Tibero

SQL 참조 안내서

Tibero 7

대한민국 경기도 성남시 분당구 황새울로258번길 29, BS 타워 9층 우)13595

Website

http://www.tmaxtibero.com

기술서비스센터

Tel : +82-1544-8629

E-Mail : info@tmax.co.kr

Open Source Software Notice

Some modules or files of this product are subject to the terms of the following licenses. : OpenSSL, RSA Data Security, Inc., Apache Foundation, Jean-loup Gailly and Mark Adler, Paul Hsieh's hash

Detailed Information related to the license can be found in the following directory : ${INSTALL_PATH}/license/oss_licenses

본 제품의 일부 파일 또는 모듈은 다음의 라이선스를 준수합니다. : OpenSSL, RSA Data Security, Inc., Apache Foundation, Jean-loup Gailly and Mark Adler, Paul Hsieh's hash

관련 상세한 정보는 제품의 다음의 디렉터리에 기재된 사항을 참고해 주십시오. : ${INSTALL_PATH}/license/oss_licenses

안내서 정보

안내서 제목: Tibero SQL 참조 안내서

발행일: 2024-08-22

소프트웨어 버전: Tibero 7.2.2

안내서 버전: v7.2.2

내용 목차

그림 목차

표 목차

예 목차

안내서에 대하여

안내서의 대상

본 안내서는 Tibero®(이하 Tibero)를 사용하여 데이터베이스 작업을 수행하거나 애플리케이션 프로그램 작성 등에 필요한 SQL를 참조하려는 모든 데이터베이스 사용자를 대상으로 기술한다.

Tibero에서 제공하는 SQL은 ANSI(American National Standard Institute)와 ISO/IEC(International Standard Organization/International Electrotechnical Commission)에서 공동으로 제정한 SQL-92와 SQL-99, SQL-2003 표준의 일부를 지원한다.

안내서의 전제 조건

본 안내서는 Tibero에서 정보를 관리하는 데 필요한 SQL 문장을 설명하는 안내서이다. 따라서 본 안내서를 원활히 이해하기 위해서는 다음과 같은 사항을 미리 알고 있어야 한다.

- 데이터베이스의 이해
- RDBMS의 이해
- SQL의 이해

안내서의 제한 조건

본 안내서는 Tibero를 실무에 적용하거나 운용하는 데 필요한 모든 사항을 포함하고 있지 않다. 따라서 설치, 환경설정 등 운용 및 관리에 대해서는 각 제품 안내서를 참고하기 바란다.

참고

Tibero의 설치 및 환경설정에 관한 내용은 "Tibero 설치 안내서"를 참고한다.

안내서 구성

Tibero SQL 참조 안내서는 총 9개의 장과 Appendix로 구성되어 있다.

각 장의 주요 내용은 다음과 같다.

- 제1장: SQL 소개

  SQL의 기본 개념과 SQL 문장의 종류에 대해 기술한다.

- 제2장: SQL 문장의 구성요소

  Tibero에서 제공하는 데이터 타입과 스키마 객체에 대해 기술한다.

- 제3장: SQL 연산

  Tibero에서 제공하는 SQL 연산자와 연산식, 조건식에 대해 기술한다.

- 제4장: 함수

  Tibero에서 제공하는 내장 함수에 대해 기술한다.

- 제5장: SQL 질의

  Tibero에서 사용할 수 있는 SQL 질의에 대해 기술한다.

- 제6장: 실체화 뷰

  Tibero에서 제공하는 실체화 뷰를 사용하는 방법에 대해 기술한다.

- 제7장: 데이터 정의어

  Tibero에서 제공하는 데이터 정의어에 대해 기술한다.

- 제8장: 데이터 조작어

  Tibero에서 제공하는 데이터 조작어에 대해 기술한다.

- 제9장: 트랜잭션 및 세션 관리 언어

  Tibero에서 제공하는 트랜잭션 및 세션 관리 언어에 대해 기술한다.

- Appendix.A: 예약어

  Tibero에서 사용하는 예약어에 대해 기술한다.

안내서 규약

표기	의미
<<AaBbCc123>>	프로그램 소스 코드의 파일명
<Ctrl>+C	Ctrl과 C를 동시에 누름
[Button]	GUI의 버튼 또는 메뉴 이름
진하게	강조
" "(따옴표)	다른 관련 안내서 또는 안내서 내의 다른 장 및 절 언급
'입력항목'	화면 UI에서 입력 항목에 대한 설명
하이퍼링크	메일 계정, 웹 사이트
>	메뉴의 진행 순서
+----	하위 디렉터리 또는 파일 있음
|----	하위 디렉터리 또는 파일 없음
참고	참고 또는 주의사항
주의	주의할 사항
[그림 1.1]	그림 이름
[예 1.1]	예제 이름
`AaBbCc123`	Java 코드, XML 문서
[*command argument*]	옵션 파라미터
< xyz >	'<'와 '>' 사이의 내용이 실제 값으로 변경됨
|	선택 사항. 예) A|B: A나 B 중 하나
…	파라미터 등이 반복되어서 나옴
${ }	환경변수

시스템 사용 환경

	요구 사항
Platform	HP-UX 11i v3(11.31)
	Solaris (Solaris 11)
	AIX (AIX 7.1/AIX 7.2/AIX 7.3)
	GNU (X86, 64, IA64)
	Red Hat Enterprise Linux 7 kernel 3.10.0 이상
	Windows(x86) 64bit
Hardware	최소 2.5GB 하드디스크 공간
	1GB 이상 메모리 공간
Compiler	PSM (C99 지원 필요)
	tbESQL/C (C99 지원 필요)

관련 안내서

안내서	설명
Tibero 설치 안내서	설치 과정에 필요한 시스템 요구사항과 설치 및 제거 방법을 기술한 안내서이다.
Tibero tbCLI 안내서	Call Level Interface인 tbCLI의 개념과 구성요소, 프로그램 구조를 소개하고 tbCLI 프로그램을 작성하는 데 필요한 데이터 타입, 함수, 에러 메시지를 기술한 안내서이다.
Tibero 애플리케이션 개발자 안내서	각종 애플리케이션 라이브러리를 이용하여 애플리케이션 프로그램을 개발하는 방법을 기술한 안내서이다.
Tibero External Procedure 안내서	External Procedure를 소개하고 이를 생성하고 사용하는 방법을 기술한 안내서이다.
Tibero JDBC 개발자 안내서	Tibero에서 제공하는 JDBC 기능을 이용하여 애플리케이션 프로그램을 개발하는 방법을 기술한 안내서이다.
Tibero tbESQL/C 안내서	C 프로그래밍 언어를 사용해 데이터베이스 작업을 수행하는 각종 애플리케이션 프로그램을 작성하는 방법을 기술한 안내서이다.
Tibero tbESQL/COBOL 안내서	COBOL 프로그래밍 언어를 사용해 데이터베이스 작업을 수행하는 각종 애플리케이션 프로그램을 작성하는 방법을 기술한 안내서이다.
Tibero tbPSM 안내서	저장 프러시저 모듈인 tbPSM의 개념과 문법, 구성요소를 소개하고, 프로그램을 작성하는 데 필요한 제어 구조, 복합 타입, 서브프로그램, 패키지와 SQL 문장을 실행하고 에러를 처리하는 방법을 기술한 안내서이다.
Tibero tbPSM 참조 안내서	저장 프러시저 모듈인 tbPSM의 패키지를 소개하고, 이러한 패키지에 포함된 각 프러시저와 함수의 프로토타입, 파라미터, 예제 등을 기술한 참조 안내서이다.
Tibero 관리자 안내서	Tibero의 동작과 주요 기능의 원활한 수행을 보장하기 위해 DBA가 알아야 할 관리 방법을 논리적 또는 물리적 측면에서 설명하고, 관리를 지원하는 각종 도구를 기술한 안내서이다.
Tibero 유틸리티 안내서	데이터베이스와 관련된 작업을 수행하기 위해 필요한 유틸리티의 설치 및 환경설정, 사용 방법을 기술한 안내서이다.
Tibero TAS 안내서	Tibero Active Storage(TAS)를 사용해서 Tibero의 파일을 관리하고자 하는 관리자를 대상으로 기술한 안내서이다.

안내서	설명
Tibero 에러 참조 안내서	Tibero를 사용하는 도중에 발생할 수 있는 각종 에러의 원인과 해결 방법을 기술한 안내서이다.
Tibero 참조 안내서	Tibero의 동작과 사용에 필요한 초기화 파라미터와 데이터 사전, 정적 뷰, 동적 뷰를 기술한 참조 안내서이다.
Tibero TDP.NET 안내서	Tibero Data Provider for .NET 기능을 기술하는 안내서이다.
Tibero IMCS 안내서	Tibero에서 제공하는 In-Memory Column Store(이하 IMCS) 기능을 기술하는 안내서이다.

제1장 SQL 소개

본 장에서는 SQL의 기본 개념과 데이터베이스 작업에 필요한 SQL 문장에 대해 간략히 소개한다.

1.1. 개요

SQL(Structured Query Language)은 영문명을 풀이하면 구조화된 질의 언어라는 의미이다. 단순하게 질의만을 수행하는 것이 아니라 데이터베이스의 모든 작업을 통제하는 비절차적(Non-procedural) 언어이다.

비절차적 언어라는 것은 데이터베이스 사용자(이하 사용자)가 SQL을 사용해 원하는 작업의 결과만 기술하고, 그 작업이 어떻게 수행될 것인지는 전혀 고려하지 않아도 된다는 것을 의미한다. 사용자가 작성한 SQL 문장을 데이터베이스 안에서 어떻게 수행할 것인가는 각 시스템에 의해서 결정된다. 데이터베이스 시스템은 데이터를 어떻게 저장할 것인지, 메모리를 어떻게 이용할 것인지, 데이터를 어떠한 순서로 읽을 것인지 등에 대한 정책과 최적화 과정을 수립하고 있다.

SQL은 일반적인 데이터베이스 작업을 기술하기 위한 SQL 문장(Statement)을 정의하고 있다. 이처럼 SQL을 이용하여 기술할 수 있는 데이터베이스 작업은 다음과 같다.

- 스키마 객체(Schema Object)의 생성, 변경, 제거
- 데이터베이스 질의
- 데이터의 삽입, 갱신, 삭제
- 트랜잭션 관리 및 세션 관리 등을 포함하는 데이터베이스 관리

1.2. SQL 표준

SQL 표준은 ANSI(American National Standard Institute)와 ISO/IEC(International Standard Organization/International Electrotechnical Commission)에서 공동으로 제정하였다.

SQL 표준은 1992년과 1999년에 각각 버전 2와 버전 3이 제정되었다. 1992년에 발표된 SQL 표준은 SQL2 또는 SQL-92라고 불리며, 관계형 데이터베이스를 위한 언어이다. 1999년에 제정된 SQL 표준은 SQL3 또는 SQL-99라고 불리며, SQL2에 객체지향 개념을 추가하여 확장한 객체관계형 데이터베이스 언어이다. SQL-2003에는 XML, OLAP, Object-Retational 관련 기능과 MERGE 문 등이 추가되었다.

SQL 표준은 구현 범위에 따라 몇 개의 단계에 걸쳐 제정된다.

SQL-92는 다음과 같이 3단계로 구분된다.

- Entry Level

Entry Level은 SQL-92 내용의 80% 이상을 포함한다.

- Intermediate Level

  Entry Level을 포함하고, 여기에 덧붙여 추가적인 내용을 포함하고 있다.

- Full Level

  SQL-92 내용의 전체를 포함한다.

SQL 표준의 내용은 매우 방대하여, 제정한 모든 내용을 지원하는 상업용 데이터베이스 시스템은 하나도 없다. Tibero에서는 Entry Level의 대부분과 Intermediate Level의 일부를 지원한다.

SQL 언어는 프로그램 언어에 포함되어 사용되기도 한다. SQL 표준으로 정해진 프로그램 언어 인터페이스로는 내장 SQL(Embedded SQL, 이하 ESQL)과 저장 프러시저(Persistent Stored Module, 이하 PSM)가 있다. Tibero에서는 각각에 대해 tbESQL과 tbPSM이라는 인터페이스를 제공하고 있다.

tbPSM은 Tibero 서버 쪽에 저장되고 실행되는 프로그램 인터페이스이다. tbPSM 프로그램은 서버 쪽에서만 실행되므로 클라이언트와의 통신이 최소화되어 작업의 수행 속도를 크게 향상시킬 수 있다. 하지만 클라이언트 쪽의 사용자가 프로그램 실행의 과정을 검토할 수 없으므로 에러를 처리하기 위한 작업을 좀 더 세밀하게 해야 한다.

또한, tbESQL이라는 별도의 SQL을 지원하는데, tbESQL은 프로그램 언어와 SQL의 장점을 융합한 것이다. 일반적으로, 프로그램 언어는 매우 복잡하고 세밀한 작업을 빠르게 수행할 수 있으며, SQL은 간단한 문장만으로 데이터베이스에 대한 직접적인 작업을 표현할 수 있다.

참고

1.tbESQL에 대한 자세한 내용은 "Tibero tbESQL/C 안내서"를 참고한다.

2.tbPSM에 대한 자세한 내용은 "Tibero tbPSM 안내서"를 참고한다.

1.3. SQL 문장의 종류

SQL 표준에서 정의하고 있는 SQL 문장은 크게 다음과 같이 세 가지로 나눌 수 있다.

- 데이터 정의어(Data Definition Language)
- 데이터 조작어(Data Manipulation Language)
- 데이터 제어어(Data Control Language)

참고

본 안내서에서는 데이터 제어어에 포함되는 COMMIT, ROLLBACK 등의 SQL 명령어를 트랜잭션 및 세션 관리 언어의 일부로 설명한다. 따라서 전체 데이터 제어어에 대한 내용은 관련 문서를 참고하기 바란다.

1.3.1. 데이터 정의어

데이터 정의어(이하 DDL)는 데이터 간에 관계를 정의하여 데이터베이스 구조를 설정하는 SQL 문장이다. 기본적으로 스키마 객체를 생성, 변경, 제거하기 위해 사용된다. 대부분의 스키마 객체에 대해서 생성, 변경, 제거를 위한 CREATE, ALTER, DROP 명령이 모두 존재하지만, 일부 객체는 한두 가지 명령어밖에 존재하지 않는다.

DDL은 그 외에 특권(Privilege)과 역할(Role)을 부여하고 회수하기 위한 명령어와 이러한 특권과 역할을 감시하기(Auditing) 위한 명령어, 테이블 객체에 대한 최적화(Optimization)를 위한 명령어를 포함한다.

Tibero에서 제공하는 DDL은 다음과 같다.

구분	명령어	설명
데이터베이스	CREATE DATABASE	데이터베이스를 생성한다.
	ALTER DATABASE	데이터베이스를 변경한다.
테이블	CREATE TABLE	테이블을 생성한다.
	ALTER TABLE	테이블을 변경한다.
	DROP TABLE	테이블을 제거한다.
테이블스페이스	CREATE TABLESPACE	테이블스페이스를 생성한다.
	ALTER TABLESPACE	테이블 스페이스를 변경한다.
	DROP TABLESPACE	테이블 스페이스를 제거한다.
인덱스	CREATE INDEX	인덱스를 생성한다.
	ALTER INDEX	인덱스를 변경한다.
	DROP INDEX	인덱스를 제거한다.
뷰	CREATE VIEW	뷰를 생성한다.
	ALTER VIEW	뷰를 변경한다.
	DROP VIEW	뷰를 제거한다.
동의어	CREATE SYNOMYM	동의어를 생성한다.
	DROP SYNOMYM	동의어를 제거한다.
사용자	CREATE USER	사용자를 생성한다.
	ALTER USER	사용자를 변경한다.
	DROP USER	사용자를 제거한다.
함수	CREATE FUNCTION	함수를 생성한다.
	ALTER FUNCTION	함수를 변경한다.
	DROP FUNCTION	함수를 제거한다.
프러시저	CREATE PROCEDURE	프러시저를 생성한다.
	ALTER PROCEDURE	프러시저를 변경한다.

구분	명령어	설명
	DROP PROCEDURE	프러시저를 제거한다.
타입	CREATE TYPE	타입을 생성한다.
	ALTER TYPE	타입을 변경한다.
	DROP TYPE	타입을 제거한다.
특권	GRANT	사용자에게 특권을 부여한다.
	REVOKE	사용자에게 특권을 회수한다.
역할	CREATE ROLE	역할을 생성한다.
	ALTER ROLE	역할을 변경한다.
	DROP ROLE	역할을 제거한다.
객체	RENAME	테이블, 뷰, 동의어, 시퀀스 등의 스키마 객체의 이름을 변경한다.
감시	AUDIT	특권의 사용을 감시한다.
	NOAUDIT	특권의 감시를 해제한다.

참고

자세한 내용은 “제7장 데이터 정의어”를 참고한다.

1.3.2. 데이터 조작어

데이터 조작어(이하 DML)는 데이터베이스에 저장된 데이터에 대한 질의, 삽입, 갱신, 삭제를 수행하기 위한 SQL 문장이다.

Tibero에서 제공하는 DML의 명령어는 다음과 같다.

명령어	설명
SELECT	데이터를 조회한다.
INSERT	데이터를 삽입한다.
UPDATE	데이터를 변경한다.
DELETE	데이터를 삭제한다.

참고

자세한 내용은 “제8장 데이터 조작어”를 참고한다.

1.3.3. 트랜잭션 및 세션 관리 언어

트랜잭션 관리 언어는 트랜잭션을 관리하는 SQL 문장이다. 즉, 트랜잭션의 특성을 설정하거나 트랜잭션을 완료하고 저장하는 등의 작업을 수행한다. 세션 관리 언어는 세션의 특성을 설정하기 위한 SQL 문장이다.

Tibero에서 제공하는 트랜잭션 관리 및 세션 관리 명령어는 다음과 같다.

명령어	설명
ALTER SESSION	세션을 변경한다.
COMMIT	트랜잭션을 완료하고 저장한다.
ROLLBACK	트랜잭션을 원래 상태로 복구한다.
SAVEPOINT	저장점을 설정한다.
SET ROLE	사용자에게 할당된 역할을 활성화하거나 비활성화한다.
SET TRANSACTION	트랜잭션의 특성을 설정한다.

참고

자세한 내용은 "제9장 트랜잭션 및 세션 관리 언어"를 참고한다.

제2장 SQL 문장의 구성요소

본 장에서는 SQL 표준에서 정의하고 있는 데이터 타입과 스키마 객체에 대해 설명한다.

2.1. 데이터 타입

Tibero에서는 SQL 표준에 기반한 여러 가지 데이터 타입을 제공한다.

다음은 Tibero에서 제공하는 데이터 타입이다.

구분	데이터 타입
문자형	CHAR, VARCHAR, VARCHAR2, NCHAR, NVARCHAR, NVARCHAR2, RAW, LONG, LONG RAW
숫자형	NUMBER, INTEGER, FLOAT, BINARY_FLOAT, BINARY_DOUBLE
날짜형	DATE, TIME, TIMESTAMP, TIMESTAMP WITH TIME ZONE, TIMESTAMP WITH LOCAL TIME ZONE
간격형	INTERVAL YEAR TO MONTH, INTERVAL DAY TO SECOND
대용량 객체형	CLOB, BLOB, XMLTYPE, JSON
내재형	ROWID
사용자 정의형	배열, 네스티드 테이블

2.1.1. 문자형

문자형은 문자열을 표현하는 데이터 타입이다. 문자형에는 CHAR, VARCHAR, VARCHAR2, NCHAR, NVARCHAR, NVARCHAR2, RAW, LONG, LONG RAW 타입 등이 있다.

CHAR

CHAR 타입은 문자열을 저장하는 데이터 타입이다. 항상 고정된 문자열 길이를 갖는다.

```
CHAR(size[BYTE|CHAR])
```

CHAR 타입은 다음과 같은 특징이 있다.

- 문자열은 최대 2,000byte나 2,000자까지 선언할 수 있다. 변환된 문자열의 길이가 2,000byte나 2,000자를 넘으면 에러가 발생한다.

Tibero에서는 CHAR 타입으로 정의한 컬럼에 입력된 문자열을 데이터베이스의 문자 집합에 맞게 변환하여 저장한다. 이때 변환된 문자열은 2,000byte나 2,000자를 초과해서는 안 된다.

- 문자열의 길이는 byte와 문자를 기준으로 지정할 수 있다.

  CHAR (10 BYTE), CHAR (10 CHAR)의 형태로 선언한다. 뒷부분에 아무런 옵션도 지정하지 않고 CHAR(10)의 형태로 선언하면, byte로 문자열의 길이가 지정된다. 각각 BYTE로 선언할 경우엔 2,000byte까지 저장할 수 있고, CHAR로 선언할 경우엔 2,000자까지 저장할 수 있다. 따라서 CHAR로 문자열의 길이를 선언할 때 실제 컬럼의 길이는 데이터베이스가 사용하는 문자 집합에 따라 좌우된다. 즉, 한 문자가 몇 byte로 표현되는지에 따라 그 길이가 달라진다.

- SQL 문장에서 CHAR 타입의 값을 표현할 때에는 항상 작은따옴표(' ')를 사용한다.

- 문자열의 길이가 0인 값은 NULL로 인식된다.

다음은 CHAR 타입을 설명하는 예이다.

```
PRODUCT_NAME CHAR(10)
```

위 예제에서 보듯이 PRODUCT_NAME 컬럼은 항상 10byte의 문자열 길이를 갖는다. 예를 들어 **'Tibero'** 문자열이 입력되었다면, 네 개의 공백 문자가 채워져서 **'Tibero____'** 문자열이 저장된다. 이처럼 선언된 길이보다 짧은 문자열이 입력되면 남은 부분은 공백 문자(space)로 채워진다.

VARCHAR

VARCHAR 타입도 CHAR 타입과 마찬가지로 문자열을 저장하는 데이터 타입이다. 단, CHAR 타입과 다른 점은 문자열 길이가 일정하지 않은 가변 길이를 갖는다는 것이다.

```
VARCHAR(size[BYTE|CHAR])
```

VARCHAR 타입은 다음과 같은 특징이 있다.

- 문자열은 최대 65,532byte나 65,532자까지 선언할 수 있다. 변환된 문자열의 길이가 65,532byte나 65,532자를 넘으면 에러가 발생한다. Tibero에서는 VARCHAR 타입으로 정의한 컬럼에 입력된 문자열을 데이터베이스의 문자 집합에 맞게 변환하여 저장한다. 이때 변환된 문자열은 65,532byte를 초과해서는 안 된다.

- 문자열의 길이는 byte와 문자를 기준으로 지정할 수 있다.

  VARCHAR (10 BYTE), VARCHAR (10 CHAR)의 형태로 선언한다. 뒷부분에 아무런 옵션도 지정하지 않고 VARCHAR(10)의 형태로 선언하면, byte로 문자열의 길이가 지정된다. 각각 BYTE로 선언할 경우엔 65,532byte까지 저장할 수 있고, CHAR로 선언할 경우엔 65,532자까지 저장할 수 있다. 따라서 CHAR로 문자열의 길이를 선언할 때 실제 컬럼의 길이는 데이터베이스가 사용하는 문자 집합에 따라 좌우된다. 즉, 한 문자가 몇 byte로 표현되는지에 따라 그 길이가 달라진다.

- SQL 문장에서 VARCHAR 타입의 값을 표현할 때에는 항상 작은따옴표(' ')를 사용한다.

- 문자열의 길이가 0인 값은 NULL로 인식된다.

다음은 VARCHAR 타입을 설명하는 예이다.

```
EMP_NAME VARCHAR(10)
```

위 예제에서 보듯이 EMP_NAME 컬럼은 10byte의 문자열 길이를 갖는다. 예를 들어 **'Peter'** 문자열이 입력되었다면 **'Peter'** 문자열이 저장된다. 다시 말해 EMP_NAME 컬럼의 문자열 길이는 10byte로 선언되었지만 실제로 저장된 문자열 길이는 5byte이다. 이처럼 VARCHAR 타입은 선언된 문자열 길이의 범위 내에서 입력된 문자열 길이와 동일한 길이를 갖는다.

VARCHAR2

VARCHAR2 타입은 VARCHAR 타입과 완전히 동일하다.

NCHAR

NCHAR 타입은 유니코드 문자열을 저장하기 위한 타입이다. 항상 고정된 문자열 길이를 갖는다.

```
NCHAR(size)
```

NCHAR 타입은 다음과 같은 특징이 있다.

- 기본적으로 CHAR 타입과 유사하지만, 문자열의 길이가 문자 기준이다.

  데이터베이스에 저장되는 타입의 길이는 다국어 문자 집합에 따라 달라진다. 예를 들어 UTF8인 경우엔 size의 최대 3배, UTF16인 경우엔 size의 최대 2배가 된다.

- NCHAR 타입의 문자열의 최대 길이는 2,000자이다.
- SQL 문장에서 NCHAR 타입의 값을 표현할 때에는 항상 작은따옴표(' ')를 사용한다.
- 문자열의 길이가 0인 값은 NULL로 인식된다.

NVARCHAR

NVARCHAR 타입은 NCHAR과 마찬가지로 유니코드 문자열을 저장하기 위한 타입이다. 단, NCHAR 타입과 다른 점은 문자열 길이가 일정하지 않은 가변 길이를 갖는다는 것이다.

```
NVARCHAR(size)
```

NVARCHAR 타입은 다음과 같은 특징이 있다.

- 기본적으로 VARCHAR 타입과 유사하지만, 문자열의 길이가 문자 기준이다.

데이터베이스에 저장되는 타입의 길이는 다국어 문자 집합에 따라 달라진다. 예를 들어 UTF8인 경우엔 size의 최대 3배, UTF16인 경우엔 size의 최대 2배가 된다.

- NVARCHAR 타입의 문자열의 최대 길이는 65,532자이다. 단, 65,532byte를 초과할 수 없다.
- SQL 문장에서 NVARCHAR 타입의 값을 표현할 때에는 항상 작은따옴표(' ')를 사용한다.
- 문자열의 길이가 0인 값은 NULL로 인식된다.

NVARCHAR2

NVARCHAR2 타입은 NVARCHAR 타입과 완전히 동일하다.

RAW

RAW 타입은 임의의 바이너리 데이터를 저장하는 데이터 타입이다. 이때 바이너리 데이터는 선언된 최대 길이 내에서 임의의 길이를 갖는다.

RAW 타입이 CHAR, VARCHAR 타입과 다른 점은 RAW 타입은 데이터 중간에 NULL 문자('\0')가 올 수 있지만 CHAR, VARCHAR 타입은 그렇지 않다. 따라서 RAW 타입은 NULL 문자로 데이터의 끝을 나타낼 수 없으므로 항상 길이 정보를 같이 저장한다.

```
RAW(size)
```

RAW 타입은 다음과 같은 특징이 있다.

- 최대 2,000byte까지 선언할 수 있다.
- 선언된 길이 내에서 가변 길이를 갖는다.
- 데이터 중간에 NULL 문자('\0')가 올 수 있다.
- 입출력을 수행할 때 RAW 타입의 데이터는 16진수로 표현된다.

  예를 들면 4byte의 데이터는 16진수로 '012345AB'로 표현되며 필요한 경우 맨 앞이 0으로 시작되어야 한다.

LONG

LONG 타입은 VARCHAR 타입을 확장한 데이터 타입이다. VARCHAR 타입과 마찬가지로 문자열이 저장된다.

```
LONG
```

LONG 타입은 다음과 같은 특징이 있다.

- 최대 2GB까지 선언할 수 있다.
- 테이블 내의 한 컬럼에만 선언할 수 있다.
- LONG 타입의 컬럼에 대해서는 인덱스를 생성할 수 없다.
- LONG 타입의 컬럼을 포함한 로우(Row)가 디스크에 저장될 때에는 다른 컬럼의 값과 함께 동일한 디스크 블록에 저장되며, 길이에 따라 여러 디스크 블록에 걸쳐 저장될 수 있다.
- LONG 타입의 데이터에 접근할 때는 항상 순차적으로만 접근할 수 있으며, 임의의 위치에 대해 연산은 할 수 없다.

LONG RAW

LONG RAW 타입은 RAW 타입을 확장한 데이터 타입이다. RAW 타입과 마찬가지로 바이너리 데이터가 저장된다.

```
LONG RAW
```

LONG RAW 타입은 다음과 같은 특징이 있다.

- 최대 2GB까지 선언할 수 있다.
- 테이블 내의 한 컬럼에만 선언할 수 있다.
- LONG RAW 타입의 컬럼에 대해서는 인덱스를 생성할 수 없다.
- LONG RAW 타입의 컬럼을 포함한 로우가 디스크에 저장될 때에는 다른 컬럼의 값과 함께 동일한 디스크 블록에 저장되며, 길이에 따라 여러 디스크 블록에 걸쳐 저장될 수 있다.
- LONG RAW 타입의 데이터에 접근할 때는 항상 순차적으로만 접근할 수 있으며, 임의의 위치에 대해 연산은 할 수 없다.

2.1.2. 숫자형

숫자형은 정수나 실수의 숫자를 저장하는 데이터 타입이다. 숫자형에는 NUMBER 타입, INTEGER 타입, FLOAT 타입, BINARY_FLOAT 타입, BINARY_DOUBLE 타입이 있다.

Tibero에서는 ANSI에서 제정한 SQL 표준의 숫자 타입의 선언을 지원한다. 즉, INTEGER 타입 또는 FLOAT 타입으로 컬럼을 선언하더라도 내부적으로 적절한 정밀도와 스케일을 설정하여 NUMBER 타입으로 변환해 준다.

참고

본 안내서에서는 INTEGER 타입, FLOAT 타입에 대한 내용은 별도로 기술하지 않는다.

NUMBER

NUMBER 타입은 정수 또는 실수를 저장하는 데이터 타입이다.

NUMBER 타입이 표현할 수 있는 수의 범위는 음양으로 절댓값이 1.0×10^{-130}보다 크거나 같고, 1.0×10^{126}보다 작은 38자리의 수를 표현할 수 있으며 0과 ±무한대를 포함한다.

NUMBER 타입은 선언할 때 다음과 같이 자릿수를 의미하는 정밀도와 스케일을 함께 정의할 수 있다.

```
NUMBER[(precision[,scale])]
```

구분	설명
precision	정밀도는 유효숫자의 최대 자릿수이다. – 가장 왼쪽의 0이 아닌 숫자부터 가장 오른쪽의 신뢰할 수 있는 숫자까지의 자리 수를 의미한다. – 정밀도를 초과하는 자릿수의 데이터는 저장할 수 없다. – 정밀도는 1 ~ 38까지 정의할 수 있다. 정밀도는 애스터리스크(*)로도 선언할 수 있다. – 정밀도를 38로 선언한 것을 의미하며, 이 경우 스케일 값을 함께 선언한다.
scale	스케일은 소수점 아래 가장 오른쪽 유효숫자까지의 자릿수이다. – 스케일은 생략할 수 있으며, 0으로 선언한 것과 동일하다. 이 경우 정수를 표현한다. – 마이너스의 스케일은 소수점 위의 자릿수이다. – 스케일을 초과하는 데이터는 반올림을 수행한다. – 스케일은 -125 ~ 130까지 정의할 수 있다.

정밀도와 스케일을 생략하여 선언한 경우 표현 가능한 최대 범위와 최대 정밀도 내에서 임의의 자릿수를 갖는 모든 데이터 값을 지원한다. 최대 정밀도 38을 초과하는 데이터는 반올림을 수행한다.

다음은 입력된 데이터가 NUMBER 타입의 정밀도와 스케일에 정의된 값에 따라 실제 데이터베이스에 어떤 형태로 저장되는지를 보여준다.

입력된 데이터	NUMBER 타입 선언	실제 저장된 데이터
12,345.678	NUMBER	12,345.678
12,345.678	NUMBER(*,3)	12,345.678
12,345.678	NUMBER(8,3)	12,345.678

입력된 데이터	NUMBER 타입 선언	실제 저장된 데이터
12,345.678	NUMBER(8,2)	12,345.68
12,345.678	NUMBER(8)	12,346
12,345.678	NUMBER(8,-2)	12,300
12,345.678	NUMBER(3)	입력된 데이터의 자릿수가 정밀도를 초과했으므로 저장할 수 없다.

BINARY_FLOAT 타입

BINARY_FLOAT 타입은 실수나 정수를 표현하고, 32비트로 저장하는 단일 정밀도 데이터 타입이다.

BINARY_FLOAT 타입은 음양으로 절댓값이 1.17549E-38보다 크거나 같고, 3.40282E+38보다 작은 수를 표현할 수 있다.

BINARY_FLOAT 타입은 다음과 같은 특징이 있다.

- 특별값인 INF, -INF, NaN(Not A Number)을 지원한다.
- 사칙연산을 모두 지원한다.
- 비교 연산자를 지원한다. 단, NaN은 다른 모든 값보다 가장 큰 값으로 취급하고, 서로 다른 NaN과 NaN은 같다.
- 각종 변환함수 및 수학함수를 지원한다.

BINARY_DOUBLE 타입

BINARY_DOUBLE 타입은 실수나 정수를 표현하고, 64비트로 저장하는 2배 정밀도 데이터 타입이다.

BINARY_DOUBLE 타입은 음양으로 절댓값이 2.22507485850720E-308보다 크거나 같고, 1.79769313486231E+308보다 작은 수를 표현할 수 있다.

BINARY_DOUBLE 타입은 다음과 같은 특징이 있다.

- 특별값인 INF, -INF, NaN(Not A Number)을 지원한다.
- 사칙연산을 모두 지원한다.
- 비교 연산자를 지원한다. 단, NaN은 다른 모든 값보다 가장 큰 값으로 취급하고, 서로 다른 NaN과 NaN은 같다.
- 각종 변환함수 및 수학함수를 지원한다.

숫자형 타입의 우선순위

Tibero에서는 서로 다른 숫자형 타입을 사용하는 연산에서 각 연산자를 어떤 데이타 타입으로 변환할 지를 우선순위에 따라 결정한다. 세 타입 중에서 BINARY_DOUBLE이 가장 높은 우선순위를 가지고, 다음으로 BINARY_FLOAT, 마지막으로 NUMBER 순이다.

- 연산자 중 하나라도 BINARY_DOUBLE일 경우엔 모든 연산자를 BINARY_DOUBLE로 변환한다.
- 연산자 중 BINARY_DOUBLE이 없고, 하나라도 BINARY_FLOAT일 경우엔 모든 연산자를 BINARY_FLOAT으로 변환한다.
- BINARY_FLOAT이나 BINARY_DOUBLE이 없는 경우엔 모든 연산자를 NUMBER로 변환한다.

다른 데이터 타입과의 관계에서는 날짜나 간격형 타입보다는 낮은 우선순위를 가지고, 나머지 모든 타입보다는 높은 우선순위를 가진다.

2.1.3. 날짜형

날짜형은 시간이나 날짜를 저장하는 데이터 타입이다. 날짜형에는 DATE 타입, TIME 타입, TIMESTAMP, TIMESTAMP WITH TIME ZONE, TIMESTAMP WITH LOCAL TIME ZONE 타입이 있다.

DATE

DATE 타입은 특정 날짜와 초 단위까지의 시간을 표현하는 데이터 타입이다.

```
DATE
```

DATE 타입은 다음과 같은 특징이 있다.

- 연도, 월, 일, 시, 분, 초를 표현할 수 있다.
- 연도는 BC 9,999 ~ AD 9,999까지 표현할 수 있다.
- 시간은 24시간 단위로 표현된다.

TIME

TIME 타입은 초 단위 소수점 9자리까지의 특정 시간을 표현하는 데이터 타입이다.

```
TIME [(fractional_seconds_precision)]
```

항목	설명
fractional_seconds_precision	초 단위의 소수점 자릿수이다. 0~9사이의 값을 사용할 수 있다. (기본값: 6)

TIME 타입은 다음과 같은 특징이 있다.

- 시, 분, 초, 10^{-9}초를 표현할 수 있다.
- 시간은 24시간 단위로 표현된다.

TIMESTAMP

TIMESTAMP 타입은 날짜와 초 단위 소수점 9자리까지의 시간을 모두 표현하는 데이터 타입이다.

```
TIMESTAMP [(fractional_seconds_precision)]
```

항목	설명
fractional_seconds_precision	초 단위의 소수점 자릿수를 말한다. 0~9사이의 값을 사용할 수 있다. (기본값: 6)

TIMESTAMP 타입은 다음과 같은 특징이 있다.

- 연도, 월, 일, 시, 분, 초, 10^{-9}초를 표현할 수 있다.
- 연도는 BC 9,999 ~ AD 9,999까지 표현 가능하다.
- 시간은 24시간 단위로 표현된다.

TIMESTAMP WITH TIME ZONE

TIMESTAMP WITH TIME ZONE 타입은 TIMESTAMP 타입을 확장하여 시간대까지 표현하는 데이터 타입이다.

```
TIMESTAMP [(fractional_seconds_precision)] WITH TIME ZONE
```

항목	설명
fractional_seconds_precision	초 단위의 소수점 자릿수를 말한다. 0~9사이의 값을 사용할 수 있다. (기본값: 6)

TIMESTAMP WITH TIME ZONE 타입은 다음과 같은 특징이 있다.

- 연도, 월, 일, 시, 분, 초, 10^{-9}초 등은 TIMESTAMP 타입과 동일한 특징을 가진다.
- 각 시간 요소들을 UTC(Coordinated Universal Time) 시간으로 정규화해서 저장한다.

- 지역 이름이나 오프셋으로 표현된 시간대를 포함하여 저장한다. 여기서 오프셋은 현재 지역의 시간과 UTC 시간과의 차이를 말한다.

TIMESTAMP WITH LOCAL TIME ZONE

TIMESTAMP WITH LOCAL TIME ZONE 타입은 특정 세션의 시간대에 따라 다르게 시간정보를 표현하는 데이터 타입이다.

```
TIMESTAMP [(fractional_seconds_precision)] WITH LOCAL TIME ZONE
```

항목	설명
fractional_seconds_precision	초 단위의 소수점 자릿수이다. 0~9사이의 값을 사용할 수 있다. (기본값: 6)

TIMESTAMP WITH LOCAL TIME ZONE 타입은 다음과 같은 특징이 있다.

- 연도, 월, 일, 시, 분, 초, 10^{-9}초 등은 TIMESTAMP 타입과 동일한 특징을 가진다.
- 각 시간 요소들을 UTC(Coordinated Universal Time) 시간으로 정규화해서 저장한다.
- TIMESTAMP WITH TIME ZONE 타입과 달리 지역이름이나 오프셋을 저장하지 않고, 사용자에 의해 조회될 때 세션의 시간대로 자동으로 바뀌어 반환된다.

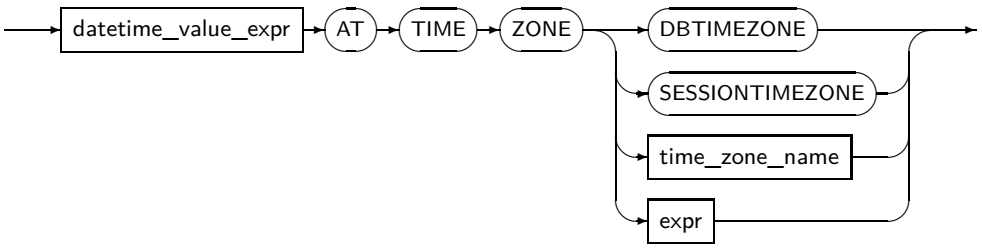
AT TIME ZONE 절을 사용하여 TIMEZONE 변환

AT TIME ZONE 절을 사용하여 날짜형 데이터 타입의 시간대를 변환할 수있다.

AT TIME ZONE 절을 사용하는 경우 해당 절과 함께 다음 중 하나를 설정할 수 있다.

- 문법

datetime_expression



- 구성요소

구성요소	설명
datetime_value_expr	datetime_value_expr에는 날짜형 데이터 타입 컬럼 혹은 날짜형 데이터 타입의 표현식이 올 수있다. 현재 TIMESTAMP 혹은 TIMESTAMP WITH TIME ZONE 날짜형 데이터 타입만 허용된다.
DBTIMEZONE	Tibero에 설정된 데이터베이스 시간대를 사용한다.
SESSIONTIMEZONE	현재 세션의 시간대를 사용한다.
time_zone_name	명시한 표준 시간대를 사용한다.
expr	표현식이 유효한 시간대 형식의 문자열인 경우 해당 시간대를 사용한다.

- 예제

  다음은 AT TIME ZONE 절에 time_zone_name을 명시하여 시간대 변환하는 예이다.

```
SQL> SELECT TO_TIMESTAMP('20190908','YYYYMMDD') AT TIME ZONE ('UTC') FROM DUAL;

TO_TIMESTAMP('20190908','YYYYMMDD')ATTIMEZONE('UTC')
---------------------------------------------------------------------------
2019/09/07 15:00:00.000000 UTC
```

2.1.4. 간격형

간격형은 시간이나 날짜 사이의 간격을 저장하는 데이터 타입이다. 간격형에는 INTERVAL YEAR TO MONTH 타입, INTERVAL DAY TO SECOND 타입이 있다.

INTERVAL YEAR TO MONTH

INTERVAL YEAR TO MONTH 타입은 연도와 월을 이용하여 시간 간격을 표현하는 데이터 타입이다.

```
INTERVAL YEAR [(year_precision)] TO MONTH
```

항목	설명
year_precision	연도 단위의 자릿수이다. (기본값: 2)

INTERVAL DAY TO SECOND

INTERVAL DAY TO SECOND 타입은 일, 시, 분, 초를 이용하여 시간 간격을 표현하는 데이터 타입이다.

```
INTERVAL DAY [(day_precision)] TO SECOND [(fractional_seconds_precision)]
```

항목	설명
day_precision	일 단위의 자릿수이다. (기본값: 2)

항목	설명
fractional_seconds_precision	초 단위의 소수점 자릿수이다. 0~9사이의 값을 사용할 수 있다. (기본값: 6)

2.1.5. 대용량 객체형

대용량 객체형은 말 그대로 대용량의 객체를 저장하기 위해 Tibero에서 제공하는 가장 큰 데이터 타입이며, CLOB 타입과 BLOB 타입, XMLTYPE, JSON 타입이 있다.

CLOB

CLOB 타입은 LONG 타입을 확장한 데이터 타입이다.

CLOB 타입은 다음과 같은 특징이 있다.

- 데이터를 최대 4GB까지 저장할 수 있다.
- 테이블 내에서 하나 이상의 컬럼에 선언할 수 있다.
- 데이터에 접근할 때 LONG 타입과 달리 임의의 위치에서 접근할 수 있다.
- CLOB 타입의 컬럼 값은 같은 테이블의 다른 타입으로 선언된 컬럼 값과 동일한 디스크 블록에 저장되지 않는다. 디스크 블록 내의 로우는 별도의 디스크 블록에 저장된 CLOB 타입의 포인터만 저장하고 있다.

BLOB

BLOB 타입은 LONG RAW 타입을 확장한 데이터 타입이며, 특징은 CLOB 타입과 유사하다.

BLOB 타입은 다음과 같은 특징이 있다.

- 데이터를 최대 4GB까지 저장할 수 있다.
- 테이블 내에서 하나 이상의 컬럼에 선언할 수 있다.
- 데이터에 접근할 때 LONG RAW 타입과 달리 임의의 위치에서 접근할 수 있다.
- BLOB 타입의 컬럼 값은 같은 테이블의 다른 타입으로 선언된 컬럼 값과 동일한 디스크 블록에 저장되지 않는다. 디스크 블록 내의 로우는 별도의 디스크 블록에 저장된 BLOB 타입의 포인터만 저장하고 있다.

XMLTYPE

XML(Extensible Markup Language)은 구조화되거나 그렇지 않은 모든 데이터를 표현하기 위해 W3C(World Wide Web Consortium)에 의해 표준으로 제정된 형식이다. Tibero에서는 이 XML 데이터를 저장하기 위해 XMLTYPE 타입을 제공하며, 내부적으로 CLOB 형식으로 저장된다.

XMLTYPE 타입은 다음과 같은 특징이 있다.

- 데이터를 CLOB의 최대 크기까지 저장할 수 있다.
- 테이블 내에서 하나 이상의 컬럼에 선언할 수 있다.
- XML 데이터에 대한 접근, 추출, 질의를 수행할 때 사용한다.

JSON

JSON(JavaScript Object Notation)은 KEY/VALUE 쌍으로 이루어진 데이터 오브젝트를 전달하기 위해 사용하는 개방형 표준 포맷이다. Tibero에서는 이 JSON 데이터를 저장 및 처리하기 위해 JSON 타입을 제공하며, 내부적으로 BLOB 형식으로 저장된다.

JSON 타입은 다음과 같은 특징이 있다.

- 데이터를 BLOB의 최대 크기까지 저장할 수 있다.
- 테이블 내에서 하나 이상의 컬럼에 선언할 수 있다.
- JSON 데이터에 대한 접근, 추출, 질의를 수행할 때 사용한다.

2.1.6. 내재형

내재형은 사용자가 명시적으로 선언하지 않아도 Tibero가 삽입되는 로우마다 자동으로 부여하는 데이터 타입이다. 내재형에는 ROWID 타입이 있다.

ROWID

ROWID는 데이터베이스의 각 로우를 식별하기 위해, Tibero가 각 로우마다 자동으로 부여하는 데이터 타입이다. 각 로우가 저장된 물리적인 위치를 포함하고 있다. ROWID에 대한 자세한 내용은 “2.5. 의사 컬럼”을 참고한다.

```
ROWID
```

2.1.7. 사용자 정의형

사용자 정의형은 사용자가 정의하여 사용하는 콜렉션 형태의 타입이다. tbPSM 타입으로 사용가능하며, Tibero 서버에서 저장된다. 사용자 정의형에는 배열과 네스티드 테이블이 있다. 자세한 내용은 "Tibero tbPSM 안내서"를 참고한다.

배열

배열는 최대 배열 길이를 가지는 동일한 타입을 구성요소로 갖는 콜렉션 형태의 사용자 정의 타입이다.

네스티드 테이블

네스티드 테이블(Nested Table)는 최대 배열 길이가 없는 동일한 타입을 구성요소로 갖는 콜렉션 형태의 사용자 정의 타입이다.

2.2. 데이터 타입 변환

다음은 Tibero에서 비교, 연산 등을 위하여 데이터 타입을 어떻게 변환하는지 살펴본다. 데이터 타입은 사용자가 명시적으로 할 수도 있고, 필요에 의하여 암시적으로 이루어지기도 한다.

2.2.1. 명시적 타입 변환

사용자가 SQL 변환 함수를 직접 사용하여 타입을 변환할 수 있다.

다음은 타입 변환 함수목록이다(행은 변환 전 타입, 열은 변환 후 타입이다).

[표 2.1] 명시적 타입 변환 (1)

from \ to	**CHAR,** **VARCHAR2,** **NCHAR,** **NVARCHAR2**	**NUMBER**	**날짜, 시간 및 인터벌**
CHAR, **VARCHAR2,** **NCHAR,** **NVARCHAR2**	TO_CHAR, TO_NCHAR	TO_NUMBER	TO_DATE, TO_TIME, TO_TIMESTAMP, TO_TIMESTAMP_TZ, TO_DSINTERVAL,

from \ to	CHAR, VARCHAR2, NCHAR, NVARCHAR2	NUMBER	날짜, 시간 및 인터벌
			TO_YMINTERVAL
NUMBER	TO_CHAR, TO_NCHAR	-	NUMTOYMINTERVAL, NUMTODSINTERVAL
날짜, 시간 및 인터벌	TO_CHAR, TO_NCHAR	X	-
RAW	RAWTOHEX	X	X
ROWID	ROWIDTOCHAR	X	X
LONG, LONG RAW	LONG_TO_CHAR	X	X
CLOB, NCLOB, BLOB	TO_CHAR, TO_NCHAR	X	X
BINARY_FLOAT	TO_CHAR, TO_NCHAR	TO_NUMBER	X
BINARY_DOUBLE	TO_CHAR, TO_NCHAR	TO_NUMBER	X
JSON	TO_CHAR	X	X

[표 2.2] 명시적 타입 변환 (2)

from \ to	BINARY_DOUBLE	RAW	ROWID	LONG, LONG RAW
CHAR, VARCHAR2, NCHAR,	TO_BINARY_DOUBLE	HEXTORAW	CHARTOROWID	LONG_TO_CHAR

from \ to	BINARY_DOUBLE	RAW	ROWID	LONG, LONG RAW
NVARCHAR2				
NUMBER	TO_BINARY_DOUBLE	X	X	X
날짜, 시간 및 인터벌	X	X	X	X
RAW	-	-	X	X
ROWID	X	X	-	X
LONG, LONG RAW	X	X	X	-
CLOB, NCLOB, BLOB	X	X	X	X
BINARY_FLOAT	X	X	X	X
BINARY_DOUBLE	X	X	X	X
JSON	X	X	X	X

[표 2.3] 명시적 타입 변환 (3)

from \ to	CLOB, NCLOB, BLOB	BINARY_FLOAT	BINARY_DOUBLE	JSON
CHAR, VARCHAR2, NCHAR, NVARCHAR2	TO_CLOB	TO_BINARY_FLOAT	TO_BINARY_DOUBLE	X
NUMBER	X	TO_BINARY_FLOAT	TO_BINARY_DOUBLE	X
날짜, 시간 및 인터벌	X	X	X	X
RAW	TO_BLOB	X	X	X

from \ to	CLOB, NCLOB, BLOB	BINARY_FLOAT	BINARY_DOUBLE	JSON
ROWID	X	X	X	X
LONG, **LONG RAW**	TO_LOB	X	X	X
CLOB, **NCLOB,** **BLOB**	TO_CLOB	X	X	X
BINARY_FLOAT	X	TO_BINARY_FLOAT	TO_BINARY_DOUBLE	X
BINARY_DOUBLE	X	TO_BINARY_FLOAT	TO_BINARY_DOUBLE	X
JSON	X	X	X	-

2.2.2. 암시적 타입 변환

사용자가 명시적으로 타입을 변환하지 않더라도, 필요하다면 암시적으로 타입을 변환하여 준다.

암시적 타입 변환이 필요한 경우는 아래와 같다.

- 컬럼에 다른 타입의 데이터를 INSERT, UPDATE하는 경우
- 조건문에서 비교하는 양쪽 값이 다른 타입인 경우

다음은 암시적 타입 변환 관계이다(행은 변환 전 타입, 열은 변환 후 타입이다).

[표 2.4] 암시적 타입 변환 (1)

	NUMBER	CHAR	VARCHAR2	RAW	DATE	TIME	TIMESTAMP
NUMBER	-	O	O	X	X	X	X
CHAR	O	-	O	O	O	O	O
VARCHAR2	O	O	-	O	O	O	O
RAW	X	O	O	-	X	X	X
DATE	X	O	O	X	-	X	O

	NUMBER	CHAR	VARCHAR2	RAW	DATE	TIME	TIMESTAMP
TIME	X	O	O	X	X	-	X
TIMESTAMP	X	O	O	X	O	X	-
INTERVAL YEAR TO MONTH	X	O	O	X	X	X	X
INTERVAL DAY TO SECOND	X	O	O	X	X	X	X
LONG	X	X	X	X	X	X	X
LONG RAW	X	X	X	X	X	X	X
BLOB	X	X	X	O	X	X	X
CLOB	X	O	O	X	X	X	X
ROWID	X	O	O	X	X	X	X
NCHAR	O	O	O	O	O	O	O
NVARCHAR2	O	O	O	O	O	O	O
NCLOB	X	O	O	X	X	X	X
TIMESTAMP WITH TIMEZONE	X	O	O	X	O	X	O
TIMESTAMP WITH LOCAL TIMEZONE	X	O	O	X	O	X	O
BINARY_FLOAT	O	O	O	X	X	X	X
BINARY_DOUBLE	O	O	O	X	X	X	X
JSON	X	O	O	X	X	X	X

[표 2.5] 암시적 타입 변환 (2)

	INTERVAL YEAR TO MONTH	INTERVAL DAY TO SECOND	LONG	LONG RAW	BLOB	CLOB	ROWID
NUMBER	X	X	O	X	X	O	X
CHAR	O	O	O	O	O	O	O
VARCHAR2	O	O	O	O	O	O	O
RAW	X	X	O	O	O	O	X
DATE	X	X	O	X	X	X	X
TIME	X	X	O	X	X	X	X
TIMESTAMP	X	X	O	X	X	X	X

	INTERVAL YEAR TO MONTH	INTERVAL DAY TO SECOND	LONG	LONG RAW	BLOB	CLOB	ROWID
INTERVAL YEAR TO MONTH	-	X	O	X	X	X	X
INTERVAL DAY TO SECOND	X	-	O	X	X	X	X
LONG	X	X	-	X	X	X	X
LONG RAW	X	X	X	-	X	X	X
BLOB	X	X	X	O	-	X	X
CLOB	X	X	O	X	X	-	X
ROWID	X	X	O	X	X	X	-
NCHAR	O	O	O	O	X	O	O
NVARCHAR2	O	O	O	O	X	O	O
NCLOB	X	X	O	X	X	O	X
TIMESTAMP WITH TIMEZONE	X	X	O	X	X	X	X
TIMESTAMP WITH LOCAL TIMEZONE	X	X	O	X	X	X	X
BINARY_FLOAT	X	X	O	X	X	X	X
BINARY_DOUBLE	X	X	O	X	X	X	X
JSON	X	X	X	X	O	O	X

[표 2.6] 암시적 타입 변환 (3)

	NCHAR	NVARCHAR2	NCLOB	TIMESTAMP WITH TIMEZONE	TIMESTAMP WITH LOCAL TIMEZONE	BINARY_FLOAT	BINARY_DOUBLE	JSON
NUMBER	O	O	X	X	X	O	O	X
CHAR	O	O	O	O	O	O	O	O
VARCHAR2	O	O	O	O	O	O	O	O
RAW	O	O	X	X	X	X	X	X
DATE	O	O	X	O	O	X	X	X

	NCHAR	NVARCHAR2	NCLOB	TIMESTAMP WITH TIME ZONE	TIMESTAMP WITH LOCAL TIME ZONE	BINARY_FLOAT	BINARY_DOUBLE	JSON
TIME	O	O	X	X	X	X	X	X
TIMESTAMP	O	O	X	O	O	X	X	X
INTERVAL YEAR TO MONTH	O	O	X	X	X	X	X	X
INTERVAL DAY TO SECOND	O	O	X	X	X	X	X	X
LONG	X	X	X	X	X	X	X	X
LONG RAW	X	X	X	X	X	X	X	X
BLOB	X	X	X	X	X	X	X	O
CLOB	O	O	O	X	X	X	X	O
ROWID	O	O	X	X	X	X	X	X
NCHAR	-	O	O	O	O	O	O	O
NVARCHAR2	O	-	O	O	O	O	O	O
NCLOB	O	O	-	X	X	X	X	O
TIMESTAMP WITH TIMEZONE	O	O	X	-	O	X	X	X
TIMESTAMP WITH LOCAL TIMEZONE	O	O	X	O	-	X	X	X
BINARY_FLOAT	O	O	X	X	X	-	O	X
BINARY_DOUBLE	O	O	X	X	X	O	-	X
JSON	X	X	X	X	X	X	X	-

타입비교

다음은 두 값에 대한 타입을 비교할 때 어느쪽 타입으로 맞춰서 비교하는지를 나타내는 표이다.

[표 2.7] 타입비교 (1)

	NUMBER	CHAR	VARCHAR2	RAW
NUMBER	NUMBER	NUMBER	NUMBER	VARCHAR2
CHAR	NUMBER	CHAR	VARCHAR2	CHAR

	NUMBER	CHAR	VARCHAR2	RAW
VARCHAR2	NUMBER	VARCHAR2	VARCHAR2	VARCHAR2
RAW	VARCHAR2	CHAR	VARCHAR2	RAW
DATE	X	DATE	DATE	VARCHAR2
TIME	X	TIME	TIME	VARCHAR2
TIMESTAMP	VARCHAR2	TIMESTAMP	TIMESTAMP	VARCHAR2
INTERVAL YEAR TO MONTH	VARCHAR2	INTERVAL YEAR TO MONTH	INTERVAL YEAR TO MONTH	VARCHAR2
INTERVAL DAY TO SECOND	VARCHAR2	INTERVAL DAY TO SECOND	INTERVAL DAY TO SECOND	VARCHAR2
LONG	X	LONG	LONG	VARCHAR2
LONG RAW	VARCHAR2	LONG	LONG	VARCHAR2
BLOB	X	X	X	X
CLOB	X	CLOB	CLOB	CLOB
ROWID	VARCHAR2	ROWID	ROWID	VARCHAR2
NCHAR	NUMBER	NCHAR	NVARCHAR2	NCHAR
NVARCHAR2	NUMBER	NVARCHAR2	NVARCHAR2	NVARCHAR2
NCLOB	X	NCLOB	NCLOB	NCLOB
TIMESTAMP WITH TIMEZONE	VARCHAR2	TIMESTAMP	TIMESTAMP	VARCHAR2
TIMESTAMP WITH LOCAL TIMEZONE	VARCHAR2	TIMESTAMP	TIMESTAMP	VARCHAR2
BINARY_FLOAT	BINARY_FLOAT	BINARY_FLOAT	BINARY_FLOAT	VARCHAR2
BINARY_DOUBLE	BINARY_DOUBLE	BINARY_DOUBLE	BINARY_DOUBLE	VARCHAR2
JSON	X	X	X	X

[표 2.8] 타입비교 (2)

	DATE	TIME	TIMESTAMP	INTERVAL YEAR TO MONTH
NUMBER	X	X	VARCHAR2	VARCHAR2
CHAR	DATE	TIME	TIMESTAMP	INTERVAL YEAR TO MONTH
VARCHAR2	DATE	TIME	TIMESTAMP	INTERVAL YEAR TO MONTH
RAW	VARCHAR2	VARCHAR2	VARCHAR2	VARCHAR2

	DATE	**TIME**	**TIMESTAMP**	**INTERVAL YEAR TO MONTH**
DATE	DATE	VARCHAR2	TIMESTAMP	VARCHAR2
TIME	VARCHAR2	TIME	VARCHAR2	VARCHAR2
TIMESTAMP	TIMESTAMP	VARCHAR2	TIMESTAMP	VARCHAR2
INTERVAL YEAR TO MONTH	VARCHAR2	VARCHAR2	VARCHAR2	INTERVAL YEAR TO MONTH
INTERVAL DAY TO SECOND	VARCHAR2	VARCHAR2	VARCHAR2	INTERVAL DAY TO SECOND
LONG	X	X	X	X
LONG RAW	VARCHAR2	VARCHAR2	VARCHAR2	VARCHAR2
BLOB	X	X	X	X
CLOB	X	X	X	X
ROWID	VARCHAR2	VARCHAR2	VARCHAR2	VARCHAR2
NCHAR	DATE	TIME	TIMESTAMP	INTERVAL YEAR TO MONTH
NVARCHAR2	DATE	TIME	TIMESTAMP	INTERVAL YEAR TO MONTH
NCLOB	X	X	X	X
TIMESTAMP WITH TIMEZONE	TIMESTAMP	VARCHAR2	TIMESTAMP WITH TIMEZONE	VARCHAR2
TIMESTAMP WITH LOCAL TIMEZONE	TIMESTAMP	VARCHAR2	TIMESTAMP WITH LOCAL TIMEZONE	VARCHAR2
BINARY_FLOAT	X	X	VARCHAR2	VARCHAR2
BINARY_DOUBLE	X	X	VARCHAR2	VARCHAR2
JSON	X	X	X	X

[표 2.9] 타입비교 (3)

	INTERVAL DAY TO SECOND	**LONG**	**LONG RAW**	**BLOB**
NUMBER	VARCHAR2	X	VARCHAR2	X
CHAR	INTERVAL DAY TO SECOND	LONG	LONG	X
VARCHAR2	INTERVAL DAY TO SECOND	LONG	LONG	X
RAW	VARCHAR2	VARCHAR2	VARCHAR2	X

	INTERVAL DAY TO SECOND	LONG	LONG RAW	BLOB
DATE	VARCHAR2	X	VARCHAR2	X
TIME	VARCHAR2	X	VARCHAR2	X
TIMESTAMP	VARCHAR2	X	VARCHAR2	X
INTERVAL YEAR TO MONTH	VARCHAR2	X	VARCHAR2	X
INTERVAL DAY TO SECOND	VARCHAR2	X	VARCHAR2	X
LONG	X	LONG	LONG	X
LONG RAW	VARCHAR2	LONG	LONG RAW	X
BLOB	X	X	X	BLOB
CLOB	X	LONG	LONG	X
ROWID	VARCHAR2	ROWID	VARCHAR2	X
NCHAR	INTERVAL DAY TO SECOND	X	X	X
NVARCHAR2	INTERVAL DAY TO SECOND	X	X	X
NCLOB	X	X	X	X
TIMESTAMP WITH TIMEZONE	VARCHAR2	X	VARCHAR2	X
TIMESTAMP WITH LOCAL TIMEZONE	VARCHAR2	X	VARCHAR2	X
BINARY_FLOAT	VARCHAR2	X	VARCHAR2	X
BINARY_DOUBLE	VARCHAR2	X	VARCHAR2	X
JSON	X	X	X	X

[표 2.10] 타입비교 (4)

	CLOB	ROWID	NCHAR	NVARCHAR2	NCLOB
NUMBER	X	VARCHAR2	NUMBER	NUMBER	X
CHAR	CLOB	ROWID	NCHAR	NVARCHAR2	NCLOB
VARCHAR2	CLOB	ROWID	NCHAR	NVARCHAR2	NCLOB
RAW	CLOB	VARCHAR2	NCHAR	NVARCHAR2	NCLOB
DATE	X	VARCHAR2	DATE	DATE	X
TIME	X	VARCHAR2	TIME	TIME	X

	CLOB	ROWID	NCHAR	NVARCHAR2	NCLOB
TIMESTAMP	X	VARCHAR2	TIMESTAMP	TIMESTAMP	X
INTERVAL YEAR TO MONTH	X	VARCHAR2	INTERVAL YEAR TO MONTH	INTERVAL YEAR TO MONTH	X
INTERVAL DAY TO SECOND	X	VARCHAR2	INTERVAL DAY TO SECOND	INTERVAL DAY TO SECOND	X
LONG	LONG	ROWID	X	X	X
LONG RAW	LONG	VARCHAR2	X	X	X
BLOB	X	X	X	X	X
CLOB	CLOB	ROWID	CLOB	CLOB	NCLOB
ROWID	ROWID	ROWID	ROWID	ROWID	X
NCHAR	CLOB	ROWID	NCHAR	NVARCHAR2	NCLOB
NVARCHAR2	CLOB	ROWID	NCHAR	NVARCHAR2	NCLOB
NCLOB	NCLOB	X	NCLOB	NCLOB	NCLOB
TIMESTAMP WITH TIMEZONE	X	VARCHAR2	TIMESTAMP	TIMESTAMP	X
TIMESTAMP WITH LOCAL TIMEZONE	X	VARCHAR2	TIMESTAMP	TIMESTAMP	X
BINARY_FLOAT	X	VARCHAR2	BINARY_FLOAT	BINARY_FLOAT	X
BINARY_DOUBLE	X	VARCHAR2	BINARY_DOUBLE	BINARY_DOUBLE	X
JSON	X	X	X	X	X

[표 2.11] 타입비교 (5)

	TIMESTAMP WITH TIME ZONE	TIMESTAMP WITH LOCAL TIMEZONE	BINARY_FLOAT	BINARY_DOUBLE	JSON
NUMBER	VARCHAR2	VARCHAR2	BINARY_FLOAT	BINARY_DOUBLE	X
CHAR	TIMESTAMP WITH	TIMESTAMP WITH LOCAL TIMEZONE	BINARY_FLOAT	BINARY_DOUBLE	X

	TIMESTAMP WITH TIME ZONE	**TIMESTAMP WITH LOCAL TIMEZONE**	**BINARY_FLOAT**	**BINARY_DOUBLE**	**JSON**
	TIME ZONE				
VARCHAR2	TIMESTAMP WITH TIME ZONE	TIMESTAMP WITH LOCAL TIMEZONE	BINARY_FLOAT	BINARY_DOUBLE	X
RAW	NCLOB	NCLOB	NCLOB	NCLOB	X
DATE	TIMESTAMP WITH TIME ZONE	TIMESTAMP WITH LOCAL TIMEZONE	X	X	X
TIME	VARCHAR2	VARCHAR2	X	X	X
TIMESTAMP	TIMESTAMP WITH TIME ZONE	TIMESTAMP WITH LOCAL TIMEZONE	VARCHAR2	VARCHAR2	X
INTERVAL YEAR TO MONTH	VARCHAR2	VARCHAR2	VARCHAR2	VARCHAR2	X
INTERVAL DAY TO SECOND	VARCHAR2	VARCHAR2	VARCHAR2	VARCHAR2	X
LONG	X	X	X	X	X
LONG RAW	VARCHAR2	VARCHAR2	VARCHAR2	VARCHAR2	X
BLOB	X	X	X	X	X
CLOB	X	X	X	X	X
ROWID	VARCHAR2	VARCHAR2	VARCHAR2	VARCHAR2	X
NCHAR	TIMESTAMP WITH	TIMESTAMP WITH LOCAL TIMEZONE	BINARY_FLOAT	BINARY_DOUBLE	X

	TIMESTAMP WITH TIME ZONE	**TIMESTAMP WITH LOCAL TIMEZONE**	**BINARY_FLOAT**	**BINARY_DOUBLE**	**JSON**
	TIME ZONE				
NVARCHAR2	TIMESTAMP WITH TIME ZONE	TIMESTAMP WITH LOCAL TIMEZONE	BINARY_FLOAT	BINARY_DOUBLE	X
NCLOB	X	X	X	X	X
TIMESTAMP WITH TIMEZONE	TIMESTAMP WITH TIME ZONE	TIMESTAMP WITH TIME ZONE	VARCHAR2	VARCHAR2	X
TIMESTAMP WITH LOCAL TIMEZONE	TIMESTAMP WITH TIME ZONE	TIMESTAMP WITH LOCAL TIMEZONE	VARCHAR2	VARCHAR2	X
BINARY_FLOAT	VARCHAR2	VARCHAR2	BINARY_FLOAT	BINARY_DOUBLE	X
BINARY_DOUBLE	VARCHAR2	VARCHAR2	BINARY_DOUBLE	BINARY_DOUBLE	X
JSON	X	X	X	X	JSON

2.3. 리터럴

리터럴(Literal)은 상수 값을 나타내는 단어이다. 상수란 변수에 대응되는 개념으로 말 그대로 변하지 않는 값을 의미한다. 문자열 리터럴은 작은따옴표를 사용하여 다른 스키마 객체와 구분한다. 리터럴은 SQL 문장에서 연산식이나 조건식의 일부로 사용할 수 있다.

2.3.1. 문자열 리터럴

문자열 리터럴은 문자열을 표현할 때 사용하는 리터럴이다.

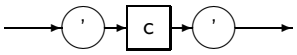
문자열 리터럴은 다음과 같은 특징이 있다.

- 최대 4,000byte까지 선언할 수 있다.
- 연산식이나 조건식에 문자열 리터럴이 사용되면, 문자열 리터럴은 CHAR 타입으로 취급된다.
- CHAR 타입의 데이터와 문자형 리터럴을 비교하는 경우 더 짧은 길이를 가진 데이터에 공백 문자를 삽입하여 비교한다.
- VARCHAR 타입의 데이터와 문자형 리터럴을 비교하는 경우 공백 문자를 삽입하지 않고 비교한다.

문자열 리터럴의 세부 내용은 다음과 같다.

- 문법

text



- 구성요소

구성요소	설명
c	사용자의 문자 집합에 속한 문자이다.
'	문자열 리터럴 안에서 이스케이프 부호(Escape Code)를 사용하려면 문자 양쪽에 작은따옴표(' ')를 붙여야 한다. 문자열 리터럴 안에 작은따옴표를 표현하려면 작은따옴표를 두 번 연속해서 사용해야 한다. 문자열 리터럴 안에 작은따옴표를 일반 문자 와 같이 취급하려면 q string을 사용하면 된다. q'[[string]]'$,q'"[string]"'$,q'|[string]|'$ q'$[string]'$ 으로 문자열 리터럴을 사용하면 [string] 내부의 작은따옴표는 일반 문자와 같이 취급된다.

- 예제

다음은 문자열 리터럴의 예이다.

```
'Tibero'
'Database'
'2009/11/11'
```

2.3.2. 숫자 리터럴

숫자 리터럴은 정수 또는 실수를 표현할 때 사용하는 리터럴이다.

숫자 리터럴은 다음과 같은 특징이 있다.

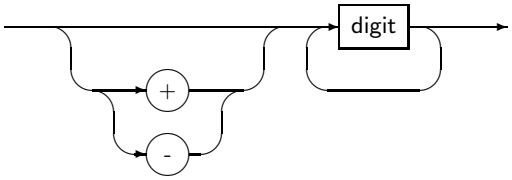
- 정수 리터럴과 실수 리터럴이 있다.
- NUMBER 타입이 표현할 수 있는 최대 38자리의 정밀도를 초과하는 경우 숫자 리터럴은 NUMBER 타입의 최대 정밀도에 맞춘 후 나머지 수를 반올림 처리한다. 또한, NUMBER 타입이 표현할 수 있는 범위를 넘어서는 숫자 리터럴이 입력되면 에러가 발생한다.
- 다음과 같은 부동 소수점 리터럴이 있다.

리터럴	**설명**
BINARY_FLOAT_NAN	단일 정밀도 NaN(Not A Number)
BINARY_FLOAT_INFINITY	단일 정밀도 양의 무한대
BINARY_DOUBLE_NAN	2배 정밀도 NaN(Not A Number)
BINARY_DOUBLE_INFINITY	2배 정밀도 양의 무한대

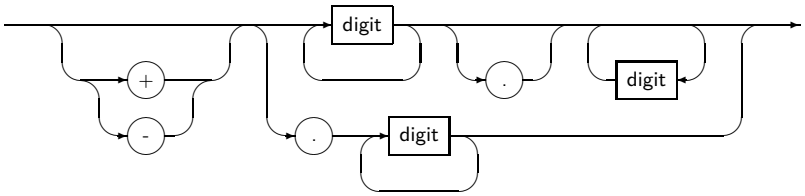
숫자 리터럴의 세부 내용은 다음과 같다.

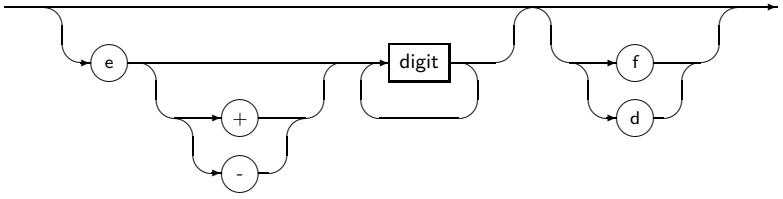
- 문법

integer



number





- 구성요소

구성요소	설명
digit	0 ~ 9 사이의 숫자 하나를 의미한다.
+ / -	숫자 앞의 부호(양수 또는 음수)를 의미한다.
.	소수점을 의미한다.
e, E	과학적 기수법(Scientific Notation)으로 표기할 때 사용한다. 예를 들어 8.33e-4는 0.000833을 8.33e+4는 83,300을 의미한다. e나 E 다음에 나오는 숫자는 지수를 나타낸다. 이때 지수는 -130 ~ 125 사이의 값이어야 한다.
f, F	BINARY_FLOAT 타입을 나타내고, 32비트 부동 소수점 숫자이다.
d, D	BINARY_DOUBLE 타입을 나타내고, 64비트 부동 소수점 숫자이다.

- 예제

  다음은 숫자 리터럴의 예이다.

```
123
+1.23
0.123
123e-123
-123
```

  다음은 부동 소수점 리터럴의 예이다.

```
123f
+1.23F
0.123d
-123D
```

2.3.3. 날짜형 리터럴

날짜형 리터럴은 날짜와 시간 정보를 표현하는 리터럴이다. 날짜형 리터럴에는 DATE 리터럴, TIME 리터럴, TIMESTAMP 리터럴, TIMESTAMP WITH TIME ZONE 리터럴이 있다.

DATE

DATE 리터럴은 날짜와 시간 정보를 표현하는 날짜형 리터럴이다.

DATE 리터럴은 다음과 같은 특징이 있다.

- 특별한 속성

  세기, 년, 월, 일, 시, 분, 초의 특별한 속성이 있다.

- 날짜형 리터럴 변환

  Tibero에서는 TO_DATE 함수를 사용하여 날짜 값을 직접 지정하거나 문자 리터럴이나 숫자 리터럴로 표현된 날짜 값을 DATE 리터럴로 변환할 수 있다. 이때 날짜를 리터럴로 표현하려면 그레고리안 달력을 사용한다.

  ```
  TO_DATE('2005/01/01 12:38:20', 'YY/MM/DD HH24:MI:SS')
  ```

  기본 날짜 형식은 'YYYY/MM/DD'이며 초기화 파라미터 파일에 NLS_DATE_FORMAT 파라미터로 정의되어 있다. NLS_DATE_FORMAT은 날짜 형식을 지정하는 파라미터이다.

  만일 시간 정보가 없는 값으로 DATE 리터럴을 표현해야 한다면 기본 시간은 자정 (HH24 00:00:00, HH 12:00:00)이다. 또한, 날짜 정보가 없는 값으로 DATE 리터럴을 표현해야 한다면 기본 날짜는 현재 시스템의 시간을 기준으로 달의 첫째 날로 지정된다.

  따라서 DATE 리터럴을 비교할 때는 리터럴의 시간정보가 포함된 에러인지 확인이 필요하다. 한쪽에만 시간 정보가 있고 다른 쪽에는 시간 정보가 없을 경우 두 날짜가 같다고 비교하기 위해서는 시간 정보를 제거하고 비교해야 하는데 이때 TRUNC 함수를 사용하면 된다.

  다음은 TRUNC 함수를 사용한 예이다.

  ```
  TO_DATE('2005/01/01', 'YY/MM/DD') =
        TRUNC(TO_DATE('2005/01/01 12:38:20', 'YY/MM/DD HH24:MI:SS'))
  ```

- ANSI 표현

  ```
  DATE '2005-01-01'
  ```

  – 시간 정보가 없다.

  – 기본 형식은 'YYYY-MM-DD'이다.

  – 구분자는 하이픈(-) 외에도 슬래시(/), 애스터리스크(*), 점(.) 등이 있다.

TIME

TIME 리터럴은 시간 정보를 표현하는 날짜형 리터럴이다.

TIME 리터럴은 다음과 같은 특징이 있다.

- 특별한 속성

  시, 분, 초, 소수점 아래의 초의 특별한 속성이 있다.

- 날짜형 리터럴 변환

  Tibero에서는 TO_TIME 함수를 사용하여 시간 값을 직접 지정하거나 문자 리터럴이나 숫자 리터럴로 표현된 시간 값을 TIME 리터럴로 변환할 수 있다.

```
TO_TIME('12:38:20.123456789', 'HH24:MI:SSXFF')
```

기본 시간 형식은 초기화 파라미터 파일에 NLS_TIME_FORMAT 파라미터로 정의되어 있다.

- ANSI 표현

```
TIME '10:23:10.123456789'
TIME '10:23:10'
TIME '10:23'
TIME '10'
```

  – 기본 형식은 'HH24:MI:SS.FF9'이다.

  – 분 이하는 생략할 수 있다.

TIMESTAMP

TIMESTAMP 리터럴은 DATE 리터럴을 확장한 날짜형 리터럴이다.

TIMESTAMP 리터럴은 다음과 같은 특징이 있다.

- 특별한 속성

  '년, 월, 일'의 날짜와 '시, 분 ,초', '소수점 아래의 초'의 특별한 속성이 있다.

- 날짜형 리터럴 변환

  Tibero에서는 TO_TIMESTAMP 함수를 사용하여 문자열 리터럴이나 숫자 리터럴로 표현된 날짜 값을 TIMESTAMP 리터럴로 변환할 수 있다.

```
TO_TIMESTAMP('09-Aug-01 12:07:15.50', 'DD-Mon-RR HH24:MI:SS.FF')
```

  기본 TIMESTAMP 형식은 초기화 파라미터 파일에 NLS_TIMESTAMP_FORMAT 파라미터로 정의되어 있다. NLS_TIMESTAMP_FORMAT은 TIMESTAMP 형식을 지정하는 파라미터이다.

- ANSI 표현

```
TIMESTAMP '2005/01/31 08:13:50.112'
TIMESTAMP '2005/01/31 08:13:50'
TIMESTAMP '2005/01/31 08:13'
TIMESTAMP '2005/01/31 08'
TIMESTAMP '2005/01/31'
```

  – 기본 형식은 'YYYY/MM/DD HH24:MI:SSxFF' 이다.

  – 날짜 부분('YYYY/MM/DD') 이외에는 생략할 수 있다.

  – 소수점 아래의 초('FF') 부분은 0~9자리까지 표현할 수 있다.

TIMESTAMP WITH TIME ZONE

TIMESTAMP WITH TIME ZONE 리터럴은 TIMESTAMP 리터럴을 확장한 날짜형 리터럴이다.

TIMESTAMP WITH TIME ZONE 리터럴은 다음과 같은 특징이 있다.

- 기본 속성

  TIMESTAMP 타입과 동일하게 '년, 월, 일'의 날짜와 '시, 분 ,초', '소수점 아래의 초'의 속성이 있다.

- 날짜형 리터럴 변환

  Tibero에서는 TO_TIMESTAMP_TZ 함수를 사용하여 문자열 리터럴이나 숫자 리터럴로 표현된 날짜 값을 TIMESTAMP WITH TIME ZONE 리터럴로 변환할 수 있다.

  ```
  TO_TIMESTAMP_TZ('2004-05-15 19:25:43 Asia/Seoul', 'YYYY-MM-DD HH24:MI:SS.FF TZR')
  TO_TIMESTAMP_TZ('1988-11-21 10:31:58.754 -07:30', 'YYYY-MM-DD HH24:MI:SS.FF TZH:
  TZM')
  ```

  기본 TIMESTAMP WITH TIME ZONE 형식은 초기화 파라미터 파일에 NLS_TIMESTAMP_TZ_FORMAT 파라미터로 정의되어 있다. NLS_TIMESTAMP_TZ_FORMAT은 TIMESTAMP WITH TIME ZONE 형식을 지정하는 파라미터이다.

- ANSI 표현

  ```
  TIMESTAMP '1993/12/11 13:37:43.27 Asia/Seoul'
  TIMESTAMP '1993/12/11 13:37:43.27 +09:00'
  TIMESTAMP '1993/12/11 13:37:43.27 +07'
  ```

  – 기본 형식은 'YYYY/MM/DD HH24:MI:SSXFF TZR' 이다.

  – 소수점 아래의 초('FF') 부분은 0~9자리까지 표현할 수 있다.

  – 시간대('TZR') 부분은 지역이름 또는 오프셋 형식으로 표현할 수 있다. 만약 시간대('TZR') 부분을 생략하는 경우엔 TIMESTAMP 타입 리터럴로 해석된다.

TIMESTAMP WITH LOCAL TIME ZONE

TIMESTAMP WITH LOCAL TIME ZONE 리터럴은 TIMESTAMP 리터럴과 동일한 형식을 사용한다.

2.3.4. 간격 리터럴

간격 리터럴(Interval literal) 은 특정 시간과 시간 사이의 간격을 표현하는 리터럴이다. 이러한 간격은 '연과 월'로 구성된 단위나 '날짜, 시간, 분, 초'로 구성된 단위 중 하나로 표현될 수 있다.

Tibero에서는 간격 리터럴을 다음과 같이 두 가지 타입으로 지원한다.

- YEAR TO MONTH

간격을 가장 가까운 월 단위로 표현하는 타입이다.

- DAY TO SECOND

  간격을 가장 가까운 분 단위로 표현하는 타입이다.

이처럼 각 타입의 리터럴은 첫 번째 필드와 생략 가능한 두 번째 필드로 구성된다. 첫 번째 필드는 표현할 날짜 또는 시간의 기본 단위를 정의하고, 두 번째 필드는 기본 단위의 최소 증가 단위를 나타낸다. 간격 리터럴은 같은 타입의 간격 리터럴끼리 서로 더하거나 뺄 수 있다.

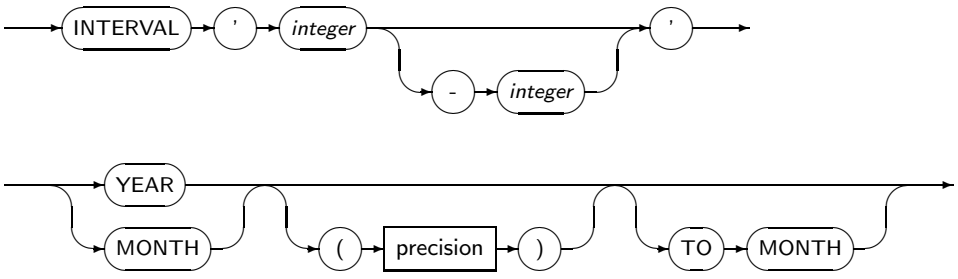
YEAR TO MONTH

YEAR TO MONTH 타입은 간격을 연과 월로 표현하는 간격 리터럴이다.

YEAR TO MONTH의 세부 내용은 다음과 같다.

- 문법

interval_year_to_month



- 구성 요소

구성 요소	설명
integer [-integer]	리터럴의 첫 번째와 생략 가능한 두 번째 필드의 값을 나타낸다. 첫 번째 필드가 YEAR이고, 두 번째 필드가 MONTH이면 두 번째 필드는 0과 11 사이의 값이어야 한다. 이때 두 번째 필드는 첫 번째 필드보다 더 작은 단위를 나타내야 한다. 즉, MONTH TO YEAR는 잘못된 표현이다.
precision	YEAR 단위의 최대 정밀도를 나타낸다. 0과 9 사이의 값이어야 하고, 기본값은 2이다.

- 예제

  다음은 YEAR TO MONTH의 예이다.

```
INTERVAL '12-3' YEAR TO MONTH
INTERVAL '123' YEAR(3)
```

```
INTERVAL '123' MONTH
INTERVAL '1' YEAR
INTERVAL '1234' MONTH(3)
```

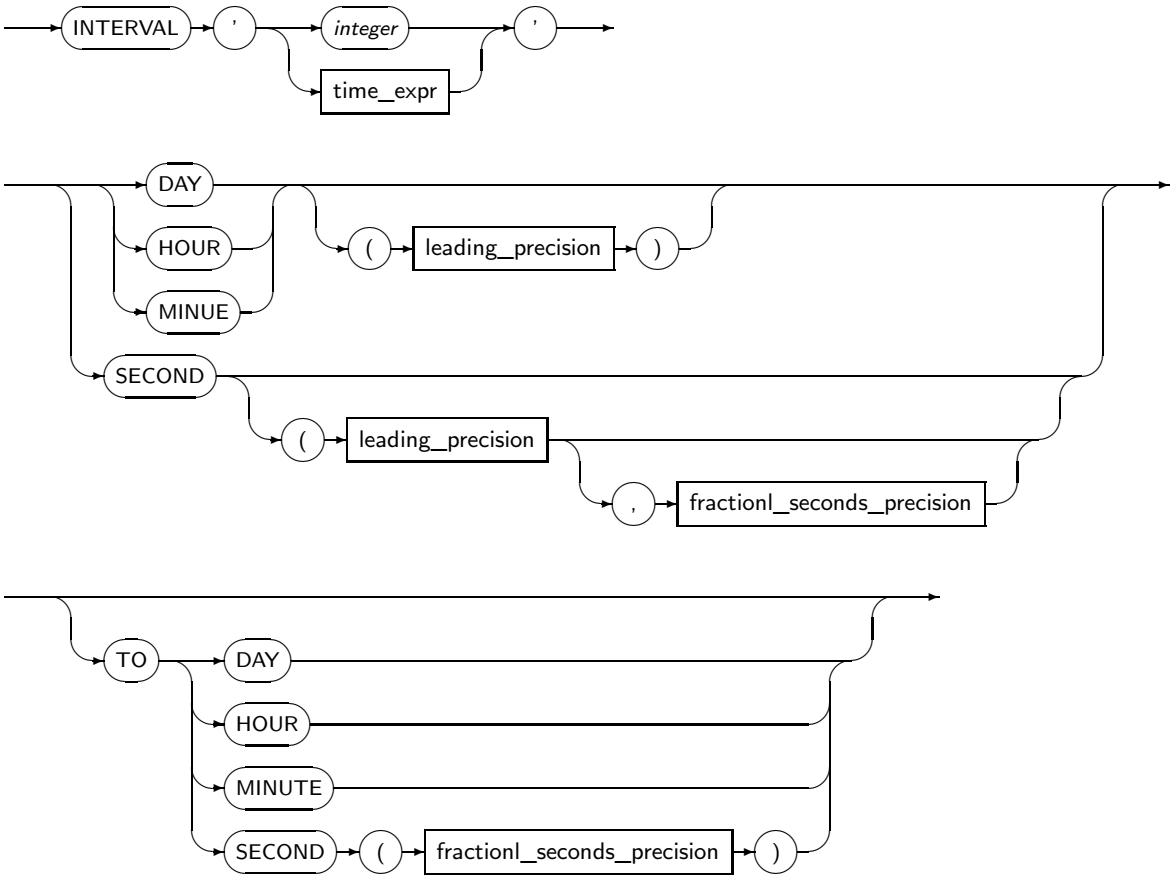
DAY TO SECOND

DAY TO SECOND 타입은 간격을 '날짜, 시간, 분, 초'로 표현하는 간격 리터럴이다.

DAY TO SECOND의 세부 내용은 다음과 같다.

- 문법

interval_day_to_second



- 구성요소

구성요소	설명
integer	일 수를 나타낸다.
time_expr	time_expr은 HH[:MI[:SS[.n]]] 또는 MI[:SS[.n]] 또는 SS[.n]과 같은 형식으로 나타낸다. n은 초의 소수점 아래 자리를 나타낸다. 만일 n이 frac

구성요소	설명
	tional_seconds_precision에 명시된 값보다 큰 자릿수의 값이면 n은 fractional_seconds_precision 값에 맞추어 반올림된다. 첫 번째 필드가 DAY인 경우에는 일 수를 표시하는 정수와 공백 문자 이후에 time_expr을 표시한다.
leading_precision	첫 번째 필드의 정밀도를 나타낸다. 0과 9 사이의 정수 값을 갖고, 기본값은 2이다.
fractional_seconds_precision	리터럴의 두 번째 필드의 정밀도를 나타낸다. 1과 9 사이의 값을 갖고, 기본값은 6이다. 이때 두 번째 필드는 첫 번째 필드보다 더 작은 단위를 나타내야 한다. 즉, MINUTE TO DAY는 잘못된 표현이다. 그리고 두 번째 필드가 HOUR, MINUTE 또는 SECOND일 경우에는 각각 0 ~ 23, 0 ~ 59, 그리고 0 ~ 59.999999999 사이의 값을 가져야 한다.

- 예제

  다음은 DAY TO SECOND의 예이다.

```
INTERVAL '1 2:3:4.567' DAY TO SECOND(3)
INTERVAL '1 2:3' DAY TO MINUTE
INTERVAL '123 4' DAY(3) TO HOUR
INTERVAL '123' DAY(3)
INTERVAL '12:34:56.1234567' HOUR TO SECOND(7)
INTERVAL '12:34' HOUR TO MINUTE
INTERVAL '12' HOUR
INTERVAL '12:34' MINUTE TO SECOND
INTERVAL '12' MINUTE
INTERVAL '12.345678' SECOND(2,6)
```

2.4. 형식 문자열

형식 문자열이란 NUMBER 타입과 날짜형 타입의 값을 문자열로 변환하기 위한 형식을 정의한 것이다.

형식 문자열은 문자열 타입으로 변환된 NUMBER 타입과 날짜형 타입의 값을 다시 원래의 타입의 값으로 변환하는 데 필요하다.

문자열 타입과 날짜형 타입, NUMBER 타입의 값 사이에는 데이터 타입의 변환을 할 수 있다. 하지만, 실제 값에 따라 변환이 불가능한 경우도 있다. 예를 들어 문자열 '12345'는 NUMBER 타입의 값으로 변환할 수 있지만, 문자열 'ABCDE'는 변환할 수 없다.

디폴트 시간 형식으로 되어 있지 않은 문자열이나 숫자 이외의 문자를 포함하는 문자열은 각각 날짜형 또는 NUMBER 타입의 값으로 변환할 수 없다. 이러한 경우 반드시 TO_DATE, TO_NUMBER 등의 변환 함수를 사용해야 한다.

형식 문자열은 TO_CHAR, TO_DATE, TO_NUMBER 함수의 파라미터로 사용된다. 만약 함수 파라미터로 형식 문자열이 주어지지 않으면, 디폴트 형식을 사용하여 변환한다. 함수에 대한 자세한 내용은 “제4장 함수”를 참고한다.

2.4.1. NUMBER 타입

NUMBER 타입의 형식 문자열은 TO_CHAR 함수와 TO_NUMBER 함수에서 파라미터로 사용할 수 있다.

함수	설명
TO_CHAR	NUMBER 타입의 값을 문자열로 변환한다.
TO_NUMBER	문자열을 NUMBER 타입의 값으로 변환한다.

NUMBER 타입의 형식 문자열은 다음과 같은 특징이 있다.

- 여러 가지 형식 요소로 구성된다. 소수점 위아래의 자릿수, 음양 부호의 출력, 콤마(,) 또는 지수 형식 등을 출력할 수 있다.
- 화폐 단위를 나타내는 기호($, W 등)를 삽입할 수 있다.
- 16진수로 출력할 수 있다.
- 별도의 문자열을 삽입할 수 없다.
- 대소문자를 구분하는 형식 요소가 없다.

다음은 NUMBER 타입의 형식 문자열에 포함될 수 있는 형식 요소이다.

형식 요소	예	설명
콤마 (,)	9,999	해당 위치에 콤마(,)를 찍는다. 콤마를 여러 개 찍을 수도 있다. 형식 문자열을 콤마로 시작할 수는 없다.
점 (.)	99.99	소수점을 출력한다. 형식 문자열 내에서는 하나의 소수점만 나올 수 있다.
$	$9999	숫자의 시작에 달러 문자($)를 출력한다.
0	0999 9990	숫자의 앞이나 뒤에 0을 채운다. 해당되는 위치에 0을 찍을 수 있는 경우에만 0이 출력되는 것을 보장한다.
9	9999	(자릿수 + 1)개의 문자를 사용해서 숫자를 출력한다.

형식 요소	예	설명
		플러스(+)나 마이너스(-)가 추가로 붙을 수 있다(양수일 때는 공백, 음수일 때는 마이너스가 출력된다). 숫자의 앞쪽에 올 수 있는 0은 출력하지 않는다. 다만, 소수점 없이 정수만을 출력할 때 정수 부분이 0일 경우는 0을 출력한다.
B	B9999	0의 값을 공백으로 출력한다.
D	99D99	해당 위치에 소수점을 출력한다. 현재는 점(.)과 기능이 같다.
EEEE	9.9EEEE	과학적 기수법에 의해 출력한다.
G	9G999	해당 위치에 콤마(,)를 찍는다. 현재는 콤마와 기능이 같다.
L 또는 U	L9999 U9999	숫자의 시작에 달러 문자($)를 출력한다. 현재는 $와 기능이 같다.
MI	9999MI	음수에 대해 마이너스를 뒤에 붙인다. 양수의 경우에는 공백을 출력한다. 형식 문자열의 맨 뒤에만 사용할 수 있다.
RN (rn)	RN rn	로마 숫자로 출력한다. RN은 대문자로, rn은 소문자로 출력한다. 숫자는 1~ 3,999 사이의 정수만 가능하다.
S	S9999 9999S	양수/음수 부호를 해당 위치에 출력한다. 형식 문자열의 맨 처음 또는 맨 끝에서만 사용할 수 있다.
TM	TM	가장 작은 수의 문자를 사용해서 숫자를 표현한다. TM9와 TMe의 형태로 사용할 수 있다. TM9는 TM과 같다. – TM9는 과학적 기수법이 아닌 고정 소수점으로 숫자를 출력한다. – TMe는 과학적 기수법으로 출력한다. – TM은 다른 형식 요소와 함께 쓰일 수 없다.

다음은 TO_NUMBER 함수를 사용했을 때 각 NUMBER 타입의 값이 형식 문자열에 따라 어떻게 출력되는지를 보여준다.

NUMBER 타입의 값	형식 문자열	출력 결과
0	99.99	' .00'
0.1	99.99	' .10'
-0.1	99.99	' -.10'
0	90.99	' 0.00'

NUMBER 타입의 값	형식 문자열	출력 결과
0.1	90.99	' 0.10'
-0.1	90.99	' -0.10'
0	9999	' 0'
1	9999	' 1'
0.1	9999	' 0'
-0.1	9999	' -0'
123.456	999.999	' 123.456'
-123.456	999.999	'-123.456'
123.456	FM999.999	'123.456'
123.45	999.009	' 123.450'
123.45	FM999.009	'123.45'
123	FM999.009	'123.00'
12345	99999S	'12345+'

2.4.2. 날짜형 타입

날짜형 타입의 형식 문자열은 TO_CHAR 함수, TO_DATE 함수, TO_TIMESTAMP 함수, TO_TIMESTAMP_TZ 함수에서 파라미터로 사용할 수 있다.

함수	설명
TO_CHAR	날짜형 타입의 값을 문자열로 변환한다.
TO_DATE	문자열을 날짜형 타입의 값으로 변환한다.
TO_TIMESTAMP	문자열을 날짜/시간형 타입의 값으로 변환한다.
TO_TIMESTAMP_TZ	문자열을 시간대를 포함하는 날짜/시간형 타입의 값으로 변환한다.

날짜형 타입의 형식 문자열은 다음과 같은 특징이 있다.

- 여러 가지 형식 요소로 구성된다. 날짜형 타입에 포함된 '연, 월, 일, 시, 분, 초' 등의 값을 각각 어떤 형식으로 출력할 것인지 지정한다. 예를 들어 연도를 나타내는 형식 요소 문자열인 'YYYY'와 'YY'의 경우 연도의 마지막 네 자리 또는 두 자리만 출력하도록 한다. 즉, 2009년의 경우 각각 '2009'와 '09'를 출력한다.
- 하이픈(-) 또는 슬래시(/)를 삽입할 수 있다. 만약 형식 문자열 내에 형식 요소 이외의 문자열을 삽입하고 싶다면 큰따옴표(" ")를 이용하여 나타낸다.
- 대소문자를 구분하는 형식 요소가 있다. 예를 들어 요일을 출력하기 위한 형식 요소인 'DAY'는 요일 문자열 전체를 대문자로, 'Day'는 맨 앞 글자만 대문자로, 'day'는 전체를 소문자로 출력한다. 월요일의 경우, 각각 'MONDAY', 'Monday', 'monday'로 출력한다.

다음은 날짜형 타입의 형식 문자열에 포함될 수 있는 형식 요소이다.

형식 요소	TO_* 함수의 입력으로 사용 가능 여부	설명
- , . ; : / "text"	-	결과 값에 해당하는 위치에 그대로 출력된다.
AD A.D. BC B.C.	예	서기 또는 기원 전을 표시한다.
AM A.M. PM P.M.	예	오전 또는 오후를 표시한다.
CC SCC	아니오	세기를 출력한다. (예: 2005년일 때는 21을 반환한다.) SCC는 기원전일 경우 마이너스(-)를 붙인다.
D	아니오	1주일 중 몇 번째 날인지 출력한다. (1-7)
DAY	아니오	요일을 출력한다. (예: THURSDAY)
DD	예	1개월 중 몇 번째 날인지 출력한다. (1-31)
DDD	예	1년 중 몇 번째 날인지 출력한다. (1-366)
DY	아니오	축약 표기한 요일을 출력한다. (예: THU)
FF[1~9]	예	소수점 이하 자리의 초를 나타낸다. FF 뒤에 명시한 숫자(1~9) 만큼 소수점 이하 자릿수가 출력된다. 명시하지 않으면 데이터 타입의 디폴트 정밀도를 따른다.

형식 요소	TO_* 함수의 입력으로 사용 가능 여부	설명
FM	예	앞뒤 공백을 제거하고 출력하도록 하는 형식 조절자이다.
FX	예	형식 문자열과 입력 문자열의 일치여부를 검사하는 형식 조절자이다.
HH HH12	예	시간을 출력한다. (1-12)
HH24	예	시간을 출력한다. (0-23)
IYYYY IYYY IYY IY	아니오	4(3/2/1)자릿수 연도 표기를 ISO 표준에 의거 출력한다.
MI	예	분을 출력한다. (0-59)
MM	예	달을 출력한다. (1-12)
MON	예	축약된 달 이름을 출력한다. (예: DEC)
MONTH	예	달을 출력한다. (예: DECEMBER)
Q	아니오	분기를 출력한다. (1-4)
RM	예	달을 로마 숫자로 출력한다. (I-XII)
RR	예	두 자릿수의 연도의 입력 값에 따라 몇 세기인지 자동으로 조절한다.
RRRR	예	반올림한 연도를 표기한다. 4자리 또는 2자리를 입력받는다. 2자리로 입력했을 경우는 RR과 똑같이 동작한다.
SS	예	초를 출력한다. (0-59)
SSSSS	예	자정을 기준으로 현재 몇 초인지 출력한다. (0-86399)
WW	아니오	1년 중 몇 번째 주인지 출력한다. (1-53) 첫 번째 주는 1월 1일에 시작하고 1월 7일에 끝난다.
W	아니오	1개월 중 몇 번째 주인지 출력한다. (1-5) 첫 번째 주는 그 달 1일에 시작하고 그 달 7일에 끝난다.
X	예	점(.)을 출력한다. TIMESTAMP를 출력하는 경우 소수점 자릿수를 표현하기 위해서 사용된다.
YEAR SYEAR	아니오	숫자로 된 연도를 단어로 풀어 쓴다. SYEAR는 기원전 연도에 마이너스(-)를 붙인다.

형식 요소	TO_* 함수의 입력으로 사용 가능 여부	설명
YYYY SYYYY	아니오	4자릿수 연도를 표기한다. SYYYY는 기원전 연도에 마이너스(-)를 붙인다.
YYY YY Y	예	3(2/1)자릿수 연도를 출력한다.
TZH	예	시간대에서의 시간을 출력한다.
TZM	예	시간대에서의 분을 출력한다.
TZR	예	시간대에서의 지역을 출력한다.
TZD	예	시간대에서의 일광 절약시간 약어를 출력한다. 이 값은 반드시 TZR에서의 지역과 일치해야 한다.
EE	예	연호를 출력한다(Japanese Imperial and Thai Buddha calendars).
E	예	연호의 약어를 출력한다(Japanese Imperial and Thai Buddha calendars).
J	예	Julian Day를 출력한다. J는 기원전 4712년 1월 1일부터 계수한 값으로 음이 아닌 정수여야 한다.

위의 표에서의 RR 형식 요소는 YY 형식 요소와 유사하나, 다른 세기의 연도 값을 더 간편하게 명시하고 저장할 수 있다.

RR 형식 요소의 정확한 규칙은 다음과 같다.

- 현재 연도의 마지막 두 자리가 00 ~ 49 사이일 경우
  - 명시한 두 자릿수 연도가 00 ~ 49 사이일 경우 반환되는 연도는 현재 연도와 앞의 두 자리가 같다.
  - 명시한 두 자릿수 연도가 50 ~ 99 사이일 경우 반환되는 연도의 앞의 두 자리는 현재 연도의 앞의 두 자리에 1을 뺀 값과 같다.
  - 다음은 2000~2049년에 수행했음을 가정했을 때의 예제이다.

```
SQL> SELECT TO_CHAR(TO_DATE('20/08/13', 'RR/MM/DD'), 'YYYY') YEAR FROM DUAL;

YEAR
----------------------------
2020

SQL> SELECT TO_CHAR(TO_DATE('98/12/25', 'RR/MM/DD'), 'YYYY') YEAR FROM DUAL;
```

```
YEAR
---------------------------
1998
```

- 현재 연도의 마지막 두 자리가 50 ~ 99 사이일 경우
  - 명시한 두 자릿수 연도가 00 ~ 49 사이일 경우 반환되는 연도의 앞의 두 자리는 현재 연도의 앞의 두 자리에 1을 더한 값과 같다.
  - 명시한 두 자릿수 연도가 50 ~ 99 사이일 경우 반환되는 연도는 현재 연도와 앞의 두 자리가 같다.
  - 다음은 1950~1999년에 수행했음을 가정했을 때의 예제이다.

```
SQL> SELECT TO_CHAR(TO_DATE('12/10/27', 'RR/MM/DD'), 'YYYY') YEAR FROM DUAL;

YEAR
---------------------------
2012

SQL> SELECT TO_CHAR(TO_DATE('92/02/08', 'RR/MM/DD'), 'YYYY') YEAR FROM DUAL;

YEAR
---------------------------
1992
```

날짜형 타입의 형식 요소에는 다음과 같은 접미어를 사용할 수 있다.

접미어	의미	형식 문자열	출력 예
TH	서수	DDTH	05TH
SP	철자로 표기한 숫자	DDSP	FIVE
SPTH 또는 THSP	철자로 표기한 서수	DDSPTH	FIFTH

참고

이 접미어는 출력할 때만 사용할 수 있으며, 숫자를 출력하는 형식 요소에만 사용할 수 있다.

2.4.3. 형식 조절자

형식 조절자는 형식 문자열 안에서 여러 번 표현할 수 있다. 이러한 경우 나타낼 때마다 각각의 기능이 비활성화되어 있으면 활성화하고, 활성화 되어 있으면 비활성화한다.

FM 형식 조절자를 통해 공백을 채우는 방식을 변경할 수 있고, FX 형식 조절자를 통해 입력 문자열과 형식 문자열이 정확히 일치하는지 검사할 수 있다.

FM

Tibero는 각각의 형식 요소에 대해 그 형식 요소가 출력하는 문자열의 최대 크기만큼 공백 문자를 채운다. 예를 들어 MONTH 형식 요소의 경우, 가장 긴 달은 'SEPTEMBER'이므로 나머지 달은 오른쪽에 공백을 채워서 아홉 글자를 맞추게 된다.

FM을 명시하면 공백을 채우는 방법이 다음과 같이 달라진다.

- TO_CHAR 함수의 DATE 타입 형식 문자열에서, FM을 사용하는 경우
  - 모든 공백 문자와 앞에 붙는 '0'이 제거된다. 예를 들어 4월을 MM으로 출력하면 04 대신 4가 출력된다.
  - FM이 비활성화된 경우는 각각의 형식 요소에 대해 항상 같은 길이의 문자열이 출력되지만, FM이 활성화되어 있을 경우는 이 문자열의 길이가 입력에 따라 바뀔 수 있다.
- TO_CHAR 함수의 NUMBER 타입 형식 문자열에서, FM을 사용하는 경우
  - 숫자 앞에 붙는 모든 공백 문자와 '9' 형식 요소에 의해 생긴 뒤에 붙는 모든 0이 제거된다. 그러므로 결과는 왼쪽으로 정렬된 형태로 출력된다.
  - FM이 비활성화된 경우 공백 문자는 숫자의 앞을 채우기 때문에, 항상 오른쪽 정렬이 된다.

FX

Tibero는 형식 문자열과 입력 문자열이 정확히 일치하는지 검사하고, 만약 하나라도 어긋나는 경우엔 에러를 발생시킨다.

FX를 명시하면 다음과 같은 제약조건을 가진다.

- 구분자나 큰따옴표(" ") 문자열의 위치가 형식 문자열에서의 위치와 정확히 일치해야 한다.
- 추가적인 공백 문자를 허용하지 않는다. 만약 FX가 비활성화된 경우엔 공백 문자는 무시된다.
- 입력 문자열에서 숫자들의 자릿수는 형식 문자열 각 요소에서 사용하는 자릿수와 정확히 일치해야 한다.

2.5. 의사 컬럼

의사 컬럼은 사용자가 명시적으로 선언하지 않아도, Tibero 시스템이 자동으로 모든 테이블에 포함하는 컬럼이다.

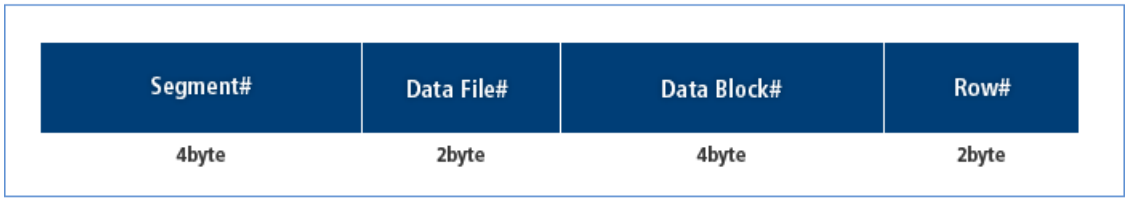
2.5.1. ROWID

ROWID는 전체 데이터베이스 내의 하나의 로우를 유일하게 참조하는 식별자이다. ROWID는 그 로우의 디스크의 물리적인 위치를 가리키고 있으며, 그 로우가 삭제될 때까지 변화되지 않는다.

Tibero에서는 데이터베이스의 저장을 위한 디스크 구조를 다단계로 구성하고 있다. ROWID를 이용하여 디스크의 특정 로우를 찾아갈 수 있으려면, ROWID는 이러한 디스크 구조를 반영해야 한다.

Tibero의 ROWID는 다음의 [그림 2.1]과 같은 구조를 갖는다.

[그림 2.1] ROWID의 구조



ROWID는 전체 12byte로 구성되어 있으며, Segment, Data File, Data Block, Row가 각각 4, 2, 4, 2byte로 되어 있다.

ROWID 값을 표현하기 위한 포맷으로는 BASE64 인코딩을 이용한다. BASE64 인코딩은 6bits에 포함된 숫자를 8bits 문자로 나타내는 방식으로, 0 ~ 63까지의 숫자를 A ~ Z, a ~ z, 0 ~ 9, +, /로 대치한다.

ROWID를 BASE64 인코딩으로 변환하면 Segment#, Data File#, Data Block#, Row#가 각각 6, 3, 6, 3byte로 되고, 'SSSSSSFFFBBBBBBRRR'의 형태를 갖는다. 예를 들어 Segment# = 100, Data File# = 20, Data Block# = 250, Row# = 0인 ROWID는 'AAAABkAAUAAAAD6AAA'로 나타낸다.

2.5.2. ROWNUM

ROWNUM은 SELECT 문장의 실행 결과로 나타나는 로우에 대하여 순서대로 번호를 부여한다. 질의 결과로 반환되는 첫 번째 로우는 ROWNUM = 1이며 두 번째 로우는 ROWNUM = 2, 세 번째 로우는 ROWNUM = 3, …, 등등의 값을 갖는다.

Tibero에서 ROWNUM이 할당되는 순서는 다음과 같다.

1. 질의를 수행한다.
2. 질의 결과로 로우가 생성된다.
3. 로우를 반환하기 직전에 그 로우에 ROWNUM이 할당된다.

   Tibero는 내부적으로 ROWNUM 카운터를 가지고 있으며, 카운터 값을 질의 결과의 로우에 할당한다.
4. ROWNUM을 할당 받은 로우에 ROWNUM에 대한 조건식을 적용한다.
5. 조건식을 만족하면 할당된 ROWNUM이 확정되고, 내부의 ROWNUM 카운터의 값이 1로 증가한다.
6. 조건식을 만족하지 않으면 그 로우는 버려지고, 내부의 ROWNUM 카운터의 값은 증가하지 않는다.

ROWNUM은 질의 결과의 로우 개수를 한정하기 위하여 많이 사용된다. 아래의 SQL 문장은 10개의 로우만을 반환하는 예이다.

```
SELECT * FROM EMP WHERE ROWNUM <= 10;
```

ROWNUM은 질의를 처리하는 거의 마지막 단계에서 할당된다. 때문에 같은 SELECT 문장이라 하더라도 내부적으로 어떤 단계로 질의를 처리하였는가에 따라 다른 결과를 가져올 수 있다. 예를 들어 질의 최적화기가 인덱스의 사용 유무를 어떻게 결정하느냐에 따라 다른 결과를 얻는다.

ROWNUM을 포함하는 질의가 항상 같은 결과를 반환하도록 하기 위하여 ORDER BY 절을 사용할 수 있다. 하지만, Tibero에서는 WHRER 절을 포함하는 모든 부질의를 처리한 다음에 ORDER BY 절을 처리한다. 따라서 ORDER BY 절을 이용해서 항상 같은 결과를 얻을 수는 없다.

예를 들어 다음의 질의는 실행할 때마다 다른 결과를 얻는다.

```
SELECT * FROM EMP WHERE ROWNUM <= 10 ORDER BY EMPNO;
```

위의 질의를 다음과 같이 변환하면 ORDER BY 절을 먼저 처리하게 되므로 항상 같은 결과를 얻을 수 있다.

```
SELECT * FROM (SELECT * FROM EMP ORDER BY EMPNO)
WHERE ROWNUM <= 10;
```

또한, 다음과 같은 SELECT 문장은 하나의 로우도 반환하지 않는다.

```
SELECT * FROM EMP WHERE ROWNUM > 1;
```

그 이유는 ROWNUM 값이 확정되기 전에 ROWNUM에 대한 조건식이 수행되기 때문이다. 위의 SELECT 문의 결과는 첫 번째 로우가 ROWNUM = 1이기 때문에 조건식을 만족하지 않는다. 조건식을 만족하지 않으면 ROWNUM 카운터의 값은 변하지 않는다. 따라서 두 번째 결과 로우도 ROWNUM = 1이므로 반환되지 않는다. 결국, 하나의 로우도 반환되지 않는다.

2.5.3. LEVEL

LEVEL은 계층 질의를 실행한 결과에 각 로우의 트리 내 계층을 출력하기 위한 컬럼 타입이다. 최상위 로우의 LEVEL 값은 1이며, 하위 로우로 갈수록 1씩 증가한다. 계층 질의와 LEVEL 컬럼 값의 출력에 대해서는 “5.5. 계층 질의”에서 자세하게 설명한다.

2.5.4. CONNECT_BY_ISLEAF

CONNECT_BY_ISLEAF 의사 컬럼은 현재 로우가 CONNECT BY 조건에 의해 정의된 트리(Tree)의 리프(Leaf)이면 1을 반환하고 그렇지 않을 경우에는 0을 반환한다. 이 정보는 해당 로우가 계층 구조(Hierarchy)를 보여주기 위해 확장될 수 있는지 없는지를 나타낸다.

다음은 CONNECT_BY_ISLEAF 의사 컬럼을 사용한 예이다.

```
SQL> SELECT ENAME, CONNECT_BY_ISLEAF, LEVEL, SYS_CONNECT_BY_PATH(ENAME,'-') "PATH"
     FROM EMP2
       START WITH ENAME = 'Clark'
       CONNECT BY PRIOR EMPNO = MGRNO
       ORDER BY ENAME;

ENAME           CONNECT_BY_ISLEAF LEVEL      PATH
--------------- ----------------- ---------- ----------------------
Alicia                          1          3 -Clark-Martin-Alicia
Allen                           1          3 -Clark-Ramesh-Allen
Clark                           0          1 -Clark
James                           1          3 -Clark-Martin-James
John                            0          3 -Clark-Ramesh-John
Martin                          0          2 -Clark-Martin
Ramesh                          0          2 -Clark-Ramesh
Ward                            1          4 -Clark-Ramesh-John-Ward
```

2.5.5. CONNECT_BY_ISCYCLE

CONNECT_BY_ISCYCLE은 계층형 질의에서 사용되는 의사 컬럼으로서 해당 로우가 자식 노드를 갖고 있음과 동시에 그 자식 노드가 해당 로우의 부모 노드가 되는지를 판별한다. 즉, 부모 노드와 자식 노드의 루프 여부를 판별하여 이러한 자식 노드가 있을 경우 1, 없을 경우 0을 반환한다.

이 의사컬럼은 CONNECT BY 절에 반드시 NOCYCLE 구문이 명시되어야만 사용할 수 있다. 만약 NOCYCLE을 명시할 경우 루프가 발생하더라도 에러를 발생시키지 않는다.

다음은 CONNECT_BY_ISCYCLE 의사 컬럼을 사용한 예이다.

```
SQL> SELECT ENAME, CONNECT_BY_ISCYCLE, LEVEL FROM EMP
     START WITH ENAME = 'Alice'
     CONNECT BY NOCYCLE PRIOR EMPNO = MGRNO
     ORDER BY ENAME;

ENAME           CONNECT_BY_ISCYCLE LEVEL
--------------- ------------------ ----------
Alice                            0          1
Smith                            1          2
Micheal                          0          3
Viki                             0          2
Jane                             1          2
Jacob                            0          4
```

2.6. NULL

한 로우에서 어떤 컬럼에 값이 없을 때 그 컬럼을 NULL이라고 하거나 NULL 값을 가진다고 한다. NULL은 NOT NULL 제약과 PRIMARY KEY 제약이 걸리지 않은 모든 데이터 타입의 칼럼에 포함될 수 있다. 실제 값을 모르거나 아무런 의미 없는 값이 필요할 때 사용할 수 있다. NULL과 0은 다르기 때문에 NULL을 0으로 나타내면 안 된다. 다만 문자 타입의 컬럼에 빈 문자열('')이 들어가면 NULL로 처리된다.

NULL을 포함한 산술연산의 결과는 항상 NULL이다. 또한, 문자열 접합 연산(||)을 제외한 NULL을 포함하는 모든 연산의 결과도 NULL이다.

```
NULL + 1 = NULL
```

2.6.1. 함수에서의 NULL

REPLACE, NVL, CONCAT을 제외한 모든 상수 함수는 함수의 파라미터가 NULL일 경우 반환 값은 NULL이다. NVL 함수를 사용하면 NULL을 다른 값으로 반환할 수 있다. 컬럼 값이 NULL일 때 NVL(column, 0) = 0이 되며, 컬럼 값이 NULL이 아닐 때 NVL(column, 0) = column이 된다. 대부분의 집단 함수는 NULL을 무시한다.

다음은 NULL을 포함한 데이터에 AVG 함수를 사용했을 때의 결과이다.

```
DATA = {1000, 500, NULL, NULL, 1500}
AVG(DATA) = (1000 + 500 + 1500) /3 = 1000
```

2.6.2. NULL에 대한 비교조건

NULL을 검사할 수 있는 비교조건은 IS NULL과 IS NOT NULL만 가능하다. NULL은 데이터가 없다는 것을 의미한다. 때문에 NULL과 NULL, NULL과 NULL이 아닌 다른 값을 서로 비교할 수 없다.

다만 DECODE 함수에서는 두 개의 NULL을 비교할 수 있다.

```
SQL> SELECT DECODE(NULL, NULL, 1) FROM DUAL;

DECODE(NULL,NULL,1)
-------------------
                  1
```

위의 예에서 DECODE 함수를 통해 NULL이 서로 비교되었으며, 그 결과로 서로 같다는 의미인 '1'이 반환되었음을 알 수 있다.

만일 NULL에 다른 비교조건을 사용하면, 결과는 UNKOWN으로 나타난다. UNKNOWN으로 판별되는 조건은 거의 대부분 FALSE처럼 처리된다. 그 예로 SELECT 문에서 WHERE 절에 UNKNOWN으로 판별되는 조건이 있을 경우 반환되는 로우가 없다. 하지만 UNKNOWN이 FALSE와 다른 점은 UNKNOWN 조건에 또 다른 연산자가 더해져도 결과는 UNKNOWN이라는 점이다.

다음은 FALES에 NOT 연산자를 사용한 결과와 UNKNOWN에 NOT 연산자를 사용한 결과의 차이를 보여준다.

```
NOT FALES = TRUE
NOT UNKNOWN = UNKNOWN
```

2.7. 주석

SQL 문장과 스키마 객체에는 주석을 삽입할 수 있다. 책이나 문서에서 주석이 낱말이나 문장의 뜻을 쉽게 풀이하는 역할을 하듯 SQL 문장에도 주석을 활용하여 해당 문장의 부연 설명을 삽입할 수 있다.

주석의 내용은 데이터베이스에서 사용하는 문자 집합으로 표현할 수 있는 문자라면 어떤 내용이라도 포함될 수 있으며, 예약어, 파라미터, 점 사이 등 어떤 곳에도 추가될 수 있다. 주석은 SQL 문장의 실행에는 전혀 영향을 주지 않는다.

주석은 애플리케이션의 소스 코드를 읽기 쉽고 관리하기 좋게 만들어 준다. 예를 들어 애플리케이션 소스 코드 내부에 있는 SQL 문장에, 그 문장의 용도와 목적 등의 주석을 삽입해 두면, 각각의 문장의 의미를 쉽게 파악할 수 있다.

주석을 삽입하는 방법은 다음과 같이 두 가지이다.

- 시작 기호(/*)로 주석의 시작을 나타내고 마침 기호(*/)로 주석을 끝낸다.

  주석의 내용을 여러 줄에 걸쳐 삽입할 수 있다. 시작 기호(/*)와 마침 기호(*/)를 내용과 구분하기 위해 공백이나 줄 바꿈을 사용할 필요는 없다.

- '--'로 주석의 시작을 나타내고 바로 뒤에 주석의 내용을 적는다.

  해당 줄의 끝이 주석의 끝을 나타내므로 주석의 내용이 다음 줄로 넘어가서는 안 된다.

다음은 SQL 문장에 주석을 삽입한 예이다.

```
SELECT emp_id, emp_name,
       e.dept_id                /* 부서가 총무과인 직원의 명단을 출력한다. */
                                /* 테이블 */
FROM emp e, dept d
WHERE e.dept_id = d.dept_id
 AND  d.dept_name = '총무과'
 AND  e.status != 1;            /* 퇴사한 사람 제외 */


SELET emp_id, emp_name, e.dept  -- 부서가 자재과인 직원의 명단을 출력한다.
                                -- 테이블
FROM emp e, dept d
WHERE e.dept_id = d.dept_id
 AND  d.dept_name = '자재과'
 AND  e.status != 1;            -- 퇴사한 사람 제외
```

위와 같이 SQL 문장뿐만 아니라 스키마 객체에도 주석을 삽입할 수 있다. 즉 COMMENT 명령을 사용하여 스키마 객체인 테이블, 뷰, 컬럼에 주석을 삽입할 수 있다. 스키마 객체에 삽입된 주석은 데이터 사전에 저장된다.

2.8. 힌트

힌트는 일종의 지시문이다. SQL 문장에 힌트를 추가하여 Tibero의 질의 최적화기(Optimizer)에 특정 행동을 지시하거나 질의 최적화기의 실행 계획을 변경한다. 질의 최적화기가 항상 최적의 실행 계획을 수립할 수는 없다. 따라서 개발자가 질의 최적화기의 실행 계획을 직접 수정할 수 있는 방법을 마련한 것이 바로 힌트이다.

SQL 문장의 한 블록당 힌트는 하나만 올 수 있으며, SELECT, UPDATE, INSERT, DELETE 절 바로 뒤에 위치해야 한다.

다음은 힌트를 사용한 예이다.

```
(DELETE|INSERT|SELECT|UPDATE) /*+ hint [hint] ... */

또는

(DELETE|INSERT|SELECT|UPDATE) --+ hint [hint] ...
```

힌트를 사용할 때 주의할 점은 다음과 같다.

- 힌트는 반드시 DELETE, INSERT, SELECT, UPDATE 절 뒤에만 올 수 있다.
- '+' 기호는 반드시 주석 구분자('/*' 또는 '--') 바로 뒤에 공백 없이 붙여 써야 한다.
- 힌트와 '+' 기호 사이에 공백은 있어도 되고, 없어도 된다.
- 문법에 맞지 않는 힌트는 주석으로 취급되며, 에러는 발생하지 않는다.

다음은 힌트의 종류이다.

구성요소	힌트	설명
질의 변형	NO_QUERY_TRANSFORMATION	질의 변형기에게 전체 쿼리에 대해서 변형을 실행하지 않도록 지시한다.
	NO_MERGE	질의 변형기에게 특정 뷰에 대한 뷰 병합(View Merging)을 하지 않도록 지시한다.
	UNNEST	질의 변형기에게 특정 부질의를 언네스팅(Unnesting)하도록 지시한다.
	NO_UNNEST	질의 변형기에게 특정 부질의에 대해 언네스팅을 수행하지 않도록 지시한다.
	NO_JOIN_ELIMINATION	질의 변형기에게 조인 제거를 수행하지 않도록 지시한다.

구성요소	힌트	설명
	STAR_TRANSFORMATION	질의 변형기에게 스타 변형(Star Transformation)을 하도록 지시한다.
최적화 방법	ALL_ROWS	전체 결과에 대한 처리량이 가장 많도록 처리과정의 최적화를 선택한다.
	FIRST_ROWS	결과를 가장 빠르게 보여줄 수 있도록 결과 표시의 최적화를 선택한다.
접근 방법	FULL	전체 테이블을 스캔하도록 지시한다.
	INDEX	명시한 인덱스를 사용한 인덱스 스캔을 하도록 지시한다.
	NO_INDEX	명시한 인덱스를 사용한 인덱스 스캔을 하지 않도록 지시한다.
	INDEX_ASC	명시한 인덱스를 사용한 인덱스 스캔을 오름차순으로 하도록 지시한다.
	INDEX_DESC	명시한 인덱스를 사용한 인덱스 스캔을 내림차순으로 하도록 지시한다.
	INDEX_FFS	명시한 인덱스를 사용한 인덱스를 사용해 빠른 전체 인덱스 스캔(Fast Full Index Scan)을 하도록 지시한다.
	NO_INDEX_FFS	명시한 인덱스를 사용한 빠른 전체 인덱스 스캔을 하지 않도록 지시한다.
	INDEX_RS	명시한 인덱스를 사용한 인덱스를 사용해 범위 인덱스 스캔(Range Index Scan)을 하도록 지시한다.
	NO_INDEX_RS	명시한 인덱스를 사용한 범위 인덱스 스캔을 하지 않도록 지시한다.
	INDEX_SS	명시한 인덱스를 사용한 인덱스를 사용해 인덱스 스킵 스캔(Index Skip Scan)을 하도록 지시한다.
	NO_INDEX_SS	명시한 인덱스를 사용한 인덱스 스킵 스캔을 하지 않도록 지시한다.
	INDEX_JOIN	명시한 테이블에 두 개 이상의 인덱스를 사용하여 자체 조인(Self Join)하도록 지시한다.
조인 순서	LEADING	먼저 조인되어야 할 테이블의 집합을 명시한다.
	ORDERED	테이블을 FROM 절에 명시된 순서대로 조인하도록 지시한다.
조인 방법	USE_NL	중첩 루프 조인을 사용하도록 지시한다.
	NO_USE_NL	중첩 루프 조인을 사용하지 않도록 지시한다.
	USE_NL_WITH_INDEX	명시한 인덱스와 두 테이블에 대한 조인 조건을 이용해 중첩 루프 조인을 사용하도록 지시한다.
	USE_MERGE	합병 조인을 사용하도록 지시한다.

구성요소	힌트	설명
	NO_USE_MERGE	합병 조인을 사용하지 않도록 지시한다.
	USE_HASH	해시 조인을 사용하도록 지시한다.
	NO_USE_HASH	해시 조인을 사용하지 않도록 지시한다.
	HASH_SJ	부질의를 언네스팅할 때 해시방법을 이용한 세미조인으로 하도록 지시한다.
	HASH_AJ	부질의를 언네스팅할 때 해시방법을 이용한 안티조인으로 하도록 지시한다.
	MERGE_SJ	부질의를 언네스팅할 때 머지방법을 이용한 세미조인으로 하도록 지시한다.
	MERGE_AJ	부질의를 언네스팅할 때 머지방법을 이용한 안티조인으로 하도록 지시한다.
	NL_SJ	부질의를 언네스팅할 때 네스티드 룹 방법을 이용한 세미조인으로 하도록 지시한다.
	NL_AJ	부질의를 언네스팅할 때 네스티드 룹 방법을 이용한 안티조인으로 하도록 지시한다.
	SWAP_JOIN_INPUTS	해시 조인을 수행하는 경우 빌드 테이블이 되도록 지시한다.
	NO_SWAP_JOIN_INPUTS	해시 조인을 수행하는 경우 조인 순서가 변경되지 않도록 지시한다.
병렬 처리	PARALLEL	지정한 개수의 스레드를 사용해 질의의 수행을 병렬로 진행하도록 지시한다.
	NO_PARALLEL	질의의 수행을 병렬로 진행하지 않도록 지시한다.
	PQ_DISTRIBUTE	조인을 포함한 질의의 병렬 처리에서 로우의 분산 방법을 지시한다.
실체화 뷰	REWRITE	비용의 비교 없이 실체화 뷰(Materialized View)를 사용하여 질의의 다시 쓰기를 지시한다.
	NO_REWRITE	질의의 다시 쓰기를 하지 않도록 지시한다.
	MATERIALIZE	With 절 안에 있는 Subquery를 실체화 뷰(Materialized View)로 만들도록 지시한다.
	INLINE	With 절 안에 있는 Subquery를 실체화 뷰(Materialized View)로 만들지 않도록 지시한다.
기타	APPEND	DML 문장에서 직접 데이터 파일에 추가하는 삽입 방법 즉 Direct-Path 방식을 수행하도록 지시한다.
	APPEND_VALUES	VALUES 절을 사용하는 INSERT 문에서 직접 데이터 파일에 추가하는 삽입 방법 즉 Direct-Path 방식을 수행하도록 지시한다.

구성요소	힌트	설명
	NOAPPEND	DML 문장에서 Direct-Path 방식을 수행하지 않도록 지시한다.
	IGNORE_ROW_ON_DUPKEY_INDEX	유일키 제약조건을 위배하는 로우가 삽입 될 때, 에러를 발생하지 않도록 한다.
	CARD	지정 테이블의 Cardinality를 지정하여, 쿼리를 최적화 할 때 이용하도록 한다.
	MONITOR	쿼리를 수행할 때 쿼리 수행 정보를 모으도록 지시한다.
	NO_MONITOR	쿼리를 수행할 때 쿼리 수행 정보를 모으지 않도록 지시한다.
	USE_CONCAT	OR expansion된 플랜만 생성한다.
	NO_EXPAND	OR expansion된 플랜을 배제한다.
	RESULT_CACHE	Query 결과를 저장하기 위해 Result Cache를 사용한다.
	NO_SUBQUERY_CACHE	쿼리를 수행하는 경우 특정 부질의에 대해 부질의 결과를 캐싱하지 않도록 강제한다.
	OPT_PARAM	쿼리가 수행되는 동안 초기화 환경 변수를 바꾸는 데 사용한다.

2.8.1. 질의 변형

질의 변형(Query Transformation)에 대한 힌트를 사용하여 Tibero의 질의 변형 방식에 영향을 줄 수 있다.

NO_QUERY_TRANSFORMATION

NO_QUERY_TRANSFORMATION는 질의 변형기(Query Transformer)가 전체 쿼리에 대해 변형을 실행하지 않도록 지시하는 힌트이다. Tibero에서는 쿼리 변형이 자동으로 수행되며, 최적화된 형태로 쿼리를 변형하여 실행계획을 생성하다. NO_QUERY_TRANSFORMATION 힌트를 사용한다면 디폴트로 수행되는 쿼리 변형을 막을 수 있다.

- 문법

no_query_transformation_hint

NO_MERGE

NO_MERGE는 질의 변형기(Query Transformer)가 특정 뷰에 대해 뷰 병합을 하지 않도록 지시하는 힌트이다. Tibero에서는 뷰 병합이 디폴트로 수행되며, 뷰가 병합이 가능할 경우 상위의 질의 블록과 결합해

하나의 질의 블록을 형성한다. NO_MERGE 힌트를 사용하면 이렇게 디폴트로 수행되는 뷰의 병합을 막을 수 있다.

- 문법

no_merge_hint

- 예제

다음은 NO_MERGE 힌트를 사용하는 예이다.

```
SELECT *
FROM T1, (SELECT /*+ NO_MERGE */ * FROM T2, T3 WHERE T2.A = T3.B) V
WHERE T1.C = V.D
```

위의 예에서처럼 NO_MERGE 힌트는 병합되기를 원하지 않는 뷰의 질의 블록에 명시한다. 힌트가 없었다면 뷰가 병합되어 질의 최적화기에서 테이블 T1, T2, T3에 대한 조인 순서와 조인 방법을 고려하게 되지만, 위와 같이 힌트가 있을 경우는 뷰가 병합되지 못하기 때문에 T2와 T3가 먼저 조인되고, 그 이후에 T1이 조인된다.

UNNEST

UNNEST는 질의 변형기가 특정 부질의(Subquery)를 언네스팅하도록 지시하는 힌트이다. Tibero는 부질의 언네스팅을 디폴트로 수행하지만, 특정 쿼리만 언네스팅을 하려면 초기화 파라미터에서 언네스팅을 해제하면 된다. 그러면 UNNEST 힌트를 이용할 수 있다. UNNEST 힌트는 부질의 블록에 명시한다.

- 문법

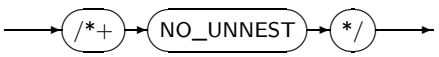
unnest_hint

NO_UNNEST

NO_UNNEST는 질의 변형기가 특정 부질의에 대해 언네스팅을 수행하지 않도록 지시하는 힌트이다. Tibero는 부질의 언네스팅을 디폴트로 수행하며 언네스팅이 가능한 경우 부질의를 조인으로 변환한다. 이때 NO_UNNEST 힌트를 사용해서 언네스팅을 막을 수 있다. NO_UNNEST 힌트는 부질의 블록에 명시한다.

- 문법

no_unnest_hint



NO_JOIN_ELIMINATION

NO_JOIN_ELIMINATION는 질의 변형기(Query Transformer)가 불필요한 조인을 찾아서 제거하지 않도록 지시하는 힌트이다. Tibero에서는 디폴트로 질의 결과를 생성하는데 필요하지 않은 조인들을 찾아서 제거하며, NO_JOIN_ELIMINATION 힌트를 사용하면 이를 막을 수 있다.

- 문법

no_join_elimination_hint

- 예제

다음은 NO_JOIN_ELIMINATION 힌트를 사용하는 예이다.

```
SELECT /*+ NO_JOIN_ELIMINATION */ T2.FK, T2.A FROM T1, T2 WHERE T2.FK = T1.PK
```

위의 예처럼 T2의 컬럼을 요청하였을 때 T1과 T2 사이에 참조 관계가 정의되어 있다면 T1과 조인을 하지 않아도 조건절에 주어진 T2.FK = T1.PK는 T2.FK가 NULL이 아닌 한 참임을 알 수 있다. 질의 변환기는 이처럼 필요하지 않은 조인을 찾아 제거하는데 NO_JOIN_ELIMINATION 힌트를 적용하면 이러한 기능을 막을 수 있다.

STAR_TRANSFORMATION

STAR_TRANSFORMATION는 스타 변형 (STAR TRANSFORMATION)이 가능할 경우 변형을 시도하도록 지시하는 힌트이다. Tibero에서는 디폴트로 스타 변형을 하지않도록 하는데, STAR_TRANSFORMATION 힌트를 사용하면 이를 사용할 수 있다.

- 문법

star_transformation_hint

- 예제

다음은 STAR_TRANSFORMATION 힌트를 사용하는 예이다.

```
SELECT /*+ STAR_TRANSFORMATION */ s.* FROM S, T1, T2
              WHERE S.C1 = T1.C1 AND S.C2 = T2.C2
```

스타 스키마(STAR SCHEMA)를 사용하는 데이터베이스 환경에서 위의 예처럼 쿼리를 쓰면 스타 변형을 할 수 있다. 이 변형은 기존 INDEX JOIN으로만 풀린 플랜에서 BITMAP KEY ITERATION을 포함한 플랜으로 풀리게 하여 더 효율적인 결과를 얻게 한다.

2.8.2. 최적화 방법

최적화 방법이 적용된 힌트를 사용하여 처리 과정과 결과 표시를 최적화할 수 있다. 만약 최적화 방법이 적용된 힌트가 사용된 질의가 있다면 해당 질의에 대해서는 통계 정보와 초기화 파라미터의 최적화 방법(OPTIMIZER MODE)의 값이 없는 것처럼 처리된다.

ALL_ROWS

ALL_ROWS는 최소한의 리소스를 사용하여 전체 결과에 대한 처리량이 가장 많도록 처리과정의 최적화 방법을 선택하는 힌트이다.

- 문법

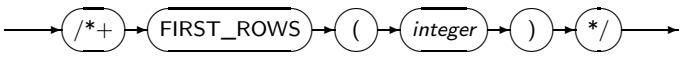
all_rows_hint

FIRST_ROWS

FIRST_ROWS는 첫 로우부터 파라미터로 입력된 번호의 로우까지 가장 빠르게 보여줄 수 있도록 결과 표시의 최적화 방법을 선택하는 힌트이다.

- 문법

first_rows_hint



2.8.3. 접근 방법

접근 방법이 적용된 힌트는 질의 최적화기가 특정 접근 방법의 사용이 가능한 경우, 그 방법을 사용하도록 명시한다. 만일 힌트에서 명시한 방법을 사용할 수 없는 경우에는 질의 최적화기는 그 힌트를 무시한다.

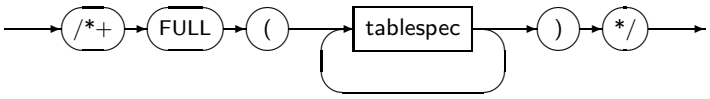
힌트에 명시하는 테이블명은 SQL 문에서 사용하는 이름과 동일해야 한다. 즉, 테이블 이름에 대한 별칭을 사용하였다면, 테이블 이름 대신에 별칭을 사용하여야 한다. SQL 문에서 테이블 이름에 스키마 이름을 포함하여 명시하였더라도 힌트에서는 테이블 이름만을 명시하여야 한다.

FULL

FULL은 명시한 테이블을 스캔할 때, 전체 테이블을 스캔하도록 지시하는 힌트이다. WHERE 절에 명시된 조건식에 맞는 인덱스가 있더라도 전체 테이블 스캔을 사용한다.

- 문법

full_hint



INDEX

INDEX는 명시한 테이블을 스캔할 때, 명시한 인덱스를 사용하여 인덱스 스캔을 하도록 지시하는 힌트이다.

- 문법

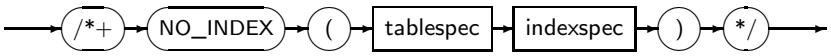
index_hint

NO_INDEX

NO_INDEX는 명시한 테이블을 스캔할 때, 명시한 인덱스를 사용하는 인덱스 스캔을 하지 않도록 지시하는 힌트이다. 만일 NO_INDEX 힌트와 INDEX 또는 INDEX_ASC, INDEX_DESC 힌트가 동일한 인덱스를 명시한다면 질의 최적화기는 이 두 힌트를 모두 무시한다.

- 문법

no_index_hint

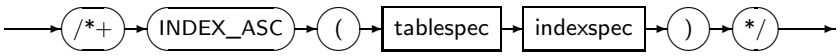


INDEX_ASC

INDEX_ASC는 명시한 테이블을 스캔할 때, 명시한 인덱스를 사용하여 인덱스 스캔을 하도록 지시하는 힌트이다. 만일 인덱스 범위 스캔을 사용하는 경우에는 인덱스를 오름차순으로 스캔하도록 한다. 현재 Tibero의 인덱스 스캔의 기본 동작이 오름차순이기 때문에 INDEX_ASC는 INDEX와 동일한 작업을 수행한다. 분할된 인덱스의 경우 분할된 각 영역 내에서 오름차순으로 스캔한다.

- 문법

index_asc_hint

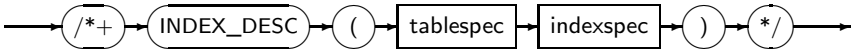


INDEX_DESC

INDEX_DESC는 명시한 테이블을 스캔할 때, 명시한 인덱스를 사용하여 인덱스 스캔을 하도록 지시하는 힌트이다. 만일 인덱스 범위 스캔을 사용하는 경우에는 인덱스를 내림차순으로 스캔하도록 한다. 분할된 인덱스의 경우 분할된 각 영역 내에서 내림차순으로 스캔한다.

- 문법

index_desc_hint

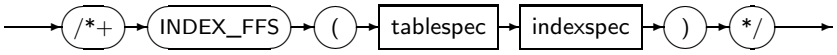


INDEX_FFS

INDEX_FFS는 명시한 테이블에 대해 명시한 인덱스를 사용하여 빠른 전체 인덱스 스캔(Fast Full Index Scan)을 사용하도록 지시하는 힌트이다.

- 문법

index_ffs_hint

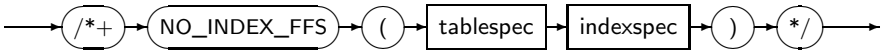


NO_INDEX_FFS

NO_INDEX_FFS는 명시한 테이블에 대해 명시한 인덱스를 사용하는 빠른 전체 인덱스 스캔을 사용하지 않도록 지시하는 힌트이다.

- 문법

no_index_ffs_hint

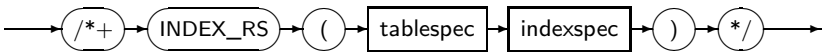


INDEX_RS

INDEX_RS는 명시한 테이블에 대해 명시한 인덱스를 사용하여 범위 인덱스 스캔(Range Index Scan)을 사용하도록 지시하는 힌트이다.

- 문법

index_rs_hint



NO_INDEX_RS

NO_INDEX_RS는 명시한 테이블에 대해 명시한 인덱스를 사용하는 범위 인덱스 스캔을 사용하지 않도록 지시하는 힌트이다.

- 문법

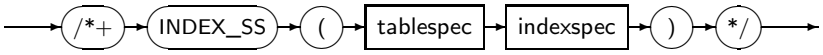
no_index_rs_hint

INDEX_SS

INDEX_SS는 명시한 테이블에 대해 명시한 인덱스를 사용하여 인덱스 스킵 스캔(Index Skip Scan)을 사용하도록 지시하는 힌트이다.

- 문법

index_ss_hint



NO_INDEX_SS

NO_INDEX_SS는 명시한 테이블에 대해 명시한 인덱스를 사용하는 인덱스 스킵 스캔을 사용하지 않도록 지시하는 힌트이다.

- 문법

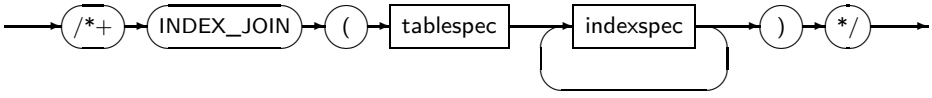
no_index_ss_hint

INDEX_JOIN

INDEX_JOIN은 명시한 테이블에 대해 명시한 두 개 이상의 힌트를 사용하여, 테이블을 스캔할 때 자체 조인(Self Join)을 사용하도록 지시하는 힌트이다.

- 문법

index_join_hint



2.8.4. 조인 순서

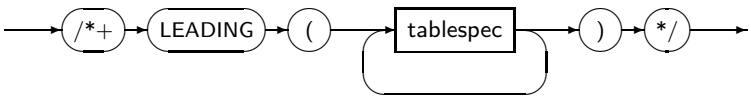
LEADING, ORDERED는 조인 순서를 결정하는 힌트이다. LEADING 힌트가 ORDERED보다 질의 최적화기를 선택할 수 있는 폭이 넓어서 LEADING을 사용하는 것이 좋다.

LEADING

LEADING은 조인에서 먼저 조인되어야 할 테이블의 집합을 명시하는 힌트이다. LEADING 힌트가 먼저 조인될 수 없는 테이블을 포함하는 경우 무시된다. 또, LEADING 힌트끼리 충돌하는 경우에는 LEADING, ORDERED 힌트가 모두 무시된다. 만일 ORDERED 힌트가 사용되는 경우에는 LEADING 힌트는 모두 무시된다.

- 문법

leading_hint



ORDERED

ORDERED는 테이블을 FROM 절에 명시된 순서대로 조인하도록 지시하는 힌트이다. 질의 최적화기는 조인의 결과 집합의 크기에 대한 정보를 추가로 알고 있다. 사용자가 그 정보를 통해 질의 최적화기의 조인 순서를 명확히 알고 있을 경우에만 ORDERED 힌트를 사용하는 것이 좋다.

- 문법

ordered_hint

2.8.5. 조인 방법

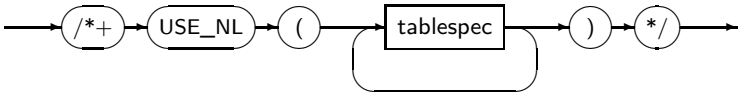
조인 방법이 적용된 힌트는 한 테이블에 대해서만 조인 방법을 지시한다. 조인 방법이 적용된 힌트는 명시한 테이블이 조인의 내부 테이블로 사용될 경우에만 참조된다. 명시한 테이블을 외부 테이블로 사용하는 경우에는 조인 방법이 적용된 힌트는 무시된다.

USE_NL

USE_NL은 명시한 테이블을 다른 테이블과 조인하는 경우 중첩 루프 조인을 사용하도록 지시하는 힌트이다.

- 문법

use_nl_hint

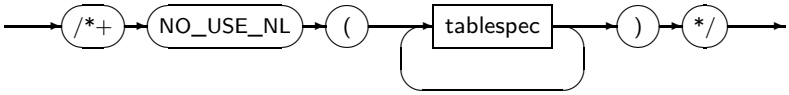


NO_USE_NL

NO_USE_NL은 명시한 테이블을 다른 테이블과 조인하는 경우 중첩 루프 조인을 사용하지 않도록 지시하는 힌트이다. 하지만, 특수한 경우에는 이 힌트가 주어졌더라도 질의 최적화기에서 중첩 루프 조인을 사용하는 플랜을 생성할 수 있다.

- 문법

no_use_nl_hint

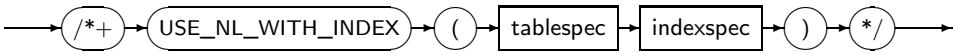


USE_NL_WITH_INDEX

USE_NL_WITH_INDEX는 명시한 테이블을 다른 테이블과 조인하는 경우 중첩 루프 조인을 사용하도록 지시하는 힌트이다. 이때 명시한 테이블에 대한 접근은 명시한 인덱스와 두 테이블에 대한 조인 조건을 이용하여 이루어져야 한다. 만일 인덱스를 사용할 수 없는 경우이면 힌트는 무시된다.

- 문법

use_nl_with_index_hint

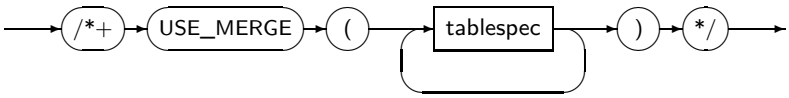


USE_MERGE

USE_MERGE는 명시한 테이블을 다른 테이블과 조인하는 경우 합병 조인을 사용하도록 지시하는 힌트이다.

- 문법

use_merge_hint

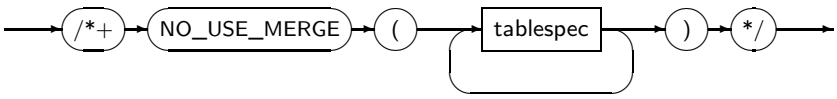


NO_USE_MERGE

NO_USE_MERGE는 명시한 테이블을 다른 테이블과 조인하는 경우 합병 조인을 사용하지 않도록 지시하는 힌트이다.

- 문법

no_use_merge_hint

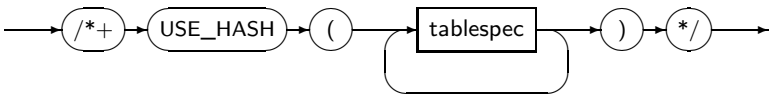


USE_HASH

USE_HASH는 명시한 테이블을 다른 테이블과 조인하는 경우 해시 조인을 사용하도록 지시하는 힌트이다.

- 문법

use_hash_hint

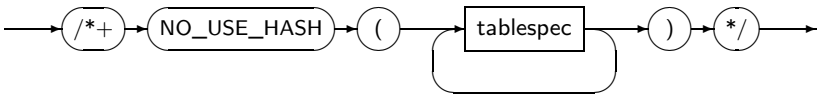


NO_USE_HASH

NO_USE_HASH는 명시한 테이블을 다른 테이블과 조인하는 경우 해시 조인을 사용하지 않도록 지시하는 힌트이다.

- 문법

no_use_hash_hint



HASH_SJ

HASH_SJ는 부질의를 언네스팅할 때 해시방법을 이용한 세미조인으로 하도록 지시하는 힌트이다.

- 문법

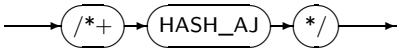
hash_sj_hint

HASH_AJ

HASH_AJ는 부질의를 언네스팅할 때 해시방법을 이용한 안티조인으로 하도록 지시하는 힌트이다.

- 문법

hash_aj_hint



MERGE_SJ

MERGE_SJ는 부질의를 언네스팅할 때 머지방법을 이용한 세미조인으로 하도록 지시하는 힌트이다.

- 문법

merge_sj_hint

MERGE_AJ

MERGE_AJ는 부질의를 언네스팅할 때 머지방법을 이용한 안티조인으로 하도록 지시하는 힌트이다.

- 문법

merge_aj_hint

NL_SJ

NL_SJ는 부질의를 언네스팅할 때 네스티드 루프 방법을 이용한 세미조인으로 하도록 지시하는 힌트이다.

- 문법

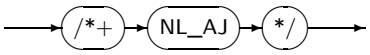
nl_sj_hint

NL_AJ

NL_AJ는 부질의를 언네스팅할 때 네스티드 루프 방법을 이용한 안티조인으로 하도록 지시하는 힌트이다.

- 문법

nl_aj_hint

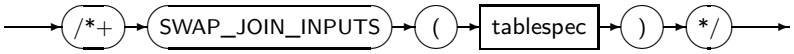


SWAP_JOIN_INPUTS

SWAP_JOIN_INPUTS는 해시 조인을 수행하는 경우 명시한 테이블을 사용하여 해시 테이블을 빌드하도록 지시하는 힌트이다.

- 문법

swap_join_inputs_hint

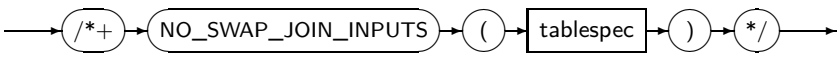


NO_SWAP_JOIN_INPUTS

NO_SWAP_JOIN_INPUTS는 해시 조인을 수행하는 경우 조인 순서가 바뀌는 경우, 명시한 테이블이 해시 테이블로 빌드되지 않도록 지시하는 힌트이다.

- 문법

no_swap_join_inputs_hint



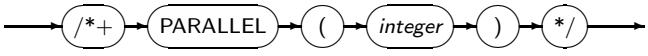
2.8.6. 병렬 처리

PARALLEL

PARALLEL은 지정한 개수의 스레드를 사용해 질의의 수행을 병렬로 진행하도록 지시하는 힌트이다.

- 문법

parallel_hint



NO_PARALLEL

NO_PARALLEL은 질의의 수행을 병렬로 진행하지 않도록 지시하는 힌트이다.

- 문법

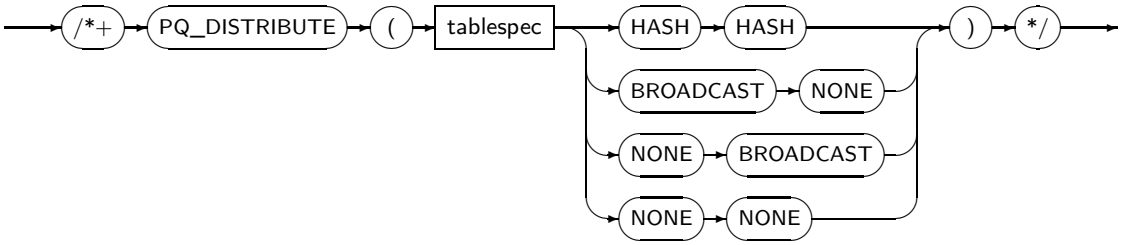
no_parallel_hint

PQ_DISTRIBUTE

PQ_DISTRIBUTE는 조인을 포함한 질의의 병렬 처리에서 조인될 로우의 분산 방법을 지시하는 힌트이다. 분산 방법으로는 HASH-HASH, BROADCAST-NONE, NONE-BROADCAST, NONE-NONE이 있으며 특정한 분산 방법을 선택함으로써 병렬 처리에서 조인의 성능을 향상시킬 수 있다.

- 문법

pq_distribute_hint



다음은 각각의 속성에 대한 설명이다.

속성	설명
NONE NONE	힌트가 없을 때와 같은 플랜이 생성된다.
BROADCAST NONE	조인의 왼쪽은 BROADCAST, 오른쪽은 PE BLOCK ITERATOR의 분산 방법으로 동작한다.
NONE BROADCAST	조인의 왼쪽은 PE BLOCK ITERATOR, 오른쪽은 BROADCAST의 분산 방법으로 동작한다.
HASH HASH	조인의 왼쪽, 오른쪽 모두 HASH의 분산 방법으로 동작한다.

2.8.7. 실체화 뷰

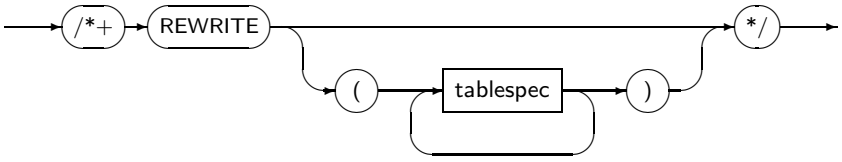
REWRITE

REWRITE는 해당 질의 블록에서 비용의 비교 없이 실체화 뷰를 사용하여 질의의 다시 쓰기를 하도록 지시하는 힌트이다. 따라서 최종으로는 REWRITE 힌트가 사용된 질의 블록만 다시 쓰기를 한 결과와 모든 블록에서 다시 쓰기를 한 결과의 비용을 비교해서 더 좋은 쪽을 질의 최적화기가 선택하게 된다.

실체화 뷰의 목록이 명시된 경우에는 목록에 있는 실체화 뷰만 사용하여 질의의 다시 쓰기를 시도한다.

- 문법

rewrite_hint



NO_REWRITE

NO_REWRITE는 해당 질의 블록에서는 질의의 다시 쓰기를 하지 않도록 지시하는 힌트이다.

- 문법

no_rewrite_hint

MATERIALIZE

MATERIALIZE는 With 절 안에 있는 Subquery를 실체화 뷰(Materialized View)로 만들도록 지시하는 힌트이다.

- 문법

materialize_hint

INLINE

INLINE은 With 절 안에 있는 Subquery를 실체화 뷰(Materialized View)로 만들지 않도록 지시하는 힌트이다.

- 문법

inline_hint

2.8.8. 기타

APPEND

APPEND는 DML 문장에서 직접 데이터 파일에 추가하는 삽입 방법 즉 Direct-Path 방식을 수행하도록 지시하는 힌트이다.

Direct-Path 방식은 일반적인 삽입 방법과 달리 항상 새로운 데이터 블록을 할당받아서 데이터 삽입을 수행하며, 버퍼 캐시를 이용하지 않고 직접 데이터 파일을 추가하기 때문에 성능 향상에 많은 이점이 있다.

- 문법

append_hint

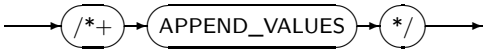
APPEND_VALUES

APPEND_VALUES는 VALUES절을 사용하는 INSERT 문에서 직접 데이터 파일에 추가하는 삽입 방법 즉 Direct-Path 방식을 수행하도록 지시하는 힌트이다.

Direct-Path 방식은 일반적인 삽입 방법과 달리 항상 새로운 데이터 블록을 할당받아서 데이터 삽입을 수행하며, 버퍼 캐시를 이용하지 않고 직접 데이터 파일을 추가하므로 일괄 삽입에 사용하면 성능 향상에 많은 이점이 있다.

- 문법

append_values_hint



NOAPPEND

NOAPPEND는 DML 문장에서 Direct-Path 방식을 수행하지 않도록 지시하는 힌트이다.

- 문법

noappend_hint

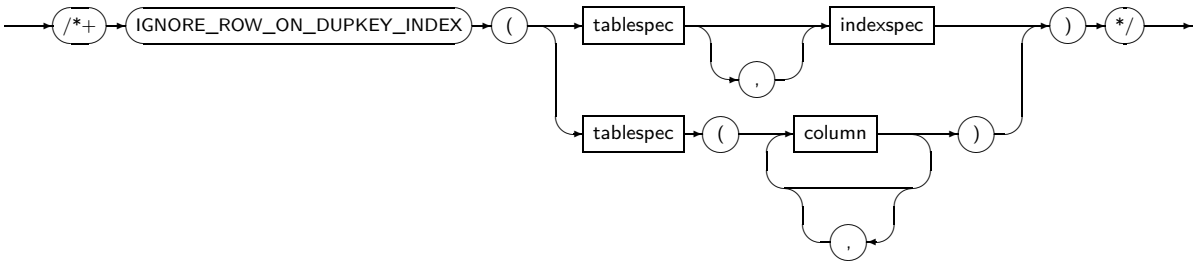
IGNORE_ROW_ON_DUPKEY_INDEX

single table INSERT문에서만 사용이 가능하다. 유일키 제약조건을 위배하면 에러를 발생시키지 않고 삽입하던 로우를 롤백하고 다음 로우부터 삽입을 재개한다.

인덱스를 명시하지 않은 경우, 여러 개의 인덱스를 명시한 경우, 명시된 인덱스가 UNIQUE 속성을 갖지 않는 경우에는 힌트이지만 에러를 발생시킨다. 이 힌트를 명시하면 APPEND, PARALLEL 힌트는 무시된다.

- 문법

ignore_row_on_dupkey_index_hint



CARD

CARD는 쿼리 최적화 할 때에 지정 테이블의 Cardinality를 주어진 값을 이용하여 계산하도록 지시하는 힌트이다.

- 문법

card_hint

MONITOR

MONITOR는 쿼리를 수행할 때 쿼리 수행 정보를 모으도록 지시하는 힌트이다.

- 문법

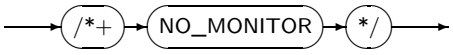
monitor_hint

NO_MONITOR

NO_MONITOR는 쿼리를 수행할 때 쿼리 수행 정보를 모으지 않도록 지시하는 힌트이다.

- 문법

nomonitor_hint



USE_CONCAT

USE_CONCAT은 OR 조건문을 UNION ALL로 쪼개어 OR 양쪽에 대하여 별도의 플랜 노드를 만들어 합치는 OR expansion된 플랜이 만들어지도록 강제하는 힌트이다.

- 문법

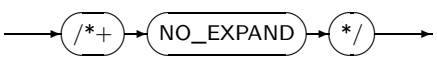
use_concat_hint

NO_EXPAND

NO_EXPAND는 OR expansion을 막음으로써 OR 조건문이 필터로 보존된 플랜을 만들도록 강제하는 힌트이다.

- 문법

no_expand_hint

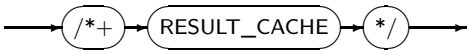


RESULT_CACHE

Query 결과를 Cache에 저장하기 위한 Result Cache를 사용하도록 지시하는 힌트이다. RESULT_CACHE_MODE 초기 파라미터 값이 "MANUAL"일 때만 유효하며, "FORCE"일 때는 힌트의 존재 여부와 관계없이 모든 Query 결과를 Result Cache에 저장한다.

- 문법

result_cache_hint



NO_SUBQUERY_CACHE

NO_SUBQUERY_CACHE는 특정 질의 블록에 대해 부질의 결과를 캐싱하지 않도록 강제하는 힌트이다. Tibero는 가능하다면 부질의 결과 캐싱 기능을 사용하는 실행 계획을 수립한다. 이때 이 힌트를 사용하여 특정 질의 블록에 대해 부질의 결과 캐싱을 강제로 막을 수 있다. NO_SUBQUERY_CACHE 힌트는 대상 질의 블록에 명시하며, 부질의의 실행 계획 최상위 노드로서 CACHE가 생기지 않는다.

- 문법

no_subquery_cache

- 예제

다음은 NO_SUBQUERY_CACHE 힌트를 사용하는 예이다.

```
SELECT * FROM T1
WHERE EXISTS (SELECT /*+ NO_SUBQUERY_CACHE */ * FROM T2 WHERE T1.A = T2.A)
```

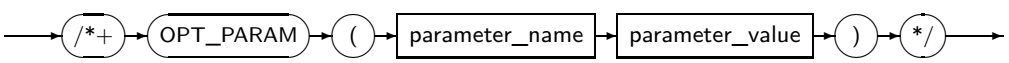
위에 예처럼 EXISTS절 질의 블록에 힌트를 명시할 경우, 해당 블록에 대해 부질의 결과를 캐싱하는 기능을 사용하지 않도록 강제한다.

OPT_PARAM

쿼리가 수행되는 도중 초기화 환경변수를 바꿔서 수행하는 힌트이다. 예를 들어 수행되는 쿼리에 /*+OPT_PARAM(OPTIMIZER_MODE FIRST_ROWS_1)*/ 이라는 힌트를 준다면, 해당 쿼리가 수행되는 동안에 한해서 OPTIMIZER_MODE가 FIRST_ROWS_1로 설정되어 수행된다.

- 문법

opt_param_hint



2.9. 스키마 객체

데이터베이스는 여러 객체로 구성된다. 각 객체는 **'데이터베이스 > 사용자 > 스키마 > 스키마 객체'**의 순으로 포함 관계를 갖는다. 하나의 데이터베이스는 여러 사용자가 공유하고 있다. 또한 사용자 중에는 데이터베이스를 관리하기 위해 특별한 권한을 부여 받고 있는 DBA가 있다.

스키마는 사용자가 소유한 객체의 모임(Collection)이다. Tibero에서는 한 사용자가 하나의 스키마만을 정의할 수 있고, 스키마의 이름은 항상 사용자의 이름과 동일하다. 이러한 스키마에 포함된 객체를 **스키마 객체**라 한다. SQL 표준에서 정의한 스키마 객체 외에, 데이터베이스 시스템에 따라 추가적인 스키마 객체를 제공하고 있다.

Tibero에서 제공하는 스키마 객체는 다음과 같다.

- 테이블(Table)
- 인덱스(Index)
- 뷰(View)
- 시퀀스(Sequence)
- 동의어(Synonym)

2.9.1. 테이블

테이블은 관계형 데이터베이스의 기본 저장 단위이다. 다른 모든 스키마 객체는 테이블을 중심으로 정의된다. 테이블은 2차원 행렬(Matrix)의 형태를 갖는다. 테이블은 하나 이상의 컬럼으로 구성되며 각 컬럼은 고유의 데이터 타입을 갖는다. 하나의 테이블은 0개 이상의 로우를 포함한다. 각 로우는 각 컬럼에 해당하는 값을 갖는다.

참고

테이블을 생성하고 변경, 제거하기 위한 SQL 문장은 각각 CREATE TABLE, ALTER TABLE, DROP TABLE이다. 이러한 문장은 DDL에 포함되며, 자세한 설명은 “제7장 데이터 정의어”를 참고한다.

다음은 회사 직원의 정보를 저장하고 있는 EMP 테이블의 예이다.

[예 2.1] EMP 테이블

```
     EMPNO ENAME        ADDR                SALARY     DEPTNO
---------- ------------ ---------------- ---------- ----------
        35 John         Houston               30000          5
        54 Alicia       Castle                25000          4
        27 Ramesh       Humble                38000          5
        69 James        Houston               35000          4
```

위의 테이블은 'EMPNO, ENAME, ADDR, SALARY, DEPTNO'의 다섯 개의 컬럼으로 구성되며, 네 개의 로우를 포함하고 있다. 테이블을 구성하는 컬럼 정보는 거의 변경되지 않으나, 테이블에 포함된 로우의 개수는 수시로 변경될 수 있다.

컬럼의 데이터 타입 중에는 문자형과 같이 최대 길이를 미리 정해야 하는 것도 있고, NUMBER 타입과 같이 정밀도와 스케일을 정해야 하는 것도 있다. 일부 컬럼에 대해서는 기본값을 선언할 수도 있다. 데이터 타입에 대해서는 “2.1. 데이터 타입”에서 자세하게 설명한다.

테이블 전체 또는 일부 컬럼에 무결성 제약조건(Integrity Constraints)을 선언할 수 있다. 무결성 제약 조건은 테이블에 어떠한 로우가 삽입되든 항상 만족되어야 한다. 예를 들어 테이블 EMP의 한 컬럼 SALARY에 대하여 다음과 같은 조건을 선언할 수 있다.

```
SALARY >= 0
```

직원의 연봉이 0보다 작을 수 없으므로, 이러한 조건을 사용해 잘못된 데이터의 입력을 미리 막을 수 있다. 무결성 제약조건을 표현하기 위한 조건식(Condition Expression)은 “3.4. 조건식”에서 설명한다.

하나의 테이블에 무결성 제약조건을 선언할 수도 있고, 두 개의 테이블에 참조 무결성(Referential Integrity) 제약조건을 선언할 수도 있다.

다음은 회사의 부서 정보를 저장하는 DEPT 테이블의 예이다.

[예 2.2] DEPT 테이블

```
    DEPTNO DNAME        LOC
---------- ------------ ----------------
         1 Accounting   Houston
         4 Research     Spring
         5 Sales        Houston
```

회사의 모든 직원이 하나의 특정 부서에 반드시 소속되어야 한다면, [예 2.1]의 테이블 EMP의 컬럼 DEPTNO의 값은 반드시 [예 2.2]의 테이블 DEPT의 컬럼 DEPTNO에 존재하는 값이어야 한다.

참고

무결성 제약조건에 대한 자세한 내용은 "Tibero 관리자 안내서"를 참고한다.

2.9.2. 인덱스

인덱스는 테이블과 별도의 저장공간을 이용하여 그 테이블의 특정 컬럼을 빠르게 검색 할 수 있도록 해주는 데이터 구조이다. 테이블의 소유자는 어떤 컬럼에 대해 하나 이상의 인덱스를 생성할 수 있다.

참고

인덱스를 생성하고 변경, 제거하기 위한 SQL 문장은 각각 CREATE INDEX, ALTER INDEX, DROP INDEX이다. 이러한 문장은 DDL에 포함되며, 자세한 설명은 "제7장 데이터 정의어"를 참고한다.

다음은 인덱스에 대한 설명이다.

- 자동 인덱싱

  Tibero에서는 모든 테이블의 기본 키(Primary Key) 컬럼에 대해 자동으로 인덱스를 생성한다. 기본 키 컬럼이란 테이블 내의 특정 로우를 유일하게 식별할 수 있는 값을 갖는 컬럼을 의미한다. 하나의 테이블 내에서 어떤 로우도 다른 로우와 동일한 기본 키 컬럼 값을 가질 수 없다. 위의 [예 2.1]와 [예 2.2]에서는 각각 컬럼 EMPNO와 DEPTNO가 기본 키 컬럼이 될 수 있다.

- 컬럼의 중복 허용

  인덱스는 컬럼 값의 중복 유무에 관계 없이 생성할 수 있다. 예를 들어 테이블 DEPT의 컬럼 DEPTNO와 같이 중복이 없는 컬럼이나 테이블 EMP의 컬럼 DEPTNO와 같이 중복이 있는 컬럼에 대해서도 인덱스를 생성할 수 있다.

- 복수 컬럼 허용

  인덱스는 하나의 컬럼뿐만 아니라 둘 이상의 컬럼 값을 하나로 접합하여 생성할 수도 있다. 예를 들어 테이블 EMP의 컬럼 ENAME과 ADDR 값을 합쳐서 인덱스를 만들 수 있다. 이러한 인덱스는 둘 이상의 컬럼이 동시에 검색 대상이 될 확률이 높은 경우에 유용하다.

- 인덱스의 적용

  사용자가 인덱스를 생성하더라도 바로 사용되지는 않는다. 인덱스가 생성되면 SQL 질의를 수행할 때, 질의 최적화기가 인덱스를 사용할 때와 사용하지 않을 때의 실행 효율성을 비교하고, 더 효율적인 방향으로 실제 질의를 실행한다. 따라서, 질의 대상 테이블에 대해서 생성된 인덱스를 이용할 것인지, 어떤 컬럼에 대해서 생성된 인덱스를 이용할 것인지는 데이터베이스 시스템에 의하여 자동적으로 결정된다.

- 인덱스의 관리

  인덱스의 관리도 데이터베이스 시스템에서 자동적으로 이루어진다. 인덱스가 생성된 후에 테이블에 하나의 로우가 삽입되거나 갱신, 삭제되면, 그 테이블에 대하여 생성된 모든 인덱스에서 그 로우에 대한 삽입, 갱신, 삭제가 이루어진다.

- 인덱스의 제거

  인덱스가 더 이상 필요하지 않으면 언제라도 그 인덱스를 제거할 수 있다. 인덱스를 너무 많이 생성해 놓으면 테이블에 대한 갱신이 이루어질 때마다 인덱스에도 함께 반영해 주어야 하므로 성능 저하의 원인이 될 수 있다. 따라서, 불필요한 인덱스는 제거하는 것이 바람직하다.

2.9.3. 뷰

뷰는 SQL 문장에 이름을 붙인 것으로, 빈번히 수행되는 질의의 결과를 테이블 형태로 이용할 수 있도록 정의한 것이다. 뷰는 테이블 또는 다른 뷰를 이용하여 정의할 수 있으며, SQL 문장 내에서 일반 테이블과 동일하게 이용할 수 있다. 뷰가 정의된 테이블을 기반 테이블(base table)이라 한다.

뷰를 정의하는 이유는, 하나의 테이블을 여러 사용자가 함께 접근할 수 있지만 사용자에 따라 테이블 내용의 일부를 숨김으로써 정보 보안을 유지하고자 하는 것이다.

참고

뷰를 생성하고 제거하기 위한 SQL 문장은 CREATE VIEW, DROP VIEW이다. 이러한 문장은 DDL에 포함되며, 자세한 설명은 "제7장 데이터 정의어"를 참고한다.

다음은 뷰에 대한 설명이다.

- 뷰 병합

  데이터베이스 시스템 내에서 뷰는 질의 문장을 문자열 형태로 관리한다. 만약 뷰를 포함한 SQL 문장이 입력되면, 데이터베이스 시스템은 그 문장을 뷰를 포함하지 않는 기반 테이블에 대한 SQL 문장으로 변환한다. 이러한 과정을 뷰 병합이라고 한다.

- 뷰의 활용과 권한

  뷰가 이용되는 경우는 위에서와 같이 빈번한 질의를 단순화하고자 하는 경우 이외에, 뷰를 정의한 기반 테이블의 일부만을 드러내고자 하는 경우에도 사용된다. 예를 들어 테이블 EMP의 내용 중에서 컬럼 SALARY의 내용을 관리직이 아닌 일반 직원에게 드러내고 싶지 않은 경우에, 일반 직원이 테이블 EMP를 직접 접근하지 못하고 뷰를 통해서만 접근하도록 할 수 있다.

  이러한 경우 스키마 객체를 접근할 수 있는 권한(Authority)과 연관이 된다. 즉, 뷰를 정의한 사용자가 일반 직원에게 그 뷰를 접근할 권한만을 부여하고 뷰가 정의된 기반 테이블 EMP를 접근할 권한을 부여하지 않으면, 일반 직원에게 테이블 EMP 내의 데이터 중에서 그 뷰에 의하여 볼 수 있는 일부만을 드러내는 것이다.

2.9.4. 시퀀스

시퀀스는 유일한 연속적인 값을 생성해 낼 수 있는 스키마 객체이다. 이 값은 보통 기본 키 또는 유일 키에 값을 채워 넣을 때 사용된다. 항상 시퀀스의 이름에는 의사 컬럼을 붙여서 사용한다.

SQL 문에서는 다음의 의사 컬럼을 통해 시퀀스의 값을 읽어 들인다.

컬럼	설명
CURRVAL	현재 세션에서 마지막으로 조회한 NEXTVAL 값을 반환한다.
NEXTVAL	시퀀스의 현재 값을 증가시키고 증가된 그 값을 반환한다.

다음은 의사 컬럼과 함께 시퀀스를 사용하는 예이다.

```
seq1.currval
seq2.nextval
```

다음은 위치에 따른 시퀀스 의사 컬럼의 사용 여부에 대해 설명한다.

사용 여부	위치
사용 가능	– SELECT 리스트 부질의 또는 뷰의 SELECT 리스트에서는 사용할 수 없다. – INSERT 문의 부질의의 SELECT 리스트 – INSERT 문의 VALUES 절 – UPDATE 문의 SET 절
사용 불가능	– SELECT, DELETE, UPDATE 문의 부질의의 내부 – 뷰의 내부 – DISTINCT가 있는 SELECT 문 – GROUP BY 절이나 ORDER BY 절이 있는 SELECT 문 – 집합 연산자(UNION, INTERSECT, MINUS)로 다른 SELECT 문과 연결된 SELECT 문 – SELECT 문의 WHERE 절 – CREATE TABLE 또는 ALTER TABLE을 실행할 때 컬럼의 DEFAULT 값 – CHECK 제약 조건

시퀀스를 생성할 때 초기 값과 증감치가 결정된다. NEXTVAL 의사 컬럼을 통해 최초로 시퀀스에 접근하면, 시퀀스는 초기 값을 반환한다. 이후 NEXTVAL을 사용할 때마다 시퀀스의 값은 증감치 만큼 증가하고, 새로 증가된 값을 반환한다. CURRVAL 의사 컬럼은 항상 시퀀스의 현재 값을 반환하며, 이것은 가장 마지막에 사용된 NEXTVAL이 반환한 값과 같다.

CURRVAL 의사 컬럼을 사용하기 전에 적어도 한 번 이상은 시퀀스에 NEXTVAL을 사용해서 초기화를 시켜야 한다. SQL 문장에 사용된 NEXTVAL 의사 컬럼은 SQL 문장이 처리하는 행의 단위별로 시퀀스의 값을 증가시킨다.

시퀀스의 값을 증가시키는 행은 다음과 같다.

- 최상위 레벨의 SELECT 문이 출력하는 행
- INSERT ... SELECT 문에서 SELECT 문이 선택하는 행
- CREATE TABLE ... AS SELECT 문에서 SELECT 문이 선택하는 행
- UPDATE 문이 갱신하는 각각의 행
- VALUES 절을 포함한 INSERT 문이 삽입하는 행

한 행에 두 번 이상의 NEXTVAL이 나왔을 경우 시퀀스 값은 처음 한 번만 증가되며, 증가된 그 값이 다음 위치에 동일하게 사용된다.

NEXTVAL과 CURRVAL이 한 행에 동시에 나왔을 경우 시퀀스 값은 역시 한 번만 증가되며, NEXTVAL에 사용된 값이 CURRVAL에 동일하게 사용된다.

2.9.5. 동의어

동의어는 특정 스키마 객체에 정의하는 일종의 별칭(Alias)이다. 대개 긴 이름을 갖는 스키마 객체에 대하여 짧은 이름의 동의어를 정의하거나, 특별한 용도의 스키마 객체에 대하여 동의어를 정의한다.

다음은 동의어에 대한 설명이다.

- 동의어와 뷰의 차이점

  동의어는 긴 이름 대신 사용한다는 점에서 뷰와 같다. 그러나 뷰는 하나의 완전한 SQL 문장에 대한 것이고, 동의어는 하나의 스키마 객체에 한정된다는 차이점이 있다. 또한 동의어는 뷰와 다르게 접근 권한이 별도로 설정되지 않는다. 만약 특정 스키마 객체에 대한 접근 권한이 있다면 그에 대한 동의어에 대해서도 접근 권한을 갖게 된다.

- 동의어의 사용 설정

  동의어는 기본적으로 동의어를 정의한 사용자만이 사용할 수 있으나, 다른 모든 사용자도 함께 사용하도록 공용으로 정의할 수도 있다. 이렇게 모든 사용자가 사용할 수 있는 동의어를 공유 동의어라고 부른다. Tibero에서는 DBA와 일반 사용자가 데이터 사전(Data Dictionary)의 정보를 쉽게 접근할 수 있도록, 여러 가지 시스템 뷰를 정의하고 각각에 대한 동의어를 정의하고 있다.

참고

동의어를 생성하고 삭제하기 위한 SQL 문장은 CREATE SYNONYM과 DROP SYNONYM이다. 동의어를 변경하기 위한 별도의 SQL 문장은 없다. 이러한 문장은 DDL에 포함되며, 자세한 설명은 “제 7장 데이터 정의어”를 참고한다.

2.9.6. 스키마 객체의 이름

스키마 객체에 따라 사용자가 각각의 부분 또는 전체에 이름을 부여할 수 있거나 꼭 부여해야만 하는 경우가 있다. 테이블, 테이블의 컬럼, 인덱스, 무결성 제약조건, 테이블 파티션, 테이블 서브 파티션, 인덱스 파티션, 인덱스 서브파티션, 패키지와 패키지 내의 함수와 프러시저 등이 이에 해당한다.

SQL 문장에서 스키마 객체의 이름을 명시할 때는 '따옴표 있는 식별자' 또는 '따옴표 없는 식별자'를 사용한다.

- 따옴표 있는 식별자는 큰따옴표(" ")로 시작해서 큰따옴표로 끝난다.
- 따옴표 없는 식별자는 큰따옴표로 열고 닫지 않는다.

스키마 객체에 이름을 부여할 때는 이 두 개 중 어떤 것을 쓰더라도 상관없다. 따옴표 없는 식별자는 대소문자를 구별하지 않으며, 모두 대문자로 간주되어 처리된다. 따옴표 있는 식별자는 대소문자를 구분한다.

예를 들어 다음의 경우는 모두 서로 다른 식별자이다.

```
department
"department"
"Department"
```

다음의 경우는 모두 같은 식별자이다.

```
department
DEPARTMENT
"DEPARTMENT"
```

식별자를 기술할 때는 다음의 규칙을 지켜야 한다.

- 길이가 30byte를 넘어서는 안 된다.
- 예약어는 따옴표 없는 식별자가 될 수 없다. 따옴표를 적용하여 예약어를 식별자로 만들 수 있으나, 권장하지는 않는다. 예약어는 “Appendix A. 예약어”를 참고한다.
- 따옴표 없는 식별자는 알파벳, 한글, 숫자, 언더바(_), $, #만 사용할 수 있다. 다만, 숫자, '$', '#'는 첫 글자로 올 수 없다.
- 따옴표 있는 식별자는 공백을 포함한 어떤 문자라도 쓸 수 있다. 다만 큰따옴표(" ")는 사용할 수 없다.
- 하나의 네임스페이스 안에서 서로 다른 두 객체가 동일한 이름을 가질 수 없다.

구분	네임스페이스	종류
스키마 객체	하나의 네임스페이스	테이블, 뷰, 실체화 뷰, 시퀀스, 동의어, 패키지, 패키지에 포함되지 않은 함수 및 프러시저
	독립적인 네임스페이스	인덱스, 트리거, 대용량 객체형 데이터 타입

구분	네임스페이스	종류
		(서로 다른 스키마의 네임스페이스는 공유되지 않는다. 그러므로 서로 다른 스키마에 있는 두 개의 테이블은 같은 이름을 가질 수 있다.)
일반 객체	독립적인 네임스페이스	사용자 역할, 공유 동의어, 테이블 스페이스

- 하나의 테이블 내의 서로 다른 컬럼은 동일한 이름을 가질 수 없다. 서로 다른 테이블에 속해 있는 컬럼은 동일한 이름을 가질 수 있다.
- 한 패키지 내의 서로 다른 프러시저나 함수는 인자의 개수나 데이터 타입이 다른 경우에 한해서 동일한 이름을 가질 수 있다.

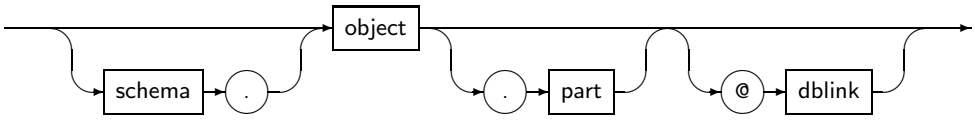
2.9.7. 스키마 객체 관련 문법

SQL 문장에서 스키마 객체와 객체의 구성 요소를 명시하는 방법을 설명한다.

스키마 객체를 명시하는 문법의 세부 내용은 다음과 같다.

- 문법

schema_object_or_part



- 구성요소

구성요소	설명
schema	객체를 포함하고 있는 스키마의 이름이다. 이 schema 부분을 명시해 사용자가 소유한 스키마가 아닌 다른 사용자의 스키마에 있는 객체를 지정할 수가 있다. 다른 사용자가 소유한 스키마 객체에 접근하기 위해서는 권한을 부여받아야 한다. schema를 생략하면 자기 자신의 스키마에 있는 객체를 명시하는 것으로 인식된다. 스키마 객체에만 schema라는 한정어를 사용할 수가 있다. 스키마 객체와 스키마 객체가 아닌 일반 객체의 종류 및 네임스페이스에 대해서는 “2.9.6. 스키마 객체의 이름”에 설명되어 있다. 참고로 공유 동의어에 대해서는 "PUBLIC"이라는 한정어를 사용한다. PUBLIC을 사용할 때는 큰따옴표(" ")를 꼭 사용해야만 한다.
object	객체의 이름이다.

구성요소	설명
part	객체의 구성 요소를 명시할 때 사용된다. 예를 들면 테이블의 구성 요소로는 컬럼, 파티션 등이 있다.
dblink	분산 데이터베이스 기능을 사용할 때 필요하다. object 부분에 명시한 객체를 포함하고 있는 데이터베이스의 이름을 명시한다. dblink를 사용해 자신의 데이터베이스가 아닌 원격 데이터베이스에 있는 객체를 명시할 수가 있다. 이 한정어를 생략하면 사용자의 데이터베이스에 있는 객체를 명시하는 것으로 인식된다.

스키마 객체를 찾는 과정

SQL 문장 내에 명시된 스키마 객체를 찾는 과정은 다음과 같다.

1. SQL 문장의 문맥에 따라 네임스페이스가 결정된다.
2. 결정된 네임스페이스 내에서 객체를 찾는다.
3. 객체의 타입이 SQL 문장 내의 객체가 명시된 위치에서 사용될 수 있는지 확인한다.
4. 사용될 수 없는 타입이라면 에러를 반환한다.

다음은 위의 과정을 설명하는 예이다.

```
SELECT * FROM employees;
```

위의 SQL 문장을 입력했다고 가정하고, 위의 문장을 예로 스키마 객체를 찾는 과정을 설명하면 다음과 같다. 각각의 숫자는 스키마 객체를 찾는 과정에서 각 단계를 설명할 때 사용된 숫자와 동일한 단계이다.

1. 위의 SQL 문장은 스키마를 명시하지 않았으므로 사용자가 소유한 스키마를 네임스페이스로 결정한다.
2. 결정된 네임스페이스에서 즉, 사용자가 소유한 스키마 내에서 employees라는 객체가 있는지 찾는다. 만일 찾지 못했다면 employees라는 공유 동의어가 존재하는지 확인한다. 공유 동의어에도 employees라는 객체가 없다면 에러를 반환한다.

   만약 찾은 객체가 동의어라면 동의어의 정의를 따라가서 동의어가 정의하고 있는 스키마 객체가 동의어인지 아닌지 확인한다. 동의어라면 또 그 동의어의 정의를 따라간다. 동의어가 아닌 스키마 객체가 나올 때까지 이 과정을 반복한다.
3. 해당 객체를 찾았으면 그 객체가 테이블인지 아니면 뷰인지 확인한다. FROM 절에는 테이블 또는 뷰만 올 수 있다.
4. 만약 테이블 또는 뷰가 아닌 다른 객체라면 에러를 반환한다.

원격 데이터베이스의 객체를 찾는 과정

원격 데이터베이스에 있는 객체를 명시하기 위해서는 객체 이름 다음에 "@" 표시와 함께 데이터베이스 링크(Database link) 이름을 명시한다. 데이터베이스 링크는 원격 데이터베이스와의 연결을 담당하는 스키마 객체이다. 데이터베이스 링크는 “7.26. CREATE DATABASE LINK”를 사용하여 생성한다.

SQL 문장에서 데이터베이스 링크를 사용했을 경우 해당 객체를 찾는 과정은 다음과 같다.

1. 먼저 사용자의 스키마에서 해당 이름을 갖는 데이터베이스 링크가 존재하는지 찾는다.
2. 존재하지 않으면 공유 데이터베이스 링크 중에서 해당 이름을 찾는다.
3. 해당 데이터베이스 링크를 찾았으면, 그 데이터베이스 링크를 이용해 원격 데이터베이스의 객체에 접근을 시도한다.
4. 해당 원격 데이터베이스에서 스키마 객체를 찾는 과정은 "스키마 객체를 찾는 과정"과 동일하다.

제3장 SQL 연산

본 장에서는 SQL 연산에 대해 설명한다. 먼저 연산자에 대해 설명하고 다음으로 일반 연산식과 조건식에 대해 설명한다. 각 연산식을 설명할 때는 도식적 문법을 사용하여 설명한다.

3.1. 개요

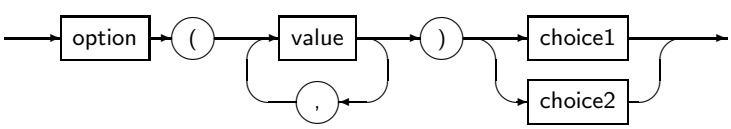
SQL 문장 내에는 여러 가지 연산식이 포함된다. 연산식은 연산자(Operator), 피연산자(Operand), 함수(Function) 등으로 구성된다. 연산식은 연산식 내의 모든 피연산자에 어떤 값을 대입하면 해당 연산 결과를 반환한다.

연산식은 크게 두 가지로 구분할 수 있는데, 일반 연산식과 조건식이 이에 해당한다.

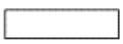
- 일반 연산식
  - 논리(Logical) 연산자와 비교(Comparison) 연산자를 제외한 모든 연산자가 포함된다.
  - 결과 값으로 임의의 산술 값, 문자열, 리스트 등이 반환된다.
- 조건식
  - 논리 연산자 또는 비교 연산자가 포함된다.
  - 결과 값으로 TRUE, FALSE, UNKNOWN이 반환된다.

다음은 연산식의 문법을 나타내는 그림이다.

[그림 3.1] 연산식 문법



위의 그림을 통해 도식을 해석하는 방법을 설명하면 다음과 같다.

도형	설명
example	전체 문법을 대표하는 이름은 왼쪽 위에 나타낸다. [그림 3.1]의 문법을 대표하는 이름은 example이다.
	사각형 내의 문자는 문장에 포함된 문법 요소를 나타내며, 적절한 다른 문자열로 대치되어야 한다.

도형	설명
	[그림 3.1]에서는 option, value, choice1, choice2 등이 여기에 해당한다.
◯	동그라미 안의 문자는 키워드(Keyword) 또는 연산식 기호로, 반드시 문장 내에 그대로 포함되어야 한다. [그림 3.1]에서는 소괄호(( ))와 콤마(,)가 여기에 해당한다.
⟶	문장을 완성하는 순서는 화살표를 따라가면 된다. 뒤쪽으로 향하는 화살표는 0번 이상 포함됨을 의미한다. 여러 갈래로 갈라지는 화살표는 여러 가지 중에 하나를 선택해야 함을 의미한다. [그림 3.1]에서는 option은 반드시 포함되어야 하며, 콤마(,)는 포함되지 않거나 한 번 이상 포함될 수 있다. 소괄호(( ))는 반드시 포함된다. value는 반드시 한 번 이상 포함되어야 하며, 콤마가 포함되면 value는 한 번 더 포함된다. choice1과 choice2 중의 하나만 문장에 포함된다.

다음에 나열된 문장은 위의 [그림 3.1]의 도식을 바탕으로 완성된 유효한 연산식의 예이다.

```
option1 (value1) choice1
option1 (value1, value2) choice1
option1 (value1, value2) choice2
option1 (value1) choice2
```

3.2. 연산자

본 절에서는 SQL 문장 내에서 사용되는 연산자에 대해 설명한다.

각 연산자를 포함한 연산식에 대해서는 “3.3. 연산식”과 “3.4. 조건식”에서 설명한다.

3.2.1. 일반 연산자

SQL 문장 내에서 사용되는 일반 연산자는 산술(Arithmetic) 연산자, 문자열(String) 연산자, 집합(Set) 연산자 등이 있다.

- 산술 연산자

  산술 연산자는 사칙연산을 수행하기 위한 연산자이다. 산술 연산자에는 덧셈, 뺄셈, 곱셈, 나눗셈을 위한 네 가지 이항(Binary) 연산자와 양수, 음수를 나타내기 위한 단항(Unary) 연산자가 있다.

- 문자열 연산자

  문자열 연산자는 접합(Concatenate) 연산자 하나뿐이다. 접합 연산자(||)는 두 개의 문자열을 하나로 연결하기 위한 연산자이다.

다음은 두 개의 문자열 'ABC'와 'DEF'를 접합 연산자를 사용해 하나로 접합시키는 예이다.

```
'ABC' || 'DEF' = 'ABCDEF'
```

- 집합 연산자

집합 연산자는 두 개의 질의 결과에 대한 연산자이며, 피연산자로 항상 SELECT 문의 결과를 받는다. 집합 연산자는 테이블을 대상으로 직접 수행되지 않는다. 집합 연산자는 연산 결과 0개 이상의 로우로 구성된 하나의 테이블을 반환한다.

다음의 표는 Tibero에서 제공하는 집합 연산자와 각 연산자의 연산 결과를 설명한다.

연산자	연산 결과
UNION	두 질의의 결과를 하나로 합친다. 질의의 결과에 중복되어 포함된 로우에 대해서는 연산 결과에 한 번만 포함한다. 교환법칙(Commutative Rule)이 성립한다.
UNION ALL	두 질의의 결과를 하나로 합친다. 하나의 로우가 두 질의의 결과에 각각 m번과 n번이 포함되어 있다면, 연산 결과에는 (m+n)번 포함된다. 교환법칙(Commutative Rule)이 성립한다.
INTERSECTION	두 질의의 결과에 공통적으로 포함된 로우를 추출한다. 질의의 결과에 중복되어 포함된 로우에 대해서는 연산 결과에 한 번만 포함한다. 교환법칙(Commutative Rule)이 성립한다.
MINUS	앞 질의의 결과로부터 뒤 질의의 결과에 포함된 모든 로우를 제거한다. 질의의 결과에 중복되어 포함된 로우에 대해서는 연산 결과에 한 번만 포함한다. 교환법칙(Commutative Rule)이 성립하지 않는다.

산술 연산자와 문자열 연산자를 사용할 때는 연산 결과가 미리 예측한 범위 내의 값이 되는지 확인해야 한다. 예를 들어 NUMBER(3,0) 타입으로 선언된 컬럼에 산술 연산 결과 1234.0을 저장하려 하거나, VARCHAR(5) 타입의 컬럼에 접합 연산 결과 'ABCDEF'를 저장하려 하면 에러가 발생한다.

산술 데이터와 문자열 데이터에 대한 복잡한 연산은 별도의 내장 함수(Built-in Function)로 수행할 수 있다. Tibero에서 제공하는 내장 함수는 “제4장 함수”에서 자세하게 설명한다.

3.2.2. 조건식에 포함되는 연산자

조건식에 포함되는 연산자는 논리 연산자와 비교 연산자가 있다.

- 논리 연산자

논리 연산자는 피연산자로 항상 조건식이 온다. Tibero에서 제공하는 논리 연산자는 NOT, AND, OR이다. NOT 연산자는 단항 연산자이며, AND와 OR 연산자는 이항 연산자이다.

Tibero에서의 논리 연산자는 일반적인 논리 연산자와 다른 점이 있다. 논리 연산의 결과로 TRUE, FALSE 외에 UNKNOWN이 반환될 수 있다는 점이다.

아래의 표는 각각 NOT, AND, OR 연산자에 대하여 피연산자가 TRUE, FALSE, UNKNOWN 값을 가질 때의 진리표(Truth Table)을 보인 것이다. AND와 OR 연산자는 교환법칙이 성립한다.

– NOT 연산자의 진리표

	TRUE	**FALSE**	**UNKNOWN**
NOT	FALSE	TRUE	UNKNOWN

– AND 연산자의 진리표

AND	**TRUE**	**FALSE**	**UNKNOWN**
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

– OR 연산자의 진리표

OR	**TRUE**	**FALSE**	**UNKNOWN**
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

- 비교 연산자

비교 연산자는 피연산자로 임의의 산술 값이나 문자열, 리스트 등이 올 수 있다. 비교 연산자는 단일 값 간의 비교와 단일 값과 리스트와의 비교를 위한 연산자로 나눌 수 있다.

다음의 표는 Tibero에서 제공하는 단일 값 간의 산술 비교 연산자를 나타낸다.

연산자	**설명**
=	두 값이 서로 같은지를 나타내는 이항 연산자
!=, ^=, ~=, <>	두 값이 서로 다른지를 나타내는 이항 연산자
>, <	두 값의 대소비교를 통해 크거나 작은지를 나타내는 이항 연산자
>=, <=	두 값의 대소비교를 통해 크거나 같은지 또는 작거나 같은지를 나타내는 이항 연산자

단일 값과 리스트와의 비교를 위한 연산자로는 산술 비교 연산자와 ANY, SOME, ALL이 결합된 연산자(> ALL, = SOME 등)과 IN 연산자 등이 있다. 이러한 비교 연산자를 포함하는 조건식의 의미와 문법에 대해서는 “3.4. 조건식”에서 구체적으로 설명한다.

- 연산자 우선순위

  하나의 연산식 내에 여러 연산자가 혼용될 수 있다. 이러한 경우 어떤 연산자를 먼저 계산할 것인지에 대한 순서가 정해져 있다. 이러한 순서를 연산자 우선순위(Operator Precedence)라고 한다.

  다음의 표는 Tibero에서 제공하는 연산자 간의 우선순위를 나타낸다. 위쪽의 연산자일수록 우선순위가 높다.

우선순위	연산자	설명
1	+, -	숫자의 부호를 나타내는 단항 연산자
2	*, /	산술 연산자
3	+, -	산술 연산자
4	=, !=, ^=, ~=, <>, <, >, <=, >=	비교 연산자
5	LIKE, BETWEEN, IN, EXISTS, IS NULL	비교 연산자
6	NOT	논리 연산자(단항 연산자)
7	AND	논리 연산자
8	OR	논리 연산자

  산술 연산자(+, -, *, /)와 논리 연산자(AND, OR)의 경우 여러 개가 나열될 수 있다. 이런 경우 우선순위가 같다면 왼쪽에 있는 연산자부터 먼저 계산한다. 비교 연산자와 단항 연산자는 여러 개가 나열될 수 없다. 연산자 간의 우선순위에 관계 없이 괄호로 둘러싸인 식이 먼저 계산된다. 중첩된 괄호의 경우 안쪽 괄호의 식을 먼저 계산한다. 계산 순서를 확실히 나타내기 위한 경우에 괄호를 사용하는 것이 좋다.

3.3. 연산식

연산식은 여러 가지 SQL 문장 내에서 사용된다. 본 장의 개요 부분에서 설명했듯이 연산식은 연산자와 피연산자, 함수 등으로 구성된다.

다음은 연산식을 포함하는 SQL 문장이다.

- SELECT 문 내의 SELECT 절(리스트)
- SELECT, UPDATE, DELETE 문 내의 WHERE 절(조건식)
- SELECT 문 내의 HAVING 절(조건식)
- SELECT 문 내의 ORDER BY 절(리스트)
- INSERT 문 내의 VALUES 절(리스트)
- UPDATE 문 내의 SET 절(할당 연산식)

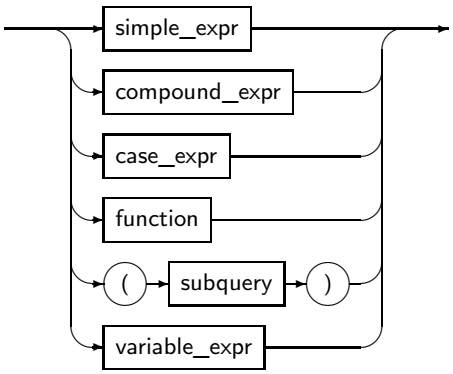
리스트 형태의 연산식인 경우에 SELECT 문의 SELECT 절과 ORDER BY 절에서는 괄호로 묶여지지 않으나, INSERT 문의 VALUES 절에서는 반드시 괄호로 묶어야 한다. UPDATE 문의 SET 절에 포함되는 할당 연산식은 등호(=)를 이용한 비교 연산식과 동일한 문법을 가지나, 등호의 왼쪽에는 반드시 컬럼의 이름이 와야 한다.

Tibero에서 사용할 수 있는 연산식은 단순 연산식(Simple Expression), 복합 연산식(Compound Expression), 함수(Function), 부질의 연산식(Subquery Expression), 리스트(List) 등이 있다.

연산식의 세부 내용은 다음과 같다.

- 문법

expr



- 구성요소

구성요소	설명
simple_expr	단순 연산식을 의미한다. 자세한 내용은 "3.3.2. 단순 연산식"을 참고한다.
compound_expr	복합 연산식을 의미한다. 자세한 내용은 "3.3.3. 복합 연산식"을 참고한다.
case_expr	CASE 연산식을 의미한다. 자세한 내용은 "3.3.4. CASE 연산식"을 참고한다.
function	함수를 의미한다. 자세한 내용은 "3.3.5. 함수"를 참고한다.
subquery	부질의 연산식을 의미한다. 부질의 연산식은 항상 소괄호(( ))로 묶인다. 자세한 내용은 "3.3.6. 부질의 연산식"을 참고한다.
variable_expr	변수를 의미한다. 자세한 내용은 "3.3.7. 변수"를 참고한다.

3.3.1. 연산식의 변환

연산식 내의 피연산자의 데이터 타입이 연산자가 요구하는 타입이 아닌 경우 Tibero에서는 가능하면 데이터 타입의 변환을 수행한다.

다음의 두 라인은 데이터 타입의 변환의 예이다.

```
'30' + 50 = 80

'YEAR' || 2004 = 'YEAR2004'
```

첫 번째 라인에서 덧셈(+) 연산자의 피연산자는 NUMBER 타입이어야 하나 문자열 '30'이 온 경우 이를 먼저 NUMBER 타입으로 변환한 후에 계산하게 된다. 두 번째 라인에서 접합(||) 연산자의 피연산자는 문자열 타입이어야 하나 NUMBER 타입의 2004가 온 경우 이를 먼저 문자열로 변환한 후에 연산을 수행한다.

모든 데이터 타입 간의 변환이 가능한 것은 아니며, 연산식의 계산 과정에서 데이터 타입의 변환이 불가능한 경우에는 에러를 반환한다. 데이터 타입의 변환이 가능하더라도 실제 값에 따라 에러가 발생할 수 있다. 예를 들어 문자열 '1234'는 NUMBER 타입으로 변환 가능하나, 'ABC'는 변환 불가능하다. 대용량 객체는 데이터 타입의 변환이 불가능하다.

다음의 표는 데이터 타입의 변환이 가능한 조합을 보여주고 있다. 세로로 나열된 타입이 변환 전의 타입이며, 가로로 나열된 타입이 변환 후의 타입이다.

데이터 타입	NUMBER	CHAR	VARCHAR	RAW	DATE	TIME	TIMESTAMP	INTERVAL	LONG	LONG RAW	BLOB	CLOB	ROWID	BINARY_FLOAT	BINARY_DOUBLE
NUMBER	-	O	O	-	-	-	-	-	O	-	-	O	-	O	O
CHAR	O	-	O	O	O	O	O	O	O	O	O	O	O	O	O
VARCHAR	O	O	-	O	O	O	O	O	O	O	O	O	O	O	O
RAW	-	O	O	-	-	-	-	-	O	O	O	O	-	-	-
DATE	-	O	O	-	-	-	O	-	O	-	-	-	-	-	-
TIME	-	O	O	-	-	-	-	-	O	-	-	-	-	-	-
TIMESTAMP	-	O	O	-	O	-	-	-	O	-	-	-	-	-	-
INTERVAL	-	O	O	-	-	-	-	-	O	-	-	-	-	-	-
LONG	-	O	O	O	-	-	-	-	-	O	-	O	-	-	-
LONG RAW	-	O	O	O	-	-	-	-	O	-	O	-	-	-	-
BLOB	-	-	-	O	-	-	-	-	-	O	-	-	-	-	-
CLOB	-	O	O	-	-	-	-	-	O	-	-	-	-	-	-
ROWID	-	O	O	-	-	-	-	-	O	-	-	-	-	-	-
BINARY_FLOAT	O	O	O	-	-	-	-	-	-	-	-	-	-	-	O

데이터 타입	NUMBER	CHAR	VARCHAR	RAW	DATE	TIME	TIMESTAMP	INTERVAL	LONG	LONG RAW	BLOB	CLOB	ROWID	BINARY_FLOAT	BINARY_DOUBLE
BINARY_DOUBLE	O	O	O	-	-	-	-	-	-	-	-	-	-	O	-

3.3.2. 단순 연산식

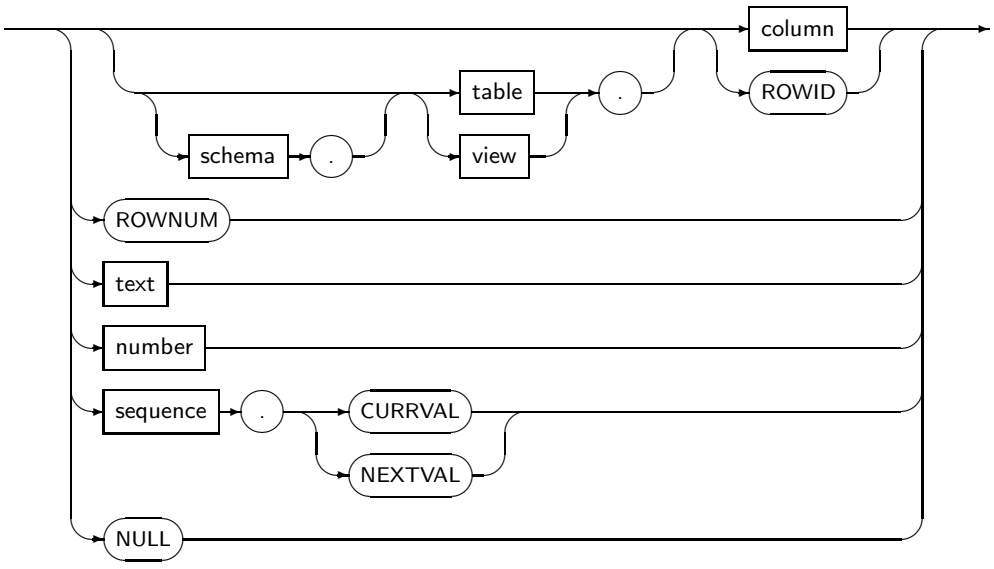
단순 연산식은 연산자 또는 함수를 포함하지 않고 하나의 피연산자만으로 이루어진 연산식이다. 도식에 나타난 것처럼 피연산자는 특정 테이블 컬럼, ROWID 타입, 문자열 또는 수치 값, NULL 값 등이다.

단순 연산식에서 스키마 이름, 테이블과 뷰의 이름, 컬럼 이름, NULL은 영문 대문자와 소문자를 구분하지 않는다. 반면에, 작은따옴표(' ')로 둘러싸인 문자열은 대소문자를 구분한다.

단순 연산식의 세부 내용은 다음과 같다.

- 문법

simple_expr



- 구성요소

구성요소	설명
schema	스키마의 이름을 의미한다.
table	테이블의 이름을 의미한다.
view	뷰의 이름을 의미한다.

구성요소	설명
column	컬럼의 이름을 의미한다.
text	문자열을 의미한다.
number	수치 데이터 값을 의미한다.
sequence	시퀀스의 이름을 의미한다.

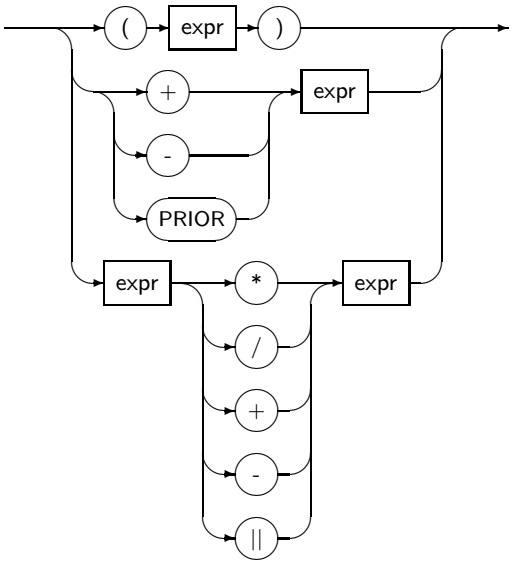
3.3.3. 복합 연산식

복합 연산식은 하나 이상의 연산식을 결합한 복잡한 형태의 연산식이다.

복합 연산식의 세부 내용은 다음과 같다.

- 문법

compound_expr



- 구성요소

구성요소	설명
expr	expr은 일반적인 연산식을 의미한다. 자세한 내용은 "3.3. 연산식"을 참고한다.
+, -	숫자의 부호를 나타내는 단항 연산자(+, -)를 의미한다.
PRIOR	로우 간의 상하 관계를 나타내기 위한 연산자이다. 자세한 내용은 "5.5. 계층 질의"를 참고한다.
*, /, +, -, ||	산술 연산자(*, /, +, -)와 문자열 접합 연산자(||)를 의미한다.

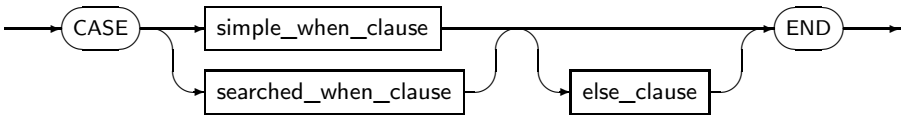
3.3.4. CASE 연산식

CASE 연산식은 SQL 문장에서 IF... THEN ... ELSE 로직을 표현한다. simple_when_clause를 사용하면 expr, comparison_expr, return_expr 그리고 else_clause를 합해서 최대 65535까지 표현식을 사용할 수 있다. searched_when_clause를 사용하면 condition, return_expr, else_clause를 합해서 최대 65535까지 표현식을 사용할 수 있다.

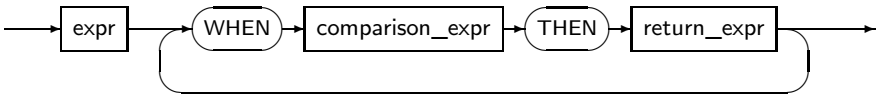
CASE 연산식의 세부 내용은 다음과 같다.

- 문법

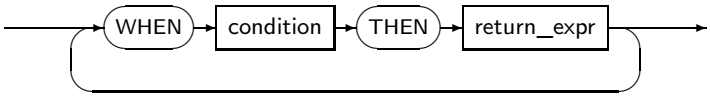
case_expr



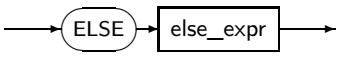
simple_when_clause



searched_when_clause



else_clause



- 구성요소

구성요소	설명
simple_when_clause	WHEN ... THEN 쌍에서 명시된 순서대로 comparison_expr과 expr을 비교하고 결과가 같다면 해당 THEN 절의 return_expr을 반환한다. 모든 comparison_expr과의 비교가 실패한다면. else_clause에 있는 else_expr이 반환된다. else_clause가 없다면 NULL이 반환된다.
searched_when_clause	WHEN ... THEN 쌍에 대해서 명시된 순서대로 condition을 평가하게 되고 결과가 TRUE이면 해당 THEN 절의 return_expr을 반환한다. TRUE로 평가되는 condition이 없다면 else_clause에 명시된 else_expr이 반환된다. else_clause가 없다면 NULL이 반환된다.

구성요소	설명
comparison_expr	비교 연산자를 의미한다. expr과 모든 comparison_expr은 동일한 타입이어야 한다. 자세한 내용은 “3.2.2. 조건식에 포함되는 연산자”를 참고한다.
condition	조건식을 의미한다. 자세한 내용은 “3.4. 조건식”을 참고한다.
return_expr	simple_when_cluase의 비교 결과와 searched_when_clause의 조건식을 만족할 경우에 return_expr을 반환한다. 모든 return_expr과 else_expr도 동일한 타입이어야 한다.
else_clause	simple_when_clause의 비교 결과와 searched_when_cluase의 조건식을 만족하지 못할 경우에 else_clause에 있는 else_expr이 반환된다.

- 예제

  다음은 CASE 연산식의 예이다.

```
SELECT CASE WHEN age > 19 THEN 'adult' ELSE 'minor' END FROM people;
```

3.3.5. 함수

함수 연산식은 함수 이름과 괄호 안에 포함된 0개 이상의 파라미터로 구성된다. 파라미터는 콤마(,)로 구분하며, 파라미터가 0개인 경우에는 괄호를 생략할 수도 있다. Tibero에서는 다양한 단일 로우 함수(Single Row Function)와 집단 함수(Aggregate Function)를 제공하고 있다.

다음은 함수를 사용한 예이다.

```
ROUND(123.456, 2)
LENGTH(ADDR)
SYSDATE
AVG(EMP.SALARY)
```

3.3.6. 부질의 연산식

부질의 연산식은 연산식 내에 SELECT 문이 포함된 것을 말한다. 일반적으로, SELECT 문은 1개 이상의 컬럼 값을 갖는 0개 이상의 로우를 반환한다. 하지만, 연산식 내에 포함되는 SELECT 문은 반드시 1개의 컬럼 값을 갖는 1개의 로우를 반환해야 한다. 만약 이러한 조건이 충족되지 않으면 연산식이 계산되는 중에 에러를 반환한다.

다음은 부질의 연산식의 예이다.

```
(SELECT MAX(SALARY) FROM EMP WHERE DEPTNO = 5) * 1.05
```

부질의 연산식은 항상 괄호 안에 둘러싸여야 한다. SELECT 문의 SELECT 절에 1개의 컬럼만 포함되어 있고, 질의 실행 결과 최댓값 1개만 반환된다. 따라서, 에러 없이 계산할 수 있다.

부질의 연산식은 SQL 문장 내에서 단순 연산식이 올 수 있는 대부분의 위치에 올 수 있다. 하지만, 아래와 같은 위치에는 부질의 연산식이 올 수 없다.

- SELECT 문의 GROUP BY 절
- 컬럼의 디폴트 값
- CHECK 제약조건의 조건식

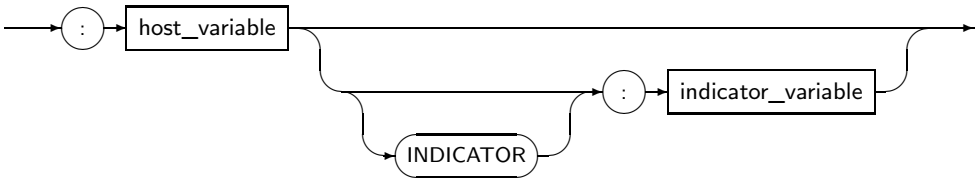
3.3.7. 변수

변수는 호스트 변수와 지시자, 지시자 변수로 구성된다. tbESQL 문장에 사용하며 프로그램의 데이터 입출력에 사용된다.

변수의 세부 내용은 다음과 같다.

- 문법

variable_expr



- 구성요소

구성요소	설명
host_variable	호스트 변수를 의미한다. 콜론(:)과 함께 쓰인다. 데이터베이스에서는 데이터의 처리 결과를 저장하기 위해 사용하고, 애플리케이션 프로그램에서는 변수로 사용된다.
INDICATOR	지시자를 의미한다.
indicator_variable	지시자 변수를 의미한다. 호스트 변수와 관계되어 사용하는 변수로 출력되는 호스트 변수의 NULL 값 확인 및 에러 검사를 하기 위해 사용한다. 호스트 변수와 마찬가지로 콜론(:)과 함께 쓰인다.

참고

변수에 관한 자세한 내용은 사용하는 프로그래밍 언어에 따라 "Tibero tbESQL/C 안내서"와 "Tibero tbESQL/COBOL 안내서"를 참고한다.

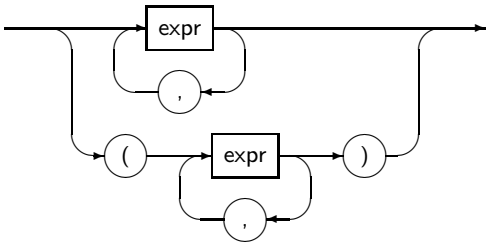
3.3.8. 리스트

리스트는 하나 이상의 연산식을 콤마(,)로 구분하여 나열한 것이다. 리스트는 SQL 문장 내의 위치에 따라 괄호에 둘러싸일 수도 있다. SELECT, UPDATE, DELETE 문장의 WHERE 절에 포함되는 리스트와 INSERT 문장의 VALUES 절에 포함되는 리스트는 괄호에 둘러싸인다.

리스트의 세부 내용은 다음과 같다.

- 문법

expr_list



- 구성요소

구성요소	설명
expr	expr은 일반적인 연산식을 의미한다. 자세한 내용은 “3.3. 연산식”을 참고한다.

- 예제

다음은 조건식의 예이다.

```
EMPNO, ENAME, ADDR
(35, 'John', 'Houston')
(20, 30, 50)
```

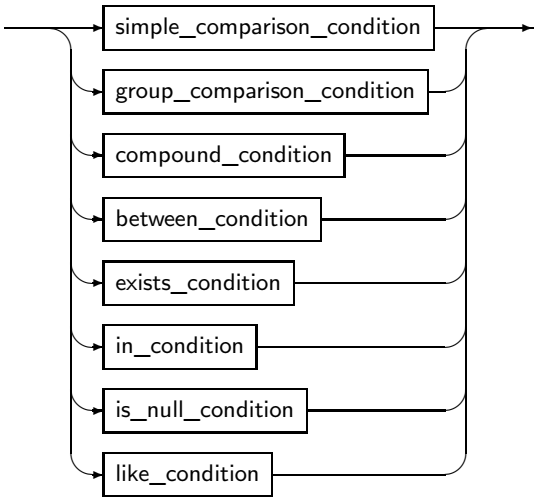
3.4. 조건식

조건식은 SELECT, UPDATE, DELETE 문장의 WHERE 절과 SELECT 문의 HAVING 절에 포함된다. Tibero에서 사용할 수 있는 조건식은 단순 조건식(Simple Condition), 그룹 조건식(Group Condition), 복합 조건식(Compound Condition), 비교 조건식(Comparison Condition) 등이 있다. 비교 조건식은 비교 연산자에 따라 BETWEEN, EXISTS, IN, IS NULL, LIKE 조건식 등이 있다.

조건식의 세부 내용은 다음과 같다.

- 문법

condition



- 구성요소

구성요소	설명
simple_comparison_condition	단순 조건식을 의미한다. 자세한 내용은 “3.4.1. 단순 조건식”을 참고한다.
group_comparison_condition	그룹 조건식을 의미한다. 자세한 내용은 “3.4.2. 그룹 조건식”을 참고한다.
compound_condition	복합 조건식을 의미한다. 자세한 내용은 “3.4.3. 복합 조건식”을 참고한다.
between_condition	BETWEEN 조건식을 의미한다. 자세한 내용은 “3.4.4. BETWEEN 조건식”을 참고한다.
exists_condition	EXISTS 조건식을 의미한다. 자세한 내용은 “3.4.5. EXISTS 조건식”을 참고한다.
in_condition	IN 조건식을 의미한다. 자세한 내용은 “3.4.6. IN 조건식”을 참고한다.
is_null_condition	IS NULL 조건식을 의미한다. 자세한 내용은 “3.4.7. IS NULL 조건식”을 참고한다.
like_condition	LIKE 조건식을 의미한다. 자세한 내용은 “3.4.8. LIKE 조건식”을 참고한다.
regexp_like_condition	REGEXP_LIKE 조건식을 의미한다. 자세한 내용은 “3.4.9. REGEXP_LIKE 조건식”을 참고한다.

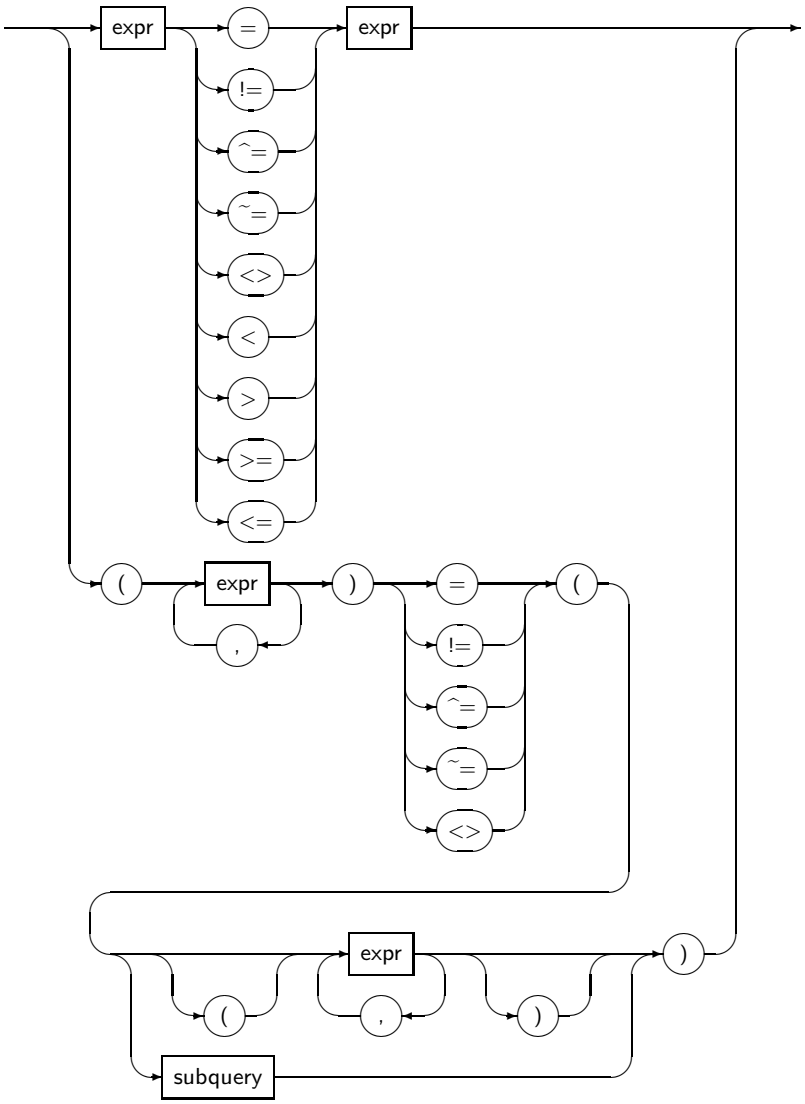
3.4.1. 단순 조건식

단순 조건식은 두 개의 데이터 값을 비교하는 조건식이다. 단순 조건식의 피연산자는 다음의 도식에서처럼 임의의 일반 연산식이 올 수 있다. 피연산자 중의 하나라도 NULL이 포함된 단순 조건식은 항상 UNKNOWN을 반환한다.

단순 조건식의 세부 내용은 다음과 같다.

- 문법

simple_comparison_condition



- 구성요소

구성요소	설명
expr	expr은 일반적인 연산식을 의미한다. 자세한 내용은 "3.3. 연산식"을 참고한다.
subquery	부질의를 의미한다. 자세한 내용은 "3.3.6. 부질의 연산식"을 참고한다.
=, !=, ^=, ~=, <>, <, >, >=, <=	비교연산자이다. 자세한 내용은 "3.2. 연산자"를 참고한다.

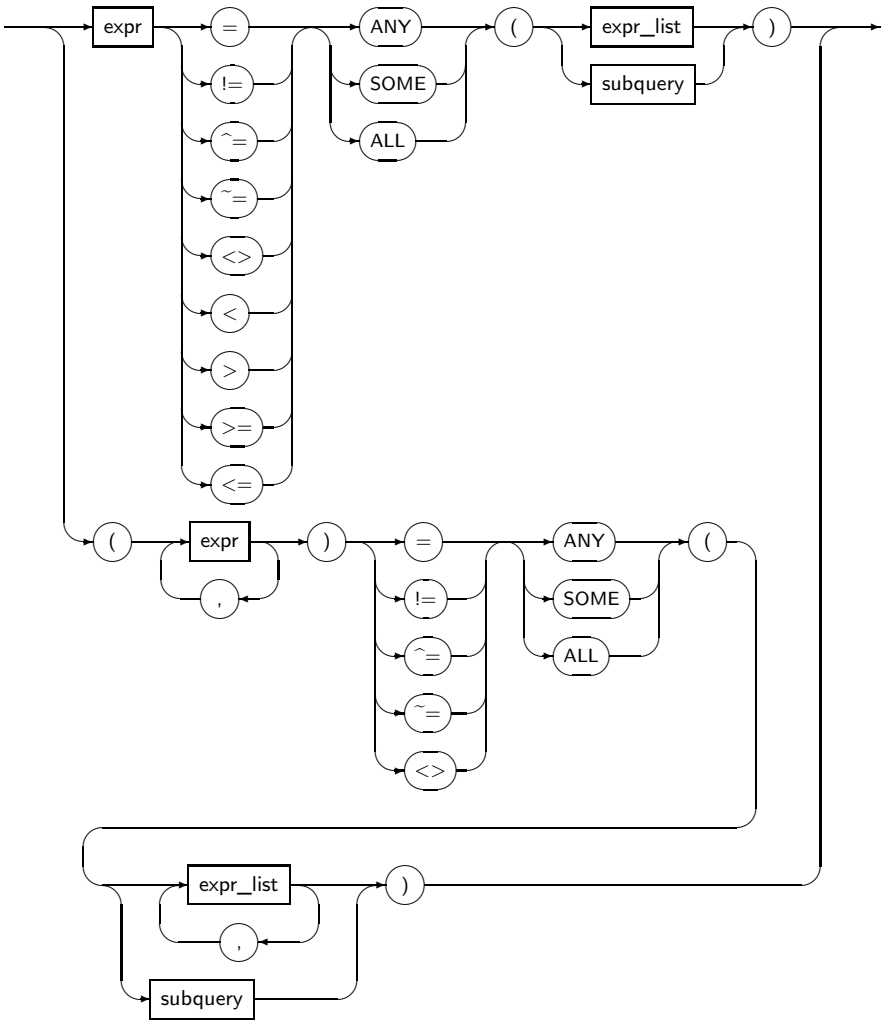
3.4.2. 그룹 조건식

그룹 조건식은 하나의 데이터 값과 리스트에 포함된 값을 비교하는 연산을 수행한다.

그룹 조건식의 세부 내용은 다음과 같다.

- 문법

group_comparison_condition



- 구성요소

구성요소	설명
expr	expr은 일반적인 연산식을 의미한다. 자세한 내용은 “3.3. 연산식”을 참고한다.
expr_list	expr_list는 리스트를 의미한다. 자세한 내용은 “3.3.8. 리스트”를 참고한다.
subquery	부질의를 의미한다. 자세한 내용은 “3.3.6. 부질의 연산식”을 참고한다.
ANY	ANY가 포함된 그룹 조건식의 경우 왼쪽의 데이터 값이 오른쪽의 리스트 내의 값 중 최소한 하나만 단순 비교 연산자를 만족하면 그룹 조건식은 TRUE를 반환한다.

구성요소	설명
SOME	SOME이 포함된 그룹 조건식의 경우 ANY와 마찬가지로 왼쪽의 데이터 값이 오른쪽의 리스트 내의 값 중 최소한 하나만 단순 비교 연산자를 만족하면 그룹 조건식은 TRUE를 반환한다.
ALL	ALL이 포함된 그룹 조건식의 경우 왼쪽의 데이터 값이 오른쪽의 리스트 내의 모든 값에 대해 단순 비교 연산자를 만족해야 그룹 조건식이 TRUE를 반환한다.

- 예제

  다음은 그룹 조건식의 예이다.

```
EMPNO = ANY (35, 54, 27, 69)
SALARY * 1.05 >= ALL (SELECT SALARY FROM EMP WHERE DEPTNO = 5)
```

  위의 예에서 첫 번째 줄은 컬럼 EMPNO 값이 리스트 내의 값 중 하나라도 동일하면 TRUE, 그렇지 않으면 FALSE를 반환한다. 두 번째 줄은 컬럼 SALARY 값에 1.05를 곱한 값이 오른쪽 부질의 결과로 반환되는 모든 값보다 크거나 같으면 TRUE, 그렇지 않으면 FALSE를 반환한다.

  오른쪽 리스트 내에 NULL이 포함된 경우 ANY 연산자가 포함된 그룹 조건식은 항상 TRUE 또는 UNKNOWN을 반환하고, ALL 연산자가 포함된 그룹 조건식은 항상 FALSE 또는 UNKNOWN을 반환한다.

  예를 들어 다음의 두 줄은 같은 결과를 반환한다.

```
DEPTNO != ALL (4, 5, NULL)
(DEPTNO != 4) AND (DEPTNO != 5) AND (DEPTNO != NULL)
```

  만약 컬럼 DEPTNO 값이 4 또는 5인 경우 두 번째 줄에서 NULL과 비교하는 마지막 단순 조건식의 결과가 UNKNOWN이 되고, 전체 조건식의 결과도 AND 연산자로 연결되어 있으므로 UNKNOWN이 된다.

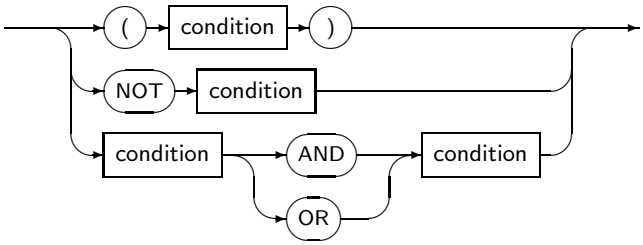
3.4.3. 복합 조건식

복합 조건식은 복합 연산식과 마찬가지로 하나 이상의 조건식을 결합한 복잡한 형태의 조건식이다. 복합 조건식은 괄호로 묶은 조건식, NOT 연산자를 붙인 연산식, 논리 연산자(AND, OR)로 결합한 조건식 등이 있다.

복합 조건식의 세부 내용은 다음과 같다.

- 문법

compound_condition



- 구성요소

구성요소	설명
condition	condition은 조건식을 의미하며, 자세한 내용은 "3.4. 조건식"의 도입부에 기술되어 있다.
NOT, AND, OR	논리 연산자 NOT, AND, OR이다. 자세한 내용은 "3.2.2. 조건식에 포함되는 연산자"를 참고한다.

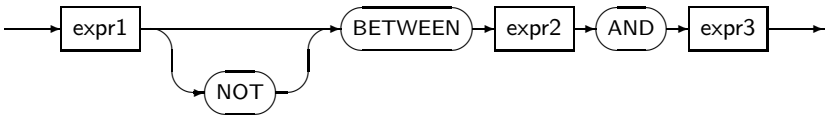
3.4.4. BETWEEN 조건식

BETWEEN 연산자는 왼쪽의 수치 값이 오른쪽의 두 수치 값 사이에 존재하는지 비교하는 연산자이다.

BETWEEN 조건식의 세부 내용은 다음과 같다.

- 문법

between_condition



- 구성요소

구성요소	설명
expr1, expr2, expr3	expr1, expr2, expr3은 모두 수치 값을 반환하는 임의의 연산식이다. expr1, expr2, exrp3 중에서 하나라도 NULL이 오면, BETWEEN 조건식의 결과는 항상 FALSE 또는 UNKNOWN을 반환한다.
BETWEEN ... AND	BETWEEN 연산자는 항상 AND를 동반하며, 왼쪽의 수치 값이 오른쪽의 두 수치 값 사이에 존재하는지 비교하는 연산자이다.
NOT	BETWEEN 예약어 앞에 NOT이 오면 BETWEEN 연산자 결과에 NOT 연산을 수행한 결과를 반환한다.

- 예제

  BETWEEN 조건식은 두 개의 산술 조건식을 AND 연산자로 연결한 것과 같은 결과를 반환한다.

  예를 들어 다음의 두 줄은 같은 결과를 반환한다.

```
SALARY BETWEEN 30000 AND 50000
(SALARY >= 30000) AND (SALARY <= 50000)
```

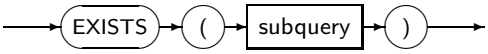
3.4.5. EXISTS 조건식

EXISTS 연산자는 오른쪽의 부질의를 실행한 결과가 하나 이상의 로우를 반환하면 TRUE, 그렇지 않으면 FALSE를 반환한다.

EXISTS 조건식의 세부 내용은 다음과 같다.

- 문법

exists_condition



- 구성요소

구성요소	설명
EXISTS	EXISTS 연산자의 오른쪽 subquery를 실행한 결과가 하나 이상의 로우를 반환하면 TRUE, 그렇지 않으면 FALSE를 반환한다.
subquery	subquery는 부질의를 의미한다. 자세한 내용은 “3.3.6. 부질의 연산식”을 참고한다.

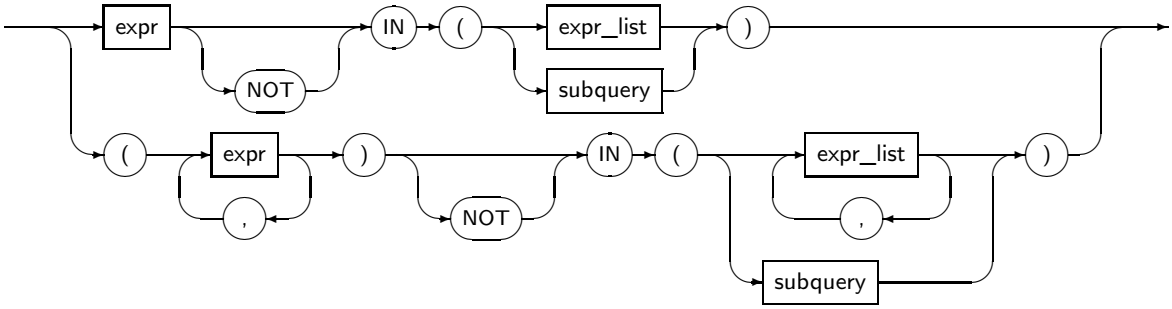
3.4.6. IN 조건식

IN 연산자는 왼쪽의 데이터 값이 오른쪽의 리스트 내에 포함되어 있는지 비교하는 연산자이다. IN 조건식의 문법은 다음의 도식과 같다. expr, expr_list, subquery는 “3.3. 연산식”에서 정의하였다.

IN 조건식의 세부 내용은 다음과 같다.

- 문법

in_condition



- 구성요소

구성요소	내용
expr	expr은 일반적인 연산식을 의미한다. 자세한 내용은 "3.3. 연산식"을 참고한다.
NOT	논리 연산자 NOT을 의미한다. 자세한 내용은 "3.2.2. 조건식에 포함되는 연산자"를 참고한다. IN 앞에 NOT이 오면 IN 연산 결과에 NOT 연산을 수행한 결과를 반환한다.
IN	IN 연산자는 왼쪽의 데이터 값이 오른쪽의 리스트 내에 포함되어 있는지 비교하는 연산자이다.
expr_list	expr 리스트를 의미한다. 자세한 내용은 "3.3.8. 리스트"를 참고한다. expr_list 자체는 괄호에 둘러싸여 있어서는 안 된다.
subquery	subquery는 부질의를 의미한다. 자세한 내용은 "3.3.6. 부질의 연산식"을 참고한다.

- 예제

IN 조건식은 ' = ANY' 또는 '= SOME' 연산자를 이용한 그룹 조건식으로 변환할 수 있다.

예를 들어 다음의 두 줄은 같은 결과를 반환한다.

```
EMPNO IN (35, 54, 27, 69)
EMPNO = ANY (35, 54, 27, 69)
```

IN 조건식이 = ANY 연산자가 포함된 그룹 조건식으로 변환될 수 있으므로, IN 연산자의 오른쪽 리스트 내에 NULL이 포함되어 있다면 IN 조건식은 항상 TRUE 또는 UNKNOWN을 반환한다. 반면에, NOT IN 조건식은 항상 FALSE 또는 UNKNOWN을 반환한다.

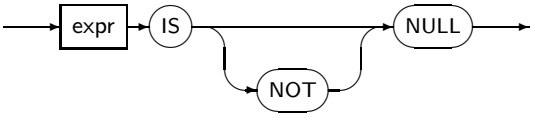
3.4.7. IS NULL 조건식

IS NULL 조건식은 왼쪽의 연산식의 결과가 NULL이면 TRUE, 그렇지 않으면 FALSE를 반환한다.

IS NULL 조건식의 세부 내용은 다음과 같다.

- 문법

is_null_condition



- 구성요소

구성요소	설명
expr	expr은 일반적인 연산식을 의미한다. 자세한 내용은 "3.3. 연산식"을 참고한다.
IS NULL	IS NULL 연산자는 왼쪽의 연산식의 결과가 NULL이면 TRUE, 그렇지 않으면 FALSE를 반환한다.
IS NOT NULL	IS NOT NULL 연산자는 IS NULL 연산자 결과에 NOT 연산을 수행한 결과를 반환한다.

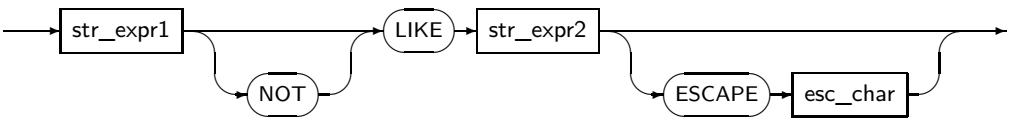
3.4.8. LIKE 조건식

LIKE 조건식은 문자열 데이터 간의 패턴을 비교한다. LIKE 연산자는 등호 연산자와 마찬가지로 영문 대문자와 소문자를 구분한다. 예를 들어 문자열 'ABCDE'와 'abcde'는 서로 다른 문자열이며, 문자열 패턴 'A%'에 대하여 문자열 'ABCDE'는 대응되지만, 'abcde'는 대응되지 않는다.

LIKE 조건식의 세부 내용은 다음과 같다.

- 문법

like_condition



- 구성요소

구성요소	설명
str_expr1, str_expr2	str_exp1, str_exp2는 문자열을 반환하는 임의의 연산식이며, 0 이상의 길이를 갖는다.
esc_char	esc_char는 이스케이프 문자를 반환하는 길이가 1인 임의의 문자이다.
LIKE	LIKE 연산자는 문자열 데이터 간의 패턴을 비교하는 연산자이다. LIKE 연산자는 왼쪽에 오는 문자열이 오른쪽에 오는 문자열 패턴에 일치하면 TRUE를 반환한다. 만약 str_expr1, str_expr2 중의 어떤 하나라도 NULL이면, LIKE 연산자는 UNKNOWN을 반환한다.
ESCAPE	와일드 카드(Wild Card)로 사용된 문자를 비교할 때는 이스케이프 문자를 사용한다.

두 문자열이 완전히 일치하는지 비교하는 데에는 등호(=) 연산자를 사용하는 반면, LIKE 연산자는 임의의 문자 또는 문자열에 대응되는 와일드 카드 문자열을 포함할 수 있다.

와일드 카드	설명
% (Percent)	길이가 0 이상인 임의의 문자열에 대응된다. 예를 들어 문자열 패턴 'A%'에 대응되는 문자열은 'A', 'Allen', 'Alice' 등이 있다.
_(Underscore)	'_'는 1개의 문자에 대응된다. 예를 들어 문자열 패턴 'A_'에 대응되는 문자열은 'AB', 'A1' 등이 있다. 문자열 'A', 'ABC', 'Allen' 등은 문자열 패턴 'A_'에 대응되지 않는다.

와일드 카드의 특징은 다음과 같다.

- 하나의 문자열 패턴에 하나 이상의 와일드 카드 문자를 포함할 수 있다.
- 와일드 카드로 사용된 문자를 비교할 때는 이스케이프 문자(escape character)를 사용한다.

  예를 들어 'A_'로 시작되는 모든 문자열을 찾을 때는 문자열 패턴으로 'A_%'를 사용한다. 이때, 백슬래시(\)가 이스케이프 문자로 사용된다. 이스케이프 문자로 사용된 백슬래시(\)를 비교하려면 이스케이프 문자를 두 번 반복해서 사용한다. 예를 들어 'C:\'로 시작되는 모든 문자열을 찾고자 한다면, 문자열 패턴으로 'C:\\%'를 사용한다.

  이스케이프 문자는 정해져 있지 않으며 LIKE 연산자를 사용할 때마다 이스케이프 문자를 다르게 지정할 수 있다. 이스케이프 문자는 항상 길이가 1이어야 한다.

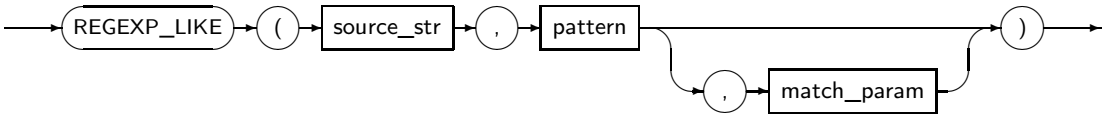
3.4.9. REGEXP_LIKE 조건식

REGEXP_LIKE 조건식은 정규표현식으로 된 문자열 패턴을 비교한다는 점을 제외하고는 LIKE와 동일하다. 이 기능은 ICU 정규 표현식 표준을 따른다.

REGEXP_LIKE 조건식의 세부 내용은 다음과 같다.

- 문법

regexp_like_condition



- 구성요소

구성요소	설명
source_str	문자열을 반환하는 임의의 연산식이다. CHAR, VARCHAR2, NCHAR, NVARCHAR2 타입을 사용할 수 있다.
pattern	정규표현식으로 작성된 문자열을 반환하는 임의의 연산식이다. CHAR, VARCHAR2, NCHAR, NVARCHAR2 타입을 사용할 수 있고, 만약 source_str과 타입이 다를 경우 source_str 타입으로 변환된다.
match_param	문자열을 반환하는 임의의 연산식으로 패턴을 검사할 방법을 설정한다. 다음과 같은 값을 사용할 수 있고, 여러 개를 동시에 지정할 수 있다. – 'i' : 대소문자를 구분하지 않는다. – 'c' : 대소문자를 구분한다. – 'n' : 점(.)이 줄바꿈 문자도 포함한다. – 'm' : 입력문자열이 한줄 이상이다. – 'x' : 공백문자를 무시한다. 예를 들어 'ic'와 같이 상호충돌하는 값을 지정하였을 경우엔 마지막값만 사용한다. 즉, 'ic'는 대소문자를 구분한다.

- 예제

다음은 REGEXP_LIKE 조건식을 사용하는 예이다.

```
SQL> SELECT 1 FROM DUAL WHERE REGEXP_LIKE('12345', '3.?4');

         1
----------
         1
```

```
1 row selected.
```

제4장 함수

본 장에서는 Tibero에서 제공하는 내장 함수에 대해 기술한다.

4.1. 개요

Tibero에서는 다양한 내장 함수를 제공하고 있다. 이러한 함수 중의 일부는 SQL 표준에 정의되어 있으며, 일부는 Tibero에서 추가적으로 제공하는 것이다. Tibero의 함수는 크게 단일 로우 함수와 집단 함수로 구분할 수 있다.

일부 함수는 파라미터가 없는 것도 있지만, 대부분의 함수는 하나 이상의 파라미터를 입력으로 받는다. 또한 모든 함수는 하나의 출력 값을 반환한다. 각 파라미터는 데이터 타입이 정해져 있다. 정해진 데이터 타입 이외의 다른 타입의 값이 입력된 경우에는 "3.3.1. 연산식의 변환"에서 설명한 대로 데이터 타입의 변환을 시도한다. 데이터 타입의 변환이 불가능한 경우에는 에러를 반환한다. 또한, 범위를 넘는 값을 컬럼에 저장할 때에도 에러를 반환한다.

대부분의 함수는 파라미터 값으로 NULL이 입력된 경우 NULL을 반환한다. NULL이 입력된 경우에도 NULL을 반환하지 않는 함수로는 CONCAT, NVL, REPLACE 등이 있다.

함수의 반환값을 컬럼에 저장할 때에는 반환값의 범위에 유의해야 한다.

- 함수의 반환값이 NUMBER 타입인 경우 컬럼의 정밀도와 스케일 범위 내의 값이어야 한다.
- 함수의 반환값이 CHAR 타입인 경우 컬럼의 최대 길이 범위 내의 값이어야 한다.
- 함수의 반환값이 VARCHAR 타입인 경우 컬럼의 최대 길이 범위 내의 값이어야 한다.

4.1.1. 단일 로우 함수

단일 로우 함수는 하나의 로우로부터 컬럼 값을 파라미터로 입력 받는 함수이다. 함수의 파라미터는 반드시 컬럼 값만 입력받는 것은 아니고 실제 데이터를 직접 입력으로 받을 수도 있다. 단일 로우 함수는 SQL 문장 내의 어떤 연산식에도 포함될 수 있다.

4.1.2. 집단 함수

집단 함수는 하나 이상의 로우로부터 컬럼 값을 파라미터로 입력받는 함수이다. 함수의 파라미터는 반드시 컬럼 값만 입력받는 것은 아니고 실제 데이터를 직접 입력으로 받을 수도 있다. 집단 함수는 SELECT 문의 SELECT 절, GROUP BY 절, HAVING 절에만 포함된다.

Tibero에서 제공하는 집단 함수에는 AVG, COUNT, MAX, MIN, SUM 등이 있다. 이러한 함수는 각각 파라미터로 주어진 컬럼에 대하여 평균, 개수, 최댓값, 최솟값, 합계 등을 구한다. 만약 파라미터로 실제 데이터 값이 주어지면 그 값을 그대로 반환한다.

SELECT 절

SELECT 문에서 SELECT 절의 집단 함수는 중첩될 수 있다. 단, 다른 위치의 집단 함수는 중첩되면 안 된다. 또한, SELECT 절의 집단 함수도 한 번의 중첩만을 허용한다.

따라서, 다음과 같은 집단 함수는 에러를 반환한다.

```
COUNT(SUM(AVG(SALARY)))
```

중첩된 집단 함수의 계산은 먼저 각 그룹에 대한 안쪽의 집단 함수를 계산하고, 여기에서 반환된 모든 값에 대하여 바깥쪽의 집단 함수를 계산한다. 예를 들어 SUM(AVG(SALARY)) 함수는 모든 그룹으로부터 SALARY 컬럼 값의 평균을 구하고, 그다음 모든 평균값의 합계를 구하여 반환한다.

집단 함수의 괄호 내에는 조건식이 아닌 임의의 연산식이 올 수 있다. SELECT 문에서 SELECT 절의 집단 함수는 다른 집단 함수를 포함하는 연산식이 올 수도 있다.

따라서, 다음과 같은 집단 함수도 유효하다.

```
SUM(AVG(SALARY) * COUNT(EMPNO) + 1.10)
```

COUNT 함수는 괄호 안에 애스터리스크(*)가 올 수도 있다. 이 경우 특정 컬럼이 아닌 전체 로우의 개수를 반환한다.

로우를 하나도 포함하지 않는 빈 테이블에 대해 집단 함수를 포함하는 SELECT 문을 실행하면, 결과 로우가 하나도 반환되지 않는다. 예외적으로, SELECT 절에 COUNT(*) 함수를 포함하면 0 값의 컬럼을 갖는 하나의 로우가 반환된다.

GROUP BY 절

집단 함수는 대개 SELECT 문 내에서 GROUP BY 절과 함께 사용한다. 집단 함수는 GROUP BY 절에 의하여 분리된 각 그룹에 포함된 모든 로우에 대하여 하나의 값을 반환한다. 만약 SELECT 문에서 GROUP BY 절을 포함하지 않으면, 전체 테이블을 하나의 그룹으로 인식한다.

다음의 SELECT 문은 GROUP BY 절을 포함한 예이다.

```
SELECT AVG(SALARY) FROM EMP
GROUP BY DEPTNO;
```

위의 문장은 테이블 EMP 내의 모든 로우 중에서 같은 DEPTNO 컬럼 값을 갖는 로우의 그룹으로 분리하고, 각 그룹에 포함된 모든 직원의 SALARY 컬럼 값의 평균을 계산한다.

HAVING 절

SELECT 문에서 HAVING 절은 그룹에 대한 조건식을 포함한다. HAVING 절은 SELECT 절이나 GROUP BY 절에 포함된 컬럼 또는 그 이외의 컬럼에 대한 집단 함수를 포함할 수 있다.

다음의 SELECT 문은 HAVING 절을 포함한 예이다. 본 예제에서는 3명 이상의 직원이 소속된 부서에 대해서만 SALARY 컬럼 값의 평균을 계산한다.

```
SELECT AVG(SALARY) FROM EMP
GROUP BY DEPTNO
HAVING COUNT(EMPNO) >= 3;
```

집단 함수의 파라미터 앞에는 DISTINCT 또는 ALL 예약어를 포함시킬 수 있다. 이러한 예약어는 중복되는 컬럼 값에 대한 처리를 정의하며, DISTINCT는 중복을 제거하고, ALL은 중복을 허용한다.

예를 들어 한 그룹 내의 로우가 갖고 있는 SALARY 컬럼 값이 20000, 20000, 20000, 40000이라면, AVG(DISTINCT SALARY) 함수의 결과는 30000이며, AVG(ALL SALARY) 함수의 결과는 25000이다. 만약 아무 것도 지정하지 않으면 디폴트는 ALL이다.

다음의 표는 Tibero에서 제공하는 집단 함수 목록이다.

집단 함수	설명
AVG	그룹 내의 모든 로우에 대한 expr 값의 평균을 구하는 함수이다.
CORR	파라미터로 주어진 expr1가 expr2의 상관계수를 계산하는 함수이다.
COUNT	쿼리가 반환하는 로우의 개수를 세는 함수이다.
COVAR_POP	expr1, expr2의 모공분산을 계산하는 함수이다.
COVAR_SAMP	expr1, expr2의 표본공분산을 계산하는 함수이다.
DENSE_RANK	각 그룹별로 로우를 정렬한 다음 그룹 내의 각 로우에 대한 순위를 반환하는 함수이다.
FIRST	정렬된 로우에서 처음에 해당하는 로우를 뽑아내어 명시된 집단함수를 적용한 결과를 반환한다.
LAST	정렬된 로우에서 마지막에 해당하는 로우를 뽑아내어 명시된 집단함수를 적용한 결과를 반환한 다.
MAX	그룹 내의 모든 로우에 대한 expr 값 중의 최댓값을 구하는 함수이다.
MIN	그룹 내의 모든 로우에 대한 expr 값 중의 최솟값을 구하는 함수이다.
PERCENT_RANK	파라미터로 주어진 값의 그룹 내의 위치를 나타내 주는 함수이다.
PERCENTILE_CONT	연속 분포 모델에서 파라미터로 주어진 백분위 값에 해당하는 값을 계산하는 역 분포 함수이다.
PERCENTILE_DISC	이산 분포를 가정한 역분산 함수로 분석 함수로도 사용할 수 있다.
RANK	그룹별로 로우를 정렬한 후 그룹 내의 각 로우의 순위를 반환하는 함수이다.

집단 함수	설명
REGR_SLOPE REGR_INTERCEPT REGR_COUNT REGR_R2 REGR_AVGX REGR_AVGY REGR_SXX REGR_SYY REGR_SXY	임의의 수치 데이터 쌍의 집합에 가장 맞는 선형 방정식을 구하기 위해 사용된다.
STATS_MODE	expr의 최빈값을 반환하는 함수이다.
STDDEV	expr의 표본 표준편차를 반환하는 함수이다.
STDDEV_POP	expr의 모표준편차를 반환하는 함수이다.
STDDEV_SAMP	expr의 누적 표본 표준편차를 반환하는 함수이다.
SUM	그룹 내의 모든 로우에 대한 expr 값의 합계를 구하는 함수이다.
VARIANCE	expr의 분산을 반환한다.
VAR_POP	expr의 모분산을 반환한다.
VAR_SAMP	expr의 표본분산을 반환하는 함수이다.
XMLAGG	XML 조각을 받고, 이를 한데 모아 XML 문서로 만들어 반환하는 함수이다.

4.1.3. 분석 함수

분석 함수는 집단 함수와 마찬가지로 특정 로우 그룹에 대한 집계 값을 구하는 데 사용된다.

집단 함수와 다른 점은 하나의 로우 그룹에 속한 모든 로우가 하나의 집계 값을 공유하지 않는다는 것이다. 각각의 로우에 대해 로우 그룹이 별개로 정의되며, 때문에 모든 로우가 별개로 각각 자신의 로우 그룹에 대한 집계 값을 갖게 된다. 이 로우 그룹을 분석 함수에서는 윈도우라고 부르며, analytic_clause 안에 정의된다. 윈도우 영역은 물리적인 로우의 개수로 정의될 수도 있고, 논리적인 어떤 계산 값을 통해서 정의될 수도 있다.

하나의 쿼리 블록 안에서 분석 함수는 ORDER BY 절을 제외하고 가장 마지막에 수행되는 연산이다. WHERE 절, GROUP BY 절, HAVING 절 모두 분석 함수가 수행되기 전에 먼저 적용된다. 그러므로 분석 함수는 SELECT 절 또는 ORDER BY 절에만 나올 수 있다.

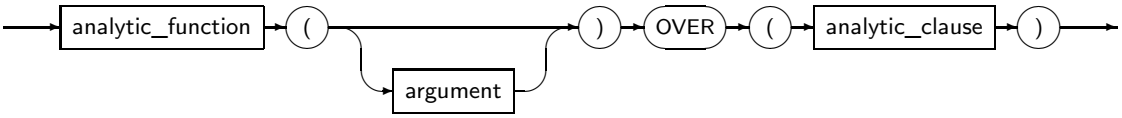
analytic_function

분석 함수는 크게 analytic_function, argument, analytic_clause로 구성된다.

analytic_function의 세부 내용은 다음과 같다.

- 문법

analytic_function



- 구성요소

구성요소	설명
analytic_function	분석 함수의 이름을 명시한다.
argument	분석 함수의 파라미터를 명시한다. 함수에 따라 파라미터의 타입이 결정된다.
OVER analytic_clause	OVER analytic_clause를 사용해 함수를 분석 함수로 수행할 수 있다.

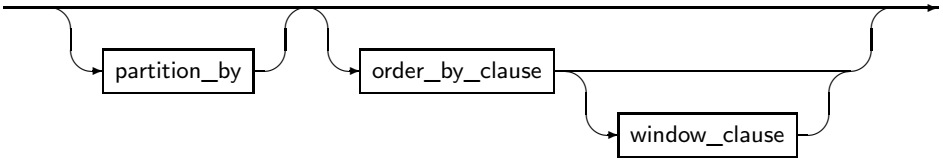
analytic_clause

OVER analytic_clause를 사용하여 함수를 분석 함수로 수행할 수 있다. 분석 함수는 ORDER BY 절을 제외한 다른 모든 절의 내용이 처리된 다음에 적용된다. 그러므로 분석 함수가 계산한 결과의 일부만 선택하고자 하면, 분석 함수를 수행한 쿼리를 뷰로 둘러싸고, 그 쿼리를 둘러싼 뷰에 WHERE 절을 적용하면 된다. analytic_clause 안에 분석 함수를 사용할 수는 없다. 그러나 부질의 내에 분석 함수를 사용하는 것은 가능하다.

analytic_clause의 세부 내용은 다음과 같다.

- 문법

analytic_clause



- 구성요소

구성요소	설명
partition_by	분석 함수를 계산하기 전에 현재 질의 블록의 결과 집합을 분할한다.
order_by_clause	partition_by에 의해 분할된 하나의 파티션 내에서 로우를 어떻게 정렬할지를 명시한다.
window_clause	window_clause를 가질 수 있는 경우가 있다. 분석 함수의 order_by_clause를 명시할 경우에만 이 window_clause를 명시할 수 있다.

partition_by

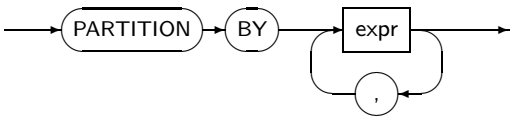
분석 함수를 계산하기 전에 현재 질의 블록의 결과 집합을 expr 또는 expr의 리스트를 기준으로 분할한다. 이 절을 명시하지 않으면, 분석 함수의 윈도우는 전체 로우 집합 내에서 움직이게 된다.

하나의 질의 블록의 SELECT 절 또는 ORDER BY 절에 여러 개의 분석 함수를 명시할 수 있으며, 각각이 서로 다른 PARTITION BY 키를 갖는 것도 가능하다.

partition_by의 세부 내용은 다음과 같다.

- 문법

partition_by



- 구성요소

구성요소	설명
expr	expr이 취할 수 있는 값은 상수, 컬럼, 분석 함수가 아닌 함수로 구성된 연산식이다.

order_by_clause

partition_by에 의해 분할된 하나의 파티션 내에서 로우를 어떻게 정렬할지를 명시한다. 정렬에 사용되는 키 값은 여러 개를 명시할 수 있다.

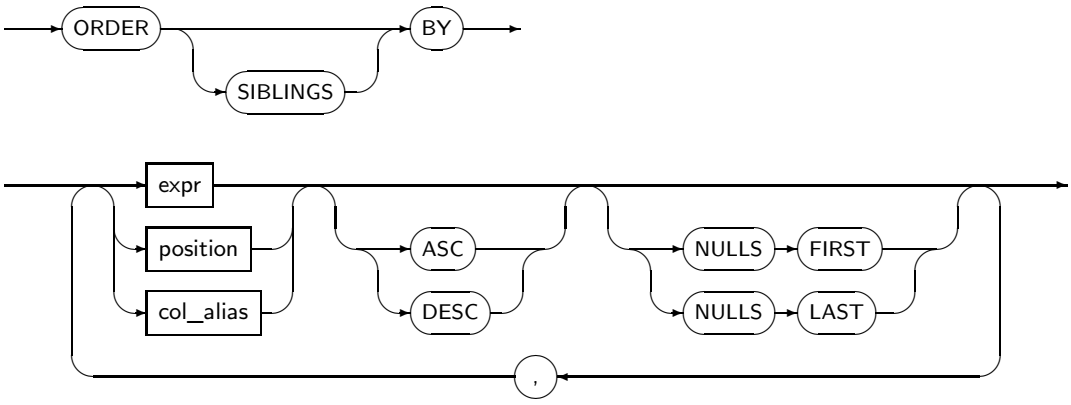
분석 함수에서 사용되는 order_by_clause 내에서는 위치 상수(ORDER BY 1과 같은)를 사용할 수 없다. SIBLINGS 역시 사용할 수 없다. SELECT 리스트 컬럼의 별칭도 사용할 수 없다. 그 이외에는 보통의 ORDER BY 절과 사용 방식이 같다.

분석 함수에 사용된 order_by_clause는 파티션 내의 로우의 순서를 결정할 뿐이지 분석 함수를 적용하고 난 쿼리 블록의 최종 결과 집합의 로우의 순서를 결정해 주는 것은 아니다. 이를 위해서는 쿼리 블록을 위한 별도의 ORDER BY 절을 추가로 명시해야만 한다.

order_by_clause의 세부 내용은 다음과 같다.

- 문법

order_by_clause



- 구성요소

구성요소	설명
SIBLINGS	order_by_clause는 계층 질의의 형제 노드 내에서 정렬 순서를 정의한다. 분석 함수에서는 사용할 수 없다.
expr	정렬의 키로 사용되는 연산식이다.
position	select_list에 명시된 expr의 위치를 지정한다. 분석 함수에서는 사용할 수 없다.
ASC	디폴트로, 정렬 순서를 명시한다. ASC는 오름차순으로 정렬한다.
DESC	정렬 순서를 명시한다. DESC는 내림차순으로 정렬한다.
NULLS FIRST	NULL 값의 정렬 순서를 명시한다. NULLS FIRST는 오름차순 정렬의 디폴트로 사용된다.
NULLS LAST	NULL 값의 정렬 순서를 명시한다. NULLS LAST는 내림차순 정렬의 디폴트로 사용된다.

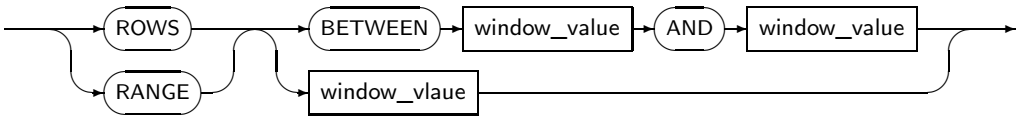
window_clause

분석 함수에 따라 window_clause를 가질 수 있는 경우가 있다. 분석 함수의 order_by_clause를 명시할 경우에만 window_clause를 명시할 수 있다. window_clause를 명시하지 않았을 때는 필요한 경우에 디폴트 윈도우로 RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW가 지정된다.

window_clause의 세부 내용은 다음과 같다.

- 문법

window_clause



- 구성요소

구성요소	설명
ROWS	윈도우 타입을 ROW로 지정한다. 분석 함수는 현재 로우가 정의하는 윈도우 내의 로우에 대해서 계산이 된다. ROWS는 윈도우를 물리적인 로우 단위로 정의한다. RANGE와는 달리 ROW로 지정된 윈도우의 로우는 order_by_clause를 통해 동률이 나왔을 경우 상이한 로우가 결과로 반환된다.
RANGE	윈도우 타입을 RANGE로 지정한다. RANGE는 현재 로우를 기준으로 논리적인 오프셋을 명시하여 윈도우를 정의한다. ROW와는 달리 RANGE로 지정된 윈도우의 로우는 언제나 항상 똑같은 로우가 반환된다. RANGE를 명시할 경우는 order_by_clause에 하나의 키만 명시할 수 있다. RANGE로 정의된 윈도우의 경우 두 개의 로우가 order_by_clause로 인해 정렬한 결과가 동률일 때는 분석 함수의 결과 값은 항상 동일하다. “4.2.157. SUM”의 분석 함수 예제에서 이를 확인해 볼 수 있다.
BETWEEN ... AND	윈도우의 시작점과 끝점을 명시한다. AND 이전에 오는 것이 시작점, AND 이후에 오는 것이 끝점이다. BETWEEN ... AND를 명시하지 않고 하나의 시점만 명시할 경우에는, 그 시점이 시작점이 되며 끝점은 현재 로우가 된다.

다음의 표는 Tibero에서 제공하는 분석 함수와 함수별 window_clause의 명시 가능 여부를 나타낸다.

분석 함수	window_clause 명시 가능 여부
AVG	예
CORR	예
COUNT	예
COVAR_POP	예
COVAR_SAMP	예
DENSE_RANK	아니오
FIRST	아니오

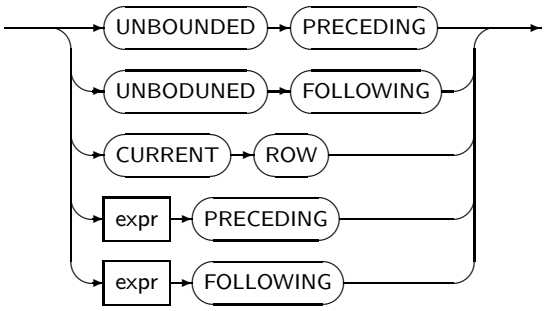
분석 함수	window_clause 명시 가능 여부
FIRST_VALUE	예
LAG	아니오
LAST	아니오
LAST_VALUE	예
LEAD	아니오
MAX	예
MIN	예
NTILE	아니오
PERCENT_RANK	아니오
PERCENTILE_CONT	아니오
PERCENTILE_DISC	아니오
RANK	아니오
RATIO_TO_REPORT	아니오
REGR_SLOPE REGR_INTERCEPT REGR_COUNT REGR_R2 REGR_AVGX REGR_AVGY REGR_SXX REGR_SYY REGR_SXY	예
ROW_NUMBER	아니오
STDDEV	예
STDDEV_POP	예
STDDEV_SAMP	예
SUM	예
VARIANCE	예
VAR_POP	예
VAR_SAMP	예

window_value

window_value의 세부 내용은 다음과 같다.

- 문법

window_value



- 구성요소

구성요소	설명
UNBOUNDED PRECEDING	시작점을 명시할 때 사용하며, 파티션의 첫 번째 로우를 지정한다. 끝점으로 사용할 수는 없다.
UNBOUNDED FOLLOWING	끝점을 명시할 때 사용하며, 파티션의 맨 마지막 로우를 지정한다. 시작점으로 사용할 수는 없다.
CURRENT ROW	시작점 또는 끝점으로 사용할 수 있다. ROW 또는 RANGE를 명시했을 경우 현재 로우 또는 현재 로우에서 계산된 값을 의미한다. 시작점으로 명시했을 때는 끝점으로 expr PRECEDING을 사용할 수 없으며, 끝점으로 명시했을 경우는 시작점으로 expr FOLLOWING을 사용할 수 없다.
expr PRECEDING	expr PRECEDING을 끝점에 명시했으면, 시작점은 항상 expr PRECEDING이 되어야 한다.
expr FOLLOWING	expr FOLLOWING을 시작점에 명시했으면, 끝점은 항상 expr FOLLOWING이 되어야 한다.

윈도우 타입별로 expr은 다음과 같이 달라진다.

– 윈도우 타입이 ROW일 경우에는 다음과 같다.

  • expr은 물리적인 오프셋을 지정한다. 이것은 양수이거나 양수로 계산되는 연산식이다.

  • 시작점의 로우는 끝점의 로우보다 먼저 나와야 한다.

– 윈도우 타입이 RANGE일 경우에는 다음과 같다.

- expr은 논리적인 오프셋을 지정한다. 이것은 0 또는 양수로 계산되는 연산식 또는 간격 리터럴이다. 간격 리터럴에 대해서는 “2.3. 리터럴”을 참고한다.
- expr에 수치 값을 사용할 수 있는 경우는 order_by_clause의 expr의 데이터 타입이 NUMBER 또는 DATE 타입일 때이다.
- expr에 간격 값을 사용할 수 있는 경우는 order_by_clause의 expr의 데이터 타입이 DATE 타입일 때이다.

4.2. 함수 목록

본 절에서는 Tibero에서 제공하는 내장 함수에 대해 설명한다.

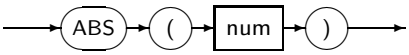
4.2.1. ABS

ABS는 주어진 파라미터 값의 절댓값(Absolute Value)을 구하는 함수이다.

ABS의 세부 내용은 다음과 같다.

- 문법

abs



- 구성요소

구성요소	설명
num	임의의 수치값을 반환하는 연산식이다. num은 숫자형 타입이거나 숫자형 타입으로 변환될 수 있는 타입이어야 한다.

- 예제

다음은 ABS 함수를 사용하는 예이다.

```
SQL> SELECT ABS(15.5), ABS(-25.5) FROM DUAL;

 ABS(15.5) ABS(-25.5)
---------- ----------
      15.5       25.5

1 row selected.
```

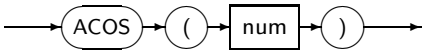
4.2.2. ACOS

ACOS는 num의 아크 코사인 값을 구하는 함수이다.

ACOS의 세부 내용은 다음과 같다.

- 문법

acos



- 구성요소

구성요소	설명
num	-1 ~ 1 사이의 값을 가져야 한다. 반환값은 0 ~ pi 사이의 라디안(Radian)값이다. num은 숫자형 타입이거나 숫자형 타입으로 변환될 수 있는 타입이어야 한다. 만약 num의 타입이 BINARY_FLOAT일 경우엔 BINARY_DOUBLE 타입을 반환하고, 그렇지 않은 경우엔 num과 동일한 숫자형 타입을 반환한다.

- 예제

다음은 ACOS 함수를 사용하는 예이다.

```
SQL> SELECT ACOS(.4) FROM DUAL;

  ACOS(.4)
----------
1.15927948

1 row selected.
```

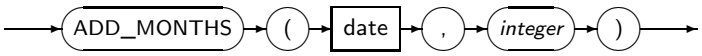
4.2.3. ADD_MONTHS

ADD_MONTHS는 date에 integer만큼의 달을 더한 결과를 구하는 함수이다.

ADD_MONTHS의 세부 내용은 다음과 같다.

- 문법

add_months



- 구성요소

구성요소	설명
date	date는 DATE 값을 반환하는 임의의 연산식이다.
integer	integer는 정수 값을 저장하는 데이터 타입이다.

- 예제

다음은 ADD_MONTHS 함수를 사용하는 예이다.

```
SQL> SELECT ADD_MONTHS (DATE'2006-01-01', 1) FROM DUAL;

ADD_MONTHS(DATE'2006-01-01',1)
------------------------------
2006/02/01

1 row selected.
```

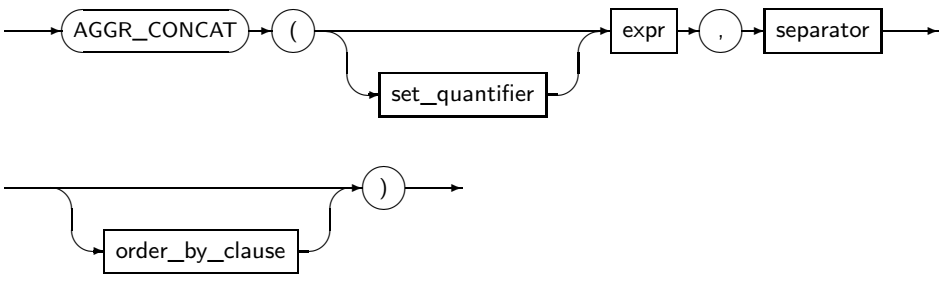
4.2.4. AGGR_CONCAT

AGGR_CONCAT는 그룹 내의 모든 로우에 대해 문자열과 구분자를 접합하여 하나의 문자열로 만들어 반환하는 함수이다. NULL 값을 반환하는 파라미터는 결과로부터 제외된다.

AGGR_CONCAT의 세부 내용은 다음과 같다.

- 문법

aggr_concat



- 구성요소

구성요소	설명
set_quantifier	질의 결과에 중복된 로우의 허용, 비허용 여부를 지정한다.

구성요소	설명
	DISTINCT, UNIQUE, ALL을 지정할 수 있다. – DISTINCT, UNIQUE : 중복된 로우를 제거한다. – ALL : 모든 로우를 선택한다. (기본값)
expr	문자열이나 문자열로 변환될 수 있는 임의의 연산식이다.
separator	expr과 접합될 구분자를 나타내는 문자 리터럴이다.
order_by_clause	접합할 문자열을 어떻게 정렬할지를 명시한다. 자세한 내용은 “4.1.3. 분석 함수”의 order_by_clause를 참고한다.

- 예제

  다음은 AGGR_CONCAT 함수를 사용하는 예이다.

```
SQL> SELECT AGGR_CONCAT(NAME, ',') AS "EMPLOYEE" FROM EMP
     GROUP BY DEPT_ID;

EMPLOYEE
--------------------------------------------------------------------
Johnny Depp,Brad Pitt,Bruce Willis
Will Smith,Nicolas Cage
Jason Statham
Angelina Jolie

4 rows selected.
```

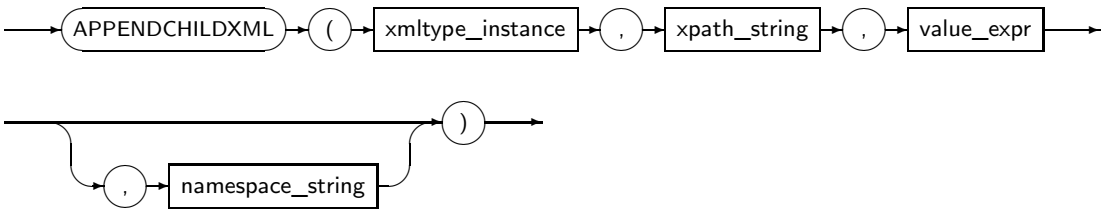
4.2.5. APPENDCHILDXML

APPENDCHILDXML은 XPath expression으로 지정된 노드에 사용자가 입력한 XML 값을 이어붙이는 함수이다. 이미 존재하고 있는 노드들의 뒤부터 삽입된다.

APPENDCHILDXML의 세부 내용은 다음과 같다.

- 문법

appendchildxml



- 구성요소

구성요소	설명
XMLType_instance	XML 타입 객체를 반환하는 임의의 연산식이다.
XPath_string	하나 이상의 자식 노드가 삽입될 위치를 나타내는 XPath 연산식이다.
value_expr	삽입될 하나 이상의 자식 노드를 나타내는 임의의 연산식이다. 이 연산식은 반드시 문자열로 변환 가능해야 한다.
namespace_string	XPath의 네임스페이스 정보를 나타낸다. 반드시 VARCHAR 타입이어야 한다.

- 예제

다음은 APPENDCHILDXML 함수를 사용하는 예이다.

```
... INFO 컬럼 '<dept><id>1</id><id>2</id></dept>' ...
SQL> UPDATE EMP SET INFO = APPENDCHILDXML(INFO, '/dept', XMLTYPE('<id>3</id>'));

SQL> SELECT INFO FROM EMP;

EMP
-----------------------------------------
<dept><id>1</id><id>2</id><id>3</id></dept>
```

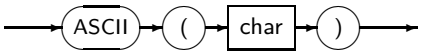
4.2.6. ASCII

ASCII는 char의 첫 번째 문자에 대해서 데이터베이스 문자 집합에서의 십진수 표시 값을 반환하는 함수이다. 만일 현재 데이터베이스의 문자 집합이 7bits ASCII라고 한다면, 이 함수는 ASCII 값을 반환한다. 만일 문자 집합이 EBCDIC이었다면, EBCDIC 값을 반환한다.

ASCII의 세부 내용은 다음과 같다.

- 문법

ascii



- 구성요소

구성요소	설명
char	char는 다음 타입 중 하나이다. – CHAR

구성요소	설명
	– VARCAHR – NCHAR – NVARCHAR char는 CLOB 타입일 수는 없다. 하지만, 타입의 변환 함수를 통해 ASCII 함수를 호출할 수 있다.

- 예제

  다음은 ASCII 함수를 사용하는 예이다.

```
SQL> SELECT ASCII('ABC') CODE FROM DUAL;

      CODE
----------
        65

1 row selected.
```

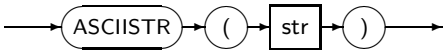
4.2.7. ASCIISTR

ASCIISTR은 주어진 문자열을 데이터베이스 문자 집합의 아스키 문자열로 반환한다. ASCII 이외의 문자는 \xxxx 형태로 변경된다. xxxx는 UTF-16 코드이다.

ASCIISTR의 세부 내용은 다음과 같다.

- 문법

  asciistr



- 구성요소

구성요소	설명
str	문자열을 반환하는 임의의 연산식이다.

- 예제

  다음은 ASCII 함수를 사용하는 예이다. 데이터베이스 문자 집합은 'MSWIN949' 이다.

```
SQL> SELECT ASCIISTR('A한글B') FROM DUAL;

ASCIISTR('A한글B')
------------------
A\D55C\AE00B

1 row selected.
```

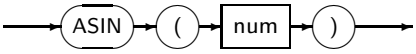
4.2.8. ASIN

ASIN는 num의 아크 사인 값을 구하는 함수이다.

ASIN의 세부 내용은 다음과 같다.

- 문법

asin



- 구성요소

구성요소	설명
num	-1 ~ 1 사이의 값을 가져야 한다. 반환값은 -pi/2 ~ pi/2 사이의 라디안 값이다. num은 숫자형 타입이거나 숫자형 타입으로 변환될 수 있는 타입이어야 한다. 만약 num의 타입이 BINARY_FLOAT일 경우엔 BINARY_DOUBLE 타입을 반환하고, 그렇지 않은 경우엔 num과 동일한 숫자형 타입을 반환한다.

- 예제

다음은 ASIN 함수를 사용하는 예이다.

```
SQL> SELECT ASIN(.4) FROM DUAL;

  ASIN(.4)
----------
.411516846

1 row selected.
```

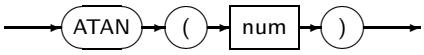
4.2.9. ATAN

ATAN은 num의 아크 탄젠트 값을 구하는 함수이다.

ATAN의 세부 내용은 다음과 같다.

- 문법

atan



- 구성요소

구성요소	설명
num	num의 값은 제한이 없으며 반환값은 -pi/2 ~ pi/2 사이의 라디안 값이다. num은 숫자형 타입이거나 숫자형 타입으로 변환될 수 있는 타입이어야 한다. 만약 num의 타입이 BINARY_FLOAT일 경우엔 BINARY_DOUBLE 타입을 반환하고, 그렇지 않은 경우엔 num과 동일한 숫자형 타입을 반환한다.

- 예제

다음은 ATAN 함수를 사용하는 예이다.

```
SQL> SELECT ATAN(.4) FROM DUAL;

  ATAN(.4)
----------
.380506377

1 row selected.
```

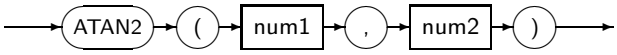
4.2.10. ATAN2

ATAN2는 아크 탄젠트 값을 구하는 함수이다. n과 m의 아크 탄젠트 값을 구한다. ATAN2(num1, num2)은 ATAN(num1/num2)과 같다.

ATAN2의 세부 내용은 다음과 같다.

- 문법

atan2



- 구성요소

구성요소	설명
num1, num2	num1, num2의 값은 제한이 없으며 반환값은 -pi/2 ~ pi/2 사이의 라디안 값이다. num1, num2는 숫자형 타입이거나 숫자형 타입으로 변환될 수 있는 타입이어야 한다. num1이나 num2의 타입이 BINARY_FLOAT이나 BINARY_DOUBLE일 경우엔 BINARY_DOUBLE 타입을 반환하고, 그렇지 않은 경우엔 NUMBER 타입을 반환한다.

- 예제

다음은 ATAN2 함수를 사용하는 예이다.

```
SQL> SELECT ATAN2(.3, .4) ATAN2 FROM DUAL;

     ATAN2
----------
.643501109

1 row selected.
```

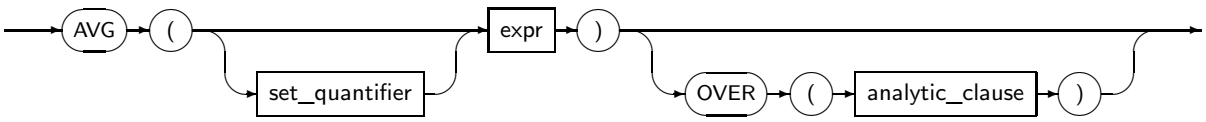
4.2.11. AVG

AVG는 그룹 내의 모든 로우에 대한 expr 값의 평균을 구하는 함수이다.

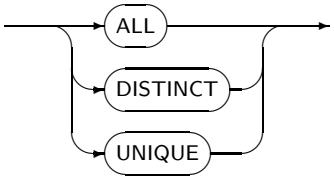
AVG의 세부 내용은 다음과 같다.

- 문법

avg



set_quantifier



- 구성요소

구성요소	설명
analytic_clause	OVER analytic_clause를 사용해 함수를 분석 함수로 수행할 수 있다. 자세한 내용은 "4.1.3. 분석 함수"의 analytic_clause를 참고한다.
expr	expr은 임의의 연산식이며, "3.3. 연산식"에서 이미 정의하였다.
ALL	ALL은 기본값이다. expr 값 중에서 중복되는 값을 제거하지 않고, 모든 값의 평균을 구한다.
DISTINCT	expr 앞에 DISTINCT 예약어를 포함시키면, 평균을 구하기 전에 expr 값 중에서 중복되는 값을 제거한다. DISTINCT를 명시할 경우는 analytic_clause에서 query_partion_clause만 명시할 수 있다. order_by_clause는 명시할 수 없다. order_by_clause를 명시할 수 없으므로 window_clause 또한 명시할 수 없다.
UNIQUE	UNIQUE는 DISTINCT와 동일하다.

- 예제

  다음은 AVG 함수를 사용하는 예이다.

```
SQL> SELECT AVG(SALARY) AVG FROM EMP GROUP BY DEPTNO;

       AVG
----------
      3255

1 row selected.
```

- 분석 함수 예제

  다음은 AVG 함수를 분석 함수로 사용하는 예이다.

```
SQL> SELECT ID, HIREDATE, SALARY, AVG(SALARY) OVER
       (ORDER BY HIREDATE ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING)
       AS AAVG
     FROM EMP;

        ID HIREDATE       SALARY       AAVG
---------- ---------- ---------- ----------
         1 1987/01/06      20000      17500
         5 1991/05/16      15000 14333.3333
         4 1999/11/25       8000 9333.33333
         2 2001/06/07       5000 6333.33333
         8 2003/03/26       6000 6666.66667
         6 2003/08/15       9000 6333.33333
```

```
         7 2004/02/08       4000 6666.66667
         3 2005/09/23       7000       5500

8 rows selected.
```

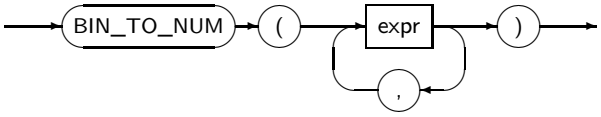
4.2.12. BIN_TO_NUM

BIN_TO_NUM는 bit vector를 해당 정수 값으로 변환해주는 함수이다.

BIN_TO_NUM의 세부 내용은 다음과 같다.

- 문법

bin_to_num



- 구성요소

구성요소	설명
expr	expr는 숫자형 타입이나 숫자형 타입으로 변환될 수 있는 값을 반환하는 임의의 연산식이다. expr의 반환 값이 숫자형 타입이 아니면 숫자형 타입으로 변환 후 정수 값으로 변환해서 사용한다. (trunc() 결과와 같음) 변환 후의 정수 값은 0 또는 1 이어야 한다.

- 예제

다음은 BIN_TO_NUM 함수를 사용하는 예이다.

```
SQL> SELECT BIN_TO_NUM(1,0,1) FROM DUAL;
BIN_TO_NUM(1,0,1)
-----------------
                5

1 row selected.
```

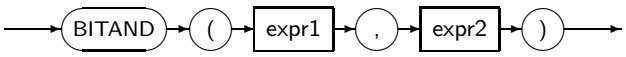
4.2.13. BITAND

BITAND는 expr1과 expr2의 각 비트를 AND 연산한 결과를 구하는 함수이다.

BITAND의 세부 내용은 다음과 같다.

- 문법

bitand



- 구성요소

구성요소	설명
expr1, expr2	expr1과 expr2는 정수 값을 반환하는 임의의 연산식이다.

- 예제

다음은 BITAND 함수를 사용하는 예이다.

```
SQL> SELECT BITAND(3, 1), BITAND(4, 1) FROM DUAL;

BITAND(3,1) BITAND(4,1)
----------- -----------
          1           0

1 row selected.
```

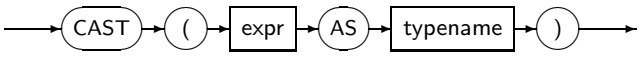
4.2.14. CAST

CAST는 하나의 데이터 타입에서 다른 데이터 타입으로 변경하는 함수이다.

CAST의 세부 내용은 다음과 같다.

- 문법

cast



- 구성요소

구성요소	문법
expr	expr은 임의의 데이터 타입을 반환하는 임의의 연산식이다.
typename	typename은 변환할 데이터 타입의 이름을 명시한다.

- 예제

다음은 CAST 함수를 사용하는 예이다.

```
SQL> SELECT CAST('1974-06-23' AS TIMESTAMP) TS FROM DUAL;

TS
--------------------------
1974/06/23 00:00:00.000000

1 row selected.
```

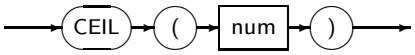
4.2.15. CEIL

CEIL은 주어진 파라미터의 값보다 크거나 같은 가장 작은 정수를 구하는 함수이다.

CEIL의 세부 내용은 다음과 같다.

- 문법

ceil



- 구성요소

구성요소	설명
num	수치 값을 반환하는 임의의 연산식이다. num은 숫자형 타입이거나 숫자형 타입으로 변환될 수 있는 타입이어야 하고, num과 동일한 숫자형 타입을 반환한다.

- 예제

다음은 CEIL 함수를 사용하는 예이다.

```
SQL> SELECT CEIL(15.5), CEIL(-15.5), CEIL(25.0) FROM DUAL;

CEIL(15.5) CEIL(-15.5) CEIL(25.0)
---------- ----------- ----------
        16         -15         25

1 row selected.
```

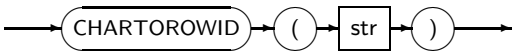
4.2.16. CHARTOROWID

CHARTOROWID는 CHAR, VARCHAR, NCHAR, NVARCHAR 형식의 값을 ROWID 형식으로 변환하는 함수이다.

CHARTOROWID의 세부 내용은 다음과 같다.

- 문법

  chartorowid

- 구성요소

구성요소	설명
str	ROWID 형식으로 변환할 임의의 값이다.

- 예제

  다음은 CHARTOROWID 함수를 사용하는 예이다.

```
SQL> SELECT DEPT_ID FROM EMP WHERE ROWID = CHARTOROWID('AAAAUcAAAAAAAxPAAA');

DEPT_ID
-------
      5

1 row selected.
```

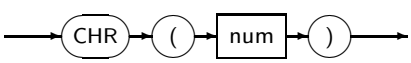
4.2.17. CHR

CHR은 num에 대응하는 문자를 구하는 함수이다.

CHR의 세부 내용은 다음과 같다.

- 문법

  chr

- 구성요소

구성요소	설명
num	num은 수치 값을 반환하는 임의의 연산식이다.

- 예제

다음은 CHR 함수를 사용하는 예이다.

```
SQL> SELECT CHR(68) || CHR(66) RSLT FROM DUAL;

RSLT
----
DB

1 row selected.
```

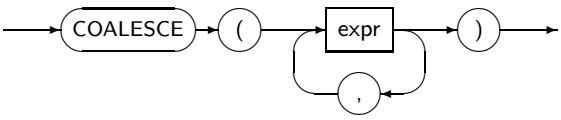
4.2.18. COALESCE

COALESCE는 NULL이 아닌 첫 파라미터를 반환하는 함수이다. 모든 파라미터가 NULL 값을 갖는다면 NULL을 반환한다.

COALESCE의 세부 내용은 다음과 같다.

- 문법

coalesce



- 구성요소

구성요소	설명
expr	expr은 다른 expr과 서로 동일하거나 호환할 수 있는 타입의 임의의 연산식이다.

- 예제

다음은 COALESCE 함수를 사용하는 예이다.

```
SQL> SELECT COALESCE(NULL, 'A', 'B') FROM DUAL;

COALESCE(NULL, 'A', 'B')
------------------------
A

1 row selected.
```

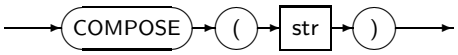
4.2.19. COMPOSE

COMPOSE는 입력받은 문자열을 NFC 정규 형태의 유니코드 문자열로 변환하는 함수이다.

COMPOSE의 세부 내용은 다음과 같다.

- 문법

  compose

- 구성요소

구성요소	설명
str	문자열을 반환하는 임의의 연산식이다.

- 예제

  다음은 COMPOSE 함수를 사용하는 예이다.

```
SQL> SELECT COMPOSE('o' || UNISTR('\0308')) FROM DUAL;

COMPOSE('O'||UNISTR('\0308'))
--------------------------------------------------------------------------------
ö

1 row selected.
```

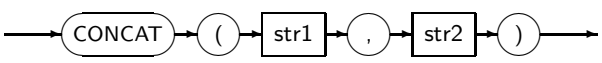
4.2.20. CONCAT

CONCAT은 두 개의 문자열을 접합하여 하나의 문자열을 반환하는 함수이다. 접합 연산자(||)를 사용했을 때와 같은 결과를 얻는다. 두 파라미터 중에서 하나가 NULL이더라도 NULL을 반환하지는 않는다.

CONCAT의 세부 내용은 다음과 같다.

- 문법

  concat

- 구성요소

구성요소	설명
str1, str2	str1과 str2는 문자열을 반환하는 임의의 연산식이다.

- 예제

  다음은 CONCAT 함수를 사용하는 예이다.

```
SQL> SELECT CONCAT('ABC', 'DEF') FROM DUAL;

CONCAT('ABC', 'DEF')
--------------------
ABCDEF

1 row selected.
```

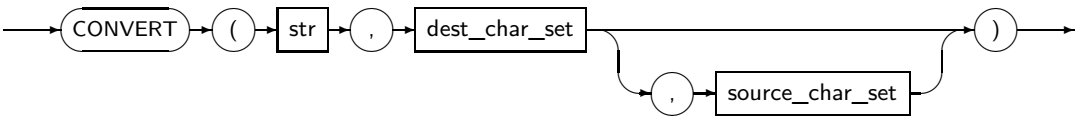
4.2.21. CONVERT

CONVERT은 주어진 문자열을 다른 문자집합의 문자열로 변환한다. 대응하는 문자가 없다면 '?'로 대체된다.

CONVERT의 세부 내용은 다음과 같다.

- 문법

convert



- 구성요소

구성요소	설명
str	문자열을 반환하는 임의의 연산식이다.
dest_char_set	변환하려는 문자 집합명을 명시한다. – 데이터베이스 문자 집합명을 사용할 수 있다. – 그 외에 ISO2022-KR, US8ICL 문자집합을 사용할 수 있다.
source_char_set	str의 문자 집합명을 명시한다. 명시하지 않으면 데이터베이스 문자 집합명이 기본값으로 사용된다.

- 예제

  다음은 CONVERT 함수를 사용하는 예이다.

```
SQL> SELECT CONVERT('A한글B', 'US7ASCII', 'MSWIN949') FROM DUAL;

CONVERT('A한글B','US7ASCII','MSWIN949')
---------------------------------------
A??B

1 row selected.
```

4.2.22. CORR

CORR은 파라미터로 주어진 expr1가 expr2의 상관계수를 계산하는 함수이다. 분석 함수로도 사용할 수 있다. 이 함수는 모든 수치 데이터 타입과 수치 데이터 타입은 아니지만 묵시적으로 수치 데이터 타입으로 변환할 수 있는 데이터 타입을 파라미터로 받아 들인다. 입력된 파라미터의 수치 데이터 타입 또는 변환된 수치 데이터 타입을 반환한다.

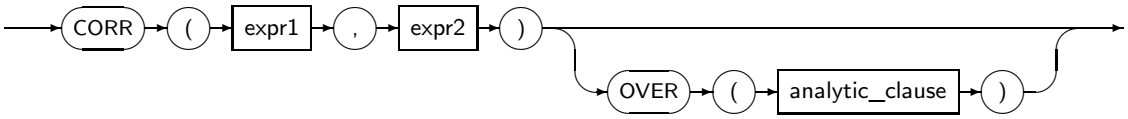
Tibero에서는 다음의 공식을 사용해 상관계수를 계산한다.

```
COVAR_POP(expr1, expr2) / (STDDEV_POP(expr1) * STDDEV_POP(expr2))
```

CORR의 세부 내용은 다음과 같다.

- 문법

corr



- 구성요소

구성요소	설명
expr1, expr2	expr1과 expr2는 수치 데이터 타입의 값을 반환하는 임의의 연산식이다.
OVER analytic_clause	OVER analytic_clause를 사용해 함수를 분석 함수로 수행할 수 있다. 자세한 내용은 “4.1.3. 분석 함수”의 analytic_clause를 참고한다.

- 예제

  다음은 CORR 함수를 사용하는 예이다.

```
SQL> SELECT CORR(AGE, SAL) FROM EMP;

CORR(AGE,SAL)
-------------
-.21144410174
```

```
1 row selected.
```

- 분석 함수 예제

  다음은 CORR 함수를 분석 함수로 사용하는 예이다.

```
SQL> SELECT DEPTNO, EMPNO, CORR(SAL, AGE)
       OVER (PARTITION BY DEPTNO) AS CORR
     FROM EMP;

    DEPTNO      EMPNO       CORR
---------- ---------- ----------
        10       7934 -.93645032
        10       7839 -.93645032
        10       7782 -.93645032
        20       7566 .567780056
        20       7788 .567780056
        20       7876 .567780056
        20       7902 .567780056
        20       7369 .567780056
        30       7654 -.33417865
        30       7698 -.33417865
        30       7521 -.33417865
        30       7499 -.33417865
        30       7844 -.33417865
        30       7900 -.33417865

14 rows selected.
```

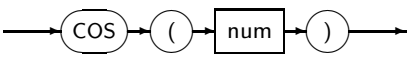
4.2.23. COS

COS는 주어진 파라미터 값의 코사인(Cosine)값을 구하는 함수이다.

COS의 세부 내용은 다음과 같다.

- 문법

  cos



- 구성요소

구성요소	설명
num	실수 값을 반환하는 임의의 연산식이다. (단위: 라디안)

구성요소	설명
	num은 숫자형 타입이거나 숫자형 타입으로 변환될 수 있는 타입이어야 한다. 만약 num의 타입이 BINARY_FLOAT일 경우엔 BINARY_DOUBLE 타입을 반환하고, 그렇지 않은 경우엔 num과 동일한 타입을 반환한다.

- 예제

다음은 COS 함수를 사용하는 예이다.

```
SQL> SELECT COS(360 * 3.14159265359/180) FROM DUAL;

COS(360 * 3.14159265359/180)
----------------------------
                           1

1 row selected.
```

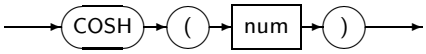
4.2.24. COSH

COSH는 주어진 파라미터 값의 하이퍼볼릭 코사인(Hyperbolic Cosine) 값을 구하는 함수이다.

COSH의 세부 내용은 다음과 같다.

- 문법

cosh



- 구성요소

구성요소	설명
num	실수 값을 반환하는 임의의 연산식이다. (단위: 라디안) num은 숫자형 타입이거나 숫자형 타입으로 변환될 수 있는 타입이어야 한다. 만약 num의 타입이 BINARY_FLOAT일 경우엔 BINARY_DOUBLE 타입을 반환하고, 그렇지 않은 경우엔 num과 동일한 타입을 반환한다.

- 예제

다음은 COSH 함수를 사용하는 예이다.

```
SQL> SELECT COSH(0) FROM DUAL;

   COSH(0)
----------
         1

1 row selected.
```

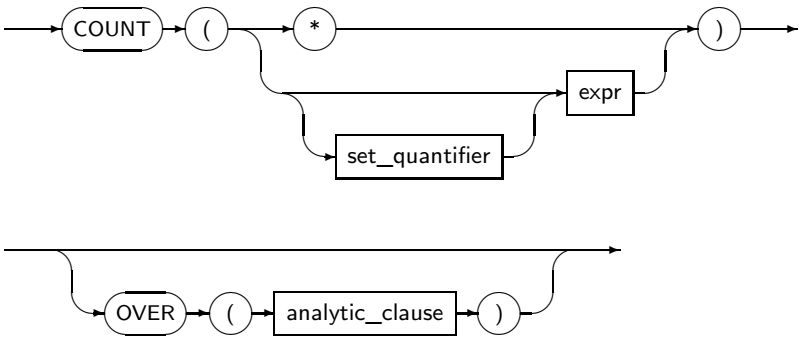
4.2.25. COUNT

COUNT는 쿼리가 반환하는 로우의 개수를 세는 함수이다. 분석 함수로도 사용할 수 있다. COUNT 함수는 항상 숫자를 반환하고 NULL을 반환하는 경우는 없다.

COUNT의 세부 내용은 다음과 같다.

- 문법

count



- 구성요소

구성요소	설명
set_quantifier	질의 결과에 중복된 로우의 허용, 비허용 여부를 지정한다. DISTINCT, UNIQUE, ALL을 지정할 수 있다. – DISTINCT, UNIQUE : 중복된 로우를 제거한다. – ALL : 모든 로우를 선택한다. (기본값)
*	애스터리스크(*)를 명시하면 중복된 로우와 그리고 NULL 값을 포함한 모든 로우를 개수에 포함시킨다.
expr	expr을 명시할 경우는, COUNT 함수는 expr의 값이 NULL이 아닐 경우에만 로우를 개수에 포함시킨다. DISTINCT를 명시함으로써 expr의 값을 구했을 때 중복을 제거한 로우만 셀 수 있다.

구성요소	설명
analytic_clause	OVER analytic_clause를 사용해 함수를 분석 함수로 수행할 수 있다. 자세한 내용은 “4.1.3. 분석 함수”의 analytic_clause를 참고한다.

- 예제

  다음은 COUNT 함수를 사용하는 예이다.

```
SQL> SELECT COUNT (*) FROM EMP;

  COUNT(*)
----------
         9

1 row selected.
```

- 분석 함수 예제

  다음은 COUNT 함수를 분석 함수로 사용하는 예이다.

```
SQL> SELECT NAME, SALARY, COUNT(*) OVER (ORDER BY SALARY RANGE
       BETWEEN 1000 PRECEDING AND 1000 FOLLOWING) AS W_COUNT
     FROM EMP;

NAME                     SALARY    W_COUNT
-------------------- ---------- ----------
Paul                       2000          4
Tom                        2500          5
Jill                       3000          6
Susan                      3000          6
Matt                       3200          5
Coon                       4000          5
Josh                       4500          2
Cathy                      6000          2
Brad                       6200          2

9 rows selected.
```

4.2.26. COVAR_POP

COVAR_POP는 expr1, expr2의 모공분산을 계산하는 함수이다. 분석 함수로도 사용할 수 있다.

이 함수는 모든 수치 데이터 타입과 수치 데이터 타입은 아니지만 묵시적으로 수치 데이터 타입으로 변환할 수 있는 데이터 타입을 파라미터로 받아 들인다. NUMBER 타입을 반환한다.

Tibero에서는 다음의 공식을 사용해 모공분산을 계산한다.

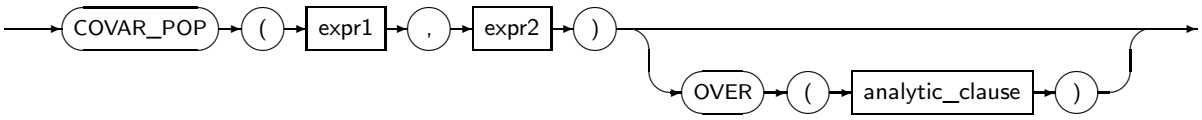
```
(SUM(expr1 * expr2) - SUM(expr2) * SUM(expr1) / n) / n
```

여기서 n은 expr1과 expr2 둘 모두가 NULL이 아닌 로우의 개수이다.

COVAR_POP의 세부 내용은 다음과 같다.

- 문법

covar_pop



- 구성요소

구성요소	설명
expr1, expr2	expr1과 expr2는 수치 데이터 타입의 값을 반환하는 임의의 연산식이다.
OVER analytic_clause	OVER analytic_clause를 사용해 함수를 분석 함수로 수행할 수 있다. 자세한 내용은 "4.1.3. 분석 함수"의 analytic_clause를 참고한다.

- 예제

다음은 COVAR_POP 함수를 사용하는 예이다.

```
SQL> SELECT COVAR_POP(AGE, SAL) AS COVAR_POP FROM EMP;

 COVAR_POP
----------
-642.09184

1 row selected.
```

- 분석 함수 예제

다음은 COVAR_POP 함수를 분석 함수로 사용하는 예이다.

```
SQL> SELECT DEPTNO, EMPNO, COVAR_POP(AGE, SAL)
       OVER (PARTITION BY DEPTNO) AS COVAR_POP
     FROM EMP;

    DEPTNO      EMPNO  COVAR_POP
---------- ---------- ----------
        10       7934 -4777.7778
        10       7839 -4777.7778
        10       7782 -4777.7778
        20       7566       1470
        20       7788       1470
```

```
        20       7876       1470
        20       7902       1470
        20       7369       1470
        30       7654 -480.55556
        30       7698 -480.55556
        30       7521 -480.55556
        30       7499 -480.55556
        30       7844 -480.55556
        30       7900 -480.55556

14 rows selected.
```

4.2.27. COVAR_SAMP

COVAR_SAMP는 expr1, expr2의 표본공분산을 계산하는 함수이다. 분석 함수로도 사용할 수 있다.

이 함수는 모든 수치 데이터 타입과 수치 데이터 타입은 아니지만 묵시적으로 수치 데이터 타입으로 변환할 수 있는 데이터 타입을 파라미터로 받아 들인다. NUMBER 타입을 반환한다.

Tibero에서는 다음의 공식을 사용해 표본공분산을 계산한다.

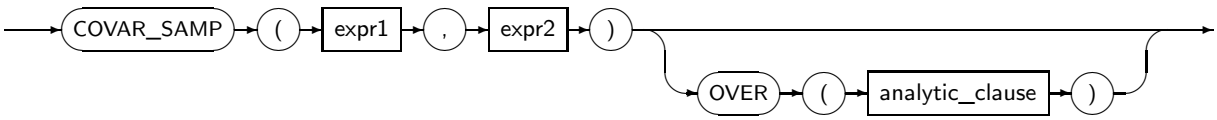
```
(SUM(expr1 * expr2) - SUM(expr1) * SUM(expr2) / n) / (n-1)
```

여기서 n은 expr1, expr2 모두 NULL이 아닌 로우의 개수이다.

COVAR_SAMP의 세부 내용은 다음과 같다.

- 문법

covar_samp



- 구성요소

구성요소	설명
expr1, expr2	expr1과 expr2는 수치 데이터 타입의 값을 반환하는 임의의 연산식이다.
OVER analytic_clause	OVER analytic_clause를 사용해 함수를 분석 함수로 수행할 수 있다. 자세한 내용은 “4.1.3. 분석 함수”의 analytic_clause를 참고한다.

- 예제

다음은 COVAR_SAMP 함수를 사용하는 예이다.

```
SQL> SELECT COVAR_SAMP(AGE, SAL) AS COVAR_SAMP FROM EMP;

COVAR_SAMP
----------
-691.48352

1 row selected.
```

- 분석 함수 예제

  다음은 COVAR_SAMP 함수를 분석 함수로 사용하는 예이다.

```
SQL> SELECT DEPTNO, EMPNO, COVAR_SAMP(AGE, SAL)
       OVER (PARTITION BY DEPTNO) AS COVAR_SAMP
     FROM EMP;

    DEPTNO      EMPNO COVAR_SAMP
---------- ---------- ----------
        10       7934 -7166.6667
        10       7839 -7166.6667
        10       7782 -7166.6667
        20       7566     1837.5
        20       7788     1837.5
        20       7876     1837.5
        20       7902     1837.5
        20       7369     1837.5
        30       7654 -576.66667
        30       7698 -576.66667
        30       7521 -576.66667
        30       7499 -576.66667
        30       7844 -576.66667
        30       7900 -576.66667

14 rows selected.
```

4.2.28. CUME_DIST

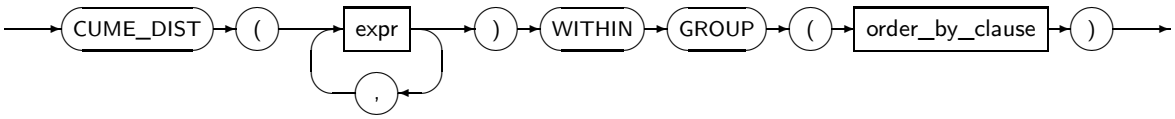
CUME_DIST는 그룹 내의 누적 분포를 계산하는 함수이다. 반환값은 NUMBER 타입이며 0 보다 크고 1 보다 작거나 같은 값을 가진다.

누적 분포를 계산하는 방법은 그룹의 로우를 정렬 스펙으로 정렬한 뒤 파라미터로 주어진 값을 갖는 가상의 로우를 삽입한 위치를 계산하여 그룹의 로우 개수로 나누는 것이다.

CUME_DIST의 세부 내용은 다음과 같다.

- 문법

cume_dist_aggregate



- 구성요소

구성요소	설명
expr	expr은 한 그룹 안에서는 상수 값이어야 하며, order_by_clasuse의 표현식과 대응되어야 한다.
order_by_clause	분할된 하나의 파티션 내에서 로우를 어떻게 정렬할지를 명시한다. 자세한 내용은 "4.1.3. 분석 함수"의 order_by_clause를 참고한다.

- 예제

다음은 CUME_DIST 함수를 사용하는 예이다.

```
SQL> SELECT CUME_DIST(1000, '1981/01/01') WITHIN GROUP
       (ORDER BY SAL, HIREDATE) AS "CUME_DIST"
     FROM EMP;

 CUME_DIST
----------
        .2

1 row selected.
```

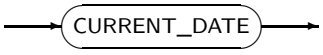
4.2.29. CURRENT_DATE

CURRENT_DATE는 현재 세션의 시간대를 기준으로 현재 날짜를 그레고리력으로 출력하는 함수이다.

CURRENT_DATE의 세부 내용은 다음과 같다.

- 문법

current_date



- 예제

다음은 CURRENT_DATE 함수를 사용하는 예이다.

```
SQL> SELECT CURRENT_DATE FROM DUAL;
```

```
CURRENT_DATE
--------------------------------------------------
2005/12/04

1 row selected.
```

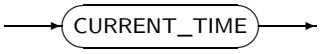
4.2.30. CURRENT_TIME

CURRENT_TIME은 현재 세션의 시간대를 기준으로 현재 시간을 출력하는 함수이다.

CURRENT_TIME의 세부 내용은 다음과 같다.

- 문법

current_time



- 예제

다음은 CURRENT_TIME 함수를 사용하는 예이다.

```
SQL> SELECT CURRENT_TIME FROM DUAL;

CURRENT_TIME
--------------------------------
20:23:18.383578

1 row selected.
```

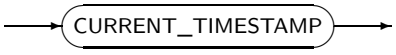
4.2.31. CURRENT_TIMESTAMP

CURRENT_TIMESTAMP는 현재 세션의 시간대를 기준으로 현재 날짜 및 시간을 출력하는 함수이다. 이 함수의 반환값은 TIMESTAMP WITH TIME ZONE 타입이다.

CURRENT_TIMESTAMP의 세부 내용은 다음과 같다.

- 문법

current_timestamp



- 예제

다음은 CURRENT_TIMESTAMP 함수를 사용하는 예이다.

```
SQL> SELECT CURRENT_TIMESTAMP FROM DUAL;

CURRENT_TIMESTAMP
-----------------------------------------
2005/12/04 20:22:26.391220 Asia/Seoul

1 row selected.
```

4.2.32. DBTIMEZONE

DBTIMEZONE는 데이터베이스의 시간대 정보를 오프셋([+|-]TZH:TZM) 형식이나 지역 이름(TZR) 형식으로 반환하는 함수이다.

DBTIMEZONE의 세부 내용은 다음과 같다.

- 문법

dbtimezone

- 예제

다음은 DBTIMEZONE 함수를 사용하는 예이다.

```
SQL> SELECT DBTIMEZONE FROM DUAL;

DBTIMEZONE
-----------------------------------
+00:00

1 row selected.
```

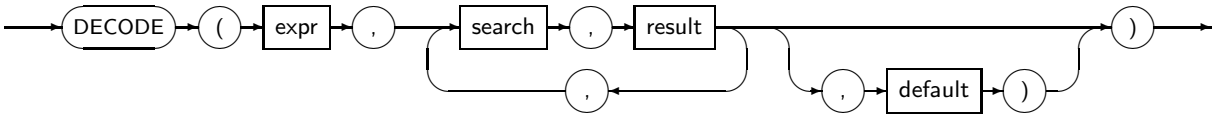
4.2.33. DECODE

DECODE는 각 search를 비교하여, 같은 값을 가지는 search에 대응하는 result 값을 구하는 함수이다. 수치, 문자 데이터 타입 모두 파라미터로 사용할 수 있다. expr, search, result 그리고 default를 합해서 최대 255까지의 표현식을 사용할 수 있다.

DECODE의 세부 내용은 다음과 같다.

- 문법

decode



- 구성요소

구성요소	설명
expr	search와 데이터 타입이 동일하거나 암묵적 변환을 통해 동일한 타입으로 변환이 가능해야 한다.
search	expr에 대응되는 조건값이다. 모든 search의 타입은 첫 번째 search의 타입과 동일하거나 데이터 타입의 암묵적 변환이 가능해야 한다.
result	search와 일치하는 경우 반환할 값이다. 모든 result의 타입은 첫 번째 result의 타입과 동일하거나 데이터 타입의 암묵적 변환이 가능해야 한다.
default	같은 값을 갖는 search가 없을 때 default에 명시된 값을 반환한다. default의 값이 명시되지 않은 경우 NULL을 반환한다. default의 타입은 첫 번째 result의 타입과 동일하거나 데이터 타입의 암묵적 변환이 가능해야 한다.

- 예제

다음은 DECODE 함수를 사용하는 예이다.

```
SQL> SELECT DECODE('1', 1, 'Male', 2, 'Female') FROM DUAL;

DECODE('1',1,'MALE',2,'FEMALE')
-------------------------------
Male

1 row selected.
```

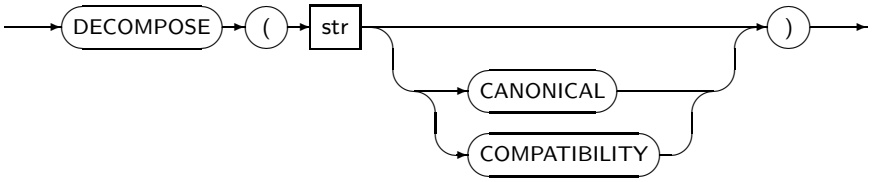
4.2.34. DECOMPOSE

DECOMPOSE는 입력받은 유니코드 문자열을 분해(decomposition)한 문자열을 반환하는 함수이다.

DECOMPOSE의 세부 내용은 다음과 같다.

- 문법

decompose



- 구성요소

구성요소	설명
str	문자열을 반환하는 임의의 연산식이다.
CANONICAL	COMPOSE 함수를 통해 원본 문자열을 복원할 수 있는 정규 분해를 한다.
COMPATIBILITY	호환 모드를 분해한다. 이 모드에서는 COMPOSE 함수를 통해 복원할 수 없다.

- 예제

다음은 DECOMPOSE 함수를 사용하는 예이다. (결과는 사용자의 문자 집합(Character set)에 따라 달라질 수 있음)

```
SQL> SELECT DECOMPOSE('Châteaux') FROM DUAL;

DECOMPOSE('Châteaux')
-------------------------------------------------------------
Cha^teaux

1 row selected.
```

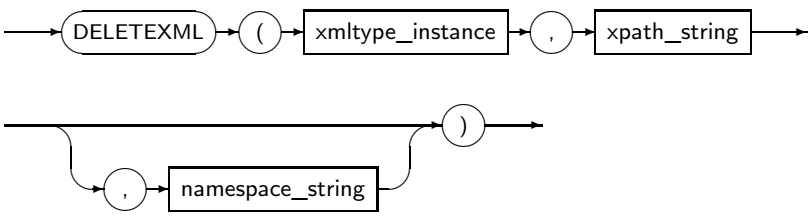
4.2.35. DELETEXML

DELETEXML은 XPath expression으로 지정된 노드를 XML에서 제거하는 함수이다.

DELETEXML의 세부 내용은 다음과 같다.

- 문법

deletexml



- 구성요소

구성요소	설명
xmltype_instance	XMLType instance다.
xpath_string	Xpath expression으로 지우려는 XML Element의 위치를 나타낸다.
namespace_string	XPath_string의 네임스페이스 정보를 나타낸다. VARCHAR 타입이어야 한다.

- 예제

다음은 DELETEXML 함수를 사용하는 예이다.

```
SQL> UPDATE warehouses SET warehouse_spec =
   DELETEXML(warehouse_spec, '/Warehouse/Building/Owner')
   WHERE warehouse_id = 2;

SQL> SELECT warehouse_id, warehouse_spec FROM warehouses
   WHERE warehouse_id in (2,3);

        ID WAREHOUSE_SPEC
---------- --------------------------------
         2
           <Warehouse>
             <Building>Rented</Building>
             <Area>50000</Area>
             <Docks>1</Docks>
             <DockType>Side load</DockType>
             <WaterAccess>Y</WaterAccess>
             <RailAccess>N</RailAccess>
             <Parking>Lot</Parking>
             <VClearance>12 ft</VClearance>
           </Warehouse>

         3
           <Warehouse>
             <Building>Rented
               <Owner>Grandco</Owner>
               <Owner>ThirdOwner</Owner>
               <Owner>LesserCo</Owner>
             </Building>
             <Area>85700</Area>
             <DockType />
             <WaterAccess>N</WaterAccess>
             <RailAccess>N</RailAccess>
             <Parking>Street</Parking>
             <VClearance>11.5 ft</VClearance>
           </Warehouse>
```

4.2.36. DENSE_RANK

DENSE_RANK는 각 그룹별로 로우를 정렬한 다음 그룹 내의 각 로우에 대한 순위를 반환하는 함수이다. 분석 함수로도 사용할 수 있다.

반환된 순위는 다음과 같은 특징이 있다.

- 데이터 타입은 NUMBER이다.
- 1부터 시작하는 연속적인 정수 값이다.
- 최댓값은 중복되는 값을 하나로 계산했을 때 전체 로우의 개수가 된다.
- 중복된 값이 나타났을 때 다음 순위의 값은 건너뛰지 않고 1이 증가한 값이 부여된다. 중복된 값은 모두 같은 순위의 값이 부여된다.

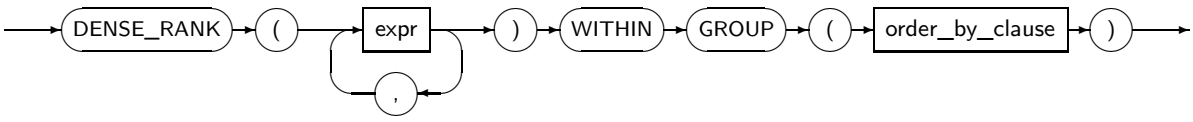
계산 방법은 다음과 같이 함수에 따라 달라진다.

함수	설명
집계 함수	파라미터 값으로 구성된 가상의 로우에 대한 순위 값을 계산하다. 파라미터는 각 그룹마다 상수 값을 가져야 하며 order_by_clause의 표현식과 대응되어야 한다.
분석 함수	각 로우의 그룹 내 순위를 반환한다. 순위는 order_by_clause 내의 expr 값을 기준으로 정렬한 결과가 부여된다.

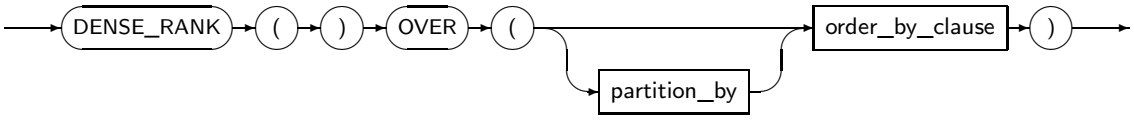
DENSE_RANK의 세부 내용은 다음과 같다.

- 문법

dense_rank_aggregation



dense_rank_analytic



- 구성요소
  - dense_rank_aggregation

구성요소	설명
expr	임의의 연산식이다.
order_by_clause	분할된 하나의 파티션 내에서 로우를 어떻게 정렬할지를 명시한다. 자세한 내용은 "4.1.3. 분석 함수"의 order_by_clause를 참고한다.

– dense_rank_analytic

구성요소	설명
partition_by	현재 질의 블록의 결과 집합을 expr 또는 expr의 리스트를 기준으로 분할한다. 자세한 내용은 "4.1.3. 분석 함수"의 partition_by를 참고한다.
order_by_clause	분할된 하나의 파티션 내에서 로우를 어떻게 정렬할지를 명시한다. 자세한 내용은 "4.1.3. 분석 함수"의 order_by_clause를 참고한다.

- 예제

다음은 DENSE_RANK 함수를 사용하는 예이다.

```
SQL> SELECT DEPTNO, DENSE_RANK(3000) WITHIN GROUP (ORDER BY SAL)
       AS DENSE_RANK
     FROM EMP GROUP BY DEPTNO;

    DEPTNO DENSE_RANK
---------- ----------
        10          3
        20          4
        30          6

3 rows selected.
```

- 분석 함수 예제

다음은 DENSE_RANK 함수를 분석 함수로 사용하는 예이다.

```
SQL> SELECT NAME, DEPTID, SALARY, DENSE_RANK()
       OVER (PARTITION BY DEPTID ORDER BY SALARY)
     FROM EMP;

NAME                    DEPTID     SALARY DENSE_RANK
-------------------- ---------- ---------- ----------
Paul                          1       3000          1
Angela                        1       3000          1
Nick                          1       3200          2
Scott                         1       4000          3
James                         1       4000          3
John                          1       4500          4
Joe                           2       4000          1
```

```
Brad                           2       4200          2
Daniel                         2       5000          3
Tom                            2       5000          3
Kathy                          2       5000          3
Bree                           2       6000          4

12 rows selected.
```

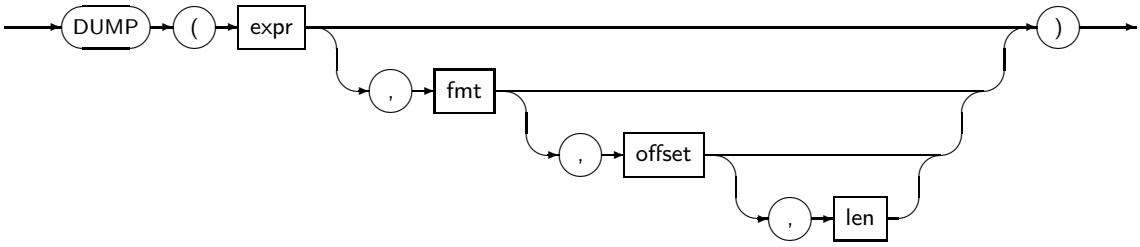
4.2.37. DUMP

DUMP는 함수는 파라미터로 주어진 expr의 내부의 표현 정보를 반환하는 함수이다. 반환되는 값은 바이트 스트림과 길이 정보이며 VARCHAR2 타입이다. 단, LONG, CLOB, BLOB 타입에는 이 함수를 적용할 수 없다.

DUMP의 세부 내용은 다음과 같다.

- 문법

dump



- 구성요소

구성요소	설명
expr	임의의 연산식이다.
fmt	바이트 스트림의 형식을 지정한다. fmt는 다음의 값을 가질 수 있다. – 8 : 8진법 표현 – 10 : 10진법 표현(기본값) – 16 : 16진법 표현 – 17 : 문자 표현
offset	시작 오프셋을 나타낸다.
len	표시할 byte의 길이를 지정한다. expr의 값이 NULL이면 NULL을 출력한다.

- 예제

다음은 DUMP 함수를 사용하는 예이다.

```
SQL> SELECT DUMP(100) FROM DUAL;

DUMP(100)
----------------
Len=2: 194,129

1 row selected.
```

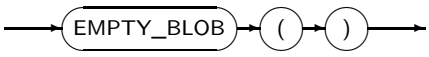
4.2.38. EMPTY_BLOB

EMPTY_BLOB은 BLOB 타입의 컬럼을 초기화하기 위해 빈 LOB Locator를 반환하는 함수이다.

EMPTY_BLOB의 세부 내용은 다음과 같다.

- 문법

  empty_blob

- 예제

  다음은 EMPTY_BLOB 함수를 사용하는 예이다.

```
SQL> UPDATE EMP SET PHOTO = EMPTY_BLOB();
```

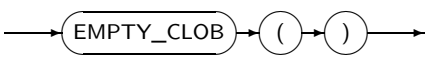
4.2.39. EMPTY_CLOB

EMPTY_CLOB은 CLOB 타입의 컬럼을 초기화하기 위해 비어 있는 LOB Locator를 반환하는 함수이다.

EMPTY_CLOB의 세부 내용은 다음과 같다.

- 문법

  empty_clob

- 예제

  다음은 EMPTY_CLOB 함수를 사용하는 예이다.

```
SQL> UPDATE NOVEL SET CONTENTS = EMPTY_CLOB();
```

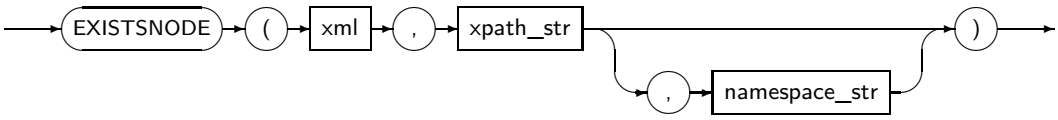
4.2.40. EXISTSNODE

EXISTSNODE는 XML 문서에서 질의한 XPath에 해당 노드가 있는지 검사하는 함수이다. 반환되는 값은 NUMBER 타입이며, 해당 노드가 있으면 1을 없으면 0을 반환한다.

EXISTSNODE의 세부 내용은 다음과 같다.

- 문법

existsnode



- 구성요소

구성요소	설명
xml	질의의 대상이 되는 XML 문서로, XML 타입이다.
xpath_str	질의할 XPath 문자열로, 최대 길이는 4000자이다.
namespace_str	XML 문서에서 네임스페이스가 필요할 때 사용하는 옵션으로 VARCHAR 타입이다.

- 예제

다음은 EXISTSNODE 함수를 사용하는 예이다.

```
SQL> SELECT EXTRACT(employee_xmldoc, '/employee/department/dname') dname
     FROM employee_xml
     WHERE EXISTSNODE(employee_xmldoc, '/employee/department/dname') = 1;

dname
----------------------
<dname>DB Lab</dname>

1 row selected.
```

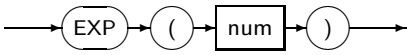
4.2.41. EXP

EXP는 자연 로그(Natural Log)의 밑(Base) e(= 2.7182818284…)의 주어진 파라미터의 제곱 값을 구하는 함수이다.

EXP의 세부 내용은 다음과 같다.

- 문법

exp



- 구성요소

구성요소	설명
num	수치 값을 반환하는 임의의 연산식이다. num은 숫자형 타입이거나 숫자형 타입으로 변환될 수 있는 타입이어야 한다. 만약 num의 타입이 BINARY_FLOAT일 경우엔 BINARY_DOUBLE 타입을 반환하고, 그렇지 않은 경우엔 num과 동일한 타입을 반환한다.

- 예제

다음은 EXP 함수를 사용하는 예이다.

```
SQL> SELECT EXP(2.0) FROM DUAL;

  EXP(2.0)
----------
 7.3890561

1 row selected.
```

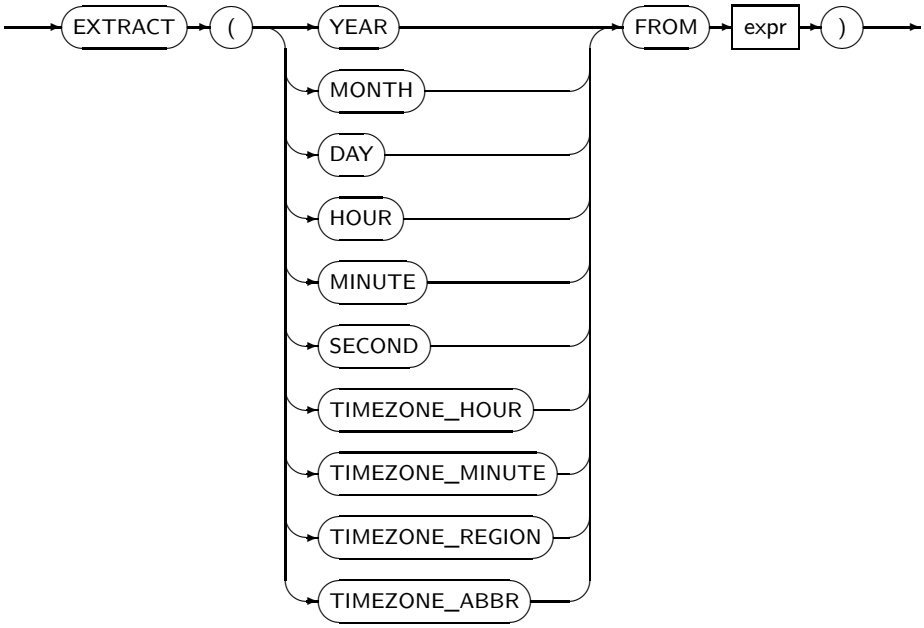
4.2.42. EXTRACT

EXTRACT는 일시(날짜형) 또는 시간 간격(Interval) 값에서 특정 부분의 값을 추출하고 싶을 때 사용하는 함수이다. 반환되는 값은 그레고리력을 따른다. 값을 추출할 때는 추출할 값이 원래의 연산식에 존재해야만 한다.

EXTRACT의 세부 내용은 다음과 같다.

- 문법

extract



- 구성요소

구성요소	설명
YEAR	연도를 저장하는 날짜형 데이터 타입이다.
MONTH	달을 저장하는 날짜형 데이터 타입이다.
DAY	날짜를 저장하는 데이터 타입이다.
HOUR	시를 저장하는 데이터 타입이다.
MINUTE	분을 저장하는 데이터 타입이다.
SECOND	초를 저장하는 데이터 타입이다.
TIMEZONE_HOUR	시간대 오프셋 시를 저장하는 데이터 타입이다.
TIMEZONE_MINUTE	시간대 오프셋 분을 저장하는 데이터 타입이다.
TIMEZONE_REGION	시간대 지역 이름을 저장하는 데이터 타입이다.
TIMEZONE_ABBR	시간대 일광절약시간 약어를 저장하는 데이터 타입이다.

- 예제

  다음은 EXTRACT 함수를 사용하는 예이다.

```
SQL> SELECT EXTRACT (MONTH FROM DATE'1996-04-01') FROM DUAL;

EXTRACT(MONTHFROMDATE'1996-04-01')
----------------------------------
                                 4

1 row selected.
```

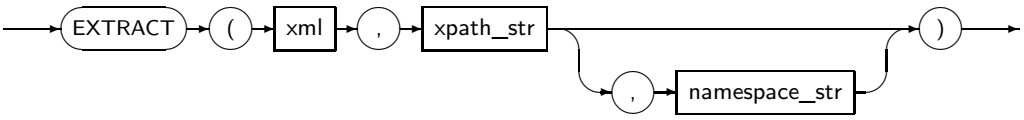
4.2.43. EXTRACT(XML)

EXTRACT(XML)는 XML 문서에서 XPath를 이용하여 해당 XML의 노드를 반환하는 함수이다.

EXTRACT(XML)의 세부 내용은 다음과 같다.

- 문법

extract_xml



- 구성요소

구성요소	설명
xml	질의의 대상이 되는 XML 문서로, XML 타입이다.
xpath_str	질의할 XPath 문자열로, 최대 길이는 4000자이다.
namespace_str	XML 문서에서 네임스페이스가 필요할 때 사용하는 옵션으로 VARCHAR 타입이다.

- 예제

다음은 EXTRACT(XML) 함수를 사용하는 예이다.

```
SQL> SELECT EXTRACT(employee_xmldoc, '/employee/department/dname') dname
     FROM employee_xml;

dname
---------------------
<dname>DB Lab</dname>

1 row selected.
```

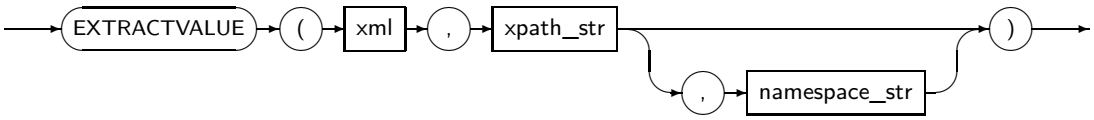
4.2.44. EXTRACTVALUE

EXTRACTVALUE는 XML 문서에서 XPath를 이용하여 해당 값을 VARCHAR 타입으로 반환하는 함수이다.

EXTRACTVALUE의 세부 내용은 다음과 같다.

- 문법

extract_value



- 구성요소

구성요소	설명
xml	질의의 대상이 되는 XML 문서로, XML 타입이다.
xpath_str	질의할 XPath 문자열로, 최대 길이는 4000자이다.
namespace_str	XML 문서에서 네임스페이스가 필요할 때 사용하는 옵션으로 VARCHAR 타입이다.

- 예제

다음은 EXTRACTVALUE 함수를 사용하는 예이다.

```
SQL> SELECT EXTRACTVALUE(employee_xmldoc, '/employee/department/dname') dname
     FROM employee_xml;

dname
------
DB Lab

1 row selected.
```

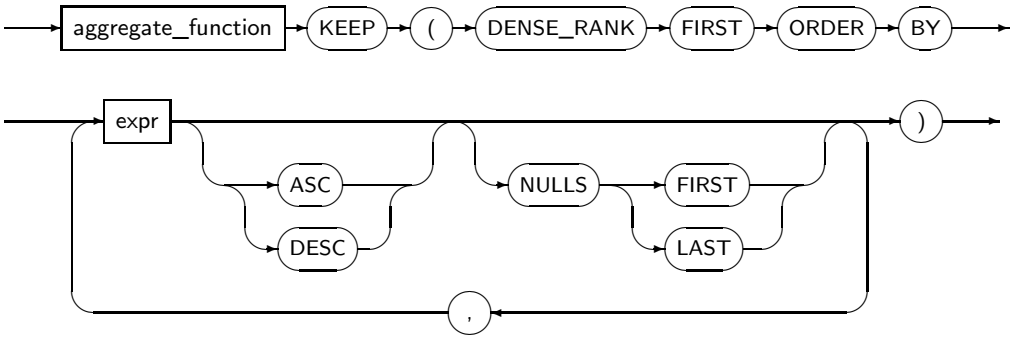
4.2.45. FIRST

FIRST는 정렬된 로우에서 처음에 해당하는 로우를 뽑아내어 명시된 집단함수를 적용한 결과를 반환한다. 현재 분석 함수는 지원하지 않는다.

FIRST 세부 내용은 다음과 같다.

- 문법

first



- 구성요소

구성요소	설명
aggregate_function	사용가능한 집단 함수는 AVG, COUNT, MIN, MAX, SUM, STADDEV, VARIANCE이다.
expr	임의의 연산식이다.

- 예제

다음은 FIRST 함수를 사용하는 예이다.

```
SQL> SELECT MIN(COMM) KEEP (DENSE_RANK FIRST ORDER BY SAL),
MAX(COMM) KEEP (DENSE_RANK FIRST ORDER BY SAL)
FROM EMP WHERE JOB = 'SALESMAN';

MIN(COMM)KEEP(DENSE_RANKFIRSTORDERBYSAL) MAX(COMM)KEEP(DENSE_RANKFIRSTORDERBYSAL)
---------------------------------------- ----------------------------------------
                                     500                                    1400

1 row selected.
```

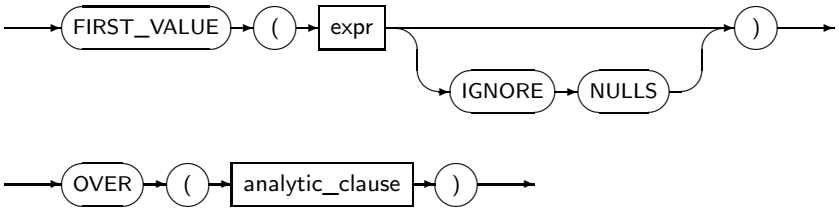
4.2.46. FIRST_VALUE

FIRST_VALUE는 정렬된 로우에서 첫 번째 값을 반환하는 분석함수이다. IGNORE NULLS를 명시하면 NULL이 아닌 첫 번째 값을 반환한다. 만약 모든 값이 NULL이라면 NULL을 반환한다.

FIRST_VALUE 세부 내용은 다음과 같다.

- 문법

first_value



- 구성요소

구성요소	설명
expr	임의의 연산식이다.

- 예제

  다음은 FIRST_VALUE 함수를 사용하는 예이다.

```
SELECT DEPTNO, ENAME, SAL, FIRST_VALUE(ENAME) OVER
(PARTITION BY DEPTNO ORDER BY SAL ASC ROWS UNBOUNDED PRECEDING) AS LOWEST_SAL
FROM EMP;

    DEPTNO ENAME             SAL LOWEST_SAL
---------- ---------- ---------- ----------
        10 MILLER           1300 MILLER
        10 CLARK            2450 MILLER
        10 KING             5000 MILLER
        20 SMITH             800 SMITH
        20 ADAMS            1100 SMITH
        20 JONES            2975 SMITH
        20 SCOTT            3000 SMITH
        20 FORD             3000 SMITH
        30 JAMES             950 JAMES
        30 WARD             1250 JAMES
        30 MARTIN           1250 JAMES
        30 TURNER           1500 JAMES
        30 ALLEN            1600 JAMES
        30 BLAKE            2850 JAMES

14 rows selected.
```

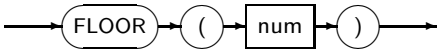
4.2.47. FLOOR

FLOOR는 주어진 파라미터의 값보다 작거나 같은 가장 큰 정수를 구하는 함수이다.

FLOOR의 세부 내용은 다음과 같다.

- 문법

floor



- 구성요소

구성요소	설명
num	수치 값을 반환하는 임의의 연산식이다.

- 예제

다음은 FLOOR 함수를 사용하는 예이다.

```
SQL> SELECT FLOOR(15.5), FLOOR(-15.5), FLOOR(25.0) FROM DUAL;

FLOOR(15.5) FLOOR(-15.5) FLOOR(25.0)
----------- ------------ -----------
         15          -16          25

1 row selected.
```

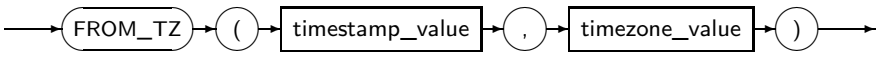
4.2.48. FROM_TZ

FROM_TZ는 주어진 TIMESTAMP 값과 시간대를 이용하여 TIMESTAMP WITH TIME ZONE 값으로 바꿔 주는 함수이다.

FROM_TZ의 세부 내용은 다음과 같다.

- 문법

from_tz



- 구성요소

구성요소	설명
timestamp_value	시간값을 반환하는 임의의 연산식이다.
timezone_value	시간대 지역 이름이나 오프셋을 반환하는 임의의 연산식이다.

- 예제

다음은 FROM_TZ 함수를 사용하는 예이다.

```
SQL> SELECT FROM_TZ(TIMESTAMP '2002/01/24 08:48:53', '8:00') FROM DUAL;

FROM_TZ(TIMESTAMP'2002/01/2408:48:53','8:00')
---------------------------------------------
2002/01/24 08:48:53.000000 +08:00

1 row selected.
```

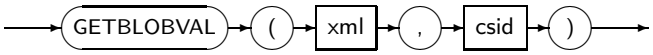
4.2.49. GETBLOBVAL

GETBLOBVAL는 XML 타입을 BLOB 형태로 변환하여 반환하는 함수이다.

GETBLOBVAL의 세부 내용은 다음과 같다.

- 문법

getblobval



- 구성요소

구성요소	설명
xml	BLOB 형태로 변환할 XML 타입 객체이다.
csid	BLOB 형태로 변환할 때 사용할 characterset의 ID이다.

- 예제

다음은 GETBLOBVAL 함수를 사용하는 예이다.

```
SQL> SELECT GETBLOBVAL(XMLTYPE('<a>1</a>'), NLS_CHARSET_ID('MSWIN949')) FROM DUAL;

GETBLOBVAL(XMLTYPE('<A>1</A>'),NLS_CHARSET_ID('MSWIN949'))
----------------------------------------------------------------------------
3C613E313C2F613E

1 row selected.
```

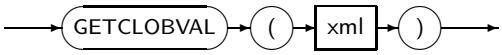
4.2.50. GETCLOBVAL

GETCLOBVAL는 XML 타입을 CLOB형태로 변환하여 반환하는 함수이다.

GETCLOBVAL의 세부 내용은 다음과 같다.

- 문법

getclobval



- 구성요소

구성요소	설명
xml	CLOB 형태로 변환할 XML 타입 객체이다.

- 예제

다음은 GETCLOBVAL 함수를 사용하는 예이다.

```
SQL> SELECT GETCLOBVAL(XMLTYPE('<a>1</a>')) FROM DUAL;
GETCLOBVAL(XMLTYPE('<A>1</A>'))
--------------------------------------------------------------------------------
<a>1</a>

1 row selected.
```

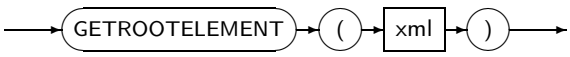
4.2.51. GETROOTELEMENT

GETROOTELEMENT는 XML 타입 instance의 root element를 VARCHAR 타입으로 반환하는 함수이다.

GETROOTELEMENT의 세부 내용은 다음과 같다.

- 문법

getrootelement



- 구성요소

구성요소	설명
xml	XML 타입 instance이다.

- 예제

다음은 GETROOTELEMENT 함수를 사용하는 예이다.

```
SQL> SELECT GETROOTELEMENT(XMLTYPE('<a>1</a>')) FROM DUAL;
GETROOTELEMENT(XMLTYPE('<A>1</A>'))
--------------------------------------------------------------------------------
```

```
a

1 row selected.
```

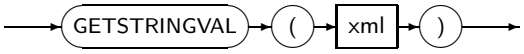
4.2.52. GETSTRINGVAL

GETSTRINGVAL는 XML 타입을 VARCHAR 타입으로 변환하여 반환하는 함수이다.

GETSTRINGVAL의 세부 내용은 다음과 같다.

- 문법

getstringval



- 구성요소

구성요소	설명
xml	VARCHAR 타입으로 변환할 XML 타입 객체이다.

- 예제

다음은 GETSTRINGVAL 함수를 사용하는 예이다.

```
SQL> SELECT GETSTRINGVAL(XMLTYPE('<a>1</a>')) FROM DUAL;
GETSTRINGVAL(XMLTYPE('<A>1</A>'))
--------------------------------------------------------------------------------
<a>1</a>

1 row selected.
```

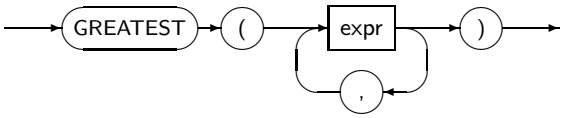
4.2.53. GREATEST

GREATEST는 파라미터 중 가장 큰 값을 구하는 함수이다.

GREATEST의 세부 내용은 다음과 같다.

- 문법

greatest



- 구성요소

구성요소	설명
expr	임의의 연산식이다. 처음 expr의 타입이 함수의 반환 타입이 된다. 두 번째 expr부터는 처음 expr과 동일한 타입 또는 암묵적 변환이 가능한 타입이어야 한다. expr 중 하나의 값이 NULL이면 함수는 NULL을 반환한다.

- 예제

다음은 GREATEST 함수를 사용하는 예이다.

```
SQL> SELECT GREATEST(1, 3, 2) FROM DUAL;

GREATEST(1,3,2)
---------------
              3

1 row selected.
```

4.2.54. GROUPING

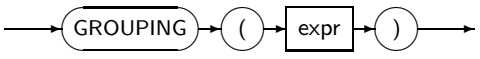
GROUPING은 Superaggreagate 로우와 Regular grouped 로우를 구분하기 위하여 사용하는 함수이다.

ROLLUP과 CUBE는 Superaggreagate 로우를 생성하는데, 이때 모든 값의 집합은 NULL로 표현된다. GROUPING 함수를 사용하면 로우의 컬럼 값 NULL과 Superaggreagate 로우에서 모든 값의 집합을 나타내기 위한 값 NULL을 구분할 수 있다. expr의 값이 모든 값의 집합을 나타낸다. NULL일 경우에는 1을 반환하고, 그렇지 않을 경우에는 0을 반환한다. 반환된 값은 NUMBER 타입이다.

GROUPING의 세부 내용은 다음과 같다.

- 문법

grouping



- 구성요소

구성요소	설명
expr	GROUP BY 절에 명시된 expr 중의 하나와 대응해야 한다.

- 예제

  다음은 GROUPING 함수를 사용하는 예이다.

```
SQL> SELECT
       DECODE(GROUPING(DNO),1,'ALL',DNO) AS DNO,
       DECODE(GROUPING(JOB),1,'ALL',JOB) AS JOB,
       SUM(PAY) AS PAY
     FROM PERSONNEL
     GROUP BY CUBE(DNO, JOB)
     ORDER BY DNO, JOB;

       DNO JOB                         PAY
---------- -------------------- ----------
        10 ANALYST                    5950
        10 MANAGER                    1000
        10 PRESIDENT                  7000
        10 ALL                       13950
        20 CLERK                      4000
        20 MANAGER                    3974
        20 ALL                        7974
        30 MANAGER                    3550
        30 SALESMAN                   4250
        30 ALL                        7800
       ALL ANALYST                    5950
       ALL CLERK                      4000
       ALL MANAGER                    8524
       ALL PRESIDENT                  7000
       ALL SALESMAN                   4250
       ALL ALL                       29724

16 rows selected.
```

4.2.55. GROUPING_ID

GROUPING_ID는 로우의 GROUPING 비트 벡터(Bit Vector)에 해당하는 NUMBER를 반환하는 함수이다.

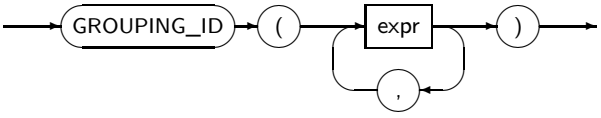
GROUPING_ID는 여러 GROUPING 함수의 결과를 비트 벡터로 통합하는 것과 같다. GROUPING_ID 함수를 사용하면 다수의 GROUPING 함수를 사용하는 것을 피할 수 있으며, 로우 필터링의 조건을 보다 쉽게 표현할 수 있다. GROUPING_ID를 이용한 로우 필터링은 GROUPING_ID = n과 같은 조건으로 간단히 처리할 수 있다.

GROUPING_ID는 GROUPING 함수와 ROLLUP 또는 CUBE가 사용된 SELECT 문에서만 사용할 수 있다. GROUP BY가 여러 번 사용된 질의의 경우 특정 로우의 GROUP BY 수준을 결정하기 위해서는 GROUPING 함수를 여러 번 사용해야 하는데, 이는 복잡한 SQL 문장을 생성하게 된다. GROUPING_ID는 이러한 경우에 유용하다.

GROUPING_ID의 세부 내용은 다음과 같다.

- 문법

grouping_id



- 구성요소

구성요소	설명
expr	GROUP BY clause에 명시된 expr 중의 하나와 대응해야 한다.

- 예제

다음은 GROUPING_ID 함수를 사용하는 예이다.

```
SQL> SELECT
       DECODE(GROUPING(DNO),1,'ALL',DNO) AS DNO,
       DECODE(GROUPING(JOB),1,'ALL',JOB) AS JOB,
       GROUPING(DNO) AS GD,
       GROUPING(JOB) AS GJ,
       GROUPING_ID(DNO, JOB) AS DJ,
       GROUPING_ID(DNO, JOB) AS JD,
       SUM(PAY) AS PAY
     FROM PERSONNEL
     GROUP BY CUBE(DNO, JOB)
     ORDER BY DNO, JOB;

       DNO JOB                   GD   GJ   DJ   JD         PAY
---------- -------------------- ---- ---- ---- ----  ----------
        10 ANALYST                 0    0    0    0        5950
        10 MANAGER                 0    0    0    0        1000
        10 PRESIDENT               0    0    0    0        7000
        10 ALL                     0    1    1    2       13950
        20 CLERK                   0    0    0    0        4000
        20 MANAGER                 0    0    0    0        3974
        20 ALL                     0    1    1    2        7974
        30 MANAGER                 0    0    0    0        3550
        30 SALESMAN                0    0    0    0        4250
```

```
        30 ALL                    0    1    1    2        7800
       ALL ANALYST                1    0    2    1        5950
       ALL CLERK                  1    0    2    1        4000
       ALL MANAGER                1    0    2    1        8524
       ALL PRESIDENT              1    0    2    1        7000
       ALL SALESMAN               1    0    2    1        4250
       ALL ALL                    1    1    3    3       29724

16 rows selected.
```

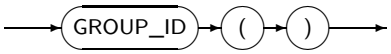
4.2.56. GROUP_ID

GROUP_ID는 GROUP BY의 결과에서 중복된 그룹을 구분하는 함수이다. 이는 쿼리 결과에서 중복된 그룹을 걸러내는 데 유용하게 쓰인다. 중복된 그룹 로우를 구분할 수 있는 NUMBER 타입의 값을 반환한다. 이 함수는 GROUP BY 절이 사용된 SELECT 문에만 사용할 수 있다.

GROUP_ID의 세부 내용은 다음과 같다.

- 문법

  group_id

- 예제

  다음은 GROUP_ID 함수를 사용하는 예이다.

```
SQL> SELECT depart_num, group_id()
     FROM employees
     GROUP BY depart_num;

   DEPART_NUM GROUP_ID()
------------- ----------
           10          1
           20          0
           30          0

3 rows selected.
```

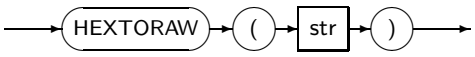
4.2.57. HEXTORAW

HEXTORAW는 16진수로 표현된 문자열의 RAW 값을 구하는 함수이다.

HEXTORAW의 세부 내용은 다음과 같다.

- 문법

hextoraw



- 구성요소

구성요소	설명
str	16진수 형태의 문자열 값을 반환하는 임의의 연산식이다.

- 예제

다음은 RAW 컬럼이 있는 테이블을 만들고, HEXTORAW 함수를 이용해 16진수의 값을 컬럼에 삽입하는 예이다.

```
SQL> SELECT HEXTORAW(UTL_RAW.CAST_TO_RAW('DB')) COL FROM DUAL;

COL
-------------------------
4442

1 row selected.
```

4.2.58. INET_ATON

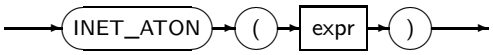
INET_ATON은 문자열 형식의 네트워크 주소를 파라미터로 받아 들여 그 주소에 해당하는 수치 값을 반환하는 함수이다. 4bytes와 8bytes의 주소 모두 사용할 수 있다.

반환된 값을 저장하기 위해서는 UNSIGEND INT를 사용해야 한다. SIGNED INT를 사용하면 주소의 첫 옥텟(Octet)의 값이 127보다 크면 올바로 저장되지 않는다.

INET_ATON의 세부 내용은 다음과 같다.

- 문법

inet_aton



- 구성요소

구성요소	설명
expr	문자열 형식의 네트워크 주소이다.

- 예제

다음은 INET_ATON 함수를 사용하는 예이다.

```
SQL> SELECT INET_ATON('123.255.0.1') FROM DUAL;

INET_ATON('123.255.0.1')
------------------------
              2080309249

1 row selected.
```

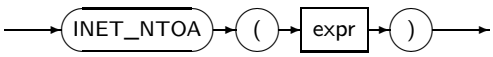
4.2.59. INET_NTOA

INET_NTOA는 수치 값의 네트워크 주소를 받아들여 문자열 형식으로 된 네트워크 주소를 반환하는 함수이다.

INET_NTOA의 세부 내용은 다음과 같다.

- 문법

inet_ntoa



- 구성요소

구성요소	설명
expr	네트워크 주소를 수치 값으로 표현한 것이다.

- 예제

다음은 INET_NTOA 함수를 사용하는 예이다.

```
SQL> SELECT INET_NTOA(2080309249) FROM DUAL;

INET_NTOA(2080309249)
-----------------------------------------------------------------
123.255.0.1
```

```
1 row selected.
```

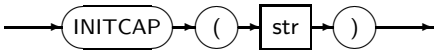
4.2.60. INITCAP

INITCAP는 첫 문자는 대문자, 나머지는 소문자로 변환된 문자열을 구하는 함수이다.

INITCAP의 세부 내용은 다음과 같다.

- 문법

  initcap

- 구성요소

구성요소	설명
str	문자열을 반환하는 임의의 연산식이다.

- 예제

  다음은 INITCAP 함수를 사용하는 예이다.

```
SQL> SELECT INITCAP('tiBero') FROM DUAL;

INITCAP('TIBERO')
-----------------
Tibero

1 row selected.
```

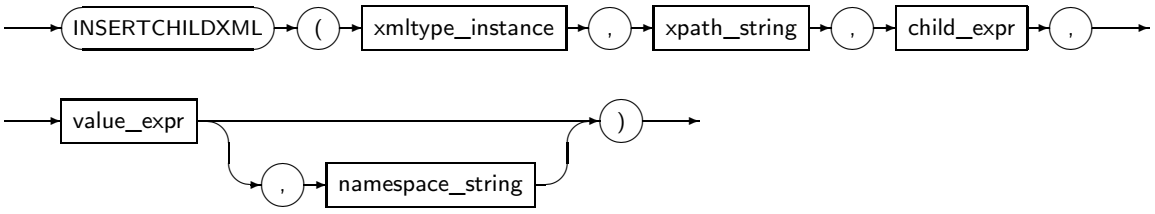
4.2.61. INSERTCHILDXML

INSERTCHILDXML은 XPath expression으로 지정된 노드의 자식 노드에 사용자가 입력한 XML 값을 삽입하는 함수로, INSERTXMLBEFORE 함수와 차이가 있다.

INSERTCHILDXML의 세부 내용은 다음과 같다.

- 문법

insertchildxml



- 구성요소

구성요소	설명
XMLType_instance	XML 타입 객체를 반환하는 임의의 연산식이다.
XPath_string	하나 이상의 자식 노드가 삽입될 위치를 나타내는 XPath 연산식이다.
child_expr	삽입될 자식 노드의 원소나 속성을 나타내는 임의의 연산식이다.
value_expr	삽입될 하나 이상의 자식 노드를 나타내는 임의의 연산식이다. 이 연산식은 반드시 문자열로 변환 가능해야 한다.
namespace_string	XPath의 네임스페이스 정보를 나타낸다. 반드시 VARCHAR 타입이어야 한다.

- 예제

다음은 INSERTCHILDXML 함수를 사용하는 예이다.

```
... INFO 컬럼 '<dept>research</dept>' ...
SQL> UPDATE EMP SET INFO =
   INSERTCHILDXML(INFO, '/dept', 'id', XMLTYPE('<id>1</id>'));

SQL> SELECT INFO FROM EMP;

EMP
-----------------------------------------
<dept>research<id>1</id></dept>
```

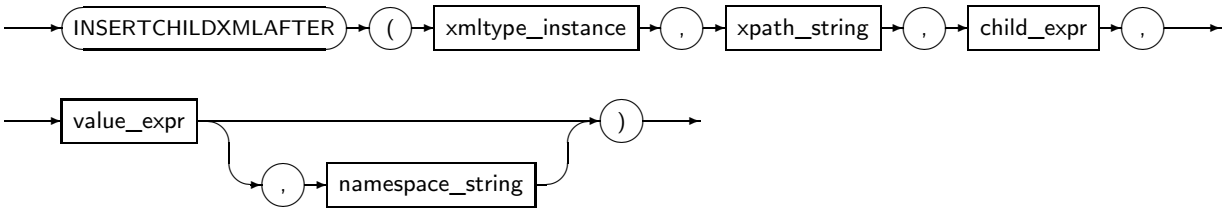
4.2.62. INSERTCHILDXMLAFTER

INSERTCHILDXMLAFTER은 XPath expression으로 지정된 노드에 사용자가 입력한 XML 값을 삽입하는 함수이다. 이전에 존재하고 있는 자식 노드들의 뒤부터 삽입된다.

INSERTCHILDXMLAFTER의 세부 내용은 다음과 같다.

- 문법

insertchildxmlafter



- 구성요소

구성요소	설명
XMLType_instance	XML 타입 객체를 반환하는 임의의 연산식이다.
XPath_string	하나 이상의 자식 노드가 삽입될 위치를 나타내는 XPath 연산식이다.
child_expr	삽입될 자식 노드의 원소나 속성을 나타내는 임의의 연산식이다.
value_expr	삽입될 하나 이상의 자식 노드를 나타내는 임의의 연산식이다. 이 연산식은 반드시 문자열로 변환 가능해야 한다.
namespace_string	XPath의 네임스페이스 정보를 나타낸다. 반드시 VARCHAR 타입이어야 한다.

- 예제

다음은 INSERTCHILDXMLAFTER 함수를 사용하는 예이다.

```
... INFO 컬럼 '<dept><id>1</id></dept>' ...
SQL> UPDATE EMP SET INFO =
   INSERTCHILDXMLAFTER(INFO, '/dept', 'id[1]', XMLTYPE('<id>2</id>'));

SQL> SELECT INFO FROM EMP;

EMP
--------------------------------------------
<dept><id>1</id><id>2</id></dept>
```

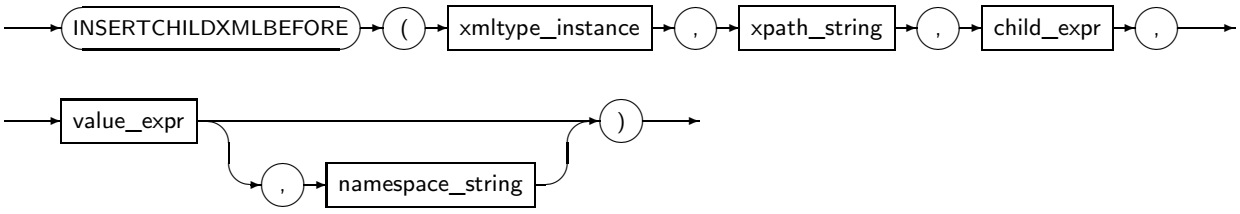
4.2.63. INSERTCHILDXMLBEFORE

INSERTCHILDXMLBEFORE은 XPath expression으로 지정된 노드에 사용자가 입력한 XML 값을 삽입하는 함수이다. 이전에 존재하고 있는 자식 노드들의 앞부터 삽입된다.

INSERTCHILDXMLBEFORE의 세부 내용은 다음과 같다.

- 문법

insertchildxmlbefore



- 구성요소

구성요소	설명
XMLType_instance	XML 타입 객체를 반환하는 임의의 연산식이다.
XPath_string	하나 이상의 자식 노드가 삽입될 위치를 나타내는 XPath 연산식이다.
child_expr	삽입될 자식 노드의 원소나 속성을 나타내는 임의의 연산식이다.
value_expr	삽입될 하나 이상의 자식 노드를 나타내는 임의의 연산식이다. 이 연산식은 반드시 문자열로 변환 가능해야 한다.
namespace_string	XPath의 네임스페이스 정보를 나타낸다. 반드시 VARCHAR 타입이어야 한다.

- 예제

다음은 INSERTCHILDXMLBEFORE 함수를 사용하는 예이다.

```
... INFO 컬럼 '<dept><id>1</id></dept>' ...
SQL> UPDATE EMP SET INFO =
   INSERTCHILDXMLBEFORE(INFO, '/dept', 'id[1]', XMLTYPE('<id>2</id>'));

SQL> SELECT INFO FROM EMP;

EMP
-----------------------------------------
<dept><id>2</id><id>1</id></dept>
```

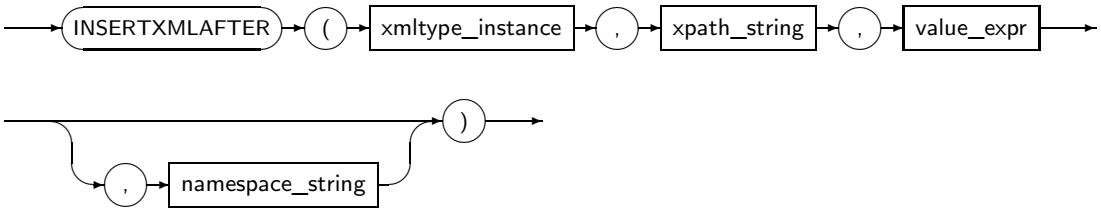
4.2.64. INSERTXMLAFTER

INSERTXMLAFTER은 XPath expression으로 지정된 노드에 사용자가 입력한 XML 값을 삽입하는 함수이다. 이전에 존재하고 있는 노드들의 뒤부터 삽입된다.

INSERTXMLAFTER의 세부 내용은 다음과 같다.

- 문법

insertxmlafter



- 구성요소

구성요소	설명
XMLType_instance	XML 타입 객체를 반환하는 임의의 연산식이다.
XPath_string	하나 이상의 자식 노드가 삽입될 위치를 나타내는 XPath 연산식이다.
value_expr	삽입될 하나 이상의 자식 노드를 나타내는 임의의 연산식이다. 이 연산식은 반드시 문자열로 변환 가능해야 한다.
namespace_string	XPath의 네임스페이스 정보를 나타낸다. 반드시 VARCHAR 타입이어야 한다.

- 예제

다음은 INSERTXMLAFTER 함수를 사용하는 예이다.

```
... INFO 컬럼 '<dept>research</dept>' ...
SQL> UPDATE EMP SET INFO =
   INSERTXMLAFTER(INFO, '/dept', XMLTYPE('<dept>sales</dept>'));

SQL> SELECT INFO FROM EMP;

EMP
----------------------------------------
<dept>research</dept>
<dept>sales</dept>
```

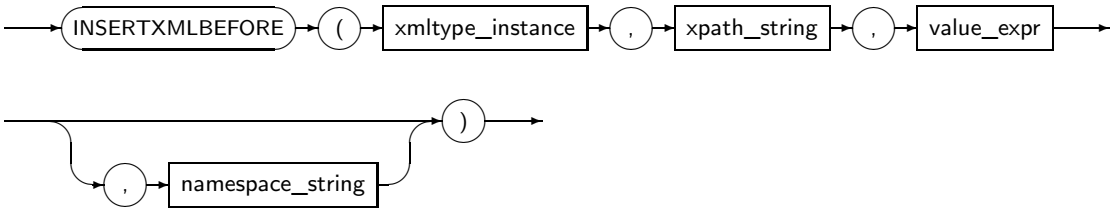
4.2.65. INSERTXMLBEFORE

INSERTXMLBEFORE은 XPath expression으로 지정된 노드에 사용자가 입력한 XML 값을 삽입하는 함수이다. 이전에 존재하고 있는 노드들의 이전부터 삽입된다.

INSERTXMLBEFORE의 세부 내용은 다음과 같다.

- 문법

insertxmlbefore



- 구성요소

구성요소	설명
XMLType_instance	XML 타입 객체를 반환하는 임의의 연산식이다.
XPath_string	하나 이상의 자식 노드가 삽입될 위치를 나타내는 XPath 연산식이다.
value_expr	삽입될 하나 이상의 자식 노드를 나타내는 임의의 연산식이다. 이 연산식은 반드시 문자열로 변환 가능해야 한다.
namespace_string	XPath의 네임스페이스 정보를 나타낸다. 반드시 VARCHAR 타입이어야 한다.

- 예제

다음은 INSERTXMLBEFORE 함수를 사용하는 예이다.

```
... INFO 컬럼 '<dept>research</dept>' ...
SQL> UPDATE EMP SET INFO =
   INSERTXMLBEFORE(INFO, '/dept', XMLTYPE('<dept>sales</dept>'));

SQL> SELECT INFO FROM EMP;

EMP
----------------------------------------
<dept>sales</dept>
<dept>research</dept>
```

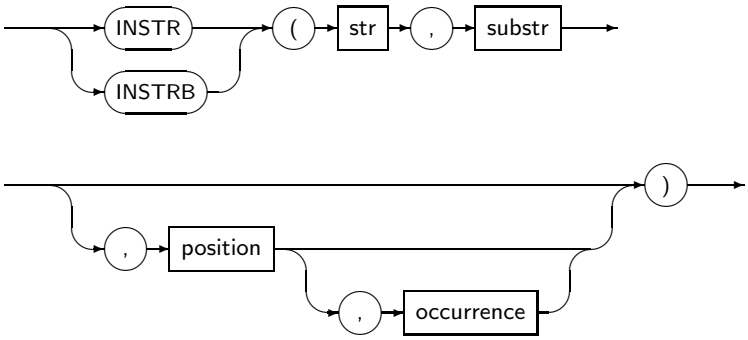
4.2.66. INSTR

INSTR은 문자열 str 내에서 문자열 substr을 찾아 그 위치를 반환하는 함수이다. 추가로 INSTRB 함수는 문자 대신 byte 단위로 위치를 계산하여 그 위치를 반환한다.

INSTR, INSTRB의 세부 내용은 다음과 같다.

- 문법

instr



- 구성요소

구성요소	설명
str, substr	모두 문자열을 반환하는 임의의 연산식이다. 만약 문자열 str 내에서 문자열 substr을 발견하지 못하면 0을 반환한다. 문자열의 위치 값은 1부터 시작된다.
position	0이 아닌 정수 값을 반환하는 임의의 연산식이다. (기본값: 1) position이 주어지면 문자열 str의 position 위치에서부터 탐색을 시작한다. 만약 position이 음수이면 문자열 str의 뒤에서부터 탐색을 시작한다.
occurrence	0이 아닌 정수 값을 반환하는 임의의 연산식이다. (기본값: 1) occurrence가 주어지면 탐색 문자열 내에서 occurrence번째에 나타나는 문자열 substr의 위치를 반환한다. occurrence는 양의 정수이어야 한다.

- 예제

다음은 INSTR 함수를 사용하는 예이다.

```
SQL> SELECT INSTR('ABCDEABCDEABCDE', 'CD') FROM DUAL;

INSTR('ABCDEABCDEABCDE','CD')
-----------------------------
                            3

1 row selected.

SQL> SELECT INSTR('ABCDEABCDEABCDE', 'CD', 5, 2) FROM DUAL;

INSTR('ABCDEABCDEABCDE','CD',5,2)
---------------------------------
                               13

1 row selected.
```

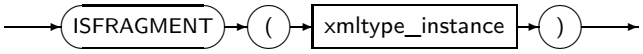
4.2.67. ISFRAGMENT

ISFRAGMENT은 xmltype_instance가 fragment이면 1을 반환하고, well-formed document이면 0을 반환하는 함수이다.

ISFRAGMENT의 세부 내용은 다음과 같다.

- 문법

isfragment



- 구성요소

구성요소	설명
xmltype_instance	질의의 대상이 되는 XML 문서로, XMLType이다.

- 예제

다음은 ISFRAGMENT 함수를 사용하는 예이다.

```
 SQL> select isfragment(XMLTYPE('<a><b>1</b></a>')) from dual;

ISFRAGMENT(XMLTYPE('<A><B>1</B></A>'))
--------------------------------------
                                     0

1 row selected.

SQL> select isfragment(XMLCONCAT(XMLTYPE('<a>1</a>'), XMLTYPE('<b>2</b>')))
from dual;

ISFRAGMENT(XMLCONCAT(XMLTYPE('<A>1</A>'),XMLTYPE('<B>2</B>')))
-------------------------------------------------------------
                                                            1

1 row selected.
```

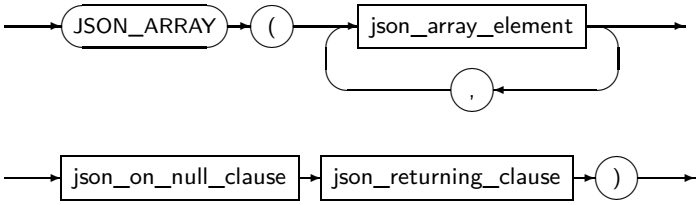
4.2.68. JSON_ARRAY

JSON_ARRAY는 인자로 사용된 표현식을 JSON 값으로 변환하고, 해당 JSON 값들이 포함된 JSON 배열을 반환하는 함수이다.

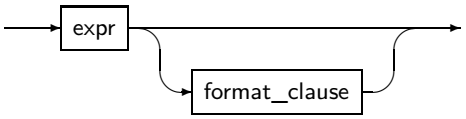
JSON_ARRAY의 세부 내용은 다음과 같다.

- 문법

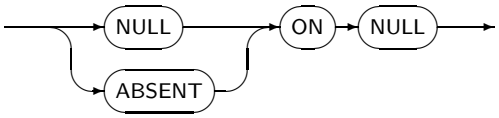
json_array



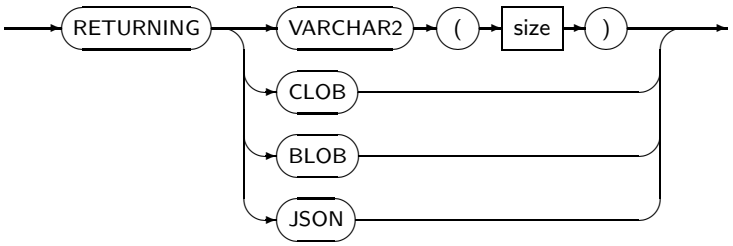
json_array_element



json_on_null_clause



json_returning_clause



- 구성요소

구성요소	설명
json_array_element	expr에는 JSON 객체, JSON 배열, 리터럴 데이터 또는 null 등을 지정할 수 있다. FORMAT JSON을 지정하여 입력 문자열이 JSON임을 나타내고, 인용되지 않은 결과를 반환한다.
json_on_null_clause	expr이 null일 때 함수의 동작을 지정하는 구문이다. 지정하지 않을 경우 기본값은 ABSENT ON NULL 이다.
json_return ing_clause	함수에서 반환된 값의 데이터 형식과 포맷을 지정하는 구문이다. 지정하지 않을 경우 기본값은 VARCHAR2(65532) 이다.

- 예제

다음은 JSON_ARRAY 함수를 사용하는 예이다.

```
SQL> SELECT JSON_ARRAY (
         JSON_OBJECT('PoNum' VALUE 5000),
         JSON_ARRAY(1,2,3),
         100,
         'Korea',
         null
         NULL ON NULL
         ) "JSON Array Example"
     FROM DUAL;

JSON Array Example
--------------------------------------------------------------------------------
[{"PoNum":5000},[1,2,3],100,"Korea",null]

1 row selected.
```

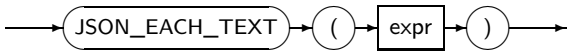
4.2.69. JSON_EACH_TEXT

JSON_EACH_TEXT는 JSON 객체를 JSON KEY/VALUE 쌍의 세트로 확장하는 함수이다. 이 함수는 FROM 절에서만 사용 가능하다.

JSON_EACH_TEXT의 세부 내용은 다음과 같다.

- 문법

json_each_text



- 구성요소

구성요소	설명
expr	expr에는 JSON 객체를 지정할 수 있다.

- 예제

다음은 JSON_EACH_TEXT 함수를 사용하는 예이다.

```
SQL> SET LINESIZE 100
SQL> COLUMN key FORMAT A40
SQL> COLUMN value FORMAT A40

SQL> SELECT * FROM JSON_EACH_TEXT ('{"a":"foo", "b":"bar"}');
```

```
key                                    value
-------------------------------------- --------------------------------------
a                                      foo
b                                      bar

2 row selected.
```

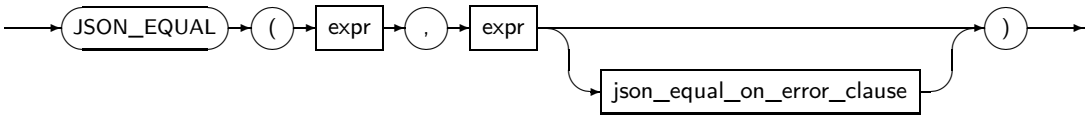
4.2.70. JSON_EQUAL

JSON_EQUAL은 두 JSON 값을 비교하여 같은지 판단하는 함수이다.

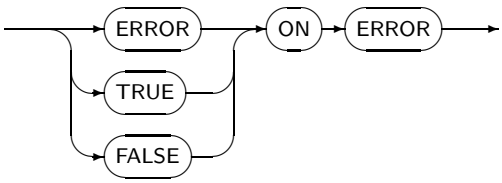
JSON_EQUAL의 세부 내용은 다음과 같다.

- 문법

json_equal



json_equal_on_error_clause



- 구성요소

구성요소	설명
expr	expr에는 JSON 객체, JSON 배열, 리터럴 데이터 또는 null 등을 지정할 수 있다.
json_equal_on_error_clause	오류가 발생할 때 반환하는 값을 지정하는 구문이다. 지정하지 않을 경우 기본값은 FALSE ON NULL 이다.

- 예제

다음은 JSON_EQUAL 함수를 사용하는 예이다.

```
SQL> CREATE TABLE T (
         id NUMBER NOT NULL,
```

```
        data1 VARCHAR2(40),
        data2 VARCHAR2(40)
    );

SQL> INSERT INTO T VALUES (1, '{}', '{}');
SQL> INSERT INTO T VALUES (2, '{"key1":"value1","key2":"value2"}',
                           '{"key1":"value1","key2":"value2"}');
SQL> INSERT INTO T VALUES (3, '{"key1":"value1","key2":"value2"}',
                           '{"key1":"value1", "key2":"value2" }');
SQL> INSERT INTO T VALUES (4, '{"key1":"value1","key2":"value2"}',
                           '{"key2":"value2","key1":"value1"}');
SQL> INSERT INTO T VALUES (5, '{"key1":"value1","key2":"value2"}',
                           '{"key2":"value2","key3":"value3"}');
SQL> INSERT INTO T VALUES (6, '{"key1":"value1","key2":"value2"}',
                           '{"key2":"value2"}');
SQL> COMMIT;

SQL> SET LINESIZE 120
SQL> COLUMN data1 FORMAT A40
SQL> COLUMN data2 FORMAT A40

SQL> SELECT id,
         data1,
         data2
         FROM T
    WHERE JSON_EQUAL(data1, data2);

   ID DATA1                                    DATA2
----- ---------------------------------------- ----------------------------------------
    1 {}                                       {}
    2 {"key1":"value1","key2":"value2"}        {"key1":"value1","key2":"value2"}
    3 {"key1":"value1","key2":"value2"}        {"key1":"value1", "key2":"value2" }
    4 {"key1":"value1","key2":"value2"}        {"key2":"value2","key1":"value1"}

4 rows selected.

SQL> SELECT id,
         data1,
         data2
         FROM T
    WHERE NOT JSON_EQUAL(data1, data2);

   ID DATA1                                    DATA2
----- ---------------------------------------- ----------------------------------------
    5 {"key1":"value1","key2":"value2"}        {"key2":"value2","key3":"value3"}
    6 {"key1":"value1","key2":"value2"}        {"key2":"value2"}
```

```
2 rows selected.
```

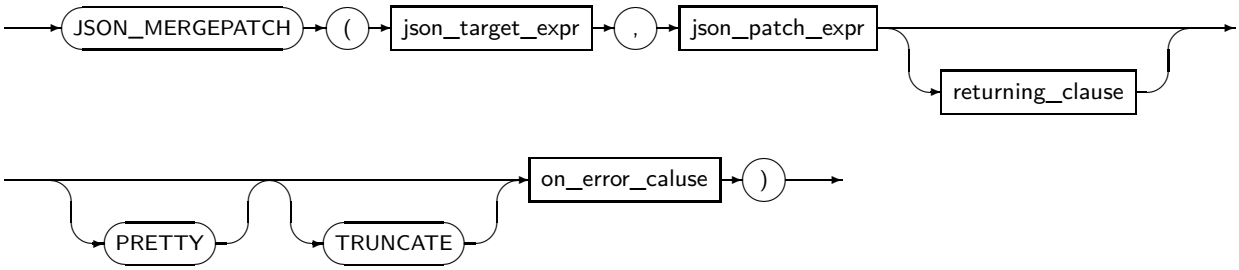
4.2.71. JSON_MERGEPATCH

JSON_MERGEPATCH는 JSON 문서를 수정하는 함수이다. JSON 문서를 수정하기 위해서는 수정 시킬 또 다른 JSON 문서가 필요하다. JSON_MERGEPATCH 함수는 RFC 7396을 따른다.

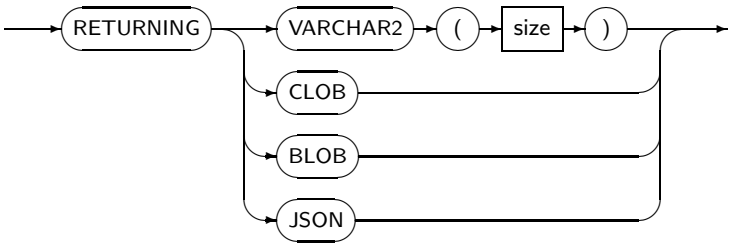
JSON_MERGEPATCH의 세부 내용은 다음과 같다.

- 문법

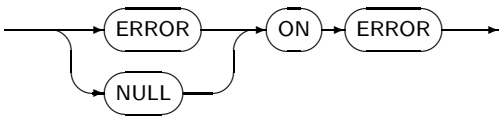
json_mergepatch



json_mergepatch_returning_clause



json_mergepatch_on_error_clause



- 구성요소

구성요소	설명
json_target_expr	수정이 되는 JSON 문서의 표현식이다.
json_patch_expr	수정 시킬 JSON 문서의 표현식이다.
json_returning_clause	함수에서 반환된 값의 데이터 형식과 포맷을 지정하는 구문이다. 지정하지 않을 경우 기본값은 VARCHAR2(65532)이다.

구성요소	설명
PRETTY	개행과 인덴트를 추가하여 결과를 출력한다.
TRUNCATE	VARCHAR2 타입으로 리턴할 때 SIZE보다 긴 경우 SIZE만큼만 출력한다.
json_mergepatch_on_error_clause	오류가 발생할 때 반환하는 값을 지정하는 구문이다. 지정하지 않을 경우 기본값은 NULL ON ERROR이다.

- 예제

  다음은 JSON_MERGEPATCH 함수를 사용하는 예이다.

```
SQL> CREATE TABLE J_TBL (J_COL JSON);
SQL> INSERT INTO J_TBL VALUES ('{"a": {"b" : 1, "c" : 2}}');

SQL> SELECT JSON_MERGEPATCH(J_COL, '{"a": {"b":null, "c" : 3}}' PRETTY) AS JSON
     FROM J_TBL;

JSON
--------------------------------------------------------------------------------
{
  "a": {
    "c": 3
  }
}

1 row selected.
```

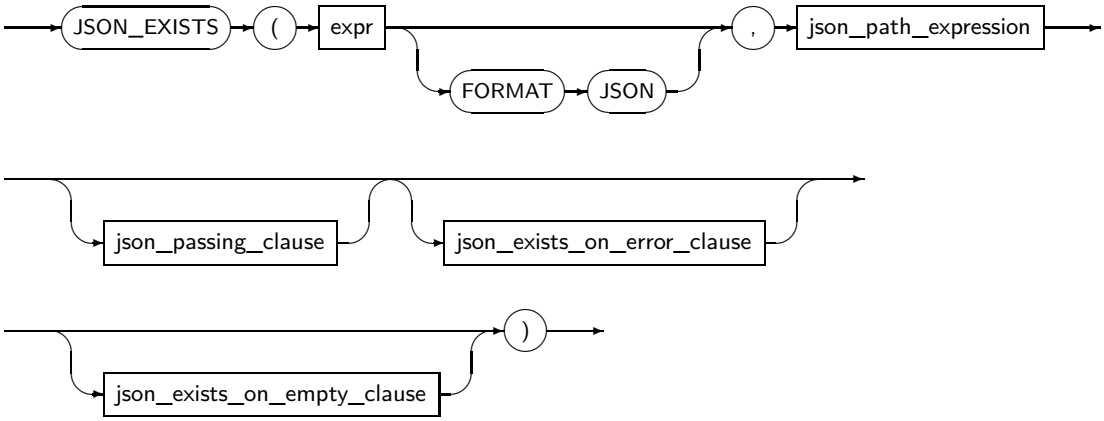
4.2.72. JSON_EXISTS

JSON_EXISTS는 지정된 JSON 값이 JSON data에 존재하는지 확인하는 함수이다.

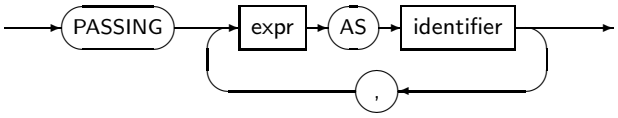
JSON_EXISTS의 세부 내용은 다음과 같다.

- 문법

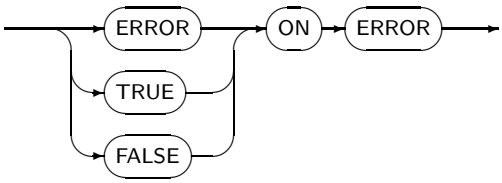
json_exists



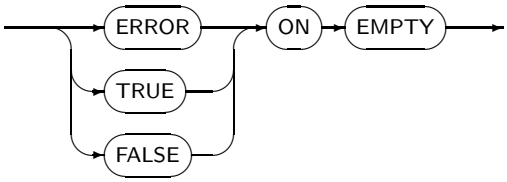
json_passing_clause



json_exists_on_error_clause



json_exists_on_empty_clause



- 구성요소

구성요소	설명
expr	expr은 질의의 대상이 되는 JSON data이다.
json_path_expression	질의할 JSON 경로 표현식이다. 경로 표현식은 리터럴 값이어야 한다.
json_passing_clause	passing 절을 사용하여 경로 표현식에 값을 전달한다.
json_exists_on_error_clause	expr이 올바른 형식의 JSON 데이터가 아닐 때 반환하는 값을 지정하는 구문이다. 지정하지 않을 경우 기본값은 FALSE ON ERROR 이다.

구성요소	설명
json_exists_on_empty_clause	일치하는 항목이 없을 때 반환하는 값을 지정하는 구문이다. 지정하지 않을 경우 json_exists_on_error_clause 구문에 따라 변환 값을 결정한다.

- 예제

다음은 JSON_EXISTS 함수를 사용하는 예이다.

```
SQL> CREATE TABLE J_TBL (J_COL JSON);
SQL> INSERT INTO J_TBL VALUES ('{"a": {"b" : 1, "c" : 2}}');

SQL> SELECT * FROM J_TBL T
     WHERE JSON_EXISTS (T.J_COL, '$.a ? (@.b == 1 && @.c > 0)');

J_COL
--------------------------------------------------------------------------------
{"a": {"b" : 1, "c" : 2}}

1 row selected.
```

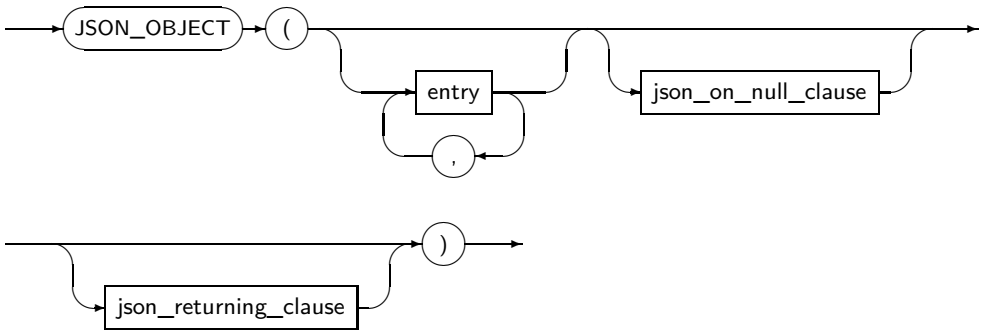
4.2.73. JSON_OBJECT

JSON_OBJECT는 KEY/VALUE 쌍의 시퀀스를 인자로 받아 각 object 멤버가 포함된 JSON 객체를 반환하는 함수이다.

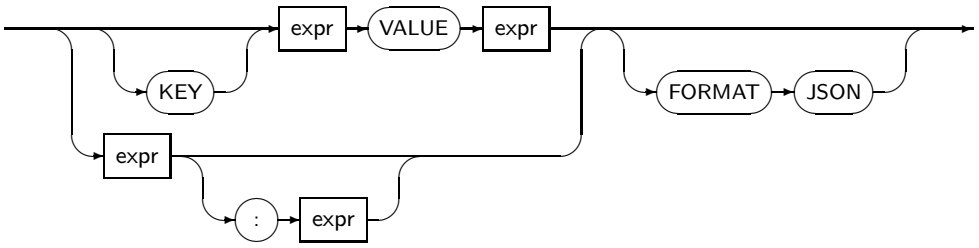
JSON_OBJECT의 세부 내용은 다음과 같다.

- 문법

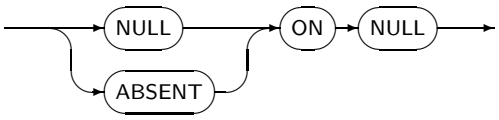
json_object



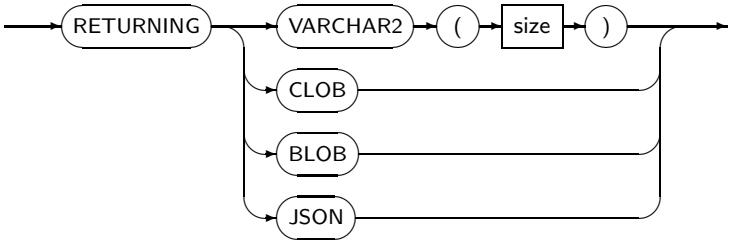
entry



json_on_null_clause



json_returning_clause



- 구성요소

구성요소	설명
entry	entry 절을 사용하여 KEY/VALUE 쌍을 지정한다. FORMAT JSON을 지정하여 입력 문자열이 JSON임을 나타내고, 인용되지 않은 결과를 반환한다.
json_on_null_clause	expr이 null일 때 함수의 동작을 지정하는 구문이다. 지정하지 않을 경우 기본값은 NULL ON NULL 이다.
json_returning_clause	함수에서 반환된 값의 데이터 형식과 포맷을 지정하는 구문이다. 지정하지 않을 경우 기본값은 VARCHAR2(65532) 이다.

- 예제

다음은 JSON_OBJECT 함수를 사용하는 예이다.

```
SQL> SELECT JSON_OBJECT ('name' value '["Foo", "Bar"]' FORMAT JSON)
     as j_obj FROM DUAL;

J_OBJ
```

```
--------------------------------------------------------------------------------
{"name":["Foo", "Bar"]}

1 row selected.

SQL> SELECT JSON_OBJECT ('name' value '["Foo", "Bar"]') as j_obj FROM DUAL;

J_OBJ
--------------------------------------------------------------------------------
{"name":"[\"Foo\", \"Bar\"]"}

1 row selected.
```

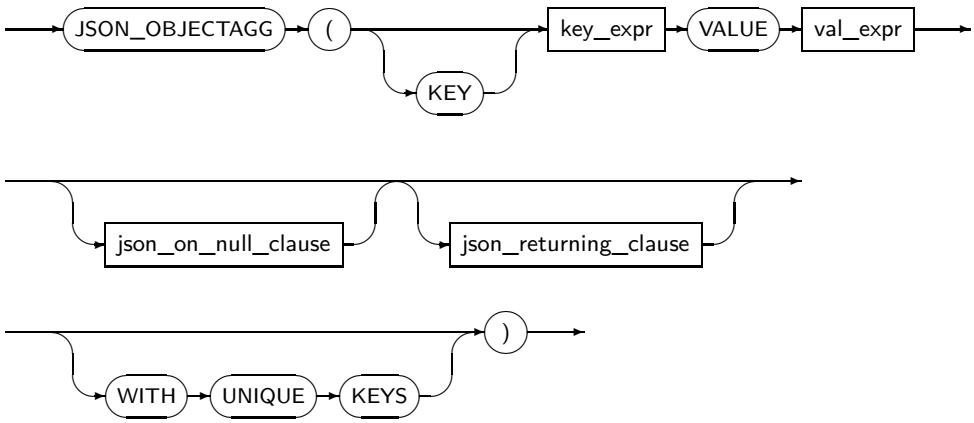
4.2.74. JSON_OBJECTAGG

JSON_OBJECTAGG는 집합 함수이다. KEY/VALUE 쌍의 시퀀스를 인자로 받는다. 이 함수는 KEY-VALUE 쌍에 대해 object를 생성하고 단일 JSON 객체를 반환하는 함수이다.

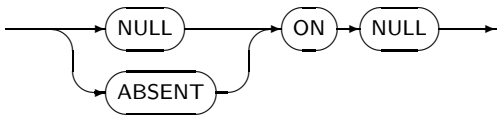
JSON_OBJECTAGG의 세부 내용은 다음과 같다.

- 문법

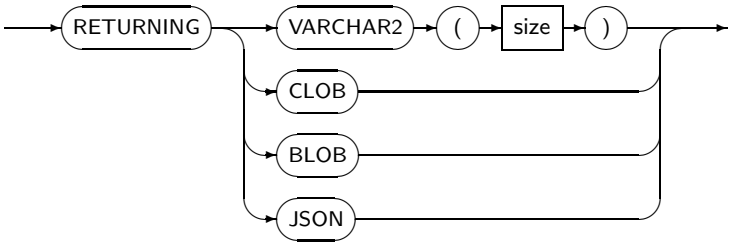
json_objectagg



json_on_null_clause



json_returning_clause



- 구성요소

구성요소	설명
expr	key_expr을 통해 KEY를 value_expr을 통해 VALUE를 지정한다.
json_on_null_clause	expr이 null일 때 함수의 동작을 지정하는 구문이다. 지정하지 않을 경우 기본값은 NULL ON NULL이다.
json_returning_clause	함수에서 반환된 값의 데이터 형식과 포맷을 지정하는 구문이다. 지정하지 않을 경우 기본값은 VARCHAR2(65532)이다.
with unique keys	생성된 JSON 객체에 unique key를 보장하려면 with unique keys를 지정한다.

- 예제

다음은 JSON_OBJECTAGG 함수를 사용하는 예이다.

```
SQL> CREATE TABLE J_TBL (C1 VARCHAR2(100), C2 VARCHAR2(100));
SQL> INSERT INTO J_TBL VALUES ('a', 'b');
SQL> INSERT INTO J_TBL VALUES ('c', 'd');
SQL> INSERT INTO J_TBL VALUES ('c', 'e');

SQL> SELECT JSON_OBJECTAGG(KEY C1 VALUE C2) AS J_OBJAGG FROM J_TBL;

J_OBJAGG
--------------------------------------------------------------------------------
{"a":"b","c":"d","c":"e"}

1 row selected.

SQL> SELECT JSON_OBJECTAGG(KEY C1 VALUE C2 WITH UNIQUE KEYS) AS J_OBJAGG
                                                              FROM J_TBL;
TBR-11075: The JSON object has duplicate keys.
```

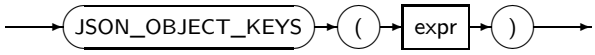
4.2.75. JSON_OBJECT_KEYS

JSON_OBJECT_KEYS는 outermost JSON 객체의 key 세트를 반환하는 함수이다.

JSON_OBJECT_KEYS의 세부 내용은 다음과 같다.

- 문법

json_object_keys



- 구성요소

구성요소	설명
expr	expr에는 JSON 객체를 지정할 수 있다.

- 예제

다음은 JSON_OBJECT_KEYS 함수를 사용하는 예이다.

```
SQL> SELECT * FROM JSON_OBJECT_KEYS('{"key1":"value1",
                                     "key2":{"key3":"value3", "key4":"value4"}}');

json_object_keys
--------------------------------------------------------------------------------
key1
key2

2 rows selected.
```

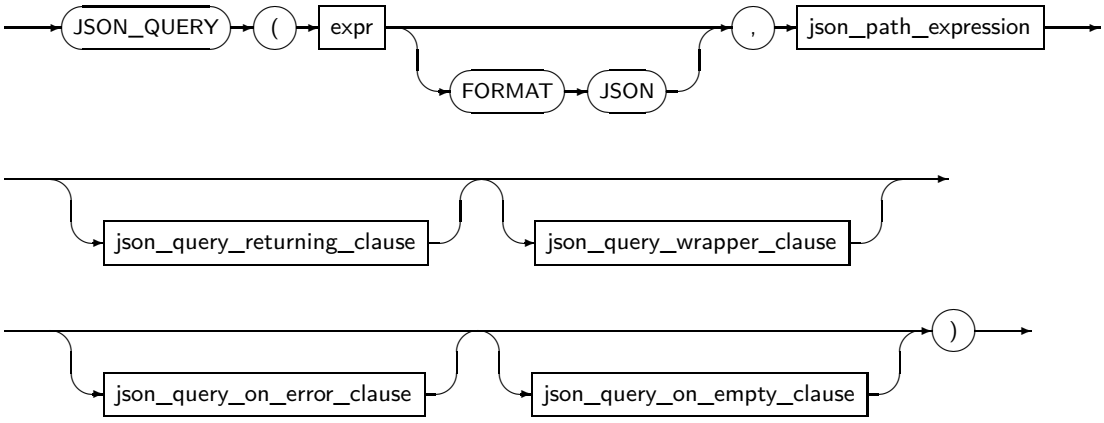
4.2.76. JSON_QUERY

JSON_QUERY는 JSON 데이터에서 경로에 해당하는 하나 이상의 값을 선택하여 반환하는 함수이다.

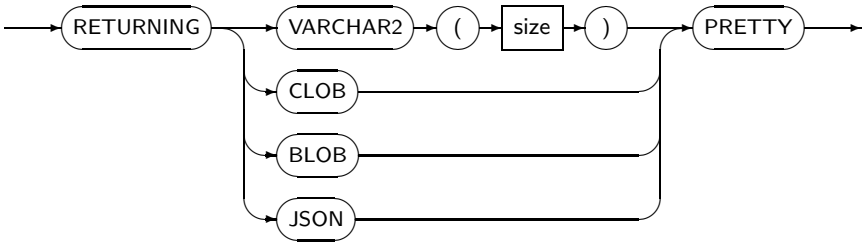
JSON_QUERY의 세부 내용은 다음과 같다.

- 문법

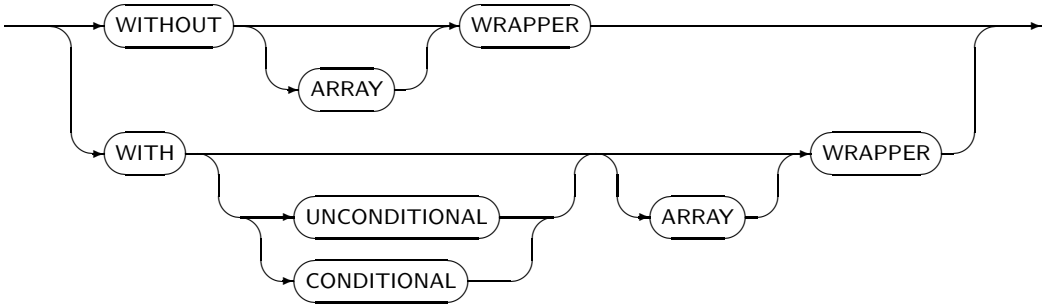
json_query



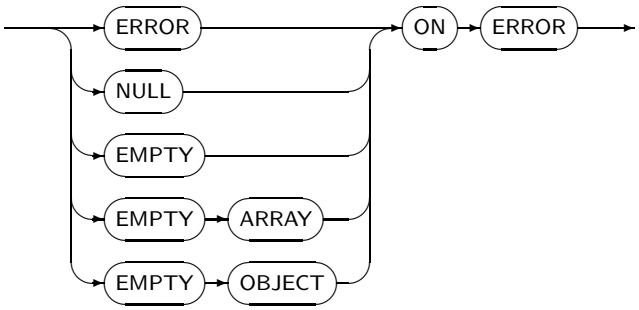
json_query_returning_clause



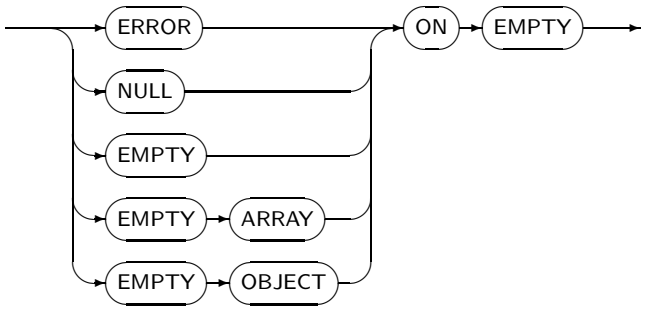
json_query_wrapper_clause



json_query_on_error_clause



json_query_on_empty_clause



- 구성요소

구성요소	설명
expr	질의의 대상이 되는 JSON data이다.
json_path_expression	질의할 JSON 경로 표현식이다. 경로 표현식은 리터럴 값이어야 한다.
json_query_returning_clause	함수에서 반환된 값의 데이터 형식과 포맷을 지정하는 구문이다. 지정하지 않을 경우 기본값은 VARCHAR2(65532)이다. PRETTY를 설정하면 개행 문자를 삽입하고 들여쓰기하여 반환 문자열을 이쁘게 출력한다.
json_query_wrapper_clause	경로 표현식에 일치하는 결과를 대괄호([])로 묶을지 지정하는 구문이다. 지정하지 않을 경우 기본값은 WITHOUT WRAPPER 이다.
json_query_on_error_clause	오류가 발생할 때 반환하는 값을 지정하는 구문이다. 지정하지 않을 경우 기본값은 NULL ON ERROR 이다.
json_query_on_empty_clause	일치하는 결과가 없을 때 반환하는 값을 지정하는 구문이다. 지정하지 않을 경우 json_query_on_error_clause 구문에 따라 반환 값을 결정한다.

- 예제

  다음은 JSON_QUERY 함수를 사용하는 예이다.

```
SQL> SELECT JSON_QUERY('{"a":10, "b":20, "c":30}', '$') AS value FROM DUAL;

VALUE
--------------------------------------------------------------------------------
{"a":10,"b":20,"c":30}

1 row selected.
```

```
SQL> SELECT JSON_QUERY('{"a":10, "b":20, "c":30}', '$.a' WITH WRAPPER)
     AS value FROM DUAL;

VALUE
--------------------------------------------------------------------------------
[10]

1 row selected.

SQL> SELECT JSON_QUERY('{"a":100, "b":200, "c":30}', '$.*' WITH WRAPPER)
     AS value FROM DUAL;

VALUE
--------------------------------------------------------------------------------
[100, 200, 30]

1 row selected.

SQL> SELECT JSON_QUERY('[{"a":10},{"b":20},{"c":300}]', '$[0]'
     WITH CONDITIONAL WRAPPER) AS value
     FROM DUAL;

VALUE
--------------------------------------------------------------------------------
{"a":10}

1 row selected.

SQL> SELECT JSON_QUERY('[{"a":10},{"b":20},{"c":30}]', '$[*]'
          WITH CONDITIONAL WRAPPER) AS value
     FROM DUAL;

VALUE
--------------------------------------------------------------------------------
[{"a":10}, {"b":20}, {"c":30}]

1 row selected.
```

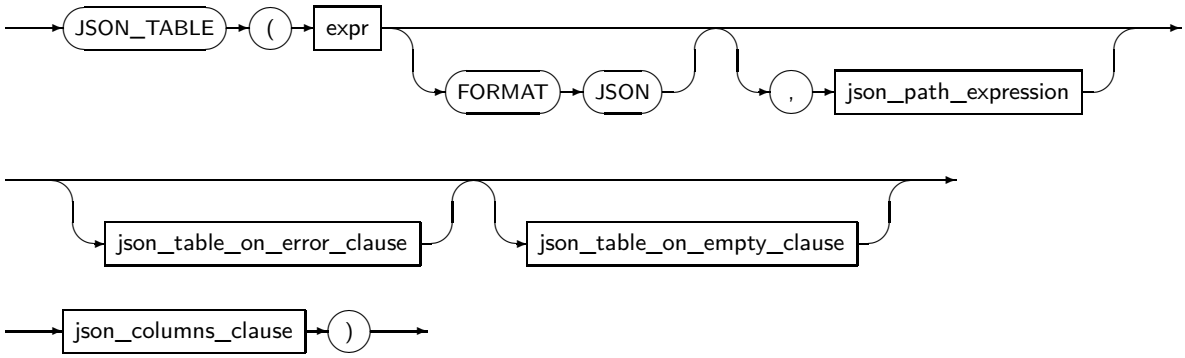
4.2.77. JSON_TABLE

JSON_TABLE은 JSON 데이터의 관계형 뷰를 생성하는 함수이다. 이 함수는 FROM 절에서만 사용 가능하다.

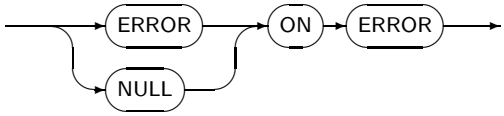
JSON_TABLE의 세부 내용은 다음과 같다.

- 문법

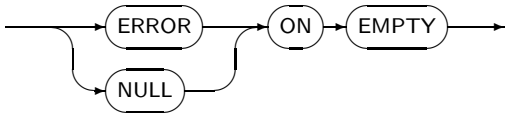
json_table



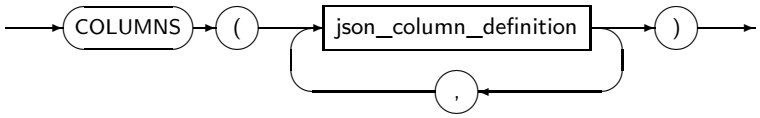
json_table_on_error_clause



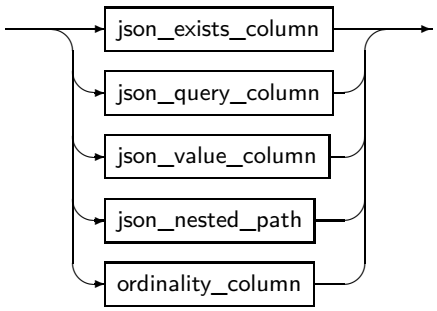
json_table_on_empty_clause



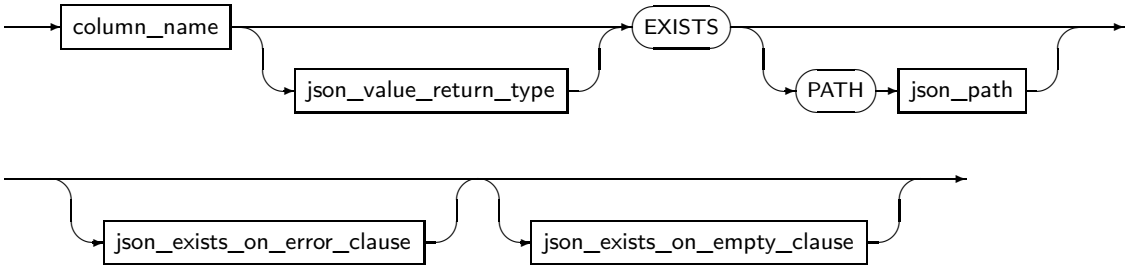
json_columns_clause



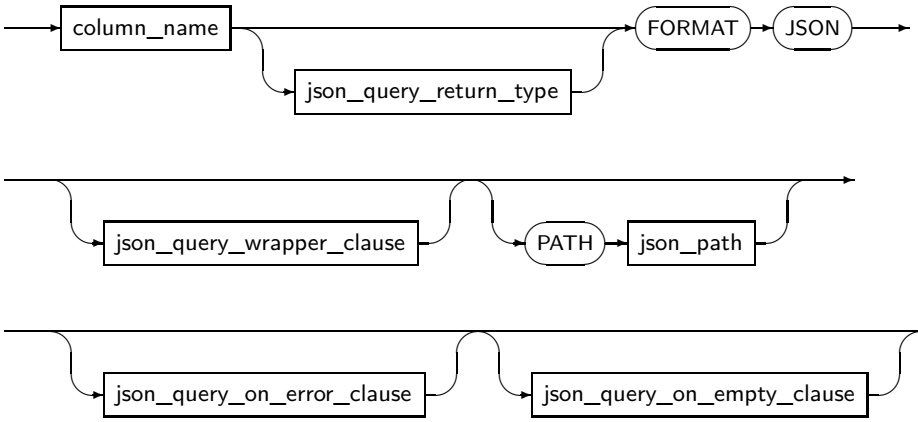
json_column_definition



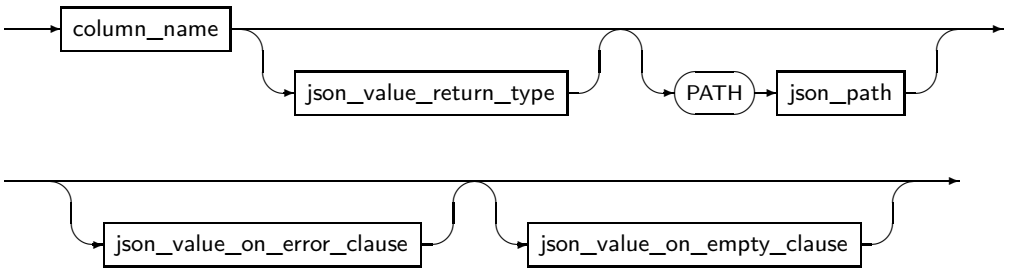
json_exists_column



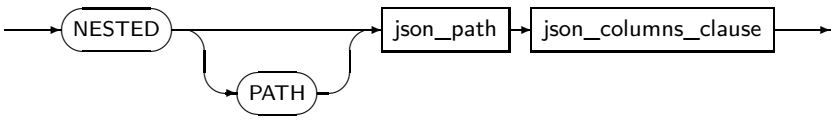
json_query_column



json_value_column



json_nested_path



ordinality_column

- 구성요소

구성요소	설명
expr	질의의 대상이 되는 JSON data이다.

구성요소	설명
json_path_expression	질의할 JSON 경로 표현식이다. 경로 표현식은 리터럴 값이어야 한다.
json_table_on_error_clause	오류가 발생할 때 반환하는 값을 지정하는 구문이다.
json_table_on_empty_clause	일치하는 결과가 없을 때 반환하는 값을 지정하는 구문이다. 지정하지 않을 경우 json_table_on_error_clause 구문에 따라 반환 값을 결정한다.
json_columns_clause	COLUMNS 절을 사용하여 JSON_TABLE 함수에서 반환된 가상 관계형 테이블의 열을 정의한다.

- 예제

  다음은 JSON_TABLE 함수를 사용하는 예이다.

```
SQL> SELECT *
     FROM JSON_TABLE('
     {"A_KEY" : "A_VALUE",
     "B_KEY" : [
                {"C_KEY": "C_VALUE",
                 "D_KEY": [1]},
                {"C_KEY": "C_VALUE_1",
                 "D_KEY": ["2", "3"]}
               ]
     }
     ', '$' COLUMNS (
            "A" VARCHAR2(10) PATH '$.A_KEY',
            NESTED PATH '$.B_KEY[*]' COLUMNS (
                "C" VARCHAR2(10) PATH '$.C_KEY',
                NESTED PATH '$.D_KEY[*]' COLUMNS (
                    "D_ARR" NUMBER PATH '$'
                )
            )
         )
     );

A          C               D_ARR
---------- ---------- ----------
A_VALUE    C_VALUE             1
A_VALUE    C_VALUE_1           2
A_VALUE    C_VALUE_1           3

3 rows selected.
```

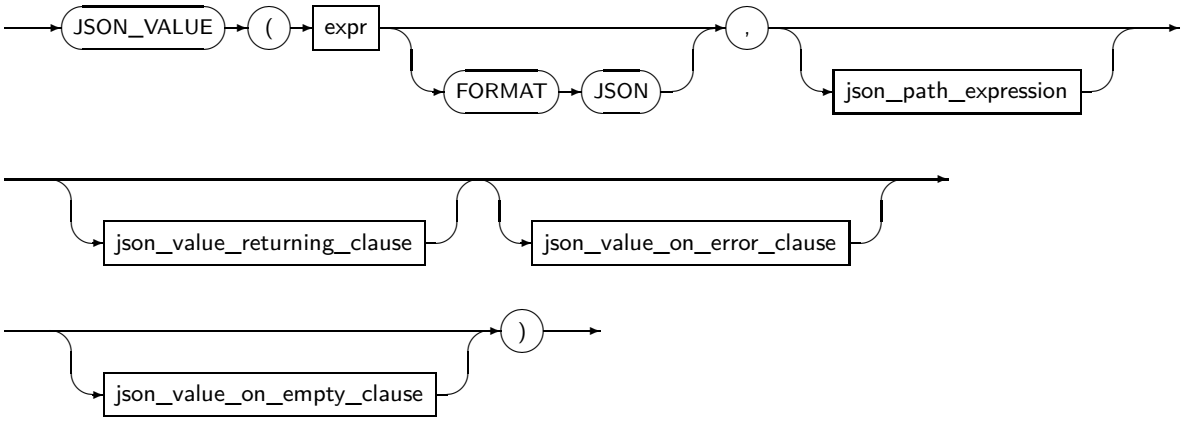
4.2.78. JSON_VALUE

JSON_VALUE는 JSON data에서 경로에 해당하는 scalar JSON 값을 반환하는 함수이다.

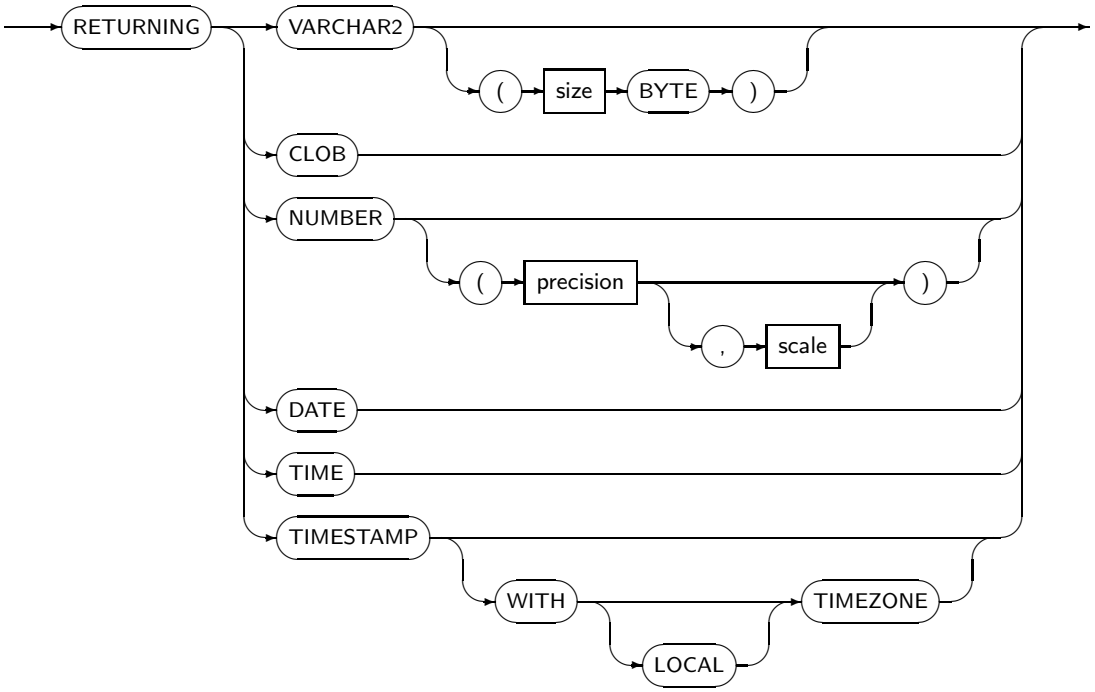
JSON_VALUE의 세부 내용은 다음과 같다.

- 문법

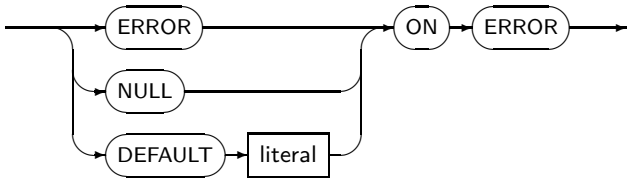
json_value



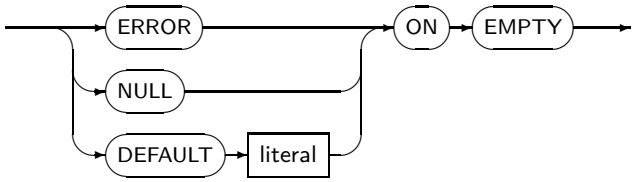
json_value_returning_clause



json_value_on_error_clause



json_value_on_empty_clause



- 구성요소

구성요소	설명
expr	질의의 대상이 되는 JSON data이다.
json_path_expression	질의할 JSON 경로 표현식이다. 경로 표현식은 리터럴 값이어야 한다.
json_value_returning_clause	함수에서 반환된 값의 데이터 형식과 포맷을 지정하는 구문이다. 지정하지 않을 경우 기본값은 VARCHAR2(4000)이다.
json_value_on_error_clause	오류가 발생할 때 반환하는 값을 지정하는 구문이다. 지정하지 않을 경우 기본값은 NULL ON ERROR 이다.
json_value_on_empty_clause	일치하는 결과가 없을 때 반환하는 값을 지정하는 구문이다. 지정하지 않을 경우 json_value_on_error_clause 구문에 따라 반환 값을 결정한다.

- 예제

다음은 JSON_VALUE 함수를 사용하는 예이다.

```
SQL> CREATE TABLE T (J_COL JSON);
SQL> INSERT INTO T VALUES ('
     {
          "json_object" : {
               "json_string" : "string"}
     }
     ');

1 row inserted.
```

```
SQL> SELECT JSON_VALUE (J_COL FORMAT JSON,  '$.json_object.json_string'
     RETURNING VARCHAR2(10)) AS JSON_VAL FROM T;
JSON_VAL
--------------------------------------------------------------------------------
string

1 row selected.

SQL> SELECT JSON_VALUE (J_COL FORMAT JSON,  '$.json_object.json_string'
     RETURNING VARCHAR2(1)) AS JSON_VAL FROM T;

JSON_VAL
--------



1 row selected.

SQL> SELECT JSON_VALUE (J_COL FORMAT JSON,  '$.json_object.json_string'
    RETURNING VARCHAR2(1) ERROR ON ERROR) FROM T;
TBR-11070: The result value is too large (Maximum value: 1)
```

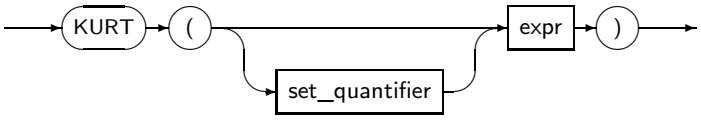
4.2.79. KURT

KURT는 expr의 첨도를 반환하는 함수이다. 파라미터의 수치 데이터 타입 또는 변환된 수치 데이터 타입을 반환한다.

KURT의 세부 내용은 다음과 같다.

- 문법

kurt



- 구성요소

구성요소	설명
set_quantifier	질의 결과에 중복된 로우의 허용, 비허용 여부를 지정한다. DISTINCT, UNIQUE, ALL을 지정할 수 있다. – DISTINCT, UNIQUE : 중복된 로우를 제거한다. – ALL : 모든 로우를 선택한다. (기본값)

구성요소	설명
expr	모든 수치 데이터 타입 또는 수치 데이터 타입은 아니지만 묵시적으로 수치 데이터 타입으로 변환할 수 있는 임의의 연산식이다.

- 예제

  다음은 KURT 함수를 사용하는 예이다.

```
SQL> SELECT KURT(SAL) FROM EMP;

 KURT(SAL)
----------
1.31945327

1 row selected.
```

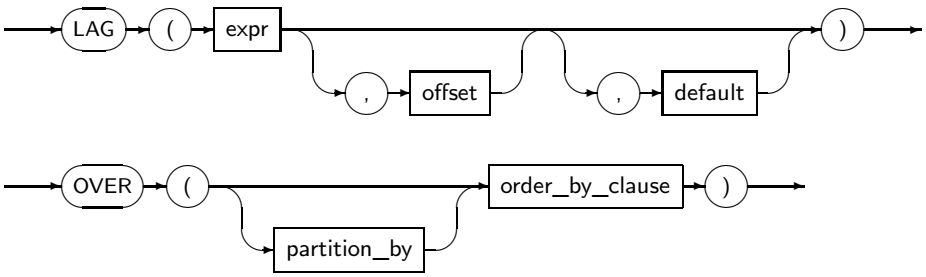
4.2.80. LAG

LAG는 자기 자신과 조인하지 않고도 한 테이블에서 여러 개의 로우를 동시에 볼 수 있는 분석 함수이다. LAG는 명시된 수만큼 현재 로우에서 앞서 있는 로우에 접근을 제공한다.

LAG의 세부 내용은 다음과 같다.

- 문법

lag



- 구성요소

구성요소	설명
expr	expr에 LAG를 포함한 다른 분석 함수를 명시할 수 없다. 즉, 분석 함수를 중첩해서 사용할 수는 없다.
offset	접근 횟수를 offset에 명시한다. offset을 명시하지 않으면 1로 간주된다.
default	offset이 윈도우의 범위를 초과하면 default에서 지정한 값이 반환된다. default를 명시하지 않으면 NULL이 반환된다.

구성요소	설명
partition_by	현재 질의 블록의 결과 집합을 expr을 기준으로 분할한다. 자세한 내용은 "4.1.3. 분석 함수"의 partition_by를 참고한다.
order_by_clause	분할된 하나의 파티션 내에서 로우를 어떻게 정렬할지를 명시한다. 자세한 내용은 "4.1.3. 분석 함수"의 order_by_clause를 참고한다.

- 예제

다음은 LAG 함수를 사용하는 예이다.

```
SQL> SELECT NAME, DEPTID, SALARY, LAG (SALARY, 2, 0)
       OVER (PARTITION BY DEPTID ORDER BY SALARY) PSAL
     FROM EMP;

NAME                      DEPTID     SALARY       PSAL
-------------------- ---------- ---------- ----------
Paul                           1       3000          0
Angela                         1       3000          0
Nick                           1       3200       3000
Scott                          1       4000       3000
James                          1       4000       3200
John                           1       4500       4000
Joe                            2       4000          0
Brad                           2       4200          0
Daniel                         2       5000       4000
Tom                            2       5000       4200
Kathy                          2       5000       5000
Bree                           2       6000       5000

12 rows selected.
```

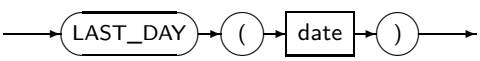
4.2.81. LAST_DAY

LAST_DAY는 파라미터에 명시된 날짜를 포함하는 연월의 마지막 날로 변환된 날짜를 구하는 함수이다.

LAST_DAY의 세부 내용은 다음과 같다.

- 문법

last_day



- 구성요소

구성요소	설명
date	날짜 값을 반환하는 임의의 연산식이다.

- 예제

  다음은 LAST_DAY 함수를 사용하는 예이다.

```
SQL> SELECT LAST_DAY('2005/06/22') FROM DUAL;

LAST_DAY('2005/06/22')
----------------------------------------------------------------
2005-06-30

1 row selected.
```

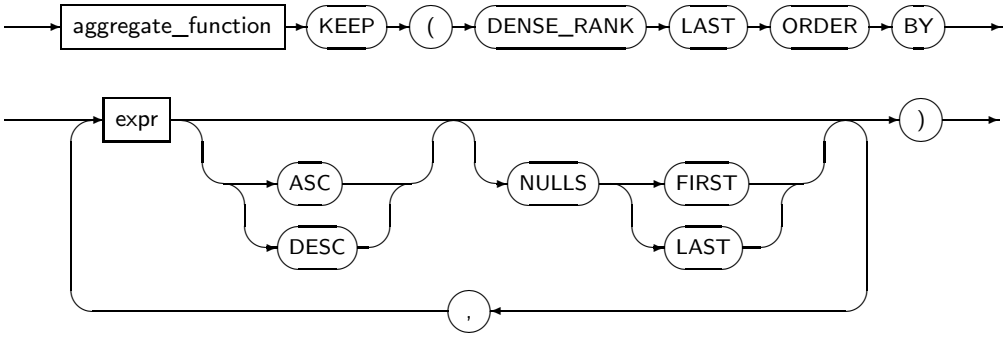
4.2.82. LAST

LAST는 정렬된 로우에서 마지막에 해당하는 로우를 뽑아내어 명시된 집단함수를 적용한 결과를 반환한다. 현재 분석 함수는 지원하지 않는다.

LAST 세부 내용은 다음과 같다.

- 문법

last



- 구성요소

구성요소	설명
aggregate_function	사용가능한 집단 함수는 AVG, COUNT, MIN, MAX, SUM, STADDEV 그리고 VARIANCE이다.
expr	임의의 연산식이다.

- 예제

다음은 LAST 함수를 사용하는 예이다.

```
SQL> SELECT DEPTNO, MIN(HIREDATE) KEEP (DENSE_RANK LAST ORDER BY SAL)
MIN_HIREDATE, MAX(HIREDATE) KEEP (DENSE_RANK LAST ORDER BY SAL) MAX_HIREDATE
FROM EMP GROUP BY DEPTNO;

    DEPTNO MIN_HIREDATE                    MAX_HIREDATE
---------- ------------------------------ ------------------------------
        10 1981/11/17                     1981/11/17
        20 1981/12/03                     1987/04/19
        30 1981/05/01                     1981/05/01

3 rows selected.
```

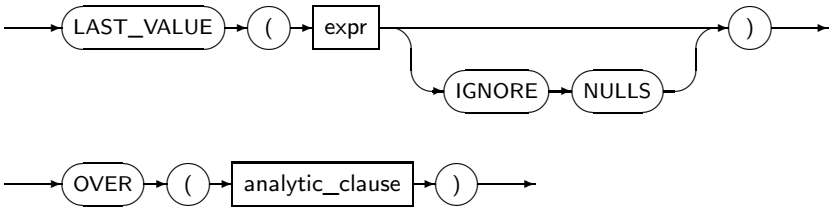
4.2.83. LAST_VALUE

LAST_VALUE는 정렬된 로우에서 마지막 값을 반환하는 분석함수이다. IGNORE NULLS를 명시하면 NULL이 아닌 첫 번째 값을 반환한다. 만약 모든 값이 NULL이라면 NULL을 반환한다.

LAST_VALUE 세부 내용은 다음과 같다.

- 문법

last_value



- 구성요소

구성요소	설명
expr	임의의 연산식이다.

- 예제

다음은 LAST_VALUE 함수를 사용하는 예이다.

```
SELECT DEPTNO, ENAME, LAST_VALUE(ENAME) OVER (PARTITION BY DEPTNO ORDER BY
HIREDATE ASC ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS NEW_EMP
FROM EMP;

    DEPTNO ENAME      NEW_EMP
```

```
---------- ---------- ----------
        10 CLARK      MILLER
        10 KING       MILLER
        10 MILLER     MILLER
        20 SMITH      ADAMS
        20 JONES      ADAMS
        20 FORD       ADAMS
        20 SCOTT      ADAMS
        20 ADAMS      ADAMS
        30 ALLEN      JAMES
        30 WARD       JAMES
        30 BLAKE      JAMES
        30 TURNER     JAMES
        30 MARTIN     JAMES
        30 JAMES      JAMES

14 rows selected.
```

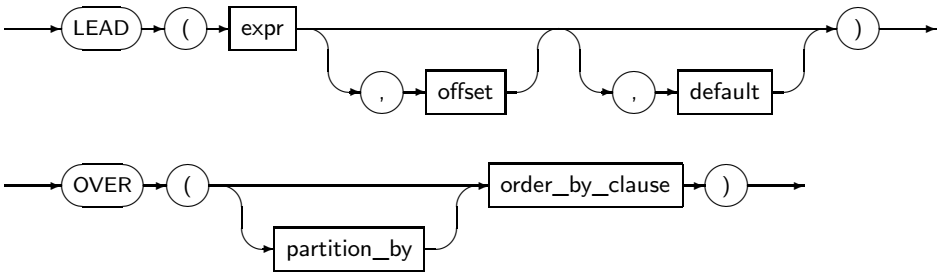
4.2.84. LEAD

LEAD는 자기 자신과 조인하지 않고도 한 테이블에서 여러 개의 로우를 동시에 볼 수 있는 분석 함수이다. LEAD는 명시된 수만큼 현재 로우에서 뒤에 나오는 로우에 접근을 제공한다.

LEAD의 세부 내용은 다음과 같다.

- 문법

lead



- 구성요소

구성요소	설명
expr	expr에 LEAD를 포함한 다른 분석 함수를 명시할 수 없다. 즉, 분석 함수를 중첩해서 사용할 수는 없다.
offset	접근 횟수를 offset에 명시한다. offset을 명시하지 않으면 1로 간주된다.
default	offset이 윈도우의 범위를 초과하면 default에서 지정한 값이 반환된다.

구성요소	설명
	default를 명시하지 않으면 NULL이 반환된다.
partition_by	현재 질의 블록의 결과 집합을 expr을 기준으로 분할한다. 자세한 내용은 "4.1.3. 분석 함수"의 partition_by를 참고한다.
order_by_clause	분할된 하나의 파티션 내에서 로우를 어떻게 정렬할지를 명시한다. 자세한 내용은 "4.1.3. 분석 함수"의 order_by_clause를 참고한다.

- 예제

다음은 LEAD 함수를 사용하는 예이다.

```
SQL> SELECT NAME, DEPTID, SALARY, LEAD (SALARY, 2, 0)
       OVER (PARTITION BY DEPTID ORDER BY SALARY) PSAL
     FROM EMP;

NAME                    DEPTID     SALARY       PSAL
-------------------- ---------- ---------- ----------
Paul                         1       3000       3200
Angela                       1       3000       4000
Nick                         1       3200       4000
Scott                        1       4000       4500
James                        1       4000          0
John                         1       4500          0
Joe                          2       4000       5000
Brad                         2       4200       5000
Daniel                       2       5000       5000
Tom                          2       5000       6000
Kathy                        2       5000          0
Bree                         2       6000          0

12 rows selected.
```

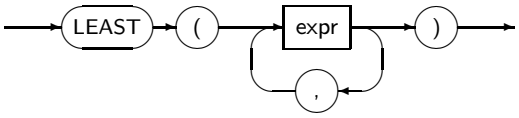
4.2.85. LEAST

LEAST는 파라미터 중 가장 작은 값을 가지는 파라미터를 구하는 함수이다.

LEAST의 세부 내용은 다음과 같다.

- 문법

least



- 구성요소

구성요소	설명
expr	임의의 연산식이다. 첫 파라미터 이후의 모든 파라미터는 첫 파라미터의 타입으로 암묵적으로 변환된 후 비교된다. expr 중 하나의 값이 NULL이면 함수는 NULL을 반환한다.

- 예제

다음은 LEAST 함수를 사용하는 예이다.

```
SQL> SELECT LEAST(1, 3, 2) FROM DUAL;

LEAST(1,3,2)
------------
           1

1 row selected.
```

4.2.86. LENGTH

LENGTH는 주어진 파라미터의 문자열의 길이를 반환하는 함수이다.

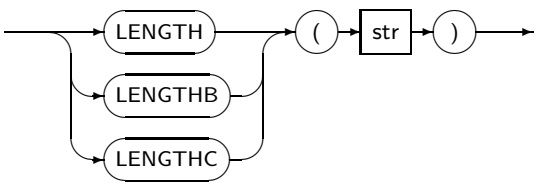
LENGTHB는 문자열의 길이를 문자 대신 byte 단위로 계산한다.

LENGTHC는 문자열의 길이를 유니코드 완전한 문자로 계산한다.

LENGTH의 세부 내용은 다음과 같다.

- 문법

length



- 구성요소

구성요소	설명
str	문자열을 반환하는 임의의 연산식이다.

- 예제

  다음은 LENGTH 함수를 사용하는 예이다.

```
SQL> SELECT LENGTH('ABCDEFG') FROM DUAL;

LENGTH('ABCDEFG')
-----------------
                7

1 row selected.
```

4.2.87. LISTAGG

LISTAGG는 "4.2.4. AGGR_CONCAT"과 동일한 기능을 하는 함수이다. Oracle과의 호환성을 위해 추가되었다.

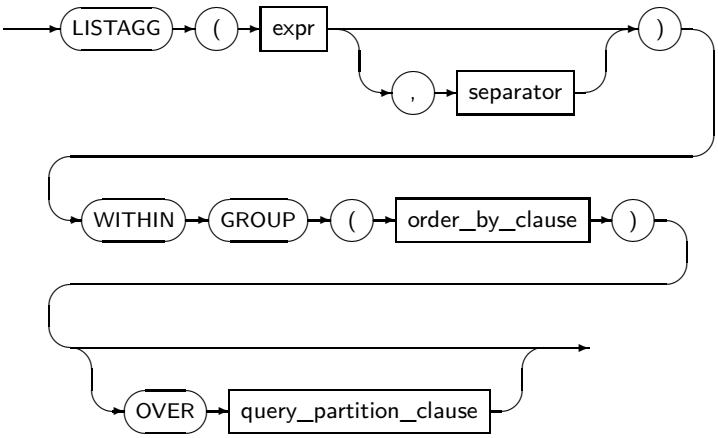
AGGR_CONCAT과는 다음과 같은 사용법에 차이가 있다.

- set_quantifier 를 사용할 수 없다.
- separator를 생략할 수 있고 기본값은 NULL이다.
- order_by_clause는 WITHIN GROUP 이후에 명시하고 생략할 수 없다.

LISTAGG의 세부 내용은 다음과 같다.

- 문법

listagg



- 구성요소

구성요소	설명
expr	문자열이나 문자열로 변환될 수 있는 임의의 연산식이다.
separator	expr과 접합될 구분자를 나타내는 문자 리터럴이다.
order_by_clause	접합할 문자열을 어떻게 정렬할지를 명시한다. 자세한 내용은 “4.1.3. 분석 함수”의 order_by_clause를 참고한다.
query_partition_clause	쿼리 결과에 대해 어떻게 그룹화 할지를 명시한다.

- 예제

다음은 LISTAGG 함수를 사용하는 예이다.

```
SQL> SELECT LISTAGG(ENAME, ',') WITHin GROUP (ORDER BY EMPNO) AS "EMPLOYEE"
   2 FROM EMP
   3 GROUP BY DEPTNO;

EMPLOYEE
--------------------------------------------------------------------------------
CLARK,KING,MILLER
SMITH,JONES,SCOTT,ADAMS,FORD
ALLEN,WARD,MARTIN,BLAKE,TURNER,JAMES

3 rows selected.
```

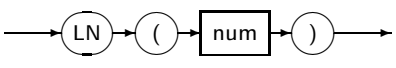
4.2.88. LN

LN은 주어진 파라미터의 자연 로그 값을 구하는 함수이다.

LN의 세부 내용은 다음과 같다.

- 문법

ln



- 구성요소

구성요소	설명
num	수치 값을 반환하는 임의의 연산식이며, 0보다 큰 실수값을 가져야 한다.

구성요소	설명
	num은 숫자형 타입이거나 숫자형 타입으로 변환될 수 있는 타입이어야 한다. num의 타입이 BINARY_FLOAT일 경우엔 BINARY_DOUBLE 타입을 반환하고, 그렇지 않은 경우엔 num과 동일한 타입을 반환한다.

- 예제

다음은 LN 함수를 사용하는 예이다.

```
SQL> SELECT TO_CHAR(LN(2.7182818284),'99') FROM DUAL;

TO_CHAR(LN(2.7182818284),'99')
------------------------------
  1

1 row selected.
```

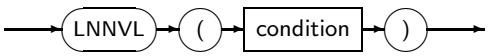
4.2.89. LNNVL

LNNVL은 condition의 계산 값이 FALSE이거나 UNKNOWN이면 TRUE를 반환하고, TRUE이면 FALSE를 반환하는 함수이다. 일반적으로 사용자가 직접 이 함수를 사용할 일은 거의 없지만, 쿼리를 최적화하는 과정에서 사용할 수 있다.

LNNVL의 세부 내용은 다음과 같다.

- 문법

lnnvl



- 구성요소

구성요소	설명
condition	임의의 조건식이다. 조건식에 관한 자세한 내용은 “3.4. 조건식”을 참고한다.

- 예제

다음은 LNNVL 함수를 사용하는 예이다.

```
SQL> SELECT 1 FROM DUAL WHERE LNNVL (1 = 2);

         1
----------
```

```
         1

1 row selected.
```

4.2.90. LOCALTIMESTAMP

LOCALTIMESTAMP는 현재 날짜 및 시간을 출력하는 함수이다. 이 함수의 반환값은 TIMESTAMP 타입이다.

LOCALTIMESTAMP의 세부 내용은 다음과 같다.

- 문법

  localtimestamp

- 예제

  다음은 LOCALTIMESTAMP 함수를 사용하는 예이다.

```
SQL> SELECT LOCALTIMESTAMP FROM DUAL;

LOCALTIMESTAMP
-----------------------------------
2011/04/14 16:55:23.375613

1 row selected.
```

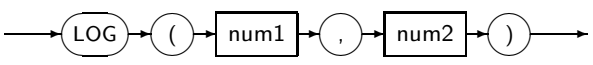
4.2.91. LOG

LOG는 밑을 num1으로 하는 num2의 로그 값을 반환하는 함수이다.

LOG의 세부 내용은 다음과 같다.

- 문법

  log

- 구성요소

구성요소	설명
num1, num2	0보다 큰 실수 값을 반환하는 임의의 연산식이며, num2는 0이나 1이면 안 된다. num1이나 num2는 숫자형 타입이거나 숫자형 타입으로 변환될 수 있는 타입이어야 한다. num1이나 num2의 타입이 BINARY_FLOAT이나 BINARY_DOUBLE일 경우엔 BINARY_DOUBLE 타입을 반환하고, 그렇지 않은 경우엔 NUMBER 타입을 반환한다.

- 예제

다음은 LOG 함수를 사용하는 예이다.

```
SQL> SELECT LOG(2, 8) FROM DUAL;

  LOG(2,8)
----------
         3

1 row selected.
```

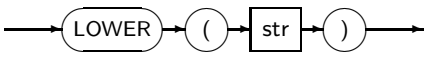
4.2.92. LOWER

LOWER는 문자열 str 내의 모든 영문자를 소문자로 변환하여 반환하는 함수이다.

LOWER의 세부 내용은 다음과 같다.

- 문법

lower



- 구성요소

구성요소	설명
str	문자열을 반환하는 임의의 연산식이다.

- 예제

다음은 LOWER 함수를 사용하는 예이다.

```
SQL> SELECT LOWER('ABCDEFG123') FROM DUAL;

LOWER('ABCDEFG123')
-------------------
```

```
abcdefg123

1 row selected.
```

4.2.93. LPAD

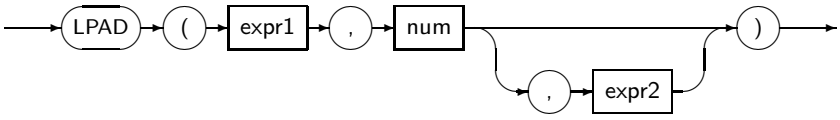
LPAD는 expr2를 expr1의 왼쪽에 반복적으로 붙인 길이 num의 문자열을 구하는 함수이다.

대부분의 문자 집합에서는 반환되는 문자열의 문자 수와 길이가 동일하지만, 한글과 같은 멀티 바이트 문자 집합의 경우에는 두 값이 다를 수 있다.

LPAD의 세부 내용은 다음과 같다.

- 문법

lpad



- 구성요소

구성요소	설명
expr1	문자열 또는 CLOB 타입, BLOB 타입을 반환하는 임의의 연산식이다. expr1의 길이가 num 보다 큰 경우 expr1에서 왼쪽부터 num만큼의 문자열을 반환한다.
expr2	문자열 또는 CLOB 타입, BLOB 타입을 반환하는 임의의 연산식이다. expr2가 명시되지 않은 경우 공백 문자가 사용된다.
num	num은 수치 값을 반환하는 임의의 연산식이다. num은 터미널에 출력되는 길이를 의미한다.

- 예제

다음은 LPAD 함수를 사용하는 예이다.

```
SQL> SELECT LPAD('LPAD', 10, '-=') FROM DUAL;

LPAD('LPAD',10,'-=')
--------------------
-=-=-=LPAD
```

```
1 row selected.
```

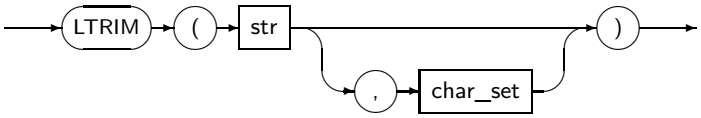
4.2.94. LTRIM

LTRIM은 문자열 str의 왼쪽으로부터 문자열 char_set 내에 포함된 모든 문자를 제거하는 함수이다.

LTRIM의 세부 내용은 다음과 같다.

- 문법

ltrim



- 구성요소

구성요소	설명
str	문자열을 반환하는 임의의 연산식이다.
char_set	문자열을 반환하는 임의의 연산식이다. 만약 파라미터 char_set의 값이 없으면, 기본값으로 공백 문자 하나를 갖는다.

- 예제

다음은 LTRIM 함수를 사용하는 예이다.

```
SQL> SELECT LTRIM(' ABCDE') FROM DUAL;

LTRIM('ABCDE')
--------------
ABCDE

1 row selected.

SQL> SELECT LTRIM('XYXABCDEXYX', 'XY') FROM DUAL;

LTRIM('XYXABCDEXYX','XY')
-------------------------
ABCDEXYX

1 row selected.
```

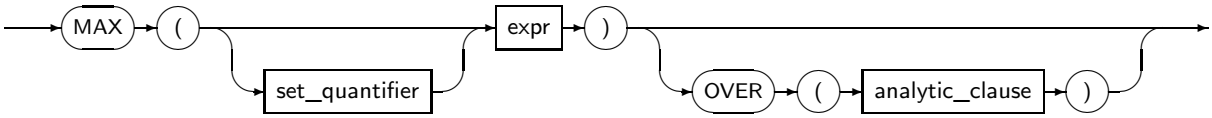
4.2.95. MAX

MAX는 그룹 내의 모든 로우에 대한 expr 값 중의 최댓값을 구하는 함수이다. 분석 함수로도 사용할 수 있다. 이 함수를 분석 함수로 사용할 때 DISTINCT 예약어를 명시하면 analytic_clause에서 partition_by만 명시할 수 있다. order_by_clause는 명시할 수 없다.

MAX의 세부 내용은 다음과 같다.

- 문법

max



- 구성요소

구성요소	설명
set_quantifier	질의 결과에 중복된 로우의 허용, 비허용 여부를 지정한다. DISTINCT, UNIQUE, ALL을 지정할 수 있다. – DISTINCT, UNIQUE : 중복된 로우를 제거한다. – ALL : 모든 로우를 선택한다. (기본값)
expr	임의의 연산식이다. expr 앞에 DISTINCT 예약어를 포함하면, 최댓값을 구하기 전에 expr 값 중에서 중복된 값을 먼저 제거한다.
analytic_clause	OVER analytic_clause를 사용해 함수를 분석 함수로 수행할 수 있다. 자세한 내용은 “4.1.3. 분석 함수”의 analytic_clause를 참고한다.

- 예제

다음은 MAX 함수를 사용하는 예이다.

```
SQL> SELECT DEPTID, MAX(SALARY) FROM EMP2 GROUP BY DEPTID;

    DEPTID MAX(SALARY)
---------- -----------
         1        4500
         2        6000

2 rows selected.
```

- 분석 함수 예제

다음은 MAX 함수를 분석 함수로 사용하는 예이다.

```
SQL> SELECT NAME, DEPTID, SALARY, MAX(SALARY)
       OVER (PARTITION BY DEPTID) AS MSAL
     FROM EMP2;

NAME                    DEPTID     SALARY       MSAL
-------------------- ---------- ---------- ----------
Paul                         1       3000       4500
Nick                         1       3200       4500
Scott                        1       4000       4500
John                         1       4500       4500
Bree                         2       6000       6000
Daniel                       2       5000       6000
Joe                          2       4000       6000
Brad                         2       4200       6000

8 rows selected.
```

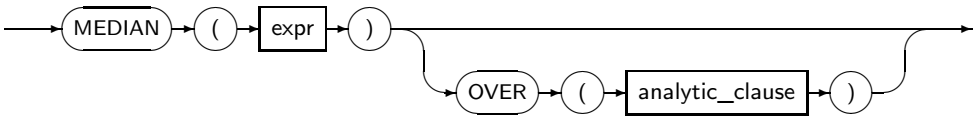
4.2.96. MEDIAN

MEDIAN은 파라미터로 주어진 expr의 그룹 안에서의 중간 값을 계산하는 함수이다. 분석 함수로도 사용할 수 있다. 계산을 할 때 NULL 값은 무시한다. 계산 방법과 계산 결과는 PERCENTILE_CONT에 파라미터 값 0.5를 사용한 것과 동일하다.

MEDIAN의 세부 내용은 다음과 같다.

- 문법

median



- 구성요소

구성요소	설명
expr	수치 값을 반환하는 임의의 연산식이다.
analytic_clause	OVER analytic_clause를 사용해 함수를 분석 함수로 수행할 수 있다. 자세한 내용은 “4.1.3. 분석 함수”의 analytic_clause를 참고한다.

- 예제

다음은 MEDIAN 함수를 사용하는 예이다.

```
SQL> SELECT DEPTNO, MEDIAN(SAL) FROM EMP AS MEDIAN GROUP BY DEPTNO;

    DEPTNO MEDIAN(SAL)
---------- -----------
        10        2450
        20        2975
        30        1375

3 rows selected.
```

- 분석 함수 예제

  다음은 MEDIAN 함수를 분석 함수로 사용하는 예이다.

```
SQL> SELECT DEPTNO, EMPNO, MEDIAN(SAL)
       OVER (PARTITION BY DEPTNO) AS MEDIAN
     FROM EMP;

    DEPTNO      EMPNO     MEDIAN
---------- ---------- ----------
        10       7934       2450
        10       7782       2450
        10       7839       2450
        20       7369       2975
        20       7876       2975
        20       7566       2975
        20       7788       2975
        20       7902       2975
        30       7900       1375
        30       7521       1375
        30       7654       1375
        30       7844       1375
        30       7499       1375
        30       7698       1375

14 rows selected.
```

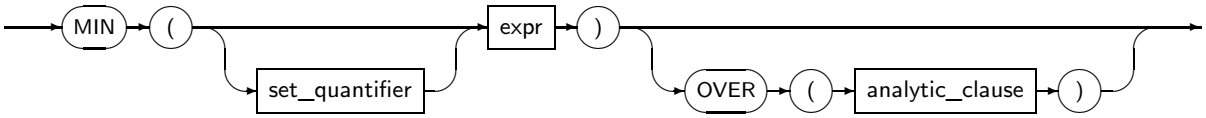
4.2.97. MIN

MIN은 그룹 내의 모든 로우에 대한 expr 값 중의 최솟값을 구하는 함수이다. 분석 함수로도 사용할 수 있다. 이 함수를 분석 함수로 사용할 때 DISTINCT 예약어를 명시하면 analytic_clause에서 partition_by만 명시할 수 있다. order_by_clause는 명시할 수 없다.

MIN의 세부 내용은 다음과 같다.

- 문법

min



- 구성요소

구성요소	설명
set_quantifier	질의 결과에 중복된 로우의 허용, 비허용 여부를 지정한다. DISTINCT, UNIQUE, ALL을 지정할 수 있다. – DISTINCT, UNIQUE : 중복된 로우를 제거한다. – ALL : 모든 로우를 선택한다. (기본값)
expr	임의의 연산식이다. expr 앞에 DISTINCT 예약어를 포함하면, 최솟값을 구하기 전에 expr 값 중에서 중복된 값을 먼저 제거한다.
analytic_clause	OVER analytic_clause를 사용해 함수를 분석 함수로 수행할 수 있다. 자세한 내용은 “4.1.3. 분석 함수”의 analytic_clause를 참고한다.

- 예제

다음은 MIN 함수를 사용하는 예이다.

```
SQL> SELECT DEPTID, MIN(SALARY) FROM EMP2 GROUP BY DEPTID;

    DEPTID MIN(SALARY)
---------- ----------
         1       3000
         2       4000

2 rows selected.
```

- 분석 함수 예제

다음은 MIN 함수를 분석 함수로 사용하는 예이다.

```
SQL> SELECT NAME, DEPTID, SALARY, MIN(SALARY)
       OVER (PARTITION BY DEPTID) AS MSAL
     FROM EMP2;

NAME                      DEPTID     SALARY       MSAL
-------------------- ---------- ---------- ----------
Paul                          1       3000       3000
Nick                          1       3200       3000
Scott                         1       4000       3000
John                          1       4500       3000
```

```
Bree                            2       6000       4000
Daniel                          2       5000       4000
Joe                             2       4000       4000
Brad                            2       4200       4000

8 rows selected.
```

4.2.98. MOD

MOD는 num1을 num2로 나눈 나머지를 반환하는 함수이다. 이 함수는 num1 또는 num2가 음수이면 전통적인 모듈러스(Modulus) 함수와는 다른 결과를 반환한다.

MOD 함수는 다음과 같이 정의할 수 있다.

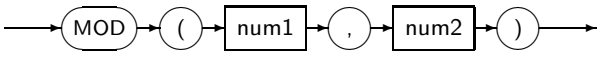
```
MOD(num1, num2) = SIGN(num1) * MOD1(ABS(num1), ABS(num2))
```

위의 수식에서 MOD1은 전통적인 모듈러스 함수이며, num1과 num2가 모두 양수이면 MOD 함수와 같은 결과를 반환한다. 위의 SIGN 함수는 num1이 양수이면 +1, 음수이면 -1을 반환하며, ABS 함수는 num1과 num2의 절댓값을 반환한다.

MOD의 세부 내용은 다음과 같다.

- 문법

mod



- 구성요소

구성요소	설명
num1, num2	임의의 수치 값을 반환하는 연산식이다. num1이나 num2는 숫자형 타입이거나 숫자형 타입으로 변환될 수 있는 타입이어야 한다. num1이나 num2은 두 타입 중 숫자형 우선순위에 따라 더 높은 순위의 타입으로 변환되고, 또한 그 타입으로 반환된다.

- 예제

다음은 MOD 함수를 사용하는 예이다.

```
SQL> SELECT MOD(13, 5), MOD(13, -5), MOD(-13, 5), MOD(-13, -5)
     FROM DUAL;

 MOD(13,5) MOD(13,-5) MOD(-13,5) MOD(-13,-5)
```

```
---------- ---------- ---------- ----------
         3          3         -3         -3

1 row selected.
```

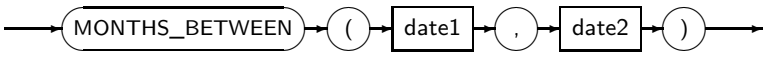
4.2.99. MONTHS_BETWEEN

MONTHS_BETWEEN은 date1과 date2 사이의 개월 차를 구하는 함수이다. 두 날짜 사이의 일수를 31로 나눈 값을 반환한다.

MONTHS_BETWEEN의 세부 내용은 다음과 같다.

- 문법

months_between



- 구성요소

구성요소	설명
date1, date2	날짜를 반환하는 임의의 연산식이다.

- 예제

다음은 MONTHS_BETWEEN 함수를 사용하는 예이다.

```
SQL> SELECT MONTHS_BETWEEN(LAST_DAY('2005/06/22'), '2005/06/22')
     FROM DUAL;

MONTHS_BETWEEN(LAST_DAY('2005/06/22'),'2005/06/22')
---------------------------------------------------
                                  .258064516129032

1 row selected.
```

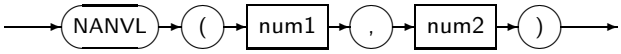
4.2.100. NANVL

NANVL은 num1이 NaN(Not A Number)가 아니면 num1을 반환하고, num1이 NaN이면 num2를 반환하는 함수이다.

NANVL의 세부 내용은 다음과 같다.

- 문법

nanvl



- 구성요소

구성요소	설명
num1, num2	num1이나 num2는 숫자형 타입이거나 숫자형 타입으로 변환될 수 있는 타입이어야 한다. num1이나 num2는 두 타입 중 숫자형 우선순위에 따라 더 높은 순위의 타입으로 변환되고, 그 타입으로 반환된다.

- 예제

다음은 NANVL 함수를 사용하는 예이다.

```
SQL> SELECT NAME, INC_PCT FROM SALARY;

NAME                    INC_PCT
-------------------- ----------
Smith                1.34E+001
Jane                        NaN

2 rows selected.

SQL> SELECT NAME, NANVL(INC_PCT, 0) INC_PCT FROM SALARY;

NAME                    INC_PCT
-------------------- ----------
Smith                1.34E+001
Jane                         0

2 rows selected.
```

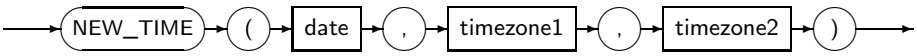
4.2.101. NEW_TIME

NEW_TIME는 timezone1에 속해있는 date값을 timezone2에 해당하는 날짜/시간값으로 변환하여 그 결과를 반환하는 함수이다.

NEW_TIME의 세부 내용은 다음과 같다.

- 문법

new_time



- 구성요소

구성요소	설명
date	날짜를 반환하는 임의의 연산식이다.
timezone1	시간대값을 반환하는 임의의 연산식으로 다음 문자열들을 사용할 수 있다. – AST, ADT : 대서양 표준시, 대서양 일광절약시간 – BST, BDT : 베링 표준시, 베링 일광절약시간 – CST, CDT : 중부 표준시, 중부 일광절약시간 – EST, EDT : 동부 표준시, 동부 일광절약시간 – GMT : 그리니치 표준시 – HST, HDT : 알래스카-하와이 표준시, 알래스카-하와이 일광절약시간 – MST, MDT : 산악 표준시, 산악 일광절약시간 – NST : 뉴펀들랜드 표준시 – PST, PDT : 태평양 표준시, 태평양 일광절약시간 – YST, YDT : 유콘 표준시, 유콘 일광절약시간
timezone2	시간대 값을 반환하는 임의의 연산식으로 timezone1과 동일한 문자열들을 사용할 수 있다.

- 예제

다음은 NEW_TIME 함수를 사용하는 예이다.

```
SQL> SELECT NEW_TIME(TO_DATE('1982/12/13 15:28:00',
     'YYYY/MM/DD HH24:MI:SS'), 'EST', 'YST') NEW_TIME FROM DUAL;

NEW_TIME
--------------------------------
1982/12/13 11:28:00

1 row selected.
```

4.2.102. NEXT_DAY

NEXT_DAY는 date와 가장 가까운 다음 요일 str의 날짜를 반환하는 함수이다.

NEXT_DAY의 세부 내용은 다음과 같다.

- 문법

next_day

- 구성요소

구성요소	설명
date	날짜를 반환하는 임의의 연산식이다.
str	요일을 나타내는 문자열 값이다.

- 예제

다음은 NEXT_DAY 함수를 사용하는 예이다.

```
SQL> SELECT NEXT_DAY('2005/06/22', 'MONDAY') FROM DUAL;

NEXT_DAY('2005/06/22','MONDAY')
-------------------------------
2005/06/27

1 row selected.
```

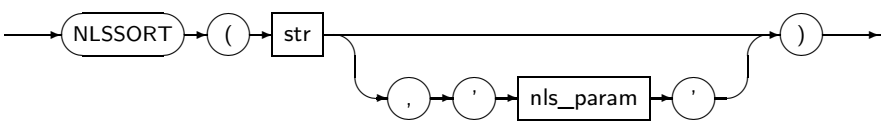
4.2.103. NLSSORT

NLSSORT는 str를 정렬하기 위해 사용되는 문자 바이트를 반환하는 함수이다.

NLSSORT의 세부 내용은 다음과 같다.

- 문법

nlssort



- 구성요소

구성요소	설명
str	정렬하기 위한 문자열 값이다. CHAR, VARCHAR, NCHAR, NVARCHAR 타입의 데이터가 올 수 있다.
nls_param	str를 정렬하기 위해 사용할 문자 집합을 정의하는 파라미터이다. CHAR, VARCHAR, NCHAR, NVARCHAR 타입의 데이터가 올 수 있다. 'NLS_SORT=sort'와 같은 형식으로 정의할 수 있다. 정의하지 않으면 세션에 정의된 값을 사용한다.

- 예제

  다음은 NLSSORT 함수를 사용하는 예이다.

```
SQL> CREATE TABLE T (NAME VARCHAR(10));
SQL> INSERT INTO T VALUES('jclee');
SQL> INSERT INTO T VALUES(' voir');
SQL> SELECT NAME FROM T ORDER BY NLSSORT(NAME, 'NLS_SORT=german');

NAME
----------
 voir
jclee

2 rows selected.
```

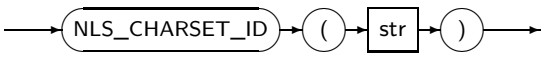
4.2.104. NLS_CHARSET_ID

NLS_CHARSET_ID는 인자로 받은 characterset name에 해당하는 characterset의 characterset id를 반환해 주는 함수이다.

NLS_CHARSET_ID의 세부 내용은 다음과 같다.

- 문법

  nls_charset_id



- 구성요소

구성요소	설명
str	Characterset id를 반환할 characterset의 characterset name 이다.

구성요소	설명
	CHAR, VARCHAR, NCHAR, NVARCHAR 타입의 데이터가 올 수 있다.

- 예제

  다음은 NLS_CHARSET_ID 함수를 사용하는 예이다.

```
SQL> SELECT NLS_CHARSET_ID('MSWIN949') FROM DUAL;

NLS_CHARSET_ID('MSWIN949')
--------------------------
                         2

1 row selected.
```

4.2.105. NLS_INITCAP

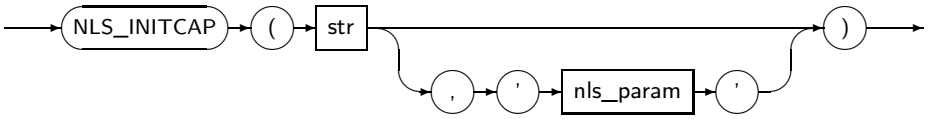
NLS_INITCAP은 각 단어의 첫 글자를 대문자로, 다른 나머지 글자는 소문자로 변환하는 함수이다.

str에는 CHAR, VARCHAR, NCHAR, NVARCHAR 형식의 문자열이 올 수 있고, 결과는 입력된 str의 문자 집합과 동일한 형식의 값이 반환된다. nls_param은 'NLS_SORT=sort'와 같은 형식으로 정의될 수 있으며, 정의되지 않았을 경우에는 세션에 정의된 값을 사용하게 된다.

NLS_INITCAP의 세부 내용은 다음과 같다.

- 문법

nls_initcap



- 구성요소

구성요소	설명
str	변환하기 위한 문자열 값이다. CHAR, VARCHAR, NCHAR, NVARCHAR 타입의 데이터가 올 수 있다. 변환 결과는 입력된 str의 문자 집합과 동일한 형식의 값이 반환된다.
nls_param	str를 변환하기 위해 사용할 문자 집합을 정의하는 파라미터이다. CHAR, VARCHAR, NCHAR, NVARCHAR 타입의 데이터가 올 수 있다.

구성요소	설명
	'NLS_SORT=sort'와 같은 형식으로 정의할 수 있다. 정의하지 않으면 세션에 정의된 값을 사용한다.

- 예제

다음은 NLS_INITCAP 함수를 사용하는 예이다.

```
SQL> SELECT NLS_INITCAP('ijland','NLS_SORT=XDutch') "NLS_INITCAP"
     FROM DUAL;

NLS_INITCAP
-----------
IJland

1 row selected.
```

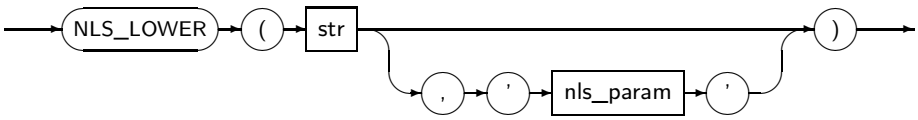
4.2.106. NLS_LOWER

NLS_LOWER는 입력된 문자열을 모두 소문자로 변환하는 함수이다.

NLS_LOWER의 세부 내용은 다음과 같다.

- 문법

nls_lower



- 구성요소

구성요소	설명
str	변환하기 위한 문자열 값이다. CHAR, VARCHAR, NCHAR, NVARCHAR 타입의 데이터가 올 수 있다.
nls_param	str를 변환하기 위해 사용할 문자 집합을 정의하는 파라미터이다. CHAR, VARCHAR, NCHAR, NVARCHAR 타입의 데이터가 올 수 있다. 'NLS_SORT=sort'와 같은 형식으로 정의할 수 있다. 정의하지 않으면 세션에 정의된 값을 사용한다.

- 예제

  다음은 NLS_LOWER 함수를 사용하는 예이다.

```
SQL> SELECT NLS_LOWER('GROBE','NLS_SORT=XGerman') "NLS_LOWER"
     FROM DUAL;

NLS_LOWER
--------
   grobe

1 row selected.
```

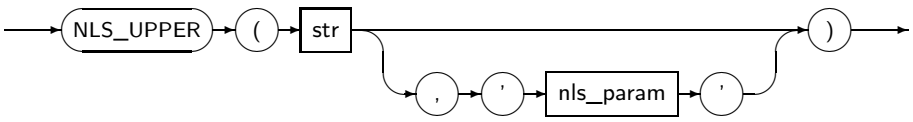
4.2.107. NLS_UPPER

NLS_UPPER는 입력된 문자열을 모두 대문자로 변환하는 함수이다.

NLS_UPPER의 세부 내용은 다음과 같다.

- 문법

nls_upper



- 구성요소

구성요소	설명
str	변환하기 위한 문자열 값이다. CHAR, VARCHAR, NCHAR, NVARCHAR 타입의 데이터가 올 수 있다.
nls_param	str를 변환하기 위해 사용할 문자 집합을 정의하는 파라미터이다. CHAR, VARCHAR, NCHAR, NVARCHAR 타입의 데이터가 올 수 있다. 'NLS_SORT=sort'와 같은 형식으로 정의할 수 있다. 정의하지 않으면 세션에 정의된 값을 사용한다.

- 예제

  다음은 NLS_UPPER 함수를 사용하는 예이다.

```
SQL> SELECT NLS_UPPER('große','NLS_SORT=XGerman') "NLS_UPPER"
     FROM DUAL;
```

```
NLS_UPPER
---------
   GROSSE

1 row selected.
```

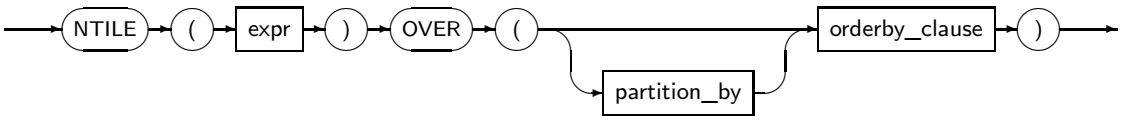
4.2.108. NTILE

NTILE은 그룹 내의 정렬된 행을 expr개의 버킷으로 나누어 버킷의 번호를 지정해 주는 분석 함수이다.

NTILE의 세부 내용은 다음과 같다.

- 문법

ntile_analytic



- 구성요소

구성요소	설명
expr	expr이 정수가 아니면 소수점 이하를 버린 값을 사용한다. expr에 LEAD를 포함한 다른 분석 함수를 명시할 수 없다. 즉, 분석 함수를 중첩해서 사용할 수는 없다.

- 예제

다음은 NTILE 함수를 사용하는 예이다.

```
SQL> SELECT NTILE(5) OVER (ORDER BY SAL) AS NTILE
     FROM EMP;

     NTILE
----------
         1
         1
         1
         2
         2
         2
         3
```

```
         3
         3
         4
         4
         4
         5
         5

14 rows selected.
```

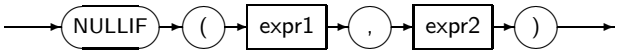
4.2.109. NULLIF

NULLIF는 만약 expr1과 expr2의 값이 동일하면 NULL을 반환하고, 그렇지 않으면 expr1 값을 반환하는 함수이다.

NULLIF의 세부 내용은 다음과 같다.

- 문법

  nullif

- 구성요소

구성요소	설명
expr1	임의의 연산식이며, expr1에 NULL이 올 수 없다.
expr2	임의의 연산식이다.

- 예제

  다음은 NULLIF 함수를 사용하는 예이다.

```
SQL> SELECT NVL (NULLIF ('A', 'A'), 'Same') FROM DUAL;

NVL(NULLIF('A','A'),'SAME')
---------------------------
Same

1 row selected.
```

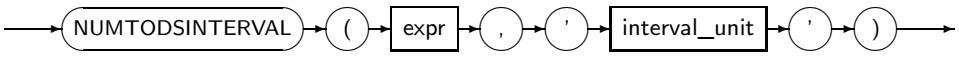
4.2.110. NUMTODSINTERVAL

NUMTODSINTERVAL은 입력으로 들어온 값을 날짜-시간 구간 형식으로 변환하는 함수이다.

NUMTODSINTERVAL의 세부 내용은 다음과 같다.

- 문법

numtodsinterval



- 구성요소

구성요소	설명
expr	변환할 입력 값이다.
interval_unit	변환의 단위를 나타내는 값이다. DAY, HOUR, MINUTE, SECOND 중 하나의 값만 올 수 있다.

- 예제

다음은 NUMTODSINTERVAL 함수를 사용하는 예이다.

```
SQL> SELECT NUMTODSINTERVAL (10, 'DAY') "NUMTODSINTERVAL"
     FROM DUAL;

NUMTODSINTERVAL
-----------------------------
+000000010 00:00:00.000000000

1 row selected.
```

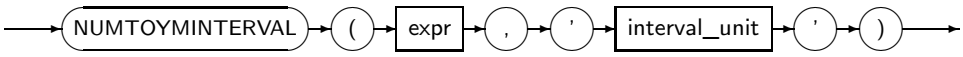
4.2.111. NUMTOYMINTERVAL

NUMTOYMINTERVAL은 입력으로 들어온 값을 년도-달 구간 형식으로 변환하는 함수이다.

NUMTOYMINTERVAL의 세부 내용은 다음과 같다.

- 문법

numtoyminterval



- 구성요소

구성요소	설명
expr	변환할 입력 값이다.
interval_unit	변환의 단위를 나타내는 값이다. YEAR, MONTH 중 하나의 값만 올 수 있다.

- 예제

  다음은 NUMTOYMINTERVAL 함수를 사용하는 예이다.

```
SQL> SELECT NUMTOYMINTERVAL (10, 'YEAR') "NUMTOYMINTERVAL"
     FROM DUAL;

NUMTOYMINTERVAL
---------------
+000000010-00

1 row selected.
```

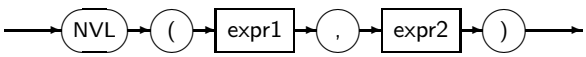
4.2.112. NVL

NVL은 expr1이 NULL이 아니면 expr1을 반환하고, expr1이 NULL이면 expr2를 반환하는 함수이다.

NVL의 세부 내용은 다음과 같다.

- 문법

  nvl



- 구성요소

구성요소	설명
expr1	임의의 연산식이다.
expr2	임의의 연산식이다.

- 예제

  다음은 NVL 함수를 사용하는 예이다.

```
SQL> SELECT NAME, ID, NVL(TO_CHAR(MGRID, '99'), 'N/A') MID
     FROM EMP3;

NAME                          ID MID
-------------------- ---------- ---
Paul                           1 N/A
```

```
John                           2   1
Linda                          3   1
Lucas                          4 N/A
Kathy                          5   4

5 rows selected.
```

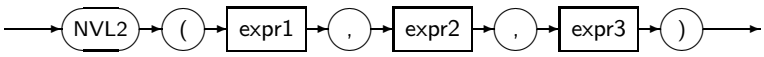
4.2.113. NVL2

NVL2는 expr1이 NULL이 아니면 expr2를 반환하고 NULL이면 expr3를 반환하는 함수이다.

NVL2의 세부 내용은 다음과 같다.

- 문법

nvl2



- 구성요소

구성요소	설명
expr1	임의의 연산식이다.
expr2	임의의 연산식이다. expr2의 데이터 타입에 따라 반환되는 타입이 결정된다.
expr3	임의의 연산식이다. expr3은 expr2의 데이터 타입으로 암묵적으로 변환된다.

- 예제

다음은 NVL2 함수를 사용하는 예이다.

```
SQL> SELECT NVL2(DUMMY, 'NOT NULL', 'NULL') FROM DUAL;

NVL2(DUMMY, 'NOT NULL', 'NULL')
--------------------------------
NOT NULL

1 row selected.
```

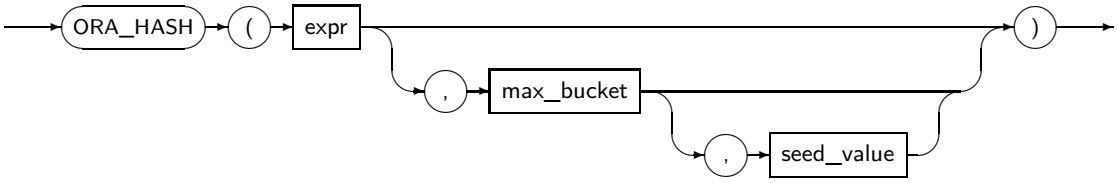
4.2.114. ORA_HASH

ORA_HASH는 주어진 데이터에 대해 해시값을 계산하고 그 결과를 반환하는 함수이다.

ORA_HASH의 세부 내용은 다음과 같다.

- 문법

ora_hash



- 구성요소

구성요소	설명
expr	해시 값을 계산하기 위한 데이터를 나타내는 임의의 연산식이다. LONG과 LOB을 제외한 모든 타입을 사용할 수 있다.
max_bucket	최대 버킷 크기를 나타내는 숫자 연산식이다. 0부터 4294967295 사이의 값을 사용할 수 있다. (기본값: 4294967295)
seed_value	같은 데이터에 대해 해시 값의 변화를 줄 수 있는 숫자 연산식이다. 0부터 4294967295 사이의 값을 사용할 수 있다. (기본값: 0)

- 예제

다음은 ORA_HASH 함수를 사용하는 예이다.

```
SQL> SELECT ORA_HASH(512, 10, 5) FROM DUAL;

ORA_HASH(512,10,5)
------------------
                 4

1 row selected.
```

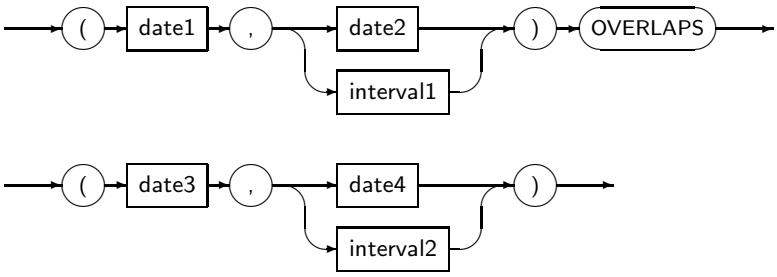
4.2.115. OVERLAPS

OVERLAPS는 두 개의 시간 간격이 서로 겹치는지의 여부를 알아보는 함수이다. 겹칠 때는 TRUE를, 겹치지 않을 때는 FALSE를 반환한다. 시간 간격은 시작점과 끝점이 한 쌍으로 명시된다. 시작점 또는 끝점은 DATE, TIMESTAMP 타입이 가능하며, 끝점에 대해서 추가로 INTERVAL을 명시하는 것도 가능하다.

OVERLAPS의 세부 내용은 다음과 같다.

- 문법

overlaps



- 구성요소

구성요소	설명
date1, date3	시작 날짜를 나타낸다. DATE 또는 TIMESTAMP 타입이다.
date2, date4	끝 날짜를 나타낸다. DATE 또는 TIMESTAMP 타입이다.
interval1, interval2	날짜의 간격을 나타내는 간격 리터럴이다.

- 예제

다음은 OVERLAPS 함수를 사용하는 예이다.

```
SQL> SELECT 1
     FROM DUAL
     WHERE (DATE'1999-01-01', DATE'2000-01-01') OVERLAPS
           (DATE'1999-03-01', INTERVAL '1' YEAR);

         1
----------
         1

1 row selected.
```

4.2.116. PERCENT_RANK

PERCENT_RANK는 파라미터로 주어진 값의 그룹 내의 위치를 나타내 주는 함수이다. 반환되는 값은 0 ~ 1 사이의 값이고 NUMBER 타입이다. 분석 함수로도 사용할 수 있다.

계산 방법은 다음과 같이 함수에 따라 달라진다.

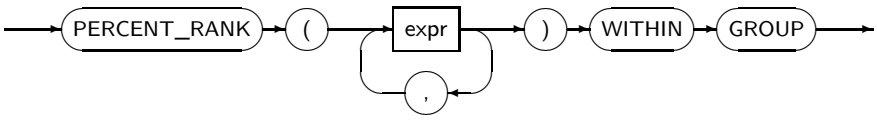
함수	설명
집계 함수	주어진 파라미터로 구성된 가상의 로우의 RANK 값을 계산하여 1을 빼고 그룹 안의 로우의 개수로 나눈다. 파라미터의 값은 그룹 안에서는 상수 값을 가져야 하며 ORDER BY절의 표현식과 대응되어야 한다.

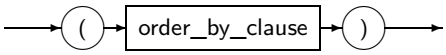
함수	설명
분석 함수	주어진 파라미터로 구성된 가상의 로우의 RANK 값에서 1을 뺀 값을 그룹 안의 로우의 개수에서 1을 뺀 값으로 나눈다.

PERCENT_RANK의 세부 내용은 다음과 같다.

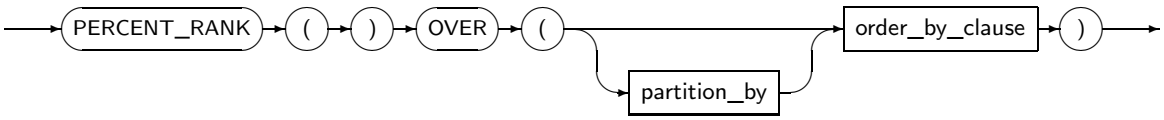
- 문법

percent_rank_aggregate





percent_rank_analytic



- 구성요소

구성요소	설명
expr	수치 값을 반환하는 임의의 연산식이다.
partition_by	현재 질의 블록의 결과 집합을 expr 또는 expr의 리스트를 기준으로 분할한다. 자세한 내용은 "4.1.3. 분석 함수"의 partition_by를 참고한다.
order_by_clause	분할된 하나의 파티션 내에서 로우를 어떻게 정렬할지를 명시한다. 자세한 내용은 "4.1.3. 분석 함수"의 order_by_clause를 참고한다.

- 예제

다음은 PERCENT_RANK 함수를 사용하는 예이다.

```
SQL> SELECT PERCENT_RANK(1000) WITHIN GROUP (ORDER BY SAL)
       AS PERCENT_RANK
     FROM EMP;

PERCENT_RANK
------------
.14285714286

1 row selected.
```

- 분석 함수 예제

다음은 PERCENT_RANK 함수를 분석 함수로 사용하는 예이다.

```
SQL> SELECT PERCENT_RANK()
       OVER (PARTITION BY DEPTNO ORDER BY SAL) AS PERCENT_RANK
     FROM EMP;

PERCENT_RANK
------------
           0
          .5
           1
           0
         .25
          .5
         .75
         .75
           0
          .2
          .2
          .6
          .8
           1

14 rows selected.
```

4.2.117. PERCENTILE_CONT

PERCENTILE_CONT는 연속 분포 모델에서 파라미터로 주어진 백분위 값에 해당하는 값을 계산하는 역분포 함수이다. 분석 함수로도 사용할 수 있다. 계산을 할 때 NULL 값은 무시한다.

파라미터의 표현식의 값은 백분위 값으로 그룹 내에서 0에서 1사이의 상수여야 한다. ORDER BY 절의 표현식은 보간을 계산할 수 있는 수치 또는 날짜형의 데이터 타입이어야 한다.

PERCENTILE_CONT는 정렬 스펙에 맞게 정렬한 후 보간법에 따라 계산한다. 이를 위해 다음과 같이 RN, CRN, FRN 값을 계산한다.

```
RN=(1+(P*(N-1)))
CRN=CEILING(RN)
FRN=FLOOR(RN)
```

구분	설명
P	파라미터로 주어진 백분위 값이다.
N	그룹 내의 NULL이 아닌 로우의 개수이다.

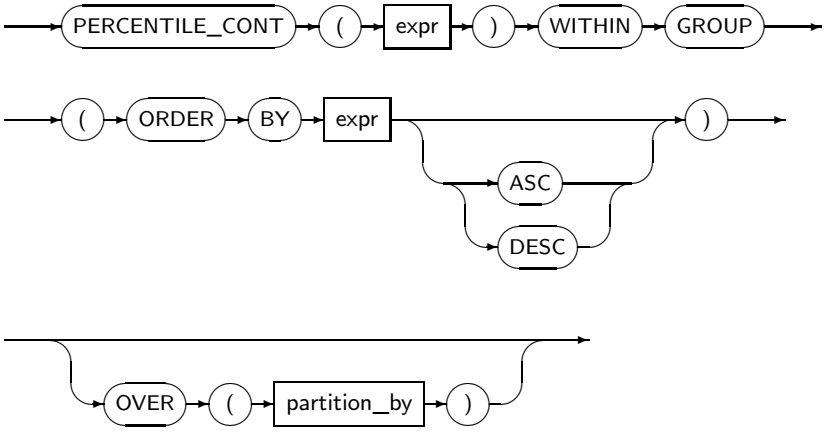
최종 결과는 CRN과 FRN이 같다면 RN번째 행의 expr의 값이고, 다르다면 다음과 같이 계산한다.

```
(CRN-RN) * (value of expr at FRN) + (RN-FRN) * (value of expr at CRN)
```

PERCENTILE_CONT의 세부 내용은 다음과 같다.

- 문법

percentile_cont



- 구성요소

구성요소	설명
expr	수치 값을 반환하는 임의의 연산식이다.
ASC DESC	– ORDER BY ASC는 결과를 오름차순으로 정렬한다. (기본값) – ORDER BY DESC는 결과를 내림차순으로 정렬한다.
partition_by	현재 질의 블록의 결과 집합을 expr 또는 expr의 리스트를 기준으로 분할한다. 자세한 내용은 “4.1.3. 분석 함수”의 partition_by를 참고한다.

- 예제

다음은 PERCENTILE_CONT 함수를 사용하는 예이다.

```
SQL> SELECT DEPTNO, PERCENTILE_CONT(0.35)
       WITHIN GROUP (ORDER BY SAL)
     FROM EMP GROUP BY DEPTNO;

    DEPTNO PERCENTILE_CONT(0.35)WITHINGRO
---------- ------------------------------
        10                           2105
        20                           1850
        30                           1250
```

```
3 rows selected.
```

- 분석 함수 예제

  다음은 PERCENTILE_CONT 함수를 분석 함수로 사용하는 예이다.

```
SQL> SELECT DEPTNO, EMPNO, PERCENTILE_CONT(0.35)
       WITHIN GROUP (ORDER BY SAL) OVER (PARTITION BY DEPTNO)
     FROM EMP;

    DEPTNO      EMPNO PERCENTILE_CONT(0.35)WITHINGRO
---------- ---------- ------------------------------
        10       7934                           2105
        10       7782                           2105
        10       7839                           2105
        20       7369                           1850
        20       7876                           1850
        20       7566                           1850
        20       7788                           1850
        20       7902                           1850
        30       7900                           1250
        30       7521                           1250
        30       7654                           1250
        30       7844                           1250
        30       7499                           1250
        30       7698                           1250

14 rows selected.
```

4.2.118. PERCENTILE_DISC

PERCENTILE_DISC는 이산 분포를 가정한 역분산 함수이다. 분석 함수로도 사용할 수 있다. 계산을 할 때 NULL 값은 무시한다. 파라미터의 표현식의 값은 백분위 값으로 그룹 내에서 0 ~ 1사이의 상수이어야 한다. ORDER BY 절로 정렬을 할 수 있는 타입이어야 한다.

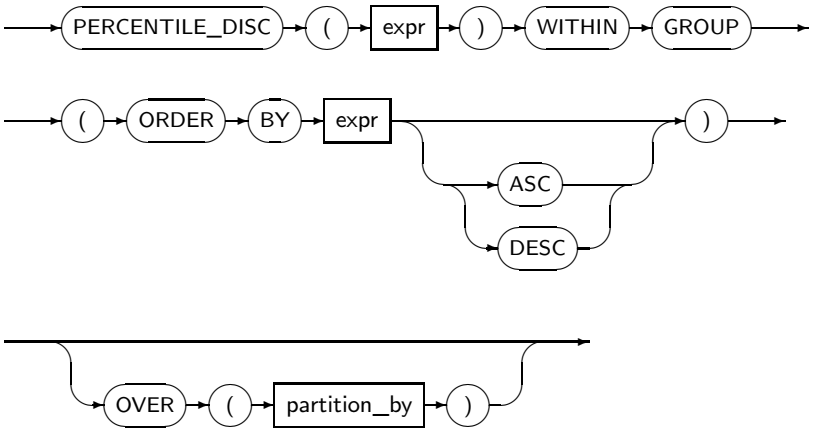
PERCENTILE_DISC의 값을 계산하기 위해 다음과 같이 RN을 계산한다. 계산된 결과 값은 CRN에 해당하는 행의 expr 값이다.

```
RN=N*P
CRN=CEILING(RN)
```

PERCENTILE_DISC의 세부 내용은 다음과 같다.

- 문법

percentile_disc



- 구성요소

구성요소	설명
expr	수치 값을 반환하는 임의의 연산식이다.
ASC DESC	– ORDER BY ASC는 결과를 오름차순으로 정렬한다. (기본값) – ORDER BY DESC는 결과를 내림차순으로 정렬한다.
partition_by	현재 질의 블록의 결과 집합을 expr 또는 expr의 리스트를 기준으로 분할한다. 자세한 내용은 “4.1.3. 분석 함수”의 partition_by를 참고한다.

- 예제

  다음은 PERCENTILE_DISC 함수를 사용하는 예이다.

```
SQL> SELECT DEPTNO, PERCENTILE_DISC(0.15)
       WITHIN GROUP (ORDER BY SAL) AS PERCENTILE_DISC
     FROM EMP GROUP BY DEPTNO;

    DEPTNO PERCENTILE_DISC
---------- ---------------
        10            1300
        20             800
        30             950

3 rows selected.
```

- 분석 함수 예제

  다음은 PERCENTILE_DISC 함수를 분석 함수로 사용하는 예이다.

```
SQL> SELECT DEPTNO, PERCENTILE_DISC(0.15)
       WITHIN GROUP (ORDER BY SAL) OVER (PARTITION BY DEPTNO) AS PERCENTILE_DISC
     FROM EMP;
```

```
    DEPTNO PERCENTILE_DISC
---------- ---------------
        10            1300
        10            1300
        10            1300
        20             800
        20             800
        20             800
        20             800
        20             800
        30             950
        30             950
        30             950
        30             950
        30             950
        30             950

14 rows selected.
```

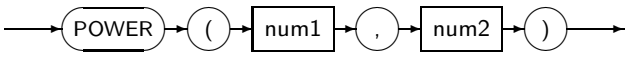
4.2.119. POWER

POWER는 num1의 num2 제곱 값(num1^{num2})을 반환하는 함수이다.

POWER의 세부 내용은 다음과 같다.

- 문법

power



- 구성요소

구성요소	설명
num1, num2	수치 값을 반환하는 임의의 연산식이다. num1이 음수인 경우 num2는 정수이어야 한다. num1이나 num2는 숫자형 타입이거나 숫자형 타입으로 변환될 수 있는 타입이어야 한다. 만약 num1이나 num2의 타입이 BINARY_FLOAT이나 BINARY_DOUBLE일 경우엔 BINARY_DOUBLE 타입을 반환하고, 그렇지 않은 경우엔 NUMBER 타입을 반환한다.

- 예제

다음은 POWER 함수를 사용하는 예이다.

```
SQL> SELECT POWER(2, 3) FROM DUAL;

POWER(2,3)
----------
         8

1 row selected.
```

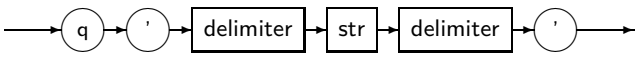
4.2.120. QOUTED_STRING

QOUTED_STRING은 문자열 str의 값을 반환하는 함수이다.

QOUTED_STRING의 세부 내용은 다음과 같다.

- 문법

quoted_string



- 구성요소

구성요소	설명
delimiter	구분자이다. 구분자로는 출력 가능한 모든 ASCII 문자(알파벳, 숫자, 특수문자)를 사용할 수 있다. 이때, 만약 시작 구분자가 열린 괄호(<, (, {, [)일 경우 종료 구분자는 이에 대응되는 닫힌 괄호(>,), },])여야 한다. 종료 구분자를 입력하지 않은 경우 오류 메시지가 출력된다.
str	문자열을 반환하는 임의의 연산식이다.

- 예제

다음은 QUOTED_STRING 함수를 사용하는 예이다.

```
SQL> select q'$Dollars$' from dual;

Q'$DOLLARS$'
------------
Dollars
```

4.2.121. RANK

RANK는 그룹별로 로우를 정렬한 후 그룹 내의 각 로우의 순위를 반환하는 함수이다. 분석 함수로도 사용할 수 있다.

순위의 데이터 타입은 NUMBER이다. 순위를 산정할 때 동률이 나타났을 경우는 동률인 모든 로우에 대해 같은 순위가 부여된다. 그 다음 부여되는 순위는 같은 순위가 부여된 로우의 개수만큼 증가된 값이 부여된다. 따라서 순위는 연속된 숫자가 아닐 수도 있다.

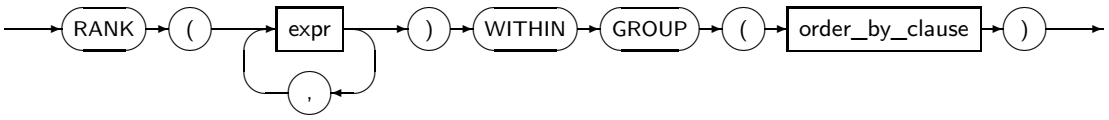
계산 방법은 다음과 같이 함수에 따라 달라진다.

함수	설명
집계 함수	파라미터 값으로 구성된 가상의 로우에 대한 순위 값을 계산한다. 파라미터는 각 그룹마다 상수 값을 가져야 하며 order_by_clause의 표현식과 대응되어야 한다.
분석 함수	각 로우의 그룹 내 순위를 반환한다. 순위는 order_by_clause 내의 expr 값을 기준으로 정렬한 결과가 부여된다.

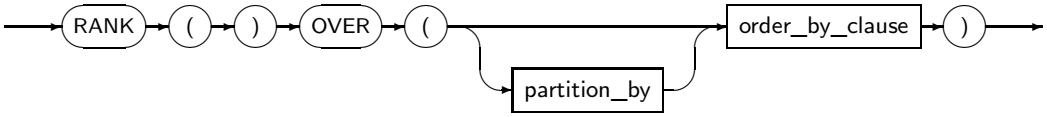
RANK의 세부 내용은 다음과 같다.

- 문법

rank_aggregation



rank_analytic



- 구성요소

구성요소	설명
expr	수치 값을 반환하는 임의의 연산식이다.
ASC DESC	– ORDER BY ASC는 결과를 오름차순으로 정렬한다. (기본값) – ORDER BY DESC는 결과를 내림차순으로 정렬한다.
partition_by	현재 질의 블록의 결과 집합을 expr 또는 expr의 리스트를 기준으로 분할한다. 자세한 내용은 "4.1.3. 분석 함수"의 partition_by를 참고한다.

- 예제

다음은 RANK 함수를 사용하는 예이다.

```
SQL> SELECT DEPTNO, RANK(3000) WITHIN GROUP (ORDER BY SAL) AS RANK
     FROM EMP
     GROUP BY DEPTNO;

    DEPTNO       RANK
---------- ----------
        10          3
        20          4
        30          7

3 rows selected.
```

- 분석 함수 예제

  다음은 RANK 함수를 분석 함수로 사용하는 예이다.

```
SQL> SELECT NAME, DEPTID, SALARY, RANK()
       OVER (PARTITION BY DEPTID ORDER BY SALARY)
     FROM EMP;

NAME                      DEPTID     SALARY RANK()OVER
-------------------- ---------- ---------- ----------
Paul                           1       3000          1
Angela                         1       3000          1
Nick                           1       3200          3
Scott                          1       4000          4
James                          1       4000          4
John                           1       4500          6
Joe                            2       4000          1
Brad                           2       4200          2
Daniel                         2       5000          3
Tom                            2       5000          3
Kathy                          2       5000          3
Bree                           2       6000          6

12 rows selected.
```

4.2.122. REGR_SLOPE

REGR_SLOPE는 임의의 수치 데이터 쌍의 집합에 가장 맞는 선형 방정식을 구하는 함수로, 집합함수와 분석함수로 사용된다.

입력으로 숫자 데이터 또는 숫자 데이터로 변환 가능한 데이터형을 갖는다. 인자로 받은 종속변수(expr1) 또는 독립변수(expr2)의 값이 NULL이면 계산에서 제외된다. 모든 로우가 NULL의 쌍이면 NULL을 결과로 낸다.

REGR_SLOPE는 다음의 계산식으로 회귀 직선(Regression Line)의 기울기를 계산해서 결과를 반환한다.

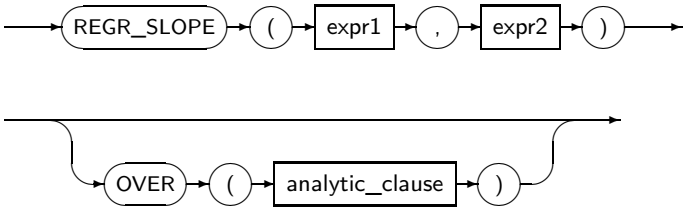
```
COVAR_POP(expr1, expr2) / VAR_POP(expr2)
```

이때 VAR_POP의 결과가 0이면 최종 결과는 NULL이 된다.

REGR_SLOPE의 세부 내용은 다음과 같다.

- 문법

regr_slope



- 구성요소

구성요소	설명
expr1	수치 값을 반환하는 임의의 연산식이다. 종속변수에 해당한다.
expr2	수치 값을 반환하는 임의의 연산식이다. 독립변수에 해당한다.

- 예제

다음은 함수를 사용하는 예이다.

```
SQL> SELECT * FROM XY;

         X          Y
---------- ----------
         1          1
         2          3

2 rows selected.

SQL> SELECT REGR_SLOPE(Y,X) FROM XY;

REGR_SLOPE(Y,X)
---------------
              2

1 row selected.

SQL> SELECT REGR_SLOPE(X,Y) FROM XY;

REGR_SLOPE(X,Y)
---------------
```

```
             .5

1 row selected.
```

4.2.123. REGR_INTERCEPT

REGR_INTERCEPT는 임의의 수치 데이터 쌍의 집합에 가장 맞는 선형 방정식을 구하기 위해 사용되는 함수, 집합 함수와 분석 함수로 사용된다.

입력으로 숫자 데이터 또는 숫자 데이터로 변환 가능한 데이터형을 갖는다. 인자로 받은 종속변수(expr1) 또는 독립변수(expr2)의 값이 NULL이면 계산에서 제외되며 모든 로우가 NULL의 쌍이면 NULL을 결과로 낸다.

REGR_INTERCEPT는 다음의 계산식으로 회귀 직선의 Y 절편을 계산해서 결과를 반환한다.

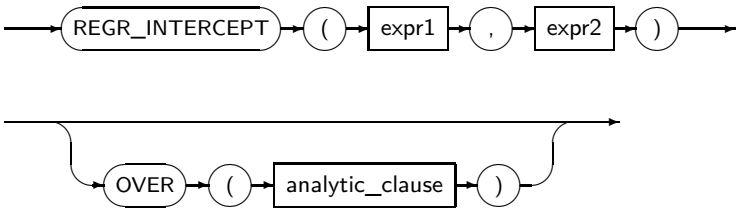
```
AVG(exr1) - REGR_SLOPE(expr1, expr2) × AVG(expr2)
```

이때 REGR_SLOPE의 결과가 NULL이면 최종 결과는 NULL이 된다.

REGR_INTERCEPT의 세부 내용은 다음과 같다.

- 문법

regr_intercept



- 구성요소

구성요소	설명
expr1	수치 값을 반환하는 임의의 연산식이다. 종속변수에 해당한다.
expr2	수치 값을 반환하는 임의의 연산식이다. 독립변수에 해당한다.

- 예제

다음은 함수를 사용하는 예이다.

```
SQL> SELECT * FROM XY;

         X          Y
---------- ----------
```

```
         1          1
         2          3

2 rows selected.

SQL> SELECT REGR_INTERCEPT(Y,X) FROM XY;

REGR_INTERCEPT(Y,X)
-------------------
                 -1

1 row selected.
```

4.2.124. REGR_COUNT

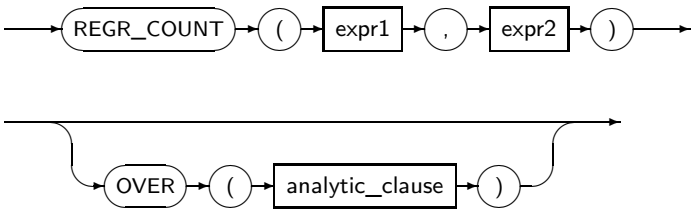
REGR_COUNT는 임의의 수치 데이터 쌍의 집합에 가장 맞는 선형 방정식을 구하는 함수로, 집합 함수와 분석 함수로 사용된다.

입력으로 숫자 데이터 또는 숫자 데이터로 변환 가능한 데이터형을 갖는다. 인자로 받은 종속변수(expr1) 또는 독립변수(expr2)의 값이 NULL이면 계산에서 제외되며 모든 로우가 NULL의 쌍이면 0을 결과로 낸다. REGR_COUNT는 회귀 직선을 구성하는데 사용된 NULL이 아닌 (expr1, expr2) 쌍의 계수를 반환한다.

REGR_COUNT의 세부 내용은 다음과 같다.

- 문법

regr_count



- 구성요소

구성요소	설명
expr1	수치 값을 반환하는 임의의 연산식이다. 종속변수에 해당한다.
expr2	수치 값을 반환하는 임의의 연산식이다. 독립변수에 해당한다.

- 예제

다음은 함수를 사용하는 예이다.

```
SQL> SELECT * FROM XY;

         X          Y
---------- ----------
         1          1
         2          3

2 rows selected.

SQL> SELECT REGR_COUNT(Y,X) FROM XY;

REGR_COUNT(Y,X)
---------------
              2

1 row selected.
```

4.2.125. REGR_R2

REGR_R2는 임의의 수치 데이터 쌍의 집합에 가장 맞는 선형 방정식을 구하는 함수로, 집합 함수와 분석 함수로 사용된다.

입력으로 숫자 데이터 또는 숫자 데이터로 변환 가능한 데이터형을 갖는다. 인자로 받은 종속변수(expr1) 또는 독립변수(expr2)의 값이 NULL이면 계산에서 제외되며 모든 로우가 NULL의 쌍이면 NULL을 결과로 낸다.

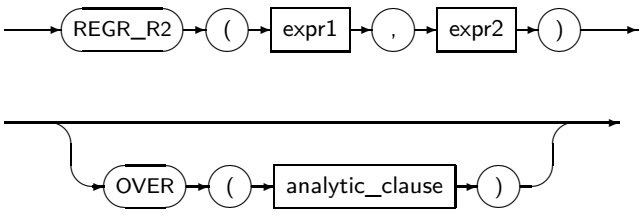
REGR_R2는 회귀에 대한 결정 계수(R-squared 또는 적합도)를 계산하는 함수로 VAR_POP(expr1)과 VAR_POP(expr2)의 값에 따라 아래와 같은 계산식을 이용한다.

- VAR_POP(expr2) = 0이면 : NULL
- VAR_POP(expr1) = 0이고 VAR_POP(expr2) > 0이면 : 1
- VAR_POP(expr1) > 0이고 VAR_POP(expr2) > 0이면 : POWER(CORR(expr1, expr2),2)

REGR_R2의 세부 내용은 다음과 같다.

- 문법

regr_r2



- 구성요소

구성요소	설명
expr1	수치 값을 반환하는 임의의 연산식이다. 종속변수에 해당한다.
expr2	수치 값을 반환하는 임의의 연산식이다. 독립변수에 해당한다.

- 예제

  다음은 함수를 사용하는 예이다.

```
SQL> SELECT * FROM XY;

         X          Y
---------- ----------
         1          1
         2          3

2 rows selected.

SQL> SELECT REGR_R2(Y,X) FROM XY;

REGR_R2(Y,X)
------------
           1

1 row selected.
```

4.2.126. REGR_AVGX

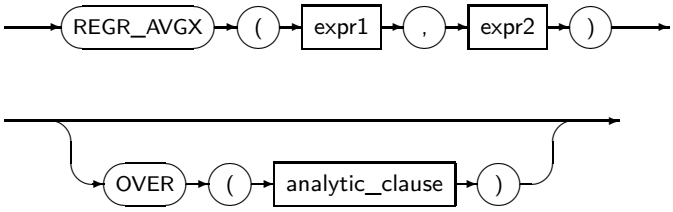
REGR_AVGX는 임의의 수치 데이터 쌍의 집합에 가장 맞는 선형 방정식을 구하는 함수로, 집합 함수와 분석 함수로 사용된다.

입력으로 숫자 데이터 또는 숫자 데이터로 변환 가능한 데이터형을 갖는다. 인자로 받은 종속변수(expr1) 또는 독립변수(expr2)의 값이 NULL이면 계산에서 제외되며 모든 로우가 NULL의 쌍이면 NULL을 결과로 낸다. REGR_AVGX는 독립변수(expr2)의 평균을 계산한다. 즉, NULL의 쌍을 제거한 후에 "AVG(expr2)"를 계산한다.

REGR_AVGX의 세부 내용은 다음과 같다.

- 문법

regr_avgx



- 구성요소

구성요소	설명
expr1	수치 값을 반환하는 임의의 연산식이다. 종속변수에 해당한다.
expr2	수치 값을 반환하는 임의의 연산식이다. 독립변수에 해당한다.

- 예제

  다음은 함수를 사용하는 예이다.

```
SQL> SELECT * FROM XY;

         X          Y
---------- ----------
         1          1
         2          3

2 rows selected.

SQL> SELECT REGR_AVGX(Y,X) FROM XY;

REGR_AVGX(Y,X)
--------------
           1.5

1 row selected.
```

4.2.127. REGR_AVGY

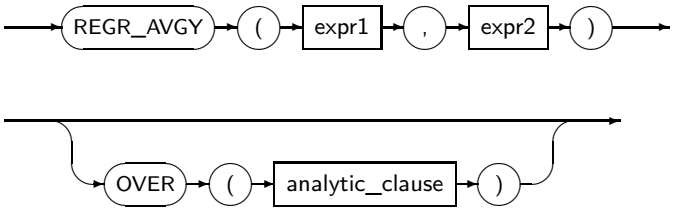
REGR_AVGY는 임의의 수치 데이터 쌍의 집합에 가장 맞는 선형 방정식을 구하는 함수로, 집합 함수와 분석 함수로 사용된다.

입력으로 숫자 데이터 또는 숫자 데이터로 변환 가능한 데이터형을 갖는다. 인자로 받은 종속변수(expr1) 또는 독립변수(expr2)의 값이 NULL이면 계산에서 제외되며 모든 로우가 NULL의 쌍이면 NULL을 결과로 낸다. REGR_AVGY는 종속변수(expr1)의 평균을 계산한다. 즉, NULL의 쌍을 제거한 후에 "AVG(expr1)"를 계산한다.

REGR_AVGY의 세부 내용은 다음과 같다.

- 문법

regr_avgy



- 구성요소

구성요소	설명
expr1	수치 값을 반환하는 임의의 연산식이다. 종속변수에 해당한다.
expr2	수치 값을 반환하는 임의의 연산식이다. 독립변수에 해당한다.

- 예제

다음은 함수를 사용하는 예이다.

```
SQL> SELECT * FROM XY;

         X          Y
---------- ----------
         1          1
         2          3

2 rows selected.

SQL> SELECT REGR_AVGY(Y,X) FROM XY;

REGR_AVGY(Y,X)
--------------
             2

1 row selected.
```

4.2.128. REGR_SXX

REGR_SXX는 임의의 수치 데이터 쌍의 집합에 가장 맞는 선형 방정식을 구하는 함수로, 집합 함수와 분석 함수로 사용된다.

입력으로 숫자 데이터 또는 숫자 데이터로 변환 가능한 데이터형을 갖는다. 인자로 받은 종속변수(expr1) 또는 독립변수(expr2)의 값이 NULL이면 계산에서 제외되며 모든 로우가 NULL의 쌍이면 NULL을 결과로 낸다.

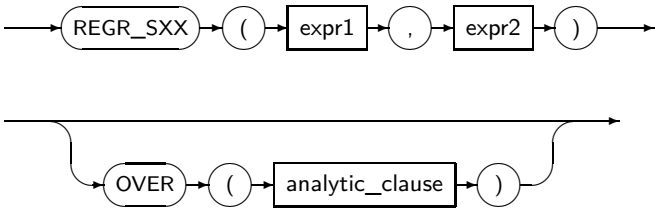
REGR_SXX는 다음의 계산식으로 회귀 분석을 위한 진단 통계를 계산하는데 사용하는 보조 함수이다.

```
REGR_COUNT(expr1, expr2) × VAR_POP(expr2)
```

REGR_SXX의 세부 내용은 다음과 같다.

- 문법

regr_sxx



- 구성요소

구성요소	설명
expr1	수치 값을 반환하는 임의의 연산식이다. 종속변수에 해당한다.
expr2	수치 값을 반환하는 임의의 연산식이다. 독립변수에 해당한다.

- 예제

  다음은 함수를 사용하는 예이다.

```
SQL> SELECT * FROM XY;

         X          Y
---------- ----------
         1          1
         2          3

2 rows selected.

SQL> SELECT REGR_SXX(Y,X) FROM XY;

REGR_SXX(Y,X)
-------------
           .5

1 row selected.
```

4.2.129. REGR_SYY

REGR_SYY는 임의의 수치 데이터 쌍의 집합에 가장 맞는 선형 방정식을 구하는 함수로, 집합 함수와 분석 함수로 사용된다.

입력으로 숫자 데이터 또는 숫자 데이터로 변환 가능한 데이터형을 갖는다. 인자로 받은 종속변수(expr1) 또는 독립변수(expr2)의 값이 NULL이면 계산에서 제외되며 모든 로우가 NULL의 쌍이면 NULL을 결과로 낸다.

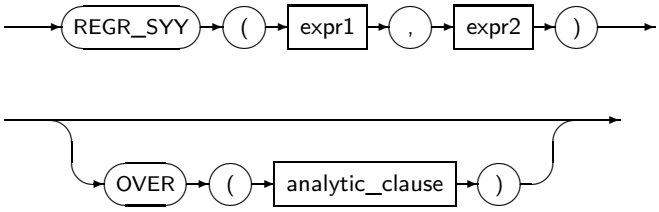
REGR_SYY는 회귀 분석을 위한 진단 통계를 계산하는데 사용하는 보조 함수이다. 계산식은 다음과 같다.

```
REGR_COUNT(expr1, expr2) × VAR_POP(expr1)
```

REGR_SYY의 세부 내용은 다음과 같다.

- 문법

regr_syy



- 구성요소

구성요소	설명
expr1	수치 값을 반환하는 임의의 연산식이다. 종속변수에 해당한다.
expr2	수치 값을 반환하는 임의의 연산식이다. 독립변수에 해당한다.

- 예제

다음은 함수를 사용하는 예이다.

```
SQL> SELECT * FROM XY;

         X          Y
---------- ----------
         1          1
         2          3

2 rows selected.

SQL> SELECT REGR_SYY(Y,X) FROM XY;

REGR_SYY(Y,X)
```

```
-------------
            2

1 row selected.
```

4.2.130. REGR_SXY

REGR_SXY는 임의의 수치 데이터 쌍의 집합에 가장 맞는 선형 방정식을 구하는 함수로, 집합 함수와 분석 함수로 사용된다.

입력으로 숫자 데이터 또는 숫자 데이터로 변환 가능한 데이터형을 갖는다. 인자로 받은 종속변수(expr1) 또는 독립변수(expr2)의 값이 NULL이면 계산에서 제외되며 모든 로우가 NULL의 쌍이면 NULL을 결과로 낸다.

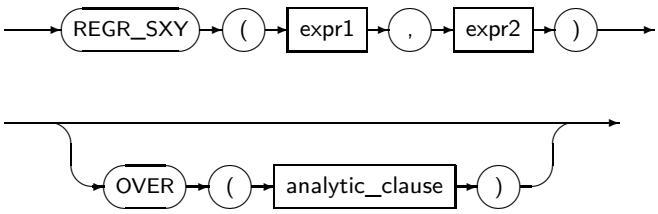
REGR_SXY는 회귀 분석을 위한 진단 통계를 계산하는데 사용하는 보조 함수이다. 계산식은 다음과 같다.

```
REGR_COUNT(expr1, expr2) × COVAR_POP(expr1, expr2)
```

REGR_SXY의 세부 내용은 다음과 같다.

- 문법

regr_sxy



- 구성요소

구성요소	설명
expr1	수치 값을 반환하는 임의의 연산식이다. 종속변수에 해당한다.
expr2	수치 값을 반환하는 임의의 연산식이다. 독립변수에 해당한다.

- 예제

다음은 함수를 사용하는 예이다.

```
SQL> SELECT * FROM XY;

         X          Y
---------- ----------
         1          1
```

```
         2          3

2 rows selected.

SQL> SELECT REGR_SXY(Y,X) FROM XY;

REGR_SXY(Y,X)
-------------
            1

1 row selected.
```

4.2.131. RATIO_TO_REPORT

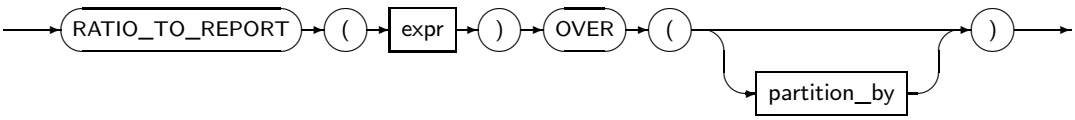
RATIO_TO_REPORT는 값의 집합의 합에 대한 집합 하나의 값의 비율을 계산하는 분석 함수이다. 이 함수는 expr 값이 NULL이면 반환값 역시 NULL이다.

값의 집합은 partition_by에 의해 결정된다. partition_by의 값을 명시하지 않으면 질의 결과로 반환된 모든 행을 대상으로 계산을 실행한다.

RATIO_TO_REPORT의 세부 내용은 다음과 같다.

- 문법

ratio_to_report



- 구성요소

구성요소	설명
expr	expr은 수치 값을 반환하는 임의의 연산식이다. expr에 RATIO_TO_REPORT 함수를 포함한 다른 분석 함수를 명시할 수 없다. 즉, 분석 함수를 중첩할 수 없다. 그러나 다른 내장 함수는 expr에 명시할 수 있다.
partition_by	현재 질의 블록의 결과 집합을 expr 또는 expr의 리스트를 기준으로 분할한다. 자세한 내용은 "4.1.3. 분석 함수"의 partition_by를 참고한다.

- 예제

다음은 RATIO_TO_REPORT 함수를 사용하는 예이다.

```
SQL> SELECT TYPE, AMOUNT, RATIO_TO_REPORT(AMOUNT) OVER () RATIO
     FROM ASSETS;

TYPE                     AMOUNT      RATIO
-------------------- ---------- ----------
funds                       200         .1
stock                       500        .25
real_estate                1000         .5
cash                        300        .15

4 rows selected.
```

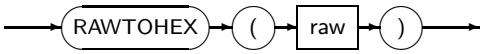
4.2.132. RAWTOHEX

RAWTOHEX는 문자열 raw 값을 16진수로 표현된 VARCHAR2 타입의 문자열로 바꾸는 함수이다.

RAWTOHEX의 세부 내용은 다음과 같다.

- 문법

rawtohex



- 구성요소

구성요소	설명
raw	문자열 값을 반환하는 임의의 연산식이다. 단, CLOB, BLOB, LONG, LONG RAW, XMLTYPE, GEOMETRY 등의 타입은 사용할 수 없다.

- 예제

다음은 RAWTOHEX 함수를 사용하는 예이다.

```
SQL> SELECT RAWTOHEX('AB') FROM DUAL;

RAWTOHEX('AB')
--------------
4142

1 row selected.
```

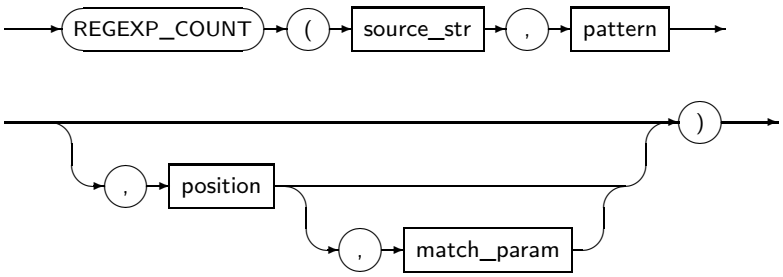
4.2.133. REGEXP_COUNT

REGEXP_COUNT는 입력 문자열 내에서 정규 표현식으로 주어진 패턴이 몇 번이나 일치하는지 반환하는 함수이다(이 기능은 ICU 정규 표현식 표준을 따른다).

REGEXP_COUNT의 세부 내용은 다음과 같다.

- 문법

regexp_count



- 구성요소

구성요소	설명
source_str	문자열을 반환하는 임의의 연산식이다. CHAR, VARCHAR2, NCHAR, NVARCHAR2, CLOB, NCLOB 타입을 사용할 수 있다.
pattern	정규 표현식으로 작성된 문자열을 반환하는 임의의 연산식이다. CHAR, VARCHAR2, NCHAR, NVARCHAR2 타입을 사용할 수 있고, 만약 source_str과 타입이 다를 경우 source_str 타입으로 변환된다.
position	숫자값을 반환하는 임의의 연산식으로 패턴검사를 시작할 위치를 지정한다. (기본값: 1)
match_param	문자열을 반환하는 임의의 연산식으로 패턴을 검사할 방법을 설정한다. 다음과 같은 값을 사용할 수 있고, 여러 개를 동시에 지정할 수 있다. – 'i' : 대소문자를 구분하지 않는다. – 'c' : 대소문자를 구분한다. – 'n' : 점(.)이 줄바꿈 문자도 포함한다. – 'm' : 입력문자열이 한줄 이상이다. – 'x' : pattern의 공백문자를 무시한다.

구성요소	설명
	예를 들어 'ic'와 같이 상호충돌하는 값을 지정하였을 경우엔 마지막 값만 사용한다. 즉, 'ic'는 대소문자를 구분한다.

- 예제

  다음은 REGEXP_COUNT 함수를 사용하는 예이다.

```
SQL> SELECT REGEXP_COUNT('abcabcabc','abc', 2) FROM DUAL;

REGEXP_COUNT('ABCABCABC','ABC',2)
---------------------------------
                                2

1 row selected.
```

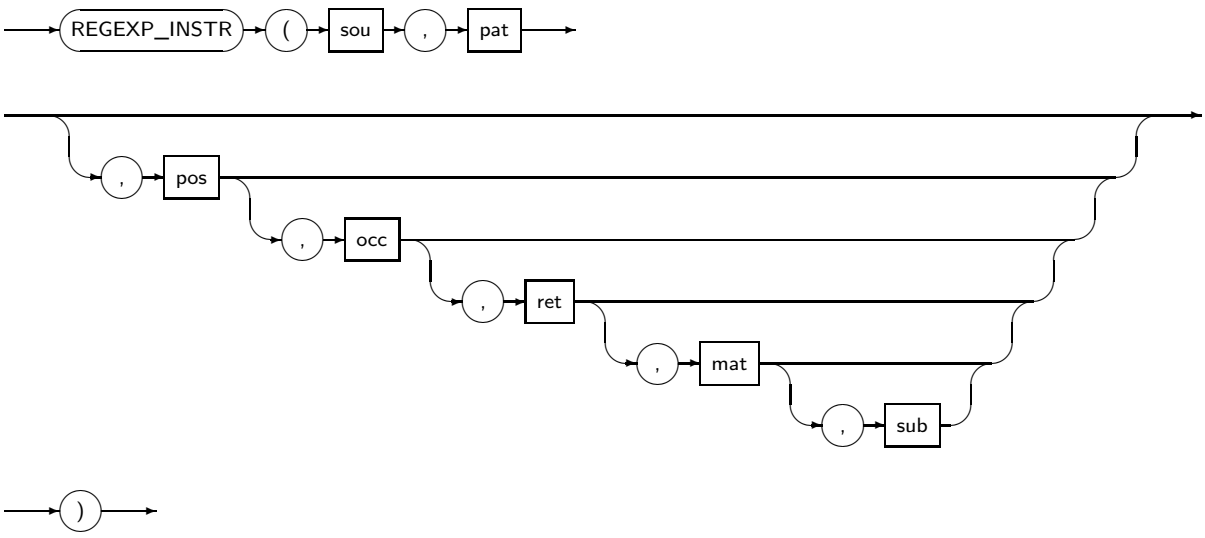
4.2.134. REGEXP_INSTR

REGEXP_INSTR는 입력 문자열 내에서 정규 표현식으로 주어진 패턴이 일치하는지 위치를 반환하는 함수이다. 만약 입력 문자열과 일치하지 않으면 0을 반환한다(이 기능은 ICU 정규 표현식 표준을 따른다).

REGEXP_INSTR의 세부 내용은 다음과 같다.

- 문법

regexp_instr



- 구성요소

구성요소	설명
sou (source_str)	문자열을 반환하는 임의의 연산식이다. CHAR, VARCHAR2, NCHAR, NVARCHAR2, CLOB, NCLOB 타입을 사용할 수 있다.
pat (pattern)	정규 표현식으로 작성된 문자열을 반환하는 임의의 연산식이다. CHAR, VARCHAR2, NCHAR, NVARCHAR2 타입을 사용할 수 있고, 만약 source_str과 타입이 다를 경우 source_str 타입으로 변환된다.
pos (position)	숫자값을 반환하는 임의의 연산식으로 패턴검사를 시작할 위치를 지정한다. (기본값: 1)
occ (occurrence)	숫자값을 반환하는 임의의 연산식으로 패턴을 몇 번 검사할지를 지정한다. (기본값: 1)
ret (return_option)	숫자값을 반환하는 임의의 연산식이다. – 0 : 패턴과 일치하는 문자열의 처음 위치를 반환한다. (기본값) – 1 : 패턴과 일치하는 문자열 다음 문자의 위치를 반환한다.
mat (match_param)	문자열을 반환하는 임의의 연산식으로 패턴을 어떤 방법으로 검사할지를 지정한다. REGEXP_COUNT 함수와 같은 특성을 가진다.
sub (sub_expr)	숫자값을 반환하는 임의의 연산식으로 0부터 9까지 사용할 수 있다. (기본값: 0) sub_expr은 pattern에서 괄호로 감싸진 각 그룹들을 왼쪽부터 숫자로 지정한다. 예를 들어 다음과 같은 정규표현식이 주어졌을 때 모두 4개의 그룹이 존재한다. `(tibero(is(a)(rdbms)))` 왼쪽부터 "tiberoisardbms", "isardbms", "a", "rdbms"이며 각 1, 2, 3, 4에 해당된다.

- 예제

  다음은 REGEXP_INSTR 함수를 사용하는 예이다.

```
SQL> SELECT REGEXP_INSTR('abcabcabc','abc', 2) FROM DUAL;

REGEXP_INSTR('ABCABCABC','ABC',2)
---------------------------------
                                4

1 row selected.
```

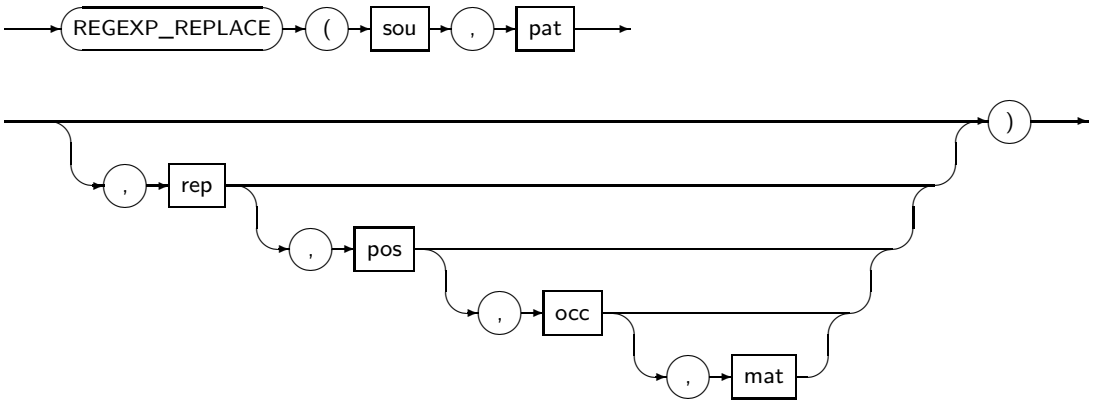
4.2.135. REGEXP_REPLACE

REGEXP_REPLACE는 입력 문자열내에서 정규 표현식으로 주어진 패턴을 탐색하여 다른 문자열로 대치하는 함수이다(이 기능은 ICU 정규 표현식 표준을 따른다). 결과 타입은 첫 번째 인자의 타입과 동일하다.

REGEXP_REPLACE의 세부 내용은 다음과 같다.

- 문법

regexp_replace



- 구성요소

구성요소	설명
sou (source_str)	문자열을 반환하는 임의의 연산식이다. CHAR, VARCHAR2, NCHAR, NVARCHAR2, CLOB, NCLOB 타입을 사용할 수 있다.
pat (pattern)	정규 표현식으로 작성된 문자열을 반환하는 임의의 연산식이다. CHAR, VARCHAR2, NCHAR, NVARCHAR2 타입을 사용할 수 있고, 만약 source_str과 타입이 다를 경우 source_str 타입으로 변환된다.
rep (replace_str)	문자열을 반환하는 임의의 연산식이다. CHAR, VARCHAR2, NCHAR, NVARCHAR2 타입을 사용할 수 있고, 만약 source_str과 타입이 다를 경우 source_str 타입으로 변환된다.
pos (position)	숫자값을 반환하는 임의의 연산식으로 패턴검사를 시작할 위치를 지정한다. (기본값: 1)
occ (occurrence)	숫자값을 반환하는 임의의 연산식으로 패턴을 몇 번 검사할지를 지정한다. (기본값: 1)

구성요소	설명
mat (match_param)	문자열을 반환하는 임의의 연산식으로 패턴을 어떤 방법으로 검사할지를 지정한다. REGEXP_COUNT 함수와 같은 특성을 가진다.

- 예제

  다음은 REGEXP_REPLACE 함수를 사용하는 예이다.

```
SQL> SELECT REGEXP_REPLACE('aaaaaaa','([[:alpha:]])', 'x') FROM DUAL;

REGEXP_REPLACE('AAAAAAA','([[:ALPHA:]])','X')
---------------------------------------------
xxxxxxx

1 row selected.
```

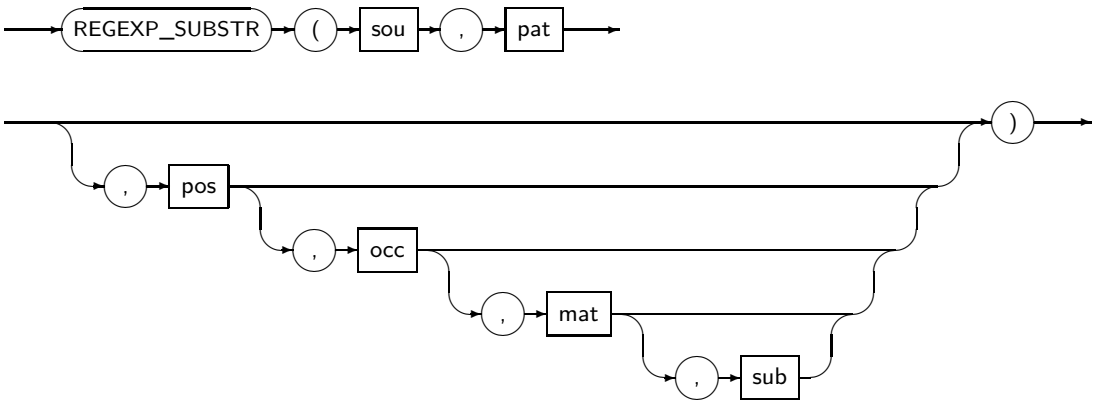
4.2.136. REGEXP_SUBSTR

REGEXP_SUBSTR는 입력 문자열내에서 정규 표현식으로 주어진 패턴을 탐색하여 해당 문자열을 반환하는 함수이다(이 기능은 ICU 정규 표현식 표준을 따른다). 결과 타입은 첫 번째 인자의 타입과 동일하다.

REGEXP_SUBSTR의 세부 내용은 다음과 같다.

- 문법

regexp_substr



- 구성요소

구성요소	설명
sou	문자열을 반환하는 임의의 연산식이다.

구성요소	설명
(source_str)	CHAR, VARCHAR2, NCHAR, NVARCHAR2, CLOB, NCLOB 타입을 사용할 수 있다.
pat (pattern)	정규 표현식으로 작성된 문자열을 반환하는 임의의 연산식이다. CHAR, VARCHAR2, NCHAR, NVARCHAR2 타입을 사용할 수 있고, 만약 source_str과 타입이 다를 경우 source_str 타입으로 변환된다.
pos (position)	숫자값을 반환하는 임의의 연산식으로 패턴검사를 시작할 위치를 지정한다. (기본값: 1)
occ (occurrence)	숫자값을 반환하는 임의의 연산식으로 패턴을 몇 번 검사할지를 지정한다. (기본값: 1)
mat (match_param)	문자열을 반환하는 임의의 연산식으로 패턴을 어떤 방법으로 검사할지를 지정한다. REGEXP_COUNT 함수와 같은 특성을 가진다.
sub (sub_expr)	숫자값을 반환하는 임의의 연산식으로 0부터 9까지 사용할 수 있다. (기본값: 0) REGEXP_INSTR 함수와 같은 특성을 가진다.

- 예제

  다음은 REGEXP_SUBSTR 함수를 사용하는 예이다.

```
SQL> SELECT REGEXP_SUBSTR('123456','3.*5', 1) FROM DUAL;

REGEXP_SUBSTR('123456','3.*5',1)
--------------------------------
345

1 row selected.
```

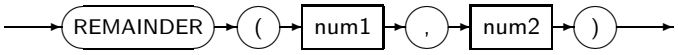
4.2.137. REMAINDER

REMAINDER은 num1을 num2로 나눈 나머지를 반환하는 함수이다.

REMAINDER의 세부 내용은 다음과 같다.

- 문법

remainder



- 구성요소

구성요소	설명
num1, num2	num1이나 num2는 숫자형 타입이거나 숫자형 타입으로 변환될 수 있는 타입이어야 한다. num1이나 num2은 두 타입 중 숫자형 우선순위에 따라 더 높은 순위의 타입으로 변환되고, 또한 그 타입으로 반환된다.

- 예제

다음은 REMAINDER 함수를 사용하는 예이다.

```
SQL> SELECT REMAINDER(3, 2), REMAINDER(7F, 3), REMAINDER(5.5D, 1.3F) FROM DUAL;

REMAINDER(3,2) REMAINDER(7F,3) REMAINDER(5.5D,1.4F)
-------------- --------------- --------------------
            -1        1.0E+000              3.0E-001

1 rows selected.
```

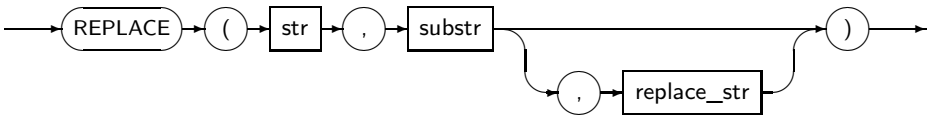
4.2.138. REPLACE

REPLACE는 문자열 str 내에서 문자열 substr을 탐색하여 문자열 replace_str로 치환하는 함수이다.

REPLACE의 세부 내용은 다음과 같다.

- 문법

replace



- 구성요소

구성요소	설명
str	문자열을 반환하는 임의의 연산식이다.
substr	문자열을 반환하는 임의의 연산식이다. 만약 substr이 NULL이면 str을 그대로 반환한다.

구성요소	설명
replace_str	문자열을 반환하는 임의의 연산식이다. 만약 replace_str이 NULL이거나 지정되지 않으면, str 내의 모든 substr은 제거된다.

- 예제

  다음은 REPLACE 함수를 사용하는 예이다.

```
SQL> SELECT REPLACE('ABCDEFG', 'CD') FROM DUAL;

REPLACE('ABCDEFG','CD')
-----------------------
ABEFG

1 row selected.

SQL> SELECT REPLACE('ABCDEFG', 'CD', 'XY') FROM DUAL;

REPLACE('ABCDEFG','CD','XY')
----------------------------
ABXYEFG

1 row selected.
```

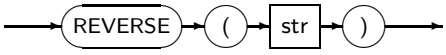
4.2.139. REVERSE

REVERSE는 주어진 문자열을 거꾸로 출력하는 함수이다.

REVERSE의 세부 내용은 다음과 같다.

- 문법

  reverse



- 구성요소

구성요소	설명
str	문자열을 반환하는 임의의 연산식이다.

- 예제

  다음은 REVERSE 함수를 사용하는 예이다.

```
SQL> SELECT REVERSE('TIBERO') FROM DUAL;

REVERSE('TIBERO')
-----------------
OREBIT

1 row selected.
```

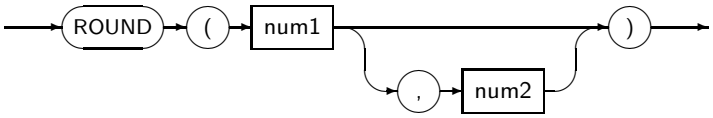
4.2.140. ROUND(number)

ROUND(number)는 num1을 소수점 아래 num2+1 위치에서 반올림한 값을 반환하는 함수이다.

ROUND(number)의 세부 내용은 다음과 같다.

- 문법

round_num



- 구성요소

구성요소	설명
num1	수치 값을 반환하는 임의의 연산식이다.
num2	수치 값을 반환하는 임의의 연산식이다. num2는 정수가 되어야 한다. 만약 num2가 지정되지 않았거나 0이면 소수점 첫째 자리에서 반올림한다. num2가 음수이면 소수점 위의 자리에서 반올림한다.

- 예제

다음은 ROUND(number) 함수를 사용하는 예이다.

```
SQL> SELECT ROUND(345.678), ROUND(345.678, 2), ROUND(345.678, -1)
     FROM DUAL;

ROUND(345.678) ROUND(345.678,2) ROUND(345.678,-1)
-------------- ---------------- -----------------
           346           345.68               350

1 row selected.
```

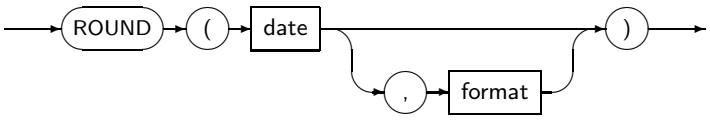
4.2.141. ROUND(date)

ROUND(date)는 date를 format에 명시된 단위로 반올림한 결과를 반환하는 함수이다.

ROUND(date)의 세부 내용은 다음과 같다.

- 문법

round_date



- 구성요소

구성요소	설명
date	날짜를 반환하는 임의의 연산식이다.
format	반올림 단위를 명시하는 포맷 모델을 나타내는 문자열이다. format이 명시되지 않으면 'DD' 형식의 문자열을 이용하여 가장 가까운 날로 반올림한다. format으로 'YEAR', 'MONTH', 'DAY' 등을 사용할 수 있다. **ROUND(date)** 함수의 format에 명시할 수 있는 형식 문자열은 다음과 같다. – CC, SCC : year 4자리 중 아래 2자리로 반올림/버림 후 1년 큰 값이다. (예: xx01년) – IYYY, IYY, IY, I : ISO year 값에 대해 7월 1일 전/후로 반올림 또는 버림을 한다. – SYYYY, YYYY, YYY, YY, Y, YEAR, SYEAR : year 값에 대해 7월 1일 전/후로 반올림 또는 버림을 한다. – Q : 분기 값에 대해 각 분기의 두 번째 달의 16번째 날 전/후로 반올림 또는 버림을 한다. – MONTH, MON, MM, RM : month 값에 대해 16번째 날 전/후로 반올림 또는 버림을 한다. – WW : 그 해의 첫 번째 날(1월 1일)을 한 주의 시작으로 해서, week 값에 대해 반올림 또는 버림을 한다. – IW : 그 주의 월요일을 한 주의 시작으로 해서, week 값에 대해 반올림 또는 버림을 한다. – W : 그 달의 첫 번째 날(1일)과 같은 요일이 되도록 week 값에 대해 반올림 또는 버림을 한다.

구성요소	설명
	– DDD, DD, J : day 값에 대해 오후 12시 전/후로 반올림 또는 버림을 한다. – DAY, DY, D : week 값에 대해 수요일 오후 12시 전/후로 반올림 또는 버림을 한다. – HH24, HH12, HH : hour 값에 대해 30분 전/후로 반올림 또는 버림을 한다. – MI : minute 값에 대해 30초 전/후로 반올림 또는 버림을 한다.

- 예제

다음은 ROUND(date) 함수를 사용하는 예이다.

```
SQL> SELECT ROUND(TO_DATE('2005/06/22'), 'YEAR') AS TO_DATE
     FROM DUAL;

TO_DATE
--------------------------------
2005/01/01

1 row selected.

SQL> SELECT TO_CHAR(ROUND(TO_DATE('1998/6/20',
       'YYYY/MM/DD'), 'CC'), 'YYYY/MM/DD') AS TO_DATE
     FROM DUAL;

TO_DATE
----------
2001/01/01

1 row selected.

SQL> SELECT TO_CHAR(ROUND(TO_DATE('-3741/01/02',
       'SYYYY/MM/DD'), 'CC'), 'SYYYY/MM/DD') AS TO_DATE
     FROM DUAL;

TO_DATE
-----------
-3700/01/01

1 row selected.

SQL> SELECT TO_CHAR(ROUND(TO_DATE('2005/01/26 12:30:14',
       'YYYY/MM/DD HH24:MI:SS'), 'DY'), 'YYYY/MM/DD') AS TO_DATE
     FROM DUAL;
```

```
TO_DATE
----------
2005/01/30

1 row selected.

SQL> SELECT ROUND(TO_DATE('2005/01/26 12:30:14',
       'YYYY/MM/DD HH24:MI:SS')) AS TO_DATE
     FROM DUAL;

TO_DATE
-------------------------------
2005/01/27

1 row selected.
```

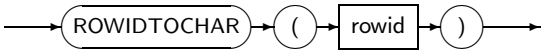
4.2.142. ROWIDTOCHAR

ROWIDTOCHAR는 ROWID의 값을 VARCHAR 타입의 값으로 변환하는 함수이다.

ROWIDTOCHAR의 세부 내용은 다음과 같다.

- 문법

rowidtochar



- 구성요소

구성요소	설명
rowid	VARCHAR 타입으로 변환할 값이다.

- 예제

다음은 ROWIDTOCHAR 함수를 사용하는 예이다.

```
SQL> SELECT LAST_NAME
     FROM EMP
     WHERE ROWIDTOCHAR(ROWID) LIKE "%AAAF%';

LAST_NAME
----------
King
```

```
1 row selected.
```

4.2.143. ROW_NUMBER

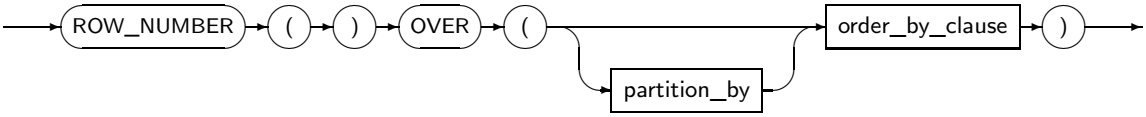
ROW_NUMBER는 함수가 적용된 모든 로우(파티션의 모든 로우 또는 질의에 의해 반환되는 모든 로우)에 대하여 order_by_clause에 명시된 순서대로 1부터 시작되는 유일한 번호를 부여하는 분석 함수이다.

ROW_NUMBER 함수를 사용하여 top-N, bottom-N, inner-N 보고를 구현할 수 있다. 일관된 결과를 얻기 위해서는 질의는 결정된 정렬 순서를 보장하여야 한다.

ROW_NUMBER의 세부 내용은 다음과 같다.

- 문법

row_number



- 구성요소

구성요소	설명
partition_by	현재 질의 블록의 결과 집합을 expr 또는 expr의 리스트를 기준으로 분할한다. 자세한 내용은 "4.1.3. 분석 함수"의 partition_by를 참고한다.
order_by_clause	분할된 하나의 파티션 내에서 로우를 어떻게 정렬할지를 명시한다. 자세한 내용은 "4.1.3. 분석 함수"의 order_by_clause를 참고한다.

- 예제

다음은 ROW_NUMBER 함수를 사용하는 예이다.

```
SQL> SELECT JOB_ID, LAST_NAME, SALARY
     FROM
     (
       SELECT JOB_ID, LAST_NAME, SALARY,
         ROW_NUMBER() OVER (PARTITION BY JOB_ID ORDER BY SALARY) RN
       FROM EMPLOYEES_DEMO
     )
     WHERE RN <= 1;

JOB_ID     LAST_NAME                      SALARY
---------- ------------------------- ----------
AD_PRES    King                           24000
```

```
AD_VP      Kochhar                        17000
IT_PROG    Ernst                           6000

3 rows selected.
```

4.2.144. RPAD

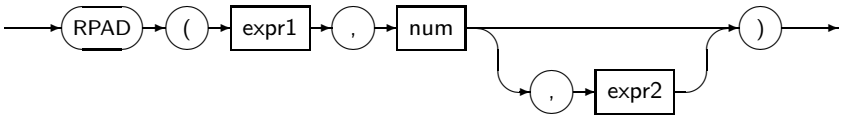
RPAD는 expr2를 expr1의 오른쪽에 반복적으로 붙인 길이가 num인 문자열을 구하는 함수이다.

대부분의 문자 집합에서는 반환되는 문자열의 문자의 수와 길이가 동일하지만, 한글과 같은 멀티 바이트 문자 집합의 경우에는 두 값이 다를 수 있다.

RPAD의 세부 내용은 다음과 같다.

- 문법

rpad



- 구성요소

구성요소	설명
expr1	문자열 또는 CLOB 타입, BLOB 타입을 반환하는 임의의 연산식이다. expr1의 길이가 num 보다 크면, expr1에서 왼쪽부터 num만큼의 문자열을 반환한다.
expr2	문자열 또는 CLOB 타입, BLOB 타입을 반환하는 임의의 연산식이다. expr2가 명시되지 않으면 공백 문자가 사용된다.
num	수치 값을 반환하는 임의의 연산식이다. num은 터미널에 출력되는 길이를 의미한다.

- 예제

다음은 RPAD 함수를 사용하는 예이다.

```
SQL> SELECT RPAD('RPAD', 10, '-=') FROM DUAL;

RPAD('RPAD',10,'-=')
--------------------
```

```
RPAD-=-=-=

1 row selected.
```

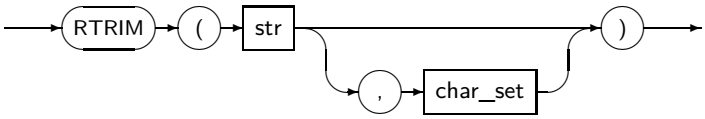
4.2.145. RTRIM

RTRIM은 문자열 str의 오른쪽으로부터 문자열 char_set 내에 포함된 모든 문자를 제거하는 함수이다.

RTRIM의 세부 내용은 다음과 같다.

- 문법

rtrim



- 구성요소

구성요소	설명
str	문자열을 반환하는 임의의 연산식이다.
char_set	문자열을 반환하는 임의의 연산식이다. 만약 파라미터 char_set의 값이 없으면, 디폴트 값으로 공백 문자 하나를 갖는다.

- 예제

다음은 RTRIM 함수를 사용하는 예이다.

```
SQL> SELECT RTRIM('ABCDE ') FROM DUAL;

RTRIM('ABCDE')
--------------
ABCDE

1 row selected.

SQL> SELECT RTRIM('XYXABCDEXYX', 'XY') FROM DUAL;

RTRIM('XYXABCDEXYX','XY')
-------------------------
XYXABCDE

1 row selected.
```

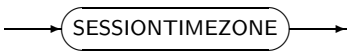
4.2.146. SESSIONTIMEZONE

SESSIONTIMEZONE는 현재 세션의 시간대를 출력하는 함수이다.

SESSIONTIMEZONE의 세부 내용은 다음과 같다.

- 문법

  sessiontimezone

- 예제

  다음은 SESSIONTIMEZONE 함수를 사용하는 예이다.

```
SQL> SELECT SESSIONTIMEZONE FROM DUAL;

SESSIONTIMEZONE
-------------------------
Asia/Seoul

1 row selected.
```

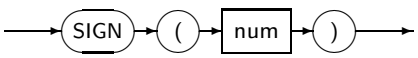
4.2.147. SIGN

SIGN은 num이 음수이면 -1을 반환하고, 0이면 0을 양수이면 +1을 반환하는 함수이다.

SIGN의 세부 내용은 다음과 같다.

- 문법

  sign

- 구성요소

구성요소	설명
num	수치 값을 반환하는 임의의 연산식이다. num은 숫자형 타입이거나 숫자형 타입으로 변환될 수 있는 타입이어야 하고, 반환 타입은 NUMBER이다. – num이 NUMBER 타입인 경우 반환값

구성요소	설명
	• 0보다 작은 경우 : -1 • 0인 경우 : 0 • 0보다 큰 경우 : 1 – num이 BINARY_FLOAT이거나 BINARY_DOUBLE 타입인 경우 반환값 • 0보다 작은 경우 : -1 • 0보다 같거나 큰 경우 또는 NaN(Not A Number)인 경우 : 1

- 예제

  다음은 SIGN 함수를 사용하는 예이다.

```
SQL> SELECT SIGN(-10), SIGN(0), SIGN(15.5) FROM DUAL;

 SIGN(-10)    SIGN(0) SIGN(15.5)
---------- ---------- ----------
        -1          0          1

1 row selected.
```

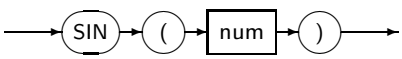
4.2.148. SIN

SIN은 주어진 파라미터 값의 사인(sine) 값을 구한다.

SIN의 세부 내용은 다음과 같다.

- 문법

  sin



- 구성요소

구성요소	설명
num	실수 값을 반환하는 임의의 연산식이다. (단위: 라디안) num은 숫자형 타입이거나 숫자형 타입으로 변환될 수 있는 타입이어야 한다. 만약 num의 타입이 BINARY_FLOAT일 경우엔 BINARY_DOUBLE 타입을 반환하고, 그렇지 않은 경우엔 num과 동일한 숫자형 타입을 반환한다.

- 예제

  다음은 SIN 함수를 사용하는 예이다.

```
SQL> SELECT SIN(3.141592654 / 2.0) FROM DUAL;

SIN(3.141592654/2.0)
--------------------
                   1

1 row selected.
```

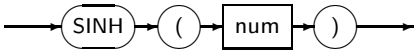
4.2.149. SINH

SINH은 주어진 파라미터 값의 하이퍼볼릭 사인 값을 구하는 함수이다.

SINH의 세부 내용은 다음과 같다.

- 문법

  sinh



- 구성요소

구성요소	설명
num	실수 값을 반환하는 임의의 연산식이다. (단위: 라디안) num은 숫자형 타입이거나 숫자형 타입으로 변환될 수 있는 타입이어야 한다. 만약 num의 타입이 BINARY_FLOAT일 경우엔 BINARY_DOUBLE 타입을 반환하고, 그렇지 않은 경우엔 num과 동일한 숫자형 타입을 반환한다.

- 예제

  다음은 SINH 함수를 사용하는 예이다.

```
SQL> SELECT SINH(1) FROM DUAL;

   SINH(1)
----------
1.17520119

1 row selected.
```

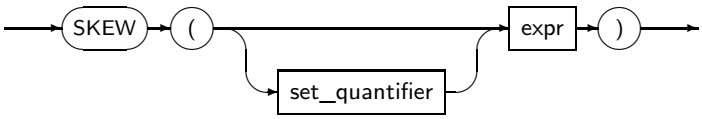
4.2.150. SKEW

SKEW는 expr의 왜도를 반환하는 함수이다. 이 함수는 모든 수치 데이터 타입과 수치 데이터 타입은 아니지만 묵시적으로 수치 데이터 타입으로 변환할 수 있는 데이터 타입을 파라미터로 받아 들인다. 입력된 파라미터의 수치 데이터 타입 또는 변환된 수치 데이터 타입을 반환한다.

SKEW의 세부 내용은 다음과 같다.

- 문법

skew



- 구성요소

구성요소	설명
set_quantifier	질의 결과에 중복된 로우의 허용, 비허용 여부를 지정한다. DISTINCT, UNIQUE, ALL을 지정할 수 있다. – DISTINCT, UNIQUE : 중복된 로우를 제거한다. – ALL : 모든 로우를 선택한다. (기본값)
expr	측정할 연산식을 명시한다.

- 예제

다음은 SKEW 함수를 사용하는 예이다.

```
SQL> SELECT SKEW(SAL) FROM EMP;

 SKEW(SAL)
----------
1.17469474

1 row selected.
```

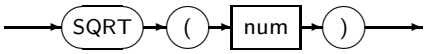
4.2.151. SQRT

SQRT는 주어진 파라미터 값의 제곱근(Square Root)을 구하는 함수이다.

SQRT의 세부 내용은 다음과 같다.

- 문법

sqrt



- 구성요소

구성요소	설명
num	num은 숫자형 타입이거나 숫자형 타입으로 변환될 수 있는 타입이어야 하고, num과 동일한 타입을 반환한다. num이 NUMBER 타입인 경우 num은 음수가 될 수 없다. – num이 BINARY_FLOAT이거나 BINARY_DOUBLE 타입인 경우 반환값 • num이 0보다 크거나 같은 경우 양의 실수 • num이 0보다 작은 경우 NaN(Not A Number)

- 예제

다음은 SQRT 함수를 사용하는 예이다.

```
SQL> SELECT SQRT(2.0) FROM DUAL;

 SQRT(2.0)
----------
1.41421356

1 row selected.
```

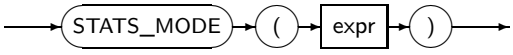
4.2.152. STATS_MODE

STATS_MODE는 expr의 최빈값을 반환하는 함수이다. 만약 최빈값이 여러 개이면 그 중 한 개의 값을 반환한다. NULL은 최빈값이 될 수 없으나 모든 값이 NULL이면 NULL을 반환한다.

STATS_MODE의 세부 내용은 다음과 같다.

- 문법

stats_mode



- 구성요소

구성요소	설명
expr	수치 값을 반환하는 임의의 연산식이다.

- 예제

다음은 STATS_MODE 함수를 사용하는 예이다.

```
SQL> SELECT STATS_MODE(SALARY) FROM EMP;

STATS_MODE(SALARY)
------------------
             15000

1 row selected.
```

4.2.153. STDDEV

STDDEV는 expr의 표본 표준편차를 반환하는 함수이다. 분석 함수로도 사용할 수 있다. Tibero는 표준편차의 값을 VARIANCE 집합 함수 값의 제곱근으로 계산한다.

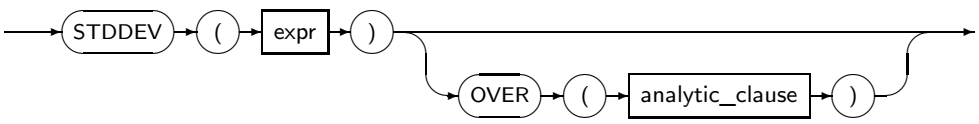
이 함수는 모든 수치 데이터 타입과 수치 데이터 타입은 아니지만 묵시적으로 수치 데이터 타입으로 변환할 수 있는 데이터 타입을 파라미터로 받아 들인다. 입력된 파라미터의 수치 데이터 타입 또는 변환된 수치 데이터 타입을 반환한다.

STDDEV_SAMP 함수와 STDDEV 함수의 차이점은 입력 데이터의 크기가 단 하나의 로우일 경우 STDDEV 함수는 0을 반환하는 반면 STDDEV_SAMP 함수는 NULL 값을 반환한다는 것이다.

STDDEV의 세부 내용은 다음과 같다.

- 문법

stddev



- 구성요소

구성요소	설명
expr	수치 값을 반환하는 임의의 연산식이다.
analytic_clause	OVER analytic_clause를 사용해 함수를 분석 함수로 수행할 수 있다. 자세한 내용은 “4.1.3. 분석 함수”의 analytic_clause를 참고한다.

- 예제

다음은 STDDEV 함수를 사용하는 예이다.

```
SQL> SELECT STDDEV(AGE) FROM EMP_AGE;

STDDEV(AGE)
-----------
3.034981237

1 row selected.
```

- 분석 함수 예제

다음은 STDDEV 함수를 분석 함수로 사용하는 예이다.

```
SQL> SELECT DEPTNO, EMPNO, STDDEV(SAL)
       OVER (PARTITION BY DEPTNO) AS STDDEV
     FROM EMP;

    DEPTNO      EMPNO     STDDEV
---------- ---------- ----------
        10       7934 1893.62967
        10       7839 1893.62967
        10       7782 1893.62967
        20       7566  1123.3321
        20       7788  1123.3321
        20       7876  1123.3321
        20       7902  1123.3321
        20       7369  1123.3321
        30       7654 668.331255
        30       7698 668.331255
        30       7521 668.331255
        30       7499 668.331255
        30       7844 668.331255
        30       7900 668.331255

14 rows selected.
```

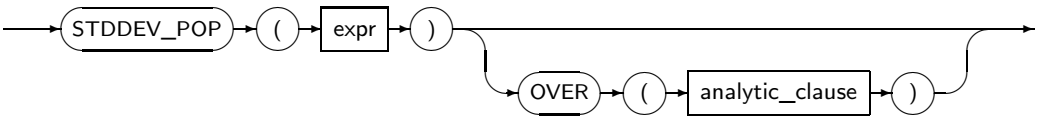
4.2.154. STDDEV_POP

STDDEV_POP는 expr의 모표준편차를 반환하는 함수이다. 분석 함수로도 사용할 수 있다. Tibero는 모표준편차의 값을 VARIANCE_POP 집합 함수 값의 제곱근으로 계산한다. 이 함수는 파라미터의 수치 데이터 타입 또는 변환된 수치 데이터 타입을 반환한다.

STDDEV_POP의 세부 내용은 다음과 같다.

- 문법

stddev_pop



- 구성요소

구성요소	설명
expr	수치 데이터 타입 또는 수치 데이터 타입은 아니지만 묵시적으로 수치 데이터 타입으로 변환할 수 있는 데이터 타입을 갖는 임의의 연산식이다.
analytic_clause	OVER analytic_clause를 사용해 함수를 분석 함수로 수행할 수 있다. 자세한 내용은 “4.1.3. 분석 함수”의 analytic_clause를 참고한다.

- 예제

다음은 STDDEV_POP 함수를 사용하는 예이다.

```
SQL> SELECT STDDEV_POP(AGE) FROM EMP_AGE;

STDDEV_POP(AGE)
---------------
2.8792360097776

1 row selected.
```

- 분석 함수 예제

다음은 STDDEV_POP 함수를 분석 함수로 사용하는 예이다.

```
SQL> SELECT DEPTNO, EMPNO, STDDEV_POP(SAL)
       OVER (PARTITION BY DEPTNO) AS STDDEV_POP
     FROM EMP;

    DEPTNO      EMPNO STDDEV_POP
---------- ---------- ----------
        10       7934 1546.14215
        10       7839 1546.14215
        10       7782 1546.14215
        20       7566 1004.73877
        20       7788 1004.73877
        20       7876 1004.73877
        20       7902 1004.73877
        20       7369 1004.73877
        30       7654 610.100174
        30       7698 610.100174
        30       7521 610.100174
        30       7499 610.100174
```

```
        30       7844 610.100174
        30       7900 610.100174

14 rows selected.
```

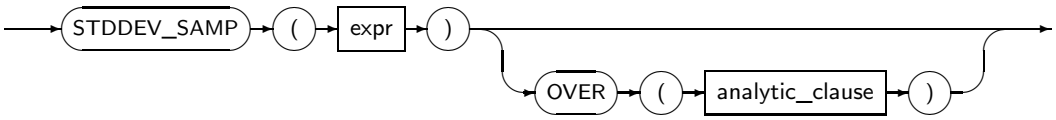
4.2.155. STDDEV_SAMP

STDDEV_SAMP는 expr의 누적 표본 표준편차를 반환하는 함수이다. 분석 함수로도 사용할 수 있다. Tibero는 누적 표본 표준편차의 값을 VARIANCE_SAMP 집합 함수 값의 제곱근으로 계산한다. 이 함수는 파라미터의 수치 데이터 타입 또는 변환된 수치 데이터 타입을 반환한다.

STDDEV_SAMP의 세부 내용은 다음과 같다.

- 문법

stddev_samp



- 구성요소

구성요소	설명
expr	수치 데이터 타입 또는 수치 데이터 타입은 아니지만 묵시적으로 수치 데이터 타입으로 변환할 수 있는 데이터 타입을 갖는 임의의 연산식이다.
analytic_clause	OVER analytic_clause를 사용해 함수를 분석 함수로 수행할 수 있다. 자세한 내용은 “4.1.3. 분석 함수”의 analytic_clause를 참고한다.

- 예제

다음은 STDDEV_SAMP 함수를 사용하는 예이다.

```
SQL> SELECT STDDEV_SAMP(AGE) FROM EMP_AGE;

STDDEV_SAMP(AGE)
----------------
3.03498123735734

1 row selected.
```

- 분석 함수 예제

다음은 STDDEV_SAMP 함수를 분석 함수로 사용하는 예이다.

```
SQL> SELECT DEPTNO, EMPNO, STDDEV_SAMP(SAL)
       OVER (PARTITION BY DEPTNO) AS STDDEV_SAMP
     FROM EMP;

    DEPTNO      EMPNO STDDEV_SAMP
---------- ---------- -----------
        10       7782  1893.62967
        10       7839  1893.62967
        10       7934  1893.62967
        20       7566   1123.3321
        20       7902   1123.3321
        20       7876   1123.3321
        20       7369   1123.3321
        20       7788   1123.3321
        30       7521  668.331255
        30       7844  668.331255
        30       7499  668.331255
        30       7900  668.331255
        30       7698  668.331255
        30       7654  668.331255

14 rows selected.
```

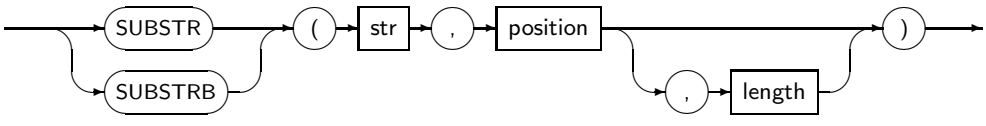
4.2.156. SUBSTR

SUBSTR은 문자열 str 내의 position 위치로부터 length 길이의 문자열을 추출하여 반환하는 함수이다. 추가로 SUBSTRB 함수는 SUBSTR 함수와 동일하지만, 위치를 계산할 때 문자가 아닌 byte를 단위로 사용한다.

SUBSTR의 세부 내용은 다음과 같다.

- 문법

substr



- 구성요소

구성요소	설명
str	문자열을 반환하는 임의의 연산식이다. 문자열 내의 문자의 위치는 1부터 시작된다.

구성요소	설명
position	정수 값을 반환하는 임의의 연산식이다. 문자열 내의 문자의 위치는 1부터 시작된다. 만약 position이 0이면 1로 처리하고, position이 0보다 작으면 문자열 str의 뒤에서부터 추출한다.
length	정수 값을 반환하는 임의의 연산식이다. 문자열 내의 문자의 위치는 1부터 시작된다. 만약 length가 지정되지 않으면 str의 position 위치에서부터 마지막까지 문자열을 추출하며, length가 1보다 작으면 NULL을 반환한다.

- 예제

  다음은 SUBSTR 함수를 사용하는 예이다.

```
SQL> SELECT SUBSTR('ABCDEFG', 3),
            SUBSTR('ABCDEFG', 3, 2)
     FROM DUAL;

SUBSTR('ABCDEFG',3) SUBSTR('ABCDEFG',3,2)
------------------- ---------------------
CDEFG               CD

1 row selected.

SQL> SELECT SUBSTR('ABCDEFG', -3),
            SUBSTR('ABCDEFG', -3, 2)
     FROM DUAL;

SUBSTR('ABCDEFG',-3) SUBSTR('ABCDEFG',-3,2)
-------------------- ----------------------
EFG                   EF

1 row selected.
```

4.2.157. SUM

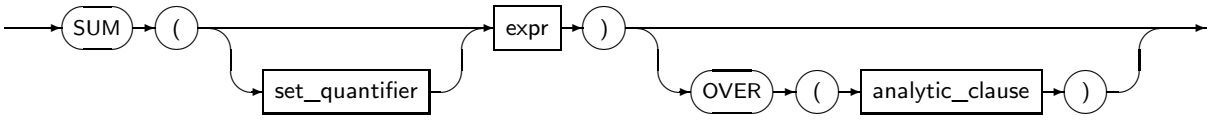
SUM은 그룹 내의 모든 로우에 대한 expr 값의 합계를 구하는 함수이다. 분석 함수로도 사용할 수 있다.

이 함수를 분석 함수로 사용할 때 DISTINCT 예약어를 명시하면 analytic_clause에서 partition_by만 명시할 수 있다. order_by_clause는 명시할 수 없다.

SUM의 세부 내용은 다음과 같다.

- 문법

sum



- 구성요소

구성요소	설명
set_quantifier	질의 결과에 중복된 로우의 허용, 비허용 여부를 지정한다. DISTINCT, UNIQUE, ALL을 지정할 수 있다. – DISTINCT, UNIQUE : 중복된 로우를 제거한다. – ALL : 모든 로우를 선택한다. (기본값)
expr	임의의 연산식이다. expr 앞에 DISTINCT 예약어를 포함하면, 합계를 구하기 전에 expr 값 중에서 중복된 것을 먼저 제거한다.
analytic_clause	OVER analytic_clause를 사용해 함수를 분석 함수로 수행할 수 있다. 자세한 내용은 “4.1.3. 분석 함수”의 analytic_clause를 참고한다.

- 예제

다음은 SUM 함수를 사용하는 예이다.

```
SQL> SELECT DEPTID, SUM(SALARY) FROM EMP2 GROUP BY DEPTID;

    DEPTID SUM(SALARY)
---------- ----------
         1      14700
         2      19200

2 rows selected.
```

- 분석 함수 예제

다음은 SUM 함수를 분석 함수로 사용하는 예이다.

```
SQL> SELECT NAME, DEPTID, SALARY, SUM(SALARY)
       OVER (PARTITION BY DEPTID)
     FROM EMP2;

NAME                    DEPTID     SALARY SUM(SALARY)
-------------------- ---------- ---------- ----------
Paul                          1       3000      14700
Nick                          1       3200      14700
```

```
Scott                          1       4000      14700
John                           1       4500      14700
Bree                           2       6000      19200
Daniel                         2       5000      19200
Joe                            2       4000      19200
Brad                           2       4200      19200

8 rows selected.

SQL> SELECT NAME, SALARY, SUM(SALARY)
       OVER (ORDER BY SALARY RANGE UNBOUNDED PRECEDING)
     FROM EMP3;

NAME                      SALARY SUM(SALARY)
-------------------- ---------- ----------
Paul                        3000       3000
Nick                        3200       9400
Scott                       3200       9400
John                        3500      12900
Bree                        4000      16900
Daniel                      4500      25900
Joe                         4500      25900

7 rows selected.
```

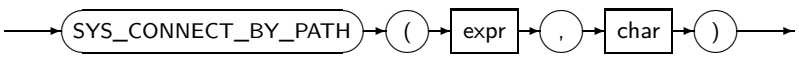
4.2.158. SYS_CONNECT_BY_PATH

SYS_CONNECT_BY_PATH는 루트(Root)에서 노드(Node)까지의 컬럼 값의 경로(Path)를 반환하는 함수이다. 이 함수는 계층 질의에 대해서만 유효하다. 반환된 경로에서 컬럼 값은 char에 의해 분리된다.

SYS_CONNECT_BY_PATH의 세부 내용은 다음과 같다.

- 문법

sys_connect_by_path



- 구성요소

구성요소	설명
expr, char	expr, char 모두 CHAR, VARCHAR2, NCHAR, NVARCHAR2 중에서 어느 타입이어도 된다. 반환된 문자열은 VARCHAR2 타입이며, 컬럼과 같은 문자 집합이다.

- 예제

다음은 SYS_CONNECT_BY_PATH 함수를 사용하는 예이다.

```
SQL> SELECT ENAME, CONNECT_BY_ROOT ENAME MANAGER,
            SYS_CONNECT_BY_PATH(ENAME, '-') PATH
     FROM EMP2
     WHERE LEVEL > 1
       CONNECT BY PRIOR EMPNO = MGRNO
       START WITH ENAME = 'Clark';

ENAME           MANAGER         PATH
--------------- --------------- ----------------------
Martin          Clark           -Clark-Martin
James           Clark           -Clark-Martin-James
Alicia          Clark           -Clark-Martin-Alicia
Ramesh          Clark           -Clark-Ramesh
Allen           Clark           -Clark-Ramesh-Allen
JohnClark       Clark           -Ramesh-John
Ward            Clark           -Clark-Ramesh-John-Ward

7 rows selected.
```

4.2.159. SYS_CONTEXT

SYS_CONTEXT는 문맥 네임스페이스(CONTEXT NAMESPACE)와 관련된 파라미터의 값을 반환하는 함수이다. 문맥 네임스페이스와 파라미터는 문자열이나 표현식으로 정의할 수 있으며, 함수의 반환값은 VARCHAR 타입이다.

Tibero에서 디폴트로 제공하고 있는 문맥 네임스페이스는 USERENV이다.

- USERENV 파라미터

파라미터	설명
CLIENT_IDENTIFIER	DBMS_SESSION.SET_IDENTIFIER 프러시저에 의해 정의된 클라이언트 식별자를 반환한다.
CURRENT_SCHEMA	현재 활성화된 스키마 이름을 반환한다. 이 값은 ALTER SESSION SET CURRENT_SCHEMA 구문에 의해 변경될 수 있다.
CURRENT_SCHEMAID	현재 활성화된 스키마 식별번호를 반환한다.
CURRENT_SQL	현재 수행 중인 SQL 문장을 반환한다.
DB_NAME	DB_NAME 초기화 파라미터에 의해 정의된 데이터베이스의 이름을 반환한다.
HOST	클라이언트가 실행 중인 장비의 이름을 반환한다.
INSTANCE_NAME	현재 인스턴스의 이름을 반환한다.

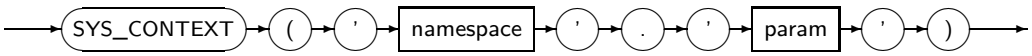
파라미터	설명
INSTANCE	현재 인스턴스의 식별번호를 반환한다.
IP_ADDR[ESS]	클라이언트의 IP 주소를 반환한다.
LANG	'LANGUAGE' 파라미터의 약어를 반환한다.
LANGUAGE	데이터베이스의 문자집합 이름을 반환한다.
MODULE	DBMS_APPLICATION_INFO.SET_MODULE 프러시저에 의해 정의된 모듈 이름을 반환한다.
NETWORK_PROTOCOL	양방향 통신을 위한 네트워크 프로토콜의 이름을 반환한다.
OS_USER	클라이언트 프로세스의 OS 사용자 이름을 반환한다.
SCHEMA	클라이언트의 접속 스키마 이름을 반환한다.
SCHEMAID	클라이언트의 접속 사용자 식별번호를 반환한다.
SERVER_ADDRESS	현재 인스턴스가 실행 중인 장비의 IP 주소를 반환한다.
SERVER_HOST	현재 인스턴스가 실행 중인 장비의 이름을 반환한다.
SESSION_USER	클라이언트의 접속 사용자 이름을 반환한다.
SESSIONID	세션 감시 식별번호를 반환한다.
TERMINAL	클라이언트의 OS 식별자를 반환한다.
TID	현재 세션 식별번호를 반환한다.

사용자 지정 문맥 네임스페이스는 CREATE CONTEXT DDL 구문을 사용하여 생성할 수 있다. 또한 이 SYS_CONTEXT 가 해당 네임스페이스의 각 파라미터에 따라 반환할 값은 DBMS_SESSION.SET_CONTEXT 프러시저를 통해 설정할 수 있다.

SYS_CONTEXT의 세부 내용은 다음과 같다.

- 문법

sys_context



- 구성요소

구성요소	설명
namespace	문맥 네임스페이스를 정의하는 값이다.
param	문맥 네임스페이스와 관련된 파라미터 이름이다.

- 예제

  다음은 SYS_CONTEXT 함수를 사용하는 예이다.

```
SQL> SELECT SYS_CONTEXT('USERENV', 'TID') "TID"
     FROM DUAL;

TID
---
  1

1 row selected.
```

다음은 지정된 패키지를 통해 ctx0 네임스페이스의 attr0 파라미터 값이 설정되었을 경우 그 값을 SYS_CONTEXT 함수를 통해 출력하는 예이다.

```
SQL> SELECT SYS_CONTEXT('ctx0', 'attr0') "ATTR0"
     FROM DUAL;

ATTR0
-----
 val0

1 row selected.
```

4.2.160. SYS_EXTRACT_UTC

SYS_EXTRACT_UTC는 시간대 정보를 포함하는 시간값을 UTC (Coordinated Universal Time) 시간대로 변환하여 반환하는 함수이다.

SYS_EXTRACT_UTC의 세부 내용은 다음과 같다.

- 문법

sys_extract_utc

- 구성요소

구성요소	설명
datetime_with_timezone	시간대 정보를 포함한 시간값을 반환하는 임의의 연산식이다.

- 예제

다음은 SYS_EXTRACT_UTC 함수를 사용하는 예이다.

```
SQL> SELECT SYS_EXTRACT_UTC(TIMESTAMP '1994/07/23 21:13:08 -8:00') FROM DUAL;

SYS_EXTRACT_UTC(TIMESTAMP'1994/07/2321:13:08-8:00')
```

```
-------------------------------------------------
1994/07/24 05:13:08.000000

1 row selected.
```

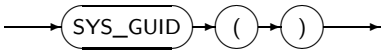
4.2.161. SYS_GUID

SYS_GUID는 시스템 전체에서 유일한 값을 반환하는 함수이다. 반환값은 16bytes이고 스레드 번호, 호스트 이름, 접속 시간, 시퀀스 번호등의 조합으로 이루어진 RAW 타입의 값이다.

SYS_GUID의 세부 내용은 다음과 같다.

- 문법

  sys_guid

- 예제

  다음은 SYS_GUID 함수를 사용하는 예이다.

```
SQL> SELECT SYS_GUID() "SYS_GUID" FROM DUAL;

SYS_GUID
--------------------------------
120000000080087893B1484201000000

1 row selected.
```

4.2.162. SYSDATE

SYSDATE는 현재의 날짜와 시간을 반환하는 함수이다. 이 함수는 파라미터가 없으며, 괄호가 따라오지 않는다. 이 함수의 반환값은 DATE 타입이다.

SYSDATE의 세부 내용은 다음과 같다.

- 문법

  sysdate

- 예제

다음은 SYSDATE 함수를 사용하는 예이다.

```
SQL> SELECT SYSDATE FROM DUAL;

SYSDATE
----------
2009-12-03

1 row selected.
```

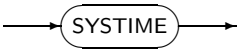
4.2.163. SYSTIME

SYSTIME은 현재 시간을 반환하는 함수이다. 이 함수는 파라미터가 없으며, 괄호가 따라 오지 않는다. 이 함수의 반환값은 TIME 타입이다.

SYSTIME의 세부 내용은 다음과 같다.

- 문법

systime



- 예제

다음은 SYSTIME 함수를 사용하는 예이다.

```
SQL> SELECT SYSTIME FROM DUAL;

SYSTIME
---------------
13:42:05.455775

1 row selected.
```

4.2.164. SYSTIMESTAMP

SYSTIMESTAMP는 현재의 날짜와 시간 및 시간대를 반환하는 함수이다. 이 함수의 반환값은 TIMESTAMP WITH TIME ZONE 타입이다.

SYSTIMESTAMP의 세부 내용은 다음과 같다.

- 문법

systimestamp

- 예제

  다음은 SYSTIMESTAMP 함수를 사용하는 예이다.

```
SQL> SELECT SYSTIMESTAMP FROM DUAL;

SYSTIMESTAMP
---------------------------------------
2009/12/03 13:42:45.816763 Asia/Seoul

1 row selected.
```

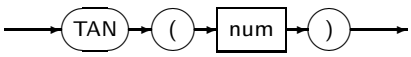
4.2.165. TAN

TAN는 주어진 파라미터의 탄젠트(Tangent) 값을 구하는 함수이다.

TAN의 세부 내용은 다음과 같다.

- 문법

  tan



- 구성요소

구성요소	설명
num	실수 값을 반환하는 임의의 연산식이다. (단위: 라디안) num은 숫자형 타입이거나 숫자형 타입으로 변환될 수 있는 타입이어야 한다. 만약 num의 타입이 BINARY_FLOAT일 경우엔 BINARY_DOUBLE 타입을 반환하고, 그렇지 않은 경우엔 num과 동일한 숫자형 타입을 반환한다.

- 예제

  다음은 TAN 함수를 사용하는 예이다.

```
SQL> SELECT TAN(3.141592654 * 45.0 / 180.0) FROM DUAL;

TAN(3.141592654*45.0/180.0)
---------------------------
```

```
                          1

1 row selected.
```

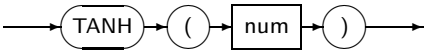
4.2.166. TANH

TANH는 주어진 파라미터의 하이퍼볼릭 탄젠트 값을 구하는 함수이다.

TANH의 세부 내용은 다음과 같다.

- 문법

tanh



- 구성요소

구성요소	설명
num	실수 값을 반환하는 임의의 연산식이다. (단위: 라디안) num은 숫자형 타입이거나 숫자형 타입으로 변환될 수 있는 타입이어야 한다. 만약 num의 타입이 BINARY_FLOAT일 경우엔 BINARY_DOUBLE 타입을 반환하고, 그렇지 않은 경우엔 num과 동일한 숫자형 타입을 반환한다.

- 예제

다음은 TANH 함수를 사용하는 예이다.

```
SQL> SELECT TANH(1) FROM DUAL;

   TANH(1)
----------
.761594156

1 row selected.
```

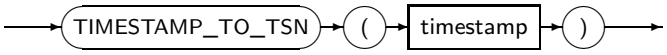
4.2.167. TIMESTAMP_TO_TSN

TIMESTAMP_TO_TSN은 주어진 시간값과 가장 근접한 TSN값을 반환하는 함수이다.

TIMESTAMP_TO_TSN의 세부 내용은 다음과 같다.

- 문법

timestamp_to_tsn



- 구성요소

구성요소	설명
timestamp	시간 값을 반환하는 임의의 연산식이다.

- 예제

다음은 TIMESTAMP_TO_TSN 함수를 사용하는 예이다.

```
SQL> SELECT TIMESTAMP_TO_TSN(SYSTIMESTAMP) FROM DUAL;

TIMESTAMP_TO_TSN(SYSTIMESTAMP)
------------------------------
                        382716

1 row selected.
```

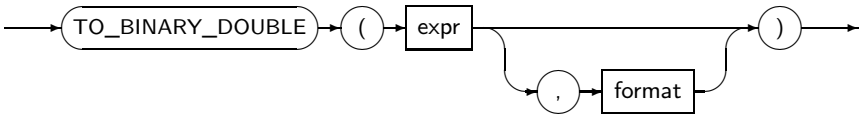
4.2.168. TO_BINARY_DOUBLE

TO_BINARY_DOUBLE은 연산식 expr을 format에 따라 BINARY_DOUBLE 타입의 값으로 변환하여 반환하는 함수이다.

TO_BINARY_DOUBLE의 세부 내용은 다음과 같다.

- 문법

to_binary_double



- 구성요소

구성요소	설명
expr	문자열이나 수치값을 반환하는 임의의 연산식이다. 다음과 같은 특수한 문자열 값을 지원하고, 대소문자는 구분하지 않는다. – 'INF'는 양의 무한대로 변환된다.

구성요소	설명
	– '-INF'는 음의 무한대로 변환된다. – 'NaN'은 NaN(Not A Number)로 변환된다. 부동 소수점 타입을 나타내는 접미사 f, F, d, D는 사용할 수 없다.
format	숫자형 타입의 형식 문자열이다. 만약 format이 지정되지 않으면 디폴트 형식으로 변환을 수행한다. format은 expr이 문자열인 경우에만 유효하다.

- 예제

다음은 TO_BINARY_DOUBLE 함수를 사용하는 예이다.

```
SQL> SELECT TO_BINARY_DOUBLE(3.141592), TO_BINARY_DOUBLE('+inf') FROM DUAL;

TO_BINARY_DOUBLE(3.141592) TO_BINARY_DOUBLE('+INF')
-------------------------- ------------------------
                3.142E+000                      Inf

1 row selected.
```

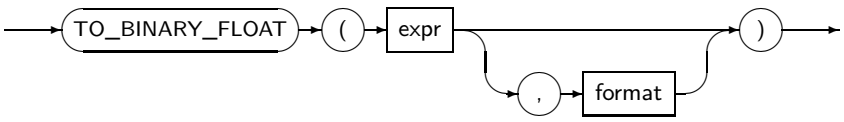
4.2.169. TO_BINARY_FLOAT

TO_BINARY_FLOAT는 연산식 expr을 format에 따라 BINARY_FLOAT 타입의 값으로 변환하여 반환하는 함수이다.

TO_BINARY_FLOAT의 세부 내용은 다음과 같다.

- 문법

to_binary_float



- 구성요소

구성요소	설명
expr	문자열이나 수치값을 반환하는 임의의 연산식이다. 다음과 같은 특수한 문자열 값을 지원하고, 대소문자는 구분하지 않는다.

구성요소	설명
	– 'INF'는 양의 무한대로 변환된다. – '-INF'는 음의 무한대로 변환된다. – 'NaN'은 NaN(Not A Number)로 변환된다. 부동 소수점 타입을 나타내는 접미사 f, F, d, D는 사용할 수 없다.
format	숫자형 타입의 형식 문자열이다. 만약 format이 지정되지 않으면 디폴트 형식으로 변환을 수행한다. format은 expr이 문자열인 경우에만 유효하다.

- 예제

  다음은 TO_BINARY_FLOAT 함수를 사용하는 예이다.

```
SQL> SELECT TO_BINARY_FLOAT(3.141592), TO_BINARY_FLOAT('-inf') FROM DUAL;

TO_BINARY_FLOAT(3.141592) TO_BINARY_FLOAT('-INF')
------------------------- -----------------------
               3.142E+000                    -Inf

1 row selected.
```

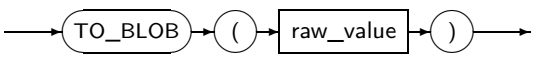
4.2.170. TO_BLOB

TO_BLOB은 파라미터로 주어진 raw를 BLOB 타입으로 변환하는 함수이다.

TO_BLOB의 세부 내용은 다음과 같다.

- 문법

  to_blob



- 구성요소

구성요소	설명
raw_value	RAW type을 반환하는 임의의 연산식이다.

- 예제

  다음은 TO_BLOB 함수를 사용하는 예이다.

```
SQL> SELECT TO_BLOB('123') FROM DUAL;

TO_BLOB('123')
-----------------
0123

1 row selected.
```

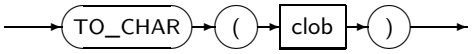
4.2.171. TO_CHAR(character)

TO_CHAR(character)는 CLOB 타입의 데이터를 데이터베이스 문자 집합을 사용한 문자열로 변환해 주는 함수이다.

TO_CHAR(character)의 세부 내용은 다음과 같다.

- 문법

to_char_char



- 구성요소

구성요소	설명
clob	문자열로 변환할 CLOB 타입의 데이터이다.

- 예제

다음은 TO_CHAR(character) 함수를 사용하는 예이다.

```
SQL> SELECT TO_CHAR(contents) FROM BOOK;
```

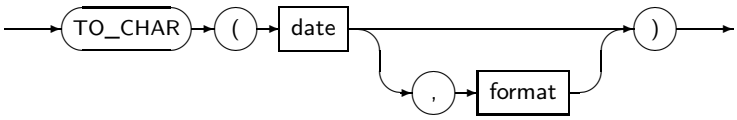
4.2.172. TO_CHAR(datetime)

TO_CHAR(datetime)은 파라미터로 주어진 날짜형 데이터 타입과 간격 리터럴의 값을 format에 따라 문자열로 변환하여 반환하는 함수이다.

TO_CHAR(datetime)의 세부 내용은 다음과 같다.

- 문법

to_char_date



- 구성요소

구성요소	설명
date	날짜형의 값을 반환하는 임의의 연산식이다.
format	날짜형 형식의 문자열이다. 만약 format이 지정되지 않으면 디폴트 형식에 따라 문자열로 변환한다. 날짜형 형식의 문자열은 “2.4.2. 날짜형 타입”을 참고한다.

- 예제

다음은 TO_CHAR(datetime) 함수를 사용하는 예이다.

```
SQL> SELECT TO_CHAR(SYSDATE) FROM DUAL;

TO_CHAR(SYSDATE)
--------------------------------
2006/12/06

1 row selected.

SQL> SELECT TO_CHAR(SYSDATE, 'DD/MM/YY DY') FROM DUAL;

TO_CHAR(SYSDATE,'DD/MM/YYDY')
-----------------------------
06/12/06 WED

1 row selected.
```

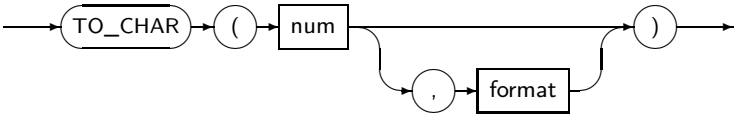
4.2.173. TO_CHAR(number)

TO_CHAR(number)는 파라미터로 주어진 숫자형 타입의 값을 format에 따라 문자열로 변환하여 반환하는 함수이다.

TO_CHAR(number)의 세부 내용은 다음과 같다.

- 문법

to_char_num



- 구성요소

구성요소	설명
num	숫자형 타입의 값을 반환하는 임의의 연산식이다.
format	숫자형 타입의 형식 문자열이다. 만약 format이 지정되지 않으면 디폴트로 모든 유효 숫자를 문자열로 변환한다. 숫자형 타입의 형식 문자열은 “2.4.1. NUMBER 타입”을 참고한다.

- 예제

  다음은 TO_CHAR (number) 함수를 사용하는 예이다.

```
SQL> SELECT TO_CHAR(MIN(SALARY)) FROM EMP;

TO_CHAR(MIN(SALARY))
--------------------
3000

1 row selected.

SQL> SELECT TO_CHAR(MIN(SALARY), '$99,999,99') FROM EMP;

TO_CHAR(MIN(SAL),'$99,999,99')
------------------------------
     $30,00

1 row selected.
```

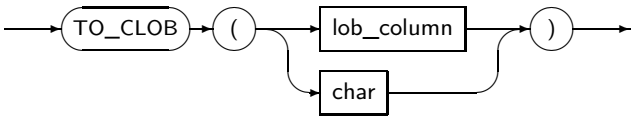
4.2.174. TO_CLOB

TO_CLOB은 파라미터로 주어진 lob_column 또는 char을 CLOB 타입으로 변환하는 함수이다.

TO_CLOB의 세부 내용은 다음과 같다.

- 문법

to_clob



- 구성요소

구성요소	설명
lob_column	CHAR 타입, VARCHAR 타입 또는 CLOB 타입의 컬럼이다.
char	문자열을 반환하는 임의의 연산식이다.

- 예제

다음은 TO_CLOB 함수를 사용하는 예이다.

```
SQL> SELECT TO_CLOB('tibero') FROM DUAL;

TO_CLOB('TIBERO')
-----------------
tibero

1 row selected.
```

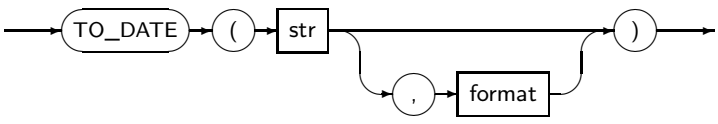
4.2.175. TO_DATE

TO_DATE은 문자열 str을 format에 따라 DATE 타입의 값으로 변환하여 반환하는 함수이다.

TO_DATE의 세부 내용은 다음과 같다.

- 문법

to_date



- 구성요소

구성요소	설명
str	문자열을 반환하는 임의의 연산식이다.
format	날짜형 형식의 문자열이다. 만약 format이 지정되지 않으면 디폴트 형식으로 변환을 수행한다.

- 예제

  다음은 TO_DATE 함수를 사용하여 DATE 타입의 데이터를 반환하는 예이다.

```
SQL> SELECT TO_DATE('25/12/2004', 'DD/MM/YYYY') FROM DUAL;

TO_DATE('25/12/2004','DD/MM/YYYY')
----------------------------------
2004/12/25

1 row selected.
```

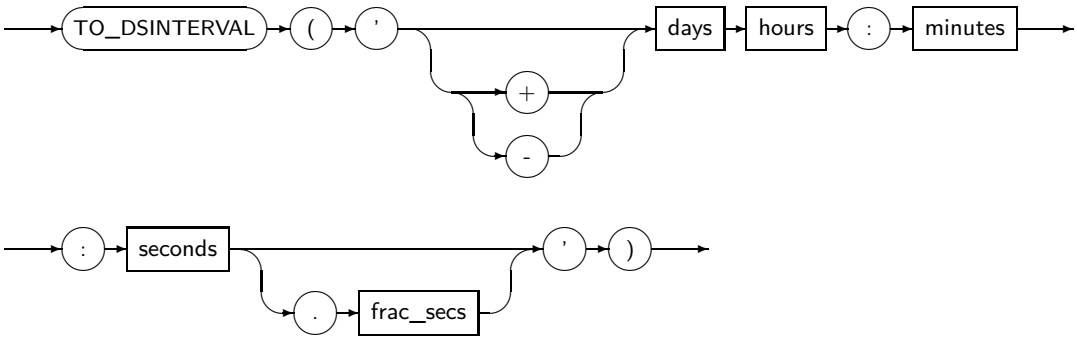
4.2.176. TO_DSINTERVAL

TO_DSINTERVAL은 CHAR, VARCHAR2, NCHAR, NVARCHAR2 타입의 문자열을 INTERVAL DAY TO SECOND 타입으로 변환하는 함수이다.

TO_DSINTERVAL의 세부 내용은 다음과 같다.

- 문법

to_dsinterval



- 구성요소

구성요소	설명
days	0~999999999 사이의 정수이다.
hours	0~23 사이의 정수이다.
minutes, seconds	0~59 사이의 정수이다.
frac_secs	0~999999999 사이의 정수이다.

- 예제

  다음은 TO_DSINTERVAL 함수를 사용하여 2008년 3월 20일을 기준으로 50일 전의 날짜를 구하는 예이다.

```
SQL> SELECT DATE '2008-03-20' - TO_DSINTERVAL('50 00:00:00')
     BEFORE FROM DUAL;

BEFORE
----------------------
2008-01-30

1 row selected.
```

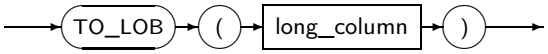
4.2.177. TO_LOB

TO_LOB은 LONG 값을 CLOB, LONG RAW 값을 BLOB 값으로 변환하는 함수이다. LONG 또는 LONG RAW 타입의 컬럼만을 인자로 가지며 INSERT 문의 서브쿼리와 UPDATE 문의 컬럼 값을 갱신하는 서브쿼리에만 사용할 수 있다.

TO_LOB의 세부 내용은 다음과 같다.

- 문법

to_lob



- 구성요소

구성요소	설명
long_column	LONG 타입 또는 LONG RAW 타입의 컬럼이다.

- 예제

다음은 TO_LOB 함수를 사용하는 예이다.

```
SQL> CREATE TABLE LONG_TABLE (ID NUMBER, LONG_COL LONG);

Table 'LONG_TABLE' created.

SQL> INSERT INTO LONG_TABLE VALUES (1, 'first');

1 row inserted.

SQL> CREATE TABLE CLOB_TABLE AS
   2 SELECT ID, TO_LOB(LONG_COL) CLOB_COL FROM LONG_TABLE;

Table 'CLOB_TABLE' created.
```

```
SQL> SELECT * FROM CLOB_TABLE;

        ID CLOB_COL
---------- ----------
         1 first

1 row selected.

SQL> INSERT INTO LONG_TABLE VALUES (2, 'second');

1 row inserted.

SQL> INSERT INTO LONG_TABLE VALUES (3, 'third');

1 row inserted.

SQL> INSERT INTO CLOB_TABLE
   2 SELECT ID, TO_LOB(LONG_COL) FROM LONG_TABLE WHERE ID = 2;

1 row inserted.

SQL> SELECT * FROM CLOB_TABLE;

        ID CLOB_COL
---------- ----------
         1 first
         2 second

2 rows selected.

SQL> UPDATE CLOB_TABLE
   2 SET CLOB_COL = (SELECT TO_LOB(LONG_COL) FROM LONG_TABLE WHERE ID = 3)
   3 WHERE ID = 2;

1 row updated.

SQL> SELECT * FROM CLOB_TABLE;

        ID CLOB_COL
---------- ----------
         1 first
         2 third

2 rows selected.
```

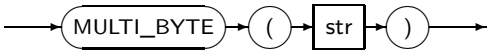
4.2.178. TO_MULTI_BYTE

TO_MULTI_BYTE는 입력으로 들어온 파라미터 str의 값을 상응하는 MULTI-BYTE 문자로 변환하는 함수이다. 입력으로 올 수 있는 파라미터의 형식은 VARCHAR, CHAR, NCHAR, NVARCHAR 타입이고, 반환 형식은 입력 형식과 같다.

TO_MULTI_BYTE의 세부 내용은 다음과 같다.

- 문법

  to_multi_byte

- 구성요소

구성요소	설명
str	MULTI-BYTE 문자로 변환할 값이다.

- 예제

  다음은 TO_MULTI_BYTE 함수를 사용하는 예이다.

```
SQL> SELECT DUMP(TO_MULTI_BYTE('A')) "TO_MULTI_BYTE" FROM DUAL;

TO_MULTI_BYTE
-----------------------
Len=3: 239,188,161

1 row selected.
```

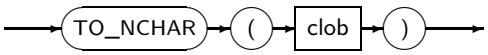
4.2.179. TO_NCHAR(character)

TO_NCHAR(character)는 CHAR, VARCHAR, CLOB, NCLOB 타입의 입력 값을 NATIONAL 문자 집합의 값으로 변환하는 함수이다.

TO_NCHAR의 세부 내용은 다음과 같다.

- 문법

  to_char_char

- 구성요소

구성요소	설명
str	NATIONAL 문자 집합으로 변환할 문자를 나타내는 값이다.

- 예제

  다음은 TO_NCHAR(character) 함수를 사용하는 예이다.

```
SQL> SELECT TO_NCHAR(LAST_NAME) "LAST_NAME" FROM EMP WHERE EMP_ID = 5;

LAST_NAME
---------
    Braun

1 row selected.
```

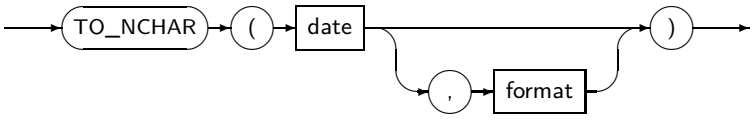
4.2.180. TO_NCHAR(datetime)

TO_NCHAR(datetime)은 파라미터로 주어진 날짜형 데이터 타입과 간격 리터럴의 값을 format에 따라 NATIONAL 문자 집합의 값으로 변환하는 함수이다.

TO_NCHAR(datetime)의 세부 내용은 다음과 같다.

- 문법

to_nchar_date



- 구성요소

구성요소	설명
date	NATIONAL 문자 집합으로 변환할 날짜형의 값을 나타낸다.
format	날짜형 형식의 문자열이다. 만약 format이 지정되지 않으면 디폴트 형식에 따라 문자열로 변환한다. 날짜형 형식의 문자열은 “2.4.2. 날짜형 타입”을 참고한다.

- 예제

  다음은 TO_NCHAR(datetime) 함수를 사용하는 예이다.

```
SQL> SQL> SELECT TO_NCHAR(SYSDATE, 'DD/MM/YY DY') FROM DUAL;

TO_NCHAR(SYSDATE,'DD/MM/YYDY')
------------------------------
26/07/21 월

1 row selected.
```

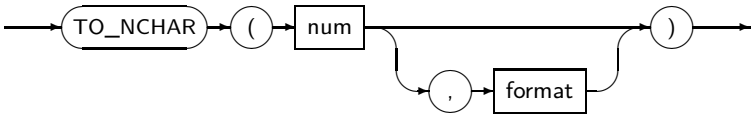
4.2.181. TO_NCHAR(number)

TO_NCHAR(number)는 파라미터로 주어진 숫자형 타입의 값을 format에 따라 NATIONAL 문자 집합의 값으로 변환하는 함수이다.

TO_NCHAR(number)의 세부 내용은 다음과 같다.

- 문법

to_nchar_num



- 구성요소

구성요소	설명
num	NATIONAL 문자 집합으로 변환할 숫자형 타입을 나타내는 값이다.
format	숫자형 타입의 형식 문자열이다. 만약 format이 지정되지 않으면 디폴트로 모든 유효 숫자를 문자열로 변환한다. 숫자형 타입의 형식 문자열은 “2.4.1. NUMBER 타입”을 참고한다.

- 예제

다음은 TO_NCHAR(number) 함수를 사용하는 예이다.

```
SQL> SQL> SELECT TO_NCHAR(30000,'$99,999,99') FROM DUAL;

TO_NCHAR(30000,'$99,999,99')
----------------------------
    $300,00

1 row selected.
```

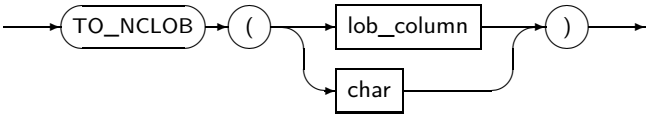
4.2.182. TO_NCLOB

TO_NCLOB은 파라미터로 주어진 lob_column 또는 char을 NCLOB 타입으로 변환하는 함수이다.

TO_NCLOB의 세부 내용은 다음과 같다.

- 문법

to_nclob



- 구성요소

구성요소	설명
lob_column	CHAR 타입, VARCHAR 타입 또는 CLOB 타입의 컬럼이다.
char	문자열을 반환하는 임의의 연산식이다.

- 예제

다음은 TO_NCLOB 함수를 사용하는 예이다.

```
SQL> SELECT TO_NCLOB('tibero') FROM DUAL;

TO_NCLOB('TIBERO')
------------------
tibero

1 row selected.
```

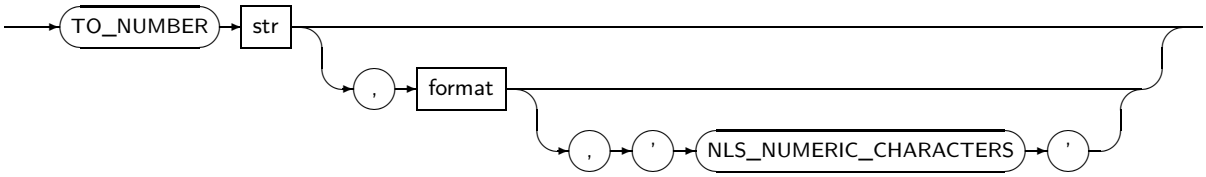
4.2.183. TO_NUMBER

TO_NUMBER는 문자열 str을 format에 따라 NUMBER 타입의 값으로 변환하여 반환하는 함수이다. NLS_NUMERIC_CHARACTERS는 그룹 구분 기호와 소숫점으로 사용할 문자를 지정한다.

TO_NUMBER의 세부 내용은 다음과 같다.

- 문법

to_number



- 구성요소

구성요소	설명
str	문자열을 반환하는 임의의 연산식이다.
format	숫자형 타입의 형식 문자열이다. 만약 format이 지정되지 않으면 디폴트 형식으로 변환을 수행한다.
NLS_NUMERIC_CHARACTERS	구분자와 소수점 지정한다.

- 예제

다음은 TO_NUMBER 함수를 사용하는 예이다.

```
SQL> SELECT TO_NUMBER('1,111.11','9G999D99',
                      ' NLS_NUMERIC_CHARACTERS=''.,'' ') NUM FROM DUAL;

       NUM
----------
   1111.11

1 row selected.
```

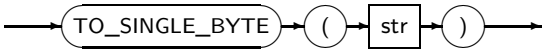
4.2.184. TO_SINGLE_BYTE

TO_SINGLE_BYTE는 입력으로 들어온 파라미터 str의 값을 상응하는 SINGLE-BYTE 문자로 변환하는 함수이다. 입력으로 올 수 있는 파라미터의 형식은 VARCHAR, CHAR, NCHAR, NVARCHAR 타입이고, 반환 형식은 입력 형식과 같다.

TO_SINGLE_BYTE의 세부 내용은 다음과 같다.

- 문법

to_single_byte



- 구성요소

구성요소	설명
str	SINGLE-BYTE 문자로 변환할 값이다.

- 예제

  다음은 TO_SINGLE_BYTE 함수를 사용하는 예이다.

```
SQL> SELECT DUMP(TO_SINGLE_BYTE(TO_MULTI_BYTE('A'))) "TO_SINGLE_BYTE" FROM DUAL;

TO_SINGLE_BYTE
--------------------------------------------------------------------------------
Len=1: 65

1 row selected.
```

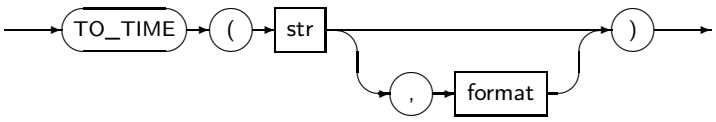
4.2.185. TO_TIME

TO_TIME은 문자열 str을 format에 따라 TIME 타입의 값으로 변환하여 반환하는 함수이다.

TO_TIME의 세부 내용은 다음과 같다.

- 문법

to_time



- 구성요소

구성요소	설명
str	문자열을 반환하는 임의의 연산식이다.
format	날짜형 형식의 문자열이다. 만약 format이 지정되지 않으면 디폴트 형식으로 변환을 수행한다.

- 예제

  다음은 TO_TIME 함수를 사용하는 예이다.

```
SQL> SELECT TO_TIME ('12:07:15.50', 'HH24:MI:SS.FF') FROM DUAL;

TO_TIME('12:07:15.50','HH24:MI:SS.FF')
-------------------------------------
12:07:15.500000000
```

```
1 row selected.
```

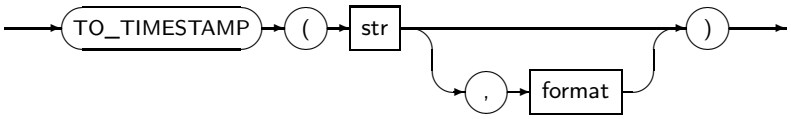
4.2.186. TO_TIMESTAMP

TO_TIMESTAMP는 문자열 str을 format에 따라 TIMESTAMP 타입의 값으로 변환하여 반환하는 함수이다.

TO_TIMESTAMP의 세부 내용은 다음과 같다.

- 문법

to_timestamp



- 구성요소

구성요소	설명
str	문자열을 반환하는 임의의 연산식이다.
format	날짜형 형식의 문자열이다. 만약 format이 지정되지 않으면 디폴트 형식으로 변환을 수행한다.

- 예제

다음은 TO_TIMESTAMP 함수를 사용하는 예이다.

```
SQL> SELECT TO_TIMESTAMP('2009/05/21 12:07:15.50','yyyy-mm-dd HH24:MI:SS.FF')
     FROM DUAL;

TO_TIMESTAMP('2009/05/2112:07:15.50','YYYY-MM-DDHH24:MI:SS.FF')
---------------------------------------------------------------
2009/05/21 12:07:15.500000

1 row selected.
```

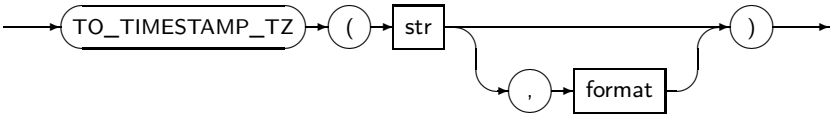
4.2.187. TO_TIMESTAMP_TZ

TO_TIMESTAMP_TZ는 문자열 str을 format에 따라 TIMESTAMP WITH TIME ZONE 타입의 값으로 변환하여 반환하는 함수이다.

TO_TIMESTAMP_TZ의 세부 내용은 다음과 같다.

- 문법

to_timestamp_tz



- 구성요소

구성요소	설명
str	문자열을 반환하는 임의의 연산식이다. 만약 문자열에 시간대 정보가 포함되지 않은 경우 세션의 디폴트 시간대를 사용한다.
format	날짜형 형식의 문자열이다. 만약 format이 지정되지 않으면 디폴트 형식으로 변환을 수행한다.

- 예제

다음은 TO_TIMESTAMP_TZ 함수를 사용하는 예이다.

```
SQL> SELECT TO_TIMESTAMP_TZ('1998/09/27 22:05:21.089','YYYY-MM-DD HH24:MI:SS.FF')
     FROM DUAL;

TO_TIMESTAMP_TZ('1998/09/2722:05:21.089','YYYY-MM-DDHH24:MI:SS.FF')
---------------------------------------------------------------------
1998/09/27 22:05:21.089000 Asia/Seoul

1 row selected.

SQL> SELECT TO_TIMESTAMP_TZ('2015-10-3 2:17:09.0 +03:00',
     'YYYY-MM-DD HH24:MI:SS.FF TZH:TZM') FROM DUAL;

TO_TIMESTAMP_TZ('2015-10-32:17:09.0+03:00','YYYY-MM-DDHH24:MI:SS.FFTZH:TZM')
-----------------------------------------------------------------------------
2015/10/03 02:17:09.000000 +03:00

1 row selected.
```

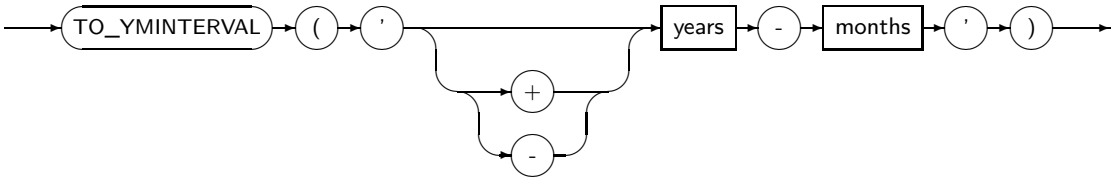
4.2.188. TO_YMINTERVAL

TO_YMINTERVAL은 CHAR, VARCHAR2, NCHAR, NVARCHAR2 타입의 문자열을 INTERVAL YEAR TO MONTH 타입으로 변환하는 함수이다.

TO_YMINTERVAL의 세부 내용은 다음과 같다.

- 문법

to_yminterval



- 구성요소

구성요소	설명
years	0~999999999 사이의 정수이다.
months	0~11 사이의 정수이다.

- 예제

다음은 TO_YMINTERVAL 함수를 사용하여 2008년 3월 20일을 기준으로 2년 7개월 이후의 날짜를 구하는 예이다.

```
SQL> SELECT DATE '2008-03-20' + TO_YMINTERVAL('2-7')
     AFTER FROM DUAL;

AFTER
----------------------
2010-10-20

1 row selected.
```

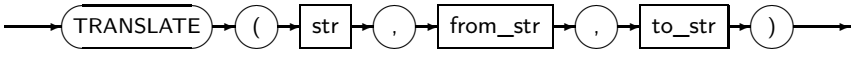
4.2.189. TRANSLATE

TRANSLATE는 문자열 str 내의 각각의 문자가 문자열 from_str에 포함되어 있다면, 해당 문자를 문자열 from_str에서의 위치와 동일한 위치에 있는 문자열 to_str의 문자로 변환시키는 함수이다.

TRANSLATE의 세부 내용은 다음과 같다.

- 문법

translate



- 구성요소

구성요소	설명
str	문자열을 반환하는 임의의 연산식이다.
from_str	문자열을 반환하는 임의의 연산식이다. 문자열 from_str의 길이가 문자열 to_str의 길이보다 긴 경우 문자열 from_str 내에는 to_str 대응되지 않는 문자가 포함된다. 만약 이렇게 to_str과 대응되지 않는 from_str의 문자가 문자열 str에 포함되어 있다면 해당 문자는 모두 제거된다.
to_str	문자열을 반환하는 임의의 연산식이다.

- 예제

다음은 TRANSLATE 함수를 사용하는 예이다.

```
SQL> SELECT TRANSLATE('ABCXYDEFZ', 'ABCDE', '12345') FROM DUAL;

TRANSLATE('ABCXYDEFZ','ABCDE','12345')
--------------------------------------
123XY45FZ

1 row selected.

SQL> SELECT TRANSLATE('ABCXYDEFZ', 'ABCDE', '123') FROM DUAL;

TRANSLATE('ABCXYDEFZ','ABCDE','123')
------------------------------------
123XYFZ

1 row selected.
```

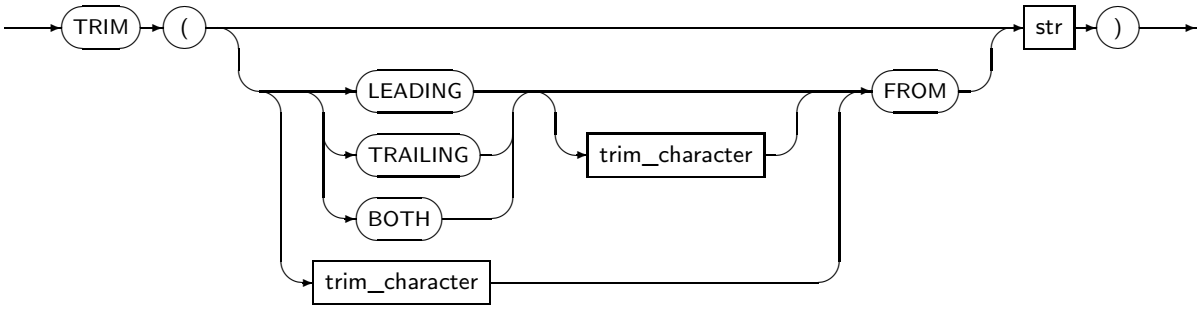
4.2.190. TRIM

TRIM은 문자열 str의 앞과 뒤에서 trim_character와 같은 문자를 모두 제거하는 함수이다.

TRIM의 세부 내용은 다음과 같다.

- 문법

trim



- 구성요소

구성요소	설명
str	문자열을 반환하는 임의의 연산식이다.
LEADING	trim_character 앞에 LEADING이 오면 str의 앞에서만 문자를 제거한다.
TRAILING	trim_character 앞에 TRAILING이 오면 str의 뒤에서만 문자를 제거한다.
BOTH	trim_character 앞에 BOTH가 오거나 아무 것도 오지 않으면 str의 앞과 뒤에서 문자를 제거한다.
trim_character	trim_character는 길이가 1인 문자이다. trim_character가 지정되지 않으면 공백 문자(스페이스)를 제거하게 된다.

- 예제

  다음은 TRIM 함수를 사용하는 예이다.

```
SQL> SELECT TRIM(' ABCDE '), LENGTH(TRIM(' ABCDE ')) FROM DUAL;

TRIM('ABCDE') LENGTH(TRIM('ABCDE'))
------------- ---------------------
ABCDE                             5

1 row selected.

SQL> SELECT TRIM(LEADING 'X' FROM 'XXYXABCDEXYXX') FROM DUAL;

TRIM(LEADING'X'FROM'XXYXABCDEXYXX')
-----------------------------------
YXABCDEXYXX

1 row selected.

SQL> SELECT TRIM(TRAILING 'X' FROM 'XXYXABCDEXYXX') FROM DUAL;

TRIM(TRAILING'X'FROM'XXYXABCDEXYXX')
```

```
-----------------------------------
XXYXABCDEXY

1 row selected.

SQL> SELECT TRIM('X' FROM 'XXYXABCDEXYXX') FROM DUAL;

TRIM('X'FROM'XXYXABCDEXYXX')
----------------------------
YXABCDEXY

1 row selected.
```

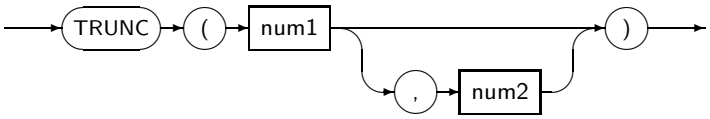
4.2.191. TRUNC(number)

TRUNC(number)는 num1을 소수점 아래 num2 위치에서 버림(Truncation)한 값을 반환하는 함수이다.

TRUNC(number)의 세부 내용은 다음과 같다.

- 문법

trunc_num



- 구성요소

구성요소	설명
num1, num2	수치 값을 반환하는 임의의 연산식이다. num1이나 num2는 숫자형 타입이거나 숫자형 타입으로 변환될 수 있는 타입이어야 한다. num2는 정수이어야 하고, 만약 num2가 주어지지 않거나 0이면 소수점 자리에서 버림을 한다. num2가 음수이면 소수점 위의 자리에서 버림을 한다.

- 예제

다음은 TRUNC(number)를 사용하는 예이다.

```
SQL> SELECT TRUNC(345.678), TRUNC(345.678, 2), TRUNC(345.678, -1)
     FROM DUAL;

TRUNC(345.678) TRUNC(345.678,2) TRUNC(345.678,-1)
-------------- ---------------- -----------------
```

```
           345           345.67               340

1 row selected.
```

4.2.192. TRUNC(date)

TRUNC(date)는 date를 format에 명시된 단위로 버림한 결과를 반환하는 함수이다.

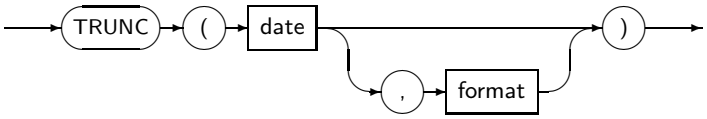
참고

TRUNC(date)함수의 format에 명시할 수 있는 형식 문자열은 ROUND(date) 함수와 동일하다.

TRUNC(date)의 세부 내용은 다음과 같다.

- 문법

trunc_date



- 구성요소

구성요소	설명
date	날짜를 반환하는 임의의 연산식이다.
format	버림 단위를 명시하는 형식 문자열이다. format이 명시되지 않는다면 'DD' 형식 문자열을 이용하여 가장 가까운 날로 버림이 된다. format으로 'YEAR', 'MONTH', 'DAY' 등을 사용할 수 있다.

- 예제

다음은 TRUNC(date) 함수를 사용하는 예이다.

```
SQL> SELECT TRUNC(TO_DATE('2005/06/22', 'YYYY/MM/DD'), 'YEAR')
     FROM DUAL;

TRUNC(TO_DATE('2005/06/22','YYYY/MM/DD'),'YEAR')
------------------------------------------------------------------
2005-01-01

1 row selected.

SQL> SELECT TO_CHAR(TRUNC(TO_DATE(
```

```
       '1998/6/20', 'YYYY/MM/DD'), 'CC'), 'YYYY/MM/DD')
     FROM DUAL;

TO_CHAR(TRUNC(TO_DATE('1998/6/20','YYYY/MM/DD'),'CC'),'YYYY/MM/DD')
--------------------------------------------------------------------
1901/01/01

1 row selected.

SQL> SELECT TO_CHAR(TRUNC(TO_DATE('-3741/01/02', 'SYYYY/MM/DD'), 'CC'),
       'SYYYY/MM/DD')
     FROM DUAL;

TO_CHAR(TRUNC(TO_DATE('-3741/01/02','SYYYY/MM/DD'),'CC'),'SYYYY/MM/DD')
-----------------------------------------------------------------------
-3800/01/01

1 row selected.

SQL> SELECT TO_CHAR(TRUNC(TO_DATE(
       '2005/01/26 12:30:14', 'YYYY/MM/DD HH24:MI:SS'), 'DY'), 'YYYY/MM/DD')
     FROM DUAL;

TO_CHAR(TRUNC(TO_DATE('2005/01/2612:30:14','YYYY/MM/DDHH24:MI:SS'),'DY'),'YYYY/M
--------------------------------------------------------------------------------
2005/01/23

1 row selected.

SQL> SELECT TRUNC(TO_DATE(
       '2005/01/26 12:30:14', 'YYYY/MM/DD HH24:MI:SS'))
     FROM DUAL;

TRUNC(TO_DATE('2005/01/2612:30:14','YYYY/MM/DDHH24:MI:SS'))
-----------------------------------------------------------
2005-01-26

1 row selected.
```

4.2.193. TSN_TO_TIMESTAMP

TSN_TO_TIMESTAMP은 주어진 TSN과 가장 근접한 시간값을 반환하는 함수이다.

TSN_TO_TIMESTAMP의 세부 내용은 다음과 같다.

- 문법

tsn_to_timestamp

- 구성요소

구성요소	설명
num	숫자 값을 반환하는 임의의 연산식이다.

- 예제

다음은 TSN_TO_TIMESTAMP 함수를 사용하는 예이다.

```
SQL> SELECT TSN_TO_TIMESTAMP(527720) FROM DUAL;

TSN_TO_TIMESTAMP(527720)
---------------------------------------------------------------
2014/11/27 09:34:01.708203000

1 row selected.
```

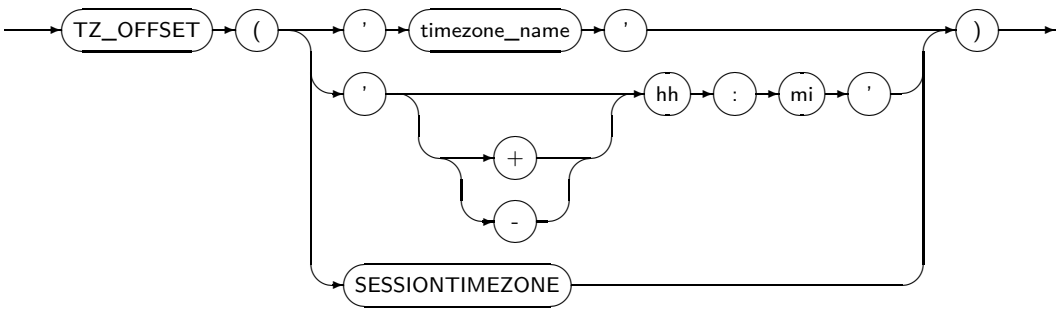
4.2.194. TZ_OFFSET

TZ_OFFSET은 주어진 시간대를 오프셋 형식으로 반환하는 함수이다.

TZ_OFFSET의 세부 내용은 다음과 같다.

- 문법

tz_offset



- 구성요소

구성요소	설명
timezone_name	시간대 지역 이름을 반환하는 문자열이다.

구성요소	설명
[+|-]hh:mi	시간대 오프셋 값을 반환하는 문자열이다.
SESSIONTIMEZONE	현재 세션의 시간대 값을 반환하는 함수이다.

- 예제

  다음은 TZ_OFFSET을 사용하는 예이다.

```
SQL> SELECT TZ_OFFSET('-5:00') FROM DUAL;

TZ_OFFSET('-5:00')
------------------------------
-05:00

1 row selected.
```

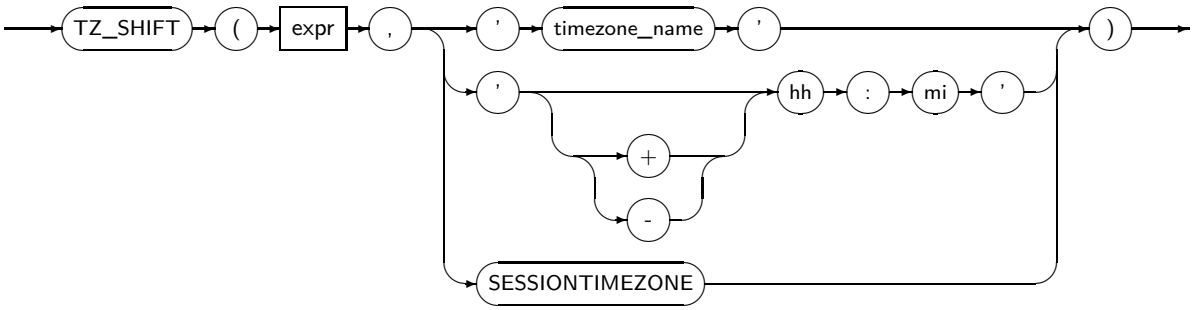
4.2.195. TZ_SHIFT

TZ_SHIFT는 주어진 시간대를 특정 시간대로 변환하는 함수이다.

TZ_SHIFT의 세부 내용은 다음과 같다.

- 문법

tz_shift



- 구성요소

구성요소	설명
expr	시간대 정보를 포함한 시간값을 반환하는 임의의 연산식이다.
timezone_name	시간대 지역 이름을 반환하는 문자열이다.
[+|-]hh:mi	시간대 오프셋 값을 반환하는 문자열이다.
SESSIONTIMEZONE	현재 세션의 시간대 값을 반환하는 함수이다.

- 예제

다음은 TZ_SHIFT을 사용하는 예이다.

```
SQL> SELECT TZ_SHIFT(FROM_TZ(TIMESTAMP '2002-09-08 21:00:00',
       '00:00'), '+09:00') SEOUL FROM DUAL;

SEOUL
---------------------------------
2002/09/09 06:00:00.000000 +09:00

1 row selected.
```

4.2.196. UID

UID는 현재 세션을 생성한 사용자의 ID를 반환하는 함수이다. 사용자 ID는 Tibero의 시스템 내에서 그 사용자를 식별해주는 유일한 정수 값이다. 파라미터는 없으며 괄호를 생략한다.

UID의 세부 내용은 다음과 같다.

- 문법

  uid

- 예제

  다음은 UID 함수를 사용하는 예이다.

```
SQL> SELECT UID FROM DUAL;

       UID
----------
      7171

1 row selected.
```

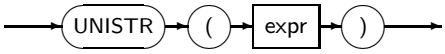
4.2.197. UNISTR

UNISTR는 유니코드 문자열을 입력받아 다국어 문자집합으로 인코딩된 문자열을 반환하는 함수이다. 입력으로 사용하는 유니코드 문자열은 '\xxxx' 형태이며, 여기서 'xxxx'는 UCS-2 포맷으로 인코딩된 16진수 문자열이다.

UNISTR의 세부 내용은 다음과 같다.

- 문법

unistr



- 구성요소

구성요소	설명
str	문자열을 반환하는 임의의 연산식이다.

- 예제

다음은 UNISTR 함수를 사용하는 예이다.

```
SQL> SELECT UNISTR('\00E5\00F1\00F6') FROM DUAL;

UNISTR('\00E5\00F1\00F6')
-------------------------
åñö

1 row selected.
```

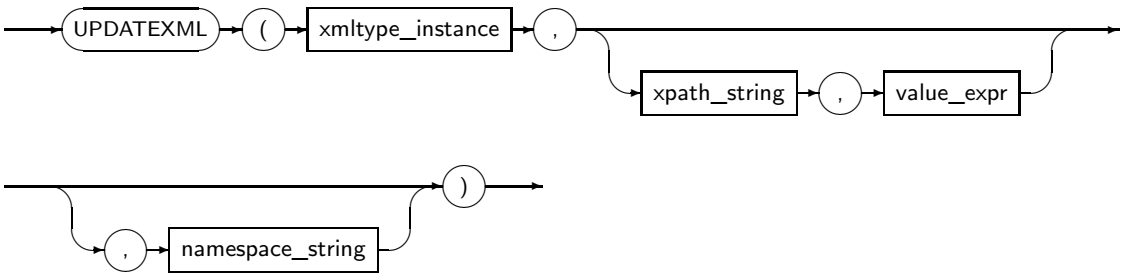
4.2.198. UPDATEXML

UPDATEXML은 XPath expression과 XMLType instance의 쌍으로 입력된 인자들로 XMLType_instance의 각 해당 xpath들을 value_expr의 value로 치환하는 함수이다. 실제 xml column 값이 바뀌지 않고, 바뀐 XMLType instance가 반환된다. 만약 해당 xpath에 해당하는 node가 존재하지 않으면 원래의 바뀌지 않은 xmltype instance가 반환된다.

UPDATEXML의 세부 내용은 다음과 같다.

- 문법

updatexml



- 구성요소

구성요소	설명
XMLType_instance	XML 타입의 instance다.
XPath_string	Xpath expression으로 업데이트할 곳의 child node들이 업데이트된다.
value_expr	업데이트할 곳의 child node들이 값이다.
namespace_string	XPath_string의 네임스페이스 정보를 나타낸다. VARCHAR 타입이어야 한다.

- 예제

  다음은 UPDATEXML 함수를 사용하는 예이다.

```
SQL> SELECT warehouse_name,
   EXTRACT(warehouse_spec, '/Warehouse/Docks') "Number of Docks"
   FROM warehouses WHERE warehouse_name = 'San Francisco';

WAREHOUSE_NAME       Number of Docks
-------------------- --------------------
San Francisco        <Docks>1</Docks>

SQL> UPDATE warehouses SET warehouse_spec =
   UPDATEXML(warehouse_spec, '/Warehouse/Docks/text()',4)
   WHERE warehouse_name = 'San Francisco';

1 row updated.

SQL>SELECT warehouse_name,
   EXTRACT(warehouse_spec, '/Warehouse/Docks') "Number of Docks"
   FROM warehouses WHERE warehouse_name = 'San Francisco';

WAREHOUSE_NAME       Number of Docks
-------------------- --------------------
San Francisco        <Docks>4</Docks>
```

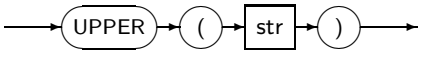
4.2.199. UPPER

UPPER는 문자열 str 내의 모든 영문자를 대문자로 변환하여 반환하는 함수이다.

UPPER의 세부 내용은 다음과 같다.

- 문법

  upper



- 구성요소

구성요소	설명
str	문자열을 반환하는 임의의 연산식이다.

- 예제

다음은 UPPER 함수를 사용하는 예이다.

```
SQL> SELECT UPPER('ABCdefg123') FROM DUAL;

UPPER('ABCDEFG123')
-------------------
ABCDEFG123

1 row selected.
```

4.2.200. USER

USER는 현재 세션을 생성한 사용자의 이름을 반환하는 함수이다. 파라미터는 없으며 괄호를 생략한다.

USER의 세부 내용은 다음과 같다.

- 문법

user

- 예제

다음은 USER 함수를 사용하는 예이다.

```
SQL> SELECT USER FROM DUAL;

USER
------------------------------
JOE

1 row selected.
```

4.2.201. USERENV

USERENV는 현재 세션의 정보를 보여주는 함수이다.

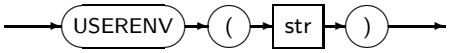
참고

USERENV 함수에서 제공하는 파라미터는 “4.2.159. SYS_CONTEXT”를 참고한다.

USERENV의 세부 내용은 다음과 같다.

- 문법

userenv



- 구성요소

구성요소	설명
str	문자열을 반환하는 임의의 연산식이다.

- 예제

  다음은 USERENV 함수를 사용하는 예이다.

```
SQL> SELECT USERENV('TID') FROM DUAL;

USERENV('TID')
----------------------------------------------------------------
7

1 row selected.
```

4.2.202. VAR_POP

VAR_POP은 expr의 모분산을 반환한다. 모분산을 계산할 때 NULL 값은 무시된다. 이 함수는 파라미터의 수치 데이터 타입 또는 변환된 수치 데이터 타입을 반환한다. 이 함수를 공집합에 적용하면 NULL 값을 반환한다.

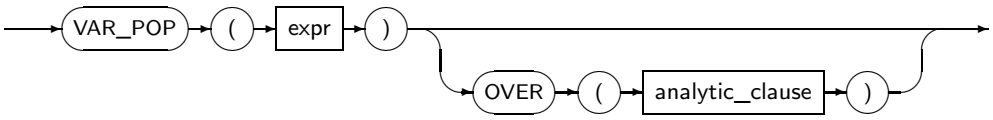
이 함수는 다음과 같은 계산식으로 계산을 수행한다.

```
(SUM(expr2) - SUM(expr)2 / COUNT(expr)) / COUNT(expr)
```

VAR_POP의 세부 내용은 다음과 같다.

- 문법

var_pop



- 구성요소

구성요소	설명
expr	수치 데이터 타입 또는 수치 데이터 타입은 아니지만 묵시적으로 수치 데이터 타입으로 변환할 수 있는 데이터 타입을 반환하는 임의의 연산식이다.
analytic_clause	OVER analytic_clause를 사용해 함수를 분석 함수로 수행할 수 있다. 자세한 내용은 “4.1.3. 분석 함수”의 analytic_clause를 참고한다.

- 예제

다음은 VAR_POP 함수를 사용하는 예이다.

```
SQL> SELECT VAR_POP(AGE) FROM EMP_AGE;

VAR_POP(AGE)
------------
        8.29

1 row selected.
```

- 분석 함수 예제

다음은 VAR_POP 함수를 분석 함수로 사용하는 예이다.

```
SQL> SELECT DEPTNO, EMPNO, VAR_POP(SAL)
       OVER (PARTITION BY DEPTNO) AS VAR_POP
     FROM EMP;

    DEPTNO      EMPNO    VAR_POP
---------- ---------- ----------
        10       7934 2390555.56
        10       7839 2390555.56
        10       7782 2390555.56
        20       7566    1009500
        20       7788    1009500
        20       7876    1009500
        20       7902    1009500
        20       7369    1009500
        30       7654 372222.222
        30       7698 372222.222
        30       7521 372222.222
        30       7499 372222.222
```

```
        30       7844 372222.222
        30       7900 372222.222

14 rows selected.
```

4.2.203. VAR_SAMP

VAR_SAMP는 expr의 표본분산을 반환하는 함수이다. 표본분산을 계산할 때 NULL 값은 무시된다. 이 함수는 파라미터의 수치 데이터 타입 또는 변환된 수치 데이터 타입을 반환한다. 이 함수를 공집합에 적용하면 NULL 값을 반환한다.

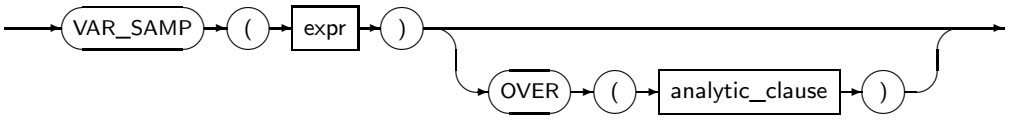
이 함수는 다음과 같은 계산식으로 계산을 수행한다.

```
(SUM(expr2) - SUM(expr)2 / COUNT(expr)) / (COUNT(expr) - 1)
```

VAR_SAMP의 세부 내용은 다음과 같다.

- 문법

var_samp



- 구성요소

구성요소	설명
expr	수치 데이터 타입 또는 수치 데이터 타입은 아니지만 묵시적으로 수치 데이터 타입으로 변환할 수 있는 데이터 타입을 반환하는 임의의 연산식이다.
analytic_clause	OVER analytic_clause를 사용해 함수를 분석 함수로 수행할 수 있다. 자세한 내용은 “4.1.3. 분석 함수”의 analytic_clause를 참고한다.

- 예제

다음은 VAR_SAMP 함수를 사용하는 예이다.

```
SQL> SELECT VAR_SAMP(AGE) FROM EMP_AGE;

VAR_SAMP(AGE)
-------------
9.21111111111

1 row selected.
```

- 분석 함수 예제

다음은 VAR_SAMP 함수를 분석 함수로 사용하는 예이다.

```
SQL> SELECT DEPTNO, EMPNO, VAR_SAMP(SAL)
       OVER (PARTITION BY DEPTNO) AS VAR_SAMP
     FROM EMP;

    DEPTNO      EMPNO   VAR_SAMP
---------- ---------- ----------
        10       7934 3585833.33
        10       7839 3585833.33
        10       7782 3585833.33
        20       7566    1261875
        20       7788    1261875
        20       7876    1261875
        20       7902    1261875
        20       7369    1261875
        30       7654 446666.667
        30       7698 446666.667
        30       7521 446666.667
        30       7499 446666.667
        30       7844 446666.667
        30       7900 446666.667

14 rows selected.
```

4.2.204. VARIANCE

VARIANCE는 expr의 분산을 반환한다. 분석 함수로도 사용할 수 있다. 이 함수는 파라미터의 수치 데이터 타입 또는 변환된 수치 데이터 타입을 반환한다. 이 함수를 공집합에 적용하면 NULL 값을 반환한다.

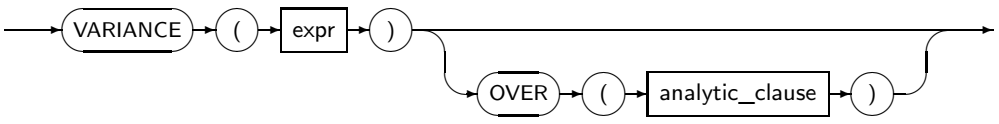
Tibero에서는 분산을 다음과 같이 계산한다.

- expr의 로우의 수가 1이면 0을 반환한다.
- expr의 로우의 수가 1보다 크면 VAR_SAMP 함수의 값을 반환한다.

VARIANCE의 세부 내용은 다음과 같다.

- 문법

variance



- 구성요소

구성요소	설명
expr	수치 데이터 타입 또는 수치 데이터 타입은 아니지만 묵시적으로 수치 데이터 타입으로 변환할 수 있는 데이터 타입을 반환하는 임의의 연산식이다.
analytic_clause	OVER analytic_clause를 사용해 함수를 분석 함수로 수행할 수 있다. 자세한 내용은 “4.1.3. 분석 함수”의 analytic_clause를 참고한다.

- 예제

  다음은 VARIANCE 함수를 사용하는 예이다.

```
SQL> SELECT VARIANCE(AGE) FROM EMP_AGE;

VARIANCE(AGE)
-------------
9.21111111111

1 row selected.
```

- 분석 함수 예제

  다음은 VARIANCE 함수를 분석 함수로 사용하는 예이다.

```
SQL> SELECT DEPTNO, EMPNO, VARIANCE(SAL)
       OVER (PARTITION BY DEPTNO) AS VARIANCE
     FROM EMP;

    DEPTNO      EMPNO    VARIANCE
---------- ---------- -----------
        10       7934 3585833.333
        10       7839 3585833.333
        10       7782 3585833.333
        20       7566     1261875
        20       7788     1261875
        20       7876     1261875
        20       7902     1261875
        20       7369     1261875
        30       7654 446666.6667
        30       7698 446666.6667
        30       7521 446666.6667
        30       7499 446666.6667
        30       7844 446666.6667
        30       7900 446666.6667

14 rows selected.
```

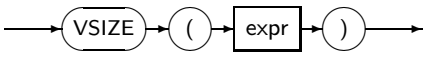
4.2.205. VSIZE

VSIZE는 expr을 표현하기 위해 내부적으로 사용되는 byte 크기를 반환하는 함수이다.

VSIZE의 세부 내용은 다음과 같다.

- 문법

vsize



- 구성요소

구성요소	설명
expr	임의의 연산식이다. expr이 NULL이면 함수는 NULL을 반환한다.

- 예제

다음은 VSIZE 함수를 사용하는 예이다.

```
SQL> SELECT VSIZE(SYSDATE) FROM DUAL;

VSIZE(SYSDATE)
--------------
             8

1 row selected.
```

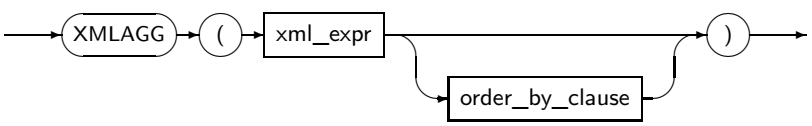
4.2.206. XMLAGG

XMLAGG는 집합 함수이다. XML 조각을 받고, 이를 한데 모아 XML 문서로 만들어 반환하는 함수이다. NULL 값을 반환하는 파라미터는 결과로부터 제외된다.

XMLAGG의 세부 내용은 다음과 같다.

- 문법

xmlagg



- 구성요소

구성요소	설명
xml_expr	XML 문장을 반환하는 임의의 연산식이다.
order_by_clause	XML 문장들을 어떻게 정렬할지를 명시한다. 자세한 내용은 “4.1.3. 분석 함수”의 order_by_clause를 참고한다.

- 예제

  다음은 XMLAGG 함수를 사용하는 예이다.

```
SQL> SELECT XMLELEMENT("EMPLOYEE",
            XMLAGG(XMLELEMENT("NAME",NAME)),
            XMLELEMENT("AGE",AGE)) AS "EMPLOYEE"
     FROM EMP_AGE
     GROUP BY AGE;

EMPLOYEE
--------------------------------------------------------------------------
<EMPLOYEE><NAME>Jim Clark</NAME><NAME>John Ronaldo</NAME><NAME>Choi S
efo</NAME><NAME>Titicaca Eboue</NAME><AGE>27</AGE></EMPLOYEE>
<EMPLOYEE><NAME>Mirko Fedor</NAME><NAME>Doug Bush</NAME><AGE>28</AGE>
</EMPLOYEE>
<EMPLOYEE><NAME>Razor Ramon</NAME><AGE>34</AGE></EMPLOYEE>
<EMPLOYEE><NAME>Mike Thai</NAME><NAME>Ryu Hayabusa</NAME><NAME>Sebast
ian Panucci</NAME><AGE>31</AGE></EMPLOYEE>

4 rows selected.
```

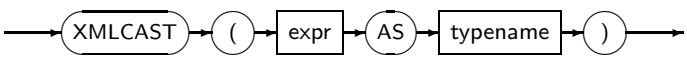
4.2.207. XMLCAST

XMLCAST는 첫 번째 파라미터의 값을 두 번째 파라미터에서 지정한 SQL 데이터 타입으로 변환하는 함수이다.

XMLCAST의 세부 내용은 다음과 같다.

- 문법

  xmlcast



- 구성요소

구성요소	설명
expr	XML 문장을 반환하는 임의의 연산식이다.

구성요소	설명
typename	변환할 SQL 데이터 타입의 이름을 명시한다.

- 예제

  다음은 XMLCAST 함수를 사용하는 예이다.

```
SQL> SELECT XMLCAST(
               XMLQUERY('/BOOK' PASSING C1
               RETURNING CONTENT) AS VARCHAR(100)) CAST_VALUE
     FROM XMLTBL;

CAST_VALUE
--------------------------------------------------------------------------------

Introduction to TIBERO

Tibero team


1 row selected.
```

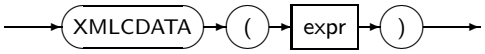
4.2.208. XMLCDATA

XMLCDATA는 expr의 결과로 CDATA 섹션을 만드는 함수이다. **<![CDATA[string]]>**와 같은 포맷의 XMLType을 반환한다. 단, 이 함수는 Solaris 환경에서 지원하지 않는다.

XMLCDATA의 세부 내용은 다음과 같다.

- 문법

  xmlcdata



- 구성요소

구성요소	설명
expr	XML CDATA 섹션의 내용이 될 문자열이다. expr이 NULL이면 함수는 NULL을 반환한다.

- 예제

  다음은 XMLCDATA 함수를 사용하는 예이다.

```
SQL> SELECT XMLCDATA('Tibero') AS "XMLCDATA" FROM DUAL;

XMLCDATA
--------------------------------------------------------------------
<![CDATA[Tibero]]>

1 row selected.
```

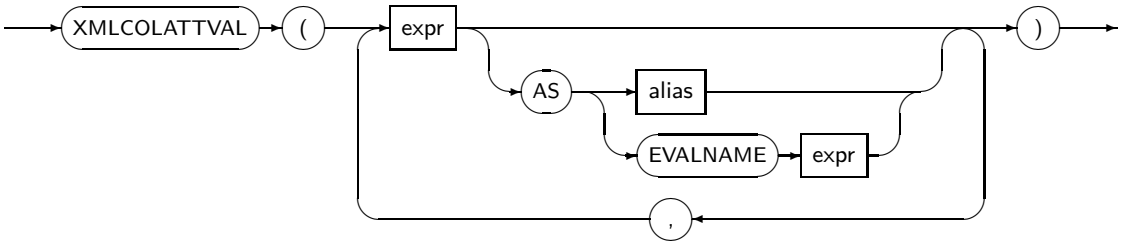
4.2.209. XMLCOLATTVAL

XMLCOLATTVAL는 파라미터 각각을 "column" 이름과 "name" 속성을 가지는 XML 노드로 변환하고, 변환된 파라미터를 결합하여 반환하는 함수이다.

XMLCOLATTVAL의 세부 내용은 다음과 같다.

- 문법

xmlcolattval



- 구성요소

구성요소	설명
expr	expr이 컬럼이면 AS 절을 생략할 수 있으며, Tibero는 컬럼명을 노드 속성 이름으로 사용한다. expr이 NULL이면, 해당 expression에 어떠한 속성도 생성되지 않는다.
alias, EVALNAME expr	AS 절을 이용하여 expr의 별칭을 명시한다. alias나 expr의 값은 반드시 문자열이어야 하며, 그 크기는 최대 4000자이다.

- 예제

다음은 XMLCOLATTVAL 함수를 사용하는 예이다.

```
SQL> SELECT XMLCOLATTVAL(LOC) AS "Locations" FROM DEPT;

Locations
```

```
--------------------------------------------------------------------------
<column name="LOC">NEW YORK</column>
<column name="LOC">DALLAS</column>

2 rows selected.
```

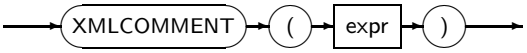
4.2.210. XMLCOMMENT

XMLCOMMENT는 expr의 결과로 XML 주석을 만드는 함수이다. **<!--string-->**와 같은 포맷의 XMLType을 반환한다.

XMLCOMMENT의 세부 내용은 다음과 같다.

- 문법

xmlcomment



- 구성요소

구성요소	설명
expr	XML 주석의 내용이 될 문자열이다. expr이 NULL이면 함수는 NULL을 반환한다.

- 예제

다음은 XMLCOMMENT 함수를 사용하는 예이다.

```
SQL> SELECT XMLCOMMENT('Tibero') AS "XMLCOMMENT" FROM DUAL;

XMLCOMMENT
--------------------------------------------------------------------------
<!--Tibero-->

1 row selected.
```

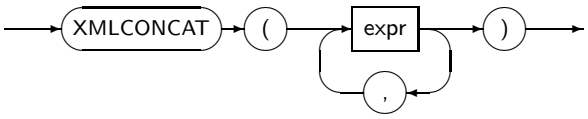
4.2.211. XMLCONCAT

XMLCONCAT는 입력으로 나열된 모든 XMLType 객체들을 접합한 결과를 반환한다. 임의의 expr이 NULL일 경우엔 해당 expr을 무시하고, 모든 expr이 NULL일 경우엔 NULL을 반환한다.

XMLCONCAT의 세부 내용은 다음과 같다.

- 문법

xmlconcat



- 구성요소

구성요소	설명
expr	XMLType 객체를 반환하는 연산식이다.

- 예제

다음은 XMLCONCAT 함수를 사용하는 예이다.

```
SQL> SELECT XMLCONCAT(XMLTYPE('<Tibero>TIBERO</Tibero>'),
     XMLTYPE('<Tibero></Tibero>')) AS "XMLCONCAT" FROM DUAL;

XMLCONCAT
------------------------------------------------------------
<Tibero>TIBERO</Tibero><Tibero></Tibero>

1 row selected.
```

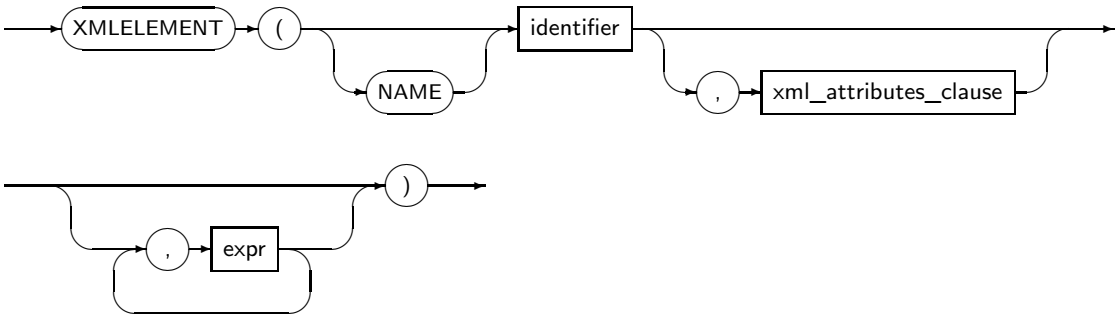
4.2.212. XMLELEMENT

XMLELEMENT는 identifier로는 엘리먼트(Element)의 이름을 받고, 엘리먼트의 속성 집합을 옵션으로 받으며, 엘리먼트의 내용을 구성하는 파라미터를 받는 함수이다. 이 함수는 중첩된 구조를 가진 XML 문서를 생성하기 위해서 보통은 중첩되어 사용된다.

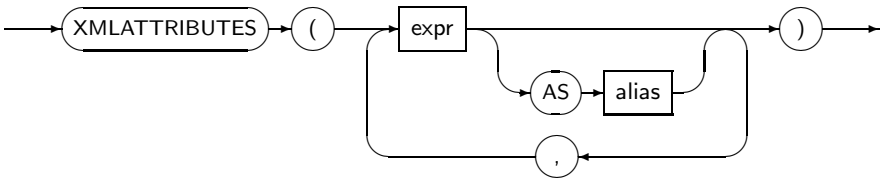
XMLELEMENT의 세부 내용은 다음과 같다.

- 문법

xmlelement



xml_attributes_clause



- 구성요소
  - xmlelement

구성요소	설명
identifier	identifier의 값은 XML의 enclosing tag로 사용되기 때문에 반드시 제공되어야 한다. identifier의 크기는 4000자까지이며, 컬럼명 또는 컬럼에 대한 참조일 필요는 없지만 표현식 또는 NULL 값일 수도 없다. 엘리먼트의 내용을 구성하는 개체는 XMLATTRIBUTES 예약어 이후에 명시한다.
xml_attributes_clause	XML의 속성(attribute)을 표현하기 위한 문법이다. 엘리먼트의 값을 나타낸다.
expr	엘리먼트의 내용을 구성하는 연산식이다.

  - xml_attributes_clause

구성요소	설명
expr	expr이 컬럼이면 AS 절을 생략할 수 있으며, Tibero는 컬럼명을 엘리먼트 이름으로 사용한다. expr이 NULL이면, 해당 value expression에 어떠한 속성도 생성되지 않는다. value expression은 XMLELEMENT의 expr을 의미한다.
alias	AS 절을 이용하여 expr의 별칭을 명시할 때 c_alias의 크기는 최대 4000자이다.

- 예제

  다음은 XMLELEMENT 함수를 사용하는 예이다.

```
SQL> SELECT XMLELEMENT("DEPT",
            XMLELEMENT("DNAME",DNAME),
            XMLELEMENT("LOCATION",LOC)) AS "DEPT"
     FROM DEPT;

DEPT
--------------------------------------------------------------------
<DEPT><DNAME>ACCOUNTING</DNAME><LOCATION>NEW YORK</LOCATION></DEPT>
<DEPT><DNAME>RESEARCH</DNAME><LOCATION>DALLAS</LOCATION></DEPT>
<DEPT><DNAME>SALES</DNAME><LOCATION>CHICAGO</LOCATION></DEPT>
<DEPT><DNAME>OPERATIONS</DNAME><LOCATION>BOSTON</LOCATION></DEPT>
```

```
4 rows selected.
```

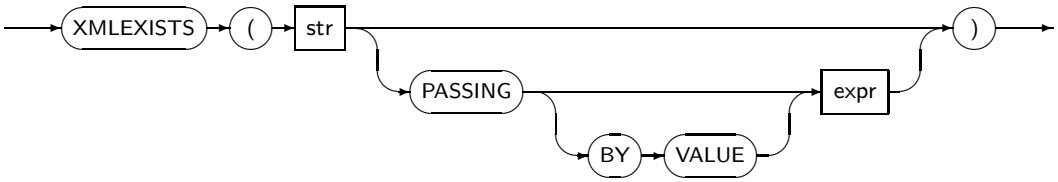
4.2.213. XMLEXISTS

XMLEXISTS는 XQuery 표현식의 결과가 비어 있는지의 여부를 검사하는 함수이다. 이 함수의 반환값이 비어 있지 않으면 TRUE를, 비어 있으면 FALSE인 Boolean 값이다. 단, 이 함수는 Linux IA64, HP PA-RISC, Windows, Solaris 환경에서 지원하지 않는다.

XMLEXISTS의 세부 내용은 다음과 같다.

- 문법

xmlexists



- 구성요소

구성요소	설명
str	XML 문서에 질의할 XQuery 문자열이다.
expr	XMLType이 결과인 표현식이다. 표현식의 결과가 XMLType이 아니면 에러를 발생한다.

- 예제

다음은 XMLEXISTS 함수를 사용하는 예이다.

```
SQL> SELECT OBJECT_VALUE AS "DEPT"
     FROM DEPT_XMLTABLE
     WHERE XMLEXISTS('/DEPT[DNAME="ACCOUNTING"]'
                     PASSING OBJECT_VALUE);

DEPT
--------------------------------------------------------------------
<DEPT><DNAME>ACCOUNTING</DNAME><LOC>NEW YORK</LOC></DEPT>

1 row selected.
```

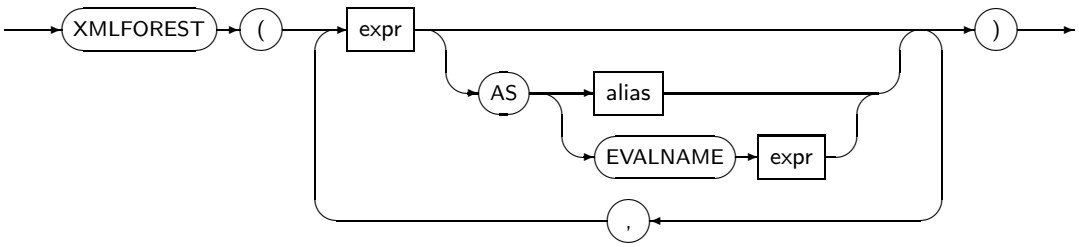
4.2.214. XMLFOREST

XMLForest는 파라미터 각각을 XML로 변환하고, 변환된 파라미터를 결합하여 반환하는 함수이다.

XMLFOREST의 세부 내용은 다음과 같다.

- 문법

xmlforest



- 구성요소

구성요소	설명
expr	expr이 컬럼이면 AS 절을 생략할 수 있으며, Tibero는 컬럼명을 엘리먼트 이름으로 사용한다. expr이 NULL이면, 해당 value expression에 어떠한 속성도 생성되지 않는다.
alias	AS 절을 이용하여 expr의 별칭을 명시할 때 c_alias의 크기는 최대 4000자이다.

- 예제

다음은 XMLFOREST 함수를 사용하는 예이다.

```
SQL> SELECT XMLELEMENT("DEPT",
            XMLFOREST(DNAME,LOC)) AS "DEPT"
     FROM DEPT;

DEPT
--------------------------------------------------------------------
<DEPT><DNAME>ACCOUNTING</DNAME><LOC>NEW YORK</LOC></DEPT>
<DEPT><DNAME>RESEARCH</DNAME><LOC>DALLAS</LOC></DEPT>
<DEPT><DNAME>SALES</DNAME><LOC>CHICAGO</LOC></DEPT>
<DEPT><DNAME>OPERATIONS</DNAME><LOC>BOSTON</LOC></DEPT>

4 rows selected.
```

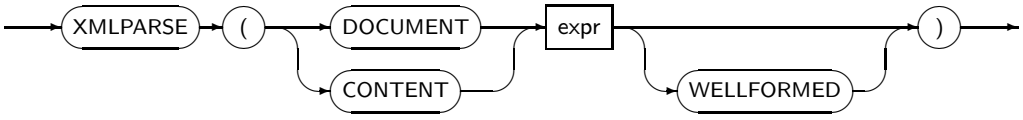
4.2.215. XMLPARSE

XMLPARSE는 expr로부터 XMLType을 만드는 함수이다.

XMLPARSE의 세부 내용은 다음과 같다.

- 문법

xmlparse



- 구성요소

구성요소	설명
DOCUMENT	expr의 결과는 반드시 루트가 하나인 XML 문서이어야 한다.
CONTENT	expr의 결과는 XML 값이어야 한다.
expr	expr의 결과는 반드시 문자열 타입이어야 한다.
WELLFORMED	CONTENT의 경우 데이터베이스 내부적으로 WELLFORMED CHECK를 하지 않는다.

- 예제

다음은 XMLPARSE 함수를 사용하는 예이다.

```
SQL> SELECT XMLPARSE(CONTENT
       'ABC<NAME>Park</NAME><ID>123</ID>' WELLFORMED) AS PO
     FROM DUAL;

PO
--------------------------------------------------------------------
ABC<NAME>Park</NAME><ID>123</ID>

1 row selected.
```

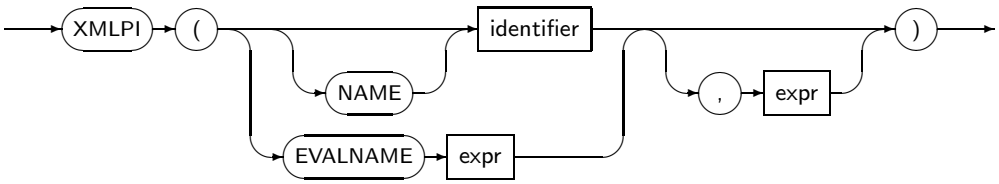
4.2.216. XMLPI

XMLPI는 XML 문서의 PI 문법을 생성하는 함수이다. XMLType을 반환한다.

XMLPI의 세부 내용은 다음과 같다.

- 문법

xmlpi



- 구성요소

구성요소	설명
identifier	XML 문서의 PI 문법에서 name에 들어갈 문자열이다. XML 내부에서 사용하는 예약어 또는 물음표(?), >를 포함할 수 없다.
expr	문자열이 결과인 expr이다. 생략되면, 길이가 0인 문자열로 간주된다.

- 예제

다음은 XMLPI 함수를 사용하는 예이다.

```
SQL> SELECT XMLPI(NAME "RDBMS", 'Tibero') AS XMLPI FROM DUAL;

XMLPI
--------------------------------------------
<?RDBMS Tibero?>

1 row selected.
```

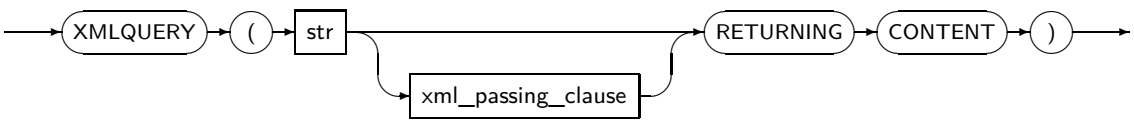
4.2.217. XMLQUERY

XMLQUERY는 지정된 XML 쿼리를 이용하여 해당 XML 데이터를 가져오는 함수이다. XMLType을 반환한다. 단, 이 함수는 Linux IA64, HP PA-RISC, Windows, Solaris 환경에서 지원하지 않는다.

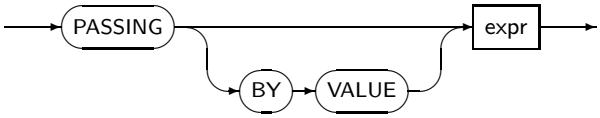
XMLQUERY의 세부 내용은 다음과 같다.

- 문법

xmlquery



xml_passing_clause



- 구성요소

구성요소	설명
str	XML 문서에 질의할 XQuery 문자열이다. 최대 4000자이다.
expr	질의의 대상이 되는 XML 문서가 결과인 표현식이다. XMLType이어야 한다.

- 예제

다음은 XMLQUERY 함수를 사용하는 예이다.

```
SQL> SELECT XMLQUERY('1, 2, 5' RETURNING CONTENT) AS XMLQRY FROM DUAL;

XMLQRY
----------------------------------------
1 2 5
```

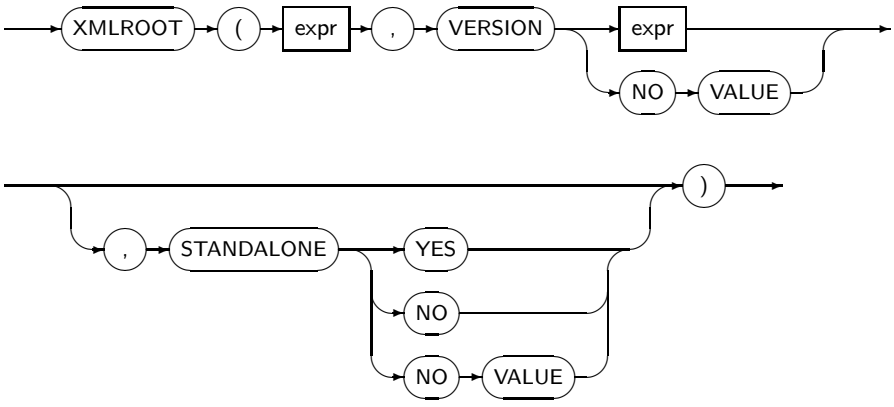
4.2.218. XMLROOT

XMLROOT는 XML 문서의 선언부를 생성하는 함수이다. **<?xml version="version" [standalone="{yes|no}"]?>**와 같은 포맷의 XMLType을 반환한다. 단, 이 함수는 Solaris 환경에서 지원하지 않는다.

XMLROOT의 세부 내용은 다음과 같다.

- 문법

xmlroot



- 구성요소

구성요소	설명
expr	XMLType이 결과인 표현식이다. 표현식의 결과가 XMLType이 아니면 에러를 발생한다.
VERSION	문자열을 반환하는 표현식이다. 'NO VALUE'가 주어진 경우 "1.0"을 기본값으로 사용한다.
STANDALONE	STANDALONE 구문이 생략되거나 'NO VALUE'가 사용된 경우엔 생략된다.

- 예제

다음은 XMLROOT 함수를 사용하는 예이다.

```
SQL> SELECT XMLROOT(XMLTYPE('<id>31679</id>'), VERSION '1.0', STANDALONE YES)
     AS "XMLDecl" FROM DUAL;

XMLDecl
--------------------------------------------------------------------------------
<?xml version="1.0" standalone="yes"?><id>31679</id>

1 rows selected.
```

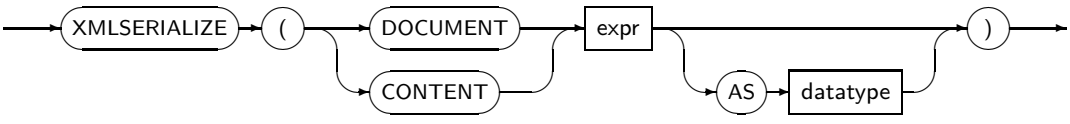
4.2.219. XMLSERIALIZE

XMLSERIALIZE는 expr로부터 문자열이나 LOB 객체를 만드는 함수이다.

XMLSERIALIZE의 세부 내용은 다음과 같다.

- 문법

xmlserialize



- 구성요소

구성요소	설명
DOCUMENT	expr의 결과는 반드시 루트가 하나인 XML 문서이어야 한다.
CONTENT	expr의 결과는 XML 값이어야 한다.
expr	XML 문서를 반환하는 임의의 연산식이다.

구성요소	설명
datatype	결과 타입을 지정한다. VARCHAR2와 CLOB을 사용할 수 있다. (기본값: CLOB)

- 예제

  다음은 XMLSERIALIZE 함수를 사용하는 예이다.

```
SQL> SELECT XMLSERIALIZE(CONTENT XMLTYPE('<A>123</A>')) AS XMLSERI FROM DUAL;

XMLSERI
-----------------------------------------
<A>123</A>

1 row selected.
```

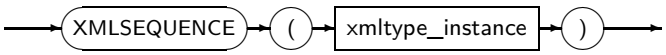
4.2.220. XMLSEQUENCE

XMLSEQUENCE는 XMLType을 입력으로 XMLType의 VARRAY를 반환하는 함수이다.

XMLSEQUENCE의 세부 내용은 다음과 같다.

- 문법

  xmlsequence



- 구성요소

구성요소	설명
xmltype_instance	질의의 대상이 되는 XML 문서로, XMLType이다.

- 예제

  다음은 XMLSEQUENCE함수를 사용하는 예이다.

```
SQL> CREATE TABLE T (col XMLTYPE);

Table 'T' created.

SQL> INSERT INTO T VALUES('<A><B>1</B><C>2</C></A>');

1 row inserted.

SQL> SELECT column_value FROM T, TABLE(XMLSEQUENCE(EXTRACT(col, '/A/B')));
```

```
COLUMN_VALUE
-----------------------------------------
<B>1</B>

1 row selected.
```

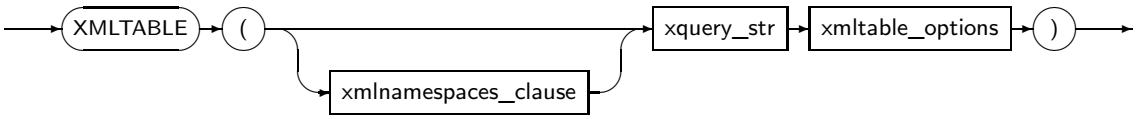
4.2.221. XMLTABLE

XMLTABLE은 XML 문서에 XML Query를 질의한 결과를 테이블 형태로 반환하는 함수이다. 각 컬럼의 값은 XML Query의 결과 XML에서 XPath를 통해 설정하며, 타입 지정을 통해서 해당 데이터 타입으로 설정되어 반환된다. 단, 이 함수는 Linux IA64, HP PA-RISC, Windows, Solaris 환경에서 지원하지 않는다.

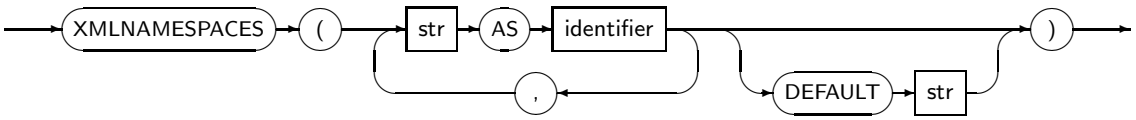
XMLTABLE의 세부 내용은 다음과 같다.

- 문법

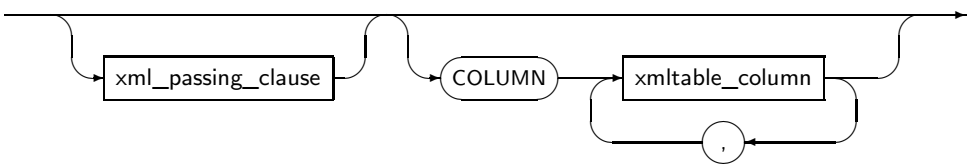
xmltable



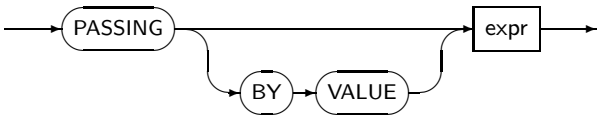
xmlnamespaces_clause



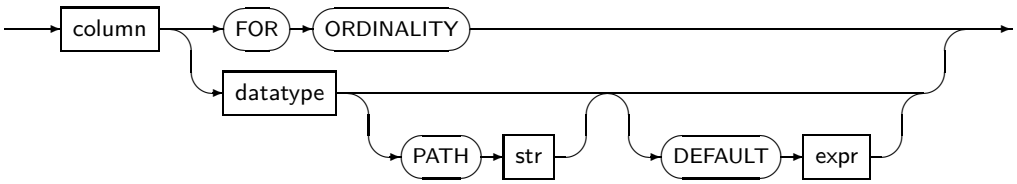
xmltable_options



xml_passing_clause



xmltable_column



- 구성요소

구성요소	설명
xquery_str	XML 문서에 질의할 XML Query이다. 최대 길이는 4000자로 제한된다.
xmlnamespace_clause의 str	XML Query를 질의할 특정 네임스페이스를 지정할 수 있다. 필요 없으면 생략할 수 있다.
xml_passing_clause의 expr	XML Query를 질의할 XML 문서이다.
xmltable_column의 FOR ORDINALITY	FOR ORDINALITY 구문은 column이 생성된 row numbers의 column이 되도록 지정한다. FOR ORDINALITY 절은 하나만 있어야 한다. NUMBER column으로 생성된다.
xmltable_column의 datatype	결과 컬럼의 데이터 타입을 명시한다.
xmltable_column의 str	결과 컬럼의 이름을 명시한다. 문자열이며 최대 4000자이다.
xmltable_column의 expr	해당 XPath에 값이 없을 경우 디폴트 값이다. 이 문법이 생략되면 NULL이 기본값이다.

- 예제

다음은 XMLTABLE 함수를 사용하는 예이다.

```
SQL> SELECT warehouse_name warehouse,
            warehouse2."Water", warehouse2."Rail"
     FROM warehouses,
          XMLTABLE('/Warehouse'
            PASSING warehouses.warehouse_spec
            COLUMNS
              "Water" varchar2(6) PATH '/Warehouse/WaterAccess',
              "Rail" varchar2(6) PATH '/Warehouse/RailAccess')
       warehouse2;

WAREHOUSE                          Water  Rail
---------------------------------- ------ ------
Southlake, Texas                   Y      N
San Francisco                      Y      N
New Jersey                         N      N
Seattle, Washington                N      Y
```

```
4 rows selected.
```

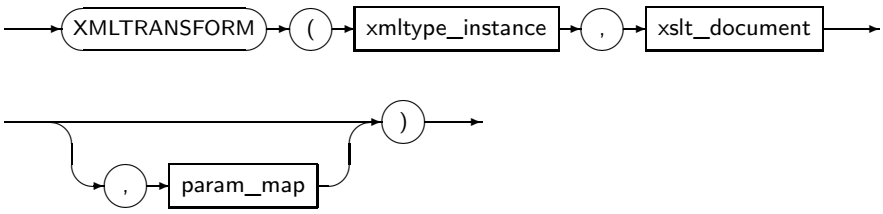
4.2.222. XMLTRANSFORM

XMLTRANSFORM은 xmltype_instance를 xslt_document에 표현된 형식으로 변형하는 함수이다. param_map을 통해 xslt_document에 정의된 파라미터를 입력할 수 있다. 단, 이 함수는 Windows 64bits, HP UX/PA-RISC, Solaris 환경에서 지원하지 않는다.

XMLTRANSFORM의 세부 내용은 다음과 같다.

- 문법

xmltransform



- 구성요소

구성요소	설명
xmltype_instance	질의의 대상이 되는 XML 문서로, XMLType이다.
xslt_document	변형 형식이 표현된 XSLT 문서로, XMLType이다.
param_map	XSLT 문서에 정의된 파라미터를 입력할 수 있다. name1=value1 name2=value2 ... 형식의 문자열이다.

- 예제

다음은 XMLTRANSFORM 함수를 사용하는 예이다.

```
SQL> select XMLTRANSFORM(XMLTYPE('<root/>'),
   XMLTYPE('<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:fo="http://www.w3.org/1999/XSL/Format"
   exclude-result-prefixes="fo">
   <xsl:output method="xml"/><xsl:param name="par1"/>
   <xsl:param name="par2"/><xsl:template match="/">
   <output><param1><xsl:value-of select="$par1"/></param1>
   <param2><xsl:value-of select="$par2"/></param2></output>
   </xsl:template></xsl:stylesheet>'),
```

```
   'par1="a" par2="b"') as result from dual;

RESULT
--------------------------------------------------------------------------------
<output><param1>a</param1><param2>b</param2></output>

1 row selected.
```

제5장 SQL 질의

본 장에서는 SELECT에 대해 자세히 설명한다.

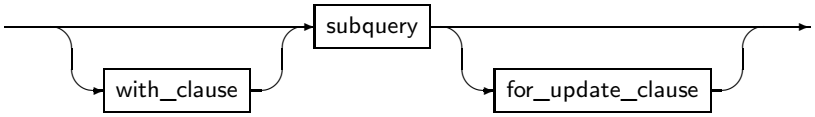
5.1. SELECT

SELECT 구문은 하나 이상의 테이블, 뷰로부터 원하는 데이터를 조회한다.

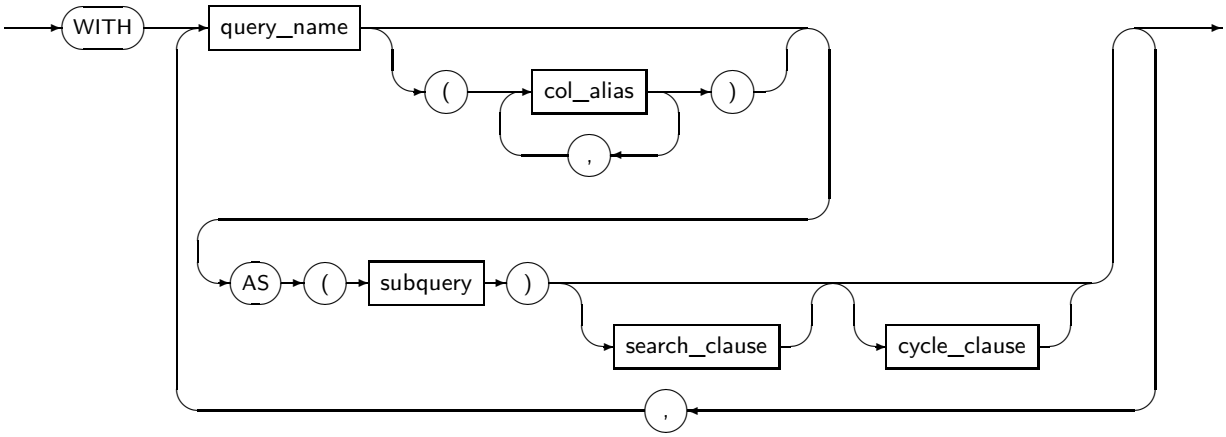
SELECT의 세부 내용은 다음과 같다.

- 문법

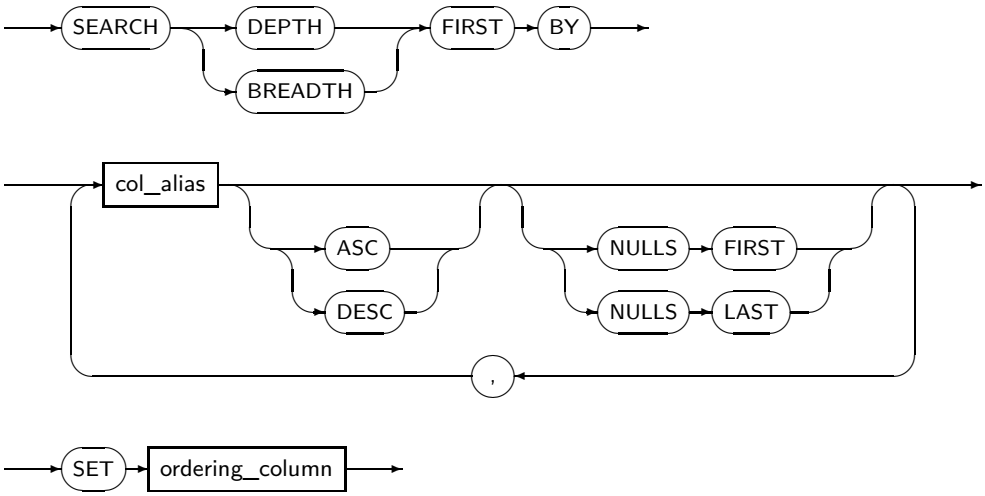
select



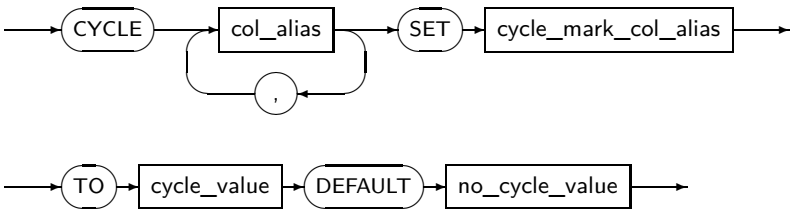
with_clause



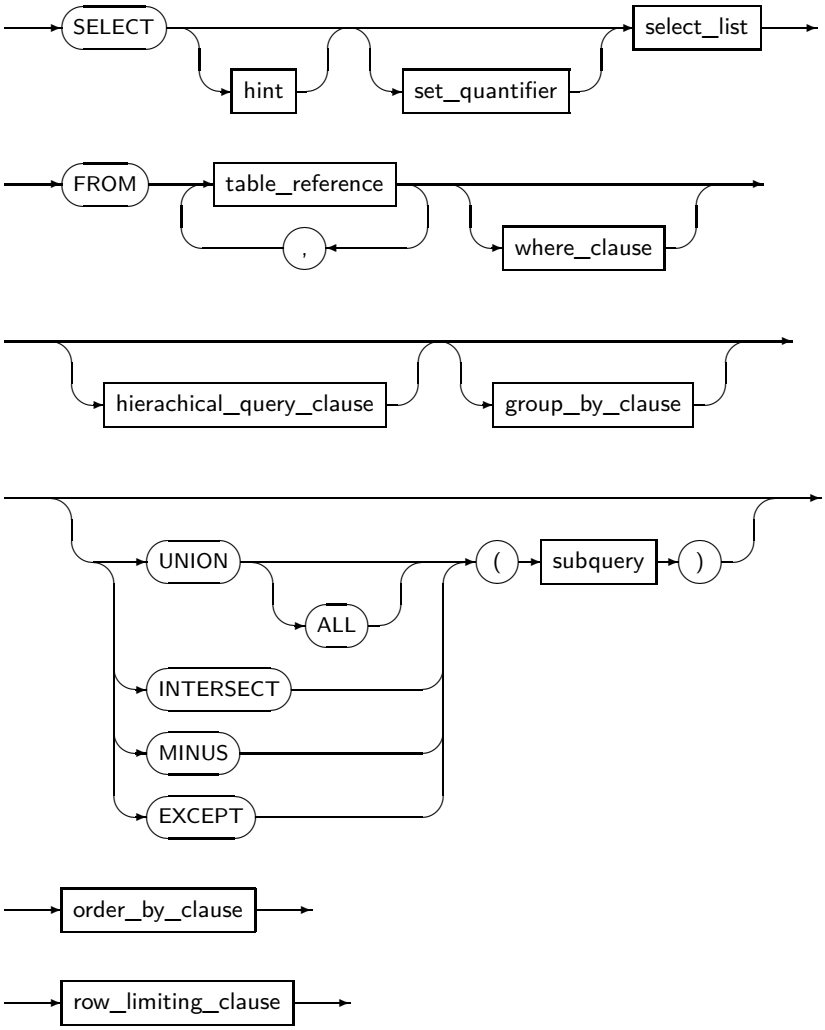
search_clause



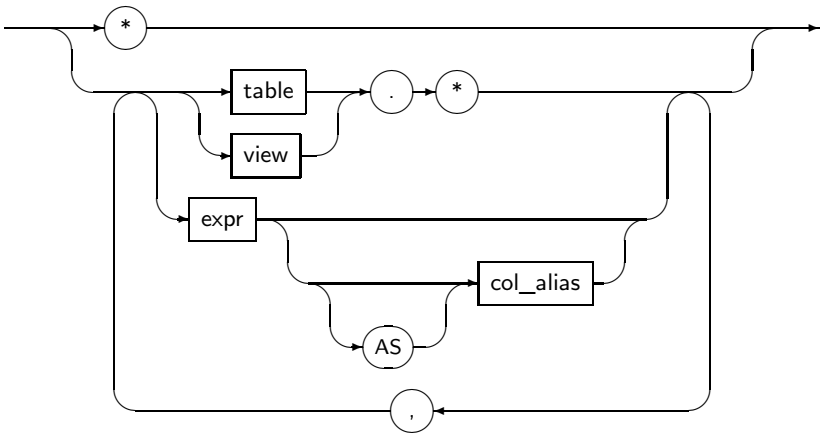
cycle_clause



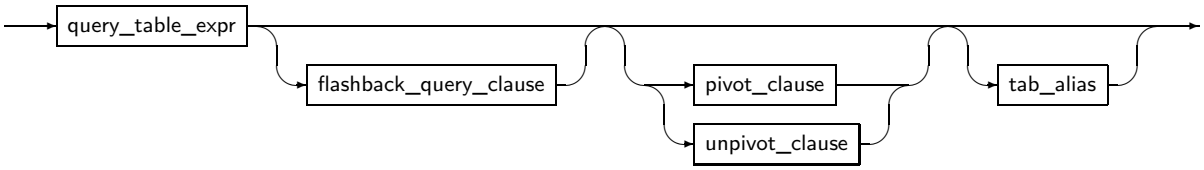
subquery



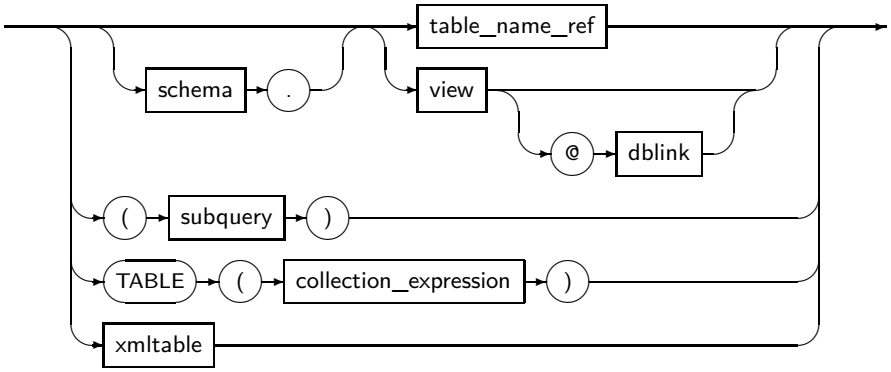
select_list



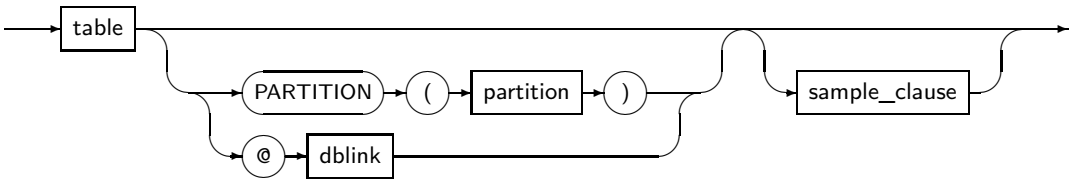
table_reference



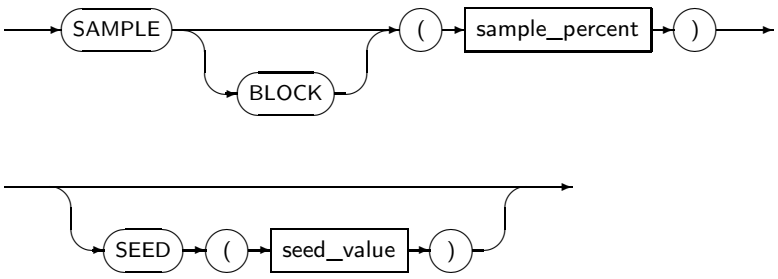
query_table_expr



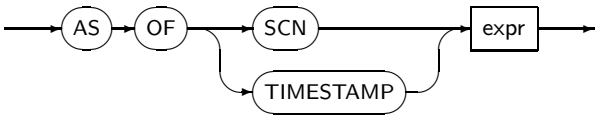
table_name_ref



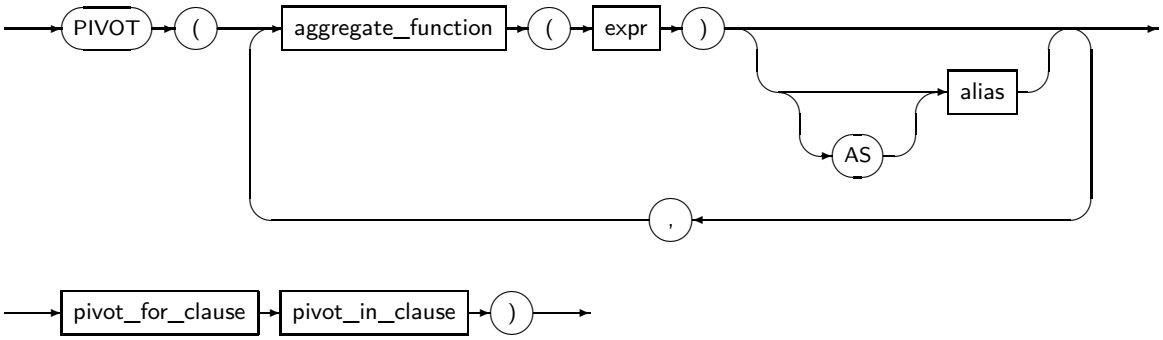
sample_clause



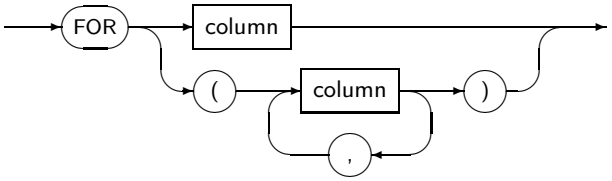
flashback_query_clause



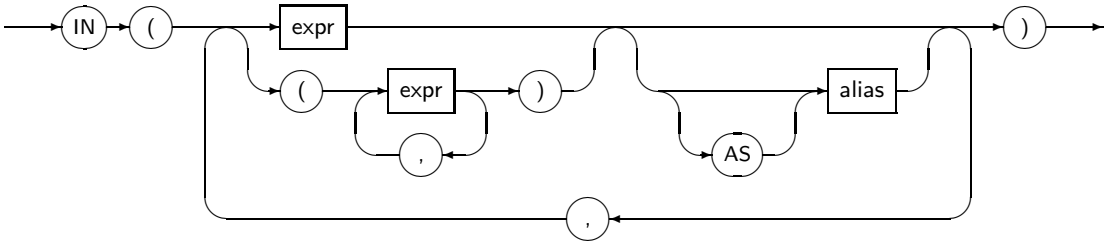
pivot_clause



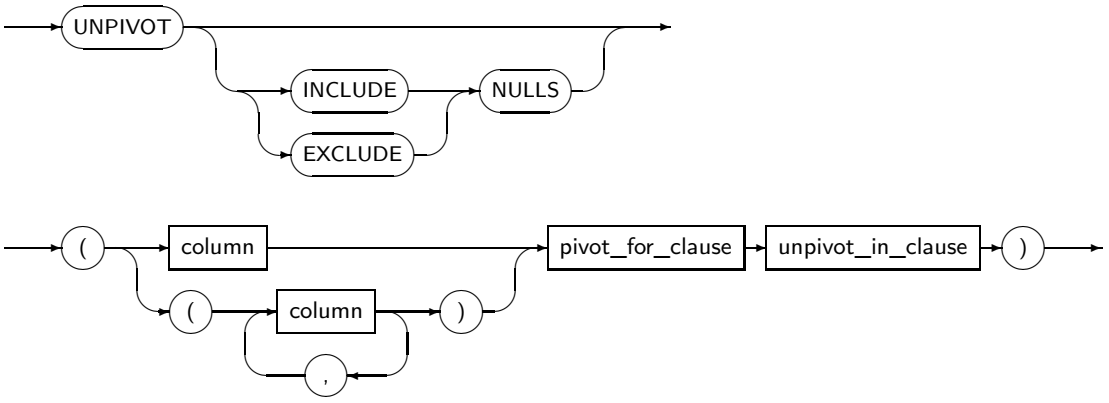
pivot_for_clause



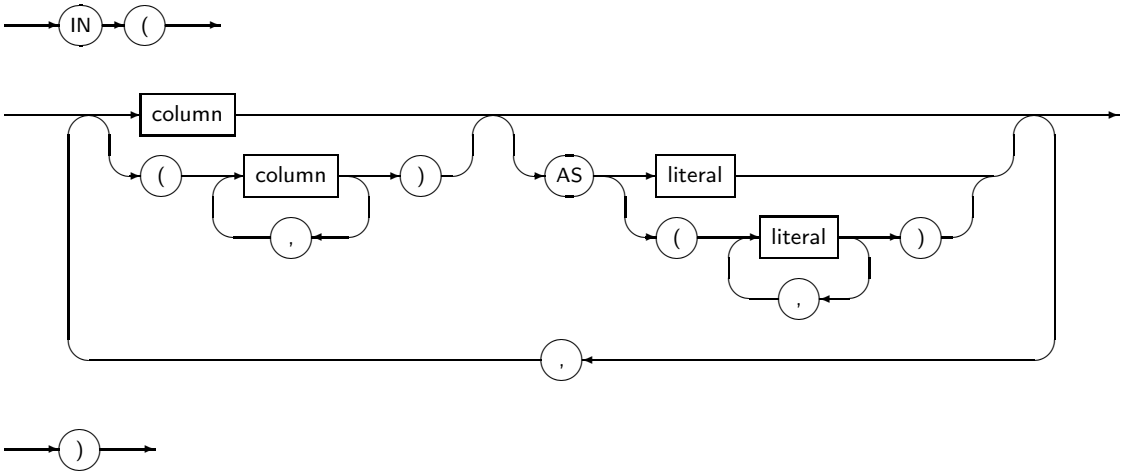
pivot_in_clause



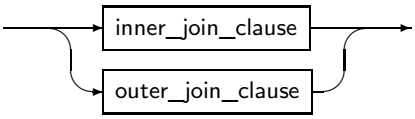
unpivot_clause



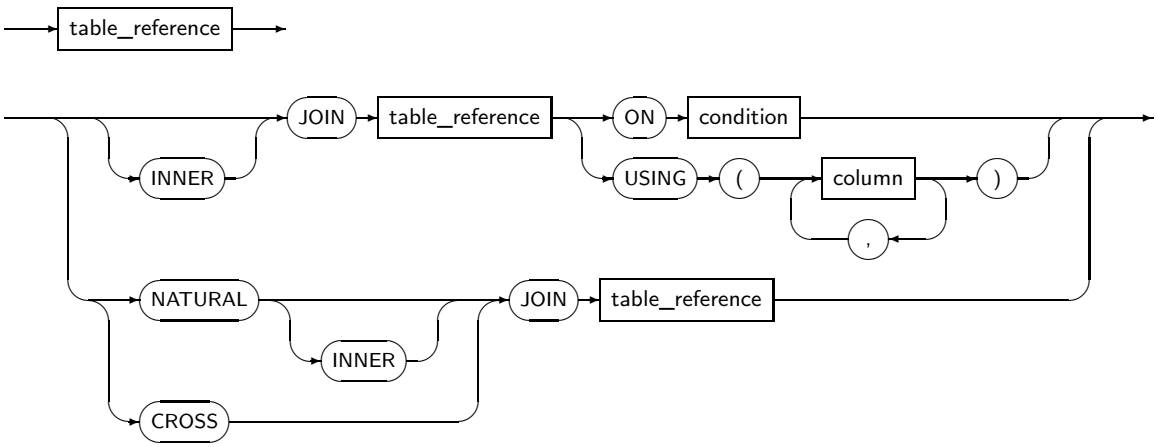
unpivot_in_clause



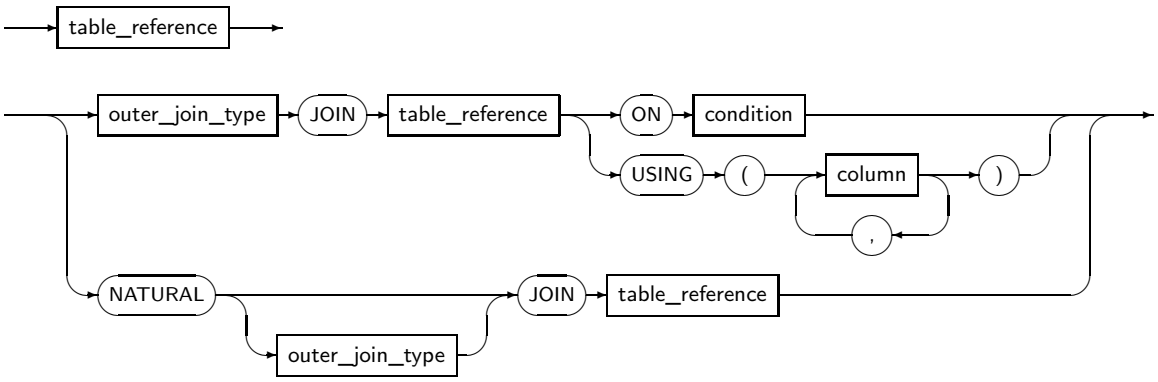
join_clause



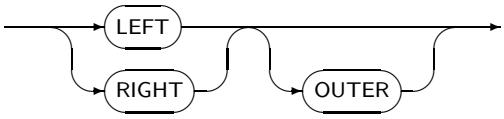
inner_join_clause



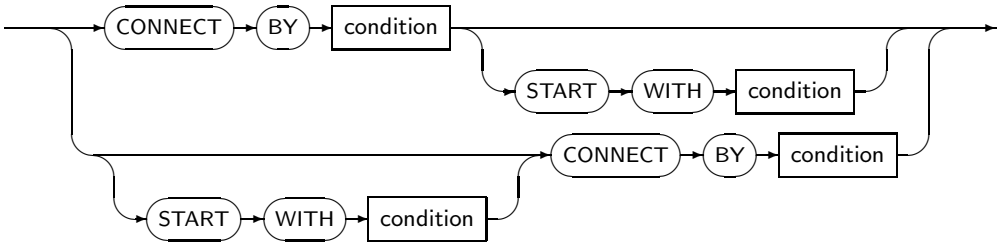
outer_join_clause



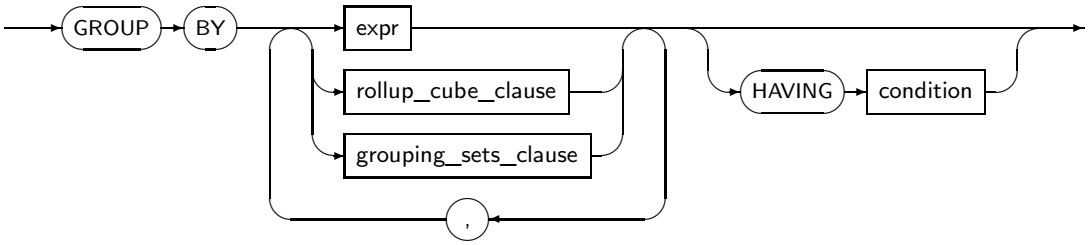
outer_join_type



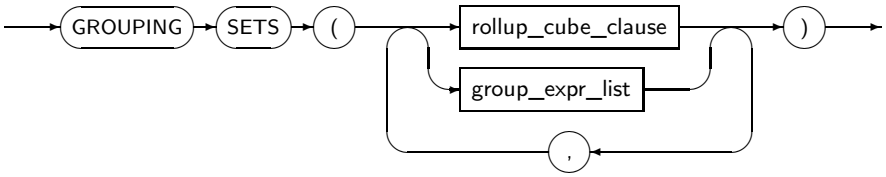
hierachical_query_clause



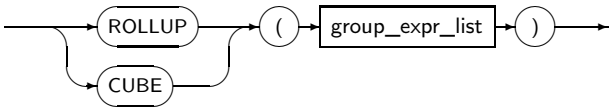
group_by_clause



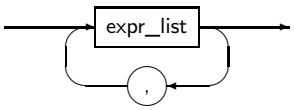
grouping_sets_clause



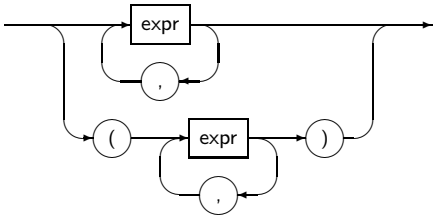
rollup_cube_clause



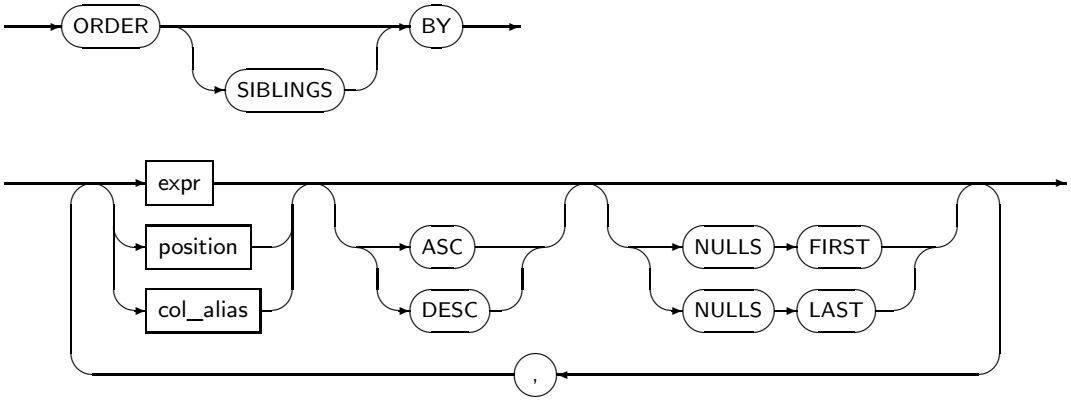
group_expr_list



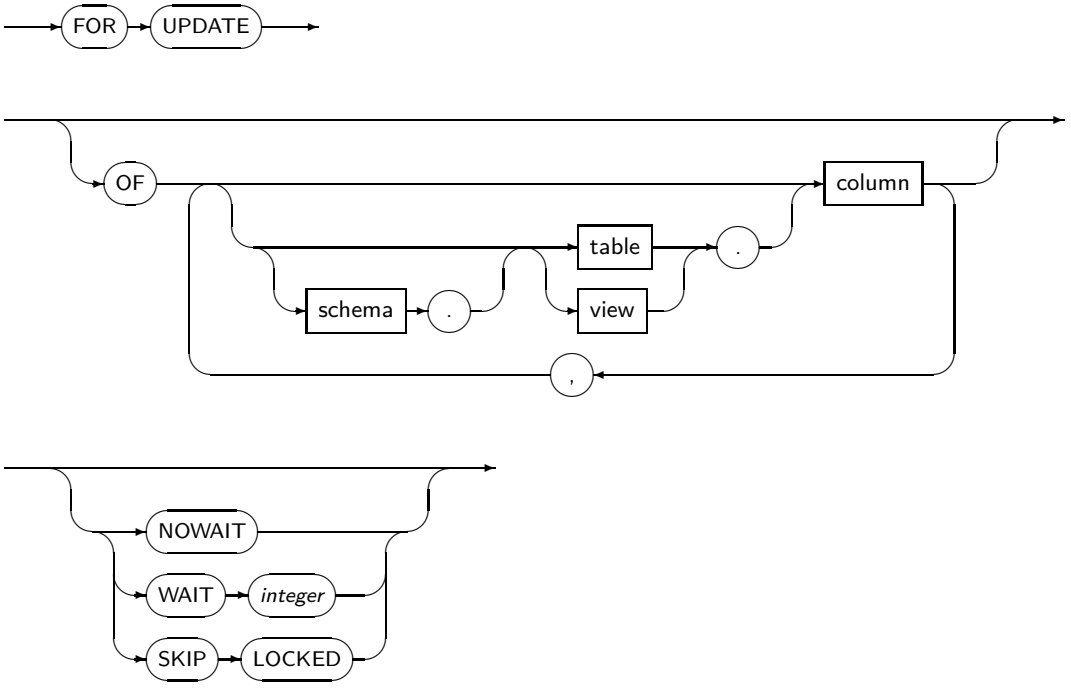
expr_list



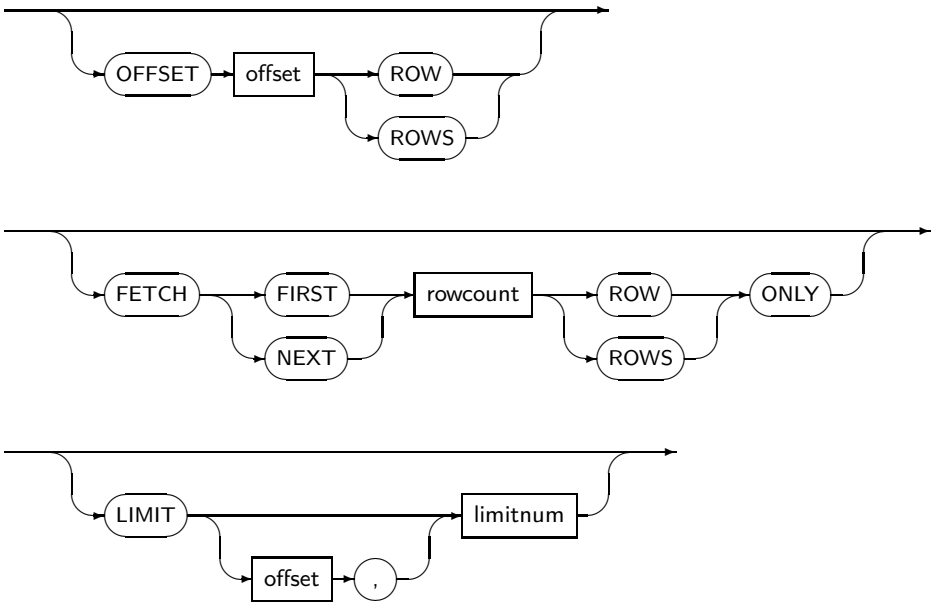
order_by_clause



for_update_clause



row_limiting_clause



- 특권

  SELECT ANY TABLE 시스템 특권을 가진 사용자는 모든 테이블과 모든 뷰를 조회할 수 있다. 테이블을 조회하기 위해서는 테이블을 사용자가 소유하고 있거나, 그 객체에 대해 SELECT 스키마 오브젝트 특권을 가지고 있어야 한다.

  뷰의 기반 테이블을 조회하기 위해서는 다음의 두 가지 조건을 동시에 만족해야 한다.

  – 사용자가 뷰를 가지고 있거나 뷰에 대한 SELECT 스키마 오브젝트 특권이 있어야 한다.

  – 뷰가 속한 스키마의 사용자가 뷰의 기반 테이블을 가지고 있거나 기반 테이블에 대해 SELECT 스키마 오브젝트 특권을 가지고 있어야 한다.

- 구성요소

  – select

구성요소	설명
with_clause	부질의를 정의하고 이름을 만든다.
subquery	질의를 명시한다.
for_update_clause	질의 결과로 반환된 모든 ROW에 잠금을 설정하여 다른 사용자가 읽거나 갱신하지 못하도록 한다. 설정된 잠금은 현재 트랜잭션이 종료될 때까지 계속 유지된다. for_update_clause는 부질의 안에는 포함할 수 없다. 둘 이상의 테이블 또는 뷰에 조인 질의를 수행하는 경우 for_update_clause는 모든 테이블 또는 뷰에 잠금을 설정한다.

구성요소	설명
	for_update_clause는 질의 결과의 ROW가 기반 테이블 내의 유일한 ROW로 결정될 수 없는 질의 문장에는 포함될 수 없다. 따라서, 다음과 같은 질의 문장 안에는 포함할 수 없다. – SELECT 절에 DISTINCT를 포함하는 문장 – 둘 이상의 테이블 또는 뷰에 대한 집합 또는 백 연산을 포함하는 문장 – GROUP BY 절을 포함하거나 SELECT 절에 집단 함수(Aggregate function)를 포함하는 문장

– with_clause

구성요소	설명
query_name	부질의의 이름이다. 주 질의와 현재 부질의 다음에 정의되는 부질의에 query_name을 명시하여 사용한다.
col_alias	subquery의 컬럼명을 재정의한다.
subquery	질의를 명시한다.
search_clause	행의 정렬방식을 지정한다.
cycle_clause	순환 절에 대한 처리 방식을 지정한다.

with_clause의 제약조건은 다음과 같다.

- query_name으로 정의한 부질의 안에서 query_name을 참조하여 재귀 활용이 가능하다.
- 집합 연산자가 있는 복합 쿼리에서는 구성 요소 쿼리의 FROM 절 안에서만 query_name을 사용할 수 있다.
- col_alias에 query_name과 중복되는 명칭을 사용할 수 없다.

– search_clause

구성요소	설명
depth first by	형제 행보다 자식 행이 먼저 반환되기를 원하는 경우 사용한다.
breadth first by	자식 행보다 형제 행이 먼저 반환되기를 원하는 경우 사용한다.
col_alias	정렬하고자 하는 칼럼을 명시한다. 앞선 query_name에 대한 col_alias 목록에 존재하는 칼럼을 지정해야 한다.
ASC	오름차순으로 정렬한다. (기본값)
DESC	내림차순으로 정렬한다.
NULL FIRST	NULL 값의 정렬 순서를 명시한다. NULLS FIRST는 내림차순 정렬의 디폴트로 사용된다.

구성요소	설명
NULL LAST	NULL 값의 정렬 순서를 명시한다. NULLS LAST는 오름차순 정렬의 디폴트로 사용된다.
ordering_column	query_name을 조회하는 쿼리에서 ordering_column을 ORDER BY 절에 사용함으로서 지정한 순서대로 행을 정렬할 수 있다. 앞선 query_name에 대한 col_alias 목록에 본 컬럼이 자동으로 추가된다.

search_clause의 제약조건은 다음과 같다.

- query_name에 대한 col_alias 목록에 존재하는 칼럼 명을 ordering_column 명으로 사용할 수 없다.

– cycle_clause

구성요소	설명
col_alias	순환 여부를 판별할 칼럼을 명시한다. 앞선 query_name에 대한 col_alias 목록에 존재하는 칼럼을 지정해야 한다.
cycle_mark_col_alias	순환 여부에 따른 특정 값을 저장하는 칼럼이다. 앞선 query_name에 대한 col_alias 목록에 본 컬럼이 자동으로 추가된다.
TO cycle_value	행에 대한 순환이 검출된 경우 cycle_mark_col_alias에 cycle_value값이 입력된다. 이 경우 해당 행에 대한 재귀는 중지되며 다른 비순환 행에 대해서는 계속해서 진행한다.
DEFAULT no_cycle_value	행에 대한 순환이 검출되지 않은 경우 cycle_mark_col_alias에 no_cycle_value 값이 입력된다.

cycle_clause의 제약조건은 다음과 같다.

- cycle 절을 생략했으나 순환이 검출되는 경우 재귀 WITH 절은 에러를 출력한다.
- search 절과 함께 사용하는 경우 cycle_mark_col_alias명은 ordering_column명과 같지 않아야 한다.
- query_name에 대한 col_alias 목록에 존재하는 칼럼명을 cycle_mark_col_alias명으로 사용할 수 없다.

– subquery

구성요소	설명
hint	힌트를 사용한다. 힌트는 일종의 지시문으로 최적화기의 특정 행동을 지시하거나 최적화기의 실행 계획을 변경한다. 자세한 내용은 “2.8. 힌트”를 참고한다.
set_quantifier	질의 결과에 중복된 ROW의 허용, 비허용 여부를 지정한다. DISTINCT, UNIQUE, ALL을 지정할 수 있다. – DISTINCT, UNIQUE: 중복된 ROW를 제거한다.

구성요소	설명
	– ALL: 모든 ROW를 선택한다. (기본값)
select_list	질의의 결과로 반환할 연산식을 명시한다. select_list에는 다음과 같은 제약조건이 있다. 만약 group_by_clause가 명시되어 있다면 select_list에서는 다음에 명시된 연산식의 조합만을 사용할 수 있다. – 상수 – 집합 함수 (aggregation function) – group_by_clause에 사용된 연산식으로 조합된 연산식 조인 뷰에서 ROWID를 선택하면 마지막에 명시된 키 보존 테이블의 ROWID가 선택된다. 키 보존 테이블이 없다면, ROWID를 선택할 수 없다. 동일한 컬럼명을 가지는 2개 이상의 테이블 또는 뷰가 조인된 경우 특정 컬럼을 선택하기 위해서는 컬럼명이 테이블명 또는 별칭과 함께 명시되어야 한다.
FROM	질의의 대상이 되는 하나 이상의 테이블, 뷰, 부질의를 명시한다.
table_reference	질의할 테이블, 뷰, 인라인 뷰 등을 명시하고 조인 관계를 설정한다.
where_clause	WHERE 절을 통하여 조건식을 만족하는 ROW만을 검색 할 수 있으며, WHERE 절이 생략된 경우에는 질의의 대상이 되는 객체의 모든 ROW를 반환한다. 자세한 내용은 “3.4. 조건식”을 참고한다.
hierarchical_query_clause	하나의 테이블 또는 조인된 둘 이상의 테이블의 ROW 간의 계층 관계를 정의하여 검색할 수 있다. 이러한 질의를 계층 질의라고 하며 START WITH … CONNECT BY 절을 이용하여 수행할 수 있다. CONNECT BY 절에는 다른 조건식에는 사용되지 않는 PRIOR 연산자가 포함된다. 계층 질의에서 정렬을 하고 싶다면 ORDER SIBLINGS BY 절을 사용한다. 자세한 내용은 “5.5. 계층 질의”를 참고한다.
group_by_clause	질의 결과로 반환된 ROW를 하나 이상의 그룹으로 분리하기 위하여 GROUP BY 절을 이용한다. CUBE 또는 ROLLUP 확장을 사용했을 경우는 ROW를 모아놓은 그룹을 다시 모아 추가로 상위 그룹을 형성할 수 있다. 그룹으로 분리하기 위한 연산식은 하나 이상이 될 수 있으며, HAVING 절을 이용하여 원하는 그룹을 반환할 수 있다. 그룹으로 분리된 결과는 정렬되어 반환되지 않을 수도 있으며, 이러한 경우에 ORDER BY 절을 이용하여 최종 결과를 정렬할 수 있다. GROUP BY 절의 연산식으로 부질의, 대용량 객체형 데이터 타입의 컬럼을 사용할 수 없다.

구성요소	설명
UNION (ALL)	양쪽 질의 결과에서 중복된 ROW를 제거하고 새로운 하나의 질의 결과를 만든다. 예약어 ALL을 명시하면 중복된 ROW를 제거하지 않는다.
INTERSECT	양쪽 질의 결과에 동일하게 나타난 ROW를 모아 새로운 하나의 질의 결과를 만든다.
MINUS	MINUS 예약어 이전에 명시된 질의 결과에서 이후에 명시된 질의 결과에 동일하게 나타나는 ROW를 제거해서 새로운 하나의 질의 결과를 만든다.
EXCEPT	EXCEPT는 MINUS와 동일하다. EXCEPT 예약어 이전에 명시된 질의 결과에서 이후에 명시된 질의 결과에 동일하게 나타나는 ROW를 제거해서 새로운 하나의 질의 결과를 만든다.
order_by_clause	검색 결과의 ROW를 정렬하여 반환하기 위해 ORDER BY 절을 이용한다. 정렬 순서를 지정하기 위해 컬럼의 이름을 포함하는 연산식이 올 수 있으며, 특정 컬럼의 위치 또는 컬럼의 별칭을 이용할 수 있다. 정렬 순서를 지정하기 위해 여러 가지 기준이 지정된 경우에는 먼저 지정된 기준에 따라 정렬을 수행하고 순서를 가리기 불가능한 경우에 다음 기준을 이용하게 된다.
row_limiting_clause	검색 결과의 수를 제한하기 위해 OFFSET, FETCH, LIMIT 절을 이용한다. – OFFSET 절: 전체 검색 결과 중 시작되는 ROW의 NUMBER를 지정할 수 있다. – FETCH 절: 몇 개의 ROW를 가져올 지 결정할 수 있다. – LIMIT 절: 쿼리의 수행 결과 중 원하는 ROW의 데이터만 가져올 수 있다.

order_by_clause의 제약조건은 다음과 같다.

- select_list에 예약어 DISTINCT가 사용되면 order_by_clause에서 select_list에 명시된 expr의 조합만을 정렬의 키로 사용할 수 있다.
- LOB, LONG 등과 같은 대용량 객체형 데이터 타입의 컬럼을 사용할 수 없다.
- group_by_clause가 명시되어 있다면 아래의 4가지 expr만 order_by_clause에서 사용할 수 있다.

개수	가능한 expr
1	상수
2	집합 함수
3	분석 함수
4	group_by_clause에 사용된 expr의 조합

– select_list

구성요소	설명
애스터리스크(*)	FROM 절에 명시된 테이블과 뷰의 모든 컬럼을 선택한다.
table.* , view.*	명시된 테이블 또는 뷰의 모든 컬럼을 선택한다.
expr	반환할 값을 계산하는 연산식이다.
AS	별칭을 명시할 때 사용하는 예약어로 생략할 수 있다.
col_alias	select_list에 명시된 연산식에 별칭을 설정한다. 동일한 질의에서는 order_by_clause에만 별칭을 사용할 수 있다.

– table_reference

구성요소	설명
query_table_expr	조회할 스키마 객체를 명시한다.
flashback_query_clause	테이블, 뷰 또는 부질의의 예전 데이터를 보기 위해 명시한다.
tab_alias	테이블, 뷰 또는 부질의에 대한 별칭을 명시한다.
join_clause	FROM 절에 명시된 테이블, 뷰, 부질의 간의 조인 관계를 명시한다.

– query_table_expr

구성요소	설명
schema	테이블이나 뷰가 속한 스키마를 명시한다. 생략하면 현재 사용자의 스키마로 인식된다.
table	테이블의 이름을 명시한다.
PARTITION (partition)	특정 파티션의 이름을 명시한다. 테이블 전체가 아니라 명시된 파티션만 읽는다.
dblink	데이터베이스 링크의 이름 전체 또는 부분을 명시한다. 데이터베이스 링크를 명시할 때는 반드시 앞에 '@'를 붙여야 한다. 링크는 다음과 같은 제약조건이 있다. – 원격 테이블에서는 Tibero에서 지원하지 않는 사용자 정의 타입 또는 REF 객체에 대한 질의를 할 수 없다. – 원격 테이블에서는 Tibero에서 지원하지 않는 ANYTYPE, ANYDATA 또는 ANYDATASET 타입의 컬럼에 대한 질의를 할 수 없다.
view	뷰의 이름을 명시한다.
sample_clause	테이블의 데이터를 무작위로 일부만 읽도록 한다.
subquery	부질의를 명시한다.

구성요소	설명
collection_expression	파이프라인드 테이블 함수(Pipelined Table Function)을 명시한다. 함수의 결과값을 테이블처럼 이용할 수 있게 해준다. 자세한 설명과 예제는 "Tibero tbPSM 안내서"의 "제9장 파이프라인드 테이블 함수"를 참조한다.
xmltable	XMLTABLE 함수를 명시한다.

– sample_clause

구성요소	설명
BLOCK	키워드를 명시하면 데이터 블록을 샘플링하고 명시하지 않으면 ROW를 샘플링한다.
sample_percent	샘플링할 ROW 또는 블록의 비율을 설정한다. 이 값은 .000001부터 100 사이의 값을 가진다. 100은 포함하지 않는다.
seed_value	이 값이 같다면 다음 수행에서 동일한 샘플링 데이터를 취한다. 명시하지 않으면 다음 수행할 때 다른 데이터를 얻게된다. 이 값은 0부터 4294967295 사이의 값을 가진다.

– flashback_query_clause

구성요소	설명
expr	예전 시점을 나타내는 표현식이다. – SCN 또는 TIMESTAMP를 통해 특정 시점을 명시할 수 있다. – AS OF SCN을 사용하면 expr은 NUMBER 값이어야 한다. – AS OF TIMESTAMP를 사용하면 expr은 TIMESTAMP 값이어야 한다.

flashback_query_clause는 다음과 같은 제약조건이 있다.

- expr에 컬럼, 부질의를 사용할 수 없다.
- with_clause에 정의된 부질의를 참조하려고 table_reference에 사용된 query_name에는 사용할 수 없다.

– pivot_clause

피봇은 테이블의 행을 열로 바꾼다. 동일 그룹에 속한 행에 집합함수를 적용해서 새로운 열의 값으로 사용한다. 집합함수를 적용할 그룹을 결정하기 위한 GROUP BY 절은 사용하지 않는다. pivot_clause에 명시되지 않은 query_table_expr의 컬럼이 묵시적으로 GROUP BY 표현식으로 사용된다.

구성요소	설명
aggregate_function	컬럼의 값을 생성할 집합 함수를 명시한다.

구성요소	설명
	alias를 명시하면 pivot_in_clause에서 생성된 컬럼명에 '_'와 alias를 추가로 붙여서 새로운 컬럼명을 생성한다.
pivot_for_clause	피봇할 컬럼을 명시한다.
pivot_in_clause	피봇할 컬럼의 값을 명시한다. 명시된 개수 많큼 새로운 컬럼이 만들어지고 값 별로 집함 함수를 적용해서 컬럼값을 계산한다. 명시되지 않은 값은 집함 함수에 적용되지 않는다. alias를 사용하지 않으면 명시된 값을 컬럼명으로 사용한다.

– unpivot_clause

UNPIVOT은 테이블의 열을 행으로 바꾼다. 한 로우의 여러 컬럼을 여러 로우의 동일 컬럼으로 변환한다.

구성요소	설명
INCLUDE|EXCLUDE NULLS	INCLUDE NULLS는 NULL 값을 포함해서 로우를 생성한다. EXCLUDE NULLS이 명시되면 NULL 값으로 이루어진 로우를 생성하지 않는다. 명시하지 않으면 EXCLUDE NULLS가 기본값이다.
column	UNPIVOT된 쿼리 결과에 생성되는 측정값 컬럼의 컬럼명을 명시한다. UNPIVOT을 수행해서 보고자 하는 결과값을 측정값이라고 한다. 동일한 타입이거나 최종 결과 타입으로 변환이 가능한 타입이어야 한다.
pivot_for_clause	UNPIVOT된 쿼리 결과에 생성되는 설명값 컬럼의 컬럼명을 명시한다. 이 컬럼에 채워지는 값은 측정값을 설명하는 용도로 사용된다.
unpivot_in_clause	column의 값은 측정값 컬럼의 값이 된다. literal을 명시하면 설명값 컬럼의 값으로 사용된다. 명시하지 않으면 컬럼명을 값으로 사용한다.

– join_clause

구성요소	설명
inner_join_clause	내부 조인을 생성한다. 내부 조인 조건을 만족하는 ROW의 조인 결과만 조인된다. 내부(Inner)조인과 자연(Natural) 조인을 명시할 수 있다. 명시하지 않는다면 내부 조인이 된다.
outer_join_clause	조인 조건에 맞는 모든 ROW와 한 쪽 테이블에서 조인 조건을 만족하지 않는 ROW가 선택된다.

구성요소	설명
	조건을 만족하지 못한 ROW는 모든 컬럼이 NULL 값을 가지는 ROW와 조인되어 선택된다.

– inner_join_clause

구성요소	설명
INNER	내부 조인을 명시한다. ON 절이나 USING 절에 의해 생성된 조인 조건을 만족하는 ROW만 조인된다.
ON condition	조인 조건을 명시한다.
USING column	두 테이블에서 동일한 이름을 가지는 컬럼을 명시한다. 명시된 컬럼명을 가지는 양 쪽 컬럼에 대해서 각각 동등 비교 조건을 모두 만족해야 한다는 조인 조건이 생성된다. 조인된 조건으로 사용된 컬럼은 조인된 ROW에 하나만 포함되고 테이블 이름이나 별칭과 함께 참조할 수 없다.
NATURAL	자연 조인을 명시한다. 두 테이블에서 동일한 이름을 갖는 컬럼은 각각 동등 비교 조건을 모두 만족해야 한다는 조인 조건이 생성된다. 조인 조건으로 사용된 컬럼은 조인된 ROW에 하나만 포함되고 테이블 이름이나 별칭과 함께 참조할 수 없다.
CROSS	크로스 조인을 생성한다. 조인 되는 두 테이블의 모든 ROW가 조인된다.

– outer_join_clause

옵션	설명
outer_join_type	외부 조인의 타입을 명시한다.
ON condition	조인 조건을 명시한다.
USING column	두 테이블에서 동일한 이름을 가지는 컬럼을 명시한다. 명시된 컬럼명을 가지는 양 쪽 컬럼에 대해서 각각 동등 비교 조건을 모두 만족해야 한다는 조인 조건이 생성된다. 조인된 조인 조건으로 사용된 컬럼은 조인된 ROW에 하나만 포함되고 테이블 이름이나 별칭과 함께 참조할 수 없다.
NATURAL	자연 조인을 명시한다. 두 테이블에서 동일한 이름을 갖는 컬럼은 각각 동등 비교 조건을 모두 만족해야 한다는 조인 조건이 생성된다. 조인 조건으로 사용된 컬럼은 조인된 ROW에 하나만 포함되고 테이블 이름이나 별칭과 함께 참조할 수 없다.

– outer_join_type

구성요소	설명
LEFT	왼쪽 외부 조인을 명시한다. 왼쪽 테이블의 모든 컬럼이 선택된다.
RIGHT	오른쪽 외부 조인을 명시한다. 오른쪽 테이블의 모든 컬럼이 선택된다.
OUTER	생략할 수 있으며, 특별한 의미는 없다.

– cross_join_clause

구성요소	설명
table_reference	조인할 테이블, 뷰, 인라인 뷰 등을 명시한다.

– hierarchical_query_clause

구성요소	설명
CONNECT BY	ROW의 상하 관계를 정의하는 조건식을 명시한다.
START WITH	계층 내의 루트 ROW를 지정하기 위한 조건식을 명시한다.
condition	조건식을 명시한다.

– group_by_clause

구성요소	설명
expr	그룹을 분리하기 위한 연산식을 명시한다.
rollup_cube_clause	ROLLUP과 CUBE 연산을 명시한다.
grouping_sets_clause	GROUPING SETS을 명시한다.
HAVING condition	원하는 그룹만 반환하도록 하는 조건식을 명시한다.

– grouping_sets_clause

구성요소	설명
GROUPING SETS	GROUPING SETS 뒤에 명시된 여러 개의 연산식을 바탕으로 선택된 ROW를 그룹으로 나누고, 각 그룹에 하나의 요약 정보 ROW를 반환한다. CUBE 또는 ROLLUP은 모든 연산식 조합에 대하여 그룹을 나누지만 GROUPING SETS를 사용하면 원하는 연산식만으로 그룹을 나누기 때문에 좀 더 효과적이다. 여러 개의 연산식에 대하여 GROUP BY를 수행한 결과에 UNION ALL을 한 것과 같은 결과를 갖기 때문에 중복된 ROW를 만들어 낼 수 있다. GROUPING SETS 뒤에 CUBE 또는 ROLLUP을 연산식 조합으로 명시 할 수 있는데 이런 경우에는 CUBE 또는 ROLLUP의 모든 연산식 조합을 직접 풀어 GROUPING SETS 뒤에 명시한 것과 같은 결과를 갖는다.

구성요소	설명
	GROUPING SETS 뒤에 여러 개의 CUBE 또는 ROLLUP을 사용한 경우에는 각 연산식 조합의 cross product로 모든 연산식 조합을 계산하여 수행한다.

– rollup_cube_clause

구성요소	설명
ROLLUP	질의에서 선택된 ROW를 ROLLUP 뒤의 n개의 연산식 중 앞에서부터 n, n-1, n-2, ...0 개의 연산식을 기반으로 하여 그룹으로 나누고, 각각의 그룹에 하나의 ROW를 반환한다. 예를 들어 SUM 집단 함수와 같이 사용했을 경우는 중간 중간의 소계를 계산하기 위한 방법으로 이것을 사용할 수 있다. 이 경우 SUM은 가장 아래 레벨의 소계에서부터 전체 총계까지를 모두 계산해 낸다.
CUBE	CUBE 뒤에 명시된 연산식의 가능한 모든 조합으로 선택된 ROW를 그룹으로 나누고, 각 그룹에 하나의 요약 정보 ROW를 반환한다. CUBE를 사용하여 교차표 값을 생성할 수 있다. 요약 정보는 그룹별 결과가 우선 출력되고, 그룹들의 요약 정보를 합친 결과가 나오는 순으로 출력되는데, 최종 결과 ROW는 전체의 요약 정보가 나오게 된다. 하지만, 반대로 전체 요약 정보부터 그룹들의 요약 정보, 그룹별 요약 정보 순으로 결과를 출력하고 싶은 경우에는 SUMMARY_FIRST_IN_GROUP BY_CUBE 파라미터를 켜줌으로써 출력 순서를 변경할 수가 있다.
group_expr_list	리스트 연산식으로 구성된 묶음이다.

– group_expr_list

구성요소	설명
expr_list	리스트 연산식이다.

– order_by_clause

구성요소	설명
SIBLINGS	hierarchical_query_clause가 명시된 질의에 사용할 수 있다. order_by_clause는 계층 질의의 형제 노드 내에서 정렬 순서를 정의하게 된다.
expr	정렬의 키로 사용되는 연산식이다.
position	select_list에 명시된 expr의 위치를 지정한다. 해당 위치의 expr이 정렬에 사용된다. 정수 값을 사용해야 한다.

구성요소	설명
col_alias	컬럼의 별칭을 명시한다.
ASC	오름차순으로 정렬한다. (기본값)
DESC	내림차순으로 정렬한다. 이 부분을 생략하면 ASC로 인식한다.
NULLS FIRST	NULL 값의 정렬 순서를 명시한다. NULLS FIRST는 내림차순 정렬의 디폴트로 사용된다.
NULLS LAST	NULL 값의 정렬 순서를 명시한다. NULLS LAST는 오름차순 정렬의 디폴트로 사용된다.

– for_update_clause

구성요소	설명
OF column	일부 테이블 또는 뷰에만 잠금을 설정하고자 할 때에 사용한다. OF 예약어 뒤에 잠금을 설정하고자 하는 테이블의 컬럼을 나열한다. 이때, 컬럼의 별칭은 사용할 수 없다.
schema	스키마의 이름을 명시한다. 생략하면 현재 사용자의 스키마로 인식된다.
table	테이블의 이름을 명시한다.
view	뷰의 이름을 명시한다.
NOWAIT	해당 ROW에 다른 사용자가 설정한 잠금이 있어도 해제될 때까지 기다리지 않는다.
WAIT	해당 ROW에 다른 사용자가 설정한 잠금이 있는 경우 해제될 때까지 integer 만큼의 시간(초) 동안 시도한다. 지정되지 않으면 잠금이 해제될 때까지 기다린다.
SKIP LOCKED	해당 ROW에 다른 사용자가 설정한 잠금이 있는 경우 해당 ROW를 건너뛰고 다음 ROW로 넘어간다.

– row_limiting_clause

구성요소	설명
OFFSET	OFFSET 키워드를 사용하여 몇 개의 row를 건너뛸 것인지 결정한다. offset은 숫자 혹은 숫자 값으로 표현되는 식이어야 한다. 이 키워드를 지정하지 않으면 0으로 간주되어 첫 번째 row부터 결과 집합을 전달한다.
ROW | ROWS	두 키워드는 의미적으로 명확성만 제공해줄 뿐 상호 교환하여 사용 가능하다.
FETCH	전달할 결과 집합의 row 수를 지정한다. 이 키워드를 지정하지 않으면 offset + 1 행부터 시작하여 모든 결과 집합을 전달한다.

구성요소	설명
FIRST | NEXT	두 키워드는 의미적으로 명확성만 제공해줄 뿐 상호 교환하여 사용 가능하다.
rowcount	rowcount는 숫자 혹은 숫자 값으로 표현되는 식이어야 한다. rowcount가 offset + 1부터 시작해서 전달할 수 있는 모든 결과 집합의 row 수보다 클 경우 모든 가능한 결과 집합을 전달한다.
ROW | ROWS	두 키워드는 의미적으로 명확성만 제공해줄 뿐 상호 교환하여 사용 가능하다.
ONLY	지정한 row 수만 정확하게 반환할 때 사용한다.
LIMIT	LIMIT 키워드를 사용하여 쿼리의 수행 결과 중 원하는 행의 데이터만 가져올 수 있다.
offset	쿼리의 수행 결과 중 가져올 행의 시작점을 의미한다. 쿼리 수행 결과의 첫 번째 행의 offset 값은 0이다.
limitnum	지정한 offset을 포함하여 limitnum의 값만큼의 행을 가져온다. offset이 지정되어 있지 않다면 offset 값을 0으로 간주한다.

– 집합 연산자

2개의 SELECT 문의 결과를 하나의 결과 집합으로 결합하는데 사용된다.

select_list에 대응되는 컬럼의 타입과 개수는 일치해야 한다. 컬럼의 길이는 다를 수 있다. 참조되는 컬럼의 이름은 제일 좌측에 명시된 SELECT문의 select_list 절이 사용된다. SELECT 문이 집합 연산자에 의해 결합되는 경우 왼쪽에서 오른쪽 순서로 질의를 수행하게 된다. 자세한 내용은 “5.4. 집합 연산자”를 참고한다.

- 예제

다음은 **SELECT**를 사용하는 예이다.

```
SELECT * FROM EMP;

SELECT ENAME, SALARY * 1.05 FROM EMP WHERE DEPTNO = 5;

SELECT ENAME, SALARY, DEPT.* FROM EMP, DEPT WHERE EMP.DEPTNO = DEPT.DEPTNO;

SELECT ENAME, SALARY, LOC FROM EMP NATURAL JOIN DEPT;

SELECT ENAME FROM EMP WHERE DEPTNO = 5
  UNION SELECT ENAME FROM EMP WHERE DEPTNO = 7;

SELECT DEPTNO, MAX(SALARY) FROM EMP GROUP BY DEPTNO HAVING DEPTNO >= 5;

SELECT DISTINCT DEPTNO FROM EMP;
```

```
SELECT * FROM EMP WHERE DEPTNO = 5 FOR UPDATE;

SELECT ENAME, D.* FROM EMP E, DEPT D WHERE E.DEPTNO = D.DEPTNO FOR UPDATE OF LOC;
```

다음은 비순환 구조에서 **WITH 절**을 재귀적으로 사용하는 예이다.

```
create table t (
    id number,
    name varchar2(100),
    parent varchar2(10),
    etc varchar2(9)
);

insert into t values (1,'DB', NULL, NULL);
insert into t values (10, 'DB1', 'DB', 'F');
insert into t values (30, 'DB2', 'DB', 'C');
insert into t values (11, 'DB1-2', 'DB1', 'F3');
insert into t values (12, 'DB1-1', 'DB1', 'F2');
insert into t values (15, 'DB2-1', 'DB2', 'C1');
insert into t values (21, 'DB2-2', 'DB2', 'C2');

commit;

column rname format a20;
column name_tree format a40;

set linesize 180;
set pagesize 100;

with recursive(recursive_level, id, name, name_tree, p_name, post_team) as
(
    select 1 recursive_level,
    id,
    name,
    name,
    parent,
    etc
    from t
    where id = 1
    union all
    select recursive.recursive_level + 1,
    t1.id,
    t1.name,
    recursive.name_tree || '/' || t1.name,
    parent,
    etc
```

```
    from t t1, recursive
    where t1.parent = recursive.name
)
search depth first by name set idx
cycle p_name set iscycle to 'Y' DEFAULT 'N'
select
recursive_level as lv,
lpad(' ', 2 * (recursive_level - 1), ' ') || name as rname,
p_name,
name_tree,
iscycle,
idx
from recursive;

 LV RNAME           P_NAME NAME_TREE                       ISCYCLE    IDX
--- --------------- ------ ------------------------------- ------- ------
  1 DB                     DB                              N            1
  2   DB1           DB     DB/DB1                          N            2
  3     DB1-1       DB1    DB/DB1/DB1-1                    N            3
  3     DB1-2       DB1    DB/DB1/DB1-2                    N            4
  2   DB2           DB     DB/DB2                          N            5
  3     DB2-1       DB2    DB/DB2/DB2-1                    N            6
  3     DB2-2       DB2    DB/DB2/DB2-2                    N            7

7 rows selected.
```

다음은 순환 구조에서 **WITH 절**을 재귀적으로 사용하는 예이다.

```
create table t (
    id number,
    name varchar2(100),
    parent varchar2(10),
    etc varchar2(9)
);

insert into t values (1,'DB', NULL, NULL);
insert into t values (10, 'DB1', 'DB', 'F');
insert into t values (30, 'DB2', 'DB', 'C');
insert into t values (11, 'DB1-2', 'DB1', 'F3');
insert into t values (12, 'DB1-1', 'DB1', 'F2');
insert into t values (15, 'DB2-1', 'DB2', 'C1');
insert into t values (21, 'DB2-2', 'DB2', 'C2');

-- make cycle
insert into t values (40, 'DB2', 'DB1-1', 'C5, B4');
insert into t values (41, 'DB1', 'DB2-1', 'C5, B4');
```

```
commit;

column rname format a20;
column name_tree format a40;

set linesize 180;
set pagesize 100;

with recursive(recursive_level, id, name, name_tree, p_name, post_team) as
(
    select 1 recursive_level,
    id,
    name,
    name,
    parent,
    etc
    from t
    where id = 1
    union all
    select recursive.recursive_level + 1,
    t1.id,
    t1.name,
    recursive.name_tree || '/' || t1.name,
    parent,
    etc
    from t t1, recursive
    where t1.parent = recursive.name
)
search depth first by name set idx
cycle p_name set iscycle to 'Y' DEFAULT 'N'
select
recursive_level as lv,
lpad(' ', 2 * (recursive_level - 1), ' ') || name as rname,
p_name,
name_tree,
iscycle,
idx
from recursive;

 LV RNAME              P_NAME NAME_TREE                          ISCYCLE   IDX
--- ------------------ ------ ---------------------------------- ------- -----
  1 DB                        DB                                 N           1
  2   DB1              DB     DB/DB1                             N           2
  3     DB1-1          DB1    DB/DB1/DB1-1                       N           3
  4       DB2          DB1-1  DB/DB1/DB1-1/DB2                   N           4
  5         DB2-1      DB2    DB/DB1/DB1-1/DB2/DB2-1             N           5
```

```
  6           DB1      DB2-1  DB/DB1/DB1-1/DB2/DB2-1/DB1        N           6
  7             DB1-1  DB1    DB/DB1/DB1-1/DB2/DB2-1/DB1/DB1-1  Y           7
  7             DB1-2  DB1    DB/DB1/DB1-1/DB2/DB2-1/DB1/DB1-2  Y           8
  5         DB2-2      DB2    DB/DB1/DB1-1/DB2/DB2-2            N           9
  3     DB1-2          DB1    DB/DB1/DB1-2                      N          10
  2   DB2              DB     DB/DB2                            N          11
  3     DB2-1          DB2    DB/DB2/DB2-1                      N          12
  4       DB1          DB2-1  DB/DB2/DB2-1/DB1                  N          13
  5         DB1-1      DB1    DB/DB2/DB2-1/DB1/DB1-1            N          14
  6           DB2      DB1-1  DB/DB2/DB2-1/DB1/DB1-1/DB2        N          15
  7             DB2-1  DB2    DB/DB2/DB2-1/DB1/DB1-1/DB2/DB2-1  Y          16
  7             DB2-2  DB2    DB/DB2/DB2-1/DB1/DB1-1/DB2/DB2-2  Y          17
  5         DB1-2      DB1    DB/DB2/DB2-1/DB1/DB1-2            N          18
  3     DB2-2          DB2    DB/DB2/DB2-2                      N          19

19 rows selected.
```

다음은 시간을 사용해서 **FLASHBACK** 쿼리를 수행한 예이다.

```
SQL> create table todo (item varchar2(20), duedate date);

Table 'TODO' created.

SQL> insert into todo values ('결혼식 참석', '2012-04-01');

1 row inserted.

SQL> insert into todo values ('장보기', '2011-12-28');

1 row inserted.

SQL> commit;

Commit completed.

SQL> select systimestamp from dual;

SYSTIMESTAMP
---------------------------------------------------------------------
2011/12/28 18:41:33.854889

1 row selected.

SQL> select * from todo;

ITEM                 DUEDATE
```

```
-------------------- --------------------------------
결혼식 참석          2012/04/01
장보기               2011/12/28

2 rows selected.

SQL> insert into todo values ('신문대금납부', '2011-12-31');

1 row inserted.

SQL>  commit;

Commit completed.

SQL> select * from todo;

ITEM                 DUEDATE
-------------------- --------------------------------
결혼식 참석          2012/04/01
장보기               2011/12/28
신문대금납부         2011/12/31

3 rows selected.

SQL> select * from todo as of timestamp '2011/12/28 18:41:33.854889';

ITEM                 DUEDATE
-------------------- --------------------------------
결혼식 참석          2012/04/01
장보기               2011/12/28

2 rows selected.
```

다음은 TSN을 사용해서 **FLASHBACK** 쿼리를 수행한 예이다. 동적뷰 V$TSN_TIME을 이용하면 TSN과 시간의 맵핑 정보를 알 수 있다.

```
SQL> create table todo (item varchar2(20), duedate date);

Table 'TODO' created.

SQL> insert into todo values ('결혼식 참석', '2012-04-01');

1 row inserted.

SQL> insert into todo values ('장보기', '2011-12-28');
```

```
1 row inserted.

SQL> commit;

Commit completed.

SQL> select systimestamp from dual;

SYSTIMESTAMP
---------------------------------------------------------------
2011/12/29 09:48:35.494024

1 row selected.

SQL> select * from todo;

ITEM                 DUEDATE
-------------------- --------------------------------
결혼식 참석          2012/04/01
장보기               2011/12/28

2 rows selected.

SQL> insert into todo values ('신문대금납부', '2011-12-31');

1 row inserted.

SQL> commit;

Commit completed.

SQL> select systimestamp from dual;

SYSTIMESTAMP
---------------------------------------------------------------
2011/12/29 09:49:05.635549

1 row selected.

SQL> select * from todo;

ITEM                 DUEDATE
-------------------- --------------------------------
결혼식 참석          2012/04/01
장보기               2011/12/28
신문대금납부         2011/12/31
```

```
3 rows selected.

SQL> select * from v$tsn_time where time >= '2011/12/29 09:48:35.494024'
and time <= '2011/12/29 09:49:05.635549';

       TSN TIME
---------- --------------------------------------------------------------
     12347 2011/12/29 09:48:36.018269
     12347 2011/12/29 09:48:37.019113
     12348 2011/12/29 09:48:38.020786
     12348 2011/12/29 09:48:39.021757
     12348 2011/12/29 09:48:40.022545
     12349 2011/12/29 09:48:42.023222
     12350 2011/12/29 09:48:43.722855
     12350 2011/12/29 09:48:45.023568
     12350 2011/12/29 09:48:46.024370
     12351 2011/12/29 09:48:48.024938
     12351 2011/12/29 09:48:49.025834
     12352 2011/12/29 09:48:50.026523
     12352 2011/12/29 09:48:51.027325
     12352 2011/12/29 09:48:52.028122
     12353 2011/12/29 09:48:54.027811
     12353 2011/12/29 09:48:55.027850
     12353 2011/12/29 09:48:56.028738
     12354 2011/12/29 09:48:58.029017
     12354 2011/12/29 09:48:59.030678
     12355 2011/12/29 09:49:00.031466
     12357 2011/12/29 09:49:02.033270
     12357 2011/12/29 09:49:03.033251
     12358 2011/12/29 09:49:04.034311

       TSN TIME
---------- --------------------------------------------------------------
     12358 2011/12/29 09:49:05.034333

24 rows selected.

SQL> select * from todo as of scn 12350;

ITEM                 DUEDATE
-------------------- --------------------------------
결혼식 참석          2012/04/01
장보기                2011/12/28

2 rows selected.
```

다음은 **PIVOT/UNPIVOT**을 사용하는 예이다.

```
SQL> SELECT *
     FROM   (SELECT deptno, job, sal
             FROM   emp
     )
     PIVOT (SUM(sal) salary_sum
            FOR deptno
            IN  (10 dept10 ,20 dept20, 30 dept30)
     );

JOB       DEPT10_SALARY_SUM DEPT20_SALARY_SUM DEPT30_SALARY_SUM
--------- ----------------- ----------------- -----------------
MANAGER                2450              2975              2850
PRESIDENT              5000
SALESMAN                                                   5600
ANALYST                                  6000
CLERK                  1300              1900               950

5 rows selected.

SQL> CREATE VIEW pivoted_emp
     as
     SELECT *
     FROM   (SELECT deptno, job, sal
             FROM   emp
     )
     PIVOT (SUM(sal) salary_sum
            FOR deptno
            IN  (10 dept10 ,20 dept20, 30 dept30)
     );

View 'PIVOTED_EMP' created.

SQL> SELECT *
     FROM   pivoted_emp
     UNPIVOT (
       salary_sum
       FOR department
       IN  (dept10_salary_sum as 10,
            dept20_salary_sum as 20,
            dept30_salary_sum as 30)
     );

JOB       DEPARTMENT SALARY_SUM
--------- ---------- ----------
```

```
MANAGER           10       2450
MANAGER           20       2975
MANAGER           30       2850
PRESIDENT         10       5000
SALESMAN          30       5600
ANALYST           20       6000
CLERK             10       1300
CLERK             20       1900
CLERK             30        950

9 rows selected.

SQL> SELECT *
     FROM   pivoted_emp
     UNPIVOT INCLUDE NULLS (
       salary_sum
       FOR department
       IN  (dept10_salary_sum as 10,
            dept20_salary_sum as 20,
            dept30_salary_sum as 30)
     );

JOB       DEPARTMENT SALARY_SUM
--------- ---------- ----------
MANAGER           10       2450
MANAGER           20       2975
MANAGER           30       2850
PRESIDENT         10       5000
PRESIDENT         20
PRESIDENT         30
SALESMAN          10
SALESMAN          20
SALESMAN          30       5600
ANALYST           10
ANALYST           20       6000
ANALYST           30
CLERK             10       1300
CLERK             20       1900
CLERK             30        950

15 rows selected.
```

5.2. 조인

조인(Join)은 두 개 또는 여러 개의 테이블이나 뷰로부터 로우를 결합하는 질의이다. Tibero에서는 FROM 절에 다수의 테이블이 있을 때 조인을 실행한다.

질의의 SELECT 절에서 조인 테이블에 속하는 컬럼을 선택할 수 있다. 만일 조인될 테이블 중에 같은 이름의 컬럼이 2개 이상이 있다면 테이블 이름을 함께 명시해 모호함을 없애야 한다.

5.2.1. 조인 조건

대부분의 조인 질의는 서로 다른 두 테이블의 컬럼을 비교하는 WHERE 절의 조건을 포함한다. 이런 조건을 조인 조건 (Join Condition)이라고 한다.

조인을 실행하기 위해서 각 테이블의 로우를 하나씩 가져와 조인 조건이 TRUE로 결정되는 경우에만 결합한다. 조인 조건에 포함되는 컬럼이 반드시 SELECT절에 포함될 필요는 없다.

세 개 이상의 테이블을 조인할 때는 우선 두 개의 테이블과 그 두 테이블의 컬럼에 대응 되는 조인 조건을 이용해서 조인을 한다. 그 후에, 그 두 테이블의 조인 결과의 컬럼과 세 번째 조인할 테이블의 컬럼에 해당하는 조인 조건으로 조인하여 새로운 결과를 만든다.

Tibero는 이러한 과정을 하나의 결과가 나올 때까지 반복한다. 최적화기(Optimizer)는 조인 조건과 테이블에 대한 인덱스와 통계 정보를 사용해 테이블 간의 조인 순서를 정한다. WHERE 절에 조인 조건 이외에 테이블에 하나에 대한 조건도 있을 수 있는데, 이러한 조건은 조인 질의로 반환되는 로우를 더욱 한정한다.

카티션 프로덕트

조인 질의에서 테이블에 대한 조인 조건이 없을 경우 카티션 프로덕트(Cartesian Products)를 반환한다. 카티션 프로덕트는 테이블의 한 로우가 다른 테이블의 모든 로우와 결합되는 것을 말한다.

예를 들어 100개의 로우를 가지는 두 개의 테이블의 카티션 프로덕트는 100 * 100 = 10,000 로우이다. 카티션 프로덕트는 이렇게 결과가 너무 많기 때문에 거의 사용되지 않는다. 특별히 카티션 프로덕트가 필요한 경우가 아니라면, 항상 조인 조건을 포함해야 한다. 만일 3개 이상의 테이블을 조인 할 때 그 중 2개의 테이블에 대한 조인 조건이 없었다면, 최적화기는 되도록 카티션 프로덕트가 생기지 않도록 조인순서를 정할 것이다.

5.2.2. 동등 조인

동등 조인(Equi Join)은 동등 연산자(=)로 구성된 조인 조건을 포함한 조인이다. 동등 조인은 정해진 컬럼에 대해 같은 값을 가지는 로우를 결합하여 결과로 반환한다.

5.2.3. 자체 조인

자체 조인(Self Join)은 하나의 테이블을 사용해서 자신에게 조인하는 것을 의미한다. 동일한 하나의 테이블이 FROM 절에 두 번 사용되기 때문에 별칭을 사용하여 컬럼을 구분한다.

5.2.4. 내부 조인

간단한 조인(Simple Join)이라고도 불리는 내부 조인(Inner Join)은 조인 조건을 만족하는 로우만 반환하는 2개 이상의 테이블에 대한 조인이다.

5.2.5. 외부 조인

외부 조인(Outer Join)은 일반 조인을 확장한 결과를 출력한다. 외부 조인은 조인 조건을 만족하는 로우뿐만 아니라, 한 테이블의 어떤 로우에 대해 반대편 테이블의 모든 로우가 조인 조건을 만족하지 못하는 경우에도 그 로우를 출력한다.

외부 조인은 왼쪽 외부 조인, 오른쪽 외부 조인, 완전 외부 조인이 있다.

- 왼쪽 외부 조인(left outer join)
  - 테이블 A와 B를 조인하는 경우 조인 조건에 맞는 로우를 출력하고, A의 로우 중에 조인 조건에 맞는 B의 로우가 없는 경우에도 그 로우를 모두 출력한다.
  - A의 로우 중 조인 조건을 만족하는 B의 로우가 없는 로우에 대해서는 조인의 출력에서 B 컬럼이 필요한 부분에 모두 NULL을 출력한다.
  - 왼쪽 외부 조인을 SQL 문장에 명시하려면 LEFT [OUTER] JOIN을 FROM 절에 명시하거나, WHERE 절의 조인 조건에 있는 B의 모든 컬럼에 외부 조인 연산자 (+)를 명시한다.
- 오른쪽 외부 조인(right outer join)
  - 테이블 A와 B를 조인하는 경우 조인 조건에 맞는 로우를 출력하고, B의 로우 중에 조인 조건에 맞는 A의 로우가 없는 경우에도 그 로우를 모두 출력한다.
  - B의 로우 중 조인 조건을 만족하는 A의 로우가 없는 로우에 대해서는 조인의 출력에서 A 컬럼이 필요한 부분에 모두 NULL을 출력한다.
  - 오른쪽 외부 조인을 SQL 문장에 명시하려면 RIGHT [OUTER] JOIN을 FROM 절에 명시하거나, WHERE 절의 조인 조건에 있는 A의 모든 컬럼에 외부 조인 연산자 '(+)'를 명시한다.
- 완전 외부 조인(full outer join)
  - 테이블 A와 B를 조인하는 경우 조인 조건에 맞는 로우를 출력하고, A의 로우 중에 조인 조건에 맞는 B 의 로우가 없는 경우에도 그 로우를 모두 출력하고, B의 로우 중에 조인 조건에 맞는 A의 로우가 없는 경우에도 그 로우를 모두 출력한다.
  - A의 로우 중 조인 조건을 만족하는 B의 로우가 없는 로우에 대해서는 조인의 출력에서 B 컬럼이 필요한 부분에 모두 NULL을 출력한다. B의 로우 중 조인 조건을 만족하는 A의 로우가 없는 로우에 대해서는 조인의 출력에서 A 컬럼이 필요한 부분에 모두 NULL을 출력한다.
  - 완전 외부 조인을 SQL 문장에 명시하려면 FULL [OUTER] JOIN을 FROM 절에 명시한다.

여러 테이블 간의 외부 조인을 수행하는 질의에서 하나의 테이블은 오직 다른 하나의 테이블에 대해서만 NULL을 제공하는 테이블의 역할을 할 수 있다. 따라서, 테이블 A와 B에 대한 조건과 테이블 B와 C에 대한 조건에서 모두 B의 컬럼 쪽에 (+) 연산자를 적용할 수는 없다.

외부 조인 연산자 (+)에는 FROM 절에 외부 조인을 명시할 경우에 다음과 같은 규칙과 제약 조건이 있다.

- FROM 절에 조인이 있는 질의 블록에는 외부 조인을 사용할 수 없다.
- (+) 연산자는 WHERE 절에만 올 수 있고, 테이블이나 뷰의 컬럼에만 적용할 수 있다.
- 테이블 A와 B의 조인 조건이 여러 개 있을 경우에는 '(+)' 연산자를 모든 조건에 사용해야 한다. 그렇지 않은 경우에는 아무런 경고나 에러 메시지 없이 일반 조인과 같이 취급한다.
- (+) 연산자를 외부 테이블과 내부 테이블에 모두 사용할 경우에는 일반 조인과 같이 취급한다.
- (+) 연산자는 컬럼에만 적용될 수 있고, 일반 연산식에는 적용될 수 없다. 단, 연산식 내의 컬럼에 '(+)' 연산자를 적용할 수는 있다.
- (+) 연산자를 포함하는 조건은 WHERE 절의 다른 조건과 OR 연산자를 통해 묶일 수 없다.
- (+) 연산자가 적용된 컬럼을 IN 연산자를 이용해 비교하는 조건을 사용할 수 없다.
- (+) 연산자가 적용된 컬럼을 부질의의 결과와 비교할 수 없다.
- 테이블 A와 B의 외부 조인의 조건 중에 B의 컬럼을 상수와 비교하는 조건이 있다면, '(+)' 연산자를 B의 컬럼에 적용해야 한다. 그렇지 않으면, 일반 조인과 같이 취급한다.

5.2.6. 안티 조인

안티 조인(Anti Join)은 프리디키트의 오른쪽 부분에 해당하는 로우가 없는 왼쪽 부분의 프리디키트에 해당하는 로우를 반환한다. 즉 프리디키트의 오른쪽 부분을 NOT IN의 부질의로 실행했을 때 일치하지 않는 로우를 반환한다.

5.2.7. 세미 조인

세미 조인(Semi Join)은 프리디키트의 오른쪽의 다수의 로우에 해당하는 왼쪽 부분의 로우를 중복 없이 처리하는 EXIST 부질의와 같은 로우를 반환한다.

부질의가 WHERE 절의 OR로 연결되어 있으면 세미 조인과 안티 조인으로 변환되지 않는다.

5.3. 부질의

질의를 사용해서 어떤 문제를 해결하고자 할 때 단계를 나누어서 수행하면 좀 더 쉽게 문제를 풀 수 있는 경우가 있다.

예를 들어 Peter라는 사람이 속해있는 부서에서 일하는 사람 모두를 구하고자 할 때 먼저 Peter의 부서를 구하는 질의를 작성한 다음, 그 질의의 결과를 이용해 최종 답을 얻는 형태로 단계를 나눈 질의를 생각해 볼 수 있다. 이처럼 하나의 질의가 내부에 또 다른 질의를 포함하고 있을 경우 이 내부에 포함된 질의를 부질의(Subquery)라고 한다.

부질의는 질의가 사용된 위치에 따라 다음과 같이 두 가지 형태로 나눌 수 있다.

부질의 종류	설명
인라인 뷰	부질의가 부모 질의의 FROM 절에서 사용되었을 경우 이런 부질의를 보통 인라인(Inline) 뷰라고 부른다.
중첩된 부질의	부 질의가 부모 질의의 SELECT 리스트 또는 WHERE 절 등에서 사용되었을 경우 이런 부질의를 중첩된(Nested) 부질의 또는 부질의라고 부른다.

부질의는 다음과 같은 경우에 종종 사용된다.

- INSERT 문을 통해 삽입할 로우의 값을 결정할 때
- CREATE TABLE을 통해 테이블을 생성함과 동시에 테이블의 내용을 채울 때
- CREATE VIEW 문을 통해 뷰가 질의하는 로우의 집합을 정의할 때
- UPDATE 문에서 UPDATE할 값을 결정할 때
- SELECT, UPDATE, DELETE 문에서 WHERE 절, HAVING 절, START WITH 절과 같은 조건을 명시할 때
- 테이블처럼 사용하고 싶은 로우의 집합을 정의할 때(SELECT의 FROM 절 또는 INSERT, UPDATE, DELETE에서 테이블을 명시할 수 있는 자리에 사용할 수 있다.)

참고

부질의는 내부에 다른 부질의를 포함할 수 있으며, Tibero에서는 부질의가 다른 부질의를 포함할 수 있는 단계에 대해 제한을 두지 않는다.

다음은 부질의와 컬럼 참조에 대한 설명이다.

- 인라인 뷰

  인라인 뷰 내부에서는 부모 질의의 FROM 절에 명시된 다른 테이블 (또는 뷰)의 컬럼을 볼 수 없다.

- 중첩된 부질의

  인라인 뷰와 달리 중첩된 부질의는 부모 질의의 FROM 절에 명시된 테이블의 컬럼을 볼 수 있으며, 부모 질의가 또 다른 질의의 중첩된 부질의일 경우 이 부모 질의의 FROM 절에 명시된 테이블의 컬럼 역시 이 자식 부질의에서 볼 수 있다.

- 서로 관련된(Correlated) 부질의

중첩된 부질의가 부모 질의의 테이블의 컬럼을 참조할 경우 이러한 부질의를 서로 관련된 부질의라고 한다.

부질의의 컬럼을 참조할 때의 규칙은 다음과 같다.

- 서로 관련된 부질의 내에서 부모 질의의 FROM 절의 테이블의 컬럼을 참조할 때 자신의 FROM 절의 테이블의 컬럼과 동일한 이름을 가지고 있는 컬럼을 참조하고자 하는 경우에는 컬럼 이름 앞에 부모 질의의 테이블 이름을 붙여 주어야 한다.
- 부모 질의의 컬럼을 참조할 때 위로 올라가는 단계에 대한 제한은 없다. 서로 관련된 부질의는 부모 질의의 각각의 로우를 처리할 때 매번 일일이 별도로 수행된다.
- 컬럼 이름에 대한 모호함이 발생하지 않는 한, 부질의에서 명시하는 (테이블 이름을 앞에 붙이지 않은) 컬럼 이름은 자신의 FROM 절의 테이블에서부터 시작해서 자신의 부모, 부모의 부모 순으로 컬럼 이름을 찾아본다.

서로 관련된 부질의는 하위 단계에서 구하고자 하는 값이 상위 단계의 각각의 로우에 따라 별도로 결정되어야 하는 경우에 사용된다.

다음과 같은 경우를 가정해 보자.

- 자신의 부서의 평균 연봉보다 많은 연봉을 받는 사원의 리스트를 구하는 질의가 있다.
- 이때, 하위 단계에서 계산하고자 하는 값은 각 사원에 대해서 그 사원이 근무하고 있는 부서의 평균 연봉이 된다.
- 또한, 상위 단계에서는 하위 단계에서 구한 평균 연봉을 현재 사원의 연봉과 비교하게 된다.

이러한 경우 평균 연봉을 각 사원에 대해 매번 구해야 하므로 이럴 경우 서로 관련된 부질의를 구사해서 문제를 해결할 수 있다.

다음은 스칼라(scalar) 부질의에 대한 설명이다. 자세한 내용은 “3.3.6. 부질의 연산식”을 참고한다.

- 어떤 부질의가 0개 또는 1개의 로우에서 1개의 컬럼만을 반환할 경우 이러한 부질의를 특별히 스칼라 부질의라고 부른다.
- 스칼라 부질의는 연산식의 하나로 연산식 expr이 올 수 있는 자리에 마치 하나의 값처럼 간주되어 자유로이 사용될 수 있다.

5.4. 집합 연산자

집합 연산자는 두 개의 질의를 하나로 결합하는 데 사용된다. 집합 연산자에 의해 결합된 질의의 select_list에 명시된 연산식의 개수는 동일해야 하고 대응되는 연산식은 같은 데이터 타입 그룹에 속해야 한다.

다음은 집합 연산자의 우선순위에 대한 설명이다.

우선순위	집합 연산자	설명
1	INTERSECT	두 개의 질의 결과 양쪽 모두에 존재하는 로우를 결과로 반환한다. (A ∩ B)
2	UNION	두 개의 질의의 결과에서 중복된 로우를 제거한 후 결과를 반환한다. (A ∪ B)
	UNION ALL	두 개의 질의의 결과에서 중복된 로우를 제거시키지 않고 모든 결과를 반환한다. (A + B)
	MINUS	앞의 질의 결과에서 뒤의 질의 결과를 뺀 결과를 반환한다. (A - B)
	EXCEPT	MINUS 집합 연산자와 동일하게 동작한다. (A - B)

두 개 이상의 질의가 집합 연산자에 의해 결합되면, 다음과 같은 규칙이 적용된다.

- 왼쪽에서 오른쪽 순서로 질의를 수행한다.
- 괄호를 사용하여 쿼리의 수행 순서를 조정할 수 있다.

집합 연산자는 다음과 같은 제약조건이 있다.

- select_list가 BLOB, CLOB 타입의 연산식을 가질 경우 집합 연산자를 사용할 수 없다.
- UNION, INTERSECT, MINUS, EXCEPT 연산자의 경우 LONG 타입의 컬럼이 허용되지 않는다.
- 먼저 명시된 질의의 select_list에 나오는 연산식에 별칭이 명시되어야만 order_by_clause에서 참조할 수 있다.
- for_update_clause를 명시할 수 없다.

대응되는 연산식이 동일한 데이터 타입의 그룹이 아니면 암시적인 형 변환은 허용되지 않으며, 동일한 데이터 타입의 그룹이면 다음과 같은 형 변환이 일어난다.

- 데이터 타입의 그룹이 숫자이면, 결과는 NUMBER 타입이다.
- 데이터 타입의 그룹이 문자이면, 다음의 규칙에 의해 결과의 데이터 타입이 결정된다.
  - 양쪽이 CHAR 타입이면, 결과는 CHAR 타입이다.
  - 하나 또는 양쪽 모두 VARCHAR2 타입이면, 결과는 VARCHAR2 타입이다.

5.5. 계층 질의

계층 질의(Hierarchical Query)란 테이블에 포함된 로우 사이에 상하 계층 관계가 성립된 경우 그 상관 관계에 따라 로우를 출력하는 질의이다.

하나의 대상 테이블에 계층 관계는 하나 이상 정의할 수 있으며, 계층 질의는 하나의 테이블 또는 조인된 둘 이상의 테이블에 대해서도 가능하다.

계층 질의를 위하여 SELECT 문장 내에 START WITH … CONNECT BY 절을 이용한다.

- START WITH 절은 계층 내의 루트 로우(root row)를 지정하기 위한 것이다.
- CONNECT BY 절은 로우 간의 상하 관계를 정의하기 위한 것이다.
- START WITH 절과 CONNECT BY 절에는 하나의 조건식이 포함되며, 단순 조건식 또는 복합 조건식일 수 있다. 자세한 내용은 "3.4. 조건식"을 참고한다.

5.5.1. 계층 질의 연산자

PRIOR

CONNECT BY 절에는 다른 조건식에서는 포함되지 않는 특별한 연산자가 사용되는데, 로우 간의 상하 관계를 나타내기 위한 PRIOR 연산자이다.

PRIOR 연산자가 포함된 조건식은 다음과 같은 형식을 갖는다.

```
PRIOR expr = expr
expr = PRIOR expr
```

PRIOR 쪽의 연산식의 결과 값을 갖는 로우가 반대 쪽의 연산식의 결과 값을 갖는 로우의 부모가 된다.

예를 들어 두 개의 컬럼 EMPNO와 MGRNO를 포함하는 테이블 EMP에 대하여 다음의 조건식을 이용하여 계층 질의를 수행한다면, 특정 (부모) 로우의 EMPNO 컬럼 값과 같은 MGRNO 컬럼 값을 갖는 모든 로우는, EMPNO 컬럼 값을 갖는 로우의 자식 로우가 된다.

```
PRIOR EMPNO = MGRNO
```

다음의 테이블 EMP2 내의 로우 중에서 EMPNO 컬럼 값이 27인 (부모) 로우에 대하여 EMPNO 컬럼 값이 35, 42인 로우가 자식 로우가 된다. 두 컬럼 모두 MGRNO 컬럼 값이 27이기 때문이다.

```
     EMPNO ENAME        ADDR                SALARY      MGRNO
---------- ------------ ---------------- ---------- ----------
        35 John         Houston              30000         27
        54 Alicia       Castle               25000         24
        27 Ramesh       Humble               38000         12
        69 James        Houston              35000         24
        42 Allen        Brooklyn             29000         27
        87 Ward         Humble               28500         35
        24 Martin       Spring               30000         12
        12 Clark        Palo Alto            45000          5
```

CONNECT BY 절의 조건식은 여러 단순 조건식이 연결된 복합 조건식이 될 수 있다. 하지만, PRIOR 연산자를 포함한 단순 조건식은 반드시 하나만 포함되어야 하며, 0개 또는 2개 이상이 포함된 경우에는 에러를 반환한다. CONNECT BY 절의 조건식은 부질의를 포함할 수 없다.

CONNECT_BY_ROOT

계층 질의에서만 사용되는 또 다른 특수 연산자로 CONNECT_BY_ROOT 연산자가 있다.

CONNECT_BY_ROOT 연산자를 사용하면 데이터베이스는 루트 로우의 데이터를 이용하여 컬럼 값을 반환한다.

다음은 CONNECT_BY_ROOT 연산자를 사용한 예이다.

```
SQL> SELECT ENAME, CONNECT_BY_ROOT ENAME MANAGER,
            SYS_CONNECT_BY_PATH(ENAME, '-') PATH
     FROM EMP2
     WHERE LEVEL > 1
       CONNECT BY PRIOR EMPNO = MGRNO
       START WITH ENAME = 'Clark';

ENAME           MANAGER         PATH
--------------- --------------- ----------------------
Martin          Clark           -Clark-Martin
James           Clark           -Clark-Martin-James
Alicia          Clark           -Clark-Martin-Alicia
Ramesh          Clark           -Clark-Ramesh
Allen           Clark           -Clark-Ramesh-Allen
JohnClark       Clark           -Ramesh-John
Ward            Clark           -Clark-Ramesh-John-Ward

7 rows selected.
```

5.5.2. 계층 질의의 조건식

- CONNECT BY 절

  CONNECT BY 절의 조건식에는 등호 이외에 다른 비교 연산자를 사용할 수 있다. 하지만, 이때 상하 관계가 순환적으로 정의될 수 있으며 무한 루프에 빠질 수 있다. 이러한 경우 Tibero에서는 에러를 반환하고 실행을 중지한다.

  상하관계의 순환여부는 의사 컬럼인 CONNECT_BY_ISCYCLE 값으로 출력할 수 있다.

- START WITH 절

START WITH 절은 계층 관계를 검색하기 위한 루트 로우에 대한 조건식을 포함한다. 조건식에 따라 0개 이상의 루트 로우로부터 시작될 수 있다. 만약 START WITH 절이 생략되면 대상 테이블 내의 모든 로우를 루트 로우로 하여 계층 관계를 검색한다.

- CONNECT BY 절과 WHERE 절의 혼합

  하나의 SELECT 문 내에 CONNECT BY 절과 WHERE 절이 함께 사용된 경우 CONNECT BY 절의 조건식을 먼저 적용한다. 만약 SELECT 문이 조인 연산을 수행하며 WHERE 절 내에 조인 조건이 포함되어 있다면, 조인 조건만 먼저 적용하여 조인을 수행한 후에 CONNECT BY 절의 조건식과 WHERE 절 내의 나머지 조건식을 차례로 적용한다.

  WHERE 절 내의 조건식은 CONNECT BY 절의 조건식에 의하여 로우 간에 상하 관계가 정해진 후에 적용되므로, 특정 로우가 WHERE 절에 의하여 제거되더라도 그 로우의 하부 로우도 최종 결과에 포함될 수 있다.

- ORDER SIBLINGS BY 절

  계층 질의에 대해 일반적인 ORDER BY를 사용할 경우 상하 관계를 무시하고 정렬이 된다. 그러나 ORDER SIBLINGS BY 절을 사용할 경우엔 계층간의 상하 관계를 유지한 상태에서 같은 레벨에 있는 로우 들에 대해서만 정렬을 하므로 원하는 결과를 얻을 수 있다.

5.5.3. 계층 질의의 실행 방식

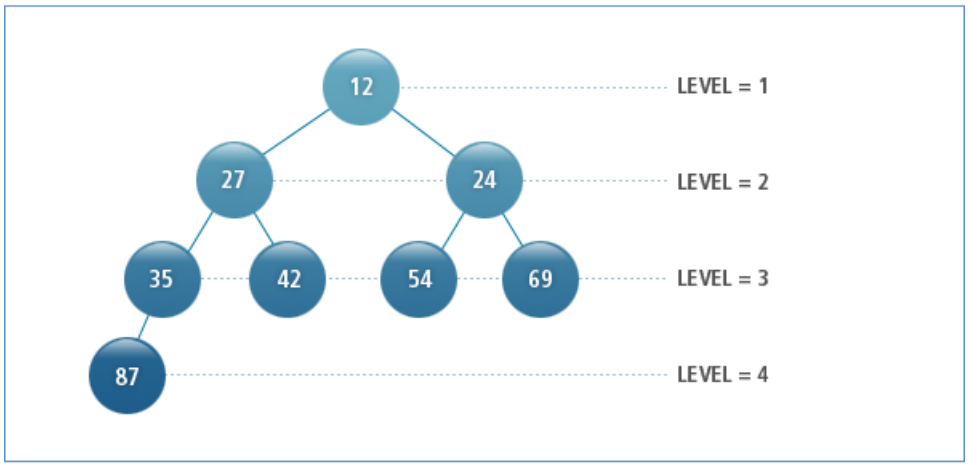
계층 질의는 재귀적(Recursive)으로 실행된다.

먼저 하나의 루트 로우에 대해 모든 자식 로우를 검색한다. 그 다음 각 자식 로우의 자식 로우를 다시 검색한다. 이러한 방법으로 다시는 자식 로우가 발견되지 않을 때까지 계속 진행한다. 만약 무한 루프가 되면 에러를 반환한다.

계층 질의의 결과를 출력하는 순서는 깊이 우선(Depth-First) 순서를 따른다.

다음의 그림은 테이블 EMP2의 계층 질의 결과로 구성된 계층 관계의 한 예이며, 원 안의 값은 EMPNO 컬럼의 값이다. 이때, 로우의 출력 순서는 12, 27, 35, 87, 42, 24, 54, 69이다.

[그림 5.1] EMP2 테이블의 계층 관계



계층 트리의 각 로우는 레벨 값을 갖는다. 루트 로우의 레벨 값은 1이며 자식 로우로 내려가면서 1씩 증가한다. 따라서, 위의 [그림 5.1]에서 EMPNO = 12인 로우는 레벨이 1이며, EMPNO = 27, 24인 로우는 레벨이 2, EMPNO = 35, 42, 54, 69인 로우는 레벨이 3, EMPNO = 87인 로우는 레벨이 4다.

레벨 값은 의사 컬럼인 LEVEL 컬럼의 값으로 출력할 수 있다.

다음은 START WITH 절을 이용하지 않고 계층 질의를 실행하는 SELECT 문의 예이다. 본 예제에서는 모든 로우를 루트 로우로 하여 하부 로우를 출력한다.

```
SQL> SELECT EMPNO, ENAME, ADDR, MGRNO
     FROM EMP2
       CONNECT BY PRIOR EMPNO = MGRNO;

     EMPNO ENAME        ADDR                MGRNO
---------- ------------ ---------------- ----------
        12 Clark        Palo Alto                5
        27 Ramesh       Humble                  12
        35 John         Houston                 27
        87 Ward         Humble                  35
        42 Allen        Brooklyn                27
        24 Martin       Spring                  12
        54 Alicia       Castle                  24
        69 James        Houston                 24
        27 Ramesh       Humble                  12
        35 John         Houston                 27
        87 Ward         Humble                  35
        42 Allen        Brooklyn                27
        24 Martin       Spring                  12
        54 Alicia       Castle                  24
        69 James        Houston                 24
        54 Alicia       Castle                  24
        69 James        Houston                 24
```

```
        35 John         Houston                  27
        87 Ward         Humble                   35
        42 Allen        Brooklyn                 27
        87 Ward         Humble                   35

21 rows selected.
```

다음은 START WITH 절을 이용하여 계층 질의를 실행하는 SELECT 문의 예이다. 본 예제에서는 EMPNO = 12인 조건식을 만족하는 하나의 루트 로우에 대해서만 하부 로우를 검색한다.

```
SQL> SELECT EMPNO, ENAME, ADDR, MGRNO
     FROM EMP2
       START WITH EMPNO = 12
       CONNECT BY PRIOR EMPNO = MGRNO;

     EMPNO ENAME        ADDR                MGRNO
---------- ------------ ---------------- ----------
        12 Clark        Palo Alto                5
        27 Ramesh       Humble                  12
        35 John         Houston                 27
        87 Ward         Humble                  35
        42 Allen        Brooklyn                27
        24 Martin       Spring                  12
        54 Alicia       Castle                  24
        69 James        Houston                 24

8 rows selected.
```

다음은 WHERE 절을 포함하여 계층 질의를 실행하는 SELECT 문의 예이다. 본 예제에서는 EMPNO = 27인 로우가 WHERE 절 때문에 제거되었으나, 그 하부 로우는 모두 그대로 출력된다.

```
SQL> SELECT EMPNO, ENAME, ADDR, MGRNO
     FROM EMP2
       WHERE ENAME != 'Ramesh'
       START WITH EMPNO = 12
       CONNECT BY PRIOR EMPNO = MGRNO;

     EMPNO ENAME        ADDR                MGRNO
---------- ------------ ---------------- ----------
        12 Clark        Palo Alto                5
        35 John         Houston                 27
        87 Ward         Humble                  35
        42 Allen        Brooklyn                27
        24 Martin       Spring                  12
        54 Alicia       Castle                  24
        69 James        Houston                 24
```

```
7 rows selected.
```

다음은 LEVEL 컬럼을 포함하여 계층 질의를 실행하는 SELECT 문의 예이다.

```
SQL> SELECT EMPNO, ENAME, ADDR, MGRNO, LEVEL
     FROM EMP2
       START WITH EMPNO = 12
       CONNECT BY PRIOR EMPNO = MGRNO;

     EMPNO ENAME        ADDR                 MGRNO      LEVEL
---------- ------------ ---------------- ---------- ----------
        12 Clark        Palo Alto                5          1
        27 Ramesh       Humble                  12          2
        35 John         Houston                 27          3
        87 Ward         Humble                  35          4
        42 Allen        Brooklyn                27          3
        24 Martin       Spring                  12          2
        54 Alicia       Castle                  24          3
        69 James        Houston                 24          3

8 rows selected.
```

5.6. 병렬 질의

병렬 질의란 하나의 SQL 문장을 여러 워킹 스레드를 사용하여 처리하는 것을 말한다. 대용량 테이블을 스캔할 때 여러 워킹 스레드가 테이블의 영역을 나눠서 처리하면 워킹 스레드 하나를 사용했을 때보다 빠르게 작업을 실행할 수 있다.

병렬 질의는 OLTP(Online transaction processing) 환경 보다는 주로 데이터 웨어하우스(Data warehouse)와 같은 대용량 데이터를 다루는 환경에서 주로 사용된다.

다음은 병렬 질의를 사용하는 예이다.

```
SELECT /*+ PARALLEL (4) */ DEPTNO, AVG(SAL) FROM EMP GROUP BY DEPTNO;
```

위의 예에서처럼 병렬 질의를 사용하기 위해서는 **PARALLEL** 힌트를 사용하면 된다. 'PARALLEL (4)'라고 명시한 부분의 숫자 4는 병렬 질의 처리에서 사용할 워킹 스레드의 개수를 나타내며 보통 **DOP**(Degree of parallelism)라고 한다. 즉 4개의 워킹 스레드를 사용해 해당 질의를 처리하도록 지시한다.

이와 같이 병렬 질의 실행이 지시되면 Tibero 서버는 유휴 상태의 워킹 스레드를 사용자가 지정한 DOP 개수만큼 확보하려 할 것이다.

현재 사용할 수 있는 워킹 스레드가 힌트에 지정된 개수보다 부족하다면 확보할 수 있는 워킹 스레드만을 사용해 병렬 질의를 실행한다. 만약 유휴 상태의 워킹 스레드가 없어서, DOP가 1이 되면 병렬 질의는 진행되지 않으며, 일반 질의처럼 처리된다. 이러한 상황을 별도의 에러 메시지를 통해 알려주지는 않는다.

Tibero가 병렬 질의로 실행할 수 있는 연산은 다음과 같다.

- TABLE FULL SCAN
- INDEX FAST FULL SCAN
- HASH JOIN
- NESTED LOOP JOIN
- MERGE JOIN
- SET 연산
- GROUP BY
- ORDER BY
- 집계 함수

Tibero는 병렬 질의를 실행하기 위해 실행 계획을 생성한다.

다음은 병렬 질의를 실행하기 위해 실행 계획을 생성한 예이다.

```
SQL> SET AUTOT TRACE EXPLAIN
SQL> SELECT EMPNO, ENAME, SAL FROM EMP ORDER BY SAL;
SQL ID: 451
Plan Hash Value: 2848347407

Explain Plan
-------------------------------------------------------------------------------
1  ORDER BY (SORT)
2    TABLE ACCESS (FULL): EMP


SQL> SELECT /*+ PARALLEL (4) */ EMPNO, ENAME, SAL FROM EMP ORDER BY SAL;
SQL ID: 449
Plan Hash Value: 979088785

Explain Plan
-------------------------------------------------------------------------------
1  PE MANAGER
2    PE SEND QC (ORDER)
3      ORDER BY (SORT)
4        PE RECV
5          PE SEND (RANGE)
6            PE BLOCK ITERATOR
7              TABLE ACCESS (FULL): EMP
```

위의 예를 보면, PARALLEL 힌트를 사용한 병렬 질의의 실행 계획에 기존에는 없던 새로운 연산 노드가 추가가 된 것을 확인 할 수 있다. 이때 새로 추가된 노드는 병렬 질의 실행에 필요한 작업을 하게 된다.

PE RECV, PE SEND 노드가 추가된 경우에는 Tibero는 병렬 질의를 실행하기 위해 2-set 모델을 사용한다. 이 경우 실제 사용하는 워킹 스레드의 개수는 DOP의 2배가 되며 2개의 set 중에서 한 곳의 워킹 스레드는 consumer의 역할을, 다른 set의 워킹 스레드는 producer의 역할을 하게 된다. 이렇게 2-set 모델을 사용하는 이유는 두 연산 노드를 동시에 병렬로 실행하여 파이프라이닝(Pipelining) 효과를 보기 위해서이다.

5.7. 듀얼 테이블

듀얼 테이블(DUAL Table)은 VARCHAR(1) 타입의 하나의 컬럼 더미(Dummy)와 문자열 'X'를 포함하는 하나의 로우를 포함하는 테이블이다.

듀얼 테이블의 특성은 다음과 같다.

- SYS 사용자뿐만 아니라 모든 사용자가 듀얼 테이블을 사용할 수 있다.
- 삽입, 삭제, 갱신 연산이 허용되지 않으며, 테이블 자체를 변경하거나 제거할 수 없다.
- 테이블의 내용과 상관없는 연산식의 계산에 사용할 수 있다.
- 하나의 로우만을 포함하기 때문에 항상 연산식의 결과는 하나다.

다음은 듀얼 테이블을 사용하는 예이다.

```
SQL> SELECT SIN(3.141592654 / 2.0) FROM DUAL;

SIN(3.141592654/2.0)
--------------------
                   1

1 row selected.
```

제6장 실체화 뷰

본 장에서는 Tibero에서 제공하는 실체화 뷰를 사용하는 방법에 대해 기술한다.

6.1. 리프레시

마스터 테이블에 변경이 일어났을 때 이를 반영하도록 실체화 뷰를 갱신하는 동작을 **리프레시**라고 한다. 리프레시를 하면 실체화 뷰의 데이터는 질의의 결과와 일치된다. 리프레시는 실체화 뷰를 생성하는 조건에 따라 자동으로 수행될 수 있으며, DBMS_MVIEW 패키지의 REFRESH 함수를 이용하여 수동으로 수행할 수도 있다.

리프레시는 완전 리프레시와 빠른 리프레시가 있다.

6.1.1. 완전 리프레시

완전 리프레시는 실체화 뷰에 있는 기존의 데이터를 모두 삭제한 후 실체화 뷰를 정의한 질의를 다시 수행하여 그 결과를 실체화 뷰에 저장하는 방식이다. 완전 리프레시를 사용하기 위한 특별한 제약조건은 없다.

6.1.2. 빠른 리프레시

빠른 리프레시는 마스터 테이블의 변경된 부분만을 실체화 뷰에 반영하는 방식이다. 완전 리프레시보다 빠르다.

일반적인 제약조건

빠른 리프레시를 사용하기 위한 제약조건은 다음과 같다. 실체화 뷰를 정의하는 질의는 다음의 제약조건을 만족해야 한다.

- 실체화 뷰는 SYSDATE와 ROWNUM과 같이 반복할 수 없는 표현식을 포함하면 안 된다.
- 실체화 뷰는 LONG 또는 LONG RAW 데이터 타입을 포함하면 안 된다.
- SELECT 리스트에 부질의를 포함하면 안 된다.
- SELECT 리스트에 분석 함수를 포함하면 안 된다.
- HAVING 절에 부질의를 포함하면 안 된다.
- ANY, ALL, NOT EXISTS를 포함하면 안 된다.
- [START WITH] CONNECT BY 절을 포함하면 안 된다.

- 서로 다른 원격에 있는 테이블을 포함하면 안 된다. 즉, 쿼리에 참가하는 모든 테이블은 같은 서버에 있어야 한다.
- 원격 테이블이 포함된 경우 해당 서버가 Tibero일 경우만 된다.
- UNION 등의 SET 연산자를 포함하면 안 된다.
- REFRESH ON COMMIT 실체화 뷰를 정의한 질의는 원격 테이블을 포함하면 안 된다.

집합 함수를 포함한 제약조건

다음은 집합 함수를 포함한 제약조건에 대한 설명이다.

- 빠른 리프레시의 일반적인 제약조건을 모두 포함한다.
- 실체화 뷰의 모든 참조 테이블에는 실체화 뷰의 로그가 있어야 하며, 실체화 뷰의 로그는 다음의 제약조건을 만족해야 한다.
  - 실체화 뷰에서 참조하는 모든 컬럼을 포함해야 한다.
  - SEQUENCE, ROWID와 INCLUDING NEW VALUES 조건이 있어야 한다.
- 함수는 SUM, COUNT, AVG, STDDEV, VARIANCE. MIN, MAX 함수만 지원된다.
- COUNT(*)는 항상 포함해야 한다.
- 집합 함수는 항상 표현식의 최상위에 있어야 한다. 단, AVG(COUNT(X)), COUNT(X) * COUNT(X)는 허용하지 않는다.
- AVG(expr)에 대응되는 COUNT(expr)가 있어야 한다.

집단 함수	필요 집단 함수
COUNT(expr)	-
MIN(expr)	-
MAX(expr)	-
SUM(expr)	COUNT(expr)
SUM(col), col에 NOT NULL 제약	-
AVG(expr)	COUNT(expr)
STDDEV(expr)	SUM(expr), COUNT(expr)
VARIANCE(expr)	SUM(expr), COUNT(expr)

- SELECT 리스트에는 모든 GROUP BY 컬럼이 있어야 한다.
- CUBE, ROLLUP은 허용하지 않는다.

6.2. 질의 다시 쓰기

실체화 뷰의 가장 큰 장점은 **질의 다시 쓰기(Query Rewrite)** 기능을 사용할 수 있다는 것이다.

질의 다시 쓰기는 주어진 질의를 분석한 후 실체화 뷰를 사용하여 동일한 결과를 내는 새로운 질의를 생성하는 기능이다. 실체화 뷰가 복잡한 조인이나 집단 함수의 결과로 정의되어 있다면 질의 다시 쓰기를 통해 실행 시간을 크게 단축할 수 있다.

질의 다시 쓰기는 사용자가 명시한 명령 없이도 질의 최적화기에 의해 수행되므로 실체화 뷰를 인덱스처럼 사용할 수 있다. INSERT, DELETE, UPDATE, MERGE의 대상이 되는 부질의를 제외한 모든 SELECT 문에 동작한다.

참고

EXPLAIN PLAN 문을 통해서 실행 계획을 보면 질의 다시 쓰기에 의해 참조된 실체화 뷰는 **MV_REWRITE**라고 표시된다.

6.2.1. 동작 조건

질의 다시 쓰기 기능을 동작하기 위한 조건은 다음과 같다.

- 실체화 뷰를 생성할 때 ENABLE QUERY REWRITE 옵션을 추가해야 한다.
- QUERY_REWRITE_ENABLED 파라미터의 값이 TRUE나 FORCE로 설정되어 있어야 한다.

```
QUERY_REWRITE_ENABLED = {TRUE | FORCE}
```

- 질의 다시 쓰기가 QUERY_REWRITE_INTEGRITY 파라미터의 설정을 만족시켜야 한다.

```
QUERY_REWRITE_INTEGRITY = {ENFORCED | STALE_TOLERATED}
```

QUERY_REWRITE_INTEGRITY 파라미터는 다음의 설정 값을 통해 질의 다시 쓰기의 정확도를 조절할 수 있다.

설정 값	설명
ENFORCED	최신 데이터가 있는 실체화 뷰만 사용하여 원본 질의와 동일한 결과를 보장한다.
STALE_TOLERATED	최신 데이터의 변경 내용이 반영되지 않은 실체화 뷰를 사용하기 때문에 원본 질의와 동일한 결과를 보장하지 않는다. 단, 질의 다시 쓰기가 될 가능성은 커진다.

6.2.2. 동작 방식

질의 다시 쓰기를 하려면 원본 질의와 사용할 실체화 뷰의 질의를 비교해서 원본 질의가 실체화 뷰를 사용해서 얻어질 수 있는지를 확인해야 한다. 현재는 완전 문자열 비교와 부분 문자열 비교 방식을 지원한다.

완전 문자열 비교

원본 질의와 사용할 실체화 뷰의 질의에 대해 전체 문자열을 비교한다. 이때 공백이나 대소문자의 차이는 무시된다.

예를 들어 다음과 같은 실체화 뷰가 있다고 가정해 보자.

```
CREATE MATERIALIZED VIEW MV_SUM_SALARY ENABLE QUERY REWRITE AS
    SELECT DNAME, SUM(SALARY) FROM DEPT, EMP WHERE DEPT.DEPTNO = EMP.DEPTNO
    GROUP BY DNAME;
```

1. 다음과 같은 질의를 실행한다면,

```
SELECT dname, SUM(salary) FROM dept,emp WHERE DEPT.DEPTNO = EMP.DEPTNO
    GROUP BY dname;
```

2. 다음과 같이 질의 다시 쓰기가 동작하게 된다.

```
SELECT * FROM MV_SUM_SALARY;
```

부분 문자열 비교

원본 질의와 사용할 실체화 뷰의 질의에 대해 FROM 절부터 ORDER BY 절 앞까지(존재할 경우)의 문자열을 비교하고, SELECT 리스트와 ORDER BY 절의 연산식에서 사용되는 컬럼의 적합성을 조사한다.

컬럼의 적합성을 조사하는 방법은 다음과 같다.

원본 질의의 SELECT 리스트 컬럼이 실체화 뷰의 컬럼을 조합해서 얻어질 수 있는지를 검사한다. 이때 조인에 의해 같아지는 컬럼을 처리한다.

예를 들어 다음과 같은 실체화 뷰가 있다고 가정해 보자.

```
CREATE MATERIALIZED VIEW MV_JOIN_DEPT_EMP ENABLE QUERY REWRITE AS
    SELECT ENAME, DNAME, DEPT.DEPTNO FROM DEPT, EMP WHERE DEPT.DEPTNO = EMP.DEPTNO;
```

1. 다음과 같은 질의를 실행한다면,

```
SELECT ENAME, DNAME, EMP.DEPTNO FROM DEPT, EMP WHERE DEPT.DEPTNO = EMP.DEPTNO;
```

2. 다음과 같이 질의 다시 쓰기가 동작하게 된다.

```
SELECT ENAME, DNAME, DEPTNO FROM MV_JOIN_DEPT_EMP;
```

또한, 연산의 교환, 결합, 분배 법칙을 이용하여 동등한 연산식의 컬럼을 처리한다.

예를 들어 다음과 같은 실체화 뷰가 있다고 가정해 보자.

```
CREATE MATERIALIZED VIEW mymv ENABLE QUERY REWRITE AS
    SELECT a*(c+4)+b c1, sum(c+a)+1 c2 FROM base GROUP BY a*(c+4)+b, b+4*a+a*c;
```

1. 다음과 같은 질의를 실행한다면,

```
SELECT b+4*a+a*c+(1+sum(a+c))*7 FROM base GROUP BY a*(c+4)+b, b+4*a+a*c;
```

2. 다음과 같이 질의 다시 쓰기가 동작하게 된다.

```
SELECT c1+c2*7 FROM mymv;
```

집단 함수를 사용할 때 **집단 함수의 적합성**을 조사하는 방법은 다음과 같다.

해당되는 집단 함수가 없어도 다른 집단 함수를 조합하여 원하는 집단 함수를 얻을 수 있다. 예를 들어 AVG(SALARY)는 SUM(SALARY) / COUNT(SALARY)로 구할 수 있다.

다음은 집단 함수의 적합성을 조사한 표이다.

목표 집단 함수	**필요 집단 함수**
AVG(expr)	SUM(expr), COUNT(expr)
SUM(expr)	AVG(expr), COUNT(expr)
STDDEV(expr)	VARIANCE(expr)
	(STDDEV_SAMP(expr), STDDEV_POP(expr), VAR_SAMP(expr), VAR_POP(expr)) 중 하나, COUNT(expr)
	SUM(expr * expr), SUM(expr), COUNT(expr)
STDDEV_SAMP(expr)	VAR_SAMP(expr)
	(STDDEV(expr), STDDEV_POP(expr), VARIANCE(expr), VAR_POP(expr)) 중 하나, COUNT(expr)
	SUM(expr * expr), SUM(expr), COUNT(expr)
STDDEV_POP(expr)	VAR_POP(expr)
	(STDDEV(expr), STDDEV_SAMP(expr), VARIANCE(expr), VAR_SAMP(expr)) 중 하나, COUNT(expr)
	SUM(expr * expr), SUM(expr), COUNT(expr)
VARIANCE(expr)	STDDEV(expr)
	(STDDEV_SAMP(expr), STDDEV_POP(expr), VAR_SAMP(expr), VAR_POP(expr)) 중 하나, COUNT(expr)

목표 집단 함수	필요 집단 함수
	SUM(expr * expr), SUM(expr), COUNT(expr)
VAR_SAMP(expr)	STDDEV_SAMP(expr)
	(STDDEV(expr), STDDEV_POP(expr), VARIANCE(expr), VAR_POP(expr)) 중 하나, COUNT(expr)
	SUM(expr * expr), SUM(expr), COUNT(expr)
VAR_POP(expr)	STDDEV_POP(expr)
	(STDDEV(expr), STDDEV_SAMP(expr), VARIANCE(expr), VAR_SAMP(expr)) 중 하나, COUNT(expr)
	SUM(expr * expr), SUM(expr), COUNT(expr)

6.2.3. 비용 기반 최적화

원본 질의와 다시 쓰여진 질의로부터 비용 기반 최적화 기법에 의해 각각의 실행 계획이 생성되고, 최종으로 두 실행 계획 중에서 비용이 적은 쪽이 선택된다. 질의 다시 쓰기는 실행 비용에 따라 최종 선택이 달라지기 때문에 원본 질의가 참조하는 테이블과 실체화 뷰를 저장하고 있는 테이블의 통계 정보를 생성한다.

6.3. 원격 저장소를 가진 실체화 뷰

이기종 데이터베이스에서 실체화 뷰를 이용하여 Tibero에 있는 베이스 테이블의 데이터를 동기화하고 싶을 때 사용하는 기능이다.

이기종 데이터베이스에서 Tibero에 있는 베이스 테이블를 조회하는 실체화 뷰를 생성해도 빠른 리프레시는 수행이 불가능하다. 따라서 이 기능을 통해 실체화 뷰 관리를 Tibero쪽 데이터베이스가 담당하게 하고, 실제 저장소는 이기종 데이터베이스에 위치하게 하여 조회가 가능하게 한다. 현재 이기종 데이터베이스는 Oracle만 지원한다.

아래 설명을 하기 전에 베이스 테이블이 위치한 데이터베이스를 A, 실체화 뷰 저장소 테이블이 위치할 오라클 서버를 B이라 가정한다.

6.3.1. 실체화 뷰 생성 사전 작업

실체화 뷰 생성에 하기 전에 아래와 같은 사전 작업이 필요하다. 베이스 테이블이 위치한 데이터베이스를 A, 실체화 뷰 저장소 테이블이 위치할 오라클 서버를 B이라 가정한다.

- B의 해당 계정에 Tibero에서 제공하는 $TB_HOME/scripts/mview_remote_install.sql 스크립트를 수행하여 기능에 필요한 PSM 패키지를 설치한다.
- A에서 B로의 데이터베이스 링크를 생성한다.
- PREBUILT 옵션만 지원하므로, 저장소 테이블을 사전에 생성한다.

6.3.2. 실체화 뷰 생성

A에서 실체화 뷰를 생성하지만, 실제 조회가 가능한 저장소 테이블은 B에 있다.

CREATE MATERIALIZED VIEW를 생성할 때 AT dblink_name 속성을 추가하여, A에서 B로의 데이터베이스 링크를 지정한다. 정확한 문법은 “7.41. CREATE TABLE”을 참고한다.

6.3.3. 리프레시 및 저장소 테이블 조회

실체화 뷰 리프레시는 A에서 수행한다. 리프레시가 된 결과는 B의 저장소 테이블에 저장된다. 그 외의 사항에 대해서는 일반 실체화 뷰와 동일하다.

6.3.4. 제약 사항

원격 저장소를 가진 실체화 뷰는 아래와 같은 제약 사항이 있다.

- 베이스 테이블이 이기종 데이터베이스에 있고 저장소 테이블을 Tibero에 저장하는 실체화 뷰는 지원하지 않는다.
- 빠른 리프레시의 경우 실체화 뷰 질의가 집합 함수가 없는 단일 베이스 테이블에 대한 질의이어야 한다.
- 그 외의 사항은 일반 실체화 뷰 질의의 제약 조건과 동일하다.

제7장 데이터 정의어

본 장에서는 DDL를 자세히 설명한다. 먼저 다수의 DDL 명령어가 공통으로 포함하는 문법 요소를 설명하고, 그다음 각 명령어를 설명한다. DDL 명령어는 알파벳 순으로 나열하고, 각 명령어에 대한 설명과 문법, 예제를 기술한다. 문법을 설명할 때는 [그림 3.1]의 형식을 그대로 따르고, 키워드와 문법의 구성요소는 별도의 표로 설명한다.

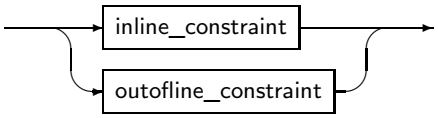
7.1. DDL 공통 문법 요소

7.1.1. 제약조건

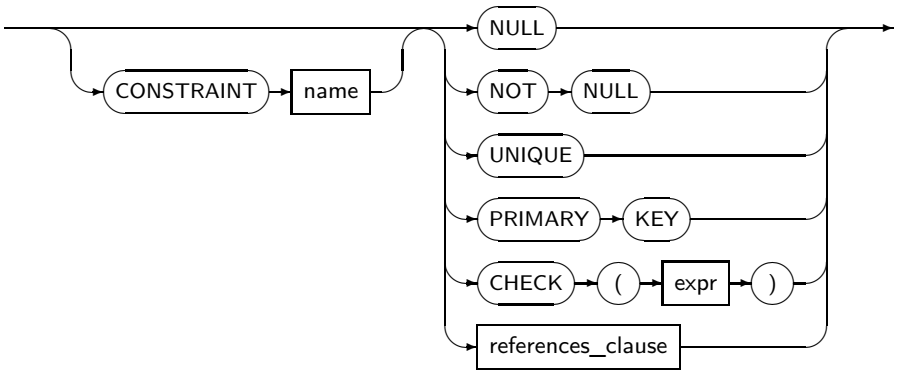
테이블 또는 뷰에 포함되는 데이터의 무결성을 위해 제한 사항을 설정할 수 있는데, 이를 제약조건이라고 한다. 제약조건의 이름은 생략할 수 있다. 생략하는 경우 시스템에서 유일한 이름으로 자동 생성된다. 제약조건에는 NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK가 있다.

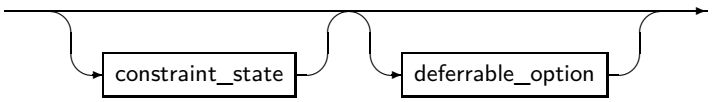
각각의 제약조건을 구성하는 문법은 다음과 같다.

constraint

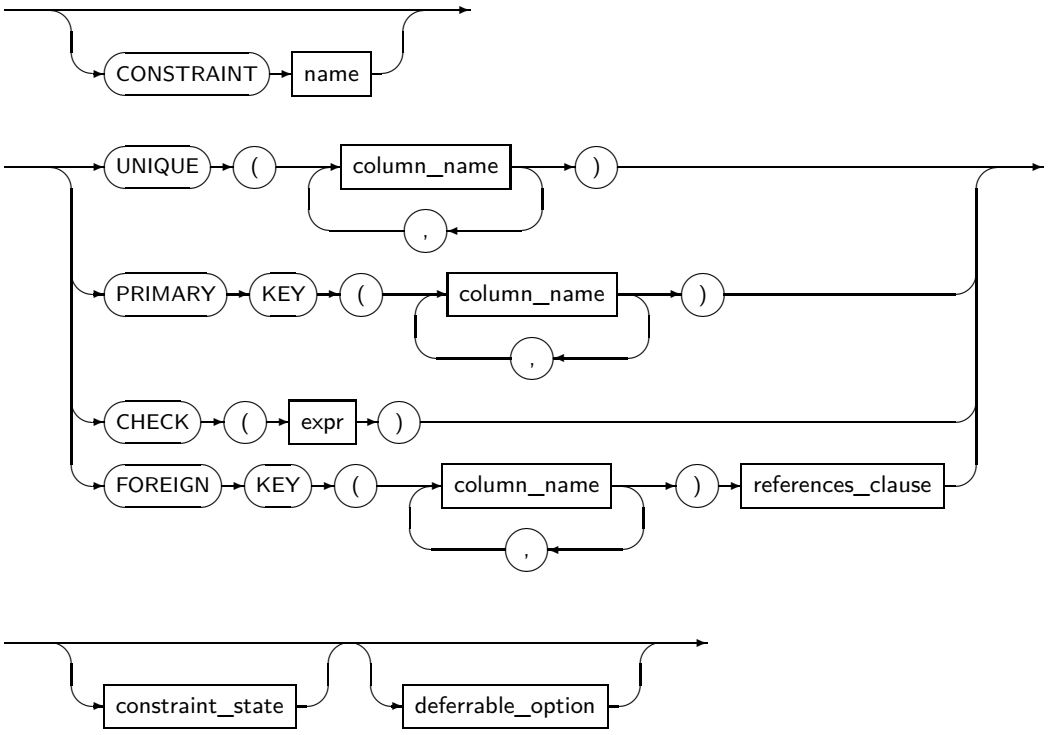


inline_constraint

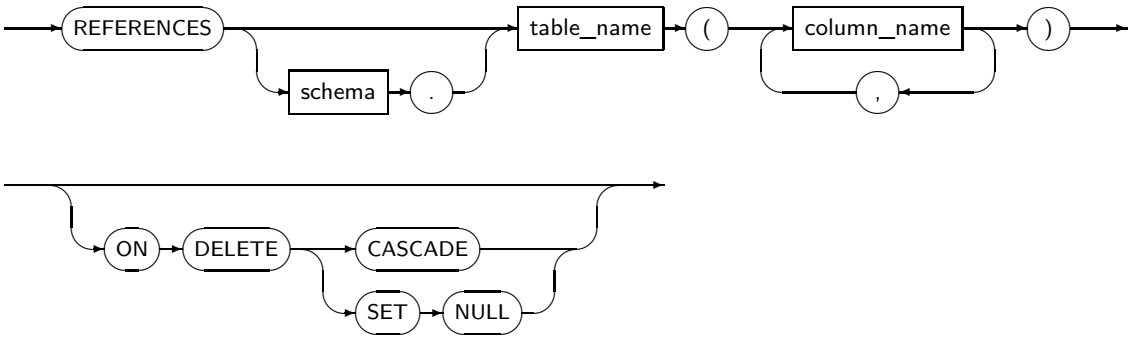




outofline_constraint



references_clause



NOT NULL

NOT NULL 제약조건은 해당 컬럼의 값으로 NULL을 허용하지 않고, 반드시 데이터를 입력해야 한다. 이와는 반대로 제약조건이 **NULL**이면 해당 컬럼은 NULL 값을 허용한다. 컬럼에 명시적으로 NULL이나 NOT NULL을 설정하지 않았다면 NULL이 컬럼의 디폴트로 설정된다.

UNIQUE

UNIQUE 제약조건은 해당 컬럼이 중복되는 데이터가 존재할 수 없는 유일성을 보장하는 제약조건이다. 한 로우(ROW)의 해당 컬럼의 조합(Combination) 값이 다른 로우(ROW)의 같은 컬럼의 조합 값으로 사용될 수 없다. 다시 말해 동시에 2개 이상의 로우(ROW)에서 동일한 컬럼의 조합 값을 갖는 것을 제한한다.

UNIQUE 제약조건은 해당 제약조건에 설정된 컬럼 값으로 NULL 값을 갖는 것을 허용한다. 만약 하나의 컬럼만을 갖는 UNIQUE 제약조건이라면 여러 개의 로우에서 해당 컬럼에 NULL 값을 가질 수 있다.

그러나 여러 개의 컬럼을 포함하는 UNIQUE 제약조건이라면 제약조건에 포함되는 컬럼 전부가 NULL인 경우를 제외하고, 컬럼의 일부가 NULL이면서 컬럼의 조합 값이 동일한 경우를 허용하지 않는다.

다음은 UNIQUE 제약조건을 설명하는 예이다.

```
UNIQUE(a, b)
```

위와 같이 제약조건이 컬럼 a, b에 설정되어 있다면 해당 컬럼들에 (1, NULL), (NULL, 2) 같은 값을 가진 로우(ROW)들은 허용이 된다. 하지만, 두 개 이상의 (1, NULL)은 허용하지 않는다. 그러나 모든 컬럼의 조합이 NULL인 경우 (NULL, NULL)와 같은 값을 가진 로우(ROW)들은 여러 개를 가질 수 있다.

PRIMARY KEY

PRIMARY KEY 제약조건은 NOT NULL 제약조건과 UNIQUE 제약조건의 결합과 같다. 테이블이나 뷰는 단 한 개의 PRIMARY KEY 제약조건을 가질 수 있다.

Tibero는 UNIQUE 제약조건이나 PRIMARY KEY 제약조건이 설정된 컬럼에 유일 인덱스(Unique Index)를 생성한다. 만약 constraint_state에서 인덱스를 설정하지 않는다면 자동으로 UNIQUE 인덱스를 생성하거나, 이미 존재하는 UNIQUE 인덱스를 사용한다.

FOREIGN KEY

FOREIGN KEY 제약조건은 같은 테이블 또는 서로 다른 두 개 테이블들의 키 컬럼 사이의 관계이다. 즉, 어떤 컬럼의 REFERENCED KEY와 FOREIGN KEY 사이의 관계이다. FOREIGN KEY로 지정된 컬럼의 값은 FOREIGN KEY가 참조하는 컬럼의 데이터 값만을 가질 수 있다. 복합(Composite) 컬럼으로 구성된 FOREIGN KEY도 마찬가지이다.

FOREIGN KEY로 지정된 컬럼과 REFERENCED KEY로 지정된 컬럼은 개수와 각각의 데이터 타입이 같아야 한다.

다음은 FOREIGN KEY 제약조건을 설명하는 예이다.

```
REFERENCED KEY = (1,2), (2,3), (4,2)
FOREIGN KEY   = (1,2), (2,3), (4,2)
```

위와 같이 REFERENCED KEY로 지정된 컬럼의 값이 (1,2), (2,3), (4,2)가 존재한다면, FOREIGN KEY로 지정된 컬럼의 값도 (1,2), (2,3), (4,2)만 가능하다. 다른 값은 허용되지 않는다. 다른 테이블의 컬럼을 참조하는 경우에는 해당 테이블에 대한 REFERENCE OBJECT 특권이 필요하다.

하나의 컬럼은 여러 개의 FOREIGN KEY 제약조건에 참여할 수 있다. REFERENCED KEY는 이미 UNIQUE나 PRIMARY KEY로 지정된 컬럼이어야만 한다.

REFERENCED KEY는 같은 테이블의 컬럼이 될 수도 있고 다른 테이블의 컬럼이 될 수도 있다. 같은 테이블의 컬럼을 참조하는 경우 테이블 이름은 생략할 수 있다. REFERENCED KEY의 PRIMARY KEY 값과 UNIQUE 값을 삭제하면 참조 무결성을 해칠 수 있기 때문에 이를 보장하기 위해 FOREIGN KEY도 같이 처리를 해주어야만 한다.

Tibero는 FOREIGN KEY 제약조건이 지정된 컬럼에 인덱스를 생성하지 않는다.

FOREIGN KEY를 지정할 때 설정할 수 있는 옵션은 다음과 같다.

옵션	설명
ON DELETE	references_clause의 ON DELETE 옵션을 이용하여 참조된 컬럼 값이 삭제될 때 참조하는 컬럼 값에 대한 동작을 설정할 수 있다.
CASCADE	참조되는 컬럼 값이 삭제될 때 참조하는 컬럼 값도 같이 삭제한다.
SET NULL	외참조되는 컬럼 값이 삭제될 때 참조하는 컬럼 값을 NULL로 변경한다.

CHECK

CHECK 제약조건은 expr로 표현한 조건이 항상 참이 되도록 유지한다. 테이블에 DML이 있을 때마다 조건을 평가한 후 그 조건을 만족하지 못하면 에러를 발생한다.

inline_constraint의 CHECK 제약조건인 경우는 해당 컬럼의 한 개만 조건의 수식에 포함할 수 있고, outofline_constraint의 CHECK 제약조건인 경우는 제약조건이 선언된 테이블의 모든 컬럼이 나올 수 있다. 다른 테이블의 컬럼은 조건의 수식에 포함할 수 없다.

참고

UNIQUE, PRIMARY KEY, FOREIGN KEY가 포함할 수 있는 컬럼 수는 최대 32개이다. UNIQUE, PRIMARY KEY 제약조건은 구성되는 컬럼으로 인덱스를 사용하기 때문에 컬럼 전체의 길이가 인덱스를 생성할 수 있는 최대 길이를 넘을 수 없다. 같은 컬럼으로 구성된 UNIQUE와 PRIMARY KEY를 동시에 설정할 수 없다.

LONG, LONG RAW는 UNIQUE, PRIMARY KEY, FOREIGN KEY 제약조건의 키 컬럼에 포함될 수 없다.

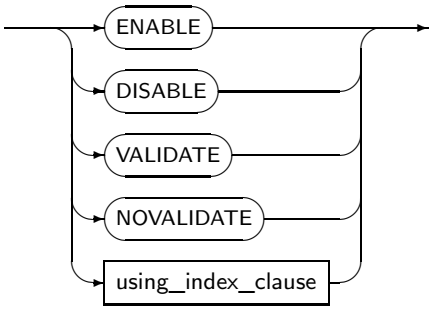
7.1.2. Constraint_state

Constraint_state는 제약조건을 활성화하거나 비활성화하고 이미 삽입된 데이터가 제약조건을 만족하는지 검증하는 역할을 한다. Constraint_state는 테이블이 처음 생성될 때나 컬럼이 새로 추가될 때처럼 제약조건이 생성되기 전에 사용할 수 있다. 또한, Constraint_state는 테이블이 생성되고 나서도 제약조건을 추가로 설정할 때 사용할 수 있다.

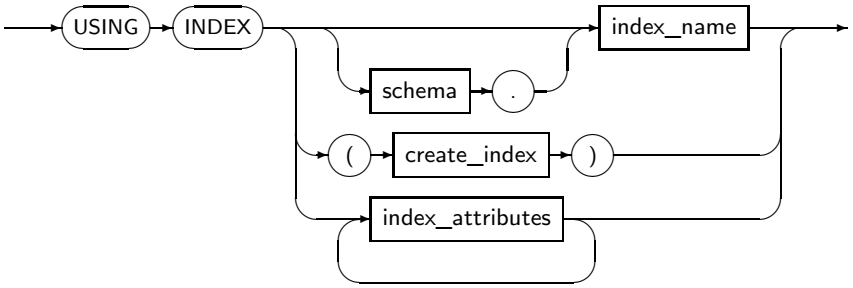
Constraint_state의 세부 내용은 다음과 같다.

- 문법

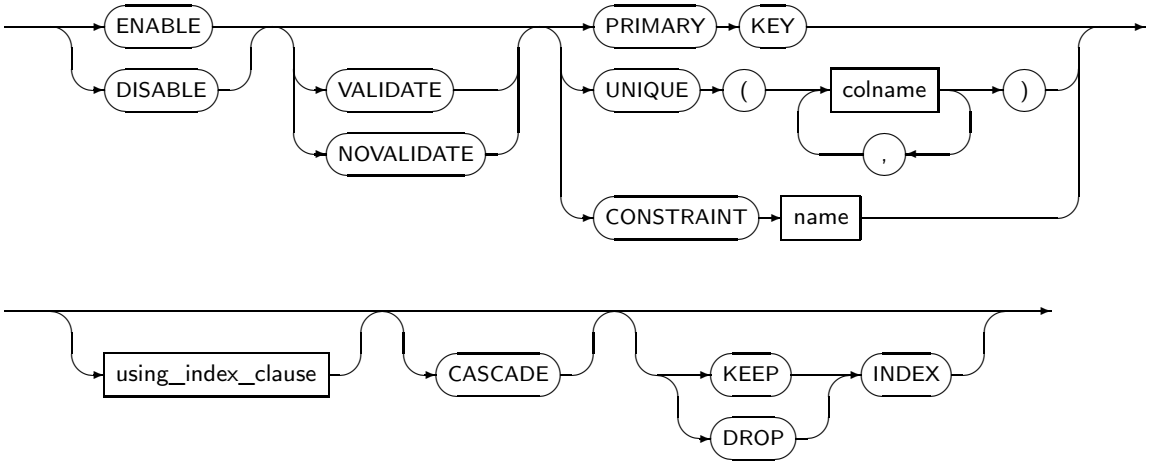
constraint_state



using_index_clause



atbl_con_alterstate_cl



- 구성요소

  – constraint_state

구성요소	설명
ENABLE	새로 삽입될 데이터에 제약조건을 적용한다. 처음 제약조건이 생성될 때 ENABLE이나 DISABLE을 지정하지 않으면 ENABLE이 디폴트로 설정된다. DISABLE 상태여서 적용되지 않은 제약조건에 ENABLE을 적용한다면, 앞으로 삽입될 데이터에 제약조건을 다시 적용시킬 수 있다.

구성요소	설명
	만약 UNIQUE나 PRIMARY KEY 제약조건에 ENABLE이 적용되면, 이미 존재하는 UNIQUE 인덱스를 사용한다. 만약 UNIQUE 인덱스가 존재하지 않는 상태라면 새로 UNIQUE 인덱스를 생성한다. FOREIGN KEY 제약조건의 경우 비유일 인덱스에 대해 같은 동작을 한다.
DISABLE	새로 삽입될 데이터에 제약조건을 적용하지 않는다. 만약 UNIQUE나 PRIMARY KEY 제약조건, FOREIGN KEY 제약조건을 DISABLE로 설정하면, 관련 인덱스를 삭제한다. DISABLE로 설정된 UNIQUE나 PRIMARY KEY를 참조하는 FOREIGN KEY는 ENABLE로 설정할 수 없다. 만약 제약조건을 생성할 때 같이 만들어진 인덱스가 아닌, 제약조건이 생성되면서 원래 존재하던 인덱스를 사용한 경우라면 인덱스가 삭제되지는 않는다.
VALIDATE	이미 테이블에 삽입된 데이터가 제약조건을 만족하는지 체크하고 보장한다. 만약 VALIDATE로 설정했지만, 데이터가 이 제약조건을 만족하지 못하는 경우에는 에러를 반환하고 VALIDATE가 들어간 DDL 문장은 실패한다. DISABLE VALIDATE 상태로 만든다면, 이미 삽입된 데이터의 제약조건에 대한 완전성을 보장하면서 앞으로 삽입될 데이터에 대해서는 보장하지 않겠다는 의미가 되기 때문에 이 상태의 제약조건이 존재하는 테이블에 대해서는 데이터의 삽입, 삭제, 갱신을 할 수 없게 된다.
NOVALIDATE	이미 테이블에 삽입된 데이터에 대해 제약조건을 만족하는지 검사하지 않는다.
using_index_clause	UNIQUE나 PRIMARY KEY를 설정할 때 사용할 인덱스를 지정하거나 생성하여 사용할 수 있다. schema.index_name을 지정하면 해당 인덱스를 찾아서 사용하고, create_index를 사용하면 인덱스를 생성하여 사용한다. create_index와 index_attributes는 “7.30. CREATE INDEX”를 참고한다.

– using_index_clause

구성요소	설명
schema	사용할 인덱스가 속해 있는 스키마를 명시한다. 생략할 경우 현재 사용자의 스키마로 인식된다.
index_name	사용할 인덱스의 이름을 명시한다.
create_index	인덱스를 새로 생성하여 사용하려고 할 때 명시한다. 자세한 내용은 “7.30. CREATE INDEX”를 참고한다.

구성요소	설명
index_attributes	인덱스의 속성을 설정할 때 사용한다. 자세한 내용은 “7.30. CREATE INDEX”를 참고한다.

– atbl_con_alterstate_cl

구성요소	설명
ENABLE	구성요소 설명 중 constraint_state를 참고한다.
DISABLE	구성요소 설명 중 constraint_state를 참고한다.
VALIDATE	구성요소 설명 중 constraint_state를 참고한다.
NOVALIDATE	구성요소 설명 중 constraint_state를 참고한다.
PRIMARY KEY	PRIMARY KEY 제약조건을 설정한다.
UNIQUE column_name	UNIQUE 제약조건을 설정한다. column_name에는 UNIQUE 제약조건을 지정할 컬럼의 이름을 명시한다.
CONSTRAINT constraint_name	상태를 변경할 제약조건의 이름을 명시한다.
using_index_clause	구성요소 설명 중 constraint_state를 참고한다.
CASCADE	참조된 UNIQUE나 PRIMARY KEY 제약조건을 비활성화할 때 관련된 FOREIGN KEY까지 함께 비활성화를 하기 위해 사용한다. FOREIGN KEY가 존재하는 제약조건을 비활성화하는 경우에는 반드시 포함되어야 한다.
KEEP INDEX	제약조건을 비활성화하면서 제약조건에서 사용한 인덱스를 제거하지 않고 그냥 유지하고자 할 때 명시한다. 기본값이므로 생략할 수 있다.
DROP INDEX	제약조건을 제거하려고 할 때 그 제약조건에서 사용한 인덱스도 함께 제거하고자 할 때 명시한다.

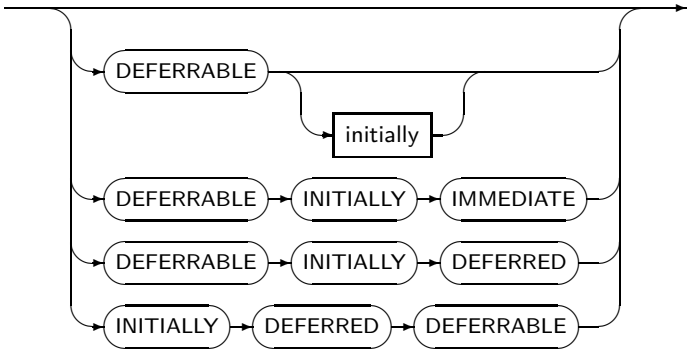
7.1.3. Deferrable_option

제약 조건을 매 DML의 경우 체크하는 것이 아니라, 커밋 시점에 마지막 커밋된 시점 이후에 DML이 된 데이터에 대하여 제약 조건을 확인하도록 하는 기능이다. 커밋 시점에 수행하는 제약 조건을 통과 하지 못했을 경우 마지막 커밋 시점으로 데이터가 전부 rollback된다. FOREIGN KEY를 제외한 제약 조건에서 사용 가능하다.

Deferrable_option의 세부 내용은 다음과 같다.

- 문법

deferrable_option



- 구성요소
  - deferrable_option

구성요소	설명
DEFERRABLE (INITIALLY (IMMEDIATE))	Deferrable constraint 기능을 사용하는 제약 조건 생성문이다. Deferrable constraint 기능은 사용하지만, 일반 제약조건과 마찬가지로 DML의 경우에 매번 바로바로 조건을 체크할 경우에 사용된다. INITIALLY 또는 IMMEDIATE가 생략되어도 의미는 같다.
DEFERRABLE INITIALLY DEFERRED	Deferrable constraint 기능을 사용하며, 제약 조건을 commit 시점에만 체크하도록 할 경우에 사용된다.
INITIALLY DEFERRED DEFERRABLE	Deferrable constraint 기능을 사용하며, 제약 조건을 commit 시점에만 체크하도록 할 경우에 사용된다.

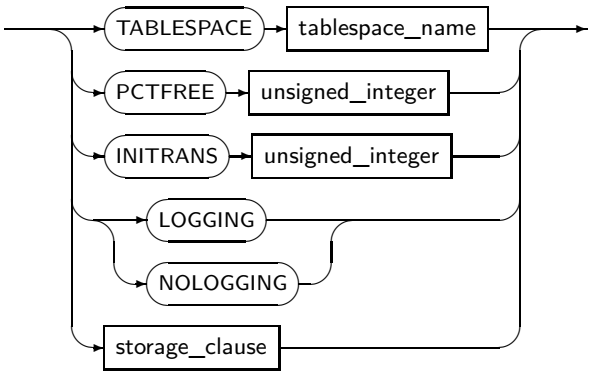
7.1.4. Sgmt_attr

저장 공간의 물리적인 성질과 테이블 스페이스 등을 지정하기 위한 문장이다.

Sgmt_attr의 세부 내용은 다음과 같다.

- 문법

sgmt_attr



- 구성요소

구성요소	설명
TABLESPACE tablespace_name	데이터가 저장되는 테이블 스페이스를 지정할 수 있다. 지정하지 않으면 사용자의 디폴트 테이블 스페이스를 사용하게 된다. 임시 테이블의 경우 디폴트로 지정된 임시 테이블 스페이스를 사용한다. tablespace_name에 테이블 스페이스의 이름을 명시한다.
PCTFREE unsigned_integer	데이터를 디스크 블록에 저장할 때 데이터가 변경되어 크기가 증가할 것에 대비하여 얼마만큼의 영역을 예비로 남겨둘지를 설정하는 값이다. 1 ~ 99 사이의 값을 설정할 수 있으며, 지정하지 않으면 기본값은 10이다. unsigned_integer에 해당 값을 명시한다.
INITRANS unsigned_integer	디스크 블록마다 트랜잭션 엔트리(Transaction Entry)를 위한 공간을 몇 개를 확보할 것인가를 나타낸다. 트랜잭션 엔트리는 블록에 공간이 남아있다면 필요할 때 확장된다. 따라서 미리 큰 값을 설정할 필요는 없다. 최솟값은 1이며, 최댓값은 디스크 블록의 크기에 따라 다르다. 지정하지 않으면 기본값은 2이다. unsigned_integer에 해당 값을 명시한다.
LOGGING / NOLOGGING	Direct-Path Loading을 이용하는 경우 Redo 로그를 남기지 않는다. 단, Archive Mode를 사용할 경우엔 로그를 남긴다. 지정하지 않으면 기본값은 LOGGING이다. 현재 파티션 테이블별로 LOGGING 옵션을 설정하는 것은 지원하지 않는다.
storage_clause	세그먼트의 세부적인 속성을 정의한다. 자세한 내용은 “7.1.5. Storage_clause”를 참고한다.

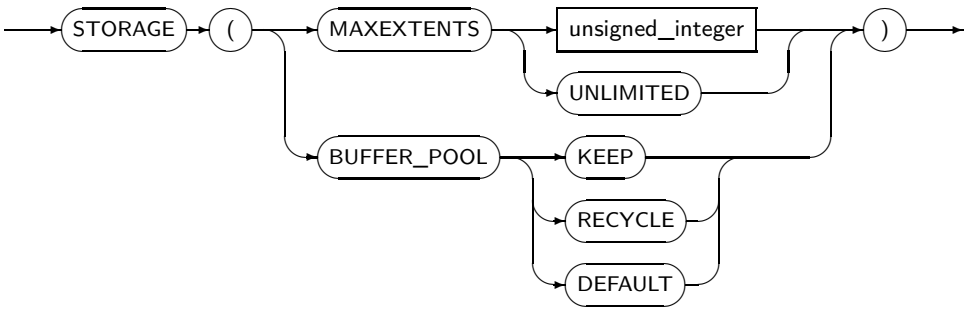
7.1.5. Storage_clause

세그먼트의 세부적인 속성을 정의하기 위한 문장이다.

Storage_clause의 세부 내용은 다음과 같다.

- 문법

storage_clause



- 구성요소

구성요소	설명
MAXEXTENTS	세그먼트에 할당되는 최대 익스텐트의 개수를 지정한다. ALTER TABLE 문에서는 예외적으로 MAXEXTENTS 값 조정이 불가능하다.
unsigned_integer	제한할 익스텐트의 개수를 명시한다.
UNLIMITED	익스텐트의 개수를 제한하지 않는다. 지정하지 않으면, 기본값은 UNLIMITED이다.
BUFFER_POOL	해당 세그먼트의 데이터 블록을 어떤 Buffer Pool에 넣을 것인지를 지정한다.
KEEP	세그먼트의 블록을 KEEP Buffer Pool에 넣어 메모리에 보존하도록 한다. 이로써 I/O 연산을 하는 시간을 줄일 수 있다. $TB_SID.tip 파일에 DB_KEEP_CACHE_SIZE 파라미터가 설정되어 있어야 버퍼 캐시에 KEEP Buffer Pool이 설정된다. DB_KEEP_CACHE_SIZE 파라미터가 설정되어 있지 않으면 BUFFER_POOL KEEP 지정은 의미가 없다.
RECYCLE	세그먼트의 블록을 RECYCLE Buffer Pool에 넣어 DEFAULT Buffer Pool이 불필요한 버퍼 캐시를 저장하지 않도록 한다. $TB_SID.tip 파일에 DB_RECYCLE_CACHE_SIZE 파라미터가 설정되어 있어야 버퍼 캐시에 RECYCLE Buffer Pool이 설정된다. DB_KEEP_CACHE_SIZE 파라미터가 설정되어 있지 않으면 BUFFER_POOL RECYCLE 지정은 의미가 없다.

구성요소	설명
DEFAULT	DEFAULT로 지정하거나 BUFFER_POOL 옵션을 지정하지 않으면, DEFAULT Buffer Pool을 사용한다.

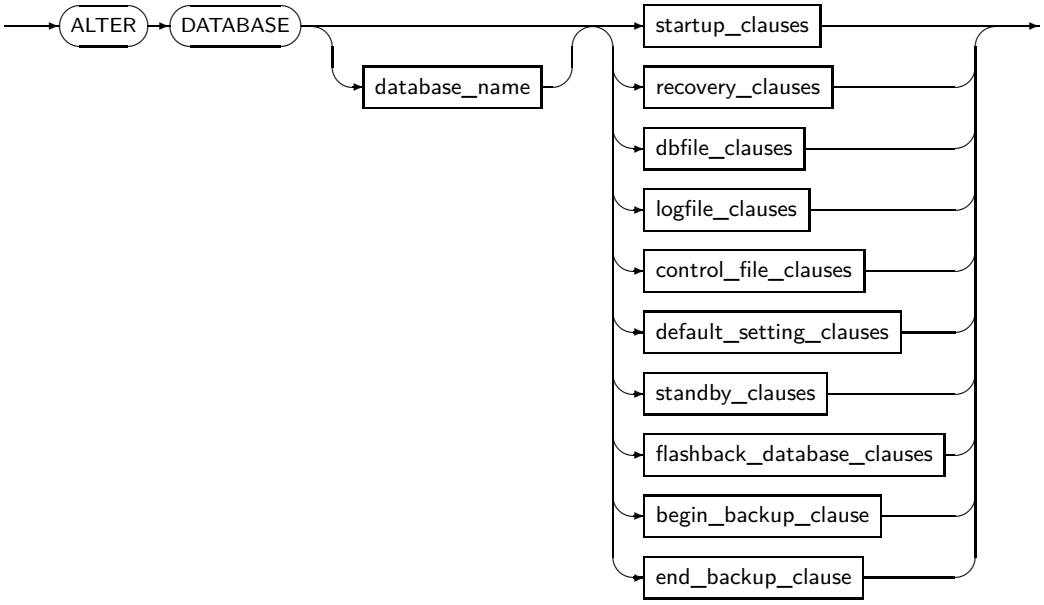
7.2. ALTER DATABASE

데이터베이스의 상태나 구성파일을 변경한다. 또한 데이터베이스를 복구하는 데에도 사용된다.

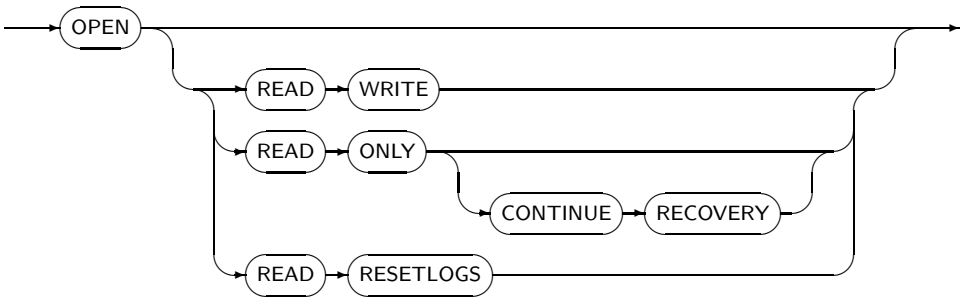
ALTER DATABASE의 세부 내용은 다음과 같다.

- 문법

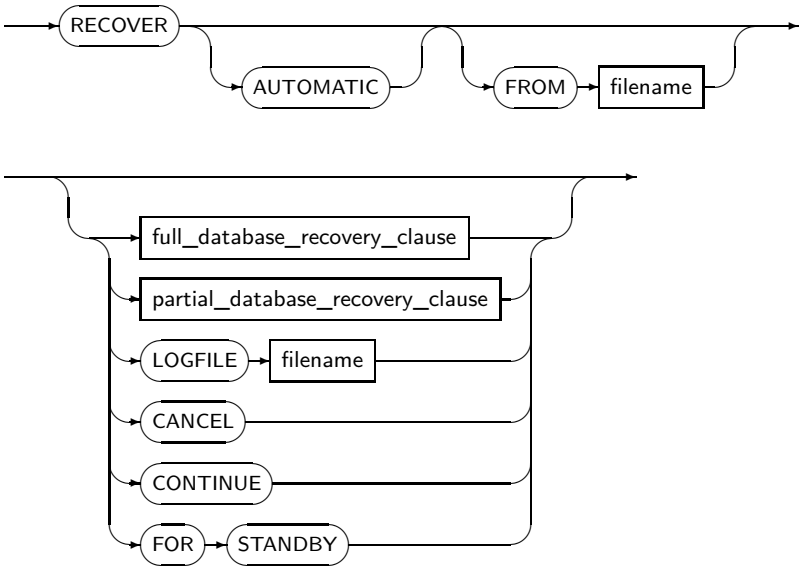
alter_database



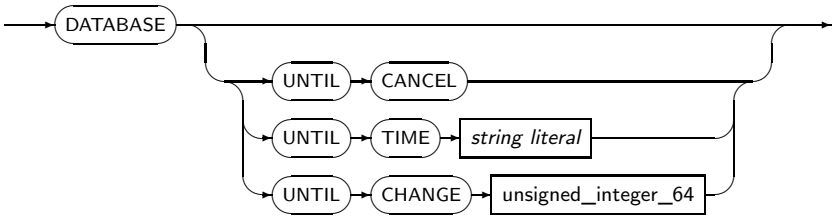
startup_clauses



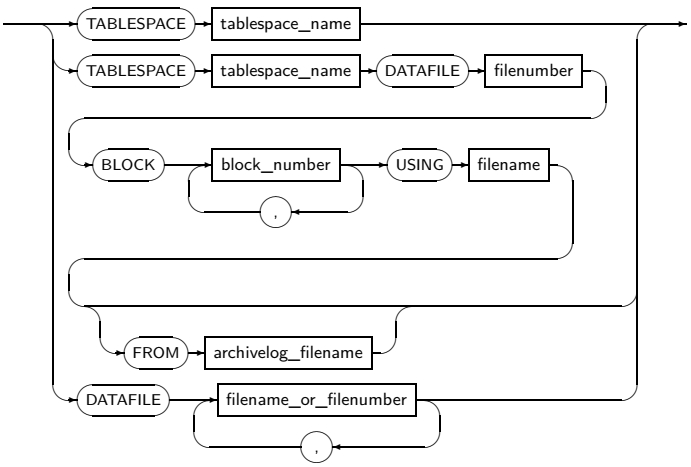
recovery_clauses



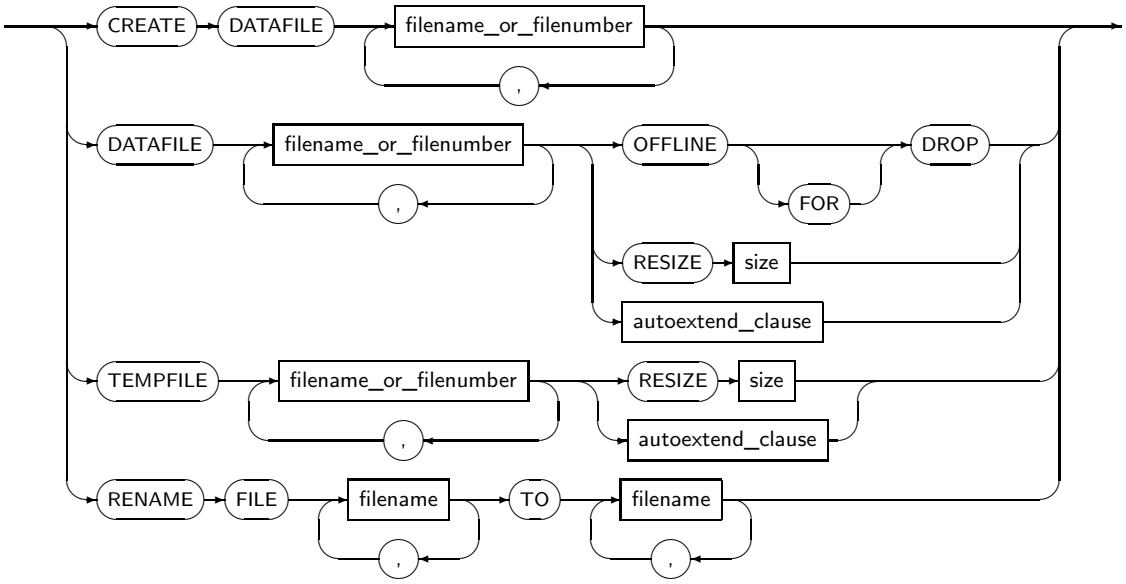
full_database_recovery_clause



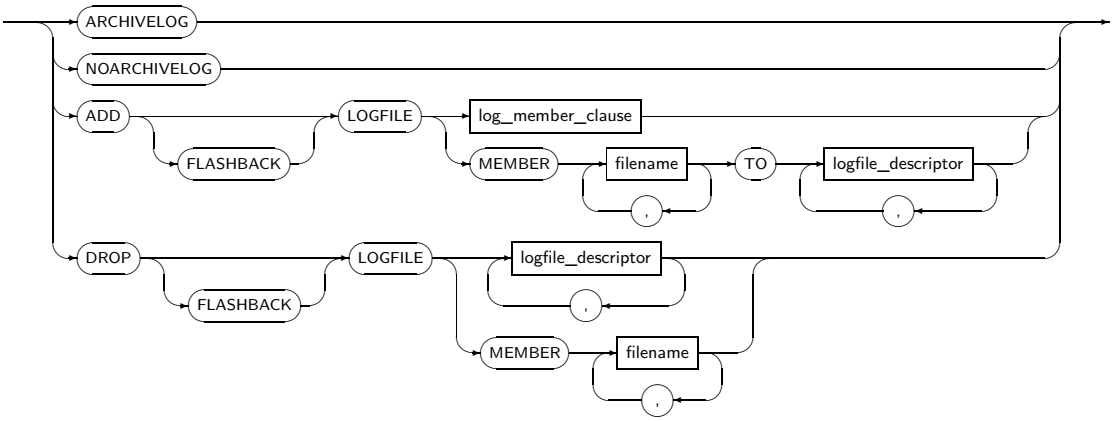
partial_database_recovery_clause



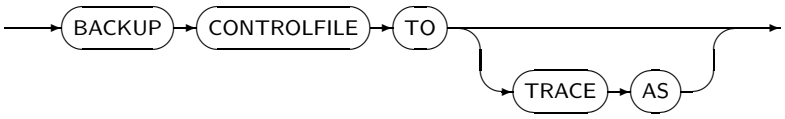
dbfile_clauses

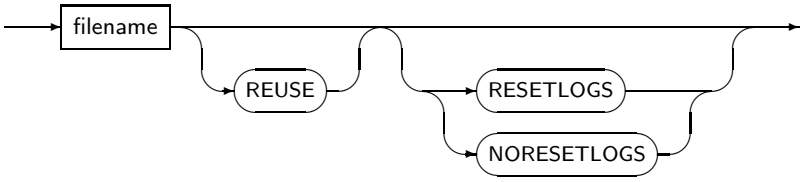


logfile_clauses

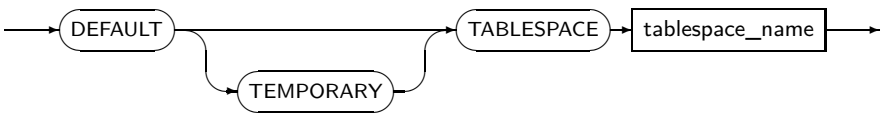


control_file_clauses

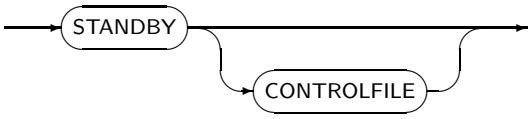




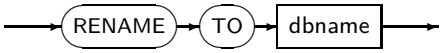
default_setting_clauses



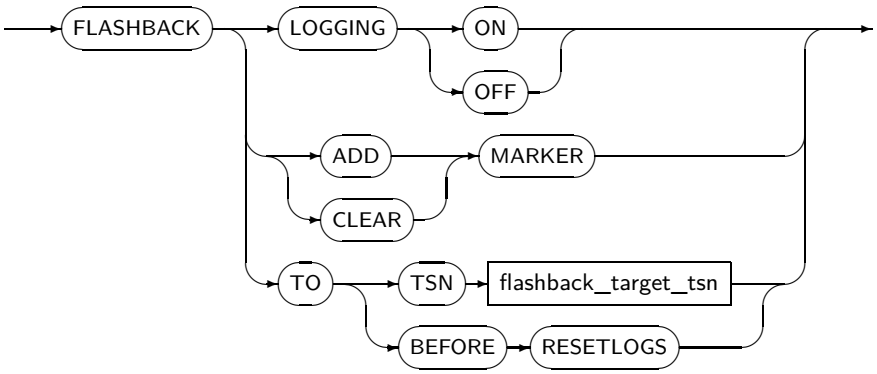
standby_clauses



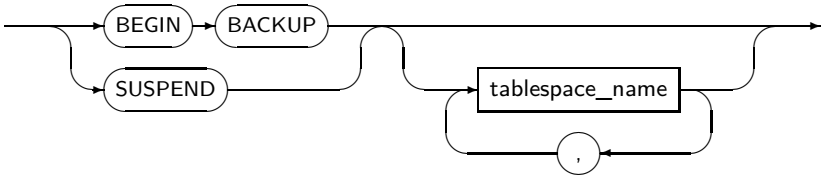
rename_clauses



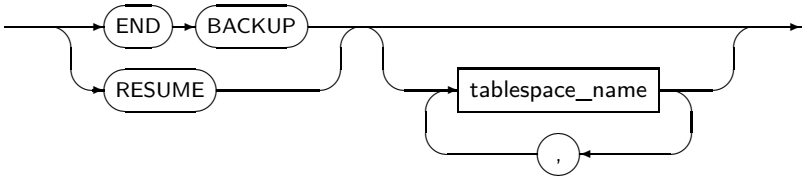
flashback_database_clauses



begin_backup_clauses



end_backup_clauses



- 특권

  SYSDBA 특권이 있어야 ALTER DATABASE를 실행할 수 있다.

- 구성요소

  – alter_database

구성요소	설명
database_name	변경할 데이터베이스를 명시한다. database_name은 생략할 수 있다. $TB_SID.tip 파일에 설정된 DB_NAME 파라미터의 값과 같아야 한다.

구성요소	설명
startup_clauses	startup_clauses는 데이터베이스를 정상적으로 사용할 수 있도록 기동할 때 사용하는 절이다. MOUNT 모드에서만 사용할 수 있다.
recovery_clauses	recovery_clauses는 미디어 복구(Media Recovery)를 위해 사용하는 절이다. MOUNT 모드에서만 사용할 수 있다.
dbfile_clauses	dbfile_clauses는 데이터 파일의 추가, 재생성 등의 작업을 하기 위해 사용하는 절이다. 파일을 복구하는 문장과 데이터 파일이나 임시 파일의 속성을 수정하는 문장으로 나뉜다. 파일의 지정은 파일의 전체 경로나 파일 번호를 명시하여 지정할 수 있다. 데이터 파일과 임시파일의 파일 번호는 다음의 뷰를 통해 알 수 있다. – V$DATAFILE – V$TEMPFILE – V$RECOVER_FILE – DBA_DATA_FILES – DBA_TEMP_FILES
logfile_clauses	logfile_clauses는 ARCHIVELOG 모드를 설정하거나 로그 파일을 추가, 제거하기 위한 절이다. MOUNT 모드에서만 사용할 수 있다.
control_file_clauses	control_file_clauses는 컨트롤 파일을 물리적으로 복사하여 명시한 파일로 백업 본을 만들기 위한 절과 컨트롤 파일을 새로 생성하는 문장을 명시한 파일에 저장하기 위한 절로 나뉜다. 후자의 경우에는 다음과 같은 여러 옵션을 함께 사용할 수 있다. 파일이 이미 존재하는 경우 REUSE 옵션을 사용해야 한다. 컨트롤 파일을 생성하는 문장은 RESETLOGS를 사용할 경우와 그렇지 않은 경우로 나눌 수 있다.
default_setting_clauses	default_setting_clauses는 데이터베이스의 디폴트 테이블 스페이스를 변경하기 위한 절이다. TEMPORARY 테이블 스페이스와 일반 테이블 스페이스를 변경할 수 있다.
standby_clauses	standby_clauses는 Standby 데이터베이스와 관련된 작업을 할 수 있다.
flashback_database_clauses	플래시백 데이터베이스와 관련된 작업을 할 수 있다.
begin_backup_clause	데이터베이스 운영 중 백업을 시작할 때 사용된다. 백업 대상 테이블 스페이스들을 별도로 지정하지 않은 경우 데이터베이스 전체에 대해 실행된다.
end_backup_clause	데이터베이스 백업을 끝낼 때 사용된다.

– startup_clauses

구성요소	설명
READ WRITE	데이터베이스를 **READ WRITE** 모드로 기동시킨다. 사용자가 읽기와 쓰기를 모두 할 수 있고, Redo 로그를 저장한다. 기본값이므로 생략할 수 있다.
READ ONLY	사용자가 읽기만을 할 수 있다. 따라서 Redo 로그를 저장하지 못한다.
READ ONLY CONTINUE RECOVERY	Standby 데이터베이스를 읽기 전용으로 전환한 상태에서 동시에 Primary 데이터베이스로부터 전송받은 Redo 로그를 적용하는 모드로 전환한다. Standby 상태에서만 사용할 수 있다. 자세한 내용은 "Tibero 관리자 안내서"를 참고한다.
READ RESETLOGS	로그의 시퀀스를 1로 초기화하고, 아카이브되지 않은 로그를 아카이브한다. 남아 있는 Redo 로그 정보는 적용될 일이 없어서 삭제된다.

– recovery_clauses

구성요소	설명
AUTOMATIC	AUTOMATIC을 사용해 복구 과정을 자동화하면, 필요한 로그 파일을 일일이 지정하지 않고 원하는 조건까지 자동 복구를 할 수 있다. 자동 복구를 위해서는 컨트롤 파일에 온라인 로그 파일과 아카이브 로그 파일이 제대로 명시되어 있어야 한다. FROM 절을 통해 사용할 아카이브 로그 파일이 있는 디렉터리를 지정할 수 있다. FROM 절로 지정하지 않으면 LOG_ARCHIVE_DEST 파라미터에 설정된 디렉터리를 사용한다.
full_database_recovery_clause	데이터베이스 전체에 대한 미디어 복구를 시작할 때 사용하는 절이다. 완전 복구 또는 불완전 복구를 지정할 수 있다. full_database_recovery_clause를 생략하면 데이터베이스 전체에 대한 완전 복구를 시작한다.
partial_database_recovery_clause	데이터베이스 전체가 아닌 특정 테이블 스페이스나 데이터 파일에 미디어 복구를 시작할 때 사용한다.
LOGFILE	적용할 로그 파일을 지정하고, 복구를 진행할 수 있다.
filename	지정할 파일의 이름을 명시한다.
CANCEL	복구를 마칠 때 사용한다.
FOR STANDBY	Primary의 Hot Backup으로 Standby를 구축할 때 사용한다.

복구는 다음과 같이 3단계로 진행된다.

단계	명칭	설명
1	시작	복구를 시작하는 단계이다.
2	진행	로그 파일을 사용하여 복구를 순차적으로 진행하는 단계이다.
3	종료	복구를 종료하는 단계이다.

복구는 다음과 같이 두 가지 종류로 구분된다.

구분	설명
완전 복구	종료 시점을 지정할 수 없으며, 가장 최근의 로그 파일의 내용까지 적용하여 복구를 수행한다.
불완전 복구	종료 시점을 명시적으로 지정할 수 있다. 종료 시점의 지정은 TSN 값이나 시간을 사용하여 지정할 수 있다. 또한, CANCEL 문을 사용해 원하는 시점에서 종료를 할 수도 있다.

– full_database_recovery_clause

구성요소	설명
UNTIL CANCEL	불완전 복구를 할 때 사용한다. UNTIL CANCEL을 사용한 뒤에 복구할 로그 파일을 명시한다.
UNTIL TIME	특정 시간까지 불완전 복구를 진행할 때 사용한다. 명시할 시간은 날짜형 리터럴을 사용한다.
string literal	UNTIL TIME을 사용할 때 날짜형 리터럴을 명시한다.
UNTIL CHANGE	데이터베이스의 TSN 값을 기준으로 복구를 할 때 사용한다. TSN 값은 V$LOG 뷰를 이용하여 조회할 수 있다.
unsigned_integer_64	UNTIL CHANGE를 사용할 때 TSN 값을 명시한다.

– partial_database_recovery_clause

구성요소	설명
TABLESPACE	특정 테이블 스페이스만 복구할 때 사용한다. 주로 OFFLINE IMMEDIATE로 오프라인된 테이블 스페이스를 온라인으로 전환할 때 사용한다.
tablespace_name	미디어 복구 대상이 되는 테이블 스페이스의 이름을 명시한다.
filename_or_filenumber	미디어 복구 대상이 되는 데이터 파일의 이름을 명시하거나 데이터 파일의 번호를 명시한다.
block_number	복구할 데이터 파일의 블록 번호를 명시한다.
archivelog_filename	복구에 사용할 아카이브 로그를 지정한다.

– dbfile_clauses

구성요소	설명
CREATE	데이터 파일이 없는 경우 빈 데이터 파일을 만드는 데 사용된다. 생성 후 미디어 복구를 통해 데이터 파일을 복구할 수 있다. MOUNT 모드에서만 사용할 수 있다.
DATAFILE	데이터 파일을 명시하는 부분이다. 파일의 이름을 명시할 수도 있고, 파일의 번호를 명시할 수도 있다. 여러 개의 파일을 동시에 명시할 수 있으며, 각각의 파일은 콤마(,)를 사용해 구분한다.
filename_or_filenumber	파일의 이름 또는 파일의 번호를 명시하는 부분이다.
OFFLINE FOR DROP	데이터 파일을 복구할 수 없을 때 해당 데이터 파일이 속한 테이블 스페이스를 제거하는 조건으로 데이터베이스를 운영할 때 사용한다. MOUNT 모드에서만 사용할 수 있다.
RESIZE size	파일의 크기를 증가시키거나 감소 시킬 때 사용한다. size 부분에 파일의 크기를 명시한다. 파일의 크기는 (BLOCK 개수 * DB_BLOCK_SIZE)이다. 단, 파일의 크기를 줄이는 경우 현재 사용하고 있는 크기와 입력받은 크기 중 큰 값이 파일 사이즈로 지정된다.
autoextend_clause	AUTOEXTEND 속성을 변경할 수 있다. 자세한 내용은 “7.25. CREATE DATABASE”를 참고한다.
TEMPFILE	임시 파일을 명시하는 부분이다. 파일의 이름을 명시할 수도 있고, 파일의 번호를 명시할 수도 있다. 여러 개의 파일을 동시에 명시할 수 있으며, 각각의 파일은 콤마(,)를 사용해 구분한다.
RENAME FILE	파일의 이름을 변경할 때 사용한다.
filename TO filename	TO의 앞부분에 변경할 파일의 이름을 명시하고, TO 뒷부분에 변경 뒤의 파일의 이름을 명시한다. 여러 개의 파일을 동시에 변경할 수 있으며, 여러 개의 파일을 동시에 변경할 경우 각각의 파일은 콤마(,)를 사용해 구분한다.

– logfile_clauses

구성요소	설명
ARCHIVELOG	ARCHIVELOG 모드를 설정할 수 있다.
NOARCHIVELOG	NOARCHIVELOG 모드를 설정할 수 있다. NOARCHIVELOG 모드로 운영되는 데이터베이스는 복구가 매우 제한적이다.
ADD LOGFILE	온라인 로그 파일을 추가할 수 있다. 로그 그룹 전체를 추가하거나 로그 그룹에 멤버를 추가할 수 있다. 추가할 로그 파일은 절대경로로 명시해야 한다. 현재 로그 그룹에 대해서는 수행할 수 없다. MOUNT 모드에서뿐만 아니라 데이터베이스 운영 중에도 가능하다.
log_member_clause	log_member_clause는 GUOUP 절을 사용해 로그 그룹의 번호를 지정하기 위한 절이다. 자세한 내용은 “7.25. CREATE DATABASE”를 참고한다.
MEMBER	로그 그룹 내의 특정 로그 멤버 파일을 지정할 때 사용한다.
filename	파일의 이름을 명시한다.
logfile_descriptor	로그 그룹이나 로그 파일을 명시한다.
DROP LOGFILE	온라인 로그 파일을 제거할 수 있다. 로그 파일을 추가할 때처럼 제거할 때도 그룹 전체를 제거하거나 그룹의 멤버만 제거할 수 있다. 로그 그룹은 반드시 2개 이상 있어야 하며, 제거할 로그 파일은 절대경로로 명시해야 한다. 현재 로그 그룹에 대해서는 수행할 수 없다. MOUNT 모드에서뿐만 아니라 데이터베이스 운영 중에도 가능하다.

– logfile_descriptor

구성요소	설명
GROUP group	로그 그룹 단위로 로그 파일을 추가하거나 제거할 때 해당 로그 그룹을 명시한다.
filename	로그 멤버 단위로 로그 파일을 추가하거나 제거할 때 해당 파일의 이름을 명시한다.

– control_file_clauses

구성요소	설명
BACKUP CONTROLFILE TO	컨트롤 파일의 물리적 복사본을 백업할 때 사용한다.

구성요소	설명
BACKUP CONTROLFILE TO TRACE AS	컨트롤 파일의 생성 문장을 백업할 때 사용한다.
filename	컨트롤 파일의 물리적 복사본 또는 생성 문장을 저장할 파일을 지정한다.
REUSE	컨트롤 파일의 생성 문장을 백업할 때 이미 존재하는 파일을 재사용하려면 REUSE 옵션을 사용해야 한다.
RESETLOGS	기존의 로그 파일은 무시하고, 로그를 초기화한다.
NORESETLOGS	기존의 유효한 로그 파일을 계속 사용한다.

– default_setting_clauses

구성요소	설명
DEFAULT	DEFAULT 테이블 스페이스를 변경할 때 사용하는 예약어이다.
TEMP	TEMPORARY 테이블 스페이스임을 지정한다. TEMPORARY 테이블 스페이스를 쓰지 않으면 일반 테이블 스페이스를 의미한다.
TABLESPACE	테이블 스페이스의 이름을 지정할 때 사용하는 예약어이다.
tablespace_name	테이블 스페이스의 이름을 명시한다.

– standby_clauses

구성요소	설명
STANDBY	데이터베이스를 Standby 모드로 전환한다. MOUNT, READ ONLY 모드에서만 사용할 수 있다.
STANDBY CONTROLFILE	초기화 파라미터 중 STANDBY_FILE_NAME_CONVERT에 지정된 경로에 따라 컨트롤 파일에 저장된 데이터베이스 파일의 경로를 변환하여 Primary 데이터베이스에서 복사한 컨트롤 파일을 Standby 데이터베이스용으로 사용할 수 있게 해준다. MOUNT 모드에서만 사용할 수 있다. 자세한 내용은 "Tibero 관리자 안내서"를 참고한다.

– rename_clauses

구성요소	설명
RENAME TO DBNAME	데이터베이스 이름을 변경한다. MOUNT 모드에서만 사용할 수 있다.

– flashback_database_clauses

구성요소	설명
FLASHBACK LOGGING ON	플래시백 로깅을 활성화한다. MOUNT 모드에서 FLASHBACK_LOG_BUFFER 초기화 파라미터가 설정되어있어야 수행 가능하다. 플래시백 로그는 데이터베이스 복구 또는 NORMAL 모드로 부팅하면서부터 플래시백 로그 파일에 기록된다.
FLASHBACK LOGGING OFF	플래시백 로깅을 비활성화한다. MOUNT 모드에서만 수행 가능하다. 플래시백 로그는 더 이상 기록되지 않는다.
FLASHBACK ADD MARKER	플래시백 로그를 남기는 기준인 MARKER를 추가한다. MARKER의 시점 이후에 처음 변경되는 데이터 파일의 블록들만 플래시백 로그에 기록된다.
FLASHBACK CLEAR MARKER	Marker의 기록들을 모두 삭제한다.
FLASHBACK TO TSN flashback_target_tsn	TSN 기준 원하는 시점으로 데이터베이스의 컨트롤 파일과 데이터 파일을 플래시백한다.
FLASHBACK TO BEFORE RESETLOGS	현 데이터베이스의 Resetlogs 부팅 직전의 시점으로 데이터베이스의 컨트롤 파일과 데이터 파일을 플래시백한다.

- 예제
  - recovery_clauses

    다음은 복구를 시작하고 로그 파일을 하나씩 적용해 완전 복구를 진행하는 예이다.

```
SQL> ALTER DATABASE RECOVER;
Database altered.

SQL> ALTER DATABASE RECOVER LOGFILE
       '/database/archive/arc-d1141964974-s3-r0.arc';
Database altered.

SQL> ALTER DATABASE RECOVER LOGFILE
       '/database/archive/arc-d1141964974-s4-r0.arc';
Database altered.

SQL> ALTER DATABASE RECOVER LOGFILE '/database/logfile002.log';
Database altered.

SQL> ALTER DATABASE RECOVER CANCEL;
Database altered.
```

    위의 예를 보면, 마지막 부분에서 **CANCEL**을 사용해 복구를 종료하는 것을 알 수 있다.

다음은 **AUTOMATIC** 옵션을 사용해 자동으로 완전 복구를 진행하는 예이다.

```
SQL> ALTER DATABASE RECOVER AUTOMATIC;
Database altered.
```

- full_database_recovery_clause

다음은 **UNTIL CANCEL**을 명시하여, 불완전 복구를 진행하는 예이다.

```
SQL> ALTER DATABASE RECOVER DATABASE UNTIL CANCEL;
Database altered.

SQL> ALTER DATABASE RECOVER LOGFILE
       '/database/archive/arc-d1141964974-s3-r0.arc';
Database altered.

SQL> ALTER DATABASE RECOVER LOGFILE
       '/database/archive/arc-d1141964974-s4-r0.arc';
Database altered.

SQL> ALTER DATABASE RECOVER LOGFILE '/database/logfile002.log';
Database altered.

SQL> ALTER DATABASE RECOVER CANCEL;
Database altered.
```

위의 예는 로그 파일 3개를 복구한 뒤에 CANCEL을 사용해 복구를 마치고 있다.

다음은 **UNTIL TIME**을 이용해 불완전 복구를 진행하는 예이다.

```
SQL> ALTER DATABASE RECOVER AUTOMATIC DATABASE
       UNTIL TIME '2006-11-15 13:59:00';
Database altered.
```

위의 예에서는 사용자가 2006년 11월 15일 14시경 실수로 테이블을 제거하였다고 가정한다. 사용자가 테이블을 제거하기 이전의 데이터베이스로 되돌리기 위해서는 불완전 복구를 해야 한다.

먼저 되돌리려는 시점 이전에 백업한 데이터 파일을 복사한 후 제거가 일어나기 전까지의 로그 파일을 적용하여 복구를 한다. 따라서 위의 예에서와 같이 2006년 11월 15일 13시 59분까지 자동복구를 하면 된다.

다음은 **UNTIL CHANGE**를 이용하여 데이터베이스의 TSN 값을 기준으로 복구를 진행하는 예이다.

```
SELECT * FROM v$log;

GROUP# SEQUENCE#      BYTES MEMBERS ARC STATUS   FIRST_CHAN  FIRST_TIME
------ --------- ---------- ------- --- -------- ----------  ----------
```

```
     0         3 2147483648       2 NO  CURRENT        3250  2006-11-15
     1         1 2147483648       2 NO  INACTIVE       1319  2006-11-15
     2         2 2147483648       2 NO  INACTIVE       2074  2006-11-15

3 rows selected.

SQL> ALTER DATABASE RECOVER AUTOMATIC DATABASE UNTIL CHANGE 2074;

Database altered.
```

위의 예에서는 먼저 V$LOG를 입력하여 로그를 조회하고, '로그 그룹 2'가 시작하는 지점까지 복구를 진행하기 위해 TSN 값 **'2074'**를 입력하여 복구를 진행하는 것을 볼 수 있다. V$LOG를 사용하면 TSN 값을 조회하거나 문제가 생긴 로그를 알아낼 수 있다.

- dbfile_clauses

다음은 **CREATE DATAFILE**을 사용하는 예이다.

```
SQL> SELECT file#, error FROM v$recover_file;

     FILE# ERROR
---------- ------------------------
         2 FILE MISSING

1 row selected.

SQL> ALTER DATABASE CREATE DATAFILE 2;
Database altered.
```

위의 예는 FILE# 2번 데이터 파일이 존재하지 않을 경우 빈 파일을 만들어 복구할 수 있도록 준비한다.

다음은 **OFFLINE FOR DROP**을 사용해 데이터 파일에 속한 테이블 스페이스를 제거하는 예이다.

```
SQL> SELECT file#, error FROM v$recover_file;

     FILE# ERROR
---------- ------------------------
         2 FILE MISSING

1 row selected.

SQL> ALTER DATABASE DATAFILE 2 OFFLINE FOR DROP;
Database altered.
```

OFFLINE FOR DROP은 데이터 파일을 복구할 수 없을 때 해당 데이터 파일이 속한 테이블 스페이스를 제거하는 조건으로 데이터베이스를 운영할 때 사용한다.

다음은 **RESIZE**를 사용해 파일의 크기를 증가시키는 예이다.

```
SQL> SELECT file_name, blocks, autoextensible FROM
       dba_data_files WHERE file_name LIKE '%ts1.dbf';

FILE_NAME                BLOCKS AUTOEXTENSIBLE
-------------------- ---------- ---------------
/tmp/ts1.dbf               2576 NO

1 row selected.

SQL> ALTER DATABASE DATAFILE '/tmp/ts1.dbf' RESIZE 20M;
Database altered.

SQL> SELECT file_name, blocks, autoextensible FROM
       dba_data_files WHERE file_name LIKE '%ts1.dbf';

FILE_NAME                BLOCKS AUTOEXTENSIBLE
-------------------- ---------- ---------------
/tmp/ts1.dbf               5120 NO

1 row selected.
```

다음은 **autoextend_clause**를 명시하여 AUTOEXTEND 속성을 활성화시키는 예이다.

```
SQL> SELECT file_name, blocks, autoextensible
     FROM dba_data_files
     WHERE file_name LIKE '%ts1.dbf';

FILE_NAME                BLOCKS AUTOEXTENSIBLE
-------------------- ---------- ---------------
/tmp/ts1.dbf               5120 NO

1 row selected.

SQL> ALTER DATABASE DATAFILE '/tmp/ts1.dbf' AUTOEXTEND ON NEXT 5M;
Database altered.

SQL> SELECT file_name, blocks, autoextensible, increment_by
     FROM dba_data_files
     WHERE file_name LIKE '%ts1.dbf';

FILE_NAME                BLOCKS AUTOEXTENSIBLE  INCREMENT_
-------------------- ---------- --------------- ----------
/tmp/ts1.dbf               5120 YES                   1296
```

```
1 row selected.
```

- logfile_clauses

다음은 데이터베이스를 **ARCHIVELOG** 모드로 설정하는 예이다.

```
SQL> ALTER DATABASE ARCHIVELOG;
```

다음은 **ADD LOGFILE**을 사용해 온라인 로그 파일을 추가하는 예이다.

```
SQL> SELECT group#, member FROM v$logfile;

    GROUP#  MEMBER
----------  -----------------------------------
         0  /database/log001.log
         0  /database/log002.log
         1  /database/log003.log
         1  /database/log004.log
         2  /database/log005.log
         2  /database/log006.log

6 rows selected.

SQL> ALTER DATABASE ADD LOGFILE GROUP 3
     2 ('/database/log010.log', '/database/log011.log');
Database altered.

SQL> SELECT group#, member FROM v$logfile;

    GROUP#  MEMBER
----------  -----------------------------------
         0  /database/log001.log
         0  /database/log002.log
         1  /database/log003.log
         1  /database/log004.log
         2  /database/log005.log
         2  /database/log006.log
         3  /database/log010.log
         3  /database/log011.log

8 rows selected.
```

위의 예에서는 두 온라인 로그 파일, '/database/log010.log', '/database/log011.log'을 멤버로 갖는 로그 그룹을 추가하고 있다. 온라인 로그 파일을 추가할 때는 그룹 전체를 추가하거나 그룹에 로그 멤버를 추가할 수 있으며, 로그 파일의 절대경로를 명시해야 한다.

다음은 로그 그룹에 로그 멤버 하나를 추가하는 예이다.

```
SQL> SELECT * FROM v$log;

GROUP# SEQUENCE#      BYTES MEMBERS ARC STATUS   FIRST_CHAN  FIRST_TIME
------ --------- ---------- ------- --- -------- ----------  ----------
     0         3 2147483648       2 NO  CURRENT        3260  2006/12/20
     1         1 2147483648       2 NO  INACTIVE       1325  2006/12/20
     2         2 2147483648       2 NO  INACTIVE       2065  2006/12/20
     3        -1 2147483648       3 NO  UNUSED            0  2006/12/20

4 rows selected.

SQL> SELECT group#, member FROM v$logfile;

    GROUP#  MEMBER
----------  -----------------------------------
         0  /database/log001.log
         0  /database/log002.log
         1  /database/log003.log
         1  /database/log004.log
         2  /database/log005.log
         2  /database/log006.log
         3  /database/log010.log
         3  /database/log011.log

8 rows selected.

SQL> ALTER DATABASE ADD LOGFILE MEMBER '/tmp/log012.log'
       TO GROUP 3;
Database altered.

SQL> SELECT group#, member FROM v$logfile;

    GROUP#  MEMBER
----------  -----------------------------------
         0  /database/log001.log
         0  /database/log002.log
         1  /database/log003.log
         1  /database/log004.log
         2  /database/log005.log
         2  /database/log006.log
         3  /database/log010.log
         3  /database/log011.log
         3  /database/log012.log
```

```
9 rows selected.
```

위의 예는 '로그 그룹 3'에 로그 파일을 추가하는 예이다. V$LOG를 사용해 현재 로그 그룹을 확인할 수 있다. 또한 로그 전환(Log Switch)를 활용해 현재 로그 그룹을 변경할 수도 있다.

위의 예에서는 V$LOG를 통해 로그 그룹이 0번인 것을 알 수 있다. '로그 그룹 3'에 로그 멤버의 추가가 가능하다는 것을 알 수 있으며, '로그 그룹 3'에 로그 멤버 'log12'를 추가하고 그 결과를 보여주고 있다.

다음은 **DROP LOGFILE**을 사용해 온라인 로그 파일을 제거하는 예이다.

```
SQL> SELECT group#, member FROM v$logfile;

    GROUP#  MEMBER
----------  -----------------------------------
         0  /database/log001.log
         0  /database/log002.log
         1  /database/log003.log
         1  /database/log004.log
         2  /database/log005.log
         2  /database/log006.log
         3  /database/log010.log
         3  /database/log011.log
         3  /database/log012.log

9 rows selected.

SQL> ALTER DATABASE DROP LOGFILE MEMBER '/database/log012.log';
Database altered.

SQL> SELECT group#, member FROM v$logfile;

    GROUP#  MEMBER
----------  -----------------------------------
         0  /database/log001.log
         0  /database/log002.log
         1  /database/log003.log
         1  /database/log004.log
         2  /database/log005.log
         2  /database/log006.log
         3  /database/log010.log
         3  /database/log011.log

8 rows selected.

SQL> ALTER DATABASE DROP LOGFILE GROUP 3;
```

```
Database altered.

SQL> SELECT group#, member FROM v$logfile;

    GROUP#  MEMBER
----------  -----------------------------------
         0  /database/log001.log
         0  /database/log002.log
         1  /database/log003.log
         1  /database/log004.log
         2  /database/log005.log
         2  /database/log006.log

6 rows selected.
```

위의 예에서는 '로그 그룹 3'의 로그 멤버인 'log012.log'만 제거하는 경우와 '로그 그룹 3' 전체를 제거하는 경우를 보여주고 있다.

- control_file_clauses

  다음은 **control_file_clauses**를 사용해 컨트롤 파일의 생성 문장을 백업하는 예이다.

```
SQL> ALTER DATABASE BACKUP CONTROLFILE TO TRACE AS
     '/backup/create_ctr.sql';
```

  위의 예는 컨트롤 파일의 생성 문장을 '/backup/create_ctr.sql'이라는 파일에 저장한다.

  다음은 **RESETLOGS**로 지정한 상태에서 컨트롤 파일의 생성 문장을 백업하는 예이다.

```
SQL> ALTER DATABASE BACKUP CONTROLFILE TO TRACE AS
     '/backup/create_ctr.sql' REUSE RESETLOGS;
```

  위의 예에서는 이미 생성된 파일에 REUSE 옵션을 사용해서 컨트롤 파일의 생성 문장을 백업하는 것을 보여준다. REUSE 옵션은 이미 존재하는 파일에 컨트롤 파일의 생성 문장을 저장하려고 할 때 사용한다. 만약 RESETLOGS를 지정하지 않으면, NORESETLOGS로 간주된다.

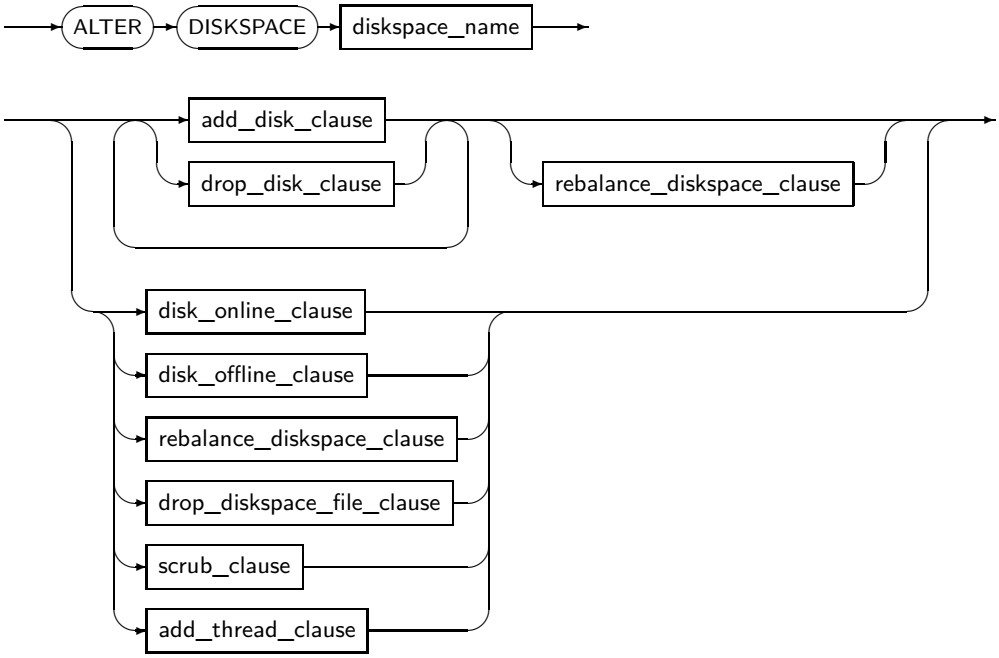
7.3. ALTER DISKSPACE

디스크스페이스와 디스크스페이스에 속한 디스크의 속성과 상태를 변경한다.

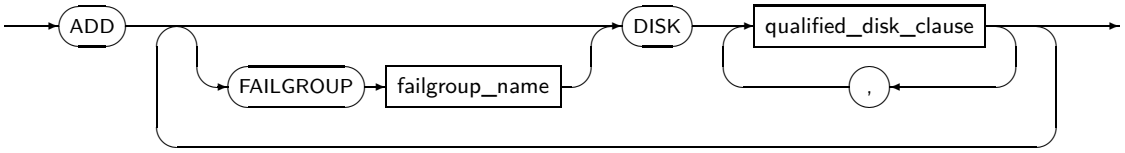
ALTER DISKSPACE의 세부 내용은 다음과 같다.

- 문법

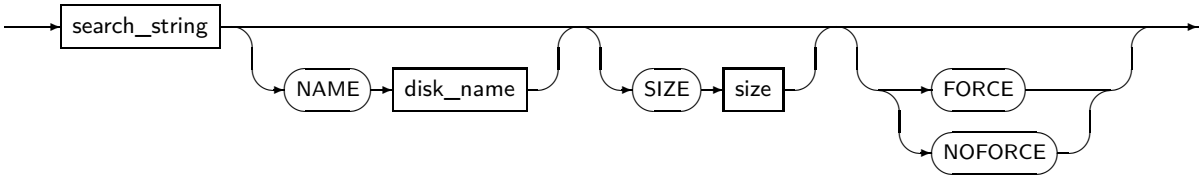
alter_diskspace



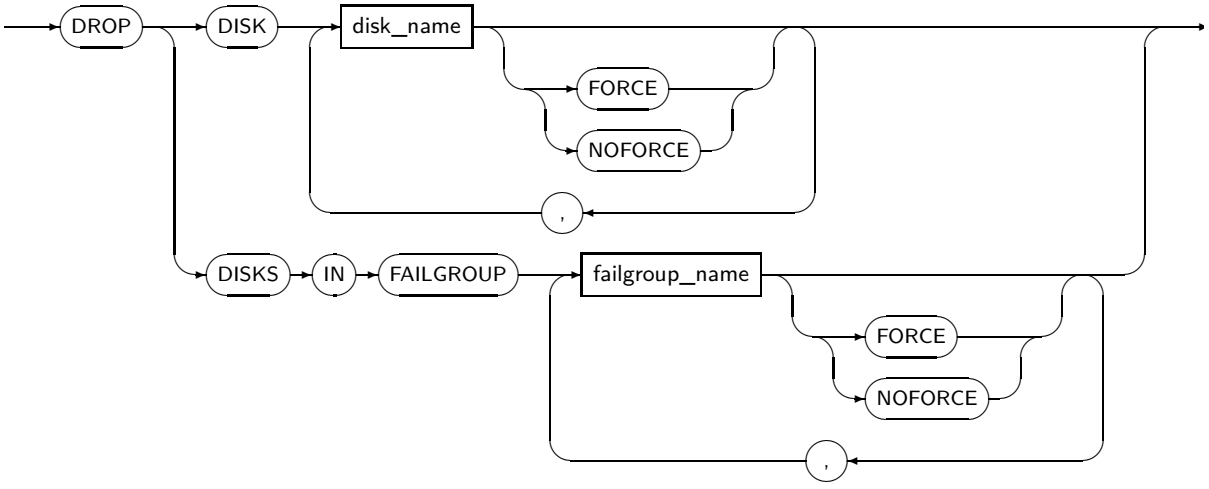
add_disk_clause



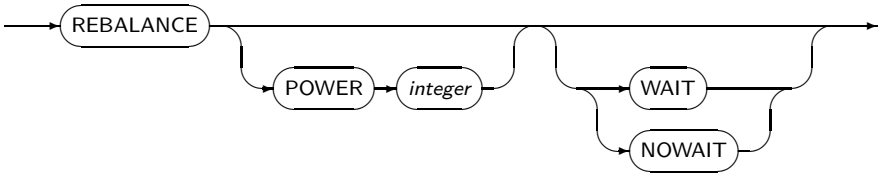
qualified_disk_clause



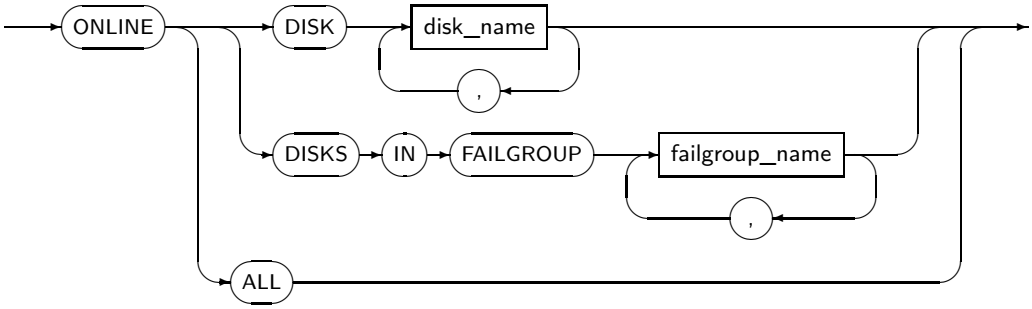
drop_disk_clause

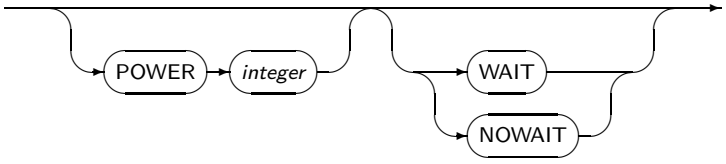


rebalance_diskspace_clause

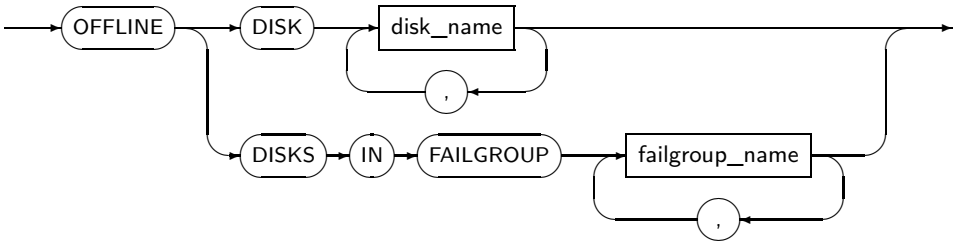


disk_online_clause

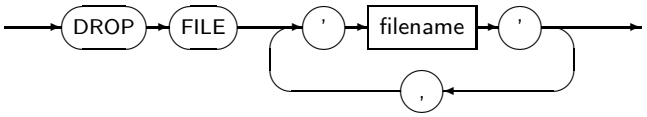




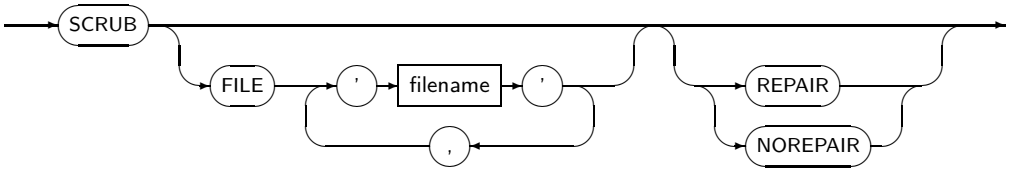
disk_offline_clause

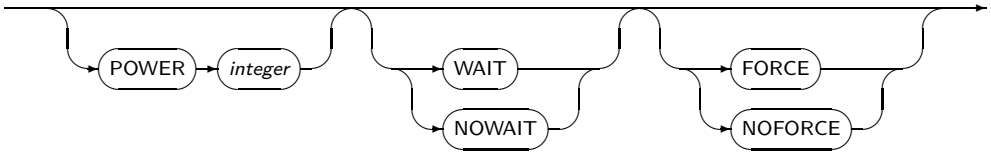


drop_diskspace_file_clause

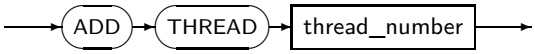


scrub_clause





add_thread_clause



- 구성요소
  - create_diskspace

구성요소	설명
diskspace_name	변경할 디스크스페이스의 이름을 명시한다.
add_disk_clause	디스크스페이스에 디스크 추가를 선언할 때 사용한다. 리밸런스를 수행하기 전까지 디스크가 사용되지 않는다.
drop_disk_clause	디스크스페이스에서 디스크 제거를 선언할 때 사용한다. 리밸런스를 수행하기 전까지 디스크가 제거되지 않는다.
rebal ance_diskspace_clause	디스크스페이스에 저장된 데이터를 이동시켜서 실제 디스크의 추가 또는 제거를 수행한다.
disk_online_clause	오프라인 상태인 디스크를 온라인 상태로 바꿀 때 사용한다. 오프라인 상태였던 디스크는 데이터 동기화 과정을 거쳐 온라인 상태가 된다.
disk_offline_clause	온라인 상태인 디스크를 오프라인 상태로 바꿀 때 사용한다. 오프라인 상태로 바뀐 디스크는 그 즉시 읽기와 쓰기가 중단된다.
drop_diskspace_file_clause	디스크스페이스에 저장된 사용자 파일을 제거할 때 사용한다.
scrub_clause	디스크스페이스에 저장된 사용자 데이터를 클리닝할 때 사용한다. NORMAL 또는 HIGH의 중복 레벨로 생성된 디스크스페이스에 대해, 자동으로 데이터 커럽션을 확인하고 복구할 수 있다.
add_thread_clause	디스크스페이스에 클러스터 구성을 위한 스레드를 추가할 때 사용한다.

  - add_disk_clause

구성요소	설명
FAILGROUP fail group_name	디스크가 속할 failgroup 이름을 명시한다. 디스크스페이스의 중복 레벨이 NORMAL 또는 HIGH인 경우에만 유효하며, 알파벳과 숫자 등으로 최대 48자를 사용할 수 있다. 명시하지 않은 경우에는 모든 디스크가 각각 하나의 failgroup으로 구성되며 디스크 이름이 failgroup 이름으로 사용된다.
DISK quali fied_disk_clause	디스크스페이스에 추가할 디스크를 정의한다.

– qualified_disk_clause

구성요소	설명
search_string	디스크스페이스를 구성할 디스크의 경로를 명시한다. 와일드카드를 사용해 여러 디스크를 나타낼 수 있으며, TAS_DISKSTRING 초기화 파라미터로 찾을 수 있는 경로이어야 한다.
NAME disk_name	search_string으로 찾은 디스크의 이름을 명시한다. search_string으로 찾은 디스크가 한 개인 경우에만 지정할 수 있으며, 알파벳과 숫자 등으로 최대 48자를 사용할 수 있다. 디스크의 이름은 TAS 내부에서만 사용되며 디스크 경로와는 무관하다. 명시하지 않은 경우 임의로 생성된 이름이 사용된다.
SIZE	search_string으로 찾은 디스크의 크기를 byte 단위로 명시한다. search_string으로 찾은 디스크가 여러 개인 경우 모든 디스크가 같은 크기로 지정된다. 명시하지 않은 경우 TAS에서 파악한 실제 디스크의 크기가 지정된다. TAS에서 실제 디스크를 크기를 알 수 없는 경우 오류가 발생하며, 이 때는 반드시 디스크 크기를 명시해 주어야 한다.
FORCE	search_string으로 찾은 디스크가 이미 다른 디스크스페이스에 사용 중이더라도 이를 무시하고 새로운 디스크스페이스 구성에 사용한다. 기존 디스크스페이스를 파기하는 경우에 명시한다.
NOFORCE	search_string으로 찾은 디스크가 이미 다른 디스크스페이스에 사용 중인 경우 오류가 발생한다. FORCE 또는 NOFORCE가 명시되지 않은 경우의 기본 동작이다.

– drop_disk_clause

구성요소	설명
DISK disk_name	제거할 디스크 이름을 명시한다.
FORCE	리밸런스를 기다리지 않고 디스크스페이스에서 즉시 디스크를 제거한다. 이후 리밸런스는 중복 저장된 다른 복제본으로 수행된다. 디스크 장애로 인해 읽기가 불가능한 디스크를 제거할 때 사용한다.
NOFORCE	리밸런스 완료 후 디스크가 완전히 제거된다. FORCE/NOFORCE를 명시하지 않은 경우의 기본 동작이다.
DISKS IN FAILGROUP failgroup_name	지정된 failgroup에 속한 모든 디스크를 제거할 때 사용한다.

– rebalance_diskspace_clause

구성요소	설명
POWER integer	리밸런스의 수행 속도를 0에서 11 사이의 값으로 지정한다. 지정된 숫자가 클수록 리밸런스가 빨리 끝나지만, 저장된 데이터의 동시성이 떨어진다. 명시하지 않은 경우의 자동으로 결정된다.
WAIT	리밸런스가 완전히 끝날 때까지 대기한다.
NOWAIT	리밸런스를 시작하고 대기하지 않는다. 명령이 성공했더라도 리밸런스가 진행 중일 수 있다.

– disk_online_clause

구성요소	설명
DISK disk_name	온라인 상태로 변경할 디스크의 이름을 명시한다.
DISKS IN FAILGROUP failgroup_name	지정된 failgroup에 속한 모든 디스크를 온라인 상태로 변경할 때 사용한다.
ALL	오프라인 상태인 모든 디스크를 온라인 상태로 변경할 때 사용한다.
POWER integer	오프라인 상태였던 디스크에 대해 데이터 동기화 속도를 1에서 11 사이의 값으로 지정한다. 지정된 숫자가 클수록 동기화가 빨리 끝나지만, 저장된 데이터의 동시성이 떨어진다.
WAIT	동기화가 완전히 끝날 때까지 대기한다.
NOWAIT	동기화를 시작하고 대기하지 않는다. WAIT를 명시하지 않은 경우의 기본 동작이다. 명령이 성공했더라도 동기화가 진행 중일 수 있다.

– disk_offline_clause

구성요소	설명
DISK disk_name	오프라인 상태로 변경할 디스크의 이름을 명시한다.
DISKS IN FAILGROUP failgroup_name	지정된 failgroup에 속한 모든 디스크를 오프라인 상태로 변경할 때 사용한다.

– drop_diskspace_file_clause

구성요소	설명
FILE 'filename'	삭제할 파일의 이름을 명시한다.

– scrub_clause

구성요소	설명
FILE 'filename'	클리닝을 수행할 파일의 이름을 명시한다. 명시하지 않으면 디스크스페이스의 모든 파일에 대해 클리닝을 수행한다.

구성요소	설명
REPAIR	복제본과 같지 않은 데이터가 발견될 경우 데이터를 동기화한다.
NOREPAIR	복제본과 같지 않은 데이터가 발견되면 Dump만 출력하고 데이터를 동기화하지 않는다. REPAIR를 명시하지 않은 경우의 기본 동작이다. TRACE_DUMP_DEST 초기화 파라미터에 지정된 경로에 Dump가 출력된다.
POWER integer	데이터 클리닝 속도를 1에서 11 사이의 값으로 지정한다. 지정된 숫자가 클수록 클리닝이 빨리 끝나지만, 저장된 데이터의 동시성이 떨어진다.
WAIT	클리닝이 완전히 끝날 때까지 대기한다.
NOWAIT	클리닝을 시작하고 대기하지 않는다. WAIT를 명시하지 않은 경우의 기본 동작이다. 명령이 성공했더라도 클리닝이 진행 중일 수 있다.
FORCE	전체 시스템의 IO 부하에 관계 없이 클리닝을 수행한다.
NOFORCE	전체 시스템의 IO 부하가 큰 경우 클리닝을 수행하지 않는다. FORCE를 명시하지 않은 경우의 기본 동작이다.

– add_thread_clause

구성요소	설명
thread_number	추가할 Thread 번호를 명시한다. Thread는 TAS 인스턴스를 클러스터로 구성할 때 사용되며, THREAD 초기화 파라미터로 지정해 사용한다.

- 예제

  다음은 **ALTER DISKSPACE**를 사용해 여러 개의 디스크를 추가, 제거하고 리밸런스를 수행하는 예이다.

```
ALTER DISKSPACE ds
    ADD DISK '/tas/dev/path30' NAME disk30 SIZE 512G,
             '/tas/dev/path31' NAME disk31 SIZE 256G;
ALTER DISKSPACE ds
    DROP DISK disk20, disk21;
ALTER DISKSPACE ds
    REBALANCE POWER 8 WAIT;
```

  다음은 앞의 디스크 추가, 제거 작업을 하나의 문장으로 수행하는 예이다.

```
ALTER DISKSPACE ds
    ADD DISK '/tas/dev/path30' NAME disk30 SIZE 512G,
             '/tas/dev/path31' NAME disk31 SIZE 256G
    DROP DISK disk20, disk21
    REBALANCE POWER 8 WAIT;
```

  다음은 **ALTER DISKSPACE**를 사용해 디스크를 온라인 상태로 변경하는 예이다.

```
ALTER DISKSPACE ds
    ONLINE DISK disk00, disk01
    POWER 5 NOWAIT;
```

다음은 **ALTER DISKSPACE**를 사용해 디스크를 오프라인 상태로 변경하는 예이다.

```
ALTER DISKSPACE ds
    OFFLINE DISKS IN FAILGROUP fg1, fg2;
```

다음은 **ALTER DISKSPACE**를 사용해 디스크스페이스의 사용자 파일을 삭제하는 예이다.

```
ALTER DISKSPACE ds
    DROP FILE '+ds/data00.dtf',
              '+ds/data01.dtf';
```

다음은 **ALTER DISKSPACE**를 사용해 스레드를 추가하는 예이다.

```
ALTER DISKSPACE ds
    ADD THREAD 1;
```

7.4. ALTER FUNCTION

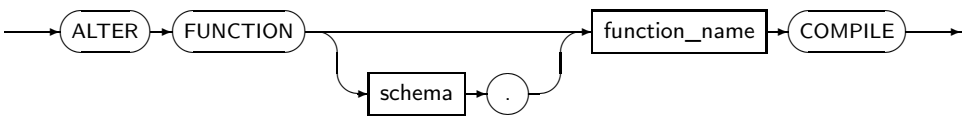
지정된 tbPSM 함수를 재컴파일한다. ALTER PROCEDURE 문과 유사하다. 일반적으로 SQL 문장에 포함된 tbPSM 함수가 유효하지 않으면, SQL 문장을 실행할 때 자동으로 재컴파일된다. tbPSM 함수 또는 프러시저의 유효성에 대해서는 "Tibero tbPSM 안내서"를 참고한다.

ALTER FUNCTION은 객체의 유효성 여부에 상관없이 재컴파일을 시도한다. 이 과정에서 직간접적으로 종속 관계를 가지는 모든 부모 객체를 재귀적으로 검사하여 유효하지 않을 경우 재컴파일을 시도한다. 또한 직간접적으로 종속된 모든 자식 객체를 무효화한다. 자세한 내용은 “7.9. ALTER PROCEDURE”를 참고한다.

ALTER FUNCTION의 세부 내용은 다음과 같다.

- 문법

alter_function



- 특권

  tbPSM 함수를 재컴파일하는 사용자가 대상 함수를 소유하고 있거나, ALTER ANY PROCEDURE 시스템 특권을 부여 받아야 한다.

- 구성요소

구성요소	설명
schema	해당 함수가 속해 있는 스키마의 이름을 명시한다.
function_name	해당 함수의 이름을 명시한다.

- 예제

  다음은 **ALTER FUNCTION**을 사용해 tbPSM 함수를 컴파일하는 예이다.

```
ALTER FUNCTION tibero.get_square COMPILE;
```

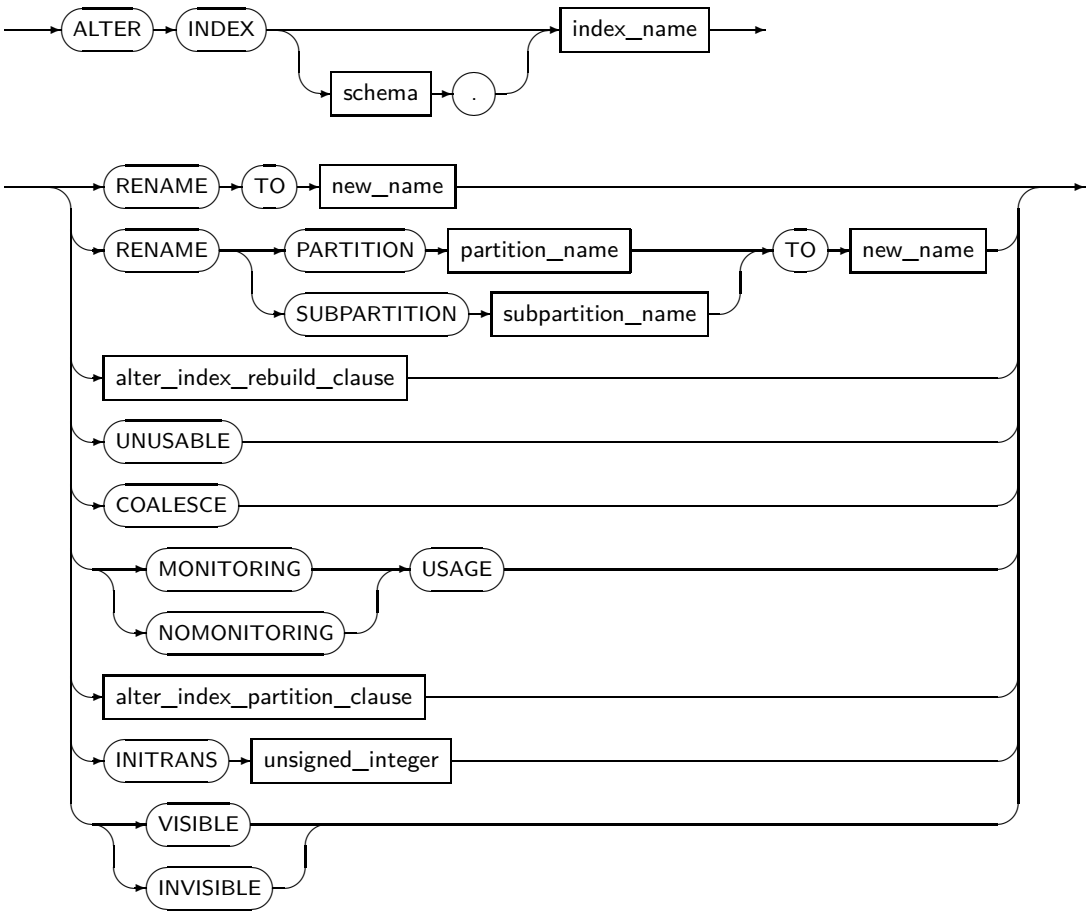
7.5. ALTER INDEX

인덱스의 속성을 변경한다. 인덱스를 다시 만들거나, 인덱스의 이름을 변경할 때도 사용된다.

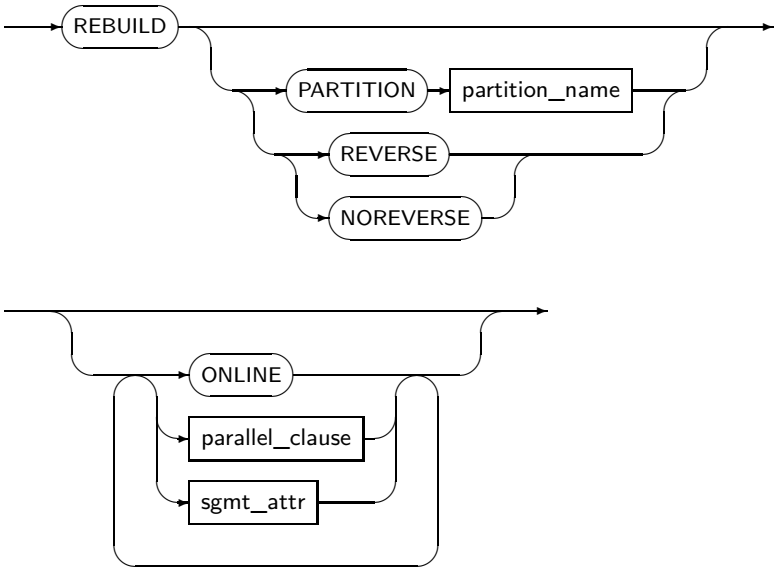
ALTER INDEX의 세부 내용은 다음과 같다.

- 문법

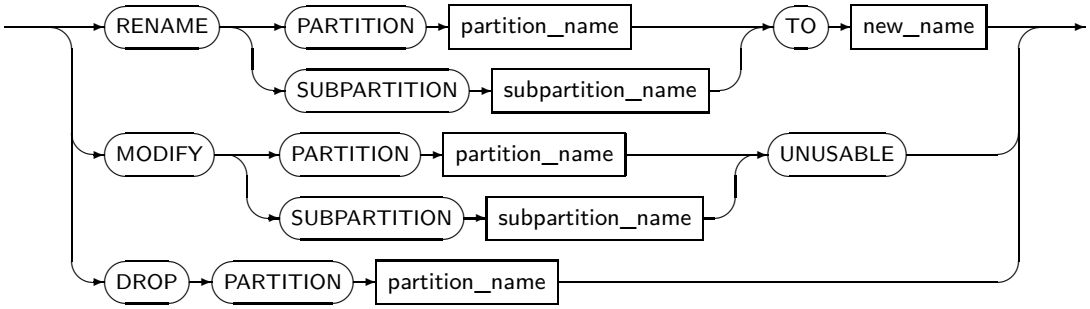
alter_index



alter_index_rebuild_clause



alter_index_partition_clause



- 특권

  다음 중 하나를 만족해야 ALTER INDEX 문을 실행할 수 있다.

  – 인덱스가 ALTER INDEX 문을 실행하는 사용자의 스키마에 포함되어 있다.

  – ALTER ANY INDEX 시스템 특권이 있다.

- 구성요소

구성요소	설명
schema	인덱스나 테이블의 스키마의 이름이다. 생략하면 현재 사용자의 스키마로 인식된다.
index_name	변경할 인덱스의 이름을 명시한다.
RENAME	인덱스를 다시 만들지 않고, 이름을 변경한다. 인덱스의 파티션(PARTITION) 또는 서브 파티션(SUBPARTITION)의 이름도 변경할 수 있다.
new_name	인덱스의 새로운 이름이다. 스키마의 이름을 지정할 수 없으므로, 스키마를 이동할 수 없다.
PARTITION	파티션의 이름을 변경하고자 할 때 사용한다.

구성요소	설명
partition_name	변경할 파티션의 이름을 명시한다.
SUBPARTITION	서브 파티션의 이름을 변경하고자 할 때 사용한다.
subpartition_name	변경할 서브 파티션의 이름을 명시한다.
REBUILD	인덱스를 다시 생성한다. 인덱스 효율이 좋지 않거나, 비활성화되어 있던 인덱스를 다시 사용하고자 할 때 사용한다. 인덱스 생성이 완료되면 비활성화되어 있던 인덱스는 다시 활성화된다.
REVERSE	REBUILD의 옵션이다. 인덱스 블록의 바이트 순서를 순차에서 역차순으로 다시 배정한다. 지정하지 않으면 인덱스의 기존 바이트 순서와 동일하게 재생성된다.
NOREVERSE	REBUILD의 옵션이다. 인덱스 블록의 바이트 순서를 기존 순서와 동일하게 재생성한다.
ONLINE	REBUILD의 옵션이다. 인덱스를 재생성하는 동안 해당 테이블에 DML을 수행할 수 있도록 한다.
COALESCE	인덱스의 비어 있는 블록을 재사용하기 위해 인덱스 블록의 내용을 모아서 저장한다.
MONITORING USAGE	인덱스 사용 여부를 모니터링한다. 모니터링 결과는 V$OBJECT_USAGE를 조회하여 확인할 수 있다.
NOMONITORING USAGE	인덱스 사용 여부 모니터링을 중지한다.
INITRANS unsigned_integer	인덱스 블록의 트랜잭션 엔트리(Transaction Entry)를 위한 공간을 최초 몇 개 확보할지 설정한다. 트랜잭션 엔트리는 필요할 때마다 확장된다. 따라서 미리 큰 값을 반드시 설정할 필요는 없다.
INVISIBLE	인덱스의 상태 옵션이다. 인덱스의 VISIBLE과 INVISIBLE 여부는 OPTIMIZER가 해당 인덱스를 고려하는지 여부와 관련이 있다. INVISIBLE 상태인 인덱스는 DML 과정에 일반 인덱스처럼 형태를 계속 유지한다. 하지만 Optimizer에서 고려할 대상으로 여기지 않는다.

- 제약 사항

  ALTER INDEX 문의 수행에는 다음과 같은 제약사항이 있다.

  – Functional Index에 대한 REBUILD ONLINE REVERSE는 수행할 수 없다.

  – REBUILD ONLINE 수행 도중 실패하거나 중단되었을 경우 반드시 DBMS_REPAIR 패키지의 ONLINE_INDEX_CLEAN 함수를 통해 인덱스 online-rebuild에 대한 클린업을 수행해야 한다.

- 예제

다음은 **RENAME**을 사용해 생성된 인덱스의 이름을 변경하는 예이다.

```
SQL> CONN u1/u1
Connected.

SQL> CREATE TABLE t (a NUMBER);
Table created.

SQL> CREATE INDEX i ON t (a);
Index created.

SQL> ALTER INDEX i RENAME TO j;
Index altered.

SQL> SELECT index_name FROM user_indexes;
 INDEX_NAME
 --------------------------
 J

1 row selected.

SQL> ALTER INDEX u1.j RENAME TO k;
Index altered.

SQL> SELECT index_name FROM user_indexes;
 INDEX_NAME
 --------------------------
 K

1 row selected.
```

다음은 **REBUILD**를 사용해 인덱스를 재생성하는 예이다.

```
SQL> ALTER INDEX i REBUILD;
Index altered.
```

다음은 **NOREVERSE** 옵션과 **ONLINE** 옵션을 지정하여 인덱스를 재생성하는 예이다.

```
SQL> ALTER INDEX i REBUILD NOREVERSE ONLINE;
Index altered.
```

다음은 **COALESCE**를 사용하는 예이다.

```
SQL> ALTER INDEX i COALESCE;
Index altered.
```

다음은 **MONITORING USAGE**를 사용하는 예이다.

```
SQL> ALTER INDEX i MONITORING USAGE;
Index altered.
```

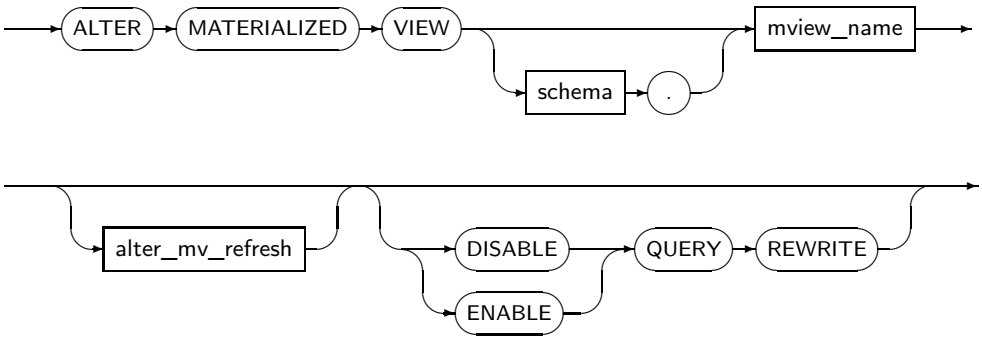
7.6. ALTER MATERIALIZED VIEW

이미 생성된 실체화 뷰를 변경한다.

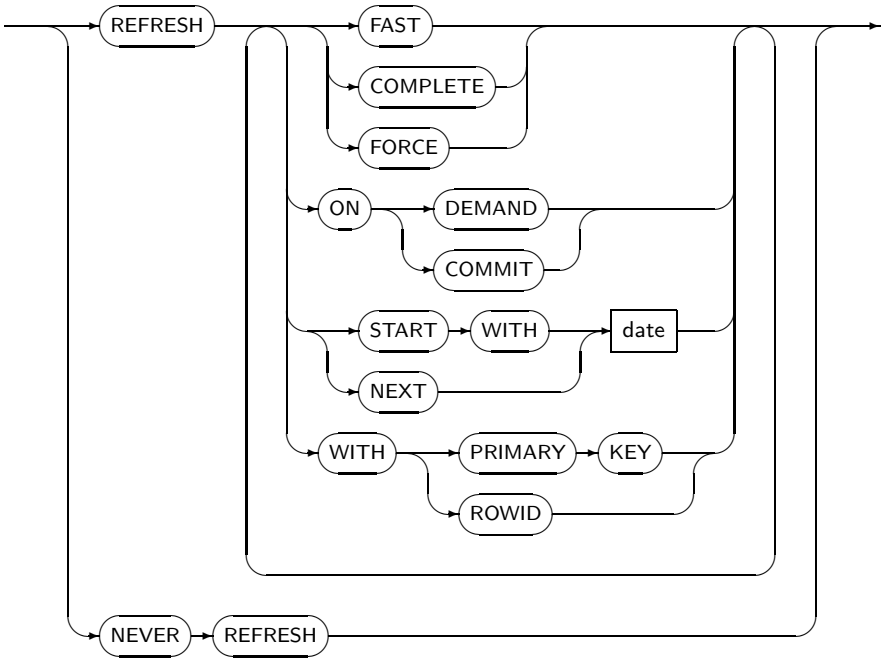
ALTER MATERIALIZED VIEW의 세부 내용은 다음과 같다.

- 문법

alter_materialized_view



alter_mv_refresh



- 특권

- 사용자가 소유한 스키마의 실체화 뷰를 변경하기 위해서는 별다른 특권이 필요하지 않다.
- 다른 사용자가 소유한 스키마의 실체화 뷰를 변경하기 위해서는 ALTER ANY MATERIALIZED 시스템 특권이 있어야 한다.

● 구성요소

- alter_materialized_view

구성요소	설명
alter_mv_refresh	실체화 뷰에 대한 리프레시(Refresh) 방법, 모드, 시간 등을 지정한다. 실체화 뷰가 참조하는 테이블에 변경이 발생하면, 테이블의 현재 데이터를 반영하기 위해 실체화 뷰를 리프레시한다. 이 문장을 이용하여 데이터베이스가 실체화 뷰를 리프레시하는 스케줄, 모드, 방법 등을 조절할 수 있다.
ENABLE	실체화 뷰를 질의 다시 쓰기에 사용될 수 있는 상태로 설정한다. (기본값)
DISABLE	실체화 뷰를 질의 다시 쓰기에 사용될 수 없는 상태로 설정한다.
QUERY REWRITE	실체화 뷰가 질의 다시 쓰기에 사용될 지의 여부를 설정한다.

- alter_mv_refresh

구성요소	설명
FAST	빠른 리프레시를 수행하려고 할 때 사용한다.
COMPLETE	실체화 뷰를 정의하는 질의를 재수행하여 완전 리프레시를 사용한다. COMPLETE가 명시되면 빠른 리프레시가 가능하더라도 완전 리프레시를 사용한다. (기본값)
FORCE	빠른 리프레시가 가능하면 빠른 리프레시를 수행하고, 그렇지 않으면 완전 리프레시를 수행한다.
ON DEMAND	사용자가 DBMS_MVIEW 패키지의 REFRESH 프러시저를 호출하는 경우에만 리프레시를 수행한다. (기본값)
ON COMMIT	ON COMMIT이 명시된 경우 마스터 테이블에 커밋이 일어날 때마다 리프레시를 수행한다. 하지만, START WITH와 NEXT는 명시할 수 없다. ON COMMIT과 ON DEMAND는 동시에 명시할 수 없다.
START WITH	처음으로 자동 리프레시가 시작될 날짜형 표현식을 명시한다. START WITH는 미래의 시간을 나타내는 값이어야 한다. NEXT 없이 START WITH만을 명시할 경우 데이터베이스는 한 번만 리프레시를 수행한다.
NEXT	자동 리프레시의 간격을 계산하기 위한 날짜형 표현식을 명시한다.

구성요소	설명
	NEXT는 미래의 시간을 나타내는 값이어야 한다. START WITH 없이 NEXT만 명시한 경우 데이터베이스는 NEXT 표현식을 평가하여 첫 리프레시의 시간을 정한다.
date	START WITH와 NEXT에 지정할 날짜형 리터럴을 명시한다.
WITH PRIMARY KEY	PRIMARY KEY를 사용하여 리프레시를 수행한다.
WITH ROWID	ROWID를 사용하여 리프레시를 수행한다.
NEVER REFRESH	자동 리프레시를 하지 않는다.

- 예제

  다음은 **ALTER MATERIALIZED VIEW**를 사용해 실체화 뷰를 변경하는 예이다.

```
ALTER MATERIALIZED VIEW MV
  REFRESH START WITH SYSDATE NEXT SYSDATE + 10/1440;
```

  위의 예에서는 START WITH와 NEXT 절을 이용하여 기존의 실체화 뷰를 10분마다 자동으로 리프레시하는 실체화 뷰로 변경하였다. 단위가 날짜이기 때문에 분 단위로 설정하기 위해 1440(24X60 = 1440)으로 나누었다.

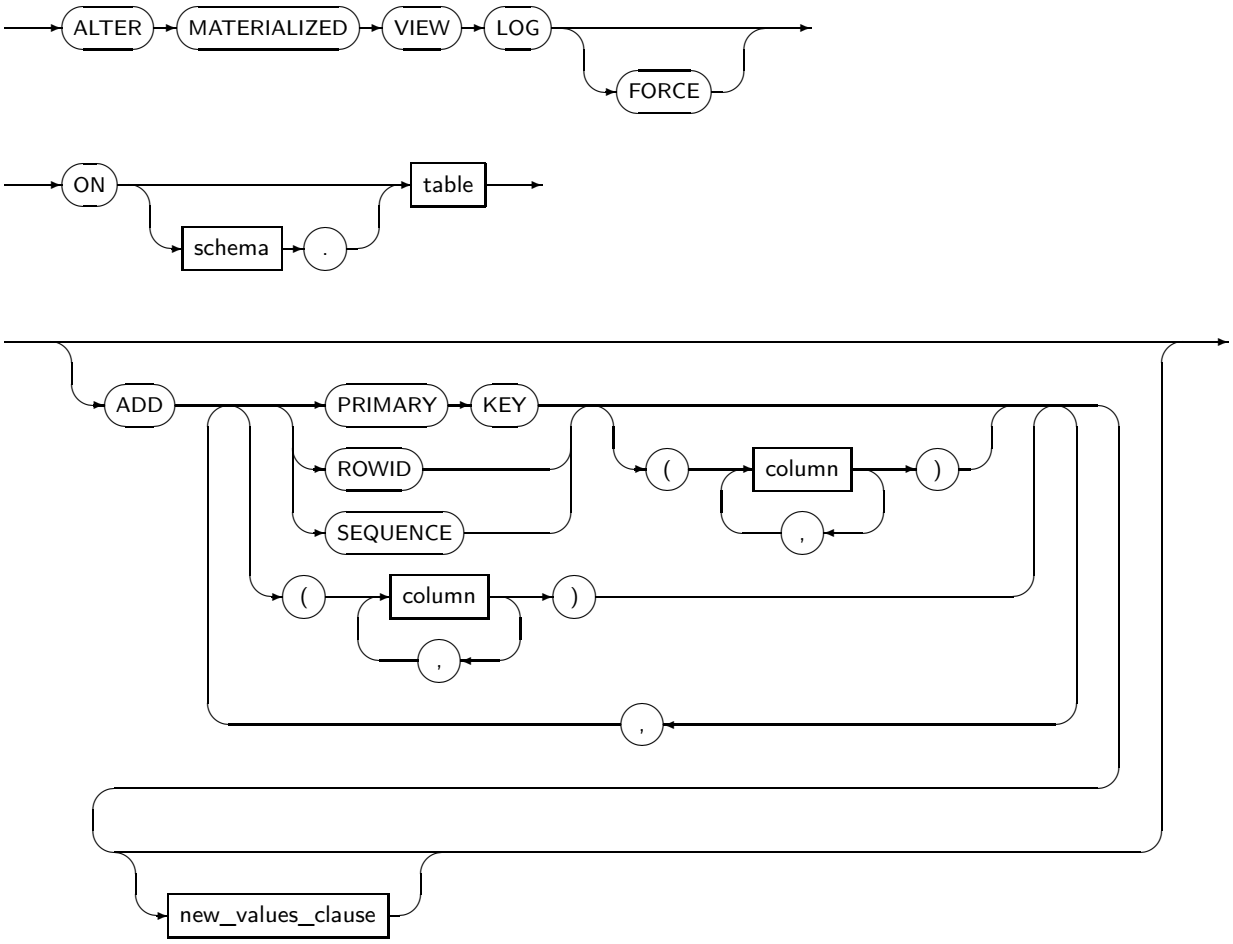
7.7. ALTER MATERIALIZED VIEW LOG

지정된 마스터 테이블의 실체화 뷰 로그를 변경한다.

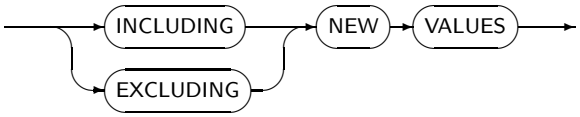
ALTER MATERIALIZED VIEW LOG의 세부 내용은 다음과 같다.

- 문법

alter_materialized_view_log



new_values_clause



- 특권

  다음 중 하나를 만족해야 ALTER MATERIALIZED VIEW LOG 문을 실행할 수 있다.

  – 마스터 테이블을 사용자가 소유하고 있어야 한다.

  – 마스터 테이블에 SELECT 권한과 ALTER 권한이 모두 있어야 한다.

- 구성요소

  – alter_materialized_view_log

구성요소	설명
FORCE	실체화 뷰 로그에 이미 존재하는 속성의 추가를 명시하여도 에러를 발생시키지 않고 존재하지 않는 속성만을 추가한다.

구성요소	설명
schema	변경할 실체화 뷰 로그에 마스터 테이블의 스키마를 명시한다. 생략하면 현재 사용자의 스키마로 인식된다.
table	변경할 실체화 뷰 로그의 마스터 테이블의 이름을 명시한다.
PRIMARY KEY	실체화 뷰 로그에 마스터 테이블의 변경된 로우의 PRIMARY KEY를 기록한다.
ROWID	실체화 뷰 로그에 마스터 테이블의 변경된 로우의 ROWID를 기록한다.
SEQUENCE	실체화 뷰 로그에 마스터 테이블의 변경된 로우의 SEQUENCE를 기록한다.
column	실체화 뷰 로그에 기록할 마스터 테이블의 컬럼을 지정한다.

– new_values_clause

구성요소	설명
INCLUDING NEW VALUES	실체화 뷰 로그에 컬럼의 변경 전/후의 값을 모두 기록한다.
EXCLUDING NEW VALUES	실체화 뷰 로그에 컬럼의 변경 전의 값만을 기록한다. (기본값)

- 예제

  다음은 **ALTER MATERIALIZED VIEW LOG**를 사용해 실체화 뷰 로그를 변경하는 예이다.

```
ALTER MATERIALIZED VIEW LOG FORCE
  ON DEPT ADD PRIMARY KEY, SEQUENCE (DNAME, LOC)
  INCLUDING NEW VALUES;
```

7.8. ALTER PACKAGE

ALTER PACKAGE를 사용해 명시적으로 PACKAGE의 SPECIFICATION과 BODY 또는 둘 다를 재컴파일할 수 있다. 명시적으로 재컴파일을 수행하면 런타임 때 발생할 수 있는 암묵적인 재컴파일을 막을 수 있다. 따라서 오버헤드와 컴파일 에러를 미연에 방지할 수가 있다.

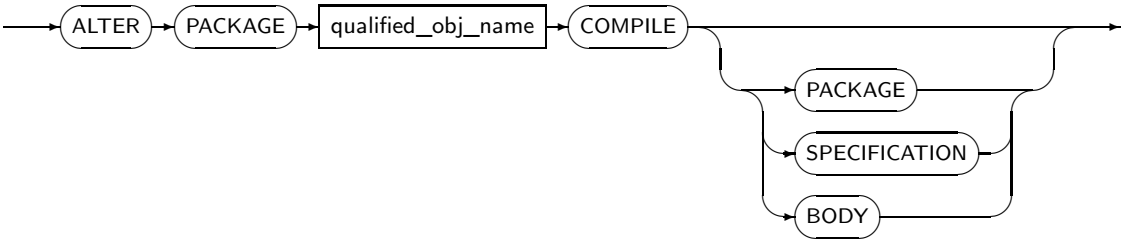
하나의 PACKAGE는 전체가 하나의 단위로 취급되기 때문에, ALTER PACKAGE 문은 PACKAGE 안에 포함된 프러시저나 함수 모두를 재컴파일한다. ALTER PROCEDURE나 ALTER FUNCTION을 사용해 PACKAGE 안에 포함된 PROCEDURE와 FUNCTION을 개별적으로 재컴파일하는 방법은 없다.

부모 객체에 대한 재컴파일과 자식 객체에 대한 무효화에 대한 자세한 내용은 “7.9. ALTER PROCEDURE”를 참고한다.

ALTER PACKAGE의 세부 내용은 다음과 같다.

- 문법

alter_package



- 특권

  PACKAGE가 자신의 스키마에 포함되어 있거나, ALTER ANY PROCEDURE 시스템 특권이 있어야 한다.

- 구성요소

구성요소	설명
qualified_obj_name	재컴파일할 PACKAGE의 이름을 명시한다.
PACKAGE	디폴트로 생략할 수 있으며, PACKAGE SPECIFICATION과 PACKAGE BODY가 모두 재컴파일된다. PACKAGE가 직간접적으로 종속 관계를 맺는 부모 객체를 모두 재귀적으로 검사하여 필요하면 재컴파일을 실행해 유효화 과정을 거친다. PACKAGE에 직간접적으로 종속되는 모든 자식 객체를 무효화한다(PACKAGE BODY는 다른 객체 그리고 명세에 종속된다).
SPECIFICATION	명세(PACKAGE SPECIFICATION)만 재컴파일된다. PACKAGE SPECIFICATION은 다른 객체에 종속되는 자식 객체가 될 수 없으므로, 오직 명세에 대한 컴파일만 일어난다. PACKAGE SPECIFICATION에 직간접적으로 종속되는 자식 객체를 모두 무효로 한다. PACKAGE BODY 또한 이런 자식 객체에 포함된다. 무효화된 PACKAGE BODY는 다음에 사용될 때 자동으로 재컴파일된다.
BODY	PACKAGE BODY이 직간접적으로 종속 관계를 가지는 모든 무효화된 부모 객체를 재컴파일한다. 명세가 무효화되어 있다면 재컴파일된다. PACKAGE 옵션과 다른 점은 BODY 옵션은 명세가 무효화 상태일 때만 재컴파일이 일어난다는 것이다. 본문에 직간접적으로 종속되는 모든 자식 객체를 무효화한다. BODY 옵션은 PACKAGE BODY를 다시 작성했거나, PACKAGE BODY만 무효화되었을 때 유용하다.

- 예제

  다음은 **ALTER PACKAGE**를 사용해 PACKAGE를 재컴파일하는 예이다.

```
ALTER PACKAGE tibero.emp_pkg COMPILE;
```

7.9. ALTER PROCEDURE

지정된 tbPSM 프러시저를 재컴파일한다. 일반적으로 SQL 문장에 포함된 tbPSM 프러시저가 유효하지 않으면, SQL 문장을 실행할 때 재컴파일된다.

프러시저의 재컴파일은 DDL에 준하는 작업이 수행되는 것이므로 수행하려고 했던 SQL의 처리 속도에 영향을 줄 수 있다. 또한, 자동으로 재컴파일을 하는 도중에 다른 무효화된 스키마 객체에 의해 컴파일의 수행이 실패할 수도 있다. 이럴 때 SQL 문장의 처리도 당연히 실패한다. 때문에 SQL 문장을 실행할 때 재컴파일이 되는 것을 원하지 않을 수 있다. 이럴 때 DBA나 사용자는 ALTER PROCEDURE 문을 미리 호출하여 무효화된 객체를 다시 유효하게 만들 수 있다.

ALTER PROCEDURE 문을 호출했을 때 재컴파일하는 단계는 다음과 같다.

단계	설명
1	프러시저가 종속성을 갖는 모든 부모 객체를 찾은 다음 상태가 무효화(INVALID)이면 재컴파일을 시도한다. 각각의 객체는 다시 상태가 유효화(VALID)되어 데이터 사전의 관련 정보가 갱신되고 커밋된다. 부모 객체를 재컴파일하는 과정에서 그 객체의 부모 객체는 각각 재귀적으로 컴파일된다. 이 중 하나라도 실패하면 ALTER PROCEDURE 문의 대상인 객체의 컴파일은 실패한다.
2	재컴파일된 모든 객체에 종속되는 자식 객체는 모두 상태가 무효화(INVALID)가 된다. 상태가 무효화된 객체는 다음번에 사용될 때 자동으로 재컴파일 되어 상태가 유효화(VALID)하게 변경될 수 있다.
3	ALTER PROCEDURE 문장의 대상 객체는 상태의 유효여부에 상관없이 재컴파일되며, 그 자식 객체는 재귀적으로 모두 상태가 무효화(INVALID)가 된다.

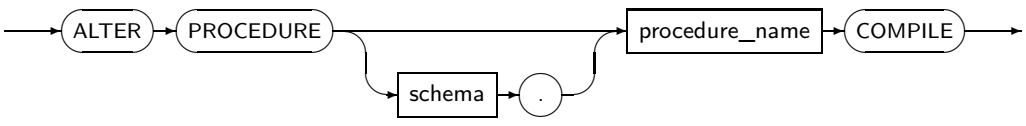
참고

ALTER PROCEDURE 문의 재컴파일 과정과 유효화, 무효화 과정은 ALTER FUNCTION 문과 ALTER PACKAGE 문과 ALTER TYPE 문에서도 동일하게 적용된다.

ALTER PROCEDURE의 세부 내용은 다음과 같다.

- 문법

alter_procedure



- 특권

  사용자가 소유한 프러시저이거나 ALTER ANY PROCEDURE 시스템 특권을 부여받아야 한다.

- 구성요소

구성요소	설명
schema	프러시저가 속해 있는 스키마의 이름을 명시한다.
procedure_name	재컴파일을 진행할 프러시저의 이름을 명시한다.

- 예제

```
ALTER PROCEDURE tibero.raise_salary COMPILE;
```

7.10. ALTER PROFILE

프로파일의 속성을 변경한다.

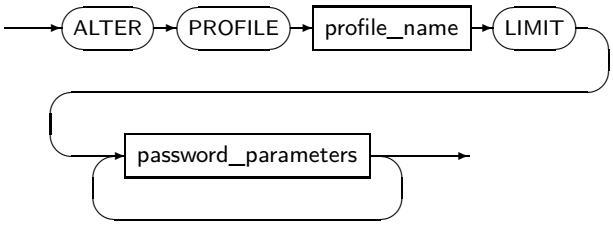
참고

프로파일을 생성, 제거하기 위해서는 "7.37. CREATE PROFILE"와 "7.58. DROP PROFILE"의 내용을 참고한다.

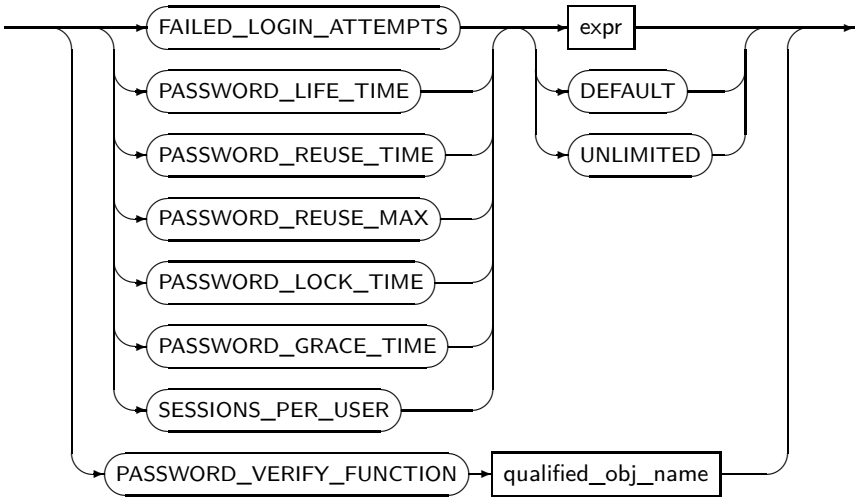
ALTER PROFILE의 세부 내용은 다음과 같다.

- 문법

alter_profile



password_parameters



- 특권

  ALTER PROFILE 시스템 특권이 있어야 한다.

- 구성요소

구성요소	설명
profile_name	변경할 프로파일의 이름을 명시한다.
password_paramenters	변경할 프로파일 속성을 지정한다.

7.11. ALTER ROLE

ROLE의 패스워드를 변경한다. ROLE에 포함된 특권 등은 GRANT나 REVOKE를 사용하여 부여하거나 회수하고, ALTER ROLE로는 단순히 ROLE의 패스워드만을 변경한다.

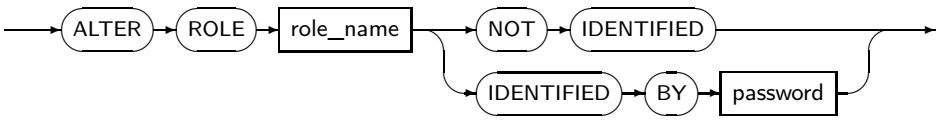
참고

ROLE의 추가와 제거에 대해서는 "7.38. CREATE ROLE", "7.59. DROP ROLE"을 참고한다.

ALTER ROLE의 세부 내용은 다음과 같다.

- 문법

alter_role



- 특권
  - ALTER ROLE로 ROLE의 패스워드를 변경하는 것은 해당 ROLE을 생성한 사용자이거나, WITH ADMIN OPTION으로 관리 특권을 부여 받은 사용자에 한해 가능하다.
  - ALTER ANY ROLE 시스템 특권이 있는 사용자는 자신이 생성하지도 않고, 관리 특권을 부여받지 않은 ROLE의 패스워드도 변경할 수 있다.
- 구성요소

구성요소	설명
role_name	패스워드를 변경할 ROLE의 이름이다. 해당 ROLE은 이미 CREATE ROLE을 통해 만들어져 있어야 한다.
NOT IDENTIFIED	ROLE의 패스워드를 제거한다.
IDENTIFIED BY	ROLE의 패스워드를 변경한다.

구성요소	설명
password	변경할 패스워드를 입력한다.

- 예제

  다음은 **ALTER ROLE**을 사용해 ROLE의 패스워드를 변경하는 예이다.

```
SQL> CONN sys/tibero
Connected

SQL> CREATE ROLE a;
Role created.

SQL> SELECT role, password_required FROM dba_roles
     WHERE role='A';

ROLE                           PAS
------------------------------ ---
A                              NO

1 row selected.

SQL> ALTER ROLE a IDENTIFIED BY 'xxx';
Role altered.

SQL> SELECT role, password_required FROM dba_roles
     WHERE role='A';

ROLE                       PAS
------------------------------ ---
A                              YES

1 row selected.

SQL> ALTER ROLE a NOT IDENTIFIED;
Role altered.

SQL> SELECT role, password_required FROM dba_roles
     WHERE role='A';

ROLE                           PAS
------------------------------ ---
A                              NO

1 row selected.
```

위의 예를 보면 우선 CREATE ROLE을 사용하여 패스워드를 지정하지 않고 ROLE을 생성한다. 그 다음 생성된 ROLE을 ALTER ROLE을 사용하여, 패스워드를 'xxx'로 지정했다가 다시 제거하고 있다.

참고

패스워드 사용과 관련된 예는 "7.38. CREATE ROLE"이나 "9.7. SET ROLE"을 참고한다.

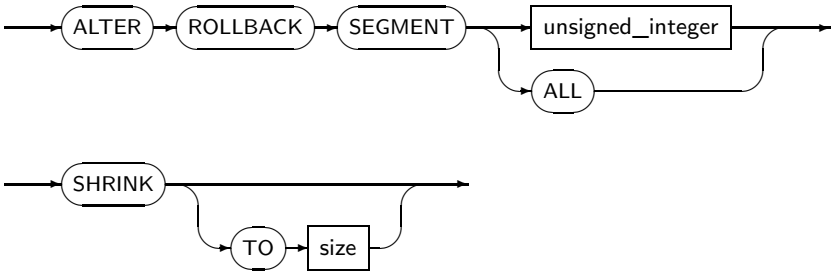
7.12. ALTER ROLLBACK SEGMENT

Undo segment를 최소 크기 또는 지정한 크기만큼 줄인다. 단, undo segment에 undo retention 시간이 지난 재사용 가능 공간이 없거나 실행 중인 트랜잭션이 많은 경우 해당 undo segment의 공간을 줄이지 못할 수도 있다.

ALTER ROLLBACK SEGMENT의 세부 내용은 다음과 같다.

- 문법

alter_rollback_segment



- 특권

  ALTER ROLLBACK SEGMENT 시스템 특권이 있어야 한다.

- 구성요소

구성요소	설명
unsigned_integer	undo segment 번호를 명시한다.
size	undo segment의 크기를 지정한다. (단위: byte)

7.13. ALTER SEQUENCE

지정된 시퀀스의 정의를 변경한다.

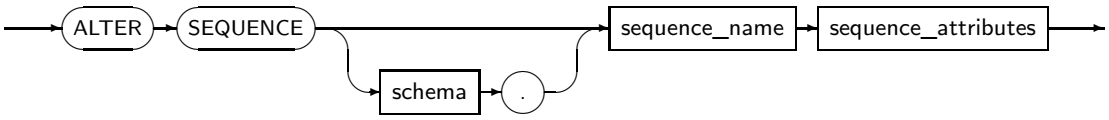
참고

시퀀스의 생성과 제거에 대해서는 "7.39. CREATE SEQUENCE"와 "7.60. DROP SEQUENCE"를 참고한다.

ALTER SEQUENCE의 세부 내용은 다음과 같다.

- 문법

alter_sequence



- 특권

  시퀀스가 사용자가 소유한 스키마에 있거나, ALTER_ANY_SEQUENCE 시스템 특권을 부여받아야 한다.

- 구성요소

구성요소	설명
schema	생성할 시퀀스를 포함하는 스키마의 이름이다. 생략하면 현재 사용자의 스키마로 인식된다.
sequence_name	생성할 시퀀스의 이름이다. 시퀀스의 이름은 VARCHAR 타입으로 저장되고 길이는 최대 30자까지 가능하다. 시퀀스의 이름은 테이블과 같은 네임스페이스를 사용한다. 따라서 스키마의 다른 시퀀스, 테이블, 동의어, PSM의 이름과 중복되어서는 안 된다.
sequence_attributes	ALTER SEQUENCE를 사용해 이미 존재하는 시퀀스의 증가값, 최솟값, 최댓값, 저장해 놓은 시퀀스 번호의 개수 등을 변경할 수 있고, 시퀀스의 속성을 변경할 수 있다. 이러한 변경 사항은 앞으로 생성될 시퀀스 번호에만 적용된다. sequence_attributes에 캐시를 사용하는 경우 ALTER SEQUENCE에 의해 몇 개의 값이 누락될 수 있다. ALTER SEQUENCE를 사용하면 이미 시퀀스 캐시에 존재하던 값을 모두 무효화시켜 버리기 때문이다. START WITH는 변경할 수 없다. 자세한 내용은 "7.39. CREATE SEQUENCE"의 sequence_attributes를 참고한다.

- 예제

  다음은 "7.39. CREATE SEQUENCE"의 예제에서 생성한 test_seq라는 시퀀스의 속성을 변경하는 예이다.

```
SQL> ALTER SEQUENCE test_seq MINVALUE 10 INCREMENT BY 3;
Altered.

SQL> SELECT test_seq.nextval FROM dual;

   NEXTVAL
----------
       105

1 row selected.

SQL> SELECT test_seq.nextval FROM dual;

   NEXTVAL
----------
       108

1 row selected.
```

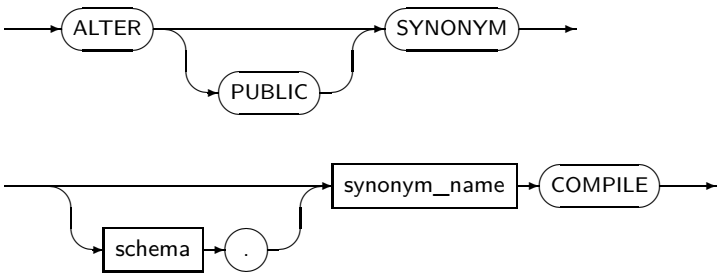
7.14. ALTER SYNONYM

현재 존재하는 SYNONYM의 상태를 수정할 때 사용한다.

ALTER SYNONYM의 세부 내용은 다음과 같다.

- 문법

alter_synonym



- 특권
  - 자신의 SCHEMA 내에 있는 SYNONYM에 대해서는 특권이 필요 없으며, 다른 SCHEMA의 SYNONYM을 수정하기 위해서는 CREATE ANY SYNONYM과 DROP ANY SYNONYM 특권이 있어야 한다.

- PUBLIC SYNONYM을 수정하기 위해서는 CREATGE PUBLIC SYNONYM과 DROP PUBLIC SYNONYM 특권이 있어야 한다.

- 구성요소

  - alter_synonym

구성요소	설명
PUBLIC	PUBLIC SYNONYM에 대한 수정을 한다.
COMPILE	SYNONYM을 revalidate 한다. SYNONYM의 target object의 status가 invalid일 경우 해당 object의 recompile 로직을 수행하고, 그 외의 경우에는 SYNONYM의 status는 target object의 status를 그대로 취한다.

- 예제

  다음은 **ALTER SYNONYM COMPILE**을 사용하여 SYNONYM을 컴파일하는 예이다.

```
ALTER SYNONYM SYN_1 COMPILE;
ALTER SYNONYM USER2.SYN_2 COMPILE;
ALTER PUBLIC SYNONYM SYN_3 COMPILE;
```

  다음은 **ALTER PUBLIC SYNONYM COMPILE**을 사용했을 때 권한이 없는 경우 에러가 발생하는 예이다.

```
SQL> ALTER PUBLIC SYNONYM PUB_SYN COMPILE;
TBR-17004: Permission denied.
```

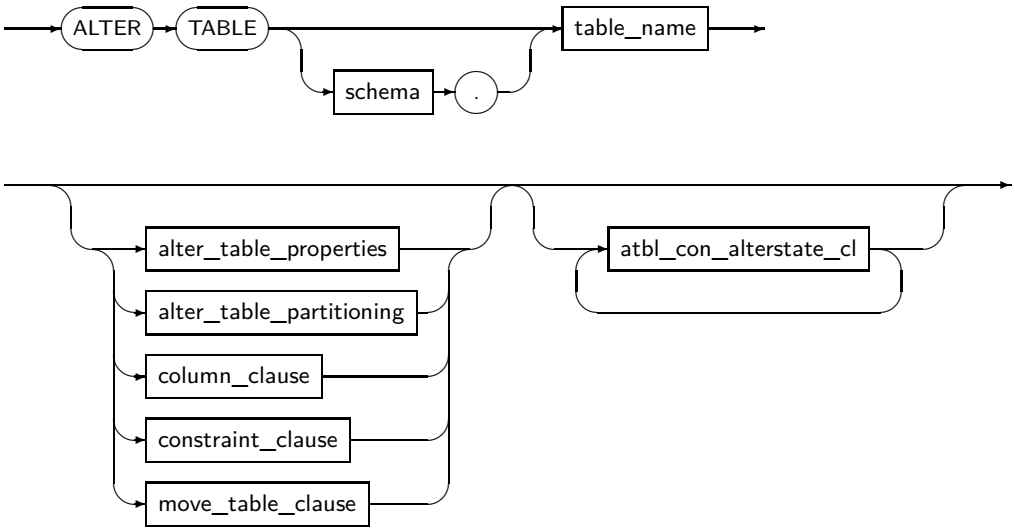
7.15. ALTER TABLE

생성된 테이블을 변경한다.

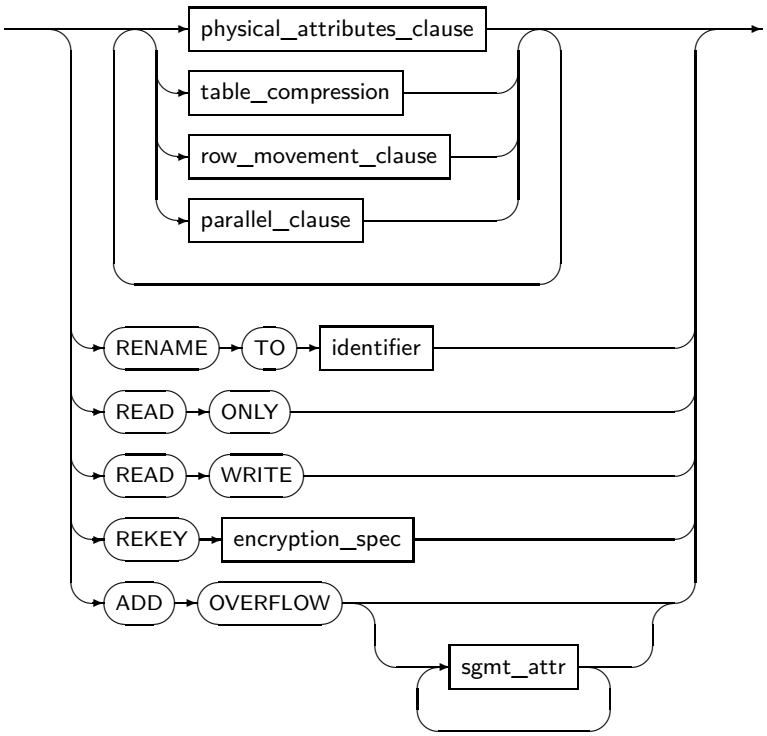
ALTER TABLE의 세부 내용은 다음과 같다.

- 문법

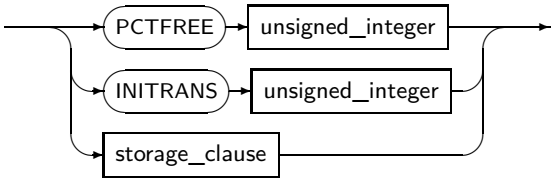
alter_table



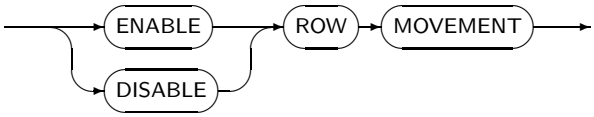
alter_table_properties



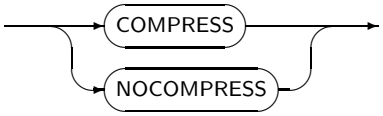
physical_attributes_clause



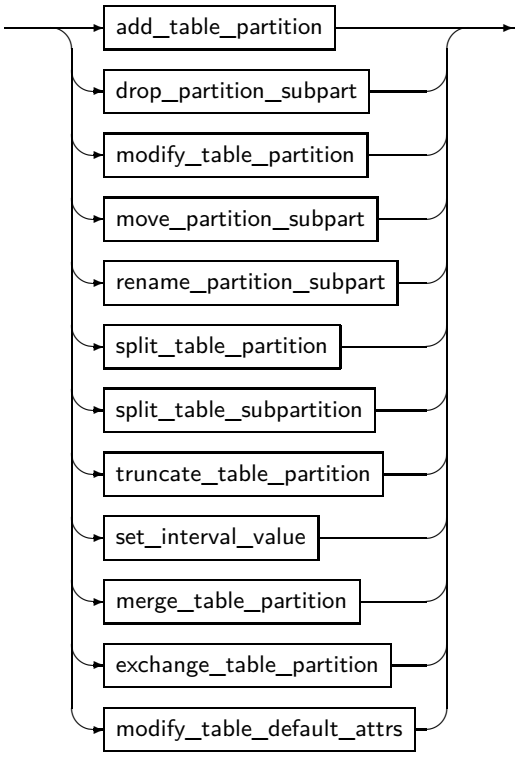
row_movement_clause



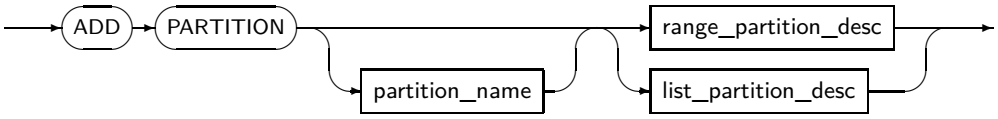
table_compression



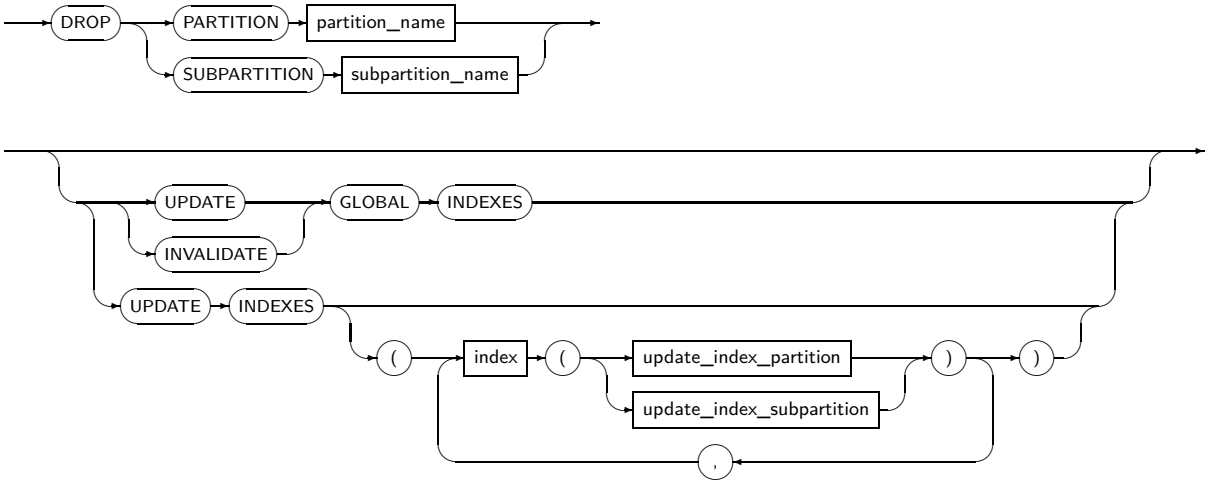
alter_table_partitioning



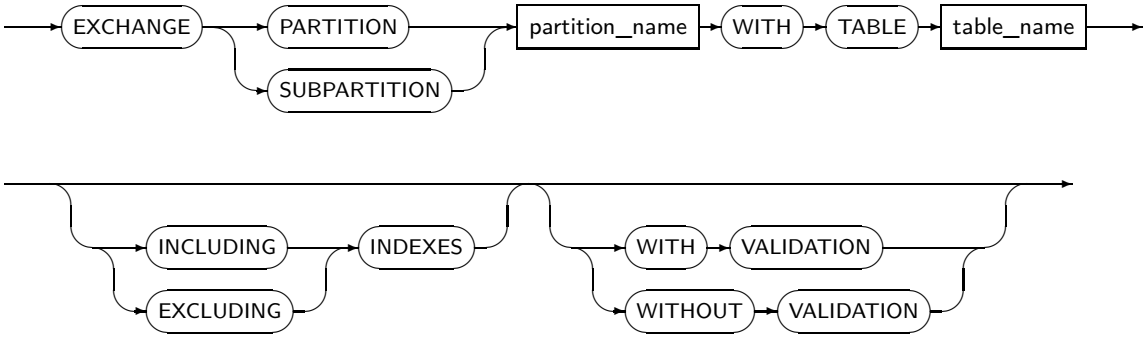
add_table_partition



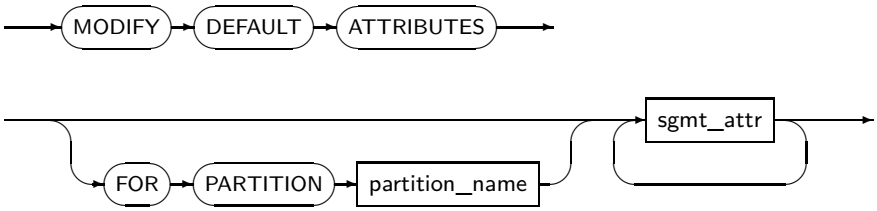
drop_partition_subpart



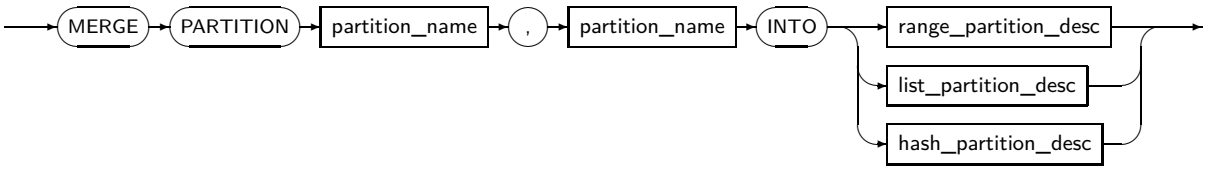
exchange_table_partition



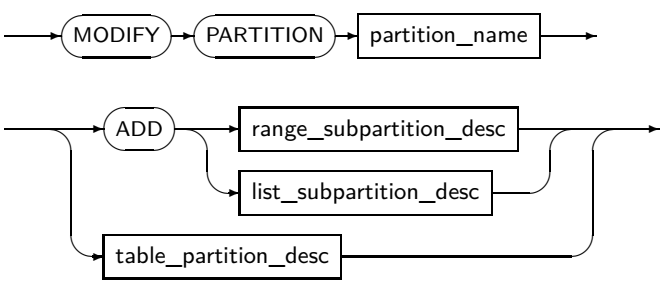
modify_table_default_attrs



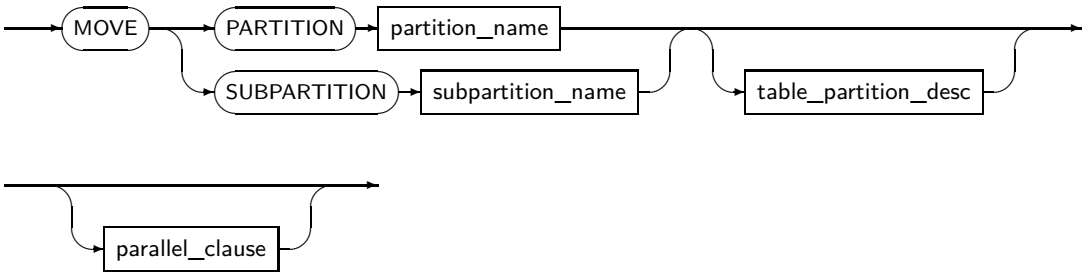
merge_table_partition



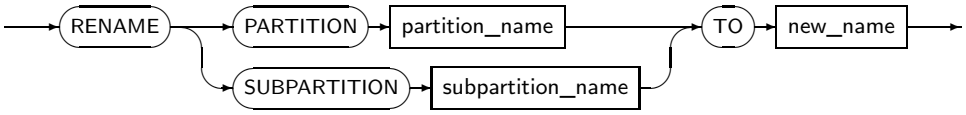
modify_table_partition



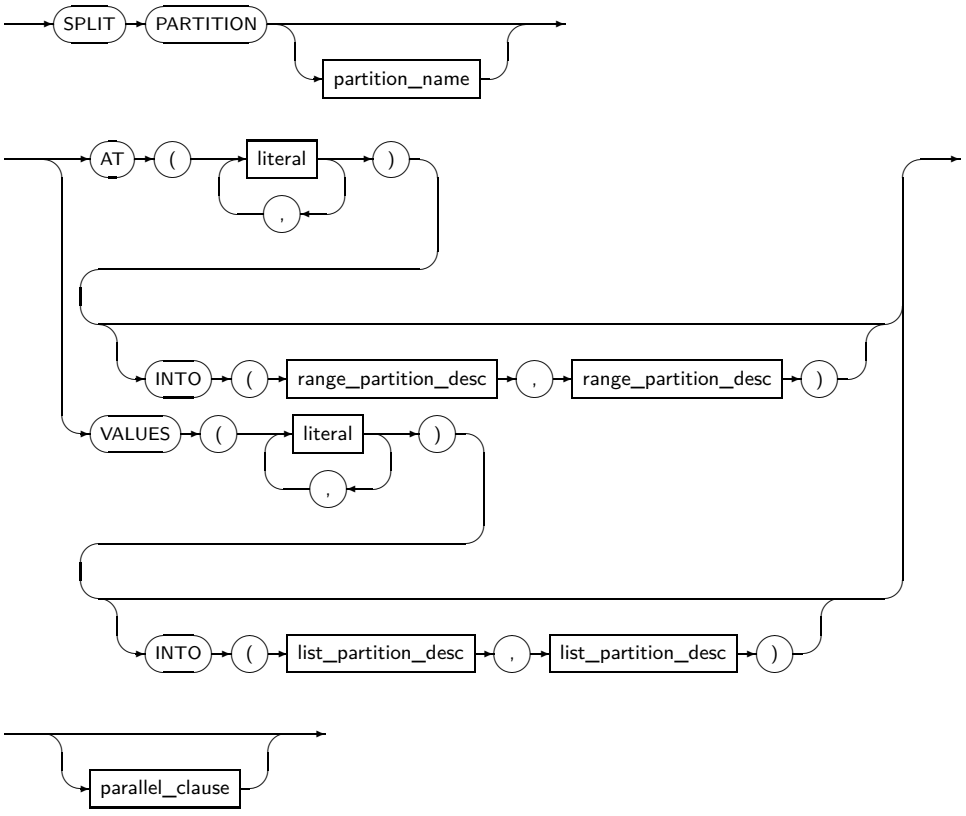
move_partition_subpart



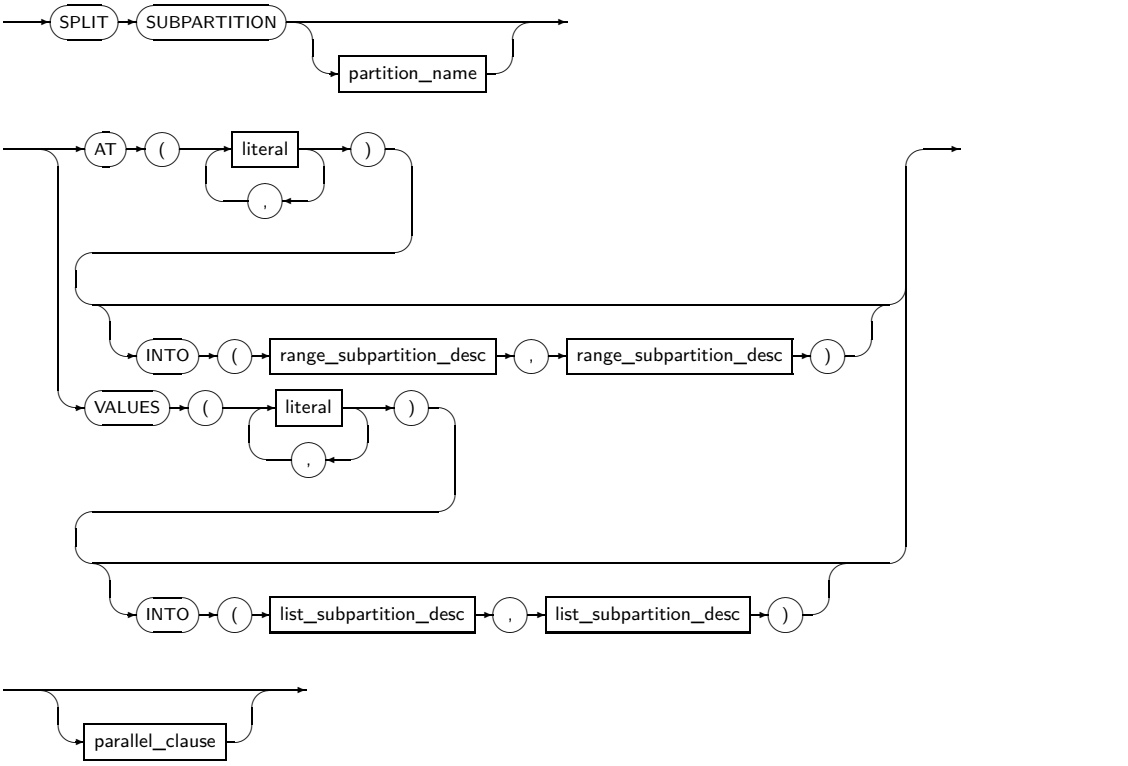
rename_partition_subpart



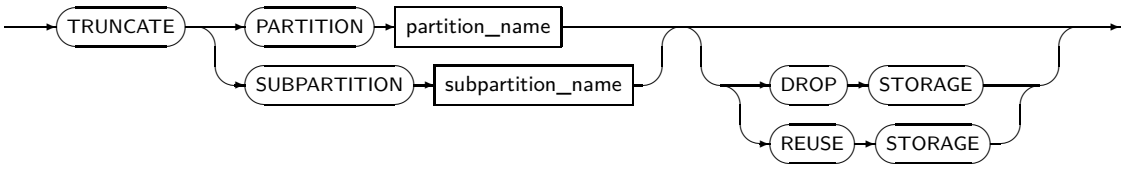
split_table_partition



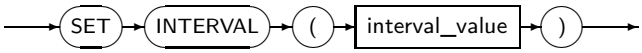
split_table_subpartition



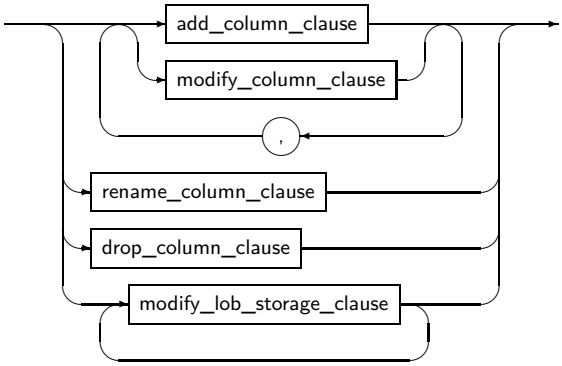
truncate_table_partition



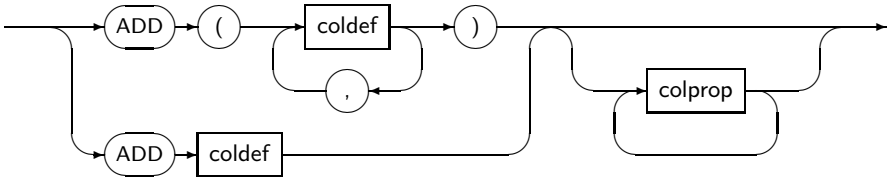
set_interval_value



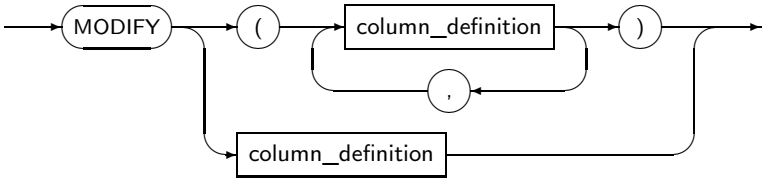
column_clause



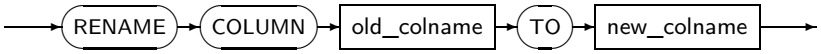
add_column_clause



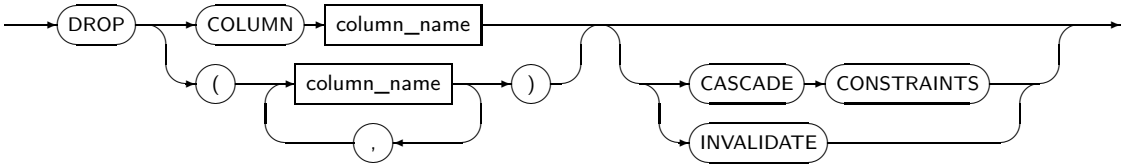
modify_column_clause



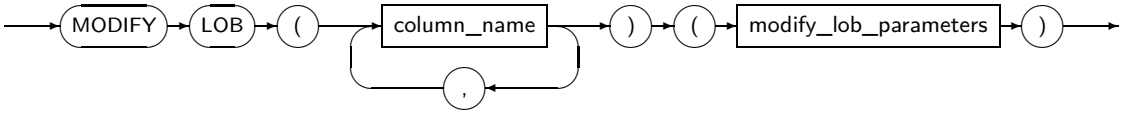
rename_column_clause



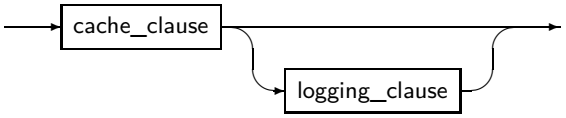
drop_column_clause



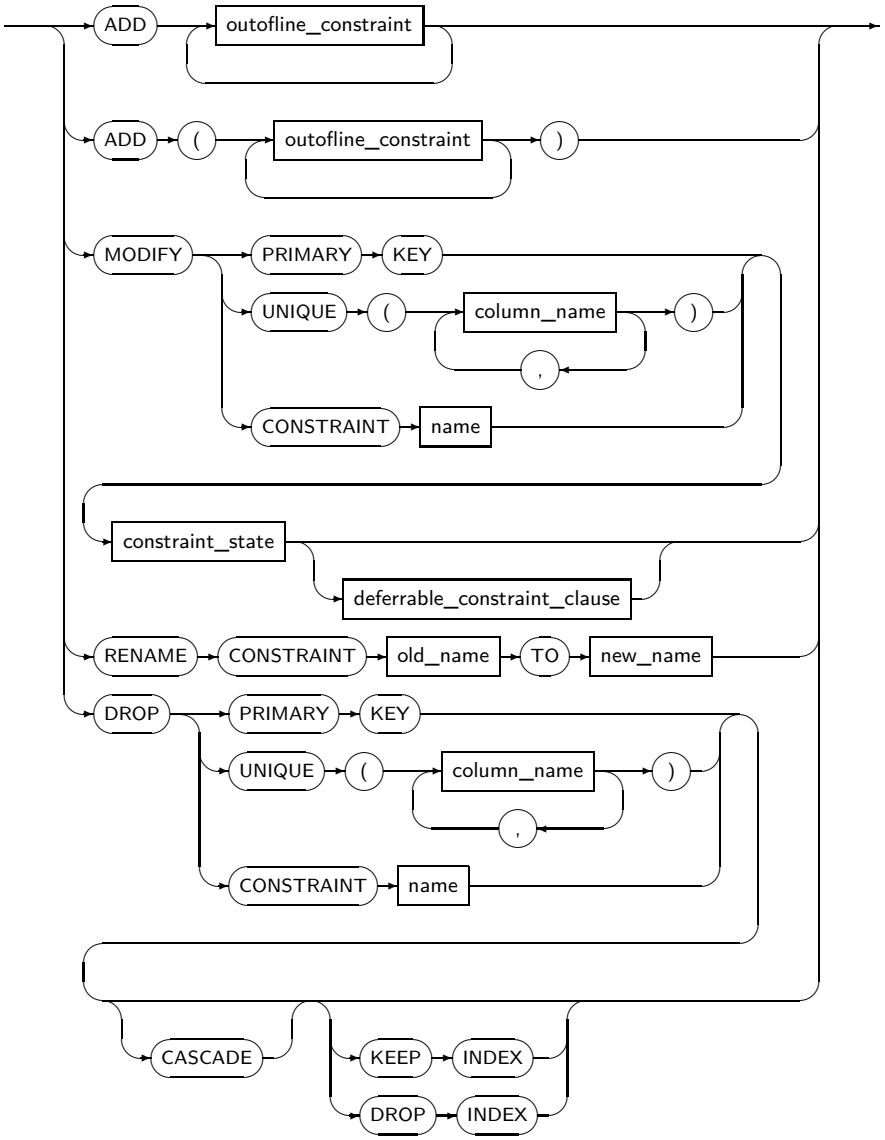
modify_lob_storage_clause



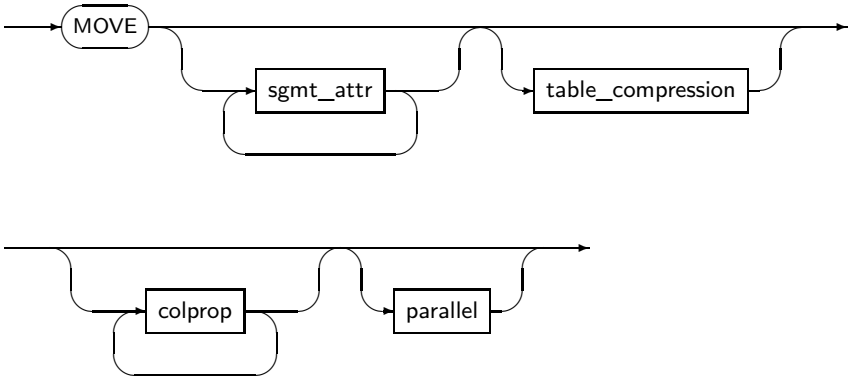
modify_lob_parameters



constraint_clause



move_table_clause



- 특권

  사용자가 소유한 스키마의 테이블을 변경하기 위해서는 별다른 특권은 필요하지 않다. 다만, 다른 사용자의 스키마의 테이블을 변경하기 위해서는 ALTER ANY TABLE 시스템 특권이 있어야 한다.

- 구성요소

  – alter_table

구성요소	설명
schema	변경할 테이블이 속해 있는 스키마의 이름을 명시한다. 생략하면 현재 사용자의 스키마로 인식된다.
table_name	변경할 테이블의 이름을 명시한다.
alter_table_properties	PCTFREE, INITRANS, storage_clause 등 물리적인 속성을 변경한다.
atbl_con_alterstate_cl	제약조건의 상태를 변경할 때 사용한다.
alter_table_partitioning	파티션 테이블에만 사용할 수 있다. 파티션 테이블에 관련된 설명은 “7.41. CREATE TABLE”을 참고한다.
column_clause	테이블의 컬럼을 추가, 수정, 제거한다.
constraint_clause	테이블의 제약조건을 추가, 수정, 제거한다.
move_table_clause	테이블에 대해 물리적인 속성을 새로 지정하여 세그먼트를 생성한 후 기존 테이블을 새 세그먼트로 이동시킨다.

  – alter_table_properties

구성요소	설명
physical_attributes_clause	PCTFREE, INITRANS, storage_clause 등 물리적인 속성을 변경한다.
table_compression	테이블의 압축을 여부를 지정한다.
parallel_clause	테이블에 대한 DML을 수행할 때 참조할 기본 DOP(Degree of Parallelism)을 설정한다.
RENAME	테이블의 이름을 변경한다.
TO identifier	변경할 테이블의 새 이름을 지정한다.
READ ONLY	테이블을 READ ONLY 모드로 지정한다.
READ WRITE	테이블을 READ WRITE 모드로 지정한다.
REKEY encryption_spec	REKEY는 데이터베이스가 새로운 암호화 키를 생성하도록 한다. ALTER TABLE에서 다른 문장과 혼용될 수 없다. 테이블의 모든 암호화된 컬럼이 새로운 키로 암호화되고, encryption_spec에 USING을 사용할 경우 새로운 암호화 알고리즘으로 암호화된다.
ADD OVERFLOW	IOT 테이블에 Overflow Segment를 추가한다.

  – physical_attributes_clause

구성요소	설명
PCTFREE unsigned_integer	데이터를 디스크 블록에 저장할 때 데이터가 변경되어 크기가 증가할 것에 대비하여 얼마만큼의 영역을 예비로 남겨둘지를 설정하는 값이다. 1 ~ 99 사이의 값을 설정할 수 있으며, 지정하지 않으면 기본값은 10이다. unsigned_integer에 해당 값을 명시한다.
INITRANS unsigned_integer	디스크 블록마다 트랜잭션 엔트리를 위한 공간을 몇 개를 확보할 것인가를 나타낸다. 트랜잭션 엔트리는 블록에 공간이 남아있다면 필요할 때 확장된다. 따라서 미리 큰 값을 설정할 필요는 없다. 최솟값은 1이며, 최댓값은 디스크 블록의 크기에 따라 다르다. 지정하지 않으면 기본값은 2이다. unsigned_integer에 해당 값을 명시한다.
storage_clause	세그먼트의 세부적인 속성을 정의한다. 자세한 내용은 “7.1.5. Storage_clause”를 참고한다.

– table_compression

구성요소	설명
COMPRESS	테이블을 압축한다.
NOCOMPRESS	테이블을 압축하지 않는다.

– row_movement_clause

구성요소	설명
ENABLE ROW MOVEMENT	데이터베이스가 ROW를 이동하도록 허용한다. 따라서 ROWID가 변경될 수 있다.
DISABLE ROW MOVEMENT	데이터베이스가 ROW를 이동하도록 허용하지 않는다.

– alter_table_partitioning

구성요소	설명
add_table_partition	파티션 테이블의 마지막 파티션 뒤에 새 파티션을 추가한다. 기존의 파티셔닝 방법에 맞게 파티션을 추가해야 한다. 파티션의 이름은 지정하지 않으면 자동으로 생성된다. – RANGE 파티션의 경우: 새 파티션의 경계 값이 맨 마지막의 파티션보다 커야 한다. 따라서 MAXVALUE 값을 가진 파티션이 있으면 파티션을 추가할 수 없다. – LIST 파티션의 경우: 새 파티션의 파티셔닝 키 값에 기존 파티션에 들어간 값이 중복되지 않아야 하며, 테이블에 기본값을 가진 파티션이 있는 경우에는 파티션을 추가할 수 없다.

구성요소	설명
	현재 HASH 파티션 테이블에 대해서는 파티션 추가를 지원하지 않는다.
drop_partition_subpart	주어진 이름의 파티션 또는 서브 파티션을 제거한다. 단, 파티션 또는 서브 파티션이 마지막 하나만 남아 있는 경우에는 제거할 수 없다.
modify_table_partition	기존 파티션의 구조를 변경한다. 현재는 서브 파티션의 추가만 지원한다. 서브 파티션의 추가는 복합 파티션 테이블에만 사용할 수 있으며, 기존 서브 파티셔닝 방법에 맞게 파티션을 추가해야 한다. 서브 파티션의 이름은 지정하지 않으면 자동으로 생성된다. 서브 파티션을 추가할 때의 제약 사항은 add_table_partition과 마찬가지이며, 자세한 문법은 **'sub_partition_desc'**를 참고한다.
move_partition_subpart	주어진 이름의 파티션 또는 서브 파티션에 대해 물리적 속성을 새로 지정하여 세그먼트를 생성한 후 기존 파티션을 새 세그먼트로 이동시킨다. 기본 파티션 테이블의 파티션과 복합 파티션 테이블의 서브 파티션을 사용할 수 있다.
rename_partition_subpart	주어진 이름의 파티션 또는 서브 파티션의 이름을 변경한다.
split_table_partition	기존 파티션을 분할한다. HASH 파티션 테이블에는 사용할 수 없다. – RANGE 파티션의 경우: AT을 사용하여 파티셔닝 키의 경계 값을 지정하면 그 값을 기준으로 파티션이 두 개로 분할된다. 따라서 AT에서 지정하는 값이 분할하는 파티션이 가지는 범위 안에 있어야 한다. – LIST 파티션의 경우: VALUES를 사용하여 파티셔닝 키 값을 지정하며, 지정한 값을 포함하는 파티션과 그 외 나머지 값을 포함하는 파티션으로 분할된다. 따라서 VALUES 절에서 지정하는 값이 분할하는 파티션의 기존 키 값에 존재해야 하며, 기존 파티셔닝 키 값 전부를 지정할 수는 없다. 분할하는 파티션이 서브 파티션을 가진 경우에는 기존과 동일한 기준으로 서브 파티션을 생성한다.
split_table_subpartition	기존 서브 파티션을 분할한다. 서브 파티셔닝 방법은 해시 파티셔닝이 아닌 복합 파티션 테이블에서 사용할 수 있다. 서브 파티션을 분할한다는 것과 SUBPARTITION 예약어를 사용한다는 것 외에는 **'split_table_partition'**과 동일하다.
merge_table_partition	두 개의 파티션을 하나의 파티션으로 합친다.
exchange_table_partition	주어진 파티션과 테이블의 세그먼트를 교환한다.

구성요소	설명
modify_table_default_attrs	테이블의 기본 attribute들을 변경한다.
truncate_table_partition	파티션에 존재하는 모든 ROW를 제거한다. 만약 파티션이 서브 파티션을 가지고 있다면, 포함하는 모든 서브 파티션의 ROW가 제거 된다. 서브 파티션의 이름을 주면 해당하는 서브 파티션의 모든 ROW가 제거 된다.
set_interval_value	Range 파티션의 Interval 값을 변경한다. 파티션 테이블에 DML이 발생할 때, 파티션 키 컬럼 값에 해당하는 파티션이 없을경우 새로운 파티션을 만들어 준다. 이 때, 기준 값에서 Interval value로 지정한 값의 배수로 새로 만들 파티션의 기준 값이 결정된다. ALTER TABLE 명령으로 interval value를 변경할 경우 Range 파티션의 마지막 파티션 값이 기준 값이 된다. 최초의 경우 CREATE TABLE 명령에서 명시한 파티션 중 마지막 파티션이 기준값으로 이용된다.

– add_table_partition

구성요소	설명
partition_name	추가할 파티션의 이름을 명시한다.
range_partition_desc	RANGE 파티션의 세부적인 설정을 지정한다.
list_partition_desc	LIST 파티션의 세부적인 설정을 지정한다.
hash_partition_desc	HASH 파티션의 세부적인 설정을 지정한다.

– drop_partition_subpart

구성요소	설명
PARTITION	파티션을 제거할 때 명시한다.
partition_name	제거할 파티션의 이름을 명시한다.
SUBPARTITION	서브 파티션을 제거할 때 명시한다.
subpartition_name	제거할 서브 파티션의 이름을 명시한다.
UPDATE GLOBAL INDEXES	GLOBAL INDEX들을 업데이트한다.
INVALIDATE GLOBAL INDEXES	GLOBAL INDEX들을 Invalid 상태로 변경한다.
UPDATE INDEXES	지정한 INDEX들을 업데이트한다.

– modify_table_partition

구성요소	설명
partition_name	변경할 파티션의 이름을 명시한다.

구성요소	설명
ADD	서브 파티션을 추가한다. (현재 LIST-HASH나 RANGE-HASH와 같이 서브 파티셔닝 방법이 HASH인 경우 지원 안함)
range_subpartition_desc	RANGE 파티션의 세부적인 설정을 지정한다.
list_subpartition_desc	LIST 파티션의 세부적인 설정을 지정한다.
hash_subpartition_desc	HASH 파티션의 세부적인 설정을 지정한다.
table_partition_desc	파티션의 물리적인 속성을 지정한다. 문법은 테이블의 속성을 지정할 때와 유사하며, 파티션의 속성을 지정한다는 것만 다르다.

– move_partition_subpart

구성요소	설명
PARTITION	파티션을 이동할 때 명시한다.
partition_name	이동할 파티션의 이름을 명시한다.
SUBPARTITION	서브 파티션을 이동할 때 명시한다.
subpartition_name	이동할 서브 파티션을 명시한다.
table_partition_desc	파티션의 물리적인 속성을 지정한다. 지정하는 방법은 테이블의 속성을 지정할 때와 유사하며, 파티션의 속성을 지정한다는 것만 다르다.

– rename_partition_subpart

구성요소	설명
PARTITION partition_name	변경할 대상 파티션의 이름을 명시한다.
subpartition_name	변경할 대상 서브 파티션의 이름을 명시한다.
TO new_name	변경할 파티션의 새로운 이름을 명시한다.

– split_table_partition

구성요소	설명
partition_name	분할할 파티션의 이름을 명시한다.
AT literal	RANGE 파티션의 경우 AT을 사용하여 파티셔닝 키의 경계 값을 지정한다. literal에 지정하는 값을 기준으로 파티션이 두 개로 분할된다.
INTO range_partition_desc	분할할 파티션의 이름과 속성을 지정한다.
VALUES literal	LIST 파티션의 경우 VALUES를 사용하여 파티셔닝 키 값을 지정하며, 지정한 값을 포함하는 파티션과 그 이외의 값을 갖는 파티션으로 분할된다. literal에 지정하는 값은 파티션의 기존 키 값으로 존재해야 하며, 기본 파티셔닝 키 값 전부를 지정할 수는 없다.
INTO list_partition_desc	분할할 파티션의 이름과 속성을 지정한다.

– split_table_subpartition

구성요소	설명
partition_name	분할할 서브 파티션의 이름을 명시한다.
AT literal	RANGE 파티션의 경우 AT을 사용하여 파티셔닝 키의 경계 값을 지정한다. literal에 지정하는 값을 기준으로 파티션이 두 개로 분할된다.
INTO range_subpartition_desc	분할할 서브 파티션의 이름과 속성을 지정한다.
VALUES literal	LIST 서브 파티션의 경우 VALUES를 사용하여 파티셔닝 키 값을 지정하며, 지정한 값을 포함하는 서브 파티션과 그 이외의 값을 갖는 서브 파티션으로 분할된다. literal에 지정하는 값은 서브 파티션의 기존 키 값으로 존재해야 하며, 기본 파티셔닝 키 값 전부를 지정할 수는 없다.
INTO list_subpartition_desc	분할할 서브 파티션의 이름과 속성을 지정한다.

– exchange_table_partition

구성요소	설명
partition_name	파티션 혹은 서브 파티션의 이름을 명시한다.
table_name	파티션과 세그먼트를 교환할 테이블의 이름을 명시한다.
INCLUDING INDEXES	로컬 인덱스 파티션 또는 서브 파티션을 해당 테이블 인덱스(파티션되지 않은 테이블의 경우) 또는 로컬 인덱스(해시 파티션된 테이블의 경우)와 교환하도록 하려면 지정한다.
EXCLUDING INDEXES	파티션에 해당하는 모든 인덱스 파티션 또는 서브 파티션과 표시할 교환 테이블의 모든 일반 인덱스 및 인덱스 파티션을 원하는지 여부를 지정한다. INCLUDING INDEXES, EXCLUDING INDEXES를 명시하지 않으면 기본으로 EXCLUDING INDEXES로 수행한다.
WITH VALIDATION	테이블의 Row들이 파티션 조건, 제약 조건 등에 맞는지 체크를 한 후 맞지 않는다면, 에러를 발생시킨다. WITH VALIDATION, WITHOUT VALIDATION을 명시하지 않으면 기본으로 VALIDATION을 수행한다.
WITHOUT VALIDATION	테이블의 Row들이 파티션 조건, 제약 조건등과 맞지 않아도 세그먼트 교환을 수행한다.

– modify_table_default_attrs

구성요소	설명
partition_name	파티션 혹은 서브 파티션의 이름을 명시한다.
sgmt_attr	sgmt_attr 관련 문법은 “7.1.4. Sgmt_attr”을 참고한다.

– merge_table_partition

구성요소	설명
first_partition_name	합병될 첫 번째 파티션의 이름을 명시한다.
second_partition_name	합병될 두 번째 파티션의 이름을 명시한다. range partition의 경우 첫 번째 partition의 바로 다음 파티션이 지정되어야 한다.
INTO partition_desc	합병으로 만들어질 파티션의 이름과 속성을 지정한다.

– truncate_table_partition

구성요소	설명
PARTITION partition_name	분할할 파티션의 이름을 명시한다.
SUBPARTITION subpartition_name	분할할 서브 파티션의 이름을 명시한다.
DROP STORAGE	파티션이 사용하고 있는 공간을 회수한다. 즉, 할당받은 EXTENTS들을 모두 회수한다. 별도로 지정하지 않으면 기본적으로 DROP STORAGE로 동작한다.
REUSE STORAGE	파티션이 사용하고 있는 공간을 회수하지 않고, 그대로 사용한다.

– column_clause

구성요소	설명
add_column_clause	테이블에 새로운 컬럼을 추가한다. 이전에 존재하던 다른 로우의 컬럼은 디폴트로 설정한 값이 삽입된다. 만약 디폴트가 설정되어 있지 않으면 NULL 값이 삽입된다. Inline 제약조건으로 설정하면 이전에 이미 삽입된 컬럼을 대상으로 검증을 하게 되므로, 만족하지 않는 경우 컬럼 추가 자체가 실패할 수도 있다. 만약 LONG 타입의 컬럼이 존재하는 테이블이라면 새로운 컬럼을 추가 할 수 없다.
modify_column_clause	테이블에 이미 존재하는 컬럼의 속성을 변경한다. 데이터 타입, 기본값, Inline 제약조건, Inline 참조 제약조건(Inline Referential Constraint)를 변경한다.
rename_column_clause	컬럼의 이름을 변경한다.
drop_column_clause	테이블에 이미 존재하는 컬럼을 제거한다. 컬럼만 제거되는 것이 아니라 제거되는 컬럼과 관련된 인덱스, 트리거, 주석 등도 같이 제거된다.
modify_lob_storage_clause	테이블에 이미 존재하는 LOB 컬럼의 속성을 변경한다. Cache 사용 여부, Logging 사용 여부를 변경한다.

구성요소	설명
REKEY encryption_spec	REKEY는 데이터베이스가 새로운 암호화 키를 생성하도록 한다. ALTER TABLE에서 다른 문장과 혼용될 수 없다. 테이블의 모든 암호화된 컬럼이 새로운 키로 암호화되고, encryption_spec에 USING을 사용할 경우 새로운 암호화 알고리즘으로 암호화된다.

– add_column_clause

구성요소	설명
coldef	컬럼의 데이터 타입과 제약조건 등을 설정한다. coldef 관련 문법은 “7.41. CREATE TABLE”을 참고한다.
colprop	컬럼별로 대용량 객체형 데이터 타입이 저장되는 방식을 설정한다. colprop 관련 문법은 “7.41. CREATE TABLE”을 참고한다.

– modify_column_clause

구성요소	설명
datatype	컬럼의 데이터 타입을 변경한다. NUMBER 타입을 CLOB 타입으로 바꾸는 등의 변경은 허락되지 않는다. NUMBER 타입에서 정밀도나 스케일을 늘리거나, VARCHAR 타입에서 컬럼의 길이를 늘리는 것은 항상 허용된다. 하지만, NUMBER 타입의 정밀도나 스케일을 줄이기 위해서는 해당 컬럼의 데이터가 전부 NULL이거나 값이 없어야 하고, CHAR, VARCHAR 타입의 컬럼의 길이를 줄이기 위해서는 줄이고자 하는 길이보다 더 큰 컬럼 값이 존재하지 않아야 한다.
DEFAULT	컬럼의 기본값을 변경한다.
inline_constraint	Inline 제약조건을 추가 또는 변경할 수 있다.

– rename_column_clause

구성요소	설명
old_colname	RENAME COLUMN은 기존 컬럼의 이름을 변경할 때 사용한다. old_colname에는 이름을 변경하고 싶은 컬럼의 기존 이름을 명시한다.
TO new_colname	new_colname에는 이름을 변경하고 싶은 컬럼의 새로운 이름을 명시한다.

– drop_column_clause

구성요소	설명
column_name	제거할 컬럼의 이름을 명시한다.
CONSTRANTS	제약조건이 있는 컬럼을 제거하기 위해서 사용한다. 단, 파티션 테이블에서 파티셔닝 키로 사용되고 있는 컬럼은 제거할 수 없다.
INVALIDATE	INVALIDATE은 지정하지 않아도 된다. 제거한 컬럼이 있는 테이블과 관련된 뷰, 트리거 등이 자동으로 무효화되기 때문이다. 무효화된 객체는 다음번에 사용될 때 다시 검증된다.

– modify_lob_storage_clause

구성요소	설명
column_name	변경할 LOB 컬럼의 이름을 명시한다.
modify_lob_parameters	변경될 LOB의 저장 속성을 명시한다.

– modify_lob_parameters

구성요소	설명
cache_clause	LOB 컬럼을 cache 를 사용해서 읽을 것인지 명시한다.
logging_clause	LOB 컬럼에 작업을 할 때 redo log를 남길 것인지 지정한다. cache option을 사용할 경우 항상 logging을 사용하여야 한다.

– constraint_clause

구성요소	설명
ADD outofline constraint	새로운 outofline 제약조건을 추가한다. outofline 제약조건과 관련된 문법은 “7.1.1. 제약조건”을 참고한다.
RENAME CONSTRAINT	기존 제약조건의 이름을 변경한다.
old_name	old_name에는 이름을 변경하고 싶은 제약조건의 기존 이름을 명시한다.
TO new_name	new_name에는 이름을 변경하고 싶은 제약조건의 새로운 이름을 명시한다.
MODIFY	기존 제약조건의 상태를 변경한다.
DROP	기존의 제약조건을 제거한다.
constraint_state	constraint_state 등의 관련 문법은 “7.1.1. 제약조건”을 참고한다.
PRIMARY KEY	변경되거나 제거될 기본 키를 의미한다.
UNIQUE column_name	변경되거나 제거될 유일 키를 의미한다.
CONSTRAINT name	변경되거나 제거될 제약조건을 의미한다.
CASCADE	다른 테이블이나 같은 테이블의 컬럼으로부터 FOREIGN KEY 제약조건으로 참조되는 기본 키나 유일 키 제약조건을 제거하기 위해서는 반

구성요소	설명
	드시 CASCADE를 설정하여 관련된 FOREIGN KEY까지 함께 제거해야만 한다. FOREIGN KEY로 참조되는 상태에서 기본 키나 유일 키만 단독으로 제거할 수는 없다.
KEEP INDEX	기본 키, 유일 키, FOREIGN KEY와 같은 인덱스를 사용하는 제약조건을 제거하려 할 때 제약조건만 제거하고 해당 제약조건이 사용했던 인덱스는 그대로 유지하고자 할 때 사용한다.
DROP INDEX	기본 키, 유일 키, FOREIGN KEY와 같은 인덱스를 사용하는 제약조건을 제거하려 할 때 인덱스도 함께 제거하기 위해서 사용한다. KEEP INDEX나 DROP INDEX를 설정하지 않으면, DROP INDEX가 기본값이다.

- 예제
  - physical_attributes_clause

    다음은 테이블을 생성한 뒤에 **PCTFREE**와 **INITRANS**를 사용하는 예이다.

```
CREATE TABLE atbl_exmp
(
    col1 NUMBER(10, 5),
    col2 VARCHAR(10),
    CONSTRAINT atbl_exm_pri_con PRIMARY KEY(COL1),
    CONSTRAINT atbl_exm_unq_con UNIQUE (COL2)
);

CREATE TABLE atbl_exmp_foreign_key
(
 col1 REFERENCES atbl_exmp (col1)
);

ALTER TABLE atbl_exmp PCTFREE 15 INITRANS 3;
```

    다음은 **atbl_con_alterstate_cl**를 사용하는 예이다.

```
ALTER TABLE atbl_exmp PCTFREE 10 DISABLE PRIMARY KEY;

ALTER TABLE atbl_exmp ENABLE NOVALIDATE PRIMARY KEY;
```

  - column_clause

    다음은 **column_clause**를 사용하는 예이다.

```
ALTER TABLE atbl_exmp ADD
(
```

```
    col3 VARCHAR(20),
    col4 CLOB
)
LOB (col4) STORE AS lob_sgmt_100 (DISABLE STORAGE IN ROW) ;

ALTER TABLE atbl_exmp RENAME COLUMN col2 TO column2;

ALTER TABLE atbl_exmp MODIFY
(
    col1 NUMBER(15, 7),
    col3 VARCHAR(15)
);
```

위의 예에서는 차례대로 add_column_clause, rename_column_cluase, modify_column_clause를 사용하는 것을 보여준다.

다음은 **add_column_clause**와 **modify_column_clause**를 사용하는 예이다.

```
ALTER TABLE atbl_exmp ADD CONSTRAINT unq_con2 UNIQUE (col1, col3);

ALTER TABLE atbl_exmp MODIFY UNIQUE(col1, col3) DISABLE NOVALIDATE;

ALTER TABLE atbl_exmp RENAME CONSTRAINT unq_con2 TO atbl_exm_2;

ALTER TABLE atbl_exmp DROP PRIMARY KEY CASCADE;

ALTER TABLE atbl_exmp DROP CONSTRAINT alter_exm_unq_con KEEP INDEX;
```

다음은 **rename_column_clause**를 사용하는 예이다.

```
ALTER TABLE atbl_exmp RENAME TO atbl_exmp_2;
```

- alter_table_partitioning

다음은 **alter_table_partitioning**을 사용하는 예이다.

```
CREATE TABLE atbl_part_exmp
(
    col1 NUMBER,
    col2 CLOB,
    col3 NUMBER
)
PARTITION BY RANGE (col1, col3)
(
    PARTITION atbl_part_1 VALUES LESS THAN (30, 40),
    PARTITION atbl_part_2 VALUES LESS THAN (50, 60)
);
```

```
ALTER TABLE atbl_part_exmp ADD PARTITION atbl_part_3
VALUES LESS THAN (60, 80);

ALTER TABLE atbl_part_exmp DROP PARTITION atbl_part_3;

ALTER TABLE atbl_part_exmp MOVE PARTITION atbl_part_2
TABLESPACE ts PCTFREE 14;

ALTER TABLE atbl_part_exmp RENAME PARTITION atbl_part_2 TO p2;

ALTER TABLE atbl_part_exmp SPLIT PARTITION p2 AT (40, 50)
INTO (PARTITION atbl_part_2, PARTITION atbl_part_3);
```

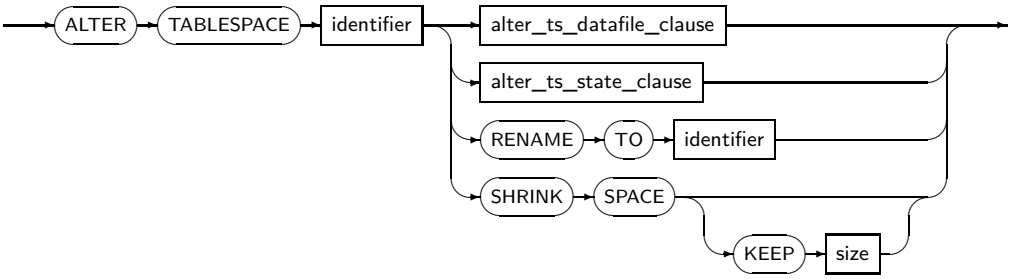
7.16. ALTER TABLESPACE

테이블 스페이스 또는 데이터 파일의 특성을 변경한다.

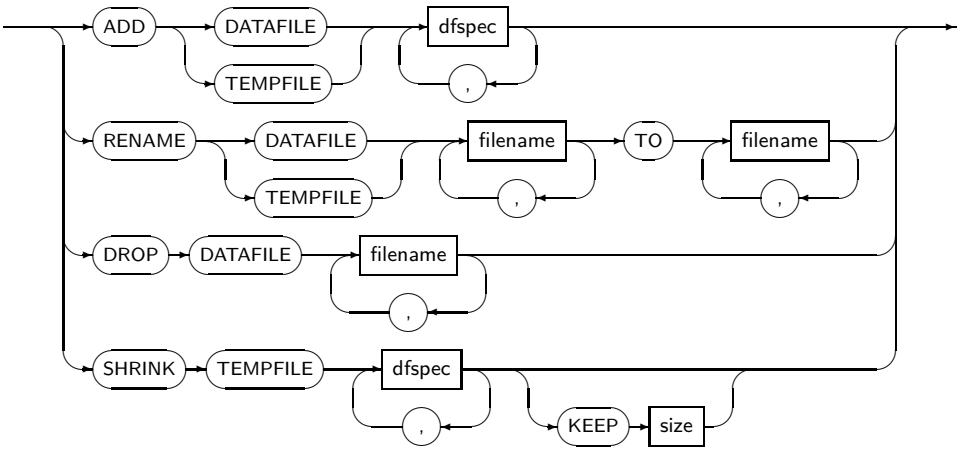
ALTER TABLESPACE의 세부 내용은 다음과 같다.

- 문법

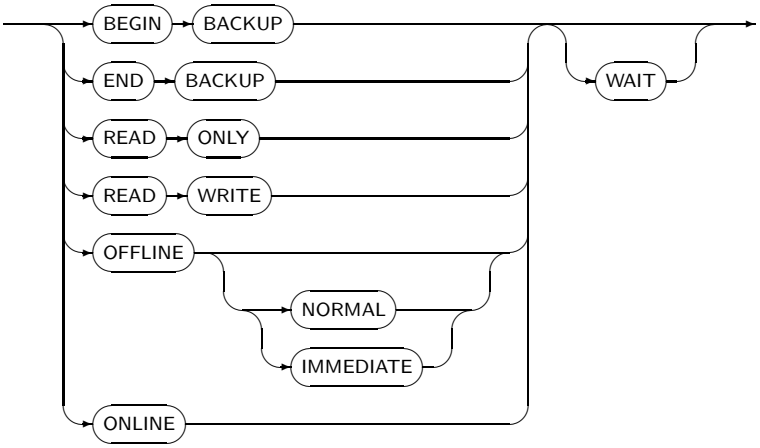
alter_tablespace



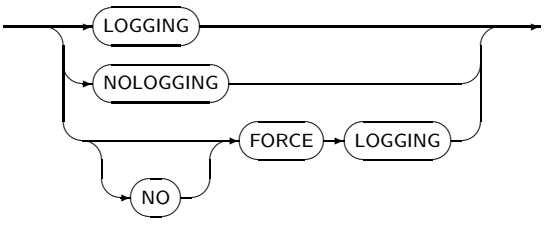
alter_ts_datafile_clause



alter_ts_state_clause



alter_ts_logging_clause



- 특권

  SYSDBA 특권이 있어야 한다.

- 구성요소
  - alter_tablespace

구성요소	설명
alter_ts_datafile_clause	테이블 스페이스에 속해 있는 파일의 특성을 변경한다.
alter_ts_state_clause	테이블 스페이스의 특성을 변경한다.
alter_ts_logging_clause	테이블 스페이스의 logging에 대한 설정을 변경한다.
SHRINK SPACE	임시 테이블 스페이스의 임시 파일들의 사용하지 않은 공간을 버려 전체 파일들의 크기를 축소시킨다. 임시 테이블 스페이스(Temporary Tablespace)일 때 사용한다. KEEP을 지정하면 최소 지정한 KEEP 크기 만큼 남기고 전체 파일을 축소시킨다.

  - alter_ts_datafile_clause

구성요소	설명
ADD DATAFILE	데이터 파일을 추가할 때 사용한다. 영속 테이블 스페이스(Permanent Tablespace) 또는 Undo 테이블 스페이스일 때 사용한다. 추가할 데이터 파일의 이름을 하나라도 지정하지 않으면, Tibero 시스템이 자동으로 데이터 파일을 생성하여 추가한다.
ADD TEMPFILE	임시 파일을 추가할 때 사용한다. 임시 테이블 스페이스(Temporary Tablespace)일 때 사용한다. 추가할 임시 파일의 이름을 하나도 지정하지 않으면 Tibero 시스템이 자동으로 임시 파일을 생성하여 추가한다.
dfspec	데이터 파일의 이름, 크기 등과 관련된 설정을 할 수 있다. 자세한 내용은 “7.25. CREATE DATABASE”를 참고한다.
RENAME DATAFILE	미디어 복구 중 데이터 파일의 경로를 바꾸고자 할 때 사용한다. 영속 테이블 스페이스 또는 Undo 테이블 스페이스일 때 사용한다.
RENAME TEMPFILE	미디어 복구 중 임시 파일의 경로를 바꾸고자 할 때 사용한다. 임시 테이블 스페이스일 때 사용한다.
DROP DATAFILE	데이터 파일을 삭제할 때 사용한다. 삭제할 파일은 내용이 비어있는 상태여야 한다.
SHRINK TEMPFILE	임시 파일의 크기를 축소시킨다. 임시 테이블 스페이스(Temporary Tablespace)일 때 사용한다. [KEEP size]는 축소시킬 수 있는 파일 크기의 하한을 나타낸다. 지정해주지 않으면 0으로 설정된다.

  - alter_ts_state_clause

구성요소	설명
BEGIN BACKUP	데이터베이스 운영 중 백업을 시작할 때 사용된다. 자세한 내용은 "Tibero 관리자 안내서"를 참고한다.
END BACKUP	데이터베이스 백업을 끝낼 때 사용된다. 자세한 내용은 "Tibero 관리자 안내서"를 참고한다.
OFFLINE	테이블 스페이스를 오프라인 상태로 변경한다. 테이블 스페이스가 오프라인 상태가 되면 테이블 스페이스에 속한 세그먼트로의 접근은 모두 차단된다. 또한, 테이블 스페이스가 가진 모든 데이타 파일도 오프라인 상태가 된다. – OFFLINE NORMAL: 해당 테이블 스페이스에 체크포인트를 수행한 후 오프라인 상태로 변경한다. 온라인 상태로 다시 전환할 때 복구 과정을 거치지 않는다. NORMAL이나 IMMEDIATE를 지정하지 않으면 NORMAL로 동작한다. – OFFLINE IMMEDIATE: NORMAL과 달리 체크포인트를 수행하지 않고 바로 오프라인 상태로 변경하기 때문에 온라인 상태로 전환하기 전에 미디어 복구를 해야 한다.
ONLINE	테이블 스페이스를 온라인 상태로 변경한다. 테이블 스페이스 복구가 필요하면 바로 온라인 상태로 변경할 수 없다. 미디어 복구를 해야만 온라인 상태로 변경할 수 있다.
WAIT	테이블 스페이스 변경을 위한 lock 획득을 기다린다. 테이블 스페이스 변경을 위해서는 테이블 스페이스 lock을 획득해야 한다. 기본 세팅의 경우 현재 다른 작업이 lock을 획득한 상태라면 lock 획득 실패라고 인지하고 DDL 구문 역시 실패 처리된다. WAIT 옵션을 주면 lock을 획득할 때까지 기다린 뒤에 구문을 수행한다.

– alter_ts_logging_clause

구성요소	설명
LOGGING / NOLOGGING	테이블 스페이스 내에 있는 오브젝트들에 대한 logging 설정을 하는 데 사용한다. – LOGGING: 오브젝트들에 대한 작업이 redo log로 남는다. – NOLOGGING: 오브젝트들에 대한 작업이 redo log로 남지 않는다. ALTER TABLESPACE로 logging 설정이 변경된 이후의 시점부터 해당 logging에 대한 설정이 적용된다.

구성요소	설명
FORCE LOGGING / NO FORCE LOGGING	– FORCE LOGGING: 테이블 스페이스 내에 있는 오브젝트들에 대한 작업이 logging이 되며, 기존에 nologging으로 설정이 되어있었어도 logging이 된다. – NO FORCE LOGGING: FORCE LOGGING 설정을 제거하는 기능이다. 단, 무조건 LOGGING을 하는 UNDO 테이블 스페이스와 LOGGING을 하지 않는 TEMP 테이블 스페이스에 대해서는 force logging 설정이 불가능하다.

- 예제

  다음은 테이블 스페이스에 데이터 파일을 추가하는 예이다. 본 예제에서는 DBA_TABLESPACES를 통해 테이블 스페이스가 사용하는 데이터 파일의 개수를 알 수 있다.

```
SQL> SELECT tablespace_name, ts_id, datafile_count
     FROM dba_tablespaces
     WHERE tablespace_name = 'T1';

TABLESPACE_NAME                     TS_ID DATAFILE_C
------------------------------ ---------- ----------
T1                                      3          1

1 row selected.

SQL> ALTER TABLESPACE t1 ADD DATAFILE 'ts2.dbf';

Tablespace altered.

SQL> SELECT tablespace_name,
            ts_id,
            datafile_count
     FROM dba_tablespaces
     WHERE tablespace_name = 'T1';

TABLESPACE_NAME                     TS_ID DATAFILE_C
------------------------------ ---------- ----------
T1                                      3          2

1 row selected.
```

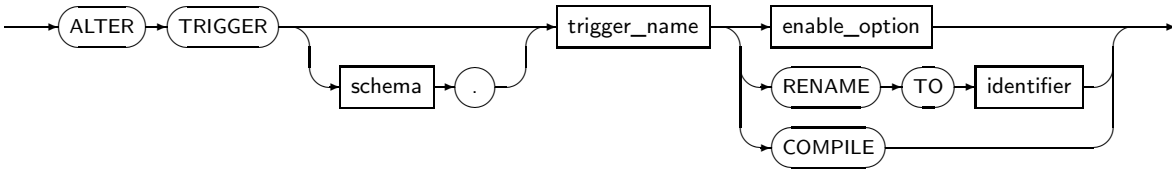
7.17. ALTER TRIGGER

데이터베이스 트리거를 컴파일하거나 트리거의 이름을 바꾼다. 또한, 해당 트리거를 활성화하거나 비활성화한다.

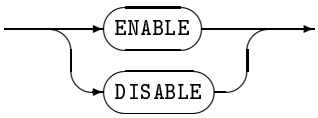
ALTER TRIGGER의 세부 내용은 다음과 같다.

- 문법

alter_trigger



enable_option



- 특권

  사용자의 스키마에 포함되어 있거나, 사용자가 ALTER ANY TRIGGER 시스템 특권을 가지고 있을 때에만 실행할 수 있다.

- 구성요소

  – alter_trigger

구성요소	설명
RENAME	트리거의 이름을 변경한다. 변경된 트리거는 이전의 상태를 계속 유지하게 된다.
COMPILE	트리거를 명시적으로 재컴파일한다. 명시적으로 재컴파일을 하면 런타임을 수행할 때 발생할 수 있는 암묵적인 재컴파일을 막을 수 있고, 오버헤드와 컴파일 에러를 미리 방지할 수가 있다. Tibero는 트리거가 참조하는 객체가 무효화된 상태이면 재컴파일을 한 뒤, 트리거를 컴파일한다. 트리거가 성공적으로 컴파일되면 그 트리거는 유효하다.

  – enable_option

구성요소	설명
ENABLE	트리거를 활성화한다.

구성요소	설명
DISABLE	트리거를 비활성화한다.

- 예제

  다음은 트리거를 비활성화하는 예이다.

```
ALTER TRIGGER update_emp_sal DISABLE;
```

7.18. ALTER TYPE

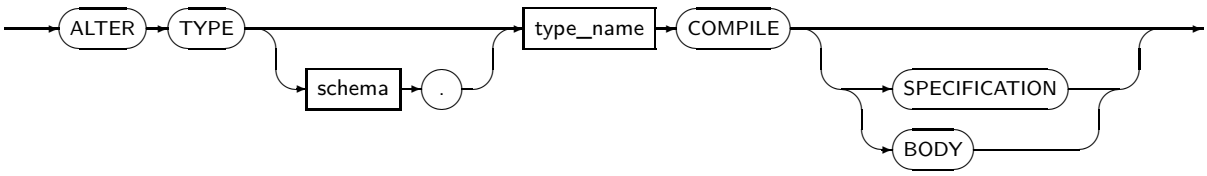
ALTER TYPE를 사용해 명시적으로 사용자 정의 타입을 재컴파일할 수 있다.

명시적으로 재컴파일을 수행하면 런타임 때 발생할 수 있는 암묵적인 재컴파일을 막을 수 있다. 따라서 오버헤드와 컴파일 에러를 미연에 방지할 수가 있다. 부모 객체에 대한 재컴파일과 자식 객체에 대한 무효화의 자세한 내용은 “7.9. ALTER PROCEDURE”를 참고한다.

ALTER TYPE의 세부 내용은 다음과 같다.

- 문법

alter_type



- 특권

  사용자가 소유한 사용자 정의 타입이거나, ALTER ANY TYPE 시스템 특권을 부여받아야 한다.

- 구성요소

구성요소	설명
schema	사용자 정의 타입이 속해 있는 스키마의 이름을 명시한다.
type_name	재컴파일을 진행할 사용자 타입의 이름을 명시한다.
SPECIFICATION/BODY	스펙과 바디 모두를 재컴파일할지 바디만 재컴파일할지를 지정한다. 옵션을 주지 않으면 스펙과 바디 모두를 재컴파일한다.

- 예제

```
CREATE TYPE tibero.two_dimensional_array  AS VARRAY (100) OF
one_dimensional_array;
```

```
/

CREATE TYPE tibero.one_dimensional_array AS VARRAY (100) OF NUMBER;
/

ALTER TYPE tibero.two_dimensional_array COMPILE;


CREATE TYPE object_type AS OBJECT(c1 NUMBER, c2 NUMBER, MEMBER
procedure print_attribute);
/

CREATE TYPE  BODY object_type
AS
    MEMBER PROCEDURE print_attribute
    AS
    BEGIN
            dbms_output.put_line( 'C1:' || c1 );
            dbms_output.put_line( 'C2:' || c2 );
    END;
END;
/

ALTER TYPE object_type COMPILE;

ALTER TYPE object_type COMPILE SPECIFICATION;

ALTER TYPE object_type COMPILE BODY;
```

7.19. ALTER USER

사용자의 정보를 변경한다.

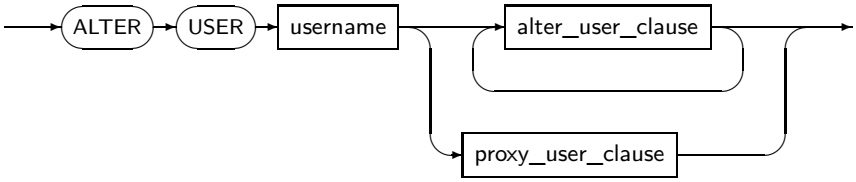
참고

사용자를 생성, 제거하기 위해서는 “7.46. CREATE USER”와 “7.67. DROP USER”의 내용을 참고한다.

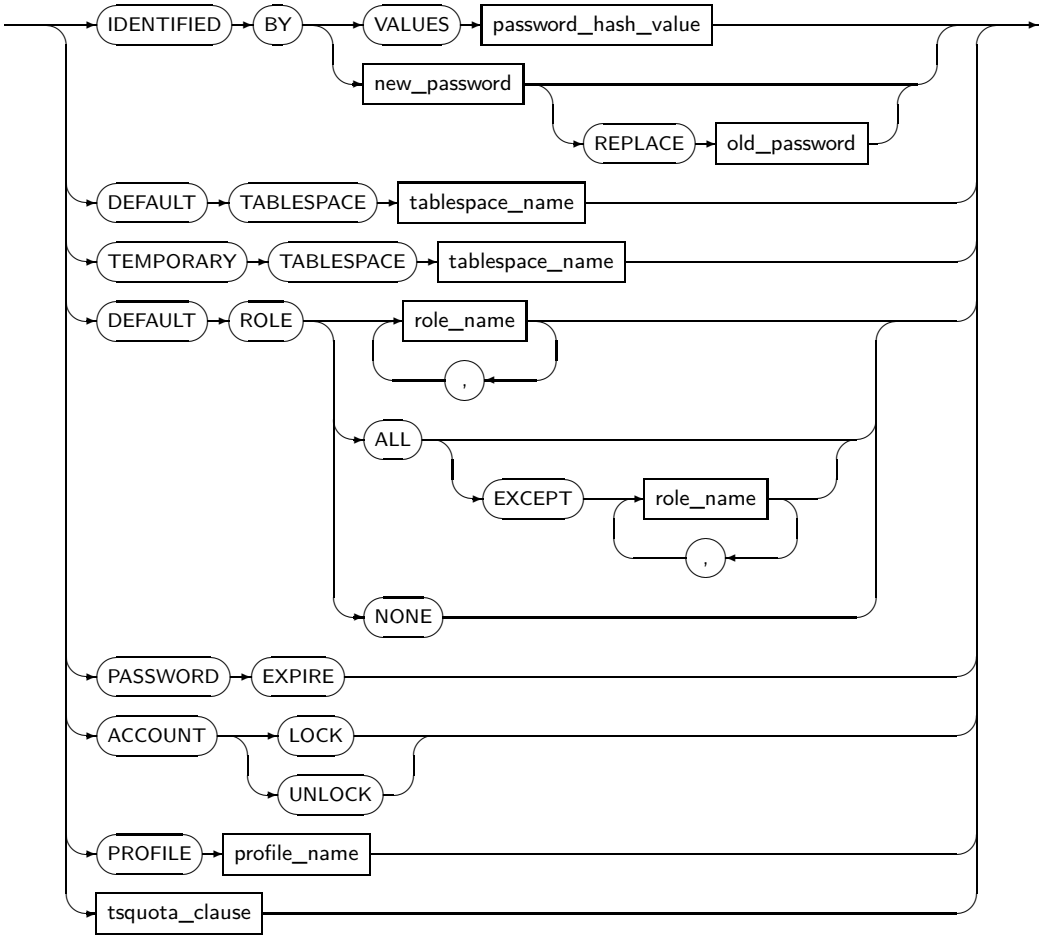
ALTER USER의 세부 내용은 다음과 같다.

- 문법

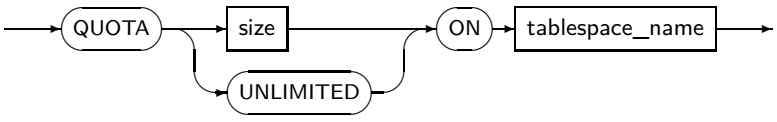
alter_user



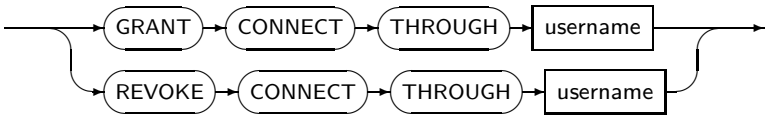
alter_user_clause



tsquota_clause



proxy_user_clause



- 특권

  ALTER USER 시스템 특권이 있어야 한다. 단, 사용자가 자신의 패스워드를 변경할 때는 제외다.

- 구성요소

  - alter_user

구성요소	설명
username	변경할 사용자의 이름이다. 변경할 사용자는 CREATE USER로 미리 만들어져 있어야 한다.
alter_user_clause	변경할 사용자의 정보이다. CREATE USER와는 다르게 생략할 수 없고, 반드시 하나 이상은 있어야 한다.
proxy_user_clause	Proxy user를 사용하기 위해 설정하는 정보이다. CREATE USER 이후에 추가적으로 해당 사용자에 대한 proxy 접근 설정을 원할 때 있어야 한다.

  - alter_user_clause

구성요소	설명
IDENTIFIED BY	사용자의 인증 패스워드를 변경한다. ALTER USER 시스템 특권이 있는 경우에만 다른 사용자의 패스워드를 변경할 수 있다. REPLACE 절은 무시된다.
VALUES 'password_hash_value'	Hash value로 encrypt된 password로 user의 비밀번호를 변경한다. Sys권한에서만 변경이 가능하다.
new_password	변경될 새로운 패스워드를 입력한다. 패스워드는 문자열로 지정하며, 길이는 63bytes까지 가능하다.
REPLACE	사용자가 자신의 인증 패스워드를 변경할 때 IDENTIFIED BY와 함께 사용한다. ALTER USER 시스템 특권이 없는 경우라도 자신의 패스워드는 변경할 수 있다. 이 경우에는 REPLACE를 생략할 수 없고, old_password를 기존 패스워드와 비교하여 같은 경우에만 변경을 허용한다.
old_password	변경 전의 기존 패스워드를 입력한다. 패스워드는 문자열로 지정하며, 길이는 63bytes까지 가능하다.
DEFAULT TABLESPACE	사용자가 사용할 디폴트 테이블 스페이스를 변경한다. CREATE TABLE을 사용하여 테이블을 생성할 때 테이블 스페이스를 명시하지 않으면 디폴트 테이블 스페이스를 사용하게 된다.
DEFAULT ROLE	사용자가 접속했을 때 사용할 디폴트 역할을 결정한다. 사용할 수 있는 역할은 GRANT를 이용하여 명시적으로 부여받은 역할에 한정된다. 즉, 부여받지 않은 역할이나 다른 역할에 의해 부여된 역할은 사용할 수 없다. 역할에 대한 자세한 내용은 “7.38. CREATE ROLE”, “7.71. GRANT”를 참고한다.
PASSWORD EXPIRE	사용자의 패스워드를 사용기간 만료 상태로 변경한다.

구성요소	설명
	패스워드가 사용기간 만료 상태가 되면 해당 사용자가 다음에 접속했을 때 패스워드 사용기간이 만료되었다는 메시지가 출력되고 패스워드를 변경해야 한다.
ACCOUNT LOCK	사용자를 잠금 상태로 변경한다. 사용자가 잠금 상태로 변경되면 해당 사용자는 데이터베이스를 사용할 수 없다.
ACCOUNT UNLOCK	사용자를 잠금 해제 상태로 변경한다.

디폴트 역할을 지정하는 방법은 다음과 같이 3가지가 있다.

번호	옵션	설명
1	role_name	부여받은 역할 중에 디폴트 역할로 지정할 역할을 나열하는 방식이다. 이는 몇 가지 역할만을 사용하고자 할 때 유용하다.
2	ALL (EXCEPT)	ALL은 거의 대부분의 역할을 디폴트 역할로 사용하고자 할 때 유용하다. 몇몇 역할을 제외한 나머지 역할 모두를 디폴트 역할로 하고자 한다면, ALL 뒤에 EXCEPT를 사용하여 제외하고자 하는 역할을 명시할 수 있다.
3	NONE	NONE은 디폴트 역할을 모두 끄고 필요한 역할만 활성화시켜서 사용하고자 할 때 유용하다. 부여받은 역할 중 디폴트 역할에서 제외된 역할은 SET ROLE 문을 사용하여 동적으로 켜거나 끌 수 있다. 자세한 내용은 “9.7. SET ROLE”을 참고한다.

– tsquota_clause

구성요소	설명
QUOTA size ON tablespace_name	사용자가 사용할 테이블 스페이스의 크기를 지정 값 만큼 제한한다.
QUOTA UNLIMITED ON tablespace_name	사용자가 사용할 테이블 스페이스의 크기를 제한하지 않는다.

– proxy_user_clause

구성요소	설명
GRANT	사용자에 proxy 접속 권한을 줄 때 사용한다. 해당 권한을 받은 proxy user에 대해서만 사용자로 접속 가능하다.
REVOKE	사용자에 proxy 접속 권한을 철회할 때 사용한다. 해당 권한이 철회된 proxy user는 사용자로 접속이 불가능하게된다.
CONNECT THROUGH	Proxy 접속 기능을 사용할 때 사용하는 구문이다. CONNECT THROUGH 구문 다음에 나오는 username이 proxy user가 된다.

- 예제

다음은 **IDENTIFIED BY**를 사용해 사용자의 패스워드를 변경하는 예이다.

```
SQL> CONN sys/tibero
Connected.

SQL> ALTER USER u1 IDENTIFIED BY 'p1';
User altered.

SQL> CONN u1/p1
Connected.

SQL> ALTER USER u1 IDENTIFIED BY 'p2';
TBR-7053: Invalid old password.

SQL> ALTER USER u1 IDENTIFIED BY 'p2' REPLACE 'p1';
User altered.
```

위의 예에서, 처음에는 ALTER USER 시스템 특권이 있는 사용자이기 때문에, 사용자 u1의 패스워드를 REPLACE 절 없이 변경할 수 있었다. 하지만, 사용자 u1의 경우에는 ALTER USER 시스템 특권이 없기 때문에 자기 자신의 패스워드만 변경할 수 있고, 그 경우에도 REPLACE가 반드시 있어야만 변경할 수 있다.

다음은 **DEFAULT TABLESPACE**를 사용해 디폴트 테이블 스페이스를 변경하는 예이다.

```
SQL> SELECT username, default_tablespace
     FROM dba_users
     WHERE username='U1';

USERNAME                       DEFAULT_TABLESPACE
------------------------------ ------------------------------
U1                             SYSTEM

1 row selected.

SQL> ALTER USER u1 DEFAULT TABLESPACE t1;
Altered.

SQL> SELECT username, default_tablespace
     FROM dba_users
     WHERE username='U1';

USERNAME                       DEFAULT_TABLESPACE
------------------------------ ------------------------------
U1                             T1

1 row selected.
```

다음은 **DEFAULT ROLE**을 사용해 디폴트 역할을 변경하는 예이다.

```
SQL> CREATE ROLE a;
Role created.

SQL> CREATE ROLE b;
Role created.

SQL> CREATE ROLE c;
Role created.

SQL> GRANT CREATE SESSION TO a;
Granted.

SQL> GRANT a TO b;
Granted.

SQL> GRANT b, c TO u1;
Granted.

SQL> SELECT grantee, granted_role, default_role
     FROM dba_role_privs
     WHERE grantee='U1';

GRANTEE GRANTED_ROLE DEF
------- ------------ ---
U1      B            YES
U1      C            YES

2 rows selected.

SQL> ALTER USER u1 DEFAULT ROLE a;
TBR-7172: cannot enable role 'a' you have not been granted

SQL> GRANT a TO u1;
granted.

SQL> ALTER USER u1 DEFAULT ROLE NONE;
User altered.

SQL> SELECT grantee, granted_role, default_role
     FROM dba_role_privs
     WHERE grantee='U1';

GRANTEE GRANTED_ROLE DEF
------- ------------ ---
```

```
U1      B            NO
U1      C            NO
U1      A            NO

3 rows selected.

SQL> ALTER USER u1 DEFAULT ROLE ALL EXCEPT a, c;
User altered.

SQL> SELECT grantee, granted_role, default_role
     FROM dba_role_privs
     WHERE grantee='U1';

GRANTEE GRANTED_ROLE DEF
------- ------------ ---
U1      B            YES
U1      C            NO
U1      A            NO

3 rows selected.
```

위의 예를 보면, 처음에 CREATE ROLE 문을 사용하여 A, B, C 이렇게 3개의 역할을 만든다. 그리고 GRANT 문을 사용하여 역할 B에게 역할 A의 특권을 부여한 뒤, 역할 B와 C를 사용자 u1에게 부여한다. 그 상태에서 사용자 U1에게 부여된 역할을 조회한다. 그리고 사용자 u1의 디폴트 역할을 역할 B에 부여된 역할 A로 지정하려 하면 사용자 u1이 역할 A를 직접 부여받은 적이 없기 때문에 에러가 발생하게 된다.

사용자 u1에게 역할 A까지 부여한 뒤 DEFAULT ROLE NONE을 사용해 디폴트 역할을 모두 제거한다. 그리고 다시 사용자 u1에게 부여된 역할을 조회하면 모든 디폴트 역할이 꺼져있음을 확인할 수 있다. 마지막으로 DEFAULT ROLE ALL을 사용해 보유한 역할을 모두 디폴트 역할로 설정한다. EXCEPT를 같이 사용하여 역할 A, C를 제외한 나머지 모든 역할을 디폴트 역할로 지정하고 있다.

다음은 **PASSWORD EXPIRE**를 사용해 사용자의 패스워드를 만료시키는 예이다.

```
SQL> ALTER USER u1 PASSWORD EXPIRE;
User altered.

SQL> CONN u1/p1
TBR-17002 :  password expired.
New password : ***
Retype new password : ***
Password changed.
Connected.
```

다음은 **ACCOUNT**를 사용해 사용자를 잠금 상태와 잠금 해제 상태로 변경하는 예이다.

```
SQL> ALTER USER u1 ACCOUNT LOCK;
User altered.

SQL> SELECT username, account_status
     FROM dba_users
     WHERE username='U1';

USERNAME                       ACCOUNT_STATUS
------------------------------ ----------------
U1                             LOCKED

1 row selected.

SQL> CONN u1/p1;
TBR-17006: account is locked.

SQL> CONN sys/tibero
Connected.

SQL> ALTER USER u1 ACCOUNT UNLOCK;
User altered.

SQL> SELECT username, account_status
     FROM dba_users
     WHERE username='U1';

USERNAME                       ACCOUNT_STATUS
------------------------------ ----------------
U1                             OPEN

1 row selected.

SQL> CONN U1/p1;
Connected.
```

7.20. ALTER VIEW

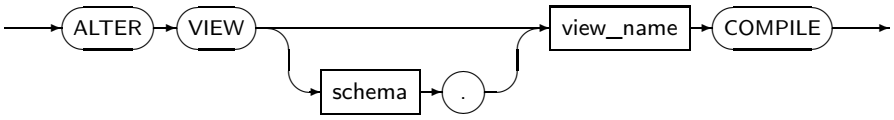
무효화된 뷰를 다시 컴파일한다. 이 명령어를 사용하기 위하여 자신의 스키마에 포함된 뷰이거나 ALTER ANY TABLE 시스템 특권을 부여 받고 있어야 한다.

ALTER TABLE 명령을 이용하여 테이블을 변경한 경우 그 테이블에 기반한 모든 뷰는 무효화된다. 무효화된 뷰는 SQL 문장 내에서 사용될 때에 자동으로 다시 컴파일된다. ALTER VIEW 명령어는 SQL 문장 내에서 사용되기 전에 미리 다시 컴파일하여 성능 저하를 막고 가능한 문제를 미리 발견하기 위하여 사용한다.

ALTER VIEW의 세부 내용은 다음과 같다.

- 문법

alter_view



- 특권

  ALTER VIEW로 뷰를 갱신하기 위해서는 자기 스키마에 속한 뷰거나 ALTER ANY VIEW 시스템 특권이 필요하다.

- 구성요소

구성요소	설명
schema	다시 컴파일할 뷰를 포함하는 스키마의 이름이다. 만약 생략하면 현재 사용자의 스키마를 가정한다.
view_name	뷰의 이름을 설정한다.
COMPILE	다시 컴파일하도록 한다. 유효하지 않은 뷰를 다시 유효하게 만들어 주거나, 참조하는 뷰나 테이블의 정의가 바뀌었을 경우 알맞게 뷰의 정의를 변경한다.

- 예제

```
ALTER VIEW tibero.MANAGER COMPILE;
```

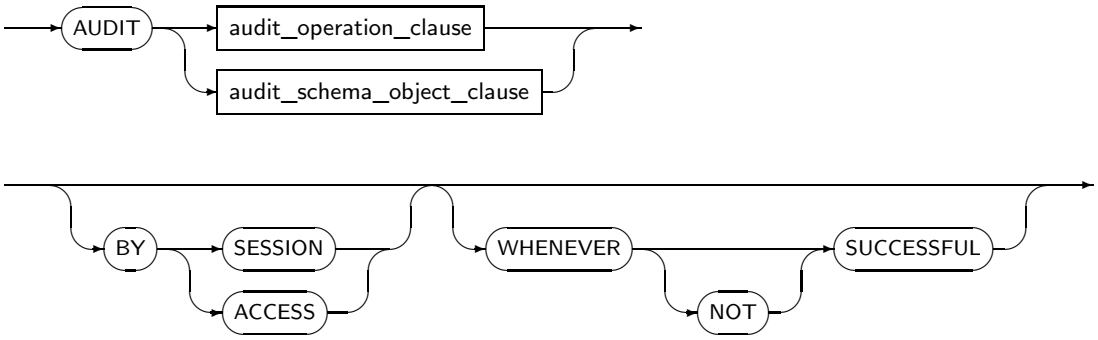
7.21. AUDIT

사용자가 시스템 특권 또는 스키마 객체 특권을 사용하는 것을 감사한다. 감사하고 있는 특권을 해제하기 위해서는 “7.72. NOAUDIT”의 내용을 참고한다. 특권에 대한 자세한 내용은 "Tibero 관리자 안내서"를 참고한다.

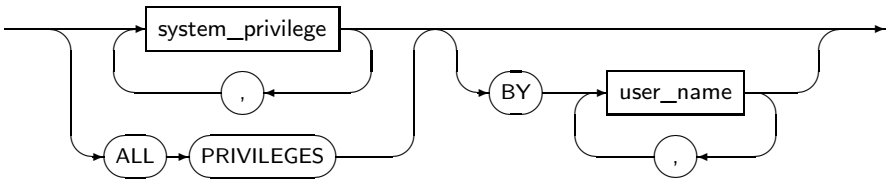
AUDIT의 세부 내용은 다음과 같다.

- 문법

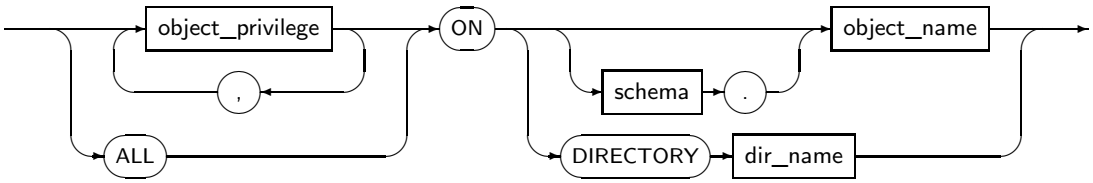
audit



audit_operation_clause



audit_schema_object_clause



- 특권
  - 시스템 특권을 감사하기 위해서는 AUDIT SYSTEM 시스템 특권을 부여받아야 한다.
  - 다른 사용자가 소유한 스키마의 객체 또는 디렉터리 객체를 감사하기 위해서는 AUDIT ANY 시스템 특권을 부여받아야 한다.
  - 감사한 내용을 기록하기 위해서는 $TB_SID.tip 파일에 `AUDIT_TRAIL` 파라미터가 NONE이 아닌 다른 옵션으로 설정해야 한다. NONE이 설정되어 있으면, 감사를 하더라도 기록은 하지 않는다.
  - **SYS 사용자**는 기본으로 감사되지 않는다. SYS 사용자를 감사하기 위해서는 $TB_SID.tip 파일의 `AUDIT_SYS_OPERATIONS` 파라미터를 Y로 설정한다. 그러면 SYS 사용자가 수행한 동작이 모두 기록된다.
- 구성요소
  - audit

구성요소	설명
audit_operation_clause	시스템 특권을 감사한다.
audit_schema_object_clause	특정 객체에 대한 스키마 객체 특권을 감사한다. 감사할 수 있는 객체의 종류는 테이블, 뷰, 시퀀스, 프러시저, 디렉터리 등이다.

구성요소	설명
BY SESSION	감사 대상이 되는 권한을 사용했을 경우 이를 어떻게 기록할 것인지를 지정한다. BY SESSION은 같은 위반을 세션 당 한 번만 기록한다.
BY ACCESS	BY ACCESS는 같은 위반이라도 매번 기록한다. $TB_SID.tip 파일의 AUDIT_TRAIL 파라미터의 값이 OS이면 BY SESSION은 무시되고 항상 BY ACCESS로 동작한다. 생략하면 기본값은 BY ACCESS이다.
WHENEVER SUCCESSFUL	감사 대상이 되는 권한을 사용했을 경우 해당 명령의 성공이나 실패 여부에 따라 어떻게 기록할 것인지를 지정한다. WHENEVER SUCCESSFUL은 명령이 성공했을 때만 기록한다. 지정하지 않으면 성공이나 실패에 관계없이 모두 기록한다.
WHENEVER NOT SUCCESSFUL	WHENEVER NOT SUCCESSFUL은 명령이 실패했을 때만 기록을 저장한다.

– audit_operation_clause

구성요소	설명
system_privilege	감사할 시스템 특권을 지정한다. 시스템 특권의 종류는 "7.71. GRANT"의 시스템 특권을 참고한다.
ALL PRIVILEGES	모든 시스템 특권을 감사한다.
BY user_name	감사할 사용자를 지정한다. 지정하지 않으면 모든 사용자에 적용한다.

– audit_schema_object_clause

구성요소	설명
object_privilege	감사할 스키마 객체 특권을 지정한다. 스키마 객체 특권의 종류는 "7.71. GRANT"의 시스템 특권을 참고한다.
ALL	해당 객체에 사용할 수 있는 모든 스키마 객체 특권을 감사한다. 대상 객체의 종류에 따라 사용할 수 있는 스키마 객체 특권은 "7.71. GRANT"의 시스템 특권을 참고한다.
ON	스키마 객체 특권을 감사할 대상이 되는 객체를 지정한다.
schema	객체가 속해 있는 스키마의 이름을 지정한다. 스키마 이름을 지정하지 않으면, 자신의 스키마 객체에서 해당 이름을 찾는다.
object_name	디렉터리가 아닌 객체의 이름을 지정한다.
DIRECTORY dir_name	디렉터리 객체의 이름을 지정한다.

- 예제

  – audit

    다음은 **BY SESSION**을 사용해 감사 기록 여부를 설정하는 예이다.

    ```
    SQL> AUDIT delete ON t BY SESSION WHENEVER SUCCESSFUL;
    Audited.
    ```

  – audit_operation_clause

    다음은 **BY user_name**을 사용해 감사할 사용자를 지정하는 예이다.

    ```
    SQL> AUDIT create table BY tibero;
    Audited.
    ```

  – audit_schema_object_clause

    다음은 **ON**을 사용해 스키마 객체 특권을 감사할 대상 객체를 지정하는 예이다.

    ```
    SQL> AUDIT insert ON t;
    Audited.
    ```

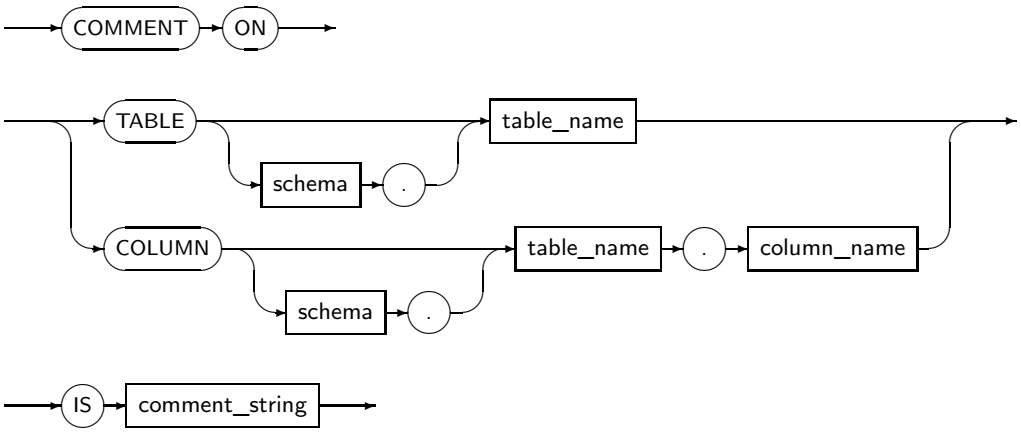
7.22. COMMENT

테이블, 뷰 또는 이에 속한 특정 컬럼에 주석을 삽입한다.

COMMENT의 세부 내용은 다음과 같다.

- 문법

comment



- 특권

  사용자가 소유한 스키마 객체에 COMMENT 문을 실행하는 경우 별다른 특권이 필요하지 않다. 하지만, 다른 사용자에게 속한 스키마 객체라면 COMMENT ANY TABLE 시스템 특권이 필요하다.

- 구성요소

구성요소	설명
TABLE	테이블이나 뷰에 주석을 삽입할 때 사용한다. 삽입된 주석의 내용은 나중에 DBA_TAB_COMMENTS, USER_TAB_COMMENTS, ALL_TAB_COMMENTS 정적 뷰로 확인할 수 있다.
COLUMN	테이블, 뷰에 속한 컬럼에 주석을 삽입하고 싶을 때 사용한다. 삽입한 주석의 내용은 나중에 DBA_COL_COMMENTS, USER_COL_COMMENTS, ALL_COL_COMMENTS 정적 뷰로 확인할 수 있다.
comment_string	주석의 내용이다. 주석은 VARCHAR 타입으로 저장되므로, 최대 4000자까지 입력할 수 있다.

- 예제

다음은 **COMMENT**를 사용해 주석을 삽입하는 예이다.

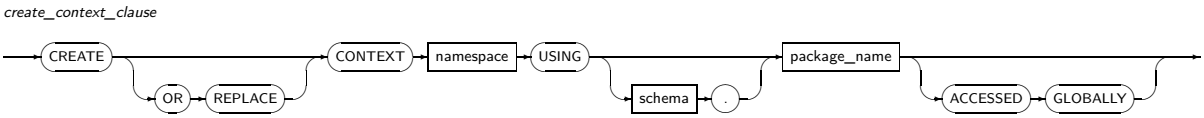
```
SQL> CREATE TABLE t1 (col1 NUMBER, col2 NUMBER);
Table created.

SQL> COMMENT ON TABLE t1 IS '이것은 예제 테이블이다.';
Commented.

SQL> SELECT * FROM user_tab_comments WHERE table_name = 'T1';

TABLE_NAME                     TABLE_TYP   COMMENTS
------------------------------ ---------   -------------------------
T1                             TABLE       이것은 예제 테이블이다.

1 row selected.

SQL> COMMENT ON COLUMN t1.col1 IS '이것은 첫 번째 컬럼이다.';
Commented.

SQL> SELECT * FROM user_col_comments WHERE table_name = 'T1';

TABLE_NAME                     COLUMN_NAME             COMMENTS
------------------------------ ---------------------- -------------------------
T1                             COL1                   이것은 첫 번째 컬럼이다.

1 row selected.
```

7.23. CREATE CONTEXT

문맥 네임스페이스를 생성(혹은 수정)하고 해당 네임스페이스의 문맥을 조작할 수 있는 패키지를 지정한다. 지정된 패키지에서 DBMS_SESSION.SET_CONTEXT 프로시져를 사용해 해당 네임스페이스의 속성 값들을 설정할 수 있다.

CREATE CONTEXT의 세부 내용은 다음과 같다.

- 문법



- 특권

  문맥 네임스페이스를 생성하기 위해서는 CREATE ANY CONTEXT 시스템 특권이 있어야만 한다.

- 구성요소

  – create_context_clause

구성요소	설명
namespace	생성(하거나 수정)할 문맥 네임스페이스의 이름을 명시한다.
schema	문맥을 조작할 패키지의 스키마를 명시한다. 스키마를 생략하면 현재 스키마를 가정한다.
package_name	문맥을 조작할 패키지의 이름을 명시한다. 패키지가 현재 존재하지 않더라도 해당 DDL 은 정상 수행된다.
ACCESSED GLOBALLY	이것을 명시하면 데이터베이스 인스턴스가 살아 있는 한 모든 세션에서 해당 문맥에 접근하여 내용을 읽을 수가 있다.

- 예제

  다음은 CREATE CONTEXT를 사용해 문맥을 생성하는 예이다. 보안 정책을 구현한 패키지는 pol_pkg 이며, 문맥 네임스페이스의 이름은 sec_ctx 이다.

```
CREATE CONTEXT sec_ctx USING pol_pkg;
```

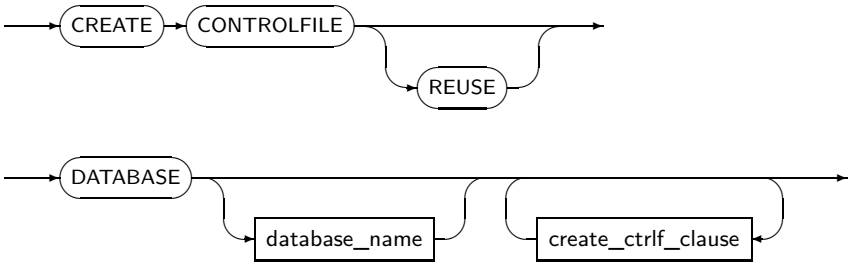
7.24. CREATE CONTROLFILE

기존에 존재하는 데이터 파일과 로그 파일의 정보를 바탕으로 컨트롤 파일을 새로 생성한다. 최대 데이터 파일의 개수를 변경하는 경우나 컨트롤 파일이 손상된 경우에 사용하며, NOMOUNT 모드에서만 사용할 수 있다. 일반적으로 ALTER DATABASE BACKUP CONTROLFILE TO TRACE를 이용하여 컨트롤 파일의 생성문을 백업한다.

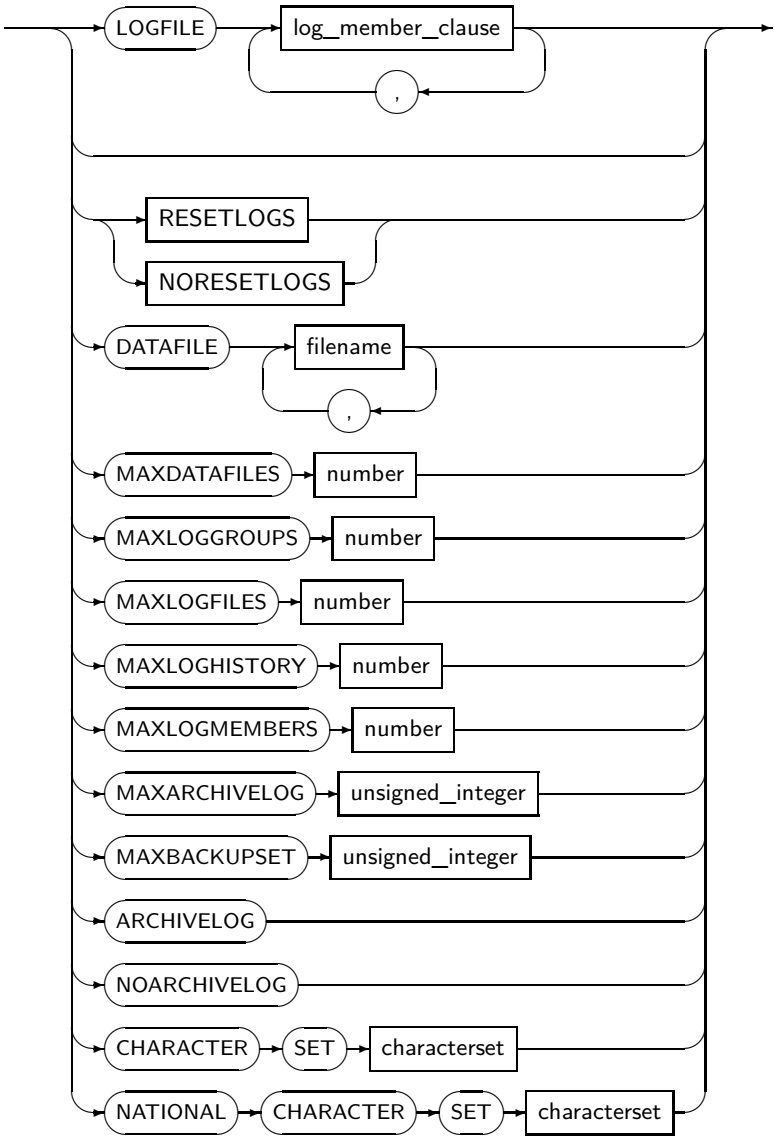
CREATE CONTROLFILE의 세부 내용은 다음과 같다.

- 문법

create_controlfile



create_ctrlf_clause



- 특권

Tibero를 NOMOUNT 모드로 기동하면, SYS 사용자만 CREATE CONTROLFILE 문을 실행할 수 있다.

- 구성요소

  – create_controlfile

구성요소	설명
REUSE	기존에 존재하는 컨트롤 파일에 덮어쓰고자 할 경우에 사용한다.
DATABASE	컨트롤 파일을 생성하고자 하는 대상 데이터베이스를 명시한다.
database_name	대상 데이터베이스의 이름을 명시한다.
create_ctrlf_clasuse	컨트롤 파일의 다양한 속성을 설정한다.

  – create_ctrlf_clasuse

구성요소	설명
LOGFILE	온라인 로그 파일을 지정한다. 온라인 로그 파일은 두 개 이상의 로그 그룹을 정의하고, 그룹별로 하나 이상의 멤버를 지정해야 한다.
log_member_clause	로그 그룹에 번호를 지정할 수 있다. 지정하지 않으면 0부터 차례대로 지정된다.
RESETLOGS	RESETLOGS를 명시하면 기존 로그 파일을 무시하고 로그를 초기화한다. RESETLOGS를 명시하지 않으면 NORESETLOGS로 간주한다.
NORESETLOGS	NORESETLOGS를 명시하면 기존의 유효한 로그 파일을 계속 사용한다. 로그 파일이 반드시 존재해야 하고, 계속 사용하는 Redo 로그 파일이어야 한다.
MAXDATAFILES	데이터베이스에서 사용할 최대 데이터 파일의 개수를 제한한다. MAXDATAFILES에 지정된 값만큼 컨트롤 파일에 데이터 파일의 공간을 만든다. 따라서 MAXDATAFILES의 값을 초과하는 데이터 파일의 개수를 정의할 수 없다. MAXDATAFILES의 값을 변경하려면 컨트롤 파일을 재생성해야 한다. MAXDATAFILES의 값이 클수록 많은 메모리가 요구되므로, 이 점을 고려해서 값을 설정해야 한다. (기본값: 100, 최솟값: 10, 최댓값: 65533)
MAXLOGGROUPS	로그 그룹의 최댓값을 제한한다. 컨트롤 파일에 로그 그룹을 위한 공간을 확보하기 위해 필요한 값이다. MAXLOGGROUPS의 값을 변경하려면 컨트롤 파일을 재생성해야 한다. (기본값: 255, 최댓값: 255)

구성요소	설명
MAXLOGFILES	MAXLOGGROUPS과 같은 의미이며, 사용자의 편의를 위해 제공된다. MAXLOGGROUPS와 MAXLOGFILES 중 하나만 사용해야 한다.
MAXLOGMEMBERS	로그 그룹 내의 로그 파일의 최대 개수를 제한한다. (기본값: 8, 최댓값: 8) MAXLOGMEMBERS의 값을 변경하려면 컨트롤 파일을 재생성해야 한다.
MAXARCHIVELOG	컨트롤 파일에 기록될 수 있는 최대 아카이브 로그의 갯수를 지정한다. 이 갯수를 넘어가는 아카이브 로그 파일이 생성되면 가장 오래된 아카이브 로그의 기록부터 덮어쓴다. (기본값: 500, 최댓값: 64000)
MAXBACKUPSET	컨트롤 파일에 기록될 수 있는 최대 백업셋의 갯수를 지정한다. MAXBACKUPSET의 값을 변경하려면 컨트롤 파일을 재생성해야 한다. (기본값: 500, 최댓값: 5000)
MAXLOGHISTORY	컨트롤 파일에 기록될 수 있는 로그 파일 스위치 내역의 최대 갯수를 지정한다. 이 이상으로 로그 파일 스위치가 발생되면 가장 오래된 기록부터 덮어쓴다. (기본값: 500, 최댓값: 64000)
ARCHIVELOG	ARCHIVELOG 모드로 설정한다. 로그 그룹은 순환적으로 재사용된다. 로그 그룹이 한 번 사용되고 나서 재사용되기 전에 시스템에 의해 아카이브될 수 있다.
NOARCHIVELOG	NOARCHIVELOG 모드로 설정한다.

- 예제

  다음은 **CREATE CONTROLFILE**을 사용해 컨트롤 파일을 생성하는 예이다.

```
CREATE CONTROLFILE REUSE DATABASE "t7db"
  LOGFILE
    GROUP 0 (
              '/disk2/log101.log',
              '/disk3/log102.log'
            ) SIZE 1M,
    GROUP 1 (
              '/disk1/log103.log',
              '/disk2/log104.log',
              '/disk3/log105.log'
            ) SIZE 2M
    NORESETLOGS
  DATAFILE
    '/disk1/system01.dbf', '/disk2/undo004.dbf'
```

```
ARCHIVELOG
MAXLOGFILES 30
MAXLOGMEMBERS 8
MAXARCHIVELOG 500
MAXBACKUPSET 500
MAXDATAFILES 200;
```

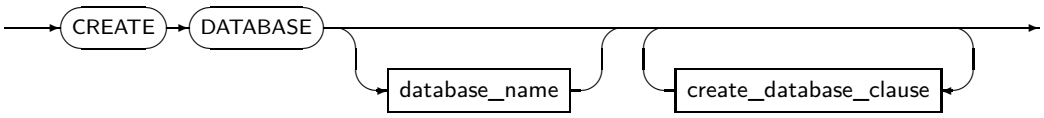
7.25. CREATE DATABASE

데이터베이스를 생성한다.

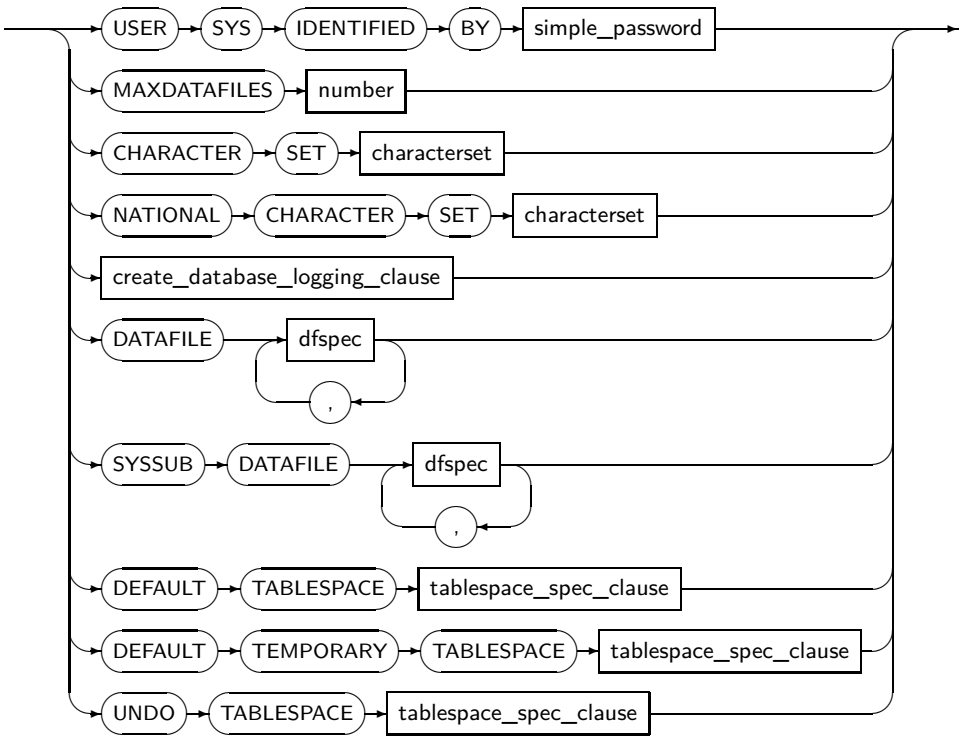
CREATE DATABASE의 세부 내용은 다음과 같다.

- 문법

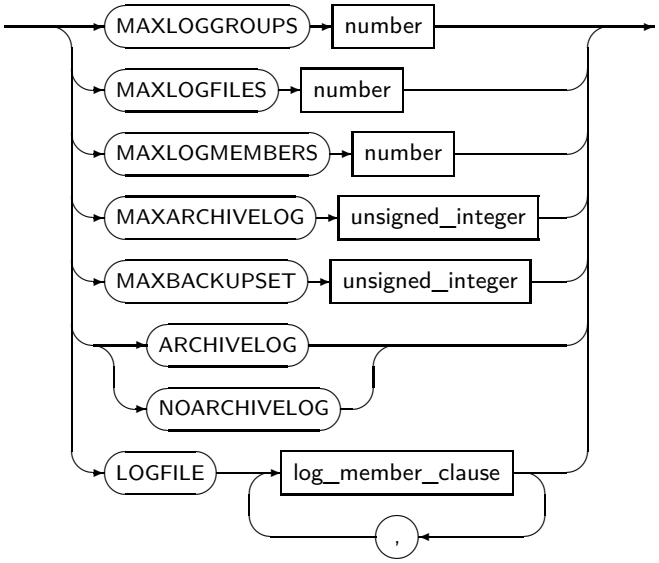
create_database



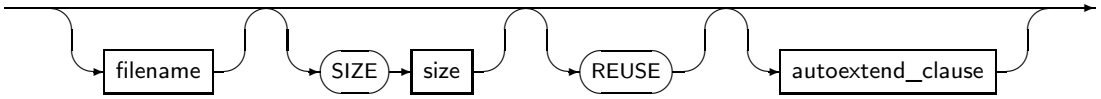
create_database_clause



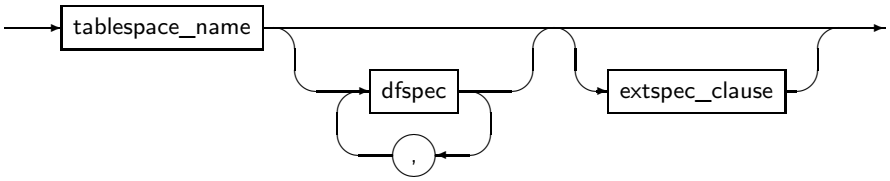
create_database_logging_clause



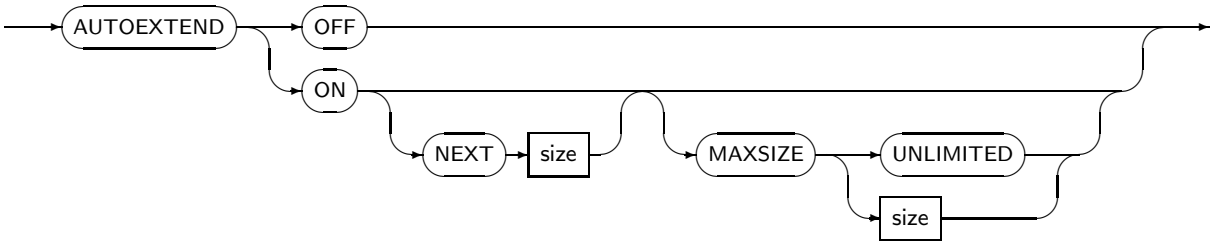
dfspec



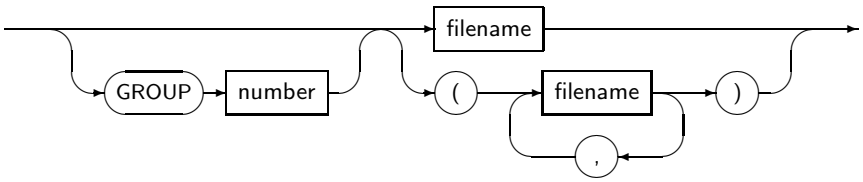
tablespace_spec_clause

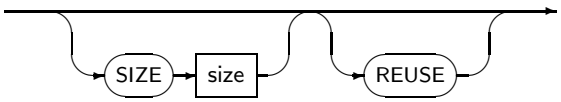


autoextend_clause



log_member_clause





- 특권

  Tibero를 NOMOUNT 모드로 기동하면, SYS 사용자만 CREATE DATABASE 문을 실행할 수 있다.

- 구성요소

  – create_database

구성요소	설명
database_name	$TB_SID.tip 파일의 DB_NAME 파라미터 값과 같아야 하며 생략할 수 있다. $TB_SID.tip 파일에 지정된 컨트롤 파일이 이미 존재하면 에러가 발생한다. 지정하지 않으면, 기본값은 환경설정 파일의 TB_SID와 동일하다.
create_database_clause	새로운 데이터베이스를 생성할 때 사용한다. NOMOUNT 모드에서만 사용할 수 있다.

  – create_database_clause

구성요소	설명
USER SYS IDENTIFIED BY	SYS 사용자의 패스워드를 설정하는 데 사용된다. (기본값: tibero) 설정된 패스워드는 ALTER USER 문을 통해 변경할 수 있다. 자세한 내용은 “7.19. ALTER USER”를 참고한다.
MAXDATAFILES	데이터베이스에서 사용할 최대 데이터 파일의 개수를 제한한다. MAXDATAFILES에 지정된 값만큼 컨트롤 파일에 데이터 파일의 공간을 만든다. 따라서 MAXDATAFILES의 값을 초과하는 데이터 파일의 개수를 정의할 수 없다. MAXDATAFILES의 값을 변경하려면 컨트롤 파일을 재생성해야 한다. MAXDATAFILES의 값이 클수록 많은 메모리가 요구되므로, 이점을 고려해서 값을 설정해야 한다. (기본값: 100, 최솟값: 10, 최댓값: 65533)
CHARACTER SET	데이터베이스에 디폴트로 사용할 문자 집합을 지정할 수 있다. 지정할 수 있는 문자 집합은 다음과 같다. – ASCII: ASCII 7-bit 영어 – EUCKR: EUC 16-bit 한국어 – MSWIN949: MS Windows 코드 페이지 949 한국어 – UTF8: 24-bit 국제 표준 다국어 (기본값) – SJIS: Shift-JIS 16-bit 일본어 – JA16SJIS: MS Windows 코드 페이지 932 일본어

구성요소	설명
	– JA16SJISTILDE: 전각물결문자를 포함하는 MS Windows 코드 페이지 932 일본어 – JA16EUC: EUC 24-bit 일본어 – JA16EUCTILDE: 전각물결문자를 포함하는 EUC 24-bit 일본어 – VN8VN3: VN3 8-bit 베트남어 – GBK: MS Windows 코드 페이지 936 중국어 – WE8MSWIN1252: MS Windows 코드 페이지 1252 서유럽어 – EL8MSWIN1253: MS Windows 코드 페이지 1253 그리스어 – AR8MSWIN1256: MS Windows 코드 페이지 1256 아랍어 – ZHT16HKSCS: HKSCS2001 홍콩어와 MS Windows 코드 페이지 950 중국어 – WE8ISO8859P1: ISO8859-1 서유럽어 – EE8ISO8859P2: ISO8859-2 동유럽어 – AR8ISO8859P6: ISO8859-6 아랍어 – EL8ISO8859P7: ISO8859-7 그리스어 – WE8ISO8859P9: ISO8859-9 서유럽어(터키어) – WE8ISO8859P15: ISO8859-15 서유럽어 – CL8MSWIN1251: MS Windows 코드 페이지 1251 키릴문자(러시아어, 불가리아어) – CL8KOI8R: KOI8-R 키릴문자(러시아어, 불가리아어) – CL8ISO8859P5: ISO8859-5 키릴문자(러시아어, 불가리아어) – EUCTW: EUC 중국어 번체 – GB18030: 중국 정부 표준 중국어 – IW8ISO8859P8: ISO 8859-8 라틴어, 히브리어 – RU8PC866: IBM-PC 코드 페이지 866 8-bit 키릴문자 (러시아어) – SJISTILDE: 전각물결문자를 포함하는 SHIFT-JIS 16-bit 일본어 – TH8TISASCII: 태국 산업 표준 620-2533-ASCII 8-bit 태국어 – UTF16: 16/32-bit 가변 길이 국제 표준 다국어

구성요소	설명
	– ZHT16BIG5: BIG5 16-bit 중국어 – ZHT16MSWIN950: MS Windows 코드 페이지 950 중국어
NATIONAL CHARACTER SET	NCHAR, NCLOB, NVARCHAR2로 정의된 컬럼을 저장할 때 사용할 국가별 문자 집합을 지정한다. 지정할 수 있는 문자 집합에는 UTF8, UTF16 등이 있다. – UTF16: 16-bit 국제 표준 다국어 (기본값)
DATAFILE	시스템 테이블 스페이스의 데이터 파일을 정의한다. 하나 이상의 데이터 파일이 정의되어야 한다. 지정하지 않으면 '$TB_HOME/database/$TB_SID/system001.dbf' 파일 하나가 테이블 스페이스에 추가된다. 'system001.dbf' 파일이 이미 존재하는 경우 'system002.dbf, system003.dbf, ...' 순으로 존재하지 않는 파일의 이름으로 생성된다($TB_HOME, $TB_SID는 각각 환경변수 TB_HOME, TB_SID이다).
DEFAULT TABLESPACE	사용자의 디폴트 테이블 스페이스를 정의할 때 지정한다. 지정하지 않으면 일반 사용자도 SYSTEM 테이블 스페이스를 디폴트 테이블 스페이스로 사용한다.
DEFAULT TEMPORARY	디폴트 임시 테이블 스페이스를 정의한다. 지정하지 않으면 'TEMP'라는 이름의 테이블 스페이스가 생성되고, '$TB_HOME/database/$TB_SID/temp001.dbf' 파일이 추가된다. 'temp001.dbf' 파일이 이미 존재할 경우 'temp002.dbf, temp003.dbf, ...' 순으로 존재하지 않는 파일의 이름으로 생성된다.
UNDO TABLESPACE	Undo 테이블 스페이스를 정의한다. Undo 테이블 스페이스는 반드시 정의되어야 하며 하나만 정의할 수 있다. 지정하지 않으면 'UNDO'라는 이름의 테이블 스페이스가 생성되고, '$TB_HOME/database/$TB_SID/undo001.dbf' 파일이 추가된다. 'undo001.dbf ' 파일이 이미 존재할 경우 'undo002.dbf, undo003.dbf, ...' 순으로 존재하지 않는 파일의 이름으로 생성된다.

– create_database_logging_clause

구성요소	설명
LOGFILE	온라인 로그 파일을 지정한다.

구성요소	설명
	온라인 로그 파일은 두 개 이상의 로그 그룹을 정의하고, 그룹별로 하나 이상의 멤버를 지정해야 한다.
MAXLOGGROUPS	로그 그룹의 최댓값을 제한한다. 컨트롤 파일의 로그 그룹을 위한 공간을 확보하기 위해 필요한 값이다. 해당 항목의 값을 변경하기 위해서는 컨트롤 파일을 재생성해야 한다. (기본값: 255, 최댓값: 255) (LOGGROUP_CNTMAX_PER_DATABASE)
MAXLOGFILES	MAXLOGGROUPS과 같은 의미이며, 사용자의 편의를 위해 제공된다. MAXLOGGROUPS과 MAXLOGFILES 중 하나만 사용해야 한다.
MAXLOGMEMBERS	로그 그룹 내의 로그 파일의 최대 개수를 제한한다. 해당 항목의 값을 변경하려면 컨트롤 파일을 재생성해야 한다. (기본값: 8, 최댓값: 8) (LOGMEMBER_CNTMAX_PER_LGGGROUP)
ARCHIVELOG	ARCHIVELOG 모드로 설정한다. 로그 그룹은 순환적으로 재사용된다. 로그 그룹이 한 번 사용되고 나서 재사용되기 전에 시스템에 의해 아카이브 될 수 있다.
MAXARCHIVELOG	컨트롤 파일에 기록될 수 있는 최대 아카이브 로그의 갯수를 지정한다. 이 갯수를 넘어가는 아카이브 로그 파일이 생성되면 가장 오래된 아카이브 로그의 기록부터 덮어쓰게 된다. (기본값: 500, 최댓값: 64000)
MAXBACKUPSET	컨트롤 파일에 기록될 수 있는 최대 백업셋의 갯수를 지정한다. MAXBACKUPSET의 값을 변경하려면 컨트롤 파일을 재생성해야 한다. (기본값: 500, 최댓값: 5000)
MAXLOGHISTORY	컨트롤 파일에 기록될 수 있는 로그 파일 스위치 내역의 최대 갯수를 지정한다. 이 이상으로 로그 파일 스위치가 발생되면 가장 오래된 기록부터 덮어쓰게 된다. (기본값: 500, 최댓값: 64000)
NOARCHIVELOG	NOARCHIVELOG 모드로 설정한다.

– dfspec

구성요소	설명
filename	filename은 절대경로와 상대경로로 지정할 수 있다. 상대경로는 '$TB_HOME/database/$TB_SID/'로 지정된다.

구성요소	설명
SIZE	byte 단위로 파일 크기를 정의한다. M(megabyte), K(kilobyte), G(gigabyte) 등의 기호를 사용하여 보다 큰 크기의 파일을 쉽게 정의할 수 있다.
REUSE	기존의 파일이 있을 때 덮어쓸지의 여부를 지정한다. REUSE가 없으면 같은 이름의 파일이 존재하면 에러가 발생한다.
autoextend_clause	저장할 데이터가 파일 크기를 초과할 경우 파일 크기의 확장 여부를 결정한다.

– tablespace_spec_clause

구성요소	설명
tablespace_name	테이블 스페이스의 이름을 지정한다.
dfspec	파일의 이름, 파일의 크기 등과 관련된 여러 가지 설정을 할 수 있다. 자세한 내용은 **'dfspec'** 항목 설명을 참고한다.
extspec_clause	테이블 스페이스의 익스텐트가 어떻게 관리될 것인지 명시한다.

– autoextend_clause

구성요소	설명
AUTOEXTEND OFF	저장할 데이터가 파일 크기를 초과할 경우 더는 저장하지 못하도록 한다. 이러한 경우 ALTER DATABASE의 RESIZE를 통해 수동으로 파일 크기를 늘려주거나 해당 테이블 스페이스에 파일을 추가하여 해결할 수 있다. 테이블 스페이스에 파일을 추가하는 방법은 “7.2. ALTER DATABASE”를 참고한다.
AUTOEXTEND ON	저장할 데이터가 파일 크기를 초과할 경우 자동으로 파일을 늘려준다.
NEXT	파일의 크기를 확장할 때 늘어나는 크기를 지정할 수 있다. 너무 작은 값을 지정하면 파일 확장이 빈번히 일어나고, 너무 큰 값을 지정하면 저장 공간을 낭비할 수 있으므로 이를 주의한다.
MAXSIZE	파일 크기의 최댓값을 지정할 수 있다.

– log_member_clause

구성요소	설명
GROUP	로그 그룹의 번호를 지정할 수 있다. 지정하지 않으면 0부터 차례대로 지정된다.
REUSE	기존에 존재하는 파일에 덮어쓰고자 할 경우에 사용한다.

- 예제

다음은 **CREATE DATABASE**를 사용해 데이터베이스를 생성하는 예이다.

```
create database "tibero"
     user sys identified by tibero
     maxinstances 8
     maxdatafiles 100
     character set MSWIN949
     national character set UTF16
     logfile
          group 1 'log001.log' size 100M,
          group 2 'log002.log' size 100M,
          group 3 'log003.log' size 100M
     maxloggroups 255
     maxlogmembers 8
     maxarchivelog 500
     maxbackupset 500
     noarchivelog
     datafile 'system001.dtf' size 100M autoextend on next 100M maxsize unlimited

     default temporary tablespace TEMP
     tempfile 'temp001.dtf' size 100M autoextend on next 100M maxsize unlimited
     extent management local autoallocate
     undo tablespace UNDO
     datafile 'undo001.dtf' size 100M autoextend on next 100M maxsize unlimited
     extent management local autoallocate;
```

7.26. CREATE DATABASE LINK

데이터베이스 링크를 생성한다. 데이터베이스 링크는 다른 데이터베이스의 객체에 접근할 수 있도록 해준다. 객체의 이름 뒤에 '@ 데이터베이스 링크 이름'을 붙여줌으로써 가능하다.

접근할 데이터베이스는 반드시 Tibero일 필요는 없다. 기능의 제약이 있기는 하지만, 다른 종류의 데이터베이스의 객체에 접근할 수 있다. 데이터베이스를 생성하고 나면, 테이블, 뷰 등과 같은 연결된 데이터베이스의 객체에 접근하여 DML 문을 수행할 수 있다.

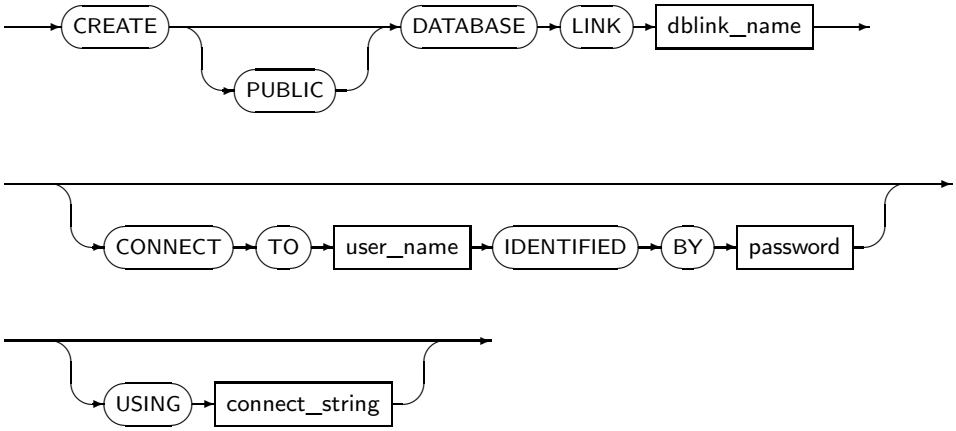
참고

데이터베이스 링크에 대한 자세한 내용은 "Tibero 관리자 안내서"를 참고한다.

CREATE DATABASE LINK의 세부 내용은 다음과 같다.

- 문법

create_database_link



- 특권
  - 사용자가 자신만 사용할 수 있는 데이터베이스 링크를 생성하기 위해서는 CREATE DATABASE LINK 시스템 특권이 있어야 한다.
  - 공유 데이터베이스 링크를 생성하기 위해서는 CREATE PUBLIC DATABASE LINK 시스템 특권이 있어야 한다.
  - 원격 데이터베이스 링크를 생성하기 위해서는 CREATE SESSION 시스템 특권이 있어야 한다.
- 구성요소

구성요소	설명
PUBLIC	모든 사용자가 사용할 수 있는 공유 데이터베이스 링크를 생성하기 위해 사용한다. 생략하면 생성한 사용자만이 사용할 수 있는 데이터베이스 링크가 생성된다.
dblink_name	생성할 데이터베이스 링크의 이름이다. 데이터베이스 링크의 이름을 지정할 때 스키마를 지정할 수 없다. 즉, 다른 사용자의 스키마에 데이터베이스 링크를 생성할 수 없다. 데이터베이스 링크에서는 점(.) 기호도 이름에 포함되는 문자로 인식한다. 공유 데이터베이스 링크가 아니면 생성한 사용자 이외의 다른 사용자는 그 데이터베이스 링크를 사용할 수 없다. 여러 사용자가 데이터베이스 링크를 사용하기 위해서는 반드시 공유 데이터베이스 링크로 생성해야 한다.
CONNECT TO user_name IDENTIFIED BY password	데이터베이스 링크를 통해 원격 데이터베이스에 접속할 때 필요한 사용자의 이름과 패스워드를 설정한다. CONNECT TO를 생략하면 현재 접속한 사용자의 이름과 패스워드로 설정된다. 원격 데이터베이스에 접속하기 위해서는 user_name과 password에 지정한 사용자의 이름과 패스워드, 그리고 원격 데이터베이스에 사용자가 생성되어 있어야 한다.

구성요소	설명
	이때, 패스워드는 기호(작은따옴표) 사이에 패스워드가 입력된 경우에만 유효하다. 예를 들어 아래 예제에서 "tmax"로 패스워드가 입력된 경우에는 데이터베이스 링크 생성에 실패하게 된다.
USING connect_string	원격 데이터베이스의 서비스의 이름을 리터럴 형태로 지정한다. 서비스는 tbdsn.tbr 파일에 지정해야 한다. 이 파일에는 서비스 이름, HOST, 포트 번호, 데이터베이스의 이름 등을 설정할 수 있다. tbdsn.tbr 파일에 지정된 서비스의 이름을 connect_string과 동일하게 입력해야 정상으로 연결할 수 있다.
USING connect_string2	tbdsn.tbr 파일을 통하지 않고 직접 connection에 필요한 정보를 string형태로 USING 절에 기입하여 데이터베이스 링크를 생성할 수 있다. connect_string2에는 HOST, 포트 번호, 데이터베이스 이름 등을 tbdsn .tbr에 적는 형식으로 구성한다. 이 경우 connect_string의 최대 길이는 128byte로 제한된다.

- 예제

  다음은 **CREATE DATABASE LINK**를 사용해 데이터베이스 링크를 생성하는 예이다.

```
CREATE DATABASE LINK remote USING 'remote_tibero';

CREATE PUBLIC DATABASE LINK public_remote
  CONNECT TO tibero IDENTIFIED BY 'tmax' USING 'remote_tibero';

CREATE DATABASE LINK remote_string
  CONNECT TO tibero IDENTIFIED BY 'tmax' USING
          '(INSTANCE=(HOST=localhost)(PORT=9999)(DB_NAME=tibero))';
```

7.27. CREATE DIRECTORY

디렉터리 객체를 생성한다. 디렉터리를 제거하기 위해서는 “7.49. DROP DIRECTORY”의 내용을 참고한다.

CREATE DIRECTORY의 세부 내용은 다음과 같다.

- 문법

create_directory

- 특권
  - 디렉터리를 생성하기 위해서는 CREATE ANY DIRECTORY 시스템 특권이 있어야 한다.
  - 디렉터리를 생성하면 그 디렉터리에 읽기와 쓰기 권한이 자동으로 부여되며, 다른 사용자 또는 역할에 권한을 부여할 수 있다.
- 구성요소

구성요소	설명
OR REPLACE	생성할 디렉터리의 이름이 기존 디렉터리와 동일하면 기존 디렉터리를 제거하고 새로 만든다. 기존의 디렉터리를 삭제한 뒤 다시 생성하는 것과의 차이는 대상 디렉터리에 관련된 기존의 특권과 참조가 그대로 유지된다는 점이다.
dir_name	생성할 디렉터리 객체의 이름이다.
AS dir_path_string	디렉터리 객체가 가리키는 디렉터리 경로를 지정한다. 지정한 디렉터리 경로의 존재 여부나 접근 권한에 대한 검사는 하지 않는다. 잘못된 경로를 지정하면 외부 테이블 등에서 이 디렉터리를 접근하면 에러가 발생한다. 디렉터리 경로는 문자열 리터럴로 표현하며, 대소문자를 구분한다.

- 예제

  다음은 **CREATE DIRECTORY**를 사용해 '/tmp' 경로를 가리키는 tmp라는 디렉터리 객체를 생성하는 예이다.

```
SQL> CREATE DIRECTORY tmp AS '/tmp';

Directory created.
```

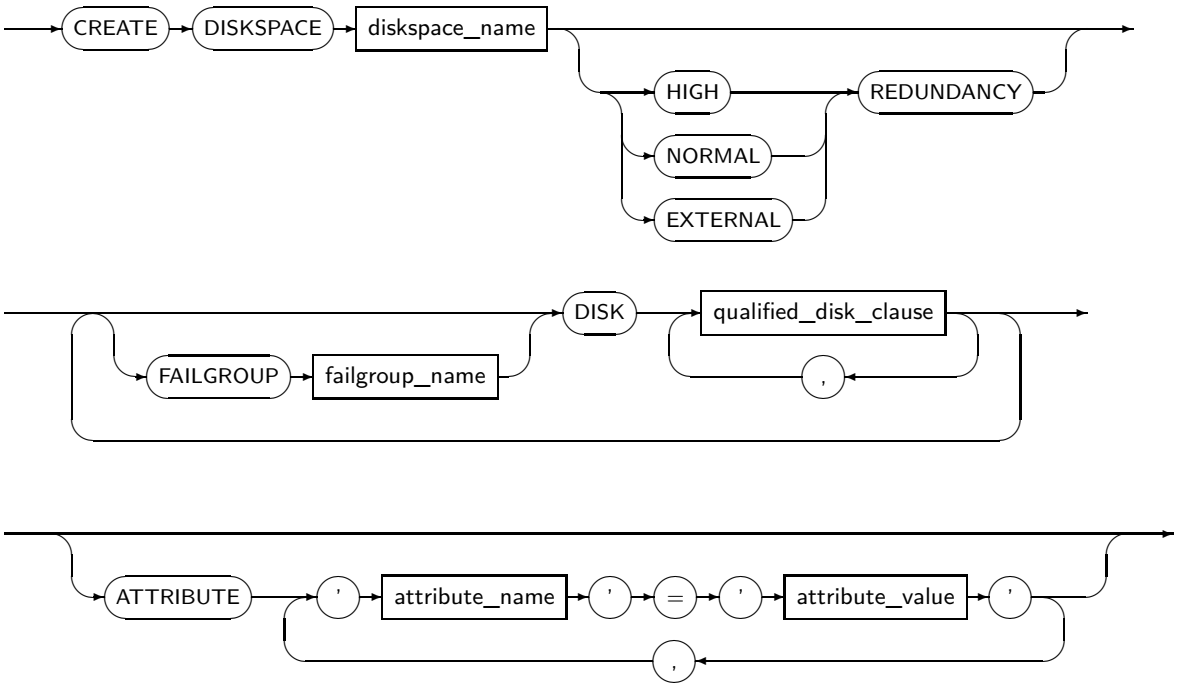
7.28. CREATE DISKSPACE

디스크스페이스를 생성한다.

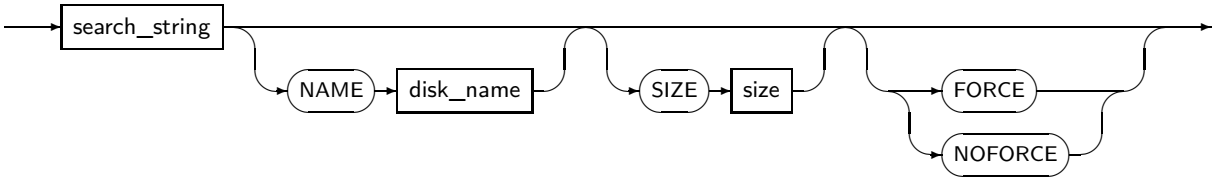
CREATE DISKSPACE의 세부 내용은 다음과 같다.

- 문법

create_diskspace



qualified_disk_clause



- 구성요소

  – create_diskspace

구성요소	설명
diskspace_name	생성할 디스크스페이스의 이름을 명시한다.
REDUNDANCY	디스크스페이스에 저장되는 사용자 파일의 중복 레벨을 지정한다. 중복 레벨이 높을수록 데이터 유실 위험이 줄어드는 대신 디스크 공간 사용과 디스크 쓰기에 대한 부하가 증가한다. 다음 3가지 중복 레벨 중 하나를 사용할 수 있다. – HIGH: 디스크스페이스의 모든 파일을 서로 다른 세 개의 failgroup에 중복해서 저장한다. 세 개 이상의 failgroup이 있는 경우에만 지정할 수 있으며, 최대 두 개의 failgroup에 장애가 발생하더라도 정상적인 서비스가 가능하다. – NORMAL: 디스크스페이스의 모든 파일을 서로 다른 두 개의 failgroup에 중복해서 저장한다. 두 개 이상의 failgroup이 있는 경우에만 지정할 수 있

구성요소	설명
	으며, 하나의 failgroup에 장애가 발생하더라도 정상적인 서비스가 가능하다. 중복 레벨이 지정되지 않은 경우의 기본값이다. – EXTERNAL: 중복 저장 기능을 사용하지 않는다. 디스크 자체의 중복 저장 기능을 사용하거나, 디스크 장애로 인한 데이터 유실을 감수할 수 있는 경우 선택한다.
FAILGROUP fail group_name	디스크가 속할 failgroup 이름을 명시한다. 디스크스페이스의 중복 레벨이 NORMAL 또는 HIGH인 경우에만 유효하며, 알파벳과 숫자 등으로 최대 48자를 사용할 수 있다. 명시하지 않은 경우에는 모든 디스크가 각각 하나의 failgroup으로 구성되며 디스크 이름이 failgroup 이름으로 사용된다.
DISK quali fied_disk_clause	디스크스페이스를 구성할 디스크를 정의한다.
ATTRIBUTE	디스크스페이스의 속성을 지정한다. 지정할 수 있는 속성은 다음과 같다. – AU_SIZE: 디스크스페이스의 공간 할당 단위를 byte로 지정한다. (기본값: 1MB) – SECTOR_SIZE: 디스크스페이스를 구성하는 디스크들의 섹터 사이즈를 지정한다. (기본값: 512)

– qualified_disk_clause

구성요소	설명
search_string	디스크스페이스를 구성할 디스크의 경로를 명시한다. 와일드카드를 사용해 여러 디스크를 나타낼 수 있으며, TAS_DISKSTRING 초기화 파라미터로 찾을 수 있는 경로이어야 한다.
NAME disk_name	search_string으로 찾은 디스크의 이름을 명시한다. search_string으로 찾은 디스크가 한 개인 경우에만 지정할 수 있으며, 알파벳과 숫자 등으로 최대 48자를 사용할 수 있다. 디스크의 이름은 TAS 내부에서만 사용되며 디스크 경로와는 무관하다. 명시하지 않은 경우 임의로 생성된 이름이 사용된다.
SIZE	search_string으로 찾은 디스크의 크기를 byte 단위로 명시한다. search_string으로 찾은 디스크가 여러 개인 경우 모든 디스크가 같은 크기로 지정된다.

구성요소	설명
	명시하지 않은 경우 TAS에서 파악한 실제 디스크의 크기가 지정된다. TAS에서 실제 디스크를 크기를 알 수 없는 경우 오류가 발생하며, 이 때는 반드시 디스크 크기를 명시해 주어야 한다.
FORCE	search_string으로 찾은 디스크가 이미 다른 디스크스페이스에 사용 중이더라도 이를 무시하고 새로운 디스크스페이스 구성에 사용한다. 기존 디스크스페이스를 파기하는 경우에 명시한다.
NOFORCE	search_string으로 찾은 디스크가 이미 다른 디스크스페이스에 사용 중인 경우 오류가 발생한다. FORCE 또는 NOFORCE가 명시되지 않은 경우의 기본 동작이다.

- 예제

  다음은 **CREATE DISKSPACE**를 사용해 중복 저장을 사용하지 않는 디스크스페이스를 생성하는 예이다.

```
CREATE DISKSPACE ds
    EXTERNAL REDUNDANCY
    DISK '/tas/dev/path0' NAME disk0 SIZE 512G,
         '/tas/dev/path1' NAME disk1 SIZE 256G;
```

  다음은 **CREATE DISKSPACE**를 사용해 세 개의 failgroup으로 구성된 디스크스페이스를 생성하는 예이다.

```
CREATE DISKSPACE ds
    HIGH REDUNDANCY
    FAILGROUP fg0 DISK
        '/tas/dev/path00' NAME disk00 SIZE 512G,
        '/tas/dev/path01' NAME disk01 SIZE 512G
    FAILGROUP fg1 DISK
        '/tas/dev/path10' NAME disk10 SIZE 512G,
        '/tas/dev/path11' NAME disk11 SIZE 512G
    FAILGROUP fg2 DISK
        '/tas/dev/path20' NAME disk20 SIZE 512G,
        '/tas/dev/path21' NAME disk21 SIZE 512G
    ATTRIBUTE 'AU_SIZE' = '4M';
```

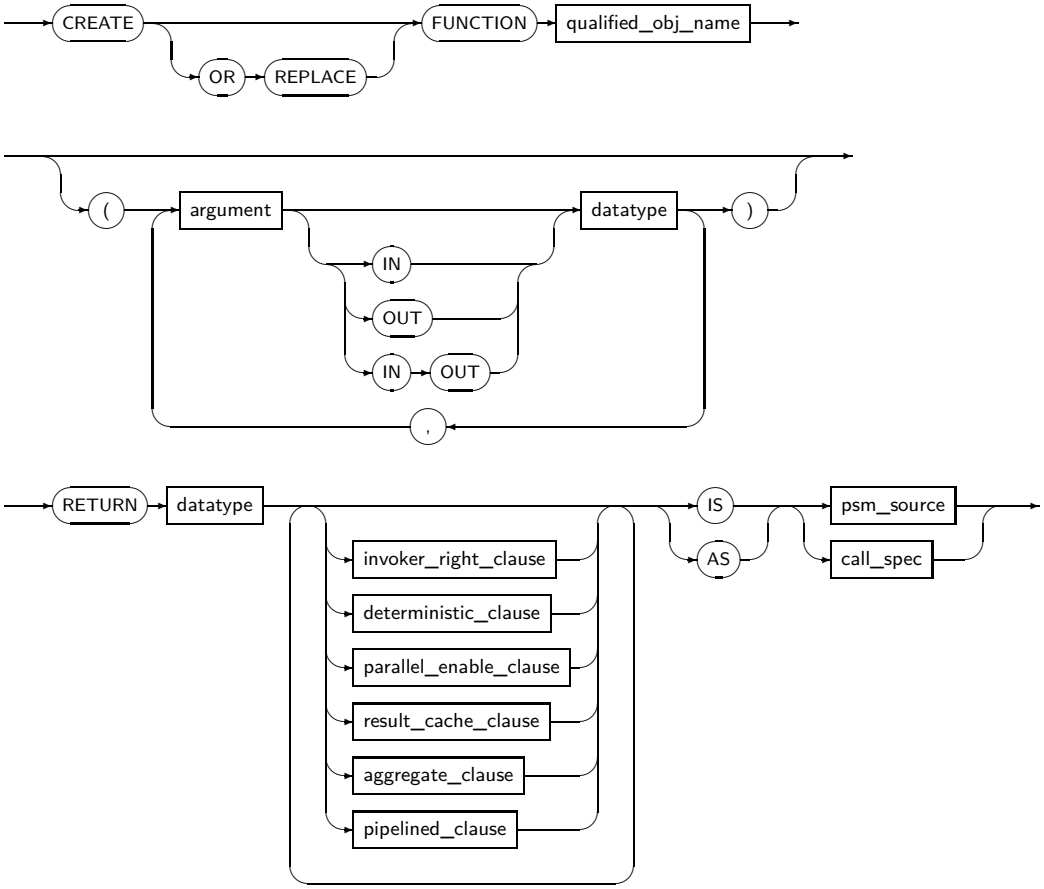
7.29. CREATE FUNCTION

사용자 함수(User Function)를 새로 정의하거나 기존의 함수를 대체한다. 사용자 함수는 반환 값이 있는 tbPSM 프로그램이며, Tibero 서버에 저장되고 실행된다. 사용자 함수가 사용자 프러시저(User procedure)와 다른 점은 반환 값이 있으며, 질의 문장 또는 DML 문장에 포함될 수 있다는 것이다.

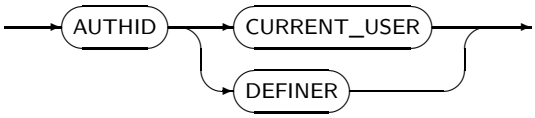
CREATE FUNCTION의 세부 내용은 다음과 같다.

- 문법

create_function



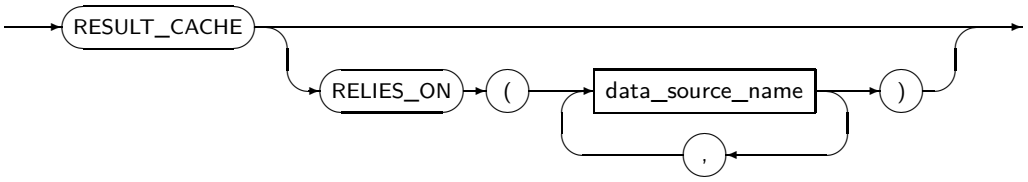
invoker_right_clause



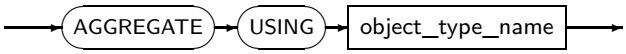
deterministic_clause

parallel_enable_clause

result_cache_clause



aggregate_clause



pipelined_clause

- 특권
  - 사용자가 자신의 스키마에 함수를 생성하기 위해서는 CREATE PROCEDURE 시스템 특권을 부여받아야 한다.
  - 다른 사용자가 소유한 스키마에 함수를 생성하기 위해서는 CREATE ANY PROCEDURE 시스템 특권을 부여받아야 하고, 변경하고자 하면 ALTER ANY PROCEDURE 시스템 특권을 부여받아야 한다.
- 구성요소
  - create_function

구성요소	설명
OR REPLACE	이미 존재하는 함수를 다시 정의하고자 할 때 사용한다. OR REPLACE 절이 포함되면 해당 함수를 재컴파일한다. 함수를 제거하고 다시 생성하는 것의 차이는 OR REPLACE 절을 이용하면 해당 함수에 기존의 특권이 그대로 유지된다는 점이다.
qualified_obj_name	함수의 스키마와 이름을 지정한다.
argument	함수의 파라미터이다. 함수의 파라미터는 0개 이상이 될 수 있다. 파라미터가 0개이면 파라미터를 감싸는 괄호를 생략한다.
IN	함수의 파라미터의 전달 방향에 따른 구분이다. IN 파라미터는 외부로부터 값을 입력받는다. 디폴트는 IN 파라미터이다. 함수에서는 현재 트랜잭션에 COMMIT, ROLLBACK, SAVEPOINT를 수행할 수 없고 DDL 문장도 실행할 수 없다. 또한, 현재 액세스 중인 테이블에도 갱신을 수행할 수 없다.
OUT	함수의 파라미터의 전달 방향에 따른 구분이다.

구성요소	설명
	OUT 파라미터는 함수 내부로부터 값을 출력한다. DML 문장에 포함된 함수는 OUT 파라미터를 가질 수 없다. DML 문장에 포함된 함수로부터 직접 또는 간접적으로 호출되는 함수는 OUT 또는 IN OUT 파라미터를 가질 수 있으나 나머지는 불가능하다.
IN OUT	함수의 파라미터의 전달 방향에 따른 구분이다. IN OUT 파라미터는 입력과 출력 모두에 사용된다. DML 문장에 포함된 함수는 IN OUT 파라미터를 가질 수 없다.
NOCOPY	파라미터의 값을 가리키는 포인터(Pointer)를 전달한다. 파라미터의 값을 복사하지 않기 때문에 전달 속도가 빠르다.
datatype	함수 파라미터 또는 반환 값의 데이터 타입이다. 함수의 파라미터 또는 반환 값의 데이터 타입은 tbPSM에서 지원하는 모든 데이터 타입이 가능하다. 반환 값의 데이터 타입을 선언할 때 문자열의 길이, 숫자의 정밀도와 스케일은 지정할 수 없다.
RETURN	함수의 반환 값의 데이터 타입을 설정한다.
invoker_right_clause	사용자 함수는 호출자 권한(Invoker-rights) 함수 또는 정의자 권한(Definer-rights) 함수로 구분할 수 있다. 이는 함수를 실행할 때에 어떤 사용자의 특권을 이용하여 실행할 것인지, 액세스하고자 하는 테이블 등의 객체는 어떤 스키마에서 찾을 것인지 등을 결정한다. 디폴트는 정의자 권한 함수이다.
deterministic_clause	함수를 캐싱 가능 함수로 선언한다.
parallel_enable_clause	함수가 병렬 실행 가능하다고 명시한다.
result_cache_clause	함수 결과를 서버 결과 캐시에 저장함을 나타낸다.
aggregate_clause	함수를 집계 함수로 식별한다.
pipelined_clause	함수를 파이프라인 함수로 선언한다.
IS	IS나 AS는 기호에 맞게 선택해서 사용하고 둘의 차이는 없다. IS 다음에는 함수의 본문이 온다.
AS	IS와 동일하다. AS 다음에는 함수의 본문이 온다.
psm_source	PSM 소스 코드가 오는 부분이다. 자세한 내용은 "Tibero tbPSM 안내서"를 참고한다.

구성요소	설명
call_spec	외부 함수를 호출하기 위한 명세를 지정한다. 자세한 내용은 "Tibero tbPSM 안내서"를 참고한다.

– invoker_right_clause

구성요소	설명
AUTHID CURRECT_USER	사용자 함수를 호출자 권한 함수로 선언한다. 생략하면 정의자 권한 함수로 선언된다. 호출자 권한 함수로 선언하면, 현재 사용자의 특권을 이용하며 현재 사용자의 스키마에서 액세스할 객체를 찾는다. 따라서, 함수를 호출하는 사용자가 달라지면 가능한 작업의 범위와 액세스하는 스키마 객체도 달라진다. 이러한 함수는 여러 사용자가 공통으로 동일한 작업을 수행하고자 할 때에 유용하다.
AUTHID DEFINER	사용자 함수를 정의자 권한 함수로 선언한다. 디폴트이므로 생략하면 정의자 권한 함수로 선언된다. 정의자 권한 함수로 선언하면, 함수를 정의한 사용자의 특권을 이용하고 그 사용자의 스키마에서 액세스할 객체를 찾는다. 따라서, 호출자에 관계없이 작업의 범위와 액세스하는 스키마 객체가 항상 일정하다. 이러한 함수는 데이터 사전 등의 시스템 데이터의 일부를 일반 사용자가 액세스할 수 있을 때 유용하다.

– deterministic_clause

구성요소	설명
DETERMINISTIC	사용자 함수를 결정적 함수로 선언하고, 함수의 반환 값을 강제로 캐싱하여서 반환한다. 생략하면 컴파일 시 캐싱 가능 여부를 판단하여 함수를 생성한다.

– parallel_enable_clause

구성요소	설명
PARALLEL_ENABLE	사용자 함수가 병렬 실행될 수 있음을 나타낸다.

– result_cache_clause

구성요소	설명
RESULT_CACHE	사용자 함수의 결과를 공유 메모리의 캐쉬에 저장한다.
RELIES_ON	함수 결과가 의존하는 데이터 소스들을 지정한다.

구성요소	설명
	더이상 사용되지 않는 기능이다. 결과 캐시 기능이 실행되는 동안 쿼리되는 모든 데이터 소스들을 감지하므로, 적는 것이 무의미하다.
data_source_name	데이터 소스의 이름이다. 스키마 이름을 포함해도 된다.

– aggregate_clause

구성요소	설명
AGGREGATE USING	사용자 함수를 집계 함수로 선언한다.
object_type_name	집계 함수가 참조하는 객체 타입의 이름을 명시한다. 객체 타입은 쿼리 수행기의 AGGREGATION 동작을 포함하는 TUDIAGGREGATEINITIALIZE, TUDIAGGREGATEITERATE, TUDIAGGREGATETERMINATE, TUDIAGGREGATEMERGE(선택가능)이 포함되어야 한다. "객체 타입의 이름" 또는 "스키마명.객체 타입"의 이름으로 쓸 수 있다.

– pipelined_clause

구성요소	설명
PIPELINED	사용자 함수를 파이프라인 함수로 선언한다.

- 예제

  다음은 **CREATE FUNCTION**을 사용해 사용자 함수를 새로 정의하는 예이다.

```
CREATE OR REPLACE FUNCTION square(origin IN NUMBER)
  RETURN NUMBER IS
    origin_square NUMBER;
  BEGIN
    origin_square := origin * origin;

    RETURN origin_square;
  END;
```

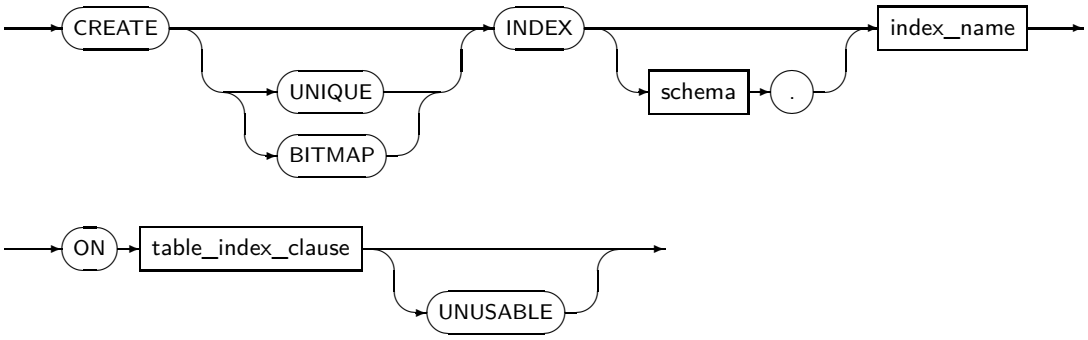
7.30. CREATE INDEX

인덱스를 생성한다. 기반 테이블의 하나 이상의 컬럼에 인덱스를 생성할 수 있다.

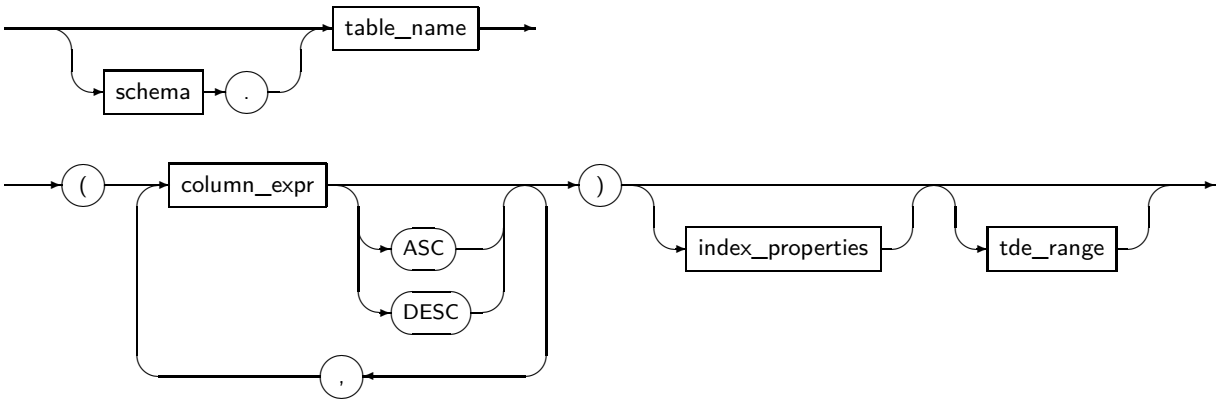
CREATE INDEX의 세부 내용은 다음과 같다.

- 문법

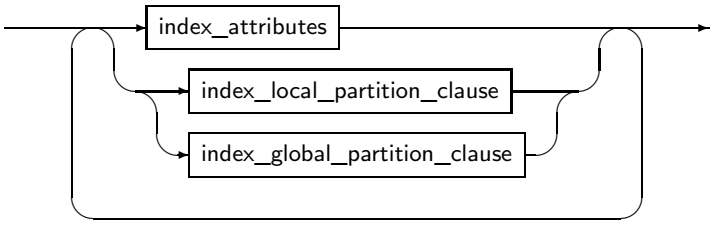
create_index



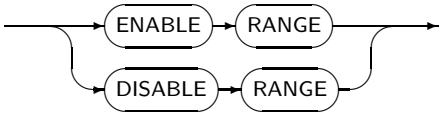
table_index_clause



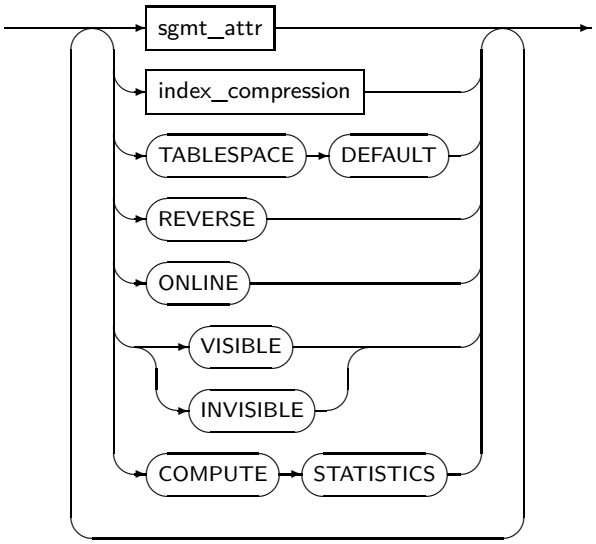
index_properties



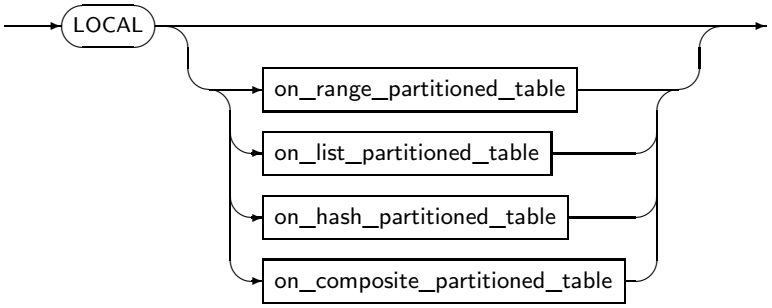
tde_range



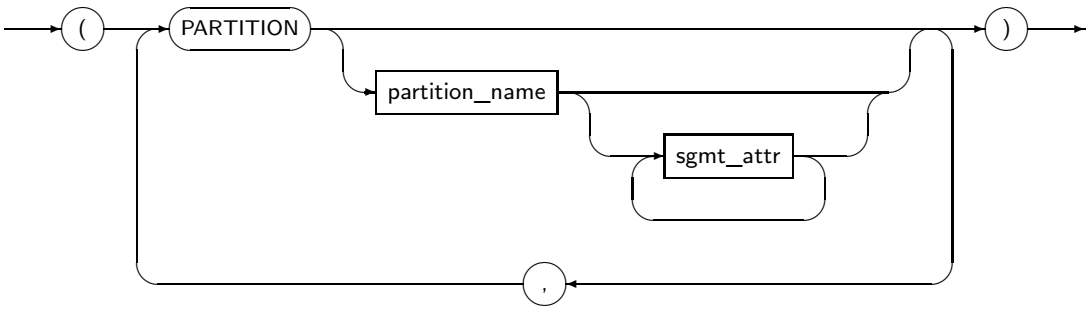
index_attributes



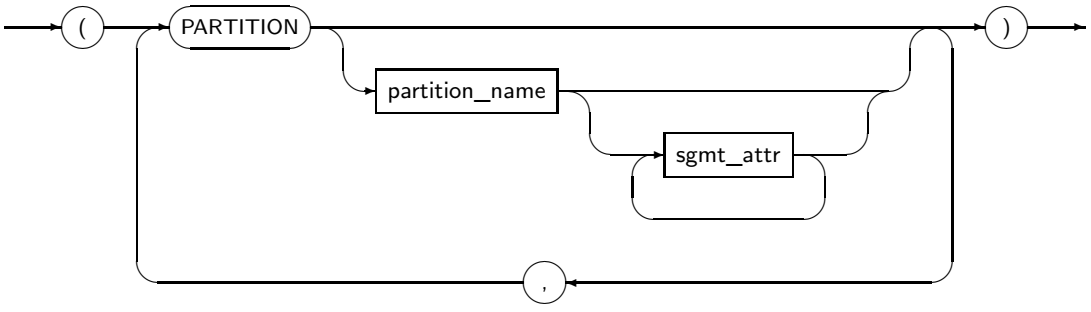
index_local_partition_clause



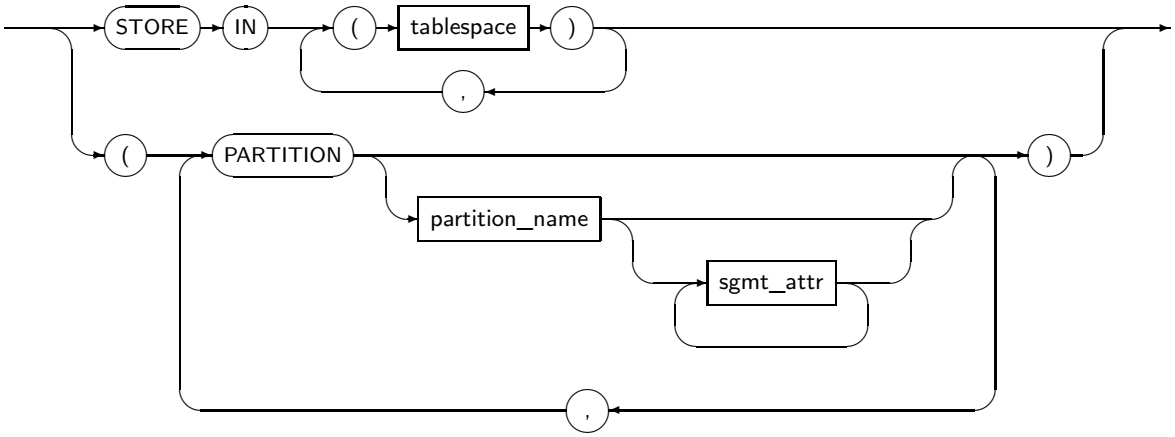
on_range_partitioned_table



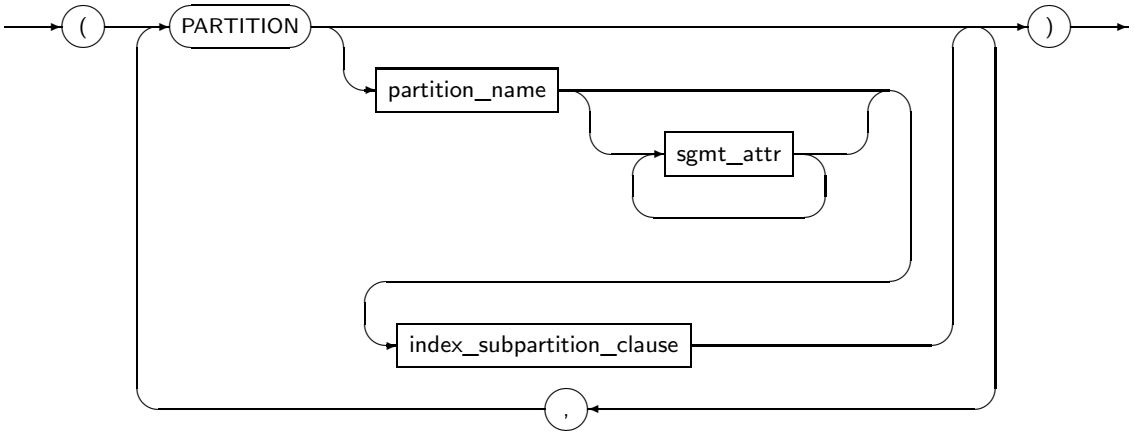
on_list_partitioned_table



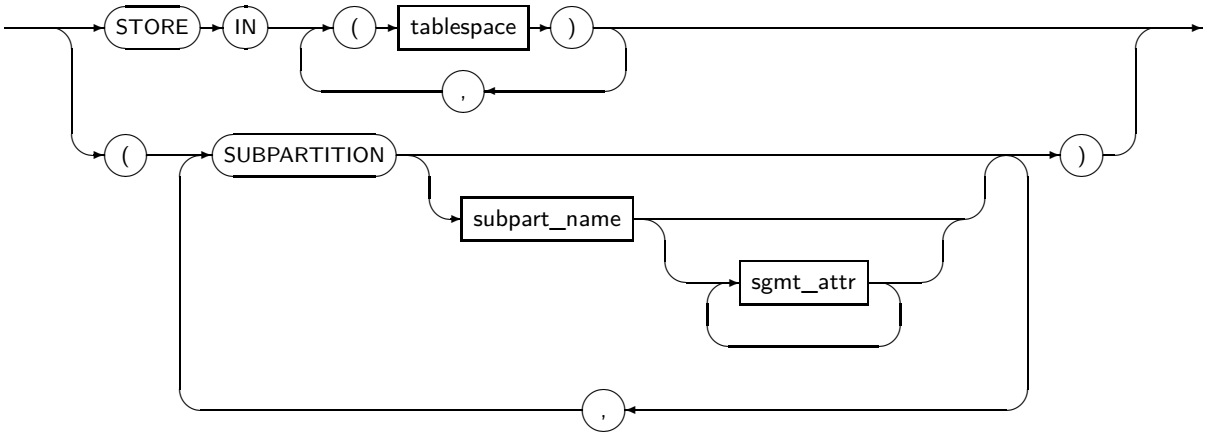
on_hash_partitioned_table



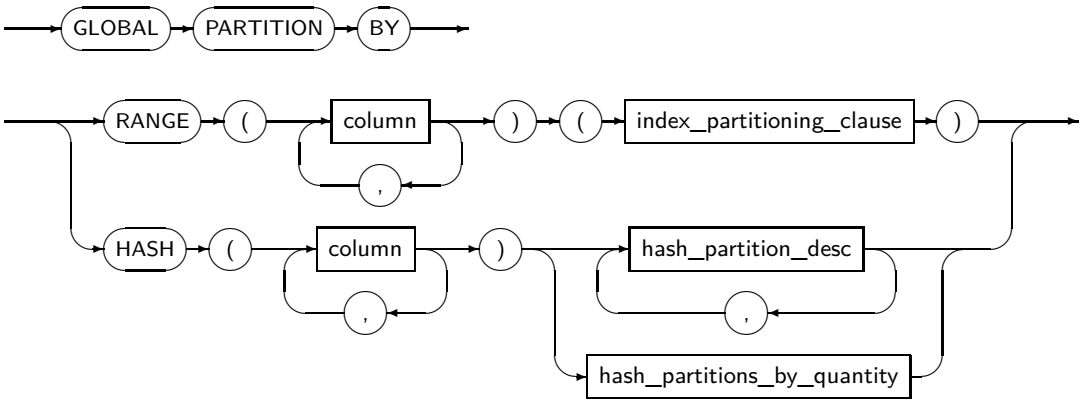
on_composite_partitioned_table



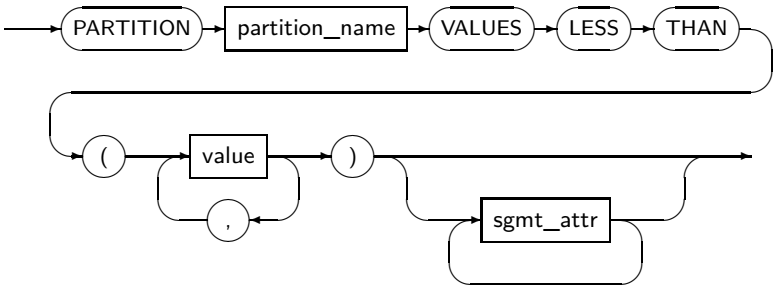
index_subpartition_clause



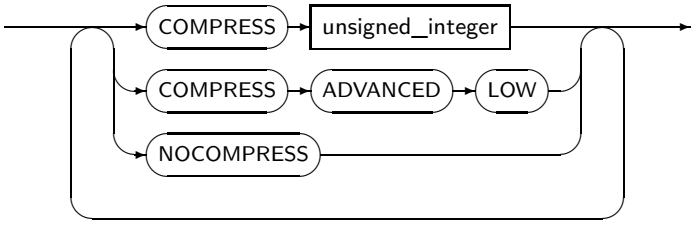
index_global_partition_clause



index_partitioning_clause



index_compression



- 특권

  다음 중 하나를 만족해야 CREATE INDEX 문을 실행할 수 있다.

  – 기반 테이블이 사용자 자신의 스키마에 포함되어 있다.

  – 기반 테이블에 대한 INDEX 스키마 객체 특권이 있다.

  – CREATE ANY INDEX 시스템 특권이 있다.

- 구성요소

  – create_index

구성요소	설명
UNIQUE	지정되면 유일 인덱스를 생성한다. 유일 인덱스에는 중복된 키 값이 저장될 수 없다.
BITMAP	지정되면 비트맵 인덱스를 생성한다.

구성요소	설명
schema	인덱스를 생성할 스키마를 명시한다. 생략하면 현재 사용자의 스키마로 인식된다.
index_name	생성할 인덱스의 이름을 명시한다.
ON table_index_clause	인덱스를 생성할 테이블의 구체적인 내용을 명시한다.
UNUSABLE	인덱스를 사용 불가능한 상태로 생성한다. 이 인덱스를 사용하기 위해서는 ALTER INDEX의 REBUILD를 통해 재생성해야 한다.

– table_index_clause

구성요소	설명
schema	인덱스를 생성할 테이블이 속해 있는 스키마를 명시한다. 생략하면 현재 사용자의 스키마로 인식된다.
table_name	인덱스를 생성할 테이블의 이름을 명시한다.
column_expr	인덱스 키로 사용될 컬럼 이름 또는 표현식을 명시한다. 인덱스 키로는 LONG, LONG RAW 타입, 대용량 객체형 등이 올 수 없다. 표현식의 결과 값도 동일한 제한을 갖는다. 같은 키를 갖는 인덱스가 데이터베이스에 이미 존재한다면 똑같은 인덱스를 생성하는 것이 제한된다. 이는 중복된 인덱스 사용으로 불필요하게 효율이 떨어지는 것을 막기 위함이다. 따라서 같은 인덱스를 만들기보다는 이미 만들어진 인덱스에 대한 권한을 얻어서 사용하는 것이 좋다. 인덱스 키 표현식에서 사용자 정의 함수를 사용하는 경우에는 함수가 반드시 DETERMINISTIC으로 선언되어야 한다. 사용한 함수가 변경되거나 삭제되면 해당 인덱스는 사용할 수 없는 상태가 된다. 또한, 인덱스 키 표현식에서 사용하는 함수는 항상 동일한 결과 값을 가져야 한다. 예를 들어 SYSDATE 등과 같이 결과 값이 변하는 함수는 사용할 수 없다.
ASC	컬럼 값의 정렬 순서를 오름차순으로 지정한다. (기본값)
DESC	컬럼 값의 정렬 순서를 내림차순으로 지정한다. 내림차순 정렬순서는 다음과 같은 경우에 사용한다. – 복합 키로 사용하는 경우: 컬럼 A에 대해서는 오름차순으로 하되, 컬럼 A가 동일한 값을 가질 때 B를 내림차순으로 정렬하고 싶은 경우(A, B DESC)를 지정하면 된다. – 키가 삽입되는 순서가 내림차순일 경우: Tibero의 인덱스는 주로 오름차순으로 삽입되는 것에 최적화되어 설계되어 있다. 따라서 키가 주로 내림차순으로 삽입되는 경우 내림차순으로 정렬해야 인덱스의 효율이 더 좋아진다.

구성요소	설명
index_properties	인덱스의 구체적인 속성을 지정한다.
tde_range	암호화 테이블의 인덱스를 이용하여 Range 스캔을 할 것인지 여부를 설정한다.

– index_properties

구성요소	설명
index_attributes	인덱스의 물리적인 속성을 지정한다.
index_local_partition_clause	로컬 파티션 인덱스를 생성한다. 테이블의 파티션과 동일한 개수로 생성되며, 테이블 파티션이 변경되면 인덱스 파티션도 자동으로 같이 변경된다.
index_global_partition_clause	글로벌 파티션 인덱스를 생성한다. 테이블의 파티션 여부와 상관없이 인덱스 고유의 파티셔닝을 지정하여 인덱스를 생성한다.

– index_attributes

구성요소	설명
sgmt_attr	sgmt_attr 관련 문법은 “7.1.4. Sgmt_attr”을 참고한다.
index_compression	인덱스의 압축 여부를 지정한다.
TABLESPACE (DEFAULT)	인덱스를 생성할 테이블 스페이스를 지정한다. 생략하거나 DEFAULT로 설정하면 기반 테이블이 있는 테이블 스페이스에 인덱스를 생성한다.
REVERSE	ROWID를 제외한 인덱스 블록의 바이트 순서를 역으로 저장한다. 비슷한 키 값이 집중된 경우 이를 분산하고 싶을 때 사용한다.
ONLINE	인덱스의 생성 도중 DML을 허용하고 싶을 때 사용한다.
INVISIBLE	인덱스의 상태 옵션이다. 인덱스의 VISIBLE과 INVISIBLE 여부는 OPTIMIZER가 해당 인덱스를 고려하는지 여부와 관련이 있다. INVISIBLE 상태인 인덱스는 DML 과정에 일반 인덱스처럼 형태를 계속 유지한다. 하지만 Optimizer에서 고려할 대상으로 여기지 않는다.
COMPUTE STATISTICS	인덱스 생성 직후에 인덱스 통계 정보를 수집하고자 할 때 사용한다. B-Tree INDEX 와 IOT Secondary INDEX일 때만 사용할 수 있다.

– index_local_partition_clause

구성요소	설명
on_range_partitioned_table	RANGE 파티션 테이블에 대한 로컬 파티션 인덱스를 생성할 때 사용한다. PARTITION 절을 사용할 경우 PARTITION 절의 개수는 테이블의 파티션 개수와 동일해야 하며, 맞지 않을 경우 에러가 발생한다. 자세한 내용은 "7.1.4. Sgmt_attr"을 참고한다.
on_list_partitioned_table	LIST 파티션 테이블에 대한 로컬 파티션 인덱스를 생성할 때 사용한다. 자세한 내용은 on_range_partitioned_table을 참고한다.
on_hash_partitioned_table	HASH 파티션 테이블에 대한 로컬 파티션 인덱스를 생성할 때 사용한다. 자세한 내용은 on_hash_partitioned_table을 참고한다.
on_composite_partitioned_table	복합 파티션 테이블에 대한 로컬 파티션 인덱스를 생성할 때 사용한다. 자세한 내용은 on_range_partitioned_table을 참고한다.

– on_range_partitioned_table

구성요소	설명
partition_name	인덱스 파티션의 이름을 지정할 때 사용하며, 지정하지 않으면 시스템이 자동으로 생성한다.
sgmt_attr	sgmt_attr는 인덱스 파티션으로 사용될 세그먼트의 속성을 지정한다. sgmt_attr 관련 문법은 "7.1.4. Sgmt_attr"을 참고한다.

– on_list_partitioned_table

구성요소	설명
partition_name	인덱스 파티션의 이름을 지정할 때 사용하며, 지정하지 않으면 시스템이 자동으로 생성한다.
sgmt_attr	sgmt_attr는 인덱스 파티션으로 사용될 세그먼트의 속성을 지정한다. sgmt_attr 관련 문법은 "7.1.4. Sgmt_attr"을 참고한다.

– on_hash_partitioned_table

구성요소	설명
STORE IN	각 파티션이 위치할 테이블 스페이스를 지정한다. 테이블 스페이스를 지정한 개수가 파티션의 개수와 일치할 필요는 없으며, 개수가 서로 다를 경우 라운드 로빈(round-robin) 방식으로 테이블 스페이스가 차례대로 지정된다.
tablespace	테이블 스페이스를 명시한다.
partition_name	인덱스 파티션의 이름을 지정할 때 사용하며, 지정하지 않으면 시스템이 자동으로 생성한다.

구성요소	설명
sgmt_attr	sgmt_attr는 인덱스 파티션으로 사용될 세그먼트의 속성을 지정한다. sgmt_attr 관련 문법은 “7.1.4. Sgmt_attr”을 참고한다.

– on_composite_partitioned_table

구성요소	설명
partition_name	인덱스 파티션의 이름을 지정할 때 사용하며, 지정하지 않으면 시스템이 자동으로 생성한다.
sgmt_attr	sgmt_attr는 인덱스 파티션으로 사용될 세그먼트의 속성을 지정한다. sgmt_attr 관련 문법은 “7.1.4. Sgmt_attr”을 참고한다.
index_subpartition_clause	각 서브 파티션의 세부적인 설정을 지정한다. 단, STORE IN은 서브 파티셔닝 방법이 HASH일 경우에만 사용할 수 있다. 서브 파티션이라는 점 외에 다른 문법은 기본 파티셔닝과 동일하다. 자세한 내용은 on_range_partitioned_table을 참고한다.

– index_subpartition_clause

구성요소	설명
STORE IN	서브 파티셔닝 방법이 HAHS일 경우에만 사용할 수 있다. STORE IN 절은 각 파티션이 위치할 테이블 스페이스를 지정한다. 테이블 스페이스를 지정한 개수가 파티션의 개수와 일치할 필요는 없으며, 개수가 서로 다를 경우 라운드 로빈 방식으로 테이블 스페이스가 차례대로 지정된다.
tablespace	테이블 스페이스를 명시한다.
subpartition_name	인덱스 서브 파티션의 이름을 지정할 때 사용하며, 지정하지 않으면 시스템이 자동으로 생성한다.
sgmt_attr	인덱스 서브 파티션으로 사용될 세그먼트의 속성을 지정한다. sgmt_attr 관련 문법은 “7.1.4. Sgmt_attr”을 참고한다.

– index_global_partition_clause

구성요소	설명
RANGE	서브 파티션의 파티셔닝 방법을 RANGE 파티션으로 설정한다.
HASH	서브 파티션의 파티셔닝 방법을 HASH 파티션으로 설정한다.
column	인덱스 키 컬럼으로 사용된 컬럼 중에서 파티션의 기준이 되는 컬럼을 지정한다. 단, 파티션 키 컬럼이 인덱스 키 컬럼의 앞부분을 반드시 포함해야 한다. 즉, 인덱스의 키가 (a, b, c)라면 (a), (a, b), (a, b, c)를 파티션 키

구성요소	설명
	컬럼으로 지정할 수 있다. 인덱스 키 컬럼의 앞부분을 포함하지 않도록 파티션 키를 정하면 에러가 발생한다. 인덱스를 어떤 기준으로 파티션할지는 테이블이 어떻게 파티션되어 있는지와는 전혀 관계가 없다. 테이블이 파티션되는 방식과 인덱스가 파티션되는 방식이 다를 수 있다. 또한, 테이블이 파티션되어 있지 않더라도 인덱스는 파티션할 수 있고, 반대로 테이블이 파티션되어 있더라도 인덱스는 파티션되지 않을 수도 있다.
index_partitioning_clause	RANGE 파티션 인덱스의 세부적인 속성을 지정한다.
hash_partition_desc	HASH 파티션의 세부적인 속성을 지정한다.
hash_partitions_by_quantity	HASH 파티션을 파티션 개수와 테이블 스페이스만 지정함으로써 간단하게 생성할 수 있도록 해 주는 문법이다.

- index_partitioning_clause

구성요소	설명
partition_name	인덱스 파티션의 이름을 지정할 때 사용하며, 지정하지 않으면 시스템이 자동으로 생성한다.
value	파티션을 나눌 기준 값을 지정한다. value로 사용된 값보다 작은 값이 해당 파티션에 포함된다. MAXVALUE로 지정하면 다른 파티션에 속하지 않는 모든 값이 해당 파티션으로 포함된다. column에서 사용된 컬럼 수와 값의 개수가 같아야 한다.
sgmt_attr	인덱스 파티션으로 사용될 세그먼트의 속성을 지정한다. sgmt_attr 관련 문법은 “7.1.4. Sgmt_attr”을 참고한다.

- index_compression

구성요소	설명
COMPRESS unsigned_integer	인덱스를 압축한다. 인덱스 키 컬럼의 상위 unsigned integer 개를 prefix key로 하여 index leaf block에 대해 중복되는 부분을 압축한다. unsigned integer의 default 값은 UNIQUE INDEX일 경우 인덱스 키 컬럼의 - 1, 아닐 경우 인덱스 키 컬럼의 값이다.
COMPERSS ADVANCED LOW	아직 지원하지 않는 기능이다.
NOCOMPRESS	인덱스를 압축하지 않는다.

- 예제

– create_index

다음은 **UNUSABLE**을 명시해 인덱스를 사용 불가능한 상태로 생성하는 예이다.

```
SQL> CREATE INDEX i ON t (a) UNUSABLE;
Index created.
```

– table_index_clause

다음은 **column_expr**을 명시해 인덱스를 생성하는 예이다.

```
SQL> CONN u1/u1 Connected.

SQL> CREATE TABLE t (a NUMBER, b VARCHAR(10));
Table created.

SQL> CREATE UNIQUE INDEX i ON t (a);
Unique created.

SQL> CREATE INDEX i2 ON t (UPPER(b));
Index created.
```

다음은 **column_expr**에 명시한 인덱스 키로 LONG 타입, LONG RAW 타입, 대용량 객체형 데이터 타입을 사용한 경우 에러가 발생하는 예이다.

```
SQL> CREATE TABLE t (a NUMBER, b LONG);
Table created.

SQL> CREATE UNIQUE INDEX i ON t(a, b);
TBR-7086: cannot create an index on columns whose type is
LONG, LONG RAW, LOB.
```

다음은 **colume_expr**에 명시한 인덱스 키와 같은 키를 갖는 인덱스가 이미 존재할 경우 에러가 발생하는 예이다.

```
SQL> CREATE UNIQUE INDEX i ON t (a);
Unique created.

SQL> CREATE UNIQUE INDEX j ON t (a);
TBR-7124: such index already exists.
```

다음은 **colume_expr**에 사용자 정의 함수를 사용할 때 사용자 정의 함수가 DETERMINISTIC으로 선언되지 않았을 경우 에러가 발생하는 예이다.

```
SQL> CREATE TABLE t (a NUMBER);
Table created.
```

```
SQL> CREATE FUNCTION plus4 (a NUMBER)
     RETURN NUMBER AS BEGIN RETURN a + 4; END;
Function created.

SQL> CREATE INDEX i ON t (plus4(a));
TBR-8082: nondeterministic function,
date or system variable not allowed here.
```

```
SQL> CREATE TABLE t (a NUMBER);
Table created.

SQL> CREATE INDEX i ON t (SYSDATE);
TBR-8082: nondeterministic function, date or system variable not
allowed here.
```

위의 예에서 SYSDATE 함수는 결과 값이 항상 변하기 때문에 사용할 수 없다.

다음은 **UNIQUE**를 명시하여 유일 인덱스를 생성하는 예이다.

```
SQL> CREATE UNIQUE INDEX i ON t (a);
Unique created.

SQL> INSERT INTO t VALUES (1);
1 row inserted.

SQL> INSERT INTO t VALUES (1);
TBR-10007: unique constraint violated.
```

다음은 **DESC**를 명시하여 인덱스를 내림차순으로 생성하는 예이다.

```
SQL> CREATE TABLE t (a NUMBER, b NUMBER);
Table created.

SQL> CREATE UNIQUE INDEX i ON t (a, b DESC);
Unique created.
```

- index_attributes

다음은 **TABLESPACE**를 명시하여 인덱스를 생성할 테이블 스페이스를 지정하는 예이다.

```
SQL> CONN sys/tibero
Connected.

SQL> CREATE TABLESPACE ts DATAFILE '/usr/ts.df' SIZE 10M;
Tablespace created.
```

```
SQL> CONN u1/u1
Connected.

SQL> CREATE INDEX i ON t (a) TABLESPACE ts;
Index created.
```

- index_local_partition_clause

  다음은 **index_local_partition_clause**를 명시하여 로컬 파티션 인덱스를 생성하는 예이다.

```
SQL> CREATE TABLE t (a NUMBER, b NUMBER, c NUMBER)
     PARTITION BY RANGE (a, b)
     (PARTITION p1 VALUES LESS THAN (10, 10)
      TABLESPACE ts1, PARTITION p2
      VALUES LESS THAN (MAXVALUE, MAXVALUE)
      TABLESPACE ts2);
Table created.

SQL> CREATE INDEX i ON t (a, b) LOCAL
     (PARTITION p1, PARTITION p2, PARTITION p3);
TBR-7159: local index partition count is different with table's

SQL> CREATE INDEX i ON t (a, b) LOCAL
     (PARTITION p1 INITRANS 10, PARTITION p2 STORAGE (MAXEXTENTS 100));
Index created.
```

  다음은 **index_local_partition_clause**를 명시하여 로컬 파티션 인덱스를 생성하는 예이다.

```
SQL> CONN sys/tibero Connected.

SQL> CREATE TABLESPACE ts1 DATAFILE '/usr/ts1.df' SIZE 10M;
Tablespace created.

SQL> CREATE TABLESPACE ts2 DATAFILE '/usr/ts2.df' SIZE 10M;
Tablespace created.

SQL> CONN u1/u1 Connected.

SQL> CREATE TABLE t (a NUMBER, b NUMBER, c NUMBER)
     PARTITION BY RANGE (a, b)
     (PARTITION p1 VALUES LESS THAN (10, 10)
      TABLESPACE ts1, PARTITION p2
      VALUES LESS THAN (MAXVALUE, MAXVALUE) TABLESPACE ts2);
Table created.

SQL> CREATE INDEX i ON t (a, b) LOCAL;
Index created.
```

– index_global_partition_clause

다음은 **index_global_partition_clause**를 명시하여 글로벌 파티션 인덱스를 생성하는 예이다.

```
SQL> CONN sys/tibero
Connected.

SQL> CREATE TABLESPACE ts1 DATAFILE '/usr/ts1.df' SIZE 10M;
Tablespace created.

SQL> CREATE TABLESPACE ts2 DATAFILE '/usr/ts2.df' SIZE 10M;
Tablespace created.

SQL> CONN u1/u1
Connected.

SQL> CREATE TABLE t (a NUMBER, b NUMBER, c NUMBER);
Table created.

SQL> CREATE INDEX i ON t (a, b, c) GLOBAL PARTITION BY RANGE (a, b)
     (PARTITION p1 VALUES LESS THAN (10, 10)
      TABLESPACE ts1, PARTITION p2
      VALUES LESS THAN (MAXVALUE, MAXVALUE) TABLESPACE ts2);
Index created.
```

다음은 **index_global_partition_clause**의 구성요소인 **column** 부분을 명시할 때 파티션 키 컬럼을 잘못 지정했을 경우 에러가 발생하는 예이다.

```
SQL> CREATE INDEX i ON t (a, b)
     GLOBAL PARTITION BY RANGE (b)
     (PARTITION p1 VALUES LESS THAN (10) TABLESPACE ts1,
      PARTITION p2 VALUES LESS THAN (MAXVALUE) TABLESPACE ts2);
TBR-7225: global partitioned index must be prefixed.
```

다음은 **index_global_partition_clause**의 구성요소인 **column** 부분을 명시할 때 DESC로 지정된 인덱스 키 컬럼을 파티션 키로 지정할 경우 에러가 발생하는 예이다.

```
SQL> CREATE INDEX i ON t (a, b DESC)
     GLOBAL PARTITION BY RANGE (a, b)
     (PARTITION p1 VALUES LESS THAN (10, 10)
     TABLESPACE ts1, PARTITION p2 VALUES LESS THAN
        (MAXVALUE, MAXVALUE) TABLESPACE ts2);
TBR-7225: global partitioned index must be prefixed.
```

다음은 **index_global_partition_clause**의 구성요소인 **value** 부분을 명시할 때 값의 수가 컬럼의 개수와 맞지 않거나 값이 정렬되지 않은 경우 에러가 발생하는 예이다.

```
SQL> CREATE INDEX i ON t (a, b)
    GLOBAL PARTITION BY RANGE (a, b)
    (PARTITION p1
      VALUES LESS THAN (10) TABLESPACE ts1,
      PARTITION p2 VALUES LESS THAN (MAXVALUE)
      TABLESPACE ts2);
TBR-7163: specified partition values are not correct.
```

다음은 **index_global_partition_clause**의 구성요소인 **value** 부분을 명시할 때 컬럼 타입이 맞지 않을 경우 에러가 발생하는 예이다.

```
SQL> CREATE INDEX i ON t (a, b)
     GLOBAL PARTITION BY RANGE (a)
     (PARTITION p1 VALUES LESS THAN ('abc') TABLESPACE ts1,
      PARTITION p2 VALUES LESS THAN (MAXVALUE) TABLESPACE ts2);
TBR-5074: given string does not represent a number in proper format.
```

다음은 **index_global_partition_clause**의 구성요소인 **value** 부분을 명시할 때 MAXVALUE를 지정하지 않을 경우 에러가 발생하는 예이다.

```
SQL> CREATE INDEX i ON t (a, b)
     GLOBAL PARTITION BY RANGE (a, b)
     (PARTITION p1 VALUES LESS THAN (10, 20) TABLESPACE ts1,
      PARTITION p2 VALUES LESS THAN (30, 40) TABLESPACE ts2);
TBR-7429: When creating global range partitioned index,
the partition whose partitioning key columns'
values are all MAXVALUE should be specified.
```

– index_compression

다음은 **index_compression**를 명시하여 compressed 인덱스를 생성하는 예이다.

```
SQL> CREATE INDEX i ON t (a, b) COMPRESS;
Index 'I' created.
```

다음은 **index_compression**의 구성요소인 **unsigned_integer** 부분을 명시할 때 잘못된 prefix length 값을 입력할 경우 에러가 발생하는 예이다.

```
SQL> CREATE INDEX i ON t (a) compress 2;
TBR-7551: Invalid COMPRESS prefix length value

SQL> CREATE INDEX i ON t (a) compress 0;
TBR-7551: Invalid COMPRESS prefix length value

SQL> CREATE UNIQUE INDEX i ON t (a, b) compress 2;
TBR-7551 : Invalid COMPRESS prefix length value
```

```
SQL> CREATE UNIQUE INDEX i on t (a) compress;
TBR-7552 : Cannot use COMPRESS option for a single column key
```

다음은 **index_compression**에서 일반 인덱스, UNIQUE 인덱스 이외의 인덱스를 압축하려 할 때 에러가 발생하는 예이다.

```
SQL> CREATE BITMAP INDEX i ON t (a) compress;
TBR-7013 : This option cannot be specified.

SQL> CREATE TABLE t (A NUMBER PRIMARY KEY, B NUMBER) ORGANIZATION INDEX;
SQL> CREATE INDEX i ON t (b) compress;
TBR-7013 : This option cannot be specified.
```

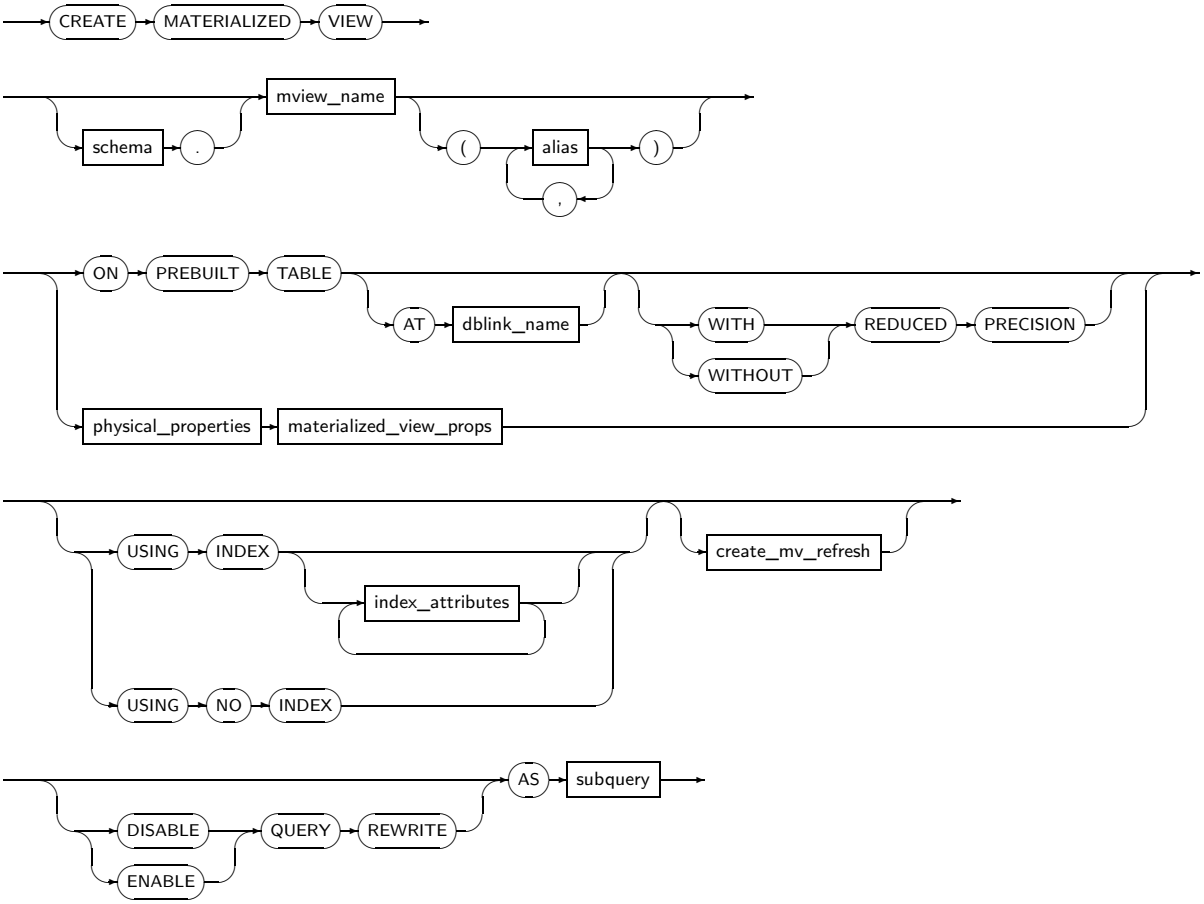
7.31. CREATE MATERIALIZED VIEW

실체화 뷰를 생성하고 실체화 뷰의 속성을 지정한다.

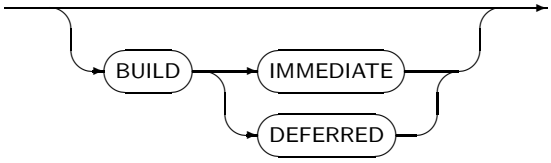
CREATE MATERIALIZED VIEW의 세부 내용은 다음과 같다.

- 문법

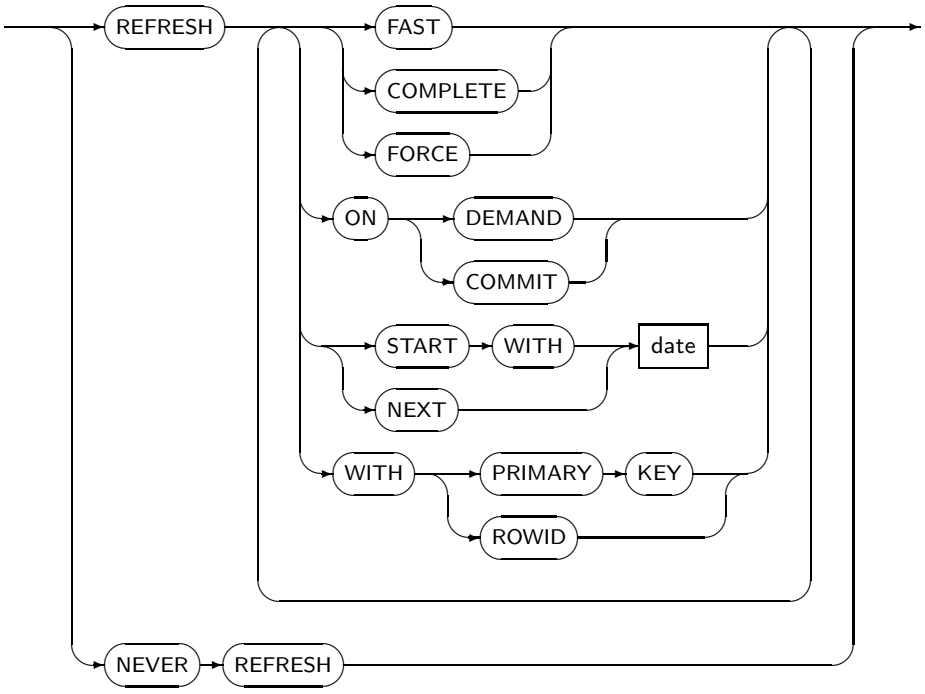
create_materialized_view



materialized_view_props



create_mv_refresh



- 특권
  - 사용자 자신이 소유한 스키마에 실체화 뷰를 생성 조건
    - CREATE MATERIALIZED VIEW 시스템 특권과 CREATE TABLE 또는 CREATE ANY TABLE 시스템 특권이 있어야 한다.
    - 실체화 뷰가 참조하는 테이블 중 사용자 소유가 아닌 테이블에 대한 SELECT 객체 시스템 특권 또는 SELECT ANY TABLE 시스템 특권이 있어야 한다.
  - 다른 사용자가 소유한 스키마에 실체화 뷰를 생성 조건
    - CREATE ANY MATERIALIZED VIEW 시스템 특권과 CREATE TABLE 시스템 특권이 있어야 한다.
    - 실체화 뷰가 참조하는 테이블 중 해당 스키마를 소유한 사용자가 소유하지 않은 테이블에 대한 SELECT 객체 시스템 특권 또는 SELECT ANY TABLE 시스템 특권이 있어야 한다.
- 구성요소
  - create_materialized_view

구성요소	설명
schema	생성할 실체화 뷰의 스키마를 명시한다. 생략하면 현재 사용자의 스키마로 인식된다.
mview_name	생성할 실체화 뷰의 이름을 명시한다.
alias	실체화 뷰의 별칭을 명시한다.

구성요소	설명
ON PREBUILT TABLE	이미 존재하는 테이블을 사용하여 실체화 뷰를 생성한다. 테이블은 생성될 실체화 뷰와 같은 스키마에 속해야 하며 같은 이름을 가지고 있어야 한다. – WITH REDUCED PRECISION: 테이블의 컬럼의 정확도가 실체화 뷰를 정의한 질의의 결과와 다른 것을 허용한다. – WITHOUT REDUCED PRECISION: 테이블의 컬럼의 정확도와 실체화 뷰를 정의한 질의의 결과가 일치해야 한다. (기본값)
AT	원격 저장소를 가진 실체화 뷰를 생성할 때 저장소가 위치하는 데이터베이스 링크를 지정한다. 자세한 내용은 “제6장 실체화 뷰”를 참고한다.
physical_properties	테이블의 물리적인 속성을 지정한다. 이와 관련된 문법은 “7.41. CREATE TABLE”을 참고한다.
materialized_view_props	실체화 뷰의 속성을 지정한다.
USING INDEX	실체화 뷰를 생성할 때 시스템에서 필요한 인덱스를 자동으로 생성한다. 인덱스와 관련된 속성을 지정할 수 있다. (기본값) 이와 관련된 문법은 관련 문법은 “7.30. CREATE INDEX”를 참고한다.
USING NO INDEX	실체화 뷰를 생성할 때 시스템에서 인덱스를 생성하지 않는다.
create_mv_refresh	데이터베이스가 실체화 뷰를 리프레시하는 스케줄, 모드, 방법 등을 조절할 수 있다. 실체화 뷰가 참조하는 테이블에 변경이 발생하면, 테이블의 현재 데이터를 반영하기 위해 실체화 뷰를 리프레시한다.
DISABLE	실체화 뷰를 질의 다시 쓰기에 사용될 수 있는 상태로 설정한다.
ENABLE	실체화 뷰를 질의 다시 쓰기에 사용될 수 있는 상태로 설정한다. (기본값)
QUERY REWRITE	실체화 뷰가 질의 다시 쓰기에 사용될지의 여부를 설정한다.
AS subquery	실체화 뷰를 정의하는 질의를 명시한다. 실체화 뷰를 생성하면 Tibero는 AS subquery의 질의를 수행하고, 그 결과를 실체화 뷰에 저장한다. 모든 SQL 문장을 사용할 수 있다. 하지만, 올바른 문장이어야 하고, 질의에 따라서 빠른 리프레시나 질의 다시 쓰기를 사용할 수 없을 수도 있다.

– materialized_view_props

구성요소	설명
BUILD	실체화 뷰에 데이터를 언제 삽입할 것인지를 설정한다.
IMMEDIATE	데이터를 실체화 뷰를 생성하는 즉시 삽입한다. 기본값이므로 생략하면 IMMEDIATE로 인식한다.
DEFERRED	처음으로 리프레시할 때 데이터를 삽입한다.

구성요소	설명
	첫 번째 리프레시는 완전 리프레시이다. 첫 번째 리프레시부터 질의 다시 쓰기에 사용될 수 있다.

– create_mv_refresh

구성요소	설명
FAST	빠른 리프레시를 수행하려고 할 때 사용한다.
COMPLETE	실체화 뷰를 정의하는 질의를 재수행하여 완전 리프레시를 사용한다. COMPLETE가 명시되면 빠른 리프레시가 가능하더라도 완전 리프레시를 사용한다. (기본값)
FORCE	빠른 리프레시가 가능하면 빠른 리프레시를 수행하고, 그렇지 않으면 완전 리프레시를 수행한다.
ON DEMAND	사용자가 DBMS_MVIEW 패키지의 REFRESH 프러시저를 호출하는 경우에만 리프레시를 수행한다. 생략하면 ON DEMAND로 인식된다. (기본값)
ON COMMIT	ON COMMIT이 명시된 경우 마스터 테이블에 커밋이 일어날 때마다 리프레시를 수행한다. 하지만, START WITH와 NEXT는 명시할 수 없다. ON COMMIT과 ON DEMAND는 동시에 명시할 수 없다.
START WITH	처음으로 자동 리프레시가 시작될 날짜형 표현식을 명시한다. START WITH는 미래의 시간을 나타내는 값이어야 한다. NEXT 없이 START WITH만을 명시할 경우 데이터베이스는 한 번만 리프레시를 수행한다.
NEXT	자동 리프레시의 간격을 계산하기 위한 날짜형 표현식을 명시한다. NEXT는 미래의 시간을 나타내는 값이어야 한다. START WITH 없이 NEXT만 명시한 경우 데이터베이스는 NEXT 표현식을 평가하여 첫 리프레시의 시간을 정한다.
date	START WITH와 NEXT에 지정할 날짜형 리터럴을 명시한다.
WITH PRIMARY KEY	PRIMARY KEY를 사용하여 리프레시를 수행한다.
WITH ROWID	ROWID를 사용하여 리프레시를 수행한다.
NEVER REFRESH	자동 리프레시를 하지 않는다.

- 예제

다음은 **CREATE MATERIALIZED VIEW**를 사용해 실체화 뷰를 생성하는 예이다.

```
CREATE MATERIALIZED VIEW MV AS SELECT * FROM T;
```

다음은 실체화 뷰를 생성하면서 **START WITH**와 **NEXT**를 사용해 리프레시하는 시간을 설정하는 예이다.

```
CREATE MATERIALIZED VIEW MV
REFRESH START WITH SYSDATE NEXT SYSDATE + 10/1440
AS SELECT * FROM T;
```

위의 예에서는 START WITH 와 NEXT 절을 이용하여 기존의 실체화 뷰를 10분마다 자동으로 리프레시하는 실체화 뷰를 생성하였다.

다음은 **ENABLE QUERY REWRITE**와 **BUILD DEFERRED**를 사용해 실체화 뷰를 생성하는 예이다.

```
CREATE MATERIALIZED VIEW MV
  BUILD DEFERRED
  ENABLE QUERY REWRITE
  AS SELECT * FROM T;
```

위의 예를 보면, ENABLE QUERY REWRITE를 사용해 질의 다시 쓰기 기능을 활성화한다. 그다음 BUILD DEFERRED를 사용해 실체화 뷰에 데이터를 삽입한다. 그리고 처음으로 리프레시할 때 데이터를 삽입하도록 설정하였다.

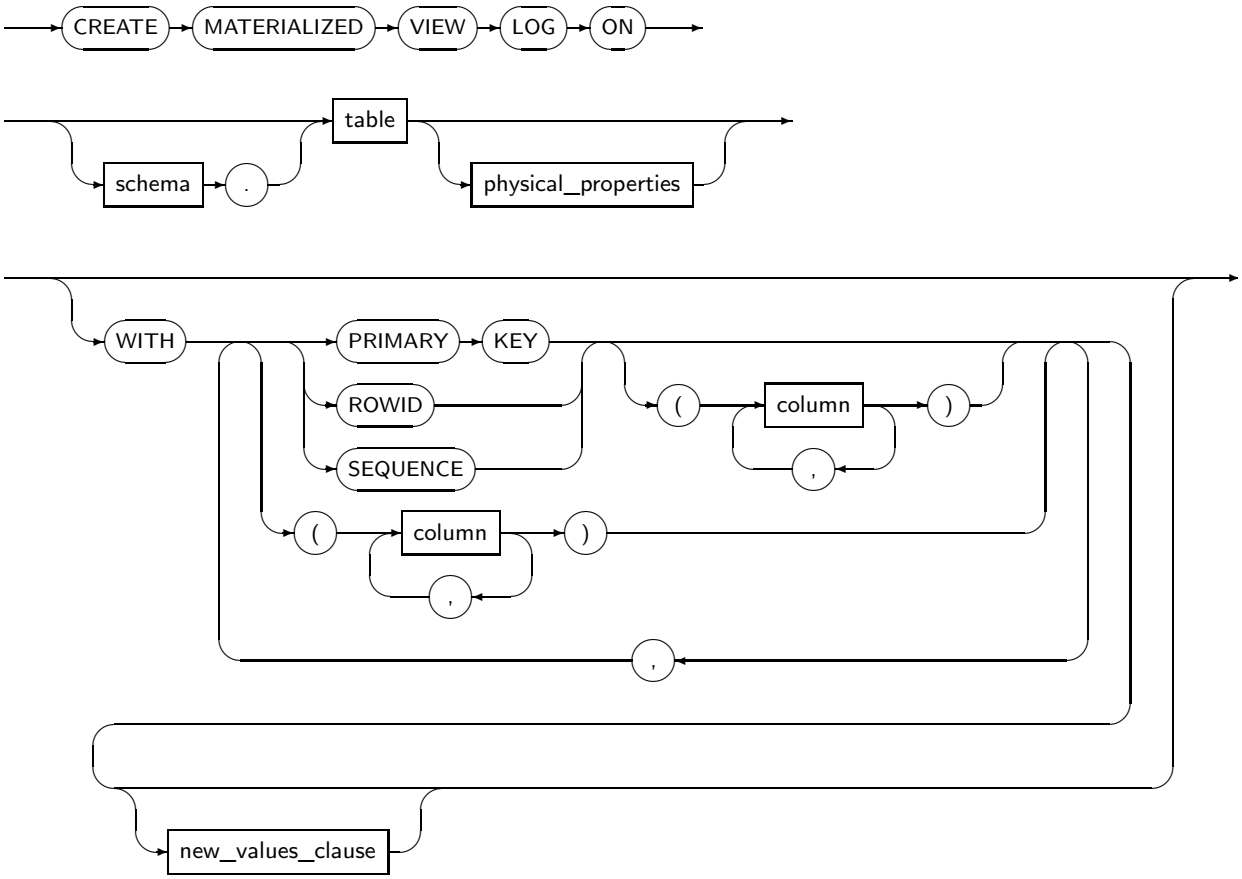
7.32. CREATE MATERIALIZED VIEW LOG

지정된 마스터 테이블에 실체화 뷰 로그를 생성하고 실체화 뷰 로그의 속성을 지정한다. 실체화 뷰 로그는 실체화 뷰를 리프레시할 때 사용되며, 마스터 테이블의 변경 정보를 기록한다.

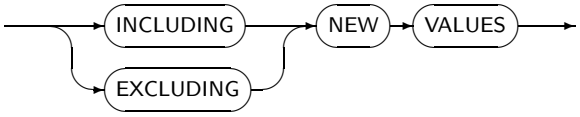
CREATE MATERIALIZED VIEW LOG의 세부 내용은 다음과 같다.

- 문법

create_materialized_view_log



new_values_clause



- 특권

  실체화 뷰 로그를 생성하려면 다음의 조건을 만족해야 한다.

  – 마스터 테이블을 사용자가 소유하고 있는 경우 CREATE TABLE 시스템 특권이 있어야 한다.

  – 다른 사용자의 스키마에 실체화 뷰 로그를 생성할 경우에는 CREATE ANY TABLE 시스템 특권과 마스터 테이블에 대한 SELECT 권한 또는 SELECT ANY TABLE 시스템 특권이 있어야 한다.

- 구성요소

  – create_materialized_view_log

구성요소	설명
schema	실체화 뷰 로그를 생성할 마스터 테이블의 스키마를 명시한다. 생략하면 현재 사용자의 스키마로 인식된다.
table	실체화 뷰 로그를 생성할 마스터 테이블의 이름을 명시한다.

구성요소	설명
physical_properties	실체화 뷰 로그 테이블의 물리적인 속성을 지정한다. 관련된 문법은 "7.41. CREATE TABLE"을 참고한다.
PRIMARY KEY	실체화 뷰 로그에 마스터 테이블의 변경된 로우의 PRIMARY KEY를 기록한다.
ROWID	실체화 뷰 로그에 마스터 테이블의 변경된 로우의 ROWID를 기록한다.
SEQUENCE	실체화 뷰 로그에 마스터 테이블의 변경된 로우의 순서를 기록한다.
column	실체화 뷰 로그에 기록할 마스터 테이블의 컬럼을 지정한다.

– new_values_clause

구성요소	설명
INCLUDING NEW VALUES	실체화 뷰 로그에 컬럼의 변경 전/후의 값을 모두 기록한다.
EXCLUDING NEW VALUES	실체화 뷰 로그에 컬럼의 변경 전의 값만을 기록한다. (기본값)

- 예제

  다음은 **CREATE MATERIALIZED VIEW LOG**를 사용해 실체화 뷰 로그를 생성하는 예이다.

```
CREATE MATERIALIZED VIEW LOG ON DEPT;
```

  다음은 실체화 뷰 로그를 생성하면서 컬럼을 지정하는 예이다. 본 예제에서는 생성된 실체화 뷰 로그에 DNAME과 LOC 컬럼을 기록한다.

```
CREATE MATERIALIZED VIEW LOG ON DEPT WITH (DNAME, LOC);
```

7.33. CREATE OUTLINE

쿼리 플랜 정보를 저장하기 위해서 아웃라인을 생성한다. 아웃라인은 해당 쿼리의 플랜 정보를 힌트의 집합으로 나타내며, 현재는 테이블 액세스 정보와 조인 순서가 힌트로 저장된다.

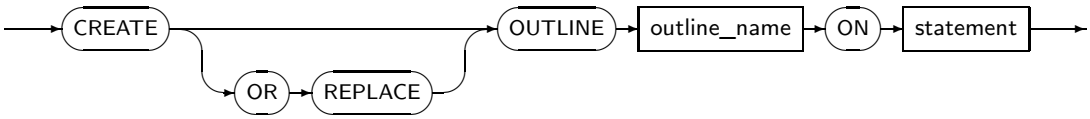
참고

아웃라인을 제거하기 위해서는 "7.55. DROP OUTLINE"의 내용을 참고한다.

CREATE OUTLINE의 세부 내용은 다음과 같다.

- 문법

create_outline



- 구성요소

구성요소	설명
OR REPALCE	생성할 아웃라인이 기존에 존재하는 아웃라인이라면, 그 아웃라인을 제거하고 새로 생성한다.
outline_name	생성할 아웃라인의 이름이다.
statement	아웃라인을 생성할 쿼리를 지정한다.

- 예제

다음은 **CREATE OUTLINE**을 사용해 아웃라인을 생성하는 예이다.

```
SQL> CREATE OR REPLACE OUTLINE ol_join
     ON SELECT e.lname, d.dname FROM emp e, dept d WHERE e.deptno = d.deptno;

Outline Created.
```

또는 CREATE_STORED_OUTLINES라는 파라미터를 켜면 수행되는 쿼리들에 대해서 SYS_OUTLINE으로 시작하는 아웃라인이 자동으로 생성된다.

```
SQL> alter session set CREATE_STORED_OUTLINES = y;

SQL> SELECT e.lname, d.dname FROM emp e, dept d WHERE e.deptno = d.deptno;
```

USE_STORED_OUTLINES라는 파라미터를 켜면 쿼리 수행시마다 적용할 수 있는 아웃라인을 찾아서 적용한다. 아웃라인은 쿼리가 정확하게 매치될 때에만 같은 쿼리로 인식하고 적용하므로 빈칸 등에 유의해야 한다.

```
SQL> alter session set USE_STORED_OUTLINES = y;

SQL> SELECT e.lname, d.dname FROM emp e, dept d WHERE e.deptno = d.deptno;
```

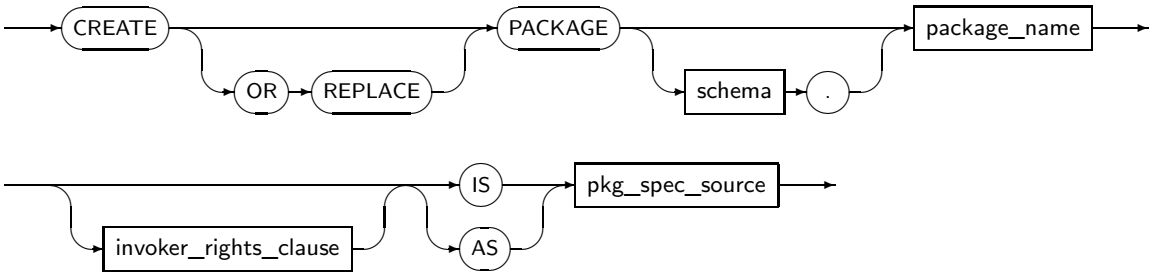
7.34. CREATE PACKAGE

새로운 패키지를 생성한다. 패키지란 서로 관련 있는 프러시저나 함수, 그리고 다른 프로그램의 객체를 데이터베이스 내에 한 영역으로 모아서 저장한 것을 의미한다. 패키지는 이러한 객체를 선언하는 역할을 하며, 패키지 본문은 객체의 실제 내용을 정의하는 역할을 한다.

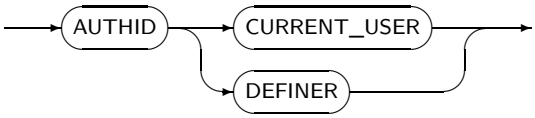
CREATE PACKAGE의 세부 내용은 다음과 같다.

- 문법

create_package



invoker_right_clause



- 특권
  - 사용자 자신의 스키마에 패키지를 생성하려면 CREATE PROCEDURE 시스템 특권이 있어야 한다.
  - 다른 사용자의 스키마에 패키지를 생성하려면 CREATE ANY PROCEDURE 시스템 특권이 있어야 한다.
- 구성요소
  - create_package

구성요소	설명
OR REPLACE	해당 항목을 설정하면 이미 존재하는 패키지라도 다시 생성할 수 있다. 패키지를 변경하고자 할 때 명세를 제거하고, 다시 생성하고, 스키마 객체 특권을 설정하는 단계를 반복하는 것을 피할 수 있다. 패키지의 명세가 바뀌면 패키지 본문은 무효가 되며, 다음번에 사용하려고 하면 패키지를 재컴파일한다.
schema	변경할 패키지가 속해 있는 스키마를 명시한다.
package_name	변경할 패키지의 이름을 명시한다.
invoker_right_clause	호출자 권한 패키지를 생성할지 아니면 정의자 권한 패키지를 생성할지를 결정한다.
IS	IS나 AS는 기호에 맞게 선택할 수 있으며, 둘의 차이는 없다. IS 다음에는 패키지의 본문이 온다.

구성요소	설명
AS	IS와 동일하다. AS 다음에는 패키지의 본문이 온다.
pkg_spec_source	패키지의 명세를 지정한다. 자세한 내용은 "Tibero tbPSM 안내서"를 참고한다.

– invoker_right_clause

구성요소	설명
AUTHID CURRENT_USER	호출자 권한 패키지를 생성한다. 이 패키지의 프러시저 또는 함수가 수행될 때는 현재 사용자의 스키마에서 수행되는 것으로 간주하며, 특권도 현재 사용자의 특권을 가지고 수행된다.
AUTHID DEFINER	정의자 권한 패키지를 생성한다. 이 패키지의 프러시저 또는 함수가 수행될 때는 정의자의 스키마와 특권을 가지고 패키지의 프러시저와 함수를 수행한다. 기본은 정의자 권한 패키지이다.

- 예제

  다음은 **CREATE PACKAGE**를 사용해 패키지를 생성하는 예이다.

```
CREATE OR REPLACE PACKAGE dept_pkg AS
  FUNCTION create (div_id NUMBER, loc_id NUMBER) RETURN NUMBER;
  PROCEDURE remove (dept_id NUMBER);
  END dept_pkg;
```

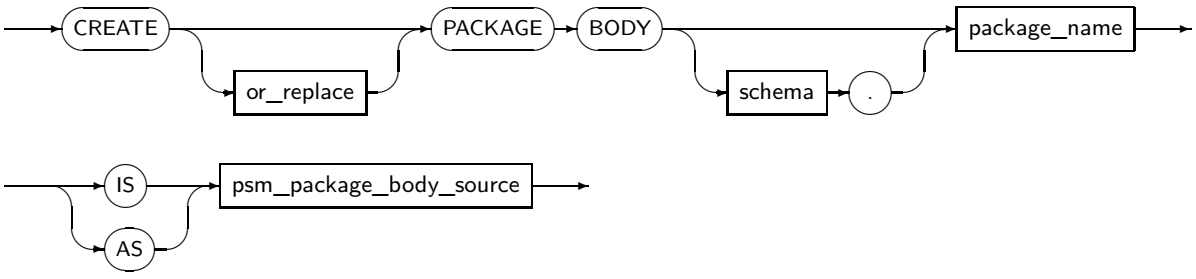
7.35. CREATE PACKAGE BODY

저장된 패키지의 본문을 생성한다.

CREATE PACKAGE BODY의 세부 내용은 다음과 같다.

- 문법

create_package_body



- 특권

  – 사용자의 스키마에 패키지를 생성하려면 CREATE PROCEDURE 시스템 특권이 있어야 한다.

– 다른 사용자의 스키마에 패키지를 생성하려면 CREATE ANY PROCEDURE 시스템 특권이 있어야 한다.

- 구성요소

구성요소	설명
OR REPLACE	OR REPLACE를 명시하면 이미 존재하는 패키지 본문이라도 다시 생성할 수 있다. 패키지 본문을 변경하고자 할 때 패키지 본문을 제거하고, 다시 생성하고, 스키마 객체 특권을 설정하는 단계를 반복하는 것을 피할 수 있다. 패키지 본문이 변경되면 패키지 본문을 컴파일해서 새로운 패키지 본문으로 적용한다.
schema	변경할 패키지가 속해 있는 스키마를 명시한다.
package_name	변경할 패키지의 이름을 명시한다.
IS	IS나 AS는 기호에 맞게 선택할 수 있으며, 둘의 차이는 없다. IS 다음에는 패키지의 본문이 온다.
AS	IS와 동일하다. AS 다음에는 패키지의 본문이 온다.
psm_package_body_source	패키지 본문을 지정한다. 자세한 내용은 "Tibero tbPSM 안내서"를 참고한다.

- 예제

다음은 **CREATE PACKAGE BODY**를 사용하는 예이다.

```
CREATE OR REPLACE PACKAGE BODY dept_pkg
  AS
  FUNCTION create (div_id NUMBER, loc_id NUMBER)
  RETURN NUMBER IS
  newid NUMBER;
  BEGIN
      SELECT dept_seq.NEXTVAL INTO newid FROM dual;
      INSERT INTO dept VALUES (newid, NULL, div_id, loc_id);
      RETURN (newid);
  END;
  PROCEDURE remove (dept_id NUMBER) IS
  BEGIN
      DELETE FROM dept WHERE dept.deptid = dept_id;
  END;
END dept_pkg;
```

7.36. CREATE PROCEDURE

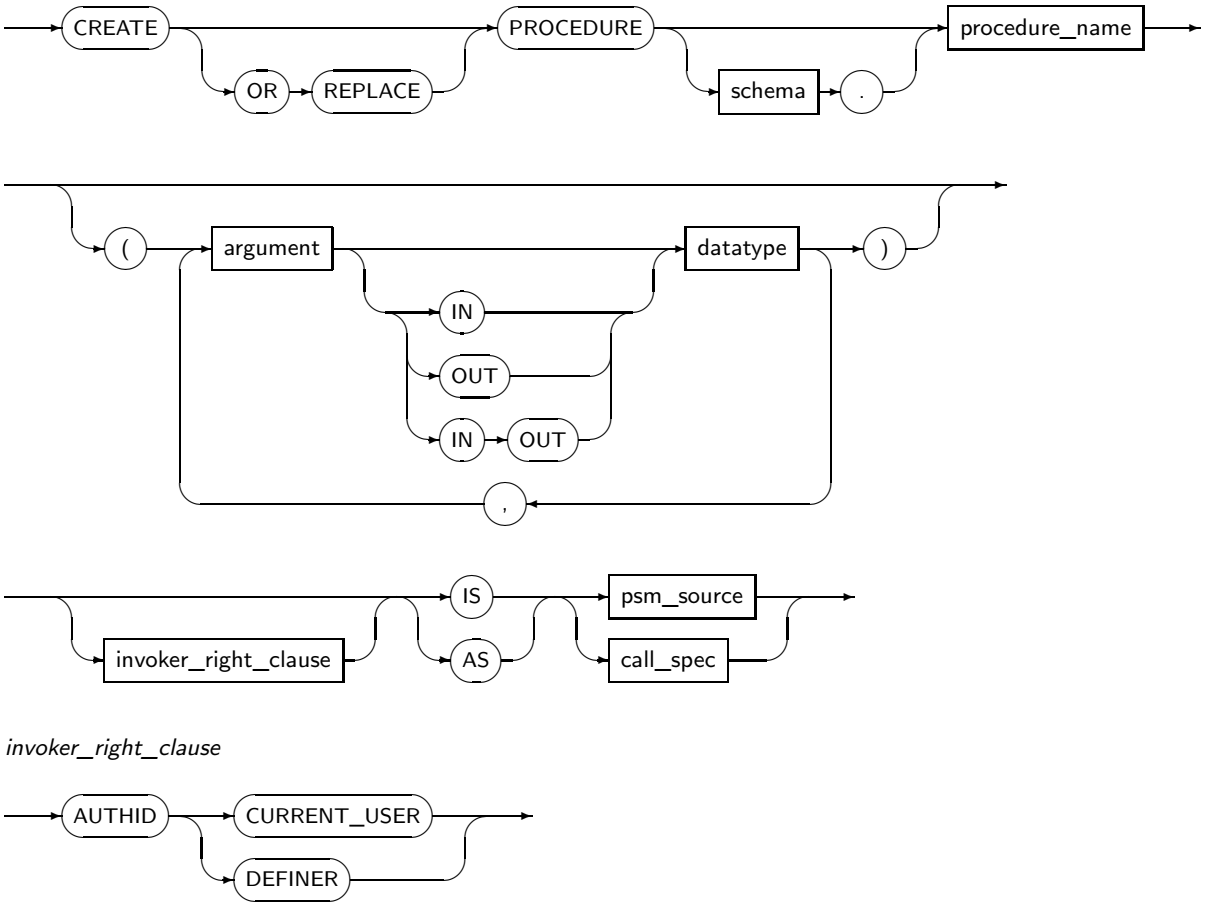
사용자 프러시저를 새로 정의하거나 기존의 프러시저로 대체한다. 사용자 프러시저는 tbPSM 프로그램이며, Tibero 서버에서 저장되고 실행된다.

사용자 프러시저가 사용자 함수와 다른 점은 반환값이 없으며 질의 또는 DML 문장에서 사용될 수 없다는 것이다. 사용자 프러시저는 다른 프러시저에서 직접 호출되거나 tbSQL 유틸리티에서 EXECUTE를 이용하여 호출한다.

CREATE PROCEDURE의 세부 내용은 다음과 같다.

- 문법

create_procedure



invoker_right_clause

- 특권
  - 사용자가 자신의 스키마에 프러시저를 생성하고자 하면 CREATE PROCEDURE 시스템 특권을 부여받아야 한다.
  - 다른 사용자의 스키마에 프러시저를 생성하거나 변경하고자 하면 CREATE ANY PROCEDURE 또는 ALTER ANY PROCEDURE 시스템 특권을 부여받아야 한다.

- 구성요소

  – create_procedure

구성요소	설명
OR REPLACE	이미 존재하는 프러시저를 다시 정의하고자 할 때 사용한다. OR REPLACE가 포함되면 해당 프러시저를 재컴파일한다. 프러시저를 제거하고 다시 생성하는 것과의 차이는 OR REPLACE를 이용하면 해당 프러시저에 기존의 특권이 그대로 유지된다는 점이다.
schema	프러시저가 속해 있는 스키마를 명시한다. 생략하면 현재 사용자의 스키마로 인식된다.
procedure_name	생성하고자 하는 프러시저의 이름을 명시한다.
argument	프러시저의 파라미터이다. 프러시저의 파라미터는 0개 이상이 될 수 있다. 파라미터가 0개이면 파라미터를 감싸는 괄호를 생략한다.
IN	프러시저의 파라미터의 전달 방향에 따른 구분이다. IN 파라미터는 외부로부터 값을 입력 받는다. 디폴트는 IN 파라미터이다. 프러시저에서는 현재 트랜잭션에 대한 COMMIT, ROLLBACK, SAVE POINT를 수행할 수 없고 DDL 문장도 실행할 수 없다. 또한, 현재 액세스 중인 테이블에 대한 갱신도 수행할 수 없다.
OUT	프러시저의 파라미터의 전달 방향에 따른 구분이다. OUT 파라미터는 프러시저 내부로부터 값을 출력한다. DML 문장에 포함된 프러시저는 OUT 파라미터를 가질 수 없다. DML 문장에 포함된 프러시저로부터 직접 또는 간접적으로 호출되는 프러시저는 OUT 또는 IN OUT 파라미터를 가질 수 있으나, 나머지는 불가능하다.
IN OUT	프러시저의 파라미터의 전달 방향에 따른 구분이다. IN OUT 파라미터는 입력과 출력 모두에 사용된다. DML 문장에 포함된 프러시저는 IN OUT 파라미터를 가질 수 없다.
NOCOPY	파라미터의 값을 가리키는 포인터(Pointer)를 전달한다. 파라미터의 값을 복사하지 않기 때문에 전달 속도가 빠르다.
datatype	프러시저의 파라미터의 데이터 타입이다. 프러시저의 파라미터 또는 반환값의 데이터 타입은 tbPSM에서 지원하는 모든 데이터 타입이 가능하다. 반환값의 데이터 타입을 선언할 때 문자열의 길이, 숫자의 정밀도와 스케일은 지정할 수 없다.
invoker_right_clause	프러시저를 호출자 권한 프러시저 또는 정의자(DEFINER) 권한 프러시저로 선언한다. 사용자 프러시저는 호출자 권한 프러시저 또는 정의자 권한

구성요소	설명
	프러시저로 구분할 수 있다. 이는 프러시저를 실행 할 때 어떤 사용자의 특권을 이용하여 실행할 것인지, 액세스하고자 하는 테이블 등의 객체는 어떤 스키마에서 찾을 것인지 등을 결정한다. 디폴트는 정의자 권한 프러시저이다.
IS	IS나 AS는 기호에 맞게 선택할 수 있으며, 둘의 차이는 없다. IS 다음에는 프러시저의 본문이 온다.
AS	IS와 동일하다. AS 다음에는 프러시저의 본문이 온다.
psm_source	PSM 소스 코드가 오는 부분이다. 자세한 내용은 "Tibero tbPSM 안내서"를 참고한다.
call_spec	외부 함수를 호출하기 위한 명세를 지정한다. 자세한 내용은 "Tibero tbPSM 안내서"를 참고한다.

– invoker_right_clause

구성요소	설명
AUTHID CURRECT_USER	프러시저를 호출자 권한 프러시저로 선언한다. 호출자 권한 프러시저로 선언하면, 현재 사용자의 특권을 이용하며 현재 사용자의 스키마에서 액세스할 객체를 찾는다. 따라서, 프러시저를 호출하는 사용자가 달라지면 가능한 작업의 범위와 액세스하는 스키마 객체도 달라진다. 이러한 프러시저는 여러 사용자가 공통으로 동일한 작업을 수행하고자 할 때에 유용하다.
AUTHID DEFINER	프러시저를 정의자 권한 프러시저로 선언한다. 정의자 권한 프러시저로 선언하면, 프러시저를 정의한 사용자의 특권을 이용하고 그 사용자의 스키마에서 액세스할 객체를 찾는다. 따라서, 호출자에 관계없이 작업의 범위와 액세스하는 스키마 객체가 항상 일정하다. 이러한 프러시저는 데이터 사전 등의 시스템 데이터의 일부를 일반 사용자가 액세스할 때에 유용하다.

- 예제

  다음은 **CREATE PROCEDURE**를 사용해 사용자 프러시저를 생성하는 예이다.

```
CREATE OR REPLACE PROCEDURE raise_salary (deptno NUMBER)
  AS
  BEGIN
  UPDATE EMP SET SALARY = SALARY * 1.05
  WHERE DEPTNO = deptno;
  END;
```

7.37. CREATE PROFILE

새로운 프로파일을 생성한다.

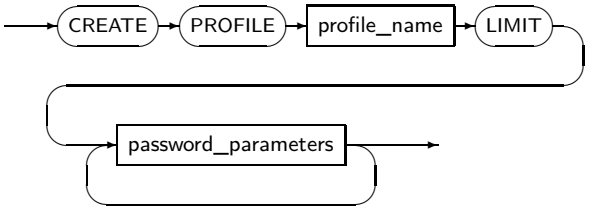
참고

프로파일의 정보를 변경하거나 제거하기 위해서는 “7.10. ALTER PROFILE”, “7.58. DROP PROFILE”의 내용을 참고한다.

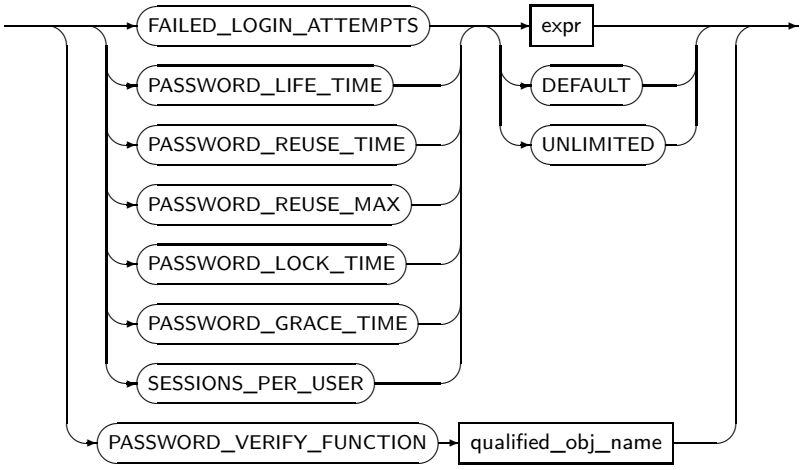
CREATE PROFILE의 세부 내용은 다음과 같다.

- 문법

create_profile



password_parameters



- 특권

  프로파일을 생성하기 위해서는 CREATE PROFILE 시스템 특권이 필요하다.

- 구성요소

  – create_profile

구성요소	설명
profile_name	생성할 프로파일의 이름이다.

구성요소	설명
	프로파일 이름은 VARCHAR 데이터 타입으로 저장되고 길이는 최대 255자까지 가능하다.
password_parameters	사용자 계정 접속 및 패스워드 관련 세부 설정 내용이다.

– password_parameters

구성요소	설명
FAILED_LOGIN_ATTEMPTS	잘못된 패스워드로 로그인 시도할 경우 사용자 계정를 잠글 때까지 로그인 시도 허용 횟수를 지정한다.
LOGIN_PERIOD	이 값이 지정되어 있을 경우 마지막 로그인 후 지정된 시간이 지나면 계정인 잠긴다.
PASSWORD_LIFE_TIME	패스워드 만료 기간을 설정한다. 숫자와 수식 두 가지 형태로 지정할 수 있다. – 숫자로 지정할 경우 단위는 일(day)이다. 예를 들어 30으로 지정하면 패스워드 만료 일자가 30일 후가 된다. – 해당 값을 1일 미만으로 지정하는 등 특별한 용도로 사용하기 위해 수식을 사용할 수 있다. 예를 들어 1/1440으로 지정하면 패스워드 만료 일자가 1분 후가 된다.
PASSWORD_REUSE_TIME	패스워드 재사용 금지 기간을 설정한다. 숫자와 수식 두 가지 형태로 지정할 수 있다. 예를 들어 해당 값을 30으로 설정하면 30일동안 동일한 패스워드로 다시 변경할 수 없다.
PASSWORD_REUSE_MAX	설정된 갯수만큼 최근 변경한 패스워드는 재사용할 수 없다. 예를 들어 해당 값을 10으로 지정하면 현재 패스워드를 재사용하기 위해서는 10회 이상 다른 값으로 먼저 패스워드를 변경해야 한다.
PASSWORD_LOCK_TIME	패스워드 오류 횟수 초과로 계정이 잠금 상태가 되었을 때 자동으로 잠금 상태를 해제하는 기간을 설정한다. 숫자와 수식 두 가지 형태로 지정할 수 있다. 예를 들어 1/1440으로 지정하면 1분 후 자동으로 잠금 상태가 해제된다.
PASSWORD_GRACE_TIME	패스워드 사용기간(PASSWORD_LIFE_TIME) 만료 후 패스워드 만료 경고를 보내는 기간을 설정한다. 숫자와 수식 두 가지 형태로 지정할 수 있다. 경고를 보내는 기간은 패스워드 사용기간 만료후 첫 접속한 시점부터 계산한다. 예를 들어 PASSWORD_LIFE_TIME을 30, PASS

구성요소	설명
	WORD_GRACE_TIME을 3으로 설정하면 30일이 지난 후 첫 접속부터 3일간 패스워드 만료 경고가 나타나고 다시 3일이 지나면 계정이 잠금 상태가 된다. 패스워드 만료 경고는 tbsql에서만 동작하며 다른 방식으로 접속하는 경우(OCI, JDBC 등) 패스워드 만료 에러를 반환한다.
SESSIONS_PER_USER	유저당 접속 가능한 세션 수를 제한한다.
PASSWORD_VERIFY_FUNCTION	패스워드를 변경할 때 패스워드 문자열을 검사하여 유효성을 확인하는 PSM 함수를 지정한다. PSM 함수는 기본으로 들어있는 함수를 지정하거나 다른 함수를 정의해서 지정할 수 있다. 아무것도 지정하지 않으면 데이터베이스를 생성할 때 미리 만들어진 NULL_VERIFY_FUNCTION이라는 함수를 기본으로 사용한다. 이 함수는 패스워드에 아무런 제약을 두지 않는다.

- 예제

  – create_profile

    다음은 프로파일을 생성하는 예이다.

    이 예제대로 프로파일을 생성하면 해당 프로파일을 사용하는 사용자는 패스워드 입력 오류를 3회 발생한 경우 1분간 계정 잠금 상태가 되며, 패스워드 유효기간은 90일이고 패스워드 변경시 최근 10개의 패스워드는 재사용할 수 없다. 또한, 패스워드 만료후 10일간 패스워드 만료 경고를 받게된다. 패스워드를 변경할 때 **verify_function**이라는 검증 함수로 패스워드의 유효성을 검사하게된다.

```
SQL> CREATE PROFILE prof LIMIT
      failed_login_attempts 3
      password_lock_time 1/1440
      password_life_time 90
      password_reuse_time 30
      password_reuse_max 10
      password_grace_time 10
      password_verify_function verify_function;
Profile 'PROF' created.
```

    아무런 limit도 지정하지 않는 경우(다음과 같이 LIMIT 절을 지정하지 않은 경우에는) 데이터베이스를 생성할 때 만들어진 프로파일인 'DEFAULT'와 동일한 limit으로 프로파일이 생성된다.

참고

SYS 사용자에 한하여 profile이 적용되어도 패스워드 기한만료, 입력 오류 등으로 인해 계정이 잠금 상태가 되지 않는다.

```
SQL> CREATE PROFILE prof
Profile 'PROF' created.
```

그리고 파라미터들 중 일부만 지정한 경우에는 지정한 값을 제외한 나머지 값들이 'DEFAULT' 프로파일과 동일한 값으로 생성된다. 즉, 'DEFAULT'라는 이름의 프로파일은 다른 프로파일을 생성할 때 기본값으로 참조되는 값이며 이 기본값을 조정하고 싶으면 ALTER PROFILE 명령을 이용하면 된다.

참고

DBA_PROFILES 뷰를 통해 각 프로파일별 파라미터 값을 조회할 수 있다.

참고

PASSWORD_REUSE_TIME, PASSWORD_REUSE_MAX 두 파라미터는 서로 함께 설정되어야 한다. 사용자가 패스워드를 재사용하려면 패스워드를 PASSWORD_REUSE_MAX 값만큼 변경했으면서 동시에 PASSWORD_REUSE_TIME 값만큼 날짜가 지나야 한다. 두 파라미터 중 하나를 UNLIMITED로 설정하면 사용자는 패스워드를 재사용할 수 없다.

7.38. CREATE ROLE

특권의 집합인 역할을 생성한다. 역할은 특권이나 다른 역할을 포함할 수 있다. 하지만, CREATE ROLE 문장으로 만들어진 역할은 아무런 특권이나 집합을 포함하지 않은 상태로 생성된다. 만들어진 역할에 어떠한 특권이나 역할을 포함하고자 할 때는 GRANT를 사용한다.

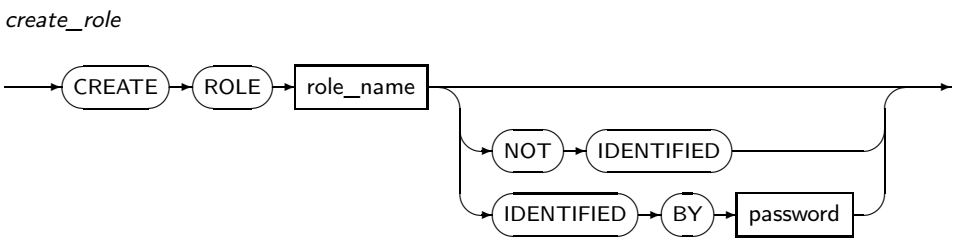
참고

1. 생성한 역할의 정보를 변경하거나 제거하고자 할 때는 “7.11. ALTER ROLE”, “7.59. DROP ROLE”을 참고한다.

2. 생성한 역할을 사용하는 방법은 “7.71. GRANT”, “7.75. REVOKE”, “9.7. SET ROLE”을 참고한다.

CREATE ROLE의 세부 내용은 다음과 같다.

- 문법



- 특권

  CREATE ROLE 시스템 특권이 필요하다.

- 구성요소

구성요소	설명
role_name	생성할 역할의 이름이다. 역할의 이름은 다음과 같은 특징이 있다. – VARCHAR 데이터 타입으로 저장되고, 길이는 최대 30자까지 가능하다. – 전체 데이터베이스의 다른 역할 또는 사용자 이름과 중복되면 안 된다.
NOT IDENTIFIED	생성하는 역할에 패스워드를 사용하지 않는다. 기본값이므로 이 부분을 생략하면 NOT IDENTIFIED로 인식된다.
IDENTIFIED BY	생성하는 역할에 패스워드를 설정한다. 패스워드가 설정된 역할은 SET ROLE 문장을 사용해 역할을 활성화시키려 할 때에, 설정된 패스워드를 입력해야 한다. 패스워드 사용에 대한 자세한 내용은 “9.7. SET ROLE”을 참고한다.
password	역할에 설정할 패스워드를 명시한다.

- 예제

  다음은 **NOT IDENTIFIED**를 사용해 역할을 생성하는 예이다.

```
SQL> CONN sys/tibero
Connected.

SQL> CREATE ROLE a;
Role created.

SQL> CREATE ROLE b NOT IDENTIFIED;
Role created.

SQL> CREATE USER u1 IDENTIFIED BY 'p1';
User created.

SQL> GRANT CREATE SESSION TO a;
Granted.

SQL> GRANT a, b TO u1;
Granted.

SQL> CONN u1/p1
Connected.

SQL> SET ROLE a;
Set.
```

```
SQL> SET ROLE b;
Set.
```

위의 예와 같이, NOT IDENTIFIED 절을 사용해 역할을 생성하거나 모두 생략해서 만든 역할은 아무런 제약 없이 SET ROLE 문장에서 사용할 수 있다.

다음은 역할을 생성할 때 **IDENTIFIED**를 사용해 패스워드를 설정하는 예이다.

```
SQL> CONN sys/tibero
Connected.

SQL> CREATE ROLE c IDENTIFIED BY 'abc';
Role created.

SQL> GRANT CREATE SESSION TO c;
Granted.

SQL> GRANT c TO u1;
Granted.

SQL> CONN u1/p1
Connected.

SQL> SET ROLE c;
TBR-7181: need password to enable the role

SQL> SET ROLE c IDENTIFIED BY 'abc';
Set.
```

IDENTIFIED를 사용하여 패스워드를 설정한 역할은 NOT IDENTIFIED를 사용한 것과 다르게 동작한다. 위의 예와 같이 SET ROLE 문장에서 역할을 활성화할 때에 생성할 때 지정한 패스워드를 입력해 주어야만 동작하는 것을 볼 수 있다.

7.39. CREATE SEQUENCE

사용자가 소유한 스키마 또는 지정된 스키마에 포함되는 시퀀스를 생성한다.

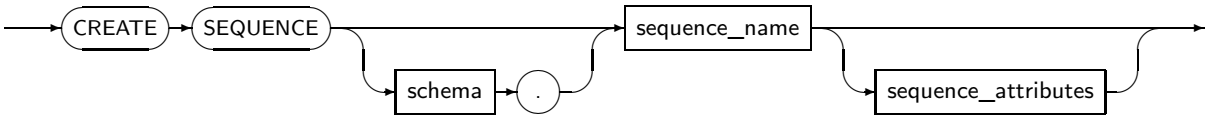
참고

시퀀스를 변경, 제거하기 위해서는 “7.13. ALTER SEQUENCE”와 “7.60. DROP SEQUENCE”의 내용을 참고한다.

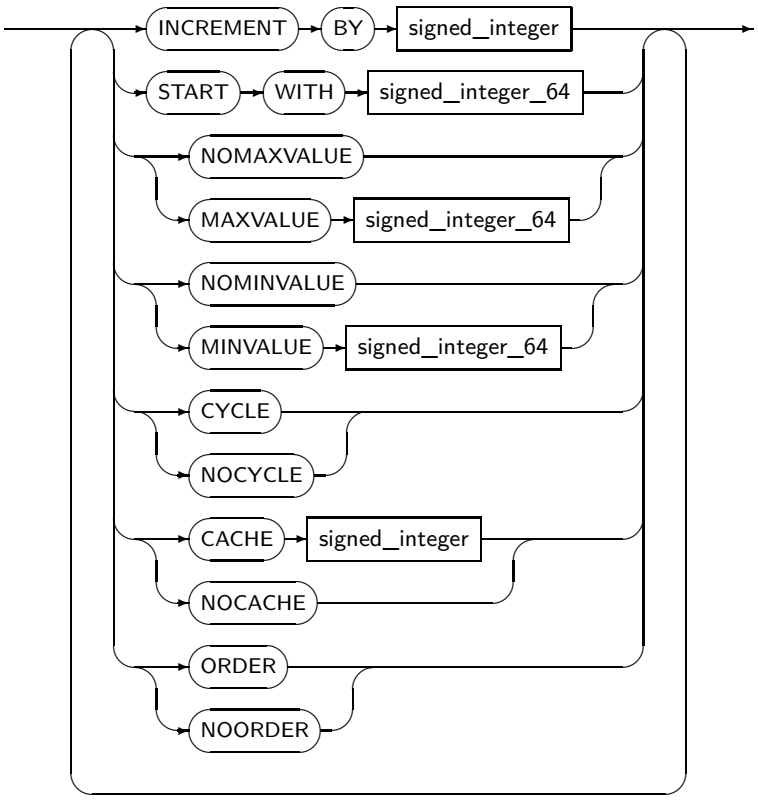
CREATE SEQUENCE의 세부 내용은 다음과 같다.

- 문법

create_sequence



sequence_attributes



- 특권
  - 시퀀스를 생성하기 위해서는 CREATE SEQUENCE 시스템 특권이 필요하다.
  - 다른 사용자가 소유한 스키마의 시퀀스를 생성하기 위해서는 CREATE ANY SEQUENCE 시스템 특권이 필요하다.
- 구성요소
  - create_sequence

구성요소	설명
schema	생성할 시퀀스를 포함하는 스키마의 이름이다. 생략하면 현재 사용자의 스키마로 인식된다.
sequence_name	생성할 시퀀스의 이름이다. 시퀀스의 이름은 다음과 같은 특징이 있다. – VARCHAR 데이터 타입으로 저장되고, 길이는 최대 30자까지 가능하다.

구성요소	설명
	– 시퀀스의 이름은 테이블과 같은 네임스페이스를 사용하므로 스키마의 다른 시퀀스, 테이블, 동의어, PSM의 이름과 중복되면 안 된다.
sequence_attributes	시퀀스의 속성을 정의하는 부분으로 생략할 수 있다.

– sequence_attributes

구성요소	설명
INCREMET BY	시퀀스의 간격을 지정한다. 이 값을 양수로 입력하면 시퀀스의 값이 증가하게 되고, 음수로 입력하면 시퀀스의 값이 감소하게 된다. 지정하지 않으면 디폴트 1로 간격이 지정된다. 이 값은 MAXVALUE에서 MINVALUE를 뺀 값보다 클 수 없다.
START WITH	시퀀스의 시작 값을 지정한다. 시작 값은 MINVALUE와 MAXVALUE 사이의 값을 지정할 수 있다. 시작 값을 지정하지 않으면 INCREMENT BY값이 양수인 경우 MINVALUE가, INCREMENT BY값이 음수인 경우 MAXVALUE가 시작 값이 된다.
NOMAXVALUE	MAXVALUE를 지정하지 않는 것과 같다. MAXVALUE는 시퀀스의 증가 값이 양수인 경우 시퀀스가 가질 수 있는 가장 큰 값인 INT64_MAX가 되고 증가 값이 음수인 경우 -1이 된다.
MAXVALUE	MAXVALUE의 값을 지정한다. MAXVALUE는 MINVALUE보다 작아서는 안 된다. 지정하지 않으면 NOMAXVALUE와 같은 값으로 동작한다.
NOMINVALUE	MINVALUE를 지정하지 않는 것과 같다. MINVALUE는 시퀀스의 증가 값이 양수인 경우 1의 값을 가지고, 증가 값이 음수인 경우 시퀀스가 가질 수 있는 가장 작은 값인 INT64_MIN이 된다.
MINVALUE	MINVALUE의 값을 지정한다. MINVALUE는 MAXVALUE보다 클 수 없다. 지정하지 않으면 NOMINVALUE와 같은 값으로 동작한다.
CYCLE	CYCLE이 지정되면 MAXVALUE나 MINVALUE에 도달해도 계속 시퀀스 값을 생성한다. 증가 값이 양수일 때 시퀀스의 값이 MAXVALUE를 넘게 되면 MINVALUE 값을 주게 되고, 증가 값이 음수일 때 시퀀스 값이 MINVALUE보다 작아지면 MAXVALUE를 주게 된다.
NOCYCLE	CYCLE과 반대로 MAXVALUE 또는 MINVALUE에 도달하게 되면 더 이상 값을 만들지 않는다. CYCLE이나 NOCYCLE 둘 다 지정하지 않으면 NOCYCLE이 기본값이 된다.

구성요소	설명
CACHE	시퀀스의 속도를 높이기 위해 시퀀스 캐시에 지정된 개수만큼의 시퀀스 값을 저장해 두고 시퀀스 값을 요청하면 시퀀스 캐시에서 받아오게 된다. 저장해 둔 시퀀스 값을 다 사용하게 되면 새로 그만큼 더 받아오게 되며, 이때만 데이터 사전의 값을 수정하게 된다. 즉, CACHE로 임의의 수 N을 지정하였다면, NOCACHE로 사용할 때보다 1/N의 데이터 사전 수정이 이루어지게 되므로 속도를 높일 수 있다. 단 비정상 종료가 발생한 경우 캐시에 저장된 값은 없어진다. (기본값: 20)
NOCACHE	시퀀스 캐시를 사용하지 않고 매번 직접 값을 받아온다. 따라서 매번 데이터 사전을 수정하게 된다.
ORDER	클러스터 환경에서 노드 간에 발급되는 시퀀스 값의 순서를 유지한다. 노드에 관계 없이 요청된 순서대로 시퀀스 값이 발급된다. 클러스터 환경으로 설정한 경우에만 지정할 수 있다.
NOORDER	ORDER와 반대로 클러스터 환경에서 노드 간에 발급되는 시퀀스 값의 순서를 유지하지 않는다. 노드별로 각각의 시퀀스 캐시를 유지하므로 한 노드 안에서는 요청된 순서대로 시퀀스 값이 발급되지만, 전체 노드에서는 요청된 순서와 발급되는 시퀀스 값의 순서가 다를 수 있다. ORDER나 NOORDER 둘 다 지정하지 않으면 NOORDER가 기본값이 된다.

- 예제

  다음은 **CREATE SEQUENCE**를 사용해 TEST_SEQ라는 시퀀스를 생성하는 예이다.

```
SQL> CREATE SEQUENCE test_seq
       START WITH 100
       MINVALUE 40
       MAXVALUE 200
       INCREMENT BY 2
       CYCLE;

Sequence created.

SQL> SELECT test_seq.NEXTVAL FROM dual;

TEST_SEQ.NEXTVAL
----------------
             100

1 row selected.
```

```
SQL> SELECT test_seq.NEXTVAL FROM dual;

TEST_SEQ.NEXTVAL
----------------
             102

1 row selected.
```

위의 예에서 TEST_SEQ는 40 ~ 200까지의 값을 가지는 시퀀스이다. TEST_SEQ.nextval을 사용하여 처음 값을 얻으면 시작 값인 100이 출력된다. 그다음부터는 2씩 증가하게 된다. 200에 도달하면 CYCLE 옵션 때문에 다시 40으로 돌아가게 된다.

7.40. CREATE SYNONYM

사용자가 소유한 스키마나 다른 사용자가 소유한 스키마에 속하는 동의어를 생성한다.

동의어를 생성할 수 있는 스키마 객체는 다음과 같다.

- 테이블
- 뷰
- 시퀀스
- PSM 함수
- PSM 프러시저
- 동의어

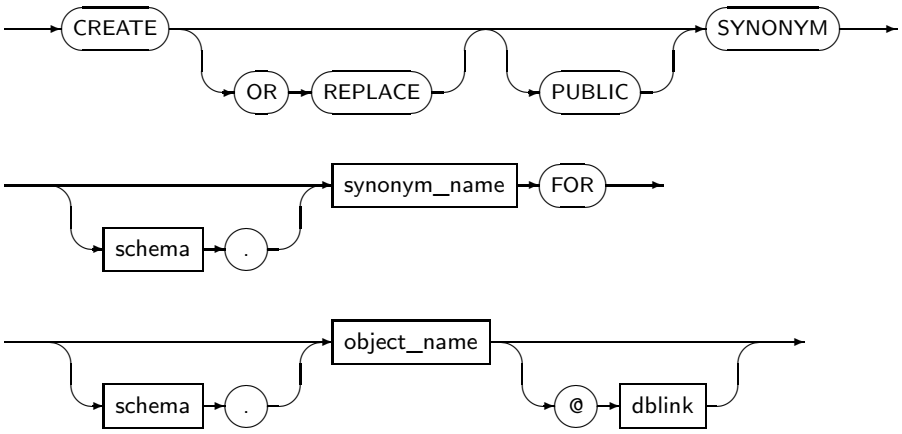
참고

동의어를 제거하기 위해서는 “7.61. DROP SYNONYM”의 내용을 참고한다.

CREATE SYNONYM의 세부 내용은 다음과 같다.

- 문법

create_synonym



- 특권
  - 사용자가 소유한 스키마의 동의어를 생성하기 위해서는 CREATE SYNOYM 시스템 특권이 필요하다.
  - 다른 사용자가 소유한 스키마의 동의어를 생성하기 위해서는 CREATE ANY SYNONYM 시스템 특권이 필요하다.
  - 공용 동의어를 생성하기 위해서는 CREATE PUBLIC SYNONYM 특권이 필요하다.
- 구성요소

구성요소	설명
OR REPALCE	생성할 동의어가 기존에 존재하는 동의어라면, 그 동의어를 제거하고 새로 생성한다. 동의어를 삭제하고 다시 생성하는 것과의 차이는 대상 동의어에 관련된 기존의 특권과 참조가 그대로 유지된다는 점이다.
PUBLIC	공용 동의어를 만든다. 공용 동의어는 모든 사용자가 접근할 수 있지만, 공용 동의어가 가리키는 객체를 사용하기 위해서는 해당 객체에 대한 권한이 있어야 한다.
schema	생성할 동의어를 포함하는 스키마의 이름이다. 생략하면 현재 사용자의 스키마로 인식된다.
synonym_name	생성할 동의어의 이름이다. 동의어의 이름은 VARCHAR 데이터 타입으로 저장되고, 길이는 최대 30자까지 가능하다. 동의어의 이름은 테이블과 같은 네임스페이스를 사용하므로 스키마의 다른 동의어, 공용 동의어, 테이블, 뷰, 시퀀스, 함수와 프러시저의 이름과 중복되면 안 된다.
FOR schema	FOR는 동의어가 가리키는 대상을 설정한다. schema는 생성할 동의어의 대상이 되는 객체를 포함하는 스키마의 이름이다. 생략하면 dblink의 명시 여부에 따라 디폴트 값이 달라진다. dblink를 명시

구성요소	설명
	했을 경우 dblink의 접속 사용자 이름으로 인식되고, 명시안했을 경우 현재 사용자의 스키마로 인식된다. 만약 명시한 dblink가 존재하지 않거나 사용할 수 없는 경우에는 null로 인식된다.
object_name	생성할 동의어의 대상이 되는 객체의 이름이다. 동의어가 가리키는 대상이 꼭 존재하지 않아도 되고, 접근할 수 있는 권한이 필요하지도 않다. 하지만, 그러한 경우에는 동의어를 사용하여 접근할 때 에러가 발생하게 된다.
dblink	데이터베이스 링크로 연결된 객체에 동의어를 생성할 때 데이터베이스 링크의 이름을 명시한다. 데이터베이스 링크 앞에는 항상 '@'를 붙여야 한다. 데이터베이스 링크에 대한 자세한 내용은 "Tibero 관리자 안내서"를 참고한다.

- 예제

다음은 **CREATE SYNONYM**을 사용해 동의어를 생성하는 예이다. 본 예제에서는 사용자(또는 스키마) tibero에 속한 emp라는 동의를 생성한다.

```
SQL> CREATE OR REPLACE SYNONYM emp FOR tibero.emp;

Created.
```

다음은 **CREATE PUBLIC SYNONYM**을 사용해 공용 동의어를 생성하는 예이다. 본 예제에서는 사용자 SYS에 속한 t에 접근할 수 있는 pt라는 공용 동의어를 생성한다.

```
SQL> CREATE PUBLIC SYNONYM pt FOR sys.t;

Synonym created.
```

공용 동의어 pt가 생성되면 모든 사용자가 SYS의 t에 접근할 수 있지만 t를 사용하기 위해서는 사용자가 그에 해당하는 특권을 가지고 있어야 한다.

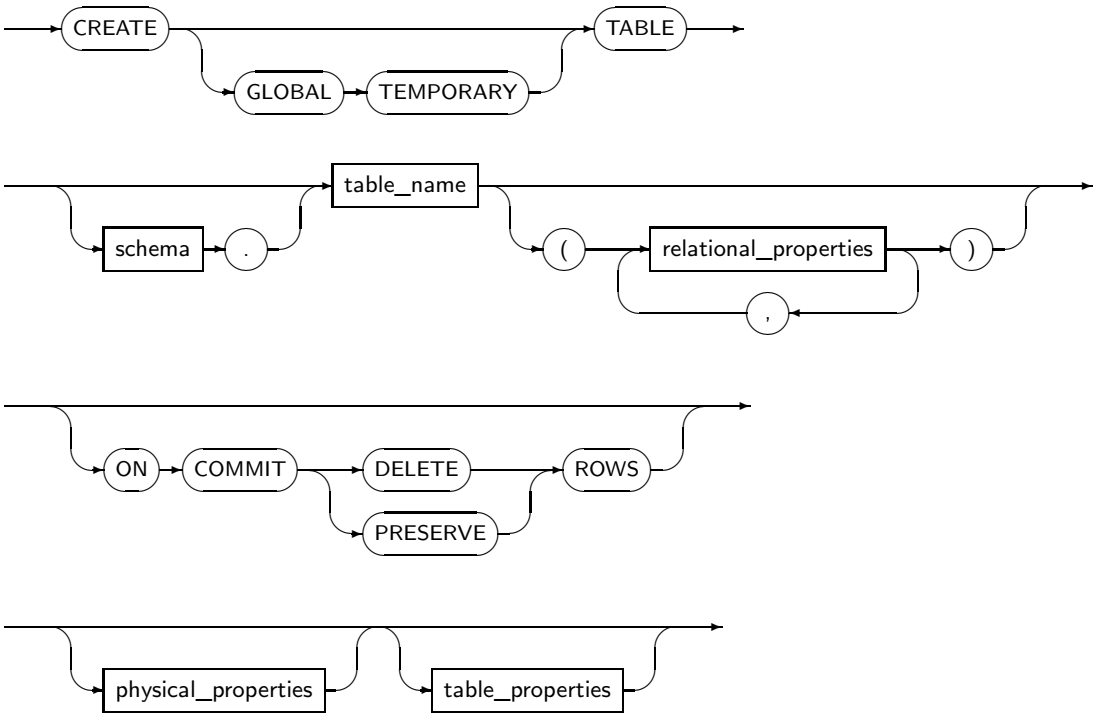
7.41. CREATE TABLE

테이블을 생성한다.

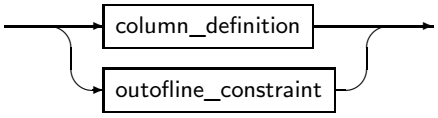
CREATE TABLE의 세부 내용은 다음과 같다.

- 문법

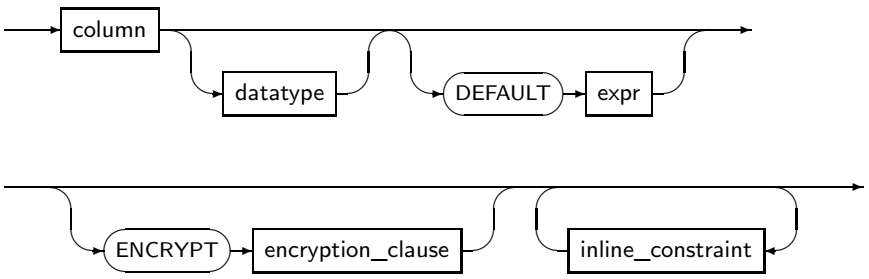
create_table



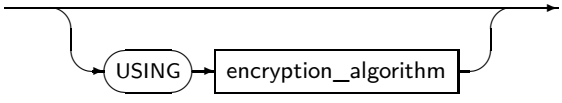
relational_properties



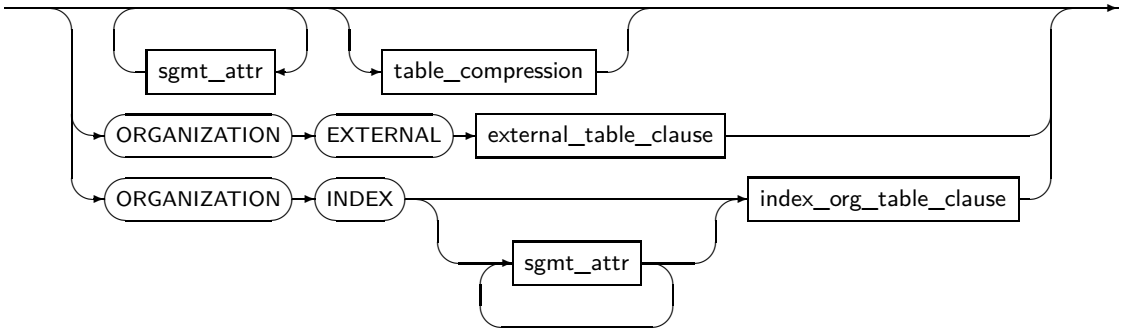
coldef



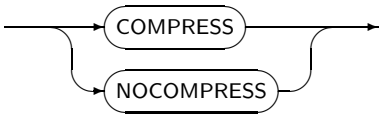
encryption_spec



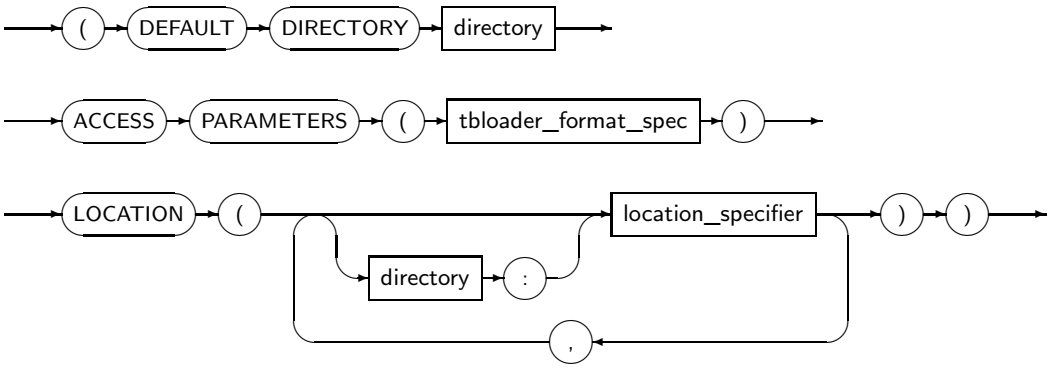
physical_properties



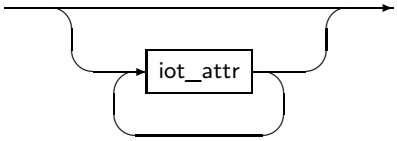
table_compression

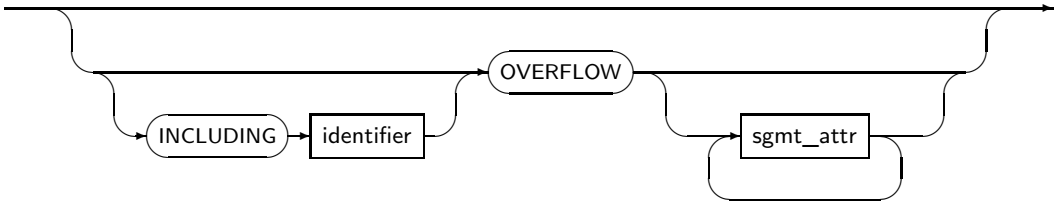


external_table_clause

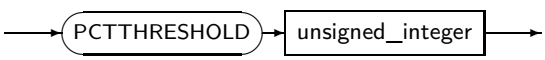


index_org_table_clause

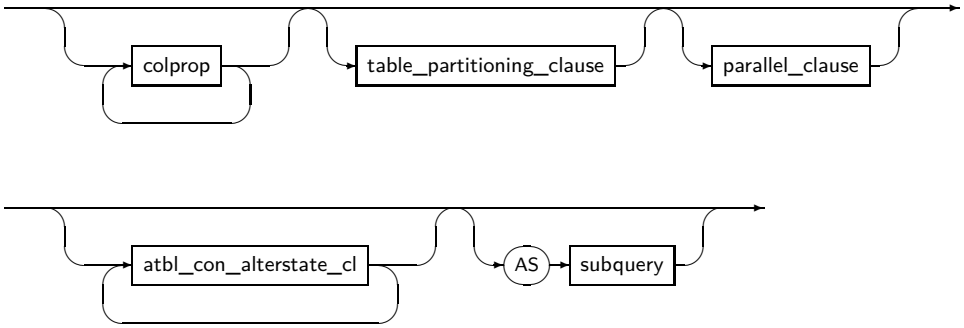




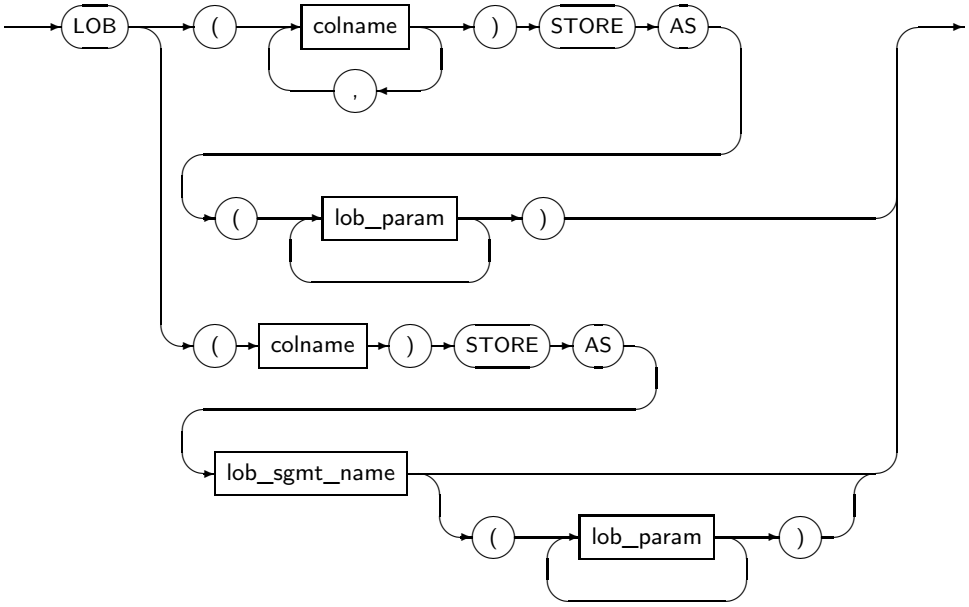
iot_attr



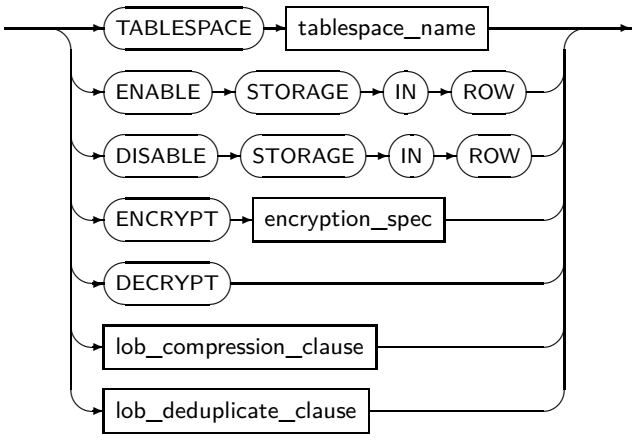
table_properties



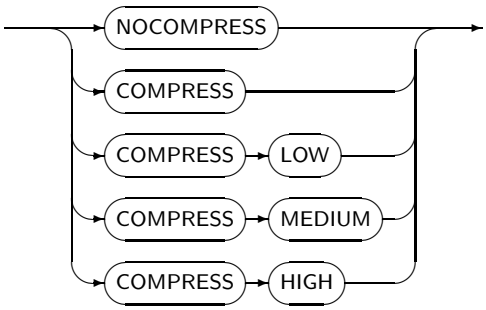
colprop



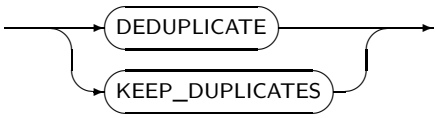
lob_param



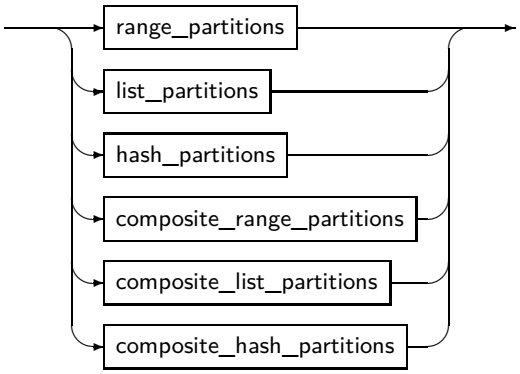
lob_compression_clause



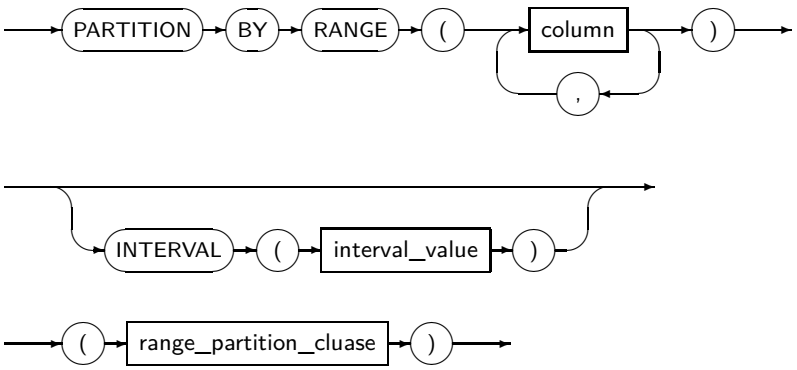
lob_deduplicate_clause



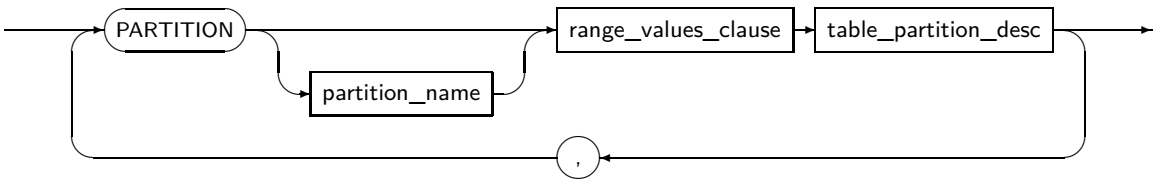
table_partitioning_clause



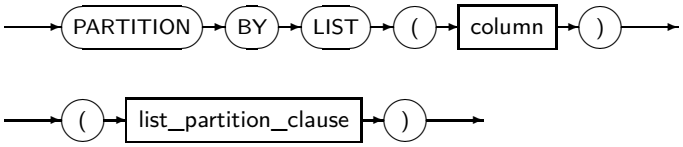
range_partitions



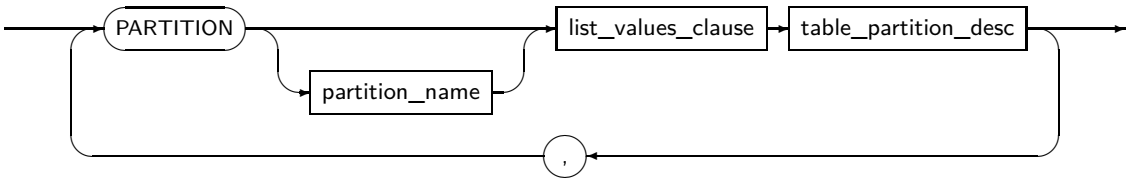
range_partition_clause



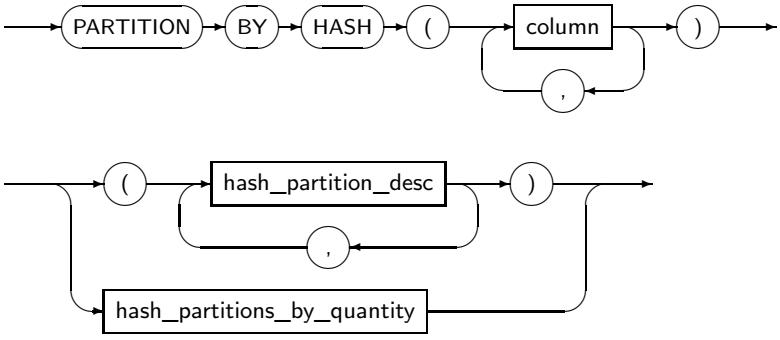
list_partitions



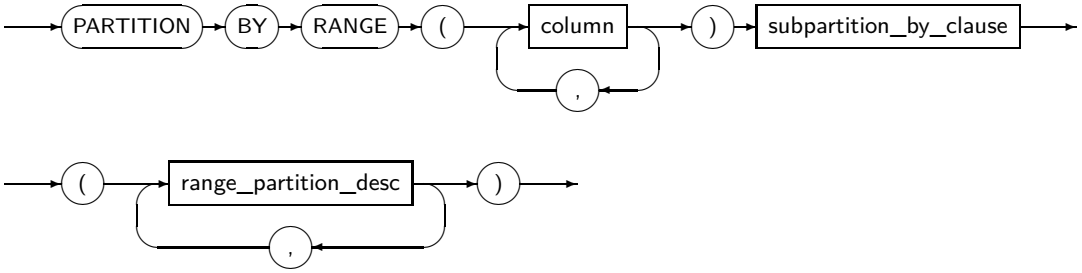
list_partition_clause



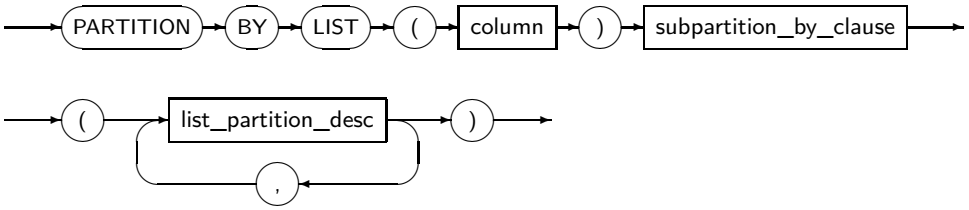
hash_partitions



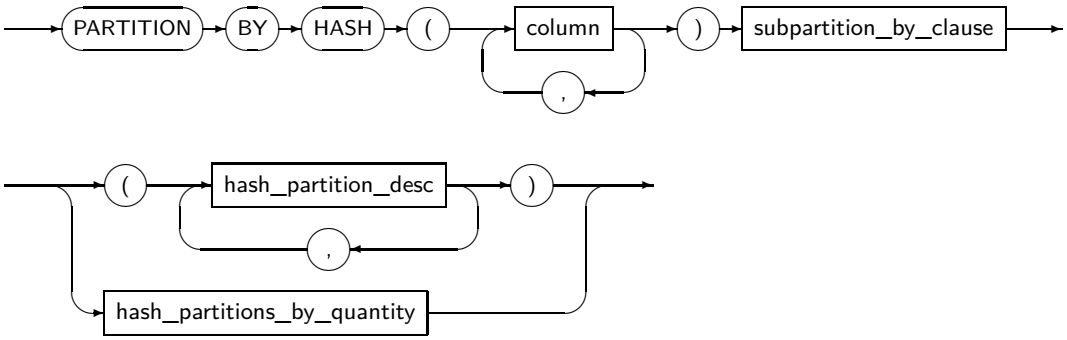
composite_range_partitions



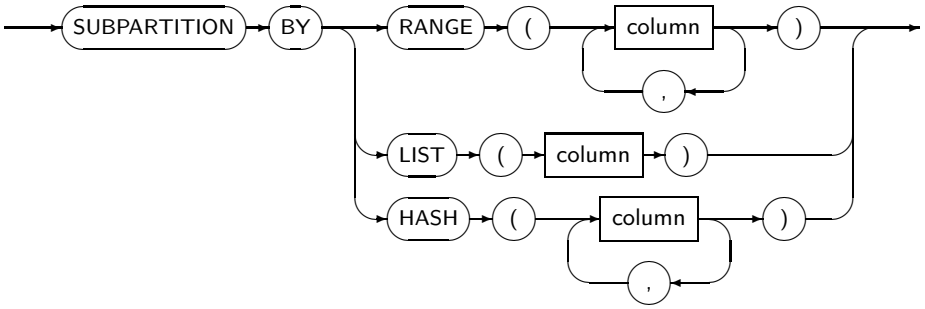
composite_list_partitions

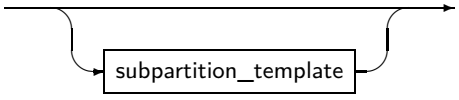


composite_hash_partitions

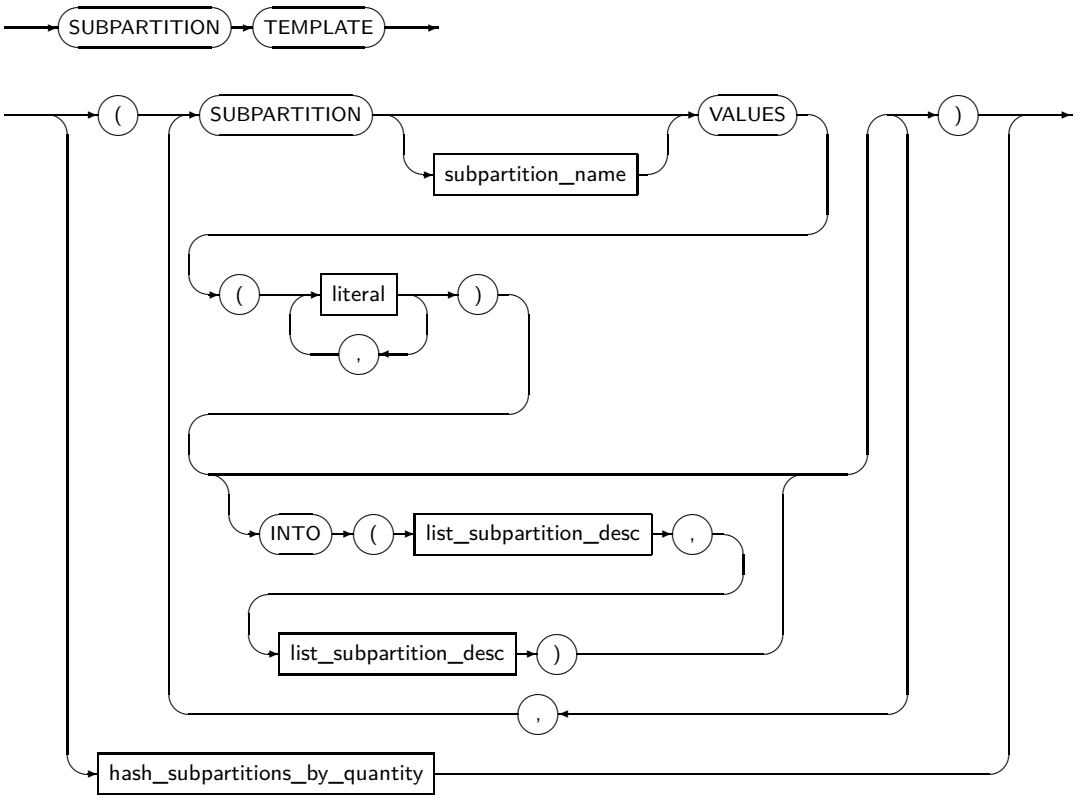


subpartition_by_clause

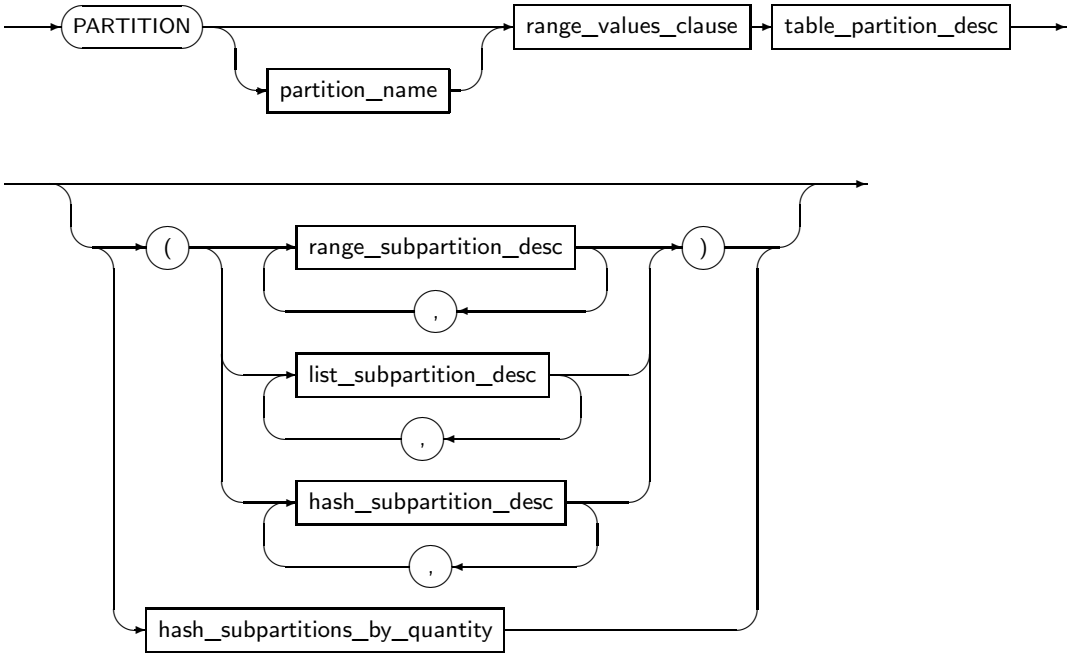




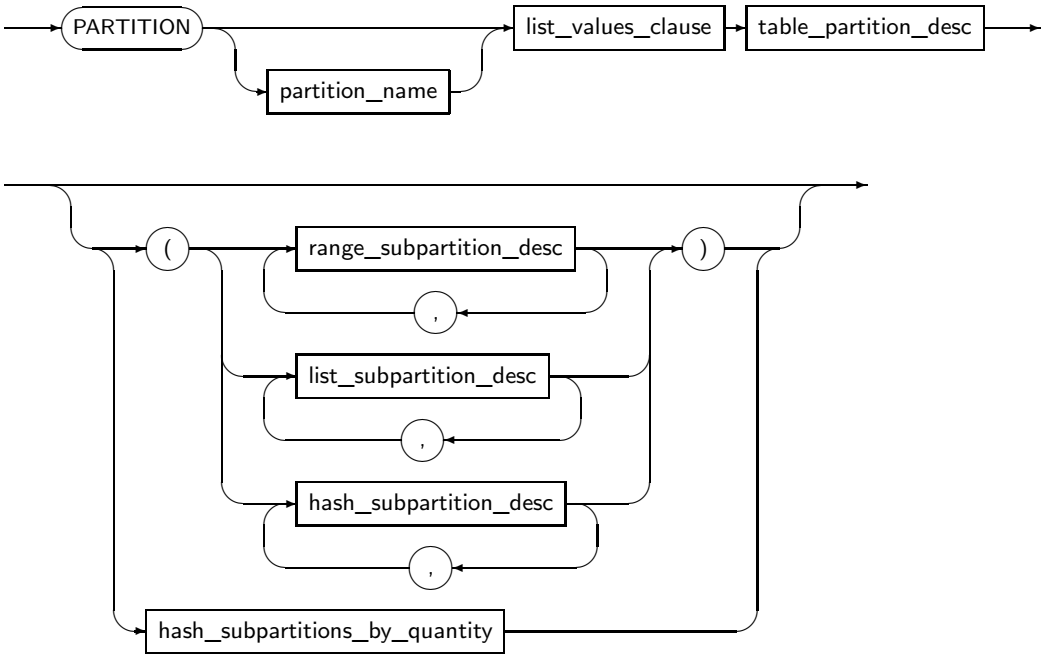
subpartition_template



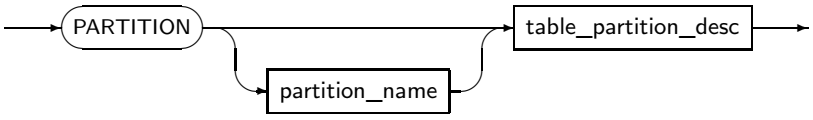
range_partition_desc



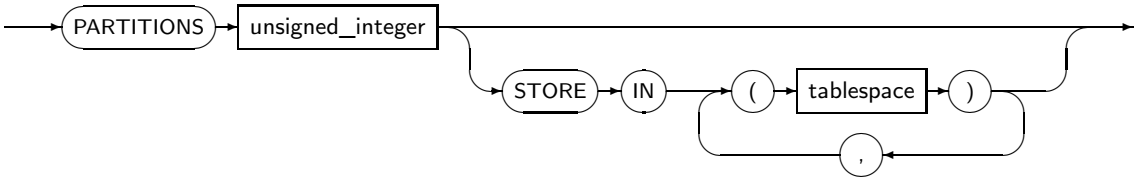
list_partition_desc



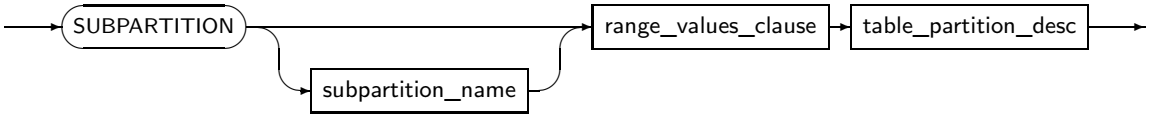
hash_partition_desc



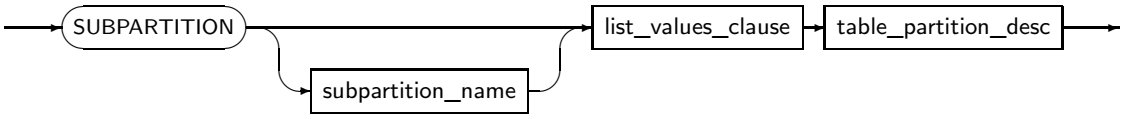
hash_partitions_by_quantity



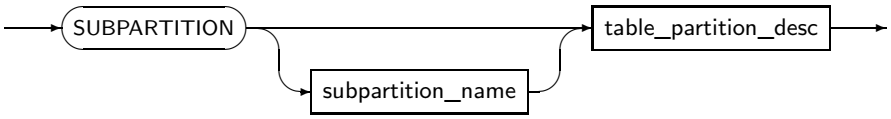
range_subpartition_desc



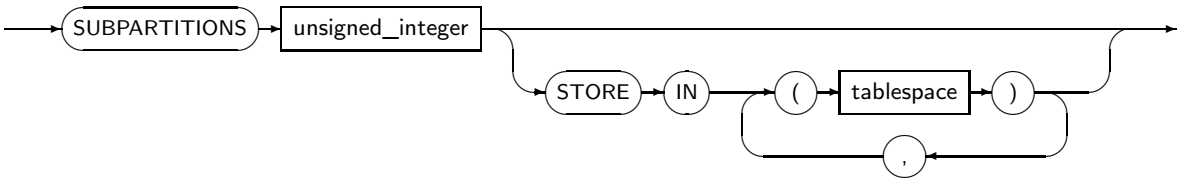
list_subpartition_desc



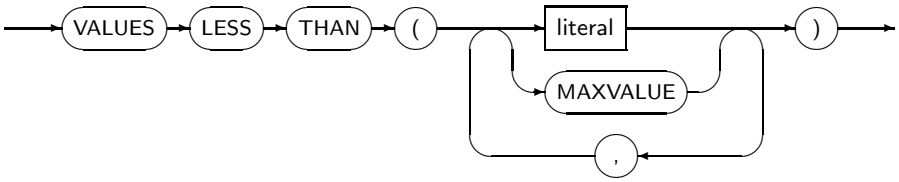
hash_subpartition_desc



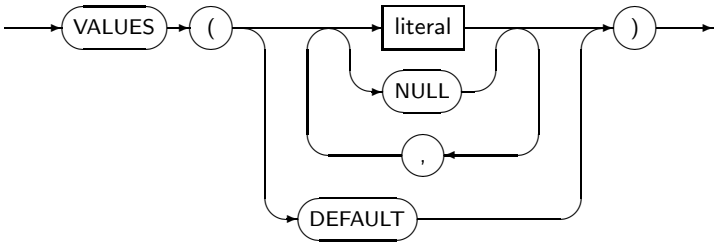
hash_subpartitions_by_quantity



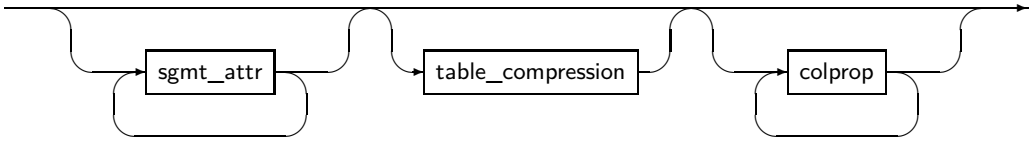
range_value_clause



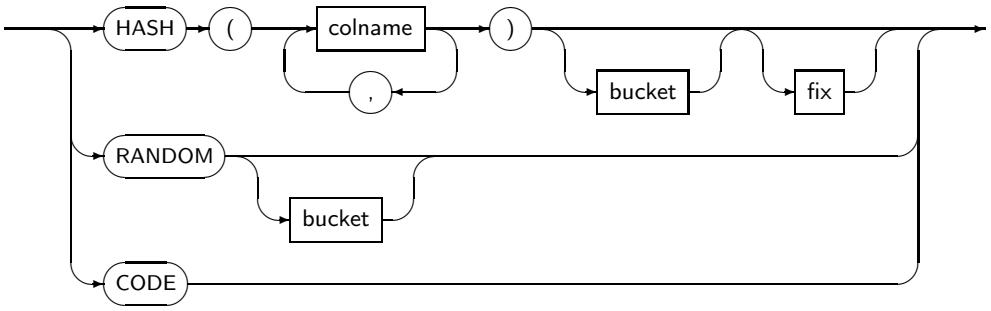
list_value_clause



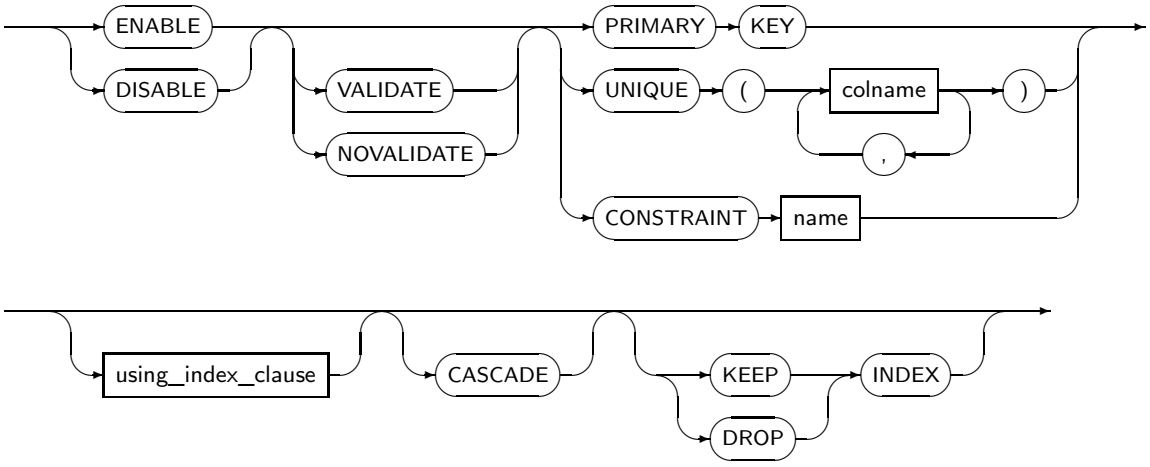
table_partition_desc



tablet_clause



atbl_con_alterstate_cl



- 특권
  - 사용자가 소유한 스키마에 테이블을 생성하기 위해서는 CREATE TABLE 시스템 특권이 있어야 한다.
  - 다른 사용자의 스키마에 테이블을 생성하기 위해서는 CREATE ANY TABLE 시스템 특권이 있어야 한다.
- 구성요소
  - create_table

구성요소	설명
GLOBAL TEMPORARY	임시 테이블(Temporary Table)을 생성하기 위해 지정한다. 임시 테이블의 스키마 자체는 일반 테이블과 다를 바 없이 모든 세션에 동일하게 보이지만, 테이블의 데이터는 각각의 세션이 따로 유지하게

구성요소	설명
	된다. 따라서 한 세션의 임시 테이블의 데이터는 다른 세션에서 접근할 수도, 볼 수도 없다. 오직 자신의 세션에서만 해당 테이블의 데이터를 볼 수 있다. 임시 테이블의 데이터는 영속 테이블 스페이스가 아닌 임시 테이블 스페이스에 저장되고, 세션의 종료와 동시에 임시 테이블의 데이터는 모두 제거된다. 트랜잭션을 커밋할 때 임시 테이블의 데이터 처리 방식을 지정할 수 있는 ON COMMIT 옵션을 지원한다. 임시 테이블은 sgmt_attr, colprop를 지정할 수 없고, FOREIGN KEY 제약조건을 지정할 수 없다.
schema	생성할 테이블이 속하게 될 스키마를 명시한다. 생략하면 현재 사용자의 스키마로 인식된다.
table_name	생성할 테이블의 이름을 명시한다.
relational_properties	테이블의 구성요소인 컬럼, 제약조건 등을 설정한다.
ON COMMIT DELETE ROWS	임시 테이블에서만 의미가 있는 옵션이다. ON COMMIT DELETE ROWS를 지정하면 트랜잭션을 커밋할 때 임시 테이블의 데이터를 깨끗하게 제거한다. 그러나 이렇게 저장된 임시 테이블의 정보는 옵션에 관계없이 세션의 종료와 동시에 모두 제거된다. 임시 테이블을 생성할 때 ON COMMIT 옵션을 지정하지 않았다면 디폴트로 ON COMMIT DELETE ROWS로 생성된다.
ON COMMIT PRESERVE ROWS	ON COMMIT PRESERVE ROWS를 지정하면 트랜잭션을 커밋할 때 데이터를 테이블에 저장하게 된다.
physical_properties	테이블의 물리적인 속성을 지정한다.
table_properties	파티셔닝, 대용량 객체형의 저장 방식 등 테이블의 속성을 지정한다.

– relational_properties

구성요소	설명
column_definition	테이블의 컬럼을 정의한다.
outofline_constraint	테이블의 제약조건을 지정한다. 자세한 내용은 “7.1.1. 제약조건”을 참고한다.

– coldef

구성요소	설명
datatype	컬럼의 데이터 타입을 명시한다.
DEFAULT expr	컬럼의 기본값을 지정한다.

구성요소	설명
ENCRYPT encryption_spec	컬럼을 암호화하고 암호화 옵션을 지정한다.
inline_constraint	테이블의 제약조건을 지정한다. 자세한 내용은 “7.1.1. 제약조건”을 참고한다.

– encryption_spec

구성요소	설명
USING encryption_algorithm	컬럼을 암호화할 알고리즘을 명시한다. encryption_algorithm은 문자열로 지정한다. Tibero에서 지원하는 암호화 알고리즘은 다음과 같다. – DES – 3DES168 – AES128 – AES192 (기본값) – AES256 – ARIA128 – ARIA192 – ARIA256 – SEED – GOST
SALT, NO SALT	보안을 강화하는 SALT 기능을 사용할지의 여부를 지정한다. SALT 기능은 컬럼 값이 같더라도 암호화 후의 값을 다르게 생성하여 보안을 강화한다. 단, SALT 기능은 같은 값을 매번 다르게 암호화하므로 해당 컬럼에 인덱스를 생성할 수 없다. 지정하지 않으면 기본값은 SALT이다.

– physical_properties

구성요소	설명
sgmt_attr	sgmt_attr 관련 문법은 “7.1.4. Sgmt_attr”을 참고한다.
table_compression	테이블의 압축 여부를 지정한다.
ORGANIZATION EXTERNAL	외부 데이터 파일을 지정할 때 external_table_clause 앞에 명시해야 한다.

구성요소	설명
external_table_clause	데이터베이스 외부 파일의 데이터에 대한 메타데이터(metadata)를 지정하여 외부 데이터를 읽기 전용 테이블로 사용할 수 있도록 한다. 외부 테이블을 생성할 때에는 컬럼의 정의 이외의 다른 속성은 지정할 수 없다.
ORGANIZATION INDEX	index organized table을 생성할 때 사용한다.
index_org_table_clause	index organized table 관련하여 정보 설정을 해줄 때 사용한다.

– table_compression

구성요소	설명
COMPRESS	DPI/DPL을 사용 중일 때만 테이블을 압축한다.
NOCOMPRESS	테이블을 압축하지 않는다.
COMPRESS FOR OLTP	DPI/DPL이 아닌 일반 DML일 때만 테이블을 압축한다.
COMPRESS FOR DIRECT_LOAD OPERATIONS	DPI/DPL을 사용 중일 때만 테이블을 압축한다.
COMPRESS FOR ALL OPERATIONS	모든 DML에 대해 테이블을 압축한다.
ROW STORE COMPRESS ADVANCED	DPI/DPL이 아닌 일반 DML일 때만 테이블을 압축한다.

– external_table_clause

구성요소	설명
DEFAULT DIRECTORY directory	외부 데이터의 파일을 지정할 때 디폴트 디렉터리를 설정한다. 디렉터리 생성에 관한 내용은 “7.27. CREATE DIRECTORY”를 참고한다.
ACCESS PARAMETERS	외부 데이터의 파일을 접근하기 위한 메타데이터를 지정한다. 자세한 내용은 "Tibero 유틸리티 안내서"를 참고한다.
tbloader_format_spec	"Tibero 유틸리티 안내서"의 tbLoader 부분을 참고한다.
LOCATION	외부 데이터의 파일의 위치를 지정한다.
directory	파일을 찾을 디렉터리를 의미하며, 지정하지 않으면 디폴트 디렉터리에서 파일을 찾게 된다.
location_specifier	문자열 리터럴로 디렉터리로부터 파일까지의 경로 및 파일명을 지정한다.

– index_org_table_clause

구성요소	설명
iot_attr	Index organized table을 생성할 때 사용하는 속성을 설정할 경우 사용한다.
INCLUDING	인덱스 블록에 primary key와 함께 저장할 수 있는 컬럼을 의미한다.
OVERFLOW	primary key를 제외하고 인덱스 블록 외에 별도로 데이터 저장을 원할 경우 사용한다.

– iot_attr

구성요소	설명
PCTTHRESHOLD	인덱스 블록에서 최대로 사용할 수 있는 사이즈를 의미한다. (단위: 퍼센트(%))

– table_properties

구성요소	설명
colprop	컬럼별로 대용량 객체형 데이터 타입이 저장되는 방식을 설정한다.
table_partitioning_clause	파티션 테이블을 생성하기 위한 문법이다. RANGE 파티셔닝, LIST 파티셔닝, HASH 파티셔닝 및 RANGE 복합 파티셔닝, LIST 복합 파티셔닝을 지원한다. 파티션 또는 서브 파티션마다 서로 다른 물리적인 속성을 지정할 수 있다. 단, LONG 또는 LONG RAW 데이터 타입을 가진 테이블은 파티셔닝할 수 없으며, 파티션 키 컬럼으로 대용량 객체형 데이터 타입, ROWID 데이터 타입의 컬럼은 사용할 수 없다. 파티션에 대한 자세한 내용은 "Tibero 관리자 안내서"를 참고한다.
parallel_clause	테이블 생성 과정을 병렬적으로 수행하며, 테이블에 수행하는 DML의 기본 DOP(Degree of Parallelism)으로 사용된다.
atbl_con_alterstate_cl	제약조건의 상태를 변경할 때 사용한다.
AS subquery	부질의를 통해 테이블의 컬럼 정의와 데이터를 옮긴다.

– colprop

구성요소	설명
colname	대용량 객체형 데이터 타입이 저장되는 방식을 설정할 컬럼의 이름을 명시한다.
lob_sgmt_param	대용량 객체형 데이터 타입의 저장 방법을 지정한다.
lob_name	대용량 객체형 데이터 타입의 이름을 설정한다.

– lob_param

구성요소	설명
TABLESPACE	대용량 객체형 데이터 타입이 저장되는 세그먼트의 테이블 스페이스를 지정할 수 있다.
ENABLE STORAGE IN ROW	약 4000bytes 이하의 작은 크기의 대용량 객체형은 LOB 세그먼트를 만들어 저장하지 않고 일반 컬럼처럼 로우의 데이터 안에 함께 저장한다. (기본값)
DISABLE STORAGE IN ROW	대용량 객체형 데이터의 크기에 상관없이 LOB 세그먼트에 저장한다.
ENCRYPT	대용량 객체형 데이터의 내용을 암호화하여 저장한다.
DECRYPT	대용량 객체형 데이터의 내용을 암호화하지 않고 저장한다.

– lob_compression_clause

구성요소	설명
NOCOMPRESS	대용량 객체형 LOB 데이터에 대한 압축을 수행하지 않는다. 별도로 지정하지 않으면 기본 설정이다.
COMPRESS	대용량 객체형 LOB 데이터에 대한 압축을 수행한다. 옵션을 명시하지 않은 경우 MEDIUM 설정으로 압축이 수행된다.
COMPRESS LOW	대용량 객체형 LOB 데이터에 대한 압축을 수행한다. 가장 낮은 압축률과 가장 빠른 속도로 압축을 한다.
COMPRESS MEDIUM	대용량 객체형 LOB 데이터에 대한 압축을 수행한다. 중간 정도의 압축률과 중간정도의 속도로 압축을 한다.
COMPRESS HIGH	대용량 객체형 LOB 데이터에 대한 압축을 수행한다. 가장 높은 압축률과 가장 느린 속도로 압축을 한다.

subpartition에는 compress 옵션을 지원하지 않는다.

– lob_deduplicate

구성요소	설명
KEEP_DUPLICATES	대용량 객체형 데이터 타입이 저장되는 세그먼트 중복된 데이터를 허용한다.
DEDUPLICATE	대용량 객체형 데이터 타입이 저장되는 세그먼트 중복된 데이터를 허용하지 않고 제거한다. 같은 데이터는 한개의 카피만 가지고 있는다.

subpartition에는 KEEP_DUPLICATES / DEDUPLICATE 옵션을 지원하지 않는다.

partitioned table의 경우 모든 파티션에 deduplicate 기능을 적용해야 한다.

– table_partitioning_clause

구성요소	설명
range_partitions	테이블을 파티션 키 컬럼 값의 범위에 따라 나눈다.
list_partitions	테이블을 파티션 키 컬럼 값의 목록에 따라 나눈다.
hash_partitions	테이블을 파티션 키 컬럼 값을 해싱한 결과에 따라 나눈다.
composite_range_partitions	RANGE 파티션으로 나누어진 파티션을 한번 더 파티셔닝하여 복합 RANGE 파티션 테이블을 생성한다.
composite_list_partitions	LIST 파티션으로 나누어진 파티션을 한번 더 파티셔닝하여 복합 LIST 파티션 테이블을 생성한다.
composite_hash_partitions	HASH 파티션으로 나누어진 파티션을 한번 더 파티셔닝하여 복합 HASH 파티션 테이블을 생성한다.

– range_partitions

구성요소	설명
column	파티션의 기준이 되는 파티션 키 컬럼을 설정한다. 로우가 어떤 파티션에 포함될 지는 해당 로우가 가지고 있는 이 컬럼의 값에 따라 결정된다.
interval_value	Range 파티션의 Interval 값을 지정한다. Interval 값이 지정된 Range 파티션으로의 DML을 수행하는 경우 파티션 키 값에 대응되는 파티션이 존재하지 않을 때, 새로운 파티션을 생성하여 DML 수행을 계속한다. 이때 새로 생성되는 파티션의 경계값을 계산하기 위해 필요한 값이 Interval_value이다. Range 파티션의 마지막 파티션의 경계값을 시작으로 Interval 값의 배수를 더한 값을 새로 만들어질 파티션의 경계값으로 갖게 된다.
range_values_clause	파티션의 상위 경계 값(Upper bound)을 지정한다. 지정한 값은 해당 파티션에 포함되지 않으며, 경계 값의 개수는 파티셔닝 컬럼의 개수와 동일해야 한다. 최댓값을 표현하고자 할 때는 MAXVALUE 예약어를 사용한다. MAXVALUE를 사용하면, NULL 값을 포함하여 이전 파티션보다 해당 컬럼의 값이 큰 모든 로우가 해당 파티션으로 들어간다.

– list_partitions

구성요소	설명
column	파티션의 기준이 되는 파티션 키 컬럼을 설정한다. 로우가 어떤 파티션에 포함될 지는 해당 로우가 가지고 있는 이 컬럼의 값에 따라 결정된다. LIST 파티션에서는 컬럼을 한 개만 지정할 수 있다는 점이 다른 파티션과 다르다.

구성요소	설명
list_values_clause	파티션에 포함될 로우의 파티션 컬럼 값을 지정한다. 즉 파티셔닝 컬럼의 값이 지정한 값과 같은 로우가 해당 파티션으로 포함된다. NULL 예약어를 사용하여 NULL 값을 가지는 로우를 지정할 수 있다. RANGE 파티션의 MAXVALUE와 유사하게, LIST 파티션에서는 DEFAULT 예약어를 사용하여 다른 파티션에 들어가지 않는 모든 로우를 포함하는 파티션을 지정할 수 있다. 단, DEFAULT 값은 다른 값과 같이 쓰일 수 없으며, 맨 마지막 파티션에만 사용할 수 있다.

– hash_partitions

구성요소	설명
column	파티션의 기준이 되는 파티션 키 컬럼을 설정한다. 로우가 어떤 파티션에 포함될 지는 해당 로우가 가지고 있는 이 컬럼의 값에 따라 결정된다.
hash_partition_desc	HASH 파티션의 세부적인 속성을 지정한다.
hash_partitions_by_quantity	HASH 파티션을 파티션 개수와 테이블 스페이스만 지정함으로써 간단하게 생성할 수 있도록 해 주는 문법이다.

– composite_range_partitions

구성요소	설명
subpartition_by_clause	RANGE 파티션을 다시 파티셔닝을 할 방법을 설정한다. 기본 파티셔닝과 마찬가지로 RANGE, LIST, HASH 중에 선택할 수 있으며, 서브 파티셔닝 컬럼을 지정한다.
range_partition_desc	복합 RANGE 파티션의 세부적인 속성을 지정한다.

– composite_list_partitions

구성요소	설명
subpartition_by_clause	LIST 파티션을 다시 파티셔닝을 할 방법을 설정한다. 기본 파티셔닝과 마찬가지로 RANGE, LIST, HASH 중에 선택할 수 있으며, 서브 파티셔닝 컬럼을 지정한다.
list_partition_desc	복합 LIST 파티션의 세부적인 속성을 지정한다.

– composite_hash_partitions

구성요소	설명
subpartition_by_clause	HASH 파티션을 다시 파티셔닝을 할 방법을 설정한다. 기본 파티셔닝과 마찬가지로 RANGE, LIST, HASH 중에 선택할 수 있으며, 서브 파티셔닝 컬럼을 지정한다.
hash_partition_desc	HASH 파티션의 세부적인 속성을 지정한다.
hash_partitions_by_quantity	HASH 파티션을 파티션 개수와 테이블 스페이스만 지정함으로써 간단하게 생성할 수 있도록 해 주는 문법이다.

– subpartition_by_clause

구성요소	설명
RANGE	서브 파티션의 파티셔닝 방법을 RANGE 파티션으로 설정한다.
LIST	서브 파티션의 파티셔닝 방법을 LIST 파티션으로 설정한다.
HASH	서브 파티션의 파티셔닝 방법을 HASH 파티션으로 설정한다.
column	파티션의 기준이 되는 파티션 키 컬럼을 설정한다.
subpartition_template	서브 파티션의 속성을 모든 파티션에 적용시킬때 사용한다.

– subpartition_template

구성요소	설명
SUBPARTITION TEMPLATE	서브 파티션의 속성을 모든 파티션에 적용시킬때 사용하는 구문이다.
list_subpartition_desc	서브 파티션의 세부적인 속성을 지정한다. 서브 파티션이라는 점과 예약어가 SUBPARTITION, SUBPARTITIONS로 바뀐 것 외에 다른 문법은 기본 파티셔닝과 동일하다.
hash_subpartitions_by_quanti ty	HASH 서브 파티션을 파티션의 개수와 테이블 스페이스만 지정함으로써 간단하게 생성할 수 있도록 해 주는 문법이다.

– range_partition_desc

구성요소	설명
PARTITION partition_name	파티션의 이름을 지정한다.
range_values_clause	파티션의 상위 경계 값(Upper bound)을 지정한다. 지정한 값은 해당 파티션에 포함되지 않으며, 경계 값의 개수는 파티셔닝 컬럼의 개수와 동일해야 한다. 최댓값을 표현하고자 할 때는 MAXVALUE 예약어를 사용한다. MAXVALUE를 사용하면, NULL 값을 포함하여 이전 파티션보다 해당 컬럼의 값이 큰 모든 로우가 해당 파티션으로 들어간다.

구성요소	설명
table_partition_desc	파티션의 물리적인 속성을 지정한다. 문법은 테이블의 속성을 지정할 때와 유사하며, 파티션의 속성을 지정한다는 것만 다르다.
range_subpartition_desc	서브 파티션의 세부적인 속성을 지정한다. 서브 파티션이라는 점과 예약어가 SUBPARTITION, SUBPARTITIONS로 바뀐 것 외에 다른 문법은 기본 파티셔닝과 동일하다.
list_subpartition_desc	서브 파티션의 세부적인 속성을 지정한다. 서브 파티션이라는 점과 예약어가 SUBPARTITION, SUBPARTITIONS로 바뀐 것 외에 다른 문법은 기본 파티셔닝과 동일하다.
hash_subparrition_desc	서브 파티션의 세부적인 속성을 지정한다. 서브 파티션이라는 점과 예약어가 SUBPARTITION, SUBPARTITIONS로 바뀐 것 외에 다른 문법은 기본 파티셔닝과 동일하다.
hash_subparrition_by_quantity	HASH 서브 파티션을 파티션의 개수와 테이블 스페이스만 지정함으로써 간단하게 생성할 수 있도록 해 주는 문법이다.

– list_partition_desc

구성요소	설명
PARTITION partition_name	파티션의 이름을 지정한다.
list_values_clause	파티션에 포함될 로우의 파티션 컬럼 값을 지정한다. 즉 파티셔닝 컬럼의 값이 지정한 값과 같은 로우가 해당 파티션으로 포함된다. NULL 예약어를 사용하여 NULL 값을 가지는 로우를 지정할 수 있다. RANGE 파티션의 MAXVALUE와 유사하게, LIST 파티션에서는 DEFAULT 예약어를 사용하여 다른 파티션에 들어가지 않는 모든 로우를 포함하는 파티션을 지정할 수 있다. 단, DEFAULT 값은 다른 값과 같이 쓰일 수 없으며, 맨 마지막 파티션에만 사용할 수 있다.
table_partition_desc	파티션의 물리적인 속성을 지정한다. 문법은 테이블의 속성을 지정할 때와 유사하며, 파티션의 속성을 지정한다는 것만 다르다.
range_subpartition_desc	서브 파티션의 세부적인 속성을 지정한다. 서브 파티션이라는 점과 예약어가 SUBPARTITION, SUBPARTITIONS로 바뀐 것 외에 다른 문법은 기본 파티셔닝과 동일하다. 자세한 문법은 range_partition_desc를 참고한다.
list_subpartition_desc	서브 파티션의 세부적인 속성을 지정한다.

구성요소	설명
	서브 파티션이라는 점과 예약어가 SUBPARTITION, SUBPARTITIONS로 바뀐 것 외에 다른 문법은 기본 파티셔닝과 동일하다. 자세한 문법은 list_partition_desc를 참고한다.
hash_subparition_desc	Tibero InfiniData에서는 지원하지 않는다.
hash_subparrition_desc	서브 파티션의 세부적인 속성을 지정한다. 서브 파티션이라는 점과 예약어가 SUBPARTITION, SUBPARTITIONS로 바뀐 것 외에 다른 문법은 기본 파티셔닝과 동일하다. 자세한 문법은 hash_partition_desc를 참고한다.
hash_subparition_by_quantity	Tibero에서는 지원하지 않는다.
hash_subparrition_by_quantity	HASH 서브 파티션을 파티션의 개수와 테이블 스페이스만 지정함으로써 간단하게 생성할 수 있도록 해 주는 문법이다.

– hash_partition_desc

구성요소	설명
PARTITION partition_name	파티션의 이름을 지정한다. 생략하면 시스템이 자동으로 생성한다.
table_partition_desc	파티션의 물리적인 속성을 지정한다. 문법은 테이블의 속성을 지정할 때와 유사하며, 파티션의 속성을 지정한다는 것만 다르다.

– hash_partition_by_quantity

구성요소	설명
PARTITIONS unsigned_integer	생성할 파티션의 개수를 지정한다.
STORE IN tablespace	파티션이 위치할 테이블 스페이스를 지정한다. 테이블 스페이스를 지정한 개수가 파티션의 개수와 일치할 필요는 없으며, 개수가 서로 다를 경우 라운드 로빈(Round-robin) 방식으로 테이블 스페이스가 차례대로 지정된다.

– range_subpartition_desc

구성요소	설명
subpartition_name	서브 파티션의 이름을 명시한다. 생략하면 시스템이 자동으로 생성한다.
range_values_clause	RANGE 파티션의 경계 값을 지정한다.
table_partition_desc	파티션의 물리적인 속성을 지정한다. 문법은 테이블의 속성을 지정할 때와 유사하며, 파티션의 속성을 지정한다는 것만 다르다.

– list_subpartition_desc

구성요소	설명
subpartition_name	서브 파티션의 이름을 명시한다. 생략하면 시스템이 자동으로 생성한다.
list_values_clause	LIST 파티션의 분류 값을 지정한다.
table_partition_desc	파티션의 물리적인 속성을 지정한다. 문법은 테이블의 속성을 지정할 때와 유사하며, 파티션의 속성을 지정한다는 것만 다르다.

– hash_subpartition_desc

구성요소	설명
subpartition_name	서브 파티션의 이름을 명시한다. 생략하면 시스템이 자동으로 생성한다.
table_partition_desc	파티션의 물리적인 속성을 지정한다. 문법은 테이블의 속성을 지정할 때와 유사하며, 파티션의 속성을 지정한다는 것만 다르다.

– hash_subpartition_by_quantity

구성요소	설명
SUBPARTITIONS unsigned_integer	생성할 서브 파티션의 개수를 지정한다.
STORE IN tablespace	서브 파티션이 위치할 테이블 스페이스를 지정한다. 테이블 스페이스를 지정한 개수가 서브 파티션의 개수와 일치할 필요는 없으며, 개수가 서로 다를 경우 라운드 로빈 방식으로 테이블 스페이스가 차례대로 지정된다.

– range_value_clause

구성요소	설명
literal	파티션의 상위 경계 값을 지정한다. 지정한 값은 해당 파티션에 포함되지 않으며, 경계 값의 개수는 파티셔닝 컬럼의 개수와 동일해야 한다.
MAXVALUE	최댓값을 표현하고자 할 때는 MAXVALUE를 사용한다. MAXVALUE를 사용하면 NULL 값을 포함하여 이전 파티션보다 해당 컬럼의 값이 큰 모든 로우가 해당 파티션으로 들어간다.

– list_value_clause

구성요소	설명
literal	파티션에 포함될 로우의 파티션 컬럼 값을 지정한다. 즉 파티셔닝 컬럼의 값이 지정한 값과 같은 로우가 해당 파티션으로 들어간다.
NULL	NULL 예약어를 사용하여 NULL 값을 가지는 로우를 지정할 수 있다.

구성요소	설명
DEFAULT	DEFAULT 예약어를 사용하여 다른 파티션에 들어가지 않는 모든 로우를 포함하는 파티션을 지정할 수 있다. 단, DEFAULT 값은 다른 값과 같이 쓰일 수 없으며, 맨 마지막 파티션에만 사용할 수 있다.

– table_partition_desc

구성요소	설명
sgmt_attr	sgmt_attr 관련 문법은 “7.1.4. Sgmt_attr”을 참고한다.
table_compression	테이블의 압축 여부를 지정한다.
colprop	컬럼별로 대용량 객체형 데이터 타입이 저장되는 방식을 설정한다.

– atbl_con_alterstate_cl

구성요소	설명
ENABLE/DISABLE VALIDATE/NOVALIDATE	테이블을 생성하면서 이 부분을 지정하지 않으면, 디폴트는 모든 제약조건을 ENABLE VALIDATE 상태로 설정한다. 자세한 내용은 “7.1.1. 제약조건”을 참고한다.
PRIMARY KEY	테이블에 기본 키 제약조건이 이미 존재하는 경우 이를 활성화 또는 비활성화할 수 있다.
UNIQUE column_name	뒤에 명시하는 column_name에 이미 유일 키 제약조건이 존재하는 경우 이를 활성화 또는 비활성화할 수 있다.
CONSTRAINT name	해당 제약조건의 이름을 명시하여 이미 존재하는 제약조건을 활성화 또는 비활성화할 수 있다.
CASCADE	다른 테이블이나 같은 테이블의 다른 컬럼으로부터 FOREIGN KEY 제약조건으로 참조되는 기본 키나 유일 키 제약조건을 비활성화할 때 관련된 FOREIGN KEY까지 함께 비활성화하기 위해 사용한다. FOREIGN KEY가 존재하는 제약조건을 비활성화하는 경우에는 반드시 포함되어야 한다.
KEEP INDEX	제약조건을 비활성화하면서 제약조건에서 사용한 인덱스를 제거하지 않고 그냥 유지하고자 할 때 명시한다. 기본값이므로 생략할 수 있다.
DROP INDEX	기본 키, 유일 키, FOREIGN KEY 같은 인덱스를 사용하는 제약조건을 제거하려 할 때 인덱스도 함께 제거하고자 할 때 명시한다. 생략하면 KEEP INDEX로 인식된다.
using_index_clause	“7.1.1. 제약조건”을 참고한다.

- 예제

  – create_table

다음은 제약조건과 기본값을 포함한 일반 테이블을 생성하는 예이다.

```
CREATE TABLE leagues ( league_id NUMBER PRIMARY KEY, name VARCHAR(20) UNIQUE );

CREATE TABLE teams (
    team_id     NUMBER       CONSTRAINT team_prim   PRIMARY KEY,
    name        VARCHAR(20)  CONSTRAINT team_uniq   UNIQUE,
    league      NUMBER       CONSTRAINT team_ref  REFERENCES leagues (league_id),

    tot_salary  NUMBER(10,3) CONSTRAINT team_sal_nn NOT NULL,
    tot_players NUMBER       CONSTRAINT team_player_limit
    CHECK (tot_players <= 40)
);

CREATE TABLE players ( player_id NUMBER,
    name      VARCHAR(20),
    nickname  VARCHAR(20),
    age       NUMBER,
    team      VARCHAR(20),
    join_date DATE          DEFAULT SYSDATE,
    PRIMARY KEY (player_id) USING INDEX
    (
        CREATE UNIQUE INDEX players_idx1 ON players(player_id)
    ),
    CHECK (age >= 15),
    CONSTRAINT players_uniq         UNIQUE (name, nickname),
    CONSTRAINT players_team         FOREIGN KEY(team) REFERENCES teams(name),
    CONSTRAINT players_join_date_nn CHECK (join_date IS NOT NULL)
);
```

다음은 **TEMPORARY**를 명시하여 임시 테이블을 생성하는 예이다.

```
CREATE GLOBAL TEMPORARY TABLE todays_sales (
    product_id   NUMBER NOT NULL,
    product_name VARCHAR(20),
    price        NUMBER )
ON COMMIT PRESERVE ROWS;
```

- physical_properties

다음은 **sgmt_attr**을 사용해 테이블을 생성하는 예이다.

```
CREATE TABLE sgmt_attr_tbl_example (
    col1 NUMBER,
    col2 NUMBER )
TABLESPACE user1_ts
PCTFREE 30
INITRANS 3;
```

다음은 **ORGANIZATION EXTERNAL**을 명시해 외부 테이블을 생성하는 예이다.

```
CREATE TABLE ext_tbl (
    col1 VARCHAR2(30),
    col2 NUMBER,
    col3 VARCHAR2(30),
    col4 NUMBER,
    col5 VARCHAR2(30),
    col6 DATE )
    ORGANIZATION EXTERNAL
    (
        DEFAULT DIRECTORY
        FILE_DIR ACCESS PARAMETERS
        (
            LOAD DATA INTO TABLE ext_tbl FIELDS
            TERMINATED BY ','
            ESCAPED BY '\\'
            LINES TERMINATED BY '\n'
            IGNORE 1 LINES (col1, col2, col3, col4, col5, col6)
        )
        LOCATION('FileInput.txt')
    );
```

– table_properties

다음은 **as_subquery**를 포함한 테이블을 생성하는 예이다.

```
CREATE TABLE young_players ( player_id, name, alais, age, team )
AS
SELECT player_id, name, nickname, age, team
FROM players
WHERE age <= 25;
```

다음은 **colprop**를 포함한 테이블을 생성하는 예이다.

```
CREATE TABLE col_prop ( col1 CLOB, col2 BLOB )
LOB (col1) STORE AS lob_col_prop_col1
(
    TABLESPACE lob_ts
    ENABLE STORAGE IN ROW
)
LOB (col2) STORE AS lob_col_prop_col2
(
    TABLESPACE lob_ts
    DISABLE STORAGE IN ROW
);
```

– table_partitioning_clause

다음은 **range_partitions**를 명시해 RANGE 파티션을 생성하는 예이다.

```
CREATE TABLE years_sales (
    product_id   NUMBER,
    product_name VARCHAR(20),
    price        NUMBER,
    sold_date    DATE
)
PARTITION BY RANGE (sold_date)
(
    PARTITION jan_sales VALUES LESS THAN (TO_DATE('31-01-2006', 'DD-MM-YYYY')),
    PARTITION feb_sales VALUES LESS THAN (TO_DATE('28-02-2006', 'DD-MM-YYYY')),

    PARTITION mar_sales VALUES LESS THAN (TO_DATE('31-03-2006', 'DD-MM-YYYY')),

    PARTITION apr_sales VALUES LESS THAN (TO_DATE('30-04-2006', 'DD-MM-YYYY')),

    PARTITION may_sales VALUES LESS THAN (TO_DATE('31-05-2006', 'DD-MM-YYYY')),
    PARTITION jun_sales VALUES LESS THAN (TO_DATE('30-06-2006', 'DD-MM-YYYY')),
    PARTITION jul_sales VALUES LESS THAN (TO_DATE('31-07-2006', 'DD-MM-YYYY')),
    PARTITION aug_sales VALUES LESS THAN (TO_DATE('31-08-2006', 'DD-MM-YYYY')),
    PARTITION sep_sales VALUES LESS THAN (TO_DATE('30-09-2006', 'DD-MM-YYYY')),

    PARTITION oct_sales VALUES LESS THAN (TO_DATE('31-10-2006', 'DD-MM-YYYY')),

    PARTITION nov_sales VALUES LESS THAN (TO_DATE('30-11-2006', 'DD-MM-YYYY')),
    PARTITION dec_sales VALUES LESS THAN (TO_DATE('31-12-2006', 'DD-MM-YYYY'))
);
```

다음은 **list_partitions**를 명시해 LIST 파티션을 생성하는 예이다.

```
CREATE TABLE country_sales (
    product_id   NUMBER,
    product_name VARCHAR(20),
    price        NUMBER,
    country      VARCHAR(20),
    sold_date    DATE
)
PARTITION BY LIST (country)
(
    PARTITION asia    VALUES ('KOREA', 'CHINA', 'JAPAN'),
    PARTITION america VALUES ('U.S.A.'),
    PARTITION europe  VALUES ('ENGLAND', 'SPAIN', 'GERMANY', 'ITALY'),
    PARTITION etc     VALUES (DEFAULT)
);
```

다음은 **hash_partitions**를 명시해 HASH 파티션을 생성하는 예이다.

```
CREATE TABLE product_sales (
    product_id   NUMBER,
    product_name VARCHAR(20),
    price        NUMBER,
    sold_date    DATE
)
PARTITION BY HASH (product_id)
PARTITIONS 4
STORE IN (ts1, ts2, ts3);
```

다음은 **conposit_range_partitions**를 명시해 복합 RANGE 파티션을 생성하는 예이다.

```
CREATE TABLE years_sales (
    product_id   NUMBER,
    product_name VARCHAR(20),
    price        NUMBER,
    sold_date    DATE
)
PARTITION BY RANGE (sold_date)
SUBPARTITION BY HASH (product_id)
(
    PARTITION jan_sales VALUES LESS THAN (TO_DATE('31-01-2006', 'DD-MM-YYYY')),

    PARTITION feb_sales VALUES LESS THAN (TO_DATE('28-02-2006', 'DD-MM-YYYY')),

    PARTITION mar_sales VALUES LESS THAN (TO_DATE('31-03-2006', 'DD-MM-YYYY')),
    PARTITION apr_sales VALUES LESS THAN (TO_DATE('30-04-2006', 'DD-MM-YYYY')),

    PARTITION may_sales VALUES LESS THAN (TO_DATE('31-05-2006', 'DD-MM-YYYY')),

    PARTITION jun_sales VALUES LESS THAN (TO_DATE('30-06-2006', 'DD-MM-YYYY')),
    PARTITION jul_sales VALUES LESS THAN (TO_DATE('31-07-2006', 'DD-MM-YYYY')),
    PARTITION aug_sales VALUES LESS THAN (TO_DATE('31-08-2006', 'DD-MM-YYYY')),
    PARTITION sep_sales VALUES LESS THAN (TO_DATE('30-09-2006', 'DD-MM-YYYY')),
    PARTITION oct_sales VALUES LESS THAN (TO_DATE('31-10-2006', 'DD-MM-YYYY'))
        SUBPARTITIONS 2,
    PARTITION nov_sales VALUES LESS THAN (TO_DATE('30-11-2006', 'DD-MM-YYYY'))
        SUBPARTITIONS 4,
    PARTITION dec_sales VALUES LESS THAN (TO_DATE('31-12-2006', 'DD-MM-YYYY'))
    (
        SUBPARTITION dec_1,
        SUBPARTITION dec_2,
        SUBPARTITION dec_3,
        SUBPARTITION dec_4
```

```
    )
);
```

- atbl_con_alterstate_cl

  다음은 **atbl_con_alterstate_cl**를 사용해 테이블의 제약조건을 활성화하거나 비활성화하는 등의 설정을 하는 예이다.

```
CREATE TABLE enable_disable (
    col1 NUMBER      PRIMARY KEY,
    col2 VARCHAR(10) UNIQUE,
    col3 NUMBER,
    col4 NUMBER,
    CONSTRAINT enable_disable_con001 CHECK (col3 > col4) ) DISABLE NOVALIDATE
    PRIMARY KEY ENABLE VALIDATE
    UNIQUE(col2) DISABLE CONSTRAINT enable_disable_con001;
```

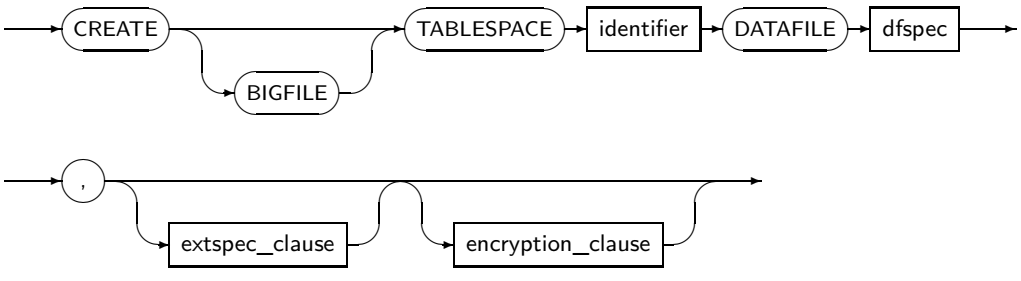
7.42. CREATE TABLESPACE

테이블 스페이스를 생성한다. 테이블 스페이스는 테이블, 인덱스 등의 스키마 객체를 저장하는 물리적, 논리적 공간이다.

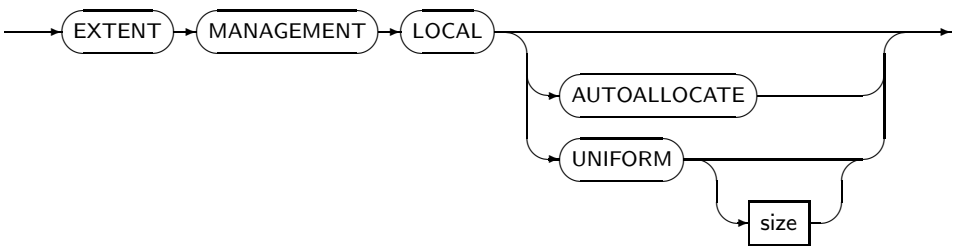
CREATE TABLESPACE의 세부 내용은 다음과 같다.

- 문법

create_tablespace

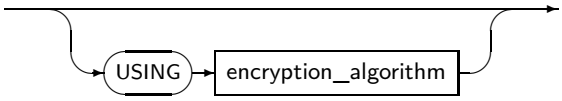


extspec_clause



encryption_clause

encryption_spec



- 특권

  SYSDBA 특권이 있어야 CREATE TABLESPACE 문을 실행할 수 있다.

- 구성요소

  – create_tablespace

구성요소	설명
BIGFILE	단일 대용량 데이터 파일인 BIGFILE 테이블 스페이스를 생성하도록 한다.
identifier	생성할 테이블 스페이스의 이름을 명시한다.
dfspec	파일의 이름, 파일의 크기 등과 관련된 설정을 할 수 있다. 자세한 내용은 "7.25. CREATE DATABASE"를 참고한다. 일반 TABLESPACE의 경우 최대 32GB까지, BIGFILE TABLESPACE의 경우 최대 32TB 크기까지 데이터 파일 크기를 지정할 수 있다. 또한, 일반 TABLESPACE는 여러 개의 데이터 파일을 지정할 수 있지만, BIGFILE TABLESPACE는 하나의 데이터 파일만 지정할 수 있다.
extspec_clause	테이블 스페이스의 익스텐트가 어떻게 관리될 것인지를 명시한다.

  – extspec_clause

구성요소	설명
EXTENT_MANAGEMENT_LOCAL	EXTENT_MANAGEMENT_LOCAL의 뒷 부분에 테이블 스페이스의 익스텐트가 어떻게 관리될 것인지를 명시한다.
AUTOALLOCATE	익스텐트의 크기를 시스템이 자동으로 설정한다.
UNIFORM	익스텐트의 크기를 사용자가 설정한다. 사용자가 설정한 크기대로 익스텐트가 항상 일정한 크기로 생성된다.
size	익스텐트의 블록 크기를 명시한다. $TB_SID.tip 파일에 설정된 DB_BLOCK_SIZE 파라미터의 값이 블록 크기의 배수가 아니면, 블록 크기 단위로 내림 계산된다. (기본값: 16개 블록 크기, 16 ~ 4194304까지의 블록 개수만큼의 크기 명시)

  – encryption_clause

  테이블 스페이스를 암호화하고 싶을 때 사용한다.

구성요소	설명
encryption_spec	암호화 알고리즘을 선택한다. USING 뒤에 암호화 알고리즘을 작은따옴표와 함께 적는 방식으로 지정한다. 지원하는 암호화 알고리즘은 다음과 같다. – DES – 3DES168 – AES128 – AES192(기본값) – AES256 – ARIA128 – ARIA192 – ARIA256 – SEED – GOST 자세한 사용법은 "Tibero 관리자 안내서"를 참고한다.

- 예제

  다음은 **CREATE TABLESPACE**를 사용해 테이블 스페이스를 생성하는 예이다.

```
CREATE TABLESPACE ts1 DATAFILE 'ts1.df', 'ts2.df'
  EXTENT MANAGEMENT LOCAL UNIFORM SIZE 256K;
```

7.43. CREATE TRIGGER

데이터베이스 트리거를 생성한다. 트리거는 테이블, 스키마, 데이터베이스와 연결된 tbPSM 블록이다.

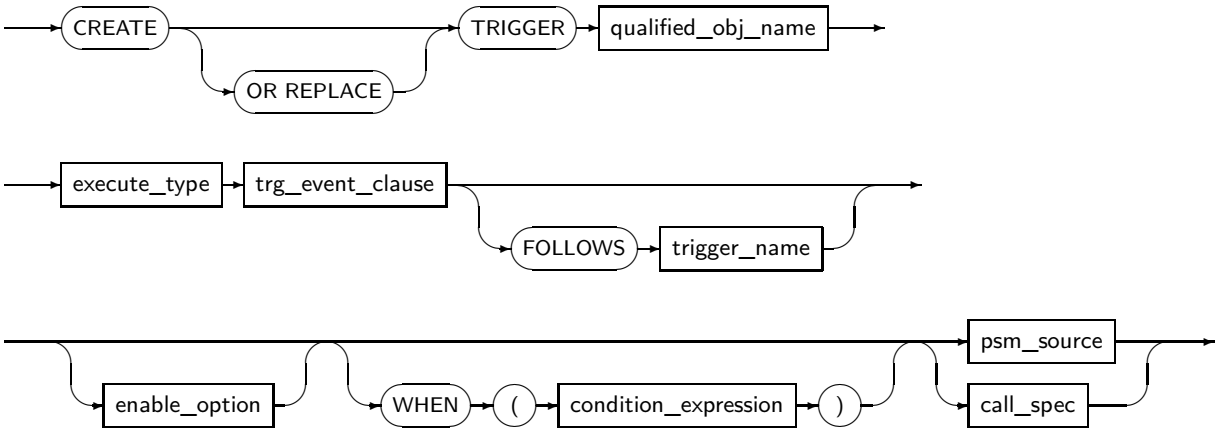
트리거는 활성화 상태로 생성되며, 생성 이후에 ALTER TRIGGER 문 또는 ALTER TABLE 문을 사용하여 활성화 또는 비활성화 상태를 변경할 수 있다. 트리거가 활성화된 상태에서는 주어진 조건이 만족되는 순간, 트리거의 내용이 수행된다.

트리거에 컴파일 에러가 발생하면 트리거는 생성되지만, 실행은 되지 않는다. 결과적으로 트리거가 비활성화 상태로 바뀌거나, 컴파일 에러가 없는 버전으로 교체되거나, 트리거가 삭제되기 전까지는 트리거 조건을 만족하는 모든 DML 문장이 블록되는 상태가 된다. DDL 또는 DB 이벤트 트리거를 사용하기 위해서는 파라미터 _DDL_TRIGGER_ENABLE을 Y로 켜야 한다.

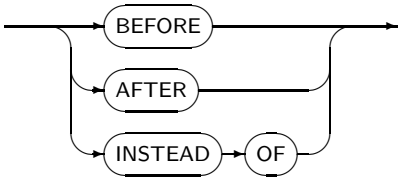
CREATE TRIGGER의 세부 내용은 다음과 같다.

- 문법

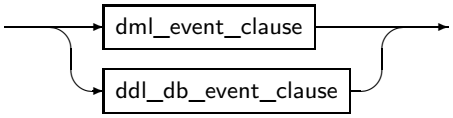
create_trigger



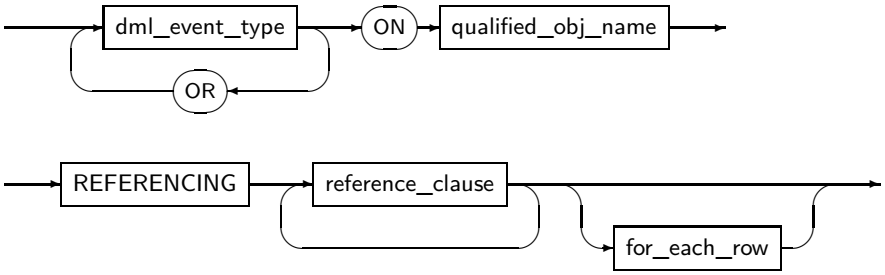
execute_type



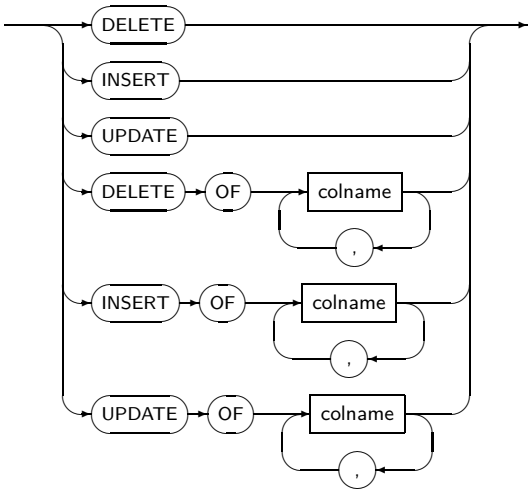
trg_event_clause



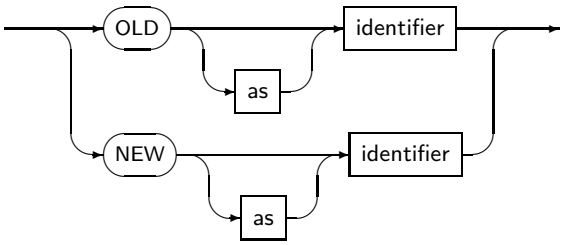
dml_event_clause



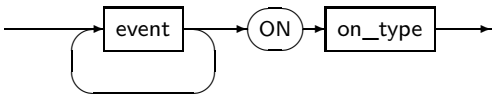
dml_event_type



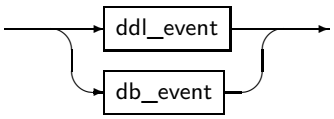
reference_clause



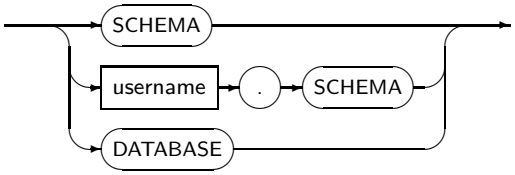
ddl_db_event_clause



ddl_or_db_event



on_type



- 특권
  - 사용자가 소유한 스키마에 트리거를 생성하려면, CREATE TRIGGER 시스템 특권이 있어야 한다.

- 다른 사용자가 소유한 스키마에 트리거를 생성하려면, CREATE ANY TRIGGER 시스템 특권이 있어야 한다.
- 데이터베이스에 연결되는 트리거를 생성하기 위해서는 ADMINISTER DATABASE TRIGGER 시스템 특권이 있어야 한다.
- 트리거가 SQL 문장을 수행하거나 프러시저 또는 함수를 호출할 때 트리거의 소유자는 이러한 작업을 수행할 수 있는 특권이 있어야 한다.

● 구성요소

- create_trigger

구성요소	설명
OR REPLACE	이미 존재하는 트리거의 내용을 변경할 때 OR REPLACE를 명시한다.
execute_type	트리거가 작동이 되는 실행 종류를 명시한다 .
BEFORE	BEFORE를 명시하면 트리거를 동작시킬 수 있는 이벤트를 수행하기 전에 트리거를 먼저 수행한다. 로우 트리거의 경우 해당 로우가 변경되기 전에 트리거가 먼저 수행된다.
AFTER	AFTER는 BEFORE와 반대로 트리거를 동작시킬 수 있는 이벤트를 먼저 수행하고 트리거를 나중에 수행한다. 로우 트리거의 경우 해당 로우를 변경한 후 트리거를 수행하게 된다.
INSTEAD OF	INSTEAD OF는 트리거를 수행하는 대신 트리거를 수행한다. 뷰에 대한 DML 이벤트일 경우에만 가능하며, 테이블에 대한 경우 또는 DDL이나 DB 이벤트일 경우에는 사용할 수 없다.
dml_event_clause	트리거를 DML 이벤트 트리거로 만든다. DML 이벤트 트리거는 지정한 DML 문장이 실행될 때마다 트리거가 수행된다.
ddl_db_event_clause	트리거를 DDL 또는 DB 이벤트 트리거로 만든다. DML 이벤트 트리거는 지정한 DDL 문장 또는 DB 이벤트가 실행될 때마다 트리거가 수행된다.
FOLLOWS trigger_name	현재 트리거를 반드시 지정한 트리거(trigger_name) 이후에 실행하도록 지정한다. 지정하지 않을 경우 트리거는 생성한 순서대로 실행된다.
enable_option	트리거 활성화 여부를 지정한다. ENABLE 또는 DISABLE을 사용할 수 있으며, 기본값은 활성화(ENABLE)이다.
WHEN condition_expression	트리거가 동작할 조건을 명시한다. 이 조건이 만족되기 전까지는 데이터베이스는 트리거를 동작시키지 않는다. 만일 DML 이벤트 트리거에 이 조건을 명시한 경우에는, 반드시 FOR EACH ROW도 함께 명시하여야 한다.
psm_source	트리거가 동작했을 때 수행될 프러시저를 명시한다. tbPSM 블록을 직접 명시하는 방법이다.
call_spec	트리거가 동작했을 때 수행될 프러시저를 명시한다. CALL 문을 사용해 저장된 프러시저를 호출한다. 자세한 내용은 “8.4. CALL”을 참고한다.

– dml_event_clause

구성요소	설명
dml_event_type	dml_event_type에 설정된 DML 문장이 실행될 때마다 트리거가 수행되도록 설정한다.
ON schema	생성된 트리거가 연결될 스키마를 명시한다. 생략하면 현재 사용자의 스키마로 인식된다.
object_name	트리거를 연결할 객체의 이름을 명시한다.
REFERENCING reference_clause	트리거에서 현재 테이블의 로우의 값에 접근하는 방법을 지정한다. 트리거에서 현재 테이블의 로우의 값에 접근할 때 기존 값은 OLD, 새 값은 NEW로 접근하게 된다. 예를 들어 테이블에 A라는 컬럼이 있다면 새 값을 지정하려고 할 때 트리거 내의 PSM 코드에서 다음과 같이 명시할 수 있다. 사용자의 필요에 따라 NEW, OLD가 아닌 다른 이름으로 현재 로우의 값에 접근하고 싶다면 이 절을 이용해 바꾸어 줄 수 있다. `:OLD.A := 1;`
FOR EACH ROW	FOR EACH ROW를 명시하면 트리거는 로우 트리거가 된다. 트리거 이벤트에 영향을 받는 각각의 로우에 로우 트리거를 수행한다. 이 절을 생략하면, 트리거는 문 단위의 트리거가 된다. 트리거 이벤트를 발생시키는 SQL 문장이 수행되면, 트리거는 한 번만 수행된다.

– dml_event_type

구성요소	설명
DELETE	테이블에서 DELETE 문이 수행되어 로우가 삭제될 때마다 트리거가 수행된다.
INSERT	테이블에 INSERT 문이 수행되어 로우가 추가될 때마다 트리거가 수행된다.
UPDATE	테이블에 저장된 값이 변경될 때마다 트리거가 수행된다.
UPDATE OF	OF에 명시된 컬럼의 값이 변경될 때마다 트리거를 수행한다.

– reference_clause

구성요소	설명
OLD	테이블의 로우의 기존 값에 접근할 때 사용된다.
NEW	테이블의 로우의 새 값에 접근할 때 사용된다.
AS identifier	OLD와 NEW 대신 다른 이름으로 지칭하고 싶을 때 사용한다.

– ddl_db_event_clause

구성요소	설명
event	DDL 또는 DB 이벤트를 명시한다.
on_type	DDL 또는 DB 이벤트가 일어난 대상을 명시한다. Schema 또는 Database가 있다.

– ddl_or_db_event

구성요소	설명
ALTER	DDL ALTER가 일어날 때 트리거가 수행한다.
ANALYZE	DDL ANALYZE가 일어날 때 트리거가 수행한다.
ASSOCIATE STATISTICS	DDL ASSOCIATE STATISTICS가 일어날 때 트리거가 수행한다.
AUDIT	DDL AUDIT이 일어날 때 트리거가 수행한다.
COMMENT	DDL COMMENT가 일어날 때 트리거가 수행한다.
CREATE	DDL CREATE가 일어날 때 트리거가 수행한다.
DISASSOCIATE STATISTICS	DDL DISASSOCIATE STATISTICS가 일어날 때 트리거가 수행한다.
DROP	DDL DROP이 일어날 때 트리거가 수행한다.
GRANT	DDL GRANT가 일어날 때 트리거가 수행한다.
NOAUDIT	DDL NOAUDIT이 일어날 때 트리거가 수행한다.
RENAME	DDL RENAME이 일어날 때 트리거가 수행한다.
REVOKE	DDL REVOKE가 일어날 때 트리거가 수행한다.
TRUNCATE	DDL TRUNCATE가 일어날 때 트리거가 수행한다.
DDL	어떠한 DDL이 일어날 때 트리거가 수행한다.
SERVERERROR	EVENT SERVERERROR가 일어날 때 트리거가 수행한다.
LOGON	EVENT LOGON이 일어날 때 트리거가 수행한다.
LOGOFF	EVENT LOGOFF가 일어날 때 트리거가 수행한다.
STARTUP	EVENT STARTUP이 일어날 때 트리거가 수행한다.
SHUTDOWN	EVENT SHUTDOWN이 일어날 때 트리거가 수행한다.
SUSPEND	EVENT SUSPEND가 일어날 때 트리거가 수행한다.

- 예제

  다음은 **CREATE TRIGGER**를 사용해 트리거를 생성하는 예이다.

```
CREATE OR REPLACE TRIGGER "tibero".tr1
 AFTER DELETE ON tbl2 FOR EACH ROW
BEGIN
 IF(DELETING) THEN
   DELETE FROM tbl3
```

```
    WHERE ID  = :OLD_ID;
 END IF;
END;
/
```

다음은 **CREATE TRIGGER**를 사용해 DDL 트리거를 생성하는 예이다.

```
CREATE OR REPLACE TRIGGER ddl_trg_example
 AFTER CREATE ON DATABASE
BEGIN
 INSERT INTO test VALUES (0);
END;
/
```

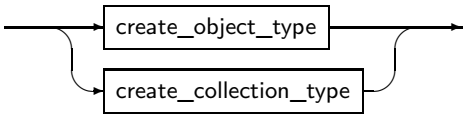
7.44. CREATE TYPE

사용자 정의 타입을 새로 정의하거나 기존의 사용자 정의 타입으로 대체한다. 사용자 정의 타입은 tbPSM 타입으로 사용가능하며, Tibero 서버에서 저장된다.

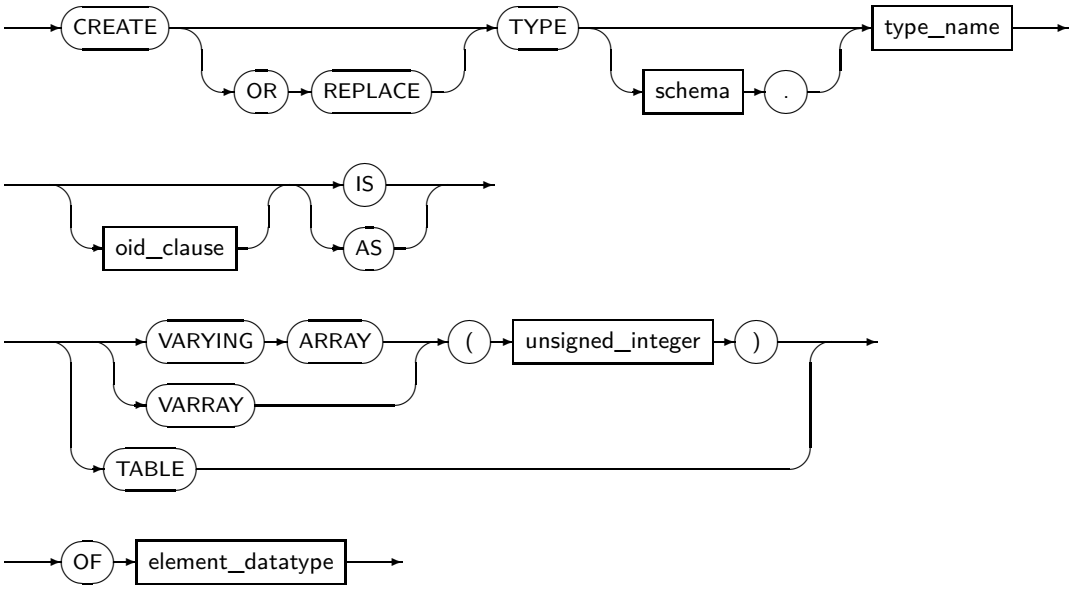
CREATE TYPE의 세부 내용은 다음과 같다.

- 문법

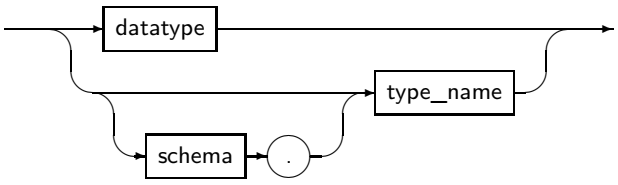
create_type



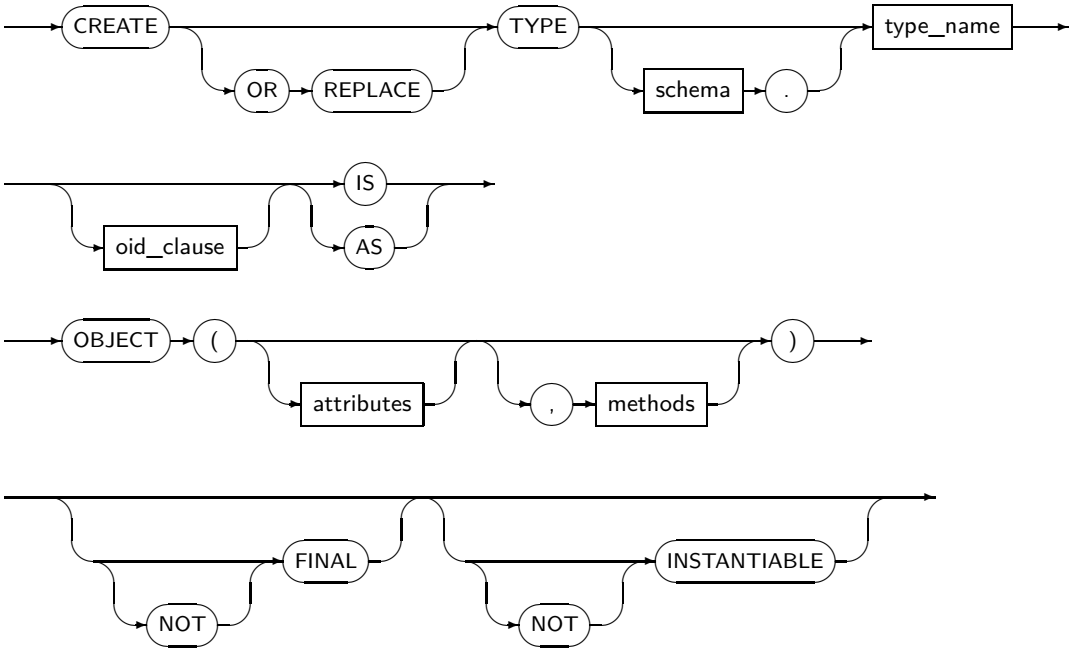
create_collection_type



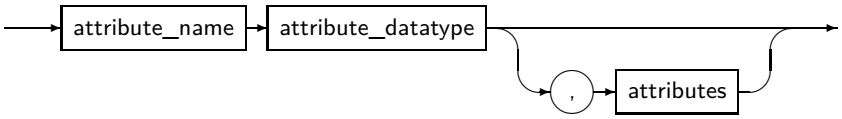
element_datatype



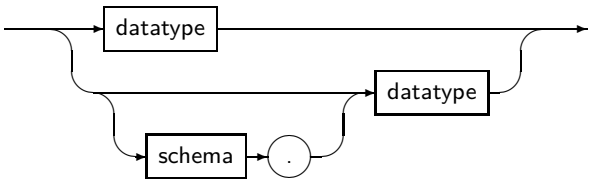
create_object_type



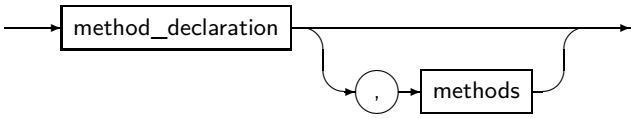
attributes



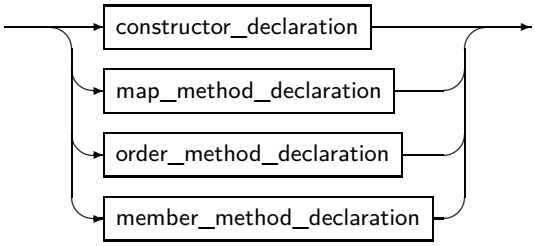
attribute_datatype



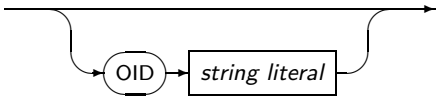
methods



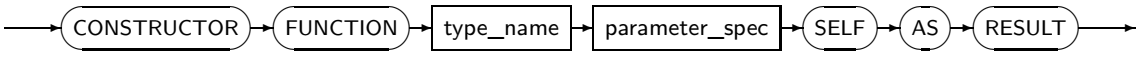
method_declaration



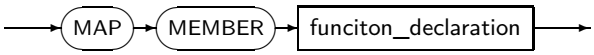
oid_clause_opt



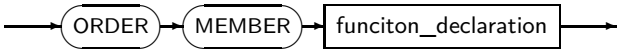
constructor_declaration



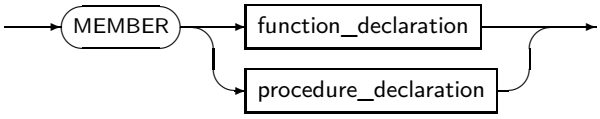
map_method_declaration



order_method_declaration



member_method_declaration



- 특권
  - 사용자가 자신의 스키마에 사용자 정의 타입을 생성하고자 하면 CREATE TYPE 시스템 특권을 부여받아야 한다.
  - 다른 사용자의 스키마에 사용자 정의 타입을 생성하거나 변경하고자 하면, CREATE ANY TYPE 또는 ALTER ANY TYPE 시스템 특권을 부여받아야 한다.
- 구성요소
  - create_type

구성요소	설명
OR REPLACE	이미 존재하는 사용자 정의 타입을 다시 정의하고자 할 때 사용한다. OR REPLACE가 포함되면 해당 타입을 재컴파일한다. 사용자 정의 타입을 제거하고 다시 생성하는 것과의 차이는 OR REPLACE를 이용하면 해당 타입에 기존의 특권이 그대로 유지된다는 점이다.
schema	사용자 정의 타입이 속해 있는 스키마를 명시한다. 생략하면 현재 사용자의 스키마로 인식된다.
type_name	생성하고자 하는 사용자 정의 타입의 이름을 명시한다.
oid_clause	현재 버전에서는 아무런 의미도 없다.
IS	IS나 AS는 기호에 맞게 선택할 수 있으며, 둘의 차이는 없다.

구성요소	설명
	IS 다음에는 타입의 본문이 온다.
AS	IS와 동일하다. AS 다음에는 타입의 본문이 온다.
VARYING ARRAY	생성할 사용자 정의 타입이 배열임을 지정한다. VARRAY와 동일하다.
VARRAY	생성할 사용자 정의 타입이 배열임을 지정한다. VARYING ARRAY와 동일하다.
unsigned_integer	배열의 최대 길이를 지정하는 양수이다.
TABLE	생성할 사용자 정의 타입이 네스티드 테이블 임을 지정한다.
element_datatype	구성 요소의 데이터 타입이다. 구성 요소의 데이터 타입은 tbPSM에서 지원하는 모든 데이터 타입과 사용자 정의 데이터 타입이 가능하다. 데이터 타입의 문자열의 길이, 숫자의 정밀도와 스케일은 지정 가능하다.
OBJECT	생성할 사용자 정의 타입이 오브젝트임을 지정한다.
attribute_name	애트리뷰트 이름을 지정한다.
attribute_datatype	애트리뷰트의 데이터 타입이다. 구성 요소의 데이터 타입은 tbPSM에서 지원하는 모든 데이터 타입과 사용자 정의 데이터 타입이 가능하다. 데이터 타입의 문자열의 길이, 숫자의 정밀도와 스케일은 지정 가능하다.
constructor_declaration	생성자의 선언부이다. 리턴타입은 생성 타입과 동일하나, SELF AS RESULT로 표현한다.
map_method_declaration	맵 매소드의 선언부이다. 리턴 타입은 항상 스칼라타입이며 파라미터는 없거나, SELF를 유일하게 갖는다.
order_method_declaration	오더 매소드의 선언부이다. 리턴 타입은 항상 숫자 타입이며 SELF를 제외하고 같은 타입의 파라미터를 유일하게 갖는다. 선택적으로 SELF를 갖을 수 있다.
normal_method_declaration	매소드의 선언부로 함수나 서브 프로그램을 갖는다. 선택적으로 SELF를 갖을 수 있다.
FINAL	추가 하위 타입의 생성 가능 여부이다. (기본값: FINAL) • FINAL : 하위 타입을 생성할 수 없음 • NOT FINAL : 하위 타입을 생성할 수 있음

구성요소	설명
INSTANTIABLE	개체 인스턴스의 구성 가능 여부이다. (기본값: INSTANTIABLE) • INSTANTIABLE : 개체 인스턴스를 구성할 수 있음 • NOT INSTANTIABLE : 기본 또는 사용자 정의 생성자를 생성할 수 없음

- 예제

  다음은 **CREATE TYPE**를 사용해 사용자 타입을 생성하는 예이다.

```
--VARYING ARRAY TYPE
CREATE OR REPLACE TYPE sample_array_type AS VARRAY(10) OF NUMBER(8, 2);
/

-- TABLE TYPE
CREATE OR REPLACE TYPE sample_nested_table_type AS TABLE OF VARCHAR(1024);
/

-- VARYING ARRAY OF VARYING ARRAY TYPE
CREATE OR REPLACE TYPE array_of_array AS VARRAY(10) OF sample_array_type;
/

-- TABLE OF TABLE TYPE
CREATE OR REPLACE TYPE table_of_table AS TABLE OF sample_nested_table_type;
/
-- VARYING ARRAY OF TABLE TYPE
CREATE OR REPLACE TYPE array_of_table AS VARRAY(10) OF sample_nested_table_type;
/
-- TABLE OF  VARYING ARRAY
CREATE OR REPLACE TYPE table_of_array AS TABLE OF  sample_array_type;
/

-- OBJECT TYPE
CREATE OR REPLACE TYPE person AS OBJECT (name VARCHAR2(1024),
age NUMBER, gender VARCHAR(6));
/

-- OBJECT OF OBJECT TYPE
CREATE OR REPLACE TYPE employee AS OBJECT(pinfo person, department VARCHAR(64));
/

-- COLLECTION OF OBJECT
CREATE OR REPLACE TYPE employees AS VARRAY(100) OF employee;
/

CREATE OR REPLACE TYPE employees AS TABLE OF employee;
```

```
/

-- COLLECTION ATTRIBUTES

CREATE OR REPLACE TYPE scores AS TABLE OF NUMBER;
/

CREATE OR REPLACE TYPE students AS TABLE OF person;
/

CREATE OR REPLACE TYPE class AS OBJECT (code CHAR(10), name VARCHAR(100),
professor person, std students,  mterm_exam scores, final_exam scores);
/



-- OBJECT METHODS
CREATE TYPE constructor_example AS OBJECT (
    attr1 NUMBER,
    attr2 NUMBER,
    CONSTRUCTOR FUNCTION constructor_example(p1 NUMBER) RETURN SELF AS RESULT
);
/

CREATE TYPE BODY map_method_example
AS
    MAP MEMBER FUNCTION map_method_example RETURN VARCHAR
    AS
    BEGIN
        RETURN to_char(attr2);
    END;
END;
/

CREATE TYPE order_method_example AS OBJECT (
    attr1 NUMBER,
    attr2 NUMBER,
    ORDER MEMBER FUNCTION order_method_example(v order_method_example)
    RETURN NUMBER
);
/

CREATE TYPE normal_method_example AS OBJECT (
    attr1 NUMBER,
    attr2 NUMBER,
    MEMBER FUNCTION  normal_function_example(value NUMBER ) RETURN NUMBER,
    MEMBER PROCEDURE normal_procedure_example(SELF IN normal_method_example)
);
```

```
/

-- FINAL & INSTANTIABLE
CREATE TYPE final_example as OBJECT (
    attr1 NUMBER,
    attr2 NUMBER
)
NOT FINAL;
/
CREATE TYPE final_sub_example under final_example;
/

CREATE TYPE instantiable_example as OBJECT (
    attr1 NUMBER,
    atrr2 NUMBER
)
INSTANTIABLE;
/
```

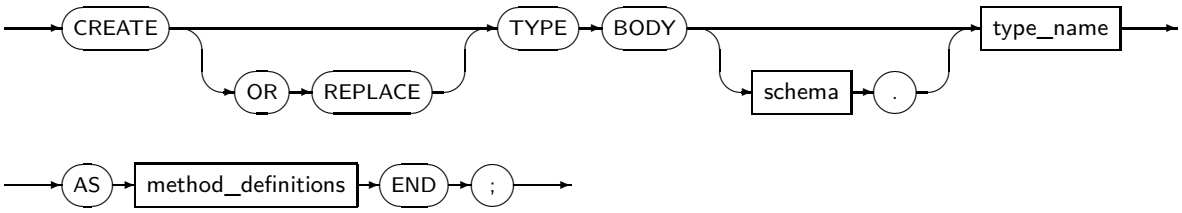
7.45. CREATE TYPE BODY

사용자 정의 타입의 바디를 정의한다. 바디를 정의할수 있는 타입은 오브젝트 타입으로 제한된다.

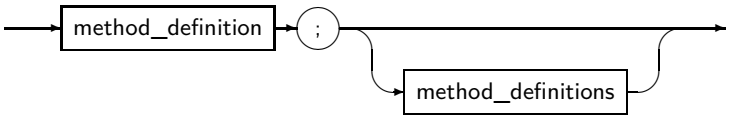
CREATE TYPE BODY의 세부 내용은 다음과 같다.

- 문법

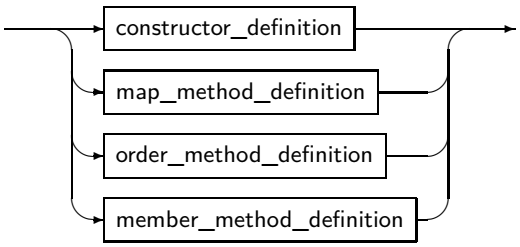
create_type_body



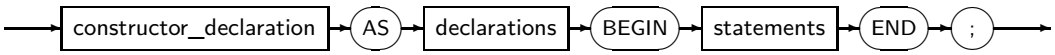
method_definitions



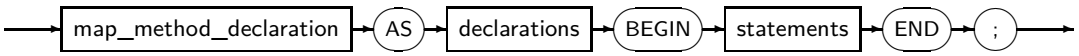
method_definition



constructor_definition

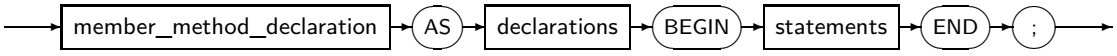


map_method_definition



order_method_definition

member_method_definition



- 특권
  - 사용자가 자신의 스키마에 사용자 정의 타입 바디를 생성하고자 하면 CREATE TYPE 시스템 특권을 부여받아야 한다.
  - 다른 사용자의 스키마에 사용자 정의 타입 바디를 생성하거나 변경하고자 하면, CREATE ANY TYPE 또는 ALTER ANY TYPE 시스템 특권을 부여받아야 한다.
- 구성요소
  - create_type

구성요소	설명
OR REPLACE	이미 존재하는 사용자 정의 타입을 다시 정의하고자 할 때 사용한다. OR REPLACE가 포함되면 해당 타입을 재컴파일한다. 사용자 정의 타입을 제거하고 다시 생성하는 것과의 차이는 OR REPLACE를 이용하면 해당 타입에 기존의 특권이 그대로 유지된다는 점이다.
schema	바디를 생성하고자 하는 사용자 정의 타입이 속해 있는 스키마를 명시한다. 생략하면 현재 사용자의 스키마로 인식된다.
type_name	바디를 생성하고자 하는 사용자 정의 타입의 이름을 명시한다.
constructor_definition	생성자 정의는 생성자 스펙과 선언들, 문장들로 구성되었다. 생성자의 스펙은 CREATE TYPE 문에 선언된 것과 동일해야 하며, 선언들과 문장은 PSM의 일반적인 경우와 동일하다.
map_method_definition	맵 메소드 정의는 생성자 스펙과 선언들, 문장들로 구성되었다. 맵 메소드 스펙은 CREATE TYPE 문과 선언된 것과 동일해야 하며, 선언들과 문장은 PSM의 일반적인 경우와 동일하다.
order_method_definition	오더 메소드 정의는 생성자 스펙과 선언들, 문장들로 구성되었다. 오더 메소드 스펙은 CREATE TYPE 문과 선언된 것과 동일해야 하며, 선언들과 문장은 PSM의 일반적인 경우와 동일하다.
member_method_definition	멤버 메소드 정의는 생성자 스펙과 선언들, 문장들로 구성되었다. 멤버 메소드 스펙은 CREATE TYPE 문과 선언된 것과 동일해야 하며, 선언들과 문장은 PSM의 일반적인 경우와 동일하다.

- 예제

  다음은 **CREATE TYPE BODY**를 사용해 사용자 타입을 생성하는 예이다.

```
-- CONSTRUCTOR를 가지는 TYPE BODY의 예제
DROP TYPE constructor_example;

CREATE TYPE constructor_example AS OBJECT (
    attr1 NUMBER,
    attr2 NUMBER,
    CONSTRUCTOR FUNCTION constructor_example(p1 NUMBER) RETURN SELF AS RESULT
);
/

CREATE TYPE BODY constructor_example
AS
    CONSTRUCTOR FUNCTION constructor_example(p1 NUMBER) RETURN SELF AS RESULT
```

```
    AS
    BEGIN
        attr1 := p1;      -- 입력값을 1번째 애트리뷰트로 지정
        attr2 := p1 * p1; -- 입력값의 제곱을 2번째 애트리뷰트 값으로 지정

        RETURN;
    END;
END;
/

DROP TYPE map_method_example;

CREATE TYPE map_method_example AS OBJECT (
    attr1 NUMBER,
    attr2 NUMBER,
    MAP MEMBER FUNCTION map_method_example RETURN VARCHAR
);
/

CREATE TYPE BODY map_method_example
AS
    -- 맵 메소드는 오브젝트 인스턴스로 부터 스칼라값을 하나 가져온다.
    MAP MEMBER FUNCTION map_method_example RETURN VARCHAR
    AS
    BEGIN
        RETURN to_char(attr2);
    END;
END;
/

DROP TYPE order_method_example;

CREATE TYPE order_method_example AS OBJECT (
    attr1 NUMBER,
    attr2 NUMBER,
    ORDER MEMBER FUNCTION order_method_example(v order_method_example)
RETURN NUMBER
);
/

CREATE TYPE BODY order_method_example
AS
    -- 오더 메소드는 두 오브젝트 타입의 인스턴스를 비교하여 숫자 타입을 리턴한다.
    ORDER MEMBER FUNCTION order_method_example(v order_method_example)
    RETURN NUMBER
    AS
    BEGIN
```

```
        IF attr1  v.attr1 THEN
            RETURN -1;
        ELSIF attr1 = v.attr1 THEN
            RETURN 0;
        ELSE
            RETURN 1;
        END IF;
    END;
END;
/

DROP TYPE normal_method_example;

CREATE TYPE normal_method_example AS OBJECT (
    attr1 NUMBER,
    attr2 NUMBER,
    MEMBER FUNCTION  normal_function_example(value NUMBER ) RETURN NUMBER,
    MEMBER PROCEDURE normal_procedure_example(SELF IN normal_method_example )
);
/

CREATE TYPE BODY normal_method_example
AS
    MEMBER FUNCTION  normal_function_example(value NUMBER ) RETURN NUMBER
    AS
    BEGIN
        return ( attr1 + attr2 ) * value;
    END;
    MEMBER PROCEDURE normal_procedure_example(SELF IN normal_method_example )
    AS
    BEGIN
        dbms_output.put_line ('attribute1:' || self.attr1);
        dbms_output.put_line ('attribute2:' || self.attr2);
    END;

END;
/
```

7.46. CREATE USER

새로운 사용자를 생성한다.

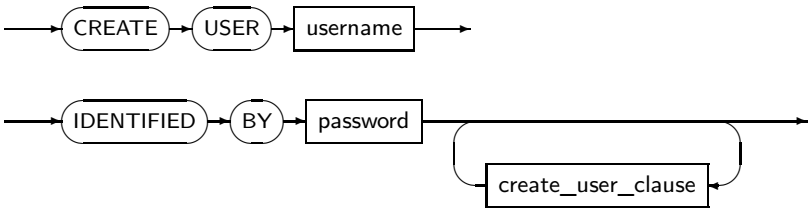
참고

사용자의 정보를 변경하거나, 제거하기 위해서는 “7.19. ALTER USER” , “7.67. DROP USER”의 내용을 참고한다.

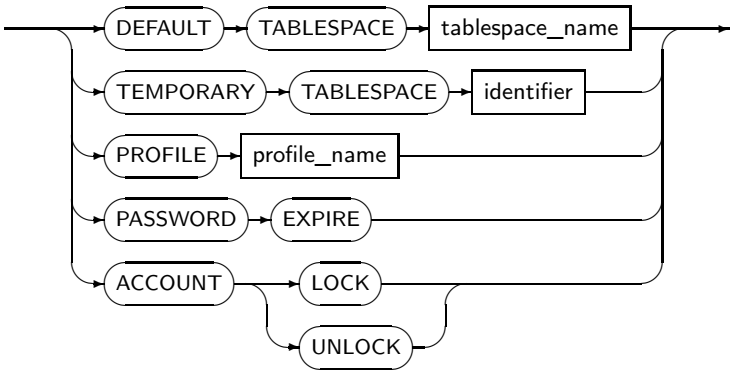
CREATE USER의 세부 내용은 다음과 같다.

- 문법

create_user



create_user_clause



- 특권
  - 사용자를 생성하기 위해서는 CREATE USER 시스템 특권이 필요하다.
  - CREATE USER로 생성한 사용자는 처음에 아무런 특권이 없는 상태이다. 따라서 생성한 사용자로 접속하기 위해서는 CREATE SESSION 시스템 특권이 필요하고, 그 밖에도 해당 시스템 특권이 필요하다. 시스템 특권에 대한 자세한 내용은 "7.71. GRANT"를 참고한다.

- 구성요소
  - create_user

구성요소	설명
username	생성할 사용자의 이름이다. 사용자의 이름은 VARCHAR 데이터 타입으로 저장되고 길이는 최대 30자까지 가능하다. 사용자의 이름은 전체 데이터베이스의 다른 사용자 또는 역할의 이름과 중복되면 안 된다.
IDENTIFIED BY password	생성한 사용자를 인증하기 위한 패스워드이다. 패스워드는 문자열로서 임의의 문자를 사용할 수 있다.

구성요소	설명
	특수 문자를 쓰기 위해서는 따옴표(' ' 또는 " ")를 사용해서 패스워드를 감싸야 한다. 대소문자가 구분되는 패스워드를 이용하고 싶다면 작은따옴표(' ')로 패스워드를 감싸야 한다. 길이는 최대 63자(ASCII 문자 기준)까지 가능하다. 특정 클라이언트나 데이터베이스 관리 프로그램에서 긴 패스워드를 지원하지 않을 수도 있으니, 패스워드는 될 수 있으면 20자 이내로 생성할 것을 권장한다.
create_user_clause	생성한 사용자의 기본값을 설정하기 위한 부분으로 생략할 수 있다.

– create_user_clause

구성요소	설명
DEFAULT TABLESPACE	생성한 사용자가 사용할 디폴트 테이블 스페이스를 지정한다. 해당 테이블 스페이스는 미리 생성되어 있어야 한다. DEFAULT TABLESPACE 부분을 생략하면, 자동으로 시스템 테이블 스페이스를 사용하게 된다. 하지만, 시스템 테이블 스페이스는 시스템 내부에서 사용하는 것이므로 될 수 있으면 별도의 테이블 스페이스를 만들어 사용할 것을 권장한다. 테이블 스페이스를 생성하는 방법은 “7.42. CREATE TABLESPACE”를 참고한다.
PROFILE	사용자의 접속 및 패스워드 정책에 대한 프로파일을 지정한다.
PASSWORD EXPIRE	사용자의 패스워드를 사용기간 만료 상태로 생성하여, 사용자가 처음 접속할 때 원하는 패스워드를 입력할 수 있도록 한다.
ACCOUNT LOCK	사용자를 잠금 상태로 생성한다. 생략하면 사용자를 잠금해제 상태로 생성한다.
ACCOUNT UNLOCK	사용자를 잠금해제 상태로 생성한다. (기본값)

- 예제

  – create_user

  다음은 사용자를 생성하고, **DEFAULT TABLESPACE** 문장을 사용해 디폴트 테이블 스페이스를 설정하는 예이다.

```
SQL> CREATE TABLESPACE t1 DATAFILE 't1.dbf' SIZE 10M
       EXTENT MANAGEMENT LOCAL UNIFORM SIZE 256K;
Tablespace created.

SQL> CREATE USER u1 IDENTIFIED BY 'p1'
```

```
       DEFAULT TABLESPACE t1;
User created.

SQL> SELECT username, default_tablespace
     FROM dba_users WHERE username='U1';

USERNAME                       DEFAULT_TABLESPACE
------------------------------ ------------------------------
U1                             T1

1 row selected.
```

- create_user_clause

다음은 사용자를 생성하고, **PASSWORD EXPIRE**를 사용해 패스워드를 만료된 상태로 변경하는 예이다.

```
SQL> CREATE USER u2 IDENTIFIED BY 'p2'
       PASSWORD EXPIRE;
User created.

SQL> GRANT CREATE SESSION TO u2;
Granted.

SQL> CONN u2/p2
TBR-17002: password expired.

New password : ***
Retype new password : ***
Password changed
Connected.
```

다음은 사용자를 생성하고, **ACCOUNT LOCK**을 사용해 사용자를 잠금 상태로 변경하는 예이다.

```
SQL> CREATE USER u3 IDENTIFIED BY 'p3'
       ACCOUNT LOCK;
User created.

SQL> SELECT username, account_status
     FROM dba_users WHERE username='U3';

USERNAME                       ACCOUNT_STATUS
------------------------------ ----------------
U3                             LOCKED

1 row selected.
```

```
SQL> GRANT CREATE SESSION TO u3;
Granted.

SQL> CONN u3/p3
TBR-17006: account is locked.
```

7.47. CREATE VIEW

사용자의 스키마 또는 지정된 다른 사용자의 스키마에 속한 뷰를 생성한다.

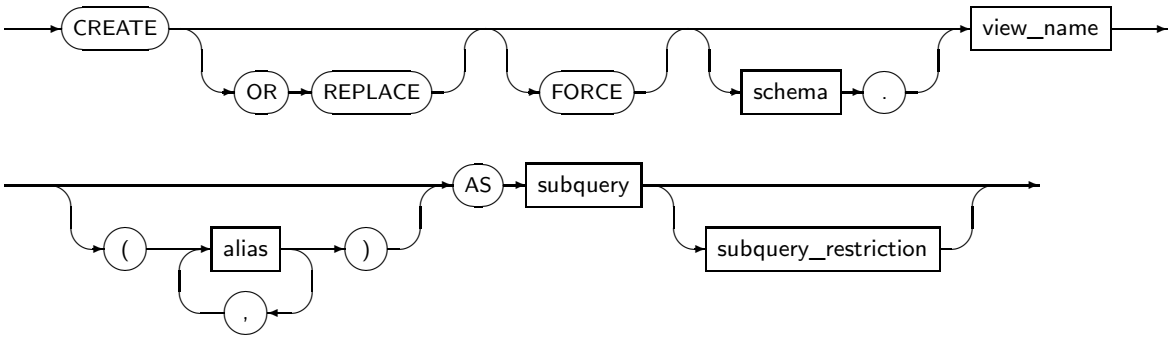
참고

뷰를 제거하기 위해서는 "7.68. DROP VIEW"의 내용을 참고한다.

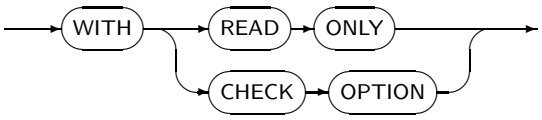
CREATE VIEW의 세부 내용은 다음과 같다.

- 문법

create_view



subquery_restriction



- 특권
  - 뷰를 생성하기 위해서는 사용자가 소유한 스키마에 속한 뷰이거나, CREATE ANY VIEW 시스템 특권이 필요하다.
  - 뷰를 포함하고 있는 스키마의 소유자는 뷰가 참조하는 모든 테이블과 뷰에 대해 SELECT, INSERT, UPDATE, DELETE 중 어느 하나가 가능하도록 해당 특권을 가지고 있어야 한다. 각 문장에 대한 자세한 내용은 각각 "5.1. SELECT", "8.1. INSERT", "8.2. UPDATE", "8.3. DELETE"를 참고한다.
- 구성요소
  - create_view

구성요소	설명
OR REPLACE	기존에 존재하는 뷰라면 제거하고 새롭게 뷰를 만든다. 뷰를 제거하고 나서 다시 생성하는 것과의 차이는 대상 뷰에 관련된 기존의 특권과 참조가 그대로 유지된다는 점이다.
FORCE	뷰가 참조하는 객체의 존재 여부, 참조하는 객체에 대한 권한과 관계없이 강제로 뷰를 생성한다. 뷰를 먼저 생성한 후에 뷰가 참조하는 객체를 생성하거나 권한을 부여받음으로써 뷰를 사용할 수 있다.
schema	생성할 뷰를 포함하는 스키마의 이름이다. 생략하면 현재 사용자의 스키마로 인식된다.
view_name	생성할 뷰의 이름이다. 뷰의 이름은 VARCHAR 데이터 타입으로 저장되고, 길이는 최대 30자까지 가능하다. 뷰의 이름은 테이블과 같은 네임스페이스를 사용하므로 스키마 내의 다른 뷰, 공유 동의어, 테이블, 시퀀스, 패키지, 함수, 프러시저의 이름과 중복되면 안 된다.
alias	부질의에서 결과로 나온 표현식을 지칭하는 별칭이다. 별칭은 VARCHAR 데이터 타입으로 저장되고, 길이는 최대 30자까지 가능하다. 별칭을 생략했을 경우 부질의의 결과로 나온 표현식이 단순한 컬럼명이라면 해당 컬럼명을 별칭으로 설정한다. 부질의의 결과 표현식이 단순한 컬럼명 이상이라면 반드시 별칭을 표시해 주어야 한다. 뷰 컬럼의 개수는 부질의 결과 로우의 컬럼 개수와 동일해야 한다. 별칭은 뷰에서 유일한 이름이어야 한다. ROWID 컬럼을 출력하는 부질의에 대해서는 별칭을 반드시 설정해야 한다.
AS subquery	뷰를 정의하는 부질의이다. 부질의를 설정하는 방법은 SELECT 문장과 같다. 자세한 내용은 “5.1. SELECT”를 참고한다. 뷰를 정의하는 부질의에는 다음과 같은 제약이 있다. – 뷰를 정의하는 부질의의 결과 컬럼으로 시퀀스를 사용할 수 없다. – 뷰를 정의하는 부질의의 결과 컬럼으로 ROWID, ROWNUM, 또는 LEVEL을 사용할 경우 별칭을 반드시 설정해야 한다. – 뷰를 정의하는 부질의의 결과 컬럼에 애스터리스크(*)를 사용했다면 참조 테이블에 컬럼을 추가했을 때 추가된 컬럼이 뷰에 자동으로 추가되지는 않는다. OR REPLACE를 설정해서 새로 만들어야 한다.
subquery_restriction	부질의로 정의되는 뷰의 속성을 지정한다.

– subquery_restriction

구성요소	설명
WITH READ ONLY	부질의에 의해 생성된 뷰를 읽기 전용으로 만들어 수정할 수 없도록 한다. 이 뷰에 대해서는 질의만 가능하다.
WITH CHECK OPTION	WITH CHECK OPTION이 명시된 뷰를 갱신할 때는 갱신되는 로우가 갱신 후에도 뷰에 의해 SELECT가 되는 경우에 한해서만 그 로우의 갱신이 허용된다. 조인 뷰의 경우는 조인 컬럼 및 반복되는 컬럼은 무조건 갱신이 불가능하다는 제약조건이 추가된다. 삽입의 경우도 갱신과 마찬가지로 삽입하는 로우가 뷰에 의해 SELECT가 되어야 한다. 하지만 이것은 기반 테이블이 하나일 경우에만 가능하며 조인 뷰의 경우에는 무조건 삽입이 불가능하다. 삭제의 경우는 삭제의 대상이 되는 키 보존 테이블이 반복되지 않으면 자체 조인을 형성하지 않기 때문에 삭제가 가능하다.

- 예제

  – create_view

  다음은 **As subquery**를 사용해 뷰를 정의하는 부질의의 결과 컬럼으로 시퀀스를 사용했을 때 발생하는 에러의 예이다.

```
SQL> CREATE SEQUENCE s;
Sequence created.

SQL> CREATE VIEW v AS SELECT s.NEXTVAL FROM t;
TBR-8074: sequence not allowed here
at line 1, column 26:
CREATE VIEW v AS SELECT s.NEXTVAL FROM t
```

  다음은 **As subquery**를 사용해 뷰를 정의하는 부질의의 결과 컬럼으로 ROWNUM을 사용할 때 별칭을 설정하지 않으면 에러가 발생하는 예이다.

```
SQL> CREATE VIEW v AS SELECT rownum FROM dual;
TBR-7084: expression not named with a column alias
```

  뷰를 정의하는 부질의의 결과 컬럼으로 ROWID, ROWNUM, 또는 LEVEL을 사용할 경우 반드시 별칭을 설정해야 한다.

  다음은 **As subquery**를 사용해 뷰를 생성할 때 별칭을 지정하지 않았을 경우의 예이다.

```
SQL> CREATE TABLE contract
       (age NUMBER, sex NUMBER, salary NUMBER,
        workplace VARCHAR(30));
```

```
Table created.

SQL> CREATE TABLE factory
       (location VARCHAR(30) UNIQUE, num_of_goods NUMBER);

Table created.

SQL> CREATE VIEW contract_in_seoul AS
     SELECT age, sex, salary * 1000 salary_won
     FROM contract
     WHERE workplace = "Seoul";
```

위의 예에서 특이한 점은 뷰의 별칭을 지정하지 않았고, 부질의의 결과 컬럼으로 표현식(salary * 1000)을 입력했다는 점이다. 하지만, 부질의의 표현식에 별칭(salary_won)을 선언하였으므로 결과적으로 그 별칭을 따르게 된다. 따라서 결과적으로 옳은 표현이다.

- subquery_restriction

다음은 **WITH READ ONLY**를 사용해 읽기 전용으로 뷰를 만드는 예이다. 읽기 전용으로 생성된 뷰는 갱신될 수 없다.

```
SQL> CREATE VIEW read_only_contract AS
    SELECT * FROM contract WITH READ ONLY;

View created.

SQL> SELECT column_name, updatable FROM user_updatable_columns
    WHERE table_name = 'READ_ONLY_CONTRACT';

   COLUMN_NAME                    UPD
   ------------------------------ ---
   AGE                            NO
   SEX                            NO
   SALARY                         NO
   WORKPLACE                      NO

   4 rows selected.
```

다음은 **WITH CHECK OPTION**을 부여하고 만든 뷰이다. 이렇게 생성된 뷰는 갱신할 때 CHECK OPTION을 통과해야 갱신을 할 수 있다.

```
SQL> CREATE VIEW contract_view_with_check AS
    SELECT * FROM contract WHERE age > 18 WITH CHECK OPTION;

View created.
```

```
SQL> INSERT INTO contract_view_with_check
     VALUES (18, 2, 1000, 'New York');
TBR-10010: view WITH CHECK OPTION where-clause violation

SQL> INSERT INTO contract_view_with_check
     VALUES (19, 2, 1000, 'New York');

1 row inserted.
```

갱신 가능한 뷰

갱신 가능한 뷰(Updatable View)는 뷰를 통해 참조 테이블에 삽입, 갱신, 제거를 가능하게 한다. 갱신 가능한 뷰를 통해 어떤 컬럼을 수정할 수 있는지를 알고 싶다면 정적 뷰 USER_UPDATABLE_COLUMNS을 참고한다. 이 뷰는 갱신 가능한 뷰의 어떤 컬럼을 수정할 수 있는지에 관한 정보를 저장하고 있다.

다음은 두 개의 테이블의 조인으로 생성된 뷰이다.

```
SQL>CREATE VIEW contract_view AS
    SELECT c.age, c.salary, f.location, f.num_of_goods
    FROM contract c, factory f
    WHERE c.workplace = f.location;

View created.

SQL> SELECT column_name, updatable FROM user_updatable_columns
    WHERE table_name = 'CONTRACT_VIEW';

COLUMN_NAME                    UPD
------------------------------ ---
AGE                            YES
SALARY                         YES
LOCATION                       NO
NUM_OF_GOODS                   NO

4 rows selected.
```

테이블 factory의 location은 CONTRACT_VIEW에서는 유일한 값을 갖지 않는다. 즉, CONTRACT_VIEW의 로우가 factory의 로우와 1:1 대응이 되지 않는다.

따라서 다음과 같이 **CONTRACT_VIEW** 뷰에 의해서 factory 테이블은 갱신될 수 없다.

```
SQL> INSERT INTO contract_view VALUES (26, 1, 'New York', 0);
TBR-8057: non updatable columns
at line 1, column 2:
INSERT INTO contract_view VALUES (26, 1, 'New York', 0)
```

```
SQL> INSERT INTO contract_view (age, salary) VALUES (26, 1);

1 row inserted.
```

7.48. DROP DATABASE LINK

데이터베이스 링크를 제거한다.

참고

데이터베이스 링크에 대한 자세한 내용은 "Tibero 관리자 안내서"를 참고한다.

DROP DATABASE LINK의 세부 사항은 다음과 같다.

- 문법

drop_database_link

- 특권
  - 사용자가 소유한 스키마에 있는 데이터베이스 링크를 제거할 때는 아무런 권한이 필요하지 않다.
  - 공유 데이터베이스 링크를 제거할 때는 DROP PUBLIC DATABASE LINK 시스템 특권이 있어야 한다.

- 구성요소

구성요소	설명
PUBLIC	공유 데이터베이스 링크를 제거하기 위해 PUBLIC 예약어를 사용한다. 생략하면 사용자 자신의 스키마에 있는 데이터베이스 링크를 제거한다.
dblink_name	제거할 데이터베이스 링크의 이름이다. 주의할 점은 데이터베이스 링크의 이름을 지정할 때 스키마를 지정할 수 없다는 것이다. 데이터베이스 링크에서는 점(.) 기호를 단순히 이름에 포함되는 한 문자로 인식한다. 따라서 다른 사용자의 스키마에 있는 데이터베이스 링크는 제거할 수 없다.

- 예제

다음은 **DROP DATABASE LINK**와 **DROP PUBLIC DATABASE LINK**를 사용하는 예이다.

```
DROP DATABASE LINK remote;

DROP PUBLIC DATABASE LINK public_remote;
```

7.49. DROP DIRECTORY

디렉터리 객체를 제거한다.

참고

디렉터리를 생성하기 위해서는 “7.27. CREATE DIRECTORY”의 내용을 참고한다.

DROP DIRECTORY의 세부 사항은 다음과 같다.

- 문법

  drop_directory

- 특권

  디렉터리를 제거하기 위해서는 DROP ANY DIRECTORY 시스템 특권이 필요하다.

- 구성요소

구성요소	설명
dir_name	제거할 디렉터리 객체의 이름이다.

- 예제

  다음은 **DROP DIRECTORY**를 사용해 'tmp'라는 디렉터리를 제거하는 예이다.

```
SQL> DROP DIRECTORY tmp;

Directory dropped.
```

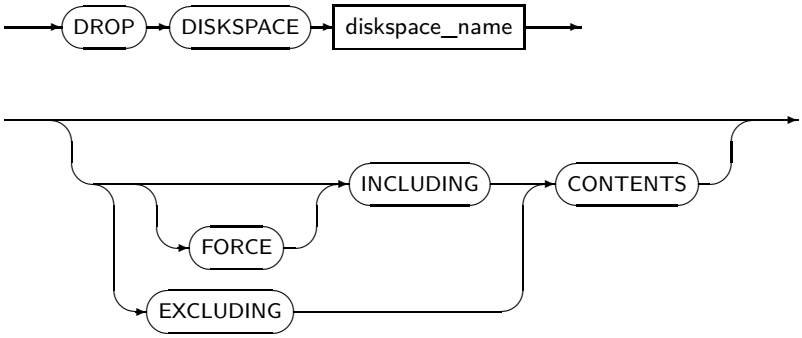
7.50. DROP DISKSPACE

디스크스페이스를 제거한다.

DROP DISKSPACE의 세부 내용은 다음과 같다.

- 문법

drop_diskspace



- 구성요소

  – drop_diskspace

구성요소	설명
diskspace_name	제거할 디스크스페이스의 이름을 명시한다.
FORCE	디스크스페이스를 구성하는 모든 디스크의 디스크 헤더를 초기화해서 다시 마운트할 수 없도록 하기 위해 명시한다.
INCLUDING CONTENTS	디스크스페이스의 모든 파일을 삭제하고 디스크스페이스를 제거한다.
EXCLUDING CONTENTS	디스크스페이스에 사용자 파일이 없는 경우에만 디스크스페이스를 제거한다. 사용자 파일이 있는 경우에는 오류가 발생한다. 제거 조건이 명시되지 않은 경우의 기본 동작이다.

- 예제

  다음은 **DROP DISKSPACE**를 사용해 디스크스페이스를 제거하는 예이다.

```
DROP DISKSPACE ds FORCE INCLUDING CONTENTS;
```

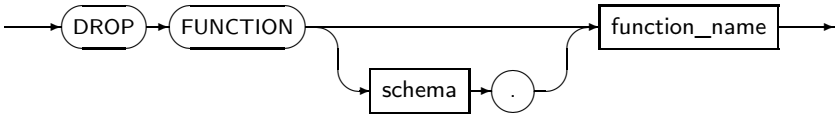
7.51. DROP FUNCTION

사용자 함수를 제거한다. 제거된 함수를 포함하는 뷰나 함수, 프러시저 등은 바로 무효화된다. 이렇게 무효가 된 문장이나 함수, 프러시저를 실행하면 다시 컴파일을 시도하지만 이미 제거되었으므로 에러를 반환한다.

DROP FUNCTION의 세부 사항은 다음과 같다.

- 문법

drop_function



- 특권

  대상 함수가 사용자가 소유한 스키마에 있거나, DROP ANY PROCEDURE 시스템 특권이 있어야 한다.

- 구성요소

구성요소	설명
schema	제거할 함수가 속해있는 스키마의 이름을 명시한다. 생략하면 현재 사용자의 스키마로 인식한다.
function_name	제거할 함수의 이름을 명시한다.

- 예제

  다음은 **DROP FUNCTION**을 사용해 사용자 함수를 제거하는 예이다.

```
DROP FUNCTION tibero.get_square;
```

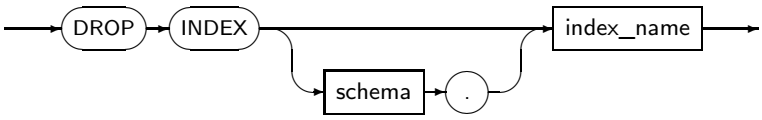
7.52. DROP INDEX

인덱스를 제거한다. 제거된 인덱스를 이용하는 SQL 문장, 함수, 프러시저 등은 모두 무효화된다. 무효화된 문장을 실행시키면 인덱스를 사용하지 않는 방법으로 최적화되거나 컴파일되어 다시 실행된다.

DROP INDEX의 세부 사항은 다음과 같다.

- 문법

  drop_index



- 특권

  다음 중 하나를 만족해야 DROP INDEX 문을 실행할 수 있다.

  – 인덱스가 자신의 스키마에 포함되어 있다.

  – DROP ANY INDEX 시스템 특권이 있다.

- 구성요소

구성요소	설명
schema	인덱스나 테이블의 스키마 이름이다. 생략하면 현재 사용자의 스키마로 인식된다.
index_name	제거할 대상 인덱스의 이름이다.

- 예제

  다음은 **DROP INDEX**를 사용해 인덱스를 제거하는 예이다.

```
DROP INDEX u2.i;
```

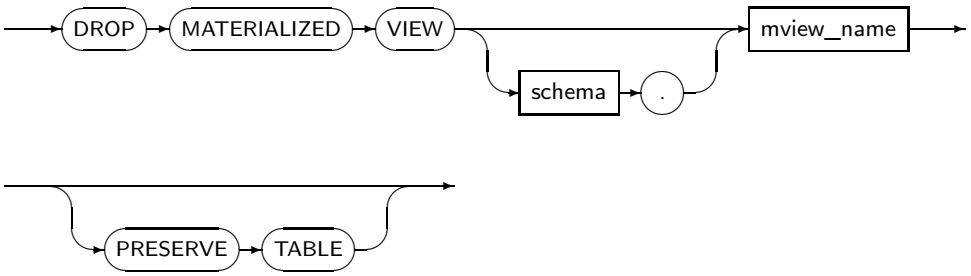
7.53. DROP MATERIALIZED VIEW

실체화 뷰를 제거한다.

DROP MATERIALIZED VIEW의 세부 사항은 다음과 같다.

- 문법

drop_materialized_view



- 특권
  - 사용자가 소유한 스키마에 실체화 뷰를 제거하기 위해서는 별다른 특권이 필요하지 않다.
  - 다른 사용자가 소유한 스키마의 실체화 뷰를 제거하기 위해서는 DROP ANY MATERIALIZED VIEW 시스템 특권이 있어야 한다. 그리고 실체화 뷰를 제거하기 위해서는 데이터베이스가 사용하는 테이블, 뷰, 인덱스 등을 제거할 수 있는 권한이 필요하다.
- 구성요소

구성요소	설명
schema	제거할 실체화 뷰가 속해 있는 스키마의 이름을 명시한다. 생략하면 현재 사용자의 스키마로 인식한다.
mview_name	제거할 실체화 뷰의 이름을 명시한다.
PRESERVE TABLE	PRESERVE TABLE을 명시하면 실체화 뷰를 제거한 후에도 실체화 뷰의 내용이 실체화 뷰와 동일한 이름의 테이블로 남게 된다.

- 예제

  다음은 **PRESERVE TABLE**를 사용해 실체화 뷰를 제거하는 예이다.

```
DROP MATERIALIZED VIEW MV
  PRESERVE TABLE;
```

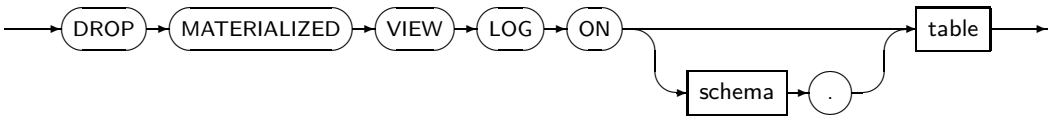
7.54. DROP MATERIALIZED VIEW LOG

지정된 마스터 테이블에 실체화 뷰 로그를 삭제한다.

DROP MATERIALIZED VIEW LOG의 세부 내용은 다음과 같다.

- 문법

  drop_materialized_view_log



- 특권

  테이블을 삭제하기 위한 특권이 있어야 한다.

- 구성요소

구성요소	설명
schema	삭제할 실체화 뷰 로그의 마스터 테이블의 스키마를 명시한다. 생략하면 현재 사용자의 스키마로 인식한다.
table	삭제할 실체화 뷰 로그의 마스터 테이블의 이름을 명시한다.

- 예제

  다음은 **ALTER MATERIALIZED VIEW LOG**를 사용해 실체화 뷰 로그를 변경하는 예이다.

```
DROP MATERIALIZED VIEW LOG ON DEPT;
```

7.55. DROP OUTLINE

생성된 아웃라인을 제거한다.

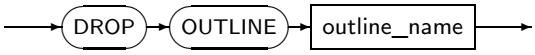
참고

새로운 아웃라인을 생성하려면 “7.33. CREATE OUTLINE”을 참고한다.

DROP OUTLINE의 세부 내용은 다음과 같다.

- 문법

drop_outline



- 구성요소

구성요소	설명
outline_name	제거할 아웃라인의 이름이다.

- 예제

다음은 **DROP OUTLINE** 을 사용해 역할을 제거하는 예이다.

```
SQL> DROP OUTLINE ol_join;
Outline 'OL_JOIN' dropped.
```

USE_STORED_OUTLINES 파라미터를 끄면 해당 쿼리의 아웃라인이 존재하더라도 적용하지 않는다.

```
SQL> alter session set USE_STORED_OUTLINES = n;
```

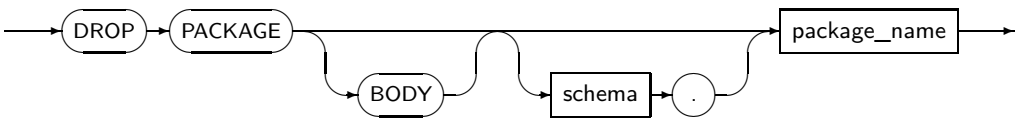
7.56. DROP PACKAGE

데이터베이스에 저장된 패키지를 제거한다.

DROP PACKAGE의 세부내용은 다음과 같다.

- 문법

drop_package



- 특권

제거할 패키지가 사용자의 스키마에 있거나, DROP ANY PROCEDURE 시스템 특권이 있어야 한다.

- 구성요소

구성요소	설명
BODY	BODY를 명시하면 패키지의 본문만 제거된다. 명시하지 않으면 패키지 명세와 본문이 함께 제거된다. 본문만 제거하면 이 패키지의 객체에 의존하는 객체를 무효화시키지는 않는다. 그러나 본문을 다시 생성하기 전까지는 패키지의 프러시저나 함수를 호출할 수 없다. 패키지 전체가 제거되면 패키지 명세에 의존하고 있는 로컬 객체가 무효화된다. 이후 이 객체를 다시 참조하려고 하면 데이터베이스는 객체에 재컴파일을 시도한다. 패키지를 다시 생성하지 않은 상태라면 에러가 발생한다.
schema	제거할 패키지가 속해 있는 스키마의 이름을 명시한다. 생략하면 현재 사용자의 스키마로 인식한다.
package_name	제거할 패키지의 이름을 명시한다.

- 예제

  다음은 **DROP PACKAGE**를 사용하는 예이다.

```
DROP PACKAGE parts_pkg;

DROP PACKAGE BODY tibero.test_pkg;
```

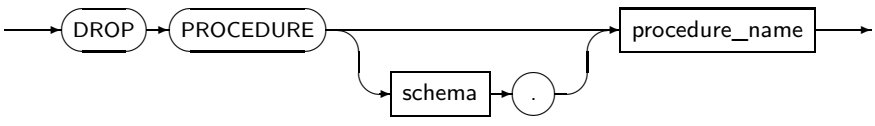
7.57. DROP PROCEDURE

사용자 프러시저를 제거한다. 제거된 사용자 프러시저를 포함하는 함수, 프러시저 등은 바로 무효화된다. 이렇게 무효화된 함수나 프러시저를 실행하려 하면 재컴파일을 시도하지만 이미 제거되었으므로 에러를 반환한다.

DROP PROCEDURE의 세부 내용은 다음과 같다.

- 문법

  drop_procedure



- 특권

  대상 프러시저가 사용자 자신의 스키마에 있거나, DROP ANY PROCEDURE 시스템 특권이 있어야 한다.

- 구성요소

구성요소	설명
schema	제거할 프러시저가 속해있는 스키마의 이름을 명시한다. 생략하면 현재 사용자의 스키마로 인식한다.
procedure_name	제거할 프러시저의 이름을 명시한다.

- 예제

  다음은 **DROP PROCEDURE**를 사용하는 예이다.

```
DROP PROCEDURE raise_salary;
```

7.58. DROP PROFILE

프로파일을 제거한다.

> **참고**
>
> 프로파일을 생성하거나 변경하기 위해서는 “7.37. CREATE PROFILE”, “7.10. ALTER PROFILE”의 내용을 참고한다.

DROP PROFILE의 세부 사항은 다음과 같다.

- 문법

  drop_profile

- 특권

  DROP PROFILE 시스템 특권이 필요하다.

- 구성요소

구성요소	설명
username	제거할 프로파일의 이름이다.
CASCADE	제거할 프로파일에 사용자가 속해 있는 경우 해당 사용자 는 자동으로 DEFAULT 프로파일을 지정한다. 사용자가 속해 있는 프로파일을 CASCADE 절을 포함하지 않은 상태에서 제거하려고 하면, 에러를 반환하고 프로파일을 제거하지 않는다.

7.59. DROP ROLE

생성된 역할을 제거한다. 역할을 제거할 때에는 자동으로 해당 역할을 부여받은 모든 사용자와 다른 역할에서부터 역할을 취소한다. 현재 그 역할을 이용 중인 세션은 영향이 없으며, 이후에 생성되는 세션은 그 역할을 사용할 수 없다.

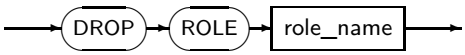
참고

새로운 역할을 생성하거나 생성한 역할의 정보를 바꾸려면 “7.38. CREATE ROLE”, “7.11. ALTER ROLE”을 참고한다.

DROP ROLE의 세부 내용은 다음과 같다.

- 문법

  drop_role

- 특권

  DROP ROLE 시스템 특권이 필요하다.

- 구성요소

구성요소	설명
role_name	제거할 역할의 이름이다. 해당 역할은 이미 CREATE ROLE을 통해 생성되어 있어야 한다.

- 예제

  다음은 **DROP ROLE**을 사용해 역할을 제거하는 예이다.

```
SQL> CONN sys/tibero
Connected.

SQL> CREATE USER u1 IDENTIFIED BY 'p1';
User created.

SQL> CREATE ROLE a;
Role created.

SQL> GRANT a TO u1;
Granted.

SQL> SELECT grantee, granted_role
     FROM dba_role_privs
```

```
     WHERE granted_role='A';

GRANTEE                        GRANTED_ROLE
------------------------------ ------------------------------
SYS                            A
U1                             A

2 rows selected.

SQL> DROP ROLE a;
Role dropped.

SQL> SELECT grantee, granted_role
     FROM dba_role_privs
     WHERE granted_role='A';

GRANTEE                        GRANTED_ROLE
------------------------------ ------------------------------

0 row selected.
```

위의 예를 보면, CREATE ROLE을 사용하여 역할을 생성하고, 생성된 역할을 u1이라는 사용자에게 허가한다. 그러고 나서 DROP ROLE로 역할을 제거하면 자동으로 사용자 u1에게 부여한 역할도 취소된다.

7.60. DROP SEQUENCE

지정된 시퀀스의 정의를 제거한다.

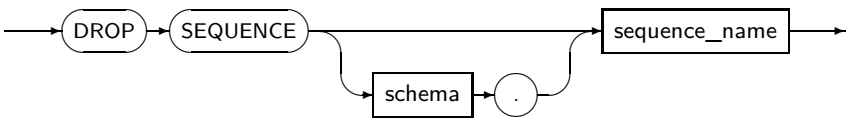
참고

시퀀스를 생성, 변경하기 위해서는 "7.39. CREATE SEQUENCE"와 "7.13. ALTER SEQUENCE"의 내용을 참고한다.

DROP SEQUENCE의 세부 내용은 다음과 같다.

- 문법

drop_sequence



- 특권

  시퀀스가 자신의 스키마에 있거나, DROP_ANY_SEQUENCE 시스템 특권이 필요하다.

- 구성요소

구성요소	설명
schema	제거할 시퀀스를 포함하는 스키마의 이름이다. 생략하면 현재 사용자의 스키마로 인식된다.
sequence_name	제거할 시퀀스의 이름이다.

- 예제

다음은 CREATE_SEQUENCE의 예제에서 생성한 'test_seq'라는 시퀀스를 제거하는 예이다.

```
SQL> DROP SEQUENCE test_seq;

Sequence dropped.
```

7.61. DROP SYNONYM

동의어를 제거한다.

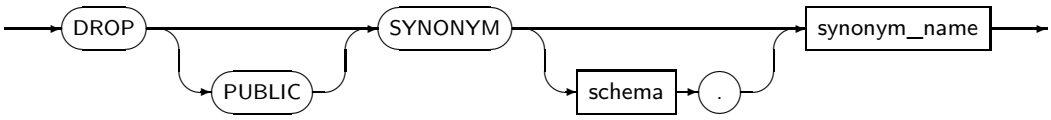
참고

동의어를 생성하기 위해서는 "7.40. CREATE SYNONYM"의 내용을 참고한다.

DROP SYNONYM의 세부 내용은 다음과 같다.

- 문법

drop_synonym



- 특권
  - 사용자가 소유한 스키마에 속한 동의어이거나, DROP ANY SYNONYM 시스템 특권이 필요하다.
  - 공유 동의어를 제거하기 위해서는 DROP PUBLIC SYNONYM 특권이 필요하다.

- 구성요소

구성요소	설명
PUBLIC	공유 동의어를 제거하려면 PUBLIC를 명시해야 한다. 이때는 동의어의 스키마를 지정할 수 없다.

구성요소	설명
schema	제거할 동의어를 포함하는 스키마의 이름이다. 생략하면 현재 사용자의 스키마로 인식한다.
synonym_name	제거할 동의어의 이름이다.

- 예제

  다음은 CREATE_SYNOMYM의 예제에서 생성한 emp와 tabs라는 동의어를 제거하는 예이다.

```
SQL> DROP SYNONYM emp;

Synonym dropped.

SQL> DROP PUBLIC SYNONYM tabs;

Synonym dropped.
```

7.62. DROP TABLE

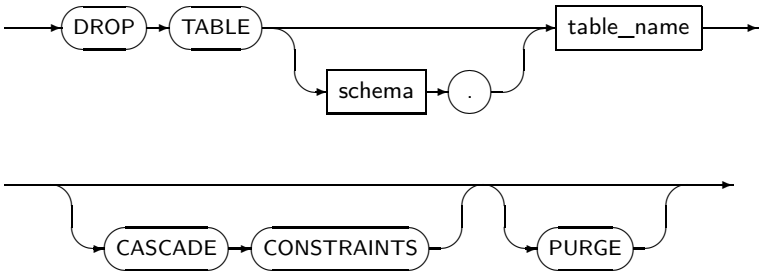
테이블을 제거한다. 테이블을 제거하면 테이블에 생성된 모든 제약 조건과 인덱스, 트리거 등 관련 객체도 제거되며, 파티션 테이블인 경우 모든 파티션이 제거된다.

테이블을 참조하고 있던 뷰, 함수, 프러시저, 패키지 등은 제거되지는 않으나 무효화 상태가 되므로 사용할 수 없게 된다. 이후에 다시 같은 이름의 테이블이 생성되고, 제거된 객체를 사용할 수 있는 조건이 된다면, 첫 번째 사용 후 다시 유효화 상태로 돌아온다.

DROP TABLE의 세부 내용은 다음과 같다.

- 문법

drop_table



- 특권

  사용자의 스키마에 있는 테이블을 제거하기 위해서는 별다른 특권이 필요하지 않다. 단, 다른 사용자의 스키마에 있는 테이블을 제거하기 위해서는 DROP ANY TABLE 시스템 특권이 있어야 한다.

- 구성요소

구성요소	설명
schema	제거할 테이블을 포함하고 있는 스키마의 이름이다. 생략하면 현재 사용자의 스키마로 인식된다.
table_name	제거할 테이블의 이름이다.
CASCADE CONSTRAINTS	제거할 테이블이 다른 테이블에 참조되는 FOREIGN KEY 제약조건이 설정된 상태라면 CASCADE CONSTRAINTS를 통해 FOREIGN KEY를 참조하는 테이블의 제약조건까지 연쇄적으로 제거되도록 설정해야만 테이블을 제거할 수 있다.
PURGE	초기화 파라미터 USE_RECYCLEBIN를 'Y'로 설정하여 RECYCLEBIN 기능을 사용하는 경우에는 PURGE 옵션을 지정하지 않으면 테이블을 제거해도 실제로 제거되지는 않고 이름만 바뀌게 된다. 반면에 초기화 파라미터 USE_RECYCLEBIN에 설정된 값이 'N'인 경우에는 PURGE 옵션과 상관없이 항상 테이블이 실제로 제거된다. 테이블을 실제로 제거하기 위해서는 PURGE 옵션을 지정해야 한다. 제거된 이후에는 FLASHBACK TABLE 문장을 통해 테이블을 복구할 수 있다. FLASHBACK TABLE 문장에 대한 자세한 내용은 "7.70. FLASHBACK TABLE"을 참고한다.

- 예제

  다음은 **DROP TABLE**을 사용해 테이블을 제거하는 예이다.

```
DROP TABLE table_exmp;

DROP TABLE table_exmp_cas CASCADE CONSTRAINTS;
```

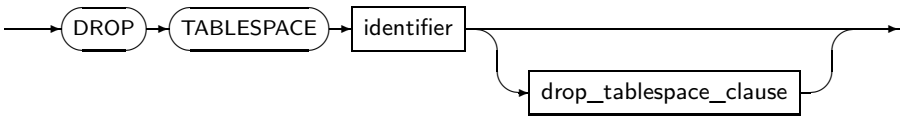
7.63. DROP TABLESPACE

테이블 스페이스를 제거한다.

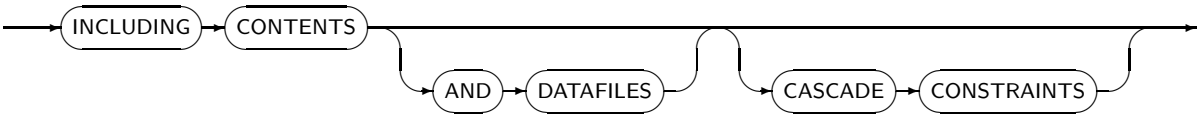
DROP TABLESPACE의 세부 내용은 다음과 같다.

- 문법

drop_tablespace



drop_tablespace_clause



- 특권

  SYSDBA 특권이 있어야 한다.

- 구성요소

구성요소	설명
identifier	제거할 테이블 스페이스의 이름을 명시한다.
INCLUDING CONTENTS	테이블 스페이스의 모든 객체를 동시에 제거한다. INCLUDING CONTENTS를 명시하지 않고, 객체를 저장하는 테이블 스페이스를 제거하려 하면, 에러를 반환하고 테이블 스페이스를 제거하지 않는다.
AND DATAFILES	테이블 스페이스 제거와 함께 구성하는 데이터 파일을 제거하려고 할 때 명시한다.
CASCADE CONSTRAINTS	제거할 테이블 스페이스에 저장된 테이블을 참조하는 모든 참조 무결성 제약조건도 함께 제거한다. 테이블 스페이스의 테이블을 참조하는 참조 무결성 제약조건이 정의되어 있는데, CASCADE CONSTRAINTS를 명시하지 않고 해당 테이블 스페이스를 제거하려고 하면, 에러를 반환하고 해당 테이블 스페이스를 제거하지 않는다.

- 예제

  다음은 테이블 스페이스를 제거할 때 **INCLUDING CONTENTS**를 포함시켜 테이블 스페이스와 해당 객체와 데이터 파일까지 모두를 제거하는 예이다.

```
DROP TABLESPACE ts1;

DROP TABLESPACE ts1 INCLUDING CONTENTS AND DATAFILES;
```

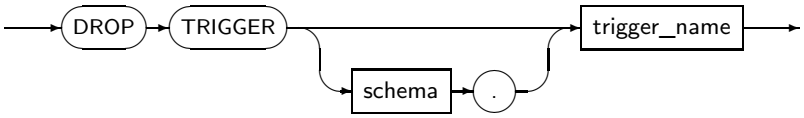
7.64. DROP TRIGGER

데이터베이스에서 트리거를 제거한다. 만약 트리거가 설정된 테이블을 DROP TABLE 문으로 제거했다면, 트리거는 자동으로 제거된다. 즉 DROP TRIGGER 문을 실행하지 않아도 된다.

DROP TRIGGER의 세부 내용은 다음과 같다.

- 문법

drop_trigger



- 특권

사용자가 소유한 스키마에 있는 트리거이거나, DROP ANY TRIGGER 시스템 특권이 있어야 한다.

- 구성요소

구성요소	설명
schema	제거할 트리거가 속해 있는 스키마를 명시한다. 생략하면 현재 사용자의 스키마로 인식한다.
trigger_name	제거할 트리거의 이름을 명시한다.

- 예제

다음은 **DROP TRIGGER**를 사용해 'emp'라는 스키마에 속해 있는 트리거를 제거하는 예이다.

```
DROP TRIGGER emp.bonus_check;
```

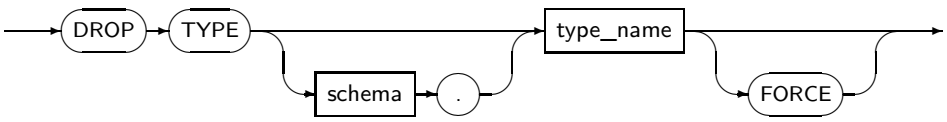
7.65. DROP TYPE

사용자 정의 타입를 제거한다. 제거된 사용자 정의 타입를 포함하는 함수, 프러시저, 패키지는 바로 무효화된다. 이렇게 무효가 된 패키지 함수, 프러시저, 패키지를 실행하면 다시 컴파일을 시도하지만 이미 제거되었으므로 에러를 반환한다. 단, 제거할 사용자 정의 타입을 포함하는 사용자 정의 타입이 존재한다면, FORCE 옵션을 주어야 사용자 정의 타입의 제거가 성공하며, 제거된 사용자 정의 타입을 포함하던 사용자 정의 타입은 바로 무효화된다. 무효화된 타입은 이를 사용하는 함수, 프러시저, 패키지의 실행될 때 또는 이 사용자 정의 타입을 포함하는 다른 사용자 정의 타입이 컴파일될 때 컴파일을 시도한다.

DROP TYPE의 세부 사항은 다음과 같다.

- 문법

drop_type



- 특권

대상 사용자 정의 타입이 사용자가 소유한 스키마에 있거나, DROP ANY TYPE 시스템 특권이 있어야 한다.

- 구성요소

구성요소	설명
schema	제거할 사용자 정의 타입이 속해 있는 스키마의 이름을 명시한다. 생략하면 현재 사용자의 스키마로 인식한다.
type_name	제거할 사용자 정의 타입의 이름을 명시한다.
FORCE	제거할 사용자 정의 타입를 포함하는 사용자 정의 타입이 있는 경우에도 제거를 강제한다.

- 예제

다음은 **DROP TYPE**을 사용해 사용자 타입를 제거하는 예이다.

```
CREATE TYPE tibero.two_dimensional_array  AS VARRAY (100) OF one_dimensional_array;
/

CREATE TYPE tibero.one_dimensional_array AS VARRAY (100) OF NUMBER;
/

DROP TYPE tibero.one_dimensional_array FORCE;

DROP TYPE tibero.two_dimensional_array;
```

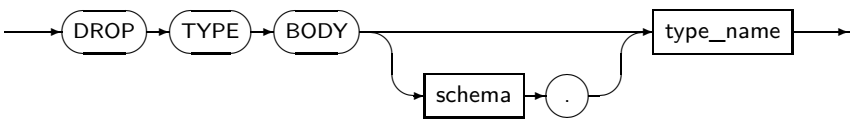
7.66. DROP TYPE BODY

바디를 가지는 사용자 정의 타입 즉 오브젝트 타입의 바디를 제거한다. 바디를 제거해도 타입의 스펙이 무효화되지 않는다. 타입의 스펙이 무효화 되지 않으므로, 스펙을 의존하는 다른 스키마 오브젝트를 무효화하지 않는다.

DROP TYPE BODY의 세부 사항은 다음과 같다.

- 문법

drop_type_body



- 특권

대상 사용자 정의 타입이 사용자가 소유한 스키마에 있거나 DROP ANY TYPE 시스템 특권이 있어야 한다.

- 구성요소

구성요소	설명
schema	바디를 제거할 사용자 정의 타입이 속해 있는 스키마의 이름을 명시한다. 생략하면 현재 사용자의 스키마로 인식한다.
type_name	바디를 제거할 사용자 정의 타입의 이름을 명시한다.

- 예제

다음은 **DROP TYPE BODY**을 사용해 사용자 타입를 제거하는 예이다.

```
CREATE TYPE object_type AS OBJECT(
    c1 NUMBER,
    c2 NUMBER,
    MEMBER PROCEDURE print_attribute);
/

CREATE TYPE  BODY object_type
AS
    MEMBER PROCEDURE print_attribute
    AS
    BEGIN
            dbms_output.put_line( 'C1:' || c1 );
            dbms_output.put_line( 'C2:' || c2 );
    END;
END;
/

DROP TYPE BODY object_type;
```

7.67. DROP USER

사용자를 제거한다.

참고

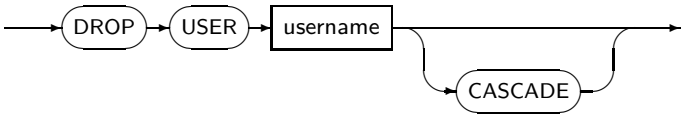
1. 사용자를 생성하거나 변경하기 위해서는 “7.46. CREATE USER” , “7.19. ALTER USER”의 내용을 참고한다.

2. SYS 사용자는 절대로 제거하면 안 된다.

DROP USER의 세부 사항은 다음과 같다.

- 문법

drop_user



- 특권

DROP USER 시스템 특권이 필요하다.

- 구성요소

구성요소	설명
username	제거할 사용자의 이름이다.
CASCADE	제거할 사용자가 객체를 가진 경우 해당 객체와 함께 사용자를 제거한다. 객체를 가진 사용자를 CASCADE 절을 포함하지 않은 상태에서 제거하려고 하면, 에러를 반환하고 사용자를 제거하지 않는다.

- 예제

다음은 **DROP USER**를 사용해 사용자를 제거하는 예이다.

```
SQL> CREATE USER u1 IDENTIFIED BY 'p1';
User created.

SQL> DROP USER u1;
User dropped.

SQL> CREATE USER u2 IDENTIFIED BY 'p2';
User created.

SQL> GRANT CREATE SESSION TO u2;
Granted.

SQL> GRANT RESOURCE TO u2;
Granted.

SQL> CONN U2/p2
Connected.

SQL> CREATE TABLE t1 (a NUMBER, b NUMBER);
Table created.

SQL> CONN sys/tibero
Connected.
```

```
SQL> DROP USER u2;
TBR-7134: cascade is required to remove this user from the system

SQL> DROP USER u2 CASCADE;
User dropped.
```

처음에 사용자를 생성했을 때는 해당 사용자가 아무런 객체를 만들지 않았으므로, DROP USER를 사용하여 사용자를 제거할 수 있었다. 두 번째 사용자를 생성할 때는 해당 사용자로 접속하여 테이블을 생성하였다. 해당 사용자는 **t1**이라는 테이블을 소유하고 있으므로, 단순한 DROP USER로는 제거할 수 없고, CASCADE 절을 덧붙여 사용했을 때만 제거할 수 있다.

7.68. DROP VIEW

사용자가 소유한 스키마나 지정된 스키마에 속한 뷰를 제거한다.

제거된 뷰를 참조하는 다른 뷰, 동의어 등은 제거되지 않고 무효화된다. 제거된 뷰를 대체하는 뷰를 생성함으로써 그러한 객체를 다시 유효화할 수 있다.

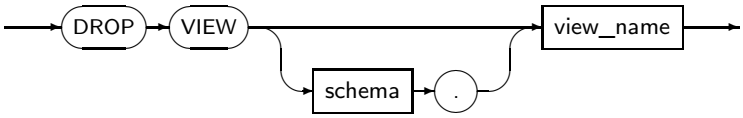
참고

뷰를 생성하기 위해서는 “7.47. CREATE VIEW”의 내용을 참고한다.

DROP VIEW의 세부 내용은 다음과 같다.

- 문법

drop_view



- 특권

  사용자가 소유한 스키마에 속한 뷰이거나 DROP ANY VIEW 시스템 특권이 필요하다.

- 구성요소

구성요소	설명
schema	제거할 뷰를 포함하는 스키마의 이름이다. 생략하면 현재 사용자의 스키마로 인식된다.
view_name	제거할 뷰의 이름이다.

- 예제

다음은 **DROP VIEW**를 사용해 뷰를 제거하는 예이다.

```
SQL> DROP VIEW v;
View dropped.
```

다음은 **DROP VIEW**를 사용해 뷰를 제거하고, 그 뷰를 참조하던 객체를 다시 **유효화**하는 예이다.

```
SQL> CREATE TABLE t (a NUMBER, b NUMBER, c NUMBER);
Table created.

SQL> INSERT INTO t VALUES (123, 456, 789);
1 row inserted.

SQL> CREATE VIEW v1 AS SELECT a, b FROM t;
View created.

SQL> CREATE VIEW v2 AS SELECT a FROM v1;
View created.

SQL> select * from v2;
         A
----------
       123

1 row selected.

SQL> DROP VIEW V1;
View dropped.

SQL> SELECT * FROM v2;
TBR-8088: view V2 has errors
at line 1, column 16:
SELECT * FROM v2;


SQL> CREATE VIEW v1 AS SELECT a FROM t;
View created.

SQL> SELECT * FROM v2;
         A
----------
       123

1 row selected.
```

위의 예에서 뷰 v1을 제거하면 v1을 참조하던 v2를 더는 사용할 수 없다. 하지만, 뷰의 정의는 달라도 대체할 수 있는 v1을 만들어 주면 다시 유효화하여 사용할 수 있다.

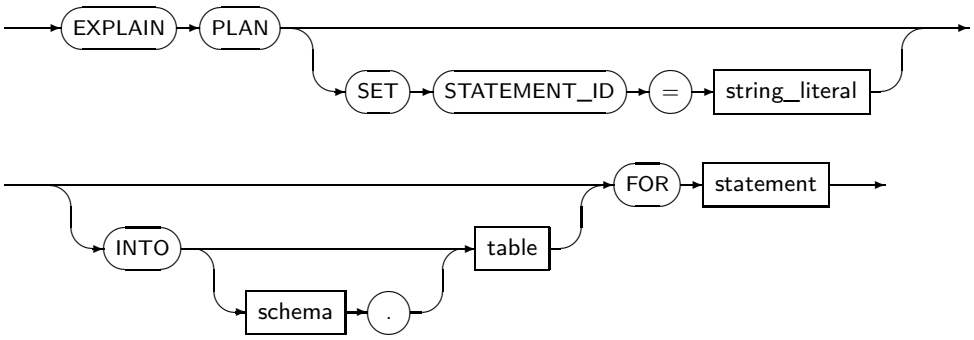
7.69. EXPLAIN PLAN

특정 SQL 문장의 실행 계획을 지정된 테이블에 저장한다. 디폴트로 PLAN_TABLE 테이블에 결과가 저장되며, PLAN_TABLE 테이블과 같은 형태의 스키마를 갖는 테이블을 지정하면 해당 테이블에 결과를 삽입할 수 있다. EXPLAIN PLAN 문은 DML에 가깝다. 그래서 SQL 문장을 실행하면 자동 커밋은 발생하지 않는다.

EXPLAIN PLAN의 세부 내용은 다음과 같다.

- 문법

explain_plan



- 특권

  결과를 저장할 테이블에 대한 쓰기 권한이 필요하다.

- 구성요소

구성요소	설명
SET STATEMENT_ID	결과를 저장할 테이블의 STATEMENT_ID 컬럼의 값을 지정한다. 결과 테이블의 다른 EXPLAIN PLAN 결과와 구별하기 위해 이 컬럼을 사용한다. 아무것도 지정하지 않으면 NULL 값이 삽입된다.
INTO table	결과를 저장할 테이블의 이름을 지정한다. EXPLAIN PLAN 문을 실행하기 전에 이미 존재하는 테이블이어야 한다. 스키마를 생략하면 사용자 자신의 스키마로 자동 지정된다.
FOR statement	실행 계획을 관찰할 SELECT, INSERT, UPDATE, DELETE 문장을 지정한다.

- 예제

  다음은 **EXPLAIN PLAN**을 사용하여 실행 계획을 결과 테이블에 저장하는 예이다.

```
EXPLAIN PLAN SET STATEMENT_ID = 'Plan Example 1' INTO plan_table
FOR UPDATE employees SET salary = salary * 1.10
    WHERE dept_id = (SELECT dept_id FROM dept WHERE location_id = 1200);
```

7.70. FLASHBACK TABLE

테이블을 특정 시점으로 돌리거나 제거한 테이블을 복원한다. 특정 시점으로 테이블을 복원할 경우 복원할 테이블의 이름을 원래 테이블의 이름과 다르게 하여 복구 후에도 기존 테이블이 남아 있도록 한다. 따라서 기존 테이블로 다른 시점의 FLASHBACK TABLE이 가능해진다.

FLASHBACK TABLE은 특정 시점의 데이터만을 복원해주며, 테이블과 관련된 인덱스, 트리거, 제약조건은 복원되지 않는다. 또한 초기화 파라미터 UNDO_RETENTION 시점까지 FLASHBACK TABLE이 가능하지만 _TSN_TIME_MAP_SIZE의 크기가 UNDO_RETENTION보다 같거나 커야 한다. 이는 $TB_SID.tip 파일에서 옵션으로 값을 설정할 수 있다.

FLASHBACK TABLE은 DDL이 수행된 이전의 시점으로는 불가능하며, 테이블 이외의 오브젝트에는 수행되지 않는다.

다음은 FLASHBACK TABLE이 불가능한 테이블이다.

- 시스템 테이블
- 가상 테이블
- AQ 테이블
- 파티션 테이블
- Cluster에 속해 있는 테이블
- TEMP 테이블
- External 테이블
- DB Link를 통해 조회할 수 있는 테이블

만약 제거한 테이블을 복원할 경우에는 초기화 파라미터 USE_RECYCLEBIN을 Y로 설정하여 RECYCLEBIN 기능을 사용해야 한다. 단, 다음과 같은 경우에는 USE_RECYCLEBIN을 Y로 설정하더라도 RECYCLEBIN으로 오브젝트가 들어가지 않아 FLASHBACK TABLE이 불가능하다.

- TEMP 테이블을 DROP할 경우
- SYS 계정의 오브젝트를 DROP할 경우
- ALTER MOVE한 경우 원본 테이블

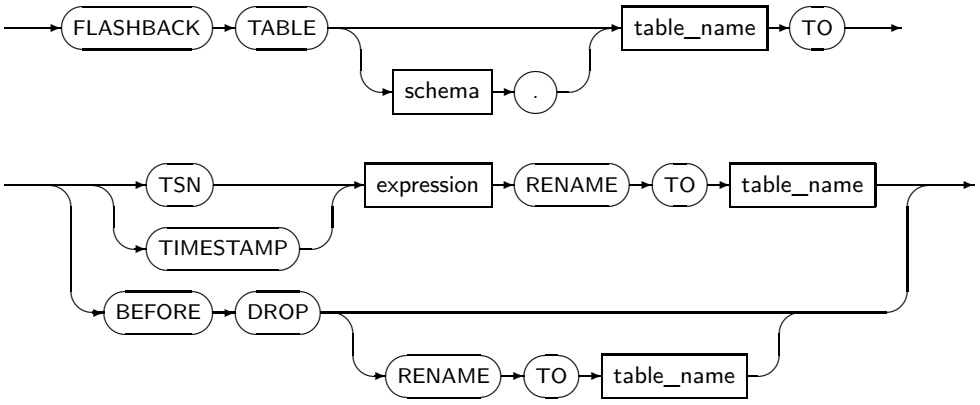
RECYCLEBIN 기능을 사용할 때 테이블을 제거하면 시스템이 테이블 및 인덱스, 트리거, 제약조건의 이름을 생성하여 변경하기만 하고, 실제로는 제거하지 않는다. 단, 제거할 테이블과 관련된 외래 키 제약조건은 제거된다.

RECYCLEBIN 기능을 사용하면 테이블을 제거할 때 RECYCLEBIN 뷰에 포함된 테이블을 복원할 수 있다. 테이블을 복원하면 테이블의 이름은 이전 상태로 복구되거나 지정한 대로 변경되지만, 인덱스, 트리거, 제약조건의 이름은 원래의 이름으로 복구되지 않는다.

FLASHBACK TABLE의 세부 내용은 다음과 같다.

- 문법

flashback_table



- 특권

  FLASHBACK_ANY_TABLE 특권이 있어야 한다.

- 구성요소

구성요소	설명
schema	복원할 테이블을 포함하고 있는 스키마의 이름이다. 생략하면 현재 사용자의 스키마로 인식한다.
table_name	복원할 테이블의 이름이다. 테이블을 제거하기 전의 이름 또는 RECYCLEBIN 뷰를 참조하여 제거한 후 시스템이 변경한 이름으로 지정할 수 있다. 단, 같은 이름의 테이블이 RECYCLEBIN 뷰에 여러 개가 존재하는 경우 원래 이름을 지정하면 가장 최근의 이름으로 사용된다.
TSN	특정 TSN으로 복원할 때 사용한다.
TIMESTAMP	특정 TIMESTAMP로 복원할 때 사용한다.
expression	TSN과 TIMESTAMP 구문으로 복원할 때 필요한 구문이다. TSN의 경우 특정 시점의 TSN이며, TIMESTAMP의 경우 특정 시간을 의미하는 구문이다.
BEFORE DROP	제거된 테이블을 복원한다. RENAME TO table_name 문을 사용하지 않을 때는 생략할 수 있다.
RENAME TO table_name	복원할 테이블의 이름을 원래 이름과 다르게 변경할 때 사용한다. 특히 이전에 사용하던 이름을 다른 테이블이 사용하고 있는 경우에 필요하다.

- 예제

  다음은 **FLASHBACK TABLE**을 사용해 테이블을 복원하는 예이다.

```
SQL> ALTER SYSTEM SET USE_RECYCLEBIN=Y;
System altered.

SQL> CREATE TABLE t (a NUMBER);
Table 'T' created.

SQL> INSERT INTO t values (1);
1 row inserted.

SQL> COMMIT;
Commit completed.

SQL> SELECT * FROM t;
          A
-----------
          1
1 row selected.

SQL> SELECT CURRENT_TSN FROM V$DATABASE;
CURRENT_TSN
-----------
      68887
1 row selected.

SQL> EXEC DBMS_LOCK.SLEEP(30);
PSM completed.

SQL> INSERT INTO t values (2);
1 row inserted.

SQL> COMMIT;
Commit completed.

SQL> SELECT * FROM t;
          A
-----------
          1
          2
2 rows selected.

SQL> FLASHBACK TABLE t TO SCN 68887 RENAME TO t_scn;
Flashbacked.

SQL> SELECT * FROM t_scn;
          A
-----------
          1
```

```
1 row selected.

SQL> FLASHBACK TABLE t TO TIMESTAMP (SYSTIMESTAMP - INTERVAL '1' MINUTE)
   2 RENAME TO t_timestamp;
Flashbacked.

SQL> SELECT * FROM t_timestamp;
          A
-----------
          1
1 row selected.

SQL> DROP TABLE t;
Table 'T' dropped.

SQL> SELECT * FROM t;  -- 테이블이 제거된다.
TBR-8033: Specified schema object was not found.
at line 1, column 16:
SELECT * FROM t

SQL> FLASHBACK TABLE t TO BEFORE DROP RENAME TO t2;
Flashbacked.

SQL> SELECT * FROM t2;

         A
----------
         1

1 row selected.
```

7.71. GRANT

시스템 특권과 역할, 스키마 객체 특권을 일반 사용자나 역할, 공유 사용자에게 부여한다.

참고

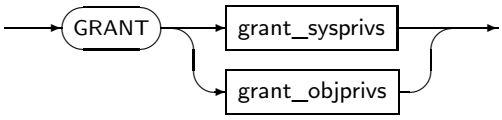
1. 부여한 특권 또는 역할을 회수하기 위해서는 “7.75. REVOKE”의 내용을 참고한다.

2. 역할을 생성하거나 정보를 변경, 제거하기 위해서는 “7.38. CREATE ROLE”, “7.11. ALTER ROLE”, “7.59. DROP ROLE”의 내용을 참고한다.

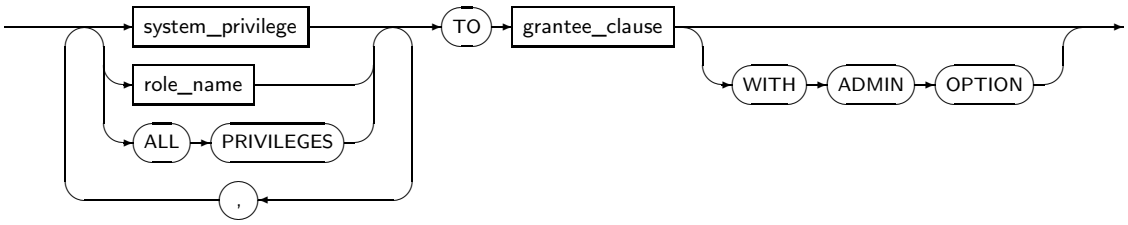
GRANT의 세부 내용은 다음과 같다.

- 문법

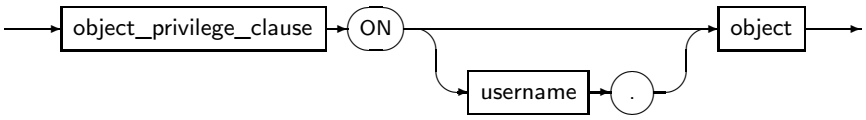
grant

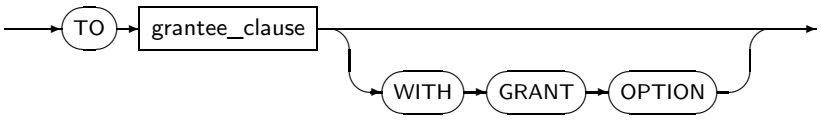


grant_sysprivs

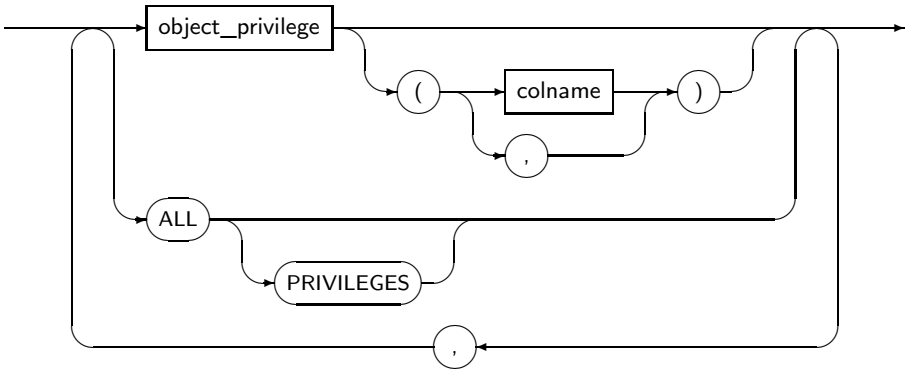


grant_objprivs

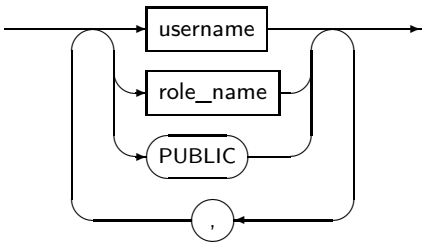




object_privilege_clause



grantee_clause



- 특권

  GRANT를 이용하여 특권 또는 권한을 부여하는 데에는 해당 특권이나 권한에 따라 다른 특권이 필요하다.

구성요소	설명
시스템 특권	사용자 자신이 사용할 수 있게 부여받은 시스템 특권에 한해 가능하다. 단, 자신이 사용할 수 있는 시스템 특권을 모두 부여할 수 있는 것은 아니다. 사용자 자신이 사용 가능하게 부여받을 때 WITH ADMIN OPTION을 사용하여 부여 받았어야 한다. GRANT ANY PRIVILEGE 시스템 특권을 가지고 있는 경우라면 자신이 부여받지 않은 시스템 특권도 모두 부여할 수 있다.
역할	시스템 특권과 마찬가지로 WITH ADMIN OPTION을 사용하여 부여받은 역할이거나 자신이 만든 역할이어야 부여가 가능하다. GRANT ANY ROLE 시스템 특권을 가지고 있는 경우라면 자신이 부여받지 않은 역할이라도 모두 부여할 수 있다.
스키마 객체 특권	스키마 객체 특권을 부여하는 것은 자신이 소유한 객체이거나 WITH GRANT OPTION을 사용하여 부여받은 스키마 객체 특권에 한해 가능하다. GRANT ANY OBJECT PRIVILEGE 시스템 특권을 부여 받았을 경우라면 자신이 부여받지 않은 스키마 객체 특권도 모두 부여할 수 있다.

- 구성요소
  - grant

구성요소	설명
grant_sysprivs	시스템 특권이나 역할을 부여한다.
grant_objprivs	스키마 객체 특권을 부여한다.

  - grant_sysprivs

구성요소	설명
system_privilege	시스템 특권을 정의한다. 자세한 내용은 "시스템 특권"을 참고한다.
role_name	역할을 정의한다. 역할에 대해서는 “7.38. CREATE ROLE”, “7.11. ALTER ROLE”, “7.59. DROP ROLE”을 참고한다.
ALL PRIVILEGES	현재 사용가능한 모든 시스템 특권을 부여한다. GRANT ANY PRIVILEGE 시스템 특권을 가지고 있는 사용자만이 ALL PRIVILEGE 절을 사용할 수 있다. 부여할 수 있는 시스템 특권은 아래의 "시스템 특권"을 참고한다.
TO grantee_clause	특권이나 역할을 부여받을 대상이 된다. 대상이 될 수 있는 것은 일반 사용자나 역할, 공유 사용자 3종류이다. 공유 사용자에게 특권 또는 역할을 부여하면, 모든 사용자에게 부여한 것과 동일한 효과를 갖게 되므로 주의해야 한다.

구성요소	설명
WITH ADMIN OPTION	시스템 특권과 역할은 해당 특권 또는 역할을 이용할 수 있는 사용 특권과 해당 특권 또는 역할을 남에게 부여할 수 있는 관리 특권으로 나눌 수 있다. WITH ADMIN OPTION을 사용하여 시스템 특권 또는 역할을 부여할 경우에는, 해당 특권 또는 역할을 사용할 수 있는 사용 특권과 함께 남에게 부여할 수 있는 관리 특권까지 같이 부여하게 된다. WITH ADMIN OPTION을 사용하지 않고 시스템 특권 또는 역할을 부여했을 경우에는, 부여받은 사용자는 부여받은 특권 또는 역할을 사용할 수만 있고, 다른 사용자 또는 역할에 부여할 수는 없다. 반대로 GRANT ANY PRIVILEGE 시스템 특권을 부여받은 사용자의 경우 자신이 부여받지 않은 시스템 특권도 마음대로 부여할 수 있다. 역할의 경우엔 GRANT ANY ROLE 시스템 특권이 같은 역할을 하게 된다. 주의해야 할 점은 WITH ADMIN OPTION을 사용하여 해당 특권 또는 역할을 남에게 부여할 수 있는 관리 특권을 주었을 경우 사용 특권은 그대로 둔 상태로 관리 특권만을 회수할 수 없다. 따라서 그 경우에는 부여한 시스템 특권 또는 역할 전체를 회수한 뒤 WITH ADMIN OPTION 없이 다시 그 시스템 특권 또는 역할을 할당해야 한다.

– object_objprivs

구성요소	설명
object_privilege_clause	스키마 객체 특권을 명시한다.
(username.) object	스키마 객체 특권의 대상이 되는 객체를 명시한다. username 부분이 생략되었을 경우 자신의 스키마 객체에서 해당 이름을 가진 객체를 찾게 된다. 만약 대상으로 동의어가 포함된 경우에는 해당 동의어의 기반 객체로 인식하고 처리하게 된다. 즉 동의어에 스키마 객체 특권을 부여하거나 회수하는 것은 동의어의 기반 객체에 특권을 부여하거나 회수하는 것과 동일한 효과를 갖게 된다.
TO grantee_clause	스키마 객체 특권을 부여받을 대상이 된다. 대상이 될 수 있는 것은 일반 사용자나 역할, 공유 사용자 3 종류이다.
WITH GRANT OPTION	시스템 특권의 WITH ADMIN OPTION과 마찬가지 역할을 한다. 즉, 부여한 스키마 객체 특권을 다른 사용자에게 부여할 수 있는 특권까지 같이 부여하고자 할 때 사용된다. 하지만, 시스템 특권의 WITH ADMIN OPTION과는 커다란 차이가 있다.

구성요소	설명
	자신의 스키마 객체가 아닌 다른 사용자의 스키마 객체를 WITH GRANT OPTION을 이용하여 스키마 객체 특권을 부여받은 사용자 A가, 또 다른 사용자 B에게 스키마 객체 특권을 부여한 경우 사용자 A의 스키마 객체 특권을 회수하면, 사용자 A가 부여한 사용자 B의 특권도 같이 회수된다. 또한, 시스템 특권에서의 역할과 마찬가지로, 자신의 스키마 객체에 대해서는 별도의 스키마 객체 특권을 부여받지 않아도, 모든 스키마 객체 특권과 해당 스키마 객체 특권을 다른 사용자에게 부여할 수 있는 특권까지 갖고 있다. GRANT ANY OBJECT PRIVILEGE 시스템 특권을 부여받은 사용자의 경우 자신이 부여받지 않은 객체에 스키마 객체 특권도 모두 부여할 수 있다. 이는 시스템 특권에서의 GRANT ANY PRIVILEGE와 동일한 역할을 한다고 할 수 있다. 또한, 부여받는 객체가 일반 사용자나 공유 사용자가 아닌 역할일 경우에는 WITH GRANT OPTION은 사용할 수 없다.

– object_privilege_clause

구성요소	설명
object_privilege	스키마 객체 특권을 명시한다. 스키마 객체 특권의 종류는 아래의 "스키마 객체 특권"을 참고한다.
colname	스키마 객체 특권은 시스템 특권과는 달리 좀 더 세부적인 설정이 가능하다. 즉, 특정 객체 전체에 대한 특권이 아닌 객체의 일부 컬럼에 대해서만 제약적으로 특권을 부여할 수도 있다. 객체의 특정 컬럼에 대해서만 제약적으로 스키마 객체 특권을 부여하고자 할 경우에는 특정 컬럼을 나열하는 방식으로 지정할 수 있다. 하지만, 모든 스키마 객체 특권이 컬럼별 특권을 지정할 수 있는 것은 아니다. 컬럼별 특권을 지정할 수 있는 스키마 객체 특권은 INSERT, REFERENCES, UPDATE뿐이다.
ALL PRIVILEGES	현재 사용가능한 모든 스키마 객체 특권을 부여한다. GRANT ANY PRIVILEGE 시스템 특권을 가지고 있는 사용자만이 ALL PRIVILEGE 절을 사용할 수 있다.

– grantee_clause

구성요소	설명
user_name	스키마 객체 특권을 부여받을 일반 사용자를 명시한다.
role_name	스키마 객체 특권을 부여받을 역할을 명시한다.

구성요소	설명
PUBLIC	스키마 객체 특권을 부여받을 공유 사용자를 명시한다. 공유 사용자에게 특권 또는 역할을 부여하면 모든 사용자에게 부여한 것과 동일한 효과를 갖게 되므로 주의해야 한다.

- 예제

  – grant_sysprivs

  다음은 사용자를 생성하고, **system_privilege**에 특권을 명시해 그 사용자에게 CREATE SESSION 시스템 특권을 부여하는 예이다.

```
SQL> CONN sys/tibero
Connected.

SQL> CREATE USER u1 IDENTIFIED BY 'a';
User created.

SQL> SELECT grantee, privilege FROM dba_sys_privs
     WHERE grantee='U1';

GRANTEE                    PRIVILEGE
-------------------------- ------------------------------------

0 row selected.

SQL> GRANT CREATE SESSION TO u1;
Granted.

SQL> SELECT grantee, privilege FROM dba_sys_privs
     WHERE grantee='U1';

GRANTEE                    PRIVILEGE
-------------------------- ------------------------------------
U1                         CREATE SESSION

1 row selected.
```

  처음 사용자를 생성하면, 첫 SELECT 문의 결과에서처럼 아무런 시스템 특권을 가지지 않은 상태가 된다. 이렇게 아무런 특권을 갖고 있지 않은 사용자가 데이터베이스에 접속해 어떤 작업을 하기 위해서는, 기본적으로 CREATE SESSION 시스템 특권이 필요하다. 위의 예에서는 생성된 사용자에게 CREATE SESSION 시스템 특권을 할당하고, 그 결과를 보여주고 있다.

  다음은 생성된 사용자 U1에게 **role_name**에 역할을 명시해 RESOURCE 역할을 부여하는 예이다.

```
SQL> SELECT grantee, granted_role FROM dba_role_privs
     WHERE grantee='U1';

GRANTEE                        GRANTED_ROLE
------------------------------ ------------------------------

0 row selected.

SQL> SELECT * FROM dba_roles;

ROLE                           PAS
------------------------------ ---
DBA                            NO
CONNECT                        NO
RESOURCE                       NO

3 rows selected.

SQL> GRANT RESOURCE TO u1;
Granted.

SQL> SELECT grantee, granted_role FROM dba_role_privs
     WHERE grantee='U1';

GRANTEE                        GRANTED_ROLE
------------------------------ ------------------------------
U1                             RESOURCE

1 row selected.
```

RESOURCE 역할은 기본 역할로, 데이터베이스가 생성될 때 자동으로 생성된다.

다음은 자신의 스키마 객체를 생성하기 위한 기본적인 시스템 특권을 보여준다.

```
SQL> SELECT grantee, privilege FROM dba_sys_privs
     WHERE grantee='RESOURCE';

GRANTEE                    PRIVILEGE
-------------------------- -----------------------------------
RESOURCE                   CREATE TABLE
RESOURCE                   CREATE SEQUENCE
RESOURCE                   CREATE PROCEDURE
RESOURCE                   CREATE TRIGGER

4 rows selected.
```

다음은 **ALL PRIVILEGE**를 사용하여 모든 시스템 특권을 부여하는 예이다.

```
SQL> GRANT ALL PRIVILEGES TO u1;
Granted.

SQL> SELECT grantee, privilege FROM dba_sys_privs
     WHERE grantee='u1';

GRANTEE                    PRIVILEGE
-------------------------- -----------------------------------
U1                         ALTER SYSTEM
U1                         CREATE SESSION
U1                         ALTER SESSION
U1                         CREATE TABLESPACE
U1                         ALTER TABLESPACE
U1                         DROP TABLESPACE
U1                         CREATE USER
U1                         ALTER USER
...
```

다음은 역할을 생성하고, 그 역할에 CREATE SESSION 시스템 특권을 부여하는 예이다.

```
SQL> CREATE ROLE aaa;
Role created.

SQL> GRANT CREATE SESSION TO aaa;
Granted.

SQL> SELECT grantee, privilege FROM dba_sys_privs
     WHERE grantee='AAA';

GRANTEE                    PRIVILEGE
-------------------------- -----------------------------------
AAA                        CREATE SESSION

1 row selected.
```

위의 예는 aaa라는 역할을 생성하고, 그 역할에 CREATE SESSION 시스템 특권을 부여하고 있다. 위의 예와 같이 역할에 특권을 부여하는 방법은 사용자에게 특권을 부여하는 방법과 동일하다.

역할에 특권을 부여하는 것처럼 역할을 다른 역할에 부여할 수도 있다. 하지만, 다음과 같이 여러 역할 서로가 서로를 부여받는 일은 허용하지 않는다.

```
SQL> CREATE ROLE aaa;
Role created.
```

```
SQL> CREATE ROLE bbb;
Role created.

SQL> GRANT aaa TO bbb;
Granted.

SQL> GRANT bbb TO aaa;
TBR-7173: circular role grant detected
```

다음은 **WITH ADMIN OPTION**을 사용했을 때와 사용하지 않았을 때의 차이에 관한 예이다.

```
SQL> CREATE USER u2 IDENTIFIED BY xxx;
User created.

SQL> CREATE USER u3 IDENTIFIED BY zzz;
User created.

SQL> GRANT CREATE SESSION TO u2;
Granted.

SQL> GRANT CREATE TABLE TO u2 WITH ADMIN OPTION;
Granted.

SQL> SELECT grantee, privilege, admin_option
     FROM dba_sys_privs WHERE grantee='u2';

GRANTEE                PRIVILEGE                       ADM
---------------------- ------------------------------- ---
U2                     CREATE SESSION                  NO
U2                     CREATE TABLE                    YES

2 rows selected.

SQL> CONN u2/xxx
Connected.

SQL> GRANT CREATE SESSION TO u3;
TBR-17004: get authorization for this action first.

SQL> GRANT CREATE TABLE TO u3;
Granted.
```

위의 예에서는 사용자 u2, u3를 생성한 뒤 u2 사용자에게 CREATE SESSION 시스템 특권을 부여했는데, 처음에는 특권만 부여하고, 그 다음에는 WITH ADMIN OPTION을 사용하여 부여하였다.

그런 뒤 사용자 u2로 접속을 해 보면, WITH ADMIN OPTION 을 이용하여 부여받은 CREATE TABLE 시스템 특권은 사용자 u3에게 부여할 수 있지만, 그렇지 않은 CREATE SESSION 시스템 특권은 부여할 수 없다.

- grant_objprivs

  다음은 **object_privilege**에 스키마 객체 특권을 명시해 사용자에게 스키마 객체 특권을 부여하는 예이다.

```
SQL> CREATE USER u5 IDENTIFIED BY xxx;
User created.

SQL> GRANT CONNECT,RESOURCE TO u5;
Granted.

SQL> CREATE USER u6 IDENTIFIED BY zzz;
User created.

SQL> GRANT CONNECT TO u6;
Granted.

SQL> CONN u5/xxx
Connected.

SQL> CREATE TABLE t1 (a NUMBER, b NUMBER, c NUMBER);
Table created.

SQL> INSERT INTO t1 VALUES (1, 2, 3);
1 row inserted.

SQL> SELECT * FROM t1;
         A          B          C
---------- ---------- ----------
         1          2          3

1 row selected.

SQL> GRANT SELECT ON t1 TO u6;
Granted.

SQL> CONN u6/zzz
Connected.

SQL> SELECT * FROM u5.t1;
         A          B          C
---------- ---------- ----------
         1          2          3
```

```
1 row selected.
```

위의 예에서는 우선 사용자 u5, u6를 생성한 뒤 사용자 u5에게는 데이터베이스에 접속해서 스키마 객체를 만들 수 있는 특권을, 사용자 u6에게는 데이터베이스에 접속할 수 있는 특권을 부여한다.

사용자 u5로 접속하여 CREATE TABLE 문을 사용하여 테이블을 생성한 뒤 간단한 데이터를 삽입하고, 해당 스키마 객체에 질의 특권을 사용자 u6에게 부여하여 사용자 u6이 u5의 테이블에 대한 질의문을 실행할 수 있음을 볼 수 있다.

만약 사용자 u5가 사용자 u6에게 특권을 부여하지 않은 경우라면 다음과 같은 에러가 발생할 것이다.

```
SQL> SELECT * FROM u5.t1;
TBR-8033: schema object does not exist
at line 1, column 16:
SELECT * FROM u5.t1;
```

다음은 컬럼별로 특권을 지정하는 예이다.

```
SQL> CONN u5/xxx;
Connected.

SQL> GRANT INSERT (a, b), UPDATE (a, b) ON t1 TO u6;
Granted.

SQL> CONN u6/zzz
Connected.

SQL> INSERT INTO u5.t1 VALUES (100, 200, 300);
TBR-8053: not enough privilege

SQL> INSERT INTO u5.t1(a, b) VALUES (100, 200);
1 row inserted.

SQL> SELECT * FROM u5.t1;
         A          B          C
---------- ---------- ----------
       100        200
         1          2          3

2 rows selected.

SQL> UPDATE u5.t1 SET c=100 WHERE a=100;
TBR-8053: not enough privilege

SQL> UPDATE u5.t1 SET a=11 WHERE a=100;
1 row updated.
```

```
SQL> SELECT * FROM u5.t1;
         A          B          C
---------- ---------- ----------
        11        200
         1          2          3

2 rows selected.
```

위의 예에서는 사용자 u5가 자신의 테이블 t1의 컬럼 A, B에 대해서 삽입과 갱신 특권을 사용자 u6에게 부여한 것이다. 사용자 u6은 사용자 u5의 테이블 t1중 컬럼 A와 B에 대해서만 삽입과 갱신을 할 수 있게 된다. 그 외의 컬럼 (위의 예에서는 C 컬럼) 에 대해 삽입 또는 갱신할 경우 **TBR-8053: not enough privilege**라는 에러가 발생하게 된다.

다음은 **GRANT ALL**을 사용하여 사용자 자신이 부여할 수 있는 모든 스키마 객체 특권을 부여하는 예이다.

```
SQL> CONN u5/xxx
Connected.

SQL> CREATE TABLE t2 (c1 NUMBER, c2 NUMBER);
Table created.

SQL> GRANT ALL ON t2 TO u6;
Granted.

SQL> CONN u6/zzz
Connected.

SQL> SELECT owner, table_name, privilege
     FROM user_tbl_privs WHERE table_name='T2';

OWNER            TABLE_NAME       PRIVILEGE
---------------- ---------------- ----------------------
U5               T2               ALTER
U5               T2               DELETE
U5               T2               INDEX
U5               T2               INSERT
U5               T2               SELECT
U5               T2               UPDATE
U5               T2               REFERENCES

7 rows selected.
```

다음은 동의어에 특권을 할당하는 예이다.

```
SQL> CONN sys/tibero
Connected.

SQL> GRANT CREATE SYNONYM TO u5;
Granted.

SQL> CONN u5/xxx
Connected.

SQL> CREATE TABLE t3 (c1 NUMBER, c2 NUMBER);
Table created.

SQL> CREATE SYNONYM s3 FOR t3;
Synonym created.

SQL> GRANT SELECT, INSERT ON s3 TO u6;
Granted.

SQL> CONN u6/zzz
Connected.

SQL> INSERT INTO u5.t3 VALUES (1, 2);
1 row inserted.

SQL> SELECT * FROM u5.s3;
        C1         C2
---------- ----------
         1          2

1 row selected.

SQL> SELECT owner, table_name, privilege
     FROM user_tbl_privs WHERE table_name='t3';

OWNER            TABLE_NAME       PRIVILEGE
---------------- ---------------- --------------------------
U5               T3               INSERT
U5               T3               SELECT

2 rows selected.

SQL> SELECT owner, table_name, privilege
     FROM user_tbl_privs WHERE table_name='s3';

OWNER            TABLE_NAME       PRIVILEGE
---------------- ---------------- --------------------------
```

```
0 row selected.
```

위의 예에서는 사용자 u5가 새로운 테이블 t3를 생성한 뒤 테이블 t3의 동의어 s3를 생성한다. 그런 뒤 동의어 s3에 대해 질의 특권과 로우를 삽입하는 특권을 사용자 u6에게 부여한다. 그러면 사용자 u6에게는 동의어가 가리키는 테이블 t3에 대한 특권이 부여되어, 직접 t3이라는 이름으로 질의 또는 로우를 삽입할 수 있게 된다.

- grantee_clause

다음은 공유 사용자에게 스키마 객체 특권을 부여하는 예이다.

```
SQL> CONN sys/tibero
Connected.

SQL> CREATE PUBLIC SYNONYM s1 FOR u5.t1;
Synonym created.

SQL> GRANT SELECT ON s1 TO PUBLIC;
Granted.

SQL> SELECT owner, table_name, privilege
     FROM dba_tbl_privs
     WHERE grantee='PUBLIC' AND owner='U5';

OWNER            TABLE_NAME       PRIVILEGE
---------------- ---------------- --------------------------
U5               T1               SELECT

1 row selected.

SQL> CREATE USER u7 IDENTIFIED BY aaa;
User created.

SQL> GRANT CONNECT TO u7;
Granted.

SQL> CONN u7/aaa
Connected.

SQL> SELECT * FROM s1;
         A          B          C
---------- ---------- ----------
        11        200
         1          2          3

2 rows selected.
```

위의 예에서 보여주는 것과 같이 공유 사용자에게 스키마 객체 특권을 부여하는 경우 주로 CREATE SYNONYM 문을 사용하여 공유 동의어를 생성하고, 그 공유 동의어에 특권을 주고자 할 때이다. 그 이외의 경우 함부로 사용하면 보안에 치명적인 문제가 생길 수 있다.

다음은 **WITH GRANT OPTION**을 사용하는 예이다.

```
SQL> CONN u5/xxx
Connected.

SQL> CREATE TABLE t4 (c1 NUMBER, c2 NUMBER);
Table created.

SQL> GRANT SELECT ON t4 TO u6 WITH GRANT OPTION;
Granted.

SQL> GRANT INSERT ON t4 TO u6;
Granted.

SQL> CONN u6/zzz
Connected.

SQL> SELECT owner, table_name, privilege, grantable
     FROM user_tbl_privs WHERE table_name='T4';

OWNER            TABLE_NAME       PRIVILEGE                GRA
---------------- ---------------- ------------------------ ---
U5               T4               INSERT                   NO
U5               T4               SELECT                   YES

2 rows selected.

SQL> INSERT INTO u5.t4 VALUES(1, 2);
1 row inserted.

SQL> SELECT * FROM u5.t4;
        C1         C2
---------- ----------
         1          2

1 row selected.

SQL> GRANT INSERT ON u5.t4 TO u7;
TBR-17004: get authorization for this action first.

SQL> GRANT SELECT ON u5.t4 TO u7;
Granted.
```

```
SQL> CONN U7/aaa;
Connected.

SQL> SELECT * FROM u5.t4;
        C1         C2
---------- ----------
         1          2

1 row selected.
```

위의 예를 보면, WITH GRANT OPTION의 용도를 알 수 있다. 시스템 특권의 WITH ADMIN OPTION과 마찬가지로 사용자 u5가 사용자 u6에게 WITH GRANT OPTION을 이용하여 부여한 질의 특권은 사용자 u6이 사용자 u7에게 다시 부여할 수 있다. 하지만, WITH GRANT OPTION 없이 부여받은 로우 삽입 특권은 사용자 u6만 사용할 수 있고, 또 다른 사용자인 u7에게는 부여할 수 없다.

다음은 **WITH ADMIN OPTION**과 **WITH GRANT OPTION**의 차이점에 관한 예이다.

```
SQL> CONN u5/xxx
Connected.

SQL> REVOKE ALL ON u4 FROM u6;
Revoked.

SQL> CONN u7/aaa
Connected.

SQL> SELECT * FROM u5.t4;
TBR-8053: not enough privilege
at line 1, column 16:
SELECT * FROM u5.t4;
```

WITH GRANT OPTION은 특권을 회수하면 특권을 받은 사용자가 부여한 또 다른 특권도 모두 같이 회수된다. 즉, 위의 예에서 사용자 u6에게 부여한 테이블 t4에 대한 스키마 객체 특권을 모두 회수하면 사용자 u6이 사용자 u7에게 부여한 스키마 객체 특권도 같이 회수가 되어 사용자 u7도 더 이상 사용자 u5의 테이블 t4에 대해 질의할 수 없게 된다.

시스템 특권

다음은 시스템 특권을 정리한 표이다.

구분	시스템 특권	설명
시스템	ALTER SYSTEM	ALTER SYSTEM 명령을 실행할 수 있다.

구분	시스템 특권	설명
	SYSDBA	SHUTDOWN, ALTER DATABASE, CREATE DATABASE, ARCHIVELOG, RECOVERY 명령을 실행할 수 있다.
세션	CREATE SESSION	데이터베이스에 세션을 생성할 수 있다. 즉, 로그인을 할 수 있다.
	ALTER SESSION	ALTER SESSION 명령을 실행할 수 있다.
테이블 스페이스	CREATE TABLESPACE	테이블 스페이스를 생성할 수 있다.
	ALTER TABLESPACE	테이블 스페이스를 변경할 수 있다.
	DROP TABLESPACE	테이블 스페이스를 제거할 수 있다.
사용자	CREATE USER	사용자를 생성할 수 있다.
	ALTER USER	사용자 정보를 변경할 수 있다.
	DROP USER	사용자를 제거할 수 있다.
테이블	CREATE TABLE	자신의 스키마에 테이블을 생성할 수 있다.
	CREATE ANY TABLE	임의의 스키마에 테이블을 생성할 수 있다.
	ALTER ANY TABLE	임의의 스키마에 있는 테이블을 변경할 수 있다.
	DROP ANY TABLE	임의의 스키마에 있는 테이블을 제거할 수 있다.
	COMMENT ANY TABLE	임의의 스키마에 있는 테이블에 주석을 추가할 수 있다.
	SELECT ANY TABLE	임의의 스키마에 속한 테이블, 동의어 테이블, 뷰, 실체화 뷰를 조회할 수 있다.
	INSERT ANY TABLE	임의의 스키마에 속한 테이블 또는 동의어 테이블에 로우를 삽입할 수 있다.
	UPDATE ANY TABLE	임의의 스키마에 속한 테이블 또는 동의어 테이블의 로우를 갱신할 수 있다.
	DELETE ANY TABLE	임의의 스키마에 있는 테이블의 로우를 제거할 수 있다.
	TRUNCATE ANY TABLE	임의의 스키마에 있는 테이블에 TRUNCATE를 수행할 수 있다. 이 권한을 사용하기 위해서는 USE_TRUNCATE_PRIVILEGE 파라미터를 Y로 설정해야 한다.
인덱스	CREATE ANY INDEX	임의의 스키마에 속한 테이블 또는 실체화 뷰의 인덱스를 생성할 수 있다.
	ALTER ANY INDEX	임의의 스키마에 있는 테이블의 인덱스를 변경할 수 있다.

구분	시스템 특권	설명
	DROP ANY INDEX	임의의 스키마에 있는 테이블의 인덱스를 제거할 수 있다.
동의어	CREATE SYNONYM	자신의 스키마에 동의어를 생성할 수 있다.
	CREATE ANY SYNONYM	임의의 스키마에 동의어를 생성할 수 있다.
	DROP ANY SYNONYM	임의의 스키마에 있는 동의어를 제거할 수 있다.
	CREATE PUBLIC SYNONYM	공유 스키마에 공용 동의어를 생성할 수 있다.
	DROP PUBLIC SYNONYM	공유 스키마에 공용 동의어를 제거할 수 있다.
뷰	CREATE VIEW	자신의 스키마에 뷰를 생성할 수 있다.
	CREATE ANY VIEW	임의의 스키마에 뷰를 생성할 수 있다.
	DROP ANY VIEW	임의의 스키마에 있는 뷰를 제거할 수 있다.
실체화 뷰	CREATE MATERIALIZED VIEW	자신의 스키마에 있는 실체화 뷰를 생성할 수 있다.
	CREATE ANY MATERIALIZED VIEW	임의의 스키마에 있는 실체화 뷰를 생성할 수 있다.
	ALTER ANY MATERIALIZED VIEW	임의의 스키마에 있는 실체화 뷰를 변경할 수 있다.
	DROP ANY MATERIALIZED VIEW	임의의 스키마에 있는 실체화 뷰를 제거할 수 있다.
시퀀스	CREATE SEQUENCE	자신의 스키마에 시퀀스를 생성할 수 있다.
	CREATE ANY SEQUENCE	임의의 스키마에 시퀀스를 생성할 수 있다.
	ALTER ANY SEQUENCE	임의의 스키마에 있는 시퀀스를 변경할 수 있다.
	DROP ANY SEQUENCE	임의의 스키마에 있는 시퀀스를 제거할 수 있다.
	SELECT ANY SEQUENCE	임의의 스키마에 속한 시퀀스에서 시퀀스 또는 동의어 시퀀스를 조회할 수 있다.
역할	CREATE ROLE	역할을 생성할 수 있다.
	DROP ANY ROLE	역할을 제거할 수 있다.
	GRANT ANY ROLE	임의의 역할을 부여할 수 있다.
	ALTER ANY ROLE	역할을 변경할 수 있다.
데이터베이스	ALTER DATABASE	데이터베이스를 변경할 수 있다.
프러시저	CREATE PROCEDURE	자신의 스키마에 PSM을 생성할 수 있다.
	CREATE ANY PROCEDURE	임의의 스키마에 PSM을 생성할 수 있다.
	ALTER ANY PROCEDURE	임의의 스키마에 속한 PSM을 변경할 수 있다.
	DROP ANY PROCEDURE	임의의 스키마에 속한 PSM을 제거할 수 있다.
	EXECUTE ANY PROCEDURE	임의의 스키마에 속한 PSM을 실행할 수 있다.

구분	시스템 특권	설명
데이터베이스 링크	CREATE DATABASE LINK	자신의 스키마에 데이터베이스 링크를 생성할 수 있다.
	CREATE PUBLIC DATABASE LINK	공유 스키마에 데이터베이스 링크를 생성할 수 있다.
	DROP PUBLIC DATABASE LINK	공유 스키마에 데이터베이스 링크를 제거할 수 있다.
디렉터리	CREATE ANY DIRECTORY	디렉터리 객체를 생성할 수 있다.
	DROP ANY DIRECTORY	디렉터리 객체를 제거할 수 있다.
감시	AUDIT SYSTEM	시스템 감시의 사용이 가능하다.
	AUDIT ANY	디렉터리 또는 임의의 스키마에 있는 객체의 감시가 가능하다.
라이브러리	CREATE LIBRARY	자신의 스키마에 라이브러리를 생성할 수 있다.
	CREATE ANY LIBRARY	임의의 스키마에 라이브러리를 생성할 수 있다.
	DROP ANY LIBRARY	임의의 스키마에 있는 라이브러리를 제거할 수 있다.
	EXECUTE ANY LIBRARY	임의의 스키마에 있는 라이브러리를 실행할 수 있다.
특권	GRANT ANY OBJECT PRIVILEGE	모든 스키마 객체 특권을 부여할 수 있다.
	GRANT ANY PRIVILEGE	모든 특권을 다 부여할 수 있다.

스키마 객체 특권

다음은 스키마 객체 특권을 정리한 표이다.

스키마 객체 특권	설명
SELECT	테이블, 뷰 또는 시퀀스에 질의문을 실행할 수 있다.
INSERT	테이블 또는 뷰에 로우를 삽입할 수 있다.
ALTER	테이블 또는 시퀀스의 정의를 변경할 수 있다.
UPDATE	테이블 또는 뷰의 로우를 갱신할 수 있다.
DELETE	테이블 또는 뷰로부터 로우를 삭제할 수 있다.
TRUNCATE	테이블에 TRUNCATE를 수행할 수 있다. 이 권한을 사용하기 위해서는 USE_TRUNCATE_PRIVILEGE 파라미터를 Y로 설정해야 한다.
INDEX	테이블에 인덱스를 생성할 수 있다.
EXECUTE	함수 또는 프러시저를 컴파일하고 실행할 수 있다.

스키마 객체 특권	설명
REFERENCES	테이블 또는 뷰에 참조 무결성 제약조건을 정의할 수 있다.
READ	디렉터리 객체에 읽기를 수행할 수 있다.
WRITE	디렉터리 객체에 쓰기를 수행할 수 있다.

스키마 객체 특권의 대상 객체

스키마 객체 특권을 정의할 수 있는 객체로는 테이블과 뷰, 시퀀스가 있다. 다음 표는 각 객체에 대해 어떤 스키마 객체 특권을 정의할 수 있는지를 나타낸 것이다.

스키마 객체 특권	테이블	뷰	시퀀스	PSM 프로그램 (프러시저, 함수 등)	디렉터리
SELECT	O	O	O		
INSERT	O	O			
ALTER	O		O		
UPDATE	O	O			
DELETE	O	O			
TRUNCATE	O				
EXECUTE				O	
INDEX	O				
REFERENCES	O	O			
READ					O
WRITE					O

7.72. NOAUDIT

감사하고 있는 시스템 특권 또는 스키마 객체 특권의 감사를 해제한다.

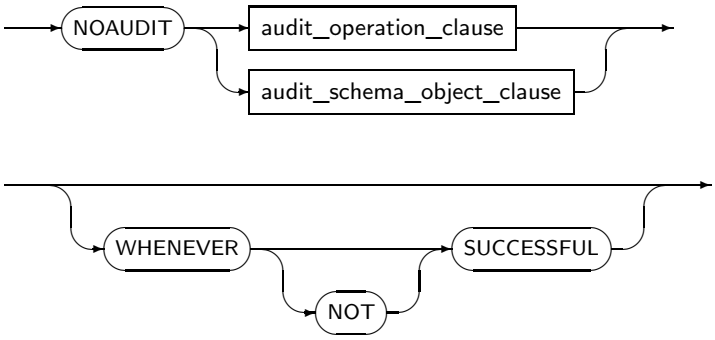
참고

특권을 감사하기 위해서는 “7.21. AUDIT”의 내용을 참고한다.

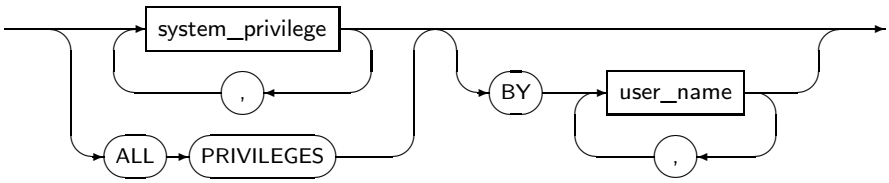
NOAUDIT의 세부 내용은 다음과 같다.

- 문법

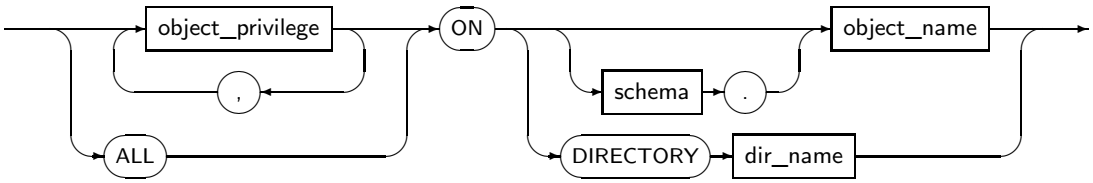
noaudit



audit_operation_clause



audit_schema_object_clause



- 특권
  - 시스템 특권의 감사를 해제하기 위해서는 AUDIT SYSTEM 시스템 특권을 부여받아야 한다.
  - 다른 사용자가 소유한 스키마의 객체 또는 디렉터리 객체의 감사를 해제하기 위해서는 AUDIT ANY 시스템 특권을 부여받아야 한다.
- 구성요소
  - noaudit

구성요소	설명
audit_operation_clause	시스템 특권의 감사를 해제한다.
audit_schema_object_clause	특정 객체에 대한 스키마 객체 특권의 감사를 해제한다.
WHENEVER SUCCESSFUL	명령의 성공, 실패에 따라 감사를 해제한다. 명령이 성공하였을 때 감사를 해제한다. 지정하지 않으면 성공, 실패 여부에 관계없이 감사를 해제한다.
WHENEVER NOT SUCCESSFUL	명령이 실패하였을 때 감사를 해제한다. 지정하지 않으면 성공, 실패 여부에 관계없이 감사를 해제한다.

– audit_operation_clause

구성요소	설명
system_privilege	감사를 해제할 시스템 특권을 지정한다. 시스템 특권의 종류는 “7.71. GRANT”의 "시스템 특권"을 참고한다.
ALL PRIVILEGES	모든 시스템 특권의 감사를 해제한다.
BY user_name	감사를 해제할 사용자를 지정한다. 지정하지 않으면 모든 사용자에 적용된다.

– audit_schema_object_clause

구성요소	설명
object_privilege	감사를 해제할 스키마 객체 특권을 지정한다. 스키마 객체 특권의 종류는 “7.71. GRANT”의 "스키마 객체 특권"을 참고한다.
ALL	해당 객체에 사용할 수 있는 모든 스키마 객체 특권의 감사를 해제한다. 대상 객체의 종류에 따라 사용할 수 있는 스키마 객체 특권은 “7.71. GRANT”의 "스키마 객체 특권의 대상 객체"를 참고한다.
ON	스키마 객체 특권의 감사를 해제할 대상이 되는 객체를 지정한다.
schema	스키마 객체 특권의 감사를 해제할 객체가 속해 있는 스키마를 명시한다. 생략하면 현재 사용자의 스키마로 인식한다.
object_name	디렉터리가 아닌 객체의 이름을 지정한다.
DIRECTORY dir_name	디렉터리 객체의 이름을 지정한다.

- 예제

– noaudit

다음은 **WHENEVER SUCCESSFUL**을 명시해 명령이 성공했을 때 감사를 해제하도록 설정하는 예이다.

```
SQL> NOAUDIT delete ON t WHENEVER SUCCESSFUL;
Noaudited.
```

– audit_operation_clause

다음은 **BY user_name**을 사용해 감사를 해제할 대상이 되는 사용자를 지정하는 예이다.

```
SQL> NOAUDIT create table BY tibero;
Noaudited.
```

– audit_schema_object_clause

다음은 **ON**을 사용해 감사를 해제할 대상이 되는 객체를 지정하는 예이다.

```
SQL> NOAUDIT insert ON t;
Noaudited.
```

7.73. PURGE

RECYCLEBIN 뷰에 포함된 스키마 객체를 완전히 제거한다. 완전히 제거된 스키마 객체는 FLASHBACK TABLE 문으로 복원할 수 있다.

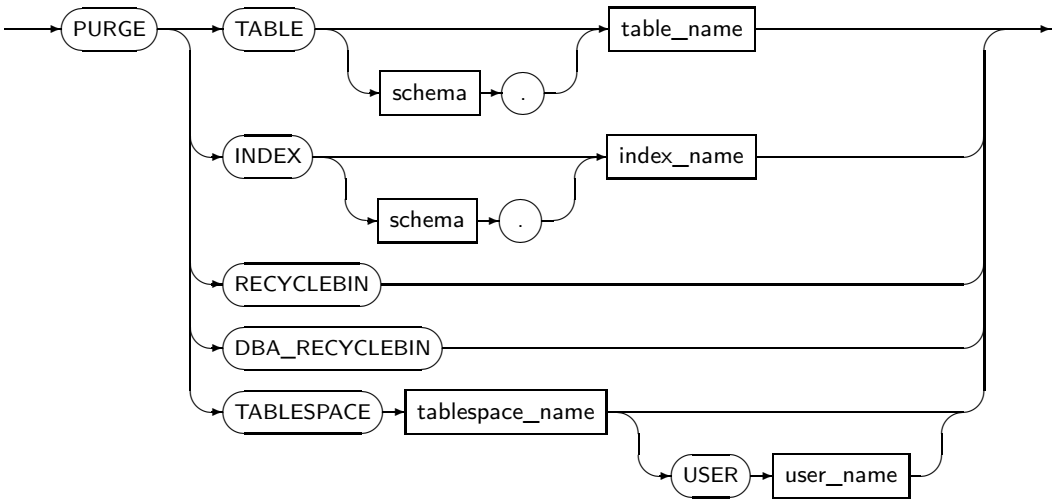
RECYCLEBIN 뷰에 포함된 스키마 객체는 다음의 질의를 통해 확인할 수 있다.

```
SELECT * FROM RECYCLEBIN;

SELECT * FROM USER_RECYCLEBIN;
```

PURGE의 세부 내용은 다음과 같다.

- 문법

purge



- 특권
  - PURGE 대상이 되는 테이블, 인덱스를 제거하기 위한 특권이 있어야 한다.
  - PURGE DBA_RECYCLEBIN 문을 실행하려면 SYSDBA 특권이 있어야 한다.

- 구성요소

구성요소	설명
TABLE table_name	특정 테이블을 완전히 제거할 때 사용한다.

구성요소	설명
	테이블을 제거하기 전의 이름 또는 RECYCLEBIN 뷰를 참조하여 제거한 후 시스템이 변경한 이름으로 지정할 수 있다. 단, 같은 이름의 테이블이 RECYCLEBIN 뷰에 여러 개가 존재하는 경우 원래 이름을 지정하면 가장 오래된 테이블이 제거된다.
INDEX index_name	특정 인덱스를 완전히 제거할 때 사용한다. 인덱스를 제거하기 전의 이름 또는 RECYCLEBIN 뷰를 참조하여 제거한 후 시스템이 변경한 이름으로 지정할 수 있다. 단, 같은 이름의 인덱스가 RECYCLEBIN 뷰에 여러 개가 존재하는 경우 원래 이름을 지정하면 가장 오래된 인덱스가 제거된다.
RECYCLEBIN	USER_RECYCLEBIN 뷰에 포함된 모든 객체를 완전히 제거한다.
DBA_RECYCLEBIN	DBA_RECYCLEBIN 뷰에 포함된 모든 객체를 완전히 제거한다.
TABLESPACE tablespace_name USER user_name	특정 테이블 스페이스에 존재하는 객체를 완전히 제거한다. USER 절을 지정하면 그 테이블 스페이스 안에서도 특정 사용자의 객체만 완전히 제거된다.

- 예제

  다음은 **PURGE**을 사용해 테이블을 완전히 제거하는 예이다.

```
SQL> ALTER SYSTEM SET USE_RECYCLEBIN=Y;
System altered.

SQL> CREATE TABLE t (a NUMBER);
Table 'T' created.

SQL> DROP TABLE t;
Table 'T' dropped.

SQL> PURGE TABLE t;
Purged.
```

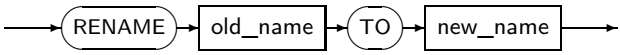
7.74. RENAME

테이블, 뷰, 동의어 등의 이름을 변경한다. 다른 스키마 객체의 이름은 변경할 수 없고 오직 사용자가 소유한 스키마 객체의 이름만 변경할 수 있다. 이름을 변경한 스키마 객체와 관련된 제약조건과 인덱스, 특권 등은 이름이 변경됨과 동시에 함께 전환된다. 하지만, 이름을 변경한 객체를 참조하는 뷰, 함수, 프러시저 등은 바로 무효화 상태가 된다.

RENAME의 세부 내용은 다음과 같다.

- 문법

rename



- 특권

특별한 권한이 필요하지 않다.

- 구성요소

구성요소	설명
old_name	변경할 스키마 객체의 원래 이름이다.
new_name	변경할 스키마 객체의 새로운 이름이다.

- 예제

다음은 **RENAME**을 사용해 사용자의 이름을 변경하는 예이다.

```
RENAME emp TO employee;
```

7.75. REVOKE

시스템 특권, 역할 또는 스키마 객체 특권을 일반 사용자, 역할 또는 공유 사용자에게서 회수한다.

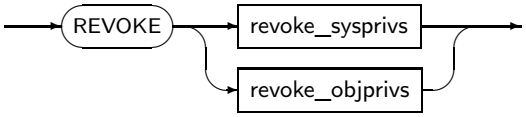
참고

1. 특권 또는 역할을 부여하기 위해서는 “7.71. GRANT”의 내용을 참고한다.
2. 역할을 생성, 변경, 제거하기 위해서는 “7.38. CREATE ROLE”, “7.11. ALTER ROLE”, “7.59. DROP ROLE”의 내용을 참고한다.

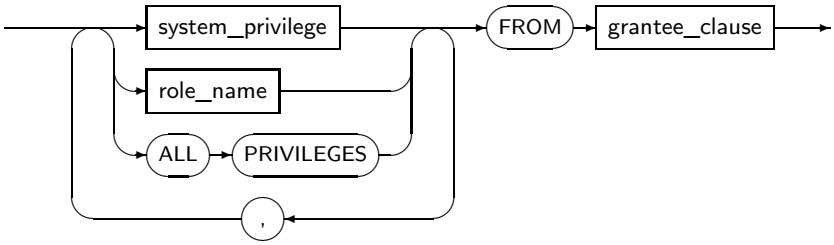
REVOKE의 세부 내용은 다음과 같다.

- 문법

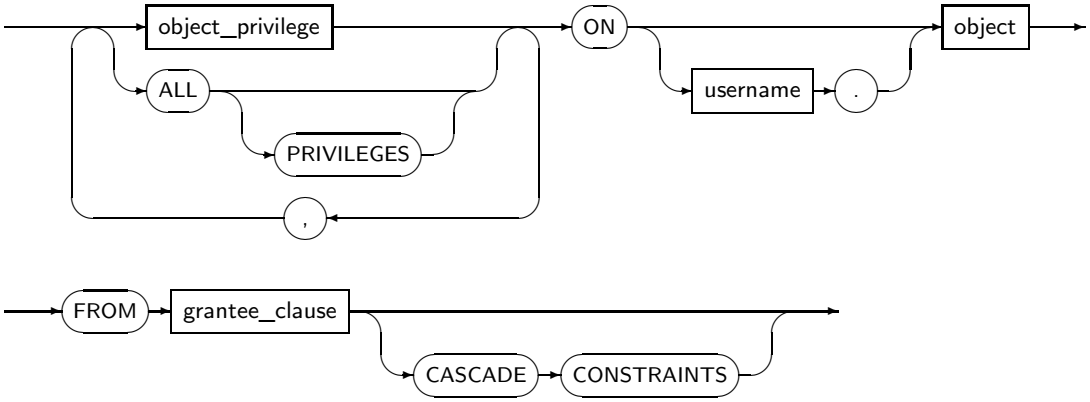
revoke



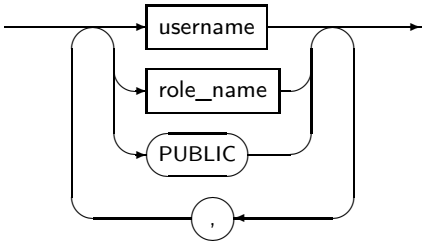
revoke_sysprivs



revoke_objprivs



grantee_clause



- 특권

REVOKE를 이용하여 특권 또는 권한을 회수하는 데에는 회수할 특권 또는 권한별로 다른 특권이 필요하다.

회수 대상	설명
시스템 특권	시스템 특권을 회수하기 위해서는, 해당 시스템 특권에 대해 자신이 사용할 수 있는 특권뿐만 아니라 관리할 수 있는 특권까지 부여를 받은 상태이어야 한다. GRANT ANY PRIVILEGE 시스템 특권을 가지고 있는 경우라면 자신이 부여받지 않은 시스템 특권도 모두 회수할 수 있다. 즉, GRANT를 사용하여 부여할 수 있는 것은 REVOKE를 이용하여 회수할 수도 있다.
역할	역할을 회수하기 위해서는 시스템 특권과 마찬가지로 WITH ADMIN OPTION을 통해 관리를 부여 받은 역할이거나 자신이 만든 역할이어야 한다.

회수 대상	설명
	GRANT ANY ROLE 시스템 특권을 가지고 있는 경우라면, 자신이 부여받지 않은 역할이라도 모두 회수할 수 있다. 즉, GRANT를 사용하여 부여할 수 있는 것은 REVOKE를 이용하여 회수할 수도 있다.
스키마 객체 특권	시스템 특권이나 역할과는 조금 다르다. 스키마 객체 특권은 자기 자신이 부여한 스키마 객체 특권만 회수할 수 있다. 자신의 스키마 객체에 대한 스키마 객체 특권이더라도 자신이 부여한 스키마 객체 특권이 아니면 회수할 수 없다. GRANT ANY OBJECT PRIVILEGE 시스템 특권을 부여받았을 경우라면, 자신이 부여하지 않은 스키마 객체 특권도 회수할 수 있다. 단, 스키마 객체 특권의 관리 특권을 가지지 않고, GRANT ANY OBJECT PRIVILEGE 시스템 특권을 사용하여 스키마 객체 특권을 부여한 경우에는, 해당 객체의 소유자가 해당 스키마 객체 특권을 회수할 수 있다.

REVOKE를 이용하여 사용자의 특권을 회수하지 못하는 경우는 다음과 같다.
- 사용자가 해당 특권 또는 역할을 부여받지 않은 경우
- 사용자가 해당 특권 또는 역할을, 다른 역할을 통해 부여받은 경우

- 구성요소
  - revoke

구성요소	설명
revoke_sysprivs	시스템 특권이나 역할을 회수한다.
grant_objprivs	사용자의 스키마 객체 특권을 회수한다.

  - revoke_sysprivs

구성요소	설명
system_privilege	시스템 특권을 정의한다. 시스템 특권의 종류는 “7.71. GRANT”의 "시스템 특권"을 참고한다.
role_name	역할을 정의한다. 역할에 대해서는 “7.38. CREATE ROLE”, “7.11. ALTER ROLE”, “7.59. DROP ROLE”을 참고한다.
ALL PRIVILEGES	현재 사용가능한 모든 시스템 특권을 회수한다. 즉, 현재 사용가능한 모든 시스템 특권을 가진 사용자에게서 모든 시스템 특권을 회수하고자 할 때에만 사용할 수 있다. 일부 시스템 특권을 가진 사용자에게 ALL PRIVILEGES 절을 사용할 경우 에러가 발생한다.

구성요소	설명
FROM grantee_clause	특권이나 역할을 회수할 대상이 된다. 대상이 될 수 있는 것은 일반 사용자나 역할, 공유 사용자 세 종류이다.

– grant_objprivs

구성요소	설명
object_privilege	스키마 객체 특권을 정의한다. 스키마 객체 특권의 종류는 “7.71. GRANT”의 "스키마 객체 특권"을 참고한다. GRANT와는 달리 객체의 컬럼별로 스키마 객체 특권을 부여한 것도 컬럼의 구분 없이 객체만 지정하여 회수할 수 있다.
ALL (PRIVILEGES)	해당 객체에 사용자 자신이 부여한 모든 스키마 객체 특권을 회수하고자 할 때 사용한다. PRIVILEGES 절은 생략 가능하다. 시스템 특권과는 달리 스키마 객체 특권에서는 모든 특권을 가지고 있지 않아도, 사용자 자신이 부여한 스키마 객체 특권을 찾아 회수한다. USE_TRUNCATE_PRIVILEGE 파라미터가 꺼져 있어도 TRUNCATE 객체 특권을 회수한다.
(username.) object	스키마 객체 특권의 대상이 되는 객체를 나타낸다. username 부분이 생략되었을 경우 해당 사용자 자신의 스키마 객체에서 해당 이름을 가진 객체를 찾게 된다. 만약 대상으로 동의어가 포함된 경우에는, 해당 동의어의 기반 객체로 인식하고 처리하게 된다. 즉 동의어에 스키마 객체 특권을 부여하거나 회수하는 것은 동의어의 기반 객체에 특권을 부여하거나 회수하는 것과 동일한 효과를 갖게 된다.
FROM grant_clause	스키마 객체 특권을 회수할 대상이 된다. 대상이 될 수 있는 것은 일반 사용자나 역할, 공유 사용자 3종류이다. WITH ANY OBJECT PRIVILEGE 시스템 특권이 없으면, 사용자가 부여한 스키마 객체 특권에 한해 회수가 가능하다.
CASCADE CONSTRAINTS	CASCADE CONSTRAINTS 절은 REFERENCES 스키마 객체 특권을 회수할 경우에만 의미가 있다. 즉, 이전에 REFERENCES 스키마 객체 특권을 허가한 상태에서 ALL (PRIVILEGES)로 허가한 모든 스키마 객체 특권을 회수할 경우에도 의미를 가지게 된다. CASCADE CONSTRAINTS 절을 사용하면 스키마 객체 특권을 회수할 대상에서 REFERENCES 스키마 객체 특권을 사용하여 생성한 참조 무결성 제약조건을 모두 지운 후 특권을 회수하게 된다.

구성요소	설명
	만약 대상자가 참조 무결성 제약조건을 생성한 상태에서, CASCADE CONSTRAINTS 절을 사용하지 않고 REFERENCES 특권을 회수하려 하면 에러가 발생한다.

– grantee_clause

구성요소	설명
user_name	스키마 객체 특권을 회수할 일반 사용자를 명시한다.
role_name	스키마 객체 특권을 회수할 역할을 명시한다.
PUBLIC	스키마 객체 특권을 회수할 공유 사용자를 명시한다.

- 예제

  – revoke_sysprivs

  다음은 **system_privilege**를 명시해 사용자에게 CREATE SESSION 시스템 특권을 부여하고 회수하는 예이다.

```
SQL> CONN sys/tibero
Connected.

SQL> CREATE USER u1 IDENTIFIED BY a;
User created.

SQL> GRANT CREATE SESSION TO u1;
Granted.

SQL> SELECT grantee, privilege FROM dba_sys_privs
     WHERE grantee='U1';

GRANTEE                        PRIVILEGE
------------------------------ --------------
U1                             CREATE SESSION

1 row selected.

SQL> REVOKE CREATE SESSION FROM u1;
Revoked.

SQL> SELECT grantee, privilege FROM dba_sys_privs
     WHERE grantee='U1';

GRANTEE                        PRIVILEGE
------------------------------ ---------
```

```
0 row selected.
```

다음은 사용자 U1에게 부여한 기본 역할 CONNECT를 **role_name**에 명시하여 회수하는 예이다.

```
SQL> SELECT role FROM dba_roles;
ROLE
------------------------------
DBA
CONNECT
RESOURCE

3 rows selected.

SQL> GRANT CONNECT TO u1;
Granted.

SQL> SELECT grantee, granted_role FROM dba_role_privs
     WHERE grantee='U1';

GRANTEE                        GRANTED_ROLE
------------------------------ ------------------------------
U1                             CONNECT

1 row selected.

SQL> REVOKE CONNECT FROM u1;
Revoked.

SQL> SELECT grantee, granted_role FROM dba_role_privs
     WHERE grantee='U1';

GRANTEE                        GRANTED_ROLE
------------------------------ ------------------------------

0 row selected.
```

다음은 **ALL PRIVILEGES**를 사용해 모든 시스템 특권을 부여하고 회수하는 예이다.

```
SQL> GRANT ALL PRIVILEGES TO u1;
Granted.

SQL> REVOKE ALL PRIVILEGES FROM u1;
Revoked.

SQL> GRANT ALL PRIVILEGES TO u1;
```

```
Granted.

SQL> REVOKE CREATE SESSION FROM u1;
Revoked.

SQL> REVOKE ALL PRIVILEGES FROM u1;
TBR-7172: cannot revoke privilege you did not grant

SQL> GRANT CREATE SESSION TO u1;
Granted.

SQL> REVOKE ALL PRIVILEGES FROM u1;
Revoked.
```

ALL PRIVILEGES 절을 사용하여 시스템 특권을 회수할 경우 회수할 사용자 또는 역할이 모든 시스템 특권을 가지고 있어야만 원하는 작업이 정상적으로 실행된다. 위의 예에서 모든 시스템 특권을 가지지 않은 사용자에게서 ALL PRIVILEGES 절을 사용하여 시스템 특권을 회수하려 할 때 **TBR-7172: cannot revoke privilege you did not grant**라는 에러 메시지가 나타나는 것도 볼 수 있다.

다음은 역할 r1을 생성한 후 기본 역할 CONNECT와 시스템 특권 CREATE SESSION을 부여하고, 회수하는 예이다.

```
SQL> CONN sys/tibero
Connected.

SQL> CREATE ROLE r1;
Created.

SQL> GRANT CONNECT, CREATE TABLE TO r1;
Granted.

SQL> SELECT grantee, granted_role FROM dba_role_privs
     WHERE grantee='R1';

GRANTEE                        GRANTED_ROLE
------------------------------ ------------------------------
R1                             CONNECT

1 row selected.

SQL> SELECT grantee, privilege FROM dba_sys_privs
     WHERE grantee='R1';

GRANTEE                        PRIVILEGE
------------------------------ ------------------------------
R1                             CREATE TABLE
```

```
1 row selected.

SQL> REVOKE CONNECT, CREATE TABLE FROM r1;
Revoked.

SQL> SELECT grantee, granted_role FROM dba_role_privs
     WHERE grantee='R1';

GRANTEE                        GRANTED_ROLE
------------------------------ ------------------------------

0 row selected.

SQL> SELECT grantee, privilege FROM dba_sys_privs
     WHERE grantee='R1';

GRANTEE                        PRIVILEGE
------------------------------ ------------------------------

0 row selected.
```

- revoke_objprivs

다음은 사용자 u1이 테이블 t1을 생성하고, 테이블에 대한 스키마 객체 특권을 사용자 u2에게 부여한 후 회수하는 예이다.

```
SQL> CONN sys/tibero
Connected.

SQL> -- 사용자를 생성한다.
SQL> CREATE USER u1 IDENTIFIED BY a;
User created.

SQL> CREATE USER u2 IDENTIFIED BY a;
User created.

SQL> CREATE USER u3 IDENTIFIED BY a;
User created.

SQL> -- 사용자에게 역할을 부여한다.
SQL> GRANT CONNECT, RESOURCE, DBA TO u1;
Granted.

SQL> GRANT CONNECT, RESOURCE TO u2;
Granted.

SQL> -- U1 사용자로 변경한다.
```

```
SQL> CONN u1/a
Connected.

SQL> -- 테스트를 위한 테이블을 생성한다.
SQL> CREATE TABLE t1 (c1 NUMBER, c2 NUMBER, c3 NUMBER);
Table created.

SQL> -- 사용자 U2에게 스키마 객체 특권을 부여하고 확인한다.
SQL> GRANT SELECT ON t1 to u2;
Granted.

SQL> GRANT INSERT (c1, c2), UPDATE (c1) ON t1 TO u2;
Granted.

SQL> SELECT owner, table_name, privilege FROM dba_tbl_privs
     WHERE grantee='U2';

OWNER      TABLE_NAME PRIVILEGE
---------- ---------- --------------------
U1         T1         SELECT

1 row selected.

SQL> SELECT grantee, owner, table_name, column_name, privilege
     FROM dba_col_privs WHERE grantee='U2';

GRANTEE    OWNER      TABLE_NAME COLUMN_NAM PRIVILEGE
---------- ---------- ---------- ---------- --------------------
U2         U1         T1         C1         INSERT
U2         U1         T1         C2         INSERT
U2         U1         T1         C1         UPDATE

3 rows selected.

SQL> -- 사용자 U2에게 부여한 스키마 객체 특권을 회수한다.
SQL> REVOKE SELECT, INSERT, UPDATE ON t1 FROM u2;
Revoked.

SQL> SELECT owner, table_name, privilege FROM dba_tbl_privs
     WHERE grantee='U2';

OWNER      TABLE_NAME PRIVILEGE
---------- ---------- --------------------

0 row selected.

SQL> SELECT grantee, owner, table_name, column_name, privilege
```

```
     FROM dba_col_privs WHERE grantee='U2';

GRANTEE    OWNER      TABLE_NAME COLUMN_NAM PRIVILEGE
---------- ---------- ---------- ---------- --------------------

0 row selected.
```

특권 부분에서 설명한 바와 같이 스키마 객체 특권을 부여할 때에는, 테이블 전체에 대한 특권과 테이블의 일정 컬럼에 대한 특권을 구분해서 부여하였지만, 특권을 회수할 때에는 구분할 필요 없이 바로 회수가 가능하다.

다음은 **ALL PRIVILEGES**를 통해 사용자 u1이 테이블 u1에 대해 사용자 u2에게 부여한 모든 스키마 객체 특권을 회수하는 예이다.

```
SQL> GRANT ALL ON t1 TO u2;
Granted.

SQL> SELECT owner, table_name, privilege FROM dba_tbl_privs
     WHERE grantee='U2';

OWNER      TABLE_NAME PRIVILEGE
---------- ---------- --------------------
U1         T1         ALTER
U1         T1         DELETE
U1         T1         INDEX
U1         T1         INSERT
U1         T1         SELECT
U1         T1         UPDATE
U1         T1         REFERENCES

7 rows selected.

SQL> REVOKE ALTER, REFERENCES ON t1 FROM u2;
Revoked.

SQL> SELECT owner, table_name, privilege FROM dba_tbl_privs
     WHERE grantee='U2';

OWNER      TABLE_NAME PRIVILEGE
---------- ---------- --------------------
U1         T1         DELETE
U1         T1         INDEX
U1         T1         INSERT
U1         T1         SELECT
U1         T1         UPDATE
```

```
5 rows selected.

SQL> REVOKE ALL ON t1 FROM u2;
Revoked.

SQL> SELECT owner, table_name, privilege FROM dba_tbl_privs
     WHERE grantee='U2';

OWNER      TABLE_NAME PRIVILEGE
---------- ---------- --------------------

0 row selected.
```

시스템 특권과는 달리, 스키마 객체 특권은 자신이 부여한 특권만을 찾아서 회수해 주기 때문에, 대상자가 해당 객체에 모든 스키마 객체 특권을 가지고 있지 않아도 동작한다.

다음은 동의어를 사용하여 스키마 객체 특권을 회수하는 예이다.

```
SQL> CREATE SYNONYM s1 FOR t1;
Synonym created.

SQL> -- 사용자 U2에게 스키마 객체 특권을 부여한다.
SQL> GRANT SELECT ON t1 TO u2;
Granted.

SQL> SELECT owner, table_name, privilege FROM dba_tbl_privs
     WHERE grantee='U2';

OWNER      TABLE_NAME PRIVILEGE
---------- ---------- --------------------
U1         T1         SELECT

1 row selected.

SQL> -- 동의어를 이용하여 권한을 회수한다.
SQL> REVOKE ALL ON s1 FROM u2;
Revoked.

SQL> SELECT owner, table_name, privilege FROM dba_tbl_privs
     WHERE grantee='U2';

OWNER      TABLE_NAME PRIVILEGE
---------- ---------- --------------------

0 row selected.
```

동의어를 사용할 경우 기반 객체에 특권을 부여 또는 회수하는 것과 동일한 효과를 갖게 되므로 위와 같이 동작하게 된다.

- grantee_clause

  WITH ANY OBJECT PRIVILEGE 시스템 특권이 없으면, 사용자가 부여한 스키마 객체 특권에 한해 회수가 가능하다.

  다음은 그와 관련된 예이다.

```
SQL> GRANT SELECT ON t1 TO u2 WITH GRANT OPTION;
Granted.

SQL> CONN u2/a
Connected.

SQL> GRANT SELECT ON u1.t1 TO u3;
Granted.

SQL> CONN u1/a
Connected.

SQL> SELECT grantee, owner, table_name, privilege
     FROM dba_tbl_privs
     WHERE owner='U1' AND table_name='t1';

GRANTEE    OWNER      TABLE_NAME PRIVILEGE
---------- ---------- ---------- --------------------
U2         U1         T1         SELECT
U3         U1         T1         SELECT

2 rows selected.

SQL> REVOKE SELECT ON t1 FROM u3;
TBR-7170: object privilege not granted to user

SQL> REVOKE SELECT ON t1 FROM u2;
Revoked.

SQL> SELECT grantee, owner, table_name, privilege
     FROM dba_tbl_privs
     WHERE owner='U1' AND table_name='t1';

GRANTEE    OWNER      TABLE_NAME PRIVILEGE
---------- ---------- ---------- --------------------

0 row selected.
```

위의 예를 보면 사용자 u1이 사용자 u2에게 테이블 t1에 대한 질의 특권을 WITH GRANT OPTION을 사용하여 부여하였다.

즉, 사용자 u2는 사용자 u1의 테이블 t1에 대한 질의 특권을 관리할 수 있는 특권까지 갖게 되었다. 따라서 사용자 u2가 사용자 u3에게 사용자 u1의 테이블 t1에 대한 질의 특권을 부여할 수 있다.

사용자 u1으로 돌아와 테이블 t1에 대한 스키마 객체 특권을 가진 사용자를 살펴보면 사용자 u2, u3이라는 것을 확인할 수 있다. 사용자 u3의 질의 특권을 회수하려 하면 자신이 부여한 스키마 객체 특권이 아니기 때문에 회수할 수 없다. 대신 사용자 u2에게 부여한 스키마 객체 특권을 회수하면, 사용자 u2가 부여한 스키마 객체 특권까지 모두 회수가 되는 것을 볼 수 있다.

WITH GRANT OPTION 에 대한 자세한 내용은 “7.71. GRANT” 부분을 참고한다.

다음은 **CASCADE CONSTRAINTS**를 사용해 참조 무결성 제약조건이 있는 스키마 객체의 특권을 회수하는 예이다.

```
SQL> CREATE TABLE t2 (c1 NUMBER PRIMARY KEY, c2 NUMBER);
Table created.

SQL> GRANT REFERENCES ON t2 TO u2;
Granted.

SQL> CONN u2/a
Connected.

SQL> CREATE TABLE t3 (c1 NUMBER, c2 NUMBER, c3 NUMBER,
                      FOREIGN KEY (c3) REFERENCES u1.t2(c1));
Table created.

SQL> CONN u1/a
Connected.

SQL> REVOKE REFERENCES ON t2 FROM u2;
TBR-17005: cascade option required to revoke.

SQL> REVOKE REFERENCES ON t2 FROM u2 CASCADE CONSTRAINTS;
Revoked.
```

사용자 u1이 테이블 t2에 대해 사용자 u2에게 참조 무결성 제약조건을 생성할 수 있는 스키마 객체 특권을 부여하였고, 사용자 u2는 이를 이용하여 테이블 t3를 생성하였다.

사용자 u1이 사용자 u2에게 부여한 스키마 객체 특권을 회수하려 하면, 사용자 u2가 만든 참조 무결성 제약조건 때문에 회수할 수 없게 된다. 하지만, CASCADE CONSTRAINTS을 사용하게 되면 사용자 u2의 문제가 되는 참조 무결성 제약조건을 제거하고 스키마 객체 특권을 회수할 수 있다.

7.76. TRUNCATE TABLE

테이블의 내용을 초기화한다. 테이블의 모든 내용을 지우는 것은 DROP TABLE을 실행한 뒤 재생성하거나 DELETE를 하는 것으로도 가능하다.

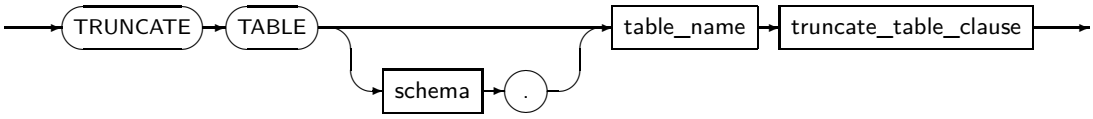
TRUNCATE TABLE은 DELETE에 비해 몇 가지 장점이 있다.

- DROP TABLE 실행하고서 재생성하는 경우 인덱스, CONSTRAINT 등이 소멸되지만, TRUNCATE TABLE을 실행하면 모두 그대로 유지된다.
- DELETE보다 TRUNCATE TABLE이 훨씬 더 빠르게 수행된다.

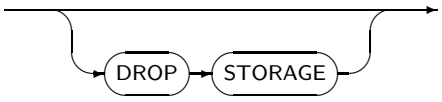
TRUNCATE TABLE의 세부 내용은 다음과 같다.

- 문법

truncate_table



truncate_table_clause



- 특권

  다음 중 하나를 만족해야 TRUNCATE TABLE 문을 실행할 수 있다.

  - 기반 테이블이 사용자 자신의 스키마에 포함되어 있다.
  - DROP ANY TABLE 시스템 특권이 있다.
  - TRUNCATE ANY TABLE 시스템 특권이 있다. 이 권한을 사용하려면 USE_TRUNCATE_PRIVILEGE 파라미터를 Y로 설정되어 있어야 한다.

- 구성요소

구성요소	설명
schema	테이블의 스키마의 이름이다. 생략하면 현재 사용자의 스키마가 사용된다.
DROP STORAGE	세그먼트의 익스텐트를 테이블 스페이스에 반환한다. (기본값)

- 예제

  다음은 **TRUNCATE TABLE**을 사용해 테이블의 내용을 초기화하는 예이다.

```
TRUNCATE TABLE u2.t;
TRUNCATE TABLE t DROP STORAGE;
```

제8장 데이터 조작어

본 장에서는 DML를 자세히 설명한다. DML 명령어는 알파벳 순으로 나열하고, 각 명령어에 대한 설명과 문법, 특권, 예제를 기술한다. 문법을 설명할 때는 "제3장 SQL 연산"의 형식을 그대로 따르고, 키워드 와 문법의 구성요소는 별도의 표로 설명한다.

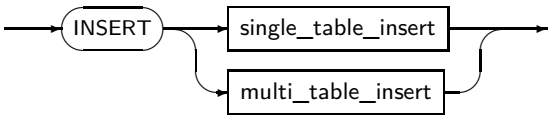
8.1. INSERT

INSERT 구문은 테이블 또는 뷰에 0개 이상의 ROW를 삽입한다.

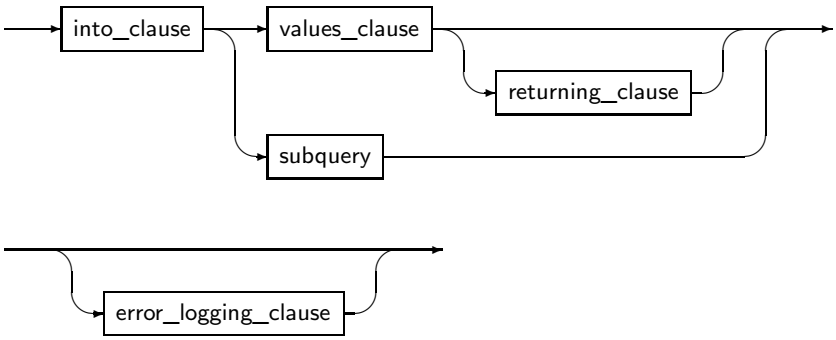
INSERT의 세부 내용은 다음과 같다.

- 문법

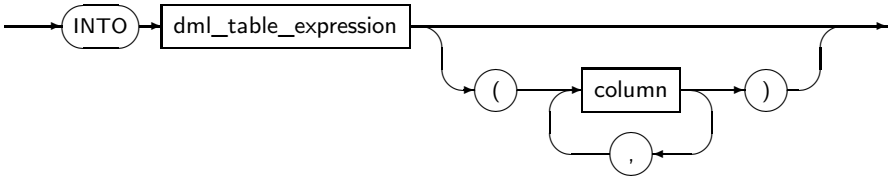
insert



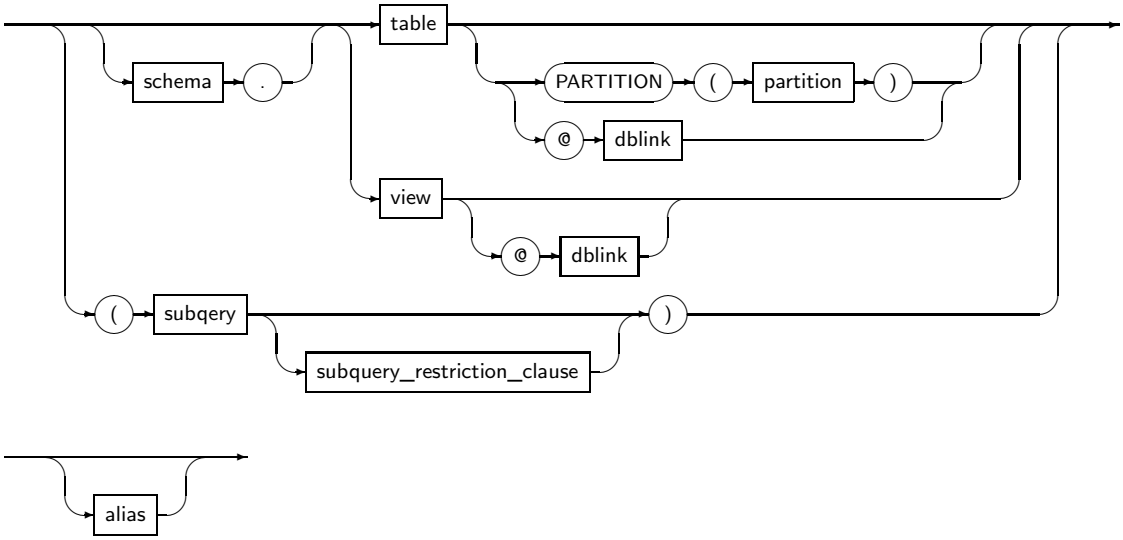
single_table_insert



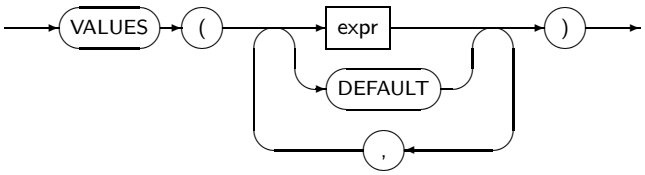
into_clause



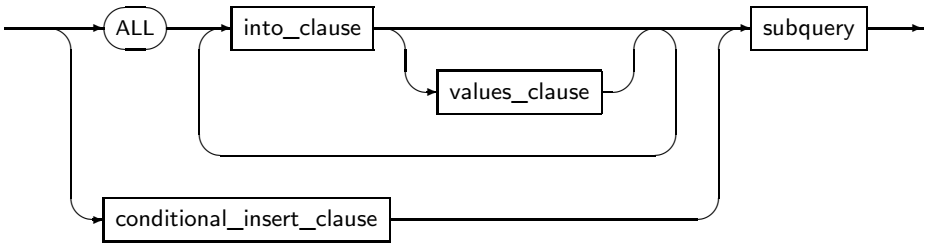
dml_table_expression



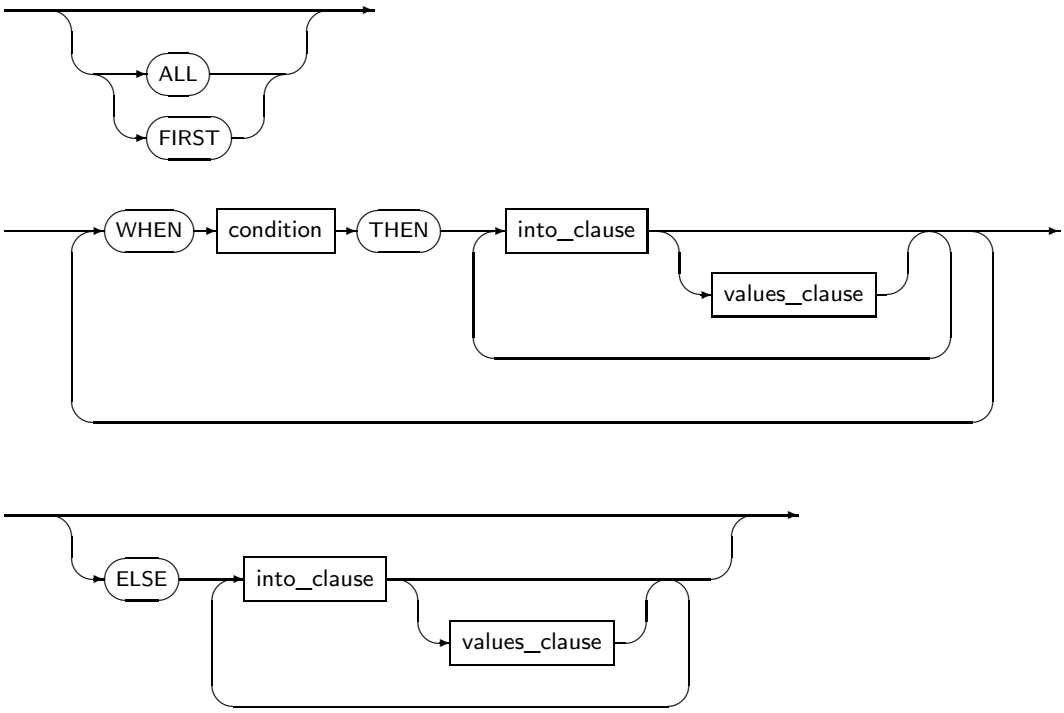
values_clause



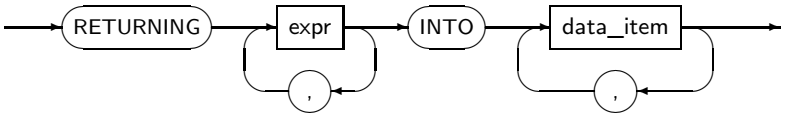
multi_table_insert



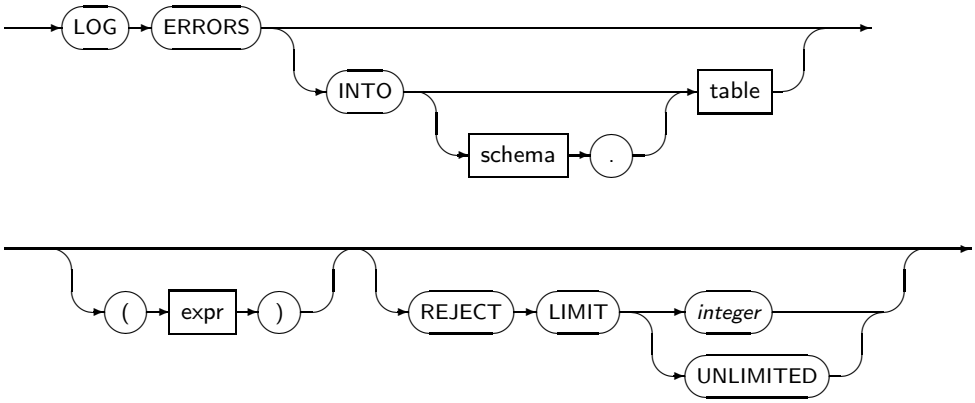
conditional_insert_clause



returning_clause



error_logging_clause



- 특권

  INSERT ANY TABLE 시스템 특권을 가진 사용자는 모든 테이블과 모든 뷰에서 ROW를 삽입할 수 있다. 테이블에 ROW를 삽입하기 위해서는 테이블을 사용자가 소유하고 있거나, 그 테이블에 대해 INSERT 스키마 오브젝트 특권을 가지고 있어야 한다.

  뷰의 기반 테이블에 ROW를 삽입하기 위해서는 다음의 두 가지 조건을 동시에 만족해야 한다.

- 사용자가 뷰를 가지고 있거나 뷰에 대한 INSERT 스키마 오브젝트 특권이 있어야 한다.
- 뷰가 속한 스키마의 사용자가 뷰의 기반 테이블을 가지고 있거나 기반 테이블에 대해 INSERT 스키마 오브젝트 특권을 가지고 있어야 한다.

● 구성요소

- insert

구성요소	설명
single_table_insert	하나의 테이블 또는 뷰에 값을 명시하여 하나의 ROW를 삽입하거나 부질의를 통해서 ROW를 삽입할 수 있다.
multi_table_insert	하나 이상의 테이블에 부질의로부터 계산된 ROW를 삽입할 수 있다. 부질의에 별칭을 지정할 수는 없고 부질의의 SELECT 리스트의 컬럼을 참조하여 values_clause 또는 WHEN 조건절을 구성할 수는 있다. multi_table_insert는 로컬 테이블에만 수행할 수 있다. 뷰 또는 원격(Remote) 테이블에는 수행될 수 없다.

- single_table_insert

구성요소	설명
into_clause	특정 컬럼에만 ROW를 삽입하고자 할 때 사용한다.
values_clause	삽입되는 ROW의 컬럼 값을 지정한다. 삽입할 컬럼 값의 나열 순서는 값을 삽입하고자 하는 컬럼과 같은 순서가 되어야 한다. 값을 삽입할 컬럼을 나열한 경우 같은 순서로 컬럼 값을 나열해야 하고, 값을 삽입할 컬럼을 나열하지 않은 경우 테이블 내에 정의된 컬럼 순서에 따라 컬럼 값을 나열해야 한다.
returning_clause	returning_clause를 사용해 삽입이 일어난 결과 ROW의 값을 반환 받을 수 있다. 삽입이 일어난 ROW로부터 연산식의 값을 계산해 그 결과를 호스트 변수 또는 tbPSM 변수에 저장한다. returning_clause에는 다음과 같은 제약조건이 있다. – expr은 단순 연산식 또는 단독으로 사용된 단일 그룹 집단 함수(GROUP BY 절이 없음)만 가능하다. – 하나의 returning_clause에 단순 연산식과 단일 그룹 집단 함수가 같이 나올 수는 없다. – 단일 그룹 집단 함수 내에 예약어 DISTINCT를 사용할 수 없다. – LONG 타입의 값을 returning_clause를 통해 받을 수 없다.

구성요소	설명
subquery	부질의 결과 반환되는 모든 ROW를 삽입한다. 구체적인 문법은 "5.1. SELECT"를 참고한다. 부질의를 통하여 컬럼 값을 삽입하는 경우 부질의 결과 반환되는 ROW의 컬럼의 개수 및 순서는 값을 삽입하고자 하는 컬럼의 개수 및 순서와 일치해야 한다. 부질의는 임의의 테이블, 뷰를 참조할 수 있으며, 삽입 대상이 되는 테이블 또는 뷰를 참조할 수도 있다. 삽입 대상 테이블 또는 뷰를 참조할 때는 별칭을 통하여 참조한다.
error_logging_clause	에러가 발생할 경우 수행을 중단하지 않고 에러 내용과 삽입하려던 데이터를 에러 로깅 테이블(Error logging table)에 저장한 후 다음 ROW에 대해서 진행한다.

– into_clause

구성요소	설명
dml_table_expression	ROW를 삽입할 객체를 명시한다.
column	특정 컬럼에만 값을 삽입할 경우에는 해당 컬럼의 이름을 나열한다. 나열되지 않은 컬럼에는 기본값이 선언되어 있으면 기본값이 삽입되고 그렇지 않으면 NULL이 삽입된다. 만약 나열되지 않은 컬럼에 대하여 NOT NULL 제약조건이 정의되어 있다면, 새로운 ROW를 삽입하지 않고 에러를 반환한다. 값을 삽입할 컬럼을 나열하지 않으면 모든 컬럼에 대하여 삽입할 값을 지정해야 한다. 만약 하나의 INSERT 문장에 의하여 ROW의 삽입이 진행되다가 에러가 발생한 경우에 이전에 삽입된 모든 ROW도 함께 제거된다.

– dml_table_expression

구성요소	설명
schema	삽입할 객체가 속해 있는 스키마명을 명시한다. 생략하면 현재 사용자의 스키마에서 객체를 찾는다.
table view subquery	테이블이나 뷰의 이름 또는 삽입할 컬럼을 가지고 있는 부질의를 명시한다. 테이블에 INSERT를 수행하면 테이블에 연결된 INSERT 트리거가 동작한다. 뷰 또는 부질의를 명시할 경우 하나의 키 보존 테이블에 대한 삽입이 수행된다. 뷰에 삽입하려면 뷰가 갱신 가능한 뷰여야 한다. 예를 들어 뷰의 정의에 다음이 포함되어 있으면 그 뷰는 갱신 가능한 뷰가 아니다. – 집합 연산자 – DISTINCT 연산자

구성요소	설명
	– 집단 함수또는 분석 함수 – GROUP BY 또는 CONNECT BY 절 – WITH READ ONLY 뷰가 WITH CHECK OPTION과 함께 정의되어 있으면 갱신한 결과를 뷰가 SELECT할 수 있어야만 갱신이 가능하다.
PARTITION (partition)	테이블이 분할 테이블인 경우 파티션명을 명시할 수 있다. 이를 사용하면 데이터베이스 성능 향상에 도움이 된다.
dblink	데이터베이스 링크의 이름 전체 또는 부분을 명시한다. 데이터베이스 링크를 명시할 때는 반드시 앞에 '@'를 붙여야 한다.
WITH CHECK OPTION	삽입 결과를 뷰 또는 부질의가 SELECT할 수 없으면 삽입이 허용되지 않도록 한다.
alias	테이블, 뷰, 부질의가 INSERT 문의 다른 곳에서 참조될 수 있도록 별칭을 지정한다.

– values_clause

구성요소	설명
expr	컬럼 값을 반환하는 임의의 연산식이다. 컬럼은 명시할 수 없다. 구체적인 문법은 “3.3. 연산식”을 참고한다.
DEFAULT	컬럼에 삽입할 값으로 DEFAULT라고 지정한 경우 해당 컬럼의 기본값이 선언되어 있으면 기본값을 삽입하고 그렇지 않으면 NULL을 삽입한다. 뷰에 대해서는 DEFAULT라고 지정할 수 없다.

– returning_clause

구성요소	설명
expr	결과 ROW로부터 returning_clause를 통해 반환할 값을 계산하는 연산식이다. 자세한 내용은 “3.3. 연산식”을 참고한다.
data_item	ROW로부터 계산한 expr 값을 저장할 호스트 변수 또는 tbPSM 변수를 지정한다.

– error_logging_clause

구성요소	설명
table	에러 로깅 테이블의 이름을 명시한다. 명시하지 않으면 **'ERR$_'**를 삽입하려는 테이블명 앞에 붙여서 사용한다.

구성요소	설명
	DBMS_ERRLOG 패키지를 이용하여 자동으로 생성할 수 있다. 사용자가 직접 생성할 수도 있지만 권장하지는 않는다.
expr	문자열을 반환하는 임의의 표현식이다. 수행문의 태그용으로 사용한다.
REJECT LIMIT	허용하는 최대 에러 개수를 명시한다. 최대 에러 개수보다 많은 에러가 발생하면 DML문은 수행이 실패하고 롤백된다. 이 절을 명시하지 않는다면 기본값은 0이다.

다음과 같은 에러를 처리할 수 있다.

- NOT NULL 제약조건
- CHECK 제약조건
- UNIQUE 제약조건
- PRIMARY KEY, FOREIGN KEY 제약조건
- 데이터 타입 에러

다음의 경우에는 사용할 수 없다.

- DEFERED 제약조건
- Direct-Path INSERT, Multi Table INSERT, MERGE 문
- UPDATE 문의 UNIQUE 제약조건, PRIMARY KEY, FOREIGN KEY 제약조건
- LONG, LOB 타입의 컬럼을 가지는 테이블

– multi_table_insert

구성요소	설명
ALL into_clause	예약어 ALL 다음에 여러 개의 into_clause를 사용하면 부질의의 결과 ROW에 대해서 각 into_clause를 한 번씩 수행한다.
values_clause	삽입되는 ROW의 컬럼 값을 지정한다. subquery의 컬럼을 사용할 수 있다. 삽입할 컬럼 값의 나열 순서는 값을 삽입하고자 하는 컬럼과 같은 순서가 되어야 한다. 값을 삽입할 컬럼을 나열한 경우 같은 순서로 컬럼 값을 나열해야 하고, 값을 삽입할 컬럼을 나열하지 않은 경우 테이블 내에 정의된 컬럼 순서에 따라 컬럼 값을 나열해야 한다.
subquery	부질의 결과 반환되는 모든 ROW를 삽입한다. 구체적인 문법은 “5.1. SELECT”를 참고한다. 부질의를 통하여 컬럼 값을 삽입하는 경우 부질의 결과 반환되는 ROW의 컬럼의 개수 및 순서는 값을 삽입하고자 하는 컬럼의 개수 및 순서와 일치해야

구성요소	설명
	한다. 부질의는 임의의 테이블, 뷰를 참조할 수 있으며, 삽입 대상이 되는 테이블 또는 뷰를 참조할 수도 있다. 삽입 대상 테이블 또는 뷰를 참조할 때는 별칭을 통하여 참조한다.
conditional_insert_clause	WHEN 조건절을 통하여 into_clause를 수행할지 안 할지를 판단한다. WHEN 조건절은 부질의의 select 리스트를 참조해야 하며, 하나의 multi_table_insert는 최대 127개의 WHEN 절을 가질 수 있다.

– conditional_insert_clause

구성요소	설명
ALL	ALL을 명시했을 경우 WHEN 조건절을 만족하는 모든 into_clause를 수행한다. ALL 또는 FIRST를 명시하지 않았을 경우 기본값은 ALL이다.
FIRST	FIRST를 명시했을 경우 WHEN 조건절을 만족하는 첫 번째 into_clause 하나만 수행하고 나머지 into_clause는 건너뛴다.
WHEN condition THEN	condition 부분에 조건을 명시한다. 조건을 만족할 경우 into_clause를 수행한다.
into_clause	특정 컬럼에만 ROW를 삽입하고자 할 때 사용한다.
values_clause	삽입되는 ROW의 컬럼 값을 지정한다. 삽입할 컬럼 값의 나열 순서는 값을 삽입하고자 하는 컬럼과 같은 순서가 되어야 한다. 값을 삽입할 컬럼을 나열한 경우 같은 순서로 컬럼 값을 나열해야 하고, 값을 삽입할 컬럼을 나열하지 않은 경우 테이블 내에 정의된 컬럼 순서에 따라 컬럼 값을 나열해야 한다.
ELSE	어느 WHEN 조건절도 만족하지 못했다면 ELSE 절 뒤에 나온 into_clause를 수행한다. ELSE 절이 없을 경우에는 아무 일도 하지 않는다.

- 예제

  다음은 **INSERT**를 사용하는 예이다.

```
INSERT INTO EMP VALUES (35, 'John', 'Houston', 30000, 5);
INSERT INTO EMP (EMPNO, ENAME, DEPTNO) VALUES (35, John, 5);
INSERT INTO EMP VALUES (35, 'John', DEFAULT, 30000, NULL);
```

  다음은 **error logging** 절을 사용하는 예이다.

```
SQL> create table p (a number primary key);

Table 'P' created.
```

```
SQL> insert into p values (1);

1 row inserted.

SQL> insert into p values (2);

1 row inserted.

SQL> insert into p values (3);

1 row inserted.

SQL> insert into p values (4);

1 row inserted.

SQL> create table f (a number references p(a));

Table 'F' created.

SQL> insert into f values (1);

1 row inserted.

SQL> insert into f values (3);

1 row inserted.

SQL> exec dbms_errlog.create_error_log('f');

PSM completed.

SQL> commit;

Commit completed.

SQL> insert into f (select 1 from dual union all select 5 from dual) log errors
reject limit 1;

1 row inserted.

SQL> select * from f;

         A
----------
         1
```

```
         3
         1

3 rows selected.

SQL> select * from err$_f;

TIB_ERR_NUMBER$
---------------
TIB_ERR_MESG$
--------------------------------------------------------------------------------
TIB_ERR_ROWID$     TIB_ERR_OPTYP$
------------------ --------------------
TIB_ERR_TAG$
--------------------------------------------------------------------------------
A
--------------------------------------------------------------------------------
         -10008
 INTEGRITY constraint ('SYS'.'SYS_CON25700497') violated: primary key not found.

                  I

5

1 row selected.
```

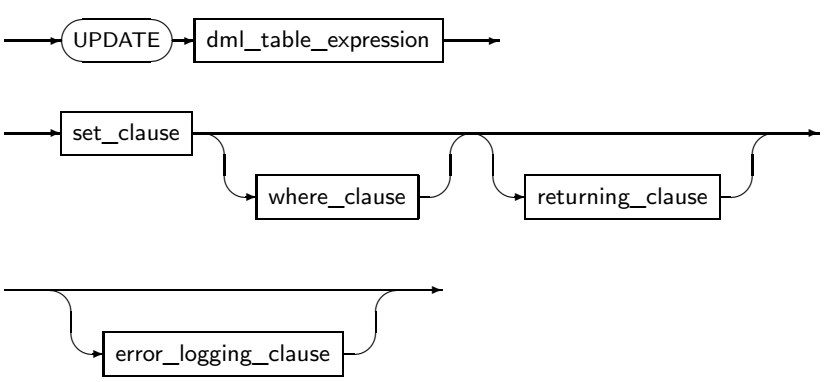
8.2. UPDATE

UPDATE 구문은 테이블 또는 뷰 내의 ROW에 지정된 컬럼의 값을 갱신한다.

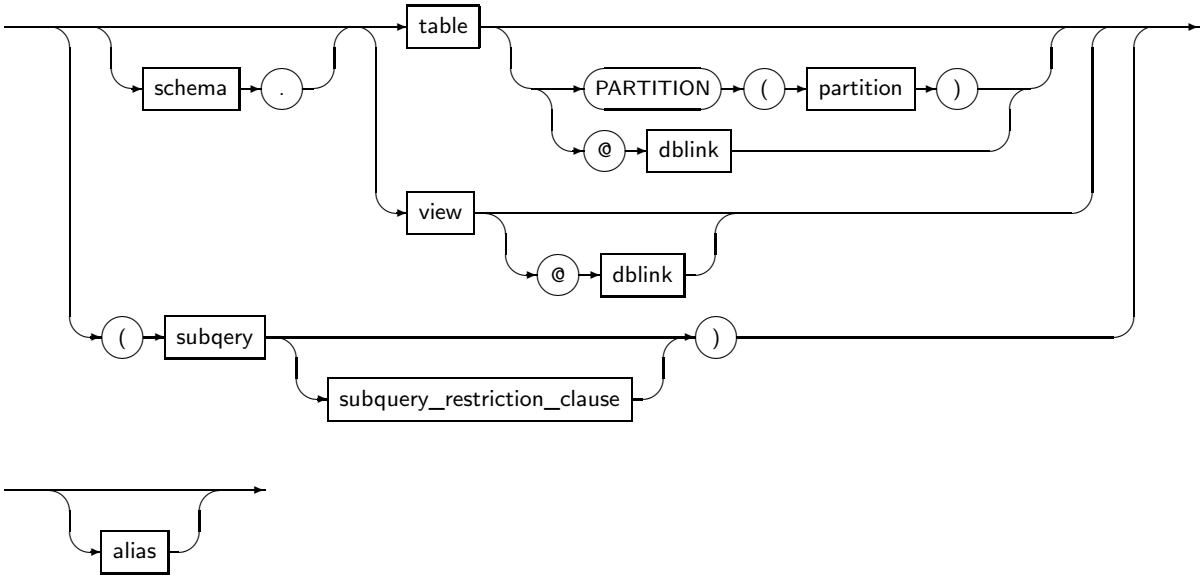
UPDATE의 세부 내용은 다음과 같다.

- 문법

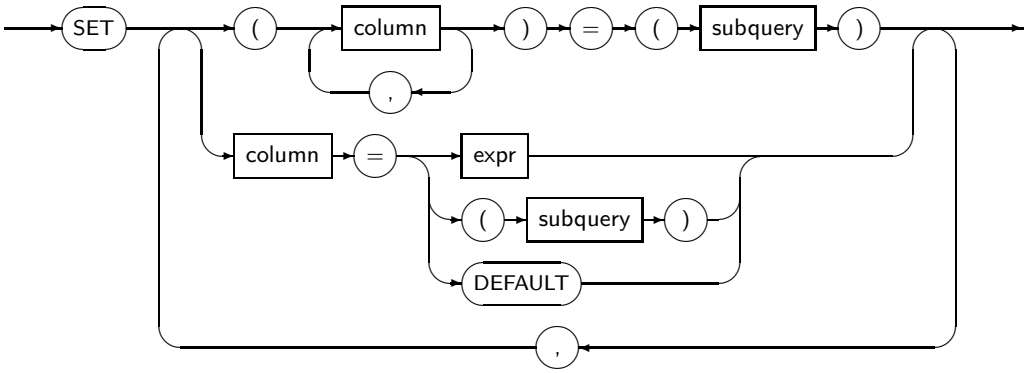
update



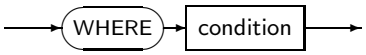
dml_table_expression



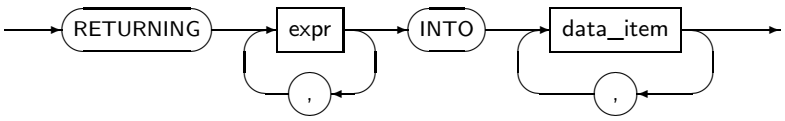
set_clause



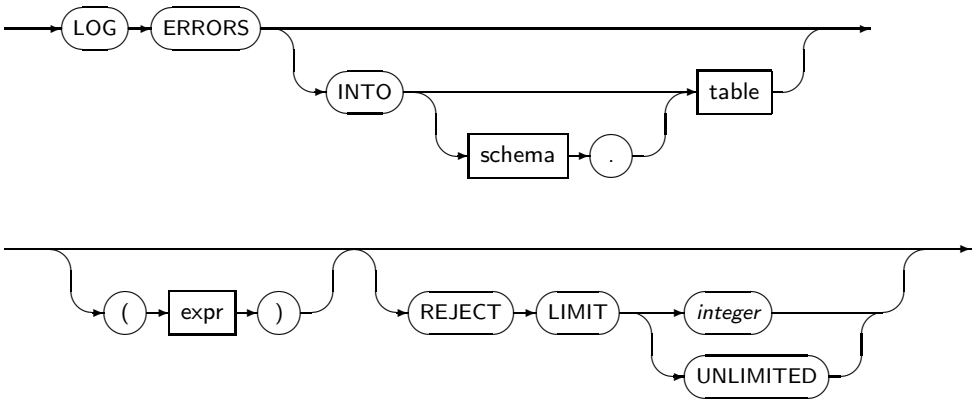
where_clause



returning_clause



error_logging_clause



- 특권

  UPDATE ANY TABLE 시스템 특권을 가지고 있으면 사용자는 모든 테이블과 모든 뷰를 갱신할 수 있다. 테이블을 갱신하기 위해서는 테이블을 사용자가 소유하고 있거나, 그 테이블에 대해 UPDATE 스키마 오브젝트 특권을 가지고 있어야 한다.

  뷰의 기반 테이블을 갱신하기 위해서는 다음의 두 가지 조건을 동시에 만족해야 한다.

  – 사용자가 뷰를 가지고 있거나 뷰에 대한 UPDATE 스키마 오브젝트 특권이 있어야 한다.

  – 뷰가 속한 스키마의 사용자가 뷰의 기반 테이블을 가지고 있거나 기반 테이블에 대해 UPDATE 스키마 오브젝트 특권을 가지고 있어야 한다.

- 구성요소

  – update

구성요소	설명
dml_table_expression	갱신할 테이블 객체를 명시한다.
set_clause	set_clause를 사용해서 갱신할 컬럼의 값을 명시한다.
where_clause	조건이 TRUE인 ROW에 대해서만 갱신을 수행하도록 지정한다. 이 절을 생략하면 테이블 또는 뷰의 모든 ROW에 대해 갱신을 수행한다.
returning_clause	returning_clause를 사용해 갱신이 일어난 결과 ROW의 값을 반환 받을 수 있다. 갱신이 일어난 ROW로부터 연산식의 값을 계산해 그 결과를 호스트 변수 또는 tbPSM 변수에 저장한다.
error_logging_clause	에러가 발생할 경우 수행을 중단하지 않고 에러 내용과 삽입하려던 데이터를 에러 로깅 테이블(Error logging table)에 저장한 후 다음 ROW에 대해서 진행한다.

  – dml_table_expression

구성요소	설명
schema	갱신할 객체가 속해 있는 스키마를 명시한다. 생략하면 현재 사용자의 스키마에서 객체를 찾는다.
table view subquery	테이블이나 뷰의 이름 또는 갱신할 컬럼을 가지고 있는 부질의를 명시한다. 테이블에 UPDATE를 수행하면 테이블에 연결된 UPDATE 트리거가 동작한다. 뷰를 명시할 경우 뷰의 기반 테이블이 갱신된다. 뷰를 갱신하려면 뷰가 갱신 가능한 뷰여야 한다. 예를 들어 뷰의 정의에 다음이 포함되어 있으면 그 뷰는 갱신 가능한 뷰가 아니다. – 집합 연산자 – DISTINCT 연산자 – 집단 함수 또는 분석 함수 – GROUP BY 또는 CONNECT BY 절 – WITH READ ONLY
dblink	데이터베이스 링크의 이름 전체 또는 부분을 명시한다. 데이터베이스 링크를 명시할 때는 반드시 앞에 '@'를 붙여야만 한다. 데이터베이스 링크에는 다음과 같은 제약조건이 있다. – 원격 테이블에서는 Tibero에서 지원하지 않는 사용자정의 타입 또는 REF 객체에 대한 질의를 할 수 없다. – 원격 테이블에서는 Tibero에서 지원하지 않는 ANYTYPE, ANYDATA 또는 ANYDATASET 타입의 컬럼에 대한 질의를 할 수 없다.
PARTITION (partition)	테이블이 분할 테이블인 경우 파티션명을 명시할 수 있다. 이를 사용하면 데이터베이스 성능 향상에 도움이 된다.
WITH CHECK OPTION	갱신의 결과를 부질의가 SELECT할 수 없으면 갱신이 허용되지 않도록 한다.
alias	테이블, 뷰, 부질의가 UPDATE 문의 다른 곳에서 참조될 수 있도록 별칭을 지정한다.

– set_clause

구성요소	설명
column	갱신할 컬럼의 이름을 명시한다. 명시하지 않은 컬럼의 값은 바뀌지 않는다. 대용량 객체형 컬럼을 명시할 경우 컬럼이 분할 테이블의 분할 키 정의에 참여하고 있을 때는 UPDATE의 결과로 인해 현재 ROW가 참여하고 있는 파티션에 이 ROW가 계속 참여할 수 없을 때는 UPDATE가 실패한다.
subquery	최대 1개의 ROW를 반환하는 스칼라 서브쿼리 (scalar subquery)를 명시한다. 여러 ROW가 반환되면 UPDATE가 실패한다.

구성요소	설명
	update_set_clause에서 하나의 컬럼을 명시했을 경우는 부질의도 SELECT 리스트를 통해 하나의 값만 반환해 주어야 한다. update_set_clause에서 여러 개의 컬럼을 명시했을 경우는 부질의가 SELECT 리스트를 통해 반환하는 값의 개수와 컬럼의 개수가 일치하여야 한다. 부질의가 ROW를 반환하지 못했을 경우 컬럼에 갱신되는 값은 NULL이다.
expr	컬럼에 갱신할 새 값을 명시하는 연산식이다. 자세한 내용은 “3.3. 연산식”을 참고한다.
DEFAULT	컬럼에 기본값이 지정된 경우 그 값을 갱신할 값으로 지정한다. 기본값이 없는데 명시했을 경우는 NULL로 갱신된다. 뷰를 갱신할 때는 DEFAULT를 명시할 수 없다.

– where_clause

구성요소	설명
condition	WHERE 예약어 뒤에 조건식을 명시한다. 조건식이 TRUE인 ROW만 갱신하도록 지정한다. 갱신의 대상이 되는 ROW를 기반으로 condition 값을 계산하며, 부질의를 포함할 수 있다. where_clause를 생략하면 테이블 또는 뷰의 모든 ROW를 갱신한다.

– returning_clause

구성요소	설명
expr	결과 ROW로부터 returning_clause 절을 통해 반환할 값을 계산하는 연산식이다. 자세한 내용은 “3.3. 연산식”을 참고한다.
data_item	ROW로부터 계산한 expr 값을 저장할 호스트 변수 또는 tbPSM 변수에 저장한다.

returning_clause에는 다음과 같은 제약조건이 있다.

- expr은 단순 연산식 또는 단독으로 사용된 단일 그룹 집단 함수(GROUP BY 절이 없음)만 가능하다.
- 하나의 returning_clause에 단순 연산식과 단일 그룹 집단 함수가 같이 나올 수는 없다.
- 단일 그룹 집단 함수 내에 예약어 DISTINCT를 사용할 수도 없다.
- LONG 타입의 값을 returning_clause를 통해 받을 수 없다.

● 예제

다음은 **UPDATE**를 사용하는 예이다.

```
UPDATE EMP SET SALARY = 35000;
UPDATE EMP SET SALARY = 35000 WHERE DEPTNO = 5;
UPDATE EMP SET DEPTNO = DEFAULT WHERE DEPTNO IS NULL;
UPDATE EMP
SET SALARY = SALARY * 1.05,
ADDR = (SELECT LOC FROM DEPT WHERE DEPTNO = 5)
WHERE DEPTNO = 5;
```

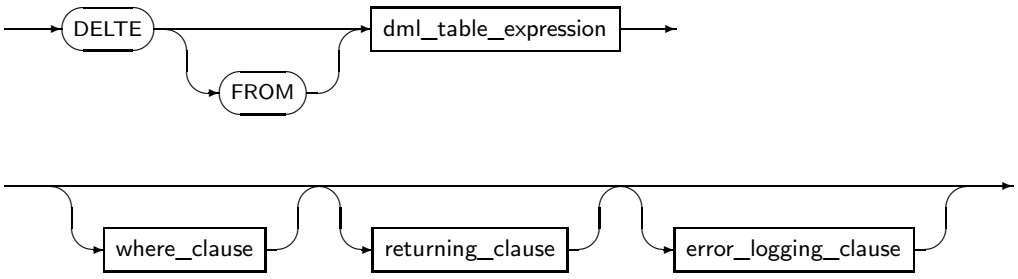
8.3. DELETE

DELETE 구문은 테이블이나 뷰에서 ROW를 삭제한다. 뷰의 경우(가능할 때만) ROW가 삭제될 기반 테이블이 하나로 정해진다. 테이블은 파티션으로 분할되어 있을 수 있다.

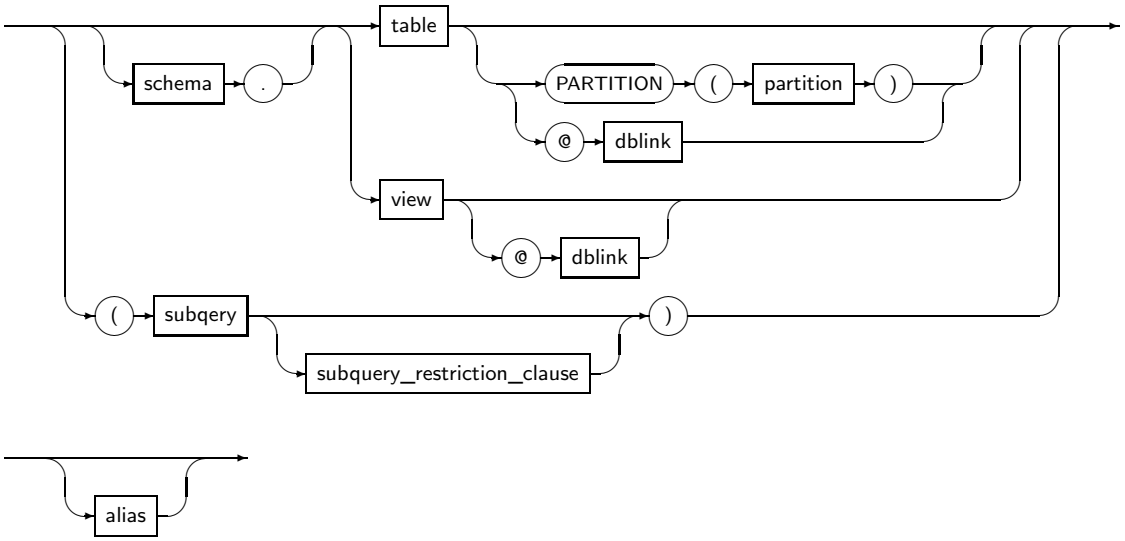
DELETE의 세부 내용은 다음과 같다.

- 문법

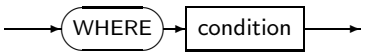
delete



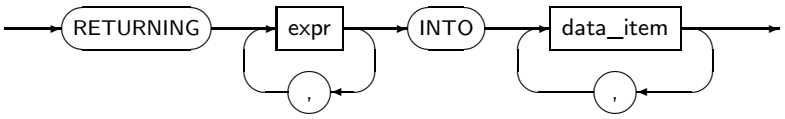
dml_table_expression



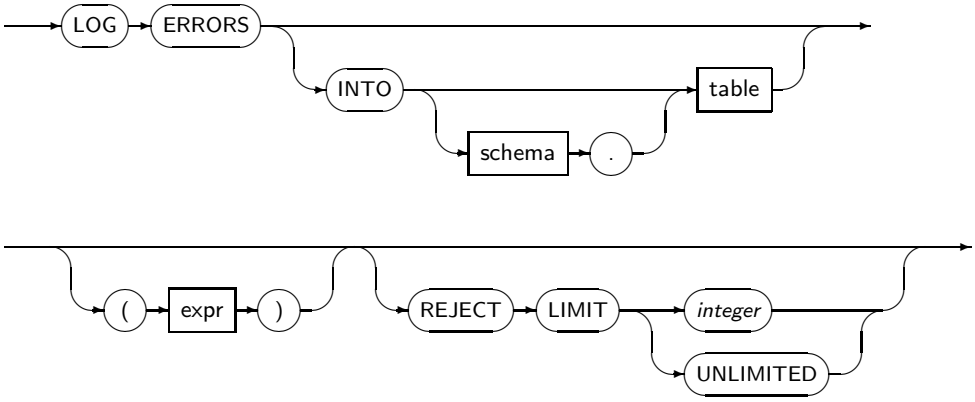
where_clause



returning_clause



error_logging_clause



- 특권

  DELETE ANY TABLE 시스템 특권을 가진 사용자는 모든 테이블과 모든 뷰에서 ROW를 삭제할 수 있다. 테이블에서 ROW를 삭제하기 위해서는 테이블을 사용자가 소유하고 있거나, 그 테이블에 대해 DELETE 스키마 오브젝트 특권을 가지고 있어야 한다.

  뷰의 기반 테이블에서 ROW를 삭제하기 위해서는 다음의 두 가지 조건을 동시에 만족해야 한다.

  – 사용자가 뷰를 가지고 있거나 뷰에 대한 DELETE 스키마 오브젝트 특권이 있어야 한다.

  – 뷰가 속한 스키마의 사용자가 뷰의 기반 테이블을 가지고 있거나 기반 테이블에 대해 DELETE 스키마 오브젝트 특권을 가지고 있어야 한다.

- 구성요소

  – delete

구성요소	설명
dml_table_expression	삭제할 테이블 객체를 명시한다.
where_clause	조건이 TRUE인 ROW만 삭제하도록 지정한다. 삭제의 대상이 되는 ROW를 기반으로 조건의 값을 계산하며, 부질의를 포함할 수 있다. 이 절을 생략하면 테이블 또는 뷰의 기반 테이블의 모든 ROW를 삭제한다. 자세한 내용은 "3.4. 조건식"을 참고한다.

구성요소	설명
returning_clause	returning_clause를 사용하면 삭제해서 없어져 버린 ROW의 값을 반환 받을 수 있다. 삭제된 ROW로부터 연산식의 값을 계산해 그 결과를 호스트 변수 또는 tbPSM 변수에 저장한다.
error_logging_clause	에러가 발생할 경우 수행을 중단하지 않고 에러 내용과 삽입하려던 데이터를 에러 로깅 테이블(Error logging table)에 저장한 후 다음 ROW에 대해서 진행한다.

– dml_table_expression

구성요소	설명
schema	테이블이나 뷰가 속해 있는 스키마를 명시한다. 생략하면 현재 사용자의 스키마에서 객체를 찾는다.
table view subquery	테이블이나 뷰의 이름 또는 삭제할 ROW를 포함하는 ROW 집합을 SELECT하는 부질의를 명시한다. 테이블에 DELETE를 수행하면 테이블에 연결된 DELETE 트리거가 동작한다. 뷰를 명시할 경우 뷰의 기반 테이블에서 ROW가 삭제된다. 뷰에서 ROW를 삭제하려면 뷰가 갱신 가능한 뷰여야 한다. 예를 들어 뷰의 정의에 다음이 포함되어 있으면 그 뷰는 갱신 가능한 뷰가 아니다. – 집합 연산자 – DISTINCT 연산자 – 집단 함수 또는 분석 함수 – GROUP BY 또는 CONNECT BY 절 – WITH READ ONLY ROW를 삭제한 후에도 테이블이나 인덱스에서 그 ROW가 차지하고 있었던 공간은 그대로 유지된다.
PARTITION (partition)	테이블이 분할 테이블인 경우 파티션명을 명시할 수 있다. 이를 사용하면 데이터베이스 성능 향상에 도움이 된다.
dblink	데이터베이스 링크의 이름 전체 또는 부분을 명시한다. 데이터베이스 링크를 명시할 때는 반드시 앞에 '@'를 붙여야만 한다. 데이터베이스 링크에는 다음과 같은 제약조건이 있다. – 원격 테이블에서는 Tibero에서 지원하지 않는 사용자정의 타입 또는 REF 객체에 대한 질의를 할 수 없다. – 원격 테이블에서는 Tibero에서 지원하지 않는 ANYTYPE, ANYDATA 또는 ANYDATASET 타입의 컬럼에 대한 질의를 할 수 없다.

구성요소	설명
subquery_restriction_clause	예약어 WITH READ ONLY가 명시되어 있으면 삭제할 수 없다. WITH CHECK OPTION은 무시된다.
alias	테이블, 뷰, 부질의가 DELETE 문의 다른 곳에서 참조될 수 있도록 별칭을 지정한다.

– where_clause

구성요소	설명
condition	WHERE 예약어 뒤에 조건식을 명시한다. 조건식이 TRUE인 ROW만 제거하도록 지정한다. 제거의 대상이 되는 ROW를 기반으로 condition 값을 계산하며, 부질의를 포함할 수 있다. where_clause를 생략하면 테이블 또는 뷰의 모든 ROW를 제거한다.

– returning_clause

구성요소	설명
expr	결과 ROW로부터 returning_clause 절을 사용해 반환할 값을 계산할 때 사용하는 연산식이다. 자세한 내용은 “3.3. 연산식”을 참고한다.
data_item	ROW로부터 계산된 expr 값을 호스트 변수 또는 tbPSM 변수에 저장한다.

returning_clause에는 다음과 같은 제약조건이 있다.

- expr은 단순 연산식 또는 단독으로 사용된 단일 그룹 집단 함수(GROUP BY 절이 없음)만 가능하다.
- 하나의 returning_clause에 단순 연산식과 단일 그룹 집단 함수가 같이 나올 수는 없다.
- 단일 그룹 집단 함수 내에 예약어 DISTINCT를 사용할 수도 없다.
- LONG 타입의 값을 returning_clause를 통해 받을 수 없다.

● 예제

다음은 **DELETE**를 사용하는 예이다.

```
DELETE FROM EMP;
DELETE FROM John.EMP WHERE SALARY < 20000
```

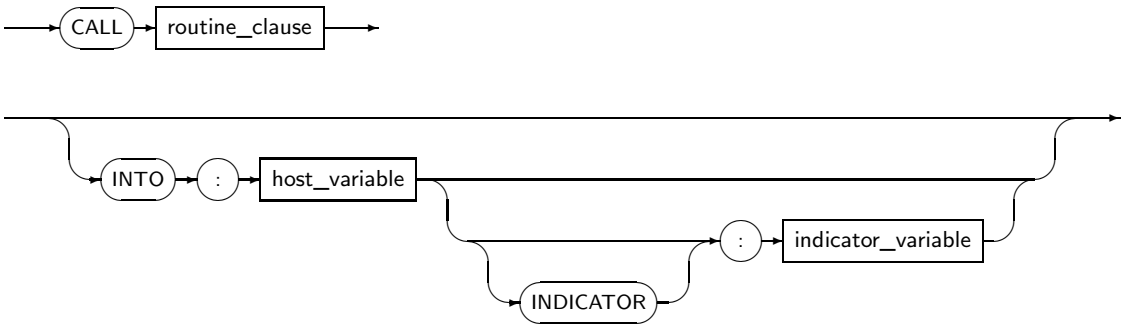
8.4. CALL

CALL 구문은 단독으로 정의되거나 패키지 내에 정의된 프러시저나 함수를 실행한다.

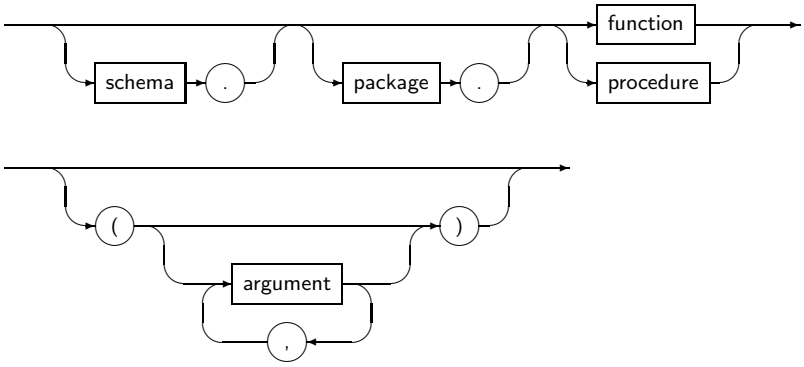
CALL의 세부 내용은 다음과 같다.

- 문법

call



routine_clause



- 특권
  - 단독으로 정의된 프러시저나 함수를 실행하려면 그 프러시저나 함수에 대한 EXECUTE 스키마 오브젝트 특권을 가지고 있어야 한다.
  - 프러시저나 함수가 패키지 내에 정의된 경우에는 패키지에 대한 EXECUTE 특권을 가지고 있어야 한다.
  - 또한 EXECUTE ANY PROCEDURE 시스템 특권이 있으면 어떤 프러시저나 함수라도 실행할 수 있다.
- 구성요소
  - call

구성요소	설명
routine_clause	단독으로 정의된 프러시저나 함수가 속해있는 스키마, 프러시저나 함수의 이름 등 세부내용을 명시한다.
INTO	함수를 실행하는 경우에만 INTO 절을 명시해야 한다.
:host_variable	반환 값을 저장할 호스트 변수를 명시한다.
:indicator_variable	호스트 변수의 값 또는 상태를 표시하기 위한 지시자 변수를 명시한다.

– routine_clause

구성요소	설명
schema	단독으로 정의된 프러시저나 함수가 속해 있는 스키마를 명시한다. 또는 프러시저나 함수를 포함하는 패키지가 속해 있는 스키마를 명시한다.
package	프러시저나 함수를 포함하는 패키지를 명시한다.
function	실행하고자 하는 함수의 이름을 명시한다.
procedure	실행하고자 하는 프러시저의 이름을 명시한다.
argument	프러시저나 함수가 파라미터를 받는 경우에는 파라미터를 명시한다. argument에는 다음과 같은 제약조건이 있다. – 파라미터에는 의사 컬럼을 사용할 수 없다. – IN OUT 또는 OUT으로 명시된 파라미터는 대응하는 호스트 변수가 있어야 한다. – 파라미터의 수는 반환 값을 포함하여 1000개까지 사용할 수 있다. – 4KB 이상의 크기를 가지는 문자열 또는 RAW, LONG RAW 데이터를 파라미터로 사용할 수 없다.

- 예제

  다음은 **CALL**을 사용하는 예이다.

```
CALL get_board_name (30);
```

8.5. MERGE

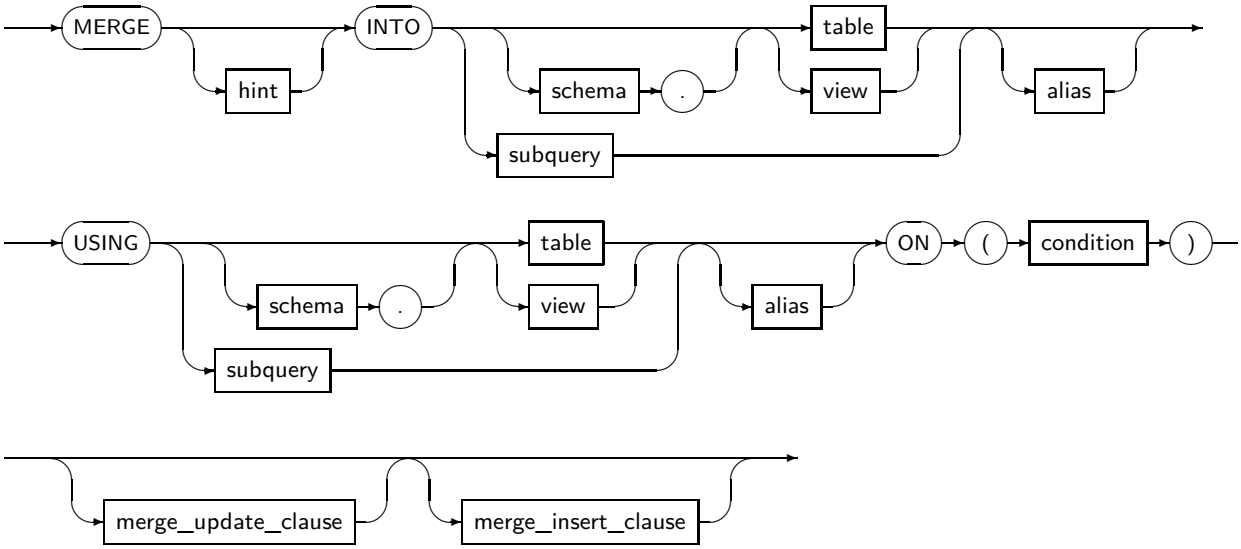
MERGE 구문은 하나, 또는 그 이상의 원본(Source) 데이터로부터 ROW를 선택하여 테이블에 삽입 또는 갱신 작업을 수행한다. 삽입, 갱신 작업 중 어느 작업을 수행할 것인지를 결정하는 조건을 명시할 수 있다.

MERGE는 여러 작업을 통합할 수 있는 편리한 방법이다. MERGE를 수행함으로써 여러 삽입, 갱신, 삭제 작업을 수행하여야 하는 수고를 덜 수 있다. 하나의 MERGE 문에서 같은 ROW를 여러 번 갱신할 수 없다.

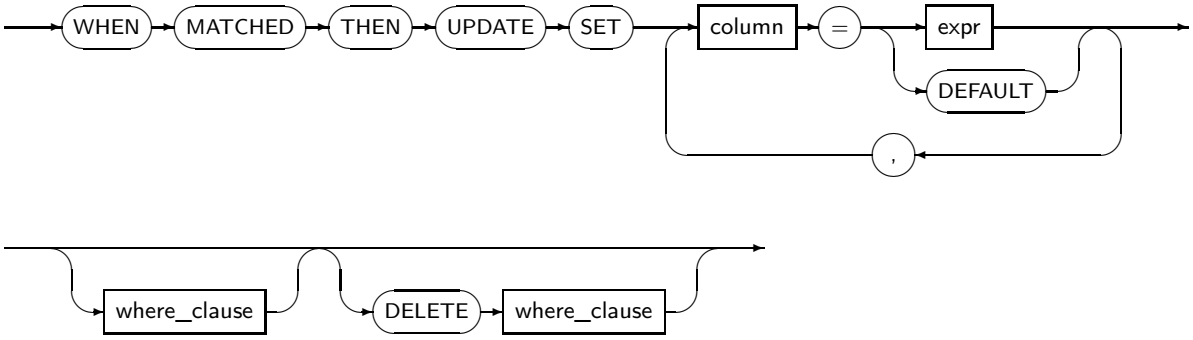
MERGE의 세부 내용은 다음과 같다.

- 문법

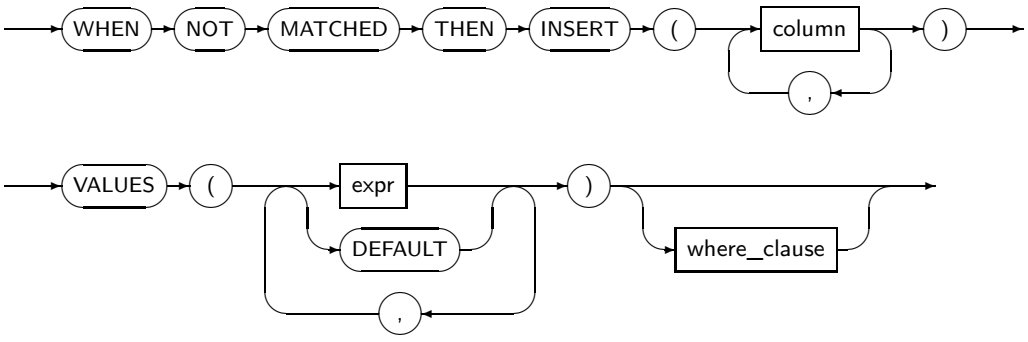
merge



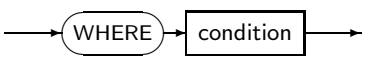
merge_update_clause



merge_insert_clause



where_clause



- 특권

– MERGE를 수행하기 위해서는 대상(Target) 데이터에 대한 INSERT 스키마 오브젝트 특권과 UPDATE 스키마 오브젝트 특권을 가지고 있어야 하며, 원본 데이터에 대한 SELECT 스키마 오브젝트 특권을 가지고 있어야 한다.

– merge_update_clause에 DELETE clause를 명시하게 위해서는 대상 데이터에 대한 DELETE 스키마 오브젝트 특권 또한 가지고 있어야 한다.

- 구성요소

  – merge

구성요소	설명
hint	힌트를 명시한다.
INTO	데이터를 삽입 또는 갱신할 테이블을 지정한다.
schema	스키마의 이름을 명시한다.
alias	별칭을 명시한다.
USING	삽입 또는 갱신할 원본 데이터를 명시하기 위해 USING 절을 사용한다. 원본 데이터로는 테이블, 뷰, 또는 부질의의 결과가 사용될 수 있다.
table, view, subquery	원본 데이터로 사용될 테이블, 뷰, 부질의를 명시한다.
ON (condition)	MERGE작업이 수행될 조건을 명시하기 위해 ON 절을 사용한다. condition 부분에 조건식을 명시한다. 대상 데이터의 테이블의 각 ROW에 대하여 조건이 TRUE일 경우 원본 데이터의 테이블의 대응하는 데이터로 ROW를 갱신한다. 모든 ROW에 대하여 조건이 FAULSE일 경우 대응하는 원본 데이터 테이블의 ROW를 대상 데이터 테이블에 삽입한다.
merge_update_clause	merge_update_clause는 대상 데이터 테이블의 새 컬럼 값을 명시한다. ON 절이 참일 경우 이 갱신을 수행한다. 갱신이 실행되면 대상 데이터 테이블에 설정된 모든 갱신 트리거가 작동된다. merge_update_clause는 단독으로 명시될 수도 있고 merge_insert_clause와 같이 명시될 수도 있다. 만약 두 가지 모두 명시될 경우 순서는 상관 없다. merge_update_clause를 먼저 명시해도 되고, merge_insert_clause를 먼저 명시해도 된다. merge_update_clause는 ON 절에서 참조된 컬럼은 갱신할 수 없다는 제약이 있다.
merge_insert_clause	merge_insert_clause는 ON 절의 조건이 거짓일 경우 대상 데이터 테이블에 삽입될 값을 명시한다. 삽입이 실행되면 대상 데이터 테이블에 정의된 모든 삽입 트리거가 작동된다. 모든 원본 데이터의 ROW를 테이블에 삽입하기 위해서 ON 절 조건에 0=1과 같이 항상 거짓으로 평가되는 필터를 사용하면 된다. 이러한 방법은 merge_update_clause를 생략하는 것과는 다르다. merge_update_clause를

구성요소	설명
	생략하는 경우에 데이터베이스는 조인을 수행하지만 필터를 사용할 경우에 데이터베이스는 조인을 하지 않고 무조건 삽입을 수행한다. merge_insert_clause는 단독으로 명시될 수도 있고 merge_update_clause와 같이 명시될 수도 있다. 두 가지 모두 명시될 경우 순서는 상관 없다.

- merge_update_clause

구성요소	설명
column	갱신을 수행할 컬럼의 이름을 명시한다.
expr	컬럼의 값을 갱신하는 데 사용될 임의의 연산식이다.
DEFAULT	컬럼이 값을 기본값으로 갱신할 때 사용된다.
where_clause	갱신 조건의 설정을 위해서 where_clause를 명시한다. 원본 데이터에 대한 조건과 대상 데이터에 대한 조건 모두 명시될 수 있다.
DELETE where_clause	갱신 중 데이터를 삭제하기 위해서 DELETE where_clause를 명시한다. DELETE where_clause에 의해 삭제되는 ROW는 갱신에 의해 영향을 받은 ROW 뿐이다. ROW가 삭제되면 설정된 모든 삭제 트리거가 작동된다.

- merge_insert_clause

구성요소	설명
column	삽입을 수행할 컬럼의 이름을 명시한다. 이 부분을 생략하면, 대상 데이터 테이블의 컬럼의 수는 VALUES 절에 명시된 값의 수와 일치하여야 한다.
expr	컬럼의 값을 삽입하는 데 사용될 임의의 연산식이다.
DEFAULT	컬럼이 값을 기본값으로 삽입할 때 사용된다.
where_clause	특정 조건을 만족시키는 ROW만을 삽입하고자 할 때에는 where_clause를 명시한다. 이 조건은 원본 데이터의 테이블의 값만을 참조할 수 있다.

- where_clause

구성요소	설명
condition	갱신 조건이나 삽입 조건을 설정하기 위해서 where_clause를 명시한다. condition에는 조건식을 명시한다.

- 예제

  다음은 **MERGE**를 사용하는 예이다.

```
MERGE INTO BONUS USING PERSONNEL
ON (B.PNUM = PERSONNEL.PNUM)
```

```
WHEN MATCHED THEN UPDATE SET BONUS.BONUS = BONUS.BONUS*1.5
DELETE WHERE (PERSONNEL.SALARY > 3000)
WHEN NOT MATCHED THEN INSERT VALUES (PERSONNEL.PNUM, PERSONNEL.SALARY*0.2);
```

8.6. 병렬 DML

Tibero는 다음의 DML 문장을 병렬로 수행할 수 있다.

- INSERT INTO SELECT

  병렬 DML도 병렬 질의와 마찬가지로 PARALLEL 힌트를 지정하여 사용할 수 있다. INSERT INTO SELECT 문장은 PARALLEL 힌트의 위치에 따라 어느 부분을 병렬화할 것인지 지정할 수 있다. SELECT 뒤에만 힌트를 주면 SELECT하는 부분만 병렬화되고 INSERT 뒤에만 힌트를 주면 실제 INSERT하는 작업만 병렬화하게 된다. 따라서 두 작업 모두 병렬화하기 위해서는 INSERT, SELECT 뒤에 모두 힌트를 주어야 한다.

- UPDATE

- DELETE

8.6.1. 수행 방법

DML을 병렬로 수행하기 위해서는 우선 다음과 같이 세션에 병렬 DML의 실행 여부를 설정해 주어야 한다.

```
SQL> ALTER SESSION ENABLE PARALLEL DML;
Session altered.
```

위의 예에서처럼 ALTER SESSION 문장으로 세션의 속성을 변경시키지 않으면 PARALLEL 힌트는 동작하지 않고 병렬 DML은 수행되지 않는다.

다음은 트랜잭션이 수행되는 중에 병렬 DML을 수행했을 때 발생하는 에러이다.

```
SQL> INSERT INTO TEMP_TBL VALUES (1, 1);
1 row inserted.

SQL> ALTER SESSION ENABLE PARALLEL DML;
TBR-12064: Unable to alter the session PDML state within a transaction
```

트랜잭션이 수행되는 중에는 COMMIT 또는 ROLLBACK 명령어로 현재 트랜잭션의 수행을 종료해야만, ALTER SESSION 문장을 수행할 수 있다.

병렬 DML을 수행할 때 트랜잭션과 관련해 다음 두 가지 제약조건이 있으며, 이러한 제약이 있는 것은 트랜잭션의 정합성을 보장하기 위해서이다.

- 병렬 DML을 수행한 트랜잭션에서는 병렬 DML로 수정한 테이블에 어떠한 접근도 할 수 없다.

- 같은 트랜잭션 내에서 수정한 테이블에 대해 병렬 DML로 접근하려 할 경우에도 병렬 DML을 수행할 수 없다.

다음은 병렬 DML을 수행한 트랜잭션에서는 병렬 DML로 수정한 테이블에 접근하는 경우 에러가 발생하는 예이다.

```
SQL> ALTER SESSION ENABLE PARALLEL DML;
Session altered.

SQL> INSERT /*+ parallel (3) */ INTO TEMP_TBL2 SELECT /*+ parallel (3) */ * FROM
     TEMP_TBL;
100000 rows inserted.

SQL> SELECT COUNT(*) FROM TEMP_TBL2;
TBR-12063: Unable to read or modify an object after modifying it in parallel
```

다음은 같은 트랜잭션에서 수정한 테이블에 대해 병렬 DML로 접근할 경우에 에러가 발생하는 예이다.

```
SQL> ALTER SESSION ENABLE PARALLEL DML;
Session altered.

SQL> INSERT INTO TEMP_TBL2 VALUES (1, 1);
1 row inserted.

SQL> INSERT /*+ parallel (3) */ INTO TEMP_TBL2 SELECT /*+ parallel (3) */ * FROM
     TEMP_TBL;
TBR-12067: Unable to modify an object with PDML after modifying it
```

8.6.2. 제약 사항

병렬 DML은 다음 같은 경우에서는 수행되지 않는다.

- 삽입, 갱신, 삭제를 하려는 테이블에 트리거가 있는 경우
- returning_clause가 있는 경우
- 대상 테이블에 대용량 객체형 컬럼이 있는 경우
- 대상 테이블의 인덱스가 online-rebuild를 수행 중인 경우
- standby replication을 사용 중인 경우
- self-referential integrity, delete cascade, deferred integrity 등이 있는 경우
- 분산 트랜잭션을 사용하는 경우
- DDL, TCS, SCS를 사용하는 경우

제9장 트랜잭션 및 세션 관리 언어

본 장에서는 트랜잭션 및 세션 관리를 위한 명령어를 자세히 설명한다. 트랜잭션 및 세션 관리를 위한 명령어는 알파벳 순으로 나열하고, 각 명령어에 대한 설명과 문법, 예제를 기술한다. 문법을 설명할 때는 "제3장 SQL 연산"의 형식을 그대로 따르고, 키워드와 문법의 구성요소는 별도의 표로 설명한다.

9.1. ALTER SESSION

실행자가 가지고 있는 세션 환경을 조정한다. ALTER SESSION으로 변경된 초기화 환경 변수나 세션 환경 변수는 실행자의 접속이 끊어질 때까지 유효하다.

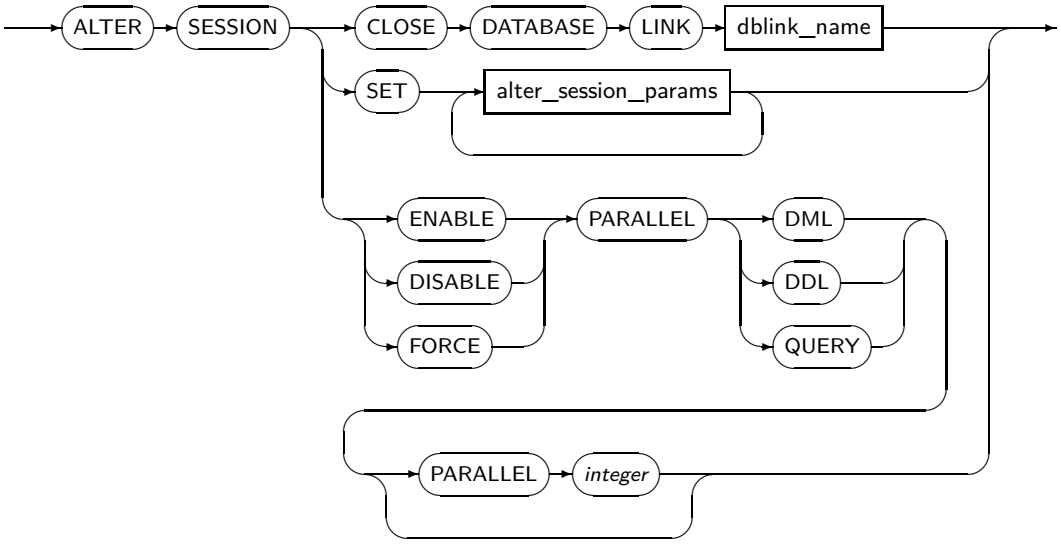
참고

현재 세션뿐만 아니라 전체 시스템의 환경을 조정하려면 "9.2. ALTER SYSTEM"의 내용을 참고한다.

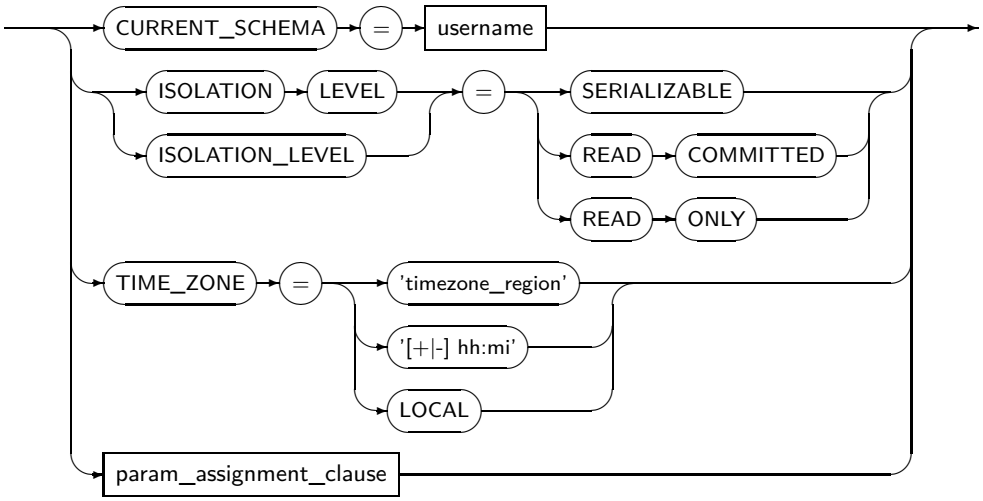
ALTER SESSION의 세부 내용은 다음과 같다.

- 문법

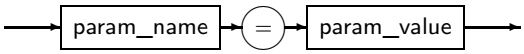
alter_session



alter_session_params



param_assignment_clause



- 특권

  아무런 특권을 요구하지 않는다.

- 구성요소

  – alter_session

구성요소	설명
CLOSE DATABASE LINK	지정한 데이터베이스 링크의 연결을 닫는다. 데이터베이스 링크를 열어 놓음으로써 발생하는 네트워크 리소스의 낭비를 줄이고 싶을 때 사용한다. CLOSE DATABASE LINK를 사용해 명시적으로 연결을 닫지 않으면 세션이 끊어질 때까지 연결이 열려 있다.
dblink_name	데이터베이스 링크의 이름을 명시한다.
SET alter_session_params	초기화 환경 변수나 세션 환경 변수의 값을 바꾸기 위해 사용한다. 변경한 값은 세션이 유지되는 동안 유효하다.

  – alter_session_params

구성요소	설명
CURRENT_SCHEMA	현재 세션의 스키마를 변경한다. 변경 이후 스키마가 지정되지 않은 객체는 변경한 스키마부터 찾아보게 된다. 세션의 스키마가 변경되었다고 해서 세션 사용자가 변경되는 것은 아니다. 즉, 객체를 찾는 데에는 스키마가 사용되지만 그 객체를 사용할 수 있는가를 검사할 때는 세션 사용자를 기준으로 한다.
ISOLATION LEVEL	현재 세션이 만드는 트랜잭션의 고립성 수준을 설정한다.

구성요소	설명
ISOLATION_LEVEL	설정할 수 있는 고립성 수준은 다음과 같다. – SERIALIZABLE – READ COMMITED (기본값) – READ ONLY
SERIALIZABLE	세션에 참가하는 트랜잭션은 SQL-92에 명시된 직렬화된 트랜잭션 고립성 모드 (Serializable Transaction Isolation Mode)로 동작한다. 이 모드에서는 직렬화된 트랜잭션이 시작되고 난 후 어떤 로우를 갱신한 다른 트랜잭션이 커밋했을 때 그 로우를 직렬화된 트랜잭션이 DML 문을 사용해 변경하려고 하면 변경이 일어나지 않고 에러가 반환된다.
READ COMMITED	이 모드에서는 어떤 트랜잭션이 로우를 변경하고 커밋하지 않았을 경우 그 로우에 현재 트랜잭션이 DML 문을 사용해 변경하려고 하면 그 로우에 설정된 잠금 때문에 대기하게 된다.
READ ONLY	기본적으로 SERIALIZABLE 모드와 똑같이 동작하지만, 데이터를 변경하려고 하면 에러가 반환된다.
TIME_ZONE	현재 세션의 시간대를 변경한다. '[+|-]hh:mi'은 시와 분으로 표현된 시간대 오프셋을 의미하며, -12:00~14:00까지의 값을 사용할 수 있다. LOCAL을 설정한 경우 처음 세션이 시작될 때의 시간대를 현재 세션의 시간대로 변경한다. 데이터베이스에서 지원하는 지역 이름을 알고 싶으면 VT_TIMEZONE_NAMES 테이블의 TZNAME 컬럼을 조회하면 된다.
param_assignment_clause	세션 초기화 파라미터를 설정한다. VT_PARAMETER 테이블 또는 VT_PARAMETER 동의어에서 파라미터를 확인할 수 있다. 이중 Dynamic, SESSION 클래스를 가지고 있는 초기화 파라미터만 ALTER SESSION을 사용하여 변경할 수 있다. Dynamic은 서버가 활성화된 상태에서 변경이 가능하다. 참고로 Static은 $TB_SID.tip 파일에 내용을 설정하기 때문에 서버를 재기동해야만 적용된다. SESSION은 세션별로 변경이 가능하다. SYSTEM은 세션별로 설정할 수 없고, 데이터베이스 전체에서 함께 사용하는 파라미터를 의미한다.

– param_assignment_clause

구성요소	설명
param_name	세션 초기화 파라미터의 이름을 명시한다.
param_value	세션 초기화 파라미터의 값을 명시한다.

- 예제

  다음은 **ALTER SESSION**을 사용해 세션 환경을 변경하는 예이다.

```
ALTER SESSION CLOSE DATABASE LINK remote;
ALTER SESSION SET CURRENT_SCHEMA = tibero;
ALTER SESSION SET ISOLATION_LEVEL = SERIALIZABLE;
ALTER SESSION SET CURSOR_SHARING = EXACT;
```

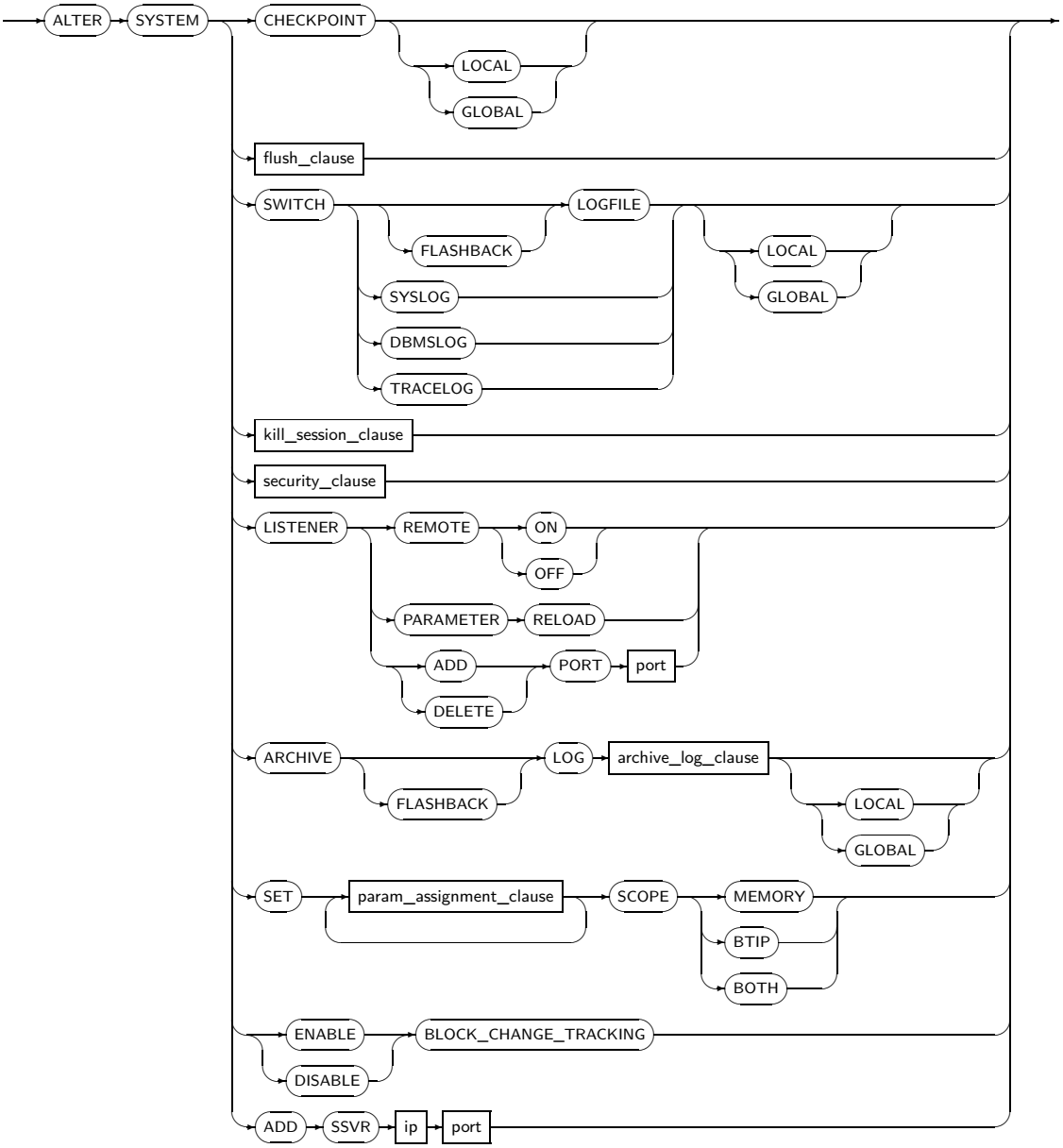
9.2. ALTER SYSTEM

체크포인트 작업을 수행하거나 진행 중인 세션을 중지하는 등의 시스템을 조작하거나, 초기화 파라미터를 동적으로 변경하는 등 시스템의 속성을 변경할 때 사용한다.

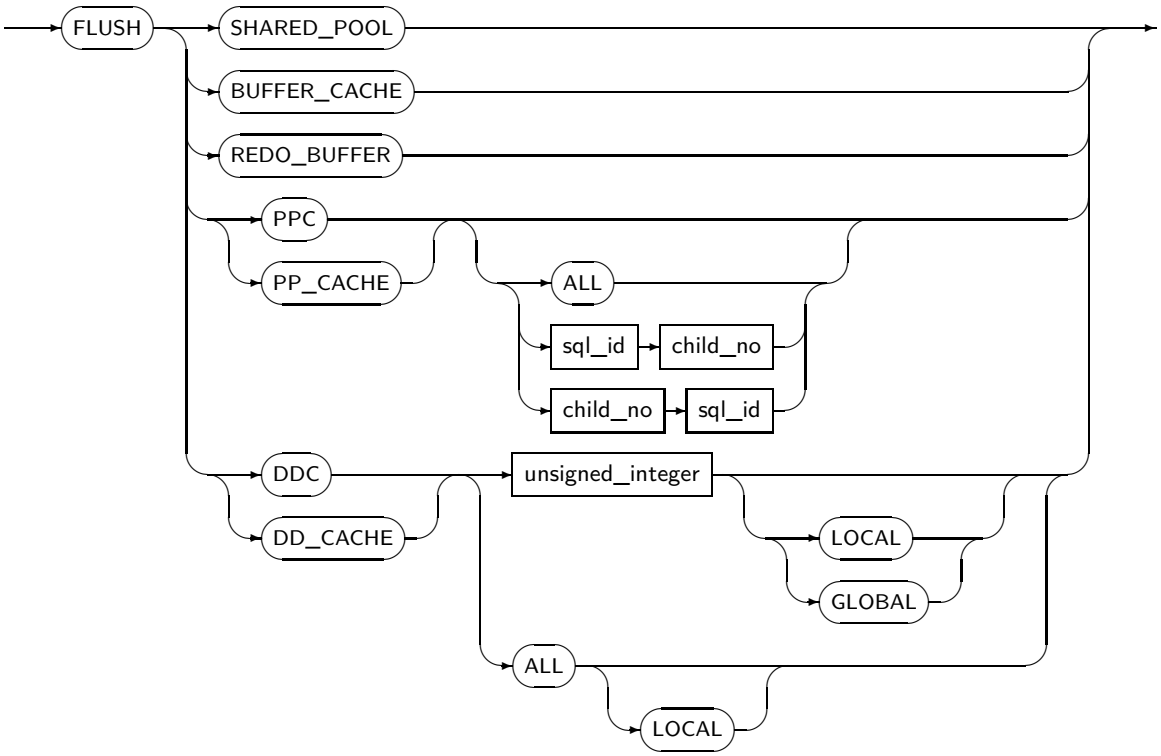
ALTER SYSTEM의 세부 내용은 다음과 같다.

- 문법

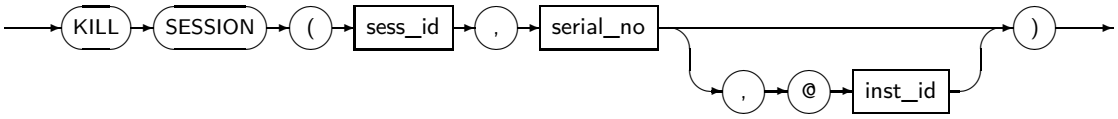
alter_system



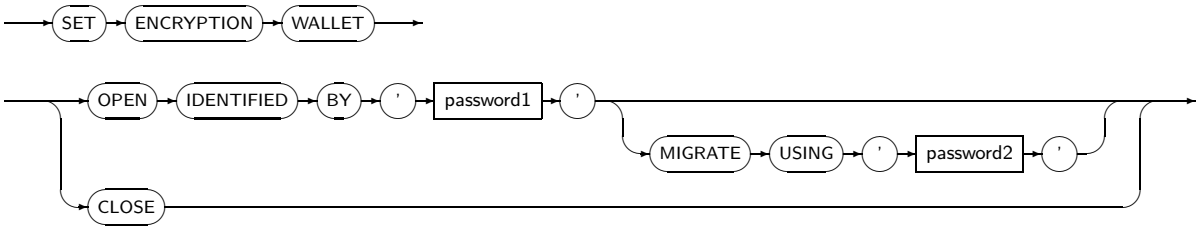
flush_clause



kill_session_clause



security_clause



- 특권

  ALTER SYSTEM 시스템 특권을 부여받아야 한다.

- 구성요소

  – alter_system

구성요소	설명
CHECKPOINT	Tibero가 체크포인트 작업을 수행하도록 한다.

구성요소	설명
	체크포인트 작업을 수행하면 버퍼 캐시의 모든 갱신된 디스크 블록이 디스크에 반영된다. 체크포인트는 트랜잭션의 커밋 여부와는 무관하다. 커밋되지 않은 트랜잭션이 변경한 디스크 블록도 디스크에 반영되며, 그렇다고 해도 다른 트랜잭션이 커밋되지 않은 블록을 볼 수는 없다.
CHECKPOINT LOCAL	Tibero TAC 환경에서 접속한 노드에 대한 체크포인트 작업을 수행하도록 한다.
CHECKPOINT GLOBAL	Tibero TAC 환경에서 전체 노드에 대한 체크포인트 작업을 수행하도록 한다.
FLUSH SHARED_POOL	공유 풀(shared pool)의 내용을 모두 비운다. 공유 풀에는 데이터 딕셔너리(data dictionary) 정보 등이 들어가며, 현재 사용 중인 정보는 비우지 않는다.
FLUSH BUFFER_CACHE	버퍼 캐시(buffer cache)의 더티 블록(dirty block)을 디스크에 기록한다. 이때 버퍼 캐시의 더티 블록을 디스크에 기록만 하고, 버퍼 캐시의 블록들을 무효화(invalidation)시키지는 않는다.
FLUSH REDO_BUFFER	Redo 로그를 강제로 flush한다. Redo 로그는 트랜잭션을 커밋하지 않더라도 주기적으로 flush되므로 따로 명령을 실행할 필요는 없으나, Redo 로그가 flush된 것을 명시적으로 나타내기 위해 명령을 실행할 수 있다.
FLUSH PPC FLUSH PPC ALL FLUSH PP_CACHE FLUSH PP_CACHE ALL	PP 캐시에서 모든 physical plan 정보를 강제로 flush한다.
FLUSH PPC sql_id child_no FLUSH PPC child_no sql_id FLUSH PP_CACHE sql_id child_no FLUSH PP_CACHE child_no sql_id	Physical plan의 sql id와 child number를 명시하면 PP 캐시에서 해당 physical plan 정보를 강제로 flush한다.
FLUSH DDC unsigned_integer	unsigned_integer에 스키마 객체 ID를 명시하면 DD 캐시에서 해당 스키마 객체 정보를 강제로 flush 한다. 시퀀스 정보는 flush하지 않는다.
FLUSH DDC unsigned_integer LOCAL	TAC 환경일 때 unsigned_integer에 스키마 객체 ID를 명시하면 접속한 노드의 DD 캐시에서 해당 스키마 객체 정보를 강제로 flush한다. 시퀀스 정보는 flush하지 않는다.

구성요소	설명
FLUSH DDC unsigned_integer GLOBAL	TAC 환경일 때 unsigned_integer에 스키마 객체 ID를 명시하면 전체 노드의 DD 캐시에서 해당 스키마 객체 정보를 강제로 flush한다. 시퀀스 정보는 flush하지 않는다.
FLUSH DDC ALL	DD 캐시에서 시퀀스를 제외한 모든 스키마 객체 정보를 강제로 flush 한다.
FLUSH DDC ALL LOCAL	TAC 환경일 때 접속한 노드의 DD 캐시에서 시퀀스를 제외한 모든 스키마 객체 정보를 강제로 flush한다.
FLUSH DD_CACHE unsigned_integer	FLUSH DDC unsigned_integer와 동일한 기능이다.
FLUSH DD_CACHE unsigned_integer LOCAL	FLUSH DDC unsigned_integer LOCAL과 동일한 기능이다.
FLUSH DD_CACHE unsigned_integer GLOBAL	FLUSH DDC unsigned_integer GLOBAL과 동일한 기능이다.
FLUSH DD_CACHE ALL	FLUSH DDC ALL과 동일한 기능이다.
FLUSH DD_CACHE ALL LOCAL	FLUSH DDC ALL LOCAL과 동일한 기능이다.
SWITCH LOGFILE	로그 그룹을 교체한다. 로그 버퍼의 모든 내용이 현재의 로그 그룹으로 저장되고, 이후 생성되는 Redo 로그는 다음의 로그 그룹에 저장된다. 로그 파일에 대한 자세한 내용은 "Tibero 관리자 안내서"를 참고한다.
SWITCH LOGFILE LOCAL	TAC 환경에서 접속한 노드에 대해서만 Redo 로그 그룹을 교체한다.
SWITCH LOGFILE GLOBAL	TAC 환경에서 전체 노드에 대해 Redo 로그 그룹을 교체한다.
SWITCH FLASHBACK LOGFILE	Flashback 로그 그룹을 교체한다. Flashback 로그 버퍼의 모든 내용이 현재의 로그 그룹으로 저장되고, 이후 생성되는 Flashback 로그는 다음의 로그 그룹에 저장된다. Flashback 로그 파일에 대한 자세한 내용은 "Tibero 관리자 안내서"를 참고한다.
SWITCH FLASHBACK LOGFILE LOCAL	TAC 환경에서 접속한 노드에 대해서만 Flashback 로그 그룹을 교체한다.
SWITCH FLASHBACK LOGFILE GLOBAL	TAC 환경에서 전체 노드에 대해 Flashback 로그 그룹을 교체한다.
SWITCH SYSLOG	syslog 파일을 교체한다. 현재 사용 중인 syslog 파일을 백업하고 새로운 syslog 파일을 생성한다.
SWITCH SYSLOG LOCAL	TAC 환경에서 접속한 노드에 대해 syslog 파일을 백업하고 새로운 syslog 파일을 생성한다.
SWITCH SYSLOG GLOBAL	TAC 환경에서 전체 노드에 대해 syslog 파일을 백업하고 새로운 syslog 파일을 생성한다.

구성요소	설명
SWITCH DBMSLOG	dbmslog 파일을 교체한다. 현재 사용 중인 dbmslog 파일을 백업하고 새로운 dbmslog 파일을 생성한다.
SWITCH DBMSLOG LOCAL	TAC 환경에서 접속한 노드에 대해 dbmslog 파일을 백업하고 새로운 dbmslog 파일을 생성한다.
SWITCH DBMSLOG GLOBAL	TAC 환경에서 전체 노드에 대해 dbmslog 파일을 백업하고 새로운 dbmslog 파일을 생성한다.
SWITCH TRACELOG	tracelog 파일을 교체한다. 현재 사용 중인 tracelog 파일을 백업하고 새로운 tracelog 파일을 생성한다.
SWITCH TRACELOG LOCAL	TAC 환경에서 접속한 노드에 대해 tracelog 파일을 백업하고 새로운 tracelog 파일을 생성한다.
SWITCH TRACELOG GLOBAL	TAC 환경에서 전체 노드에 대해 tracelog 파일을 백업하고 새로운 tracelog 파일을 생성한다.
kill_session_clause	세션을 중지시킨다. 트랜잭션 종료를 기다리지 않고, 마치 접속이 종료된 것처럼 처리한다.
security_clause	컬럼 암호화 등의 보안 기능에 사용하는 보안 지갑(Wallet)을 열고 닫는다.
LISTENER REMOTE ON	로컬 호스트가 아닌 외부 클라이언트의 네트워크 접속을 허용한다.
LISTENER REMOTE OFF	로컬 호스트가 아닌 외부 클라이언트의 네트워크 접속을 차단한다. 단, 로컬 호스트의 tbdsn.tbr 파일에서 IP가 'localhost'라고 설정된 경우에만 해당한다. 그리고 이미 접속된 클라이언트의 접속은 유지된다.
LISTENER PARAMETER RELOAD	서버 운영중에 LSNR_INVITED_IP 또는 LSNR_DENIED_IP의 설정을 변경한 경우 이를 적용한다.
LISTENER ADD PORT port	LISTENER_PORT 이외의 데이터베이스의 접속 포트 port를 추가한다.
LISTENER DELETE PORT port	LISTENER_PORT 이외의 추가된 데이터베이스의 접속 포트 port를 삭제한다.
ARCHIVE LOG archive_log_clause	Current가 아닌 모든 Redo 로그 그룹들에 대한 아카이브 로그를 생성한다.
ARCHIVE LOG archive_log_clause LOCAL	TAC 환경에서 접속한 노드에만 해당되는 Current가 아닌 모든 Redo 로그 그룹들에 대한 아카이브 로그를 생성한다.
ARCHIVE LOG archive_log_clause GLOBAL	TAC 환경에서 전체 노드에 해당되는 Current가 아닌 모든 Redo 로그 그룹들에 대한 아카이브 로그를 생성한다.
ARCHIVE FLASHBACK LOG archive_log_clause	Current가 아닌 모든 Flashback 로그 그룹들에 대한 아카이브 로그를 생성한다.
ARCHIVE FLASHBACK LOG archive_log_clause LOCAL	TAC 환경에서 접속한 노드에만 해당되는 Current가 아닌 모든 Flashback 로그 그룹들에 대한 아카이브 로그를 생성한다.

구성요소	설명
ARCHIVE FLASHBACK LOG archive_log_clause GLOBAL	TAC 환경에서 전체 노드에 해당되는 Current가 아닌 모든 Flashback 로그 그룹들에 대한 아카이브 로그를 생성한다.
SET param_assign ment_clause	시스템 초기화 파라미터를 설정한다. ALTER SESSION의 param_assignment_clause와 동일하지만, Dynamic, System 클래스를 가지고 있는 초기화 파라미터를 변경한다는 점이 다르다.
ENABLE BLOCK CHANGE TRACKING	Block Change Tracking 모드를 활성화한다. Block Change Tracking에 대한 자세한 내용은 "Tibero 관리자 안내서"를 참고한다.
DISABLE BLOCK CHANGE TRACKING	Block Change Tracking 모드를 비활성화한다. Block Change Tracking에 대한 자세한 내용은 "Tibero 관리자 안내서"를 참고한다.
ADD SSVR ip port	TAS 가동 중에 새로운 SSVR를 추가한다. 해당 ip 및 port에 맞는 SSVR가 추가되며 TAS 인스턴스에서 수행해야 한다.

– kill_session_clause

구성요소	설명
sess_id	접속을 끊을 세션의 ID이다. VT_SESSION을 통해 확인할 수 있으며, 해당 세션이 없는 경우 에러가 발생한다.
serial_no	접속을 끊을 세션의 일련번호이다. VT_SESSION을 통해 확인할 수 있으며, 해당 세션이 없는 경우 에러가 발생한다.
inst_id	접속을 끊을 세션의 인스턴스의 ID이다. TAC 환경에서 다른 노드의 세션의 접속을 끊을 때에만 입력한다. GV$SESSION을 통해 확인할 수 있으며, TAC 환경이 아니거나 해당 인스턴스가 없는 경우 에러가 발생한다.

– security_clause

구성요소	설명
OPEN IDENTIFIED BY 'pass word1'	보안 지갑을 열 때 사용한다. password1에는 보안 지갑의 패스워드를 입력한다. 패스워드는 보안 지갑을 생성할 때 입력한 것과 같아야 하며, 보안 지갑은 $TB_HOME/bin 디렉터리에 있는 tbwallet_gen 프로그램을 이용하여 생성한다.

구성요소	설명
OPEN IDENTIFIED BY 'password1' MIGRATE USING 'password2'	HSM 장비 사용 환경에서 보안 지갑을 열 때 사용한다. HSM 장비를 사용하는 경우 HSM 장비의 패스워드 password1과 보안 지갑의 패스워드 password2를 사용하여 보안 지갑을 열 수 있다.
CLOSE	보안 지갑을 닫을 때 사용한다.

– archive_log_clause

구성요소	설명
ALL	아직 아카이브 되지 않은 Redo 로그를 모두 아카이브한다. 현재 사용 중인 Redo 로그는 대상에서 제외된다. 로그 파일과 아카이브에 대한 자세한 내용은 "Tibero 관리자 안내서"를 참고한다.

- 예제

다음은 **ALTER SYSTEM**을 사용하여 시스템의 속성을 변경하는 예이다.

```
ALTER SYSTEM CHECKPOINT;
ALTER SYSTEM SWITCH LOGFILE;
ALTER SYSTEM KILL SESSION (1, 1);
ALTER SYSTEM KILL SESSION (1, 1, @1);
ALTER SYSTEM SET ENCRYPTION WALLET OPEN IDENTIFIED BY "password";
ALTER SYSTEM SET ENCRYPTION WALLET CLOSE;
ALTER SYSTEM SET DBMS_LOG_DEST = '/tmp/log';
```

다음은 **KILL SESSION**을 사용했을 때 해당 세션이 없는 경우 에러가 발생하는 예이다.

```
SQL> ALTER SYSTEM KILL SESSION (1, 1);
TBR-7204: Requested session may be closed already or not exist.
```

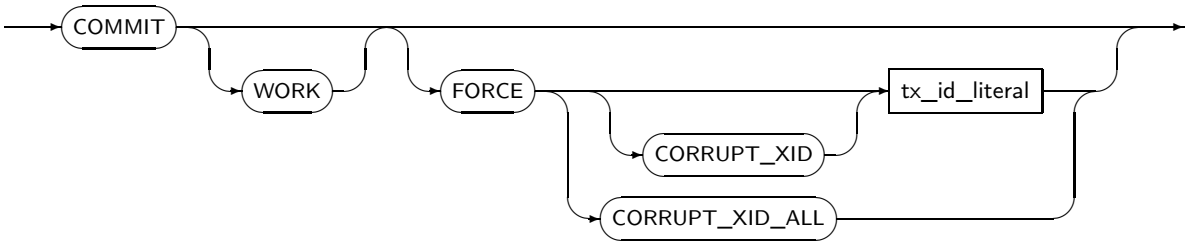
9.3. COMMIT

현재 트랜잭션을 종료하고 트랜잭션의 갱신된 내용을 데이터베이스에 반영한다. 동시에 모든 저장점을 삭제하고 로우에 설정된 잠금을 해제한다.

COMMIT의 세부 내용은 다음과 같다.

- 문법

commit



- 구성요소

구성요소	설명
WORK	SQL 표준과 호환성을 위해 허용된 문법이다. 생략해도 아무런 차이가 없다.
FORCE	특정 트랜잭션을 강제로 커밋할 때 사용한다.
CORRUPT_XID	트랜잭션 복구를 실패한 트랜잭션을 강제로 커밋한다. 트랜잭션 복구를 실패한 트랜잭션은 동적 뷰 V$CORRUPT_XID로 조회할 수 있다. 단, 강제로 커밋된 트랜잭션은 복구되지 않으므로 데이터 일관성이 보장되지 않는다.
CORRUPT_XID_ALL	트랜잭션 복구를 실패한 경우 실패한 모든 트랜잭션을 강제로 커밋한다. 단, 강제로 커밋된 트랜잭션은 복구되지 않으므로 데이터 일관성이 보장되지 않는다.
tx_id_literal	XA에서 지연된 트랜잭션을 처리할 때 사용한다. DBA_2PC_PENDING 뷰의 LOCAL_TRAN_ID 컬럼에 해당하는 값을 입력한다. 작은따옴표(' ')를 사용하는 것을 유의한다.

- 예제

다음은 **COMMIT**을 실행하는 예이다.

```
COMMIT;
COMMIT WORK;
COMMIT FORCE '2.16.18';
```

존재하지 않는 트랜잭션을 지정하면, 다음과 같은 에러가 발생한다.

```
SQL> COMMIT FORCE '2.16.18';
TBR-21022: No prepared transaction found with transaction id 2.16.18.
```

9.4. LOCK TABLE

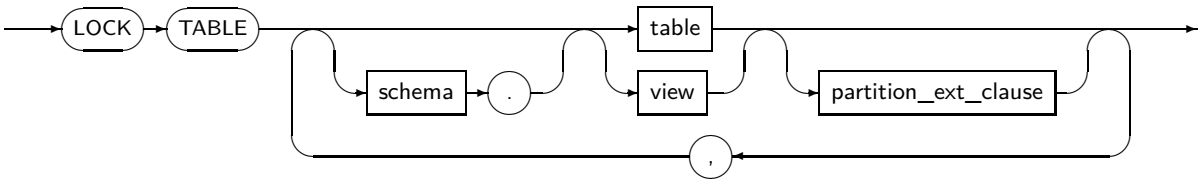
하나 이상의 테이블, 테이블 파티션, 테이블 서브파티션을 특정 모드로 잠근다. DML 수행 등으로 인한 자동 잠김보다 상위 모드로 적용되며, 작업 수행 중 다른 사용자의 테이블에 대한 접근을 금지하거나 허용할

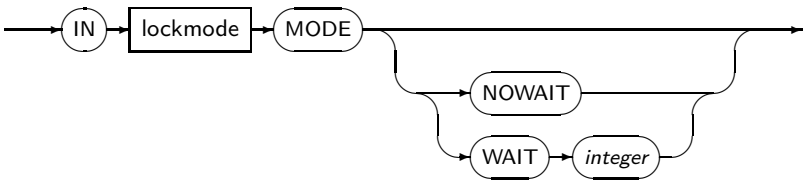
수 있다. 잠긴 테이블은 트랜잭션을 커밋하거나 롤백할 때까지 풀리지 않는다. 테이블을 조회하는 작업은 테이블 잠금을 필요로하지 않으므로, 테이블을 잠그더라도 다른 사용자의 테이블 조회를 막지 않는다.

LOCK TABLE의 세부 내용은 다음과 같다.

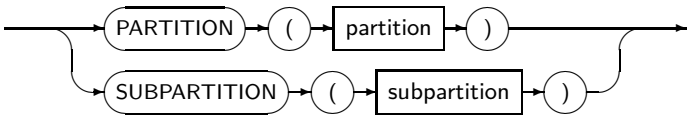
- 문법

lock_table

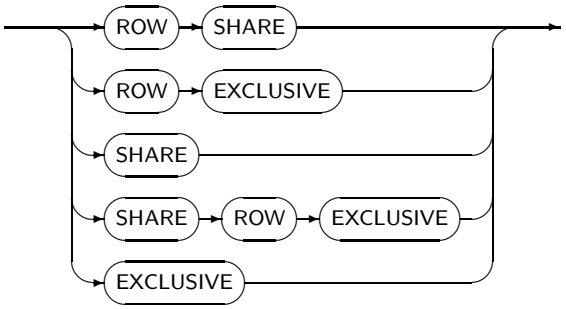




partition_ext_clause



lockmode



- 특권

  사용자의 스키마에 있는 테이블을 잠그기 위해서는 별다른 특권이 필요하지 않다. 단, 다른 사용자의 스키마에 있는 테이블을 잠그기 위해서는 LOCK ANY TABLE 시스템 특권이 있어야 한다.

- 구성요소

  – lock_table

구성요소	설명
schema	잠글 테이블을 포함하고 있는 스키마의 이름이다. 생략하면 현재 사용자의 스키마로 인식된다.

구성요소	설명
table, view	잠글 테이블 또는 뷰의 이름이다.
partition_ext_clause	잠글 파티션 또는 서브파티션을 지정한다.
lockmode MODE	잠글 모드를 설정한다.
NOWAIT	다른 사용자가 먼저 테이블을 잠그고 있어서 테이블을 잠글 수 없는 경우 기다리지 않고 곧바로 실패하도록 설정한다.
WAIT integer	다른 사용자가 먼저 테이블을 잠그고 있어서 테이블을 잠글 수 없는 경우 일정한 시간만큼 기다린다. 지정된 시간(초)이 초과하면 실패한다.

– partition_ext_clause

구성요소	설명
partition	잠글 테이블 파티션의 이름이다.
subpartition	잠글 테이블 서브파티션의 이름이다.

– lockmode

구성요소	설명
ROW SHARE	테이블에 대한 동시 접근을 허용하지만 배타적 접근을 허용하지 않는다.
ROW EXCLUSIVE	ROW SHARE와 같지만 SHARE 모드로 잠그는 것도 허용하지 않는다. UPDATE, INSERT, DELETE를 수행할 때 자동으로 설정되는 모드이다.
SHARE	동시 조회는 가능하지만 테이블 수정은 허용하지 않는다.
SHARE ROW EXCLUSIVE	전체 테이블을 조회할 때 사용된다. 다른 사용자가 테이블을 조회하는 것은 허용하지만 SHARE 모드로 테이블을 잠그거나 테이블을 수정하는 것은 허용하지 않는다.
EXCLUSIVE	테이블 조회는 가능하지만 테이블에 대한 다른 모든 동작은 허용하지 않는다.

- 예제

  다음은 **LOCK TABLE**을 사용해 테이블을 잠그는 예이다.

```
LOCK TABLE table_exmp IN EXCLUSIVE MODE NOWAIT;

LOCK TABLE table_exmp PARTITION (part_exmp1) IN SHARE MODE WAIT 3;
```

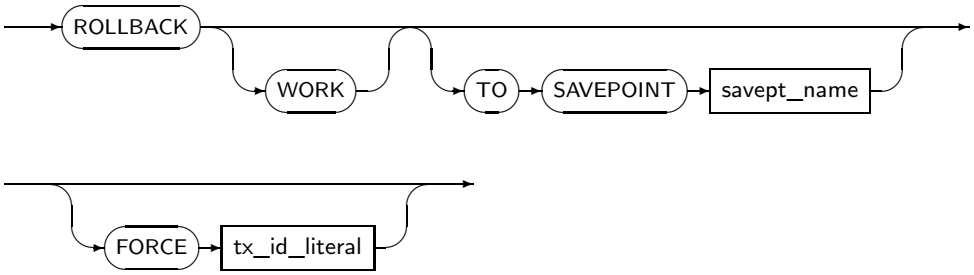
9.5. ROLLBACK

현재 트랜잭션을 종료하고 트랜잭션에서 갱신된 내용을 모두 취소한다. 동시에, 모든 저장점을 삭제하고 로우에 설정된 잠금을 해제한다.

ROLLBACK의 세부 내용은 다음과 같다.

- 문법

rollback



- 구성요소

구성요소	설명
WORK	SQL 표준과 호환성을 위해 허용된 문법이다. 생략해도 아무런 차이가 없다.
SAVEPOINT	특정 저장점까지만 롤백을 하고자 할 때 사용한다. 지정되지 않은 저장점을 명시하면, 에러가 발생한다. 저장점의 이름은 대소문자를 구분하지 않는다.
savept_name	저장점의 이름을 명시한다.
FORCE	특정 트랜잭션을 강제로 롤백할 때 사용한다.
tx_id_literal	XA에서 지연된 트랜잭션을 처리할 때 사용한다. DBA_2PC_PENDING 뷰의 LOCAL_TRAN_ID 컬럼에 해당하는 값을 입력한다. 작은따옴표(' ')를 사용하는 것을 유의한다.

- 예제

다음은 **SAVEPOINT**로 저장점을 지정하고, **ROLLBACK**을 사용해 지정된 저장점까지 롤백하는 예이다.

```
SQL> INSERT INTO T VALUES(1);

1 row inserted.

SQL> COMMIT;
```

```
Commit completed.

SQL> SELECT * FROM T;
        A
---------
        1
1 selected.

SQL> UPDATE T SET A=2;
1 updated.

SQL> SAVEPOINT SP1;
Savepoint created.

SQL> UPDATE T SET A=3;
1 updated.

SQL> SELECT * FROM T;
        A
---------
        3
1 selected.

SQL> ROLLBACK TO SAVEPOINT SP1;
Rollback succeeded.

SQL> SELECT * FROM T;
        A
---------
        2
1 selected.

SQL> ROLLBACK;
Rollback succeeded.

SQL> SELECT * FROM T;
        A
---------
        1
1 selected.

SQL> ROLLBACK TO SAVEPOINT SP1;
TBR-21008 :  No such savepoint: 'SP1'
```

다음은 존재하지 않는 트랜잭션을 지정했을 때 발생하는 에러이다.

```
SQL> ROLLBACK FORCE '2.16.18';
TBR-21022: No prepared transaction found with transaction id 2.16.18.
```

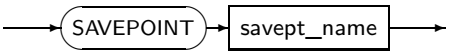
9.6. SAVEPOINT

현재 트랜잭션에 저장점을 설정한다. 부분 롤백을 수행하기 위해서는 반드시 저장점을 미리 설정해야 한다. 저장점 설정은 트랜잭션의 실행에 전혀 영향을 주지 않는다.

SAVEPOINT의 세부 내용은 다음과 같다.

- 문법

savepoint



- 구성요소

구성요소	설명
savept_name	저장점의 이름은 현재 트랜잭션에서 유일해야 한다. 만약 같은 트랜잭션에서 이전에 설정한 저장점의 이름과 동일한 저장점을 설정하면, 이전의 저장점은 제거된다. 저장점의 이름은 대소문자를 구분하지 않는다.

- 예제

다음은 **SAVEPOINT**를 사용해 저장점을 설정하는 예이다.

```
SAVEPOINT sp1;
```

9.7. SET ROLE

사용자에게 할당된 역할을 활성화하거나 비활성화한다.

참고

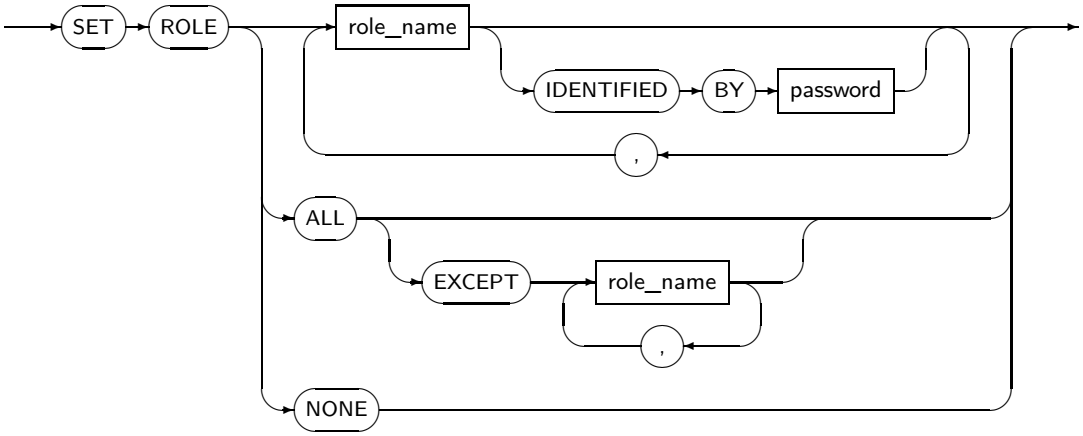
1. 역할을 생성, 변경, 제거하기 위해서는 “7.38. CREATE ROLE”, “7.11. ALTER ROLE”, “7.59. DROP ROLE”의 내용을 참고한다.

2. 역할에 특권을 부여하거나 회수하기 위해서는 “7.71. GRANT”, “7.75. REVOKE”의 내용을 참고한다.

SET ROLE의 세부 내용은 다음과 같다.

- 문법

set_role



- 특권

활성화 또는 비활성화하려는 역할은 해당 사용자에게 이미 사용이 허가된 역할이어야 한다.

- 구성요소

구성요소	설명
role_name	부여받은 역할 중에서 활성화하려는 역할을 나열한다. 이곳에 포함되지 않은 역할은 자동으로 비활성화된다.
IDENTIFIED BY	패스워드가 설정된 경우에는 해당 역할을 사용하려고 할 때 해당 패스워드를 입력해야 한다. 역할에 패스워드를 설정하는 방법은 ALTER ROLE과 CREATE ROLE을 참고한다.
ALL	부여받은 모든 역할을 사용할 수 있도록 설정한다. 단, 사용할 수 있는 역할 중에 패스워드가 설정된 역할이 있으면 해당 역할을 EXCEPT 절로 제외하지 않는 이상 ALL 절을 이용해 사용할 수 있게 설정할 수 없다.
EXCEPT	ALL 절을 사용해 부여받은 모든 역할을 사용할 수 있도록 설정하려고 할 때 설정에서 제외할 역할이 포함된다. 즉, EXCEPT 절에 나열한 역할을 제외하고는 다른 모든 역할을 사용할 수 있게 설정된다.

- 예제

다음은 사용자에게 역할을 할당하고, 할당된 역할을 **SET ROLE**을 사용해 활성화하는 예이다.

```
SQL> CONN sys/tibero
Connected.

SQL> CREATE USER u1 IDENTIFIED BY xxx; -- u1 사용자를 생성한다.
User created.

SQL> GRANT CREATE SESSION TO u1;
```

```
Granted.

SQL> CREATE ROLE a; -- 역할을 생성한다.
Role created.

SQL> CREATE ROLE b;
Role created.

SQL> CREATE ROLE c;
Role created.

SQL> CREATE ROLE d IDENTIFIED BY aaa; --일부 역할은 패스워드를 사용한다.
Role created.

SQL> CREATE ROLE e IDENTIFIED BY bbb;
Role created.

SQL> GRANT a, b, c, d, e TO u1; -- U1 사용자에게 역할을 허가한다.
Granted.

SQL> CONN u1/xxx
Connected.

SQL> SET ROLE a, b, c;
Set.

SQL> SELECT * FROM session_roles;
ROLE
------------------------------
A
B
C

3 rows selected.

SQL> SET ROLE a, b, c, d;
TBR-7181: need password to enable the role

SQL> SET ROLE c, d IDENTIFIED BY aaa, e IDENTIFIED BY bbb;
Set.

SQL> SELECT * FROM session_roles;
ROLE
------------------------------
C
D
E
```

```
3 rows selected
```

위의 예를 보면, 역할 A, B, C, D, E를 생성하는데, 그 중 역할 D와 역할 E는 패스워드를 설정하여 생성하였다. 패스워드가 설정되지 않은 역할 A, B, C는 아무런 제약 없이 사용할 수 있지만, 패스워드가 설정된 역할 D와 E는 IDENTIFIED BY 절을 사용하여 패스워드를 입력해야만 사용할 수 있다. 또한, 마지막 SET ROLE 문장을 보면 역할 A, B를 제외한 C, D, E만 사용할 수 있게 설정했으므로, 기존에 사용할 수 있게 설정되었던 역할 A, B가 사용할 수 없는 상태로 설정이 변경된 것을 볼 수 있다.

다음은 **EXCEPT**를 사용하는 예이다.

```
SQL> SET ROLE ALL;
TBR-7181: need password to enable the role

SQL> SET ROLE ALL EXCEPT d, e;
Set.

SQL> SELECT * FROM session_roles;
ROLE
------------------------------
A
B
C

3 rows selected.
```

위의 예에서, 역할 D, E는 패스워드가 설정되어 있다. ALL 절에서는 패스워드를 입력할 방법이 없으므로, ALL 절을 이용해서는 역할 D, E를 사용할 수 있게 설정할 수 없다. 패스워드를 사용하는 역할을 EXCEPT 절을 이용하여 제외하면, 제외되고 남은 나머지 역할을 ALL 절을 이용하여 사용할 수 있게 설정할 수 있다.

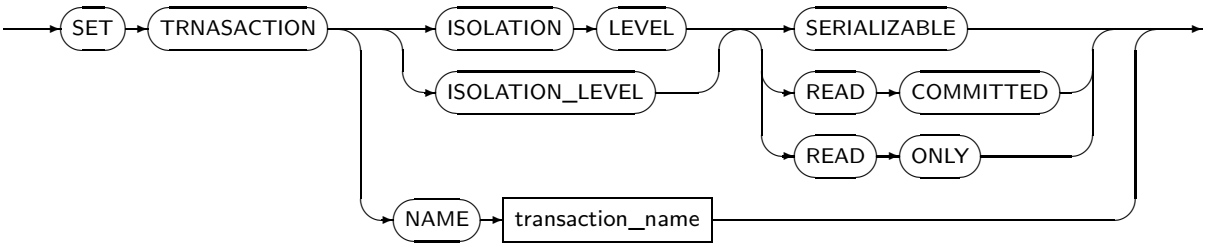
9.8. SET TRANSACTION

현재 트랜잭션의 고립성 수준이나 이름을 설정한다. 이 명령어는 현재 트랜잭션에서 실행하는 최초의 문장이어야 한다.

SET TRANSACTION의 세부 내용은 다음과 같다.

- 문법

set_transaction



- 구성요소

구성요소	설명
ISOLATION LEVEL 또는 ISOLATION_LEVEL	ISOLATION LEVEL 문으로 트랜잭션의 고립성을 설정할 수 있다. 설정할 수 있는 고립성 수준은 다음과 같다. – SERIALIZABLE – READ COMMITED (기본값) – READ ONLY
SERIALIZABLE	세션에 참가하는 트랜잭션은 SQL-92에 명시된 직렬화된 트랜잭션 고립성 모드 (Serializable Transaction Isolation Mode)로 동작한다. 이 모드에서는, 직렬화된 트랜잭션이 시작되고 난 후 어떤 로우를 갱신한 다른 트랜잭션이 커밋했을 때 그 로우를 직렬화된 트랜잭션이 DML 문을 사용해 변경하려고 하면 변경이 일어나지 않고 에러가 반환된다.
READ COMMITED	이 모드에서는 어떤 트랜잭션이 로우를 변경하고 커밋하지 않았을 경우 그 로우에 현재 트랜잭션이 DML 문을 사용해 변경하려고 하면 그 로우에 설정된 잠금 때문에 대기하게 된다.
READ ONLY	기본적으로 SERIALIZABLE 모드와 똑같이 동작하지만, 데이터를 변경하려고 하면 에러가 반환된다.
NAME	NAME 문으로 트랜잭션의 이름을 설정할 수 있다. (기본값: NULL) 트랜잭션 이름은 V$TRANSACTION을 조회해 보면 알 수 있다.
transaction_name	설정할 트랜잭션의 이름이다.

- 예제

다음은 현재 트랜잭션을 **READ COMMITTED**로 설정한 후 동적 뷰 **V$TRANSACTION**을 조회하는 예이다.

```
SQL> SET TRANSACTION ISOLATION_LEVEL READ COMMITTED;
Set.
```

```
SQL> SELECT name, state FROM v$transaction;

NAME                      STATE
------------------- ----------
                             3

1 row selected.

SQL> SET TRANSACTION NAME 'tx1';
TBR-7191 : Unable to execute SET TRANSACTION: transaction has already started.
```

위와 같이 SET TRANSACTION 문은 트랜잭션 최초의 문장이어야 하며, 그렇지 않은 경우 에러가 발생한다.

다음과 같이 트랜잭션의 이름을 설정할 수도 있다.

```
SQL> SELECT name, state FROM v$transaction;

NAME                      STATE
------------------- ----------

0 row selected.

SQL> SET TRANSACTION NAME 'tx1';
Set.

SQL> SELECT name, state FROM v$transaction;

NAME                      STATE
------------------- ----------
tx1                          3

1 row selected.
```

Appendix A. 예약어

본 장에서는 Tibero에서 사용하는 예약어를 알파벳 순으로 기술한다.

A.1. A

```
ACCESS
ADD
ALL
ALTER
AND
ANY
AS
ASC
AUDIT
```

A.2. B

```
BETWEEN
BY
```

A.3. C

```
CHAR
CHECK
CLUSTER
COLUMN
COMMENT
COMPRESS
CREATE
CURRENT
```

A.4. D

```
DATE
DECIMAL
DEFAULT
DELETE
```

```
DESC
DISTINCT
DROP
```

A.5. E

```
ELSE
EXCEPT
EXCLUSIVE
EXISTS
```

A.6. F

```
FILE
FLOAT
FOR
FOREIGN
FROM
```

A.7. G

```
GRANT
GROUP
```

A.8. H

```
HAVING
```

A.9. I

```
IDENTIFIED
IMMEDIATE
IN
INDEX
INDEXES
INITIAL
INSERT
INTEGER
INTERSECT
INTO
```

```
INVALIDATE
IS
```

A.10. L

```
LEVEL
LESS
LIKE
LOCK
LONG
```

A.11. M

```
MAXEXTENTS
MINUS
MODE
MODIFY
```

A.12. N

```
NOAUDIT
NOCOMPRESS
NOT
NOWAIT
NULL
NUMBER
```

A.13. O

```
OF
OFFLINE
ON
ONLINE
OPTION
OR
ORDER
```

A.14. P

```
PRIMARY
PRIOR
```

```
PRIVILEGES
PUBLIC
```

A.15. R

```
RAW
RENAME
REVOKE
ROW
ROWID
ROWNUM
ROWS
```

A.16. S

```
SELECT
SESSION
SET
SHARE
SIZE
SMALLINT
START
SUCCESSFUL
SYNONYM
SYSDATE
```

A.17. T

```
TABLE
THAN
THEN
TO
TRIGGER
```

A.18. U

```
UID
UNION
UNIQUE
```

```
UPDATE
USER
```

A.19. V

```
VALIDATE
VALUES
VARCHAR
VARCHAR2
VIEW
```

A.20. W

```
WHENEVER
WHERE
WITH
```

색인

Symbols

A

B

C

D

E

F

G

H

I

J

O

P

Q

R

S

T

U

V

W

X

ㄱ

ㄴ

ㄷ

ㄹ

ㅁ

O

ㅈ

ㅋ

ㅌ

표

함