IBM DB2 10.1
for Linux, UNIX, and Windows

开发用户定义的例程（SQL 和外部例程）

S151-1761-00

IBM DB2 10.1
for Linux, UNIX, and Windows

开发用户定义的例程（SQL 和外部例程）

S151-1761-00

注意

使用此信息及其支持的产品前，请先阅读第 295 页的附录 B，『声明』下的常规信息。

修订版声明

此文档包含 IBM 的所有权信息。它在许可协议中提供，且受版权法的保护。本出版物中包含的信息不包括对任何产品的保证，且提供的任何语句都不需要如此解释。

您可在线或通过当地的 IBM 代表处订购 IBM 出版物。

- 要在线订购出版物，请转至 IBM 出版物中心，网址为: http://www.ibm.com/shop/publications/order
- 要查找当地的 IBM 代表处，请转至 IBM 全球联系人目录，网址为: http://www.ibm.com/planetwide/

要从美国或加拿大的 DB2 市场和销售部订购 DB2 出版物，请致电 1-800-IBM-4YOU（426-4968）。

您发送信息给 IBM 后，即授予 IBM 非独占权限，IBM 可以按它认为适当的任何方式使用或分发您所提供的任何信息而无须对您承担任何责任。

目录

关于本书

本书提供关于在没有任何内置例程提供了所需功能时开发用户定义的例程的信息。

本书的使用对象

本书面向所有层次的数据库架构设计师、数据库管理员和应用程序开发者。

- 数据库架构设计师，他们希望了解如何创建例程对象以及使用例程对象将可以在数据库体系结构中的多个上下文中反复使用的 SQL 和相关逻辑模块化。
- 数据库管理员，他们希望了解如何通过在数据库管理系统中使用用户定义的例程来创建、管理、部署和保护系统以及诊断系统问题和提高系统性能。
- 应用程序开发者，他们希望了解如何以及何时将 SQL 语句和应用程序逻辑封装到例程中以便提高应用程序的模块化水平和性能，并希望了解设计、创建和构建用户定义的例程的逐步方法。应用程序开发者应该有使用其中一种受支持的例程开发编程语言（C、C++、Java、COBOL、C#、Visual Basic 或另一种受支持的 .NET CLR 编程语言）来编写 SQL 语句和进行编程的经验。

第 1 章 开发例程

通常，在不存在能够提供所需功能的内置例程时，您需要开发例程。

开始之前

- 阅读并理解基本例程概念:
  - 要了解例程的类型、有用的例程应用、用于开发例程的工具、例程最佳实践以及其他信息，请参阅以下主题:
    - 第 3 页的第 2 章，『例程概述』
- 了解可以帮助您更快更方便地开发例程的可用例程开发工具:
  - 要了解可用于开发例程的工具，请参阅以下主题:
    - 第 29 页的『用于开发例程的工具』

关于此任务

例程和例程实现分为不同的功能类型，但是，对于所有例程而言，开发例程的基本步骤通常相同。您必须确定所要创建的例程的类型、要使用的实现、定义例程接口、开发例程逻辑、执行 SQL 以创建例程、测试例程并部署例程以供广泛使用。

根据您选择开发的例程的类型不同，必须遵循一些特定的过程。本主题将指示您转到适当的主题以开始开发例程。

过程

1. 确定是否已有能够满足您的例程需求的现有内置例程。
   - 如果有内置例程能够满足需要，那么您可能想参阅 第 263 页的第 10 章，『调用例程』。
2. 确定要开发的例程的功能类型。
3. 确定要使用的例程实现。
   - 如果需要 SQL 例程，请参阅有关 SQL 例程 (SQL PL) 的信息。
   - 如果需要外部例程，请参阅有关第 51 页的第 3 章，『外部例程』的信息。

下一步做什么

尽管 SQL 例程与外部例程的开发方法类似，但还是有所差别。对于这两类例程而言，您必须先设计逻辑，然后，为了在数据库中创建例程，必须执行特定于例程功能类型的 CREATE 语句。这些例程创建语句包括 CREATE PROCEDURE、CREATE FUNCTION 和 CREATE METHOD。特定于每个 CREATE 语句的子句将定义例程的特征，其中包括例程名、例程参数的数目和类型以及关于例程逻辑的详细信息。DB2® 使用这些子句所提供的信息来标识何时调用例程，并在调用时运行该例程。成功地执行用于创建例程的 CREATE 语句之后，就会在数据库中创建该例程。例程的特征存储在可供用户查询的 DB2 系统目录表中。执行 CREATE 语句以创建例程的过程又称为定义例程或注册例程。

由于外部例程在用户创建的库或类中实现它们的逻辑（这些库和类驻留在数据库文件系统中），因此，您需要执行其他步骤以进行逻辑编程、执行构建以及正确地放置所生成的库或类文件。

在开发例程之后，您可能想执行下列操作:

- 调试例程
- 将例程部署到生产环境
- 将执行例程所需的特权授予用户
- 调用例程
- 调整例程的性能

例程

例程是可以封装编程和数据库逻辑的数据库对象，而编程和数据库逻辑可以像编程子例程一样从各种 SQL 接口调用。

例程可以是内置的（这意味着例程随产品提供）或用户定义的（这意味着用户可以创建例程）。可以使用 SQL 语句及/或编程语言来实现例程。不同类型的例程会提供不同的接口，这些接口可用来扩展 SQL 语句、客户机应用程序以及某些数据库对象的功能。

有关 DB2 数据库系统所支持的例程和实现类型的完整视图，请参阅主题: 第 4 页的『例程的类型』。

例程具有许多功能，可以说明例程为什么会有如此多的有用应用。

第 2 章 例程概述

例程是一种数据库对象，您可以使用这种数据库对象来封装可以像编程子例程一样调用的逻辑。在数据库或数据库应用程序体系结构中有许多实用的例程应用程序。您可以使用例程来改进整体的数据库设计、数据库性能和数据安全性，以及实现基本审计机制等。

使用例程的益处

使用例程有下列益处:

封装可以从 SQL 接口调用的应用程序逻辑

在包含许多具有公共要求的不同客户机应用程序的环境中，有效地使用例程有助于简化代码复用、代码标准化和代码维护。如果需要在使用了例程的环境中更改公共应用程序行为的特定方面，那么只需修改封装了该行为的例程。如果未使用例程，就必须在每个应用程序中更改应用程序逻辑。

启用对其他数据库对象的受控访问

可以使用例程来控制对数据库对象进行的访问。用户可能无权一般化地发出特定 SQL 语句（例如 CREATE TABLE）；但是，您可以授权该用户调用包含该语句的一个或多个特定实现的例程，从而通过封装特权简化特权管理工作。

通过降低网络流量来提高应用程序性能

在客户机上运行应用程序时，将把每个 SQL 语句单独地从客户机发送到数据库服务器以便执行，并且将单独地返回每个结果集。这可能会导致网络流量过高。如果能够确定需要进行大量数据库交互但不需要进行很多用户交互的工作，那么最好将此部分工作安装在服务器上，以便最大程度地降低网络流量以及在功能更强大的数据库服务器上完成该工作。

能够更快更高效地执行 SQL

因为例程是数据库对象，所以它们在传输 SQL 请求和数据方面比客户机应用程序更高效。因此，在例程中执行 SQL 语句的效果比在客户机应用程序中执行这些语句更好。如果创建例程时指定了 NOT FENCED 子句，那么这些例程将在数据库管理器所在的进程中运行，并因此可以使用共享内存进行通信，这有助于提高应用程序性能。

允许使用不同编程语言实现的逻辑进行互操作

由于代码模块可能由不同程序员使用不同编程语言实现，并且通常最好尽可能地复用代码，因此 DB2 例程设计成支持较高的互操作性水平。

- 使用一种编程语言编写的客户机应用程序可以调用使用另一编程语言实现的例程。例如，C 客户机应用程序可以调用 .NET 公共语言运行时例程。
- 例程可以调用其他例程，而与例程类型或例程实现无关。例如，Java 过程可以调用嵌入式 SQL 标量函数。
- 在运行于一款操作系统上的 DB2 客户机中，可以调用在运行于另一款操作系统上的数据库服务器中创建的例程。

这些益处仅仅是使用例程可以实现的众多益处的一部分。使用例程可以使各种用户受益，这些用户包括数据库管理员、数据库架构设计师和数据库应用程序开发者。因此，您可能想对例程的许多非常有用的应用领域进行探查。

您可以通过各种类型的例程来满足特定功能需求和各种例程实现。对例程类型和实现所作的选择可影响以前列出的益处的表现水平。通常，例程是一种功能强大的逻辑封装方法，使您能够扩展 SQL、改善结构和维护以及有可能提高应用程序性能。

例程的类型

有许多不同类型的例程。例程可以按不同方式进行分组，但主要是按系统或用户定义、功能以及实现进行分组。

支持的例程定义如下所示:

- 第 6 页的『内置例程』
- 第 6 页的『用户定义的例程』

支持的例程功能类型如下所示:

- 第 9 页的『例程: 过程』（也称为存储过程）
- 第 10 页的『例程: 函数』
- 第 14 页的『例程: 方法』

支持的例程实现如下所示:

- 第 18 页的『内置例程实现』
- 第 18 页的『有源例程实现』
- 第 18 页的『SQL 例程实现』
- 第 18 页的『外部例程实现』

下图说明例程的分类层次结构。所有例程可以是内置例程或用户定义的例程。例程的功能类型是在深灰色/蓝色框中，而支持的例程实现是在浅灰色/橘黄色框中。会着重说明内置例程实现，因为这种实现是唯一的。

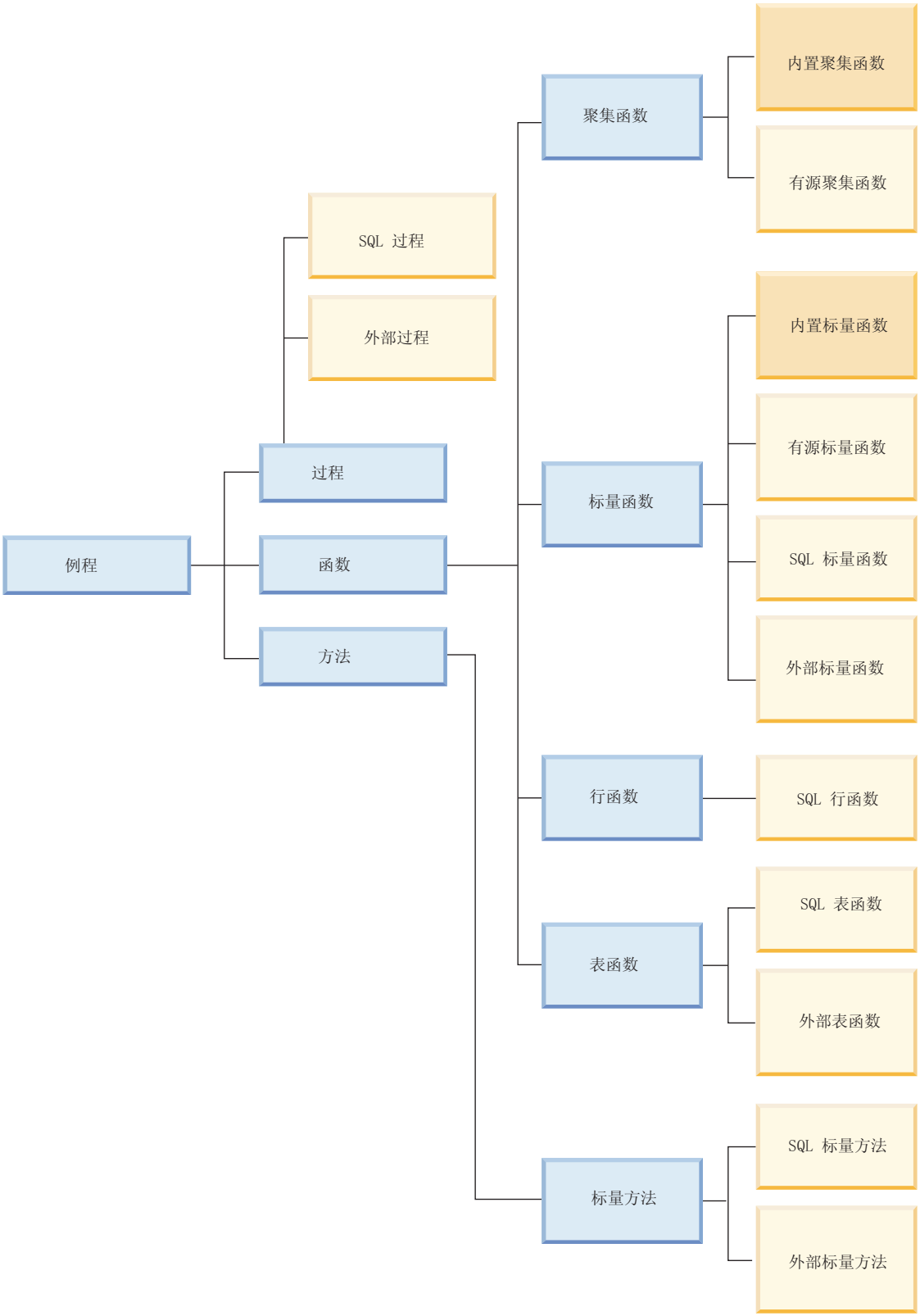


图 1. 例程分类

各种例程都针对扩展 SQL 语言功能以及开发更多模块化数据库应用程序提供广泛支持。

内置和用户定义的例程

对例程进行分类的最简单直接的一种方法是将例程划分为内置例程和用户定义的例程。

*内置例程*是随产品提供的例程。这些例程为从管理功能到数据库系统和目录报告等任务提供广泛支持。它们可立即使用，且不需要任何必备的设置或注册步骤，虽然用户需要必要的特权才能调用这些例程。

*用户定义的例程*是用户自己创建的例程。用户定义的例程提供一种方法，让用户可以扩展 SQL 语言，以实现超前支持。可以使用各种方法来实现用户定义的例程，其中包括将内置例程用作源（复用内置例程的逻辑）、仅使用 SQL 语句或者将 SQL 与另一种编程语言搭配使用。

内置例程

内置例程是随产品提供的例程。这些例程为从管理功能到数据库系统和目录报告等任务提供广泛例程支持。

它们具有立即可供使用以及不要求进行先决条件设置或执行例程注册步骤的特征，尽管用户必须具有特权才能调用这些例程。这些例程可包括内置例程，还称为 SQL 管理例程。

内置例程提供了标准的运算符支持以及基本标量函数和聚集函数支持。您应该首先选择使用内置例程，这是因为，它们具有强类型化特征并能够提供最佳性能。请不要创建与内置例程的行为重复的外部例程。外部例程无法实现与内置例程相当的性能和安全性。

您可以使用的其他内置例程随 DB2 数据库系统在 SYSPROC、SYSFUN 和 SYSTOOLS 模式中提供。这些例程基本上是由系统定义并随产品提供的 SQL 和外部例程。虽然这些附加例程随 DB2 数据库系统提供，但它们不是内置例程。而是，它们作为预先安装的用户定义的例程实现。这些例程通常封装实用程序函数。其中一些示例包括：SNAP_GET_TAB、SNAP_WRITE_FILE 和 REBIND_ROUTINE_PACKAGE。如果在 CURRENT PATH 专用寄存器中指定 SYSPROC 模式和 SYSFUN 模式，那么您可以立即使用这些函数和过程。如果您正在考虑实现用于执行管理功能的外部例程，那么最好先仔细查看 DB2 数据库系统提供的内置例程的集合。

特别是，您可能会发现 ADMIN_CMD 过程特别有用，这是因为，此过程提供了用于通过 SQL 接口执行许多常用 DB2 数据库命令的标准接口。

由于内置例程立即可供您使用，因此使您能够更快更方便地实现复杂的 SQL 查询和功能强大的数据库应用程序。

用户定义的例程

DB2 数据库系统提供例程来捕获最常用算术、字符串以及强制类型转换函数的功能。但是，DB2 数据库系统也允许您创建例程以封装您自己的逻辑。这些例程称为用户定义的例程。

可以使用例程类型所支持的实现样式来创建您自己的过程、函数和方法。通常，如果指的是过程和方法，那么不会使用前缀“用户定义的”。用户定义的函数通常又称为 UDF。

用户定义的例程创建

通过执行适合于例程类型的 CREATE 语句在数据库中创建用户定义过程、函数以及方法。这些例程创建语句包括:

- CREATE PROCEDURE
- CREATE FUNCTION
- CREATE METHOD

特定于每个 CREATE 语句的子句定义例程的特征，例如例程名称、例程参数数目和类型以及例程逻辑的详细信息。DB2 数据库系统使用这些子句所提供的信息来标识何时调用例程，并在调用时运行该例程。成功地执行用于创建例程的 CREATE 语句之后，就会在数据库中创建该例程。例程的特征是存储在用户可以查询的 DB2 目录视图中。执行 CREATE 语句以创建例程的过程又称为定义例程或注册例程。

用户定义的例程定义是存储在 SYSTOOLS 系统目录表模式中。

用户定义的例程逻辑实现

可以使用三种实现样式来指定例程的逻辑:

- 有源: 用户定义的例程可以*源自*现有内置例程的逻辑。
- SQL: 可仅使用 SQL 语句实现用户定义的例程。
- 外部: 可使用一组受支持编程语言中的一个来实现用户定义的例程。

  使用非 SQL 编程语言来创建例程时，根据代码构建的库或类是通过 EXTERNAL NAME 子句中所指定的值与例程定义相关联。调用例程时，将运行与该例程相关联的库或类。

用户定义的例程可以包括各种 SQL 语句，但不能包括所有 SQL 语句。

用户定义的例程是强类型，但例程开发者必须开发或增强类型处理和错误处理机制。

在数据库升级后，可能必须验证或更新例程实现。

通常，用户定义的例程性能良好，但不及内置例程。

用户定义的例程可以调用内置例程，以及其他以任何支持格式来实现的用户定义的例程。此灵活性允许用户在本质上能够构建可以复用的例程模块库。

通常，用户定义的例程提供一种方法来扩展 SQL 语言以及对逻辑进行模块化（多个不存在内置例程的查询或数据库应用程序将复用此逻辑）。

对内置例程与用户定义的例程进行比较

了解内置例程和用户定义的例程之间的差别有助于确定您是否确实需要构建您自己的例程或是否可以复用现有例程。如果能够确定何时复用现有例程以及何时开发您自己的例程，那么可以节省时间和工作以及确保您将例程性能发挥至极致。

内置例程和用户定义的例程在各个方面都存在差别。下表概括了这些差别:

表 1. 对内置例程与用户定义的例程进行比较

特征	内置例程	用户定义的例程
功能支持	可立即使用广泛的数值运算符、字符串操作以及管理功能。 要使用这些例程，只需从支持的接口调用这些例程。	虽然并非所有 SQL 语句都受用户定义的例程支持，但其中许多仍受支持。如果您需要扩展内置例程的功能，那么也可以在用户定义的例程中包括对内置例程的调用。用户定义的例程为例程逻辑实现提供了无限机会。 要使用这些例程，您必须先开发这些例程，然后可以从支持的接口调用这些例程。
维护	不需要维护。	外部例程要求您管理相关联的外部例程库。
升级	升级所带来的影响较小或没有。	发行版至发行版升级可能要求您验证例程。
性能	性能高于等价的用户定义的例程。	性能通常不及等价的内置例程。
稳定性	错误处理。	例程开发者必须对错误处理进行编程。

只要可能，就应该选择使用内置例程。提供这些例程是为了加快 SQL 语句构造和应用程序开发；这些例程已进行优化，可取得良好的性能。用户定义的例程可让您灵活地构建自己的例程，以解决无法使用内置例程来执行您所要实现的特定业务逻辑的问题。

确定何时使用内置例程或用户定义的例程

*内置例程*提供能够节省时间并且立即可供使用的封装功能，而*用户定义的例程*使您能够在没有任何内置例程包含您所需的全部功能时灵活定义自己的例程。

过程

要确定是使用内置例程还是使用用户定义的例程，请执行以下操作:

1. 确定要让该例程封装的功能。
2. 检查可用的内置例程的列表，以了解是否有任何例程满足您的部分或全部需求。
   - 如果存在能够满足您的部分需求但无法满足全部需求的内置例程:
     - 确定所缺少的功能是否是可以方便地在应用程序中添加的功能。如果是的话，请使用内置例程，并修改应用程序以使其涵盖所缺少的功能。如果无法方便地将缺少的功能添加到应用程序中，或者必须要在多个位置重复使用所缺少的功能，请考虑创建用户定义的例程并使其包含所缺少的功能并调用内置例程。
     - 如果您预计例程需求将有所发展，并且您必须频繁地修改例程定义，请考虑使用用户定义的例程，而不要使用内置例程。
     - 确定是否要对该例程传入/传出其他参数。如果有的话，请考虑创建用户定义的例程并使其封装对内置例程的调用。
   - 如果没有任何内置例程足够地捕获您所要封装的功能，请创建用户定义的例程。

结果

为了节省时间和人工，每当有可能的时候，请考虑使用内置例程。有时，内置例程未提供您所需的功能。对于这些情况，您必须创建用户定义的例程。在其他情况下，可以从涵盖所需额外功能的用户定义的例程中调用内置例程。

例程的功能类型

例程具有不同的功能类型。每种功能类型支持从不同的接口调用例程以实现不同的用途。例程的每种功能类型都提供了一组不同的功能和 SQL 支持。

- 过程也称为存储过程，它们用作对客户机应用程序、例程、触发器和动态复合语句的子例程扩展。通过执行对过程进行引用的 CALL 语句来调用过程。过程可以具有输入、输出和输入/输出参数，可以执行各种 SQL 语句并将多个结果集返回给调用者。
- 函数是输入数据值的集合与一组结果值之间的关系。函数使您能够扩展和定制 SQL。您从 SQL 语句的元素（例如 SELECT 列表、表达式或 FROM 子句）中调用函数。函数分为四类: 聚集函数、标量函数、行函数和表函数。
- 方法允许您访问用户定义的类型属性以及为用户定义的类型定义附加的行为。结构化类型是用户定义数据类型，它包含一个或多个指定的属性（每个属性都具有数据类型）。属性是用于描述类型实例的特性。例如，几何图形可以具有笛卡尔坐标列表之类的属性。对于结构化类型而言，方法通常作为对该结构化类型的属性执行的操作实现。对于几何形状而言，方法可以计算该形状的体积。

有关每种功能性例程类型的具体详细信息，请参阅每种例程类型的相应主题。

例程: 过程

过程也称为存储过程，它们是您通过执行 CREATE PROCEDURE 语句创建的数据库对象。过程可以封装逻辑和 SQL 语句，并可用作对客户机应用程序、例程、触发器和动态复合语句的子例程扩展。

通过执行对过程进行引用的 CALL 语句来调用过程。过程可以具有输入、输出和输入/输出参数，可以执行各种 SQL 语句并将多个结果集返回给调用者。

功能

- 允许封装构成特定子例程模块的逻辑元素和 SQL 语句
- 可以从客户机应用程序、其他例程、触发器和动态复合语句中调用 - 从任何可以执行 CALL 语句的位置调用
- 返回多个结果集
- 支持执行大量 SQL 语句，其中包括用于读取或修改单一分区数据库和多分区数据库中的表数据的 SQL 语句
- 对输入、输出以及输入/输出参数的参数支持
- 支持嵌套的过程调用和函数调用
- 支持对过程进行递归调用
- 在过程中，支持保存点和事务控制

限制

- 不能从除 CALL 语句以外的 SQL 语句中调用过程。作为替代，可以使用函数来表示那些变换列值的逻辑。

- 过程调用的输出参数值和结果集不能被另一 SQL 语句直接使用。必须使用应用程序逻辑将这些输出参数值和结果集指定给可以在后续 SQL 语句中使用的变量。
- 过程无法在调用之间保留状态。

常见用法

- 应用程序逻辑的标准化
  - 如果多个应用程序必须以类似的方式访问或修改数据库，那么过程可以提供该逻辑的单一接口。于是，该过程便可供复用。万一需要更改该接口以适应业务逻辑更改，也只需要修改这个单一过程。
- 在应用程序中将数据库操作与非数据库逻辑隔离
  - 过程使您能够方便地实现子例程，该子例程可以封装与可以在多个实例中复用的特定任务相关联的逻辑和数据库访问。例如，职员管理应用程序可以封装特定于“聘用职员”这一任务的数据库操作。这样的过程可以将职员信息插入到多个表中、根据输入参数来计算职员的每周薪水以及将每周薪水值作为输出参数返回。另一个过程可以对表中的数据执行统计分析并返回包含分析结果的结果集。
- 简化对一组 SQL 语句的特权的管理。
  - 通过允许将多个 SQL 语句封装到一个指定的数据库对象中，过程减少了数据库管理员所需管理的特权。他们只需管理调用该例程所需的特权，而不必授予执行该例程中每个 SQL 语句所需的特权。

受支持的实现

- 内置过程立即可供用户使用，用户也可以创建用户定义过程。过程支持下列用户定义的实现:
  - SQL 实现
  - 外部实现

例程: 函数

函数是输入数据值集和结果值集之间的关系。函数使您能够扩展和定制 SQL。从 SQL 语句元素（例如查询列表或 FROM 子句）中调用函数。

有四种函数:

- 聚集函数
- 标量函数
- 行函数
- 表函数

聚集函数

又称为列函数，这种函数返回一个标量值（对一组相似输入值进行求值的结果）。例如，可以通过表中的列或通过 VALUES 子句中的元组来指定相似的输入值。这组值称为自变量集。例如，下列查询通过使用 SUM 聚集函数来求出库存中或订单上螺钉的总数量:

```
SELECT SUM (qinstock + qonorder)
FROM inventory
WHERE description LIKE '%Bolt%'
```

标量函数

标量函数是一种针对每组标量参数（一个或多个）返回单一标量值的函数。标量函数示例包括 LENGTH 函数和 SUBSTR 函数。也可以创建标量函数以针对函数输入参数执行复杂的数学计算。可以在 SQL 语句的任何有效表达式（例如查询列表或 FROM 子句）中引用标量函数。下列示例显示一个引用内置 LENGTH 标量函数的查询:

```
SELECT lastname, LENGTH(lastname)
FROM employee
```

行函数 行函数是一种针对每组标量参数（一个或多个）返回单行的函数。只能将行函数用作变换函数（将结构化类型的属性映射至行中的内置数据类型值）。

表函数 表函数是针对一组参数（一个或多个）集，将表返回给对表进行引用的 SQL 语句的函数。只能在 SELECT 语句的 FROM 子句中引用表函数。由表函数所返回的表可以参与连接、分组操作、集合操作（例如 UNION）以及任何可以应用于只读视图的操作。下列示例演示一个 SQL 表函数，该表函数更新库存表，并返回对更新库存表所作查询的结果集:

```
CREATE FUNCTION updateInv(itemNo VARCHAR(20), amount INTEGER)
   RETURNS TABLE (productName VARCHAR(20),
                  quantity INTEGER)
   LANGUAGE SQL
   MODIFIES SQL DATA
   BEGIN ATOMIC

     UPDATE Inventory as I
       SET quantity = quantity + amount
         WHERE I.itemID = itemNo;

     RETURN
       SELECT I.itemName, I.quantity
         FROM Inventory as I
           WHERE I.itemID = itemNo;
   END
```

函数支持下列功能:

- 各种 DB2 品牌数据库产品都支持函数，除了别的以外，这些数据库产品还包括 DB2、DB2 z/OS® 版以及 DB2 System i® 版数据库
- 对 SQL 语句执行的适度支持
- 对输入参数和标量或聚集函数返回值的参数支持
- 有效地将函数逻辑编译至对函数进行引用的查询
- 外部函数支持在个别函数子调用（针对每一行或每个值执行）之间存储中间值

用户可随时使用内置函数，或创建用户定义的函数。可以将函数实现为 SQL 函数或外部函数。SQL 函数可以是编译型函数或内联型函数。内联型函数的性能要高于编译型函数，但是内联型函数只能执行 SQL PL 语言子集。请参阅 CREATE FUNCTION 语句，以了解更多信息。

例程: 标量函数:

标量函数是一种针对每组标量参数（一个或多个）返回单一标量值的函数。标量函数示例包括 LENGTH 函数和 SUBSTR 函数。

也可以创建标量函数以针对函数输入参数执行复杂的数学计算。可以在 SQL 语句的任何有效表达式（例如查询列表或 FROM 子句）中引用标量函数。

功能

- 内置标量函数的性能较佳。
- 内置标量函数具有强类型化特征。
- 您可以在任何支持表达式的位置使用 SQL 语句来引用标量函数。
- 逻辑将作为引用该标量函数的 SQL 语句的组成部分在服务器上执行。
- 标量 UDF 的输出可以由引用该函数的语句直接使用。
- 在谓词中使用标量 UDF 有助于提高整体查询性能。将标量函数应用于服务器上的一组候选行时，该函数可以充当过滤器，从而对必须返回到客户机的行数进行限制。
- 对于用户定义的外部标量函数，可以使用暂存区在函数迭代调用之间维护状态。

限制

- 根据设计，它们只返回单一标量值。
- 在标量函数中，不支持事务管理。无法在标量函数体中执行落实和回滚。
- 无法从标量函数返回结果集。
- 在单一分区数据库中，用户定义的外部标量 UDF 可以包含 SQL 语句。这些语句可以从表中读取数据，但无法修改表中的数据。
- 在多分区数据库环境中，用户定义的标量 UDF 不能包含 SQL 语句。
- 请参阅：有关标量函数的限制。

常见用法

- 在 SQL 语句中处理字符串。
- 在 SQL 语句中执行基本算术运算。
- 可以创建用户定义标量函数以扩展内置标量函数的现有集合。例如，通过将现有的内置标量函数与其他逻辑配合使用，可以创建复杂的数学函数。

受支持的实现

- 有源实现
- 外部实现
  - 请参阅以下主题：第 19 页的『对支持用于开发外部例程的 API 和编程语言的比较』。

例程：行函数：

行函数是只能与用户定义的结构化类型配合使用的函数，对于每个由一个或多个标量参数组成的集合，此函数返回单一行。

只能将行函数用作变换函数（将结构化类型的属性映射至行中的内置数据类型值）。行函数不能以独立方式使用或者在抽象数据类型的上下文外部的 SQL 语句中使用。

功能

- 允许您将结构化类型属性映射到由内置数据类型值组成的行。

限制

- 不能以独立方式使用或者在用户定义的结构化类型的上下文外部的 SQL 语句中使用。

- 请参阅“有关行函数的限制”主题。

常见用法

使结构化类型属性在查询或操作中可访问。例如，考虑名为“manager”的用户定义的结构化数据类型，此类型扩展另一个结构化类型 person 并且同时包含人员属性以及经理所特有的属性。如果您希望在查询中引用这些值，那么可创建一个行函数，以将属性值转换为一行可以引用的值。

受支持的实现

- SQL 实现

例程：表函数：

表函数是用于由一个或多个参数组成的集合的函数，它向引用它的 SQL 语句返回一个表。

只能在 SELECT 语句的 FROM 子句中引用表函数。表函数所返回的表可以参与连接、分组操作、UNION 之类的集合操作以及任何可以应用于只读视图的操作。

功能

- 返回一组数据值以进行处理。
- 可以作为 SQL 查询的组成部分被引用。
- 可以进行操作系统调用、从文件读取数据甚至通过网络访问单一分区数据库中的数据。
- 表函数调用的结果可以由引用该表函数的 SQL 语句直接访问。
- SQL 表函数可以封装那些修改 SQL 表数据的 SQL 语句。外部表函数不能封装 SQL 语句。
- 对于单一表函数引用，可以多次以迭代方式调用表函数并使用暂存区在这些调用之间维护状态。

限制

- 在用户定义的表函数中，不支持事务管理。不能在表 UDF 中执行落实和回滚。
- 无法从表函数返回结果集。
- 并非为单一调用而设计。
- 只能在查询的 FROM 子句中引用表函数。
- 用户定义的外部表函数可以读取 SQL 数据，但不能修改 SQL 数据。作为替代，可以使用 SQL 表函数来包含那些修改 SQL 数据的 SQL 语句。
- 请参阅“有关表函数的限制”主题。

常见用法

- 封装复杂但常用的子查询。
- 提供非关系数据的表接口。例如，用户定义的外部表函数可以读取电子表格并生成值表，您可以将该值表直接插入到表中，也可以在查询中立即直接访问该值表。

受支持的实现

- SQL 实现
- 外部实现

例程: 方法

提供一些方法以允许您访问结构化类型属性，以及定义结构化类型的其他行为。

结构化类型是用户定义数据类型，它包含一个或多个指定的属性（每个属性都具有数据类型）。属性是用于描述类型实例的特性。例如，几何图形可以具有笛卡尔坐标列表之类的属性。

通常，为结构化类型实现方法以表示对该结构化类型的属性执行的操作。对于几何形状而言，方法可以计算该形状的体积。方法共享标量函数的所有功能。

功能

- 能够访问结构化类型属性
- 能够设置结构化类型属性
- 能够创建对结构化类型属性执行的操作并返回函数值
- 对主体类型的动态类型敏感

限制

- 只能返回标量值
- 只能与结构化类型配合使用
- 不能对类型表调用

常见用法

- 创建对结构化类型执行的操作
- 封装结构化类型

受支持的实现

没有内置方法。用户可以为现有的用户定义的结构化类型创建用户定义的方法。可以使用下列其中一种实现来实现方法:

- 第 18 页的『SQL 例程实现』
- 第 18 页的『外部例程实现』: C、C++、Java、C#（使用 OLE API）、Visual Basic（使用 OLE API）

SQL 方法易于实现，但通常与结构化类型一起设计。外部方法提供了对灵活逻辑实现的更大支持，并允许用户使用其喜爱的编程语言来开发方法逻辑。

例程功能类型比较

了解过程、函数以及方法之间的差别有助于确定构建您自己的例程时要实现的功能类型，并且有助于确定在何处以及如何引用现有例程。这可以节省您的时间和工作，以及确保您可以将例程的功能和性能发挥至极致。

过程、函数和方法在各个方面都存在差别。下表描述了这些差别:

表 2. 例程功能类型比较

特征	过程	函数	方法
唯一功能特征和实用应用程序	• 支持对逻辑和 SQL 语句进行封装。 • 充当客户机应用程序、例程、触发器以及动态复合语句的子例程扩展。 • 通过执行对过程进行引用的 CALL 语句来调用过程。 • 支持嵌套的过程调用 • 支持递归过程调用 • 对输入、输出以及输入/输出参数的参数支持 • 对 SQL 语句执行的广泛支持 • 可以返回一个或多个结果集 • 保存点和事务控制	• 支持对逻辑和 SQL 语句进行封装。 • 函数是输入数据值集和结果值集之间的关系。 • 函数使您能够扩展和定制 SQL。 • 从 SQL 语句元素（例如查询列表或 FROM 子句）中调用函数。 • 对 SQL 语句执行的适度支持。 • 对输入参数和标量或聚集函数返回值的参数支持。 • 外部函数支持使用暂存区在个别函数子调用（针对每一行或每个值执行）之间存储中间值。 • 有效地将函数逻辑编译至对函数进行引用的查询。	• 支持对逻辑和 SQL 语句进行封装。 • 提供一些方法以允许您访问结构化类型属性，以及定义结构化类型的其他行为。 • 能够访问结构化类型属性。 • 能够设置结构化类型属性。 • 能够在结构化类型属性上创建操作，然后返回函数值。
例程的功能子类型	• 不适用	• 标量函数 • 聚集函数 • 行函数 • 表函数	• 不适用
调用接口	• 通过执行对过程进行引用的 CALL 语句来完成调用。 • 只要支持 CALL 语句，就支持过程调用。	• 除了在别的位置之外，还可以在 SQL 语句的列查询列表、表达式或者在 SELECT 语句的 FROM 子句中完成调用。	• 在引用与方法相关联的结构化类型的 SQL 语句中完成调用。
具有此类型的任何内置例程吗？	• 有，很多。 • 请参阅“SQL 引用”以获取内置过程的列表。	• 有，很多。 • 请参阅“SQL 引用”以获取内置函数的列表。	• 否
支持的用户定义的例程实现	• SQL • 外部 – C/C++（带有嵌入式 SQL 或 CLI API 调用） – COBOL – Java (JDBC) – Java (SQLJ) – .NET CLR – OLE: Visual Basic、Visual C++	• SQL • 外部 – C/C++ – Java (JDBC) – Java (SQLJ) – .NET CLR – OLE DB: Visual Basic、Visual C++（仅限于表函数）	• SQL • 外部 – C – C++

表 2. 例程功能类型比较 (续)

特征	过程	函数	方法
嵌套调用支持	• 是	• 否。但是，会针对输入集中的每个值重复调用函数，然后使用暂存区来存储中间值。	• 否
性能	• 如果例程逻辑高效且采用了最佳实践，那么性能良好。	• 如果例程逻辑高效且采用了最佳实践，那么性能良好。 • 如果逻辑只查询数据，而不修改数据，那么性能高于逻辑上等价的过程。	• 性能良好
可移植性	• 高度可移植 • 如果使用 SQL 实现，那么可移植性特别强。 • 各种编程语言都支持 32 位和 64 位外部例程	• 高度可移植 • 如果使用 SQL 实现，那么可移植性特别强。 • 各种编程语言都支持 32 位和 64 位外部例程	• 高度可移植
互操作性	• 过程可以调用其他过程，并且可以包含对函数（其 SQL 访问级别小于或等于过程的 SQL 访问级别）进行调用的 SQL 语句。	• 函数可以包含对其他函数进行调用的 SQL 语句，并且可以调用过程（其 SQL 访问级别小于或等于函数的 SQL 访问级别）。	• 方法可以调用函数（其 SQL 访问级别小于或等于方法的 SQL 访问级别）。 • 方法不能调用过程或其他方法
限制		• 表函数只能返回必须在 SELECT 语句的 FROM 子句中引用的单个表引用。输出。	

通常，例程的功能特征和应用确定应该使用的例程类型。但是，性能以及支持的例程实现也在确定应该使用的例程类型时起到重要作用。

确定要使用的例程功能类型

过程、函数和方法提供了不同的功能例程和功能支持。确定要使用或实现的例程类型将确定您可以从什么位置以及如何引用和调用例程功能，将影响您可以使用的例程实现，并且可以影响例程能够包含的功能类型。

开始之前

在开始实现例程之前，确定哪种例程类型最适合于您的需要有助于节省时间以及减少将来可能的挫折。

阅读关于例程的功能类型的内容，以了解其特征。

过程

要确定是使用过程、函数还是方法，请执行下列操作:

1. 确定要让例程封装的功能、要从中调用例程的接口以及要使用的例程实现。
   - 请参阅以下主题:

– 第 14 页的『例程功能类型比较』

以确定哪些功能例程类型支持这些需求。

2. 确定要包括在例程中的 SQL 语句。
   - 请参阅以下主题:
     – 第 30 页的『可以在例程和触发器中执行的 SQL 语句』
   - 确定哪些功能例程支持执行所需的 SQL 语句。
3. 如果该例程将只包含一个或多个查询，那么请考虑使用 SQL 函数。在此情况下，SQL 函数的性能较佳，这是因为，经过编译的函数已将引用它们的 SQL 语句直接插入到其中，而过程却是单独进行编译和调用的。
4. 确定将来是否需要扩展该例程的功能以包括另一例程类型的功能（例如，与函数相比，过程支持更多 SQL 语句，并且通常支持更多 SQL 功能）。为了避免将来必须将函数重新编写为过程，现在请考虑实现过程。

结果

通常，功能需求和 SQL 需求是您选择要实现的例程的功能类型的动机。但是，在某些情况下，可以创建具有不同功能类型但逻辑等同的例程。例如，可以将大多数返回单一结果集的基本过程重新编写为表函数。另外，还可以方便地将只有一个输出参数的基本过程重新编写为标量函数。

下一步做什么

在确定所要使用的例程的功能类型之后，您可能希望了解更多有关例程实现的信息或者希望确定要使用的例程实现。

例程的实现

您可以通过多种方式来实现例程。实际上，例程实现是例程的底层形式，它包含该例程被调用时运行的逻辑。了解不同的受支持例程实现可以帮助您理解例程的工作方式，并可以帮助您确定实现用户定义的例程时应该选择的例程实现。

可用的例程实现包括:

- 第 18 页的『内置例程实现』
- 第 18 页的『有源例程实现』
- 第 18 页的『SQL 例程实现』
- 第 18 页的『外部例程实现』

第 6 页的『内置例程』可以作为内置例程、SQL 例程或外部例程实现。但是，它们的实现在本质上对用户不可视，并且通常与用户不大相关。

第 6 页的『用户定义的例程』可以作为有源例程、SQL 例程或外部例程实现。

每种实现的特征各不相同，并且会导致或多或少的功能支持。在决定采用特定的实现之前，最好通过阅读有关每种实现的内容以及以下主题来查看与每种实现相关联的受支持功能以及限制:

- 第 23 页的『比较例程实现』

良好地理解例程实现可以帮助您作出良好的实现决策，并可以帮助您调试现有的例程以及对其进行故障诊断。

内置例程实现

内置例程已内置到 DB2 数据库管理器的代码中。这些例程包含数据库代码所固有的逻辑，因此具有强类型化和性能优异特征。

这些例程包含在 SYSIBM 模式中。内置标量函数和聚集函数的一些示例是:

- 内置标量函数: +、-、*、/、substr、concat、length、char、decimal 和 days
- 内置聚集函数: avg、count、min、max、stdev、sum 和 variance

内置函数用于执行大多数常用的强制类型转换、字符串处理和算术功能。您可以在 SQL 语句中立即使用这些函数。要获取可用的内置函数的列表，请参阅《*SQL 引用*》。

有源例程实现

使用有源例程实现来实现的例程是一个复制其他函数（称为例程的源函数）的语义的例程。

当前，只能将标量和聚集函数用作有源函数。在允许单值类型有选择地继承其源类型的语义方面，有源函数特别有用。有源函数本质上是函数的特殊形式 SQL 实现。

SQL 例程实现

SQL 例程实现只包含 SQL 语句。

SQL 例程实现的特征是对例程逻辑进行定义的 SQL 语句都包括在用来在数据库中创建例程的 CREATE 语句中。因为 SQL 例程的语法简单，所以 SQL 例程可以简单快捷地实现，并且由于与 DB2 数据库系统关系密切，因此性能良好。

SQL 过程语言 (SQL PL) 是基本 SQL 的语言扩展，基本 SQL 包含可用来以 SQL 实现编程逻辑的语句和语言元素。SQL PL 包含一组语句，用来声明变量和条件处理程序（DECLARE 语句）以将值指定给变量（赋值语句），以及用来实现过程逻辑（控制语句），例如 IF、WHILE、FOR、GOTO、LOOP 和 SIGNAL 等。可以使用 SQL 和 SQL PL 来创建 SQL 过程、函数、触发器以及复合 SQL 语句。SQL 过程和函数以及 SQL 全局变量、用户定义的条件和数据类型可以一起归类到模块。

外部例程实现

外部例程实现是指，例程逻辑由驻留在数据库外部的编程语言代码定义。与其他例程实现相同，您通过执行 CREATE 语句在数据库中创建具有外部实现的例程。

例程逻辑存储在经过编译的库中，该库驻留在数据库服务器上的特殊目录路径中。例程名称与外部代码应用程序之间的关联由 CREATE 语句中指定的 EXTERNAL 子句推断。

可使用任何受支持外部例程编程语言来编写外部例程。请参阅第 19 页的『用于开发外部例程的受支持 API 和编程语言』。

外部例程实现可能比 SQL 例程实现略为复杂。但是，其功能极为强大，这是因为它们允许您充分利用所选实现编程语言的功能和性能。外部函数还具有能够访问和处理驻留在数据库外部的实体（例如网络或文件系统）这一优点。对于只需要与 DB2 数据库进行少量交互但必须包含大量逻辑或非常复杂的逻辑的例程而言，外部例程实现是良好的选择。

例如，外部例程适合用于实现对内置数据类型执行操作以及提高其利用率的新函数，例如对 VARCHAR 数据类型执行操作的新字符串函数或者对 DOUBLE 数据类型执行操作的复杂数学函数。外部例程实现还适合于实现可能涉及外部操作（例如发送电子邮件）的逻辑。

如果您已熟悉使用其中一种受支持的外部例程编程语言进行编程，并且需要封装更着重于编程逻辑而非数据访问的逻辑，那么在了解创建具有外部实现的例程所涉及的步骤之后，您很快就会发现它们的巨大潜力。

用于开发外部例程的受支持 API 和编程语言: 您可以使用下列 API 及相关联的编程语言来开发 DB2 外部例程（过程和函数）：

- ADO.NET
  - .NET 公共语言运行时编程语言
- CLI
- 嵌入式 SQL
  - C
  - C++
  - COBOL（只支持过程）
- JDBC
  - Java
- OLE
  - Visual Basic
  - Visual C++
  - 任何其他支持此 API 的编程语言。
- OLE DB（只支持表函数）
  - 任何支持此 API 的编程语言。
- SQLJ
  - Java

对支持用于开发外部例程的 API 和编程语言的比较:

在开始实现外部例程之前，必须考虑各种支持的外部例程应用程序编程接口 (API) 和编程语言的特征与限制。这将确保您从开始就选择正确的实现，以及您所需的例程功能是可用的。

表 3. 外部例程 API 和编程语言比较

API 和编程语言	功能支持	性能	安全性	可伸缩性	限制
SQL（包括 SQL PL）	• SQL 是高级语言，易于学习和使用，使实现能快速进行。 • SQL 过程语言 (SQL PL) 元素允许将控制流逻辑用于 SQL 操作和查询。	• 非常好。 • SQL 例程的性能高于 Java 例程。 • SQL 例程的性能相当于使用 NOT FENCED 子句来创建的 C 和 C++ 外部例程。	• 非常安全。 • SQL 过程始终与数据库管理器在同一内存中运行。这对应于缺省情况下使用关键字 NOT FENCED 来创建的例程。	• 高度可伸缩。	• 无法访问数据库服务器文件系统。 • 无法调用位于数据库外部的应用程序。
嵌入式 SQL（包括 C 和 C++）	• 低级但功能强大的编程语言。	• 非常好。 • C 和 C++ 例程的性能高于 Java 例程。 • 使用 NOT FENCED 子句创建的 C 和 C++ 例程的性能相当于 SQL 例程。	• C 和 C++ 例程容易发生编程错误。 • 程序员必须精通 C，才能避免犯一些常见的内存和指针操作错误，此类错误会使例程实现更冗长、更耗时。 • 应该使用 FENCED 子句和 NOT THREADSAFE 子句来创建 C 和 C++ 例程，以避免在运行时例程发生异常的情况下，损坏数据库管理器。这些是缺省子句。使用这些子句会对性能造成一定的负面影响，但是可确保安全执行。请参阅例程安全性。	• 使用 FENCED 和 NOT THREADSAFE 子句来创建 C 和 C++ 例程时，会降低可伸缩性。在数据库管理器进程之外的单独 **db2fmp** 进程中运行这些例程。每个并发执行的例程都需要一个 **db2fmp** 进程。	• 多种支持的参数传递样式会造成混淆。用户应该尽可能地使用参数样式 SQL。

表 3. 外部例程 API 和编程语言比较 (续)

API 和编程语言	功能支持	性能	安全性	可伸缩性	限制
嵌入式 SQL（COBOL）	• 高级编程语言适合于开发业务应用程序（通常面向文件）。 • 过去广泛用于生产业务应用程序，虽然其普及面现正在缩小。 • COBOL 不包含指针支持，是线性迭代编程语言。	• COBOL 例程的性能低于使用任何其他外部例程实现选项来创建的例程。	• 目前未提供任何信息。	• 目前未提供任何信息。	• 您可以在 64 位 DB2 实例中创建和调用 32 位 COBOL 过程实例，但是这些例程的性能低于 64 位 DB2 实例中 64 位 COBOL 过程。
JDBC（Java）和 SQLJ（Java）	• 高级面向对象程序设计语言，适合于开发独立应用程序、applet 以及 servlet。 • Java 对象和数据类型可方便建立数据库连接、执行 SQL 语句以及操作数据。	• Java 例程的性能低于 C 和 C++ 例程或 SQL 例程。	• Java 例程较之 C 和 C++ 例程安全，因为是由 Java 虚拟机（JVM）处理对危险操作的控制。这将提高可靠性，让一个 Java 例程的代码很难损害正在同一进程中运行的另一个例程。	• 良好的可伸缩性 • 使用 FENCED THREADSAFE 子句（缺省子句）创建的 Java 例程具有良好的可伸缩性。所有 FENCED Java 例程将共享几个 JVM。如果特定 db2fmp 进程的 Java 堆接近耗竭，那么系统上可能正在使用多个 JVM。	• 只有不允许从 Java 例程执行 Java 本机接口（JNI）调用，才能避免潜在的危险操作。

表 3. 外部例程 API 和编程语言比较 (续)

API 和编程语言	功能支持	性能	安全性	可伸缩性	限制
.NET 公共语言运行时支持的语言（包括 C#、Visual Basic 以及其他）	• Microsoft .NET 受管代码模型的一部分。 • 会将源代码编译为可由 Microsoft .NET Framework 公共语言运行时进行解释的中间语言（IL）字节码。 • 可以从子组合件（从不同的 .NET 编程语言源代码进行编译）构建 CLR 组合件，这允许用户复用和集成使用各种语言编写的代码模块。	• 只能使用 FENCED NOT THREADSAFE 子句来创建 CLR 例程，才能将运行时中断数据库管理器的可能性降至最小。这会对性能造成一定的负面影响 • 使用缺省子句值可将运行时中断数据库管理器的可能性降至最小；但是，因为 CLR 例程必须以 FENCED 方式运行，所以它们的性能可能稍小于可以指定为以 NOT FENCED 方式来运行的外部例程。	• 只能使用 FENCED NOT THREADSAFE 子句来创建 CLR 例程。如此才能让 CLR 例程安全运行，因为它们将在数据库管理器外部的单独 **db2fmp** 进程中运行。	• 未提供任何信息。	• 请参阅“.NET CLR 例程限制”主题。
• OLE	• 可以使用 Visual C++、Visual Basic 以及其他受 OLE 支持的语言来实现 OLE 例程。	• OLE 自动化例程的速度取决于用来实现这些例程的语言。通常，它们的速度要慢于非 OLE C/C++ 例程。 • OLE 例程只能以 FENCED NOT THREADSAFE 方式运行，因此 OLE 自动化例程没有良好的可伸缩性。	• 未提供任何信息。	• 未提供任何信息。	• 未提供任何信息。

表 3. 外部例程 API 和编程语言比较 (续)

API 和编程语言	功能支持	性能	安全性	可伸缩性	限制
• OLE DB	• 可以使用 OLE DB 来创建用户定义的表函数。 • OLE DB 函数连接至外部 OLE DB 数据源。	• OLE DB 函数的性能取决于 OLE DB 提供者，但是，通常 OLE DB 函数的性能高于逻辑上等价的 Java 函数，但低于逻辑上等价的 C、C++ 或 SQL 函数。但是，查询（在其中调用了函数）中的某些谓词可能在 OLE DB 提供者处进行求值，因此减少了 DB2 数据库系统必须处理的行数，这常常可以改进性能。	• 未提供任何信息。	• 未提供任何信息。	• OLE DB 只能用来创建用户定义的表函数。

比较例程实现

了解支持例程实现之间的差别有助于确定构建您自己的例程时要使用的例程实现。这可以节省您的时间和工作，以及确保您可以将例程的功能和性能发挥至极致。

内置、有源、SQL 以及外部例程实现在各个方面都存在差别。下表描述了这些差别:

表 4. 比较例程实现

特征	内置	有源	SQL	外部
功能和使用	• 性能非常高，因为它们的逻辑是数据库管理器代码的本机逻辑。 • 许多常用强制类型转换、字符串操作以及算术内置函数都位于 SYSIBM 模式中。	• 用于提供内置函数功能的基本扩展。 • SQL 和 SQL PL 提供高级编程语言支持，使例程逻辑实现简单快捷。	• 用于通过可以执行 SQL 语句的较复杂函数来扩展内置函数集。	• 开发者可以使用他们选择的支持编程语言来编写逻辑。 • 可以实现复杂逻辑。 • 直接支持外部操作（在数据库外部起作用的操作）。这可以包括读/写服务器文件系统、调用服务器上的应用程序或脚本以及发出 SQL、有源或内置实现不支持的 SQL 语句。
是否将实现构建到数据库管理器代码？	• 是	• 否	• 否	• 否

表 4. 比较例程实现 (续)

特征	内置	有源	SQL	外部
可以具有此实现的支持功能例程类型	• 不适用	• 函数 – 标量函数 – 聚集函数	• 过程 • 函数 • 方法	• 过程 • 函数 • 方法
支持的 SQL 语句	• 不适用	• 不适用	• 可以在例程中执行大多数 SQL 语句，其中包括所有 SQL PL 语句。 • 请参阅“可以在例程中执行的 SQL 语句”主题。	• 可以在例程中执行许多 SQL 语句，其中包括 SQL PL 语句子集。 • 请参阅“可以在例程中执行的 SQL 语句”主题。
性能	• 非常快	• 通常，与内置函数速度相当。	• 性能非常高（如果有效地编写 SQL、强调数据库操作多于编程逻辑以及采用 SQL 例程最佳实践）。 • 请参阅“SQL 例程最佳实践”主题。	• 性能非常高（如果有效地编写编程逻辑以及采用外部例程最佳实践）。 • 请参阅“外部例程最佳实践”主题。
可移植性	• 不适用	• 可以轻松地在其他 DB2 数据库中删除和重新创建有源函数。	• 可以在其他数据库中轻松地删除和重新创建 SQL 函数。	• 可以在其他数据库中删除和重新创建外部函数，但是必须小心确保环境是兼容的，且必需的支持软件是可用的。 • 请参阅“部署外部例程”主题。
互操作性	• 不适用	• 只要可以引用内置函数，就可以引用有源函数。有源函数无法调用其他函数。	• 可以在 SQL 语句的许多部件中引用 SQL 例程。SQL 例程可以调用其他 SQL 和外部例程，前提是外部例程的 SQL 访问级别等于或小于 SQL 例程的 SQL 访问级别。	• 外部例程可以调用外部例程和其他 SQL 例程，前提是 SQL 例程的 SQL 访问级别等于或小于外部例程的 SQL 访问级别。

通常，例程的功能特征和应用确定应该使用的例程类型。但是，性能以及支持的例程实现也在确定应该使用的例程类型时起到重要作用。

确定要使用的例程实现

选择使用或创建具有内置、有源、SQL 还是外部例程实现的例程将影响该例程能够提供的功能、该例程的性能以及发生要求进行调试的运行时问题的可能性。

关于此任务

每当有可能的时候，如果存在能够提供您所需支持的现有内置例程，请使用该例程。您应该尽可能使用现有的内置例程。如果所需的功能与现有内置函数的功能非常类似，请考虑创建一个扩展该内置函数的有源函数。

如果您必须创建例程，请使用以下过程。在过分深入地进行例程设计之前确定所要使用的例程实现至关重要。

过程

要在创建例程时确定是使用有源、SQL 还是外部例程实现，请执行以下操作:

1. 确定是是要创建过程、函数还是方法。在开发例程时，这始终是您执行的第一步。并且，确定该例程类型所支持的实现。请参阅:
   - 第 14 页的『例程功能类型比较』
2. 确定要包括在例程中的 SQL 语句。要在例程中执行的 SQL 语句集合可能会限制您对例程实现的选择。请参阅:
   - 第 36 页的『确定可以在例程中执行的 SQL 语句』
3. 确定例程逻辑是现在还是将来必须访问驻留在数据库外部的数据、文件或应用程序。数据、文件或应用程序可能驻留在数据库服务器的文件系统中，也可能驻留在可用的网络中。
   - 如果例程逻辑必须访问数据库外部的实体，那么必须使用外部例程实现。
4. 确定要包括在例程中的查询数目与过程流逻辑量的对比。
   - 如果例程逻辑主要包含过程流逻辑，而包含的查询非常少，请创建外部例程。
   - 如果例程逻辑包含许多查询和少量过程流逻辑，请创建 SQL 例程。

例程用法

可以使用例程来解决数据库架构设计师、数据库管理员以及应用程序开发者等所面临的许多常见问题。例程可以帮助改进应用程序的结构、维护以及性能。

以下列表中提供了您可以使用例程的一些方案示例:

- 使用例程来管理数据库
- 使用用户定义的函数来扩展 SQL 函数支持
- 使用例程和其他 SQL 功能来审计数据更改

使用内置例程来管理数据库

在引入可明确执行管理功能的内置例程后，通过应用程序来管理数据库是可行的且变得更容易。

关于此任务

从 V8.1 开始，DB2 在 SYSPROC、SYSFUN 和 SYSTOOLS 模式中提供一组内置过程和函数，可随时使用这些模式来执行管理任务，其中包括通过 SQL 接口执行 DB2 命令、修改配置参数、程序包管理以及与快照相关的任务等。如果您要求应用程序执行

管理任务，需要通过 SQL 接口来访问管理任务的结果（以便过滤、排序、修改或复用另一个查询中的结果）以及不想创建自己的例程来执行此操作，那么可以选择使用内置管理例程。

从 DB2 for Linux, UNIX, and Windows V9.1 开始，提供了名为 ADMIN_CMD 的新内置管理例程。它与许多其他内置例程一起提供综合管理支持。

ADMIN_CMD，用于通过 SQL 接口来调用 DB2 命令

从 V9.1 开始，提供称为 ADMIN_CMD 的新内置管理例程，可让您通过 SQL 接口执行 DB2 命令。本质上，此例程允许您将 DB2 命令（作为自变量）以及相应的标志和值作为字符串参数来传入。例程会执行包含 DB2 命令的字符串，然后以表格或标量格式返回结果，这些结果可用作更大查询或操作的一部分。此功能使编写管理性数据库应用程序不曾如此容易。

内置管理例程

其他内置例程示例包括：
SNAPSHOT_TABLE、HEALTH_DB_HI、SNAPSHOT_FILEW 和 REBIND_ROUTINE_PACKAGE。可以从 CLP 或从数据库应用程序中支持调用指定例程的任何位置，使用这些及许多其他内置例程。

只要将 SYSPROC 模式名和 SYSFUN 模式名包括在 CURRENT PATH 值中（缺省情况下，这些模式名是包括在该值中），就可以使用 ADMIN_CMD 例程以及其他内置例程。

要获取有关如何使用内置例程的示例，请参阅特定于内置例程的参考文档。

使用用户定义的函数来扩展 SQL 函数支持

关于此任务

如果没有任何内置函数封装了您所需的逻辑，那么您可以创建自己的用户定义的函数。用户定义的函数是一种很好的扩展基本 SQL 函数集的方法。无论您或一组用户是需要一个函数来实现复杂的数学公式、特定的字符串处理还是对值执行某种语义变换，都可以方便地创建高性能的 SQL 函数来完成此任务，并且，您可以像引用任何现有内置 SQL 函数那样引用该函数。

例如，假定用户需要一个函数将一种货币单位的值转换为另一货币单位的值。内置例程集中未提供这样的函数。但是，您可以将此函数创建为用户定义的 SQL 标量函数。一旦创建此函数之后，就可以在 SQL 语句中任何支持标量函数的位置引用此函数。

另一个用户可能需要更复杂的函数，即，每当对表中特定的列进行更改时，该函数发送一封电子邮件。内置例程集中未提供这样的函数。但是，您可以使用 C 编程语言实现将此函数创建为用户定义的外部过程。一旦创建此过程，就可以在任何支持过程的位置引用此过程，包括从触发器中引用此过程。

这些示例说明您可以方便地通过创建用户定义的例程来扩展 SQL 语言。

使用 SQL 表函数进行审计

对监视数据库用户所作的表数据访问和表数据修改感兴趣的数据库管理员，可以通过创建和使用可修改 SQL 数据的 SQL 表函数来审计表上的事务。

开始之前

任何包括用于执行商业任务（例如，更新职员的个人信息）的 SQL 语句的表函数还可以另外包括 SQL 语句，这些 SQL 语句会在单独的表中记录有关调用该函数的用户所作的表访问或修改的详细信息。甚至可以编写 SQL 表函数，以便它返回表函数主体所访问或修改的表行结果集。可以将返回的行结果集插入和存储至单独的表，作为对表所作更改的历史记录。

有关创建和注册 SQL 表函数时所需的特权列表，请参阅下列语句:

- CREATE FUNCTION（SQL 标量、表或行）语句

SQL 表函数的定义者也必须有权运行 SQL 表函数主体所封装的 SQL 语句。请参阅每个封装的 SQL 语句所需的特权列表。要将表的 INSERT、UPDATE 和 DELETE 特权授予给用户，请参阅下列语句:

- GRANT（表、视图或昵称特权）语句

SQL 表函数所访问的表必须存在于 SQL 表函数调用之前。

示例

示例 1: 使用 SQL 表函数来审计表数据访问

此函数访问输入自变量 deptno 所指定的部门中所有职员的薪水数据。它也会在审计表 audit_table 中记录调用该函数的用户标识、从中读取的表的名称、所访问信息的描述以及当前时间。请注意，该表函数是使用关键字 MODIFIES SQL DATA 创建，因为它包含会修改 SQL 数据的 INSERT 语句。

```
CREATE FUNCTION sal_by_dept (deptno CHAR(3))
  RETURNS TABLE (lastname VARCHAR(10),
                 firstname VARCHAR(10),
                 salary INTEGER)
  LANGUAGE SQL
  MODIFIES SQL DATA
  NO EXTERNAL ACTION
  NOT DETERMINISTIC
  BEGIN ATOMIC
    INSERT INTO audit_table(user, table, action, time)
      VALUES (USER,
              'EMPLOYEE',
              'Read employee salaries in department: ' || deptno,
              CURRENT_TIMESTAMP);
    RETURN
      SELECT lastname, firstname, salary
        FROM employee as E
          WHERE E.dept = deptno;
  END
```

示例 2: 使用 SQL 表函数审计表数据更新

此函数会更新 updEmpNum 所指定职员的薪水（更新金额由 amount 指定），并且也会在审计表 audit_table 中记录调用例程的用户、所修改表的名称以及用户所作修改的类型。在 FROM 子句中引用数据更改语句（此处是 UPDATE 语句）的 SELECT 语句用来返回所更新的行值。请注意，该表函数是使用关键字 MODIFIES SQL DATA 创建，因为它包含 INSERT 语句以及对数据更改语句 UPDATE 进行引用的 SELECT 语句。

```
CREATE FUNCTION update_salary(updEmpNum CHAR(4), amount INTEGER)
  RETURNS TABLE (emp_lastname VARCHAR(10),
                 emp_firstname VARCHAR(10),
                 newSalary INTEGER)
```

```
  LANGUAGE SQL
  MODIFIES SQL DATA
  NO EXTERNAL ACTION
  NOT DETERMINISTIC
  BEGIN ATOMIC
    INSERT INTO audit_table(user, table, action, time)
    VALUES (USER,
            'EMPLOYEE',
            'Update emp salary. Values: '
               || updEmpNum || ' ' || char(amount),
            CURRENT_TIMESTAMP);
    RETURN
      SELECT lastname, firstname, salary
        FROM FINAL TABLE(UPDATE employee
                         SET salary = salary + amount
                         WHERE employee.empnum = updEmpNum);
  END
```

示例 3: 调用用于审计事务的 SQL 表函数

下列显示用户如何调用例程以更新职员的薪水（更新金额为 500 日元）：

```
SELECT emp_lastname, emp_firstname, newsalary
  FROM TABLE(update_salary(CHAR('1136'), 500)) AS T
```

所返回结果集包含职员的姓氏、名字以及更新后的薪水。此函数的调用者将不知道所作的审计记录。

```
EMP_LASTNAME EMP_FIRSTNAME NEWSALARY
------------ ------------- -----------
JONES        GWYNETH             90500
```

审计表会包含如下所示的新记录:

```
USER     TABLE      ACTION                              TIME
-------- ---------- ----------------------------------  --------------------------
MBROOKS  EMPLOYEE   Update emp salary. Values: 1136 500  2003-07-24-21.01.38.459255
```

示例 4: 检索 SQL 表函数主体中所修改的行

此函数会更新职员编号 EMPNUM 所指定职员的薪水（更新金额由 amount 指定），然后将所修改行的初始值返回给调用者。本示例利用在 FROM 子句中引用数据更改语句的 SELECT 语句。在此语句的 FROM 子句中指定 OLD TABLE 会标记下列操作: 从作为 UPDATE 语句目标的表 employee 返回初始行数据。使用 FINAL TABLE 取代 OLD TABLE 会标记下列操作: 返回更新表 employee 后的行值。

```
CREATE FUNCTION update_salary (updEmpNum CHAR(4), amount DOUBLE)
  RETURNS TABLE (empnum CHAR(4),
                 emp_lastname  VARCHAR(10),
                 emp_firstname VARCHAR(10),
                 dept CHAR(4),
                 newsalary integer)
  LANGUAGE SQL
  MODIFIES SQL DATA
  NO EXTERNAL ACTION
  DETERMINISTIC
  BEGIN ATOMIC
    RETURN
      SELECT empnum, lastname, firstname, dept, salary
        FROM OLD TABLE(UPDATE employee
                               SET salary = salary + amount
                                 WHERE employee.empnum = updEmpNum);
  END
```

用于开发例程的工具

可以使用各种开发环境和工具来开发过程和函数。

在这些工具中，有些是随 DB2 Database for Linux, UNIX, and Windows 提供，而有些是常用集成开发环境中的集成组件。可以使用图形和非图形界面与工具来开发过程和函数。

下列图形用户界面 (GUI) 工具随 DB2 数据库系统提供，可用于在 DB2 数据库服务器中开发例程:

- IBM® Data Studio

下列命令行界面随 DB2 数据库系统提供，可用于在 DB2 数据库服务器中开发例程:

- DB2 命令行处理器 (DB2 CLP)

若干 IBM 软件产品提供用于在 DB2 数据库服务器中开发例程的图形工具，包括但不限于:

- IBM 分布式统一调试器
- IBM Optim™ Development Studio
- IBM Rational® Application Developer
- IBM Rational Web Developer
- IBM WebSphere® Studio

若干开放式源代码软件产品提供用于在 DB2 数据库服务器中开发例程的图形工具，包括但不限于:

- 用于 Eclipse 框架的 DB2 Web 工具

可以使用某些 DB2 功能来添加图形工具支持，以从其他供应商提供的软件中开发例程，其中包括:

- IBM Database Add-Ins for Microsoft Visual Studio

IBM Data Studio 例程开发支持

IBM Data Studio 提供易于使用的开发环境来创建、构建、调试、测试以及部署存储过程。

IBM Data Studio 提供图形工具，可让您着眼于存储过程逻辑，而非拘泥于有关生成基本 CREATE 语句、构建以及在 DB2 数据库服务器上安装存储过程的细节，从而简化了创建例程的过程。此外，您可以在一个操作系统上开发存储过程，然后在另一个服务器操作系统上构建这些存储过程。

IBM Data Studio 是一个支持快速开发的图形应用程序。您可以使用该应用程序来执行下列任务:

- 创建新的存储过程。
- 在本地和远程 DB2 数据库服务器上构建存储过程。
- 修改和重建现有存储过程。
- 测试和调试已安装存储过程的执行。

IBM Data Studio 允许您在项目中管理工作。每个 IBM Data Studio 项目都会保存与特定数据库的连接，例如与 DB2 z/OS 版服务器的连接。此外，您可以创建过滤器以显示每个数据库上存储过程的子集。打开新的或现有 Data Studio 项目时，您可以过滤存储过程，以根据名称、模式、语言或集合标识来查看存储过程（用于 DB2 z/OS 版服务器）。

有关 IBM Data Studio 的更多信息，请访问 IDM 信息中心: http://publib.boulder.ibm.com/infocenter/idm/v2r2/index.jsp

可以在例程和触发器中执行的 SQL 语句

能否在例程中成功执行 SQL 语句取决于是否满足特定先决条件的限制和约束。但是，可以在例程和触发器中执行许多 SQL 语句。

如果语句调用例程，那么该语句的有效 SQL 数据访问级别为下列两个级别中的较高级别:

- 下表中语句的 SQL 数据访问级别。
- 创建例程时指定的例程 SQL 数据访问级别。

例如，CALL 语句具有 SQL 数据访问级别 CONTAINS SQL。但是，如果调用定义为 READS SQL DATA 的存储过程，那么 CALL 语句的有效 SQL 数据访问级别为 READS SQL DATA。

下表列出了所有支持的 SQL 语句（其中包括 SQL PL 控制语句），并标识是否可以在各种类型的例程中执行每个 SQL 语句。对于第一列中所列出的每个 SQL 语句，每个后序列都会显示一个 X 以指示该语句是否可以在例程中执行。最后一列标识允许成功执行语句所需的最低 SQL 访问级别。例程调用 SQL 语句时，语句的有效 SQL 数据访问指标不得超过对例程声明的 SQL 数据访问指标。例如，定义为 READS SQL DATA 的函数不能调用定义为 MODIFIES SQL DATA 的过程。除非在脚注中另有说明，否则可静态或动态地执行所有 SQL 语句。

表 5. 可以在例程中执行的 SQL 语句

SQL 语句	可在复合 SQL（编译型）语句中执行（1）	可在复合 SQL（内联型）语句中执行（2）	可在外部过程中执行	可在外部函数中执行	最低的必需 SQL 数据访问级别
ALLOCATE CURSOR	X		X	X	MODIFIES SQL DATA
ALTER {BUFFERPOOL, DATABASE PARTITION GROUP, FUNCTION, METHOD, NICKNAME, PROCEDURE, SEQUENCE, SERVER, TABLE, TABLESPACE, TYPE, USER MAPPING, VIEW}			X	X	MODIFIES SQL DATA

表 5. 可以在例程中执行的 SQL 语句 (续)

SQL 语句	可在复合 SQL（编译型）语句中执行（1）	可在复合 SQL（内联型）语句中执行（2）	可在外部过程中执行	可在外部函数中执行	最低的必需 SQL 数据访问级别
ASSOCIATE LOCATORS	X				
AUDIT			X	X	MODIFIES SQL DATA
BEGIN DECLARE SECTION			X	X	NO SQL(3)
CALL	X	X	X	X	CONTAINS SQL (12)
CASE	X	X			CONTAINS SQL
CLOSE	X		X	X	READS SQL DATA
COMMENT ON	X		X	X	MODIFIES SQL DATA
COMMIT	X(6)		X(6)		MODIFIES SQL DATA
复合 SQL	X	X	X	X	CONTAINS SQL
CONNECT(2)					
CREATE {ALIAS, BUFFERPOOL, DATABASE PARTITION GROUP, DISTINCT TYPE, EVENT MONITOR, FUNCTION, FUNCTION MAPPING, GLOBAL TEMPORARY TABLE(11), INDEX(11), INDEX EXTENSION, METHOD, NICKNAME, PROCEDURE, SCHEMA, SEQUENCE, SERVER, TABLE(11), TABLESPACE, TRANSFORM, TRIGGER, TYPE, TYPE MAPPING, USER MAPPING, VIEW(11), WRAPPER }	X (8)		X		MODIFIES SQL DATA

表 5. 可以在例程中执行的 SQL 语句 (续)

SQL 语句	可在复合 SQL（编译型）语句中执行（1）	可在复合 SQL（内联型）语句中执行（2）	可在外部过程中执行	可在外部函数中执行	最低的必需 SQL 数据访问级别
DECLARE CURSOR	X	X	X		NO SQL(3)
DECLARE GLOBAL TEMPORARY TABLE	X		X	X	MODIFIES SQL DATA
DELETE	X	X	X	X	MODIFIES SQL DATA
DESCRIBE(9)			X	X	READS SQL DATA
DISCONNECT(4)					
DROP	X(8)		X	X	MODIFIES SQL DATA
END DECLARE SECTION			X	X	NO SQL(3)
EXECUTE	X		X	X	CONTAINS SQL(5)
EXECUTE IMMEDIATE	X		x	X	CONTAINS SQL(5)
EXPLAIN	X		X	X	MODIFIES SQL DATA
FETCH	X		X	X	READS SQL DATA
FREE LOCATOR			X	X	CONTAINS SQL
FLUSH EVENT MONITOR			X	X	MODIFIES SQL DATA
FLUSH PACKAGE CACHE			X	X	MODIFIES SQL DATA
FOR	X	X			READS SQL DATA
FREE LOCATOR	X		X	X	CONTAINS SQL
GET DIAGNOSTICS	X	X			READS SQL DATA
GOTO	X	X			CONTAINS SQL
GRANT	X		X	X	MODIFIES SQL DATA
IF	X	X			CONTAINS SQL
INCLUDE			X	X	NO SQL
INSERT	X	X	X	X	MODIFIES SQL DATA
ITERATE	X	X			CONTAINS SQL
LEAVE	X	X			CONTAINS SQL
LOCK TABLE	X		X	X	CONTAINS SQL

表 5. 可以在例程中执行的 SQL 语句 (续)

SQL 语句	可在复合 SQL（编译型）语句中执行（1）	可在复合 SQL（内联型）语句中执行（2）	可在外部过程中执行	可在外部函数中执行	最低的必需 SQL 数据访问级别
LOOP	X	X			CONTAINS SQL
MERGE	X	X	X	X	MODIFIES SQL DATA
OPEN	X		X	X	READS SQL DATA(7)
PREPARE	X		X	X	CONTAINS SQL
REFRESH TABLE			X	X	MODIFIES SQL DATA
RELEASE(4)					
RELEASE SAVEPOINT	X		X	X	MODIFIES SQL DATA
RENAME TABLE			X	X	MODIFIES SQL DATA
RENAME TABLESPACE			X	X	MODIFIES SQL DATA
REPEAT	X	X			CONTAINS SQL
RESIGNAL	X				MODIFIES SQL DATA
RETURN	X				CONTAINS SQL
REVOKE			X	X	MODIFIES SQL DATA
ROLLBACK(6)	X		X		
ROLLBACK TO SAVEPOINT	X		X	X	MODIFIES SQL DATA
SAVEPOINT	X				MODIFIES SQL DATA
SELECT 语句	X		X	X	READS SQL DATA
SELECT INTO	X		X(10)	X(10)	READS SQL DATA(7)
SET CONNECTION(4)					
SET INTEGRITY			X		MODIFIES SQL DATA
SET 专用寄存器	X	X	X	X	CONTAINS SQL
SET 变量	X	X			CONTAINS SQL
SIGNAL	X	X			MODIFIES SQL DATA
TRANSFER OWNERSHIP			X	X	MODIFIES SQL DATA

表 5. 可以在例程中执行的 SQL 语句 (续)

SQL 语句	可在复合 SQL（编译型）语句中执行（1）	可在复合 SQL（内联型）语句中执行（2）	可在外部过程中执行	可在外部函数中执行	最低的必需 SQL 数据访问级别
TRUNCATE			X	X	MODIFIES SQL DATA
UPDATE	X	X	X		MODIFIES SQL DATA
VALUES INTO	X		X	X	READS SQL DATA
WHENEVER	X		X		NO SQL(3)
WHILE	X	X			

注:

1. 可以将复合 SQL（编译型）语句用作 SQL 过程、SQL 函数和触发器的主体，或用作独立语句。
2. 可以将复合 SQL（内联型）语句用作 SQL 函数、SQL 方法和触发器的主体，或用作独立语句。
3. 虽然 NO SQL 选项意味着无法指定任何 SQL 语句，但未对不可执行的语句加以限制。
4. 任何例程执行上下文中都不允许连接管理语句。
5. 此情况取决于所执行的语句。在生效的特定 SQL 访问级别的上下文中，必须允许针对 EXECUTE 语句指定的语句。例如，如果生效的 SQL 访问级别为 READS SQL DATA，那么语句不能是 INSERT、UPDATE 或 DELETE。
6. 可以在存储过程中使用 COMMIT 语句和 ROLLBACK 语句（不带 TO SAVEPOINT 子句），但只有在从应用程序直接地调用该存储过程，或从应用程序通过嵌套的存储过程调用间接地调用该存储过程时，才能这样做。如果存储过程的调用链中存在任何触发器、函数、方法或原子复合语句，那么不允许对工作单元执行 COMMIT 或 ROLLBACK。
7. 如果生效的 SQL 访问级别为 READS SQL DATA，那么无法将任何 SQL 数据更改语句嵌入至 SELECT INTO 语句，也无法嵌入至 OPEN 语句所引用的游标。
8. SQL 过程只能对索引、表以及视图发出 CREATE 和 DROP 语句。
9. DESCRIBE SQL 语句的语法可能有别于 CLP DESCRIBE 命令的语法。
10. 仅受嵌入式 SQL 例程支持。
11. 只能静态地执行 SQL 过程中所引用的语句。
12. 与受影响的当前级别相比，调用的过程必须具有相同或更加严格的 SQL 数据访问级别。例如，定义为 MODIFIES SQL DATA 的例程可以调用定义为 MODIFIES SQL DATA、READS SQL DATA、CONTAINS SQL 或 NO SQL 的过程。定义为 CONTAINS SQL 的例程可以调用定义为 CONTAINS SQL 或 NO SQL 的例程。为该过程指定的自变量可能还需要另一个数据访问级别。例如，作为自变量的标量全查询要求该语句的数据访问级别为 READS SQL DATA。

错误

第 30 页的表 5 指示是否允许第一列所指定的 SQL 语句在具有指定的 SQL 数据访问级别的例程中执行。如果语句超出该数据访问级别，那么在执行例程时会返回错误。

- 如果在定义了 NO SQL 数据访问级别的例程中遇到可执行的 SQL 语句，那么将返回 SQLSTATE 38001。
- 对于任何其他执行上下文都不支持的 SQL 语句，将返回 SQLSTATE 38003 错误。
- 对于 CONTAINS SQL 上下文不允许的其他 SQL 语句，将返回 SQLSTATE 38004。
- 在 READS SQL DATA 上下文中，将返回 SQLSTATE 38002。
- 在创建 SQL 例程期间，与 SQL 数据访问级别不匹配的语句将返回 SQLSTATE 42985 错误。

例程中的 SQL 访问级别

例程能够执行 SQL 语句的程度由该例程的 SQL 访问级别确定。例程的 SQL 访问级别由允许特定类型例程执行的 SQL 访问以及定义该例程时使用的 CREATE 语句中明确指定的限制确定。

SQL 访问级别如下所示:

- NO SQL
- CONTAINS SQL
- READS SQL
- MODIFIES SQL

此 SQL 访问级别子句用于向数据库管理器提供关于语句的信息，以使数据库管理器能够安全地执行该语句并获得最佳性能。

不同类型的例程的缺省和最大 SQL 访问级别如下表所示:

表 6. 例程的缺省和最大 SQL 访问级别

例程类型	缺省 SQL 访问级别	允许的最大 SQL 访问级别
SQL 过程	MODIFIES SQL DATA	MODIFIES SQL DATA
SQL 函数（标量函数）	READS SQL DATA	READS SQL DATA
SQL 函数（表函数）	READS SQL DATA	MODIFIES SQL DATA
外部过程	MODIFIES SQL DATA	MODIFIES SQL DATA
外部函数（标量函数）	READS SQL DATA	READS SQL DATA
外部函数（表函数）	READS SQL DATA	READS SQL DATA

当例程 CREATE 语句指定限制性最强的有效 SQL 访问子句时，例程的性能最佳。

在例程的 CREATE 语句中:

- 如果明确指定 READS SQL DATA，那么该例程中的所有 SQL 语句都无法修改数据。
- 如果明确指定 CONTAINS SQL DATA，那么该例程中的所有 SQL 语句都无法修改或读取数据。
- 如果明确指定 NO SQL，那么该例程不能包含可执行的 SQL 语句。

确定可以在例程中执行的 SQL 语句

可以在例程中执行许多 SQL 语句，但并非可以执行所有 SQL 语句。特定 SQL 语句在例程中的执行取决于该例程的类型、该例程的实现、您对该例程指定的最大 SQL 访问级别以及例程定义者和调用者的特权。

开始之前

通过在实现例程之前确定可以在该例程中执行的 SQL 语句，可以确保从一开始就正确选择例程类型和实现。

要在例程中成功地执行 SQL 语句，必须满足下列先决条件:

- 例程的 SQL 访问级别必须允许执行特定的 SQL 语句。
  - 在例程的 CREATE 语句中指定了例程的 SQL 访问级别。
  - 某些 SQL 访问级别对于某些类型的例程而言不受支持。请参阅以下限制。
- 例程定义者必须具有执行 SQL 语句所必需的特权。
  - 《SQL 参考》提供了执行每个受支持的 SQL 语句所需的特权。
- 没有其他单独的限制会限制语句的执行。
  - 请参阅《SQL 参考》以获取特定于给定 SQL 语句的限制列表。

限制

下列限制将限制可以在例程中执行的 SQL 语句的集合。特别是，这些限制将限制可以对特定类型的例程指定的 SQL 访问级别:

- 不能对外部函数指定 MODIFIES SQL DATA 访问级别。
- 不能对将从触发器中调用的外部过程指定 MODIFIES SQL DATA 访问级别。

过程

要确定可以在特定例程中调用哪些 SQL 语句，请执行以下操作:

1. 确定例程的 SQL 访问级别。如果它是现有例程，请检查先前用来创建该例程的 CREATE 语句。您可能已在 DDL 中将 SQL 访问级别子句显式定义为下列其中一项: NO SQL、CONTAINS SQL、READS SQL DATA 或 MODIFIES SQL DATA。如果未显式指定此类子句，那么例程的缺省值以隐式方式指定。
   - 对于 SQL 过程，缺省值是 MODIFIES SQL DATA。
   - 对于 SQL 函数，缺省值是 MODIFIES SQL DATA。
   - 对于外部过程，缺省值是 MODIFIES SQL DATA。
   - 对于外部函数，缺省值是 READS SQL DATA。
2. 请参阅“可以在例程中执行的 SQL 语句”主题中的表。请按名称查找您感兴趣的 SQL 语句。
3. 检查特定类型的例程和实现是否支持该 SQL 语句。
4. 验证执行该语句所需的 SQL 访问级别是否与例程的 SQL 访问级别匹配。
5. 仔细阅读任何用法说明或脚注，以确保不存在有关该 SQL 语句的执行的其他限制。

结果

如果指示可以在例程中执行该 SQL 语句，例程 SQL 访问级别满足在例程中执行该语句的先决条件，并且所有其他先决条件均满足，那么应该能够在例程中成功地执行该 SQL 语句。

例程可移植性

例程可移植性是指部署例程的方便程度。可移植性包括操作系统兼容性、运行时环境兼容性、软件兼容性、调用接口兼容性之类的因素，并包括有关支持在例程中执行 SQL 语句的兼容性之类的其他例程实现因素。

如果要将例程部署到的环境与例程的开发环境并不完全相同，那么例程可移植性至关重要。通常，DB2 例程在操作系统之间高度可移植，即使在不同 DB2 数据库产品和版本之间亦如此。在开始开发例程之前，最好考虑潜在的可移植性问题，以便最大程度地降低以后需要进行返工的可能性。

下列主题包括有关可能会限制例程可移植性的因素的信息:

- 受支持的 DB2 Database for Linux, UNIX, and Windows 版本
- 受支持的开发和编译器软件
- 可以在例程中执行的 SQL 语句
- 有关例程的限制
- 部署例程

例程互操作性

不同类型以及具有不同编程实现的例程之间的互操作性确保例程在数据库系统生命周期内可以成为高度可复用的模块。

由于代码模块通常由具有编程技术的不同程序员使用不同编程语言实现，并且通常最好尽可能地复用代码以缩短开发时间并降低成本，因此 DB2 例程基础结构设计成支持较高的例程互操作性水平。

例程互操作性的特征是，能够从例程中无缝地引用和调用其他具有不同类型和实现的例程，并且没有任何附加要求。DB2 例程的互操作性体现在下列方面:

- 使用一种编程语言编写的客户机应用程序可以调用使用另一编程语言实现的例程。
  - 例如，C 客户机应用程序可以调用 .NET 公共语言运行时例程。
- 一个例程可以调用另一个例程，而与该例程的例程类型或实现语言无关。
  - 例如，Java 过程（这是一种例程）可以调用 SQL 标量函数（这是使用另一实现语言的另一种例程）。
- 在运行于一款操作系统上的 DB2 客户机中，可以调用在运行于另一款操作系统上的数据库服务器中创建的例程。

您可以通过各种类型的例程来满足特定功能需求和各种例程实现。对例程类型和实现所作的选择可影响以前列出的益处的表现水平。通常，例程是一种功能强大的逻辑封装方法，使您能够扩展 SQL、改善结构和维护以及有可能提高应用程序性能。

例程的性能

例程的性能受到各种因素的影响，其中包括例程的类型和实现、例程中的 SQL 语句数、例程中 SQL 的复杂度、例程的参数个数、例程实现中逻辑的效率以及例程中的错误处理等。

因为用户通常选择实现例程以改进应用程序的性能，所以必须将例程性能发挥至极致。

下表概述其中一些会影响例程性能的常规因素，并且提供一些有关如何通过改变每个因素来改进例程性能的建议。有关影响特定例程类型的性能因素的更多详细信息，请参阅特定例程类型的性能和调整主题。

表 7. 性能注意事项和例程性能建议

性能注意事项	性能建议
例程类型：过程、函数和方法	• 过程、函数和方法具有不同的用途，且在不同的位置进行引用。因为它们的功能存在差别，所以很难直接比较其性能。 • 通常，有时可以将过程重新编写为函数（特别是如果函数返回标量值且只查询数据），以此稍微改进性能，但是通常是通过简化实现 SQL 逻辑所需的 SQL 来获取这些益处。 • 具有复杂初始化的用户定义的函数可以利用暂存区来存储第一次调用中所需的任何值，以便可以在后续调用中使用这些值。
例程实现：内置或用户定义	• 对于等价的逻辑，内置例程性能最佳，用户定义的例程次之，因为较之用户定义的例程，它们与数据库引擎的关系要密切一些。 • 如果编写的用户定义的例程很出色且遵循下列最佳实践，那么可以实现非常高的性能。
例程实现：SQL 或外部例程实现	• SQL 例程较之外部例程更有效，因为 SQL 例程是由 DB2 数据库服务器直接执行。 • SQL 过程较之逻辑上等价的外部例程通常具有更高的性能。 • 对于简单逻辑，SQL 函数性能与等价外部函数相当。 • 对于复杂逻辑（例如需要较少 SQL 的数学算法和字符串操作函数），较好的作法是使用低级编程语言（例如 C）编写外部例程，因为这样做对 SQL 支持的依赖性较低。 • 请参阅比较例程实现，以对受支持的各个外部例程编程语言选项的特征（其中包括性能）进行比较。

表 7. 性能注意事项和例程性能建议 (续)

性能注意事项	性能建议
外部例程实现编程语言	• 请参阅外部例程 API 和编程语言比较，以对在选择外部例程实现时应当考虑的性能特征进行比较。 • Java（JDBC 和 SQLJ API） – 最好是在指定 FENCED NOT THREADSAFE 子句的情况下，创建存在大量内存需求的 Java 例程。可以在指定 FENCED THREADSAFE 子句的情况下创建存在一般内存需求的 Java 例程。 – 对于 FENCED THREADSAFE Java 例程调用，DB2 数据库系统会尝试选择其 Java 堆大小足以运行例程的线程 Java FENCED 方式进程。如果未能将大型堆使用者限制于其自己的进程，那么会导致多线程 Java **db2fmp** 进程发生“Java 堆不足”错误。相比之下，FENCED THREADSAFE 例程具有更高的性能，因为它们共享几个 JVM。 • C 和 C++ – 通常，C 和 C++ 例程较之其他外部例程实现以及 SQL 例程具有更高的性能。 – 要将性能发挥至极致，应该以 32 位格式（如果要将 C 和 C++ 例程部署至 32 位 DB2 实例）和 64 位格式（如果要将 C 和 C++ 例程部署至 64 位 DB2 实例）编译 C 和 C++ 例程。 • COBOL – 通常，COBOL 的性能很好，但 COBOL 不是建议的例程实现。
例程中的 SQL 语句数	• 例程应该包含多个 SQL 语句，否则例程调用的开销无法达到性能成本效益。 • 如果逻辑必须执行若干数据库查询，处理中间结果，然后最终返回所处理数据的子集，那么该逻辑是例程封装的最佳逻辑。此类型的逻辑示例为复杂数据挖掘以及需要查询相关数据的大型更新。在数据库服务器上完成高负荷 SQL 处理，然后只将较小的数据结果集传递回至调用者。
例程中 SQL 语句的复杂度	• 合理的作法是将非常复杂的查询包含在例程中，以便利用数据库服务器的更大内存和性能能力。 • 不必担心 SQL 语句变得过度复杂。
例程中的静态或动态 SQL 执行	• 通常，静态 SQL 较之动态 SQL 具有更高的性能。除此之外，在例程中使用静态 SQL 还是动态 SQL 不存在任何其他差别。
例程的参数个数	• 将参数个数减到最少可以改进例程性能，因为这会将例程和例程调用程序之间所传递的缓冲区数减到最少。

表 7. 性能注意事项和例程性能建议 (续)

性能注意事项	性能建议
例程参数的数据类型	• 您可以在例程定义中使用 VARCHAR 参数取代 CHAR 参数来改进例程的性能。使用 VARCHAR 数据类型取代 CHAR 数据类型可以防止 DB2 数据库系统在传递参数之前使用空格来填充参数，并且可以缩短通过网络传输参数所需的时间量。 例如，如果您的客户机应用程序将字符串“A SHORT STRING”传递至期望 CHAR(200) 参数的例程，那么 DB2 数据库系统必须使用 186 个空格来填充该参数，以 null 结束字符串，然后通过网络将整个由 200 个字符组成的字符串和一个 null 终止符发送至例程。 相比之下，将同一字符串 *"A SHORT STRING"* 传递至期望 VARCHAR(200) 参数的例程，会导致 DB2 数据库系统只需通过网络传递一个由 14 个字符组成的字符串和一个 null 终止符。
例程参数初始化	• 最好是始终初始化例程的输入参数，特别是如果输入例程参数值为 null。对于 null 值例程参数，可以将较短或空的缓冲区（而不是完整大小的缓冲区）传递至例程，这样做可以改进性能。
例程中的局部变量数	• 将例程中所声明的局部变量数减到最少，可以将例程中执行的 SQL 语句数减到最少，从而改进性能。 • 通常，目标是使用尽可能少的变量。如果复用变量不会导致语义混淆，那么可以复用变量。
例程中局部变量的初始化	• 如果可能，那么较好的作法是在单一 SQL 语句中初始化多个局部变量，因为这样做会节省例程的合计 SQL 执行时间。
过程所返回的结果集数	• 如果您可以减少例程所返回的结果集数，那么可以改进例程性能。
例程所返回的结果集大小	• 确保对于例程所返回的每个结果集，对结果进行定义的查询会尽可能多地过滤所返回的列数和行数。返回不必要的数据列或数据行不仅无效，而且会导致例程性能欠佳。
例程中逻辑的效率	• 和任何应用程序一样，未很好实现的算法会限制例程的性能。进行例程编程时目标是尽可能地高效，并且尽可能应用常规的最佳编程实践建议。 • 分析您的 SQL，而且只要有可能，就将查询简化成简单的格式。通过使用 CASE 表达式取代 CASE 语句，或将多个 SQL 语句折叠为使用 CASE 表达式作为开关的单一语句，常常可以实现此目的。

表 7. 性能注意事项和例程性能建议 (续)

性能注意事项	性能建议
例程的运行时方式（FENCED 或 NOT FENCED 子句规范）	**NOT FENCED 子句用法:** • 通常，使用 NOT FENCED 子句来创建例程（这将使例程与 DB2 数据库管理器在同一进程中运行），较之使用 FENCED 子句来创建例程（这将使例程在引擎地址空间外部的特殊 DB2 数据库进程中运行）更可取。 • 虽然以 NOT FENCED 方式运行例程时可以期望改进的例程性能，但未防护例程中的用户代码可能会意外地或恶意地毁坏数据库或损坏数据库控制结构。只有在您需要将性能益处发挥至极致，且认为例程安全时，才应该使用 NOT FENCED 子句。有关评估和降低将 C/C++ 例程注册为 NOT FENCED 例程的风险的信息，请参阅例程安全性。如果例程不够安全，不能在数据库管理器的进程中运行，请在创建例程时使用 FENCED 子句。为了限制创建和运行可能不安全的代码，DB2 数据库系统要求用户具有特权 CREATE_NOT_FENCED_ROUTINE 才能创建 NOT FENCED 例程。 • 如果在您运行 NOT FENCED 例程时发生异常终止，且例程已注册为 NO SQL，那么数据库管理器将尝试适当的恢复操作。但是，对于未定义为 NO SQL 的例程，数据库管理器将失败。 • 如果 NOT FENCED 例程使用 GRAPHIC 或 DBCLOB 数据，那么必须使用 WCHARTYPE NOCONVERT 选项预编译 NOT FENCED 例程。

表 7. 性能注意事项和例程性能建议 (续)

性能注意事项	性能建议
例程的运行时方式（FENCED 或 NOT FENCED 子句规范）	**FENCED THREADSAFE 子句用法** • 使用 FENCED THREADSAFE 子句来创建的例程与其他例程在同一进程中运行。更具体地说，非 Java 例程会共享一个进程，而 Java 例程会共享另一个进程（和使用其他语言编写的例程是分离的）。此分离可保护 Java 例程，使其不受其他语言编写且可能更易出错的例程影响。此外，Java 例程的进程还包含一个 JVM，此 JVM 会引发高昂的内存成本，因此不为其他例程类型所使用。FENCED THREADSAFE 例程的多个调用会共享资源，因此所引发的系统开销小于 FENCED NOT THREADSAFE 例程（每个 FENCED NOT THREADSAFE 例程都在其自己的专用进程中运行）。 • 如果您认为例程足够安全，可以与其他例程在同一例程中运行，请在注册该例程时使用 THREADSAFE 子句。和 NOT FENCED 例程一样，有关评估和缓解将 C/C++ 例程注册为 FENCED THREADSAFE 的风险的信息可在『例程安全性注意事项』主题中找到。 • 如果 FENCED THREADSAFE 例程会异常结束，那么将仅终止正在运行此例程的线程。进程中的其他例程会继续运行。但是，导致此线程异常结束的故障会对进程中的其他例程线程产生负面影响，使这些线程进入陷阱、挂起或具有损坏的数据。在一个线程异常结束后，进程就不再用于新的例程调用。在所有活动用户完成其在此进程中的作业后，会终止此进程。 • 当您注册 Java 例程时，除非另有声明，否则会将这些例程视为 THREADSAFE。缺省情况下，所有其他 LANGUAGE 类型都是 NOT THREADSAFE。不能将使用 LANGUAGE OLE 和 OLE DB 的例程指定为 THREADSAFE。 • NOT FENCED 例程必须为 THREADSAFE。无法将例程注册为 NOT FENCED NOT THREADSAFE (SQLCODE -104)。 • UNIX 上的用户可通过查找 **db2fmp** (Java) 或 **db2fmp** (C) 来查看其 Java 和 C THREADSAFE 进程。
例程的运行时方式（FENCED 或 NOT FENCED 子句规范）	**FENCED NOT THREADSAFE 方式** • 每个 FENCED NOT THREADSAFE 例程都在其自己的专用进程中运行。如果正在运行许多例程，那么会对数据库系统性能产生不利影响。如果例程不够安全，不能与其他例程在同一进程中运行，请在注册该例程时使用 NOT THREADSAFE 子句。 • 在 UNIX 上，NOT THREADSAFE 进程对于合用的 NOT THREADSAFE **db2fmp** 显示为 `db2fmp (pid)`（其中 *pid* 是使用 FENCED 方式进程的代理程序的进程标识）或 `db2fmp (idle)`。

表 7. 性能注意事项和例程性能建议 (续)

性能注意事项	性能建议
例程中 SQL 访问的级别: NO SQL、CONTAINS SQL、READS SQL DATA 以及 MODIFIES SQL DATA	• 使用较低级别的 SQL 访问子句来创建的例程，较之使用较高级别 SQL 访问子句来创建的例程具有更高的性能。因此，您应该使用最高限制级别的 SQL 访问子句来声明您的例程。例如，如果例程只读取 SQL 数据，请不要使用 MODIFIES SQL DATA 子句来创建该例程，而是使用限制性较强的 READS SQL DATA 子句来创建该例程。
例程确定性（DETERMINISTIC 或 NOT DETERMINISTIC 子句规范）	• 使用 DETERMINISTIC 或 NOT DETERMINISTIC 子句来声明例程不会影响例程性能。
例程所执行的外部操作数目和复杂度（EXTERNAL ACTION 子句规范）	• 例程性能可能会受阻碍，取决于外部例程所执行的外部操作数目以及外部操作复杂度。导致受阻碍的因素是网络流量、对所读写的文件进行访问、执行外部操作所需的时间以及与外部操作代码或行为中的挂起相关联的风险。
输入参数为 null 时的例程调用（CALLED ON NULL INPUT 子句规范）	• 如果收到 null 输入参数值会导致不执行任何逻辑，且导致例程立即返回，那么可以修改您的例程，以便在检测到 null 输入参数值时不会对例程进行完整调用。要创建一个在收到例程输入参数时提前结束调用的例程，请在创建时指定 CALLED ON NULL INPUT 子句。
类型为 XML 的过程参数	• 在使用 C 或 JAVA 编程语言来实现的外部过程中，传递数据类型为 XML 的参数的效率较之在 SQL 过程中明显低得多。传递一个或多个数据类型为 XML 的参数时，请考虑使用 SQL 过程取代外部过程。 • 将 XML 数据作为 IN、OUT 或 INOUT 参数传递至存储过程时，会具体化此 XML 数据。如果使用的是 Java 存储过程，那么可能需要根据 XML 自变量的数量和大小，以及正在并发执行的外部存储过程数来增加堆大小（**java_heap_sz** 配置参数）。

在创建并部署例程后，可能更难确定哪些环境和例程特定因素会影响例程性能，因此在设计例程时必须将性能问题牢记在心。

例程安全性

例程安全性非常重要，可确保例程连续运作，以将篡改风险降至最低并保护数据库系统环境。不同的风险级别分别具有几个类别的例程安全性注意事项。在开发或维护例程时，必须注意这些风险以尽可能缓解不合适的结果。

对可以创建例程的人员进行安全性控制

授予用户必要的特权来执行在数据库中创建例程时所需的 CREATE 语句时，例程安全性即开始出现。授予这些特权时，必须了解对应的风险:

- 具有针对例程执行 CREATE 语句的特权的用户可以创建多个例程。
- 具有针对例程执行 CREATE 语句的特权的用户可以创建例程，这些例程可修改数据库布局或数据库数据（取决于用户具有的其他特权）。
- 会自动向成功创建例程的用户授予调用例程所需的 EXECUTE 特权。

- 会自动向成功创建例程的用户授予修改例程所需的 ALTER ROUTINE 特权。

要将用户修改数据库和数据所带来的风险降至最低，请执行下列操作:

- 将具有创建例程的特权的用户数减至最少。
- 确保除去离职职员的用户标识，或者如果复用这些用户标识，请确保评估过程相关特权。

请参阅有关控制对数据库对象和数据的访问权的主题，以了解有关如何向一个、许多或所有数据库用户授予特权以及撤销其特权的更多信息。

对可以调用例程的人员进行安全性控制

确定用户何时需要特权很容易: 他们无法完成某项工作。确定用户何时不再需要这些特权有些难度。这对于具有调用例程特权的用户而言尤为如此，因为允许他们保留特权会带来风险:

- 已被授予 EXECUTE 特权来调用例程的用户将继续能够调用例程，除非此特权已除去。如果例程包含敏感逻辑或处理敏感数据，那么可能会造成业务风险。

要将用户修改数据库和数据所带来的风险降至最低，请执行下列操作:

- 将具有调用例程的特权的用户数减至最少。
- 确保除去离职职员的用户标识，或者如果复用这些用户标识，请确保评估过程相关特权。
- 如果您怀疑某人恶意地调用例程，那么应该针对其中的每个例程撤销 EXECUTE 特权。

对定义了 FENCED 或 NOT FENCED 子句的例程的安全性控制

构造例程的 CREATE 语句时，您必须确定是否需要指定 FENCED 子句或 NOT FENCED 子句。在了解创建 FENCED 或 NOT FENCED 例程的益处后，必须评估与运行例程（其外部实现是 NOT FENCED）相关联的风险。

- 使用 NOT FENCED 子句创建的例程可能会意外或恶意地毁坏数据库管理器的共享内存、损坏数据库控制结构或访问数据库管理器资源，导致数据库管理器失败。此外，还有毁坏数据库及其表的风险。

要确保数据库管理器及其数据库的完整性，请执行下列操作:

- 彻底地屏蔽您打算创建的例程（会指定 NOT FENCED 子句）。这些例程必须经过完整测试、调试，且不会显示任何意外的负面效应。在检查例程代码时，请密切注意内存管理和静态变量的使用。当代码错误地管理内存或错误地使用静态变量时，极有可能会发生毁坏。这些问题在 Java 和 .NET 编程语言之外的语言中非常普遍。

需要 CREATE_NOT_FENCED_ROUTINE 权限才能注册 NOT FENCED 例程。授予 CREATE_NOT_FENCED_ROUTINE 权限时，请注意接收方可能会取得对数据库管理器及其所有资源的不受限访问权。

注: Common Criteria 兼容配置不支持 NOT FENCED 例程。

保护例程的安全

创建例程时，必须确保牢记通过例程安全性来管理例程、例程库（如果是外部例程）以及将与例程交互的用户的特权。

开始之前

虽然，可能不必像例程安全性策略那样复杂，但是在保护例程的安全性时，牢记构成例程安全性的因素并遵循严谨的方法很实用。

- 阅读主题“例程安全性”。
- 要在数据库系统中充分保护例程的安全，您必须具有:
  - 数据库服务器操作系统上的 root 用户访问权。
  - SECADM 或 ACCESSCTRL 权限之一。

关于此任务

不论您是创建例程还是访问现有例程，保护例程安全的过程都类似。

过程

1. 限制具有创建例程所需的特权的用户标识数，并确保这些用户有权具备这些特权。
   - 针对例程成功执行 CREATE 语句后，会自动向此用户标识授予其他特权（其中包括 EXECUTE 特权，此特权允许用户调用例程）和 GRANT EXECUTE 特权（此特权允许用户向其他用户授予调用例程的能力）。
   - 确保只有少数用户具有此特权，并且只有适当的用户才获取此特权。
2. 评估例程中是否有潜在的恶意代码，或是否有未经过充分复审或测试的代码。
   - 考虑例程的来源。提供例程的一方是否可靠？
   - 查找恶意代码，例如，尝试读/写数据库服务器文件系统及/或替换其中的文件的代码。
   - 查找实现质量不高的代码，例如，与内存管理、指针操作以及静态变量使用（可能会导致例程失败）的代码。
   - 验证是否充分测试代码。
3. 拒绝极其不安全或编写质量非常差的例程 - 使用这些例程，弊大于利。
4. 包含与可能有点潜在风险的例程相关联的风险。
   - SQL 用户定义的 SQL 例程在缺省情况下创建为 NOT FENCED THREADSAFE 例程，因为这些例程可以在数据库管理器内存空间中安全运行。对于这些例程，您不需要执行任何处理。
   - 在例程的 CREATE 语句中指定 FENCED 子句。这将确保例程操作不会影响数据库管理器。这是缺省子句。
   - 如果例程是多线程例程，请在例程的 CREATE 语句中指定 NOT THREADSAFE 子句。这将确保该例程中的任何失败或恶意代码都不会影响其他可能在共享线程进程中运行的例程。
5. 如果例程是外部例程，那么您必须将例程实现库或类文件放在数据库服务器上。请遵循用于部署例程的常规建议，以及用于部署外部例程库或类文件的特定建议。

对包含 SQL 的例程进行授权和绑定

讨论例程级别权限时，必须定义一些与例程相关的角色、确定角色以及定义与这些角色相关的特权:

程序包所有者

参与例程实现的特定程序包的所有者。程序包所有者是执行 **BIND** 命令以将程序

包与数据库绑定在一起的用户，除非使用 **OWNER PRECOMPILE** 或 **BIND** 命令参数覆盖程序包所有者资格并将其设置为另一用户。执行 **BIND** 命令时，程序包所有者被授予对该程序包的 EXECUTE WITH GRANT 特权。例程库或可执行文件可以包含多个程序包，因此可以具有多个相关联的程序包所有者。

例程定义者

发出 CREATE 语句以注册例程的标识。例程定义者一般是 DBA，但通常也是例程程序包所有者。在调用例程期间装入程序包时，会根据定义者对例程的相关联程序包的执行权限（而不是根据例程调用程序的权限）来检查例程的运行权限。要成功调用例程，例程定义者必须具有下列其中一项特权或权限:

- 例程程序包的 EXECUTE 特权和例程的 EXECUTE 特权
- DATAACCESS 权限

如果例程定义者和例程程序包所有者是同一用户，那么例程定义者将具有程序包的必需 EXECUTE 特权。如果定义者不是程序包所有者，那么必须由具有程序包的 ACCESSCTRL 或 SECADM 权限、CONTROL 或 EXECUTE WITH GRANT OPTION 特权的用户，将程序包的 EXECUTE 特权显式地授予给定义者。（程序包的创建者会自动接收程序包的 CONTROL 和 EXECUTE WITH GRANT OPTION。）

在发出可以注册例程的 CREATE 语句时，会隐式地将例程的 EXECUTE WITH GRANT OPTION 特权授予给定义者。

例程定义者的角色是在一个授权标识下封装例程的相关联程序包的运行特权，以及将例程的 EXECUTE 特权授予给 PUBLIC 用户或特定用户（需要调用例程）的特权。

注: 对于 SQL 例程，例程定义者也就是程序包所有者。因此，定义者在执行例程的 CREATE 语句时，将具有例程和例程程序包的 EXECUTE WITH GRANT OPTION。

例程调用者

调用例程的标识。要确定将成为例程调用者的用户，必须考虑如何调用例程。可以从命令窗口或从嵌入式 SQL 应用程序调用例程。如果是方法和 UDF，那么会将例程引用嵌入另一个 SQL 语句。使用 CALL 语句来调用过程。对于应用程序中的动态 SQL，调用者是包含例程调用的直接较高级别例程或应用程序的运行时授权标识（但是，此标识也取决于用来绑定较高级别例程或应用程序的 **DYNAMICRULES** 选项）。对于静态 SQL，调用者是包含例程引用的程序包的 **OWNER PRECOMPILE** 或 **BIND** 命令参数的值。这些用户将需要例程的 EXECUTE 特权，才能成功调用例程。任何具有例程 EXECUTE WITH GRANT OPTION 特权（这包括例程定义者，除非已显式地撤销该特权）、ACCESSCTRL 或 SECADM 权限的用户，都可以通过显式地发出 GRANT 语句来授予此特权。

例如，如果使用 **DYNAMICRULES** BIND 来绑定与包含动态 SQL 的应用程序相关联的程序包，那么程序包的运行时授权标识将是程序包所有者（而不是调用程序包的用户）。此外，程序包所有者还将成为实际绑定者或 **OWNER PRECOMPILE** 或 **BIND** 命令参数的值。在此情况下，例程的调用者会采用此值（而不是执行应用程序的用户标识）。

注:

1. 对于例程中的静态 SQL，程序包所有者的特权必须足以执行例程主体中的 SQL 语句。如果存在例程的嵌套引用，那么这些 SQL 语句可能需要表访问特权或执行特权。
2. 对于例程中的动态 SQL，由例程主体的 **BIND** 的 **DYNAMICRULES** 选项控制将进行特权验证的用户标识。
3. 例程程序包所有者必须将程序包的 EXECUTE 特权授予给例程定义者。可以在注册例程之前或之后完成此操作，但必须在调用例程之前完成，否则将返回错误 (SQLSTATE 42051)。

管理例程的执行特权所涉及的步骤，会在后续图和文本中加以详细说明:

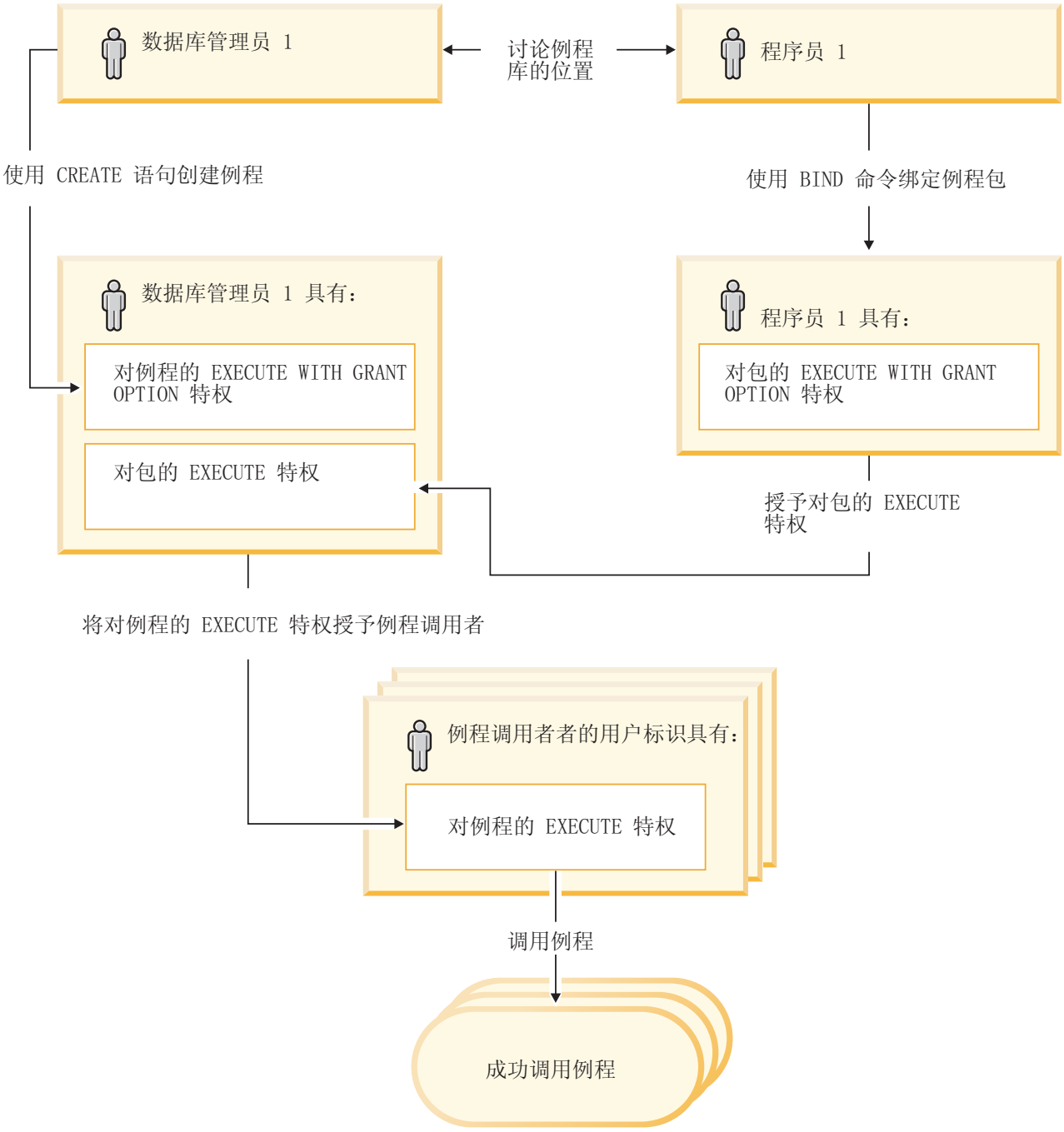


图 2. 管理例程的 EXECUTE 特权

1. 定义者执行相应的 CREATE 语句以注册例程。这将以期望的 SQL 访问级别在 DB2 数据库系统中注册例程，建立例程特征符，此外还指向例程可执行文件。如果定义者并非同时为程序包所有者，那么定义者需要与例程程序的程序包所有者和作者沟通，以清楚了解例程库所在的位置，这样就可以在 CREATE 语句的 EXTERNAL 子句中正确地指定此位置。由于成功执行 CREATE 语句，因此定义者具有例程的 EXECUTE WITH GRANT 特权，但是定义者尚不具有例程程序包的 EXECUTE 特权。
2. 定义者必须将例程的 EXECUTE 特权授予给任何有权使用例程的用户。（如果此例程的程序包将递归地调用此例程，那么必须在下一步之前完成此步骤。）

3. 程序包所有者预编译和绑定例程程序，或让其他用户代表他们完成这些操作。在成功预编译和绑定后，会将各个程序包的 EXECUTE WITH GRANT OPTION 特权隐式地授予给程序包所有者。此步骤遵循此列表中的第 1 步，只涵盖例程中 SQL 递归的可能性。如果任何特定情况下都不存在此类递归，那么预编译/绑定可以在针对例程发出 CREATE 语句之前执行。
4. 每个程序包所有者都必须将各个例程程序包的 EXECUTE 特权授予给例程的定义者。在完成上一步之后，必须等一段时间，再执行此步骤。如果程序包所有者也是例程定义者，那么可以跳过此步骤。
5. 例程的静态用法：必须确实将例程的 EXECUTE 特权，授予给引用该例程的程序包绑定所有者，因此必须在此刻完成上一步。当例程执行时，DB2 数据库系统会验证定义者是否对任何所需的程序包具有 EXECUTE 特权，因此必须针对每个这样的程序包完成第 3 步。
6. 例程的动态用法：授权标识（由调用应用程序的 **DYNAMICRULES** 选项控制）必须对例程具有 EXECUTE 特权（第 4 步），而例程的定义者必须对程序包具有 EXECUTE 特权（第 3 步）。

过程读/写表时数据发生冲突

要维持数据库的完整性，必须避免读/写表时发生冲突。

例如，假设应用程序正在更新 EMPLOYEE 表，且语句会调用例程。假设例程尝试读取 EMPLOYEE 表，但遇到应用程序正在更新的行。从例程的角度，该行处于不确定状态，可能是行的有些列已更新，而有些列未更新。如果例程操作此部分更新的行，那么可能会执行不正确的操作。为了避免此类问题，DB2 数据库系统不允许在任何表上执行冲突的操作。

为了描述 DB2 数据库系统如何避免在通过例程读/写表时发生冲突，需要下列两个术语:

顶层语句

顶层语句是从应用程序发出，或从作为顶层语句来调用的存储过程发出的任何 SQL 语句。如果在动态复合语句或触发器中调用过程，那么导致触发器触发的复合语句或语句是顶层语句。如果 SQL 函数或 SQL 方法包含嵌套的 CALL 语句，那么调用该函数或方法的语句是顶层语句。

表访问上下文

表访问上下文是指允许表上存在冲突操作的作用域。只要发生下列情况，就会创建表访问上下文:

- 顶层语句发出 SQL 语句。
- 调用 UDF 或方法。
- 从触发器、动态复合语句、SQL 函数或 SQL 方法调用过程。

例如，当应用程序调用存储过程时，CALL 是顶层语句，因此会获取表访问上下文。如果存储过程执行 UPDATE，那么 UPDATE 也是顶层语句（因为已将存储过程作为顶层语句来调用），因此会获取表访问上下文。如果 UPDATE 调用 UDF，那么 UDF 会获取单独的表访问上下文，且 UDF 中的 SQL 语句不是顶层语句。

在存取表以进行读/写后，会保护此表，防止进行访问的顶层语句发生冲突。可以从不同的顶层语句读/写，或从例程（从不同的顶层语句调用）读/写此表。

适用下列规则:

1. 在表访问上下文中，可以读/写给定表，而不会导致冲突。
2. 如果正在表访问上下文中读取表，那么其他上下文也可以读取此表。但是，如果任何其他上下文尝试写入此表，那么将发生冲突。
3. 如果正在表访问上下文中写入表，那么任何其他上下文都不能读/写此表，因此不会导致冲突。

如果发生冲突，那么会将错误 (SQLCODE -746, SQLSTATE 57053) 返回给导致冲突的语句。

下列是表读/写冲突示例:

假设应用程序发出语句:

```
UPDATE t1 SET c1 = udf1(c2)
```

UDF1 包含语句:

```
DECLARE cur1 CURSOR FOR SELECT c1, c2 FROM t1
OPEN cur1
```

这将导致冲突，因为违反规则 3。只能通过重新设计应用程序或 UDF 来解决这种冲突。

下列不会导致冲突:

假设应用程序发出语句:

```
DECLARE cur2 CURSOR FOR SELECT udf2(c1) FROM t2
OPEN cur2
FETCH cur2 INTO :hv
UPDATE t2 SET c2 = 5
```

UDF2 包含语句:

```
DECLARE cur3 CURSOR FOR SELECT c1, c2 FROM t2
OPEN cur3
FETCH cur3 INTO :hv
```

借助游标，UDF2 可以读取表 T2，因为两个表访问上下文可以读取同一表。应用程序可以更新 T2（即使 UDF2 正在读取表亦如此），因为在有别于更新的应用程序级别语句中调用了 UDF2。

第 3 章 外部例程

外部例程是一些使用数据库服务器的文件系统中，位于数据库外部的编程语言应用程序来实现其逻辑的例程。

通过在例程的 CREATE 语句中指定 EXTERNAL 子句来声明例程与外部代码应用程序的关联。

您可以创建外部过程、外部函数以及外部方法。虽然它们全都可以使用外部编程语言来实现，但是每种例程功能类型都具有不同的功能。在决定实现外部例程前，必须先通过阅读主题“外部例程概述”来了解何谓外部例程、外部例程的实现和使用方式。在了解这些信息后，您可以在相关链接所指向的主题中了解外部例程的更多信息，以就何时以及如何在数据库环境中使用外部例程作出有见识的决策。

外部例程功能

外部例程提供大多数公共例程功能以及 SQL 例程所不支持的附加功能。

外部例程的独特功能如下所示:

访问驻留在数据库外部的文件、数据和应用程序

外部例程可以访问和处理驻留在数据库本身外部的数据或文件。它们还可以调用驻留在数据库外部的应用程序。例如，数据、文件或应用程序可以驻留在数据库服务器文件系统或可用的网络中。

各种外部例程参数样式选项

可以使用选择的参数样式以编程语言实现外部例程。虽然所选编程语言可能有首选的参数样式，但有时也有选项供您选择。某些参数样式支持通过名为 dbinfo 的结构与例程传递附加的数据库和例程属性信息，此结构在例程逻辑中可能非常有用。

通过暂存区在外部函数调用之间保存状态

外部用户定义的函数支持为一组值在函数调用之间保存状态。此任务通过名称 scratchpad 的结构完成。这对于返回聚集值的函数以及需要初始设置逻辑（例如初始化缓冲区）的函数而言特别有用。

调用类型标识各个外部函数调用

将针对一组值多次调用外部用户定义的函数。每次调用都由可以在函数逻辑中引用的调用类型值标识。例如，对于函数的第一次调用、数据访存调用和最终调用都由特殊的调用类型。由于特定的逻辑可以与特定的调用类型相关联，因此调用类型非常有用。

外部函数和方法功能

外部函数和外部方法支持那些对于一组给定输入数据可能被调用多次并生成一组输出值的函数。

要了解更多有关外部函数和方法的功能的信息，请参阅下列主题:

- 第 52 页的『外部标量函数』

- 第 53 页的『外部标量函数和方法处理模型』
- 第 54 页的『外部表函数』
- 第 54 页的『外部表函数处理模型』
- 第 55 页的『Java 表函数执行模型』
- 第 56 页的『外部函数和方法的暂存区』
- 第 59 页的『32 位和 64 位操作系统上的暂存区』

这些都是外部函数和方法的独特功能，并不适用于 方法和 SQL 方法。

外部标量函数

外部标量函数是一些使用外部编程语言来实现其逻辑的标量函数。

可以开发并使用这些函数来扩展现有 SQL 函数的集合，并且这些函数的调用方式与 DB2 内置函数（例如 LENGTH 和 COUNT）的调用方式相同。即，可以在 SQL 语句中表达式有效的任何位置引用这些函数。

可以在 DB2 数据库服务器上执行外部标量函数逻辑，但是，与内置或用户定义 SQL 标量函数不同的是，外部函数的逻辑可以访问数据库服务器文件系统、执行系统调用或访问网络。

外部标量函数可以读取 SQL 数据，但无法修改 SQL 数据。

可以针对函数的单一引用重复调用外部标量函数，并且外部标量函数可以使用暂存区（内存缓冲区）在这些调用之间维护状态。如果函数需要一些初始但开销很大的设置逻辑，那么此方法的功能很强大。可以在第一次调用时，使用暂存区来存储一些可供标量函数的后续调用访问或更新的值，以完成设置逻辑。

外部标量函数的功能

- 可以在 SQL 语句中支持表达式的任何位置，将外部标量函数作为 SQL 语句的一部分来引用。
- 可直接通过调用 SQL 语句来使用标量函数的输出。
- 对于用户定义的外部标量函数，可以使用暂存区在函数迭代调用之间维护状态。
- 在谓词中使用时可以提供性能优势，因为它们是在服务器上执行。如果可以将函数应用于服务器上的候选行，那么该函数通常不考虑此行，就将其传输至客户机，从而减少必须从服务器传递客户机的数据量。

限制

- 无法在标量函数中执行事务管理。即，无法在标量函数中发出 COMMIT 或 ROLLBACK。
- 无法返回结果集。
- 标量函数可以根据输入集来返回单一标量值。
- 外部标量函数无法用于单一调用。这样的设计是针对函数的单一引用以及给定的输入集，根据输入调用函数一次，并返回单一标量值。在第一次调用时，标量函数可以执行一些设置工作，或存储一些可以在后续调用中访问的信息。SQL 标量函数更适合于需要单一调用的功能。

- 在单一分区数据库中，外部标量函数可以包含 SQL 语句。这些语句可以从表中读取数据，但无法修改表中的数据。如果数据库具有多个分区，那么外部标量函数中不能具有 SQL 语句。SQL 标量函数可以包含读取或修改数据的 SQL 语句。

常见用法

- 扩展 DB2 内置函数集。
- 在 SQL 语句中执行 SQL 无法以本机方式执行的逻辑。
- 在 SQL 语句中封装通常作为子查询来复用的标量查询。例如，给定邮政编码，在表中搜索可找到该邮政编码的城市。

支持的语言

- C
- C++
- Java
- OLE
- .NET 公共语言运行时语言

注:

1. 用于创建聚集函数的功能是受限的。这些函数又称为列函数，可接收一组类似值（数据列）并返回单一答案。只有在以内置聚集函数作为源时，才能创建用户定义的聚集函数。例如，如果存在定义了基本类型 INTEGER 的单值类型 SHOESIZE，那么可以将函数 AVG(SHOESIZE) 定义为以现有内置聚集函数 AVG(INTEGER) 作为源的聚集函数。
2. 您也可以创建返回行的函数。这些函数称为行函数，且只能用作结构化类型的变换函数。行函数的输出是单一行。

外部标量函数和方法处理模型

定义了 FINAL CALL 规范的方法和标量 UDF 的处理模型如下所示:

FIRST 调用

这是 NORMAL 调用的特殊情况，标识为 FIRST 的目的是使函数能够执行任何初始处理。将对自变量进行求值，并将结果传递至函数。正常情况下，此函数对于此调用将返回一个值，但也可能返回错误，在出错情况下，将不会执行 NORMAL 或 FINAL 调用。如果 FIRST 调用返回了错误，那么方法或 UDF 在返回前必须执行清理工作，这是因为，将不会执行 FINAL 调用。

NORMAL 调用

这些是对函数进行的第二次直至倒数第二次调用，这由语句的数据和逻辑指定。对于每次 NORMAL 调用，期望函数在对自变量进行求值和传递后返回一个值。如果 NORMAL 调用返回了错误，那么将不会执行进一步的 NORMAL 调用，但将执行 FINAL 调用。

FINAL 调用

这是在执行语句结束处理时（或关闭游标时）进行的特殊调用，进行此调用的条件是 FIRST 调用成功。执行 FINAL 调用时，不会传递任何自变量值。会作出此调用以便函数可以清除任何资源。在执行此调用时，函数不会返回值，但可能会返回错误。

对于未使用 FINAL CALL 定义的方法或标量 UDF，将只对函数进行 NORMAL 调用，这通常会对每次调用返回一个值。如果 NORMAL 调用返回了错误，或者语句遇到另一个错误，那么不会对该函数进行其他调用。

注: 此模型描述方法和标量 UDF 的普通错误处理。在发生系统故障或通信问题时，无法进行此错误处理模型所指示的调用。例如，对于 FENCED UDF，如果 `db2udf` 受保护进程不知何故提前终止，那么 DB2 无法执行所指示的调用。

外部表函数

用户定义的表函数将表传递至引用该表的 SQL。

表 UDF 引用只有在 SELECT 语句的 FROM 子句才有效。使用表函数时，请遵循下列事项:

- 即使表函数传递表，在 DB2 数据库系统和 UDF 之间的物理接口也是一次一行。会对表函数发出五种调用: OPEN、FETCH、CLOSE、FIRST 以及 FINAL。是否存在 FIRST 和 FINAL 调用取决于您如何定义 UDF。可用于标量函数的同一*调用类型*机制也可用来区分这些调用。
- 表函数的 CREATE FUNCTION 语句的 RETURNS 子句中所定义的每个结果列不一定都返回。CREATE FUNCTION 的 DBINFO 关键字以及对应的 dbinfo 自变量会启用优化，这样就只需返回特定表函数引用所需的那些列。
- 所返回的各个列值在格式上符合标量函数所返回的值。
- 表函数的 CREATE FUNCTION 语句具有 CARDINALITY 规范。此规范使定义者能够通知 DB2 优化器大致的结果大小，以便优化器可以作出更好的决策来确定何时引用函数。

  不论将什么内容指定为表函数的 CARDINALITY，在编写具有无限基数的函数（即，只要执行 FETCH 调用，函数总是会返回一行）时，必须格外小心。在许多情况下，DB2 数据库系统期望表末尾条件作为其查询处理中的触媒。使用 GROUP BY 或 ORDER BY 就是这样的示例。DB2 数据库系统无法构成聚集组（除非达到表末尾）且无法排序（除非具有所有数据）。因此，如果将永不返回表末尾条件 (SQL-state value '02000') 的表函数与 GROUP BY 或 ORDER BY 子句搭配使用，那么会导致无限处理循环。

外部表函数处理模型

定义了 FINAL CALL 规范的表 UDF 的处理模型如下所示:

FIRST 调用

在执行第一个 OPEN 调用之前发出此调用，其目的是使函数能够执行任何初始处理。在此调用之前清除暂存区。将对自变量进行求值，并将结果传递至函数。此函数不会返回行。如果此函数返回错误，那么不对此函数作出进一步调用。

OPEN 调用

作出此调用以使函数能够执行特定于扫描的特殊 OPEN 处理。不会在调用之前清除暂存区（如果存在）。会对自变量进行求值，并传递结果。在执行 OPEN 调用时，此函数不会返回行。如果函数从 OPEN 调用返回错误，那么不会作出任何 FETCH 或 CLOSE 调用，但仍会在语句结束时作出 FINAL 调用。

FETCH 调用

继续作出 FETCH 调用，除非函数返回指示表结束的 SQLSTATE 值。UDF 就

是在这些调用上发起并返回数据行。可以将自变量值传递至函数，但这些值指向执行 OPEN 调用时所传递的值。因此，自变量值可能不是当前值，不可靠。如果确实需要在表函数的两次调用之间维护当前值，请使用暂存区。函数可能会在执行 FETCH 调用时返回错误，但仍将发出 CLOSE 调用。

CLOSE 调用

在扫描或语句结束时作出此调用（只要成功执行了 OPEN 调用）。任何自变量值都不会是当前值。函数可能会传回错误。

FINAL 调用

在语句结束时作出 FINAL 调用（只要成功执行了 FIRST 调用）。会作出此调用以便函数可以清除任何资源。在执行此调用时，函数不会返回值，但可能会返回错误。

对于未定义 FINAL CALL 的表 UDF，只会对函数作出 OPEN、FETCH 以及 CLOSE 调用。在每个 OPEN 调用前，会清除暂存区（如果存在）。

在检查涉及连接或子查询（其中，表函数访问是“内部”访问）的方案时，可以查看定义了 FINAL CALL 的表 UDF 和定义了 NO FINAL CALL 的表 UDF 之间的差别。例如，在下列类似语句中:

```
SELECT x,y,z,... FROM table_1 as A,
  TABLE(table_func_1(A.col1,...)) as B
  WHERE ...
```

在此情况下，优化器会扫描 table_func_1 以获取 table_1 的每一行。这是因为使用 table_1 的 col1 值（传递至 table_func_1）来定义表函数扫描。

对于 NO FINAL CALL 表 UDF，会针对 table_1 的每一行重复 OPEN、FETCH、FETCH、...、CLOSE 调用序列。请注意，每个 OPEN 调用都将获取全新的暂存区。因为表函数不知道在每次扫描结束时是否有更多扫描，所以它必须在 CLOSE 处理期间执行彻底清除。如果必须重复重要的一次性打开处理，那么这样做可能效率不高。

FINAL CALL 表 UDF，提供一次性 FIRST 调用和一次性 FINAL 调用。使用这些调用在表函数的所有扫描之间分摊初始化开销和终止成本。如前所述，会对外部表的每一行作出 OPEN、FETCH、FETCH、...、CLOSE 调用，但是表函数知道它将获取 FINAL 调用，因此它不需要在执行其 CLOSE 调用时清除所有内容（以及在执行后续 OPEN 时重新分配）。另请注意，不会在扫描之间清除暂存区，这主要是因为表函数资源将跨越扫描。

以管理两种其他调用类型为代价，表 UDF 可以取得高于这些连接和子查询方案的效率。决定是否将表函数定义为 FINAL CALL 取决于其期望用途。

Java 表函数执行模型

对于使用 Java 编写且使用 PARAMETER STYLE DB2GENERAL 的表函数，必须了解 DB2 数据库系统在处理给定语句的每个时刻所发生的情况。

下表针对典型表函数详细说明了此信息。阐述了 NO FINAL CALL 和 FINAL CALL 情况（在这两种情况下都采用 SCRATCHPAD）。

扫描时间点	NO FINAL CALL LANGUAGE JAVA SCRATCHPAD	FINAL CALL LANGUAGE JAVA SCRATCHPAD
在第一次 OPEN 表函数之前	• 没有调用。	• 调用类构造函数（意味着新的暂存区）。使用 FIRST 调用来调用 UDF。 • 构造函数会初始化类和暂存区变量。方法连接至 Web 服务器。
在每次 OPEN 表函数时	• 调用类构造函数（意味着新的暂存区）。使用 OPEN 调用来调用 UDF 方法。 • 构造函数会初始化类和暂存区变量。方法连接至 Web 服务器，并扫描 Web 数据。	• 使用 OPEN 调用来打开 UDF 方法。 • 方法会扫描它所需的任何 Web 数据。（可能可以避免在 CLOSE 重定位之后重新打开，取决于暂存区中所保存的内容。）
在每次 FETCH 表函数的新数据行时	• 使用 FETCH 调用来调用 UDF。 • 方法访存并返回下一数据行或 EOT。	• 使用 FETCH 调用来调用 UDF。 • 方法访存并返回新的数据行或 EOT。
在每次 CLOSE 表函数时	• 使用 CLOSE 调用来调用 UDF 方法。会调用 close() 方法（如果类存在此方法）。 • 方法会关闭它的 Web 扫描，并断开与 Web 服务器的连接。close() 不需要执行任何操作。	• 使用 CLOSE 调用来调用 UDF 方法。 • 方法可重定位至扫描顶部，或关闭扫描。它可以将任何状态保存至暂存区，该暂存区会持续保存。
在最终 CLOSE 表函数之后	• 没有调用。	• 使用 FINAL 调用来调用 UDF。会调用 close() 方法（如果类存在此方法）。 • 方法会断开与 Web 服务器的连接。close() 方法不需要执行任何操作。

注:

1. 术语“UDF 方法”是指实现 UDF 的 Java 类方法。在 CREATE FUNCTION 语句的 EXTERNAL NAME 子句中标识此方法。

外部函数和方法的暂存区

*暂存区*使用户定义的函数或方法能够在两次调用之间保存其状态。

例如，在下列两种情况下，在两次调用之间保存状态很有益处:

1. 函数或方法是否正确取决于保存状态。

   此类函数或方法的示例是一个简单计数器函数，该计数器会在第一次调用时增加“1”，然后在后续每次调用时将结果增加 1。在某些情况下，可以使用此类函数来计算 SELECT 结果的行数:

   ```
   SELECT counter(), a, b+c, ...
     FROM tablex
     WHERE ...
   ```

该函数需要一个位置来存放两次调用之间该计数器的当前值，在位置中，将保证值对于后续调用是相同的。随后在每次调用时，会增加此值并将其作为函数结果来返回。

此类型的例程是 NOT DETERMINISTIC。它的输出并不只取决于它的 SQL 自变量值。

2. 可通过执行某些初始化动作来改进性能的函数或方法。

   此类函数或方法（可能是文档应用程序的一部分）的示例是 *match* 函数，如果给定的文档包含给定的字符串，那么返回“Y”；否则，返回“N”：

```
SELECT docid, doctitle, docauthor
  FROM docs
  WHERE match('myocardial infarction', docid) = 'Y'
```

   此语句返回包含第一个自变量所表示的特定文本字符串值的所有文档。*match* 的可能用途如下：

   - 仅第一次调用时。

     在 DB2 数据库系统外部维护的文档应用程序中，检索所有包含字符串“myocardial infarction”的文档标识列表。此检索过程的开销很大，因此函数只会执行此过程一次，并将列表保存在便于后续调用的某个位置。

   - 每次调用时。

     使用第一次调用期间保存的文档标识列表，来查看列表中是否包含作为第二个自变量来传递的文档标识。

   此类型的例程是 DETERMINISTIC。它的应答只取决于它的输入自变量值。此处显示的是函数的性能（而非正确性），取决于能否在两次调用之间保存信息。

可通过在 CREATE 语句中指定 SCRATCHPAD 来满足这两种需求：

```
CREATE FUNCTION counter()
  RETURNS int ... SCRATCHPAD;

CREATE FUNCTION match(varchar(200), char(15))
  RETURNS char(1) ... SCRATCHPAD 10000;
```

SCRATCHPAD 关键字告知 DB2 数据库系统分配和维护例程的暂存区。暂存区的缺省大小是 100 字节，但是您可以确定暂存区的大小（以字节计）。*match* 示例是 10000 字节长。在第一次调用之前，DB2 数据库系统会将暂存区初始化为二进制零。如果正在为表函数定义暂存区，且表函数也定义了 NO FINAL CALL（缺省值），那么 DB2 数据库系统会在每个 OPEN 调用之前刷新暂存区。如果您指定表函数选项 FINAL CALL，那么 DB2 数据库系统不会在暂存区初始化之后，检查或更改暂存区的内容。对于定义了暂存区的标量函数，DB2 数据库系统也不会在暂存区初始化之后，检查或更改暂存区的内容。暂存区的指针会在每次调用时传递至例程，并且 DB2 数据库系统会将例程的状态信息保存在暂存区中。

因此，对于 *counter* 示例，会将返回的最后一个值保留在暂存区中。如果暂存区足够大，那么 *match* 示例会将文档列表保留在暂存区中；否则，它会为列表分配内存，并将所获取内存的地址保留在暂存区中。暂存区的长度可变：长度是在例程的 CREATE 语句中定义。

暂存区只适用于语句中对例程的个别引用。如果对语句中的例程进行了多次引用，那么每个引用都有自己的暂存区，因此无法使用暂存区在两个引用之间通信。暂存区只适用于单一 DB2 代理程序（代理程序是对语句的所有方面进行处理的 DB2 实体）。未提供“全局暂存区”来协调暂存区信息在代理程序之间的共享。这对于 DB2 数据库系统建立多个代理程序来处理语句（在单分区或多分区数据库中）这一情况特别重要。在这些情况下，即使可能只对语句中的例程进行单一引用，也可能有多个代理程序正在处理工作，且每个代理程序都会有自己的暂存区。在多分区数据库（其中，引用 UDF 的语句正在处理多个分区上的数据，且正在调用每个分区上的 UDF）中，暂存区将只适用于单一分区。因此，在执行 UDF 的每个分区上，都会有一个暂存区。

如果函数的正确执行取决于对函数的每个引用都具有单一暂存区，那么会将函数注册为 DISALLOW PARALLEL。这将强制函数在单一分区上运行，从而保证对函数的每个引用将只有单一暂存区。

因为已知 UDF 或方法可能需要系统资源，所以可以在 UDF 或方法中定义 FINAL CALL 关键字。此关键字会指示 DB2 数据库系统在处理语句结尾时调用 UDF 或方法，以便 UDF 或方法可以释放其系统资源。例程释放它所获取的任何资源非常重要；在重复调用该语句的环境中，即使很小的泄漏也会变成很大的泄漏，而大泄漏会导致 DB2 数据库崩溃。

因为暂存区具有固定大小，所以 UDF 或方法本身可以包含内存分配，并因此可以利用最终调用来释放内存。例如，前面的 *match* 函数无法预测与给定文本字符串匹配的文档数。因此，下列 *match* 定义更合适:

```
CREATE FUNCTION match(varchar(200), char(15))
  RETURNS char(1) ... SCRATCHPAD 10000 FINAL CALL;
```

对于使用暂存区且在子查询中引用的 UDF 或方法，DB2 数据库系统可能会执行最终调用（如果以此方式指定 UDF 或方法），并在两次调用子查询之间刷新暂存区。如果曾在子查询中使用 UDF 或方法，那么可通过在 UDF 或方法中定义 FINAL CALL 并使用调用类型自变量，或通过始终检查暂存区的二*进制零* 状态来作出保护，以免出现此可能性。

如果的确指定 FINAL CALL，请注意您的 UDF 或方法会接收类型为 FIRST 的调用。这可用来获取和初始化某些持久资源。

下列是使用 Java 编写的简单 UDF 示例，此 UDF 使用暂存区来计算列中条目的平方之和。本示例接受列，并返回包含条目（从列顶部到当前行条目）的平方之累积和的列:

```
CREATE FUNCTION SumOfSquares(INTEGER)
RETURNS INTEGER
EXTERNAL NAME 'UDFsrv!SumOfSquares'
DETERMINISTIC
NO EXTERNAL ACTION
FENCED
NOT NULL CALL
LANGUAGE JAVA
PARAMETER STYLE DB2GENERAL
NO SQL
SCRATCHPAD 10
FINAL CALL
DISALLOW PARALLEL
NO DBINFO@
```

```
// Sum Of Squares using Scratchpad UDF
public void SumOfSquares(int inColumn,
                         int outSum)
throws Exception
{
  int sum = 0;
  byte[] scratchpad = getScratchpad();

  // variables to read from SCRATCHPAD area
  ByteArrayInputStream byteArrayIn = new ByteArrayInputStream(scratchpad);
  DataInputStream dataIn = new DataInputStream(byteArrayIn);

  // variables to write into SCRATCHPAD area
  byte[] byteArrayCounter;
  int i;
  ByteArrayOutputStream byteArrayOut = new ByteArrayOutputStream(10);
  DataOutputStream dataOut = new DataOutputStream(byteArrayOut);

  switch(getCallType())
  {
    case SQLUDF_FIRST_CALL:
      // initialize data
   sum = (inColumn * inColumn);
      // save data into SCRATCHPAD area
      dataOut.writeInt(sum);
      byteArrayCounter = byteArrayOut.toByteArray();
      for(i = 0; i < byteArrayCounter.length; i++)
      {
        scratchpad[i] = byteArrayCounter[i];
      }
      setScratchpad(scratchpad);
   break;
    case SQLUDF_NORMAL_CALL:
      // read data from SCRATCHPAD area
      sum = dataIn.readInt();
      // work with data
      sum = sum + (inColumn * inColumn);
      // save data into SCRATCHPAD area
      dataOut.writeInt(sum);
      byteArrayCounter = byteArrayOut.toByteArray();
      for(i = 0; i < byteArrayCounter.length; i++)
      {
        scratchpad[i] = byteArrayCounter[i];
      }
      setScratchpad(scratchpad);
break;
  }
  //set the output value
  set(2, sum);
} // SumOfSquares UDF
```

请注意，有一个内置 DB2 函数可以执行 SumOfSquares UDF 所执行的任务。选择本示例是为了演示暂存区的用法。

32 位和 64 位操作系统上的暂存区

要使 UDF 或方法能够在 32 位和 64 位操作系统之间移植，您必须特别注意创建和使用包含 64 位值的暂存区的方法。建议不要针对包含一个或多个 64 位值的 scratchpad 结构声明显式长度变量，例如 64 位指针或 sqlint64 BIGINT 变量。

下列是暂存区的样本结构声明:

```
struct sql_scratchpad
{
   sqlint32 length;
   char data[100];
};
```

在例程针对暂存区定义自己的结构时，有两个选项:

1. 重新定义整个暂存区 sql_scratchpad，在此情况下，它需要包含一个显式长度字段。例如:

```
struct sql_spad
{
  sqlint32 length;
  sqlint32 int_var;
  sqlint64 bigint_var;
};
void SQL_API_FN routine( ..., struct sql_spad* scratchpad, ... )
{
  /* Use scratchpad */
}
```

2. 只重新定义暂存区 sql_scratchpad 的数据部分，在此情况下，不需要长度字段。

```
struct spaddata
{
  sqlint32 int_var;
  sqlint64 bigint_var;
};
void SQL_API_FN routine( ..., struct sql_scratchpad* spad, ... )
{
  struct spaddata* scratchpad = (struct spaddata*)spad->data;
  /* Use scratchpad */
}
```

因为应用程序无法更改暂存区的长度字段中的值，所以编写如示例 1 中所示的例程没有明显的益处。示例 2 也可以在具有不同字大小的计算机之间进行移植，因此是编写例程的首选方法。

外部例程中的 SQL

使用诸如 C、Visual Basic、C# 和 Java 之类的外部编程语言编写的所有例程都可以包含 SQL。

例程（存储过程和 UDF）的 CREATE 语句，或方法的 CREATE TYPE 语句可以包含一个定义例程或方法 SQL 访问级别的子句。根据例程中所包含 SQL 的性质，您必须选择适用的子句:

NO SQL

例程根本不包含任何 SQL

CONTAINS SQL

包含 SQL，但是既不读取也不写入数据（例如: SET SPECIAL REGISTER）。

READS SQL DATA

包含可以读取表的 SQL（SELECT 和 VALUES 语句），但不会修改表数据。

MODIFIES SQL DATA

包含可以更新表的 SQL: 直接更新用户表（INSERT、UPDATE 以及 DELETE 语句）或隐式地更新 DB2 的目录表（DDL 语句）。此子句只适用于存储过程和 SQL 主体表函数。

DB2 数据库系统将在执行时验证例程是否超出其定义级别。例如，如果定义为 CONTAINS SQL 的例程尝试对表执行 SELECT，那么会因为尝试读取 SQL 数据，而导致错误 (SQLCODE -579, SQLSTATE 38004)。另请注意，嵌套的例程引用必须处于相同 SQL 级别，或处于限制性更强且包含该引用的 SQL 级别。例如，修改 SQL 数据的例程可以调用读取 SQL 数据的例程，但只能读取 SQL 数据的例程（使用 READS SQL DATA 子句进行定义）无法调用修改 SQL 数据的例程。

例程会在调用应用程序的数据库连接范围内执行 SQL 语句。例程无法建立它自己的连接，也无法重置调用应用程序的连接 (SQLCODE -751, SQLSTATE 38003)。

只有定义为 MODIFIES SQL DATA 的存储过程才能发出 COMMIT 和 ROLLBACK 语句。其他类型的例程（UDF 和方法）无法发出 COMMIT 或 ROLLBACK (SQLCODE -751, SQLSTATE 38003)。即使定义为 MODIFIES SQL DATA 的存储过程尝试对事务执行 COMMIT 或 ROLLBACK，也建议从调用应用程序执行 COMMIT 或 ROLLBACK，这样就不会意外地落实更改。如果已从应用程序（与数据库建立了 2 类连接）调用存储过程，那么存储过程无法发出 COMMIT 或 ROLLBACK 语句。

此外，只有定义为 MODIFIES SQL DATA 的存储过程才能建立它们自己的保存点，以及在保存点中回滚它们自己的工作。其他类型的例程（UDF 和方法）无法建立它们自己的保存点。当存储过程完成时，不会释放在存储过程中创建的保存点。应用程序将能够回滚该保存点。类似地，存储过程可以回滚应用程序中所定义的保存点。当例程返回时，DB2 数据库系统将隐式地释放该例程所建立的任何保存点。

例程通过将 SQLSTATE 值指定给 DB2 数据库系统传递至该例程的 sqlstate 自变量，可以通知 DB2 它是否已成功执行。某些参数样式（PARAMETER STYLE JAVA、GENERAL 以及 GENERAL WITH NULLS）不支持交换 SQLSTATE 值。

如果在处理例程所发出的 SQL 时，DB2 数据库系统遇到错误，那么会将该错误返回给例程（对任何应用程序也是执行此操作）。对于一般用户错误，例程有机会执行备用操作或更正操作。例如，如果例程尝试对表执行 INSERT，但获取“键重复”错误 (SQLCODE -813)，那么它可以改为对现有表行执行 UPDATE 操作。

但是，可能会发生某些较严重的错误，导致 DB2 数据库系统无法以正常方式继续执行。这些错误示例包括死锁、数据库分区失败或用户中断。其中一些错误会一直传播到调用应用程序。其他与工作单元相关的严重错误，会一直传播到回退时下列两项中先出现的一项：(a) 应用程序，或 (b) 有权发出事务控制语句（COMMIT 或 ROLLBACK）的存储过程。

如果在执行例程所发出的 SQL 期间发生其中一个错误，那么会将该错误返回给例程，但是 DB2 数据库系统会记住已发生严重错误。此外，在此情况下，DB2 数据库系统将自动使此例程及任何调用例程所发出的任何后续 SQL 失败 (SQLCODE -20139, SQLSTATE 51038)。此情况的唯一例外是，错误只回退到有权发出事务控制语句的最外层存储过程。在此情况下，此存储过程可以继续发出 SQL。

例程可以发出静态和动态 SQL，在任一情况下，如果使用嵌入式 SQL，那么必须预编译和绑定例程。对于静态 SQL，在预编译/绑定进程中使用的信息，与使用嵌入式 SQL 的任何客户机应用程序中所使用的信息相同。对于动态 SQL，您可以使用 **DYNAMICRULES** precompile/bind 选项来控制嵌入式动态 SQL 的当前模式和当前认证标识。对于例程和应用程序，此行为会有所不同。

会遵循针对例程程序包或语句定义的隔离级别。这会导致例程以限制性较强或较弱的隔离级别（相对于调用应用程序）来运行。调用其隔离级别不太受限（相对于调用语句）的例程时，必须考虑此重要事项。例如，如果从可重复读的应用程序调用游标稳定性函数，那么 UDF 可以表现出不可重复读的特征。

例程对专用寄存器值所作的更改不会影响调用应用程序或例程。例程从调用程序继承可更新的专用寄存器。不会将对可更新专用寄存器所作的更改传递回至调用程序。不可更新的专用寄存器会获取它们的缺省值。有关可更新及不可更新专用寄存器的更多详细信息，请参阅相关主题『专用寄存器』。

例程可以使用客户机应用程序所使用的方法，对游标执行 OPEN、FETCH 以及 CLOSE 操作。多次调用（例如，在递归情况下）同一函数可分别获取其自己的游标实例。UDF 和方法必须关闭其游标，然后调用语句才能完成，否则将发生错误 (SQLCODE -472, SQLSTATE 24517)。对 UDF 或方法所作的最终调用是关闭任何仍处于打开状态的游标的好时机。存储过程中任何未在完成前关闭的已打开游标，会作为结果集返回至客户机应用程序或调用例程。

不会自动将传递至例程的自变量视为主变量。这意味着，如果例程要在其 SQL 中将参数用作主变量，那么必须声明其自己的主变量并将参数值复制到此主变量。

注: 必须在 **DATETIME** 选项设为 ISO 的情况下，预编译和绑定嵌入式 SQL 例程。

外部例程的参数样式

外部例程实现必须符合特定约定才能交换例程参数值。这些约定称为*参数样式*。

通过指定 PARAMETER STYLE 子句来创建例程时，会指定外部例程参数样式。参数样式描述规范和顺序（将参数值传递至外部例程实现的顺序）的特征。它们也指定在将任何其他值传递至外部例程实现时该怎么办。例如，某些参数样式会作出下列指定：对于每个例程参数值，会将其他单独的 null 指示符值传递至例程实现，以提供参数可空性的信息（否则，无法使用本机编程语言数据类型来轻松确定）。

下表提供可用参数样式列表、支持每种参数样式的例程实现、支持每种参数样式的功能例程类型以及参数样式的描述:

表 8. 参数样式

参数样式	支持的语言	支持的例程类型	描述
SQL [1]	• C/C++ • OLE • .NET 公共语言运行时语言 • COBOL [2]	• UDF • 存储过程 • 方法	除调用期间传递的参数之外，会按下列顺序将下列自变量传递至例程: • 在 CREATE 语句中声明的每个参数或结果的 null 指示符。 • 要返回给 DB2 数据库系统的 SQLSTATE。 • 例程的限定名。 • 例程的特定名称。 • 要返回给 DB2 数据库系统的 SQL 诊断字符串。 可以按下列顺序将下列自变量传递至例程，取决于选项（在 CREATE 语句中指定）和例程类型: • 暂存区的缓冲区。 • 例程的调用类型。 • dbinfo 结构（包含数据库的信息）。
DB2SQL [1]	• C/C++ • OLE • .NET 公共语言运行时语言 • COBOL	• 存储过程	除调用期间传递的参数之外，会按下列顺序将下列自变量传递至存储过程: • 包含 CALL 语句上每个参数的 null 指示符的向量。 • 要返回给 DB2 数据库系统的 SQLSTATE。 • 存储过程的限定名。 • 存储过程的特定名称。 • 要返回给 DB2 数据库系统的 SQL 诊断字符串。 如果在 CREATE PROCEDURE 语句中指定 DBINFO 子句，那么会将 dbinfo 结构（包含数据库的信息）传递至存储过程。
JAVA	• Java	• UDF • 存储过程	PARAMETER STYLE JAVA 例程使用符合 Java 语言和 SQLJ 例程规范的参数传递约定。 对于存储过程，会将 INOUT 和 OUT 参数作为单一项目数组来传递，以加快值的返回。除 IN、OUT 以及 INOUT 参数之外，存储过程的 Java 方法特征符包含每个结果集（在 CREATE PROCEDURE 语句的 DYNAMIC RESULT SETS 子句中指定）的参数（类型为 ResultSet[]）。 对于 PARAMETER STYLE JAVA UDF 和方法，不会传递例程调用中指定的对象的其他自变量。 PARAMETER STYLE JAVA 例程不支持 DBINFO 或 PROGRAM TYPE 子句。对于 UDF，如果未将任何结构化数据类型指定为参数，且未将任何结构化类型、CLOB、DBCLOB 或 BLOB 数据类型指定为返回类型 (SQLSTATE 429B8)，那么只能指定 PARAMETER STYLE JAVA。此外，PARAMETER STYLE JAVA UDF 不支持表函数、调用类型或暂存区。
DB2GENERAL	• Java	• UDF • 存储过程 • 方法	此类型的例程将使用定义为与 Java 方法一起使用的参数传递约定。除非要开发表 UDF 或具有暂存区的 UDF，或者需要访问 dbinfo 结构，否则，建议您使用 PARAMETER STYLE JAVA。 对于 PARAMETER STYLE DB2GENERAL 例程，不会传递例程调用中指定的对象的其他自变量。

表 8. 参数样式 (续)

参数样式	支持的语言	支持的例程类型	描述
GENERAL	• C/C++ • .NET 公共语言运行时语言 • COBOL	• 存储过程	PARAMETER STYLE GENERAL 存储过程会在调用应用程序或例程中从 CALL 语句接收参数。如果在 CREATE PROCEDURE 语句中指定 DBINFO 子句，那么会将 dbinfo 结构（包含数据库的信息）传递至存储过程。 GENERAL 等价于 DB2 z/OS 版的 SIMPLE 存储过程。
GENERAL WITH NULLS	• C/C++ • .NET 公共语言运行时语言 • COBOL	• 存储过程	PARAMETER STYLE GENERAL WITH NULLS 存储过程会在调用应用程序或例程中从 CALL 语句接收参数。此外，还附带一个包含 CALL 语句上每个参数的 null 指示符的向量。如果在 CREATE PROCEDURE 语句中指定 DBINFO 子句，那么会将 dbinfo 结构（包含数据库的信息）传递至存储过程。 GENERAL WITH NULLS 等价于 DB2 z/OS 版的 SIMPLE WITH NULLS 存储过程。

注:

1. 对于 UDF 和方法，PARAMETER STYLE SQL 等价于 PARAMETER STYLE DB2SQL。
2. 只能使用 COBOL 来开发存储过程。
3. 不支持 .NET 公共语言运行时运行时。

PROGRAM TYPE MAIN 或 PROGRAM TYPE SUB 过程中的参数处理

过程可以接受主例程或子例程样式中的参数。当您在 CREATE PROCEDURE 语句中注册过程时，会确定这一点。

PROGRAM TYPE SUB 的 C 或 C++ 过程接受自变量的方法，与 C 或 C++ 子例程接受自变量的方法相同。会将参数作为指针来传递。例如，下列 C 过程特征符会接受类型为 INTEGER、SMALLINT 和 CHAR(3) 的参数:

```
int storproc (sqlint32 *arg1, sqlint16 *arg2, char *arg3)
```

Java 过程只接受自变量作为子例程。将 IN 参数作为简单自变量来传递。将 OUT 和 INOUT 参数作为具有单一元素的数组来传递。下列参数样式 Java 过程特征符会接受类型为 INTEGER 的 IN 参数、类型为 SMALLINT 的 OUT 参数以及类型为 CHAR(3) 的 INOUT 参数:

```
int storproc (int arg1, short arg2[], String arg[])
```

要编写 C 过程来接受诸如 C 程序中主函数之类的自变量，请在 CREATE PROCEDURE 语句中指定 PROGRAM TYPE MAIN。必须编写 PROGRAM TYPE MAIN 过程以符合下列规范:

- 过程通过两个自变量来接受参数:
  - 参数计数器变量；例如，*argc*
  - 参数的数组指针；例如，*char **argv*
- 必须将过程构建为共享库

在 PROGRAM TYPE MAIN 过程中，DB2 数据库系统会将 *argv* 数组中第一个元素 (*argv[0]*) 的值设为过程的名称。*argv* 数组的剩余元素对应于过程的 PARAMETER STYLE 所定义的参数。例如，下列嵌入式 C 过程会将一个 IN 参数作为 *argv[1]* 来传递，并将两个 OUT 参数作为 *argv[2]* 和 *argv[3]* 来返回。

PROGRAM TYPE MAIN 示例的 CREATE PROCEDURE 语句如下所示:

```
CREATE PROCEDURE MAIN_EXAMPLE (IN job CHAR(8),
   OUT salary DOUBLE, OUT errorcode INTEGER)
   DYNAMIC RESULT SETS 0
   LANGUAGE C
   PARAMETER STYLE GENERAL
   NO DBINFO
   FENCED
   READS SQL DATA
   PROGRAM TYPE MAIN
   EXTERNAL NAME 'spserver!mainexample'
```

过程的下列代码会将 *argv[1]* 的值复制到 CHAR(8) 主变量 *injob*，然后将 DOUBLE 主变量 *outsalary* 的值复制到 *argv[2]* 并将 SQLCODE 作为 *argv[3]* 来返回:

```
SQL_API_RC SQL_API_FN main_example (int argc, char **argv)
{
  EXEC SQL INCLUDE SQLCA;

  EXEC SQL BEGIN DECLARE SECTION;
    char injob[9];
    double outsalary;
  EXEC SQL END DECLARE SECTION;

  /* argv[0] contains the procedure name. */
  /* Parameters start at argv[1]         */
  strcpy (injob, (char *)argv[1]);

  EXEC SQL SELECT AVG(salary)
    INTO :outsalary
    FROM employee
    WHERE job = :injob;

  memcpy ((double *)argv[2], (double *)&outsalary, sizeof(double));

  memcpy ((sqlint32 *)argv[3], (sqlint32 *)&SQLCODE, sizeof(sqlint32));

  return (0);

} /* end main_example function */
```

受支持的例程编程语言

通常，通过支持在数据库服务器上执行应用程序功能，使用例程来改进数据库管理系统的整体性能。这些工作所取得的性能增益量在某种程度上受限于选择用来编写例程的语言。

在使用特定语言实现例程前，您应该考虑下列一些问题:

- 使用特定语言和环境来开发例程的可用技能。
- 语言实现代码的可靠性和安全性。
- 使用特定语言编写的例程的可伸缩性。

为了帮助评估前面的标准，下面列出了各种支持语言的某些特征:

SQL

- SQL 例程的速度要快于 Java 例程，并且性能大致相当于 NOT FENCED C/C++ 例程。
- SQL 例程完全使用 SQL 来编写，并且可以包括 SQL 过程语言 (SQL PL) 的元素，而 SQL PL 包含可用来实现逻辑的 SQL 控制语句。
- DB2 数据库系统视 SQL 例程为“安全”例程，因为它们只包含 SQL 语句。SQL 例程始终直接在数据库引擎中运行，这使 SQL 例程具有良好的性能和可伸缩性。

C/C++

- C/C++ 嵌入式 SQL 和 DB2 CLI 例程的速度都快于 Java 例程。以 NOT FENCED 方式运行时，它们的性能大致相当于 SQL 例程。
- C/C++ 例程容易出错。建议将 C/C++ 例程注册为 FENCED NOT THREADSAFE，因为使用这些语言编写的例程极可能造成内存毁坏，从而中断 DB2 数据库引擎的运作。以 FENCED NOT THREADSAFE 方式运行时，虽然较安全，但是会引发性能开销。

  有关评估和缓解将 C/C++ 例程注册为 NOT FENCED 或 FENCED THREADSAFE 的风险的信息，请参阅『例程安全性注意事项』主题。
- 缺省情况下，C/C++ 例程以 FENCED NOT THREADSAFE 方式运行，以防止损坏其他例程的执行。因此，在数据库服务器上并发执行的 C/C++ 例程都会有一个 db2fmp 进程。这会导致某些系统出现可伸缩性问题。

Java

- Java 例程的速度要慢于 C/C++ 或 SQL 例程。
- Java 例程比 C/C++ 例程安全，因为危险操作是由 JVM 进行处理。这样 Java 例程就很难损坏正在同一进程中运行的另一个例程，从而提高了可靠性。

  注: 只有不允许从 Java 例程执行 Java 本机接口（JNI）调用，才能避免潜在的危险操作。如果需要从 Java 例程调用 C/C++ 代码，那么您可以通过调用单独编目的 C/C++ 例程来实现此目的。
- 在 FENCED THREADSAFE 方式（缺省值）下运行时，Java 例程具有良好的可伸缩性。所有 FENCED Java 例程都共享几个 JVM（如果特定 **db2fmp** 进程的 Java 堆接近耗竭，那么系统上可能正在使用多个 JVM）。
- 当前不支持 NOT FENCED Java 例程。将调用定义为 NOT FENCED 的 Java 例程，如同它定义为 FENCED THREADSAFE 一样。

.NET 公共语言运行时语言

- .NET 公共语言运行时 (CLR) 例程编译为可由 .NET Framework 的 CLR 进行解释的中间语言 (IL) 字节码。可以使用任何 .NET Framework 支持语言来编写 CLR 例程的源代码。
- 使用 .NET CLR 例程，用户可以灵活地使用所选 .NET CLR 支持编程语言来编写代码。
- 可以从子组合件（从不同的 .NET 编程语言源代码进行编译）构建 CLR 组合件，这允许用户复用和集成使用各种语言编写的代码模块。
- 只能将 CLR 例程创建为 FENCED NOT THREADSAFE 例程。这可以将毁坏引擎的可能性降至最小，但也还意味着这些例程无法受益于使用 NOT FENCED 例程时所具有的性能机会。

OLE

- 可以使用 Visual C++、Visual Basic 以及其他受 OLE 支持的语言来实现 OLE 例程。
- OLE 自动化例程的速度取决于用来实现这些例程的语言。通常，它们的速度要慢于非 OLE C/C++ 例程。
- OLE 例程只能以 FENCED NOT THREADSAFE 方式运行。这可以将毁坏引擎的可能性降至最小。这也意味着 OLE 自动化例程没有良好的可伸缩性。

OLE DB

- OLE DB 只能用来定义表函数。
- OLE DB 表函数连接至外部 OLE DB 数据源。
- 根据 OLE DB 提供者，OLE DB 表函数的速度通常快于 Java 表函数，但慢于 C/C++ 或 SQL 主体表函数。但是，查询（在其中调用了函数）中的某些谓词可能在 OLE DB 提供者处进行求值，因此可减少 DB2 数据库系统必须处理的行数。这常常可以改进性能。
- OLE DB 例程只能以 FENCED NOT THREADSAFE 方式运行。这可以将毁坏引擎的可能性降至最小。这也意味着 OLE DB 自动化表函数没有良好的可伸缩性。

对支持用于开发外部例程的 API 和编程语言的比较

在开始实现外部例程之前，必须考虑各种支持的外部例程应用程序编程接口 (API) 和编程语言的特征与限制。这将确保您从开始就选择正确的实现，以及您所需的例程功能是可用的。

表 9. 外部例程 API 和编程语言比较

API 和编程语言	功能支持	性能	安全性	可伸缩性	限制
SQL（包括 SQL PL）	• SQL 是高级语言，易于学习和使用，使实现能快速进行。 • SQL 过程语言 (SQL PL) 元素允许将控制流逻辑用于 SQL 操作和查询。	• 非常好。 • SQL 例程的性能高于 Java 例程。 • SQL 例程的性能相当于使用 NOT FENCED 子句来创建的 C 和 C++ 外部例程。	• 非常安全。 • SQL 过程始终与数据库管理器在同一内存中运行。这对应于缺省情况下使用关键字 NOT FENCED 来创建的例程。	• 高度可伸缩。	• 无法访问数据库服务器文件系统。 • 无法调用位于数据库外部的应用程序。

表 9. 外部例程 API 和编程语言比较 (续)

API 和编程语言	功能支持	性能	安全性	可伸缩性	限制
嵌入式 SQL（包括 C 和 C++）	• 低级但功能强大的编程语言。	• 非常好。 • C 和 C++ 例程的性能高于 Java 例程。 • 使用 NOT FENCED 子句创建的 C 和 C++ 例程的性能相当于 SQL 例程。	• C 和 C++ 例程容易发生编程错误。 • 程序员必须精通 C，才能避免犯一些常见的内存和指针操作错误，此类错误会使例程实现更冗长、更耗时。 • 应该使用 FENCED 子句和 NOT THREADSAFE 子句来创建 C 和 C++ 例程，以避免在运行时例程发生异常的情况下，损坏数据库管理器。这些是缺省子句。使用这些子句会对性能造成一定的负面影响，但是可确保安全执行。请参阅例程安全性。	• 使用 FENCED 和 NOT THREADSAFE 子句来创建 C 和 C++ 例程时，会降低可伸缩性。在数据库管理器进程之外的单独 **db2fmp** 进程中运行这些例程。每个并发执行的例程都需要一个 **db2fmp** 进程。	• 多种支持的参数传递样式会造成混淆。用户应该尽可能地使用参数样式 SQL。
嵌入式 SQL（COBOL）	• 高级编程语言适合于开发业务应用程序（通常面向文件）。 • 过去广泛用于生产业务应用程序，虽然其普及面现正在缩小。 • COBOL 不包含指针支持，是线性迭代编程语言。	• COBOL 例程的性能低于使用任何其他外部例程实现选项来创建的例程。	• 目前未提供任何信息。	• 目前未提供任何信息。	• 您可以在 64 位 DB2 实例中创建和调用 32 位 COBOL 过程实例，但是这些例程的性能低于 64 位 DB2 实例中 64 位 COBOL 过程。

表 9. 外部例程 API 和编程语言比较 (续)

API 和编程语言	功能支持	性能	安全性	可伸缩性	限制
JDBC（Java）和 SQLJ（Java）	• 高级面向对象程序设计语言，适合于开发独立应用程序、applet 以及 servlet。 • Java 对象和数据类型可方便建立数据库连接、执行 SQL 语句以及操作数据。	• Java 例程的性能低于 C 和 C++ 例程或 SQL 例程。	• Java 例程较之 C 和 C++ 例程安全，因为是由 Java 虚拟机（JVM）处理对危险操作的控制。这将提高可靠性，让一个 Java 例程的代码很难损害正在同一进程中运行的另一个例程。	• 良好的可伸缩性 • 使用 FENCED THREADSAFE 子句（缺省子句）创建的 Java 例程具有良好的可伸缩性。所有 FENCED Java 例程将共享几个 JVM。如果特定 db2fmp 进程的 Java 堆接近耗竭，那么系统上可能正在使用多个 JVM。	• 只有不允许从 Java 例程执行 Java 本机接口（JNI）调用，才能避免潜在的危险操作。
.NET 公共语言运行时支持的语言（包括 C#、Visual Basic 以及其他）	• Microsoft .NET 受管代码模型的一部分。 • 会将源代码编译为可由 Microsoft .NET Framework 公共语言运行时进行解释的中间语言（IL）字节码。 • 可以从子组合件（从不同的 .NET 编程语言源代码进行编译）构建 CLR 组合件，这允许用户复用和集成使用各种语言编写的代码模块。	• 只能使用 FENCED NOT THREADSAFE 子句来创建 CLR 例程，才能将运行时中断数据库管理器的可能性降至最小。这会对性能造成一定的负面影响 • 使用缺省子句值可将运行时中断数据库管理器的可能性降至最小；但是，因为 CLR 例程必须以 FENCED 方式运行，所以它们的性能可能稍小于可以指定为以 NOT FENCED 方式来运行的外部例程。	• 只能使用 FENCED NOT THREADSAFE 子句来创建 CLR 例程。如此才能让 CLR 例程安全运行，因为它们将在数据库管理器外部的单独 **db2fmp** 进程中运行。	• 未提供任何信息。	• 请参阅“.NET CLR 例程限制”主题。

表 9. 外部例程 API 和编程语言比较 (续)

API 和编程语言	功能支持	性能	安全性	可伸缩性	限制
• OLE	• 可以使用 Visual C++、Visual Basic 以及其他受 OLE 支持的语言来实现 OLE 例程。	• OLE 自动化例程的速度取决于用来实现这些例程的语言。通常，它们的速度要慢于非 OLE C/C++ 例程。 • OLE 例程只能以 FENCED NOT THREADSAFE 方式运行，因此 OLE 自动化例程没有良好的可伸缩性。	• 未提供任何信息。	• 未提供任何信息。	• 未提供任何信息。
• OLE DB	• 可以使用 OLE DB 来创建用户定义的表函数。 • OLE DB 函数连接至外部 OLE DB 数据源。	• OLE DB 函数的性能取决于 OLE DB 提供者，但是，通常 OLE DB 函数的性能高于逻辑上等价的 Java 函数，但低于逻辑上等价的 C、C++ 或 SQL 函数。但是，查询（在其中调用了函数）中的某些谓词可能在 OLE DB 提供者处进行求值，因此减少了 DB2 数据库系统必须处理的行数，这常常可以改进性能。	• 未提供任何信息。	• 未提供任何信息。	• OLE DB 只能用来创建用户定义的表函数。

开发例程时的性能注意事项

开发例程的一个最显著益处不是扩展客户机应用程序，而是性能。

选择例程实现的方法时，请考虑下列性能影响。

NOT FENCED 方式

NOT FENCED 例程与数据库管理器在同一进程中运行。通常，以 NOT FENCED 方式运行例程，较之以 FENCED 方式运行例程具有更高的性能，因为 FENCED 例程是在引擎地址空间外部的特殊 DB2 进程中运行。

虽然以 NOT FENCED 方式运行例程时可以期望改进的例程性能，但用户代码可能会意外地或恶意地毁坏数据库或损坏数据库控制结构。只有在您需要将性

能益处发挥至极致，且认为例程安全时，才应该使用 NOT FENCED 例程。有关评估和缓解将 C/C++ 例程注册为 NOT FENCED 的风险的信息，请参阅『例程安全性注意事项』主题。如果例程不够安全，不能在数据库管理器的进程中运行，请在注册例程时使用 FENCED 子句。为了限制创建和运行可能不安全的代码，DB2 数据库系统要求用户具有特权 CREATE_NOT_FENCED_ROUTINE 才能创建 NOT FENCED 例程。

如果在您运行 NOT FENCED 例程时发生异常终止，且例程已注册为 NO SQL，那么数据库管理器将尝试适当的恢复操作。但是，对于未定义为 NO SQL 的例程，数据库管理器将失败。

如果 NOT FENCED 例程使用 GRAPHIC 或 DBCLOB 数据，那么必须使用 WCHARTYPE NOCONVERT 选项预编译 NOT FENCED 例程。

FENCED THREADSAFE 方式

FENCED THREADSAFE 例程与其他例程在同一进程中运行。更具体地说，非 Java 例程会共享一个进程，而 Java 例程会共享另一个进程（和使用其他语言编写的例程是分离的）。此分离可保护 Java 例程，使其不受其他语言编写且可能更易出错的例程影响。此外，Java 例程的进程还包含一个 JVM，此 JVM 会引发高昂的内存成本，因此不为其他例程类型所使用。FENCED THREADSAFE 例程的多个调用会共享资源，因此所引发的系统开销小于 FENCED NOT THREADSAFE 例程（每个 FENCED NOT THREADSAFE 例程都在其自己的专用进程中运行）。

如果您认为例程足够安全，可以与其他例程在同一例程中运行，请在注册该例程时使用 THREADSAFE 子句。和 NOT FENCED 例程一样，有关评估和缓解将 C/C++ 例程注册为 FENCED THREADSAFE 的风险的信息可在『例程安全性注意事项』主题中找到。

如果 FENCED THREADSAFE 例程会异常结束，那么将仅终止正在运行此例程的线程。进程中的其他例程会继续运行。但是，导致此线程异常结束的故障会对进程中的其他例程线程产生负面影响，使这些线程进入陷阱、挂起或具有损坏的数据。在一个线程异常结束后，进程就不再用于新的例程调用。在所有活动用户完成其在此进程中的作业后，会终止此进程。

当您注册 Java 例程时，除非另有声明，否则会将这些例程视为 THREADSAFE。缺省情况下，所有其他 LANGUAGE 类型都是 NOT THREADSAFE。不能将使用 LANGUAGE OLE 和 OLE DB 的例程指定为 THREADSAFE。

NOT FENCED 例程必须为 THREADSAFE。无法将例程注册为 NOT FENCED NOT THREADSAFE (SQLCODE -104)。

UNIX 上的用户可通过查找 `db2fmp (Java)` 或 `db2fmp (C)` 来查看其 Java 和 C THREADSAFE 进程。

FENCED NOT THREADSAFE 方式

每个 FENCED NOT THREADSAFE 例程都在其自己的专用进程中运行。如果正在运行许多例程，那么会对数据库系统性能产生不利影响。如果例程不够安全，不能与其他例程在同一进程中运行，请在注册该例程时使用 NOT THREADSAFE 子句。

在 UNIX 上，NOT THREADSAFE 进程对于合用的 NOT THREADSAFE db2fmp 显示为 db2fmp（pid）（其中 pid 是使用 FENCED 方式进程的代理程序的进程标识）或 db2fmp（空闲）。

Java 例程

如果打算运行存在大量内存需求的 Java 例程，建议您将该例程注册为 FENCED NOT THREADSAFE。对于 FENCED THREADSAFE Java 例程调用，DB2 数据库系统会尝试选择其 Java 堆大小足以运行例程的线程 Java FENCED 方式进程。如果未能将大型堆使用者限制于其自己的进程，那么会导致多线程 Java db2fmp 进程发生“Java 堆不足”错误。如果 Java 例程不属于此类别，那么 FENCED 例程以线程安全方式运行时将实现更高的性能（在该方式下，例程可以共享几个 JVM）。

当前不支持 NOT FENCED Java 例程。将调用定义为 NOT FENCED 的 Java 例程，如同它定义为 FENCED THREADSAFE 一样。

C/C++ 例程

C 或 C++ 例程通常比 Java 例程要快，但更易出现错误、内存毁坏以及崩溃。由于这些原因，执行内存操作的能力使 C 或 C++ 例程成为 THREADSAFE 或 NOT FENCED 方式注册的风险候选者。这些风险可通过遵循安全例程的编程实践来缓解（请参阅『例程安全性注意事项』主题），以及通过彻底测试您的例程来缓解。

SQL 例程

SQL 例程（特别是 SQL 过程）通常也比 Java 例程要快，且性能往往与 C 例程相当。SQL 例程始终以 NOT FENCED 方式运行，提供远非外部例程可比的性能益处。如果包含复杂逻辑，那么以 C 编写的 UDF 的运行速度要快于以 SQL 编写的 UDF。如果逻辑很简单，那么 SQL UDF 将可以与任何外部 UDF 相比。

暂存区 暂存区是可以分配给 UDF 和方法的内存块。暂存区只适用于 SQL 语句中对例程的个别引用。如果对语句中的例程进行了多次引用，那么每个引用都有自己的暂存区。暂存区使 UDF 或方法能够在两次调用之间保存其状态。

对于具有复杂初始化的 UDF 和方法，您可以使用暂存区来存储第一次调用中所需的任何值，以在后续的所有调用中使用。其他 UDF 和方法的逻辑可能也要求在两次调用之间保存中间值。

使用 VARCHAR 参数取代 CHAR 参数

您可以在例程定义中使用 VARCHAR 参数取代 CHAR 参数来改进例程的性能。使用 VARCHAR 数据类型取代 CHAR 数据类型可以防止 DB2 数据库系统在传递参数之前使用空格来填充参数，并且可以缩短通过网络传输参数所需的时间量。

例如，如果您的客户机应用程序将字符串“A SHORT STRING”传递至期望 CHAR(200) 参数的例程，那么 DB2 数据库系统必须使用 186 个空格来填充该参数，以 null 结束字符串，然后通过网络将整个由 200 个字符组成的字符串和一个 null 终止符发送至例程。

相比之下，将同一字符串“A SHORT STRING”传递至期望 VARCHAR(200) 参数的例程，会导致 DB2 数据库系统只需通过网络传递一个由 14 个字符组成的字符串和一个 null 终止符。

例程安全性注意事项

开发和部署例程向您提供一个机会来极大地改进数据库应用程序的性能和效率。但是，如果数据库管理员未正确地管理例程部署，那么可能存在安全性风险。

下列部分描述安全性风险及可用来减缓这些风险的方法。安全性风险后面的部分描述如何安全地部署安全性未知的例程。

安全性风险

NOT FENCED 例程可以访问数据库管理器资源

NOT FENCED 例程与数据库管理器在同一进程中运行。由于它们与数据库引擎非常接近，因此 NOT FENCED 例程可能会意外地或恶意地毁坏数据库管理器的共享内存，或损坏数据库控制结构。任一形式的损坏都会导致数据库管理器失败。NOT FENCED 例程也可能会毁坏数据库及其表。

要确保数据库管理器及其数据库的完整性，您必须彻底地屏蔽您打算注册为 NOT FENCED 的例程。这些例程必须经过完整测试、调试，且不会显示任何意外的负面效应。在检查例程时，请密切注意内存管理和静态变量的使用。当代码错误地管理内存或错误地使用静态变量时，极有可能会发生毁坏。这些问题在 Java 和 .NET 编程语言之外的语言中非常普遍。

需要 CREATE_NOT_FENCED_ROUTINE 权限才能注册 NOT FENCED 例程。授予 CREATE_NOT_FENCED_ROUTINE 权限时，请注意接收方可能会取得对数据库管理器及其所有资源的不受限访问权。

FENCED THREADSAFE 例程可以访问其他 FENCED THREADSAFE 例程所使用的内存

FENCED THREADSAFE 例程作为共享进程中的线程来运行。其中的每个例程都可以读取同一进程中其他例程线程所使用的内存。因此，一个线程例程可从线程进程中的其他例程收集敏感数据。共享单一进程固有的另一个风险是一个内存管理存在故障的例程线程可能会毁坏其他例程线程，或导致整个线程进程崩溃。

要确保其他 FENCED THREADSAFE 例程的完整性，您必须彻底地屏蔽您打算注册为 FENCED THREADSAFE 的例程。这些例程必须经过完整测试、调试，且不会显示任何意外的负面效应。在检查例程时，请密切注意内存管理和静态变量的使用。这是最可能发生毁坏的情况，特别是在非 Java 语言中。

需要 CREATE_EXTERNAL_ROUTINE 权限才能登录 FENCED THREADSAFE 例程。授予 CREATE_EXTERNAL_ROUTINE 权限时，请注意接收方可能会监视或毁坏其他 FENCED THREADSAFE 例程的内存。

FENCED 进程所有者对数据库服务器的写访问权可能会导致数据库管理器毁坏

由 **db2icrt**（创建实例）或 **db2iupdt**（更新实例）系统命令定义运行 FENCED 进程所使用的用户标识。此用户标识不能对例程库和类的存储目录具有写访问权（在 UNIX 环境中，此目录是 `sqllib/function`；在 Windows 环境中，此目录是 `sqllib\function`）。此用户标识也不能对数据库服务器上的任何数据库、操作系统或其他重要文件和目录具有读或写访问权。

如果 FENCED 进程所有者确实对数据库服务器上的各种重要资源具有写访问权，那么系统可能会发生毁坏。例如，数据库管理员会将从未知源接收到的例程注册为 FENCED NOT THREADSAFE，并认为可通过将例程隔离在其自己的进程中来防止任何可能的危害。但是，拥有 FENCED 进程的用户标识对 `sqllib/function` 目录具有写访问权。如果用户调用此例程，但此例程对于他们

是未知的，那么它会使用注册为 NOT FENCED 的例程实体备用版本来覆盖 sqllib/function 中的库。第二个例程对整个数据库管理器具有不受限的访问权，并且会因此分发数据库表中的敏感信息，毁坏数据库，收集认证信息，或使数据库管理器崩溃。

确保拥有 FENCED 进程的用户标识对数据库服务器上的重要文件或目录不具有写访问权（特别是 sqllib/function 和数据库数据目录）。

例程库和类的脆弱性

如果未控制对例程库和类的存储目录的访问权，那么可能会删除或覆盖例程库和类。如先前项中所讨论的那样，使用恶意（或编程质量不高）的例程替换 NOT FENCED 例程主体会严重地破坏数据库服务器及其资源的稳定性、完整性以及保密性。

要保护例程的完整性，您必须管理对包含例程库和类的目录的访问权。确保尽可能少的用户可以访问此目录及其文件。分配对此目录的写访问权时，请注意此特权可以向用户标识的所有者，提供对数据库管理器及其所有资源的不受限访问权。

部署可能不安全的例程

如果您偶尔从未知源获取例程，请确保准确地了解其用途，然后再构建、注册以及调用该例程。建议将该例程注册为 FENCED 和 NOT THREADSAFE，除非您已彻底测试该例程并且它未显示任何意外的负面效果。

如果您需要部署一个不符合安全例程标准的例程，请将该例程注册为 FENCED 和 NOT THREADSAFE。FENCED 和 NOT THREADSAFE 例程必须执行下列操作，才能确保维护数据库完整性:

- 在单独的 DB2 进程中运行（不与任何其他例程共享此进程）。如果它们异常终止，那么不会影响数据库管理器。
- 使用未被数据库所使用的内存。意外的赋值错误将不会影响数据库管理器。

例程代码页注意事项

会将字符数据传递至代码页中的外部例程（由建立例程时使用的 PARAMETER CCSID 选项隐式指定）。类似地，数据库假设从例程输出的字符串使用由 PARAMETER CCSID 选项隐式指定的代码页。

如果客户机程序（例如使用代码页 C）访问具有不同代码页（例如，代码页 S）的节，而该代码页会使用不同的代码页（例如，代码页 R）来调用例程，那么会出现下列事件:

1. 调用 SQL 语句时，会将输入字符数据从客户机应用程序的代码页 (C) 转换为与该节相关联的代码页 (S)。对于 BLOB 或将用作 FOR BIT DATA 的数据，不会进行转换。
2. 如果例程的代码页与节的代码页不同，那么会在调用例程前将输入字符数据（BLOB 和 FOR BIT DATA 除外）转换为例程的代码页（R）。

   强烈建议您使用调用服务器例程所依据的代码页（R）来预编译、编译以及绑定服务器例程。并非在所有情况下都可以这样做。例如，您可以在 Windows 环境中创建 Unicode 数据库。但是，如果 Windows 环境没有 Unicode 代码页，那么您必须在

Windows 代码页中预编译、编译并绑定创建该例程的应用程序。如果应用程序不含预编译器无法理解的任何特殊定界字符，那么该例程将起作用。

3. 如果例程完成，那么数据库管理器会根据需要将所有输出字符数据，从例程代码页 (R) 转换为节代码页 (S)。如果例程在执行期间抛出错误，那么也会将例程中的 SQLSTATE 和诊断消息，从例程代码页转换为节代码页。BLOB 或 FOR BIT DATA 字符串不会进行转换。
4. 如果语句完成，那么会将输出字符数据，从节代码页 (S) 转换回为客户机应用程序的代码页 (C)。对于 BLOB 或已用作 FOR BIT DATA 的数据，不会进行转换。

通过在 CREATE FUNCTION、CREATE PROCEDURE 以及 CREATE TYPE 语句上使用 DBINFO 选项，会将例程代码页传递至例程。使用此信息，可以编写对代码页敏感的例程，以在许多不同的代码页中操作。

32 位和 64 位应用程序和例程支持

DB2 Database for Linux, UNIX, and Windows 支持在各种平台上开发和部署应用程序和例程，其中包括过程和用户定义的函数 (UDF)。要让应用程序和例程正常工作，必须复审和了解 DB2 数据库 32 位和 64 位支持注意事项。

要开始，最好是先澄清下列几点:

- 32 位硬件平台正在运行 32 位操作系统，而 64 位硬件平台正在运行 64 位操作系统。
- 您可以将 DB2 数据库的 32 位实例安装至 32 位操作系统或 64 位操作系统，但只能将 DB2 实例的 64 位实例安装至 64 位操作系统。
- 32 位应用程序是在 32 位操作系统上构建的应用程序。
- 64 位应用程序是在 64 位操作系统上构建的应用程序。

下表概述了客户机应用程序和例程的 DB2 数据库 32 位和 64 位支持，且作出下列假设:

表 10. 支持在 32 位或 64 位硬件平台上运行 32 位和 64 位应用程序

	32 位硬件 + 操作系统	**64 位硬件 + 操作系统**
32 位应用程序	是	是
64 位应用程序	否	是

下表指示支持从 DB2 客户机应用程序创建与 DB2 数据库服务器的连接。

表 11. 支持从 32 位和 64 位客户机连接至 32 位和 64 位服务器

	32 位服务器	**64 位服务器**
32 位客户机	是	是
64 位应用程序	是	是

表 12. 支持在 32 位或 64 位硬件平台上运行 32 位和 64 位应用程序

	32 位硬件和操作系统	**64 位硬件和操作系统**
32 位应用程序	是	是
64 位应用程序	否	是

下表指示支持从 DB2 客户机应用程序创建与 DB2 数据库服务器的连接。

表 13. 支持从 32 位和 64 位客户机连接至 32 位和 64 位服务器

	32 位服务器	**64 位服务器**
32 位客户机	是	是
64 位应用程序	是	是

表 14. 支持在 32 位和 64 位服务器上运行 FENCED 和 NOT FENCED 过程与 UDF

	32 位服务器	**64 位服务器**
32 位 FENCED 过程或 UDF	是	是[1、2、3]
64 位 FENCED 过程或 UDF	否	是
32 位 NOT FENCED 过程或 UDF	是	否[2]
64 位 NOT FENCED 过程或 UDF	否	是

注:

1. 在 64 位服务器上运行 32 位过程可能很慢。
2. 32 位例程必须创建为 FENCED 和 NOT THREADSAFE，才能在 64 位服务器上工作。
3. 无法在 Linux/IA-64 数据库服务器上调用 32 位例程。

对外部例程的 32 位和 64 位支持

由例程的 CREATE 语句中下列两个子句之一的规范来确定 32 位和 64 位外部例程的值: FENCED 子句或 NOT FENCED 子句。

外部例程的例程主体是使用编程语言来编写，且编译为可以在调用例程时装入和运行的库或类文件。FENCED 或 NOT FENCED 子句的规范确定外部例程是在有别于数据库管理器的 FENCED 环境中运行，还是在数据库管理器所在的寻址空间中运行（通过使用共享内存取代 TCPIP 进行通信，从而取得更好的性能）。缺省情况下，不论选择的其他子句为何，例程始终创建为 FENCED。

下表说明在运行相同操作系统的 32 位和 64 位数据库服务器上，DB2 数据库系统对运行 FENCED 和 NOT FENCED 32 位和 64 位例程的支持。

表 15. 对 32 位和 64 位外部例程的支持

例程的位宽	**32 位服务器**	**64 位服务器**
32 位 FENCED 过程或 UDF	支持	支持[1]
64 位 FENCED 过程或 UDF	不支持[3]	支持
32 位 NOT FENCED 过程或 UDF	支持	支持[1、2]
64 位 NOT FENCED 过程或 UDF	不支持[3]	支持

注:

1. 在 64 位服务器上运行 32 位例程不如在 64 位服务器上运行 64 位例程那样快。

2. 32 位例程必须创建为 FENCED 和 NOT THREADSAFE，才能在 64 位服务器上工作。
3. 不能在 32 位寻址空间中运行 64 位应用程序和例程。

表中需要注意的重要事项是 32 位 NOT FENCED 过程不能在 64 位 DB2 数据库服务器上运行。如果必须将 32 位 NOT FENCED 例程部署至 64 位平台，请在对这些例程进行编目之前，从这些例程的 CREATE 语句中除去 NOT FENCED 子句。

64 位数据库服务器上使用 32 位库的例程的性能

可以在 64 位 DB2 数据库服务器上调用使用 32 位例程库的例程。但是，此调用的性能低于在 64 位服务器上调用 64 位例程。

性能降级的原因是，每次尝试在 64 位服务器上执行 32 位例程之前，会先尝试将其作为 64 位库来调用。如果这样做失败，那么随后会将库作为 32 位库来调用。尝试将 32 位库作为 64 位库来调用失败时，会在 db2diag 日志文件中产生错误消息 (SQLCODE -444)。

Java 类独立于位宽。只有 Java 虚拟机（JVM）才会分类为 32 位或 64 位。DB2 数据库系统仅支持使用位宽与实例（在其中使用 JVM）相同的 JVM。换句话说，在 32 位 DB2 实例中，只能使用 32 位 JVM，而在 64 位 DB2 实例中，只能使用 64 位 JVM。这将确保 Java 例程正常运作，并尽可能将性能发挥至极致。

外部例程中的 XML 数据类型支持

使用下列编程语言编写的外部过程和函数支持数据类型为 XML 的参数和变量:

- C
- C++
- COBOL
- Java
- .NET CLR 语言

外部 OLE 和 OLEDB 例程不支持数据类型为 XML 的参数。

在外部例程代码中表示 XML 数据类型值的方法与表示 CLOB 数据类型的方法相同。

声明数据类型为 XML 的外部例程参数时，将用来在数据库中创建例程的 CREATE PROCEDURE 和 CREATE FUNCTION 语句，必须指定会将 XML 数据类型存储为 CLOB 数据类型。CLOB 值的大小应该接近 XML 参数所表示 XML 文档的大小。

下列 CREATE PROCEDURE 语句显示使用 C 编程语言来实现且具有名为 parm1 的 XML 参数的外部过程的 CREATE PROCEDURE 语句:

```
CREATE PROCEDURE myproc(IN parm1 XML AS CLOB(2M), IN parm2 VARCHAR(32000))
LANGUAGE C
FENCED
PARAMETER STYLE SQL
EXTERNAL NAME 'mylib!myproc';
```

在创建外部 UDF 时适用类似注意事项，如下列示例中所示:

```
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'
```

将 XML 数据作为 IN、OUT 或 INOUT 参数传递至存储过程时，会具体化此 XML 数据。如果使用的是 Java 存储过程，那么可能需要根据 XML 自变量的数量和大小，以及正在并发执行的外部存储过程数来增加堆大小（**java_heap_sz** 配置参数）。

在外部例程代码中访问、设置以及修改 XML 参数和变量值的方法，与在数据库应用程序中执行相应操作的方法相同。

外部例程限制

外部例程适用下列限制，在开发或调试外部例程时应该考虑这些限制。

所有外部例程所适用的限制:

- 无法在外部例程中创建新线程。
- 无法从外部函数或外部方法中调用连接级别 API。
- 外部例程无法从键盘接收输入并将输出显示至标准输出。请不要使用标准输入/输出流。例如:
  - 在外部 Java 例程代码中，请不要发出 System.out.println() 方法。
  - 在外部 C 或 C++ 例程代码中，请不要发出 printf()。
  - 在外部 COBOL 例程代码中，请不要发出 display。

  虽然外部例程无法将数据显示至标准输出，但是它们可以包含将数据写入数据库服务器文件系统上的文件的代码。

  对于在 UNIX 环境中运行的 FENCED 例程，要在其中创建文件的目标目录或文件本身必须具有适当的许可权，如此才能让 `sqllib/adm/.fenced` 文件的所有者创建或写入文件。对于 NOT FENCED 例程，实例所有者必须对在其中打开文件的目录，具有创建和读/写许可权。

  注: DB2 数据库系统不会尝试将例程所执行的任何外部输入或输出与其自己的事务同步。因此，如果 UDF 在事务期间写入文件，并且稍后由某些原因而回退该事务，那么不会尝试发现或撤销对文件所作的写入。
- 无法在外部例程中执行连接相关的语句或命令。此限制适用于下列语句和命令:
  - **BACKUP DATABASE**
  - CONNECT
  - CONNECT TO
  - CONNECT RESET
  - **CREATE DATABASE**
  - **DROP DATABASE**
  - FORWARD RECOVERY

  - **RESTORE DATABASE**
- 不推荐在例程中使用操作系统函数。不会限制使用这些函数，下列情况除外:
  - 不能为外部例程安装用户定义的信号处理程序。如果未遵循此限制，那么可能会导致意外的外部例程运行时失败、数据库异常结束或其他问题。安装信号处理程序也可能会干扰 Java 例程的 JVM 操作。
  - 终止进程的系统调用可能会异常终止其中一个 DB2 数据库系统进程，并导致数据库系统或数据库应用程序失败。

    如果其他系统调用干扰 DB2 数据库管理器的正常操作，那么也可能会导致问题。例如，如果函数尝试从内存卸载包含用户定义的函数的库，那么可能会导致严重问题。在编写和测试包含系统调用的外部例程时，请格外小心。
- 从 DB2 pureScale® for Linux V9.8 FP2 开始，不能在不受保护的例程中使用导致创建新进程的操作系统函数用法。这些函数包括 fork()、popen() 和 system()。使用这些函数可能会干扰 DB2 服务器与集群高速缓存设施之间的通信并导致该例程返回 SQL0430N 错误。
- 外部例程不能包含会终止当前进程的命令。外部例程必须始终在不终止当前进程的情况下，将控制权返回给 DB2 数据库管理器。
- 当数据库处于活动状态时，不能更新外部例程库、类或组合件（特殊情况除外）。如果在 DB2 数据库管理器处于活动状态时需要更新，且不能选择停止和启动实例，请使用不同的实例为例程创建新的库、类或组合件。然后，使用 ALTER 语句来更改外部例程的 EXTERNAL NAME 子句值，以便它引用新库、类或组合件文件的名称。
- 环境变量 **DB2CKPTR** 在外部例程中不可用。会在启动数据库管理器时捕获所有其他名称以“DB2”开头的环境变量，供外部例程使用。
- FENCED 外部例程无法使用某些名称不是以“DB2”开头的环境变量。例如，无法使用 **LIBPATH** 环境变量。但是，NOT FENCED 外部例程可以使用这些变量。
- 外部例程无法使用在启动 DB2 数据库管理器后设置的环境变量值。
- 应该限制在外部例程中使用受保护资源（这些资源一次只能供一个进程访问）。如果使用，请尝试降低在两个外部例程试图访问受保护资源发生死锁的可能性。如果试图访问受保护资源时两个或更多外部例程发生死锁，那么 DB2 数据库管理器将无法检测到或解决这种情况。这将导致外部例程进程挂起。
- 不应该在 DB2 数据库服务器上显式地分配外部例程参数的内存。DB2 数据库管理器会根据例程的 CREATE 语句中的参数声明来自动分配存储器。请不要在外部例程中改变参数的任何存储器指针。如果试图使用本地创建的存储器指针来更改指针，那么可能会导致内存泄漏、数据毁坏或异常结束。
- 请不要在外部例程中使用静态或全局数据。DB2 数据库系统无法保证静态或全局变量所使用的内存在两次外部例程调用之间保留不变。对于 UDF 和方法，您可以使用暂存区来存储两次调用之间使用的值。
- 会缓存所有 SQL 参数值。这意味着会创建值副本并将其传递至外部例程。如果对外部例程的输入参数作出了更改，那么这些更改将不会影响 SQL 值或处理。但是，如果外部例程将超出 CREATE 语句所指定数量的数据写入至输入或输出参数，那么会发生内存毁坏，而且例程会异常结束。
- **LOAD** 实用程序不支持装入含有引用受防护过程的列的表中。如果对此类表发出 **LOAD** 命令，那么将接收到错误消息 SQL1376N。要解决此限制，可以重新定义要取消防护的例程或使用导入实用程序。

仅适用于外部过程的限制

- 从嵌套存储过程中返回结果集时，您可以在多个嵌套级别使用同一名称打开游标。但是，低于版本 8 的应用程序将只能访问打开的第一个结果集。使用不同程序包级别打开的游标不适用此限制。

仅适用于外部函数的限制

- 外部函数无法返回结果集。在完成函数的最终调用时，必须关闭外部函数中打开的所有游标。
- 应该在外部例程返回之前释放外部例程中的动态内存分配。如果未这样做，那么将导致内存泄漏，且 DB2 进程消耗会持续增长，导致数据库系统内存不足。

  对于用户定义的外部函数和外部方法，可以使用暂存区来分配多个函数调用所需的动态内存。如果以此方式使用暂存区，请在 CREATE FUNCTION 或 CREATE METHOD 语句中指定 FINAL CALL 属性。这将确保在例程返回之前释放所分配的内存。

创建外部例程

使用创建包括其他实现的例程的相似方法来创建包括过程和函数的外部例程，但是，需要执行其他几个步骤，因为例程实现需要编写、编译以及部署源代码。

开始之前

- 必须安装 IBM 数据服务器客户机。
- 数据库服务器必须正在运行支持所选实现编程语言编译器和开发软件的操作系统。
- 必须在数据库服务器上安装所选编程语言的必需编译器和运行时支持。
- 有权执行 CREATE PROCEDURE、CREATE FUNCTION 或 CREATE METHOD 语句。

限制

有关与外部例程相关联的限制列表，请参阅:

- 第 78 页的『外部例程限制』

关于此任务

在下列情况下，您不妨选择实现外部例程:

- 您需要在例程中封装复杂逻辑，以访问数据库或在数据库外部执行操作。
- 您要求从下列任何项调用封装的逻辑: 多个应用程序、CLP、另一个例程（过程、函数（UDF）或方法）或触发器。
- 您最擅长使用编程语言（而不是 SQL 和 SQL PL 语句）来编写此逻辑。
- 您需要例程逻辑执行数据库外部操作，例如写入或读取数据库服务器上的文件、运行另一个应用程序或运行无法使用 SQL 和 SQL PL 语句来表示的逻辑。

过程

1. 使用所选编程语言来编写例程逻辑。
   - 有关外部例程、例程功能以及例程功能实现的常规信息，请参阅“先决条件”部分中所引用的主题。

- 使用或导入支持 SQL 语句执行时所需的任何必需头文件。
- 正确地使用映射至 DB2 SQL 数据类型的编程语言数据类型来声明变量和参数。

2. 必须根据所选编程语言的参数样式所需的格式来声明参数。有关参数和原型声明的更多信息，请参阅:
   - 第 62 页的『外部例程的参数样式』
3. 将代码构建到库或类文件。
4. 将库或类文件复制到数据库服务器上的 DB2 *function* 目录。建议将 DB2 例程的相关联组合件或库存储至函数目录。要了解有关函数目录的更多信息，请参阅下列任一语句的 EXTERNAL 子句: CREATE PROCEDURE 或 CREATE FUNCTION。

   如果需要，您可以将组合件复制到服务器上的另一个目录，但是为了成功调用例程，您必须记下组合件的标准路径名，因为下一步需要该标准路径名。
5. 动态或静态地执行适合于例程类型的 SQL 语言 CREATE 语句: CREATE PROCEDURE 或 CREATE FUNCTION。
   - 使用所选 API 或编程语言的适当值来指定 LANGUAGE 子句。示例包括: CLR、C 以及 JAVA。
   - 使用支持的参数样式（在例程代码中实现）的名称来指定 PARAMETER STYLE 子句。
   - 使用库、类或组合件文件（将通过下列其中一个值与例程相关联）的名称来指定 EXTERNAL 子句:
     - 例程库、类或组合件文件的标准路径名。
     - 例程库、类或组合件文件的相对路径名（相对于函数目录）。

     缺省情况下，DB2 将在函数目录中按名称查找库、类或组合件文件，除非在 EXTERNAL 子句中指定库、类或组合件文件的标准或相对路径名。
   - 如果您的例程是过程且它会将一个或多个结果集返回给调用者，请指定具有数值的 DYNAMIC RESULT SETS。
   - 指定描述例程的特征时所需的任何其他子句。

下一步做什么

要调用外部例程，请参阅例程调用

编写例程

开始之前

就编写例程而言，三种例程（过程、UDF 以及方法）具有许多共同点。例如，这三种例程采用其中一些相同的参数样式，支持通过各种客户机接口（嵌入式 SQL、CLI 以及 JDBC）来使用 SQL，以及都可以调用其他例程。最后，下列步骤表示用于编写例程的单一方法。

例程类型具有一些特定的例程功能。例如，结果集特定于存储过程，而暂存区特定于 UDF 和方法。如果您遇到不适用于您所开发的例程类型的步骤，请转至该步骤后面的步骤。

在编写例程之前，您必须决定下列事项:

- 所需的例程类型。

- 将用来编写例程的编程语言。
- 例程中需要 SQL 语句时要使用的接口。

另请参阅有关安全性、库和类管理以及性能注意事项的主题。

过程

要创建例程主体，您必须执行下列操作:

1. *只适用于外部例程*。接受来自调用应用程序或例程的输入参数，并声明输出参数。例程如何接受参数依赖于用来创建例程的参数样式。每种参数样式都定义可传递至例程主体的参数集以及参数的传递顺序。

   例如，下列是 PARAMETER STYLE SQL 中使用 C（使用 sqludf.h）编写的 UDF 主体的特征符:

   ```
   SQL_API_RC SQL_API_FN product ( SQLUDF_DOUBLE *in1,
                                   SQLUDF_DOUBLE *in2,
                                   SQLUDF_DOUBLE *outProduct,
                                   SQLUDF_NULLIND *in1NullInd,
                                   SQLUDF_NULLIND *in2NullInd,
                                   SQLUDF_NULLIND *productNullInd,
                                   SQLUDF_TRAIL_ARGS )
   ```

2. 添加例程要执行的逻辑。您可以在例程主体中采用的某些功能如下所示:
   - 调用其他例程（嵌套）或调用当前例程（递归）。
   - 在定义为具有 SQL（CONTAINS SQL、READS SQL 或 MODIFIES SQL）的例程中，例程可以发出 SQL 语句。例程的注册方式控制可以调用的语句类型。
   - 在外部 UDF 和方法中，使用暂存区来保存两个调用之间的状态。
   - 在 SQL 过程中，使用条件处理程序来确定出现指定条件时 SQL 过程的行为。您可以根据 SQLSTATE 来定义条件。
3. *只适用于存储过程*。返回一个或多个结果集。除与调用应用程序交换的个别参数之外，存储过程还能够返回多个结果集。只有 SQL 例程和 CLI、ODBC、JDBC 以及 SQLJ 例程和客户机才能接受结果集。

结果

除了编写例程之外，您还需要注册该例程，然后才能调用。这是使用与您所开发的例程类型匹配的 CREATE 语句来完成。通常，编写和注册例程的顺序无关紧要。但是，如果例程发出对自身进行引用的 SQL，那么必须先注册例程，然后构建例程。在此情况下，必须已注册例程，才能成功绑定。

调试例程

在生产服务器上部署例程之前，您必须在测试服务器上对其进行充分测试和调试。

关于此任务

这对于需要注册为 NOT FENCED 例程的例程而言尤其重要，这是因为，它们可以对数据库管理器的内存、数据库和数据库控制结构进行不受限制的访问。FENCED THREADSAFE 例程与其他例程共享内存，因此也需要密切注意。

常见例程问题的核对表

要确保例程正确地执行，请检查下列事项:

- 是否已正确地注册该例程。CREATE 语句中提供的参数必须与例程体所处理的自变量匹配。在牢记这一点的情况下，检查下列具体事项:
  - 例程体所使用的自变量的数据类型适合于 CREATE 语句中定义的参数类型。
  - 例程写入输出变量的字节数不超出 CREATE 语句为相应结果定义的字节数。
  - 如果使用相应的 CREATE 选项来注册例程，那么存在 SCRATCHPAD、FINAL CALL 和 DBINFO 的例程自变量。
  - 对于外部例程而言，CREATE 语句中 EXTERNAL NAME 子句的值必须与例程库和入口点匹配（随操作系统不同，是否区分大小写也有所变化）。
  - 对于 C++ 例程，C++ 编译器将对入口点名称应用类型声明。您或者应该在 EXTERNAL NAME 子句中指定已进行类型装饰的名称，或者应该在用户代码中将入口点定义为 `extern "C"`。
  - 调用期间指定的例程名必须与该例程的注册名（由 CREATE 语句定义）匹配。缺省情况下，例程标识将转换为大写。这并不适用于定界标识，这些标识不会转换为大写，因此区分大小写。

    必须将例程放入 CREATE 语句所指定的目录路径，未指定路径时，DB2 数据库系统在缺省情况下将在此路径中查找该例程。对于 UDF、方法和受防护过程，这是 `sqllib/function` (UNIX) 或 `sqllib\function` (Windows)。对于未受防护过程而言，这是 `sqllib/function/unfenced` (UNIX) 或 `sqllib\function\unfenced` (Windows)。
- 已使用正确的调用序列、预编译（对于嵌入式 SQL）、编译和链接选项来构建例程。
- 已将应用程序与数据库绑定（使用 CLI、ODBC 或 JDBC 编写的应用程序除外）。如果例程包含 SQL 并且未使用任何这些接口，那么也需要对其进行绑定。
- 例程准确地将任何错误信息返回到客户机应用程序。
- 如果对例程定义了 FINAL CALL，那么将考虑所有适用的调用类型。
- 释放例程所使用的系统资源。
- 如果您尝试调用例程，但接收到指示您的特权不足以执行此操作的错误 (SQLCODE -551, SQLSTATE 42501)，这可能是因为您对该例程不具有 EXECUTE 特权。具有 SECADM 或 ACCESSCTRL 权限的用户或者任何对该例程具有 EXECUTE WITH GRANT OPTION 的用户可以将此特权授予例程的任何调用者。有关授权和例程的相关主题提供了有关如何有效管理对此特权的使用的详细信息。

例程调试技术

要调试例程，请使用下列技术:

- 开发中心提供了用于调试包含 SQL 主体的过程和 Java 过程的众多调试工具。
- 不可能将诊断数据从例程写至屏幕。如果您打算将诊断数据写入文件，请确保写入可全局访问的目录，例如 `\tmp`。不要写入由数据库管理器或数据库使用的目录。

对于过程而言，一种安全的替代方法是，将诊断数据写入 SQL 表。必须使用 MODIFIES SQL DATA 子句来注册您所测试的过程，这样才能写入 SQL 表。如果您需要现有过程将数据写入 SQL 表或者不再将数据写入 SQL 表，那么必须删除该过程，然后在指定或不指定 MODIFIES SQL DATA 子句的情况下重新注册该过程。在删除并重新注册过程之前，请了解其从属项。

- 您可以通过编写直接调用例程入口点的简单应用程序来调试该例程。有关使用所提供的调试器的信息，请查阅编译器文档。

外部例程库和类管理

要成功开发和调用外部例程，必须正确部署和管理外部例程库和类文件。

如果最初创建外部例程以及部署库和类文件时小心谨慎，那么可以将外部例程库和类文件的管理工作量减至最少。

外部例程管理的主要注意事项如下所示:

- 部署外部例程库和类文件
- 保护外部例程库和类文件的安全
- 解析外部例程库和类
- 修改外部例程库和类文件
- 备份和复原外部例程库和类文件
- 验证所有例程库是否在 `sqllib/function` 目录中并且它们是否在正确的库中。选择包含例程库最终版本的成员。该库与执行 **db2iupdt** 命令的最后一个成员的库是同一个库。

系统管理员、数据库管理员以及数据库应用程序开发者都应该承担责任，确保在执行例程开发和数据库管理任务期间，外部例程库和类文件是安全的且已正确保留。

部署外部例程库和类

部署外部例程库和类是指在根据源代码构建外部例程库和类后，将其复制到数据库服务器。

必须将外部例程库、类或组合件文件复制到数据库服务器上的 DB2 数据库系统的函数目录或其子目录。此位置是建议的外部例程部署位置。要了解有关函数目录的更多信息，请参阅下列任一 SQL 语句的 EXTERNAL 子句的描述: CREATE PROCEDURE 或 CREATE FUNCTION。

您可以将外部例程类、库或组合件复制到服务器上的其他目录位置（取决于用来实现例程的 API 和编程语言），但是，通常不推荐这样做。如果这样做，那么必须专门记录标准路径名并确保此值是与 EXTERNAL NAME 子句搭配使用，才能成功调用例程。

可以使用各种常用的文件传输工具将库和类文件复制到数据库服务器文件系统。可以将 Java 例程从安装了 DB2 客户机的计算机复制到 DB2 数据库服务器（可以使用专门针对此用途而设计的特殊内置过程进行复制）。请参阅有关 Java 例程的主题，以了解更多详细信息。

执行适合于例程类型的 SQL 语言 CREATE 语句（CREATE PROCEDURE 或 CREATE FUNCTION）时，请确保指定相应的子句并特别注意 EXTERNAL NAME 子句。

- 使用所选 API 或编程语言的适当值来指定 LANGUAGE 子句。示例包括: CLR、C 以及 JAVA。
- 使用支持的参数样式（在例程代码中实现）的名称来指定 PARAMETER STYLE 子句。
- 使用库、类或组合件文件（将通过下列其中一个值与例程相关联）的名称来指定 EXTERNAL 子句:
  - 例程库、类或组合件文件的标准路径名。
  - 例程库、类或组合件文件的相对路径名（相对于函数目录）。

缺省情况下，DB2 数据库系统将在函数目录中按名称查找库、类或组合件文件，除非在 EXTERNAL 子句中指定库、类或组合件文件的标准或相对路径名。

外部例程库或类文件的安全性

外部例程库是存储在数据库服务器的文件系统上，而 DB2 数据库管理器不会以任何方式对外部例程库进行备份或保护。要使例程能够继续成功被调用，与例程相关联的库必须继续存在于 CREATE 语句（用来创建例程）的 EXTERNAL 子句所指定的位置。

在创建例程后，请不要移动或删除例程库；否则，会导致例程调用失败。

要防止意外或蓄意删除或替换例程库，您必须限制对数据库服务器上包含例程库的目录的访问权，并限制对例程库文件的访问权。要进行限制，请使用操作系统命令来设置目录和文件许可权。

解析外部例程库和类

DB2 外部例程库解析在 DB2 实例级别执行。这意味着，在包含多个 DB2 数据库的 DB2 实例中，可以在一个数据库中创建外部例程并让那些外部例程使用已用于另一数据库中的例程的外部例程库。

实例级外部例程解析通过允许多个例程定义与单一库相关联来支持代码复用。未以此方式复用外部例程库，而是让数据库服务器的文件系统包含外部例程库的副本时，库名可能会发生冲突。当单一实例中存在多个数据库，并且每个数据库中的例程与它们自己的库副本和例程主体类相关联时，尤其会发生这种情况。如果一个数据库中的例程所使用的库或类的名称与同一实例中另一数据库中的例程所使用的库或类的名称完全相同，那么将发生冲突。

为了最大程度地降低发生这种情况的可能性，建议在实例级函数目录（sqllib/function 目录）中存储例程库的单一副本，并让每个数据库中所有例程定义的 EXTERNAL 子句引用唯一的库。

如果必须创建两个功能不同的同名例程库，那么执行两个附加步骤以便最大程度地降低库名冲突可能性至关重要。

对于 C、C++、COBOL 和 ADO.NET 例程:

可以通过执行下列操作来最大程度地减少或解决库名冲突:

1. 将包含例程主体的库存储在每个数据库的不同目录中。
2. 创建例程时，通过 EXTERNAL NAME 子句值来指定给定的库的完整路径（而不是指定相对路径）。

对于 Java 例程:

无法通过将相应的类文件移入不同的目录来解决类名冲突，这是因为，

CLASSPATH 环境变量将应用于整个实例。在 **CLASSPATH** 中遇到的第一个类就是所使用的类。因此，如果有两个不同的 Java 例程并且它们引用同名的类，那么其中一个例程将使用不正确的类。有两种可能的解决方案：将受影响类重命名，或者为每个数据库创建不同的实例。

修改外部例程库和类文件

在部署现有的外部例程并在生产数据库系统环境中使用该例程之后，可能需要对其逻辑进行修改。您可以对现有例程进行修改，但重要的是仔细地进行修改，以便定义清晰的接管时间点执行更新以及最大程度地降低中断任何并发例程调用的风险。

如果需要更新外部例程库，请不要重新编译该例程并将其重新链接到数据库管理器运行期间使用的当前例程的目标文件（例如 `sqllib/function/foo.a`）。如果当前例程调用正在访问高速缓存的例程进程版本，并且底层库被替换，那么可能会导致例程调用失败。如果有必要在不停止然后重新启动 DB2 数据库管理器的情况下更改例程的主体，请完成下列步骤:

1. 创建使用另一个库或类文件名的新外部例程库。
2. 如果它是嵌入式 SQL 例程，请使用 **BIND** 命令对例程程序包与数据库进行绑定。
3. 使用 ALTER ROUTINE 语句来更改例程定义，以使 EXTERNAL NAME 子句引用更新后的例程库或类。如果要更新的例程主体由多个数据库中编目的例程使用，那么必须对每个受影响的数据库执行本节描述的操作。
4. 要更新已构建到 JAR 文件中的 Java 例程，必须发出 CALL SQLJ.REFRESH_CLASSES() 语句以强制 DB2 数据库管理器装入新类。如果更新 Java 例程类之后未发出 CALL SQLJ.REFRESH_CLASSES() 语句，那么 DB2 数据库系统将继续使用先前的类版本。DB2 数据库系统将在 COMMIT 或 ROLLBACK 发生时刷新类。

一旦例程定义被更新，对该例程进行的所有后续调用都将装入并运行新的外部例程库或类。

备份和复原外部例程库和类文件

执行数据库备份时，不会将外部例程库与其他数据库对象一起备份。类似地，复原数据库时，不会复原外部例程库。

如果数据库备份和复原的目的是为了重新部署数据库，那么必须以此方式将外部例程库文件从原始数据库服务器文件系统复制到目标数据库服务器文件系统，以保留外部例程库的相对路径名。

外部例程库管理和性能

外部例程库管理会影响例程性能，这是因为，DB2 数据库管理器动态地对外部例程库进行高速缓存，以便提高使用例程的性能。

为了获得最佳的外部例程性能，请考虑下列各项:

- 保持每个库中的例程数尽可能地少。使用众多较小的外部例程库比使用几个大库更好。
- 将通常一起调用的例程的例程函数一起分组到源代码中。将此代码编译到外部例程库中时，经常调用的各个例程的入口点之间的距离较近，这使数据库管理器能够提

高更好的高速缓存支持。改善高速缓存支持的原因是，装入一次单一外部例程库并调用该库中的多个外部例程函数有助于提高效率。

对于使用 C 或 C++ 编程语言实现的外部例程而言，装入库的成本只对 C 例程中始终使用的库发生一次。调用一次例程之后，该进程中同一线程的所有后续调用都不需要重新装入该例程的库。

第 4 章 .NET 公共语言运行时 (CLR) 例程

在 DB2 数据库系统中，公共语言运行时 (CLR) 例程是一个通过执行 CREATE PROCEDURE 或 CREATE FUNCTION 语句（将 .NET 组合件作为其外部代码主体来引用）来创建的外部例程。

下列术语在 CLR 例程的上下文中很重要:

.NET Framework
一个 Microsoft 应用程序开发环境，包含 CLR 和 .NET Framework 类库（设计为提供用于开发和集成代码段的一致编程环境）。

公共语言运行时 (CLR)
所有 .NET Framework 应用程序的运行时解释器。

中间语言（IL）
由 .NET Framework CLR 解释的已编译字节码的类型。所有 .NET 兼容语言的源代码都编译为 IL 字节码。

组合件 包含 IL 字节码的文件。这可以是库或可执行文件。

您可以使用任何可编译为 IL 组合件的语言来实现 CLR 例程。这些语言包括但不限于 Managed C++、C#、Visual Basic 以及 J#。

在开发 CLR 例程前，必须了解例程基本原理以及特定于 CLR 例程的特有功能和特征。要了解例程和 CLR 例程的更多信息，请参阅:

- 第 3 页的『使用例程的益处』
- 第 91 页的『.NET CLR 例程中的 SQL 数据类型表示』
- 第 92 页的『.NET CLR 例程中的参数』
- 第 94 页的『从 .NET CLR 过程中返回结果集』
- 第 96 页的『.NET CLR 例程限制』
- 第 105 页的『与 .NET CLR 例程相关的错误』

开发 CLR 例程很容易。有关如何开发 CLR 例程的逐步指示信息以及完整示例，请参阅:

- 第 98 页的『从 DB2 命令窗口创建 .NET CLR 例程』
- 第 107 页的『C# .NET CLR 过程示例』
- 第 133 页的『C# .NET CLR 函数示例』

对使用 .NET CLR 语言进行外部例程开发的支持

要使用 .NET CLR 语言开发以及成功运行外部例程，您将需要使用支持的操作系统、DB2 数据库服务器和客户机版本以及开发软件。

可以使用可由 Microsoft .NET Framework 编译成 IL 组合件的语言来实现 .NET CLR 外部例程。这些语言包括但不限于 Managed C++、C#、Visual Basic 以及 J#。

您可以在下列操作系统上开发 .NET CLR 例程:

- Windows 2000
- Windows XP（32 位和 64 位版本）
- Windows Server 2003（32 位和 64 位版本）
- Windows Server 2008（32 位和 64 位版本）

必须针对 .NET CLR 例程开发安装版本 9 或更高版本的数据服务器客户机。数据库服务器必须正在运行 DB2 版本 9 或更高版本的数据库产品。

必须将 Microsoft .NET Framework 软件的支持版本安装至 DB2 数据库服务器所在的计算机。Microsoft .NET Framework 是单独提供的，或作为 Microsoft .NET Framework 软件开发包的一部分提供。

用于开发 .NET CLR 例程的工具

工具可以使开发 .NET CLR 例程以与 DB2 数据库交互的任务更快速、更容易。

可以在 Microsoft Visual Studio .NET 中使用下列软件提供的图形工具来开发 .NET CLR 例程:

- IBM Database Add-Ins for Microsoft Visual Studio

也可以使用随 DB2 数据库系统提供的下列命令行界面，在 DB2 数据库服务器上开发 .NET CLR 例程:

- DB2 命令行处理器 (DB2 CLP)
- DB2 命令窗口

设计 .NET CLR 例程

设计 .NET CLR 例程时，应该考虑常规外部例程设计注意事项以及 .NET CLR 特定设计注意事项。

.NET 应用程序开发的知识和经验以及对外部例程的一般了解。下列主题可以向您提供一些必备信息。

有关外部例程的功能和使用的更多信息，请参阅:

- 第 18 页的『外部例程实现』

有关 .NET CLR 例程的特征的更多信息，请参阅:

- 第 89 页的第 4 章，『.NET 公共语言运行时 (CLR) 例程』

掌握必备知识之后，设计嵌入式 SQL 例程的主要注意事项是了解 .NET CLR 例程的特有功能和特征:

- 包含组合件，支持在 .NET CLR 例程中执行 SQL 语句（IBM.Data.DB2）
- .NET CLR 例程中的受支持 SQL 数据类型
- .NET CLR 例程的参数
- 返回 .NET CLR 例程的结果集
- .NET CLR 例程的安全性和执行控制方式设置
- .NET CLR 例程限制

- 从 .NET CLR 过程中返回结果集

在了解 .NET CLR 特征后，请参阅第 97 页的『创建 .NET CLR 例程』。

.NET CLR 例程中的 SQL 数据类型表示

.NET CLR 例程可以引用作为例程参数的 SQL 数据类型值、用作 SQL 语句执行一部分的参数值以及用作变量的参数值，但是必须使用相应的 IBM SQL 数据类型值、IBM Data Server Provider for .NET 数据类型值以及 .NET Framework 数据类型值来确保在访问或检索这些值时不会截断或丢失数据。

对于 CREATE PROCEDURE 或 CREATE FUNCTION 语句中用来创建 .NET CLR 例程的例程参数规范，会使用 DB2 SQL 数据类型值。可以为例程参数指定大多数 SQL 数据类型，但是存在一些例外。

必须使用 IBM Data Server Provider for .NET 对象，才能指定要用作需执行的 SQL 语句一部分的参数值。使用 `DB2Parameter` 对象来表示要添加至 `DB2Command` 对象（表示 SQL 语句）的参数。指定参数的数据类型值时，必须使用 `IBM.Data.DB2Types` 名称空间中所提供的 IBM Data Server Provider for .NET 数据类型值。`IBM.Data.DB2Types` 名称空间提供类和结构来表示每种支持的 DB2 SQL 数据类型。

对于可能临时存放 SQL 数据类型值的参数和局部变量，必须使用如 IBM.Data.DB2Types 名称空间中所定义的相应 IBM Data Server Provider for .NET 数据类型。

注: 会将 `dbinfo` 结构作为参数传递至 CLR 函数和过程。也会将 CLR UDF 的暂存区和调用类型作为参数传递至 CLR 例程。有关适合于这些参数的 CLR 数据类型的信息，请参阅相关主题:

- CLR 例程中的参数

下表显示 DB2Type 数据类型、DB2 数据类型、Informix® 数据类型、Microsoft .NET Framework 类型以及 DB2Types 类和结构之间的映射。

类别	DB2Types 类和结构	DB2Type 数据类型	DB2 数据类型	Informix 数据类型	.NET 数据类型
数字	DB2Int16	SmallInt	SMALLINT	BOOLEAN 和 SMALLINT	Int16
数字	DB2Int32	Integer	INT	INTEGER、INT 和 SERIAL	Int32
数字	DB2Int64	BigInt	BIGINT	BIGINT、BIGSERIAL、INT8 和 SERIAL8	Int64
数字	DB2Real 和 DB2Real370	Real	REAL	REAL 和 SMALLFLOAT	Single
数字	DB2Double	Double	DOUBLE PRECISION	DECIMAL（≤31）和 DOUBLE PRECISION	Double
数字	DB2Double	Float	FLOAT	DECIMAL（32）和 FLOAT	Double

1. DB2 .NET 公共语言运行时例程不支持将这些数据类型用作参数。

2. 类型为 DB2Type.Xml 的 DB2ParameterClass.ParameterName 属性可以接受下列类型的变量: String、byte[]、DB2Xml 以及 XmlReader。

3. 这些数据类型只适用于 DB2 z/OS 版。

4. 仅 DB2 z/OS 版版本 9 和更高版本以及 DB2 Linux 版、UNIX 版和 Windows 版版本 9.5 和更高发行版支持此数据类型。

类别	DB2Types 类和结构	DB2Type 数据类型	DB2 数据类型	Informix 数据类型	.NET 数据类型
数字	DB2Decimal	Decimal	DECIMAL	MONEY	Decimal
数字	DB2DecimalFloat	DecimalFloat	DECFLOAT（16|34）[1, 4]		Decimal
数字	DB2Decimal	Numeric	DECIMAL	DECIMAL（≤31）和 NUMERIC	Decimal
日期/时间	DB2Date	Date	DATE	DATETIME（日期精度）	Datetime
日期/时间	DB2Time	Time	TIME	DATETIME（时间精度）	TimeSpan
日期/时间	DB2TimeStamp	Timestamp	TIMESTAMP	DATETIME（时间和日期精度）	DateTime
XML	DB2Xml	Xml[2]	XML		Byte[]
字符数据	DB2String	Char	CHAR	CHAR	String
字符数据	DB2String	VarChar	VARCHAR	VARCHAR	String
字符数据	DB2String	LongVarChar[1]	LONG VARCHAR	LVARCHAR	String
二进制数据	DB2Binary	Binary	CHAR FOR BIT DATA		Byte[]
二进制数据	DB2Binary	Binary[3]	BINARY		Byte[]
二进制数据	DB2Binary	VarBinary[3]	VARBINARY		Byte[]
二进制数据	DB2Binary	LongVarBinary[1]	LONG VARCHAR FOR BIT DATA		Byte[]
图形数据	DB2String	Graphic	GRAPHIC		String
图形数据	DB2String	VarGraphic	VARGRAPHIC		String
图形数据	DB2String	LongVarGraphic[1]	LONG VARGRAPHIC		String
LOB 数据	DB2Clob	Clob	CLOB	CLOB 和 TEXT	String
LOB 数据	DB2Blob	Blob	BLOB	BLOB 和 BYTE	Byte[]
LOB 数据	DB2Clob	DbClob	DBCLOB		String
行标识	DB2RowId	RowId	ROWID		Byte[]

.NET CLR 例程中的参数

.NET CLR 例程中的参数声明必须符合其中一种支持参数样式的要求，且必须遵循用于例程的特定 .NET 语言的参数关键字要求。

如果例程将使用暂存区或 dbinfo 结构，或者将具有 PROGRAM TYPE MAIN 参数接口，那么需要考虑其他详细信息。本主题提出所有 CLR 参数注意事项。

CLR 例程支持的参数样式

在创建例程时，必须在例程的 CREATE 语句的 EXTERNAL 子句中指定例程的参数样式。必须在外部 CLR 例程代码的实现中精确地反映此参数样式。CLR 例程支持下列 DB2 参数样式:

- SQL（受过程和函数支持）
- GENERAL（只受过程支持）
- GENERAL WITH NULLS（只受过程支持）
- DB2SQL（受过程和函数支持）

有关这些参数样式的更多信息，请参阅:

- 第 62 页的『外部例程的参数样式』

CLR 例程参数 null 指示符

如果为 CLR 例程选择的参数样式要求为参数指定 null 指示符，那么当参数样式调用 null 指示符的向量时，会将 null 指示符作为 System.Int16 类型值传递至 CLR 例程，或在 System.Int16[] 值中传递。

如果参数样式指示必须将 null 指示符作为单值参数传递至例程（参数样式 SQL 要求这样做），那么每个参数需要一个 System.Int16 null 指示符。

在 .NET 语言中，必须在单值参数前面加上一个关键字，以指示是通过值还是通过引用来传递参数。必须将用于例程参数的同一关键字用于相关联的 null 指示符参数。下节更详细地讨论了用来指示是通过值还是通过引用来传递自变量的关键字。

有关参数样式 SQL 及其他支持的参数样式的更多信息，请参阅:

- 第 62 页的『外部例程的参数样式』

通过值或通过引用来传递 CLR 例程参数

编译为中间语言（IL）字节码的 .NET 语言例程要求在参数前面加上关键字，以指示参数的特定属性，例如是通过值还是通过引用来传递参数，或者参数是仅输入参数还是仅输出参数。

参数关键字特定于 .NET 语言。例如，为了在 C# 中通过引用来传递参数，参数关键字是 ref，而在 Visual Basic 中，通过引用参数是由 byRef 关键字指示。必须使用关键字来指示例程的 CREATE 语句中所指定的 SQL 参数使用 （IN、OUT 或 INOUT）。

将参数关键字应用于 DB2 例程中的 .NET 语言例程参数时，会适用下列规则:

- 在 C# 中声明 IN 类型参数时*不能* 使用参数关键字，在 Visual Basic 中声明这些类型参数时必须使用 byVal 关键字。
- 必须使用语言特定关键字（指示通过引用传递参数）来声明 INOUT 类型参数。在 C# 中，相应的关键字是 ref。在 Visual Basic 中，相应的关键字是 byRef。
- 必须使用语言特定关键字（指示参数是仅输出参数）来声明 OUT 类型参数。在 C# 中，使用 out 关键字。在 Visual Basic 中，必须使用 byRef 关键字来声明参数。在例程返回至调用者时，必须对仅输出参数赋值。如果例程未对仅输出参数赋值，那么在编译 .NET 例程时，将抛出错误。

下列是返回单一输出参数 language 的例程的 C# 参数样式 SQL 过程原型形式。

```
public static void Counter  (out String language,
                             out Int16  languageNullInd,
                             ref String sqlState,
                                 String funcName,
                                 String funcSpecName,
                             ref String sqlMsgString,
                                 Byte[] scratchPad,
                                 Int32  callType);
```

很明显，是由于额外的 null 指示符参数 languageNullInd（与输出参数 language 相关联）、用于传递 SQLSTATE 的参数、例程名称、例程特定名称以及可选的用户定义 SQL 错误消息，而实现了参数样式 SQL。为参数指定了参数关键字，如下所示:

- 在 C# 中，仅输入参数不需要参数关键字。
- 在 C# 中，“out”关键字指示变量是仅输出参数，且指示调用者尚未初始化参数值。
- 在 C# 中，“ref”关键字指示调用者已初始化参数，且指示例程可选择性地修改此值。

请参阅关于参数传递的 .NET 语言特定文档，以了解有关该语言中的参数关键字的信息。

注: DB2 数据库系统控制所有参数的内存分配，以及维护对传入或传出例程的所有参数的 CLR 引用。

过程结果集不需要参数标记

在过程的过程声明中，将返回给调用者的结果集不需要参数标记。任何未从 CLR 存储过程内部关闭的游标语句将作为结果集传递回至其调用者。

有关 CLR 例程中结果集的更多信息，请参阅:

- 『从 .NET CLR 过程中返回结果集』

作为 CLR 参数的 Dbinfo 结构

CLR 例程通过使用 IL `dbinfo` 类来支持用于将其他数据库信息参数传入和传出例程的 `dbinfo` 结构。此类包含所有可在 C 语言 `sqludf_dbinfo` 结构中找到的元素（与字符串相关联的长度字段除外）。可以使用每个字符串的 .NET 语言 `Length` 属性来找到特定字符串的长度。

要访问 `dbinfo` 类，只需在包含例程的文件中包括 `IBM.Data.DB2` 组合件，然后在例程特征符中由使用的参数样式所指定的位置，添加类型为 `sqludf_dbinfo` 的参数。

作为 CLR 参数的 UDF 暂存区

如果为用户定义的函数请求暂存区，那么会将它传递至作为具有指定大小的 `System.Byte[]` 参数的例程。

CLR UDF 调用类型或最终调用参数

对于已请求最终调用参数的用户定义的函数或对于表函数，会将调用类型参数作为 `System.Int32` 数据类型传递至例程。

CLR 过程支持的 PROGRAM TYPE MAIN

.NET CLR 过程支持程序类型 MAIN。定义为使用程序类型 MAIN 的过程必须具有下列特征符:

```
void functionname(Int32 NumParams, Object[] Params)
```

从 .NET CLR 过程中返回结果集

您可以开发将结果集返回给调用例程或应用程序的 CLR 过程。无法从 CLR 函数（UDF）返回结果集。

开始之前

结果集的 .NET 表示是可从 DB2Command 对象的各种执行调用之一返回的 DB2DataReader 对象。如果 DB2DataReader 对象的 Close() 方法未在过程返回之前进行显式地调用，那么可以返回该对象。将结果集返回给调用者的顺序，与 DB2DataReader 对象的实例化顺序相同。函数定义中不需要其他参数，即可返回结果集。

了解如何创建 CLR 例程将有助于您执行以下过程（用于从 CLR 过程返回结果）中的步骤。

- 第 98 页的『从 DB2 命令窗口创建 .NET CLR 例程』

过程

要从 CLR 过程中返回结果集，请执行下列操作:

1. 在 CLR 例程的 CREATE PROCEDURE 语句中，除指定任何其他相应的语句之外，您还必须指定 DYNAMIC RESULT SETS 子句（其值等于过程将要返回的结果集数）。
2. 在过程声明中，将返回给调用者的结果集不需要参数标记。
3. 在 CLR 例程的 .NET 语言实现中，创建 DB2Connection 对象、DB2Command 对象以及 DB2Transaction 对象。DB2Transaction 对象负责回滚和落实数据库事务。
4. 将 DB2Command 对象的事务属性初始化为 DB2Transaction 对象。
5. 将字符串查询指定给 DB2Command 对象的 CommandText 属性以定义要返回的结果集。
6. 实例化 DB2DataReader 并对其指定 DB2Command 对象方法 ExecuteReader 的调用结果。会将查询的结果集包含在 DB2DataReader 对象中。
7. 在过程返回至调用者之前的任何时候，请不要执行 DB2DataReader 对象的 Close() 方法。会将仍处于打开状态的 DB2DataReader 对象作为结果集返回给调用者。

   如果在过程返回时有多个 DB2DataReader 保持处于打开状态，那么会将 DB2DataReaders 以其创建顺序返回给调用者。只将 CREATE PROCEDURE 语句中所指定数目的结果集返回给调用者。
8. 编译 .NET CLR 语言过程，并将组合件安装至 CREATE PROCEDURE 语句中的 EXTERNAL 子句所指定的位置。针对 CLR 过程执行 CREATE PROCEDURE 语句（如果尚未这样做）。
9. 在将 CLR 过程组合件安装至相应位置且成功执行 CREATE PROCEDURE 语句后，您可以使用 CALL 语句来调用该过程以查看返回给调用者的结果集。

CLR 例程的安全性和执行方式

作为数据库管理员或应用程序开发者，您可能需要保护与 DB2 外部例程相关的组合件，以限制运行时对例程所执行的操作，从而避免令人厌烦的篡改。DB2 .NET 公共语言运行时 (CLR) 例程支持指定执行控制方式，以标识运行时允许例程执行的操作类型。在运行时，DB2 数据库系统可以检测例程尝试执行的操作是否超出对例程指定的执行控制方式的作用域，这有助于确定组合件是否遭破坏。

要设置 CLR 例程的执行控制方式，请在例程的 CREATE 语句中指定可选的 EXECUTION CONTROL 子句。有效方式如下:

- SAFE
- FILEREAD

- FILEWRITE
- NETWORK
- UNSAFE

要在现有 CLR 例程中修改执行控制方式，请执行 ALTER PROCEDURE 或 ALTER FUNCTION 语句。

如果未对 CLR 例程指定 EXECUTION CONTROL 子句，那么缺省情况下，将使用限制性最强的执行控制方式来运行 CLR 例程: SAFE。使用此执行控制方式来创建的例程只能访问由数据库管理器控制的资源。限制性较弱的执行控制方式允许例程访问文件（FILEREAD 或 FILEWRITE），或执行网络操作，例如访问 Web 页面（NETWORK）。执行控制方式 UNSAFE 指定不对例程的行为实施任何限制。以 UNSAFE 执行控制方式定义的例程可以执行二进制代码。

这些方式表示可允许操作的层次结构，并且高级方式包括层次结构中其下所允许的操作。例如，执行控制方式 NETWORK 允许例程访问因特网上的 Web 页面、读/写文件以及访问由数据库管理器控制的资源。建议使用限制性尽可能强的执行控制方式，且避免使用 UNSAFE 方式。

如果 DB2 数据库系统在运行时检测到 CLR 例程正尝试在其执行控制方式作用域外部执行操作，那么 DB2 数据库系统将返回错误 (SQLSTATE 38501)。

只能为 LANGUAGE CLR 例程指定 EXECUTION CONTROL 子句。EXECUTION CONTROL 子句的适用性作用域限制于 .NET CLR 例程本身，且不扩展至它可能调用的任何其他例程。

请参阅相应例程类型的 CREATE 语句的语法，以取得支持的执行控制方式的完整描述。

.NET CLR 例程限制

适用于所有外部例程或特定例程类（过程或 UDF）的一般实现限制也适用与 CLR 例程。另外，还有一些 CLR 例程所特有的限制。这些限制列示如下。

不支持带有 LANGUAGE CLR 子句的 CREATE METHOD 语句

不能为引用了 CLR 组合件的 DB2 数据库结构化类型创建外部方法。不支持使用指定了值为 CLR 的 LANGUAGE 子句的 CREATE METHOD 语句。

不能将 CLR 过程作为 NOT FENCED 过程实现

CLR 过程不能作为未受防护过程运行。用于创建 CLR 过程的 CREATE PROCEDURE 语句不能指定 NOT FENCED 子句。

EXECUTION CONTROL 子句对例程中包含的逻辑有所限制

EXECUTION CONTROL 子句和相关联的值确定了可以在 .NET CLR 例程中执行的逻辑和操作的类型。缺省情况下，EXECUTION CONTROL 子句值设置为 SAFE。对于读取文件、写入文件或访问因特网的例程逻辑而言，必须对 EXECUTION CONTROL 子句指定限制性没那么强的非缺省值。

在 CLR 例程中，最大十进制精度为 29，最大十进制小数位数为 28

在 DB2 数据库中，DECIMAL 数据类型表示为精度 32 位且小数位数 28 位。.NET CLR System.Decimal 数据类型限制为精度 29 位且小数位数 28 位。因此，DB2 外部 CLR 例程不能将大于 (2^96)-1（可以使用 29 位精度和 28 位小数表示的最大值）的值指定给 System.Decimal 数据类型。如果发生这样的赋值，那么 DB2 数据库系统将发出运行时错误 (SQLSTATE 22003, SQLCODE -413)。执行用于创建例程的 CREATE 语句时，如果将 DECIMAL 数据类型参数定义为小数位数大于 28，那么 DB2 数据库系统将发出错误 (SQLSTATE 42613, SQLCODE -628)。

如果您要求例程处理最大精度和小数位数受 DB2 数据库系统支持的十进制值，那么可以使用 Java 之类的另一编程语言来实现外部例程。

在 CLR 例程中不受支持的数据类型

在 CLR 例程中，不支持下列 DB2 SQL 数据类型:

- LONG VARCHAR
- LONG VARCHAR FOR BIT DATA
- LONG GRAPHIC
- ROWID

在 64 位实例上运行 32 位 CLR 例程

CLR 例程不能在 64 位实例上运行，这是因为，目前无法将 .NET Framework 安装在 64 位操作系统上。

不支持使用 .NET CLR 来实现安全性插件

不支持使用 .NET CLR 来编译和链接安全性插件库的源代码。

创建 .NET CLR 例程

创建 .NET CLR 例程包括执行用于在 DB2 数据库服务器中定义该例程的 CREATE 语句以及开发对应例程定义的例程实现。

开始之前

- 复审第 89 页的第 4 章，『.NET 公共语言运行时 (CLR) 例程』。
- 确保您可以访问 DB2 版本 9 服务器（其中包括实例和数据库）。
- 确保操作系统处于 DB2 数据库产品支持的版本级别。
- 确保 Microsoft .NET 开发软件为支持 .NET CLR 例程开发的版本级别。请参阅第 89 页的『对使用 .NET CLR 语言进行外部例程开发的支持』。
- 有权执行 CREATE PROCEDURE 或 CREATE FUNCTION 语句。

有关与 CLR 例程相关联的限制列表，请参阅:

- 第 96 页的『.NET CLR 例程限制』

关于此任务

可用来创建 .NET CLR 例程的方法如下所示:

- 使用 IBM Database Add-Ins for Microsoft Visual Studio 随附的图形工具
- 使用 DB2 命令窗口

通常，使用 IBM Database Add-Ins for Microsoft Visual Studio 创建 .NET CLR 例程最简单。如果无法使用此软件，那么 DB2 命令窗口可通过命令行界面提供类似的支持。

从下列其中一个界面创建 .NET CLR 例程:

过程

- Visual Studio .NET（如果也安装了 IBM Database Add-Ins for Microsoft Visual Studio）。在安装此 Add-In 后，集成至 Visual Studio .NET 的图形工具可用于创建 .NET CLR 可以在 DB2 数据库服务器中工作的 .NET CLR 例程。
- DB2 命令窗口

下一步做什么

要从 DB2 命令窗口创建 .NET CLR 例程，请参阅:

- 『从 DB2 命令窗口创建 .NET CLR 例程』

从 DB2 命令窗口创建 .NET CLR 例程

使用创建任何外部例程的方法来创建对中间语言组合件进行引用的过程和函数。

开始之前

- 了解 CLR 例程实现。要了解 CLR 例程的常规信息以及 CLR 功能的信息，请参阅:
  - 第 89 页的第 4 章，『.NET 公共语言运行时 (CLR) 例程』
- 数据库服务器必须正在运行支持 Microsoft .NET Framework 的 Windows 操作系统。
- 必需将 Microsoft .NET Framework 软件的支持版本安装到该服务器上。.NET Framework 是单独提供或作为 Microsoft .NET Framework 软件开发包的一部分提供。
- 必须安装支持的 DB2 数据库产品或 IBM 数据服务器客户机。请参阅 DB2 数据库产品的安装要求。
- 有权对外部例程执行 CREATE 语句。有关执行 CREATE PROCEDURE 语句或 CREATE FUNCTION 语句所需的特权，请参阅相应语句的详细信息。

限制

有关与 CLR 例程相关联的限制列表，请参阅:

- 第 96 页的『.NET CLR 例程限制』

关于此任务

在下列情况下，您不妨选择使用 .NET 语言来实现外部例程:

- 您需要在例程中封装复杂逻辑，以访问数据库或在数据库外部执行操作。
- 您要求从下列任何项调用封装的逻辑: 多个应用程序、CLP、另一个例程（过程、函数（UDF）或方法）或触发器。
- 您最擅长使用 .NET 语言来编写此逻辑。

过程

1. 使用 CLR 所支持的任何语言来编写例程逻辑。
   - 有关 .NET CLR 例程和 .NET CLR 例程功能的常规信息，请参阅“开始之前”部分中所引用的主题。
   - 如果您的例程将执行 SQL，请使用或导入 IBM.Data.DB2 组合件。
   - 正确地使用映射至 DB2 SQL 数据类型的数据类型来声明主变量和参数。有关 DB2 和 .NET 数据类型之间的数据类型映射的信息，请参阅:
     – 第 91 页的『.NET CLR 例程中的 SQL 数据类型表示』
   - 必须使用 DB2 所支持的其中一种参数样式，且根据 .NET CLR 例程的参数要求来声明参数和参数 null 指示符。此外，会将 UDF 的暂存区以及 DBINFO 类作为参数传递至 CLR 例程。有关参数和原型声明的更多信息，请参阅:
     – 第 92 页的『.NET CLR 例程中的参数』
   - 如果例程是过程且您需要将结果集返回给例程调用者，那么您不需要该结果集的任何参数。有关返回 CLR 例程的结果集的更多信息，请参阅:
     – 第 94 页的『从 .NET CLR 过程中返回结果集』
   - 根据需要设置例程返回值。CLR 标量函数要求在返回前设置返回值。CLR 表函数要求指定返回码作为表函数每个调用的输出参数。CLR 过程不会返回任何返回值。
2. 将代码构建为由 CLR 执行的中间语言（IL）组合件。 有关如何构建访问 DB2 数据库的 CLR .NET 例程的信息，请参阅以下主题:
   - *开发 ADO.NET 和 OLE DB 应用程序* 中的『构建公共语言运行时 (CLR) .NET 例程』
3. 将组合件复制到数据库服务器上的 DB2 *function* 目录。建议将 DB2 例程的相关联组合件或库存储至函数目录。要了解有关函数目录的更多信息，请参阅下列任一语句的 EXTERNAL 子句: CREATE PROCEDURE 或 CREATE FUNCTION。

   如果需要，您可以将组合件复制到服务器上的另一个目录，但是为了成功调用例程，您必须记下组合件的标准路径名，因为下一步需要该标准路径名。
4. 动态或静态地执行适合于例程类型的 SQL 语言 CREATE 语句: CREATE PROCEDURE 或 CREATE FUNCTION。
   - 使用值 CLR 来指定 LANGUAGE 子句。
   - 使用支持的参数样式（在例程代码中实现）的名称来指定 PARAMETER STYLE 子句。
   - 使用组合件（将通过下列其中一个值与例程相关联）的名称来指定 EXTERNAL 子句:
     – 例程组合件的标准路径名。
     – 例程组合件的相对路径名（相对于函数目录）。

     缺省情况下，DB2 数据库系统将在函数目录中按名称查找组合件，除非在 EXTERNAL 子句中指定了库的标准或相对路径名。

     当执行 CREATE 语句时，如果 DB2 数据库系统找不到在 EXTERNAL 子句中指定的组合件，那么将接收到错误 (SQLCODE -20282)，原因码为 1。
   - 使用整数值（等价于例程可能返回的最大结果集数）来指定 DYNAMIC RESULT SETS 子句。

- 不能对 CLR 过程指定 NOT FENCED 子句。缺省情况下，CLR 过程作为 FENCED 过程来执行。

构建 .NET CLR 例程代码

在编写 .NET CLR 例程实现代码后，必须先构建，然后才能部署例程组合件并调用例程。构建 .NET CLR 例程所需的步骤类似于构建任何外部例程所需的步骤，但是存在某些差别。

过程

可以使用三种方法来构建 .NET CLR 例程:

- 使用 IBM Database Add-Ins for Microsoft Visual Studio 随附的图形工具
- 使用 DB2 样本批处理文件
- 从 DB2 命令窗口输入命令

例程的 DB2 样本构建脚本和批处理文件设计用于通过缺省的支持编译器，针对特定操作系统构建 DB2 样本例程（过程和用户定义的函数）以及用户创建的例程。
C# 和 Visual Basic 分别具有一组 DB2 样本构建脚本和批处理文件。通常，使用图形工具或构建脚本（可根据需要轻松修改）来构建 .NET CLR 例程非常容易，但是，如果也知道如何从 DB2 命令窗口构建例程，往往很有帮助。

使用样本构建脚本来构建 .NET 公共语言运行时 (CLR) 例程代码

构建 .NET 公共语言运行时 (CLR) 例程源代码是创建 .NET CLR 例程的子任务。可以使用 DB2 样本批处理文件来简单快捷地完成此任务。

可以将样本构建脚本用于带有或不带 SQL 语句的源代码。构建脚本处理下列操作: 将构建的组合件编译、链接以及部署至 `function` 目录。

作为替代选择，您可以简化构建 .NET CLR 例程代码的任务；要完成简化，请使用 Visual Studio .NET 或使用 DB2 样本构建脚本来手动执行这些步骤。请参阅:

- 使用 Visual Studio .NET 来构建 .NET 公共语言运行时 (CLR) 例程
- 使用 DB2 命令窗口来构建 .NET 公共语言运行时 (CLR) 例程

用于构建 C# 和 Visual Basic .NET CLR 例程的编程语言特定样本构建脚本称为 **bldrtn**。这些脚本及其用于构建的样本程序都位于 DB2 目录，如下所示:

- 对于 C: `sqllib/samples/cs/`
- 对于 C++: `sqllib/samples/vb/`

可以使用 **bldrtn** 脚本来构建包含过程和用户定义的函数的源代码文件。脚本完成下列操作:

- 建立与用户指定数据库的连接
- 编译和链接源代码以生成带有 .DLL 文件后缀的组合件
- 将组合件复制到数据库服务器上的 DB2 函数目录

bldrtn 脚本接受两个自变量:

- 不带任何文件后缀的源代码文件名
- 将与其建立连接的数据库的名称

数据库参数是可选的。如果未提供任何数据库名，那么程序将使用缺省样本数据库。因为必须在数据库所在的相同实例上构建例程，所以用户标识和密码不需要任何自变量。

先决条件

- 必须满足必需的 .NET CLR 例程操作系统以及开发软件先决条件。请参阅: “对 .NET CLR 例程开发的支持”。
- 包含一个或多个例程实现的源代码文件。
- 当前 DB2 实例中将创建例程的数据库的名称。

过程

要构建包含一个或多个例程代码实现的源代码文件，请执行以下步骤。

1. 打开 DB2 命令窗口。
2. 将源代码文件复制到 **bldrtn** 脚本文件所在的相同目录。
3. 如果将在样本数据库中创建例程，请输入构建脚本名称，后跟不带 .cs 或 .vb 文件扩展名的源代码文件名:

   ```
   bldrtn file-name
   ```

   如果将在另一个数据库中创建例程，请输入构建脚本名称、不带任何文件扩展名的源代码文件名以及数据库名:

   ```
   bldrtn file-name database-name
   ```

   脚本会编译和链接源代码并产生组合件。然后，脚本会将组合件复制到数据库服务器上的函数目录
4. 如果这不是您首次构建包含例程实现的源代码文件，请停止然后重新启动数据库以确保 DB2 数据库系统使用的是新版本的共享库。要完成此任务，请在命令行中输入 **db2stop**，接着输入 **db2start**。

在成功构建例程共享库并将其部署至数据库服务器上的函数目录后，您应该完成与创建 C 和 C++ 例程的任务相关联的步骤。

创建 .NET CLR 例程的过程包括针对源代码文件中实现的每个例程执行 CREATE 语句的步骤。在完成创建例程后，您可以调用例程。

从 DB2 命令窗口构建 .NET 公共语言运行时 (CLR) 例程代码

构建 .NET CLR 例程源代码是创建 .NET CLR 例程的子任务。可以手动从 DB2 命令窗口完成此任务。可以遵循相同的过程，而不论例程代码中是否存在 SQL 语句。任务步骤包括将使用 .NET CLR 支持编程语言编写的源代码，编译至文件后缀为 .DLL 的组合件。

开始之前

作为替代选择，您可以简化构建 .NET CLR 例程代码的任务；要完成简化，请使用 Visual Studio .NET 或使用 DB2 样本构建脚本。请参阅:

- 使用 Visual Studio .NET 来构建 .NET 公共语言运行时 (CLR) 例程
- 使用样本构建脚本来构建 .NET 公共语言运行时 (CLR) 例程
- 已满足必需的操作系统和 .NET CLR 例程开发软件先决条件。

- 使用支持的 .NET CLR 编程语言编写的源代码（包含一个或多个 .NET CLR 例程实现）。
- 当前 DB2 实例中将创建例程的数据库的名称。
- 构建 .NET CLR 例程时所需的操作特定编译和链接选项。

过程

要构建包含一个或多个 .NET CLR 例程代码实现的源代码文件:

1. 打开 DB2 命令窗口。
2. 浏览至包含源代码文件的目录。
3. 将建立与数据库（将在其中创建例程）的连接。
4. 编译源代码文件。
5. 链接源代码文件以生成共享库。这需要使用一些特定于 DB2 数据库系统的编译和链接选项。
6. 将文件后缀为 .DLL 的组合件文件复制到数据库服务器上的 DB2 函数目录。
7. 如果这不是您首次构建包含例程实现的源代码文件，请停止然后重新启动数据库以确保 DB2 数据库系统使用的是新版本的共享库。 可通过先发出 **db2stop** 命令然后发出 **db2start** 命令来完成此操作。

结果

在成功构建和部署例程库后，您应该完成与创建 .NET CLR 例程的任务相关联的步骤。创建 .NET CLR 例程的过程包括针对源代码文件中实现的每个例程执行 CREATE 语句的步骤。也必须先完成此步骤，才能调用例程。

示例

下列示例演示如何重新构建 .NET CLR 源代码文件。会显示 Visual Basic 代码文件 myVBfile.vb（包含例程实现）以及 C# 代码文件 myCSfile.cs 的步骤。在 Windows 2000 操作系统上构建了这些例程（使用 Microsoft .NET Framework 1.1 来生成 64 位组合件）。

1. 打开 DB2 命令窗口。
2. 浏览至包含源代码文件的目录。
3. 将建立与数据库（将在其中创建例程）的连接。

   ```
   db2 connect to database-name
   ```

4. 使用建议的编译和链接选项来编译源代码文件（其中，$DB2PATH 是 DB2 实例的安装路径。在运行命令之前，请替换此值）:

   ```
   C# example
   ==================
   csc /out:myCSfile.dll /target:library
       /reference:$DB2PATH%\bin\netf11\IBM.Data.DB2.dll myCSfile.cs

   Visual Basic example
   ====================
   vbc /target:library /libpath:$DB2PATH\bin\netf11
       /reference:$DB2PATH\bin\netf11\IBM.Data.DB2.dll
       /reference:System.dll
       /reference:System.Data.dll myVBfile.vb
   ```

如果发生任何错误，那么编译器将生成输出。此步骤会生成名为 myfile.exp 的导出文件。

5. 将共享库复制到数据库服务器上的 DB2 函数目录。

```
C# example
====================
rm -f ~HOME/sqllib/function/myCSfile.DLL
cp myCSfile $HOME/sqllib/function/myCSfile.DLL

Visual Basic example
====================
rm -f ~HOME/sqllib/function/myVBfile.DLL
cp myVBfile $HOME/sqllib/function/myVBfile.DLL
```

此步骤确保例程库位于 DB2 将在其中查找例程库的缺省目录。请参阅有关创建 .NET CLR 例程的主题，以了解有关如何部署例程库的更多信息。

6. 停止然后重新启动数据库，因为此步骤会重新构建先前构建的例程源代码文件。

```
db2stop
db2start
```

通常，可以使用操作特定样本构建脚本非常轻松地构建 .NET CLR 例程，这些脚本也可用作如何从命令行构建例程的参考。

CLR .NET 例程的编译和链接选项

在 Windows 上使用 Microsoft Visual Basic .NET 编译器或 Microsoft C# 编译器来构建公共语言运行时 (CLR) .NET 例程时，使用 DB2 中提供的编译和链接选项，如 samples\.NET\cs\bldrtn.bat 和 samples\.NET\vb\bldrtn.bat 批处理文件中所示。

使用 Microsoft C# 编译器时 bldrtn 的编译和链接选项:

使用 Microsoft C# 编译器时的编译和链接选项:

csc Microsoft C# 编译器。

/out:%1.dll /target:library
将动态链接库作为存储过程组合件 DLL 进行输出。

/debug 使用调试器。

/lib: "%DB2PATH%"\bin\netf20
对 .NET Framework V2.0 使用库路径。

应用程序支持多个 .NET Framework 版本: V2.0、V3.0 和 V3.5。每个版本都有一个动态链接库。对于 .NET Framework V1.1，请使用 "%DB2PATH%"\bin\netf11 子目录。对于 .NET Framework V2.0、V3.0 和 V3.5，请使用 "%DB2PATH%"\bin\netf20 子目录。

/reference:IBM.Data.DB2.dll
对 IBM Data Server Provider for .NET 使用 DB2 动态链接库

请参阅编译器文档，以了解其他编译器选项。

使用 Microsoft Visual Basic .NET 编译器时 bldrtn 的编译和链接选项:

vbc Microsoft Visual Basic .NET 编译器。

/out:%1.dll /target:library
将动态链接库作为存储过程组合件 DLL 进行输出。

/debug 使用调试器。

/libpath:"%DB2PATH%"\bin\netf20
对 .NET Framework V2.0 使用库路径。

应用程序支持多个 .NET Framework 版本: V2.0、V3.0 和 V3.5。每个版本都有一个动态链接库。对于 .NET Framework V1.1，请使用 "%DB2PATH%"\bin\netf11 子目录。对于 .NET Framework V2.0、V3.0 和 V3.5，请使用 "%DB2PATH%"\bin\netf20 子目录。

/reference:IBM.Data.DB2.dll
对 IBM Data Server Provider for .NET 使用 DB2 动态链接库。

/reference:System.dll
引用 Microsoft Windows 系统动态链接库。

/reference:System.Data.dll
引用 Microsoft Windows 系统数据动态链接库。

请参阅编译器文档，以了解其他编译器选项。

调试 .NET CLR 例程

如果无法创建例程或调用例程，或者在调用时例程的性能或性能不符合预期，那么可能需要调试 .NET CLR 例程。

关于此任务

调试 .NET CLR 例程时，请考虑下列事项:

过程

- 验证是否使用支持的操作系统来开发 .NET CLR 例程。
- 验证是否使用支持的 DB2 数据库服务器和 DB2 客户机来开发 .NET CLR 例程。
- 验证是否使用支持的 Microsoft .NET Framework 开发软件。
- 如果创建例程失败，请执行下列操作:
  - 验证用户是否具有必需的权限和特权来执行 CREATE PROCEDURE 或 CREATE FUNCTION 语句。
- 如果调用例程失败，请执行下列操作:
  - 验证用户是否有权执行例程。如果发生错误 (SQLCODE -551, SQLSTATE 42501)，那么这可能是因为调用者对例程不具有 EXECUTE 特权。任何具有 SECADM 权限或 ACCESSCTRL 权限的用户，或任何对例程具有 EXECUTE WITH GRANT OPTION 特权的用户都可以授予此特权。
  - 验证在例程的 CREATE 语句中使用的例程参数特征符，是否匹配例程实现中的例程参数特征符。
  - 验证例程实现中使用的数据类型，是否与 CREATE 语句的例程参数特征符中指定的数据类型兼容。
  - 验证在例程实现中，用来指示方法（必须依据该方法传递参数: 通过值或通过引用）的 .NET CLR 语言特定关键字是否有效。

- 验证在 CREATE PROCEDURE 或 CREATE FUNCTION 语句的 EXTERNAL 子句中指定的值，是否匹配包含例程实现的 .NET CLR 组合件，在安装 DB2 数据库服务器的计算机文件系统上的位置。
- 如果例程是函数，请验证是否已在例程实现中正确地编程所有适用的调用类型。如果在例程中定义了 FINAL CALL 子句，那么这特别重要。

• 如果例程的行为不符合预期，请执行下列操作:
- 修改例程以将诊断信息输出至位于全局可访问目录中的文件。无法从 .NET CLR 例程将诊断信息输出至屏幕。请不要将输出定向至 DB2 数据库管理器或 DB2 数据库所使用目录中的文件。
- 要在本地调试例程，请编写简单 .NET 应用程序以直接调用例程入口点。有关如何使用 Microsoft Visual Studio .NET 中的调试功能的信息，请参阅 Microsoft Visual Studio .NET 编译器文档。

结果

有关与 .NET CLR 例程创建和调用相关的常见错误的更多信息，请参阅:

• 『与 .NET CLR 例程相关的错误』

与 .NET CLR 例程相关的错误

虽然外部例程共享一个通用实现，但是可能会发生一些特定于 CLR 例程的 DB2 数据库系统错误。

此参考按 SQLCODE 或行为列出最常见的 .NET CLR 相关错误以及一些调试建议。与例程相关的 DB2 数据库系统错误可按如下所示进行分类:

例程创建时间错误
执行例程的 CREATE 语句时发生的错误。

例程运行时错误
例程调用或执行期间发生的错误。

不管 DB2 数据库系统抛出 DB2 例程相关错误的时间为何，错误消息文本都会详细说明错误原因以及用户为解决问题而应该执行的操作。可以在 **db2diag** 诊断日志文件中找到其他例程错误场景信息。

CLR 例程创建时间错误

SQLCODE -451, SQLSTATE 42815
如果尝试执行 CREATE TYPE 语句，但该语句包含使用值 CLR 来指定 LANGUAGE 子句的外部方法声明，那么会抛出此错误。目前，无法为引用 CLR 组合件的结构化类型创建 DB2 外部方法。更改 LANGUAGE 子句以便它为方法指定支持的语言，然后使用该备用语言来实现方法。

SQLCODE -449, SQLSTATE 42878
CLR 例程的 CREATE 语句在 EXTERNAL NAME 子句中包含的库或函数标识未经有效地格式化。对于语言 CLR，EXTERNAL 子句值必须专门采用以下格式: '<a>:<b>!<c>'，如下所示:

- <a> 是类所在的 CLR 组合件文件。
- <b> 是要调用的方法所在的类。
- <c> 是要调用的方法。

在单引号、对象标识和分隔字符之间不允许使用前导或尾部空白字符（例如，' <a> ! <b> ' 无效）。但是，如果平台允许，那么路径名和文件名可以包含空格。对于所有文件名，可以使用短格式文件名（示例：math.dll）或标准路径名（示例：d:\udfs\math.dll）来指定文件。如果使用短格式文件名、平台是 UNIX 或例程是 LANGUAGE CLR 例程，那么该文件必须位于函数目录中。如果平台是 Windows 且例程不是 LANGUAGE CLR 例程，那么该文件必须位于系统 PATH 中。应该始终在文件名中包含文件扩展名，示例：.a（在 UNIX 上）和 .dll（在 Windows 上）。

CLR 例程运行时错误

SQLCODE -20282, SQLSTATE 42724，原因码 1

找不到例程 CREATE 语句中 EXTERNAL 子句所指定的外部组合件。

- 检查 EXTERNAL 子句是否指定正确的例程组合件名称，以及组合件是否位于指定的位置。如果 EXTERNAL 子句未指定期望组合件的标准路径名，那么 DB2 数据库系统假定所提供的路径名是组合件的相对路径名（相对于 DB2 数据库系统函数目录）。

SQLCODE -20282, SQLSTATE 42724，原因码 2

在例程 CREATE 语句的 EXTERNAL 子句所指定的位置中找到组合件，但是在组合件中找不到与 EXTERNAL 子句中所指定的类相匹配的类。

- 检查 EXTERNAL 子句中所指定的组合件名称是否是例程的正确组合件，以及它是否存在于指定的位置。
- 检查 EXTERNAL 子句中所指定的类名是否是正确的类名，以及它是否存在于指定的组合件。

SQLCODE -20282, SQLSTATE 42724，原因码 3

在例程 CREATE 语句的 EXTERNAL 子句所指定的位置中找到组合件，该组合件具有正确匹配的类定义，但是例程方法特征符与例程 CREATE 语句中所指定的例程特征符不匹配。

- 检查 EXTERNAL 子句中所指定的组合件名称是否是例程的正确组合件，以及它是否存在于指定的位置。
- 检查 EXTERNAL 子句中所指定的类名是否是正确的类名，以及它是否存在于指定的组合件。
- 检查参数样式实现是否与例程 CREATE 语句中所指定的参数样式匹配。
- 检查参数实现的顺序是否与例程 CREATE 语句中的参数声明顺序匹配，以及它是否遵循参数样式的额外参数要求。
- 检查是否已正确地将 SQL 参数数据类型映射至 CLR .NET 支持数据类型。

SQLCODE -4301, SQLSTATE 58004，原因码 5 或 6

尝试启动 .NET 解释器或与其通信时发生错误。DB2 数据库系统无法装入从属 .NET 库 [原因码 5]，或调用 .NET 解释器失败 [原因码 6]。

- 确保 DB2 实例已正确配置为运行 .NET 过程或函数（mscoree.dll 必须存在于系统 PATH）。确保 db2clr.dll 存在于 sqllib/bin 目录，以及已将 IBM.Data.DB2 安装至全局组合件高速缓存。如果这些都不存在，请确保已将 .NET Framework V1.1 或更高版本安装至数据库服务器，且该数据库服务器正在运行 DB2 版本 8.2 或更高发行版。

SQLCODE -4302, SQLSTATE 38501

执行例程时、准备执行例程时或在执行例程后发生未处理的异常。导致此异常的原因可能是例程逻辑编程错误未处理，或发生内部处理错误。对于此类型的错误，会将 .NET 堆栈回溯（指示发生未处理异常的位置）写入 db2diag 日志文件。

如果例程尝试执行的操作，超出例程的指定执行方式所允许的操作范围，那么也会发生此错误。在此情况下，将在 db2diag 日志文件中记录一个条目，用以专门指示发生异常的原因是执行控制违例。此外，还将包含异常堆栈回溯（指示发生违例的位置）。

确定例程的组合件是否受损或最近是否进行了修改。如果对例程进行的修改是有效的，那么可能会发生此问题，因为例程的 EXECUTION CONTROL 方式已不再设为适合于所更改逻辑的方式。如果您确信组合件未遭篡改，那么可以根据情况使用 ALTER PROCEDURE 或 ALTER FUNCTION 语句来修改例程的执行方式。请参阅下列主题，以了解更多信息:

- 第 95 页的『CLR 例程的安全性和执行方式』

.NET CLR 例程示例

开发 .NET CLR 例程时，参考示例以获取 CREATE 语句的含义和 .NET CLR 例程代码的形式会有所帮助。

关于此任务

下列主题包含 .NET CLR 过程和函数（其中包括标量函数和表函数）的示例:

.NET CLR 过程

- Visual Basic .NET CLR 过程示例
- C# .NET CLR 过程示例

.NET CLR 函数

- Visual Basic .NET CLR 函数示例
- C# .NET CLR 函数示例

C# .NET CLR 过程示例

在了解过程（又称为存储过程）的基本原理，以及 .NET 公共语言运行时例程的基础知识之后，您可以开始在应用程序中使用 CLR 过程。

开始之前

在使用 CLR 过程示例之前，您不妨阅读下列概念主题:

- 第 89 页的第 4 章，『.NET 公共语言运行时 (CLR) 例程』
- 第 98 页的『从 DB2 命令窗口创建 .NET CLR 例程』
- 构建公共语言运行时 (CLR) .NET 例程

关于此任务

本主题包含使用 C# 来实现的 CLR 过程示例；本示例说明支持的参数样式、传递参数（其中包括 dbinfo 结构）以及如何返回结果集等。要获取使用 C# 编写的 CLR UDF 示例，请参阅:

- 第 133 页的『C# .NET CLR 函数示例』

下列示例利用 SAMPLE 数据库中包括的名为 EMPLOYEE 的表。

过程

在创建您自己的 C# CLR 过程时，请使用下列示例作为参考:

- 108
- 108
- 109
- 111
- 112
- 112
- 113

示例

C# 外部代码文件

下列示例显示各种 C# 过程实现。每个示例都包含两个部分: CREATE PROCEDURE 语句以及过程的外部 C# 代码实现（可根据其构建相关联的组合件）。

包含下列示例过程实现的 C# 源文件称为 gwenProc.cs 且具有下列格式:

```
using System;
using System.IO;
using IBM.Data.DB2;

namespace bizLogic
{
   class empOps
   {          ...
     // C# procedures
              ...
   }
}
```

在此文件顶部指示文件包含。如果此文件中的任何过程包含 SQL，那么将需要 IBM.Data.DB2 包含。此文件中有一个名称空间声明，以及一个包含过程的类 empOps。可选择性地使用名称空间。如果使用名称空间，那么该名称空间必须出现在组合件路径名（在 CREATE PROCEDURE 语句的 EXTERNAL 子句中提供）中。

必须记下此文件的名称、名称空间以及包含给定过程实现的类的名称。这些名称很重要，因为每个过程的 CREATE PROCEDURE 语句的 EXTERNAL 子句必须指定此信息，以便 DB2 可以找到 CLR 过程的组合件和类。

示例 1: C# 参数样式 GENERAL 过程

本示例显示下列内容:

- 参数样式 GENERAL 过程的 CREATE PROCEDURE 语句
- 参数样式 GENERAL 过程的 C# 代码

此过程接受职员 ID 和当前奖金金额作为输入。它会检索职员的姓名和薪水。如果当前奖金金额为零，那么将根据职员薪水来计算新的奖金，然后将此奖金与职员姓名一起返回。如果找不到此职员，那么将返回一个 NULL 字符串。

```
CREATE PROCEDURE setEmpBonusGEN(IN empID CHAR(6),
                                INOUT bonus Decimal(9,2),
                                OUT empName VARCHAR(60))
SPECIFIC SetEmpBonusGEN
LANGUAGE CLR
PARAMETER STYLE GENERAL
MODIFIES SQL DATA
EXECUTION CONTROL SAFE
FENCED
THREADSAFE
DYNAMIC RESULT SETS 0
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenProc.dll:bizLogic.empOps!SetEmpBonusGEN' ;
```

```
public static void SetEmpBonusGEN(    String empID,
                                   ref Decimal bonus,
                                   out String empName)
{
   // Declare local variables
   Decimal salary = 0;

   DB2Command myCommand = DB2Context.GetCommand();
   myCommand.CommandText =
                    "SELECT FIRSTNME, MIDINIT, LASTNAME, SALARY "
                  + "FROM EMPLOYEE "
                  + "WHERE EMPNO = '" + empID + '";

   DB2DataReader reader = myCommand.ExecuteReader();

   if (reader.Read())  // If employee record is found
   {
      // Get the employee's full name and salary
      empName = reader.GetString(0) + " " +
                reader.GetString(1) + ". " +
                reader.GetString(2);

      salary = reader.GetDecimal(3);

      if (bonus == 0)
      {
         if (salary > 75000)
         {
            bonus = salary * (Decimal)0.025;
         }
         else
         {
            bonus = salary * (Decimal)0.05;
         }
      }
   }
   else  // Employee not found
   {
      empName = "";  // Set output parameter
   }

   reader.Close();
}
```

示例 2: C# 参数样式 GENERAL WITH NULLS 过程

本示例显示下列内容:

- 参数样式 GENERAL WITH NULLS 过程的 CREATE PROCEDURE 语句
- 参数样式 GENERAL WITH NULLS 过程的 C# 代码

此过程接受职员 ID 和当前奖金金额作为输入。如果输入参数不为 NULL，那么将检索职员的姓名和薪水。如果当前奖金金额为零，那么将根据薪水来计算新的奖金，然后将此奖金与职员姓名一起返回。如果找不到职员数据，那么将返回一个 NULL 字符串和一个整数。

```
CREATE PROCEDURE SetEmpbonusGENNULL(IN empID CHAR(6),
                                    INOUT bonus Decimal(9,2),
                                    OUT empName VARCHAR(60))
SPECIFIC SetEmpbonusGENNULL
LANGUAGE CLR
PARAMETER STYLE GENERAL WITH NULLS
DYNAMIC RESULT SETS 0
MODIFIES SQL DATA
EXECUTION CONTROL SAFE
FENCED
THREADSAFE
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenProc.dll:bizLogic.empOps!SetEmpBonusGENNULL'
;
```

```
 public static void SetEmpBonusGENNULL(    String empID,
                                       ref Decimal bonus,
                                       out String empName,
                                           Int16[] NullInds)
 {
    Decimal salary = 0;
    if (NullInds[0] == -1) // Check if the input is null
    {
      NullInds[1] = -1;    // Return a NULL bonus value
      empName = "";        // Set output value
      NullInds[2] = -1;    // Return a NULL empName value
    }
    else
    {
       DB2Command myCommand = DB2Context.GetCommand();
        myCommand.CommandText =
                       "SELECT FIRSTNME, MIDINIT, LASTNAME, SALARY "
                     + "FROM EMPLOYEE "
                     + "WHERE EMPNO = '" + empID + "'";
       DB2DataReader reader = myCommand.ExecuteReader();

       if (reader.Read())  // If employee record is found
       {
          // Get the employee's full name and salary
          empName = reader.GetString(0) + " "
          +
                    reader.GetString(1) + ". " +
                    reader.GetString(2);
          salary = reader.GetDecimal(3);

          if (bonus == 0)
          {
            if (salary > 75000)
            {
               bonus = salary * (Decimal)0.025;
               NullInds[1] = 0; // Return a non-NULL value
            }
            else
            {
               bonus = salary * (Decimal)0.05;
               NullInds[1] = 0; // Return a non-NULL value
            }
          }
       }
       else  // Employee not found
       {
          empName = "*sdq;;         // Set output parameter
```

```
            NullInds[2] = -1;     // Return a NULL value
        }

        reader.Close();
    }
}
```

示例 3: C# 参数样式 SQL 过程

本示例显示下列内容:

- 参数样式 SQL 过程的 CREATE PROCEDURE 语句
- 参数样式 SQL 过程的 C# 代码

此过程接受职员 ID 和当前奖金金额作为输入。它会检索职员的姓名和薪水。如果当前奖金金额为零，那么将根据薪水来计算新的奖金，然后将此奖金与职员姓名一起返回。如果找不到此职员，那么将返回一个 NULL 字符串。

```
CREATE PROCEDURE SetEmpbonusSQL(IN empID CHAR(6),
                                INOUT bonus Decimal(9,2),
                                OUT empName VARCHAR(60))
SPECIFIC SetEmpbonusSQL
LANGUAGE CLR
PARAMETER STYLE SQL
DYNAMIC RESULT SETS 0
MODIFIES SQL DATA
FENCED
THREADSAFE
EXECUTION CONTROL SAFE
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenProc.dll:bizLogic.empOps!SetEmpBonusSQL' ;
```

```
 public static void SetEmpBonusSQL(    String empID,
                                    ref Decimal bonus,
                                    out String empName,
                                        Int16  empIDNullInd,
                                    ref Int16  bonusNullInd,
                                    out Int16  empNameNullInd,
                                    ref string sqlStateate,
                                        string funcName,
                                        string specName,
                                    ref string sqlMessageText)
 {
    // Declare local host variables
    Decimal salary eq; 0;

    if (empIDNullInd == -1) // Check if the input is null
    {
       bonusNullInd = -1;   // Return a NULL bonus value
       empName = "";
       empNameNullInd = -1; // Return a NULL empName value
    }
    else
    {
       DB2Command myCommand = DB2Context.GetCommand();
       myCommand.CommandText =
                        "SELECT FIRSTNME, MIDINIT, LASTNAME, SALARY
                        "
                      + "FROM EMPLOYEE "
                      + "WHERE EMPNO = '" + empID + "'";

       DB2DataReader reader = myCommand.ExecuteReader();

       if (reader.Read())  // If employee record is found
       {
          // Get the employee's full name and salary
          empName = reader.GetString(0) + " "
          +
```

```
            reader.GetString(1) + ". " +
            reader.GetString(2);
            empNameNullInd = 0;
            salary = reader.GetDecimal(3);

            if (bonus == 0)
            {
               if (salary > 75000)
               {
                  bonus = salary * (Decimal)0.025;
                  bonusNullInd = 0;  // Return a non-NULL value
               }
               else
               {
                  bonus = salary * (Decimal)0.05;
                  bonusNullInd = 0;  // Return a non-NULL value
               }
            }
         }
         else  // Employee not found
         {
            empName = "";  // Set output parameter
            empNameNullInd = -1;  // Return a NULL value
         }

         reader.Close();
      }
   }
```

示例 4: 返回结果集的 C# 参数样式 GENERAL 过程

本示例显示下列内容:

- 返回结果集的外部 C# 过程的 CREATE PROCEDURE 语句
- 返回结果集的参数样式 GENERAL 过程的 C# 代码

此过程接受表名作为参数。它返回一个包含输入参数所指定的全部表行的结果集。通过为过程返回时打开的给定查询结果集保留一个 DB2DataReader 来实现此目的。具体地说，如果未执行 reader.Close()，那么将返回结果集。

```
CREATE PROCEDURE ReturnResultSet(IN tableName
                                 VARCHAR(20))
SPECIFIC ReturnResultSet
DYNAMIC RESULT SETS 1
LANGUAGE CLR
PARAMETER STYLE GENERAL
FENCED
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenProc.dll:bizLogic.empOps!ReturnResultSet' ;
```

```
public static void ReturnResultSet(string tableName)
{
   DB2Command myCommand = DB2Context.GetCommand();

   // Set the SQL statement to be executed and execute it.
   myCommand.CommandText = "SELECT * FROM " + tableName;
   DB2DataReader reader = myCommand.ExecuteReader();

   // The DB2DataReader contains the result of the query.
   // This result set can be returned with the procedure,
   // by simply NOT closing the DB2DataReader.
   // Specifically, do NOT execute reader.Close();
}
```

示例 5: 访问 dbinfo 结构的 C# 参数样式 SQL 过程

本示例显示下列内容:

- 访问 dbinfo 结构的过程的 CREATE PROCEDURE 语句

- 访问 dbinfo 结构的参数样式 SQL 过程的 C# 代码

必须在 CREATE PROCEDURE 语句中指定 DBINFO 子句，才能访问 dbinfo 结构。CREATE PROCEDURE 语句中的 dbinfo 结构不需要参数，但是，必须在外部例程代码中为此结构创建参数。此过程只返回 dbinfo 结构中 dbname 字段的当前数据库名值。

```
CREATE PROCEDURE ReturnDbName(OUT dbName VARCHAR(20))
SPECIFIC ReturnDbName
DYNAMIC RESULT SETS 0
LANGUAGE CLR
PARAMETER STYLE SQL
FENCED
THREADSAFE
EXECUTION CONTROL SAFE
DBINFO
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenProc.dll:bizLogic.empOps!ReturnDbName'
;
```

```
 public static void ReturnDbName(out string dbName,
                                  out Int16  dbNameNullInd,
                                  ref string sqlState,
                                      string funcName,
                                      string specName,
                                  ref string sqlMessageText,
                                      sqludf_dbinfo dbinfo)
 {
    // Retrieve the current database name from the
    // dbinfo structure and return it.
    // ** Note! ** dbinfo field names are case sensitive
    dbName = dbinfo.dbname;
    dbNameNullInd = 0;  // Return a non-null value;

    // If you want to return a user-defined error in
    // the SQLCA you can specify a 5 digit user-defined
    // sqlStateate and an error message string text.
    // For example:
    //
    //   sqlStateate = "ABCDE";
    //   sqlMessageText = "A user-defined error has occurred"
    //
    //  DB2 returns the above values to the client in the
    //  SQLCA structure.  The values are used to generate a
    //  standard DB2 sqlStateate error.
 }
```

示例 6: 使用 PROGRAM TYPE MAIN 样式的 C# 过程

本示例显示下列内容:

- 使用主程序样式的过程的 CREATE PROCEDURE 语句
- 使用 MAIN 程序样式的 C# 参数样式 GENERAL WITH NULLS 代码

必须在带有值 MAIN 的 CREATE PROCEDURE 语句中指定 PROGRAM TYPE 子句，才能使用主程序样式来实现例程。在 CREATE PROCEDURE 语句中指定参数，但是在代码实现中，会通过 argc 整数参数和 argv 数组参数，将参数传递至例程。

```
CREATE PROCEDURE MainStyle( IN empID CHAR(6),
                            INOUT bonus Decimal(9,2),
                            OUT empName VARCHAR(60))
SPECIFIC MainStyle
DYNAMIC RESULT SETS 0
LANGUAGE CLR
PARAMETER STYLE GENERAL WITH NULLS
```

```
MODIFIES SQL DATA
FENCED
THREADSAFE
EXECUTION CONTROL SAFE
PROGRAM TYPE MAIN
EXTERNAL NAME 'gwenProc.dll:bizLogic.empOps!main' ;

 public static void main(Int32 argc, Object[]
 argv)
 {
   String empID = (String)argv[0];   // argv[0] has nullInd:argv[3]
   Decimal bonus = (Decimal)argv[1]; // argv[1] has nullInd:argv[4]
                                     // argv[2] has nullInd:argv[5]
   Decimal salary = 0;
   Int16[] NullInds = (Int16[])argv[3];

   if ((NullInds[0]) == (Int16)(-1)) // Check if empID is null
   {
      NullInds[1] = (Int16)(-1);  // Return a NULL bonus value
      argv[1] = (String)"";       // Set output parameter empName
      NullInds[2] = (Int16)(-1);  // Return a NULL empName value
      Return;
   }
   else
   {
      DB2Command myCommand = DB2Context.GetCommand();
      myCommand.CommandText =
                     "SELECT FIRSTNME, MIDINIT, LASTNAME, salary "
                   + "FROM EMPLOYEE "
                   + "WHERE EMPNO = '" + empID + "'";

      DB2DataReader reader = myCommand.ExecuteReader();

      if (reader.Read())  // If employee record is found
      {
        // Get the employee's full name and salary
        argv[2] = (String) (reader.GetString(0) + " " +
                            reader.GetString(1) + ".
                            " +
                            reader.GetString(2));
        NullInds[2] = (Int16)0;
        salary = reader.GetDecimal(3);

        if (bonus == 0)
        {
           if (salary > 75000)
           {
              argv[1] = (Decimal)(salary * (Decimal)0.025);
              NullInds[1] = (Int16)(0);  // Return a non-NULL value
           }
           else
           {
              argv[1] = (Decimal)(salary * (Decimal)0.05);
              NullInds[1] = (Int16)(0);  // Return a non-NULL value
           }
        }
      }
      else  // Employee not found
      {
         argv[2] = (String)("");         // Set output parameter
         NullInds[2] = (Int16)(-1);      // Return a NULL value
      }

      reader.Close();
    }
 }
```

Visual Basic .NET CLR 函数示例

在了解用户定义的函数 (UDF) 的基本原理以及 CLR 例程的基础知识之后，您可以开始在应用程序和数据库环境中利用 CLR UDF。本主题包含一些帮您入门的 CLR UDF 示例。

开始之前

在使用 CLR UDF 示例之前，您不妨阅读下列概念主题:

- 第 89 页的第 4 章，『.NET 公共语言运行时 (CLR) 例程』
- 第 98 页的『从 DB2 命令窗口创建 .NET CLR 例程』
- 第 52 页的『外部标量函数』
- 构建公共语言运行时 (CLR) .NET 例程

关于此任务

要获取使用 Visual Basic 编写的 CLR 过程示例，请参阅:

- 第 119 页的『Visual Basic .NET CLR 过程示例』

下列示例利用 SAMPLE 数据库中包括的名为 EMPLOYEE 的表。

过程

在创建您自己的 Visual Basic CLR UDF 时，请使用下列示例作为参考:

- 115
- 115
- 118

示例

Visual Basic 外部代码文件

下列示例显示各种 Visual Basic UDF 实现。为每个 UDF 提供了 CREATE FUNCTION 语句及对应的 Visual Basic 源代码（可根据其构建相关联的组合件）。包含下列示例中所使用函数声明的 Visual Basic 源文件称为 gwenVbUDF.cs 且具有下列格式:

```
using System;
using System.IO;
using IBM.Data.DB2;

Namespace bizLogic

   ...
   ' Class definitions that contain UDF declarations
   ' and any supporting class definitions
   ...

End Namespace
```

必须将函数声明包含在 Visual Basic 文件的类中。可选择性地使用名称空间。如果使用名称空间，那么该名称空间必须出现在组合件路径名（在 CREATE PROCEDURE 语句的 EXTERNAL 子句中提供）中。如果函数包含 SQL，那么将需要 IBM.Data.DB2. 包含。

示例 1: Visual Basic 参数样式 SQL 表函数

本示例显示下列内容:

- 参数样式 SQL 表函数的 CREATE FUNCTION 语句
- 参数样式 SQL 表函数的 Visual Basic 代码

此表函数会返回一个包含根据数据数组建立的职员数据行的表。本示例有两个相关联的类。类 person 表示职员，而类 empOps 包含使用类 person 的例程表 UDF。会根据输入参数的值来更新职员薪水信息。第一次调用表函数时，会在此表函数本身中创建本示例中的数据数组。此外，还可通过从文件系统上的文本文件读取数据来创建了此类数组。数组数据值将写入暂存区，以便可以在表函数的后续调用中访问此数据。

在每次调用表函数时，会从数组中读取一个记录，并在函数所返回的表中生成一行。通过将表函数的输出参数设为期望的行值，在表中生成行。在最终调用表函数之后，会返回所生成行的表。

```
CREATE FUNCTION TableUDF(double)
RETURNS TABLE (name varchar(20),
               job varchar(20),
               salary double)
EXTERNAL NAME 'gwenVbUDF.dll:bizLogic.empOps!TableUDF'
LANGUAGE CLR
PARAMETER STYLE SQL
NOT DETERMINISTIC
FENCED
SCRATCHPAD 10
FINAL CALL
DISALLOW PARALLEL
NO DBINFO
EXECUTION CONTROL SAFE
```

```
Class Person
' The class Person is a supporting class for
' the table function UDF, tableUDF, below.

  Private name As String
  Private position As String
  Private salary As Int32

  Public Sub New(ByVal newName As String, _
                 ByVal newPosition As String, _
                 ByVal newSalary As Int32)

    name = newName
    position = newPosition
    salary = newSalary
  End Sub

  Public Property GetName() As String
    Get
      Return name
    End Get

    Set (ByVal value As String)
      name = value
    End Set
  End Property

  Public Property GetPosition() As String
    Get
      Return position
    End Get
```

```
    Set (ByVal value As String)
      position = value
    End Set
  End Property

  Public Property GetSalary() As Int32
    Get
      Return salary
    End Get

    Set (ByVal value As Int32)
      salary = value
    End Set
  End Property

End Class

 Class empOps

   Public Shared Sub TableUDF(byVal factor as Double, _
                              byRef name As String, _
                              byRef position As String, _
                              byRef salary As Double, _
                              byVal factorNullInd As Int16, _
                              byRef nameNullInd As Int16, _
                              byRef positionNullInd As Int16, _
                              byRef salaryNullInd As Int16, _
                              byRef sqlState As String, _
                              byVal funcName As String, _
                              byVal specName As String, _
                              byRef sqlMessageText As String, _
                              byVal scratchPad As Byte(), _
                              byVal callType As Int32)

     Dim intRow As Int16

     intRow = 0

     ' Create an array of Person type information
     Dim staff(2) As Person
     staff(0) = New Person("Gwen", "Developer", 10000)
     staff(1) = New Person("Andrew", "Developer", 20000)
     staff(2) = New Person("Liu", "Team Leader", 30000)

     ' Initialize output parameter values and NULL indicators
     salary = 0
     name = position = ""
     nameNullInd = positionNullInd = salaryNullInd = -1

     Select callType
        Case -2   ' Case SQLUDF_TF_FIRST:
        Case -1   ' Case SQLUDF_TF_OPEN:
          intRow = 1
          scratchPad(0) = intRow  ' Write to scratchpad
        Case 0    ' Case SQLUDF_TF_FETCH:
          intRow = scratchPad(0)
          If intRow > staff.Length
             sqlState = "02000"  ' Return an error SQLSTATE
          Else
             ' Generate a row in the output table
             ' based on the staff array data.
             name = staff(intRow).GetName()
             position = staff(intRow).GetPosition()
             salary = (staff(intRow).GetSalary()) * factor
             nameNullInd = 0
             positionNullInd = 0
             salaryNullInd = 0
          End If
```

```
            intRow = intRow + 1
            scratchPad(0) = intRow  ' Write scratchpad

          Case 1    ' Case SQLUDF_TF_CLOSE:

          Case 2    ' Case SQLUDF_TF_FINAL:
        End Select

      End Sub

    End Class
```

示例 2: Visual Basic 参数样式 SQL 标量函数

本示例显示下列内容:

- 参数样式 SQL 标量函数的 CREATE FUNCTION 语句
- 参数样式 SQL 标量函数的 Visual Basic 代码

此标量函数会返回每个输入值所操作的单一计数值。对于输入值集第 n 个位置中的输入值，输出标量值是值 n。对于标量函数的每个调用（每个调用都与行或值输入集中的每一行或值相关联），计数值会增加 1，并返回计数的当前值。然后，会将计数保存在暂存区内存缓冲区中，以在标量函数的每个调用之间保留计数值。

例如，如果我们具有如下定义的表，那么可以很容易地调用此标量函数:

```
CREATE TABLE T (i1 INTEGER);
INSERT INTO T VALUES 12, 45, 16, 99;
```

可以使用如下简单查询来调用标量函数:

```
SELECT my_count(i1) as count, i1 FROM T;
```

此类查询的输出会是:

```
COUNT           I1
-----------     ----------
1               12
2               45
3               16
4               99
```

此标量 UDF 很简单。不只是返回行数，您还可以使用标量函数来格式化现有列中的数据。例如，您可能会将字符串附加至地址列中的每个值、根据一系列输入字符串构建复杂字符串，或者对您必须在其中存放中间结果的数据集执行复杂数学求值。

```
CREATE FUNCTION mycount(INTEGER)
RETURNS INTEGER
LANGUAGE CLR
PARAMETER STYLE SQL
NO SQL
SCRATCHPAD 10
FINAL CALL
FENCED
EXECUTION CONTROL SAFE
NOT DETERMINISTIC
EXTERNAL NAME 'gwenUDF.dll:bizLogic.empOps!CountUp';
```

```
Class empOps
  Public Shared Sub CountUp(byVal input As Int32, _
                            byRef outCounter As Int32, _
                            byVal nullIndInput As Int16, _
                            byRef nullIndOutCounter As Int16, _
                            byRef sqlState As String, _
```

```
                                byVal qualName As String, _
                                byVal specName As String, _
                                byRef sqlMessageText As String, _
                                byVal scratchPad As Byte(), _
                                byVal callType As Int32)

        Dim counter As Int32
        counter = 1

        Select callType
           case -1           ' case SQLUDF_TF_OPEN_CALL
              scratchPad(0) = counter
              outCounter = counter
              nullIndOutCounter = 0
           case 0             'case SQLUDF_TF_FETCH_CALL:
              counter = scratchPad(0)
              counter = counter + 1
              outCounter = counter
              nullIndOutCounter = 0
              scratchPad(0) = counter
           case 1            'case SQLUDF_CLOSE_CALL:
              counter = scratchPad(0)
              outCounter = counter
              nullIndOutCounter = 0
           case Else          ' Should never enter here
              ' These cases won't occur for the following reasons:
              ' Case -2  (SQLUDF_TF_FIRST)     —>No FINAL CALL in CREATE stmt
              ' Case 2   (SQLUDF_TF_FINAL)     —>No FINAL CALL in CREATE stmt
              ' Case 255 (SQLUDF_TF_FINAL_CRA) —>No SQL used in the function
              '
              ' * Note!*
              ' ---------
              ' The Else is required so that at compile time
              ' out parameter outCounter is always set *
              outCounter = 0
              nullIndOutCounter = -1
        End Select
     End Sub

  End Class
```

Visual Basic .NET CLR 过程示例

在了解过程（又称为存储过程）的基本原理，以及 .NET 公共语言运行时例程的基础知识之后，您可以开始在应用程序中使用 CLR 过程。本主题包含一些使用 Visual Basic 实现的 CLR 过程示例；这些示例说明了受支持的参数样式、传递参数（其中包括 dbinfo 结构）以及如何返回结果集和其他信息。

开始之前

在使用 CLR 过程示例之前，您不妨阅读下列概念主题:

- 第 89 页的第 4 章，『.NET 公共语言运行时 (CLR) 例程』
- 第 98 页的『从 DB2 命令窗口创建 .NET CLR 例程』
- 第 3 页的『使用例程的益处』
- 构建公共语言运行时 (CLR) .NET 例程

关于此任务

要获取使用 Visual Basic 编写的 CLR UDF 示例，请参阅:

- 第 115 页的『Visual Basic .NET CLR 函数示例』

下列示例利用 SAMPLE 数据库中包括的名为 EMPLOYEE 的表。

过程

在创建您自己的 Visual Basic CLR 过程时，请使用下列示例作为参考:

- 120
- 120
- 121
- 122
- 124
- 124
- 125

示例

Visual Basic 外部代码文件

下列示例显示各种 Visual Basic 过程实现。每个示例都包含两个部分: CREATE PROCEDURE 语句以及过程的外部 Visual Basic 代码实现（可根据其构建相关联的组合件）。

包含下列示例过程实现的 Visual Basic 源文件称为 gwenVbProc.vb 且具有下列格式:

```
using System;
using System.IO;
using IBM.Data.DB2;

Namespace bizLogic

   Class empOps
              ...
     ' Visual Basic procedures
              ...
   End Class
End Namespace
```

在此文件顶部指示文件包含。如果此文件中的任何过程包含 SQL，那么将需要 IBM.Data.DB2 包含。此文件中有一个名称空间声明，以及一个包含过程的类 empOps。可选择性地使用名称空间。如果使用名称空间，那么该名称空间必须出现在组合件路径名（在 CREATE PROCEDURE 语句的 EXTERNAL 子句中提供）中。

必须记下此文件的名称、名称空间以及包含给定过程实现的类的名称。这些名称很重要，因为每个过程的 CREATE PROCEDURE 语句的 EXTERNAL 子句必须指定此信息，以便 DB2 可以找到 CLR 过程的组合件和类。

示例 1: Visual Basic 参数样式 GENERAL 过程

本示例显示下列内容:

- 参数样式 GENERAL 过程的 CREATE PROCEDURE 语句
- 参数样式 GENERAL 过程的 Visual Basic 代码

此过程接受职员 ID 和当前奖金金额作为输入。它会检索职员的姓名和薪水。如果当前奖金金额为零，那么将根据职员薪水来计算新的奖金，然后将此奖金与职员姓名一起返回。如果找不到此职员，那么将返回一个 NULL 字符串。

```
CREATE PROCEDURE SetEmpBonusGEN(IN empId CHAR(6),
                                INOUT bonus Decimal(9,2),
                                OUT empName VARCHAR(60))
SPECIFIC setEmpBonusGEN
LANGUAGE CLR
PARAMETER STYLE GENERAL
DYNAMIC RESULT SETS 0
FENCED
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenVbProc.dll:bizLogic.empOps!SetEmpBonusGEN'
```

```
  Public Shared Sub SetEmpBonusGEN(ByVal empId As String, _
                                   ByRef bonus As Decimal, _
                                   ByRef empName As String)

     Dim salary As Decimal
     Dim myCommand As DB2Command
     Dim myReader As DB2DataReader

     salary = 0

     myCommand = DB2Context.GetCommand()
     myCommand.CommandText = _
               "SELECT FIRSTNME, MIDINIT, LASTNAME, SALARY " _
             + "FROM EMPLOYEE " _
             + "WHERE EMPNO = '" + empId + "'"
     myReader = myCommand.ExecuteReader()

     If myReader.Read()  ' If employee record is found
        ' Get the employee's full name and salary
        empName = myReader.GetString(0) + " " _
                + myReader.GetString(1) + ". " _
                + myReader.GetString(2)

        salary = myReader.GetDecimal(3)

        If bonus = 0
           If salary > 75000
              bonus = salary * 0.025
           Else
              bonus = salary * 0.05
           End If
        End If
     Else  ' Employee not found
        empName = ""       ' Set output parameter
     End If

     myReader.Close()

  End Sub
```

示例 2: Visual Basic 参数样式 GENERAL WITH NULLS 过程

本示例显示下列内容:

- 参数样式 GENERAL WITH NULLS 过程的 CREATE PROCEDURE 语句
- 参数样式 GENERAL WITH NULLS 过程的 Visual Basic 代码

此过程接受职员 ID 和当前奖金金额作为输入。如果输入参数不为 NULL，那么将检索职员的姓名和薪水。如果当前奖金金额为零，那么将根据薪水来计算新的奖金，然后将此奖金与职员姓名一起返回。如果找不到职员数据，那么将返回一个 NULL 字符串和一个整数。

```
CREATE PROCEDURE SetEmpBonusGENNULL(IN empId CHAR(6),
                                    INOUT bonus Decimal(9,2),
                                    OUT empName VARCHAR(60))
SPECIFIC SetEmpBonusGENNULL
```

```
LANGUAGE CLR
PARAMETER STYLE GENERAL WITH NULLS
DYNAMIC RESULT SETS 0
FENCED
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenVbProc.dll:bizLogic.empOps!SetEmpBonusGENNULL'

  Public Shared Sub SetEmpBonusGENNULL(ByVal empId As String, _
                                       ByRef bonus As Decimal, _
                                       ByRef empName As String, _
                                       byVal nullInds As Int16())

     Dim salary As Decimal
     Dim myCommand As DB2Command
     Dim myReader As DB2DataReader

     salary = 0

     If nullInds(0) = -1   ' Check if the input is null
        nullInds(1) = -1   ' Return a NULL bonus value
        empName = ""       ' Set output parameter
        nullInds(2) = -1   ' Return a NULL empName value
        Return
     Else
        myCommand = DB2Context.GetCommand()
        myCommand.CommandText = _
                  "SELECT FIRSTNME, MIDINIT, LASTNAME, SALARY " _
                + "FROM EMPLOYEE " _
                + "WHERE EMPNO = '" + empId + "'"

        myReader = myCommand.ExecuteReader()

        If myReader.Read()  ' If employee record is found
           ' Get the employee's full name and salary
           empName = myReader.GetString(0) + " " _
                   + myReader.GetString(1) + ". " _
                   + myReader.GetString(2)

           salary = myReader.GetDecimal(3)

           If bonus = 0
              If salary > 75000
                 bonus = Salary * 0.025
                 nullInds(1) = 0 'Return a non-NULL value
              Else
                 bonus = salary * 0.05
                 nullInds(1) = 0 ' Return a non-NULL value
              End If
           Else  'Employee not found
              empName = ""       ' Set output parameter
              nullInds(2) = -1   ' Return a NULL value
           End If
        End If

        myReader.Close()

     End If

  End Sub
```

示例 3: Visual Basic 参数样式 SQL 过程

本示例显示下列内容:

- 参数样式 SQL 过程的 CREATE PROCEDURE 语句
- 参数样式 SQL 过程的 Visual Basic 代码

此过程接受职员 ID 和当前奖金金额作为输入。它会检索职员的姓名和薪水。如果当前奖金金额为零，那么将根据薪水来计算新的奖金，然后将此奖金与职员姓名一起返回。如果找不到此职员，那么将返回一个 NULL 字符串。

```
CREATE PROCEDURE SetEmpBonusSQL(IN empId CHAR(6),
                                INOUT bonus Decimal(9,2),
                                OUT empName VARCHAR(60))
SPECIFIC SetEmpBonusSQL
LANGUAGE CLR
PARAMETER STYLE SQL
DYNAMIC RESULT SETS 0
FENCED
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenVbProc.dll:bizLogic.empOps!SetEmpBonusSQL'
```

```
  Public Shared Sub SetEmpBonusSQL(byVal empId As String, _
                                   byRef bonus As Decimal, _
                                   byRef empName As String, _
                                   byVal empIdNullInd As Int16, _
                                   byRef bonusNullInd As Int16, _
                                   byRef empNameNullInd As Int16, _
                                   byRef sqlState As String, _
                                   byVal funcName As String, _
                                   byVal specName As String, _
                                   byRef sqlMessageText As String)

    ' Declare local host variables
    Dim salary As Decimal
    Dim myCommand As DB2Command
    Dim myReader As DB2DataReader

    salary = 0

    If empIdNullInd = -1   ' Check if the input is null
       bonusNullInd = -1   ' Return a NULL Bonus value
       empName = ""
       empNameNullInd = -1 ' Return a NULL empName value
    Else
      myCommand = DB2Context.GetCommand()
      myCommand.CommandText = _
                 "SELECT FIRSTNME, MIDINIT, LASTNAME, SALARY " _
               + "FROM EMPLOYEE " _
               + " WHERE EMPNO = '" + empId + "'"

       myReader = myCommand.ExecuteReader()

       If myReader.Read()  ' If employee record is found
          ' Get the employee's full name and salary
          empName = myReader.GetString(0) + " "
                  + myReader.GetString(1) _
                  + ". " +  myReader.GetString(2)
          empNameNullInd = 0
          salary = myReader.GetDecimal(3)

          If bonus = 0
             If salary > 75000
                bonus = salary * 0.025
                bonusNullInd = 0  ' Return a non-NULL value
             Else
                bonus = salary * 0.05
                bonusNullInd = 0  ' Return a non-NULL value
             End If
          End If
       Else  ' Employee not found
          empName = ""            ' Set output parameter
          empNameNullInd = -1     ' Return a NULL value
       End If
```

```
            myReader.Close()
        End If

    End Sub
```

示例 4: 返回结果集的 Visual Basic 参数样式 GENERAL 过程

本示例显示下列内容:

- 返回结果集的外部 Visual Basic 过程的 CREATE PROCEDURE 语句
- 返回结果集的参数样式 GENERAL 过程的 Visual Basic 代码

此过程接受表名作为参数。它返回一个包含输入参数所指定的全部表行的结果集。通过为过程返回时打开的给定查询结果集保留一个 DB2DataReader 来实现此目的。具体地说，如果未执行 reader.Close()，那么将返回结果集。

```
CREATE PROCEDURE ReturnResultSet(IN tableName VARCHAR(20))
SPECIFIC ReturnResultSet
DYNAMIC RESULT SETS 1
LANGUAGE CLR
PARAMETER STYLE GENERAL
FENCED
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenVbProc.dll:bizLogic.empOps!ReturnResultSet'
```

```
Public Shared Sub ReturnResultSet(byVal tableName As String)

    Dim myCommand As DB2Command
    Dim myReader As DB2DataReader

    myCommand = DB2Context.GetCommand()

    ' Set the SQL statement to be executed and execute it.
    myCommand.CommandText = "SELECT * FROM " + tableName
    myReader = myCommand.ExecuteReader()

    ' The DB2DataReader contains the result of the query.
    ' This result set can be returned with the procedure,
    ' by simply NOT closing the DB2DataReader.
    ' Specifically, do NOT execute reader.Close()

End Sub
```

示例 5: 访问 dbinfo 结构的 Visual Basic 参数样式 SQL 过程

本示例显示下列内容:

- 访问 dbinfo 结构的过程的 CREATE PROCEDURE 语句
- 访问 dbinfo 结构的参数样式 SQL 过程的 Visual Basic 代码

必须在 CREATE PROCEDURE 语句中指定 DBINFO 子句，才能访问 dbinfo 结构。CREATE PROCEDURE 语句中的 dbinfo 结构不需要参数，但是，必须在外部例程代码中为此结构创建参数。此过程只返回 dbinfo 结构中 dbname 字段的当前数据库名值。

```
CREATE PROCEDURE ReturnDbName(OUT dbName VARCHAR(20))
SPECIFIC ReturnDbName
LANGUAGE CLR
PARAMETER STYLE SQL
DBINFO
FENCED
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenVbProc.dll:bizLogic.empOps!ReturnDbName'
```

```
  Public Shared Sub ReturnDbName(byRef dbName As String, _
                                 byRef dbNameNullInd As Int16, _
                                 byRef sqlState As String, _
                                 byVal funcName As String, _
                                 byVal specName As String, _
                                 byRef sqlMessageText As String, _
                                 byVal dbinfo As sqludf_dbinfo)

    ' Retrieve the current database name from the
    ' dbinfo structure and return it.
    dbName = dbinfo.dbname
    dbNameNullInd = 0  ' Return a non-null value

    ' If you want to return a user-defined error in
    ' the SQLCA you can specify a 5 digit user-defined
    ' SQLSTATE and an error message string text.
    ' For example:
    '
    ' sqlState = "ABCDE"
    ' msg_token = "A user-defined error has occurred"
    '
    ' These will be returned by DB2 in the SQLCA.  It
    ' will appear in the format of a regular DB2 sqlState
    ' error.
  End Sub
```

示例 6: 使用 PROGRAM TYPE MAIN 样式的 Visual Basic 过程

本示例显示下列内容:

- 使用主程序样式的过程的 CREATE PROCEDURE 语句
- 使用 MAIN 程序样式的 Visual Basic 参数样式 GENERAL WITH NULLS 代码

必须在带有值 MAIN 的 CREATE PROCEDURE 语句中指定 PROGRAM TYPE 子句，才能使用主程序样式来实现例程。在 CREATE PROCEDURE 语句中指定参数，但是在代码实现中，会通过 argc 整数参数和 argv 数组参数，将参数传递至例程。

```
CREATE PROCEDURE MainStyle(IN empId CHAR(6),
                           INOUT bonus Decimal(9,2),
                           OUT empName VARCHAR(60))
SPECIFIC mainStyle
DYNAMIC RESULT SETS 0
LANGUAGE CLR
PARAMETER STYLE GENERAL WITH NULLS
FENCED
PROGRAM TYPE MAIN
EXTERNAL NAME 'gwenVbProc.dll:bizLogic.empOps!Main'
```

```
  Public Shared Sub Main( byVal argc As Int32, _
                          byVal argv As Object())

    Dim myCommand As DB2Command
    Dim myReader As DB2DataReader
    Dim empId As String
    Dim bonus As Decimal
    Dim salary As Decimal
    Dim nullInds As Int16()

    empId = argv(0)  ' argv[0] (IN)    nullInd = argv[3]
    bonus = argv(1)  ' argv[1] (INOUT) nullInd = argv[4]
                     ' argv[2] (OUT)   nullInd = argv[5]
    salary = 0
    nullInds = argv(3)
```

```
If nullInds(0) = -1     ' Check if the empId input is null
   nullInds(1) = -1     ' Return a NULL Bonus value
   argv(1) = ""         ' Set output parameter empName
   nullInds(2) = -1     ' Return a NULL empName value
   Return
Else
   ' If the employee exists and the current bonus is 0,
   ' calculate a new employee bonus based on the employee's
   ' salary.  Return the employee name and the new bonus
   myCommand = DB2Context.GetCommand()
   myCommand.CommandText = _
             "SELECT FIRSTNME, MIDINIT, LASTNAME, SALARY " _
          + " FROM EMPLOYEE " _
          + " WHERE EMPNO = '" + empId + "'"

   myReader = myCommand.ExecuteReader()


   If myReader.Read()  ' If employee record is found
      ' Get the employee's full name and salary
      argv(2) = myReader.GetString(0) + " "  _
              + myReader.GetString(1) + ". " _
              + myReader.GetString(2)
      nullInds(2) = 0
      salary = myReader.GetDecimal(3)

      If bonus = 0
         If salary > 75000
            argv(1) = salary * 0.025
            nullInds(1) = 0  ' Return a non-NULL value
         Else
            argv(1) = Salary * 0.05
            nullInds(1) = 0  ' Return a non-NULL value
         End If
      End If
   Else  ' Employee not found
      argv(2) = ""      ' Set output parameter
      nullInds(2) = -1  ' Return a NULL value
   End If

   myReader.Close()
End If

End Sub
```

示例: C# .NET CLR 过程中的 XML 和 XQuery 支持

在了解过程的基本原理、.NET 公共语言运行时例程的基础知识、XQuery 和 XML 之后，您可以开始创建并使用具有 XML 功能的 CLR 过程。

下列示例演示一个带有类型为 XML 的参数的 C# .NET CLR 过程，以及演示如何更新和查询 XML 数据。

先决条件

在使用 CLR 过程示例之前，您不妨阅读下列概念主题:

- .NET 公共语言运行时 (CLR) 例程
- 从 DB2 命令窗口创建 .NET CLR 例程
- 使用例程的益处

下列示例利用名为如下定义的表 xmlDataTable:

```
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))@
```

过程 在创建您自己的 C# CLR 过程时，请使用下列示例作为参考:

- 『C# 外部代码文件』
- 第 128 页的『示例 1: 具有 XML 功能的 C# 参数样式 GENERAL 过程』

C# 外部代码文件

本示例包含两个部分: CREATE PROCEDURE 语句以及过程的外部 C# 代码实现（可根据其构建相关联的组合件）。

包含下列示例过程实现的 C# 源文件称为 gwenProc.cs 且具有下列格式:

```
using System;
using System.IO;
using System.Data;
using IBM.Data.DB2;
using IBM.Data.DB2Types;

namespace bizLogic
{
   class empOps
   {          ...
     // C# procedures
              ...
   }
}
```

在此文件顶部指示文件包含。如果此文件中的任何过程包含 SQL，那么将需要 IBM.Data.DB2 包含。如果此文件中的任何过程包含类型为 XML 的参数或变量，那么将需要 IBM.Data.DB2Types 包含。此文件中有一个名称空间声明，以及一个包含过程的类 empOps。可选择性地使用名称空间。如果使用名称空间，那么该名称空间必须出现在组合件路径名（在 CREATE PROCEDURE 语句的 EXTERNAL 子句中提供）中。

必须记下此文件的名称、名称空间以及包含给定过程实现的类的名称。这些名称很重要，因为每个过程的 CREATE PROCEDURE 语句的 EXTERNAL 子句必须指定此信息，以便 DB2 数据库系统可以找到 CLR 过程的组合件和类。

示例 1: 具有 XML 功能的 C# 参数样式 GENERAL 过程

本示例显示下列内容:

- 参数样式 GENERAL 过程的 CREATE PROCEDURE 语句
- 带有 XML 参数的参数样式 GENERAL 过程的 C# 代码

此过程接受两个参数（整数 inNum 和 inXML）。会将这些值插入表 xmlDataTable。然后使用 XQuery 来检索 XML 值。使用 SQL 来检索另一个 XML 值。会将检索到的 XML 值分别指定给两个输出参数（outXML1 和 outXML2）。不返回结果集。

```
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;
}
```

示例: C 过程中的 XML 和 XQuery 支持

在了解过程的基本原理、C 例程的基础知识、XQuery 和 XML 之后，您可以开始创建并使用具有 XML 功能的 C 过程。

下列示例演示一个带有类型为 XML 的参数的 C 过程，以及演示如何更新和查询 XML 数据。

先决条件

在使用 C 过程示例之前，您不妨阅读下列概念主题:

- 使用例程的益处

下列示例利用名为如下定义的表 xmlDataTable:

```
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))
```

过程 在创建您自己的 C 过程时，请使用下列示例作为参考:

- 『C 外部代码文件』
- 『示例 1: 具有 XML 功能的 C 参数样式 SQL 过程』

C 外部代码文件

本示例包含两个部分: CREATE PROCEDURE 语句以及过程的外部 C 代码实现（可根据其构建相关联的组合件）。

包含下列示例过程实现的 C 源文件称为 gwenProc.SQC 且具有下列格式:

```
#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
               ...
```

在此文件顶部指示文件包含。嵌入式 SQL 例程中的 XML 支持不需要额外的包含文件。

必须记下此文件的名称以及对应于过程实现的函数的名称。这些名称很重要，因为每个过程的 CREATE PROCEDURE 语句的 EXTERNAL 子句必须指定此信息，以便 DB2 数据库管理器可以找到对应于 C 过程的库和入口点。

示例 1: 具有 XML 功能的 C 参数样式 SQL 过程

本示例显示下列内容:

- 参数样式 SQL 过程的 CREATE PROCEDURE 语句
- 带有 XML 参数的参数样式 SQL 过程的 C 代码

此过程接收两个输入参数。第一个输入参数为 inNum 且属于 INTEGER 类型。第二个输入参数为 inXML 且属于 XML 类型。使用输入参数的值将行插入表 xmlDataTable。然后使用 SQL 语句来检索 XML 值。使用 XQuery 表达式来检索另一个 XML 值。会将检索到的 XML 值分别指定给两个输出参数（out1XML 和 out2XML）。不返回结果集。

```
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;
}
```

C# .NET CLR 函数示例

在了解用户定义的函数 (UDF) 的基本原理以及 CLR 例程的基础知识之后，您可以开始在应用程序和数据库环境中利用 CLR UDF。本主题包含一些帮您入门的 CLR UDF 示例。

开始之前

在使用 CLR UDF 示例之前，您不妨阅读下列概念主题:

- 第 89 页的第 4 章，『.NET 公共语言运行时 (CLR) 例程』
- 第 98 页的『从 DB2 命令窗口创建 .NET CLR 例程』
- 第 52 页的『外部标量函数』
- 构建公共语言运行时 (CLR) .NET 例程

下列示例利用 SAMPLE 数据库中包括的名为 EMPLOYEE 的表。

关于此任务

要获取使用 C# 编写的 CLR 过程示例，请参阅:

- 第 107 页的『C# .NET CLR 过程示例』

过程

在创建您自己的 C# CLR UDF 时，请使用下列示例作为参考:

- 133
- 134
- 136

示例

C# 外部代码文件

下列示例显示各种 C# UDF 实现。为每个 UDF 提供了 CREATE FUNCTION 语句及对应的 C# 源代码（可根据其构建相关联的组合件）。包含下列示例中所使用函数声明的 C# 源文件称为 gwenUDF.cs 且具有下列格式:

```
using System;
using System.IO;
using IBM.Data.DB2;

namespace bizLogic
{
   ...
   // Class definitions that contain UDF declarations
   // and any supporting class definitions
   ...
}
```

必须将函数声明包含在 C# 文件的类中。可选择性地使用名称空间。如果使用名称空间，那么该名称空间必须出现在组合件路径名（在 CREATE PROCEDURE 语句的 EXTERNAL 子句中提供）中。如果函数包含 SQL，那么将需要 `IBM.Data.DB2.` 包含。

示例 1: C# 参数样式 SQL 表函数

本示例显示下列内容:

- 参数样式 SQL 表函数的 CREATE FUNCTION 语句
- 参数样式 SQL 表函数的 C# 代码

此表函数会返回一个包含根据数据数组建立的职员数据行的表。本示例有两个相关联的类。类 `person` 表示职员，而类 `empOps` 包含使用类 `person` 的例程表 UDF。会根据输入参数的值来更新职员薪水信息。第一次调用表函数时，会在此表函数本身中创建本示例中的数据数组。此外，还可通过从文件系统上的文本文件读取数据来创建了此类数组。数组数据值将写入暂存区，以便可以在表函数的后续调用中访问此数据。

在每次调用表函数时，会从数组中读取一个记录，并在函数所返回的表中生成一行。通过将表函数的输出参数设为期望的行值，在表中生成行。在最终调用表函数之后，会返回所生成行的表。

```
CREATE FUNCTION tableUDF(double)
RETURNS TABLE (name varchar(20),
               job varchar(20),
               salary double)
EXTERNAL NAME 'gwenUDF.dll:bizLogic.empOps!tableUDF'
LANGUAGE CLR
PARAMETER STYLE SQL
NOT DETERMINISTIC
FENCED
THREADSAFE
SCRATCHPAD 10
FINAL CALL
EXECUTION CONTROL SAFE
DISALLOW PARALLEL
NO DBINFO
```

```
// The class Person is a supporting class for
// the table function UDF, tableUDF, below.
class Person
{
     private String name;
     private String position;
     private Int32 salary;

     public Person(String newName, String newPosition, Int32
     newSalary)
     {
        this.name = newName;
        this.position = newPosition;
```

```
      this.salary = newSalary;
    }

    public String getName()
    {
      return this.name;
    }

    public String getPosition()
    {
      return this.position;
    }

    public Int32 getSalary()
    {
      return this.salary;
    }
}

class empOps
{

  public static void TableUDF( Double factor, out String name,
                   out String position, out Double salary,
                   Int16 factorNullInd, out Int16 nameNullInd,
                   out Int16 positionNullInd, out Int16 salaryNullInd,
                   ref String sqlState, String funcName,
                   String specName, ref String sqlMessageText,
                   Byte[] scratchPad, Int32 callType)
  {

    Int16 intRow = 0;

    // Create an array of Person type information
    Person[] Staff = new
    Person[3];
    Staff[0] = new Person("Gwen", "Developer", 10000);
    Staff[1] = new Person("Andrew", "Developer", 20000);
    Staff[2] = new Person("Liu", "Team Leader", 30000);

    salary = 0;
    name = position = "";
    nameNullInd = positionNullInd = salaryNullInd = -1;

    switch(callType)
    {
      case (-2):  // Case SQLUDF_TF_FIRST:
        break;

      case (-1):  // Case SQLUDF_TF_OPEN:
        intRow = 1;
        scratchPad[0] = (Byte)intRow;  // Write to scratchpad
        break;
      case (0):   // Case SQLUDF_TF_FETCH:
        intRow = (Int16)scratchPad[0];
        if (intRow > Staff.Length)
        {
          sqlState = "02000";  // Return an error SQLSTATE
        }
        else
        {
          // Generate a row in the output table
          // based on the Staff array data.
          name =
          Staff[intRow-1].getName();
          position = Staff[intRow-1].getPosition();
          salary = (Staff[intRow-1].getSalary()) * factor;
          nameNullInd = 0;
```

```
                positionNullInd = 0;
                salaryNullInd = 0;
              }
              intRow++;
              scratchPad[0] = (Byte)intRow;  // Write scratchpad
              break;

            case (1):   // Case SQLUDF_TF_CLOSE:
              break;

            case (2):   // Case SQLUDF_TF_FINAL:
              break;
          }
        }
      }
```

示例 2: C# 参数样式 SQL 标量函数

本示例显示下列内容:

- 参数样式 SQL 标量函数的 CREATE FUNCTION 语句
- 参数样式 SQL 标量函数的 C# 代码

此标量函数会返回每个输入值所操作的单一计数值。对于输入值集第 n 个位置中的输入值，输出标量值是值 n。对于标量函数的每个调用（每个调用都与行或值输入集中的每一行或值相关联），计数值会增加 1，并返回计数的当前值。然后，会将计数保存在暂存区内存缓冲区中，以在标量函数的每个调用之间保留计数值。

例如，如果我们具有如下定义的表，那么可以很容易地调用此标量函数:

```
CREATE TABLE T (i1 INTEGER);
INSERT INTO T VALUES 12, 45, 16, 99;
```

可以使用如下简单查询来调用标量函数:

```
  SELECT countUp(i1) as count, i1 FROM T;
```

此类查询的输出会是:

```
COUNT           I1
-----------     ----------
1               12
2               45
3               16
4               99
```

此标量 UDF 很简单。不只是返回行数，您还可以使用标量函数来格式化现有列中的数据。例如，您可能会将字符串附加至地址列中的每个值、根据一系列输入字符串构建复杂字符串，或者对您必须在其中存放中间结果的数据集执行复杂数学求值。

```
CREATE FUNCTION countUp(INTEGER)
RETURNS INTEGER
LANGUAGE CLR
PARAMETER STYLE SQL
SCRATCHPAD 10
FINAL CALL
NO SQL
FENCED
THREADSAFE
NOT DETERMINISTIC
EXECUTION CONTROL SAFE
EXTERNAL NAME 'gwenUDF.dll:bizLogic.empOps!CountUp' ;
```

```
class empOps
{
   public static void CountUp(     Int32 input,
                                out Int32 outCounter,
                                    Int16 inputNullInd,
                                out Int16 outCounterNullInd,
                                ref String sqlState,
                                    String funcName,
                                    String specName,
                                ref String sqlMessageText,
                                    Byte[] scratchPad,
                                    Int32 callType)
   {
      Int32 counter = 1;

      switch(callType)
      {
         case -1: // case SQLUDF_FIRST_CALL
           scratchPad[0] = (Byte)counter;
           outCounter = counter;
           outCounterNullInd = 0;
           break;
         case 0:  // case SQLUDF_NORMAL_CALL:
           counter = (Int32)scratchPad[0];
           counter = counter + 1;
           outCounter = counter;
           outCounterNullInd = 0;
           scratchPad[0] =
           (Byte)counter;
           break;
         case 1:  // case SQLUDF_FINAL_CALL:
           counter =
           (Int32)scratchPad[0];
           outCounter = counter;
           outCounterNullInd = 0;
           break;
         default: // Should never enter here
                  // * Required so that at compile time
                  //   out parameter outCounter is always set *
           outCounter = (Int32)(0);
           outCounterNullInd = -1;
           sqlState="ABCDE";
           sqlMessageText = "Should not get here: Default
           case!";
           break;
      }
   }
}
```

第 5 章 C 和 C++ 例程

C 和 C++ 例程是通过执行 CREATE PROCEDURE、CREATE FUNCTION 或 CREATE METHOD 语句（将从 C 或 C++ 源代码构建的库作为其外部代码主体来引用）来创建的外部例程。

C 和 C++ 例程可选择性地通过包括嵌入式 SQL 语句来执行 SQL 语句。

下列术语在 C 和 C++ 例程的上下文中很重要:

CREATE 语句
用来在数据库中创建例程的 SQL 语言 CREATE 语句。

例程实体源代码
包含 C 或 C++ 例程实现（对应于 CREATE 语句 EXTERNAL 子句规范）的源代码文件。

预编译器
DB2 数据库实用程序，可以预先解析例程源代码实现以验证代码中所包含的 SQL 语句，然后生成程序包。

编译器 编译和链接源代码实现时所需的编程语言特定软件。

程序包 包含例程访问路径信息（在例程运行时，DB2 数据库系统将使用此信息来执行例程代码实现中所包含的 SQL 语句）的文件。

例程库 包含已编译形式的例程源代码的文件。在 Windows 中，此文件有时称为 DLL，因为其文件扩展名为 .dll。

在开发 C 或 C++ 例程前，必须了解例程基本原理以及特定于 C 和 C++ 例程的特有功能和特征。了解嵌入式 SQL API 以及嵌入式 SQL 应用程序开发的基本原理也很重要。要了解这些主题的更多信息，请参阅下列主题:

- 外部例程
- 嵌入式 SQL
- C 和 C++ 例程的包含文件
- C 和 C++ 例程中的参数
- C 和 C++ 例程的限制

开发 C 或 C++ 例程涉及遵循逐步指示信息以及查看 C 或 C++ 例程示例。请参阅:

- 创建 C 和 C++ 例程
- C 过程示例
- C 用户定义的函数示例

对使用 C 语言进行外部例程开发的支持

要使用 C 语言来开发外部例程，您必须使用支持的编译器和开发软件。

所有支持使用 C 语言进行 DB2 数据库应用程序开发的编译器和开发软件也支持使用 C 进行外部例程开发。

对使用 C++ 进行外部例程开发的支持

要使用 C++ 来开发外部例程，您必须使用支持的编译器和开发软件。

所有支持使用 C 进行 DB2 数据库应用程序开发的编译器和开发软件也支持使用 C++ 进行外部例程开发。

用于开发 C 和 C++ 例程的工具

支持 C 和 C++ 例程的工具与那些支持嵌入式 SQL C 和 C++ 应用程序的工具相同。

未提供任何其他 DB2 开发环境或图形用户界面工具来开发、调试或部署嵌入式 SQL 应用程序或例程。

通常使用下列命令行界面来开发、调试以及部署嵌入式 SQL 应用程序和例程:

- DB2 命令行处理器
- DB2 命令窗口

这些界面支持执行在数据库中创建例程所需的 SQL 语句。此外，还可以从这些界面发出构建 C 和 C++ 例程（包含嵌入式 SQL）所需的 PREPARE 语句和 **BIND** 命令。

设计 C 和 C++ 例程

设计 C 和 C++ 例程是一项应该在创建 C 和 C++ 例程之前执行的任务。设计 C 和 C++ 例程通常与设计使用其他编程语言来实现的外部例程以及设计嵌入式 SQL 应用程序相关。

开始之前

- 对外部例程的一般了解
- C 或 C++ 编程经验
- 可选: 有关嵌入式 SQL 或 CLI 应用程序开发的知识和经验（如果例程将执行 SQL 语句）

下列主题可以向您提供一些必备信息。

有关外部例程的功能和使用的更多信息，请参阅下列主题:

- 请参阅主题第 18 页的『外部例程实现』。

有关嵌入式 SQL API 的特征的更多信息，请参阅下列主题:

- 请参阅《*开发嵌入式 SQL 应用程序*》中的『嵌入式 SQL 简介』主题

关于此任务

掌握必备知识之后，设计嵌入式 SQL 例程的主要注意事项是了解 C 和 C++ 例程的特有功能和特征:

过程

- 第 141 页的『C 和 C++ 例程开发所需的包含文件 (sqludf.h)』
- 第 142 页的『C 和 C++ 例程中的参数』
- 第 143 页的『参数样式 SQL C 和 C++ 过程』

- 第 146 页的『参数样式 SQL C 和 C++ 函数』
- 第 157 页的『C 和 C++ 例程中的 SQL 数据类型处理』
- 第 176 页的『C 和 C++ 例程中的图形主变量』
- 第 178 页的『从 C 和 C++ 过程中返回结果集』
- 第 177 页的『C++ 类型装饰』
- 第 78 页的『外部例程限制』

结果

在了解 C 和 C++ 特征之后，您不妨参阅:

- 第 179 页的『创建 C 和 C++ 例程』

C 和 C++ 例程开发所需的包含文件 (sqludf.h)

sqludf.h 包含文件包括编写例程实现时所需的结构、定义以及值。

虽然文件名中含有“udf”（由于历史原因），但此文件对于存储过程和方法也很实用。编译例程时，您需要引用其中包含此文件的 DB2 安装 include 目录。

建议使用此文件中的对象，以确保使用适合于特定操作系统和操作系统位宽的 C 数据类型。

sqludf.h 文件包含结构定义及其描述。下列是其内容的简要摘要:

- SQL 数据类型的宏定义（支持将这些数据类型用作不需要表示为 C 或 C++ 结构的外部例程的参数）。在此文件中，定义的名称格式如下: SQLUDF_*x* 和 SQLUDF_*x*_FBD，其中 *x* 是 SQL 数据类型名称，而 FBD 表示那些以二进制格式存储的数据类型的 FOR BIT DATA。

  此外，还包括根据 AS LOCATOR 子句定义的自变量或结果的 C 语言类型。这只适用于 UDF 和方法。
- 表示下列 SQL 数据类型和特殊参数时所需的 C 结构定义:
  - VARCHAR FOR BIT DATA 数据类型
  - LONG VARCHAR 数据类型
  - LONG VARCHAR FOR BIT DATA 数据类型
  - LONG VARGRAPHIC 数据类型
  - BLOB 数据类型
  - CLOB 数据类型
  - DBCLOB 数据类型
  - scratchpad 结构
  - dbinfo 结构

  使用具有多个字段值的结构（而不是简单 C 数据类型）来表示上述定义。

  scratchpad 结构定义缓冲区，此缓冲区会传递至用户定义的函数以便在函数调用期间使用。但是，与变量不同的是，存储在暂存区中的数据，在单一调用中的多次用户定义的函数调用之间是持久的。这对于返回聚集值的函数以及需要初始设置逻辑的函数会非常有用。

dbinfo 结构是一种包含数据库和例程信息的结构，当且仅当将 DBINFO 子句包括在例程的 CREATE 语句中时，才能这些信息作为额外自变量传递至例程实现以及将这些消息作为额外自变量从例程实现中传出。

- scratchpad 和 call-type 自变量的 C 语言类型的定义。对 call-type 自变量指定 enum 类型定义。

  将针对一组值多次调用外部用户定义的函数。使用 call-type 来标识个别外部函数调用。每次调用都由可以在函数逻辑中引用的调用类型值标识。例如，对于函数的第一次调用、数据访存调用和最终调用都由特殊的调用类型。由于特定的逻辑可以与特定的调用类型相关联，因此调用类型非常有用。call-type 示例包括：FIRST 调用、FETCH 调用和 FINAL 调用。

- 可用来定义用户定义的函数 (UDF) 原型所需的标准结尾自变量的宏。结尾自变量包括 SQL-state、function-name、specific-name、diagnostic-message、scratchpad 和 call-type UDF 调用自变量。此外，还包括对这些构造进行引用的定义以及各种有效的 SQLSTATE 值。提供各种宏定义，其差别在于包含或排除 scratchpad 和 call-type 自变量。这些对应于在函数定义中使用或不使用 SCRATCHPAD 子句和 FINAL CALL 子句。

  通常，当定义用户定义的函数时，建议使用宏 SQLUDF_TRAIL_ARGS 来简化函数原型，如以下示例所示:

```
void SQL_API_FN ScalarUDF(SQLUDF_CHAR *inJob,
                          SQLUDF_DOUBLE *inSalary,
                          SQLUDF_DOUBLE *outNewSalary,
                          SQLUDF_SMALLINT *jobNullInd,
                          SQLUDF_SMALLINT *salaryNullInd,
                          SQLUDF_SMALLINT *newSalaryNullInd,
                          SQLUDF_TRAIL_ARGS)
```

- 可用来检测 SQL 自变量是否包含 null 值的宏定义。

要查看如何使用文件 sqludf.h 中所定义的各种定义、宏以及结构，请参阅 C 和 C++ 样本应用程序和例程。

C 和 C++ 例程中的参数

C 和 C++ 例程中的参数声明必须符合其中一种支持参数样式的要求以及程序类型的要求。

如果例程将使用暂存区或 dbinfo 结构，或者将具有 PROGRAM TYPE MAIN 参数接口，那么需要考虑其他详细信息，其中包括:

- 第 143 页的『C 和 C++ 例程支持的参数样式』
- 第 143 页的『C 和 C++ 例程中的参数 null 指示符』
- 第 143 页的『参数样式 SQL C 和 C++ 过程』
- 第 146 页的『参数样式 SQL C 和 C++ 函数』
- 第 148 页的『在 C 和 C++ 例程中通过值或通过引用来传递参数』
- 第 148 页的『C 和 C++ 过程结果集不需要参数』
- 第 148 页的『作为 C 或 C++ 例程参数的 dbinfo 结构』
- 第 151 页的『作为 C 或 C++ 函数参数的暂存区』
- 第 152 页的『C 和 C++ 过程的程序类型 MAIN 支持』

正确实现 C 和 C++ 例程的参数接口非常重要。只需稍微谨慎一些，即可轻松实现，从而确保根据规范来选择以及实现正确的参数样式和数据类型。

C 和 C++ 例程支持的参数样式

C 和 C++ 例程支持下列参数样式:

- SQL（受过程和函数支持; 建议）
- GENERAL（受过程支持）
- GENERAL WITH NULLS（受过程支持）

强烈建议将参数样式 SQL 用于所有 C 和 C++ 例程。此参数样式支持 NULL 值、提供标准接口来报告错误以及提供支持暂存区和调用类型。

要指定用于例程的参数样式，必须在创建例程时，在例程的 CREATE 语句中指定 PARAMETER STYLE 子句。

必须在 C 或 C++ 例程代码的实现中精确地反映此参数样式。

有关这些参数样式的更多信息，请参阅“用于将参数传递至 C 和 C++ 例程的语法”。

C 和 C++ 例程中的参数 null 指示符

如果为 C 或 C++ 例程（过程或函数）选择的参数样式要求为每个 SQL 参数指定一个 null 指示符参数（如参数样式 SQL 和 GENERAL 所要求的那样），那么会将 null 指示符作为数据类型为 SQLUDF_NULLIND* 的参数来传递。对于参数样式 GENERAL WITH NULLS，它们必须作为类型为 SQLUDF_NULLIND 的数组来传递。

在嵌入式 SQL 应用程序和例程包含文件 `sqludf.h` 中定义了此数据类型。

null 指示符参数指示对应参数值是否等价于 SQL 中的 NULL，或是否具有字面值。如果参数的 null 指示符值是 0，那么指示参数值不是 null。如果参数的 null 指示符值是 -1，那么会将参数视为具有等价于 SQL 值 NULL 的值。

使用 null 指示符时，必须在您的例程中包括执行下列操作的代码:

- 在使用输入参数的 null 指示符值前，对这些值进行检查。
- 在例程返回前，设置输出参数的 null 指示符值。

有关参数 SQL 的更多信息，请参阅:

- 第 62 页的『外部例程的参数样式』
- 『参数样式 SQL C 和 C++ 过程』
- 第 146 页的『参数样式 SQL C 和 C++ 函数』

参数样式 SQL C 和 C++ 过程

应该在 CREATE PROCEDURE 语句中使用 PARAMETER STYLE SQL 子句来创建 C 和 C++ 过程。应该在对应的过程代码实现中实现传递此参数样式的约定的参数。

过程所需的 C 和 C++ PARAMETER STYLE SQL 特征符实现遵循此格式:

```
SQL_API_RC SQL_API_FN function-name (
                                      SQL-arguments,
                                      SQL-argument-inds,
                                      sqlstate,
```

```
                                    routine-name,
                                    specific-name,
                                    diagnostic-message )
```

SQL_API_RC SQL_API_FN

SQL_API_RC 和 SQL_API_FN 可以随支持的操作系统而改变的宏，用来指定 C 或 C++ 过程的返回类型和调用约定。对于 C 和 C++ 例程，必须使用这些宏。在嵌入式 SQL 应用程序和例程包含文件 sqlsystm.h 中声明这些宏。

function-name

代码文件中 C 或 C++ 函数的名称。此值不必与对应 CREATE PROCEDURE 语句中所指定的过程名称相同。但是，此值与库名组合使用，且必须在 EXTERNAL NAME 子句中指定此值，以在库中标识所要使用的正确函数入口点。对于 C++ 例程，C++ 编译器将对入口点名称应用类型声明。您或者应该在 EXTERNAL NAME 子句中指定已进行类型装饰的名称，或者应该在用户代码中将入口点定义为 extern "C"。必须显式地导出函数名。

SQL-arguments

对应于 CREATE PROCEDURE 语句中所指定 SQL 参数集的 C 或 C++ 自变量。会使用个别指针值来传递 IN、OUT 和 INOUT 方式参数。

SQL-argument-inds

对应于 CREATE PROCEDURE 语句中所指定 SQL 参数集的 C 或 C++ null 指示符。对于每个 IN、OUT 和 INOUT 方式参数，必须有相关联的 null 指示符参数。可以将 null 指示符作为类型为 SQLUDF_NULLIND 的个别自变量来传递，或者作为单一 null 指示符数组（定义为 SQLUDF_NULLIND*）的一部分来传递。

sqlstate

例程用来指示警告或错误状况的输入/输出参数值。通常，使用此自变量来指定用户定义 SQLSTATE 值（对应于可传递回至调用者的错误或警告）。SQLSTATE 值的格式为 38xxx，其中 xxx 是任何适用于用户定义的 SQLSTATE 错误值的数值。SQLSTATE 值的格式为 01Hxx，其中 xx 是任何适用于用户定义的 SQLSTATE 警告值的数值。

routine-name

包含限定例程名称的输入参数值。此值由 DB2 数据库系统生成，且以 <schema-name>.<routine-name> 格式传递至例程，其中 <schema-name> 和 <routine-name> 分别对应于 SYSCAT.ROUTINES 目录视图中例程的 ROUTINESCHEMA 列值和 ROUTINENAME 列值。如果多个不同的例程定义使用单个例程实现，那么此值会很有用。将例程定义名称传递至例程时，可以根据所使用的定义，有条件地执行逻辑。构造诊断信息（其中包括错误消息）时，或写入日志文件时，例程名称也会很有用。

specific-name

包含唯一例程特定名称的输入参数值。此值由 DB2 数据库系统生成且传递至例程。此值对应于 SYSCAT.ROUTINES 视图中例程的 SPECIFICNAME 列值。此值和例程名的用法相同。

诊断消息

例程选择性地使用以将消息文本返回给调用应用程序或例程的输出参数值。此参数预期用作 SQLSTATE 自变量的补充。可以使用此参数来指定用户定义的错误消息，以附加用户定义的 SQLSTATE 值，此值可以向例程的调用者提供更详细的诊断错误或警告信息。

注: 要简化 C 和 C++ 过程特征符的编写，可以在过程特征符中使用 sqludf.h 中所定义的宏定义 SQLUDF_TRAIL_ARGS 来取代个别自变量，以实现非 SQL 数据类型自变量。

下列是 C 或 C++ 过程实现示例，此过程可接受单一输入参数，然后返回单一输出参数和一个结果集:

```
/*****************************************************************
 Routine:   cstp

  Purpose:  Returns an output parameter value based on an input
            parameter value

            Shows how to:
             - define a procedure using PARAMETER STYLE SQL
             - define NULL indicators for the parameter
             - execute an SQL statement
             - how to set a NULL indicator when parameter is
               not null

  Parameters:

  IN:       inParm
  OUT:      outParm

            When PARAMETER STYLE SQL is defined for the routine
            (see routine registration script spcreate.db2), in
            addition to the parameters passed during invocation,
            the following arguments are passed to the routine
            in the following order:

             - one null indicator for each IN/INOUT/OUT parameter
               ordered to match order of parameter declarations
             - SQLSTATE to be returned to DB2 (output)
             - qualified name of the routine (input)
             - specific name of the routine (input)
             - SQL diagnostic string to return an optional
               error message text to DB2 (output)

            See the actual parameter declarations below to see
            the recommended datatypes and sizes for them.

            CODE TIP:
            --------
            Instead of coding the 'extra' parameters:
              sqlstate, qualified name of the routine,
              specific name of the routine, diagnostic message,
            a macro SQLUDF_TRAIL_ARGS can be used instead.
            This macro is defined in DB2 include file sqludf.h

            TIP EXAMPLE:
            ------------
            The following is equivalent to the actual prototype
            used that makes use of macro definitions included in
            sqludf.h.  The form actually implemented is simpler
            and removes datatype concerns.

   extern "C"  SQL_API_RC SQL_API_FN OutLanguage(
                                                 sqlint16 *inParm,
                                                 double *outParm,
                                                 sqlint16 *inParmNullInd,
                                                 sqlint16 *outParmNullInd,
                                                 char sqlst[6],
                                                 char qualName[28],
                                                 char specName[19],
                                                 char diagMsg[71])
```

```
                                                )
*************************************************************/

extern "C" SQL_API_RC SQL_API_FN cstp ( sqlint16 *inParm,
                                        double *outParm,
                                        SQLUDF_NULLIND *inParmNullInd,
                                        SQLUDF_NULLIND *outParmNullInd,
                                        SQLUDF_TRAIL_ARGS )
{
  EXEC SQL INCLUDE SQLCA;

  EXEC SQL BEGIN DECLARE SECTION;
     sqlint16 sql_inParm;
  EXEC SQL END DECLARE SECTION;

  sql_inParm = *inParm;

  EXEC SQL DECLARE cur1 CURSOR FOR
   SELECT value
   FROM table01
    WHERE index = :sql_inParm;

  *outParm = (*inParm) + 1;
  *outParmNullInd = 0;

  EXEC SQL OPEN cur1;

  return (0);
}
```

下列是此过程的对应 CREATE PROCEDURE 语句:

```
CREATE PROCEDURE cproc( IN inParm INT, OUT outParm INT )
LANGUAGE c
PARAMETER STYLE sql
DYNAMIC RESULT SETS 1
FENCED
THREADSAFE
RETURNS NULL ON NULL INPUT
EXTERNAL NAME 'c_rtns!cstp'
```

前面的语句假设 C 或 C++ 过程实现是在库文件 c_rtns 和函数 cstp 中。

参数样式 SQL C 和 C++ 函数

应该在 CREATE FUNCTION 语句中使用 PARAMETER STYLE SQL 子句来创建 C 和 C++ 用户定义的函数。应该在相应的源代码实现中实现用于传递此参数样式的约定的参数。

用户定义的函数所需的 C 和 C++ PARAMETER STYLE SQL 特征符实现遵循此格式:

```
SQL_API_RC SQL_API_FN function-name ( SQL-arguments,
 SQL-argument-inds,
 SQLUDF_TRAIL_ARGS )
```

SQL_API_RC SQL_API_FN

SQL_API_RC 和 SQL_API_FN 可以随支持的操作系统而改变的宏，用来指定 C 或 C++ 用户定义的函数的返回类型和调用约定。对于 C 和 C++ 例程，必须使用这些宏。在嵌入式 SQL 应用程序和例程包含文件 sqlsystm.h 中声明这些宏。

function-name

代码文件中 C 或 C++ 函数的名称。此值不必与对应 CREATE FUNCTION 语句中所指定的函数名称相同。但是，此值与库名组合使用，且必须在 EXTER-

NAL NAME 子句中指定此值，以在库中标识所要使用的正确函数入口点。对于 C++ 例程，C++ 编译器将对入口点名称应用类型声明。需要在 EXTERNAL NAME 子句中指定类型装饰名称，或者源代码文件中的函数声明应该以 extern "C" 为前缀，如下列示例中所示：extern ″C″ SQL_API_RC SQL_API_FN OutLanguage(char *, sqlint16 *, char *, char *, char *, char *);

SQL-arguments

对应于 CREATE FUNCTION 语句中所指定 SQL 参数集的 C 或 C++ 自变量。

SQL-argument-inds

对于每个 SQL 自变量，需要一个 null 指示符参数来指定是否打算在例程实现中将参数值解释为 SQL 中的 NULL 值。必须使用数据类型 SQLUDF_NULLIND 来指定 null 指示符。在嵌入式 SQL 例程包含文件 sqludf.h 中定义此数据类型。

SQLUDF_TRAIL_ARGS

嵌入式 SQL 例程包含文件 sqludf.h 中定义的宏在展开后，会定义完整参数样式 SQL 特征符所需的其他结尾自变量。可以使用两个宏：SQLUDF_TRAIL_ARGS 和 SQLUDF_TRAIL_ARGS_ALL。SQLUDF_TRAIL_ARGS 在展开时，如 sqludf.h 中所定义，等价于增加下列例程自变量:

```
SQLUDF_CHAR *sqlState,
SQLUDF_CHAR qualName,
SQLUDF_CHAR specName,
SQLUDF_CHAR *sqlMessageText,
```

通常，不需要这些自变量，或者将这些自变量用作用户定义的函数逻辑的一部分。它们表示要传递回至函数调用者的输出 SQLSTATE 值、输入标准函数名、输入函数特定名称以及要与 SQLSTATE 一起返回的输出消息文本。SQLUDF_TRAIL_ARGS_ALL 在展开时，如 sqludf.h 中所定义，等价于增加下列例程自变量:

```
SQLUDF_CHAR  qualName,
SQLUDF_CHAR  specName,
SQLUDF_CHAR  sqlMessageText,
SQLUDF_SCRAT *scratchpad
SQLUDF_CALLT *callType
```

如果 UDF CREATE 语句包含 SCRATCHPAD 子句或 FINAL CALL 子句，那么必须使用宏 SQLUDF_TRAIL_ARGS_ALL。除随 SQLUDF_TRAIL_ARGS 提供的自变量之外，此宏还包含 scratchpad 结构的指针，以及调用类型值。

下列是简单 C 或 C++ UDF 的示例，该 UDF 在输出参数中返回其两个输入参数值之积的值:

```
SQL_API_RC SQL_API_FN product ( SQLUDF_DOUBLE *in1,
                                SQLUDF_DOUBLE *in2,
                                SQLUDF_DOUBLE *outProduct,
                                SQLUDF_NULLIND *in1NullInd,
                                SQLUDF_NULLIND *in2NullInd,
                                SQLUDF_NULLIND *productNullInd,
                                SQLUDF_TRAIL_ARGS )
{

  /* Check that input parameter values are not null
     by checking the corresponding null indicator values
         0  : indicates parameter value is not NULL
```

```
        -1 : indicates parameter value is NULL

    If values are not NULL, calculate the product.
    If values are NULL, return a NULL output value. */

  if ((*in1NullInd != -1) &&
       *in2NullInd != -1))
  {
    *outProduct = (*in1) * (*in2);
    *productNullInd = 0;
  }
  else
  {
    *productNullInd = -1;
  }
  return (0);
}
```

可用来创建此 UDF 的对应 CREATE FUNCTION 语句可能如下所示:

```
CREATE FUNCTION product( in1 DOUBLE, in2 DOUBLE )
  RETURNS DOUBLE
  LANGUAGE C
  PARAMETER STYLE SQL
  NO SQL
  FENCED THREADSAFE
  DETERMINISTIC
  RETURNS NULL ON NULL INPUT
  NO EXTERNAL ACTION
  EXTERNAL NAME 'c_rtns!product'
```

前面的 SQL 语句假设 C 或 C++ 函数是在函数目录的库文件 c_rtns 中。

在 C 和 C++ 例程中通过值或通过引用来传递参数

对于 C 和 C++ 例程，必须通过引用将参数值传递至使用指针的例程。

这对于仅输入、输入/输出以及输出参数是必需的（通过引用）。

null 指示符参数也必须通过引用传递至使用指针的例程。

注: DB2 数据库系统控制所有参数的内存分配，以及维护对传入或传出例程的所有参数的 C 或 C++ 引用。不需要分配或释放与例程参数和 null 指示符相关联的内存。

C 和 C++ 过程结果集不需要参数

过程的 CREATE PROCEDURE 语句特征符中或关联的过程实现中不需要参数，即可将结果集传回给调用者。

使用游标来返回 C 过程所返回的结果集。

有关从 LANGUAGE C 过程中返回的结果集的更多信息，请参阅:

- 第 178 页的『从 C 和 C++ 过程中返回结果集』

作为 C 或 C++ 例程参数的 dbinfo 结构

dbinfo 结构是一种包含数据库和例程信息的结构，当且仅当将 DBINFO 子句包括在例程的 CREATE 语句中时，才能这些信息作为额外自变量传递至例程实现以及将这些消息作为额外自变量从例程实现中传出。

通过使用 sqludf_dbinfo 结构，dbinfo 结构在 LANGUAGE C 例程中受支持。在 DB2 数据库系统包含文件 sqludf.h（位于 sqllib\include 目录）中定义了 C 结构。

sqludf_dbinfo 结构的定义如下所示:

```
SQL_STRUCTURE sqludf_dbinfo
{
   unsigned short  dbnamelen;                      /* Database name length     */
   unsigned char   dbname[SQLUDF_MAX_IDENT_LEN];   /* Database name            */
   unsigned short  authidlen;                      /* Authorization ID length  */
   unsigned char   authid[SQLUDF_MAX_IDENT_LEN];   /* Authorization ID         */
   union db_cdpg   codepg;                         /* Database code page       */
   unsigned short  tbschemalen;                    /* Table schema name length */
   unsigned char   tbschema[SQLUDF_MAX_IDENT_LEN]; /* Table schema name        */
   unsigned short  tbnamelen;                      /* Table name length        */
   unsigned char   tbname[SQLUDF_MAX_IDENT_LEN];   /* Table name               */
   unsigned short  colnamelen;                     /* Column name length       */
   unsigned char   colname[SQLUDF_MAX_IDENT_LEN];  /* Column name              */
   unsigned char   ver_rel[SQLUDF_SH_IDENT_LEN];   /* Database version/release */
   unsigned char   resd0[2];                       /* Alignment                */
   sqluint32       platform;                       /* Platform                 */
   unsigned short  numtfcol;                       /* # of entries in TF column*/
                                                   /* List array               */
   unsigned char   resd1[2];                       /* Reserved                 */
   sqluint32       procid;                         /* Current procedure ID     */
   unsigned char   resd2[32];                      /* Reserved                 */
   unsigned short  *tfcolumn;                      /* Tfcolumn to be allocated */
                                                   /* dynamically if a table   */
                                                   /* function is defined;     */
                                                   /* else a NULL pointer      */
   char            *appl_id;                       /* Application identifier   */
   sqluint32       dbpartitionnum;                 /* Database partition number*/
                                                   /* where routine executed   */
   sqluint32       numdbpartitions;                /* number of entries in     */
                                                   /* dbpartitions array       */
   sqluint32       *dbpartitions;                  /* allocated dynamically if */
                                                   /* routine is processed in  */
                                                   /* parallel. Otherwise, this*/
                                                   /* will be a null pointer.  */
   unsigned char   resd3[16];                      /* Reserved                 */
};
```

虽然可能并非 dbinfo 结构中的所有字段都在例程中很有用，但在构造诊断错误消息信息时，此结构的字段中的若干个值可能很有用。例如，如果例程中发生错误，那么返回数据库名、数据库名长度、数据库代码页、当前授权标识以及当前授权标识的长度可能很有用。

要在 LANGUAGE C 例程实现中引用 sqludf_dbinfo 结构，请执行下列操作:

- 在定义例程的 CREATE 语句中添加 DBINFO 子句。
- 在包含例程实现的文件的顶部包括 sqludf.h 头文件。
- 将类型为 sqludf_dbinfo 的参数添加至使用的参数样式所指定位置中的例程特征符。

使用 dbinfo 结构的 C 过程的示例

以下示例是带有 PARAMETER STYLE GENERAL 的 C 过程，用于说明如何使用 dbinfo 结构。

下列是过程的 CREATE PROCEDURE 语句。过程实现位于名为 spserver 的库文件中，该文件包含 EXTERNAL NAME 子句指定的名为 DbinfoExample 的 C 函数:

```
  CREATE PROCEDURE DBINFO_EXAMPLE (IN job CHAR(8),
                                   OUT salary DOUBLE,
                                   OUT dbname CHAR(128),
                                   OUT dbversion CHAR(8),
                                   OUT errorcode INTEGER)
  DYNAMIC RESULT SETS 0
  LANGUAGE C
  PARAMETER STYLE GENERAL
  DBINFO
  FENCED NOT THREADSAFE
  READS SQL DATA
  PROGRAM TYPE SUB
  EXTERNAL NAME 'spserver!DbinfoExample'@
```

下列是对应于过程定义的 C 过程实现:

```
/*****************************************************************
 Routine:    DbinfoExample

   IN:       inJob         - a job type, used in a SELECT predicate
   OUT:      salary        - average salary of employees with job injob
             dbname        - database name retrieved from DBINFO
             dbversion     - database version retrieved from DBINFO
             outSqlError   - sqlcode of error raised (if any)
             sqludf_dbinfo - pointer to DBINFO structure

  Purpose:  This routine takes in a job type and returns the
            average salary of all employees with that job, as
            well as information about the database (name,
            version of database).  The database information
            is retrieved from the dbinfo object.

  Shows how to:
              - define IN/OUT parameters in PARAMETER STYLE GENERAL
              - declare a parameter pointer to the dbinfo structure
              - retrieve values from the dbinfo structure
*****************************************************************/
SQL_API_RC SQL_API_FN DbinfoExample(char inJob[9],
                                    double *salary,
                                    char dbname[129],
                                    char dbversion[9],
                                    sqlint32 *outSqlError,
                                    struct sqludf_dbinfo * dbinfo
                                    )
{
  /* Declare a local SQLCA */
  struct sqlca sqlca;

  EXEC SQL WHENEVER SQLERROR GOTO return_error;

  /* SQL host variable declaration section */
  /* Each host variable names must be unique within a code
     file, or the the precompiler raises SQL0307 error */
  EXEC SQL BEGIN DECLARE SECTION;
  char dbinfo_injob[9];
  double dbinfo_outsalary;
  sqlint16 dbinfo_outsalaryind;
  EXEC SQL END DECLARE SECTION;

  /* Initialize output parameters - se strings to NULL */
  memset(dbname, '\0', 129);
  memset(dbversion, '\0', 9);
  *outSqlError = 0;

  /* Copy input parameter into local host variable */
  strcpy(dbinfo_injob, inJob);
```

```
  EXEC SQL SELECT AVG(salary) INTO:dbinfo_outsalary
             FROM employee
               WHERE job =:dbinfo_injob;

  *salary = dbinfo_outsalary;

  /* Copy values from the DBINFO structure into the output parameters
     You must explicitly null-terminate the strings.
     Information such as the database name, and the version of the
     database product can be found in the DBINFO structure as well as
     other information fields. */

  strncpy(dbname, (char *)(dbinfo->dbname), dbinfo->dbnamelen);
  dbname[dbinfo->dbnamelen] = '\0';
  strncpy(dbversion, (char *)(dbinfo->ver_rel), 8);
  dbversion[8] = '\0';

  return 0;

  /* Copy SQLCODE to OUT parameter if SQL error occurs */

  return_error:
  {
    *outSqlError = SQLCODE;
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    return 0;
  }
} /* DbinfoExample function */
```

作为 C 或 C++ 函数参数的暂存区

在 C 和 C++ 例程中，通过使用 sqludf_scrat 结构来支持 scratchpad 结构（用于在每个 UDF 输入值的调用之间存储 UDF 值）。

此 C 结构是在 DB2 数据库系统包含文件 sqludf.h 中定义的。

要引用 sqludf_scrat 结构，请在包含 C 或 C++ 函数实现的文件顶部包括 sqludf.h 头文件，并在例程实现的特征符中使用 SQLUDF_TRAIL_ARGS_ALL 宏。

下列示例演示一个包括类型为 SQLUDF_TRAIL_ARGS_ALL 的参数的 C 标量函数实现:

```
  #ifdef __cplusplus
  extern "C"
  #endif
  void SQL_API_FN ScratchpadScUDF(SQLUDF_INTEGER *outCounter,
                                  SQLUDF_SMALLINT *counterNullInd,
                                  SQLUDF_TRAIL_ARGS_ALL)
  {
    struct scalar_scratchpad_data *pScratData;

    /* SQLUDF_CALLT and SQLUDF_SCRAT are */
    /* parts of SQLUDF_TRAIL_ARGS_ALL */

    pScratData = (struct scalar_scratchpad_data *)SQLUDF_SCRAT->data;
    switch (SQLUDF_CALLT)
    {
      case SQLUDF_FIRST_CALL:
        pScratData->counter = 1;
        break;
      case SQLUDF_NORMAL_CALL:
        pScratData->counter = pScratData->counter + 1;
        break;
      case SQLUDF_FINAL_CALL:
        break;
```

```
    }

    *outCounter = pScratData->counter;
    *counterNullInd = 0;
  } /* ScratchpadScUDF */
```

SQLUDF_TRAIL_ARGS_ALL 宏可展开以定义其他参数值，其中包括一个 SQLUDF_SCRAT 参数值（定义可用作暂存区的缓冲区参数）。如果针对一组值调用标量函数，那么每次调用标量函数时，都会将缓冲区作为参数传递至函数。可以访问缓冲区

SQLUDF_TRAIL_ARGS_ALL 宏值也定义另一个参数 SQLUDF_CALLT。使用此参数来指示调用类型值。可以使用调用类型值来标识是第一次、最后一次还是处理过程中针对一组值调用函数。

C 和 C++ 过程的程序类型 MAIN 支持

虽然通常建议将缺省 PROGRAM TYPE 子句值 SUB 用于 C 过程，但 CREATE PROCEDURE 语句（其中 LANGUAGE 子句值是 C）支持 PROGRAM TYPE 子句值 MAIN。

参数个数超过 90 的例程需要 PROGRAM TYPE 子句值 MAIN。

如果指定 PROGRAM TYPE MAIN 子句，那么必须使用与 C 源代码文件中主例程的缺省样式一致的特征符来实现过程。这并不意味着必须使用函数 main 来实现例程，而是必须以通常与使用典型 C 编程 argc 和 argv 自变量的缺省类型主例程应用程序实现相关联的格式来传递参数。

下列是遵循 PGRAM TYPE MAIN 规范的 C 或 C++ 例程特征符示例:

```
SQL_API_RC SQL_API_FN functionName(int argc, char **argv)
{
  ...
}
```

由 argc 值指定函数的总自变量数。会将自变量值作为 argv 数组中的数组元素来传递。自变量的数目和顺序取决于 CREATE PROCEDURE 语句中所指定的 PARAMETER STYLE 子句值。

例如，设想 C 过程（指定具有 PROGRAM TYPE MAIN 样式和建议的 PARAMETER STYLE SQL）的下列 CREATE PROCEDURE 语句:

```
CREATE PROCEDURE MAIN_EXAMPLE (
  IN job CHAR(8),
  OUT salary DOUBLE)
SPECIFIC CPP_MAIN_EXAMPLE
DYNAMIC RESULT SETS 0
NOT DETERMINISTIC
LANGUAGE C
PARAMETER STYLE SQL
NO DBINFO
FENCED NOT THREADSAFE
READS SQL DATA
PROGRAM TYPE MAIN
EXTERNAL NAME 'spserver!MainExample'@
```

下列是对应于此 CREATE PROCEDURE 语句的例程特征符实现:

```
//*****************************************************
//  Stored Procedure: MainExample
//
//  SQL parameters:
//     IN:      argv[1] - job    (char[8])
//     OUT:     argv[2] - salary (double)
//*****************************************************
SQL_API_RC SQL_API_FN MainExample(int argc, char **argv)
{
  ...
}
```

因为使用 PARAMETER STYLE SQL，所以除过程调用时传递的 SQL 参数值之外，还会将该样式所需的其他参数传递至例程。

可通过引用源代码中相关 argv 数组元素来访问参数值。对于以前给定示例，argc 和 argv 数组元素包含下列值:

```
argc   : Number of argv array elements
argv[0]: The function name
argv[1]: Value of parameter job (char[8], input)
argv[2]: Value of parameter salary (double, output)
argv[3]: null indicator for parameter job
argv[4]: null indicator for parameter salary
argv[5]: sqlstate (char[6], output)
argv[6]: qualName (char[28], output)
argv[7]: specName (char[19], output)
argv[8]: diagMsg (char[71], output)
```

C 和 C++ 例程支持的 SQL 数据类型

列示例程的 SQL 数据类型与 C 数据类型之间的受支持映射。

与每种 C/C++ 数据类型一起提供的就是 sqludf.h 中所定义的相应类型。

表 16. 映射至 C/C++ 声明的 SQL 数据类型

SQL 列类型	C/C++ 数据类型	SQL 列类型描述
SMALLINT	sqlint16 SQLUDF_SMALLINT	16 位带符号整数
INTEGER	sqlint32 SQLUDF_INTEGER	32 位带符号整数
BIGINT	sqlint64 SQLUDF_BIGINT	64 位带符号整数
REAL FLOAT(*n*)，其中 1<=*n*<=24	float SQLUDF_REAL	单精度浮点数
DOUBLE FLOAT FLOAT(*n*)，其中 25<=*n*<=53	double SQLUDF_DOUBLE	双精度浮点数
DECIMAL(*p*, *s*)	不支持。	要传递十进制值，请将参数定义为能够从 DECIMAL 强制转换的数据类型（例如 CHAR 或 DOUBLE）并显式地将自变量强制转换为此类型。

表 16. 映射至 C/C++ 声明的 SQL 数据类型 (续)

SQL 列类型	C/C++ 数据类型	SQL 列类型描述
CHAR(*n*)	char[*n*+1], 其中 *n* 应该大到足以存放该数据 1<=*n*<=254 SQLUDF_CHAR	以 null 结束的固定长度字符串
CHAR(*n*) FOR BIT DATA	char[*n*], 其中 *n* 应该大到足以存放该数据 1<=*n*<=254 SQLUDF_CHAR	不是以 null 结束的固定长度字符串
VARCHAR(*n*)	char[*n*+1], 其中 *n* 应该大到足以存放该数据 1<=*n*<=32 672 SQLUDF_VARCHAR	以 null 结束的变长字符串

VARCHAR(*n*) FOR BIT DATA	struct {
	sqluint16 length;
	char[*n*]
	}
	
	1<=*n*<=32 672
	
	SQLUDF_VARCHAR_FBD
		并非以 null 结束的变长字符串

LONG VARCHAR	struct {
	sqluint16 length;
	char[*n*]
	}
	
	1<=*n*<=32 700
	
	SQLUDF_LONG
		并非以 null 结束的变长字符串

CLOB(*n*)	struct {
	sqluint32 length;
	char data[*n*];
	}
	
	1<=*n*<=2 147 483 647
	
	SQLUDF_CLOB
		带有 4 字节字符串长度指示符而且并非以 null 结束的变长字符串

表 16. 映射至 C/C++ 声明的 SQL 数据类型 (续)

SQL 列类型	C/C++ 数据类型	SQL 列类型描述

BLOB(*n*)	struct {
	sqluint32 length;
	char data[*n*];
	}
	
	1<=*n*<=2 147 483 647
	
	SQLUDF_BLOB
		带有 4 字节字符串长度指示符而且并非以 null 结束的变长二进制字符串

DATE	char[11] SQLUDF_DATE	具有下列格式的以 null 结束的字符串: yyyy-mm-dd
TIME	char[9] SQLUDF_TIME	具有下列格式的以 null 结束的字符串: hh.mm.ss
TIMESTAMP	char[20] - char[33] SQLUDF_STAMP	具有下列格式的以 null 结束的字符串: yyyy-mm-dd-hh.mm.ss.nnnnnnnnnnnn 字符串的长度可达 19-32 个字节，取决于所指定的小数秒数。（可选）可以指定 TIMESTAMP 数据类型的小数秒部分，以使时间戳记精度为 0-12 位。 例如: (VALUES(CURRENT TIMESTAMP(0)) 1 ------------------- 2008-07-09-14.48.36 1 record(s) selected. LENGTH (VALUES(CURRENT TIMESTAMP(0)) 1 ------------------- 19 1 record(s) selected. (VALUES(CURRENT TIMESTAMP(12)) 1 ------------------- 2008-07-09-14.48.36.123456789012 1 record(s) selected. LENGTH (VALUES(CURRENT TIMESTAMP(0)) 1 ------------------- 32 将时间戳记值指定给秒小数位数不同的时间戳记变量时，该值将被截断或者填充 0 以便与时间戳记变量的格式匹配。
LOB LOCATOR	sqluint32 SQLUDF_LOCATOR	32 位带符号整数

表 16. 映射至 C/C++ 声明的 SQL 数据类型 (续)

SQL 列类型	C/C++ 数据类型	SQL 列类型描述
GRAPHIC(*n*)	sqldbchar[*n*+1], 其中 *n* 应该大到足以存放该数据 1<=*n*<=127 SQLUDF_GRAPH	以 null 结束的双字节固定长度字符串
VARGRAPHIC(*n*)	sqldbchar[*n*+1], 其中 *n* 应该大到足以存放该数据 1<=*n*<=16 336 SQLUDF_GRAPH	以 null 结束的可变长度双字节字符串

LONG VARGRAPHIC	struct {
	sqluint16 length;
	sqldbchar[*n*]
	}
	
	1<=*n*<=16 350
	
	SQLUDF_LONGVARG
		并非以 null 结束的可变长度双字节字符串

DBCLOB(*n*)	struct {
	sqluint32 length;
	sqldbchar data[*n*];
	}
	
	1<=*n*<=1 073 741 823
	
	SQLUDF_DBCLOB
		带有 4 字节字符串长度指示符而且并非以 null 结束的变长字符串

XML AS CLOB	struct {
	sqluint32 length;
	char data[*n*];
	}
	
	1<=*n*<=2 147 483 647
	
	SQLUDF_CLOB
		不是以 null 结束的可变长度序列化字符串（带有 4 字节字符串长度指示符）。

注: 在使用 C 或 C++ 实现的外部例程中，只能将 XML 数据类型作为 CLOB 数据类型来实现。

注: 在使用 WCHARTYPE NOCONVERT 选项进行预编译时，下列数据类型只适用于 DBCS 或 EUC 环境:

- GRAPHIC(*n*)
- VARGRAPHIC(*n*)
- LONG VARGRAPHIC
- DBCLOB(*n*)

C 和 C++ 例程中的 SQL 数据类型处理

本节识别例程参数及结果的有效类型，并且它指定应该如何在您的 C 或 C++ 语言例程中定义对应的自变量。必须将例程中的所有自变量作为指针传递至相应的数据类型。

请注意，如果您使用 sqludf.h 包含文件及其中定义的类型，那么可以自动生成适合于不同数据类型和编译器的语言变量及结构。例如，对于 BIGINT，您可以使用 SQLUDF_BIGINT 数据类型来隐藏 BIGINT 表示所需的类型在不同编译器之间的差别。

它是例程的 CREATE 语句（控制自变量值的格式）中所定义的每个参数的数据类型。可能需要提升自变量的数据类型，才能获取相应格式的值。DB2会自动对自变量值执行此类提升操作。但是，如果在例程代码中指定的数据类型不正确，那么会出现无法预测的行为，例如丢失数据或异常结束。

对于标量函数或方法的结果，它是 CREATE FUNCTION 语句的 CAST FROM 子句（定义格式）中所指定的数据类型。如果不存在任何 CAST FROM 子句，那么 RETURNS 子句中所指定的数据类型会定义格式。

在下列示例中，CAST FROM 子句的存在意味着例程主体会返回 SMALLINT，并且 DB2 会将值强制类型转换成 INTEGER，然后再将它传递至执行函数引用的语句:

```
... RETURNS INTEGER CAST FROM SMALLINT ...
```

在此情况下，必须编写例程才能生成 SMALLINT（如本节后续部分所定义）。请注意，CAST FROM 数据类型*必须可强制类型转换* 成 RETURNS 数据类型，因此无法任意选择另一种数据类型。

下列是 SQL 类型及其 C/C++ 语言表示的列表。它包含有关每种类型对于参数或结果是否有效的信息。此外，还包含这些类型在 C 或 C++ 语言例程中如何表示为自变量定义的示例:

- SMALLINT

  有效。在 C 中表示为 SQLUDF_SMALLINT 或 sqlint16。

  示例:

  ```
  sqlint16    *arg1;          /* example for SMALLINT */
  ```

  定义整型例程参数时，请考虑使用 INTEGER（而不是 SMALLINT），因为 DB2 不会将 INTEGER 自变量提升成 SMALLINT。例如，假设您按如下所示定义 UDF:

  ```
  CREATE FUNCTION SIMPLE(SMALLINT)...
  ```

  如果使用 INTEGER 数据来调用 SIMPLE 函数数据 (... SIMPLE(1)...)，那么您将收到指示找不到函数的 SQLCODE -440 (SQLSTATE 42884) 错误，而且此函数的最终用户可能无法理解此消息的原因。在前面的示例中，1 是一个 INTEGER，因此您可以将它强制类型转换成 SMALLINT 或将参数定义为 INTEGER。

- INTEGER 或 INT

  有效。在 C 中表示为 SQLUDF_INTEGER 或 sqlint32。必须使用 #include sqludf.h 或 #include sqlsystm.h 来捡取此定义。

  示例:

  ```
  sqlint32 *arg2;          /* example for INTEGER */
  ```

- BIGINT

  有效。在 C 中表示为 SQLUDF_BIGINT 或 sqlint64。

  示例:

  ```
  sqlint64 *arg3;          /* example for INTEGER */
  ```

  DB2定义了 sqlint64 C 语言类型，以克服 64 位带符号整数定义在编译器和操作系统之间的差别。必须使用 #include sqludf.h 或 #include sqlsystm.h 来捡取此定义。

- REAL 或 FLOAT(*n*) 其中 1 <= *n* <= 24

  有效。在 C 中表示为 SQLUDF_REAL 或 float。

  示例:

  ```
  float *result;          /* example for REAL */
  ```

- DOUBLE、DOUBLE PRECISION、FLOAT 或 FLOAT(*n*) 其中 25 <= *n* <= 53

  有效。在 C 中表示为 SQLUDF_DOUBLE 或 double。

  示例:

  ```
  double  *result;        /* example for DOUBLE   */
  ```

- DECIMAL(p,s) 或 NUMERIC(p,s)

  无效 因为没有 C 语言表示。如果要传递小数值，那么必须将参数定义为可从 DECIMAL 强制类型转换的数据类型（例如，CHAR 或 DOUBLE），并且显式地将自变量强制类型转换成此类型。如果是 DOUBLE，那么您不需要显式地将 DECIMAL 自变量强制类型转换成 DOUBLE 参数，因为 DB2 会自动提升此自变量。

  示例:

  假设您具有两列（类型为 DECIMAL(5,2) 的 WAGE，类型为 DECIMAL(4,1) 的 HOURS），并且您希望编写 UDF 以根据工资、工作小时数以及其他一些因素来计算周薪。UDF 可能如下所示:

  ```
  CREATE FUNCTION WEEKLY_PAY (DOUBLE, DOUBLE, ...)
         RETURNS DECIMAL(7,2) CAST FROM DOUBLE
         ...;
  ```

  对于前面的 UDF，前两个参数对应于工资和小时数。您可以在 SQL SELECT 语句中调用 UDF WEEKLY_PAY，如下所示:

  ```
  SELECT WEEKLY_PAY (WAGE, HOURS, ...) ...;
  ```

  请注意，不需要执行显式强制类型转换，因为 DECIMAL 自变量可强制类型转换成 DOUBLE。

  此外，您还可以使用 CHAR 自变量来定义 WEEKLY_PAY，如下所示:

  ```
  CREATE FUNCTION WEEKLY_PAY (VARCHAR(6), VARCHAR(5), ...)
         RETURNS DECIMAL (7,2) CAST FROM VARCHAR(10)
         ...;
  ```

  您可以对其进行调用，如下所示:

  ```
  SELECT WEEKLY_PAY (CHAR(WAGE), CHAR(HOURS), ...) ...;
  ```

请注意，需要显式强制类型转换，因为 DECIMAL 自变量无法提升为 VARCHAR。

使用浮点参数的优点是可以很容易地对例程中的值执行算术运算；使用字符参数的优点是始终可以精确表示小数值。这对于浮点值而言并非始终如此。

- 带有或不带 FOR BIT DATA 修饰符的 CHAR(n) 或 CHARACTER(n)。

  有效。在 C 中表示为 SQLUDF_CHAR 或 char...[*n*+1]（这是以 null 结束的 C 字符串）。

  示例:

  ```
  char     arg1[14];      /* example for CHAR(13)   */
  char    *arg1;          /* also acceptable */
  ```

  数据类型为 CHAR 的输入例程参数始终自动以 null 结束。对于 CHAR(n) 输入参数，其中 *n* 是 CHAR 数据类型的长度，系统会将 *n* 字节的数据移至例程实现中的缓冲区，并将 *n* + 1 位置中的字符设为 ASCII null 终止符 (X'00')。

  例程必须显式地以 null 结束数据类型为 CHAR 的过程输出参数和函数返回值。对于由 RETURNS 子句（例如 RETURNS CHAR(n)）指定的 UDF 返回值，或指定为 CHAR(n) 的过程输出参数（其中 n 是 CHAR 值的长度），缓冲区的前 *n*+1 个字节中必须存在一个 null 终止符。如果在缓冲区的前 *n*+1 个字节中找到 null 终止符，那么剩余字节（最多可达到字节 *n*）都会设为 ASCII 空白字符 X'20'）。如果找不到任何 null 终止符，那么会产生 SQL 错误 (SQLSTATE 39501)。

  对于数据类型为 CHAR 且也指定 FOR BIT DATA 子句（指示以二进制格式操作数据）的过程输入和输出参数或函数返回值，不会使用 null 终止符来指示参数值的末尾。对于 RETURNS CHAR*n*) FOR BIT DATA 函数返回值或 CHAR(*n*) FOR BIT DATA 输出参数，会改写缓冲区的前 *n* 个字节，而不管字符串 null 终止符在前 *n* 个字节中的出现次数。缓冲区中标识的 null 终止符会忽略为 null 终止符，且只视为一般数据。

  在操作 FOR BIT DATA 值的例程中使用一般 C 字符串处理函数时需格外小心，因为其中的许多函数会查找 null 终止符来定界字符串自变量，并且 null 终止符 (X'00') 可以合理地出现在 FOR BIT DATA 值中间。对 FOR BIT DATA 值使用 C 函数可能会导致非预期地截断数据值。

  定义字符串例程参数时，请考虑使用 VARCHAR 取代 CHAR，因为 DB2 不会将 VARCHAR 自变量提升为 CHAR 且会自动将字符串文字视为 VARCHAR。例如，假设您按如下所示定义 UDF:

  ```
  CREATE FUNCTION SIMPLE(INT,CHAR(1))...
  ```

  如果使用 VARCHAR 数据来调用 SIMPLE 函数数据 (... SIMPLE(1,'A')...)，那么您将收到指示找不到函数的 SQLCODE -440 (SQLSTATE 42884) 错误，而且此函数的最终用户可能无法理解此消息的原因。在前面的示例中，'A' 是一个 VARCHAR，因此您可以将它强制类型转换成 CHAR 或将参数定义为 VARCHAR。

- VARCHAR(n) FOR BIT DATA 或带有或不带 FOR BIT DATA 修饰符的 LONG VARCHAR。

有效。在 C 中将 VARCHAR(n) FOR BIT DATA 表示为 SQLUDF_VARCHAR_FBD。在 C 中将 LONG VARCHAR 表示为 SQLUDF_LONG。否则，在 C 中将这两种 SQL 类型表示为类似 sqludf.h 包含文件中下列结构的结构:

```
struct sqludf_vc_fbd
{
   unsigned short length;         /* length of data */
   char           data[1];        /* first char of data */
};
```

[1] 指示传递至编译器的数组。这并不意味着只传递一个字符；因为传递的是结构地址，而不是实际结构，所以结构地址提供一种方法来使用数组逻辑。

这些值表示为以 null 结束的 C 字符串，因为 NULL 字符可以合理地成为数据值的一部分。对于参数，会使用结构变量 length 将长度显式地传递至例程。对于 RETURNS 子句，传递至例程的长度是缓冲区的长度。例程主体必须使用结构变量 length 来传回的长度是数据值的实际长度。

示例:

```
struct sqludf_vc_fbd *arg1;  /* example for VARCHAR(n) FOR BIT DATA */
struct sqludf_vc_fbd *result; /* also for LONG VARCHAR FOR BIT DATA */
```

- 不带 FOR BIT DATA 的 VARCHAR(n)。

  有效。在 C 中表示为 SQLUDF_VARCHAR 或 char...[*n*+1]。（这是以 null 结束的 C 字符串。）

  对于 VARCHAR(n) 参数，DB2 会在 (*k*+1) 位置中放入 NULL，其中 *k* 是特定字符串的长度。C 字符串处理函数非常适合于操作这些值。对于 RETURNS VARCHAR(n) 值或存储过程的输出参数，例程主体必须使用 NULL 来定界实际值，因为 DB2 将根据此 NULL 字符来确定结果长度。

  示例:

  ```
  char     arg2[51];      /* example for VARCHAR(50)   */
  char    *result;        /* also acceptable */
  ```

- DATE

  有效。在 C 中也表示为 SQLUDF_DATE 或 CHAR(10)（即 char...[11]）。日期值始终以 ISO 格式传递至例程:

  yyyy-mm-dd

  示例:

  ```
  char     arg1[11];      /* example for DATE       */
  char    *result;        /* also acceptable */
  ```

  注: 对于 DATE、TIME 以及 TIMESTAMP 返回值，DB2 会要求字符使用已定义的格式，如果情况并非如此，那么 DB2 可能会错误解释此值（例如，会将 2001-04-03 解释成 4 月 3 日，即使预期值为 3 月 4 日亦如此），或者会导致错误 (SQLCODE -493, SQLSTATE 22007)。

- TIME

  有效。在 C 中也表示为 SQLUDF_TIME 或 CHAR(8)（即 char...[9]）。时间值始终以 ISO 格式传递至例程:

hh.mm.ss

示例:

```
char    *arg;           /* example for TIME          */
char     result[9];     /* also acceptable */
```

- TIMESTAMP

有效。在 C 表示为 SQLUDF_STAMP 或 CHAR(19)-CHAR(32)（即 char[20]-char[33]）。时间戳记值具有下列格式:

yyyy-mm-dd-hh.mm.ss.nnnnnnnnnnnn

其中:

yyyy 表示年。

mm 表示月。

dd 表示日。

hh 表示小时。

mm 表示分钟。

ss 表示秒。

nnnnnnnnnnnn

表示小数秒。（可选）可以指定 TIMESTAMP 数据类型的小数秒部分，以使时间戳记精度为 0-12 位。

将时间戳记值指定给秒小数位数不同的时间戳记变量时，该值将被截断或者填充 0 以便与时间戳记变量的格式匹配。

字符串的长度可达 19-32 个字节，取决于所指定的小数秒数。（可选）可以指定 TIMESTAMP 数据类型的小数秒部分，以使时间戳记精度为 0-12 位。

例如:

```
(VALUES(CURRENT TIMESTAMP(0))
1
-------------------
2008-07-09-14.48.36

  1 record(s) selected.

LENGTH (VALUES(CURRENT TIMESTAMP(0))
1
-------------------
19

  1 record(s) selected.
(VALUES(CURRENT TIMESTAMP(12))
1
-------------------
2008-07-09-14.48.36.123456789012

  1 record(s) selected.

LENGTH (VALUES(CURRENT TIMESTAMP(0))
1
-------------------
32
```

下列是可以存放 TIMESTAMP(12) 值的变量声明:

```
char     arg1[33];      /* example for TIMESTAMP  */
char    *result;        /* also acceptable */
```

- GRAPHIC(n)

  有效。在 C 中表示为 SQLUDF_GRAPH 或 sqldbchar[n+1]。（这是以 null 结束的图形字符串）。请注意，您可以在 wchar_t 长度定义为 2 个字节的操作系统上使用 wchar_t[n+1]；但是，建议使用 sqldbchar。

  对于 GRAPHIC(*n*) 参数，DB2 数据库系统会将 *n* 双字节字符移至缓冲区，并将下列两个字节设为 NULL。数据以 DBCS 格式从 DB2 数据库系统传递至例程，且传回的结果应该采用 DBCS 格式。此行为等同于使用 WCHARTYPE NOCONVERT 预编译器选项。对于 RETURNS GRAPHIC(*n*) 值或存储过程的输出参数，DB2 数据库系统会查找嵌入式 GRAPHIC Null CHAR，并且如果找到 GRAPHIC Null CHAR，就会使用 GRAPHIC 空白字符将值填充到 *n* 位。

  在定义图形例程参数时，请考虑使用 VARGRAPHIC 取代 GRAPHIC，因为 DB2 数据库系统不会将 VARGRAPHIC 自变量提升为 GRAPHIC。例如，假设您按如下所示定义例程:

  ```
  CREATE FUNCTION SIMPLE(GRAPHIC)...
  ```

  如果使用 VARGRAPHIC 数据来调用 SIMPLE 函数数据（... SIMPLE ('*graphic_literal*')...），那么您将收到指示找不到函数的 SQLCODE -440 (SQLSTATE 42884) 错误，而且此函数的最终用户可能无法理解此消息的原因。在前面的示例中，*图形文字* 是一个解释为 VARGRAPHIC 数据的文字 DBCS 字符串，因此您可以将它强制类型转换成 GRAPHIC 或将参数定义为 VARGRAPHIC。

  示例:

  ```
  sqldbchar   arg1[14];      /* example for GRAPHIC(13)   */
  sqldbchar  *arg1;          /* also acceptable */
  ```

- VARGRAPHIC(n)

  有效。在 C 中表示为 SQLUDF_GRAPH 或 sqldbchar[n+1]。（这是以 null 结束的图形字符串）。请注意，您可以在 wchar_t 长度定义为 2 个字节的操作系统上使用 wchar_t[n+1]；但是，建议使用 sqldbchar。

  对于 VARGRAPHIC(*n*) 参数，DB2 数据库系统会在 (k+1) 位置中放入 GRAPHIC NULL，其中 *k* 是特定出现项的长度。GRAPHIC NULL 是指图形字符串最后一个字符的所有字节都包含二进制零（'\0'）的情况。数据以 DBCS 格式从 DB2 数据库系统传递至例程，且传回的结果应该采用 DBCS 格式。此行为等同于使用 WCHARTYPE NOCONVERT 预编译器选项。对于 RETURNS VARGRAPHIC(*n*) 值或存储过程的输出参数，例程主体必须使用 GRAPHIC NULL 来定界实际值，因为 DB2 数据库系统将根据此 GRAPHIC NULL 字符来确定结果长度。

  示例:

  ```
  sqldbchar   args[51],    /* example for VARGRAPHIC(50) */
  sqldbchar   *result,     /* also acceptable  */
  ```

- LONG VARGRAPHIC

  有效。在 C 中表示为 SQLUDF_LONGVARG 或结构:

```
struct sqludf_vg
{
   unsigned short length;         /* length of data */
   sqldbchar      data[1];        /* first char of data */
};
```

请注意，在前面的结构中，您可以在 wchar_t 长度定义为 2 个字节的操作系统上，使用 wchar_t 来取代 sqldbchar；但是，建议使用 sqldbchar。

[1] 仅仅指示传递至编译器的数组。它并不意味着仅传递一个图形字符。因为传递的是结构地址，而不是实际结构，所以结构地址提供一种方法来使用数组逻辑。

这些并不表示为以 null 结束的图形字符串。对于参数，会使用结构变量 length 将长度（以双字节字符表示）显式地传递至例程。数据以 DBCS 格式从 DB2 数据库系统传递至例程，且传回的结果应该采用 DBCS 格式。此行为等同于使用 WCHARTYPE NOCONVERT 预编译器选项。对于 RETURNS 子句或存储过程的输出参数，传递至例程的长度是缓冲区的长度。例程主体必须使用结构变量 length 来传回的长度是数据值的实际长度（以双字节字符表示）。

示例:

```
struct sqludf_vg *arg1;  /* example for VARGRAPHIC(n)   */
struct sqludf_vg *result; /* also for LONG VARGRAPHIC   */
```

- BLOB(n) 和 CLOB(n)

  有效。在 C 中表示为 SQLUDF_BLOB、SQLUDF_CLOB 或结构:

  ```
  struct sqludf_lob
  {
      sqluint32     length;      /* length in bytes */
      char          data[1];      /* first byte of lob */
  };
  ```

  [1] 仅仅指示传递至编译器的数组。这并不意味着只传递一个字符；因为传递的是结构地址，而不是实际结构，所以结构地址提供一种方法来使用数组逻辑。

  这些并不表示为以 null 结束的 C 字符串。对于参数，会使用结构变量 length 将长度显式地传递至例程。对于 RETURNS 子句或存储过程的输出参数，传递回至例程的长度是缓冲区的长度。例程主体必须使用结构变量 length 来传回的长度是数据值的实际长度。

  示例:

  ```
  struct sqludf_lob *arg1;  /* example for BLOB(n), CLOB(n) */
  struct sqludf_lob *result;
  ```

- DBCLOB(n)

  有效。在 C 中表示为 SQLUDF_DBCLOB 或结构:

  ```
  struct sqludf_lob
  {
      sqluint32 length;      /* length in graphic characters */
      sqldbchar data[1];         /* first byte of lob */
  };
  ```

  请注意，在前面的结构中，您可以在 wchar_t 长度定义为 2 个字节的操作系统上，使用 wchar_t 来取代 sqldbchar；但是，建议使用 sqldbchar。

[1] 仅仅指示传递至编译器的数组。这并不意味着只传递一个图形字符；因为传递的是结构地址，而不是实际结构，所以结构地址提供一种方法来使用数组逻辑。

这些并不表示为以 null 结束的图形字符串。对于参数，会使用结构变量 length 将长度显式地传递至例程。数据以 DBCS 格式从 DB2 数据库系统传递至例程，且传回的结果应该采用 DBCS 格式。此行为等同于使用 WCHARTYPE NOCONVERT 预编译器选项。对于 RETURNS 子句或存储过程的输出参数，传递至例程的长度是缓冲区的长度。例程主体必须使用结构变量 length 来传回的长度是数据值的实际长度（其中的所有长度都以双字节字符表示）。

示例:

```
struct sqludf_lob *arg1;  /* example for DBCLOB(n) */
struct sqludf_lob *result;
```

- 单值类型

  有效或无效，取决于基本类型。会以 UDT 的基本类型格式将单值类型传递至 UDF，因此只有此基本类型有效时，才能指定单值类型。

  示例:

```
struct sqludf_lob *arg1;  /* for distinct type based on BLOB(n) */
double            *arg2;  /* for distinct type based on DOUBLE  */
char              res[5]; /* for distinct type based on CHAR(4) */
```

- XML

  有效。在 C 中表示为 SQLUDF_XML，或者以表示 CLOB 数据类型的方法来表示；下列是一个结构示例:

```
struct sqludf_lob
{
    sqluint32     length;      /* length in bytes */
    char          data[1];      /* first byte of lob */
};
```

  [1] 仅仅指示传递至编译器的数组。这并不意味着只传递一个字符；因为传递的是结构地址，而不是实际结构，所以结构地址提供一种方法来使用数组逻辑。

  这些并不表示为以 null 结束的 C 字符串。对于参数，会使用结构变量 length 将长度显式地传递至例程。对于 RETURNS 子句或存储过程的输出参数，传递回至例程的长度是缓冲区的长度。例程主体必须使用结构变量 length 来传回的长度是数据值的实际长度。

  示例:

```
struct sqludf_lob *arg1;  /* example for XML(n) */
struct sqludf_lob *result;
```

  在 C 和 C++ 外部例程代码中对 XML 参数与变量值进行赋值和访问的方法，与 CLOB 值相同。

- 单值类型 AS LOCATOR 或任何 LOB 类型 AS LOCATOR

  对于 UDF 和方法的参数与结果有效。它只能用于修改 LOB 类型或任何基于 LOB 类型的单值类型。在 C 中表示为 SQLUDF_LOCATOR 或四字节整数。

定位器值可以指定给任何具有兼容类型的定位器主变量，然后用于 SQL 语句。这意味着定位器变量只适用于定义了 CONTAINS SQL 或更高级别的 SQL 访问指示符的 UDF 和方法。为了确保与现有 UDF 和方法兼容，定位器 API 仍支持 NOT FENCED NO SQL UDF。不推荐在新函数中使用这些 API。

示例:

```
     sqludf_locator       *arg1;  /* locator argument */
     sqludf_locator       *result; /* locator result */

 EXEC SQL BEGIN DECLARE SECTION;
   SQL TYPE IS CLOB LOCATOR arg_loc;
   SQL TYPE IS CLOB LOCATOR res_loc;
EXEC SQL END DECLARE SECTION;

/* Extract some characters from the middle */
/* of the argument and return them         */
*arg_loc = arg1;
EXEC SQL VALUES SUBSTR(arg_loc, 10, 20) INTO :res_loc;
*result = res_loc;
```

- 结构化类型

  对于 UDF 和方法（具有相应的变换函数）的参数与结果有效。会在 FROM SQL 变换函数的结果类型中将结构化类型参数传递至函数或方法。会在 TO SQL 变换函数的参数类型中传递结构化类型结果。

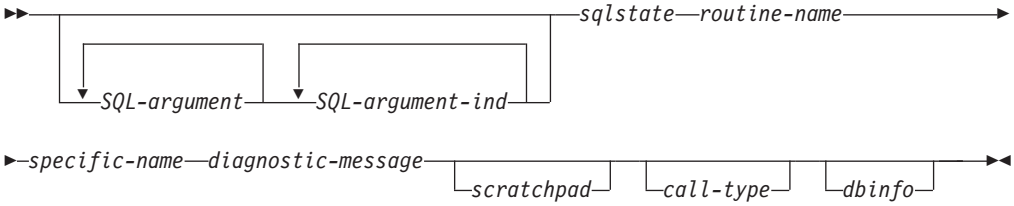
将自变量传递至 C、C++、OLE 或 COBOL 例程

除了在 DML 引用中针对例程指定的 SQL 自变量之外，DB2 数据库系统还会将其他自变量传递至外部例程主体。由注册例程时使用的参数样式确定这些自变量的性质和顺序。

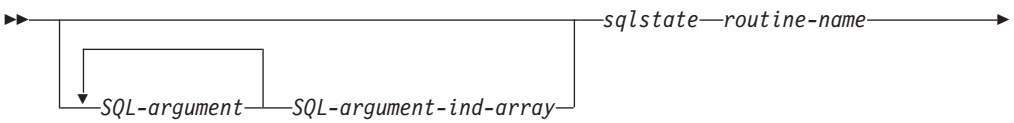
要确保在调用程序和例程主体之间正确地交换信息，您必须确保例程会根据所使用的参数样式，以自变量的传递顺序来接受自变量。sqludf 包含文件可以帮助您处理和使用这些自变量。

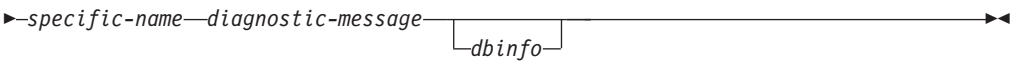
下列参数样式只适用于 C 语言、OLE 语言以及 COBOL 语言例程。

PARAMETER STYLE SQL 例程



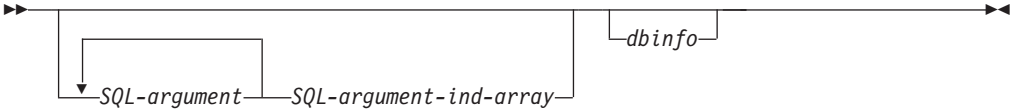
PARAMETER STYLE DB2SQL 过程





PARAMETER STYLE GENERAL 过程

PARAMETER STYLE GENERAL WITH NULLS 过程



注: 对于 UDF 和方法，PARAMETER STYLE SQL 等价于 PARAMETER STYLE DB2SQL。

上述参数样式的自变量如下所述:

SQL-argument...

每个 *SQL-argument* 都表示建立例程时所定义的一个输入或输出值。按下列规则确定自变量列表:

- 对于标量函数，函数的每个输入参数的一个自变量，后跟函数结果的一个 *SQL-argument*。
- 对于表函数，函数的每个输入参数的一个自变量，后跟有函数结果表中每个列的一个 *SQL-argument*。
- 对于方法，方法主题类型的一个 *SQL-argument*，然后是方法的每个输入参数的一个自变量，后跟方法结果的一个 *SQL-argument*。
- 对于存储过程，存储过程的每个参数的一个 *SQL-argument*。

按下列规则使用每个 *SQL-argument*:

- 函数或方法的输入参数、方法的主题类型或存储过程的 IN 参数

  在调用例程之前，由 DB2 数据库系统设置此自变量。从例程调用中所指定的表达式获取其中每个自变量的值。以 CREATE 语句中对应参数定义的数据类型表示该值。

- 函数或方法的结果，或存储过程的 OUT 参数

  在返回到 DB2 数据库系统之前，由例程设置此自变量。DB2 数据库系统分配缓冲区，并将其地址传递至例程。例程会将结果值放入缓冲区。DB2 数据库系统会分配足够的缓冲区空间来包含数据类型所表示的值。对于字符类型和 LOB，这意味着分配 CREATE 语句中所定义的最大大小。

  对于标量函数和方法，结果数据类型是在 CAST FROM 子句（如果存在）中定义，或在 RETURNS 子句中定义（如果不存在任何 CAST FROM 子句）。

对于表函数，DB2 数据库系统会定义性能优化，这样就不必将所定义的每个列都返回给 DB2。如果您编写 UDF 以利用此功能，那么此 UDF 只返回语句（对表函数进行引用）所需的列。例如，设想对定义了 100 个结果列的表函数使用 CREATE FUNCTION 语句。如果对此函数进行引用的给定语句只对其中两个感兴趣，那么此优化使 UDF 能够只返回每一行中所感兴趣的两个列，而不会在其他 98 列上花费任何时间。请参阅下面的 dbinfo 自变量，以了解有关此优化的更多信息。

对于返回的每个值，例程所返回的字节数不应该超过结果的数据类型和长度所需的字节数。在创建例程的目录条目期间定义最大值。如果例程执行覆盖操作，那么会导致不可预测的结果或异常终止。

- 存储过程的 INOUT 参数

  此自变量兼有 IN 和 OUT 参数的行为，因此遵循如上所示的两组规则。在调用存储过程之前，DB2 数据库系统会设置此自变量。由 DB2 数据库系统为此自变量分配的缓冲区大小，足以匹配 CREATE PROCEDURE 语句中所定义参数的数据类型的最大大小。例如，CHAR 类型的 INOUT 参数可能有一个 10 字节 varchar 传入存储过程，而且有一个 100 字节 varchar 从存储过程中传出。在返回到 DB2 数据库系统之前，由存储过程设置此缓冲区。

DB2 数据库系统会根据数据类型和服务器操作系统（又称为平台）来对齐 *SQL-argument* 的数据。

SQL-argument-ind...

每个传递至例程的 *SQL-argument* 都具有一个 *SQL-argument-ind*。第 *n* 个 *SQL-argument-ind* 对应于第 *n* 个 *SQL-argument*，且指示 *SQL-argument* 的值是否为 NULL。

按下列规则使用每个 *SQL-argument-ind*:

- 函数或方法的输入参数、方法的主题类型或存储过程的 IN 参数

  在调用例程之前，由 DB2 数据库系统设置此自变量。它包含下列其中一个值:

  0 自变量存在，且值不为 NULL。

  -1 自变量存在，且值为 NULL。

  如果例程中定义了 RETURNS NULL ON NULL INPUT，那么例程主体不需要检查值是否为 NULL。但是，如果例程中定义了 CALLED ON NULL INPUT，那么任何自变量的值都可以为 NULL，并且例程应该先检查 *SQL-argument-ind*，然后再使用对应的 *SQL-argument*。

- 函数或方法的结果，或存储过程的 OUT 参数

  在返回到 DB2 数据库系统之前，由例程设置此自变量。例程会使用此自变量来指示特定结果值是否为 NULL:

  0 结果值不为 NULL。

  -1 结果值为 NULL。

  即使例程中定义了 RETURNS NULL ON NULL INPUT，例程主体也必须设置结果的 *SQL-argument-ind*。例如，在分母为零时，除函数可以将结果设为 NULL。

对于标量函数和方法，如果下列情况成立，那么 DB2 数据库系统会将 NULL 结果视为算术错误:

– 数据库配置参数 **dft_sqlmathwarn** 的值为 YES

– 其中一个输入自变量由于算术错误而变为 NULL

这对于函数中定义了 RETURNS NULL ON NULL INPUT 选项的情况也成立

对于表函数，如果 UDF 利用优化（使用结果列列表），那么只需要设置对应于所需列的指示符。

- 存储过程的 INOUT 参数

  此自变量兼有 IN 和 OUT 参数的行为，因此遵循如上所示的两组规则。在调用存储过程之前，DB2 数据库系统会设置此自变量。在返回到 DB2 数据库系统之前，由存储过程设置 *SQL-argument-ind*。

每个 *SQL-argument-ind* 都接受 SMALLINT 值格式。DB2 数据库系统会根据数据类型和服务器操作系统来对齐 *SQL-argument-ind* 的数据。

SQL-argument-ind-array

传递至存储过程的每个 SQL-argument 的 *SQL-argument-ind-array* 中都有一个元素。*SQL-argument-ind-array* 中的第 *n* 个元素对应于第 *n* 个 SQL-argument，且指示 *SQL-argument* 的值是否为 NULL

按下列规则使用 *SQL-argument-ind-array* 中的每个元素:

- 存储过程的 IN 参数

  在调用例程之前，由 DB2 数据库系统设置此元素。它包含下列其中一个值:

  0 自变量存在，且值不为 NULL。

  -1 自变量存在，且值为 NULL。

  如果存储过程中定义了 RETURNS NULL ON NULL INPUT，那么存储过程体不需要检查值是否为 NULL。但是，如果存储过程中定义了 CALLED ON NULL INPUT，那么任何自变量的值都可以为 NULL，并且存储过程应该先检查 *SQL-argument-ind*，然后再使用对应的 *SQL-argument*。

- 存储过程的 OUT 参数

  在返回到 DB2 数据库系统之前，由例程设置此元素。例程会使用此自变量来指示特定结果值是否为 NULL:

  0 或正数

  结果值不为 NULL。

  负数 结果值为 NULL。

- 存储过程的 INOUT 参数

  此元素兼有 IN 和 OUT 参数的行为，因此遵循如上所示的两组规则。在调用存储过程之前，DB2 数据库系统会设置此自变量。在返回到 DB2 数据库系统之前，由存储过程设置 *SQL-argument-ind-array* 的元素。

SQL-argument-ind-array 的每个元素都接受 SMALLINT 值格式。DB2 数据库系统会根据数据类型和服务器操作系统来对齐 *SQL-argument-ind-array* 的数据。

sqlstate

在返回到 DB2 数据库系统之前，由例程设置此自变量。例程可以使用此自变量来指示警告或错误情况。例程可以将此自变量设为任何值。值“00000”意味着未检测到任何警告或错误情况。前缀为“01”的值意味着检测到警告情况。如果值的前缀不是“00”或“01”，那么意味着检测到错误情况。在调用例程时，自变量会包含值“00000”。

对于错误情况，例程会返回 SQLCODE -443。对于警告情况，例程会返回 SQLCODE +462。如果 SQLSTATE 是 38001 或 38502，那么 SQLCODE 是 -487。

sqlstate 接受 CHAR(5) 值格式。DB2 数据库系统会根据数据类型和服务器操作系统来对齐 *sqlstate* 的数据。

routine-name

在调用例程之前，由 DB2 数据库系统设置此自变量。它是从 DB2 数据库系统传递至例程的限定函数名

所传递的 *例程名* 格式为:

模式.*例程*

使用句点隔开各个部分。如下是两个示例:

```
PABLO.BLOOP   WILLIE.FINDSTRING
```

此格式使您能够将同一例程主体用于多个外部例程，且在调用该例程主体时仍能够区分这些例程。

注: 虽然可以在对象名和模式名中包含句点，但不建议这样做。例如，如果函数 ROTATE 是在模式 OBJ.OP 中，那么传递至该函数的例程名称是 OBJ.OP.ROTATE，但如果模式名是 OBJ 或 OBJ.OP，那么例程名称不太明显。

routine-name 接受 VARCHAR(257) 值格式。DB2 数据库系统会根据数据类型和服务器操作系统来对齐 *routine-name* 的数据。

specific-name

在调用例程之前，由 DB2 数据库系统设置此自变量。它是从 DB2 数据库系统传递至例程的例程特定名称。

如下是两个示例:

```
WILLIE_FIND_FEB99   SQL9904281052440430
```

第一个值是由用户在他的 CREATE 语句中提供。第二个值是在用户未指定值时，由 DB2 数据库系统根据当前时间戳记生成。

和 *routine-name* 自变量一样，传递此值的原因是向例程提供一种方法来确切识别对其进行调用的特定例程。

specific-name 接受 VARCHAR(18) 值格式。DB2 数据库系统会根据数据类型和服务器操作系统来对齐 *specific-name* 的数据。

diagnostic-message

在返回到 DB2 数据库系统之前，由例程设置此自变量。例程可以使用此自变量将消息文本插入到 DB2 数据库消息中。

在例程使用如上所述的 *sqlstate* 自变量来返回错误或警告时，它可以在此处包含描述信息。DB2 数据库系统将此信息作为标记包含在其消息中。

在调用例程之前，DB2 数据库系统会将第一个字符设为 NULL。在返回时，它会将此字符串视为以 null 结束的 C 字符串。对于错误情况，此字符串会作为标记包含在 SQLCA 中。SQLCA 或 DB2 CLP 消息中至少会出现此字符串的第一个部分。但是，所出现的实际字符数取决于其他标记的长度，因为 DB2 数据库系统会截断标记以符合 SQLCA 所实施的标记总长度限制。避免在文本中使用 X'FF'，因为在 SQLCA 中使用此字符来定界标记。

例程所返回的文本不应该超出传递至该例程的 VARCHAR(70) 缓冲区。如果例程执行覆盖操作，那么会导致不可预测的结果或异常结束。

DB2 数据库系统假定从例程返回给 DB2 数据库系统的任何消息标记都使用例程所使用的代码页。您的例程应该确保符合此情况。如果您使用 7 位不变量 ASCII 子集，那么例程可以使用任何代码页来返回消息标记。

diagnostic-message 接受 VARCHAR(70) 值格式。DB2 数据库系统会根据数据类型和服务器操作系统来对齐 *diagnostic-message* 的数据。

scratchpad

在调用 UDF 或方法之前，由 DB2 数据库系统设置此自变量。仅对于注册期间指定了 SCRATCHPAD 关键字的函数和方法，此自变量是存在的。此自变量是一种结构（和用来传递任何 LOB 数据类型值的结构完全一样），具有下列元素:

- INTEGER，包含暂存区的长度。更改暂存区的长度会导致 SQLCODE -450 (SQLSTATE 39501)
- 实际的暂存区，完全初始化为二进制 0，如下所示:
  - 对于标量函数和方法，会在第一个调用之前初始化暂存区，此后，DB2 数据库系统通常不会查看或修改该暂存区。
  - 对于表函数，如果对 CREATE FUNCTION 指定 FINAL CALL，那么会在对 UDF 执行 FIRST 调用之前初始化该暂存区。在此调用之后，暂存区内容完全在表函数的控制之下。如果对表函数指定 NO FINAL CALL 或将 NO FINAL CALL 设为表函数的缺省值，那么会针对每个 OPEN 调用初始化该暂存区，且在 OPEN 调用之间，暂存区内容完全在表函数的控制之下。（这对于在连接或子查询中使用的表函数会非常重要。如果在每次执行 OPEN 调用时必须保留暂存区的内容，那么必须在 CREATE FUNCTION 语句中指定 FINAL CALL。如果指定了 FINAL CALL，那么除一般的 OPEN、FETCH 以及 CLOSE 调用之外，表函数也将会接收 FIRST 和 FINAL 调用，供暂存区维护和资源释放之用。）

可以在例程中使用与 CLOB 或 BLOB 相同的类型来映射暂存区，因为所传递的自变量具有相同的结构。

确保例程代码不会在暂存区缓冲区外部作出更改。如果例程执行覆盖操作，那么会导致不可预测的结果或异常结束，且可能导致 DB2 数据库系统发生非正常故障。

如果在子查询中引用了使用暂存区的标量 UDF 或方法，那么 DB2 数据库系统可能会决定在两次调用此子查询之间刷新暂存区。如果对 UDF 指定了 FINAL CALL，那么会在作出最终调用后执行此刷新操作。

DB2 数据库系统会初始化暂存区，以针对任何数据类型的存储器对齐数据字段。这可能会导致整个 scratchpad 结构（其中包括长度字段）错误地对齐。

call-type

在调用 UDF 或方法之前，由 DB2 数据库系统设置此自变量（如果存在）。对于所有表函数以及注册期间指定了 FINAL CALL 的标量函数和方法，此自变量是存在的

后跟 *call-type* 的所有当前可能值。UDF 或方法应该包含显式地检测所有预期值的 Switch 或 Case 语句，而不是包含“if A do AA, else if B do BB, else it must be C so do CC”类型逻辑。这是因为以后可能会添加其他调用类型，并且如果您未显式地检测条件 C，那么以后可能很难添加新的可能情况。

注:

1. 对于 *call-type* 的所有值，让例程设置 *sqlstate* 和 *diagnostic-message* 返回值可能很合适。将不会在每个 *call-type* 的下列描述中重复此信息。对于所有调用，DB2 数据库系统都会执行前面针对这些自变量所作的描述中指示的操作。
2. 包含文件 `sqludf.h` 预期与例程一起使用。该文件包含下列 *call-type* 值（可作为常量完全拼出）的符号定义。

对于标量函数和方法，*call-type* 包含:

SQLUDF_FIRST_CALL (-1)

这是针对此语句的例程所作的 FIRST 调用。调用例程时，会将 *scratchpad*（如果有）设为二进制零。会传递所有自变量值，并且例程应该完成一次性初始化操作所要完成的任何工作。此外，对标量 UDF 或方法所作的 FIRST 调用类似于 NORMAL 调用，应该产生并返回应答。

注: 如果指定 SCRATCHPAD，而不指定 FINAL CALL，那么例程将不会具有这个用来标识第一个调用的 *call-type* 自变量。相反地，它将必须依赖于暂存区的全零状态。

SQLUDF_NORMAL_CALL (0)

这是 NORMAL 调用。会传递 SQL 输入值，且例程应该产生并返回结果。例程也可以返回 *sqlstate* 和 *diagnostic-message* 信息。

SQLUDF_FINAL_CALL (1)

这是 FINAL 调用，即，不会传递任何 *SQL-argument* 或 *SQL-argument-ind* 值，并且尝试检查这些值会导致不可预测的结果。如果也传递 *scratchpad*，那么它会保持先前调用时的原状。此刻，例程应该释放资源。

SQLUDF_FINAL_CRA (255)

这是 FINAL 调用，与上述 FINAL 调用相同，不过此调用具有一项其他特征，即，对定义为能够发出 SQL 的例程执行此调用，并且在执行此调用时，该例程不能发出除 CLOSE 游标之外的任何 SQL。(SQLCODE -396, SQLSTATE 38505) 例如，如果 DB2 数据库系统正在执行 COMMIT 处理，那么不容许新的 SQL，并且此时对例程发出的任何 FINAL 调用都会是 255 FINAL 调用。未定义为包含任何 SQL 访问级别的例程都永不会接收 255 FINAL 调用，但是，可能会对确实使用 SQL 的例程指定任一类型的 FINAL 调用。

释放资源: 标量 UDF 或方法应该释放它所获取的资源，例如内存。如果对例程指定了 FINAL CALL，那么只要也指定了 SCRATCHPAD 且使用它来跟踪资

源，该 FINAL 调用是释放资源的自然位置。如果未指定 FINAL CALL，那么应该执行相同调用来释放所获取的任何资源。

对于表函数，*call-type* 包含:

SQLUDF_TF_FIRST (-2)

这是 FIRST 调用，仅当对 UDF 指定了 FINAL CALL 关键字时，才会执行此调用。在执行此调用之前，会将 *scratchpad* 设为二进制零。会将自变量值传递至表函数。表函数可以获取内存，或执行其他仅一次性资源初始化操作。这不是 OPEN 调用。此调用后面是 OPEN 调用。在执行 FIRST 调用时，表函数不应该将任何数据返回给 DB2 数据库系统，因为 DB2 数据库系统会忽略此数据。

SQLUDF_TF_OPEN (-1)

这是 OPEN 调用。如果指定 NO FINAL CALL，那么将初始化 *scratchpad*，但在其他情况下，不一定会初始化。在执行 OPEN 时，会将所有 SQL 自变量值传递至表函数。在执行 OPEN 调用时，表函数不应该将任何数据返回给 DB2 数据库系统。

SQLUDF_TF_FETCH (0)

这是 FETCH 调用，并且 DB2 数据库系统期望表函数返回一行（包含一组返回值）或表结束条件（由 SQLSTATE 值“02000”所指示）。如果将 *scratchpad* 传递至 UDF，那么在进入时，它会保持先前调用时的原状。

SQLUDF_TF_CLOSE (1)

这是对表函数所作的 CLOSE 调用。它会平衡 OPEN 调用，且可用来执行任何外部 CLOSE 处理（例如，关闭源文件）以及释放资源（特别是 NO FINAL CALL 情况）。

如果涉及连接或子查询，那么 OPEN/FETCH.../CLOSE 调用序列可以在语句执行中重复，但只有一个 FIRST 调用和一个 FINAL 调用。仅当对表函数指定 FINAL CALL 时，才会执行 FIRST 和 FINAL 调用。

SQLUDF_TF_FINAL (2)

这是 FINAL 调用，只有在对表函数指定了 FINAL CALL 时才会执行。它会平衡 FIRST 调用，并且只有在每次执行语句时才执行一次。它预期用于释放资源。

SQLUDF_TF_FINAL_CRA (255)

这是 FINAL 调用，与上述 FINAL 调用相同，不过此调用具有一项其他特征，即，对定义为能够发出 SQL 的 UDF 执行此调用，并且在执行此调用时，该 UDF 不能发出除 CLOSE 游标之外的任何 SQL 语句。 (SQLCODE -396, SQLSTATE 38505) 例如，如果 DB2 数据库系统正在执行 COMMIT 处理，那么不容许新的 SQL，并且此时对 UDF 发出的任何 FINAL 调用都会是 255 FINAL 调用。请注意，未定义为包含任何 SQL 访问级别的 UDF 都永不会接收 255 FINAL 调用，但是，可以对确实使用 SQL 的 UDF 指定任一类型的 FINAL 调用。

释放资源: 编写例程以释放它们所获取的任何资源。对于表函数，有两个自然位置可执行此释放操作: CLOSE 调用和 FINAL 调用。CLOSE 调用会平衡每个 OPEN 调用，并且可以在执行语句时多次执行。FINAL 调用只有在对 UDF 指定 FINAL CALL 时才会执行，并且只有在每次执行语句时才执行一次。

如果您可以在执行 UDF 的所有 OPEN/FETCH/CLOSE 序列期间应用资源，请编写 UDF 以在执行 FIRST 调用时获取资源并在执行 FINAL 调用时释放资源。暂存区是跟踪此资源的自然位置。对于表函数，如果指定 FINAL CALL，那么只有在执行 FIRST 调用之前才初始化暂存区。如果未指定 FINAL CALL，那么会在执行每个 OPEN 调用之前重新初始化暂存区。

如果资源特定于每个 OPEN/FETCH/CLOSE 序列，请编写 UDF 以执行 CLOSE 调用来释放资源。

注: 如果表函数在子查询或连接中，那么 OPEN/FETCH/CLOSE 序列很可能会出现多次，取决于 DB2 优化器选择如何组织语句的执行。

call-type 接受 INTEGER 值格式。DB2 数据库系统会根据数据类型和服务器操作系统来对齐 *call-type* 的数据。

dbinfo

在调用例程之前，由 DB2 数据库系统设置此自变量。此自变量只有在例程的 CREATE 语句指定 DBINFO 关键字时才存在。此自变量是头文件 `sqludf.h` 中所定义的 `sqludf_dbinfo` 结构。此结构中包含名称和标识的变量在长度上可能超过此 DB2 数据库发行版中可能的最长值，但以此方式定义变量是为了与将来发行版保持兼容。您可以使用对每个名称和标识变量进行补充的长度变量来读取或抽取实际使用的变量部分。dbinfo 结构包含下列元素:

1. 数据库名长度（dbnamelen）

   数据库名 的长度。此字段是 unsigned short integer。

2. 数据库名（dbname）

   当前已连接数据库的名称。此字段是一个由 128 个字符组成的长标识。如上所述的*数据库名长度* 字段会标识此字段的实际长度。它不包含 null 终止符或任何填充字符。

3. 应用程序授权标识长度（authidlen）

   应用程序授权标识 的长度。此字段是 unsigned short integer。

4. 应用程序授权标识（authid）

   应用程序运行时授权标识。此字段是一个由 128 个字符组成的长标识。它不包含 null 终止符或任何填充字符。上述*应用程序授权标识长度* 字段会标识此字段的实际长度。

5. 环境代码页（codepg）

   这是三个 48 字节结构的并集；一个结构是所有 DB2 数据库产品的公共结构（cdpg_db2），另一个是针对旧版 DB2 资料库编写的例程所使用的结构（cdpg_cs），最后一个是供旧版 DB2 Universal Database™ for z/OS and OS/390®使用的结构（cdpg_mvs）。为了确保可移植性，建议在所有例程中使用公共结构 cdpg_db2。

   cdpg_db2 结构由一个数组（db2_ccsids_triplet）所构成，此数组包含三组表示数据库中如下所示的可能编码方案的代码页信息:

   a. ASCII 编码方案。请注意，为了确保与较低版本的 DB2数据库兼容，如果数据库是 Unicode 数据库，那么 Unicode 编码方案的信息会放在此处并且会出现在第三个元素中。

b. EBCDIC 编码方案

c. Unicode 编码方案

编码方案信息后面是例程编码方案（db2_encoding_scheme）的数组下标。数组的每个元素都包含三个字段:

- db2_sbcs。单字节代码页，unsigned long integer。
- db2_dbcs。双字节代码页，无符号长整数。
- db2_mixed。组合代码页（又称为混合代码页），unsigned long integer。

6. 模式名长度 (tbschemalen)

   模式名 的长度。如果未传递表名，那么包含 0（零）。此字段是 unsigned short integer。

7. 模式名（tbschema）

   表名 的模式。此字段是一个由 128 个字符组成的长标识。它不包含 null 终止符或任何填充字符。如上所述的*模式名长度* 字段会标识此字段的实际长度。

8. 表名长度（tbnamelen）

   表名 的长度。如果未传递表名，那么包含 0（零）。此字段是 unsigned short integer。

9. 表名 (tbname)

   这是所更新或插入的表的名称。仅当例程引用位于 UPDATE 语句中 SET 子句的右侧，或者是 INSERT 语句的 VALUES 列表中的项时，才会设置此字段。此字段是一个由 128 个字符组成的长标识。它不包含 null 终止符或任何填充字符。如上所述的*表名长度* 字段会标识此字段的实际长度。如上所述的*模式名* 字段与此字段一起构成标准表名。

10. 列名长度（colnamelen）

    列名 的长度。如果未传递列名，那么包含 0（零）。此字段是 unsigned short integer。

11. 列名（colname）

    与表名的条件完全相同，此字段包含所更新或插入的列的名称；否则，无法预测此字段。此字段是一个由 128 个字符组成的长标识。它不包含 null 终止符或任何填充字符。如上所述的*列名长度* 字段会标识此字段的实际长度。

12. 版本号/发行版号（ver_rel）一个 8 字符字段，会标识产品及其版本、发行版和修改级别，格式为 *pppvvrrm*，其中:

    - *ppp* 会标识产品，如下所示:

      DSN DB2 z/OS 版 or OS/390

      ARI SQL/DS 或 DB2 VM 版或 VSE 版

      QSQ DB2 System i 版数据库

      SQL DB2 Database for Linux, UNIX, and Windows

- *vv* 是一个两位数版本标识。
- *rr* 是一个两位数发行版标识。
- *m* 是一个一位数修改级别标识。

13. 保留字段（resd0）

    此字段供将来使用。

14. 平台（platform）

    应用程序服务器的操作系统（平台），如下所示:

    SQLUDF_PLATFORM_AIX
    AIX®

    SQLUDF_PLATFORM_HP
    HP-UX

    SQLUDF_PLATFORM_LINUX
    Linux

    SQLUDF_PLATFORM_MVS
    OS/390

    SQLUDF_PLATFORM_NT
    Windows 2000 或 Windows XP

    SQLUDF_PLATFORM_SUN
    Solaris 操作系统

    SQLUDF_PLATFORM_WINDOWS95
    Windows 95、Windows 98 或 Windows Me

    SQLUDF_PLATFORM_UNKNOWN
    未知的操作系统或平台

    对于以前列表未包含的其他操作系统，请参阅 `sqludf.h` 文件的内容。

15. 表函数列列表条目数（numtfcol）

    表函数列列表 字段所指定的表函数列列表中的非零条目数。

16. 保留字段（resd1）

    此字段供将来使用。

17. 对当前例程进行调用的存储过程的例程标识（procid）

    存储过程的例程标识匹配 SYSCAT.ROUTINES 中的 ROUTINEID 列，此标识可用来检索调用存储过程的名称。此字段是 32 位带符号整数。

18. 保留字段（resd2）

    此字段供将来使用。

19. 表函数列列表（tfcolumn）

    如果这是表函数，那么此字段是 DB2 数据库系统动态分配的短整数数组的指针。如果这是任何其他类型的例程，那么此指针是 NULL。

此字段仅供表函数使用。只有前 *n* 个条目是相关的，其中 *n* 是在*表函数列列表条目数* 字段（numtfcol）中指定。*n* 可以等于 0，并且 *n* 小于或等于在 CREATE FUNCTION 语句的 RETURNS TABLE(...) 子句中针对函数定义的结果列数。这些值对应于此语句需要从表函数中获取的列序数。值“1”表示已定义的第一个结果列，值“2”表示已定义的第二个结果列，依次类推；并且这些值可以采用任意顺序。请注意，*n* 可以等于零，即，对于类似于 `SELECT COUNT(*) FROM TABLE(TF(...)) AS QQ` 的语句（其中，查询不需要任何实际列值），变量 `numtfcol` 可以是零。

此数组表示优化机会。UDF 并不需要返回表函数中所有结果列的所有值，只需返回特定上下文所需的值（这些值所属的列是数组中按编号进行标识的列）。由于在提升性能益处时，此优化会使 UDF 逻辑复杂化，因此 UDF 可以选择返回每个已定义的列。

20. 唯一应用程序标识（appl_id）

    此字段是以 null 结束的 C 字符串（可唯一标识应用程序与 DB2 数据库系统的连接）的指针。在连接时，由 DB2 数据库系统生成此字符串。

    此字符串的最大长度是 32 个字符，且字符串的精确格式取决于在客户机和 DB2 数据库系统之间建立的连接的类型。通常，它接受下列格式:

    ```
    x.y.ts
    ```

    其中，*x* 和 *y* 随连接类型而改变，而 *ts* 是一个 12 字符时间戳记（格式为 YYMMDDHHMMSS），DB2 数据库系统可能会调整此时间戳记以确保唯一性。

    示例: `*LOCAL.db2inst.980707130144`

21. 保留字段（resd3）

    此字段供将来使用。

C 和 C++ 例程中的图形主变量

使用 C 或 C++ 编写且通过其参数输入或输出来接收或返回图形数据的任何例程，通常都应该使用 WCHARTYPE NOCONVERT 选项进行预编译。这是因为通过这些参数传递的图形数据被视为使用 DBCS 格式，而不是 `wchar_t` 进程代码格式。使用 NOCONVERT 意味着在例程的 SQL 语句中操作的图形数据也将采用 DBCS 格式，以匹配参数数据的格式。

借助 WCHARTYPE NOCONVERT，不需要在图形主变量和数据库管理器之间执行字符代码转换。将图形主变量中的数据发送至数据库管理器以及从中接收这些数据（在发送和接收期间，数据是未改变的 DBCS 字符）。如果不使用 WCHARTYPE NOCONVERT，那么您仍可以操作例程中 `wchar_t` 格式的图形数据；但是，您必须手动执行输入和输出转换。

CONVERT 可以在 FENCED 例程中使用，并且它将影响例程中 SQL 语句内的图形数据，但不会影响通过例程的参数来传递的数据。必须使用 NOCONVERT 选项来构建 NOT FENCED 例程。

总的来说，通过例程输入或输出参数传递至例程，或从例程中返回的图形数据都采用 DBCS 格式，而不论使用 WCHARTYPE 选项来预编译例程的方式。

C++ 类型装饰

可以重载 C++ 函数的名称。如果两个同名 C++ 函数具有不同的自变量，那么可以共存，例如:

```
int func( int i )
```

和

```
int func( char c )
```

缺省情况下，C++ 编译器会对函数名进行类型装饰或“重整”。这意味着会将自变量类型名称附加至其函数名以解析函数名，如先前两个示例中的 func__Fi 和 func__Fc。重整名称在每个操作系统上会有不同，因此无法移植显式地使用重整名称的代码。

在 Windows 操作系统上，可以根据 .obj（对象）文件来确定类型装饰函数名。

借助 Windows 上的 Microsoft Visual C++ 编译器，您可以使用 dumpbin 命令根据 .obj（对象）文件来确定类型装饰函数名，如下所示:

```
dumpbin /symbols myprog.obj
```

其中，myprog.obj 是程序对象文件。

在 UNIX 操作系统上，可以使用 nm 命令根据 .o（对象）文件或根据共享库来确定类型装饰函数名。此命令可能会产生相当大的输出，因此建议通过 grep 来传输输出以查找正确的行，如下所示:

```
nm myprog.o | grep myfunc
```

其中，myprog.o 是程序对象文件，而 myfunc 是程序源文件中的函数。

所有这些命令所产生的输出包括具有重整函数名的行。例如，在 UNIX 上，此行如下所示:

```
myfunc__FPlT1PsT3PcN35|     3792|unamex|          | ...
```

在从前面的其中一个命令获取重整函数名后，您可以在相应的命令中使用该函数名。将在本节的后续内容中演示如何使用从前面的 UNIX 示例获取的重整函数名。会以相同方式使用在 Windows 上获取的重整函数名。

在 CREATE 语句中注册例程时，EXTERNAL NAME 子句必须指定重整函数名。例如:

```
CREATE FUNCTION myfunco(...) RETURNS...
       ...
       EXTERNAL NAME '/whatever/path/myprog!myfunc__FPlT1PsT3PcN35'
       ...
```

如果例程库不包含重载的 C++ 函数名，那么可以选择使用 extern "C" 来强制编译器不对函数名进行类型装饰。（请注意，您始终可以重载对 UDF 指定的 SQL 函数名，因为 DB2 数据库系统会根据名称以及接受的参数来解析所调用的库函数。）

```
#include <string.h>
#include <stdlib.h>
#include "sqludf.h"

/*------------------------------------------------------------------*/
/* function fold: output = input string is folded at point indicated  */
/*                         by the second argument.                    */
/*         inputs: CLOB,                input string                  */
/*                 LONG                 position to fold on           */
```

```
/*         output: CLOB                  folded string                 */
/*--------------------------------------------------------------------*/
extern "C" void fold(
      SQLUDF_CLOB       *in1,                   /* input CLOB to fold */
   ...
   ...
}
/* end of UDF: fold */

/*--------------------------------------------------------------------*/
/* function find_vowel:                                               */
/*             returns the position of the first vowel.               */
/*             returns error if no vowel.                             */
/*             defined as NOT NULL CALL                               */
/*         inputs: VARCHAR(500)                                       */
/*         output: INTEGER                                            */
/*--------------------------------------------------------------------*/
extern "C" void findvwl(
      SQLUDF_VARCHAR    *in,                    /* input smallint */
   ...
   ...
}
/* end of UDF: findvwl */
```

在本示例中，UDF fold 和 findvwl 不会由编译器进行类型装饰，而是应该使用其未装饰名称在 CREATE FUNCTION 语句中注册。类似地，如果在 C++ 存储过程或方法中编写了 extern "C"，那么会在 CREATE 语句中使用其未装饰名称。

从 C 和 C++ 过程中返回结果集

您开发的 C 和 C++ 过程可以向使用支持检索过程结果集的 API 实现的调用例程或应用程序返回结果集。

大多数 API 支持检索过程结果集，但嵌入式 SQL 不支持此操作。

结果集的 C 和 C++ 表示是 SQL 游标。在一个过程返回之前，可以将任何已在该过程中声明、打开并且未显式关闭的 SQL 游标返回给调用者。将结果集返回给调用者的顺序就是在该例程中打开游标对象的顺序。要返回结果集，并不需要在 CREATE PROCEDURE 语句或过程实现中指定附加的参数。

先决条件

一般地了解如何创建 C 和 C++ 例程有助于您执行以下过程中用于从 C 或 C++ 过程返回结果的步骤。

创建 C 和 C++ 例程

在 C 或 C++ 嵌入式 SQL 过程中声明的游标不是可滚动游标。

过程

要从 C 或 C++ 过程中返回结果集，请执行下列操作:

1. 在 C 或 C++ 过程的 CREATE PROCEDURE 语句中，除指定任何其他相应的语句之外，您还必须指定 DYNAMIC RESULT SETS 子句（其值等于过程将要返回的最大结果集数）。
2. 在过程声明中，将返回给调用者的结果集不需要参数标记。
3. 在例程的 C 或 C++ 过程实现中，在用于声明主变量的声明节中使用 DECLARE CURSOR 语句来声明游标。游标声明使 SQL 与游标相关联。

4. 在 C 或 C++ 例程代码中，通过执行 OPEN 语句来打开游标。这将执行 DECLARE CURSOR 语句中指定的查询并使查询结果与该游标相关联。
5. 可选：使用 FETCH 语句来访存与该游标相关联的结果集中的行。
6.

   请勿在过程返回到调用者之前的任何位置执行用于关闭游标的 CLOSE 语句。打开的游标将在过程返回时作为结果集返回给调用者。

   在过程返回后，如果有多个游标处于打开状态，那么与那些游标相关联的结果集将按它们的打开顺序返回给调用者。过程返回的结果集数不能超出 DYNAMIC RESULT SETS 子句值所指定的最大数目。如果过程实现中打开的游标数大于 DYNAMIC RESULT SETS 子句所指定的值，那么不会返回多余的结果集。在此情况下，DB2 数据库系统不会发出错误或警告。

一旦成功地完成 C 或 C++ 过程的创建之后，就可以从 DB2 命令行处理器或 DB2 命令窗口中使用 CALL 语句来调用该过程，以便验证是否成功地将结果集返回给调用者。

有关调用过程和其他类型的例程的信息，请参阅:

- 例程调用

创建 C 和 C++ 例程

以创建具有其他实现的外部例程的方式来创建对 C 或 C++ 库进行引用的过程和函数。此任务由几个步骤组成，其中包括构造例程的 CREATE 语句，编写例程实现，预编译、编译和链接代码以及部署源代码。

开始之前

- 了解 C 和 C++ 例程实现。要了解 C 和 C++ 例程的常规信息，请参阅:
  - 第 139 页的第 5 章，『C 和 C++ 例程』
- 必须在客户机上安装附带应用程序开发支持的 IBM 数据服务器客户机。
- 数据库服务器必须正在运行支持 DB2 数据库系统所支持的 C 或 C++ 编译器（用于例程开发）的操作系统。
- 必须在数据库服务器上安装必需的编译器。
- 有权对外部例程执行 CREATE 语句。有关执行 CREATE PROCEDURE 语句或 CREATE FUNCTION 语句所需的特权，请参阅语句的文档。

关于此任务

在下列情况下，您不妨选择实现 C 或 C++ 例程:

- 您需要在例程中封装复杂逻辑，以访问数据库或在数据库外部执行操作。
- 您要求从下列任何项调用封装的逻辑：多个应用程序、CLP、另一个例程（过程、函数（UDF）或方法）或触发器。
- 您最擅长使用诸如 C 或 C++ 之类的嵌入式 SQL 编程语言来编写此逻辑。

过程

1. 使用所选编程语言 C 或 C++ 来编写例程逻辑。

- 包括其他 C 功能所需的任何 C 或 C++ 头文件，以及 SQL 数据类型和 SQL 执行支持所需的 DB2 数据库系统 C 或 C++ 头文件。包括下列头文件: `sqludf.h`、`sql.h`、`sqlda.h`、`sqlca.h` 和 `memory.h`。
- 必须使用其中一种支持的参数样式来实现例程参数特征符。强烈建议将参数样式 SQL 用于所有 C 和 C++ 例程。暂存区和 dbinfo 结构会作为参数传递至 C 和 C++ 例程。有关参数特征符和参数实现的更多信息，请参阅:
  - 第 142 页的『C 和 C++ 例程中的参数』
  - 第 143 页的『参数样式 SQL C 和 C++ 过程』
  - 第 146 页的『参数样式 SQL C 和 C++ 函数』
- 使用针对嵌入式 SQL C 和 C++ 应用程序声明主变量和参数标记的方法来声明主变量和参数标记。确保正确地使用映射至 DB2 SQL 数据类型的数据类型。有关 DB2 和 C 或 C++ 数据类型之间的数据类型映射的更多信息，请参阅:
  - 第 153 页的『C 和 C++ 例程支持的 SQL 数据类型』
- 包括例程逻辑。例程逻辑可以包含 C 或 C++ 编程语言所支持的任何代码。此外，它能够以执行嵌入式 SQL 应用程序的方法来执行嵌入式 SQL 语句。

  有关在嵌入式 SQL 中执行 SQL 语句的更多信息，请参阅:
  - 《*开发嵌入式 SQL 应用程序*》中的『在嵌入式 SQL 应用程序中执行 SQL 语句』
- 如果例程是过程且您需要将结果集返回给例程调用者，那么您不需要该结果集的任何参数。有关返回例程的结果集的更多信息，请参阅:
  - 第 178 页的『从 C 和 C++ 过程中返回结果集』
- 在例程末尾设置例程返回值。

2. 构建代码以产生库文件。 有关如何构建嵌入式 SQL C 和 C++ 例程的信息，请参阅:
   - 第 181 页的『构建 C 和 C++ 例程代码』
3. 将库复制到数据库服务器上的 DB2 *function* 目录。建议将 DB2 例程的相关联库存储至函数目录。要了解有关函数目录的更多信息，请参阅下列任一语句的 EXTERNAL 子句: CREATE PROCEDURE 或 CREATE FUNCTION。

   您可以将库复制到服务器上的另一个目录，但是为了成功调用例程，您必须记下库的标准路径名，因为下一步需要该标准路径名。
4. 动态或静态地执行适合于例程类型的 SQL 语言 CREATE 语句: CREATE PROCEDURE 或 CREATE FUNCTION。
   - 使用值 C 来指定 `LANGUAGE` 子句
   - 使用支持的参数样式（在例程代码中实现）的名称来指定 `PARAMETER STYLE` 子句。强烈建议使用 PARAMETER STYLE SQL。
   - 使用库（将通过下列其中一个值与例程相关联）的名称来指定 `EXTERNAL` 子句:
     - 例程库的标准路径名
     - 例程库的相对路径名（相对于函数目录）

     缺省情况下，DB2 数据库系统将在函数目录中查找库，除非在 EXTERNAL 子句中指定库的标准或相对路径名。

- 如果您的例程是过程且它会将一个或多个结果集返回给调用者，请指定具有数值的 DYNAMIC RESULT SETS。
- 在 CREATE 语句中指定要用来描述例程的特征的任何其他非缺省子句值。

结果

要调用 C 或 C++ 例程，请参阅第 263 页的第 10 章，『调用例程』。

构建 C 和 C++ 例程代码

在编写嵌入式 SQL C 或 C++ 例程实现代码后，必须将代码构建到库，并加以部署，然后才能调用例程。虽然构建嵌入式 SQL C 和 C++ 例程时所需的步骤，与构建嵌入式 SQL C 和 C++ 应用程序时所需的步骤类似，但存在一些差别。如果例程中不存在嵌入式 SQL 语句，那么可以遵循相同的步骤 - 过程将更快、更简单。

过程

可以使用两种方法来构建 C 和 C++ 例程:

- 使用 DB2 样本构建脚本 (UNIX) 或构建批处理文件 (Windows)
- 从 DB2 命令窗口输入 DB2 和 C 或 C++ 编译器命令

例程的 DB2 样本构建脚本和批处理文件设计用于通过缺省的支持编译器，针对特定操作系统构建 DB2 样本例程（过程和用户定义的函数）以及用户创建的例程。
C 和 C++ 分别具有一组 DB2 样本构建脚本和批处理文件。通常，使用构建脚本或批处理文件（可根据需要轻松修改）来构建嵌入式 SQL 例程非常容易，但是，如果也知道如何从 DB2 命令窗口构建例程，往往很有帮助。

使用样本 bldrtn 脚本来构建 C 和 C++ 例程代码

构建 C 和 C++ 例程源代码是创建 C 和 C++ 例程的子任务。可以使用 DB2 样本构建脚本 (UNIX) 和批处理文件 (Windows) 来简单快捷地完成此任务。可以将样本构建脚本用于带有或不带嵌入式 SQL 语句的源代码。构建脚本可以预编译、编译以及链接 C 和 C++ 源代码，否则，必须从命令行执行单独的步骤来完成预编译、编译以及链接。构建脚本还可以将任何程序包与指定的数据库绑定。

开始之前

用于构建 C 和 C++ 例程的样本构建脚本称为 **bldrtn**。这些脚本及其用于构建的样本程序都位于 DB2 目录，如下所示:

- 对于 C: `sqllib/samples/c/`
- 对于 C++: `sqllib/samples/cpp/`

可以使用 **bldrtn** 脚本来构建包含过程和函数实现的源代码文件。脚本完成下列操作:

- 建立与用户指定数据库的连接
- 预编译用户指定的源代码文件
- 将程序包与当前数据库绑定
- 编译并链接源代码以生成共享库
- 将共享库复制到数据库服务器上的 DB2 函数目录

bldrtn 脚本接受两个自变量:

- 不带任何文件扩展名的源代码文件名
- 将与其建立连接的数据库的名称

数据库参数是可选的。如果未提供任何数据库名，那么程序将使用缺省样本数据库。因为必须在数据库所在的相同实例上构建例程，所以用户标识和密码不需要任何自变量。

- 包含一个或多个例程实现的源代码文件。
- 当前 DB2 实例中将创建例程的数据库的名称。

过程

要构建包含一个或多个例程代码实现的源代码文件:

1. 打开 DB2 命令窗口。
2. 将源代码文件复制到 **bldrtn** 脚本文件所在的相同目录。
3. 如果将在样本数据库中创建例程，请输入构建脚本名称，后跟不带 .sqc 或 .sqC 文件扩展名的源代码文件名:

   ```
   bldrtn file-name
   ```

   如果将在另一个数据库中创建例程，请输入构建脚本名称、不带任何文件扩展名的源代码文件名以及数据库名:

   ```
   bldrtn file-name database-name
   ```

   脚本会预编译、编译和链接源代码并产生共享库。然后，脚本会将共享库复制到数据库服务器上的函数目录。
4. 如果这不是您首次构建包含例程实现的源代码文件，请停止然后重新启动数据库以确保 DB2 数据库系统使用的是新版本的共享库。要完成此任务，请在命令行中输入 **db2stop**，接着输入 **db2start**。

下一步做什么

在成功构建例程共享库并将其部署至数据库服务器上的函数目录后，您应该完成与创建 C 和 C++ 例程的任务相关联的步骤。在完成创建例程后，您将能够调用例程。

使用样本构建脚本来构建 C 或 C++ 例程 (UNIX)

关于此任务

DB2 提供了用于编译和链接 C 和 C++ 程序的构建脚本。这些脚本与可以使用这些文件构建的样本程序一起放在 `sqllib/samples/c` 目录（对于 C 例程）和 `sqllib/samples/cpp` 目录（对于 C++ 例程）中。

脚本 `bldrtn` 包含用于构建例程（存储过程和用户定义的函数）的命令。此脚本将例程编译到可以由数据库管理器装入并由客户机应用程序调用的共享库中。

第一个参数 `$1` 指定源文件的名称。第二个参数 `$2` 指定所要连接的数据库的名称。

数据库参数是可选的。如果未提供数据库名，那么程序将使用缺省的 `sample` 数据库。没有用户标识和密码参数，这是因为，必须在数据库所在的实例中构建存储过程。

下列示例说明如何构建包含以下内容的例程共享库:

- 存储过程
- 非嵌入式 SQL 用户定义的函数 (UDF)
- 嵌入式 SQL 用户定义的函数 (UDF)

存储过程共享库

要根据源文件 spserver.sqc (C) 和 spserver.sqC (C++) 来构建样本程序 spserver，请执行下列操作:

1. 如果已连接到sample 数据库，请输入构建脚本名和程序名:

   ```
   bldrtn spserver
   ```

   如果已连接到另一个数据库，那么还需输入数据库名:

   ```
   bldrtn spserver 数据库
   ```

   此脚本将把共享库复制到服务器上的路径 sqllib/function 中。
2. 接着，通过在服务器上运行 spcat 脚本对例程进行编目:

   ```
   spcat
   ```

   此脚本将连接到 sample 数据库，对例程取消编目（如果先前已通过调用 spdrop.db2 对其进行编目的话），然后通过调用 spcreate.db2 对其进行编目，最后断开与数据库的连接。另外，也可以单独调用 spdrop.db2 和 spcreate.db2 脚本。
3. 然后，如果这不是第一次构建存储过程，请停止然后重新启动数据库以确保识别新版本的共享库。要完成此任务，请在命令行中输入 db2stop，接着输入 db2start。

构建共享库 spserver 完成后，您就可以构建需要访问该共享库的客户机应用程序 spclient。

可以使用脚本 bldapp 来构建 spclient。

要调用共享库中的存储过程，请通过输入以下命令来运行样本客户机应用程序:
spclient *database userid password*

其中:

数据库 是所要连接的数据库的名称。名称可以是 sample 或其别名，也可以是另一个数据库名。

用户标识
是有效的用户标识。

密码 是用户标识的有效密码。

此客户机应用程序将访问共享库 spserver 并执行服务器数据库中的多个存储过程函数。输出将被返回到客户机应用程序。

嵌入式 SQL UDF 共享库

要根据源文件 udfemsrv.sqc (C) 和 udfemsrv.sqC (C++) 来构建嵌入式 SQL 用户定义的函数程序 udfemsrv，并且已连接到 sample 数据库，请输入构建脚本名和程序名:

```
bldrtn udfemsrv
```

如果已连接到另一个数据库，那么还需输入数据库名:

```
bldrtn udfemsrv 数据库
```

此脚本将把 UDF 复制到 sqllib/function 目录。

构建 udfemsrv 完成后，您就可以构建需要调用它的客户机应用程序 udfemcli。您可以使用脚本 bldapp 来根据 sqllib/samples/c 中的源文件 udfemcli.sqc 构建 udfemcli 客户机程序。

要调用共享库中的 UDF，请通过输入以下命令来运行此客户机应用程序: udfemcli *database userid password*

其中:

database
是所要连接的数据库的名称。名称可以是 sample 或其别名，也可以是另一个数据库名。

userid 是有效的用户标识。

password
是用户标识的有效密码。

此客户机应用程序将访问共享库 udfemsrv 并执行服务器数据库中的用户定义的函数。输出将被返回到客户机应用程序。

注: 构建嵌入式 SQL 过程时，将忽略 LD_LIBRARY_PATH 操作系统环境变量。

在 Windows 上构建 C/C++ 例程

关于此任务

DB2 提供了用于编译和链接使用 C 和 C++ 编写的 DB2 API 程序和嵌入式 SQL 程序的构建脚本。这些脚本与可以使用这些文件构建的样本程序一起放在 sqllib\samples\c 和 sqllib\samples\cpp 目录中。

批处理文件 bldrtn.bat 包含用于构建嵌入式 SQL 例程（存储过程和用户定义的函数）的命令。这个批处理文件用于在服务器上构建 DLL。它接受 2 个参数，在批处理文件中，它们分别由变量 %1 和 %2 表示。

第一个参数 %1 指定源文件的名称。此批处理文件使用源文件名作为 DLL 名称。第二个参数 %2 指定所要连接的数据库的名称。由于必须在数据库所在的实例中构建 DLL，因此没有用户标识和密码参数。

只有第一个参数（源文件名）是必需的。数据库名是可选的。如果未提供数据库名，那么程序将使用缺省的 sample 数据库。

以下示例说明如何构建包含以下内容的例程 DLL:

- 存储过程
- 非嵌入式 SQL 用户定义的函数 (UDF)
- 嵌入式 SQL 用户定义的函数 (UDF)

存储过程 DLL

要根据 C 源文件 spserver.sqc 或 C++ 源文件 spserver.sqx 来构建 spserver DLL，请执行下列操作:

1. 输入批处理文件名和程序名:

   ```
   bldrtn spserver
   ```

   如果已连接到另一个数据库，那么还需输入数据库名:

   ```
   bldrtn spserver 数据库
   ```

   此批处理文件使用样本程序所在目录中的模块定义文件 spserver.def 来构建 DLL。此批处理文件将 DLL spserver.dll 复制到服务器上的路径 sqllib\function 中。

2. 接着，通过在服务器上运行 spcat 脚本对例程进行编目:

   ```
   spcat
   ```

   此脚本将连接到 sample 数据库，对例程取消编目（如果先前已通过调用 spdrop.db2 对其进行编目的话），然后通过调用 spcreate.db2 对其进行编目，最后断开与数据库的连接。另外，也可以单独调用 spdrop.db2 和 spcreate.db2 脚本。

3. 然后，停止然后重新启动数据库，以便识别新的 DLL。必要时，请为 DLL 设置文件方式，以使 DB2 实例能够对其进行访问。

构建 DLL spserver 完成后，您就可以构建需要调用它的客户机应用程序 spclient。

可以使用批处理文件 bldapp.bat 来构建 spclient。

要调用该 DLL，请通过输入以下命令来运行样本客户机应用程序:

spclient *database userid password*

其中:

数据库 是所要连接的数据库的名称。名称可以是 sample 或其别名，也可以是另一个数据库名。

用户标识
是有效的用户标识。

密码 是用户标识的有效密码。

此客户机应用程序将访问 DLL spserver 并对服务器数据库执行多个例程。输出将被返回到客户机应用程序。

非嵌入式 SQL UDF DLL

要根据源文件 udfsrv.c 来构建用户定义的函数 udfsrv，请输入:

```
bldrtn udfsrv
```

此批处理文件使用样本程序文件所在目录中的模块定义文件 udfsrv.def 来构建用户定义的函数 DLL。此批处理文件将用户定义的函数 DLL udfsrv.dll 复制到服务器上的路径 sqllib\function 中。

构建 udfsrv 完成后，您就可以构建需要调用它的客户机应用程序 udfcli。我们提供了 CLI 以及此程序的嵌入式 SQL C 和 C++ 版本。

可以使用批处理文件 bldapp 来根据 sqllib\samples\cli 中的 udfcli.c 源文件构建 CLI udfcli 程序。

可以使用批处理文件 bldapp 来根据 sqllib\samples\c 中的 udfcli.sqc 源文件构建嵌入式 SQL C udfcli 程序。

可以使用批处理文件 bldapp 来根据 sqllib\samples\cpp 中的 udfcli.sqx 源文件构建嵌入式 SQL C++ udfcli 程序。

要运行 UDF，请输入:

```
udfcli
```

调用应用程序将调用 udfsrv DLL 中的 ScalarUDF 函数。

嵌入式 SQL UDF DLL

要根据 sqllib\samples\c 中的 C 源文件 udfemsrv.sqc 或 sqllib\samples\cpp 中的 C++ 源文件 udfemsrv.sqx 来构建嵌入式 SQL 用户定义的函数库 udfemsrv，请输入:

```
bldrtn udfemsrv
```

如果已连接到另一个数据库，那么还需输入数据库名:

```
bldrtn udfemsrv 数据库
```

此批处理文件使用样本程序所在目录中的模块定义文件 udfemsrv.def 来构建用户定义的函数 DLL。此批处理文件将用户定义的函数 DLL udfemsrv.dll 复制到服务器上的路径 sqllib\function 中。

构建 udfemsrv 完成后，您就可以构建需要调用它的客户机应用程序 udfemcli。您可以使用批处理文件 bldapp 来根据 sqllib\samples\c 中的 C 源文件 udfemcli.sqc 或 sqllib\samples\cpp 中的 C++ 源文件 udfemcli.sqx 构建 udfemcli。

要运行 UDF，请输入:

```
udfemcli
```

调用应用程序将调用 udfemsrv DLL 中的 UDF。

从 DB2 命令窗口构建 C 和 C++ 例程代码

构建 C 和 C++ 例程源代码是创建 C 和 C++ 例程的子任务。可以手动从命令行完成此任务。可以遵循相同的过程，而不论 C 或 C++ 例程代码中是否存在嵌入式 SQL 语句。任务步骤包括预编译、编译和链接包含例程实现的 C 和 C++ 源代码，绑定生成的程序包（如果有嵌入式 SQL 语句）以及部署例程库。在测试预编译器、编译器或绑定选项的用法时，如果要将绑定例程程序包延迟到以后，或您正在开发定制的构建脚本，那么可选择从 DB2 命令窗口执行此任务。

开始之前

作为替代选择，您可以使用 DB2 样本构建脚本来简化此任务。请参阅: 使用样本构建脚本来构建嵌入式 SQL C 和 C++ 例程代码。

- 包含一个或多个嵌入式 SQL C 或 C++ 例程实现的源代码文件。
- 当前 DB2 实例中将创建例程的数据库的名称。

- 构建 C 和 C++ 例程时所需的操作特定编译和链接选项。请参阅本主题相关链接部分所引用的主题。

过程

要构建包含一个或多个例程代码实现的源代码文件:

1. 打开 DB2 命令窗口。
2. 浏览至包含源代码文件的目录。
3. 将建立与数据库（将在其中创建例程）的连接。
4. 预编译源代码文件。
5. 将生成的程序包与数据库绑定。
6. 编译源代码文件。
7. 链接源代码文件以生成共享库。 这需要利用所使用编译器的一些特定于 DB2 数据库的编译和链接选项。
8. 将共享库复制到数据库服务器上的 DB2 函数目录。
9. 如果这不是您首次构建包含例程实现的源代码文件，请停止然后重新启动数据库以确保 DB2 数据库系统使用的是新版本的共享库。可通过先发出 **db2stop** 命令然后发出 **db2start** 命令来完成此操作。

结果

在成功构建和部署例程库后，您应该完成与创建 C 和 C++ 例程的任务相关联的步骤。创建 C 和 C++ 例程的过程包括针对源代码文件中实现的每个例程执行 CREATE 语句的步骤。也必须先完成此步骤，才能调用例程。

示例

下列示例演示如何重新构建包含例程实现的嵌入式 SQL C++ 源代码文件 myfile.sqC。将在使用缺省的支持 IBM VisualAge® C++ 编译器来生成 32 位例程库的 AIX 操作系统上，构建这些例程。

1. 打开 DB2 命令窗口。
2. 浏览至包含源代码文件的目录。
3. 将建立与数据库（将在其中创建例程）的连接。

   ```
   db2 connect to database-name
   ```

4. 使用 PREPARE 命令来预编译源代码文件。

   ```
   db2 prep myfile.sqC bindfile
   ```

   预编译器将生成输出，用以指示预编译是成功继续，还是发生任何错误。此步骤会生成名为 myfile.bnd 的绑定文件，在下一步中可使用它生成程序包。

5. 使用 **BIND** 命令对所生成的程序包与数据库进行绑定。

   ```
   db2 bind myfile.bnd
   ```

   绑定实用程序将生成输出，用以指示绑定是成功继续，还是发生任何错误。

6. 使用建议的编译和链接选项来编译源代码文件:

   ```
   xlC_r -qstaticinline -I$HOME/sqllib/include -c $myfile.C
   ```

如果发生任何错误，那么编译器将生成输出。此步骤会生成名为 myfile.exp 的导出文件。

7. 链接源代码文件以生成共享库。

```
xlC_r -qmkshrobj -o $1 $1.o -L$ HOME/sqllib/include/lib32 -lDB2
```

如果发生任何错误，那么链接器将生成输出。此步骤会生成共享库文件名 myfile。

8. 将共享库复制到数据库服务器上的 DB2 函数目录。

```
rm -f ~HOME/sqllib/function/myfile
cp myfile $HOME/sqllib/function/myfile
```

此步骤确保例程库位于 DB2 数据库系统将在其中查找例程库的缺省目录。请参阅有关创建 C 和 C++ 例程的主题，以了解有关如何部署例程库的更多信息。

9. 停止然后重新启动数据库，因为此步骤会重新构建先前构建的例程源代码文件。

```
db2stop
db2start
```

通常，可以使用操作特定样本构建脚本非常轻松地构建 C 和 C++ 例程，这些脚本也可用作如何从命令行构建例程的参考。

注: 构建嵌入式 SQL 过程时，将忽略 **LD_LIBRARY_PATH** 操作系统环境变量。

C 和 C++ 例程的编译和链接选项

AIX C 例程的编译和链接选项

使用 AIX IBM C 编译器来构建 C 例程（存储过程和用户定义的函数）时，使用 DB2 中提供的编译和链接选项，如 bldrtn 构建脚本中所示。

bldrtn 的编译和链接选项

编译选项:

xlc_r 使用 IBM C 编译器的多线程版本，以使例程能够在其他例程所在的进程中运行（THREADSAFE）或者在引擎本身中运行（NOT FENCED）。

$EXTRA_CFLAG
对于已启用 64 位支持的实例，此选项包含“-q64”；否则，它不包含任何值。

-I$DB2PATH/include
指定 DB2 包含文件的位置。例如: $HOME/sqllib/include。

-c 只执行编译；不进行链接。编译和链接是不同的步骤。

链接选项:

xlc_r 使用编译器的多线程版本作为链接程序的前端。

$EXTRA_CFLAG
对于已启用 64 位支持的实例，此选项包含“-q64”；否则，它不包含任何值。

-qmkshrobj
创建共享库。

-o $1 指定输出文件名。

$1.o 指定对象文件。

-ldb2 与 DB2 库链接。

-L$DB2PATH/$LIB
指定 DB2 运行时共享库的位置。例如: $HOME/sqllib/$LIB。如果未指定 -L 选项，那么编译器将采用以下路径: /usr/lib:/lib。

-bE:$1.exp
指定导出文件。导出文件包含例程的列表。

请参阅编译器文档，以了解其他编译器选项。

AIX C++ 例程的编译和链接选项

使用 AIX IBM XL C/C++ 编译器来构建 C++ 例程（存储过程和用户定义的函数）时，使用 DB2 中提供的编译和链接选项，如 bldrtn 构建脚本中所示。

bldrtn 的编译和链接选项

编译选项:

xlC_r 使用 IBM XL C/C++ 编译器的多线程版本，以使例程能够在其他例程所在的进程中运行（THREADSAFE）或者在引擎本身中运行（NOT FENCED）。

$EXTRA_CFLAG
对于已启用 64 位支持的实例，此选项包含“-q64”；否则，它不包含任何值。

-I$DB2PATH/include
指定 DB2 包含文件的位置。例如: $HOME/sqllib/include。

-c 只执行编译；不进行链接。编译和链接是不同的步骤。

链接选项:

xlC_r 使用编译器的多线程版本作为链接程序的前端。

$EXTRA_CFLAG
对于已启用 64 位支持的实例，此选项包含“-q64”；否则，它不包含任何值。

-qmkshrobj
创建共享库。

-o $1 指定输出作为共享库文件。

$1.o 指定程序对象文件。

-L$DB2PATH/$LIB
指定 DB2 运行时共享库的位置。例如: $HOME/sqllib/$LIB。如果未指定 -L 选项，那么编译器将采用以下路径: /usr/lib:/lib。

-ldb2 与 DB2 库链接。

-bE:$1.exp
指定导出文件。导出文件包含例程的列表。

请参阅编译器文档，以了解其他编译器选项。

HP-UX C 例程的编译和链接选项

使用 HP-UX C 编译器来构建 C 例程（存储过程和用户定义的函数）时，使用 DB2 中提供的编译和链接选项，如 bldrtn 构建脚本中所示。

bldrtn 的编译和链接选项

编译选项:

cc C 编译器。

$EXTRA_CFLAG
如果 HP-UX 平台为 IA64 并且启用了 64 位支持，那么此标志包含值 **+DD64**；如果启用了 32 位支持，那么此标志包含值 **+DD32**。

+DD64 必须使用此选项才能为 IA64 上的 HP-UX 生成 64 位代码。

+DD32 必须使用此选项才能为 IA64 上的 HP-UX 生成 32 位代码。

+u1 允许访问未对齐的数据。仅当应用程序使用未对齐的数据时，才需要使用此选项。

+z 生成与位置无关的代码。

-Ae 启用 HP ANSI 扩展方式。

-I$DB2PATH/include
指定 DB2 包含文件的位置。例如: -I$DB2PATH/include。

-D_POSIX_C_SOURCE=199506L
定义用于确保 _REENTRANT 的 POSIX 线程库选项，以使例程能够在其他例程所在的进程中运行（THREADSAFE）或者在引擎本身中运行（NOT FENCED）。

-c 只执行编译；不进行链接。编译和链接是不同的步骤。

链接选项:

ld 使用链接程序进行链接。

-b 创建共享库，而不是创建正常的可执行文件。

-o $1 指定输出作为共享库文件。

$1.o 指定程序对象文件。

$EXTRA_LFLAG
指定运行时路径。如果设置了此选项，那么对于 32 位，它包含值“+b$HOME/sqllib/lib32”，对于 64 位，它包含值“+b$HOME/sqllib/lib64”。如果未设置此选项，那么它不包含任何值。

-L$DB2PATH/$LIB
指定 DB2 运行时共享库的位置。对于 32 位: $HOME/sqllib/lib32; 对于 64 位: $HOME/sqllib/lib64。

-ldb2 与 DB2 库链接。

-lpthread
与 POSIX 线程库链接。

请参阅编译器文档，以了解其他编译器选项。

HP-UX C++ 例程的编译和链接选项

使用 HP-UX C++ 编译器来构建 C++ 例程（存储过程和用户定义的函数）时，使用 DB2 中提供的编译和链接选项，如 bldrtn 构建脚本中所示。

bldrtn 的编译和链接选项

编译选项:

aCC HP aC++ 编译器。

$EXTRA_CFLAG
如果 HP-UX 平台为 IA64 并且启用了 64 位支持，那么此标志包含值 **+DD64**；如果启用了 32 位支持，那么此标志包含值 **+DD32**。

+DD64 必须使用此选项才能为 IA64 上的 HP-UX 生成 64 位代码。

+DD32 必须使用此选项才能为 IA64 上的 HP-UX 生成 32 位代码。

+u1 允许访问未对齐的数据。

+z 生成与位置无关的代码。

-ext 启用各种 C++ 扩展，其中包括“long long”支持。

-mt 启用 HP aC++ 编译器的线程支持，以使例程能够在其他例程所在的进程中运行（THREADSAFE）或者在引擎本身中运行（NOT FENCED）。

-I$DB2PATH/include
指定 DB2 包含文件的位置。例如: $DB2PATH/include。

-c 只执行编译；不进行链接。编译和链接是不同的步骤。

链接选项:

aCC 使用 HP aC++ 编译器作为链接程序的前端。

$EXTRA_CFLAG
如果 HP-UX 平台为 IA64 并且启用了 64 位支持，那么此标志包含值 **+DD64**；如果启用了 32 位支持，那么此标志包含值 **+DD32**。

+DD64 必须使用此选项才能为 IA64 上的 HP-UX 生成 64 位代码。

+DD32 必须使用此选项才能为 IA64 上的 HP-UX 生成 32 位代码。

-mt 启用 HP aC++ 编译器的线程支持，以使例程能够在其他例程所在的进程中运行（THREADSAFE）或者在引擎本身中运行（NOT FENCED）。

-b 创建共享库，而不是创建正常的可执行文件。

-o $1 指定可执行文件。

$1.o 指定程序对象文件。

$EXTRA_LFLAG
指定运行时路径。如果设置了此选项，那么对于 32 位，它包含值 -Wl,+b$HOME/sqllib/lib32，对于 64 位，它包含值 -Wl,+b$HOME/sqllib/lib64。如果未设置此选项，那么它不包含任何值。

-L$DB2PATH/$LIB
指定 DB2 运行时共享库的位置。对于 32 位: “$HOME/sqllib/lib32”；对于 64 位: “$HOME/sqllib/lib64”。

-ldb2 与 DB2 库链接。

请参阅编译器文档，以了解其他编译器选项。

Linux C 例程的编译和链接选项

使用 Linux C 编译器来构建 C 例程（存储过程和用户定义的函数）时，使用 DB2 中提供的编译和链接选项，如 `bldrtn` 构建脚本中所示。

bldrtn 的编译和链接选项

编译选项:

$CC gcc 或 xlc_r 编译器。

$EXTRA_C_FLAGS
包含下列其中一个标志:

- `-m31`（仅限于 Linux for zSeries®），用于构建 32 位库;
- `-m32`（仅限于 Linux for x86、x64 和 POWER®），用于构建 32 位库;
- `-m64`（仅限于 Linux for zSeries、POWER 和 x64），用于构建 64 位库;或者
- 不包含任何值（Linux for IA64），用于构建 64 位库。

-I$DB2PATH/include
指定 DB2 包含文件的位置。

-c 只执行编译；不进行链接。此脚本文件包含独立的编译和链接步骤。

-D_REENTRANT
定义 _REENTRANT，以使例程能够在其他例程所在的进程中运行（THREADSAFE）或者在引擎本身中运行（NOT FENCED）。

链接选项:

$CC gcc 或 xlc_r 编译器；使用编译器作为链接程序的前端。

$LINK_FLAGS
包含值“$EXTRA_C_FLAGS $SHARED_LIB_FLAG”。

$EXTRA_C_FLAGS
包含下列其中一个标志:

- `-m31`（仅限于 Linux for zSeries），用于构建 32 位库;
- `-m32`（仅限于 Linux for x86、x64 和 POWER），用于构建 32 位库;
- `-m64`（仅限于 Linux for zSeries、POWER 和 x64），用于构建 64 位库;或者
- 不包含任何值（Linux for IA64），用于构建 64 位库。

$SHARED_LIB_FLAG
包含 `-shared`（对于 gcc 编译器）或 `-qmkshrobj`（对于 xlc_r 编译器）。

-o $1 指定可执行文件。

$1.o 包括程序对象文件。

$EXTRA_LFLAG
指定 DB2 共享库在运行时的位置。对于 32 位，它包含值“-Wl,-rpath,$DB2PATH/lib32”。对于 64 位，它包含值“-Wl,-rpath,$DB2PATH/lib64”。

-L$DB2PATH/$LIB
指定 DB2 静态库和共享库在链接时的位置。例如，对于 32 位：$HOME/sqllib/lib32；对于 64 位：$HOME/sqllib/lib64。

-ldb2 与 DB2 库链接。

-lpthread
与 POSIX 线程库链接。

请参阅编译器文档，以了解其他编译器选项。

Linux C++ 例程的编译和链接选项

以下是使用 Linux C++ 编译器来构建 C++ 例程（存储过程和用户定义的函数）时使用的编译和链接选项，如 bldrtn 构建脚本中所示。

bldrtn 的编译和链接选项

编译选项:

g++ GNU/Linux C++ 编译器。

$EXTRA_C_FLAGS
包含下列其中一个标志:

- -m31（仅限于 Linux for zSeries），用于构建 32 位库;
- -m32（仅限于 Linux for x86、x64 和 POWER），用于构建 32 位库;
- -m64（仅限于 Linux for zSeries、POWER 和 x64），用于构建 64 位库;或者
- 不包含任何值（Linux for IA64），用于构建 64 位库。

-fpic 生成与位置无关的代码。

-I$DB2PATH/include
指定 DB2 包含文件的位置。

-c 只执行编译；不进行链接。此脚本文件包含独立的编译和链接步骤。

-D_REENTRANT
定义 _REENTRANT，以使例程能够在其他例程所在的进程中运行（THREADSAFE）或者在引擎本身中运行（NOT FENCED）。

链接选项:

g++ 使用编译器作为链接程序的前端。

$EXTRA_C_FLAGS
包含下列其中一个标志:

- -m31（仅限于 Linux for zSeries），用于构建 32 位库;
- -m32（仅限于 Linux for x86、x64 和 POWER），用于构建 32 位库;

- **-m64**（仅限于 Linux for zSeries、POWER 和 x64），用于构建 64 位库；或者
- 不包含任何值（Linux for IA64），用于构建 64 位库。

-shared
生成共享库。

-o $1 指定可执行文件。

$1.o 包括程序对象文件。

$EXTRA_LFLAG
指定 DB2 共享库在运行时的位置。对于 32 位，它包含值“-W1,-rpath,$DB2PATH/lib32”。对于 64 位，它包含值“-Wl,-rpath,$DB2PATH/lib64”。

-L$DB2PATH/$LIB
指定 DB2 静态库和共享库在链接时的位置。例如，对于 32 位：`$HOME/sqllib/lib32`；对于 64 位：`$HOME/sqllib/lib64`。

-ldb2 与 DB2 库链接。

-lpthread
与 POSIX 线程库链接。

请参阅编译器文档，以了解其他编译器选项。

Solaris C 例程的编译和链接选项

以下是您使用 Forte C 编译器来构建 C 例程（存储过程和用户定义的函数）时，DB2 建议您使用的编译和链接选项，如 `bldrtn` 构建脚本中所示。

bldrtn 的编译和链接选项

编译选项:

cc C 编译器。

-xarch=$CFLAG_ARCH
此选项确保与 `libdb2.so` 链接时，编译器生成有效的可执行文件。$CFLAG_ARCH 的值的设置如下所示:
- “v8plusa”: Solaris SPARC 上的 32 位应用程序
- “v9”: Solaris SPARC 上的 64 位应用程序
- “sse2”: Solaris x64 上的 32 位应用程序
- “amd64”: Solaris x64 上的 64 位应用程序

-mt 启用多线程支持，以使例程能够在其他例程所在的进程中运行（THREADSAFE）或者在引擎本身中运行（NOT FENCED）。

-DUSE_UI_THREADS
启用 Sun 的“UNIX 国际”线程 API。

-Kpic 为共享库生成与位置无关的代码。

-I$DB2PATH/include
指定 DB2 包含文件的位置。

-c 只执行编译；不进行链接。此脚本包含独立的编译和链接步骤。

链接选项:

cc 使用编译器作为链接程序的前端。

-xarch=$CFLAG_ARCH
此选项确保与 libdb2.so 链接时，编译器生成有效的可执行文件。$CFLAG_ARCH 的值设置为“v8plusa”（表示 32 位）或“v9”（表示 64 位）。

-mt 因为 DB2 库是使用 -mt 进行链接的，所以此选项是必需的。

-G 生成共享库。

-o $1 指定可执行文件。

$1.o 包括程序对象文件。

-L$DB2PATH/$LIB
指定 DB2 静态库和共享库在链接时的位置。例如，对于 32 位: $HOME/sqllib/lib32；对于 64 位: $HOME/sqllib/lib64。

$EXTRA_LFLAG
指定 DB2 共享库在运行时的位置。对于 32 位，包含值“-R$DB2PATH/lib32”；对于 64 位，包含值“-R$DB2PATH/lib64”。

-ldb2 与 DB2 库链接。

请参阅编译器文档，以了解其他编译器选项。

Solaris C++ 例程的编译和链接选项

以下是您使用 Forte C++ 编译器来构建 C++ 例程（存储过程和用户定义的函数）时，DB2 建议您使用的编译和链接选项，如 bldrtn 构建脚本中所示。

bldrtn 的编译和链接选项

编译选项:

CC C++ 编译器。

-xarch=$CFLAG_ARCH
此选项确保与 libdb2.so 链接时，编译器生成有效的可执行文件。$CFLAG_ARCH 的值的设置如下所示:

- “v8plusa”: Solaris SPARC 上的 32 位应用程序
- “v9”: Solaris SPARC 上的 64 位应用程序
- “sse2”: Solaris x64 上的 32 位应用程序
- “amd64”: Solaris x64 上的 64 位应用程序

-mt 启用多线程支持，以使例程能够在其他例程所在的进程中运行（THREADSAFE）或者在引擎本身中运行（NOT FENCED）。

-DUSE_UI_THREADS
启用 Sun 的“UNIX 国际”线程 API。

-Kpic 为共享库生成与位置无关的代码。

-I$DB2PATH/include
指定 DB2 包含文件的位置。

-c 只执行编译；不进行链接。此脚本包含独立的编译和链接步骤。

链接选项:

CC 使用编译器作为链接程序的前端。

-xarch=$CFLAG_ARCH
此选项确保与 libdb2.so 链接时，编译器生成有效的可执行文件。$CFLAG_ARCH 的值设置为“v8plusa”（表示 32 位）或“v9”（表示 64 位）。

-mt 因为 DB2 库是使用 -mt 进行链接的，所以此选项是必需的。

-G 生成共享库。

-o $1 指定可执行文件。

$1.o 包括程序对象文件。

-L$DB2PATH/$LIB
指定 DB2 静态库和共享库在链接时的位置。例如，对于 32 位: $HOME/sqllib/lib32；对于 64 位: $HOME/sqllib/lib64。

$EXTRA_LFLAG
指定 DB2 共享库在运行时的位置。对于 32 位，包含值“-R$DB2PATH/lib32”；对于 64 位，包含值“-R$DB2PATH/lib64”。

-ldb2 与 DB2 库链接。

请参阅编译器文档，以了解其他编译器选项。

Windows C 和 C++ 例程的编译和链接选项

在 Windows 上使用 Microsoft Visual C++ 编译器来构建 C 和 C++ 例程（存储过程和用户定义的函数）时，使用 DB2 中提供的编译和链接选项，如 bldrtn.bat 批处理文件中所示。

bldrtn 的编译和链接选项

编译选项:

%BLDCOMP%
编译器的变量。缺省值为 cl，即 Microsoft Visual C++ 编译器。另外，还可以将其设置为 icl（用于 32 位和 64 位应用程序的 Intel C++ 编译器）或 ecl（用于 Itanium 64 位应用程序的 Intel C++ 编译器）。

-Zi 启用调试信息。

-Od 禁止优化。

-c 只执行编译；不进行链接。编译和链接是不同的步骤。

-W2 输出警告、错误、严重和不可恢复错误消息。

-DWIN32
Windows 操作系统所必需的编译器选项。

-MD 使用 MSVCRT.LIB 进行链接。

链接选项:

link 使用链接程序进行链接。

-debug 包括调试信息。

-out:%1.dll
构建 .DLL 文件。

%1.obj 包括对象文件。

db2api.lib
与 DB2 库链接。

-def:%1.def
模块定义文件。

请参阅编译器文档，以了解其他编译器选项。

使用配置文件来构建以 C 或 C++ 编写的嵌入式 SQL 存储过程

在 AIX 上，可以使用 sqllib/samples/c 和 sqllib/samples/cpp 中的配置文件 stp.icc 来构建以 C 和 C++ 语言编写的 DB2 嵌入式 SQL 存储过程。

过程

要使用配置文件来根据源文件 spserver.sqc 构建嵌入式 SQL 存储过程共享库 spserver，请执行下列操作:

1. 通过输入以下命令，将 STP 环境变量设置为程序名:
   - 对于 bash 或 Korn shell:
     ```
     export STP=spserver
     ```
   - 对于 C shell:
     ```
     setenv STP spserver
     ```
2. 如果工作目录中有一个使用 stp.icc 文件构建另一个程序时生成的 stp.ics 文件，请使用以下命令来删除 stp.ics 文件:
   ```
   rm stp.ics
   ```
   不必删除先前为您将要再次构建的同一程序生成的现有 stp.ics 文件。
3. 通过输入以下命令，编译样本程序:
   ```
   vacbld stp.icc
   ```
   注: **vacbld** 命令由 VisualAge C++ 提供。

下一步做什么

存储过程共享库将被复制到服务器上的 sqllib/function 路径中。

接着，通过在服务器上运行 **spcat** 脚本对共享库中的存储过程进行编目:

```
spcat
```

此脚本将连接到 sample 数据库，对存储过程取消编目（如果先前已通过调用 **spdrop.db2** 对其进行编目的话），然后通过调用 **spcreate.db2** 对其进行编目，最后断开与数据库的连接。另外，也可以单独调用 **spdrop.db2** 和 **spcreate.db2** 脚本。

然后，停止并重新启动数据库，以便识别新的共享库。必要时，请为共享库设置文件方式，以使 DB2 实例能够对其进行访问。

构建存储过程共享库 spserver 完成后，您就可以构建需要调用其中的存储过程的客户机应用程序 spclient。您可以使用配置文件 emb.icc 来构建 spclient。

要调用此存储过程，请通过输入以下命令来运行样本客户机应用程序：spclient *database userid password*

其中：

数据库 是所要连接的数据库的名称。此名称可以是 sample 或其远程别名，也可以是另外某个名称。

用户标识
是有效的用户标识。

密码 是有效的密码。

此客户机应用程序将访问共享库 spserver 并执行服务器数据库中的多个存储过程函数。输出将被返回到客户机应用程序。

使用配置文件来构建以 C 或 C++ 编写的用户定义的函数（AIX）

关于此任务

sqllib/samples/c 和 sqllib/samples/cpp 中的配置文件 udf.icc 允许您在 AIX 上构建以 C 和 C++ 语言编写的用户定义的函数。

要使用配置文件来根据源文件 udfsrv.c 构建用户定义的函数程序 udfsrv，请执行下列操作：

过程

1. 通过输入以下命令，将 UDF 环境变量设置为程序名：
   - 对于 bash 或 Korn shell：
     ```
     export UDF=udfsrv
     ```
   - 对于 C shell：
     ```
     setenv UDF udfsrv
     ```
2. 如果工作目录中有一个使用 udf.icc 文件构建另一个程序时生成的 udf.ics 文件，请使用以下命令来删除 udf.ics 文件：
   ```
   rm udf.ics
   ```
   不必删除先前为您将要再次构建的同一程序生成的现有 udf.ics 文件。
3. 通过输入以下命令，编译样本程序：
   ```
   vacbld udf.icc
   ```
   注：vacbld 命令由 VisualAge C++ 提供。

下一步做什么

UDF 库将被复制到服务器上的 sqllib/function 路径中。

必要时，请为用户定义的函数设置文件方式，以使 DB2 实例能够运行该函数。

构建 udfsrv 完成后，您就可以构建需要调用它的客户机应用程序 udfcli。我们提供了此程序的 DB2 CLI 版本和嵌入式 SQL 版本。

您可以使用配置文件 cli.icc 来根据 sqllib/samples/cli 中的源文件 udfcli.c 构建 DB2 CLI udfcli 程序。

您可以使用配置文件 emb.icc 来根据 sqllib/samples/c 中的源文件 udfcli.sqc 构建嵌入式 SQL udfcli 程序。

要调用此 UDF，请通过输入可执行文件名来运行样本调用应用程序:

```
udfcli
```

此调用应用程序将调用 udfsrv 库中的 ScalarUDF 函数。

重建 DB2 例程共享库

一旦装入用于存储过程和用户定义的函数的共享库，DB2 数据库系统就会对那些库进行高速缓存。

如果您正在开发例程，那么您可能想测试多次装入同一个共享库，此高速缓存操作可能会导致无法采用共享库的最新版本。避免高速缓存问题的方法取决于例程类型。

1. **受防护但不具有线程安全性的例程。**数据库管理器配置关键字 **keepfenced** 的缺省值为 YES。此将使处于受防护方式的进程保持活动。此缺省设置会干扰库的重新装入。在开发 FENCED NOT THREADSAFE 例程时，最好是将此关键字的值更改为 NO，然后在您准备装入最终版本共享库时，将其改回 YES。有关更多信息，请参阅『更新数据库管理器配置文件』。
2. **可信或具有线程安全性的例程。**除 SQL 例程（包括 SQL 过程）以外，将 DB2 例程库用于可信或具有线程安全性的例程时，确保采用该库的更新版本的唯一方法是通过从命令行依次输入 **db2stop** 和 **db2start** 来重新启动 DB2 例程。对于 SQL 例程而言，不需要执行此操作，这是因为，重新创建 SQL 例程时，编译器将使用新的唯一库名以防止可能的冲突。

对于除 SQL 例程以外的例程而言，还可以使用名称不同的库（例如，foo.a 变为 foo.1.a）创建新版本例程，然后对新库使用 ALTER PROCEDURE 或 ALTER FUNCTION SQL 语句，从而避免高速缓存问题。

更新数据库管理器配置文件

关于此任务

此文件包含用于应用程序开发的重要设置。

关键字 **KEEPFENCED** 具有缺省值 YES。对于 FENCED NOT THREADSAFE 例程（存储过程和 UDF），这将保持例程进程处于活动状态。在开发这些例程时，最好是将此关键字的值更改为 NO，然后在您准备装入最终版本共享库时，将其改回为 YES。有关更多信息，请参阅『重建 DB2 例程共享库』。

注: 在 DB2 数据库产品的低级版本中，**KEEPFENCED** 被称为 **KEEPDARI**。

对于 Java 应用程序开发，您需要使用 Java Development Kit 的安装路径来更新 **JDK_PATH** 关键字。

注: 在 DB2 数据库产品的低级版本中，**JDK_PATH** 被称为 **JDK11_PATH**。

过程

要更改这些设置，请输入:

```
db2 update dbm cfg using keyword value
```

例如，要将关键字 **KEEPFENCED** 设为 NO，请执行下列操作:

```
db2 update dbm cfg using KEEPFENCED NO
```

要将关键字 **JDK_PATH** 设为目录 /home/db2inst/jdk13，请执行下列操作:

```
db2 update dbm cfg using JDK_PATH /home/db2inst/jdk13
```

结果

要查看数据库管理器配置文件中的当前设置，请输入:

```
db2 get dbm cfg
```

注: 在 Windows 上，您需要在 DB2 命令窗口中输入这些设置。

第 6 章 COBOL 过程

使用编写 COBOL 子程序的相似方法来编写 COBOL 过程。

在 COBOL 过程中处理参数

必须在 LINKAGE SECTION 中声明过程所接受或传递的每个参数。例如，下列代码片段来自接受两个 IN 参数（CHAR(15) 和 INT）且传递一个 OUT 参数 (INT) 的过程:

```
LINKAGE SECTION.
01  IN-SPERSON    PIC X(15).
01  IN-SQTY       PIC S9(9)  USAGE COMP-5.
01  OUT-SALESSUM  PIC S9(9)  USAGE COMP-5.
```

确保您声明的 COBOL 数据类型会正确映射至 SQL 数据类型。有关 SQL 与 COBOL 之间的数据类型映射的详细列表，请参阅“COBOL 支持的 SQL 数据类型”。

然后，必须在 PROCEDURE DIVISION 中列出每个参数。下列示例显示对应于先前 LINKAGE SECTION 示例中的参数定义的 PROCEDURE DIVISION。

```
PROCEDURE DIVISION USING IN-SPERSON
                         IN-SQTY
                         OUT-SALESSUM.
```

退出 COBOL 过程

要正确地退出过程，请使用下列命令:

```
MOVE SQLZ-HOLD-PROC TO RETURN-CODE.
GOBACK.
```

使用这些命令，过程会正确地返回至客户机应用程序。这在本地 COBOL 客户机应用程序调用过程时特别重要。

构建 COBOL 过程时，强烈建议使用针对您的操作系统和编译器编写的构建脚本。可在 sqllib/samples/cobol_mf 目录中找到 Micro Focus COBOL 的构建脚本。可在 sqllib/samples/cobol 目录中找到 IBM COBOL 的构建脚本。

下列是 COBOL 过程示例，此过程可接受两个输入参数，然后返回一个输出参数和一个结果集:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.    "NEWSALE".
DATA DIVISION.

WORKING-STORAGE SECTION.
01  INSERT-STMT.
    05  FILLER   PIC X(24) VALUE "INSERT INTO SALES (SALES".
    05  FILLER   PIC X(24) VALUE "_PERSON,SALES) VALUES ('".
    05  SPERSON  PIC X(16).
    05  FILLER   PIC X(2) VALUE "',".
    05  SQTY     PIC S9(9).
    05  FILLER   PIC X(1) VALUE ")".
    EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01  INS-SMT-INF.
    05  INS-STMT.
    49  INS-LEN   PIC S9(4) USAGE COMP.
    49  INS-TEXT  PIC X(100).
```

```
       01  SALESSUM      PIC S9(9)  USAGE COMP-5.
           EXEC SQL END DECLARE SECTION END-EXEC.
           EXEC SQL INCLUDE SQLCA END-EXEC.

       LINKAGE SECTION.
       01  IN-SPERSON    PIC X(15).
       01  IN-SQTY       PIC S9(9)  USAGE COMP-5.
       01  OUT-SALESSUM  PIC S9(9)  USAGE COMP-5.

       PROCEDURE DIVISION USING IN-SPERSON
                                IN-SQTY
                                OUT-SALESSUM.
       MAINLINE.
           MOVE 0 TO SQLCODE.
           PERFORM INSERT-ROW.
           IF SQLCODE IS NOT EQUAL TO 0
              GOBACK
           END-IF.
           PERFORM SELECT-ROWS.
           PERFORM GET-SUM.
           GOBACK.
       INSERT-ROW.
           MOVE IN-SPERSON TO SPERSON.
           MOVE IN-SQTY TO SQTY.
           MOVE           INSERT-STMT TO INS-TEXT.
           MOVE LENGTH OF INSERT-STMT TO INS-LEN.
           EXEC SQL EXECUTE IMMEDIATE :INS-STMT END-EXEC.
       GET-SUM.
           EXEC SQL
              SELECT SUM(SALES) INTO :SALESSUM FROM SALES
           END-EXEC.
           MOVE SALESSUM TO OUT-SALESSUM.
       SELECT-ROWS.
           EXEC SQL
              DECLARE CUR CURSOR WITH RETURN FOR SELECT * FROM SALES
           END-EXEC.
           IF SQLCODE = 0
              EXEC SQL OPEN CUR END-EXEC
           END-IF.
```

此过程的对应 CREATE PROCEDURE 语句如下所示:

```
CREATE PROCEDURE NEWSALE ( IN SALESPERSON CHAR(15),
                           IN SALESQTY INT,
                           OUT SALESSUM INT)
  RESULT SETS 1
  EXTERNAL NAME 'NEWSALE!NEWSALE'
  FENCED
  LANGUAGE COBOL
  PARAMETER STYLE SQL
  MODIFIES SQL DATA
```

前面的语句假设 COBOL 函数存在于库 NEWSALE。

注: 如果在 Windows 操作系统上注册 COBOL 过程，那么在 CREATE 语句的 EXTERNAL NAME 子句中标识存储过程时，请采取下列预防措施。如果使用绝对路径标识来标识过程体，那么必须附加 .dll 扩展名。例如:

```
CREATE PROCEDURE NEWSALE ( IN SALESPERSON CHAR(15),
                           IN SALESQTY INT,
                           OUT SALESSUM INT)
  RESULT SETS 1
  EXTERNAL NAME 'NEWSALE!NEWSALE'
  FENCED
```

```
LANGUAGE COBOL
PARAMETER STYLE SQL
MODIFIES SQL DATA
EXTERNAL NAME 'd:\mylib\NEWSALE.dll'
```

对使用 COBOL 开发外部过程的支持

要开发 COBOL 外部过程，您必须使用支持的 COBOL 开发软件。

所有支持使用 COBOL 进行数据库应用程序开发的开发软件，也可以用来开发 COBOL 外部过程。

COBOL 嵌入式 SQL 应用程序中的受支持 SQL 数据类型

某些预定义的 COBOL 数据类型与 DB2 数据库列类型相对应。只有这些 COBOL 数据类型可以被声明为主变量。

下表列示每种列类型的等同 COBOL 类型。当预编译器找到主变量声明时，它确定适当的 SQL 类型值。数据库管理器使用此值对它自己与应用程序之间交换的数据进行转换。

并非主变量的每种可能数据描述都能被识别。COBOL 数据项必须与下表中描述的数据项一致。如果使用其他数据项，那么可能会发生错误。

表 17. 映射到 COBOL 声明的 SQL 数据类型

SQL 列类型[1]	COBOL 数据类型	SQL 列类型描述
SMALLINT （500 或 501）	01 名称 PIC S9(4) COMP-5.	16 位带符号整数
INTEGER （496 或 497）	01 名称 PIC S9(9) COMP-5.	32 位带符号整数
BIGINT （492 或 493）	01 名称 PIC S9(18) COMP-5.	64 位带符号整数
DECIMAL(*p*,*s*) （484 或 485）	01 名称 PIC S9(*m*)V9(*n*) COMP-3.	压缩十进制
REAL[2] （480 或 481）	01 名称 USAGE IS COMP-1.	单精度浮点数
DOUBLE[3] （480 或 481）	01 名称 USAGE IS COMP-2.	双精度浮点数
CHAR(*n*) （452 或 453）	01 名称 PIC X(*n*).	定长字符串

表 17. 映射到 COBOL 声明的 SQL 数据类型 (续)

SQL 列类型[1]	COBOL 数据类型	SQL 列类型描述
VARCHAR(*n*) （448 或 449）	01 名称. 49 长度 PIC S9(4) COMP-5. 49 名称 PIC X(*n*). 1<=*n*<=32 672	变长字符串
LONG VARCHAR[6] （456 或 457）	01 名称. 49 长度 PIC S9(4) COMP-5. 49 数据 PIC X(*n*). 32 673<=*n*<=32 700	可变长度长字符串
CLOB(*n*) （408 或 409）	01 MY-CLOB USAGE IS SQL TYPE IS CLOB(n). 1<=*n*<=2 147 483 647	大对象变长字符串
CLOB 定位器变量[4] （964 或 965）	01 MY-CLOB-LOCATOR USAGE IS SQL TYPE IS CLOB-LOCATOR.	标识驻留在服务器上的 CLOB 实体
CLOB 文件引用变量[4] （920 或 921）	01 MY-CLOB-FILE USAGE IS SQL TYPE IS CLOB-FILE.	包含 CLOB 数据的文件的描述符
BLOB(*n*) （404 或 405）	01 MY-BLOB USAGE IS SQL TYPE IS BLOB(n). 1<=*n*<=2 147 483 647	大对象变长二进制字符串
BLOB 定位器变量[4] （960 或 961）	01 MY-BLOB-LOCATOR USAGE IS SQL TYPE IS BLOB-LOCATOR.	标识驻留在服务器上的 BLOB 实体
BLOB 文件引用变量[4] （916 或 917）	01 MY-BLOB-FILE USAGE IS SQL TYPE IS BLOB-FILE.	包含 BLOB 数据的文件的描述符
DATE （384 或 385）	01 标识 PIC X(10).	10 字节字符串
TIME （388 或 389）	01 标识 PIC X(8).	8 字节字符串
TIMESTAMP(*p*) （392 或 393）	01 标识 PIC X(*p*+*20*). 0<=*p*<=12	19 到 32 字节字符串 *p* 为 0 时，可以使用 19 个字节的字符串。
XML[5] （988 或 989）	01 名称 USAGE IS SQL TYPE IS XML AS CLOB (大小).	XML 值

只有在 DBCS 环境中，下列数据类型才可用。

表 18. 映射到 COBOL 声明的 SQL 数据类型

SQL 列类型[1]	COBOL 数据类型	SQL 列类型描述
GRAPHIC(*n*) （468 或 469）	01 名称 PIC G(*n*) DISPLAY-1.	双字节固定长度字符串
VARGRAPHIC(*n*) （464 或 465）	01 名称. 49 长度 PIC S9(4) COMP-5. 49 名称 PIC G(*n*) DISPLAY-1. 1<=*n*<=16 336	带有 2 字节字符串长度指示符的可变长度双字节字符串
LONG VARGRAPHIC[6] （472 或 473）	01 名称. 49 长度 PIC S9(4) COMP-5. 49 名称 PIC G(*n*) DISPLAY-1. 16 337<=*n*<=16 350	带有 2 字节字符串长度指示符的可变长度双字节字符串
DBCLOB(*n*) （412 或 413）	01 MY-DBCLOB USAGE IS SQL TYPE IS DBCLOB (n). 1<=*n*<=1 073 741 823	带有 4 字节字符串长度指示符的大对象可变长度双字节字符串
DBCLOB 定位器变量[4] （968 或 969）	01 MY-DBCLOB-LOCATOR USAGE IS SQL TYPE IS DBCLOB-LOCATOR.	标识驻留在服务器上的 DBCLOB 实体
DBCLOB 文件引用变量[4] （924 或 925）	01 MY-DBCLOB-FILE USAGE IS SQL TYPE IS DBCLOB-FILE.	包含 DBCLOB 数据的文件的描述符

注:

1. **SQL 列类型**下的第一个数字表示未提供指示符变量，第二个数字表示提供了指示符变量。指示符变量用于指示 NULL 值或者用于存放已截断的字符串的长度。这些是 SQLDA 的 SQLTYPE 字段对于这些数据类型将包含的值。
2. FLOAT(*n*)（其中，0 < *n* < 25）是 REAL 的同义词。在 SQLDA 中，REAL 与 DOUBLE 之间的差别是长度值（4 或 8）。
3. 下列 SQL 类型是 DOUBLE 的同义词:
   - FLOAT
   - FLOAT(*n*)（其中，24 < *n* < 54）是 DOUBLE 的同义词。
   - DOUBLE PRECISION
4. 这不是列类型，而是主变量类型。
5. 只有 DESCRIBE 请求才会返回 SQL_TYP_XML/SQL_TYP_NXML 值。应用程序不能直接使用此值将应用程序资源与 XML 值绑定。
6. 建议您不要使用 LONG VARCHAR 和 LONG VARGRAPHIC 数据类型，在将来的发行版中，可能会除去这些数据类型。请改为选择 CLOB 或 DBCLOB 数据类型。

用于所支持 COBOL 数据类型的规则列表是:

- 在列示了 PIC S9 和 COMP-3/COMP-5 的位置，它们是必需的。
- 对于除 VARCHAR、LONG VARCHAR、VARGRAPHIC、LONG VARGRAPHIC 以及所有 LOB 变量类型以外的所有列类型，可以使用级别号 77 来代替 01。

- 为 DECIMAL(p,s) 列类型声明主变量时，请使用下列规则。请参阅以下样本:

  ```
  01 标识 PIC S9(m)V9(n) COMP-3
  ```

  – 使用 V 来指示小数点。
  – *n* 和 *m* 的值必须大于或等于 1。
  – *n* + *m* 的值不能超过 31。
  – *s* 的值等于 *n* 的值。
  – *p* 的值等于 *n* + *m* 的值。
  – 重复系数 *(n)* 和 *(m)* 是可选的。下列示例全部有效:

  ```
  01 标识 PIC S9(3)V COMP-3
  01 标识 PIC SV9(3) COMP-3
  01 标识 PIC S9V COMP-3
  01 标识 PIC SV9 COMP-3
  ```

  – 可以使用 PACKED-DECIMAL 来代替 COMP-3。
- COBOL 预编译器不支持数组。

构建 COBOL 例程

COBOL 例程的编译和链接选项

AIX IBM COBOL 例程的编译和链接选项

在 AIX 上使用 IBM COBOL for AIX 编译器来构建 COBOL 例程（存储过程）时，使用 DB2 中提供的编译和链接选项，如 `bldrtn` 构建脚本中所示。

bldrtn 的编译和链接选项

编译选项:

cob2 IBM COBOL for AIX 编译器。

-qpgmname\(mixed\)
指示编译器允许调用具有混合大小写名称的库入口点。

-qlib 指示编译器处理 COPY 语句。

-c 只执行编译；不进行链接。编译和链接是不同的步骤。

-I$DB2PATH/include/cobol_a
指定 DB2 包含文件的位置。例如: `$HOME/sqllib/include/cobol_a`。

链接选项:

cob2 使用编译器进行链接编辑。

-o $1 指定输出作为共享库文件。

$1.o 指定存储过程对象文件。

checkerr.o
包括用于执行错误检查的实用程序对象文件。

-bnoentry
不指定共享库的缺省入口点。

-bE:$1.exp
指定导出文件。导出文件包含存储过程的列表。

-L$DB2PATH/$LIB
指定 DB2 运行时共享库的位置。例如: $HOME/sqllib/lib32。

-ldb2 与数据库管理器库链接。

请参阅编译器文档，以了解其他编译器选项。

AIX Micro Focus COBOL 例程的编译和链接选项

在 AIX 上使用 Micro Focus COBOL 编译器来构建 COBOL 例程（存储过程）时，使用 DB2 中提供的编译和链接选项，如 bldrtn 构建脚本中所示。注意，可以通过设置 COBCPY 环境变量来找到 DB2 MicroFocus COBOL 包含文件，因此不需要在编译步骤中指定 -I 标志。请参阅 bldapp 脚本以获取示例。

bldrtn 的编译和链接选项

编译选项:

cob MicroFocus COBOL 编译器。

-c 只执行编译；不进行链接。编译和链接是不同的步骤。

$EXTRA_COBOL_FLAG="-C MFSYNC"
启用 64 位支持。

-x 与 -c 选项配合使用时，编译为对象模块。

链接选项:

cob 使用编译器作为链接程序的前端。

-x 生成共享库。

-o $1 指定可执行程序。

$1.o 指定程序对象文件。

-L$DB2PATH/$LIB
指定 DB2 运行时共享库的位置。例如: $HOME/sqllib/lib32。

-ldb2 与 DB2 库链接。

-ldb2gmf
与 Micro Focus COBOL 的 DB2 异常处理程序库链接。

请参阅编译器文档，以了解其他编译器选项。

HP-UX Micro Focus COBOL 例程的编译和链接选项

在 HP-UX 上使用 Micro Focus COBOL 编译器来构建 COBOL 例程（存储过程）时，使用 DB2 中提供的编译和链接选项，如 bldrtn 构建脚本中所示。

bldrtn 的编译和链接选项

编译选项:

cob COBOL 编译器。

$EXTRA_COBOL_FLAG
如果 HP-UX 平台启用了 IA64 和 64 位支持，那么此标志包含“-C MFSYNC”。

链接选项:

-y 指定需要的输出是共享库。

-o $1 指定可执行文件。

-L$DB2PATH/$LIB
指定 DB2 运行时共享库的位置。

-ldb2 与 DB2 共享库链接。

-ldb2gmf
与 Micro Focus COBOL 的 DB2 异常处理程序库链接。

请参阅编译器文档，以了解其他编译器选项。

Solaris Micro Focus COBOL 例程的编译和链接选项

以下是在 Solaris 上使用 Micro Focus COBOL 编译器来构建 COBOL 例程（存储过程）使用的编译和链接选项，如 `bldrtn` 构建脚本中所示。

bldrtn 的编译和链接选项

编译选项:

cob COBOL 编译器。

-cx 编译为对象模块。

$EXTRA_COBOL_FLAG
对于 64 位支持，包含值"-C MFSYNC"；否则，它不包含任何值。

链接选项:

cob 使用编译器作为链接程序的前端。

-y 创建自成体系的独立共享库。

-o $1 指定可执行程序。

$1.o 指定程序对象文件。

-L$DB2PATH/$LIB
指定 DB2 运行时共享库的位置。例如: `$HOME/sqllib/lib64`。

-ldb2 与 DB2 库链接。

-ldb2gmf
与 Micro Focus COBOL 的 DB2 异常处理程序库链接。

请参阅编译器文档，以了解其他编译器选项。

Linux Micro Focus COBOL 例程的编译和链接选项

以下是在 Linux 上使用 Micro Focus COBOL 编译器来构建 COBOL 例程（存储过程）时使用的编译和链接选项，如 `bldrtn` 构建脚本中所示。

bldrtn 的编译和链接选项

编译和链接选项:

cob COBOL 编译器。

$EXTRA_COBOL_FLAG
对于 64 位支持，包含值“-C MFSYNC”；否则，它不包含任何值。

-y 指定编译为独立的可调用共享对象。

-o $1 指定可执行文件。

$1.cbl 指定源文件。

-L$DB2PATH/$LIB
指定 DB2 运行时共享库的位置。

-ldb2 与 DB2 库链接。

-ldb2gmf
与 Micro Focus COBOL 的 DB2 异常处理程序库链接。

请参阅编译器文档，以了解其他编译器选项。
注: 在编译存储过程之后，必须将其复制或链接到 `/usr/lib`。编辑 `ld.so.conf` 并不足够。

Windows IBM COBOL 例程的编译和链接选项

在 Windows 上使用 IBM VisualAge COBOL 编译器来构建 COBOL 例程（存储过程和用户定义的函数）时，使用 DB2 中提供的编译和链接选项，如 `bldrtn.bat` 批处理文件中所示。

bldrtn 的编译和链接选项

编译选项:

cob2 IBM VisualAge COBOL 编译器。

-qpgmname(mixed)
指示编译器允许调用具有混合大小写名称的库入口点。

-c 只执行编译；不进行链接。此批处理文件包含独立的编译和链接步骤。

-qlib 指示编译器处理 COPY 语句。

-I路径 指定 DB2 包含文件的位置。例如: `-I"%DB2PATH%\include\cobol_a"`。

%EXTRA_COMPFLAG%
如果将“`set IBMCOB_PRECOMP=true`”取消注释，那么将使用 IBM COBOL 预编译器来预编译嵌入式 SQL。根据输入参数的不同，将使用下列其中一种格式来调用预编译器:

-q"SQL('database sample CALL_RESOLUTION DEFERRED')"
使用缺省的 sample 数据库进行预编译并推迟调用解析。

-q"SQL('database %2 CALL_RESOLUTION DEFERRED')"
使用用户指定的数据库进行预编译并推迟调用解析。

链接选项:

ilink 使用 IBM VisualAge COBOL 链接程序。

/free 自由格式。

/nol 无徽标。

/dll 创建具有源程序名称的 DLL。

db2api.lib
与 DB2 库链接。

%1.exp 包括导出文件。

%1.obj 包括程序对象文件。

iwzrwin3.obj
包括 IBM VisualAge COBOL 所提供的对象文件。

请参阅编译器文档，以了解其他编译器选项。

Windows Micro Focus COBOL 例程的编译和链接选项

在 Windows 上使用 Micro Focus COBOL 编译器来构建 COBOL 例程（存储过程和用户定义的函数）时，使用 DB2 中提供的编译和链接选项，如 `bldrtn.bat` 批处理文件中所示。

bldrtn 的编译和链接选项

编译选项:

cobol Micro Focus COBOL 编译器。

/case 禁止将外部符号转换为大写。

链接选项:

cbllink
使用 Micro Focus COBOL 链接程序进行链接编辑。

/d 创建 .dll 文件。

db2api.lib
与 DB2 API 库链接。

请参阅编译器文档，以了解其他编译器选项。

在 AIX 上构建 IBM COBOL 例程

关于此任务

DB2 提供了用于编译和链接 COBOL 嵌入式 SQL 和 DB2 管理 API 程序的构建脚本。这些脚本与可以使用这些文件构建的样本程序一起放在 `sqllib/samples/cobol` 目录中。

`sqllib/samples/cobol` 中的脚本 `bldrtn` 包含用于构建例程（存储过程）的命令。此脚本将例程编译到可以由客户机应用程序调用的共享库中。

第一个参数 `$1` 指定源文件的名称。第二个参数 `$2` 指定所要连接的数据库的名称。由于必须在数据库所在的实例中构建共享库，因此没有用户标识和密码参数。

只有第一个参数（源文件名）是必需的。此脚本使用源文件名 $1 作为共享库的名称。数据库名是可选的。如果未提供数据库名，那么程序将使用缺省的 sample 数据库。

要在连接到 sample 数据库的情况下根据源文件 outsrv.sqb 来构建样本程序 outsrv，请输入:

```
bldrtn outsrv
```

如果已连接到另一个数据库，那么还需输入数据库名:

```
bldrtn outsrv 数据库
```

此脚本文件将把共享库复制到服务器上的路径 sqllib/function 中。

构建例程共享库 outsrv 完成后，您就可以构建需要调用该库中的例程的客户机应用程序 outcli。您可以使用脚本文件 bldapp 来构建 outcli。

要调用该例程，请通过输入以下命令来运行样本客户机应用程序:

```
outcli database userid password
```

其中:

数据库 是所要连接的数据库的名称。名称可以是 sample 或其远程别名，也可以是另外某个名称。

用户标识
是有效的用户标识。

密码 是用户标识的有效密码。

此客户机应用程序将访问共享库 outsrv 并对服务器数据库执行同名的例程，然后将输出返回给客户机应用程序。

构建 UNIX Micro Focus COBOL 例程

DB2 数据库产品提供了用于编译和链接 Micro Focus COBOL 嵌入式 SQL 和 DB2 API 程序的构建脚本。这些脚本与可以使用这些文件构建的样本程序一起放在 sqllib/samples/cobol_mf 目录中。

关于此任务

脚本 **bldrtn** 包含用于构建例程（存储过程）的命令。此脚本将例程源文件编译到可以由客户机应用程序调用的共享库中。

第一个参数 $1 指定源文件的名称。此脚本使用源文件名作为共享库的名称。第二个参数 $2 指定所要连接的数据库的名称。由于必须在数据库所在的实例中构建共享库，因此没有用户标识和密码参数。

只有第一个参数（源文件名）是必需的。数据库名是可选的。如果未提供数据库名，那么程序将使用缺省的 sample 数据库。

过程

1. 在构建 Micro Focus COBOL 例程之前，必须运行以下命令:

```
db2stop
db2set DB2LIBPATH=$LD_LIBRARY_PATH
db2set DB2ENVLIST="COBDIR LD_LIBRARY_PATH"
db2set
db2start
```

请确保 **db2stop** 停止数据库。发出最后一条 **db2set** 命令的目的是检查设置：请确保正确地设置 **DB2LIBPATH** 和 **DB2ENVLIST**。

2. 要在连接到 sample 数据库的情况下根据源文件 outsrv.sqb 来构建样本程序 outsrv，请输入：`bldrtn outsrv`。

   如果已连接到另一个数据库，那么还需输入数据库名。例如:

   `bldrtn outsrv` *数据库*

   此脚本文件将把共享库复制到服务器上的路径 `sqllib/function` 中。

   构建存储过程 outsrv 完成后，您就可以构建需要调用它的客户机应用程序 outcli。您可以使用脚本文件 **bldapp** 来构建 outcli。

3. 要调用此存储过程，请通过输入以下命令来运行样本客户机应用程序：`outcli` *数据库 用户标识 密码*

   其中:

   数据库 是所要连接的数据库的名称。名称可以是 `sample` 或其别名，也可以是另一个名称。

   用户标识
   是有效的用户标识。

   密码 是用户标识的有效密码。

   此客户机应用程序将访问共享库 `outsrv` 并执行服务器数据库中的同名存储过程函数。接着，输出将被返回到客户机应用程序。

在 Windows 上构建 IBM COBOL 例程

关于此任务

DB2 提供了用于编译和链接那些使用 IBM COBOL 编写的 DB2 API 和嵌入式 SQL 程序的构建脚本。这些脚本与可以使用这些文件构建的样本程序一起放在 `sqllib\samples\cobol` 目录中。

在 Windows 上，DB2 支持使用两款预编译器来构建 IBM COBOL 应用程序，即 DB2 预编译器和 IBM COBOL 预编译器。缺省情况下，使用 DB2 预编译器。您可以通过在所使用的批处理文件中取消注释适当的行来选择 IBM COBOL 预编译器。对 IBM COBOL 进行的预编译由编译器本身使用特定的预编译选项完成。

批处理文件 `bldrtn.bat` 包含用于构建嵌入式 SQL 例程（存储过程）的命令。这个批处理文件用于在服务器上将例程编译成 DLL。它接受 2 个参数，在批处理文件中，它们分别由变量 `%1` 和 `%2` 表示。

第一个参数 `%1` 指定源文件的名称。此批处理文件将源文件名 `%1` 用作 DLL 名称。第二个参数 `%2` 指定所要连接的数据库的名称。由于必须在数据库所在的实例中构建存储过程，因此没有用户标识和密码参数。

只有第一个参数（源文件名）是必需的。数据库名是可选的。如果未提供数据库名，那么程序将使用缺省的 sample 数据库。

如果您使用缺省的 DB2 预编译器，那么 bldrtn.bat 将把参数传递给预编译和绑定文件 embprep.bat。

如果您使用 IBM COBOL 预编译器，那么 bldrtn.bat 将把 .sqb 源文件复制到 .cbl 源文件。编译器使用特定的预编译选项对 .cbl 源文件执行预编译。

要在连接到 sample 数据库的情况下根据源文件 outsrv.sqb 来构建样本程序 outsrv，请输入：

```
bldrtn outsrv
```

如果已连接到另一个数据库，那么还需输入数据库名：

```
bldrtn outsrv 数据库
```

此批处理文件将把 DLL 复制到服务器上的路径 sqllib\function 中。

构建 DLL outsrv 完成后，您就可以构建需要调用该 DLL 中的例程（与该 DLL 同名）的客户机应用程序 outcli。可以使用批处理文件 bldapp.bat 来构建 outcli。

要调用 outsrv 例程，请通过输入以下命令来运行样本客户机应用程序：

```
outcli database userid password
```

其中：

数据库 是所要连接的数据库的名称。名称可以是 sample 或其远程别名，也可以是另外某个名称。

用户标识
是有效的用户标识。

密码 是用户标识的有效密码。

此客户机应用程序将访问 DLL outsrv 并对服务器数据库执行同名的例程，然后将输出返回给客户机应用程序。

在 Windows 上构建 Micro Focus COBOL 例程

关于此任务

DB2 提供了用于编译和链接那些使用 Micro Focus COBOL 编写的 DB2 API 和嵌入式 SQL 程序的构建脚本。这些脚本与可以使用这些文件构建的样本程序一起放在 sqllib\samples\cobol_mf 目录中。

批处理文件 bldrtn.bat 包含用于构建嵌入式 SQL 例程（存储过程）的命令。这个批处理文件用于在服务器上将例程编译成 DLL。此批处理文件接受 2 个参数，在批处理文件中，它们分别由变量 %1 和 %2 表示。

第一个参数 %1 指定源文件的名称。此批处理文件将源文件名 %1 用作 DLL 名称。第二个参数 %2 指定所要连接的数据库的名称。由于必须在数据库所在的实例中构建存储过程，因此没有用户标识和密码参数。

只有第一个参数（源文件名）是必需的。数据库名是可选的。如果未提供数据库名，那么程序将使用缺省的 sample 数据库。

要在连接到 sample 数据库的情况下根据源文件 outsrv.sqb 来构建样本程序 outsrv，请输入：

```
bldrtn outsrv
```

如果已连接到另一个数据库，那么还需输入数据库名：

```
bldrtn outsrv 数据库
```

此脚本文件将把 DLL 复制到服务器上的路径 sqllib/function 中。

构建 DLL outsrv 完成后，您就可以构建需要调用该 DLL 中的例程（与该 DLL 同名）的客户机应用程序 outcli。可以使用批处理文件 bldapp.bat 来构建 outcli。

要调用 outsrv 例程，请通过输入以下命令来运行样本客户机应用程序：

```
outcli database userid password
```

其中：

数据库 是所要连接的数据库的名称。名称可以是 sample 或其别名，也可以是另一个名称。

用户标识
是有效的用户标识。

密码 是用户标识的有效密码。

此客户机应用程序将访问 DLL outsrv 并对服务器数据库执行同名的例程。接着，输出将被返回到客户机应用程序。

第 7 章 Java 例程

Java 例程是具有 Java 编程语言实现的外部例程。

通过执行 CREATE PROCEDURE 或 CREATE FUNCTION 语句在数据库中创建 Java 例程。此语句必须使用 LANGUAGE JAVA 子句来指示使用 Java 实现此例程。此语句也必须使用 EXTERNAL 子句来指定实现此语句的 Java 类。

可以使用 Java 来创建外部过程、函数以及方法。

Java 例程可以执行 SQL 语句。

下列术语在 Java 例程的上下文中很重要:

JDBC 一个应用程序编程接口，用于访问数据库中支持动态 SQL 执行的数据。

SQLJ 一个应用程序编程接口，用于访问数据库中支持动态和静态 SQL 执行的数据。

CREATE 语句
用来在数据库中创建例程的 SQL 语言 CREATE 语句。

例程实体源代码
包含 Java 例程实现的源代码文件。Java 例程可以使用任一 JDBC 或 SQLJ 应用程序编程接口来访问数据库。

JDBC 一种应用程序编程接口，为 Java 代码中的动态 SQL 语句执行提供支持。

SQLJ 一种应用程序编程接口，为 Java 代码中的静态 SQL 语句执行提供支持。

SDK for Java
为 Java 源代码编译提供了所需的 Java 软件开发包。

例程类 包含已编译格式的 Java 例程源代码的 Java 源代码文件。Java 类文件可以单独存在，也可以是 JAR 文件中的其中一个 Java 类文件集合。

受支持的 Java 例程开发软件

要开发和部署使用 Java 编写的外部例程，您必须使用受支持的 Java 开发软件。

Java 例程开发所支持的最低软件开发包（SDK）版本是 IBM SDK for Java 1.4.2。但是，建议您不要使用 IBM SDK for Java 1.4.2 支持，在将来的发行版中，可能会除去此支持。

Java 例程开发所支持的最高 SDK 版本是:

- IBM SDK for Java 7

  IBM SDK for Java 7 在下列操作系统上支持存储过程和用户定义的函数:
  - AIX 5
  - 用于基于 Itanium 的系统的 HP-UX
  - 基于 x86 的 Linux
  - 基于 AMD64/EM64T 的 Linux

– 基于 POWER 的 Linux
– 基于 x86 的 Windows
– 用于 AMD64/EM64T 的基于 x64 的 Windows

建议您使用随 DB2 数据库 Linux 版、UNIX 版和 Windows 版产品一起安装的 SDK for Java，但可以指定备用 SDK for Java。如果指定备用的 SDK for Java，那么它的位宽度必须与 DB2 实例相同。

使用 Java 进行 DB2 数据库应用程序开发所支持的所有其他开发软件均可用于使用 Java 进行的外部例程开发。

对 Java 例程的 JDBC 和 SQLJ 应用程序编程接口支持

使用 Java 开发的外部例程可以利用下列应用程序编程接口 (API): JDBC 和 SQLJ。

从 DB2 V10 开始，唯一驱动程序是 IBM Data Server Driver for JDBC and SQLJ。此驱动程序支持 JDBC 和 SQLJ API，且可用来开发外部 Java 例程。

不论使用的 API 为何，用于实现 Java 例程的过程都相同。

SDK for Java 例程开发规范 (UNIX)

开始之前

要在 UNIX 环境中构建并运行 Java 例程代码，必须将 DB2 数据库管理器配置参数 **JDK_PATH** 设置为安装在 DB2 数据库服务器上的 SDK for Java 的路径。如果 DB2 Linux 版、UNIX 版和 Windows 版数据库产品的安装进程将安装 SDK for Java，那么缺省情况下会将 **JDK_PATH** 参数设置为该 SDK for Java 的路径:

- *$INSTDIR*/sqllib/java/jdk32，对于安装在 x86 环境的 Linux 中的 DB2 数据库服务器
- *$INSTDIR*/sqllib/java/jdk64，对于安装在所有其他环境中提供 SDK 的 DB2 数据库服务器

可以更改 **JDK_PATH** 参数值以指定计算机上所安装的另一个 SDK for Java，但是 SDK 的位宽必须与 DB2 实例的位宽相同。

- 用户必须能够访问 DB2 数据库服务器。
- 用户必须有权读取和更新数据库管理器配置文件。
- 用户必须有权在安装 DB2 实例的文件系统中安装 SDK for Java。

过程

1. 请从 DB2 命令窗口发出以下命令来检查 **JDK_PATH** 参数值:

   ```
   db2 get dbm cfg
   ```

   您可能需要将输出重定向至文件以方便查看。 **JDK_PATH** 参数值将显示在输出开头附近。
2. 如果要使用不同的 SDK for Java，请将它安装至 DB2 数据库服务器，并记下安装路径。 下一步将需要该安装路径值。

3. 要更新 **JDK_PATH** 参数值，请从 DB2 命令窗口发出以下命令，其中 *path* 是其他 SDK for Java 的安装路径:

   ```
   db2 update dbm cfg using JDK_PATH path
   ```

4. 要停止然后重新启动 DB2 实例，请从 DB2 命令窗口发出下列 DB2 命令:

   ```
   db2stop;

   db2start;
   ```

5. 要验证是否正确设置了 **JDK_PATH** 参数值，请从 DB2 命令窗口发出以下命令:

   ```
   db2 get dbm cfg
   ```

下一步做什么

完成这些步骤后，会将指定的 SDK for Java 用于构建和运行 Java 例程。会自动设置环境中的 **CLASSPATH**、**PATH** 和 **LIBPATH** 环境变量。

Java 例程驱动程序的规范

Java 例程开发和调用要求指定 JDBC 或 SQLJ 驱动程序。

Java 例程使用 IBM 数据服务器 JDBC 和 SQLJ 驱动程序 V4.0。

IBM Data Server Driver for JDBC and SQLJ V4.0 的 `db2jcc4.jar` 包括一些 JDBC 版本 4.0 功能。DB2 版本 9.5 及更高版本支持该驱动程序。

缺省情况下，DB2 数据库系统使用 IBM Data Server Driver for JDBC and SQLJ。如果 Java 例程包含以下各项，那么需要此驱动程序:

- 数据类型为 XML 的参数
- 数据类型为 XML 的变量
- XML 数据引用
- XML 函数引用
- 任何其他本机 XML 功能

用于开发 Java（JDBC 和 SQLJ）例程的工具

IBM Data Studio、DB2 命令行处理器和 DB2 命令窗口之类的工具可帮助您快速轻松地开发 Java 例程。

下列 DB2 数据库工具针对开发、调试以及部署 Java 例程提供图形用户界面支持:

- IBM Data Studio

也可以使用下列命令行界面来开发、调试以及部署 Java 例程:

- DB2 命令行处理器
- DB2 命令窗口

其他 IBM 软件产品提供用于开发 Java 例程的图形工具，其中包括:

- IBM Optim Development Studio
- IBM Rational Application Developer
- 分布式统一调试器

设计 Java 例程

设计 Java 例程是一项应该在创建 Java 例程之前执行的任务。设计 Java 例程与设计使用其他编程语言来实现的外部例程以及设计 Java 数据库应用程序相关。

开始之前

嵌入式 SQL 应用程序开发的知识和经验以及对外部例程的一般了解。下列主题可以向您提供一些必备信息。

有关外部例程的功能和使用的更多信息，请参阅:

- 第 18 页的『外部例程实现』

有关如何使用 JDBC 或 SQLJ 应用程序编程接口来编写基本 Java 应用程序的更多信息，请参阅:

- *Developing Java Applications* 中的『简单 JDBC 应用程序示例』
- *Developing Java Applications* 中的『简单 SQLJ 应用程序示例』

关于此任务

掌握必备知识之后，设计 Java 例程的主要注意事项是了解 Java 例程的特有功能和特征:

过程

- 『Java 例程支持的 SQL 数据类型』
- 第 221 页的『Java 例程中的参数』
- 第 221 页的『参数样式 JAVA 过程』
- 第 222 页的『PARAMETER STYLE JAVA Java 函数和方法』
- 第 232 页的『从 JDBC 过程中返回结果集』
- 第 233 页的『从 SQLJ 过程中返回结果集』
- 第 235 页的『Java 例程限制』
- 第 55 页的『Java 表函数执行模型』

下一步做什么

在了解 Java 特征之后，您不妨参阅:

- 第 238 页的『从命令行创建 Java 例程』

Java 例程支持的 SQL 数据类型

必须根据 JDBC 和 SQLJ 应用程序编程接口规范，在 Java 源代码中使用 Java 编程语言数据类型来存储 SQL 数据类型值。DB2 JDBC 版和 SQLJ 版驱动程序会根据特定的数据类型映射，转换在 Java 源代码和 DB2 数据库之间交换的数据。

数据映射对下列各项有效:

- Java 数据库应用程序
- 定义为 PARAMETER STYLE JAVA 且使用 PARAMETER STYLE JAVA 来实现的 Java 例程。

映射至 SQL 数据类型的 Java 数据类型如下所示:

表 19. 映射至 Java 声明的 SQL 数据类型

SQL 列类型	**Java 数据类型**	**SQL 列类型描述**
SMALLINT （500 或 501）	short, boolean	16 位带符号整数
INTEGER （496 或 497）	int	32 位带符号整数
BIGINT[1]（492 或 493）	long	64 位带符号整数
REAL （480 或 481）	float	单精度浮点数
DOUBLE （480 或 481）	double	双精度浮点数
DECIMAL(*p*,*s*) （484 或 485）	java.math.BigDecimal	压缩十进制
CHAR(*n*) （452 或 453）	java.lang.String	长度为 *n* 的固定长度字符串，其中 *n* 为 1 到 254
CHAR(*n*) FOR BIT DATA	byte[]	长度为 *n* 的固定长度字符串，其中 *n* 为 1 到 254
VARCHAR(*n*) （448 或 449）	java.lang.String	变长字符串
VARCHAR(*n*) FOR BIT DATA	byte[]	变长字符串
LONG VARCHAR （456 或 457）	java.lang.String	可变长度长字符串
LONG VARCHAR FOR BIT DATA	byte[]	可变长度长字符串
BLOB(*n*) （404 或 405）	java.sql.Blob	大对象变长二进制字符串
CLOB(*n*) （408 或 409）	java.sql.Clob	大对象变长字符串
DBCLOB(*n*) （412 或 413）	java.sql.Clob	大对象可变长度双字节字符串
DATE （384 或 385）	java.sql.Date	10 字节字符串
TIME （388 或 389）	java.sql.Time	8 字节字符串
TIMESTAMP （392 或 393）	java.sql.Timestamp	字符串的长度可达 19-32 个字节，取决于所指定的小数秒数。 （可选）可以指定 TIMESTAMP 数据类型的小数秒部分，以使时间戳记精度为 0-12 位。 将时间戳记值指定给具有不同小数秒数的时间戳记变量时，会截断此值或使用零来填充此值，以匹配变量的格式。

表 19. 映射至 Java 声明的 SQL 数据类型 (续)

SQL 列类型	Java 数据类型	SQL 列类型描述
GRAPHIC(*n*) （468 或 469）	java.lang.String	双字节固定长度字符串
VARGRAPHIC(*n*) （464 或 465）	java.lang.String	并非以 null 结束的可变长度双字节字符串（带有 2 字节字符串长度指示符）
LONGVARGRAPHIC （472 或 473）	java.lang.String	并非以 null 结束的可变长度双字节字符串（带有 2 字节字符串长度指示符）
XML(*n*) （408 或 409）	java.sql.Clob	以表示 CLOB 数据类型的方法来表示 XML 数据类型；即，表示为大对象变长字符串
ARRAY	java.sql.Array	SQL 数据的数组。

注:

1. 对于从 DB2 Universal Database V8.1 客户机连接至 DB2 Universal Database V7.1（或 V7.2）服务器的 Java 应用程序，请注意下列事项: 使用 getObject() 方法来检索 BIGINT 值时，会返回 java.math.BigDecimal 对象。
2. 会将 SQL 数组数据类型的参数映射至类 com.ibm.db2.ARRAY。
3. ARRAY 数据类型不支持 LONG VARCHAR、LONG VARGRAPHIC、XML、REFERENCE、UDT 以及 ARRAY。

SQLJ 例程中的连接上下文

在 DB2 Universal Database V8 中推出多线程例程后，SQLJ 例程必须避免使用缺省连接上下文。即，每个 SQL 语句都必须显式地指示 ConnectionContext 对象，并且必须在 Java 方法中显式地实例化上下文。

例如，在 DB2 数据库产品的先前发行版中，可以按如下所示编写 SQLJ 例程:

```
class myClass
{
  public static void myRoutine( short myInput )
  {
    DefaultContext ctx = DefaultContext.getDefaultContext();
    #sql { some SQL statement };
  }
}
```

使用缺省上下文会导致多线程环境中的所有线程使用同一连接上下文，而这又会导致意外失败。

必须按如下所示更改此 SQLJ 例程:

```
#context MyContext;

class myClass
{
  public static void myRoutine( short myInput )
  {
    MyContext ctx = new MyContext( "jdbc:default:connection", false );
    #sql [ctx] { some SQL statement };
    ctx.close();
  }
}
```

这样，每次调用例程时都会创建单独的唯一 ConnectionContext（以及底层 JDBC 连接），从而避免并发线程的意外干扰。

Java 例程中的参数

Java 例程中的参数声明必须符合其中一种支持参数样式的要求。

Java 例程支持下列两种参数样式:

- PARAMETER STYLE JAVA
- PARAMETER STYLE DB2GENERAL

强烈建议您在例程 CREATE 语句中指定 PARAMETER STYLE JAVA 子句。借助 PARAMETER STYLE JAVA，例程将使用符合 Java 语言和 SQLJ 例程规范的参数传递约定。

某些 Java 例程无法使用 PARAMETER STYLE JAVA 来实现，或无法与 PARAMETER STYLE JAVA 一起使用。这些例程如下所示:

- 表函数
- 函数中的暂存区
- 访问函数中的 DBINFO 结构
- 能够对函数或方法发出 FINAL CALL（以及单独的第一次调用）

如果您需要实现这些功能，那么可以使用 C 来实现例程，或使用 Java 通过参数样式 DB2GENERAL 来实现例程。

除这些特定情况之外，您还应该始终使用 PARAMETER STYLE JAVA 来创建和实现 Java 例程。

参数样式 JAVA 过程

Java 过程实现的建议参数样式是 PARAMETER STYLE JAVA。

PARAMETER STYLE JAVA 存储过程的特征符遵循下列格式:

```
public static void 方法名 ( SQL-arguments, ResultSet[] 结果集数组 )
                                  throws SQLException
```

方法名 方法的名称。注册例程期间，在 CREATE PROCEDURE 语句的 EXTERNAL NAME 子句中使用类名来指定此值。

SQL-arguments

对应于 CREATE PROCEDURE 语句中的输入参数列表。会将 OUT 或 INOUT 方式参数作为单一元素数组来传递。对于 CREATE PROCEDURE 语句的 DYNAMIC RESULT SETS 子句中所指定的每个结果集，会将类型为 ResultSet 的单一元素数组附加至参数列表。

结果集数组

ResultSet 对象数组的名称。对于 CREATE PROCEDURE 语句的 DYNAMIC RESULT SETS 参数中所声明的每个结果集，必须在 Java 方法特征符中声明类型为 ResultSet[] 的参数。

下列是 Java 存储过程示例，此存储过程可接受输入参数，然后返回一个输出参数和一个结果集:

```
public static void javastp( int inparm,
                            int[] outparm,
                            ResultSet[] rs
                          )
                    throws SQLException
{
  Connection con = DriverManager.getConnection( "jdbc:default:connection" );
  PreparedStatement stmt = null;
  String sql = "SELECT value FROM table01 WHERE index = ?";

  //Prepare the query with the value of index
  stmt = con.prepareStatement( sql );
  stmt.setInt( 1, inparm );

  //Execute query and set output parm
  rs[0] = stmt.executeQuery();
  outparm[0] = inparm + 1;

  //Close open resources
  if (stmt != null) stmt.close();
  if (con != null) con.close();

  return;
}
```

此存储过程的对应 CREATE PROCEDURE 语句如下所示:

```
CREATE PROCEDURE javaproc( IN in1 INT, OUT out1 INT )
  LANGUAGE java
  PARAMETER STYLE java
  DYNAMIC RESULT SETS 1
  FENCED THREADSAFE
  EXTERNAL NAME 'myjar:stpclass.javastp'
```

前面的语句假设该方法位于类 stpclass 中，而这个类位于已编目到数据库的 JAR 文件（Jar ID 为 myjar）中

注:

1. PARAMETER STYLE JAVA 例程使用异常将错误数据传递回至调用程序。有关完整信息（其中包括异常调用堆栈），请参阅管理通知日志。除此详细信息之外，未提供任何其他有关调用 PARAMETER STYLE JAVA 例程的特殊注意事项。
2. Java 例程不支持 JNI 调用。但是，可通过嵌套 C 例程的调用，从 Java 例程调用 C 功能。这涉及将期望的 C 功能移至例程，注册该功能，然后从 Java 例程中调用该功能。

PARAMETER STYLE JAVA Java 函数和方法

Java 函数和方法的建议参数样式是 PARAMETER STYLE JAVA。

PARAMETER STYLE JAVA 函数和方法的特征符遵循下列格式:

```
public static return-type 方法名 ( SQL-arguments ) throws SQLException
```

return-type
标量例程所要返回的值的数据类型。在例程中，通过 return 语句将返回值传递回至调用程序。

方法名 方法的名称。注册例程期间，在例程 CREATE 语句的 EXTERNAL NAME 子句中使用类名来指定此值。

SQL-arguments
对应于例程 CREATE 语句中的输入参数列表。

下列是 Java 函数示例，此函数返回其两个输入自变量之积:

```
public static double product( double in1, double in2 ) throws SQLException
{
  return in1 * in2;
}
```

此标量函数的对应 CREATE FUNCTION 语句如下所示:

```
CREATE FUNCTION product( in1 DOUBLE, in2 DOUBLE )
  RETURNS DOUBLE
  LANGUAGE java
  PARAMETER STYLE java
  NO SQL
  FENCED THREADSAFE
  DETERMINISTIC
  RETURNS NULL ON NULL INPUT
  NO EXTERNAL ACTION
  EXTERNAL NAME 'myjar:udfclass.product'
```

前面的语句假设该方法位于类 udfclass 中，而这个类位于已安装至数据库服务器的 JAR 文件（Jar ID 为 myjar）中。可以使用 INSTALL_JAR 内置过程将 JAR 文件安装至数据库服务器。

DB2GENERAL 例程

使用 Java 来编写 PARAMETER STYLE DB2GENERAL 例程。创建 DB2GENERAL 例程与使用其他支持的编程语言来创建例程非常相似。在创建并注册这些例程后，您可以从使用任何语言编写的程序调用这些例程。通常，您可以从存储过程调用 JDBC API，但无法从 UDF 调用 JDBC API。

使用 Java 开发例程时，强烈建议您在 CREATE 语句中使用 PARAMETER STYLE JAVA 子句来注册这些例程。仍可以使用 PARAMETER STYLE DB2GENERAL 在 Java 例程中实现下列功能:

- 表函数
- 暂存区
- 访问 DBINFO 结构
- 能够对函数或方法发出 FINAL CALL（以及单独的第一次调用）

如果您的 PARAMETER STYLE DB2GENERAL 例程未使用上述任何功能，那么建议您将它们移植到 PARAMETER STYLE JAVA。

DB2GENERAL UDF: 和在其他语言中一样，您可以在 Java 中创建和使用 UDF（与 C UDF 相比，只有些许差别）。在编写 UDF 之后，您可以在数据库中注册 UDF。然后，您可以在应用程序中引用该 UDF。

通常，如果您声明一个 UDF，该 UDF 接受 SQL 类型 *t1*、*t2* 和 *t3* 的自变量，返回类型 *t4*，那么会将该 UDF 作为具有预期 Java 特征符的 Java 方法来调用:

```
public void 名称 (T1 a, T2 b, T3 c, T4 d)  {.....}
```

其中:

- *名称* 是 Java 方法名

- *T1* 至 *T4* 是对应于 SQL 类型 *t1* 至 *t4* 的 Java 类型。
- *a*、*b* 以及 *c* 是输入自变量的变量名称。
- *d* 是表示输出自变量的变量名称。

例如，如果 UDF sample!test3 返回 INTEGER 且接受类型为 CHAR(5)、BLOB (10K) 以及 DATE 的自变量，那么 DB2 数据库系统期望该 UDF 的 Java 实现具有下列特征符:

```
import COM.ibm.db2.app.*;
public class sample extends UDF {
   public void test3(String arg1, Blob arg2, String arg3,
                     int result)  {... }
}
```

实现表函数的 Java 例程需要较多的自变量。除表示输入的变量之外，还会针对结果行中的每个列显示一个附加变量。例如，可以将表函数声明为:

```
public void test4(String arg1, int result1,
                  Blob result2, String result3);
```

由未初始化的 Java 变量表示 SQL NULL 值。如果这些变量属于基本类型，那么其值为零；如果这些变量属于对象类型，那么根据 Java 规则，其值为 Java null。要区分 SQL NULL 和一般零，您可以针对任何输入自变量调用函数 isNull:

```
{ ....
   if (isNull(1)) { /* argument #1 was a SQL NULL */ }
   else           { /* not NULL */ }
}
```

在此示例中，自变量数从 1 开始。isNull() 函数，类似于其他后续函数，都继承自 COM.ibm.db2.app.UDF 类。

要返回标量或表 UDF 的结果，请在 UDF 中使用 set() 方法，如下所示:

```
{ ....
   set(2, value);
}
```

其中，“2”是输出自变量的下标，而 value 是兼容类型的字面值或变量。自变量数是所选输出的自变量列表中的下标。在此部分的第一个示例中，int result 变量的下标是 4; 在第二个示例中，result1 至 result3 具有下标 2 至 4。

类似于 UDF 和存储过程中所使用的 C 模块，您不能在 Java 例程中使用 Java 标准 I/O 流（System.in、System.out 以及 System.err）。

请记住，您用来实现例程的所有 Java 类文件（或者是包含这些类的 JAR）都必须位于 sqllib/function 目录或者位于在数据库管理器的 **CLASSPATH** 中指定的目录中。

通常，DB2 数据库系统会多次调用 UDF（针对查询中输入或结果集的每行调用一次）。如果已在 UDF 的 CREATE FUNCTION 语句中指定 SCRATCHPAD，那么 DB2 数据库系统可以识别两次连续调用 UDF 之间所需的某种“连续性”，因此不会针对每次调用实例化实现 Java 类，但一般说来，会针对每个语句的每个 UDF 引用实例化一次。通常，它会在第一个调用之前实例化，并且随后加以使用，但是对于表函数而言，可以经常实例化。但是，如果对 UDF（标量或表函数）指定 NO SCRATCHPAD，那么可以针对 UDF 的每个调用实例化一个全新实例。

可以使用暂存区在 UDF 调用之间保存信息。虽然 Java 和 OLE UDF 可以使用实例变量或设置暂存区以在两次调用之间实现连续性，但是 C 和 C++ UDF 必须使用暂存区。Java UDF 使用 COM.ibm.db2.app.UDF 中提供的 getScratchPad() 和 setScratchPad() 方法来访问暂存区。

对于使用暂存区的 Java 表函数，通过在 CREATE FUNCTION 语句上使用 FINAL CALL 或 NO FINAL CALL 选项来控制何时获取新的暂存区实例。

通过暂存区来实现两次 UDF 调用之间的连续性的能力，是由 CREATE FUNCTION 的 SCRATCHPAD 和 NO SCRATCHPAD 选项所控制，而不管是使用 DB2 暂存区还是实例变量。

对于标量函数，您对整个语句使用同一实例。

请注意，会单独处理对查询中 Java UDF 的每个引用，即使多次引用同一 UDF 亦如此。对于 OLE、C 以及 C++ UDF，情况相同。在查询末尾，如果您对标量函数指定 FINAL CALL 选项，那么将调用对象的 close() 方法。对于表函数，会始终根据此子节后面的子节中所作的指示来调用 close() 方法。如果您未针对 UDF 类定义 close() 方法，那么存根函数会接管控制权且事件会被忽略。

如果在 CREATE FUNCTION 语句中针对 Java UDF 指定 ALLOW PARALLEL 子句，那么 DB2 数据库系统可能选择 UDF 并行求值。如果发生这种情况，那么可能会在不同分区上创建若干单值 Java 对象。每个对象会接收一个行子集。

和其他 UDF 一样，Java UDF 可以是 FENCED 或 NOT FENCED UDF。NOT FENCED UDF 在数据库引擎的地址空间中运行；FENCED UDF 在单独的进程中运行。虽然 Java UDF 不会意外地毁坏其嵌入进程的地址空间，但它们可能会终止进程或降低进程的运行速度。因此，在调试使用 Java 编写的 UDF 时，您应该将它们作为 FENCED UDF 来运行。

从 V10.1 开始，可定义外部 Java 表函数，该表函数可根据函数的输入自变量来输出不同模式的表。它又称为通用表函数。要定义外部 Java 通用表函数，请在 CREATE FUNCTION 语句中指定 PARAMETER STYLE DB2GENERAL 并在引用该外部通用表函数时声明输出表函数的形状。

DB2GENERAL 例程支持的 SQL 数据类型:

当您调用 PARAMETER STYLE DB2GENERAL 例程时，DB2 会自动将 SQL 类型转换为 Java 类型或反之。

Java 程序包 COM.ibm.db2.app 中提供了其中若干个类。

表 20. DB2 SQL 类型和 Java 对象

SQL 列类型	Java 数据类型
SMALLINT	short
INTEGER	int
BIGINT	long
REAL[1]	float
DOUBLE	double
DECIMAL(p,s)	java.math.BigDecimal

表 20. DB2 SQL 类型和 Java 对象 (续)

SQL 列类型	Java 数据类型
NUMERIC(p,s)	java.math.BigDecimal
CHAR(*n*)	java.lang.String
CHAR(*n*) FOR BIT DATA	COM.ibm.db2.app.Blob
VARCHAR(*n*)	java.lang.String
VARCHAR(*n*) FOR BIT DATA	COM.ibm.db2.app.Blob
LONG VARCHAR	java.lang.String
LONG VARCHAR FOR BIT DATA	COM.ibm.db2.app.Blob
GRAPHIC(*n*)	java.lang.String
VARGRAPHIC(*n*)	String
LONG VARGRAPHIC[2]	String
BLOB(*n*)[2]	COM.ibm.db2.app.Blob
CLOB(*n*)[2]	COM.ibm.db2.app.Clob
DBCLOB(*n*)[2]	COM.ibm.db2.app.Clob
DATE[3]	String
TIME[3]	String
TIMESTAMP[3]	String

注:

1. 在 SQLDA 中，REAL 与 DOUBLE 之间的差别是长度值（4 或 8）。
2. COM.ibm.db2.app 程序包中提供了 Blob 和 Clob 类。这些接口包含一些例程，用于生成 InputStream 和 OutputStream 以读/写 Blob，以及生成 Reader 和 Writer 以读/写 Clob。
3. SQL DATE、TIME 和 TIMESTAMP 值在 Java 中使用 ISO 字符串编码，如同它们在用 C 编写的 UDF 中使用 ISO 字符串编码一样。

类 COM.ibm.db2.app.Blob 和 COM.ibm.db2.app.Clob 实例表示 LOB 数据类型（BLOB、CLOB 以及 DBCLOB）。这些类提供一个有限接口来读取作为输入传递的 LOB，以及写入作为输出返回的 LOB。通过标准 Java I/O 流对象来读/写 LOB。对于 Blob 类，例程 getInputStream() 和 getOutputStream() 会返回 InputStream 或 OutputStream 对象（可通过该对象每次按一定的字节数来处理 BLOB 内容）。对于 Clob，例程 getReader() 和 getWriter() 将返回 Reader 或 Writer 对象（可通过该对象每次按一定的字符数来处理 CLOB 或 DBCLOB 内容）。

如果使用 set() 方法将此类对象作为输出来返回，那么可应用代码页转换以表示数据库代码页中的 Java Unicode 字符。

用于 DB2GENERAL 例程的 Java 类:

此接口提供下列例程来访存嵌入式应用程序上下文的 JDBC 连接:

```
public java.sql.Connection getConnection()
```

您可以使用此句柄来运行 SQL 语句。StoredProc 接口的其他方法列示在文件 sqllib/samples/java/StoredProc.java 中。

您可以将五个类/接口与 Java 存储过程或 UDF 配合使用:

- COM.ibm.db2.app.StoredProc

- COM.ibm.db2.app.UDF
- COM.ibm.db2.app.Lob
- COM.ibm.db2.app.Blob
- COM.ibm.db2.app.Clob

COM.ibm.db2.app.UDF 类支持外部 Java 通用表 UDF。

DB2GENERAL Java 类: COM.ibm.db2.app.StoredProc:

包含预期作为 PARAMETER STYLE DB2GENERAL 存储过程来调用的方法的 Java 类必须是 public 类，且必须实现此 Java 接口。

您必须按如下所示声明这样的类:

```
public class 用户-STP-类 extends COM.ibm.db2.app.StoredProc{ ... }
```

只能在当前执行存储过程的上下文中调用 COM.ibm.db2.app.StoredProc 接口的继承方法。例如，您不能在存储过程返回之后对 LOB 自变量、结果设置调用或状态设置调用执行操作。如果违反此规则，那么将抛出 Java 异常。

自变量相关调用会使用列索引来标识所引用的列。索引值起始于 1（1 表示第一个自变量）。PARAMETER STYLE DB2GENERAL 存储过程的所有自变量都被视为 INOUT，因此都是输入和输出。

数据库会捕捉从存储过程返回的任何异常，并将其返回给调用者 (SQLCODE -4302, SQLSTATE 38501)。JDBC SQLException 或 SQLWarning 会加以特殊处理，并将其自己的 SQLCODE 和 SQLSTATE 等逐字传递至调用应用程序。

下列方法与 COM.ibm.db2.app.StoredProc 类相关联:

```
public StoredProc() [default constructor]
```

数据库会在存储过程调用之前调用此构造函数。

```
public boolean isNull(int) throws Exception
```

此函数会测试具有给定索引的输入自变量是否为 SQL NULL。

```
public void set(int, short) throws Exception
public void set(int, int) throws Exception
public void set(int, double) throws Exception
public void set(int, float) throws Exception
public void set(int, java.math.BigDecimal) throws Exception
public void set(int, String) throws Exception
public void set(int, COM.ibm.db2.app.Blob) throws Exception
public void set(int, COM.ibm.db2.app.Clob) throws Exception
```

此函数会设置输出自变量（给定索引设为给定值）。索引必须引用有效的输出自变量，数据类型必须匹配，以及值必须具有可接受的长度和内容。具有 Unicode 字符的字符串必须可以在数据库代码页中进行表示。错误会导致抛出异常。

```
public java.sql.Connection getConnection() throws Exception
```

此函数返回表示调用应用程序与数据库的连接的 JDBC 对象。它类似于 C 存储过程中的 null SQLConnect() 调用。

DB2GENERAL Java 类: COM.ibm.db2.app.UDF:

如果 Java 类包含预期作为带有 DB2GENERAL 参数样式的 UDF 调用的方法，那么该类必须是公用类并且必须实现 COM.IBM.db2.app.UDF Java 接口。

您必须按如下所示声明这样的类:

```
public class user-UDF-class extends COM.ibm.db2.app.UDF{ ... }
```

只能在当前执行 UDF 的上下文中调用 COM.ibm.db2.app.UDF 接口的方法。例如，您不能在 UDF 从执行返回后对 LOB 自变量、结果或状态设置调用等使用操作。如果违反此规则，那么将抛出 Java 异常。

自变量相关调用会使用列索引来标识所设置的列。列索引起始于 1（表示第一个自变量）。输出自变量的编号大于输入自变量的编号。例如，具有三项输入的标量 UDF 会对输出使用索引值 4。

数据库会捕捉从 UDF 返回的任何异常，并将其返回给调用者 (SQLCODE -4302, SQLSTATE 38501)。

下列方法与 COM.ibm.db2.app.UDF 类相关联:

```
public UDF() [default constructor]
```

在一系列 UDF 调用开始之前，数据库会调用此构造函数。先调用此构造函数，然后再对 UDF 执行第一次调用。

```
public void close()
```

如果使用 FINAL CALL 选项创建了 UDF，那么在 UDF 求值结束时，数据库会调用此函数。它类似于 C UDF 的最终调用。对于表函数，在对 UDF 方法执行 CLOSE 调用（如果编写或缺省使用 NO FINAL CALL）或执行 FINAL 调用（如果编写 FINAL CALL）之后，会调用 close()。如果 Java UDF 类未实现此函数，那么无操作存根将处理和忽略此事件。

```
public int getCallType() throws Exception
```

表函数 UDF 方法使用 getCallType() 来找出特定调用的调用类型。它会返回如下所示的值（在 COM.ibm.db2.app.UDF 类定义中为这些值提供了符号定义）:

- -2 FIRST 调用
- -1 OPEN 调用
- 0 FETCH 调用
- 1 CLOSE 调用
- 2 FINAL 调用

```
public boolean isNull(int) throws Exception
```

此函数会测试具有给定索引的输入自变量是否为 SQL NULL。

```
public boolean needToSet(int) throws Exception
```

此函数会测试是否必须设置具有给定索引的输出自变量。如果声明了 DBINFO 的表 UDF 的调用者未使用该列，那么对于该 UDF，测试结果可能是 false。

```
public void set(int, short) throws Exception
public void set(int, int) throws Exception
public void set(int, double) throws Exception
public void set(int, float) throws Exception
public void set(int, java.math.BigDecimal) throws Exception
```

```
public void set(int, String) throws Exception
public void set(int, COM.ibm.db2.app.Blob) throws Exception
public void set(int, COM.ibm.db2.app.Clob) throws Exception
```

此函数会设置输出自变量（给定索引设为给定值）。索引必须引用有效的输出自变量，数据类型必须匹配，以及值的长度和内容必须可接受。具有 Unicode 字符的字符串必须可以在数据库代码页中进行表示。错误会导致抛出异常。

```
public void setSQLstate(String) throws Exception
```

可以从 UDF 调用此函数以设置要从此调用返回的 SQLSTATE。表 UDF 可以使用“02000”调用此函数，以指示表末尾条件。如果 SQLSTATE 无法接受此字符串，那么将抛出异常。

```
public void setSQLmessage(String) throws Exception
```

此函数类似 setSQLstate 函数。它设置 SQL 消息结果。如果无法接受此字符串（例如，字符串长度超过 70 个字符），那么将抛出异常。

```
public String getFunctionName() throws Exception
```

此函数返回执行 UDF 的名称。

```
public String getSpecificName() throws Exception
```

此函数返回执行 UDF 的特定名称。

```
public byte[] getDBinfo() throws Exception
```

此函数将执行 UDF 的原始未处理 DBINFO 结构作为字节数组来返回。您必须先使用 DBINFO 选项来声明此函数。

```
public String getDBname() throws Exception
public String getDBauthid() throws Exception
public String getDBtbschema() throws Exception
public String getDBtbname() throws Exception
public String getDBcolname() throws Exception
public String getDBver_rel() throws Exception
public String getDBplatform() throws Exception
public String getDBapplid() throws Exception
```

这些函数会返回执行 UDF 的 DBINFO 结构的相应字段值。

```
public int getDBprocid() throws Exception
```

此函数返回直接或间接调用此例程的过程的例程标识。例程标识匹配 SYSCAT.ROUTINES 中的 ROUTINEID 列，此 ROUTINEID 列可用来检索调用过程的名称。如果从应用程序调用执行例程，那么 getDBprocid() 将返回 0。

```
public int[] getDBcodepg() throws Exception
```

此函数会返回 DBINFO 结构中数据库的 SBCS、DBCS 以及复合代码页编号。返回的整数数组将下列数字用作其前三个元素。

```
public byte[] getScratchpad() throws Exception
```

此函数返回当前执行 UDF 的暂存区副本。您必须先使用 SCRATCHPAD 选项来声明该 UDF。

```
public void setScratchpad(byte[]) throws Exception
```

此函数使用给定字节数组的内容来覆盖当前执行 UDF 的暂存区。您必须先使用 SCRATCHPAD 选项来声明该 UDF。字节数组的大小必须与 getScratchpad() 返回的大小相同。

COM.ibm.db2.app.UDF 类包含以下用于在分区数据库环境使 Java UDF 顺利执行的方法:

```
public int[] getDBPartitions() throws Exception
```

此函数返回该表函数中包括的所有分区的列表。

```
public int getCurrentDBPartitionNum() throws Exception
```

此函数返回该表函数正对其执行的节点的分区号。

COM.ibm.db2.app.UDF 类包含以下用于获取创建外部通用表函数时所需的信息的方法:

```
public int getNumColumns() throws Exception
```

对于表 UDF，此函数返回输出列数。对于其他 UDF，此函数返回“1”。

```
public int getColumnType(int position) throws Exception
```

此函数返回指定输出列的数据类型。

DB2GENERAL Java 类: COM.ibm.db2.app.Lob:

此类提供实用程序例程来创建要例程内部进行计算的临时 Blob 或 Clob 对象。

下列方法与 COM.ibm.db2.app.Lob 类相关联:

```
public static Blob newBlob() throws Exception
```

此函数创建临时 Blob。如果可能，将使用 LOCATOR 来实现。

```
public static Clob newClob() throws Exception
```

此函数创建临时 Clob。如果可能，将使用 LOCATOR 来实现。

DB2GENERAL Java 类: COM.ibm.db2.app.Blob:

此类的实例是由数据库传递以将 BLOB 表示为例程输入，且可以传递回为输出。

应用程序可创建实例，但只能在执行例程的上下文中创建。在此类上下文外部使用这些对象将抛出异常。

下列方法与 COM.ibm.db2.app.Blob 类相关联:

```
public long size() throws Exception
```

此函数返回 BLOB 的长度（以字节计）。

```
public java.io.InputStream getInputStream() throws Exception
```

此函数返回新的 InputStream 以读取 BLOB 的内容。可以在该对象上执行高效的寻道/标记操作。

```
public java.io.OutputStream getOutputStream() throws Exception
```

此函数返回新的 OutputStream 以将字节附加至 BLOB。对此对象的 getInputStream() 调用所产生的所有现有 InputStream 实例可立即看到附加的字节。

DB2GENERAL Java 类: COM.ibm.db2.app.Clob:

此类的实例是由数据库传递以将 CLOB 或 DBCLOB 表示为例程输入，且可以传递回为输出。

应用程序可创建实例，但只能在执行例程的上下文中创建。在此类上下文外部使用这些对象将抛出异常。

Clob 实例将字符存储在数据库代码页中。某些 Unicode 字符无法使用此代码页来表示，因此可能会导致转换期间抛出异常。这可能会在执行附加操作或执行 UDF 或 StoredProc set() 调用期间发生。必须对 Java 程序员隐藏 CLOB 和 DBCLOB 之间的这种差别。

下列方法与 COM.ibm.db2.app.Clob 类相关联:

```
public long size() throws Exception
```

此函数返回 CLOB 的长度（以字符计）。

```
public java.io.Reader getReader() throws Exception
```

此函数返回新的 Reader 以读取 CLOB 或 DBCLOB 的内容。可以在该对象上执行高效的寻道/标记操作。

```
public java.io.Writer getWriter() throws Exception
```

此函数返回新的 Writer 以将字符附加至此 CLOB 或 DBCLOB。对此对象的 GetReader() 调用所产生的所有现有 Reader 实例可立即看到附加的字符。

将数据类型为 ARRAY 的参数传递至 Java 例程

DB2 9.5 支持将数据类型为 ARRAY 的参数传递至 Java 过程以及从中传出这些参数。

关于此任务

如果需要执行下列操作，那么不妨选择实现在应用程序和 Java 存储过程之间传递数组的用法:

- 使用单一参数将大量类型相同的数据传递至过程。
- 只使用单一参数将数据类型相同的可变数目的输入传递至过程。

例如，您可以在不知道一个班的学生人数的情况下，使用单一参数将所有学生的姓名传递至过程。如果未使用 ARRAY 数据类型，那么每个学生姓名需要一个参数才能完成此操作。

过程

要传递类型为 ARRAY 的参数，请执行下列操作:

1. 必须已定义 ARRAY 数据类型。必须执行 CREATE TYPE 语句才能定义数组类型。
2. 过程定义必须包含属于已定义类型的参数。例如，如果通过执行 CREATE TYPE 语句创建了 ARRAY 数据类型 IntArray，那么必须执行下列操作才能将此类型的参数传递至过程:

示例

```
CREATE PROCEDURE inArray (IN input IntArray)
LANGUAGE JAVA
PARAMETER STYLE JAVA
EXTERNAL NAME 'MyProcs:MyArrayProcs!inArray';
```

在过程定义中，数组参数的类型为 java.sql.Array。在该过程中，会使用 getArray() 方法将自变量映射至 Java 数组，如下列示例所示。请注意，对数组使用 Integer，而不是 int（或其他基本类型）。

```
static void inArray(java.sql.Array input)
{
Integer[] inputArr = (Integer [])input.getArray();
int sum = 0;
for(int i=0, i < inputArr.length; i++)
{
sum += inputArr[i];
}
}
```

从 JDBC 过程中返回结果集

您可以开发将结果集返回给调用例程或应用程序的 JDBC 过程。在 JDBC 过程中，使用 ResultSet 对象来处理结果集的返回。

过程

要从 JDBC 过程中返回结果集，请执行下列操作:

1. 对于要返回的每个结果集，请将一个类型为 ResultSet[] 的参数包含在过程声明中。例如，下列函数特征符会接受 ResultSet 对象数组:

   ```
   public static void getHighSalaries(
     double inSalaryThreshold,         // double input
     int[] errorCode,                  // SQLCODE output
     ResultSet[] rs)                   // ResultSet output
   ```

2. 使用 Connection 对象来打开调用者的数据库连接:

   ```
   Connection con =
       DriverManager.getConnection("jdbc:default:connection");
   ```

3. 使用 PreparedStatement 对象来编译将生成结果集的 SQL 语句。在下列示例中，随后的准备工作是，将输入变量（称为 inSalaryThreshold- 请参阅以上所示的函数特征符示例）分配到查询语句中参数标记的值。参数标记由“?”或者后跟名称的冒号（:*名称*）指示。

   ```
   String query =
       "SELECT name, job, CAST(salary AS DOUBLE) FROM staff " +
       "  WHERE salary > ? " +
       "  ORDER BY salary";

   PreparedStatement stmt = con.prepareStatement(query);
   stmt.setDouble(1, inSalaryThreshold);
   ```

4. 执行语句:

   ```
   rs[0] = stmt.executeQuery();
   ```

5. 结束过程体。

下一步做什么

如果您尚未这样做，请开发一个会接受存储过程结果集的客户机应用程序或调用者例程。

从 SQLJ 过程中返回结果集

您可以开发将结果集返回给调用例程或应用程序的 SQLJ 过程。在 SQLJ 过程中，使用 ResultSet 对象来处理结果集的返回。

过程

要从 SQLJ 过程中返回结果集，请执行下列操作:

1. 声明迭代器类以处理查询数据。例如:

```
#sql iterator SpServerEmployees(String, String, double);
```

2. 对于要返回的每个结果集，请将一个类型为 ResultSet[] 的参数包含在过程声明中。例如，下列函数特征符会接受 ResultSet 对象数组:

```
public static void getHighSalaries(
  double inSalaryThreshold,         // double input
  int[] errorCode,                  // SQLCODE output
  ResultSet[] rs)                   // ResultSet output
```

3. 实例化迭代器对象。例如:

```
SpServerEmployees c1;
```

4. 将生成结果集的 SQL 语句指定给迭代器。在下列示例中，在查询 WHERE 子句中使用主变量（称为 inSalaryThreshold - 请参阅以上显示的函数特征符示例）:

```
#sql c1 = {SELECT name, job, CAST(salary AS DOUBLE)
             FROM staff
             WHERE salary > :inSalaryThreshold
             ORDER BY salary};
```

5. 执行语句并获取结果集:

```
rs[0] = c1.getResultSet();
```

下一步做什么

如果您尚未这样做，请开发一个会接受过程结果集的客户机应用程序或调用者例程。

在 JDBC 应用程序和例程中接收过程结果集

可以接收过程（从 JDBC 例程或应用程序调用）的结果集。

过程

要从 JDBC 例程或应用程序中接受过程结果集，请执行下列操作:

1. 使用 Connection 对象来打开数据库连接:

```
Connection con =
    DriverManager.getConnection("jdbc:db2:sample", userid, passwd);
```

2. 编译将调用过程来返回结果集（使用 CallableStatement 对象）的 CALL 语句。在下列示例中，会调用过程 GET_HIGH_SALARIES。编译后，是将先前语句中参数标记的值指定给输入变量（称为 inSalaryThreshold - 要传递至过程的数值）。参数标记由“?”或者后跟名称的冒号（:*名称*）指示。

```
String query = "CALL GET_HIGH_SALARIES(?)";

CallableStatement stmt = con.prepareCall(query);
stmt.setDouble(1, inSalaryThreshold);
```

3. 调用过程:

```
stmt.execute();
```

4. 使用 CallableStatement 对象的 getResultSet() 方法来接受过程的第一个结果集，然后使用 fetchAll() 方法从结果集访存行:

```
ResultSet rs = stmt.getResultSet();

// Result set rows are fetched and printed to screen.
while (rs.next())
{
  r++;
  System.out.print("Row: " + r + ": ");
  for (int i=1; i <= numOfColumns; i++)
  {
    System.out.print(rs.getString(i));
    if (i != numOfColumns)
    {
      System.out.print(", ");
    }
  }
  System.out.println();
}
```

5. 对于多个结果集，使用 CallableStatement 对象的 getNextResultSet() 方法来启用读取后续结果集。然后，重复先前步骤中的过程，其中 ResultSet 对象会接受当前结果集，并访存结果集行。例如:

```
while (callStmt.getMoreResults())
{
  rs = callStmt.getResultSet()

  ResultSetMetaData stmtInfo = rs.getMetaData();
  int numOfColumns = stmtInfo.getColumnCount();
  int r = 0;

  // Result set rows are fetched and printed to screen.
  while (rs.next())
  {
    r++;
    System.out.print("Row: " + r + ": ");
    for (int i=1; i <= numOfColumns; i++)
    {
      System.out.print(rs.getString(i));
      if (i != numOfColumns)
      {
        System.out.print(", ");
      }
    }
    System.out.println();
  }
}
```

6. 使用 close() 方法来关闭 ResultSet 对象:

```
rs.close();
```

在 SQLJ 应用程序和例程中接收过程结果集

可以接收过程（从 SQLJ 例程或应用程序调用）的结果集。

过程

要从 SQLJ 例程或应用程序中接受过程结果集，请执行下列操作:

1. 使用 Connection 对象来打开数据库连接:

```
Connection con =
    DriverManager.getConnection("jdbc:db2:sample", userid, passwd);
```

2. 使用 DefaultContext 对象来设置缺省上下文:

```
DefaultContext ctx = new DefaultContext(con);
DefaultContext.setDefaultContext(ctx);
```

3. 使用 ExecutionContext 对象来设置执行上下文:

```
ExecutionContext execCtx = ctx.getExecutionContext();
```

4. 调用返回结果集的过程。在下列示例中，会调用过程 GET_HIGH_SALARIES 并传递输入变量（称为 inSalaryThreshold）:

```
#sql {CALL GET_HIGH_SALARIES(:in inSalaryThreshold, :out outErrorCode)};
```

5. 声明 ResultSet 对象，并使用 ExecutionContext 对象的 getNextResultSet() 方法来接受过程结果集。对于多个结果集，将 getNextResultSet() 调用放入循环结构。过程所返回的每个结果集都会衍生一个循环迭代。在循环中，您可以访存结果集行方法，然后使用 ResultSet 对象的 close() 方法来关闭结果集对象。例如:

```
ResultSet rs = null;

while ((rs = execCtx.getNextResultSet()) != null)
{
  ResultSetMetaData stmtInfo = rs.getMetaData();
  int numOfColumns = stmtInfo.getColumnCount();
  int r = 0;

  // Result set rows are fetched and printed to screen.
  while (rs.next())
  {
    r++;
    System.out.print("Row: " + r + ": ");
    for (int i=1; i <= numOfColumns; i++)
    {
      System.out.print(rs.getString(i));
      if (i != numOfColumns)
      {
        System.out.print(", ");
      }
    }
    System.out.println();
  }

  rs.close();
}
```

Java 例程限制

Java 例程适用下列限制:

- 在使用 DB2 Universal JDBC Driver 时，无法在 Java 数据库应用程序中调用内置过程 install_jar（用来将 JAR 文件中的 Java 例程代码部署至数据库服务器文件系统）。此驱动程序不支持此过程。

  建议的替代选择是使用 DB2 命令行处理器。

- 不论指定的 PARAMETER STYLE 子句值为何，Java 例程的 CREATE PROCEDURE 或 CREATE FUNCTION 语句都不支持 PROGRAM TYPE MAIN 子句。

- 参数样式 JAVA 不支持下列功能:
  - 表函数
  - 函数中的暂存区
  - 访问函数中的 DBINFO 结构
  - 函数中的 FINAL CALL 调用

  如果您需要这些功能，那么建议的替代选择是使用参数样式 DB2GENERAL 来创建 Java 函数，或使用 C 或 C++ 编程语言来创建该函数。
- 不支持从 Java 例程执行 Java 本机接口（JNI）调用。

  如果需要从 Java 例程调用 C 或 C++ 代码，那么您可以通过调用单独定义的 C 或 C++ 例程来实现此目的。
- 当前不支持 NOT FENCED Java 例程。将调用定义为 NOT FENCED 的 Java 例程，如同它定义为 FENCED THREADSAFE 一样。
- Java 存储过程不能依赖于任何非系统资源，例如属性文件。如果您调用依赖于非系统资源的 Java 存储过程，那么不会装入这些资源，并且不会返回错误。
- 必须使用 DB2GENERAL 参数样式创建 Java 通用表函数。

Java 表函数执行模型

对于使用 Java 编写且使用 PARAMETER STYLE DB2GENERAL 的表函数，必须了解 DB2 数据库系统在处理给定语句的每个时刻所发生的情况。

下表针对典型表函数详细说明了此信息。阐述了 NO FINAL CALL 和 FINAL CALL 情况（在这两种情况下都采用 SCRATCHPAD）。

扫描时间点	NO FINAL CALL LANGUAGE JAVA SCRATCHPAD	FINAL CALL LANGUAGE JAVA SCRATCHPAD
在第一次 OPEN 表函数之前	• 没有调用。	• 调用类构造函数（意味着新的暂存区）。使用 FIRST 调用来调用 UDF。 • 构造函数会初始化类和暂存区变量。方法连接至 Web 服务器。
在每次 OPEN 表函数时	• 调用类构造函数（意味着新的暂存区）。使用 OPEN 调用来调用 UDF 方法。 • 构造函数会初始化类和暂存区变量。方法连接至 Web 服务器，并扫描 Web 数据。	• 使用 OPEN 调用来打开 UDF 方法。 • 方法会扫描它所需的任何 Web 数据。（可能可以避免在 CLOSE 重定位之后重新打开，取决于暂存区中所保存的内容。）
在每次 FETCH 表函数的新数据行时	• 使用 FETCH 调用来调用 UDF。 • 方法访存并返回下一数据行或 EOT。	• 使用 FETCH 调用来调用 UDF。 • 方法访存并返回新的数据行或 EOT。

扫描时间点	NO FINAL CALL LANGUAGE JAVA SCRATCHPAD	FINAL CALL LANGUAGE JAVA SCRATCHPAD
在每次 CLOSE 表函数时	• 使用 CLOSE 调用来调用 UDF 方法。会调用 close() 方法（如果类存在此方法）。 • 方法会关闭它的 Web 扫描，并断开与 Web 服务器的连接。close() 不需要执行任何操作。	• 使用 CLOSE 调用来调用 UDF 方法。 • 方法可重定位至扫描顶部，或关闭扫描。它可以将任何状态保存至暂存区，该暂存区会持续保存。
在最终 CLOSE 表函数之后	• 没有调用。	• 使用 FINAL 调用来调用 UDF。会调用 close() 方法（如果类存在此方法）。 • 方法会断开与 Web 服务器的连接。close() 方法不需要执行任何操作。

注:

1. 术语“UDF 方法”是指实现 UDF 的 Java 类方法。在 CREATE FUNCTION 语句的 EXTERNAL NAME 子句中标识此方法。

创建 Java 例程

创建 Java 例程包括执行用于在 DB2 数据库服务器中定义该例程的 CREATE 语句以及开发对应例程定义的例程实现。

开始之前

- 复审 第 215 页的第 7 章，『Java 例程』。
- 确保您可以访问 DB2 版本 9 服务器（其中包括实例和数据库）。
- 请确保操作系统处于 DB2 数据库产品支持的版本级别。
- 确保 Java 开发软件为支持 Java 例程开发的版本级别。请参阅第 215 页的『受支持的 Java 例程开发软件』。
- 确保 第 217 页的『Java 例程驱动程序的规范』 开发有效。
- 有权执行 CREATE PROCEDURE 或 CREATE FUNCTION 语句。

有关与 Java 例程相关联的限制列表，请参阅:

- 第 235 页的『Java 例程限制』

关于此任务

可用来创建 Java 例程的方法如下所示:

- 使用 IBM Data Studio
- 使用 IBM Optim Development Studio
- 使用 IBM Rational Application Developer 中的 DB2 例程开发功能
- 使用 DB2 命令窗口

通常，使用 IBM Data Studio 创建 Java 例程最容易，虽然许多开发者喜欢从集成 Java 开发环境（随 IBM Rational Application Developer 提供）创建 Java 例程。如果无法使用这些图形工具，那么 DB2 命令窗口可通过命令行界面提供类似的支持。

执行下列其中一个过程来创建 Java 例程:

过程

- 使用 IBM Data Studio 来创建 Java 例程
- 使用 Rational Application Developer 创建 Java 例程
- 『从命令行创建 Java 例程』

从命令行创建 Java 例程

以创建具有其他实现的外部例程的方式来创建对 Java 类进行引用的过程和函数。此任务由几个步骤组成，其中包括构造例程的 CREATE 语句、编写和编译（转换）例程实现以及将 Java 类部署至 DB2 数据库服务器。

开始之前

- 复审 第 215 页的第 7 章，『Java 例程』。
- 确保您可以访问 DB2 版本 9 数据库服务器（其中包括实例和数据库）。
- 请确保操作系统处于 DB2 数据库产品支持的版本级别。
- 确保 Java 开发软件为支持 Java 例程开发的版本级别。请参阅第 215 页的『受支持的 Java 例程开发软件』。
- 确保 第 217 页的『Java 例程驱动程序的规范』 开发有效。
- 有权执行 CREATE PROCEDURE 或 CREATE FUNCTION 语句。

关于此任务

在下列情况下，您不妨选择实现 Java 例程:

- 您需要在例程中封装复杂逻辑，以访问数据库或在数据库外部执行操作。
- 您要求从下列任何项调用封装的逻辑: 多个应用程序、CLP、另一个例程（过程、函数（UDF）或方法）或触发器。
- 您最擅长使用 Java 以及其中一个 JDBC 或 SQLJ 应用程序编程接口来编写此逻辑。

过程

1. 使用 Java 编写例程逻辑。
   - 必须使用其中一种支持的参数样式来实现例程参数特征符。强烈建议将参数样式 JAVA 用于所有 Java 例程。有关参数特征符和参数实现的更多信息，请参阅:
     - 第 221 页的『Java 例程中的参数』
     - 第 221 页的『参数样式 JAVA 过程』
     - 第 222 页的『PARAMETER STYLE JAVA Java 函数和方法』
   - 使用针对 Java 数据库应用程序声明变量的方法来声明变量。确保正确地使用映射至 DB2 SQL 数据类型的数据类型。

     有关 DB2 和 Java 数据类型之间的数据类型映射的更多信息，请参阅:
     - *Developing Java Applications* 中的『映射至 Java 应用程序中的数据库数据类型的数据类型』

- 包括例程逻辑。例程逻辑可以包含 Java 编程语言中所支持的任何代码。此外，它能够以在 Java 数据库应用程序中执行 SQL 语句的方法来执行 SQL 语句。

  有关在 Java 代码中执行 SQL 语句的更多信息，请参阅:

  – *Developing Java Applications* 中的『用于执行 SQL 的 JDBC 接口』
  – *Developing Java Applications* 中的『SQLJ 应用程序中的 SQL 语句执行』
- 如果例程是过程且您可能需要将结果集返回给例程调用者，那么您不需要该结果集的任何参数。有关返回 Java 例程的结果集的更多信息，请参阅:
  – 第 232 页的『从 JDBC 过程中返回结果集』
  – 第 233 页的『从 SQLJ 过程中返回结果集』
- 在例程末尾设置例程返回值。

2. 构建代码以产生包含 Java 类文件集合的 Java 类文件或 JAR 文件。 有关如何构建 Java 例程代码的信息，请参阅:
   - *Developing Java Applications* 中的『构建 JDBC 例程』
   - 《*Developing Java Applications*》中的『构建 SQL 例程』
3. 将类文件复制到 DB2 数据库服务器，或将 JAR 文件安装至 DB2 数据库服务器。有关如何实现此目标的信息，请参阅:
   - 第 243 页的『将 Java 例程类文件部署至 DB2 数据库服务器』
   - 第 244 页的『数据库服务器上的 JAR 文件管理』

   建议将 DB2 例程的相关联类文件存储至 *function* 目录。要了解有关函数目录的更多信息，请参阅与下列其中一个语句中的 EXTERNAL 子句相关的信息: CREATE PROCEDURE 或 CREATE FUNCTION。

   如果需要，您可以将库复制到服务器上的另一个目录，但是为了成功调用例程，您必须记下库的标准路径名，因为下一步需要该标准路径名。
4. 动态或静态地执行适合于例程类型的 CREATE 语句: CREATE PROCEDURE 或 CREATE FUNCTION。
   - 使用 JAVA 来指定 LANGUAGE 子句
   - 使用支持的参数样式（在例程代码中实现）的名称来指定 PARAMETER STYLE 子句。强烈建议使用 PARAMETER STYLE JAVA，除非只有在使用 PARAMETER STYLE DB2GENERAL 时才支持您所需要的功能。
   - 使用 JAR 文件或 Java 类（将通过下列其中一个值与例程相关联）的名称来指定 EXTERNAL 子句:
     – Java 类文件的标准路径名
     – 例程 Java 类文件的相对路径名（相对于函数目录）。
     – 数据库服务器上包含 Java 类的 JAR 文件的 JAR 文件标识

     缺省情况下，DB2 数据库系统将在函数目录中查找库，除非在 EXTERNAL 子句中指定库的 JAR 文件标识和类、标准路径名或相对路径名。
   - 如果您的例程是过程且它会将一个或多个结果集返回给调用者，请指定具有数值的 DYNAMIC RESULT SETS。
   - 在 CREATE 语句中指定要用来描述例程的特征的任何其他非缺省子句值。

下一步做什么

要调用 Java 例程，请参阅第 263 页的第 10 章，『调用例程』。

构建 Java 例程代码

在编写 Java 例程实现代码后，必须先构建，然后才能部署例程组合件并调用例程。构建 Java 例程所需的步骤类似于构建任何外部例程所需的步骤，但是存在某些差别。

过程

可以使用几种方法来构建 Java 例程:

- 使用随 IBM Data Studio 提供的图形工具
- 使用 IBM Optim Development Studio 中提供的图形工具
- 使用 IBM Rational Application Developer 中提供的图形工具
- 使用 DB2 样本构建脚本
- 从 DB2 命令窗口输入命令

可以定制图形工具和 DB2 数据库系统以针对各种操作系统且使用各种设置来构建 Java 例程。例程的样本构建脚本和批处理文件设计用于通过缺省的支持开发软件，针对特定操作系统构建 DB2 样本例程（过程和用户定义的函数）。
使用 JDBC 和 SQLJ 创建的 Java 例程分别具有一组 DB2 样本构建脚本和批处理文件。通常，使用图形工具或构建脚本（可根据需要轻松修改）来构建 Java 例程非常容易，但是，如果也知道如何从 DB2 命令窗口构建例程，往往很有帮助。

构建 JDBC 例程

您可以使用 Java `makefile` 或 javac 命令来构建 JDBC 例程。构建那些例程之后，需要对其进行编目。

关于此任务

下列步骤说明如何构建和运行这些例程:

- `SpServer` 样本 JDBC 存储过程
- `UDFsrv` 样本用户定义的函数，此函数未包含任何 SQL 语句
- `UDFsqlsv` 样本用户定义的函数，此函数包含 SQL 语句

过程

- 要在服务器上从命令行构建和运行 SpServer.java 存储过程，请执行下列操作:
  1. 使用以下命令编译 SpServer.java，以便生成 SpServer.class 文件:

     ```
     javac SpServer.java
     ```

  2. 将 `SpServer.class` 复制到 `sqllib\function` 目录（对于 Windows 操作系统）或 `sqllib/function` 目录（对于 UNIX 操作系统）。
  3. 通过在服务器上运行 spcat 脚本，对例程进行编目。 spcat 脚本将连接到 sample 数据库，对例程取消编目（如果先前已通过调用 SpDrop.db2 对其进行编目的话），然后通过调用 SpCreate.db2 对其进行编目，最后断开与数据库的连接。另外，也可以单独运行 SpDrop.db2 和 SpCreate.db2 脚本。

4. 停止然后重新启动数据库，以便识别新的类文件。必要时，请将类文件的文件方式设置为“读取”，以使其可以被受防护用户读取。
5. 编译并运行 SpClient 客户机应用程序，以便访问存储过程类。

- 要在服务器上从命令行构建和运行 UDFsrv.java 用户定义的函数程序（不包含 SQL 语句的用户定义的函数），请执行下列操作:
  1. 使用以下命令编译 UDFsrv.java，以便生成 UDFsrv.class 文件:

     ```
     javac UDFsrv.java
     ```

  2. 将 UDFsrv.class 复制到 sqllib\function 目录（对于 Windows 操作系统）或 sqllib/function 目录（对于 UNIX 操作系统）。
  3. 编译并运行调用了 UDFsrv 的客户机程序。

     要访问 UDFsrv 库，您可以使用 Java 应用程序 UDFcli.java 或者 SQLJ 客户机应用程序 UDFcli.sqlj。这两个版本的客户机程序都包含 SQL 语句 CREATE FUNCTION（用于向数据库注册用户定义的函数），并包含使用了用户定义的函数的 SQL 语句。
- 要在服务器上从命令行构建和运行 UDFsqlsv.java 用户定义的函数程序（包含 SQL 语句的用户定义的函数），请执行下列操作:
  1. 使用以下命令编译 UDFsqlsv.java，以便生成 UDFsqlsv.class 文件:

     ```
     javac UDFsqlsv.java
     ```

  2. 将 UDFsqlsv.class 复制到 sqllib\function 目录（对于 Windows 操作系统）或 sqllib/function 目录（对于 UNIX 操作系统）。
  3. 编译并运行调用了 UDFsqlsv 的客户机程序。

     要访问 UDFsqlsv 库，您可以使用 Java 应用程序 UDFsqlcl.java。此客户机程序包含 SQL 语句 CREATE FUNCTION（用于向数据库注册用户定义的函数），并包含使用了用户定义的函数的 SQL 语句。

构建 SQL 例程

您可以使用 Java makefile 或 bldsqljs 构建文件来构建 SQLJ 例程。构建那些例程之后，需要对其进行编目。

关于此任务

下列步骤演示如何构建和运行 SpServer 样本 SQLJ 存储过程。这些步骤使用构建文件 bldsqljs (UNIX) 或 bldsqljs.bat (Windows)，此构建文件包含用于构建 SQLJ applet 或应用程序的命令。

此构建文件接受 6 个参数: 在 UNIX 上，它们是 $1、$2、$3、$4、$5 和 $6; 在 Windows 上，它们是 %1、%2、%3、%4、%5 和 %6。第一个参数指定程序的名称。第二个参数指定数据库实例的用户标识，第三个参数指定密码。第四个参数指定服务器名称。第五个参数指定端口号。第六个参数指定数据库名。对于除第一个参数（程序名）以外的所有参数，可以使用缺省值。有关使用缺省参数值的详细信息，请参阅构建文件。

过程

1. 使用以下命令来构建存储过程应用程序:

   ```
   bldsqljs SpServer <用户标识> <密码> <服务器名称> <端口号> <数据库名>
   ```

2. 使用以下命令对存储过程进行编目:

   ```
   spcat
   ```

   此脚本将连接到 sample 数据库，对例程取消编目（如果先前已通过调用 SpDrop.db2 对其进行编目的话），然后通过调用 SpCreate.db2 对其进行编目，最后断开与数据库的连接。另外，也可以单独运行 SpDrop.db2 和 SpCreate.db2 脚本。
3. 停止然后重新启动数据库，以便识别新的类文件。必要时，请将类文件的文件方式设置为“读取”，以使其可以被受防护用户读取。
4. 编译并运行 SpClient 客户机应用程序，以便访问存储过程类。 可以使用应用程序构建文件 bldsqlj (UNIX) 或 bldsqlj.bat (Windows) 来构建 SpClient。

Java（SQLJ）例程的编译和链接选项

用于 UNIX 的 SQLJ 例程选项

bldsqljs 构建脚本用于在 UNIX 操作系统上构建 SQLJ 例程。bldsqljs 指定了一组 SQLJ 转换程序和定制程序选项。

建议: 在 UNIX 平台上构建 SQLJ 例程时，请使用 bldsqljs 所使用的 SQLJ 转换程序和定制程序选项。

bldsqljs 包括的选项如下所示:

sqlj SQLJ 转换程序（同时编译程序）。

"${progname}.sqlj"
SQLJ 源文件。progname=${1%.sqlj} 命令用于除去输入文件名中的扩展名（如果有的话），以便再次添加扩展名时不会导致扩展名重复。

db2sqljcustomize
SQLJ 概要文件定制程序。

-url 指定用于建立数据库连接的 JDBC URL，例如 jdbc:db2://servername:50000/sample。

-user 指定用户标识。

-password
指定密码。

"${progname}_SJProfile0"
指定程序的序列化概要文件。

用于 Windows 的 SQLJ 例程选项

bldsqljs.bat 批处理文件用于在 Windows 操作系统上构建 SQLJ 例程。bldsqljs.bat 指定了一组 SQLJ 转换程序和定制程序选项。

建议: 在 Windows 操作系统上构建 SQLJ 例程时，请使用 bldsqljs.bat 所使用的 SQLJ 转换程序和定制程序选项。

在 Windows 操作系统上，bldsqljs.bat 批处理文件使用下列 SQLJ 转换程序和定制程序选项。DB2 建议使用以下选项来构建 SQLJ 例程（存储过程和用户定义的函数）。

sqlj SQLJ 转换程序（同时编译程序）。

%1.sqlj
SQLJ 源文件。

db2sqljcustomize
DB2 for Java 概要文件定制程序。

-url 指定用于建立数据库连接的 JDBC URL，例如 jdbc:db2://servername:50000/sample。

-user 指定用户标识。

-password
指定密码。

%1_SJProfile0
指定程序的序列化概要文件。

将 Java 例程类文件部署至 DB2 数据库服务器

必须将 Java 例程实现部署至 DB2 数据库服务器文件系统，以便在执行例程调用时可以找到、装入和运行这些实现。

可以将一个或多个 Java 例程实现包括在个别 Java 类文件中。可以将包含 Java 例程实现的 Java 类文件全部收集到 JAR 文件。用来实现例程的 Java 类文件必须位于您安装在 DB2 数据库的 JAR 文件中。

先决条件:

- 标识要在其中部署例程类的 DB2 数据服务器。
- 在 UNIX 操作系统上，标识 DB2 实例所有者的用户标识。如果此用户标识是未知的，请与您的数据库管理员联系。

要部署个别 Java 例程类文件，请执行下列操作:

- 将 Java 例程类部署至 DB2 函数目录。

  在 UNIX 操作系统上，函数目录定义为: *安装路径*/function，其中 *安装路径* 是 DB2 数据库管理器安装路径。例如，*$HOME*/sqllib/function，其中 *$HOME* 是实例所有者的主目录。

  在 Windows 操作系统上，函数目录定义为: *实例概要文件路径*\function，其中，*实例概要文件路径* 是在 **db2icrt**（创建实例）命令中指定的路径。要查找实例概要文件路径名，请发出如下所示的 **db2set** 命令:

  ```
  db2set DB2INSTPROF
  ```

  例如，C:\Documents and Settings\All Users\Application Data\IBM\DB2\db2copy1\function

  如果声明要成为 Java 程序包一部分的类，请在函数目录中创建对应于标准类名的子目录，然后将相关类文件放入对应的子目录。例如，如果您为 Linux 操作系统创建一个名为 ibm.tests.test1 的类，请将对应的 Java 字节码文件（test1.class）存储在 *$HOME*/sqllib/function/ibm/tests 中，其中 *$HOME* 是实例所有者的主目录。

要部署包含 Java 例程类文件的 JAR 文件，请执行下列操作:

- 必须将包含 Java 例程类文件的 JAR 文件安装至 DB2 数据库服务器文件系统。请参阅『数据库服务器上的 JAR 文件管理』。

在部署 Java 例程类文件以及执行 CREATE 语句以在数据库中定义例程后，您可以调用该例程。

将具有从属类的 Java 例程类文件部署至 DB2 数据库服务器

如果 Java 例程类文件所依赖的类不是标准 Java 或 DB2 类的一部分，请针对每个从属类重复上一部分中所标识的步骤。

或者，可以将 DB2 数据库配置为在 **CLASSPATH** 环境变量中搜索目录，以检测从属类。在 Windows 操作系统上，数据库服务器将自动在 **CLASSPATH** 系统环境变量中搜索指定的目录。在 UNIX 操作系统上，如果将文本“CLASSPATH”指定为 **DB2ENVLIST** 环境变量的一部分，那么数据库服务器将搜索实例所有者的 **CLASSPATH** 环境变量。强烈建议安装从属类，而不要依赖于 **CLASSPATH** 环境变量。

数据库服务器上的 JAR 文件管理

要部署包含 Java 例程类文件的 JAR 文件，您必须将 JAR 文件安装至 DB2 数据库服务器。要从 IBM Data Server Client 完成此操作，请使用会在 DB2 数据库服务器上安装、替换或除去 JAR 文件的内置例程。

要在 DB2 数据库系统实例中安装、替换或除去 JAR 文件，请使用随 DB2 提供的存储过程:

安装

```
sqlj.install_jar( jar-url, jar-id )
```

注: sqlj.install_jar 调用者的授权标识具有的特权必须至少包括下列其中一项:

- 对隐式或显式指定的模式的 CREATEIN 特权
- DBADM 权限

替换

```
sqlj.replace_jar( jar-url, jar-id )
```

除去

```
sqlj.remove_jar( jar-id )
```

- *jar-url*: 包含要安装或替换的 JAR 文件的 URL。唯一支持的 URL 方案是“file:”。
- *jar-id*: 唯一的字符串标识，长度可达 128 个字节。它在数据库中指定与 *jar-url* 文件相关联的 JAR 标识。

注: 从应用程序调用时，存储过程 sqlj.install_jar 和 sqlj.remove_jar 具有其他参数。它是一个整数值，规定在指定的 JAR 文件中使用部署描述符。目前，不支持部署参数，且会拒绝任何指定非零值的调用。

下列是一系列指示如何使用前面的 JAR 文件管理存储过程的示例。

要在数据库实例中将位于路径 /home/bob/bobsjar.jar 的 JAR 注册为 MYJAR，请执行下列操作:

```
CALL sqlj.install_jar( 'file:/home/bob/bobsjar.jar', 'MYJAR' )
```

使用 bobsjar.jar 文件的后续 SQL 命令会使用名称 MYJAR 来引用该文件。

要使用不同的 JAR（包含某些已更新的类）来替换 MYJAR，请执行下列操作:

```
CALL sqlj.replace_jar( 'file:/home/bob/bobsnewjar.jar', 'MYJAR' )
```

要从数据库目录中除去 MYJAR，请执行下列操作:

```
CALL sqlj.remove_jar( 'MYJAR' )
```

注: 在 Windows 操作系统上，DB2 数据库系统会将 JAR 文件存储至 **DB2INSTPROF** 实例特定注册表设置所指定的路径。要使 JAR 文件成为实例的特有 JAR 文件，您必须为该实例的 **DB2INSTPROF** 指定唯一值。

更新 Java 例程类

如果要更改 Java 例程的逻辑，必须更新例程源代码、编译（转换）代码并接着更新部署到 DB2 数据库服务器的 Java 类版本或 JAR 文件版本。

关于此任务

要确保 DB2 数据库管理器使用新版本的 Java 例程，必须执行一个内置过程将新版本的 Java 类装入内存。

过程

要更新 Java 例程类，请执行下列操作:

1. 将新的 Java 类或 JAR 文件部署到 DB2 数据库服务器。
2. 对于受防护例程，执行以下内置过程:

   ```
   CALL SQLJ.REFRESH_CLASSES()
   ```

   这将强制 DB2 数据库管理器在执行下次落实或回滚操作后将新类装入内存。

   对于未受防护例程而言，此步骤无效。对于未受防护例程而言，您必须显式地停止然后重新启动 DB2 数据库管理器才能装入并使用新版本的 Java 例程类。

结果

如果您未执行以上列出的步骤，那么在更新 Java 例程类之后，DB2 数据库管理器将继续使用较低版本的类。

Java (JDBC) 例程示例

开发使用 JDBC 应用程序编程接口的 Java 例程时，参考示例以获取 CREATE 语句的含义和 Java 例程代码的形式会有所帮助。

关于此任务

下列主题包含 Java 过程和函数示例:

过程

- Java (JDBC) 过程示例
- 具有 XML 功能的 Java (JDBC) 过程示例

- Java (JDBC) 函数示例

示例: Java (JDBC) 过程中的数组数据类型

使用数组数据类型的 Java 例程示例。

下列示例说明了带有数组数据类型的 IN 和 OUT 参数的 Java 例程框架。

```
CREATE TYPE phonenumbers AS VARCHAR(20) ARRAY[10] %
CREATE PROCEDURE javaproc( IN in1 phonenumbers,
                           OUT out1 phonenumbers)
  LANGUAGE java
  PARAMETER STYLE java
  FENCED THREADSAFE
  EXTERNAL NAME 'myjar:stpclass.javastp' %

import java.sql.Array;

public static void javaproc(Array input, Array[] output)
{
output[0] = input;
}
```

示例: Java (JDBC) 过程中的 XML 和 XQuery 支持

在了解 Java 过程的基本原理、如何在 Java 中使用 JDBC 应用程序编程接口 (API) 进行编程，以及 XQuery 之后，您可以开始创建并使用可查询 XML 数据的 Java 过程。

本 Java 过程示例说明:

- 参数样式 JAVA 过程的 CREATE PROCEDURE 语句
- 参数样式 JAVA 过程的源代码
- 数据类型为 XML 的输入和输出参数
- 在查询中使用 XML 输入参数
- 将 XQuery 的结果（XML 值）指定给输出参数
- 将 SQL 语句的结果（XML 值）指定给输出参数

先决条件

在使用本 Java 过程示例之前，您不妨阅读下列主题:

- Java 例程
- 例程
- 构建 Java 例程代码

下列示例利用名为如下定义的表 xmlDataTable （包含如下所示的数据）:

```
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))@
```

过程 在创建您自己的 Java 过程时，请使用下列示例作为参考:

- 『Java 外部代码文件』
- 第 248 页的『示例 1: 带有 XML 参数的参数样式 JAVA 过程』

Java 外部代码文件

本示例显示一个 Java 过程实现。本示例包含两个部分: CREATE PROCEDURE 语句以及过程的外部 Java 代码实现（可根据其构建相关联的 Java 类）。

包含下列示例的过程实现的 Java 源文件称为 `stpclass.java`（随附于 JAR 文件 `myJAR`）。该文件具有下列格式:

```
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
           ...
}
```

在此文件顶部指示 Java 类文件导入。如果此文件中的任何过程包含类型为 XML 的参数或变量，那么将需要 com.ibm.db2.jcc.DB2Xml 导入。

必须记下此类文件的名称以及包含给定过程实现的 JAR 的名称。这些名称很重要，因为每个过程的 CREATE PROCEDURE 语句的 EXTERNAL 子句必须指定此信息，以便 DB2 数据库系统可以在运行时找到此类。

示例 1: 带有 XML 参数的参数样式 JAVA 过程

本示例显示下列内容:

- 参数样式 JAVA 过程的 CREATE PROCEDURE 语句
- 带有 XML 参数的参数样式 JAVA 过程的 Java 代码

此过程接受输入参数（inXML），将包含该值的行插入表，使用 SQL 语句和 XQuery 表达式来查询 XML 数据，然后设置两个输出参数（outXML1 和 outXML2）。

```
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 ;
}
```

第 8 章 OLE 自动化例程设计

对象链接与嵌入（OLE）自动化是 Microsoft 公司的 OLE 2.0 体系结构的组成部分。借助 OLE 自动化，无论使用哪种语言来编写应用程序，应用程序都可以在 OLE 自动化对象中公布它们的属性和方法。

于是，其他应用程序（例如 Lotus Notes® 或 Microsoft Exchange）可以通过 OLE 自动化来利用这些属性和方法，从而集成这些对象。

公布属性和方法的应用程序被称为 OLE 自动化服务器或对象，而访问那些属性和方法的应用程序被称为 OLE 自动化控制器。OLE 自动化服务器是实现了 OLE IDispatch 接口的 COM 组件（对象）。OLE 自动化控制器是 COM 客户机，它通过 IDispatch 接口与自动化服务器进行通信。COM 是 OLE 的基础。对于 OLE 自动化例程而言，DB2 数据库系统充当 OLE 自动化控制器。借助此机制，DB2 数据库系统可以将 OLE 自动化对象的方法作为外部例程进行调用。

注意，所有 OLE 自动化主题都假定您熟悉 OLE 自动化术语和概念。要获取有关 OLE 自动化的概述，请参阅 *Microsoft Corporation: The Component Object Model Specification*, October 1995。有关 OLE 自动化的详细信息，请参阅 *OLE Automation Programmer's Reference*, Microsoft Press, 1996, ISBN 1-55615-851-3。

创建 OLE 自动化例程

将 OLE 自动化例程作为 OLE 自动化对象的公用方法来实现。

关于此任务

OLE 自动化对象必须可由 OLE 自动化控制器在外部进行创建（在此情况下，控制器是 DB2 数据库系统），并且支持后期绑定（又称为基于 IDispatch 的绑定）。OLE 自动化对象必须在 Windows 注册表中使用类标识（CLSID）以及可选的 OLE 程序化标识（progID）进行注册，才能加以标识。progID 可以标识进程中（.DLL）或本地（.EXE）OLE 自动化服务器，或通过 DCOM（分布式 COM）来标识远程服务器。

过程

要注册 OLE 自动化例程，请执行下列操作:

在编写 OLE 自动化对象后，您需要使用 CREATE 语句将对象的方法创建为例程。创建 OLE 自动化例程与注册 C 或 C++ 例程非常相似，但是您必须使用下列选项:

- LANGUAGE OLE
- FENCED NOT THREADSAFE，因为 OLE 自动化例程必须以 FENCED 方式运行，而不能以 THREADSAFE 方式运行。

外部名包含标识 OLE 自动化对象和方法名（以惊叹号 ! 隔开）的 OLE progID:

```
CREATE FUNCTION bcounter () RETURNS INTEGER
  EXTERNAL NAME 'bert.bcounter!increment'
  LANGUAGE OLE
  FENCED
  NOT THREADSAFE
```

```
SCRATCHPAD
FINAL CALL
NOT DETERMINISTIC
NULL CALL
PARAMETER STYLE DB2SQL
NO SQL
NO EXTERNAL ACTION
DISALLOW PARALLEL;
```

OLE 方法实现的调用约定与使用 C 或 C++ 编写的例程的约定相同。先前方法的 BASIC 语言实现类似如下（请注意，在 BASIC 中，参数缺省定义为通过引用进行调用）：

```
Public Sub increment(output As Long, _
                     indicator As Integer, _
                     sqlstate As String, _
                     fname As String, _
                     fspecname As String, _
                     sqlmsg As String, _
                     scratchpad() As Byte, _
                     calltype As Long)
```

OLE 例程对象实例和暂存区注意事项

OLE 自动化 UDF 和方法（OLE 自动化对象的方法）应用于 OLE 自动化对象的实例。DB2 数据库系统将为 SQL 语句中的每个 UDF 或方法引用创建一个对象实例。可以将对象实例反复用于 SQL 语句中的 UDF 或方法引用的后续方法调用，也可以在执行方法调用后释放该实例并为每个后续方法调用创建新实例。可以通过 CREATE 语句中的 SCRATCHPAD 选项来指定适当的行为。对于 LANGUAGE OLE 子句，SCRATCHPAD 选项具有 C 或 C++ 所没有的附加语义，即，创建单一对象实例并将其反复用于整个查询，如果指定了 NO SCRATCHPAD，那么可以在执行每次调用方法时创建新的对象实例。

通过使用暂存区，方法就可以跨函数或方法调用在对象的实例变量中维护状态信息。这还可以提高性能，其原因在于，只创建一次对象实例并将其反复用于后续调用。

OLE 自动化支持的 SQL 数据类型

DB2 数据库系统处理 SQL 类型与 OLE 自动化类型之间的类型转换。

下表概括支持的数据类型及其映射方式。

表 21. SQL 至 OLE 自动化数据类型的映射

SQL 类型	OLE 自动化类型	OLE 自动化类型描述
SMALLINT	short	16 位带符号整数
INTEGER	long	32 位带符号整数
REAL	float	32 位 IEEE 浮点数
FLOAT 或 DOUBLE	double	64 位 IEEE 浮点数
DATE	DATE	64 位浮点小数（自 1899 年 12 月 30 日以来的天数）
TIME	DATE	64 位浮点小数（自 1899 年 12 月 30 日以来的天数）

表 21. SQL 至 OLE 自动化数据类型的映射 (续)

SQL 类型	OLE 自动化类型	OLE 自动化类型描述
TIMESTAMP	DATE	64 位浮点小数（自 1899 年 12 月 30 日以来的天数）
CHAR(*n*)	BSTR	在 *OLE 自动化程序员参考* 中描述了长度前缀字符串。
VARCHAR(*n*)	BSTR	在 *OLE 自动化程序员参考* 中描述了长度前缀字符串。
LONG VARCHAR	BSTR	在 *OLE 自动化程序员参考* 中描述了长度前缀字符串。
CLOB(*n*)	BSTR	在 *OLE 自动化程序员参考* 中描述了长度前缀字符串。
GRAPHIC(*n*)	BSTR	在 *OLE 自动化程序员参考* 中描述了长度前缀字符串。
VARGRAPHIC(*n*)	BSTR	在 *OLE 自动化程序员参考* 中描述了长度前缀字符串。
LONG GRAPHIC	BSTR	在 *OLE 自动化程序员参考* 中描述了长度前缀字符串。
DBCLOB(*n*)	BSTR	在 *OLE 自动化程序员参考* 中描述了长度前缀字符串。
CHAR(*n*)	SAFEARRAY[unsigned char]	由一些 8 位无符号数据项组成的 1 维 Byte() 数组。在 *OLE 自动化程序员参考* 中描述了 SAFEARRAY。
VARCHAR(*n*)	SAFEARRAY[unsigned char]	由一些 8 位无符号数据项组成的 1 维 Byte() 数组。在 *OLE 自动化程序员参考* 中描述了 SAFEARRAY。
LONG VARCHAR	SAFEARRAY[unsigned char]	由一些 8 位无符号数据项组成的 1 维 Byte() 数组。在 *OLE 自动化程序员参考* 中描述了 SAFEARRAY。
CHAR(*n*) FOR BIT DATA	SAFEARRAY[unsigned char]	由一些 8 位无符号数据项组成的 1 维 Byte() 数组。在 *OLE 自动化程序员参考* 中描述了 SAFEARRAY。
VARCHAR(*n*) FOR BIT DATA	SAFEARRAY[unsigned char]	由一些 8 位无符号数据项组成的 1 维 Byte() 数组。在 *OLE 自动化程序员参考* 中描述了 SAFEARRAY。
LONG VARCHAR FOR BIT DATA	SAFEARRAY[unsigned char]	由一些 8 位无符号数据项组成的 1 维 Byte() 数组。在 *OLE 自动化程序员参考* 中描述了 SAFEARRAY。
BLOB(*n*)	SAFEARRAY[unsigned char]	由一些 8 位无符号数据项组成的 1 维 Byte() 数组。在 *OLE 自动化程序员参考* 中描述了 SAFEARRAY。

在 DB2 和 OLE 自动化例程之间传递的数据，是作为通过引用来执行的调用传递。不支持表中未列出的 SQL 类型（例如 BIGINT、DECIMAL 或 LOCATORS）或 OLE 自动化类型（例如 Boolean 或 CURRENCY）。映射至 BSTR 的字符和图形数据会从数据库代码页转换为 UCS-2 方案。UCS-2 又称为 Unicode（IBM 代码页 13488）。在返

回时，会将数据从 UCS-2 转换回为数据库代码页。不论使用的数据库代码页为何，都会执行这些转换。如果未安装这些代码页转换表，那么您将接收到 SQLCODE -332 (SQLSTATE 57017)。

使用 BASIC 和 C++ 编写的 OLE 自动化例程

您可以使用任何语言来实现 OLE 自动化例程。本节显示如何使用 BASIC 或 C++ 作为样本语言来实现 OLE 自动化例程。

下表显示了 OLE 自动化类型至 BASIC 和 C++ 中数据类型的映射。

表 22. SQL 和 OLE 数据类型至 BASIC 和 C++ 数据类型的映射

SQL 类型	OLE 自动化类型	BASIC 类型	C++ 类型
SMALLINT	short	Integer	short
INTEGER	long	Long	long
REAL	float	Single	float
FLOAT 或 DOUBLE	double	Double	double
DATE、TIME 和 TIMESTAMP	DATE	Date	DATE
CHAR(*n*)	BSTR	String	BSTR
CHAR(*n*) FOR BIT DATA	SAFEARRAY[unsigned char]	Byte()	SAFEARRAY
VARCHAR(*n*)	BSTR	String	BSTR
VARCHAR(*n*) FOR BIT DATA	SAFEARRAY[unsigned char]	Byte()	SAFEARRAY
LONG VARCHAR	BSTR	String	BSTR
LONG VARCHAR FOR BIT DATA	SAFEARRAY[unsigned char]	Byte()	SAFEARRAY
BLOB(*n*)	BSTR	String	BSTR
BLOB(*n*) FOR BIT DATA	SAFEARRAY[unsigned char]	Byte()	SAFEARRAY
GRAPHIC(*n*)、VARGRAPHIC(*n*) 和 LONG GRAPHIC, DBCLOB(*n*)	BSTR	String	BSTR

使用 BASIC 实现 OLE 自动化

要使用 BASIC 来实现 OLE 自动化例程，您需要使用对应于已映射至 OLE 自动化类型的 SQL 数据类型的 BASIC 数据类型。

OLE 自动化 UDF `bcounter` 的 BASIC 声明类似如下:

```
Public Sub increment(output As Long, _
                     indicator As Integer, _
                     sqlstate As String, _
                     fname As String, _
                     fspecname As String, _
                     sqlmsg As String, _
                     scratchpad() As Byte, _
                     calltype As Long)
```

使用 C++ 实现 OLE 自动化

OLE 自动化 UDF `increment` 的 C++ 声明如下所示:

```
STDMETHODIMP Ccounter::increment (long   *output,
                                 short  *indicator,
                                 BSTR   *sqlstate,
```

```
                                    BSTR   *fname,
                                    BSTR   *fspecname,
                                    BSTR   *sqlmsg,
                                    SAFEARRAY **scratchpad,
                                    long   *calltype );
```

OLE 支持描述 OLE 自动化对象的属性和方法的类型库。使用对象描述语言（ODL）来描述所提供的对象、属性以及方法。以前显示的 C++ 方法的 ODL 描述如下所示:

```
HRESULT increment ([out]    long  *output,
                   [out]    short *indicator,
                   [out]    BSTR  *sqlstate,
                   [in]     BSTR  *fname,
                   [in]     BSTR  *fspecname,
                   [out]    BSTR  *sqlmsg,
                   [in,out] SAFEARRAY (unsigned char) *scratchpad,
                   [in]     long *calltype);
```

ODL 描述允许您指定参数是输入（in）、输出（out）还是输入/输出（in,out）参数。对于 OLE 自动化例程，会将例程输入参数和输入指示符指定为 [in] 参数，并且将例程输出参数和输出指示符指定为 [out] 参数。对于例程结尾自变量，sqlstate 是 [out] 参数，fname 和 fspecname 是 [in] 参数，scratchpad 是 [in,out] 参数，而 calltype 是 [in] 参数。

OLE 自动化可以定义 BSTR 数据类型以处理字符串。会将 BSTR 定义为 OLECHAR 的指针: typedef OLECHAR *BSTR。对于分配和释放 BSTR，OLE 会实施一种规则，即，被调用的例程会释放作为 by-reference 参数所传入的 BSTR，然后再将新值指定给该参数。由被调用例程接收的一维字节数组（作为 SAFEARRAY**）会适用同一规则。对于 DB2 和 OLE 自动化例程，此规则意外着下列事项:

- [in] 参数: DB2 数据库系统分配和释放 [in] 参数。
- [out] 参数: DB2 数据库系统传入 NULL 指针。[out] 参数必须由被调用的例程分配，且由 DB2 数据库系统释放。
- [in,out] 参数: DB2 数据库系统最初分配 [in,out] 参数。被调用的例程可以释放和重新分配这些参数。这对于 [out] 参数同样成立，DB2 数据库系统释放最终返回的参数。

会将所有其他参数作为指针来传递。DB2 数据库系统分配和管理被引用的内存。

OLE 自动化提供一组数据操作函数来处理 BSTR 和 SAFEARRAY。在 *OLE 自动化程序员参考* 中描述了数据操作函数。

下列 C++ 例程会返回 CLOB 输入参数的前 5 个字符:

```
// UDF DDL: CREATE FUNCTION crunch (CLOB(5k)) RETURNS CHAR(5)

STDMETHODIMP Cobj::crunch (BSTR *in,          // CLOB(5K)
                           BSTR *out,         // CHAR(5)
                           short *indicator1,  // input indicator
                           short *indicator2,  // output indicator
                           BSTR *sqlstate,    // pointer to NULL
                           BSTR *fname,       // pointer to function name
                           BSTR *fspecname,   // pointer to specific name
                           BSTR *msgtext)     // pointer to NULL
  {
     // Allocate BSTR of 5 characters
     // and copy 5 characters of input parameter

     // out is an [out] parameter of type BSTR, that is,
```

```
    // it is a pointer to NULL and the memory does not have to be freed.
    // DB2 will free the allocated BSTR.

    *out = SysAllocStringLen (*in, 5);
    return NOERROR;
};
```

OLE 自动化服务器可以作为*可创建的单用途*或*可创建的多用途*。借助可创建的单用途，使用 CoGetClassObject 来连接至 OLE 自动化对象的每个客户机（即，DB2 FENCED 进程）都将使用它自己的类工厂实例，并在必要时运行 OLE 自动化服务器的新副本。借助可创建的多用途，许多客户机可以连接至同一类工厂。即，由已经处于运行状态的 OLE 服务器副本（如果有的话）提供类工厂的每个实例。如果不存在处于运行状态的 OLE 服务器副本，那么会自动启动一个副本以提供类对象。实现自动化服务器时，您可以选择是使用单用途还是多用途 OLE 自动化服务器。建议使用单用途服务器以获取更高的性能。

第 9 章 OLE DB 用户定义表函数

Microsoft OLE DB 是一组 OLE/COM 接口，可让应用程序对各种信息源中所存储的数据进行一致访问。OLE DB 组件 DBMS 体系结构定义 OLE DB 使用者和 OLE DB 提供者。OLE DB 使用者是任何使用 OLE DB 接口的系统或应用程序；OLE DB 提供者是任何提供 OLE DB 接口的组件。有两类 OLE DB 提供者：*OLE DB 数据提供者* - 拥有数据并将其表格格式的数据提供为行集；*OLE DB service providers* - 没有自己的数据，但通过 OLE DB 接口来产生和使用数据，以封装一些服务。

DB2 数据库系统通过支持您定义可访问 OLE DB 数据源的表函数来简化 OLE DB 应用程序的创建。DB2 是可访问任何 OLE DB 数据或服务提供者的 OLE DB 使用者。您可以对数据源执行操作（其中包括 GROUP BY、JOIN 以及 UNION）以通过 OLE DB 接口提供其数据。例如，您可以定义 OLE DB 表函数以从 Microsoft Access 数据库或 Microsoft Exchange 地址簿返回表，然后创建报告以无缝地组合来自此 OLE DB 表函数的数据以及 DB2 数据库中的数据。

使用 OLE DB 表函数时，可以提供对任何 OLE DB 提供者的内置访问，从而减少应用程序开发工作。对于 C、Java 以及 OLE 自动化表函数，开发者需要实现表函数，而在使用 OLE DB 表函数的情况下，会将通用内置 OLE DB 使用者接口与任何 OLE DB 提供者搭配以检索数据。您只需将表函数注册为 LANGUAGE OLEDB，然后将 OLE DB 提供者以及相关行集作为数据源来引用。不必执行任何 UDF 编程，即可利用 OLE DB 表函数。

要将 OLE DB 表函数与 DB2 数据库一起使用，您必须安装 OLE DB 2.0 或更高版本（可从 Microsoft `http://www.microsoft.com` 下载）。如果未先安装 OLE DB，就尝试调用 OLE DB 表函数，那么 DB2 会发出错误 (SQLCODE -465, SQLSTATE 58032)，原因码 35。有关系统要求以及可用作数据源的 OLE DB 提供者，请参阅数据源文档。有关 OLE DB 规范，请参阅 *Microsoft OLE DB 2.0 Programmer's Reference and Data Access SDK*，Microsoft Press 1998。

OLE DB 表函数的使用限制：OLE DB 表函数无法连接至 DB2 数据库。

创建 OLE DB 表 UDF

过程

要使用单一 CREATE FUNCTION 语句来定义 OLE DB 表函数，您必须执行下列操作:

- 定义 OLE DB 提供者所返回的表
- 指定 LANGUAGE OLEDB
- 标识 OLE DB 行集，并在 EXTERNAL NAME 子句中提供 OLE DB 提供者连接字符串

OLE DB 数据源会以称为*行集* 的表格格式来提供其数据。行集是一组行，每行分别具有一组列。RETURNS TABLE 子句只包含与用户相关的列。根据列名将表函数列绑定至 OLE DB 数据源处行集的列。如果 OLE DB 提供者区分大小写，请使用引号将列名引起来；例如，`"UPPERcase"`。
EXTERNAL NAME 子句可以接受下列任一格式:

```
'server!rowset'
或
'!rowset!connectstring'
```

其中:

server 标识使用 CREATE SERVER 语句来注册的服务器

rowset 标识 OLE DB 提供者所提供的行集或表; 如果表的输入参数要通过命令文本传递至 OLE DB 提供者, 那么此值应该为空。

connectstring

包含连接至 OLE DB 提供者时所需的初始化属性。有关连接字符串的完整语法和语义, 请参阅 *Microsoft OLE DB 2.0 Programmer's Reference and Data Access SDK*（Microsoft Press 1998 年）中的"OLE DB 核心组件的数据链路 API"。

可以在 CREATE FUNCTION 语句的 EXTERNAL NAME 子句中使用 *connection string*, 或者在 CREATE SERVER 语句中指定 CONNECTSTRING 选项。
例如, 您可以使用下列 CREATE FUNCTION 和 SELECT 语句来定义 OLE DB 表函数, 并从 Microsoft Access 数据库返回表:

```
CREATE FUNCTION orders ()
  RETURNS TABLE (orderid INTEGER, ...)
  LANGUAGE OLEDB
  EXTERNAL NAME '!orders!Provider=Microsoft.Jet.OLEDB.3.51;
                 Data Source=c:\msdasdk\bin\oledb\nwind.mdb';

SELECT orderid, DATE(orderdate) AS orderdate,
                DATE(shippeddate) AS shippeddate
FROM TABLE(orders()) AS t
WHERE orderid = 10248;
```

您可以创建并使用服务器名称, 而不是将连接字符串放入 EXTERNAL NAME 子句。例如, 假设您已定义服务器 Nwind, 您可以使用下列 CREATE FUNCTION 语句:

```
CREATE FUNCTION orders ()
  RETURNS TABLE (orderid INTEGER, ...)
  LANGUAGE OLEDB
  EXTERNAL NAME 'Nwind!orders';
```

OLE DB 表函数也允许您指定一个属于任何字符串数据类型的输入参数。使用输入参数将命令文本直接传递至 OLE DB 提供者。如果您定义输入参数, 请不要在 EXTERNAL NAME 子句中提供行集名称。DB2 数据库系统会将命令文本传递至要执行的 OLE DB 提供者, 然后 OLE DB 提供者会将一个行集返回给 DB2 数据库系统。结果行集的列名和数据类型需要与 CREATE FUNCTION 语句中的 RETURNS TABLE 定义兼容。您必须确保正确地命名列, 因为将根据匹配的列名来绑定行集的列名。
下列示例会注册 OLE DB 表函数, 该表函数从 Microsoft SQL Server 7.0 数据库检索商店信息。在 EXTERNAL NAME 子句中提供连接字符串。表函数的输入参数是通过命令文本传递至 OLE DB 提供者, 因此未在 EXTERNAL NAME 子句中指定行集名称。查询示例传入 SQL 命令文本, 以从 SQL Server 数据库中检索前三个商店的信息。

```
CREATE FUNCTION favorites (VARCHAR(600))
  RETURNS TABLE (store_id CHAR (4), name VARCHAR (41), sales INTEGER)
  SPECIFIC favorites
  LANGUAGE OLEDB
  EXTERNAL NAME '!!Provider=SQLOLEDB.1;Persist Security Info=False;
  User ID=sa;Initial Catalog=pubs;Data Source=WALTZ;
  Locale Identifier=1033;Use Procedure for Prepare=1;
  Auto Translate=False;Packet Size=4096;Workstation ID=WALTZ;
  OLE DB Services=CLIENTCURSOR;';
```

```
SELECT *
FROM TABLE (favorites (' select top 3 sales.stor_id as store_id,  ' ||
                       '        stores.stor_name as name,         ' ||
                       '        sum(sales. qty) as sales          ' ||
                       ' from sales, stores                       ' ||
                       ' where sales.stor_id = stores.stor_id     ' ||
                       ' group by sales.stor_id, stores.stor_name ' ||
                       ' order by sum(sales.qty) desc')) as f;
```

标准行集名称

某些行集需要在 EXTERNAL NAME 子句中通过*标准名称* 来标识。

标准名称包括下列任一项或两项:

- 关联目录名称，该名称需要下列信息:
  - 提供程序是否支持目录名称
  - 目录名称在标准名称中的放置位置
  - 要使用的目录名称分隔符
- 关联模式名，该名称需要下列信息:
  - 提供程序是否支持模式名
  - 要使用的模式名分隔符

有关 OLE DB 提供者针对目录名称和模式名提供的支持的信息，请参阅 OLE DB 提供者的字面值信息的文档。

如果在提供程序的字面值信息中，DBLITERAL_CATALOG_NAME 不是 NULL，请使用目录名称并且将 DBLITERAL_CATALOG_SEPARATOR 的值用作分隔符。要确定目录名称是位于标准名称的开头还是末尾，请参阅 OLE DB 提供者的属性集 DBPROPSET_DATASOURCEINFO 中 DBPROP_CATALOGLOCATION 的值。

如果在提供程序的字面值信息中，DBLITERAL_SCHEMA_NAME 不是 NULL，请使用模式名并且将 DBLITERAL_SCHEMA_SEPARATOR 的值用作分隔符。

如果名称包含特殊字符或匹配关键字，请使用针对 OLE DB 提供者指定的引号字符将名称引起来。在 OLE DB 提供者的字面值信息中，将引号字符定义为 DBLITERAL_QUOTE_PREFIX 和 DBLITERAL_QUOTE_SUFFIX。例如，在下列 EXTERNAL NAME 中，指定的行集包含行集 *authors* 的目录名称 *pubs* 和模式名 *dbo*，且使用引号字符 " 将名称引起来。

```
EXTERNAL NAME '!"pubs"."dbo"."authors"!Provider=SQLOLEDB.1;...';
```

有关构造标准名称的更多信息，请参阅 *Microsoft OLE DB 2.0 Programmer's Reference and Data Access SDK*（Microsoft Press 1998 年）和 OLE DB 提供者的文档。

OLE DB 支持的 SQL 数据类型

下表显示 DB2 数据类型如何映射至 OLE DB 数据类型（要获取描述，请参阅 *Microsoft OLE DB 2.0 Programmer's Reference and Data Access SDK* Microsoft Press 1998 年）。使用映射表在 OLE DB 表函数中定义相应的 RETURNS TABLE 列。例如，如果您在 OLE DB 表函数中定义一个数据类型为 INTEGER 的列，那么 DB2 将从 OLE DB 提供者请求类型为 DBTYPE_I4 的数据。

对于 OLE DB 提供者源数据类型至 OLE DB 数据类型的映射，请参阅 OLE DB 提供者文档。要获取 ANSI SQL、Microsoft Access 以及 Microsoft SQL Server 提供程序如何将其各种数据类型映射至 OLE DB 数据类型的示例，请参阅 *Microsoft OLE DB 2.0 Programmer's Reference and Data Access SDK*，Microsoft Press 1998 年。

表 23. 将 DB2 数据类型映射至 OLE DB

DB2 数据类型	OLE DB 数据类型
SMALLINT	DBTYPE_I2
INTEGER	DBTYPE_I4
BIGINT	DBTYPE_I8
REAL	DBTYPE_R4
FLOAT/DOUBLE	DBTYPE_R8
DEC(p, s)	DBTYPE_NUMERIC(p, s)
DATE	DBTYPE_DBDATE
TIME	DBTYPE_DBTIME
TIMESTAMP	DBTYPE_DBTIMESTAMP
CHAR(N)	DBTYPE_STR
VARCHAR(N)	DBTYPE_STR
LONG VARCHAR	DBTYPE_STR
CLOB(N)	DBTYPE_STR
CHAR(N) FOR BIT DATA	DBTYPE_BYTES
VARCHAR(N) FOR BIT DATA	DBTYPE_BYTES
LONG VARCHAR FOR BIT DATA	DBTYPE_BYTES
BLOB(N)	DBTYPE_BYTES
GRAPHIC(N)	DBTYPE_WSTR
VARGRAPHIC(N)	DBTYPE_WSTR
LONG GRAPHIC	DBTYPE_WSTR
DBCLOB(N)	DBTYPE_WSTR

注：有关 OLE DB 数据类型转换规则的定义，请参阅 *Microsoft OLE DB 2.0 Programmer's Reference and Data Access SDK*，Microsoft Press 1998 年。例如：

- 数据可以转换为 OLE DB 数据类型 DBTYPE_NUMERIC(19,4)（映射至 DB2 数据类型 DEC(19,4)），以检索 OLE DB 数据类型 DBTYPE_CY。
- 数据可以转换为 OLE DB 数据类型 DBTYPE_I2（映射至 DB2 数据类型 SMALLINT），以检索 OLE DB 数据类型 DBTYPE_I1。

- 数据可以转换为 OLE DB 数据类型 DBTYPE_BYTES（映射至 DB2 数据类型 CHAR (12) FOR BIT DATA），以检索 OLE DB 数据类型 DBTYPE_GUID。

第 10 章 调用例程

在数据库中通过发出 CREATE 语句来开发和创建例程后，如果已向例程定义者和例程调用者授予相应的例程特权，那么可以调用例程。

每种例程的用途和使用方式各不相同。调用例程的先决条件是公共的，但是各自的调用实现不同。

例程调用的先决条件

- 必须已使用 CREATE 语句在数据库中创建例程。
- 对于外部例程，必须在 CREATE 语句的 EXTERNAL 子句所指定的位置中安装库或类文件，否则将发生错误 (SQLCODE SQL0444, SQLSTATE 42724)。
- 例程调用者必须对例程具有 EXECUTE 特权。如果未授权调用者执行例程，那么将发生错误 (SQLSTATE 42501)。

过程调用

通过执行对过程进行引用的 CALL 语句来调用过程。

CALL 语句支持过程调用、将参数传递至过程以及接收过程所返回的参数。在成功返回过程后，可以处理过程所返回的任何可访问结果集。

可以从支持 CALL 语句的任何位置调用过程，其中包括:

- 客户机应用程序
- 外部例程（过程、UDF 或方法）
- SQL 例程（过程、UDF 或方法）
- 触发器（前触发器、后触发器或替代触发器）
- 动态复合语句
- 命令行处理器 (CLP)

如果选择从客户机应用程序或外部例程调用过程，那么可以使用编写该过程的语言来编写客户机应用程序或外部例程。例如，使用 C++ 编写的客户机应用程序可以使用 CALL 语句来调用使用 Java 编写的过程。这将使程序员能够非常灵活地使用他们选择的语言进行编程，以及集成使用不同语言编写的代码段。

此外，可以在有别于过程所在操作系统的操作系统上执行用来调用该过程的客户机应用程序。例如，在 Windows 操作系统上运行的客户机应用程序可以使用 CALL 语句来调用位于 Linux 数据库服务器上的过程。

根据从中调用过程的位置，可能存在其他注意事项。

函数调用

预期在 SQL 语句中引用函数。

SQL 语句中，只要是在允许使用表达式的位置，就可以引用内置函数、有源聚集函数以及用户定义标量函数。例如，在查询的查询列表中，或在 INSERT 语句的 VALUES 子

句中。只能在 FROM 子句中引用表函数。例如，在查询或数据更改语句的 FROM 子句中。

方法调用

方法类似于标量函数，只是使用方法来指定结构化类型的行为。方法调用与用户定义标量函数调用相同，只是方法的其中一个参数必须是方法可以操作的结构化类型。

例程调用相关任务

要调用特定类型的例程，请执行下列操作:

- 第 270 页的『从应用程序或外部例程中调用过程』
- 第 271 页的『从触发器或 SQL 例程调用过程』
- 请参阅*Call Level Interface Guide and Reference Volume 1* 中的『从 CLI 应用程序调用过程』
- 第 273 页的『从命令行处理器 (CLP) 调用过程』
- 第 284 页的『调用标量函数或方法』
- 第 285 页的『调用用户定义的表函数』

对包含 SQL 的例程进行授权和绑定

讨论例程级别权限时，必须定义一些与例程相关的角色、确定角色以及定义与这些角色相关的特权:

程序包所有者

参与例程实现的特定程序包的所有者。程序包所有者是执行 **BIND** 命令以将程序包与数据库绑定在一起的用户，除非使用 **OWNER PRECOMPILE** 或 **BIND** 命令参数覆盖程序包所有者资格并将其设置为另一用户。执行 **BIND** 命令时，程序包所有者被授予对该程序包的 EXECUTE WITH GRANT 特权。例程库或可执行文件可以包含多个程序包，因此可以具有多个相关联的程序包所有者。

例程定义者

发出 CREATE 语句以注册例程的标识。例程定义者一般是 DBA，但通常也是例程程序包所有者。在调用例程期间装入程序包时，会根据定义者对例程的相关联程序包的执行权限（而不是根据例程调用程序的权限）来检查例程的运行权限。要成功调用例程，例程定义者必须具有下列其中一项特权或权限:

- 例程程序包的 EXECUTE 特权和例程的 EXECUTE 特权
- DATAACCESS 权限

如果例程定义者和例程程序包所有者是同一用户，那么例程定义者将具有程序包的必需 EXECUTE 特权。如果定义者不是程序包所有者，那么必须由具有程序包的 ACCESSCTRL 或 SECADM 权限、CONTROL 或 EXECUTE WITH GRANT OPTION 特权的用户，将程序包的 EXECUTE 特权显式地授予给定义者。（程序包的创建者会自动接收程序包的 CONTROL 和 EXECUTE WITH GRANT OPTION。）

在发出可以注册例程的 CREATE 语句时，会隐式地将例程的 EXECUTE WITH GRANT OPTION 特权授予给定义者。

例程定义者的角色是在一个授权标识下封装例程的相关联程序包的运行特权，以及将例程的 EXECUTE 特权授予给 PUBLIC 用户或特定用户（需要调用例程）的特权。

注: 对于 SQL 例程，例程定义者也就是程序包所有者。因此，定义者在执行例程的 CREATE 语句时，将具有例程和例程程序包的 EXECUTE WITH GRANT OPTION。

例程调用者

调用例程的标识。要确定将成为例程调用者的用户，必须考虑如何调用例程。可以从命令窗口或从嵌入式 SQL 应用程序调用例程。如果是方法和 UDF，那么会将例程引用嵌入另一个 SQL 语句。使用 CALL 语句来调用过程。对于应用程序中的动态 SQL，调用者是包含例程调用的直接较高级别例程或应用程序的运行时授权标识（但是，此标识也取决于用来绑定较高级别例程或应用程序的 **DYNAMICRULES** 选项）。对于静态 SQL，调用者是包含例程引用的程序包的 **OWNER PRECOMPILE** 或 **BIND** 命令参数的值。这些用户将需要例程的 EXECUTE 特权，才能成功调用例程。任何具有例程 EXECUTE WITH GRANT OPTION 特权（这包括例程定义者，除非已显式地撤销该特权）、ACCESSCTRL 或 SECADM 权限的用户，都可以通过显式地发出 GRANT 语句来授予此特权。

例如，如果使用 **DYNAMICRULES** BIND 来绑定与包含动态 SQL 的应用程序相关联的程序包，那么程序包的运行时授权标识将是程序包所有者（而不是调用程序包的用户）。此外，程序包所有者还将成为实际绑定者或 **OWNER PRECOMPILE** 或 **BIND** 命令参数的值。在此情况下，例程的调用者会采用此值（而不是执行应用程序的用户标识）。

注:

1. 对于例程中的静态 SQL，程序包所有者的特权必须足以执行例程主体中的 SQL 语句。如果存在例程的嵌套引用，那么这些 SQL 语句可能需要表访问特权或执行特权。
2. 对于例程中的动态 SQL，由例程主体的 **BIND** 的 **DYNAMICRULES** 选项控制将进行特权验证的用户标识。
3. 例程程序包所有者必须将程序包的 EXECUTE 特权授予给例程定义者。可以在注册例程之前或之后完成此操作，但必须在调用例程之前完成，否则将返回错误 (SQLSTATE 42051)。

管理例程的执行特权所涉及的步骤，会在后续图和文本中加以详细说明:

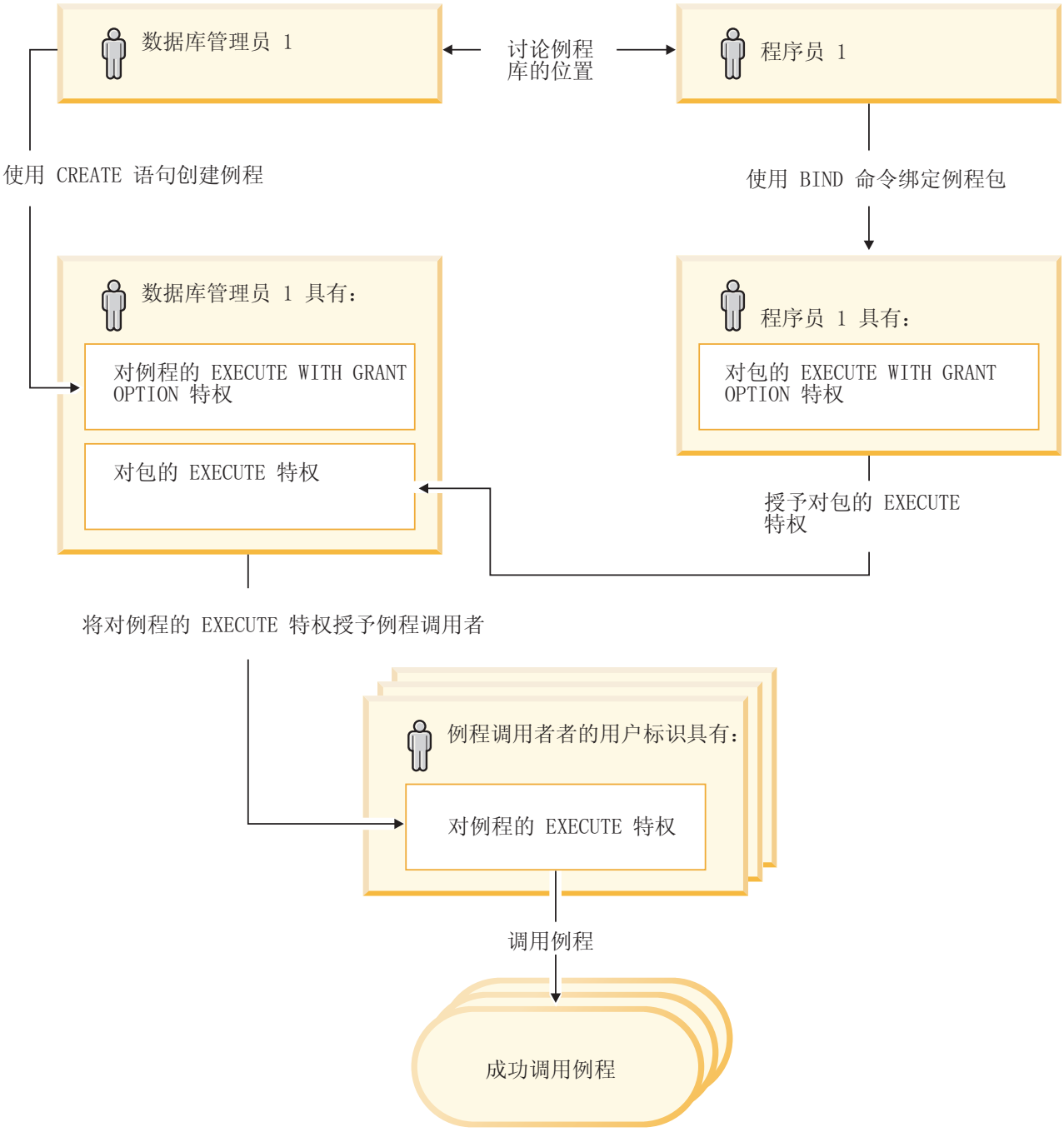


图 3. 管理例程的 EXECUTE 特权

1. 定义者执行相应的 CREATE 语句以注册例程。这将以期望的 SQL 访问级别在 DB2 数据库系统中注册例程，建立例程特征符，此外还指向例程可执行文件。如果定义者并非同时为程序包所有者，那么定义者需要与例程程序的程序包所有者和作者沟通，以清楚了解例程库所在的位置，这样就可以在 CREATE 语句的 EXTERNAL 子句中正确地指定此位置。由于成功执行 CREATE 语句，因此定义者具有例程的 EXECUTE WITH GRANT 特权，但是定义者尚不具有例程程序包的 EXECUTE 特权。
2. 定义者必须将例程的 EXECUTE 特权授予给任何有权使用例程的用户。（如果此例程的程序包将递归地调用此例程，那么必须在下一步之前完成此步骤。）

3. 程序包所有者预编译和绑定例程程序，或让其他用户代表他们完成这些操作。在成功预编译和绑定后，会将各个程序包的 EXECUTE WITH GRANT OPTION 特权隐式地授予给程序包所有者。此步骤遵循此列表中的第 1 步，只涵盖例程中 SQL 递归的可能性。如果任何特定情况下都不存在此类递归，那么预编译/绑定可以在针对例程发出 CREATE 语句之前执行。
4. 每个程序包所有者都必须将各个例程程序包的 EXECUTE 特权授予给例程的定义者。在完成上一步之后，必须等一段时间，再执行此步骤。如果程序包所有者也是例程定义者，那么可以跳过此步骤。
5. 例程的静态用法：必须确实将例程的 EXECUTE 特权，授予给引用该例程的程序包绑定所有者，因此必须在此刻完成上一步。当例程执行时，DB2 数据库系统会验证定义者是否对任何所需的程序包具有 EXECUTE 特权，因此必须针对每个这样的程序包完成第 3 步。
6. 例程的动态用法：授权标识（由调用应用程序的 **DYNAMICRULES** 选项控制）必须对例程具有 EXECUTE 特权（第 4 步），而例程的定义者必须对程序包具有 EXECUTE 特权（第 3 步）。

例程名称和路径

存储过程或 UDF 的限定名是 schema-name.routine-name。每当引用存储过程或 UDF 时，都可以使用此限定名。方法的限定名是 schema-name.type.method-name。

例如:

```
SANDRA.BOAT_COMPARE    SMITH.FOO    SYSIBM.SUBSTR    SYSFUN.FLOOR
```

但是，还可以省略 schema-name.，在此情况下，DB2 数据库系统将尝试标识您所引用的存储过程或 UDF。例如:

```
BOAT_COMPARE    FOO    SUBSTR    FLOOR
```

在您未使用 schema-name 时，*SQL 路径*这一概念对于 DB2 数据库系统解析*未限定*的引用来说非常重要。SQL 路径是模式名的有序列表。它提供了一组模式来解析对存储过程、UDF 和类型进行的未限定引用。如果一个引用与路径中的多个模式中的存储过程、类型或 UDF 匹配，那么将通过模式在路径中的顺序来解析此匹配。对于静态 SQL 而言，SQL 路径通过预编译和绑定命令的 **FUNCPATH** 选项确定。对于动态 SQL 而言，SQL 路径由 SET PATH 语句设置。SQL 路径的缺省值如下:

```
"SYSIBM","SYSFUN","SYSPROC", "ID"
```

这既适用于静态 SQL 也适用于动态 SQL，其中 *ID* 表示当前语句授权标识。

例程名可以*重载*，这意味着，多个例程（即使同一模式中的例程）可以同名。多个同名的函数或方法可以具有相同数目的参数，但数据类型不能相同。对于存储过程而言，情况并非如此；即，多个同名的存储过程必须具有不同数目的参数。不同例程类型的实例不会相互重载，但方法除外，它们能够重载函数。要使一个方法重载函数，必须使用 WITH FUNCTION ACCESS 子句来注册该方法。

函数、存储过程和方法可以具有完全相同的*特征符*并在同一个模式中，而不会相互重载。在例程的上下文中，特征符是限定的例程名与所有参数的已定义数据类型（按参数的定义顺序排列）的并置。

方法将针对其相关联的结构化类型的实例进行调用。创建子类型时，除继承属性以外，它还将继承为超类型定义的方法。因此，对于超类型，还可以针对其子类型的任何实例运行该超类型的方法。定义子类型时，可以*覆盖*超类型的方法。覆盖方法意味着专门为给定的子类型重新实现该方法。这使您能够方便地动态分派方法（也称为多态性），即，应用程序将根据结构化类型实例的类型（例如，该方法在结构化类型层次结构中所处的位置）来执行最具体的方法。

每种例程类型都有自己的选择算法，该算法将考虑重载事实（对于方法而言，将考虑覆盖事实）和 SQL 路径，以便选择每个例程引用的最佳匹配项。

嵌套的例程调用

在例程的上下文中，*嵌套*是指一个例程调用另一例程的情况。

即，一个例程发出的 SQL 可以引用另一个例程，后者可以发出再次引用另一例程的 SQL，依此类推。如果所引用的例程序列包含先前已引用的例程，那么称为*递归*嵌套情况。

您可以在 DB2 例程中使用嵌套和递归，但存在下列限制:

64 层嵌套

嵌套例程调用时，深度可达 64 层。请考虑例程 A 调用例程 B，而例程 B 调用例程 C 这种情况。在此示例中，例程 C 的执行在嵌套层 3 进行。还有可能进行另外的 61 层嵌套。

其他限制

例程不能调用使用较高 SQL 数据访问级别进行编目的目标例程。例如，使用 CONTAINS SQL 子句创建的 UDF 可以调用使用 CONTAINS SQL 子句或 NO SQL 子句创建的存储过程。但是，此例程不能调用使用 READS SQL DATA 子句或 MODIFIES SQL DATA 子句创建的存储过程 (SQLCODE -577, SQLSTATE 38002)。这是因为，调用者的 SQL 级别不允许执行任何读取或修改操作（所调用的例程将继承此 SQL 级别）。

嵌套例程时的另一项限制是，对表的访问受限制，以避免例程之间发生读写操作冲突。

在 64 位数据库服务器上调用 32 位例程

在 64 位 DB2 实例中，可以调用那些引用 32 位外部例程库的 C 和 COBOL 例程，但是，必须将这些例程指定为以受防护并且不具有线程安全性的方式运行。

关于此任务

要完成此任务，请在创建新例程时在例程 CREATE 语句中同时指定 FENCED 子句和 NOT THREADSAFE 子句。对于已在 64 位实例中创建的例程而言，可以使用 ALTER FUNCTION 或 ALTER PROCEDURE 语句来修改例程定义。第一次在 64 位环境中调用此类 32 位例程时，性能将不如人意。对 32 位存储过程执行后续调用时，性能将与等同的 64 位例程相当。建议您不要在 64 位 DB2 实例中使用 32 位例程。

要成功地在 64 位 DB2 实例中对 64 位数据库服务器调用 Java 过程，需要 64 位的 Java 虚拟机（JVM）。在 64 位 DB2 实例中，不支持使用 32 位 JVM 来运行例程。

由于 Java 类独立于平台，因此，使用 32 位软件开发套件编译的 Java 类可以成功地通过 64 位 JVM 运行。这样做并不会影响例程性能。

过程

要在 64 位服务器上调用现有的 32 位例程，请执行下列操作:

1. 将例程类或库复制到数据库例程目录:
   - UNIX: `sqllib/function`
   - Windows: `sqllib\function`
2. 使用 CREATE PROCEDURE 语句来注册存储过程。
3. 使用 CALL 语句来调用存储过程。

对过程的引用

请使用 CALL 语句来调用存储过程，并且，在引用存储过程时，请先指定限定名（模式名和存储过程名），然后指定括在圆括号内的自变量列表。另外，还可以在不指定模式名的情况下调用存储过程，这将导致在不同模式中选择具有相同参数数目的可能存储过程。

传递给存储过程的每个参数都可以是主变量、参数标记、表达式或 NULL。有关存储过程参数的限制如下所示:

- OUT 和 INOUT 参数必须是主变量。
- 除非 SQL 数据类型映射到 Java 类类型，否则不能将 NULL 传递给 Java 存储过程。
- 不能将 NULL 传递给 PARAMETER STYLE GENERAL 存储过程。

自变量的位置至关重要，并且必须与存储过程定义一致才能使语义正确。自变量位置以及存储过程定义都必须与存储过程体本身一致。DB2 数据库系统并不会尝试调整自变量位置以便更好地与存储过程定义匹配，DB2 数据库系统也无法理解各个存储过程参数的语义。

调用过程

完成创建过程（也称为存储过程）所需执行的活动之后，就可以使用 CALL 语句来调用过程。CALL 语句是一个 SQL 语句，它允许调用过程、将参数传递给该过程以及接收该过程返回的参数。

关于此任务

在成功返回过程后，可以处理过程所返回的任何可访问结果集。可以从支持 CALL 语句的任何位置调用过程，其中包括:

- 嵌入式 SQL 客户机应用程序
- 外部例程（过程、UDF 或方法）
- SQL 例程（过程、UDF 或方法）
- SQL 触发器（BEFORE TRIGGER，AFTER TRIGGER 或 INSTEAD OF TRIGGER）
- SQL 动态复合语句
- 从命令行处理器 (CLP) 中

如果选择从客户机应用程序或外部例程调用过程，那么可以使用编写该过程的语言来编写客户机应用程序或外部例程。例如，使用 C++ 编写的客户机应用程序可以使用 CALL 语句来调用使用 Java 编写的过程。这将使程序员能够非常灵活地使用他们选择的语言进行编程，以及集成使用不同语言编写的代码段。

此外，可以在有别于过程所在平台的平台上执行用来调用该过程的客户机应用程序。例如，在 Windows 操作系统上运行的客户机应用程序可以使用 CALL 语句来调用位于 Linux 数据库服务器上的过程。

自主过程是这样的一个过程：被调用时，它在新事务中以独立于原始事务的方式执行。自主过程成功完成后，它将落实该过程中执行的工作，但如果不成功，那么该过程将回滚它所执行的任何工作。无论自主过程的结果如何，调用自主过程的事务都不受影响。要将一个过程指定为自主过程，请在 CREATE PROCEDURE 语句中指定 AUTONOMOUS 关键字。

调用过程时，将应用某些有关具体选择哪个过程的规则。过程选择部分取决于您通过指定模式名对过程进行限定。DB2 数据库管理器还根据调用过程时指定的自变量数量和任何自变量名称来执行检查。有关过程选择的详细信息，请参阅有关 CALL 语句的信息。

从应用程序或外部例程中调用过程

通过在应用程序中进行一些简单的设置工作并使用 CALL 语句，可以方便地从客户机应用程序或者与外部例程相关联的应用程序中调用封装了逻辑的过程（也称为存储过程）。

开始之前

必须已通过执行 CREATE PROCEDURE 语句在数据库中创建过程。

对于外部过程而言，库或类文件必须驻留在 CREATE PROCEDURE 语句中的 EXTERNAL 子句所指定的位置。

过程调用者必须具有执行 CALL 语句所需的特权。在此情况下，过程调用者是执行应用程序的用户标识，但是，如果对该应用程序使用了 DYNAMICRULES 绑定选项，那么有一些特殊规则适用。

过程

如果要让应用程序调用过程，那么该应用程序必须包含一些特定的元素。在编写应用程序时，您必须完成下列任务:

1. 声明、分配和初始化 CALL 语句所需的可选数据结构、主变量或参数标记的存储器。要完成此任务，请执行下列操作:
   - 指定要用于每个过程参数的主变量或参数标记。
   - 初始化与 IN 或 INOUT 参数相对应的主变量或参数标记。
2. 建立数据库连接。要完成此任务，请执行嵌入式 SQL 语言的 CONNECT TO 语句或者编码隐式的数据库连接。
3. 编码过程调用。在数据库连接代码之后，可以编码过程调用。要完成此任务，请执行 SQL 语言的 CALL 语句。务必对该过程所需的每个 IN、INOUT 和 OUT 参数指定主变量、常量或参数标记。

4. 添加代码，以处理 OUT 和 INOUT 参数以及结果集。此代码必须位于 CALL 语句执行之后。
5. 编码数据库 COMMIT 或 ROLLBACK。在执行 CALL 语句并对该过程返回的输出参数值或数据进行求值之后，您可能想让应用程序落实或回滚该事务。要完成此任务，请指定 COMMIT 或 ROLLBACK 语句。过程可以包含 COMMIT 或 ROLLBACK 语句，但是，建议您在客户机应用程序中进行事务管理。

   注: 与数据库建立 2 类连接的应用程序中调用的过程不能发出 COMMIT 或 ROLLBACK 语句。
6. 与数据库断开连接。
7. 准备、编译、链接并绑定应用程序。如果该应用程序用于外部例程，请发出 CREATE 语句以创建该例程并将外部代码库放在操作系统的适当函数路径中，以使数据库管理器能够找到该库。
8. 运行该应用程序或者调用外部例程。该应用程序中嵌入的 CALL 语句将被调用。

结果

注: 您可以在步骤 2 到 5 之间的任何位置编码 SQL 语句和例程逻辑。

从触发器或 SQL 例程调用过程

从 SQL 例程、触发器或动态复合语句调用过程基本上相同。使用相同的步骤来实现此调用。本主题说明关于使用触发器方案的步骤。从例程或动态复合语句调用过程的任何不同先决条件或步骤都会加以陈述。

开始之前

- 必须已通过执行 CREATE PROCEDURE 语句在数据库中创建过程。
- 对于外部过程，库或类文件必须位于 CREATE PROCEDURE 语句的 EXTERNAL 子句所指定的位置。
- 包含 CALL 语句的触发器的创建者必须具有执行 CALL 语句的特权。在运行时，如果已激活触发器，那么会检查触发器创建者的权限，以确定是否有执行 CALL 语句的特权。如果用户执行包含 CALL 语句的动态复合语句，那么必须具有针对该过程执行 CALL 语句的特权。
- 要调用触发器，用户必须具有执行与触发器事件相关联的数据更改语句的特权。类似地，要成功地调用 SQL 例程或动态复合语句，用户必须对该例程具有 EXECUTE 特权。

限制

从 SQL 触发器、SQL 例程或动态复合语句中调用过程时，会适用下列限制:

- 在分区型数据库环境中，无法从触发器或 SQL UDF 调用过程。
- 在对称多处理器（SMP）机器上，可以在单一处理器上从触发器执行过程调用。
- 要从触发器调用的过程不能包含 COMMIT 语句或试图回滚工作单元的 ROLLBACK 语句。过程支持 ROLLBACK TO SAVEPOINT 语句，但是，指定的保存点必须是在该过程中。
- 从触发器中回滚 CALL 语句将不会回滚任何受过程影响的外部操作，例如，写入文件系统。

- 过程不能修改任何已联合的表。这意味着过程不能包含昵称的搜索式 UPDATE、昵称的搜索式 DELETE 或对昵称执行的 INSERT。
- 将无法从直接插入 SQL PL 语句访问对过程指定的结果集。
- 如果在执行编译触发器时打开定义为 **WITH RETURN TO CLIENT** 的游标，那么将丢弃来自该游标的结果集。

如果 BEFORE 触发器包含的 CALL 语句会引用一个使用访问级别 MODIFIES SQL DATA 来创建的过程，那么无法创建这些 BEFORE 触发器。对此类触发器执行 CREATE TRIGGER 语句会失败并且生成错误 (SQLSTATE 42987)。有关例程中 SQL 访问级别的更多信息，请参阅:

- 第 35 页的『例程中的 SQL 访问级别』
- 第 30 页的『可以在例程和触发器中执行的 SQL 语句』

过程

此过程部分说明如何创建和调用包含 CALL 语句的触发器。从触发器调用过程所需的 SQL 就是从 SQL 例程或动态复合语句调用过程所需的 SQL。

1. 编写一个指定预期触发器属性的基本 CREATE TRIGGER 语句。请参阅 CREATE TRIGGER 语句。
2. 在触发器的触发器操作部分，您可以声明过程所指定的任何 IN、INOUT 以及 OUT 参数的 SQL 变量。请参阅 DECLARE 语句。要了解如何初始化或设置这些变量，请参阅赋值语句。此外，还可以将触发器转换变量用作过程的参数。
3. 在触发器的触发器操作部分，为过程添加 CALL 语句。为过程的每个 IN、INOUT 以及 OUT 参数指定值或表达式。
4. 对于 SQL 过程，您可以选择性地使用 GET DIAGNOSTICS 语句来捕获过程的返回状态。要执行此操作，您将需要使用整数类型变量来保存返回状态。紧随 CALL 语句之后，只需添加一个 GET DIAGNOSTICS 语句以将 RETURN_STATUS 指定给局部触发器返回状态变量。
5. 在完成编写 CREATE TRIGGER 语句之后，您现在可以静态地（从应用程序中）或动态地（从 CLP）执行该语句以正式地在数据库中创建触发器。
6. 调用触发器。要执行此操作，请针对您的触发器事件所对应的相应数据更改语句执行。
7. 针对表执行数据更改语句时，会触发针对该表定义的相应触发器。执行触发器操作时，会执行该操作中所包含的 SQL 语句（其中包括 CALL 语句）。

结果

如果过程试图读或写触发器也会读或写的表，可能会发生运行时错误，那么当检测到读/写冲突时，可能会抛出错误。必须从过程所修改的表中排除触发器所修改的表集（其中包括针对其定义了该触发器的表）。

示例: 从触发器调用 SQL 过程

此示例说明如何嵌入 CALL 语句以在触发器中调用过程，以及如何使用 GET DIAGNOSTICS 语句来捕获过程调用的返回状态。下列 SQL 语句会创建一些必要的表、一个 SQL PL 语言过程和一个后触发器。

```
CREATE TABLE T1 (c1 INT, c2 CHAR(2))@
CREATE TABLE T2 (c1 INT, c2 CHAR(2))@

CREATE PROCEDURE proc(IN val INT, IN name CHAR(2))
LANGUAGE SQL
DYNAMIC RESULTSETS 0
MODIFIES SQL DATA
BEGIN
  DECLARE rc INT DEFAULT 0;
  INSERT INTO TABLE T2 VALUES (val, name);
  GET DIAGNOSTICS rc = ROW_COUNT;
  IF ( rc > 0 ) THEN
      RETURN 0;
  ELSE
      RETURN -200;
  END IF;
END@

CREATE TRIGGER trig1 AFTER UPDATE ON t1
REFERENCING NEW AS n
FOR EACH ROW
WHEN (n.c1 > 100);
BEGIN ATOMIC
   DECLARE rs INTEGER DEFAULT 0;
   CALL proc(n.c1, n.c2);
   GET DIAGNOSTICS rs = RETURN_STATUS;
   VALUES(CASE WHEN rc < 0 THEN RAISE_ERROR('70001', 'PROC CALL failed'));
END@
```

发出下列 SQL 语句将促使触发器触发并且将调用过程。

```
UPDATE T1 SET c1 = c1+1 WHERE c2 = 'CA'@
```

从命令行处理器 (CLP) 调用过程

您可以从 DB2 命令行处理器界面，使用 CALL 语句来调用存储过程。必须在 DB2 数据库系统目录表中定义所调用的存储过程。

过程

要调用存储过程，请先连接至数据库:

```
db2 connect to sample user 用户标识 using 密码
```

其中，*用户标识* 和 *密码* 是 sample 数据库所在的实例的用户标识和密码。
要使用 CALL 语句，请输入存储过程名称和任何 IN 或 INOUT 参数值以及“?”（作为每个 OUT 参数值的占位符）。
在程序源文件的存储过程的 CREATE PROCEDURE 语句中给出了存储过程的参数。

示例

SQL 过程示例

示例 1。

在 whiles.db2 文件中，DEPT_MEDIAN 过程特征符的 CREATE PROCEDURE 语句如下所示:

```
CREATE PROCEDURE DEPT_MEDIAN
(IN deptNumber SMALLINT, OUT medianSalary DOUBLE)
```

要调用此过程，请使用 CALL 语句（必须在其中指定过程名称和相应的参数自变量，在此情况下，是 IN 参数的值和表示 OUT 参数值的问号“?”）。过程的

SELECT 语句在 STAFF 表的 DEPT 列上使用 deptNumber 值，因此，IN 参数必须是来自 DEPT 列的有效值（例如，值“51”）才能获取有意义的输出:

```
db2 call dept_median (51, ?)
```

注: 在 UNIX 操作系统上，括号对于命令 shell 具有特殊意义，因此必须在其前面加上“\”字符或使用引号将其引起来，如下所示:

```
db2 "call dept_median (51, ?)"
```

如果使用的是命令行处理器的交互方式，那么不要使用引号。
运行此命令后，您应该收到以下结果:

```
Value of output parameters
--------------------------
Parameter Name  : MEDIANSALARY
Parameter Value : +1.76545000000000E+004

Return Status = 0
```

示例 2。

此示例说明如何使用数组参数来调用过程。类型 phonenumbers 如下定义:

```
CREATE TYPE phonenumbers AS VARCHAR(12) ARRAY[1000]
```

以下示例中定义的过程 find_customers 具有类型为 phonenumbers 的 IN 和 OUT 参数。该过程会在 numbers_in 中搜索以给定 area_code 开头的数字，然后在 numbers_out 中报告这些数字。

```
CREATE PROCEDURE find_customers(
IN numbers_in phonenumbers,
IN area_code CHAR(3),
OUT numbers_out phonenumbers)
BEGIN
DECLARE i, j, max INTEGER;

SET i = 1;
SET j = 1;
SET numbers_out = NULL;
SET max = CARDINALITY(numbers_in);

WHILE i <= max DO
IF substr(numbers_in[i], 1, 3) = area_code THEN
SET numbers_out[j] = numbers_in[i];
SET j = j + 1;
END IF;
SET i = i + 1;
END WHILE;
END
```

要调用该过程，您可以使用下列 CALL 语句:

```
db2 CALL find_customers(ARRAY['416-305-3745',
                              '905-414-4565',
                              '416-305-3746'],
                              '416',
                               ?)
```

如 CALL 语句中所示，当过程具有数组数据类型的输入参数时，可以使用包含文字值列表的数组构造函数来指定输入自变量。

在运行该命令后，您应该会接收到类似如下的结果:

```
Value of output parameters
  --------------------------
  Parameter Name  : OUT_PHONENUMBERS
  Parameter Value : ['416-305-3745',
                     '416-305-3746']

  Return Status = 0
```

C 存储过程示例

您还可以使用命令行处理器来调用从支持的主语言创建的存储过程。在 samples/c 目录 (UNIX) 和 samples\c 目录 (Windows) 中，DB2 数据库系统提供了用于创建存储过程的文件。spserver 共享库包含一些可根据源文件 spserver.sqc 来创建的存储过程。spcreate.db2 文件对存储过程进行编目。

在 spcreate.db2 文件中，MAIN_EXAMPLE 过程的 CREATE PROCEDURE 语句以下列内容开头:

```
CREATE PROCEDURE MAIN_EXAMPLE (IN job CHAR(8),
                               OUT salary DOUBLE,
                               OUT errorcode INTEGER)
```

要调用此存储过程，您需要放入 IN 参数的 CHAR 值 job 以及表示每个 OUT 参数的问号“?”。过程的 SELECT 语句在 EMPLOYEE 表的 JOB 列上使用 job 值，因此，IN 参数必须是来自 JOB 列的有效值才能获取有意义的输出: 调用该存储过程的 C 样本程序 spclient 会将 'DESIGNER' 用作 JOB 值。我们可以执行相同的操作，如下所示:

```
db2 "call MAIN_EXAMPLE ('DESIGNER', ?, ?)"
```

运行此命令后，您应该收到以下结果:

```
Value of output parameters
--------------------------
Parameter Name  : SALARY
Parameter Value : +2.37312500000000E+004

Parameter Name  : ERRORCODE
Parameter Value : 0

Return Status = 0
```

ERRORCODE 为零时指示成功结果。

从 CLI 应用程序中调用存储过程

CLI 应用程序通过执行 CALL 过程 SQL 语句来调用存储过程。本主题描述如何从 CLI 应用程序中调用存储过程。

开始之前

在调用存储过程之前，请确保您已初始化 CLI 应用程序。

关于此任务

如果未对所调用的存储过程进行编目，请确保它未调用任何 CLI 模式函数。不支持从未编目的存储过程中调用 CLI 模式函数。

CLI 模式函数包括: SQLColumns()、SQLColumnPrivileges()、SQLForeignKeys()、SQLPrimaryKeys()、SQLProcedureColumns()、SQLProcedures()、

SQLSpecialColumns()、SQLStatistics()、SQLTables() 和 SQLTablePrivileges()。

过程

要调用存储过程，请执行下列操作:

1. 声明与存储过程的每个 IN、INOUT 和 OUT 参数相对应的应用程序主变量。 确保应用程序变量的数据类型和长度与存储过程特征符中自变量的数据类型和自变量匹配。CLI 支持调用将所有 SQL 类型作为参数标记的存储过程。
2. 初始化 IN、INOUT 和 OUT 参数的应用程序变量。
3. 发出 SQL 语句 CALL。 例如:

   ```
   SQLCHAR *stmt = (SQLCHAR *)"CALL OUT_LANGUAGE (?)";
   ```

   或

   ```
   SQLCHAR *stmt = (SQLCHAR *)"CALL OUT_LANGUAGE (:language)";
   ```

   切记: 要使用指定的参数标记（例如 `:language`），必须通过将 **EnableNamedParameterSupport** 配置关键字设置为 TRUE 来显式地允许处理指定的阐述标记。

   为了获得最佳性能，应用程序应该对 CALL 过程字符串中的存储过程自变量使用参数标记，然后将主变量与那些参数标记绑定。但是，如果必须将入站存储过程自变量指定为字符串文字而不是参数标记，请在 CALL 过程语句中包括 ODBC 调用转义子句限定符 { }。例如:

   ```
   SQLCHAR *stmt = (SQLCHAR *)"{CALL IN_PARAM (123, 'Hello World!')}";
   ```

   在 CALL 过程语句中使用字符串文字和 ODBC 转义子句时，只能将字符串文字指定为 IN 方式存储过程自变量。INOUT 和 OUT 方式存储过程自变量必须仍使用参数标记进行指定。
4. 可选: 通过调用 SQLPrepare() 来准备 CALL 语句。
5. 通过调用 SQLBindParameter() 来绑定 CALL 过程语句的每个参数。

   注: 确保正确地绑定每个参数（与 SQL_PARAM_INPUT、SQL_PARAM_OUTPUT 或 SQL_PARAM_INPUT_OUTPUT 绑定），否则执行 CALL 过程语句时可能会发生意外的结果。例如，如果错误地将输入参数与 *InputOutputType* 值 SQL_PARAM_OUTPUT 绑定，那么将发生这种情况。

   注: CALL 过程语句不支持使用 SQL_ATTR_PARAMSET_SIZE 属性的参数标记的数组输入。
6. 使用 SQLExecDirect() 来执行 CALL 过程语句，或者，如果已在步骤 4 中准备 CALL 过程语句，请使用 SQLExecute()。

   注: 如果调用了存储过程的应用程序或线程在该存储过程完成前终止，那么该存储过程的执行也将终止。重要的是，存储过程应该包含逻辑以确保数据库在该存储过程提前终止时处于一致且适当的状态。
7. 在返回函数后检查 SQLExecDirect() 或 SQLExecute() 的返回码，以确定执行 CALL 过程语句或存储过程期间是否发生了任何错误。 如果返回码为 SQL_SUCCESS_WITH_INFO 或 SQL_ERROR，请使用 CLI 诊断函数 SQLGetDiagRec() 和 SQLGetDiagField() 来确定出错原因。

如果存储过程执行成功，那么任何作为 OUT 参数绑定的变量都可能包含该存储过程传递回给 CLI 应用程序的数据。如果适用的话，存储过程还可能通过不可滚动的游标返回一个或多个结果集。CLI 应用程序应该像处理通过执行 SELECT 语句生成的结果集一样处理存储过程结果集。

注: 如果 CLI 应用程序不确定存储过程返回的结果集的列类型的数目，可在此结果集上调用 SQLNumResultCols()、SQLDescribeCol()、和 SQLColAttribute() 函数（按此顺序依次进行）以确定此信息。

结果

执行 CALL 语句后，可以检索存储过程所返回的结果集（如果适用的话）。

注:

在返回给 DB2 CLI 应用程序的过程结果集中，如果 DATETYPE 数据类型值未以 ISO 格式返回，那么它的数字月份和日期部分将以逆序出现。例如，使用本地格式时，就是这种情况。要确保客户机应用程序正确地解释 DATETYPE 数据类型值信息，应该将过程与采用独立于语言环境的 DATETIME 格式（例如 ISO）的数据库绑定。例如:

```
db2set DB2_SQLROUTINE_PREPOPTS="DATETIME ISO"
```

注:

CLI 程序包将在数据库创建或升级时自动与数据库绑定。

匿名块的结果集

从 V9.7 FP2 开始，每当应用程序发送以 BEGIN（而不是 BEGIN COMPOUND）开始的 SQL 语句时，将为结果集准备 CLI。 CLI 将解释从服务器返回的游标，并允许应用程序检索结果集。

示例 1: 使用 SQLExecDirect

```
opt caller on
opt echo on

quickc 1 1 sample

SQLAllocStmt 1 1
getmem 1 1 SQL_C_LONG

SQLExecDirect 1 "drop table t1" -3
SQLExecDirect 1 "create table t1 (c1 int)" -3
SQLExecDirect 1 "insert into t1 values (10)" -3
SQLExecDirect 1 "insert into t1 values (20)" -3
SQLExecDirect 1 "insert into t1 values (30)" -3

SQLExecDirect 1 "begin declare c1 cursor with return to client with hold
for select c1 from t1; end" -3
SQLBindCol 1 1 sql_c_long 1
FetchAll 1

SQLFreeStmt 1 SQL_DROP
SQLTransact 1 1 SQL_COMMIT

killenv 1
```

过程结果集

除了交换参数之外，过程还可以通过返回结果集将信息传递至调用者。

SQL 主体例程以及使用下列接口编程的例程和应用程序可以接受结果集:

- CLI
- JDBC
- SQLJ

- ODBC

存储过程通过游标将结果集传递至其调用者。过程体必须包含您需要返回的每个结果集的游标。当您在过程中从结果集游标访存行时，只将未访存的行作为结果集传递至调用者。退出过程时，请保持对应于结果集的游标打开。会以您打开结果集游标的顺序返回多个结果集。

声明结果集的游标时，强烈建议您在 DECLARE CURSOR 语句的 WITH RETURN TO 子句中指定目标（这对于 SQL 过程是必需的）。要将结果集返回给调用者（不论调用者是应用程序还是例程），请指定 WITH RETURN TO CALLER。要将结果集直接返回给应用程序（绕过任何中间嵌套例程），请指定 WITH RETURN TO CLIENT。在外部例程中，游标会缺省定义为 WITH RETURN TO CALLER，除非它们已显式地定义为 WITH RETURN TO CLIENT。

在 CREATE PROCEDURE 语句中注册过程时，请使用 DYNAMIC RESULT SETS 子句来指示该过程返回的结果集数。此值在 SYSCAT.ROUTINES 视图的 RESULT_SETS 列中。如果从过程返回的结果集数有别于在 CREATE PROCEDURE 语句中指定的结果集数，那么会发出警告 (SQLCODE +464, SQLSTATE 0100E)。对于 PARAMETER STYLE JAVA 存储过程，CREATE PROCEDURE 语句中的结果集数必须匹配 Java 方法特征符中的 ResultSet[] 参数数目。

调用者可以描述收到的结果集。请注意，如果在多个嵌套级别上打开了同一游标，那么在 DB2 Universal Database V7 客户机上运行的应用程序只能描述所打开的第一个结果集。

调用者必须以串行方式处理结果集（如果调用者不是 SQL 主体例程）。会在第一个结果集上自动打开游标，并且会提供一个特殊调用（对于 DB2 CLI，为 SQLMoreResults; 对于 JDBC，为 getMoreResults; 对于 SQLJ，为 getNextResultSet）以关闭一个结果集上的游标，并在下一个结果集上打开游标。

要在 SQL 主体例程中接收结果集，您必须对期望返回结果集的过程执行结果集定位器声明（DECLARE）和关联（ASSOCIATE）操作。然后，您必须将期望返回的每个游标分配（ALLOCATE）给结果集定位器。在完成此操作后，您可以从结果集访存行。

如果在触发器中调用过程，那么将无法访问动态复合语句、SQL 函数或 SQL 方法、任何结果集。

注: 从过程中或从应用程序发出的 COMMIT 将关闭任何不适用于 WITH HOLD 游标的结果集。从应用程序或存储过程发出的 ROLLBACK 将关闭所有结果集游标。从过程中发出 COMMIT 或 ROLLBACK 后，可以打开游标并将其作为结果集来返回。

从 SQL 数据更改语句检索结果集: 使用 INSERT、UPDATE 或 DELETE 语句来修改表的应用程序可能要求对修改的行进行其他处理。要加快此处理，您可以在 SELECT 和 SELECT INTO 语句的 FROM 子句中嵌入 SQL 数据更改操作。在单一工作单元中，应用程序可以从 SQL 数据更改操作所修改的表或视图中，检索包含所修改行的结果集。

例如，下列语句会更新 SAMPLE 数据库中 EMPLOYEE 表内所有记录的薪水，然后返回所有已更新行的职员编号和新薪水。

```
SELECT empno, salary FROM FINAL TABLE
 (UPDATE employee SET salary = salary * 1.10 WHERE job = 'CLERK')
```

为了成功返回数据，检索 SQL 数据更改操作的结果集的 SELECT 语句要求 SQL 数据更改操作成功运行。成功执行 SQL 数据更改操作包括成功处理所有约束和触发器（如果适用）。

例如，假设对 EMPLOYEE 表具有 SELECT 特权但不具有 INSERT 特权的用户尝试对 EMPLOYEE 表执行 SELECT FROM INSERT 语句。INSERT 操作会失败，因为缺少特权，并且整个 SELECT 语句会因此而失败。

请考虑下列查询，会在该查询中查询 EMPLOYEE 表的记录，然后将记录插入到另一个表（名为 EMP）中。此 SELECT 语句将失败。

```
SELECT empno FROM FINAL TABLE
 (INSERT INTO emp(name, salary)
 SELECT firstnme || midinit || lastname, salary
 FROM employee)
```

如果 EMPLOYEE 表具有 100 行，且第 90 行具有 SALARY 值 $9,999,000.00，那么增加 $10,000.00 会导致发生十进制溢出问题。此溢出会强制数据库管理器回滚对 EMP 表所作的插入。

中间结果表

表或视图（SELECT 语句的 FROM 子句中 SQL 数据更改操作的目标）的已修改行组成中间结果表。除 SQL 数据更改操作中所定义的任何包含列之外，中间结果表还包括目标表或视图的所有列。您可以在查询列表、ORDER BY 子句或 WHERE 子句中按名称引用中间结果表中的所有列。

中间结果表的内容取决于 FROM 子句中的所指定限定词。必须在 SELECT 语句中包括下列其中一个 FROM 子句限定词，以检索结果集作为中间结果表。

OLD TABLE
: 中间结果表中的行将包含目标表行中，时间紧靠前触发器和 SQL 数据更改操作执行之前的值。OLD TABLE 限定词不适用于 UPDATE 和 DELETE 操作。

NEW TABLE
: 中间结果表中的行将包含目标表行中，时间紧跟 SQL 数据更改语句执行之后，但在引用完整性评估和任何后触发器触发之前的值。NEW TABLE 限定词适用于 UPDATE 和 INSERT 操作。

FINAL TABLE
: 此限定词返回与 NEW TABLE 相同的中间结果表。此外，使用 FINAL TABLE 可保证任何后触发器或引用完整性约束都不会进一步修改 UPDATE 或 INSERT 操作的目标。FINAL TABLE 限定词适用于 UPDATE 和 INSERT 操作。

FROM 子句限定词确定中间结果表中目标数据的版本。这些限定词不会影响目标表行的插入、删除或更新。

目标表和视图

查询 SQL 数据更改操作的结果集时，目标可以是任一表或视图。

在针对视图执行的 SQL 数据更改操作中，结果表不能包含不再满足 NEW TABLE 和 FINAL TABLE 的视图定义的行。如果您嵌入一个在 SELECT 语句中对视图进行引用

的 INSERT 或 UPDATE 语句，那么必须将该视图定义为 WITH CASCADED CHECK OPTION。此外，该视图必须满足可允许您将该视图定义为 WITH CASCADED CHECK OPTION 的限制。

如果在 SELECT 语句的 FROM 子句中嵌入的 SQL 数据更改操作的目标是全查询，那么结果表可以包括全查询中任何不再符合条件的行。这是因为不会根据更新的值对 WHERE 子句中的谓词进行重新求值。

根据 INPUT SEQUENCE 对结果集进行排序

要以行插入至目标表或视图的顺序对行执行 SELECT，请在 ORDER BY 子句中使用 INPUT SEQUENCE 关键字。使用 INPUT SEQUENCE 关键字不会强制以提供行的顺序来插入行。

下列示例演示如何在 ORDER BY 子句中使用 INPUT SEQUENCE 关键字，对 INSERT 操作的结果集进行排序。

```
CREATE TABLE orders (purchase_date DATE,
                     sales_person VARCHAR(16),
                     region VARCHAR(10),
                     quantity SMALLINT,
                     order_num INTEGER NOT NULL
                       GENERATED ALWAYS AS IDENTITY (START WITH 100,
                                     INCREMENT BY 1))

SELECT * FROM FINAL TABLE
         (INSERT INTO orders
          (purchase_date, sales_person, region, quantity)
          VALUES (CURRENT DATE,'Judith','Beijing',6),
                 (CURRENT DATE,'Marieke','Medway',5),
                 (CURRENT DATE,'Hanneke','Halifax',5))
         ORDER BY INPUT SEQUENCE

PURCHASE_DATE SALES_PERSON     REGION     QUANTITY ORDER_NUM
------------- ---------------- ---------- -------- -----------
07/18/2003    Judith           Beijing           6         100
07/18/2003    Marieke          Medway            5         101
07/18/2003    Hanneke          Halifax           5         102
```

您也可以使用包含列对结果集行进行排序。

使用游标从 SQL 数据更改语句检索结果集: 您可以针对查询（从 SQL 数据更改操作检索结果集）声明游标。例如:

```
DECLARE C1 CURSOR FOR SELECT salary FROM FINAL TABLE
        (INSERT INTO employee (name, salary, level)
            SELECT name, income, band FROM old_employee)
```

从游标（其定义包含 SQL 数据更改操作）访存时发生的错误，将不会导致回滚所修改的行。即使错误会导致游标关闭，但行修改仍保持不变，因为这些修改是在应用程序打开游标时完成。

在打开此类游标时，数据库管理器会完全执行 SQL 数据更改操作，并且会将结果集存储在临时表中。如果打开游标时发生错误，那么将回滚 SQL 数据更改操作所作的更改。对目标表或视图所作的进一步更新将不会出现在游标（从 SQL 数据更改操作检索结果集）的结果表行中。例如，应用程序会声明并打开游标，执行访存并更新表，然后访存其他行。在 UPDATE 语句之后执行的访存，将返回执行 UPDATE 语句之前打开游标过程中确定的那些值。

您可以针对查询（从 SQL 数据更改操作检索结果集）声明可滚动游标。已将数据修改写入目标表或视图，因为会在您打开（OPEN）游标时生成结果表。必须将游标（其查询可从 SQL 数据更改操作查询行）定义为 INSENSITIVE 或 ASENSITIVE。

注: 只有 CLI、JDBC 以及 SQLJ 应用程序才支持可滚动游标。

如果对游标声明 WITH HOLD 选项且应用程序执行 COMMIT，那么将落实所有数据更改。未声明为 WITH HOLD 的游标以相同方式操作。对于所有游标，会在访存任何行之前，对查询中包括的 SQL 数据更改操作进行完全求值。

如果对 OPEN CURSOR 语句执行显式回滚，或在 OPEN CURSOR 语句之前回滚至保存点，那么将撤销对该游标所作的所有数据更改。对于游标（其查询会从 SQL 数据更改操作检索结果集），将在回滚后撤销所有数据更改，但会保留该游标，因此仍将能够访存先前插入的行。

对函数的引用

不论函数是 UDF 还是内置函数，每个函数引用都包含下列语法:

在前面的语法图中，function_name 可以为非限定函数名，也可以为限定函数名。自变量数介于 0 和 90 之间，且自变量是表达式。组成表达式的某些组件示例如下所示:

- 列名（限定或非限定）
- 常量
- 主变量
- 专用寄存器
- 参数标记

自变量的位置非常重要，且必须符合函数定义才能使语义正确。自变量的位置以及函数定义都必须符合函数主体本身。DB2 数据库系统不会尝试改变自变量的顺序以更好地匹配函数定义，并且无法理解个别函数参数的语义。

在 UDF 自变量表达式中使用列名要求包含列的表引用具有正确的范围。对于连接中所引用且使用任何自变量（包含来自另一个表或表函数的列）的表函数，所引用的表或表函数必须在其 FROM 子句中包含引用的表函数之前。

为了在函数中使用参数标记，不能只是编写下列代码:

```
BLOOP(?)
```

因为函数选择逻辑不知道自变量可能转变成的数据类型，所以它无法解除引用。您可以使用 CAST 规范来提供参数标记的类型。例如 INTEGER，然后函数选择逻辑可以继续执行操作:

```
BLOOP(CAST(? AS INTEGER))
```

函数调用的某些有效示例如下:

```
AVG(FLOAT_COLUMN)
BLOOP(COLUMN1)
BLOOP(FLOAT_COLUMN + CAST(? AS INTEGER))
BLOOP(:hostvar :indicvar)
BRIAN.PARSE(CHAR_COLUMN CONCAT USER, 1, 0, 0, 1)
CTR()
FLOOR(FLOAT_COLUMN)
PABLO.BLOOP(A+B)
PABLO.BLOOP(:hostvar)
"search_schema"(CURRENT FUNCTION PATH, 'GENE')
SUBSTR(COLUMN2,8,3)
SYSFUN.FLOOR(AVG(EMP.SALARY))
SYSFUN.AVG(SYSFUN.FLOOR(EMP.SALARY))
SYSIBM.SUBSTR(COLUMN2,11,LENGTH(COLUMN3))
SQRT((SELECT SUM(length*length)
     FROM triangles
     WHERE id= 'J522'
     AND legtype <> 'HYP'))
```

如果以上任何函数是表函数，那么对函数进行引用的语法与先前所描述的稍有不同。例如，如果 PABLO.BLOOP 是表函数，请使用下列代码来正确地引用该表函数:

```
TABLE(PABLO.BLOOP(A+B)) AS Q
```

函数选择

对于限定和非限定函数引用，函数选择算法会查看所有符合下列条件的*应用*函数（内置函数和用户定义的函数）：给定名称；在函数引用中作为自变量的相同数目的已定义参数；并且每个参数与对应自变量的类型完全相同或从此类型提升。

适用函数是限定引用中命名模式中的函数，或非限定引用的 SQL 路径的模式中的函数。该算法会查找完全匹配项，如果失败，那么查找这些函数之间的最佳匹配项。在仅非限定引用的情况下，如果在不同模式中找到两个一样好的匹配项，那么将使用 SQL 路径作为判定因素。

例外：如果存在对函数 RID 的非限定引用，且使用与子查询的 FROM 子句中的表引用匹配的单一自变量来调用函数，那么模式为 SYSIBM 且调用内置 RID 函数。

您可以嵌套函数引用，甚至可以嵌套对同一函数的引用。通常，这对于内置函数以及 UDF 是成立的；但是，如果调用列函数，那么存在一些限制。

例如:

```
CREATE FUNCTION BLOOP (INTEGER) RETURNS INTEGER ...
CREATE FUNCTION BLOOP (DOUBLE) RETURNS INTEGER ...
```

现在，请考虑下列 DML 语句:

```
SELECT BLOOP( BLOOP(COLUMN1)) FROM T
```

在此语句中，如果 column1 是 DECIMAL 或 DOUBLE 列，那么内部 BLOOP 引用会解析为第二个 BLOOP。因为此 BLOOP 会返回 INTEGER，所以外部 BLOOP 会解析为第一个 BLOOP。

另外，在此 DML 语句中，如果 column1 是 SMALLINT 或 INTEGER 列，那么内部 bloop 引用会解析为第一个 BLOOP。因为此 BLOOP 会返回 INTEGER，所以外部 BLOOP 也会解析为第一个 BLOOP。在此情况下，您将看到对同一函数的嵌套引用。

通过定义具有其中一个 SQL 运算符名称的函数，您实际上可以使用*中缀表示法* 来调用 UDF。例如，假设您可以针对具有单值类型 BOAT 的值将某种含义附加至 "+" 运算符。您可以定义下列 UDF:

```
CREATE FUNCTION "+" (BOAT, BOAT) RETURNS ...
```

然后，您可以编写下列有效 SQL 语句:

```
SELECT BOAT_COL1 + BOAT_COL2
FROM BIG_BOATS
WHERE BOAT_OWNER = 'Nelson Mattos'
```

但是您也可以编写同样有效的语句:

```
SELECT "+"(BOAT_COL1, BOAT_COL2)
FROM BIG_BOATS
WHERE BOAT_OWNER = 'Nelson Mattos'
```

请注意，您无权以此方式重载内置条件运算符，例如 >、=、LIKE 和 IN 等。

将单值类型用作 UDF 或方法参数

可以将 UDF 和方法定义为使用单值类型作为参数或结果。DB2 数据库系统将按单值类型的源数据类型的格式将值传递给 UDF 或方法。

过程

如果单值类型值源自主变量并且被用作 UDF 的自变量，而该 UDF 将其相应参数定义为单值类型，那么**用户必须显式地将单值类型值强制转换为单值类型**。 单值类型没有相应的主语言类型。DB2 数据库系统中的强类型化要求执行此操作，否则结果可能不明确。请考虑基于 BLOB 定义的 BOAT 单值类型，并考虑按如下方式定义的 BOAT_COST UDF:

```
CREATE FUNCTION BOAT_COST (BOAT)
  RETURNS INTEGER
  ...
```

在 C 语言应用程序的以下片段中，主变量 :ship 存放传递给 BOAT_COST 函数的 BLOB 值:

```
EXEC SQL BEGIN DECLARE SECTION;
  SQL TYPE IS BLOB(150K) ship;
EXEC SQL END DECLARE SECTION;
```

下面这两条语句都将正确地解析为 BOAT_COST 函数，这是因为，这两条语句都将 :ship 主变量强制转换为 BOAT 类型:

```
... SELECT BOAT_COST (BOAT(:ship)) FROM ...
... SELECT BOAT_COST (CAST(:ship AS BOAT)) FROM ...
```

如果数据库中有多个 BOAT 单值类型，或者其他模式包含 BOAT UDF，那么您必须谨慎地指定 SQL 路径。否则，结果可能不明确。

作为 UDF 参数的 LOB 值

可以针对 UDF 定义具有任意 LOB 类型（BLOB、CLOB 或 DBCLOB）的参数或结果。

关于此任务

DB2 数据库系统将在存储器中具体化整个 LOB 值，然后再调用此类函数，即使值的源是 *LOB 定位器*主变量亦如此。例如，请考虑下列 C 语言应用程序片段:

```
EXEC SQL BEGIN DECLARE SECTION;
  SQL TYPE IS CLOB(150K) clob150K ;        /* LOB host var */
  SQL TYPE IS CLOB_LOCATOR clob_locator1;  /* LOB locator host var */
  char                    string[40];      /* string host var */
EXEC SQL END DECLARE SECTION;
```

主变量 :clob150K 或 :clob_locator1 适合用作其对应参数已定义为 CLOB(500K) 的函数的自变量。例如，假设您按如下所示注册 UDF:

```
CREATE FUNCTION FINDSTRING (CLOB(500K, VARCHAR(200)))
  ...
```

下列两个 FINDSTRING 调用在程序中是有效的:

```
... SELECT FINDSTRING (:clob150K, :string) FROM ...
... SELECT FINDSTRING (:clob_locator1, :string) FROM ...
```

可以使用 AS LOCATOR 修饰符来创建具有其中一种 LOB 类型的 UDF 参数或结果。在此情况下，不会在调用之前具体化整个 LOB 值。相反地，会将 LOB LOCATOR 传递至 UDF，UDF 随后可以使用 SQL 来操作 LOB 值的实际字节。

您也可以在 UDF 参数或结果（其单值类型基于 LOB）上使用此功能。请注意，此类函数的自变量可以是属于已定义类型的任何 LOB 值；它不必是一个定义为其中一种 LOCATOR 类型的主变量。主变量定位器作为自变量的用法完全独立于 AS LOCATOR 在 UDF 参数和结果定义中的用法。

调用标量函数或方法

内置标量函数、用户定义标量函数以及方法的调用非常相似。只能在 SQL 语句中支持表达式的位置调用标量函数和方法。

开始之前

- 对于内置函数，SYSIBM 必须在 CURRENT PATH 专用寄存器中。缺省情况下，SYSIBM 是在 CURRENT PATH 中。
- 对于用户定义标量函数，必须已使用 CREATE FUNCTION 或 CREATE METHOD 语句在数据库中创建该函数。
- 对于用户定义的外部标量函数，与函数相关联的库或类文件必须位于 CREATE FUNCTION 或 CREATE METHOD 语句的 EXTERNAL 子句所指定的位置。
- 要调用用户定义的函数或方法，用户必须对该函数或方法具有 EXECUTE 特权。如果所有用户都将使用该函数或方法，那么可以将该函数或方法的 EXECUTE 特权授予给 PUBLIC。有关与特权相关的更多信息，请参阅特定的 CREATE 语句参考。

过程

要调用标量 UDF 或方法，请执行下列操作:

在 SQL 语句所包含的表达式中，标量 UDF 或方法将处理一个或多个输入值的位置，包括对该标量 UDF 或方法的引用。只要是表达式为有效的位置，就可以调用函数和方法。可以在其中对标量 UDF 或方法进行引用的位置示例包括查询的查询列表或 VALUES 子句。

示例

例如，假设您已创建用户定义标量函数 TOTAL_SAL，该函数将 EMPLOYEE 表中每个职员行的基本薪水和奖金相加。

```
CREATE FUNCTION TOTAL_SAL
  (SALARY DECIMAL(9,2), BONUS DECIMAL(9,2))
  RETURNS DECIMAL(9,2)
  LANGUAGE SQL
  CONTAINS SQL
  NO EXTERNAL ACTION
  DETERMINISTIC
  RETURN SALARY+BONUS
```

下列是利用 TOTAL_SAL 的 SELECT 语句:

```
SELECT LASTNAME, TOTAL_SAL(SALARY, BONUS) AS TOTAL
  FROM EMPLOYEE
```

调用用户定义的表函数

在编写用户定义的表函数并在数据库中注册该表函数后，您可以在 SELECT 语句的 FROM 子句中调用该表函数。

开始之前

- 必须已通过执行 CREATE FUNCTION 在数据库中创建表函数。
- 对于用户定义的外部表函数，与函数相关联的库或类文件必须位于 CREATE FUNCTION 的 EXTERNAL 子句所指定的位置。
- 要调用用户定义的表函数，用户必须对函数具有 EXECUTE 特权。有关与特权相关的更多信息，请参阅 CREATE FUNCTION 参考。

过程

要调用用户定义的表函数，请在将处理一组输入值的 SQL 语句的 FROM 子句中引用该函数。对表函数的引用必须在 TABLE 子句后面，且以方括号括起。
例如，下列 CREATE FUNCTION 语句定义一个返回指定部门编号中的职员数的表函数。

```
CREATE FUNCTION DEPTEMPLOYEES (DEPTNO VARCHAR(3))
  RETURNS TABLE (EMPNO CHAR(6),
                 LASTNAME VARCHAR(15),
                 FIRSTNAME VARCHAR(12))
  LANGUAGE SQL
  READS SQL DATA
  NO EXTERNAL ACTION
  DETERMINISTIC
  RETURN
    SELECT EMPNO, LASTNAME, FIRSTNME FROM EMPLOYEE
      WHERE EMPLOYEE.WORKDEPT = DEPTEMPLOYEES.DEPTNO
```

下列是利用 DEPTEMPLOYEES 的 SELECT 语句:

```
SELECT EMPNO, LASTNAME, FIRSTNAME FROM TABLE(DEPTEMPLOYEES('A00')) AS D
```

附录 A. DB2 技术信息概述

DB2 技术信息以多种可以通过多种方法访问的格式提供。

您可以通过下列工具和方法获得 DB2 技术信息:

- DB2 信息中心
  - 主题（任务、概念和参考主题）
  - 样本程序
  - 教程
- DB2 书籍
  - PDF 文件（可下载）
  - PDF 文件（在 DB2 PDF DVD 中）
  - 印刷版书籍
- 命令行帮助
  - 命令帮助
  - 消息帮助

注: DB2 信息中心主题的更新频率比 PDF 书籍或硬拷贝书籍的更新频率高。要获取最新信息，请安装可用的文档更新或者参阅 ibm.com 上的 DB2 信息中心。

您可以在线访问 ibm.com 上的其他 DB2 技术信息，例如技术说明、白皮书和 IBM Redbooks® 出版物。请访问以下网址处的 DB2 信息管理软件资料库站点: http://www.ibm.com/software/data/sw-library/。

文档反馈

我们非常重视您对 DB2 文档的反馈。如果您想就如何改善 DB2 文档提出建议，请向 db2docs@ca.ibm.com 发送电子邮件。DB2 文档小组将阅读您的所有反馈，但无法直接给您答复。请尽可能提供具体的示例，这样我们才能更好地了解您所关心的问题。如果您要提供有关具体主题或帮助文件的反馈，请加上标题和 URL。

请不要使用以上电子邮件地址与 DB2 客户支持机构联系。如果您遇到文档无法解决的 DB2 技术问题，请与您当地的 IBM 服务中心联系以获得帮助。

硬拷贝或 PDF 格式的 DB2 技术库

下列各表描述 IBM 出版物中心（网址为 www.ibm.com/e-business/linkweb/publications/servlet/pbi.wss）所提供的 DB2 资料库。可从 www.ibm.com/support/docview.wss?rs=71&uid=swg2700947 下载 PDF 格式的 DB2 V10.1 手册的英文版本和翻译版本。

尽管这些表标识书籍有印刷版，但可能未在您所在国家或地区提供。

每次更新手册时，表单号都会递增。确保您正在阅读下面列示的手册的最新版本。

注: *DB2 信息中心*的更新频率比 PDF 或硬拷贝书籍的更新频率高。

表 24. DB2 技术信息

书名	书号	是否提供印刷版	最近一次更新时间
Administrative API Reference	SC27-3864-00	是	2012 年 4 月
Administrative Routines and Views	SC27-3865-00	否	2012 年 4 月
Call Level Interface Guide and Reference Volume 1	SC27-3866-00	是	2012 年 4 月
Call Level Interface Guide and Reference Volume 2	SC27-3867-00	是	2012 年 4 月
Command Reference	SC27-3868-00	是	2012 年 4 月
数据库管理概念和配置参考	S151-1758-00	是	2012 年 4 月
Data Movement Utilities Guide and Reference	S151-1756-00	是	2012 年 4 月
数据库监视指南和参考	S151-1759-00	是	2012 年 4 月
数据恢复及高可用性指南与参考	S151-1755-00	是	2012 年 4 月
数据库安全性指南	S151-1753-01	是	2012 年 4 月
DB2 Workload Management Guide and Reference	SC27-3891-00	是	2012 年 4 月
开发 ADO.NET 和 OLE DB 应用程序	S151-1765-00	是	2012 年 4 月
开发嵌入式 SQL 应用程序	S151-1763-00	是	2012 年 4 月
Developing Java Applications	SC27-3875-00	是	2012 年 4 月
Developing Perl, PHP, Python, and Ruby on Rails Applications	SC27-3876-00	否	2012 年 4 月
开发用户定义的例程（SQL 和外部例程）	S151-1761-00	是	2012 年 4 月
数据库应用程序开发入门	G151-1764-00	是	2012 年 4 月
Linux 和 Windows 上的 DB2 安装和管理入门	G151-1769-00	是	2012 年 4 月
全球化指南	S151-1757-00	是	2012 年 4 月
安装 DB2 服务器	G151-1768-00	是	2012 年 4 月
安装 IBM Data Server Client	G151-1751-00	否	2012 年 4 月
消息参考第 1 卷	S151-1767-00	否	2012 年 4 月
消息参考第 2 卷	S151-1766-00	否	2012 年 4 月
Net Search Extender 管理和用户指南	S151-1078-00	否	2012 年 4 月

表 24. DB2 技术信息 (续)

书名	书号	是否提供印刷版	最近一次更新时间
分区和集群指南	S151-1754-00	是	2012 年 4 月
pureXML 指南	S151-1775-00	是	2012 年 4 月
Spatial Extender User's Guide and Reference	SC27-3894-00	否	2012 年 4 月
《SQL 过程语言: 应用程序启用和支持》	S151-1762-00	是	2012 年 4 月
SQL Reference Volume 1	SC27-3885-00	是	2012 年 4 月
SQL Reference Volume 2	SC27-3886-00	是	2012 年 4 月
Text Search Guide	SC27-3888-00	是	2012 年 4 月
故障诊断和调整数据库性能	S151-1760-00	是	2012 年 4 月
升级到 DB2 V10.1	S151-1770-00	是	2012 年 4 月
DB2 V10.1 新增内容	S151-1752-00	是	2012 年 4 月
XQuery 参考	S151-1774-00	否	2012 年 4 月

表 25. 特定于 DB2 Connect 的技术信息

书名	书号	是否提供印刷版	最近一次更新时间
DB2 Connect 安装和配置 DB2 Connect Personal Edition	S151-1773-00	是	2012 年 4 月
DB2 Connect 安装和配置 DB2 Connect 服务器	S151-1772-00	是	2012 年 4 月
DB2 Connect 用户指南	S151-1771-00	是	2012 年 4 月

从命令行处理器显示 SQL 状态帮助

DB2 产品针对可能充当 SQL 语句结果的条件返回 SQLSTATE 值。SQLSTATE 帮助说明 SQL 状态和 SQL 状态类代码的含义。

过程

要启动 SQL 状态帮助，请打开命令行处理器并输入:

```
? sqlstate or ? class code
```

其中，*sqlstate* 表示有效的 5 位 SQL 状态，*class code* 表示该 SQL 状态的前 2 位。例如，`? 08003` 显示 08003 SQL 状态的帮助，而 `? 08` 显示 08 类代码的帮助。

访问不同版本的 DB2 信息中心

您可以在 ibm.com® 上的不同信息中心中找到其他版本 DB2 产品的文档。

关于此任务

对于 DB2 V10.1 主题，*DB2 信息中心* URL 是 http://publib.boulder.ibm.com/infocenter/db2luw/v10r1。

对于 DB2 V9.8 主题，*DB2 信息中心* URL 是 http://publib.boulder.ibm.com/infocenter/db2luw/v9r8/。

对于 DB2 V9.7 主题，*DB2 信息中心* URL 是 http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/。

对于 DB2 V9.5 主题，*DB2 信息中心* URL 是 http://publib.boulder.ibm.com/infocenter/db2luw/v9r5。

对于 DB2 V9.1 主题，*DB2 信息中心* URL 是 http://publib.boulder.ibm.com/infocenter/db2luw/v9/。

对于 DB2 V8 主题，请转至 *DB2 信息中心* URL: http://publib.boulder.ibm.com/infocenter/db2luw/v8/。

更新安装在计算机或内部网服务器上的 DB2 信息中心

安装在本地的 DB2 信息中心必须定期进行更新。

开始之前

必须已安装 DB2 V10.1 信息中心。有关详细信息，请参阅安装 DB2 服务器中的“使用 DB2 安装向导来安装 DB2 信息中心”主题。所有适用于安装信息中心的先决条件和限制同样适用于更新信息中心。

关于此任务

可以自动或手动更新现有的 DB2 信息中心:

- 自动更新将更新现有的信息中心功能部件和语言。自动更新的一个优点是，与手动更新相比，信息中心的不可用时间较短。另外，自动更新可设置为作为定期运行的其他批处理作业的一部分运行。
- 可以使用手动更新方法来更新现有的信息中心功能部件和语言。自动更新可以缩短更新过程中的停机时间，但如果您想添加功能部件或语言，那么必须执行手动过程。例如，如果本地信息中心最初安装的是英语和法语版，而现在还要安装德语版；那么手动更新将安装德语版，并更新现有信息中心的功能和语言。但是，手动更新要求您手动停止、更新和重新启动信息中心。在整个更新过程期间信息中心不可用。在自动更新过程中，信息中心仅在更新完成后停止工作以重新启动信息中心。

此主题详细说明了自动更新的过程。有关手动更新的指示信息，请参阅“手动更新安装在您的计算机或内部网服务器上的 DB2 信息中心”主题。

过程

要自动更新安装在计算机或内部网服务器上的 DB2 信息中心:

1. 在 Linux 操作系统上，
   a. 浏览至信息中心的安装位置。缺省情况下，DB2 信息中心安装在 `/opt/ibm/db2ic/V10.1` 目录中。
   b. 从安装目录浏览至 `doc/bin` 目录。
   c. 运行 `update-ic` 脚本:

```
update-ic
```

2. 在 Windows 操作系统上，
   a. 打开命令窗口。
   b. 浏览至信息中心的安装位置。缺省情况下，DB2 信息中心安装在 <Program Files>\IBM\DB2 Information Center\V10.1 目录中，其中 <Program Files> 表示 Program Files 目录的位置。
   c. 从安装目录浏览至 doc\bin 目录。
   d. 运行 update-ic.bat 文件:

```
update-ic.bat
```

结果

DB2 信息中心将自动重新启动。如果更新可用，那么信息中心会显示新的以及更新后的主题。如果信息中心更新不可用，那么会在日志中添加消息。日志文件位于 doc\eclipse\configuration 目录中。日志文件名称是随机生成的编号。例如，1239053440785.log。

手动更新安装在计算机或内部网服务器上的 DB2 信息中心

如果您已在本地安装 DB2 信息中心，那么可从 IBM 获取文档更新并进行安装。

关于此任务

手动更新安装在本地的 *DB2 信息中心*要求您:

1. 停止计算机上的 *DB2 信息中心*，然后以独立方式重新启动信息中心。如果以独立方式运行信息中心，那么网络上的其他用户将无法访问信息中心，因而您可以应用更新。DB2 信息中心的工作站版本总是以独立方式运行。
2. 使用“更新”功能部件来查看可用的更新。如果有您必须安装的更新，那么请使用“更新”功能部件来获取并安装这些更新。

   注: 如果您的环境要求在一台未连接至因特网的机器上安装 *DB2 信息中心*更新，请使用一台已连接至因特网并已安装 *DB2 信息中心*的机器将更新站点镜像至本地文件系统。如果网络中有许多用户将安装文档更新，那么可以通过在本地也为更新站点制作镜像并为更新站点创建代理来缩短每个人执行更新所需要的时间。
   如果提供了更新包，请使用“更新”功能部件来获取这些更新包。但是，只有在单机方式下才能使用“更新”功能部件。
3. 停止独立信息中心，然后在计算机上重新启动 *DB2 信息中心*。

注: 在 Windows 2008、Windows Vista 和更高版本上，稍后列示在此部分的命令必须作为管理员运行。要打开具有全面管理员特权的命令提示符或图形工具，请右键单击快捷方式，然后选择**以管理员身份运行**。

过程

要更新安装在您的计算机或内部网服务器上的 *DB2 信息中心*:

1. 停止 *DB2 信息中心*。
   - 在 Windows 上，单击**开始 > 控制面板 > 管理工具 > 服务**。右键单击 **DB2 信息中心**服务，并选择**停止**。

- 在 Linux 上，输入以下命令:

  ```
  /etc/init.d/db2icdv10 stop
  ```

2. 以独立方式启动信息中心。
   - 在 Windows 上:
     a. 打开命令窗口。
     b. 浏览至信息中心的安装位置。缺省情况下，*DB2 信息中心*安装在 *Program_Files*\IBM\DB2 Information Center\V10.1 目录中，其中 *Program Files* 表示 Program Files 目录的位置。
     c. 从安装目录浏览至 doc\bin 目录。
     d. 运行 help_start.bat 文件:

        ```
        help_start.bat
        ```

   - 在 Linux 上:
     a. 浏览至信息中心的安装位置。缺省情况下，*DB2 信息中心*安装在 /opt/ibm/db2ic/V10.1 目录中。
     b. 从安装目录浏览至 doc/bin 目录。
     c. 运行 help_start 脚本:

        ```
        help_start
        ```

   系统缺省 Web 浏览器将打开以显示独立信息中心。
3. 单击**更新**按钮（  ）。（必须在浏览器中启用 JavaScript。）在信息中心的右边面板上，单击**查找更新**。将显示现有文档的更新列表。
4. 要启动安装过程，请检查您要安装的选项，然后单击**安装更新**。
5. 在安装进程完成后，请单击**完成**。
6. 要停止独立信息中心，请执行下列操作:
   - 在 Windows 上，浏览至安装目录中的 doc\bin 目录并运行 help_end.bat 文件:

     ```
     help_end.bat
     ```

     注: help_end 批处理文件包含安全地停止使用 help_start 批处理文件启动的进程所需的命令。不要使用 Ctrl-C 或任何其他方法来停止 help_start.bat。
   - 在 Linux 上，浏览至安装目录中的 doc/bin 目录并运行 help_end 脚本:

     ```
     help_end
     ```

     注: help_end 脚本包含安全地停止使用 help_start 脚本启动的进程所需的命令。不要使用任何其他方法来停止 help_start 脚本。
7. 重新启动 *DB2 信息中心*。
   - 在 Windows 上，单击**开始 > 控制面板 > 管理工具 > 服务**。右键单击 **DB2 信息中心**服务，并选择**启动**。
   - 在 Linux 上，输入以下命令:

     ```
     /etc/init.d/db2icdv10 start
     ```

结果

更新后的 *DB2 信息中心*将显示新的以及更新后的主题。

DB2 教程

DB2 教程帮助您了解 DB2 数据库产品的各个方面。这些课程提供了逐步指示信息。

开始之前

您可以在信息中心中查看 XHTML 版的教程: http://publib.boulder.ibm.com/infocenter/db2luw/v10r1/。

某些课程使用了样本数据或代码。有关其特定任务的任何先决条件的描述，请参阅教程。

DB2 教程

要查看教程，请单击标题。

*pureXML 指南*中的『**pureXML®**』

设置 DB2 数据库以存储 XML 数据以及对本机 XML 数据存储器执行基本操作。

DB2 故障诊断信息

我们提供了各种各样的故障诊断和问题确定信息来帮助您使用 DB2 数据库产品。

DB2 文档

您可以在*故障诊断和调整数据库性能*或者 *DB2 信息中心*的"数据库基础"部分中找到故障诊断信息，这些信息包含以下内容:

- 有关如何使用 DB2 诊断工具和实用程序来隔离和确定问题的信息。
- 一些最常见问题的解决方案。
- 旨在帮助您解决 DB2 数据库产品使用过程中可能会遇到的其他问题的建议。

IBM 支持门户网站

如果您遇到问题并且希望得到帮助以查找可能的原因和解决方案，请访问 IBM 支持门户网站。这个技术支持站点提供了指向最新 DB2 出版物、技术说明、授权程序分析报告（APAR 或错误修订）、修订包和其他资源的链接。可搜索此知识库并查找问题的可能解决方案。

访问 IBM 支持门户网站: http://www.ibm.com/support/entry/portal/Overview/Software/Information_Management/DB2_for_Linux,_UNIX_and_Windows

信息中心条款和条件

如果符合以下条款和条件，那么授予您使用这些出版物的许可权。

适用性: 用户需要遵循 IBM Web 站点的使用条款及以下条款和条件。

个人使用: 只要保留所有的专有权声明，您就可以为个人、非商业使用复制这些出版物。未经 IBM 明确同意，您不可以分发、展示或制作这些出版物或其中任何部分的演绎作品。

商业使用: 只要保留所有的专有权声明，您就可以仅在企业内复制、分发和展示这些出版物。未经 IBM 明确同意，您不可以制作这些出版物的演绎作品，或者在您的企业外部复制、分发或展示这些出版物或其中的任何部分。

权利：除非本许可权中明确授予，否则不得授予对这些出版物或其中包含的任何信息、数据、软件或其他知识产权的任何许可权、许可证或权利，无论是明示的还是暗含的。

IBM 保留根据自身的判断，认为对出版物的使用损害了 IBM 的权益（由 IBM 自身确定）或未正确遵循以上指示信息时，撤回此处所授予权限的权利。

只有您完全遵循所有适用的法律和法规，包括所有的美国出口法律和法规，您才可以下载、出口或再出口该信息。

IBM 对这些出版物的内容不作任何保证。这些出版物“按现状”提供，不附有任何种类的（无论是明示的还是暗含的）保证，包括但不限于暗含的关于适销和适用于某种特定用途的保证。

附录 B. 声明

本信息是为在美国提供的产品和服务编写的。有关非 IBM 产品的信息是基于首次出版此文档时的可获信息且会随时更新。

IBM 可能在其他国家或地区不提供本文档中讨论的产品、服务或功能特性。有关您当前所在区域的产品和服务的信息，请向您当地的 IBM 代表咨询。任何对 IBM 产品、程序或服务的引用并非意在明示或暗示只能使用 IBM 的产品、程序或服务。只要不侵犯 IBM 的知识产权，任何同等功能的产品、程序或服务，都可以代替 IBM 产品、程序或服务。但是，评估和验证任何非 IBM 产品、程序或服务，则由用户自行负责。

IBM 公司可能已拥有或正在申请与本文档内容有关的各项专利。提供本文档并未授予用户使用这些专利的任何许可。您可以用书面方式将许可查询寄往:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

有关双字节字符集 (DBCS) 信息的许可查询，请与您所在国家或地区的 IBM 知识产权部门联系，或用书面方式将查询寄往:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

本条款不适用英国或任何这样的条款与当地法律不一致的国家或地区: International Business Machines Corporation“按现状”提供本出版物，不附有任何种类的（无论是明示的还是暗含的）保证，包括但不限于暗含的有关非侵权、适销和适用于某种特定用途的保证。某些国家或地区在某些交易中不允许免除明示或暗含的保证。因此本条款可能不适用于您。

本信息中可能包含技术方面不够准确的地方或印刷错误。此处的信息将定期更改；这些更改将编入本资料的新版本中。IBM 可以随时对本资料中描述的产品和/或程序进行改进和/或更改，而不另行通知。

本信息中对非 IBM Web 站点的任何引用都只是为了方便起见才提供的，不以任何方式充当对那些 Web 站点的保证。那些 Web 站点中的资料不是此 IBM 产品资料的一部分，使用那些 Web 站点带来的风险将由您自行承担。

IBM 可以按它认为适当的任何方式使用或分发您所提供的任何信息而无须对您承担任何责任。

本程序的被许可方如果要了解有关程序的信息以达到如下目的: (i) 允许在独立创建的程序和其他程序（包括本程序）之间进行信息交换，以及 (ii) 允许对已经交换的信息进行相互使用，请与下列地址联系:

IBM Canada Limited
U59/3600
3600 Steeles Avenue East
Markham, Ontario L3R 9Z7
CANADA

只要遵守适当的条款和条件，包括某些情形下的一定数量的付费，都可获得这方面的信息。

本资料中描述的许可程序及其所有可用的许可资料均由 IBM 依据 IBM 客户协议、IBM 国际软件许可协议或任何同等协议中的条款提供。

此处包含的任何性能数据都是在受控环境中测得的。因此，在其他操作环境中获得的数据可能会有明显的不同。有些测量可能是在开发级的系统上进行的，因此不保证与一般可用系统上进行的测量结果相同。此外，有些测量是通过推算而估计的，实际结果可能会有差异。本文档的用户应当验证其特定环境的适用数据。

涉及非 IBM 产品的信息可从这些产品的供应商、其出版说明或其他可公开获得的资料中获取。IBM 没有对这些产品进行测试，也无法确认其性能的精确性、兼容性或任何其他关于非 IBM 产品的声明。有关非 IBM 产品性能的问题应当向这些产品的供应商提出。

所有关于 IBM 未来方向或意向的声明都可随时更改或收回，而不另行通知，它们仅仅表示了目标和意愿而已。

本信息可能包含在日常业务操作中使用的数据和报告的示例。为了尽可能完整地说明这些示例，示例中可能会包括个人、公司、品牌和产品的名称。所有这些名称都是虚构的，与实际商业企业所用的名称和地址的任何雷同纯属巧合。

版权许可:

本信息包括源语言形式的样本应用程序，这些样本说明不同操作平台上的编程方法。如果是为按照在编写样本程序的操作平台上的应用程序编程接口 (API) 进行应用程序的开发、使用、经销或分发，您可以任何形式对这些样本程序进行复制、修改、分发，而无须向 IBM 付费。这些示例并未在所有条件下作全面测试。因此，IBM 不能担保或暗示这些程序的可靠性、可维护性或功能。此样本程序“按现状”提供，且不附有任何种类的保证。对于使用此样本程序所引起的任何损坏，IBM 将不承担责任。

凡这些样本程序的每份拷贝或其任何部分或任何衍生产品，都必须包括如下版权声明:

©（*贵公司的名称*）（*年份*）. 此部分代码是根据 IBM 公司的样本程序衍生出来的。© Copyright IBM Corp. *（输入年份）*. All rights reserved.

商标

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other prod-

uct and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at 『Copyright and trademark information』 at www.ibm.com/legal/copytrade.shtml.

The following terms are trademarks or registered trademarks of other companies

- Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
- Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle, its affiliates, or both.
- UNIX is a registered trademark of The Open Group in the United States and other countries.
- Intel, Intel logo, Intel Inside, Intel Inside logo, Celeron, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.
- Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

索引

[A]

[B]

[C]

[D]

[E]

[F]

[K]

[L]

[M]

[N]

[P]

[Q]

[S]

[T]

[W]

[X]

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

[特别字符]

Printed in China

S151-1761-00

Spine information:

IBM DB2 10.1 for Linux, UNIX, and Windows

开发用户定义的例程（SQL 和外部例程）