UNIT 08

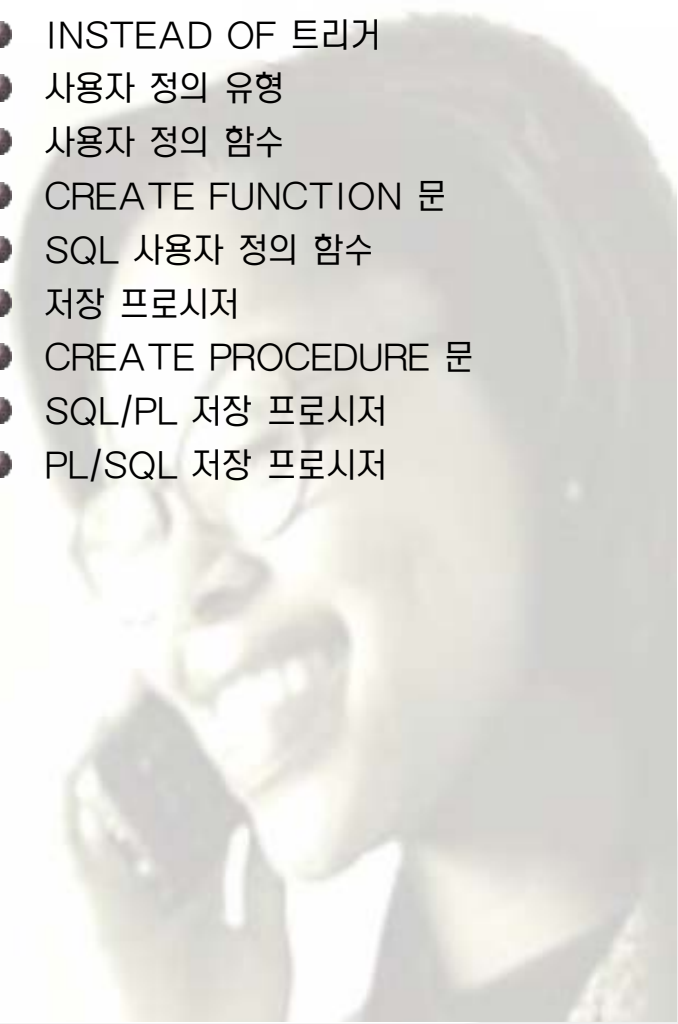
데이터베이스 오브젝트

한 데이터베이스에는 테이블, 뷰, 인덱스, 트리거, 시퀀스 등의 다양한 오브젝트들을 생성할 수 있습니다. 버퍼풀과 테이블스페이스를 제외한 여러 가지 데이터베이스 오브젝트에 대한 소개와 CREATE 문, ALTER 문, DROP 문을 이용한 생성, 변경, 제거 방법을 설명합니다.

DB2 9.7 운영자 가이드

Administrator Edition

- 데이터베이스 파티션 그룹
- 스키마
- 스키마 지정 방법
- 테이블
- CREATE TABLE 문
- ALTER TABLE 문
- 데이터 유형
- NULL 값과 DEFAULT 값
- 테이블스페이스 지정
- 고유키
- 기본키
- 외부키
- 참조 무결성
- 점검 제한 조건
- IDENTITY 컬럼
- NOT LOGGED INITIALLY 옵션
- 뷰
- CREATE VIEW 문
- MQT
- 인덱스
- CREATE INDEX문
- 시퀀스
- 트리거
- CREATE TRIGGER
- AFTER 트리거
- BEFORE 트리거
- INSTEAD OF 트리거
- 사용자 정의 유형
- 사용자 정의 함수
- CREATE FUNCTION 문
- SQL 사용자 정의 함수
- 저장 프로시저
- CREATE PROCEDURE 문
- SQL/PL 저장 프로시저
- PL/SQL 저장 프로시저



TOPIC 0801 데이터베이스 파티션 그룹

Point

DPF를 이용하여 다중 데이터베이스 파티션을 구성하면 데이터베이스 파티션의 묶음인 데이터베이스 파티션 그룹을 정의할 수 있습니다. CREATE DATABASE PARTITION GROUP, DROP DATABASE PARTITION GROUP 문으로 관리합니다.

1 데이터베이스를 생성하면 3개의 데이터베이스 파티션 그룹이 기본적으로 생성됩니다.

파티션그룹	설명
IBMCATGROUP	카탈로그 테이블스페이스가 생성되는 파티션입니다.
IBMDEFAULTTEMPGROUP	시스템 임시 테이블스페이스가 생성되는 파티션입니다.
IBMDEFAULTGROUP	사용자 테이블스페이스가 생성되는 기본 파티션입니다.

2 create database partition group 문의 형식은 다음과 같습니다.

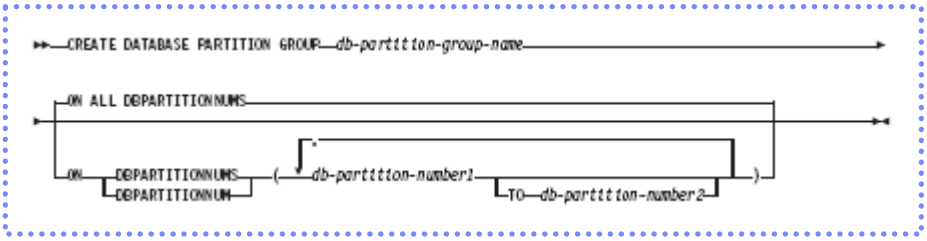


Figure 0801A • • • CREATE DATABASE PARTITION GROUP 문

옵션	설명
<DB 파티션 그룹명>	임의의 고유한 이름으로 지정합니다.
ON ALL DBPARTITIONNUMS	모든 파티션에 생성되도록 합니다.
ON DBPARTITIONNUMS	한 개 이상의 지정한 파티션에 생성되도록 합니다.
<파티션 번호>	db2nodes.cfg 파일에 정의된 파티션 번호입니다.

Tip

SYSADM, SYSCTRL 권한이 필요합니다.

3 create database partition group 명령어에서 한 개 이상의 데이터베이스 파티션 번호를 이용하여 새로운 데이터베이스 파티션 그룹을 생성합니다.

```
$ db2 "create database partition group <DB 파티션 그룹명>  on
    dbpartitionnum (<파티션 번호 1>, <파티션 번호 2>)"
```

4 drop database partition group 명령어로 데이터베이스 파티션을 제거합니다.

```
$ db2 "drop database partition group <DB 파티션 그룹명>"
$ db2 "drop nodegroup <DB 파티션 그룹명>"
```

Tip

IBMDEFAULTTEMPGROUP 은 list nodegroups 명령어로 표시되지 않습니다.

5 list database partition groups 명령어를 이용하여 정의된 데이터베이스 파티션 그룹의 정보를 확인합니다. list nodegroups 명령어를 사용해도 됩니다.

```
$ db2 list database partition groups  show detail
```

6 SYSCAT.NODEGROUPS 뷰에서 연관된 정보를 확인합니다.

```
$ db2 list database partition groups   show detail
```

TOPIC 08 02 스키마

Point

데이터베이스 오브젝트의 이름은 <스키마명>.<오브젝트명> 과 같이 2-part 형식으로 구성됩니다. 스키마는 오브젝트의 이름을 수식하는 수식자 역할을 합니다. CREATE SCHEMA, DROP SCHEMA 문으로 관리합니다.

1 데이터베이스를 생성하면 다음과 같이 4가지의 스키마가 기본적으로 생성됩니다.

스키마	설명
SYSIBM	시스템 카탈로그 테이블의 스키마입니다.
SYSCAT	시스템 카탈로그 뷰의 스키마입니다.
SYSSTAT	통계 자료와 관련된 시스템 카탈로그 뷰의 스키마입니다.
SYSFUN	기본적으로 제공되는 사용자 정의 함수의 스키마입니다.

2 create schema 문의 형식은 다음과 같습니다.

옵션	설명
<스키마명>	임의의 고유한 이름으로 지정합니다.
AUTHORIZATION	스키마의 소유자를 지정합니다.

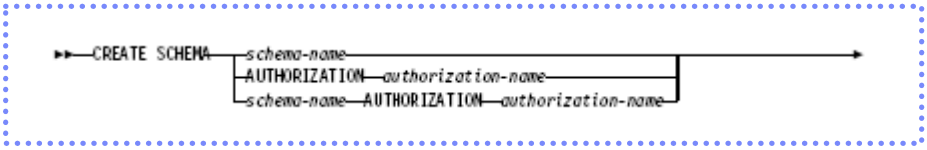


Figure 0802A CREATE SCHEMA 문

3 create schema 문을 이용하여 새로운 스키마를 생성합니다.

```
$ db2 "create schema <스키마명>"
```

4 create schema 문에서 AUTHORIZATION 옵션을 이용하여 스키마의 소유자를 지정할 수 있습니다.

```
$ db2 "create schema <스키마명> AUTHORIZATION <스키마의 소유자명>"
```

5 drop schema 문을 이용하여 기존의 스키마를 제거합니다. 반드시 RESTICT 옵션을 지정하도록 합니다.

```
$ db2 "drop schema <스키마명> restrict"
```

6 list tables 명령어에서 FOR SCHEMA 옵션을 이용하면 동일한 스키마를 가지는 테이블과 뷰의 목록을 확인할 수 있습니다.

```
$ db2 list tables for schema <스키마명>
```

7 생성된 스키마에 대한 정보는 SYSCAT.SCHEMA 뷰를 이용해서 확인합니다.

```
$ db2 "select * from syscat.schemata"
```

Tip
SYSADM, DBADM 권한이 필요합니다.

Tip
SYS로 시작되는 스키마명은 사용하지 않도록 합니다.

Tip
새로 생성된 스키마를 이용하려는 사용자는 스키마의 소유자로부터 CREATEIN, ALTERIN, DROPIN 등의 특권을 부여받아야 합니다.

Tip
제거하려는 스키마명을 가진 오브젝트가 존재하면, SQL0478N 오류 코드가 반환되고, 스키마는 제거되지 않습니다.

TOPIC 08 03 스키마 지정 방법

Point

테이블 등의 데이터베이스 오브젝트의 이름을 명시할 때는 <스키마명>.<오브젝트명> 형식의 2-part name 을 사용하는 것이 원칙입니다. <스키마명>을 명시적으로 지정하지 않으면 접속 사용자명이 스키마 명입니다.

1. SQL문에서 <스키마명> 없이 <테이블명>만 지정하면, <현재 세션의 로그온 사용자명>이 기본 <스키마명>으로 인식됩니다. <테이블명>은 <사용자명1>.<테이블명>으로 인식됩니다.

```
$ login <사용자명1>
$ db2 connect to <데이터베이스명>
$ db2 "select * from <테이블명>"
```

2. 데이터베이스에 접속하는 connect 문에서 USER 와 USING 옵션을 이용하면, <현 세션의 로그온 사용자명>에 관계 없이 <데이터베이스 접속 시에 사용된 사용자명>이 기본 <스키마명>으로 인식됩니다. <테이블명>은 <사용자명2>.<테이블명>으로 인식됩니다.

```
$ login <사용자명1>
$ db2 connect to <데이터베이스명> user <사용자명2> using <암호명2>
$ db2 "select * from <테이블명>"
```

Tip

set current schema 문은 데이터베이스에 접속한 상태에서실행할 수 있으며, 접속이 해제되면 <로그온 사용자명>으로 복원됩니다.

3. CURRENT SCHEMA 특수 레지스터리 변수는 스키마명을 명시적으로 지정하지 않는 경우에 기본 스키마로 적용될 값을 저장하고 있습니다. values 문으로 현재값을 확인할 수 있습니다. set current schema 문으로 CURRENT SCHEMA 특수 레지스터리 변수를 변경하면, <데이터베이스 접속시 사용된 사용자명> 보다 우선적으로 적용됩니다. <테이블명>은 <스키마명 1>.<테이블명>으로 인식됩니다.

```
$ login <사용자명1>
$ db2 connect to <데이터베이스명> user <사용자명2> using <암호명2>
$ db2 values(current schema)
$ db2 set current schema <스키마명1>
$ db2 values(current schema)
$ db2 "select * from <테이블명>"
```

4. 데이터베이스의 오브젝트를 지정할 때는 개별적인 SQL문에서 <스키마명>을 명시적으로 지정하는 것이 권장됩니다. <현재 세션의 로그온 사용자명>, <데이터베이스 접속시에 사용된 사용자명>, <CURRENT SCHEMA 특수 레지스터리 변수의 현재값> 보다 SQL문에서 명시적으로 지정한 <스키마명>이 가장 우선적으로 적용됩니다. <테이블명>은 <스키마명2>.<테이블명>로 인식됩니다.

```
$ login <사용자명1>
$ db2 connect to <데이터베이스명> user <사용자명2> using <암호명2>
$ db2 set current schema <스키마명1>
$ db2 "select * from <스키마명2>.<테이블명>"
```

TOPIC 0804 테이블

Point

사용자의 데이터는 테이블에 저장됩니다. 테이블의 데이터는 기본적으로 한 개의 테이블스페이스에 저장됩니다. CREATE TABLE, ALTER TABLE, DROP TABLE 문으로 관리합니다.

Tip

테이블스페이스의 페이지 크기 별 제약 사항은 정보 센터를 참조합니다.

Tip

테이블의 한 행의 총 길이는 지정한 테이블 스페이스의 페이지 크기보다 작아야 합니다.

1 한 테이블스페이스에는 한 개 이상의 테이블이 저장됩니다.

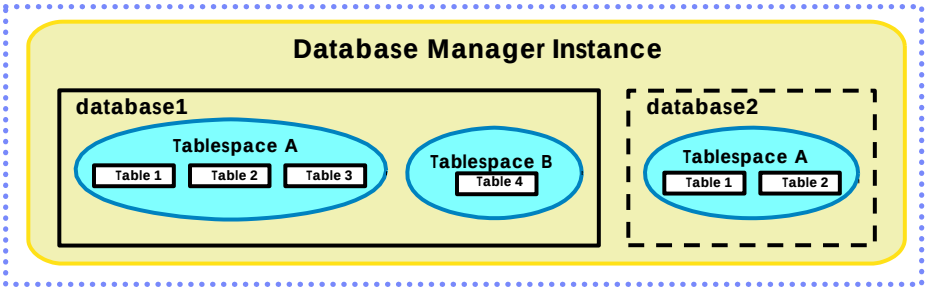


Figure 0804A ••• 테이블스페이스와 테이블

2 create table 문을 이용하여 테이블을 정의합니다. IMPLICIT_SCHEMA 특권이 있으면, 존재하지 않는 <스키마명>을 이용하여 테이블을 정의할 수 있습니다.

```
$ db2 "create table <스키마명>.<테이블명> (<컬럼정의>)"
```

3 alter table 문을 이용하여 컬럼 추가, 고유키 추가 및 제거, 기본키 추가 및 제거, 외부키 추가 및 제거, 점검 제한 조건 추가 및 제거 등의 변경 작업이 가능합니다.

```
$ db2 "alter table <스키마명>.<테이블명> ADD <제한조건>"
```

4 drop table문으로 테이블을 제거합니다.

```
$ db2 "drop table <스키마명>.<테이블명>"
```

5 list tables 명령어에서 테이블의 목록을 확인할 수 있습니다.

```
$ db2 list tables
$ db2 list tables for schema <스키마명>
$ db2 list tables for system
$ db2 list tables for all
```

6 테이블에 대한 정보는 SYSCAT.TABLES 뷰를 이용해서 확인합니다.

```
$ db2 "select * from syscat.tables"
```

7 describe table 문으로 테이블의 컬럼에 대한 정보를 확인합니다.

```
$ db2 "describe table <스키마명>.<테이블명>"
```

8 db2look 명령어로 테이블에 대한 DDL문을 추출할 수 있습니다.

```
$ db2look –d <DB명> –e –z <스키마명> –t <테이블명> -o <출력파일명>
```

TOPIC 08 05 CREATE TABLE 문

Point

테이블을 생성할 때 이용하는 SQL문입니다. 컬럼, 고유키, 기본키, 외부키, 점검 제한 조건 등을 정의하고, 데이터와 인덱스를 저장할 테이블스페이스를 지정합니다.

Tip

SYSADM, DBADM 권한 또는 데이터베이스에 대한 CREATAB 특권이 필요할 수 있습니다.

Tip

새로운 스키마명을 이용하여 테이블명을 지정하려면, 데이터베이스에 대한 IMPLICIT_SCHEMA 특권이 필요합니다.

Tip

기존의 스키마명을 이용하여 테이블명을 지정하려면, 스키마에 대한 CREATIN 특권이 필요합니다.

Tip

IN, INDEX IN, LONG IN 옵션을 사용하려면 테이블스페이스에 대한 USE 특권이 필요합니다.

Tip

외부키를 정의하려면, 부모 테이블에 대한 REFERENCES 특권이 필요합니다.

Tip

NOT LOGGED INITIALLY 특성은 ALTER TABLE 문으로 활성화할 때만 적용됩니다.

1 create table 문의 형식은 다음과 같습니다.

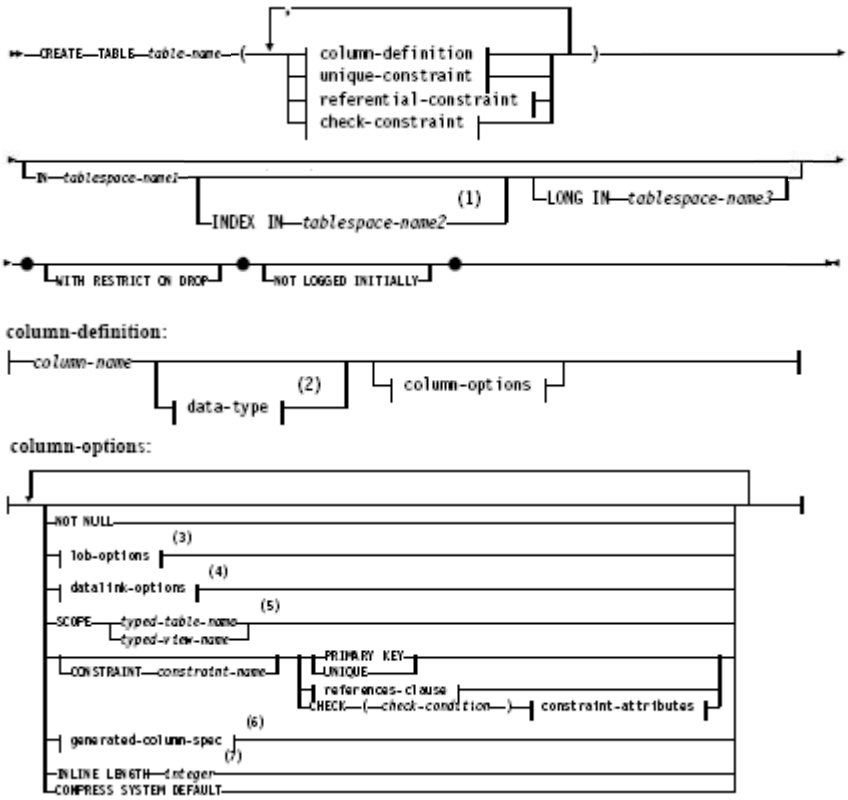


Figure 0804A ••• CREATE TABLE 문

2 옵션에 대한 설명은 다음과 같습니다.

옵션	설명
<테이블명>	임의의 고유한 이름으로 지정합니다. <스키마명>을 생략하면 CURRENT SCHEMA 특수 레지스터리 변수의 값이 기본 스키마명으로 사용됩니다. 기존의 스키마명을 사용하려면,
<컬럼명>	테이블 내에서 고유한 임의의 이름으로 지정합니다.
<데이터 유형>	컬럼의 데이터 유형을 지정합니다.
<NULL 허용 여부>	NULL값을 허용하지 않으려면 'NOT NULL' 로 지정합니다.
<기본값>	WITH DEFAULT 옵션으로 기본값을 지정합니다.
IN <TS명>	테이블의 데이터가 저장될 테이블스페이스명을 지정합니다.
INDEX IN <TS명>	인덱스 데이터가 저장될 테이블스페이스명을 지정합니다.
LONG IN <TS명>	LONG 데이터가 저장될 테이블스페이스명을 지정합니다.
NOT LOGGED INITIALLY	트랜잭션에서 해당 테이블에 대한 변경 사항을 로그에 기록하지 않게 합니다.

TOPIC 08 06 ALTER TABLE 문

Point

테이블의 특성을 변경할 때 사용하는 SQL문입니다. 컬럼 추가, 고유키 추가 및 제거, 기본키 추가 및 제거, 외부키 추가 및 제거, 점검 제한 조건 추가 및 제거 등의 변경 작업이 가능합니다.

Tip

SYSADM, DBADM 권한 또는 테이블에 대한 CONTROL, ALTER 특권이 필요할 수 있습니다.

Tip

기존의 스키마명을 이용하여 테이블을 변경하려면, 스키마에 대한 ALTERIN 특권이 필요합니다.

Tip

외부키를 정의하려면, 부모 테이블에 대한 REFERENCES 특권이 필요합니다.

1 alter table 문의 형식은 다음과 같습니다.

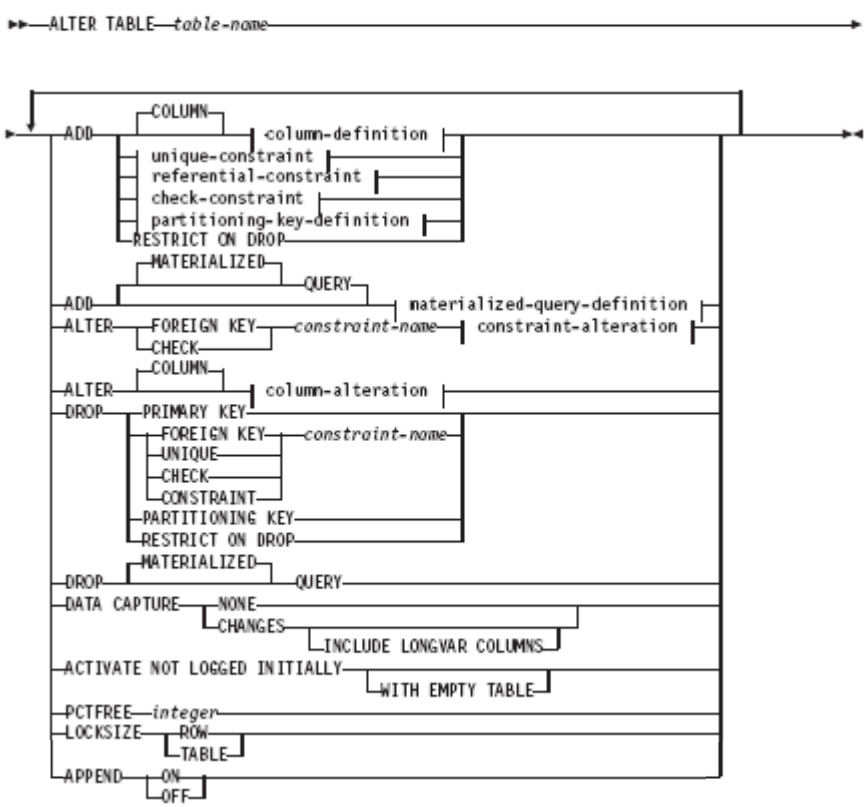


Figure 0805A ··· ALTER TABLE 문

2 옵션에 대한 설명은 다음과 같습니다.

옵 션	설 명
<테이블명>	변경할 테이블명을 지정합니다.
ADD <컬럼정의>	컬럼을 추가합니다.
ADD <고유키제한조건>	고유키를 추가합니다.
ADD <기본키제한조건>	기본키를 추가합니다.
ADD <외부키제한조건>	외부키를 추가합니다.
ADD <점검제한조건>	점검 제한 조건을 추가합니다.
ALTER <컬럼명>	VARCHAR 유형에서 컬럼의 길이를 증가시킬 수 있습니다.
DROP <제한조건명>	지정된 <제한조건>을 제거합니다.
ACTIVATE NOT LOGGED INITIALLY	NOT LOGGED 모드로 전환합니다. 트랜잭션이 종료될 때까지는 해당 테이블에 대한 변경 작업이 로깅되지 않습니다.

TOPIC 0807 데이터 유형

Point

기본적으로 지원되는 컬럼의 데이터 유형은 다음과 같습니다. CREATE DISTINCT TYPE 문으로 사용자가 새로운 데이터 유형을 추가로 생성하여 사용할 수도 있습니다.

1 기본적으로 지원되는 데이터의 유형은 다음과 같습니다.

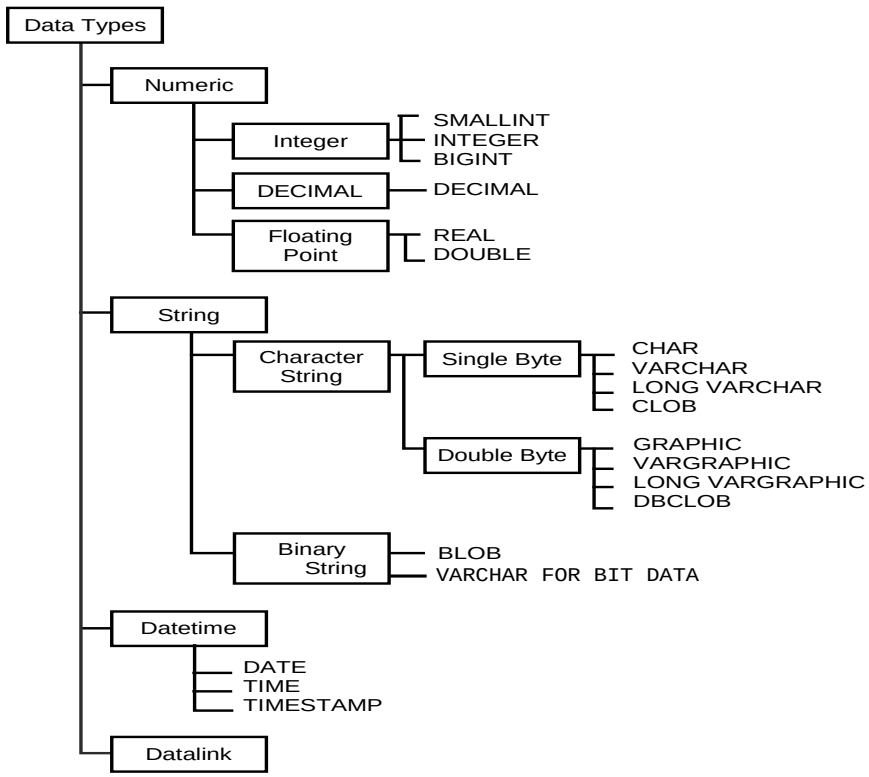


Figure 0807A ··· 데이터 유형

2 대표적인 데이터 유형에 대한 설명은 다음과 같습니다.

구 분	유 형	저장 BYTE 수	최 대 범 위
숫 자	SMALLINT	2	-32,768 ~ +-32,767
	INT	4	-2,147,483,648 ~ +2,147,483,647
	BIGINT	8	-9,223,372,036,854,775,808 ~ +9,223,372,036,854,775,807
	DEC(p,s)	(p+s)/2+1	31 자리
	DOUBLE	8	-1.79769E+308 ~ +1.79769E+308
문 자	CHAR(n)	n	254 바이트
	VARCHAR(n)	n + 4	32672 바이트 (32K 페이지인 경우)
날 짜	DATE	10	0001-01-01 ~ 9999-12-31
	TIME	8	00:00:00 ~ 24:00:00
	TIMESTAMP	26	0001-01-01-00.00.00.000000 ~ 9999-12-31-24.00.00.000000

Tip

VARCHAR의 최대 길이는 페이지 크기에 따라 다릅니다.

Tip

LOB 유형의 최대 길이는 2G입니다 .

TOPIC 0808 NULL 값과 DEFAULT 값

Point

컬럼에 NULL 값과 DEFAULT 값을 허용하게 할 수 있습니다. DEFUALT 속성을 가지지 않는 컬럼은 반드시 명시적으로 값을 지정해야 합니다. CREATE TABLE 문에서 NOT NULL 옵션과 WITH DEFULAT 옵션을 이용합니다.

Tip

NULL 값은 0 또는 공백 (blank) 또는 empty string 이 아닙니다. empty string은 길이가 0인 값을 의미하며, 공백 문자와는 다릅니다.

1. NULL 값은 알려지지 않은 값을 의미합니다. 테이블을 정의할 때, 컬럼에 NULL 값을 허용하지 않으려면 CREATE TABLE 문에서 NOT NULL 옵션을 이용합니다.

```
$ db2 "create table <테이블명> ( <컬럼명>  <데이터유형명> NOT NULL, ....)"
```

2. CREATE TABLE 문에서 WITH DEFAULT 옵션만 지정하면 시스템 기본값이 제공됩니다.

```
$ db2 "create table <테이블명> ( <컬럼명>  <데이터유형명> WITH DEFAULT)"
```

유 형	기본값	설 명
숫자	0	0
문자	EMPTY STRING	길이가 0 인 문자
DATE	CURRENT DATE	현재 시스템 날짜
TIME	CURRENT TIME	현재 시스템 시간
TIMESTAMP	CURRENT TIMESTAMP	현재 시스템 시간 소인

Tip

날짜 유형의 표현식은 데이터베이스의 코드 페이지에 따라 달라집니다. 코드 페이지가 970(ko_KR)인 경우에 DATE 유형은 'yyyy-mm-dd', TIME 유형은 'hh:mm:ss' 가 되고, TIMESTAMP 유형은 'yyyy-mm-dd-hh.mm.ss.uuuuuu' 으로 표현됩니다.

3. CREATE TABLE 문에서 WITH DEFAULT <기본값> 옵션을 지정하면 사용자가 지정한 값이 기본값으로 사용됩니다.

```
$ db2 "create table <테이블명> ( <컬럼명>  <데이터유형명> WITH DEFAULT
    <기본값>)"
```

유 형	기 본 값	예
숫자	WITH DEFAULT <숫자>	WITH DEFAULT 10
문자	WITH DEFAULT '<문자열>'	WITH DEFAULT 'xx'
DATE	WITH DEFAULT '<날짜>'	WITH DEFAULT '2006-04-17'
TIME	WITH DEFAULT '<시간>'	WITH DEFAULT '14:12:30'
TIMESTAMP	WITH DEFAULT '<시간소인>'	WITH DEFAULT '2006-04-17-14.12.30.694001'

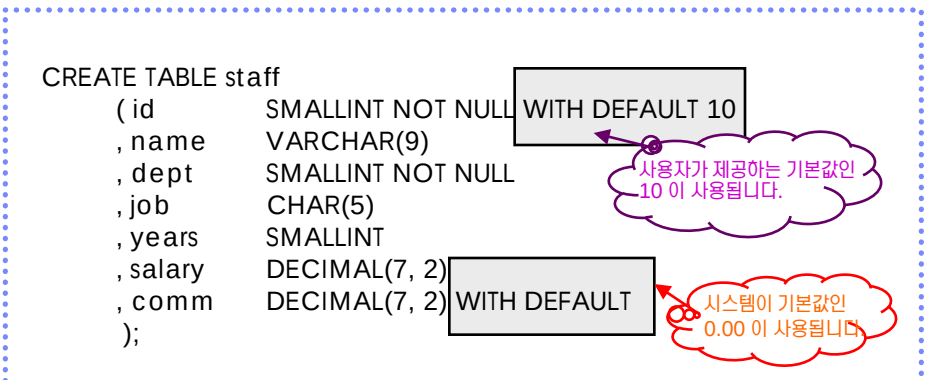


Figure 0808A • • • 기본값 지정

TOPIC 08 09 테이블스페이스 지정

Point

CREATE TABLE 문에서 IN 키워드를 이용하여 테이블스페이스를 지정할 수 있습니다. INDEX IN, LONG IN 키워드로 테이블, 인덱스, LONG 데이터를 개별적인 DMS 테이블스페이스에 저장할 수 있습니다. 지정한 테이블스페이스는 변경될 수 없습니다.

Tip

해당 테이블의 행의 총 길이를 수용할 수 있는 적합한 기본 테이블스페이스가 없다면, SQL0286N 오류가 반환됩니다.

Tip

특정한 테이블 스페이스를 지정하려면, 해당 테이블 스페이스에 대한 USE 특권이 있어야 합니다.

Tip

사용자가 정의한 테이블스페이스 중에서 해당 테이블의 행의 총 길이를 수용할 수 있는 페이지 크기를 가진 첫 번째 REGULAR 유형의 테이블스페이스가 기본 사용자 테이블스페이스로 사용됩니다.

Tip

사용자가 정의한 테이블스페이스가 없다면, USERSPACE1이 기본 사용자 테이블스페이스입니다.

Tip

한 테이블이 여러 테이블스페이스에 저장되었다면, 테이블스페이스는 함께 drop 되어야 합니다.

1. CREATE TABLE 문에서 IN 옵션을 지정하지 않으면, 테이블은 기본 사용자 테이블스페이스에 저장됩니다.

```
$ db2 "create table <스키마명>.<테이블명> (<컬럼 정의>) "
```

2. CREATE TABLE 문에서 IN 옵션으로 테이블이 저장될 테이블스페이스를 지정합니다. 테이블의 모든 데이터와 인덱스 데이터는 동일한 테이블스페이스에 저장됩니다.

```
$ db2 "create table <스키마명>.<테이블명> (<컬럼 정의>) IN <테이블스페이스명>"
```

3. CREATE TABLE 문에서 INDEX IN 키워드를 이용하여 인덱스를 위한 데이터를 별도의 테이블스페이스에 저장할 수 있습니다. IN 옵션과 INDEX IN 옵션에서 지정한 테이블스페이스는 DMS 방식의 REGULAR 유형이어야 합니다. INDEX IN 옵션만 지정할 수는 없습니다.

```
$ db2 "create table <스키마명>.<테이블명> (<컬럼 정의>) IN <테이블스페이스명> INDEX IN <테이블스페이스명> "
```

4. CREATE TABLE 문에서 LONG IN 키워드를 이용하여 LONG 데이터를 별도의 테이블스페이스에 저장할 수 있습니다. IN 옵션에서 지정한 테이블스페이스는 DMS 방식의 REGULAR 유형이고, LONG IN 옵션에서 지정한 테이블스페이스는 DMS 방식의 LARGE 유형이어야 합니다. LONG IN 옵션만 지정할 수는 없습니다.

```
$ db2 "create table <스키마명>.<테이블명> (<컬럼 정의>) IN <테이블스페이스명> LONG IN <테이블스페이스명> "
```

5. CREATE TABLE 문에서 IN, INDEX IN, LONG 옵션을 모두 사용하여 테이블 데이터, 인덱스 데이터, LONG 데이터를 별도의 DMS 테이블스페이스를 저장할 수 있습니다.

```
$ db2 "create table <스키마명>.<테이블명> (<컬럼 정의>) IN <테이블스페이스명> INDEX IN <테이블스페이스명> LONG IN <테이블스페이스명> "
```

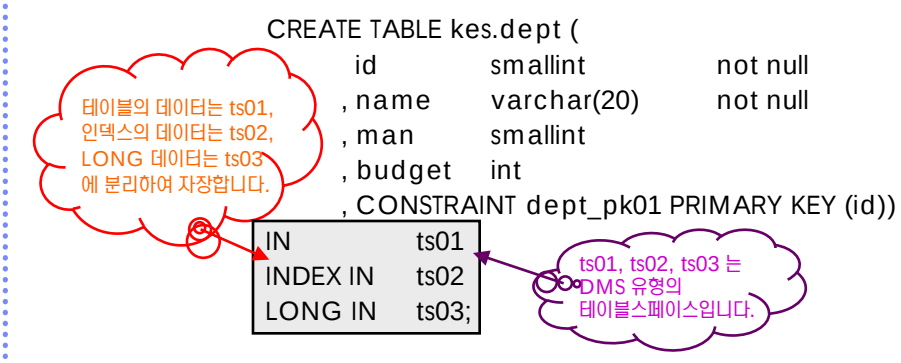


Figure 0809A ••• 테이블스페이스 지정

TOPIC 08-10 고유키

Point

고유 키는 한 개 이상의 컬럼들로 구성되어 테이블의 각 행을 고유하게 구별하는 값입니다. 한 테이블에 한 개 이상의 고유키를 지정할 수 있습니다. 고유 키를 정의하면, 해당 컬럼들로 구성된 고유 인덱스가 자동으로 생성됩니다.

Tip

고유키에 대응하는 인덱스가 이미 존재하면, SQL0000W 라는 경고 메시지가 반환되지만, 무시해도 됩니다.

Tip

<제한조건명>은 데이터베이스 내에서 고유해야 합니다.

Tip

<제한조건명>을 명시하지 않으면, 엔진이 생성한 이름으로 관리하게 됩니다. ALTER 또는 DROP문으로 관리하려면, 사용자가 제한조건명을 명시하는 것이 편리합니다.

1 고유키 제한 조건을 정의하는 구문은 다음과 같습니다. 고유키를 구성하는 각 컬럼은 NOT NULL 속성을 지정해야 합니다.

Figure 0810A ••• 고유키 제한 조건절

2 create table 문에서 CONSTRAINT ~ UNIQUE 라는 옵션으로 지정합니다. <제한조건명>과 동일한 이름을 가진 고유 인덱스가 자동으로 생성됩니다. 고유키는 한 테이블에 여러 개 정의할 수 있습니다. CONSTRAINT 옵션을 지정하지 않으면, 제한조건명과인덱스의 이름은 'SQLyymmddhhmmssxxx' 형식으로 엔진이 부여합니다.

```
$ db2 "create  table <스키마명>.<테이블명> (..., <고유키 제한 조건절>, ...)"
```

3 alter table 문을 이용하여 고유키를 추가할 수 있습니다.

```
$ db2 "alter  table <스키마명>.<테이블명> ADD <고유키 제한 조건절>"
```

4 alter table 문을 이용하여 고유키를 제거할 수 있습니다.

```
$ db2 "alter  table <스키마명>.<테이블명> DROP CONSTRAINT  <제한조건
    명>"
```

```
CREATE TABLE kes.empl (
        id          smallint        not null
      , name        varchar(30)     not null
      , sex         char(1)
      , mydept      smallint
      , salary      smallint
      , email       varchar(30)     not null
      , hiredate    date
      , CONSTRAINT empl_uk01 UNIQUE(email)
);

ALTER TABLE kes.empl DROP CONSTRAINT empl_uk01;

ALTER TABLE kes.empl ADD CONSTRAINT empl_uk01
                     UNIQUE (email) ;
```

email 컬럼은 NOT NULL로 지정합니다.

email 컬럼이 고유키가 됩니다.

제한조건명은 empl_uk01 입니다.

자동으로 생성되는 고유 인덱스의 이름도 empl_uk01 입니다.

Figure 0810B ••• 고유키 생성

TOPIC 08 11 기본키

Point

기본키는 고유 키와 동일한 특성을 갖지만, 한 테이블에 한 개만 지정할 수 있습니다. 기본키는 한 개 이상의 컬럼들로 구성될 수 있으며, 해당 컬럼들로 구성된 고유 인덱스가 자동으로 생성됩니다.

Tip

고유키와 기본키는 기능적으로 동일합니다. 단, IMPORT 명령어의 INSERT_UPDATE 옵션을 사용할 때에는 반드시 기본키가 필요합니다.

Tip

고유키에 대응하는 인덱스가 이미 존재하면, SQL0000W 라는 경고 메시지가 반환되지만, 무시해도 됩니다.

Tip

<제한조건명>은 데이터베이스 내에서 고유해야 합니다.

Tip

<제한조건명>을 명시하지 않으면, 엔진이 생성한 이름으로 관리하게 됩니다. ALTER 또는 DROP문으로 관리하려면, 사용자가 제한조건명을 명시하는 것이 편리합니다.

1 기본키 제한 조건을 정의하는 구문은 다음과 같습니다. 기본키를 구성하는 각 컬럼은 NOT NULL 속성을 지정해야 합니다.

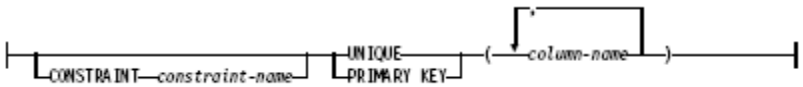


Figure 0811A ••• 기본키 제한 조건절

2 create table 문에서 CONSTRAINT ~ PRIMARY KEY 라는 옵션으로 지정합니다. <제한조건명>과 동일한 이름을 가진 고유 인덱스가 자동으로 생성됩니다. 기본키는 한 테이블에 한 개만 정의할 수 있습니다. CONSTRAINT 옵션을 지정하지 않으면, 제한조건명과 인덱스의 이름은 'SQLyymmddhhmmssxxx' 형식으로 엔진이 부여합니다.

```
$ db2 "create  table <스키마명>.<테이블명> (..., <기본키 제한 조건절>, ...)"
```

3 alter table 문을 이용하여 기본키를 추가할 수 있습니다.

```
$ db2 "alter  table <스키마명>.<테이블명> ADD <기본키 제한 조건절>"
```

4 alter table 문을 이용하여 기본키를 제거할 수 있습니다.

```
$ db2 "alter  table <스키마명>.<테이블명> DROP CONSTRAINT  <제한조건
    명>"
```

```
CREATE TABLE kes.empl (
        id          smallint        not null
      , name        varchar(30)     not null
      , sex         char(1)
      , mydept      smallint
      , salary      smallint
      , email       varchar(30)     not null
      , hiredate    date
      , CONSTRAINT empl_pk01 PRIMARY KEY (id)
);

ALTER TABLE kes.empl DROP CONSTRAINT empl_pk01 ;

ALTER TABLE kes.empl ADD CONSTRAINT empl_pk01
                     PRIMARY KEY(id) ;
```


Figure 0811B ••• 기본키 생성

TOPIC 08-12 외부키

Point

외부 키는 부모 테이블의 고유키 또는 기본키를 참조하는 키입니다. 참조하는 고유키 또는 기본키와 호환되는 컬럼들로 구성되어야 하며, 각 컬럼은 NULL 값을 허용합니다.

Tip
<제한조건명>은 데이터베이스 내에서 고유해야 합니다.

Tip
<제한조건명>을 명시하지 않으면, 엔진이 생성한 이름으로 관리하게 됩니다. ALTER 또는 DROP문으로 관리하려면, 사용자가 제한조건명을 명시하는 것이 편리합니다.

Tip
외부키는 SQL문에서 부모 테이블의 기본키 또는 고유키와 JOIN 하게 되므로, 외부키에 대한 인덱스를 생성하는 것이 좋습니다.

1. 외부키 제한 조건을 정의하는 구문은 다음과 같습니다.

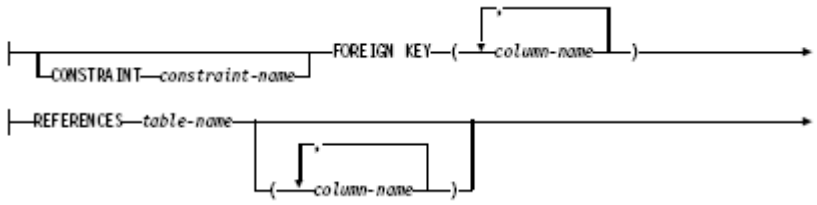


Figure 0812A ••• 외부키 제한 조건절

2. create table 문에서 CONSTRAINT ~ FOREIGN KEY 라는 옵션으로 지정합니다. 외부키는 한 테이블에 여러 개 정의할 수 있습니다. CONSTRAINT 옵션을 지정하지 않으면, 제한조건명은 'SQLyymmddhhmmssxxx' 형식으로 엔진이 부여합니다.

```
$ db2 "create  table <스키마명>.<테이블명> (..., <외부키 제한 조건절>, ...)"
```

3. alter table 문을 이용하여 외부키를 추가할 수 있습니다.

```
$ db2 "alter  table <스키마명>.<테이블명> ADD <외부키 제한 조건절>"
```

4. alter table 문을 이용하여 외부키를 제거할 수 있습니다.

```
$ db2 "alter  table <스키마명>.<테이블명> DROP CONSTRAINT  <제한조건명>"
```

```
CREATE TABLE kes.empl (
        id          smallint        not null
      , name        varchar(30)
      , sex         char(1)
      , mydept      smallint
      , salary      smallint
      , email       varchar(30)     not null
      , hiredate    date
      , CONSTRAINT empl_fk01 FOREIGN KEY(mydept)
        REFERENCES kes.dept
);

ALTER TABLE kes.empl DROP CONSTRAINT empl_fk01

ALTER TABLE kes.empl
        ADD CONSTRAINT empl_fk01 FOREIGN KEY(mydept)
        REFERENCES kes.dept ;
```

mydept 컬럼은 NULL 값을 허용할 수 있습니다.

mydept 컬럼이 외부키가 됩니다.

부모 테이블의 이름은 kes.dept 입니다.

제한조건명은 empl_fk01 입니다.

Figure 0812B ••• 외부키 생성

TOPIC 08-13 참조 무결성

Point

두 테이블이 고유키와 외부키로 연결되어 외부키를 가진 테이블에 데이터를 추가, 변경하는 경우에 데이터의 참조 무결성이 유지됩니다. 고유키를 가진 테이블에 데이터를 변경, 제거하는 경우에는 UPDATE 규칙과 DELETE 규칙이 적용됩니다.

Tip

부모 테이블에 INSERT문을 실행할 때, 점검 규칙은 필요하지 않습니다.

Tip

외부키는 자신의 테이블에 있는 고유키를 참조할 수도 있습니다.

Tip

동일한 부모 테이블의 고유키를 여러 자손 테이블의 외부키가 참조할 수 있습니다.

Tip

한 개의 DELETE 문 또는 UPDATE 문을 실행할 때, 두 개 이상의 RI가 존재하는 경우는 흔하지 않으므로, 보통은 NO ACTION과 RESTRICT 를 동일하게 생각하면 됩니다.

1 외부키 제한 조건에서 UPDATE 규칙과 DELETE 규칙을 정의하는 옵션은 다음과 같습니다.

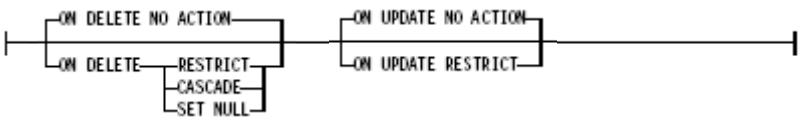


Figure 0813A • • • UPDATE 규칙과 DELETE 규칙 옵션

2 외부키를 가진 자손 테이블에 INSERT 문으로 데이터를 추가할 때, 제공된 외부키가 부모 테이블의 고유키에 존재하는 값인지 점검합니다. 존재하지 않는 값인 경우에는 SQL0530N 오류 코드가 반환되고, INSERT 문은 실패합니다. 자손 테이블에 외부키에 입력된 데이터는 부모 테이블의 고유키에 존재하는 값이므로 항상 참조가 가능합니다. 이러한 기능을 '참조 무결성 (RI, Refrential Integrity)' 라고 합니다.

3 고유키를 가진 부모 테이블에서 UPDATE 문을 실행할 때는 다음과 같이 2 가지의 UPDATE 규칙을 적용받게 할 수 있습니다. CREATE TABLE 문에서 외부키를 정의할 때 ON UPDATE 옵션을 이용하여 지정합니다.

UPDATE 규칙	설 명
NO ACTION	다른 RI 관계의 UPDATE 규칙을 먼저 적용하고, 자신의 UPDATE 규칙을 적용합니다. 변경되는 행의 고유키 값을 참조하고 있는 자손 테이블의 행이 존재하면, SQL0531N 오류 코드가 반환되고, UPDATE 문이 실패합니다. 기본값으로 사용됩니다.
RESTICT	다른 RI 관계의 UPDATE 규칙보다 자신의 UPDATE 규칙을 먼저 적용하는 점을 제외하고, NO ACTION과 동일합니다.

4 고유키를 가진 부모 테이블에서 DELETE 문을 실행할 때는 다음과 같이 4 가지의 DELETE 규칙을 적용받게 할 수 있습니다. CREATE TABLE 문에서 외부키를 정의할 때 ON DELETE 옵션을 이용하여 지정합니다.

DELETE 규칙	설 명
NO ACTION	다른 RI 관계의 DELETE 규칙을 먼저 적용하고, 자신의 DELETE 규칙을 적용합니다. 삭제되는 행의 고유키 값을 참조하고 있는 자손 테이블의 행이 존재하면, SQL0532N 오류 코드가 반환되고, DELETE 문이 실패합니다. 기본값으로 사용됩니다.
RESTICT	다른 RI 관계의 DELETE 규칙보다 자신의 DELETE 규칙을 먼저 적용하는 점을 제외하고, NO ACTION과 동일합니다.
CASCADE	삭제되는 행의 고유키 값을 참조하고 있는 자손 테이블의 행을 함께 삭제합니다.
SET NULL	삭제되는 행의 고유키 값을 참조하고 있는 자손 테이블의 행의 외부키의 값을 NULL 값으로 변경합니다. 자손 테이블의 외부키 컬럼이 NULL 을 허용하는 컬럼일 때 지정할 수 있습니다.

TOPIC 08-14 점검 제한 조건

Point

테이블의 컬럼에 입력되는 값을 제한하는 조건입니다. INSERT와 UPDATE 문을 실행하면 자동으로 점검이 실행되어 조건에 맞지 않는 값인 경우에는 오류를 반환합니다.

Tip

<제한조건명>은 데이터베이스 내에서 고유해야 합니다.

Tip

<제한조건명>을 명시하지 않으면, 엔진이 생성한 이름으로 관리하게 됩니다. ALTER 또는 DROP문으로 관리하려면, 사용자가 제한조건명을 명시하는 것이 편리합니다.

Tip

점검 제한 조건을 표현하는 방법은 SQL 문의 WHERE 조건절의 표현식과 동일합니다.

1 점검 제한 조건을 정의하는 구문은 다음과 같습니다.

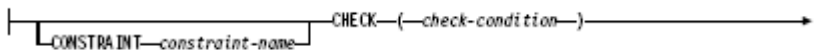


Figure 0814A ••• 점검 제한 조건절

2 create table 문에서 CONSTRAINT ~ CHECK 라는 옵션으로 지정합니다. 점검 제한 조건은 한 테이블에 여러 개 정의할 수 있습니다. CONSTRAINT 옵션을 지정하지 않으면, 제한조건명은 'SQLyymmddhhmmssxxx' 형식으로 엔진이 부여합니다.

```
$ db2 "create  table <스키마명>.<테이블명> (..., <점검 제한 조건절>, ...)"
```

3 alter table 문을 이용하여 점검 제한 조건을 추가할 수 있습니다.

```
$ db2 "alter  table <스키마명>.<테이블명> ADD <점검 제한 조건절>"
```

4 alter table 문을 이용하여 점검 제한 조건을 제거할 수 있습니다.

```
$ db2 "alter  table <스키마명>.<테이블명> DROP CONSTRAINT  <제한조건
    명>"
```

```
CREATE TABLE kes.empl (
        id          smallint          not null
      , name        varchar(30)       not null
      , sex         char(1)
      , mydept      smallint
      , salary      smallint
      , email       varchar(30)       not null
      , hiredate    date
      , CONSTRAINT empl_cc01 CHECK (sex in ('M','F'))
);

ALTER TABLE kes.empl DROP CONSTRAINT empl_cc01 ;

ALTER TABLE kes.empl ADD CONSTRAINT empl_cc01
                     CHECK (sex = 'M' or sex ='F') ;
```

sex 컬럼은 NULL값을 허용할 수 있습니다.

제한조건명은 empl_cc01 입니다.

sex 컬럼에는 'M' 또는 'F' 만 허용됩니다.

표현식은 SQL문의 WHERE 절과 동일합니다.

Figure 0814B ••• 외부키 생성

TOPIC 08-15 Identity 컬럼

Point

INSERT 문이 실행되면 자동으로 그 값이 증가되는 컬럼입니다. 컬럼의 데이터 유형은 숫자 유형으로 한 테이블에 한 개만 정의할 수 있습니다. CREATE TABLE 문에서 GENERATED 라는 옵션으로 지정할 수 있습니다.

1 create table 문에서 GENERATED 라는 옵션으로 지정합니다.

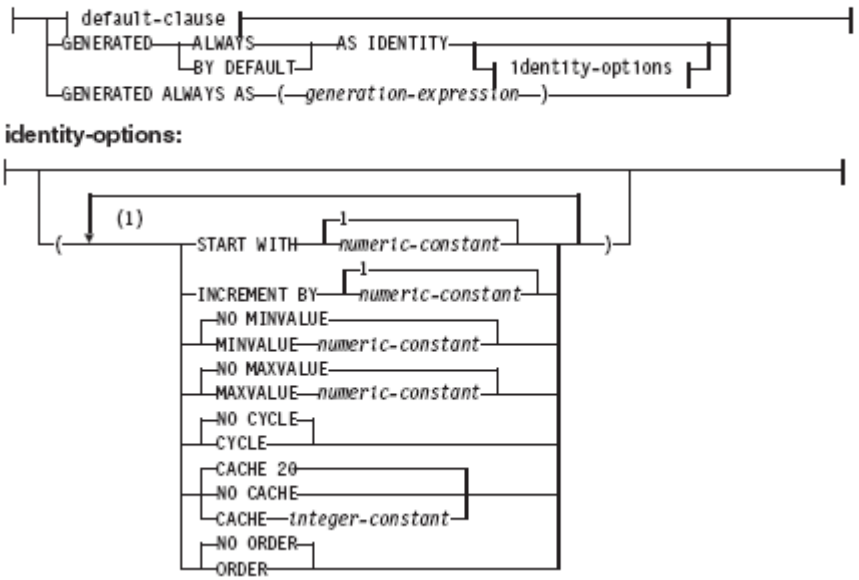


Figure 0815A ••• INDENTITY 컬럼 정의

2 주요한 옵션은 다음과 같습니다.

옵션	설명
GENERATED ALWAYS	엔진에 의해 자동으로 생성된 값만 허용됩니다. 사용자가 명시적으로 값을 지정하여 입력할 수 없습니다.
GENERATED BY DEFAULT	사용자가 값을 지정하지 않는 경우에만 엔진이 자동으로 값을 생성합니다. 값의 고유성을 보장할 수 없습니다.
START WITH	양수 또는 음수의 시작값을 지정합니다.
INCREMENT BY	양수 또는 음수의 증가값을 지정합니다.
MAXVALUE	양수 또는 음수의 최대값을 지정합니다.
CYCLE	최대값 또는 최소값에 도달하면 최소값 또는 최대값을 생성합니다.
CACHE	지정된 개수의 생성값을 미리 캐쉬에 보관하여 성능에 유리합니다.

Tip

데이터베이스가 비활성화되면, 사용되지 않고 캐쉬에 남아있던 자동 생성 값은 유실됩니다.

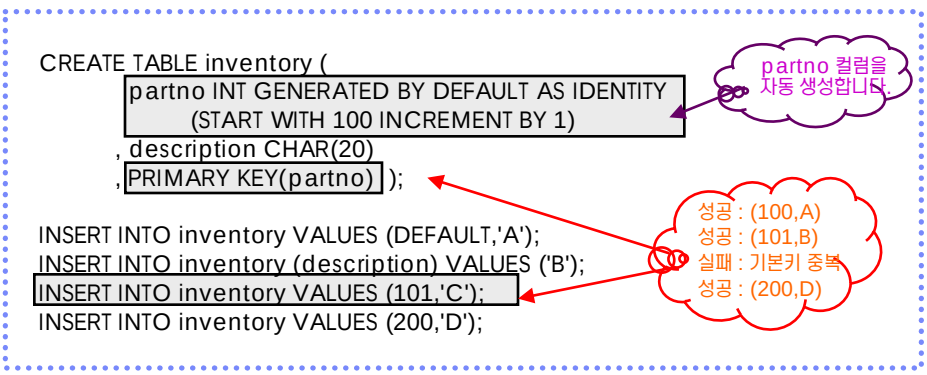


Figure 0815B ••• GENEARTED BY DEFAULT 로 생성된 IDENTITY 컬럼

TOPIC 08 16 NOT LOGGED INITIALLY 옵션

Point

필요시에 ALTER TABLE 문을 이용하여 NOT LOGGED 모드를 활성화시키면, 데이터베이스 로깅 없이 SQL문을 실행할 수 있으므로 대량의 데이터를 입력하는 경우에 유리합니다. UOW가 실패하면, 테이블은 재생성되어야 합니다.

Tip

NOT LOGGED 모드로 테이블에 입력된 데이터는 롤포워드 복구시에 rollforward db 명령어로 복구될 수 없습니다.

1 create table 문에서 NOT LOGGED INITIALLY 라는 옵션으로 지정해야 합니다.

```
$ db2 "create table <테이블명> (<컬럼 정의>) NOT LOOGED INITIALLY"
```

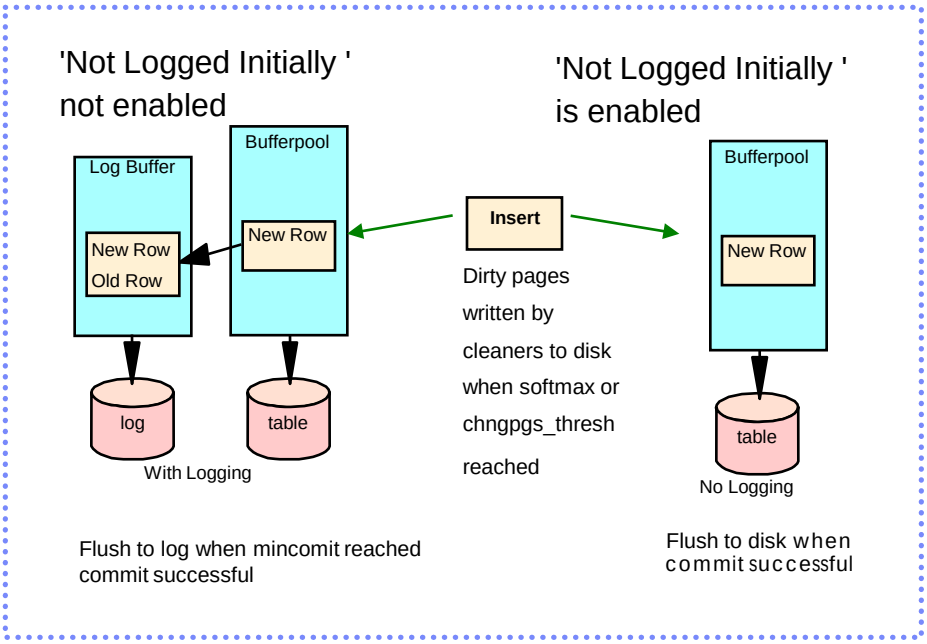


Figure 0816A • • • NOT LOGGED 모드와 LOGGED 모드

Tip

create table 문을 실행하고 commit 하면 'Not Logged Initially' 상태가 해제됩니다.

2 NOT LOGGED INITIALLY 옵션은 기본적으로 비활성화 상태이므로 테이블의 변경 내역은 데이터베이스 로그 파일에 기록됩니다. alter table 문으로 NOT LOGGED INITIALLY 옵션을 활성화시키면, COMMIT 또는 ROLLBACK 문이 실행될 때까지 해당 테이블에 대한 데이터베이스 로깅이 중지됩니다.

Tip

ACTIVATE NOT LOGGED INITIALLY WITH EMPTY TABLE 옵션을 이용하면 테이블의 기존 데이터를 로깅하지 않고 삭제합니다.

```
$ vi <입력파일명>
ALTER TABLE <테이블명> ACTIVATE NOT LOGGED INITIALLY;
INSERT INTO ... SELECT FROM ... ;
COMMIT;
$ db2 +c –svtf <입력파일명>
```

3 NOT LOGGED 옵션은 UOW가 성공적으로 종료되면 자동으로 비활성화되고, 테이블에 대한 데이터베이스 로깅은 다시 시작됩니다.

4 UOW가 실패하면 해당 테이블은 재생성해야 합니다.

```
$ db2 drop table <테이블명>
$ db2look –d <데이터베이스명> -z <스키마명> -t <테이블명> -e –o <출력파일명>
db2 –svtf <출력파일명>
```

TOPIC 0817 뷰

Point

사용자가 원본 테이블의 특정 행과 특정 컬럼들만 액세스할 수 있도록 하려면 뷰를 생성합니다. 한 개 이상의 원본 테이블의 데이터를 조인하여 뷰를 생성하는 것도 가능합니다. 기본 테이블에 대해 최소한 SELECT 특권이 있어야 합니다..

Tip

원본 테이블이 제거되면 뷰는 작동 불능 상태가 되어서 액세스를 할 수 없습니다. 원본 테이블이 다시 생성되어도 작동 불능 상태의 뷰는 액세스할 수 없으므로, 재생성이 필요합니다.

1. create view 문으로 생성하며, SELECT문으로 액세스가 허용되는 데이터를 제한합니다.

```
$ db2 "create view kes.t1_v1 as select * from kes.t1 where c2 > 200 "
```

2. WITH CHECK OPTION을 이용하면, 뷰의 정의에 맞지 않는 데이터를 추가, 삭제, 변경할 수 없습니다. INSERT 또는 UPDATE를 실행할 때 뷰의 정의에 맞는지를 확인합니다.

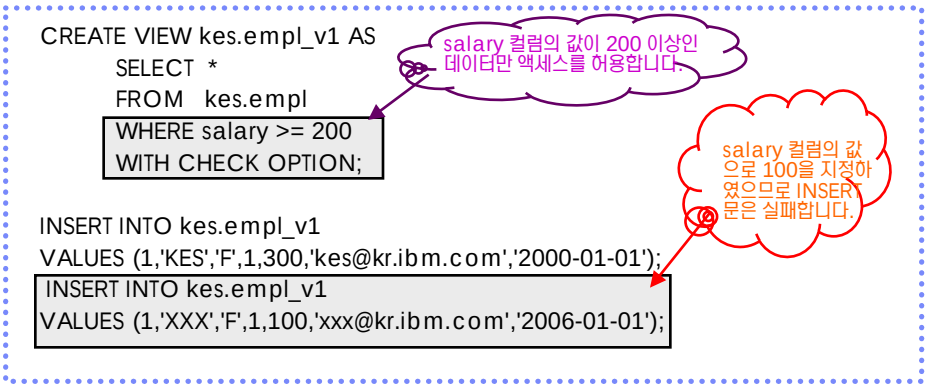


Figure 0817A ••• WITH CHECK OPTION 옵션이 있는 뷰

3. VIEW를 통한 액세스가 가능합니다.

```
$ db2 "insert into kes.t1_v1 values (1,300)"
$ db2 "select * from kes.t1_v1"
```

4. drop view 문을 이용하여 제거합니다.

```
$ db2 drop view kes.t1_v1
```

5. list tables 명령어에서 뷰의 목록을 확인할 수 있습니다.

```
$ db2 list tables
$ db2 list tables for schema <스키마명>
$ db2 list tables for system
$ db2 list tables for all
```

6. 뷰에 대한 정보는 SYSCAT.VIEWS, SYSCAT.VIEWDEP, SYSCAT.TABLES 뷰를 이용해서 확인합니다. SYSCAT.TABLES 뷰에서 TYPE 컬럼의 값이 'V' 입니다.

```
$ db2 "select * from syscat.tables"
```

7. db2look 명령어를 이용하여 뷰에 대한 DDL을 추출합니다.

```
$ db2look –d <DB명>  –e –z <스키마명> –v <뷰명> -o <출력파일명>
```

TOPIC 08-18 CREATE VIEW 문

Point

뷰를 생성하는 SQL문입니다. SELECT 문을 이용하여 뷰를 통하여 액세스할 수 있는 데이터의 범위를 결정하며, WITH CHECK OPTION 옵션으로 뷰의 정의에 맞지 않는 데이터에 대한 액세스를 허용하지 않게 합니다.

Tip

SYSADM, DBADM 권한 또는 테이블에 대한 SEELCT 특권이 필요할 수 있습니다.

Tip

새로운 스키마명을 이용하여 테이블명을 지정하려면, 데이터베이스에 대한 IMPLICIT_SCHEMA 특권이 필요합니다.

Tip

기존의 스키마명을 이용하여 테이블명을 지정하려면, 스키마에 대한 CREATIN 특권이 필요합니다.

1 create view 문의 형식은 다음과 같습니다.

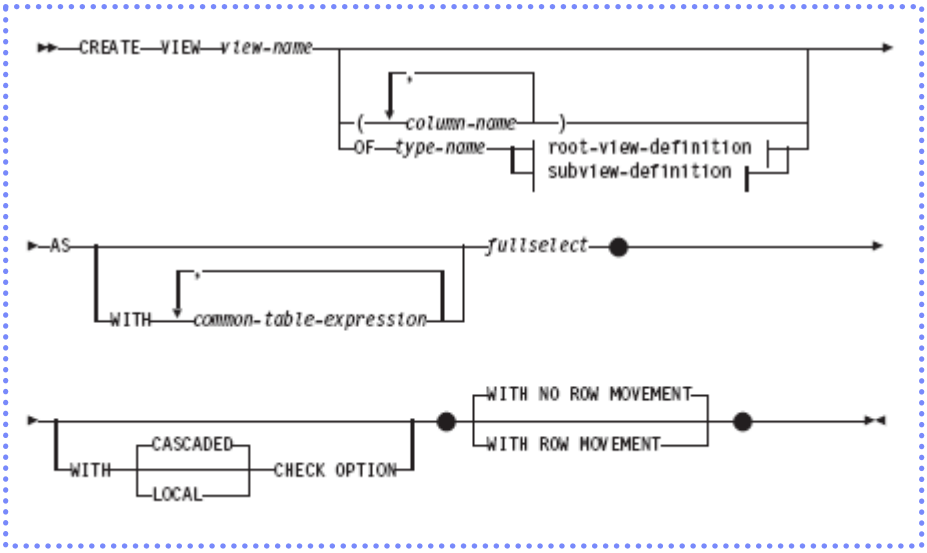


Figure 0818A CREATE VIEW 문

2 옵션에 대한 설명은 다음과 같습니다.

모 드	설 명
<뷰명>	임의의 고유한 이름으로 지정합니다.
<컬럼명>	지정하지 않으면 베이스 테이블의 컬럼명 또는 SELECT문의 결과 컬럼명이 사용됩니다.
WITH <공통 테이블 표현식>	SELECT문에서 사용될 공통 테이블을 정의합니다.
AS <select문>	뷰의 내용이 되는 SELECT문을 지정합니다.
WITH CASCADE CHECK OPTION	뷰의 정의에 맞지 않는 데이터를 처리하지 않습니다. 해당 뷰를 이용한 다른 뷰를 생성했를 때, 이 특성을 전달합니다.
WITH LOCAL CHECK OPTION	뷰의 정의에 맞지 않는 데이터를 처리하지 않습니다. 해당 뷰를 이용한 다른 뷰를 생성했를 때, 이 특성을 전달하지 않습니다.

TOPIC 08-19 MQT

Point

집계 함수를 이용한 summary table 을 미리 정의해 두면, 집계 함수를 이용한 복잡한 쿼리의 실행 시간을 단축시킬 수 있습니다. CREATE TABLE 문으로 생성하고, REFRESH TABLE 문으로 관리합니다.

Tip

SUMMARY 테이블이라고도 합니다.

Tip

immediate 옵션을 사용해도 최초에 한 번은 refresh table 명령어를 실행해야 합니다.

Tip

refresh deferred 옵션을 사용하면, refresh table 문을 실행하기 전까지는 완벽한 데이터 일관성을 보장하지 않습니다.

1 Meterialized Query Table의 약자입니다. create table 문에서 REFRESH 옵션으로 생성합니다.

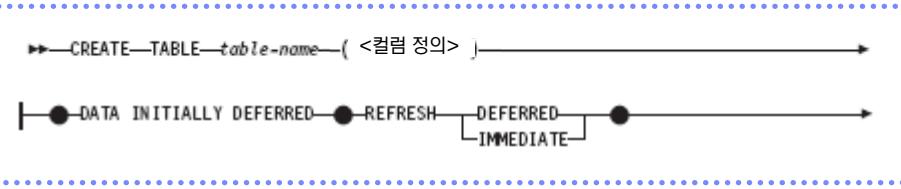


Figure 0819A CREATE TABLE 문의 MQT 생성 옵션

2 사용하기 전에 REFRESH TABLE 문을 이용하여 최초의 결과 집합을 생성해야 합니다.

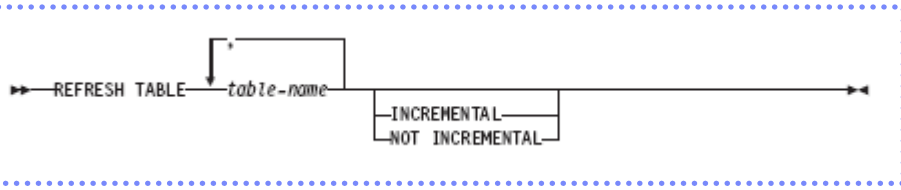


Figure 0819B REFRESH TABLE 문

3 SELECT문에서 직접 MQT를 이용할 수 있습니다.

```
$ db2 "select * from <MQT명>"
```

4 베이스 테이블을 대상으로 하는 SELECT문에서 명시한 조건문이 MQT의 정의 부분과 일치하면, MQT를 이용한 SELECT문으로 자동으로 변환됩니다.

```
CREATE TABLE staff_summary AS
  (SELECT   dept
           ,COUNT(*) AS count_rows
           ,SUM(id)  AS sum_id
   FROM     staff
   GROUP BY dept)
DATA INITIALLY DEFERRED REFRESH IMMEDIATE;

REFRESH TABLE staff_summary;

ORIGINAL QUERY
==============
SELECT   dept
        ,AVG(id)
FROM     staff
GROUP BY dept

OPTIMIZED QUERY
===============================
SELECT  Q1.dept AS "dept"
       ,Q1.sum_id / Q1.count_rows
FROM    staff_summary AS Q1
```

SELECT문이 자동으로 MQT를 사용할 수 있도록 변환됩니다.

AVG는 SUM/COUNT 로 변형이 가능합니다.

Figure 0819C MQT를 이용하도록 변환되는 쿼리

5 drop table 문을 이용하여 제거합니다.

```
$ db2 drop table <MQT명>
```

TOPIC 08 20 인덱스

Point

효율적인 데이터 액세스를 위해서 한 테이블에 한 개 이상의 인덱스를 생성할 수 있습니다. CREATE INDEX 문과 DROP INDEX 문으로 관리합니다.

Tip

고유 인덱스는 NULL 값을 허용하며, NULL 값을 가진 행은 한 개만 허용됩니다.

Tip

테이블에 기본키 또는 고유키를 정의하면 해당 컬럼에 대한 자동으로 UNIQUE 인덱스가 생성됩니다.

Tip

CLUSTER 옵션을 가진 인덱스는 한 테이블에 한 개만 가능하므로, 가장 중요한 인덱스를 CLUSTER 인덱스로 정의합니다.

Tip

INCLUDE 옵션은 UNIQUE 인덱스에서만 사용 가능합니다.

1. create index 문으로 컬럼명과 컬럼별 정렬 순서를 지정합니다. 기본적으로 인덱스는 중복된 값을 허용하므로 중복된 행을 허용하지 않는 인덱스를 생성하려면 UNIQUE 옵션을 이용합니다.

```
$ db2 "create index <스키마명>.<인덱스명> on <테이블명> (<컬럼명>)"
$ db2 "create UNIQUE index <인덱스명> on <테이블명> (<컬럼명>)"
```

2. CLUSTER 옵션을 이용하면, 해당 인덱스의 정렬 순서를 기준으로 테이블의 데이터가 물리적으로 배치되므로 효율적인 액세스가 가능합니다.

```
$ db2 "create unique index <인덱스명> on <테이블명>(<컬럼명>) CLUSTER"
```

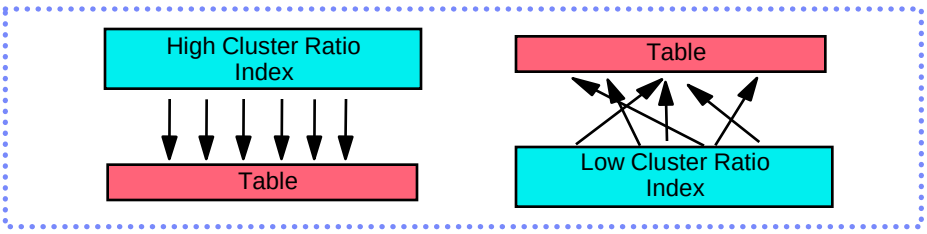


Figure 0820A • • • 인덱스의 클러스터 비율

3. INCLUDE 옵션으로 추가된 컬럼들은 인덱스의 데이터 페이지에 RID 와 함께 저장되어, 인덱스 전용 액세스를 가능하게 합니다. 반드시 UNIQUE 옵션을 함께 지정해야 합니다.

```
$ db2 "create UNIQUE  index <인덱스명> on <테이블명> (<컬럼명 1>)
INCLUDE (<컬럼명 2>, <컬럼명 3>, ...)"
```

4. ALLOW REVERSE SCANS 옵션으로 생성된 인덱스는 양방향 액세스를 허용합니다.

```
$ db2 "create unique index <인덱스명> on <테이블명> (<컬럼명>) cluster
    ALLOW REVERS SCANS"
```

5. drop index 문으로 제거하며, 테이블이 제거되면 자동으로 제거됩니다.

```
$ db2 "drop index <스키마명>.<인덱스명>"
```

6. 인덱스에 대한 정보는 SYSCAT.INDEXES 뷰 또는 describe indexes 명령어를 이용하여 확인합니다.

```
$ db2 "select * from syscat.indexes"
$ db2 describe indexes for table <스키마명>.<테이블명> show detail
```

7. db2look 명령어를 이용하여 테이블과 함께 인덱스에 대한 DDL을 추출합니다.

```
$ db2look –d <DB명> –e –z <스키마명> –v <테이블명> -o <출력파일명>
```

TOPIC 08 21 CREATE INDEX 문

Point

테이블에 인덱스를 생성하는 SQL문입니다. UNIQUE, CLUSTER, INCLUDE, ALLOW REVERSE SCANS 등의 옵션을 가진 인덱스를 생성할 수 있습니다.

Tip

SYSADM, DBADM 권한 또는 테이블에 대한 CONTROL, INDEX 특권이 필요할 수 있습니다.

Tip

새로운 스키마명을 이용하여 인덱스명을 지정하려면, 데이터베이스에 대한 IMPLICIT_SCHEMA 특권이 필요합니다.

Tip

기존의 스키마명을 이용하여 인덱스명을 지정하려면, 스키마에 대한 CREATIN 특권이 필요합니다.

Tip

INCLUDE 옵션에서 지정한 컬럼은 인덱스를 구성하는 컬럼이 아니므로, 조건식에서 이 컬럼을 이용하여 검색을 요청할 때, index scan 이 선택되지 않습니다.

1 create index 문의 형식은 다음과 같습니다.

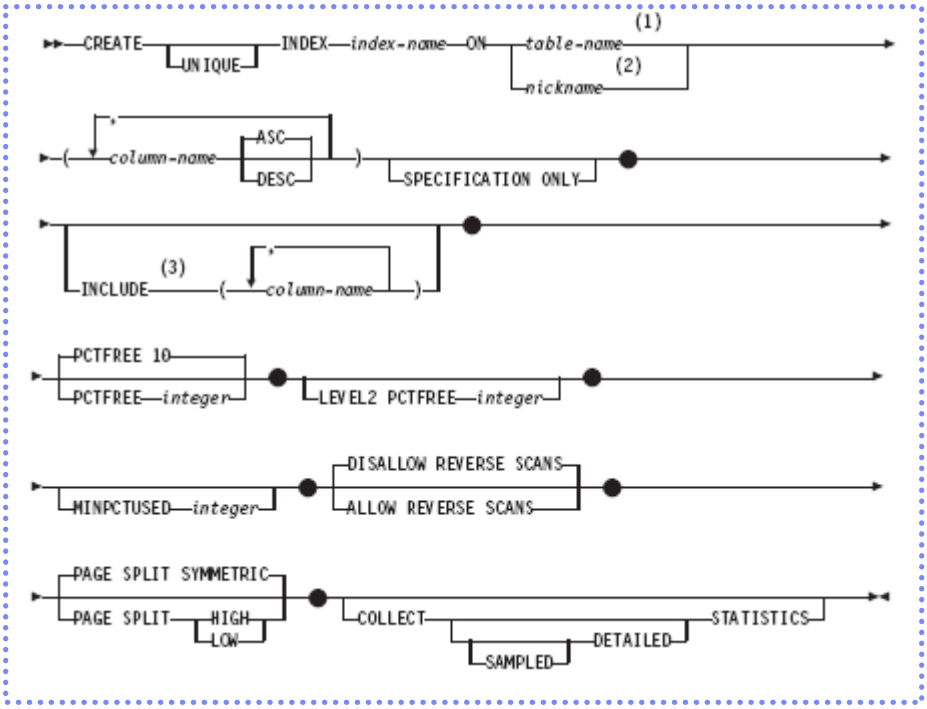


Figure 0821A CREATE INDEX 문

2 옵션에 대한 설명은 다음과 같습니다.

모 드	설 명
UNIQUE	고유 인덱스로 생성합니다.
<인덱스명>	임의의 고유한 이름으로 지정합니다.
ON <테이블명>	인덱스가 생성될 테이블명을 지정합니다.
<컬럼명>	인덱스를 구성하는 컬럼명을 지정합니다. 한 개 이상인 경우에는 ,(컴마)로 구분합니다.
ASC / DESC	컬럼별 정렬 순서를 오름차순(ASC) 또는 내림차순(DESC)로 지정합니다. 기본값은 ASC 입니다.
INCLUDE (<컬럼명>)	인덱스의 리프 페이지에 RID와 함께 저장되는 컬럼을 지정합니다. 한 개 이상인 경우에는 ,(컴마)로 구분합니다.
CLUSTER	클러스터 인덱스로 지정합니다.
PCTFREE	인덱스의 각 페이지의 여유 공간 비율로 백분율로 지정합니다. 기본값은 10 입니다.
MINPCTUSED	인덱스의 리프 페이지의 최소 사용 공간 비율로 백분율로 지정합니다. 50 이하로 지정하는 것이 성능에 유리합니다.
ALLOW REVERSE SCANS	역방향 액세스를 허용합니다.

TOPIC 08 22 시퀀스

Point

데이터베이스 지원에서 제공되는 자동 생성 일련 번호를 시퀀스라고 합니다. CREATE SEQUENCE 문, ALTER SEQUENCE 문, DROP SEQUENCE 문으로 관리합니다.

Tip

데이블의 identity 컬럼과 기능상 동일합니다. identity 컬럼은 해당 테이블에서만 유효하며, 시퀀스는 데이터베이스 수준에서 운영됩니다.

1 create sequence 문의 형식은 다음과 같습니다.

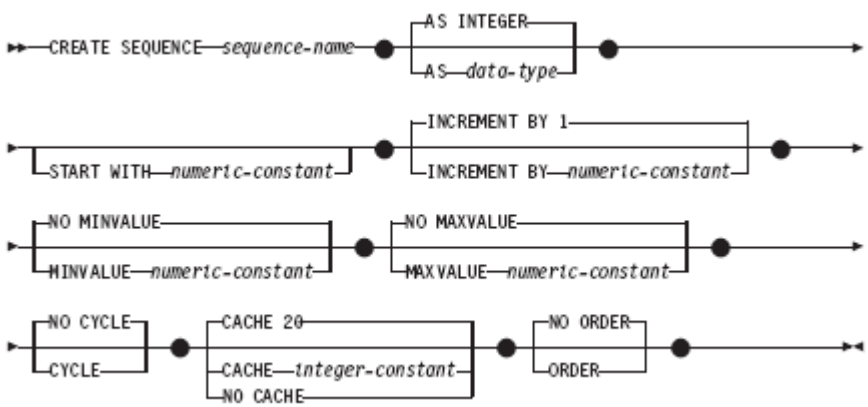


Figure 0822A ••• CREATE SEQUENCE 문

Tip

데이터베이스가 비활성화되면, 사용되지 않고 캐쉬에 남아있던 자동 생성 값은 유실됩니다.

2 주요한 옵션은 다음과 같습니다.

옵션	설명
<시퀀스명>	임의의 고유한 이름으로 지정합니다.
AS <데이터유형>	SAMLLINT, INT, BIGINT, DECIMAL 중에서 원하는 데이터 유형을 선택합니다.
START WITH	양수 또는 음수의 시작값을 지정합니다.
INCREMENT BY	양수 또는 음수의 증가값을 지정합니다.
MINVALUE	양수 또는 음수의 최소값을 지정합니다.
MAXVALUE	양수 또는 음수의 최대값을 지정합니다.
CYCLE	최대값 또는 최소값에 도달하면 최소값 또는 최대값을 생성합니다.
CACHE	지정된 개수의 생성값을 미리 캐쉬에 보관하여 성능에 유리합니다.

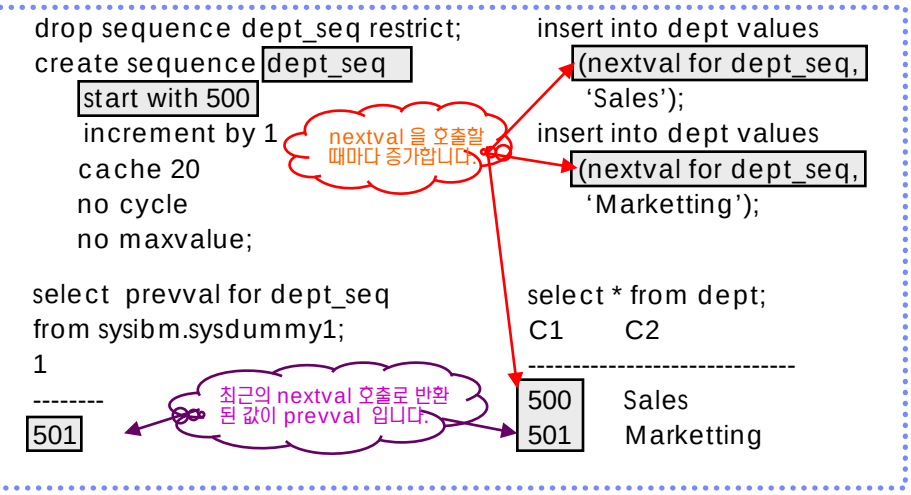


Figure 0822B ••• PREVVAL 과 NEXTVAL

TOPIC 08 23 트리거

Point

특정 테이블에 INSERT, UPDATE, DELETE 문이 실행될 때 자동으로 실행되는 일련의 작업들을 정의합니다. 트리거의 이벤트 유형은 BEFORE, AFTER, INSTEAD OF가 있으며, CREATE TRIGGER 문과 DROP TRIGGER 문으로 관리합니다.

Tip

트리거가 구현하는 비즈니스 규칙은 데이터베이스 오브젝트에 대한 일련의 변경 작업 또는 예외 처리 로직입니다.

Tip

트리거는 특정 조건을 만족하는 경우에만 SQL문의 요청을 허용하기 위한 용도로 사용되기도 합니다.

1 한 데이터베이스의 특정 테이블에 대한 변경 작업이 다른 테이블에 영향을 미칠 수 있습니다. 응용프로그램의 로직에서 이러한 비즈니스 규칙을 구현하면, 비즈니스 규칙이 변경될 때마다 응용프로그램의 로직을 수정해야 합니다. 트리거는 특정 테이블 또는 뷰에 대한 변경 작업이 요청될 때마다 자동으로 실행되어야 하는 일련의 작업들을 데이터베이스 수준에서 정의합니다. 비즈니스 규칙이 변경되되어도 데이터베이스에 존재하는 트리거의 정의만 변경하면 되므로, 모든 응용프로그램은 추가적인 로직의 변경 없이 새로운 비즈니스 규칙을 적용할 수 있습니다.

Figure 0823A ••• 비즈니스 규칙

2 트리거에 정의된 비즈니스 로직을 실행하는 시점에 의해 3가지 유형으로 분류됩니다.

유 형	설 명
AFTER 트리거	특정 테이블에 대해 요청된 INSERT, UPDATE, DELETE 문을 먼저 실행하고, 정의된 일련의 작업들을 실행합니다.
BEFORE 트리거	정의된 일련의 작업들을 먼저 실행하고, 특정 테이블에 대해 요청된 INSERT, UPDATE, DELETE 문을 실행합니다.

3 트리거는 발생시키는 SQL문의 유형에 의해 3가지 유형으로 분류됩니다.

유 형	설 명
INSERT 트리거	특정 테이블에 대해 INSERT문이 요청된 경우에 실행됩니다.
UPDATE 트리거	특정 테이블에 대해 UPDATE문이 요청된 경우에 실행됩니다. 특정 컬럼을 지정할 수도 있습니다.
DELETE 트리거	특정 테이블에 대해 DELETE문이 요청된 경우에 실행됩니다.

4 트리거의 기준이 되는 대상은 2 가지로 분류됩니다.

유 형	설 명
테이블 트리거	특정 테이블에 INSERT, UPDATE, DELETE 문이 요청될 때 요청된 SQL문과 정의된 일련의 작업들을 실행합니다.
뷰 트리거 INSTEAD OF 트리거	특정 뷰에 INSERT, UPDATE, DELETE 문을 요청하면, 요청된 SQL문 대신에 정의된 일련의 작업들을 실행합니다.

TOPIC 08-24 CREATE TRIGGER 문

Point

트리거를 생성하는 SQL문입니다. 단일 SQL문 또는 SQL/PL을 이용하여 트리거가 실행할 로직을 구현합니다. INSERT, UPDATE, DELETE 트리거는 개별적으로 생성하고, 트리거의 정의에 변경이 있을 때는 재 생성합니다.

Tip
SYSADM, DBADM 권한 또는 테이블에 대한 CONTROL 특권이 필요할 수 있습니다

Tip
IMPLICIT_SCHEMA 데이터베이스 특권, 스키마에 대한 ALTERIN, CREATEIN 특권이 필요합니다.

Tip
BEFORE, AFTER 트리거를 정의할 때는 테이블에 대한 ALTER 특권이 필요합니다.

Tip
INSTEAD OF 트리거를 정의할 때는 뷰에 대한 CONTROL 특권이 필요합니다.

Tip
TRANSITION 변수 또는 테이블을 사용할 때는 테이블에 대한 SELECT 특권이 필요합니다.

Tip
transition 행 또는 테이블은 트리거의 비즈니스 로직에서 이용할 수 있습니다. DELETE, UPDATE 트리거에서 사용됩니다.

Tip
UPDATE 옵션은 모든 컬럼의 변경에 적용됩니다. OF 옵션은 지정한 컬럼의 UPDATE 시에만 적용됩니다.

Tip
트리거 본문에 정의되는 비즈니스 로직은 기본적으로 단일 SQL문입니다. 여러 개의 SQL문과 논리적인 로직을 구현하려면 BEGIN~END 블록을 이용하여SQL/PL을 이용합니다.

1 create trigger 문의 형식은 다음과 같습니다.

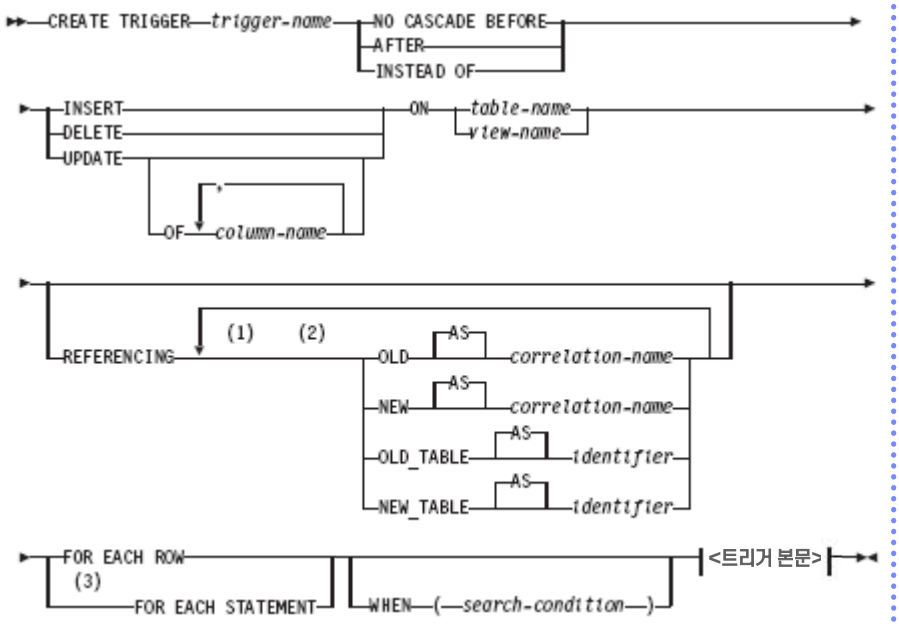


Figure 0824A • • • CREATE TRIGGER 문

2 옵션에 대한 설명은 다음과 같습니다.

모 드	설 명
<트리거명>	임의의 고유한 이름으로 지정합니다.
NO CASCADE BEFORE	BEFORE 트리거를 생성합니다.
AFTER	AFTER 트리거를 생성합니다.
INSTEAD OF	INSTEAD OF 트리거를 생성합니다.
INSERT	INSERT 트리거를 생성합니다.
DELETE	DELETE 트리거를 생성합니다.
UPDATE OF <컬럼명>	UPDATE 트리거를 생성합니다.
REFERENCING	transition 테이블 또는 행의 이름을 지정합니다.
OLD AS <id> OLD_TABLE AS <id>	요청된 SQL문이 실행되기 전의 행 또는 테이블의 값을 보관하고 있는 transition 행 또는 테이블 id 입니다.
NEW AS <id> NEW_TABLE AS <id>	요청된 SQL문이 실행된 후의 행 또는 테이블의 값을 보관하고 있는 transition 행 또는 테이블 id 입니다.
FOR EACH ROW	조건에 맞는 모든 행에 개별적으로 트리거가 적용됩니다.
FOR EACH STATEMENT	조건에 맞는 행의 개수에 상관없이 한 번만 실행됩니다.
WHEN <조건식>	<조건식>을 만족하는 경우에만 본문의 로직을 실행합니다.
<트리거 본문>	SQL/PL 을 이용하여 로직을 작성합니다.

TOPIC 08 25 BEFORE 트리거

Point

특정 테이블에 대한 INSERT, UPDATE, DELETE 문을 실행하기 전에 먼저 정의된 일련의 작업들을 실행합니다. CREATE TRIGGER 문에서 NO CASCADE BEFORE 옵션을 사용하여 생성합니다.

Tip

INSERT, UPDATE, DELETE 문에 대한 트리거를 한 개 이상 생성할 수 있습니다.

1 create trigger 문에서 NO CASCADE BEFORE 옵션으로 생성합니다.

```
CREATE TABLE kes.class (
        number      smallint    not null
      , name        varchar(20)
      , length      smallint
);

CREATE TABLE kes.test (
        id          int   not null
      , number      smallint
      , test_date   date
      , start_time  time
      , seat        smallint
      , score       smallint
);

INSERT INTO kes.class VALUES (1,'DB2',60);

CREATE TRIGGER pre9
NO CASCADE BEFORE INSERT ON kes.test
REFERENCING NEW AS N
FOR EACH ROW
MODE DB2SQL
WHEN (N.START_TIME < '09:00:00')
      SIGNAL SQLSTATE '70003'
      ('09:00 이전에는 좌석을 예약할 수 없습니다!');

INSERT INTO kes.test VALUES (1,1,'2006-04-19','08:30:00',2,NULL);

CREATE TRIGGER aft5
NO CASCADE BEFORE INSERT ON kes.test
REFERENCING NEW AS N
FOR EACH ROW
MODE DB2SQL
WHEN (N.START_TIME +
      (SELECT SMALLINT(LENGTH)
       FROM test
       WHERE NUMBER = N.NUMBER) MINUTES > '17:00:00')
       SIGNAL SQLSTATE '70004'
       ('16:00 이후에는 좌석을 예약할 수 없습니다!');

INSERT INTO kes.test VALUES (3,1,'2006-04-19','16:30:00',1,NULL);
INSERT INTO kes.test VALUES (2,1,'2006-04-19','10:00:00',1,NULL);

SELECT    id, number, start_time
FROM      kes.test;
```

실패 : SQL0438N 응용프로그램이 진단 텍스트 "09:00 이전에는 좌석을 예약할 수 없습니다!"과(와)함께 오류를 표시했습니다. SQLSTATE=70003

실패 : SQL0438N 응용프로그램이 진단 텍스트 "16:00 이후에는 좌석을 예약할 수 없습니다!"과(와) 함께 오류를 표시했습니다. SQLSTATE=70004

```
ID          NUMBER START_TIME
----------- ------ ----------
          2      1 10:00:00
```

Figure 0825A ··· BEFORE 트리거

TOPIC 08-26 AFTER 트리거

Point

특정 테이블에 대해 요청된 INSERT, UPDATE, DELETE 문을 먼저 실행하고, 정의된 비즈니스 로직을 실행합니다. 최대 16 레벨까지 다른 트리거를 연쇄적으로 수행할 수 있습니다. CREATE TRIGGER 문에서 AFTER 옵션을 사용하여 생성합니다.

Tip

INSERT, UPDATE, DELETE 문에 대한 트리거를 한 개 이상 생성할 수 있습니다.

1 AFTER 트리거를 생성하는 예는 다음과 같습니다.

```
CREATE TABLE kes.empl (
       id          smallint    not null
     , name        varchar(30)  not null
     , sex         char(1)
     , mydept      smallint
     , salary      smallint
     , email       varchar(30)  not null
     , hiredate    date
);

INSERT INTO kes.empl VALUES
       (1,'KES','F',1,100,'kes@kr.ibm.com','1993-01-30')
     , (2,'KHY','F',3,250,'khy@kr.ibm.com','1992-03-17')
     , (3,'JHS','F',2,300,'jhs@kr.ibm.com','1997-02-03')
     , (4,'JJY','M',2,280,'jjy@kr.ibm.com','1998-07-22');

CREATE TABLE kes.log (
       id          smallint
     , osalary     smallint
     , nsalary     smallint
     , ts          timestamp);

CREATE TRIGGER kes.empl_trig01
    AFTER UPDATE OF salary ON kes.empl
    REFERENCING OLD AS o NEW AS n
    FOR EACH ROW
    MODE DB2SQL
        WHEN ( n.salary > o.salary * 1.2 )
            INSERT INTO kes.log
            VALUES(o.id,o.salary,n.salary,current timestamp );

UPDATE  kes.empl
SET     salary = salary + 30
WHERE   hiredate < '2000-01-01' ;

SELECT     id, salary, hiredate
FROM       kes.empl
ORDER BY id ;

SELECT  id, osalary, nsalary
FROM    kes.log;
```

ID SALARY HIREDATE
1 100 1993-01-30
2 250 1992-03-17
3 300 1997-02-03
4 280 1998-07-22

변경된 salary 값이 120% 이상인 경우에만 kes.log에 추가합니다.

ID SALARY HIREDATE
1 130 1993-01-30
2 280 1992-03-17
3 330 1997-02-03
4 310 1998-07-22

ID OSALARY NSALARY
1 100 130

Figure 0826A ••• AFTER 트리거

TOPIC 08-27 INSTEAD OF 트리거

Point

뷰를 대상으로 정의된 트리거로 테이블에 대한 변경 SQL문을 뷰를 통해서 관리하거나, 뷰에 직접 허용되지 않는 변경 SQL문을 트리거 로직으로 대신 실행할 때도 사용됩니다. CREATE TRIGGER 문에서 INSTEAD OF 옵션을 사용하여 생성합니다.

Tip

특정 뷰에 대한 INSERT, UPDATE, DELETE 문을 처리하는 트리거는 한 개씩만 생성할 수 있습니다.

Tip

FOR EACH STATEMENT 옵션은 사용할 수 없습니다.

1 create trigger 문에서 INSTEAD OF 옵션으로 생성합니다.

```
CREATE TABLE kes.empl (
        id              smallint    not null
      , name            varchar(30)  not null
      , sex             char(1)
      , mydept          smallint
      , salary          smallint
      , email           varchar(30)  not null
      , hiredate        date
);

CREATE VIEW kes.empl_v AS
SELECT          id, name, sex
FROM            kes.empl;

CREATE TRIGGER kes.empl_v_insert
    INSTEAD OF INSERT ON kes.empl_v
    REFERENCING NEW AS N
    FOR EACH ROW
    MODE DB2SQL
        INSERT INTO kes.empl VALUES (
                n.id
              , n.name
              , CASE RTRIM(LTRIM(UPPER(n.sex)))
                    WHEN  'M' THEN 'M'
                    WHEN  'F' THEN 'F'
                    ELSE NULL
                END
              , NULL
              , NULL
              , LOWER(n.name)||'@kr.ibm.com'
              , CURRENT DATE – 2 days);

INSERT INTO kes.empl_v VALUES (1,'KES','');
INSERT INTO kes.empl_v VALUES (2,'KHY','f');
INSERT INTO kes.empl_v VALUES (3,'JHS','F');
INSERT INTO kes.empl_v VALUES (4,'JJY','m');
INSERT INTO kes.empl_v VALUES (5,'XXX','X');

SELECT    id
        , substr(name,1,3) name
        , sex
        . email
FROM    kes.empl_v
ORDER BY id;
```

sex 컬럼의 값이 'M', 'm', 'F', 'f' 인 경우에 대문자로 변환하여 입력합니다. '' 값인 경우에는 NULL 값으로 입력합니다.

email 컬럼과 hiredate 컬럼에는 name컬럼의 값과 시스템 날짜를 이용한 값이 입력됩니다.

```
ID     NAME  SEX EMAIL
------ ----- --- ---------------
     1 KES   -   kes@kr.ibm.com
     2 KHY   F   khy@kr.ibm.com
     3 JHS   F   jhs@kr.ibm.com
     4 JJY   M   jjy@kr.ibm.com
     5 XXX   -   xxx@kr.ibm.com
```

Figure 0827A ··· INSTEAD OF 트리거

TOPIC 08-28 사용자 정의 데이터 유형

Point

사용자가 추가적으로 정의할 수 있는 데이터 유형입니다. 일반적으로 도량형과 관련된 데이터를 저장할 때 이용합니다. CREATE DISTINCT TYPE 문과 DROP DISTINCT TYPE 문으로 관리합니다.

1 단위가 서로 다른 의미를 가지는 두 개의 값을 단순히 값으로만 비교하는 것은 잘못된 결과를 만들게 됩니다.

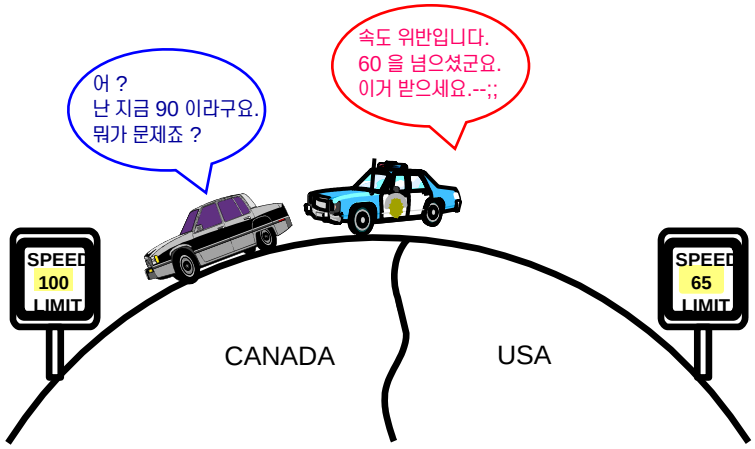


Figure 0828A • • • UDT 의 필요성

Tip

with comparisos 옵션은 새로 생성한 UDT 형으로 형 변환을 할 수 있는 형변환(CAST) 함수를 기본적으로 제공합니다.

2 단위가 다른 두 값을 직접 비교하는 것을 방지하기 위해 사용자가 create distinct type 문으로 새로운 데이터 유형을 생성할 수 있습니다. 유형이 다른 두 데이터는 서로 직접 비교될 수 없습니다.

```
CREATE DISTINCT TYPE dollar AS INTEGER WITH COMPARISONS ;
CREATE DISTINCT TYPE won AS INTEGER WITH COMPARISONS ;

CREATE TABLE kes.person VALUES
  ( f_name        VARCHAR (30)
  , money_d       dollar NOT NULL
  , money_w       won NOT NULL ) ;

INSERT INTO kes.person
VALUES (KES', 80, 70),('JHS',80,100);

SELECT f_name FROM  kes.person
WHERE money_d < money_w;

SELECT f_name FROM  kes.person
WHERE money_d < dollar(money_w)
OR      won(money_d) < money_w;
```

money_d 와 money_w 컬럼을 다른 유형이므로 직접 비교가 불가능합니다.

money_w 컬럼을 dollar 유형으로 형변환하였으므로 비교가 가능합니다.

money_d 컬럼을 won 유형으로 형변환하였으므로 비교가 가능합니다.

Figure 0828B • • • 다른 데이터 유형간의 비교 실패

TOPIC 08 29 사용자 정의 함수

Point

사용자가 직접 구현하여 엔진에 추가적으로 정의한 함수입니다. 사용자 정의 함수는 기존의 내장 SQL 함수와 동일한 방법으로 SQL문에서 사용됩니다. CREATE FUNCTION문과 DROP FUNCTION 문으로 관리합니다.

1 DB2에서 사용하는 함수는 생성 주체에 따라 2 가지 유형으로 구분됩니다.

유 형	설 명
빌트인	엔진에 의해 기본적으로 제공되는 함수입니다. 내장 함수라고 합니다.
사용자 정의	사용자가 새롭게 작성하여 엔진에 추가한 함수입니다.

2 적용되는 대상과 반환하는 값의 유형에 따라 함수는 4 가지 유형으로 구분됩니다.

유 형	설 명
스칼라	조건에 맞는 모든 행에 대해서 각각 함수를 실행하여 결과를 반환합니다. 문자 함수, 수학 함수, 날짜 함수등이 있습니다.
컬럼	조건에 맞는 행들을 지정된 값에 의해 그룹으로 분류하고, 각 그룹별로 함수를 적용시킨 결과값을 반환합니다. 집계 함수가 해당됩니다.
행	조건에 맞는 결과 행을 Row 형태로 반환하는 함수입니다.
테이블	조건에 맞는 결과 집합을 table 구조로 반환합니다. SELECT 문의 FROM 절에서 사용되며, 일반 테이블과 동일한 방법으로 액세스할 수 있습니다. 스냅샷 함수가 해당됩니다.

3 사용자 정의 함수의 유형은 작성하는 언어와 반환하는 결과의 유형에 따라 7 가지로 분류됩니다.

유 형	설 명
외부 스칼라	C, Java 등의 프로그래밍 언어로 작성된 라이브러리를 이용하며, 스칼라 값을 반환합니다.
외부 테이블	C, Java 등의 프로그래밍 언어로 작성된 라이브러리를 이용하며, 테이블을 반환합니다.
OLE DB 외부 테이블	OLE DB 공급자로부터 데이터를 액세스하기 위한 함수입니다.
소스 함수	기존의 내장, 외부, SQL, 소스 함수를 이용하여 작성된 함수입니다.
SQL 스칼라	SQL/PL 로 작성되며, 스칼라값을 반환합니다.
SQL 행	SQL/PL 로 작성되며, 행을 반환합니다.
SQL 테이블	SQL/PL 로 작성되며, 테이블을 반환합니다.

4 사용자 정의 함수는 SQL/PL, C, Java, OLE 등을 이용하여 생성합니다.

5 사용자 정의 함수에 대한 정보는 SYSCAT.ROUTINES 뷰를 이용해서 확인합니다.

```
$ db2 "select * from syscat.routines"
```

6 db2look 명령어로 SQL 사용자 정의 함수에 대한 DDL문을 추출할 수 있습니다.

```
$ db2look –d <DB명> –e  -o <출력파일명>
```

Tip

기존의 함수에서 UDT 유형으로 정의된 컬럼을 사용될 수 없습니다. 기존의 함수를 소스 함수로 하는 새로운 UDF가 필요합니다.

Tip

Java 언어를 이용하면, JAR 형태로 UDF를 관리할 수 있습니다.

Tip

사용자 정의 함수를 작성한 언어에 따라 라이브러리를 생성하는 방법이 다릅니다. SQL/PL로 작성하는 경우에는 별도의 생성 과정 없이 create function 문으로 생성합니다.

TOPIC 08-30 CREATE FUNCTION 문

Point

SQL/PL을 이용하여 작성한 사용자 정의 함수의 라이브러리를 생성하고 시그너처를 등록하는 SQL문입니다. C, Java, OLE 등으로 적성된 사용자 정의 함수는 시그너처만 등록되며, 개별적인 방법으로 함수의 라이브러리를 생성해야 합니다.

Tip

SYSADM, DBADM 권한 또는 테이블에 대한 CONTROL, SELECT 특권이 필요할 수 있습니다.

Tip

새로운 스키마명을 이용하여 인덱스명을 지정하려면, 데이터베이스에 대한 IMPLICIT_SCHEMA 특권이 필요합니다.

Tip

기존의 스키마명을 이용하여 인덱스명을 지정하려면, 스키마에 대한 CREATIN 특권이 필요합니다.

1 create function 문의 형식은 다음과 같습니다.

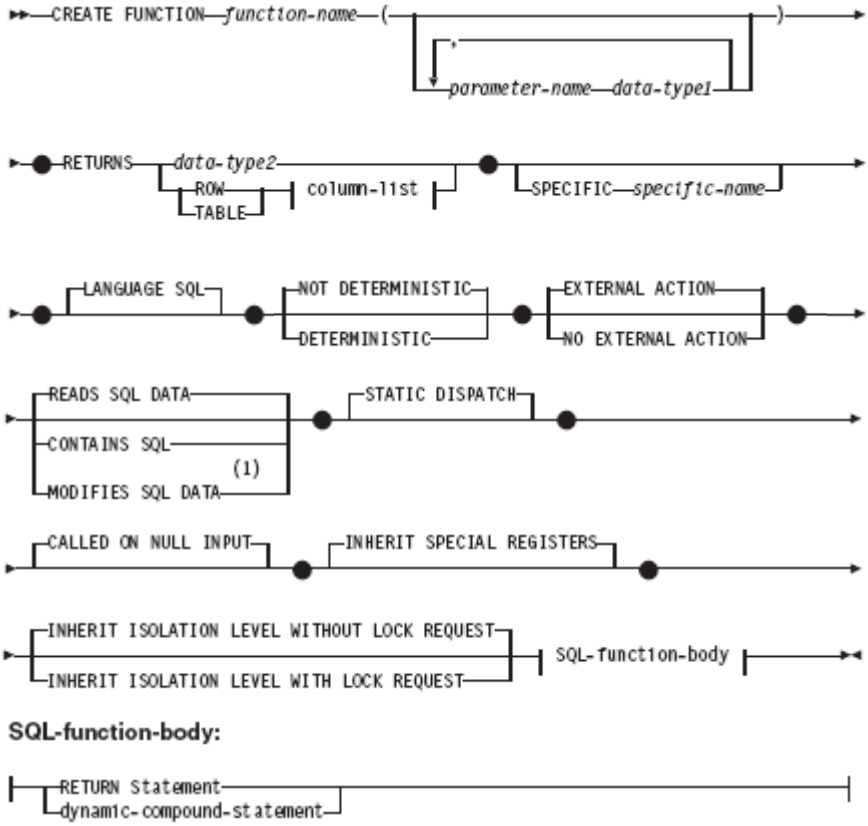


Figure 0830A ··· SQL 사용자 정의 함수를 위한 CREATE FUNCTION 문

2 옵션에 대한 설명은 다음과 같습니다.

모 드	설 명
<함수명>	함수의 이름을 지정합니다.
<인수이름> <데이터유형>	입력 인수의 이름과 데이터 유형을 지정합니다.
RETURNS <데이터유형>	반환되는 값의 데이터 유형입니다.
SPECIFIC <고유함수명>	임의의 고유한 이름으로 지정합니다.
LANGUAGE SQL	SQL/PL을 이용하여 로직을 작성합니다.
DETERMINISTIC	동일한 입력 인수에 대해서 항상 동일한 값을 반환합니다.
CONTAINS SQL	데이터의 조회, 변경을 위한 SQL문을 포함하지 않습니다.
READS SQL DATA	데이터를 변경하는 SQL문을 포함할 수 없습니다.
MODIFIES SQL DATA	지원되는 모든 SQL문을 사용합니다.
CALLED ON NULL INPUT	입력 인수의 값이 NULL인 경우에도 호출됩니다.
<SQL 함수 본문>	단일 SQL문 또는 SQL/PL 블록을 이용하여 구현합니다.

Tip

함수의 시그너처가 다르면 동일한 이름으로 함수를 생성할 수 있습니다. 동일한 이름으로 함수를 오버로딩할 때는 SPECIFIC 옵션으로 고유한 함수명을 지정하는 것이 권장됩니다.

Tip

사용자 정의 함수의 <SPECIFIC명>은 <함수명>과 동일하게 지정할 수 있습니다. 옵션을 지정하지 않으면, SQLyymmddhhmmssxxx 형식으로 엔진이 자동 생성합니다.

Tip

SPECIFIC 옵션은 drop 문 또는 다른 함수에서 참조되는 이름으로만 사용됩니다. SQL문에서 실제로 함수를 사용할 때는 <스키마명>.<함수명>을 사용해야 합니다.

TOPIC 08 31 SQL 사용자 정의 함수

Point

단일 SQL문 또는 SQL/PL을 이용하여 작성한 사용자 정의 함수를 SQL 사용자 정의 함수라고 합니다. CREATE FUNCTION 문으로 함수의 라이브러리가 생성되고, 함수의 시그너처도 등록됩니다. 내장 함수와 동일한 방법으로 SQL문에서 사용합니다.

Tip

단일한 SQL문으로 표현되지 않는 로직을 구현하려면, SQL/PL 의 BEGIN ~ END 블록을 이용합니다.

Tip

SQL/PL에서 ; (세미콜론)은 문장의 구분자로 사용됩니다. create function 문의 끝을 구별할 때는 @ 또는 ! 등의 문자를 사용합니다.

1 SQL 사용자 정의 함수를 위한 로직을 작성하여 임의의 <파일명>으로 저장합니다.

```
$ cat <파일명>
CREATE FUNCTION todate (x varchar(8))
RETURNS date
SPECIFIC TODATE01
LANGUAGE SQL
CONTAINS SQL
NO EXTERNAL ACTION
DETERMINISTIC
RETURN date(SUBSTR(X,1,4)||'-'||SUBSTR(X,5,2)||'-'||SUBSTR(X,7,2))

@
CREATE FUNCTION tan (X DOUBLE)
RETURNS DOUBLE
    LANGUAGE SQL
    CONTAINS SQL
    NO EXTERNAL ACTION
    DETERMINISTIC
    RETURN SIN(X)/COS(X) @

CREATE FUNCTION kes.percent(number int, rate int)
RETURNS decimal(6,2)
F1: BEGIN ATOMIC
   RETURN (number * rate) / 100;
END@
```

Tip

SQL 사용자 정의 함수는 엔진에 내장된 특별한 컴파일러를 이용하여 라이브러리를 작성합니다. 별도의 C 컴파일러는 필요하지 않습니다.

2 create function 문을 이용하여 SQL 사용자 정의 함수를 생성하고, 등록합니다.

```
$ db2 connect to <데이터베이스명>
$ db2 -td@ -svf  <파일명>
```

Tip

UDF를 참조하려면 EXECUTE 권한이 필요합니다. UDF명은 대소문자를 구분하지 않습니다.

3 SQL문에서 기존의 내장 함수와 동일한 방법으로 사용자 정의 함수를 참조합니다.

```
$ db2  –x  " values(TODATE('20040101'))"
2004-01-01

$ db2 -x "SELECT id, salary, kes.percent(salary,5) FROM kes.empl"
    1    130     6.00
    2    280    14.00
    3    330    16.00
    4    310    15.00
```

TOPIC 08-32 저장 프로시저

Point

서버에 저장된 프로그램 로직입니다. 클라이언트에서 CALL 문으로 서버의 저장 프로시저를 호출하면, 서버에서 로직이 실행되어 결과만 클라이언트로 반환됩니다. CREATE PROCEDURE 문과 DROP PROCEDURE 문으로 관리합니다.

1 저장 프로시저는 클라이언트 머신의 응용프로그램에서 실행해야 하는 로직을 서버의 데이터베이스에 저장하여 서버 머신에서 직접 실행함으로써 클라언트와 서버 간의 데이터 전송량을 줄이고, 성능을 향상시킵니다.

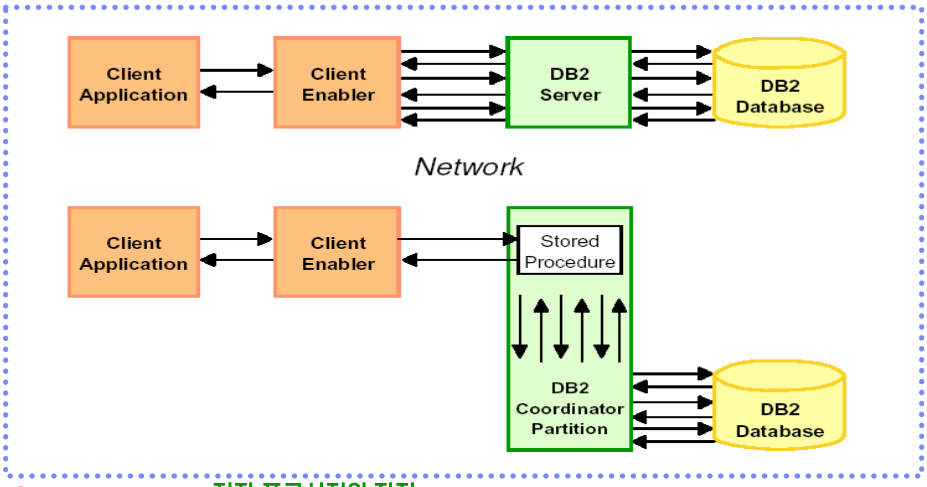


Figure 0832A • • • 저장 프로시저의 장점

2 공용 로직을 저장 프로시저로 만들어서 사용하면 관리가 용이합니다. 클라이언트 응용프로그램의 개별적인 코딩으로 인한 오류와 소스를 반복적으로 작성해야 하는 부담을 줄일 수 있습니다. 프로시저의 로직이 변경되면, 서버의 저장 프로시저만 재생성하면 됩니다.

3 저장 프로시저는 실행시에 서버의 자원을 사용하여 실행됩니다. 일반적으로 클라이언트 머신보다 서버 머신의 사양이 좋으므로, 동일한 로직을 실행할 때 실행 시간이 단축될 수 있습니다.

4 서버 머신의 OS 에 의존적인 로직의 구현이 가능합니다. 클라이언트가 Windows 이고, 서버가 UNIX인 경우에 UNIX 에서만 지원되는 기능을 프로시저의 로직에 포함시킬 수 있습니다.

5 저장 프로시저의 유형은 작성하는 언어에 의해 2 가지로 분류됩니다.

유 형	설 명
외부 소스	ESQL, C, Java 등의 프로그래밍 언어로 작성된 라이브러리를 이용하며, 스칼라 값 또는 결과 집합을 반환합니다.
SQL	SQL/PL 또는 PL/SQL로 작성되며, 스칼라 값 또는 결과 집합을 반환합니다.

6 저장 프로시저는 SQL/PL, PL/SQL, ESQL, C, Java 등의 언어를 이용하여 생성합니다.

7 저장 프로시저에 대한 정보는 SYSCAT.ROUTINES 뷰를 이용해서 확인합니다.

```
$ db2 "select * from syscat.routines"
```

8 db2look 명령어로 SQL 저장 프로시저에 대한 DDL문을 추출할 수 있습니다.

```
$ db2look –d <DB명> –e  -o <출력파일명>
```

Tip

PL/SQL은 DB2 9.7이후 지원되는 프로시저 언어입니다.

Tip

저장 프로시저를 작성한 언어에 따라 라이브러리를 생성하는 방법이 다릅니다. SQL/PL 또는 PL/SQL로 작성하는 경우에는 별도의 생성 과정 없이 create proceudre 문으로 생성합니다.

TOPIC 08 33 CREATE PROCEDURE 문

Point

SQL/PL 저장 프로시저의 라이브러리를 생성하고 시그너처를 등록하는 SQL문입니다. ESQL, C, Java 등으로 적성된 저장 프로시저는 시그너처만 등록되며, 개별적인 방법으로 저장 프로시저의 라이브러리를 생성해야 합니다.

Tip

SYSADM, DBADM 권한 또는 데이터베이스에 대한 BINDADD 특권이 필요할 수 있습니다.

1 create procedure 문의 형식은 다음과 같습니다.

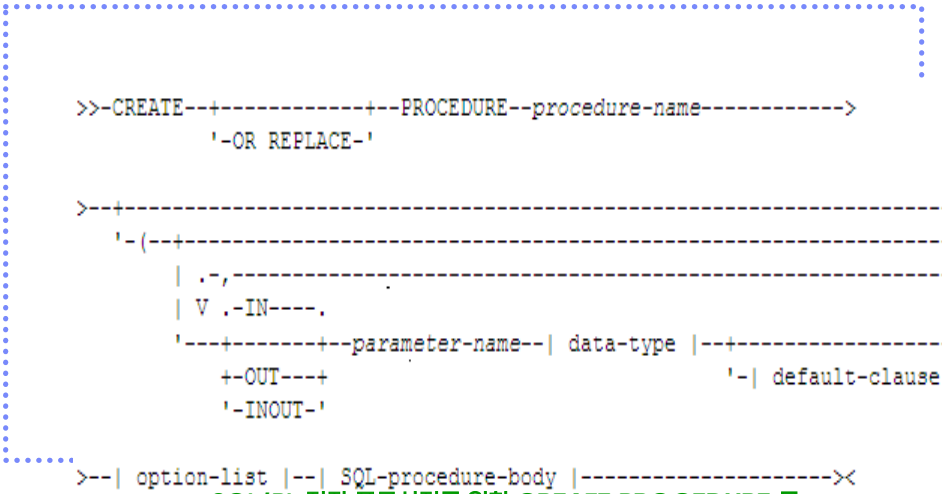


Figure 0833A SQL/PL 저장 프로시저를 위한 CREATE PROCEDURE 문

Tip

사용자 정의 함수의 <SPECIFIC명>은 <함수명>과 동일하게 지정할 수 있습니다. 옵션을 지정하지 않으면, SQLyymmddhhmmssxxx 형식으로 엔진이 자동 생성합니다.

2 옵션에 대한 설명은 다음과 같습니다.

모 드	설 명
<프로시저명>	프로시저의 이름을 지정합니다.
<인수 입출력 유형>	인수의 유형은 IN, OUT, INOUT 으로 지정합니다.
<인수 이름>	인수의 이름을 지정합니다.
<데이터유형>	인수의 데이터 유형을 지정합니다.
SPECIFIC	임의의 고유한 이름으로 지정합니다.
DYNAMIC RESULT SETS	반환할 결과 집합의 개수를 지정합니다.
CONTAINS SQL	데이터의 조회, 변경을 위한 SQL문을 포함하지 않습니다.
READS SQL DATA	데이터를 변경하는 SQL문을 포함할 수 없습니다.
MODIFIES SQL DATA	지원되는 모든 SQL문을 사용합니다.
DETERMINISTIC	동일한 입력 인수에 대해서 항상 동일한 값을 반환합니다.
CALLED ON NULL INPUT	입력 인수의 값이 NULL인 경우에도 호출됩니다.
LANGUAGE SQL	SQL/PL 로 작성한 SP입니다.
NO EXTERNAL ACTION	데이터베이스 시스템이 관리하지 않는 외부 오브젝트에 대한 상태를 변경시키는 로직의 포함 여부를 지정합니다.
<SQL 함수 본문>	SQL/PL 블록을 이용하여 구현합니다.

TOPIC 08-34 SQL/PL 프로시저

Point

SQL/PL 언어를 이용하여 저장 프로시저를 생성합니다. CALL 문을 이용하여 호출합니다.

Tip

SQL/PL에서 ; (세미콜론)은 문장의 구분자로 사용됩니다. create function 문의 끝을 구별할 때는 @ 또는 ! 등의 문자를 사용합니다.

1 SQL 저장 프로시저를 위한 로직을 작성하여 임의의 <파일명>으로 저장합니다.

```
$ cat <파일명>
CREATE PROCEDURE myproc (IN deptNumber CHAR(3),
                         OUT medianSalary DOUBLE)
LANGUAGE SQL
BEGIN
  DECLARE SQLCODE INTEGER;
  DECLARE SQLSTATE CHAR(5);
  DECLARE v_numRecords INT DEFAULT 1;
  DECLARE v_counter INT DEFAULT 0;

  DECLARE c1 CURSOR FOR
       SELECT CAST(salary AS DOUBLE) FROM employee
       WHERE workdept = deptNumber
       ORDER BY salary;
  DECLARE EXIT HANDLER FOR NOT FOUND
       SET medianSalary = 6666;

  SET medianSalary = 0;
  SELECT COUNT(*) INTO v_numRecords FROM employee
  WHERE workdept = deptNumber;
  OPEN c1;
  WHILE v_counter < (v_numRecords / 2 + 1) DO
       FETCH c1 INTO medianSalary;
       SET v_counter = v_counter + 1;
  END WHILE;
  CLOSE c1;
END @
```

2 create procedure 문을 이용하여 SQL 저장 프로시저를 생성하고, 등록합니다.

```
$ db2 connect to <데이터베이스명>
$ db2 -td@ -svf  <파일명>
```

Tip

SP를 호출하려면 EXECUTE 권한이 필요합니다. SP명은 대소문자를 구분하지 않습니다.

3 CALL 문을 이용하여 저장 프로시저를 호출합니다.

```
$ db2  "call myproc('A00',?)"
Value of output parameters
--------------------------
Parameter Name  : MEDIANSALARY
Parameter Value : +4.65000000000000E+004
Return Status = 0
```

Tip

CLP에서 SP를 호출하려면, OUT 유형의 인수값에는 출력용 변수명 대신에 ?(물음표)를 이용합니다.

TOPIC 08 35 PL/SQL 프로시저

Point

PL/SQL 언어를 이용하여 저장 프로시저를 생성합니다. CALL 문을 이용하여 호출합니다.

Tip

9.7 이후로는 PL/SQL을 이용하여 저장 프로시저를 생성할 수 있습니다.

1 PL/SQL을 이용하여 저장 프로시저를 생성하고자 하는 경우에는 Registry 변수에 compatibility vector를 '800'로 설정한 후, 데이터베이스를 생성합니다.

```
$ db2set compatibility Vector = 800
```

2 create procedure 문을 이용하여 SQL 저장 프로시저를 개발합니다.

```
$ cat script.db2
set sqlcompat plsql/

CREATE OR REPLACE FUNCTION emp_comp (
    p_sal          NUMBER,
    p_comm         NUMBER )
RETURN NUMBER
IS
BEGIN
   RETURN (p_sal + NVL(p_comm, 0)) * 24;
END emp_comp
/

CREATE OR REPLACE PROCEDURE update_comp(p_name IN
    VARCHAR) AS
BEGIN
   UPDATE emp SET tot_comp = emp_comp(salary, comm)
    WHERE name = p_name;
END update_comp
/
```

작성된 구문이 PL/SQL구문임을 알려주는 기능

Tip

작성된 스크립트에서 구분자가 '/' 또는 ';'인 경우에는 -td/-vf로 생성합니다.

3 DB2CLP 창에서 컴파일 및 생성합니다.

```
$ db2 –td/ -vf script.db2
```

4 CLP창에서 프로시저를 호출할 때에는 CALL을 사용합니다.

```
$ db2 call update_comp('Curly')
```

Memo