Tech Notes (vol.5)

데이터베이스 환경 설정

개요

이 글은 Informix Press/Prentice Hall PTR에서 발간한 Informix OnLine Dynamic Server Handbook에서 5장 "Building a Database Environment"를 발췌한 것으로서, 데이터베이스 관리 문제를 중심으로 이와 관련된 Informix Dynamic Server의 기능과 메커니즘을 주요 내용으로 다루고 있습니다.

또한 데이터베이스와 테이블 생성 및 분할, 테이블에서 자료를 구축하는 방법, 새로 추가되거나 향상된 SQL 문에 대해 살펴 보겠습니다. 따라서 Informix Dynamic Server 인스턴스에서의 데이터베이스 생성 및 운영에 관련된 기본 작업을 어느 정도 파악할 수 있게 될 것입니다.

지금까지 데이터베이스 관리 작업을 수행할 때는, 주로 dbaccess 질의어나 I-SQL 도구를 통해 직접 SQL 문을 실행하여 데이터베이스 관리 명령이나 DDL 문을 생성하거나 실행해 왔습니다. 이 글의 일부 예제에서 이 방법을 사용하기는 했지만, Informix Dynamic Server의 dbaccess 유틸리티에서 제공하는 링 메뉴 옵션을 사용하여 여기서 설명하는 모든 데이터베이스 관리 작업을 수행할 수 있습니다. 따라서 선호도와 상황에 따라 사용하기 편리한 방법을 선택하는 것이 중요합니다. SQL 문보다는 dbaccess 링 메뉴를 사용하는 것이 훨씬 편리하지만, 제어 범위가 넓고 작업 속도가 빠르다는 이유로 일부 데이터베이스 관리자는 SQL 문을 더 선호하기도 합니다.

로깅모드

데이터베이스의 자료를 변경한 다음 이를 디스크에 기록해 두면, 시스템에 장애가 발생하더라도 자료의 무결성을 확보할 수 있습니다. 해당 데이터베이스에 로깅 모드를 설정하여 이러한 레코드들이 기록되도록 하거나, 로깅 모드를 비활성화시켜 레코드가 기록되지 않도록 할 수 있습니다.

데이터베이스의 로깅 모드를 활성화시키면 인스턴스의 물리 로그 및 논리 로그 구조를 사용하여 변경 내용이 해당 자료에 기록됩니다. 같은 인스턴스의 데이터베이스라도 로깅 모드를 다르게 설정할 수 있습니다. 데이터베이스의 로깅 모드는 생성 단계에서 설정되며 언제든지 모드를 바꿀 수 있습

니다. 로깅 모드가 설정된 데이터베이스 환경에서는 개별 사용자 또는 응용프로그램에서 "set log" 명령을 사용하여 로깅 모드를 부분적으로 변경할 수 있습니다.

로깅 모드에는 네 가지가 있으며, 그 중 두 가지는 로그 버퍼가 디스크로 플러시되는 시기만 다를 뿐 거의 비슷합니다. "로깅 미지정(No Logging)" 모드로 설정하면 논리 로그에 정보가 거의 로깅되지 않고, 단지 create/drop table, create/drop index, create/drop procedure, rename table/column, alter table 등의 DDL 문만 로깅됩니다. "로깅 미지정" 모드에서는 이러한 구문 외에 행의 변경은 로깅되지 않으며, 명령을 실행한 다음 그 결과 코드만 되돌려 집니다. "로깅 미지정"으로 설정하면 처리율은 높을지 모르지만 나중에 중대한 인스턴스 오류가 발생할 경우, 변경된 데이터베이스 내용을 재구축할 수 없습니다. 따라서 디스크에 기록된 변경 내용은 언제든지 사용할 수 있으나, 복원해야 할 경우에는 마지막으로 생성한 인스턴스 백업으로만 복원할 수 있습니다.

"ANSI 표준(Mode ANSI)" 로깅은 뒤에서 설명할 버퍼 미지정 모드와 비슷하지만, 이 모드에서는 ANSI 규격의 트랜잭션도 실행할 수 있다는 점이 다릅니다. ANSI 규격의 트랜잭션에서는 각 테이블 참조에 대해 고유한 소유자 이름을 지정하고, 테이블 수준 권한에 따라 기본값을 다르게 지정할 수 있는 기능이 포함됩니다. 또한 커서의 갱신 기능과 읽기 기능을 다르게 하고, 자료형 오버플로 및 정의 구문에 대해 문자와 숫자의 반응 방법을 다르게 할 수 있습니다.

Informix Dynamic Server에서는 Mode ANSI 데이터베이스 환경에서 ANSI 표준 이외의 구문도 실행할 수 있습니다. ANSI가 아닌 SQL 문을 실행하면 경고 메시지가 나타날 수 있지만 해당 구문은 계속 처리됩니다. 그러나 운영 환경 상 반드시 ANSI 표준이 필요한 경우가 아니라면, "표준 ANSI" 모드로 설정하지 않는 것이 좋습니다.

"버퍼 미지정(Unbuffered Logging)"과 "버퍼 지정 로깅(Buffered Logging)"은 레코드가 디스크에 로깅되는 시기만 다를 뿐 똑같은 방법으로 운영됩니다. 두 로깅 모드 모두 DDL 문을 로깅 처리하며, select 구문을 제외한 모든 DML 문(예: insert, update, delete 명령)이 로깅됩니다.

select 구문 중에서는 "select into temp" 구문만 로깅됩니다. "select into temp" 문을 로깅할 때 응

용프로그램이나 질의에서 많은 양의 자료 부집합을 생성한 다음 삭제하면 오류가 발생할 수 있습니다. 이러한 오류를 방지하려면 "with no log" 키워드를 사용하여 임시 DB영역에 임시 테이블을 만들어야 합니다.

이 두 로깅 모드는 로그 자료를 디스크의 로그에 기록할 때 차이가 있습니다. 버퍼 미지정 로깅으로 설정한 데이터베이스 환경에서는 트랜잭션이 완료될 때마다 트랜잭션 정보가 들어 있는 논리 및 물리 로그 버퍼를 플러시합니다. 그러나 버퍼 지정 로그 데이터베이스 환경에서는 버퍼가 가득 차거나, 검사점이 발생하거나, 트랜잭션을 생성한 사용자의 연결이 종료되어 트랜잭션이 더 이상 로그에 기록되지 않을 때까지 논리 및 물리 로그 버퍼에 트랜잭션 정보를 저장합니다.

버퍼 지정 로깅 데이터베이스의 트랜잭션 정보가 해제되는 또 다른 조건이 있으며, 이는 한 인스턴스에 존재하는 논리 로그 버퍼 집합이 하나뿐이기 때문입니다. 이 조건은 버퍼 미지정 데이터베이스에서 트랜잭션이 수행될 때 일어나며, 이 때 버퍼(모드) 로그 정보가 버퍼 미지정(모드) 로그 정보와 함께 디스크에 로깅됩니다.

이 두 가지 로깅 모드는 각각 장단점이 있습니다. 버퍼 미지정 로깅에서는 중대한 인스턴스 오류가 발생해도 자료 무결성이나 일관성 등을 확보할 수 있는 반면, 트랜잭션이 완료될 때마다 버퍼의 내용을 디스크로 플러시하므로 디스크 입출력 작업이 빈번하게 일어납니다. 또한 디스크로 플러시할 때 현재 처리 중인 트랜잭션에 대해 버퍼에 보관된 정보를 논리 로그에 기록하므로, 시간이 지나면 논리 로그 페이지에 불필요한 자료가 많아집니다. 따라서 버퍼 지정 데이터베이스 환경에서보다 디스크의 로그 파일이 더 빨리 채워지지만 로그 파일에 쌓인 모든 자료들이 "실제로" 꼭 필요한 자료는 아니라는 것입니다.

버퍼 지정 로깅 데이터베이스 환경에서는 일반적으로 버퍼가 가득 찼을 때 디스크로 플러시하므로 각 트랜잭션에 따른 디스크 입출력 작업이 훨씬 적고 따라서 인스턴스 실행 속도가 더 빠릅니다. 그러나 트랜잭션 정보가 공유 메모리에 보관되기 때문에 중대한 인스턴스 오류가 발생했을 때 미처 로그 버퍼의 내용을 디스크로 플러시하지 못했을 수가 있으므로 문제가 생길 수 있습니다. 즉, 인스턴스의 공유 메모리가 다른 작업에서 사용할 수 있도록 해제되면 디스크에 기록되지 않은 트랜잭션

정보가 손실된다는 것입니다.

로깅 모드를 사용할 때는 다음과 같은 사항을 고려해야 합니다.

- 자료의 휘발성
- 필요한 전체 데이터베이스 처리량
- 트랜잭션 손실로 인한 사업적 영향
- 트랜잭션 재생성력

  이러한 사항을 고려하여 사용 중인 환경에 맞는 모드를 선택합니다.

데이터베이스 생성

데이터베이스는 dbaccess의 링 메뉴 옵션 또는 dbimport 유틸리티를 사용하거나, dbaccess에서 SQL 문을 실행하여 생성할 수 있습니다. 데이터베이스를 만든 다음에는 DB영역을 만들고 로깅 모드를 지정합니다.

dbaccess의 **Database:Create** 옵션을 선택하면 작업을 수행할 수 있는 프롬프트가 나타나는데, 여기에 데이터베이스 이름을 입력하면 DB영역과 로깅 모드를 지정할 수 있는 링 메뉴가 나타납니다. Database:Create:Dbspace 옵션을 선택하면 인스턴스에 있는 DB영역 목록이 나타나며, 기본적으로는 rootdb 영역이 선택됩니다. 목록에 나타난 DB영역 중 하나를 선택하고 "Enter" 또는 "Return" 키를 누릅니다.

Database:Create:Log옵션을 사용하면, 링 메뉴에서 앞에서 설명한 네 가지 로깅 모드 중 하나를 선택할 수 있습니다. 기본값은 "None" 즉, 로깅을 하지 않는 것입니다. 이 두 화면에서 아무 것도 선택하지 않고 그대로 종료하면 기본값을 사용하여 데이터베이스가 생성됩니다.

Database:Create 옵션을 종료할 때 데이터베이스를 만들 것인지(기본값), 아니면 입력된 정보를 무시하고 데이터베이스 생성 작업을 취소할 것인지를 선택할 수 있습니다. 이 때 새로 만들어진 데이터베이스는 dbaccess 세션의 "현재" 데이터베이스가 됩니다.

데이터베이스를 생성하는 또 하나의 방법은 dbimport 유틸리티를 사용하는 것입니다. 이 방법을 사용하면 데이터베이스뿐 아니라 테이블, 색인, 제한 조건 등을 생성할 수 있습니다. 이 유틸리티를 실행하면 디스크나 테이프에 ASCII 파일로 저장된 자료로 데이터베이스가 구축됩니다.

OnLine에서는 dbimport 세션이 하나의 트랜잭션으로 인식됩니다. 만약 dbimport 세션 중 로깅 데이터베이스를 만들도록 로그 모드 플래그를 설정한 상태에서 로드할 자료 행이 많을 때는, 트랜잭션 처리가 길어져서 임포트가 전상회복됩니다. 따라서, dbimport를 사용하려면 로그 모드 플래그를 설정하지 말고 나중에 로깅 모드를 설정하는 것이 좋습니다.

DB영역 생성과 로깅 모드 선택 작업은 명령에 플래그를 붙여 수행할 수 있습니다. 이 두 옵션의 구문은 다음과 같습니다.

dbimport db_name -d dbspace명 [-l | -l buffered | -ansi]

"-l" 옵션을 사용하면 버퍼 미지정 로깅 모드로 설정되고, "-l buffered" 옵션을 사용하면 버퍼 지정 로깅 모드로 설정됩니다. 로그 모드 플래그를 사용하지 않으면 로깅을 하지 않는 상태로 데이터베이스가 만들어지며, DB영역을 생성하지 않으면 데이터베이스가 기본적으로 rootdbs 영역에 만들어집니다.

dbaccess에서 SQL 문을 사용하면 데이터베이스를 빨리 생성할 수 있습니다. SQL 문은 데이터베이스로 보내지기 때문에, 데이터베이스의 이름을 지정하지 않고 dbaccess를 호출하면 **Query:New**를 선택할 때 자동으로 **Database:Selection** 화면이 나타납니다. "interrupt" 키를 누르면 이 화면 대신 **Query:New** 화면이 나타납니다. SQL 구문은 다음과 같이 비교적 간단합니다.

create database db_name

[in dbspace 명]

[with log | with buffered log | log mode ansi]

각 옵션들을 특별히 지정하지 않는 한 dbimport 유틸리티와 마찬가지로, 데이터베이스가 로깅 미지정 모드로 rootdbs에 만들어집니다. 예를 들어,

create database chap5_db in dbs_0 with log;

를 실행하면 "chap5_db"라는 이름의 데이터베이스가 "dbs_0" DB영역에 버퍼 미지정 모드로 생성되어 운영됩니다.

테이블과 색인 생성 및 분할

데이터베이스를 만든 다음에는 실제로 자료를 저장할 테이블을 만들어야 합니다. 데이터베이스를 구축한 방식과, 자료의 무결성이나 접근 속도를 높이기 위해 사용한 방법은 데이터베이스의 전체 성능에 영향을 줍니다. 여기서는 디스크 입출력 로드를 조절하고, 색인을 생성하지 않은 상태에서 이에 따르는 삽입 및 갱신 오버헤드를 감당할 수 있도록 테이블을 구축하는 방법에 대해 알아 보겠습니다.

- 테이블과 색인 생성

테이블과 색인은 앞에서 설명한 세 가지 데이터베이스 생성 방법을 사용하여 만들 수 있습니다. 그러나 이 세 가지 방법을 사용하면 테이블을 분할할 수는 있지만 색인을 분할할 수는 없습니다. 색인을 분할하려면 SQL 문을 실행해야 하는데, 이 구문은 dbimport나 dbaccess 유틸리티를 통해서만 가능합니다.

dbimport 유틸리티를 사용하면, dbexport 처리 중 생성된 데이터베이스 스키마가 사용되기 때문에, 테이블과 색인은 스키마에 있는 구문을 사용하여 생성됩니다. dbexport 작업에서 "-ss"나 "server specific" 플래그를 사용하면, 임포트 시 사용할 수 있도록 DB영역 위치와 익스텐트 크기 정보가 저장됩니다. 이 기능은 기존 버전의 엔진과 비교해 볼 때 매우 뛰어난 Informix Dynamic Server의 관리 기능 중 하나입니다. 테이블의 크기나 위치를 변경하려면 데이터베이스를 임포트하기 전에 해야 하며,

스키마 파일을 수정함으로써 이런 변경 작업을 손쉽게 할 수 있습니다. dbschema 스크립트에서는 열 이름 외의 열 정보를 수정할 수 없습니다. 만약 다른 정보를 수정하면 자료를 테이블로 다시 읽어 들일 때 자료형이 일치하지 않는 문제가 발생할 수 있습니다.

dbaccess 메뉴 옵션을 사용하여 테이블을 생성하거나 분할하는 방법은 데이터베이스 생성 방법과 비슷합니다. Table:Create나 Table:Alter 옵션을 선택하면, 여러 가지 링 메뉴 옵션을 통해 열 정의를 입력하거나 변경할 수 있는 화면이 나타납니다. 테이블을 만들 때 DB영역 생성, 익스텐트 크기, 로크 모드, 분할 정보 등을 설정할 수 있는 추가 옵션이 제공됩니다.

이 화면에서 따로 지정하지 않으면 데이터베이스를 생성한 DB영역에 페이지 수준 로크를 사용하여 테이블이 만들어지고, 테이블 크기는 16KB로 지정됩니다. 제한 조건과 색인도 dbaccess를 통해 생성할 수 있지만, 작업 절차가 다소 까다로우므로 대신 SQL 문을 사용하는 것이 좋습니다.

SQL 문을 직접 실행하여 테이블이나 색인을 만드는 구문 규칙은 제품과 함께 제공되는 Informix Guide to SQL - Syntax를 참조하십시오. 여기서는 테이블과 색인 생성 구문을 자세히 살펴 보기보다 이러한 구문을 실행하는 응용프로그램을 중심으로 설명하겠습니다. 자세한 스키마 디자인과 표준에 대해서는 여기서 설명하지 않으나 이에 따라 적합한 분할 응용 프로그램이 달라집니다. 예를 들어, 다음과 같은 구문으로 테이블과 색인을 생성할 수 있습니다.

```
create table store_sales

( division smallint,

  store smallint,

  category char(3),

  sales_date date,
```

```
    amt_sold decimal(8,2)

) in dbs_12

    extent size 550000 next size 55000

    lock mode row;

create index ix_strsls_div on store_sales (division);

create index ix_strsls_cat on store_sales (category);

create index ix_strsls_dt on store_sales (sales_date);
```

- 테이블 분할

Informix Dynamic Server의 새로운 기능 중 하나는 테이블과 색인을 분할할 수 있다는 점입니다. 이는 말 그대로, 성능을 향상시키기 위해 테이블과 색인을 나누어 여러 DB영역에 분산시킬 수 있는 기능입니다. RAID 수준 0(스트라이핑)과는 달리, 분할 스킴에 논리 규칙을 적용하여 입출력 성능 및 가용도를 향상시킬 수 있습니다. Informix Dynamic Server에서는 분할 테이블에 접근할 수 있도록 $DATASKIP 매개변수를 설정할 수 있으며, 이 기능은 분할 테이블이 다운된 DB영역에 있더라도 가능합니다.

테이블과 색인 모두 분할될 수 있습니다. 테이블 분산 기법에는 라운드 로빈 방식과 표현식 기반의 분산 기법 두 가지가 있으며, 라운드 로빈 기법으로는 색인을 분할할 수 없습니다. 색인은 표현식을 통해 여러 DB영역으로 분할할 수 있습니다. 테이블에 있는 DB영역이 아닌 다른 영역으로 분할할 수도

있고, 앞의 스크립트에서 색인을 생성하듯이 테이블 내에 또는 테이블과 같은 DB영역으로 분할할 수 있습니다.

어떤 방법을 사용하든지 익스텐트 크기에 따라 분할과 분할이 있는 DB영역이 달라집니다. 테이블 크기를 정확히 지정해야만, 익스텐트가 인터리브되거나 데이터를 읽어 들일 영역 전체로 디스크 헤드가 이동하는 문제 등을 방지할 수 있습니다. 분할 테이블의 초기 익스텐트와 추가 익스텐트의 크기를 설정할 때는, 모든 테이블 분할이 똑같은 크기로 생성되고 확장되어야 한다는 점을 명심해야 합니다. 단, 다음 "참고" 부분의 내용에 따라 만들어진 분할 부분은 예외입니다. 익스텐트 크기는 가끔 사용하는 분할 테이블을 확장할 필요가 있을 때 지정해야 하며, 모든 DB영역을 차지하도록 지정해서는 안됩니다. 이 때 테이블이 차지하는 공간의 크기는 테이블 생성 구문에서 "next size" 매개변수를 사용하여 지정합니다.

참고

각 분할의 초기 크기를 서로 다르게 지정할 수 있습니다. 먼저 같은 스키마를 사용하여 두 개 이상의 "로그(rogue)" 테이블을 만듭니다. 이 때 초기 익스텐트와 추가 익스텐트 크기를 서로 다르게 합니다. 그런 다음 뒷부분에서 설명할 "alter fragment" 명령으로 각 로그(rogue) 테이블을 모두 하나로 묶어 새로운 테이블을 만듭니다. "alter fragment" 명령을 다시 사용하여 분할식을 원하는 대로 수정하면 다양한 크기의 여러 DB영역으로 분할된 테이블을 얻을 수 있습니다.

- 라운드 로빈 분할

라운드 로빈 방식으로 분할한 테이블로 로드된 자료를, 분할 명령을 사용하여 모든 DB영역에 똑같이 분산시킬 수 있습니다. 단, 테이블에 자료가 이미 저장되어 있으면 다시 분산되지 않습니다. 자료는 언로드, 삭제, 재로드하지 않는 한 계속 같은 위치에 있으며, 이 위치에서는 자료 흐름이 새로 삽입할 행으로 인식됩니다.

라운드 로빈 분할 방식의 장단점은 RAID 스트라이핑과 같습니다. 질의 시 색인을 사용하더라도 엔진

은 요청한 자료를 여전히 모든 DB영역에서 찾습니다. 따라서, 테이블 크기에 따라 다르기는 하지만, 라운드 로빈 방식으로 테이블을 분할하면 단일 DB영역에 놓는 것보다 성능이 향상됩니다.

주의

라운드 로빈 방식으로 테이블을 분할할 경우, 테이블에 생성된 모든 색인은 테이블 내에 있지 않아야 합니다. 테이블 내에 남아 있는 색인은 테이블처럼 분할되므로 색인 페이지가 분할 DB영역 전체에 걸쳐 순차적으로 분산됩니다.

그 결과, 엔진에서 필요한 정보를 찾을 때 전체 DB영역에 분산된 색인을 사용하게 되므로 성능이 떨어지게 됩니다. 따라서, 라운드 로빈 방식으로 분할한 색인과 테이블은 없어지지는 않지만 여러 DB영역으로 나뉘게 됩니다.

다음 구문을 통해 라운드 로빈 분할 방식으로 분할한 테스트 테이블을 재생성할 수 있습니다.

```
create table store_sales

        ( division   smallint,

          store smallint,

          category   char(3),

          sales_date date,

          amt_sold   decimal(8,2)

        ) fragment by round robin in
```

```
    dbs_9, dbs_10, dbs_11, dbs_12

    extent size 95000 next size 9500

    lock mode row;
```

익스텐트 크기를 임의로 설정하여 사용하더라도, 다른 DB영역으로의 색인 재배치뿐 아니라 네 개의 DB영역에 분산된 자료를 고려하여 초기 익스텐트와 추가 익스텐트 크기가 변경된 것을 알 수 있습니다.

- "표현식" 기반의 분산 기법

테이블이나 색인의 자료를 지역 규칙에 따라 여러 DB영역으로 나누는 방법을 "표현식" 기반의 분산 기법이라고 합니다. 이 규칙은 간단한 SQL 문을 사용하여 만들 수 있으며, 테이블 생성 시 적용하거나 나중에 적용할 수 있습니다.

이 글의 "표현식 계산" 부분에 나열된 자료형을 제외하면, 대부분의 테이블 열에서 분할식을 사용할 수 있습니다. 라운드 로빈 분할 방식과는 달리, 표현식을 사용하여 데이터가 저장된 테이블이나 색인을 분할하면 분할 규칙에 따라 행이 재분산됩니다. 이 때 데이터베이스의 로깅 모드와 이동할 행 수에 따라 트랜잭션이 길어질 수 있습니다. 분할식을 구성하는 칼럼에서 값이 변경된 행이나, 더 이상 기존의 DB영역에 저장할 가치가 없는 행은 다른 DB영역으로 이동시킬 수 있습니다.

표현식 분할 방식의 장점 중 하나는 테이블에 생성된 색인 수를 줄여 질의 응답 성능을 향상시킬 수 있다는 것입니다. 질의 최적화 기능은 질의를 처리할 때 테이블의 분할 표현식을 사용하여, 읽기 요구로부터 일부 DB영역을 제외시킵니다. 이렇게 하면 요구한 자료가 있는 DB영역들 중 표현식에서 제외된 DB영역은 검색하지 않습니다.

위에서 생성한 테스트 테이블에서 자료 영역이 여섯 개 영역으로 구분되어 있고, 이 중 실제로 활성화된 것은 네 개뿐이라고 가정합시다. 이러한 경우, 분할식을 사용하여 네 개의 활성 영역과 두 개의 비활성 영역을 각각 다른 DB영역으로 구분할 수 있습니다. 이렇게 하면 따로 분리시킨 영역의 자료뿐 아니라 그 영역의 색인도 같이 제외됩니다. 다음 구문을 통해 이러한 작업을 수행할 수 있습니다.

```
create table store_sales

        ( division    smallint,

          store smallint,

          category    char(3),

          sales_date date,

          amt_sold    decimal(8,2)

        ) fragment by expression

            division = 1 in dbs_9,

            division = 2 in dbs_10,

            division = 4 in dbs_11,

            division = 5 in dbs_12,
```

```
    (division = 3) or (division = 6) in dbs_13

    extent size 950 next size 95

    lock mode row;
```

이 경우, 익스텐트 크기는 세 개의 구분 영역과 여섯 개의 구분 영역을 수용하도록 설정되었습니다. 익스텐트 크기를 더 작게 설정하면 분할이 dbs_13 DB영역을 초과하지 않도록 할 수 있습니다. 위의 가상 DB영역 설계에서는, 다른 구분 영역의 분할이 해당 DB영역의 유일한 테이블 구조이므로, 익스텐트 크기가 작아서 야기될 수 있는 테이블 익스텐트 인터리브는 문제가 되지 않습니다. 익스텐트가 추가되면 서로 연결될 뿐입니다.

- 표현식 계산

분할식은 위에서 아래로, 오른쪽에서 왼쪽 방향으로 계산됩니다. 이들은 서로 독립적이어야 하지만, 그렇지 않을 때는 첫째 조건에 따라 행이 배치됩니다.

표현식 구문에서는 테이블에 있는 모든 칼럼을 사용할 수 있지만 BLOB 칼럼은 사용할 수 없습니다. 표현식에 serial, date, datetime 자료형을 사용하면 행 분산에 균형이 잡히지 않으므로, 뒤에 나올 "분할 삭제" 부분에서 설명할 경우를 제외하고는 이런 자료형을 사용하지 않는 것이 좋습니다. 분할식에서 이런 자료형을 사용하면 테이블에서 행을 재분산할 때 일반적인 기준에 따라 표현식을 다시 작성해야 하기 때문입니다. 또한 표현식에 사용된 date나 datetime 자료형을 계산이 가능한 integer 형으로 변환해야 하므로 표현식의 구문 해석 속도가 떨어지게 됩니다.

SQL 동등 연산자를 사용하여 mod를 포함하는 표현식 구문을 만들 수 있습니다. 표현식에서 mod 연산자를 사용한 것을 보통 "해시(Hash)" 표현식이라고 하며, 이 구문을 사용하면 자료가 저장되어 있는

새로운 분할 테이블의 행을 고르게 분산할 수 있습니다. 다음은 해시 표현식의 예입니다.

.

```
        .,

        cust_id integer

        ) fragment by expression

                mod(cust_id, 3) = 2 in dbs_8,

                mod(cust_id, 3) = 1 in dbs_7,

                mod(cust_id, 3) = 0 in dbs_6

          lock mode row;
```

분할 표현식은 테이블에 행이 추가되거나 행이 수정될 때마다 계산되므로 간단하게 만들어야 합니다. 표현식에서 범위를 사용할 때는 가장 제한적인 조건이 제일 먼저 오도록 나열해야, 첫 DB영역에 행이 집중적으로 배치되는 것을 방지할 수 있습니다. 예를 들면 다음과 같습니다.

.

```
        .,

        region_code smallint
```

```
) fragment by expression

        (region_code >= 200) in dbs_4,

        (region_code >= 150) in dbs_3,

        (region_code >= 100) in dbs_2,

        remainder in dbs_5;
```

위의 예에서는 "remainder" 속성을 사용했으나, 이 속성은 되도록 사용하지 않는 것이 좋습니다.
그 이유는 분할식의 다른 부분에서 요구한 자료가 있는 DB영역을 지정해도, "remainder" 절은 나머지 분할 영역에 필요한 자료가 있을 수도 있다는 의미를 포함하기 때문에, 해당 테이블에 대한 모든 질의에 대해 "remainder"로 지정된 분할 영역을 찾기 때문입니다. 결과적으로 질의 옵티마이저에서는 나머지 분할 영역에도 질의 매개변수와 일치하는 행이 있는지 항상 검색하게 됩니다.

"remainder"로 지정한 분할 영역의 크기가 클 경우에는 이것이 질의 성능을 저하시키는 요인이 될 수도 있습니다. 따라서 "remainder" 절을 사용하는 것보다, 다른 식에서 지정하지 않은 행은 다른 DB영역에 포함되도록 분할식을 작성하는 것이 좋습니다. 예를 들어, 앞에서 작성한 분할식을 다음과 같이 다시 작성하여 "remainder" 절을 삭제할 수 있습니다.

.

```
.,

        region_code smallint
```

```
) fragment by expression

        (region_code >= 200) in dbs_4,

        (region_code >= 150) in dbs_3,

        (region_code >= 100) in dbs_2,

        (region_code < 100) in dbs_5;
```

- 색인 분할

앞에서 언급했듯이 라운드 로빈 방식으로는 색인을 분할할 수 없지만 표현식으로는 가능합니다. 간단한 표현식을 작성하여 테이블이 있는 DB영역이 아닌 다른 영역에 색인을 간단히 생성할 수 있습니다. 이러한 작업을 흔히 색인을 "분할한다"고 합니다. 다음은 색인 분할의 예입니다.

```
create index ix_strsls_cat on store_sales (category)

        in idx_space_1;

    create index ix_strsls_dt on store_sales (sales_date)

        in idx_space_2;
```

색인을 테이블로부터 분할하면 성능이 상당히 향상될 수 있습니다.

색인을 분할할 때는 다른 DB영역에서 참조할 테이블이나 분할 테이블이 거의 없는 디스크의 DB영역에 색인을 두는 것이 좋습니다. 이렇게 하면, 색인을 사용하여 질의를 실행할 때 테이블이 있는 DB영

역에서 테이블이나 색인에 대한 읽기/쓰기 작업으로 인한 충돌이 없어져 작업 속도를 훨씬 높일 수 있습니다.

데이터베이스에서 자주 사용하는 테이블은 반드시 사용 가능한 디스크 축 주위에 있어야 하듯이, 주로 사용하는 색인도 사용 가능한 디스크에 분산되어 있어야 합니다.

색인을 분할하기 위해 표현식 기반의 분산 기법을 사용할 때는 색인 내의 모든 칼럼을 사용하여 표현식을 작성할 수 있으나, 이 경우에도 위에서 언급한 자료형 예외 규칙이 적용됩니다. 다음 예제를 참조하십시오.

```
create index major_idx on store_sales

        (division, store, category)

        fragment by expression

                division > 4 in idx_space_4,

                category = "A" or category =

                "F" in idx_space_5,

                division <= 4 in idx_space_1;
```

- 분할 변경

"alter fragment" 명령을 사용하면 언제든지 테이블이나 색인 분할을 변경할 수 있습니다.

이 명령을 사용하면 분할을 추가, 삭제, 수정할 수 있을 뿐 아니라 아예 없앨 수도 있습니다. "alter fragment" 명령을 사용하여 똑같은 스키마의 두 테이블을 하나의 분할 테이블로 조인하거나, 분할된 테이블을 사용하여 새로운 테이블을 만들 수 있습니다.

주의

로깅 모드를 사용하는 데이터베이스이거나 혹은 변경할 테이블이 사용 중일 때는 "alter fragment" 명령을 사용하지 마십시오. 테이블 분할 변경은 하나의 트랜잭션으로 실행되어 테이블을 배타적 모드에서 사용하게 되므로 테이블이 로크됩니다. 따라서 "alter fragment" 명령에 의해 이동할 자료의 양이 많을 경우에는 트랜잭션이 길어질 수도 있습니다.

분할이 변경되는 동안에는 "디스크 공간이 두 배로 늘어나는 현상"이 발생합니다. 테이블 변경 시 기존 분할 영역은, 새로운 분할 영역이 완전히 구축되고 색인이 모두 재동기화될 때까지 삭제되지 않고 원래의 위치에 계속 남아 있게 됩니다. 이는 분할 변경 트랜잭션 시 자료의 무결성을 보장할 수 있는 기능으로, 전상회복이 될 경우 새로운 분할 영역이 삭제되고 기존의 분할 영역이 다시 활성화됩니다. 그러므로 분할 영역을 변경하기 전에 먼저 해당 DB영역에 기존의 분할 영역과 새로운 분할 영역을 모두 수용할 만큼 충분한 여유 공간이 있는지를 확인해야 합니다.

- 분할 초기화, 추가, 수정

이 글의 도입 부분에서 만들었던 store_sales 테이블을 사용하여 alter fragment 명령을 살펴보겠습니다. 이 테이블을 분할된 테이블로 변경하려면 분할 방법에 따라 다음과 같은 구문을 사용합니다.

```
alter fragment on table store_sales

   init fragment by round robin in dbs_2, dbs_3, dbs_4;
```

또는

```
alter fragment on table store_sales

        init fragment by expression

                division = 1 in dbs_9,

                division = 2 in dbs_10,

                division = 4 in dbs_11,

                division = 5 in dbs_12,

                (division = 3) or (division = 6) in dbs_13;
```

참고

"alter fragment" 문에서는 "alter table" 명령과는 달리 여러 작업을 함께 수행할 수 없습니다. 따라서 테이블에 분할을 추가하고 같은 테이블에 있는 다른 분할식을 수정하려면 두 가지 구문을 따로 실행해야 합니다.

테이블을 분할되기 전 상태로 되돌리려면 다음과 같은 구문을 사용합니다.

alter fragment on table store_sales init in dbs_12;

다음은 분할 조건을 수정하는 구문입니다.

alter fragment on table store_sales

modify dbs_11 to ((division = 4) or (division = 3)),

modify dbs_13 to division = 6;

또한 분할 영역을 추가하는 구문은 다음과 같습니다.

alter fragment on table store_sales add store > 400 in dbs_11;

(표현식 기반 분산 기법)

alter fragment on table store_sales add dbs_7;

(라운드 로빈 분산 기법)

라운드 로빈 분산 기법으로 테이블을 분할하면 다른 DB 영역을 추가해도

이미 저장된 행이 재분산되지 않습니다.

단지 새로운 DB 영역에는 새로 추가되는 행이 기록됩니다.

참고

분할 조건을 수정하거나, 새로운 분할을 추가하는 표현식을 작성하여 분할 구문을 만들면 테이블 표현식 구문의 맨 끝 부분에 추가됩니다. 그러므로 분할을 변경하거나 추가한 다음, 자료가 고루 분산되고 표현식이 적절한 순서로 배열되도록 테이블의 dbschema를 작성해야 합니다. 표현식을 적절한 순서로 배열하려면 일련의 분할 추가, 수정, 삭제 명령을 사용해야 합니다.

-분할 삭제

분할을 삭제할 때는 해당 분할 영역에 저장된 자료를 어느 위치로 옮길 것인지를 생각해야 합니다. 기본적으로는 자료가 "remainder"로 지정한 나머지 분할 영역 으로 옮겨집니다. 그러나 앞에서 언급했듯이, 질의 성능이 떨어질 수 있으므로 자료를 나머지 분할 영역에 배치하는 일은 되도록 피해야 합니다.

결국 테이블 분할을 삭제하는 작업은 처음 인스턴스를 생성할 때 rootdb에서 논리 로그를 다른 위치로 옮기는 것과 별 차이가 없습니다. 따라서 삭제할 분할 영역에 있는 행을 옮길 수 있도록 기존 분할식을 변경하거나 새 분할을 추가한 다음 분할을 삭제해야 합니다.

새로 추가하거나 변경한 분할 조건에서 삭제할 분할 영역의 행이 다른 곳에 저장될 수 없게 하면 "-776" 오류와 "-772" 오류가 발생하여, 분할 삭제 트랜잭션이 전상으로 회복됩니다. 다음은 분할을 삭제하는 코드의 예입니다.

alter fragment on table store_sales drop dbs_12;

- 테이블 Attach

SQL 명령인 "attach" 명령으로 스키마가 동일한 둘 이상의 테이블을 조인하여 하나의 테이블을 생성할 수 있습니다.

이렇게 생성한 테이블은 연결하기 전에 있던 행들이 모두 원래의 테이블에 그대로 유지되므로 라운드 로빈 방식으로 분할되어 있습니다.

분할을 변경하려면 처음부터 새로운 분할 방법을 사용하거나 "alter fragment" 문에서 "init" 키워드를 사용합니다.

"attach"된 테이블의 분할 방법을 라운드 로빈 방식에서 표현식 기반의 분산 기법으로 바꾸면 다음에 설명되어 있듯이 다른 테이블을 attach할 수 없게 됩니다. 테이블을 다른 테이블에 attach시켜도 각 테이블의 원래의 익스텐트 크기 매개변수는 그대로 유지됩니다. 그러므로 테이블을 attach하기 전에는 분할 크기가 늘어나도 DB영역의 공간이 모자라지 않는지 확인해야 합니다.

다음은 테이블을 attach시키는 구문입니다.

alter fragment on table store_sales attach store_sales, midwest_sales;

위의 테이블 attach 구문에서 둘째 테이블(midwest_sales)은 "보조" 테이블이 되어 ID를 잃게 되고, store_sales는 "기본" 테이블이 되어 원래의 이름을 그대로 갖게 됩니다.

테이블을 attach시킬 때는 다음 사항을 주의해야 합니다.

- 테이블을 attach시킬 때는 먼저 생성된 제한 조건을 모두 지워야 합니다. 제한 조건이 있는 테이블을 attach시키면 SQL "-888" 오류가 발생합니다.
- 기본 테이블의 분할 방법인 라운드 로빈 방식(기본값)을 표현식 기반의 방식으로 바꾸면 SQL "-782" 오류가 발생하여 테이블을 더 이상 attach할 수 없습니다. 그 이유는 새로운 분할 작업을 판단할 표현식 구문이 없기 때문입니다.
-
- 비록 다른 칼럼이라도 두 테이블에 모두 색인이 있으면 연결한 후에는 보조 테이블의 색인은 삭제되고, 기본 테이블의 색인만 남게 됩니다.

  - 분할 Detach

  기본" 테이블에서 분할된 테이블로 "detach"된 테이블을 새로 생성할 수 있습니다.
  표현식을 사용하여 기본 테이블을 분할하면 분할 영역에 포함된 행의 범위를 지정할 수 있으며, 분할을 detach하여 특정 분할 영역에서 좀더 중요한 자료를 처리할 수 있습니다.

  물론 분할에서 사용된 표현식에 따라 달라질 수는 있으만, 분할을 detach하는 것은 스키마가 같은 테이블로서 이미 기한이 지났거나 따로 보관하고자 하는 자료만을 두는 새로운 테이블(historical table)을 만들기에 아주 쉬운 방법입니다. 이 경우에는 분할식에서 date나 serial 자료형을 사용하는 것이 좋습니다. 테이블에서 date 칼럼을 사용하여 분할식을 만들 수 있습니다. 분할 영역에 특정 날짜보다 최근의 날짜를 포함하는 행이 추가되거나, 분할 영역이 지정된 크기보다 커지면 앞으로 추가되는 행을 기록할 새로운 날짜를 범위로 하는 분할이 생성됩니다. 그런 다음 비활성화된 분할이 기본 테이블에서 분리되어 historical 테이블이 됩니다.

  detach된 테이블에서는 기본 테이블의 익스텐트 크기와 로크 모드 매개변수만 그대로 유지될 뿐, 그 외의 정보는 모두 상실됩니다. 색인, 키, 제한 조건 등은 분리된 테이블에서 새로 생성해야 합니다. 테이블 분리 구문은 분리할 분할이 있는 DB영역의 이름과 새로운 테이블 이름만 지정하여 간단히 작성할 수 있습니다. 다음 예제를 참조하십시오.

```
alter fragment on table store_sales detach dbs_12 div3_sales;
```

제한 조건(Constraints), 참조 무결성(Referential Integrity), 색인

데이터베이스와 테이블을 만든 다음에는 데이터베이스에 있는 자료의 무결성을 유지하는 것이 무엇보다 중요한 DBA의 임무입니다.

자료 무결성이란 자료에 대한 상호 의존 관계나 의미상의 규칙을 올바르고 정확하게 저장하는 것을 의미합니다.

내장 프로시저와 여러 가지 제한 조건은 데이터베이스의 성능 향상뿐 아니라 이러한 무결성 보장과 유지에도 주요 역할을 합니다. 자료의 무결성을 유지하는 일도 DBA가 수행해야 할 작업의 일부이므로 제한 조건과 색인의 정의와 그 기능에 대해 살펴보겠습니다.

내장 프로시저

내장 프로시저는 SQL과 SPL 언어를 조합하여 만든 함수입니다.

프로시저는 각 인스턴스의 데이터베이스 엔진에 컴파일된 형태로 저장되며, 보안, 테이블 로그 작업, 자료 처리, 데이터베이스 성능 향상 등의 적을 위해 사용됩니다.

내장 프로시저의 일반적인 구문은 C 언어와 비슷하지만, SPL 언어에는 변수 지정 명령과 "if-then-else", "while", "for" 문 같은 기본 프로그램 제어 구문 이상의 논리는 없습니다. 내장 프로시저의 나머지 기능들은 SQL 구문에서 유래됩니다.

프로시저는 최종 사용자 응용프로그램이나 데이터베이스 엔진 내에서 작성되고 저장된 작업 방식의 트리거에 의해 호출되고 실행됩니다.

트리거도 SQL과 SPL을 함께 사용한 내장 프로시저 형태로 작성되며, 내장 프로시저와 유사한 기능들

예를 들어, 테이블의 행이 수정될 때마다 또는 테이블의 행에서 특정 칼럼이 수정될 때마다 지정한 작업이 실행되도록 트리거를 작성할 수 있습니다. 데이터베이스에서 트리거에 연결된 특정 작업이 발생하면 트리거는 해당 논리를 실행합니다. 또는 트리거를 실행하여 훨씬 강력한 프로그래밍 구문을 가진 내장 프로시저를 호출할 수도 있습니다.

트리거와 내장 프로시저에 논리를 포함시킬 수 있으므로, 응용프로그램보다는 데이터베이스 수준에서 경영 원칙을 적용할 수 있습니다. 예를 들어, 영업상의 순이익을 계산하는 질의 구문을 사용자 응용프로그램으로 작성하지 않고 내장 프로시저로 만들어 데이터베이스 엔진에 보관하면, 응용프로그램 코드가 훨씬 간단해질 뿐 아니라 일관된 결과를 얻을 수 있습니다. 삽입 트리거와 내장 프로시저를 함께 사용하여 여러 테이블에 상호 참조 정보를 넣은 다음 행을 특정 테이블에 삽입할 수도 있고, 트리거와 내장 프로시저를 수정하여 제품 테이블에서 제품의 가격이 변경될 때마다 주문품의 총액을 자동으로 다시 계산하도록 할 수도 있습니다.

제한 조건(Constraints)과 색인

제한 조건과 색인을 사용하여 질의 성능을 향상시키고, 자료 요소 간의 관계를 유지하고, 자료의 값 범위를 정의할 수 있습니다.
참고로, 제한 조건은 Informix Dynamic Server 엔진 버전 5.01부터 포함된 기능입니다.

고유 색인, 기본 키, 단일 제한 조건의 차이는 두 가지로 요약할 수 있습니다. 첫째, 논리 데이터베이스 설계라는 측면에서 보면 이들이 수행하는 기능이 각각 다릅니다. 둘째, 이 세 가지 모두 색인에 의해 처리되지만 Informix Dynamic Server 엔진에서는 색인을 키나 제한 조건과는 다른 방법으로 처리합니다. 키, 제한 조건, 색인을 데이터베이스 설계 측면에서 살펴 보면 다음과 같습니다.

- 제한 조건은 테이블에서 경영 원칙을 적용하는 데 사용됩니다. 제한 조건은 크게 두 가지로 나눌 수 있는데 첫째는 행의 고유성을 처리하는 제한 조건입니다. 예를 들어, 고객 테이블에서 고객 ID는 반드시 고유해야 하며, 주소 테이블에서는 고객 ID와 납품 주소의 조합이 고유

해야 합니다. 이러한 제한 조건을 처리하려면 색인을 작성해야 하므로 이것을 "색인 방식(index-based)"의 제한자라고 합니다. "check" 제한자라고 하는 또 하나의 제한 조건은 칼럼에 대한 값의 범위를 지정하거나 유추할 때 사용합니다. 예를 들어, 가격 필드의 값이 최저 주문 가격인 5,000원 이상이 되게 하거나, 유지 보수 비용 필드의 값을 제품 가격의 18%로 설정할 수 있습니다. 이런 종류의 제한 조건은 색인으로는 처리할 수 없으며 엔진에 제한 조건을 처리할 수 있는 규칙을 추가해야 합니다.

- 기본 키와 외부 키는 테이블 간의 참조 무결성과 자료 관계를 정의하는 데 사용됩니다. 이러한 관계를 상위-하위(parent-child) 테이블 관계라고 합니다. 이 키를 사용하면 상위 테이블과 상호 참조 관계가 있는 하위 테이블에만 자료를 추가할 수 있으며, 하위 행이 존재할 경우에는 상위 테이블에서 해당 자료를 삭제할 수 없습니다.

이와 같은 예는 주문 처리 시스템에서 찾아 볼 수 있습니다.시스템에서 각 고객은 고유한 고객 ID(기본 키)로 구분되며, 주문 테이블의 각 행에는 고유한 주문 번호(기본 키)뿐 아니라 유효한 고객 ID(외부 키)가 있어야 합니다.또한 주문 항목 테이블에는 테이블에 있는 주문 라인 항목에 대한 유효한 주문 번호(외부 키)가 있어야 합니다.이 예에서 보듯이, 테이블에는 기본 키와 외부 키를 모두 지정할 수 있습니다. 위의 세 테이블에서 한 행을 삭제할 때 이 행과 관련된 다른 행을 삭제하지 않으면 전체 주문 시스템에 문제가 생깁니다.

참고

상위 테이블에서 행을 삭제할 수 없도록 지정하면 하위 테이블의 행이 연속적으로 삭제되는 것도 방지할 수 있습니다. Informix Dynamic Server에서는 기본적으로 하위 테이블에서 상위 행을 참조할 경우, 상위 테이블의 행을 삭제할 수 없도록 되어 있습니다. 그러나 하위 테이블의 외부 키 제한 조건 구문에 "on delete cascade"를 추가하면, 상위 테이블에서 행을 삭제하는 경우 해당 하위 테이블에 있는 관련 행이 모두 삭제됩니다. 이 기능을 사용하면 응용프로그램에서 코딩 오버헤드를 줄일 수 있습니다.

상위 행을 삭제하려면 먼저 하위 행을 모두 삭제해야 합니다. 상위 테이블의 행에서 삭제 구

문을 실행할 때 하위 테이블 중 하나라도 연속 삭제가 비활성화되어 있으면, 삭제 트랜잭션이 전상회복되어 어떤 행도 삭제되지 않습니다.

- 엔티티로서의 색인은 질의 성능을 향상시키는 목적으로만 사용합니다. 따라서 테이블에서 고유 색인을 단독 엔티티로 작성해서는 안됩니다. 고유성이 요구되면 이는 경영 원칙이므로 제한 조건으로 코딩해야 합니다. 위에서 언급했듯이, 일부 제한 조건은 색인 구조를 통해 처리됩니다. 질의 옵티마이저에서는 이러한 제한 조건을 지원하는 색인을 사용하여 질의 성능을 향상시킵니다. 이는 제한 조건의 보조 기능이므로 경영 원칙 외의 다른 질의 성능을 향상시킬 필요가 있을 때는 제한 조건 대신 색인을 사용합니다.

Informix Dynamic Server에서는 제한 조건이 색인과는 다르게 처리되므로, 이는 결과적으로 SQL 연산 방법과 응용프로그램 작성 방법에 영향을 줍니다.

그러므로 이러한 차이를 이해하고 응용프로그램 개발자를 돕는 것도 Informix Dynamic Server 관리자의 역할입니다.

색인과 제한 조건의 기능적 차이는 다음과 같습니다.

- 색인은 트랜잭션 상태나 엔진의 로깅 모드에 상관없이 즉시 처리됩니다.
- 제한 조건은 트랜잭션이 끝날 때 처리됩니다.

예를 들어, 고객 정보 테이블에 정수형의 고객 ID 칼럼이 있고, 1부터 100까지 차례대로 번호가 매겨진 행이 있을 때, 처음 10개의 고객 ID 번호를 사용하지 않고 남겨 두려면 다음 SQL 문을 사용합니다.

```
begin work;
```

lock table customer in exclusive mode;

update customer set cust_id = (cust_id + 10);

cust_id 칼럼에 고유 색인이 있는 경우 첫 행을 갱신하려고 하면 오류가 발생합니다.

이는 첫 행의 cust_id가 11로 변경되지만, 색인 구조에 cust_id 11번이 이미 존재하므로 행을 변경할 수 없기 때문입니다.

그러나 cust_id 칼럼이 모두 변경된 후라면 11부터 110까지의 값을 가지게 되므로 유일한 키 값이어야 한다는 제한 조건에 위배되지 않을 것입니다. 이처럼, cust_id 칼럼에 단일 제한 조건이 있는 경우에는 모든 행이 새로운 값으로 갱신된 다음에 고유성 확인 작업이 실행되므로 트랜잭션을 완료할 수 있습니다. cust_id 칼럼이 다른 테이블의 상위 칼럼일 때는 하위 테이블이 먼저 갱신된 다음 cust_id 칼럼이 갱신됩니다. 그러므로 트랜잭션을 완료하려면 제한 조건을 "지연 모드(defered mode)"로 설정해야 합니다.

기본적으로, 제한 조건은 트랜잭션이 끝날 때 처리됩니다. 이는 앞에서 예로 든 cust_id 갱신과 같은 단일 문으로 된 트랜잭션에서는 문제가 없습니다. 그러나 다중 문이 단일 트랜잭션에서 실행될 때는 트랜잭션에서 add, update, delete 문이 종결될 때마다 제한 조건이 처리됩니다.

상위 테이블과 하위 테이블을 갱신해야 하는 경우, 각 문이 끝날 때마다 제한 조건을 처리하면 테이블에 아직 갱신되지 않은 자료가 남아 있게 되므로 트랜잭션이 전상회복됩니다. 이러한 문제는 트랜잭션을 시작할 때 "set constraints deferred" 절을 사용하여 해결할 수 있습니다. 제한 조건이 지연되면 제한 조건이 다중 문으로 된 트랜잭션이 끝날 때 처리됩니다. 이 때 제한 조건은 해당 문이 포함된 트랜잭션이 끝날 때까지만 지연할 수 있습니다. 트랜잭션이 완료되거나 전상회복되면 제한 조건이 "즉시 모드(immediate mode)"로 재설정됩니다.

주의

제한 조건을 삭제할 때는 제한 조건에 의해 생성된 색인이나, SYSCONSTRAINT 테이블에 있는 제한 조건에 대한 정보를 삭제하지 마십시오. 반드시 "alter table" 문을 사용하여 제한 조건을 작성할 때 사

용자가 지정하거나 인스턴스에서 자동으로 생성된 제한 조건 이름을 명시해야 합니다. 다른 방법으로 제한 조건을 삭제하면 치명적인 오류가 발생하여, 결국은 백업본에서 테이블을 복원해야 하는 경우가 생깁니다.

제한 조건 분할

제한 조건 생성 구문에는 색인과 같이 분할에 대한 규칙이 따로 없습니다. 이는 제한 조건을 처리하기

이터베이스에서 색인과 제한 조건을 쉽게 분리할 수 있도록 데이터베이스 스키마를 작성할 수 있습니다.

데이터베이스에 자료 입력

데이터베이스를 구축하고, 테이블을 생성/분할하고, 제한 조건과 내장 프로시저를 사용하여 처리할 경영 원칙을 수립한 다음에는, 자료를 입력해야 합니다. 이러한 작업을 수행하는 여러 가지 유틸리티가 있으며, 주어진 상황에 적합한 유틸리티를 선택하는 것이 중요합니다. 다음에서 유틸리티들에 대해 알아보도록 하겠습니다.

dbimport dbimport 유틸리티는 이미 위의 데이터베이스 생성 부분에서 설명된 바 있습니다. dbimport는 dbexport와 함께 사용되는 유틸리티로서, 데이터베이스 환경을 구축한 다음 데이터베이스의 각 테

니다. 로드 플래그는 "***load table***"과 같은 형식을 취합니다. 색인 및 제한 조건 생성 문에서 스키마 파일을 분리하지 않도록 지정하면, 색인 생성 문 다음에 있는 이 로드 플래그를 앞으로 이동하여 색인 없이 테이블을 로드할 수 있습니다.

dbimport 유틸리티의 또 다른 이점은 트리거와 내장 프로시저가 파일 임포트에서 마지막에 생성된다는 점입니다. 따라서 테이블 로드 중 트리거와 내장 프로시저를 실행해도 오버헤드가 생기지 않지만, 실제적으로는 임포트한 자료가 원래 데이터베이스의 내장 프로시저에 의해 이미 처리되었으므로 이를 실행할 필요가 없습니다.

SQL "load" 문

SQL "load" 문을 사용할 때는 앞에서 설명한 유틸리티들과 같은 유연성을 거의 기대할 수 없습니다. 이 명령은 로그되는 데이터베이스에서 중소형 크기의 자료에 사용되거나, 로그되지 않는 데이터베이스의 대형 자료에 사용됩니다. 배타적 모드로 테이블을 로크하지 않으면 테이블 로드 중에도 테이블에 접근할 수 있습니다. load 명령은 구분 문자로 분리된 ASCII 파일을 읽어 들여 칼럼 순서에 따라 대상 테이블로 로드합니다. 대상 테이블의 색인 수에 따라 이 명령을 사용하여 자료를 빨리 로드할 수 있습니다.

"load" 명령을 사용하는 데는 몇 가지 잠재적인 문제점이 있습니다. 로그되는 데이터베이스에서 로드 작업은 하나의 트랜잭션으로 처리되므로, 로드할 자료의 양에 따라 트랜잭션이 길어지거나 로크 테이블 오버플로가 발생할 수도 있습니다. 테이블을 배타적 모드에서 로크하여 문제가 생기는 것을 방지할 수도 있으나 이렇게 되면 테이블에서 다른 작업을 할 수 없습니다.

또한, 자료 파일에서 예외 상황을 처리할 메커니즘이 없는 점도 문제가 됩니다. 만일 숫자 칼럼으로 로드할 필드에 문자가 있는 경우, 엔진에서 자료형 변환 오류가 발생하여 로드 작업이 중단됩니다. 이 유틸리티에서는 로드 파일에서 문제가 있는 행을 쉽게 찾을 수 있도록 로드된 행 수를 표시해 줍니다.

"dbload" 유틸리티

"dbload" 유틸리티를 사용하면 ASCII 파일로 보다 유연성 있게 작업할 수 있습니다. "dbload" 유틸리

티의 기능은 다음과 같습니다.

- 구분 문자로 분리되거나, 구분 문자로 분리되지 않고 고정 길이를 사용하는 파일을 로드합니다.
- 테이블 순서대로 칼럼을 로드합니다.
- 로드를 시작하기 전에 로드 파일에서 N개의 행을 건너뛰고 로드합니다.
- 로드 과정에서 잘못된 행이 있을 때는 나중에 수정할 수 있도록 이를 로그 파일에 기록합니다.
- ASCII BLOB을 로드합니다.
- X개의 행마다 COMMIT WORK 문을 실행하여 트랜잭션이 길어지는 것을 방지합니다.
- NULL이 아닌 칼럼에 NULL 대신 값을 넣습니다.

  "명령" 파일을 사용하여 "dbload" 유틸리티와 그 기능을 제어할 수 있습니다.

  이 파일에는 로드할 파일 이름, 대상 테이블, 칼럼 순서, 처리 명령문 등이 기록됩니다.

  또한 몇 개의 행에 오류가 발생할 때 로드 프로세스를 중단할지 지정할 수 있습니다.

  dbload 유틸리티를 사용할 때는 로드 프로세스 중 일정 간격으로 완료 문을 실행할 수 있으므로 테이블을 배타적 모드로 로크할 필요가 없습니다. 그러나 테이블에 색인이 있을 경우에는 "load" 명령에서와 같은 성능 상의 문제가 발생합니다.

"onload" 유틸리티

onload 유틸리티는 자료를 가장 빨리 로드할 수 있는 유틸리티 중의 하나입니다. 이 유틸리티는 onunload 유틸리티와 함께 사용되며, 테이프로부터 테이블의 이진 복사본을 읽어 들여 요청한 DB영역에 테이블과 색인을 배치합니다.

또한 DB영역 생성 매개변수를 설정하여 색인의 이름을 변경하거나 테이블 및 색인 분할 DB영역을 새로 배열할 수 있습니다.

므로 로드할 자료 및 자료의 순서를 변경하거나 수정할 수 없습니다.

onload 유틸리티를 사용하면 내보낼 때의 총 테이블 익스텐트 크기뿐 아니라 원래의 익스텐트 정보가 그대로 유지됩니다. 따라서 DB영역에서 테이블 익스텐트 수를 줄여야 할 때 유용합니다. onunload 유틸리티를 사용하여 모든 테이블을 테이프로 복사하고, 원래의 테이블을 삭제한 다음 onload 유틸리티를 사용하여 테이프에 복사해 둔 테이블을 다시 로드합니다. 이 작업이 완료되면 전체 테이블이 DB영역에서 하나의 익스텐트를 차지하게 됩니다.

고성능 병렬 로더(High Performance Parallel Loader)

Informix Dynamic Server 버전7.2에는 "onpload"라는 로드 유틸리티가 새로 추가되었습니다. 이 유틸리티를 사용하여 다양한 종류의 자료 파일을 처리할 수 있으며, 로드 프로세스 중 자료 관리를 용이하게 할 수 있고, 로드 속도를 높일 수도 있습니다.

onpload를 사용하여 테이블을 언로드하거나 로드할 수 있습니다. 로드 모드에서는 다른 사용자의 테이블 접근을 허용할 수도 있고(Deluxe Mode), 테이블을 배타적 모드로 로크할 수도 있습니다(Express Mode). 익스프레스 모드에서 테이블을 로드할 때는 레벨 0 백업을 해야 테이블을 사용할 수 있으며, 두 경우 모두 로드가 완료될 때 색인이 재구축되고 제한 조건이 처리됩니다.

입력 파일은 디스크나 테이프를 사용할 수 있으며, 형식을 변환(예: EBCDIC을 ASCII로)할 수 있고, 로더를 통해 로드할 때 자료를 걸러낼 수도 있습니다. 또한 자료 요소를 결합하거나 해석하여 다른 테이블로 로드할 수도 있습니다.

4GL 응용프로그램을 통한 평면 파일 로드

테이블을 로드할 때 4GL 응용프로그램을 사용하는 것은 앞에서 설명한 유틸리티들을 사용하는 것보다 사실 적합한 방법이 아닙니다.

Informix 4GL은 자료 로드 프로그램을 작성하는 데 사용되긴 하지만, 원래는 강력한 오류 처리 기능과 자료 조작 기능을 제공할 수 있는 언어입니다.

그러나 ASCII 파일로부터 데이터를 읽고, 필요 없는 자료를 지능적으로 삭제하며, 테이블에 자료를 입

력하는 데는 어떤 유틸리티보다 기능이 월등하므로, onpload 유틸리티 대신 Informix 4GL을 사용하는 것이 좋습니다.

어떤 Informix 시스템 엔지니어가 두 개의 명령을 우연히 발견했는데, 이 명령이 바로 4GL 응용프로그램을 통해 ASCII 파일의 자료를 직접 읽을 수 있도록 하는 명령이었습니다. fglgets는 파일 이름을 매개변수로 사용하며 열려 있지 않은 파일을 열고 파일의 첫째 행을 되돌리는 기능을 합니다. 파일이 이미 열려 있으면 다음 행을 되돌립니다. fglgetret 명령은 실제적으로 파일로부터 행을 가져 오는 C 언어 팝(pop) 명령의 리턴 코드를 확인합니다.

명령이 구현된 코드를 예로 들어 보면 그 기능을 쉽게 이해할 수 있을 것입니다. 다음을 참조하십시오.

```
Database something_or_another

main

  define fname varchar(30)

  define in_str varchar(255)

  define nrecs integer

      ## please be aware that no error handling is included

      ## in this snippet of code.

  prompt "Please enter pathed file name to load:
```

```
" for fname

let fname = fname clipped

                ## open file and get first row

call fglgets(fname) returning in_str

                ## check the status of the open and "pop"

while fglgetret() = 0

   let nrecs = nrecs + 1

   if nrecs mod 100 = 0 then

      display "Loading row: ", nrecs using

      "###,###" at 8,2

   end if

   call process_row(in_str)

      ## continue to get rows until EOF

   call fglgets(fname) returning in_str
```

```
    end while   ## fglgetret = 0

end main
```

다음은 4GL을 사용하여 ASCII 파일을 로드할 때 참고할 사항입니다.

- fglgets와 fglgetret를 사용하려면 fglgets.c 파일을 오브젝트 파일 형태로 컴파일하여 makefile에서 참조하는 다른 표준 라이브러리와 함께 포함시킵니다.
- fglgets 코드는 파일을 열기만 하고 닫지는 않습니다.
- fglgets는 파일을 아래 방향으로만 읽을 수 있으며, 포인터를 윗방향으로 되돌리는 것이 불가능합니다("C" 언어에서는 가능).
- 문자열은 256바이트까지 읽을 수 있습니다.
- Informix 코드로는 한 번에 파일을 8개까지 열어 처리할 수 있으며, 그 이상의 파일을 처리하려면 라이브러리를 사용하는 응용프로그램을 종료한 다음 재시작해야 합니다. 이는 코드 내의 MAXOPEN 매개변수에 의해 제어되며 필요에 따라 변경할 수 있습니다.
- 새 파일 이름으로 fglgets를 호출할 때마다 실제로 존재하지 않거나 열 수 없는 파일일 경우에도 모두 MAXOPEN 합계에 추가됩니다.
- 열린 파일의 수가 MAXOPEN을 넘으면 오류 메시지와 자료가 응용프로그램으로 되돌려지지 않습니다. 응용프로그램측에서 보면 마치 파일이 비어 있는 것처럼 보입니다. 그러므로 MAXOPEN 값을 늘리거나 MAXOPEN 값만큼 파일을 연 후에는 더 이상 파일을 열 수 없도록 설정해야 합니다.
- 호출된 함수에서 fglgets를 호출하면 MAXOPEN 카운터에 영향을 주지 않습니다. 카운터는 모듈 수준이 아니라 응용프로그램 수준에서 실행됩니다.

병행성 및 격리 수준

데이터베이스를 구축하고 자료를 입력한 다음에는 자료에 대한 여러 사용자의 접근을 제어하고 보호해

야 합니다.

자료에 대한 보호는 Informix Dynamic Server 인스턴스에서 로크를 사용함으로써 구현할 수 있습니다. 접근 제어는 대부분 응용프로그램에서 설정하며, 인스턴스에서 다양한 종류의 로크를 실행하는 방법과 응용프로그램에서 로크된 자료를 요구할 때의 인스턴스 반응 방법을 결정합니다. 다음에 간단히 설명되어 있듯이 이런 방법들은 데이터베이스의 로깅 모드에 따라 결정됩니다.

병행성이란 여러 사용자가 동시에 자료에 접근하는 것을 의미하며, 격리 수준이란 데이터베이스에 작업 요청 중 다른 작업 요청을 허용하는 정도를 말합니다. 이 두 가지의 디자인 및 구축 방법은 상호 보완적입니다. 격리 수준을 제한할수록 데이터베이스에 동시에 접근할 수 있는 사용자의 수가 줄어듭니다. 동시에 여러 사용자가 접근하려면 격리 수준이 낮거나, 응용프로그램의 쓰기 트랜잭션이 작고 쉴 틈 없이 수행되거나, 응용프로그램이 여러 수준으로 격리되어야 합니다.

로크의 유형 및 모드

Informix Dynamic Server의 로크 유형은 다음의 세 가지로 나눌 수 있습니다.

- **Exclusive(배타적 로크)**

  배타적 모드로 자료 요소를 로크하면 다른 사용자들은 여기에 접근할 수 없습니다. 갱신 및 삭제 연산을 할 때는 행 단위로 로크가 되며, SQL 문을 사용하여 사용자 명령을 수행할 때는 테이블이나 데이터베이스에 로크됩니다.

사용자는 SQL 문을 사용하여 공유 로크나 배타적 로크를 테이블이나 데이터베이스에 설정합니다. Informix Dynamic Server는 주요 DDL 명령(예: "alter/rename/drop table", "create/drop index", "rename column") 실행 시 배타적 테이블 로크를 호출합니다. 인스턴스에서는 사용자가 별도의 로크 명령문으로 DDL 작업을 시작하지 않으면 작업이 완료되자마자 로크를 해제합니다. 사용자가 시작한 로크는 수동으로 삭제하지 않는 한, 로그되거나 로그되지 않는 데이터베이스에서 트랜잭션이 끝날 때까지 계속 유지됩니다.

격리수준설정

격리 수준에 따라 인스턴스에서 읽기 요구를 처리하는 방법이 달라집니다.

이 수준은 동시 작업이나 여러 사용자의 자료 접근에 영향을 줍니다. 격리 수준에는 네 가지가 있습니다.

- RR(Repeatable Read)

  가장 제한적인 격리 수준으로, 모든 행이 전체 트랜잭션 동안 공유 로크로 잠깁니다. 색인이 없는 읽기(non-indexed read)나 순차적 스캔을 통해 읽은 행에도 로크가 설정되어 SELECT 문에 대한 적합성이 평가됩니다.

  공유 로크는 읽으려는 행에 대해 트랜잭션이 진행 중일 때만 설정됩니다. 즉, SELECT문을 실행했을 때, 격리 수준을 RR로 설정하는 것만으로 이 격리 수준에서 제공하는 보호 기능이 실행되지는 않으며, 실제로 트랜잭션이 진행 중이 아니면 다른 프로세스로 인해 행이 변경될 수도 있습니다.

  MODE ANSI 데이터베이스의 기본 격리 수준입니다.

- CS(Cursor Stability)

  RR 격리 수준과 마찬가지로 커서가 있는 행에 공유 로크가 설정되나, 커서 문에서 다음 자료 행을 요청하면, 이전에 읽은 행의 로크는 해제됩니다.

  또한, RR 격리 수준처럼 커서가 있는 행에 대해 트랜잭션이 진행 중일 때만 공유 로크가 설정됩니다. 이 격리 수준은 SQL 구문으로 설정할 수 있지만, I-SQL이나 dbaccess 세션에서는

사용할 수 없으며, I-4GL로 작성한 커서와 같이 응응프로그램에 선언된 커서에만 적용됩니다.

- CR(Committed Read))

  MODE ANSI를 사용하지 않는 로깅 데이터베이스에서 사용되는 기본 격리 수준으로서, 현재 트랜잭션이 완료된 행만 읽을 수 있습니다.

  응용프로그램에서 읽기를 요청하면 데이터베이스 서버는 대상 행에 공유 로크가 설정되어 있는지 먼저 확인합니다. 실제로 행이 로크되지는 않으므로 DR 격리 수준과 비슷한 속도로 작업할 수 있습니다.

  해당 행에 대해 트랜잭션이 진행 중이라도 다른 프로세스에서 변경 중인 자료는 볼 수 없으며, 자료가 UPDATE, INSERT, DELETE 작업 때문에 로크된 경우에도 읽을 수 없습니다.

- DR(Dirty Read)

  가장 낮은 격리 수준으로서 트랜잭션이 완료되지 않은 행을 포함하여 기존의 행을 모두 읽을 수 있습니다.

  여기에는 진행 중인 트랜잭션으로 인해 행이 갱신되거나 삽입되어 값이 변경될 수 있는 행까지 모두 포함됩니다.

  이 격리 수준에서는 트랜잭션이 전상회복되거나 다른 사용자의 작업으로 인해 행이 변경되거나 삭제될 위험이 있습니다.

이러한 수준을 설정하려면 "set isolation" 명령을 사용합니다. 읽어 들일 자료의 무결성이나 응용프로그램에서 특정 기능을 수행할 때 필요한 제한에 따라, 다양한 격리 수준이 사용됩니다.

변경하지 않는 참조 자료의 경우에는 DR(Dirty Read)로 설정해도 됩니다. 중요한 자료 요소를 요구할 때는 동시 접근의 필요성에 따라 CS(cursor-stability read)나 RR(Repeatable Read)로 설정하여, 응용프로그램에서 사용하는 동안 외부 작업으로 인해 자료가 변경되는 것을 방지해야 합니다.

OnLine SQL 문

Informix Dynamic Server 버전 7.1.UD1이 출시되면서부터 몇 가지의 SQL 문이 새로 소개되었고, 기존의 다른 SQL 문들도 다소 수정되었습니다.

. 여기서는 가장 중요한 몇 개의 명령문에 대해서 살펴보겠습니다.

위배(Violation) 및 진단(Diagnostics),
제한 조건(Constraint) 및 색인 설정, 필터링(Filtering)

7.1.UD1 이전 버전에서는 고유 색인이나 색인 방식의 제한 조건이 있는 테이블에 행을 삽입 또는 갱신할 때 오류가 발생하면 수동으로 처리해야만 했습니다.

즉, 발생한 오류와 해결 방법을 sqlca.sqlerrd 코드를 트랩하여 알아 낸 다음, 오류가 발생한 행은 보류 테이블로 옮기고 수동으로 처리해야 했습니다.

그러나 이제는 엔진에서 이러한 작업을 대부분 처리할 수 있게 되어 응용프로그램에 필요한 오류 처리 코드가 줄었습니다. 이는 특정 대상 테이블에 위배 및 진단 테이블을 설정한 다음, 고유 색인과 색인 방식의 제한자에 대해 새롭게 추가된 오류 처리 기능을 사용함으로써 가능합니다.

"start violations table" 문을 호출하면 갱신·삽입 작업을 하는 인스턴스의 대상 테이블 또는 대상 테이블과 동일한 스키마를 가진 분할된 두 개의 테이블이 생성됩니다. 그 중 한 테이블에는 "위배(violation)" 정보가 들어 있고, 다른 테이블에는 "진단(diagnostics)" 정보가 들어 있습니다. 두 테이블의 이름은 기본적으로 삽입 또는 갱신 작업을 하는 대상 테이블 이름 뒤에 각각 "_vio"와 "_dia"가 붙지만, 사용자가 원하는 대로 지정할 수 있고 테이블에 포함될 전체 행 수를 지정할 수도 있습니다.

위배 테이블의 스키마는 추가로 두세 개의 칼럼이 더 있는 것을 제외하면 대상 테이블과 동일합니다. 이 칼럼에는 숫자 카운터와 오류를 발생시킨 작업을 표시하는 약어, 해당 작업을 요청한 사용자의 ID가 기록됩니다.

진단 테이블에는 위배 테이블의 행에 대한 숫자 상호 참조가 있는 숫자 칼럼이 있습니다. 그러나, 이 칼럼과 위배 테이블에 있는 칼럼이 기본 키와 외부 키 관계로 연결되어 있지는 않습니다. 진단 테이블

에도 색인 및 제한 조건 오류의 발생 여부, 위배된 색인 및 제한 조건 이름, 해당 색인 및 제한 조건의 소유자를 나타내는 칼럼이 있습니다.

위배 테이블과 진단 테이블을 생성한 다음에는 테이블에서 색인 및 제한 조건의 "객체 모드"와 "오류 옵션"을 활성화해야 합니다. "객체 모드"란 단순히 객체의 운영 상태를 뜻하며 여기에서 "객체"는 색인과 제한 조건을 가리킵니다.객체 모드는 테이블의 모든 색인이나 제한 조건에 대해 전역적으로 설정하거나 각 제한 조건에 개별적으로 설정할 수 있습니다.
다음의 세 가지 객체 모드를 참조하십시오.

- Enabled(설정 모드)

  기본 객체 모드로서, 색인 및 제한 조건이 있으면 데이터베이스 서버에서 자료 처리 절차에 이를 적용합니다. 요청한 UPDATE, DELETE, INSERT작업이 실패하면 처리 절차가 중단됩니다.

- Disabled(해제 모드)

  데이터베이스 서버에서 색인이나 제한 조건을 처리하지 않습니다.

- Filtering(필터링 모드)

  색인 및 제한 조건이 설정 모드에 있는 것처럼 작동하지만, 요청한 작업에 해당되는 행만 제대로 처리되고, 처리되지 않은 행들은 이 대상 테이블에 연결된 위배 테이블에 복사되며, 관련 정보는 진단 테이블에 기록됩니다.

  제한 조건이나 색인이 필터링 모드에 있고 작업 요청이 실패했을 경우 인스턴스가 반응할 수 있는 작업을 지정할 수 있는데, 이를 "오류 옵션"이라고 합니다. 인스턴스에서 응용프로그램에 오류 코드를 되돌리거나("with error") 되돌리지 않도록 설정할 수 있으며("without error"), 응용프로그램 로직과 후속 프로시저에서 현재 상황에 가장 적합한 옵션을 지정합니다.

  위배 테이블과 진단 테이블을 삭제하려면 먼저 객체 모드를 필터링 모드에서 활성 모드나 비활성 모드로 전환하고 "stop violations" 문을 실행한 다음, 다른 일반 테이블처럼 삭제합니다.

"set" 명령을 사용하여 중복 색인과 트리거를 설정하거나 해제할 수 있지만, 필터링 모드에서는 사용할 수 없으며 이와 관련된 작업은 위배 및 진단 테이블에 로깅되지 않습니다.

롤(Role)

데이터베이스 환경에서의 "롤"은 UNIX 환경의 "그룹" 개념과 비슷합니다.

롤은 테이블, 칼럼, 내장 프로시저의 보안 및 접근을 관리하는 데 사용됩니다.

Informix Dynamic Server 5.x 이전 버전에서는 사용자 ID 기준으로만 사용자 관리가 가능했습니다. 이런 환경에서는 사용 권한에 따라 데이터베이스 사용을 제한해야 할 경우 관리 상의 문제가 발생할 수 있습니다. 이러한 이유로 일반 로그인 ID를 사용하여 사용자가 특정 작업을 수행할 수 있는 사이트가 많았는데, 이럴 경우 데이터베이스 작업을 수행하는 사용자를 쉽게 식별할 수 없으므로 별도의 응용프로그램을 작성하지 않는 한 감사가 거의 불가능했습니다.

Informix Dynamic Server에서는 여전히 사용자 ID별로 권한을 관리하지만, 사용자가 "롤"이라는 더 큰 관리 엔티티에 포함됩니다. 사용 권한을 사용자 ID가 아닌 롤에 따라 부여하고, 각기 다른 사용자 ID를 사용하므로 보다 효율적인 감사를 할 수 있습니다.

데이터베이스 사용 허가

"set session authorization" 명령은 UNIX의 "su" 명령과 같습니다.

데이터베이스에 대해 DBA 권한이 있는 사용자는 이 명령을 사용하여 다른 사용자에게 권한이 있는 기능을 사용할 수 있습니다.

이 기능을 통해 DBA 권한이 있는 사용자는 본래 사용 권한이 없는 테이블이나 칼럼에 대한 작업을 수행할 수 있으며, 다른 사용자에게 테이블이나 칼럼에 대한 접근 권한을 부여할 수도 있습니다.

데이터베이스 이름 변경

이전 버전에서는 DBA가 데이터베이스의 이름을 변경하려면 데이터베이스를 삭제한 다음 새 이름으로 다시 생성해야 했습니다.

따라서 서버가 한 대 밖에 없고 여러 인스턴스를 실행할 리소스가 충분하지 않을 경우, 개발 및 테스

트 환경에서 생산 환경으로의 변환이 매우 복잡했으나, 새 버전에서는 칼럼, 테이블, 색인, 제한 조건 등의 이름을 쉽게 변경할 수 있게 되었습니다.

데이터베이스 생성자가 DBA 권한이 있고 데이터베이스가 현재 사용 중이 아니면 "rename database" 명령으로 데이터베이스 이름을 쉽게 바꿀 수 있습니다.

팁

데이터베이스를 생성한 사용자가 아니더라도 "set session authorization" 명령을 사용하여 데이터베이스의 이름을 바꿀 수 있습니다. 다음은 위 명령을 사용하여 데이터베이스 이름을 바꾸는 절차입니다.

1. I-SQL이나 dbaccess를 통해 이름을 바꿀 데이터베이스에 연결합니다.
2. "set session authorization" 명령을 사용하여 데이터베이스 생성자의 ID로 재설정합니다. 단, "데이터베이스 사용 허가" 부분에서 설명했듯이 해당 데이터베이스에 대해 DBA 권한이 있어야 합니다.
3. 데이터베이스 연결을 끊고 다른 데이터베이스에 연결하거나, dbaccess나 isql 링 메뉴에서 Database 메뉴의 select 옵션을 선택한 다음 인터럽트 키로 중단합니다.이렇게 하면 현재 데이터베이스와의 연결은 끊기지만 질의어 도구를 사용할 수는 있습니다.
4. "rename database" 명령을 실행합니다.

맺음말

지금까지 데이터베이스 관리에 대해 전반적으로 살펴보았습니다.

여기에서 다루지 않은 관리자 기능도 있지만, 이 글을 읽고 나면 데이터베이스와 테이블, 색인을 생성하고, 사용자 환경에 적합하도록 색인이나 테이블의 분할 여부를 지정할 수 있을 것입니다. 또한 테이블과 색인을 라운드 로빈 방식과 표현식 기반 방식으로 분할하고, 가장 효율적인 방법으로 데이터베이스에 테이블을 구축하는 방법도 판단할 수 있으며, 마지막으로 격리 수준을 설정하여 자료 무결성을 유지하면서 여러 사용자가 효율적으로 데이터베이스를 접근할 수 있다는 것을 이해할 수 있게 됩니다.

데이터베이스를 관리할 때는 다음 사항에 유의해야 합니다.

- DBA가 데이터베이스를 관리할 때 dbaccess 링 메뉴를 사용하는 것이 훨씬 편리하지만, SQL 문을 사용하면 제어 범위가 넓고 작업 속도가 빠릅니다.
- 라운드 로빈 분산 기법을 사용하면 RAID 스트라이핑의 단점이 있으나 단일 DB영역에서 테이블을 사용하는 것보다 성능이 향상됩니다. 라운드 로빈 방식으로 분할한 테이블에서 색인을 생성할 때는 "테이블 안"에 만들지 않고 반드시 다른 DB영역으로 분할해야 합니다.
- 먼저 가장 제한적인 조건의 분할식을 작성합니다. 표현식은 자료를 추가하거나 갱신할 때마다 다시 계산되므로 되도록 간단하게 만들어야 합니다.
- 분할식에서는 되도록 "remainder" 속성을 사용하지 않는 것이 좋습니다. 이 속성은 해당 분할 영역에 있는 자료가 가변적이라는 것을 의미하므로 질의를 실행할 때마다 이 분할 영역을 계속 검색하게 됩니다.
- 테이블 attach에 대한 내용은 "테이블 Attach" 부분을 참조하십시오.

웹 클라이언트/서버 응용프로그램

소개

클라이언트/서버 응용프로그램의 플랫폼으로 월드 와이드 웹을 선택해야 하는 이유가 몇 가지 있습니다. 실제로 모든 사용자들이 웹 브라우저를 사용하고 있으며, 이런 웹 브라우저가 (일종의) 응용프로그램 플랫폼의 역할을 하므로 응용프로그램의 실행, 배포, 변경 작업이 쉽기 때문입니다.

웹의 원래 구조는 대부분의 클라이언트/서버 방식의 응용프로그램에는 적합하지 않았습니다. 그러나 새로운 웹 기능이 개발됨에 따라 웹 구조와 웹 브라우저를 클라이언트/서버 응용프로그램 플랫폼으로 사용할 수 있는 가능성이 높아졌습니다.

여기서는 그러한 주요 기능들을 몇 가지 살펴보겠습니다.

웹 구조

웹은, 편리한 하이퍼텍스트 인터페이스를 통해 보편적인 문서 형태를 제공하기 위해 만들어진 기술로서, HTTP(HyperText Transport Protocol)와 HTML(HyperText Markup Language)을 기본 프로토콜로 사용

HTTP(HyperText Transport Protocol)와 HTML(HyperText Markup Language)을 기본 프로토콜로 사용하고 있습니다. HTTP는 웹 브라우저와 웹 서버 사이의 통신 계층 역할을 하며, HTML은 브라우저에서 웹 서버로부터 받은 정보나 지역 파일의 내용을 지정된 형식으로 보여 주도록 정의하는 데 사용됩니다.

HTTP는 사용자(정확히 말하면 사용자의 웹 브라우저)로부터의 각각의 요구를 하나의 독립된 트랜잭션으로 처리하는, 상태 전환이 없는 프로토콜(stateless protocol)입니다. 따라서, 각 요구에는 해당 요구를 수행하는 데 필요한 모든 정보가 포함되어 있어야 합니다. 이러한 HTTP 프로토콜의 특징은 웹의 탄생 이후에 구현되어 온 모든 기능에 영향을 주고 있습니다.

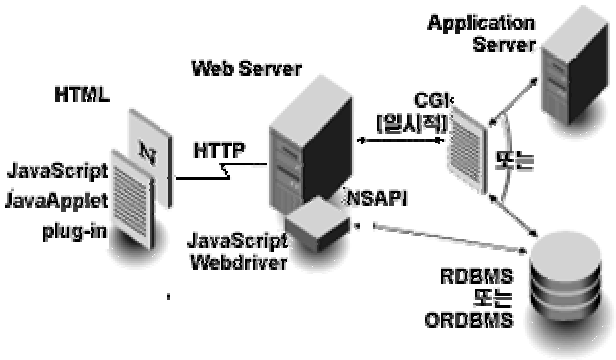


그림 1: 웹 구조

CGI(Common Gateway Interface)는 웹 응용프로그램을 구현하는 주요 기능이 되어 왔으며, 아직까지도 가장 널리 사용되는 응용프로그램 인터페이스입니다. 셸 스크립트, PERL 스크립트, TCL 스크립트, 컴파일된 프로그램 등이 CGI 프로그램이 될 수 있으며, 자원이 많이 필요하지 않은 소형 프로그램에 적합합니다. CGI는 HTTP 프로토콜을 따르며, 질의 중에만 실행되고 그 다음에는 종료됩니다. 따라서 다음 질의를 위해 정보를 보존할 수 없습니다. 이런 점은 특히 데이터베이스에 접근해야 할 경우에 성능에 중대한 영향을 줄 수 있습니다. 간단한 테스트를 해 보면, CGI 프로그램을 사용하는 경우에는 데이터베이스에 지속적으로 연결되어 있는 프로그램에서보다 CPU 이용도가 50 ~ 120배 높은 것을 확인할 수 있습니다.

이러한 성능 문제는, CGI 프로그램에서 데이터베이스로 직접 질의를 보내지 않고 대신 데이터베이스 연결을 유지할 응용프로그램 서버로 질의를 보내 해결할 수 있습니다. 응용프로그램 서버는 요구자의 상태 정보를 보존하고, 해당 콘텍스트(proper context)에 다시 연결할 때 사용할 수 있는 식별자(ID)를 되돌려 줄 수도 있습니다. 응용프로그램 서버는 사용자가 다른 질의를 보내지 않을 경우도 고려해야 하며, 이것은 일반적으로 타임아웃 값으로 지정할 수 있습니다. 그러나 이 값을 너무 작게 지정하면 사용 중에 응용프로그램이 종료될 수 있으며, 너무 크게 지정하면 서버의 자원이 낭비될 것입니다.

응용프로그램 서버를 구현할 때는 다중 사용자 환경을 고려해야 합니다. 응용프로그램을 여러 사용자들이 사용하는 경우에는, 사용자와 응용프로그램 서버 간에 일대일 관계가 성립될 수 없습니다. 이러한 문제는 데이터베이스 공급자들이 항상 고심해 온 것이며 서버 구축자도 이 점을 해결해야 합니다. 데이터베이스 공급자들이 채택한 해결책은 다중 스레드 방식을 사용하는 서버입니다. 그러나, 이렇게 하면 서버에서의 내부 실행 절차들이 지나치게 복잡해질 수 있습니다.

CGI를 실행하는 대신 웹 서버 API를 사용할 수도 있습니다. 이 인터페이스는 '웹 서버의 장치 제어기'라고 보면 됩니다. 서버측 JavaScript 언어는 웹 서버 API를 사용하여 실행 프로그램을 구현한 한 예입니다.

웹 서버 API를 사용하여 작성한 실행 프로그램은 웹 서버의 일부분입니다. 따라서 웹 서버의 성능을 향상시킬 수 있으며 각 요구 간의 정보를 보존할 수 있습니다. 이 인터페이스를 사용하여 응용프로그램 서버와 통신하거나 데이터베이스 서버와 직접 접속할 수도 있습니다.

또한 웹 서버 API를 사용하여 구현된 응용프로그램 코드는 웹 서버의 일부분이기 때문에 서버의 작업 방식을 따르게 됩니다. 웹 서버에서 여러 개의 사본을 시작하여 확장성(Scalability)을 제공할 경우에는 사용자의 환경에 미칠 영향을 고려해야 합니다. 클라이언트가 동일한 물리적 서버에 다시 연결하지 않으면 이전에 사용하던 클라이언트 콘텍스트(client context)는 사용할 수 없습니다. 웹 서버가 다중 스레드 방식으로 동작할 때, 블로킹 작업이 생길 경우 웹 서버를 여러 개의 프로세스로 나눕니다. 이와 같은 상황에서는 하나의 활성 데이터베이스 연결만 가능하며, 질의가 길어지면 다른 사용자들은 서버를 사용할 수 없게 됩니다.

웹 브라우저

웹 브라우저에서는 웹 응용프로그램 개발을 위한 여러 가지 방법이 제공됩니다. 이 중 하나가 플러그인으로, 여러 개의 플러그인을 필요한 만큼 사용할 수 있습니다. 플러그인은 브라우저에서 실행되는 응용프로그램으로서, 별도로 설치해야 하며, 사용자 인터페이스를 제공하는 경우는 해당 플랫폼의 함수를 사용해야 하기 때문에 표준 클라이언트 응용프로그램보다 다소 불편합니다.

Netscape LiveConnect 환경에서 사용자 인터페이스는 JavaScript를 사용한 HTML 또는 Java Script를 통해 처리되는 반면 플러그인은 백그라운드 처리로만 사용할 수 있습니다. 비록 플러그인 프로그램을 별도로 설치해야 하는 문제를 해결할 수는 없으나 갱신 횟수를 줄일 수는 있습니다.

클라이언트측 JavaScript는 이벤트 방식으로 특정 작업을 실행합니다. 즉, 실행할 작업에 해당하는 단추를 누르면 JavaScript가 시작되고 해당 작업이 실행됩니다. JavaScript를 사용하는 주 목적은 클라이언트가 특정 작업의 실행을 제어할 수 있도록 하여 HTML의 기능을 향상시키는 것입니다.

마지막으로, 웹 브라우저에서는 웹 서버에서 다운로드한 Java 애플릿을 실행할 수 있습니다. 이것은 응용프로그램에 유연성을 제공하므로 매우 유용합니다. 또한, 응용프로그램을 수정할 경우 한 곳에서만 갱신하면 됩니다. 이제 남은 문제는, 다운로드한 애플릿을 사용하면서 데이터베이스에 접근하는 것입니다.

Informix Java API

Java 객체 인터페이스(Java Interface for Object - OIJ)라고도 불리는 Informix Java API는 클라이언트측이나 서버측 Java 환경에서 데이터베이스 서버 서비스와 자료를 사용할 수 있도록 인터페이스를 제공합니다. Java API는 기존의 C++ API를 토대로 설계되었으며, JDBC API는 Java API의 일종입니다. Java API를 사용하여 응용프로그램이나 애플릿을 개발하려면 버전 1.1.1 이상의 JDK(Java Development Kit)를 사용해야 합니다.

Java API는 직접 연결 방식(direct connection)과 원격 연결 방식(remote connection)의 두 가지 연결 방식을 제공합니다. 직접 연결 방식에서는 응용프로그램이나 애플릿에서 데이터베이스 서버와 직접 통신할 수 있습니다. 단, Java API 클래스를 찾을 수 있고 공유 라이브러리가 탐색 경로의 일부가 되도록 클라이언트

환경을 설정해야 합니다. 기본적으로 Java API는 $InformixDIR/lib/java에 설치됩니다. 여기서 $InformixDIR은 Informix 제품이 설치된 디렉토리입니다. 이 경우, UNIX Solaris 환경 변수는 다음과 같이 설정할 수 있습니다.

CLASSPATH=$CLASSPATH:$InformixDIR/lib/java/lib/classes

LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$InformixDIR/lib/java/classes/sparc:$InformixDIR/lib/esql:$InformixDIR/lib/dmi

Windows NT 환경에서는 PATH 환경 변수에 라이브러리의 디렉토리 경로가 포함되어야 합니다. 다음은 PATH 환경 변수의 예입니다.

PATH=%PATH%;%InformixJAPI%₩lib₩pc;%InformixDIR%₩lib₩esql;%InformixDIR%₩lib₩dmi

Java RMI(Remote Method Invocation) 서버는 클라이언트 응용프로그램에서 서버를 원격으로 연결할 수 있게 해 줍니다. RMI 서버는 기본 포트 1099를 사용하는 서비스로, Java 응용프로그램이나 애플릿은 이 기능을 이용하기 위해 Java 객체 인터페이스(OIJ)를 사용합니다. 데이터베이스에 접속하기 위해 클라이언트 시스템에 별도의 소프트웨어를 설치할 필요는 없습니다. 즉, 클라이언트측에는 사실상 소프트웨어 유지보수가 없어도 된다는 웹 응용프로그램 구조의 주요 특징이 적용됩니다.

Informix Java 객체 인터페이스

Informix Java 객체 인터페이스(OIJ)는 하나의 간단한 인터페이스를 제공하며, 이를 통해 직접 연결과 원격 연결 간의 차이를 최소화시킬 수 있습니다. 하나의 애플릿이나 응용프로그램 내에 직접 연결과 원격 연결의 두 가지 방식을 모두 사용할 수 있습니다.

Java 응용프로그램이나 애플릿은 먼저, Credentials 인터페이스를 수행하는 객체와 상호 작용을 합니다. 이 객체는 DBMS와 연결하기 위해 필요한 정보를 응용프로그램에서 가져옵니다. 한 번 생성된 이 매개변수의 정보는 변경할 수 없습니다. 객체 추출의 "provider" 모델은 인터페이스 전체에서 사용됩니다. 일반적으로 Java API 객체는 "new" 연산자로 만들지 않고, 다른 Java API 객체로부터 추출하는 방식으로 생성됩니다.

Credentials 객체는 이 트리 구조에서 루트가 됩니다.

직접 연결 인터페이스를 사용하려면 응용프로그램에서 다음과 같이 임포트해야 합니다.

import Informix.api.*;

import Informix.api.direct.*;

RMI 연결을 사용하려면, Java 응용프로그램에 다음이 포함되어야 합니다.

import Informix.api.*;

import Informix.api.remote.rmi.*;

직접 연결 Credential은 두 가지 생성자(Constructor)를 제공합니다.

첫째 생성자는 매개변수가 없으며 서버에 연결하기 위해 필요한 값은 환경 변수에서 가져옵니다.

둘째 생성자는 다음과 같습니다.

public DirectCredentials(String db, String user, String sys,

String passwd) throws DBClientException

위에서 db는 데이터베이스 이름, user는 사용자 이름, sys는 데이터베이스 서버 이름, passwd는 사용자 암호를 각각 나타냅니다. 생성자는 다음과 같이 사용합니다.

Credentials cred;

try {

cred = new DirectCredentials(db, user, sys, passwd);

}...

원격 연결을 사용할 경우에는 위에서 DirectCredentials 대신에 RMICredentials 호출을 사용해야 하며, 호출 형식은 다음과 같습니다.

public RMICredentials(String appServer, String db, String user, String sys,

String passwd) throws DBException, MalformedURLException,

UnknownHostException

RMICredentials 호출은 DirectCredentials 호출에서 사용한 매개변수 외에, RMI 서버의 위치를 나타내는 appServer 매개변수를 추가로 지정해야 합니다. 이 문자열 변수의 형식은 다음과 같습니다.

rmi://hostname<:port>/

hostname은 RMI 서버가 있는 컴퓨터의 이름이고, 포트 번호는 RMI 서버에서 사용하는 서비스 포트입니다. 기본 포트 번호인 1099를 사용할 때는 이 매개변수를 생략해도 됩니다.

일단 시스템 사용 허가를 받으면 직접 연결이나 원격 연결에 상관없이 데이터베이스 객체는 동일합니다. 데이터베이스에 연결하고 접속한 다음 질의를 보내는 이벤트의 일반적인 순서는 다음과 같습니다.

```
Connection conn;
Query query;
conn = cred.Connection();
query = conn.query();
...
conn.open();
```

이렇게 하면 질의가 질의 객체로 전달되고 응용프로그램에서 그 결과를 수집합니다.

Java 애플릿

Java 애플릿을 사용하여 웹에서 클라이언트/서버 응용프로그램을 구현할 수 있습니다.

첫째 단계는 웹 서버에 웹 페이지를 요청하는 것입니다. 웹 서버는, Netscape Server API(NSAPI)나 Information Server API(ISAPI)를 통해 Informix 제품 중 하나를 사용하여, 해당 페이지가 있는 Informix 데이터베이스에 접근할 수 있습니다.

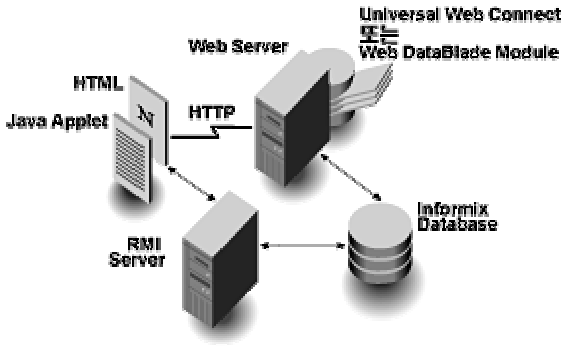


그림 2: 클라이언트/서버 Java 애플릿

추가적인 상호 작용 없이 간단히 어떤 페이지만 요구한다는 점에서 웹 서버를 통한 데이터베이스 접근은 웹의 일반적인 구조와 일치합니다.

웹 서버를 통해 데이터베이스에 접근하면 애플릿을 확인하는 페이지만 되돌려집니다.
다음은 웹 페이지에서 주요 섹션의 예입니다.

<APPLET CODE="myapplet.class"

NAME="myapplet" WIDTH=500 HEIGHT=500 > </APPLET>

이렇게 하면 애플릿에서 브라우저의 화면을 인계 받고 해당 응용프로그램이 시작됩니다. 애플릿은 호스트와 통신이 가능하고, RMI 서버와 연결할 수 있으며, 데이터베이스에 접근할 수 있습니다. 또한, 사용자와의 상호 작용이나 데이터베이스 접근을 제어할 수 있습니다. 따라서 애플릿은 하나의 클라이언트/서버 응용프로그램입니다.

웹 표준 이용

필요가 없으며 응용프로그램 코드는 플랫폼에 종속되지 않아야 한다는 웹 응용프로그램의 조건을 만족합니다. 그러나 몇 가지 단점이 아직 남아 있습니다.

그 중 한 가지는 사용자와의 상호 작용이 모두 Java 애플릿을 통해서 이루어진다는 것입니다. 즉, 애플릿 작성자는 사용자 인터페이스를 구현하기 위해 반드시 Java AWT(Abstract Windowing Toolkit)를 사용해야 합니다. 그런데 이와 같은 윈도용 프로그래밍 환경에 익숙해지려면 상당한 기간이 소요됩니다. 따라서 Java를 전략적인 언어로 반드시 사용해야 하는 경우가 아니라면 다른 간단한 방법을 찾는 것이 좋습니다. 이를 위해 사용자 인터페이스를 간단히 작성할 수 있는 몇 가지 제품들이 나와 있습니다.

전체 응용프로그램에 대형 Java 애플릿이 필요할 때는 다운로드 시간이 문제가 됩니다. 따라서 Java 애플릿은 인트라넷 환경에서 사용하는 것이 더 적합합니다.

결국 애플릿은 사용자와의 상호 작용이 비교적 적은 소형 응용프로그램에서 많이 사용됩니다. 이런 상황에서는 Web DataBlade 모듈을 사용하여 문제를 해결하는 것이 좋습니다.

표준화된 기능들을 이용하면 선택의 범위를 넓힐 수 있습니다. 다음은 몇 가지 표준화된 기능들입니다.

- JavaScript -- JavaScript 언어는 Netscape에서 처음으로 도입하여 지금은 여러 브라우저에서 사용되고 있습니다. 클라이언트측 JavaScript는 브라우저에서 실행되는 언어입니다. JavaScript의 이벤트 처리 기능을 통해, 사용자가 원하는 기능을 실행하기 위해 해당 단추를 누르면 사용자의 시스템에 있는 응용프로그램에서 브라우저를 제어할 수 있습니다. 클라이언트측 JavaScript는 Java 애플릿에서 메서드(method)나 함수를 호출할 수도 있습니다.

- 프레임(Frames) -- 페이지를 로드할 때 이전 페이지의 전체 내용은 손실되지만, 화면을 여러 개의 프레임으로 나누면 각 프레임을 개별적으로 로드할 수 있습니다. 예를 들어, 브라우저의 목차 부분은 그대로 두고 내용만 새로 로드할 수 있습니다.

이 두 가지 기능을 통해 웹 브라우저의 기능을 편리하게 이용할 수 있습니다. JavaScript에서는 다른 프레임에 있는 객체를 참조할 수 있으며, 이 기능은, 새로운 구현 방법의 기초를 제공합니다.

기본 개념은 브라우저를 두 개의 프레임으로 나누는 것입니다. 주 프레임은 응용프로그램에 사용되고 보조 프레임에는 Java 애플릿이 포함됩니다. 애플릿은 사용자와 상호 작용을 하지 않으므로 보조 프레임의 크기는 최소로 해야 하며, 상태 메시지를 나타내는 데 사용할 수도 있습니다.

이렇게 하면 두 개의 프레임을 포함하는 HTML 페이지를 로딩할 수 있습니다.

각 프레임은 다음과 같이 각각 다른 HTML 페이지를 참조합니다.

```
<HTML>
<HEAD>
<TITLE>SQL 데모</TITLE>
</HEAD>

<BODY>
<frameset ROWS ="*, 20" >
<frame border = 0 name="body" src="sqlframe1.html"
scrolling="auto" >
<frame border = 0 name="message" src="sqlframe2.html"
scrolling="no" >
</frameset>

<NOFRAME>
Frame needed!
</NOFRAME>

</BODY>
```

</HTML>

이 HTML 페이지는 한 프레임 집합에 두 개의 프레임이 정의되어 있습니다.

각 프레임에는 프레임 이름과 프레임에 표시할 내용이 있는 파일이 지정됩니다.

message라고 하는 보조 프레임은 매우 간단하며 프레임에 로드될 애플릿을 식별합니다.

다음은 보조 프레임의 예입니다.

<HTML>

<HEAD>

<TITLE>DB 인터페이스< /TITLE>

</HEAD>

<BODY>

<APPLET CODE="sqlapplet.class" NAME="dbi" WIDTH=10 HEIGHT=10>

</APPLET>

</BODY>

</HTML>

앞에서 설명했듯이 응용프로그램 설계자는 이 페이지에 메시지 영역을 추가할 수 있습니다. 위의 코드에, 페이지를 언로드하는 JavaScript 함수(OnUnLoad)를 추가하면 데이터베이스 연결을 순차적으로 종료할 수 있습니다.

이제 남은 작업은 지역 처리와 데이터베이스 서버와의 통신을 위해 JavaScript를 사용하는 HTML 페이지를

설계하는 것입니다. 처리 과정을 자세히 이해하려면 브라우저에서 사용되는 객체의 계층 구조를 알아야 합니다.

이 항목에 대한 자세한 내용은 다음 인터넷 사이트를 참조하십시오.

http://www.netscape.com/eng/mozilla/2.0/handbook/javascript/index.html

웹 브라우저는 창으로 시작하는 객체의 계층 구조를 사용합니다. 창 페이지에는 위치, 사용 기록, 문서 등의 여러 하위 객체가 항상 포함됩니다. 이 객체는 각각의 속성을 가지고 있습니다. 예를 들면 문서는 제목이라는 속성을 가지고 있습니다. 문서의 내용은 페이지의 구조에 따라 달라집니다. 창의 일반적인 계층 구조는 다음과 같습니다.

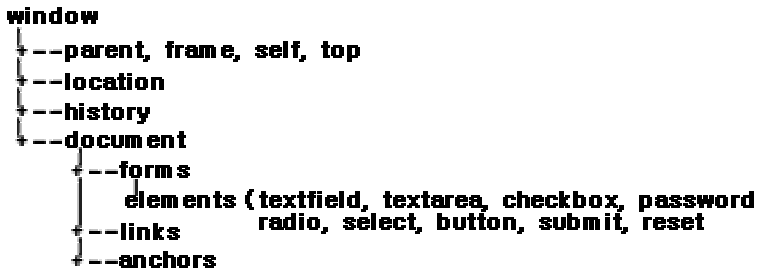


그림 3: 일반적인 객체의 계층 구조

계층 구조에는 여러 개의 창이 포함될 수 있습니다. 따라서 다른 창을 추가할 수 있으며 여러 창 간에 통신도 가능합니다. 계층 구조 내에서 객체의 상위 창을 "parent"라고 합니다.

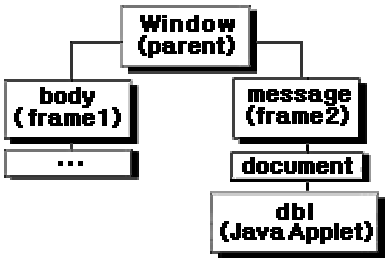


그림 4: 객체의 계층 구조

그림 4는 예제 응용프로그램에 대한 계층 구조를 묘사한 것입니다.

일반적인 객체의 계층 구조에서 볼 수 있듯이 창 아래에 바로 프레임이 있습니다. 모든 객체는 이름을 사용하여 참조할 수 있습니다. 예제 HTML 페이지에서 보면 "body"와 "message"라는 두 개의 프레임이 정의되어 있습니다. 계층 구조에 대한 자세한 정의를 찾아 보려면 각 프레임을 구성하고 있는 페이지를 보면 됩니다. "message" 프레임의 내용은 앞에서 "dbi"라는 애플릿으로 정의되었습니다.

"body" 프레임은 parent.message.document.dbi를 참조하여 Java 애플릿을 액세스할 수 있습니다. "document" 레벨은 "message" 프레임의 HTML 페이지에 있습니다. 애플릿 메서드는 애플릿 객체의 속성으로 실행할 수 있습니다. 애플릿에 getstatus()라는 메서드가 있으면, parent.message.document.dbi.getstatus()로 메서드를 호출할 수 있습니다.

"body" 페이지의 내용은, 주로 지역 JavaScripts 함수 호출을 포함하는 HTML입니다. body 프레임의 첫째 내용이 로그인 HTML 페이지라면, 이 페이지에는 해당 입력 필드로 구성된 화면 폼이 포함됩니다. 다음 정의를 사용하여, 확인 단추를 눌렀을 때 해당 작업을 시작하게 할 수 있습니다.

단추에 ONCLICK 이벤트가 발생하면 전체 폼을 인수로 하여 upd JavaScript 함수가 실행되어 로그인하기 위해 필요한 매개변수를 추출합니다. 이 함수는 데이터베이스 연결의 성공 여부에 따라 "body" 프레임의 내용을 변경합니다. 다음은 upd 함수의 예입니다.

```
<SCRIPT LANGUAGE="JavaScript">
function upd(form) {
parent.message.document.dbi.doConnect(form.dbname.value,
form.username.value, form.servname.value,
form.password.value);
if (0 == parent.message.document.dbi.getStatus() )
form.errninfo.value =
"데이터베이스 서버에 연결할 수 없습니다.";
else
parent.body.location.href="sqlframe1b.html";
}
</SCRIPT>
```

이 함수는, dbname, username, servname, password, errninfo 같은 필드가 포함되어 있는 폼이 하나의 인수로 전달된 것으로 간주합니다. 이 함수는 먼저 서버에 연결하기 위해 Java 애플릿을 호출하며, 로그인의 성공 여부에 따라 다음과 같이 이벤트가 발생합니다.

- 실패: errninfo 필드에 오류 메시지가 표시됩니다.
- 성공: "body" 프레임 소스 참조가 다른 페이지로 변경됩니다. 이렇게 되면 프레임의 내용이 다른 페이지로 변경되고, 새 페이지는 데이터베이스 서버와 통신하기 위해 애플릿과 상호 작용을 합니다.

신중하게 설계한 Java 애플릿은, JavaScript 언어에 대한 데이터베이스 API로 사용할 수 있습니다. 이렇게 하면 모든 사용자 상호 작용이 웹 환경 내에서 이루어지고 최신 HTML 표준 기능을 이용할 수 있습니다.

맺음말

웹 응용프로그램을 개발하는 방법에는 여러 가지가 있습니다. 상황에 따라 Universal Web Architecture를

사용하여 응용프로그램을 개발하는 것이 적합할 때도 있고, 좀 더 복잡한 방법을 사용해야 할 경우도 있습니다. 처음에는 가장 유연성있는 방법을 사용하는 것이 좋습니다. 여기서는 현재의 웹 표준을 이용하는 여러 가지 방법에 대해 설명했습니다. HTML, JavaScript, Java 간의 상호 작용을 통해 웹의 기능을 활용하면서 클라이언트/서버 응용프로그램을 보다 잘 구현할 수 있습니다.

웹은 계속 발전하고 있으며, 앞으로도 여러 가지 방법이 더 추가될 것입니다. 따라서 새로운 표준을 이끌어 내는 웹의 발전을 주시할 필요가 있습니다. 현재의 표준 내에서 이 새로운 기능들을 어떻게 조화시킬 것인가를 탐구하면, 보다 쉽고 나은 방법을 설계하고 구현할 수 있을 것입니다.

참조

다음은 도움이 되는 정보를 얻을 수 있는 참조 자료입니다.

- Informix Java API Programmer's Guide, Version 1.04, published by Informix Press/Prentice Hall PTR,
- The JavaScript Authoring Guide, provided by Netscape Communications Corporation at the following Web location: http://www.netscape.com/eng/mozilla/2.0/handbook/javascript/index.html

고성능 웹 응용프로그램 구축

소개

Informix Web DataBlade Module의 구조와 조정에 대해 자세히 살펴보겠습니다.

성능조정방법

성능 조정 작업을 할 때 제일 먼저 알아 두어야 할 것은, 일반적인 조정 방법이 정해져 있는 것이 아니라 특정 상황에 따라 방법을 선택해야 한다는 것입니다. 특히 조정 후 성능이 얼마나 향상되었는지 확인하려면 현재 상태를 파악하고 있어야 합니다.

따라서, 먼저 하나의 문제를 기준으로 하여 조정 작업을 시작합니다. 예를 들어, 특정 페이지에서 타임아웃이 너무 많다는 사용자들의 불만을 해결하기 위해 조정 작업을 할 경우, 먼저 이런 현상이 얼마나 자주 발생하는지 확인합니다. 가능하면 이런 상황을 직접 확인하고, 조정 작업에 따른 결과를 바로 추적할 수 있도록 이 문제만 따로 떼어 살펴봅니다. 그러나 특정 기간에만 발생하거나 여러 개의 작업이 동시에 진행되는 경우 즉, 복합적인 환경이 조성되어야 발생하는 문제들이 있기 때문에 항상 이런 식으로 문제를 따로 뽑아 살펴볼 수는 없습니다.

일단 문제가 확인되고 그 문제만을 따로 뽑아 냈으면 목표를 정해야 합니다. 목표는 사용자가 해당 페이지에 접근할 때 10초 이상 기다리지 않도록 하는 것입니다.

각각의 조건을 하나씩 변경할 때마다 계속 성능을 평가해 보면서, 성능을 향상시킨 조건과 그렇지 못한 조건을 정확히 찾아 냅니다.

각 테스트를 진행할 때는 테스트 과정에서 발생하는 작업들을 분석합니다. 목표에 따라 처리량 (초당 전체 시스템의 트랜잭션 처리량), 응답 시간(페이지가 화면에 표시되는 시간), 또는 특정 자원(CPU, 메모리, 디스크, 네트워크)의 이용도 등 관심 분야가 다를 수 있습니다.

특히 Informix Universal Data Option과 Informix Web DataBlade Module을 사용하는 웹 환경에서 성능 조정이 복잡해 질 수 있는 것은 구성요소가 많다는 것입니다.

데이터베이스를 포함하는 웹 서버 환경에는 다음과 같은 네 가지의 주요 구성요소가 있습니다.

- 웹 브라우저 - Netscape Navigator나 Internet Explorer 등이 있으며, Java, JavaScript, Active-X를 지원하는 고급 브라우저도 고려해 볼 수 있습니다.
- 웹 서버 - Netscape Enterprise Server, Microsoft Internet Information Server, Apache Server 등이 있습니다.
- 데이터베이스 서버 - 여기서는 IDS/UDO와 Illustra Server를 중심으로 설명합니다. 이 제품들과 Informix Web DataBlade Module을 사용하여, HTML이나 이미지 또는 웹 페이지를 작성하고 전송하는 데 필요한 기타 자료 등을 데이터베이스에 저장할 수 있습니다.
- 네트워크 환경 - 인트라넷(회사 내에서 웹 페이지 전송), 익스트라넷(협력 업체에 웹 페이지를 전송, 인터넷 사용 가능), 인터넷 등이 있습니다

성능을 조정할 때는 최소한의 노력으로 최상의 결과를 얻을 수 있도록 하는 것이 중요합니다. 표 1은 Illustra Server와 IDS/UDO에서 웹 응용프로그램을 사용하여 성능을 조정해 본 결과입니다.

	영역	조정에 따른	작업	응용프로그램에

	▪ 구성	20%	높음	없음
	▪ 페이지 내장 SQL	15%	중간	낮음
	▪ 페이지 구축	10%	중간	낮음

표 1: 성능 조정 영역

문제를 해결하는 데는 여러 방법을 사용할 수 있습니다. 네트워크 성능에 관련된 문제라면 네트워크의 대역폭을 넓히거나 전송되는 정보의 양을 줄여 해결할 수 있을 것입니다. 또한 구성요소들의 버전을 최신 버전으로 유지하는 것도 중요합니다.

위 표를 볼 때, 응용프로그램에 미치는 영향(웹 페이지를 얼마나 많이 수정해야 하는가) 즉, 작업 효율(성능을 향상시키기 위해 얼마나 많은 작업이 필요한가)에 따라 각 항목을 구별해 보는 것도 하나의 좋은 방법입니다. 이런 관점으로, 요구되는 작업이 적은 조정부터 시작해 보면 표 2와 같이 순서를 매길 수 있습니다.

	영역	조정에 따른 성능 향상률 %	작업 효율	응용프로그램에 미치는 영향
	Web DataBlade			
	▪ 구성	20%	높음	없음
	데이터베이스 서버			
	▪ 구성	10%	높음	없음
	운영 체제	5%	높음	없음
	웹 서버	5%	높음	없음

	데이터베이스 서버			
	▪ 색인	25%	중간	없음
	▪ 디스크 이용도	10%	중간	없음
	네트워크	5%	중간	없음
	Web DataBlade			
	▪ 페이지 내장 SQL	15%	중간	낮음
	▪ 페이지 구축	10%	중간	낮음
	데이터베이스 서버			
	▪ 스키마	25%	낮음	높음

표 2: 응용프로그램에 미치는 영향을 기준으로 나열한 성능 조정 영역

이 분석 결과에 따르면 웹 구성부터 시작하는 것이 가장 좋으며, 그 다음으로 데이터베이스 서버 구성과 운영 체제 환경을 고려해 볼 수 있습니다. 데이터베이스 스키마 변경은 제일 아래에 나와 있지만, 이것이 이 영역을 변경할 필요가 없다는 것을 의미하는 것은 아니며 단지 제일 먼저 시작할 작업 영역이 아니라는 것입니다. 이 테스트는 데이터베이스 설계를 바탕으로 한 것이기 때문에, 결론이 다소 한쪽으로 치우칠 수도 있다는 것을 염두에 두어야 합니다.

성능을 모니터하려면 도구들이 필요합니다. 표 3에는 주의해야 할 증상들과 이에 대한 정보를 수집할 수 있는 도구들이 나와 있습니다.

		증상	OS 유틸	Informix 모니터링 유틸리티

			리티	
	CPU	하나 이상의 CPU에서 CPU 이용도가 100%에 근접함. 로드 불균형	vmstat, sar, ps	onstat -g act, ath, glo, ntd, ntt, ntu, qst, rea, sch, ses, sle, spi, sql, sts, tpf, wai, wst
	메모리	페이지 아웃, 페이지 스캐닝, 스와핑	vmstat, sar, top, ps, ipcs	onstat -g ffr, dic, iob, mem, mgm, nsc, nsd, nss, seg, ufr
	입출력 작업	하나 이상의 디스크에서 입출력 장치의 이용도가 100%에 근접함. 로드 불균형	iostat	onstat -g iof, iog, ioq, iov
	네트워크	충돌, 재전송, 타임아웃, 전송되는 정보의 양이 너무 많음	ping, netstat, -a, -n, -s	onstat -g nsd, ntd, ntt, ntu

표 3: 증상에 따른 성능 모니터링

조정 작업에서 중요한 것은 문제의 원인을 개선시키는 것입니다. 예를 들어, CPU 이용도가 98% 정도로 너무 높다면 CPU를 추가하여 문제를 해결할 수 있을 것입니다. 그러나 CPU 이용도가 20% 정도로 낮으면 CPU를 추가해도 도움이 되지 않습니다.

메모리 이용도는 특히 중요합니다. 시스템에 메모리가 충분히 있는지 그리고 데이터베이스 서버가 이 메모리를 사용할 수 있도록 제대로 구성되어 있는지 확인하십시오. 조정 작업을 할 때는 페이지 스와핑을 모니터하십시오.

"페이지 아웃(page out)" 값은 작업 메모리의 페이지가 메모리에서 디스크로 이동하는 빈도를 나타냅니다. 데이터베이스 관리 시스템(DBMS)에서 이 값은 큰 변수가 되므로, DBMS에서 자체적으로 관리 기능을 수행하도록 하는 것이 좋습니다. "페이지 아웃"이 너무 자주 발생하는 경우에는, IDS/UDO나 Illustra Server가 사용하는 메모리의 양을 줄이십시오.

Windows NT 상에서 리소스 모니터링

Windows NT에는 다음과 같은 여러 가지의 유틸리티와 도구들이 있습니다. 시스템 성능 모니터는 모든 시스템 자원을 모니터하고, IIS와 FTP 통계를 제공하는 유틸리티입니다. 작업 관리자는 프로세스와 이에 필요한 CPU 및 메모리의 양을 모니터합니다. 또한 Windows NT 리소스 키트에 포함되어 있는 pstat, pmon 등의 명령행 도구들이 있으며 이 도구들은 UNIX 관리자들에게는 이미 익숙할 것입니다.

이러한 다양한 도구들을 사용하여 성능을 정확하게 모니터할 수 있습니다.

데이터베이스 관리 기본 지침

Web DataBlade를 사용하여 웹 페이지를 개발하면, DBMS에서 이중 임무를 수행하게 됩니다. 하나는 응용프로그램의 모든 운용 자료를 보관하는 것이고, 둘째는 HTML 형식의 웹 페이지의 소스 코드, 응용프로그램에서 사용하는 기타 객체(이미지, 오디오, 비디오 등)들을 보관하는 것입니다. 따라서 사용자는 DBMS를 조정할 수 있어야 합니다.

다음은 DBMS를 조정하는 몇 가지 지침입니다.

- 메모리 - 운영 체제에서 페이징을 하지 않으면 메모리를 추가로 할당합니다.
- 분리 - 사용할 수 있는 디바이스들에 자료를 분산시킵니다.
- 분할 - 하나의 테이블을 여러 개의 디바이스에 분산시킬 수 있는 IDS/UDO의 기능을 이용합니다.
- 색인 - 색인이 올바른지, 또한 색인이 제대로 사용되는지 확인합니다.
- 색인 유지 관리 - INSERT, UPDATE, DELETE 연산을 계속 사용하게 되면 시간이 지날수록 색인의 율성이 떨어지게 되며, 이렇게 되면 색인을 다시 작성해야 합니다. Illustra Server에서는 특히 그렇습니다.

- 색인 통계 - 색인 통계는 주어진 질의를 어떻게 최적화할 것인가의 정보를 서버에 제공합니다. 이 통계를 갱신하면 생성되는 질의 플랜을 향상시킬 수 있습니다. (각주 1 참조).

전형적인 웹 환경에서는 읽는 작업이 쓰는 작업보다 많습니다. 이것은, 쓰는 작업은 없고 많은 양의 간결한 요구들을 신속히 처리하는 OLTP와 유사합니다. 따라서 메모리 조정 작업은 OLTP 환경에서의 조정 작업과 비슷한 경향이 있습니다.

참고 지침

다음은 데이터베이스를 설계할 때 도움이 되는 몇 가지 참고 사항입니다.

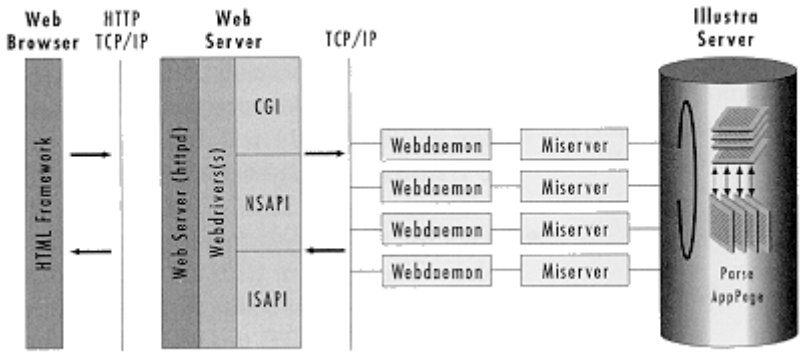
- 테이블 키의 이름을 짧게 지정할 것 - 이렇게 하면 참조 테이블에 기록할 외부 키의 글자 길이를 줄일 수 있으며, 색인의 길이도 줄일 수 있습니다.
- 적합한 자료형을 사용할 것 - 대리 키를 생성하려면 SERIAL 자료형을 사용하십시오. 해당 작업에 적합한 크기의 정수를 사용하십시오. 사용하는 정수의 범위가 너무 좁으면 예상되는 모든 값을 표시할 수 없으며, 너무 넓으면 자료를 저장하는 공간이 낭비됩니다.
- NULL은 적합한 경우에만 사용할 것 - NULL은 여러 가지로 해석될 수 있습니다. 작성한 테이블에서 NULL을 사용하는 이유와 용도를 정확하게 파악해야 합니다.
- 색인 추가 - 까다로운 질의에 사용할 수 있는 적합한 색인이 있는지 확인하십시오. 색인을 추가하면 INSERT, UPDATE, DELETE 연산이 느려지지만 웹 환경에서 이것은 별 문제가 되지 않으며] 오히려 SELECT 연산을 더 빠르게 수행하는 것이 중요합니다.
- 색인을 너무 많이 사용하지 말 것(한 테이블에 3 ~ 5개를 넘지 말 것) - 색인이 너무 많으면 색인이 모두 사용되지 않는 문제가 생깁니다. 또한, INSERT, UPDATE, DELETE 연산을 사용할 때마다 불필요한 작업이 수행됩니다. IDS/UDO에서는 참조 무결성을 선언하기 위해 색인을 추가합니다.
- 업무 시간 중에 유지 관리를 수행하지 말 것 - 유지 관리 작업은 업무 외 시간에 수행할 수 있습니다. 업무가 한참 진행 중인 오후 2시에 Illustra Server에서 VACUUM ... WITH(STATISTICS) 연산을 수행한다거나, IDS/UDO에서 UPDATE STATISTICS 연산을 수행하지 마십시오.
- 참조 제한 조건이나 기타 제한 조건들을 선택적으로 완화할 것 - 일반적으로 이러한 제한 조건들은 시간적 일관성을 보증하므로 제한 조건을 완화시키는 것이 권장 사항은 아닙니다. 또한 성능에 미

치는 영향이 너무 크고, 응용프로그램에서 이 제한 조건들을 위반하는 것이 허용되지 않을 때도 있습니다. 한 응용프로그램에서, 업무가 진행되는 낮에는 제한 조건을 다소 완화시키고 업무가 없는 밤 시간에 잘못된 내용을 바로 잡을 수 있습니다.

- 선택적으로 비정규화시킬 것 - 때로는 테이블에 자료를 중복해서 기록해야 할 필요가 있습니다. 이렇게 하면 실행해야 할 조인의 수를 줄여 성능을 향상시킬 수 있습니다. 비정규화시키는 방법에는 여러 가지가 있으며, 필요한 값을 미리 계산하여 테이블에 복사해 두거나, 테이블을 수평이나 수직으로 분할할 수도 있습니다.

Web DataBlade 구조

그림 1은 Illustra Server용 Informix Web DataBlade Module 버전 2.2의 구조입니다. Web DataBlade 버전 2.2에서는 웹 드라이버가 웹 서버와 통합되어 있으나, 각각의 동시 접근에 대해서는 독립된 Webdaemon과 Miserver 프로세스가 있습니다. Webdaemon은 고정된 수의 연결을 유지하며 간단한 라운드 로빈 방식으로 선택됩니다.



스가 유연성 있는 TCP/IP 스레드들로 대체되었고, 웹 드라이버와 데이터베이스 간의 통신이 가능하도록 개선되었습니다. 스레드 수는 상황에 따라 늘리거나 줄여 설정할 수 있습니다

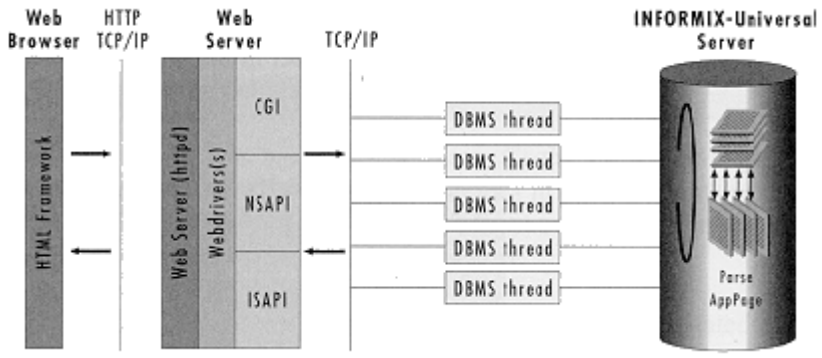


그림 2: Informix Web DataBlade 버전 3.3의 향상된 기능

Web DataBlade 조정

NSAPI나 ISAPI 사용

NSAPI나 ISAPI를 사용하면 CGI 방식의 인터페이스를 사용할 때보다 페이지의 전송 성능이 UNIX 시스템에서는 25% 정도, Windows NT에서는 40% 이상 향상됩니다. IDS/UDO에서는 개방형 데이터베이스 연결을 유지했던 Webdaemon 프로세스를 사용하지 않기 때문에, Illustra Server와 비교해 볼 때 CGI를 사용하는 것이 다소 효율이 떨어집니다. 따라서 이제는 웹 드라이버 CGI가 프로세스를 시작하여, 연결을 개시하고, 질의를 보내며, 각 접속에 대해 연결을 종료해야 합니다. 웹 서버 API를 사용하면 개방형 연결을 유지하고, 메모리에 상주하여 CGI 시작으로 인한 손실을 막을 수 있습니다.

Windows NT에서 Microsoft IIS로 ISAPI 인터페이스를 구성하려면 다음을 수행해야 합니다.

- 웹 서버에서 실행 권한을 가진 디렉토리에 가상 매핑을 합니다.
- driverISAPI.dll을 해당 디렉토리에 놓습니다.

- WEB.CNF 파일에서 다음과 같이 응용프로그램 기본값을 설정합니다.

  WEB_HOME /web/driverISAPI.dll?

큰 객체 캐싱

큰 객체와 페이지를 캐싱하여 성능을 향상시킬 수 있습니다. web.cnf 파일(이전 버전의 Illustra Web DataBlade에서는 web.conf 파일)에서 다음 값을 설정합니다.

	MI_WEBCACHEDIR	/web/l_o cache	웹 캐시를 위한 디렉토리를 지정합니다.
	MI_WEBCACHEMAXLO	1024000	캐시할 객체의 최대 크기를 바이트 단위로 설정합니다. 기본값은 64K입니다. 이 예에서는 1MB로 설정했습니다.
	MI_WEBCACHELIFE	2d	큰 객체가 캐시에 남아 있는 기간을 결정합니다. 이 예에서는 2일로 설정했습니다.
	MI_WEBCACHECRON	2h	캐시를 비우는 간격을 설정합니다. 이 예에서는 2시간마다 실행되도록 설정했습니다.
	MI_WEBCACHEPAGE	on	웹 페이지 캐싱을 설정합니다.

표 4: 큰 객체 캐싱

웹 페이지 캐싱을 활성화하면(웹 드라이버의 cache_enable 옵션으로 특정 페이지에 대한 캐싱이 활성화되어 있는 경우) 정적 페이지와 동적 페이지가 캐싱됩니다. 그러나 POST 메서드로 검색한 AppPage 나 HTTPHEADER 함수를 포함하는 AppPage 는 캐싱할 수 없습니다.

특정 페이지에 대한 캐싱을 활성화하려면 운영 체제의 프롬프트에서 다음 명령을 사용합니다.

webdriver cache_enable pagename

이 페이지에 대한 캐싱을 다시 비활성화하려면 다음 명령을 사용합니다.

webdriver cache_disable pagename

캐시에서 어떤 페이지를 제거하려면 다음 명령을 사용합니다.

webdriver cache_purge pagename

정적 페이지는 캐싱하는 것이 좋습니다. 동적 페이지는 전달되는 인수의 변화가 적은 경우에만 캐싱합니다. 페이지가 요구될 때마다 인수가 매번 다른 경우에는 캐싱을 하지 않는 것이 낫습니다.

페이지와 큰 객체의 캐싱한 사본은 web.cnf 파일에서 MI_WEBCACHEDIR 변수로 지정된 특정 디렉토리에 저장되므로, 캐싱한 객체를 포함하는 각 데이터베이스마다 디렉토리가 하나씩 생깁니다.

격리 수준

Illustra Server 의 기본 격리 수준은 Serializable 입니다. IDS/UDO 에서 Web DataBlade 를 사용할 때의 기본 설정은 CR(Committed Read)입니다. Serializable 은 페이지를 읽을 때 읽기 로크가 지속된다는 것을 의미하며, 다양한 용도로 사용하기에는 너무 제한적인 수준입니다. 제한을 완화시키려면 DR(Dirty Read)이나 CR(Committed Read) 수준을 사용합니다. 대부분의 응용프로그램에서 DR(Dirty Read)은 제한이 거의 없는 수준입니다. 그러나 웹 페이지를 읽을 때는 DR(Dirty Read)로 설정하고, 대부분의 데이터베이스 접근에서는 CR(Committed Read)로 설정하는 것이 적당합니다.

격리 수준을 DR 로 설정하면 로크가 되어 있어도 읽을 수 있습니다. 이것은 동시 실행을 최대로 가능하게 하지만 다른 사용자에 의해 로크가 되어 이미 수정되거나 삭제된 존재하지 않는 행이 결과에 나타날 수도 있습니다. (응용프로그램은 변경 내용을 임시로 적용하지만 전상회복됩니다.)

CR 은 DR 보다 다소 제한적인 수준이므로, 변경 사항이 모두 적용된 행만을 보여 줍니다. 그러나 이 설정에는 로크가 되어 있지 않으므로 같은 자료를 바로 다시 읽어도 다른 사용자들이 이를 변경했을 가능성이 있습니다.

SERIALIZABLE 옵션을 사용하면 해당 페이지에 로크가 설정되고, 이 페이지에 대한 MiSQL 태그가 추가되며 사용자를 위해 형식이 지정됩니다.

MiSQL 태그를 사용하여 작성한 페이지에 질의 격리 수준을 설정합니다(각주 2 참조).

격리 수준을 한 번 설정하면 이 페이지에 다른 격리 수준을 설정하기 전까지는, 다음에 추가하는 모든 MiSQL 태그들은 이 격리 수준으로 사용됩니다.

한 페이지 안에서 격리 수준을 변경하면 동적 페이지가 구축될 때 영향을 받는 것은 트랜잭션의 수명뿐입니다. 웹 서버 API 의 경우에는 동일한 스레드를 다시 사용하더라도 새 요구에 대한 격리 수준이 항상 기본 모드로 적용됩니다.

네트워크 사용량 최소화

네트워크 사용량을 최소화하려면 네트워크를 통해 클라이언트로 이동하는 자료의 양을 제한하고, 서버에서 필터링을 수행합니다. 질의에 일치되는 25 행으로 이루어진 둘째 세트에, MiSQL WINSTART 와 WINSIZE (이전의 MAXROWS) 옵션을 사용합니다.

```
〈?MiSQL WINSTART=26 WINSIZE=25

        SQL+"SELECT column FROM table;"〉 $1
```

개발 팁

페이지의 파일 크기를 작게 설계하거나 같이 추출되는 요소들의 크기를 작게 하면 페이지의 동시 실행성을 향상시킬 수 있습니다. 특히 HTML 형식으로 추출될 때 조합되는 동적 태그를 사용하여 섹션별로 페이지를 작성합니다. 태그를 사용하면, 웹으로 추출할 웹 페이지 섹션들을 모두 조합하는 것보다 오버헤드가 훨씬 적습니다.

사용자 정의 태그를 지정하면 공통된 페이지의 코드를 일관적으로 사용할 수 있으므로 작업의 효율을 높일 수 있습니다. 따라서 가능하면 태그를 사용하는 것이 좋습니다. 최근의 10~20개의 태그는 WebExplode에 의해 내부적으로 캐싱됩니다.

웹 페이지가 7,500바이트를 넘으면 IDS/UDO에 오버헤드가 추가로 생깁니다. 심각한 영향은 없지만 이렇게 되면, 페이지의 내용을 보려는 읽기가 하나 더 있는 것입니다.

복잡한 논리에 대해서는 내장 프로시저를 사용하십시오. 필수 SQL을 실행하는 데는 이 방법이 더 효율적인 경우가 있습니다. 필요하면, 보다 정밀한 오류 처리를 수행하기 위해 임시 테이블을 사용하고 웹 페이지를 간단하게 하십시오(각주 3 참조).

질의 플랜 추적과 검토

WEB.CNF 파일에서 다음과 같이 두 값을 설정하여 서버로 보내지는 응용프로그램의 SQL 문을 모두 캡처합니다.

	MI_WEBEXPLEVEL	32	실행되는 SQL 문을 로깅합니다.
	MI_WEBEXPLOG	/web/sql.log	trace 로그의 위치

표 5: 응용프로그램의 SQL 문을 캡처하기 위한 값

다음에 지정된 값들을 더하여 MI_WEBEXPLEVEL 설정 값을 결정합니다.

Trace 값	표시정보
1	HTML 주석 내용을 로깅(내용이 많으면 페이지가 표시되는 속도가 느려질 수 있음)
2	조건 정보를 로깅
4	태그의 속성 정보를 로깅
8	변수 정보를 로깅
16	동적 태그의 처리 결과를 로깅
32	실행되는 SQL 문을 로깅

표 6: Trace 값

사용자 자신의 정보를, trace 로그(MI_WEBEXPLevel 값이나, MI_WEBEXPLOG가 설정되어 있지 않으면 /tmp 디렉토리에 있는 파일로 지정할 수 있음)에 기록할 수도 있습니다. 다음과 같이 MiVAR 태그를 사용하여 레코드를 쓸 수 있습니다.

또한 서버에서 질의에 응답할 때 어떤 색인을 사용하는지 확인하십시오. 모든 질의에 대해 색인을 확인하는 것은 무리이나 키 값 질의에 대한 색인은 확인해 놓는 것이 좋습니다.

DBACCESS에서:

SET EXPLAIN ON:

SELECT column

FROM table

WHERE key = 1;

SET EXPLAIN OFF;

맺음말

Informix Web DataBlade Module 기반의 응용프로그램을 조정하는 것은 다른 조정 작업과 비슷합니다. Informix Web DataBlade Module을 사용하면 브라우저, 웹 서버, 데이터베이스 서버, 네트워크가 모두 사용되므로 어디부터 조정 작업을 시작해야 할지 결정하기가 쉬운 것만은 아닙니다. 이러한 작업 순서 결정에 도움이 될 수 있도록 여기서는 작업 순서를 선택하는 방법과 여러 가지 유용한 도구들에 대해 설명했습니다. 여러 웹 사이트에서 입증되었듯이 Informix Web DataBlade Module을 사용하여 고성능 응용프로그램을 개발할 수 있습니다.

Informix Dynamic Server Extended Parallel Option의 유틸리티 소개

소개

Informix Dynamic Server Extended Parallel Option(IDS/XPO, 버전 8)은 MPP(Massively Parallel Processing) 환경을 위한 Informix의 병렬 데이터베이스 서버입니다. IDS/XPO는 자료를 저장하기 위해 거대한 데이터웨어하우스를 사용하는 의사 결정 지원 시스템(DSS)을 지원합니다. IDS/XPO의 구조는 공유 자원이 없는 환경에서 MPP 시스템의 기능을 이용하도록 설계되었습니다. MPP 시스템은 두 개 이상의 시스템(노드)으로 구성되며 이들 노드들은 고속의 전용 네트워크로 연결되어 있습니다. 공유 자원이 없는 환경이란 각 노드들이 다른 노드들과 CPU, 메모리, 디스크 영역 같은 시스템 자원을 공유하지 않고 독립적으로 작업을 수행하는 것을 말합니다. 이렇게 공유 자원이 없는 환경에서 MPP 시스템의 기능을 이용하면 하나의 큰 질의를 여러 부분으로 나누어 모든 노드에서 병렬로 실행하여, 하나의 응답 세트를 만들 수 있기 때문에, 질의 처리에 소요되는 시간이 한 노드에서 질의를 처리할 때에 비해 줄어 듭니다.

IDS/XPO는 Informix Dynamic Server(OnLine Dynamic Server) 버전 7의 기본 코드에 공유 자원이 없는 MPP 환경을 지원하는 코드를 추가하였습니다. IDS/XPO에서는 MPP 시스템의 각 노드에 서버(IDS/XPO에서 coserver)를 배치하여 병렬 처리를 실행합니다. coserver는 분리된 별개의 서버로서 각각 DBMS의 기본 기능을 수행하지만, MPP 시스템의 병렬 처리 기능을 이용하여 여러 coserver에서 데이터베이스 업무를 나누어 처리할 수도 있습니다.

병렬 처리를 지원하기 위해 IDS/XPO에 다음과 같은 새로운 기능들이 추가되었습니다.

- 분할(Fragmentation) - 여러 노드에서 한 테이블을 분리하여 사용할 수 있는 기능
- 확장성(Scalability) - coserver의 수를 늘려 성능을 향상시킬 수 있는 기능
- 병렬 로딩 - 여러 coserver들 간에 자료를 로드하고 언로드할 수 있는 기능

Informix에서는 Informix Dynamic Server에서 기존에 사용되던 유틸리티들을 보강하였으며 특히 IDS/XPO의 병렬 처리를 지원하는 새로운 유틸리티들을 개발하였습니다. 여기서는 다음 표에 나열되어 있는 유틸리티들의 특징과 기능을 설명하고 이들의 용도에 대해 살펴보겠습니다.

	유틸리티 이름	설명	Informix Dynamic Server에서의 사용 여부
	XCTL	여러 데이터베이스 노드에서 유틸리티들을 동시에 실행할 수 있도록 합니다.	No
	ONINIT	IDS/XPO 환경을 시작하고 초기화합니다.	Yes
	ONUTIL	병렬 처리에 필요한 새로운 저장 공간을 정의하는 명령을 지원하는 데이터베이스 객체를 생성합니다.	No
	ONSTAT	IDS/XPO 환경을 모니터합니다.	Yes
	PLOAD	IDS/XPO 환경에서 병렬로 자료를 로드하고 언로드합니다.	No* *Informix Dynamic Server에도 유사한 이름을 가진 유틸리티가 있으나 IDS/XPO에서와 같은 기능을 수행하는 것은 아닙니다.

표 1: Informix Dynamic Server Extended Parallel Option 의

최신 및 갱신된 유틸리티

ONBAR(보관 유틸리티), IECC(Integrated Enterprise Command Center) 등의 기타 유틸리티들이 IDS/XPO에 추가되었습니다. 그러나 이 유틸리티들은 현재 베타 테스트 중입니다. 또한, DBACCESS 유틸리티는 IDS/XPO에서도 Informix Dynamic Server에서와 동일하게 사용되므로 여기서는 설명하지

않습니다.

유틸리티 위치

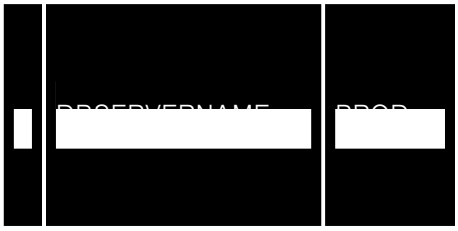
IDS/XPO를 설치하면, 표 1에 나열되어 있는 유틸리티들이 MPP 환경에서 지정된 노드와 시스템의 $InformixDIR/bin 디렉토리에 설치됩니다. 유틸리티가 설치되어 있지 않은 노드에서는 NFS를 사용하여 원하는 유틸리티들을 실행할 수 있습니다. 그러나 이렇게 하면 유틸리티와 실행 파일 크기, 사용 빈도에 따라 네트워크를 통해 전송되는 정보의 양이 너무 많아질 수 있습니다. 따라서 해당 노드에 설치되어 있지 않은 유틸리티를 실행하기 위해 NFS를 사용하는 것보다는 각 노드의 디렉토리에 유틸리티들을 복사해 두는 것이 좋습니다.

최신 및 갱신된 유틸리티

다음은 Informix Dynamic Server Extended Parallel Option의 최신 및 갱신된 유틸리티들에 대한 설명입니다.

xctl 유틸리티

IDS/XPO에 xctl 유틸리티가 추가되었습니다. 이 유틸리티는 두 개 이상의 데이터베이스 노드에서 동시에 같은 IDS/XPO 명령이나 UNIX 명령을 실행할 수 있게 합니다. 앞에서 설명했듯이 IDS/XPO 서버의 각 노드에는 coserver라고 하는 Informix 서버가 있습니다. 모든 노드와 coserver의 번호 및 이름은 데이터베이스 관리자(DBA)가 $InformixDIR/etc 디렉토리에 있는 onconfig.std 파일에 입력합니다. 이 구성 파일은 IDS/XPO 환경의 모든 coserver에서 사용됩니다. 다음의 표 2는 네 개의 노드로 구성된 IDS/XPO 환경에서 사용하는 onconfig.std 파일의 예입니다. 이 구성 파일에서는 서버의 이름이 "PROD"이고, 1 ~ 4번의 coserver가 있으며 노드 이름은 "node1"부터 "node4"까지 입니다.



	COSERVER NODE END	1 NODE1
	COSERVER NODE END	2 NODE2
	COSERVER NODE END	3 NODE3
	COSERVER NODE END	4 NODE4

표 2: 네 개의 노드로 구성된 IDS/XPO 환경에서

사용하는 onconfig.std 파일의 예

이 유틸리티를 실행하려면 시스템의 모든 노드 이름이 /etc/hosts 파일에 포함되어 있어야 합니다. 또한, $InformixDIR과 $ONCONFIG 환경 변수가 xctl 유틸리티를 실행하기 전에 먼저 설정되어야 합니다.

xctl 유틸리티 실행

다음의 표에서 보듯이 xctl 유틸리티에는 다양한 매개변수가 사용됩니다.

xctl 유틸리티는 Informix coserver가 있는 노드에서만 실행할 수 있습니다.

	옵션	설명
	-x *s*	's'의 값은 onconfig.std 파일에 정의되어 있는 coserver의 번호와 같습니다. Informix에서는 등호('=')를 coserver의 대체 기호로 사용할 것을 권장합니다.
	-b / -l	*xctl* 유틸리티를 백그라운드에서 실행하거나 대화형 명령으로 실행합니다.
	-w	각 노드에서 명령이 실행되기 전의 대기 시간입니다.
	-d	명령이 실행될 원격 디렉토리로서 명령을 개시한 노드에 있는 사용자의 홈 디렉토리가 기본으로 지정됩니다.
	-e	실행하기 전에 환경 변수를 내보냅니다.
	-c start-end	서버 번호의 범위입니다.

표 3: xctl 유틸리티의 매개변수

xctl 유틸리티를 실행하는 구문은 다음과 같이 두 부분으로 나뉩니다.

1. xctl 명령 구문

2. 각 노드에서 실행될 유틸리티 구문

다음 표를 참조하십시오.

	예	첫째 부분	둘째 부분
	1	*xctl* -b -c 1-8	*onstat* -D
	2	*xctl*	ps -fu Informix

표 4: xctl 유틸리티 구문 예

coserver 1 ~ 8번에서 백그라운드 모드로 Informix 입출력 모니터링 유틸리티인 onstat -D를 실행하려면 표 4의 예 1을 참조하십시오. 예 2는 Informix 사용자 계정을 사용하여 실행 중인 스레드를 배치하는 UNIX 명령을 실행하기 위해 xctl 명령을 사용합니다.

백그라운드 모드(-b) 옵션이 지정되어 있지 않으면 xctl 유틸리티를 사용한 결과는 화면에 출력됩니다. -b 옵션을 지정하면 xctl 유틸리티를 사용한 결과는 stdout나 stderr, 그리고 $DBSERVERNAME(coserver의 번호에 따라 1부터 N까지 있음) 파일에 저장됩니다. 이 출력 파일들은 백그라운드 모드에서 xctl 유틸리티를 처음 실행할 때 생성되며 다음에 이 유틸리티를 다시 실행하면 결과가 그 뒤에 계속 추가됩니다. 이 파일들을 새로 만들려면 먼저 기존 파일을 삭제한 다음 xctl 유틸리티를 다시 실행해야 합니다.

oninit 유틸리티

IDS/XPO 환경에서 oninit 유틸리티를 실행하려면 xctl 유틸리티를 사용해야 합니다. oninit 유틸리티는 IDS/XPO Informix 서버를 시작하고 초기화하기 위해 사용됩니다. 이 유틸리티를 xctl 유틸리티와 함께 사용하면 IDS/XPO 환경에 있는 모든 서버들을 하나의 명령문을 사용하여 시작하거나 초기화할 수 있습니다. 예를 들면 다음의 명령을 사용하여 모든 서버를 시작할 수 있습니다.

xctl -b -X= oninit -yX=

IDS/XPO Informix 서버를 시작하고 초기화하기 위해 oninit 유틸리티를 사용하지만 이 유틸리티를 사용하여 IDS/XPO 서버를 종료할 수는 없습니다. onmode 유틸리티는 IDS/XPO 서버를 종료하고 상태를 변경하기 위해 사용됩니다. 이 유틸리티와 xctl 유틸리티를 함께 사용하면 IDS/XPO 서버를 종료시킬 수 있습니다. 예를 들면 다음과 같은 명령을 수행할 수 있습니다.

xctl onmode -ky 이 명령은 모든 coserver를 종료시킵니다.
xctl onmode -sy 이 명령은 IDS/XPO를 대기 모드로 전환합니다.

onmode 유틸리티는 모든 coserver에서 xctl 유틸리티의 -X= 구문 없이 사용할 수 있습니다.

ONUTIL 유틸리티

병렬 IDS/XPO 환경을 관리하는 데 필요한 새로운 데이터베이스 객체를 지원하기 위해 ONUTIL 유틸리티가 Informix에 추가되었습니다. ONUTIL 유틸리티를 사용하여 실행할 수 있는 명령은 저장 공간 정의와 자료 일관성 검사 명령입니다. 다음 표는 이 유틸리티를 사용하여 실행할 수 있는 명령을 모두 나열한 것입니다.

	유틸리티 이름	지원 명령

		drop dbspace
		drop log
		drop logslice
		check table data
		check table info

표 5: ONUTIL 명령을 사용하여 실행할 수 있는

저장 공간 정의와 자료 일관성 검사 명령

이 유틸리티는 $PDQPRIORITY 환경 변수에 설정된 값에 상관없이 병렬 데이터베이스 질의(PDQ)의 우선 순위가 없다고 간주하여 실행됩니다. PDQ 환경 변수는 CPU, 메모리, 디스크 입출력 등 사용자가 이용하는 시스템 자원의 양을 제어합니다. PDQ 우선 순위가 100이면 모든 시스템 자원이 해당 결과를 얻기 위해 사용된다는 것을 의미합니다. PDQ 우선 순위는 1부터 100까지 사이의 값입니다.

ONUTIL 유틸리티를 실행하려면 반드시 Informix 사용자 계정을 사용해야 하며, 다음의 두 모드를 사용하여 UNIX 프롬프트에서 호출할 수 있습니다.

1. 대화형 모드
2. 파일 입력 모드

대화형 모드

대화형 모드에서 세션을 시작하려면 UNIX 프롬프트에서 ONUTIL유틸리티를 실행합니다. 다음은 ONUTIL의 예입니다.

1> create cogroup cogroup_odd from prod.1, prod.3, prod.5;

2> cogroup[successfully created

ONUTIL 세션에서 명령을 실행하려면 Enter 키를 누르기 전에 각 명령 뒤에 세미콜론을 넣어야 합니다.

대화형 세션을 끝내려면 Ctrl-D를 누릅니다.

파일 입력 모드

파일 입력 모드에서는 ONUTIL 유틸리티 다음에 파일 이름만 입력하거나 파일 이름과 -i 옵션을 같이 입력하여 유틸리티를 시작할 수 있습니다. -i 옵션을 사용하면 파일에 있는 모든 명령문들이 오류 발생 여부와 관계 없이 처리됩니다. 그러나 파일 이름만을 사용하는 경우에는 파일에 있는 명령문을 처리하는 과정에서 오류가 발생하면 프로세스가 중단됩니다.

ONUTIL과 매크로

매크로는 Informix coserver의 번호나 이름을 자동으로 생성해 줍니다.

번호 자동 생성 매크로를 사용하면 prod.1, prod.2, prod.3 등과 같이 각 coserver의 이름을 코딩할 필요 없이 하나의 명령문 구문에서 여러 노드를 지정할 수 있습니다. IDS/XPO 유틸리티 구문에서 여러 개의 서버 이름이 필요할 경우에 매크로를 사용해야 합니다. 다음의 표는 IDS/XPO에서 지원되는 매크로들을 나열한 것입니다.

	매크로 이름	설명

	%o	DB슬라이스 내의 DB영역 순서

표 6: 매크로 이름과 설명

다음 예는 %r 매크로를 사용하여 coserver 1부터 5까지를 cogroup_account라는 이름의 cogroup에 포함시킨 것입니다. cogroup이란 IDS/XPO 환경에서 관리를 용이하게 하기 위해 coserver를 하나의 이름을 사용하여 그룹으로 묶은 것입니다.

```
1> create cogroup cogroup_account from prod.%r(1..5);
2> cogroup[ successfully created
```

ONSTAT 유틸리티

onstat 유틸리티에서는 Informix에 할당된 공유메모리 세그먼트를 읽어 IDS/XPO 서버에서 진행되는 작업의 상태와 통계를 알려 줍니다. 이와 동일한 정보가 sysmaster 데이터베이스에 있는 시스템 감시 인터페이스(SMI) 테이블에도 기록되어 있으나, IDS/XPO에서는 이 시스템 테이블이 로크되는 것을 피하기 위해 공유메모리를 사용합니다.

- ONSTAT 유틸리티 실행

다음의 두 모드를 사용하여 UNIX 프롬프트에서 ONSTAT 유틸리티를 호출할 수 있습니다.

1. 대화형 모드
2. 단일 모드

- 대화형 모드

onstat -i 명령을 사용하여 onstat 유틸리티를 시작하면 대화형 세션이 설정됩니다. 이 세션에서는 여러 개의 옵션을 한 번에 하나씩 입력하고 실행할 수 있습니다. 각 명령이 종료되어도 onstat 세션이 계속되므로 추가로 명령을 실행할 수 있습니다.

맺음말

지금까지 IDS/XPO 유틸리티의 기능들을 살펴보았습니다. 일부 구문들은 복잡해 보이기도 하지만 IDS/XPO가 보다 발달하게 되면 이런 유틸리티들도 기능이 향상될 것이며, IECC(Integrated Enterprise Command Center) 도구를 사용하여 복잡한 명령과 유틸리티를 GUI(Graphical User Interface) 환경에서 쉽게 사용할 수 있게 될 것입니다. 또한 BMC, Compuware, Platinum 등의 타사 소프트웨어 공급자들이 보다 쉽고 빠르면서도 성능이 우수한 새로운 IDS/XPO 유틸리티를 개발할 예정입니다.