document in the table. Do not create any indexes on
the new table. Replace the table name in the query below with the
newly created table name and execute the following XQuery.
DATA #3 : String, 187 bytes
xquery for $i in db2-fn:xmlcolumn("ADUA.T.X")[/*:root/*:x/text()="hello world"]
return
<Result>
 <ProblemDocument> {$i} </ProblemDocument>
 <ProblemValue>{$i/*:root/*:x/text()/..} </ProblemValue>
</Result>;
```

Um den fehlgeschlagenen Knotenwert zu finden, gehen Sie wie folgt vor:

1. Erstellen Sie eine neue Tabelle mit denselben Spalten wie in der Originaltabelle:
   ```
   CREATE TABLE t2 LIKE t1;
   ```
2. Fügen Sie das fehlerhafte Dokument in die neue Tabelle ein:
   ```
   INSERT INTO t2 VALUES (XMLPARSE (DOCUMENT
   'The beginning of the documenthello world'
   STRIP WHITESPACE));
   ```
3. Kopieren Sie die generierte XQuery-Anweisung aus der Protokolldatei **db2diag** und ersetzen Sie den Tabellennamen in der XQuery-Anweisung durch den Namen der neuen Tabelle:
   ```
   xquery for $i in db2-fn:xmlcolumn("ADUA.T2.X")[/*:root/*:x/text()="hello world"]
   return
   {$i}
   {$i/*:root/*:x/text()/..}
   ;
   ```
4. Führen Sie die XQuery-Anweisung für die neue Tabelle aus. Das Ergebnis der Abfrageanweisung sieht wie folgt aus (mit geringfügigen Änderungen an der Formatierung):
   ```
   <Result>
      <ProblemDocument>
         <root>The beginning of the document<x>hello world</x></root>
      </ProblemDocument>
      <ProblemValue><x>hello world</x></ProblemValue>
   </Result>
   ```

Korrigieren Sie den Fehler:

Das Dokument kann so geändert werden, dass das Element <x> einen numerischen Wert enthält, der erfolgreich in den Datentyp DOUBLE umgesetzt werden kann:

```
INSERT INTO t1 VALUES (
   XMLPARSE (DOCUMENT
   '<root>The beginning of the document<x>123</x></root>'
   STRIP WHITESPACE))
```

Fehlerbehebung bei Nachricht SQL20306N von der Anweisung CREATE INDEX für aufgefüllte Tabellen

Um die Ursache für eine Fehlernachricht SQL20306N zu ermitteln, lesen Sie den Abschnitt "Fehlerbestimmung für die Fehlernachrichten SQL20305N und SQL20306N" unter , und führen Sie anschließend die folgenden Schritte aus:

Vorgehensweise

1. Um die fehlerhaften Knotenwerte im gespeicherten Dokument zu ermitteln, suchen Sie nach der Zeichenfolge SQL20306N in der Protokolldatei **db2diag** und vergleichen Sie den Ursachencode. Im Anschluss an den Ursachencode finden Sie eine Reihe von Anweisungen und danach eine generierte XQuery-Anweisung, die Sie verwenden können, um in dem Dokument denjenigen Wert ausfindig zu machen, der den Fehler verursacht hat.
   - Bei kleinen Knotenwerten wird der vollständige Wert im XQuery-Vergleichselement verwendet.
   - Bei Knotenwerten, die zu lang sind, um in die Protokolldatei **db2diag** geschrieben zu werden, werden im XQuery-Vergleichselement die Startbyte des Wertes mit der Funktion fn:starts-with und die Endbyte des Wertes mit der Funktion fn:ends-with verwendet.
2. Führen Sie die XQuery-Anweisung aus, um das gesamte Dokument und das Dokumentfragment mit dem Wert, der den Fehler verursachte, abzurufen. Um Kontextinformationen zur Verfügung zu stellen, die angeben, wo im Dokument der Fehler aufgetreten ist, gibt die XQuery-Anweisung das Dokumentfragment aus, das mit dem übergeordneten Element des Knotenwertes beginnt, der den Fehler verursacht hat.
3. Verwenden Sie das XML-Muster des Index, um die Gruppe der übereinstimmenden XML-Knoten zu identifizieren, die überprüft werden sollen. Da die generierte XQuery-Anweisung Platzhalterzeichen für Namensbereiche verwendet, ist es möglich, dass mehrere Problemwerte, die unterschiedliche Namensbereiche aufweisen, übereinstimmen. Dies kommt jedoch nicht häufig vor. Tritt dieser Umstand jedoch ein, müssen Sie die Namensbereichsdeklaration im XML-Muster des Index verwenden, um die korrekte Gruppe der übereinstimmenden XML-Knoten zu ermitteln. Wird im Vergleichselement nicht der vollständige Wert zum Filtern der Ergebnisse verwendet, muss das XML-Muster des Index verwendet werden, um die übereinstimmenden Problemwerte, die von der XQuery-Anweisung zurückgegeben werden, zu überprüfen.
4. Sobald Sie den fehlerhaften Wert im Dokument ermittelt haben, müssen Sie das XML-Muster von CREATE INDEX entsprechend ändern, um das Problem zu lösen, oder das XQuery-Vergleichselement verwenden, um das Dokument mit dem fehlerhaften Wert zu aktualisieren oder zu löschen.

Beispiel: Fehlschlagen der Anweisung CREATE INDEX

In diesem Beispiel überschreitet der qualifizierte Textwert im gespeicherten Dokument die Länge von VARCHAR(4) im XML-Muster des Index, sodass die Anweisung CREATE INDEX fehlschlägt. Bei großen Werten verwendet die generierte XQuery-Anweisung die Funktionen fn:starts-with und fn:ends-with im Vergleichselement. Bitte beachten Sie, dass *N in der Fehlernachricht als Platzhalter verwendet wird, wenn Schemainformationen nicht anwendbar sind:

```
INSERT INTO t VALUES (XMLPARSE (DOCUMENT '
   <x>This is the beginning of the document
       <y>test
          <z>rld12345678901234567890123412345678901234567890123
   4567890123456789012345678900987654321123456789009876 5
   432112344567788098765432112345678904 55</z>
       </y>
   </x>' strip whitespace))

DB20000I  Der SQL-Befehl wurde erfolgreich ausgeführt.

CREATE INDEX i1 ON t(x)
   GENERATE KEY USING XMLPATTERN '/x/y//text()'
   AS SQL VARCHAR(4)
```

```
DB21034E  Der Befehl wurde als SQL-Anweisung verarbeitet, da es sich um
keinen gültigen Befehl des Befehlszeilenprozessors handelte.
Während der SQL-Verarbeitung wurde Folgendes ausgegeben:

SQL20306N  Ein Index für eine XML-Spalte kann nicht erstellt werden,
weil beim Einfügen der XML-Werte in den Index ein Fehler festgestellt wurde.
 Ursachencode = "1".
Bei Ursachencodes für ein XML-Schema ist "*N" die XML-Schemakennung
und "*N" der XML-Schemadatentyp.
SQLSTATE=23526
```

Die Ausgabe in der Protokolldatei **db2diag** sieht wie folgt aus (mit geringfügigen Änderungen an der Formatierung):

```
2007-03-06-12.08.48.437571-480 I10148A1082        LEVEL: Warning
PID     : 1544348              TID  : 1801        PROC : db2sysc
INSTANCE: adua                 NODE : 000         DB   : ADTEST
APPHDL  : 0-30                 APPID: *LOCAL.adua.070306200844
AUTHID  : ADUA
EDUID   : 1801                 EDUNAME: db2agent (ADTEST)
FUNCTION: DB2 UDB, Xml Storage and Index Manager,
          xmlsIkaProcessErrorMsg, probe:361
MESSAGE : ZRC=0x80A50412=-2136669166=XMS_XML_CRIX_ERROR
          "XML node value error during create XML Index"
DATA #1 : String, 36 bytes
SQL Code: SQL20306N ; Reason Code: 1
DATA #2 : String, 72 bytes
To locate the value in the document that caused the error, execute
the following XQuery.
DATA #3 : String, 435 bytes
xquery for $doc in db2-fn:xmlcolumn("ADUA.T.X")[/*:x/*:y/*:z/text()
[fn:starts-with(., "rld12345678901234567890123412345678901234567890123")
and fn:ends-with(., "567890098765432112344567788098765432112345678904555")]]
return
<Result>
 <ProblemDocument> {$doc} </ProblemDocument>
 <ProblemValue> {$doc/*:x/*:y/*:z/text()/..} </ProblemValue>
</Result>;
```

Das Ergebnis der Abfrageanweisung sieht wie folgt aus (mit geringfügigen Änderungen an der Formatierung):

```
<Result>
    <ProblemDocument>
         <x>This is the beginning of the document
           <y>test
              <z>rld12345678901234567890123412345678901234567890123
       456789012345678901234567890098765432112345678900987654
       321123445677880987654321123456789045555</z>
           </y>
       </x>
</ProblemDocument>
 <ProblemValue>
     <z>rld12345678901234567890123412345678901234567890123
     456789012345678901234567890098765432112345678900987654
     321123445677880987654321123456789045555</z>
 </ProblemValue>
</Result>
```

Um den Fehler zu korrigieren, gehen Sie wie folgt vor:

Sie können das XML-Muster von CREATE INDEX ändern, um die maximal zulässige Länge für VARCHAR zu erhöhen:

```
CREATE INDEX i1 ON t(x)
   GENERATE KEY USING XMLPATTERN '/x/y//text()'
   AS SQL VARCHAR(200)
```

Kapitel 7. XML-Daten aktualisieren

Zum Aktualisieren von Daten in einer XML-Spalte können Sie die SQL-Anweisung UPDATE benutzen. Wenn Sie nur bestimmte Zeilen aktualisieren möchten, müssen Sie eine WHERE-Klausel in die Anweisung aufnehmen.

Der gesamte Spaltenwert wird ersetzt. Als Eingabe für die XML-Spalte muss ein korrekt formatiertes XML-Dokument angegeben werden. Als Anwendungsdatentyp kann ein XML-, Zeichen- oder Binärdatentyp verwendet werden.

Beim Aktualisieren einer XML-Spalte kann es erforderlich sein, das XML-Eingabedokument auf der Basis eines registrierten XML-Schemas zu prüfen. Dieser Arbeitsschritt kann mit der Funktion XMLVALIDATE ausgeführt werden.

Sie können die Werte in XML-Spalten verwenden, um anzugeben, welche Zeilen aktualisiert werden sollen. Um Werte in XML-Dokumenten zu suchen, müssen Sie XQuery-Ausdrücke verwenden. XQuery-Ausdrücke können beispielsweise mit dem Vergleichselement XMLEXISTS angegeben werden, mit dem Sie einen XQuery-Ausdruck angeben und feststellen können, ob die Verwendung des Ausdrucks zu einer leeren Sequenz führt. Wenn XMLEXISTS in einer Klausel vom Typ WHERE angegeben wird, werden die Zeilen aktualisiert, wenn der XQuery-Ausdruck keine leere Sequenz zurückgibt.

In den folgenden Beispielen wird dargestellt, wie XML-Daten in XML-Spalten aktualisiert werden können. In den Beispielen wird mit der Tabelle 'MYCUSTOMER' gearbeitet, bei der es sich um eine Kopie der Beispieltabelle 'CUSTOMER' handelt. In den Beispielen wird davon ausgegangen, dass 'MYCUSTOMER' bereits eine Zeile mit einer Kunden-ID (Cid) enthält, die den Wert 1004 aufweist. Es wird davon ausgegangen, dass die XML-Daten, mit denen die vorhandenen Spaltendaten aktualisiert werden sollen, sich in der Datei 'c7.xml' befinden, deren Inhalt folgendes Format hat:

```
<customerinfo Cid="1004">
  <name>Christine Haas</name>
  <addr country="Canada">
    <street>12 Topgrove</street>
        <city>Toronto</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>N9Y-8G9</pcode-zip>
  </addr>
  <phone type="work">905-555-5238</phone>
  <phone type="home">416-555-2934</phone>
</customerinfo>
```

Beispiel: In einer JDBC-Anwendung müssen Sie die XML-Daten aus der Datei 'c7.xml' als Binärdaten einlesen und diese zum Aktualisieren der Daten in einer XML-Spalte verwenden:

```
PreparedStatement updateStmt = null;
String sqls = null;
int cid = 1004;
sqls = "UPDATE MyCustomer SET Info=? WHERE Cid=?";
updateStmt = conn.prepareStatement(sqls);
updateStmt.setInt(1, cid);
File file = new File("c7.xml");
updateStmt.setBinaryStream(2, new FileInputStream(file), (int)file.length());
updateStmt.executeUpdate();
```

Beispiel: In einer eingebetteten C-Anwendung müssen Sie die Daten in einer XML-Spalte über eine binäre XML-Hostvariable aktualisieren:

```
EXEC SQL BEGIN DECLARE SECTION;
      sqlint64 cid;
      SQL TYPE IS XML AS BLOB (10K) xml_hostvar;
EXEC SQL END DECLARE SECTION;
...
cid=1004;
/* Read data from file c7.xml into xml_hostvar */
...
EXEC SQL UPDATE MyCustomer SET Info=:xml_hostvar WHERE Cid=:cid;
```

In diesen Beispielen wird der Wert für das Attribut 'Cid' im Element <customerinfo> auch in der relationalen Spalte für 'CID' gespeichert. Aus diesem Grund hat die WHERE-Klausel in den UPDATE-Anweisungen die relationale Spalte 'CID' zur Angabe der zu aktualisierenden Spalten verwendet. In den Fällen, in denen die Werte, über die die zu aktualisierenden Zeilen festgelegt werden, nur in den XML-Dokumenten selbst gefunden wurden, kann das Vergleichselement XMLEXISTS verwendet werden. Beispiel: Die Anweisung UPDATE in dem zuvor aufgeführten Beispiel für die eingebettete C-Anwendung kann wie folgt geändert werden, um XMLEXISTS zu verwenden:

```
EXEC SQL UPDATE MyCustomer SET Info=:xml_hostvar
       WHERE XMLEXISTS ('$doc/customerinfo[@Cid = $c]'
                        passing INFO as "doc", cast(:cid as integer) as "c");
```

Beispiel: In nachstehendem Beispiel werden die vorhandenen XML-Daten aus der Tabelle 'MYCUSTOMER' aktualisiert. Die SQL-Anweisung UPDATE wird für eine Zeile in der Tabelle 'MYCUSTOMER' ausgeführt und ersetzt das Dokument in der Spalte INFO der Zeile durch die logische Momentaufnahme des durch den Umsetzungsausdruck geänderten Dokuments:

```
UPDATE MyCustomer
SET info = XMLQUERY(
   'transform
    copy $newinfo := $info
    modify do insert <status>Current</status> as last into $newinfo/customerinfo
    return $newinfo' passing info as "info")
WHERE cid = 1004
```

Aktualisierungsausdrücke in einem Umsetzungsausdruck verwenden

In DB2 XQuery müssen Aktualisierungsausdrücke innerhalb der Klausel **MODIFY** eines Umsetzungsausdrucks verwendet werden. Die Aktualisierungsausdrücke wirken sich auf die kopierten Knoten aus, die von der Klausel **COPY** des Umsetzungsausdrucks erstellt werden.

Bei den folgenden Ausdrücken handelt es sich um Aktualisierungsausdrücke:

- Löschausdrücke
- Einfügeausdrücke
- Umbenennungsausdrücke
- Ersetzungsausdrücke
- FLWOR-Ausdrücke, die einen Aktualisierungsausdruck in ihrer Klausel **RETURN** enthalten
- Bedingungsausdrücke, die einen Aktualisierungsausdruck in ihrer Klausel **THEN** oder **ELSE** enthalten
- Zwei oder mehr durch Kommata getrennte Aktualisierungsausdrücke, bei denen alle Operanden entweder Aktualisierungsausdrücke oder eine leere Sequenz sind

Bei ungültigen Aktualisierungsausdrücke gibt DB2 XQuery einen Fehler zurück. Beispiel: DB2 XQuery gibt einen Fehler zurück, wenn eine Verzweigung eines Bedingungsausdrucks einen Aktualisierungsausdruck enthält und die andere Verzweigung einen Nichtaktualisierungsausdruck enthält, der keine leere Sequenz ist.

Ein Umsetzungsausdruck ist kein Aktualisierungsausdruck, da er keine vorhandenen Knoten ändert. Ein Umsetzungsausdruck erstellt geänderte Kopien von vorhandenen Knoten. Das Ergebnis eines Umsetzungsausdrucks kann Knoten umfassen, die von Aktualisierungsausdrücken in der Klausel **MODIFY** des Umsetzungsausdrucks erstellt wurden, sowie Kopien bereits vorhandener Knoten.

XQuery-Aktualisierungsoperationen verarbeiten

In einem Umsetzungsausdruck kann die Klausel **MODIFY** mehrere Aktualisierungen angeben. Die Klausel **MODIFY** kann beispielsweise zwei Aktualisierungsausdrücke enthalten - einen zum Ersetzen eines vorhandenen Wertes und einen anderen zum Einfügen eines neuen Elements. Wenn die Klausel **MODIFY** mehrere Aktualisierungsausdrücke enthält, wird jeder Aktualisierungsausdruck unabhängig von den anderen ausgewertet und resultiert in einer Liste von Änderungsoperationen mit bestimmten Knoten, die von der Klausel **COPY** des Umsetzungsausdrucks erstellt wurden.

In einer Klausel **MODIFY** können Aktualisierungsausdrücke keine neuen Knoten ändern, die von anderen Aktualisierungsausdrücken hinzugefügt werden. Wenn ein Aktualisierungsausdruck beispielsweise einen neuen Elementknoten hinzufügt, kann ein anderer Aktualisierungsausdruck den Knotennamen des neu erstellten Knotens nicht ändern.

Alle in der Klausel **MODIFY** des Umsetzungsausdrucks angegebenen Änderungsoperationen werden erfasst und effektiv in der folgenden Reihenfolge angewandt:

1. Die folgenden Aktualisierungsoperationen werden in einer nicht deterministischen Reihenfolge ausgeführt:
   - Einfügeoperationen ohne Schlüsselwörter für die Sortierung wie beispielsweise **before**, **after**, **as first** oder **as last**.
   - Alle Umbenennungsoperationen.
   - Ersetzungsoperationen, bei denen die Schlüsselwörter **value of** angegeben sind und bei denen der Zielknoten ein Attribut-, Text-, Kommentar- oder Verarbeitungsanweisungsknoten ist.
2. Einfügeoperationen mit Schlüsselwörtern für die Sortierung wie beispielsweise **before**, **after**, **as first** oder **as last**.
3. Ersetzungsoperationen, bei denen die Schlüsselwörter **value of** nicht angegeben sind.
4. Ersetzungsoperationen, bei denen die Schlüsselwörter **value of** angegeben sind und bei denen der Zielknoten ein Elementknoten ist.
5. Alle Löschoperationen.

Die Reihenfolge, in der Änderungsoperationen angewandt werden, stellt sicher, dass eine Reihe mehrerer Änderungen zu einem deterministischen Ergebnis führt. Der letzte XQuery-Ausdruck unter „Beispiele” auf Seite 220 zeigt anhand eines Beispiels, wie die Reihenfolge der Aktualisierungsoperationen dafür sorgt, dass eine Reihe von mehreren Änderungen ein deterministisches Ergebnis hat.

Ungültige XQuery-Aktualisierungsoperationen

Bei der Verarbeitung eines Umsetzungsausdrucks gibt DB2 XQuery einen Fehler zurück, wenn eine der folgenden Bedingungen eintritt:

- Zwei oder mehr Umbenennungsoperationen werden auf denselben Knoten angewandt.
- Zwei oder mehr Ersetzungsoperationen, die die Schlüsselwörter **value of** verwenden, werden auf denselben Knoten angewandt.
- Zwei oder mehr Ersetzungsoperationen, die die Schlüsselwörter **value of** nicht verwenden, werden auf denselben Knoten angewandt.
- Das Ergebnis des Umsetzungsausdrucks ist keine gültige XMD-Instanz.

  Eine XDM-Instanz, die ein Element mit zwei Attributen desselben Namens enthält, ist beispielsweise ungültig.
- Die XDM-Instanz enthält inkonsistente Namensbereichsbindungen.

  Die folgenden Beispiele stehen für inkonsistente Namensbereichsbindungen:
  - Eine Namensbereichsbindung im QName eines Attributknotens ist mit den Namensbereichsbindungen ihres übergeordneten Elementknotens nicht kompatibel.
  - Die Namensbereichsbindungen in zwei Attributknoten mit demselben übergeordneten Element sind miteinander nicht kompatibel.

Beispiele

Im folgenden Beispiel bindet die Klausel **COPY** eines Umsetzungsausdrucks die Variable `$product` an eine Kopie eines Elementknotens, und die Klausel **MODIFY** des Umsetzungsausdrucks verwendet zwei Aktualisierungsausdrücke zum Ändern des kopierten Knotens:

```
  xquery
transform
copy $product := db2-fn:sqlquery(
  "select description from product where pid='100-100-01'")/product
modify(
  do replace value of $product/description/price with 349.95,
  do insert <status>Available</status> as last into $product )
return $product
```

Das nachstehende Beispiel verwendet einen XQuery-Umsetzungsausdruck innerhalb einer SQL-Anweisung UPDATE zum Ändern von XML-Daten in der Tabelle CUSTOMER. Die SQL-Anweisung UPDATE wird für eine Zeile in der Tabelle CUSTOMER ausgeführt. Der Umsetzungsausdruck erstellt eine Kopie des XML-Dokuments anhand der Spalte INFO der Zeile und fügt der Kopie des Dokuments ein Element `status` hinzu. Die Anweisung UPDATE ersetzt das Dokument in der Spalte INFO der Zeile durch die Kopie des durch den Umsetzungsausdruck geänderten Dokuments:

```
UPDATE customer
SET info = xmlquery( 'transform
   copy $newinfo := $info
   modify do insert <status>Current</status> as last into $newinfo/customerinfo
   return $newinfo' passing info as "info")
WHERE cid = 1003
```

In nachstehendem Beispiel wird die Tabelle CUSTOMER aus der DB2-Beispieldatenbank SAMPLE verwendet. In der Tabelle CUSTOMER enthält die XML-Spalte INFO die Adresse und Telefonnummer von Kunden.

In nachstehendem Beispiel wird die SQL-Anweisung SELECT für eine Zeile in der Tabelle CUSTOMER ausgeführt. Die Klausel **COPY** des Umsetzungsausdrucks erstellt eine Kopie des XML-Dokuments anhand der Spalte INFO. Der Löschausdruck löscht Adressinformationen und Telefonnummern (phone) aus der Kopie des Dokuments. Vom Löschen ausgenommen sind die Telefonnummern vom Typ 'work' (Geschäftsnummer). Die Klausel **RETURN** verwendet das Attribut für die Kunden-ID (cid) und das Attribut für das Land (country) aus dem ursprünglichen Dokument der Tabelle CUSTOMER:

```
SELECT XMLQUERY( 'transform
  copy $mycust := $d
  modify
    do delete ( $mycust/customerinfo/addr,
       $mycust/customerinfo/phone[@type != "work"] )
  return
  <custinfo>
    <Cid>{data($d/customerinfo/@Cid)}</Cid>
    {$mycust/customerinfo/*}
    <country>{data($d/customerinfo/addr/@country)}</country>
  </custinfo>'
  passing INFO as "d")
FROM CUSTOMER
WHERE CID = 1003
```

Bei Ausführung für die Datenbank SAMPLE gibt die Anweisung das folgende Ergebnis zurück:

```
<custinfo>
  <Cid>1003</Cid>
    <name>Robert Shoemaker</name>
    <phone type="work">905-555-7258</phone>
  <country>Canada</country>
</custinfo>
```

In nachstehendem Beispiel zeigt der XQuery-Ausdruck, wie die Reihenfolge der Aktualisierungsoperation dafür sorgt, dass eine Reihe von mehreren Änderungen ein deterministisches Ergebnis hat. Der Einfügeausdruck fügt ein Element vom Typ `status` nach dem Element `phone` ein, und der Ersetzungsausdruck ersetzt das Element `phone` durch das Element `email`:

```
  xquery
let $email  := <email>jnoodle@my-email.com</email>
let $status := <status>current</status>
return
  transform
  copy $mycust := db2-fn:sqlquery('select info from customer where cid = 1002')
  modify (
    do replace $mycust/customerinfo/phone with $email,
    do insert $status after $mycust/customerinfo/phone[@type = "work"] )
  return $mycust
```

In der Klausel **MODIFY** steht der Ersetzungsausdruck vor dem Einfügeausdruck. Beim Aktualisieren der kopierten Knotensequenz `$mycust` hingegen wird die Einfügeoperation vor der Ersetzungsoperation ausgeführt, um ein deterministisches Ergebnis zu gewährleisten. Bei Ausführung für die Datenbank SAMPLE gibt der Ausdruck das folgende Ergebnis zurück:

```
<customerinfo Cid="1002">
                                      <name>Jim Noodle</name>
    <addr country="Canada">
                                           <street>25 EastCreek</street>
        <city>Markham</city>
                                           <prov-state>Ontario</prov-state>
        <pcode-zip>N9C 3T6</pcode-zip>
```

```
                                        </addr>
  <email>jnoodle@my-email.com</email>
  <status>current</status>
</customerinfo>
```

Wenn die Ersetzungsoperation zuerst ausgeführt würde, befände sich das Element `phone` nicht in der Knotensequenz, und die Operation zum Einfügen des Elements `status` nach dem Element `phone` wäre sinnlos.

Der Abschnitt „XQuery-Aktualisierungsoperationen verarbeiten" auf Seite 219 enthält Informationen zur Reihenfolge von Aktualisierungsoperationen.

Aktualisieren von XML-Dokumenten mit Informationen aus anderen Tabellen

Sie können Daten aus anderen Datenbankspalten verwenden, um XML-Dokumente zu aktualisieren. Wenn Sie beispielsweise eine Tabelle haben, die Kundeninformationen enthält, können Sie SQL/XML-Anweisungen und XQuery-Ausdrücke verwenden, um die Kundeninformationen in XML-Dokumenten zu aktualisieren.

Erstellen Sie mithilfe der folgenden SQL-Anweisungen eine Beispieltabelle mit neuen Telefonnummern (NewPhones) von Kunden:

```
CREATE TABLE NewPhones (
   CID BIGINT NOT NULL PRIMARY KEY, PhoneNo VARCHAR(20), Type VARCHAR(10))~

INSERT INTO NewPhones (CID, PhoneNo, Type) VALUES (1001, '111-222-3333', 'cell' )~
INSERT INTO NewPhones (CID, PhoneNo, Type) VALUES (1002, '222-555-1111', 'home' )~
INSERT INTO NewPhones (CID, PhoneNo, Type) VALUES (1003, '333-444-2222', 'home' )~
```

Aktualisieren Sie mithilfe der folgenden SQL-Anweisung die Telefonnummer eines Kunden in der Tabelle CUSTOMER. Die Anweisung verwendet die Funktion XMLQUERY sowie einen XQuery-Ausdruck, der aus einem FLWOR-Ausdruck und einem Umsetzungsausdruck (transform) mit einem Einfügeausdruck (insert) besteht:

```
UPDATE CUSTOMER SET INFO = XMLQUERY(
   'let $myphone := db2-fn:sqlquery(''SELECT XMLELEMENT(Name "phone",
        XMLATTRIBUTES( NewPhones.Type as "type" ), NewPhones.PhoneNo )
        FROM NewPhones WHERE CID = parameter(1)'', $mycid )
   return
     transform
     copy $mycust := $d
     modify
      do insert $myphone after $mycust/customerinfo/phone[last()]
     return
       $mycust'
  passing INFO as "d", 1002 as "mycid" )
WHERE CID = 1002~
```

Die Funktion XMLQUERY führt einen XQuery-Ausdruck aus, der den Kundeninformation einen Elementknoten `phone` hinzufügt und die geänderten Informationen an die Anweisung UPDATE zurückgibt. In der Klausel **LET** des XQuery-FLWOR-Ausdrucks führt die Funktion db2-fn:sqlquery eine SQL-Fullselect-Anweisung aus. Der Fullselect verwendet die Kunden-ID, die von XMLQUERY an den XQuery-Ausdruck übergeben wurde.

Die Fullselect-Anweisung muss einen XML-Datentyp zurückgeben. Um einen XML-Datentyp aus den Daten von PHONENO und TYPE zu erstellen, die von der Anweisung SELECT zurückgegeben werden, erstellen die Funktionen XMLELEMENT und XMLATTRIBUTES einen Elementknoten `phone` auf der Grundlage der gelieferten Daten.

In diesem Beispiel erstellt der von db2-fn:sqlquery in der Klausel **LET** ausgeführte Fullselect den folgenden Elementknoten phone:

```
<phone type="home">222-555-1111<phone>
```

Führen Sie die folgende SQL-Anweisung SELECT aus, um die Kundeninformationen anzuzeigen, die jetzt sowohl eine geschäftliche als auch eine private Telefonnummer enthalten:

```
SELECT INFO FROM CUSTOMER WHERE CID = 1002~
```

Löschen von XML-Daten aus Tabellen

Verwenden Sie zum Löschen von Zeilen, die XML-Dokumente enthalten, die SQL-Anweisung DELETE. Wenn Sie nur bestimmte Zeilen löschen möchten, müssen Sie eine WHERE-Klausel in die Anweisung aufnehmen.

Sie können die zu löschenden Zeilen auf der Basis von Werten angeben, die in XML-Spalten enthalten sind. Um Werte in XML-Dokumenten zu suchen, müssen Sie XQuery-Ausdrücke verwenden. XQuery-Ausdrücke können beispielsweise mit dem Vergleichselement XMLEXISTS angegeben werden, mit dem Sie einen XQuery-Ausdruck angeben und feststellen können, ob die Verwendung des Ausdrucks zu einer leeren Sequenz führt. Wenn XMLEXISTS in einer Klausel vom Typ WHERE angegeben wird, werden die Zeilen gelöscht, wenn der XQuery-Ausdruck keine leere Sequenz zurückgibt.

Eine XML-Spalte muss entweder den Wert NULL aufweisen oder ein korrekt formatiertes XML-Dokument enthalten. Um ein XML-Dokument aus einer XML-Spalte zu löschen, ohne die Zeile selbst zu löschen, müssen Sie die SQL-Anweisung UPDATE mit der Option SET NULL angeben, um die Spalte auf den Wert NULL zu setzen, wenn in der Spalte Nullwerte enthalten sein dürfen. Um Objekte wie beispielsweise Attribute oder Elemente aus einem vorhandenen XML-Dokument zu löschen, müssen Sie die SQL-Anweisung UPDATE mit XQuery-Aktualisierungsausdrücken verwenden. Mithilfe von XQuery-Aktualisierungsausdrücken können Änderungen an einer Kopie eines vorhandenen XML-Dokuments vorgenommen werden. Anschließend wendet die Anweisung UPDATE die vom XQuery-Aktualisierungsausdruck zurückgegebene geänderte Kopie auf die XML-Spalte für die angegebene Zeile an.

In den folgenden Beispielen wird dargestellt, wie XML-Daten aus XML-Spalten gelöscht werden können. Die Beispiele arbeiten mit der Tabelle 'MyCustomer', bei der es sich um eine Kopie der Beispieltabelle 'Customer' handelt. Außerdem wird davon ausgegangen, dass 'MyCustomer' mit allen Daten aus der Tabelle 'Customer' gefüllt wurde.

Beispiel: Löschen Sie die Zeilen aus der Tabelle 'MyCustomer', deren Spalte für die Kunden-ID (Cid) den Wert 1002 aufweist.

```
DELETE FROM MyCustomer WHERE Cid=1002
```

Beispiel: Löschen Sie die Zeilen aus der Tabelle 'MyCustomer', deren Element city den Wert Markham aufweist. Mit dieser Anweisung wird die Zeile gelöscht, deren Kunden-ID 1002 lautet.

```
DELETE FROM MyCustomer
 WHERE XMLEXISTS ('$d//addr[city="Markham"]' passing INFO as "d")
```

Beispiel: Löschen Sie in der Zeile von 'MyCustomer' das XML-Dokument, dessen Wert für das Element `city` `Markham` lautet, während die Zeile selbst jedoch beibehalten wird. Diese Anweisung löscht die XML-Daten aus der Spalte 'Info' der Zeile, deren Kunden-ID 1002 lautet.

```
UPDATE MyCustomer SET Info = NULL
  WHERE XMLEXISTS ('$d//addr[city="Markham"]' passing INFO as "d")
```

Beispiel: In nachstehendem Beispiel werden die Telefonnummern aus vorhandenen XML-Daten aus der Tabelle 'MyCustomer' gelöscht. Die SQL-Anweisung UPDATE wird für eine Zeile in der Tabelle 'MyCustomer' ausgeführt. Der XQuery-Umsetzungsausdruck erstellt eine Kopie des XML-Dokuments aus der Spalte INFO der Zeile und entfernt mithilfe des XQuery-Löschausdrucks die Telefonnummern (phone) des Typs 'work' (am Arbeitsplatz) aus der Kopie des Dokuments. Die Anweisung UPDATE ersetzt das Dokument in der Spalte INFO der Zeile durch die Kopie des durch den Umsetzungsausdruck geänderten Dokuments:

```
UPDATE MyCustomer
SET info = XMLQUERY(
   'transform
    copy $newinfo := $info
    modify do delete ($newinfo/customerinfo/phone[@type="work"])
    return $newinfo' passing info as "info")
WHERE cid = 1004
```

Kapitel 8. XML-Schema-Repository

Das XML-Schema-Repository (XSR) ist ein Repository für alle XML-Artefakte, die zur Verarbeitung von XML-Instanzdokumenten verwendet werden, die in XML-Spalten gespeichert sind. Der Zweck des XML-Schema-Repositorys besteht darin, Sie bei der Durchführung von Tasks zu unterstützen, die von diesen XML-Artefakten abhängen.

XML-Instanzdokumente können einen Verweis auf eine URI (Uniform Resource Identifier) enthalten, die auf ein zugehöriges XML-Schema, eine Dokumenttypdeklaration (DTD) oder eine andere externe Entität verweist. Diese URI ist zur Verarbeitung des Instanzdokuments erforderlich. Das DB2-Datenbanksystem verwaltet Abhängigkeiten zu derartigen XML-Artefakten mit externen Verweisen mithilfe des XSR, ohne dass hierbei Änderungen am URI-Positionsverweis erforderlich sind.

Ohne diesen Mechanismus zur Speicherung zugehöriger XML-Schemata, DTDs und externer Entitäten kann auf eine externe Ressource möglicherweise nicht zugegriffen werden, wenn diese von der Datenbank benötigt wird. Außerdem können in diesem Fall auch Änderungen an der Ressource auftreten, ohne dass gleichzeitig die entsprechenden Änderungen an einem bereits geprüften und mit Annotationen versehenen XML-Dokument durchgeführt werden, das in der Datenbank gespeichert ist. Das XSR eliminiert auch zusätzliche Systemaufwände, die beim Suchen externer Dokumente anfallen, sowie die sich daraus ergebenden negativen Auswirkungen auf die Systemleistung.

Jede Datenbank enthält ein XML-Schema-Repository, das sich im Datenbankkatalog befindet und Katalogtabellen, Katalogsichten und bestimmte integrierte gespeicherte Prozeduren umfasst, mit denen Daten in diese Katalogtabellen eingegeben werden können.

XSR-Objekte

Das XML-Schema-Repository (XSR) unterstützt die Erstellung einer Kopie der Daten, die in einem XML-Schema, einer Dokumenttypdeklaration (DTD) oder einer externen Entität als XSR-Objekt enthalten sind. Diese Informationen werden zur Auswertung und Verarbeitung von XML-Instanzdokumenten verwendet, die in XML-Spalten gespeichert sind.

Neue XSR-Objekte müssen vor der Verwendung explizit zum XML-Schema-Repository (XSR) hinzugefügt werden. Hierzu wird für die Objekte eine Registrierung durchgeführt, in deren Verlauf das XML-Schema, die DTD oder externe Entität identifiziert wird. XSR-Objekte können über Java-Anwendungen, gespeicherte Prozeduren oder den Befehlszeilenprozessor registriert werden.

Das am häufigsten verwendete XSR-Objekt ist das XML-Schema. Jedes XML-Schema im XML-Schema-Repository kann aus einem oder aus mehreren XML-Schemadokumenten bestehen. Wenn das XML-Schema aus mehreren Dokumenten besteht, wird zum Starten des Registrierungsprozesses das primäre XML-Schemadokument benutzt. Wenn das XML-Schema aus nur einem Dokument besteht, dann ist dieses gleichzeitig das primäre XML-Schemadokument.

Eine Beschreibung der Syntax von gespeicherten Prozeduren und Befehlen für XSR finden Sie in Anhang C, „Gespeicherte Prozeduren und Befehle für XSR", auf Seite 517.

Registrierung von XSR-Objekten

Bevor ein XML-Schema, eine Dokumenttypdeklaration (DTD) oder eine externe Entität für für die Verarbeitung von XML-Dokumenten verwendet werden kann, muss diese Komponente im XML-Schema-Repository (XSR) registriert werden. Durch die Registrierung im XML-Schema-Repository wird ein XSR-Objekt erstellt.

Zur Registrierung der meisten XML-Schemata muss der Wert für den Konfigurationsparameter für die Größe des Anwendungszwischenspeichers (applheapsz) erhöht werden. Zur Registrierung sehr komplexer XML-Schemata unter einem 32-Bit-Windows-Betriebssystem muss möglicherweise auch der Konfigurationsparameter für die Stapelspeichergröße des Agenten (agent_stack_sz) erhöht werden. Informationen zur Änderung dieser Konfigurationsparameter finden Sie unter den im Folgenden aufgeführten Links.

Bei XML-Schemata umfasst die XSR-Objektregistrierung die folgenden Arbeitsschritte:

1. Registrieren des primären XML-Schemadokuments im XML-Schema-Repository.
2. Angeben zusätzlicher XML-Schemadokumente, die in das XSR-Objekt eingebunden werden sollen. Dieser Schritt ist nur erforderlich, wenn das XML-Schema aus mehreren Schemadokumenten besteht.
3. Ausführen des Registrierungsprozesses im XML-Schema-Repository.

Bei DTDs und externen Entitäten muss zur XSR-Objektregistrierung im XML-Schema-Repository nur ein einziger Arbeitsschritt ausgeführt werden.

Die Arbeitsschritte zur XSR-Objektregistrierung können mithilfe der folgenden Komponenten ausgeführt werden:

- Java-Anwendungen
- Gespeicherte Prozeduren
- Befehlszeilenprozessor

Bitte beachten Sie, dass keine Befehle von Befehlszeilenprozessoren (CLPs) verwendet werden können, um XML-Schemata von Hostanwendungen zu registrieren, da die erforderlichen Dateiinformationen über diese Befehle nicht übergeben werden können. Um XML-Schemata von Anwendungen zu registrieren, die eine Verbindung zu einer DB2-Datenbank über einen CLI/ODBC- oder JDBC-Treiber herstellen, müssen Sie gespeicherte Prozeduren verwenden.

In der Beschreibung dieser Methoden wird das folgende Beispiel eines XML-Schemas, das aus zwei XML-Schemadokumenten besteht, verwendet: "PO.xsd" und "address.xsd", die beide lokal in C:\TEMP gespeichert sind. Der Benutzer möchte dieses Schema unter dem zweiteiligen SQL-Namen "user1.POschema" registrieren. Das XML-Schema verfügt über eine zugeordnete Eigenschaftendatei mit dem Namen 'schemaProp.xml'. Diese Eigenschaftendatei wird ebenfalls lokal im selben Verzeichnis C:\TEMP gespeichert. Den beiden XML-Schemadokumenten sind keine Eigenschaften zugeordnet. Der Benutzer definiert die URI, unter der dieses Schema extern identifiziert werden kann, als "http://myPOschema/PO".

Zugriffsrechte

Jeder Benutzer mit der Berechtigung DBADM kann ein XSR-Objekt registrieren. Bei allen anderen Benutzern richten sich die Zugriffsrechte nach dem SQL-Schema, das während des Registrierungsprozesses bereitgestellt wird. Wenn das SQL-Schema nicht vorhanden ist, dann ist die Berechtigung IMPLICIT_SCHEMA für die Datenbank erforderlich, um das Schema zu registrieren. Wenn das SQL-Schema hingegen vorhanden ist, benötigt der Benutzer, der das Schema registriert, das Zugriffsrecht CREATEIN für das SQL-Schema.

Bei XML-Schemata muss der Benutzer, der die XSR-Objektregistrierung startet (z. B. über die gespeicherte Prozedur XSR_REGISTER), auch zusätzliche XML-Schemadokumente (sofern vorhanden) angeben und den Registrierungsprozess ausführen.

Das Zugriffsrecht USAGE für ein XSR-Objekt wird dem Ersteller des XSR-Objekts automatisch zugeteilt.

Registrieren von XSR-Objekten über gespeicherte Prozeduren

Wenn eine Datenbank erstellt wird, werden auch die gespeicherten Prozeduren erstellt, die zum Registrieren des XML-Schemas benötigt werden. Um XML-Schemata über eine gespeicherte Prozedur zu registrieren, müssen die gespeicherten Prozeduren XSR_REGISTER, XSR_ADDSCHEMADOC und XSR_COMPLETE mit der Anweisung CALL aufgerufen werden.

Ein XML-Schemadokument wird beim Registrieren oder Hinzufügen von Dokumenten nicht auf seine Richtigkeit überprüft. Dokumentprüfungen werden nur beim Abschluss der XML-Schemaregistrierung durchgeführt.

XML-Schemata registrieren:

1. Registrieren Sie das primäre XML-Schemadokument, indem Sie die gespeicherte Prozedur SYSPROC.XSR_REGISTER aufrufen:

   ```
   CALL SYSPROC.XSR_REGISTER ('user1', 'POschema', 'http://myPOschema/PO',
                                                    :content_host_var, NULL)
   ```

2. Vor Abschluss der Registrierung müssen Sie alle weiteren XML-Schemadokumente hinzufügen, die in das primäre XML-Schema eingebunden werden sollen. Bitte beachten Sie, dass jedes zusätzliche Schemadokument jeweils nur ein Mal eingeschlossen werden kann. Im vorliegenden Beispiel ist dieser Arbeitsschritt nicht optional, weil das XML-Schema aus zwei XML-Schemadokumenten besteht, die beide registriert werden müssen. Verwenden Sie die gespeicherte Prozedur XSR_ADDSCHEMADOC, um weitere XML-Schemadokumente hinzuzufügen. Das folgende Beispiel veranschaulicht das Hinzufügen von Schemakonstrukten für Adressen zum XSR-Objekt:

   ```
   CALL SYSPROC.XSR_ADDSCHEMADOC ('user1', 'POschema', 'http://myPOschema/address',
                                                    :content_host_var, NULL)
   ```

3. Führen Sie die Registrierung aus, indem Sie die gespeicherte Prozedur SYSPROC.XSR_COMPLETE aufrufen. Im folgenden Beispiel gibt der letzte Parameter an, dass das XML-Schema nicht für die Dekomposition verwendet wird. (Der Wert 1 gibt hingegen an, dass das XML-Schema für die Dekomposition verwendet wird.)

   ```
   CALL SYSPROC.XSR_COMPLETE ('user1', 'POschema', :schemaproperty_host_var, 0)
   ```

Zugriffsrechte

Jeder Benutzer mit der Berechtigung DBADM kann ein XML-Schema registrieren. Bei allen anderen Benutzern richten sich die Zugriffsrechte nach dem SQL-Schema, das während des Registrierungsprozesses bereitgestellt wird. Wenn das SQL-Schema nicht vorhanden ist, dann ist die Berechtigung IMPLICIT_SCHEMA für die Datenbank erforderlich, um das Schema zu registrieren. Wenn das SQL-Schema hingegen vorhanden ist, benötigt der Benutzer, der das Schema registriert, das Zugriffsrecht CREATEIN für das SQL-Schema.

Das Zugriffsrecht USAGE für ein XSR-Objekt wird dem Ersteller des XSR-Objekts automatisch zugeteilt.

Registrieren von XSR-Objekten über den Befehlszeilenprozessor

Um XML-Schemata über den Befehlszeilenprozessor zu registrieren, müssen Sie die Befehle REGISTER XMLSCHEMA, ADD XMLSCHEMA DOCUMENT und COMPLETE XMLSCHEMA verwenden.

Ein XML-Schemadokument wird beim Registrieren oder Hinzufügen von Dokumenten nicht auf seine Richtigkeit überprüft. Dokumentprüfungen werden nur beim Abschluss der Schemaregistrierung durchgeführt.

Bitte beachten Sie, dass keine Befehle von Befehlszeilenprozessoren (CLPs) verwendet werden können, um XML-Schemata von Hostanwendungen zu registrieren, da die erforderlichen Dateiinformationen über diese Befehle nicht übergeben werden können. Um XML-Schemata von Anwendungen zu registrieren, die eine Verbindung zu einer DB2-Datenbank über einen CLI/ODBC- oder JDBC-Treiber herstellen, müssen Sie gespeicherte Prozeduren verwenden.

XML-Schemata registrieren:

1. Registrieren Sie das primäre XML-Schemadokument, indem Sie den Befehl REGISTER XMLSCHEMA eingeben:
   ```
   REGISTER XMLSCHEMA 'http://myPOschema/PO'
   FROM 'file://c:/TEMP/PO.xsd'
   AS user1.POschema
   ```
2. Vor Abschluss der Registrierung können Sie optional weitere XML-Schemadokumente hinzufügen, die in das primäre XML-Schema eingebunden werden sollen. Mit dem Befehl ADD XMLSCHEMA DOCUMENT können Sie zusätzliche XML-Schemadokumente hinzufügen. Bitte beachten Sie, dass jedes zusätzliche Schemadokument jeweils nur ein Mal eingeschlossen werden kann. Das folgende Beispiel veranschaulicht das Hinzufügen von Schemakonstrukten für Adressen zum Speicher:
   ```
   ADD XMLSCHEMA DOCUMENT TO user1.POschema
         ADD 'http://myPOschema/address'
         FROM 'file://c:/TEMP/address.xsd'
   ```
3. Führen Sie die Registrierung aus, indem Sie den Befehl COMPLETE XMLSCHEMA eingeben:
   ```
   COMPLETE XMLSCHEMA user1.POschema
   WITH 'file://c:TEMP/schemaProp.xml'
   ```

Zugriffsrechte

Jeder Benutzer mit der Berechtigung DBADM kann ein XML-Schema registrieren. Bei allen anderen Benutzern richten sich die Zugriffsrechte nach dem SQL-Schema,

das während des Registrierungsprozesses bereitgestellt wird. Wenn das SQL-Schema nicht vorhanden ist, dann ist die Berechtigung IMPLICIT_SCHEMA für die Datenbank erforderlich, um das Schema zu registrieren. Wenn das SQL-Schema hingegen vorhanden ist, benötigt der Benutzer, der das Schema registriert, das Zugriffsrecht CREATEIN für das SQL-Schema.

Das Zugriffsrecht USE für XSR-Objekte wird dem Ersteller des XSR-Objekts automatisch zugeteilt.

Java-Unterstützung für die XML-Schemaregistrierung und -entfernung

Der IBM Data Server Driver for JDBC and SQLJ bietet verschiedene Methoden zum Schreiben von Java-Anwendungsprogrammen zur Registrierung und Entfernung von XML-Schemata und deren Komponenten.

Folgende Methoden sind möglich:

DB2Connection.registerDB2XMLSchema
Registriert mithilfe eines oder mehrerer XML-Schemadokumente ein XML-Schema in DB2. Diese Methode liegt in zwei Formaten vor: Einem Format für XML-Schemadokumente, bei denen es sich um die Eingabe über Eingabedatenstromobjekte (InputStream) handelt, und einem Format für XML-Schemadokumente in einer Zeichenfolge (`String`).

DB2Connection.deregisterDB2XMLObject
Entfernt eine XML-Schemadefinition aus DB2.

DB2Connection.updateDB2XmlSchema
Ersetzt die XML-Schemadokumente in einem registrierten XML-Schema mit den XML-Schemadokumenten aus einem anderen registrierten XML-Schema. Löscht optional das XML-Schema, dessen Inhalt kopiert wird. Diese Methode steht nur für Verbindungen zu DB2 Database for Linux, UNIX and Windows zur Verfügung.

Bevor Sie diese Methoden aufrufen können, müssen die gespeicherten Prozeduren, die diese Methoden unterstützen, auf dem DB2-Datenbankserver installiert werden.

Beispiel: Registrierung eines XML-Schemas: Das folgende Beispiel zeigt die Registrierung eines XML-Schemas in DB2 mit 'registerDB2XmlSchema' mithilfe eines einzelnen XML-Schemadokuments (customer.xsd), das aus einem Eingabedatenstrom gelesen wird. Der SQL-Schemaname des registrierten Schemas ist SYSXSR. Es werden keine weiteren Eigenschaften registriert.

```
public static void registerSchema(
  Connection con,
  String schemaName)
  throws SQLException {
  // Define the registerDB2XmlSchema parameters
  String[] xmlSchemaNameQualifiers = new String[1];
  String[] xmlSchemaNames = new String[1];
  String[] xmlSchemaLocations = new String[1];
  InputStream[] xmlSchemaDocuments = new InputStream[1];
  int[] xmlSchemaDocumentsLengths = new int[1];
  java.io.InputStream[] xmlSchemaDocumentsProperties = new InputStream[1];
  int[] xmlSchemaDocumentsPropertiesLengths = new int[1];
  InputStream xmlSchemaProperties;
  int xmlSchemaPropertiesLength;
  //Set the parameter values
  xmlSchemaLocations[0] = "";
  FileInputStream fi = null;
```

```
  xmlSchemaNameQualifiers[0] = "SYSXSR";
  xmlSchemaNames[0] = schemaName;
  try {
    fi = new FileInputStream("customer.xsd");
    xmlSchemaDocuments[0] = new BufferedInputStream(fi);
  } catch (FileNotFoundException e) {
         e.printStackTrace();
  }
  try {
     xmlSchemaDocumentsLengths[0] = (int) fi.getChannel().size();
     System.out.println(xmlSchemaDocumentsLengths[0]);
  } catch (IOException e1) {
         e1.printStackTrace();
  }
  xmlSchemaDocumentsProperties[0] = null;
  xmlSchemaDocumentsPropertiesLengths[0] = 0;
  xmlSchemaProperties = null;
  xmlSchemaPropertiesLength = 0;
  DB2Connection ds = (DB2Connection) con;
  // Invoke registerDB2XmlSchema
  ds.registerDB2XmlSchema(
    xmlSchemaNameQualifiers,
    xmlSchemaNames,
    xmlSchemaLocations,
    xmlSchemaDocuments,
    xmlSchemaDocumentsLengths,
    xmlSchemaDocumentsProperties,
    xmlSchemaDocumentsPropertiesLengths,
    xmlSchemaProperties,
    xmlSchemaPropertiesLength,
    false);
}
```

Beispiel: Entfernung eines XML-Schemas: Das folgende Beispiel zeigt die Entfernung eines XML-Schemas aus DB2 mit 'deregisterDB2XmlObject'. Der SQL-Schemaname des registrierten Schemas ist SYSXSR.

```
public static void deregisterSchema(
  Connection con,
  String schemaName)
  throws SQLException {
  // Define and assign values to the deregisterDB2XmlObject parameters
  String xmlSchemaNameQualifier = "SYSXSR";
  String xmlSchemaName = schemaName;
  DB2Connection ds = (DB2Connection) con;
  // Invoke deregisterDB2XmlObject
  ds.deregisterDB2XmlObject(
    xmlSchemaNameQualifier,
    xmlSchemaName);
}
```

Beispiel: Aktualisierung eines XML-Schemas: Das folgende Beispiel bezieht sich nur auf Verbindungen zu DB2 Database for Linux, UNIX and Windows. Es veranschaulicht die Verwendung von updateDB2XmlSchema zur Aktualisierung des Inhalts eines XML-Schemas durch den Inhalt eines anderen XML-Schemas. Das kopierte Schema verbleibt im Repository. Der SQL-Schemaname für beide registrierte Schemata ist SYSXSR.

```
public static void updateSchema(
  Connection con,
  String schemaNameTarget,
  String schemaNameSource)
  throws SQLException {
  // Define and assign values to the updateDB2XmlSchema parameters
  String xmlSchemaNameQualifierTarget = "SYSXSR";
  String xmlSchemaNameQualifierSource = "SYSXSR";
```

```
  String xmlSchemaNameTarget = schemaNameTarget;
  String xmlSchemaNameSource = schemaNameSource;
  boolean dropSourceSchema = false;
  DB2Connection ds = (DB2Connection) con;
  // Invoke updateDB2XmlSchema
  ds.updateDB2XmlSchema(
    xmlSchemaNameQualifierTarget,
    xmlSchemaNameTarget,
  xmlSchemaNameQualifierSource,
  xmlSchemaNameSource,
    dropSourceSchema);
}
```

Ändern registrierter XSR-Objekte

Nach der Registrierung im XML-Schema-Repository können die XSR-Objekte geändert werden, um die Dekomposition zu aktivieren oder zu inaktivieren. Darüber hinaus können diese Objekte gelöscht oder mit einem Kommentar versehen werden. Für die registrierten XSR-Objekte können außerdem Nutzungszugriffsrechte erteilt oder widerrufen werden.

Informationen zu diesem Vorgang

Das XML-Schema-Repository wird zur Verwaltung von Abhängigkeiten von XML-Dokumenten zu bestimmten XML-Schemata, Dokumenttypdeklarationen (DTDs) oder externen Einheiten verwendet. Alle diese XML-Schemata, DTDs oder externen Einheiten müssen zuerst als neues XSR-Objekt im XML-Schema-Repository registriert werden.

Weiterentwicklung eines XML-Schemas

Ein im XSR (XML-Schema-Repository) registriertes Schema kann zu einem neuen, kompatiblen XML-Schema weiterentwickelt werden, ohne dass bereits gespeicherte XML-Instanzdokumente erneut auf Gültigkeit geprüft werden müssen.

Es wird lediglich das im XSR registrierte XML-Schema aktualisiert. Die gespeicherten XML-Instanzdokumente, einschließlich ihrer URI-Kennungen, bleiben unverändert.

Vorbereitende Schritte

Ein Schema kann nur dann weiterentwickelt werden, wenn das neue Schema mit dem ursprünglichen Schema kompatibel ist. Sind die beiden Schemata nicht kompatibel, gibt die gespeicherte Prozedur XSR_UPDATE bzw. der Befehl UPDATE XMLSCHEMA einen Fehler zurück, und es findet keine Weiterentwicklung des Schemas statt. Der Abschnitt *Kompatibilitätsanforderungen für das Weiterentwickeln von XML-Schemata* enthält Informationen hierzu.

Informationen zu diesem Vorgang

Um ein XML-Schema im XSR weiterzuentwickeln, gehen Sie wie folgt vor:

Vorgehensweise

1. Rufen Sie die gespeicherte Prozedur XSR_REGISTER auf, oder führen Sie den Befehl REGISTER XMLSCHEMA aus, um das neue XML-Schema im XSR zu registrieren.

2. Rufen Sie anschließend die gespeicherte Prozedur XSR_UPDATE auf, oder führen Sie den Befehl UPDATE XMLSCHEMA aus, um das neue XML-Schema im XSR zu aktualisieren.

Nächste Schritte

Bei einer erfolgreichen Weiterentwicklung des Schemas wird das ursprüngliche XML-Schema ersetzt. Im Anschluss an die Weiterentwicklung ist nur noch das aktualisierte XML-Schema verfügbar.

Kompatibilitätsanforderungen für das Weiterentwickeln von XML-Schemata

Um ein XML-Schema im XSR (XML-Schema-Repository) weiterentwickeln zu können, müssen das ursprüngliche XML-Schema und das neue, für die Aktualisierung verwendete XML-Schema hinreichend ähnlich sein.

Wenn die beiden XML-Schemata nicht kompatibel sind, schlägt die Aktualisierung fehl und es wird eine Fehlernachricht generiert. Die folgenden zehn Kompatibilitätskriterien müssen erfüllt sein, damit eine Aktualisierung durchgeführt werden kann. Zur Veranschaulichung werden Beispiele von Schemata gezeigt, die die beschriebenen Anforderungen nicht erfüllen.

Attributinhalt

Attribute, die in einem komplexen Typ im ursprünglichen XML-Schema deklariert sind oder auf die dort verwiesen wird, müssen auch im neuen XML-Schema vorhanden sein. Ebenso gilt, dass im neuen XML-Schema keine erforderlichen Attribute vorhanden sein dürfen, die nicht auch im ursprünglichen XML-Schema vorkommen.

Beispiel 1

Ursprüngliches XML-Schema:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="root">
                <xs:complexType>
      <xs:attribute name="a" type="xs:string"/>
      <xs:attribute name="b" use="optional" type="xs:string"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Neues XML-Schema:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="root">
                <xs:complexType>
      <xs:attribute name="a" type="xs:string"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Beispiel 2

Ursprüngliches XML-Schema:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="root">
```

```
                <xs:complexType>
      <xs:attribute name="a" type="xs:string"/>
  </xs:complexType>
  </xs:element>
</xs:schema>
```

Neues XML-Schema:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="root">
                <xs:complexType>
      <xs:attribute name="a" type="xs:string"/>
      <xs:attribute name="b" type="xs:string" use="required" />
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Elementinhalt

Elemente, die in einem komplexen Typ im ursprünglichen XML-Schema deklariert sind oder auf die dort verwiesen wird, müssen auch im neuen XML-Schema vorhanden sein. Im neuen XML-Schema dürfen keine erforderlichen Elemente vorhanden sein, die nicht auch im ursprünglichen XML-Schema vorkommen. Lediglich optionale Elemente dürfen hinzugefügt werden.

Beispiel 1

Ursprüngliches XML-Schema:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="root">
                <xs:complexType>
      <xs:sequence>
        <xs:element name="a" type="xs:string"/>
        <xs:element name="b" minOccurs="0" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Neues XML-Schema:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="root">
                <xs:complexType>
      <xs:sequence>
        <xs:element name="a" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Beispiel 2

Ursprüngliches XML-Schema:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="root">
                <xs:complexType>
      <xs:sequence>
        <xs:element name="a" type="xs:string"/>
```

```
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Neues XML-Schema:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="root">
                <xs:complexType>
      <xs:sequence>
        <xs:element name="a" type="xs:string"/>
            <xs:element name="b" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Beispiel 3

Ursprüngliches XML-Schema:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="a" type="xs:string"/>
  <xs:element name="b" substitutionGroup="a"/>
  <xs:element name="root">
                <xs:complexType>
      <xs:sequence>
            <xs:element ref="a"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Neues XML-Schema:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="a" type="xs:string"/>
  <xs:element name="b" type="xs:string"/>
  <xs:element name="root">
                <xs:complexType>
      <xs:sequence>
            <xs:element ref="a"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Fassettenkonflikt

Der Fassettenwert eines einfachen Typs im neuen XML-Schema muss mit dem Wertebereich des im ursprünglichen XML-Schema definierten einfachen Typs kompatibel sein.

Beispiel 1

Ursprüngliches XML-Schema:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="foo" >
       <xs:simpleType>
```

```
        <xs:restriction base="xs:decimal" />
        </xs:simpleType>
      </xs:element>
</xs:schema>
```

Neues XML-Schema:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="foo">
    <xs:simpleType>
      <xs:restriction base="xs:decimal">
        <xs:totalDigits value="7"/>
      </xs:restriction>    </xs:simpleType>
  </xs:element>
</xs:schema>
```

Beispiel 2

Ursprüngliches XML-Schema:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="foo">
    <xs:simpleType>
      <xs:restriction base="xs:decimal">
        <xs:totalDigits value="7"/>
        <xs:fractionDigits value="3"/>
        <xs:maxInclusive value="300.00"/>
        <xs:minInclusive value="1.0"/>      </xs:restriction>
    </xs:simpleType>
  </xs:element>
</xs:schema>
```

Neues XML-Schema:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="foo">
    <xs:simpleType>
      <xs:restriction base="xs:decimal">
        <xs:totalDigits value="5"/>
        <xs:fractionDigits value="2"/>
        <xs:pattern value="(0|1|2|3|4|5|6|7|8|9|\.)*"/>
        <xs:maxInclusive value="100.00"/>
        <xs:minInclusive value="10.00"/>      </xs:restriction>
    </xs:simpleType>
  </xs:element>
</xs:schema>
```

Inkompatibler Typ

Der Typ eines Elements oder Attributs im neuen XML-Schema ist nicht kompatibel, wenn bereits eingefügte XML-Dokumente eine Gültigkeitsprüfung aufgrundlage des neuen Schemas nicht bestehen würden oder wenn das Schema eine Annotation eines einfachen Typs enthält, die sich von der Annotation im ursprünglichen XML-Schema unterscheidet.

Beispiel

Ursprüngliches XML-Schema:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <xs:element name="a" type="xs:string"/>
</xs:schema>
```

Neues XML-Schema:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="a" type="xs:integer"/>
</xs:schema>
```

Gemischter Inhalt und nicht gemischter Inhalt

Wenn das Inhaltsmodell eines komplexen Typs im ursprünglichen XML-Schema als gemischt ('mixed' mit dem Wert 'true') deklariert ist, darf das Modell im neuen XML-Schema nicht als 'nicht gemischt' ('mixed' mit dem Wert 'false') deklariert sein.

Beispiel

Ursprüngliches XML-Schema:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="root">
                <xs:complexType>
      <xs:complexContent mixed="true">
        <xs:restriction base="xs:anyType">
          <xs:attribute name="a" type="xs:string"/>
        </xs:restriction>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Neues XML-Schema:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="root">
                <xs:complexType>
      <xs:complexContent mixed="false">
        <xs:restriction base="xs:anyType">
          <xs:attribute name="a" type="xs:string"/>
        </xs:restriction>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Auf null festlegbar und nicht auf null festlegbar

Wenn das Attribut 'nillable' (auf null festlegbar) in einer Elementdeklaration des ursprünglichen XML-Schemas aktiviert ist (true), muss es auch im neuen XML-Schema aktiviert sein.

Beispiel

Ursprüngliches XML-Schema:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="root">
                <xs:complexType>
      <xs:sequence>
        <xs:element name="a" nillable="true" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Neues XML-Schema:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="root">
               <xs:complexType>
      <xs:sequence>
        <xs:element name="a" nillable="false" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Entferntes Element

Globale Elemente, die im ursprünglichen XML-Schema deklariert sind, müssen auch im neuen XML-Schema vorhanden sein und dürfen nicht abstrakt sein ('abstract' mit dem Wert 'true').

Beispiel 1

Ursprüngliches XML-Schema:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="a" type="xs:string"/>
      <xs:element name="b" type="xs:string"/>
</xs:schema>
```

Neues XML-Schema:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="a" type="xs:string"/>
</xs:schema>
```

Beispiel 2

Ursprüngliches XML-Schema:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="a" type="xs:string"/>
      <xs:element name="b" type="xs:string"/>
</xs:schema>
```

Neues XML-Schema:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="a" type="xs:string"/>
  <xs:element name="b" abstract="true" type="xs:string"/>
</xs:schema>
```

Entfernter Typ

Wenn das ursprüngliche XML-Schema einen globalen Typ enthält, der von einem anderen Typ abgeleitet ist, muss der globale Typ auch im neuen XML-Schema vorhanden sein.

Beispiel

Ursprüngliches XML-Schema:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="root" type="t1"/>
  <xs:complexType name="t1">
    <xs:complexContent>
```

```
      <xs:extension base="xs:anyType">
        <xs:attribute name="a" use="required"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="t2">
    <xs:complexContent>
      <xs:extension base="t1">
        <xs:attribute name="b" use="required"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType></xs:schema>
```

Neues XML-Schema:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="root" type="t1"/>
  <xs:complexType name="t1">
    <xs:complexContent>
      <xs:extension base="xs:anyType">
        <xs:attribute name="a" use="required"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:schema>
```

Einfacher und komplexer Inhalt

Ein komplexer Typ mit einfachem Inhalt im ursprünglichen XML-Schema darf im aktualisierten XML-Schema nicht für komplexen Inhalt neu definiert werden.

Beispiel

Ursprüngliches XML-Schema:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="root">
                <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:string">
          <xs:attribute name="a" type="xs:string"/>
        </xs:extension>
      </xs:simpleContent>    </xs:complexType>
  </xs:element>
</xs:schema>
```

Neues XML-Schema:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="root">
                <xs:complexType>
      <xs:complexContent base="xs:anyType">
        <xs:extension base="xs:anyType">
          <xs:attribute name="a" type="xs:string"/>
        </xs:extension>
      </xs:complexContent>    </xs:complexType>
  </xs:element>
</xs:schema>
```

Einfacher Inhalt

Einfache Typen, die im ursprünglichen XML-Schema und im neuen XML-Schema definiert sind, müssen die gleichen Basistypen verwenden.

Beispiel

Ursprüngliches XML-Schema:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="foo" >
       <xs:simpleType>
        <xs:restriction base="xs:decimal" />
        </xs:simpleType>
      </xs:element>
</xs:schema>
```

Neues XML-Schema:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="foo" >
       <xs:simpleType>
        <xs:restriction base="xs:string" />
        </xs:simpleType>
      </xs:element>
</xs:schema>
```

Szenario: Weiterentwicklung eines XML-Schemas

Das folgende Szenario zeigt, wie ein im XSR (XML-Schema-Repository) registriertes XML-Schema weiterentwickelt wird.

Jane, Managerin eines kleinen Geschäfts, verwaltet eine Datenbank, in der alle im Geschäft verkauften Produkte in einer Reihe von XML-Dokumenten aufgelistet sind. Diese XML-Produktlisten entsprechen dem folgenden Schema:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:complexType name="prodType">
      <xsd:sequence>
        <xsd:element name="name"    type="xsd:string" />
    <xsd:element name="sku" type="xsd:string" />
        <xsd:element name="price"   type="xsd:integer" />
      </xsd:sequence>
  <xsd:attribute name="color" type="xsd:string" />
    <xsd:attribute name="weight"  type="xsd:integer" />
  </xsd:complexType>

<xsd:element name="products">
  <xsd:complexType>
        <xsd:sequence>
      <xsd:element name="product" type="prodType" maxOccurs="unbounded" />
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>

</xsd:schema>
```

Das XML-Schema wurde anfänglich mithilfe der folgenden Befehle im XSR registriert:

```
REGISTER XMLSCHEMA 'http://product'
FROM 'file://c:/schemas/prod.xsd'
AS STORE.PROD

COMPLETE XMLSCHEMA STORE.PROD
```

Nach der Registrierung des XML-Schemas wurden die XML-formatierten Produktlisten anhand des Schemas auf Gültigkeit geprüft und in die Datenbank des Geschäfts eingefügt.

Jane beschließt, zusätzliche zu den Namen, Artikelnummern (SKU) und Preisen der Produkte jeweils auch eine Produktbeschreibung in die Listen aufzunehmen. Anstatt ein neues XML-Schema zu erstellen und alle vorhandenen XML-Dokumente anhand dieses Schemas erneut auf Gültigkeit zu prüfen, entscheidet sich Jane dafür, das ursprüngliche XML-Schema zu aktualisieren, um die zusätzlichen Beschreibungen hinzuzufügen. Hierzu muss dem ursprünglichen XML-Schema ein neues Element vom Typ 'description' (Beschreibung) wie folgt hinzugefügt werden:

```
<xsd:complexType name="prodType">
      <xsd:sequence>
        <xsd:element name="name"    type="xsd:string" />
    <xsd:element name="sku" type="xsd:string" />
        <xsd:element name="price"   type="xsd:integer" />
    <xsd:element name="description" type="xsd:string" minOccurs="0" />
      </xsd:sequence>
  <xsd:attribute name="color" type="xsd:string" />
    <xsd:attribute name="weight"  type="xsd:integer" />
</xsd:complexType>
```

In dem einzufügenden XML-Schemasegment wird das Attribut "minOccurs" (Mindestvorkommen) auf "0" gesetzt. Dies ist wichtig, da das Element 'description' andernfalls zu einem verbindlichen Element des Inhaltsmodells werden würde und alle vorhandenen XML-Dokumente, die anhand des ursprünglichen Schemas geprüft und in die Datenbanktabellen eingefügt wurden, nicht länger gültig wären. Um ein XML-Schema weiterentwickeln zu können, müssen die ursprüngliche und die neue Version des betreffenden Schemas miteinander kompatibel sein. Der Abschnitt *Kompatibilitätsanforderungen für das Weiterentwickeln von XML-Schemata* enthält ausführliche Informationen hierzu.

Vor der Aktualisierung muss das neue XML-Schema wie folgt im XSR registriert werden:

```
REGISTER XMLSCHEMA 'http://newproduct'
FROM 'file://c:/schemas/newprod.xsd'
AS STORE.NEWPROD

COMPLETE XMLSCHEMA STORE.NEWPROD
```

Als Nächstes führt Jane mithilfe der gespeicherten Prozedur XSR_UPDATE die Aktualisierung durch:

```
CALL SYSPROC.XSR_UPDATE(
  'STORE',
  'PROD',
  'STORE',
  'NEWPROD',
  1)
```

Das ursprüngliche XML-Schema wird weiterentwickelt. Sämtliche externen Abhängigkeiten, die im XSR für XML-Instanzdokumente verwaltet werden und zuvor anhand des XML-Schemas STORE.PROD auf Gültigkeit überprüft wurden, werden aufgrundlage des Inhalts des XML-Schemas STORE.NEWPROD aktualisiert. Da der Parameter *dropnewschema* durch Übergabe eines Wertes ungleich null gesetzt wird, wird das neue Schema STORE.NEWPROD nach der Aktualisierung des ursprünglichen Schemas gelöscht.

Alle vorhandenen XML-Dokumente, die bereits anhand des ursprünglichen XML-Schemas auf Gültigkeit geprüft wurden, werden nach der Aktualisierung nicht er-

neut geprüft. Stattdessen wird während der Aktualisierung eine Überprüfung durchgeführt, um festzustellen, ob das ursprüngliche XML-Schema mit dem neuen Schema kompatibel ist. Hierdurch wird sichergestellt, dass alle zuvor anhand des ursprünglichen XML-Schemas geprüften Dokumente auch gemäß dem neuen Schema gültig sind. Im vorherigen Beispiel ist es für die Kompatibilität der beiden XML-Schemata erforderlich, das Attribut "minOccurs" im neuen Element "description" auf den Wert "0" zu setzen. Alle XML-Dokumente, die nach der Weiterentwicklung des Schemas eingefügt werden, werden anhand der neuen, aktualisierten Version von STORE.PROD geprüft, und diese Dokumente können nun für jedes Produkt des Geschäfts ein Element vom Typ "description" enthalten.

Beispiele für die Extraktion von XML-Schemainformationen

Auflisten der im XSR registrierten XML-Schemata

Die folgenden Beispiele zeigen, wie vollständig im XML-Schema-Repository registrierte XML-Schemata mit SQL-Anweisungen abgefragt werden können. Bevor ein XML-Schema als vollständig registriert identifiziert werden kann, muss der Registrierungsprozess abgeschlossen sein.

Beispiel 1: Alle registrierten XML-Schemata auflisten

Dieses Beispiel gibt das SQL-Schema und die SQL-Kennung aller XML-Schemata zurück, die im XML-Schema-Repository (XSR) registriert sind.

```
SELECT OBJECTNAME, OBJECTSCHEMA
   FROM SYSCAT.XSROBJECTS
   WHERE OBJECTTYPE='S' AND STATUS='C'
```

Beispiel 2: Zielnamensbereich und Schemaposition zurückgeben

Dieses Beispiel gibt die URIs (Uniform Resource Identifiers) der Zielnamensbereiche und Schemapositionen für alle registrierten XML-Schemata (*targetNamespace* und *schemaLocation*) zurück.

```
SELECT TARGETNAMESPACE, SCHEMALOCATION
   FROM SYSCAT.XSROBJECTS
   WHERE OBJECTTYPE='S' AND STATUS='C'
```

Beispiel 3: Objektinformationsdokument zurückgeben

Dieses Beispiel gibt das Objektinformationsdokument für alle registrierten Schemata (*schemaInfo*) zurück. Dieses XML-Dokument wird während der Schemaregistrierung generiert und beschreibt jedes XML-Schemadokument, das zu einem im XML-Schema-Repository (XSR) registrierten XML-Schema gehört.

```
SELECT OBJECTINFO
   FROM SYSCAT.XSROBJECTS
   WHERE OBJECTTYPE='S' AND STATUS='C'
```

Abrufen aller Komponenten eines im XSR registrierten XML-Schemas

Die folgenden Beispiele zeigen, wie alle einzelnen XML-Schemadokumente, aus denen ein registriertes XML-Schema besteht, aus dem XML-Schema-Repository abgerufen werden können.

Beispiel 1: Zugehörige XML-Schemadokumente eines registrierten XML-Schemas zusammen mit dem Zielnamensbereich und der Schemaposition (*targetNamespace* und *schemaLocation*) zurückgeben:

```
SELECT COMPONENT, TARGETNAMESPACE, SCHEMALOCATION
   FROM SYSCAT.XSROBJECTCOMPONENTS
   WHERE OBJECTSCHEMA  = ? AND OBJECTNAME = ?
```

Die XML-Schemadokumente der Komponente werden als BLOB-Werte zurückgegeben.

Beispiel 2: XML-Schemadokument eines registrierten XML-Schemas mit dem Objektnamen CUSTOMER zurückgeben. Bei Ausführung für die Datenbank SAMPLE gibt die Anweisung das XML-Schemadokument zurück, das zur Prüfung der XML-Dokumente in der Spalte INFO der Tabelle CUSTOMER verwendet wird.

```
SELECT XMLPARSE(document COMPONENT) FROM SYSCAT.XSROBJECTCOMPONENTS
WHERE OBJECTNAME = 'CUSTOMER'
```

Das XML-Schemadokument wird als XML-Wert zurückgegeben.

Abrufen des XML-Schemas eines XML-Dokuments

Das folgende Beispiel zeigt, wie das XML-Schema, das einem XML-Dokument zugeordnet ist, aus dem XML-Schema-Repository abgerufen werden kann.

Beispiel 1: Objekt-ID des XML-Schemas eines XML-Dokuments abrufen:

```
SELECT DOC, XMLXSROBJECTID(DOC)
      FROM T
```

Beispiel 2: Objekt-ID und zweiteilige SQL-Kennung des XML-Schemas eines XML-Dokuments abrufen:

```
SELECT XMLXSROBJECTID(DOC),
   CAT.OBJECTSCHEMA, CAT.OBJECTNAME
   FROM T, SYSCAT.XSROBJECTS AS CAT
   WHERE XMLXSROBJECTID(DOC) = CAT.OBJECTID
```

Kapitel 9. Versetzen von XML-Daten

Unterstützung für das Versetzen von XML-Daten wird durch die Dienstprograme LOAD, IMPORT und EXPORT bereitgestellt. Unterstützung für das Versetzen von Tabellen, die XML-Spalten enthalten, ohne in den Offlinestatus zu wechseln, wird durch die gespeicherte Prozedur ADMIN_MOVE_TABLE bereitgestellt.

Importieren von XML-Daten

Das Dienstprogramm IMPORT kann zum Einfügen von XML-Dokumenten in eine reguläre, relationale Tabelle verwendet werden. Nur korrekt formatierte XML-Dokumente können importiert werden.

Verwenden Sie die Option XML FROM des Befehls IMPORT zum Angeben der Speicherposition der XML-Dokumente, die importiert werden sollen. Die Option XMLVALIDATE gibt an, wie die importierten Dokumente auf ihre Gültigkeit überprüft werden sollen. Sie können auswählen, ob die importierten XML-Daten auf der Basis des im Befehl IMPORT angegebenen Schemas, auf der Basis eines Schemapositionshinweises im XML-Quellendokument oder auf der Basis eines Schemas überprüft werden sollen, das von der XML-Datenkennung in der Hauptdatendatei angegeben wird. Sie können auch die Option XMLPARSE verwenden, um anzugeben, wie Leerzeichen beim Importieren des XML-Dokuments verarbeitet werden sollen. Die Dateitypänderungswerte `xmlchar` und `xmlgraphic` ermöglichen Ihnen die Angabe der Codierungsmerkmale für die importierten XML-Daten.

Laden von XML-Daten

Das Dienstprogramm LOAD bietet eine effiziente Methode zum Einfügen großer XML-Datenvolumina in eine Tabelle. Darüber hinaus können mit diesem Dienstprogramm auch bestimmte Optionen verwendet werden, die beim Dienstprogramm IMPORT nicht verfügbar sind, wie beispielsweise die Möglichkeit zum Laden von Daten aus einem benutzerdefinierten Cursor.

Ebenso wie beim Befehl IMPORT können Sie mit dem Befehl LOAD die Speicherposition der zu ladenden XML-Daten, die Prüfungsoptionen für die XML-Daten sowie die Art der Leerzeichenverarbeitung angeben. Und ebenso wie bei IMPORT können Sie die Dateitypänderungswerte `xmlchar` und `xmlgraphic` verwenden, um die Codierungsmerkmale für die geladenen XML-Daten anzugeben.

Exportieren von XML-Daten

Daten können aus Tabellen exportiert werden, die eine oder auch mehrere Spalten mit dem XML-Datentyp umfassen. Exportierte XML-Daten werden separat von der Hauptdatendatei gespeichert, die die exportierten relationalen Daten enthält. Informationen zu den einzelnen exportierten XML-Dokumenten werden in der Hauptdatei der exportierten Daten durch eine XML-Datenkennung (XDS) dargestellt. Die XML-Datenkennung besteht aus einer Zeichenfolge, in der der Name der Systemdatei angegeben ist, in der das XML-Dokument gespeichert ist. Darüber hinaus enthält diese Zeichenfolge auch die exakte Position und Länge des XML-Dokuments innerhalb dieser Datei sowie das XML-Schema, das zur Gültigkeitsprüfung des XML-Dokuments verwendet wird.

Sie können die Parameter XMLFILE, XML TO und XMLSAVESCHEMA des Befehls EXPORT verwenden, um detaillierte Informationen zur Speicherung exportierter XML-Dokumente anzugeben. Die Dateitypänderungswerte `xmlinsepfiles`, `xmlnodeclaration`, `xmlchar` und `xmlgraphic` ermöglichen Ihnen die Angabe weiterführender Details zur Speicherposition und zur Codierung der exportierten XML-Daten.

Versetzen von Tabellen im Onlinestatus

Mithilfe der gespeicherten Prozedur ADMIN_MOVE_TABLE werden die Daten in einer aktiven Tabelle in ein neues Tabellenobjekt mit demselben Namen versetzt, wobei die Daten online und im Zugriff bleiben. Die Tabelle kann eine oder auch mehrere Spalten mit dem XML-Datentyp enthalten. Versetzen Sie Tabellen online statt offline, wenn die Verfügbarkeit für Sie eine höhere Priorität hat als Kosten, Speicherplatz, Versetzungsleistung und Transaktionsaufwand.

Sie können die Prozedur einmal oder mehrmals aufrufen, wobei die Prozedur für jede Operation einen eigenen Aufruf ausführt. Wenn Sie mehrere Aufrufe verwenden, stehen Ihnen weitere Optionen zur Verfügung. Sie können beispielsweise das Versetzen abbrechen oder bestimmen, wann die Zieltabelle zur Aktualisierung in den Offlinestatus wechseln soll.

Versetzen von XML-Daten - zentrale Aspekte

Beim Importieren oder Exportieren von XML-Daten ist eine Reihe von Einschränkungen, Voraussetzungen und Erinnerungen zu berücksichtigen. Prüfen Sie diese Aspekte, bevor Sie XML-Daten importieren oder exportieren.

Berücksichtigen Sie die folgenden Aspekte, wenn Sie XML-Daten exportieren oder importieren:

- Exportierte XML-Daten werden immer separat von der Hauptdatendatei gespeichert, die die exportierten relationalen Daten enthält.
- Standardmäßig schreibt das Dienstprogramm EXPORT XML-Daten im Unicode-Format. Verwenden Sie den Dateitypänderungswert `xmlchar`, um XML-Daten in der Zeichencodepage zu schreiben, oder den Dateitypänderungswert `xmlgraphic`, um XML-Daten in UTF-16 (der grafischen Codepage) zu schreiben, unabhängig von der Anwendungscodepage.
- XML-Daten können in Nicht-Unicode-Datenbanken gespeichert werden und die in eine XML-Spalte einzufügenden Daten werden vor dem Einfügen aus der Datenbankcodepage in UTF-8 konvertiert. Um während des XML-Parsings die Einführung von Substitutionszeichen zu vermeiden, sollten einzufügende Zeichendaten ausschließlich aus Codepunkten bestehen, die Teil der Datenbankcodepage sind. Bei Einstellung des Konfigurationsparameters `enable_xmlchar` auf `no` wird das Einfügen von Zeichendatentypen während des XML-Parsings blockiert und so das Einfügen auf Datentypen beschränkt, die keiner Codepagekonvertierung unterzogen werden, beispielsweise BIT DATA, BLOB und XML.
- Beim Importieren oder Laden von XML-Daten wird davon ausgegangen, dass die XML-Daten im Unicode-Format codiert sind, sofern das zu importierende XML-Dokument keinen Deklarationstag enthält, der ein Codierungsattribut umfasst. Sie können den Dateitypänderungswert `xmlchar` verwenden, um anzugeben, dass die zu importierenden XML-Dokumente in der Zeichencodepage codiert werden, während der Dateitypänderungswert `xmlgraphic` angibt, dass die zu importierenden XML-Dokumente in UTF-16 codiert werden.
- Zeilen, die nicht korrekt formatierte Dokumente enthalten, werden von den Dienstprogrammen IMPORT und LOAD zurückgewiesen.

- Wenn für die Dienstprogramme IMPORT und LOAD die Option XMLVALIDATE angegeben wird, werden Dokumente, die erfolgreich anhand ihres übereinstimmenden Schemas überprüft werden konnten, während des Einfügens in eine Tabelle mit Annotationen versehen, die die Schemadaten enthalten. Zeilen, die Dokumente enthalten, deren Gültigkeitsprüfung auf der Basis des zugeordneten Schemas fehlgeschlagen ist, werden zurückgewiesen.
- Wenn für ein Dienstprogramm LOAD oder IMPORT die Option XMLVALIDATE angegeben wird und mehrere XML-Schemata zur Prüfung von XML-Dokumenten verwendet werden, müssen Sie unter Umständen den Konfigurationsparameter **catalogcache_sz** für die Katalogcachegröße erhöhen. Ist die Erhöhung von **catalogcache_sz** nicht möglich, können Sie den integrierten Import- oder Ladebefehl in mehrere Befehle aufteilen, die weniger Schemadokumente verwenden.
- Wenn Sie beim Exportieren von XML-Daten eine XQuery-Anweisung angeben, können Sie XDM-Instanzen (XDM = XQuery and XPath Data Model; XQuery- und XPath-Datenmodell) exportieren, bei denen es sich nicht um korrekt formatierte XML-Dokumente handelt. Exportierte XML-Dokumente, die nicht korrekt formatiert sind, können nicht direkt in eine XML-Spalte importiert werden, weil Spalten, für die der XML-Datentyp definiert wurde, nur vollständige, korrekt formatierte XML-Dokumente enthalten dürfen.
- Die Einstellung **CPU_PARALLELISM** wird während einer Ladeoperation auf den Wert 1 reduziert, wenn Statistikdaten erfasst werden.
- Für eine XML-Ladeoperation muss gemeinsam genutzter Sortierspeicher verwendet werden, um weiterarbeiten zu können. Aktivieren Sie den Parameter **SHEAPTHRES_SHR** oder **INTRA_PARALLEL** oder den Verbindungskonzentrator. Der Parameter **SHEAPTHRES_SHR** wird standardmäßig eingestellt, damit gemeinsam genutzter Sortierspeicher für die Standardkonfiguration verfügbar ist.
- Sie können die Option **SOURCEUSEREXIT** oder den Parameter **SAVECOUNT** des Befehls LOAD nicht angeben, wenn Sie eine Tabelle laden, die eine XML-Spalte enthält.
- Ebenso wie LOB-Dateien müssen auch XML-Dateien bei der Verwendung des Befehls LOAD auf der Serverseite gespeichert werden.
- Wenn Sie XML-Daten in einer Umgebung mit partitionierten Datenbanken auf mehrere Datenbankpartitionen laden, müssen alle Datenbankpartitionen Zugriff auf die Dateien haben, die die XML-Daten enthalten. Sie können den Zugriff auf die Dateien beispielsweise ermöglichen, indem Sie die Dateien kopieren oder einen NFS-Mount erstellen.

Abfrage- und XPath-Datenmodell

Auf XML-Daten in Tabellen können Sie entweder mithilfe der XQuery-Funktionen, die in SQL zur Verfügung stehen, oder durch direktes Aufrufen von XQuery zugreifen. Bei einer Instanz des XQuery- und XPath-Datenmodells (XDM) kann es sich um korrekt formatierte XML-Dokumente, um Folgen von Knoten oder atomaren Werten oder um beliebige Kombination aus Knoten und atomaren Werten handeln.

Einzelne XDM-Instanzen können mithilfe des Befehls EXPORT in eine XML-Datei oder auch in mehrere XML-Dateien geschrieben werden.

Verhalten von LOB- und XML-Dateien hinsichtlich IMPORT und EXPORT

LOB- und XML-Dateien weisen im Hinblick auf ihr Verhalten und die Kompatibilität bestimmte Gemeinsamkeiten auf, die beim Importieren (IMPORT) und Exportieren (EXPORT) von Daten genutzt werden können.

Export Wenn beim Exportieren von Daten mit der Option LOBS TO mindestens ein LOB-Pfad angegeben wird, dann schreibt das Dienstprogramm EXPORT reihum die aufeinander folgenden LOB-Werte in die entsprechenden LOB-Dateien. Wird mit der Option XML TO mindestens ein XML-Pfad angegeben, dann schreibt das Dienstprogramm EXPORT ebenfalls reihum alle aufeinander folgenden XDM-Instanzen (XDM = XQuery- und XPath-Datenmodell) in die entsprechenden XML-Dateien. Standardmäßig werden LOB-Werte und XDM-Instanzen in den Pfad geschrieben, in dem auch die exportierten relationalen Daten abgelegt werden. Sofern nicht der Dateitypmodifikator OBSINSEPFILES oder XMLINSEPFILES definiert wurde, sind sowohl für LOB- als auch für XML-Dateien mehrere verknüpfte Werte für dieselbe Datei zulässig.

Die Option LOBFILE bietet eine Möglichkeit zur Angabe des Basisdateinamens der LOB-Dateien, die vom Dienstprogramm EXPORT erstellt wurden. Ihn ähnlicher Weise bietet die Option XMLFILE die Möglichkeit, den Basisdateinamen der XML-Dateien anzugeben, die vom Dienstprogramm EXPORT generiert wurden. Der Standardbasisdateiname der LOB-Datei stimmt mit dem Namen der exportierten Datendatei überein, verfügt jedoch über die Erweiterung `.lob`. Der Standardbasisdateiname der XML-Datei stimmt mit dem Namen der exportierten Datendatei überein, verfügt jedoch über die Erweiterung `.xml`. Der vollständige Name der exportierten LOB- oder XML-Datei besteht deshalb aus dem Basisdateinamen und einer numerischen Erweiterung, die auf drei Ziffern aufgefüllt wird, sowie der Erweiterung `.lob` oder `.xml`.

Import

Beim Importieren von Daten ist ein LLS (LOB Location Specifier) mit einer XML-Zielspalte und eine XML-Datenkennung (XDS) mit einer LOB-Zielspalte kompatibel. Wird die Option LOBS FROM nicht angegeben, dann geht das System davon aus, dass die zu importierenden LOB-Dateien sich im selben Pfad wie die relationale Eingabedatendatei befinden. Wenn die Option XML FROM nicht angegeben wurde, geht das System davon aus, dass die zu importierenden XML-Dateien sich im selben Pfad wie die relationalen Eingabedatendatei befinden.

Beispiele zum Exportieren

In dem folgenden Beispiel werden alle LOB-Werte in die Datei `/mypath/t1export.del.001.lob` geschrieben, und alle XDM-Instanzen werden in die Datei `/mypath/t1export.del.001.xml` geschrieben:

```
EXPORT TO /mypath/t1export.del OF DEL MODIFIED BY LOBSINFILE
SELECT * FROM USER.T1
```

In dem folgenden Beispiel wird der erste LOB-Wert in die Datei `/lob1/t1export.del.001.lob` und der zweite in die Datei `/lob2/t1export.del.002.lob` geschrieben, während der dritte an die Datei `/lob1/t1export.del.001.lob` und der vierte an die Datei `/lob2/t1export.del.002.lob` angehängt wird usw.:

```
EXPORT TO /mypath/t1export.del OF DEL LOBS TO /lob1,/lob2
MODIFIED BY LOBSINFILE SELECT * FROM USER.T1
```

In dem folgenden Beispiel wird die erste XDM-Instanz in die Datei /xml1/xmlbase.001.xml, die zweite in die Datei /xml2/xmlbase.002.xml, die dritte in die Datei /xml1/xmlbase.003.xml und die vierte in die Datei /xml2/xmlbase.004.xml geschrieben usw.:

```
EXPORT TO /mypath/t1export.del OF DEL XML TO /xml1,/xml2 XMLFILE xmlbase
      MODIFIED BY XMLINSEPFILES SELECT * FROM USER.T1
```

Beispiele zum Importieren

Für eine Tabelle mit dem Namen "mytable", die eine einzige XML-Spalte enthält, und den unten aufgeführten Befehl IMPORT gilt Folgendes:

```
IMPORT FROM myfile.del of del LOBS FROM /lobpath XML FROM /xmlpath
MODIFIED BY LOBSINFILE XMLCHAR replace into mytable
```

Wenn "myfile.del" die folgenden Daten enthält:

```
mylobfile.001.lob.123.456/
```

Das Dienstprogramm IMPORT versucht in diesem Fall, ein XML-Dokument aus der Datei /lobpath/mylobfile.001.lob zu importieren. Hierbei wird an der relativen Dateiposition (Offset) 123 begonnen und mit einer Länge von 456 Byte gearbeitet.

Es wird davon ausgegangen, dass sich die Datei "mylobfile.001.lob" in LOB-Pfad und nicht im XML-Pfad befindet, da auf den Wert mithilfe eines LLS (LOB Location Specifier) und nicht mit einer XML-Datenkennung verwiesen wird.

Außerdem geht das System davon aus, dass das Dokument in der Zeichencodepage codiert ist, da der Dateitypmodifikator XMLCHAR angegeben wurde.

XML-Datenkennung

XML-Daten, die mit den Dienstprogrammen EXPORT, IMPORT und LOAD versetzt werden, müssen in Dateien gespeichert werden, die separat von der Hauptdatendatei abgelegt sind. In der Hauptdatendatei werden die XML-Daten durch eine XML-Datenkennung (XDS = XML Data Specifier) dargestellt.

Die XML-Datenkennung besteht aus einer Zeichenfolge, die durch einen XML-Tag mit dem Namen "XDS" dargestellt wird, und verfügt über Attribute, die Informationen zu den eigentlichen XML-Daten in der Spalte enthalten. Hierzu gehören z. B. Angaben zum Namen der Datei, in der die eigentlichen XML-Daten enthalten sind, sowie Angaben zur relativen Position und Länge der XML-Daten in dieser Datei. Die Attribute der XML-Datenkennung werden in der folgenden Liste beschrieben.

FIL Der Name der Datei (File), die die XML-Daten enthält. Sie können keine benannte Pipe angeben. Das Importieren oder Laden von XML-Dokumenten aus einer benannten Pipe wird nicht unterstützt.

OFF Die relative Byteadresse (Offset) der XML-Daten in der Datei, die im Attribut FIL angegeben wurde. Hierbei beginnt die relative Adresse bei 0.

LEN Die Länge (Length) der XML-Daten, die sich in der im Attribut FIL angegebenen Datei befinden, in Byte.

SCH Die vollständig qualifizierte SQL-Kennung des XML-Schemas, das zur Gültigkeitsprüfung dieses XML-Dokuments verwendet wird. Die Schema- und

Namenskomponenten der SQL-Kennung werden als Werte für "OBJECTSCHEMA" und "OBJECTNAME" der Zeile in der Katalogtabelle SYSCAT.XSROBJECTS gespeichert, die diesem XML-Schema zugeordnet ist.

Die XML-Datenkennung (XDS) wird als Zeichenfeld in der Datendatei interpretiert und unterliegt dem Parsingverhalten für Zeichenspalten des Dateiformats. Beim ASCII-Dateiformat mit Begrenzern (DEL) muss der Zeichenbegrenzer z. B. verdoppelt werden, sofern dieser in der XML-Datenkennung vorhanden ist. Die Sonderzeichen (<, >, &, ', ") in den Attributwerten müssen immer mit einem Escapezeichen versehen werden. Objektnamen, bei denen die Groß-/Kleinschreibung zu beachten ist, müssen zwischen Zeicheneinheiten vom Typ " eingeschlossen werden.

Beispiele

Gehen Sie von einem Attribut vom Typ FIL mit dem Wert `abc&"def".del` aus. Um diese XMS-Datenkennung in eine ASCII-Datei mit Begrenzern einzufügen, wobei als Zeichenbegrenzer das doppelte Anführungszeichen (") verwendet wird, müssen die doppelten Anführungszeichen (") verdoppelt und Sonderzeichen mit einem Escapezeichen versehen werden:

```
<XDS FIL=""abc&amp;&quot;def&quot;.del"" />
```

Das nachstehende Beispiel zeigt eine XML-Datenkennung (XDS), die in einer ASCII-Datendatei mit Begrenzern angegeben sein kann. XML-Daten werden in der Datei `xmldocs.xml.001` gespeichert, wobei bei der relativen Byteadresse 100 begonnen und eine Länge von 300 Byte verwendet wird. Da sich diese XML-Datenkennung in einer ASCII-Datei befindet, bei der als Begrenzer doppelte Anführungszeichen verwendet werden, müssen die doppelten Anführungszeichen innerhalb des XDS-Tags verdoppelt werden.

```
"<XDS FIL = ""xmldocs.xml.001"" OFF=""100"" LEN=""300"" />"
```

Das nachstehende Beispiel zeigt die vollständig qualifizierte SQL-Kennung `ANTHONY.purchaseOrderTest`. Der Teil der Kennung, bei dem die Groß-/Kleinschreibung zu beachten ist, muss in der XML-Datenkennung zwischen Zeicheneinheiten vom Typ " eingeschlossen werden:

```
"<XDS FIL='/home/db2inst1/xmlload/a.xml' OFF='0' LEN='6758'
   SCH='ANTHONY.&quot;purchaseOrderTest&quot;' />"
```

Exportieren von XML-Daten

Beim Exportieren von XML-Daten werden die dabei generierten XDM-Instanzen (XDM = XQuery Data Model) in eine oder mehrere Dateien geschrieben, die separat von der Hauptdatendatei abgelegt werden, die die exportierten relationalen Daten enthält. Dies gilt auch dann, wenn weder die Option XMLFILE noch die Option XML TO angegeben ist.

Standardmäßig werden alle exportierten QDM-Instanzen mit derselben XML-Datei verknüpft. Mit dem Dateitypmodifikator `XMLINSEPFILES` können Sie angeben, dass alle QDM-Instanzen in eine separate Datei geschrieben werden sollen.

Die XML-Daten werden jedoch in der Hauptdatendatei durch eine XML-Datenkennung (XDS = XML Data Specifier) dargestellt. Die XML-Datenkennung besteht aus einer Zeichenfolge, die durch einen XML-Tag mit dem Namen "XDS" dargestellt wird, und verfügt über Attribute, die Informationen zu den eigentlichen XML-Daten in der Spalte enthalten. Hierzu gehören z. B. Angaben zum Namen der Datei,

in der die eigentlichen XML-Daten enthalten sind, sowie Angaben zur relativen Position und Länge der XML-Daten in dieser Datei.

Die Zielpfade und Basisdateinamen der exportierten XML-Dateien können mit den Optionen XML TO und XMLFILE angegeben werden. Bei Angabe der Option XML TO oder XMLFILE lautet das Namensformat der exportierten und im Attribut FIL der XDS gespeicherten XML-Dateien `xmlfilespec.xxx.xml`, wobei `xmlfilespec` für den für die Option XMLFILE angegebenen Wert und `xxx` für eine vom Dienstprogramm EXPORT generierte Folgenummer für XML-Dateien steht. Ansonsten lautet das Namensformat der exportierten XML-Dateien `exportfilename.xxx.xml`, wobei `exportfilename` für den Namen der im Befehl EXPORT angegebenen exportierten Ausgabedatei und `xxx` für eine vom Dienstprogramm EXPORT generierte Folgenummer für XML-Dateien steht.

Standardmäßig werden exportierte XML-Dateien in den Pfad geschrieben, in dem sich auch die exportierte Datendatei befindet. Der Standardwert für den Basisdateinamen stimmt bei exportierten XML-Dateien mit dem Namen der exportierten Datendatei überein, wobei eine aus drei Ziffern bestehende Folgenummer und die Erweiterung `.xml` angefügt werden.

Beispiele

Gehen Sie in den folgenden Beispielen von einer Tabelle USER.T1 aus, die vier Spalten und zwei Zeilen enthält:

```
C1 INTEGER
   C2 XML
   C3 VARCHAR(10)
   C4 XML
```

Tabelle 27. USER.T1

C1	C2	C3	C4
2	<?xml version="1.0" encoding="UTF-8" ?><note time="12:00:00"><to>You</to><from> Me</from><heading>note1</heading><body>Hello World!</body></note>	'char1'	<?xml version="1.0" encoding="UTF-8" ?><note time="13:00:00"><to>Him</to><from> Her</from><heading>note2</heading><body>Hello World!</body></note>
4	NULL	'char2'	?xml version="1.0" encoding="UTF-8" ?><note time="14:00:00">to>Us</to><from> Them</from><heading>note3</heading><body>Hello World!</body></note>

Beispiel 1

Mit dem folgenden Befehl können Sie den Inhalt von USER.T1 im ASCII-Format mit Begrenzern (DEL) in die Datei "/mypath/t1export.del" exportieren. Da die Optionen XML TO und XMLFILE nicht angegeben wurden, werden die in den Spalten C2 und C4 enthaltenen XML-Dokumente in denselben Pfad geschrieben wie die exportierte Hauptdatei "/mypath". Der Basisdateiname für diese Dateien lautet "t1export.del.xml". Die Option XMLSAVESCHEMA gibt an, dass die XML-Schemainformationen während der Exportprozedur gespeichert werden.

```
EXPORT TO /mypath/t1export.del OF DEL XMLSAVESCHEMA SELECT * FROM USER.T1
```

Die exportierte Datei "/mypath/t1export.del" enthält Folgendes:

```
   2,"<XDS FIL='t1export.del.001.xml' OFF='0' LEN='144' />","char1",
   "<XDS FIL='t1export.del.001.xml' OFF='144' LEN='145' />"
   4,,"char2","<XDS FIL='t1export.del.001.xml' OFF='289'
LEN='145' SCH='S1.SCHEMA_A' />"
```

Die exportierte XML-Datei "/mypath/t1export.del.001.xml" enthält Folgendes:

```
<?xml version="1.0" encoding="UTF-8" ?><note time="12:00:00"><to>You</to>
         <from>Me</from><heading>note1</heading><body>Hello World!</body>
         </note><?xml version="1.0" encoding="UTF-8" ?><note time="13:00:00"><to>Him
         </to><from>Her</from><heading>note2</heading><body>Hello World!
         </body></note><?xml version="1.0" encoding="UTF-8" ?><note time="14:00:00">
   <to>Us</to><from>Them</from>heading>note3</heading><body>
   Hello World!</body></note>
```

Beispiel 2

Mit dem folgenden Befehl können Sie den Inhalt von USER.T1 im ASCII-Format mit Begrenzern (DEL) in die Datei "t1export.del" exportieren. XML-Dokumente, die in den Spalten C2 und C4 enthalten sind, werden in den Pfad "/home/user/xmlpath" geschrieben. Den XML-Dateien wird der Basisdateiname "xmldocs" zugeordnet, und es werden mehrere exportierte XML-Dokumente in dieselbe XML-Datei geschrieben. Die Option XMLSAVESCHEMA gibt an, dass die XML-Schemainformationen während der Exportprozedur gespeichert werden.

```
EXPORT TO /mypath/t1export.del OF DEL XML TO /home/user/xmlpath
XMLFILE xmldocs XMLSAVESCHEMA SELECT * FROM USER.T1
```

Die exportierte DEL-Datei "/home/user/t1export.del" enthält Folgendes:

```
   2,"<XDS FIL='xmldocs.001.xml' OFF='0' LEN='144' />","char1",
   "<XDS FIL='xmldocs.001.xml' OFF='144' LEN='145' />"
   4,,"char2","<XDS FIL='xmldocs.001.xml' OFF='289'
LEN='145' SCH='S1.SCHEMA_A' />"
```

Die exportierte XML-Datei "/home/user/xmlpath/xmldocs.001.xml" enthält Folgendes:

```
<?xml version="1.0" encoding="UTF-8" ?><note time="12:00:00"><to>You</to>
         <from>Me</from><heading>note1</heading><body>Hello World!</body>
         </note><?xml version="1.0" encoding="UTF-8" ?><note time="13:00:00">
         <to>Him</to><from>Her</from><heading>note2</heading><body>
         Hello World!</body></note><?xml version="1.0" encoding="UTF-8" ?>
         <note time="14:00:00"><to>Us</to><from>Them</from><heading>
   note3</heading><body>Hello World!</body></note>
```

Beispiel 3

Der folgende Befehl ist mit dem Befehl aus Beispiel 2 vergleichbar. Eine Ausnahme bildet hierbei allerdings die Tatsache, dass alle exportierten XML-Dokumente in eine separate XML-Datei geschrieben werden.

```
EXPORT TO /mypath/t1export.del OF DEL XML TO /home/user/xmlpath
XMLFILE xmldocs MODIFIED BY XMLINSEPFILES XMLSAVESCHEMA
SELECT * FROM USER.T1
```

Die exportierte Datei "/mypath/t1export.del" enthält Folgendes:

```
2,"<XDS FIL='xmldocs.001.xml' />","char1","XDS FIL='xmldocs.002.xml' />"
   4,,"char2","<XDS FIL='xmldocs.004.xml' SCH='S1.SCHEMA_A' />"
```

Die exportierte XML-Datei "/home/user/xmlpath/xmldocs.001.xml" enthält Folgendes:

```
<?xml version="1.0" encoding="UTF-8" ?><note time="12:00:00"><to>You</to>
         <from>Me</from><heading>note1</heading><body>Hello World!</body>
         </note>
```

Die exportierte XML-Datei "/home/user/xmlpath/xmldocs.002.xml" enthält Folgendes:

```
?xml version="1.0" encoding="UTF-8" ?>note time="13:00:00">to>Him/to>
   from>Her/from>heading>note2/heading>body>Hello World!/body>
   /note>
```

Die exportierte XML-Datei "/home/user/xmlpath/xmldocs.004.xml" enthält Folgendes:

```
<?xml version="1.0" encoding="UTF-8" ?><note time="14:00:00"><to>Us</to>
         <from>Them</from><heading>note3</heading><body>Hello World!</body>
         </note>
```

Beispiel 4

Mit dem folgenden Befehl können Sie die Ergebnisse einer XQuery-Operation in eine XML-Datei schreiben.

```
EXPORT TO /mypath/t1export.del OF DEL XML TO /home/user/xmlpath
XMLFILE xmldocs MODIFIED BY XMLNODECLARATION select
xmlquery( '$m/note/from/text()' passing by ref c4 as "m" returning sequence)
      from USER.T1
```

Die exportierte DEL-Datei "/mypath/t1export.del" enthält Folgendes:

```
   "<XDS FIL='xmldocs.001.xml' OFF='0' LEN='3' />"
"<XDS FIL='xmldocs.001.xml' OFF='3' LEN='4' />"
```

Die exportierte XML-Datei "/home/user/xmlpath/xmldocs.001.xml" enthält Folgendes:

```
HerThem
```

Anmerkung: Das Ergebnis dieser speziellen XQuery-Operation generiert keine korrekt formatierten XML-Dokumente. Aus diesem Grund kann die in diesem Beispiel exportierte Datei nicht direkt in eine XML-Spalte importiert werden.

Importieren von XML-Daten

Das Dienstprogramm IMPORT kann zum Importieren von XML-Daten in eine XML-Tabellenspalte eingesetzt werden, indem entweder der Tabellenname oder ein Kurzname für ein Quellendatenobjekt von DB2 Database for Linux, UNIX and Windows verwendet wird.

Beim Importieren von Daten in eine XML-Tabellenspalte können Sie die Option XML FROM verwenden, um die Pfade der XML-Eingabedatei oder der XML-Eingabedateien anzugeben. Für die XML-Datei "/home/user/xmlpath/xmldocs.001.xml", die zuvor exportiert wurde, können Sie beispielsweise den folgenden Befehl verwenden, um die Daten zurück in die Tabelle zu importieren:

```
IMPORT FROM t1export.del OF DEL XML FROM /home/user/xmlpath INSERT INTO USER.T1
```

Eingefügte Dokumente anhand von Schemata prüfen

Die Option XMLVALIDATE ermöglicht die Gültigkeitsprüfung von XML-Dokumenten mithilfe von XML-Schemata, während diese importiert werden. Im folgen-

den Beispiel wird die Gültigkeit eingehender XML-Dokumente auf der Basis der Schemainformationen überprüft, die während des Exports der XML-Dokumente gespeichert wurden:

```
IMPORT FROM t1export.del OF DEL XML FROM /home/user/xmlpath XMLVALIDATE
            USING XDS INSERT INTO USER.T1
```

Optionen für das Parsing angeben

Mit der Option XMLPARSE können Sie angeben, ob Leerzeichen in den importierten XML-Dokumenten beim Parsing (d. h. bei der syntaktischen Analyse) beibehalten oder gelöscht werden sollen. Im folgenden Beispiel wird die Gültigkeit aller importierten XML-Dokumente auf der Basis der XML-Schemainformationen überprüft, die gespeichert wurden, als die XML-Dokumente exportiert wurden. Für diese Dokumente werden die Leerzeichen beim Parsing beibehalten.

```
IMPORT FROM t1export.del OF DEL XML FROM /home/user/xmlpath XMLPARSE PRESERVE
WHITESPACE XMLVALIDATE USING XDS INSERT INTO USER.T1
```

Laden von XML-Daten

Mit dem Dienstprogramm LOAD können große XML-Datenmengen effizient in Tabellen versetzt werden.

Beim Laden von Daten in eine XML-Tabellenspalte können Sie die Option XML FROM verwenden, um die Pfade der XML-Eingabedatei oder der XML-Eingabedateien anzugeben. Um beispielsweise Daten aus der Datei `/home/user/xmlpath/xmlfile1.xml` zu laden, könnten Sie den folgenden Befehl verwenden:

```
LOAD FROM data1.del OF DEL XML FROM /home/user/xmlpath INSERT INTO USER.T1
```

Die ASCII-Eingabedatei mit Begrenzern `data1.del` enthält eine XML-Datenkennung (XDS), die die Speicherposition der zu ladenden XML-Daten angibt. Die folgende XDS beispielsweise beschreibt ein XML-Dokument an der relativen Adresse (Offset) bei 123 Byte in der Datei `xmldata.ext` mit einer Länge von 456 Byte:

```
<XDS FIL='xmldata.ext' OFF='123' LEN='456' />
```

Das Laden von XML-Daten mithilfe eines deklarierten Cursors wird unterstützt. Im folgenden Beispiel wird ein Cursor deklariert und mit dem Befehl **LOAD** verwendet, um Daten aus der Tabelle CUSTOMERS in der Tabelle LEVEL1_CUSTOMERS hinzuzufügen:

```
DECLARE cursor_income_level1 CURSOR FOR
   SELECT * FROM customers
   WHERE XMLEXISTS('$DOC/customer[income_level=1]');

LOAD FROM cursor_income_level1 OF CURSOR INSERT INTO level1_customers;
```

Der Dateitypänderungswert `ANYORDER` des Befehls **LOAD** wird beim Laden von XML-Daten in eine XML-Spalte unterstützt.

Während des Ladevorgangs werden für Spalten des Typs XML keine Verteilungsstatistikdaten erfasst.

Laden von XML-Daten in einer Umgebung mit partitionierten Datenbanken

Bei Tabellen, die über Datenbankpartitionen verteilt sind, können Sie XML-Daten aus XML-Datendateien parallel in die Tabellen laden. Beim Laden von XML-Daten in Tabellen müssen die XML-Datendateien über Lesezugriff auf alle Datenbankpar-

titionen verfügen, in denen das Laden stattfindet.

Eingefügte Dokumente anhand von Schemata prüfen

Die Option XMLVALIDATE ermöglicht die Gültigkeitsprüfung von XML-Dokumenten mithilfe von XML-Schemata während des Ladens. Im folgenden Beispiel wird die Gültigkeit eingehender XML-Dokumente auf der Basis des Schemas überprüft, das durch die XDS in der ASCII-Eingabedatei mit Begrenzern `data2.del` identifiziert wird:

```
LOAD FROM data2.del OF DEL XML FROM /home/user/xmlpath XMLVALIDATE
     USING XDS INSERT INTO USER.T2
```

In diesem Fall enthält die XDS ein Attribut vom Typ SCH mit der vollständig qualifizierten SQL-Kennung des für die Gültigkeitsprüfung zu verwendenden XML-Schemas "S1.SCHEMA_A":

```
<XDS FIL='xmldata.ext' OFF='123' LEN='456' SCH='S1.SCHEMA_A' />
```

Optionen für das Parsing angeben

Mit der Option XMLPARSE können Sie angeben, ob Leerzeichen in den geladenen XML-Dokumenten beim Parsing (d. h. bei der syntaktischen Analyse) beibehalten oder gelöscht werden sollen. Im folgenden Beispiel wird die Gültigkeit aller geladenen XML-Dokumente auf der Basis des Schemas mit der SQL-Kennung "S2.SCHEMA_A" überprüft, und für diese Dokumente werden die Leerzeichen beim Parsing beibehalten:

```
LOAD FROM data2.del OF DEL XML FROM /home/user/xmlpath XMLPARSE PRESERVE
WHITESPACE XMLVALIDATE USING SCHEMA S2.SCHEMA_A INSERT INTO USER.T1
```

Indexierungsfehler beim Laden von XML-Daten beheben

Probleme mit Ladeoperationen, die aufgrund von Indexierungsfehlern fehlschlagen, können behoben werden, indem die Protokolldatei **db2diag** zusammen mit dem Dienstprogramm IMPORT verwendet wird, um die Problemwerte in den XML-Daten zu identifizieren und zu korrigieren.

Informationen zu diesem Vorgang

Wenn eine Ladeoperation die Fehlernachricht SQL20305N (sqlcode -20305) zurückgibt, bedeutet dies, dass mindestens ein XML-Knotenwert nicht indexiert werden konnte. Die Fehlernachricht gibt den jeweiligen Ursachencode für den Fehler zurück. Geben Sie `? SQL20305N` in den Befehlszeilenprozessor ein, um die Erläuterung und Benutzeraktion für den entsprechenden Ursachencode anzuzeigen.

Bei Indexierungsfehlern, die bei Einfügeoperationen auftreten, wird eine generierte XQuery-Anweisung als Ausgabe in die Protokolldatei **db2diag** geschrieben, um Sie beim Lokalisieren der fehlgeschlagenen XML-Knotenwerte im Dokument zu unterstützen. Ausführliche Informationen dazu, wie die XQuery-Anweisung verwendet wird, um die fehlerhaften XML-Knotenwerte zu lokalisieren, finden Sie in "Allgemeine Aspekte der XML-Indexierung".

Bei Indexierungsfehlern, die bei Ladeoperationen auftreten, werden die generierten XQuery-Anweisungen jedoch nicht in die Protokolldatei **db2diag** geschrieben. Um diese XQuery-Anweisungen zu generieren, muss das Dienstprogramm IMPORT für die fehlgeschlagenen, nicht geladenen Zeilen ausgeführt werden. Da die zurückgewiesenen Zeilen in der Tabelle nicht vorhanden sind, können die XQuery-Anweisungen nicht für die fehlgeschlagenen Dokumente ausgeführt werden. Um dieses

Problem zu lösen, müssen Sie eine neue Tabelle mit derselben Definition ohne Indizes erstellen. Die fehlgeschlagenen Zeilen können dann in die neue Tabelle geladen werden, und die XQuery-Anweisungen können anschließend für die neue Tabelle ausgeführt werden, um die fehlerhaften XML-Knotenwerte in den Dokumenten zu lokalisieren.

Um die Indexierungsfehler zu beheben, führen Sie die folgenden Schritte aus:

Vorgehensweise

1. Ermitteln Sie anhand der Satznummern in den Ausgabeinformationen, welche Zeilen während der Ladeoperation zurückgewiesen wurden.
2. Erstellen Sie eine Datei mit der Erweiterung '.del', die ausschließlich die zurückgewiesenen Zeilen enthält.
3. Erstellen Sie eine neue Tabelle (beispielsweise T2) mit denselben Spalten wie in der Originaltabelle (T1). Erstellen Sie für die neue Tabelle keine Indizes.
4. Laden Sie die zurückgewiesenen Zeilen in die neue Tabelle (T2).
5. Führen Sie für jede zurückgewiesene Zeile in der Originaltabelle T1 Folgendes aus:
   a. Importieren Sie die zurückgewiesenen Zeilen in T1, um die Fehlernachricht SQL20305N zu erhalten. Die Importoperation stoppt beim ersten Fehler, der auftritt.
   b. Überprüfen Sie die Protokolldatei **db2diag** und suchen Sie die generierte XQuery-Anweisung. Um die fehlerhaften Knotenwerte im Eingabedokument zu ermitteln, suchen Sie nach der Zeichenfolge 'SQL20305N' in der Protokolldatei **db2diag** und vergleichen Sie den Ursachencode. Im Anschluss an den Ursachencode finden Sie eine Reihe von Anweisungen und danach eine generierte XQuery-Anweisung, die Sie verwenden können, um in dem Dokument denjenigen Problemwert ausfindig zu machen, der den Fehler verursacht hat.
   c. Ändern Sie die XQuery-Anweisung entsprechend, um die neue Tabelle T2 zu verwenden.
   d. Führen Sie die XQuery-Anweisung für T2 aus, um den Problemwert im Dokument zu lokalisieren.
   e. Korrigieren Sie den Problemwert in der .xml-Datei, die das Dokument enthält.
   f. Wiederholen Sie Schritt a, und importieren Sie die zurückgewiesenen Zeilen erneut in T1. Die Zeile, die zuvor zum Abbruch der Importoperation geführt hatte, sollte jetzt erfolgreich eingefügt werden. Liegt in der .del-Datei eine weitere zurückgewiesene Zeile vor, stoppt das Dienstprogramm IMPORT beim nächsten Fehler und gibt eine weitere Nachricht vom Typ SQL20305N aus. Wiederholen Sie die beschriebenen Schritte, bis die Importoperation erfolgreich ausgeführt wird.

Beispiel

In nachstehendem Beispiel wurde der Index 'BirthdateIndex' mit dem Datentyp *date* für Datumsangaben erstellt. Die Option REJECT INVALID VALUES ist angegeben, sodass sämtliche XML-Musterwerte von */Person/Confidential/Birthdate* für den Datentyp *date* gültig sein müssen. Sobald ein XML-Musterwert nicht in diesen Datentyp umgesetzt werden kann, wird ein Fehler zurückgegeben.

Bei Verwendung der folgenden XML-Dokumente sollen eigentlich fünf Zeilen geladen werden. Die erste und vierte Zeile werden jedoch zurückgewiesen, weil die

Werte für das Geburtsdatum (Birthdate) nicht indexiert werden können. In der Datei `person1.xml` liegt der Wert `March 16, 2002` nicht im korrekten Datumsformat vor. In der Datei `person4.xml` weist der Wert `20000-12-09` in der Jahresangabe eine zusätzliche Null auf. So handelt es sich zwar um einen gültigen XML-Datumswert, der jedoch außerhalb des Bereichs liegt, der in DB2 für Jahresangaben zulässig ist (0001 bis 9999). Ein Teil der Beispielausgabe wurde bearbeitet, um das Beispiel kürzer zu gestalten.

Die folgenden fünf XML-Dateien sollen geladen werden:

person1.xml (mit ungültigem Wert für das Geburtsdatum 'Birthdate')

```
<?xml version="1.0"?>
<Person gender="Male">
  <Name>
    <Last>Cool</Last>
    <First>Joe</First>
  </Name>
  <Confidential>
    <Age unit="years">5</Age>
    <Birthdate>March 16, 2002</Birthdate>
    <SS>111-22-3333</SS>
  </Confidential>
  <Address>5224 Rose St. San Jose, CA 95123</Address>
</Person>
```

person2.xml (mit gültigem Wert für 'Birthdate')

```
<?xml version="1.0"?>
<Person gender="Male">
  <Name>
    <Last>Cool</Last>
    <First>Joe</First>
  </Name>
  <Confidential>
    <Age unit="years">5</Age>
    <Birthdate>2002-03-16</Birthdate>
    <SS>111-22-3333</SS>
  </Confidential>
  <Address>5224 Rose St. San Jose, CA 95123</Address>
</Person>
```

person3.xml (mit gültigem Wert für 'Birthdate')

```
<?xml version="1.0"?>
<Person gender="Female">
  <Name>
    <Last>McCarthy</Last>
    <First>Laura</First>
  </Name>
  <Confidential>
    <Age unit="years">6</Age>
    <Birthdate>2001-03-12</Birthdate>
    <SS>444-55-6666</SS>
  </Confidential>
  <Address>5960 Daffodil Lane, San Jose, CA 95120</Address>
</Person>
```

person4.xml (mit ungültigem Wert für 'Birthdate')

```
<?xml version="1.0"?>
<Person gender="Female">
  <Name>
    <Last>Wong</Last>
    <First>Teresa</First>
  </Name>
```

```
  <Confidential>
    <Age unit="years">7</Age>
    <Birthdate>20000-12-09</Birthdate>
    <SS>555-66-7777</SS>
  </Confidential>
  <Address>5960 Tulip Court, San Jose, CA 95120</Address>
</Person>
```

person5.xml (mit gültigem Wert für 'Birthdate')

```
<?xml version="1.0"?>
<Person gender="Male">
  <Name>
    <Last>Smith</Last>
    <First>Chris</First>
  </Name>
  <Confidential>
    <Age unit="years">10</Age>
    <Birthdate>1997-04-23</Birthdate>
    <SS>666-77-8888</SS>
  </Confidential>
  <Address>5960 Dahlia Street, San Jose, CA 95120</Address>
</Person>
```

Die Eingabedatei `person.del` enthält Folgendes:

```
1, <XDS FIL='person1.xml'/>
2, <XDS FIL='person2.xml'/>
3, <XDS FIL='person3.xml'/>
4, <XDS FIL='person4.xml'/>
5, <XDS FIL='person5.xml'/>
```

Die DDL- und LOAD-Anweisungen sehen wie folgt aus:

```
CREATE TABLE T1 (docID INT, XMLDoc XML);

CREATE INDEX BirthdateIndex ON T1(xmlDoc)
   GENERATE KEY USING XMLPATTERN '/Person/Confidential/Birthdate' AS SQL DATE
   REJECT INVALID VALUES;

LOAD FROM person.del OF DEL INSERT INTO T1
```

Um die Indexierungsfehler zu beheben, die bei dem Versuch auftreten würden, die oben angeführte XML-Dateigruppe zu laden, müssten Sie folgende Schritte ausführen:

1. Ermitteln Sie anhand der Satznummern in den Ausgabeinformationen, welche Zeilen während der Ladeoperation zurückgewiesen wurden. Aus der folgenden Ausgabe geht hervor, dass der Satz mit der Nummer 1 und der Satz mit der Nummer 4 zurückgewiesen wurden:

   ```
   SQL20305N  Ein XML-Wert kann nicht eingefügt oder aktualisiert werden,
   weil beim Einfügen in den Index bzw. beim Aktualisieren des Index,
   der durch "IID = 3" in Tabelle "LEECM.T1" angegeben ist,
   ein Fehler festgestellt wurde. Ursachencode = "5".
   Bei Ursachencodes für ein XML-Schema ist "*N" die
   XML-Schemakennung und "*N" der XML-Schemadatentyp.
   SQLSTATE=23525

   SQL3185W  Der vorherige Fehler trat bei der Verarbeitung von Daten aus Zeile "F0-1"
   der Eingabedatei auf.

   SQL20305N  Ein XML-Wert kann nicht
   eingefügt oder aktualisiert werden, weil beim Einfügen in den Index bzw.
   beim Aktualisieren des Index, der durch "IID = 3" in Tabelle
   "LEECM.T1" angegeben ist, ein Fehler festgestellt wurde. Ursachencode = "4".
   Bei Ursachencodes für ein XML-Schema ist "*N" die XML-Schemakennung
   ```

```
und "*N" der XML-Schemadatentyp.
SQLSTATE=23525

SQL3185W  Der vorherige Fehler trat bei der Verarbeitung von Daten aus Zeile "F0-4"
der Eingabedatei auf.

SQL3227W  Satztoken "F0-1" bezieht sich auf Benutzersatznummer "1".

SQL3227W  Satztoken "F0-4" bezieht sich auf Benutzersatznummer "4".

SQL3107W  Die Nachrichtendatei enthält mindestens eine Warnung.

Anzahl gelesener Zeilen           = 5
Anzahl übersprungener Zeilen      = 0
Anzahl geladener Zeilen           = 3
Anzahl zurückgewiesener Zeilen    = 2
Anzahl gelöschter Zeilen          = 0
Anzahl festgeschriebener Zeilen   = 5
```

2. Erstellen Sie eine neue Datei namens `reject.del` mit den zurückgewiesenen Zeilen:

   ```
   1, <XDS FIL='person1.xml'/>
   4, <XDS FIL='person4.xml'/>
   ```

3. Erstellen Sie eine neue Tabelle (T2) mit denselben Spalten wie in der Originaltabelle T1. Erstellen Sie für die neue Tabelle keine Indizes.

   ```
   CREATE TABLE T2 LIKE T1
   ```

4. Laden Sie die zurückgewiesenen Zeilen in die neue Tabelle (T2).

   ```
   LOAD FROM reject.del OF DEL INSERT INTO T2;
   ```

5. Führen Sie für die erste zurückgewiesene Zeile in der Originaltabelle T1 Folgendes aus:

   a. Importieren Sie die zurückgewiesenen Zeilen in T1, um die Nachricht -20305 zu erhalten:

      ```
      IMPORT FROM reject.del OF DEL INSERT INTO T1
      SQL3109N  Das Dienstprogramm beginnt mit dem Laden von Daten
      aus der Datei "reject.del".

      SQL3306N  SQL-Fehler "-20305" beim Einfügen einer Zeile in die Tabelle.

      SQL20305N  Ein XML-Wert kann nicht
      eingefügt oder aktualisiert werden, weil beim Einfügen in den Index bzw.
      beim Aktualisieren des Index, der durch "IID = 3" in Tabelle "LEECM.T1"
      angegeben ist, ein Fehler festgestellt wurde. Ursachencode = "5".
      Bei Ursachencodes für ein XML-Schema ist "*N" die XML-Schemakennung
      und "*N" der XML-Schemadatentyp.
      SQLSTATE=23525

      SQL3110N  Die Verarbeitung durch das Dienstprogramm ist abgeschlossen.
      "1" Zeile(n) aus der Eingabedatei wurde(n) gelesen.
      ```

   b. Überprüfen Sie die Protokolldatei **db2diag** und suchen Sie die generierte XQuery-Anweisung.

      ```
      FUNCTION: DB2 UDB, Xml Storage and Index Manager, xmlsDumpXQuery, probe:608
      DATA #1 : String, 36 bytes
      SQL Code: SQL20305N ; Reason Code: 5
      DATA #2 : String, 265 bytes
      To locate the value in the document that caused the error, create a
      table with one XML column and insert the failing document in the table.
      Replace the table and column name in the query below with the created
      table and column name and execute the following XQuery.
      DATA #3 : String, 247 bytes
      xquery for $i in db2-fn:xmlcolumn(
        "LEECM.T1.XMLDOC")[/*:Person/*:Confidential/*:Birthdate="March 16, 2002"]
      return
      ```

```
<Result>
 <ProblemDocument> {$i} </ProblemDocument>
 <ProblemValue>{$i/*:Person/*:Confidential/*:Birthdate/..} </ProblemValue>
</Result>;
```

c. Ändern Sie die XQuery-Anweisung entsprechend, um die neue Tabelle T2 zu verwenden.

```
xquery for $i in db2-fn:xmlcolumn(
  "LEECM.T2.XMLDOC")[/*:Person/*:Confidential/*:Birthdate="March 16, 2002"]
return
<Result>
 <ProblemDocument> {$i} </ProblemDocument>
 <ProblemValue>{$i/*:Person/*:Confidential/*:Birthdate/..} </ProblemValue>
</Result>;
```

d. Führen Sie die XQuery-Anweisung für Tabelle T2 aus, um den Problemwert im Dokument zu lokalisieren:

```
<Result><ProblemDocument><Person gender="Male">
  <Name>
    <Last>Cool</Last>
    <First>Joe</First>
  </Name>
  <Confidential>
    <Age unit="years">5</Age>
    <Birthdate>March 16, 2002</Birthdate>
    <SS>111-22-3333</SS>
  </Confidential>
  <Address>5224 Rose St. San Jose, CA 95123</Address>
</Person></ProblemDocument><ProblemValue><Confidential>
    <Age unit="years">5</Age>
    <Birthdate>March 16, 2002</Birthdate>
    <SS>111-22-3333</SS>
  </Confidential></ProblemValue></Result>
```

e. Korrigieren Sie den Problemwert in der Datei `person1.xml`, die das Dokument enthält. Der Wert `March 16, 2002` liegt nicht im korrekten Datumsformat vor und muss daher in den Wert `2002-03-16` geändert werden:

```
<?xml version="1.0"?>
<Person gender="Male">
  <Name>
    <Last>Cool</Last>
    <First>Joe</First>
  </Name>
  <Confidential>
    <Age unit="years">5</Age>
    <Birthdate>2002-03-16</Birthdate>
    <SS>111-22-3333</SS>
  </Confidential>
  <Address>5224 Rose St. San Jose, CA 95123</Address>
</Person>
```

f. Wiederholen Sie Schritt a., um die zurückgewiesenen Zeilen erneut in Tabelle T1 zu importieren.

6. (Erste Wiederholung von Schritt 5)

a. Importieren Sie die zurückgewiesenen Zeilen in Tabelle T1. Die erste Zeile wurde nun erfolgreich importiert, da zwei Zeilen aus der Importdatei gelesen wurden. Ein neuer Fehler tritt bei der zweiten Zeile auf:

```
IMPORT FROM reject.del OF DEL INSERT INTO T1
SQL3109N  Das Dienstprogramm beginnt mit dem Laden von Daten
aus der Datei "reject.del".

SQL3306N  SQL-Fehler "-20305" beim Einfügen einer Zeile in die Tabelle.

SQL20305N  Ein XML-Wert kann nicht eingefügt oder aktualisiert werden,
weil beim Einfügen in den Index bzw. beim Aktualisieren des Index,
```

```
der durch "IID = 3" in Tabelle "LEECM.T1" angegeben ist,
ein Fehler festgestellt wurde. Ursachencode = "4".
Bei Ursachencodes für ein XML-Schema ist "*N" die XML-Schemakennung
und "*N" der XML-Schemadatentyp.
SQLSTATE=23525

SQL3110N  Die Verarbeitung durch das Dienstprogramm ist abgeschlossen.
"2" Zeilen aus der Eingabedatei wurden gelesen.
```

b. Überprüfen Sie die Protokolldatei **db2diag** und suchen Sie die generierte XQuery-Anweisung.

```
FUNCTION: DB2 UDB, Xml Storage and Index Manager, xmlsDumpXQuery, probe:608
DATA #1 : String, 36 bytes
SQL Code: SQL20305N ; Reason Code: 4
DATA #2 : String, 265 bytes
To locate the value in the document that caused the error, create a
table with one XML column and insert the failing document in the table.
Replace the table and column name in the query below with the created
table and column name and execute the following XQuery.
DATA #3 : String, 244 bytes
xquery for $i in db2-fn:xmlcolumn("LEECM.T1.XMLDOC")
   [/*:Person/*:Confidential/*:Birthdate="20000-12-09"]
return
<Result>
 <ProblemDocument> {$i} </ProblemDocument>
 <ProblemValue>{$i/*:Person/*:Confidential/*:Birthdate/..} </ProblemValue>
</Result>;
```

c. Ändern Sie die XQuery-Anweisung entsprechend, um Tabelle T2 zu verwenden:

```
xquery for $i in db2-fn:xmlcolumn("LEECM.T2.XMLDOC")
   [/*:Person/*:Confidential/*:Birthdate="20000-12-09"]
return
<Result>
 <ProblemDocument> {$i} </ProblemDocument>
 <ProblemValue>{$i/*:Person/*:Confidential/*:Birthdate/..} </ProblemValue>
</Result>;
```

d. Führen Sie die XQuery-Anweisung aus, um den Problemwert im Dokument zu lokalisieren:

```
<Result><ProblemDocument><Person gender="Female">
  <Name>
    <Last>Wong</Last>
    <First>Teresa</First>
  </Name>
  <Confidential>
    <Age unit="years">7</Age>
    <Birthdate>20000-12-09</Birthdate>
    <SS>555-66-7777</SS>
  </Confidential>
  <Address>5960 Tulip Court, San Jose, CA 95120</Address>
</Person></ProblemDocument><ProblemValue><Confidential>
    <Age unit="years">7</Age>
    <Birthdate>20000-12-09</Birthdate>
    <SS>555-66-7777</SS>
  </Confidential></ProblemValue></Result>
```

e. Korrigieren Sie den Problemwert in der Datei `person4.xml`, die das Dokument enthält. Der Wert `20000-12-09` weist in der Jahresangabe eine zusätzliche Null auf, sodass die Jahresangabe außerhalb des Bereichs liegt, der in DB2 für Jahresangaben zulässig ist (0001 bis 9999). Der Wert wird in `2000-12-09` geändert:

```
<?xml version="1.0"?>
<Person gender="Female">
  <Name>
    <Last>Wong</Last>
```

```
    <First>Teresa</First>
  </Name>
  <Confidential>
    <Age unit="years">7</Age>
    <Birthdate>2000-12-09</Birthdate>
    <SS>555-66-7777</SS>
  </Confidential>
  <Address>5960 Tulip Court, San Jose, CA 95120</Address>
</Person>
```

f. Wiederholen Sie Schritt a, um die zurückgewiesenen Zeilen erneut in Tabelle T1 zu importieren.

7. (Zweite Wiederholung von Schritt 5)

   a. Importieren Sie die zurückgewiesenen Zeilen in Tabelle T1:

```
IMPORT FROM reject.del OF DEL INSERT INTO T1
SQL3109N  Das Dienstprogramm beginnt mit dem Laden von Daten
aus der Datei "reject.del".

SQL3110N  Die Verarbeitung durch das Dienstprogramm ist abgeschlossen.
"2" Zeilen aus der Eingabedatei der Eingabedatei auf.

SQL3221W  ...COMMIT WORK beginnt. Eingabesatzzähler = "2".

SQL3222W  ...COMMIT für alle Änderungen der Datenbank wurde erfolgreich ausgeführt.

SQL3149N  "2" Zeilen aus der Eingabedatei wurden verarbeitet. "2" Zeile(n) wurden
ohne Fehler der Tabelle hinzugefügt. "0" Zeile(n) wurden zurückgewiesen.

Anzahl gelesener Zeilen           = 2
Anzahl übersprungener Zeilen      = 0
Anzahl eingefügter Zeilen         = 2
Anzahl aktualisierter Zeilen      = 0
Anzahl zurückgewiesener Zeilen    = 0
Anzahl festgeschriebener Zeilen   = 2
```

   Das Problem ist nun behoben. Alle Zeilen aus der Datei `person.del` wurden erfolgreich in Tabelle T1 eingefügt.

Kapitel 10. Anwendungsprogrammiersprachenunterstützung

Sie können Anwendungen zum Speichern von XML-Daten in DB2-Datenbanktabellen, zum Abrufen von Daten aus Tabellen oder zum Aufrufen gespeicherter Prozeduren bzw. benutzerdefinierter Funktionen mit XML-Parametern schreiben.

Sie können die folgenden Sprachen verwenden, um Anwendungen zu schreiben:
- C oder C++ (eingebettetes SQL oder CLI)
- COBOL
- Java (JDBC oder SQLJ)
- C# und Visual Basic (IBM Data Server Provider für .NET)
- PHP
- Perl

Ein Anwendungsprogramm kann ein vollständiges Dokument oder ein Fragment eines Dokuments aus einer XML-Spalte abrufen. Allerdings können in XML-Spalten nur vollständige Dokumente gespeichert werden.

Gespeicherte Prozeduren und benutzerdefinierte Funktionen können XML-Werte in Ein- und Ausgabeparametern übergeben. XML-Daten werden umgesetzt, wenn sie als Parameter IN, OUT oder INOUT an gespeicherte Prozeduren übergeben werden. Wenn Sie mit gespeicherten Java-Prozeduren arbeiten, müssen eventuell die Zwischenspeichergröße (Konfigurationsparameter **java_heap_sz**) je nach Anzahl und Größe der XML-Argumente sowie die Anzahl externer gespeicherter Prozeduren, die gleichzeitig ausgeführt werden, erhöht werden. Zum Aufrufen einer gespeicherten Prozedur oder benutzerdefinierten Funktion, die über XML- oder XML AS CLOB-Parameter verfügt, müssen Sie eine Anweisung vom Typ CALL mit kompatiblen Datentypen ausführen.

Wenn eine Anwendung einen XML-Wert für einen DB2-Datenbankserver bereitstellt, konvertiert der Datenbankserver die Daten aus dem serialisierten XML-Zeichenfolgeformat in das hierarchische XML-Format, wobei die UTF-8-Unicode-Codierung verwendet wird.

Wenn eine Anwendung Daten aus XML-Spalten abruft, konvertiert der DB2-Datenbankserver die Daten aus dem hierarchischen XML-Format in das serialisierte XML-Zeichenfolgeformat. Darüber hinaus muss der Datenbankserver die Ausgabedaten möglicherweise aus der UTF-8-Codierung in die Codierung der Anwendung konvertieren.

Wenn Sie XML-Daten abrufen, müssen Sie berücksichtigen, dass es bei der Codepagekonvertierung zu Datenverlusten kommen kann. Dieser Fall kann eintreten, wenn Zeichen in der Quellencodepage in der Zielcodepage nicht dargestellt werden können.

Eine Anwendung kann ein vollständiges XML-Dokument oder eine bestimmte Sequenz aus einer XML-Spalte abrufen.

Wenn Sie ein vollständiges XML-Dokument abrufen, wird das Dokument in eine Anwendungsvariable abgerufen.

Wenn Sie eine XML-Sequenz abrufen, haben Sie die folgenden Optionen:
- Direktes Ausführen eines XQuery-Ausdrucks.

Zum Ausführen eines XQuery-Ausdrucks in einer Anwendung müssen Sie dem XQuery-Ausdruck die Zeichenfolge 'XQUERY' voranstellen und die resultierende Zeichenfolge dynamisch ausführen.

Wenn Sie einen XQuery-Ausdruck direkt ausführen, gibt der DB2-Datenbankserver die Sequenz, die als Ergebnis der XQuery-Anweisung generiert wurde, als Ergebnistabelle zurück. Jede Zeile in der Ergebnistabelle stellt ein Element in der Sequenz dar.

- Innerhalb einer SQL-Operation SELECT oder SELECT INTO für eine einzelne Zeile können Sie die integrierte Funktion XMLQUERY oder XMLTABLE aufrufen und einen XQuery-Ausdruck als Argument übergeben.

  Diese Vorgehensweise kann bei statischem und dynamischem SQL und bei allen Anwendungsprogrammiersprachen verwendet werden. XMLQUERY ist eine Skalarfunktion, die die gesamte Sequenz in einer Anwendungsvariablen zurückgibt. XMLTABLE ist eine Tabellenfunktion, die die einzelnen Elemente der Sequenz als Zeilen der Ergebnistabelle zurückgibt. Die Spalten in der Ergebnistabelle enthalten Werte aus dem abgerufenen Sequenzelement.

Parametermarken und Hostvariablen

Parametermarken oder Hostvariablen dürfen innerhalb eines XQuery-Ausdrucks nicht angegeben werden. Dies gilt auch für den SQL-Code, der in einem XQuery-Ausdruck angegeben wird. Die XQuery-Funktion 'db2-fn:sqlquery' beispielsweise ermöglicht Ihnen das Angeben einer SQL-Fullselect-Anweisung mit einem XQuery-Ausdruck, um die detaillierte Beschreibung eines Produkts zu extrahieren:

```
  xquery
db2-fn:sqlquery("select description from product where pid='100-103-01'")
                /product/description/details/text()
```

In dem XQuery-Ausdruck dürfen Sie keine Parametermarke und keine Hostvariable angeben, auch nicht innerhalb der Fullselect-Anweisung. Der folgende Ausdruck ist falsch und wird nicht unterstützt (und gibt SQLSTATE-Wert 42610, SQLCODE-Wert -418 zurück):

```
  xquery
db2-fn:sqlquery("select description from product where pid=?")
                /product/description/details/text()
```

Um Anwendungswerte an XQuery-Ausdrücke zu übergeben, müssen Sie die SQL/XML-Funktionen XMLQUERY und XMLTABLE verwenden. Die Klausel PASSING dieser Funktionen ermöglicht Ihnen die Verwendung von Anwendungswerten bei der Auswertung des XQuery-Ausdrucks.

Die folgende Abfrage zeigt, wie die vorherige falsche Abfrage mithilfe von SQL/XML umgeschrieben werden kann, um ein entsprechendes Ergebnis zu erzielen:

```
SELECT XMLQUERY ('$descdoc/product/description/details/text()'
   passing description as "descdoc")
FROM product
WHERE pid=?
```

CLI

XML-Datenverarbeitung in CLI-Anwendungen - Übersicht

CLI-Anwendungen können XML-Daten mithilfe des Datentyps SQL_XML abrufen. Dieser Datentyp entspricht dem nativen XML-Datentyp der DB2-Datenbank, mit dem Spalten zum Speichern korrekt formatierter XML-Dokumente definiert werden. Den Datentyp SQL_XML können Sie an folgende C-Typen binden: SQL_C_BI-

NARY, SQL_C_CHAR, SQL_C_WCHAR und SQL_C_DBCHAR. Verwenden Sie anstelle der Zeichentypen den Standardtyp SQL_C_BINARY, um einen möglichen Datenverlust oder eine mögliche Datenbeschädigung zu vermeiden, der bzw. die möglicherweise bei der Verwendung von Zeichentypen aufgrund der Codepagekonvertierung auftritt.

Binden Sie zum Speichern von XML-Daten in einer XML-Spalte einen Binärpuffer (SQL_C_BINARY) oder einen Zeichenpuffer (SQL_C_CHAR, SQL_C_WCHAR oder SQL_C_DBCHAR), der den XML-Wert enthält, an den SQL-Typ SQL_XML, und führen Sie die Anweisung INSERT oder UPDATE SQL aus. Binden Sie zum Abrufen von XML-Daten aus der Datenbank die Ergebnismenge an einen Binärtyp (SQL_C_BINARY) oder einen Zeichentyp (SQL_C_CHAR, SQL_C_WCHAR oder SQL_C_DBCHAR). Sie sollten Zeichentypen aufgrund möglicher Probleme bei der Codierung mit Vorsicht verwenden.

Beim Abrufen eines XML-Werts in einen Anwendungsdatenpuffer führt der DB2-Server eine implizite Serialisierung des XML-Werts durch, um ihn aus dem gespeicherten hierarchischen Format in das serialisierte Zeichenfolgeformat umzuwandeln. Bei Zeichentyppuffern wird der XML-Wert implizit in die Zeichencodeseite der Anwendung serialisiert, die dem Zeichentyp zugeordnet ist.

Standardmäßig wird eine XML-Deklaration in die serialisierte Ausgabezeichenfolge eingebunden. Sie können dieses Standardverhalten ändern, indem Sie die Anweisung oder das Verbindungsattribut SQL_ATTR_XML_DECLARATION oder das CLI/ODBC-Konfigurationsschlüsselwort XMLDeclaration in der Datei `db2cli.ini` festlegen.

Sie können XQuery-Ausdrücke und SQL/XML-Funktionen in CLI-Anwendungen ausgeben und ausführen. SQL/XML-Funktionen werden in derselben Weise wie alle anderen SQL-Anweisungen ausgegeben und ausgeführt. Sie müssen ein Präfix zu XQuery-Ausrücken mit den Schlüsselwort **`XQUERY`**, bei dem die Groß- und Kleinschreibung keine Rolle spielt, hinzufügen oder Sie müssen das Anweisungsattribut SQL_ATTR_XQUERY_STATEMENT für die Anweisungskennung festlegen, das dem XQuery-Ausdruck zugeordnet ist.

Anmerkung: DB2 ab Version 9.7 Fixpack 5 unterstützt den Datentyp SQL_XML für DB2-Server ab Version 7 Release 1.

XML-Spalteneinfügungen und -aktualisierungen in CLI-Anwendungen

Wenn Sie Daten in XML-Spalten einer Tabelle aktualisieren oder einfügen, müssen die Eingabedaten im serialisierten Zeichenfolgeformat vorliegen.

Wenn Sie bei XML-Daten Parametermarken mit `SQLBindParameter()` an Eingabedatenpuffer binden, können Sie für den Datentyp des Eingabedatenpuffers SQL_C_BINARY, SQL_C_CHAR, SQL_C_DBCHAR oder SQL_C_WCHAR angeben.

Wenn Sie einen Datenpuffer, der XML-Daten enthält, als SQL_C_BINARY binden, verarbeitet CLI die XML-Daten als intern codierte Daten. Dies ist die bevorzugte Methode, da auf diese Weise eine weitere Ressourcennutzung sowie möglicher Datenverlust durch Zeichenkonvertierung bei der Verwendung von Zeichentypen vermieden wird.

Wichtig: Wenn die XML-Daten in einem Codierungsschema und einer CCSID codiert werden, das bzw. die nicht mit dem Codierungsschema der Anwendungsco-

depage übereinstimmt, müssen Sie interne Codierung in die Daten einschließen und sie als SQL_C_BINARY binden, um die Zeichenkonvertierung zu umgehen.

Wenn Sie einen Datenpuffer, der XML-Daten enthält, als SQL_C_CHAR, SQL_C_DBCHAR oder SQL_C_WCHAR binden, verarbeitet CLI die XML-Daten als extern codierte Daten. CLI ermittelt die Codierung der Daten wie folgt:

- Wenn der C-Typ SQL_C_WCHAR lautet, nimmt CLI an, dass die Daten als UCS-2 codiert sind.
- Wenn der C-Typ SQL_C_CHAR oder SQL_C_DBCHAR lautet, nimmt CLI an, dass die Daten im Codierungsschema der Anwendungscodepage codiert sind.

Wenn der Datenbankserver für die Daten implizites Parsing ausführen soll, bevor diese in einer XML-Spalte gespeichert werden, muss der Datentyp der Parametermarke in `SQLBindParameter()` als SQL_XML angegeben werden.

Es empfiehlt sich implizites Parsing, da beim expliziten Parsing eines Zeichentyps mit XMLPARSE Codierungsprobleme auftreten können.

Das folgende Beispiel zeigt, wie XML-Daten in einer XML-Spalte mithilfe des empfohlenen Typs SQL_C_BINARY aktualisiert werden.

```
char xmlBuffer[10240];
integer length;

// Assume a table named dept has been created with the following statement:
// CREATE TABLE dept (id CHAR(8), deptdoc XML)

// xmlBuffer contains an internally encoded XML document that is to replace
// the existing XML document
length = strlen (xmlBuffer);
SQLPrepare (hStmt, "UPDATE dept SET deptdoc = ? WHERE id = '001'", SQL_NTS);
SQLBindParameter (hStmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY, SQL_XML, 0, 0,
                  xmlBuffer, 10240, &length);
SQLExecute (hStmt);
```

Abrufen von XML-Daten in CLI-Anwendungen

Wenn Sie in einer Tabelle Daten aus XML-Spalten auswählen, liegen die Ausgabedaten im serialisierten Zeichenfolgeformat vor.

Wenn Sie bei XML-Daten Spalten in einer Abfrageergebnismenge mit `SQLBindCol()` an Anwendungsvariablen binden, können Sie für den Datentyp des Eingabedatenpuffers SQL_C_BINARY, SQL_C_CHAR, SQL_C_DBCHAR oder SQL_C_WCHAR angeben. Wenn Sie eine Ergebnismenge aus einer XML-Spalte abrufen, empfiehlt es sich, Ihre Anwendungsvariable an den Typ SQL_C_BINARY zu binden. Das Binden an Zeichentypen kann Datenverlust aufgrund der Codepagekonvertierung zur Folge haben. Dieser Fall kann eintreten, wenn Zeichen in der Quellencodepage in der Zielcodepage nicht dargestellt werden können. Durch das Binden Ihrer Variablen an den C-Typ SQL_C_BINARY lassen sich solche Probleme umgehen.

XML-Daten werden als intern codierte Daten an die Anwendung zurückgegeben. CLI ermittelt die Codierung der Daten wie folgt:

- Wenn der C-Typ SQL_C_BINARY lautet, gibt CLI die Daten im UTF-8-Codierungsschema zurück.
- Wenn der C-Typ SQL_C_CHAR oder SQL_C_DBCHAR lautet, gibt CLI die Daten im Codierungsschema der Anwendungscodepage zurück.
- Wenn der C-Typ SQL_C_WCHAR lautet, gibt CLI die Daten im UCS-2-Codierungsschema zurück.

Der Datenbankserver führt eine implizite Serialisierung der Daten aus, bevor er sie an die Anwendung zurückgibt. Durch Aufruf der Funktion XMLSERIALIZE können Sie die XML-Daten explizit in einen bestimmten Datentyp serialisieren. Es empfiehlt sich jedoch eine implizite Serialisierung, da bei der expliziten Serialisierung in Zeichentypen mit XMLSERIALIZE Codierungsprobleme auftreten können.

Das folgende Beispiel zeigt, wie XML-Daten aus einer XML-Spalte in eine binäre Anwendungsvariable abgerufen werden.

```
char xmlBuffer[10240];
// xmlBuffer is used to hold the retrieved XML document
integer length;

// Assume a table named dept has been created with the following statement:
// CREATE TABLE dept (id CHAR(8), deptdoc XML)

length = sizeof (xmlBuffer);
SQLExecute (hStmt, "SELECT deptdoc FROM dept WHERE id='001'", SQL_NTS);
SQLBindCol (hStmt, 1, SQL_C_BINARY, xmlBuffer, &length, NULL);
SQLFetch (hStmt);
SQLCloseCursor (hStmt);
// xmlBuffer now contains a valid XML document encoded in UTF-8
```

Ändern der Standardverarbeitung von XML-Datentypen in CLI-Anwendungen

CLI unterstützt CLI/ODBC-Konfigurationsschlüsselwörter, die Kompatibilität für Anwendungen bieten, die nicht die Standardtypen erwarten, die bei der Beschreibung oder Angabe von SQL_C_DEFAULT für XML-Spalten und -Parametermarken zurückgegeben werden. Ältere CLI- und ODBC-Anwendungen erkennen oder erwarten möglicherweise bei der Beschreibung von XML-Spalten oder -Parametern nicht den Standardtyp SQL_XML. Bestimmte CLI- oder ODBC-Anwendungen erwarten möglicherweise auch bei XML-Spalten und -Parametermarken einen anderen Standardtyp als SQL_C_BINARY. Zur Bereitstellung von Kompatibilität für diese Anwendungstypen unterstützt CLI die Schlüsselwörter MapXMLDescribe und MapXMLCDefault.

MapXMLDescribe gibt an, welcher SQL-Datentyp bei der Beschreibung von XML-Spalten oder -Parametermarken zurückgegeben wird.

MapXMLCDefault gibt an, welcher C-Typ bei der Angabe von SQL_C_DEFAULT für XML-Spalten und -Parametermarken in CLI-Funktionen verwendet wird.

Eingebettetes SQL

Deklarieren von XML-Hostvariablen in Anwendungen mit eingebettetem SQL

Zum Austausch von XML-Daten zwischen dem Datenbankserver und einer Anwendung mit eingebettetem SQL müssen Sie Hostvariablen in Ihrem Anwendungsquellcode deklarieren.

Informationen zu diesem Vorgang

DB2 V9.1 enthält einen XML-Datentyp, der XML-Daten in einer strukturierten Gruppe von Knoten im Format einer Baumstruktur speichert. Spalten mit diesem XML-Datentyp werden als SQL_TYP_XML-Spalte SQLTYPE beschrieben, und Anwendungen können diverse sprachspezifische Datentypen für die Eingabe in diese

Spalten oder Parameter bzw. für die Ausgabe daraus binden. Auf XML-Spalten kann direkt über SQL, die SQL/XML-Erweiterungen oder XQuery zugegriffen werden. Der XML-Datentyp gilt nicht nur für Spalten. Funktionen können Argumente in Form von XML-Werten haben und auch XML-Werte erstellen. In ähnlicher Weise können gespeicherte Prozeduren XML-Werte als Ein- und Ausgabeparameter annehmen. XQuery-Ausdrücke schließlich können XML-Werte unabhängig davon erstellen, ob sie auf XML-Spalten zugreifen oder nicht.

XML-Daten sind normalerweise Zeichen und besitzen eine Codierung, die den verwendeten Zeichensatz angibt. Die Codierung der XML-Daten kann extern ermittelt werden, abgeleitet aus dem Basisanwendungstyp, der die serialisierte Zeichenfolgedarstellung des XML-Dokuments enthält. Sie kann auch intern ermittelt werden, was eine Interpretation der Daten erforderlich macht. Für Dokumente, die mit Unicode codiert sind, empfiehlt sich eine Byteanordnungsmarkierung (BOM), die aus einem Unicode-Zeichencode am Anfang eines Datenstroms besteht. Die BOM dient als Signatur, mit der die Byteanordnung und das Format der Unicode-Codierung definiert werden.

Zum Abrufen und Einfügen von Daten können neben XML-Hostvariablen vorhandene Zeichen- und binäre Typen wie CHAR, VARCHAR, CLOB und BLOB verwendet werden. Diese unterliegen jedoch im Gegensatz zu SML-Hostvariablen nicht implizitem XML-Parsing. Stattdessen wird eine explizite XMLPARSE-Funktion mit standardmäßiger Leerzeichenlöschung integriert und angewendet.

Einschränkungen für XML und XQuery bei der Entwicklung von Anwendungen mit eingebettetem SQL

Gehen Sie wie folgt vor, um XML-Hostvariablen in Anwendungen mit eingebettetem SQL zu deklarieren:

Deklarieren Sie die XML-Hostvariablen im Deklarationsabschnitt der Anwendung als LOB-Datentypen:

- `SQL TYPE IS XML AS CLOB(n) <hostvariablenname>`

  Dabei ist <hostvariablenname> eine CLOB-Hostvariable, die XML-Daten enthält, die in der gemischten Codepage der Anwendung codiert sind.

- `SQL TYPE IS XML AS DBCLOB(n) <hostvariablenname>`

  Dabei ist <hostvariablenname> eine DBCLOB-Hostvariable, die XML-Daten enthält, die in der grafischen Codepage der Anwendung codiert sind.

- `SQL TYPE IS XML AS BLOB(n) <hostvariablenname>`

  Dabei ist <hostvariablenname> eine BLOB-Hostvariable, die intern codierte XML-Daten enthält [1].

- `SQL TYPE IS XML AS CLOB_FILE <hostvariablenname>`

  Dabei ist <hostvariablenname> eine CLOB-Datei, die XML-Daten enthält, die in der gemischten Codepage der Anwendung codiert sind.

- `SQL TYPE IS XML AS DBCLOB_FILE <hostvariablenname>`

Dabei ist <hostvariablenname> eine DBCLOB-Datei, die XML-Daten enthält, die in der grafischen Codepage der Anwendung codiert sind.

- ```
  SQL TYPE IS XML AS BLOB_FILE <hostvariablenname>
  ```

  Dabei ist <hostvariablenname> eine DBCLOB-Datei, die intern codierte XML-Daten enthält [1].

Anmerkung:

1. Lesen Sie die Informationen über den Algorithmus zum Ermitteln der Codierung bei XML 1.0-Spezifikationen (`http://www.w3.org/TR/REC-xml/#sec-guessing-no-ext-info`).

Beispiel: Verweisen auf XML-Hostvariablen in Anwendungen mit eingebettetem SQL

Mit den folgenden Beispielanwendungen wird gezeigt, wie in C und COBOL auf XML-Hostvariablen verwiesen wird.

Beispiel: C-Anwendung mit eingebettetem SQL:

```
Das folgende Codebeispiel wurde zur besseren Übersichtlichkeit formatiert:
EXEC SQL BEGIN DECLARE;
  SQL TYPE IS XML AS CLOB( 10K ) xmlBuf;
  SQL TYPE IS XML AS BLOB( 10K ) xmlblob;
  SQL TYPE IS CLOB( 10K ) clobBuf;
EXEC SQL END DECLARE SECTION;

// as XML AS CLOB
// The XML value written to xmlBuf will be prefixed by an XML declaration
// similar to: <?xml version = "1.0" encoding = "ISO-8859-1" ?>
// Note: The encoding name will depend upon the application codepage
EXEC SQL SELECT xmlCol INTO :xmlBuf
   FROM myTable
   WHERE id = '001';
EXEC SQL UPDATE myTable
   SET xmlCol = :xmlBuf
   WHERE id = '001';

// as XML AS BLOB
// The XML value written to xmlblob will be prefixed by an XML declaration
// similar to: <?xml version = "1.0" encoding = "UTF-8"?>
EXEC SQL SELECT xmlCol INTO :xmlblob
   FROM myTable
   WHERE id = '001';
EXEC SQL UPDATE myTable
   SET xmlCol = :xmlblob
   WHERE id = '001';

// as CLOB
// The output will be encoded in the application character codepage,
// but will not contain an XML declaration
EXEC SQL SELECT XMLSERIALIZE (xmlCol AS CLOB(10K)) INTO :clobBuf
   FROM myTable
   WHERE id = '001';
EXEC SQL UPDATE myTable
   SET xmlCol = XMLPARSE (:clobBuf PRESERVE WHITESPACE)
   WHERE id = '001';
```
```

Beispiel: COBOL-Anwendung mit eingebettetem SQL:

```
Das folgende Codebeispiel wurde zur besseren Übersichtlichkeit formatiert:
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
  01 xmlBuf USAGE IS SQL TYPE IS XML as CLOB(5K).
  01 clobBuf USAGE IS SQL TYPE IS CLOB(5K).
  01 xmlblob  USAGE IS SQL TYPE IS BLOB(5K).
EXEC SQL END DECLARE SECTION END-EXEC.

* as XML
EXEC SQL SELECT xmlCol INTO :xmlBuf
   FROM myTable
   WHERE id = '001' END-EXEC.
EXEC SQL UPDATE myTable
   SET xmlCol = :xmlBuf
   WHERE id = '001' END-EXEC.

* as BLOB
EXEC SQL SELECT xmlCol INTO :xmlblob
   FROM myTable
   WHERE id = '001' END-EXEC.
EXEC SQL UPDATE myTable
   SET xmlCol = :xmlblob
   WHERE id = '001' END-EXEC.

* as CLOB
EXEC SQL SELECT XMLSERIALIZE (xmlCol AS CLOB(10K)) INTO :clobBuf
   FROM myTable
   WHERE id= '001' END-EXEC.
EXEC SQL UPDATE myTable
   SET xmlCol = XMLPARSE(:clobBuf) PRESERVE WHITESPACE
   WHERE id = '001' END-EXEC.
```

Ausführen von XQuery-Ausdrücken in Anwendungen mit eingebettetem SQL

Vorbereitende Schritte

Mithilfe von XQuery-Ausdrücken können Sie XML-Daten in Ihren Tabellen speichern und über eingebettete SQL-Anwendungen auf die XML-Spalten zugreifen. Verwenden Sie XML-Hostvariablen für den Zugriff auf XML-Daten, anstatt die Daten in Zeichen- oder binäre Datentypen umzusetzen. Wenn Sie keine XML-Hostvariablen verwenden, stellen FOR BIT DATA- oder BLOB-Datentypen die beste Alternative für den Zugriff auf XML-Daten dar, da sich damit eine Codepagekonvertierung vermeiden lässt.

- Deklarieren Sie XML-Hostvariablen in Ihren eingebetteten SQL-Anwendungen.

Informationen zu diesem Vorgang

- Zum Abrufen von XML-Werten in einer statischen SQL-Anweisung `SELECT INTO` muss ein XML-Datentyp verwendet werden.
- Wenn in Situationen, in denen ein XML-Wert erwartet wird, eine Hostvariable des Typs CHAR, VARCHAR, CLOB oder BLOB zur Eingabe verwendet wird, richtet sich der Wert nach einer Operation der Funktion XMLPARSE mit standardmäßiger Leerzeichenverarbeitung (STRIP). Andernfalls ist eine XML-Hostvariable erforderlich.

Zum direkten Ausführen von XQuery-Ausdrücken in Anwendungen mit eingebettetem SQL müssen Sie dem Ausdruck das Schlüsselwort "XQUERY" voranstellen. Verwenden Sie bei statischem SQL die Funktion XMLQUERY. Beim Aufruf der Funktion XMLQUERY wird "XQUERY" dem XQuery-Ausdruck nicht vorangestellt.

Die folgenden Beispiele geben Daten von den XML-Dokumenten in Tabelle CUSTOMER von der Beispieldatenbank zurück.

Beispiel 1: Direktes Ausführen von XQuery-Ausdrücken in C- und C++-Anwendungen in dynamischem SQL durch Voranstellen des Schlüsselworts "XQUERY"

In C- und C++-Anwendungen können XQuery-Ausdrücke wie folgt ausgeführt werden:

```
EXEC SQL INCLUDE SQLCA;
EXEC SQL BEGIN DECLARE SECTION;
  char stmt[16384];
  SQL TYPE IS XML AS BLOB( 10K ) xmlblob;
EXEC SQL END DECLARE SECTION;

sprintf( stmt, "XQUERY (for $a in db2-fn:xmlcolumn("CUSTOMER.INFO")
        /*:customerinfo[*:addr/*:city = "Toronto"]/@Cid return data($a))");

EXEC SQL PREPARE s1 FROM :stmt;
EXEC SQL DECLARE c1 CURSOR FOR s1;
EXEC SQL OPEN c1;

while( sqlca.sqlcode == SQL_RC_OK )
{
  EXEC SQL FETCH c1 INTO :xmlblob;
  /* Display results */
}

EXEC SQL CLOSE c1;
EXEC SQL COMMIT;
```

Beispiel 2: Ausführen von XQuery-Ausdrücken in statischem SQL mit der Funktion XMLQUERY und dem Prädikat XMLEXISTS

SQL-Anweisungen, in denen die Funktion XMLQUERY enthalten ist, können folgendermaßen statisch vorbereitet werden:

```
EXEC SQL INCLUDE SQLCA;
EXEC SQL BEGIN DECLARE SECTION;
  SQL TYPE IS XML AS BLOB( 10K ) xmlblob;
EXEC SQL END DECLARE SECTION;

EXEC SQL DECLARE C1 CURSOR FOR SELECT XMLQUERY(data($INFO/*:customerinfo/@Cid)')
FROM customer
WHERE XMLEXISTS('$INFO/*:customerinfo[*:addr/*:city = "Toronto"]');

EXEC SQL OPEN c1;

while( sqlca.sqlcode == SQL_RC_OK )
{
  EXEC SQL FETCH c1 INTO :xmlblob;
  /* Display results */
}

EXEC SQL CLOSE c1;
EXEC SQL COMMIT;
```

Beispiel 3: Ausführen von XQuery-Ausdrücken in COBOL-Anwendungen mit eingebettetem SQL

In COBOL-Anwendungen können XQuery-Ausdrücke wie folgt ausgeführt werden:

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
     01 stmt pic x(80).
   01 xmlBuff USAGE IS SQL TYPE IS XML AS BLOB (10K).
EXEC SQL END DECLARE SECTION END-EXEC.

MOVE "XQUERY (for $a in db2-fn:xmlcolumn("CUSTOMER.INFO")/*:customerinfo
            [*:addr/*:city = "Toronto"]/@Cid return data($a)))" TO stmt.
EXEC SQL PREPARE s1 FROM :stmt END-EXEC.
EXEC SQL DECLARE c1 CURSOR FOR s1 END-EXEC.
EXEC SQL OPEN c1 USING :host-var END-EXEC.

*Call the FETCH and UPDATE loop.
Perform Fetch-Loop through End-Fetch-Loop
   until SQLCODE does not equal 0.

EXEC SQL CLOSE c1 END-EXEC.
EXEC SQL COMMIT END-EXEC.

Fetch-Loop Section.
    EXEC SQL FETCH c1 INTO :xmlBuff END-EXEC.
```

```
    if SQLCODE not equal 0
       go to End-Fetch-Loop.
* Display results
End-Fetch-Loop. exit.
```

Empfehlungen für die Entwicklung von Anwendungen mit eingebettetem SQL mit XML und XQuery

Für die Verwendung von XML und XQuery in Anwendungen mit eingebettetem SQL gelten folgende Empfehlungen und Einschränkungen:

- Anwendungen müssen auf alle XML-Daten im serialisierten Zeichenfolgeformat zugreifen.
  - Sie müssen alle Daten, einschließlich numerischer und Datum-/Uhrzeitdaten, in deren serialisiertem Zeichenfolgeformat darstellen.
- Die Menge der extern bereitgestellten XML-Daten ist auf 2 GB beschränkt.
- Alle Cursor, die XML-Daten enthalten, sind nicht blockierend (bei jeder Abrufoperation wird eine Anforderung des Datenbankservers erstellt).
- Immer wenn Hostzeichenvariablen serialisierte XML-Daten enthalten, wird angenommen, dass die Anwendungscodepage zur Codierung der Daten verwendet wird und mit der gegebenenfalls in den Daten vorhandenen internen Codierung übereinstimmt.
- Sie müssen einen LOB-Datentyp als Basistyp für eine XML-Hostvariable angeben.
- Für statisches SQL gelten folgende Empfehlungen und Einschränkungen:
  - XML-Werte können nicht mit Hostzeichenvariablen und binären Hostvariablen aus einer `SELECT INTO`-Operation abgerufen werden.
  - Immer wenn ein XML-Datentyp als Eingabe erwartet wird, richtet sich die Verwendung der Hostvariablen CHAR, VARCHAR, CLOB und BLOB nach einer `XMLPARSE`-Operation mit standardmäßiger Leerzeichenverarbeitung (`('STRIP WHITESPACE')`. Jeder andere Hostvariablentyp als XML wird zurückgewiesen.
  - Statische XQuery-Ausdrücke werden nicht unterstützt. Versuche zur Vorkompilierung eines XQuery-Ausdrucks schlagen mit einem Fehler fehl. Sie können XQuery-Ausdrücke nur über die Funktion XMLQUERY ausführen.
- Ein XQuery-Ausdruck kann dynamisch ausgeführt werden, indem Sie ihm die Zeichenfolge "XQUERY" voranstellen.

Angeben von XML-Werten in einem SQL-Deskriptorbereich

Um anzugeben, dass ein Basistyp XML-Daten enthält, muss das Feld 'sqlname' von SQLVAR wie folgt aktualisiert werden:

- `sqlname.length` muss eine Feldlänge von 8 haben.
- Die beiden ersten Byte von sqlname.data müssen `X'0000'` lauten.
- Das dritte und vierte Byte von sqlname.data muss `X'0000'` lauten.
- Das vierte Byte von `sqlname.data` muss `X'01'` lauten (es wird nur dann als XML-Subtypbezugswert bezeichnet, wenn die beiden ersten Bedingungen erfüllt sind).
- Die übrigen Byte müssen `X'000000'` lauten.

Wenn der XML-Subtypbezugswert in SQLVAR festgelegt ist, dessen SQLTYPE kein großes Objekt ist, wird während der Laufzeit ein SQL0804-Fehler (rc=115) zurückgegeben.

Anmerkung: SQL_TYP_XML kann nur von der Anweisung DESCRIBE zurückgegeben werden. Dieser Typ kann für keine anderen Anforderungen verwendet werden. Die Anwendung muss den SQL-Deskriptorbereich so ändern, dass er einen gültigen Zeichen- oder Binärtyp enthält, und das Feld `sqlname` so festlegen, dass angezeigt wird, dass die Daten im Format XML vorliegen.

Java

XML-Binärformat in Java-Anwendungen

Ab Version 4.9 kann IBM Data Server Driver for JDBC and SQLJ XML-Daten als XML-Binärdaten (d. h. Daten, die das Extensible Dynamic Binary XML DB2 Client/Server Binary XML Format aufweisen) an den Datenserver senden oder vom Datenserver abrufen; Voraussetzung dafür ist, dass der Datenserver XML-Binärdaten unterstützt.

IBM Data Server Driver for JDBC and SQLJ stellt die Eigenschaft 'xmlFormat' in den Schnittstellen 'DriverManager' und 'DataSource' bereit, um zu steuern, ob die Datenübertragung im XML-Textformat oder im XML-Binärformat stattfindet. Wird 'xmlFormat' auf XML_FORMAT_BINARY gesetzt, wird die Verwendung des XML-Binärformats ermöglicht. Wenn der Datenserver das XML-Binärformat unterstützt, ist XML_FORMAT_BINARY das Standardformat. Ansonsten ist XML_FORMAT_TEXTUAL das Standardformat. Das Format der XML-Daten ist für die Anwendung transparent. Für die Speicherung und den Abruf von XML-Binärdaten ist IBM Data Server Driver for JDBC and SQLJ Version 4.9 oder höher erforderlich. Wenn Sie in SQLJ-Anwendungen mit XML-Binärdaten arbeiten, ist ebenfalls das Paket 'sqlj4.zip' der Version 4.9 oder höher erforderlich.

Im folgenden Beispiel sehen Sie die Anweisungen, um unter Verwendung der Schnittstelle 'DataSource' für die Datenübertragung das XML-Binärformat festzulegen:

```
import com.ibm.db2.jcc.DB2SimpleDataSource;
...
DB2SimpleDataSource ds = new DB2SimpleDataSource();
ds.setXmlFormat(DB2BaseDataSource.XML_FORMAT_BINARY);
```

Um für die Datenübertragung das XML-Textformat festzulegen, können Sie eine Anweisung wie die folgende verwenden:

```
ds.setXmlFormat(DB2BaseDataSource.XML_FORMAT_TEXTUAL);
```

In der Schnittstelle Connection ist für die Eigenschaft 'xmlFormat' keine setXXX-Methode definiert. Aus diesem Grund müssen Sie, um den Wert für 'xmlFormat' mit der Schnittstelle Connection festzulegen, 'xmlFormat' als Eigenschaft angeben, wenn Sie die Methode DriverManager.getConnection ausführen; dies wird im folgenden Beispiel dargestellt:

```
properties.put("xmlFormat", DB2BaseDataSource.XML_FORMAT_BINARY);
DriverManager.getConnection(url, properties);
```

IBM Data Server Driver for JDBC and SQLJ präsentiert der Anwendung XML-Binärdaten nur über XML-Objektschnittstellen. Dem Benutzer werden die Daten nicht im XML-Binärformat angezeigt.

Bei der Verwendung von XML-Binärdaten dürfen die XML-Daten, die an IBM Data Server Driver for JDBC and SQLJ übergeben werden, keine externen Entitäten, internen Entitäten oder internen DTDs referenzieren. Externe DTDs werden nur unterstützt, wenn sie zuvor in der Datenquelle registriert wurden.

Das XML-Binärformat ist am effizientesten für Fälle, in denen die Ein- oder Ausgabedaten nichttextuell dargestellt werden, z. B. als SAX, StAX oder DOM. Mit den folgenden Methoden beispielsweise werden XML-Daten abgerufen, die nichttextuell dargestellt werden:
- getSource(SAXSource.class)
- getSource(StAXSource.class)
- getSource(DOMSource.class)

Mit diesen Methoden werden XML-Spalten mit Daten aktualisiert, die nichttextuell dargestellt werden:
- setResult(SAXResult.class)
- setResult(StAXResult.class)
- setResult(DOMResult.class)

Die SAX-Darstellung ist die effizienteste Methode zum Abrufen von Daten, die das XML-Binärformat aufweisen, da die Daten nicht extra aus dem Binär- in das Textformat konvertiert werden müssen.

Angenommen, Sie setzen 'xmlFormat' auf XML_FORMAT_BINARY (1). Im folgenden JDBC-Beispiel ruft IBM Data Server Driver for JDBC and SQLJ Daten im XML-Binärformat ab; die Anwendung verwendet den SAX-Parser, um die abgerufenen Daten zu parsen.

```
...
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("SELECT XMLCOL FROM XMLTABLE");
ContentHandler handler = new MyContentHandler();
while (rs.next()) {
 SQLXML sqlxml = rs.getSQLXML(1);
 SAXSource source = sqlxml.getSource(SAXSource.class);
 XMLReader reader = source.getXMLReader();
 reader.setContentHandler(handler);
 reader.parse(source.getInputSource());
}
...
```

Mit dem folgenden SQLJ-Beispiel werden dieselben Aktionen ausgeführt.

```
#sql iterator  SqlXmlIter(java.sql.SQLXML);
{
 ...
 SqlXmlIter SQLXMLiter = null;
 java.sql.SQLXML outSqlXml = null;
 ContentHandler handler = new MyContentHandler();
 #sql [ctx] SQLXmlIter = {SELECT XMLCOL FROM XMLTABLE};
 #sql {FETCH :SqlXmlIter INTO :outSqlXml};
 while (!SQLXMLIter.endFetch()) {
  SAXSource source = outSqlXml.getSource(SAXSource.class);
  XMLReader reader = source.getXMLReader();
  reader.setContentHandler(handler);
  reader.parse(source.getInputSource());
  #sql {FETCH :SqlXmlIter INTO :outSqlXml};
 }
 ...
}
```

JDBC

XML-Daten in JDBC-Anwendungen

In JDBC-Anwendungen können Sie Daten in XML-Spalten speichern und Daten aus XML-Spalten abrufen.

In Datenbanktabellen werden mithilfe des integrierten XML-Datentyps XML-Daten in einer Spalte als strukturierte Gruppe von Knoten im Format einer Baumstruktur gespeichert.

JDBC-Anwendungen können XML-Daten in einem der folgenden Formate an den Datenserver senden oder vom Datenserver abrufen:

- Als XML-Textdaten
- Als XML-Binärdaten, sofern die Unterstützung durch den Datenserver hierfür gegeben ist

In JDBC-Anwendungen haben Sie folgende Möglichkeiten:

- Speichern eines vollständigen XML-Dokuments in einer XML-Spalte mithilfe von setXXX-Methoden.
- Abrufen eines vollständigen XML-Dokuments aus einer XML-Spalte mithilfe von getXXX-Methoden.
- Abrufen einer Sequenz aus einem Dokument in einer XML-Spalte mit der SQL-Funktion XMLQUERY zum Abrufen der Sequenz in eine serialisierte Sequenz in der Datenbank und zum nachfolgenden Abrufen der Daten in eine Anwendungsvariable mithilfe von getXXX-Methoden.
- Abrufen einer Sequenz aus einem Dokument in einer XML-Spalte mit einem XQuery-Ausdruck, dem die Zeichenfolge 'XQUERY' vorangestellt ist, zum Abrufen der Elemente der Sequenz in eine Ergebnistabelle in der Datenbank, in der jede Zeile der Ergebnistabelle ein Element in der Sequenz darstellt. Anschließend können Sie die Daten mithilfe von getXXX-Methoden in Anwendungsvariablen abrufen.
- Abrufen einer Sequenz aus einem Dokument in einer XML-Spalte als benutzerdefinierte Tabelle mit der SQL-Funktion XMLTABLE zum Definieren und Abrufen der Ergebnistabelle. Anschließend können Sie die Daten mithilfe von getXXX-Methoden aus der Ergebnistabelle in Anwendungsvariablen abrufen.

Mit den JDBC 4.0-Objekten `java.sql.SQLXML` können Daten in XML-Spalten abgerufen und aktualisiert werden. Beim Aufruf einer Metadatenmethode wie ResultSetMetaData.getColumnTypeName wird der ganzzahlige Wert `java.sql.Types.SQLXML` für einen XML-Spaltentyp zurückgegeben.

XML-Spaltenaktualisierungen in JDBC-Anwendungen

In einer JDBC-Anwendung können Sie Daten in XML-Spalten einer Tabelle auf einem DB2-Datenserver mithilfe von XML-Textdaten aktualisieren oder einfügen. Wenn der Datenserver XML-Daten mit Extensible Dynamic Binary XML DB2 Client/Server Binary XML Format unterstützt, können Sie Daten unter Verwendung dieses Binärformats in XML-Spalten aktualisieren oder einfügen.

In der folgenden Tabelle sind die Methoden und die entsprechenden Eingabedatentypen aufgeführt, mit denen Sie Daten in XML-Spalten einfügen können.

Tabelle 28. Methoden und Datentypen zur Aktualisierung von XML-Spalten

Methode	Eingabedatentyp
PreparedStatement.setAsciiStream	InputStream
PreparedStatement.setBinaryStream	InputStream
PreparedStatement.setBlob	Blob
PreparedStatement.setBytes	byte[]
PreparedStatement.setCharacterStream	Reader

Tabelle 28. Methoden und Datentypen zur Aktualisierung von XML-Spalten (Forts.)

Methode	Eingabedatentyp
PreparedStatement.setClob	Clob
PreparedStatement.setObject	byte[], Blob, Clob, SQLXML, DB2Xml (veraltet), InputStream, Reader, String
PreparedStatement.setSQLXML[1]	SQLXML
PreparedStatement.setString	String

Anmerkung:

1. Für diese Methode ist mindestens JDBC 4.0 erforderlich.

Die Codierung von XML-Daten kann aus den Daten selbst abgeleitet werden (*intern codierte* Daten) oder aus externen Quellen (*extern codierte* Daten). Als Binärdaten zum Datenbankserver gesendete XML-Daten werden wie intern codierte Daten behandelt. Als Zeichendaten gesendete XML-Daten werden als extern codierte Daten angesehen.

Die externe Codierung für Java-Anwendungen ist stets Unicode.

Extern codierte Daten können eine interne Codierung haben. Die Daten können also als Zeichendaten an die Datenquelle gesendet werden, enthalten jedoch Codierungsinformationen. Die Datenquelle behandelt Inkompatibilitäten zwischen interner und externer Codierung wie folgt:

- Wenn die Datenquelle DB2 Database for Linux, UNIX and Windows ist, generiert sie einen Fehler, wenn die externe und die interne Codierung nicht kompatibel sind, es sei denn, die externe und interne Codierung ist Unicode. Ist die externe und interne Codierung Unicode, ignoriert die Datenquelle die interne Codierung.
- Wenn die Datenquelle DB2 for z/OS ist, ignoriert die Datenquelle die interne Codierung.

Daten in XML-Spalten werden in der Codierung UTF-8 gespeichert. Die Datenquelle verarbeitet die Konvertierung der Daten aus ihrer internen oder externen Codierung in UTF-8.

Beispiel: Das folgende Beispiel zeigt, wie Daten aus einem SQLXML-Objekt in eine XML-Spalte eingefügt werden. Die Daten sind Zeichenfolgedaten, sodass die Datenquelle sie wie extern codierte Daten behandelt.

```
public void insertSQLXML()
{
 Connection con = DriverManager.getConnection(url);
  SQLXML info = con.createSQLXML();
                            // Create an SQLXML object
  PreparedStatement insertStmt = null;
  String infoData =
    "<customerinfo xmlns=""http://posample.org"" " +
    "Cid=""1000"">...</customerinfo>";
  info.setString(infoData);
                            // Populate the SQLXML object
  int cid = 1000;
  try {
    sqls = "INSERT INTO CUSTOMER (CID, INFO) VALUES (?, ?)";
    insertStmt = con.prepareStatement(sqls);
    insertStmt.setInt(1, cid);
    insertStmt.setSQLXML(2, info);
                            // Assign the SQLXML object value
```

```
                // to an input parameter
      if (insertStmt.executeUpdate() != 1) {
        System.out.println("insertSQLXML: No record inserted.");
      }
    }
    catch (IOException ioe) {
     ioe.printStackTrace();
    }
    catch (SQLException sqle) {
      System.out.println("insertSQLXML: SQL Exception: " +
        sqle.getMessage());
      System.out.println("insertSQLXML: SQL State: " +
        sqle.getSQLState());
      System.out.println("insertSQLXML: SQL Error Code: " +
        sqle.getErrorCode());

    }
  }
```

Beispiel: Das folgende Beispiel zeigt, wie Daten aus einer Datei in eine XML-Spalte eingefügt werden. Die Daten werden als Binärdaten eingefügt, sodass der Datenbankserver die interne Codierung berücksichtigt.

```
  public void insertBinStream(Connection conn)
  {
    PreparedStatement insertStmt = null;
    String sqls = null;
    int cid = 0;
    Statement stmt=null;
    try {
      sqls = "INSERT INTO CUSTOMER (CID, INFO) VALUES (?, ?)";
      insertStmt = conn.prepareStatement(sqls);
      insertStmt.setInt(1, cid);
      File file = new File(fn);
      insertStmt.setBinaryStream(2, new FileInputStream(file), (int)file.length());
      if (insertStmt.executeUpdate() != 1) {
        System.out.println("insertBinStream: No record inserted.");
      }
    }
    catch (IOException ioe) {
     ioe.printStackTrace();
    }
    catch (SQLException sqle) {
      System.out.println("insertBinStream: SQL Exception: " +
        sqle.getMessage());
      System.out.println("insertBinStream: SQL State: " +
        sqle.getSQLState());
      System.out.println("insertBinStream: SQL Error Code: " +
        sqle.getErrorCode());

    }
  }
```

Beispiel: Das folgende Beispiel zeigt, wie XML-Binärdaten aus einer Datei in eine XML-Spalte eingefügt werden.

```
...
SQLXML info = conn.createSQLXML();
OutputStream os = info.setBinaryStream ();
FileInputStream fis = new FileInputStream("c7.xml");
int read;
while ((read = fis.read ()) != -1) {
  os.write (read);
}

PreparedStatement insertStmt = null;
String sqls = null;
```

```
int cid = 1015;
sqls = "INSERT INTO MyCustomer (Cid, Info) VALUES (?, ?)";
insertStmt = conn.prepareStatement(sqls);
insertStmt.setInt(1, cid);
insertStmt.setSQLXML(2, info);
insertStmt.executeUpdate();
```

Abrufen von XML-Daten in JDBC-Anwendungen

In JDBC-Anwendungen rufen Sie Daten mit ResultSet.getXXX- oder ResultSet.getObject-Methoden aus XML-Spalten ab.

In einer JDBC-Anwendung können Sie Daten aus XML-Spalten in einer DB2-Tabelle als XML-Textdaten abrufen. Sofern der Datenserver XML-Binärdaten unterstützt, können Sie Daten aus XML-Spalten in einer Tabelle als XML-Binärdaten abrufen (d. h. Daten, die das Extensible Dynamic Binary XML DB2 Client/Server Binary XML Format aufweisen).

Sie können die XML-Daten mit einem der folgenden Verfahren abrufen:

- Rufen Sie die Daten mit der ResultSet.getSQLXML-Methode ab. Rufen Sie die Daten anschließend mit einer SQLXML.getXXX-Methode in einen kompatiblen Ausgabedatentyp ab. Für dieses Verfahren ist mindestens JDBC 4.0 erforderlich.

  So können Sie beispielsweise Daten mithilfe der SQLXML.getBinaryStream- oder der SQLXML.getSource-Methode abrufen.
- Rufen Sie die Daten mit einer anderen ResultSet.getXXX-Methode als ResultSet.getObject in einen kompatiblen Ausgabedatentyp ab.
- Rufen Sie die Daten mit der ResultSet.getObject-Methode ab, setzen Sie sie anschließend in den Typ DB2Xml um, und weisen Sie sie einem DB2Xml-Objekt zu. Rufen Sie die Daten anschließend mit einer DB2Xml.getDB2XXX- oder DB2Xml.getDB2XmlXXX-Methode in einen kompatiblen Ausgabedatentyp ab.

  Sie müssen dieses Verfahren anwenden, wenn Sie keine Version von IBM Data Server Driver for JDBC and SQLJ verwenden, die JDBC 4.0 unterstützt.

In der folgenden Tabelle sind die ResultSet-Methoden und die entsprechenden Ausgabedatentypen zum Abrufen von XML-Daten aufgeführt.

Tabelle 29. ResultSet-Methoden und Datentypen zum Abrufen von XML-Daten

Methode	Ausgabedatentyp
ResultSet.getAsciiStream	InputStream
ResultSet.getBinaryStream	InputStream
ResultSet.getBytes	byte[]
ResultSet.getCharacterStream	Reader
ResultSet.getObject	Object
ResultSet.getSQLXML	SQLXML
ResultSet.getString	String

In der folgenden Tabelle sind die Methoden aufgeführt, durch deren Aufruf Sie Daten aus java.sql.SQLXML- oder com.ibm.db2.jcc.DB2Xml-Objekten abrufen, sowie die entsprechenden Ausgabedatentypen und die Codierungstypen in den XML-Deklarationen.

Tabelle 30. SQLXML- und DB2Xml-Methoden, Datentypen und hinzugefügte Codierungsangaben

Methode	Ausgabedatentyp	Hinzugefügter Typ der XML-Deklaration für interne Codierung
SQLXML.getBinaryStream	InputStream	Keine
SQLXML.getCharacterStream	Reader	Keine
SQLXML.getSource	Source[1]	Keine
SQLXML.getString	String	Keine
DB2Xml.getDB2AsciiStream	InputStream	Keine
DB2Xml.getDB2BinaryStream	InputStream	Keine
DB2Xml.getDB2Bytes	byte[]	Keine
DB2Xml.getDB2CharacterStream	Reader	Keine
DB2Xml.getDB2String	String	Keine
DB2Xml.getDB2XmlAsciiStream	InputStream	US-ASCII
DB2Xml.getDB2XmlBinaryStream	InputStream	Mit dem Parameter *targetEncoding* von getDB2XmlBinaryStream angegeben
DB2Xml.getDB2XmlBytes	byte[]	Mit dem Parameter *targetEncoding* von DB2Xml.getDB2XmlBytes angegeben
DB2Xml.getDB2XmlCharacterStream	Reader	ISO-10646-UCS-2
DB2Xml.getDB2XmlString	String	ISO-10646-UCS-2

Anmerkung:

1. Die zurückgegebene Klasse wird vom Aufrufer von getSource angegeben, die Klasse muss jedoch javax.xml.transform.Source erweitern.

Wenn die Anwendung die Funktion XMLSERIALIZE für die Daten ausführt, die zurückgegeben werden sollen, haben die Daten nach der Ausführung der Funktion den Datentyp, der in der Funktion XMLSERIALIZE angegeben ist, nicht den XML-Datentyp. Daher behandelt der Treiber die Daten wie den angegebenen Typ und ignoriert interne Codierungsdeklarationen.

Beispiel: Das folgende Beispiel zeigt, wie Daten aus einer XML-Spalte in ein SQLXML-Objekt und anschließend mit der Methode SQLXML.getString in eine Zeichenfolge abgerufen werden.

```
public void fetchToSQLXML(long cid, java.sql.Connection conn)
  {
    System.out.println(">> fetchToSQLXML: Get XML data as an SQLXML object " +
    "using getSQLXML");
    PreparedStatement selectStmt = null;
    String sqls = null, stringDoc = null;
    ResultSet rs = null;

    try{
      sqls = "SELECT info FROM customer WHERE cid = " + cid;
      selectStmt = conn.prepareStatement(sqls);
      rs = selectStmt.executeQuery();

      // Get metadata
      // Column type for XML column is the integer java.sql.Types.OTHER
      ResultSetMetaData meta = rs.getMetaData();
      int colType = meta.getColumnType(1);
      System.out.println("fetchToSQLXML: Column type = " + colType);
      while (rs.next()) {
        // Retrieve the XML data with getSQLXML.
        // Then write it to a string with
```

```
        // explicit internal ISO-10646-UCS-2 encoding.
        java.sql.SQLXML xml = rs.getSQLXML(1);
        System.out.println (xml.getString());
      }
      rs.close();
    }
    catch (SQLException sqle) {
      System.out.println("fetchToSQLXML: SQL Exception: " +
        sqle.getMessage());
      System.out.println("fetchToSQLXML: SQL State: " +
        sqle.getSQLState());
      System.out.println("fetchToSQLXML: SQL Error Code: " +
        sqle.getErrorCode());
    }
  }
```

Beispiel: Das folgende Beispiel zeigt, wie Daten aus einer XML-Spalte in ein SQLXML-Objekt und anschließend mit der Methode SQLXML.getBinaryStream als Binärdaten in einen Eingabedatenstrom (InputStream) abgerufen werden.

```
String sql = "SELECT INFO FROM Customer WHERE Cid='1000'";
PreparedStatement pstmt = con.prepareStatement(sql);
ResultSet resultSet = pstmt.executeQuery();
// Get the result XML as a binary stream
SQLXML sqlxml = resultSet.getSQLXML(1);
InputStream binaryStream = sqlxml.getBinaryStream();
```

Beispiel: Das folgende Beispiel zeigt, wie Daten aus einer XML-Spalte in eine Zeichenfolgevariable abgerufen werden.

```
public void fetchToString(long cid, java.sql.Connection conn)
  {
    System.out.println(">> fetchToString: Get XML data " +
    "using getString");
    PreparedStatement selectStmt = null;
    String sqls = null, stringDoc = null;
    ResultSet rs = null;

    try{
      sqls = "SELECT info FROM customer WHERE cid = " + cid;
      selectStmt = conn.prepareStatement(sqls);
      rs = selectStmt.executeQuery();

      // Get metadata
      // Column type for XML column is the integer java.sql.Types.OTHER
      ResultSetMetaData meta = rs.getMetaData();
      int colType = meta.getColumnType(1);
      System.out.println("fetchToString: Column type = " + colType);

      while (rs.next()) {
        stringDoc = rs.getString(1);
        System.out.println("Document contents:");
        System.out.println(stringDoc);
 }
    catch (SQLException sqle) {
      System.out.println("fetchToString: SQL Exception: " +
        sqle.getMessage());
      System.out.println("fetchToString: SQL State: " +
        sqle.getSQLState());
      System.out.println("fetchToString: SQL Error Code: " +
        sqle.getErrorCode());
    }
  }
```

Beispiel: Das folgende Beispiel zeigt, wie Daten aus einer XML-Spalte in ein DB2Xml-Objekt und anschließend mit der Methode DB2Xml.getDB2XmlString in

eine Zeichenfolge mit einer hinzugefügten XML-Deklaration mit einer ISO 10646 UCS-2-Codierungsspezifikation abgerufen werden.

```
public void fetchToDB2Xml(long cid, java.sql.Connection conn)
  {
    System.out.println(">> fetchToDB2Xml: Get XML data as a DB2XML object " +
    "using getObject");
    PreparedStatement selectStmt = null;
    String sqls = null, stringDoc = null;
    ResultSet rs = null;

    try{
      sqls = "SELECT info FROM customer WHERE cid = " + cid;
      selectStmt = conn.prepareStatement(sqls);
      rs = selectStmt.executeQuery();

      // Get metadata
      // Column type for XML column is the integer java.sql.Types.OTHER
      ResultSetMetaData meta = rs.getMetaData();
      int colType = meta.getColumnType(1);
      System.out.println("fetchToDB2Xml: Column type = " + colType);
      while (rs.next()) {
        // Retrieve the XML data with getObject, and cast the object
        // as a DB2Xml object. Then write it to a string with
        // explicit internal ISO-10646-UCS-2 encoding.
        com.ibm.db2.jcc.DB2Xml xml =
          (com.ibm.db2.jcc.DB2Xml) rs.getObject(1);
        System.out.println (xml.getDB2XmlString());
      }
      rs.close();
    }
    catch (SQLException sqle) {
      System.out.println("fetchToDB2Xml: SQL Exception: " +
        sqle.getMessage());
      System.out.println("fetchToDB2Xml: SQL State: " +
        sqle.getSQLState());
      System.out.println("fetchToDB2Xml: SQL Error Code: " +
        sqle.getErrorCode());
    }
  }
```

Aufruf von Routinen mit XML-Parametern in Java-Anwendungen

Java-Anwendungen können gespeicherte Prozeduren aus DB2 Database for Linux, UNIX and Windows- oder DB2 for z/OS-Datenquellen mit XML-Parametern aufrufen.

Bei nativen SQL-Prozeduren haben XML-Parameter in der Definition der gespeicherten Prozedur den Datentyp XML. Bei externen gespeicherten Prozeduren und benutzerdefinierten Funktionen in DB2 Database for Linux, UNIX and Windows-Datenquellen haben XML-Parameter in der Definition der Routine den Datentyp XML AS CLOB. Wenn Sie eine gespeicherte Prozedur oder benutzerdefinierte Funktion aufrufen, die XML-Parameter hat, müssen Sie in der aufrufenden Anweisung einen kompatiblen Datentyp verwenden.

Verwenden Sie zum Aufruf einer Routine mit XML-Eingabeparametern aus einem JDBC-Programm Parameter des Typs java.sql.SQLXML oder com.ibm.db2.jcc.DB2Xml. Registrieren Sie XML-Ausgabeparameter als java.sql.Types.SQLXML oder com.ibm.db2.jcc.DB2Types.XML. (Die Datentypen com.ibm.db2.jcc.DB2Xml und com.ibm.db2.jcc.DB2Types.XML sind veraltet.)

Beispiel: JDBC-Programm, das eine gespeicherte Prozedur aufruft, die drei XML-Parameter annimmt: IN, OUT und INOUT. Für dieses Beispiel ist mindestens JDBC 4.0 erforderlich.

```
java.sql.SQLXML in_xml = xmlvar;
java.sql.SQLXML out_xml = null;
java.sql.SQLXML inout_xml = xmlvar;
                                  // Declare an input, output, and
                                  // INOUT XML parameter
Connection con;
CallableStatement cstmt;
ResultSet rs;
...
cstmt = con.prepareCall("CALL SP_xml(?,?,?)");
                                  // Create a CallableStatement object
cstmt.setObject (1, in_xml);      // Set input parameter
cstmt.setObject (3, inout_xml);   // Set inout parameter
cstmt.registerOutParameter (2, java.sql.Types.SQLXML);
                                  // Register out and input parameters
cstmt.registerOutParameter (3, java.sql.Types.SQLXML);
cstmt.executeUpdate();            // Call the stored procedure
out_xml = cstmt.getSQLXML(2);     // Get the OUT parameter value
inout_xml = cstmt.getSQLXML(3);   // Get the INOUT parameter value
System.out.println("Parameter values from SP_xml call: ");
System.out.println("Output parameter value ");
MyUtilities.printString(out_xml.getString());
                          // Use the SQLXML.getString
                          // method to convert the out_xml
                          // value to a string for printing.
                          // Call a user-defined method called
                          // printString (not shown) to print
                          // the value.
System.out.println("INOUT parameter value ");
MyUtilities.printString(inout_xml.getString());
                          // Use the SQLXML.getString
                          // method to convert the inout_xml
                          // value to a string for printing.
                          // Call a user-defined method called
                          // printString (not shown) to print
                          // the value.
```

Verwenden Sie zum Aufruf einer Routine mit XML-Parametern aus einem SQLJ-Programm Parameter des Typs java.sql.SQLXML oder com.ibm.db2.jcc.DB2Xml.

Beispiel: SQLJ-Programm, das eine gespeicherte Prozedur aufruft, die drei XML-Parameter annimmt: IN, OUT und INOUT. Für dieses Beispiel ist mindestens JDBC 4.0 erforderlich.

```
java.sql.SQLXML in_xml = xmlvar;
java.sql.SQLXML out_xml = null;
java.sql.SQLXML inout_xml = xmlvar;
                                  // Declare an input, output, and
                                  // INOUT XML parameter
...
#sql [myConnCtx] {CALL SP_xml(:IN in_xml,
                              :OUT out_xml,
                              :INOUT inout_xml)};
                                  // Call the stored procedure
System.out.println("Parameter values from SP_xml call: ");
System.out.println("Output parameter value ");
MyUtilities.printString(out_xml.getString());
                          // Use the SQLXML.getString
                          // method toconvert the out_xml value
                          // to a string for printing.
                          // Call a user-defined method called
                          // printString (not shown) to print
                          // the value.
System.out.println("INOUT parameter value ");
MyUtilities.printString(inout_xml.getString());
                          // Use the SQLXML.getString
```

```
// method to convert the inout_xml
// value to a string for printing.
// Call a user-defined method called
// printString (not shown) to print
// the value.
```

SQLJ

XML-Daten in SQLJ-Anwendungen

In SQLJ-Anwendungen können Sie Daten in XML-Spalten speichern und aus XML-Spalten abrufen.

In DB2-Tabellen werden mithilfe des integrierten XML-Datentyps XML-Daten in einer Spalte als strukturierte Gruppe von Knoten im Format einer Baumstruktur gespeichert.

SQLJ-Anwendungen können XML-Daten in einem der folgenden Formate an den Datenserver senden oder vom Datenserver abrufen:

- Als XML-Textdaten
- Als XML-Binärdaten (d. h. Daten, die das Extensible Dynamic Binary XML DB2 Client/Server Binary XML Format aufweisen), sofern die Unterstützung durch den Datenserver hierfür gegeben ist

In SQLJ-Anwendungen haben Sie folgende Möglichkeiten:

- Speichern eines vollständigen XML-Dokuments in einer XML-Spalte mithilfe von INSERT-, UPDATE- oder MERGE-Anweisungen.
- Abrufen eines vollständigen XML-Dokuments aus einer XML-Spalte mithilfe von SELECT-Anweisungen für eine einzelne Zeile oder mit Iteratoren.
- Abrufen einer Sequenz aus einem Dokument in einer XML-Spalte mit der SQL-Funktion XMLQUERY zum Abrufen der Sequenz in der Datenbank und zum nachfolgenden Abrufen der serialisiertes XML-Zeichenfolgedaten in eine Anwendungsvariable mithilfe von SELECT-Anweisungen für eine einzelne Zeile oder mit Iteratoren.
- Abrufen einer Sequenz aus einem Dokument in einer XML-Spalte mit einem XQuery-Ausdruck, dem die Zeichenfolge 'XQUERY' vorangestellt ist, zum Abrufen der Elemente der Sequenz in eine Ergebnistabelle in der Datenbank, in der jede Zeile der Ergebnistabelle ein Element in der Sequenz darstellt. Anschließend können Sie die Daten mithilfe von SELECT-Anweisungen für eine einzelne Zeile oder mit Iteratoren in Anwendungsvariablen abrufen.
- Abrufen einer Sequenz aus einem Dokument in einer XML-Spalte als benutzerdefinierte Tabelle mit der SQL-Funktion XMLTABLE zum Definieren und Abrufen der Ergebnistabelle. Anschließend können Sie die Daten aus der Ergebnistabelle mithilfe von SELECT-Anweisungen für eine einzelne Zeile oder mit Iteratoren in Anwendungsvariablen abrufen.
- Sie können XML-Daten als XML-Textdaten aktualisieren oder abrufen. Alternativ dazu können Sie für Verbindungen zu einem Datenserver, der XML-Binärdaten unterstützt, XML-Daten als XML-Binärdaten aktualisieren oder abrufen. Zur Steuerung des Datenformats (XML-Textformat oder XML-Binärformat) verwenden Sie die Eigenschaft Datasource bzw. Connection für 'xmlFormat'. Das Format der XML-Daten ist für die Anwendung transparent. Für die Speicherung und den Abruf von XML-Binärdaten auf einem DB2 for z/OS-Datenserver ist IBM Data Server Driver for JDBC and SQLJ Version 4.9 oder höher erforderlich. Für die Speicherung und den Abruf von XML-Binärdaten auf einem DB2 Database

for Linux, UNIX and Windows-Datenserver ist IBM Data Server Driver for JDBC and SQLJ Version 4.11 oder höher erforderlich.

Mit den JDBC 4.0-Objekten `java.sql.SQLXML` können Daten in XML-Spalten abgerufen und aktualisiert werden. Beim Aufruf einer Metadatenmethode wie ResultSetMetaData.getColumnType wird der ganzzahlige Wert `java.sql.Types.SQLXML` für einen XML-Spaltentyp zurückgegeben.

Aktualisierungen von XML-Spalten in SQLJ-Anwendungen

In einer SQLJ-Anwendung können Sie Daten in XML-Spalten einer Tabelle auf einem DB2-Datenserver mithilfe von XML-Textdaten aktualisieren oder einfügen. Sofern der Datenserver XML-Binärdaten unterstützt, können Sie Daten in XML-Spalten einer Tabelle mithilfe von XML-Binärdaten (d. h. Daten, die das Extensible Dynamic Binary XML DB2 Client/Server Binary XML Format aufweisen), aktualisieren oder einfügen.

Sie können zum Aktualisieren von XML-Spalten folgende Datentypen für Hostausdrücke verwenden:

- java.sql.SQLXML (erfordert SDK für Java Version 6 oder höher und IBM Data Server Driver for JDBC and SQLJ Version 4.0 oder höher)
- com.ibm.db2.jcc.DB2Xml (veraltet)
- String
- byte
- Blob
- Clob
- sqlj.runtime.AsciiStream
- sqlj.runtime.BinaryStream
- sqlj.runtime.CharacterStream

Die Codierung von XML-Daten kann aus den Daten selbst abgeleitet werden (*intern codierte* Daten) oder aus externen Quellen (*extern codierte* Daten). Als Binärdaten zum Datenbankserver gesendete XML-Daten werden wie intern codierte Daten behandelt. Als Zeichendaten gesendete XML-Daten werden als extern codierte Daten angesehen. Die externe Codierung ist die Standardcodierung für die JVM.

Die externe Codierung für Java-Anwendungen ist stets Unicode.

Extern codierte Daten können eine interne Codierung haben. Die Daten können also als Zeichendaten an die Datenquelle gesendet werden, enthalten jedoch Codierungsinformationen. Die Datenquelle behandelt Inkompatibilitäten zwischen interner und externer Codierung wie folgt:

- Wenn die Datenquelle DB2 Database for Linux, UNIX and Windows ist, generiert sie einen Fehler, wenn die externe und die interne Codierung nicht kompatibel sind, es sei denn, die externe und interne Codierung ist Unicode. Ist die externe und interne Codierung Unicode, ignoriert die Datenquelle die interne Codierung.
- Wenn die Datenquelle DB2 for z/OS ist, ignoriert die Datenquelle die interne Codierung.

Daten in XML-Spalten werden in der Codierung UTF-8 gespeichert.

Beispiel: Sie fügen Daten aus dem String-Hostausdruck xmlString in eine XML-Spalte einer Tabelle ein. xmlString ist ein Zeichentyp, daher wird seine externe Codierung unabhängig davon verwendet, ob er eine interne Codierungsangabe hat.

```
#sql [ctx] {INSERT INTO CUSTACC VALUES (1, :xmlString)};
```

Beispiel: Sie kopieren die Daten aus xmlString in eine Bytefeldgruppe mit der Codierung CP500. Die Daten enthalten eine XML-Deklaration mit einer Codierungsdeklaration für CP500. Dann fügen Sie Daten aus dem byte[]-Hostausdruck in eine XML-Spalte einer Tabelle ein.

```
byte[] xmlBytes = xmlString.getBytes("CP500");
#sql[ctx] {INSERT INTO CUSTACC VALUES (4, :xmlBytes)};
```

Eine Bytefolge wird als intern codierte Daten betrachtet. Die Daten werden bei Bedarf aus ihrem internen Codierungsschema in UTF-8 konvertiert und in ihrem hierarchischen Format in der Datenquelle gespeichert.

Beispiel: Sie kopieren die Daten aus xmlString in eine Bytefeldgruppe mit der Codierung US-ASCII. Dann erstellen Sie einen Hostausdruck vom Typ sqlj.runtime.AsciiStream und fügen Daten aus sqlj.runtime.AsciiStream in eine XML-Spalte einer Tabelle einer Datenquelle ein.

```
byte[] b = xmlString.getBytes("US-ASCII");
java.io.ByteArrayInputStream xmlAsciiInputStream =
  new java.io.ByteArrayInputStream(b);
sqlj.runtime.AsciiStream sqljXmlAsciiStream =
  new sqlj.runtime.AsciiStream(xmlAsciiInputStream,  b.length);
#sql[ctx] {INSERT INTO CUSTACC VALUES (4, :sqljXmlAsciiStream)};
```

sqljXmlAsciiStream ist ein Datenstromtyp, daher wird seine interne Codierung verwendet. Die Daten werden aus ihrer internen Codierung in UTF-8 konvertiert und in ihrem hierarchischen Format in der Datenquelle gespeichert.

Beispiel: sqlj.runtime.CharacterStream-Hostausdruck: Sie erstellen einen Hostausdruck vom Typ sqlj.runtime.CharacterStream und fügen Daten aus sqlj.runtime.CharacterStream in eine XML-Spalte einer Tabelle ein.

```
java.io.StringReader xmlReader =
  new java.io.StringReader(xmlString);
sqlj.runtime.CharacterStream sqljXmlCharacterStream =
  new sqlj.runtime.CharacterStream(xmlReader,  xmlString.length());
#sql [ctx] {INSERT INTO CUSTACC VALUES (4, :sqljXmlCharacterStream)};
```

sqljXmlCharacterStream ist ein Zeichentyp, daher wird seine externe Codierung unabhängig davon verwendet, ob er eine interne Codierungsangabe hat.

Beispiel: Sie rufen ein Dokument aus einer XML-Spalte in einen Hostausdruck vom Typ java.sql.SQLXML ab und fügen die Daten in eine XML-Spalte einer Tabelle ein.

```
java.sql.ResultSet rs = s.executeQuery ("SELECT * FROM CUSTACC");
rs.next();
java.sql.SQLXML xmlObject = (java.sql.SQLXML)rs.getObject(2);
#sql [ctx] {INSERT INTO CUSTACC VALUES (6, :xmlObject)};
```

Die Daten liegen nach dem Abruf weiterhin in der Codierung UTF-8 vor, sodass beim Einfügen der Daten in eine andere XML-Spalte keine Konvertierung erfolgt.

Beispiel: Sie rufen ein Dokument aus einer XML-Spalte in einen Hostausdruck vom Typ com.ibm.db2.jcc.DB2Xml ab und fügen die Daten in eine XML-Spalte einer Tabelle ein.

```
java.sql.ResultSet rs = s.executeQuery ("SELECT * FROM CUSTACC");
rs.next();
com.ibm.db2.jcc.DB2Xml xmlObject = (com.ibm.db2.jcc.DB2Xml)rs.getObject(2);
#sql [ctx] {INSERT INTO CUSTACC VALUES (6, :xmlObject)};
```

Die Daten liegen nach dem Abruf weiterhin in der Codierung UTF-8 vor, sodass beim Einfügen der Daten in eine andere XML-Spalte keine Konvertierung erfolgt.

Abrufen von XML-Daten in SQLJ-Anwendungen

Wenn Sie Daten aus XML-Spalten einer Datenbanktabelle in einer SQLJ-Anwendung abrufen, müssen die Ausgabedaten explizit oder implizit serialisiert sein.

Sie können zum Abrufen von XML-Spalten folgende Datentypen für Hostausdrücke oder Iteratoren verwenden:

- java.sql.SQLXML (erfordert SDK für Java Version 6 oder höher und IBM Data Server Driver for JDBC and SQLJ Version 4.0 oder höher)
- com.ibm.db2.jcc.DB2Xml (veraltet)
- String
- byte[]
- sqlj.runtime.AsciiStream
- sqlj.runtime.BinaryStream
- sqlj.runtime.CharacterStream

Wenn die Anwendung vor dem Datenabruf nicht die Funktion XMLSERIALIZE aufruft, werden die Daten aus UTF-8 in die externe Anwendungscodierung (bei Zeichendatentypen) oder die interne Codierung (bei Binärdatentypen) konvertiert. Es wird keine XML-Deklaration hinzugefügt. Wenn der Hostausdruck ein Objekt des Typs java.sql.SQLXML oder com.ibm.db2.jcc.DB2Xml ist, müssen Sie eine zusätzliche Methode aufrufen, um die Daten aus diesem Objekt abzurufen. Die von Ihnen aufgerufene Methode ermittelt, welche Codierung die Ausgabedaten haben und ob eine XML-Deklaration mit einer Codierungsangabe hinzugefügt wird.

In der folgenden Tabelle sind die Methoden aufgeführt, durch deren Aufruf Sie Daten aus java.sql.SQLXML- oder com.ibm.db2.jcc.DB2Xml-Objekten abrufen, sowie die entsprechenden Ausgabedatentypen und die Codierungstypen in den XML-Deklarationen.

Tabelle 31. SQLXML- und DB2Xml-Methoden, Datentypen und hinzugefügte Codierungsangaben

Methode	Ausgabedatentyp	Hinzugefügter Typ der XML-Deklaration für interne Codierung
SQLXML.getBinaryStream	InputStream	Keine
SQLXML.getCharacterStream	Reader	Keine
SQLXML.getSource	Source	Keine
SQLXML.getString	String	Keine
DB2Xml.getDB2AsciiStream	InputStream	Keine
DB2Xml.getDB2BinaryStream	InputStream	Keine
DB2Xml.getDB2Bytes	byte[]	Keine
DB2Xml.getDB2CharacterStream	Reader	Keine
DB2Xml.getDB2String	String	Keine
DB2Xml.getDB2XmlAsciiStream	InputStream	US-ASCII
DB2Xml.getDB2XmlBinaryStream	InputStream	Mit dem Parameter *targetEncoding* von getDB2XmlBinaryStream angegeben
DB2Xml.getDB2XmlBytes	byte[]	Mit dem Parameter *targetEncoding* von DB2Xml.getDB2XmlBytes angegeben
DB2Xml.getDB2XmlCharacterStream	Reader	ISO-10646-UCS-2
DB2Xml.getDB2XmlString	String	ISO-10646-UCS-2

Wenn die Anwendung die Funktion XMLSERIALIZE für die Daten ausführt, die zurückgegeben werden sollen, haben die Daten nach der Ausführung der Funktion den Datentyp, der in der Funktion XMLSERIALIZE angegeben ist, nicht den XML-Datentyp. Daher behandelt der Treiber die Daten wie den angegebenen Typ und ignoriert interne Codierungsdeklarationen.

Beispiel: Sie rufen Daten aus einer XML-Spalte in einen String-Hostausdruck ab.

```
#sql iterator  XmlStringIter (int, String);
#sql [ctx] siter = {SELECT C1, CADOC from CUSTACC};
#sql {FETCH :siter INTO :row, :outString};
```

Der Datentyp 'String' ist ein Zeichentyp, sodass die Daten aus UTF-8 in die externe Codierung (die JVM-Standardkonvertierung) konvertiert und ohne XML-Deklaration zurückgegeben werden.

Beispiel: Sie rufen Daten aus einer XML-Spalte in einen byte[]-Hostausdruck ab.

```
#sql iterator  XmlByteArrayIter (int, byte[]);
XmlByteArrayIter biter = null;
#sql [ctx] biter =  {SELECT c1, CADOC from CUSTACC};
#sql {FETCH :biter INTO :row, :outBytes};
```

Der Datentyp 'byte[]' ist ein binärer Typ, sodass keine Datenkonvertierung erfolgt und die Daten ohne XML-Deklaration zurückgegeben werden.

Beispiel: Sie rufen ein Dokument aus einer XML-Spalte in einen Hostausdruck vom Typ java.sql.SQLXML ab, benötigen die Daten jedoch in einem Binärdatenstrom.

```
#sql iterator  SqlXmlIter (int, java.sql.SQLXML);
SqlXmlIter SQLXMLiter = null;
java.sql.SQLXML outSqlXml = null;
#sql [ctx] SqlXmlIter = {SELECT c1, CADOC from CUSTACC};
#sql {FETCH :SqlXmlIter INTO :row, :outSqlXml};
java.io.InputStream XmlStream = outSqlXml.getBinaryStream();
```

Die Anweisung FETCH ruft die Daten in der Codierung UTF-8 in das SQLXML-Objekt ab. SQLXML.getBinaryStream speichert die Daten in einem Binärdatenstrom.

Beispiel: Sie rufen ein Dokument aus einer XML-Spalte in einen Hostausdruck vom Typ com.ibm.db2.jcc.DB2Xml ab, benötigen die Daten jedoch in einer Bytefolge mit einer XML-Deklaration, die eine interne Codierungsangabe für UTF-8 enthält.

```
#sql iterator  DB2XmlIter (int, com.ibm.db2.jcc.DB2Xml);
DB2XmlIter db2xmliter = null;
com.ibm.db2.jcc.DB2Xml outDB2Xml = null;
#sql [ctx] db2xmliter = {SELECT c1, CADOC from CUSTACC};
#sql {FETCH :db2xmliter INTO :row, :outDB2Xml};
byte[] byteArray = outDB2XML.getDB2XmlBytes("UTF-8");
```

Die Anweisung FETCH ruft die Daten in der Codierung UTF-8 in das DB2Xml-Objekt ab. Die Methode getDB2XmlBytes mit dem Argument UTF-8 fügt eine XML-Deklaration mit UTF-8-Codierungsangabe hinzu und speichert die Daten in einer Bytefeldgruppe.

PHP

PHP-Anwendungsentwicklung für IBM Datenserver

PHP: Hypertext Preprocessor (PHP) ist eine interpretierte Programmiersprache, die bei der Entwicklung von Webanwendungen weit verbreitet ist. PHP ist inzwischen eine gängige Sprache für die Webentwicklung, weil sie leicht zu erlernen ist, weil ihr Schwerpunkt auf praktischen Lösungen liegt und sie Unterstützung für die in Webanwendungen am häufigsten benötigten Funktionen bietet.

PHP ist eine modulare Sprache, mit der Sie den verfügbaren Funktionsumfang mithilfe von Erweiterungen anpassen können. Mit diesen Erweiterungen lassen sich Aufgaben vereinfachen, beispielsweise das Lesen, Schreiben und Bearbeiten von XML, das Erstellen von SOAP-Clients und -Servern oder das Verschlüsseln der Kommunikation zwischen dem Server und dem Browser. Die gängigsten Erweiterungen für PHP jedoch bieten Lese- und Schreibzugriff auf Datenbanken, sodass Sie ohne großen Aufwand eine dynamische, datenbankgesteuerte Website erstellen können.

IBM stellt die aufgelisteten PHP-Erweiterungen für den Zugriff auf IBM Datenserverdatenbanken zur Verfügung:

ibm_db2
: Eine prozedurbasierte Anwendungsprogrammierschnittstelle (API), die neben den üblichen Datenbankoperationen wie Erstellen, Lesen, Aktualisieren und Schreiben einen umfangreichen Zugriff auf die Datenbankmetadaten bereitstellt. Sie können ibm_db2 mit PHP 4 oder PHP 5 kompilieren. Diese Erweiterung wird von IBM geschrieben, gepflegt und unterstützt.

pdo_ibm
: Ein Treiber für die Erweiterung PHP Data Objects (PDO), die über die in PHP 5.1 eingeführte, objektorientierte Standarddatenbankschnittstelle Zugriff auf IBM Datenserverdatenbanken bietet.

Diese Erweiterungen sind als Bestandteil von IBM Data Server Driver Package (DS Driver) bei Version 1.7.0 im Lieferumfang enthalten. Diese Version und nachfolgende Versionen werden unterstützt, um eine Verbindung zu IBM DB2 Version 9.7 for Linux, UNIX and Windows zu ermöglichen. Sie können die Version der Erweiterung von ibm_db2 überprüfen, indem Sie den Befehl **php --re ibm_db2** ausgeben.

Die aktuellsten Versionen von ibm_db2 und pdo_ibm stehen in der PECL-Bibliothek (PECL - PHP Extension Community Library) unter http://pecl.php.net/ zur Verfügung.

PHP-Anwendungen können auf die aufgelisteten Datenbanken des IBM-Datenservers zugreifen:

- IBM DB2 Version 9.1 for Linux, UNIX and Windows, Fixpack 2 und höher
- IBM DB2 Universal Database (DB2 UDB) Version 8 for Linux, UNIX and Windows, Fixpak 15 und höher
- Ferne Verbindungen zu IBM DB2 for IBM i V5R3
- Ferne Verbindungen zu IBM DB2 for IBM i ab Version 5.4
- Ferne Verbindungen zu IBM DB2 for z/OS ab Version 8

Eine dritte Erweiterung, Unified ODBC, hat bisher Zugriff auf DB2-Datenbanksysteme geboten. Für neue Anwendungen können Sie entweder ibm_db2 oder pdo_ibm verwenden, da beide eine gute Leistung und hohe Stabilität gegenüber Uni-

fied ODBC bieten. Mit der API der Erweiterung ibm_db2 wird die Portierung einer Anwendung, die für Unified ODBC geschrieben wurde, beinahe so einfach wie die globale Änderung des Funktionsnamens `odbc_` in `db2_` im Quellcode Ihrer Anwendung.

Abrufen von XML-Daten mithilfe von PHP-Anwendungen

Die API ibm_db2 ist eine PHP-Erweiterung für den Zugriff auf IBM-Datenserverdatenbanken. Die API ibm_db2 unterstützt das Herstellen von Verbindungen zu DB2-Datenbanken und das Zurückgeben von XML-Daten aus XML-Spalten.

Darüber hinaus unterstützt die API ibm_db2 die Verwendung von DB2-Funktionen wie XMLTABLE sowie das Ausführen von XQuery-Ausdrücken in SQL-Anweisungen. Verwenden Sie in den SQL-Anweisungen die Funktion XMLQUERY zum Aufrufen von XQuery-Ausdrücken.

Weitere Informationen zum Entwickeln von PHP-Anwendungen mit der API ibm_db2 finden Sie unter "Application development in PHP with ibm_db2" in *Developing Perl, PHP, Python, and Ruby on Rails Applications*.

PHP-Downloads und zugehörige Ressourcen

Für die Entwicklung von PHP-Anwendungen für IBM Datenserver stehen zahlreiche Ressourcen zur Verfügung.

Tabelle 32. PHP-Downloads und zugehörige Ressourcen

Downloads	
Vollständiger PHP-Quellencode [1]	http://www.php.net/downloads.php
ibm_db2 und pdo_ibm aus der PHP Extension Community Library (PECL)	http://pecl.php.net/
IBM Data Server Driver Package (DS Driver)	http://www.ibm.com/software/data/support/data-server-clients/index.html
Zend Server	http://www.zend.com/en/products/server/downloads
PHP-Handbuch	http://www.php.net/docs.php
API-Dokumentation zu ibm_db2	http://www.php.net/ibm_db2
API-Dokumentation zu PDO	http://php.net/manual/en/book.pdo.php
PHP-Website	http://www.php.net/

1. Umfasst Windows-Binärprogramme. Die meisten Linux-Varianten enthalten bereits vorkompiliertes PHP.

Perl

pureXML und Perl

Der Treiber DBD::DB2 unterstützt DB2 pureXML. Die Unterstützung von pureXML ermöglicht über den Treiber DBD::DB2 einen direkteren Zugriff auf Ihre Daten und eine transparentere Kommunikation zwischen Ihrer Anwendung und der Datenbank, was zur Verringerung der Anwendungslogik beiträgt.

Dank der Unterstützung von pureXML können Sie XML-Dokumente direkt in Ihre DB2-Datenbank einfügen. Ihre Anwendung braucht XML-Dokumente nicht mehr syntaktisch zu analysieren, weil der pureXML-Parser beim Einfügen von XML-Da-

ten in die Datenbank automatisch ausgeführt wird. Die Verlagerung der Syntaxanalyse von Dokumenten nach außerhalb Ihrer Anwendung verbessert die Anwendungsleistung und verringert den Verwaltungsaufwand. Auch der Abruf von gespeicherten XML-Daten mit dem Treiber DBD::DB2 ist einfach: Sie können über ein großes Binärobjekt (BLOB) oder über einen Satz auf die Daten zugreifen.

Informationen zu DB2 Perl Database Interface (Perl DBI) und zur Vorgehensweise beim Herunterladen des neuesten DBD::DB2-Treibers finden Sie unter http://www.ibm.com/software/data/db2/perl.

Beispiel

Das folgende Beispiel ist ein Perl-Programm, das pureXML verwendet:

```
#!/usr/bin/perl
use DBI;
use strict ;

# Use DBD:DB2 module:
#   to create a simple DB2 table with an XML column
#   Add one row of data
#   retreive the XML data as a record or a LOB (based on $datatype).

# NOTE: the DB2 SAMPLE database must already exist.

my $database='dbi:DB2:sample';
my $user='';
my $password='';

my $datatype = "record" ;
#  $datatype = "LOB" ;

my $dbh = DBI->connect($database, $user, $password)
  or die "Can't connect to $database: $DBI::errstr";

# For LOB datatype, LongReadLen = 0 -- no data is retrieved on initial fetch
$dbh->{LongReadLen} = 0 if $datatype eq "LOB" ;

# SQL CREATE TABLE to create test table
my $stmt = "CREATE TABLE xmlTest (id INTEGER, data XML)";
my $sth = $dbh->prepare($stmt);
$sth->execute();

#insert one row of data into table
insertData() ;

# SQL SELECT statement returns home phone element from XML data
$stmt = qq(
 SELECT XMLQUERY ('
 \$d/*:customerinfo/*:phone[\@type = "home"] '
 passing data as "d")
 FROM xmlTest
) ;


# prepare and execute SELECT statement
$sth = $dbh->prepare($stmt);
$sth->execute();

# Print data returned from select statement
if($datatype eq "LOB") {
    printLOB() ;
}
else {
 printRecord() ;
```

```
}

# Drop table
$stmt = "DROP TABLE xmlTest" ;
$sth = $dbh->prepare($stmt);
$sth->execute();

warn $DBI::errstr if $DBI::err;

$sth->finish;
$dbh->disconnect;


##############


sub printRecord {
 print "output data as as record\n" ;

 while( my @row = $sth->fetchrow )
 {
  print $row[0] . "\n";
 }

 warn $DBI::errstr if $DBI::err;
}


sub printLOB {
 print "output as Blob data\n" ;

 my $offset = 0;
 my $buff="";
 $sth->fetch();
 while( $buff = $sth->blob_read(1,$offset,1000000)) {
  print $buff;
  $offset+=length($buff);
  $buff="";
 }
 warn $DBI::errstr if $DBI::err;
}


sub insertData {

 # insert a row of data
 my $xmlInfo = qq(\'
 <customerinfo xmlns="http://posample.org" Cid="1011">
   <name>Bill Jones</name>
   <addr country="Canada">
     <street>5 Redwood</street>
    <city>Toronto</city>
     <prov-state>Ontario</prov-state>
     <pcode-zip>M6W 1E9</pcode-zip>
   </addr>
   <phone type="work">416-555-9911</phone>
   <phone type="home">416-555-1212</phone>
 </customerinfo>
 \') ;

 my $catID = 1011 ;

 # SQL statement to insert data.
 my $Sql = qq(
  INSERT INTO xmlTest (id, data)
      VALUES($catID, $xmlInfo )
 );
```

```
 $sth = $dbh->prepare( $Sql )
   or die "Can't prepare statement: $DBI::errstr";

 my $rc = $sth->execute
   or die "Can't execute statement: $DBI::errstr";

 # check for problems
 warn $DBI::errstr if $DBI::err;
}
```

Datenbankverbindungen in Perl

Der Treiber DBD::DB2 bietet Unterstützung für von der API DBI definierte Verbindungsfunktionen für Standarddatenbanken.

Um Perl für das Laden des DBI-Moduls zu aktivieren, müssen Sie die `use DBI;`-Zeile in Ihre Anwendung einfügen.

Das DBI-Modul lädt automatisch den Treiber DBD::DB2, wenn Sie mithilfe der Anweisung **`DBI->connect`** eine Datenbankkennung mit der nachfolgenden Syntax erstellen:

```
my $dbhandle = DBI->connect('dbi:DB2:dsn', $userID, $password);
```

Dabei gilt:

$dbhandle
: steht für die Datenbankkennung, die von der Anweisung 'connect' zurückgegeben wird

dsn
: steht bei lokalen Verbindungen für einen DB2-Aliasnamen, der in Ihrem DB2-Datenbankverzeichnis katalogisiert ist

  bei fernen Verbindungen steht 'dsn' für eine vollständige Verbindungszeichenfolge mit Hostname, Portnummer, Protokoll, Benutzer-ID und Kennwort zum Herstellen einer Verbindung zum fernen Host

$userID
: steht für die Benutzer-ID, die zum Herstellen der Verbindung zur Datenbank verwendet wird

$password
: steht für das Kennwort zu der Benutzer-ID, die zum Herstellen der Verbindung zur Datenbank verwendet wird

Weitere Informationen zur API DBI finden Sie unter der Adresse http://search.cpan.org/~timb/DBI/DBI.pmhttp://search.cpan.org/~timb/DBI/DBI.pm.

Beispiel

Beispiel 1: Herstellen einer Verbindung zu einer Datenbank auf dem lokalen Host (Client und Server befinden sich auf derselben Workstation)

```
use DBI;

$DATABASE = 'dbname';
$USERID = 'username';
$PASSWORD = 'password';
```

```
my $dbh = DBI->connect("dbi:DB2:$DATABASE", $USERID, $PASSWORD, {PrintError => 0})
or die "Couldn't connect to database: " . DBI->errstr;

$dbh->disconnect;
```

Beispiel 2: Herstellen einer Verbindung zu einer Datenbank auf dem fernen Host (Client und Server befinden sich auf unterschiedlichen Workstations)

```
use DBI;

$DSN="DATABASE=sample; HOSTNAME=host; PORT=60000; PROTOCOL=TCPIP; UID=username;
PWD=password";

my $dbh = DBI->connect("dbi:DB2:$DSN", $USERID, $PASSWORD, {PrintError => 0})
or die "Couldn't connect to database: " . DBI->errstr;

$dbh->disconnect;
```

Einschränkungen von Perl

Einige Einschränkungen gelten für die Unterstützung, die für die Anwendungsentwicklung in Perl zur Verfügung steht.

Das DBI-Modul von Perl unterstützt nur dynamisches SQL. Wenn Sie eine Anweisung mehrmals ausführen müssen, können Sie die Leistung Ihrer Perl-Anwendungen verbessern, indem Sie zur Vorbereitung der Anweisung einen **prepare**-Aufruf absetzen.

Perl unterstützt keinen Datenbankzugriff mit Multithread.

Aktuelle Informationen zu den Einschränkungen der Version des DBD::DB2-Treibers, den Sie auf Ihrer Workstation installieren, finden Sie im DBD::DB2-Treiberpaket in der Datei CAVEATS.

Routinen

SQL-Prozeduren

XML- und XQuery-Unterstützung in SQL-Prozeduren

SQL-Prozeduren unterstützen Parameter und Variablen des Datentyps XML. Sie können in SQL-Anweisungen in gleicher Weise wie Variablen eines anderen Datentyps verwendet werden. Außerdem können Variablen des Datentyps XML in XMLEXISTS-, XMLQUERY- und XMLTABLE-Ausdrücken als Parameter an XQuery-Ausdrücke übergeben werden.

Das folgende Beispiel zeigt die Deklaration, Verwendung und Zuweisung von XML-Parametern und Variablen in einer SQL-Prozedur:

```
CREATE TABLE T1(C1 XML) %

CREATE PROCEDURE proc1(IN parm1 XML, IN parm2 VARCHAR(32000))
LANGUAGE SQL
BEGIN
   DECLARE var1 XML;

   /* check if the value of XML parameter parm1
      contains an item with a value less than 200 */
   IF(XMLEXISTS('$x/ITEM[value < 200]' passing by ref parm1 as "x"))THEN

     /* if it does, insert the value of parm1 into table T1 */
     INSERT INTO T1 VALUES(parm1);
```

```
    END IF;

    /* parse parameter parm2's value and assign it to a variable */
    SET var1 = XMLPARSE(document parm2 preserve whitespace);

    /* insert variable var1 into table T1
    INSERT INTO T1 VALUES(var1);

  END %
```

In diesem Beispiel gibt es eine Tabelle `T1` mit einer XML-Spalte. Die SQL-Prozedur akzeptiert zwei Parameter des Datentyps XML namens `parm1` und `parm2`. In der SQL-Prozedur wird eine XML-Variable namens `var1` deklariert.

Die Logik der SQL-Prozedur prüft, ob der Wert des XML-Parameters `parm1` ein Element enthält, dessen Wert kleiner als 200 ist. In diesem Fall wird der XML-Wert direkt in die Spalte `C1` der Tabelle `T1` eingefügt.

Anschließend wird der Wert des Parameters `parm2` mit der Funktion XMLPARSE syntaktisch analysiert und der XML-Variablen `var1` zugewiesen. Der XML-Variablenwert wird dann ebenfalls in die Spalte `C1` der Tabelle `T1` eingefügt.

Die Möglichkeit, Steuerungsflusslogik im Umfeld von XQuery-Operationen zu implementieren, vereinfacht die Entwicklung komplexer Algorithmen zum Abfragen von und Zugreifen auf XML-Daten, die in einer Datenbank gespeichert sind.

Cursor für XQuery-Ausdrücke in SQL-Prozeduren

SQL-Prozeduren unterstützen die Definition von Cursorn für XQuery-Ausdrücke. Mit einem Cursor für einen XQuery-Ausdruck können Sie über die Elemente der von dem Ausdruck zurückgegebenen XQuery-Sequenz iterieren.

Im Gegensatz zu Cursorn für SQL-Anweisungen, die statisch oder dynamisch definiert werden können, lassen sich Cursor für XQuery-Ausdrücke nicht dynamisch definieren. Beim dynamischen Deklarieren eines Cursors muss eine Variable vom Typ CHAR oder VARCHAR so deklariert werden, dass sie den XQuery-Ausdruck enthält, mit dem die Ergebnismenge des Cursors definiert wird. Der XQuery-Ausdruck muss vorbereitet werden, damit der Cursor geöffnet und die Ergebnismenge aufgelöst werden kann.

Nachstehend finden Sie ein Beispiel für eine SQL-Prozedur, die einen Cursor dynamisch für einen XQuery-Ausdruck deklariert, den Cursor öffnet und XML-Daten abruft:

```
CREATE PROCEDURE xmlProc(IN inCust XML, OUT resXML XML)
SPECIFIC xmlProc
LANGUAGE SQL
BEGIN
    DECLARE SQLSTATE CHAR(5);
    DECLARE stmt_text VARCHAR (1024);
    DECLARE customer XML;
    DECLARE cityXml XML;
    DECLARE city VARCHAR (100);
    DECLARE stmt STATEMENT;
    DECLARE cur1 CURSOR FOR stmt;

   -- Get the city of the input customer
   SET cityXml = XMLQUERY('$cust/customerinfo//city' passing inCust as "cust");
   SET city = XMLCAST(cityXml as VARCHAR(100));

  -- Iterate over all the customers from the city using an XQUERY cursor
```

```
  -- and collect the customer name values into the output XML value

   SET stmt_text = 'XQUERY for $cust
                    in db2-fn:xmlcolumn("CUSTOMER.INFO")
                       /*:customerinfo/*:addr[*:city= "' || city ||'"]
                    return <Customer>{$cust/../@Cid}{$cust/../*:name}</Customer>';

  -- Use the name of the city for the input customer data as a prefix
  SET resXML = cityXml;

  PREPARE stmt FROM stmt_text;
  OPEN cur1;

  FETCH cur1 INTO customer;
  WHILE (SQLSTATE = '00000') DO
    SET resXML = XMLCONCAT(resXML, customer);
    FETCH cur1 INTO customer;
  END WHILE;

  set resXML = XMLQUERY('<result> {$res} </result>'
                        passing resXML as "res");

END
```

Diese SQL-Prozedur erfasst die IDs und Namen der in einer Tabelle namens CUSTOMER definierten Kunden, die in derselben Stadt wie der Kunde wohnen, für den XML-Daten als Eingabeparameter bereitgestellt wurden.

Diese SQL-Prozedur kann durch die Ausführung der Anweisung CALL wie folgt aufgerufen werden:

```
  CALL xmlProc(xmlparse(document '<customerinfo Cid="5002">
                                   <name>Jim Noodle</name>
                                    <addr country="Canada">
                                      <street>25 EastCreek</street>
                                       <city>Markham</city>
                                       <prov-state>Ontario</prov-state>
                                       <pcode-zip>N9C-3T6</pcode-zip>
                                    </addr>
                                    <phone type="work">905-566-7258</phone>
                                  </customerinfo>' PRESERVE WHITESPACE),?)
```

Wenn diese SQL-Prozedur erstellt und für die Datenbank SAMPLE ausgeführt wird, gibt sie XML-Daten für zwei Kunden zurück.

Da bei XML-Werten keine Parametermarken unterstützt werden, können Sie diese Einschränkung umgehen, indem Sie aus Fragmenten einer verknüpften Anweisung, die den Wert einer oder mehrerer lokaler Variablen enthalten, eine dynamische SQL-Anweisung erstellen.

Beispiel:

```
DECLARE person_name VARCHAR(128);

SET person_name = "Joe";
SET stmt_text = 'XQUERY for $fname in db2-fn:sqlquery
   ("SELECT doc
      FROM T1
      WHERE DOCID=1")//fullname where $fname/first = ''' person_name || ''';
```

Dieses Beispiel gibt eine Ergebnismenge in einer Variablenzuweisung für eine XQuery-Anweisung zurück, die einen SQL-Fullselect enthält. Die Ergebnismenge enthält die vollständigen Namen aller Personen mit dem Vornamen Joe. Dabei wählt der SQL-Abschnitt in der Tabelle T1 aus der Spalte doc die XML-Dokumente

mit der ID 1 aus. Anschließend wählt der XQuery-Abschnitt in den XML-Dokumenten, in denen "first" (Vorname) den Wert `Joe` hat, den Wert für "fullname" (vollständiger Name) aus.

Auswirkungen von Commit- und Rollbackoperationen auf XML-Parameter- und -Variablenwerte in SQL-Prozeduren

Commit- und Rollbackoperationen in SQL-Prozeduren beeinflussen die Werte von Parametern und Variablen des Datentyps XML. Wenn während der Ausführung von SQL-Prozeduren Commit- oder Rollbackoperationen durchgeführt werden, sind die Werte, die den XML-Parametern und XML-Variablen zugewiesen wurden, nicht mehr verfügbar.

Wenn Sie versuchen, nach einer Commit- oder Rollbackoperation auf eine SQL-Variable oder einen SQL-Parameter des Datentyps XML zu verweisen, tritt ein Fehler auf (SQL1354N, 560CE).

Um nach einer Commit- oder Rollbackoperation erfolgreich auf XML-Parameter und -Variablen zu verweisen, müssen Sie diesen zunächst neue Werte zuweisen.

Berücksichtigen Sie die Verfügbarkeit der Werte von XML-Parametern und -Variablen, wenn Sie ROLLBACK- und COMMIT-Anweisungen zu SQL-Prozeduren hinzufügen.

SQL-Funktionen

Parameter und Variablen vom Datentyp XML in SQL-Funktionen

DB2-Datenbanksysteme unterstützen den Datentyp XML für integrierte SQL-Funktionen, die Sie mithilfe der Anweisung CREATE FUNCTION (SQL Skalar, Tabelle oder Zeile) oder mithilfe der Anweisung CREATE FUNCTION (abgeleitet oder Schablone) erstellen.

In einer integrierten benutzerdefinierten Funktion, die mit der Anweisung CREATE FUNCTION (SQL-Skalar, Tabelle oder Zeile) erstellt wurde, können Sie XML-Variablen in SQL-Anweisungen in gleicher Weise wie Variablen eines anderen Datentyps verwenden. Beispielsweise können Sie in einer benutzerdefinierten Funktion Variablen mit XML-Datentyp als Parameter an XQuery-Ausdrücke in einem Vergleichselement XMLEXISTS oder einer Funktion wie XMLQUERY oder XMLTABLE übergeben.

In einer benutzerdefinierten Funktion, die mit der Anweisung CREATE FUNCTION (abgeleitet oder nach Vorlage) erstellt wurde und deren Quellenfunktion eine benutzerdefinierte SQL-Skalarfunktion ist, können Sie den XML-Datentyp als Eingabe-, Ausgabe- oder Ein-/Ausgabeparameter verwenden.

XML-Werte werden durch Verweise in einer benutzerdefinierten Funktion zugeordnet.

Parameter und Variablen vom Datentyp XML werden in kompilierten SQL-Funktionen nicht unterstützt.

Beispiel

Bei der folgenden Beispielfunktion handelt es sich um eine integrierte SQL-Skalarfunktion, bei der der XML-Datentyp als Eingabeparameter und Variable verwendet

wird. Die Funktion extrahiert mithilfe eines XQuery-Ausdrucks das Element 'Telefonnummer' aus einem XML-Dokument und gibt das Element 'Telefonnummer' zurück:

```
CREATE FUNCTION phone_number ( dept_doc XML )
RETURNS XML
LANGUAGE SQL
NO EXTERNAL ACTION
BEGIN ATOMIC
DECLARE tmp_xml XML;
 IF (XMLEXISTS('$test/department/phone' passing by ref dept_doc as "test"))
 THEN
  SET tmp_xml = XMLQUERY('document
       {<phone_list>{$doc/department/phone}</phone_list>}'
    PASSING dept_doc as "doc");
 ELSE
 SET tmp_xml = XMLPARSE(document '<phone_list><phone>N/A</phone></phone_list>');
END IF;
RETURN tmp_xml;
END
```

Bei der folgenden Anweisung SELECT wird die Funktion PHONE_NUMBER zum Abrufen der Telefonnummer aus einem XML-Dokument in einer Tabelle mit Mitarbeiterinformationen verwendet.

```
SELECT PHONE_NUMBER(info) FROM employees WHERE empid = 12356
```

Bei der Anweisung SELECT wird vorausgesetzt, dass die Tabelle ähnlich einer mit der folgenden Anweisung CREATE TABLE erstellten Tabelle ist und Daten enthält, die den mit der folgenden INSERT-Anweisung eingefügten Informationen ähnlich sind:

```
CREATE TABLE employees (empid BIGINT, info XML )
```

```
INSERT INTO EMPLOYEES VALUES ( 12356, '
    <department id="marketing">
        <empid>12356</empid>
        <phone>555-123-4567</phone>
     </department> ')
```

Unter Verwendung der vorherigen Tabelle und Informationen gibt die Anweisung SELECT die folgende Information zur Telefonnummer zurück:

```
<phone_list><phone>555-123-4567</phone></phone_list>
```

Integrierte SQL-Funktionen und kompilierte SQL-Funktionen

Für SQL-Funktionen stehen zwei Implementierungstypen zur Verfügung: integrierte SQL-Funktionen und kompilierte SQL-Funktionen.

Integrierte SQL-Funktionen

Integrierte SQL-Funktionen sind SQL-Funktionen, die mithilfe der Anweisung CREATE FUNCTION erstellt werden, deren Hauptteil entweder aus einer RETURN-Anweisung oder einer integrierten Compound-Anweisung besteht. Integrierte Compound-Anweisungen werden mit den Schlüsselwörtern BEGIN ATOMIC und END definiert.

Integrierte SQL-Funktionen können SQL-Anweisungen und Inline SQL PL-Anweisungen, eine Untergruppe von SQL PL-Anweisungen, enthalten.

Kompilierte SQL-Funktionen

Kompilierte SQL-Funktionen sind SQL-Funktionen, die mithilfe der Anweisung CREATE FUNCTION erstellt werden, deren Hauptteil entweder aus

einer RETURN-Anweisung oder einer kompilierten Compound-Anweisung besteht. Kompilierte Compound-Anweisungen werden mit den Schlüsselwörtern BEGIN und END definiert.

Wenn die ATOMIC-Klausel weggelassen wird, werden SQL-Funktionen kompiliert, die als solche mehr SQL PL-Komponenten einschließen bzw. auf mehr SQL PL-Komponenten verweisen können als integrierte SQL-Funktionen. Kompilierte SQL-Funktionen können die folgenden Komponenten enthalten, die in integrierten SQL-Funktionen nicht unterstützt werden:

- SQL PL-Anweisungen, einschließlich:
  - CASE-Anweisung
  - REPEAT-Anweisung
- Cursorverarbeitung
- Dynamisches SQL
- Bedingungshandler

Externe Routinen

Unterstützung des XML-Datentyps in externen Routinen

Externe Prozeduren und Funktionen, die in den folgenden Programmiersprachen geschrieben sind, unterstützen Parameter und Variablen des Datentyps XML:

- C
- C++
- COBOL
- Java
- .NET-CLR-Sprachen

Externe OLE- und OLEDB-Routinen bieten keine Unterstützung für Parameter des Datentyps XML.

XML-Datentypwerte werden im Code externer Routinen genauso dargestellt wie CLOB-Datentypen.

Wenn Sie in externen Routinen Parameter des Datentyps XML deklarieren, müssen die Anweisungen CREATE PROCEDURE und CREATE FUNCTION, mit deren Hilfe die Routinen in der Datenbank erstellt werden, angeben, dass der XML-Datentyp als CLOB-Datentyp gespeichert werden soll. Die Größe des CLOB-Werts sollte im Bereich der Größe des XML-Dokuments liegen, das von dem XML-Parameter dargestellt wird.

Die folgende Anweisung CREATE PROCEDURE zeigt eine Anweisung CREATE PROCEDURE für eine externe Prozedur, die mit einem XML-Parameter namens `parm1` in der Programmiersprache C implementiert wurde:

```
CREATE PROCEDURE myproc(IN parm1 XML AS CLOB(2M), IN parm2 VARCHAR(32000))
LANGUAGE C
FENCED
PARAMETER STYLE SQL
EXTERNAL NAME 'mylib!myproc';
```

Ähnliche Überlegungen gelten beim Erstellen externer UDFs, wie im folgenden Beispiel gezeigt:

```
CREATE FUNCTION myfunc (IN parm1 XML AS CLOB(2M))
RETURNS SMALLINT
LANGUAGE C
PARAMETER STYLE SQL
DETERMINISTIC
NOT FENCED
NULL CALL
NO SQL
NO EXTERNAL ACTION
EXTERNAL NAME 'mylib1!myfunc'
```

XML-Daten werden umgesetzt, wenn sie als Parameter IN, OUT oder INOUT an gespeicherte Prozeduren übergeben werden. Wenn Sie mit gespeicherten Java-Prozeduren arbeiten, müssen eventuell die Zwischenspeichergröße (Konfigurationsparameter **java_heap_sz**) je nach Anzahl und Größe der XML-Argumente sowie die Anzahl externer gespeicherter Prozeduren, die gleichzeitig ausgeführt werden, erhöht werden.

Im Code externer Routinen werden die Werte von XML-Parametern und Variablen in gleicher Weise geöffnet, festgelegt und geändert wie in Datenbankanwendungen.

Angeben eines Treibers für Java-Routinen

Bei der Entwicklung und dem Aufruf von Java muss ein JDBC- oder SQLJ-Treiber angegeben werden.

Java-Routinen verwenden die IBM Data Server Driver for JDBC and SQLJ Version 4.0.

IBM Data Server Driver for JDBC and SQLJ Version 4.0 `db2jcc4.jar` enthält eine Reihe von Leistungsmerkmalen von JDBC Version 4.0. Der Treiber wird von DB2 ab Version 9.5 unterstützt.

Standardmäßig verwenden DB2-Datenbanksysteme den IBM Data Server-Treiber für JDBC und SQLJ. Dieser Treiber wird vorausgesetzt, wenn Java-Routinen folgenden Inhalt aufweisen:

- Parameter des Datentyps XML
- Variablen des Datentyps XML
- Verweise auf XML-Daten
- Verweise auf XML-Funktionen
- Andere Funktionen mit nativem XML

Beispiel: XML- und XQuery-Unterstützung in Java-Prozeduren (JDBC)

Sobald Sie die grundlegenden Informationen zu Java-Prozeduren, zur Programmierung in Java über die JDBC-Anwendungsprogrammierschnittstelle (API) und zu XQuery verstanden haben, können Sie mit der Erstellung und Verwendung von Java-Prozeduren beginnen, mit denen XML-Daten abgefragt werden.

Im nachstehenden Beispiel einer Java-Prozedur wird Folgendes gezeigt:

- Anweisung CREATE PROCEDURE einer JAVA-Prozedur in Parameterdarstellung
- Quellcode einer JAVA-Prozedur in Parameterdarstellung
- Ein- und Ausgabeparameter des Datentyps XML
- Verwendung eines XML-Eingabeparameters in einer Abfrage

- Zuweisung des Ergebnisses einer XQuery-Abfrage, eines XML-Werts, zu einem Ausgabeparameter
- Zuweisung des Ergebnisses einer SQL-Anweisung, eines XML-Werts, zu einem Ausgabeparameter

Voraussetzungen

Bevor Sie mit diesem Beispiel einer Java-Prozedur arbeiten, empfiehlt es sich, die Abschnitte zu den folgenden Konzepten zu lesen:

- Java-Routinen
- Routinen
- Erstellen von Java-Routinencode

Für die folgenden Beispiele wird eine Tabelle namens `xmlDataTable` definiert und verwendet, die folgende Daten enthält:

```
CREATE TABLE xmlDataTable
(
   num INTEGER,
   xdata XML
)@

INSERT INTO xmlDataTable VALUES
   (1, XMLPARSE(DOCUMENT '<doc>
                              <type>car</type>
                              <make>Pontiac</make>
                              <model>Sunfire</model>
                              </doc>' PRESERVE WHITESPACE)),
   (2, XMLPARSE(DOCUMENT '<doc>
                              <type>car</type>
                              <make>Mazda</make>
                              <model>Miata</model>
                              </doc>' PRESERVE WHITESPACE)),
   (3, XMLPARSE(DOCUMENT '<doc>
                              <type>person</type>
                              <name>Mary</name>
                              <town>Vancouver</town>
                              <street>Waterside</street>
                              </doc>' PRESERVE WHITESPACE)),
   (4, XMLPARSE(DOCUMENT '<doc>
                              <type>person</type>
                              <name>Mark</name>
                              <town>Edmonton</town>
                              <street>Oak</street>
                              </doc>' PRESERVE WHITESPACE)),
   (5, XMLPARSE(DOCUMENT '<doc>
                              <type>animal</type>
                              <name>dog</name>
                              </doc>' PRESERVE WHITESPACE)),
   (6, NULL),
   (7, XMLPARSE(DOCUMENT '<doc>
                              <type>car</type>
                              <make>Ford</make>
                              <model>Taurus</model>
                              </doc>' PRESERVE WHITESPACE)),
   (8, XMLPARSE(DOCUMENT '<doc>
                              <type>person</type>
                              <name>Kim</name>
                              <town>Toronto</town>
                              <street>Elm</street>
                              </doc>' PRESERVE WHITESPACE)),
   (9, XMLPARSE(DOCUMENT '<doc>
                              <type>person</type>
                              <name>Bob</name>
                              <town>Toronto</town>
                              <street>Oak</street>
```

```
                                    </doc>' PRESERVE WHITESPACE)),
        (10, XMLPARSE(DOCUMENT '<doc>
                                     <type>animal</type>
                                     <name>bird</name>
                                     </doc>' PRESERVE WHITESPACE))@
```

Prozedur

Verwenden Sie bei der Erstellung eigener Java-Prozeduren das folgende Beispiel als Referenz:

- „Datei mit externem Java-Code"
- „Beispiel 1: JAVA-Prozedur in Parameterdarstellung mit XML-Parametern"

Datei mit externem Java-Code

Das Beispiel zeigt die Implementierung einer Java-Prozedur. Das Beispiel besteht aus zwei Abschnitten: der Anweisung CREATE PROCEDURE und der externen Java-Codeimplementierung der Prozedur, aus der die zugeordnete Java-Klasse erstellt werden kann.

Die Java-Quellendatei, in der die Prozedurimplementierungen der folgenden Beispiele enthalten sind, heißt `stpclass.java` und befindet sich in einer JAR-Datei mit dem Namen `myJAR`. Die Datei hat folgendes Format:

```
using System;
import java.lang.*;
import java.io.*;
import java.sql.*;
import java.util.*;
import com.ibm.db2.jcc.DB2Xml;

 public class stpclass
 {          ...
    // Java procedure implementations
            ...
 }
```

Die importierten Java-Klassendateien sind am Anfang der Datei aufgeführt. Der Import von `com.ibm.db2.jcc.DB2Xml` ist erforderlich, wenn die Prozeduren in der Datei Parameter enthalten oder wenn XML-Variablen verwendet werden.

Es ist wichtig, die Namen der Klassendatei und der JAR-Datei zu notieren, die eine bestimmte Prozedurimplementierung enthält. Diese Namen sind wichtig, weil die Klausel EXTERNAL der Anweisung CREATE PROCEDURE diese Informationen bei jeder Prozedur angeben muss, damit DB2-Datenbanksysteme die Klasse während der Laufzeit finden kann.

Beispiel 1: JAVA-Prozedur in Parameterdarstellung mit XML-Parametern

Dieses Beispiel zeigt Folgendes:

- Anweisung CREATE PROCEDURE einer JAVA-Prozedur in Parameterdarstellung
- Java-Code einer JAVA-Prozedur in Parameterdarstellung mit XML-Parametern

Diese Prozedur nimmt den Eingabeparameter `inXML` an, fügt eine Zeile mit diesem Wert in eine Tabelle ein, fragt mit einer SQL-Anweisung und einem XQuery-Ausdruck XML-Daten ab und legt die beiden Ausgabeparameter `outXML1` und `outXML2` fest.

```
CREATE PROCEDURE  xmlProc1   ( IN inNUM  INTEGER,
                               IN inXML  XML as CLOB (1K),
                               OUT out1XML XML as CLOB (1K),
                               OUT out2XML XML as CLOB (1K)
                                  )
DYNAMIC RESULT SETS 0
DETERMINISTIC
LANGUAGE JAVA
PARAMETER STYLE JAVA
MODIFIES SQL DATA
FENCED
THREADSAFE
DYNAMIC RESULT SETS 0
PROGRAM TYPE SUB
NO DBINFO
EXTERNAL NAME 'myJar:stpclass.xmlProc1'@
//*************************************************************************
//  Stored Procedure: XMLPROC1
//
//  Purpose:   Inserts XML data into XML column; queries and returns XML data
//
//  Parameters:
//
//   IN:      inNum -- the sequence of XML data to be insert in xmldata table
//              inXML -- XML data to be inserted
//    OUT:     out1XML -- XML data to be returned
//              out2XML -- XML data to be returned
//
//*************************************************************************
public void xmlProc1(int inNum,
                     DB2Xml inXML ,
                     DB2Xml[] out1XML,
                     DB2Xml[] out2XML
                    )
throws Exception
{
       Connection con = DriverManager.getConnection("jdbc:default:connection");

       // Insert data including the XML parameter value into a table
       String query = "INSERT INTO xmlDataTable (num, inXML ) VALUES ( ?, ? )" ;
       String xmlString = inXML.getDB2String() ;

       stmt = con.prepareStatement(query);
       stmt.setInt(1, inNum);
       stmt.setString (2, xmlString );
       stmt.executeUpdate();
       stmt.close();

       // Query and retrieve a single XML value from a table using SQL
       query = "SELECT xdata from xmlDataTable WHERE num = ? " ;

       stmt = con.prepareStatement(query);
       stmt.setInt(1, inNum);
       ResultSet rs = stmt.executeQuery();

       if ( rs.next() )
       {  out1Xml[0] = (DB2Xml) rs.getObject(1);  }

       rs.close() ;
       stmt.close();

       // Query and retrieve a single XML value from a table using XQuery
       query = "XQUERY for $x in db2-fn:xmlcolumn(\"xmlDataTable.xdata\")/doc
                             where $x/make = \'Mazda\'
                             return <carInfo>{$x/make}{$x/model}</carInfo>";
```

```
        stmt = con.createStatement();

        rs = stmt.executeQuery( query );

        if ( rs.next() )
        {  out2Xml[0] = (DB2Xml) rs.getObject(1) ;  }

        rs.close();
        stmt.close();
        con.close();

        return ;
}
```

Beispiel: XML- und XQuery-Unterstützung in C#-.NET-CLR-Prozeduren

Sobald Sie die grundlegenden Informationen zu Prozeduren, .NET-CLR-Routinen, XQuery und XML verstanden haben, können Sie anfangen, CLR-Prozeduren mit XML-Funktionen zu erstellen und zu verwenden.

Das folgende Beispiel zeigt eine C#-.NET-CLR-Prozedur mit Parametern des Typs XML sowie die Aktualisierung und Abfrage von XML-Daten.

Voraussetzungen

Bevor Sie mit diesem Beispiel einer CLR-Prozedur arbeiten, empfiehlt es sich, die Abschnitte zu den folgenden Konzepten zu lesen:

- .NET-CLR-Routinen (CLR - Common Language Runtime)
- Erstellen von .NET CLR-Routinen über das DB2-Befehlsfenster
- Vorteile der Verwendung von Routinen

Das folgende Beispiel verwendet eine Tabelle namens `xmlDataTable`, die wie folgt definiert ist:

```
CREATE TABLE xmlDataTable
(
   num INTEGER,
   xdata XML
)

INSERT INTO xmlDataTable VALUES
   (1, XMLPARSE(DOCUMENT '<doc>
                              <type>car</type>
                              <make>Pontiac</make>
                              <model>Sunfire</model>
                              </doc>' PRESERVE WHITESPACE)),
   (2, XMLPARSE(DOCUMENT '<doc>
                              <type>car</type>
                              <make>Mazda</make>
                              <model>Miata</model>
                              </doc>' PRESERVE WHITESPACE)),
   (3, XMLPARSE(DOCUMENT '<doc>
                              <type>person</type>
                              <name>Mary</name>
                              <town>Vancouver</town>
                              <street>Waterside</street>
                              </doc>' PRESERVE WHITESPACE)),
   (4, XMLPARSE(DOCUMENT '<doc>
                              <type>person</type>
                              <name>Mark</name>
                              <town>Edmonton</town>
                              <street>Oak</street>
                              </doc>' PRESERVE WHITESPACE)),
   (5, XMLPARSE(DOCUMENT '<doc>
                              <type>animal</type>
```

```
                                <name>dog</name>
                                </doc>' PRESERVE WHITESPACE)),
        (6, NULL),
        (7, XMLPARSE(DOCUMENT '<doc>
                                <type>car</type>
                                <make>Ford</make>
                                <model>Taurus</model>
                                </doc>' PRESERVE WHITESPACE)),
        (8, XMLPARSE(DOCUMENT '<doc>
                                <type>person</type>
                                <name>Kim</name>
                                <town>Toronto</town>
                                <street>Elm</street>
                                </doc>' PRESERVE WHITESPACE)),
        (9, XMLPARSE(DOCUMENT '<doc>
                                <type>person</type>
                                <name>Bob</name>
                                <town>Toronto</town>
                                <street>Oak</street>
                                </doc>' PRESERVE WHITESPACE)),
        (10, XMLPARSE(DOCUMENT '<doc>
                                 <type>animal</type>
                                 <name>bird</name>
                                 </doc>' PRESERVE WHITESPACE))@
```

Prozedur

Verwenden Sie bei der Erstellung eigener C#-CLR-Prozeduren die folgenden Beispiele als Referenz:

- „Datei mit externem C#-Code"
- „Beispiel 1: C#-Prozedur GENERAL in Parameterdarstellung mit XML-Funktionen" auf Seite 303

Datei mit externem C#-Code

Das Beispiel besteht aus zwei Abschnitten: der Anweisung CREATE PROCEDURE und der externen C#-Codeimplementierung der Prozedur, aus der die zugehörige Baugruppe (Assembly) erstellt werden kann.

Die C#-Quellendatei, in der die Prozedurimplementierungen der folgenden Beispiele enthalten sind, heißt `gwenProc.cs` und hat folgendes Format:

```
using System;
using System.IO;
using System.Data;
using IBM.Data.DB2;
using IBM.Data.DB2Types;

namespace bizLogic
{
   class empOps
   {          ...
     // C# procedures
              ...
   }
}
```

Die Dateieinbindungen sind am Anfang der Datei aufgeführt. Die Einbindung von `IBM.Data.DB2` ist erforderlich, wenn die Prozeduren in der Datei SQL enthalten. Die Einbindung von `IBM.Data.DB2Types` ist erforderlich, wenn die Prozeduren in der Datei Parameter oder Variablen des Typs XML enthalten. In dieser Datei ist eine Namensbereichsdeklaration sowie eine Klasse namens `empOps` vorhanden, die die Prozeduren enthält. Die Verwendung der Namensbereiche ist optional. Wird ein

Namensbereich verwendet, muss er im Pfadnamen der Baugruppe angezeigt werden, der in der Klausel EXTERNAL der Anweisung CREATE PROCEDURE bereitgestellt wird.

Es ist wichtig, die Namen der Datei, des Namensbereichs und der Klasse zu notieren, die eine bestimmte Prozedurimplementierung enthält. Diese Namen sind wichtig, weil die Klausel EXTERNAL der Anweisung CREATE PROCEDURE diese Informationen bei jeder Prozedur angeben muss, damit DB2-Datenbanksysteme die Baugruppe und Klasse der CLR-Prozedur finden können.

Beispiel 1: C#-Prozedur GENERAL in Parameterdarstellung mit XML-Funktionen

Dieses Beispiel zeigt Folgendes:

- Anweisung CREATE PROCEDURE einer Prozedur GENERAL in Parameterdarstellung
- C#-Code einer Prozedur GENERAL in Parameterdarstellung mit XML-Parametern

Diese Prozedur nimmt zwei Parameter an, die ganzen Zahlen `inNum` und `inXML`. Diese Werte werden in die Tabelle `xmlDataTable` eingefügt. Danach wird mit XQuery ein XML-Wert abgerufen. Ein weiterer XML-Wert wird mit SQL abgerufen. Die abgerufenen XML-Werte werden den beiden Ausgabeparametern `outXML1` und `outXML2` zugewiesen. Es werden keine Ergebnismengen zurückgegeben.

```
CREATE PROCEDURE  xmlProc1   ( IN inNUM  INTEGER,
                               IN inXML  XML as CLOB (1K),
                               OUT inXML  XML as CLOB (1K),
                               OUT inXML  XML as CLOB (1K)
                             )
LANGUAGE CLR
PARAMETER STYLE GENERAL
DYNAMIC RESULT SETS 0
FENCED
THREADSAFE
DETERMINISTIC
NO DBINFO
MODIFIES SQL DATA
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenProc.dll:bizLogic.empOps!xmlProc1' ;
```

```
//*************************************************************************
//  Stored Procedure: xmlProc1
//
//  Purpose:   insert XML data into XML column
//
//  Parameters:
//
//   IN:    inNum -- the sequence of XML data to be insert in xmldata table
//          inXML -- XML data to be inserted
//   OUT:   outXML1 -- XML data returned - value retrieved using XQuery
//          outXML2 -- XML data returned - value retrieved using SQL
//*************************************************************************
    public static void xmlProc1 (     int      inNum, DB2Xml   inXML,
                                  out DB2Xml   outXML1, out DB2Xml   outXML2 )
    {
       // Create new command object from connection context
       DB2Parameter parm;
       DB2Command cmd;
       DB2DataReader reader = null;
       outXML1 = DB2Xml.Null;
       outXML2 = DB2Xml.Null;
```

```
    // Insert input XML parameter value into a table
    cmd = DB2Context.GetCommand();
    cmd.CommandText = "INSERT INTO "
                    + "xmlDataTable( num , xdata ) "
                    + "VALUES( ?, ? )";

    parm = cmd.Parameters.Add("@num", DB2Type.Integer );
    parm.Direction = ParameterDirection.Input;
    cmd.Parameters["@num"].Value = inNum;
    parm = cmd.Parameters.Add("@data", DB2Type.Xml);
    parm.Direction = ParameterDirection.Input;
    cmd.Parameters["@data"].Value = inXML ;
    cmd.ExecuteNonQuery();
    cmd.Close();

    // Retrieve XML value using XQuery
                  and assign value to an XML output parameter
    cmd = DB2Context.GetCommand();
    cmd.CommandText = "XQUERY for $x " +
                      "in db2-fn:xmlcolumn(\"xmlDataTable.xdata\")/doc "+
                      "where $x/make = \'Mazda\' " +
                      "return <carInfo>{$x/make}{$x/model}</carInfo>";
    reader = cmd.ExecuteReader();
    reader.CacheData= true;

    if (reader.Read())
    {  outXML1 = reader.GetDB2Xml(0);  }
    else
    {  outXML1 = DB2Xml.Null;  }

    reader.Close();
    cmd.Close();

    // Retrieve XML value using SQL
                  and assign value to an XML output parameter value
    cmd = DB2Context.GetCommand();
    cmd.CommandText = "SELECT xdata "
                      + "FROM xmlDataTable "
                      + "WHERE num = ?";

    parm = cmd.Parameters.Add("@num", DB2Type.Integer );
    parm.Direction = ParameterDirection.Input;
    cmd.Parameters["@num"].Value = inNum;
    reader = cmd.ExecuteReader();
    reader.CacheData= true;

    if (reader.Read())
    {  outXML2 = reader.GetDB2Xml(0);  }
    else
    {  outXML = DB2Xml.Null;  }

    reader.Close() ;
    cmd.Close();

    return;
}
```

Beispiel: XML- und XQuery-Unterstützung in C-Prozeduren

Sobald Sie die grundlegenden Informationen zu Prozeduren, C-Routinen, XQuery und XML verstanden haben, können Sie anfangen, C-Prozeduren mit XML-Funktionen zu erstellen und zu verwenden.

Das folgende Beispiel zeigt eine C-Prozedur mit Parametern des Typs XML sowie die Aktualisierung und Abfrage von XML-Daten.

Voraussetzungen

Bevor Sie mit diesem Beispiel einer C-Prozedur arbeiten, empfiehlt es sich, den Abschnitt zu dem folgenden Konzept zu lesen:

- Vorteile der Verwendung von Routinen

Das folgende Beispiel verwendet eine Tabelle namens `xmlDataTable`, die wie folgt definiert ist:

```
CREATE TABLE xmlDataTable
(
   num INTEGER,
   xdata XML
)

INSERT INTO xmlDataTable VALUES
   (1, XMLPARSE(DOCUMENT '<doc>
                              <type>car</type>
                              <make>Pontiac</make>
                              <model>Sunfire</model>
                              </doc>' PRESERVE WHITESPACE)),
   (2, XMLPARSE(DOCUMENT '<doc>
                              <type>car</type>
                              <make>Mazda</make>
                              <model>Miata</model>
                              </doc>' PRESERVE WHITESPACE)),
   (3, XMLPARSE(DOCUMENT '<doc>
                              <type>person</type>
                              <name>Mary</name>
                              <town>Vancouver</town>
                              <street>Waterside</street>
                              </doc>' PRESERVE WHITESPACE)),
   (4, XMLPARSE(DOCUMENT '<doc>
                              <type>person</type>
                              <name>Mark</name>
                              <town>Edmonton</town>
                              <street>Oak</street>
                              </doc>' PRESERVE WHITESPACE)),
   (5, XMLPARSE(DOCUMENT '<doc>
                              <type>animal</type>
                              <name>dog</name>
                              </doc>' PRESERVE WHITESPACE)),
   (6, NULL),
   (7, XMLPARSE(DOCUMENT '<doc>
                              <type>car</type>
                              <make>Ford</make>
                              <model>Taurus</model>
                              </doc>' PRESERVE WHITESPACE)),
   (8, XMLPARSE(DOCUMENT '<doc>
                              <type>person</type>
                              <name>Kim</name>
                              <town>Toronto</town>
                              <street>Elm</street>
                              </doc>' PRESERVE WHITESPACE)),
   (9, XMLPARSE(DOCUMENT '<doc>
                              <type>person</type>
                              <name>Bob</name>
                              <town>Toronto</town>
                              <street>Oak</street>
                              </doc>' PRESERVE WHITESPACE)),
   (10, XMLPARSE(DOCUMENT '<doc>
                               <type>animal</type>
                               <name>bird</name>
                               </doc>' PRESERVE WHITESPACE))
```

Prozedur

Verwenden Sie bei der Erstellung eigener C-Prozeduren die folgenden Beispiele als Referenz:

- „Datei mit externem C-Code"
- „Beispiel 1: C-SQL-Prozedur in Parameterdarstellung mit XML-Funktionen"

Datei mit externem C-Code

Das Beispiel besteht aus zwei Abschnitten: der Anweisung CREATE PROCEDURE und der externen C-Codeimplementierung der Prozedur, aus der die zugehörige Baugruppe (Assembly) erstellt werden kann.

Die C-Quellendatei, in der die Prozedurimplementierungen der folgenden Beispiele enthalten sind, heißt `gwenProc.SQC` und hat folgendes Format:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sqlda.h>
#include <sqlca.h>
#include <sqludf.h>
#include <sql.h>
#include <memory.h>

// C procedures
               ...
```

Die Dateieinbindungen sind am Anfang der Datei aufgeführt. Für die XML-Unterstützung in eingebetteten SQL-Routinen sind keine zusätzlichen Kopfdatendateien erforderlich.

Es ist wichtig, die Namen der Datei und der Funktion zu notieren, die eine bestimmte Prozedurimplementierung enthält. Diese Namen sind wichtig, weil die Klausel EXTERNAL der Anweisung CREATE PROCEDURE diese Informationen bei jeder Prozedur angeben muss, damit der DB2-Datenbankmanager die Bibliothek und den Eingangspunkt der C-Prozedur finden kann.

Beispiel 1: C-SQL-Prozedur in Parameterdarstellung mit XML-Funktionen

Dieses Beispiel zeigt Folgendes:

- Anweisung CREATE PROCEDURE einer SQL-Prozedur in Parameterdarstellung
- C-Code einer SQL-Prozedur in Parameterdarstellung mit XML-Parametern

Diese Prozedur empfängt zwei Eingabeparameter. Der erste Eingabeparameter heißt `inNum` und hat den Datentyp INTEGER. Der zweite Eingabeparameter heißt `inXML` und hat den Datentyp XML. Mit den Werten der Eingabeparameter wird eine Zeile in die Tabelle `xmlDataTable` eingefügt. Anschließend wird über eine SQL-Anweisung ein XML-Wert abgerufen. Ein weiterer XML-Wert wird mit einem XQuery-Ausdruck abgerufen. Die abgerufenen XML-Werte werden jeweils zwei Ausgabeparametern zugewiesen, `out1XML` und `out2XML`. Es werden keine Ergebnismengen zurückgegeben.

```
CREATE PROCEDURE  xmlProc1   ( IN inNUM  INTEGER,
                               IN inXML  XML as CLOB (1K),
                               OUT inXML  XML as CLOB (1K),
                               OUT inXML  XML as CLOB (1K)
                             )
LANGUAGE C
PARAMETER STYLE SQL
DYNAMIC RESULT SETS 0
FENCED
```

```
  THREADSAFE
  DETERMINISTIC
  NO DBINFO
  MODIFIES SQL DATA
  PROGRAM TYPE SUB
  EXTERNAL NAME 'gwenProc!xmlProc1' ;
  //*************************************************************************
  //  Stored Procedure: xmlProc1
  //
  //  Purpose:   insert XML data into XML column
  //
  //  Parameters:
  //
  //   IN:    inNum -- the sequence of XML data to be insert in xmldata table
  //          inXML -- XML data to be inserted
  //   OUT:   out1XML -- XML data returned - value retrieved using XQuery
  //          out2XML -- XML data returned - value retrieved using SQL
  //*************************************************************************
#ifdef __cplusplus
extern "C"
#endif
SQL_API_RC SQL_API_FN testSecA1(sqlint32* inNum,
                                SQLUDF_CLOB* inXML,
                                SQLUDF_CLOB* out1XML,
                                SQLUDF_CLOB* out2XML,
                                SQLUDF_NULLIND *inNum_ind,
                                SQLUDF_NULLIND *inXML_ind,
                                SQLUDF_NULLIND *out1XML_ind,
                                SQLUDF_NULLIND *out2XML_ind,
                                SQLUDF_TRAIL_ARGS)
{
  char *str;
  FILE *file;

  EXEC SQL INCLUDE SQLCA;

  EXEC SQL BEGIN DECLARE SECTION;
       sqlint32 hvNum1;
       SQL TYPE IS XML AS CLOB(200) hvXML1;
       SQL TYPE IS XML AS CLOB(200) hvXML2;
       SQL TYPE IS XML AS CLOB(200) hvXML3;
  EXEC SQL END DECLARE SECTION;

  /* Check null indicators for input parameters */
  if ((*inNum_ind < 0) ||  (*inXML_ind < 0)) {
     strcpy(sqludf_sqlstate, "38100");
     strcpy(sqludf_msgtext, "Received null input");
      return 0;
  }

  /* Copy input parameters to host variables */
  hvNum1 = *inNum;
  hvXML1.length = inXML->length;
  strncpy(hvXML1.data, inXML->data, inXML->length);

  /* Execute SQL statement */
  EXEC SQL
     INSERT INTO xmlDataTable (num, xdata) VALUES (:hvNum1, :hvXML1);

  /* Execute SQL statement */
  EXEC SQL
     SELECT xdata INTO :hvXML2
        FROM xmlDataTable
           WHERE num = :hvNum1;

  sprintf(stmt5, "SELECT XMLQUERY('for $x in $xmldata/doc
```

```
                                    return <carInfo>{$x/model}</carInfo>'
                                    passing by ref xmlDataTable.xdata
                                        as \"xmldata\" returning sequence)
                        FROM xmlDataTable WHERE num = ?");

  EXEC SQL PREPARE selstmt5 FROM :stmt5 ;
  EXEC SQL DECLARE c5 CURSOR FOR selstmt5;
  EXEC SQL OPEN c5 using :hvNum1;
  EXEC SQL FETCH c5 INTO :hvXML3;

  exit:

  /* Set output return code */
  *outReturnCode = sqlca.sqlcode;
  *outReturnCode_ind = 0;

   return 0;
}
```

Leistung von Routinen

Die Leistung von Routinen wird von verschiedenen Faktoren beeinflusst. Dazu zählen der Typ und die Implementierung der Routine, die Anzahl der SQL-Anweisungen in der Routine, der Komplexitätsgrad der SQL in der Routine, die Anzahl der Parameter für die Routine, die Effizienz der Logik in der Routinenimplementierung, die Fehlerbehandlung innerhalb der Routinen usw.

Da Benutzer bei der Implementierung von Routinen oft das Ziel haben, die Leistung von Anwendungen zu steigern, ist es wichtig, die Leistung der Routinen zu optimieren.

In der folgenden Tabelle finden Sie eine Beschreibung der allgemeinen Faktoren, die Einfluss auf die Routinenleistung haben, und Empfehlungen zur Steigerung der Routinenleistung durch eine Veränderung der einzelnen Faktoren. Genauere Informationen zu den Leistungsfaktoren, die Einfluss auf bestimmte Routinetypen haben, finden Sie in den Abschnitten zur Leistung und Optimierung des entsprechenden Routinentyps.

Tabelle 33. Leistungsaspekte und Empfehlungen zur Routinenleistung

Leistungsaspekt	Empfehlung zur Leistung
Routinentyp: Prozedur, Funktion, Methode	• Prozeduren, Funktionen und Methoden haben unterschiedliche Aufgaben und werden an unterschiedlichen Stellen referenziert. Aufgrund ihrer Funktionsunterschiede ist es schwer, ihre Leistung direkt zu vergleichen. • Im Allgemeinen können Prozeduren manchmal als Funktionen neu erstellt werden (insbesondere wenn sie einen skalaren Wert zurückgeben und nur Daten abfragen) und dadurch eine leichte Leistungsverbesserung erzielen. Diese Vorteile ergeben sich jedoch in der Regel aus einer Vereinfachung des für die Implementierung der SQL-Logik erforderlichen SQL. • Benutzerdefinierte Funktionen mit komplexen Initialisierungen können mithilfe von Arbeitspuffern die beim ersten Aufruf erforderlichen Werte speichern und bei den nachfolgenden Aufrufen verwenden.

Tabelle 33. Leistungsaspekte und Empfehlungen zur Routinenleistung (Forts.)

Leistungsaspekt	Empfehlung zur Leistung
Routinenimplementierung: systemdefiniert oder benutzerdefiniert	• Bei gleichwertiger Logik zeigen integrierte Routinen die beste Leistung, gefolgt von systemdefinierten Routinen, da diese eine engere Beziehung zur Datenbanksteuerkomponente als benutzerdefinierte Routinen haben. • Benutzerdefinierte Routinen können eine sehr gute Leistung erbringen, wenn sie gut codiert sind und sich an bewährte Verfahren anlehnen.
Routinenimplementierung: SQL oder externe Routinenimplementierung	• SQL-Routinen sind effizienter als externe Routinen, weil sie direkt vom DB2-Datenbankserver ausgeführt werden. • SQL-Prozeduren sind in der Regel leistungsstärker als logisch gleichwertige externe Prozeduren. • Bei einfacher Logik ist die Leistung einer SQL-Funktion mit der Leistung einer gleichwertigen externen Funktion vergleichbar. • Bei komplexer Logik, wie mathematischen Algorithmen und Funktionen zur Bearbeitung von Zeichenfolgen, für die wenig SQL erforderlich ist, empfiehlt sich die Verwendung einer externen Routine in einer Programmiersprache der unteren Ebene wie C, da diese weniger abhängig von SQL-Unterstützung ist. • Im Abschnitt zum Vergleich von Routinenimplementierungen finden Sie einen Vergleich der Funktionen (einschließlich der Leistung) der Programmiersprachenoptionen der unterstützten externen Routinen.

Tabelle 33. Leistungsaspekte und Empfehlungen zur Routinenleistung (Forts.)

Leistungsaspekt	Empfehlung zur Leistung
Programmiersprache für die Implementierung externer Routinen	• Im Abschnitt zum Vergleich externer Routinen-APIs und Programmiersprachen finden Sie einen Vergleich der Leistungsmerkmale, die Sie bei der Auswahl einer externen Routinenimplementierung berücksichtigen sollten. • Java (JDBC- und SQLJ-APIs) – Java-Routinen mit sehr großem Speicherbedarf lassen sich am besten unter Angabe der Klausel FENCED NOT THREADSAFE erstellen. Java-Routinen mit mittlerem Speicherbedarf können mit der Klausel FENCED THREADSAFE angegeben werden. – Beim Aufrufen von abgeschirmten, threadsicheren Java-Routinen versuchen DB2-Datenbanksysteme, einen Java-Prozess im abgeschirmten Modus mit Thread zu wählen, dessen Java-Zwischenspeicher groß genug für die Ausführung der Routine ist. Wenn es nicht gelingt, große Zwischenspeicherverbraucher in deren eigenem Prozess zu isolieren, können Fehler aufgrund von ungenügendem Java-Zwischenspeicher in Java **db2fmp**-Prozessen mit Multithread die Folge sein. FENCED THREADSAFE-Routinen dagegen sind leistungsstärker, weil sie eine geringe Anzahl von JVMs gemeinsam nutzen können. • C und C++ – Generell leisten C- und C++-Routinen mehr als andere externe Routinenimplementierungen und genauso viel wie SQL-Routinen. – Zur Optimierung ihrer Leistung sollten Sie C- und C++-Routinen im 32-Bit-Format kompilieren, wenn sie in einer DB2-Instanz mit 32-Bit implementiert werden, und im 64-Bit-Format, wenn sie in einer DB2-Instanz mit 64-Bit implementiert werden. • COBOL – Im Allgemeinen ist die Leistung von COBOL gut, jedoch wird COBOL nicht zu Routinenimplementierung empfohlen.
Anzahl der SQL-Anweisungen in der Routine	• Routinen sollten mehrere SQL-Anweisungen enthalten, da andernfalls der Systemaufwand für den Routinenaufruf in keinem Verhältnis zur Leistung steht. • Logik, die mehrere Datenbankabfragen ausführen, Zwischenergebnisse verarbeiten und schließlich eine Teilmenge der bearbeiteten Daten zurückgeben muss, ist am besten für die Kapselung von Routinen geeignet. Beispiele für diese Art von Logik sind die Filterung komplexer Daten oder umfangreiche Aktualisierungen, bei denen nach zugehörigen Daten gesucht werden muss. Auf dem Datenbankserver erfolgt einer umfangreiche SQL-Verarbeitung, und nur die kleinere Datenergebnismenge wird an das aufrufende Programm zurückgegeben.

Tabelle 33. Leistungsaspekte und Empfehlungen zur Routinenleistung (Forts.)

Leistungsaspekt	Empfehlung zur Leistung
Komplexität der SQL-Anweisungen in der Routine	• Es ist sinnvoll, sehr komplexe Abfragen in Ihre Routinen einzubinden, um die höheren Speicher- und Durchsatzkapazitäten des Datenbankservers auszunutzen. • Machen Sie sich keine Sorgen, dass die SQL-Anweisungen zu komplex sein könnten.
Ausführung von statischem oder dynamischem SQL in Routinen	• Im Allgemeinen ist statisches SQL leistungsstärker als dynamisches SQL. In Routinen gibt es keine weiteren Unterschiede bei der Verwendung von statischem bzw. dynamischem SQL.
Anzahl der Parameter für Routinen	• Eine möglichst geringe Anzahl der Parameter für Routinen kann eine Leistungssteigerung der entsprechenden Routine bewirken, da auf diese Weise die Anzahl der Puffer minimiert wird, die zwischen der Routine und deren aufrufendem Programm übergeben werden müssen.
Datentypen der Routinenparameter	• Sie können die Leistung von Routinen verbessern, indem Sie in der Routinendefinition VARCHAR-Parameter anstelle von CHAR-Parametern verwenden. Die Verwendung von VARCHAR-Datentypen anstelle von CHAR-Datentypen verhindert, dass DB2-Datenbanksysteme die Parameter vor der Übergabe mit Leerzeichen auffüllen, und verringert die Übertragungsdauer der Parameter im Netz. Wenn Ihre Clientanwendung beispielsweise die Zeichenfolge "KURZE ZEICHENFOLGE" an eine Routine übergibt, die einen CHAR(200)-Parameter erwartet, muss die DB2-Datenbank den Parameter mit 182 Leerzeichen auffüllen, ein Nullabschlusszeichen an die Zeichenfolge anhängen und anschließend die gesamte Zeichenfolge mit 200 Zeichen und dem Nullabschlusszeichen über das Netz an die Routine senden. Wird dagegen dieselbe Zeichenfolge *"KURZE ZEICHENFOLGE"* an eine Routine übergeben, die einen VARCHAR(200)-Parameter erwartet, übergibt die DB2-Datenbank einfach die Zeichenfolge mit 18 Zeichen und einem Nullabschlusszeichen über das Netz.
Initialisierung von Parametern für Routinen	• Es empfiehlt sich, Eingabeparameter stets für Routinen zu initialisieren, besonders wenn die Eingabewerte der Routinenparameter Null sind. Bei Routinenparametern mit Nullwerten kann anstelle eines Puffers in Normalgröße ein kürzerer oder leerer Puffer an die Routine übergeben werden, was eine Leistungssteigerung bewirken kann.
Anzahl der lokalen Variablen in Routinen	• Die Minimierung der Anzahl der in einer Routine deklarierten lokalen Variablen kann der Leistungssteigerung dienen, weil dadurch die Anzahl der in der Routine ausgeführten SQL-Anweisungen minimiert wird. • Versuchen Sie generell, so wenige Variablen wie möglich zu verwenden. Verwenden Sie Variablen mehrfach, sofern dies nicht zu semantischen Unklarheiten führt.

Tabelle 33. Leistungsaspekte und Empfehlungen zur Routinenleistung (Forts.)

Leistungsaspekt	**Empfehlung zur Leistung**
Initialisierung von lokalen Variablen in Routinen	• Es empfiehlt sich, nach Möglichkeit mehrere lokale Variablen in einer einzigen SQL-Anweisung zu initialisieren, da dies die Gesamtdauer der SQL-Ausführung für die Routine verkürzt.
Anzahl der von Prozeduren zurückgegebenen Ergebnismengen	• Wenn Sie die Anzahl der Ergebnismengen verringern können, die von einer Routine zurückgegeben werden, kann die Routinenleistung verbessert werden.
Größe der von Routinen zurückgegebenen Ergebnismengen	• Stellen Sie sicher, dass die Abfrage, mit der das Ergebnis definiert wird, die zurückgegebene Spalten- und Zeilenanzahl bei jeder von einer Routine zurückgegebenen Ergebnismenge möglichst stark filtert. Die Rückgabe überflüssiger Datenspalten oder -zeilen ist ineffizient und kann dafür führen, dass die Routinenleistung nicht optimal ist.
Effizienz der Logik in Routinen	• Wie bei jeder Anwendung kann die Leistung einer Routine durch einen schlecht implementierten Algorithmus beeinträchtigt werden. Versuchen Sie, bei der Programmierung von Routinen möglichst effizient vorzugehen, und verwenden Sie generell so viele empfohlene, bewährte Codierungsverfahren wie möglich. • Analysieren Sie Ihr SQL, und reduzieren Sie nach Möglichkeit Ihre Abfrage auf die einfachste Form. Dies kann oft dadurch erreicht werden, dass Sie CASE-Ausdrücke anstelle von CASE-Anweisungen verwenden oder mehrere SQL-Anweisungen in eine einzige Anweisung komprimieren, die einen CASE-Ausdruck als Schalter verwendet.

Tabelle 33. Leistungsaspekte und Empfehlungen zur Routinenleistung (Forts.)

Leistungsaspekt	Empfehlung zur Leistung
Laufzeitmodus der Routine (Angabe der Klausel FENCED oder NOT FENCED)	**Verwendung der Klausel NOT FENCED:** • Im Allgemeinen ist es besser, Ihre Routine mit der Klausel NOT FENCED zu erstellen, was ihre Ausführung in demselben Prozess wie der DB2-Datenbankmanager bewirkt, als mit der Klausel FENCED, was zur Folge hat, dass sie in einem speziellen DB2-Datenbankprozess ausgeführt wird, der sich außerhalb des Adressraums der Steuerkomponente befindet. • Zwar lässt die Ausführung von Routinen im nicht abgeschirmten Modus eine bessere Routinenleistung erwarten, der Benutzercode in nicht abgeschirmten Routinen kann jedoch die versehentliche oder beabsichtigte Beschädigung der Datenbank oder deren Steuerstrukturen zur Folge haben. Verwenden Sie die Klausel NOT FENCED nur, wenn Sie die maximale Leistung benötigen und die Routine für sicher halten. (Informationen zur Beurteilung und Abmilderung der Risiken bei der Registrierung von C/C++-Routinen als NOT FENCED finden Sie im Abschnitt zur Sicherheit von Routinen. Wenn die Routine nicht sicher genug für die Ausführung im Datenbankmanagerprozess ist, sollten Sie sie mit der Klausel FENCED erstellen. Damit möglichst wenig Code erstellt und ausgeführt wird, der möglicherweise unsicher ist, verlangt die DB2-Datenbank von einem Benutzer, der NOT FENCED-Routinen erstellen will, das besondere Zugriffsrecht CREATE_NOT_FENCED_ROUTINE. • Wenn während der Ausführung einer NOT FENCED-Routine eine abnormale Beendigung erfolgt, versucht der Datenbankmanager, eine geeignete Fehlerbehebung auszuführen, wenn die Routine als NO SQL registriert ist. Bei Routinen, die nicht als NO SQL definiert sind, schlägt der Datenbankmanager jedoch fehl. • Wenn in einer NOT FENCED-Routine Daten des Typs GRAPHIC oder DBCLOB verwendet werden, muss sie mit der Option WCHARTYPE NOCONVERT vorkompiliert werden.

Tabelle 33. Leistungsaspekte und Empfehlungen zur Routinenleistung (Forts.)

Leistungsaspekt	Empfehlung zur Leistung
Laufzeitmodus der Routine (Angabe der Klausel FENCED oder NOT FENCED)	**Verwendung der Klausel FENCED THREADSAFE** • Mit der Klausel FENCED THREADSAFE erstellte Routinen werden in demselben Prozess wie andere Routinen ausgeführt. Genauer gesagt, Nicht-Java-Routinen nutzen gemeinsam einen bestimmten Prozess, Java-Routinen dagegen einen anderen, von den in anderen Sprache geschriebenen Routinen getrennten Prozess. Diese Trennung schützt die Java-Routinen vor den potenziell fehlerträchtigeren Routinen, die in anderen Sprachen geschrieben sind. Darüber hinaus enthalten Java-Routinen eine JVM, die einen höheren Speicheraufwand bedingt und von anderen Routinetypen nicht verwendet wird. Mehrere Aufrufe von FENCED THREADSAFE-Routinen nutzen Ressourcen gemeinsam und verursachen dadurch weniger Systemaufwand als Routinen des Typs FENCED NOT THREADSAFE, die jeweils in einem eigenen dedizierten Prozess ausgeführt werden. • Wenn Sie glauben, dass Ihre Routine sicher genug ist, um in demselben Prozess wie andere Routinen ausgeführt zu werden, können Sie sie mit der Klausel THREADSAFE registrieren. Ähnlich wie bei NOT FENCED-Routinen finden Sie Informationen zur Beurteilung und Abmilderung der Risiken bei der Registrierung von C/C++-Routinen als FENCED THREADSAFE im Abschnitt zu den Sicherheitsaspekten von Routinen. • Bei der abnormalen Beendigung einer Routine des Typs FENCED THREADSAFE wird nur der Thread beendet, der diese Routine ausführt. Die Ausführung der übrigen Routinen in dem Prozess wird fortgesetzt. Die Störung, die die abnormale Beendigung dieses Threads verursachte, kann jedoch andere Routinenthreads in dem Prozess beeinträchtigen und zu Traps, Blockierungen und Datenbeschädigungen führen. Nachdem ein Thread abnormal beendet wurde, werden mit dem Prozess keine neuen Routinen mehr aufgerufen. Sobald alle aktiven Benutzer ihre Vorgänge in diesem Prozess abgeschlossen haben, wird er beendet. • Java-Routinen werden bei der Registrierung als THREADSAFE eingestuft, sofern Sie nicht anderes angeben. Alle übrigen LANGUAGE-Typen sind standardmäßig NOT THREADSAFE. Routinen, die LANGUAGE OLE und OLE DB verwenden, können nicht als THREADSAFE angegeben werden. • Routinen des Typs NOT FENCED müssen THREADSAFE sein. Es ist nicht möglich, Routinen als NOT FENCED NOT THREADSAFE zu registrieren (SQLCODE -104). • UNIX-Benutzer können ihre Java- und C-Prozesse vom Typ THREADSAFE anzeigen, indem sie nach **db2fmp** (Java) oder **db2fmp** (C) suchen.

Tabelle 33. Leistungsaspekte und Empfehlungen zur Routinenleistung (Forts.)

Leistungsaspekt	Empfehlung zur Leistung
Laufzeitmodus der Routine (Angabe der Klausel FENCED oder NOT FENCED)	**Modus FENCED NOT THREADSAFE** • Routinen des Typs FENCED NOT THREADSAFE werden jeweils in einem eigenen dedizierten Prozess ausgeführt. Wenn Sie viele Routinen ausführen, kann dies die Leistung des Datenbanksystems beeinträchtigen. Wenn die Routine nicht sicher genug ist, um in demselben Prozess wie andere Routinen ausgeführt zu werden, sollten Sie sie mit der Klausel NOT THREADSAFE registrieren. • Unter UNIX werden Prozesse vom Typ NOT THREADSAFE bei einem **db2fmp**-Pool vom Typ NOT THREADSAFE als `db2fmp (`*`pid`*`)` (dabei steht *pid* für die Prozess-ID des Agenten, der den Prozess im abgeschirmten Modus verwendet) oder als `db2fmp (inaktiv)` angezeigt.
Ebene des SQL-Zugriffs in der Routine: NO SQL, CONTAINS SQL, READS SQL DATA, MODIFIES SQL DATA	• Routinen, die mit einer Klausel für eine niedrigere SQL-Zugriffsebene erstellt werden, sind leistungsstärker als Routinen, die mit einer Klausel für eine höhere SQL-Zugriffsebene erstellt werden. Deklarieren Sie daher Ihre Routinen mit der Klausel für die restriktivste SQL-Zugriffsebene. Erstellen Sie beispielsweise eine Routine, die nur SQL-Daten liest, nicht mit der Klausel MODIFIES SQL DATA, sondern mit der restriktiveren Klausel READS SQL DATA.
Angabe, ob die Routine deterministisch ist (Angabe der Klausel DETERMINISTIC oder NOT DETERMINISTIC)	• Ob Sie eine Routine mit der Klausel DETERMINISTIC oder NOT DETERMINISTIC deklarieren, hat keinen Einfluss auf deren Leistung.
Anzahl und Komplexität externer, von der Routine ausgeführter Aktionen (Angabe der Klausel EXTERNAL ACTION)	• Je nach der Anzahl der externen Aktionen und der Komplexität der von einer externen Routine ausgeführten Aktionen kann die Leistung der Routine beeinträchtigt werden. Folgende Einflussgrößen sind dabei relevant: Datenaustausch im Netz, Lese- oder Schreibzugriff auf Dateien, benötigte Zeit für die Ausführung der externen Aktion und das Risiko von Blockierungen im Code oder Verhalten der externen Aktion.
Routinenaufruf bei Eingabeparametern mit Nullwerten (Angabe der Klausel CALLED ON NULL INPUT)	• Wenn der Empfang von Nullwerten in Eingabeparametern dazu führt, dass keine Logik ausgeführt wird und die Routine die Werte sofort zurückgibt, können Sie Ihre Routine so modifizieren, dass sie bei der Erkennung von Nullwerten in Eingabeparametern nicht vollständig aufgerufen wird. Zum Erstellen einer Routine, deren Aufruf in einem frühen Stadium beendet wird, wenn Eingabeparameter der Routine empfangen werden, müssen Sie die Routine erstellen und die Klausel CALLED ON NULL INPUT angeben.

Tabelle 33. Leistungsaspekte und Empfehlungen zur Routinenleistung (Forts.)

Leistungsaspekt	Empfehlung zur Leistung
Prozedurparameter des Typs XML	• Die Übergabe von Parametern mit dem Datentyp XML ist bei externen Prozeduren, die in der Programmiersprache Java C oder JAVA implementiert sind, deutlich ineffizienter als bei SQL-Prozeduren. Ziehen Sie bei der Übergabe eines oder mehrerer XML-Parameter die Verwendung von SQL-Prozeduren anstelle von externen Prozeduren in Betracht. • XML-Daten werden umgesetzt, wenn sie als Parameter IN, OUT oder INOUT an gespeicherte Prozeduren übergeben werden. Wenn Sie mit gespeicherten Java-Prozeduren arbeiten, müssen eventuell die Zwischenspeichergröße (Konfigurationsparameter **java_heap_sz**) je nach Anzahl und Größe der XML-Argumente sowie die Anzahl externer gespeicherter Prozeduren, die gleichzeitig ausgeführt werden, erhöht werden.

Sobald Routinen erstellt und implementiert sind, lässt sich unter Umständen schwerer ermitteln, welche umgebungs- und routinespezifischen Faktoren die Routinenleistung beeinflussen. Daher ist es wichtig, beim Entwurf von Routinen Leistungsaspekte zu berücksichtigen.

Beispielanwendungen

pureXML - Beispiele

Mithilfe der Funktion pureXML lassen sich korrekt formatierte XML-Dokumente in ihrem hierarchischen Format in Spalten einer Tabelle speichern. XML-Spalten werden mit dem neuen XML-Datentyp definiert. Da die Funktion pureXML vollständig in das DB2-Datenbanksystem integriert ist, können Sie auf die gespeicherten XML-Daten zugreifen und diese verwalten, indem Sie die DB2-Datenserverfunktionalität nutzen. Diese Funktionalität umfasst Verwaltungsunterstützung, Unterstützung für Anwendungsentwicklung sowie effiziente Such- und Abruffunktionen von XML dank der Unterstützung von XQuery, SQL oder dem kombinierten Einsatz von SQL/XML-Funktionen.

Zur Veranschaulichung der XML-Unterstützung werden diverse Beispiele bereitgestellt, die sich in folgende Kategorien einteilen lassen:

Verwaltungsbeispiele

Diese Beispiele veranschaulichen folgende Funktionen:

- XML-Schemaunterstützung: Schemaregistrierung und Prüfung von XML-Dokumenten
- Unterstützung für die Indexierung von XML-Daten: Indizes für verschiedene Knotentypen von XML-Werten
- Unterstützung von Dienstprogrammen für XML: Unterstützung von IMPORT, EXPORT, RUNSTATS, db2look und db2batch für den XML-Datentyp

Beispiele für Anwendungsentwicklung

Diese Beispiele veranschaulichen folgende Funktionen:

- XML-Einfügung, -Aktualisierung und -Löschung: Einfügen von XML-Werten in XML-Spalten, Aktualisieren und Löschen vorhandener Werte

- Unterstützung für XML-Parsing, -Prüfung und -Serialisierung: Implizites und explizites Parsing kompatibler Datentypen, Prüfen von XML-Dokumenten, Serialisieren von XML-Daten
- Hybridnutzung von SQL und XQuery: Verwendung von SQL/XML-Funktionen wie XMLTABLE, XMLQUERY sowie des Vergleichselements XMLEXISTS
- Unterstützung des XML-Datentyps in SQL- und externen Prozeduren: Übergeben von XML-Daten an SQL-Prozeduren und externe Prozeduren durch Einschließen von Parametern des Datentyps XML
- Unterstützung für die Dekomposition mittels eines mit Annotationen versehenen XML-Schemas: Dekomposition von XML-Dokumenten auf der Basis von XML-Schemata, die mit Annotationen versehen sind
- XML-Veröffentlichungsfunktionen: Erstellen von XML-Werten mithilfe von Funktionen

XQuery-Beispiele
Diese Beispiele veranschaulichen die Verwendung von Achsen, FLWOR-Ausdrücken, und Abfragen, die mit XQuery und SQL/XML geschrieben wurden.

Diese Beispiele befinden sich an der folgenden Position:

- Unter Windows: `%DB2PATH%\sqllib\samples\xml` (dabei ist `%DB2PATH%` eine Variable, die bestimmt, wo der DB2-Datenbankserver installiert wird)
- Unter UNIX: `$HOME/sqllib/samples/xml` (dabei ist $HOME das Ausgangsverzeichnis des Instanzeigners)

pureXML - Verwaltungsbeispiele

Diese Beispiele veranschaulichen die Unterstützung von pureXML für verschiedene Verwaltungsfunktionen, nämlich XML-Schemaunterstützung, Dienstprogrammunterstützung und Unterstützung für die Indexierung von XML-Daten.

Diese Beispiele sind in verschiedenen Programmiersprachen verfügbar und befinden sich in den sprachenspezifischen Unterverzeichnissen an der folgenden Position:

- Unter Windows: `%DB2PATH%\sqllib\samples\xml` (dabei ist `%DB2PATH%` eine Variable, die bestimmt, wo der DB2-Datenbankserver installiert wird)
- Unter UNIX: `$HOME/sqllib/samples/xml` (dabei ist $HOME das Ausgangsverzeichnis des Instanzeigners)

Tabelle 34. XML-Schemaunterstützung - Beispiele für Schemaregistrierung, -Prüfung und Weiterentwicklung kompatibler Schemata

Beispiele nach Sprache	Beispielprogrammname	Programmbeschreibung
CLI	`xsupdate.c`	Aktualisiert ein registriertes XML-Schema und stellt dabei sicher, dass das ursprüngliche Schema mit dem neuen Schema kompatibel ist.
C	`xmlschema.sqc`	Registriert das XML-Schema für die Datenbank und verwendet das registrierte Schema zum Prüfen und Einfügen eines XML-Dokuments.

Tabelle 34. XML-Schemaunterstützung - Beispiele für Schemaregistrierung, -Prüfung und Weiterentwicklung kompatibler Schemata (Forts.)

Beispiele nach Sprache	Beispielprogrammname	Programmbeschreibung
CLP	`xmlschema.db2`	Registriert das XML-Schema für die Datenbank und verwendet das registrierte Schema zum Prüfen und Einfügen eines XML-Dokuments.
	`xsupdate.db2`	Aktualisiert ein registriertes XML-Schema und stellt dabei sicher, dass das ursprüngliche Schema mit dem neuen Schema kompatibel ist.
JDBC	`XmlSchema.java`	Registriert das XML-Schema für die Datenbank und verwendet das registrierte Schema zum Prüfen und Einfügen eines XML-Dokuments.
	`XsUpdate.java`	Aktualisiert ein registriertes XML-Schema und stellt dabei sicher, dass das ursprüngliche Schema mit dem neuen Schema kompatibel ist.
SQLJ	`XmlSchema.sqlj`	Registriert das XML-Schema für die Datenbank und verwendet das registrierte Schema zum Prüfen und Einfügen eines XML-Dokuments.

Tabelle 35. Dienstprogrammunterstützung: Beispiele für die Unterstützung von IMPORT, EXPORT, RUNSTATS, db2look, reorg und db2batch für den XML-Datentyp

Beispiele nach Sprache	Beispielprogrammname	Programmbeschreibung
C	`xmlrunstats.sqc`	Führt RUNSTATS in einer Tabelle aus, die XML-Spalten enthält.
	`lobstoxml.sqc`	Verschiebt LOB-Daten mit den Befehlen IMPORT und EXPORT in eine XML-Spalte.
	`impexpxml.sqc`	Importiert und exportiert XML-Dokumente.
	`xmlload.sqc`	Lädt XML-Dokumente mithilfe der verschiedenen Optionen des Befehls LOAD in DB2-Tabellen.

Tabelle 35. Dienstprogrammunterstützung: Beispiele für die Unterstützung von IMPORT, EXPORT, RUNSTATS, db2look, reorg und db2batch für den XML-Datentyp (Forts.)

Beispiele nach Sprache	Beispielprogrammname	Programmbeschreibung
CLP	`xmlrunstats.db2`	Führt RUNSTATS in einer Tabelle aus, die XML-Spalten enthält.
	`xmlolic.db2`	Zeigt, wie die für eine Tabelle definierten Indizes sowie die nicht partitionierten Indizes für eine bereichspartitionierte Tabelle reorganisiert werden.
	`xmldb2batch.db2`	Unterstützung von db2batch für den XML-Datentyp.
	`xmldb2look.db2`	Unterstützung von db2look für den XML-Datentyp.
	`lobstoxml.db2`	Verschiebt LOB-Daten mit den Befehlen IMPORT und EXPORT in eine XML-Spalte.
	`impexpxml.db2`	Importiert und exportiert XML-Dokumente.
	`xmlload.db2`	Lädt XML-Dokumente mithilfe der verschiedenen Optionen des Befehls LOAD in DB2-Tabellen.
JDBC	`XmlRunstats.java`	Führt RUNSTATS in einer Tabelle aus, die XML-Spalten enthält.

Tabelle 36. Unterstützung für die Indexierung von XML-Daten: Beispiele für Indizes zu XML-Daten

Beispiele nach Sprache	Beispielprogrammname	Programmbeschreibung
C	`xmlindex.sqc`	Erstellt einen Index und verwendet ihn in einer XQuery-Abfrage.
	`xmlconst.sqc`	Erstellt mithilfe von XML-Mustern einen Index mit Längenbeschränkungen für UNIQUE und VARCHAR.
CLI	`xmlindex.c`	Erstellt einen Index und verwendet ihn in einer XQuery-Abfrage.
	`xmlconst.c`	Erstellt mithilfe von XML-Mustern einen Index mit Längenbeschränkungen für UNIQUE und VARCHAR.
CLP	`xmlindex.db2`	Erstellt einen Index und verwendet ihn in einer XQuery-Abfrage.
	`xmlconst.db2`	Erstellt mithilfe von XML-Mustern einen Index mit Längenbeschränkungen für UNIQUE und VARCHAR.

Tabelle 36. Unterstützung für die Indexierung von XML-Daten: Beispiele für Indizes zu XML-Daten (Forts.)

Beispiele nach Sprache	Beispielprogrammname	Programmbeschreibung
JDBC	`XmlIndex.java`	Erstellt einen Index und verwendet ihn in einer XQuery-Abfrage.
	`XmlConst.java`	Erstellt mithilfe von XML-Mustern einen Index mit Längenbeschränkungen für UNIQUE und VARCHAR.
SQLJ	`XmlIndex.sqlj`	Erstellt einen Index und verwendet ihn in einer XQuery-Abfrage.
	`XmlConst.sqlj`	Erstellt mithilfe von XML-Mustern einen Index mit Längenbeschränkungen für UNIQUE und VARCHAR.

pureXML - Anwendungsentwicklungsbeispiele

Diese Beispiele veranschaulichen die XML-Unterstützung für Funktionen zur Anwendungsentwicklung wie Einfügen, Aktualisieren und Löschen, XML-Parsing, -Prüfung und -Serialisierung, Hybridnutzung von SQL/XML, XML-Datentypunterstützung in SQL-Prozeduren und externen gespeicherten Prozeduren, XML-Dekomposition und SQL/XML-Veröffentlichungsfunktionen.

Diese Beispiele sind in verschiedenen Programmiersprachen verfügbar und befinden sich in den sprachenspezifischen Unterverzeichnissen an der folgenden Position:

- Unter Windows: `%DB2PATH%\sqllib\samples\xml` (dabei ist `%DB2PATH%` eine Variable, die bestimmt, wo der DB2-Datenbankserver installiert wird)
- Unter UNIX: `$HOME/sqllib/samples/xml` (dabei ist $HOME das Ausgangsverzeichnis des Instanzeigners)

Tabelle 37. pureXML - Anwendungsentwicklungsbeispiele

Beispiele nach Sprache	Beispielprogrammname	Programmbeschreibung
CLI	`xmlinsert.c`	Fügt ein XML-Dokument in eine Spalte des Datentyps XML ein.
	`xmlupdel.c`	Aktualisiert und löscht XML-Dokumente in den Tabellen.
	`xmlread.c`	Liest XML-Daten, die in Tabellen gespeichert sind.
	`reltoxmldoc.c`	Erstellt mithilfe verschiedener SQL/XML-Veröffentlichungsfunktionen ein XML-Dokument direkt aus Daten, die in relationalen Tabellen gespeichert sind.

Tabelle 37. pureXML - Anwendungsentwicklungsbeispiele (Forts.)

Beispiele nach Sprache	Beispielprogrammname	Programmbeschreibung
	xmltotable.c	Fügt mithilfe von SQL/XML-Funktionen wie XMLTABLE, XMLQUERY sowie des Vergleichselements XMLEXISTS die Daten aus einem XML-Dokument in relationale Tabellen ein.
	simple_xmlproc.c	Einfache gespeicherte Prozedur mit XML-Parametern.
	simple_xmlproc_client.c	Clientprogramm für den Aufruf der Routine in simple_xmlproc.c.
	simple_xmlproc_create.db2	CLP-Script zum Registrieren der gespeicherten Prozedur in simple_xmlproc.c.
	simple_xmlproc_drop.db2	CLP-Script zum Löschen der gespeicherten Prozedur in simple_xmlproc.c.
C	xmlinsert.sqc	Fügt ein XML-Dokument in eine Spalte des Datentyps XML ein.
	xmludfs.sqc	Übergibt den XML-Datentyp als Eingabeparameter, deklariert lokale Variablen mit XML-Datentyp und gibt Werte zurück, wenn Skalarfunktionen, abgeleitete Funktionen, benutzerdefinierte Funktionen mit SQL-Hauptteil und benutzerdefinierte Funktionen für Tabellen verwendet werden.
	xmludfs.c	Übergibt den XML-Datentyp als Eingabeparameter, deklariert lokale Variablen mit XML-Datentyp und gibt Werte zurück, wenn Skalarfunktionen, abgeleitete Funktionen, benutzerdefinierte Funktionen mit SQL-Hauptteil und benutzerdefinierte Funktionen für Tabellen verwendet werden.
	xmlupdel.sqc	Aktualisiert und löscht XML-Dokumente in den Tabellen.
	xmlread.sqc	Liest XML-Daten, die in Tabellen gespeichert sind.
	reltoxmltype.sqc	Erstellt mithilfe verschiedener SQL/XML-Veröffentlichungsfunktionen ein XML-Objekt aus Daten, die in relationalen Tabellen gespeichert sind.

Tabelle 37. pureXML - Anwendungsentwicklungsbeispiele (Forts.)

Beispiele nach Sprache	Beispielprogrammname	Programmbeschreibung
	`xmldecomposition.sqc`	Führt die Dekomposition von Daten durch, die in einer XML-Datei gespeichert sind, und fügt die Daten in Tabellen ein. Gibt die Reihenfolge der Einfügungen beim Zerlegen von XML-Dokumenten an.
	`recxmldecomp.sqc`	Registriert ein rekursives XML-Schema beim XML-Schema-Repository (XSR) und aktiviert es für die Dekomposition.
	`simple_xmlproc.sqc`	Einfache gespeicherte Prozedur mit XML-Parametern.
	`simple_xmlproc_client.db2`	CLP-Script zum Aufrufen der Routine in simple_xmlproc.sqc.
	`simple_xmlproc_create.db2`	CLP-Script zum Registrieren der gespeicherten Prozedur in simple_xmlproc.sqc.
	`simple_xmlproc_drop.db2`	CLP-Script zum Löschen der gespeicherten Prozedur in simple_xmlproc.sqc.
	`xmltrig.sqc`	Verwendet die Triggerverarbeitungsfunktion zum Erzwingen einer automatischen Prüfung eingehender XML-Dokumente.
	`xmlintegrate.sqc`	Verwendet die Funktionen XMLROW und XMLGROUP zum Zuordnen von relationalen Daten zu XML. Zeigt den standardmäßigen Übergabemechanismus von XMLQuery und die standardmäßige Spaltenangabe von XMLTABLE.
	`xmlcheckconstraint.sqc`	Erstellt mit den Vergleichselementen IS VALIDATED und IS NOT VALIDATED Tabellen mit Prüfungen auf Integritätsbedingung für eine XML-Spalte und gibt mit der Klausel ACCORDING TO XMLSCHEMA ein oder mehrere Schemata an.
	`xmlxslt.sqc`	Konvertiert mit der Funktion XSLTRANSFORM XML-Dokumente, die sich in der Datenbank befinden, mithilfe von Style-Sheets in HTML, Klartext oder andere XML-Formate.
CLP	`xmlinsert.db2`	Fügt ein XML-Dokument in eine Spalte des Datentyps XML ein.

Tabelle 37. pureXML - Anwendungsentwicklungsbeispiele (Forts.)

Beispiele nach Sprache	Beispielprogrammname	Programmbeschreibung
	xrpart.db2	Verwendet XML in bereichspartitionierten Tabellen und unterstützt lokale und globale Indizes.
	xmlpartition.db2	Verwendet XML in Umgebungen mit partitionierten Datenbanken, Tabellen mit mehrdimensionalem Clustering (MDC) und bereichspartitionierten Tabellen.
	xmlmdc.db2	Versetzt Daten von MDC-Tabellen in Tabellen ohne mehrdimensionales Clustering und verwendet Blockindizes, globale Indizes und eine schnellere Einfügung und Löschung.
	xmludfs.db2	Übergibt den XML-Datentyp als Eingabeparameter und gibt Werte zurück, wenn Skalarfunktionen, abgeleitete Funktionen, benutzerdefinierte Funktionen mit SQL-Hauptteil und benutzerdefinierte Funktionen für Tabellen verwendet werden.
	xmldbafn.db2	Verwendet integrierte DBA-Funktionen zum Ermitteln der geschätzten integrierten Länge von XML-Dokumenten.
	xmlolic.db2	Zeigt, wie die für eine Tabelle definierten Indizes sowie die nicht partitionierten Indizes für eine bereichspartitionierte Tabelle reorganisiert werden.
	xmlindgtt.db2	Verwendet deklarierte, globale, temporäre Tabellen mit dem Datentyp XML.
	xmlupdel.db2	Aktualisiert und löscht XML-Dokumente in den Tabellen.
	reltoxmldoc.db2	Erstellt mithilfe verschiedener SQL/XML-Veröffentlichungsfunktionen ein XML-Dokument direkt aus Daten, die in relationalen Tabellen gespeichert sind.
	reltoxmltype.db2	Erstellt mithilfe verschiedener SQL/XML-Veröffentlichungsfunktionen ein XML-Objekt aus Daten, die in relationalen Tabellen gespeichert sind.

Tabelle 37. pureXML - Anwendungsentwicklungsbeispiele (Forts.)

Beispiele nach Sprache	Beispielprogrammname	Programmbeschreibung
	`xmldecomposition.db2`	Führt die Dekomposition von Daten durch, die in einer XML-Datei gespeichert sind, und fügt die Daten in Tabellen ein. Gibt die Reihenfolge der Einfügungen beim Zerlegen von XML-Dokumenten an.
	`recxmldecomp.db2`	Registriert ein rekursives XML-Schema beim XML-Schema-Repository (XSR) und aktiviert es für die Dekomposition.
	`simple_xmlproc.db2`	Einfache gespeicherte Prozedur mit XML-Parametern.
	`xmltotable.db2`	Fügt mithilfe von SQL/XML-Funktionen wie XMLTABLE, XMLQUERY sowie des Vergleichselements XMLEXISTS die Daten aus einem XML-Dokument in relationale Tabellen ein.
	`xmltrig.db2`	Verwendet die Triggerverarbeitungsfunktion zum Erzwingen einer automatischen Prüfung eingehender XML-Dokumente.
	`xmlintegrate.db2`	Verwendet die Funktionen XMLROW und XMLGROUP zum Zuordnen von relationalen Daten zu XML. Zeigt den standardmäßigen Übergabemechanismus von XMLQuery und die standardmäßige Spaltenangabe von XMLTABLE.
	`xmlcheckconstraint.db2`	Erstellt mit den Vergleichselementen IS VALIDATED und IS NOT VALIDATED Tabellen mit Prüfungen auf Integritätsbedingung für eine XML-Spalte und gibt mit der Klausel ACCORDING TO XMLSCHEMA ein oder mehrere Schemata an.
	`xmlxslt.db2`	Konvertiert mit der Funktion XSLTRANSFORM XML-Dokumente, die sich in der Datenbank befinden, mithilfe von Style-Sheets in HTML, Klartext oder andere XML-Formate.
JDBC	`XmlInsert.java`	Fügt ein XML-Dokument in eine Spalte des Datentyps XML ein.

Tabelle 37. pureXML - Anwendungsentwicklungsbeispiele (Forts.)

Beispiele nach Sprache	Beispielprogrammname	Programmbeschreibung
	`XmlMdc.java`	Versetzt Daten von MDC-Tabellen in Tabellen ohne mehrdimensionales Clustering und verwendet Blockindizes, globale Indizes und eine schnellere Einfügung und Löschung.
	`XmlUdfs.java`	Übergibt den XML-Datentyp als Eingabeparameter und gibt Werte zurück, wenn Skalarfunktionen, abgeleitete Funktionen, benutzerdefinierte Funktionen mit SQL-Hauptteil und benutzerdefinierte Funktionen für Tabellen verwendet werden.
	`XmlUpDel.java`	Aktualisiert und löscht XML-Dokumente in den Tabellen.
	`XmlRead.java`	Liest XML-Daten, die in Tabellen gespeichert sind.
	`RelToXmlDoc.java`	Erstellt mithilfe von SQL/XML-Veröffentlichungsfunktionen ein XML-Dokument direkt aus Daten, die in relationalen Tabellen gespeichert sind.
	`RelToXmlType.java`	Erstellt mithilfe verschiedener SQL/XML-Veröffentlichungsfunktionen ein XML-Objekt aus Daten, die in relationalen Tabellen gespeichert sind.
	`XmlDecomposition.java`	Führt die Dekomposition von Daten durch, die in einer XML-Datei gespeichert sind, und fügt die Daten in Tabellen ein. Gibt die Reihenfolge der Einfügungen beim Zerlegen von XML-Dokumenten an.
	`RecXmlDecomp.java`	Registriert ein rekursives XML-Schema beim XML-Schema-Repository (XSR) und aktiviert es für die Dekomposition.
	`Simple_XmlProc.java`	Einfache gespeicherte Prozedur mit XML-Parametern.
	`Simple_XmlProc_Client.java`	Clientprogramm für den Aufruf der Routine in Simple_XmlProc.java.
	`Simple_XmlProc_Create.db2`	CLP-Script zum Registrieren der gespeicherten Prozedur in Simple_XmlProc.java.
	`Simple_XmlProc_Drop.db2`	CLP-Script zum Löschen der gespeicherten Prozedur in Simple_XmlProc.java.

Tabelle 37. pureXML - Anwendungsentwicklungsbeispiele (Forts.)

Beispiele nach Sprache	Beispielprogrammname	Programmbeschreibung
	`XmlToTable.java`	Fügt mithilfe von SQL/XML-Funktionen wie XMLTABLE, XMLQUERY sowie des Vergleichselements XMLEXISTS die Daten aus einem XML-Dokument in relationale Tabellen ein.
	`XmlTrig.java`	Verwendet die Triggerverarbeitungsfunktion zum Erzwingen einer automatischen Prüfung eingehender XML-Dokumente.
	`XmlCheckConstraint.java`	Erstellt mit den Vergleichselementen IS VALIDATED und IS NOT VALIDATED Tabellen mit Prüfungen auf Integritätsbedingung für eine XML-Spalte und gibt mit der Klausel ACCORDING TO XMLSCHEMA ein oder mehrere Schemata an.
SQLJ	`XmlInsert.sqlj`	Fügt ein XML-Dokument in eine Spalte des Datentyps XML ein.
	`XmlUpDel.sqlj`	Aktualisiert und löscht XML-Dokumente in den Tabellen.
	`XmlRead.sqlj`	Liest XML-Daten, die in Tabellen gespeichert sind.
	`RelToXmlDoc.sqlj`	Erstellt mithilfe von SQL/XML-Veröffentlichungsfunktionen ein XML-Dokument direkt aus Daten, die in relationalen Tabellen gespeichert sind.
	`RelToXmlType.sqlj`	Erstellt mithilfe von SQL/XML-Veröffentlichungsfunktionen ein XML-Objekt aus Daten, die in relationalen Tabellen gespeichert sind.
	`XmlToTable.sqlj`	Fügt mithilfe von SQL/XML-Funktionen wie XMLTABLE, XMLQUERY sowie des Vergleichselements XMLEXISTS die Daten aus einem XML-Dokument in relationale Tabellen ein.
	`XmlIntegrate.sqlj`	Verwendet die Funktionen XMLROW und XMLGROUP zum Zuordnen von relationalen Daten zu XML. Zeigt den standardmäßigen Übergabemechanismus von XMLQuery und die standardmäßige Spaltenangabe von XMLTABLE.

Tabelle 37. pureXML - Anwendungsentwicklungsbeispiele (Forts.)

Beispiele nach Sprache	Beispielprogrammname	Programmbeschreibung
	`XmlXslt.sqlj`	Konvertiert mit der Funktion XSLTRANSFORM XML-Dokumente, die sich in der Datenbank befinden, mithilfe von Style-Sheets in HTML, Klartext oder andere XML-Formate.
PHP	`XmlFlwor_DB2.php`	Verwendet den XQuery FLWOR-Ausdruck.
	`XmlIndex_DB2.php`	Erstellt einen Index und verwendet ihn in einer XQuery-Abfrage.
	`XmlInsert_DB2.php`	Fügt ein XML-Dokument in eine Spalte des XML-Datentyps ein.
	`XmlRead_DB2.php`	Liest XML-Daten, die in Tabellen gespeichert sind.
	`XmlRelToXmlDOC_DB2.php`	Erstellt mithilfe von SQL/XML-Veröffentlichungsfunktionen ein XML-Dokument direkt aus Daten, die in relationalen Tabellen gespeichert sind.
	`XmlRelToXmlType_DB2.php`	Erstellt mithilfe von SQL/XML-Veröffentlichungsfunktionen ein XML-Dokument aus relationalen Daten und XML-Daten.
	`XmlRunstats_DB2.php`	Führt RUNSTATS in einer Tabelle aus, die XML-Spalten enthält.
	`XmlSchema_DB2.php`	Registriert das XML-Schema für die Datenbank und verwendet das registrierte Schema zum Prüfen und Einfügen von XML-Dokumenten.
	`XmlSQLXQuery_DB2.php`	Verwendet SQL/XML-Abfragen.
	`XmlUniqueIndexes_DB2.php`	Erstellt einen Index mit Längenbeschränkungen für UNIQUE und VARCHAR.
	`XmlUpAndDel_DB2.php`	Aktualisiert und löscht XML-Dokumente in den Tabellen.
	`XmlToTable_DB2.php`	Fügt mithilfe von SQL/XML Daten von einem XML-Dokument in relationale Tabellen ein.
	`XmlXPath_DB2.php`	Führt einfache XPath-Abfragen aus.
	`XmlXQuery_DB2.php`	Führt verschachtelte XQuery FLWOR-Ausdrücke aus.

Kapitel 11. XML-Leistung

In den folgenden Abschnitten werden Aspekte zur Leistungsverbesserung erörtert, die Sie beim Arbeiten mit der Funktion pureXML berücksichtigen können.

Zugehörige Informationen:

15 Best Practices zur Leistungsoptimierung von pureXML in DB2

Funktion pureXML und Datenorganisationsschemas

Mithilfe der Funktion pureXML in Kombination mit Datenorganisationsschemata wie Umgebungen mit Datenbankpartitionierung, Tabellenpartitionierung und mehrdimensionales Clustering lässt sich die Abfrageleistung erheblich verbessern und der Aufwand für Datenpflegeaufgaben wie Reorganisierungs-, Einfüge- und Löschvorgänge deutlich senken.

Die drei folgenden Klauseln in der Anweisung CREATE TABLE enthalten einen Algorithmus, mit dem angezeigt wird, wie die Daten zu organisieren sind:

- DISTRIBUTE BY - verteilt Daten gleichmäßig über Datenbankpartitionen in einer Umgebung mit partitionierten Datenbanken (Datenbankpartitionierung).
- PARTITION BY - gibt die Tabellenpartitionierungsschemata für eine Tabelle an (Tabellenpartitionierung).
- ORGANIZE BY - gibt eine Dimension für jede Spalte oder Gruppe von Spalten an, die ein Cluster der Tabellendaten bilden (mehrdimensionales Clustering).

Weitere Informationen zu Einschränkungen bei der Verwendung der Funktion pureXML finden Sie in den Abschnitten "Einschränkungen bei der Funktion pureXML" und "Einschränkungen von Indizes für XML-Daten".

Umgebungen mit partitionierten Datenbanken

Bei der Verwendung von Tabellen in einer Umgebung mit partitionierten Datenbanken können Tabellen, die XML-Spalten enthalten, in einer Mehrpartitionsdatenbank auf mehreren Systemen gespeichert werden. Die XML-Daten können mit der Funktion DB2 pureXML verwaltet werden.

Wenn die XML-Daten über Datenbankpartitionen verteilt werden, können mehrere Prozessoren auf mehreren Systemen Informationsanforderungen verarbeiten. Datenabruf- und Aktualisierungsanforderungen werden automatisch in Teilanforderungen zerlegt und parallel auf den betreffenden Datenbankpartitionen ausgeführt.

Partitionierte Tabellen

Eine partitionierte Tabelle kann eine oder mehrere Spalten mit dem XML-Datentyp und einen oder mehrere Indizes zu XML-Daten enthalten. Die vom Benutzer erstellten Indizes zu XML-Daten können partitioniert oder nicht partitioniert sein. Ein partitionierter Index reduziert den Aufwand für Datenpflegeoperationen in Umgebungen, in denen die Klauseln ATTACH PARTITION und DETACH PARTITION der Anweisung ALTER TABLE für Rollin- und Rolloutoperationen für Tabellendaten verwendet werden.

Tabellen mit mehrdimensionalem Clustering

Tabellen mit mehrdimensionalem Clustering (MDC-Tabelle) kann eine oder mehrere Spalten mit einem XML-Datentyp enthalten und für eine XML-Spalte können ein oder mehrere Indizes zu XML-Daten erstellt werden. Mithilfe eines Index zu XML-Daten in Verbindung mit einem MDC-Blockindex lässt sich die Abfrageleistung verbessern. Ferner kann das logische Verknüpfen von Indizes über AND (Index ANDing) mithilfe eines MDC-Blockindizes mit einem Index zu XML-Daten ausgeführt werden.

Beispiel für die Verwendung eines XQuery-Umsetzungsausdrucks in einer Umgebung mit partitionierten Datenbanken.

Der XQuery-Umsetzungsausdruck wird in einer Umgebung mit partitionierten Datenbanken unterstützt und kann in Szenarios verwendet werden, die Extraktions-, Umsetzungs- und Ladeoperationen (ETL) verwenden.

Beispiel: Wenn XML-Daten aus dem Ergebnis eines Joins der beiden Quellentabellen ORDERS und PRODUCT extrahiert, in ein gewünschtes Format umgesetzt und als XML-Daten in die Zieltabelle SALES eingefügt werden, kann ein XQuery-Umsetzungsausdruck verwendet werden. Ein Umsetzungsausdruck liefert jedoch die beste Leistung, wenn die Eingabe für den Ausdruck nur eine einzelne Tabelle anstelle eines Joins mehrerer Quellentabellen referenziert.

Wenn Sie einen XQuery-Umsetzungsausdruck mit mehreren Tabellen verwenden und dabei eine einzige Anweisung schreiben, in der der Umsetzungsausdruck direkt auf das Joinergebnis angewendet wird, erzielen Sie damit normalerweise eine akzeptable Leistung, falls das Joinergebnis im Verhältnis zur Quellentabelle relativ klein ist. Ist jedoch die Anzahl der durch den Join generierten Zeilen ähnlich hoch oder höher als die Anzahl der Zeilen in den Quellentabellen, sollten Sie in Betracht ziehen, den Join und den Umsetzungsausdruck in zwei Anweisungen zu teilen.

Das folgende Beispiel veranschaulicht ein allgemeines Verfahren zum Teilen einer einzelnen Anweisung, die einen Join und eine XQuery-Umsetzungsoperation verwendet, in zwei Anweisungen. Durch die zwei Anweisungen wird sichergestellt, dass die Umsetzungsoperation in einer Umgebung mit partitionierten Datenbanken parallelisiert werden kann. Das Verfahren umfasst die folgenden Schritte:

1. Erstellen Sie eine neue Zwischentabelle in derselben Partitionsgruppe mit demselben Verteilungsschlüssel wie die Zieltabelle. Die Zwischentabelle kann eine deklarierte, globale, temporäre Tabelle sein.
2. Extrahieren Sie die Daten aus allen Quellentabellen und fügen Sie sie in die Zwischentabelle ein.
3. Setzen Sie die XML-Daten von der Zwischentabelle um und fügen Sie die umgesetzten Daten in die Zieltabelle ein.

Sowohl Schritt 2 als auch Schritt 3 können die Vorteile der Parallelverarbeitung nutzen, die in einer Umgebung mit partitionierten Datenbanken verfügbar sind, um skalierbar zu sein. In Schritt 3 sollte ein verschachtelt Umsetzungsausdruck vermieden werden.

Im Beispiel verwendete Tabellen und Daten

In dem Beispiel werden die Quellentabelle ORDERS und PRODUCTS, die Zieltabelle SALES und eine deklarierte, globale, temporäre Tabelle TEMPSALES verwendet. Es werden Daten aus der Tabelle ORDER abgerufen, die Daten werden mit

Preisinformationen aus der Tabelle PRODUCTS durch eine Joinoperation verknüpft, die daraus resultierenden Daten werden formatiert und die formatierten Daten werden in die Tabelle SALES eingefügt.

Die Tabelle ORDERS enthält die täglichen Aufträge, wobei sich die Auftrags-ID (OID) und die Auftragsinformationen in einer XML-Spalte (ORDERDETAIL) befinden. Jedes XML-Dokument enthält eine Kunden-ID (CID) und die bestellten Produkte. Die Informationen zu jedem bestellten Produkt umfassen die Produkt-ID, die Menge und die Lieferbedingungen. Mit der folgenden Anweisung CREATE wird die Beispieltabelle ORDERS erstellt:

```
CREATE TABLE ORDERS(OID BIGINT, ORDERDETAIL XML)DISTRIBUTE BY HASH (OID);
```

Mit der folgenden Anweisung INSERT wird ein Beispielauftrag in die Tabelle ORDERS eingefügt:

```
INSERT into ORDERS
values (5003, '<order>
                <cid>1001</cid>
                <product>
                  <pid>2344</pid><qty>10</qty>
                  <delivery>Overnight</delivery>
                </product>
                <product>
                   <pid>537</pid><qty>3</qty>
                   <delivery>Ground</delivery>
                </product>
               </order>');
```

Die Tabelle PRODUCTS enthält die Produktinformationen, die Produkt-ID (PID), den Produktpreis (PRICE) sowie Details zu dem Produkt in einer XML-Spalte (PRODDETAIL). Mit der folgenden Anweisung CREATE wird die Beispieltabelle PRODUCTS erstellt:

```
CREATE TABLE PRODUCTS(PID BIGINT, PRICE FLOAT, PRODDETAIL XML);
```

Mit der folgenden Anweisung INSERT werden Produktdaten in die Tabelle PRODUCTS eingefügt:

```
INSERT into PRODUCTS
values(2344, 4.99, '<product>
                      <name>10 D-Cell batteries</name>
                      <desc>D Cell battery, 10-pack</desc>
                    </product>')

INSERT into PRODUCTS
values(537, 8.99, '<product>
                     <name>Ice Scraper, small</name>
                     <desc>Basic ice scraper, 4 inches wide</desc>
                   </product>');
```

Die Tabelle SALES enthält eine Auftrags-ID (OID), eine Kunden-ID (CID), eine Produkt-ID (PID), die Auftragssumme (ITEMTOTAL) und Details zu den einzelnen Aufträgen in einer XML-Spalte (SALEDETAIL). Die einzelnen Zeilen in der Tabelle SALES enthalten Informationen zu den jeweiligen Produkten eines Auftrags. Die Tabelle SALES wird auf die Spalte mit der Auftrags-ID (OID) verteilt. Mit der folgenden Anweisung CREATE wird die Beispieltabelle SALES erstellt:

```
CREATE TABLE SALES(OID BIGINT, CID BIGINT, PID BIGINT, ITEMTOTAL FLOAT,
    SALESDETAIL XML) DISTRIBUTE BY HASH (OID);
```

Joinoperation für Tabellen und XQuery-Umsetzungsausdruck in einer einzelnen Anweisung

Mit der folgenden Anweisung INSERT werden Daten aus den Tabellen ORDER und PRODUCT mithilfe eines Joins verknüpft, es wird ein Umsetzungsausdruck für die daraus resultierenden XML-Dokumente angewendet und die aktualisierten Dokumente werden in die Tabelle SALES eingefügt.

```
INSERT into SALES
select T.OID, T.CID, T.PID, T.ITEMTOTAL,
   XMLQUERY('
     copy $new := $temp
     modify (do delete ($new/info/cid,
     $new/info/product/pid,
     $new/info/product/qty),
       do insert <orderdate>{fn:current-date()}</orderdate>
       as first into $new/info)
    return $new' passing T.SALESDETAIL as "temp")
from(
SELECT O.OID, OX.CID, OX.PID, P.PRICE * OX.QTY, OX.SALESDETAIL
FROM PRODUCTS P,
     ORDERS O,
     XMLTABLE('for $i in $details/order/product
               return document{<info> {$details/order/cid} {$i} </info>}'
               passing O.ORDERDETAIL as "details"
     columns
        CID bigint path './info/cid',
        PID bigint path './info/product/pid',
        QTY int    path './info/product/qty',
        SALESDETAIL xml path '.') as OX
WHERE P.PID = OX.PID) as T(OID, CID, PID, ITEMTOTAL, SALESDETAIL);
```

Der nächste Abschnitt veranschaulicht, wie der Tabellenjoin und die XQuery-Umsetzung, die in der vorherigen Anweisung durchgeführt wurden, in zwei separaten Anweisungen durchgeführt werden können.

Joinoperation für Tabellen und XQuery-Umsetzungsausdruck in separaten Anweisungen

Wenn das Ergebnis des Joins in der zweiten Hälfte der Anweisung in „Joinoperation für Tabellen und XQuery-Umsetzungsausdruck in einer einzelnen Anweisung" im Vergleich zur Größe der Tabellen ORDER und PRODUCT signifikant ist, kann durch Teilen der Anweisung in zwei Anweisungen eine Leistungsverbesserung erzielt werden. Beim Teilen der Anweisung in zwei Anweisungen fügt die erste Anweisung die Ergebnisse des Joins der Tabellen ORDER und PRODUCT in eine temporäre Tabelle ein. Die zweite Anweisung wendet die Umsetzung auf die XML-Dokumente in der temporären Tabelle an und fügt die aktualisierten Dokumente in die Tabelle SALES ein.

Die temporäre Tabelle ähnelt der Tabelle SALES und speichert die temporären Abfragergebnisse. Die deklarierte globale temporäre Tabelle TEMPSALES dient zur Verarbeitung der täglichen Aufträge. Damit das Optimierungsprogramm von DB2 die Abfragen, mit denen die temporäre Tabelle und die Tabelle SALES aktualisiert werden, ordnungsgemäß optimieren kann, müssen die temporäre Tabelle TEMPSALES und die Zieltabelle SALES dieselben Verteilungsschlüssel aufweisen und zu derselben Partitionsgruppe gehören. Mit der folgenden Anweisung DECLARE wird die temporäre Tabelle TEMPSALES erstellt:

```
DECLARE GLOBAL TEMPORARY TABLE SESSION.TEMPSALES LIKE SALES
    DISTRIBUTE BY HASH (OID);
```

Die folgende Anweisung SELECT enthält eine Verknüpfung der Tabellen PRODUCTS und ORDERS. Die Anweisung berechnet die Werte für die Spalte ITEMTOTAL mithilfe der Informationen aus den Tabellen PRODUCTS und ORDERS und erstellt XML-Dokumente, die in die Tabelle TEMPSALES eingefügt werden. Der Join entspricht dem Join in der Anweisung INSERT in „Joinoperation für Tabellen und XQuery-Umsetzungsausdruck in einer einzelnen Anweisung" auf Seite 332.

```
INSERT INTO SESSION.TEMPSALES
SELECT O.OID, OX.CID, OX.PID, P.PRICE * OX.QTY, OX.SALESDETAIL
FROM PRODUCTS P,
     ORDERS O,
     XMLTABLE('for $i in $details/order/product
               return document{<info> {$details/order/cid} {$i} </info>}'
               passing O.ORDERDETAIL as "details"
     columns
        CID bigint path './info/cid',
        PID bigint path './info/product/pid',
        QTY int    path './info/product/qty',
        SALESDETAIL xml path '.') as OX
WHERE P.PID = OX.PID;
```

Die von der Anweisung INSERT verwendete vorherige Anweisung SELECT enthält keinen XQuery-Umsetzungsausdruck.

Die temporäre Tabelle enthält die von der Tabelle SALES benötigten relationalen Informationen. In den XML-Dokumenten in der temporären Tabelle müssen die Produktinformationen entfernt werden und ein Auftragsdatum muss hinzugefügt werden.

Die folgende Anweisung INSERT wählt die täglichen Verkaufsinformationen in der temporären Tabelle aus und fügt sie in die Tabelle SALES ein. Die Anweisung INSERT enthält eine Anweisung SELECT, die denselben XQuery-Umsetzungsausdruck verwendet, der auch in der Anweisung INSERT in „Joinoperation für Tabellen und XQuery-Umsetzungsausdruck in einer einzelnen Anweisung" auf Seite 332 verwendet wird. Der Ausdruck modifiziert die einzelnen XML-Dokumente der temporären Tabelle, bevor er sie in die Tabelle SALES einfügt. Da die Tabellen TEMPSALES und SALES sich in derselben Partitionsgruppe befinden und einen allgemeinen Verteilungsschlüssel gemeinsam nutzen, kann das Einfügen parallelisiert werden.

```
INSERT into SALES
select T.OID, T.CID, T.PID, T.ITEMTOTAL,
   XMLQUERY('
     copy $new := $temp
     modify (do delete ($new/info/cid,
     $new/info/product/pid,
     $new/info/product/qty),
       do insert <orderdate>{fn:current-date()}</orderdate>
       as first into $new/info)
    return $new' passing T.SALESDETAIL as "temp")
from SESSION.TEMPSALES T;
```

Verwendung deklarierter temporärer Tabellen mit XML-Daten

Deklarierte globale temporäre Tabellen können bei der Implementierung von Lösungen hilfreich sein, bei denen nicht persistente Tabellen erstellt werden müssen. Eine Anwendung kann beispielsweise eine deklarierte temporäre Tabelle erstellen, um Zwischenergebnisse zu verarbeiten. Wird die Anwendungssitzung beendet, wird die Tabelle gelöscht.

Eine deklarierte temporäre Tabelle ist nur in der Sitzung vorhanden, in der sie deklariert wurde. Die Tabelle kann nicht mit anderen Sitzungen gemeinsam genutzt werden. Wenn die Sitzung beendet wird, werden die Zeilen der Tabelle und die Beschreibung der temporären Tabelle gelöscht. Deklarierte temporäre Tabellen haben keine Probleme mit Katalogkonflikten, da für deklarierte temporäre Tabellen kein Katalogeintrag vorhanden ist.

Deklarierte globale temporäre Tabellen können XML-Spalten enthalten und XML-Daten können abgefragt und aktualisiert werden. Deklarierte globale temporäre Tabellen können auch in einer Umgebung mit partitionierten Datenbanken verwendet werden. Wenn die temporäre Tabelle eine Spalte enthält, die als Partitionierungsschlüssel definiert wurde, können die für die XML-Daten in der temporären Tabelle auszuführenden Tätigkeiten auf die Datenbankpartitionen verteilt werden.

Beispiele

Bei den folgenden Beispielen wird davon ausgegangen, dass eine Tabelle mit Unternehmensmitarbeitern XML-Daten enthält. Die Mitarbeiterdaten ähneln den folgenden Mitarbeiterinformationen:

```
<company name="MyFirstComany">
  <emp id="31201" salary="60000" gender="Female">
    <name>
      <first>Laura</first>
      <last>Brown</last>
    </name>
    <dept id="M25">Finance</dept>
  </emp>
</company>
```

Mit der folgenden Anweisung CREATE wird eine Mitarbeitertabelle mit einer XML-Spalte erstellt.

```
CREATE TABLE COMPANYINFO (ID INT, DOC XML)
```

Mit den beiden folgenden Anweisungen wird eine globale temporäre Tabelle erstellt, die mit der Mitarbeitertabelle verwendet werden kann. Die beiden Anweisungen erstellen eine temporäre Tabelle, die dieselben Spaltennamen und Beschreibungen wie die Tabelle COMPANYINFO enthält. Die erste Anweisung erstellt eine temporäre Tabelle mithilfe von Spaltendefinitionen. Die zweite Anweisung erstellt eine temporäre Tabelle mithilfe der Klausel LIKE. Die Spalten der Tabelle haben denselben Namen und dieselbe Beschreibung die die Spalten der angegebenen Tabelle. Die Klausel ON COMMIT DELETE ROWS gibt an, dass alle Zeilen der Tabelle gelöscht werden, wenn während der Ausführung einer COMMIT-Operation kein WITH HOLD-Cursor für die Tabelle geöffnet ist.

```
DECLARE GLOBAL TEMPORARY TABLE INSTMPTB (ID INT, DOC XML)
    ON COMMIT DELETE ROWS in USR_TBSP

DECLARE GLOBAL TEMPORARY TABLE INSTMPTB LIKE COMPANYINFO
    ON COMMIT DELETE ROWS in USR_TBSP
```

Mit der folgenden Anweisung DECLARE wird eine temporäre Tabelle mit Basistabellenzeilenspeicherung von XML-Dokumenten erstellt. Wenn das XML-Dokument kleiner als die angegebene integrierte Länge ist, wird es in der Basistabelle gespeichert.

```
DECLARE GLOBAL TEMPORARY TABLE TEMPTB (ID INT,DOC XML INLINE LENGTH 3000)
   ON COMMIT PRESERVE ROWS NOT LOGGED
```

Wenn Sie große Datenmengen in eine globale temporäre Tabelle einfügen und Abfragen für die Daten ausführen, können Sie zur Leistungssteigerung Indizes zu XML-Daten erstellen. Mit den folgenden CREATE INDEX-Anweisungen werden beispielsweise zwei Indizes zu XML-Daten erstellt. Der erste Index bezieht sich auf die Mitarbeiter-ID in den XML-Dokumenten. Der zweite Index bezieht sich auf den Familiennamen des Mitarbeiters.

```
CREATE INDEX SESSION.TEMP_IDX ON SESSION.INSTMPTB (DOC)
     GENERATE KEY USING XMLPATTERN '/company/emp/@id'
  AS SQL INTEGER

CREATE UNIQUE INDEX SESSION.TEMP_IDX2 ON SESSION.INSTMPTB (DOC)
  GENERATE KEY USING XMLPATTERN '/company/name/last'
  AS SQL VARCHAR(100)
```

Sie können eine deklarierte globale temporäre Tabelle in einer Umgebung mit partitionierten Datenbanken erstellen, um die Vorteile der Datenbankpartitionierung zu nutzen. Mit der folgenden Anweisung DECLARE wird eine temporäre Tabelle mit dem Verteilungsschlüssel DOCID erstellt. Nachdem Sie XML-Daten zu der temporären Tabelle hinzugefügt haben, können Abfragen und andere Operationen die Vorteile der Umgebung mit partitionierten Datenbanken nutzen.

```
DECLARE GLOBAL TEMPORARY TABLE INSTMPTB (ID INT, DOC XML)
   ON COMMIT DELETE ROWS
   IN USR_TBSP
   DISTRIBUTE BY HASH (ID)
```

In dem nachstehenden XQuery-Ausdruck werden die globale temporäre Tabelle und die Tabelle COMPANYINFO verwendet. Die temporäre Tabelle enthält eine Teilmenge der Dokumente in der Tabelle COMPANYINFO. Der XQuery-Ausdruck gibt die Mitarbeiterinformationen aus der Tabelle COMANYINFO für die Mitarbeiter in der temporären Tabelle zurück, die in der Finanzabteilung arbeiten.

```
XQUERY
   for $i in db2-fn:xmlcolumn("SESSION.INSTMPTB.DOC")/company/emp
   for $j in db2-fn:xmlcolumn("COMPANYINFO.DOC ")[/company/emp/@id = $i/@id ]
   where $i/dept = "Finance"
      return $j;
```

Mit der folgenden Anweisung INSERT werden Daten aus der temporären Tabelle in die Tabelle COMPANYINFO eingefügt.

```
INSERT INTO COMPANYINFO FROM
 (SELECT ID, DOC FROM SESSION.INSTMPTB)
```

Hinweise

Bei Verwendung einer deklarierten globalen temporären Tabelle gilt Folgendes:

- Bei einer deklarierten temporären Tabelle ist die Datenzeilenkomprimierung aktiviert. Wenn der Datenbankmanager eine Leistungssteigerung ermittelt, werden die Tabellenzeilenobjekte komprimiert, einschließlich der XML-Dokumente, für die eine integrierte Speicherung im Basistabellenobjekt vorgenommen wurde. Die Datenkomprimierung des XML-Speicherobjekts einer deklarierten temporären Tabelle wird dagegen nicht unterstützt.
- Verwenden Sie in einer Umgebung mit partitionierten Datenbanken die Klausel DISTRIBUTE BY, um eine deklarierte temporäre Tabelle mit Partitionierungsschlüssel zu erstellen. Wenn Sie eine deklarierte temporäre Tabelle mithilfe der Klausel LIKE erstellen und die Quellentabelle einen Partitionierungsschlüssel hat, besitzt die temporäre Tabelle keinen Partitionierungsschlüssel.

- Für Indizes, die für deklarierte temporäre Tabellen erstellt wurden, einschließlich vom Benutzer erstellter Indizes zu XML-Daten, wird die Indexkomprimierung aktiviert.
- Vom Benutzer erstellte Indizes zu XML-Daten unterliegen denselben Einschränkungen wie Indizes, die für nicht temporäre Tabellen erstellt wurden. In einer Umgebung mit partitionierten Datenbanken werden beispielsweise eindeutige XML-Indizes zu XML-Daten nicht unterstützt.

Verwendung von Optimierungsrichtlinien mit XML-Daten und XQuery-Ausdrücken

Die in Optimierungsprofilen verwendeten DB2-Optimierungsrichtlinien unterstützen XML-Daten. Sie können Profile erstellen, die mithilfe von Richtlinien das Verfahren zum Versetzen von XML-Daten in einer Umgebung mit partitionierten Datenbanken, die Verknüpfungsfolge für XML-Datentypen und die Verwendung benutzerdefinierter Indizes zu XML-Daten steuern.

Sie können in einer Optimierungsrichtlinie für Abfragen, die auf XML-Daten zugreifen oder Indizes zu XML-Daten verwenden, folgende Optimierungstypen angeben:

- Steuerung des Verfahrens zum Versetzen von XML-Daten zwischen Partitionen in einer Umgebung mit partitionierten Datenbanken mithilfe des allgemeinen Anforderungselements DPFXMLMOVEMENT
- Steuerung der Verknüpfungsfolge bei Verknüpfungen für XML-Datentypen in Planoptimierungsrichtlinien durch die Einstellung des Attributs FIRST auf TRUE in Zugriffsanforderungselementen oder durch Verwendung von Verknüpfungsanforderugnselementen.
- Steuerung der Verwendung von Indizes zu XML-Daten mithilfe von Zugriffsanforderungselementen:
  - Angabe der Verwendung einer einzigen XML-Indexsuche für den Zugriff auf eine Tabelle mit dem Zugriffsanforderungselement XISCAN
  - Angabe der Verwendung mehrerer XML-Indexsuchen für den Zugriff auf eine Tabelle mit dem Zugriffsanforderungselement XANDOR
  - Angabe der Verwendung mehrerer relationaler und XML-Indexsuchen mit dem Zugriffsanforderungselement IXAND, wobei das Attribut TYPE auf XMLINDEX eingestellt ist
  - Angabe der Durchführung einer kostenbasierten Analyse durch das DB2-Optimierungsprogramm und Auswahl einer bestimmten XML-Indexzugriffsart für den Zugriff auf eine Tabelle mithilfe des Zugriffsanforderungselements ACCESS, wobei das Attribut TYPE auf XMLINDEX gesetzt wird. Das Optimierungsprogramm kann beispielsweise einen XML-Indexzugriff wie eine XML-Indexsuche, XANDOR oder logische Verknüpfungen von Indizes über AND mit mindestens einem XML-Index verwenden.
  - Angabe der Verwendung aller anwendbaren relationalen Indizes und Indizes zu XML-Daten für den Zugriff auf die angegebene Tabelle, unabhängig von den Kosten, mithilfe des Zugriffsanforderungselements ACCESS sowie der Einstellung XMLINDEX für das Attribut TYPE und der Einstellung TRUE für das Attribut ALLINDEXES
  - Angabe der Verwendung aller anwendbaren relationalen Indizes und Indizes zu XML-Daten in einem IXAND-Plan für den Zugriff auf die angegebene Tabelle, unabhängig von den Kosten, mithilfe des Zugriffsanforderungselements IXAND sowie der Einstellung XMLINDEX für das Attribut TYPE und der Einstellung TRUE für das Attribut ALLINDEXES

Optimierungsrichtlinien mit XML-Daten - Beispiele

Die Beispiele der Optimierungsprofile enthalten Richtlinien zu allgemeinen Anforderungen, Zugriffsmethoden und Verknüpfungsfolgen, mit deren Hilfe sich die Optimierung von Abfragen steuern lässt, die auf XML-Daten zugreifen.

Richtlinie zum Versetzen von XML-Dokumenten als Verweise

Im folgenden Optimierungsprofil gibt das allgemeine Anforderungselement DPFXMLMOVEMENT in der Richtlinie an, dass Verweise auf XML-Dokumente über den TQ-Operator im Zugriffsplan versetzt werden.

```
<?xml version="1.0" encoding="UTF-8"?>
<OPTPROFILE VERSION="9.7.0.0">
<STMTPROFILE ID="Security Tables">
  <STMTKEY SCHEMA="ST">
        SELECT *
    FROM security
    WHERE XMLEXISTS('$SDOC/Security/SecurityInfo
      /StockInfo[Industry= "OfficeSupplies"]')
  </STMTKEY>
  <OPTGUIDELINES>
    <DPFXMLMOVEMENT VALUE="REFERENCE"/>
  </OPTGUIDELINES>
</STMTPROFILE>
</OPTPROFILE>
```

Richtlinie zur Verwendung eines bestimmten Index zu XML-Daten

Im folgenden Optimierungsprofil gibt das Zugriffselement XISCAN in der Richtlinie an, dass der Zugriff auf die Tabelle SECURITY mithilfe des Index SEC_INDUSTRY erfolgen soll. Wenn das Element XISCAN keinen Indexnamen mit dem Attribut INDEX angibt, verwendet das Optimierungsprogramm für den Zugriff auf die Tabelle SECURITY den kostengünstigsten Index zu XML-Daten.

```
<?xml version="1.0" encoding="UTF-8"?>
<OPTPROFILE VERSION="9.7.0.0">
<STMTPROFILE ID="Security Tables">
  <STMTKEY SCHEMA="ST">
        SELECT *
    FROM security
    WHERE XMLEXISTS('$SDOC/Security/SecurityInfo
      /StockInfo[Industry= "OfficeSupplies"]')
  </STMTKEY>
  <OPTGUIDELINES>
    <XISCAN TABLE='SECURITY' INDEX='SEC_INDUSTRY'/>
  </OPTGUIDELINES>
</STMTPROFILE>
</OPTPROFILE>
```

Richtlinie zur Verwendung von Indizes zu XML-Daten mit Zugriff per XANDOR

In der Richtlinie im folgenden Optimierungsprofil gibt das Element XANDOR an, dass der Zugriff auf die Tabelle SECURITY mithilfe eines XANDOR-Plans aller anwendbaren Indizes zu XML-Daten erfolgen soll. Relationale Indizes werden nicht verwendet, weil diese nicht von einem XANDOR-Plan verwendet werden können.

```
<?xml version="1.0" encoding="UTF-8"?>
<OPTPROFILE VERSION="9.7.0.0">
<STMTPROFILE ID="Security Tables by Date">
  <STMTKEY SCHEMA="STBD">
        SELECT *
    FROM security
```

```
    WHERE trans_date = CURRENT DATE
      AND XMLEXISTS('$SDOC/Security/SecurityInfo
           /StockInfo[Industry= "Software"]')
      AND XMLEXISTS('$SDOC/Security/Symbol[.="IBM"]')
  </STMTKEY>
  <OPTGUIDELINES>
    <XANDOR TABLE='SECURITY' />
  </OPTGUIDELINES>
</STMTPROFILE>
</OPTPROFILE>
```

Richtlinie zur Verwendung mehrerer angegebener Indizes zu XML-Daten mit IXAND

Im folgenden Optimierungsprofil gibt das Element IXAND in der Optimierungsrichtlinie an, dass der Zugriff auf die Tabelle SECURITY mithilfe zweier Indizes zu XML-Daten, SEC_INDUSTRY und SEC_SYMBOL, erfolgen soll. Das Optimierungsprogramm generiert einen IXAND-Plan mit den beiden Indizes als Teilen des IXAND-Plans in der aufgeführten Reihenfolge.

```
<?xml version="1.0" encoding="UTF-8"?>
<OPTPROFILE VERSION="9.7.0.0">
<STMTPROFILE ID="Security Tables by Date">
  <STMTKEY SCHEMA="STBD">
        SELECT *
    FROM security
    WHERE trans_date = CURRENT DATE
      AND XMLEXISTS('$SDOC/Security/SecurityInfo
          /StockInfo[Industry= "Software"]')
      AND XMLEXISTS('$SDOC/Security/Symbol[.="IBM"]')
  </STMTKEY>
  <OPTGUIDELINES>
    <IXAND TABLE='SECURITY' TYPE='XMLINDEX'>
      <INDEX IXNAME='SEC_INDUSTRY'/>
      <INDEX IXNAME='SEC_SYMBOL'/>
    </IXAND>
  </OPTGUIDELINES>
</STMTPROFILE>
</OPTPROFILE>
```

Richtlinie zur Verwendung aller Indizes zu XML-Daten mit IXAND

Im folgenden Optimierungsprofil gibt das Element IXAND in der Optimierungsrichtlinie an, dass der Zugriff auf die Tabelle SECURITY mithilfe aller anwendbaren relationalen Indizes und Indizes zu XML-Daten erfolgen soll. Sofern ein relationaler Index für TRANS_DATE sowie Indizes zu XML-Daten für SEC_INDUSTRY und SEC_SYMBOL vorhanden sind, werden alle drei Indizes in einer vom Optimierungsprogramm festgelegten Reihenfolge logisch über AND miteinander verknüpft.

```
<?xml version="1.0" encoding="UTF-8"?>
<OPTPROFILE VERSION="9.7.0.0">
<STMTPROFILE ID="Security Tables by Date">
  <STMTKEY SCHEMA="STBD">
        SELECT *
    FROM security
    WHERE trans_date = CURRENT DATE
    AND XMLEXISTS('$SDOC/Security/SecurityInfo
         /StockInfo[Industry= "Software"]')
    AND XMLEXISTS('$SDOC/Security/Symbol[.="IBM"]')
  </STMTKEY>
  <OPTGUIDELINES>
```

```
    <IXAND TABLE='SECURITY' TYPE='XMLINDEX' ALLINDEXES='TRUE' />
  </OPTGUIDELINES>
</STMTPROFILE>
</OPTPROFILE>
```

Richtlinie zur Verwendung eines bestimmten führenden Index zu XML-Daten mit IXAND

Im folgenden Optimierungsprofil gibt das Element IXAND in der Optimierungsrichtlinie an, dass der Zugriff auf die Tabelle SECURITY mithilfe des IXAND-Plans erfolgen soll und dass der Index zu XML-Daten, SEC_INDUSTRY, der erste Index im Element IXAND sein muss. Das Optimierungsprogramm führt eine kostenbasierte Auswahl der Zusatzindizes für den IXAND-Plan aus. Wenn ein relationaler Index für die Spalte TRANS_DATE und ein Index zu XML-Daten für den Pfad SYMBOL verfügbar ist, werden einer davon oder beide Indizes als zusätzliche Teile des IXAND-Plans dargestellt, wenn das Optimierungsprogramm dies als vorteilhaft betrachtet.

```
<?xml version="1.0" encoding="UTF-8"?>
<OPTPROFILE VERSION="9.7.0.0">
<STMTPROFILE ID="Security Tables by Date">
  <STMTKEY SCHEMA="STBD">
        SELECT *
    FROM security
    WHERE trans_date = CURRENT DATE
      AND XMLEXISTS('$SDOC/Security/SecurityInfo
            /StockInfo[Industry= "Software"]')
      AND XMLEXISTS('$SDOC/Security/Symbol[.="IBM"]')
  </STMTKEY>
  <OPTGUIDELINES>
    <IXAND TABLE='SECURITY' TYPE='XMLINDEX' INDEX='SEC_INDUSTRY' />
  </OPTGUIDELINES>
</STMTPROFILE>
</OPTPROFILE>
```

Richtlinie zur Verwendung bestimmter XML-Indizes für die Zugriffsart

Im folgenden Optimierungsprofil gibt die Richtlinie an, dass für den Zugriff auf die Tabelle SECURITY nur bestimmte XML-Indizes verwendet werden sollen. Das Optimierungsprogramm verwendet einen XISCAN-, IXAND-, XANDOR- oder IXOR-Plan mit kostenbasierter Analyse.

```
<?xml version="1.0" encoding="UTF-8"?>
<OPTPROFILE VERSION="9.7.0.0">
<STMTPROFILE ID="Security Tables">
  <STMTKEY SCHEMA="ST">
        SELECT *
     FROM security
     WHERE XMLEXISTS('$SDOC/Security/SecurityInfo
           /StockInfo[Industry= "OfficeSupplies"]')
  </STMTKEY>
  <OPTGUIDELINES>
     <ACCESS TABLE='SECURITY' TYPE='XMLINDEX' />
  </OPTGUIDELINES>
</STMTPROFILE>
</OPTPROFILE>
```

Richtlinie zur Verwendung aller anwendbaren Indizes zu XML-Daten für die Zugriffsart

Im folgenden Optimierungsprofil gibt die Richtlinie an, dass für den Zugriff auf die Tabelle SECURITY alle anwendbaren Indizes verwendet werden soll. Die Aus-

wahl des Verfahrens bleibt dem Optimierungsprogramm überlassen. Es wird vorausgesetzt, dass zwei Indizes zu XML-Daten erstellt wurden, SEC_INDUSTRY und SEC_SYMBOL, die mit den beiden Vergleichselementen in XMLEXISTS übereinstimmen. Das Optimierungsprogramm kann entweder einen XANDOR- oder einen IXAND-Plan verwenden. Das Optimierungsprogramm trifft die Auswahl zwischen den beiden Zugriffsplänen auf der Basis einer Kostenanalyse.

```
<?xml version="1.0" encoding="UTF-8"?>
<OPTPROFILE VERSION="9.7.0.0">
<STMTPROFILE ID="Security Tables 2">
  <STMTKEY SCHEMA="ST2">
        SELECT *
    FROM security
    WHERE XMLEXISTS('$SDOC/Security/SecurityInfo
             /StockInfo[Industry= "Software"]')
    AND XMLEXISTS('$SDOC/Security/Symbol[.="IBM"]')
  </STMTKEY>
  <OPTGUIDELINES>
    <ACCESS TABLE='SECURITY' TYPE='XMLINDEX' ALLINDEXES='TRUE' />
  </OPTGUIDELINES>
</STMTPROFILE>
</OPTPROFILE>
```

Richtlinie zur Angabe von Indizes zu XML-Daten für den Zugriff mithilfe eines speziellen Index

Im folgenden Optimierungsprofil gibt die Richtlinie an, dass der Zugriff auf die Tabelle SECURITY mindestens mithilfe des Index SEC_INDUSTRY erfolgen soll. Das Optimierungsprogramm wählt auf der Basis einer Kostenanalyse einen der folgenden Zugriffspläne aus:

1. Einen XISCAN-Plan mit dem Index SEC_INDUSTRY.
2. Einen IXAND-Plan, dessen erster Teil der angegebene Index ist. Das Optimierungsprogramm kann auf der Basis einer Kostenanalyse in einem IXAND-Plan mehrere Indizes verwenden. Wenn ein relationaler Index für die Spalte TRANS_DATE verfügbar ist, wird dieser Index - wie im nachstehenden Beispiel gezeigt - als zusätzlicher Teil des IXAND-Plans dargestellt, wenn das Optimierungsprogramm dies als vorteilhaft betrachtet.
3. Einen XANDOR-Plan mit dem angegebenen Index und anderen anwendbaren Indizes zu XML-Daten.

```
<?xml version="1.0" encoding="UTF-8"?>
<OPTPROFILE VERSION="9.7.0.0">
<STMTPROFILE ID="Security Tables by Date">
  <STMTKEY SCHEMA="STBD">
        SELECT *
    FROM security
    WHERE trans_date = CURRENT DATE
      AND XMLEXISTS('$SDOC/Security/SecurityInfo
             /StockInfo[Industry= "Software"]')
      AND XMLEXISTS('$SDOC/Security/Symbol[.="IBM"]')
  </STMTKEY>
  <OPTGUIDELINES>
    <ACCESS TABLE='SECURITY' TYPE='XMLINDEX' INDEX='SEC_INDUSTRY' />
  </OPTGUIDELINES>
</STMTPROFILE>
</OPTPROFILE>
```

Die folgende Optimierungsrichtlinie gibt an, dass auf der Basis einer Kostenanalyse einer der folgenden Zugriffspläne ausgewählt werden soll:

- IXAND-Plan mit den Indizes SEC_INDUSTRY und SEC_SYMBOL in der angegebenen Reihenfolge

- XANDOR-Plan mit allen anwendbaren XML-Indizes

```
<OPTGUIDELINES>
  <ACCESS TABLE='SECURITY' TYPE='XMLINDEX'>
    <INDEX IXNAME='SEC_INDUSTRY'/>
    <INDEX IXNAME='SEC_SYMBOL'/>
  </ACCESS>
</OPTGUIDELINES>
```

Richtlinie zur Steuerung der Verknüpfungsfolge und zur Angabe von Indizes zu XML-Daten für die Zugriffsart

Im folgenden Optimierungsprofil enthalten die Optimierungsrichtlinien zwei Elemente. Das erste Richtlinienelement gibt an, dass die Tabelle CUSTACC beim Verknüpfen der Tabellen in der Klausel FROM an der äußersten Position angezeigt werden muss und dass für den Zugriff auf die Tabelle CUSTACC bestimmte Indizes zu XML-Daten verwendet werden müssen. Das zweite Richtlinienelement gibt an, dass der Zugriff auf die Tabelle ORDER mithilfe eines XANDOR-Plans erfolgen soll. Das Optimierungsprogramm verwendet in dem XANDOR-Plan alle anwendbaren Indizes zu XML-Daten. Die Reihenfolge der Indizes wird vom Optimierungsprogramm ausgewählt.

```
<?xml version="1.0" encoding="UTF-8"?>
<OPTPROFILE VERSION="9.7.0.0">
<STMTPROFILE ID="Order and Security Tables">
  <STMTKEY SCHEMA="OST">
     SELECT ordqty, orddate, ordid, security, lasttrade
     FROM order, security, custacc,
       XMLTABLE('$ODOC/FIXML/Order'
          COLUMNS ordid VARCHAR(10) PATH '@ID',
                  orddate date PATH '@TrdDt',
                  ordqty float PATH 'OrdQty/@Qty') AS T1,
       XMLTABLE(' $SDOC/Security'
          COLUMNS security varchar(50) PATH 'Name',
                  lasttrade float PATH 'Price/LastTrade') AS T2
          WHERE XMLEXISTS('
         $SDOC/Security[Symbol/fn:string(.)
             = $ODOC/FIXML/Order/Instrmt/@Sym/fn:string(.)]')
       and XMLEXISTS(
         '$ODOC/FIXML/Order[@Acct/fn:string(.)
              = $CADOC/Customer/Accounts/Account/@id/fn:string(.)]')
        and XMLEXISTS('$CADOC/Customer[@id = 1011]')
       ORDER BY ordqty desc
  </STMTKEY>
  <OPTGUIDELINES>
      <ACCESS TABLE='CUSTACC' TYPE='XMLINDEX' FIRST='TRUE' />
      <XANDOR TABLE='ORDER' />
  </OPTGUIDELINES>
</STMTPROFILE>
</OPTPROFILE>
```

Richtlinien für einen XQuery-Ausdruck

Das folgende Optimierungsprofil enthält einen XQuery-Ausdruck, der aus der Tabelle SECURITY1 Bestandsinformationen zu Unternehmen zurückgibt, die in der Softwarebranche tätig sind. Die Richtlinie gibt an, dass der Zugriff auf die Tabelle über den einzelnen XML-Index 'XI1' erfolgen soll.

```
<?xml version="1.0" encoding="UTF-8"?>
<OPTPROFILE VERSION="9.7.0.0">
<STMTPROFILE ID="Security Tables by Industry">
  <STMTKEY SCHEMA="STBD">
      xquery
    for $sinfo1 in db2-fn:xmlcolumn("SECURITY1.SDOC")/Security/SecurityInfo
        /StockInfo[Industry="Software"]
```

```
    return $sinfo1
  </STMTKEY>
  <OPTGUIDELINES>
    <XISCAN TABLE='SECURITY1' INDEX='XI1'/>
  </OPTGUIDELINES>
</STMTPROFILE>
</OPTPROFILE>
```

Das folgende Optimierungsprofil enthält einen XQuery-Ausdruck, der Bestandsinformationen zu einem Unternehmen zurückgibt, das in der Software- und Elektronikbranche tätig ist.

Die Richtlinie gibt an, dass die Tabelle SECURITY2 die äußere Tabelle der Verknüpfung sein soll und dass der Zugriff auf die Tabelle SECURITY2 über den Index zu XML-Daten 'XI2' erfolgen soll. Die Richtlinie gibt außerdem an, dass die Tabelle SECURITY1 die innere Tabelle der Verknüpfung sein soll und dass der Zugriff auf die Tabelle SECURITY1 über den Index zu XML-Daten 'XI1' erfolgen soll.

```
<?xml version="1.0" encoding="UTF-8"?>
<OPTPROFILE VERSION="9.7.0.0">
<STMTPROFILE ID="Security Tables by Industry">
  <STMTKEY SCHEMA="STBD">
    <![CDATA[ xquery
     for $sinfo1 in db2-fn:xmlcolumn("SECURITY1.SDOC")/Security
          /SecurityInfo/StockInfo[Industry="Software"]
     for $sinfo2 in db2-fn:xmlcolumn("SECURITY2.SDOC")/Security
          /SecurityInfo/StockInfo[Industry="Electronics"]
     where $sinfo1 = $sinfo2
     return <stock> {$sinfo1} </stock> ]]>
  </STMTKEY>
  <OPTGUIDELINES>
     <JOIN>
        <ACCESS TABLE='SECURITY2' TYPE='XMLINDEX' INDEX='XI2'/>
        <ACCESS TABLE='SECURITY1' TYPE='XMLINDEX' INDEX='XI1'/>
     </JOIN>
  </OPTGUIDELINES>
</STMTPROFILE>
</OPTPROFILE>
```

Ein Abschnitt CDATA, der mit <![CDATA[beginnt und mit]]> endet, umschließt den Anweisungsschlüssel im Element STMTKEY, da der Anweisungsschlüssel die XML-Sonderzeichen < und > enthält. Der Profil-Parser ignoriert die XML-Tags im Abschnitt CDATA, aber das Optimierungsprogramm verwendet den gesamten Anweisungsschlüssel, um ein Anweisungsprofil mit der entsprechenden Anweisung in einer Anwendung abzugleichen.

Bevorzugung von DMS-Tabellenbereichen für die Leistung des pureXML-Datenspeichers

Für leistungsempfindliche Anwendungen, insbesondere für solche, die mit umfangreichen INSERT-Aktivitäten verbunden sind, wird dringend empfohlen, vom Datenbankmanager verwaltete Tabellenbereiche (DMS-Tabellenbereiche) zu verwenden.

Wenn Sie eine Verschlechterung der Abfrageleistung im Zusammenhang mit dem pureXML-Datenspeicher feststellen und vom System verwaltete Tabellenbereiche (SMS-Tabellenbereiche) verwenden, sollten Sie einen Wechsel zu DMS-Tabellenbereichen in Betracht ziehen.

Die Verwendung der DMS-Funktionalität bietet außerdem die Möglichkeit, autonome Computerfunktionen (Autonomic-Computing) in DB2 zu nutzen.

Kapitel 12. XML-Datencodierung

Die Codierung von XML-Daten kann aus den Daten selbst abgeleitet werden (*intern codierte* Daten) oder aus externen Quellen (*extern codierte* Daten).

Der Anwendungsdatentyp, der dazu verwendet wird, die XML-Daten zwischen der Anwendung und der XML-Spalte auszutauschen, bestimmt, wie die Codierung abgeleitet wird.

- XML-Daten, die den Anwendungsdatentyp 'Zeichendaten' oder 'Grafikdaten' aufweisen, werden als extern codiert betrachtet. Wie bei Zeichendaten und Grafikdaten wird bei XML-Daten mit diesem Datentyp davon ausgegangen, dass sie in der Anwendungscodepage codiert sind.
- XML-Daten mit einem binären Anwendungsdatentyp oder Binärdaten mit einem Zeichendatentyp werden als intern codiert betrachtet.

Extern codierte XML-Daten können interne Codierungen enthalten, z. B. wenn ein XML-Dokument mit einem Zeichendatentyp eine Codierungsdeklaration enthält. Wenn Sie extern codierte Daten an eine DB2-Datenbank senden, prüft der Datenbankmanager, ob interne Codierungen vorhanden sind.

Wenn als externe und interne Codierung nicht Unicode verwendet wird, dann muss die gültige CCSID (ID des codierten Zeichensatzes), die der internen Codierung zugeordnet ist, mit der externen Codierung übereinstimmen. Andernfalls tritt ein Fehler auf. Wenn als externe und interne Codierung Unicode verwendet wird und die Codierungsschemata nicht übereinstimmen, ignoriert der DB2-Datenbankserver die interne Codierung.

Intern codierte XML-Daten

Für XML-Daten in einem binären Anwendungsdatentyp gilt die interne Codierung. Bei der internen Codierung bestimmt der Inhalt der Daten die entsprechende Codierung. Das DB2-Datenbanksystem leitet die interne Codierung entsprechend dem XML-Standard vom Dokumentinhalt ab.

Die interne Codierung wird von drei Komponenten abgeleitet:

Unicode-BOM (BOM = Byte Order Mark, Byteanordnungsmarkierung)
: Eine Bytefolge am Anfang der XML-Daten, die aus einem Unicode-Zeichencode besteht. Die BOM gibt die Byteanordnung des nachfolgenden Texts an. Der DB2-Datenbankmanager erkennt eine BOM nur für XML-Daten. Bei XML-Daten, die in einer Nicht-XML-Spalte gespeichert sind, interpretiert der Datenbankmanager einen BOM-Wert wie normale Zeichen oder Binärwerte.

XML-Deklaration
: Eine Verarbeitungsinstruktion am Anfang eines XML-Dokuments. Die Deklaration enthält Details zum Rest des XML-Dokuments.

Codierungsdeklaration
: Ein optionaler Teil der XML-Deklaration, der die Codierung für die im Dokument verwendeten Zeichen angibt.

Der DB2-Datenbankmanager verwendet die folgende Prozedur zur Bestimmung der Codierung:

1. Wenn die Daten eine Unicode-BOM enthalten, bestimmt die BOM die Codierung. In der folgenden Tabelle sind die BOM-Typen und die daraus resultierenden Datencodierungen aufgeführt:

Tabelle 38. Byteanordnungsmarkierungen und die daraus resultierende Dokumentcodierung

BOM-Typ	BOM-Wert	Codierung
UTF-8	X'EFBBBF'	UTF-8
UTF-16 Big Endian	X'FEFF'	UTF-16
UTF-16 Little Endian	X'FFFE'	UTF-16
UTF-32 Big Endian	X'0000FEFF'	UTF-32
UTF-32 Little Endian	X'FFFE0000'	UTF-32

2. Wenn die Daten eine XML-Deklaration enthalten, ist die Codierung davon abhängig, ob eine Codierungsdeklaration vorhanden ist.
   - Wenn eine Codierungsdeklaration vorhanden ist, ist die Codierung der Wert des Codierungsattributs. Beispiel: Die Codierung ist EUC-JP für XML-Daten mit der folgenden XML-Deklaration:
     ```
     <?xml version="1.0" encoding="EUC-JP"?>
     ```
   - Wenn eine Codierungsdeklaration und eine BOM vorhanden sind, muss die Codierungsdeklaration mit der Codierung der BOM übereinstimmen. Andernfalls tritt ein Fehler auf.
   - Wenn weder eine Codierungsdeklaration noch eine BOM vorhanden ist, stellt der Datenbankmanager die Codierung anhand der XML-Deklaration fest:
     - Wenn die XML-Deklaration in ASCII-Einzelbytezeichen vorliegt, ist die Codierung des Dokuments UTF-8.
     - Wenn die XML-Deklaration in ASCII-Doppelbytezeichen vorliegt, ist die Codierung des Dokuments UTF-16.
3. Wenn keine XML-Deklaration und keine BOM vorhanden sind, ist die Codierung des Dokuments UTF-8.

Codierungsaspekte bei der Speicherung oder Weitergabe von XML-Daten

XML-Daten müssen für die Speicherung in einer DB2-Tabelle korrekt codiert sein. Die Codierung muss beachtet werden, wenn die Daten aus der Tabelle abgerufen und mit gespeicherten DB2-Prozeduren oder benutzerdefinierten Funktionen verwendet werden bzw. wenn sie mit externen Java-Anwendungen verwendet werden.

Codierungsaspekte der Eingabe von XML-Daten in eine Datenbank

Die interne und die externe Codierung müssen beim Speichern von XML-Daten in einer DB2-Tabelle berücksichtigt werden.

Die folgenden Regeln müssen beachtet werden:

- Wenn als interne und externe Codierung nicht Unicode verwendet wird, *muss* bei extern codierten XML-Daten (Daten, die unter Verwendung von Zeichendatentypen an den Datenbankserver gesendet werden) jede intern codierte Deklaration mit der externen Codierung übereinstimmen. Andernfalls tritt ein Fehler auf, und der Datenbankmanager weist den Befehl zurück.

Wenn als externe und interne Codierung Unicode verwendet wird und die Codierungsschemata nicht übereinstimmen, ignoriert der DB2-Datenbankserver die interne Codierung.

- Bei intern codierten XML-Daten (Daten die unter Verwendung von Binärdatentypen an den Datenbankserver gesendet werden), *muss* die Anwendung sicherstellen, dass die Daten präzise Codierungsinformationen enthalten.

Codierungsaspekte beim Abrufen von XML-Daten aus einer Datenbank

Beim Abrufen von XML-Daten aus einer DB2-Tabelle dürfen keine Daten verloren gehen oder abgeschnitten werden.

Zu Datenverlust kann es kommen, wenn Zeichen in den Quellendaten nicht in der Codierung der Zieldaten dargestellt werden können. Daten werden möglicherweise abgeschnitten, wenn die Konvertierung in den Zieldatentyp zu einer Erweiterung der Daten führt.

Datenverlust ist bei Java- und .NET-Anwendungen ein geringeres Problem als bei anderen Anwendungstypen, da Java- und .NET-Zeichenfolgedatentypen die Unicode-Codierung UTF-16 oder UCS-2 verwenden. Ein Abschneiden der Daten kann auftreten, da es zu einer Erweiterung der Daten kommen kann, wenn UTF-8-Zeichen in die UTF-16- oder UCS-2-Codierung konvertiert werden.

Codierungsaspekte bei der Weitergabe von XML-Daten in Routinenparametern

In einem DB2-Datenbanksystem stehen verschiedene XML-Datentypen für Parameter in einer gespeicherten Prozedur oder der Definition einer benutzerdefinierten Funktion zur Verfügung.

Die folgenden XML-Datentypen stehen zur Verfügung:

XML Für SQL-Prozeduren.

XML AS CLOB
Für externe SQL-Prozeduren und externe benutzerdefinierte Funktionen.

Daten in XML AS CLOB-Parametern unterliegen der Zeichenkonvertierung, wenn die Anwendungscodierung nicht UTF-8 ist. Vermeiden Sie den Systemaufwand der Zeichenkonvertierung in einer externen benutzerdefinierten Funktion oder gespeicherten Prozedur. Für die Parameter in der aufrufenden Anwendung kann eine beliebiger Zeichendaten- oder Grafikdatentyp verwendet werden, die Quellendaten dürfen jedoch keine Codierungsdeklaration enthalten. Es kann eine zusätzliche Codepagekonvertierung stattfinden, wodurch die Codierungsinformationen ungenau werden. Wenn in der Anwendung eine weitere syntaktische Analyse erfolgt, führt dies möglicherweise zu fehlerhaften Daten.

Codierungsaspekte bei XML-Daten in JDBC-, SQLJ- und .NET-Anwendungen

Normalerweise sind für Java-Anwendungen weniger XML-Codierungsaspekte zu berücksichtigen als für CLI-Anwendungen oder Anwendungen mit eingebettetem SQL. Während die Codierungsaspekte für intern codierte XML-Daten für alle Anwendungen identisch sind, ist die Situation bei extern codierten Daten in Java-Anwendungen vereinfacht, da die Anwendungscodepage stets Unicode ist.

Allgemeine Empfehlungen für die Eingabe von XML-Daten in Java-Anwendungen

- Wenn sich die Eingabedaten in einer Datei befinden, lesen Sie die Daten als Binärdatenstrom ein (`setBinaryStream`), damit sie vom Datenbankmanager als intern codierte Daten verarbeitet werden.
- Befinden sich die Eingabedaten in einer Java-Anwendungsvariablen, bestimmt die Auswahl des Anwendungsvariablentyps, ob der DB2-Datenbankmanager eine interne Codierung verwendet. Wenn die Daten als Zeichendatentyp (z. B. `setString`) eingegeben werden, werden sie vom Datenbankmanager von UTF-16 (Anwendungscodepage) in UTF-8 konvertiert, bevor sie gespeichert werden.

Allgemeine Empfehlungen für die Ausgabe von XML-Daten in Java-Anwendungen

- Bei der Ausgabe von XML-Daten als nicht-binäre Daten in eine Datei muss zu den Ausgabedaten eine interne XML-Codierung hinzugefügt werden.

  Die Codierung für das Dateisystem ist möglicherweise nicht Unicode, sodass Zeichenfolgedaten beim Speichern in der Datei einer Konvertierung unterzogen werden können. Wenn Daten als Binärdaten in eine Datei geschrieben werden, findet keine Konvertierung statt.

  Bei Java-Anwendungen fügt der Datenbankserver keine explizite Deklaration für eine implizite XML-Serialisierungsoperation hinzu. Wenn Sie die Ausgabedaten in den Typ `com.ibm.db2.jcc.DB2Xml` umsetzen und eine der Methoden `getDB2Xml`*xxx* aufrufen, fügt der JDBC-Treiber eine Codierungsdeklaration hinzu, wie in der folgenden Tabelle dargestellt.

getDB2Xml*xxx*	**Codierung in der Deklaration**
`getDB2XmlString`	ISO-10646-UCS-2
`getDB2XmlBytes(String` *`targetEncoding`*`)`	Durch *targetEncoding* angegebene Codierung
`getDB2XmlAsciiStream`	US-ASCII
`getDB2XmlCharacterStream`	ISO-10646-UCS-2
`getDB2XmlBinaryStream(String` *`targetEncoding`*`)`	Durch *targetEncoding* angegebene Codierung

  Bei einer expliziten Funktion XMLSERIALIZE mit der Angabe INCLUDING XMLDECLARATION fügt der Datenbankserver die Codierung hinzu, und der JDBC-Treiber ändert sie nicht. Die vom Datenbankserver hinzugefügte explizite Codierung ist UTF-8. Abhängig davon, wie der Wert von der Anwendung abgerufen wird, entspricht die tatsächliche Codierung der Daten möglicherweise nicht der expliziten internen Codierung.
- Wenn die Anwendung die Ausgabedaten an einen XML-Parser sendet, müssen Sie die Daten in einer binären Anwendungsvariable mit der Codierung UTF-8, UCS-2 oder UTF-16 abrufen.

Auswirkungen der XML-Codierung und -Serialisierung auf die Datenkonvertierung

Das Verfahren zur Angabe der Codierung von XML-Daten (intern oder extern) und das Verfahren zur XML-Serialisierung beeinflussen die Umsetzung der XML-Daten bei deren Übergabe zwischen einer Datenbank und einer Anwendung.

Codierungsszenarios für die Eingabe intern codierter XML-Daten in eine Datenbank

Die Beispiele veranschaulichen, wie sich die interne Codierung auf die Konvertierung und das mögliche Abschneiden von Daten während der Eingabe von XML-Daten in eine XML-Spalte auswirkt.

Grundsätzlich werden durch die Verwendung eines binären Anwendungsdatentyps Codepagekonvertierungsprobleme während der Eingabe in eine Datenbank auf ein Minimum reduziert.

Szenario 1

Codierungsquelle	Wert
Datencodierung	UTF-8 Unicode-Eingabedaten mit oder ohne UTF-8-BOM oder XML-Codierungsdeklaration
Anwendungsdatentyp	Binär
Anwendungscodepage	Nicht verfügbar

Beispieleingabeanweisungen:

```
INSERT INTO T1 (XMLCOL) VALUES (?)
INSERT INTO T1 (XMLCOL) VALUES
    (XMLPARSE(DOCUMENT CAST(? AS BLOB) PRESERVE WHITESPACE))
```

Zeichenkonvertierung: Keine.

Datenverlust: Keiner.

Abgeschnittene Daten: Keine.

Szenario 2

Codierungsquelle	Wert
Datencodierung	UTF-16 Unicode-Eingabedaten mit UTF-16-BOM oder XML-Codierungsdeklaration
Anwendungsdatentyp	Binär
Anwendungscodepage	Nicht verfügbar

Beispieleingabeanweisungen:

```
INSERT INTO T1 (XMLCOL) VALUES (?)
INSERT INTO T1 (XMLCOL) VALUES
    (XMLPARSE(DOCUMENT CAST(? AS BLOB) PRESERVE WHITESPACE))
```

Zeichenkonvertierung: Der DB2-Datenbankserver konvertiert die Daten aus UTF-16 in UTF-8, wenn er die XML-Syntaxanalyse (das XML-Parsing) für das Speichern in der XML-Spalte durchführt.

Datenverlust: Keiner.

Abgeschnittene Daten: Keine.

Szenario 3

Codierungsquelle	Wert
Datencodierung	ISO-8859-1-Eingabedaten mit XML-Codierungsdeklaration
Anwendungsdatentyp	Binär
Anwendungscodepage	Nicht verfügbar

Beispieleingabeanweisungen:

```
INSERT INTO T1 (XMLCOL) VALUES (?)
INSERT INTO T1 (XMLCOL) VALUES
    (XMLPARSE(DOCUMENT CAST(? AS BLOB) PRESERVE WHITESPACE))
```

Zeichenkonvertierung: Das DB2-Datenbanksystem konvertiert die Daten von CCSID 819 in UTF-8, wenn es die XML-Syntaxanalyse für das Speichern in der XML-Spalte durchführt.

Datenverlust: Keiner.

Abgeschnittene Daten: Keine.

Szenario 4

Codierungsquelle	Wert
Datencodierung	Shift_JIS-Eingabedaten mit XML-Codierungsdeklaration
Anwendungsdatentyp	Binär
Anwendungscodepage	Nicht verfügbar

Beispieleingabeanweisungen:

```
INSERT INTO T1 (XMLCOL) VALUES (?)
INSERT INTO T1 (XMLCOL) VALUES
    (XMLPARSE(DOCUMENT CAST(? AS BLOB) PRESERVE WHITESPACE))
```

Zeichenkonvertierung: Das DB2-Datenbanksystem konvertiert die Daten von CCSID 943 in UTF-8, wenn es die XML-Syntaxanalyse für das Speichern in der XML-Spalte durchführt.

Datenverlust: Keiner.

Abgeschnittene Daten: Keine.

Codierungsszenarios für die Eingabe extern codierter XML-Daten in eine Datenbank

Die Beispiele veranschaulichen, wie sich die externe Codierung auf die Konvertierung und das mögliche Abschneiden von Daten während der Eingabe von XML-Daten in eine XML-Spalte auswirkt.

Grundsätzlich treten bei der Verwendung eines Anwendungszeichendatentyps keine Probleme bei der Codepagekonvertierung während der Eingabe in eine Datenbank auf.

Nur Szenario 1 und Szenario 2 treffen auf Java- und .NET-Anwendungen zu, da die Anwendungscodepage für Java- und .NET-Anwendungen stets Unicode ist.

Szenario 1

Codierungsquelle	Wert
Datencodierung	UTF-8 Unicode-Eingabedaten mit oder ohne entsprechende Codierungsdeklaration oder BOM
Anwendungsdatentyp	Zeichen
Anwendungscodepage	1208 (UTF-8)

Beispieleingabeanweisungen:

```
INSERT INTO T1 (XMLCOL) VALUES (?)
INSERT INTO T1 (XMLCOL) VALUES
    (XMLPARSE(DOCUMENT CAST(? AS CLOB) PRESERVE WHITESPACE))
```

Zeichenkonvertierung: Keine.

Datenverlust: Keiner.

Abgeschnittene Daten: Keine.

Szenario 2

Codierungsquelle	Wert
Datencodierung	UTF-16 Unicode-Eingabedaten mit oder ohne entsprechende Codierungsdeklaration oder BOM
Anwendungsdatentyp	Grafikdaten
Anwendungscodepage	Beliebige SBCS-Codepage oder CCSID 1208

Beispieleingabeanweisungen:

```
INSERT INTO T1 (XMLCOL) VALUES (?)
INSERT INTO T1 (XMLCOL) VALUES
    (XMLPARSE(DOCUMENT CAST(? AS DBCLOB) PRESERVE WHITESPACE))
```

Zeichenkonvertierung: Das DB2-Datenbanksystem konvertiert die Daten von UTF-16 in UTF-8, wenn es die XML-Syntaxanalyse für das Speichern in der XML-Spalte durchführt.

Datenverlust: Keiner.

Abgeschnittene Daten: Bei der Konvertierung von UTF-16 in UTF-8 werden aufgrund der Erweiterung der Daten möglicherweise Daten abgeschnitten.

Szenario 3

Codierungsquelle	Wert
Datencodierung	ISO-8859-1-Eingabedaten mit oder ohne entsprechende Codierungsdeklaration
Anwendungsdatentyp	Zeichen
Anwendungscodepage	819

Beispieleingabeanweisungen:

```
INSERT INTO T1 (XMLCOL) VALUES (?)
INSERT INTO T1 (XMLCOL) VALUES
    (XMLPARSE(DOCUMENT CAST(? AS CLOB) PRESERVE WHITESPACE))
```

Zeichenkonvertierung: Das DB2-Datenbanksystem konvertiert die Daten von CCSID 819 in UTF-8, wenn es die XML-Syntaxanalyse für das Speichern in der XML-Spalte durchführt.

Datenverlust: Keiner.

Abgeschnittene Daten: Keine.

Szenario 4

Codierungsquelle	Wert
Datencodierung	Shift_JIS-Eingabedaten mit oder ohne entsprechende Codierungsdeklaration
Anwendungsdatentyp	Grafikdaten
Anwendungscodepage	943

Beispieleingabeanweisungen:

```
INSERT INTO T1 VALUES (?)
INSERT INTO T1 VALUES
    (XMLPARSE(DOCUMENT CAST(? AS DBCLOB)))
```

Zeichenkonvertierung: Das DB2-Datenbanksystem konvertiert die Daten von CCSID 943 in UTF-8, wenn es die XML-Syntaxanalyse für das Speichern in der XML-Spalte durchführt.

Datenverlust: Keiner.

Abgeschnittene Daten: Keine.

Codierungsszenarios für das Abrufen intern codierter XML-Daten aus einer Datenbank

Beispiele veranschaulichen, wie sich die Zielcodierung und die Anwendungscodepage während des Abrufens von XML-Daten mit impliziter Serialisierung auf die Datenkonvertierung, das Abschneiden von Daten und die interne Codierung auswirken.

Nur Szenario 1 und Szenario 2 treffen auf Java- und .NET-Anwendungen zu, da die Anwendungscodepage für Java-Anwendungen stets Unicode ist. Im Allgemeinen stellt die Codepagekonvertierung für Java- und .NET-Anwendungen kein Problem dar.

Szenario 1

Codierungsquelle	Wert
Zieldatencodierung	UTF-8 Unicode
Zielanwendungsdatentyp	Binär
Anwendungscodepage	Nicht verfügbar

Beispielausgabeanweisungen:

```
SELECT XMLCOL FROM T1
```

Zeichenkonvertierung: Keine.

Datenverlust: Keiner.

Abgeschnittene Daten: Keine.

Interne Codierung in serialisierten Daten: Bei Anwendungen, bei denen es sich nicht um Java- oder .NET-Anwendungen handelt, wird den Daten die folgende XML-Deklaration vorangestellt:

```
<?xml version="1.0" encoding="UTF-8" ?>
```

Bei Java-Anwendungen wird keine Codierungsdeklaration hinzugefügt, es sei denn, die Daten werden als `com.ibm.db2.jcc.DB2Xml`-Typ umgesetzt, und zum Abrufen der Daten wird die Methode `getDB2Xml`*xxx* verwendet. Welche Deklaration hinzugefügt wird, ist abhängig von der verwendeten Methode `getDB2Xml`*xxx*.

Bei .NET-applications, no encoding declaration is added or removed.

Szenario 2

Codierungsquelle	Wert
Zieldatencodierung	UTF-16 Unicode
Zielanwendungsdatentyp	Grafikdaten
Anwendungscodepage	Beliebige SBCS-Codepage oder CCSID 1208

Beispielausgabeanweisungen:

```
SELECT XMLCOL FROM T1
```

Zeichenkonvertierung: Die Daten werden von UTF-8 in UTF-16 konvertiert.

Datenverlust: Keiner.

Abgeschnittene Daten: Bei der Konvertierung von UTF-8 in UTF-16 werden aufgrund der Erweiterung der Daten möglicherweise Daten abgeschnitten.

Interne Codierung in serialisierten Daten: Bei Anwendungen, bei denen es sich nicht um Java- oder .NET-Anwendungen handelt, werden den Daten eine UTF-16-BOM (Byte Order Mark, Byteanordnungsmarkierung) sowie die folgende XML-Deklaration vorangestellt:

```
<?xml version="1.0" encoding="UTF-16" ?>
```

Bei Java-Anwendungen wird keine Codierungsdeklaration hinzugefügt, es sei denn, die Daten werden als `com.ibm.db2.jcc.DB2Xml`-Typ umgesetzt, und zum Abrufen der Daten wird die Methode `getDB2Xml`*xxx* verwendet. Welche Deklaration hinzugefügt wird, ist abhängig von der verwendeten Methode `getDB2Xml`*xxx*.

Bei .NET-Anwendungen wird keine Codierungsdeklaration hinzugefügt oder entfernt.

Szenario 3

Codierungsquelle	Wert
Zieldatencodierung	ISO-8859-1-Daten
Zielanwendungsdatentyp	Zeichen
Anwendungscodepage	819

Beispielausgabeanweisungen:

```
SELECT XMLCOL FROM T1
```

Zeichenkonvertierung: Die Daten werden von UTF-8 in CCSID 819 konvertiert.

Datenverlust: Möglich. Einige UTF-8-Zeichen können in CCSID 819 nicht dargestellt werden. Das DB2-Datenbanksystem generiert einen Fehler.

Abgeschnittene Daten: Keine.

Interne Codierung in serialisierten Daten: Den Daten wird die folgende XML-Deklaration vorangestellt:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
```

Szenario 4

Codierungsquelle	Wert
Zieldatencodierung	Windows-31J-Daten (Untermenge von Shift_JIS)
Zielanwendungsdatentyp	Grafikdaten
Anwendungscodepage	943

Beispielausgabeanweisungen:

```
SELECT XMLCOL FROM T1
```

Zeichenkonvertierung: Die Daten werden von UTF-8 in CCSID 943 konvertiert.

Datenverlust: Möglich. Einige UTF-8-Zeichen können in CCSID 943 nicht dargestellt werden. Das DB2-Datenbanksystem generiert einen Fehler.

Abgeschnittene Daten: Bei der Konvertierung von UTF-8 in CCSID 943 werden aufgrund der Erweiterung der Daten möglicherweise Daten abgeschnitten.

Interne Codierung in serialisierten Daten: Den Daten wird die folgende XML-Deklaration vorangestellt:

```
<?xml version="1.0" encoding="Windows-31J" ?>
```

Codierungsszenarios für das Abrufen von XML-Daten mit explizitem XMLSERIALIZE

Beispiele veranschaulichen, wie sich die Zielcodierung und die Anwendungscodepage während des Abrufens von XML-Daten mit einem expliziten Aufruf von XMLSERIALIZE auf die Datenkonvertierung, das Abschneiden von Daten und die interne Codierung auswirken.

Nur Szenario 1 und Szenario 2 treffen auf Java- und .NET-Anwendungen zu, da die Anwendungscodepage für Java-Anwendungen stets Unicode ist.

Szenario 1

Codierungsquelle	Wert
Zieldatencodierung	UTF-8 Unicode

Codierungsquelle	Wert
Zielanwendungsdatentyp	Binär
Anwendungscodepage	Nicht verfügbar

Beispielausgabeanweisungen:

```
SELECT XMLSERIALIZE(XMLCOL AS BLOB(1M) INCLUDING XMLDECLARATION) FROM T1
```

Zeichenkonvertierung: Keine.

Datenverlust: Keiner.

Abgeschnittene Daten: Keine.

Interne Codierung in serialisierten Daten: Den Daten wird die folgende XML-Deklaration vorangestellt:

```
<?xml version="1.0" encoding="UTF-8" ?>
```

Szenario 2

Codierungsquelle	Wert
Zieldatencodierung	UTF-16 Unicode
Zielanwendungsdatentyp	Grafikdaten
Anwendungscodepage	Beliebige SBCS-Codepage oder CCSID 1208

Beispielausgabeanweisungen:

```
SELECT XMLSERIALIZE(XMLCOL AS CLOB(1M) EXCLUDING XMLDECLARATION) FROM T1
```

Zeichenkonvertierung: Die Daten werden von UTF-8 in UTF-16 konvertiert.

Datenverlust: Keiner.

Abgeschnittene Daten: Bei der Konvertierung von UTF-8 in UTF-16 werden aufgrund der Erweiterung der Daten möglicherweise Daten abgeschnitten.

Interne Codierung in serialisierten Daten: Keine, da EXCLUDING XMLDECLARATION angegeben ist. Wenn INCLUDING XMLDECLARATION angegeben wird, gibt die interne Codierung UTF-8 anstelle von UTF-16 an. Dies kann zu XML-Daten führen, die von Anwendungsprozessen, die den Codierungsnamen benötigen, nicht syntaktisch analysiert werden können.

Szenario 3

Codierungsquelle	Wert
Zieldatencodierung	ISO-8859-1-Daten
Zielanwendungsdatentyp	Zeichen
Anwendungscodepage	819

Beispielausgabeanweisungen:

```
SELECT XMLSERIALIZE(XMLCOL AS CLOB(1M) EXCLUDING XMLDECLARATION) FROM T1
```

Zeichenkonvertierung: Die Daten werden von UTF-8 in CCSID 819 konvertiert.

Datenverlust: Möglich. Einige UTF-8-Zeichen können in CCSID 819 nicht dargestellt werden. Wenn ein Zeichen nicht in CCSID 819 dargestellt werden kann, fügt der DB2-Datenbankmanager ein Substitutionszeichen in die Ausgabe ein und gibt eine Warnung aus.

Abgeschnittene Daten: Keine.

Interne Codierung in serialisierten Daten: Keine, da EXCLUDING XMLDECLARATION angegeben ist. Wenn INCLUDING XMLDECLARATION angegeben wird, fügt der Datenbankmanager die interne Codierung für UTF-8 anstelle von ISO-8859-1 ein. Dies kann zu XML-Daten führen, die von Anwendungsprozessen, die den Codierungsnamen benötigen, nicht syntaktisch analysiert werden können.

Szenario 4

Codierungsquelle	Wert
Zieldatencodierung	Windows-31J-Daten (Untermenge von Shift_JIS)
Zielanwendungsdatentyp	Grafikdaten
Anwendungscodepage	943

Beispielausgabeanweisungen:

```
SELECT XMLSERIALIZE(XMLCOL AS CLOB(1M) EXCLUDING XMLDECLARATION) FROM T1
```

Zeichenkonvertierung: Die Daten werden von UTF-8 in CCSID 943 konvertiert.

Datenverlust: Möglich. Einige UTF-8-Zeichen können in CCSID 943 nicht dargestellt werden. Wenn ein Zeichen nicht in CCSID 943 dargestellt werden kann, fügt der Datenbankmanager ein Substitutionszeichen in die Ausgabe ein und gibt eine Warnung aus.

Abgeschnittene Daten: Bei der Konvertierung von UTF-8 in CCSID 943 werden aufgrund der Erweiterung der Daten möglicherweise Daten abgeschnitten.

Interne Codierung in serialisierten Daten: Keine, da EXCLUDING XMLDECLARATION angegeben ist. Wenn INCLUDING XMLDECLARATION angegeben wird, gibt die interne Codierung UTF-8 anstelle von Windows-31J an. Dies kann zu XML-Daten führen, die von Anwendungsprozessen, die den Codierungsnamen benötigen, nicht syntaktisch analysiert werden können.

Zuordnung zwischen intern codierten XML-Daten und CCSIDs

Der DB2-Datenbankmanager ermittelt die CCSID auf der Grundlage der internen XML-Codierung, wenn Daten von XML in einen anderen Datentyp umgesetzt werden, und er ermittelt den Namen für die interne XML-Codierung auf der Grundlage der CCSID, wenn Daten in den XML-Datentyp umgesetzt werden.

Zuordnen von Codenamen zu effektiven CCSIDs für gespeicherte XML-Daten

Wenn sich Daten, die in einer XML-Spalte gespeichert werden, in einer binären Anwendungsvariable befinden, oder wenn es sich bei diesen Daten um einen intern codierten XML-Typ handelt, prüft der DB2-Datenbankmanager die Daten, um die

Codierung festzustellen. Verfügen die Daten über eine Codierungsdeklaration, ordnet der Datenbankmanager den Codierungsnamen einer CCSID zu.

Tabelle 39 enthält eine Liste dieser Zuordnungen. Wenn ein Codierungsname nicht in Tabelle 39 aufgeführt ist, gibt der Datenbankmanager einen Fehler zurück.

Der normalisierte Codierungsname in der ersten Spalte von Tabelle 39 ist das Ergebnis der Konvertierung des Codierungsnamens in Großbuchstaben und das Entfernen aller Bindestriche, Pluszeichen, Unterstreichungszeichen, Doppelpunkte, Punkte und Leerzeichen. So ist ISO88591 beispielsweise der normalisierte Codierungsname für ISO 8859-1, ISO-8859-1 und iso-8859-1.

Tabelle 39. Codierungsnamen und effektive CCSIDs

Normalisierter Codierungsname	CCSID
437	437
646	367
813	813
819	819
850	850
852	852
855	855
857	857
862	862
863	863
866	866
869	869
885913	901
885915	923
88591	819
88592	912
88595	915
88597	813
88598	62210
88599	920
904	904
912	912
915	915
916	916
920	920
923	923
ANSI1251	1251
ANSIX341968	367
ANSIX341986	367
ARABIC	1089
ASCII7	367

Tabelle 39. Codierungsnamen und effektive CCSIDs (Forts.)

Normalisierter Codierungsname	CCSID
ASCII	367
ASMO708	1089
BIG5	950
CCSID00858	858
CCSID00924	924
CCSID01140	1140
CCSID01141	1141
CCSID01142	1142
CCSID01143	1143
CCSID01144	1144
CCSID01145	1145
CCSID01146	1146
CCSID01147	1147
CCSID01148	1148
CCSID01149	1149
CP00858	858
CP00924	924
CP01140	1140
CP01141	1141
CP01142	1142
CP01143	1143
CP01144	1144
CP01145	1145
CP01146	1146
CP01147	1147
CP01148	1148
CP01149	1149
CP037	37
CP1026	1026
CP1140	1140
CP1141	1141
CP1142	1142
CP1143	1143
CP1144	1144
CP1145	1145
CP1146	1146
CP1147	1147
CP1148	1148
CP1149	1149
CP1250	1250

Tabelle 39. Codierungsnamen und effektive CCSIDs (Forts.)

Normalisierter Codierungsname	CCSID
CP1251	1251
CP1252	1252
CP1253	1253
CP1254	1254
CP1255	1255
CP1256	1256
CP1257	1257
CP1258	1258
CP1363	1363
CP1383	1383
CP1386	1386
CP273	273
CP277	277
CP278	278
CP280	280
CP284	284
CP285	285
CP297	297
CP33722	954
CP33722C	954
CP367	367
CP420	420
CP423	423
CP424	424
CP437	437
CP500	500
CP5346	5346
CP5347	5347
CP5348	5348
CP5349	5349
CP5350	5350
CP5353	5353
CP813	813
CP819	819
CP838	838
CP850	850
CP852	852
CP855	855
CP857	857
CP858	858

Tabelle 39. Codierungsnamen und effektive CCSIDs (Forts.)

Normalisierter Codierungsname	CCSID
CP862	862
CP863	863
CP864	864
CP866	866
CP869	869
CP870	870
CP871	871
CP874	874
CP904	904
CP912	912
CP915	915
CP916	916
CP920	920
CP921	921
CP922	922
CP923	923
CP936	1386
CP943	943
CP943C	943
CP949	970
CP950	950
CP964	964
CP970	970
CPGR	869
CSASCII	367
CSBIG5	950
CSEBCDICCAFR	500
CSEBCDICDKNO	277
CSEBCDICES	284
CSEBCDICFISE	278
CSEBCDICFR	297
CSEBCDICIT	280
CSEBCDICPT	37
CSEBCDICUK	285
CSEBCDICUS	37
CSEUCKR	970
CSEUCPKDFMTJAPANESE	954
CSGB2312	1383
CSHPROMAN8	1051
CSIBM037	37

Tabelle 39. Codierungsnamen und effektive CCSIDs (Forts.)

Normalisierter Codierungsname	CCSID
CSIBM1026	1026
CSIBM273	273
CSIBM277	277
CSIBM278	278
CSIBM280	280
CSIBM284	284
CSIBM285	285
CSIBM297	297
CSIBM420	420
CSIBM423	423
CSIBM424	424
CSIBM500	500
CSIBM855	855
CSIBM857	857
CSIBM863	863
CSIBM864	864
CSIBM866	866
CSIBM869	869
CSIBM870	870
CSIBM871	871
CSIBM904	904
CSIBMEBCDICATDE	273
CSIBMTHAI	838
CSISO128T101G2	920
CSISO146SERBIAN	915
CSISO147MACEDONIAN	915
CSISO2INTLREFVERSION	367
CSISO646BASIC1983	367
CSISO88596I	1089
CSISO88598I	916
CSISOLATIN0	923
CSISOLATIN1	819
CSISOLATIN2	912
CSISOLATIN5	920
CSISOLATIN9	923
CSISOLATINARABIC	1089
CSISOLATINCYRILLIC	915
CSISOLATINGREEK	813
CSISOLATINHEBREW	62210
CSKOI8R	878

Tabelle 39. Codierungsnamen und effektive CCSIDs (Forts.)

Normalisierter Codierungsname	CCSID
CSKSC56011987	970
CSMACINTOSH	1275
CSMICROSOFTPUBLISHING	1004
CSPC850MULTILINGUAL	850
CSPC862LATINHEBREW	862
CSPC8CODEPAGE437	437
CSPCP852	852
CSSHIFTJIS	943
CSUCS4	1236
CSUNICODE11	1204
CSUNICODE	1204
CSUNICODEASCII	1204
CSUNICODELATIN1	1204
CSVISCII	1129
CSWINDOWS31J	943
CYRILLIC	915
DEFAULT	367
EBCDICATDE	273
EBCDICCAFR	500
EBCDICCPAR1	420
EBCDICCPBE	500
EBCDICCPCA	37
EBCDICCPCH	500
EBCDICCPDK	277
EBCDICCPES	284
EBCDICCPFI	278
EBCDICCPFR	297
EBCDICCPGB	285
EBCDICCPGR	423
EBCDICCPHE	424
EBCDICCPIS	871
EBCDICCPIT	280
EBCDICCPNL	37
EBCDICCPNO	277
EBCDICCPROECE	870
EBCDICCPSE	278
EBCDICCPUS	37
EBCDICCPWT	37
EBCDICCPYU	870
EBCDICDE273EURO	1141

Tabelle 39. Codierungsnamen und effektive CCSIDs (Forts.)

Normalisierter Codierungsname	CCSID
EBCDICDK277EURO	1142
EBCDICDKNO	277
EBCDICES284EURO	1145
EBCDICES	284
EBCDICFI278EURO	1143
EBCDICFISE	278
EBCDICFR297EURO	1147
EBCDICFR	297
EBCDICGB285EURO	1146
EBCDICINTERNATIONAL500EURO	1148
EBCDICIS871EURO	1149
EBCDICIT280EURO	1144
EBCDICIT	280
EBCDICLATIN9EURO	924
EBCDICNO277EURO	1142
EBCDICPT	37
EBCDICSE278EURO	1143
EBCDICUK	285
EBCDICUS37EURO	1140
EBCDICUS	37
ECMA114	1089
ECMA118	813
ELOT928	813
EUCCN	1383
EUCJP	954
EUCKR	970
EUCTW	964
EXTENDEDUNIXCODEPACKEDFORMATFORJAPANESE	954
GB18030	1392
GB2312	1383
GBK	1386
GREEK8	813
GREEK	813
HEBREW	62210
HPROMAN8	1051
IBM00858	858
IBM00924	924
IBM01140	1140
IBM01141	1141
IBM01142	1142

Tabelle 39. Codierungsnamen und effektive CCSIDs (Forts.)

Normalisierter Codierungsname	CCSID
IBM01143	1143
IBM01144	1144
IBM01145	1145
IBM01146	1146
IBM01147	1147
IBM01148	1148
IBM01149	1149
IBM01153	1153
IBM01155	1155
IBM01160	1160
IBM037	37
IBM1026	1026
IBM1043	1043
IBM1047	1047
IBM1252	1252
IBM273	273
IBM277	277
IBM278	278
IBM280	280
IBM284	284
IBM285	285
IBM297	297
IBM367	367
IBM420	420
IBM423	423
IBM424	424
IBM437	437
IBM500	500
IBM808	808
IBM813	813
IBM819	819
IBM850	850
IBM852	852
IBM855	855
IBM857	857
IBM862	862
IBM863	863
IBM864	864
IBM866	866
IBM867	867

Tabelle 39. Codierungsnamen und effektive CCSIDs (Forts.)

Normalisierter Codierungsname	CCSID
IBM869	869
IBM870	870
IBM871	871
IBM872	872
IBM902	902
IBM904	904
IBM912	912
IBM915	915
IBM916	916
IBM920	920
IBM921	921
IBM922	922
IBM923	923
IBMTHAI	838
IRV	367
ISO10646	1204
ISO10646UCS2	1200
ISO10646UCS4	1232
ISO10646UCSBASIC	1204
ISO10646UNICODELATIN1	1204
ISO646BASIC1983	367
ISO646IRV1983	367
ISO646IRV1991	367
ISO646US	367
ISO885911987	819
ISO885913	901
ISO885915	923
ISO885915FDIS	923
ISO88591	819
ISO885921987	912
ISO88592	912
ISO885951988	915
ISO88595	915
ISO885961987	1089
ISO88596	1089
ISO88596I	1089
ISO885971987	813
ISO88597	813
ISO885981988	62210
ISO88598	62210

Tabelle 39. Codierungsnamen und effektive CCSIDs (Forts.)

Normalisierter Codierungsname	CCSID
ISO88598I	916
ISO885991989	920
ISO88599	920
ISOIR100	819
ISOIR101	912
ISOIR126	813
ISOIR127	1089
ISOIR128	920
ISOIR138	62210
ISOIR144	915
ISOIR146	915
ISOIR147	915
ISOIR148	920
ISOIR149	970
ISOIR2	367
ISOIR6	367
JUSIB1003MAC	915
JUSIB1003SERB	915
KOI8	878
KOI8R	878
KOI8U	1168
KOREAN	970
KSC56011987	970
KSC56011989	970
KSC5601	970
L1	819
L2	912
L5	920
L9	923
LATIN0	923
LATIN1	819
LATIN2	912
LATIN5	920
LATIN9	923
MAC	1275
MACEDONIAN	915
MACINTOSH	1275
MICROSOFTPUBLISHING	1004
MS1386	1386
MS932	943

Tabelle 39. Codierungsnamen und effektive CCSIDs (Forts.)

Normalisierter Codierungsname	CCSID
MS936	1386
MS949	970
MSKANJI	943
PCMULTILINGUAL850EURO	858
R8	1051
REF	367
ROMAN8	1051
SERBIAN	915
SHIFTJIS	943
SJIS	943
SUNEUGREEK	813
T101G2	920
TIS20	874
TIS620	874
UNICODE11	1204
UNICODE11UTF8	1208
UNICODEBIGUNMARKED	1200
UNICODELITTLEUNMARKED	1202
US	367
USASCII	367
UTF16	1204
UTF16BE	1200
UTF16LE	1202
UTF32	1236
UTF32BE	1232
UTF32LE	1234
UTF8	1208
VISCII	1129
WINDOWS1250	1250
WINDOWS1251	1251
WINDOWS1252	1252
WINDOWS1253	1253
WINDOWS1254	1254
WINDOWS1255	1255
WINDOWS1256	1256
WINDOWS1257	1257
WINDOWS1258	1258
WINDOWS28598	62210
WINDOWS31J	943
WINDOWS936	1386

Tabelle 39. Codierungsnamen und effektive CCSIDs (Forts.)

Normalisierter Codierungsname	CCSID
XEUCTW	964
XMSWIN936	1386
XUTF16BE	1200
XUTF16LE	1202
XWINDOWS949	970

Zuordnen von CCSIDs zu Codenamen für serialisierte XML-Ausgabedaten

Als Teil einer impliziten oder expliziten XMLSERIALIZE-Operation fügt der DB2-Datenbankmanager möglicherweise eine Codierungsdeklaration am Anfang von serialisierten XML-Ausgabedaten hinzu.

Diese Deklaration hat das folgende Format:

```
<?xml version="1.0" encoding="codierung-name"?>
```

Im Allgemeinen beschreibt die Zeichensatzkennung in der Codierungsdeklaration die Codierung der Zeichen in der Ausgabezeichenfolge. Wenn beispielsweise XML-Daten für die CCSID serialisiert werden, die dem Datentyp der Zielanwendung entspricht, beschreibt die Codierungsdeklaration die CCSID-Zielanwendungsvariable. Eine Ausnahme stellt der Fall dar, in dem die Anwendung eine explizite Funktion XMLSERIALIZE mit der Option INCLUDING XMLDECLARATION ausführt. Bei der Angabe von INCLUDING XMLDECLARATION generiert der Datenbankmanager eine Codierungsdeklaration für UTF-8. Wenn der Zieldatentyp ein CLOB- oder DBCLOB-Typ ist, findet möglicherweise eine zusätzliche Codepagekonvertierung statt, wodurch die Codierungsinformationen ungenau werden können. Wenn in der Anwendung eine weitere syntaktische Analyse erfolgt, führt dies möglicherweise zu fehlerhaften Daten.

Wenn möglich, verwendet der DB2-Datenbankmanager den IANA-Registrynamen für die CCSID, wie durch den XML-Standard vorgeschrieben.

Tabelle 40. CCSIDs und entsprechende Codierungsnamen

CCSID	Codierungsname
37	IBM037
273	IBM273
277	IBM277
278	IBM278
280	IBM280
284	IBM284
285	IBM285
297	IBM297
367	US-ASCII
420	IBM420
423	IBM423
424	IBM424

Tabelle 40. CCSIDs und entsprechende Codierungsnamen (Forts.)

CCSID	Codierungsname
437	IBM437
500	IBM500
808	IBM808
813	ISO-8859-7
819	ISO-8859-1
838	IBM-Thai
850	IBM850
852	IBM852
855	IBM855
857	IBM857
858	IBM00858
862	IBM862
863	IBM863
864	IBM864
866	IBM866
867	IBM867
869	IBM869
870	IBM870
871	IBM871
872	IBM872
874	TIS-620
878	KOI8-R
901	ISO-8859-13
902	IBM902
904	IBM904
912	ISO-8859-2
915	ISO-8859-5
916	ISO-8859-8-I
920	ISO-8859-9
921	IBM921
922	IBM922
923	ISO-8859-15
924	IBM00924
932	Shift_JIS
943	Windows-31J
949	EUC-KR
950	Big5
954	EUC-JP
964	EUC-TW
970	EUC-KR

Tabelle 40. CCSIDs und entsprechende Codierungsnamen (Forts.)

CCSID	Codierungsname
1004	Microsoft-Publish
1026	IBM1026
1043	IBM1043
1047	IBM1047
1051	hp-roman8
1089	ISO-8859-6
1129	VISCII
1140	IBM01140
1141	IBM01141
1142	IBM01142
1143	IBM01143
1144	IBM01144
1145	IBM01145
1146	IBM01146
1147	IBM01147
1148	IBM01148
1149	IBM01149
1153	IBM01153
1155	IBM01155
1160	IBM-Thai
1161	TIS-620
1162	TIS-620
1163	VISCII
1168	KOI8-U
1200	UTF-16BE
1202	UTF-16LE
1204	UTF-16
1208	UTF-8
1232	UTF-32BE
1234	UTF-32LE
1236	UTF-32
1250	windows-1250
1251	windows-1251
1252	windows-1252
1253	windows-1253
1254	windows-1254
1255	windows-1255
1256	windows-1256
1257	windows-1257
1258	windows-1258

Tabelle 40. CCSIDs und entsprechende Codierungsnamen (Forts.)

CCSID	Codierungsname
1275	MACINTOSH
1363	KSC_5601
1370	Big5
1381	GB2312
1383	GB2312
1386	GBK
1392	GB18030
4909	ISO-8859-7
5039	Shift_JIS
5346	windows-1250
5347	windows-1251
5348	windows-1252
5349	windows-1253
5350	windows-1254
5351	windows-1255
5352	windows-1256
5353	windows-1257
5354	windows-1258
5488	GB18030
8612	IBM420
8616	IBM424
9005	ISO-8859-7
12712	IBM424
13488	UTF-16BE
13490	UTF-16LE
16840	IBM420
17248	IBM864
17584	UTF-16BE
17586	UTF-16LE
62209	IBM862
62210	ISO-8859-8
62211	IBM424
62213	IBM862
62215	ISO-8859-8
62218	IBM864
62221	IBM862
62222	ISO-8859-8
62223	windows-1255
62224	IBM420
62225	IBM864

Tabelle 40. CCSIDs und entsprechende Codierungsnamen (Forts.)

CCSID	Codierungsname
62227	ISO-8859-6
62228	windows-1256
62229	IBM424
62231	IBM862
62232	ISO-8859-8
62233	IBM420
62234	IBM420
62235	IBM424
62237	windows-1255
62238	ISO-8859-8-I
62239	windows-1255
62240	IBM424
62242	IBM862
62243	ISO-8859-8-I
62244	windows-1255
62245	IBM424
62250	IBM420

Kapitel 13. Dekomposition mithilfe eines mit Annotationen versehenen XML-Schemas

Die Dekomposition mithilfe eines mit Annotationen versehenen XML-Schemas (auch als 'Zerlegung' bezeichnet) ist das Speichern des Inhalts eines XML-Dokuments in Spalten relationaler Tabellen. Bei der Dekomposition werden Annotationen verwendet, die in einem XML-Schema angegeben sind.

Nach der Zerlegung eines XML-Dokuments haben die eingefügten Daten den SQL-Datentyp der Spalte, in die sie eingefügt werden.

Ein XML-Schema besteht aus einem oder mehreren XML-Schemadokumenten. Bei der Dekomposition mithilfe eines mit Annotationen versehenen XML-Schemas (auch als "schemabasierte Dekomposition" bezeichnet) steuern Sie die Dekomposition, indem Sie das XML-Schema eines Dokuments mit Dekompositionsannotationen versehen. Diese Annotationen geben Details an, wie beispielsweise den Namen der Zieltabelle und Spalte, in der die XML-Daten gespeichert werden sollen, das SQL-Standardschema, das verwendet werden soll, wenn das SQL-Schema der Zieltabelle nicht identifiziert ist, die Reihenfolge, in der XML-Daten in Zieltabellen eingefügt werden sollen sowie eine mögliche Konvertierung des Inhalts vor dem Speichern. Die Zusammenfassung der Dekompositionsannotationen enthält weitere Beispiele zu den Informationen, die mithilfe dieser Annotationen angegeben werden können.

Die mit Annotationen versehenen Schemadokumente müssen im XML-Schema-Repository (XSR) gespeichert und dort registriert sein. Das Schema muss dann für die Dekomposition aktiviert werden.

Nach der erfolgreichen Registrierung des mit Annotationen versehenen Schemas kann die Dekomposition für ein einzelnes XML-Dokument entweder durch Aufrufen einer der gespeicherten Dekompositionsprozeduren oder durch Ausführen des Befehls DECOMPOSE XML DOCUMENT durchgeführt werden. Verwenden Sie zur Dekomposition mehrerer in einer Spalte gespeicherter XML-Dokumente die gespeicherte Prozedur XDB_DECOMP_XML_FROM_QUERY oder den Befehl DECOMPOSE XML DOCUMENTS.

Die schemabasierte Dekomposition kann inaktiviert oder nicht ausführbar gemacht werden. Weitere Informationen hierzu finden Sie im Thema zur Inaktivierung der Dekomposition.

Vorteile der Dekomposition mithilfe eines mit Annotationen versehenen XML-Schemas

Die Dekomposition mithilfe eines mit Annotationen versehenen XML-Schemas kann eine Lösung zur Speicherung von XML-Dokumenten darstellen, die mit einem bestimmten XML-Schema arbeiten, wobei dieses XML-Schema jedoch nicht unmittelbar mit den Definitionen der Tabellen übereinstimmt, in denen die Dokumente gespeichert werden.

Wenn das XML-Schema nicht genau mit der Tabellenstruktur übereinstimmt, kann es erforderlich sein, entweder das XML-Schema oder das relationale Schema (oder auch beide Schemata) so anzupassen, dass die Dokumente in die Tabellenstruktur

passen. Änderungen an dem XML-Schema bzw. am relationalen Schema sind jedoch nicht in jedem Fall möglich oder könnten mit extremen Kosten verbunden sein, insbesondere wenn vorhandene Anwendungen eine bestimmte Struktur des relationalen Schemas voraussetzen.

Die Dekomposition mithilfe eines mit Annotationen versehenen XML-Schemas bietet eine Lösung für dieses Problem, indem sie Ihnen eine Zerlegung von Dokumenten, die auf neuen oder vorhandenen XML-Schemata basieren, in neue oder vorhandene Tabellen ermöglicht. Sie ist das Resultat verschiedener Eigenschaften, die durch die Dekomposition mithilfe eines mit Annotationen versehenen XML-Schemas zur Verfügung gestellt werden. Diese Eigenschaften, die in Form von Annotationen ausgedrückt werden, die den XML-Schemadokumenten hinzugefügt werden, bieten flexible Möglichkeiten für die Zuordnung zwischen einer XML-Schemastruktur und einer relationalen Tabellenstruktur.

Dekomposition von XML-Dokumenten mithilfe von mit Annotationen versehenen XML-Schemata

Wenn Sie Teile eines XML-Dokuments in Spalten einer oder mehrerer Tabellen speichern möchten, können Sie hierfür die Dekomposition mithilfe von mit Annotationen versehenen XML-Schemata verwenden. Bei dieser Dekompositionsart wird ein XML-Dokument für das Speichern in Tabellen zerlegt, wobei die in einem registrierten, mit Annotationen versehenen XML-Schema angegebenen Annotationen als Basis verwendet werden.

Informationen zu diesem Vorgang

Dekomposition von XML-Dokumenten mit XML-Schemata, die mit Annotationen versehen sind:

Vorgehensweise

1. Wenn Sie mit einer Datenbank arbeiten, die mit älteren Versionen von DB2-Datenbankprodukten erstellt wurde: Den Befehl BIND mithilfe der Listendatei `xdb.lst` ausführen, die sich im Verzeichnis `sqllib/bnd` befindet.
2. Die Schemadokumente mit Annotationen zur XML-Dekomposition versehen.
3. Die Schemadokumente registrieren und das Schema für die Dekomposition aktivieren.
4. Wenn eines oder mehrere der registrierten Schemadokumente, die zum XML-Schema gehören, geändert wurden: Alle Dokumente für dieses XML-Schema erneut registrieren und das XML-Schema für die Dekomposition aktivieren.
5. Je nach Position des zu zerlegenden XML-Dokuments eine der angegebenen Methoden verwenden und dabei den XSR-Objektnamen für das XML-Schema angeben:
   - Bei einem einzelnen XML-Dokument im Dateisystem eine der folgenden Methoden verwenden:
     – Eine der gespeicherten XDBDECOMPXML-Prozeduren aufrufen, die der Größe des Dokuments, für das die Dekomposition durchgeführt wird, am besten entspricht.[3]

3. Wenn Sie Scripts oder Anwendungen verwenden, um die Dekomposition für eine Reihe von Dokumenten durchzuführen, deren Größe nicht bekannt ist, sollten Sie anstelle der gespeicherten Prozedur XDBDECOMPXML den Befehl DECOMPOSE XML DOCUMENT für die Dekomposition verwenden, da der Befehl automatisch die gespeicherte Prozedur aufruft, die der Größe des Dokuments entspricht.

- Den Befehl DECOMPOSE XML DOCUMENT eingeben.

- Bei einem oder mehreren Dokumenten, die in einer binären oder XML-Spalte gespeichert sind, eine der folgenden Methoden verwenden:
  - Den Befehl DECOMPOSE XML DOCUMENTS eingeben.
  - Die gespeicherte Prozedur XDB_DECOMP_XML_FROM_QUERY aufrufen.

Ergebnisse

Registrieren und Aktivieren von XML-Schemata für die Dekomposition

Sobald ein mit Annotationen versehenes Schema erfolgreich registriert und für die Dekomposition aktiviert wurde, können Sie es verwenden, um eine Dekomposition von XML-Dokumenten durchführen.

Vorbereitende Schritte

- Vergewissern Sie sich, dass mindestens eine Element- oder Attributdeklaration im XML-Schema mit einer XML-Dekompositionsannotation versehen ist. Dieses mit Annotationen versehene Element oder Attribut muss ein Nachkomme eines globalen Elements mit einem komplexen Typ oder selbst ein solches Element sein.
- Vergewissern Sie sich, dass alle Tabellen und Spalten, auf die in der Gruppe der mit Annotationen versehenen Schemadokumente verwiesen wird und aus denen das XML-Schema besteht, in der Datenbank vorhanden sind. Zwischen jeder Tabelle, auf die im Schema verwiesen wird, und dem XSR-Objekt, das diesem Schema entspricht, wird eine Abhängigkeit erstellt.
- Stellen Sie sicher, dass der Konfigurationsparameter **applheapsz** mindestens auf 1024 gesetzt ist.

Vorgehensweise

Wählen Sie eine der folgenden Methoden, um XML-Schemata für die Dekomposition zu registrieren und zu aktivieren: [4]

- Gespeicherte Prozeduren:
  1. Rufen Sie die gespeicherte Prozedur XSR_REGISTER auf, und übergeben Sie das primäre Schemadokument.
  2. Wenn das XML-Schema aus mehr als einem Schemadokument besteht, rufen Sie die gespeicherte Prozedur XSR_ADDSCHEMADOC für jedes Schemadokument auf, das noch nicht registriert ist.
  3. Rufen Sie die gespeicherte Prozedur XSR_COMPLETE mit dem Wert 1 für den Parameter **isusedfordecomposition** auf.
- Befehlszeile:
  - Besteht das XML-Schema nur aus einem Schemadokument, geben Sie den Befehl **REGISTER XML SCHEMA** mit den Optionen **COMPLETE** und **ENABLE DECOMPOSITION** ein.
  - Besteht das XML-Schema aus mehreren Schemadokumenten, gehen Sie wie folgt vor:

4. Wenn das XML-Schema zu einem früheren Zeitpunkt mithilfe einer dieser Methoden registriert, jedoch nicht für die Dekomposition aktiviert wurde, können Sie das Schema für die Dekomposition aktivieren, indem Sie die SQL-Anweisung ALTER XSROBJECT mit der Option ENABLE DECOMPOSITION ausgeben.

1. Geben Sie für jedes Schemadokument - mit Ausnahme des letzten - den Befehl **REGISTER XML SCHEMA** aus.
2. Geben Sie für das letzte Schemadokument, das noch nicht registriert ist, den Befehl **REGISTER XML SCHEMA** mit den Optionen **COMPLETE** und **ENABLE DECOMPOSITION** ein.

- JDBC-Schnittstelle:
  1. Rufen Sie die Methode DB2Connection.registerDB2XMLSchema auf und setzen Sie den Booleschen Parameter **isUsedForDecomposition** auf den Wert 'true', um die Dekomposition zu aktivieren. [5]

Nächste Schritte

Wenn ein XML-Schema für die Dekomposition aktiviert wird, wird eine Abhängigkeit zwischen jeder Tabelle, auf die im Schema verwiesen wird, und dem XSR-Objekt, das diesem Schema entspricht, erstellt. Diese Abhängigkeit verhindert, dass Tabellen, auf die im Schema verwiesen wird, umbenannt werden. Wenn eine Tabelle, auf die verwiesen wird, umbenannt werden soll, muss das XSR-Objekt für das XML-Schema für die Dekomposition inaktiviert werden. Die Tabellen, auf die XSR-Objekte verweisen, befinden Sich in der Katalogsicht SYSCAT.XSROBJECTDEP.

Beispiele für die Dekomposition (Zerlegung) mehrerer XML-Dokumente

Der Befehl DECOMPOSE XML DOCUMENTS zerlegt mithilfe eines einzigen XML-Schemas eine Gruppe von XML-Dokumenten, die in einer binären oder in einer XML-Spalte gespeichert sind. Die Daten aus den XML-Dokumenten werden auf der Grundlage der in dem Schema angegebenen Annotationen in den Spalten von relationalen Tabellen gespeichert.

Beispiele

Im folgenden Beispiel wird angenommen, dass die relationale Tabelle ABC.SALESTAB die beiden Spalten SALESDOC und DOCID enthält. Die Spalte SALESDOC enthält XML-Dokumente, während DOCID die Dokumentkennung für die in SALESDOC gespeicherten XML-Dokumente enthält. Alle Dokumente entsprechen einem XML-Schema, das im XML-Schema-Repository (XSR) als ABC.SALES registriert wurde. Das Schema wurde mit Informationen zur Dekomposition (Zerlegung) annotiert und für die Dekomposition aktiviert. Rufen Sie den folgenden Befehl DECOMPOSE XML DOCUMENTS auf, um alle in ABC.SALESTAB.SALESDOC gespeicherten Dokumente mithilfe des Schemas ABC.SALES zu zerlegen:

```
DECOMPOSE XML DOCUMENTS IN 'SELECT DOCID, SALESDOC FROM ABC.SALESTAB'
  XMLSCHEMA ABC.SALES
  MESSAGES /home/myid/errors/errorreport.xml
```

Alternativ hierzu können Sie die XML-Dokumente mit der gespeicherten Prozedur XDB_DECOMP_XML_FROM_QUERY zerlegen. Die folgende gespeicherte Prozedur bewirkte dieselbe Dekomposition wie der vorangegangene Befehl.

```
XDB_DECOMP_XML_FROM_QUERY ('ABC', 'SALES', 'SELECT DOCID, SALESDOC FROM SALESTAB',
  0, 0, 0, NULL, NULL, 1, numInput, numDecomposed,
  errorreportBuf);
```

5. Diese Methode liegt in zwei Formaten vor: Einem Format für XML-Schemadokumente, bei denen es sich um die Eingabe über ein Eingabedatenstromobjekt (`InputStream`) handelt, und einem Format für XML-Schemadokumente in einer Zeichenfolge (`String`).

Mit dem folgenden Befehl werden die Kundeninformationen in INFO mit dem Schema CUST_SHRED aus der Tabelle CUSTOMER zerlegt, wobei der Wert für CUSTID größer als 1003 ist. Bei dem Beispiel wird angenommen, dass das Schema CUST_SHRED im XSR registriert ist.

```
DECOMPOSE XML DOCUMENTS IN 'SELECT CUSTID, INFO FROM CUSTOMER WHERE CUSTID > 1003'
  XMLSCHEMA CUST_SHRED
  MESSAGES /home/myid/errors/errorreport.xml
```

Dekomposition mithilfe eines mit Annotationen versehenen XML-Schemas und rekursive XML-Dokumente

XML-Schemata mit Rekursion können im XML-Schema-Repository (XSR) registriert und für die Dekomposition aktiviert werden. Die rekursiven Beziehungen selbst können jedoch nicht als skalare Werte zerlegt und in eine Zieltabelle eingefügt werden.

Mithilfe entsprechender Schemaannotationen können die rekursiven Abschnitte gespeichert und zu einem späteren Zeitpunkt als serialisierte Markupdatei abgerufen werden.

Rekursionstypen

Ein XML-Schema wird dann als rekursiv bezeichnet, wenn die in dem Schema definierten Typen es zulassen, dass in ihren eigenen Definitionen Elemente desselben Namens und Typs vorkommen. Es gibt explizite und implizite Rekursion.

Explizite Rekursion

Eine explizite Rekursion erfolgt dann, wenn ein Element durch sich selbst definiert wird. Dies wird in folgendem Beispiel gezeigt, in dem mithilfe des Elementdeklarationsattributs ref explizit auf das Element <root> in dessen eigener Definition verwiesen wird:

```
<xs:element name="root">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="a" type="xs:string"/>
      <xs:element name="b">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="c" type="xs:string"/>
            <xs:element ref="root" minOccurs="0"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
   </xs:element>
```

Bei der expliziten Rekursion wird eine rekursive Verzweigung wie folgt begrenzt:

- Eine rekursive Verzweigung beginnt mit einer Deklaration des Elements `Y`, dessen Vorfahren nicht aus einer weiteren Elementdeklaration vom Typ `Y` bestehen. Der Beginn einer rekursiven Verzweigung kann mehrere Verzweigungen von Nachkommen haben. Weist die Verzweigung eines Nachkommens eine weitere Deklaration von `Y` auf, gilt diese Verzweigung als rekursive Verzweigung.

- Eine rekursive Verzweigung wird mit der Deklaration des Elements Y der höchsten Ebene beendet, die ein Nachkomme des Beginns der Verzweigung ist. Bitte beachten Sie, dass das Ende einer Verzweigung insbesondere ein Elementverweis ist.

Der Knoten, der den Beginn einer rekursiven Verzweigung darstellt, kann als Startknoten für mehrere rekursive Verzweigungen dienen. Das folgende Beispiel enthält zwei explizite rekursive Verzweigungen:

1. <root> (*), <b>, <root> (**)
2. <root> (*), <b>, <root> (***)

```
<xs:element name="root"> <!-- * -->
  <xs:complexType>
  <xs:sequence>
    <xs:element name="a" type="xs:string"/>
    <xs:element name="b">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="c" type="xs:string"/>
        <xs:element ref="root" minOccurs="1"/>  <!-- ** -->
        <xs:element ref="root" minOccurs="1"/>  <!-- *** -->
      </xs:sequence>
    </xs:complexType>
    </xs:element>
  </xs:sequence>
  </xs:complexType>
   </xs:element>
```

Eine rekursive Verzweigung beschreibt, wie ihre Memberelemente zerlegt werden. Im Instanzdokument können das Element Y, das dem Beginn der rekursiven Verzweigung entspricht, und dessen Nachkommen bis zum Element Y, das dem Ende der Verzweigung entspricht, als skalare Werte zerlegt werden. Das Element Y, das im Instanzdokument dem Ende der rekursiven Verzweigung entspricht, markiert den rekursiven Bereich. Der rekursive Bereich beginnt mit der Anfangselementmarkierung dieses Vorkommens von Y und endet mit der Endelementmarkierung dieses Vorkommens. Alle Elemente und Attribute im Instanzdokument, die sich in diesem rekursiven Bereich befinden, können als Markupdatei oder als Zeichenfolgewerte zerlegt werden, je nachdem, welcher Wert für die Dekompositionsannotation db2-xdb:contentHandling angegeben ist.

Implizite Rekursion

Eine implizite Rekursion erfolgt, wenn ein Element, das als komplexer Typ (complexType) definiert ist, ein weiteres Element enthält, das auch als komplexer Typ definiert ist, wobei das letztere Element als Typattribut den Namen einer komplexen Typdefinition aufweist, zu der es selbst gehört. Dies wird in folgendem Beispiel gezeigt, in dem das Element <beginRecursion> auf den Typ "rootType" verweist und das Element <beginRecursion> selbst Teil des definierten Typs "rootType" ist:

```
<xs:element name="root" type="rootType"/>
<xs:complexType name="rootType">
  <xs:sequence>
  <xs:element name="a" type="xs:string"/>
    <xs:element name="b">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="c" type="xs:string"/>
          <xs:element name="beginRecursion" type="rootType" minOccurs="0"/>
        </xs:sequence>
```

```
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
```

Bei der impliziten Rekursion wird eine rekursive Verzweigung wie folgt begrenzt:

- Eine rekursive Verzweigung beginnt mit einer Deklaration des Elements `Y` vom Typ `CT` (complexType), dessen Vorfahren nicht aus einer weiteren Elementdeklaration vom Typ `CT` bestehen. Der Beginn einer rekursiven Verzweigung kann mehrere Verzweigungen von Nachkommen haben. Weist die Verzweigung eines Nachkommens eine weitere Elementdeklaration (`Z`) vom Typ `CT` auf, gilt diese Verzweigung als rekursive Verzweigung.
- Eine rekursive Verzweigung wird mit der Deklaration des Elements vom Typ `CT` der höchsten Ebene beendet, die ein Nachkomme des Beginns der Verzweigung ist.

Der Knoten, der den Beginn einer rekursiven Verzweigung darstellt, kann als Startknoten für mehrere rekursive Verzweigungen dienen. Das folgende Beispiel enthält zwei implizite rekursive Verzweigungen:

1. `<root>, <b>, <beginRecursion>`
2. `<root>, <b>, <anotherRecursion>`

```
<xs:element name="root" type="rootType"/>
<xs:complexType name="rootType">
  <xs:sequence>
  <xs:element name="a" type="xs:string"/>
  <xs:element name="b">
    <xs:complexType>
    <xs:sequence>
      <xs:element name="c" type="xs:string"/>
      <xs:element name="beginRecursion" type="rootType" minOccurs="2"/>
      <xs:element name="anotherRecursion" type="rootType" minOccurs="0"/>
    </xs:sequence>
    </xs:complexType>
  </xs:element>
  </xs:sequence>
</xs:complexType>
```

Die Zerlegung bei der impliziten Rekursion weist geringfügige Unterschiede zur Zerlegung bei der expliziten Rekursion auf. Im Instanzdokument können das Element `Y`, das dem Beginn der rekursiven Verzweigung entspricht, und dessen Nachkommen bis zum Element `Z`, das dem Ende der Verzweigung entspricht, als skalare Werte zerlegt werden. Dieses Vorkommen von `Z` im Instanzdokument markiert den rekursiven Bereich. Der rekursive Bereich beginnt *nach* der Anfangselementmarkierung von `Z` und endet unmittelbar *vor* der Endelementmarkierung von `Z`. Alle Elementnachkommen dieses Vorkommens von `Z` liegen in diesem rekursiven Bereich. Die Attribute dieses Vorkommens befinden sich jedoch außerhalb des rekursiven Bereichs und können daher als skalare Werte zerlegt werden.

Dekompositionsverhalten rekursiver Verzweigungen

Bei beiden Rekursionstypen beschreibt die rekursive Verzweigung die nicht rekursiven und rekursiven Bereiche im entsprechenden Teil des Instanzdokuments. Nur die nicht rekursiven Bereiche eines XML-Instanzdokuments können als skalare Werte zerlegt und in eine Zieldatenbanktabelle eingefügt werden. Diese Einschränkung gilt auch für alle nicht rekursiven Bereiche innerhalb einer Verzweigung, die zu einem rekursiven Bereich einer einschließenden Verzweigung gehören. Daher

gilt: Wenn die rekursive Verzweigung RB2 vollständig von der rekursiven Verzweigung RB1 umfasst wird, dann kann bei einigen Instanzen von RB2 im XML-Instanzdokument deren nicht rekursiver Bereich innerhalb des rekursiven Bereichs einer Instanz von RB1 liegen. In diesem Fall kann dieser nicht rekursive Bereich nicht als skalare Werte zerlegt werden. Stattdessen ist er Teil der Markupdatei, die das Dekompositionsergebnis für RB1 darstellt. Bei allen Instanzen von RB2 kann nur der nicht rekursive Bereich der Instanz, der sich nicht innerhalb anderer rekursiver Bereiche befindet, als skalare Werte zerlegt werden.

Das folgende XML-Schema beispielsweise enthält zwei rekursive Verzweigungen:

1. RB1 (<root> (durch * gekennzeichnet), <b>, <root> (durch ** gekennzeichnet))
2. RB2 (<d>, <d>)

```
<xs:element name="d">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="d">
    </xs:sequence>
    <xs:attribute name="id" type="xs:int"/>
  </xs:complexType>
</xs:element>
<xs:element name="root"> <!-- * -->
  <xs:complexType>
  <xs:sequence>
    <xs:element name="a" type="xs:string"/>
       <xs:element ref="d"/>
    <xs:element name="b">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="c" type="xs:string"/>
        <xs:element ref="root" minOccurs="1"/>  <!-- ** -->
      </xs:sequence>
    </xs:complexType>
    </xs:element>
  </xs:sequence>
  </xs:complexType>
   </xs:element>
```

Die rekursiven Bereiche eines zugeordneten Instanzdokuments werden im folgenden Beispiel hervorgehoben. Das Instanzdokument enthält zwei Instanzen von RB2 (<d>, <d>), aber nur der nicht rekursive Bereich der ersten Instanz von RB2 (<d>, durch # gekennzeichnet) kann als skalare Werte zerlegt werden. Dies bedeutet, dass das Attribut id="1" zerlegt werden kann. Der nicht rekursive Bereich der zweiten Instanz von RB2 befindet sich gänzlich innerhalb des zweiten hervorgehobenen Bereichs, bei dem es sich um einen rekursiven Bereich der Instanz von RB1 handelt. Daher kann das Attribut id="2" nicht zerlegt werden.

```
<root>
  <a>a str1</a>
  <d id="1"> <d id="11"> </d> </d>
  <b>
    <c>c str1</c>
    <root>
      <a>a str11</a>
      <d id="2"> <d id="22"> </d> </d>
      <b>
      <c>c str11</c>
      </b>
    </root>
  </b>
</root>
```

Beispiel: Verwendung der Dekompositionsannotation 'db2-xdb-:contentHandling' mit beiden Rekursionstypen

Dieses Beispiel zeigt das Dekompositionsverhalten sowohl der expliziten als auch der impliziten Rekursion sowie die Ergebnisse bei Einstellung unterschiedlicher Werte für die Annotation db2-xdb:contentHandling. In den folgenden beiden XML-Instanzdokumenten werden die rekursiven Bereiche hervorgehoben.

In Dokument 1 beginnt die Rekursion an der Stelle, an der das Element <root> unter sich selbst angezeigt wird:

```
<root>
  <a>a str1</a>
  <b>
    <c>c str1</c>
    <root>
      <a>a str11</a>
      <b>
        <c>c str11</c>
      </b>
    </root>
  </b>
</root>
```

In Dokument 2 beginnt die Rekursion für Elemente unterhalb des Elements <beginRecursion>:

```
<root>
  <a>a str2</a>
  <b>
    <c>c str2</c>
    <beginRecursion>
      <a>a str22</a>
      <b>
        <c>c str22</c>
      </b>
    </beginRecursion>
  </b>
</root>
```

In einem Instanzdokument können die Elemente oder Attribute sowie deren Inhalte, die zwischen dem Anfang und dem Ende der Rekursion angezeigt werden, nicht als skalare Werte zerlegt und in Tabellen/Spalten eingefügt werden. Es kann jedoch die Version einer Markupdatei mit den Elementen zwischen dem Anfang und dem Ende der Rekursion abgerufen werden, indem ein Element (vom Typ 'complexType') in der rekursiven Verzweigung mit dem Attribut db2-xdb:contentHandling annotiert wird, das die Einstellung "serializeSubtree" aufweist. Eine Textserialisierung aller Zeichendaten in diesem Teil kann ebenfalls abgerufen werden, indem 'db2-xdb:contentHandling' auf "stringValue" gesetzt wird. Im Allgemeinen gilt, dass der Inhalt oder die Markupdatei des rekursiven Pfads abgerufen werden kann, indem das Attribut db2-xdb:contentHandling für ein beliebiges Element vom Typ 'complexType' der rekursiven Verzweigung oder für ein Element, das ein Vorfahre der Elemente in der rekursiven Verzweigung ist, entsprechend gesetzt wird.

Beispiel: Nehmen Sie an, das Element <b> in folgendem XML-Schema wird mit einer Annotation versehen:

```
<xs:element name="root">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="a" type="xs:string"/>
      <xs:element name="b"
```

```
            db2-xdb:rowSet="TABLEx"
            db2-xdb:column="COLx"
            db2-xdb:contentHandling="serializeSubtree">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="c" type="xs:string"/>
            <xs:element ref="root" minOccurs="0"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
   </xs:element>
```

Das Ergebnis ist die Einfügung des folgenden XML-Fragments in eine Zeile von Tabelle X (TABLEx), Spalte X (COLx) bei der Dekomposition von Dokument 1:

```
<b>
  <c>c str1</c>
  <root>
    <a>a str11</a>
    <b>
      <c>c str11</c>
    </b>
  </root>
</b>
```

Ähnlich ist die Annotierung des Elements "beginRecursion" in folgendem XML-Schema:

```
<xs:element name="root" type="rootType"/>
<xs:complexType name="rootType">
  <xs:sequence>
    <xs:element name="a" type="xs:string"/>
    <xs:element name="b">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="c" type="xs:string"/>
          <xs:element name="beginRecursion"
            type="rootType" minOccurs="0"
            db2-xdb:rowSet="TABLEx"
            db2-xdb:column="COLx"
            db2-xdb:contentHandling="serializeSubtree"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
```

Das Ergebnis hierbei ist die Einfügung des folgenden XML-Fragments in eine Zeile von Tabelle X (TABLEx), Spalte X (COLx) bei der Dekomposition von Dokument 2:

```
    <beginRecursion>
      <a>a str22</a>
      <b>
        <c>c str22</c>
      </b>
    </beginRecursion>
```

Inaktivierung der Dekomposition mithilfe eines mit Annotationen versehenen XML-Schemas

Die Dekomposition mithilfe eines mit Annotationen versehenen XML-Schemas kann von DB2 unter bestimmten Bedingungen unausführbar gemacht oder vom Benutzer explizit inaktiviert werden.

Bedingungen, die eine Dekomposition unausführbar machen

Die schemabasierte Dekomposition wird bei mit Annotationen versehenen Schemata, die zuvor registriert und für die Dekomposition aktiviert wurden, automatisch unausführbar gemacht, wenn mindestens eine der folgenden Bedingungen erfüllt ist. (Ein XML-Schema, das für die Dekomposition unausführbar gemacht wird, kann dennoch für die Prüfung verwendet werden, die außerhalb des Kontexts der Dekomposition erfolgt, z. B. mit der SQL/XML-Funktion XMLVALIDATE.) Die Korrekturmaßnahmen, die zur Reaktivierung der Dekomposition erforderlich sind, sind für die einzelnen Bedingungen aufgeführt.

Tabelle 41. Bedingungen, die eine Dekomposition unausführbar machen, und entsprechende Korrekturen

Bedingung	**Aktion zur Reaktivierung der Dekomposition**
Die Tabelle, auf die in der Annotation verwiesen wird, wurde gelöscht.	Entfernen Sie den Verweis auf die gelöschte Tabelle aus dem Schemadokument, registrieren Sie das gesamte, mit Annotationen versehene Schema erneut, und aktivieren Sie das Schema für die Dekomposition.
Für eine Spalte, auf die in der Annotation verwiesen wird, wurde der Datentyp in einen Typ geändert, der mit dem XML-Schematyp kompatibel ist.	Reaktivieren Sie das Schema für die Dekomposition, indem Sie die SQL-Anweisung ALTER XSROBJECT mit der Option ENABLE DECOMPOSITION ausführen.
Für eine Spalte, auf die in der Annotation verwiesen wird, wurde der Datentyp in einen Typ geändert, der mit dem XML-Schematyp nicht kompatibel ist.	Passen Sie die Annotationen wie erforderlich an, registrieren Sie das gesamte, mit Annotationen versehene Schema erneut, und aktivieren Sie das Schema für die Dekomposition.
Ein Dokument, das dem mit Annotationen versehenen Schema zugeordnet ist, wurde geändert.	Registrieren Sie alle Dokumente, aus denen das betreffende Schema besteht, erneut, und aktivieren Sie das Schema für die Dekomposition.

Weitere Informationen finden Sie in der aufgabenorientierten Dokumentation zur Registrierung von mit Annotationen versehenen Schemata und zur Aktivierung der Dekomposition.

Explizite Inaktivierung

Sie können die schemabasierte Dekomposition explizit inaktivieren, indem Sie eine der folgenden SQL-Anweisungen ausführen und dabei das XSR-Objekt angeben, das dem mit Annotationen versehenen Schema entspricht, das Sie inaktivieren möchten:

- ALTER XSROBJECT mit der Option DISABLE DECOMPOSITION

  Anmerkung: Ein XML-Schema, das für die Dekomposition inaktiviert ist, kann dennoch zur Prüfung verwendet werden.
- DROP mit der Option XSROBJECT

Anmerkung: Die auszuwählende Option ist vom Verwendungszweck des XML-Schemas abhängig. Wenn das Schema für die Prüfung benötigt wird, sollte es für die Dekomposition inaktiviert, nicht gelöscht werden. Wenn das Schema nur für die Dekomposition verwendet und voraussichtlich nicht mehr benötigt wird, können Sie das XSR-Objekt löschen.

xdbDecompXML-Prozeduren für die Dekomposition mithilfe eines mit Annotationen versehenen Schemas

Die Dekomposition mithilfe eines mit Annotationen versehenen XML-Schemas kann durch Aufrufen einer von zehn integrierten Prozeduren durchgeführt werden, die zur Zerlegung eines einzelnen XML-Dokuments dienen.

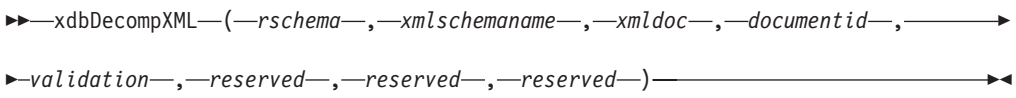
Prozeduren zur Dekomposition mithilfe eines mit Annotationen versehenen XML-Schemas:

- xdbDecompXML
- xdbDecompXML10MB
- xdbDecompXML25MB
- xdbDecompXML50MB
- xdbDecompXML75MB
- xdbDecompXML100MB
- xdbDecompXML500MB
- xdbDecompXML1GB
- xdbDecompXML1_5GB
- xdbDecompXML2GB

Diese Prozeduren unterscheiden sich nur in der Größe des Arguments *xmldoc*, das die Größe des Eingabedokuments für die Dekomposition angibt. Rufen Sie die Prozedur auf, deren Größe gerade für die Größe des Dokuments ausreicht, für das Sie die Dekomposition ausführen möchten, und begrenzen Sie so die Systemspeicherbelastung auf ein Minimum. Verwenden Sie beispielsweise die Prozedur xdbDecompXML, um die Dekomposition eines Dokuments mit einer Größe von 1 MB durchzuführen.

Die Syntax für xdbDecompXML ist im folgenden Abschnitt dargestellt. Die Beschreibung zum Argument *xmldoc* enthält die Spezifikationen des Arguments *xmldoc* für die Prozeduren xdbDecompXML10MB, xdbDecompXML25MB, xdbDecompXML50MB, xdbDecompXML75MB, xdbDecompXML100MB, xdbDecompXML500MB, xdbDecompXML1GB, xdbDecompXML1_5GB und xdbDecompXML2GB.

Syntax



Das Schema der Prozedur ist SYSPROC.

Berechtigung

Die Berechtigungs-ID der Anweisung, die diese Prozedur aufruft, muss über eine der folgenden Berechtigungen oder eines der folgenden Zugriffsrechte verfügen:

- Alle folgenden Zugriffsrechte:
  - Zugriffsrecht INSERT für alle Zieltabellen, auf die im mit Annotationen versehenen Schema verwiesen wird

  - Zugriffsrecht SELECT, INSERT, UPDATE oder DELETE (je nach Gültigkeit) für die jeweilige Tabelle, auf die durch die Annotation db2-xdb:expression bzw. db2-xdb:condition verwiesen wird
- Zugriffsrecht CONTROL für alle Zieltabellen, auf die in der Gruppe der mit Annotationen versehenen Schemadokumente verwiesen wird
- Berechtigung DATAACCESS

Wenn *validation* den Wert '1' hat, muss die Berechtigungs-ID der Anweisung, die diese Prozedur aufruft, das Zugriffsrecht USAGE für das XML-Schema besitzen.

Prozedurparameter

rschema
Ein Eingabeargument des Typs VARCHAR(128), das den SQL-Schemateil des zweiteiligen XSR-Objektnamens angibt, der im XML-Schema-Repository registriert ist. Ist dieser Wert `NULL`, wird davon ausgegangen, dass der SQL-Schemateil der aktuelle Wert des Sonderregisters CURRENT SCHEMA ist.

xmlschemaname
Ein Eingabeargument des Typs VARCHAR(128), das den Schemanamen des zweiteiligen XSR-Objektnamens angibt, der im XML-Schema-Repository registriert ist. Dieser Wert darf nicht `NULL` sein.

xmldoc
Ein Eingabeargument des Typs BLOB(1M), das den Puffer angibt, der das XML-Dokument für die Dekomposition enthält.

Anmerkung:

- Für die Prozedur xdbDecompXML10MB hat dieses Argument den Typ BLOB(10M).
- Für die Prozedur xdbDecompXML25MB hat dieses Argument den Typ BLOB(25M).
- Für die Prozedur xdbDecompXML50MB hat dieses Argument den Typ BLOB(50M).
- Für die Prozedur xdbDecompXML75MB hat dieses Argument den Typ BLOB(75M).
- Für die Prozedur xdbDecompXML100MB hat dieses Argument den Typ BLOB(100M).
- Für die Prozedur xdbDecompXML500MB hat dieses Argument den Typ BLOB(500M).
- Für die Prozedur xdbDecompXML1GB hat dieses Argument den Typ BLOB(1G).
- Für die Prozedur xdbDecompXML1_5GB hat dieses Argument den Typ BLOB(1.5G).
- Für die Prozedur xdbDecompXML2GB hat dieses Argument den Typ BLOB(2G).

documentid
Ein Eingabeargument des Typs VARCHAR(1024), das die ID für das XML-Eingabedokument angibt, für das die Dekomposition ausgeführt werden soll. Der in diesem Argument angegebene Wert wird für jede Angabe von $DECOMP-_DOCUMENTID in der Annotation db2-xdb:expression oder db2-xdb:condition im entsprechenden XML-Schema eingesetzt.

validation
Ein Eingabeargument des Typs INTEGER, das angibt, ob für das Dokument vor der Dekomposition eine Prüfung durchgeführt werden soll. Folgende Werte sind möglich:

0 Es wird vor der Dekomposition keine Prüfung für das Eingabedokument durchgeführt.

1 Für das Eingabedokument wird eine Prüfung auf der Basis von DTDs oder XML-Schemadokumenten, die zu einem früheren Zeitpunkt im XML-Schema-Repository registriert wurden, durchgeführt. Für das XML-Eingabedokument wird nur dann eine Dekomposition durchgeführt, wenn die Prüfung erfolgreich verläuft.

reserved
Die Argumente *reserved* sind Eingabeargumente, die für die zukünftige Verwendung reserviert sind. Die für diese Argumente übergebenen Werte müssen `NULL` sein.

Ausgabe

Für diese Prozedur ist kein explizites Ausgabeargument vorhanden. Prüfen Sie das SQLCODE-Feld der SQLCA-Struktur auf Fehler, die möglicherweise während der Dekomposition aufgetreten sind. Die möglichen SQLCODE-Werte nach Abschluss der Dekomposition sind:

0 Die Dekomposition des Dokuments wurde erfolgreich durchgeführt.

Positive ganze Zahl
Die Dekomposition des Dokuments wurde erfolgreich durchgeführt, es traten jedoch Warnungsbedingungen auf. Die Warnungen sind in den Protokolldateien mit der Bezeichnung **db2diag** protokolliert, die sich im FODC-Speicherverzeichnis (FODC = First Occurence Data Capture; Datenerfassung beim ersten Vorkommen) befinden.

Negative ganze Zahl
Die Dekomposition des Dokuments war nicht möglich. Der SQLCODE-Wert gibt die Fehlerursache an. Detaillierte Informationen zum Fehler finden Sie in den **db2diag**-Protokolldateien.

Hinweise

Die Prozeduren werden mit der Isolationsstufe 'Lesestabilität' ausgeführt.

Die Prozeduren werden automatisch ausgeführt. Schlägt eine Prozedur während der Ausführung fehl, werden alle von ihr vorgenommenen Änderungen zurückgesetzt. Für das Commit der von der Prozedur vorgenommenen Änderungen muss die aufrufende Funktion die SQL-Anweisung COMMIT ausführen, da die Prozedur selbst kein Commit ausführt.

Wenn Sie Scripts oder Anwendungen verwenden, um die Dekomposition für eine Reihe von Dokumenten durchzuführen, deren Größe nicht bekannt ist, sollten Sie anstelle einer Prozedur xdbDecompXML den Befehl **DECOMPOSE XML DOCUMENT** für die Dekomposition verwenden, da der Befehl automatisch die richtige Prozedur für die Größe des Dokuments aufruft.

Beispiel

Das mit Annotationen versehene XML-Schema, das die Tabelle und die Spalte angibt, in die die XML-Werte zerlegt werden, wurde im XML-Schema-Repository als ABC.TEST registriert. Zur Dekomposition des XML-Dokuments in eine relationale Tabelle rufen Sie die entsprechende Prozedur wie folgt auf:

```
CALL xdbDecompXML ('ABC', 'TEST', BLOB('<Element1>Hello world</Element1>'),
     'DOCID', 0 , NULL, NULL, NULL)
```

DECOMPOSE XML DOCUMENT

Dieser Befehl ruft eine gespeicherte Prozedur auf, um mithilfe eines dekompositionsfähigen XML-Schemas die Dekomposition eines einzelnen XML-Dokuments durchzuführen.

Berechtigung

Dazu ist eine der folgenden Gruppen von Zugriffsrechten oder Berechtigungen erforderlich:

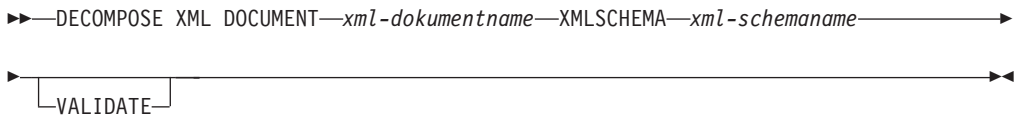
- Eine der folgenden Berechtigungen:
  - Zugriffsrecht CONTROL für alle Zieltabellen, auf die in der Gruppe der mit Annotationen versehenen Schemadokumente verwiesen wird
  - Berechtigung DATAACCESS
- Alle folgenden Zugriffsrechte:
  - Zugriffsrecht INSERT für die Zieltabelle, wie für die in der Aktionsdatei angegebene Operation erforderlich
  - Zugriffsrecht SELECT, INSERT, UPDATE oder DELETE (je nach Gültigkeit) für die jeweilige Tabelle, auf die durch die Annotation db2-xdb:expression bzw. db2-xdb:condition verwiesen wird

Wenn die Option **VALIDATE** angegeben ist, ist auch das Zugriffsrecht USAGE für das XML-Schema erforderlich.

Erforderliche Verbindung

Datenbank

Befehlssyntax



Befehlsparameter

DECOMPOSE XML DOCUMENT *xml-dokumentname*
xml-dokumentname ist der Dateipfad und -name des XML-Eingabedokuments, für das eine Dekomposition durchgeführt werden soll.

XMLSCHEMA *xml-schemaname*
xml-schemaname ist der Name eines vorhandenen XML-Schemas, das bei dem XML-Schema-Repository registriert ist, mit dem die Dekomposition des Dokuments durchgeführt werden soll. *xml-schemaname* ist eine qualifizierte SQL-Kennung, die aus einem optionalen SQL-Schemanamen, einem

nachfolgenden Punkt und dem XML-Schemanamen besteht. Wird der SQL-Schemaname nicht angegeben, wird angenommen, dass er den Wert des DB2-Sonderregisters CURRENT SCHEMA hat.

VALIDATE
Dieser Parameter gibt an, dass das XML-Eingabedokument zunächst geprüft und nur dann eine Dekomposition durchgeführt werden soll, wenn es sich um ein gültiges Dokument handelt. Wird **VALIDATE** nicht angegeben, wird das XML-Eingabedokument vor der Dekomposition nicht geprüft.

Beispiele

Im folgenden Beispiel wird angegeben, dass das XML-Dokument `./gb/document1.xml` mit dem registrierten XML-Schema DB2INST1.GENBANKSCHEMA geprüft und zerlegt werden soll.

```
DECOMPOSE XML DOCUMENT ./gb/document1.xml
              XMLSCHEMA DB2INST1.GENBANKSCHEMA
              VALIDATE
```

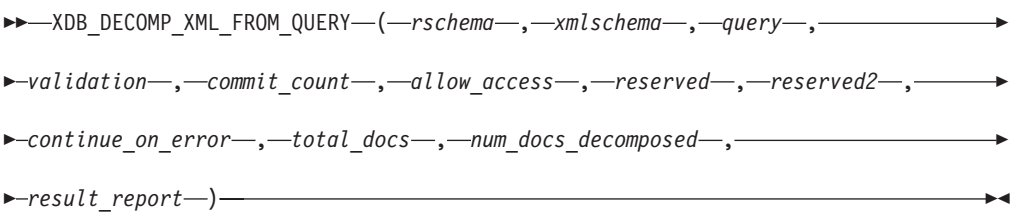
Im folgenden Beispiel wird angegeben, dass die Dekomposition des XML-Dokuments `./gb/document2.xml` ohne Prüfung mit dem registrierten XML-Schema DB2INST2."GENBANK SCHEMA1" durchgeführt werden soll, unter der Annahme, dass der Wert des DB2-Sonderregisters CURRENT SCHEMA auf DB2INST2 festgelegt ist.

```
DECOMPOSE XML DOCUMENT ./gb/document2.xml
              XMLSCHEMA "GENBANK SCHEMA1"
```

Gespeicherte Prozedur XDB_DECOMP_XML_FROM_QUERY für die Dekomposition mithilfe eines mit Annotationen versehenen Schemas

Die gespeicherte Prozedur zerlegt ein oder mehrere XML-Dokumente aus einer binären oder XML-Spalte. Die Daten aus den XML-Dokumenten werden auf der Grundlage der in einem XML-Schema angegebenen Annotationen in den Spalten von relationalen Tabellen gespeichert.

Syntax



Das Schema der gespeicherten Prozedur ist SYSPROC.

Die Prozedur wird mit der Isolationsstufe 'Lesestabilität' ausgeführt.

Berechtigung

Dazu ist eine der folgenden Berechtigungen oder eines der folgenden Zugriffsrechte erforderlich:

- Alle folgenden Zugriffsrechte:

  - Zugriffsrecht INSERT für alle Zieltabellen, auf die im mit Annotationen versehenen Schema verwiesen wird
  - Zugriffsrecht SELECT für die Tabelle, den Aliasnamen oder die Sicht, die bzw. der die Spalte mit den Eingabedokumenten enthält
  - Zugriffsrecht SELECT, INSERT, UPDATE oder DELETE (je nach Gültigkeit) für die jeweilige Tabelle, auf die durch die Annotation db2-xdb:expression bzw. db2-xdb:condition verwiesen wird
- Zugriffsrecht CONTROL für alle Tabellen, auf die in der Gruppe der mit Annotationen versehenen Schemadokumente verwiesen wird, und für die Tabelle, den Aliasnamen oder die Sicht, die bzw. der die Spalte mit den Eingabedokumenten enthält
- Berechtigung DATAACCESS

Wenn *validation* den Wert '1' hat, ist auch das Zugriffsrecht USAGE für das XML-Schema erforderlich.

Prozedurparameter

rschema
: Ein Eingabeargument des Typs VARCHAR(128), das den SQL-Schemateil des zweiteiligen XSR-Objektnamens angibt, der im XML-Schema-Repository (XSR) registriert ist. Der Wert kann NULL sein. Ist der Wert NULL, wird davon ausgegangen, dass der SQL-Schemateil der aktuelle Wert des Sonderregisters CURRENT SCHEMA ist.

xmlschema
: Ein Eingabeargument des Typs VARCHAR(128), das den Namen des zweiteiligen XSR-Objektnamens angibt, der im XSR registriert ist. Dieser Wert darf nicht NULL sein.

query
: Ein Eingabeargument des Typs CLOB(1MB). Dieser Wert darf nicht NULL sein. *query* entspricht den Regeln einer SQL-Anweisung SELECT und muss eine Ergebnismenge mit zwei Spalten zurückgeben. Die erste Spalte ist die Dokumentkennung. Eine Dokumentkennung identifiziert ein zu zerlegendes XML-Dokument eindeutig. Die Spalte muss einen Zeichentyp aufweisen oder in einen Zeichentyp umsetzbar sein. Die zweite Spalte enthält die XML-Dokuments, die zerlegt werden sollen. Für die Dokumentspalte werden folgende Typen unterstützt: XML, BLOB, VARCHAR FOR BIT DATA und LONG VARCHAR FOR BIT DATA. Die Spalte, die die XML-Dokumente enthält, muss in eine Spalte einer zugrunde liegenden Basistabelle aufgelöste werden, bei der Spalte darf es sich nicht um eine generierte Spalte handeln.

  Beispiel: Die Spalte DOCID in der folgenden Anweisung SELECT enthält die eindeutigen Kennungen für die in der Spalte SALESDOC gespeicherten XML-Dokumente.

  ```
  SELECT DOCID, SALESDOC FROM SALESTAB
  ```

validation
: Ein Eingabeargument des Typs INTEGER, das angibt, ob für die Dokumente vor der Dekomposition eine Prüfung durchgeführt werden soll. Folgende Werte sind möglich:

  0 Es wird vor der Dekomposition keine Prüfung für die Eingabedokumente durchgeführt.

  Wenn der Wert '0' übergeben und keine Prüfung durchgeführt wird, ist der Benutzer dafür verantwortlich, die Dokumente vor dem Aufruf der gespei-

cherten Prozedur zu prüfen. Der Benutzer kann beispielsweise XMLVALIDATE verwenden, wenn er die XML-Dokumente in die Spalte einfügt, oder einen XML-Prozessor, bevor die Dokumente eingefügt werden. Wenn ein XML-Eingabedokument nicht gültig ist und '0' für diesen Parameter angegeben wird, sind die Ergebnisse der Dekomposition nicht definiert.

1 Für die Eingabedokumente wird eine Prüfung auf der Basis von DTDs oder XML-Schemadokumenten, die zu einem früheren Zeitpunkt im XML-Schema-Repository registriert wurden, durchgeführt. Für die XML-Eingabedokumente wird nur dann eine Dekomposition durchgeführt, wenn die Prüfung erfolgreich verläuft.

commit_count
Ein Eingabeargument des Typs INTEGER. Folgende Werte sind möglich:

0 Die gespeicherte Prozedur führt nie ein COMMIT aus.

n, **eine positive ganze Zahl**
Nach jeweils *n* erfolgreichen Dokumentdekompositionen wird ein COMMIT ausgeführt.

allow_access
Ein Eingabeargument des Typs INTEGER. Folgende Werte sind möglich:

0 Die gespeicherte Prozedur fordert eine exklusive Sperre (X) für alle Tabellen mit Zuordnungen im XML-Schema an. Es werden nicht alle Tabellen zwangsläufig in die Dekomposition der einzelnen Dokumente einbezogen. Dennoch werden alle Tabellen gesperrt, um die Wahrscheinlichkeit von Deadlocks während einer langen Arbeitseinheit zu verringern.

1 Während der Anforderung der Sperre wartet die gespeicherte Prozedur und überschreitet unter Umständen das Zeitlimit.

reserved
Das Argument *reserved* ist ein Eingabeargument, das für die zukünftige Verwendung reserviert ist. Der für dieses Argument übergebene Wert muss NULL sein.

reserved2
Das Argument *reserved2* ist ein Eingabeargument, das für die zukünftige Verwendung reserviert ist. Der für dieses Argument übergebene Wert muss NULL sein.

continue_on_error
Ein Eingabeargument des Typs INTEGER. Folgende Werte sind möglich:

0 Die gespeicherte Prozedur stoppt beim ersten Dokument, das nicht erfolgreich zerlegt werden kann. Wenn bei der Dekomposition eines Dokuments ein Fehler auftritt, werden die Datenbankänderungen, die während der Dekomposition des Dokuments vorgenommen werden, rückgängig gemacht.

1 Die gespeicherte Prozedur stoppt bei dokumentspezifischen Fehlern nicht und versucht, alle mit *query* angegebenen Dokumente zu zerlegen. Wenn bei der Dekomposition eines Dokuments ein Fehler auftritt, werden die Datenbankänderungen, die während der Dekomposition des Dokuments vorgenommen werden, rückgängig gemacht. Anschließend versucht die gespeicherte Prozedur, das nächste Dokument zu zerlegen. In *result_report* werden die Informationen zu den nicht erfolgreich zerlegten Dokumenten protokolliert.

Ungeachtet des Werts für *continue_on_error* wird die gespeicherte Prozedur bei schwerwiegenden und nicht dokumentspezifischen Fehlern nicht fortgesetzt.

total_docs
Ein Ausgabeargument des Typs INTEGER, das die Gesamtzahl der Eingabedokumente angibt, die die gespeicherte Prozedur XDB_DECOMP_XML_FROM-_QUERY zu zerlegen versuchte.

num_docs_decomposed
Ein Ausgabeargument des Typs INTEGER, das die Anzahl der erfolgreich zerlegten Dokumente angibt.

result_report
Ein Ausgabeargument des Typs BLOB(100MB). Ein Puffer, der ein XML-Dokument im UTF-8-Format enthält, das die Namen der nicht erfolgreich zerlegten Eingabedateien mit der jeweiligen Diagnosenachricht auflistet. Dieser Bericht wird nur generiert, wenn mindestens ein XML-Dokument nicht erfolgreich zerlegt werden konnte.

Das XML-Dokument in *result_report* hat folgendes Format:

```
<?xml version='1.0'?>
<xdb:errorReport xmlns:xdb="http://www.ibm.com/xmlns/prod/db2/xdb1">
   <xdb:document>
      <xdb:documentId>sssss</xdb:documentId>
      <xdb:errorMsg>qqqqq</xdb:errorMsg>
   </xdb:document>
   <xdb:document>
      . . .
   </xdb:document>
   . . .
</xdb:errorReport>
```

Der documentId-Wert *sssss* ist der mit *query* angegebene Wert aus der ersten Spalte. Dieser Wert gibt das XML-Dokument an, das nicht erfolgreich zerlegt wurde. Der errorMsg-Wert *qqqqq* ist der Fehler, der bei dem Versuch auftrat, das Dokument zu zerlegen.

Ausgabe

Die SQLCA-Struktur gibt den Rückgabestatus der Prozedur nach dem Versuch zu Dekomposition der XML-Dokumente an. Die Prozedur kann einen der folgenden SQLCODE-Werte zurückgeben:

0 Alle mit *query* angegebenen Dokumente wurden erfolgreich zerlegt.

16278
Die Dekomposition eines oder mehrerer Dokumente ist fehlgeschlagen. Die Anzahl der erfolgreich zerlegten Dokumente wird als Ausgabeparameter *num-_docs_decomposed* an die gespeicherte Prozedur übergeben. Die jeweiligen Fehlernachrichten für die einzelnen fehlgeschlagenen Dokumente werden in *result-_report* dokumentiert. Weitere Details zur Diagnose der einzelnen Fehler werden in der Protokolldatei **db2diag** protokolliert.

Negative ganze Zahl
Es wurden keine Dokumente zerlegt. Der SQLCODE-Wert gibt die Fehlerursache an. Detaillierte Informationen zum Fehler finden Sie in der Protokolldatei **db2diag**.

Hinweis

Die gespeicherte Prozedur wird mit den folgenden Merkmalen deklariert:

```
DYNAMIC RESULT SETS 0
NOT DETERMINISTIC
UNFENCED
```

```
THREADSAFE
MODIFIES SQL DATA
PARAMETERSTYLE SQL
CALLED ON NULL INPUT
NEW SAVEPOINT LEVEL
DBINFO
```

Beispiel

Im folgenden Beispiel wird angenommen, dass die Tabelle ABC.SALESTAB die beiden Spalten SALESDOC und DOCID enthält. Die Spalte SALESDOC enthält XML-Dokumente, während DOCID die eindeutige Kennung für die in SALESDOC enthaltenen XML-Dokumente enthält. Alle XML-Dokumente entsprechen einem XML-Schema, das als ABC.SALES registriert wurde. Das Schema wurde mit Informationen zur Dekomposition (Zerlegung) annotiert und für die Dekomposition aktiviert. Mit dem folgenden Beispiel wird die gespeicherte Prozedur zur Dekomposition der in SALESDOC gespeicherten Dokumente mithilfe des Schemas ABC.SALES aufgerufen:

```
XDB_DECOMP_XML_FROM_QUERY ('ABC', 'SALES',
    'SELECT DOCID, SALESDOC FROM ABC.SALESTAB', 0, 0, 0,
    NULL, NULL, 1, numInput, numDecomposed, errorreportBuf);
```

XML-Dekompositionsannotationen

Die Dekomposition mithilfe eines mit Annotationen versehenen XML-Schemas basiert auf Annotationen, die XML-Schemadokumenten hinzugefügt werden. Diese Dekompositionsannotationen fungieren als Zuordnungen zwischen den Elementen oder Attributen des XML-Dokuments und den entsprechenden Zieltabellen und -spalten in der Datenbank. Bei der Dekompositionsverarbeitung werden diese Annotationen verwendet, um zu ermitteln, wie ein XML-Dokument zu zerlegen ist.

Die XML-Dekompositionsannotationen gehören zum Namensbereich http://www.ibm.com/xmlns/prod/db2/xdb1 und werden in der gesamten Dokumentation durch das Präfix "db2-xdb" identifiziert. Sie können ein eigenes Präfix auswählen; in diesem Fall müssen Sie das Präfix jedoch an den folgenden Namensbereich binden: http://www.ibm.com/xmlns/prod/db2/xdb1. Der Dekompositionsprozess erkennt nur Annotationen, die sich zum Zeitpunkt, an dem das XML-Schema für die Dekomposition aktiviert wird, in diesem Namensbereich befinden.

Die Dekompositionsannotationen werden vom Dekompositionsprozess nur dann erkannt, wenn sie zu den Element- und Attributdeklarationen oder als globale Annotationen im Schemadokument hinzugefügt werden. Sie werden entweder als Attribute oder als Teil eines untergeordneten <xs:annotation>-Elements der Element- oder Attributdeklaration angegeben. Annotationen, die zu komplexen Typen, Referenzen oder anderen XML-Schemakonstrukten hinzugefügt werden, werden ignoriert.

Obwohl diese Annotationen in den XML-Schemadokumenten vorhanden sind, haben sie keine Auswirkung auf die ursprüngliche Struktur des Schemadokuments und werden nicht bei der Prüfung von XML-Dokumenten berücksichtigt. Sie werden nur vom XML-Dekompositionsprozess verwendet.

Zwei Annotationen, die zentrale Funktionen des Dekompositionsprozesses darstellen, sind: db2-xdb:rowSet und db2-xdb:column. Diese Annotationen geben die Zieltabelle bzw. die Zielspalte des zerlegten Werts an. Diese beiden Annotationen müssen angegeben werden, damit der Dekompositionsprozess erfolgreich ausgeführt

werden kann. Andere Annotationen sind optional, können jedoch zur weiteren Steuerung des Dekompositionsprozesses verwendet werden.

XML-Dekompositionsannotationen - Spezifikation und Geltungsbereich

Annotationen zur Dekomposition können entweder als Element- oder als Attributdeklarationen in einem XML-Schemadokument angegeben werden.

Annotationen können in folgenden Formen angegeben werden:

- Als einfaches Attribut in einer Element- oder Attributdeklaration
- Als strukturiertes (komplexes) untergeordnetes Element einer Element- oder Attributdeklaration, die aus einfachen Elementen und Attributen besteht

Annotationen als Attribute

Annotationen, die als einfache Attribute in Element- oder Attributdeklarationen angegeben werden, gelten nur für das Element bzw. Attribut, in dem sie angegeben sind.

Zum Beispiel können die Dekompositionsannotationen 'db2-xdb:rowSet' und 'db2-xdb:column' als Attribute angegeben werden. Diese Annotationen werden wie folgt angegeben:

```
<xs:element name="isbn" type="xs:string"
db2-xdb:rowSet="TEXTBOOKS" db2-xdb:column="ISBN"/>
```

Die Annotationen 'db2-xdb:rowSet' und 'db2-xdb:column' gelten nur für das Element mit dem Namen 'isbn'.

Annotationen als strukturierte untergeordnete Elemente

Annotationen, die als strukturierte untergeordnete Elemente einer Element- oder Attributdeklaration angegeben werden, müssen im Schemadokument innerhalb der durch das XML-Schema definierten Hierarchie <xs:annotation><xs:appinfo></xs:appinfo></xs:annotation> angegeben werden.

Zum Beispiel können die Annotationen 'db2-xdb:rowSet' und 'db2-xdb:column' als untergeordnete Elemente (sie sind untergeordnete Elemente der Annotation '<db2-xdb:rowSetMapping>') wie folgt angegeben werden:

```
<xs:element name="isbn" type="xs:string">
          <xs:annotation>
                      <xs:appinfo>
      <db2-xdb:rowSetMapping>
        <db2-xdb:rowSet>TEXTBOOKS</db2-xdb:rowSet>
        <db2-xdb:column>ISBN</db2-xdb:column>
      </db2-xdb:rowSetMapping>
                      </xs:appinfo>
          </xs:annotation>
  </xs:element>
```

Die Angabe der Annotationen 'db2-xdb:rowSet' und 'db2-xdb:column' als untergeordnete Elemente ist äquivalent mit der Angabe dieser Annotationen als Attribute (wie zuvor gezeigt). Die Angabe von Annotationen als untergeordnete Elemente ist zwar aufwendiger als die Attributmethode, jedoch erforderlich, wenn mehr als eine Zuordnung vom Typ <db2-xdb:rowSetMapping> für ein Element oder Attribut angegeben werden muss, das heißt, wenn mehrere Zuordnungen in derselben Element- oder Attributdeklaration angegeben werden müssen.

Globale Annotationen

Wenn eine Annotation als untergeordnetes Element des Elements <xs:schema> angegeben wird, ist sie eine globale Annotation, die für alle XML-Schemadokumente gilt, aus denen das XML-Schema besteht.

Zum Beispiel gibt die Annotation <db2-xdb:defaultSQLSchema> das SQL-Standardschema für alle nicht qualifizierten Tabellen an, auf die im XML-Schema verwiesen wird. Die Annotation <db2-xdb:defaultSQLSchema> muss als untergeordnetes Element von <xs:schema> angegeben werden:

```
<xs:schema>
          <xs:annotation>
                        <xs:appinfo>
      <db2-xdb:defaultSQLSchema>admin</db2-xdb:defaultSQLSchema>
                        </xs:appinfo>
          </xs:annotation>
  ...
</xs:schema>
```

Diese Deklaration gibt an, dass alle nicht qualifizierten Tabellen in allen Schemadokumenten, aus denen dieses XML-Schema besteht, mit dem SQL-Schema "admin" qualifiziert werden.

Informationen dazu, wie eine bestimmte Annotation angegeben werden kann, finden Sie in der Dokumentation zu dieser Annotation.

XML-Dekompositionsannotationen - Zusammenfassung

DB2 unterstützt eine Reihe von Annotationen, die von der Dekomposition mithilfe mit Annotationen versehener XML-Schemata verwendet werden, um Elemente und Attribute aus einem XML-Dokument Zieldatenbanktabellen zuzuordnen. Die folgende Zusammenfassung einiger dieser XML-Dekompositionsannotationen ist nach den Aufgaben und Aktionen unterteilt, für die die entsprechenden Annotationen verwendet werden.

Weitere Informationen zu einer bestimmten Annotation finden Sie in der zugehörigen ausführlichen Dokumentation.

Tabelle 42. Angabe des SQL-Schemas

Aktion	**XML-Dekompositionannotationen**
SQL-Standardschema für alle Tabellen angeben, die kein entsprechendes SQL-Schema angeben	db2-xdb:defaultSQLSchema
Vom Standardwert abweichendes SQL-Schema für eine bestimmte Tabelle angeben	db2-xdb:table (<db2-xdb:SQLSchema>, untergeordnetes Element)

Tabelle 43. XML-Elemente oder -Attribute Zielbasistabellen zuordnen

Aktion	**XML-Dekompositionannotationen**
Einzelnes Element oder Attribut einem einzelnen Spalten-Tabellen-Paar zuordnen	db2-xdb:rowSet mit db2-xdb:column als Attributannotationen oder db2-xdb:rowSetMapping
Einzelnes Element oder Attribut einem oder mehreren bestimmten Spalten-Tabellen-Paaren zuordnen	db2-xdb:rowSetMapping

Tabelle 43. XML-Elemente oder -Attribute Zielbasistabellen zuordnen (Forts.)

Aktion	XML-Dekompositionannotationen
Mehrere Elemente oder Attribute einem einzelnen Spalten-Tabellen-Paar zuordnen	db2-xdb:table
Abhängigkeiten zwischen Zieltabellen für die Sortierung angeben	db2-xdb:rowSetOperationOrder, db2-xdb:rowSet, db2-xdb:order

Tabelle 44. XML-Daten für die Dekomposition angeben

Aktion	XML-Dekompositionannotationen
Typ des Inhalts angeben, der für ein Element mit komplexem Typ eingefügt werden soll (Text, Zeichenfolge oder Markup)	db2-xdb:contentHandling
Inhaltskonvertierung angeben, die vor dem Einfügen angewendet werden soll	db2-xdb:normalization, db2-xdb:expression, db2-xdb:truncate
Daten für die Dekomposition auf der Basis des Elementinhalts oder -kontexts filtern	db2-xdb:condition db2-xdb:locationPath

Dekompositionsannotation db2-xdb:defaultSQLSchema

Die Annotation db2-xdb:defaultSQLSchema gibt das SQL-Standardschema für alle Tabellennamen an, auf die im XML-Schema verwiesen wird und die nicht explizit durch die Annotation db2-xdb:table qualifiziert sind.

<db2-xdb:defaultSQLSchema> gehört zur Gruppe der Dekompositionsannotationen, die einem XML-Schemadokument hinzugefügt werden können, um die Zuordnung zwischen Elementen und Attributen von XML-Dokumenten zu DB2-Basistabellen zu beschreiben. Der Dekompositionsprozess verwendet das mit Annotationen versehene XML-Schema, um festzustellen, wie die Elemente und Attribute eines XML-Dokuments zerlegt und in DB2-Tabellen eingefügt werden sollen.

Annotationstyp

Untergeordnetes Element von <xs:appinfo>, das ein untergeordnetes Element eines globalen Elements <xs:annotation> ist.

Angabe

db2-xdb:defaultSQLSchema wird wie folgt angegeben (dabei stellt *wert* einen gültigen Wert für die Annotation dar):

```
<xs:schema>
          <xs:annotation>
                        <xs:appinfo>
      <db2-xdb:defaultSQLSchema>wert</db2-xdb:defaultSQLSchema>
                        </xs:appinfo>
          </xs:annotation>
  ...
</xs:schema>
```

Namensbereich

http://www.ibm.com/xmlns/prod/db2/xdb1

Gültige Werte

Normaler oder mit Begrenzern versehener SQL-Schemaname. Bei normalen SQL-Schemanamen (d. h. ohne Begrenzer) muss die Groß-/Kleinschreibung nicht beachtet werden. Verwenden Sie zur Angabe eines SQL-Schemas mit Begrenzern Anführungszeichen, die normalerweise als Begrenzer für SQL-IDs verwendet werden. Für Schemanamen, die die Sonderzeichen '<' und '&' enthalten, müssen im XML-Schemadokument Escapezeichen verwendet werden.

Details

Alle Tabellen, auf die in mit Annotationen versehenen Schemata verwiesen wird, müssen für das entsprechende SQL-Schema qualifiziert sein. Für die Qualifizierung von Tabellen gibt es zwei Möglichkeiten: Entweder durch die explizite Angabe des untergeordneten Elements <db2-xdb:SQLSchema> der Annotation <db2-xdb:table> oder durch die Verwendung der globalen Annotation <db2-xdb:defaultSQLSchema>. Für alle nicht qualifizierten Tabellennamen wird der in <db2-xdb:defaultSQLSchema> angegebene Wert als SQL-Schemaname verwendet. Wenn mehrere Schemadokumente in einem mit Annotationen versehenen Schema diese Annotation angeben, müssen alle Werte gleich sein.

Beispiel

Das folgende Beispiel zeigt, wie die normale SQL-ID (SQL-ID ohne Begrenzer) `admin` als SQL-Schema für alle nicht qualifizierten Tabellen im mit Annotationen versehenen Schema definiert wird:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
                xmlns:db2-xdb="http://www.ibm.com/xmlns/prod/db2/xdb1">
          <xs:annotation>
                        <xs:appinfo>
      <db2-xdb:defaultSQLSchema>admin</db2-xdb:defaultSQLSchema>
    </xs:appinfo>
  </xs:annotation>
  ...
</xs:schema>
```

Das folgende Beispiel zeigt, wie die SQL-ID mit Begrenzern `admin schema` als SQL-Schema für alle nicht qualifizierten Tabellen im mit Annotationen versehenen Schema definiert wird. Es ist zu beachten, dass `admin schema` in Anführungszeichen eingeschlossen werden muss.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:db2-xdb="http://www.ibm.com/xmlns/prod/db2/xdb1">
          <xs:annotation>
                        <xs:appinfo>
      <db2-xdb:defaultSQLSchema>"admin schema"</db2-xdb:defaultSQLSchema>
    </xs:appinfo>
  </xs:annotation>
  ...
</xs:schema>
```

Dekompositionsannotation db2-xdb:rowSet

Die Annotation db2-xdb:rowSet gibt eine Zuordnung für ein XML-Element oder -Attribut zu einer Zielbasistabelle an.

db2-xdb:rowSet gehört zur Gruppe der Dekompositionsannotationen, die einem XML-Schemadokument hinzugefügt werden können, um die Zuordnung zwischen Elementen und Attributen von XML-Dokumenten zu DB2-Basistabellen zu beschreiben. Der Dekompositionsprozess verwendet das mit Annotationen versehene

XML-Schema, um festzustellen, wie die Elemente und Attribute eines XML-Dokuments zerlegt und in DB2-Tabellen eingefügt werden sollen.

Annotationstyp

Attribut <xs:element> oder <xs:attribute> oder untergeordnetes Element <db2-xdb:rowSetMapping> oder <db2-xdb:order>

Angabe

db2-xdb:rowSet kann auf eine der nachfolgend beschriebenen Arten angegeben werden (dabei stellt *wert* einen gültigen Wert für die Annotation dar):

- `<xs:element db2-xdb:rowSet="wert" />`
- `<xs:attribute db2-xdb:rowSet="wert" />`
- ```
  <db2-xdb:rowSetMapping>
    <db2-xdb:rowSet>wert</db2-xdb:rowSet>
    ...
  </db2-xdb:rowSetMapping>
  ```
- ```
  <db2-xdb:order>
    <db2-xdb:rowSet>wert</db2-xdb:rowSet>
    ...
  </db2-xdb:order>
  ```

Namensbereich

http://www.ibm.com/xmlns/prod/db2/xdb1

Gültige Werte

Jede ID, die den Regeln für SQL-IDs entspricht. Weitere Informationen finden Sie in der Dokumentation zu den IDs.

Details

Die Annotation db2-xdb:rowSet ordnet ein XML-Element oder -Attribut einer Zielbasistabelle zu. Diese Annotation kann entweder eine Tabellennamen direkt identifizieren oder - in komplexeren Zuordnungen - eine Zeilengruppe, die dann mithilfe der Annotation db2-xdb:table einem Tabellennamen zugeordnet wird. Bei einfachen Zuordnungen gibt diese Annotation den Namen der Tabelle an, in die der Wert mithilfe der Dekomposition eingefügt werden soll. Bei komplexeren Zuordnungen, in denen mehrere Zeilengruppen (jeweils mit einem eindeutigen Namen) derselben Tabelle zugeordnet werden, benennt diese Annotation anstelle des Tabellennamens die Zeilengruppe.

Die Zielbasistabelle, in die der Wert dieses XML-Elements bzw. -Attributs mithilfe der Dekomposition eingefügt wird, wird durch die Angabe weiterer Annotationen in der Gruppe der Schemadokumente bestimmt, aus denen das mit Annotationen versehene Schema besteht:

- Wenn der Wert von db2-xdb:rowSet nicht mit einem der untergeordneten Elemente <db2-xdb:rowSet> der globalen Annotation <db2-xdb:table> übereinstimmt, ist der Name der Zieltabelle der durch diese Annotation angegebene Wert, qualifiziert durch ein SQL-Schema, das durch die globale Annotation <db2-xdb:defaultSQLSchema> definiert wird. Diese Verwendung von db2-xdb:rowSet gilt für den Fall, in dem für eine bestimmte Tabelle nur eine Gruppe von Elementen bzw. Attributen vorhanden ist, die der Tabelle zugeordnet wird.

- Wenn der Wert von db2-xdb:rowSet mit einem untergeordneten Element <db2-xdb:rowSet> der globalen Annotation <db2-xdb:table> übereinstimmt, ist der Name der Zieltabelle die im untergeordnet Element <db2-xdb:name> von <db2-xdb:table> genannte Tabelle. Diese Verwendung von db2-xdb:rowSet gilt für die komplexeren Fälle, in denen für eine bestimmte Tabelle mehrere (sich möglicherweise überschneidende) Gruppen von Elementen bzw. Attributen vorhanden sind, die dieser Tabelle zugeordnet werden.

Wichtig: Stellen Sie sicher, dass die Tabelle, auf die diese Annotation verweist, in der Datenbank vorhanden ist, wenn das XML-Schema im XML-Schema-Repository registriert wird. (Die in den db2-xdb:column-Annotationen angegebenen Spalten müssen bei der Registrierung des XML-Schemas ebenfalls vorhanden sein.) Ist die Tabelle nicht vorhanden, wird ein Fehler zurückgegeben, wenn das XML-Schema für die Dekomposition aktiviert wird. Wenn <db2-xdb:table> ein Objekt angibt, bei dem es sich nicht um eine Tabelle handelt, wird ebenfalls ein Fehler zurückgegeben.

Wenn die Annotation db2-xdb:rowSet verwendet wird, muss entweder die Annotationdb2-xdb:column oder die Annotation db2-xdb:condition angegeben werden. Die Kombination aus db2-xdb:rowSet und db2-xdb:column beschreibt die Tabelle und die Spalte, in die dieses Element oder Attribut durch die Dekomposition eingefügt wird. Die Kombination aus db2-xdb:rowSet und db2-xdb:condition gibt die Bedingung an, die erfüllt sein muss, damit Zeilen dieser Zeilengruppe in die Tabelle (auf die direkt oder indirekt über die Annotation <db2-xdb:table> verwiesen wird) eingefügt werden.

Beispiel

Die beiden Verwendungsmöglichkeiten der oben beschriebenen Annotation db2-xdb:rowSet sind nachfolgend dargestellt.

Zuordnung einer einzelnen Gruppe von Elementen oder Attributen zu einer Tabelle

Für den folgenden Abschnitt eines mit Annotationen versehenen Schemas wird davon ausgegangen, dass die Tabelle BOOKCONTENTS zu dem durch <db2-xdb:defaultSQLSchema> angegebenen SQL-Schema gehört und dass kein globales Element <db2-xdb:table> vorhanden ist, das über ein untergeordnetes Element <db2-xdb:rowSet> verfügt, das mit "BOOKCONTENTS" übereinstimmt.

```
<xs:element name="book">
   <xs:complexType>
     <xs:sequence>
       <xs:element name="authorID" type="xs:integer" />
       <xs:element name="chapter" type="chapterType" maxOccurs="unbounded" />
     </xs:sequence>
     <xs:attribute name="isbn" type="xs:string"
                   db2-xdb:rowSet="BOOKCONTENTS" db2-xdb:column="ISBN" />
     <xs:attribute name="title" type="xs:string" />
   </xs:complexType>
 </xs:element>

 <xs:complexType name="chapterType">
   <xs:sequence>
     <xs:element name="paragraph" type="paragraphType" maxOccurs="unbounded"
                 db2-xdb:rowSet="BOOKCONTENTS" db2-xdb:column="CHPTCONTENT" />
   </xs:sequence>
   <xs:attribute name="number" type="xs:integer"
                 db2-xdb:rowSet="BOOKCONTENTS" db2-xdb:column="CHPTNUM" />
   <xs:attribute name="title" type="xs:string"
```

```
                db2-xdb:rowSet="BOOKCONTENTS" db2-xdb:column="CHPTTITLE" />
  </xs:complexType>

  <xs:simpleType name="paragraphType">
    <xs:restriction base="xs:string"/>
  </xs:simpleType>
```

Betrachten Sie das folgende Element aus einem XML-Dokument:

```
<book isbn="1-11-111111-1" title="My First XML Book">
  <authorID>22</authorID>
  <!-- this book does not have a preface -->
  <chapter number="1" title="Introduction to XML">
    <paragraph>XML is fun...</paragraph>
    ...
  </chapter>
  <chapter number="2" title="XML and Databases">
    <paragraph>XML can be used with...</paragraph>
    </chapter>
  ...
  <chapter number="10" title="Further Reading">
    <paragraph>Recommended tutorials...</paragraph>
  </chapter>
  ...
</book>
```

Die Tabelle BOOKCONTENTS wird wie folgt mit Daten gefüllt:

Tabelle 45. BOOKCONTENTS

ISBN	CHPTNUM	CHPTTITLE	CHPTCONTENT
1-11-111111-1	1	Introduction to XML	XML is fun...
1-11-111111-1	2	XML and Databases	XML can be used with...
...	...	...	...
1-11-111111-1	10	Further Reading	Recommended tutorials...

Zuordnung mehrerer Gruppen von Elementen oder Attributen zur selben Tabelle

Für den Fall, in dem ein untergeordnetes Element <db2-xdb:rowSet> der globalen Annotation <db2-xdb:table> vorhanden ist, das mit dem in der Annotation db2-xdb:rowSet angegebenen Wert übereinstimmt, wird das Element bzw. Attribut über die Annotation <db2-xdb:table> einer Tabelle zugeordnet. Für den folgenden Abschnitt eines mit Annotationen versehenen Schemas wird davon ausgegangen, dass die Tabelle ALLBOOKS zu dem durch <db2-xdb:defaultSQLSchema> angegebenen SQL-Schema gehört.

```
<!-- global annotation -->
<xs:annotation>
  <xs:appinfo>
    <db2-xdb:table>
       <db2-xdb:name>ALLBOOKS</db2-xdb:name>
       <db2-xdb:rowSet>book</db2-xdb:rowSet>
       <db2-xdb:rowSet>textbook</db2-xdb:rowSet>
    </db2-xdb:table>
  </xs:appinfo>
</xs:annotation>

<xs:element name="book">
  <xs:complexType>
```

```
    <xs:sequence>
      <xs:element name="authorID" type="xs:integer"
                  db2-xdb:rowSet="book" db2-xdb:column="AUTHORID" />
      <xs:element name="chapter" type="chapterType" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="isbn" type="xs:string"
                  db2-xdb:rowSet="book" db2-xdb:column="ISBN" />
    <xs:attribute name="title" type="xs:string"
                  db2-xdb:rowSet="book" db2-xdb:column="TITLE" />
  </xs:complexType>
</xs:element>
<xs:element name="textbook">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="isbn" type="xs:string"
                  db2-xdb:rowSet="textbook" db2-xdb:column="ISBN" />
      <xs:element name="title" type="xs:string"
                  db2-xdb:rowSet="textbook" db2-xdb:column="TITLE" />
      <xs:element name="primaryauthorID" type="xs:integer"
                  db2-xdb:rowSet="textbook" db2-xdb:column="AUTHORID" />
      <xs:element name="coauthorID" type="xs:integer"
                  minOccurs="0" maxOccurs="unbounded" />
      <xs:element name="subject" type="xs:string" />
      <xs:element name="edition" type="xs:integer" />
      <xs:element name="chapter" type="chapterType" maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:complexType name="chapterType">
  <xs:sequence>
    <xs:element name="paragraph" type="paragraphType" maxOccurs="unbounded" />
  </xs:sequence>
  <xs:attribute name="number" type="xs:integer" />
  <xs:attribute name="title" type="xs:string" />
</xs:complexType>

<xs:simpleType name="paragraphType">
  <xs:restriction base="xs:string"/>
</xs:simpleType>
```

Betrachten Sie die folgenden Elemente aus einem XML-Dokument:

```
<book isbn="1-11-111111-1" title="My First XML Book">
  <authorID>22</authorID>
  <!-- this book does not have a preface -->
  <chapter number="1" title="Introduction to XML">
    <paragraph>XML is fun...</paragraph>
  </chapter>
  <chapter number="2" title="XML and Databases">
    <paragraph>XML can be used with...</paragraph>
    </chapter>
  <chapter number="10" title="Further Reading">
    <paragraph>Recommended tutorials...</paragraph>
    </chapter>
</book>

<textbook>
  <isbn>0-11-011111-0</isbn>
  <title>Programming with XML</title>
  <primaryauthorID>435</primaryauthorID>
  <subject>Programming</subject>
  <edition>4</edition>
  <chapter number="1" title="Programming Basics">
    <paragraph>Before you being programming...</paragraph>
  </chapter>
  <chapter number="2" title="Writing a Program">
```

```
    <paragraph>Now that you have learned the basics...</paragraph>
    </chapter>
  ...
  <chapter number="10" title="Advanced techniques">
    <paragraph>You can apply advanced techniques...</paragraph>
  </chapter>
</textbook>
```

Dieses Beispiel zeigt zwei Gruppen von Elementen bzw. Attributen, die der Tabelle ALLBOOKS zugeordnet werden:

- /book/@isbn, /book/@authorID, /book/title
- /textbook/isbn, /textbook/primaryauthorID, /textbook/title

Die Gruppen werden voneinander unterschieden, indem jeder unterschiedliche rowSet-Namen zugeordnet werden.

Tabelle 46. ALLBOOKS

ISBN	TITLE	AUTHORID
1-11-111111-1	My First XML Book	22
0-11-011111-0	Programming with XML	435

Dekompositionsannotation db2-xdb:table

Die Annotation <db2-xdb:table> ordnet mehrere XML-Elemente oder Attribute derselben Zielspalte zu; oder sie ermöglicht es Ihnen, eine Zieltabelle anzugeben, deren SQL-Schema von dem durch <db2-xdb:defaultSQLSchema> angegebenen Standard-SQL-Schema abweicht.

<db2-xdb:table> gehört zur Gruppe der Dekompositionsannotationen, die einem XML-Schemadokument hinzugefügt werden können, um die Zuordnung zwischen Elementen und Attributen von XML-Dokumenten zu DB2-Basistabellen zu beschreiben. Der Dekompositionsprozess verwendet das mit Annotationen versehene XML-Schema, um festzustellen, wie die Elemente und Attribute eines XML-Dokuments zerlegt und in DB2-Tabellen eingefügt werden sollen.

Annotationstyp

Globales untergeordnetes Element von <xs:appinfo> (ein untergeordnetes Element von <xs:annotation>).

Namensbereich

http://www.ibm.com/xmlns/prod/db2/xdb1

Gültige Struktur

Im Folgenden sind unterstützte untergeordnete Elemente von <db2-xdb:table> in der Reihenfolge aufgeführt, in der sie verwendet werden müssen, wenn sie angegeben werden:

<db2-xdb:SQLSchema>
(Optional) Das SQL-Schema der Tabelle.

<db2-xdb:name>
Der Name der Basistabelle. Dieser Tabellenname muss, wenn er durch den Wert der vorhergehenden Annotation <db2-xdb:SQLSchema> oder der Annotation <db2-xdb:defaultSQLSchema> qualifiziert ist, für alle <db2-xdb:ta-

ble>-Annotationen und alle XML-Schemadokumente, aus denen das mit Annotationen versehene Schema besteht, eindeutig sein.

<db2-xdb:rowSet>
Alle Elemente und Attribute, die denselben Wert für <db2-xdb:rowSet> angeben, bilden eine Zeile. Da mehr als ein <db2-xdb:rowSet>-Element für denselben Wert von <db2-xdb:name> angegeben werden kann, können einer einzelnen Tabelle mehrere Zuordnungsgruppen zugeordnet werden. Die Kombination des <db2-xdb:rowSet>-Werts mit den in der Annotation db2-xdb:column angegebenen Spalten ermöglicht es, dass mehrere Element- bzw. Attributgruppen aus einem einzelnen XML-Dokument den Spalten derselben Tabelle zugeordnet werden.

Es muss mindestens ein <db2-xdb:rowSet>-Element angegeben werden, und jedes <db2-xdb:rowSet>-Element muss für alle <db2-xdb:table>-Annotationen und alle XML-Schemadokumente, aus denen das mit Annotationen versehene Schema besteht, eindeutig sein, damit die Annotation gültig ist.

Leerzeichen im Zeichendateninhalt der untergeordneten Elemente von <db2-xdb:table> sind signifikant und werden nicht normalisiert. Der Inhalt dieser Elemente muss den Regeln für die Schreibweise von SQL-IDs entsprechen. Bei Werten ohne Begrenzer muss die Groß-/Kleinschreibung nicht beachtet werden; bei Werten mit Begrenzer werden Anführungszeichen als Begrenzer verwendet. Bei SQL-IDs, die die Sonderzeichen < und & enthalten, müssen Escapezeichen verwendet werden.

Details

Die Annotation <db2-xdb:table> muss in folgenden Fällen verwendet werden:

- Wenn mehrere Abstammungslinien derselben Spalte einer Tabelle zugeordnet werden (für Zuordnungen mit einzelnen Verzeichnispfaden, d. h., wenn nur eine Gruppe von Spaltenzuordnungen für die Tabelle vorhanden ist, müssen diese Annotation nicht verwendet werden; stattdessen kann die Annotation db2-xdb:rowSet verwendet werden).
- Wenn die Tabelle, die die Daten nach der Dekomposition enthalten soll, nicht dasselbe SQL-Schema aufweist wie das von der Annotation <db2-xdb:defaultSQLSchema> definierte.

Es können nur Basistabellen angegeben werden; andere Tabellentypen, wie beispielsweise typisierte, zusammenfassende oder temporäre Tabellen oder MQTs (Materialized Query Tables, gespeicherte Abfragetabellen), werden für diese Zuordnung nicht unterstützt. Kurznamen können nur Datenquellenobjekte von DB2 Database for Linux, UNIX and Windows angegeben werden. Aliasnamen für Sichten und Tabellen sind zum gegenwärtigen Zeitpunkt für diese Annotation nicht zulässig.

Beispiel

Das folgende Beispiel zeigt, wie die Annotation <db2-xdb:table> dazu verwendet werden kann, zusammengehörende Elemente und Attribute in einer Zeile zusammenzufassen, wenn mehrere Verzeichnispfade derselben Spalte zugeordnet werden. Betrachten Sie zunächst die folgenden Elemente aus einem XML-Dokument (sie weichen geringfügig von den Beispiel für andere Annotationen ab).

```
<root>
  ...
  <book isbn="1-11-111111-1" title="My First XML Book">
      <authorID>22</authorID>
```

```
      <email>author22@anyemail.com</email>
    <!-- this book does not have a preface -->
    <chapter number="1" title="Introduction to XML">
        <paragraph>XML is fun...</paragraph>
    ...
    </chapter>
    <chapter number="2" title="XML and Databases">
        <paragraph>XML can be used with...</paragraph>
    </chapter>
  ...
    <chapter number="10" title="Further Reading">
        <paragraph>Recommended tutorials...</paragraph>
    </chapter>
  </book>
...
  <author ID="0800" email="author800@email.com">
    <firstname>Alexander</firstname>
    <lastname>Smith</lastname>
    <activeStatus>0</activeStatus>
  </author>
...
<root>
```

Angenommen, der Zweck dieser Dekompositionszuordnung ist das Einfügen von Zeilen, die aus Autoren-IDs und den zugehörigen E-Mail-Adressen bestehen, in dieselbe Tabelle AUTHORSCONTACT. Die Autoren-IDs und E-Mail-Adressen sind sowohl im Element <book> als auch im Element <author> enthalten. Dies bedeutet, dass mehrere Verzeichnispfade denselben Spalten derselben Tabelle zugeordnet werden müssen. Daher muss die Annotation <db2-xdb:table> verwendet werden. Im Folgenden ist ein Abschnitt aus dem mit Annotationen versehenen Schema dargestellt, der zeigt, wie <db2-xdb:table> dazu verwendet wird, mehrere Pfade derselben Tabelle zuzuordnen.

```
<!-- global annotation -->
  <xs:annotation>
                    <xs:appinfo>
      <db2-xdb:defaultSQLSchema>adminSchema</db2-xdb:defaultSQLSchema>
      <db2-xdb:table>
        <db2-xdb:SQLSchema>user1</db2-xdb:SQLSchema>
        <db2-xdb:name>AUTHORSCONTACT</db2-xdb:name>
        <db2-xdb:rowSet>bookRowSet</db2-xdb:rowSet>
        <db2-xdb:rowSet>authorRowSet</db2-xdb:rowSet>
      </db2-xdb:table>
                    </xs:appinfo>
        </xs:annotation>

    <xs:element name="book">
              <xs:complexType>
      <xs:sequence>
              <xs:element name="authorID" type="xs:integer"
                  db2-xdb:rowSet="bookRowSet" db2-xdb:column="AUTHID" />
              <xs:element name="email" type="xs:string"
                  db2-xdb:rowSet="bookRowSet" db2-xdb:column="EMAILADDR" />
            <xs:element name="chapter" type="chapterType" maxOccurs="unbounded" />
      </xs:sequence>
           <xs:attribute name="isbn" type="xs:string" />
          <xs:attribute name="title" type="xs:string" />
    </xs:complexType>
    </xs:element>
    <xs:element name="author">
              <xs:complexType>
      <xs:sequence>
              <xs:element name="firstname" type="xs:string" />
              <xs:element name="lastname" type="xs:string" />
              <xs:element name="activeStatus" type="xs:boolean" />
      </xs:sequence>
```

```
                <xs:attribute name="ID" type="xs:integer"
                  db2-xdb:rowSet="authorRowSet" db2-xdb:column="AUTHID" />
            <xs:attribute name="email" type="xs:string"
                  db2-xdb:rowSet="authorRowSet" db2-xdb:column="EMAILADDR" />
    </xs:complexType>
    </xs:element>
```

Die Annotation <db2-xdb:table> identifiziert den Namen der Zieltabelle für eine Zuordnung unter Verwendung des untergeordneten Elements db2-xdb:name. In diesem Beispiel ist AUTHORSCONTACT die Zieltabelle. Um sicherzustellen, dass die IDs und E-Mail-Adressen aus dem Element <book> von denen des Elements <author> getrennt sind (d. h., jede Zeile enthält logisch zusammengehörende Werte), wird das Element <db2-xdb:rowSet> zur Zuordnung zusammengehörender Informationen verwendet. Obwohl es sich in diesem Beispiel bei den Elementen <book> und <author> um getrennte Entitäten handelt, kann es Fälle geben, in denen die zuzuordnenden Entitäten nicht getrennt sind und daher eine logische Trennung erforderlich machen, die durch die Verwendung von Zeilengruppen erreicht werden kann.

Beim SQL-Schema der Tabelle AUTHORSCONTACT handelt es sich nicht um das SQL-Standardschema; das Element <db2-xdb:SQLSchema> wird dazu verwendet, dies anzugeben. Die daraus resultierende Tabelle AUTHORSCONTACT ist in Tabelle 1 dargestellt:

Tabelle 47. AUTHORSCONTACT

AUTHID	EMAILADDR
22	author22@anyemail.com
0800	author800@email.com

Dieses Beispiel veranschaulicht, wie die logische Gruppierung von Werten mithilfe von Zeilengruppen sicherstellt, dass nicht zusammengehörende Werte nicht versehentlich demselben Tabellen-Spalten-Paar zugeordnet werden. In diesem Beispiel werden /root/book/authorID und /root/author/@ID demselben Tabellen-Spalten-Paar zugeordnet. Entsprechend werden /root/book/email und /root/author/@email demselben Tabellen-Spalten-Paar zugeordnet. Ohne Zeilengruppen würde sich der Fall wie folgt darstellen: Wenn z. B. das Element '/root/book/email' in einer Instanz des Elements <author> nicht vorhanden ist und keine Zeilengruppen verwendet werden können, wäre es nicht möglich, zu bestimmen, ob 'email' aus dem Element <author> '/root/book/authorID' oder '/root/author/@ID' oder beidem zugeordnet werden soll. So hilft die Zuordnung einer Zeilengruppe zu einer einzelnen Tabelle in der Annotation <db2-xdb:table> bei der logischen Unterscheidung zwischen verschiedenen Gruppen von Zeilen.

Dekompositionsannotation db2-xdb:column

Die Annotation db2-xdb:column gibt einen Spaltennamen der Tabelle an, der ein XML-Element oder Attribut zugeordnet wurde.

db2-xdb:column gehört zur Gruppe der Dekompositionsannotationen, die einem XML-Schemadokument hinzugefügt werden können, um die Zuordnung zwischen Elementen und Attributen von XML-Dokumenten zu DB2-Basistabellen zu beschreiben. Der Dekompositionsprozess verwendet das mit Annotationen versehene XML-Schema, um festzustellen, wie die Elemente und Attribute eines XML-Dokuments zerlegt und in DB2-Tabellen eingefügt werden sollen.

Annotationstyp

Attribut von <xs:element> oder <xs:attribute> oder untergeordnetes Element von <db2-xdb:rowSetMapping>

Angabe

db2-xdb:column kann auf eine der nachfolgend beschriebenen Arten angegeben werden (dabei stellt *wert* einen gültigen Wert für die Annotation dar):

- `<xs:element db2-xdb:rowSet="wert" db2-xdb:column="wert" />`
- `<xs:attribute db2-xdb:rowSet="wert" db2-xdb:column="wert" />`
-
```
<db2-xdb:rowSetMapping>
  <db2-xdb:rowSet>wert</db2-xdb:rowSet>
  <db2-xdb:column>wert</db2-xdb:column>
  ...
</db2-xdb:rowSetMapping>
```

Namensbereich

http://www.ibm.com/xmlns/prod/db2/xdb1

Gültige Werte

Eine beliebige Basistabellenspalte, die die folgenden Bedingungen erfüllt:

- Bei Spaltennamen ohne Begrenzer wird die Groß-/Kleinschreibung nicht beachtet. Bei Spaltennamen mit Begrenzern muss für den Begrenzer das Escapezeichen `&quot;` angegeben werden. Beispiel: Zur Angabe des zweiteiligen Spaltennamens "Erste Spalte" muss db2-xdb:column wie folgt definiert werden:
  ```
  db2-xdb:column="&Erste Spalte&quot;"
  ```
  (Die genannten Bedingungen sind für diese Annotation spezifisch.)
- Nur Spalten mit den folgenden Datentypen können in dieser Annotation angegeben werden: Alle von der SQL-Anweisung CREATE TABLE unterstützten Datentypen, mit Ausnahme von benutzerdefinierten strukturierten Typen.

Details

Die Annotation db2-xdb:column, die als Attribut in der Deklaration eines XML-Elements oder -Attributs oder als untergeordnetes Element von <db2-xdb:rowSet-Mapping> angegeben wird, ordnet ein XML-Element oder -Attribut dem Spaltennamen einer Zieltabelle zu. Wenn diese Annotation verwendet wird, muss auch die Annotation db2-xdb:rowSet angegeben werden. Zusammen beschreiben sie die Tabelle und die Spalte, die den Wert für dieses Element bzw. Attribut nach der Dekomposition enthalten sollen.

Beispiel

Das folgende Beispiel zeigt, wie der Inhalt aus dem Element <book> mithilfe der Annotation db2-xdb:column in die Spalten einer Tabelle mit der Bezeichnung BOOKCONTENTS eingefügt werden kann. Zunächst ist ein Abschnitt aus dem mit Annotationen versehenen Schema dargestellt.

```
<xs:element name="book">
            <xs:complexType>
  <xs:sequence>
```

```
            <xs:element name="authorID" type="xs:integer" />
            <xs:element name="chapter" type="chapterType" maxOccurs="unbounded" />
    </xs:sequence>
        <xs:attribute name="isbn" type="xs:string"
                  db2-xdb:rowSet="BOOKCONTENTS" db2-xdb:column="ISBN" />
        <xs:attribute name="title" type="xs:string" />
  </xs:complexType>
</xs:element>

<xs:complexType name="chapterType">
  <xs:sequence>
        <xs:element name="paragraph" type="paragraphType" maxOccurs="unbounded"
                db2-xdb:rowSet="BOOKCONTENTS"
                db2-xdb:column="CHPTCONTENT" />
  </xs:sequence>
    <xs:attribute name="number" type="xs:integer"
                db2-xdb:rowSet="BOOKCONTENTS"
                db2-xdb:column="CHPTNUM" />
    <xs:attribute name="title" type="xs:string"
                db2-xdb:rowSet="BOOKCONTENTS"
                db2-xdb:column="CHPTTITLE" />
</xs:complexType>

<xs:simpleType name="paragraphType">
    <xs:restriction base="xs:string"/>
</xs:simpleType>
```

Nachfolgend werden das Element <book>, für das eine Zuordnung vorgenommen wird, und die daraus resultierende Tabelle BOOKCONTENTS nach Abschluss der Dekomposition dargestellt. .

```
<book isbn="1-11-111111-1" title="My First XML Book">
    <authorID>22</authorID>
    <!-- this book does not have a preface -->
    <chapter number="1" title="Introduction to XML">
        <paragraph>XML is fun...</paragraph>
    ...
    </chapter>
    <chapter number="2" title="XML and Databases">
        <paragraph>XML can be used with...</paragraph>
    </chapter>
  ...
    <chapter number="10" title="Further Reading">
        <paragraph>Recommended tutorials...</paragraph>
    </chapter>
  </book>
```

Tabelle 48. BOOKCONTENTS

ISBN	CHPTNUM	CHPTTITLE	CHPTCONTENT
1-11-111111-1	1	Introduction to XML	XML is fun...
1-11-111111-1	2	XML and Databases	XML can be used with...
...	...	...	...
1-11-111111-1	10	Further Reading	Recommended tutorials...

Dekompositionsannotation db2-xdb:locationPath

Die Annotation db2-xdb:locationPath ordnet ein XML-Element oder -Attribut, das global deklariert oder Teil einer wiederverwendbaren Gruppe ist, unterschiedlichen Tabelle-Spalte-Paaren zu. Dies erfolgt in Abhängigkeit von den übergeordneten Ele-

menten (der Abstammung) des Elements bzw. Attributs. Wiederverwendbare Gruppen sind global deklarierte, benannte komplexe Typen, benannte Modellgruppen und benannte Attributgruppen.

db2-xdb:locationPath gehört zur Gruppe der Dekompositionsannotationen, die einem XML-Schemadokument hinzugefügt werden können, um die Zuordnung zwischen Elementen und Attributen von XML-Dokumenten zu DB2-Basistabellen zu beschreiben. Der Dekompositionsprozess verwendet das mit Annotationen versehene XML-Schema, um festzustellen, wie die Elemente und Attribute eines XML-Dokuments zerlegt und in DB2-Tabellen eingefügt werden sollen.

Annotationstyp

Attribut <xs:element> oder <xs:attribute> oder Attribut <db2-xdb:rowSetMapping>

Angabe

db2-xdb:locationPath kann auf eine der nachfolgend beschriebenen Arten angegeben werden (dabei stellt *wert* einen gültigen Wert für die Annotation dar):

- `<xs:element db2-xdb:locationPath="wert" />`
- `<xs:attribute db2-xdb:locationPath="wert" />`
-
```
<db2-xdb:rowSetMapping db2-xdb:locationPath="wert">
  <db2-xdb:rowSet>wert</db2-xdb:rowSet>
  ...
</db2-xdb:rowSetMapping>
```

Namensbereich

http://www.ibm.com/xmlns/prod/db2/xdb1

Gültige Werte

Der Wert von db2-xdb:locationPath für den Positionspfad muss die folgende Syntax aufweisen:

```
location path := '/' (locationstep '/')* lastlocationstep
locationstep := (prefix':')? name
lastlocationstep := locationstep | '@' (prefix:)? name
```

Hierbei ist `name` ein Element- oder Attributname, und `präfix` ist ein Namensbereichspräfix.

Anmerkungen:

- Alle Namensbereichspräfixe, die im Positionspfad ('locationpath') verwendet werden, müssen einem Namensbereich in dem Schemadokument zugeordnet sein, das die Annotation enthält, die diesen Positionspfad angibt.
- Eine Namensbereichspräfixbindung kann erstellt werden, indem eine Namensbereichsdeklaration dem Element <xs:schema> des Schemadokuments hinzugefügt wird.
- Wenn `präfix` leer ist, wird angenommen, dass sich `name` in keinem Namensbereich befindet. Wenn im Schemadokument ein Standardnamensbereich deklariert wird und ein Name in `positionsschritt` zu diesem Namensbereich gehört, muss ein Namensbereichspräfix für den Standardnamensbereich deklariert und

als Qualifikationsmerkmal für den Namen verwendet werden. In der Annotation db2-xdb:locationPath verweist ein leeres Präfix nicht auf den Standardnamensbereich.

Details

Die Annotation db2-xdb:locationPath wird dazu verwendet, die Zuordnungen für Elemente oder Attribute zu beschreiben, die global deklariert sind oder Teil von Folgendem sind:

- einer benannten Modellgruppe
- einer benannten Attributgruppe
- einer globalen Deklaration eines komplexen Typs
- einem globalen Element bzw. Attribut mit einfachem oder komplexem Typ

Für Element- oder Attributdeklarationen, die nicht wiederverwendet werden können (lokale Deklarationen, die nicht Teil von Definitionen benannter komplexer Typen oder von benannten Modell- oder Attributgruppen sind), ist die Annotation db2-xdb:locationPath nicht wirksam.

Verwenden Sie db2-xdb:locationPath, wenn globale Element- oder Attributdeklarationen als Verweise von verschiedenen übergeordneten Linien (Abstammungslinien) verwendet werden. (Beispiel: <xs:element ref="abc">). Da Annotationen nicht direkt für Referenzen angegeben werden können, müssen sie stattdessen für die entsprechende globale Element- oder Attributdeklaration angegeben werden. Da die entsprechende Element- oder Attributdeklaration global ist, kann von einer Vielzahl verschiedener Kontexte innerhalb des XML-Schemas aus auf das Element verwiesen werden. Im Allgemeinen sollte db2-xdb:locationPath dazu verwendet werden, die Zuordnungen in diesen verschiedenen Kontexten voneinander zu unterscheiden. Bei benannten komplexen Typen, Modellgruppen und Attributgruppen müssen die Element- und Attributdeklarationen für jeden Kontext, in dem sie für die Dekomposition zugeordnet sind, mit Annotationen versehen werden. Verwenden Sie die Annotation db2-xdb:locationPath, um das Zielpaar aus Zeilengruppe und Spalte für jeden Positionspfad anzugeben. Derselbe db2-xdb:locationPath-Wert kann für verschiedne Paare aus Zeilengruppe und Spalte verwendet werden.

Beispiel

Das folgende Beispiel zeigt, wie dasselbe Attribut, abhängig vom Kontext, in dem es vorkommt, verschiedenen Tabellen zugeordnet werden kann. Zunächst ist ein Abschnitt aus dem mit Annotationen versehenen Schema dargestellt.

```
<!-- global attribute -->
  <xs:attribute name="title" type="xs:string"
              db2-xdb:rowSet="BOOKS"
              db2-xdb:column="TITLE"
              db2-xdb:locationPath="/books/book/@title">
          <xs:annotation>
                      <xs:appinfo>
      <db2-xdb:rowSetMapping db2-xdb:locationPath="/books/book/chapter/@title">
        <db2-xdb:rowSet>BOOKCONTENTS</db2-xdb:rowSet>
        <db2-xdb:column>CHPTTITLE</db2-xdb:column>
      </db2-xdb:rowSetMapping>
                      </xs:appinfo>
          </xs:annotation>
</xs:attribute>

  <xs:element name="books">
              <xs:complexType>
                  <xs:sequence>
```

```
        <xs:element name="book">
                    <xs:complexType>
                        <xs:sequence>
                   <xs:element name="authorID" type="xs:integer" />
                   <xs:element name="chapter" type="chapterType" maxOccurs="unbounded" />
                        </xs:sequence>
                  <xs:attribute name="isbn" type="xs:string" />
                 <xs:attribute ref="title" />
                       </xs:complexType>
      </xs:element>
                   </xs:sequence>
                 </xs:complexType>
  </xs:element>

  <xs:complexType name="chapterType">
                  <xs:sequence>
           <xs:element name="paragraph" type="paragraphType" maxOccurs="unbounded" />
                  </xs:sequence>
      <xs:attribute name="number" type="xs:integer" />
      <xs:attribute ref="title" />
            </xs:complexType>

  <xs:simpleType name="paragraphType">
      <xs:restriction base="xs:string"/>
  </xs:simpleType>
```

Bitte beachten Sie, dass nur eine Attributdeklaration mit der Bezeichnung "title" vorhanden ist, jedoch zwei Referenzen auf dieses Attribut in unterschiedlichen Kontexten. Eine Referenz stammt vom Element <book>, die andere vom Element <chapter>. Für den Wert des Attributs "title" muss abhängig vom Kontext eine Dekomposition in verschiedene Tabellen durchgeführt werden. Dieses mit Annotationen versehene Schema gibt an, dass für einen "title"-Wert eine Dekomposition in eine Tabelle BOOKS durchgeführt wird, wenn es sich um einen Buchtitel handelt, und in die Tabelle 'BOOKCONTENTS', wenn es sich um einen Kapiteltitel handelt.

Als Nächstes werden das Element <books>, für das eine Zuordnung vorgenommen wird, und anschließend die daraus resultierende Tabelle BOOKS nach Abschluss der Dekomposition dargestellt.

```
<books>
  <book isbn="1-11-111111-1" title="My First XML Book">
      <authorID>22</authorID>
      <!-- this book does not have a preface -->
      <chapter number="1" title="Introduction to XML">
          <paragraph>XML is fun...</paragraph>
      ...
      </chapter>
      <chapter number="2" title="XML and Databases">
          <paragraph>XML can be used with...</paragraph>
      </chapter>
    ...
      <chapter number="10" title="Further Reading">
          <paragraph>Recommended tutorials...</paragraph>
      </chapter>
  </book>
...
</books>
```

Tabelle 49. BOOKS

ISBN	TITLE	CONTENT
NULL	My First XML Book	NULL

Tabelle 50. BOOKCONTENTS

ISBN	CHPTNUM	CHPTTITLE	CHPTCONTENT
NULL	NULL	Introduction to XML	NULL
NULL	NULL	XML and Databases	NULL
...	...	...	...
NULL	NULL	Further Reading	NULL

Dekompositionsannotation db2-xdb:expression

Die Annotation db2-xdb:expression gibt einen angepassten Ausdruck an, dessen Ergebnis in die Tabelle eingefügt wird, der dieses Element zugeordnet ist.

db2-xdb:expression gehört zur Gruppe der Dekompositionsannotationen, die einem XML-Schemadokument hinzugefügt werden können, um die Zuordnung zwischen Elementen und Attributen von XML-Dokumenten zu DB2-Basistabellen zu beschreiben. Der Dekompositionsprozess verwendet das mit Annotationen versehene XML-Schema, um festzustellen, wie die Elemente und Attribute eines XML-Dokuments zerlegt und in DB2-Tabellen eingefügt werden sollen.

Annotationstyp

Attribut von <xs:element> oder <xs:attribute> oder optionales untergeordnetes Element von <db2-xdb:rowSetMapping>; nur wirksam bei Annotationen mit Spaltenzuordnung.

Angabe

db2-xdb:expression kann auf eine der nachfolgend beschriebenen Arten angegeben werden (dabei stellt *wert* einen gültigen Wert für die Annotation dar):

- `<xs:element db2-xdb:expression="wert" db2-xdb:column="wert" />`
- `<xs:attribute db2-xdb:expression="wert" db2-xdb:column="wert" />`
-
```
<db2-xdb:rowSetMapping>
  <db2-xdb:rowSet>wert</db2-xdb:rowSet>
  <db2-xdb:column>wert</db2-xdb:column>
  <db2-xdb:expression>wert</db2-xdb:expression>
  ...
</db2-xdb:rowSetMapping>
```

Namensbereich

http://www.ibm.com/xmlns/prod/db2/xdb1

Gültige Werte

Der Wert von db2-xdb:expression muss die folgende Syntax aufweisen, die eine Untermenge von SQL-Ausdrücken darstellt:

```
ausdruck := funktion (argumentliste) | konstante | $DECOMP_CONTENT | $DECOMP_ELEMENTID |
   $DECOMP_DOCUMENTID | (skalarer fullselect) | ausdruck operator ausdruck |
   (ausdruck) | sonderregister | CAST (ausdruck AS datentyp) |
   XMLCAST (ausdruck AS datentyp) | XML-funktion

operator := + | - | * | / | CONCAT

argumentliste := ausdruck | argumentliste, ausdruck
```

Details

Mit der Annotation db2-xdb:expression können Sie einen angepassten Ausdruck angeben, der für den Inhalt des XML-Elements oder -Attributs angewendet wird, das bei der Verwendung von $DECOMP_CONTENT annotiert wird. Das Ergebnis der Auswertung dieses Ausdrucks wird dann in die Spalte eingefügt, die bei der Dekomposition angegeben wird.

Diese Annotation ist darüber hinaus in den Fällen nützlich, in denen konstante Werte (z. B. der Name eines Elements) oder generierte Werte eingefügt werden sollen, die im Dokument nicht enthalten sind.

db2-xdb:expression muss mit gültigen SQL-Ausdrücken angegeben werden, und der Typ des ausgewerteten Ausdrucks muss statisch bestimmbar und mit dem Typ der Zielspalte kompatibel sein, in die der Wert eingefügt werden soll. Die folgende Untermenge von SQL-Ausdrücken wird unterstützt; alle anderen SQL-Ausdrücke, die in der folgenden Liste nicht beschrieben sind, werden nicht unterstützt und weisen ein nicht definiertes Verhalten im Kontext dieser Annotation auf.

funktion (argumentliste)
: Eine integrierte oder benutzerdefinierte SQL-Skalarfunktion. Die Argumente einer Skalarfunktion sind einzelne skalare Werte. Eine Skalarfunktion gibt einen einzelnen Wert zurück (null ist möglich). Die Dokumentation zu Funktionen enthält weitere Informationen hierzu.

konstante
: Ein Wert, auch als Literal bezeichnet, bei dem es sich um eine Zeichenfolgekonstante oder eine numerische Konstante handelt. Die Dokumentation zu Konstanten enthält weitere Informationen hierzu.

$DECOMP_CONTENT
: Der Wert des zugeordneten XML-Elements oder -Attributs aus dem Dokument, der entsprechend der Einstellung der Annotation 'db2-xdb:content-Handling' erstellt wird. Die Dokumentation zu den Dekompositionsschlüsselwörtern enthält weitere Informationen hierzu.

$DECOMP_ELEMENTID
: Eine vom System generierte ganzzahlige ID, die das Element oder Attribut, das durch diese Annotation beschrieben wird, innerhalb des XML-Dokuments eindeutig identifiziert. Die Dokumentation zu den Dekompositionsschlüsselwörtern enthält weitere Informationen hierzu.

$DECOMP_DOCUMENTID
: Der Zeichenfolgewert, der im Eingabeparameter *documentid* der gespeicherten Prozedur 'xdbDecompXML' angegeben wird und der das XML-Dokument für die Dekomposition identifiziert. Die Dokumentation zu den Dekompositionsschlüsselwörtern enthält weitere Informationen hierzu.

(skalarer fullselect)
: Ein Fullselect, in runde Klammern eingeschlossen, der eine einzelne Zeile mit einem einzelnen Spaltenwert zurückgibt. Wenn der Fullselect keine Zeile zurückgibt, ist das Ergebnis des Ausdrucks ein NULL-Wert.

ausdruck operator ausdruck
: Das Ergebnis zweier unterstützter Ausdrucksoperanden, wie in der oben angeführten Liste der unterstützten Werte definiert. Die Dokumentation zu Ausdrücken enthält Einzelheiten zu den Ausdruckoperationen.

(ausdruck)
Ein in runde Klammern eingeschlossener Ausdruck, der der oben angeführten Liste der unterstützten Ausdrücke entspricht.

sonderregister
Der Name eines unterstützten Sonderregisters. Diese Einstellung gibt den Wert des Sonderregisters für den aktuellen Server an. Die Dokumentation zu Sonderregistern enthält eine vollständige Liste der unterstützten Sonderregister.

CAST (ausdruck AS datentyp)
Der in den angegebenen SQL-Datentyp umgesetzte Ausdruck, falls der Ausdruck nicht NULL ist. Ist der Ausdruck NULL, ist das Ergebnis ein Nullwert mit dem angegebenen SQL-Datentyp. Beim Einfügen eines NULL-Werts in eine Spalte muss der Ausdruck NULL in einen kompatiblen Spaltentyp umsetzen (z. B. CAST (NULL AS INTEGER) für eine Spalte mit dem Typ 'Integer').

XMLCAST (ausdruck AS datentyp)
Der in den angegebenen Datentyp umgesetzte Ausdruck, falls der Ausdruck nicht NULL ist. Der Ausdruck oder der Zieldatentyp muss der XML-Typ sein. Ist der Ausdruck NULL, muss als Zieltyp XML verwendet werden. Als Ergebnis wird ein XML-Nullwert ermittelt.

XML-funktion
Eine beliebige unterstützte SQL/XML-Funktion.

Beispiel

Das folgende Beispiel zeigt, wie die Annotation db2-xdb:expression dazu verwendet werden kann, einen Wert aus dem XML-Dokument für eine benutzerdefinierte Funktion anzuwenden. Das von der benutzerdefinierten Funktion zurückgegebene Ergebnis wird dann in die Datenbank eingefügt, nicht der Wert aus dem Dokument selbst. Zunächst ist ein Abschnitt aus dem mit Annotationen versehenen Schema dargestellt.

```
<xs:element name="author">
            <xs:complexType>
  <xs:sequence>
            <xs:element name="firstname" type="xs:string" />
            <xs:element name="lastname" type="xs:string" />
            <xs:element name="activeStatus" type="xs:boolean" />
                <xs:attribute name="ID" type="xs:integer"
                  db2-xdb:rowSet="AUTHORS" db2-xdb:column="NUMBOOKS"
                  db2-xdb:expression="AuthNumBooks (INTEGER ($DECOMP_CONTENT))" />
  </xs:sequence>
</xs:complexType>
 </xs:element>
```

Angenommen, eine benutzerdefinierte Funktion mit der Bezeichnung 'AuthNumBooks' akzeptiert einen Parameter mit dem Typ 'Integer', der die ID des Autors darstellt, und gibt die Gesamtzahl der Bücher zurück, über die der Autor im System verfügt.

Nun wird das Element <author>, für das eine Zuordnung vorgenommen wird, dargestellt.

```
<author ID="22">
      <firstname>Ann</firstname>
    <lastname>Brown</lastname>
      <activeStatus>1</activeStatus>
  </author>
```

$DECOMP_CONTENT wird durch den Wert "22" aus der Instanz des ID-Attributs ersetzt. Da $DECOMP_CONTENT stets durch einen Zeichentyp ersetzt wird und da die benutzerdefinierte Funktion 'AuthNumBooks' einen Parameter mit dem Typ 'Integer' benötigt, muss die Annotation db2-xdb:expression $DECOMP_CONTENT in den Typ 'Integer' umsetzen. Angenommen, die benutzerdefinierte Funktion gibt die Zahl 8 für diesen Autor zurück, dessen ID 22 ist, dann wird 8 in die Spalte NUMBOOKS der Tabelle AUTHORS eingefügt, wie nachfolgend dargestellt.

Tabelle 51. AUTHORS

AUTHID	FIRSTNAME	SURNAME	ACTIVE	NUMBOOKS
NULL	NULL	NULL	NULL	8

Dekompositionsannotation db2-xdb:condition

Die Annotation db2-xdb:condition gibt eine Bedingung an, die festlegt, ob eine Zeile in eine Tabelle eingefügt wird. Eine Zeile, die die Bedingung erfüllt, kann eingefügt werden (gegebenenfalls abhängig von anderen Bedingungen für die Zeilengruppe); eine Zeile, die die Bedingung nicht erfüllt, wird nicht eingefügt.

db2-xdb:condition gehört zur Gruppe der Dekompositionsannotationen, die einem XML-Schemadokument hinzugefügt werden können, um die Zuordnung zwischen Elementen und Attributen von XML-Dokumenten zu DB2-Basistabellen zu beschreiben. Der Dekompositionsprozess verwendet das mit Annotationen versehene XML-Schema, um festzustellen, wie die Elemente und Attribute eines XML-Dokuments zerlegt und in DB2-Tabellen eingefügt werden sollen.

Annotationstyp

Attribut von <xs:element> oder <xs:attribute> oder optionales untergeordnetes Element von <db2-xdb:rowSetMapping>. Die Bedingung wird angewendet, unabhängig davon, ob die Annotation, zu der sie gehört, eine Spaltenzuordnung enthält.

Angabe

db2-xdb:condition kann auf eine der nachfolgend beschriebenen Arten angegeben werden (dabei stellt *wert* einen gültigen Wert für die Annotation dar):

- `<xs:element db2-xdb:condition="wert" />`
- `<xs:attribute db2-xdb:condition="wert" />`
- ```
  <db2-xdb:rowSetMapping>
    <db2-xdb:rowSet>wert</db2-xdb:rowSet>
    <db2-xdb:condition>wert</db2-xdb:condition>
    ...
  </db2-xdb:rowSetMapping>
  ```

Namensbereich

http://www.ibm.com/xmlns/prod/db2/xdb1

Gültige Werte

SQL-Vergleichselemente des folgenden Typs: basic, quantified, BETWEEN, EXISTS, IN, IS VALIDATED, LIKE, NULL und XMLEXISTS. Die Vergleichselemente müssen ebenfalls aus Ausdrücken bestehen, die von der Annotation db2-xdb:expression, Spaltennamen oder beidem unterstützt werden.
  ```

Details

Die Annotation db2-xdb:condition ermöglicht es, Bedingungen anzugeben, unter denen bei der Dekomposition Werte in die Datenbank eingefügt werden. Diese Annotation filtert Zeilen durch die Anwendung benutzerdefinierter Bedingungen. Die Zeilen, die die angegebenen Bedingungen erfüllen, werden in die Datenbank eingefügt; Zeilen, die die Bedingungen nicht erfüllen, werden bei der Dekomposition nicht eingefügt.

Wenn die Annotation db2-xdb:condition für mehrere Element- oder Attributdeklarationen derselben Zeilengruppe angegeben wird, wird die Zeile nur dann eingefügt, wenn die logische AND-Verknüpfung aller Bedingungen "true" ergibt.

Spaltennamen in db2-xdb:condition

Da db2-xdb:condition aus SQL-Vergleichselementen besteht, können in dieser Annotation Spaltennamen angegeben werden. Wenn eine Annotation db2-xdb:condition, die sich auf eine Zeilengruppe bezieht, einen nicht qualifizierten Spaltennamen enthält, muss unter allen Zuordnungen für diese Zeilengruppe eine Zuordnung zu dieser Spalte vorhanden sein. Andere Spaltennamen, die in Vergleichselementen mit SELECT-Anweisungen verwendet werden, müssen qualifiziert sein. Wenn db2-xdb:condition einen nicht qualifizierten Spaltennamen angibt, jedoch für das Element oder Attribut, für das db2-xdb:condition angegeben wird, keine Spaltenzuordnung festgelegt ist, ist bei der Auswertung der Bedingung der ausgewertete Wert der Inhalt des Elements bzw. Attributs, das dem referenzierten Spaltennamen zugeordnet ist.

Betrachten Sie das folgende Beispiel:

```
<xs:element name="a" type="xs:string"
            db2-xdb:rowSet="rowSetA" db2-xdb:condition="columnX='abc'" />
<xs:element name="b" type="xs:string"
            db2-xdb:rowSet="rowSetB" db2-xdb:column="columnX" />
```

In diesem Beispiel ist für <a> keine Spaltenzuordnung angegeben, doch in der Bedingung wird auf die Spalte "columnX" verwiesen. Bei der Auswertung der Bedingung wird "columnX" in der Bedingung durch den Wert von <b> ersetzt, da <b> eine Spaltenzuordnung für "columnX" angibt, wogegen <a> keine Spaltenzuordnung aufweist. Wenn das XML-Dokument Folgendes enthält:

```
<a>abc</a>
<b>def</b>
```

ergibt die Bedingung in diesem Fall "false", da der Wert von <b>, "def", in der Bedingung ausgewertet wird.

Wenn $DECOMP_CONTENT (ein Dekompositionsschlüsselwort, das den Wert des zugeordneten Elements oder Attributs als Zeichendaten angibt) anstelle des Spaltennamens in der Annotation db2-xdb:condition verwendet wird, die der Elementdeklaration <a> zugeordnet ist, wird die Bedingung unter Verwendung des Werts <a>, nicht <b>, ausgewertet.

```
<xs:element name="a" type="xs:string"
            db2-xdb:rowSet="rowSetA" db2-xdb:condition="$DECOMP_CONTENT='abc'" />
<xs:element name="b" type="xs:string"
            db2-xdb:rowSet="rowSetB" db2-xdb:column="columnX" />
```

Wenn das XML-Dokument Folgendes enthält:

```
<a>abc</a>
<b>def</b>
```

ergibt die Bedingung in diesem Fall "true", da der Wert von <a>, "abc", in der Auswertung verwendet wird.

Diese bedingte Verarbeitung mit Spaltennamen und $DECOMP_CONTENT ist in den Fällen sinnvoll, in denen eine Dekomposition nur eines Wertes auf der Basis des Wertes eines anderen Elements bzw. Attributs durchgeführt werden soll, der nicht in die Datenbank eingefügt wird.

Bedingungen für zugeordnete Elemente oder Attribute, die nicht im Dokument enthalten sind

Wenn eine Bedingung für ein Element oder Attribut angegeben wird, das nicht im XML-Dokument enthalten ist, wird die Bedingung dennoch angewendet. Beispiel: Nachfolgend ist eine Elementzuordnung aus einem mit Annotationen versehenen Schemadokument dargestellt:

```
<xs:element name="intElem" type="xs:integer"
            db2-xdb:rowSet="rowSetA" db2-xdb:column="colInt"
            db2-xdb:condition="colInt > 100" default="0" />
```

Wenn das Element <intElem> nicht im XML-Dokument enthalten ist, wird die Bedingung "colInt > 100" dennoch ausgewertet. Da <intElem> nicht vorhanden ist, wird in der Bedingungsauswertung für "colInt" der Standardwert 0 verwendet. Die Bedingung wird somit als 0 > 100 ausgewertet, d. h. "false". Die entsprechende Zeile wird daher bei der Dekomposition nicht eingefügt.

Beispiel

Nachfolgend ist ein Element <author> eines XML-Dokuments dargestellt:

```
<author ID="0800">
    <firstname>Alexander</firstname>
    <lastname>Smith</lastname>
      <activeStatus>1</activeStatus>
  </author>
```

Abhängig von den durch db2-xdb:condition angegebenen Bedingungen werden die Werte dieses Elements <author> bei der Dekomposition in die Zieltabellen eingefügt oder nicht. Zwei Fälle sind nachfolgend dargestellt.

Alle Bedingungen erfüllt

Der folgende Abschnitt aus dem mit Annotationen versehenen Schema zum Element <author>, das im vorherigen Beispiel beschrieben ist, gibt an, dass für dieses Element nur dann eine Dekomposition durchgeführt werden soll, wenn die ID von <author> zwischen 1 und 999 liegt, die Elemente <firstname> und <lastname> nicht NULL sind und der Wert des Elements <activeStatus> gleich 1 ist.

```
  <xs:element name="author">
              <xs:complexType>
             <xs:sequence>
            <xs:element name="firstname" type="xs:string"
                  db2-xdb:rowSet="AUTHORS" db2-xdb:column="GIVENNAME"
                  db2-xdb:condition="$DECOMP_CONTENT IS NOT NULL" />
                  <xs:element name="lastname" type="xs:string"
                  db2-xdb:rowSet="AUTHORS" db2-xdb:column="SURNAME"
                  db2-xdb:condition="$DECOMP_CONTENT IS NOT NULL" />
            <xs:element name="activeStatus" type="xs:integer"
                  db2-xdb:rowSet="AUTHORS" db2-xdb:column="statusCode"
                  db2-xdb:condition="$DECOMP_CONTENT=1" />
                  <xs:attribute name="ID" type="xs:integer"
                  db2-xdb:rowSet="AUTHORS" db2-xdb:column="AUTHID"
```

```
                db2-xdb:condition="$DECOMP_CONTENT BETWEEN 1 and 999 />
            </xs:sequence>
</xs:complexType>
 </xs:element>
```

Da alle von db2-xdb:condition angegebenen Bedingungen durch die Werte im oben beschriebenen Beispielelement <author> erfüllt werden, wird die Tabelle AUTHORS mit den Daten aus dem Element <author> gefüllt.

Tabelle 52. AUTHORS

AUTHID	GIVENNAME	SURNAME	STATUSCODE	NUMBOOKS
0800	Alexander	Smith	1	NULL

Eine Bedingung nicht erfüllt

Das folgende, mit Annotationen versehene Schema gibt an, dass für das Element <author> nur dann eine Dekomposition durchgeführt werden soll, wenn die ID des Autors zwischen 1 und 100 liegt und die Elemente <firstname> und <lastname> nicht NULL sind.

```
<xs:element name="author">
            <xs:complexType>
           <xs:sequence>
          <xs:element name="firstname" type="xs:string"
                db2-xdb:rowSet="AUTHORS" db2-xdb:column="GIVENNAME"
                db2-xdb:condition="$DECOMP_CONTENT IS NOT NULL"/>
                <xs:element name="lastname" type="xs:string"
                db2-xdb:rowSet="AUTHORS" db2-xdb:column="SURNAME"
                db2-xdb:condition="$DECOMP_CONTENT IS NOT NULL"/>
          <xs:element name="activeStatus" type="xs:integer" />
                <xs:attribute name="ID" type="xs:integer"
                  db2-xdb:rowSet="AUTHORS" db2-xdb:column="AUTHID"
                  db2-xdb:condition="$DECOMP_CONTENT BETWEEN 1 and 100 />
           </xs:sequence>
</xs:complexType>
 </xs:element>
```

Obwohl die Elemente <firstname> und <lastname> des Beispielelements <author> die angegebenen Bedingungen erfüllen, gilt dies nicht für den Wert des ID-Attributs, daher wird die gesamte Zeile bei der Dekomposition nicht eingefügt. Ursache hierfür ist, dass die logische AND-Verknüpfung aller drei Bedingungen, die für die Tabelle AUTHORS angegeben ist, ausgewertet wird. In diesem Fall ist eine der Bedingungen "false", wodurch auch die logische AND-Verknüpfung als "false" ausgewertet wird; somit werden keine Zeilen eingefügt.

Dekompositionsannotation db2-xdb:contentHandling

Die Annotation db2-xdb:contentHandling gibt den Typ des Inhalts an, der bei der Dekomposition für ein Element mit komplexem oder einfachen Typ in eine Tabelle eingefügt wird.

db2-xdb:contentHandling gehört zur Gruppe der Dekompositionsannotationen, die einem XML-Schemadokument hinzugefügt werden können, um die Zuordnung zwischen Elementen und Attributen von XML-Dokumenten zu DB2-Basistabellen zu beschreiben. Der Dekompositionsprozess verwendet das mit Annotationen versehene XML-Schema, um festzustellen, wie die Elemente und Attribute eines XML-Dokuments zerlegt und in DB2-Tabellen eingefügt werden sollen.

Annotationstyp

Attribut von <xs:element> oder Attribut von <db2-xdb:rowSetMapping> für Elementdeklarationen mit komplexem oder einfachem Typ.

Angabe

db2-xdb:contentHandling kann auf eine der nachfolgend beschriebenen Arten angegeben werden (dabei stellt *wert* einen gültigen Wert für die Annotation dar):

- `<xs:element db2-xdb:contentHandling="wert" />`
- ```
  <db2-xdb:rowSetMapping db2-xdb:contentHandling="wert">
    <db2-xdb:rowSet>wert</db2-xdb:rowSet>
    ...
  </db2-xdb:rowSetMapping>
  ```

Namensbereich

http://www.ibm.com/xmlns/prod/db2/xdb1

Gültige Werte

Eines der folgenden Token (die Groß-/Kleinschreibung muss beachtet werden):

- text
- stringValue
- serializeSubtree

Details

Die Annotation db2-xdb:contentHandling, die als Attribut in der Deklaration eines XML-Elements angegeben wird, gibt an, welcher Wert bei der Dekomposition in die Tabellen und Spalten eingefügt werden soll, die durch db2-xdb:rowSet bzw. db2-xdb:column angegeben werden. Die drei gültigen Werte für db2-xdb:contentHandling lauten:

text

- Was wird eingefügt: Die Verknüpfung von Zeichendaten (einschließlich des Zeicheninhalts von CDATA-Abschnitten) in diesem Element.
- Was wird weggelassen: Die Kommentare und Verarbeitungsanweisungen dieses Elements, CDATA-Abschnittsbegrenzer ("<![CDATA[" "]]>") sowie die untergeordneten Elemente (Nachkommen) dieses Elements (einschließlich Tags und Inhalt).

stringValue

- Was wird eingefügt: Die Verknüpfung von Zeichendaten dieses Elements (einschließlich des Zeicheninhalts von CDATA-Abschnitten) mit den Zeichendaten in den untergeordneten Elementen (Nachkommen) dieses Elements in der Dokumentreihenfolge.
- Was wird weggelassen: Kommentare, Verarbeitungsanweisungen, CDATA-Abschnittsbegrenzer ("<![CDATA[" "]]>") sowie die Start- und Endtags der untergeordneten Elemente (Nachkommen) dieses Elements.

serializeSubtree

- Was wird eingefügt: Markup für alles zwischen den Start- und Endtags dieses Elements (einschließlich der Start- und Endtags dieses Elements). Hierzu gehören auch Kommentare, Verarbeitungsanweisungen und CDATA-Abschnittsbegrenzer ("<![CDATA[" "]]>").
  ```

- Was wird weggelassen: Nichts.
-

  Anmerkungen: Die eingefügte serialisierte Zeichenfolge ist möglicherweise nicht identisch mit dem entsprechenden Abschnitt im XML-Dokument; hierbei können die folgenden Faktoren eine Rolle spielen: im XML-Schema angegebene Standardwerte, die Erweiterung von Entitäten, die Reihenfolge der Attribute, die Leerzeichennormalisierung von Attributen und die Verarbeitung von CDATA-Abschnitten.

  Da es sich bei der serialisierten Zeichenfolge, die aus dieser Einstellung resultiert, um eine XML-Entität handelt, müssen bestimmte Codepageaspekte berücksichtigt werden. Wenn die Zielspalte einen Zeichendaten- oder Grafikdatentyp aufweist, wird das XML-Fragment in der Codepage der Datenbank eingefügt. Wenn eine solche Entität von einer Anwendung an einen XML-Prozessor übergeben wird, muss die Anwendung den Prozessor explizit über die Codierung der Entität informieren, da der Prozessor normalerweise keine Codierungen erkennen kann, bei denen es sich nicht um UTF-8 handelt. Weist die Zielspalte jedoch den Typ BLOB auf, wird die XML-Entität in UTF-8-Codierung eingefügt. In diesem Fall kann die XML-Entität an den XML-Prozessor übergeben werden, ohne dass eine Codierung angegeben werden muss.

Wenn eine XML-Elementdeklaration, die mit Annotationen für die Dekomposition versehen wurde, einen komplexen Typ aufweist und komplexe Inhalte umfasst, db2-xdb:contentHandling jedoch nicht angegeben wurde, dann entspricht das Standardverhalten der Einstellung "serializeSubtree". In allen anderen Fällen von mit Annotationen versehenen Elementdeklarationen entspricht das Standardverhalten, wenn db2-xdb:contentHandling nicht angegeben wurde, der Einstellung "stringValue".

Wenn für ein Element ein komplexer Typ deklariert wird und wenn dieses ein reines Elementmodell oder ein leeres Inhaltsmodell (d. h. das Attribut "mixed" der Elementdeklaration ist nicht auf "true" oder "1" gesetzt) aufweist, kann für db2-xdb:contentHandling nicht die Einstellung "text" verwendet werden.

Die Angabe der Annotation db2-xdb:contentHandling für ein Element hat keine Auswirkungen auf die Dekomposition der untergeordneten Elemente (Nachkommen) dieses Elements.

Die Einstellung für db2-xdb:contentHandling wirkt sich auf den Wert aus, der für $DECOMP_CONTENT in einer der Annotationen db2-xdb:expression oder db2-xdb:condition eingesetzt wird. Der eingesetzte Wert wird zunächst der Einstellung für db2-xdb:contentHandling entsprechend verarbeitet und anschließend zur Auswertung übergeben.

Wenn vor oder während der Dekomposition eine Prüfung durchgeführt wurde, sind die Entitäten des Inhalts, der von db2-xdb:contentHandling verarbeitet wird, bereits aufgelöst.

Beispiel

Das folgende Beispiel veranschaulicht, wie die verschiedenen Einstellungen der Annotation db2-xdb:contentHandling dazu verwendet werden können, unterschiedliche Ergebnisse in der Zieltabelle zu erzielen. Zunächst ist das mit Annotationen versehene Schema dargestellt, das zeigt, wie das Element <paragraph> mit

der Annotation db2-xdb:contentHandling versehen wird. (Das mit Annotationen versehene Schema wird nur einmal dargestellt, wobei für db2-xdb:contentHandling der Wert "text" definiert ist. Bei den nachfolgenden Beispielen in diesem Abschnitt wird dasselbe mit Annotationen versehene Schema zugrunde gelegt, der Unterschied besteht lediglich in dem Wert, der für db2-xdb:contentHandling festgelegt wird.)

```
<xs:schema>
    <xs:element name="books">
                <xs:complexType>
                    <xs:sequence>
          <xs:element name="book">
                      <xs:complexType>
                          <xs:sequence>
                      <xs:element name="authorID" type="xs:integer" />
                      <xs:element name="chapter" type="chapterType" maxOccurs="unbounded" />
                            </xs:sequence>
                  <xs:attribute name="isbn" type="xs:string"
                            db2-xdb:rowSet="BOOKCONTENTS" db2-xdb:column="ISBN" />
                  <xs:attribute name="title" type="xs:string" />
                        </xs:complexType>
                    </xs:element>
                      </xs:sequence>
                  </xs:complexType>
            </xs:element>

  <xs:complexType name="chapterType">
                  <xs:sequence>
          <xs:element name="paragraph" type="paragraphType" maxOccurs="unbounded"
                  db2-xdb:rowSet="BOOKCONTENTS" db2-xdb:column="CHPTCONTENT"
                  db2-xdb:contentHandling="text" />
                  </xs:sequence>
      <xs:attribute name="number" type="xs:integer"
                  db2-xdb:rowSet="BOOKCONTENTS" db2-xdb:column="CHPTNUM" />
      <xs:attribute name="title" type="xs:string"
                  db2-xdb:rowSet="BOOKCONTENTS" db2-xdb:column="CHPTTITLE" />
              </xs:complexType>

    <xs:complexType name="paragraphType" mixed="1">
          <xs:choice>
            <xs:element name="b" type="xs:string" minOccurs="0" maxOccurs="unbounded" />
    </xs:choice>
              </xs:complexType>
</xs:schema>
```

Nun wird das Element <books>, für das eine Zuordnung vorgenommen wird, dargestellt.

```
<books>
  <book isbn="1-11-111111-1" title="My First XML Book">
      <authorID>22</authorID>
      <chapter number="1" title="Introduction to XML">
            <paragraph>XML is <b>lots of</b> fun...</paragraph>
      </chapter>
      <chapter number="2" title="XML and Databases">
            <paragraph><!-- Start of chapter -->XML can be used with...</paragraph>
      <paragraph><?processInstr example?>
                Escapezeichen wie <![CDATA[ <, > und & ]]>...</paragraph>
      </chapter>
    ...
      <chapter number="10" title="Further Reading">
          <paragraph>Recommended tutorials...</paragraph>
      </chapter>
    </book>
  ...
<books>
```

Die folgenden drei Tabellen zeigen das Ergebnis der Dekomposition desselben XML-Elements mit anderen Werten für db2-xdb:contentHandling.

Anmerkung: Die folgenden Tabellen enthalten Anführungszeichen vor und nach den Werten in den Spalten CHPTTITLE und CHPTCONTENT. Diese Anführungszeichen tauchen in den Spalten nicht auf, sondern werden nur hier dargestellt, um die Begrenzungen und Leerzeichen der eingefügten Zeichenfolgen zu verdeutlichen.

db2-xdb:contentHandling="text"

Tabelle 53. BOOKCONTENTS

ISBN	CHPTNUM	CHPTTITLE	CHPTCONTENT
1-11-111111-1	1	"Introduction to XML"	"XML is fun..."
1-11-111111-1	2	"XML and Databases"	"XML can be used with..."
1-11-111111-1	2	"XML and Databases"	" Escapezeichen wie <, > und & ..."
...	...	...	...
1-11-111111-1	10	"Further Reading"	"Recommended tutorials..."

Wie Sie erkennen können, wird der Inhalt des Elements <b> im ersten Absatz von Kapitel 1 nicht eingefügt, wenn die Einstellung "text" verwendet wird. Ursache hierfür ist, dass die Einstellung "text" den Inhalt von untergeordneten Elementen (Nachkommen) ausschließt. Darüber hinaus werden der Kommentar und die Verarbeitungsanweisung des ersten Absatzes von Kapitel 2 weggelassen, wenn die Einstellung "text" verwendet wird. Leerzeichen aus der Verknüpfung von Zeichendaten der <paragraph>-Elemente werden beibehalten.

db2-xdb:contentHandling="stringValue"

Tabelle 54. BOOKCONTENTS

ISBN	CHPTNUM	CHPTTITLE	CHPTCONTENT
1-11-111111-1	1	"Introduction to XML"	"XML is lots of fun..."
1-11-111111-1	2	"XML and Databases"	"XML can be used with..."
1-11-111111-1	2	"XML and Databases"	" Escapezeichen wie <, > und & ..."
...	...	...	...
1-11-111111-1	10	"Further Reading"	"Recommended tutorials..."

Der Unterschied zwischen dieser Tabelle und der vorhergehenden findet sich in der Spalte CHPTCONTENT der ersten Zeile. Die Zeichenfolge "lots of", die aus dem untergeordneten Element <b> des Elements <paragraph> stammt, wurde ein-

gefügt. Als für db2-xdb:contentHandling die Einstellung "text" definiert wurde, wurde diese Zeichenfolge weggelassen, da die Einstellung "text" den Inhalt von untergeordneten Elementen (Nachkommen) ausschließt. Die Einstellung "stringValue" schließt jedoch den Inhalt aus untergeordneten Elementen ein. Wie bei der Einstellung "text" werden Kommentare und Verarbeitungsanweisungen nicht eingefügt, und die Leerzeichen werden beibehalten.

db2-xdb:contentHandling="serializeSubtree"

Tabelle 55. BOOKCONTENTS

ISBN	CHPTNUM	CHPTTITLE	CHPTCONTENT
1-11-111111-1	1	"Introduction to XML"	"<paragraph>XML is lots of fun...</paragraph>"
1-11-111111-1	2	"XML and Databases"	"<paragraph><!-- Start of chapter -->XML can be used with...</paragraph>"
1-11-111111-1	2	"XML and Databases"	"<paragraph><?processInstr example?> Escapezeichen wie <![CDATA[<, > und &]]>...</paragraph>"
...	...	...	...
1-11-111111-1	10	"Further Reading"	"<paragraph>Recommended tutorials...</paragraph>"

Der Unterschied zwischen dieser Tabelle und den beiden vorhergehenden Tabellen besteht darin, dass die gesamte Markupformatierung der untergeordneten Elemente (Nachkommen) der <paragraph>-Elemente eingefügt wurde (einschließlich der Start- und Endtags von <paragraph>). Hierzu gehören auch die Start- und Endtags von in der Spalte CHPTCONTENT der ersten Zeile sowie der Kommentar und die Verarbeitungsanweisung in der zweiten bzw. dritten Zeile. Wie in den beiden vorhergehenden Beispielen werden die Leerzeichen aus dem XML-Dokument beibehalten.

Dekompositionsannotation db2-xdb:normalization

Die Annotation db2-xdb:normalization gibt die Normalisierung von Leerzeichen in den XML-Daten an, die eingefügt oder für $DECOMP_CONTENT eingesetzt werden sollen (bei Verwendung mit db2-xdb:expression).

db2-xdb:normalization gehört zur Gruppe der Dekompositionsannotationen, die einem XML-Schemadokument hinzugefügt werden können, um die Zuordnung zwischen Elementen und Attributen von XML-Dokumenten zu DB2-Basistabellen zu beschreiben. Der Dekompositionsprozess verwendet das mit Annotationen versehene XML-Schema, um festzustellen, wie die Elemente und Attribute eines XML-Dokuments zerlegt und in DB2-Tabellen eingefügt werden sollen.

Annotationstyp

Attribut <xs:element> oder <xs:attribute> oder Attribut <db2-xdb:rowSetMapping>

Angabe

db2-xdb:normalization kann auf eine der nachfolgend beschriebenen Arten angegeben werden (dabei stellt *wert* einen gültigen Wert für die Annotation dar):

- `<xs:element db2-xdb:normalization="wert" />`
- `<xs:attribute db2-xdb:normalization="wert" />`
- ```
  <db2-xdb:rowSetMapping db2-xdb:normalization="wert">
    <db2-xdb:rowSet>wert</db2-xdb:rowSet>
    ...
  </db2-xdb:rowSetMapping>
  ```

Namensbereich

http://www.ibm.com/xmlns/prod/db2/xdb1

Gültige Werte

Eines der folgenden Token (die Groß-/Kleinschreibung muss beachtet werden):

- canonical
- original (Standardwert)
- whitespaceStrip

Anmerkung: Das Attribut db2-xdb:normalization ist nur für Zuordnungen zwischen bestimmten XML-Schematypen und SQL-Zeichentypen gültig. Eine Liste der unterstützten XML-Schematypen, die für SQL-Zeichendatenspalten normalisiert werden können, finden Sie im Abschnitt 'Details'.

Details

Beim Einfügen von XML-Werten in Zielspalten mit Zeichendatentypen (CHAR, VARCHAR, LONG VARCHAR, CLOB, DBCLOB, GRAPHIC, VARGRAPHIC, LONG VARGRAPHIC) ist es möglicherweise erforderlich, die einzufügenden Daten zu normalisieren. Bei der Annotation db2-xdb:normalization können unterschiedliche Arten der Normalisierung angegeben werden. Die gültigen Werte, bei denen die Groß-/Kleinschreibung beachtet werden muss, sind nachfolgend aufgeführt.

canonical
: Der XML-Wert wird seinem XML-Schematyp entsprechend in sein kanonisches Format konvertiert, bevor er in die Zielspalte eingefügt oder für Vorkommen von $DECOMP_CONTENT in derselben Zuordnung wie die jeweilige Annotation db2-xdb:normalization eingesetzt wird.

original
: Die ursprünglichen Zeichendaten (ggf. nach der Verarbeitung durch einen XML-Parser) des Elementinhalts oder Attributwerts (abhängig davon, ob sich diese Zuordnung auf ein XML-Element oder ein XML-Attribut bezieht) werden in die Zielspalte eingefügt oder für Vorkommen von $DECOMP_CONTENT in derselben Zuordnung wie die jeweilige Annotation db2-xdb:normalization eingesetzt. Wenn das Attribut db2-xdb:normalization für eine Zuordnung, für die diese Annotation relevant ist, nicht angegeben wird, normalisiert der Dekompositionsprozess die Daten der Einstellung "original" entsprechend.

whitespaceStrip
: Aus dem XML-Wert werden alle führenden und abschließenden Leerzeichen entfernt, und aufeinanderfolgende Leerzeichen werden zu einem ein-
  ```

zelnen Leerzeichen komprimiert, bevor der Wert in die Zielspalte eingefügt oder für Vorkommen von $DECOMP_CONTENT in derselben Zuordnung wie die jeweilige Annotation db2-xdb:normalization eingesetzt wird.

db2-xdb:normalization ist anwendbar, wenn ein Element oder Attribut, das einen der nachfolgend aufgeführten atomaren XML-Schematypen aufweist (oder von diesem abgeleitet ist), einer Spalte mit einem Zeichendatentyp (CHAR, VARCHAR, LONG VARCHAR, CLOB, DBCLOB, GRAPHIC, VARGRAPHIC und LONG VARGRAPHIC) zugeordnet wird.

- byte, unsigned byte
- integer, positiveInteger, negativeInteger, nonPositiveInteger, nonNegativeInteger
- int, unsignedInt
- long, unsignedLong
- short, unsignedShort
- decimal
- float
- double
- boolean
- time
- date
- dateTime

db2-xdb:normalization wird ignoriert, wenn es für einen anderen Typ angegeben wird. Bitte beachten Sie, dass es sich hierbei um XML-Schematypen handelt, für die die W3C-Empfehlung (XML Schema Part 2: Datatypes Second Edition) eine kanonische Darstellung angibt.

Da die Annotation db2-xdb:normalization nur für bestimmte Zuordnungen von XML-Schematypen zu SQL-Zeichendatentypen gültig ist, wird sie ignoriert, wenn sie für nicht unterstützte Zuordnungen angegeben wird.

Beispiel

Das folgende Beispiel zeigt, wie die Normalisierung von Leerzeichen mithilfe der Annotation db2-xdb:normalization gesteuert werden kann. Zunächst ist das mit Annotationen versehene Schema dargestellt.

```
<xs:element name="author">
            <xs:complexType>
  <xs:sequence>
          <xs:element name="firstname" type="xs:string"
                db2-xdb:rowSet="AUTHORS" db2-xdb:column="FIRSTNAME" />
                <xs:element name="lastname" type="xs:string"
                db2-xdb:rowSet="AUTHORS" db2-xdb:column="SURNAME"
                db2-xdb:normalization="whitespaceStrip" />
                <xs:element name="activeStatus" type="xs:boolean"
                db2-xdb:rowSet="AUTHORS" db2-xdb:column="ACTIVE"
                db2-xdb:normalization="canonical" />
                <xs:attribute name="ID" type="xs:integer"
                db2-xdb:rowSet="AUTHORS" db2-xdb:column="AUTHID"
                db2-xdb:normalization="whitespaceStrip" />
</xs:sequence>
          </xs:complexType>
      </xs:element>
```

Das Element <author>, für das die Zuordnung vorgenommen wird, wird als Nächstes dargestellt (im folgenden Beispiel werden relevante Leerzeichen zu De-

monstrationszwecken durch das Unterstreichungszeichen '_' dargestellt), gefolgt von der daraus resultierenden Tabelle AUTHORS nach Abschluss der Dekomposition.

```
<author ID="__22">
       <firstname>Ann</firstname>
   <lastname>__Brown_</lastname>
       <activeStatus>1</activeStatus>
   </author>
```

Tabelle 56. AUTHORS

AUTHID	FIRSTNAME	SURNAME	ACTIVE	NUMBOOKS
22	Ann	__Brown_	true	NULL

Die Einstellung "whitespaceStrip" bewirkt, dass das führende Leerzeichen des Attributs "ID" entfernt wird, bevor der Wert in die Zieltabelle eingefügt wird. Dagegen werden das führende und das abschließende Leerzeichen des Elements <lastname> nicht entfernt, obwohl, die Einstellung "whitespaceStrip" angegeben wurde. Der Grund hierfür ist, dass das Element <lastname> den XML-Schematyp "string" aufweist, bei dem es sich nicht um einen gültigen Typ für db2-xdb:normalization handelt. Das untergeordnete Element <activeStatus> von <author> ist als Boolescher Typ definiert, und die kanonische Darstellung von Booleschen Typen ist das Literal "true" oder "false". Die Einstellung "canonical" für das Element <activeStatus> bewirkt, dass die kanonische Form von "1", also "true", in die Spalte ACTIVE der Tabelle AUTHORS eingefügt wird.

Wenn im oben dargestellten XML-Schema das Attribut "ID" stattdessen mit db2-xdb:normalization="original" annotiert worden wäre, wäre der ursprüngliche Wert aus dem Dokument, also "__22", (wobei das Unterstreichungszeichen für ein Leerzeichen steht) in die Spalte AUTHID eingefügt worden.

Dekompositionsannotation db2-xdb:order

Die Annotation db2-xdb:order gibt die Reihenfolge an, in der Zeilen jeweils in verschiedenen Tabellen eingefügt werden.

db2-xdb:order gehört zur Gruppe der Dekompositionsannotationen, die einem XML-Schemadokument hinzugefügt werden können, um die Zuordnung zwischen Elementen und Attributen von XML-Dokumenten zu DB2-Basistabellen zu beschreiben. Der Dekompositionsprozess verwendet das mit Annotationen versehene XML-Schema, um festzustellen, wie die Elemente und Attribute eines XML-Dokuments zerlegt und in DB2-Tabellen eingefügt werden sollen.

Annotationstyp

Untergeordnetes Element von <db2-xdb:rowSetOperationOrder>

Angabe

db2-xdb:order wird wie folgt angegeben (dabei stellt *wert* einen gültigen Wert für die Annotation dar):

```
<xs:schema>
          <xs:annotation>
                       <xs:appinfo>
      <db2-xdb:rowSetOperationOrder>
        <db2-xdb:order>
          <db2-xdb:rowSet>wert</db2-xdb:rowSet>
          ...
```

```
        </db2-xdb:order>
      </db2-xdb:rowSetOperationOrder>
                        </xs:appinfo>
          </xs:annotation>
  ...
</xs:schema>
```

Namensbereich

http://www.ibm.com/xmlns/prod/db2/xdb1

Gültige Struktur

Das folgende Element wird als untergeordnetes Element von <db2-xdb:order> unterstützt:

db2-xdb:rowSet
Gibt eine XML-Element- oder XML-Attributzuordnung zu einer Zielbasistabelle an.

Details

Die Annotation db2-xdb:order wird verwendet, um die Reihenfolge beim Einfügen der Zeilen, die zu einer bestimmten Zeilengruppe (rowSet) gehören, in Abhängigkeit von den Zeilen, die zu einer anderen Zeilengruppe gehören, zu definieren. Auf diese Weise können XML-Daten so in Zieltabellen eingefügt werden, dass sämtliche referenziellen Integritätsbedingungen erfüllt werden, die als Teil des relationalen Schemas für die Tabellen definiert sind.

Alle Zeilen einer bestimmten Zeilengruppe (rowSet) RS1 werden vor allen Zeilen eingefügt, die zu einer anderen Zeilengruppe RS2 gehören, wenn RS1 in db2-xdb:order vor RS2 aufgelistet ist. Von diesem Element können mehrere Instanzen angegeben werden, um mehrere Hierarchien für die Reihenfolge beim Einfügen zu definieren. Die Zeilen von Zeilengruppen (rowSets), die in keinem Element vorkommen, können in beliebiger Reihenfolge in Bezug auf die Zeilen anderer Zeilengruppen eingefügt werden. Auch muss der Inhalt jedes Elements vom Typ <db2-xdb:rowSet> entweder eine explizit definierte Zeilengruppe sein oder der Name einer vorhandenen Tabelle, für die keine explizite Zeilengruppendeklaration vorgenommen wurde.

Es können mehrere Einfügehierarchien für Zeilengruppen definiert werden, wenngleich eine bestimmte Zeilengruppe jeweils nur in einer einzigen Instanz von <db2-xdb:order> und dort auch nur einmal vorkommen darf.

Bei SQL-IDs mit Begrenzern, die in den untergeordneten Elementen angegeben sind, muss der Anführungszeichenbegrenzer im Zeichendateninhalt enthalten sein, und es brauchen hierfür keine Escapezeichen angegeben zu werden. Für die in SQL-IDs verwendeten Zeichen '&' und '<' müssen jedoch Escapezeichen angegeben werden.

Beispiel

Das folgende Beispiel zeigt die Verwendung der Annotation db2-xdb:order:

```
<xs:schema>
          <xs:annotation>
                        <xs:appinfo>
      <db2-xdb:rowSetOperationOrder>
```

```
        <db2-xdb:order>
          <db2-xdb:rowSet>CUSTOMER</db2-xdb:rowSet>
          <db2-xdb:rowSet>PURCHASE_ORDER</db2-xdb:rowSet>
        </db2-xdb:order>

        <db2-xdb:order>
          <db2-xdb:rowSet>ITEMS_MASTER</db2-xdb:rowSet>
          <db2-xdb:rowSet>PO_ITEMS</db2-xdb:rowSet>
        </db2-xdb:order>

      </db2-xdb:rowSetOperationOrder>
                    </xs:appinfo>
        </xs:annotation>
</xs:schema>
```

In diesem Beispiel sind zwei nicht schneidende Hierarchien für die Reihenfolge beim Einfügen angegeben. Die erste Hierarchie gibt an, dass der gesamte Inhalt der Zeilengruppe bzw. Tabelle CUSTOMER vor dem für PURCHASE_ORDER erfassten Inhalt eingefügt wird, und die zweite Hierarchie gibt an, dass der gesamte Inhalt der Zeilengruppe bzw. Tabelle ITEMS_MASTER eingefügt wird, bevor Inhalt in PO_ITEMS eingefügt wird. Hierbei ist zu beachten, dass die Reihenfolge zwischen den beiden Hierarchien untereinander nicht definiert ist. So kann beispielsweise jeder Inhalt für die Zeilengruppe bzw. Tabelle PURCHASE_ORDER eingefügt werden, bevor oder nachdem Inhalt in ITEMS_MASTER eingefügt wird.

Einschränkungen

Das Angeben einer Reihenfolge für Einfügungen in Zeilengruppen unterliegt den folgenden Einschränkungen:

- Bei 32-Bit-Systemen kann die Dekomposition großer Dokumente mit definierter Einfügereihenfolge zu einem Systemspeichermangel führen.
- Bei 64-Bit-Systemen kann es zu einem Speichermangel kommen, wenn der Administrator den den für einen Prozess zulässigen virtuellen Arbeitsspeicher beschränkt hat. Ein Speichermangel kann zwar vermieden werden, indem für Prozesse ein ausreichender oder unbegrenzter virtueller Arbeitsspeicher zugeordnet wird; dies wiederum kann jedoch negative Auswirkungen auf die Gesamtleistung des Systems haben.

Dekompositionsannotation db2-xdb:truncate

Die Annotation db2-xdb:truncate gibt an, ob beim Einfügen eines XML-Werts in eine Zeichenzielspalte das Abschneiden von Daten zulässig ist.

db2-xdb:truncate gehört zur Gruppe der Dekompositionsannotationen, die einem XML-Schemadokument hinzugefügt werden können, um die Zuordnung zwischen Elementen und Attributen von XML-Dokumenten zu DB2-Basistabellen zu beschreiben. Der Dekompositionsprozess verwendet das mit Annotationen versehene XML-Schema, um festzustellen, wie die Elemente und Attribute eines XML-Dokuments zerlegt und in DB2-Tabellen eingefügt werden sollen.

Annotationstyp

Attribut <xs:element> oder <xs:attribute> oder Attribut <db2-xdb:rowSetMapping>

Angabe

db2-xdb:truncate kann auf eine der nachfolgend beschriebenen Arten angegeben werden (dabei stellt *wert* einen gültigen Wert für die Annotation dar):

- `<xs:element db2-xdb:truncate="wert" />`
- `<xs:attribute db2-xdb:truncate="wert" />`
-
```
<db2-xdb:rowSetMapping db2-xdb:truncate="wert">
  <db2-xdb:rowSet>wert</db2-xdb:rowSet>
  ...
</db2-xdb:rowSetMapping>
```

Namensbereich

http://www.ibm.com/xmlns/prod/db2/xdb1

Gültige Werte

Eines der folgenden Token:

- 0 (entspricht 'false'; Standardwert)
- 1 (entspricht 'true')
- false (Groß-/Kleinschreibung muss beachtet werden; Standardwert)
- true (Groß-/Kleinschreibung muss beachtet werden)

Details

Ein XML-Wert, der in eine Zielspalte mit Zeichendaten eingefügt wird, überschreitet möglicherweise die Spaltengröße; in diesem Fall muss der Wert abgeschnitten werden, um die Dekomposition erfolgreich durchzuführen. Das Attribut db2-xdb:truncate gibt an, ob das Abschneiden der Daten zulässig ist, wenn der Wert für die Zielspalte zu groß ist. Wenn für dieses Attribut "false" oder "0" definiert wird, um anzugeben, dass das Abschneiden nicht zulässig ist, und der einzufügende XML-Wert für die Zielspalte zu groß ist, tritt bei der Dekomposition des XML-Dokuments ein Fehler auf, und der Wert wird nicht eingefügt. Die Definition "true" oder "1" gibt an, dass das Abschneiden der Daten während des Einfügens zulässig ist.

db2-xdb:truncate ist nur dann anwendbar, wenn für die Zielspalte eine der folgenden Bedingungen zutrifft:

- Sie weist einen Zeichentyp auf.
- Sie weist den Typ DATE, TIME oder TIMESTAMP auf, und der XML-Wert weist den Typ xs:date, xs:time oder xs:dateTime auf.

Wenn die Annotation db2-xdb:expression für dieselbe Element- oder Attributdeklaration angegeben wird wie db2-xdb:truncate, wird der Wert von db2-xdb:truncate ignoriert, da der Ausdruck bei entsprechender Definition das Abschneiden der Daten vornehmen kann.

Bei der Dekomposition in SQL-Spalten mit Datums-/Zeitwerten muss für XML-Werte, die eine Zeitzone angeben und den XML-Schematyp 'date', 'time' oder 'timestamp' aufweisen, db2-xdb:truncate auf "true" oder "1" gesetzt werden. Der Grund hierfür ist, dass die Struktur von SQL-Typen für Datums-/Zeitwerte die Angabe von Zeitzone nicht zulässt.

Beispiel

Das folgende Beispiel zeigt, wie das Abschneiden von Daten für ein Element <author> angewendet werden kann. Zunächst ist ein Abschnitt aus dem mit Annotationen versehenen Schema dargestellt.

```
<xs:element name="author">
            <xs:complexType>
  <xs:sequence>
          <xs:element name="firstname" type="xs:string"
                  db2-xdb:rowSet="AUTHORS" db2-xdb:column="FIRSTNAME"
                  db2-xdb:truncate="true" />
            <xs:element name="lastname" type="xs:string" />
            <xs:element name="activeStatus" type="xs:boolean" />
          <xs:element name="activated" type="xs:date"
                  db2-xdb:truncate="true" />
          <xs:attribute name="ID" type="xs:integer" />
  <xs:sequence>
</xs:complexType>
 </xs:element>
```

Nachfolgend wird das Element <author>, für das eine Zuordnung vorgenommen wird, dargestellt.

```
<author ID="0800">
    <firstname>Alexander</firstname>
    <lastname>Smith</lastname>
    <activeStatus>0</activeStatus>
    <activated>2001-10-31Z</activated>
  </author>
```

Angenommen, die Spalte FIRSTNAME wurde mit dem SQL-Typ CHAR und Größe 7 definiert und die Spalte ACTIVEDATE mit dem SQL-Typ DATE. Die Tabelle AUTHORS, die aus der Dekomposition resultiert, ist nachfolgend dargestellt.

Tabelle 57. AUTHORS

AUTHID	FIRSTNAME	SURNAME	ACTIVE	ACTIVEDATE	NUMBOOKS
NULL	Alexand	NULL	NULL	2001-10-31	NULL

Da der Wert "Alexander" für <firstname> die Größe der SQL-Spalte überschreitet, ist das Abschneiden der Daten erforderlich, damit der Wert eingefügt werden kann. Da das Element <activated> eine Zeitzone im XML-Dokument enthielt, wurde darüber hinaus für db2-xdb:truncate "true" definiert, um sicherzustellen, dass das Datum bei der Dekomposition erfolgreich eingefügt wurde.

Da für das Einfügen eines Werts aus dem Element <firstname> oder dem Element <activated> das Abschneiden von Daten erforderlich ist, wird, wenn für db2-xdb:truncate keine Angabe gemacht wird, der Standardwert für db2-xdb:truncate angenommen (kein Abschneiden zulässig), und es wird ein Fehler generiert, der angibt, dass die Zeile nicht eingefügt wurde.

Dekompositionsannotation db2-xdb:rowSetMapping

Die Annotation <db2-xdb:rowSetMapping> ordnet ein einzelnes XML-Element oder -Attribut einem oder mehreren Spalten-Tabellen-Paaren zu.

<db2-xdb:rowSetMapping> gehört zur Gruppe der Dekompositionsannotationen, die einem XML-Schemadokument hinzugefügt werden können, um die Zuordnung zwischen Elementen und Attributen von XML-Dokumenten zu DB2-Basistabellen zu beschreiben. Der Dekompositionsprozess verwendet das mit Annotationen versehene XML-Schema, um festzustellen, wie die Elemente und Attribute eines XML-Dokuments zerlegt und in DB2-Tabellen eingefügt werden sollen.

Annotationstyp

Untergeordnetes Element von <xs:appinfo> (ein untergeordnetes Element von <xs:annotation>), das ein untergeordnetes Element von <xs:element> oder <xs:attribute> ist.

Angabe

db2-xdb:rowSetMapping kann auf eine der nachfolgend beschriebenen Arten angegeben werden (dabei stellt *wert* einen gültigen Wert für die Annotation dar):

- ```
  <xs:element>
              <xs:annotation>
                          <xs:appinfo>
          <db2-xdb:rowSetMapping>
            <db2-xdb:rowSet>wert</db2-xdb:rowSet>
            ...
          </db2-xdb:rowSetMapping>
                          </xs:appinfo>
              </xs:annotation>
      ...
       </xs:element>
  ```
- ```
  <xs:attribute>
              <xs:annotation>
                          <xs:appinfo>
          <db2-xdb:rowSetMapping>
            <db2-xdb:rowSet>wert</db2-xdb:rowSet>
            ...
          </db2-xdb:rowSetMapping>
                          </xs:appinfo>
              </xs:annotation>
     ...
  </xs:attribute>
  ```

Namensbereich

http://www.ibm.com/xmlns/prod/db2/xdb1

Gültige Struktur

Gültige Attribute von <db2-xdb:rowSetMapping> sind nachfolgend aufgeführt:

db2-xdb:contentHandling
Ermöglicht die Angabe des Typs des Inhalts, der für ein Element mit komplexem Typ durch die Dekomposition in eine Tabelle eingefügt wird.

db2-xdb:locationPath
Ermöglicht die Zuordnung eines XML-Elements oder -Attributs, das als Teil einer wiederverwendbaren Gruppe deklariert wurde, zu unterschiedlichen Tabellen-Spalten-Paaren, abhängig von den übergeordneten Elementen (Vorfahren) des Elements bzw. Attributs.

db2-xdb:normalization
Ermöglicht die Angabe des Normalisierungsverhaltens für den Inhalt des XML-Elements oder -Attributs, das einer Zielspalte mit einem Zeichendatentyp zugeordnet wird, bevor der Inhalt eingefügt wird.

db2-xdb:truncate
Ermöglicht die Angabe, ob das Abschneiden von Daten zulässig ist, wenn ein XML-Wert in eine Zielspalte mit Zeichendatentyp eingefügt wird.

Diese Attribute von <db2-xdb:rowSetMapping> sind auch als Attribute von XML-Element- oder -Attributdeklarationen verfügbar; für diese Attribute gelten dieselben Verhaltensweisen und Voraussetzungen, unabhängig davon, ob sie Attribute von <db2-xdb:rowSetMapping> oder <xs:element> oder <xs:attribute> sind. Die jeweilige Dokumentation zu diesen Annotationen enthält Details hierzu.

Im Folgenden sind unterstützte untergeordnete Elemente von <db2-xdb:rowSetMapping> in der Reihenfolge aufgeführt, in der sie verwendet werden müssen, wenn sie angegeben werden:

<db2-xdb:rowSet>
Ordnet ein XML-Element oder -Attribut einer Zielbasistabelle zu.

<db2-xdb:column>
(Optional) Ordnet ein XML-Element oder -Attribut einer Basistabellenspalte zu. Dieses Element ist erforderlich, wenn db2-xdb:expression in der Annotation db2-xdb:rowSetMapping vorhanden ist.

<db2-xdb:column> kann in den Fällen optional sein, in denen ein Wert nicht in die Tabelle eingefügt werden soll, sondern nur für die bedingte Verarbeitung verwendet wird. Soll beispielsweise die Dekomposition für ein Element auf der Basis des Werts eines anderen Elements durchgeführt werden, ist für das andere Element keine Spaltenzuordnung erforderlich, da es nicht eingefügt wird.

<db2-xdb:expression>
(Optional) Gibt einen angepassten Ausdruck an, dessen Ergebnis in die Tabelle eingefügt wird, die durch das Attribut db2-xdb:rowSet angegeben wird.

Wenn $DECOMP_CONTENT in 'db2-xdb:expression' angegeben wird und 'db2-xdb:normalization' in derselben Zuordnung angegeben wird, wird der Wert von $DECOMP_CONTENT für 'db2-xdb:expression' normalisiert, bevor er ggf. zur Auswertung an den Ausdruck übergeben wird.

<db2-xdb:condition>
(Optional) Gibt eine Bedingung zur Auswertung an.

Diese untergeordneten Elemente von <db2-xdb:rowSetMapping> weisen dieselbe Semantik und Syntax auf wie die entsprechenden Attributannotationen, mit der Ausnahme, dass für Anführungszeichen keine Escapezeichen erforderlich sind.

Die entsprechende Dokumentation zu den Attributversionen dieser Annotationen enthält weitere Details hierzu.

Details

<db2-xdb:rowSetMapping> kann dazu verwendet werden, ein XML-Element oder -Attribut einer einzelnen Zieltabelle und -spalte, mehreren Zielspalten derselben Tabelle oder mehreren Tabellen und Spalten zuzuordnen. Für die Zuordnung zu einer einzelnen Tabelle und Spalte stehen zwei gleichwertige Methoden zur Verfügung: Die Kombination aus den Annotationen db2-xdb:rowSet und db2-xdb:column (bei denen es sich um Attribute des Elements bzw. Attributs handelt, für das die Zuordnung ausgeführt wird); oder die Angabe von <db2-xdb:rowSetMapping> (einem untergeordneten Element des Elements bzw. Attributs, für das die Zuordnung ausgeführt wird). Beide Methoden liefern dasselbe Ergebnis und unterscheiden sich nur in ihrer Notation.

Alle Leerzeichen im Zeichendateninhalt der untergeordneten Elemente von <db2-xdb:rowSetMapping> sind signifikant; es wird keine Leerzeichennormalisierung durchgeführt. Bei SQL-IDs mit Begrenzern, die in den untergeordneten Elementen angegeben sind, muss der Anführungszeichenbegrenzer im Zeichendateninhalt enthalten sein, es dürfen hierfür keine Escapezeichen angegeben werden. Für die in SQL-IDs verwendeten Zeichen '&' und '<' müssen jedoch Escapezeichen angegeben werden.

Beispiel

Das folgende Beispiel zeigt, wie ein einzelnes Attribut mit der Bezeichnung "isbn" mit der Annotation <db2-xdb:rowSetMapping> mehr als einer Tabelle zugeordnet werden kann. Zunächst wird ein Abschnitt aus dem mit Annotationen versehenen Schema dargestellt. Er zeigt, wie der ISBN-Wert sowohl der Tabelle BOOKS als auch der Tabelle BOOKCONTENTS zugeordnet wird.

```
<xs:element name="book">
            <xs:complexType>
  <xs:sequence>
          <xs:element name="authorID" type="xs:integer"/>
          <xs:element name="chapter" type="chapterType" maxOccurs="unbounded" />
  </xs:sequence>
      <xs:attribute name="isbn" type="xs:string">
    <xs:annotation>
                        <xs:appinfo>
        <db2-xdb:rowSetMapping>
          <db2-xdb:rowSet>BOOKS</db2-xdb:rowSet>
          <db2-xdb:column>ISBN</db2-xdb:column>
        </db2-xdb:rowSetMapping>
        <db2-xdb:rowSetMapping>
          <db2-xdb:rowSet>BOOKCONTENTS</db2-xdb:rowSet>
          <db2-xdb:column>ISBN</db2-xdb:column>
        </db2-xdb:rowSetMapping>
                        </xs:appinfo>
            </xs:annotation>
  </xs:attribute>
      <xs:attribute name="title" type="xs:string" />
</xs:complexType>
 </xs:element>
```

Nachfolgend werden das Element <book>, für das eine Zuordnung vorgenommen wird, und die daraus resultierenden Tabellen BOOKS und BOOKCONTENTS nach Abschluss der Dekomposition dargestellt.

```
<book isbn="1-11-111111-1" title="My First XML Book">
    <authorID>22</authorID>
    <!-- this book does not have a preface -->
    <chapter number="1" title="Introduction to XML">
        <paragraph>XML is fun...</paragraph>
    ...
    </chapter>
  ...
  </book>
```

Tabelle 58. BOOKS

ISBN	TITLE	CONTENT
1-11-111111-1	NULL	NULL

Tabelle 59. BOOKCONTENTS

ISBN	CHPTNUM	CHPTTITLE	CHPTCONTENT
1-11-111111-1	NULL	NULL	NULL

Alternative Zuordnung mithilfe der Kombination aus <db2-xdb:rowSetMapping>, db2-xdb:rowSet und db2-xdb:column

Der folgende Abschnitt eines mit Annotationen versehenen Schemas entspricht dem oben dargestellten XML-Schemafragment insofern, als er dasselbe Dekompositionsergebnis liefert. Der Unterschied zwischen den beiden Schemata besteht darin, dass dieses Schema eine Zuordnung durch die Kombination aus db2-xdb:rowSet und db2-xdb:column ersetzt, anstatt nur die Annotation <db2-xdb:rowSetMapping> zu verwenden.

```
<xs:element name="book">
            <xs:complexType>
  <xs:sequence>
          <xs:element name="authorID" type="xs:integer"/>
          <xs:element name="chapter" type="chapterType" maxOccurs="unbounded" />
  </xs:sequence>
      <xs:attribute name="isbn" type="xs:string"
                  db2-xdb:rowSet="BOOKS" db2-xdb:column="ISBN" >
    <xs:annotation>
                        <xs:appinfo>
       <db2-xdb:rowSetMapping>
         <db2-xdb:rowSet>BOOKCONTENTS</db2-xdb:rowSet>
         <db2-xdb:column>ISBN</db2-xdb:column>
       </db2-xdb:rowSetMapping>
                        </xs:appinfo>
          </xs:annotation>
  </xs:attribute>
      <xs:attribute name="title" type="xs:string" />
</xs:complexType>
 </xs:element>
```

Dekompositionsannotation db2-xdb:rowSetOperationOrder

Die Annotation db2-xdb:rowSetOperationOrder fungiert als übergeordnetes Element für mindestens ein Element vom Typ db2-xdb:order. Der Abschnitt zum Element db2-xdb:order enthält Informationen dazu, wie es verwendet wird, um zu definieren, in welcher Reihenfolge Zeilen in verschiedene Tabellen jeweils eingefügt werden.

db2-xdb:rowSetOperationOrder gehört zur Gruppe der Dekompositionsannotationen, die einem XML-Schemadokument hinzugefügt werden können, um die Zuordnung zwischen Elementen und Attributen von XML-Dokumenten zu DB2-Basistabellen zu beschreiben. Der Dekompositionsprozess verwendet das mit Annotationen versehene XML-Schema, um festzustellen, wie die Elemente und Attribute eines XML-Dokuments zerlegt und in DB2-Tabellen eingefügt werden sollen.

Annotationstyp

Untergeordnetes Element von <xs:appinfo>, das ein untergeordnetes Element eines globalen Elements <xs:annotation> ist.

Angabe

db2-xdb:rowSetOperationOrder wird wie folgt angegeben (dabei stellt *wert* einen gültigen Wert für die Annotation dar):

```
<xs:schema>
         <xs:annotation>
                     <xs:appinfo>
```

```
      <db2-xdb:rowSetOperationOrder>
        <db2-xdb:order>
          <db2-xdb:rowSet>wert</db2-xdb:rowSet>
          ...
        </db2-xdb:order>
      </db2-xdb:rowSetOperationOrder>
                      </xs:appinfo>
        </xs:annotation>
  ...
</xs:schema>
```

Namensbereich

http://www.ibm.com/xmlns/prod/db2/xdb1

Gültige Struktur

Das folgende Element wird als untergeordnetes Element von <db2-xdb:rowSetOperationOrder> unterstützt:

db2-xdb:order

Details

<db2-xdb:rowSetOperationOrder> dient zur Gruppierung von Elementen vom Typ <db2-xdb:order>. Es können mehrere Instanzen des untergeordneten Elements <db2-xdb:order> vorhanden sein, sodass mehrere Hierarchien für das Einfügen definiert werden können.

Da Sie die Möglichkeit haben, die Reihenfolge zu steuern, in der Inhalte aus XML-Dokumenten eingefügt werden, stellen die Annotationen db2-xdb:rowSetOperationOrder und db2-xdb:order zusammen eine Möglichkeit dar, um sicherzustellen, dass während der XML-Schemadekomposition alle referenzielle Integritätsbedingungen für Zieltabellen erfüllt werden und auch alle anderen Anwendungsanforderungen eingehalten werden, die vorgeben, dass Zeilen einer bestimmten Tabelle vor den Zeilen einer anderen Tabelle eingefügt werden sollen.

Die Annotation db2-xdb:rowSetOperationOrder darf in einem XML-Schema nur einmal verwendet werden.

Beispiel

Der Abschnitt zur Annotation db2-xdb:order enthält Beispiele, die zeigen, wie die Reihenfolge beim Einfügen von Zeilengruppen (rowSet) angegeben wird.

Schlüsselwörter für die Dekomposition mithilfe eines mit Annotationen versehenen XML-Schemas

Die Dekomposition mithilfe eines mit Annotationen versehenen XML-Schemas stellt Dekompositionsschlüsselwörter bereit, die in den Annotationen 'db2-xdb:condition' und 'db2-xdb:expression' verwendet werden können.

$DECOMP_CONTENT

Der Wert des zugeordneten XML-Elements oder -Attributs aus dem Dokument, der entsprechend der Einstellung der Annotation 'db2-xdb:contentHandling' erstellt wird. Der Wert, der im Ausdruck für $DECOMP_CONTENT eingesetzt wird, sollte immer als Zeichentyp betrachtet werden. Informationen zur maximalen Zeichenfolgelänge und maximalen Anzahl von unterstützten $DECOMP_CONTENT-Instanzen finden Sie in der Do-

kumentation zu Begrenzungen und Einschränkungen. Wenn $DECOMP-_CONTENT in 'db2-xdb:expression' angegeben wird und 'db2-xdb:normalization' in derselben Zuordnung angegeben wird, wird der Wert von $DECOMP_CONTENT für 'db2-xdb:expression' normalisiert, bevor er zur Auswertung an den Ausdruck übergeben wird, falls möglich.

Mit $DECOMP_CONTENT kann der Wert eines zugeordneten Elements oder Attributs mithilfe angepasster Ausdrücke verarbeitet werden, anstatt ihn direkt in eine Tabelle einzufügen.

$DECOMP_DOCUMENTID
Der Zeichenfolgewert, der im Eingabeparameter *documentid* der gespeicherten Prozedur 'xdbDecompXML' angegeben wird und der das XML-Dokument für die Dekomposition identifiziert. Bei der Dekomposition des Dokuments wird der für die gespeicherte Prozedur 'xdbDecompXML' angegebene Eingabewert als der Wert verwendet, der für $DECOMP_DOCUMENTID eingesetzt wird.

Anwendungen können eindeutig generierte Dokument-IDs an die Prozedur 'xdbDecompXML' übergeben. Diese IDs können anschließend direkt in die Datenbank eingefügt werden. Die IDs können auch an Ausdrücke übergeben werden, die eindeutige Kennungen für Elemente oder Attribute generieren. Daher kann $DECOMP_DOCUMENTID dazu verwendet werden, eindeutige Kennungen einzufügen, die nicht im XML-Dokument vorhanden sind.

$DECOMP_ELEMENTID
Eine vom System generierte ganzzahlige ID, die das Element oder Attribut, das durch diese Annotation beschrieben wird, innerhalb des XML-Dokuments eindeutig identifiziert. Dieser Wert bleibt für dasselbe XML-Dokument zwischen Dekompositionsoperationen unverändert, solange das Dokument nicht auf eine der folgenden Arten geändert wird: Hinzufügen von Elementen, Löschen von Elementen oder Ändern der Position des Elements in der Reihenfolge des Dokument. Wenn das Dokument auf diese Weisen geändert und eine erneute Dekomposition ausgeführt wird, haben Elemente möglicherweise nicht dieselbe ID wie nach der vorherigen Dekomposition. Die Angabe von $DECOMP_ELEMENTID in einem Attribut ist als Wert von $DECOMP_ELEMENTID für das Element definiert, zu dem das betreffende Attribut gehört.

Der von $DECOMP_ELEMENTID generierte Wert kann auch zur Angabe der Reihenfolge von Elementen im Originaldokument verwendet werden. Dies kann in Fällen nützlich sein, in denen das XML-Dokument aus relationalen Tabellen wieder zusammengesetzt werden muss (Rekomposition).

Bildung von Dekompositionsergebnissen bei der Dekomposition mithilfe eines mit Annotationen versehenen XML-Schemas

Während Dekompositionsprozesse in der Regel nur Inhalte von XML-Elementen oder XML-Attributen zerlegen, unterstützt die Dekomposition mithilfe eines mit Annotationen versehenen XML-Schemas die Einfügung von Werten, die in dem XML-Dokument nicht unbedingt vorhanden sind.

Den zerlegten Inhalt können Datenwerte der folgenden Arten bilden:

- Der Wert eines Attributs im XML-Dokument
- Der Wert eines Elements im XML-Dokument, wobei der exakte Inhalt von der Einstellung der Annotation '<db2-xdb:contentHandling>' abhängig ist:

  - text - Zeichendaten nur aus diesem Element (nicht aus zugehörigen Nachkommen)
  - stringValue - Zeichendaten aus diesem Element und zugehörigen Nachkommen
  - serializedSubtree - Markup des gesamten Inhalts zwischen dem Anfangs- und Endbefehlen dieses Elements

  Weitere Informationen finden Sie in der Dokumentation zu '<db2-xdb:contentHandling>'.
- Ein Wert auf der Basis des Inhalts eines zugeordneten Attributs oder Elements im XML-Dokument
- Ein generierter Wert, der von allen Werten im XML-Dokument unabhängig ist

Die beiden letzteren Werte sind durch die Annotation 'db2-xdb:expression' realisierbar. Diese Annotation gibt Ihnen die Möglichkeit, einen Ausdruck anzugeben, dessen Ergebnis bei der Dekomposition eingefügt wird.

Der Wert aus einem XML-Dokument kann auf einen Ausdruck angewendet werden, um ein Ergebnis zu generieren und so die Daten umzuwandeln, bevor sie in die Zielspalte eingefügt werden. Ein Ausdruck kann darüber hinaus einen Wert generieren, der auf dem zugeordneten Element bzw. Attribut basiert (z. B. dem Namen des Elements). Die Annotation 'db2-xdb:expression' ermöglicht auch die Angabe von Konstanten, wobei die Konstante auf den zugeordneten Wert aus dem XML-Dokument bezogen sein kann oder auch nicht. Mit der Annotation 'db2-xdb:expression' können Sie beliebige dieser Techniken kombinieren, um einen Wert zum Einfügen zu generieren.

Beachten Sie, dass der Ausdruck so oft aufgerufen wird, wie das Element oder Attribut, dem er zugeordnet ist, im XML-Dokument vorhanden ist.

Auswirkung der Prüfung auf die Ergebnisse der XML-Dekomposition

Bei der Dekomposition mithilfe eines mit Annotationen versehenen XML-Schemas ist eine Prüfung der Eingabedokumente nicht erforderlich. Es wird jedoch empfohlen, vor oder während der Dekomposition eine solche Prüfung durchzuführen, weil sie verschiedene Vorteile bietet.

Sie können die Prüfung vor der Dekomposition (mithilfe der SQL/XML-Funktion XMLVALIDATE) durchführen oder während der Dekomposition als Teil des Aufrufs der gespeicherten Prozedur xdbDecompXML bzw. des Befehls DECOMPOSE XML DOCUMENT. Durch die Prüfung der XML-Dokumente für die Dekomposition wird Folgendes sichergestellt:

- Werte werden nur dann mithilfe der Dekomposition in Tabellen eingefügt, wenn das gesamte Dokument dem angegebenen XML-Schema entsprechend gültig ist (hierdurch wird sichergestellt, dass nur gültige Werte in der Datenbank gespeichert werden).
- Für ein Element oder Attribut definierte Standardwerte werden in die Datenbank eingefügt (bei der Prüfung mithilfe einer der gespeicherten xdbDecompXML-Dekompositionsprozeduren, wenn das Element oder Attribut nicht im XML-Dokument vorhanden ist).
- Alle Entitäten im XML-Dokument werden bei der Prüfung während der Dekomposition aufgelöst (falls eine Entität im XML-Dokument vor der Dekomposition nicht registriert wurde, wird ein Fehler zurückgegeben).

- Eine von der Standardeinstellung abweichende Leerzeichennormalisierung findet statt, wie im Schema angegeben (wenn die Prüfung mithilfe einer der gespeicherten xdbDecompXML-Dekompositionsprozeduren durchgeführt wird).

Die Prüfung der Eingabedokumente gegenüber dem registrierten XML-Schema wird empfohlen, da der Dekompositionsprozess davon ausgeht, dass die Eingabedokumente dem jeweiligen mit Annotationen versehenen Schema entsprechend gültig sind. Wenn keine Prüfung durchgeführt wird und die Eingabedokumente ungültig sind, werden durch die Dekomposition möglicherweise andere Zeilen für dasselbe Eingabedokument eingefügt (verglichen mit dem Ergebnis bei durchgeführter Prüfung, aufgrund der Auflösung von Entitäten oder des Hinzufügens von Standardattributen), oder die Dekomposition liefert unerwartete Ergebnisse. Die Ergebnisse der Dekomposition eines ungültigen Dokuments sowie die Auswirkungen auf vorhandene Daten sind nicht definiert.

Fehler im Schema, wie beispielsweise nicht deterministische Inhaltsmodelle oder fehlerhafte Typableitungen, können zum Fehlschlagen des Dekompositionsprozesses führen, wenn die Prüfung während der Dekomposition durchgeführt wird. Stellen Sie sicher, dass das mit Annotationen versehene Schema korrekt ist, und registrieren Sie das Schema erneut, bevor Sie die Dekomposition wiederholen.

Behandlung von CDATA-Abschnitten bei der Dekomposition mithilfe eines mit Annotationen versehenen XML-Schemas

Der Inhalt von CDATA-Abschnitten wird für Elemente, die mit Annotationen zur Dekomposition versehen sind, in die Datenbank eingefügt. Die CDATA-Abschnittsbegrenzer ("<![CDATA[" und "]]>") können ebenfalls eingefügt werden. Der CDATA-Inhalt unterliegt der Zeilenendenormalisierung des XML-Parsers.

Ohne das Attribut db2-xdb:contentHandling wird der Inhalt von CDATA-Abschnitten für Elemente, die mit Annotationen zur Dekomposition versehen sind, in die Datenbank eingefügt. Die CDATA-Abschnittsbegrenzer werden nicht eingefügt.

Wenn die XML-Elementdeklaration im XML-Schema mit dem Attribut db2-xdb:contentHandling="serializeSubtree" annotiert ist, wird der CDATA-Abschnitt einschließlich der CDATA-Begrenzer bei der Dekomposition von nicht syntaktisch analysierten XML-Dokumenten eingefügt.

Unterschied der Dekompositionsergebnisse zwischen XML mit und ohne syntaktische Analyse

Bei der Dekomposition eines CDATA-Abschnitts treten Unterschiede auf, wenn die entsprechende Elementdeklaration mit dem Attribut db2-xdb:contentHandling="serializeSubtree" annotiert ist. Das Ergebnis hängt davon ab, ob das XML-Eingabedokument syntaktisch analysiertes XML enthält oder nicht. Beispiel: Ein XML-Dokument wird in einer CLOB-Spalte als nicht syntaktisch analysiertes XML und in einer XML-Spalte als syntaktisch analysiertes XML gespeichert.

Im Dekompositionsergebnis werden die Begrenzungen und der Originalinhalt aller CDATA-Abschnitte in dem Element beibehalten, wenn das XML-Eingabedokument aus einer Nicht-XML-Spalte stammt. Stammt das Eingabedokument aus einer XML-Spalte, werden die CDATA-Abschnitte in den Dekompositionsergebnissen nicht beibehalten. Beispiel: Ein XML-Dokument enthält das folgende Fragment:

```
<a> before cdata <![CDATA[ in cdata & <>]]> after cdata</a>
```

Die Dekomposition führt bei der Zuordnung der Spalte zum Element 'a' zu folgendem Ergebnis, wenn bei der Zuordnung zu Element 'a' db2-xdb:contentHandling="serializeSubtree" angegeben wird, falls das Dokument in einer CLOB-Spalte gespeichert wird:

```
<a>before cdata <![CDATA[ in cdata & <>]]> after cdata</a>
```

Wenn das XML-Dokument in einer XML-Spalte gespeichert wird, führt die Dekomposition dieses XML-Fragments zu folgendem Ergebnis:

```
<a> before cdata in cdata &amp; < &gt; after cdata</a>
```

Der ursprüngliche CDATA-Abschnitt wird immer dann nicht beibehalten, wenn das Eingabedokument bei einer Dekomposition aus einer XML-Spalte stammt. Die Nichtbeibehaltung des ursprünglichen CDATA-Abschnitts stellt zwar kein Problem für die Korrektheit dar, da die Alternative mit dem Original logisch gleichwertig ist, die Ausgabe unterscheidet sich jedoch möglicherweise von der erwarteten Ausgabe.

Nullwerte und leere Zeichenfolgen bei der Dekomposition mithilfe eines mit Annotationen versehenen XML-Schemas

Bei der Dekomposition mithilfe eines mit Annotationen versehenen XML-Schemas werden unter bestimmten Bedingungen Nullwerte oder leere Zeichenfolgen eingefügt.

XML-Elemente

In der folgenden Tabelle ist dargestellt, wann für Elemente im XML-Dokument eine leere Zeichenfolge oder ein Nullwert in die Datenbank eingefügt wird.

Tabelle 60. NULL-Verarbeitung für zugeordnete Elemente

Bedingung	Leere Zeichenfolge	NULL-Wert
Das Element ist im Dokument nicht vorhanden.		X
Das Element erfüllt alle der folgenden Bedingungen: • Es ist im Dokument vorhanden. • Es enthält das Attribut `xsi:nil="true"` oder `xsi:nil="1"` im Starttag.		X
Das Element erfüllt alle der folgenden Bedingungen: • Es ist im Dokument vorhanden und leer. • Es enthält das Attribut `xsi:nil="true"` oder `xsi:nil="1"` im Starttag nicht. • Es ist als Typ "list", Typ "union", komplexer Typ mit gemischtem Inhalt oder als einer der folgenden integrierten atomaren Typen deklariert bzw. von diesen abgeleitet: xsd:string, xsd:normalizedString, xsd:token, xsd:hexBinary, xsd:base64Binary, xsd:anyURI, xsd:anySimpleType; andere Typen führen zu einem Fehler.	X	

Tabelle 60. NULL-Verarbeitung für zugeordnete Elemente (Forts.)

Bedingung	Leere Zeichenfolge	NULL-Wert
Anmerkung: 1. Wenn eine Zuordnung die Annotation db2-xdb:condition oder db2-xdb:expression enthält, wird die leere Zeichenfolge oder der NULL-Wert (wie in der Tabelle dargestellt) als Argument für die Auswertung des Ausdrucks übergeben. 2. Wenn eine Zielspalte den Typ CHAR oder GRAPHIC aufweist, wird eine leere Zeichenfolge als Folge von Leerzeichen eingefügt.		

XML-Attribute

Die folgende Tabelle zeigt, wann eine leere Zeichenfolge bzw. ein NULL-Wert in die Datenbank eingefügt wird, falls die für die Dekomposition mit Annotationen versehenen XML-Attribute NULL-Werte im Dokument enthalten oder nicht vorhanden sind.

Tabelle 61. NULL-Verarbeitung für zugeordnete Attribute

Bedingung	Leere Zeichenfolge	NULL-Wert
Das Attribut ist im Dokument nicht vorhanden (entweder weil keine Prüfung durchgeführt wurde oder weil die Prüfung keinen Standardwert bereitgestellt hat).		X
Das Attribut erfüllt alle der folgenden Bedingungen: • Es ist im Dokument vorhanden und leer. • Es ist als Typ "list" bzw. "union" oder als einer der folgenden integrierten atomaren Typen deklariert bzw. von diesen abgeleitet: xsd:string, xsd:normalizedString, xsd:token, xsd:hexBinary, xsd:base64Binary, xsd:anyURI, xsd:anySimpleType; andere Typen führen zu einem Fehler.	X	
Anmerkung: Wenn eine Zuordnung die Annotation db2-xdb:condition oder db2-xdb:expression enthält, wird die leere Zeichenfolge oder der NULL-Wert (wie in der Tabelle dargestellt) als Argument für die Auswertung des Ausdrucks übergeben.		

Prüfliste für die Dekomposition mithilfe eines mit Annotationen versehenen XML-Schemas

Die Dekomposition mithilfe eines mit Annotationen versehenen XML-Schemas kann eine komplexe Task darstellen. Um diese Task einfacher bewerkstelligen zu können, sollten Sie verschiedene Aspekte berücksichtigen.

Bei der Dekomposition mithilfe eines mit Annotationen versehenen XML-Schemas müssen bei Bedarf mehrere XML-Elemente und -Attribute mehreren Spalten und Tabellen in der Datenbank zugeordnet werden. Diese Zuordnung kann außerdem die Konvertierung der XML-Daten vor dem Einfügen oder das Anwenden von Bedingungen für die Einfügung umfassen.

Die folgenden Punkte sind beim Annotieren eines XML-Schemas zu beachten (Verweise auf zugehörige Dokumentation sind ebenfalls aufgeführt):

- Informieren Sie sich darüber, welche Dekompositionsannotationen Ihnen zur Verfügung stehen.
- Stellen Sie bei der Zuordnung sicher, dass der Typ der Spalte mit dem XML-Schematyp des Elements bzw. Attributs kompatibel ist, dem sie zugeordnet wird.
- Strukturieren Sie das XML-Schema so, dass die Belastung der Systemspeicherressourcen auf ein Minimum beschränkt wird.
- Stellen Sie sicher, dass komplexe Typen, die durch Einschränkung oder Erweiterung abgeleitet wurden, korrekt mit Annotationen versehen werden.
- Vergewissern Sie sich, dass keine Grenzwerte und Einschränkungen für die Dekomposition verletzt werden.
- Stellen Sie sicher, dass die Tabellen und Spalten, auf die in der Annotation verwiesen wird, zu dem Zeitpunkt, an dem das Schema im XSR registriert wird, vorhanden sind.

Annotationen abgeleiteter komplexer Typen für die Dekomposition mithilfe eines mit Annotationen versehenen XML-Schemas

Bei der Annotation von komplexen Typen, die durch Einschränkung oder Erweiterung für die Dekomposition abgeleitet werden, müssen zusätzliche Zuordnungen angewendet werden.

Durch Einschränkung abgeleitet

Komplexe Typen, die durch Einschränkung (RESTRICTION) abgeleitet werden, erfordern eine Wiederholung der gemeinsamen Elemente und Attribute aus dem Basistyp in der Definition des abgeleiteten Typs. Die im Basistyp vorhandenen Dekompositionsannotationen müssen daher auch in den abgeleiteten Typ eingefügt werden.

Durch Erweiterung abgeleitet

In der Definition komplexer Typen, die durch Erweiterung (EXTENSION) abgeleitet werden, werden nur die Elemente und Attribute angegeben, die zusätzlich zum Basistyp vorhanden sein sollen. Wenn sich die Dekompositionszuordnungen für den abgeleiteten Typ von den Zuordnungen des Basistyps unterscheiden, müssen dem Basistyp Dekompositionsannotationen hinzugefügt werden, um die Zuordnungen des Basistyps klar von den abgeleiteten Typen zu unterscheiden.

Das folgende Beispiel zeigt, wie ein durch Erweiterung abgeleiteter Typ `outOfPrintBookType` einer anderen Tabelle als der zugehörige Basistyp `bookType` zugeordnet werden kann. Beachten Sie, wie die Annotation 'db2-xdb:locationPath' im Basistyp `bookType` angegeben wird, um klar zu unterscheiden, welche Zuordnungen für den Basistyp und welche für den abgeleiteten Typ gilt. Die Elemente '<lastPublished>' und '<publisher>' des abgeleiteten Typs `outOfPrintType` erfordern in diesem Beispiel keine Annotation 'db2-xdb:locationPath', da diese Elemente lediglich an einer einzigen Zuordnung beteiligt sind.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
                        xmlns:db2-xdb="http://www.ibm.com/xmlns/prod/db2/xdb1">
           <xs:annotation>
                            <xs:appinfo>
      <db2-xdb:table>
        <db2-xdb:name>BOOKS</db2-xdb:name>
        <db2-xdb:rowSet>inPrintRowSet</db2-xdb:rowSet>
      </db2-xdb:table>
```

```
    <db2-xdb:table>
      <db2-xdb:name>OUTOFPRINT</db2-xdb:name>
      <db2-xdb:rowSet>outOfPrintRowSet</db2-xdb:rowSet>
    </db2-xdb:table>
                    </xs:appinfo>
       </xs:annotation>
  <xs:element name="books">
              <xs:complexType>
         <xs:choice>
             <xs:element name="book" type="bookType"
                                      minOccurs="0" maxOccurs="unbounded"/>
             <xs:element name="outOfPrintBook" type="outOfPrintBookType"
                                      minOccurs="0" maxOccurs="unbounded"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
  <xs:complexType name="bookType">
               <xs:sequence>
         <xs:element name="authorID" type="xs:integer"/>
         <xs:element name="chapter" type="chapterType" maxOccurs="unbounded"/>
  </xs:sequence>
    <xs:attribute name="title" type="xs:string"
                db2-xdb:locationPath="/books/book/@title"
                db2-xdb:rowSet="inPrintRowSet" db2-xdb:column="TITLE">
           <xs:annotation>
                       <xs:appinfo>
       <db2-xdb:rowSetMapping db2-xdb:locationPath="/books/outOfPrintBook/@title">
         <db2-xdb:rowSet>outOfPrintRowSet</db2-xdb:rowSet>
         <db2-xdb:column>TITLE</db2-xdb:column>
       </db2-xdb:rowSetMapping>
                       </xs:appinfo>
           </xs:annotation>
  </xs:attribute>
      <xs:attribute name="isbn" type="xs:string"
                db2-xdb:locationPath="/books/book/@isbn"
                db2-xdb:rowSet="inPrintRowSet" db2-xdb:column="ISBN">
           <xs:annotation>
                       <xs:appinfo>
       <db2-xdb:rowSetMapping db2-xdb:locationPath="/books/outOfPrintBook/@isbn">
         <db2-xdb:rowSet>outOfPrintRowSet</db2-xdb:rowSet>
         <db2-xdb:column>ISBN</db2-xdb:column>
       </db2-xdb:rowSetMapping>
                       </xs:appinfo>
           </xs:annotation>
  </xs:attribute>
</xs:complexType>
  <xs:complexType name="outOfPrintBookType">
  <xs:complexContent>
         <xs:extension base="bookType">
                  <xs:sequence>
                <xs:element name="lastPublished" type="xs:date"
               db2-xdb:rowSet="outOfPrintRowSet" db2-xdb:column="LASTPUBDATE"/>
                <xs:element name="publisher" type="xs:string"
               db2-xdb:rowSet="outOfPrintRowSet" db2-xdb:column="PUBLISHER"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:simpleType name="paragraphType">
    <xs:restriction base="xs:string"/>
</xs:simpleType>
<xs:complexType name="chapterType">
               <xs:sequence>
        <xs:element name="paragraph" type="paragraphType" maxOccurs="unbounded"
                db2-xdb:locationPath="/books/book/chapter/paragraph"
                db2-xdb:rowSet="inPrintRowSet" db2-xdb:column="CONTENT">
             <xs:annotation>
```

```
                        <xs:appinfo>
            <db2-xdb:rowSetMapping
              db2-xdb:locationPath="/books/outOfPrintBook/chapter/paragraph">
              <db2-xdb:rowSet>outOfPrintBook</db2-xdb:rowSet>
              <db2-xdb:column>CONTENT</db2-xdb:column>
            </db2-xdb:rowSetMapping>
                        </xs:appinfo>
                  </xs:annotation>
      </xs:element>
    </xs:sequence>
        <xs:attribute name="number" type="xs:integer"/>
        <xs:attribute name="title" type="xs:string"/>
  </xs:complexType>
</xs:schema>
```

Die Annotationen geben an, dass Werte aus dem Element '<book>' durch die Dekomposition in die Tabelle BOOKS eingefügt werden, während Werte aus dem Element '<outOfPrintBook>' in die Tabelle OUTOFPRINT eingefügt werden.

Betrachten Sie das folgende Element aus einem XML-Dokument:

```
<books>
  <book isbn="1-11-111111-1" title="My First XML Book">
      <authorID>22</authorID>
      <chapter number="1" title="Introduction to XML">
          <paragraph>XML is fun...</paragraph>
      </chapter>
      <chapter number="2" title="XML and Databases">
          <paragraph>XML can be used with...</paragraph>
      </chapter>
    </book>
    <outOfPrintBook isbn="7-77-777777-7" title="Early XML Book">
        <authorID>41</authorID>
        <chapter number="1" title="Introductory XML">
            <paragraph>Early XML...</paragraph>
      </chapter>
        <chapter number="2" title="What is XML">
            <paragraph>XML is an emerging technology...</paragraph>
      </chapter>
        <lastPublished>2000-01-31</lastPublished>
        <publisher>Early Publishers Group</publisher>
    </outOfPrintBook>
</books>
```

Die folgenden Tabellen sind das Ergebnis der Dekomposition des Dokuments, zu dem dieses Element gehört, wobei das oben gezeigte, mit Annotationen versehene Schema verwendet wird:

Tabelle 62. BOOKS

ISBN	TITLE	CONTENT
1-11-111111-1	My First XML Book	XML is fun...
1-11-111111-1	My First XML Book	XML can be used with...

Tabelle 63. OUTOFPRINT

ISBN	TITLE	CONTENT	LASTPUBDATE	PUBLISHER
7-77-777777-7	Early XML Book	Early XML...	2000-01-31	Early Publishers Group
7-77-777777-7	Early XML Book	XML is an emerging technology...	2000-01-31	Early Publishers Group

Empfehlungen zur Strukturierung von XML-Schemata für die Dekomposition

Sie können die Belastung der Speicherressourcen Ihres Systems, die durch die Dekomposition mithilfe eines mit Annotationen versehenen XML-Schemas entsteht, auf ein Minimum reduzieren, indem Sie die Reihenfolge der Elemente im betreffenden XML-Schema entsprechend anpassen.

Bei sehr großen Dokumenten können diese Empfehlungen bewirken, dass die Dekomposition des Dokuments durchgeführt werden kann, ohne die Menge an verfügbarem Speicherplatz für den DB2-Datenbankserver erhöhen zu müssen. Bei gleichgeordneten Elementen, die für die Dekomposition annotiert werden, sollten Elemente mit einfachem Typ im mit Annotationen versehenen Schema vor den gleichgeordneten Elementen mit komplexem Typ positioniert werden. Ebenso sollten gleichgeordnete Elemente, für die das Attribut 'maxOccurs' auf 1 gesetzt ist, vor den gleichgeordneten Elementen positioniert werden, für die das Attribut 'maxOccurs' einen Wert > 1 aufweist.

Die für die Dekomposition mithilfe mit Annotationen versehener Schemata anfallende Speicherbelegung wird durch die Struktur des XML-Schemas beeinflusst, da jedes Element, das eine Zeile bildet, im Speicher verbleiben muss, bis alle Elemente, aus denen die Zeile besteht, verarbeitet sind. Bei diesen Empfehlungen zur Schemastrukturierung werden die Elemente einer Zeile so organisiert, dass die Anzahl der Elemente, die im Speicher verbleiben müssen, auf ein Minimum reduziert wird.

Das folgende Beispiel zeigt die empfohlene XML-Schemastrukturierung für die Zuordnung gleichgeordneter Elemente gegenüber einer nicht optimalen Strukturierung. Im Beispiel für die nicht optimale Strukturierung ist <complexElem>, das einen komplexen Typ aufweist, vor <status> positioniert, das einen einfachen Typ aufweist, Die Positionierung von <complexElem> nach den Elementen <id> und <status> verbessert die Laufzeiteffizienz der Dekomposition.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
                    xmlns:db2-xdb="http://www.ibm.com/xmlns/prod/db2/xdb1">
    <-- Empfohlene Strukturierung, bei der die einfachen Typen vor dem
            wiederholten Element <wrapper> (komplexer Typ) stehen -->
    <xs:complexType name="typeA">
    <xs:sequence>
            <xs:element name="id" type="xs:integer"
                  db2-xdb:rowSet="relA" db2-xdb:column="ID" />
            <xs:element name="status" type="xs:string"
                  db2-xdb:rowSet="relA" db2-xdb:column="status" />
            <xs:element name="wrapper" type="typeX" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

    <-- Nicht optimale Strukturierung, bei der das wiederholte Element
            (komplexer Typ) vor dem Element mit einfachem Typ steht -->
  <--
    <xs:complexType name="typeA">
    <xs:sequence>
            <xs:element name="id" type="xs:integer"
                  db2-xdb:rowSet="relA" db2-xdb:column="ID" />
            <xs:element name="wrapper" type="typeX" maxOccurs="unbounded"/>
            <xs:element name="status" type="xs:string"
                  db2-xdb:rowSet="relA" db2-xdb:column="status" />
    </xs:sequence>
    </xs:complexType> -->

    <xs:complexType name="typeX">
    <xs:sequence>
```

```
            <xs:element name="elem1" type="xs:string"
                  db2-xdb:rowSet="relA" db2-xdb:column="elem1" />
            <xs:element name="elem2" type="xs:long"
                  db2-xdb:rowSet="relA" db2-xdb:column="elem2" />
    </xs:sequence>
  </xs:complexType>

    <xs:element name="A" type="typeA" />

</xs:schema>
```

Beachten Sie, dass <id>, <status>, <elem1> und <elem2> derselben Zeilengruppe zugeordnet werden, d. h. zusammen eine Zeile bilden. Der Speicherplatz, der einer Zeile zugeordnet wird, wird freigegeben, wenn eine Zeile vollständig ist. Im oben dargestellten nicht optimalen Beispiel kann keine der Zeilen, die der Zeilengruppe 'relA' zugeordnet sind, als vollständig betrachtet werden, bis das Element <status> im Dokument erreicht ist. Das Element <wrapper> muss jedoch zuerst verarbeitet werden, da es vor dem Element <status> steht. Dies bedeutet, dass alle Instanzen von <wrapper> im Speicher gepuffert werden müssen, bis das Element <status> erreicht ist (bzw. bis das Ende von <A> erreicht ist, falls <status> nicht im Dokument enthalten ist).

Die Auswirkungen dieser Struktur werden deutlich, wenn eine große Anzahl von Instanzen eines Elements vorhanden sind. Wenn beispielsweise 10.000 Instanzen des Elements <wrapper> vorhanden sind, müssen alle 10.000 Instanzen im Speicher behalten werden, bis die Zeilengruppe verarbeitet ist. Im oben dargestellten optimalen Fall kann der Speicherplatz, der den Zeilen der Zeilengruppe 'relA' zugeordnet ist, jedoch freigegeben werden, wenn das Element <elem2> erreicht ist.

Beispiele für Zuordnungen bei der Dekomposition mithilfe eines mit Annotationen versehenen XML-Schemas

Die Dekomposition mithilfe eines mit Annotationen versehenen XML-Schemas verwendet Zuordnungen, um zu bestimmen, wie ein XML-Dokument in Tabellen zu zerlegen ist. Zuordnungen werden als Annotationen ausgedrückt, die einem XML-Schemadokument hinzugefügt werden. Diese Zuordnungen beschreiben, wie ein XML-Dokument in Tabellen zerlegt werden soll. Die folgenden Beispiele zeigen einige gängige Zuordnungsszenarios:

Gängige Zuordnungsszenarios:

Zeilengruppen in der Dekomposition mithilfe eines mit Annotationen versehenen XML-Schemas

Die Annotation 'db2-xdb:rowSet' gibt die Zieltabelle an, in die ein Wert durch die Dekomposition eingefügt wird. Diese Annotation kann entweder auf einen Tabellennamen oder einen Zeilengruppennamen (rowSet-Namen) gesetzt werden.

Eine Zeilengruppe (rowSet) wird mit der Annotation 'db2-xdb:rowSet' angegeben. Diese Annotation wird dem XML-Schemadokument entweder als Attribut einer Element- oder Attributdeklaration oder als untergeordnetes Element der Annotation <db2-xdb:rowSetMapping> hinzugefügt.

Die Gruppe von Zuordnungen (in allen Schemadokumenten, aus denen sich das XML-Schema zusammensetzt), die den gleichen 'db2-xdb:rowSet'-Wert für eine Instanz eines Elements oder Attributs hat, definiert eine Zeile.

Betrachten Sie zum Beispiel das folgende XML-Dokument:

```
<publications>
  <textbook title="Programming with XML">
    <isbn>0-11-011111-0</isbn>
    <author>Mary Brown</author>
    <author>Alex Page</author>
    <publicationDate>2002</publicationDate>
    <university>University of London</university>
  </textbook>
  <childrensbook title="Children's Fables">
    <isbn>5-55-555555-5</isbn>
    <author>Bob Carter</author>
    <author>Melaine Snowe</author>
    <publicationDate>1999</publicationDate>
  </childrensbook>
</publications>
```

Zur Dekomposition dieses Dokuments in der Weise, dass die ISBN-Nummer und der Titel jedes Buchs (sei es ein Lehrbuch oder ein Kinderbuch) in dieselbe Tabelle (mit dem Namen ALLPUBLICATIONS) eingefügt wird, müssen mehrere Zeilengruppen definiert werden: eine Zeilengruppe zur Gruppierung von Werten, die zu Lehrbüchern ('textbooks') gehören, und eine weitere zur Gruppierung von Werten, die zu Kinderbüchern ('childrensbook') gehören.

In diesem Fall stellen Zeilengruppen sicher, dass nur Werte, die semantisch zusammengehören, zusammengruppiert werden, um eine Zeile zu bilden. Das heißt, durch die Verwendung von Zeilengruppen wird der ISBN-Wert für ein Lehrbuch mit seinem Titel zusammengruppiert und der ISBN-Wert für ein Kinderbuch mit seinem Titel zusammengruppiert. Dies sorgt dafür, dass keine Zeile den ISBN-Wert eines Lehrbuchs und den Titel eines Kinderbuchs enthält.

Ohne Zeilengruppen lässt sich nicht bestimmen, welche Werte zusammengruppiert werden sollten, um eine Zeile zu bilden, die semantisch korrekt bleibt.

Die Anwendung von Zeilengruppen (rowSets) in einem XML-Schemadokument wird im Folgenden vorgestellt. Die beiden Zeilengruppen 'textbk_rowSet' und 'childrens_rowSet' werden jeweils in der Deklaration des Elements 'isbn' der Elements <textbook> bzw. <childrensbook> angegeben. Anschließend werden diese Zeilengruppen durch die Annotation '<db2-xdb:table>' der Tabelle ALLPUBLICATIONS zugeordnet.

Beachten Sie, dass die Verwendung der rowSet-Annotation zur Angabe nicht einer Tabelle, sondern einer Zeilengruppe, eine einfache Änderung von Tabellennamen ermöglicht, auf die in dem XML-Schema verwiesen wird. Wenn der Wert von 'db2-xdb:rowSet' jedoch einen Bezeichner und keinen Tabellennamen darstellt, müssen Sie die Annotation <db2-xdb:table><db2-xdb:name></db2-xdb:name></db2-xdb:table> zur eigentlichen Angabe des Tabellennamens verwenden. Mit dieser Methode brauchen Sie im Bedarfsfall den Tabellennamen nur an einer Stelle zu aktualisieren.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:db2-xdb="http://www.ibm.com/xmlns/prod/db2/xdb1"
           elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:annotation>
    <xs:appinfo>
      <db2-xdb:defaultSQLSchema>admin</db2-xdb:defaultSQLSchema>
      <db2-xdb:table>
        <db2-xdb:name>ALLPUBLICATIONS</db2-xdb:name>
        <db2-xdb:rowSet>textbk_rowSet</db2-xdb:rowSet>
        <db2-xdb:rowSet>childrens_rowSet</db2-xdb:rowSet>
      </db2-xdb:table>
```

```
    </xs:appinfo>
  </xs:annotation>
    <xs:element name="publications">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="textbook" maxOccurs="unbounded">
                       <xs:complexType>
              <xs:sequence>
                <xs:element name="isbn" type="xs:string"
                       db2-xdb:rowSet="textbk_rowSet" db2-xdb:column="PUBS_ISBN"/>
                <xs:element name="author" type="xs:string" maxOccurs="unbounded"/>
                <xs:element name="publicationDate" type="xs:gYear"/>
                <xs:element name="university" type="xs:string"
                             maxOccurs="unbounded"/>
              </xs:sequence>
              <xs:attribute name="title" type="xs:string" use="required"
                      db2-xdb:rowSet="textbk_rowSet" db2-xdb:column="PUBS_TITLE"/>
            </xs:complexType>
          </xs:element>
          <xs:element name="childrensbook" maxOccurs="unbounded">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="isbn" type="xs:string"
                    db2-xdb:rowSet="childrens_rowSet" db2-xdb:column="PUBS_ISBN"/>
                <xs:element name="author" type="xs:string" maxOccurs="unbounded"/>
                <xs:element name="publicationDate" type="xs:gYear"/>
              </xs:sequence>
              <xs:attribute name="title" type="xs:string" use="required"
                   db2-xdb:rowSet="childrens_rowSet" db2-xdb:column="PUBS_TITLE"/>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
                  </xs:complexType>
                   </xs:element>
</xs:schema>
```

Die aus der Dekomposition mithilfe dieses mit Annotationen versehenen XML-Schemas resultierende Tabelle sieht wie folgt aus:

Tabelle 64. ALLPUBLICATIONS

ISBN	PUBS_TITLE
0-11-011111-0	Programming with XML
5-55-555555-5	Children's Fables

Das oben dargestellte Beispiel zeigt den einfachen Fall einer Dekomposition mit Zeilengruppen. Zeilengruppen können jedoch auch in komplexeren Zuordnungen verwendet werden, um mehrere Elemente aus verschiedenen Teilen eines XML-Schemas zusammenzugruppieren und Zeilen für dasselbe Paar aus Tabelle und Spalte zu bilden.

Bedingte Umsetzungen

Zeilengruppen bieten die Möglichkeit, verschiedene Umsetzungen auf die Werte anzuwenden, die zerlegt werden, und zwar abhängig von den Werten selbst.

Betrachten Sie zum Beispiel die beiden folgenden Instanzen eines Elements mit dem Namen "temperature":

```
<temperature unit="Celsius">49</temperature>
<temperature unit="Fahrenheit">49</temperature>
```

Wenn die Werte dieser Elemente in dieselbe Tabelle eingefügt werden und die Tabelle anschließend konsistente Werte (zum Beispiel nur Celsius-Werte) enthalten soll, müssen Sie die Werte mit dem Attribut 'unit="Fahrenheit"' vor dem Einfügen in Celsius umwandeln. Dies können Sie erreichen, indem Sie alle Elemente mit dem Attribut 'unit="Celsius"' einer Zeilengruppe zuordnen und alle Elemente mit dem Attribut 'unit="Fahrenheit"' einer anderen Zeilengruppe zuordnen. Auf die Zeilengruppe für Celsius-Werte kann anschließend vor dem Einfügen eine Umrechnungsformel angewendet werden.

Bitte beachten Sie, dass die Zuordnung für die Attributdeklaration "unit" keine db2-xdb:column-Spezifikation enthält. Dies bedeutet, dass dieser Wert des Elements lediglich zur Auswertung von Bedingungen verwendet wird und nicht zum Speichern in der von der db2-xdb:rowSet-Spezifikation angegebenen Tabelle.

Mithilfe des folgenden XML-Schemadokuments könnten die Celsius- und umgerechneten Fahrenheit-Wert in dieselbe Tabelle eingefügt werden:

```
....
<!-- Global annotation -->
<db2-xdb:table>
  <db2-xdb:name>TEMPERATURE_DATA</db2-xdb:name>
  <db2-xdb:rowSet>temp_celsius</db2-xdb:rowSet>
  <db2-xdb:rowSet>temp_fahrenheit</db2-xdb:rowSet>
</db2-xdb:table>
...
<xs:element name="temperature">
  <xs:annotation>
    <xs:appinfo>
      <db2-xdb:rowSetMapping>
        <db2-xdb:rowSet>temp_celsius</db2-xdb:rowSet>
        <db2-xdb:column>col1</db2-xdb:column>
      </db2-xdb:rowSetMapping>
      <db2-xdb:rowSetMapping>
        <db2-xdb:rowSet>temp_fahrenheit</db2-xdb:rowSet>
        <db2-xdb:column>col1</db2-xdb:column>
        <db2-xdb:expression>
   myudf_convertTocelsius($DECOMP_CONTENT)
 </db2-xdb:expression>
      </db2-xdb:rowSetMapping>
    </xs:appinfo>
  </xs:annotation>
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:int">
        <xs:attribute name="unit" type="xs:string">
          <xs:annotation>
            <xs:appinfo>
              <db2-xdb:rowSetMapping>
                <db2-xdb:rowSet>temp_celsius</db2-xdb:rowSet>
                <db2-xdb:condition>
    $DECOMP_CONTENT = 'Celsius'
  </db2-xdb:condition>
              </db2-xdb:rowSetMapping>
              <db2-xdb:rowSetMapping>
                <db2-xdb:rowSet>temp_fahrenheit</db2-xdb:rowSet>
                <db2-xdb:condition>
    $DECOMP_CONTENT = 'fahrenheit'
  </db2-xdb:condition>
              </db2-xdb:rowSetMapping>
            </xs:appinfo>
          </xs:annotation>
        </xs:attribute>
```

```
    </xs:extension>
   </xs:simpleContent>
  </xs:complexType>
 </xs:element>
```

Annotationsbeispiel für die Dekomposition: Zuordnung zu einer XML-Spalte

Bei der Dekomposition mithilfe eines mit Annotationen versehenen XML-Schemas können Sie ein XML-Fragment einer Spalte zuordnen, die mit dem XML-Datentyp definiert wurde.

Betrachten Sie das folgende XML-Dokument:

```
<publications>
  <textbook title="Programming with XML">
    <isbn>0-11-011111-0</isbn>
    <author>Mary Brown</author>
    <author>Alex Page</author>
    <publicationDate>2002</publicationDate>
    <university>University of London</university>
  </textbook>
</publications>
```

Wenn Sie das XML-Element <textbook> und den Buchtitel wie folgt speichern wollen, müssen Sie im entsprechenden XML-Schemadokument Annotationen zu den Deklarationen des Elements <textbook> und des Titelattributs hinzufügen. Die Annotationen müssen die Spalten DETAILS und TITLE angeben, wobei die Spalte DETAILS mit dem XML-Typ definiert sein sollte, und außerdem die Tabelle TEXTBOOKS.

Tabelle 65. TEXTBOOKS

TITLE	DETAILS

Programming with XML	`<textbook title="Programming with XML">`
	`  <isbn>0-11-011111-0</isbn>`
	`  <author>Mary Brown</author>`
	`  <author>Alex Page</author>`
	`  <publicationDate>2002</publicationDate>`
	`  <university>University of London</university>`
	`</textbook>`

Abhängig von der jeweiligen Annotation kann diese im Schemadokument als Attribut oder als Element angegeben werden. Einige Annotationen können in beiden Formen angegeben werden. Informationen dazu, wie die einzelnen Annotationen angegeben werden können, finden Sie in der Dokumentation zur jeweiligen Annotation.

Geben Sie die Zieltabelle und die Zielspalte an, indem Sie entweder db2-xdb:rowSet und db2-xdb:column als Attribute von <xs:element> oder <xs:attribute> oder die untergeordneten Elemente <db2-xdb:rowSet> und <db2-xdb:column> von <db2-xdb:rowSetMapping> benutzen. Die Angabe dieser Zuordnungen als Elemente oder Attribute ist funktional gleichwertig.

Das folgende Fragment des XML-Schemadokuments zeigt, wie dem Element <textbook> und dem Titelattribut durch Angabe von Annotationen als Attribute zwei Zuordnungen hinzugefügt werden.

```
<xs:element name="publications">
  <xs:complexType>
    <xs:sequence>
```

```
      <xs:element name="textbook" maxOccurs="unbounded"
                  db2-xdb:rowSet="TEXTBOOKS" db2-xdb:column="DETAILS">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="isbn" type="xs:string"/>
            <xs:element name="author" type="xs:string" maxOccurs="unbounded"/>
            <xs:element name="publicationDate" type="xs:gYear"/>
            <xs:element name="university" type="xs:string" maxOccurs="unbounded"/>
          </xs:sequence>
         <xs:attribute name="title" type="xs:string" use="required"
                       db2-xdb:rowSet="TEXTBOOKS" db2-xdb:column="TITLE"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Die db2-xdb:rowSet-Annotationen geben den Namen der Zieltabelle und die db2-xdb:column-Annotationen den Namen der Zielspalte an. Weil das Element <textbook> einen komplexen Typ aufweist und komplexe Inhalte enthält, und die Annotation db2-xdb:contentHandling nicht angegeben wurde, werden standardmäßig alle Markup-Komponenten im Element (einschließlich des Start- und des Endetags) gemäß der serializeSubtree-Einstellung von db2-xdb:contentHandling in die XML-Spalte eingefügt. Leerzeichen im XML-Dokument werden beibehalten. Weitere Informationen hierzu finden Sie in der Dokumentation zu 'db2-xdb:contentHandling'.

Beispiel zur Dekomposition mit Annotationen: Ein einer einzigen Tabelle zugeordneter Wert, der eine einzige Zeile liefert

Die Zuordnung eines Werts aus einem XML-Dokument zu einem einzigen Paar aus Tabelle und Spalte ist eine einfache Zuordnungsform bei der Dekomposition mithilfe eines mit Annotationen versehenen XML-Schemas. Dieses Beispiel zeigt den einfacheren Fall einer Eins-zu-eins-Beziehung zwischen Werten in einer Zeilengruppe.

Das Ergebnis einer solchen Zuordnung hängt von der Beziehung zwischen Elementen ab, die derselben Zeilengruppe (rowSet) zugeordnet werden. Wenn die Werte, die zusammen einer einzigen Zeilengruppe (rowSet) zugeordnet werden, eine Eins-zu-eins-Beziehung haben, wie dies durch den Wert des Attributs 'maxOccurs' des Elements oder der übergeordneten Modellgruppendeklaration angegeben wird, wird nur eine Zeile für jede Instanz des zugeordneten Elements im XML-Dokument gebildet. Haben die Werte in einer einzigen Zeilengruppe eine Eins-zu-viele-Beziehung, in der ein Wert jeweils nur einmal im Dokument für viele Instanzen eines anderen Elements vorkommt, wie dies durch den Wert des Attributs 'maxOccurs' angegeben wird, resultieren daraus mehrere Zeilen, wenn das XML-Dokument zerlegt wird.

Betrachten Sie das folgende XML-Dokument:

```
<publications>
  <textbook title="Programming with XML">
    <isbn>0-11-011111-0</isbn>
    <author>Mary Brown</author>
    <author>Alex Page</author>
    <publicationDate>2002</publicationDate>
    <university>University of London</university>
  </textbook>
</publications>
```

Wenn die Werte der Elemente <isbn> und <publicationDate> sowie des Attributs 'title' wie folgt in die Tabelle TEXTBOOKS zerlegt werden sollen, müssen Sie den

Deklarationen für diese Elemente und Attribute im entsprechenden XML-Schemadokument Annotationen hinzufügen. Die Annotationen geben die Tabelle und die Spaltennamen an, denen die einzelnen Elemente zugeordnet werden.

Tabelle 66. TEXTBOOKS

ISBN	TITLE	DATE
0-11-011111-0	Programming with XML	2002

Abhängig von der jeweiligen Annotation kann diese im Schemadokument als Attribut oder als Element angegeben werden. Einige Annotationen können in beiden Formen angegeben werden. Informationen dazu, wie die einzelnen Annotationen angegeben werden können, finden Sie in der Dokumentation zur jeweiligen Annotation.

Für den Fall der Zuordnung eines Werts zu einem einzigen Paar aus Tabelle und Spalte müssen Sie die Tabelle und die Spalte für den zuzuordnenden Wert angeben. Dies geschieht mithilfe von 'db2-xdb:rowSet' und ' db2-xdb:column' als Attribute von <xs:element> oder <xs:attribute> oder mithilfe der untergeordneten Elemente '<db2-xdb:rowSet>' und '<db2-xdb:column>' von <db2-xdb:rowSetMapping>. Die Angabe dieser Zuordnungen als Elemente oder Attribute ist funktional gleichwertig.

Das folgende Beispiel zeigt, wie Elemente und Attribute aus dem Element <textbook> der Tabelle TEXTBOOKS durch die Angabe von Annotationen als Attribute zugeordnet werden.

```
<xs:element name="publications">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="textbook" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="isbn" type="xs:string"
                        db2-xdb:rowSet="TEXTBOOKS" db2-xdb:column="ISBN"/>
            <xs:element name="author" type="xs:string" maxOccurs="unbounded"/>
            <xs:element name="publicationDate" type="xs:gYear"
                        db2-xdb:rowSet="TEXTBOOKS" db2-xdb:column="DATE"/>
            <xs:element name="university" type="xs:string" maxOccurs="unbounded"/>
          </xs:sequence>
         <xs:attribute name="title" type="xs:string" use="required"
                       db2-xdb:rowSet="TEXTBOOKS" db2-xdb:column="TITLE"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
   </xs:element>
```

Das XML-Schemaattribut 'maxOccurs' hat den Standardwert 1, sodass jedes der Elemente, die der Zeilengruppe (rowSet) TEXTBOOKS zugeordnet werden, eine Eins-zu-eins-Beziehung zu den anderen hat. Aufgrund dieser Eins-zu-eins-Beziehung wird für jede Instanz des Elements <textbook> nur eine Zeile gebildet.

Beispiel zur Dekomposition mit Annotationen: Ein einer einzigen Tabelle zugeordneter Wert, der mehrere Zeilen liefert

Die Zuordnung eines Werts aus einem XML-Dokument zu einem einzigen Paar aus Tabelle und Spalte ist eine einfache Zuordnungsform bei der Dekomposition mit-

hilfe eines mit Annotationen versehenen XML-Schemas. Dieses Beispiel zeigt den etwas komplexeren Fall einer Eins-zu-viele-Beziehung zwischen Werten in einer Zeilengruppe (rowSet).

Das Ergebnis einer solchen Zuordnung hängt von der Beziehung zwischen Elementen ab, die derselben Zeilengruppe (rowSet) zugeordnet werden. Wenn die Werte, die zusammen einer einzigen Zeilengruppe (rowSet) zugeordnet werden, eine Eins-zu-eins-Beziehung haben, wie dies durch den Wert des Attributs 'maxOccurs' des Elements oder der übergeordneten Modellgruppendeklaration angegeben wird, wird nur eine Zeile für jede Instanz des zugeordneten Elements im XML-Dokument gebildet. Haben die Werte in einer einzigen Zeilengruppe eine Eins-zu-viele-Beziehung, in der ein Wert jeweils nur einmal im Dokument für viele Instanzen eines anderen Elements vorkommt, wie dies durch den Wert des Attributs 'maxOccurs' angegeben wird, resultieren daraus mehrere Zeilen, wenn das XML-Dokument zerlegt wird.

Betrachten Sie das folgende XML-Dokument:

```
<textbook title="Programming with XML">
  <isbn>0-11-011111-0</isbn>
  <author>Mary Brown</author>
  <author>Alex Page</author>
  <publicationDate>2002</publicationDate>
  <university>University of London</university>
</textbook>
```

Wenn Sie die ISBN-Nummer und die Autoren für ein Lehrbuch wie folgt speichern möchten, würden Sie den Deklarationen der Elemente <isbn> und <author> im entsprechenden XML-Schemadokument Annotationen hinzufügen. Die Annotationen sollten die Spalten ISBN und AUTHNAME sowie die Tabelle TEXTBOOK_AUTH angeben.

Tabelle 67. TEXTBOOKS_AUTH

ISBN	AUTHNAME
0-11-011111-0	Mary Brown
0-11-011111-0	Alex Page

Eine Annotation kann im Schemadokument entweder als Attribut oder als Element angegeben werden, je nachdem, um welche Annotation es sich handelt. Einige Annotationen können in beiden Formen angegeben werden. Informationen dazu, wie die einzelnen Annotationen angegeben werden können, finden Sie in der Dokumentation zur jeweiligen Annotation.

Für den Fall der Zuordnung eines Werts zu einem einzigen Paar aus Tabelle und Spalte müssen Sie die Tabelle und die Spalte für den zuzuordnenden Wert angeben. Dies geschieht mithilfe von 'db2-xdb:rowSet' und ' db2-xdb:column' als Attribute von <xs:element> oder <xs:attribute> oder mithilfe der untergeordneten Elemente '<db2-xdb:rowSet>' und '<db2-xdb:column>' von <db2-xdb:rowSetMapping>.

Die Angabe dieser Zuordnungen als Elemente oder Attribute ist äquivalent. Die Zuordnungen sind im nachfolgend dargestellten XML-Schemadokument als Elemente angegeben.

```
<xs:element name="textbook" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="isbn" type="xs:string">
```

```
        <xs:annotation>
          <xs:appinfo>
            <db2-xdb:rowSetMapping>
              <db2-xdb:rowSet>TEXTBOOKS_AUTH</db2-xdb:rowSet>
              <db2-xdb:column>ISBN</db2-xdb:column>
            </db2-xdb:rowSetMapping>
          </xs:appinfo>
        </xs:annotation>
      </xs:element>
      <xs:element name="author" type="xs:string" maxOccurs="unbounded">
        <xs:annotation>
          <xs:appinfo>
            <db2-xdb:rowSetMapping>
              <db2-xdb:rowSet>TEXTBOOKS_AUTH</db2-xdb:rowSet>
              <db2-xdb:column>AUTHNAME</db2-xdb:column>
            </db2-xdb:rowSetMapping>
          </xs:appinfo>
        </xs:annotation>
      </xs:element>
      <xs:element name="publicationDate" type="xs:gYear"/>
      <xs:element name="university" type="xs:string" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="title" type="xs:string" use="required"/>
  </xs:complexType>
   </xs:element>
```

Beachten Sie, dass das Element <isbn> nur einmal der Spalte ISBN zugeordnet wird, jedoch in zwei Zeilen der Tabelle auftritt. Dies geschieht beim Dekompositionsprozess automatisch, weil mehrere Autoren (Element 'author') pro ISBN-Wert vorhanden sind. Der Wert von <isbn> wird für jede Zeile mit einem Autor dupliziert.

Diese Funktionsweise resultiert daraus, dass eine Eins-zu-viele-Beziehung zwischen den Elementen <isbn> und <author> erkannt wird, da der Wert des Attributs 'maxOccurs' für <author> größer als 1 ist.

Beachten Sie, dass eine Eins-zu-viele Beziehung mehr als zwei Elemente betreffen und Gruppen von Elementen mit einbeziehen kann. Die Eins-zu-viele-Beziehung kann außerdem tief verschachtelt werden, wenn ein Element, das bereits an einer Eins-zu-viele-Beziehung beteiligt ist, in eine weitere Eins-zu-viele-Beziehung einbezogen wird.

Beispiel zur Dekomposition mit Annotationen: Ein mehreren Tabellen zugeordneter Wert

Ein Einzelwert aus einem XML-Dokument kann mehreren Tabellen zugeordnet werden. Dieses Beispiel zeigt, wie ein XML-Schemadokument mit Annotationen zu versehen ist, um einen einzelnen Wert zwei Tabellen zuzuordnen.

Betrachten Sie das folgende XML-Dokument:

```
<textbook title="Programming with XML">
      <isbn>0-11-011111-0</isbn>
    <author>Mary Brown</author>
    <author>Alex Page</author>
    <publicationDate>2002</publicationDate>
    <university>University of London</university>
</textbook>
```

Wenn Sie die ISBN-Nummer eines Lehrbuchs den beiden folgenden Tabellen zuordnen wollen, müssen Sie zwei Zuordnungen für das Element <isbn> erstellen.

Dies kann durch das Hinzufügen mehrerer <db2-xdb:rowSetMapping>-Elemente in der Deklaration des Elements <isbn> im XML-Schemadokument geschehen.

Tabelle 68. TEXTBOOKS

ISBN	TITLE
0-11-011111-0	Programming with XML

Tabelle 69. SCHOOLPUBS

ISBN	SCHOOL
0-11-011111-0	University of London

Das folgende Fragment des XML-Schemadokuments zeigt, wie der Deklaration des Elements <isbn> zwei Zuordnungen hinzugefügt werden, um die Zuordnungen zu zwei Tabellen anzugeben. Der Wert des Attributs 'title' und das Element <university> werden in die Zuordnungen ebenfalls mit einbezogen.

```
         <xs:element name="textbook" maxOccurs="unbounded">
             <xs:complexType>
   <xs:sequence>
     <xs:element name="isbn" type="xs:string">
       <xs:annotation>
                           <xs:appinfo>
           <db2-xdb:rowSetMapping>
             <db2-xdb:rowSet>TEXTBOOKS</db2-xdb:rowSet>
             <db2-xdb:column>ISBN</db2-xdb:column>
           </db2-xdb:rowSetMapping>
           <db2-xdb:rowSetMapping>
             <db2-xdb:rowSet>SCHOOLPUBS</db2-xdb:rowSet>
             <db2-xdb:column>ISBN</db2-xdb:column>
           </db2-xdb:rowSetMapping>
                           </xs:appinfo>
             </xs:annotation>
             </xs:element>
                   <xs:element name="author" type="xs:string" maxOccurs="unbounded"/>
                   <xs:element name="publicationDate" type="xs:gYear"/>
           <xs:element name="university" type="xs:string" maxOccurs="unbounded">
      <xs:annotation>
                           <xs:appinfo>
           <db2-xdb:rowSetMapping>
             <db2-xdb:rowSet>SCHOOLPUBS</db2-xdb:rowSet>
             <db2-xdb:column>SCHOOL</db2-xdb:column>
           </db2-xdb:rowSetMapping>
                           </xs:appinfo>
             </xs:annotation>
             </xs:element>
   </xs:sequence>
       <xs:attribute name="title" type="xs:string" use="required">
     <xs:annotation>
                        <xs:appinfo>
       <db2-xdb:rowSetMapping>
         <db2-xdb:rowSet>TEXTBOOKS</db2-xdb:rowSet>
         <db2-xdb:column>TITLE</db2-xdb:column>
       </db2-xdb:rowSetMapping>
                        </xs:appinfo>
         </xs:annotation>
   </xs:attribute>
 </xs:complexType>
 </xs:element>
```

Mehrfach auftretende komplexe Typen

Wenn auf einen komplexen Typ an mehreren Stellen in einem XML-Schema Bezug genommen wird, können Sie ihn abhängig von seiner Position im Schema mithilfe der Annotation 'db2-xdb:locationPath' verschiedenen Tabellen und Spalten zuordnen.

In diesem Fall muss die Element- bzw. Attributdeklaration des komplexen Typs mit mehreren Annotationen <db2-xdb:rowSetMapping> versehen werden (eine für jede Zuordnung), wobei jede Zuordnung durch das Attribut 'db2-xdb:locationPath' unterschieden wird.

Beispiel zur Dekomposition mit Annotationen: Gruppieren mehrerer Werte, die einer einzigen Tabelle zugeordnet werden

Bei der Dekomposition mithilfe eines mit Annotationen versehenen XML-Schemas können Sie mehrere Werte aus nicht zusammengehörigen Elementen derselben Tabelle zuordnen und gleichzeitig die Beibehaltung der Beziehung zwischen logisch zusammengehörigen Werten sicherstellen. Dies lässt sich durch die Deklaration mehrerer Zeilengruppen (rowSets) bewerkstelligen, die zur Gruppierung zusammengehöriger Elemente verwendet werden, um eine Zeile zu bilden (siehe vorliegendes Beispiel).

Betrachten Sie zum Beispiel das folgende XML-Dokument:

```
<publications>
  <textbook title="Programming with XML">
    <isbn>0-11-011111-0</isbn>
    <author>Mary Brown</author>
    <author>Alex Page</author>
    <publicationDate>2002</publicationDate>
    <university>University of London</university>
  </textbook>
  <childrensbook title="Children's Fables">
    <isbn>5-55-555555-5</isbn>
    <author>Bob Carter</author>
    <author>Melaine Snowe</author>
    <publicationDate>1999</publicationDate>
  </childrensbook>
</publications>
```

Zur Generierung der folgenden Tabelle nach der Dekomposition müssen Sie sicherstellen, dass Werte, die zu einem Lehrbuch ('textbook') gehören, nicht in die gleiche Zeile wie Werte gruppiert werden, die einem Kinderbuch ('childrensbook') zugeordnet sind. Verwenden Sie mehrere Zeilengruppen (rowSets), um zusammengehörige Werte zu gruppieren und logisch sinnvolle Zeilen zu erhalten.

Tabelle 70. ALLPUBLICATIONS

PUBS_ISBN	PUBS_TITLE
0-11-011111-0	Programming with XML
5-55-555555-5	Children's Fables

In einem einfachen Zuordnungsszenario, in dem Sie einen Wert einem einzigen Paar aus Tabelle und Spalte zuordnen, könnten Sie einfach die Tabelle und die Spalte angeben, denen der Wert zugeordnet werden soll.

Dieses Beispiel demonstriert jedoch einen etwas komplexeren Fall, in dem mehrere Werte der gleichen Tabelle zugeordnet werden und logisch gruppiert werden müs-

sen. Wenn Sie einfach jede ISBN-Nummer und jeden Titel ('title') den Spalten PUBS_ISBN und PUBS_TITLE ohne Zeilengruppen (rowSets) zuordnen würden, könnte der Dekompositionsprozess nicht bestimmen, welcher ISBN-Wert zu welchem TITLE-Wert gehört. Durch die Verwendung von Zeilengruppen können Sie zusammengehörige Werte logisch gruppieren, um eine sinnvolle Zeile zu bilden.

Das folgende XML-Schemadokument zeigt, wie zwei Zeilengruppen (rowSets) definiert werden, um Werte des Elements <textbook> von Werten des Elements <childrensbook> zu unterscheiden.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:db2-xdb="http://www.ibm.com/xmlns/prod/db2/xdb1"
           elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:annotation>
    <xs:appinfo>
      <db2-xdb:table>
        <db2-xdb:name>ALLPUBLICATIONS</db2-xdb:name>
        <db2-xdb:rowSet>textbk_rowSet</db2-xdb:rowSet>
        <db2-xdb:rowSet>childrens_rowSet</db2-xdb:rowSet>
      </db2-xdb:table>
    </xs:appinfo>
  </xs:annotation>
  <xs:element name="publications">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="textbook" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="isbn" type="xs:string"
                          db2-xdb:rowSet="textbk_rowSet" db2-xdb:column="PUBS_ISBN"/>
              <xs:element name="author" type="xs:string" maxOccurs="unbounded"/>
              <xs:element name="publicationDate" type="xs:gYear"/>
              <xs:element name="university" type="xs:string" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="title" type="xs:string" use="required"
                        db2-xdb:rowSet="textbk_rowSet" db2-xdb:column="PUBS_TITLE"/>
          </xs:complexType>
        </xs:element>
        <xs:element name="childrensbook" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="isbn" type="xs:string"
                     db2-xdb:rowSet="childrens_rowSet" db2-xdb:column="PUBS_ISBN"/>
              <xs:element name="author" type="xs:string" maxOccurs="unbounded"/>
              <xs:element name="publicationDate" type="xs:gYear"/>
            </xs:sequence>
            <xs:attribute name="title" type="xs:string" use="required"
                    db2-xdb:rowSet="childrens_rowSet" db2-xdb:column="PUBS_TITLE"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Beachten Sie, dass die 'db2-xdb:rowSet'-Zuordnungen in jeder der Element- und Attributdeklarationen nicht den Namen einer Tabelle, sondern den Namen einer Zeilengruppe (rowSet) angeben. Die Zeilengruppen werden der Tabelle ALLPUBLICATIONS in der Annotation <db2-xdb:table> zugeordnet, die als untergeordnetes Element des Elements <xs:schema> angegeben werden muss.

Durch die Angabe mehrerer Zeilengruppen, die eine Zuordnung zur selben Tabelle bewirken, können Sie sicherstellen, dass logisch zusammengehörige Werte eine Zeile in der Tabelle bilden.

Beispiel zur Dekomposition mit Annotationen: Mehrere Werte aus verschiedenen Kontexten, die einer einzigen Tabelle zugeordnet werden

Bei der Dekomposition mithilfe eines mit Annotationen versehenen XML-Schemas können Sie mehrere Werte derselben Tabelle und Spalte zuordnen, sodass eine einzige Spalte Werte enthalten kann, die aus verschiedenen Teilen eines Dokuments stammen. Dies lässt sich durch die Deklaration mehrerer Zeilengruppen (rowSets) bewerkstelligen (siehe vorliegendes Beispiel).

Betrachten Sie zum Beispiel das folgende XML-Dokument:

```
<publications>
  <textbook title="Principles of Mathematics">
    <isbn>1-11-111111-1</isbn>
    <author>Alice Braun</author>
    <publisher>Math Pubs</publisher>
    <publicationDate>2002</publicationDate>
    <university>University of London</university>
  </textbook>
</publications>
```

Sie können sowohl den Autor ('author') als auch den Verlag ('publisher') derselben Tabelle zuordnen, die Ansprechpartner für ein bestimmtes Buch enthält.

Tabelle 71. BOOKCONTACTS

ISBN	CONTACT
1-11-111111-1	Alice Braun
1-11-111111-1	Math Pubs

Die Werte der Spalte CONTACT der resultierenden Tabelle stammen aus verschiedenen Teilen des XML-Dokuments: Eine Zeile kann den Namen eines Autors (aus dem Element <author>) enthalten, während eine andere Zeile den Namen eines Verlags (aus dem Element <publisher>) enthält.

Das folgende XML-Schemadokument zeigt, wie mehrere Zeilengruppen (rowSets) zur Generierung dieser Tabelle verwendet werden können.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:db2-xdb="http://www.ibm.com/xmlns/prod/db2/xdb1"
           elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:annotation>
    <xs:appinfo>
      <db2-xdb:table>
        <db2-xdb:name>BOOKCONTACTS</db2-xdb:name>
        <db2-xdb:rowSet>author_rowSet</db2-xdb:rowSet>
        <db2-xdb:rowSet>publisher_rowSet</db2-xdb:rowSet>
      </db2-xdb:table>
    </xs:appinfo>
  </xs:annotation>
  <xs:element name="publications">
     <xs:complexType>
       <xs:sequence>
         <xs:element name="textbook" maxOccurs="unbounded">
           <xs:complexType>
             <xs:sequence>
               <xs:element name="isbn" type="xs:string">
                 <xs:annotation>
                   <xs:appinfo>
                     <db2-xdb:rowSetMapping>
                       <db2-xdb:rowSet>author_rowSet</db2-xdb:rowSet>
```

```
                <db2-xdb:column>ISBN</db2-xdb:column>
              </db2-xdb:rowSetMapping>
              <db2-xdb:rowSetMapping>
                <db2-xdb:rowSet>publisher_rowSet</db2-xdb:rowSet>
                <db2-xdb:column>ISBN</db2-xdb:column>
              </db2-xdb:rowSetMapping>
            </xs:appinfo>
          </xs:annotation>
        </xs:element>
        <xs:element name="author" type="xs:string" maxOccurs="unbounded">
          <xs:annotation>
            <xs:appinfo>
              <db2-xdb:rowSetMapping>
                <db2-xdb:rowSet>author_rowSet</db2-xdb:rowSet>
                <db2-xdb:column>CONTACT</db2-xdb:column>
              </db2-xdb:rowSetMapping>
            </xs:appinfo>
          </xs:annotation>
        </xs:element>
        <xs:element name="publisher" type="xs:string">
          <xs:annotation>
            <xs:appinfo>
              <db2-xdb:rowSetMapping>
                <db2-xdb:rowSet>publisher_rowSet</db2-xdb:rowSet>
                <db2-xdb:column>CONTACT</db2-xdb:column>
              </db2-xdb:rowSetMapping>
            </xs:appinfo>
          </xs:annotation>
        </xs:element>
        <xs:element name="publicationDate" type="xs:gYear"/>
        <xs:element name="university" type="xs:string"
                    maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="title" type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element>
 </xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

Beachten Sie, dass die 'db2-xdb:rowSet'-Zuordnungen in jeder der Elementdeklarationen nicht den Namen einer Tabelle, sondern den Namen einer Zeilengruppe (rowSet) angeben. Die Zeilengruppen werden der Tabelle BOOKCONTACTS in der Annotation <db2-xdb:table> zugeordnet, die als untergeordnetes Element des Elements <xs:schema> angegeben werden muss.

Kompatibilität von XML-Schematyp und SQL-Typ bei der Dekomposition mithilfe eines mit Annotationen versehenen XML-Schemas

Die Dekomposition mithilfe eines mit Annotationen versehenen XML-Schemas ermöglicht das Speichern von XML-Werten in Tabellenspalten. XML-Werte können nur für kompatible SQL-Spalten zerlegt werden. In der folgenden Tabelle wird aufgeführt, welche XML-Schematypen mit welchen SQL-Spaltentypen kompatibel sind.

Tabelle 72. Kompatible XML-Schematypen und SQL-Datentypen

XML-Schematyp	SQL SMALLINT	SQL INTEGER	SQL BIGINT	SQL REAL	SQL DECIMAL	SQL DOUBLE	SQL DECFLOAT (16)	SQL DECFLOAT (34)	SQL DATE	SQL TIME	SQL TIMESTAMP	SQL CHAR	SQL VARCHAR	SQL LONG VARCHAR	SQL CLOB	SQL GRAPHIC	SQL VARGRAPHIC	SQL LONG VARGRAPHIC	SQL DBCLOB	SQL CHAR FBD†	SQL BLOB	SQL VARCHAR FBD†	SQL LONG VARCHAR FBD†
string, normalizedString, token	1	1	1	1	1	1	1	1	2	3	4	6	5	5	5	6a	5a	5a	5a	7a	7	7	7
base64Binary, hexBinary	-	-	-	-	-	-	-	-	-	-	-	8a	8	8	8	-	-	-	-	8c	8b	8b	8b
byte, unsigned byte	0a	0a	0a	0a	0a	0a	0a	0a	-	-	-	9a*	9*	9*	9*	-	-	-	-	-	-	-	-
integer, positiveInteger, negativeInteger, nonNegativeInteger, nonPositiveInteger	10	10	10	11	11	11	10	10	-	-	-	9a*	9*	9*	9*	-	-	-	-	-	-	-	-
int	10	0a	0a	11	11	0a	0a	0a	-	-	-	9a*	9*	9*	9*	-	-	-	-	-	-	-	-
unsignedInt	10	10	0a	11	11	0a	0a	0a	-	-	-	9a*	9*	9*	9*	-	-	-	-	-	-	-	-
long	10	10	0a	11	11	11	10	0a	-	-	-	9a*	9*	9*	9*	-	-	-	-	-	-	-	-
unsignedLong	10	10	10	11	11	11	10	0a	-	-	-	9a*	9*	9*	9*	-	-	-	-	-	-	-	-
short	0a	0a	0a	0a	0a	0a	0a	0a	-	-	-	9a*	9*	9*	9*	-	-	-	-	-	-	-	-
unsignedShort	10	0a	0a	0a	0a	0a	0a	0a	-	-	-	9a*	9*	9*	9*	-	-	-	-	-	-	-	-
decimal	21	21	21	11	11	11	11	11	-	-	-	9a*	9*	9*	9*	-	-	-	-	-	-	-	-
float	22	22	22	17	16	17	0a	0a	-	-	-	9a*	9*	9*	9*	-	-	-	-	-	-	-	-
double	22	22	22	16	16	17	11	11	-	-	-	9a*	9*	9*	9*	-	-	-	-	-	-	-	-
boolean	12	12	12	12	12	12	12	12	-	-	-	9a*	9*	9*	9*	-	-	-	-	-	-	-	-
time	-	-	-	-	-	-	-	-	-	14	-	13a*	13*	13*	13*	-	-	-	-	-	-	-	-
dateTime	-	-	-	-	-	-	-	-	15	15	19	13a*	13*	13*	13*	-	-	-	-	-	-	-	-
duration, gMonth, gYear, gDay, gMonthDay, gYearMonth	-	-	-	-	-	-	-	-	-	-	-	13a	13	13	13	-	-	-	-	-	-	-	-
date	-	-	-	-	-	-	-	-	20	-	-	13a*	13*	13*	13*	-	-	-	-	-	-	-	-
Name, NCName, NOTATION, ID, IDREF, QName, NMTOKEN, ENTITY	-	-	-	-	-	-	-	-	-	-	-	6	5	5	5	6a	5a	5a	5a	7a	7	7	7
ENTITIES, NMTOKENS, IDREFS, Listentypen	-	-	-	-	-	-	-	-	-	-	-	6b	5b	5b	5b	6c	5c	5c	5c	7c	7b	7b	7b
anyURI	-	-	-	-	-	-	-	-	-	-	-	18a	18	18	18	-	-	-	-	7a	7	7	7
language	-	-	-	-	-	-	-	-	-	-	-	6	5	5	5	-	-	-	-	7a	7	7	7
anySimpleType, Union-Verknüpfungstypen	-	-	-	-	-	-	-	-	-	-	-	6d	5d	5d	5d	6e	5e	5e	5e	7e	7d	7d	7d
anyType	-	-	-	-	-	-	-	-	-	-	-	6d	5d	5d	5d	6e	5e	5e	5e	7e	7d	7d	7d

Legende

† FOR BIT DATA

* Die Annotation db2-xdb:normalization wird verwendet, um das Format der Zeichenfolge zu ermitteln, die in die Datenbank eingefügt wird.

– Datentypen (DATA) sind nicht kompatibel für die Dekomposition mithilfe mit Annotationen versehener XML-Schemata.

0 Datentypen sind kompatibel.

0a Kompatibel; wenn sich -0 im Wertebereich des XML-Typs befindet, wird -0 in der Datenbank als 0 gespeichert.

1 Kompatibel, wenn sich die Zeichenfolge in einem akzeptablen lexikalischen Format für den SQL-Zieltyp befindet und in einen numerischen Wert im Bereich des SQL-Typs konvertiert werden kann. Hierbei kann es allerdings zu Verlusten signifikanter Ziffern kommen.

2 Kompatibel, wenn die Zeichenfolge ein gültiges Datumsformat aufweist: *jjjj-mm-tt*, *mm/tt/jjjj* oder *tt.mm.jjjj*.

3 Kompatibel, wenn die Zeichenfolge ein gültiges Zeitformat aufweist: *hh.mm.ss*, *hh:mm AM oder PM* oder *hh:mm:ss*.

4 Kompatibel, wenn die Zeichenfolge ein gültiges Zeitmarkenformat aufweist: *jjjj-mm-tt-hh.mm.ss.nnnnnn* oder *jjjj-mm-tt hh.mm.ss.nnnnnn*.

5 Kompatibel, wenn die Länge der XML-Eingabezeichenfolge (in Byte) kleiner oder gleich der Länge der Zielspalte (in Byte) ist. Wenn die Eingabezeichenfolge länger als die Zielspalte ist, ist die Zeichenfolge nur kompatibel, wenn db2-xdb:truncate für diese Spaltenzuordnung auf "true" oder "1" gesetzt wird. Die Länge der Zeichenfolge wird nach der Normalisierung berechnet, wobei die Eingabezeichenfolge der Leerzeichenfassette des XML-Schematyps entsprechend normalisiert wird.

5a Kompatibel gemäß den unter 5 beschriebenen Bedingungen. Zusätzlich muss die Eingabezeichenfolge aus Doppelbytezeichen bestehen.

5b Kompatibel gemäß den unter 5 beschriebenen Bedingungen. Zusätzlich ist der in die Zielspalte eingefügte Wert die Zeichenfolge aus miteinander verknüpften Listenelementen, die durch ein einzelnes Leerzeichen voneinander getrennt sind (entsprechend der Einstellung "collapse" der Leerzeichenfassette für Listen).

5c Kompatibel gemäß den unter 5a beschriebenen Bedingungen. Zusätzlich ist der in die Zielspalte eingefügte Wert die Zeichenfolge aus miteinander verknüpften Listenelementen, die durch ein einzelnes Leerzeichen voneinander getrennt sind (entsprechend der Einstellung "collapse" der Leerzeichenfassette für Listen).

5d Kompatibel, wenn die Länge der XML-Eingabezeichenfolge (in Byte) kleiner oder gleich der Länge der Zielspalte (in Byte) ist. Wenn die Eingabezeichenfolge länger als die Zielspalte ist, ist die Zeichenfolge nur kompatibel, wenn db2-xdb:truncate für diese Spaltenzuordnung auf "true" oder "1" gesetzt wird. Der Wert, der in beiden Fällen in die Zielspalte eingefügt wird, ist der Zeicheninhalt des Elements oder Attributs.

5e Kompatibel gemäß den unter 5d beschriebenen Bedingungen. Zusätzlich muss die Eingabezeichenfolge aus Doppelbytezeichen bestehen.

6 Kompatibel, wenn die Länge der XML-Eingabezeichenfolge (in Byte) kleiner oder gleich der Länge der Zielspalte (in Byte) ist. Wenn die Eingabezeichenfolge länger als die Zielspalte ist, ist die Zeichenfolge nur kompatibel, wenn db2-xdb:truncate für diese Spaltenzuordnung auf "true" oder "1" gesetzt wird. Die Länge der Zeichenfolge wird nach der Normalisierung berechnet, wobei die Eingabezeichenfolge der Leerzeichenfassette des XML-Schematyps entsprechend normalisiert wird. Wenn die Länge der XML-Eingabezeichenfolge geringer ist als die definierte Länge der Zielspalte, wird die Zeichenfolge beim Einfügen rechts mit Leerzeichen aufgefüllt.

6a Kompatibel gemäß den unter 6 beschriebenen Bedingungen. Zusätzlich muss die Eingabezeichenfolge aus Doppelbytezeichen bestehen.

6b Kompatibel gemäß den unter 6 beschriebenen Bedingungen. Zusätzlich ist der in die Zielspalte eingefügte Wert die Zeichenfolge aus miteinander verknüpften Listenelementen, die durch ein einzelnes Leerzeichen voneinander getrennt sind (entsprechend der Einstellung "collapse" der Leerzeichenfassette für Listen).

6c Kompatibel gemäß den unter 6a beschriebenen Bedingungen. Zusätzlich ist der in die Zielspalte eingefügte Wert die Zeichenfolge aus miteinander verknüpften Listenelementen, die durch ein einzelnes Leerzeichen voneinander getrennt sind (entsprechend der Einstellung "collapse" der Leerzeichenfassette für Listen).

6d Kompatibel, wenn die Länge der XML-Eingabezeichenfolge (in Byte) kleiner oder gleich der Länge der Zielspalte (in Byte) ist. Wenn die Eingabezeichenfolge länger als die Zielspalte ist, ist die Zeichenfolge nur kompatibel, wenn db2-xdb:truncate für diese Spaltenzuordnung auf "true" oder "1" gesetzt wird. Der Wert, der in beiden Fällen in die Zielspalte eingefügt wird, ist der Zeicheninhalt des Elements oder Attributs. Wenn die Länge der XML-Eingabezeichenfolge geringer ist als die definierte Länge der Zielspalte, wird die Zeichenfolge beim Einfügen rechts mit Leerzeichen aufgefüllt.

6e Kompatibel gemäß den unter 6d beschriebenen Bedingungen. Zusätzlich muss die Eingabezeichenfolge aus Doppelbytezeichen bestehen.

7 Kompatibel, wenn die Länge der XML-Eingabezeichenfolge (in Byte) kleiner oder gleich der Länge der Zielspalte (in Byte) ist. Wenn die Eingabezeichenfolge länger als die Zielspalte ist, ist die Zeichenfolge nur kompatibel, wenn db2-xdb:truncate für diese Spaltenzuordnung auf "true" oder "1" gesetzt wird. Die Länge der Zeichenfolge wird nach der Normalisierung berechnet, wobei die Eingabezeichenfolge der Leerzeichenfassette des XML-Schematyps entsprechend normalisiert wird.

7a Kompatibel gemäß den unter 7 beschriebenen Bedingungen. Zusätzlich gilt: Wenn die Länge der XML-Eingabezeichenfolge geringer ist als die definierte Länge der Zielspalte, wird die Zeichenfolge beim Einfügen rechts mit Leerzeichen aufgefüllt.

7b Kompatibel gemäß den unter 7 beschriebenen Bedingungen. Zusätzlich ist der in die Zielspalte eingefügte Wert die Zeichenfolge aus miteinander verknüpften Listenelementen, die durch ein einzelnes Leerzeichen voneinander getrennt sind (entsprechend der Einstellung "collapse" der Leerzeichenfassette für Listen).

7c Kompatibel gemäß den unter 7b beschriebenen Bedingungen. Zusätzlich gilt: Wenn die Länge der XML-Eingabezeichenfolge geringer ist als die definierte Länge der Zielspalte, wird die Zeichenfolge beim Einfügen rechts mit Leerzeichen aufgefüllt.

7d Kompatibel, wenn die Länge der XML-Eingabezeichenfolge (in Byte) kleiner oder gleich der Länge der Zielspalte (in Byte) ist. Wenn die Eingabezeichenfolge länger als die Zielspalte ist, ist die Zeichenfolge nur kompatibel, wenn db2-xdb:truncate für diese Spaltenzuordnung auf "true" oder "1" gesetzt wird. Der Wert, der in beiden Fällen in die Zielspalte eingefügt wird, ist der Zeicheninhalt des Elements oder Attributs.

7e Kompatibel gemäß den unter 7d beschriebenen Bedingungen. Zusätzlich gilt: Wenn die Länge der XML-Eingabezeichenfolge geringer ist als die definierte Länge der Zielspalte, wird die Zeichenfolge beim Einfügen rechts mit Leerzeichen aufgefüllt.

8 Kompatibel, wenn die Länge der XML-Eingabezeichenfolge (in Byte) klei-

ner oder gleich der Länge der Zielspalte (in Byte) ist. Wenn die Eingabezeichenfolge länger als die Zielspalte ist, ist die Zeichenfolge nur kompatibel, wenn db2-xdb:truncate für diese Spaltenzuordnung auf "true" oder "1" gesetzt wird. Die codierte (ursprüngliche) Zeichenfolge wird eingefügt.

8a Kompatibel gemäß den unter 8 beschriebenen Bedingungen. Zusätzlich gilt: Wenn die Länge der XML-Eingabezeichenfolge geringer ist als die definierte Länge der Zielspalte, wird die Zeichenfolge beim Einfügen rechts mit Leerzeichen aufgefüllt.

8b Kompatibel, wenn die Länge der XML-Eingabezeichenfolge (in Byte) kleiner oder gleich der Länge der Zielspalte (in Byte) ist. Wenn die Eingabezeichenfolge länger als die Zielspalte ist, ist die Zeichenfolge nur kompatibel, wenn db2-xdb:truncate für diese Spaltenzuordnung auf "true" oder "1" gesetzt wird. Der Wert, der in die Zielspalte eingefügt wird, ist die decodierte Zeichenfolge.

8c Kompatibel gemäß den unter 8b beschriebenen Bedingungen. Zusätzlich gilt: Wenn die Länge der XML-Eingabezeichenfolge geringer ist als die definierte Länge der Zielspalte, wird die Zeichenfolge beim Einfügen rechts mit Leerzeichen aufgefüllt.

9 Kompatibel, wenn die Länge der XML-Eingabezeichenfolge, die nach der Verarbeitung entsprechend der db2-xdb:normalization-Einstellung berechnet wird, kleiner oder gleich der Länge der Zielspalte ist. Darüber hinaus kompatibel, wenn db2-xdb:truncate für diese Spaltenzuordnung auf "true" oder "1" gesetzt wird.

9a Kompatibel gemäß den unter 9 beschriebenen Bedingungen. Zusätzlich gilt: Wenn die Länge der XML-Eingabezeichenfolge geringer ist als die definierte Länge der Zielspalte, wird die Zeichenfolge beim Einfügen rechts mit Leerzeichen aufgefüllt.

10 Kompatibel, wenn sich der XML-Typ im Bereich des SQL-Typs befindet. Wenn sich -0 im Wertebereich des XML-Typs befindet, wird -0 in der Datenbank als 0 gespeichert.

11 Kompatibel, wenn sich der XML-Wert im Bereich des SQL-Typs befindet. Hierbei kann es allerdings zu Verlusten signifikanter Ziffern kommen. Wenn sich -0 im Wertebereich des XML-Typs befindet, wird -0 in der Datenbank als 0 gespeichert.

12 Kompatibel; der eingefügte Wert ist '0' (für 'false') oder '1' (für 'true').

13 Kompatibel, wenn die Länge der XML-Eingabezeichenfolge, die nach der Verarbeitung entsprechend der db2-xdb:normalization-Einstellung berechnet wird, kleiner oder gleich der Länge der Zielspalte ist. Darüber hinaus kompatibel, wenn db2-xdb:truncate für diese Spaltenzuordnung auf "true" oder "1" gesetzt wird.

13a Kompatibel gemäß den unter 13 beschriebenen Bedingungen. Zusätzlich gilt: Wenn die Länge der XML-Eingabezeichenfolge geringer ist als die definierte Länge der Zielspalte, wird die Zeichenfolge beim Einfügen rechts mit Leerzeichen aufgefüllt.

14 Bei XML-Werten mit Sekundenbruchteilen nur kompatibel, wenn die Dekompositionsannotation für db2-xdb:truncate "true" oder "1" angibt. Bei XML-Werten mit Zeitzonenindikatoren kompatibel, wenn für db2-xdb:truncate "true" oder "1" angegeben wird. Die Werte werden ohne die Zeitzone eingefügt.

15 Kompatibel, wenn die Jahresangabe aus vier Ziffern besteht und ihr kein

Minuszeichen (-) vorangestellt ist. Kompatibel, wenn der XML-Wert keinen Zeitzonenindikator aufweist. Enthält der XML-Wert einen Zeitzonenindikator, sind die Werte kompatibel, wenn für db2-xdb:truncate "true" oder "1" angegeben wird.

16 Kompatibel, wenn der Wert sich im Bereich des SQL-Typs befindet und nicht "INF", "-INF" oder "NaN" lautet. Wenn sich -0 im Wertebereich des XML-Typs befindet, wird -0 in der Datenbank als 0 gespeichert.Hierbei kann es allerdings zu Verlusten signifikanter Ziffern kommen.

17 Kompatibel, wenn der Wert nicht "INF", "-INF" oder "NaN" ist. Wenn sich -0 im Wertebereich des XML-Typs befindet, wird -0 in der Datenbank als 0 gespeichert.

18 Kompatibel, wenn die Länge der Zeichenfolge der URI (in Byte) kleiner oder gleich der Länge der Zielspalte (in Byte) ist. Wenn die Eingabezeichenfolge länger als die Zielspalte ist, ist die Zeichenfolge nur kompatibel, wenn db2-xdb:truncate für diese Spaltenzuordnung auf "true" oder "1" gesetzt wird. Bitte beachten Sie, dass die URI selbst eingefügt wird, nicht die Ressource, auf die sie verweist.

18a Kompatibel gemäß den unter 18 beschriebenen Bedingungen. Zusätzlich gilt: Wenn die Länge der XML-Eingabezeichenfolge geringer ist als die definierte Länge der Zielspalte, wird die Zeichenfolge beim Einfügen rechts mit Leerzeichen aufgefüllt.

19 Kompatibel, wenn die Jahresangabe aus vier Ziffern besteht und ihr kein Minuszeichen (-) vorangestellt ist. Bei XML-Werten mit Zeitzonenindikatoren kompatibel, wenn für db2-xdb:truncate "true" oder "1" angegeben wird. (Die Werte werden in diesem Fall ohne Zeitzone eingefügt.) Bei der Angabe von Sekundenbruchteilen mit mehr als sechs Ziffern kompatibel, wenn für db2-xdb:truncate "true" oder "1" angegeben wird.

20 Kompatibel, wenn die Jahresangabe aus vier Ziffern besteht und ihr kein Minuszeichen (-) vorangestellt ist. Bei XML-Werten mit Zeitzonenindikatoren kompatibel, wenn für db2-xdb:truncate "true" oder "1" angegeben wird. (Die Datumswerte werden in diesem Fall ohne Zeitzone eingefügt.)

21 Der Bruchteil der Zahl wird abgeschnitten. Kompatibel, wenn sich der ganzzahlige Teil im Bereich des SQL-Typs befindet. Wenn sich -0 im Wertebereich des XML-Typs befindet, wird -0 in der Datenbank als 0 gespeichert.

22 Der Bruchteil der Zahl wird abgeschnitten. Kompatibel, wenn sich der ganzzahlige Teil im Bereich des SQL-Typs befindet und der Wert nicht "INF", "-INF" oder "NaN" lautet. Wenn sich -0 im Wertebereich des XML-Typs befindet, wird -0 in der Datenbank als 0 gespeichert.

Grenzwerte und Einschränkungen für die Dekomposition mithilfe eines mit Annotationen versehenen XML-Schemas

Für die Dekomposition mithilfe eines mit Annotationen versehenen XML-Schemas gelten bestimmte Grenzwerte und Einschränkungen.

Grenzwerte

Tabelle 73. Grenzwerte für die Dekomposition mithilfe eines mit Annotationen versehenen XML-Schemas

Bedingung	Grenzwert
Maximal zulässige Größe des für die Dekomposition vorgesehenen Dokuments	2 GB
Maximal zulässige Anzahl der Tabellen, auf die in einem einzelnen mit Annotationen versehenen XML-Schema verwiesen wird	100
Maximale Anzahl von $DECOMP_CONTENT- oder $DECOMP_ELEMENTID-Instanzen in einer Annotation db2-xdb:expression	10
Maximale Anzahl von Schritten in 'db2-xdb:locationPath'	100
Maximal zulässige Anzahl der explizit im Attribut "namespace" von <xs:any> oder <xs:anyAttribute> aufgelisteten Namensbereiche (wenn die Liste die Sonderwerte `##targetNamespace` oder `##local` enthält, zählen diese ebenfalls für den Grenzwert mit)	25
Maximal zulässige Zeichenfolgelänge des Werts von 'db2-xdb:name' (Tabellenname), 'db2-xdb:column', 'db2-xdb:defaultSQLSchema' oder 'db2-xdb:SQLSchema'	Stimmt mit dem Grenzwert für das entsprechende DB2-Objekt überein.
Maximal zulässige Zeichenfolgelänge des Werts von 'db2-xdb:rowSet'	Stimmt mit dem Grenzwert für 'db2-xdb:name' überein
Maximal zulässige Zeichenfolgelänge für den Wert von $DECOMP_CONTENT	4096 Byte

Einschränkungen

- Die Dekomposition mithilfe mit Annotationen versehener XML-Schemata bietet keine Unterstützung für:
  - Die Dekomposition von Platzhalterelementen bzw. -attributen: Für Elemente oder Attribute im XML-Dokument, die der Deklaration '<xs:any>' oder '<xs:anyAttribute>' im XML-Schema entsprechen, wird keine Dekomposition durchgeführt.

    Wenn es sich bei diesen Elementen oder Attributen jedoch um untergeordnete Elemente von Elementen handelt, für die eine Dekomposition mit der Angabe "serializeSubtree" oder "stringValue" für 'db2-xdb:contentHandling' durchgeführt wurde, wird für den Inhalt der Platzhalterelemente bzw. -attribute eine Dekomposition als Teil des serialisierten Unterverzeichnisstruktur- oder Zeichenfolgewerts durchgeführt. Diese Platzhalterelemente bzw. -attribute müssen jedoch die Vorgaben für Namensbereiche erfüllen, die in der entsprechenden Deklaration '<xs:any>' oder '<xs:anyAttribute>' angegeben sind, damit sie Teil der Serialisierung sind.
  - Substitutionsgruppen: Es wird ein Fehler generiert, wenn ein Mitglied einer Substitutionsgruppe im XML-Dokument an der Stelle vorkommt, an der im

XML-Schema der Gruppenkopfsatz aufgeführt ist. Dies gilt für Fälle, bei denen die Substitutionsgruppenmitglieder nicht nur als Stammelement des Dokuments verwendet werden.

Um dieses Problem zu umgehen, können stattdessen die Elementdeklarationen des Kopfsatzes und der Mitglieder der Substitutionsgruppe in eine benannte Modellgruppe vom Typ xs:choice geändert werden. Hierbei können Sie z. B. die Substitutionsgruppendeklarationen

```
<xs:element name="head" type="BaseType" />
<xs:element name="member1" type="derived1FromBaseType" substitutionGroup="head"/>
<xs:element name="member2" type="derived2FromBaseType" substitutionGroup="head"/>
<xs:element name="member3" type="derived3FromBaseType" substitutionGroup="head"/>
```

in die folgenden, gleichwertigen benannten Modellgruppen geändert werden:

```
<xs:group name="mysubstitutiongrp">
        <xs:choice>
        <xs:element name="head" type="BaseType"/>
        <xs:element name="member1" type="derived1FromBaseType"/>
        <xs:element name="member2" type="derived2FromBaseType"/>
          <xs:element name="member3" type="derived3FromBaseType"/>
  </xs:choice>
</xs:group>
```

Vorkommen des Elements <head> können anschließend im XML-Dokument durch die neu definierte, benannte Modellgruppe ersetzt werden.

– Laufzeitsubstitution unter Verwendung von 'xsi:type': Für ein Element wird die Dekomposition anhand der Zuordnungen in dem Schematyp durchgeführt, der dem Elementnamen im Schema zugeordnet ist. Die Angabe eines anderen Typs für ein Element im Dokument durch die Verwendung von 'xsi:type' führt zu einem Fehler, der während der Dekomposition zurückgegeben wird.

  Vergewissern Sie sich, dass der Typ eines Elements, das im XML-Dokument mit xsi:type angegeben wurde, mit dem Typ übereinstimmt, der für dieses Element im Kontext angegeben wurde. Wenn der Inhalt des Elements oder seiner Nachkommen nicht einzeln zerlegt werden muss, kann der Elementtyp im XML-Schema in xs:anyType geändert werden. Durch diese Änderung ist es nicht erforderlich, die XML-Dokumente zu ändern.

– Rekursive Elemente: XML-Schemata mit Rekursion können im XML-Schema-Repository (XSR) registriert und für die Dekomposition aktiviert werden. Die rekursiven Abschnitte eines zugeordneten XML-Instanzdokuments können jedoch nicht als skalare Werte zerlegt und in eine Zieltabelle eingefügt werden. Mithilfe entsprechender Schemaannotationen können die rekursiven Abschnitte gespeichert und zu einem späteren Zeitpunkt als serialisierte Markupdatei abgerufen werden.
– Aktualisierungen oder Löschung vorhandener Zeilen in Zieltabellen: Die Dekomposition unterstützt nur die Einfügung neuer Zeilen. (Außerhalb des XML-Dekompositionsprozesses können Zeilen weiterhin aktualisiert oder gelöscht werden.)
– Attribute eines einfachen, aus NOTATION abgeleiteten Typs: Die Dekomposition fügt nur den Notationsnamen ein.
– Attribute des Typs ENTITY: Die Dekomposition fügt nur den Entitätsnamen ein.
– Mehrere Zuordnungen zur gleichen Zeilengruppe (rowSet) und zur gleichen Spalte mit 'db2-xdb:expression' und 'db2-xdb:condition': Wenn mehrere Elemente nach den Zuordnungsregeln zulässigerweise derselben Zeilengruppe und Spalte zugeordnet werden können, dürfen die Zuordnungen keine Annotationen 'db2-xdb:expression' oder 'db2-xdb:condition' enthalten.

- In einer Umgebung mit partitionierten Datenbanken wird die Dekomposition mithilfe eines mit Annotationen versehenen XML-Schemas nur in der Datenbankpartition unterstützt, die die Datenbankkatalogtabellen enthält (die Datenbankpartition IBMCATGROUP).

Fehlerbehebung für die Dekomposition mithilfe eines mit Annotationen versehenen XML-Schemas

Wenn die Dekomposition nicht die erwarteten Ergebnisse liefert, sollte eine Reihe von Aspekten berücksichtigt werden.

Allgemeine Aspekte

- Überprüfen Sie, ob das XSR-Objekt, das Ihrem XML-Schema zugeordnet ist, in der Spalte DECOMPOSITION der Katalogsicht SYSCAT.XSROBJECTS als aktiviert ausgewiesen ist. Ist das XSR-Objekt nicht aktiviert, sollten Sie ggf. die Fehlerberichtigungsmaßnahmen ausführen, die in der Dokumentation zur Inaktivierung angegeben sind.
- Stellen Sie sicher, dass gegen keine Begrenzungen und Einschränkungen für die XML-Dekomposition verstoßen wird.
- Vergewissern Sie sich, dass das XML-Dokument in Bezug auf sein XML-Schema gültig ist. Die Gültigkeitsprüfung muss für die Dekomposition nicht zwingend ausgeführt werden, wenn Sie jedoch eine bestimmte Funktionsweise wie z. B. die Zeichenentitätserweiterung erwarten, dann müssen Sie während der Dekomposition eine Gültigkeitsprüfung durchführen.

Aspekte im Zusammenhang mit XML-Schemata

- Stellen Sie sicher, dass das XML-Schema keine Fehler wie zum Beispiel nicht deterministische Inhaltsmodelle enthält, da solche Fehlertypen zum Fehlschlagen der Dekomposition führen können, wenn die Gültigkeitsprüfung ausgeführt wird. Es kann auch zu nicht definierten Dekompositionsergebnissen kommen, wenn die Gültigkeitsprüfung nicht ausgeführt wird.
- Vergewissern Sie sich, dass die nicht globalen Annotationen nur in Element- oder Attributdeklarationen und nicht in komplexen Typen, Element-/Attributverweisen, Modellgruppen oder anderen XML-Schemakonstrukten deklariert wurden. Prüfen Sie außerdem, ob die Annotationen in ihrem unterstützten Format deklariert wurden: als Attribute, Elemente oder globale Annotationen. (Detaillierte Informationen zur Angabe einer Annotation finden Sie in der Dokumentation zur jeweiligen Annotation.)
- Stellen Sie sicher, dass komplexe Typen, die durch Erweiterung oder Einschränkung abgeleitet sind, korrekt annotiert wurden.

Spezielle Fehler

Durch die Anpassung der Datenbankkonfigurationsparameter können die folgenden Fehler behoben werden:

- Fehler SQL0954C, wenn das mit Annotationen versehene XML-Schema eine große Anzahl von Zeilengruppen (rowSets) enthält: Erhöhen Sie die Größe des Zwischenspeichers für die Anwendung und verwenden Sie hierzu den Konfigurationsparameter applheapsz.
- Fehler SQL0954C, wenn das mit Annotationen versehene XML-Schema in allen Zeilengruppenelementen (rowSets) komplexe oder sehr viele Ausdrücke enthält: Erhöhen Sie die Größe des Zwischenspeichers für die Anwendung und verwenden Sie hierzu den Konfigurationsparameter applheapsz.

- Fehler SQL0964C, wenn die Dekomposition zu einer großen Anzahl von Zeilen führt: Erhöhen Sie die Anzahl der verfügbaren primären oder sekundären Protokolldateien und verwenden Sie hierzu die Konfigurationsparameter **logprimary** und **logsecond**. Sie können auch die Größe der Dateien für das primäre und das sekundäre Protokoll erhöhen. Dieser Arbeitsschritt kann mithilfe des Konfigurationsparameters **logfilsiz** ausgeführt werden.

Sperren und gemeinsamer Zugriff

Wenn Sie bei der Dekomposition von Dokumenten eine Sperreneskalation oder Deadlocks feststellen, müssen Sie die Steuerung des gemeinsamen Zugriffs über Ihre Anwendung anpassen. Wenn in einer Anwendung mehrere gleichzeitige Aufrufe für eine der gespeicherten xdbDecompXML-Prozeduren ausgeführt werden, bei denen zahlreiche identische Tabellen für die verschiedenen Dekompositionsoperationen benötigt werden, dann muss die Anwendung den gleichzeitigen Zugriff auf diese Tabellen koordinieren, um eine Sperreneskalation und Deadlocks zu vermeiden.

Eine Möglichkeit zur Anpassung der Steuerung des gemeinsamen Zugriffs besteht im expliziten Sperren aller Tabellen, die bei einer Dekomposition benötigt werden, bevor die gespeicherte Prozedur xdbDecompXML aufgerufen wird. Anschließend müssen Sie die Anweisung COMMIT oder ROLLBACK ausführen, nachdem die Ergebnisse der gespeicherten Prozedur zurückgegeben wurden. Da es bei der Dekomposition sehr umfangreicher Dokumente zu einer großen Zahl einzufügender Zeilen kommen kann und weil jede Zeile während einer Einfügeoperation standardmäßig gesperrt wird, kann eine Anwendung, die viele Zeilen einfügt, zahlreiche Zeilensperren haben, wodurch es zu einer Sperreneskalation kommen kann. Wenn stattdessen Tabellensperren angefordert werden, können Sie den Systemaufwand für das Anfordern von Zeilensperren und die Sperreneskalation vermeiden.

Wenn die Reduzierung des gleichzeitigen Zugriffs durch das Anfordern von Tabellensperren in Ihrer Anwendung nicht praktikabel ist, können Sie den Wert des Datenbankkonfigurationsparameters maxlocks und/oder des Datenbankkonfigurationsparameters locklist erhöhen und so die Wahrscheinlichkeit für das Auftreten einer Sperreneskalation reduzieren.

Definieren Sie den Datenbankkonfigurationsparameter locktimeout, um zu vermeiden, dass eine Anwendung unbegrenzte Zeit auf die Zuteilung einer Sperre wartet.

Zuordnungsprüfung in Katalogsicht

Wenn bei der Dekomposition weiterhin Fehler auftreten, obwohl Sie sichergestellt haben, dass die oben aufgeführten Bedingungen erfüllt sind, vergewissern Sie sich, dass die Spalte MAPPINGDESCRIPTION der Katalogsicht SYSCAT.XDBMAPSHREDTREES mit den gewünschten Zuordnungen übereinstimmt. Die Spalte MAPPINGDESCRIPTION enthält Details dazu, wie die einzelnen Elemente einer Zeilengruppe zugeordnet wurden. Dies betrifft zum Beispiel folgende Elemente:

- Name der Zielspalte
- Typ der Zielspalte
- XML-Schematyp des Elements
- Für 'db2-xdb:contentHandling', 'db2-xdb:normalization', 'db2-xdb:truncate', 'db2-xdb:expression' und 'db2-xdb:condition' angegebene Werte

Beachten Sie, dass die Spalten von SYSCAT.XDBMAPSHREDTREES (mit Ausnahme von MAPPINGDESCRIPTION) für die DB2-Kundenunterstützung vorgesehen sind.

Schema für XML-Dekompositionsannotationen

Die Dekomposition mithilfe mit Annotationen versehener XML-Schemata unterstützt eine Gruppe von Dekompositionsannotationen, die es Ihnen ermöglichen, anzugeben, wie XML-Dokumente zerlegt und in Datenbanktabellen eingefügt werden sollen. In diesem Abschnitt wird das XML-Schema für das mit Annotationen versehene Schema gemäß Definition durch die XML-Dekomposition gezeigt:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
                      xmlns="http://www.ibm.com/xmlns/prod/db2/xdb1"
                      targetNamespace="http://www.ibm.com/xmlns/prod/db2/xdb1"
                      elementFormDefault="qualified" >
    <xs:element name="defaultSQLSchema" type="xs:string"/>
    <xs:attribute name="rowSet" type="xs:string"/>
    <xs:attribute name="column" type="xs:string"/>
    <xs:attribute name="locationPath" type="xs:string"/>
    <xs:attribute name="truncate" type="xs:boolean"/>
    <xs:attribute name="contentHandling">
    <xs:simpleType>
      <xs:restriction base="xs:string">
                <xs:enumeration value="text"/>
                <xs:enumeration value="serializeSubtree"/>
                <xs:enumeration value="stringValue"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
    <xs:attribute name="normalization" >
    <xs:simpleType>
      <xs:restriction base="xs:string">
                <xs:enumeration value="original"/>
                <xs:enumeration value="whitespaceStrip"/>
                <xs:enumeration value="canonical"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
    <xs:attribute name="expression" type="xs:string"/>
    <xs:attribute name="condition" type="xs:string"/>
    <xs:element name="table">
                <xs:complexType>
      <xs:sequence>
                <xs:element name="SQLSchema" type="xs:string" minOccurs="0"/>
                <xs:element name="name" type="xs:string"/>
                <xs:element name="rowSet" type="xs:string"
                                      maxOccurs="unbounded" form="qualified"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
    <xs:element name="rowSetMapping">
                <xs:complexType>
      <xs:sequence>
                <xs:element name="rowSet" type="xs:string" />
                <xs:element name="column" type="xs:string" minOccurs="0"/>
                <xs:element name="expression" type="xs:string" minOccurs="0" />
                <xs:element name="condition" type="xs:string" minOccurs="0"/>
      </xs:sequence>
            <xs:attribute ref="truncate" />
            <xs:attribute ref="locationPath" />
            <xs:attribute ref="normalization" />
            <xs:attribute ref="contentHandling" />
    </xs:complexType>
  </xs:element>
  <xs:element name='rowSetOperationOrder'>
```

```
              <xs:complexType>
      <xs:choice minOccurs='1'  maxOccurs='1'>
        <xs:element name='order' type='orderType' minOccurs='1'
 maxOccurs='unbounded'/>
      </xs:choice>
    </xs:complexType>
  </xs:element>
  <xs:complexType name='orderType'>
    <xs:sequence>
      <xs:element name='rowSet' type='xsd:string' minOccurs='2'
      maxOccurs='unbounded'/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Kapitel 14. Einschränkungen bei pureXML

Einschränkungen bei der Funktion pureXML

Die Funktion pureXML unterliegt bestimmten Einschränkungen bei XML-Spaltendefinitionen, beim Anhängen von Tabellen an partitionierte Tabellen und in einer Umgebung mit partitionierten Datenbanken.

Einschränkungen bei den XML-Spaltendefinitionen

XML-Spalten unterliegen den folgenden Einschränkungen:

- Wenn Sie einen Index für eine XML-Spalte erstellen, müssen Sie die Klausel GENERATE KEY USING XMLPATTERN verwenden und der Index darf nicht zu einem zusammengesetzten Index gehören. Für eine XML-Spalte können mehrere Indizes erstellt werden.
- Auf die XML-Spalten kann in Prüfungen auf Integritätsbedingungen mit CHECK nur zusammen mit einem Vergleichselement VALIDATED verwiesen werden.
- XML-Spalten dürfen keine Standardwerte aufweisen, die mit der Klausel WITH DEFAULT angegeben wurden. Wenn in der Spalte Nullwerte zulässig sind, dann lautet der Standardwert für die Spalte NULL.
- XML-Spalten können nicht in Bereichsclustertabellen verwendet werden.
- XML-Spalten können nicht als Spalten für Schlüssel aufgenommen werden. Hierzu zählen Primär- und Fremdschlüssel sowie eindeutige Schlüssel, Dimensionsschlüssel (in der Klausel ORGANIZE BY) von MDC-Tabellen (MDC = Multi-Dimensional Clustering), Sortierschlüssel von Bereichsclustertabellen, Tabellenpartitionierungsschlüssel von partitionierten Tabellen und Verteilungsschlüssel einer Tabelle in einer Umgebung mit partitionierten Datenbanken.
- Eine partitionierte Tabelle mit XML-Spalten muss mindestens eine Nicht-XML-Spalte mit einem Datentyp enthalten, dessen Verwendung als Schlüsselspalte für die Tabellenpartitionierung unterstützt wird.
- XML-Spalten können nicht in typisierte Tabellen und Sichten aufgenommen werden.
- XML-Spalten können nicht in generierten Spalten referenziert werden.
- XML-Spalten können nicht in der SELECT-Liste verschiebbarer Cursor angegeben werden.
- XML-Spalten können zur Inaktivierung der Cursorblockung führen, wenn XML-Daten abgerufen werden.
- Wenn Sie XML-Spalten mit der Anweisung ALTER TABLE löschen, müssen Sie alle XML-Spalten der Tabelle mithilfe einer einzigen Anweisung ALTER TABLE löschen.
- Ab DB2 Version 9.7 Fixpack 1 können Verteilungsstatistikdaten für Indizes zu XML-Daten erfasst werden, die für eine XML-Spalte definiert sind. Die folgenden Einschränkungen gelten für die Erfassung von Verteilungsstatistikdaten für eine XML-Spalte:
  - Verteilungsstatistikdaten werden für alle Indizes zu XML-Daten erfasst, die für die XML-Spalte angegeben sind. Der für den Index angegebene Datentyp muss VARCHAR, DOUBLE, INTEGER, DECIMAL, TIMESTAMP oder DATE sein. Für Indizes zu XML-Daten mit dem Typ VARCHAR HASHED werden keine Verteilungsstatistikdaten erfasst.

– Die Verteilungsstatistikdaten für jeden Index zu XML-Daten verwenden standardmäßig maximal 250 Quantile. Der Standardwert kann geändert werden, indem bei der Eingabe des Befehls **RUNSTATS** ein Wert für den Parameter **NUM_QUANTILES** in der Klausel **ON COLUMNS** oder **DEFAULT** angegeben wird. Der Datenbankkonfigurationsparameter **num_quantiles** wird bei der Erfassung von XML-Verteilungsstatistikdaten ignoriert.
– XML-Verteilungsstatistikdaten werden nicht erstellt, wenn Daten mit der Option **STATISTICS** geladen werden.
– XML-Verteilungsstatistikdaten werden nicht für partitionierte Indizes zu XML-Daten erfasst, die für eine partitionierte Tabelle definiert sind.

Einschränkungen bei Triggern

In einem Hauptteil eines Triggers vom Typ BEFORE oder AFTER können Übergangsvariablen, die auf Spalten vom Typ XML in den betreffenden Zeilen verweisen, nur für die Überprüfung durch die Funktion XMLVALIDATE verwendet werden, um XML-Spaltenwerte auf NULL zu setzen oder um XML-Spaltenwerte unverändert zu lassen.

Einschränkungen beim Anhängen von Partitionen an partitionierte Tabellen

Wenn Sie mithilfe von ALTER ATTACH eine Partition an eine partitionierte Tabelle mit XML-Spalten anhängen, muss die integrierte Länge (INLINE LENGTH) der einzelnen XML-Spalten der anzuhängenden Tabelle (Quellentabelle) mit der integrierten Länge der entsprechenden XML-Spalte der Zieltabelle übereinstimmen.

Wenn eine Tabelle eine XML-Spalte enthält, die Version 9.5 oder ein früheres XML-Satzformat verwendet, wird das Anhängen der Tabelle an eine partitionierte Tabelle mit XML-Spalten, die Version 9.7 oder ein früheres Satzformat verwendet, nicht unterstützt. Vor dem Anhängen der Tabelle müssen Sie das XML-Satzformat der Tabelle so aktualisieren, dass es mit dem Satzformat der partitionierten Zieltabelle übereinstimmt. Mit einer der beiden Methoden können Sie das XML-Satzformat einer Tabelle aktualisieren:

- Verschieben Sie die Tabelle online mithilfe der Prozedur ADMIN_MOVE_TABLE.
- Führen Sie dazu folgende Schritte aus:
  1. Verwenden Sie den Befehl **EXPORT**, um eine Kopie der Tabellendaten zu erstellen.
  2. Verwenden Sie die Anweisung TRUNCATE, um alle Zeilen aus der Tabelle zu löschen und den der Tabelle zugeordneten Speicher freizugeben.
  3. Verwenden Sie den Befehl **LOAD**, um die Daten zur Tabelle hinzuzufügen.

Nach dem Aktualisieren des XML-Satzformats der Tabelle, ordnen Sie die Tabelle der partitionierten Zieltabelle zu.

Einschränkungen in einer Umgebung mit partitionierten Datenbanken

Wird die Funktion pureXML in einer Umgebung mit partitionierten Datenbanken verwendet, gelten folgende Regeln:

- Eine XML-Spalte kann nicht als Verteilungsschlüssel verwendet werden. Daraus ergeben sich folgende Einschränkungen:
  – Eine Tabelle, die ausschließlich XML-Spalten enthält, kann nicht verteilt werden.

  - Bei einer Tabelle mit einem Verteilungsschlüssel kann weder einen Primärschlüssel, noch eine eindeutige Integritätsbedingung noch ein eindeutigen Index in einer XML-Spalte definiert werden.
  - Eine Tabelle mit einem Verteilungsschlüssel mit XML-Spalten muss mindestens eine Nicht-XML-Spalte mit einem Datentyp enthalten, dessen Verwendung als Verteilungsschlüssel unterstützt wird.
- Wenn Sie XML-Daten aus XML-Datendateien parallel in partitionierte Tabellen laden, müssen alle Partitionen, auf denen der parallele Ladevorgang erfolgt, Lesezugriff auf die XML-Datendateien haben.
- Wenn Sie einen Dateityp CURSOR mit dem Befehl **LOAD** zum Laden von XML-Daten in eine Mehrpartitionsdatenbank verwenden, werden die Modi `PARTITION_ONLY` und `LOAD_ONLY` nicht unterstützt.
- Bei Verwendung des Befehls **REDISTRIBUTE DATABASE PARTITION GROUP** mit der Option **NOT ROLLFORWARD RECOVERABLE** verwendet die Umverteilungsoperation bei Tabellen, die XML-Spalten enthalten, die Option **INDEXING MODE DEFERRED**. Wenn eine Tabelle keine XML-Spalte enthält, verwendet die Umverteilungsoperation beim Absetzen des Befehls den angegebenen Indexierungsmodus.
- Tabellen mit XML-Spalten, die das XML-Satzformat von Version 9.5 oder älter verwenden, können nicht umverteilt werden. Verwenden Sie die gespeicherte Prozedur ADMIN_MOVE_TABLE, um die Tabelle in das neue Format zu konvertieren.

Einschränkung bei der Datenzeilenkomprimierung

Die Option COMPRESS YES der Anweisung ALTER TABLE oder CREATE TABLE ermöglicht die Datenzeilenkomprimierung in einer Tabelle. Die Datenkomprimierung im XML-Speicherobjekt einer Tabelle wird nicht unterstützt, wenn die Tabelle XML-Spalten enthält, die das XML-Satzformat von Version 9.5 oder älter verwenden. Wenn Sie eine solche Tabelle für die Datenzeilenkomprimierung aktivieren, werden nur die Tabellenzeilendaten im Tabellenobjekt komprimiert.

Wenn eine Tabelle für die Datenzeilenkomprimierung aktiviert wurde, das XML-Speicherobjekt der Tabelle bei einer Einfüge-, Lade- oder Reorganisierungsoperation jedoch nicht komprimiert werden kann, wird eine Nachricht in eine Protokolldatei **db2diag** geschrieben.

Damit die Daten im XML-Speicherobjekt der Tabelle für die Komprimierung infrage kommen, müssen Sie die Tabelle mit der gespeicherten Prozedur ADMIN_MOVE_TABLE in das neue Format konvertieren und anschließend die Datenzeilenkomprimierung für die migrierte Tabelle aktivieren.

Weitere Einschränkungen

Verwendung serialisierter XML-Daten: Es bestehen zwar keine architekturbedingten Einschränkungen für die Größe eines XML-Werts in einer Datenbank; für serialisierte XML-Daten, die mit der Datenbank ausgetauscht werden sollen, besteht jedoch eine effektive Beschränkung auf eine Größe von 2 GB.

Verwendung des Befehls RUNSTATS: Wenn sich nach einer **ALLOW READ ACCESS**-Ladeoperation mit XML-Daten eine Tabelle im Status 'Festlegen der Integrität anstehend' befindet, kann der Befehl **RUNSTATS** für diese Tabelle abgesetzt werden. In diesem Szenario kann die **RUNSTATS**-Operation die nicht sichtbaren XML-Indexschlüssel der vorhergehenden Ladeoperation nicht erkennen und gibt einen Fehler zurück. Zur Umgehung des Problems muss die Anweisung SET INTEGRITY vor dem Befehl **RUNSTATS** ausgeführt werden.

Verwendung des Befehls LOAD: Wenn Sie XML-Daten mit dem Befehl **LOAD** laden, wird die Angabe einer Ladeausnahmetabelle mit der Klausel **FOR EXCEPTION** in den folgenden Fällen nicht unterstützt:

- Bei Verwendung von kennsatzbasierter Zugriffssteuerung (LBAC)
- Beim Laden von Daten in eine partitionierte Tabelle

Erstellen von Indizes für XML-Spalten: Für das Erstellen von Indizes zu XML-Spalten und für das Umsetzen mit XSLT-Formatvorlagen (XSLT = Extensible Stylesheet Language Transformation) gelten zusätzliche Einschränkungen.

Verwendung von 'maxOccurs'-Attributwerten über 5000: Wenn in DB2 Version 9.7 Fixpack 1 und höheren Versionen ein im DB2-XSR registriertes XML-Schema das Attribut 'maxOccurs' mit einem Wert über 5000 verwendet, wird der Attributwert für 'maxOccurs' wie die Angabe 'unbounded' (unbegrenzt) behandelt. Da Dokumentelemente mit einem 'maxOccurs'-Attributwert über 5000 entsprechend der Angabe 'unbounded' verarbeitet werden, besteht ein XML-Dokument die Prüfung bei der Verwendung der Funktion XMLVALIDATE möglicherweise erfolgreich, selbst wenn die Anzahl der Vorkommen eines Elements den durch das für die Dokumentprüfung verwendete XML-Schema festgelegten Maximalwert überschreitet. Zusätzliche Informationen und eine empfohlene Fehlerumgehung finden Sie in den Informationen zur Funktion XMLVALIDATE.

Verwendung des Befehls RESTORE DATABASE: Es ist nicht möglich, die Option **TRANSPORT** zum Transportieren von Tabellenbereichen und SQL-Schemas zu verwenden, wenn eine der Tabellen im Schema eine XML-Spalte enthält.

Anhang A. Codierungszuordnungen

Zuordnen von Codenamen zu effektiven CCSIDs für gespeicherte XML-Daten

Wenn sich Daten, die in einer XML-Spalte gespeichert werden, in einer binären Anwendungsvariable befinden, oder wenn es sich bei diesen Daten um einen intern codierten XML-Typ handelt, prüft der DB2-Datenbankmanager die Daten, um die Codierung festzustellen. Verfügen die Daten über eine Codierungsdeklaration, ordnet der Datenbankmanager den Codierungsnamen einer CCSID zu.

Tabelle 39 auf Seite 355 enthält eine Liste dieser Zuordnungen. Wenn ein Codierungsname nicht in Tabelle 39 auf Seite 355 aufgeführt ist, gibt der Datenbankmanager einen Fehler zurück.

Der normalisierte Codierungsname in der ersten Spalte von Tabelle 39 auf Seite 355 ist das Ergebnis der Konvertierung des Codierungsnamens in Großbuchstaben und das Entfernen aller Bindestriche, Pluszeichen, Unterstreichungszeichen, Doppelpunkte, Punkte und Leerzeichen. So ist ISO88591 beispielsweise der normalisierte Codierungsname für ISO 8859-1, ISO-8859-1 und iso-8859-1.

Tabelle 74. Codierungsnamen und effektive CCSIDs

Normalisierter Codierungsname	CCSID
437	437
646	367
813	813
819	819
850	850
852	852
855	855
857	857
862	862
863	863
866	866
869	869
885913	901
885915	923
88591	819
88592	912
88595	915
88597	813
88598	62210
88599	920
904	904
912	912

Tabelle 74. Codierungsnamen und effektive CCSIDs (Forts.)

Normalisierter Codierungsname	CCSID
915	915
916	916
920	920
923	923
ANSI1251	1251
ANSIX341968	367
ANSIX341986	367
ARABIC	1089
ASCII7	367
ASCII	367
ASMO708	1089
BIG5	950
CCSID00858	858
CCSID00924	924
CCSID01140	1140
CCSID01141	1141
CCSID01142	1142
CCSID01143	1143
CCSID01144	1144
CCSID01145	1145
CCSID01146	1146
CCSID01147	1147
CCSID01148	1148
CCSID01149	1149
CP00858	858
CP00924	924
CP01140	1140
CP01141	1141
CP01142	1142
CP01143	1143
CP01144	1144
CP01145	1145
CP01146	1146
CP01147	1147
CP01148	1148
CP01149	1149
CP037	37
CP1026	1026
CP1140	1140
CP1141	1141

Tabelle 74. Codierungsnamen und effektive CCSIDs (Forts.)

Normalisierter Codierungsname	CCSID
CP1142	1142
CP1143	1143
CP1144	1144
CP1145	1145
CP1146	1146
CP1147	1147
CP1148	1148
CP1149	1149
CP1250	1250
CP1251	1251
CP1252	1252
CP1253	1253
CP1254	1254
CP1255	1255
CP1256	1256
CP1257	1257
CP1258	1258
CP1363	1363
CP1383	1383
CP1386	1386
CP273	273
CP277	277
CP278	278
CP280	280
CP284	284
CP285	285
CP297	297
CP33722	954
CP33722C	954
CP367	367
CP420	420
CP423	423
CP424	424
CP437	437
CP500	500
CP5346	5346
CP5347	5347
CP5348	5348
CP5349	5349
CP5350	5350

Tabelle 74. Codierungsnamen und effektive CCSIDs (Forts.)

Normalisierter Codierungsname	CCSID
CP5353	5353
CP813	813
CP819	819
CP838	838
CP850	850
CP852	852
CP855	855
CP857	857
CP858	858
CP862	862
CP863	863
CP864	864
CP866	866
CP869	869
CP870	870
CP871	871
CP874	874
CP904	904
CP912	912
CP915	915
CP916	916
CP920	920
CP921	921
CP922	922
CP923	923
CP936	1386
CP943	943
CP943C	943
CP949	970
CP950	950
CP964	964
CP970	970
CPGR	869
CSASCII	367
CSBIG5	950
CSEBCDICCAFR	500
CSEBCDICDKNO	277
CSEBCDICES	284
CSEBCDICFISE	278
CSEBCDICFR	297

Tabelle 74. Codierungsnamen und effektive CCSIDs (Forts.)

Normalisierter Codierungsname	CCSID
CSEBCDICIT	280
CSEBCDICPT	37
CSEBCDICUK	285
CSEBCDICUS	37
CSEUCKR	970
CSEUCPKDFMTJAPANESE	954
CSGB2312	1383
CSHPROMAN8	1051
CSIBM037	37
CSIBM1026	1026
CSIBM273	273
CSIBM277	277
CSIBM278	278
CSIBM280	280
CSIBM284	284
CSIBM285	285
CSIBM297	297
CSIBM420	420
CSIBM423	423
CSIBM424	424
CSIBM500	500
CSIBM855	855
CSIBM857	857
CSIBM863	863
CSIBM864	864
CSIBM866	866
CSIBM869	869
CSIBM870	870
CSIBM871	871
CSIBM904	904
CSIBMEBCDICATDE	273
CSIBMTHAI	838
CSISO128T101G2	920
CSISO146SERBIAN	915
CSISO147MACEDONIAN	915
CSISO2INTLREFVERSION	367
CSISO646BASIC1983	367
CSISO88596I	1089
CSISO88598I	916
CSISOLATIN0	923

Tabelle 74. Codierungsnamen und effektive CCSIDs (Forts.)

Normalisierter Codierungsname	CCSID
CSISOLATIN1	819
CSISOLATIN2	912
CSISOLATIN5	920
CSISOLATIN9	923
CSISOLATINARABIC	1089
CSISOLATINCYRILLIC	915
CSISOLATINGREEK	813
CSISOLATINHEBREW	62210
CSKOI8R	878
CSKSC56011987	970
CSMACINTOSH	1275
CSMICROSOFTPUBLISHING	1004
CSPC850MULTILINGUAL	850
CSPC862LATINHEBREW	862
CSPC8CODEPAGE437	437
CSPCP852	852
CSSHIFTJIS	943
CSUCS4	1236
CSUNICODE11	1204
CSUNICODE	1204
CSUNICODEASCII	1204
CSUNICODELATIN1	1204
CSVISCII	1129
CSWINDOWS31J	943
CYRILLIC	915
DEFAULT	367
EBCDICATDE	273
EBCDICCAFR	500
EBCDICCPAR1	420
EBCDICCPBE	500
EBCDICCPCA	37
EBCDICCPCH	500
EBCDICCPDK	277
EBCDICCPES	284
EBCDICCPFI	278
EBCDICCPFR	297
EBCDICCPGB	285
EBCDICCPGR	423
EBCDICCPHE	424
EBCDICCPIS	871

Tabelle 74. Codierungsnamen und effektive CCSIDs (Forts.)

Normalisierter Codierungsname	CCSID
EBCDICCPIT	280
EBCDICCPNL	37
EBCDICCPNO	277
EBCDICCPROECE	870
EBCDICCPSE	278
EBCDICCPUS	37
EBCDICCPWT	37
EBCDICCPYU	870
EBCDICDE273EURO	1141
EBCDICDK277EURO	1142
EBCDICDKNO	277
EBCDICES284EURO	1145
EBCDICES	284
EBCDICFI278EURO	1143
EBCDICFISE	278
EBCDICFR297EURO	1147
EBCDICFR	297
EBCDICGB285EURO	1146
EBCDICINTERNATIONAL500EURO	1148
EBCDICIS871EURO	1149
EBCDICIT280EURO	1144
EBCDICIT	280
EBCDICLATIN9EURO	924
EBCDICNO277EURO	1142
EBCDICPT	37
EBCDICSE278EURO	1143
EBCDICUK	285
EBCDICUS37EURO	1140
EBCDICUS	37
ECMA114	1089
ECMA118	813
ELOT928	813
EUCCN	1383
EUCJP	954
EUCKR	970
EUCTW	964
EXTENDEDUNIXCODEPACKEDFORMATFORJAPANESE	954
GB18030	1392
GB2312	1383
GBK	1386

Tabelle 74. Codierungsnamen und effektive CCSIDs (Forts.)

Normalisierter Codierungsname	CCSID
GREEK8	813
GREEK	813
HEBREW	62210
HPROMAN8	1051
IBM00858	858
IBM00924	924
IBM01140	1140
IBM01141	1141
IBM01142	1142
IBM01143	1143
IBM01144	1144
IBM01145	1145
IBM01146	1146
IBM01147	1147
IBM01148	1148
IBM01149	1149
IBM01153	1153
IBM01155	1155
IBM01160	1160
IBM037	37
IBM1026	1026
IBM1043	1043
IBM1047	1047
IBM1252	1252
IBM273	273
IBM277	277
IBM278	278
IBM280	280
IBM284	284
IBM285	285
IBM297	297
IBM367	367
IBM420	420
IBM423	423
IBM424	424
IBM437	437
IBM500	500
IBM808	808
IBM813	813
IBM819	819

Tabelle 74. Codierungsnamen und effektive CCSIDs (Forts.)

Normalisierter Codierungsname	CCSID
IBM850	850
IBM852	852
IBM855	855
IBM857	857
IBM862	862
IBM863	863
IBM864	864
IBM866	866
IBM867	867
IBM869	869
IBM870	870
IBM871	871
IBM872	872
IBM902	902
IBM904	904
IBM912	912
IBM915	915
IBM916	916
IBM920	920
IBM921	921
IBM922	922
IBM923	923
IBMTHAI	838
IRV	367
ISO10646	1204
ISO10646UCS2	1200
ISO10646UCS4	1232
ISO10646UCSBASIC	1204
ISO10646UNICODELATIN1	1204
ISO646BASIC1983	367
ISO646IRV1983	367
ISO646IRV1991	367
ISO646US	367
ISO885911987	819
ISO885913	901
ISO885915	923
ISO885915FDIS	923
ISO88591	819
ISO885921987	912
ISO88592	912

Tabelle 74. Codierungsnamen und effektive CCSIDs (Forts.)

Normalisierter Codierungsname	CCSID
ISO885951988	915
ISO88595	915
ISO885961987	1089
ISO88596	1089
ISO88596I	1089
ISO885971987	813
ISO88597	813
ISO885981988	62210
ISO88598	62210
ISO88598I	916
ISO885991989	920
ISO88599	920
ISOIR100	819
ISOIR101	912
ISOIR126	813
ISOIR127	1089
ISOIR128	920
ISOIR138	62210
ISOIR144	915
ISOIR146	915
ISOIR147	915
ISOIR148	920
ISOIR149	970
ISOIR2	367
ISOIR6	367
JUSIB1003MAC	915
JUSIB1003SERB	915
KOI8	878
KOI8R	878
KOI8U	1168
KOREAN	970
KSC56011987	970
KSC56011989	970
KSC5601	970
L1	819
L2	912
L5	920
L9	923
LATIN0	923
LATIN1	819

Tabelle 74. Codierungsnamen und effektive CCSIDs (Forts.)

Normalisierter Codierungsname	CCSID
LATIN2	912
LATIN5	920
LATIN9	923
MAC	1275
MACEDONIAN	915
MACINTOSH	1275
MICROSOFTPUBLISHING	1004
MS1386	1386
MS932	943
MS936	1386
MS949	970
MSKANJI	943
PCMULTILINGUAL850EURO	858
R8	1051
REF	367
ROMAN8	1051
SERBIAN	915
SHIFTJIS	943
SJIS	943
SUNEUGREEK	813
T101G2	920
TIS20	874
TIS620	874
UNICODE11	1204
UNICODE11UTF8	1208
UNICODEBIGUNMARKED	1200
UNICODELITTLEUNMARKED	1202
US	367
USASCII	367
UTF16	1204
UTF16BE	1200
UTF16LE	1202
UTF32	1236
UTF32BE	1232
UTF32LE	1234
UTF8	1208
VISCII	1129
WINDOWS1250	1250
WINDOWS1251	1251
WINDOWS1252	1252

Tabelle 74. Codierungsnamen und effektive CCSIDs (Forts.)

Normalisierter Codierungsname	CCSID
WINDOWS1253	1253
WINDOWS1254	1254
WINDOWS1255	1255
WINDOWS1256	1256
WINDOWS1257	1257
WINDOWS1258	1258
WINDOWS28598	62210
WINDOWS31J	943
WINDOWS936	1386
XEUCTW	964
XMSWIN936	1386
XUTF16BE	1200
XUTF16LE	1202
XWINDOWS949	970

Zuordnen von CCSIDs zu Codenamen für serialisierte XML-Ausgabedaten

Als Teil einer impliziten oder expliziten XMLSERIALIZE-Operation fügt der DB2-Datenbankmanager möglicherweise eine Codierungsdeklaration am Anfang von serialisierten XML-Ausgabedaten hinzu.

Diese Deklaration hat das folgende Format:

```
<?xml version="1.0" encoding="codierung-name"?>
```

Im Allgemeinen beschreibt die Zeichensatzkennung in der Codierungsdeklaration die Codierung der Zeichen in der Ausgabezeichenfolge. Wenn beispielsweise XML-Daten für die CCSID serialisiert werden, die dem Datentyp der Zielanwendung entspricht, beschreibt die Codierungsdeklaration die CCSID-Zielanwendungsvariable. Eine Ausnahme stellt der Fall dar, in dem die Anwendung eine explizite Funktion XMLSERIALIZE mit der Option INCLUDING XMLDECLARATION ausführt. Bei der Angabe von INCLUDING XMLDECLARATION generiert der Datenbankmanager eine Codierungsdeklaration für UTF-8. Wenn der Zieldatentyp ein CLOB- oder DBCLOB-Typ ist, findet möglicherweise eine zusätzliche Codepagekonvertierung statt, wodurch die Codierungsinformationen ungenau werden können. Wenn in der Anwendung eine weitere syntaktische Analyse erfolgt, führt dies möglicherweise zu fehlerhaften Daten.

Wenn möglich, verwendet der DB2-Datenbankmanager den IANA-Registrynamen für die CCSID, wie durch den XML-Standard vorgeschrieben.

Tabelle 75. CCSIDs und entsprechende Codierungsnamen

CCSID	Codierungsname
37	IBM037
273	IBM273
277	IBM277

Tabelle 75. CCSIDs und entsprechende Codierungsnamen (Forts.)

CCSID	Codierungsname
278	IBM278
280	IBM280
284	IBM284
285	IBM285
297	IBM297
367	US-ASCII
420	IBM420
423	IBM423
424	IBM424
437	IBM437
500	IBM500
808	IBM808
813	ISO-8859-7
819	ISO-8859-1
838	IBM-Thai
850	IBM850
852	IBM852
855	IBM855
857	IBM857
858	IBM00858
862	IBM862
863	IBM863
864	IBM864
866	IBM866
867	IBM867
869	IBM869
870	IBM870
871	IBM871
872	IBM872
874	TIS-620
878	KOI8-R
901	ISO-8859-13
902	IBM902
904	IBM904
912	ISO-8859-2
915	ISO-8859-5
916	ISO-8859-8-I
920	ISO-8859-9
921	IBM921
922	IBM922

Tabelle 75. CCSIDs und entsprechende Codierungsnamen (Forts.)

CCSID	Codierungsname
923	ISO-8859-15
924	IBM00924
932	Shift_JIS
943	Windows-31J
949	EUC-KR
950	Big5
954	EUC-JP
964	EUC-TW
970	EUC-KR
1004	Microsoft-Publish
1026	IBM1026
1043	IBM1043
1047	IBM1047
1051	hp-roman8
1089	ISO-8859-6
1129	VISCII
1140	IBM01140
1141	IBM01141
1142	IBM01142
1143	IBM01143
1144	IBM01144
1145	IBM01145
1146	IBM01146
1147	IBM01147
1148	IBM01148
1149	IBM01149
1153	IBM01153
1155	IBM01155
1160	IBM-Thai
1161	TIS-620
1162	TIS-620
1163	VISCII
1168	KOI8-U
1200	UTF-16BE
1202	UTF-16LE
1204	UTF-16
1208	UTF-8
1232	UTF-32BE
1234	UTF-32LE
1236	UTF-32

Tabelle 75. CCSIDs und entsprechende Codierungsnamen (Forts.)

CCSID	Codierungsname
1250	windows-1250
1251	windows-1251
1252	windows-1252
1253	windows-1253
1254	windows-1254
1255	windows-1255
1256	windows-1256
1257	windows-1257
1258	windows-1258
1275	MACINTOSH
1363	KSC_5601
1370	Big5
1381	GB2312
1383	GB2312
1386	GBK
1392	GB18030
4909	ISO-8859-7
5039	Shift_JIS
5346	windows-1250
5347	windows-1251
5348	windows-1252
5349	windows-1253
5350	windows-1254
5351	windows-1255
5352	windows-1256
5353	windows-1257
5354	windows-1258
5488	GB18030
8612	IBM420
8616	IBM424
9005	ISO-8859-7
12712	IBM424
13488	UTF-16BE
13490	UTF-16LE
16840	IBM420
17248	IBM864
17584	UTF-16BE
17586	UTF-16LE
62209	IBM862
62210	ISO-8859-8

Tabelle 75. CCSIDs und entsprechende Codierungsnamen (Forts.)

CCSID	Codierungsname
62211	IBM424
62213	IBM862
62215	ISO-8859-8
62218	IBM864
62221	IBM862
62222	ISO-8859-8
62223	windows-1255
62224	IBM420
62225	IBM864
62227	ISO-8859-6
62228	windows-1256
62229	IBM424
62231	IBM862
62232	ISO-8859-8
62233	IBM420
62234	IBM420
62235	IBM424
62237	windows-1255
62238	ISO-8859-8-I
62239	windows-1255
62240	IBM424
62242	IBM862
62243	ISO-8859-8-I
62244	windows-1255
62245	IBM424
62250	IBM420

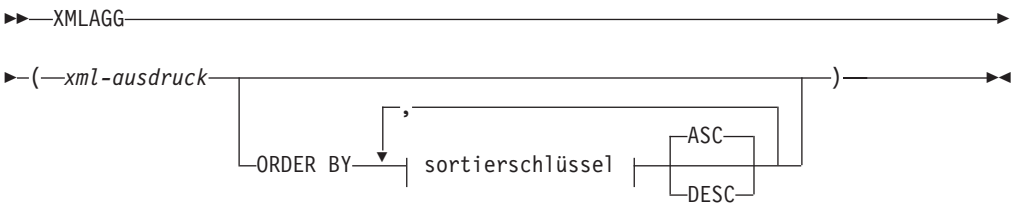
Anhang B. SQL/XML-Veröffentlichungsfunktionen

In den folgenden Abschnitten wird die Syntax der SQL/XML-Veröffentlichungsfunktionen von DB2 beschrieben.

Informationen zur Verwendung der Funktionen finden Sie unter „SQL/XML-Veröffentlichungsfunktionen für das Erstellen von XML-Werten" auf Seite 132.

XMLAGG

Die Funktion XMLAGG gibt eine XML-Sequenz zurück, die ein Element für jeden Wert ungleich Null in einer Menge von XML-Werten enthält.



Das Schema heißt SYSIBM. Der Funktionsname kann nicht als qualifizierter Name angegeben werden.

xml-ausdruck
Gibt einen Ausdruck des Datentyps XML an.

ORDER BY
Gibt die Reihenfolge der Zeilen aus demselben Gruppierungsset an, die in der Spaltenberechnung verarbeitet werden. Wenn die Klausel ORDER BY weggelassen wird oder die Reihenfolge der Spaltendaten nicht unterscheiden kann, werden die Zeilen in demselben Gruppierungsset willkürlich geordnet.

sortierschlüssel
Der Sortierschlüssel kann ein Spaltenname oder ein Sortierschlüsselausdruck (*sortierschlüsselausdruck*) sein. Beachten Sie Folgendes: Wenn der Sortierschlüssel eine Konstante ist, verweist er nicht auf die Position der Ausgabespalte (wie in der normalen Klausel ORDER BY), sondern stellt einfach eine Konstante dar, was keinen Sortierschlüssel impliziert.

Der Datentyp des Ergebnisses ist XML.

Die Funktion wird auf die Wertegruppe angewendet, die durch den Ausschluss von Nullwerten aus den Argumentwerten abgeleitet wurde.

Wenn das Argument *xml-ausdruck* Null sein kann, dann kann das Ergebnis gleich Null sein. Wenn die Wertegruppe leer ist, ergibt sich als Ergebnis ein Nullwert. Andernfalls ist das Ergebnis eine XML-Sequenz, die ein Element für jeden Wert in der Gruppe enthält.

Falls eine SELECT-Klausel eine ARRAY_AGG-Funktion umfasst, dann müssen alle Aufrufe der Funktionen ARRAY_AGG, LISTAGG, XMLAGG und XMLGROUP in derselben SELECT-Klausel dieselbe Reihenfolge oder keine Reihenfolge angeben (SQLSTATE 428GZ).

Anmerkungen

- **Unterstützung in OLAP-Ausdrücken**: XMLAGG kann nicht als Spaltenfunktion einer OLAP-Spaltenberechnungsfunktion verwendet werden (SQLSTATE 42601).

Beispiel

Erstellen Sie für jede Abteilung ein Element 'Department', das eine Liste mit den nach Nachname sortierten Mitarbeitern enthält.

```
SELECT XMLSERIALIZE(
  CONTENT XMLELEMENT(
    NAME "Department", XMLATTRIBUTES(
      E.WORKDEPT AS "name"
    ),
    XMLAGG(
      XMLELEMENT(
        NAME "emp", E.LASTNAME
      )
      ORDER BY E.LASTNAME
    )
  )
  AS CLOB(110)
)
AS "dept_list"
FROM EMPLOYEE E
WHERE E.WORKDEPT IN ('C01','E21')
GROUP BY WORKDEPT
```

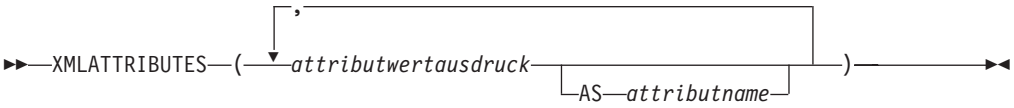
Diese Abfrage führt zu folgendem Ergebnis:

```
dept_list
----------------------...
<Department name="C01">
  <emp>KWAN</emp>
  <emp>NICHOLLS</emp>
  <emp>QUINTANA</emp>
</Department>
<Department name="E21">
  <emp>GOUNOT</emp>
  <emp>LEE</emp>
  <emp>MEHTA</emp>
  <emp>SPENSER</emp>
</Department>
```

Anmerkung: XMLAGG fügt keine Leerzeichen oder Zeilenvorschubzeichen in die Ausgabe ein. Die gesamte Beispielausgabe wurde zur besseren Lesbarkeit formatiert.

XMLATTRIBUTES

Die Funktion XMLATTRIBUTES konstruiert XML-Attribute aus den Argumenten.



Das Schema heißt SYSIBM. Der Funktionsname kann nicht als qualifizierter Name angegeben werden.

Diese Funktion kann nur als Argument der Funktion XMLELEMENT verwendet werden. Das Ergebnis ist eine XML-Sequenz, die für jeden Eingabewert ungleich Null einen XQuery-Attributknoten enthält.

attributwertausdruck
Eine Ausdruck, dessen Ergebnis der Attributwert ist. Der Datentyp von *attributwertausdruck* darf kein strukturierter Typ sein (SQLSTATE 42884). Der Ausdruck kann ein beliebiger SQL-Ausdruck sein. Wenn der Ausdruck kein einfacher Spaltenverweis ist, muss ein Attributname angegeben werden.

attributname
Gibt einen Attributnamen an. Der Name ist eine SQL-Kennung, die das Format eines qualifizierten XML-Namens (QName) haben muss (SQLSTATE 42634). Weitere detaillierte Informationen zu gültigen Namen finden Sie in W3C-XML-Namensbereichsspezifikationen. Der Attributname darf nicht `xmlns` lauten oder das Präfix `xmlns:` enthalten. Ein Namensbereich wird mit der Funktion XMLNAMESPACES deklariert. Doppelte Attributnamen sind nicht zulässig, weder implizit noch explizit (SQLSTATE 42713).

Wenn *attributname* nicht angegeben wird, muss *attributwertausdruck* ein Spaltenname sein (SQLSTATE 42703). Der Attributname wird über die Zuordnung mit vollständiger Zeichenersetzung aus einem Spaltennamen zu einem XML-Attributnamen erstellt.

Der Datentyp des Ergebnisses ist XML. Wenn das Ergebnis von *attributwertausdruck* Null sein kann, dann kann das Ergebnis Null sein. Wenn jedes Vorkommen von *attributwertausdruck* Null als Ergebnis hat, ist das Ergebnis der Nullwert.

Beispiele

Anmerkung: XMLATTRIBUTES fügt keine Leerzeichen oder Zeilenvorschubzeichen in die Ausgabe ein. Die gesamte Beispielausgabe wurde zur besseren Lesbarkeit formatiert.

- *Beispiel 1*: Element mit Attributen erzeugen.

```
SELECT E.EMPNO, XMLELEMENT(
  NAME "Emp",
  XMLATTRIBUTES(
    E.EMPNO, E.FIRSTNME ||' '|| E.LASTNAME AS "name"
  )
)
AS "Result"
FROM EMPLOYEE E WHERE E.EDLEVEL = 12
```

Diese Abfrage führt zu folgendem Ergebnis:

```
EMPNO  Result
000290 <Emp EMPNO="000290" name="JOHN PARKER"></Emp>
000310 <Emp EMPNO="000310" name="MAUDE SETRIGHT"></Emp>
200310 <Emp EMPNO="200310" name="MICHELLE SPRINGER"></Emp>
```

- *Beispiel 2*: Element mit einer Namensbereichsdeklaration erzeugen, die in keinem QName verwendet wird. Das Präfix wird in einem Attributwert verwendet.

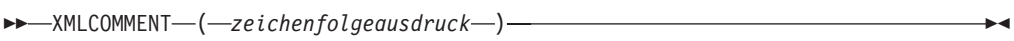
```
VALUES XMLELEMENT(
  NAME "size",
  XMLNAMESPACES(
    'http://www.w3.org/2001/XMLSchema-instance' AS "xsi",
    'http://www.w3.org/2001/XMLSchema' AS "xsd"
  ),
  XMLATTRIBUTES(
    'xsd:string' AS "xsi:type"
  ), '1'
)
```

Diese Abfrage führt zu folgendem Ergebnis:

```
<size xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xsi:type="xsd:string">1</size>
```

XMLCOMMENT

Die Funktion XMLCOMMENT gibt einen XML-Wert mit einem einzigen XQuery-Kommentarknoten und dem Eingabeargument als Inhalt zurück.



Das Schema heißt SYSIBM. Der Funktionsname kann nicht als qualifizierter Name angegeben werden.

zeichenfolgeausdruck
Ein Ausdruck, dessen Wert einen Zeichenfolgetyp hat: CHAR, VARCHAR oder CLOB. Das Ergebnis von *zeichenfolgeausdruck* wird syntaktisch analysiert, um die Konformität mit den Anforderungen an ein XML-Dokument gemäß den Regeln von XML 1.0 zu prüfen. Das Ergebnis von *zeichenfolgeausdruck* muss dem folgenden regulären Ausdruck entsprechen:

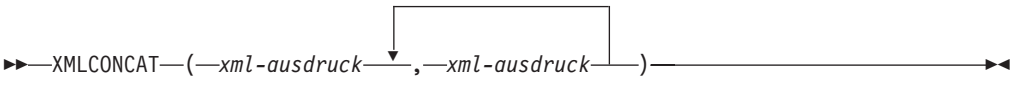
```
((Char - '-') | ('-' (Char - '-')))*
```

Dabei steht `Char` für ein beliebiges Unicode-Zeichen mit Ausnahme der Ersatzzeichenblöcke X'FFFE' und X'FFFF'. Der XML-Kommentar darf also keine zwei Bindestriche hintereinander enthalten und nicht mit einem Bindestrich enden (SQLSTATE 2200S).

Der Datentyp des Ergebnisses ist XML. Wenn das Ergebnis von *zeichenfolgeausdruck* Null sein kann, dann kann das Ergebnis Null sein. Wenn die Eingabe den Wert Null hat, ist das Ergebnis der Nullwert.

XMLCONCAT

Die Funktion XMLCONCAT gibt eine Sequenz mit der Verkettung einer variablen Anzahl von XML-Eingabeargumenten zurück.



Das Schema heißt SYSIBM. Der Funktionsname kann nicht als qualifizierter Name angegeben werden.

xml-ausdruck
Gibt einen Ausdruck des Datentyps XML an.

Der Datentyp des Ergebnisses ist XML. Das Ergebnis ist eine XML-Zeichenfolge mit der Verknüpfung der XML-Eingabewerte, die ungleich null sind. Nullwerte bei der Eingabe werden ignoriert. Wenn das Ergebnis von *xml-ausdruck* immer Null sein kann, dann kann das Ergebnis Null sein. Ist das Ergebnis jeder Eingabe Null, ist das Ergebnis der Nullwert.

Beispiel

Anmerkung: XMLCONCAT fügt keine Leerzeichen oder Zeilenvorschubzeichen in die Ausgabe ein. Die gesamte Beispielausgabe wurde zur besseren Lesbarkeit formatiert.

Erstellen Sie für die Abteilungen A00 und B01 ein Element 'Department', das eine Liste mit den nach Vorname sortierten Mitarbeitern enthält. Binden Sie direkt vor dem Abteilungselement einen einführenden Kommentar ein.

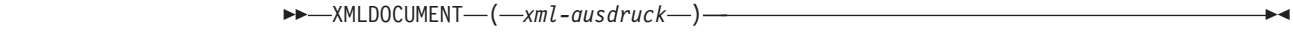
```
SELECT XMLCONCAT(
  XMLCOMMENT(
    'Confirm these employees are on track for their product schedule'
  ),
  XMLELEMENT(
    NAME "Department",
    XMLATTRIBUTES(
      E.WORKDEPT AS "name"
    ),
    XMLAGG(
      XMLELEMENT(
        NAME "emp", E.FIRSTNME
      )
    ORDER BY E.FIRSTNME
    )
  )
)
FROM EMPLOYEE E
WHERE E.WORKDEPT IN ('A00', 'B01')
GROUP BY E.WORKDEPT
```

Diese Abfrage führt zu folgendem Ergebnis:

```
<!--Confirm these employees are on track for their product schedule-->
<Department name="A00">
<emp>CHRISTINE</emp>
<emp>DIAN</emp>
<emp>GREG</emp>
<emp>SEAN</emp>
<emp>VINCENZO</emp>
</Department>
<!--Confirm these employees are on track for their product schedule-->
<Department name="B01">
<emp>MICHAEL</emp>
</Department>
```

XMLDOCUMENT

Die Funktion XMLDOCUMENT gibt einen XML-Wert mit einem einzigen XQuery-Dokumentknoten mit null oder mehr untergeordneten Knoten zurück.



Das Schema heißt SYSIBM. Der Funktionsname kann nicht als qualifizierter Name angegeben werden.

xml-ausdruck
Ein Ausdruck, der einen XML-Wert zurückgibt. Ein Sequenzelement in dem XML-Wert darf kein Attributknoten sein (SQLSTATE 10507).

Der Datentyp des Ergebnisses ist XML. Wenn das Ergebnis von *xml-ausdruck* Null sein kann, dann kann das Ergebnis Null sein. Wenn die Eingabe den Wert Null hat, ist das Ergebnis der Nullwert.

Die untergeordneten Elemente des resultierenden Dokumentknotens werden entsprechend der Beschreibung in den folgenden Schritten erstellt. Der Eingabeausdruck ist eine Sequenz von Knoten oder atomaren Werten, die in diesen Schritten als Inhaltssequenz bezeichnet werden.

1. Wenn die Inhaltssequenz einen Dokumentknoten enthält, wird dieser Dokumentknoten in der Inhaltssequenz durch seine untergeordneten Elemente ersetzt.
2. Jede benachbarte Sequenz mit einem oder mehreren atomaren Werten in der Inhaltssequenz wird durch einen Textknoten ersetzt, der das Ergebnis der Umsetzung der einzelnen atomaren Werte in eine Zeichenfolge enthält, wobei zwischen benachbarten Werten ein einzelnes Leerzeichen eingefügt wird.
3. Für jeden Knoten in der Inhaltssequenz wird eine neue tiefe Kopie des Knotens erstellt. Unter einer tiefen Kopie eines Knotens versteht man eine Kopie der gesamten Unterverzeichnisstruktur, die von diesem Knoten ausgeht, einschließlich des Knotens selbst und seiner Nachkommen. Jeder kopierte Knoten hat eine neue Knotenidentität.
4. Die Knoten in der Inhaltssequenz werden zu den untergeordneten Elementen des neuen Dokumentknotens.

Die Funktion XMLDOCUMENT führt den mit XQuery berechneten Dokumentkonstruktor effektiv aus. Das Ergebnis von

```
XMLQUERY('document {$E}' PASSING BY REF XML-ausdruck AS "E")
```

ist gleich gleichwertig mit

```
XMLDOCUMENT( XML-ausdruck )
```

außer in den Fällen, in denen *xml-ausdruck* gleich Null ist und XMLQUERY die leere Sequenz zurückgibt, im Gegensatz zu XMLDOCUMENT, was den Nullwert zurückgibt.

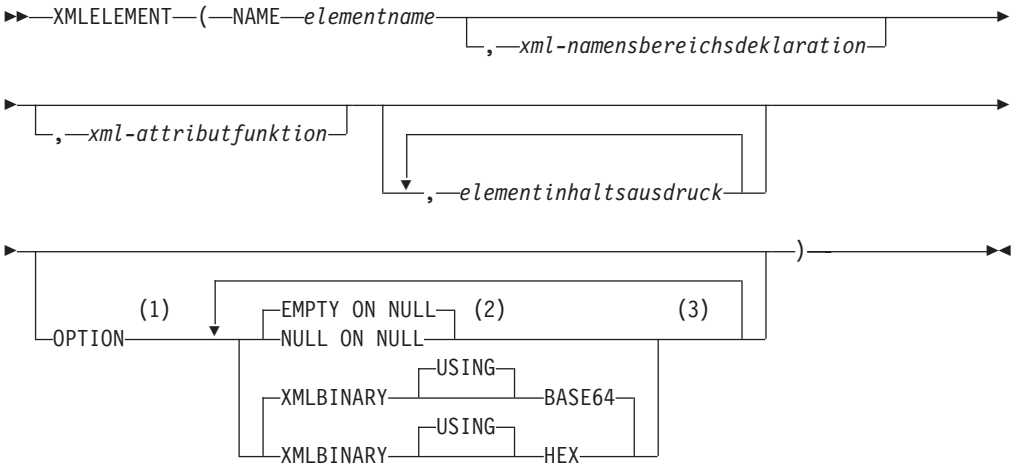
Beispiel

Fügen Sie ein erstelltes Dokument in eine XML-Spalte ein.

```
INSERT INTO T1 VALUES(
  123, (
    SELECT XMLDOCUMENT(
      XMLELEMENT(
        NAME "Emp", E.FIRSTNME || ' ' || E.LASTNAME, XMLCOMMENT(
          'This is just a simple example'
        )
      )
    )
    FROM EMPLOYEE E
    WHERE E.EMPNO = '000120'
  )
)
```

XMLELEMENT

Die Funktion XMLELEMENT gibt einen XML-Wert zurück, der ein XQuery-Elementknoten ist.



Anmerkungen:

1 Die Klausel OPTION kann nur angegeben werden, wenn mindestens ein Vorkommen von *xml-attributfunktion* oder *elementinhaltsausdruck* angegeben wird.

2 NULL ON NULL oder EMPTY ON NULL kann nur angegeben werden, wenn mindestens ein Vorkommen von *elementinhaltsausdruck* angegeben wird.

3 Eine Klausel darf nicht mehrmals angegeben werden.

Das Schema heißt SYSIBM. Der Funktionsname kann nicht als qualifizierter Name angegeben werden.

NAME *elementname*
Gibt den Namen eines XML-Elements an. Der Name ist eine SQL-Kennung, die das Format eines qualifizierten XML-Namens (QName) haben muss (SQLSTATE 42634). Weitere detaillierte Informationen zu gültigen Namen finden Sie in W3C-XML-Namensbereichsspezifikationen. Wenn der Name qualifiziert ist, muss das Namensbereichspräfix innerhalb des Geltungsbereichs deklariert werden (SQLSTATE 42635).

xml-namensbereichsdeklaration
Gibt die XML-Namensbereichsdeklarationen an, die das Ergebnis der Deklaration XMLNAMESPACES darstellen. Die deklarierten Namensbereichen befinden sich im Geltungsbereich der Funktion XMLELEMENT. Die Namensbereiche gelten für alle in der Funktion XMLELEMENT verschachtelten XML-Funktionen unabhängig davon, ob sie in einem anderen Subselect angezeigt werden.

Wird *xml-namensbereichsdeklaration* nicht angegeben, werden Namensbereichsdeklarationen dem erstellten Element nicht zugeordnet.

xml-attributfunktion
Gibt die XML-Attribute für das Element an. Die Attribute sind das Ergebnis der Funktion XMLATTRIBUTES.

elementinhaltsausdruck
Der Inhalt des generierten XML-Elementknotens wird mit einem Ausdruck

oder einer Liste von Ausdrücken angegeben. Der Datentyp von *elementinhaltsausdruck* kann kein strukturierter Typ sein (SQLSTATE 42884). Der Ausdruck kann ein beliebiger SQL-Ausdruck sein.

Wird *elementinhaltsausdruck* nicht angegeben, wird eine leere Zeichenfolge als Inhalt des Elements verwendet, und OPTION NULL ON NULL oder EMPTY ON NULL darf nicht angegeben werden.

OPTION
Gibt Zusatzoptionen zum Erstellen des XML-Elements an. Wenn keine Klausel OPTION angegeben wird, ist EMPTY ON NULL XMLBINARY USING BASE64 der Standardwert. Diese Klausel hat keinen Einfluss auf verschachtelte Aufrufe von XMLELEMENT, die in *elementinhaltsausdruck* angegeben werden.

EMPTY ON NULL oder NULL ON NULL
Gibt an, ob ein Nullwert oder ein leeres Element zurückgegeben werden soll, wenn jedes Vorkommen von *elementinhaltsausdruck* einen Nullwert hat. Diese Option wirkt sich nur auf die Nullwertbehandlung von Elementinhalt aus, nicht auf Attributwerte. Der Standardwert ist EMPTY ON NULL.

EMPTY ON NULL
Wenn der Wert jedes Vorkommens von *elementinhaltsausdruck* Null ist, wird ein leeres Element zurückgegeben.

NULL ON NULL
Wenn der Wert jedes Vorkommens von *elementinhaltsausdruck* Null ist, wird ein Nullwert zurückgegeben.

XMLBINARY USING BASE64 oder XMLBINARY USING HEX
Gibt die angenommene Codierung von binären Eingabedaten, Zeichenfolgedaten mit dem Attribut FOR BIT DATA oder von einem einzigartigen Datentyp an, der auf einem dieser Typen basiert. Die Codierung gilt für den Elementinhalt oder für die Attributwerte. Der Standardwert ist XMLBINARY USING BASE64.

XMLBINARY USING BASE64
Gibt an, dass die angenommene Codierung base64-Zeichen sind, wie für die xs:base64Binary-Codierung des XML-Schematyps definiert. Die Codierung base64 verwendet eine Untermenge von US-ASCII mit 65 Zeichen (10 Ziffern, 26 Kleinbuchstaben, 26 Großbuchstaben, '+' und '/') und stellt damit alle 6 Bit der Binär- oder Bitdaten mit einem einzigen druckbaren Zeichen in der Untermenge dar. Diese Zeichen werden ausgewählt und sind dadurch universell darstellbar. Bei Verwendung dieser Methode werden die codierten Daten um 33 % größer als die ursprünglichen Binär- oder Bitdaten.

XMLBINARY USING HEX
Gibt an, dass die angenommene Codierung Hexadezimalzeichen sind, wie für die xs:hexBinary-Codierung des XML-Schematyps definiert. Bei der hexadezimalen Codierung wird jedes Byte (8 Bit) mit zwei Hexadezimalzeichen dargestellt. Bei Verwendung dieser Methode werden die codierten Daten doppelt so groß wie die ursprünglichen Binär- oder Bitdaten.

Diese Funktion nimmt einen Elementnamen, eine optionale Gruppe von Namensbereichsdeklarationen, eine optionale Gruppe von Attributen und null oder mehr Argumente an, die den Inhalt des XML-Elements bilden. Das Ergebnis ist eine XML-Sequenz, die einen XML-Elementknoten oder den Nullwert enthält.

Der Datentyp des Ergebnisses ist XML. Wenn irgendeines der Argumente von *elementinhaltsausdruck* Null sein kann, dann kann das Ergebnis Null sein. Wenn alle Argumentwerte von *elementinhaltsausdruck* Null sind und die Option NULL ON NULL wirksam ist, dann ist das Ergebnis der Nullwert.

Anmerkungen

- Beim Erstellen von Elementen, die als Inhalt eines anderen Elements kopiert werden, das Namensbereiche definiert, sollte die Deklaration der Standardnamensbereiche explizit aufgehoben werden, um mögliche Fehler zu vermeiden, die bei der Übernahme des Standardnamensbereichs aus dem neuen übergeordneten Element auftreten könnten. Die vordefinierten Namensbereichspräfixe ('xs', 'xsi', 'xml' und 'sqlxml') müssen ebenfalls explizit deklariert werden, wenn sie verwendet werden.
- **Erstellen eines Elementknotens**: Der resultierende Elementknoten wird wie folgt erstellt:
  1. Mit *xml-namensbereichsdeklaration* wird eine Gruppe gültiger Namensbereiche für das erstellte Element hinzugefügt. Jeder gültige Namensbereich ordnet ein Namensbereichspräfix (oder den Standardnamensbereich) einer Namensbereichs-URI zu. Die gültigen Namensbereiche definieren die Gruppe der Namensbereichspräfixe, die für das Interpretieren qualifizierter Namen (QNames) innerhalb des Geltungsbereichs des Elements verfügbar sind.
  2. Bei Angabe des Arguments 'xml-attributfunktion' wird dieses ausgewertet, und das Ergebnis ist eine Sequenz von Attributknoten.
  3. Jedes Vorkommen von *elementinhaltsausdruck* wird ausgewertet, und das Ergebnis wird wie folgt in eine Sequenz von Knoten konvertiert:
     - Ist der Ergebnistyp nicht XML, wird er in einen XML-Textknoten konvertiert, dessen Inhalt das Ergebnis der Zuordnung von *elementinhaltsausdruck* zu XML entsprechend den Regeln für die Zuordnung von SQL-Datenwerten zu XML-Datenwerten ist (weitere Informationen enthält die Tabelle mit der Beschreibung der unterstützten Umsetzungen von Nicht-XML-Werten in XML-Werte im Abschnitt „Umsetzung zwischen Datentypen").
     - Ist der Ergebnistyp XML, ist das Ergebnis im Allgemeinen eine Sequenz von Elementen. Möglicherweise sind einige der Elemente in dieser Sequenz Dokumentknoten. Jeder Dokumentknoten in der Sequenz wird durch die Sequenz der ihm direkt untergeordneten Elemente ersetzt. Anschließend wird für jeden Knoten in der resultierenden Sequenz eine neue tiefe Kopie des Knotens erstellt, einschließlich seiner untergeordneten Elemente und seiner Attribute. Jeder kopierte Knoten hat eine neue Knotenidentität. Kopierte Element- und Attributknoten behalten ihre Typenannotation bei. Für jede benachbarte Sequenz mit einem oder mehreren atomaren Werten, die in der Sequenz zurückgegeben werden, wird ein neuer Textknoten erstellt, der das Ergebnis der Umsetzung der einzelnen atomaren Werte in eine Zeichenfolge enthält, wobei zwischen benachbarten Werten ein einzelnes Leerzeichen eingefügt wird. Benachbarte Textknoten in der Inhaltssequenz werden zu einem einzigen Textknoten zusammengeführt, indem ihr Inhalt ohne trennende Leerzeichen verknüpft wird. Nach der Verknüpfung werden alle Textknoten, deren Inhalt eine Zeichenfolge mit Nulllänge ist, aus der Inhaltssequenz gelöscht.
  4. Die Ergebnissequenz der XML-Attribute und die resultierenden Sequenzen aller Spezifikationen von *elementinhaltsausdruck* werden zu einer einzigen Sequenz verknüpft, die als Inhaltssequenz bezeichnet wird. Alle Sequenzen von benachbarten Textknoten in der Inhaltssequenz werden zu einem einzigen Textknoten zusammengeführt. Wenn alle Argumente von *elementinhaltsaus-*

druck leere Zeichenfolgen sind oder ein Argument von *elementinhaltsausdruck* nicht angegeben wird, wird ein leeres Element zurückgegeben.

5. Die Inhaltssequenz darf keinen Attributknoten enthalten, der auf einen Knoten folgt, der kein Attributknoten ist (SQLSTATE 10507). Die Attributknoten, die in der Inhaltssequenz vorkommen, werden zu den Attributen des neuen Elementknotens. Zwei oder mehr dieser Attributknoten dürfen nicht denselben Namen haben (SQLSTATE 10503). Es wird eine Namensbereichsdeklaration entsprechend den in den Namen der Attributknoten verwendeten Namensbereichen erstellt, falls sich die Namensbereichs-URI nicht in den gültigen Namensbereichen des erstellten Elements befindet.
6. Die Element-, Text-, Kommentar- und Verarbeitungsanweisungsknoten in der Inhaltssequenz werden zu den untergeordneten Elementen des erstellten Elementknotens.
7. Dem erstellten Elementknoten wird die Typenannotation `xs:anyType` zugeordnet, jedes seiner Attribute erhält die Typenannotation `xdt:untypedAtomic`. Der Knotenname des erstellten Elementknotens ist 'element-name' und muss nach dem Schlüsselwort NAME angegeben werden.

- **Regeln für die Verwendung von Namensbereichen in XMLELEMENT**: Beachten Sie die folgenden Regeln für die Gültigkeit von Namensbereichen.
  - Die in der Deklaration XMLNAMESPACES deklarierten Namensbereiche sind die gültigen Namensbereiche des mit der Funktion XMLELEMENT erstellten Elementknotens. Wird der Elementknoten serialisiert, wird jeder seiner gültigen Namensbereiche als Namensbereichsattribut serialisiert, sofern es sich nicht um einen gültigen Namensbereich des übergeordneten Elements des Elementknotens handelt und dieses Element ebenfalls serialisiert wird.
  - Wenn in einem Argument von *elementinhaltsausdruck* ein Schlüsselwort XMLQUERY oder XMLEXISTS vorkommt, werden die Namensbereiche zu den statistisch bekannten Namensbereichen des XQuery-Ausdrucks von XMLQUERY oder XMLEXISTS. Statistisch bekannte Namensbereiche werden zur Auflösung der qualifizierten Namen (QNames) in dem XQuery-Ausdruck verwendet. Wird im XQuery-Prolog ein Namensbereich mit demselben Präfix im Geltungsbereich des XQuery-Ausdrucks deklariert, überschreibt der im Prolog deklarierte Namensbereich die in der Deklaration XMLNAMESPACES deklarierten Namensbereiche.
  - Wenn ein Attribut des erstellten Elements aus einem Elementinhaltsausdruck (*elementinhaltsausdruck*) stammt, ist dessen Namensbereich möglicherweise noch nicht als gültiger Namensbereich des erstellten Elements deklariert. In diesen Fall wird für das Attribut ein neuer Namensbereich erstellt. Wenn dies zu einem Konflikt führen würde, weil das Präfix des Attributnamens bereits durch einen gültigen Namensbereich an eine anderen URI gebunden ist, generiert DB2 ein Präfix, das keinen solchen Konflikt verursacht. Das in dem Attributnamen verwendete Präfix wird in das neue Präfix geändert, für das ein Namensbereich erstellt wird. Das generierte Präfix hat folgendes Muster: "db2ns-xx". Dabei ist "x" ein aus der Gruppe [A-Z,a-z,0-9] gewähltes Zeichen. Beispiel:

    ```
    VALUES XMLELEMENT(
      NAME "c", XMLQUERY(
        'declare namespace ipo="www.ipo.com"; $m/ipo:a/@ipo:b'
          PASSING XMLPARSE(
            DOCUMENT '<tst:a xmlns:tst="www.ipo.com" tst:b="2"/>'
          ) AS "m"
      )
    )
    ```

    Rückgabe:

```
<c xmlns:tst="www.ipo.com" tst:b="2"/>
```

Weiteres Beispiel:

```
VALUES XMLELEMENT(
  NAME "tst:c", XMLNAMESPACES(
    'www.tst.com' AS "tst"
  ),
  XMLQUERY(
    'declare namespace ipo="www.ipo.com"; $m/ipo:a/@ipo:b'
      PASSING XMLPARSE(
        DOCUMENT '<tst:a xmlns:tst="www.ipo.com" tst:b="2"/>'
      ) AS "m"
  )
)
```

Rückgabe:

```
<tst:c xmlns:tst="www.tst.com" xmlns:db2ns-a1="www.ipo.com"
  db2ns-a1:b="2"/>
```

Beispiele

Anmerkung: XMLELEMENT fügt keine Leerzeichen oder Zeilenvorschubzeichen in die Ausgabe ein. Die gesamte Beispielausgabe wurde zur besseren Lesbarkeit formatiert.

- *Beispiel 1*: Erstellen Sie ein Element mit der Option NULL ON NULL.

```
SELECT E.FIRSTNME, E.LASTNAME, XMLELEMENT(
  NAME "Emp", XMLELEMENT(
    NAME "firstname", E.FIRSTNME
  ),
  XMLELEMENT(
    NAME "lastname", E.LASTNAME
  )
  OPTION NULL ON NULL
)
AS "Result"
FROM EMPLOYEE E
WHERE E.EDLEVEL = 12
```

Diese Abfrage führt zu folgendem Ergebnis:

```
FIRSTNME     LASTNAME        Emp
JOHN         PARKER          <Emp><firstname>JOHN</firstname>
                               <lastname>PARKER</lastname></Emp>
MAUDE        SETRIGHT        <Emp><firstname>MAUDE</firstname>
                               <lastname>SETRIGHT</lastname></Emp>
MICHELLE     SPRINGER        <Emp><firstname>MICHELLE</firstname>
                               <lastname>SPRINGER</lastname></Emp>
```

- *Beispiel 2*: Erzeugen Sie ein Element mit einer Liste von Elementen, die als untergeordnete Elemente verschachtelt sind.

```
SELECT XMLELEMENT(
  NAME "Department", XMLATTRIBUTES(
    E.WORKDEPT AS "name"
  ),
  XMLAGG(
    XMLELEMENT(
      NAME "emp", E.FIRSTNME
    )
    ORDER BY E.FIRSTNME
  )
)
```

```
   AS "dept_list"
   FROM EMPLOYEE E
   WHERE E.WORKDEPT IN ('A00', 'B01')
   GROUP BY WORKDEPT
```

Diese Abfrage führt zu folgendem Ergebnis:

```
dept_list
<Department name="A00">
<emp>CHRISTINE</emp>
<emp>SEAN</emp>
<emp>VINCENZO</emp>
</Department>
<Department name="B01">
<emp>MICHAEL</emp>
</Department>
```

- *Beispiel 3*: Erstellen von vrschachtelten XML-Elementen und Angabe eines XML-Standardelementnamensbereichs mithilfe eines Subselects.

```
SELECT XMLELEMENT(
         NAME "root",
         XMLNAMESPACES(DEFAULT 'http://mytest.uri'),
         XMLATTRIBUTES(cid),
            (SELECT
               XMLAGG(
                 XMLELEMENT(
NAME "poid", poid
                   )
                )
              FROM purchaseorder
              WHERE purchaseorder.custid = customer.cid
             )
       )
FROM customer
WHERE cid = '1002'
```

Die Anweisung gibt das folgende XML-Dokument mit dem im Stammelement deklarierten Standardelementnamensbereich zurück:

```
<root xmlns="http://mytest.uri" CID="1002">
  <poid>5000</poid>
  <poid>5003</poid>
  <poid>5006</poid>
</root>
```

- *Beispiel 4*: Verwenden eines allgemeinen Tabellenausdrucks mit XML-Namensbereichen.

  Wenn ein XML-Element mit einem allgemeinen Tabellenausdruck konstruiert wird und das Element an anderer Stelle in derselben SQL-Anweisung verwendet wird, dann sollten alle Namensbereichsdeklarationen als Teil der Elementkonstruktion angegeben werden. Die folgende Anweisung gibt den XML-Standardnamensbereich sowohl im allgemeinen Tabellenausdruck, der die Tabelle PURCHASEORDER zur Erstellung der POID-Elemente verwendet, als auch in der Anweisung SELECT an, die die Tabelle CUSTOMER zur Erstellung des Stammelements verwendet.

```
WITH tempid(id, elem) AS
  (SELECT custid, XMLELEMENT(NAME "poid",
     XMLNAMESPACES(DEFAULT 'http://mytest.uri'),
       poid)
   FROM purchaseorder )
SELECT XMLELEMENT(NAME "root",
         XMLNAMESPACES(DEFAULT 'http://mytest.uri'),
            XMLATTRIBUTES(cid),
            (SELECT XMLAGG(elem)
             FROM tempid
```

```
            WHERE tempid.id = customer.cid )
       )
FROM customer
WHERE cid = '1002'
```

Die Anweisung gibt das folgende XML-Dokument mit einem im Stammelement deklarierten Standardelementnamensbereich zurück.

```
<root xmlns="http://mytest.uri" CID="1002">
  <poid>5000</poid>
  <poid>5003</poid>
  <poid>5006</poid>
</root>
```

In der folgenden Anweisung ist der Standardelementnamensbereich nur in der Anweisung SELECT deklariert, die die Tabelle CUSTOMER zur Erstellung des Stammelements verwendet:

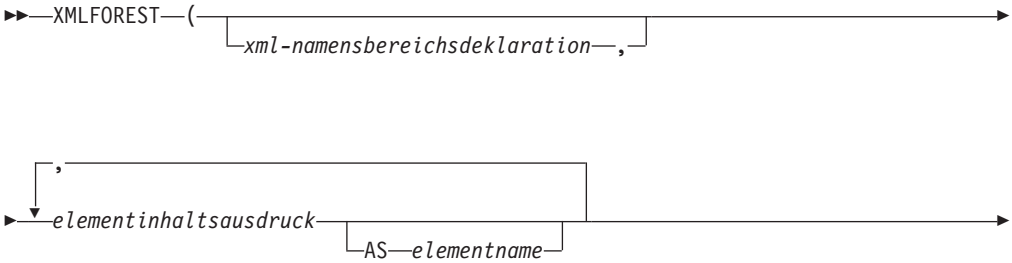
```
WITH tempid(id, elem) AS
    (SELECT custid, XMLELEMENT(NAME "poid", poid)
     FROM purchaseorder )
SELECT XMLELEMENT(NAME "root",
         XMLNAMESPACES(DEFAULT 'http://mytest.uri'),
            XMLATTRIBUTES(cid),
            (SELECT XMLAGG(elem)
             FROM tempid
             WHERE tempid.id = customer.cid )
       )
FROM customer
WHERE cid = '1002'
```

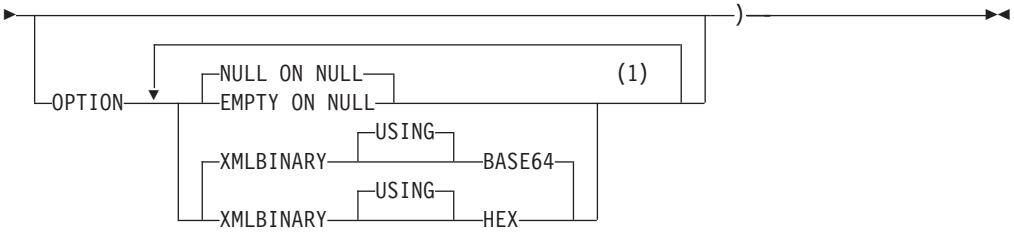
Die Anweisung gibt das folgende XML-Dokument mit dem im Stammelement deklarierten Standardelementnamensbereich zurück. Da die POID-Elemente im allgemeinen Tabellenausdruck ohne eine Deklaration des Standardelementnamensbereichs erstellt werden, ist der Standardelementnamensbereich für die POID-Elemente nicht definiert. Im XML-Dokument wird der Standardelementnamensbereich für die POID-Elemente auf eine leere Zeichenfolge "" gesetzt, weil der Stammelementnamensbereich für die POID-Elemente nicht definiert ist und die POID-Elemente nicht zum Standardelementnamensbereich des Stammelements `xmlns="http://mytest.uri"` gehören.

```
<root xmlns="http://mytest.uri" CID="1002">
  <poid xmlns="">5000</poid>
  <poid xmlns="">5003</poid>
  <poid xmlns="">5006</poid>
</root>
```

XMLFOREST

Die Funktion XMLFOREST gibt einen XML-Wert zurück, der eine Sequenz von XQuery-Elementknoten darstellt.





Anmerkungen:

1 Eine Klausel darf nicht mehrmals angegeben werden.

Das Schema heißt SYSIBM. Der Funktionsname kann nicht als qualifizierter Name angegeben werden.

xml-namensbereichsdeklaration
Gibt die XML-Namensbereichsdeklarationen an, die das Ergebnis der Deklaration XMLNAMESPACES darstellen. Die deklarierten Namensbereichen befinden sich im Geltungsbereich der Funktion XMLFOREST. Die Namensbereiche gelten für alle in der Funktion XMLFOREST verschachtelten XML-Funktionen unabhängig davon, ob sie in einem anderen Subselect angezeigt werden.

Wird *xml-namensbereichsdeklaration* nicht angegeben, werden Namensbereichsdeklarationen den erstellten Elementen nicht zugeordnet.

elementinhaltsausdruck
Der Inhalt des generierten XML-Elementknotens wird mit einem Ausdruck angegeben. Der Datentyp von *elementinhaltsausdruck* darf kein strukturierter Typ sein (SQLSTATE 42884). Der Ausdruck kann ein beliebiger SQL-Ausdruck sein. Wenn der Ausdruck kein einfacher Spaltenverweis ist, muss ein Elementname angegeben werden.

AS *elementname*
Gibt den XML-Elementnamen als SQL-Kennung an. Der Elementname muss das Format eines qualifizierten XML-Namens (QName) haben (SQLSTATE 42634). Weitere detaillierte Informationen zu gültigen Namen finden Sie in W3C-XML-Namensbereichsspezifikationen. Wenn der Name qualifiziert ist, muss das Namensbereichspräfix innerhalb des Geltungsbereichs deklariert werden (SQLSTATE 42635). Wenn *elementname* nicht angegeben wird, muss *elementinhaltsausdruck* ein Spaltenname sein (SQLSTATE 42703). Der Elementname wird über die Zuordnung mit vollständiger Zeichenersetzung aus einem Spaltennamen zu einem qualifizierten Namen erstellt.

OPTION
Gibt Zusatzoptionen zum Erstellen des XML-Elements an. Wenn keine Klausel OPTION angegeben wird, ist NULL ON NULL XMLBINARY USING BASE64 der Standardwert. Diese Klausel hat keinen Einfluss auf verschachtelte Aufrufe von XMLELEMENT, die in *elementinhaltsausdruck* angegeben werden.

EMPTY ON NULL oder NULL ON NULL
Gibt an, ob ein Nullwert oder ein leeres Element zurückgegeben werden soll, wenn jedes Vorkommen von *elementinhaltsausdruck* einen Nullwert hat. Diese Option wirkt sich nur auf die Nullwertbehandlung von Elementinhalt aus, nicht auf Attributwerte. Der Standardwert ist NULL ON NULL.

EMPTY ON NULL
Wenn der Wert jedes Vorkommens von *elementinhaltsausdruck* Null ist, wird ein leeres Element zurückgegeben.

NULL ON NULL
Wenn der Wert jedes Vorkommens von *elementinhaltsausdruck* Null ist, wird ein Nullwert zurückgegeben.

XMLBINARY USING BASE64 oder XMLBINARY USING HEX
Gibt die angenommene Codierung von binären Eingabedaten, Zeichenfolgedaten mit dem Attribut FOR BIT DATA oder von einem einzigartigen Datentyp an, der auf einem dieser Typen basiert. Die Codierung gilt für den Elementinhalt oder für die Attributwerte. Der Standardwert ist XMLBINARY USING BASE64.

XMLBINARY USING BASE64
Gibt an, dass die angenommene Codierung base64-Zeichen sind, wie für die xs:base64Binary-Codierung des XML-Schematyps definiert. Die Codierung base64 verwendet eine Untermenge von US-ASCII mit 65 Zeichen (10 Ziffern, 26 Kleinbuchstaben, 26 Großbuchstaben, '+' und '/') und stellt damit alle 6 Bit der Binär- oder Bitdaten mit einem einzigen druckbaren Zeichen in der Untermenge dar. Diese Zeichen werden ausgewählt und sind dadurch universell darstellbar. Bei Verwendung dieser Methode werden die codierten Daten um 33 % größer als die ursprünglichen Binär- oder Bitdaten.

XMLBINARY USING HEX
Gibt an, dass die angenommene Codierung Hexadezimalzeichen sind, wie für die xs:hexBinary-Codierung des XML-Schematyps definiert. Bei der hexadezimalen Codierung wird jedes Byte (8 Bit) mit zwei Hexadezimalzeichen dargestellt. Bei Verwendung dieser Methode werden die codierten Daten doppelt so groß wie die ursprünglichen Binär- oder Bitdaten.

Diese Funktion nimmt eine optionale Gruppe von Namensbereichsdeklarationen und ein oder mehrere Argumente an, die den Namen und den Elementinhalt für ein oder mehrere Elementknoten bilden. Das Ergebnis ist eine XML-Sequenz, die eine Sequenz von XQuery-Elementknoten oder den Nullwert enthält.

Der Datentyp des Ergebnisses ist XML. Wenn irgendeines der Argument von *elementinhaltsausdruck* Null sein kann, dann kann das Ergebnis Null sein. Wenn alle Argumentwerte von *elementinhaltsausdruck* Null sind und die Option NULL ON NULL wirksam ist, dann ist das Ergebnis der Nullwert.

Die Funktion XMLFOREST lässt sich mit XMLCONCAT und XMLELEMENT ausdrücken. So haben beispielsweise die beiden folgenden Ausdrücke denselben semantischen Wert.

```
XMLFOREST(xml-namensbereichsdeklaration, arg1 AS name1, arg2 AS name2 ...)
```

```
XMLCONCAT(
  XMLELEMENT(
    NAME name1, xml-namensbereichsdeklaration, arg1
  ),
  XMLELEMENT(
    NAME name2, xml-namensbereichsdeklaration, arg2
  )
  ...
)
```

Anmerkungen

- Beim Erstellen von Elementen, die als Inhalt eines anderen Elements kopiert werden, das Namensbereiche definiert, sollte die Deklaration der Standardnamensbereiche explizit aufgehoben werden, um mögliche Fehler zu vermeiden, die bei der Übernahme des Standardnamensbereichs aus dem neuen übergeordneten Element auftreten könnten. Die vordefinierten Namensbereichspräfixe ('xs', 'xsi', 'xml' und 'sqlxml') müssen ebenfalls explizit deklariert werden, wenn sie verwendet werden.

Beispiel

Anmerkung: XMLFOREST fügt keine Leerzeichen oder Zeilenvorschubzeichen in die Ausgabe ein. Die gesamte Beispielausgabe wurde zur besseren Lesbarkeit formatiert.
Erstellen Sie eine Gesamtstruktur von Elementen mit einem Standardnamensbereich.

```
SELECT EMPNO,
  XMLFOREST(
    XMLNAMESPACES(
      DEFAULT 'http://hr.org', 'http://fed.gov' AS "d"
    ),
    LASTNAME, JOB AS "d:job"
  )
AS "Result"
FROM EMPLOYEE
WHERE EDLEVEL = 12
```

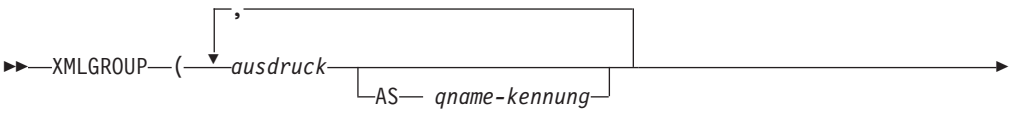
Diese Abfrage führt zu folgendem Ergebnis:

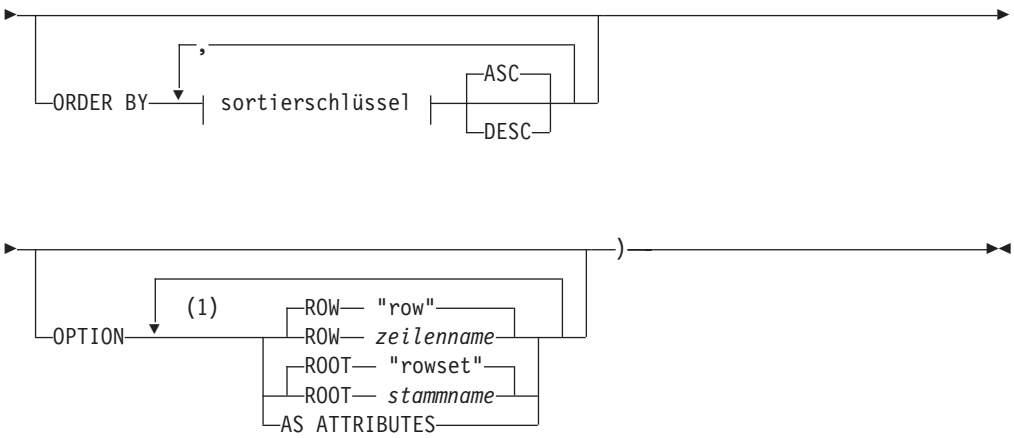
```
EMPNO  Result
000290 <LASTNAME xmlns="http://hr.org" xmlns:d="http://fed.gov">PARKER
  </LASTNAME>
<d:job xmlns="http://hr.org" xmlns:d="http://fed.gov">OPERATOR</d:job>

000310 <LASTNAME xmlns="http://hr.org" xmlns:d="http://fed.gov">SETRIGHT
  </LASTNAME>
<d:job xmlns="http://hr.org" xmlns:d="http://fed.gov">OPERATOR</d:job>
200310 <LASTNAME xmlns="http://hr.org" xmlns:d="http://fed.gov">SPRINGER
  </LASTNAME>
<d:job xmlns="http://hr.org" xmlns:d="http://fed.gov">OPERATOR</d:job>
```

XMLGROUP

Die Funktion XMLGROUP gibt einen XML-Wert mit einem einzigen XQuery-Dokumentknoten zurück, der einen Elementknoten der obersten Ebene enthält. Dies ist ein zusammengefasster Ausdruck, der ein XML-Dokument mit nur einem Stammelement aus einer Gruppe von Zeilen zurückgibt, wobei jede Zeile einem Unterelement "row" zugeordnet wird.





Anmerkungen:

1 Eine Klausel darf nicht mehrmals angegeben werden.

Das Schema heißt SYSIBM. Der Funktionsname kann nicht als qualifizierter Name angegeben werden.

ausdruck
Der Inhalt jedes generierten XML-Elementknotens (oder der Wert jedes generierten Attributs) wird mit einem Ausdruck angegeben. Der Datentyp von *ausdruck* darf kein strukturierter Typ sein (SQLSTATE 42884). Der Ausdruck kann ein beliebiger SQL-Ausdruck sein. Wenn der Ausdruck kein einfacher Spaltenverweis ist, muss eine QName-Kennung (*qname-kennung*) angegeben werden.

AS *qname-kennung*
Gibt den XML-Elementnamen oder den Attributnamen als SQL-Kennung an. *qname-kennung* muss das Format eines qualifizierten XML-Namens (QName) haben (SQLSTATE 42634). Weitere detaillierte Informationen zu gültigen Namen finden Sie in "W3C-XML-Namensbereichsspezifikationen". Wenn der Name qualifiziert ist, muss das Namensbereichspräfix innerhalb des Geltungsbereichs deklariert werden (SQLSTATE 42635). Wenn *qname-kennung* nicht angegeben wird, muss *ausdruck* ein Spaltenname sein (SQLSTATE 42703. Der Element- oder Attributname wird über die Zuordnung mit vollständiger Zeichenersetzung aus einem Spaltennamen zu einem qualifizierten Namen erstellt.

OPTION
Gibt Zusatzoptionen zum Erstellen des XML-Werts an. Wird die Klausel OPTION nicht angegeben, wird das Standardverhalten angewendet.

ROW *zeilenname*
Gibt den Namen des Elements an, dem jede Zeile zugeordnet ist. Wenn diese Option nicht angegeben wird, lautet der Standardelementname "row".

ROOT *stammname*
Gibt den Namen des Stammelementknotens an. Wenn diese Option nicht angegeben wird, lautet der Name des Stammelementknotens "rowset".

AS ATTRIBUTES
Gibt an, dass jeder Ausdruck einem Attributwert mit einem Spaltennamen oder einer *qname-kennung* zugeordnet wird, der bzw. die als Attributname dient.

ORDER BY
Gibt die Reihenfolge der Zeilen aus demselben Gruppierungsset an, die in der Spaltenberechnung verarbeitet werden. Wenn die Klausel ORDER BY weggelassen wird oder die Reihenfolge der Spaltendaten nicht unterscheiden kann, werden die Zeilen in demselben Gruppierungsset willkürlich geordnet.

sortierschlüssel
Der Sortierschlüssel kann ein Spaltenname oder ein *sortierschlüsselausdruck* sein. Beachten Sie Folgendes: Wenn der Sortierschlüssel eine Konstante ist, verweist er nicht auf die Position der Ausgabespalte (wie in der normalen Klausel ORDER BY), sondern stellt einfach eine Konstante dar, was keinen Sortierschlüssel impliziert.

Regeln

- Falls eine SELECT-Klausel eine ARRAY_AGG-Funktion umfasst, dann müssen alle Aufrufe der Funktionen ARRAY_AGG, LISTAGG, XMLAGG und XMLGROUP in derselben SELECT-Klausel dieselbe Reihenfolge oder keine Reihenfolge angeben (SQLSTATE 428GZ).

Anmerkungen

Das Standardverhalten definiert eine einfache Zuordnung zwischen einer Ergebnismenge und einem XML-Wert. Zusätzlich gelten folgende Anmerkungen zum Verhalten von Funktionen:

- Jede Zeile wird standardmäßig in ein XML-Element mit dem Namen "row" umgesetzt, jede Spalte in ein verschachteltes Element, dessen Spaltenname als Elementname dient.
- Das Verhalten für die Nullwertbehandlung ist NULL ON NULL. Ein Nullwert in einer Spalte entspricht dem Nichtvorhandensein des Unterelements. Wenn alle Spaltenwerte null sind, wird kein Zeilenelement generiert.
- Das binäre Codierungsschema für die Datentypen BLOB und FOR BIT DATA ist base64Binary-Codierung.
- Standardmäßig sind die Elemente, die den Zeilen in der Gruppe entsprechen, untergeordnete Elemente eines Stammelements namens "rowset".
- Die Unterelemente "row" im Stammelement befinden sich in derselben Reihenfolge, in der die Zeilen in der Abfrageergebnismenge zurückgegeben werden.
- Zu dem Stammelement wird implizit ein Dokumentknoten hinzugefügt, damit das XML-Ergebnis ein korrekt formatiertes XML-Dokument mit nur einem Stammelement ist.

Beispiele

Die Beispiele hier beruhen auf der folgenden Tabelle T1 mit den ganzzahligen Spalten C1 und C2, die numerische Daten enthalten, die in einem relationalen Format gespeichert sind.

```
C1          C2
----------- -----------
          1           2
          -           2
          1 -          -           -

  4 Satz/Sätze ausgewählt.
```

- *Beispiel 1*: Das folgende Beispiel zeigt eine XMLGroup-Abfrage und ein Ausgabefragment mit Standardverhalten, wobei ein einzelnes Element der höchsten Ebene verwendet wird, um die Tabelle darzustellen.:

```
SELECT XMLGROUP(C1, C2)FROM T1

<rowset>
   <row>
      <C1>1</C1>
      <C2>2</C2>
   </row>
   <row>
      <C2>2</C2>
   </row>
   <row>
      <C1>1</C1>
   </row>
</rowset>

  1 Satz/Sätze ausgewählt.
```

- *Beispiel 2*: Das folgende Beispiel zeigt eine XMLGroup-Abfrage und ein Ausgabefragment mit einer attributzentrierten Zuordnung. Die relationalen Daten werden im Gegensatz zum vorigen Beispiel nicht als verschachtelte Elemente angezeigt, sondern Elementattributen zugeordnet:

```
SELECT XMLGROUP(C1, C2 OPTION AS ATTRIBUTES) FROM T1

<rowset>
   <row C1="1" C2="2"/>
   <row C2="2"/>
   <row C1="1"/>
</rowset>

  1 Satz/Sätze ausgewählt.
```

- *Beispiel 3*: Das folgende Beispiel zeigt eine XMLGroup-Abfrage und ein Ausgabefragment, bei dem das Standardstammelement <rowset> durch <document> und das Standardelement <row> durch <entry> ersetzt wurde. Die Spalten C1 und C2 werden als Elemente <column1> und <column2> zurückgegeben, und die Rückgabemenge wird nach der Spalte C1 sortiert:

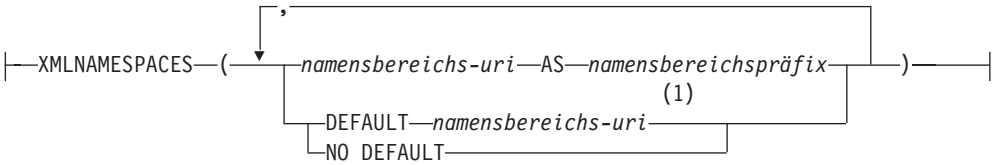
```
SELECT XMLGROUP(
   C1 AS "column1", C2 AS "column2"
   ORDER BY C1 OPTION ROW "entry" ROOT "document")
FROM T1

<document>
   <entry>
      <column1>1</column1>
      <column2>2</column2>
   </entry>
   <entry>
      <column1>1</column1>
   </entry>
   <entry>
      <column2>2</column2>
    </entry>
</document>
```

XMLNAMESPACES

Die Deklaration XMLNAMESPACES erstellt Namensbereichsdeklarationen aus den Argumenten.

xml-namensbereichsdeklaration:



Anmerkungen:

1 DEFAULT oder NO DEFAULT kann in den Argumenten von XMLNAMESPACES nur einmal angegeben werden.

Das Schema heißt SYSIBM. Der Deklarationsname kann nicht als qualifizierter Name angegeben werden.

Diese Deklaration kann nur als Argument für bestimmte Funktionen wie XMLELEMENT, XMLFOREST und XMLTABLE verwendet werden. Das Ergebnis ist mindestens eine XML-Namensbereichsdeklaration mit gültigen Namensbereichen für jeden Eingabewert ungleich null.

namensbereichs-uri
Gibt die einheitliche Referenz-ID (URI) des Namensbereichs als eine SQL-Zeichenfolgenkonstante an. Diese Zeichenfolgenkonstante darf bei der Verwendung mit *namensbereichspräfix* nicht leer sein (SQLSTATE 42815).

namensbereichspräfix
Gibt ein Namensbereichspräfix an. Das Präfix ist eine SQL-Kennung, die das Format eines qualifizierten XML-Namens (QName) haben muss (SQLSTATE 42634). Weitere detaillierte Informationen zu gültigen Namen finden Sie in W3C-XML-Namensbereichsspezifikationen. Das Präfix darf nicht `xml` oder `xmlns` lauten und muss in der Liste der Namensbereichsdeklarationen eindeutig sein (SQLSTATE 42635).

DEFAULT *namensbereichs-uri*
Gibt den Standardnamensbereich an, der im Geltungsbereich dieser Namensbereichsdeklaration verwendet werden soll. Der Wert von *namensbereichs-uri* gilt für nicht qualifizierte Namen im Geltungsbereich, sofern er nicht in einem verschachtelten Geltungsbereich durch eine andere Deklaration DEFAULT oder NO DEFAULT überschrieben wird.

NO DEFAULT
Gibt an, dass kein Standardnamensbereich im Geltungsbereich dieser Namensbereichsdeklaration verwendet werden soll. Es gibt keinen Standardnamensbereich im Geltungsbereich, sofern diese Einstellung nicht in einem verschachtelten Geltungsbereich durch eine Deklaration DEFAULT überschrieben wird.

Der Datentyp des Ergebnisses ist XML. Das Ergebnis ist eine XML-Namensbereichsdeklaration für jeden angegebenen Namensbereich. Das Ergebnis darf nicht Null sein.

Beispiele

Anmerkung: XMLNAMESPACES fügt keine Leerzeichen oder Zeilenvorschubzeichen in die Ausgabe ein. Die gesamte Beispielausgabe wurde zur besseren Lesbarkeit formatiert.

- *Beispiel 1*: Erzeugen Sie ein XML-Element mit dem Namen adm:employee und ein XML-Attribut adm:department, die beide dem Namensbereich zugeordnet sind, dessen Präfix adm lautet.

```
SELECT EMPNO, XMLELEMENT(
  NAME "adm:employee", XMLNAMESPACES(
    'http://www.adm.com' AS "adm"
  ),
  XMLATTRIBUTES(
    WORKDEPT AS "adm:department"
  ),
LASTNAME
)
FROM EMPLOYEE
WHERE JOB = 'ANALYST'
```

Diese Abfrage führt zu folgendem Ergebnis:

```
000130 <adm:employee xmlns:adm="http://www.adm.com" adm:department="C01">
  QUINTANA</adm:employee>
000140 <adm:employee xmlns:adm="http://www.adm.com" adm:department="C01">
  NICHOLLS</adm:employee>
200140 <adm:employee xmlns:adm="http://www.adm.com" adm:department="C01">
  NATZ</adm:employee>
```

- *Beispiel 2*: Erzeugen Sie ein XML-Element mit dem Namen 'employee', das einem Standardnamensbereich zugeordnet ist, und ein Unterelement mit dem Namen 'job', das den Standardnamensbereich nicht verwendet, dessen Unterelement mit dem Namen 'department' jedoch den Standardnamensbereich verwendet.

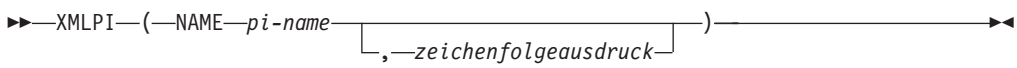
```
SELECT EMP.EMPNO, XMLELEMENT(
  NAME "employee", XMLNAMESPACES(
    DEFAULT 'http://hr.org'
  ),
  EMP.LASTNAME, XMLELEMENT(
    NAME "job", XMLNAMESPACES(
      NO DEFAULT
    ),
    EMP.JOB, XMLELEMENT(
      NAME "department", XMLNAMESPACES(
        DEFAULT 'http://adm.org'
      ),
      EMP.WORKDEPT
    )
  )
)
FROM EMPLOYEE EMP
WHERE EMP.EDLEVEL = 12
```

Diese Abfrage führt zu folgendem Ergebnis:

```
000290 <employee xmlns="http://hr.org">PARKER<job xmlns="">OPERATOR
  <department xmlns="http://adm.org">E11</department></job></employee>
000310 <employee xmlns="http://hr.org">SETRIGHT<job xmlns="">OPERATOR
  <department xmlns="http://adm.org">E11</department></job></employee>
200310 <employee xmlns="http://hr.org">SPRINGER<job xmlns="">OPERATOR
  <department xmlns="http://adm.org">E11</department></job></employee>
```

XMLPI

Die Funktion XMLPI gibt einen XML-Wert mit einem einzigen XQuery-Verarbeitungsanweisungsknoten zurück.



Das Schema heißt SYSIBM. Der Funktionsname kann nicht als qualifizierter Name angegeben werden.

NAME *pi-name*
Gibt den Namen einer Verarbeitungsanweisung an. Der Name ist eine SQL-Kennung, die das Format eines qualifizierten XML-Namens (QName) haben muss (SQLSTATE 42634). Weitere detaillierte Informationen zu gültigen Namen finden Sie in W3C-XML-Namensbereichsspezifikationen. Der Name darf nicht das Wort 'xml' in irgendeiner Kombination von Groß-/Kleinschreibung sein (SQLSTATE 42634).

zeichenfolgeausdruck
Ein Ausdruck, der einen Wert zurückgibt, der eine Zeichenfolge ist. Die resultierende Zeichenfolge wird in UTF-8 konvertiert und muss dem Inhalt einer gemäß den Regeln von XML 1.0 angegebenen XML-Verarbeitungsanweisung entsprechen (SQLSTATE 2200T):

- Die Zeichenfolge darf nicht die Unterzeichenfolge '?>' enthalten, da mit dieser Unterzeichenfolge eine Verarbeitungsanweisung beendet wird.
- Jedes Zeichen der Zeichenfolge kann ein beliebiges Unicode-Zeichen sein, mit Ausnahme der Ersatzzeichenblöcke X'FFFE' und X'FFFF'.

Die resultierende Zeichenfolge wird zum Inhalt des erstellten Verarbeitungsanweisungsknotens.

Der Datentyp des Ergebnisses ist XML. Wenn das Ergebnis von *zeichenfolgeausdruck* Null sein kann, dann kann das Ergebnis Null sein. Wenn *zeichenfolgeausdruck* Null als Ergebnis hat, ist das Ergebnis der Nullwert. Wenn *zeichenfolgeausdruck* eine leere Zeichenfolge ist oder nicht angegeben wird, wird ein leerer Verarbeitungsanweisungsknoten zurückgegeben.

Beispiele

- *Beispiel 1*: Generieren Sie einen XML-Verarbeitungsanweisungsknoten.

```
SELECT XMLPI(
  NAME "Instruction", 'Push the red button'
)
FROM SYSIBM.SYSDUMMY1
```

Diese Abfrage führt zu folgendem Ergebnis:

```
<?Instruction Push the red button?>
```

- *Beispiel 2*: Generieren Sie einen leeren XML-Verarbeitungsanweisungsknoten.

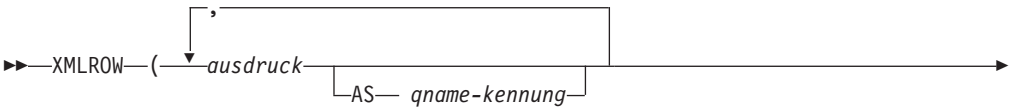
```
SELECT XMLPI(
  NAME "Warning"
)
FROM SYSIBM.SYSDUMMY1
```

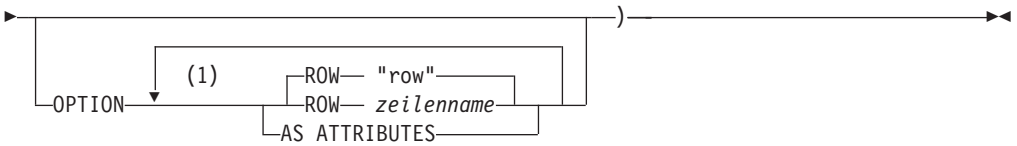
Diese Abfrage führt zu folgendem Ergebnis:

```
<?Warning ?>
```

XMLROW

Die Funktion XMLROW gibt einen XML-Wert mit einem einzigen XQuery-Dokumentknoten zurück, der einen Elementknoten der obersten Ebene enthält.





Anmerkungen:

1 Eine Klausel darf nicht mehrmals angegeben werden.

Das Schema heißt SYSIBM. Der Funktionsname kann nicht als qualifizierter Name angegeben werden.

ausdruck
Der Inhalt jedes generierten XML-Elementknotens wird mit einem Ausdruck angegeben. Der Datentyp des Ausdrucks darf kein strukturierter Typ sein (SQLSTATE 42884). Der Ausdruck kann ein beliebiger SQL-Ausdruck sein. Wenn der Ausdruck kein einfacher Spaltenverweis ist, muss ein Elementname angegeben werden.

AS *qname-kennung*
Gibt den XML-Elementnamen oder den Attributnamen als SQL-Kennung an. *qname-kennung* muss das Format eines qualifizierten XML-Namens (QName) haben (SQLSTATE 42634). Weitere detaillierte Informationen zu gültigen Namen finden Sie in "W3C-XML-Namensbereichsspezifikationen". Wenn der Name qualifiziert ist, muss das Namensbereichspräfix innerhalb des Geltungsbereichs deklariert werden (SQLSTATE 42635). Wenn *qname-kennung* nicht angegeben wird, muss *ausdruck* ein Spaltenname sein (SQLSTATE 42703). Der Element- oder Attributname wird über die Zuordnung mit vollständiger Zeichenersetzung aus einem Spaltennamen zu einem qualifizierten Namen erstellt.

OPTION
Gibt Zusatzoptionen zum Erstellen des XML-Werts an. Wird die Klausel OPTION nicht angegeben, wird das Standardverhalten angewendet.

AS ATTRIBUTES
Gibt an, dass jeder Ausdruck einem Attributwert mit einem Spaltennamen oder einer QName-Kennung (*qname-kennung*) zugeordnet wird, der bzw. die als Attributname dient.

ROW *zeilenname*
Gibt den Namen des Elements an, dem jede Zeile zugeordnet ist. Wenn diese Option nicht angegeben wird, lautet der Standardelementname "row".

Anmerkungen

Standardmäßig wird jede Zeile in der Ergebnismenge einem XML-Wert wie folgt zugeordnet:

- Jede Zeile wird in ein XML-Element mit dem Namen "row" umgesetzt, jede Spalte in ein verschachteltes Element, dessen Name der Spaltenname ist.
- Das Verhalten für die Nullwertbehandlung ist NULL ON NULL. Ein Nullwert in einer Spalte entspricht dem Nichtvorhandensein des Unterelements. Wenn alle Spaltenwerte null sind, gibt die Funktion einen Nullwert zurück.
- Das binäre Codierungsschema für die Datentypen BLOB und FOR BIT DATA ist base64Binary-Codierung.

- Zu dem Zeilenelement wird implizit ein Dokumentknoten hinzugefügt, damit das XML-Ergebnis ein korrekt formatiertes XML-Dokument mit nur einem Stammelement ist.

Beispiele

Angenommen, Sie haben die folgende Tabelle T1 mit den Spalten C1 und C2, in denen numerische Daten in einem relationalen Format gespeichert sind:

```
C1          C2
----------- -----------
          1           2
          -           2
          1 -          -           -

  4 Satz/Sätze ausgewählt.
```

- *Beispiel 1*: Das folgende Beispiel zeigt eine XMLRow-Abfrage und ein Ausgabefragment mit Standardverhalten, wobei eine Sequenz von Zeilenelementen verwendet wird, um die Tabelle darzustellen:

```
SELECT XMLROW(C1, C2) FROM T1
```

```
<row><C1>1</C1><C2>2</C2></row>
<row><C2>2</C2></row>
<row><C1>1</C1></row>

  4 Satz/Sätze ausgewählt.
```

- *Beispiel 2*: Das folgende Beispiel zeigt eine XMLRow-Abfrage und ein Ausgabefragment mit attributzentrierter Zuordnung. Die relationalen Daten werden im Gegensatz zum vorigen Beispiel nicht als verschachtelte Elemente angezeigt, sondern Elementattributen zugeordnet:

```
SELECT XMLROW(C1, C2 OPTION AS ATTRIBUTES) FROM T1
```

```
<row C1="1" C2="2"/>
<row C2="2"/>
<row C1="1"/>

  4 Satz/Sätze ausgewählt.
```

- *Beispiel 3*: Das folgende Beispiel zeigt eine XMLRow-Abfrage und ein Ausgabefragment, bei dem das Standardelement <row> durch <entry> ersetzt wurde. Die Spalten C1 und C2 werden als Elemente <column1> und <column2> zurückgegeben, und die Summe von C1 und C2 wird in einem Element <total> zurückgegeben:

```
SELECT XMLROW(
   C1 AS "column1", C2 AS "column2",
   C1+C2 AS "total" OPTION ROW "entry")
FROM T1
```

```
<entry><column1>1</column1><column2>2</column2><total>3</total></entry>
<entry><column2>2</column2></entry>
<entry><column1>1</column1></entry>

  4 Satz/Sätze ausgewählt.
```

XMLTEXT

Die Funktion XMLTEXT gibt einen XML-Wert mit nur einem XQuery-Textknoten mit dem Eingabeargument als Inhalt zurück.

Das Schema heißt SYSIBM. Der Funktionsname kann nicht als qualifizierter Name angegeben werden.

zeichenfolgeausdruck
Ein Ausdruck, dessen Wert einen Zeichenfolgetyp hat: CHAR, VARCHAR oder CLOB.

Der Datentyp des Ergebnisses ist XML. Wenn das Ergebnis von *zeichenfolgeausdruck* Null sein kann, dann kann das Ergebnis Null sein. Wenn die Eingabe den Wert Null hat, ist das Ergebnis der Nullwert. Wenn das Ergebnis von *zeichenfolgeausdruck* eine leere Zeichenfolge ist, ist der Ergebniswert ein leerer Textknoten.

Beispiele

- *Beispiel 1*: Erstellen Sie eine einfache XMLTEXT-Abfrage.

```
VALUES(
  XMLTEXT(
    'The stock symbol for Johnson&Johnson is JNJ.'
  )
)
```

Diese Abfrage führt zu folgendem serialisierten Ergebnis:

```
1
--------------------------------------------------
The stock symbol for Johnson&amp;Johnson is JNJ.
```

Beachten Sie, dass das Zeichen '&' bei einem serialisierten Textknoten '&' zugeordnet wird.

- *Beispiel 2*: Verwenden Sie XMLTEXT mit XMLAGG, um einen gemischten Inhalt zu erstellen. Angenommen, der Inhalt der Tabelle T lautet wie folgt:

```
seqno  plaintext                                                    emphtext
------ ------------------------------------------------------------
-------------
1      This query shows how to construct                            mixed content
2       using XMLAGG and XMLTEXT. Without                           XMLTEXT
3      XMLAGG will not have text nodes to group with other nodes,   mixed content
       therefore, cannot generate
```

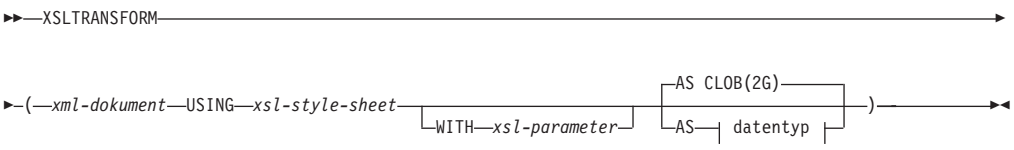
```
SELECT XMLELEMENT(
  NAME "para", XMLAGG(
    XMLCONCAT(
      XMLTEXT(
        PLAINTEXT
      ),
      XMLELEMENT(
        NAME "emphasis", EMPHTEXT
      )
    )
    ORDER BY SEQNO
  ), '.'
) AS "result"
FROM T
```

Diese Abfrage führt zu folgendem Ergebnis:

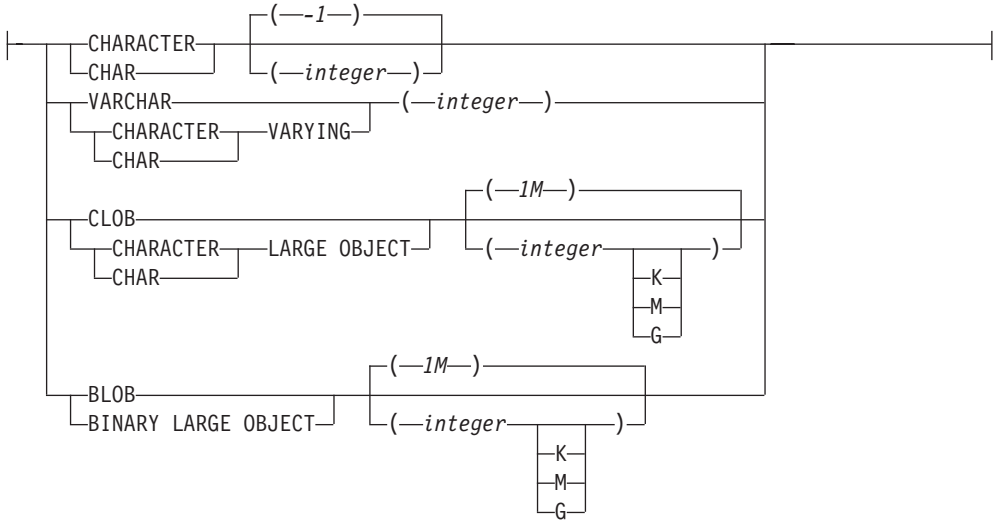
```
result
--------------------------------------------------------------------------------
<para>This query shows how to construct <emphasis>mixed content</emphasis>
using XMLAGG and XMLTEXT. Without <emphasis>XMLTEXT</emphasis> , XMLAGG
will not have text nodes to group with other nodes, therefore, cannot generate
<emphasis>mixed content</emphasis>.</para>
```

XSLTRANSFORM

Verwenden Sie XSLTRANSFORM zum Konvertieren von XML-Daten in andere Formate. Dies umfasst auch die Konvertierung von XML-Dokumenten, die einem bestimmten XML-Schema entsprechen, in Dokumente, die einem anderen XML-Schema entsprechen.



datentyp:



Das Schema heißt SYSIBM. Diese Funktion kann nicht als qualifizierter Name angegeben werden.

Mit der Funktion XSLTRANSFORM kann ein XML-Dokument in ein anderes Datenformat umgesetzt werden. Die Daten können in jedes für den XSLT-Prozessor mögliche Format umgesetzt werden, einschließlich XML, HTML oder Klartext.

XSLTRANSFORM verwendet ausschließlich datenbankinterne Pfade. Dieser Befehl kann derzeit nicht direkt für Dateien oder Style-Sheets verwendet werden, die sich in einem externen Dateisystem befinden.

xml-dokument
Ein Ausdruck, der ein korrekt formatiertes XML-Dokument mit dem Datentyp XML, CHAR, VARCHAR, CLOB oder BLOB zurückgibt. Dieses Dokument wird mit dem in *xsl-style-sheet* angegebenen XSL-Style-Sheet umgesetzt.

Das XML-Dokument darf nur ein einziges Stammelement haben und muss korrekt formatiert sein.

xsl-style-sheet
Ein Ausdruck, der ein korrekt formatiertes XML-Dokument mit dem Datentyp XML, CHAR, VARCHAR, CLOB oder BLOB zurückgibt. Das Dokument ist ein XSL-Style-Sheet, das den Empfehlungen in 'W3C XSLT Version 1.0 Recommendation' entspricht. Style-Sheets mit XQUERY-Anweisungen oder der Deklarati-

on xsl:include werden nicht unterstützt. Dieses Style-Sheet wird angewendet, um den in *xml-dokument* angegebenen Wert umzusetzen.

xsl-parameter
Ein Ausdruck, der ein korrekt formatiertes XML-Dokument oder Null mit dem Datentyp XML, CHAR, VARCHAR, CLOB oder BLOB zurückgibt. Dieses Dokument stellt Parameterwerte für das in *xsl-style-sheet* angegebene XSL-Style-Sheet bereit. Der Wert des Parameters kann als Attribut oder als Textknoten angegeben werden.

Das Parameterdokument hat folgende Syntax:

```
<params xmlns="http://www.ibm.com/XSLTransformParameters">
<param name="..." value="..."/>
<param name="...">enter value here</param>
  ...
</params>
```

Das Style-Sheet-Dokument muss xsl:param-Elemente mit Namensattributwerten enthalten, die mit den im Parameterdokument angegebenen Werten übereinstimmen.

AS *datentyp*
Gibt den Ergebnisdatentyp an. Das implizite oder explizite Längenattribut das angegebenen Ergebnisdatentyps muss ausreichen, um die umgesetzte Ausgabe zu enthalten (SQLSTATE 22001). Der standardmäßige Ergebnisdatentyp ist CLOB(2G).

Wenn das Argument *xml-dokument* oder *xsl-style-sheet* Null ist, dann ist das Ergebnis Null.

Wenn eines der oben genannten Dokumente in einer Spalte des Typs CHAR, VARCHAR oder CLOB gespeichert wird, erfolgt möglicherweise eine Codepagekonvertierung, was zu einem Verlust von Zeichen führen könnte.

Beispiel

Dieses Beispiel zeigt, wie XSLT als Formatierungssteuerkomponente verwendet wird. Zur Vorbereitung müssen Sie zunächst die beiden folgenden Beispieldokumente in die Datenbank einfügen:

```
INSERT INTO XML_TAB VALUES
(1,
          '<?xml version="1.0"?>
<students xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation = "/home/steffen/xsd/xslt.xsd">
<student studentID="1" firstName="Steffen" lastName="Siegmund"
    age=â€23â€ university=â€Rostockâ€/>
</students>',
    '<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
            xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:param name="headline"/>
<xsl:param name="showUniversity"/>
<xsl:template match="students">
                <html>
                   <head/>
                      <body>
                              <h1><xsl:value-of select="$headline"/></h1>
                            <table border="1">
                                    <th>
                                        <tr>
                              <td width="80">StudentID</td>
                              <td width="200">First Name</td>
                              <td width="200">Last Name</td>
```

```
                              <td width="50">Age</td>
                              <xsl:choose>
       <xsl:when test="$showUniversity ='true'">
                                                  <td width="200">University</td>
                                  </xsl:when>
       </xsl:choose>
                           </tr>
                              </th>
                              <xsl:apply-templates/>
                            </table>
                     </body>
              </html>
             </xsl:template>
                    <xsl:template match="student">
                  <tr>
                <td><xsl:value-of select="@studentID"/></td>
                <td><xsl:value-of select="@firstName"/></td>
                <td><xsl:value-of select="@lastName"/></td>
                <td><xsl:value-of select="@age"/></td>
                <xsl:choose>
                    <xsl:when test="$showUniversity = 'true' ">
                      <td><xsl:value-of select="@university"/></td>
                    </xsl:when>
                </xsl:choose>
                       </tr>
                  </xsl:template>
</xsl:stylesheet>'
);
```

Rufen Sie als Nächstes die Funktion XSLTRANSFORM auf, um die XML-Daten in HTML umzusetzen und anzuzeigen:

```
SELECT XSLTRANSFORM (XML_DOC USING XSL_DOC AS CLOB(1M)) FROM XML_TAB;
```

Als Ergebnis wird folgendes Dokument angezeigt:

```
<html>
<head>
<META http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<h1></h1>
<table border="1">
<th>
<tr>
<td width="80">StudentID</td>
<td width="200">First Name</td>
<td width="200">Last Name</td>
<td width="50">Age</td>
</tr>
</th>
 <tr>
<td>1</td>
<td>Steffen</td><td>Siegmund</td>
<td>23</td>
</tr>
             </table>
</body>
</html>
```

In diesem Beispiel liegt die Ausgabe in HTML vor, und die Parameter beeinflussen lediglich, welche HTML erstellt wird und welche Daten umgewandelt werden. Somit veranschaulicht das Beispiel die Verwendung von XSLT als Formatierungssteuerkomponente für die Endbenutzerausgabe.

Hinweis

Für die Umsetzung von XML-Dokumenten stehen mehrere Methoden zur Verfügung, z. B. die Funktion XSLTRANSFORM, ein XQuery-Aktualisierungsausdruck sowie die XSLT-Verarbeitung durch einen externen Anwendungsserver. Für Dokumente, die in einer DB2-XML-Spalte gespeichert sind, können viele Umsetzungen effizienter mithilfe eines XQuery-Aktualisierungsausdrucks durchgeführt werden als mit XSLT, da bei XSLT stets eine Syntaxanalyse der umzusetzenden XML-Dokumente erforderlich ist. Wenn Sie für die Umsetzung von XML-Dokumenten XSLT verwenden möchten, sollten Sie sorgfältig abwägen, ob das Dokument in der Datenbank oder in einem Anwendungsserver umgesetzt werden soll.

Anhang C. Gespeicherte Prozeduren und Befehle für XSR

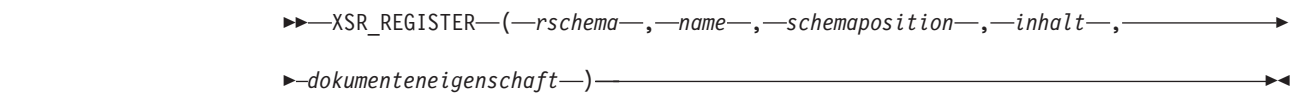
In den folgenden Abschnitten wird die Syntax der gespeicherten Prozeduren und Befehle für XSR von DB2 beschrieben.

Informationen zur Verwendung der gespeicherten Prozeduren und Befehle finden Sie unter „XSR-Objekte" auf Seite 225.

Gespeicherte XSR-Prozeduren

XSR_REGISTER

Die Prozedur XSR_REGISTER ist die erste Prozedur, die im Rahmen der XML-Schemaregistrierung aufgerufen werden muss, um XML-Schemata beim XML-Schema-Repository (XSR) zu registrieren.



Das Schema heißt SYSPROC.

Berechtigung

Die Berechtigungs-ID des Programms, das die Prozedur aufruft, muss mindestens über eines der folgenden Zugriffsrechte (bzw. Berechtigungen) verfügen:

- Berechtigung DBADM
- Datenbankberechtigung IMPLICIT_SCHEMA, wenn das SQL-Schema nicht vorhanden ist
- Zugriffsrecht CREATEIN, wenn das SQL-Schema vorhanden ist

rschema
: Ein Eingabe- und Ausgabeargument des Typs VARCHAR (128), das das SQL-Schema des XML-Schemas angibt. Das SQL-Schema ist ein Teil der SQL-Kennung, die dieses XML-Schema im XML-Schema-Repository identifiziert. (Der andere Teil der SQL-Kennung wird vom Argument 'name' bereitgestellt.) Dieses Argument kann einen Nullwert annehmen. Dies zeigt an, dass das SQL-Standardschema verwendet wird, das im Sonderregister CURRENT SCHEMA definiert ist. Die Regeln für gültige Zeichen und Begrenzer, die für alle SQL-Kennungen gelten, haben auch für dieses Argument Gültigkeit. Relationale Schemata, die mit der Zeichenfolge 'SYS' anfangen, dürfen nicht für diesen Wert verwendet werden. XSR-Objekte haben keine Namenskonflikte mit Datenbankobjekten außerhalb des XSR, da XSR-Objekte in einem anderen Namensbereich vorkommen als Objekte, die sich außerhalb des XML-Schema-Repositorys befinden.

name
: Ein Eingabe- und Ausgabeargument des Typs VARCHAR (128), das den Namen des XML-Schemas angibt. Die vollständige SQL-Kennung für das XML-Schema lautet *rschema.name* und muss unter allen Objekten im XML-Schema-Repository eindeutig sein. Dieses Argument akzeptiert einen Nullwert. Wenn dieses Argument einen Nullwert annimmt, wird ein eindeutiger Wert generiert

und im XML-Schema-Repository gespeichert. Die Regeln für gültige Zeichen und Begrenzer, die für alle SQL-Kennungen gelten, haben auch für dieses Argument Gültigkeit.

schemaposition
Ein Eingabeargument des Typs VARCHAR (1000), das einen Nullwert annehmen kann, zur Angabe der Schemaposition des primären XML-Schemadokuments. Dieses Argument ist der externe Name des XML-Schemas, d. h., das primäre Dokument kann unter den XML-Instanzdokumenten mit dem Attribut xsi:schemaLocation angegeben werden.

inhalt
Ein Eingabeparameter des Typs BLOB (30M), der den Inhalt des primären XML-Schemadokuments enthält. Dieses Argument kann keinen Nullwert annehmen. Es muss ein XML-Schemadokument bereitgestellt werden.

dokumenteneigenschaft
Ein Eingabeparameter des Typs BLOB (5M), der die Eigenschaften des primären XML-Schemadokuments angibt. Dieser Parameter kann einen Nullwert annehmen. Andernfalls ist der Wert ein XML-Dokument.

Beispiele

- *Beispiel 1*: Das folgende Beispiel zeigt, wie die Prozedur XSR_REGISTER über die Befehlszeile aufgerufen wird:

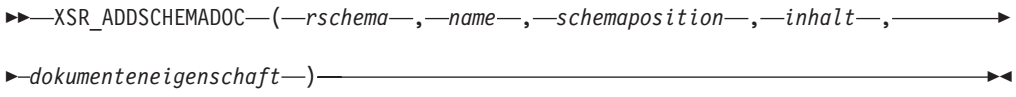
```
CALL SYSPROC.XSR_REGISTER(
  'user1',
  'POschema',
  'http://myPOschema/PO.xsd',
  :content_host_var,
  :docproperty_host_var)
```

- *Beispiel 2*: Das folgende Beispiel zeigt, wie die Prozedur XSR_REGISTER über ein Java-Anwendungsprogramm aufgerufen wird:

```
stmt = con.prepareCall("CALL SYSPROC.XSR_REGISTER (?, ?, ?, ?, ?)");
String xsrObjectName = "myschema1";
String xmlSchemaLocation = "po.xsd";
stmt.setNull(1, java.sql.Types.VARCHAR);
stmt.setString(2, xsrObjectName);
stmt.setString(3, xmlSchemaLocation);
stmt.setBinaryStream(4, buffer, (int)length);
stmt.setNull(5, java.sql.Types.BLOB);
stmt.registerOutParameter(1, java.sql.Types.VARCHAR);
stmt.registerOutParameter(2, java.sql.Types.VARCHAR);
stmt.execute();
```

XSR_ADDSCHEMADOC

Jedes XML-Schema im XML-Schema-Repository (XSR) kann aus einem oder aus mehreren XML-Schemadokumenten bestehen. Wenn ein XML-Schema aus mehreren Dokumenten besteht, wird die Prozedur XSR_ADDSCHEMADOC verwendet, um alle XML-Schemas außer dem primären XML-Schemadokument hinzuzufügen.



Das Schema heißt SYSPROC.

Berechtigung

Die Berechtigungs-ID des Programms, das die Prozedur aufruft, muss der Eigner des XSR-Objekts sein, der in der Katalogsicht SYSCAT.XSROBJECTS gespeichert ist.

rschema
Ein Eingabeargument des Typs VARCHAR (128), das das SQL-Schema des XML-Schemas angibt. Das SQL-Schema ist ein Teil der SQL-Kennung, die dieses XML-Schema im XML-Schema-Repository identifiziert, das in den Status 'Abgeschlossen' versetzt werden muss. (Der andere Teil der SQL-Kennung wird vom Argument 'name' bereitgestellt.) Dieses Argument kann einen Nullwert annehmen. Dies zeigt an, dass das SQL-Standardschema verwendet wird, das im Sonderregister CURRENT SCHEMA definiert ist. Die Regeln für gültige Zeichen und Begrenzer, die für alle SQL-Kennungen gelten, haben auch für dieses Argument Gültigkeit. XSR-Objekte haben keine Namenskonflikte mit Datenbankobjekten außerhalb des XSR, da XSR-Objekte in einem anderen Namensbereich vorkommen als Objekte, die sich außerhalb des XML-Schema-Repositorys befinden.

name
Ein Eingabeargument des Typs VARCHAR (128), das den Namen des XML-Schemas angibt. Die vollständige SQL-Kennung für das XML-Schema lautet *rschema.name*. Der XML-Schemaname muss durch einen Aufruf der Prozedur XSR_REGISTER bereits vorhanden sein, und die Registrierung des XML-Schemas darf noch nicht abgeschlossen sein. Dieses Argument darf keinen Nullwert annehmen. Die Regeln für gültige Zeichen und Begrenzer, die für alle SQL-Kennungen gelten, haben auch für dieses Argument Gültigkeit.

schemaposition
Ein Eingabeargument des Typs VARCHAR (1000), des einen Nullwert annehmen kann, zur Angabe der Schemaposition des primären XML-Schemadokuments, zu dem das XML-Schemadokument hinzugefügt wird. Dieses Argument ist der externe Name des XML-Schemas, d. h., das primäre Dokument kann unter den XML-Instanzdokumenten mit dem Attribut xsi:schemaLocation angegeben werden.

inhalt
Ein Eingabeparameter des Typs BLOB (30M), der den Inhalt des XML-Schemadokuments enthält, das hinzugefügt wird. Dieses Argument kann keinen Nullwert annehmen. Es muss ein XML-Schemadokument bereitgestellt werden.

dokumenteneigenschaft
Ein Eingabeparameter des Typs BLOB (5M), der die Eigenschaften des XML-Schemadokuments angibt, das hinzugefügt wird. Dieser Parameter kann einen Nullwert annehmen. Andernfalls ist der Wert ein XML-Dokument.

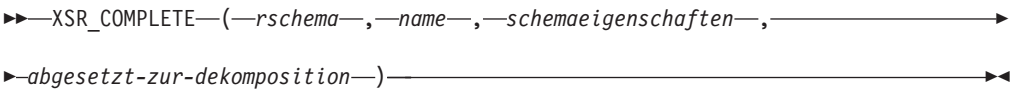
Beispiel

```
CALL SYSPROC.XSR_ADDSCHEMADOC(
  'user1',
  'POschema',
  'http://myPOschema/address.xsd',
  :content_host_var,
  0)
```

XSR_COMPLETE

Die Prozedur XSR_COMPLETE ist die letzte Prozedur, die im Rahmen der XML-Schemaregistrierung aufgerufen werden muss, um XML-Schemata beim XML-Sche-

ma-Repository (XSR) zu registrieren. Ein XML-Schema kann erst geprüft werden, wenn die Schemaregistrierung durch einen Aufruf dieser Prozedur abgeschlossen wird.



Das Schema heißt SYSPROC.

Berechtigung:

Die Berechtigungs-ID des Programms, das die Prozedur aufruft, muss der Eigner des XSR-Objekts sein, der in der Katalogsicht SYSCAT.XSROBJECTS gespeichert ist.

rschema
Ein Eingabeargument des Typs VARCHAR (128), das das SQL-Schema des XML-Schemas angibt. Das SQL-Schema ist ein Teil der SQL-Kennung, die dieses XML-Schema im XML-Schema-Repository identifiziert, das in den Status 'Abgeschlossen' versetzt werden muss. (Der andere Teil der SQL-Kennung wird vom Argument 'name' bereitgestellt.) Dieses Argument kann einen Nullwert annehmen. Dies zeigt an, dass das SQL-Standardschema verwendet wird, das im Sonderregister CURRENT SCHEMA definiert ist. Die Regeln für gültige Zeichen und Begrenzer, die für alle SQL-Kennungen gelten, haben auch für dieses Argument Gültigkeit. XSR-Objekte haben keine Namenskonflikte mit Datenbankobjekten außerhalb des XSR, da XSR-Objekte in einem anderen Namensbereich vorkommen als Objekte, die sich außerhalb des XML-Schema-Repositorys befinden.

name
Ein Eingabeargument des Typs VARCHAR (128), das den Namen des XML-Schemas angibt. Die vollständige SQL-Kennung für das XML-Schema, dessen Beendigungsstatus geprüft werden muss, lautet *rschema.name*. Der XML-Schemaname muss durch einen Aufruf der Prozedur XSR_REGISTER bereits vorhanden sein, und die Registrierung des XML-Schemas darf noch nicht abgeschlossen sein. Dieses Argument darf keinen Nullwert annehmen. Die Regeln für gültige Zeichen und Begrenzer, die für alle SQL-Kennungen gelten, haben auch für dieses Argument Gültigkeit.

schemaeigenschaften
Ein Eingabeargument des Typs BLOB (5M), das die dem XML-Schema zugeordneten Eigenschaften (sofern vorhanden) angibt. Der Wert für dieses Argument ist entweder der Nullwert, wenn keine Eigenschaften zugeordnet sind, oder ein XML-Dokument, das die Eigenschaften des XML-Schemas darstellt.

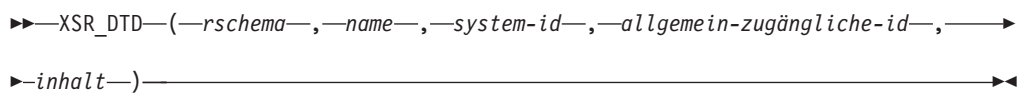
abgesetzt-zur-dekomposition
Ein Eingabeparameter des Typs INTEGER, der angibt, ob ein XML-Schema zur Dekomposition verwendet wird. Wenn ein XML-Schema zur Dekomposition verwendet wird, muss dieser Wert auf 1 eingestellt werden, andernfalls auf 0.

Beispiel

```
CALL SYSPROC.XSR_COMPLETE(
  'user1',
  'POschema',
  :schemaproperty_host_var,
  0)
```

XSR_DTD

Die Prozedur XSR_DTD registriert eine Dokumenttypdeklaration (DTD) beim XML-Schema-Repository (XSR).



Das Schema heißt SYSPROC.

Berechtigung

Die Berechtigungs-ID des Programms, das die Prozedur aufruft, muss mindestens über eines der folgenden Zugriffsrechte (bzw. Berechtigungen) verfügen:

- Berechtigung DBADM
- Datenbankberechtigung IMPLICIT_SCHEMA, wenn das SQL-Schema nicht vorhanden ist
- Zugriffsrecht CREATEIN, wenn das SQL-Schema vorhanden ist

rschema
: Ein Eingabe- und Ausgabeargument des Typs VARCHAR (128), das das SQL-Schema der DTD angibt. Das SQL-Schema ist ein Teil der SQL-Kennung, die diese DTD im XML-Schema-Repository identifiziert. (Der andere Teil der SQL-Kennung wird vom Argument *name* bereitgestellt.) Dieses Argument kann einen Nullwert annehmen. Dies zeigt an, dass das SQL-Standardschema verwendet wird, das im Sonderregister CURRENT SCHEMA definiert ist. Die Regeln für gültige Zeichen und Begrenzer, die für alle SQL-Kennungen gelten, haben auch für dieses Argument Gültigkeit. Relationale Schemata, die mit der Zeichenfolge 'SYS' anfangen, dürfen nicht für diesen Wert verwendet werden. XSR-Objekte haben keine Namenskonflikte mit Datenbankobjekten außerhalb des XSR, da XSR-Objekte in einem anderen Namensbereich vorkommen als Objekte, die sich außerhalb des XML-Schema-Repositorys befinden.

name
: Ein Eingabe- und Ausgabeargument des Typs VARCHAR (128), das den Namen der DTD angibt. Die vollständige SQL-Kennung für die DTD lautet *rschema.name* und muss unter allen Objekten im XML-Schema-Repository eindeutig sein. Dieses Argument akzeptiert einen Nullwert. Wenn dieses Argument einen Nullwert annimmt, wird ein eindeutiger Wert generiert und im XML-Schema-Repository gespeichert. Die Regeln für gültige Zeichen und Begrenzer, die für alle SQL-Kennungen gelten, haben auch für dieses Argument Gültigkeit.

system-id
: Ein Eingabeparameter des Typs VARCHAR (1000), der die System-ID der DTD angibt. Die System-ID der DTD muss mit der URI der DTD in der DOCTYPE-Deklaration des XML-Instanzdokuments oder in einer ENTITY-Deklaration (mit dem Schlüsselwort SYSTEM als Präfix, falls verwendet) übereinstimmen. Dieses Argument darf keinen Nullwert annehmen. Die System-ID kann zusammen mit einer allgemein zugänglichen ID angegeben werden.

allgemein-zugängliche-id
: Ein Eingabeparameter des Typs VARCHAR (1000), der die allgemein zugängliche ID der DTD angibt. Die allgemein zugängliche ID einer DTD muss mit der URI der DTD in der DOCTYPE-Deklaration des XML-Instanzdokuments oder in einer ENTITY-Deklaration (mit dem Schlüsselwort PUBLIC als Präfix, falls verwendet) übereinstimmen. Dieses Argument akzeptiert einen Nullwert und

darf nur verwendet werden, wenn es auch in der DOCTYPE-Deklaration des XML-Instanzdokuments oder in einer ENTITY-Deklaration angegeben wird.

inhalt
Ein Eingabeparameter des Typs BLOB (30M), der den Inhalt des DTD-Dokuments enthält. Dieses Argument darf keinen Nullwert annehmen.

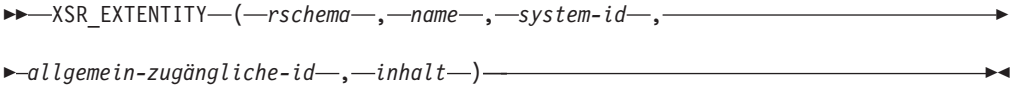
Beispiel

Registrierung der mit der System-ID *http://www.test.com/person.dtd* und der allgemein zugänglichen ID *http://www.test.com/person* angegebenen DTD:

```
CALL SYSPROC.XSR_DTD ( 'MYDEPT' ,
   'PERSONDTD' ,
   'http://www.test.com/person.dtd' ,
   'http://www.test.com/person',
   :content_host_variable
   )
```

XSR_EXTENTITY

Die Prozedur XSR_EXTENTITY registriert eine externe Entität beim XML-Schema-Repository (XSR).



Das Schema heißt SYSPROC.

Berechtigung

Die Berechtigungs-ID des Programms, das die Prozedur aufruft, muss mindestens über eines der folgenden Zugriffsrechte (bzw. Berechtigungen) verfügen:

- Berechtigung DBADM
- Datenbankberechtigung IMPLICIT_SCHEMA, wenn das SQL-Schema nicht vorhanden ist
- Zugriffsrecht CREATEIN, wenn das SQL-Schema vorhanden ist

rschema
Ein Eingabe- und Ausgabeargument des Typs VARCHAR (128), das das SQL-Schema der externen Entität angibt. Das SQL-Schema ist ein Teil der SQL-Kennung, die diese externe Entität im XML-Schema-Repository identifiziert. (Der andere Teil der SQL-Kennung wird vom Argument *name* bereitgestellt.) Dieses Argument kann einen Nullwert annehmen. Dies zeigt an, dass das SQL-Standardschema verwendet wird, das im Sonderregister CURRENT SCHEMA definiert ist. Die Regeln für gültige Zeichen und Begrenzer, die für alle SQL-Kennungen gelten, haben auch für dieses Argument Gültigkeit. Relationale Schemata, die mit der Zeichenfolge 'SYS' anfangen, dürfen nicht für diesen Wert verwendet werden. XSR-Objekte haben keine Namenskonflikte mit Datenbankobjekten außerhalb des XSR, da XSR-Objekte in einem anderen Namensbereich vorkommen als Objekte, die sich außerhalb des XML-Schema-Repositorys befinden.

name
Ein Eingabe- und Ausgabeargument des Typs VARCHAR (128), das den Namen der externen Entität angibt. Die vollständige SQL-Kennung für die externe

Entität lautet *rschema.name* und muss unter allen Objekten im XML-Schema-Repository eindeutig sein. Dieses Argument akzeptiert einen Nullwert. Wenn dieses Argument einen Nullwert annimmt, wird ein eindeutiger Wert generiert und im XML-Schema-Repository gespeichert. Die Regeln für gültige Zeichen und Begrenzer, die für alle SQL-Kennungen gelten, haben auch für dieses Argument Gültigkeit.

system-id
Ein Eingabeparameter des Typs VARCHAR (1000), der die System-ID der externen Entität angibt. Die System-ID der externen Entität muss mit der URI der externen Entität in der ENTITY-Deklaration (mit dem Schlüsselwort SYSTEM als Präfix, falls verwendet) übereinstimmen. Dieses Argument darf keinen Nullwert annehmen. Die System-ID kann zusammen mit einer allgemein zugänglichen ID angegeben werden.

allgemein-zugängliche-id
Ein Eingabeparameter des Typs VARCHAR (1000), der die allgemein zugängliche ID der externen Entität angibt. Die allgemein zugängliche ID einer externen Entität muss mit der URI der externen Entität in der ENTITY-Deklaration (mit dem Schlüsselwort PUBLIC als Präfix, falls verwendet) übereinstimmen. Dieses Argument akzeptiert einen Nullwert und darf nur verwendet werden, wenn es auch in der DOCTYPE-Deklaration des XML-Instanzdokuments oder in einer ENTITY-Deklaration angegeben wird.

inhalt
Ein Eingabeparameter des Typs BLOB (30M), der den Inhalt des Dokuments der externen Entität enthält. Dieses Argument darf keinen Nullwert annehmen.

Beispiel

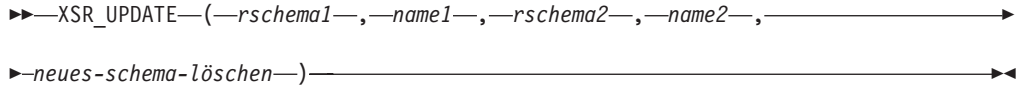
Registrierung der mit den System-IDs *http://www.test.com/food/chocolate.txt* und *http://www.test.com/food/cookie.txt* angegebenen externen Entitäten:

```
CALL SYSPROC.XSR_EXTENTITY ( 'FOOD' ,
   'CHOCLATE' ,
   'http://www.test.com/food/chocolate.txt' ,
   NULL ,
   :content_of_chocolate.txt_as_a_host_variable
   )

CALL SYSPROC.XSR_EXTENTITY ( 'FOOD' ,
   'COOKIE' ,
   'http://www.test.com/food/cookie.txt' ,
   NULL ,
   :content_of_cookie.txt_as_a_host_variable
   )
```

XSR_UPDATE

Mit der Prozedur XSR_UPDATE wird ein vorhandenes XML-Schema im XML-Schema-Repository (XSR) weiterentwickelt. Dadurch können Sie ein vorhandenes XML-Schema so ändern oder erweitern, dass Sie es zur Prüfung vorhandener und neu eingefügter XML-Dokumente verwenden können.



Das Schema heißt SYSPROC.

Das ursprüngliche und das neue XML-Schema, die beide als Argumente von XSR_UPDATE angegeben werden, müssen vor dem Aufruf der Prozedur registriert und abgeschlossen werden. Diese XML-Schemata müssen außerdem kompatibel sein. Der Abschnitt *Kompatibilitätsanforderungen für das Weiterentwickeln von XML-Schemata* enthält ausführliche Informationen zu den Kompatibilitätsanforderungen.

Berechtigung

Die Berechtigungs-ID des Programms, das die Prozedur aufruft, muss zumindest über eines der folgenden Zugriffsrechte verfügen:

- Berechtigung DBADM
- Zugriffsrecht SELECT für die Katalogsichten SYSCAT.XSROBJECTS und SYSCAT.XSROBJECTCOMPONENTS und eine der folgenden Gruppen von Zugriffsrechten:
  - OWNER für das mit dem SQL-Schema *rschema1* und mit dem Objektnamen *name1* angegebene XML-Schema
  - Zugriffsrecht ALTERIN für das XML-Schema, das mit dem Argument *rschema1* angegeben wird, und - falls das Argument *neues-schema-löschen* ungleich Null ist - Zugriffsrecht DROPIN für das mit dem Argument *rschema2* angegebene SQL-Schema

rschema1
: Ein Eingabeargument des Typs VARCHAR (128), das das SQL-Schema des ursprünglichen XML-Schemas angibt, das aktualisiert werden soll. Das SQL-Schema ist ein Teil der SQL-Kennung, die dieses XML-Schema im XML-Schema-Repository identifiziert. (Der andere Teil der SQL-Kennung wird vom Argument *name1* bereitgestellt.) Dieses Argument darf keinen Nullwert annehmen. Die Regeln für gültige Zeichen und Begrenzer, die für alle SQL-Kennungen gelten, haben auch für dieses Argument Gültigkeit.

name1
: Ein Eingabeargument des Typs VARCHAR (128), das den Namen des ursprünglichen XML-Schemas angibt, das aktualisiert werden soll. Die vollständige SQL-Kennung für das XML-Schema lautet *rschema1.name1*. Dieses XML-Schema muss bereits im XML-Schema-Repository registriert und abgeschlossen sein. Dieses Argument darf keinen Nullwert annehmen. Die Regeln für gültige Zeichen und Begrenzer, die für alle SQL-Kennungen gelten, haben auch für dieses Argument Gültigkeit.

rschema2
: Ein Eingabeargument des Typs VARCHAR (128), das das SQL-Schema des neuen XML-Schemas angibt, mit dem das ursprüngliche XML-Schema aktualisiert werden soll. Das SQL-Schema ist ein Teil der SQL-Kennung, die dieses XML-Schema im XML-Schema-Repository identifiziert. (Der andere Teil der SQL-Kennung wird vom Argument *name2* bereitgestellt.) Dieses Argument darf keinen Nullwert annehmen. Die Regeln für gültige Zeichen und Begrenzer, die für alle SQL-Kennungen gelten, haben auch für dieses Argument Gültigkeit.

name2
: Ein Eingabeargument des Typs VARCHAR (128), das den Namen des neuen XML-Schemas angibt, mit dem das ursprüngliche XML-Schema aktualisiert werden soll. Die vollständige SQL-Kennung für das XML-Schema lautet *rschema2.name2*. Dieses XML-Schema muss bereits im XML-Schema-Repository registriert und abgeschlossen sein. Dieses Argument darf keinen Nullwert annehmen. Die Regeln für gültige Zeichen und Begrenzer, die für alle SQL-Kennungen gelten, haben auch für dieses Argument Gültigkeit.

neues-schema-löschen
Ein Eingabeparameter des Typs INTEGER, der angibt, ob das neue XML-Schema nach der Aktualisierung des ursprünglichen XML-Schemas gelöscht werden soll. Wenn für diesen Parameter ein Wert ungleich null eingestellt wird, wird das neue XML-Schema gelöscht. Dieses Argument darf keinen Nullwert annehmen.

Beispiel

```
CALL SYSPROC.XSR_UPDATE(
  'STORE',
  'PROD',
  'STORE',
  'NEWPROD',
  1)
```

Der Inhalt des XML-Schemas STORE.PROD wird durch den Inhalt von STORE.NEWPROD aktualisiert, und das XML-Schema STORE.NEWPROD wird gelöscht.

XSR-Befehle

REGISTER XMLSCHEMA

Registriert ein XML-Schema beim XML-Schema-Repository (XSR).

Berechtigung

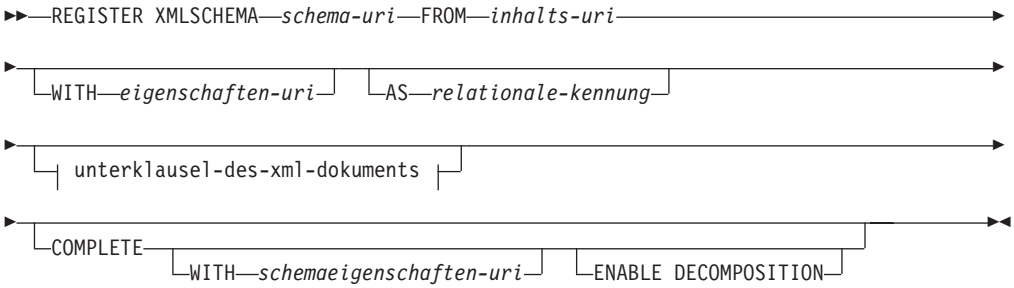
Eine der folgenden Berechtigungen:

- DBADM
- Datenbankberechtigung IMPLICIT_SCHEMA, wenn das SQL-Schema nicht vorhanden ist
- Zugriffsrecht CREATEIN, wenn das SQL-Schema vorhanden ist

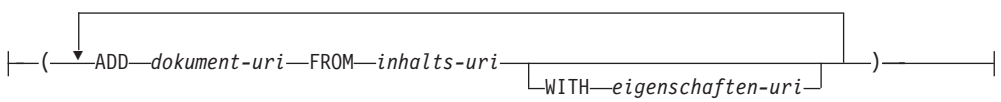
Erforderliche Verbindung

Datenbank

Befehlssyntax



unterklausel-des-xml-dokuments:



Befehlsparameter

schema-uri
Gibt die von XML-Instanzdokumenten referenzierte URI des XML-Schemas an, das gerade registriert wird.

FROM *inhalts-uri*
Gibt die URI an, bei der sich das XML-Schemadokument befindet. Es wird nur eine lokale Datei unterstützt, die von einer Dateischema-URI angegeben wird.

WITH *eigenschaften-URI*
Gibt die URI eines Eigenschaftendokuments für das XML-Schema an. Es wird nur eine lokale Datei unterstützt, die von einer Dateischema-URI angegeben wird.

AS *relationale-kennung*
Gibt einen Namen an, der zum Verweis auf das XML-Schema verwendet werden kann, das gerade registriert wird. Der relationale Name kann als zweiteilige SQL-Kennung angegeben werden, die aus dem SQL-Schema und dem XML-Schemanamen besteht und das folgende Format hat: SQL-schema.name. Das standardmäßige relationale Schema, das im Sonderregister CURRENT SCHEMA definiert ist, wird verwendet, wenn kein Schema angegeben wird. Wird kein Name bereitgestellt, wird ein eindeutiger Wert generiert.

COMPLETE
Gibt an, dass keine weiteren XML-Schemadokumente vorhanden sind, die hinzugefügt werden sollen. Bei dieser Angabe wird das Schema geprüft und als verwendbar markiert, wenn keine Fehler gefunden werden.

WITH *schemaeigenschaften-uri*
Gibt die URI eines Eigenschaftendokuments für das XML-Schema an. Es wird nur eine lokale Datei unterstützt, die von einer Dateischema-URI angegeben wird.

ENABLE DECOMPOSITION
Gibt an, dass dieses Schema zur Dekomposition von XML-Dokumenten verwendet werden soll.

ADD *dokument-uri*
Gibt die URI eines XML-Schemadokuments an, das zu diesem Schema hinzugefügt werden soll, als würde ein anderes XML-Dokument auf dieses Dokument verweisen.

FROM *inhalts-uri*
Gibt die URI an, bei der sich das XML-Schemadokument befindet. Es wird nur eine lokale Datei unterstützt, die von einer Dateischema-URI angegeben wird.

WITH *eigenschaften-URI*
Gibt die URI eines Eigenschaftendokuments für das XML-Schema an. Es wird nur eine lokale Datei unterstützt, die von einer Dateischema-URI angegeben wird.

Beispiele

```
REGISTER XMLSCHEMA 'http://myPOschema/PO.xsd'
FROM 'file:///c:/TEMP/PO.xsd'
WITH 'file:///c:/TEMP/schemaProp.xml'
AS user1.POschema
```

Hinweise

- Damit auf ein XML-Schemadokument verwiesen werden kann und es zur Prüfung und Annotation verfügbar ist, muss es zunächst beim XML-Schema-Repository registriert werden. Dieser Befehl führt den ersten Schritt im Registrierungsprozess des XML-Schemas aus, indem er das primäre XML-Schemadokument registriert. Beim letzten Schritt der XML-Schemaregistrierung muss der Befehl **COMPLETE XMLSCHEMA** erfolgreich für das XML-Schema ausgeführt werden. Als Alternative können Sie, falls keine anderen XML-Schemadokumente einzubinden sind, den Befehl **REGISTER XMLSCHEMA** mit dem Schlüsselwort **COMPLETE** ausführen, um die Registrierung mit einem einzigen Schritt durchzuführen.
- Bei der Registrierung eines XML-Schemas in der Datenbank ist je nach der Größe des XML-Schemas möglicherweise ein größerer Anwendungsheapspeicher (**applheapsz**) erforderlich. Die empfohlene Größe beträgt 1024, aber bei größeren Schemata ist ein zusätzlicher Speicher erforderlich.

ADD XMLSCHEMA DOCUMENT

Fügt vor dem Abschluss der Registrierung ein oder mehrere XML-Schemadokumente zu einem vorhandenen, aber unvollständigen XML-Schema hinzu.

Berechtigung

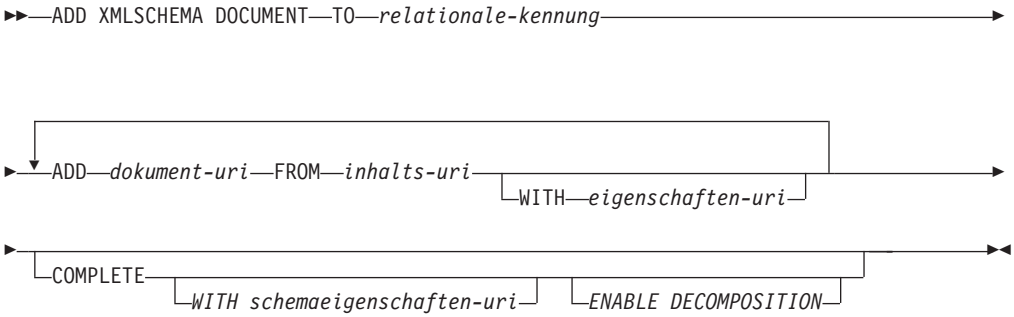
Dazu ist die folgende Berechtigung erforderlich:

- Die Benutzer-ID muss der Eigner des XSR-Objekts sein, der in der Katalogsicht SYSCAT.XSROBJECTS gespeichert ist.

Erforderliche Verbindung

Datenbank

Befehlssyntax



Beschreibung

TO *relationale-kennung*
Gibt den relationalen Namen eines registrierten, aber unvollständigen XML-Schemas an, zu dem weitere Schemadokumente hinzugefügt werden.

ADD *dokument-uri*
Gibt die einheitliche Ressourcen-ID (URI, Universal Resource Identifier) eines XML-Schemadokuments an, das zu diesem Schema hinzugefügt werden soll, als würde ein anderes XML-Dokument auf dieses Dokument verweisen.

FROM *inhalts-uri*
Gibt die URI an, bei der sich das XML-Schemadokument befindet. Es wird nur eine Dateischema-URI unterstützt.

WITH *eigenschaften-URI*
Gibt die URI eines Eigenschaftendokuments für das XML-Schema an. Es wird nur eine Dateischema-URI unterstützt.

COMPLETE
Gibt an, dass keine weiteren XML-Schemadokumente vorhanden sind, die hinzugefügt werden sollen. Bei dieser Angabe wird das Schema geprüft und als verwendbar markiert, wenn keine Fehler gefunden werden.

WITH *schemaeigenschaften-uri*
Gibt die URI eines Eigenschaftendokuments für das XML-Schema an. Es wird nur eine Dateischema-URI unterstützt.

ENABLE DECOMPOSITION
Gibt an, dass dieses Schema zur Dekomposition von XML-Dokumenten verwendet werden soll.

Beispiel

```
ADD XMLSCHEMA DOCUMENT TO JOHNDOE.PRODSCHEMA
   ADD 'http://myPOschema/address.xsd'
   FROM 'file:///c:/TEMP/address.xsd'
```

COMPLETE XMLSCHEMA

Dieser Befehl schließt den Registrierungsprozess eines XML-Schemas im XML-Schema-Repository (XSR) ab.

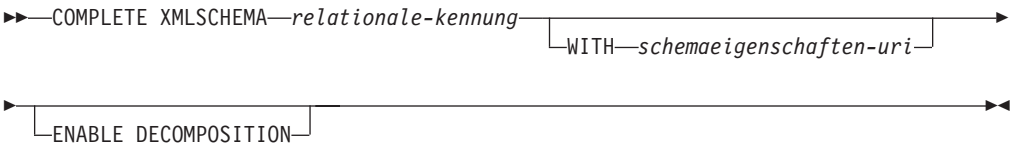
Berechtigung

- Die Benutzer-ID muss der Eigner des XSR-Objekts sein, der in der Katalogsicht SYSCAT.XSROBJECTS gespeichert ist.

Erforderliche Verbindung

Datenbank

Befehlssyntax



Beschreibung

relationale-kennung
Gibt den relationalen Namen eines zuvor mit dem Befehl **REGISTER XMLSCHEMA** registrierten XML-Schemas an. Der relationale Name kann als zweiteilige SQL-Kennung angegeben werden, die aus dem SQL-Schema und dem XML-Schemanamen besteht und das folgende Format hat: *SQL-schema.name*. Das SQL-Standardschema, das im Sonderregister CURRENT SCHEMA definiert ist, wird verwendet, wenn kein Schema angegeben wird.

WITH *schemaeigenschaften-uri*
Gibt die einheitliche Ressourcen-ID (URI, Universal Resource Identifier) eines Eigenschaftendokuments für das XML-Schema an. Es wird nur eine lokale Datei unterstützt, die von einer Dateischema-URI angegeben wird. Ein Schemaeigenschaftendokument kann nur im Abschlussstadium der Registrierung des XML-Schemas angegeben werden.

ENABLE DECOMPOSITION
Gibt an, dass das Schema zur Dekomposition von XML-Instanzdokumenten verwendet werden kann.

Beispiel

```
COMPLETE XMLSCHEMA user1.POschema WITH 'file:///c:/TEMP/schemaProp.xml'
```

Hinweise

Ein XML-Schema kann erst referenziert oder zur Validierung oder Annotation verwendet werden, wenn der Registrierungsprozess des XML-Schemas abgeschlossen ist. Dieser Befehl schließt die XML-Schemaregistrierung für ein XML-Schema ab, die mit dem Befehl **REGISTER XMLSCHEMA** begonnen wurde.

REGISTER XSROBJECT

Registriert ein XML-Objekt in den Datenbankkatalogen. Unterstützte Objekte sind DTDs und externe Entitäten.

Berechtigung

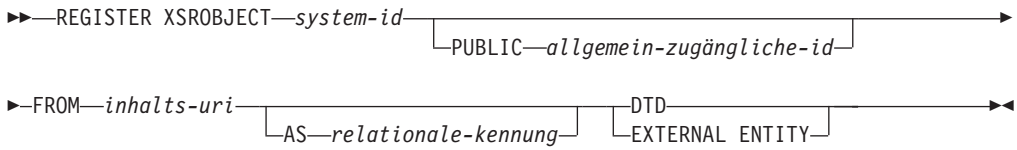
Eine der folgenden Berechtigungen:

- DBADM
- Datenbankberechtigung IMPLICIT_SCHEMA, wenn das SQL-Schema nicht vorhanden ist
- Zugriffsrecht CREATEIN, wenn das SQL-Schema vorhanden ist

Erforderliche Verbindung

Datenbank

Befehlssyntax



Befehlsparameter

system-id
Gibt die System-ID an, die in der Deklaration des XML-Objekts angegeben ist.

PUBLIC *allgemein-zugängliche-id*
Gibt in der Deklaration des XML-Objekts eine optionale, allgemein zugängliche ID an.

FROM *inhalts-uri*
Gibt die URI an, bei der sich der Inhalt eines XML-Schemadokuments befindet. Es wird nur eine lokale Datei unterstützt, die von einer Dateischema-URI angegeben wird.

AS *relationale-kennung*
Gibt einen Namen an, der zum Verweis auf das XML-Objekt verwendet werden kann, das gerade registriert wird. Der relationale Name kann als zweiteilige SQL-Kennung angegeben werden, die aus dem relationalen Schema, einem Punkt als Trennzeichen und dem Namen besteht. Beispiel: "JOHNDOE.EMPLOYEEDTD". Wird kein relationales Schema angegeben, wird das standardmäßige relationale Schema verwendet, das im Sonderregister CURRENT SCHEMA definiert ist. Wird kein Name angegeben, wird automatisch ein Name generiert.

DTD Gibt an, dass das gerade registrierte Objekt ein Dokumenttypdeklarationsdokument (DTD-Dokument) ist.

EXTERNAL ENTITY
Gibt an, dass das gerade registrierte Objekt eine externe Entität ist.

Beispiele

1. Nachstehend finden Sie ein XML-Beispieldokument, das auf eine externe Entität verweist:

```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE copyright [
  <!ELEMENT copyright (#PCDATA)>

]>
<copyright>c</copyright>
```

   Damit dieses Dokument erfolgreich in eine XML-Spalte eingefügt werden kann, muss die externe Entität registriert werden. Mit dem folgenden Befehl wird eine Entität registriert, wobei der Inhalt der Entität lokal im Pfad `C:\TEMP` gespeichert wird:

```
REGISTER XSROBJECT 'http://www.xmlwriter.net/copyright.xml'
   FROM 'c:\temp\copyright.xml' EXTERNAL ENTITY
```

2. Nachstehend finden Sie ein XML-Dokumentfragment, das auf eine DTD verweist:

```
<!--inform the XML processor
  that an external DTD is referenced-->
<?xml version="1.0" standalone="no" ?>

<!--define the location of the
  external DTD using a relative URL address-->
<!DOCTYPE document SYSTEM "http://www.xmlwriter.net/subjects.dtd">

<document>
  <title>Subjects available in Mechanical Engineering.</title>
  <subjectID>2.303</subjectID>
    <subjectname>Fluid Mechanics</subjectname>
 ...
```

   Damit dieses Dokument erfolgreich in eine XML-Spalte eingefügt werden kann, muss die DTD registriert werden. Mit dem folgenden Befehl wird eine DTD registriert, wobei die DTD-Definition lokal im Pfad `C:\TEMP` gespeichert wird und die relationale Kennung, die der DTD zugeordnet werden soll, "TEST.SUBJECTS" lautet:

```
REGISTER XSROBJECT 'http://www.xmlwriter.net/subjects.dtd'
   FROM 'file:///c:/temp/subjects.dtd' AS TEST.SUBJECTS DTD
```

3. Nachstehend finden Sie ein XML-Beispieldokument, das auf eine allgemein zugängliche externe Entität verweist:

```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE copyright [
  <!ELEMENT copyright (#PCDATA)>

]>
<copyright>c</copyright>
```

   Damit dieses Dokument erfolgreich in eine XML-Spalte eingefügt werden kann, muss die allgemein zugängliche externe Entität registriert werden. Mit dem folgenden Befehl wird eine Entität registriert, wobei der Inhalt der Entität lokal im Pfad `C:\TEMP` gespeichert wird:

```
REGISTER XSROBJECT 'http://www.w3.org/xmlspec/copyright.xml'
   PUBLIC '-//W3C//TEXT copyright//EN' FROM 'file:///c:/temp/copyright.xml'
   EXTERNAL ENTITY
```

UPDATE XMLSCHEMA

Aktualisiert ein XML-Schema durch ein anderes XML-Schema im XML-Schema-Repository (XSR).

Berechtigung

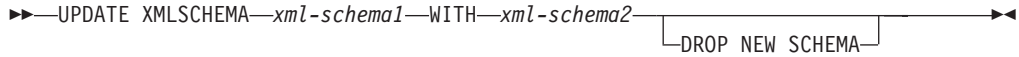
Eine der folgenden Berechtigungen:

- DBADM
- Zugriffsrecht SELECT für die Katalogsichten SYSCAT.XSROBJECTS und SYSCAT.XSROBJECTCOMPONENTS und eine der folgenden Gruppen von Zugriffsrechten:
  - Zugriffsrecht ALTERIN für das XML-Schema, das aktualisiert werden soll, und Zugriffsrecht DROPIN für das neue XML-Schema, falls die Option **DROP NEW SCHEMA** angegeben wird
  - OWNER für das mit xml-schema1 angegebene XML-Schema

Erforderliche Verbindung

Datenbank

Befehlssyntax



Befehlsparameter

UPDATE XMLSCHEMA *xml-schema1*
: Gibt die SQL-Kennung des ursprünglichen XML-Schemas an, das aktualisiert werden soll.

WITH *xml-schema2*
: Gibt die SQL-Kennung des neuen XML-Schemas an, mit dem das ursprüngliche XML-Schema aktualisiert werden soll.

DROP NEW SCHEMA
Gibt an, dass das neue XML-Schema nach der Aktualisierung des ursprünglichen XML-Schemas gelöscht werden soll.

Beispiel

```
UPDATE XMLSCHEMA JOHNDOE.OLDPROD
WITH JOHNDOE.NEWPROD
DROP NEW SCHEMA
```

Der Inhalt des XML-Schemas JOHNDOE.OLDPROD wird durch den Inhalt von JOHNDOE.NEWPROD aktualisiert, und das XML-Schema JOHNDOE.NEWPROD wird gelöscht.

Hinweise

- Das ursprüngliche und das neue XML-Schema müssen kompatibel sein. Der Abschnitt „Kompatibilitätsanforderungen für das Weiterentwickeln von XML-Schemata" enthält ausführliche Informationen zu den Kompatibilitätsanforderungen.
- Damit ein XML-Schema aktualisiert werden kann, müssen das ursprüngliche und das neue Schema im XML-Schema-Repository (XSR) registriert werden.

Anhang D. Übersicht über technische Informationen zu DB2

Technische Informationen zu DB2 liegen in verschiedenen Formaten vor, die auf unterschiedliche Weise abgerufen werden können.

Die technischen Informationen zu DB2 stehen über die folgenden Tools und Methoden zur Verfügung:

- DB2 Information Center
  - Themen (zu Tasks, Konzepten und Referenzinformationen)
  - Beispielprogramme
  - Lernprogramme
- DB2-Bücher
  - PDF-Dateien (für den Download verfügbar)
  - PDF-Dateien (auf der DB2-PDF-DVD)
  - Gedruckte Bücher
- Hilfe für Befehlszeile
  - Hilfe für Befehle
  - Hilfe für Nachrichten

Anmerkung: Die Themen des DB2 Information Center werden häufiger aktualisiert als die PDF- und Hardcopybücher. Um stets die neuesten Informationen zur Verfügung zu haben, sollten Sie die Dokumentationsaktualisierungen installieren, sobald diese verfügbar sind, oder das DB2 Information Center unter ibm.com aufrufen.

Darüber hinaus können Sie auf zusätzliche technische Informationen zu DB2, wie beispielsweise technische Hinweise (Technotes), White Papers und IBM Redbooks, online über ibm.com zugreifen. Rufen Sie dazu die Website 'DB2 Information Management - Software - Library' unter http://www.ibm.com/software/data/sw-library/ auf.

Feedback zur Dokumentation

Senden Sie uns Ihr Feedback zur DB2-Dokumentation! Wenn Sie Anregungen zur Verbesserung der DB2-Dokumentation haben, senden Sie eine E-Mail an db2docs@ca.ibm.com. Das DB2-Dokumentationsteam bearbeitet das gesamte Feedback, kann jedoch nicht im Einzelnen auf Ihre E-Mails antworten. Nennen Sie uns, wenn möglich, konkrete Beispiele, sodass wir die Problemstellung besser beurteilen können. Wenn Sie uns Feedback zu einem bestimmten Thema oder einer bestimmten Hilfedatei senden, geben Sie den entsprechenden Titel sowie die URL an.

Verwenden Sie diese E-Mail-Adresse nicht, wenn Sie sich an den DB2-Kundendienst wenden möchten. Wenn ein technisches Problem bei DB2 vorliegt, das Sie mithilfe der Dokumentation nicht beheben können, fordern Sie beim zuständigen IBM Service-Center Unterstützung an.

Bibliothek mit technischen Informationen zu DB2 im Hardcopy- oder PDF-Format

Die folgenden Tabellen enthalten eine Beschreibung der DB2-Bibliothek, die im IBM Publications Center unter www.ibm.com/e-business/linkweb/publications/servlet/pbi.wss zur Verfügung steht. Über die folgende Adresse können Sie englische Handbücher im PDF-Format sowie übersetzte Versionen zu DB2 Version 10.1 herunterladen: www.ibm.com/support/docview.wss?rs=71&uid=swg2700947.

In den Tabellen sind die Bücher, die in gedruckter Form zur Verfügung stehen, gekennzeichnet; möglicherweise sind diese in Ihrem Land oder Ihrer Region jedoch nicht verfügbar.

Die Formnummer wird bei jeder Aktualisierung eines Handbuchs erhöht. Anhand der nachfolgenden Liste können Sie sicherstellen, dass Sie die jeweils neueste Version des Handbuchs lesen.

Anmerkung: Das *DB2 Information Center* wird häufiger aktualisiert als die PDF- und Hardcopybücher.

Tabelle 76. Technische Informationen zu DB2

Name	IBM Form	In gedruckter Form verfügbar	Letzte Aktualisierung
Administrative API Reference	SC27-3864-00	Ja	April 2012
Administrative Routines and Views	SC27-3865-00	Nein	April 2012
Call Level Interface Guide and Reference Volume 1	SC27-3866-00	Ja	April 2012
Call Level Interface Guide and Reference Volume 2	SC27-3867-00	Ja	April 2012
Command Reference	SC27-3868-00	Ja	April 2012
Datenbankverwaltung - Konzepte und Konfiguration - Referenzinformationen	SC12-4673-00	Ja	April 2012
Dienstprogramme für das Versetzen von Daten - Handbuch und Referenz	SC12-4691-00	Ja	April 2012
Datenbanküberwachung - Handbuch und Referenz	SC12-4674-00	Ja	April 2012
Datenrecovery und hohe Verfügbarkeit - Handbuch und Referenz	SC12-4692-00	Ja	April 2012
Datenbanksicherheit	SC12-4693-00	Ja	April 2012
DB2 Workload Management - Handbuch und Referenz	SC12-4683-00	Ja	April 2012

Tabelle 76. Technische Informationen zu DB2 (Forts.)

Name	IBM Form	In gedruckter Form verfügbar	Letzte Aktualisierung
Developing ADO.NET and OLE DB Applications	SC27-3873-00	Ja	April 2012
Developing Embedded SQL Applications	SC27-3874-00	Ja	April 2012
Developing Java Applications	SC27-3875-00	Ja	April 2012
Developing Perl, PHP, Python, and Ruby on Rails Applications	SC27-3876-00	Nein	April 2012
Developing User-defined Routines (SQL and External)	SC27-3877-00	Ja	April 2012
Getting Started with Database Application Development	GI13-2046-00	Ja	April 2012
Installation und Verwaltung von DB2 unter Linux und Windows - Erste Schritte	GI11-3285-00	Ja	April 2012
Globalisierung	SC12-4694-00	Ja	April 2012
DB2-Server - Installation	SC12-4677-00	Ja	April 2012
IBM Data Server-Clients - Installation	SC12-4678-00	Nein	April 2012
Fehlernachrichten, Band 1	SC12-4686-00	Nein	April 2012
Fehlernachrichten, Band 2	SC12-4687-00	Nein	April 2012
Net Search Extender - Verwaltung und Benutzerhandbuch	SC12-4689-00	Nein	April 2012
Partitionierung und Clustering	SC12-4695-00	Ja	April 2012
pureXML - Handbuch	SC12-4684-00	Ja	April 2012
Spatial Extender - Benutzer- und Referenzhandbuch	SC12-4688-00	Nein	April 2012
SQL Procedural Languages: Application Enablement and Support	SC27-3896-00	Ja	April 2012
SQL Reference Volume 1	SC27-3885-00	Ja	April 2012
SQL Reference Volume 2	SC27-3886-00	Ja	April 2012
Text Search	SC12-4690-00	Ja	April 2012
Fehlerbehebung und Optimieren der Datenbankleistung	SC12-4675-00	Ja	April 2012

Tabelle 76. Technische Informationen zu DB2 (Forts.)

Name	IBM Form	In gedruckter Form verfügbar	Letzte Aktualisierung
Upgrade auf DB2 Version 10.1	SC12-4676-00	Ja	April 2012
Neuerungen in DB2 Version 10.1	SC12-4682-00	Ja	April 2012
XQuery - Referenz	SC12-4685-00	Nein	April 2012

Tabelle 77. Technische Informationen zu DB2 Connect

Name	IBM Form	In gedruckter Form verfügbar	Letzte Aktualisierung
DB2 Connect - Installation und Konfiguration von DB2 Connect Personal Edition	SC12-4679-00	Ja	April 2012
DB2 Connect - Installation und Konfiguration von DB2 Connect-Servern	SC12-4680-00	Ja	April 2012
DB2 Connect - Benutzerhandbuch	SC12-4681-00	Ja	April 2012

Aufrufen der Hilfe für den SQL-Status über den Befehlszeilenprozessor

DB2-Produkte geben für Bedingungen, die aufgrund einer SQL-Anweisung generiert werden können, einen SQLSTATE-Wert zurück. Die SQLSTATE-Hilfe erläutert die Bedeutung der SQL-Statuswerte und der SQL-Statusklassencodes.

Vorgehensweise

Zum Starten der Hilfe für SQL-Statuswerte müssen Sie den Befehlszeilenprozessor öffnen und Folgendes eingeben:

```
? SQL-Status oder ? Klassencode
```

Hierbei steht *SQL-Status* für einen gültigen fünfstelligen SQL-Statuswert und *Klassencode* für die ersten beiden Ziffern dieses Statuswerts.
So kann beispielsweise durch die Eingabe von `? 08003` Hilfe für den SQL-Statuswert 08003 angezeigt werden, durch die Eingabe von `? 08` Hilfe für den Klassencode 08.

Zugriff auf verschiedene Versionen des DB2 Information Center

Die Dokumentation für andere Versionen der DB2-Produkte finden Sie in den jeweiligen Information Centers unter ibm.com.

Informationen zu diesem Vorgang

Für Themen aus DB2 Version 10.1 lautet die URL für das *DB2 Information Center* http://publib.boulder.ibm.com/infocenter/db2luw/v10r1.

Für Themen aus DB2 Version 9.8 lautet die URL des *DB2 Information Center* http://publib.boulder.ibm.com/infocenter/db2luw/v9r8/.

Für Themen aus DB2 Version 9.7 lautet die URL des *DB2 Information Center* http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/.

Für Themen aus DB2 Version 9.5 lautet die URL des *DB2 Information Center* http://publib.boulder.ibm.com/infocenter/db2luw/v9r5.

Für Themen aus DB2 Version 9.1 lautet die URL des *DB2 Information Center* http://publib.boulder.ibm.com/infocenter/db2luw/v9/.

Für Themen aus DB2 Version 8 lautet die URL des *DB2 Information Center* http://publib.boulder.ibm.com/infocenter/db2luw/v8/.

Aktualisieren des auf Ihrem Computer oder Intranet-Server installierten DB2 Information Center

Ein lokal installiertes DB2 Information Center muss regelmäßig aktualisiert werden.

Vorbereitende Schritte

Ein DB2 Version 10.1 Information Center muss bereits installiert sein. Einzelheiten hierzu finden Sie unter „Installation des DB2 Information Center mit dem DB2-Installationsassistenten" in *DB2-Server - Installation*. Alle für die Installation des Information Center geltenden Voraussetzungen und Einschränkungen gelten auch für die Aktualisierung des Information Center.

Informationen zu diesem Vorgang

Ein vorhandenes DB2 Information Center kann automatisch oder manuell aktualisiert werden:

- Mit automatischen Aktualisierungen werden vorhandene Komponenten und Sprachen des Information Center aktualisiert. Ein Vorteil von automatischen Aktualisierungen ist, dass das Information Center im Vergleich zu einer manuellen Aktualisierung nur für einen kurzenZeitraum nicht verfügbar ist. Darüber hinaus können automatische Aktualisierungen so konfiguriert werden, dass sie als Teil anderer, regelmäßig ausgeführter Stapeljobs ausgeführt werden.
- Mit manuellen Aktualisierungen können Sie vorhandene Komponenten und Sprachen des Information Center aktualisieren. Automatische Aktualisierungen reduzieren die Ausfallzeiten während des Aktualisierungsprozesses, Sie müssen jedoch den manuellen Prozess verwenden, wenn Sie Komponenten oder Sprachen hinzufügen möchten. Beispiel: Ein lokales Information Center wurde ursprünglich sowohl mit englischer als auch mit französischer Sprachunterstützung installiert; nun soll auch die deutsche Sprachunterstützung installiert werden. Bei einer manuellen Aktualisierung werden sowohl eine Installation der deutschen Sprachunterstützung als auch eine Aktualisierung der vorhandenen Komponenten und Sprachen des Information Center durchgeführt. Sie müssen jedoch bei einer manuellen Aktualisierung das Information Center manuell stoppen, aktualisieren und erneut starten. Das Information Center ist während des gesamten Aktualisierungsprozesses nicht verfügbar. Während des automatischen Aktualisierungsprozesses kommt es zu einem Ausfall des Information Center, und es wird erst wieder nach der Aktualisierung erneut gestartet.

Dieser Abschnitt enthält Details zum Prozess der automatischen Aktualisierung. Anweisungen zur manuellen Aktualisierung finden Sie im Abschnitt „Manuelles Aktualisieren des auf Ihrem Computer oder Intranet-Server installierten DB2 Information Center".

Vorgehensweise

Gehen Sie wie folgt vor, um das auf Ihrem Computer bzw. Intranet-Server installierte DB2 Information Center automatisch zu aktualisieren:

1. Unter Linux:
   a. Navigieren Sie zu dem Pfad, in dem das Information Center installiert ist. Standardmäßig ist das DB2 Information Center im Verzeichnis `/opt/ibm/db2ic/V10.1` installiert.
   b. Navigieren Sie vom Installationsverzeichnis in das Verzeichnis `doc/bin`.
   c. Führen Sie das Script `update-ic` aus:
      ```
      update-ic
      ```
2. Unter Windows:
   a. Öffnen Sie ein Befehlsfenster.
   b. Navigieren Sie zu dem Pfad, in dem das Information Center installiert ist. Standardmäßig ist das DB2 Information Center im Verzeichnis `<Programme>\IBM\DB2 Information Center\Version 10.1` installiert, wobei `<Programme>` das Verzeichnis der Programmdateien angibt.
   c. Navigieren Sie vom Installationsverzeichnis in das Verzeichnis `doc\bin`.
   d. Führen Sie die Datei `update-ic.bat` aus:
      ```
      update-ic.bat
      ```

Ergebnisse

Das DB2 Information Center wird automatisch erneut gestartet. Standen Aktualisierungen zur Verfügung, zeigt das Information Center die neuen und aktualisierten Abschnitte an. Waren keine Aktualisierungen für das Information Center verfügbar, wird eine entsprechende Nachricht zum Protokoll hinzugefügt. Die Protokolldatei befindet sich im Verzeichnis `doc\eclipse\configuration`. Der Name der Protokolldatei ist eine Zufallszahl. Beispiel: `1239053440785.log`.

Manuelles Aktualisieren des auf Ihrem Computer oder Intranet-Server installierten DB2 Information Center

Wenn Sie das DB2 Information Center lokal installiert haben, können Sie Dokumentationsaktualisierungen von IBM abrufen und installieren.

Informationen zu diesem Vorgang

Zur manuellen Aktualisierung des lokal installierten *DB2 Information Center* sind die folgenden Schritte erforderlich:

1. Stoppen Sie das *DB2 Information Center* auf Ihrem Computer und starten Sie das Information Center im Standalone-Modus erneut. Die Ausführung des Information Center im Standalone-Modus verhindert, dass andere Benutzer in Ihrem Netz auf das Information Center zugreifen, und ermöglicht das Anwenden von Aktualisierungen. Die Workstationversion des DB2 Information Center wird stets im Standalone-Modus ausgeführt.

2. Verwenden Sie die Aktualisierungsfunktion, um zu prüfen, welche Aktualisierungen verfügbar sind. Falls Aktualisierungen verfügbar sind, die Sie installieren müssen, können Sie die Aktualisierungsfunktion verwenden, um diese abzurufen und zu installieren.

   Anmerkung: Wenn es in der verwendeten Umgebung erforderlich ist, die Aktualisierungen für das *DB2 Information Center* auf einer Maschine zu installieren, die nicht über ein Verbindung zum Internet verfügt, spiegeln Sie die Aktualisierungssite auf ein lokales Dateisystem und verwenden Sie dabei eine Maschine, die mit dem Internet verbunden ist und auf der das *DB2 Information Center* installiert ist. Wenn viele Benutzer Ihres Netzes die Dokumentationsaktualisierungen installieren sollen, können Sie die Zeit, die jeder einzelne Benutzer für die Aktualisierungen benötigt, reduzieren, indem Sie die Aktualisierungssite lokal spiegeln und ein Proxy dafür erstellen.
   Ist dies der Fall, verwenden Sie die Aktualisierungsfunktion, um die Pakete abzurufen. Die Aktualisierungsfunktion ist jedoch nur im Standalone-Modus verfügbar.
3. Stoppen Sie das im Standalone-Modus gestartete Information Center und starten Sie das *DB2 Information Center* auf Ihrem Computer erneut.

Anmerkung: Unter Windows 2008 und Windows Vista (und neueren Versionen) müssen die in diesem Abschnitt aufgeführten Befehle mit Administratorberechtigung ausgeführt werden. Zum Öffnen einer Eingabeaufforderung oder eines Grafiktools mit vollen Administratorberechtigungen klicken Sie mit der rechten Maustaste die Verknüpfung an und wählen Sie **Als Administrator ausführen** aus.

Vorgehensweise

Gehen Sie wie folgt vor, um das auf Ihrem Computer bzw. Intranet-Server installierte *DB2 Information Center* zu aktualisieren:

1. Stoppen Sie das *DB2 Information Center*.
   - Unter Windows: Klicken Sie **Start** > **Systemsteuerung** > **Verwaltung** > **Dienste** an. Klicken Sie mit der rechten Maustaste das **DB2 Information Center** an und wählen Sie **Beenden** aus.
   - Unter Linux: Geben Sie den folgenden Befehl ein:
     ```
     /etc/init.d/db2icdv10 stop
     ```
2. Starten Sie das Information Center im Standalone-Modus.
   - Unter Windows:
     a. Öffnen Sie ein Befehlsfenster.
     b. Navigieren Sie zu dem Pfad, in dem das Information Center installiert ist. Standardmäßig ist das *DB2 Information Center* im Verzeichnis *Programme*\IBM\DB2 Information Center\Version 10.1 installiert, wobei *Programme* das Verzeichnis der Programmdateien angibt.
     c. Navigieren Sie vom Installationsverzeichnis in das Verzeichnis doc\bin.
     d. Führen Sie die Datei help_start.bat aus:
        ```
        help_start.bat
        ```
   - Unter Linux:
     a. Navigieren Sie zu dem Pfad, in dem das Information Center installiert ist. Standardmäßig ist das *DB2 Information Center* im Verzeichnis /opt/ibm/db2ic/V10.1 installiert.
     b. Navigieren Sie vom Installationsverzeichnis in das Verzeichnis doc/bin.

c. Führen Sie das Script `help_start` aus:

```
help_start
```

Der standardmäßig auf dem System verwendete Web-Browser wird geöffnet und zeigt die Standalone-Version des Information Center an.

3. Klicken Sie die Aktualisierungsschaltfläche () an. (JavaScript muss im verwendeten Browser aktiviert sein.) Klicken Sie im rechten Fenster des Information Center die Schaltfläche für die Suche nach Aktualisierungen an. Eine Liste der Aktualisierungen für die vorhandene Dokumentation wird angezeigt.
4. Wählen Sie zum Initiieren des Installationsprozesses die gewünschten Aktualisierungen aus und klicken Sie anschließend die Schaltfläche für die Installation der Aktualisierungen an.
5. Klicken Sie nach Abschluss des Installationsprozesses **Fertigstellen** an.
6. Stoppen Sie das im Standalone-Modus gestartete Information Center:
   - Unter Windows: Navigieren Sie innerhalb des Installationsverzeichnisses zum Verzeichnis `doc\bin`, und führen Sie die Datei `help_end.bat` aus:

     ```
     help_end.bat
     ```

     Anmerkung: Die Stapeldatei `help_end` enthält die Befehle, die erforderlich sind, um die Prozesse, die mit der Stapeldatei `help_start` gestartet wurden, ordnungsgemäß zu stoppen. Verwenden Sie nicht die Tastenkombination `Strg+C` oder eine andere Methode, um `help_start.bat` zu stoppen.
   - Unter Linux: Navigieren Sie innerhalb des Installationsverzeichnisses zum Verzeichnis `doc/bin`, und führen Sie das Script `help_end` aus:

     ```
     help_end
     ```

     Anmerkung: Das Script `help_end` enthält die Befehle, die erforderlich sind, um die Prozesse, die mit dem Script `help_start` gestartet wurden, ordnungsgemäß zu stoppen. Verwenden Sie keine andere Methode, um das Script `help_start` zu stoppen.
7. Starten Sie das *DB2 Information Center* erneut.
   - Unter Windows: Klicken Sie **Start** > **Systemsteuerung** > **Verwaltung** > **Dienste** an. Klicken Sie mit der rechten Maustaste das **DB2 Information Center** an und wählen Sie **Start** aus.
   - Unter Linux: Geben Sie den folgenden Befehl ein:

     ```
     /etc/init.d/db2icdv10 start
     ```

Ergebnisse

Im aktualisierten *DB2 Information Center* werden die neuen und aktualisierten Themen angezeigt.

DB2-Lernprogramme

Die DB2-Lernprogramme unterstützen Sie dabei, sich mit den unterschiedlichen Aspekten der DB2-Produkte vertraut zu machen. Die Lerneinheiten bieten eine in einzelne Schritte unterteilte Anleitung.

Vorbereitungen

Die XHTML-Version des Lernprogramms kann über das Information Center unter http://publib.boulder.ibm.com/infocenter/db2luw/v10r1/ angezeigt werden.

In einigen der Lerneinheiten werden Beispieldaten und Codebeispiele verwendet. Informationen zu bestimmten Voraussetzungen für die Ausführung der Tasks finden Sie in der Beschreibung des Lernprogramms.

DB2-Lernprogramme

Klicken Sie zum Anzeigen des Lernprogramms den Titel an.

„pureXML" in *pureXML - Handbuch*
Einrichten einer DB2-Datenbank, um XML-Daten zu speichern und Basisoperationen mit dem nativen XML-Datenspeicher auszuführen.

Informationen zur Fehlerbehebung in DB2

Es steht eine breite Palette verschiedener Informationen zur Fehlerbestimmung und Fehlerbehebung zur Verfügung, um Sie bei der Verwendung von DB2-Datenbankprodukten zu unterstützen.

DB2-Dokumentation
Informationen zur Fehlerbehebung stehen im Handbuch *Fehlerbehebung und Optimieren der Datenbankleistung* oder im Abschnitt mit grundlegenden Informationen zu Datenbanken im *DB2 Information Center* zur Verfügung, darunter:

- Informationen zum Eingrenzen und Aufdecken von Problemen mithilfe der Diagnosetools und -dienstprogramme von DB2.
- Lösungsvorschläge zu den am häufigsten auftretenden Problemen.
- Ratschläge zum Lösen anderer Probleme, die bei Verwendung der DB2-Datenbankprodukte auftreten können.

IBM Support Portal
Im IBM Support Portal finden Sie Informationen zu Problemen und den möglichen Ursachen und Fehlerbehebungsmaßnahmen. Die Website mit technischer Unterstützung enthält Links zu den neuesten DB2-Veröffentlichungen, technischen Hinweisen (TechNotes), APARs (Authorized Program Analysis Reports) und Fehlerkorrekturen, Fixpacks sowie weiteren Ressourcen. Sie können diese Wissensbasis nach möglichen Lösungen für aufgetretene Probleme durchsuchen.

Sie können auf das IBM Support Portal über die folgende Website zugreifen: http://www.ibm.com/support/entry/portal/Overview/Software/Information_Management/DB2_for_Linux,_UNIX_and_Windows.

Bedingungen

Die Berechtigungen zur Nutzung dieser Veröffentlichungen werden Ihnen auf der Basis der folgenden Bedingungen gewährt.

Anwendbarkeit: Diese Bedingungen gelten zusätzlich zu den Nutzungsbedingungen für die IBM Website.

Persönliche Nutzung: Sie dürfen diese Veröffentlichungen für Ihre persönliche, nicht kommerzielle Nutzung unter der Voraussetzung vervielfältigen, dass alle Eigentumsvermerke erhalten bleiben. Sie dürfen diese Veröffentlichungen oder Teile dieser Veröffentlichungen ohne ausdrückliche Genehmigung von IBM nicht weitergeben, anzeigen oder abgeleitete Werke davon erstellen.

Kommerzielle Nutzung: Sie dürfen diese Veröffentlichungen nur innerhalb Ihres Unternehmens und unter der Voraussetzung, dass alle Eigentumsvermerke erhalten bleiben, vervielfältigen, weitergeben und anzeigen. Sie dürfen diese Veröffentlichungen oder Teile dieser Veröffentlichungen ohne ausdrückliche Genehmigung von IBM außerhalb Ihres Unternehmens nicht vervielfältigen, weitergeben, anzeigen oder abgeleitete Werke davon erstellen.

Rechte: Abgesehen von den hier gewährten Berechtigungen erhalten Sie keine weiteren Berechtigungen, Lizenzen oder Rechte (veröffentlicht oder stillschweigend) in Bezug auf die Veröffentlichungen oder darin enthaltene Informationen, Daten, Software oder geistiges Eigentum.

IBM behält sich das Recht vor, die in diesem Dokument gewährten Berechtigungen nach eigenem Ermessen zurückzuziehen, wenn sich die Nutzung der Veröffentlichungen für IBM als nachteilig erweist oder wenn die obigen Nutzungsbestimmungen nicht genau befolgt werden.

Sie dürfen diese Informationen nur in Übereinstimmung mit allen anwendbaren Gesetzen und Vorschriften, einschließlich aller US-amerikanischen Exportgesetze und Verordnungen, herunterladen und exportieren.

IBM übernimmt keine Gewährleistung für den Inhalt dieser Informationen. Diese Veröffentlichungen werden auf der Grundlage des gegenwärtigen Zustands (auf "as-is"-Basis) und ohne eine ausdrückliche oder stillschweigende Gewährleistung für die Handelsüblichkeit, die Verwendungsfähigkeit oder die Freiheit der Rechte Dritter zur Verfügung gestellt.

IBM Marken: IBM, das IBM Logo und ibm.com sind Marken oder eingetragene Marken der International Business Machines Corporation. Weitere Produkt- oder Servicenamen können Marken von oder anderen Herstellern sein. IBM oder anderen Herstellern sein. Eine aktuelle Liste der IBM Marken finden Sie auf der Webseite www.ibm.com/legal/copytrade.shtml.

Anhang E. Bemerkungen

Die vorliegenden Informationen wurden für Produkte und Services entwickelt, die auf dem deutschen Markt angeboten werden. Die Informationen über Produkte anderer Hersteller als IBM basieren auf den zum Zeitpunkt der ersten Veröffentlichung dieses Dokuments verfügbaren Informationen und können geändert werden.

Möglicherweise bietet IBM die in dieser Dokumentation beschriebenen Produkte, Services oder Funktionen in anderen Ländern nicht an. Informationen über die gegenwärtig im jeweiligen Land verfügbaren Produkte und Services sind beim zuständigen IBM Ansprechpartner erhältlich. Hinweise auf IBM Lizenzprogramme oder andere IBM Produkte bedeuten nicht, dass nur Programme, Produkte oder Services von IBM verwendet werden können. Anstelle der IBM Produkte, Programme oder Services können auch andere, ihnen äquivalente Produkte, Programme oder Services verwendet werden, solange diese keine gewerblichen oder anderen Schutzrechte von IBM verletzen. Die Verantwortung für den Betrieb von Produkten, Programmen und Services anderer Anbieter liegt beim Kunden.

Für in diesem Handbuch beschriebene Erzeugnisse und Verfahren kann es IBM Patente oder Patentanmeldungen geben. Mit der Auslieferung dieses Handbuchs ist keine Lizenzierung dieser Patente verbunden. Lizenzanforderungen sind schriftlich an folgende Adresse zu richten (Anfragen an diese Adresse müssen auf Englisch formuliert werden):

IBM Director of Licensing
IBM Europe, Middle East & Africa
Tour Descartes
2, avenue Gambetta
92066 Paris La Defense
France

Trotz sorgfältiger Bearbeitung können technische Ungenauigkeiten oder Druckfehler in dieser Veröffentlichung nicht ausgeschlossen werden. Die hier enthaltenen Informationen werden in regelmäßigen Zeitabständen aktualisiert und als Neuausgabe veröffentlicht. IBM kann ohne weitere Mitteilung jederzeit Verbesserungen und/oder Änderungen an den in dieser Veröffentlichung beschriebenen Produkten und/oder Programmen vornehmen.

Verweise in diesen Informationen auf Websites anderer Anbieter werden lediglich als Service für den Kunden bereitgestellt und stellen keinerlei Billigung des Inhalts dieser Websites dar. Das über diese Websites verfügbare Material ist nicht Bestandteil des Materials für dieses IBM Produkt. Die Verwendung dieser Websites geschieht auf eigene Verantwortung.

Werden an IBM Informationen eingesandt, können diese beliebig verwendet werden, ohne dass eine Verpflichtung gegenüber dem Einsender entsteht.

Lizenznehmer des Programms, die Informationen zu diesem Produkt wünschen mit der Zielsetzung: (i) den Austausch von Informationen zwischen unabhängig voneinander erstellten Programmen und anderen Programmen (einschließlich des vorliegenden Programms) sowie (ii) die gemeinsame Nutzung der ausgetauschten Informationen zu ermöglichen, wenden sich an folgende Adresse:

IBM Canada Limited
U59/3600
3600 Steeles Avenue East
Markham, Ontario L3R 9Z7
CANADA

Die Bereitstellung dieser Informationen kann unter Umständen von bestimmten Bedingungen - in einigen Fällen auch von der Zahlung einer Gebühr - abhängig sein.

Die Lieferung des im Dokument aufgeführten Lizenzprogramms sowie des zugehörigen Lizenzmaterials erfolgt auf der Basis der IBM Rahmenvereinbarung bzw. der Allgemeinen Geschäftsbedingungen von IBM, der IBM Internationalen Nutzungsbedingungen für Programmpakete oder einer äquivalenten Vereinbarung.

Alle in diesem Dokument enthaltenen Leistungsdaten stammen aus einer kontrollierten Umgebung. Die Ergebnisse, die in anderen Betriebsumgebungen erzielt werden, können daher erheblich von den hier erzielten Ergebnissen abweichen. Einige Daten stammen möglicherweise von Systemen, deren Entwicklung noch nicht abgeschlossen ist. Eine Gewährleistung, dass diese Daten auch in allgemein verfügbaren Systemen erzielt werden, kann nicht gegeben werden. Darüber hinaus wurden einige Daten unter Umständen durch Extrapolation berechnet. Die tatsächlichen Ergebnisse können davon abweichen. Benutzer dieses Dokuments sollten die entsprechenden Daten in ihrer spezifischen Umgebung prüfen.

Alle Informationen zu Produkten anderer Anbieter stammen von den Anbietern der aufgeführten Produkte, deren veröffentlichen Ankündigungen oder anderen allgemein verfügbaren Quellen. IBM hat diese Produkte nicht getestet und kann daher keine Aussagen zu Leistung, Kompatibilität oder anderen Merkmalen machen. Fragen zu den Leistungsmerkmalen von Produkten anderer Anbieter sind an den jeweiligen Anbieter zu richten.

Aussagen über Pläne und Absichten von IBM unterliegen Änderungen oder können zurückgenommen werden und repräsentieren nur die Ziele von IBM.

Diese Veröffentlichung kann Beispiele für Daten und Berichte des alltäglichen Geschäftsablaufes enthalten. Sie sollen nur die Funktionen des Lizenzprogramms illustrieren; sie können Namen von Personen, Firmen, Marken oder Produkten enthalten. Alle diese Namen sind frei erfunden; Ähnlichkeiten mit tatsächlichen Namen und Adressen sind rein zufällig.

COPYRIGHTLIZENZ:

Diese Veröffentlichung enthält Musteranwendungsprogramme, die in Quellensprache geschrieben sind und Programmiertechniken in verschiedenen Betriebsumgebungen veranschaulichen. Sie dürfen diese Musterprogramme kostenlos kopieren, ändern und verteilen, wenn dies zu dem Zweck geschieht, Anwendungsprogramme zu entwickeln, zu verwenden, zu vermarkten oder zu verteilen, die mit der Anwendungsprogrammierschnittstelle für die Betriebsumgebung konform sind, für die diese Musterprogramme geschrieben werden. Diese Beispiele wurden nicht unter allen denkbaren Bedingungen getestet. Daher kann IBM die Zuverlässigkeit, Wartungsfreundlichkeit oder Funktion dieser Programme weder zusagen noch gewährleisten. Die Musterprogramme werden ohne Wartung (auf "as-is"-Basis) und ohne jegliche Gewährleistung zur Verfügung gestellt. IBM haftet nicht für Schäden, die durch Verwendung der Musterprogramme entstehen.

Kopien oder Teile der Musterprogramme bzw. daraus abgeleiteter Code müssen folgenden Copyrightvermerk beinhalten:

© (*Name Ihrer Firma*) (*Jahr*). Teile des vorliegenden Codes wurden aus Musterprogrammen der IBM Corp. abgeleitet. © Copyright IBM Corp. *_Jahr/Jahre angeben_*. Alle Rechte vorbehalten.

Marken

IBM, das IBM Logo und ibm.com sind Marken oder eingetragene Marken der IBM Corporation in den USA und/oder anderen Ländern. Weitere Produkt- oder Servicenamen können Marken von oder anderen Herstellern sein. IBM oder anderen Herstellern sein. Eine aktuelle Liste der IBM Marken finden Sie auf der Webseite „Copyright and trademark information" unter www.ibm.com/legal/copytrade.shtml.

Die folgenden Namen sind Marken oder eingetragene Marken anderer Unternehmen.

- Linux ist eine eingetragene Marke von Linus Torvalds in den USA und/oder anderen Ländern.
- Java und alle auf Java basierenden Marken und Logos sind Marken oder eingetragene Marken von Oracle und/oder ihren verbundenen Unternehmen.
- UNIX ist eine eingetragene Marke von The Open Group in den USA und anderen Ländern.
- Intel, das Intel-Logo, Intel Inside, Intel Inside logo, Celeron, Intel SpeedStep, Itanium und Pentium sind Marken oder eingetragene Marken der Intel Corporation oder deren Tochtergesellschaften in den USA und anderen Ländern.
- Microsoft, Windows, Windows NT und das Windows-Logo sind Marken der Microsoft Corporation in den USA und/oder anderen Ländern.

Weitere Unternehmens-, Produkt- oder Servicenamen können Marken anderer Hersteller sein.

Index

Sonderzeichen

A

B

C

D

E

F

G

H

I

R

S

T

U

V

W

X

Z

SC12-4684-00

Spine information:

IBM DB2 10.1 for Linux, UNIX and Windows

pureXML - Handbuch