

**ALTIBASE Application Development**

# **SQL User's Manual**

**Release 5.1.5**



---

ALTIbase Application Development SQL User's Manual  
Release 5.1.5  
Copyright © 2001~2009 ALTIbase Corp. All Rights Reserved.  
본 문서의 저작권은 알티베이스에 있습니다. 이 문서에 대하여 당사의 동의 없이  
무단으로 복제 또는 전용할 수 없습니다.

알티베이스  
152-790 서울시 구로구 구로동 182-13 대륭포스트타워 II 10 층  
전화: 02-2082-1114 팩스: 02-2082-1099  
e-mail: [support@altibase.com](mailto:support@altibase.com) homepage: <http://www.altibase.com>

---

# 목 차

<b>서문</b> .....	i
이 매뉴얼에 대하여 .....	ii
<b>Part I</b> .....	<b>1</b>
<b>1. 알티베이스 SQL 소개</b> .....	<b>3</b>
SQL 개요 .....	4
SQL 분류 .....	9
<b>2. 자료형</b> .....	<b>13</b>
자료형의 개요 .....	14
문자형 데이터 타입 .....	19
숫자형 데이터 타입 .....	22
날짜형 데이터 타입 .....	35
이진 데이터 타입 .....	48
LOB 데이터 타입 .....	52
공간 데이터 타입 .....	56
<b>Part II</b> .....	<b>57</b>
<b>3. 데이터 정의어</b> .....	<b>58</b>
ALTER DATABASE .....	59
ALTER INDEX .....	65
ALTER REPLICATION .....	67
ALTER SEQUENCE .....	72
ALTER TABLE .....	75
ALTER TABLESPACE .....	91
ALTER TRIGGER .....	99
ALTER USER .....	101
ALTER VIEW .....	103

CREATE DATABASE	105
CREATE DATABASE LINK	107
CREATE DIRECTORY	112
CREATE INDEX	114
CREATE QUEUE	122
CREATE REPLICATION	124
CREATE SEQUENCE	128
CREATE SYNONYM	136
CREATE TABLE	140
CREATE DISK TABLESPACE	140
CREATE MEMORY TABLESPACE	166
CREATE VOLATILE TABLESPACE	171
CREATE TEMPORARY TABLESPACE	175
CREATE TRIGGER	177
CREATE USER	184
CREATE VIEW	187
DROP DATABASE	191
DROP DATABASE LINK	193
DROP DIRECTORY	195
DROP INDEX	196
DROP QUEUE	197
DROP REPLICATION	198
DROP SEQUENCE	199
DROP SYNONYM	200
DROP TABLE	202
DROP TABLESPACE	204
DROP TRIGGER	207
DROP USER	208
DROP VIEW	210
GRANT	211
RENAME TABLE	227
REVOKE	229
TRUNCATE TABLE	233
<b>4. 데이터 조작어</b>	<b>235</b>

DELETE .....	236
INSERT.....	241
LOCK TABLE.....	245
SELECT.....	250
UPDATE .....	285
MOVE .....	291
ENQUEUE.....	294
DEQUEUE.....	295
<b>5. 데이터 제어어.....</b>	<b>297</b>
ALTER SESSION.....	298
ALTER SYSTEM .....	301
COMMIT.....	303
SAVEPOINT .....	304
ROLLBACK .....	306
SET TRANSACTION .....	309
<b>Part III .....</b>	<b>313</b>
<b>6. 집합 연산자 .....</b>	<b>315</b>
합집합 (UNION) .....	316
합집합 (UNION ALL) .....	317
교집합.....	318
차집합.....	319
우선순위 규칙 .....	320
<b>7. 함수.....</b>	<b>323</b>
함수의 종류 .....	324
그룹 함수.....	325
숫자 함수.....	329
문자 함수.....	344
날짜 함수.....	362
변환 함수.....	372
기타 함수.....	379
암호화 함수 .....	390
<b>8. 산술 연산자 .....</b>	<b>395</b>

종류	396
단항 연산자	397
사칙 연산자	398
연결 연산자	401
변환 연산자	402
<b>9. 조건 연산자</b>	<b>403</b>
종류	404
논리 연산자	405
비교조건	408
단항 비교	410
<b>Part IV</b>	<b>419</b>
<b>A. 부록 : 스키마</b>	<b>421</b>
예제 테이블 정보	422
E-R 다이어그램과 샘플 데이터	425
Altibase 객체들의 최대값	432
<b>찾아보기</b>	<b>433</b>

# 서문

## 이 매뉴얼에 대하여

이 매뉴얼은 데이터 베이스에서 사용되는 SQL(Structured Query Language)의 사용법에 대해 설명한다.

## 대상 사용자

이 매뉴얼은 다음과 같은 알티베이스 사용자를 대상으로 작성되었다.

- 데이터베이스 관리자
- 성능 관리자
- 데이터베이스 사용자
- 응용 프로그램 개발자
- 기술지원부

다음과 같은 배경 지식을 가지고 이 매뉴얼을 읽는 것이 좋다.

- 컴퓨터, 운영 체제 및 운영 체제 유ти리티 운용에 필요한 기본 지식
- 관계형 데이터베이스 사용 경험 또는 데이터베이스 개념에 대한 이해
- 컴퓨터 프로그래밍 경험
- 데이터베이스 서버 관리, 운영 체제 관리 또는 네트워크 관리 경험

## 소프트웨어 환경

이 매뉴얼은 데이터베이스 서버로 알티베이스 버전 5.1.5 을 사용한다는 가정 하에 작성되었다.

## 이 매뉴얼의 구성

이 매뉴얼은 다음과 같이 구성되어 있다.

- 제 1장 알티베이스 SQL 소개  
이 장은 알티베이스를 사용하기 위한 SQL의 개요, 분류, 구조에 대해서 설명한다.
- 제 2장 자료형  
이 장은 알티베이스에서 제공하는 데이터 타입에 대해서 설명한다.

- 제 3장 데이터 정의어  
이 장은 알티베이스에서 사용되는 데이터 정의어(DDL)에 대해서 각각 설명한다.
- 제 4장 데이터 조작어  
이 장은 알티베이스에서 사용되는 데이터 조작어(DML)에 대해서 각각 설명한다.
- 제 5장 데이터 제어어  
이 장은 알티베이스에서 사용되는 데이터 제어어(DCL)에 대해서 각각 설명한다.
- 제 6장 집합 연산자  
이 장은 알티베이스의 SQL에서 사용되는 집합 연산자에 대해서 설명한다.
- 제 7장 함수  
이 장은 알티베이스에서 제공하는 함수들에 대해서 설명한다.
- 제 8장 산술 연산자  
이 장은 알티베이스의 SQL에서 사용되는 산술 연산자에 대해서 설명한다.
- 제 9장 조건 연산자  
이 장은 알티베이스의 SQL문에서 조건절에 사용할 수 있는 조건 연산자에 대해서 설명한다.
- A. 부록 : 스키마  
이 장은 예제 테이블 정보와 ER다이어그램과 샘플데이터에 대해서 설명한다.

## 문서화 규칙

이 절에서는 이 매뉴얼에서 사용하는 규칙에 대해 설명한다. 이 규칙을 이해하면 이 매뉴얼과 설명서 세트의 다른 매뉴얼에서 정보를 쉽게 찾을 수 있다.

여기서 설명하는 규칙은 다음과 같다.

- 구문 다이어그램
- 샘플 코드 규칙

## 구문 다이어그램

이 매뉴얼에서는 다음 구성 요소로 구축된 다이어그램을 사용하여, 명령문의 구문을 설명한다.

구성 요소	의미
예약어	명령문이 시작한다. 완전한 명령문이 아닌 구문 요소는 화살표로 시작한다.

	명령문이 다음 라인에 계속된다. 완전한 명령문이 아닌 구문 요소는 이 기호로 종료한다.
	명령문이 이전 라인으로부터 계속된다. 완전한 명령문이 아닌 구문 요소는 이 기호로 시작한다.
	명령문이 종료한다.
	필수 항목
	선택적 항목
	선택사항이 있는 필수 항목. 한 항목만 제공해야 한다.
	선택사항이 있는 선택적 항목.
	선택적 항목. 여러 항목이 허용된다. 각 반복 앞부분에 콤마가 와야 한다.

## 샘플 코드 규칙

코드 예제는 SQL, Stored Procedure, iSQL 또는 다른 명령 라인 구문들을 예를 들어 설명한다.

아래 테이블은 코드 예제에서 사용된 인쇄 규칙에 대해 설명한다.

규칙	의미	예제
[ ]	선택 항목을 표시	VARCHAR [(size)] [[FIXED  ] VARIABLE]
{ }	필수 항목 표시. 반드시 하나 이상을 선택해야 되는 표시	{ ENABLE   DISABLE   COMPILE }
	선택 또는 필수 항목 표시의 인자 구분 표시	{ ENABLE   DISABLE   COMPILE } [ ENABLE   DISABLE   COMPILE ]
.	그 이전 인자의 반복 표시 예제 코드들의 생략되는 것을 표시	SQL> SELECT ename FROM employee;

.		ENAME ----- --- SWNO HJNO HSCHOI . . . 20 rows selected.
그 밖에 기호	위에서 보여진 기호 이 외에 기호들	EXEC :p1 := 1; acc NUMBER(11,2);
기울임 꼴	구문 요소에서 사용자가 지정해야 하는 변수, 특수한 값을 제공해야만 하는 위치 지정자	SELECT * FROM <i>table_name</i> ; CONNECT <i>userID/password</i> ;
소문자	사용자가 제공하는 프로그램의 요소들, 예를 들어 테이블 이름, 칼럼 이름, 파일 이름 등	SELECT ename FROM employee;
대문자	시스템에서 제공하는 요소들 또는 구문에 나타나는 키워드	DESC SYSTEM_.SYS_INDICES_;

---

## 관련 자료

자세한 정보를 위하여 다음 문서 목록을 참조하기 바란다.

- ALTIBASE Administration Installation User's Manual
- ALTIBASE Administration Administrator's Manual
- ALTIBASE Administration Replication User's Manual
- ALTIBASE Application Development Precompiler User's Manual
- ALTIBASE Application Development ODBC User's Manual
- ALTIBASE Application Development Application Program Interface User's Manual
- ALTIBASE Tools iSQL User's Manual
- ALTIBASE Tools Utilities User's Manual
- ALTIBASE Message Error Message Reference

---

## 온라인 매뉴얼

알티베이스 다운로드 센터(<http://adc.altibase.com/>)에서 국문 및 영문 매뉴얼(PDF, HTML)을 받을 수 있다.

---

## 알티베이스는 여러분의 의견을 환영합니다.

이 매뉴얼에 대한 여러분의 의견을 보내주시기 바랍니다. 사용자의 의견은 다음 버전의 매뉴얼을 작성하는데 많은 도움이 됩니다.

보내실 때에는 아래 내용과 함께

기술지원센터([support@altibase.com](mailto:support@altibase.com))로 보내주시기 바랍니다.

- 사용 중인 매뉴얼의 이름과 버전
- 매뉴얼에 대한 의견
- 사용자의 성함, 주소, 전화번호

이 외에도 알티베이스 기술지원 설명서의 오류와 누락된 부분 및 기타 기술적인 문제들에 대해서 이 주소로 보내주시면 정성껏 처리하겠습니다. 기술적인 부분과 관련하여 즉각적인 도움이 필요한 경우에는 기술지원센터로 연락하시기 바랍니다.

여러분의 의견에 항상 감사드립니다.

# **Part I**



# 1. 알티베이스 SQL 소개

이 장에서는 알티베이스 SQL의 특징과 구성에 대해서 간략하게 소개하고 있다.

## SQL 개요

SQL(Structured Query Language)은 데이터베이스로 저장되는 데이터를 조작하고, 관리하며, 검색하기 위한 언어이다.

## 알티베이스 SQL 특징

### 빠른 질의 수행

알티베이스의 경우 대부분의 질의들이 SQL 준비(PREPARE)과정 후 실행하기 전 시스템 카탈로그 정보가 변하지 않는 점에 차안 해 준비과정 시점에 최적화된 실행 계획 트리(execution plan tree)를 미리 만들어 두고 여러 번 실행할 수 있도록 하여 실제 질의 실행 시 수행속도를 대폭 향상시켰다. 이와 같은 방법은 초기 데이터베이스 스키마 생성 후 데이터 정의어(DDL) 수행은 거의 일어나지 않고 데이터 조작어(DML) 작업이 빈번한 응용 어플리케이션에 매우 유용하다.

### SQL/92 표준 지원

알티베이스 SQL은 SQL/92 표준 사양을 지원하므로 기존 SQL 사용자들은 별도의 숙지 없이 알티베이스를 쉽게 사용할 수 있다.

### 강력한 부연질의(subquery) 기능 지원

일반적으로 부연질의는 수식의 일부, IN, CREATE TABLE, INSERT 등에서 주로 사용된다. 여기에 사용되는 부연질의는 대부분 여러 칼럼의 여러 레코드를 검색해 주는 기능이다.

알티베이스의 경우 부연질의의 결과가 한 레코드, 한 칼럼일 경우 데이터가 위치할 수 있는 임의의 위치에 치환해 사용할 수 있으며, 빠른 검색 속도의 부연질의를 지원하므로 여러 SQL 문으로 처리 가능한 질의문을 한 SQL 문으로 처리할 수 있다. 따라서 복잡한 응용 어플리케이션에서 유용하게 사용 가능하다.

### 다양한 시스템 제공 함수 지원

SQL/92의 표준 사양 이외에 사용자가 유용하게 사용할 수 있는 다양한 시스템 함수를 지원한다.

## SQL의 수행

알티베이스에서 SQL 문의 처리 과정은 크게 준비(PREPARE)과정과 실행(EXECUTE)과정으로 나뉘어진다.

준비과정은 SQL 문의 문법을 분석하여 정당성을 검사하고 최적화 한 후 실행 계획을 수립해 실행 계획 트리(execution plan tree)를 생성하는 과정이다. 이 과정에서 메타 테이블에 접근해 테이블 정보, 인덱스 정보 등을 읽고 최적화된 접근 계획을 수립한다. 따라서 준비와 실행이 한번에 이루어지지 않고 나뉘어져 수행된다면 준비 후 메타의 변경 사항이 일어나지 않아야 준비(PREPARE)과정시에 생성한 실행 계획 트리를 수정 없이 그대로 사용할 수 있다. 예를 들어, 준비과정시 존재했던 인덱스가 실행과정에 실제로 존재하지 않는다면 준비과정시 인덱스를 사용하는 환경에서 최적화한 실행 계획 트리는 실행과정시 무효한 실행 계획이 되어 사용할 수 없게 된다.

실행과정은 준비과정에서 생성된 실행 계획에 따라 실제 질의문을 수행하는 과정이다.

한 SQL 문이 호스트 변수를 사용하여 여러개의 값으로 변경하여 여러번 수행하는 경우 준비과정은 한번 수행되고 변수값 설정과 실행과정은 여러번 수행된다.

## 주석

알티베이스는 SQL 문의 임의의 위치에 다음 두 가지 형식의 주석을 사용할 수 있다.

- `/* */`  
C언어에서 사용한 주석 형식과 동일한 방식으로 주석의 시작에 `'/*'`를 명시하고 주석의 끝에 `'*/'`를 명시한다. 여러 줄의 주석을 쓸 수 있다.
- `--`  
`'--'`를 연이어 두 번 명시하여 한 줄의 주석을 쓸 때 사용한다.

## 지원 언어

다음은 알티베이스가 제공하는 언어이다.

- US7ASCII: 영어
- KO16KSC5601, MS949: 한국어
- ZHT16BIG5: 대만어

- ZHS16CGB231280: 중국어
- UTF8: 유니코드
- SHIFT-JIS: 일본어

알티베이스에서 지원하는 언어를 오류없이 사용하기 위해 서버와 클라이언트의 문자집합 설정이 일치하여야 한다. 문자집합에 따라 문자 처리 단위가 달라지므로 정확한 처리를 위해 문자집합 설정에 주의가 필요하다. 문자집합에 관한 자세한 내용은 *Starting User's Manual*의 NLS\_USE 프로퍼티 부분을 참고한다.

---

## 알티베이스 객체 (Object)

알티베이스에서 제공하는 데이터베이스 객체(object)는 스키마 객체와 비스키마 객체로 나누어지고 그 종류는 다음과 같다.

### 스키마 객체

- 제약조건 (constraint)
- 인덱스 (index)
- 시퀀스 (sequence)
- 시노님 (synonym)
- 테이블 (table)
- 저장 프로시저 (stored procedure)
- 뷰 (view)
- 트리거 (trigger)

### 비 스키마 객체

- 사용자 (user)
- 이중화 (replication)
- 테이블스페이스 (tablespace)
- 디렉토리 (directory)

---

## 객체 이름 생성 규칙

### 객체 이름

테이블, 뷰, 시퀀스, 시노님, 저장프로시저는 동일한 이름 공간(Name Space)를 가지므로 같은 이름의 객체는 존재할 수 없다. 그 외에 다른 객체들은 각각의 이름 공간을 가진다.

알티베이스의 객체들은 한 사용자 이름 내에서 유일한 이름을

사용해야 한다. 따라서 제약조건의 이름도 한 테이블 내에서 유일한 값이 아니고 한 사용자 내에서 유일한 이름이어야 한다.

객체의 이름은 최대 40 바이트 즉 40 자까지 사용할 수 있으며, 영문의 경우 대문자와 소문자를 구별하지 않는다.

A-Z, a-z, 0-9, \_, \$ 만을 사용할 수 있으며 알티베이스의 예약어는 사용할 수 없다. 알티베이스의 예약어는 다음 절을 참고한다.

첫 글자는 반드시 문자여야 한다.

객체에 대해서는 Administrator's Manual 을 참고한다.

## 비밀번호

사용자가 알티베이스에 접속하기 위해 사용하는 비밀번호 역시 객체 이름과 유사한 제약조건을 가진다.

A-Z, a-z, 0-9, \_, \$ 만을 사용할 수 있으며 알티베이스의 예약어는 사용할 수 없다.

첫 글자는 반드시 문자여야 하고 비밀번호의 최대 크기는 운영체제에 따라 다르며 8~40 자 사이이다. Solaris10, Window XP 의 경우 40 자를 지원한다.

## 예약어

다음 단어들은 알티베이스에서 예약되어 있는 단어들로 테이블, 칼럼, 사용자 같은 데이터베이스 객체 이름이나 비밀번호로 사용할 수 없다.

ADD	ALL	ALTER
AND	ANY	AS
ASC	BEGIN	BETWEEN
BY	CASCADE	CASE
CHECK	CLOSE	COLUMN
COMMIT	CONNECT	CONSTANT
CONSTRAINT	CONSTRAINTS	CONTINUE
CREATE	CUBE	CURSOR
CYCLE	DATABASE	DECLARE
DEFAULT	DELETE	DESC
DISCONNECT	DISTINCT	DROP
ELSE	ELSEIF	ELSIF
END	ESCAPE	EXCEPTION
EXEC	EXECUTE	EXISTS
EXIT	EXTENTSIZE	FALSE
FETCH	FIXED	FOR

FOREIGN	FROM	FULL
FUNCTION	GET	GOTO
GRANT	GROUP	GROUPING
HAVING	IDENTIFIED	IF
IN	INDEX	INNER
INSERT	INTERSECT	INTO
IS	ISOLATION	JOIN
KEY	LEFT	LEVEL
LIKE	LIMIT	LOCALUNIQUE
LOCK		
LOOP	MAXROWS	MINUS
MODE	NATIVE	NO
NOCYCLE	NOT	NULL
OFF	OFFLINE	ON
OPEN	OR	ORDER
OTHERS	OUT	OUTER
PRIMARY	PRIOR	PRIVILEGES
PROCEDURE	RAISE	READ
REFERENCES	RENAME	REPLACE
REPLICATION	RESTRICT	RETURN
REVERSE	REVOKE	RIGHT
ROLLBACK	ROLLUP	ROW
ROWCOUNT	ROWTYPE	SAVEPOINT
SELECT	SEQUENCE	SESSION
SET	SETS	SOME
SQLCODE	SQLERRM	START
STEP	SYNONYM	SYSTEM
TABLE	TABLESPACE	TEMPORARY
THEN	TO	TRANSACTION
TRIGGER	TRUE	TRUNCATE
UNION	UNIQUE	UNTIL
UPDATE	USER	VALUES
VARIABLE	VIEW	WAIT
WHEN	WHERE	WHILE
WITH	WORK	WRITE

## SQL 분류

알티베이스가 제공하는 전체 SQL 문을 분류하면 다음과 같다.

각 SQL 문에 대한 자세한 사용방법은 3 장 데이터 정의어, 4 장 데이터 조작어, 5 장 데이터 제어어를 참조한다.

### 데이터 정의어(DDL)

SQL 문	설명
ALTER DATABASE	데이터베이스 정의 변경
ALTER INDEX	PERSISTENT 인덱스 변경
ALTER REPLICATION	이중화의 시작, 종료
ALTER SEQUENCE	시퀀스의 정의 변경
ALTER TABLE	테이블의 정의 변경
ALTER TABLESPACE	테이블스페이스 정의 변경
ALTER TRIGGER	트리거 정의 변경
ALTER USER	사용자의 암호 변경
ALTER VIEW	뷰 재 컴파일
CREATE DATABASE	데이터베이스 생성
CREATE DIRECTORY	디렉토리 생성
CREATE INDEX	인덱스 생성
CREATE QUEUE	큐 생성
CREATE REPLICATION	이중화 생성
CREATE SEQUENCE	시퀀스 생성
CREATE SYNONYM	시노님 생성
CREATE TABLE	테이블 생성
CREATE TABLESPACE	테이블스페이스 생성
CREATE TRIGGER	트리거 생성
CREATE USER	사용자 생성
CREATE VIEW	뷰 생성
DROP DIRECTORY	디렉토리 삭제
DROP INDEX	인덱스 삭제
DROP REPLICATION	이중화 삭제
DROP SEQUENCE	시퀀스 삭제
DROP SYNONYM	시노님 삭제
DROP TABLE	테이블 삭제
DROP TABLESPACE	테이블스페이스 삭제
DROP TRIGGER	트리거 삭제
DROP USER	사용자 삭제
DROP VIEW	뷰 삭제

GRANT	권한 부여
RENAME	테이블, 시퀀스, 뷰 이름 변경
REVOKE	권한 삭제
TRUNCATE TABLE	테이블의 모든 레코드 삭제

[표 1-1] 데이터 정의어 목록

메타 정보가 변경되는 SQL 문들로 위의 문들을 수행하면 현재 시작되어 있는 트랜잭션은 종료되고 새로운 트랜잭션으로 위의 문들을 처리한 후 그 트랜잭션은 종료된다. 즉, 한 트랜잭션으로 처리하는 SQL 문들이다. 다시 말해서 자동반영(AUTOCOMMIT) 모드가 비설정(OFF) 되어 있는 상태에서 데이터 조작어(DML)를 수행하고 명시적으로 반영(COMMIT)을 호출하지 않았다 하더라도 위의 SQL 문들을 수행하면 이전의 데이터 조작어(DML)들이 모두 묵시적으로 반영된다. 즉, 위 SQL 문을 수행하기 전 데이터 조작어들은 위의 SQL 문 수행 후 룰백문을 이용해 철회될 수 없다.

## 데이터 조작어(DML)

SQL 문	설명
DELETE	데이터의 삭제
INSERT	데이터의 삽입
LOCK TABLE	특정한 모드에서 테이블 잠금
SELECT	데이터의 검색
UPDATE	데이터의 변경
MOVE	데이터의 이동
ENQUEUE	메시지를 큐에 삽입
DEQUEUE	메시지를 큐에서 갖고와 삭제

[표 1-2] 데이터 조작어 목록

자동반영(AUTOCOMMIT) 모드가 비설정(OFF) 되어 있는 상태에서 위의 SQL 문을 수행하면 데이터 정의어(DDL)와 달리 수행 후 묵시적으로 반영되지 않는 SQL 문들이다. 따라서, 자동반영 비설정(AUTOCOMMIT OFF) 상태에서 여러 개의 데이터 조작어(DML)를 수행하고 룰백(rollback)을 호출하면 수행된 모든 데이터 조작어(DML)들이 철회된다.

## 데이터 제어어(DCL)

### 시스템 제어문

SQL 문	설명
ALTER SYSTEM	checkpoint, backup 수행

명시적으로 디스크에 반영(checkpoint) 또는 현재 데이터베이스 상태 저장(online-backup)을 할 때 사용된다.

### 작업 제어문

SQL 문	설명
ALTER SESSION	작업의 설정 사항 변경

한 개의 작업 설정 사항을 변경하는 것으로 현재 작업에만 적용된다.

주의) 현재 동작 중인 트랜잭션이 없어야 한다.

### 트랜잭션 제어문

SQL 문	설명
COMMIT	트랜잭션 정상 종료
ROLLBACK or ROLLBACK TO SAVEPOINT <i>savepoint_name</i>	트랜잭션 철회
SAVEPOINT <i>savepoint_name</i>	지금까지의 트랜잭션을 임시저장
SET TRANSACTION	현재 트랜잭션의 설정 사항 변경

트랜잭션의 설정 사항(ISOLATION LEVEL)의 설정을 변경하고 트랜잭션의 종료와 철회를 사용자가 명시적으로 할 때 사용하는 SQL 문이다.

위의 SQL 문은 현재 작업에만 영향을 미치고 다른 작업에는 영향을 미치지 않는다.



## 2. 자료형

SQL을 사용하여 데이터베이스에 데이터를 저장하고 질의하기 위해서는 데이터베이스의 자료형에 대한 이해가 선행되어야 한다.

이 장에서는 알티베이스의 데이터형에 대해서 자세히 설명하고 있다.

# 자료형의 개요

## 데이터형의 종류

알티베이스에서 제공하는 데이터형은 다음과 같다.

### 문자형 데이터형

타입	Length	Size
CHAR	1 ~ 65535	length + 2
VARCHAR	1 ~ 65535	length + 2

### 숫자형 데이터형

Non-native	타입	Precision	Scale	Size (bytes)	비고
	NUMERIC	38	0	3+((precision )+2)/2	*고정 소수점
	NUMERIC(p)	1 ~ 38	0		*DECIMAL은
	NUMERIC(p,s)	1 ~ 38	-84 ~ 128		NUMERIC과
	DECIMAL	38	0		동일한 데이터
	DECIMAL(p)	1 ~ 38	0		타입이다.
	DECIMAL(p,s)	1 ~ 38	-84 ~ 128		
	NUMBER(p)	1 ~ 38	0		
	NUMBER(p,s)	1 ~ 38	-84 ~ 128		
	NUMBER	38	X	3+((precision )+2)/2	*부동 소수점
Native	FLOAT	38	X		
	FLOAT(p)	1 ~ 38	X		
	타입	호환 C Type	Size (bytes)	비고	
	DOUBLE	double	8	*실수형 부동 소수점	
	REAL	float	4		
	BIGINT	long or long long	8		*정수형
	INTEGER	int	4		
	SMALLINT	short	2		

### 날짜 데이터형

타입	Size (byte)
DATE	8

## 이진 데이터형

타입	Length	Size (byte)
BLOB/CLOB		1~2147483647
BYTE	1~65533	length + 2
NIBBLE	1~255	length/2 + 1
BIT	1~60576	length/8 + 4
VARBIT	1~131070	length/8 + 4

## 공간 데이터형

타입	Length	Size (byte)
GEOMETRY	16~104857600	length + 56

- 실제 레코드의 크기는 위에 명시된 크기(bytes)에서 헤더 정보 크기 만큼 추가된다. 헤더 정보는 OS 환경에 따라 다를 수 있다.
- CHAR, VARCHAR, BYTE 타입의 경우 VARIABLE로 선언된 경우를 의미한다.

## 고정 소수점형

### Size 계산

---

$$(3 + ((p) + 2) / 2)$$

### Precision 0 | Scale 0 | 홀수인 경우 (+ 1)

---

– NUMERIC

예) NUMERIC(38, 0)

$$\text{Size} = 3 + 40/2 = 23 \text{ Bytes}$$

– NUMERIC(p)

NUMERIC(p, 0)

예) NUMERIC(10)

$$\text{Size} = 3 + 12/2 = 9 \text{ Bytes}$$

– NUMERIC(p, s)

예) NUMERIC(10, 9)

$$\text{Size} = 3 + 12/2 = 9 \text{ Bytes}$$

– DECIMAL : NUMERIC 과 동일

– DECIMAL(p) : NUMERIC(p)와 동일

- DECIMAL(p,s) : NUMERIC(p,s)와 동일
- NUMBER(p) : NUMERIC(p) 와 동일
- NUMBER(p,s) : NUMERIC(p,s)와 동일

## 부동 소수점형

### Size 계산

---

$$( 3 + ( ( p ) + 2 ) / 2 )$$

- FLOAT

예) FLOAT(38)

$$\text{Size} = 3 + 40/2 = 23 \text{ Bytes}$$

- FLOAT(p)

예) FLOAT(20)

$$\text{Size} = 3 + 22/2 = 14 \text{ Bytes}$$

- NUMBER: FLOAT 과 동일

---

## NULL

데이터 행을 테이블에 삽입할 때 열의 값을 모르거나 값이 아직 없는 경우, 즉 값이 존재 하지 않는 것을 나타내는 것이 널(NULL)이다. 따라서 이는 0 또는 공백과는 다른 의미를 나타내며 비교연산이나 저장시 특별하게 취급된다.

어떤 연산에서 수식에 널이 포함되면 NVL(), IS NULL 조건, IS NOT NULL 조건을 제외하고 최종 연산의 결과는 널이 된다. 즉, 널이 포함되면 비교 또는 연산이 의미가 없어지게 된다.

생성할 때 NOT NULL 또는 PRIMARY KEY로 정의되지 않은 모든 데이터 유형의 컬럼은 널 값을 포함할 수 있다.

---

## 데이터 타입 변환과 호환

다음의 테이블은 데이터 타입의 변환가능 행렬을 나타낸다.

비교 연산은 같은 데이터형인 경우 가능하다.

그러나 비교되는 자료의 데이터 형이 다른 경우 데이터형 변환을 수행 후 비교한다.

단, 문자형 데이터일 경우에는 다른 데이터형으로 변환되어 수행된다.

		c	v	c	b	d	d	f	i	n	n	r	s	d	b	b	n	b	v	g
		h	a	l	i	e	o	l	n	u	u	e	m	a	l	y	i	a	e	
		a	r	o	g	c	u	o	t	m	m	a	a	t	o	t	b	r	o	
		r	c	b	i	i	b	a	e	b	e	l	l	e	b	e	b	b	m	
		h			n	m	l	t	g	e	r	l	l	i			l	i	e	
		a			t	a	e	e	r	i	c	i	n			l	e	t	t	
		r			l			r											r	y
char		o	o		o	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	o	o	o	o	
clob				o																
bigint		o	o		o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	
decimal		o	o		o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	
double		o	o		o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	
float		o	o		o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	
integer		o	o		o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	
number		o	o		o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	
numeric		o	o		o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	
real		o	o		o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	
smallint		o	o		o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	
date		o	o													o				
blob																	o			
byte																o	o			
nibble																	o			
bit																	o	o		
varbit			o														o	o		
geometry																		o		

## 데이터 형식 변환

### 구문

datatype '문자 또는 상수 literal'

### 설명

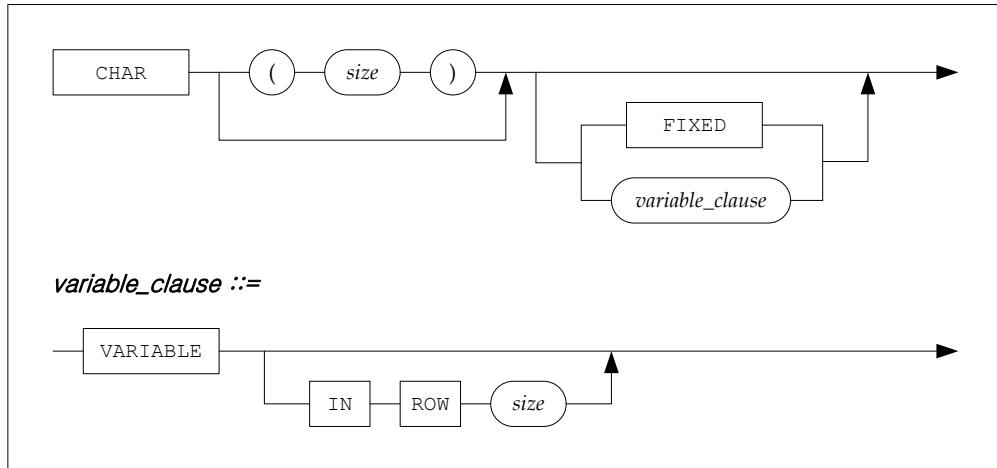
상수 테이터가 가지는 데이터 타입을 명시적으로 다른 데이터 타입으로 변환한다. 예를 들어, 다음은 157.27의 숫자 값을 '157.27'의 문자열로 변환한다.

CHAR '157.27'

# 문자형 데이터 타입

## CHAR

### 흐름도



### 구문

CHAR [(size)] [[**FIXED** |] **VARIABLE** ( IN ROW size ) ]

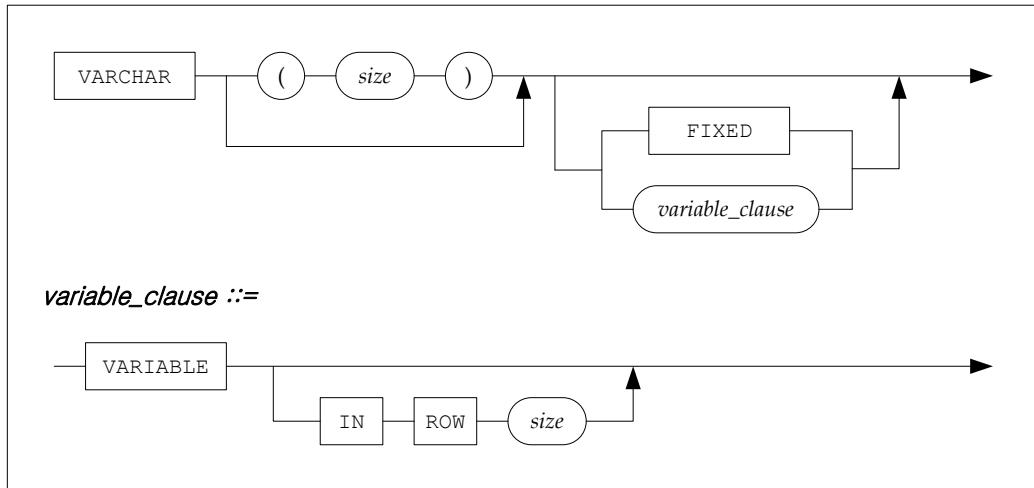
### 설명

명시된 크기(size)만큼 고정 길이를 가지는 문자형 데이터 타입이다.  
명시된 전체 크기에 비해 입력 값의 크기가 작을 경우 뒷부분에  
공백으로 문자가 채워진다.

CHAR 칼럼의 크기는 기본값으로 1 바이트이며, 최대 길이는 선언  
방식에 따라 다르다. FIXED로 선언될 경우 페이지 크기(8K) 내에서  
지정할 수 있으며, VARIABLE로 선언될 경우 32KB 즉 32768  
바이트까지 저장할 수 있다.

## VARCHAR

### 흐름도



### 구문

**VARCHAR [(size)] [[**FIXED** |] **VARIABLE** ( IN ROW size ) ]**

### 설명

명시된 크기 내에서 가변 길이를 가지는 문자형 데이터 타입이다,  
FIXED로 선언될 경우 명시된 크기 만큼 저장소의 크기가 설정되며,  
VARIABLE로 선언될 경우는 명시된 크기 내에서 가변 길이를 갖는다.

예를 들어, 저장해야 하는 데이터가 "magic"일 경우 VARCHAR(10)  
FIXED로 정의하면 "magic"으로 10 바이트 만큼의 저장공간을  
사용하며, VARCHAR(10) VARIABLE로 정의하면 "magic"으로 5  
바이트 만큼의 저장 공간을 사용한다.

FIXED는 저장공간이 낭비되는 반면, 저장공간의 크기가 고정되어  
있어 처리 속도가 빨라지는 장점이 있다.

VARIABLE은 데이터마다 저장 공간의 크기가 달라 처리 속도가 느린  
반면, 저장공간을 효율적으로 사용하는 장점이 있다.

프로퍼티 VARIABLE\_COLUMN\_IN\_ROW\_SIZE를 지정하지 않은  
경우, 메모리 테이블에 대해서는 데이터 크기가 30 바이트 이하이면,  
FIXED 영역에 저장되며, 초과하면 VARIABLE로 인식된다. 디스크

테이블에 대해서는 크기에 관계없이 VARIABLE로 인식된다.

VARCHAR 칼럼의 크기는 기본값으로 1 바이트이다. 최대 길이는, FIXED로 선언될 경우 한 페이지 크기(8K) 내에서 지정할 수 있으며, VARIABLE로 선언될 경우 32KB 즉 32768 바이트까지 저장할 수 있다.

\* VARCHAR는 값의 길이가 열의 길이보다 짧은 경우, 데이터의 실제 길이 만큼만 저장하는 가변 길이 데이터 타입이다. 반면, CHAR 데이터 타입의 경우, 열의 길이보다 짧은 값을 넣으면, 값의 오른쪽에서 열 끝까지 공백으로 채워진다. 예를 들어, 열이 CHAR(10)으로 정의되었는데 저장할 데이터가 "magic"이면, 이 데이터는 "magic\_\_\_\_\_ "으로 저장된다. 여기서 "\_"은 공백을 나타낸다.

\* VARCHAR의 데이터의 크기가 30이면 실제 메모리 공간은 32를 차지한다. 다시 말해, 실제 차지하는 메모리 공간이 32이하일 경우 FIXED로 인식된다.

#### IN ROW size

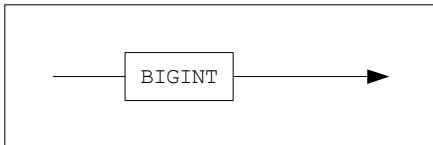
메모리 테이블의 경우 사용자는 VARIABLE 지정시에 'IN ROW' 구문을 이용해서 fixed 영역에 들어갈 데이터 길이와 variable 영역에 들어갈 데이터 길이를 지정할 수 있다. 예를 들어 'VARCHAR(400) in row 200'으로 설정된 칼럼의 경우, 데이터 삽입 시에 길이가 200 바이트 이하이면 fixed 영역에, 200 바이트를 초과하면 variable 영역에 저장된다.

특별한 경우가 아니면 매번 'IN ROW size'를 지정할 필요 없이 VARIABLE\_COLUMN\_IN\_ROW\_SIZE 프로퍼티에 디폴트 in row size를 지정하고 개별 선언문에서는 생략할 수 있다. 해당 프로퍼티 대한 상세한 설명은 Starting User's Manual의 알터베이스 프로퍼티 부분을 참조한다.

## 숫자형 데이터 타입

### BIGINT

#### 흐름도



#### 구문

BIGINT

#### 설명

8 바이트 크기의 정수형 데이터 타입이다.

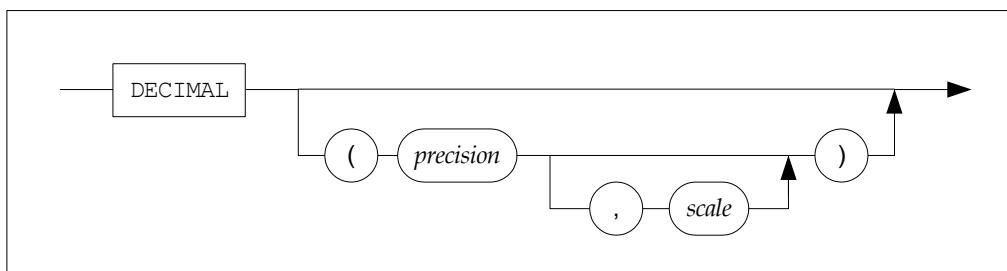
C의 long(64 bit)이나 long long(32 bit)과 동일한 데이터 타입이다.

$-2^{63} + 1$ (-9223372036854775807)에서  $2^{63} - 1$ (9223372036854775807) 사이의 정수 데이터(전체 숫자)이다.

\* 최소값(-9223372036854775808)은 널 값으로 사용된다.

### DECIMAL

#### 흐름도



#### 구문

DECIMAL [(*precision*[, *scale*])]

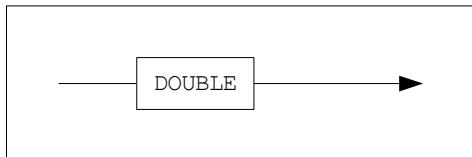
## 설명

DECIMAL 은 NUMERIC 데이터 타입과 동일한 데이터 타입이다.

---

## DOUBLE

### 흐름도



### 구문

DOUBLE

## 설명

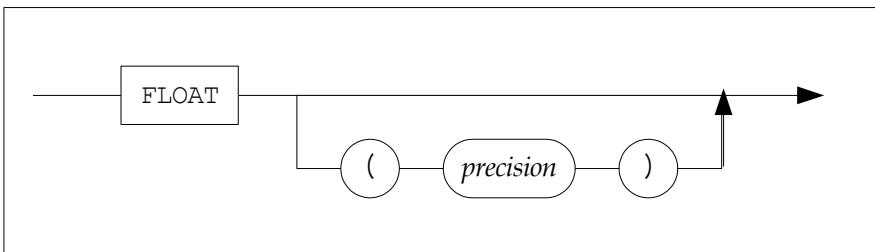
8 바이트 크기의 부동 소수점형이다.

C 의 double 과 동일한 데이터 타입이다.

---

## FLOAT

### 흐름도



### 구문

FLOAT [(precision)]

## 설명

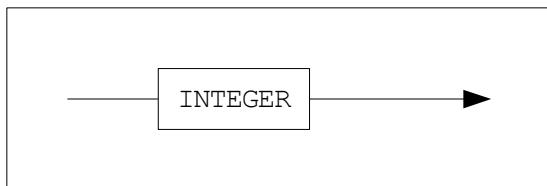
$-1E+120$ 에서  $1E+120$ 까지의 부동 소수점 숫자 데이터다.  
Precision은 정밀도 표시하기 위해 FLOAT 숫자의 가수를 유효숫자  
표기법으로 저장하는 데 사용되는 비트 수이다. Precision의 범위는  
1에서 38까지이다.

Precision이 생략되면 기본값으로 38이 설정된다.

---

## INTEGER

### 흐름도



### 구문

INTEGER

### 설명

4 바이트 크기의 정수형 데이터 타입이다.

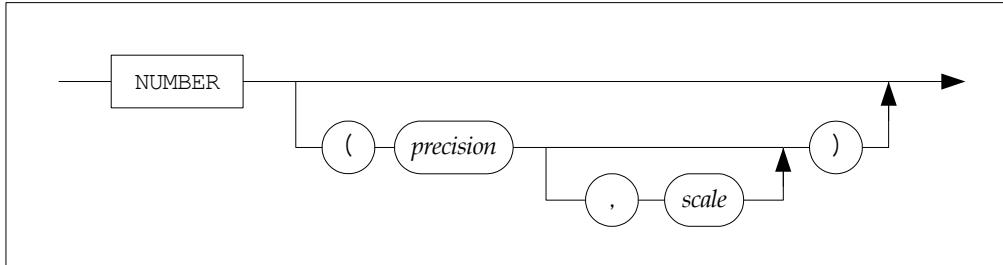
C의 int와 동일한 데이터 타입이다.

정수값  $-2,147,483,647$ 에서  $2,147,483,647$ 까지의 값을 가질 수 있다.

\*  $-2,147,483,648$ 은 널값으로 사용된다.

## NUMBER

### 흐름도



### 구문

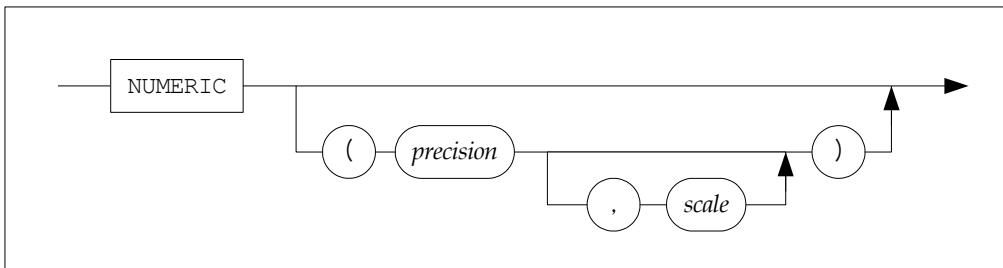
NUMBER [(precision, scale)]

### 설명

NUMERIC type의 alias 형이다. 단, precision과 scale이 명시되지 않으면 FLOAT과 동일하게 취급된다.

## NUMERIC

### 흐름도



### 구문

NUMERIC [(precision, scale)]

### 설명

Precision 과 scale 을 가지는 숫자형 데이터 타입으로 precision 만큼의 유효 숫자와 scale 만큼의 소수점 이하 정밀도를 가지는 고정 소수점형이다. FLOAT 데이터 타입이 실수를 표현하는 형식인 부동소수점 형식인 반면 NUMERIC 데이터 타입은 precision 과 scale 이 모두 생략되면 precision 은 38, scale 은 0 인 정수를 표현하는 형식인 고정소수점 으로 사용된다.

- Precision은 1부터 38까지의 값을 명시할 수 있다.
- Scale은 -84에서 126까지의 값을 명시할 수 있다.
- Precision이 생략되면 기본값으로 38이 설정된다.
- Scale이 생략되면 기본값으로 0이 설정된다.

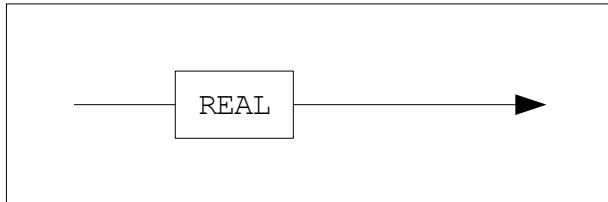
다음은 각각의 NUMERIC 타입으로 정의한 후 값이 1234567.89 일때의 변환된 값을 나열한 것이다.

- NUMERIC => 1234568
- NUMERIC(9) => 1234568
- NUMERIC(9, 2) => 1234567.89
- NUMERIC(9, 1) => 1234567.9
- NUMERIC(6) => precision 초과
- NUMERIC(7, -2)=> 1234500
- NUMERIC(7, 2) => precision 초과

---

## REAL

### 흐름도



### 구문

REAL

### 설명

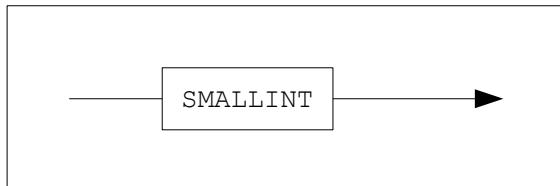
4 바이트 크기의 부동 소수점형이다.

C 의 float 과 동일한 데이터 타입이다.

---

## SMALLINT

### 흐름도



### 구문

SMALLINT

### 설명

2 바이트 크기의 정수형 데이터 타입이다.

C 의 short 와 동일한 데이터 타입이다.

$-2^{15} + 1(-32,767)$ 에서  $2^{15} - 1(32,767)$  사이의 정수 데이터다.

\* 최소값(-32768)은 널값으로 사용된다.

---

## 숫자형 데이터 형식

TO\_CHAR 함수나 TO\_NUMBER 등의 데이터 변환 함수를 사용할 때 숫자형 데이터에 대하여 다음과 같이 형식을 지정할 수 있다. 숫자 데이터 형식은 하나 이상의 숫자 표시 인자로 구성되며 이장에서는 각각의 인자와 관련된 데이터 형식의 예를 설명한다.

---

, (쉼표)

### 설명

지정한 위치에 쉼표를 출력한다. 쉼표는 여러 번 사용할 수 있다.

#### 제한사항

쉼표는 숫자의 끝이나 마침표의 오른쪽에 올 수 없고, 숫자의 맨 앞자리에도 쉼표를 사용할 수 없다.

### 예제

```
iSQL> SELECT TO_CHAR (1234, '99,99') FROM dual;  
TO_CHAR (1234, '99,99')
```

```
-----  
12,34  
1 row selected.
```

```
iSQL> SELECT TO_NUMBER ( '12,34', '99,99') FROM dual;  
TO_NUMBER ( '12,34', '99,99')
```

```
-----  
1234  
1 row selected.
```

---

. (마침표)

### 설명

마침표를 지정한 위치에 추가로 반환한다.

#### 제한사항

한 숫자 내에서 마침표는 한번만 사용할 수 있다.

### 예제

```
iSQL> SELECT TO_CHAR (1.234, '99.999') FROM dual;  
TO_CHAR (1.234, '99.999')
```

```
-----  
1.234  
1 row selected.
```

```
iSQL> SELECT TO_NUMBER ( '1.234', '99.999') FROM dual;
```

```
TO_NUMBER ( '1.234', '99.999')
```

---

```
1,234  
1 row selected.
```

```
$
```

---

## 설명

숫자 앞에 \$ 기호를 붙인다.

## 예제

```
iSQL> SELECT TO_CHAR (123, '$9999') FROM dual;  
TO_CHAR (123, '$9999')
```

---

```
$123  
1 row selected.
```

```
iSQL> SELECT TO_NUMBER ( '$0123', '09$99') FROM dual;  
TO_NUMBER ( '$0123', '09$99')
```

---

```
123  
1 row selected.
```

## 0(숫자 0)

---

## 설명

정수 부분의 유효 자리수가 실제 숫자의 자리수 보다 많을 경우 실제 숫자 앞에 0 을 채워서 반환한다. 그 외의 특성은 9 와 같다.

## 예제

```
iSQL> SELECT TO_CHAR (123, '0999') FROM dual;  
TO_CHAR (123, '0999')
```

---

```
0123
```

## 9(숫자 9)

---

## 설명

출력할 숫자의 자릿수를 숫자 9 를 이용해서 표시한다. 실제 숫자의 자릿수 보다 9 의 개수가 더 많으면 앞에 공백문자를 출력하여 길이를 맞추고, 정수 부분의 9 의 개수가 실제 숫자보다 더 적으면 숫자의 길이만큼 # 를 출력한다. #의 개수는 사용자가 지정한 형식에 쓰인 문자의 개수 + 1(부호문자)이다.

9 사이에 오는 마침표는 숫자의 정수 부분과 소수 부분을 구분하여 출력하게 한다.

첫 번째 인자에 소수가 있는데 사용자가 지정한 형식에 소수를 표현하는 부분이 없거나, 더 적으면 반올림해서 사용자가 지정한 형식의 소수 부분의 길이에 맞춘다.

## 예제

```
iSQL> SELECT TO_CHAR (123, '99999') FROM dual;  
TO_CHAR (123, '99999')
```

```
123
```

```
iSQL> SELECT TO_CHAR (123.55, '999') FROM dual;  
TO_CHAR (123.55, '999')
```

```
124
```

```
1 row selected.
```

```
iSQL> SELECT TO_CHAR (123.4567, '999999') FROM dual;  
TO_CHAR (123.4567, '999999')
```

```
123
```

```
1 row selected.
```

```
iSQL> SELECT TO_CHAR (1234.578, '9999.99') FROM dual;  
TO_CHAR (1234.578, '9999.99')
```

```
1234.58
```

```
1 row selected.
```

```
iSQL> SELECT TO_CHAR (1234.578, '999.99999') FROM dual;  
TO_CHAR (1234.578, '999.99999')
```

```
#####
```

```
1 row selected.
```

```
iSQL> SELECT TO_NUMBER ( '123', '99999') FROM dual;  
TO_NUMBER ( '123', '99999')
```

```
123
```

```
1 row selected.
```

```
iSQL> SELECT TO_NUMBER ( '1234.58', '9999.99') FROM dual;  
TO_NUMBER ( '1234.58', '9999.99')
```

```
1234.58
```

```
1 row selected.
```

## B

### 설명

결과값이 0 일 경우, 0 을 공백(Blank)로 반환한다.

## 예제

```
iSQL> SELECT TO_CHAR (0.4, 'B9') FROM T1;  
TO_CHAR (0.4, 'B9')
```

```
1 row selected.
```

## EEEE

---

### 설명

입력 받은 숫자를 지수 표기법을 이용하여 표기한다.

### 제한사항

EEEE 는 항상 오른쪽 끝에 와야 한다. 단 S, PR, MI 보다는 왼쪽에 오는 것이 가능하다. 쉼표와 같이 사용할 수 없다.  
TO\_NUMBER 함수에서 사용할 수 없다.

### 예제

```
iSQL> SELECT TO_CHAR (1234, '9.9EEEE') FROM dual;  
TO_CHAR (1234, '9.9EEEE')  
-----  
1.2E+03  
1 row selected.
```

## MI

---

### 설명

MI 를 숫자 표현 형식의 오른쪽 끝에 사용하면 입력 받은 수가 음수일 경우 마이너스(–) 기호를 숫자 끝에 붙여서 반환한다. 양수일 경우에는 공백문자가 들어간다.

### 제한사항

MI 는 항상 숫자 형식 표현의 오른쪽 끝에 와야 한다. S, PR 과 같이 사용할 수 없다.

### 예제

```
iSQL> SELECT TO_CHAR (-123, '999MI') FROM dual;  
TO_CHAR (-123, '999MI')  
-----  
123–  
1 row selected.
```

```
iSQL> SELECT TO_NUMBER ( '123–', '999MI') FROM dual;  
TO_NUMBER ( '123–', '999MI')  
-----  
-123  
1 row selected.
```

## PR

---

### 설명

PR 를 숫자 표현 형식의 오른쪽 끝에 사용하면 입력 받은 수가 음수일 경우 마이너스 기호(–) 대신 <숫자> 형태로 출력된다.

### 제한사항

PR 은 항상 숫자 표현 형식의 오른쪽 끝에 와야 한다.

S, MI 와 같이 사용할 수 없다.

#### 예제

```
iSQL> SELECT TO_CHAR (-123, '999PR') FROM dual;  
TO_CHAR (-123, '999PR')
```

```
-----  
<123>  
1 row selected.
```

```
iSQL> SELECT TO_NUMBER ('<123>', '999PR') FROM dual;  
TO_NUMBER ('<123>', '999PR')
```

```
-----  
-123  
1 row selected.
```

#### RN

---

#### 설명

입력 받은 수를 로마 숫자로 변환한다. 입력할 수 있는 수는 1부터 3999 까지이다. 숫자 표현 형식을 소문자 rn 으로 사용하면 로마 숫자가 소문자로 출력된다.

#### 제한사항

다른 숫자 형식과 같이 사용할 수 없다. TO\_NUMBER 함수에서 사용할 수 없다.

#### 예제

```
iSQL> SELECT TO_CHAR (14, 'RN') FROM dual;  
TO_CHAR (14, 'RN')
```

```
-----  
XIV  
1 row selected.
```

#### S

---

#### 설명

숫자 표현 형식의 처음이나 끝에 와서 입력 받은 수의 기호에 따라서 마이너스(–) 또는 플러스(+) 기호를 붙인다.

#### 제한사항

S 는 숫자 표현 형식의 맨 앞이나 맨 뒤에만 올 수 있다. MI, PR 과 같이 사용할 수 없다.

#### 예제

```
iSQL> SELECT TO_CHAR (123, 'S999.99') FROM dual;  
TO_CHAR (123, 'S999.99')
```

```
-----  
+123.00  
1 row selected.
```

```
iSQL> SELECT TO_CHAR (-123, '999.99S') FROM dual;
```

```
TO_CHAR (-123, '999.99S')
-----
123.00-
1 row selected.

iSQL> SELECT TO_NUMBER ( '+123', 'S999.99') FROM dual;
TO_NUMBER ( '+123', 'S999.99')

-----
123
1 row selected.

iSQL> SELECT TO_NUMBER ( '123.00-', '999.99S') FROM dual;
TO_NUMBER ( '123.00-', '999.99S')

-----
-123
1 row selected.
```

V

### 설명

V 다음에 있는 9의 개수와 10을 곱하고 그 값을 인자로 받은 숫자와 곱한다. V 앞의 9의 개수는 첫 번째 인자의 유효숫자의 개수를 의미한다.

### 제한사항

마침표와 같이 사용할 수 없다. TO\_NUMBER 함수에서 사용할 수 없다.

### 예제

```
iSQL> SELECT TO_CHAR (12, '99V99') FROM dual;
TO_CHAR (12, '99V99')

-----
1200
1 row selected.

iSQL> SELECT TO_CHAR (1200, '99V99') FROM dual;
TO_CHAR (1200, '99V99')

-----
#####
1 row selected.

iSQL> SELECT TO_CHAR (-123.456, '999V999EEEEMI') from dual;
TO_CHAR (-123.456, '999V999EEEEMI')

-----
1235E+02-
1 row selected.
```

XXXX

### 설명

입력 받은 수를 16 진수로 변환한다. 만약 정수가 아니라면

반올림하여 16 진수로 변환한다. xxxx 는 16 진수 중 문자를 소문자로 반환한다.

### 제한사항

다른 숫자 표현 형식과 같이 사용할 수 없다. 변환할 수는 0 이상이어야 한다.

### 예제

```
iSQL> SELECT TO_CHAR (123, 'XXXX') FROM dual;  
TO_CHAR (123, 'XXXX')
```

```
_____  
7B  
1 row selected.
```

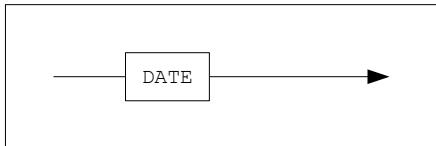
```
iSQL> SELECT TO_NUMBER ('ABC', 'XXXX') FROM dual;  
TO_NUMBER ('ABC', 'XXXX')
```

```
_____  
2748  
1 row selected.
```

## 날짜형 데이터 타입

### DATE

#### 흐름도



### 구문

DATE

### 설명

날짜를 표현하는 데이터 타입이다.

날짜 관련 함수의 경우 일부 플랫폼에서 1년 1월 1일부터 9999년 12월 31일의 범위 내에서 사용 가능하다.

8 바이트의 크기를 가지며, '1999/11/18 00:00:00' 또는 '2000/01/31 15:23:50'과 같이 문자열로 표현된다.

## 날짜형 데이터 형식

날짜형 데이터 타입의 데이터는 데이터베이스 내부적으로는 숫자형 데이터로 관리하지만 사용자는 TO\_CHAR/TO\_DATE 변환 함수 등을 사용해서 문자열로 표시할 수 있다. 변환 함수를 사용할 때 사용자는 보고자 하는 형식에 맞게 날짜형 데이터 형식 문자열을 지정해 주어야 한다.

날짜형 데이터 형식은 다음과 같은 기본요소들로 구성된다.

- AM, PM
- CC
- D, DD, DDD, DAY, DY
- HH, HH12, HH24
- MM, MON, MONTH

- MI
- Q
- SS, SSSSS, SSSSSS, SSSSSSS, FF[1..6]
- WW, W
- Y, YYYY, YYYY, YY, RR, RRRR

위의 기본 요소들과 함께 다음의 다음의 구두점과 특수 문자들도 날짜형 데이터형식을 구성하는 요소이다.

- 하이픈(-)
- 슬래시(/)
- 쉼표(,)
- 마침표(.)
- 콜론(:)
- 홑따옴표(')

각각의 기본 요소들이 의미하는 바와 활용 예를 다음에서 살펴보자.

## AM, PM

---

### 설명

정오를 기준으로 오전/오후를 구분한다. ('AM' 또는 'PM')

### 예제

```
iSQL> SELECT TO_CHAR ( TO_DATE( '13', 'HH' ), 'AM' ) FROM dual;
TO_CHAR ( TO_DATE( '13', 'HH' ), 'AM' )
```

---

```
PM
1 row selected.
```

```
iSQL> SELECT TO_DATE('1980-12-28 PM', 'YYYY-MM-DD AM') FROM dual;
TO_DATE('1980-12-28 PM', 'YYYY-MM-DD AM')
```

---

```
1980/12/28 12:00:00
1 row selected.
```

## CC

---

### 설명

세기를 표시한다.

- 4자리 년도 중 뒤의 2자리 값이 01~99이면, 4자리 년도 중 앞의 두 자리의 값에 1을 더해서 반환한다.
- 4자리 년도 중 뒤의 2자리 값이 00이면, 4자리 년도 중 앞의 두 자리 값을 그대로 반환한다.

TO\_DATE 함수에서는 인자로 사용할 수 없다.

### 예제

```
iSQL> SELECT TO_CHAR ( '28-DEC-1980', 'CC' ) FROM dual;
TO_CHAR ( '28-DEC-1980', 'CC' )
```

---

```
20  
1 row selected.
```

## D

---

### 설명

일주일 중 몇 번째 날인지를 나타내는 1 ~ 7 까지의 숫자이다.  
일요일부터 1로 시작한다.

TO\_DATE 함수에서 인자로 사용할 수 없다.

### 예제

```
iSQL> SELECT TO_CHAR ( '28-DEC-1980', 'D' ) FROM dual;  
TO_CHAR ( '28-DEC-1980', 'D' )
```

---

```
1  
1 row selected.
```

## DAY

---

### 설명

요일의 영문이름을 나타낸다. (SUNDAY, MONDAY,...)

TO\_DATE 함수에서 사용할 수 없다.

### 예제

```
iSQL> SELECT TO_CHAR ( '28-DEC-1980', 'DAY' ) FROM dual;  
TO_CHAR ( '28-DEC-1980', 'DAY' )
```

---

```
SUNDAY  
1 row selected.
```

## DD

---

### 설명

한달 중 몇 번째 날인지를 나타낸다. (1 ~ 31)

### 예제

```
iSQL> SELECT TO_CHAR ( '28-DEC-1980', 'DD' ) FROM dual;  
TO_CHAR ( '28-DEC-1980', 'DD' )
```

---

```
28  
1 row selected.
```

```
iSQL> SELECT TO_DATE( '1980-12-28', 'YYYY-MM-DD' ) FROM dual;  
TO_DATE( '1980-12-28', 'YYYY-MM-DD' )
```

---

```
1980/12/28 00:00:00  
1 row selected.
```

## DDD

---

## 설명

일 년 중 몇 번째 날인지를 나타낸다. (1 ~ 366)

TO\_DATE 함수에서 사용할 수 없다.

## 예제

```
iSQL> SELECT TO_CHAR( '28-DEC-1980', 'DDD' ) FROM dual;  
TO_CHAR( '28-DEC-1980', 'DDD' )  
_____  
363  
1 row selected.
```

## DY

---

## 설명

요일의 이름을 약자로 나타낸다. (SUN, MON, TUE, ...)

TO\_DATE 함수에서 사용할 수 없다.

## 예제

```
iSQL> SELECT TO_CHAR( '28-DEC-1980', 'DY' ) FROM dual;  
TO_CHAR( '28-DEC-1980', 'DY' )  
_____  
SUN  
1 row selected.
```

## FF [1..6]

---

## 설명

FF 다음의 1 ~ 6 까지의 숫자를 이용하여 마이크로 초의 자리 수를 나타낸다. (0 ~ 999999). FF 형식은 FF6 과 같은 같은 결과를 반환한다.

TO\_DATE 함수에서 사용할 수 없다.

## 예제

```
iSQL> SELECT TO_CHAR( SYSDATE, 'FF5' ) FROM dual;  
TO_CHAR( SYSDATE, 'FF5' )  
_____  
34528  
1 row selected.
```

## HH12

---

## 설명

시간을 12 시간 단위로 나타낸다.(1 ~ 12)

## 예제

```
iSQL> SELECT TO_CHAR( TO_DATE( '2008-12-28 17:30:29', 'YYYY-MM-DD  
HH:MI:SS' ), 'HH12' ) FROM dual;  
TO_CHAR( TO_DATE( '2008-12-28 17:30:29',
```

---

05  
1 row selected.

```
iSQL> SELECT TO_CHAR( TO_DATE( '08-12-28 05:30:29', 'RR-MM-DD  
HH12:MI:SS' ), 'RR-MM-DD HH12:MI:SS' ) FROM dual;  
TO_CHAR( TO_DATE( '08-12-28 05:30:29', 'R
```

---

08-12-28 05:30:29  
1 row selected.

## HH, HH24

---

### 설명

시간을 24 시간 단위로 나타낸다.(0 ~ 23)

### 예제

```
iSQL> SELECT TO_CHAR ( TO_DATE( '2008-12-28 17:30:29', 'YYYY-MM-DD  
HH:MI:SS' ), 'HH' ) FROM dual;  
TO_CHAR ( TO_DATE( '2008-12-28 17:30:29'
```

---

17  
1 row selected.

```
iSQL> SELECT TO_CHAR ( TO_DATE( '2008-12-28 17:30:29', 'YYYY-MM-DD  
HH24:MI:SS' ), 'YYYY-MM-DD HH24:MI:SS' ) FROM dual;  
TO_CHAR ( TO_DATE( '2008-12-28 17:30:29',
```

---

2008-12-28 17:30:29  
1 row selected.

## MI

---

### 설명

분 (0 ~ 59)

### 예제

```
iSQL> SELECT TO_CHAR ( TO_DATE( '1980-12-28 17:30:29', 'YYYY-MM-DD  
HH:MI:SS' ), 'HH' ) FROM dual;  
TO_CHAR ( TO_DATE( '1980-12-28 17:30:29'
```

---

17  
1 row selected.

```
iSQL> SELECT TO_DATE ( '05-12-28 14:30:29', 'RR-MM-DD HH:MI:SS' ) FROM  
dual;  
TO_DATE ( '05-12-28 14:30:29', 'RR-MM-DD
```

---

2005/12/28 14:30:29  
1 row selected.

## MM

---

### 설명

월 (01 ~ 12)

예제

```
iSQL> SELECT TO_CHAR ( TO_DATE( '1980-12-28 17:30:29', 'YYYY-MM-DD
HH:MI:SS' ), 'HH' ) FROM dual;
TO_CHAR ( TO_DATE( '1980-12-28 17:30:29'
_____
17
1 row selected.

iSQL> SELECT TO_DATE ( '05-12-28 14:30:29', 'RR-MM-DD HH:MI:SS' ) FROM
dual;
TO_DATE ( '05-12-28 14:30:29', 'RR-MM-DD
_____
2005/12/28 14:30:29
1 row selected.
```

MON

---

설명

월의 이름을 약자로 표시한다.( JAN, FEB, MAR, ... )

예제

```
SQL> SELECT TO_CHAR (TO_DATE ('1995-12-05', 'YYYY-MM-DD'), 'MON')
FROM dual;
TO_
_____
DEC
```

MONTH

---

설명

월의 이름을 표시한다. (JANUARY, FEBRUARY, ... )

예제

```
iSQL> SELECT TO_CHAR ( '28-DEC-1980', 'Month' ) FROM dual;
TO_CHAR ( '28-DEC-1980', 'Month' )
_____
December
```

1 row selected.

```
iSQL> SELECT TO_DATE ( '05-APRIL-28 14:30:29', 'RR-MONTH-DD
HH:MI:SS' ) FROM dual;
TO_DATE ( '05-APRIL-28 14:30:29', 'RR-MO
_____
2005/04/28 14:30:29
```

1 row selected.

## Q

---

### 설명

분기를 표시한다. (1 ~ 4)

TO\_DATE 함수에서 사용할 수 없다.

### 예제

```
iSQL> SELECT TO_CHAR ( '28-DEC-1980', 'Q' ) FROM dual;  
TO_CHAR ( '28-DEC-1980', 'Q' )
```

---

```
4  
1 row selected.
```

## RM

---

### 설명

로마 숫자로 월을 나타낸다. (I, II, III, IV,... )

### 예제

```
iSQL> SELECT TO_CHAR ( '28-DEC-1980', 'RM' ) FROM dual;  
TO_CHAR ( '28-DEC-1980', 'RM' )
```

---

```
XII  
1 row selected.
```

```
iSQL> SELECT TO_DATE ('28-V-1980', 'DD-MON-YYYY') FROM dual;  
TO_DATE ('28-V-1980', 'DD-MON-YYYY')
```

---

```
1980/05/28 00:00:00  
1 row selected.
```

## RR

---

### 설명

년도를 두자리 정수로 표시한다. 날짜를 표기할 때, 두 자리만 표기한 경우 50 미만인 경우에만 21 세기라고 가정하여 2000 을 더하고, 50 이상인 경우에는 1900 을 더해서 연도를 표시한다. 따라서 표시 가능한 연도는 1950 ~ 2049 까지 이다.

### 예제

```
iSQL> SELECT TO_CHAR ( '28-DEC-80', 'RR' ) FROM dual;  
TO_CHAR ( '28-DEC-80', 'RR' )
```

---

```
80  
1 row selected.
```

```
iSQL> SELECT TO_DATE ( '28-DEC-80', 'DD-MON-RR' ) FROM dual;  
TO_DATE ( '28-DEC-80', 'DD-MON-RR' )
```

---

```
1980/12/28 00:00:00
```

```
1 row selected.
```

## RRRR

---

### 설명

연도 (0 ~ 9999)

네자리, 두자리의 년도를 모두 입력으로 받아서, 숫자가 50 미만인 경우 2000 을 더하고, 50 이상 100 미만인 경우 1900 을 더해서 연도를 나타낸다. 4 자리의 숫자인 경우 그대로가 년도로 표시된다.

### 예제

```
iSQL> SELECT TO_CHAR ( '28-DEC-1980', 'RRRR' ) FROM dual;  
TO_CHAR ( '28-DEC-1980', 'RRRR' )
```

---

```
1980
```

```
1 row selected.
```

```
iSQL> SELECT TO_CHAR ( '28-DEC-1980', 'DD-MON-RRRR' ) FROM dual;  
TO_CHAR ( '28-DEC-1980', 'DD-MON-RRRR' )
```

---

```
28-DEC-1980
```

```
1 row selected.
```

## SS

---

### 설명

초 (0 ~ 59)

### 예제

```
iSQL> SELECT TO_CHAR ( TO_DATE( '1980-12-28 17:30:29', 'YYYY-MM-DD  
HH:MI:SS' ), 'HH' ) FROM dual;  
TO_CHAR ( TO_DATE( '1980-12-28 17:30:29'
```

---

```
17
```

```
1 row selected.
```

```
iSQL> SELECT TO_DATE ( '05-12-28 14:30:29', 'RR-MM-DD HH:MI:SS' ) FROM  
dual;  
TO_DATE ( '05-12-28 14:30:29', 'RR-MM-DD
```

---

```
2005/12/28 14:30:29
```

```
1 row selected.
```

## SSSSS

---

### 설명

지난 자정부터 몇 초가 경과 되었는지 나타낸다. (0 ~ 86399)

### 예제

```
iSQL> SELECT TO_CHAR ( TO_DATE( '1980-12-28 17:30:29', 'YYYY-MM-DD  
HH24:MI:SS' ), 'SSSSS' ) FROM dual;  
TO_CHAR ( TO_DATE( '1980-12-28 17:30:29'
```

```
-----  
62940  
1 row selected.
```

```
iSQL> SELECT TO_DATE('1980-12-28 12345', 'YYYY-MM-DD SSSSS') FROM  
dual;  
TO_DATE('1980-12-28 12345', 'YYYY-MM-DD
```

```
-----  
1980/12/28 03:25:45  
1 row selected.
```

### SSSSSS

---

#### 설명

날짜 데이터 타입의 값의 마이크로 초를 표시한다.(0 ~ 999999)

#### 예제

```
iSQL> SELECT TO_CHAR (SYSDATE, 'SSSSSS') FROM dual;  
TO_CHAR (SYSDATE, 'SSSSSS')
```

```
-----  
490927  
1 row selected.
```

```
iSQL> SELECT TO_CHAR ( TO_DATE('1980-12-28 123456', 'YYYY-MM-DD  
SSSSSS'), 'SSSSSS' ) FROM dual;  
TO_CHAR ( TO_DATE('1980-12-28 123456', '
```

```
-----  
123456  
1 row selected.
```

### SSSSSSSS

---

#### 설명

초 + 마이크로 초를 나타낸다. 앞의 2 개의 숫자는 초를 나타내고,  
나머지 6 개의 숫자가 마이크로 초를 나타낸다. (0 ~ 59999999)

#### 예제

```
iSQL> SELECT TO_CHAR (SYSDATE, 'SSSSSSSS') FROM dual;  
TO_CHAR (SYSDATE, 'SSSSSSSS')
```

```
-----  
48987403  
1 row selected.
```

```
iSQL> SELECT TO_DATE ( '12.345678', 'SS.SSSSSS' ) FROM dual;  
TO_DATE ( '12.345678', 'SS.SSSSSS' )
```

```
-----  
2005/12/01 00:00:12  
1 row selected.
```

```
iSQL> SELECT TO_CHAR( TO_DATE( '12,345678', 'SS,SSSSSS' ), 'SSSSSS' ) FROM dual;
TO_CHAR( TO_DATE( '12,345678', 'SS,SSSSSS' )
_____
345678
1 row selected.
```

## WW

---

### 설명

일 년 중 몇 번째 주인지를 나타낸다. 1 월 1 일부터 그 주의 토요일까지가 그 해의 첫 번째 주이다. (1 ~ 54)

TO\_DATE 함수에서 사용할 수 없다.

### 예제

```
iSQL> SELECT TO_CHAR( '28-DEC-1980', 'WW' ) FROM dual;
TO_CHAR( '28-DEC-1980', 'WW' )
_____
53
1 row selected.
```

## W

---

### 설명

한 달 중 몇 번째 주인지를 나타낸다. 1 일부터 그 주의 토요일까지가 그 달의 첫 번째 주이다. (1 ~ 6)

TO\_DATE 함수에서 사용할 수 없다.

### 예제

```
iSQL> SELECT TO_CHAR( '28-DEC-1980', 'W' ) FROM dual;
TO_CHAR( '28-DEC-1980', 'W' )
_____
5
1 row selected.
```

## Y,YYY

---

### 설명

연도를 나타내는 숫자 중 임의의 위치에 ,(comma)를 삽입할 수 있다. 맨 앞이나 뒤에 와도 상관 없다.

TO\_DATE 함수에서 사용할 수 없다.

### 예제

```
iSQL> SELECT TO_CHAR( '28-DEC-1980', 'Y,YYY' ) FROM dual;
TO_CHAR( '28-DEC-1980', 'Y,YYY' )
_____
1,980
1 row selected.
```

## YYYY

---

### 설명

네 자리 숫자를 그대로 연도로 간주한다. (0 ~ 9999)

### 예제

```
iSQL> SELECT TO_CHAR ( '28-DEC-1980', 'YYYY' ) FROM dual;  
TO_CHAR ( '28-DEC-1980', 'YYYY' )
```

---

1980  
1 row selected.

```
iSQL> SELECT TO_DATE ( '28-DEC-1980', 'DD-MON-YYYY' ) FROM dual;  
TO_DATE ( '28-DEC-1980', 'DD-MON-YYYY' )
```

---

1980/12/28 00:00:00  
1 row selected.

## YY

---

### 설명

21 세기라고 가정하고 2000 을 더한 값을 연도로 간주한다. (2000 ~ 2099)

### 예제

```
iSQL> SELECT TO_CHAR ( '28-DEC-1980', 'YY' ) FROM dual;  
TO_CHAR ( '28-DEC-1980', 'YY' )
```

---

80  
1 row selected.

```
iSQL> SELECT TO_DATE ( '28-DEC-80', 'DD-MON-YY' ) FROM dual;  
TO_DATE ( '28-DEC-80', 'DD-MON-YY' )
```

---

2080/12/28 00:00:00  
1 row selected.

\* YYYY, YY, RRRR, RR 참고

[YYYY]: 숫자를 그대로 연도로 간주

'23-FEB-5' : 0005 년 2 월 23 일

'23-FEB-05' : 0005 년 2 월 23 일

'23-FEB-2005': 2005 년 2 월 23 일

'23-FEB-95' : 0095 년 2 월 23 일

[YY]: 2000 + YY

'23-FEB-5' : 2005 년 2 월 23 일

‘23–FEB–05’ : 2005 년 2 월 23 일

‘23–FEB–2005’ : 예러

‘23–FEB–95’ : 2095 년 2 월 23 일

‘23–FEB–05’ : 2005 년 2 월 23 일

‘23–FEB–2005’ : 예러

‘23–FEB–95’ : 2095 년 2 월 23 일

[RRRR]: 4 자리 숫자를 그대로 연도로 간주, 숫자가 50 미만인 경우 2000 을, 50 이상 100 미만인 경우 1900 을 더한다.

‘23–FEB–5’ : 2005 년 2 월 23 일

‘23–FEB–05’ : 2005 년 2 월 23 일

‘23–FEB–2005’ : 2005 년 2 월 23 일

‘23–FEB–95’ : 1995 년 2 월 23 일

‘23–FEB–100’ : 0100 년 2 월 23 일

‘23–FEB–0005’ : 0005 년 2 월 23 일

[RR]: 숫자가 50 미만인 경우 2000 을, 50 이상 100 미만인 경우 1900 을 더한다.

‘23–FEB–5’ : 2005 년 2 월 23 일

‘23–FEB–05’ : 2005 년 2 월 23 일

‘23–FEB–2005’ : 예러

‘23–FEB–95’ : 1995 년 2 월 23 일

## 예제

```
iSQL> CREATE TABLE timetbl(i1 INTEGER, t1 DATE, etc VARCHAR(10));
```

```
Create success.
```

```
iSQL> INSERT INTO timetbl VALUES (1, SYSDATE, 'Start');
```

```
1 row inserted.
```

```
iSQL> INSERT INTO timetbl VALUES (2, TO_DATE('2003-02-20 12:15:50',
```

```
'YYYY-MM-DD HH:MI:SS'), 'The end');
```

```
1 row inserted.
```

```
iSQL> SELECT TO_CHAR(T1, 'YYYY YY MM MON Mon mon DD HH MI SS
```

```
SSSSSS D DDD') Date_format FROM timetbl WHERE i1 = 2;
```

```
DATE_FORMAT
```

---

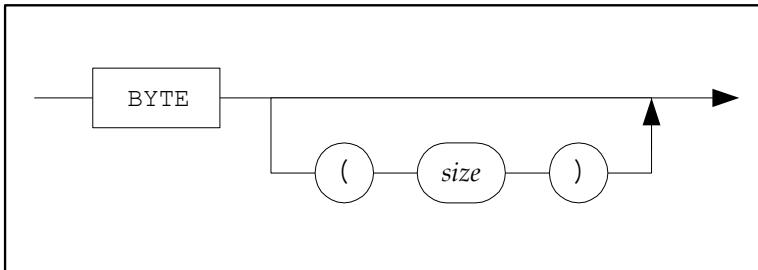
```
2003 03 02 FEB Feb 20 12 15 50 000000 5 051
```

1 row selected.

## 이진 데이터 타입

### BYTE

#### 흐름도



#### 구문

**BYTE [(size)] [[FIXED |] VARIABLE ( IN ROW size ) ]**

#### 설명

명시된 크기만큼 고정된 길이를 가지는 이진 데이터 타입이다.

BYTE 칼럼의 크기는 기본값으로 1 바이트이다. 최대 길이는, FIXED로 선언될 경우 한 페이지 크기(8K) 내에서 지정할 수 있으며, VARIABLE로 선언될 경우 32KB 즉 32768 바이트까지 저장할 수 있다.

'OFAE13'과 같이 문자형으로 표현 가능하며 이 때 사용하는 문자는 0에서 9 까지, A에서 F 까지의 문자이다.

입력 시와 검색 시 반드시 정의한 길이를 맞추어야 한다. 1 바이트는 2 개의 문자를 입력할 수 있다.

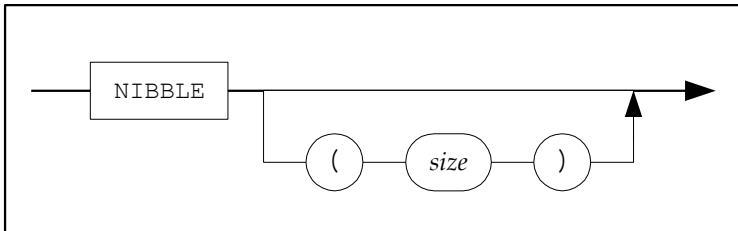
예를 들어 Byte (3) 이라고 정의 하면 '000000' 부터 'FFFFFF' 까지 입력할수 있다.

\* 소문자 'a'부터 'f'를 입력할 경우 대문자로 변환되어 저장된다.

\* 최소값(OxFF)은 널값으로 사용된다.

## NIBBLE

### 흐름도



### 구문

**NIBBLE [(size)]**

### 설명

명시된 크기만큼 가변 길이를 가지는 이진 데이타 타입이다.

NIBBLE 칼럼의 크기는 기본값으로 한 개의 문자 크기이며, 최대 255 크기까지 허용된다.

BYTE 와 마찬가지로 문자형으로 표현 가능하며 이 때 사용하는 문자는 0에서 9 까지, A에서 F 까지의 문자이다.

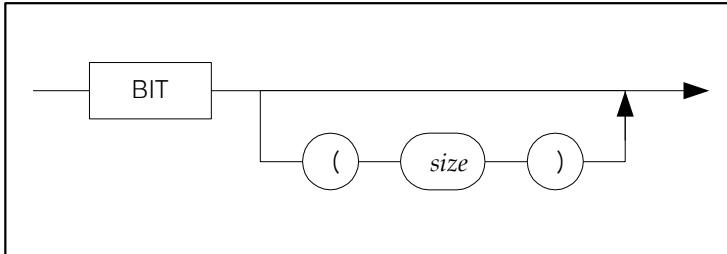
BYTE 와 달리 명시된 size 만큼의 문자만을 입력할 수 있다.

예를 들어 NIBBLE(6) 이라고 정의 하면 '000000' 부터 'FFFFFF' 까지 입력할 수 있다.

\* 소문자 'a'부터 'f'를 입력할 경우 대문자로 변환되어 저장된다.

## BIT

### 흐름도



### 구문

BIT [(size)]

### 설명

0 과 1로만 이루어진 고정 길이를 갖는 이진 데이터 타입이다.

BIT 칼럼의 크기는 기본값으로 1BIT이며, 최대 길이는 60576 비트이다.

명시한 길이보다 긴 스트링을 입력했을 경우 *Invalid length of data type* 에러가 나타난다.

명시한 길이보다 짧은 길이로 입력했을 때는 오른쪽을 0으로 채운다.

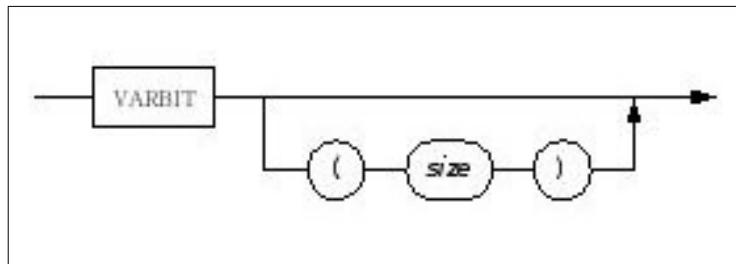
0 또는 1 이외의 값이 입력될 경우 *Invalid literal* 에러가 나타난다.

### 예제

```
iSQL> CREATE TABLE T1 ( I1 BIT(1), I2 BIT(5) );
Create success.
iSQL> INSERT INTO T1 VALUES ( BIT'1', BIT'011' );
1 row inserted.
iSQL> SELECT TO_CHAR(I1), TO_CHAR(I2) FROM T1;
TO_CHAR(I1)  TO_CHAR(I2)
_____
1  01100
1 row selected.
iSQL> INSERT INTO T1 VALUES ( BIT'1111', BIT'011' );
[ERR-2100D : Invalid length of the data type]
iSQL> INSERT INTO T1 VALUES ( BIT'1', BIT'1234' );
[ERR-21011 : Invalid literal]
```

## VARBIT

### 흐름도



### 구문

**VARBIT [(size)]**

### 설명

0 과 1 로만 이루어진 가변 길이를 갖는 이진 데이터 타입이다.

BIT 칼럼의 크기는 기본값으로 1Bit이며, 최대 길이는 131070 비트(128KBits)이다.

명시한 길이보다 긴 스트링을 입력했을 경우 Invalid length of data type 에러가 발생한다.

명시한 길이보다 짧은 스트링을 입력했을 경우 오른쪽을 0 으로 채운다.

0 또는 1 이외의 값이 입력될 경우 Invalid literal 에러가 발생한다.

### 예제

```
iSQL> CREATE TABLE T1 ( I1 VARBIT(1), I2 VARBIT(5) );
Create success.
iSQL> INSERT INTO T1 VALUES ( VARBIT'1', VARBIT'011' );
1 row inserted.
iSQL> SELECT TO_CHAR(I1), TO_CHAR(I2) FROM T1;
TO_CHAR(I1)  TO_CHAR(I2)
-----
1 011
1 row selected.
iSQL> INSERT INTO T1 VALUES ( VARBIT'1111', VARBIT'011' );
[ERR-2100D : Invalid length of the data type]
iSQL> INSERT INTO T1 VALUES ( VARBIT'1', VARBIT'1234' );
[ERR-21011 : Invalid literal]
```

# LOB 데이터 타입

## 개요

LOB(Large Object)은 대용량 데이터를 저장할 수 있는 데이터 타입이다. LOB으로 저장이 가능한 데이터 크기는 최대 2G 이다.

테이블을 생성할 때 다른 타입들과 달리 길이를 명시하지 않는다. 그리고 하나의 테이블에 하나 이상의 LOB 타입 칼럼을 정의할 수 있다.

LOB 데이터 타입은 이미지, 동영상 파일들과 같은 이진 데이터를 저장하는 BLOB(Binary Large Object)과 문자열 데이터를 저장하는 CLOB(Character Large Object)으로 구분된다.

## LOB의 특징

알티베이스가 제공하는 LOB은 다음과 같은 특징이 있다.

- 데이터 저장 기능
- 부분 읽기(Partial Read)
- 부분 갱신(Partial Update)
- 디스크 LOB 파티션

### 데이터 저장 기능

ODBC SQLPutLob 함수 또는 JDBC File Stream 을 이용하여 CLOB, BLOB 데이터를 저장한다.

### 부분 읽기(Partial Read)

LOB 데이터의 특정 구간에 대한 데이터 조각을 읽는 기능이다. SQLGetLob 함수를 이용하여 특정 오프셋의 크기를 읽는다.

### 부분 갱신(Partial Update)

LOB 데이터의 특정 구간에 대한 특정 크기 데이터를 다른 데이터로 갱신(replace)시킨다.

### 디스크 LOB의 파티션

디스크 LOB 데이터는 테이블이 속한 테이블스페이스가 아닌 다른 디스크 테이블스페이스로 저장이 가능하며, 마치 파티션을 이용하는 것처럼 구성할 수 있다.

## LOB 칼럼의 저장

메모리 테이블의 LOB 데이터는 대부분의 경우 레코드 영역 밖의 가변 영역에 저장된다. 또한 LOB 칼럼의 크기가 크지 않을 때에는 in row 옵션을 사용하여 레코드 영역(고정 영역) 안에 저장하기도 한다.

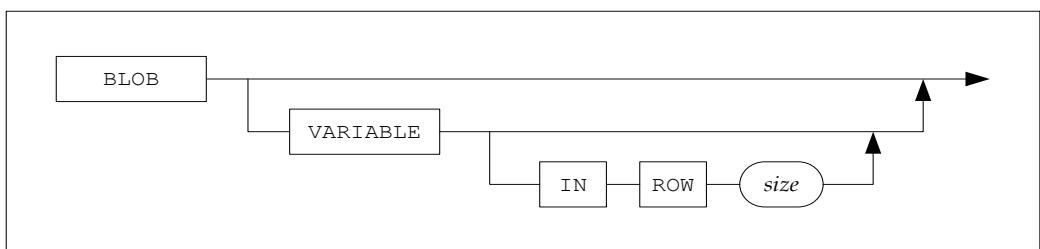
그러나 디스크 테이블은 크기와 상관없이 항상 가변 영역에 저장한다.

그러나 가변 영역에 저장되는 LOB 칼럼의 데이터는 크기가 매우 크기 때문에, 레코드 영역과 같은 테이블스페이스에 저장되는 것은 공간 사용 측면에서 효율성을 떨어뜨린다.

디스크 테이블은 LOB 칼럼 데이터를 LOB 칼럼이 속한 테이블과 별도의 테이블스페이스에 저장할 수 있다. 그러나 메모리 테이블은 별도로 저장할 수 없고 테이블과 동일한 테이블스페이스에만 저장할 수 있다.

## BLOB

### 흐름도



### 구문

**BLOB [ VARIABLE ( IN ROW *size* ) ]**

### 설명

BLOB은 이진형 대용량 데이터를 저장하기 위한 것으로, 2GB 크기 내에서 가변 길이를 가지는 이진형 데이터 타입이다.

BLOB은 FIXED|VARIABLE을 명시할 수 있으나 모두 VARIABLE로 처리되어 실제로 의미가 없다.

메모리 테이블에 저장된 BLOB은 데이터의 크기가 64 바이트 이하일 경우, 테이블의 고정 영역에 저장되며 64 바이트를 초과하면 가변 영역에 저장된다. VARIABLE\_LOB\_COLUMN\_IN\_ROW\_SIZE 프로퍼티로 고정 영역에 저장되는 디폴트 크기를 명시할 수 있다. 디스크 테이블에 대해서는 크기에 관계없이 가변 영역에 저장된다.

#### IN ROW size

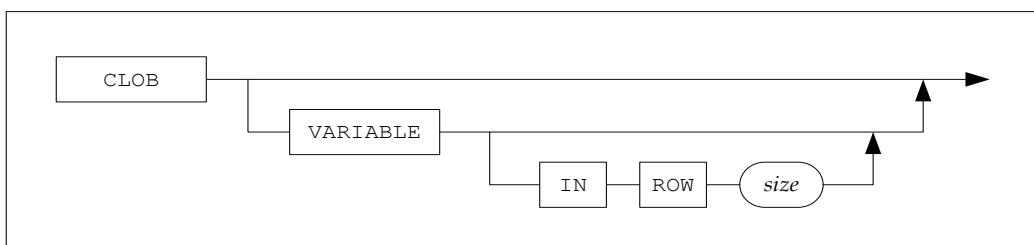
메모리 테이블의 경우, VARIABLE 지정시에 ‘IN ROW’ 구문을 이용해서 고정 데이터 영역에 들어갈 데이터 길이와 가변 영역에 들어갈 길이를 지정할 수 있다. 예를 들어, ‘BLOB in row 200’으로 설정된 칼럼의 경우, 삽입되는 데이터 길이가 200 바이트 이하이면 고정 영역에, 200 바이트를 초과하면 가변 영역에 저장된다.

특별한 경우가 아니면 매번 in row 크기를 지정할 필요 없이 VARIABLE\_LOB\_COLUMN\_IN\_ROW\_SIZE 프로퍼티에 디폴트 in row 크기를 지정하고 테이블 생성 시에는 생략할 수 있다. 해당 프로퍼티에 대한 상세한 설명은 Starting User's Manual의 알터베이스 프로퍼티 부분을 참조한다.

---

## CLOB

### 흐름도



### 구문

CLOB [ VARIABLE ( IN ROW *size* ) ]

### 설명

CLOB은 문자형 대용량 데이터를 저장하기 위한 것으로, 2GB 크기 내에서 가변 길이를 가지는 문자형 데이터 타입이다.

CLOB도 BLOB이나 VARCHAR 타입처럼 FIXED|VARIABLE을 명시할 수 있으나 모두 VARIABLE로 처리되어 실제로 의미가 없다.

메모리 테이블에 저장된 CLOB은 BLOB과 마찬가지로, 데이터의 크기가 64 바이트 이하일 경우, 테이블의 고정 영역에 저장되며 64

바이트를 초과하면 가변 영역에 저장된다.

VARIABLE\_LOB\_COLUMN\_IN\_ROW\_SIZE 프로퍼티로 FIXED 영역에 저장되는 디폴트 크기를 명시할 수 있다. 디스크 테이블에 대해서는 크기에 관계없이 VARIABLE 영역에 저장된다.

#### *IN ROW size*

메모리 테이블의 CLOB 칼럼의 경우, VARIABLE 지정시에 ‘IN ROW’ 구문을 이용해서 고정 데이터 영역에 들어갈 데이터 길이와 가변 영역에 들어갈 길이를 지정할 수 있다. 이에 대한 세부내용은 BLOB 과 같다.

---

### 제한 사항

- Stored Procedure나 트리거에서는 LOB 탑입을 사용할 수 없다.
- 커서에서 LOB 탑입을 사용할 수 없다.
- 휘발성 테이블과 디스크 임시 테이블스페이스에서는 LOB 탑입을 사용할 수 없다.
- 디스카드 된 테이블스페이스의 LOB 칼럼은 사용이 불가능하다.
- LOB 탑입은 파티션 컬럼으로 사용할 수 없다. 파티션 컬럼은 대소 비교가 가능해야 한다.
- LOB 칼럼에는 인덱스를 생성할 수 없다.

## 공간 데이터 타입

공간 데이터 타입을 SQL 형식으로 정의할 때 현재 지원되는 타입은 Geomerty 한가지 뿐이며, 이 Geometry 타입은 내부적으로 7 개의 하위 데이터 타입인 Point, LineString, Polygon, GeomCollection, MultiPolygon, MultiLineString, MultiPoint 이다. 공간 데이터 타입에 관한 자세한 내용은 *Spatial SQL User's Manual* 을 참조한다

## Part II

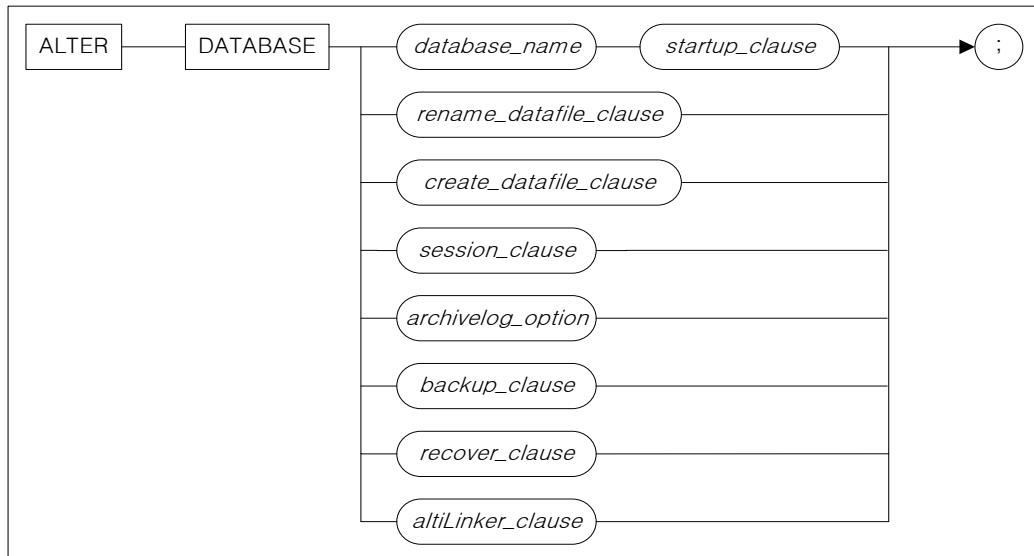
### 3. 데이터 정의어

이 장에서는 데이터베이스 오브젝트를 생성하기 위해서 사용하는 SQL 문장인 DDL 문장의 문법과 특징을 사용 예제를 들어서 상세하게 설명하고 있다.

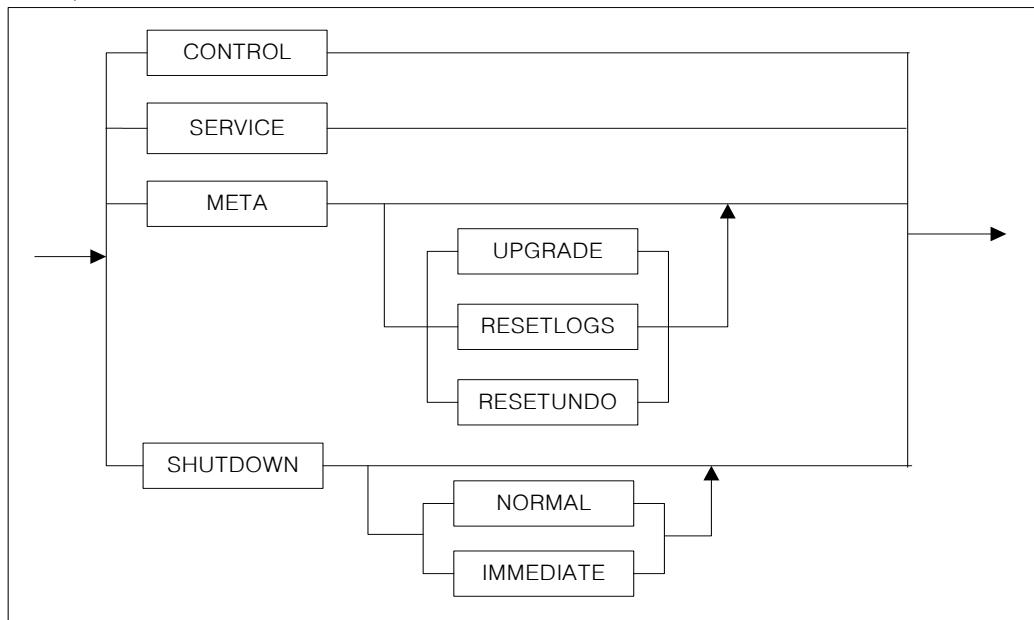
# ALTER DATABASE

## 구문

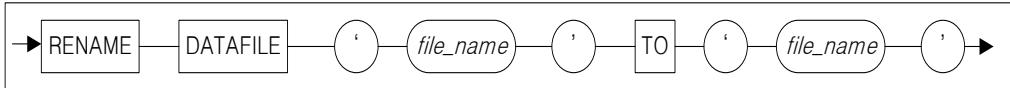
*alter\_database ::=*



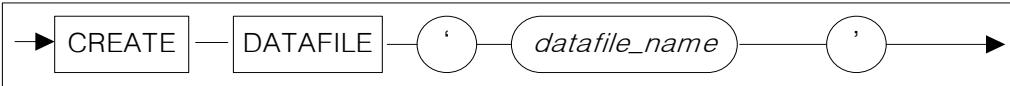
*startup\_clause ::=*



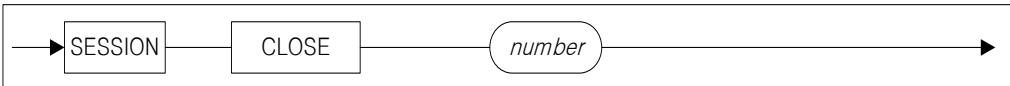
*rename\_clause* ::=



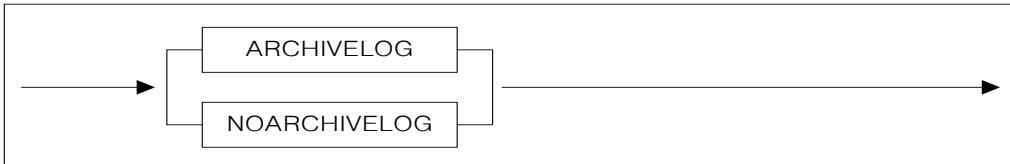
*create\_clause* ::=



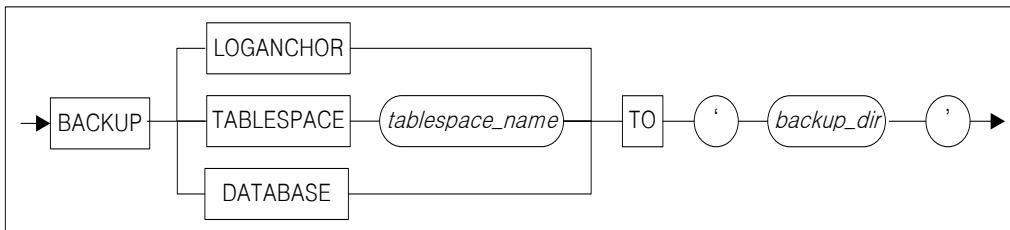
*session\_clause* ::=



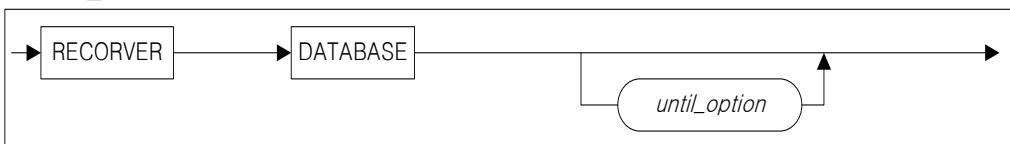
*archivelog\_option* ::=



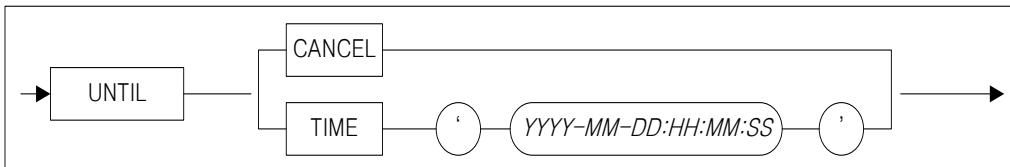
*backup\_clause* ::=



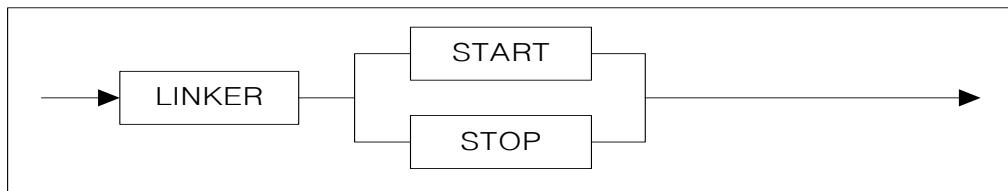
*recover\_clause* ::=



*until\_option* ::=



*alterLinker\_clause ::=*



## 전제 조건

데이터베이스 다단계 구동에서 서비스 전 단계에서만 수행할 수 있는 SQL 문으로 SYS 사용자가 -sysdba 관리자 모드에서만 수행할 수 있다.

## 설명

기존 데이터베이스의 정의를 변경하는 구문이다.

*database\_name*

변경될 데이터베이스 이름을 명시한다.

*startup\_clauses*

데이터베이스를 open 하여 사용자들이 접근 사용할 수 있도록 한다.

CONTROL

데이터베이스 구동 단계를 CONTROL 단계로 변경한다.

데이터베이스 다단계 구동(startup) 시 DB 미디어 복구와 이중화 수행을 할 수 있는 단계이다. 또한, 테이블스페이스를 Discard 할 수 있는 단계이다. 데이터베이스 다단계 구동 단계에 대한 자세한 설명은 Administrator's Manual 을 참조한다.

다음 단계를 수행하기 위해 ALTER DATABASE *dababase\_name* META; 구문을 수행해야 한다.

SERVICE

데이터베이스 구동 단계를 SERVICE 단계로 변경한다. 데이터베이스 다단계 구동 시 메모리 DB 와 디스크 DB 에 로딩을 수행하고, 이중화 또는 SNMP 등의 확장 서비스를 모두 구동할 수 있다. 이 옵션을 수행 시 모든 복구가 완료된 후 정상 서비스를 하는 단계이다.

META

데이터베이스 구동 단계를 META 단계로 변경한다. 데이터베이스 다단계 구동할 때 메타를 로딩하는 단계이다.

다음 단계를 수행하기 위해 ALTER DATABASE database\_name SERVICE; 구문을 수행한다.

#### META UPGRADE

데이터베이스 구동 단계를 META UPGRADE 단계로 변경한다.  
데이터베이스 다단계 구동 시 메타를 갱신하는 단계로, 모든 복구가 완료된 상태이다.

다음 단계를 수행하기 위해 ALTER DATABASE database\_name SERVICE; 구문을 수행해야 한다.

#### META RESETLOGS

데이터베이스 다단계 startup 시 컨트롤 단계에서 불완전 복구를 한 경우, 서버를 정상 구동하기 위해 필요한 작업이다. 불완전 복구로 불필요한 로그 레코드들을 초기화한다.

다음 단계를 수행하기 위해 ALTER DATABASE database\_name SERVICE; 구문을 수행해야 한다.

#### META RESETUNDO

SYS\_TBS\_DISK\_UNDO 테이블스페이스를 초기화한다. 그러나 파일 크기는 재설정하지 않는다. 단, 데이터베이스의 무결성을 디스크 가비지 콜레션이 수행된 것을 확인하고, 서버를 정상 종료한 이후에 이 구문을 실행한다.

#### SHUTDOWN NORMAL

서버에 접속한 모든 클라이언트의 연결이 정상적으로 해제될 때까지 대기한 후 서버를 정상 종료한다.

#### SHUTDOWN IMMEDIATE

서버에 접속된 모든 클라이언트의 연결을 강제로 해제한 후, 서버를 정상 종료한다.

#### RENAME DATAFILE

데이터베이스에 속한 데이터 파일을 새로운 이름으로 변경한다. TO file\_name 의 새로운 데이터 파일이 미리 생성되어야 한다.

#### CREATE DATAFILE

데이터 파일이 유실되었을 때, 로그 앤커의 정보를 참고하여 데이터 파일을 생성한다. 이 구문을 실행한 후에는 매체 완전 복구를 수행하여 데이터 파일을 복구한다.

datafile\_path 에 데이터 파일을 생성할 디렉토리의 상대 경로 또는 절대 경로를 입력한다. 상대 경로를 입력할 때는 \$ALTIBASE\_HOME/dbs/에서 입력한다.

#### SESSION CLOSE

세션을 강제로 종료시킨다. SESSION CLOSE 다음에 session\_id 을 넣어주면된다. 이때에는 Transaction 이 바로 멈춰진다.

\* 참고: 단계마다의 상태 변화는 위에서 기술한 순서대로 가능하고, 단계 서로간의 jump 가 가능하다.

shutdown normal(immediate)은 SERVICE 단계에서만 가능하고, 다른 모드에서는 shutdown abort 만 가능하다.

알티베이스 구동단계에 대해서는 Altibase Administrator Manual 을 참조한다.

#### *archivelog\_option*

서버 컨트롤 단계에서 데이터베이스 모드를 아카이브로그 모드 또는 노아카이브로그 모드로 변경한다.

#### BACKUP LOGANCHOR

데이터베이스를 아카이브로그 모드로 운영할 때, 서비스를 중지하지 않은 상태에서 로그 앵커를 온라인 백업한다.

#### BACKUP TABLESPACE

데이터베이스를 아카이브로그 모드로 운영할 때, 서비스를 중지하지 않은 상태에서 지정된 테이블스페이스를 백업 디렉토리에 백업한다.

#### BACKUP DATABASE

데이터베이스를 아카이브로그 모드로 운영할 때, 서비스를 중지하지 않은 상태에서 모든 메모리 테이블스페이스, 디스크 테이블스페이스, 로그앵커가 백업된다.

#### RECOVER DATABASE

매체 완전복구를 한다. 아카이브 로그 디렉토리의 로그 파일을 판독하여 매체 오류가 발생한 데이터 파일들을 현재 시점의 데이터 파일로 복구한다.

#### RECOVER DATABASE UNTIL TIME

데이터가 백업된 시점부터 그 이후의 특정 과거 시점까지에 대한 전체 데이터베이스를 복구한다.

#### RECOVER DATABASE UNTIL CANCEL

데이터가 백업된 시점부터 그 이후의 유효한 로그 파일 또는 로그 레코드를 포함한 시점까지 전체 데이터베이스를 복구한다.

#### *AltiLinker\_clause*

데이터베이스 링크에서 원격 서버와의 데이터 교환을 매개해 주는 AltiLinker 프로세스를 종료시키거나 재시작한다.

#### LINKER START

AltiLinker 프로세스를 시작한다. 단, AltiLinker 가 존재하지 않아야 한다.

LINKER STOP

AltiLinker 프로세스를 종료한다. 단, 종료 실행시 데이터베이스 링크를 사용하는 트랜잭션이 없어야 한다. 데이터베이스 링크를 사용하는 트랜잭션이 존재하면, 이 연산은 실패한다.

---

## 예제

〈질의〉 다음은 다단계 startup 시 SERVICE 단계를 이용해서 데이터베이스 mydb 가 정상 서비스 수행이 가능하도록 하는 구문이다.

```
iSQL> ALTER DATABASE mydb SERVICE;
```

〈질의〉 아카이브로그 모드로 데이터베이스를 운영한다.

```
iSQL> ALTER DATABASE ARCHIVELOG;
```

〈질의〉 불완전 복구된 데이터베이스를 정상 구동시킨다.

```
iSQL> ALTER DATABASE mydb META RESETLOGS;
```

〈질의〉 SYS\_TBS\_DISK\_UNDO 테이블스페이스를 초기화한다.

```
iSQL> ALTER DATABASE mydb META RESETUNDO;
```

〈질의〉 SYS\_TBS\_DISK\_DATA 테이블스페이스를 /altibase\_backup 에 백업한다.

```
iSQL> ALTER DATABASE TABLESPACE SYS_TBS_DISK_DATA TO  
'/altibase_backup/' ;
```

〈질의〉 백업된 시점부터 2008년 2월 16일 오후 12시 시점으로 데이터베이스를 복원한다.

```
iSQL> ALTER DATABASE RECOVER DATABASE UNTIL TIME '2008-02-  
16:12:00:00' ;
```

〈질의〉 백업된 시점부터 유실된 로그파일 20001 번 이전인 로그파일 20000 번까지 데이터베이스를 복원한다.

```
iSQL> ALTER DATABASE RECOVER DATABASE UNTIL CANCEL;
```

〈질의〉 데이터베이스 링크를 위한 AltiLinker 를 구동한다.

```
iSQL> ALTER DATABASE LINKER START;
```

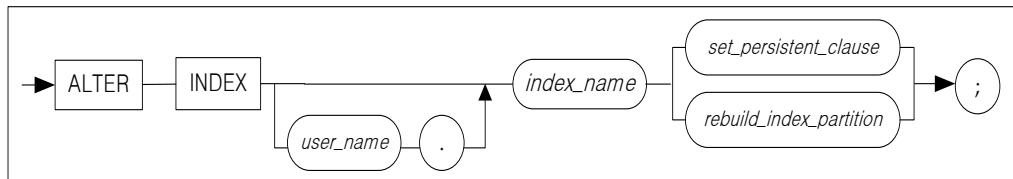
〈질의〉 AltiLinker 를 종료한다.

```
iSQL> ALTER DATABASE LINKER STOP;
```

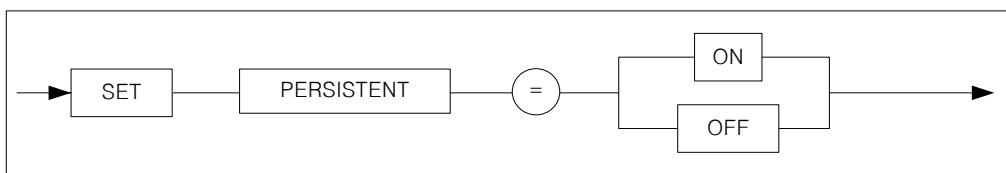
# ALTER INDEX

## 구문

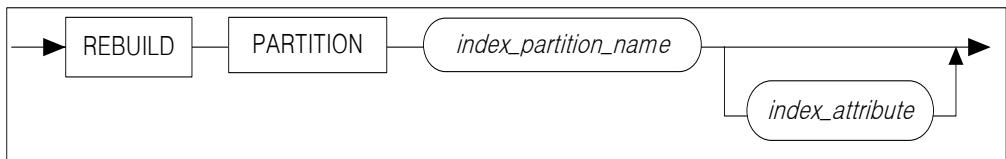
*alter\_index ::=*



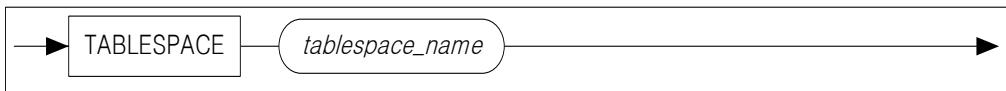
*set\_persistent\_clause ::=*



*rebuild\_index\_partition ::=*



*index\_attribute ::=*



## 전제 조건

SYS 사용자 이거나 인덱스가 현재 사용자의 스키마에 속하거나 또는  
ALTER ANY INDEX 시스템 권한을 가진 사용자만이 인덱스를  
변경할 수 있다.

## 설명

기존 인덱스 정의를 변경한다.

*user\_name*

변경될 인덱스의 소유자 이름을 명시한다.

생략하면 알티베이스는 현재 세션에 연결된 사용자의 스키마에 속한 것으로 간주한다.

*index\_name*

변경될 인덱스의 이름을 명시한다.

SET PERSISTENT 절

PERSISTENT 색인을 생성 후 PERSISTENT 색인을 변경하는 문이다. 자세한 내용은 CREATE INDEX 참고한다

*rebuild\_index\_partition*

지정한 테이블스페이스에 인덱스 파티션을 재구축한다.

*index\_attribute*

테이블스페이스 명을 명시한다.

---

## 주의 사항

변경할 인덱스가 이중화 테이블에 걸려있는 경우에는 사용할 수 없다. 그러나 ‘ALTER INDEX SET PERSISTENT = ON/OFF’, ‘ALTER INDEX REBUILD PARTITION’은 사용할 수 있다.

---

## 예제

### PERSISTENT 인덱스 변경

〈질의〉 인덱스 emp\_idx1에 PERSISTENT 인덱스를 추가하라.

```
iSQL> ALTER INDEX emp_idx1 SET PERSISTENT = ON;  
Alter success.
```

〈질의〉 인덱스 const1에 PERSISTENT 인덱스를 추가하라.

```
iSQL> ALTER INDEX const1 SET PERSISTENT = ON;
```

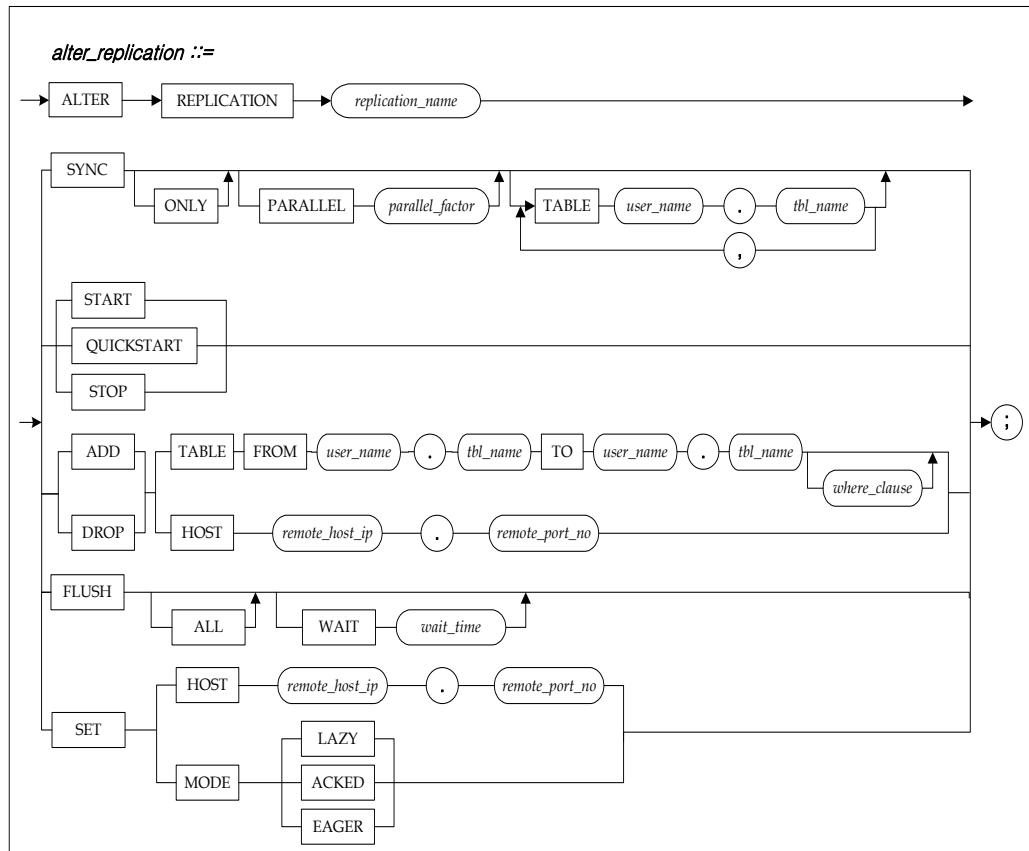
### 인덱스 파티션 재구축

〈질의〉 인덱스 파티션 IDX\_P5를 테이블스페이스 TBS1에 재구축하라.

```
iSQL> ALTER INDEX IDX1 REBUILD PARTITION IDX_P5 TABLESPACE TBS1;
```

# ALTER REPLICATION

## 구문



## 설명

이중화 생성(CREATE REPLICATION) 후 이중화의 동작을 제어하는 문장으로 다음과 같은 종류가 있다. 이중화에 관한 자세한 내용은 *Replication User's Manual* 참조한다.

*replication\_name*

이중화 이름을 명시한다.

SYNC

지역 서버에 있는 이중화 테이블들의 모든 데이터를 원격 서버의

해당 테이블로 전송한 후 이중화를 시작한다.

#### SYNC ONLY

지역 서버에 있는 이중화 테이블들의 모든 데이터를 원격 서버의 해당 테이블로 전송한다. 송신 쓰레드는 생성되지 않는다.

#### PARALLEL parallel\_factor

*Parallel\_factor* 값은 생략 가능하며, 생략할 경우 1로 인식된다.

*Parallel\_factor*의 최대값은 CPU 개수 \* 2이며, 최대값을 초과하여 설정해도 최대 값을 넘지 못한다. 0 또는 음의 값을 설정한 경우엔 오류 메시지가 나타난다.

#### TABLE user\_name.table\_name

지역서버 이중화 테이블 중에서 SYNC 대상 테이블을 지정한다.  
이중화 시작 시에는 마지막 이중화 시점부터 이중화를 시작하며,  
TABLE 절을 사용하지 않았을 때에는 현재 로그의 위치부터  
이중화를 시작한다.

#### START

가장 최근의 마지막 이중화 시점부터 이중화를 시작한다.

#### QUICKSTART

현재 시점부터 이중화를 시작한다.

#### STOP

이중화를 중지한다.

#### ADD TABLE

이중화 할 테이블을 추가한다. 이중화가 중지되어 있는 상태에서  
이중화 할 테이블을 추가할 수 있다.

#### *where\_clause*

SELECT 구문의 *where\_clause*를 참고한다.

이중화 조건절은 WHERE user\_name.table\_name.column\_name  
{< | > | ◇ | >= | <= | = | !=} value [{AND | OR} ... ] 사용하여  
조건에 해당하는 로그만 이중화 대상으로 한다. 단, 이중화 조건절은  
ADD TABLE 에서만 사용할 수 있다.

#### DROP TABLE

이중화 테이블을 삭제한다. 이중화가 중지되어 있는 상태에서 이중화  
테이블을 삭제할 수 있다.

#### *user\_name*

이중화 테이블의 소유자 이름을 명시한다.

#### *tbl\_name*

이중화 테이블 이름을 명시한다.

## FLUSH

이중화 sender 쓰레드에 의해서 현재 로그 까지의 변경 내용이 상대방 서버에 전송 되도록 wait\_time 초 만큼 기다린다. 만약, All 이 같이 사용되면 현재 로그가 아닌 가장 최근 로그 까지의 변경 내용이 상대방 서버에 전송 되도록 기다린다.

---

## 주의 사항

### 전제 조건

- 입력/수정/삭제시 충돌(confliction)이 발생하면 무시(discard) 처리하고 에러 로그 파일에 남긴다.

\* 에러 로그 파일에 제외되는 경우

deadlock: 원격 서버에서 데드락으로 인해 이중화 트랜잭션이 룰백될 경우에는 에러 로그를 남기지 않는다.

network error: 이중화 도중 네트워크(TCP/IP) 에러로 인한 데이터 손실을 막을수 없다. 또한 에러 로그도 남기지 않는다.

- 이중화 작업 도중 발생한 에러에 대해 부분 룰백한다. 즉 여러 개의 자료 입력 중 한 개의 중복 데이터가 있으면 중복 데이터만 취소하고 나머지는 완료할 수 있다.
- 이중화 속도는 서비스 속도보다 매우 느릴 수 있다.
- 알티베이스 프로퍼티의 AUTO\_REMOVE\_ARCHIVE\_LOG = 1로 세팅한 경우라도 완료되지 않은 이중화 작업 로그는 로그 파일에서 삭제되지 않는다. 만일 0으로 세팅할 경우 아래의 5번은 이중화 작업에 영향이 없다.
- 현재 이중화 송신 쓰레드가 작업하는 로그 파일과 현재 로그 파일의 차이가 알티베이스 프로퍼티의 최대 로그 파일 개수(REPLICATION\_MAX\_LOGFILE)보다 큰 시점에 체크 포인트가 발생하면 이중화 작업이 완료되지 않은 로그 파일도 삭제되며, 이중화 작업은 이전의 로그 파일은 무시하고 현재 로그 파일부터 다시 시작한다.

### 데이터 제약조건

- 복제되는 테이블에 대해 반드시 기본키가 존재해야 한다.
- 복제되는 테이블에 대해 기본키에 대한 수정이 없어야 한다.
- 양쪽 테이블의 칼럼정보, 기본키, NOT NULL 정보가 동일해야 한다.

## 연결 제약조건

- 최대 이중화 연결은 하나의 알티베이스 데이터베이스에 32개이다.

---

## 예제

**이중화 이름을 *rep1* 이라고 가정하고 이중화를 시작하는 질의는 다음 3 가지 중 한가지를 선택할 수 있다.**

〈질의〉 지역서버의 데이터를 원격서버로 전송한 후 이중화를 시작하라.

```
iSQL> ALTER REPLICATION rep1 SYNC;  
Alter success.
```

〈질의〉 가장 최근의 마지막 *rep1* 이중화 시점부터 *rep1* 이중화를 시작하라.

```
iSQL> ALTER REPLICATION rep1 START;  
Alter success.
```

〈질의〉 현재부터 이중화를 시작하라.

```
iSQL> ALTER REPLICATION rep1 QUICKSTART;  
Alter success.
```

**이중화 이름을 *rep1* 이라고 가정하고 이중화 테이블을 삭제한다.**

〈질의〉 이중화 테이블 *employee*를 삭제하라.

```
iSQL> ALTER REPLICATION rep1 STOP;  
Alter success.  
iSQL> ALTER REPLICATION rep1 DROP TABLE FROM sys.employee TO  
sys.employee;  
Alter success.
```

**이중화 이름을 *rep1* 이라고 가정하고 이중화 할 테이블을 추가한다.**

〈질의〉 이중화 할 테이블 *employee*를 추가하라.

```
iSQL> ALTER REPLICATION rep1 STOP;  
Alter success.  
iSQL> ALTER REPLICATION rep1 ADD TABLE FROM sys.employee TO  
sys.employee;  
Alter success.
```

**이중화 이름을 *rep1* 이라고 가정하고 이중화를 중지하는 질의이다.**

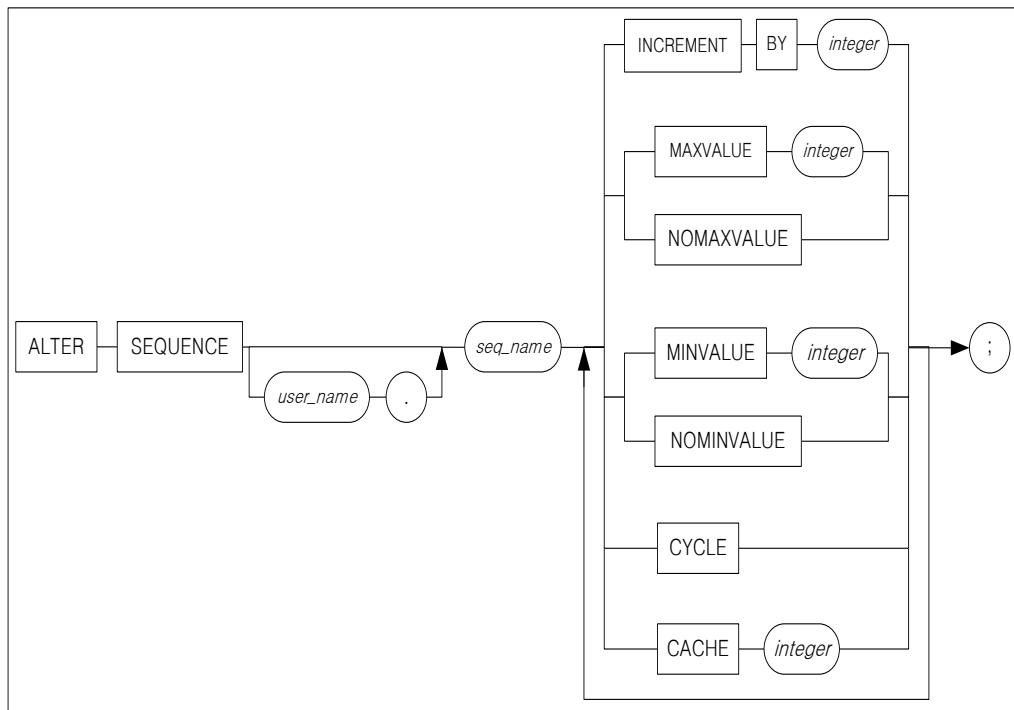
〈질의〉 이중화를 중지하라.

```
iSQL> ALTER REPLICATION rep1 STOP;  
Alter success.
```

# ALTER SEQUENCE

## 구문

*alter\_sequence ::=*



## 전제 조건

SYS 사용자이거나 시퀀스가 사용자의 스키마에 속하거나 시퀀스에 대해 ALTER 객체 접근 권한을 가지거나 또는 ALTER ANY SEQUENCE 시스템 권한을 가진 사용자만이 시퀀스를 변경할 수 있다.

## 설명

시퀀스 생성(CREATE SEQUENCE) 후 시퀀스의 정의를 변경하는 문으로 메타내 해당 시퀀스의 정의가 변경된다. CREATE SEQUENCE 참고한다.

*user\_name*

변경될 시퀀스의 소유자 이름을 명시한다. 생략하면 알터베이스는 현재 세션에 연결된 사용자의 스키마에 속한 것으로 간주한다.

*seq\_name*

변경될 시퀀스 이름을 명시한다.

INCREMENT BY

시퀀스 값의 증감분

MAXVALUE

시퀀스의 최대값

MINVALUE

시퀀스의 최소값

CYCLE

시퀀스가 MAXVALUE 또는 MINVALUE 한계에 도달하면 해당 시퀀스의 추가 값을 할당하기 위하여 오름차순 시퀀스인 경우는 minimum value를 내림차순 시퀀스인 경우는 maximum value를 생성한다.

CACHE

해당 시퀀스 값을 더 빠르게 액세스하기 위하여 명시된 값 만큼 시퀀스 값을 메모리에 캐시한다. 캐시는 시퀀스를 처음 참조할 때 채워지며 다음에 시퀀스 값을 요청할 때마다 캐시 된 시퀀스에서 검색된다. 마지막 시퀀스를 사용한 이후의 시퀀스 요청은 시퀀스의 다른 캐시를 메모리로 가져온다. 시퀀스 생성시 기본값은 20이다.

---

## 주의 사항

시퀀스 정의 변경의 경우 이미 시퀀스 생성 이후 이므로 START WITH 절을 사용할 수 없다.

---

## 예제

\* CREATE SEQUENCE 참조한다.

〈질의〉 시퀀스 *seq1* 을 최소값 0, 최대값 100 을 가지고 1 씩 증가하도록 변경하라.

```
iSQL> ALTER SEQUENCE seq1  
      INCREMENT BY 1  
      MINVALUE 0
```

```
    MAXVALUE 100;
    Alter success.
```

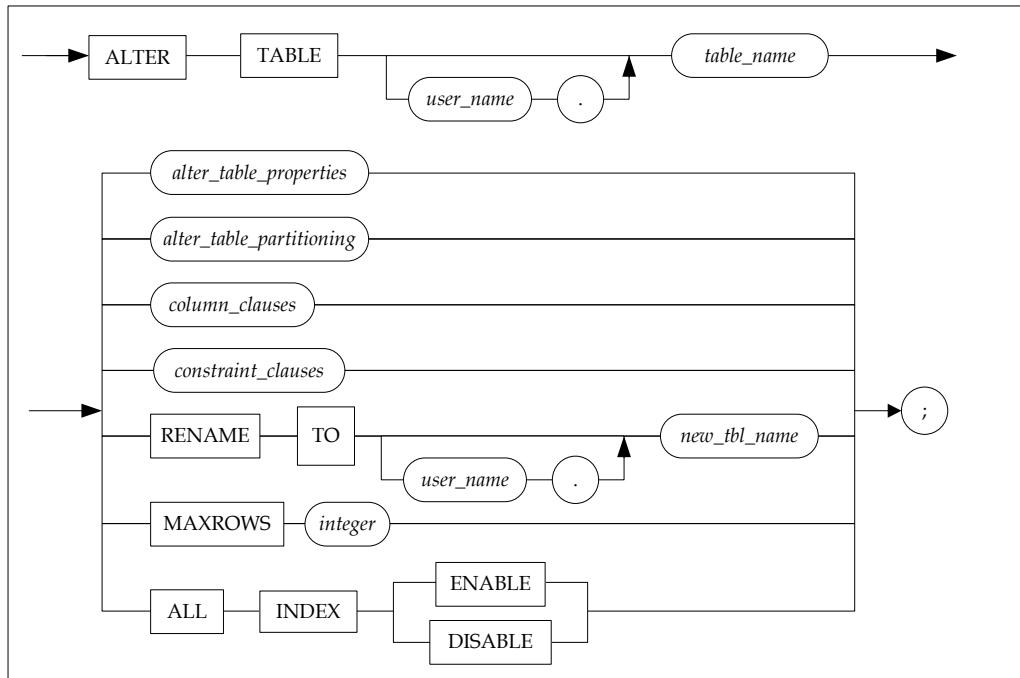
〈질의〉 시퀀스 *seq2*의 최소값, 최대값을 무한대로 변경하라.

```
iSQL> ALTER SEQUENCE seq2
      NOMAXVALUE
      NOMINVALUE;
      Alter success.
```

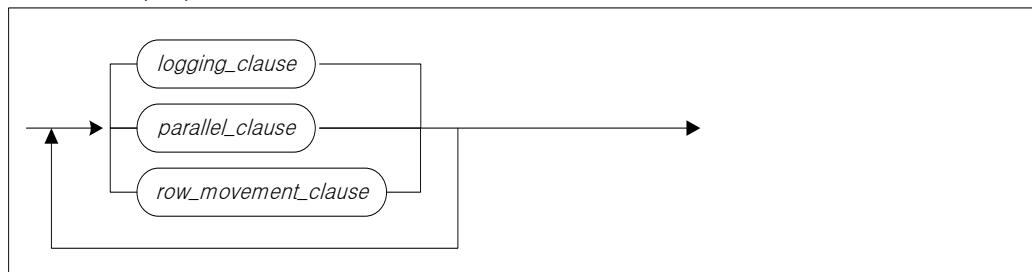
# ALTER TABLE

## 구문

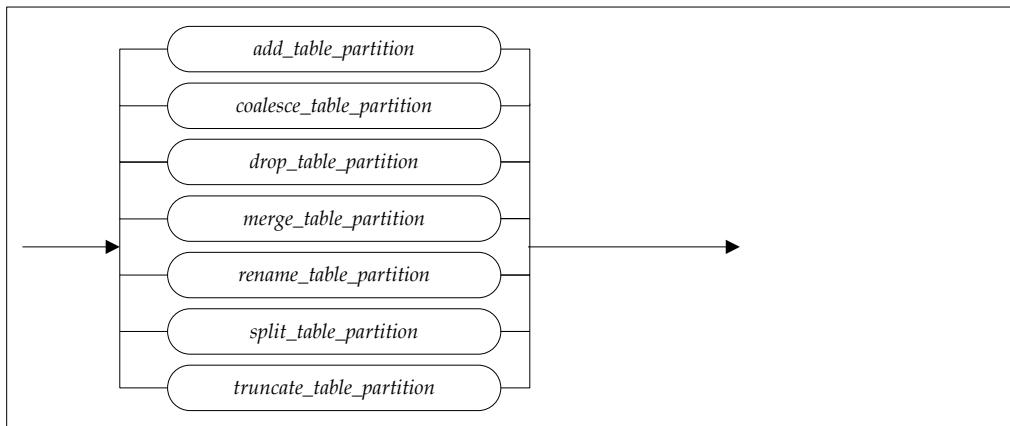
*alter\_table ::=*



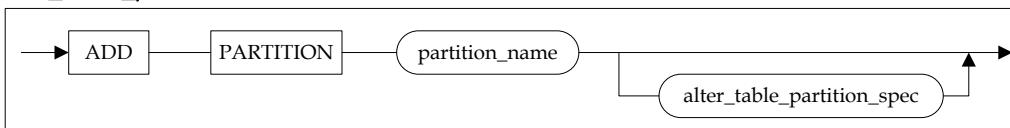
*alter\_table\_properties ::=*



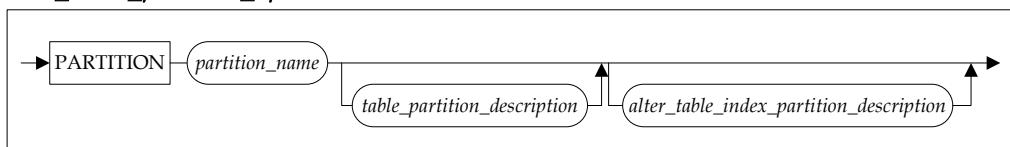
*alter\_table\_partitioning ::=*



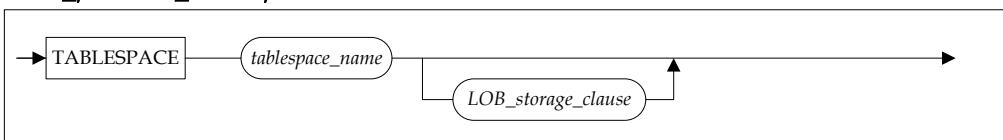
*add\_table\_partition ::=*



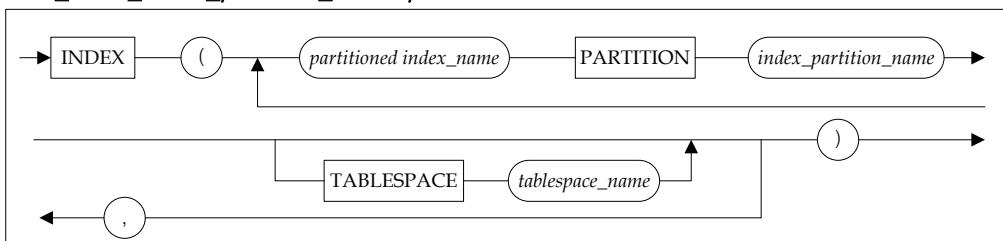
*alter\_table\_partition\_spec ::=*



*table\_partition\_description ::=*



*alter\_table\_index\_partition\_description ::=*



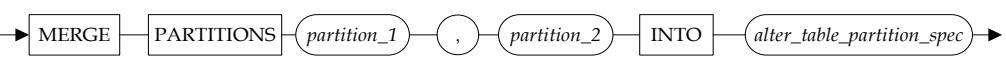
*coalesce\_table\_partition ::=*



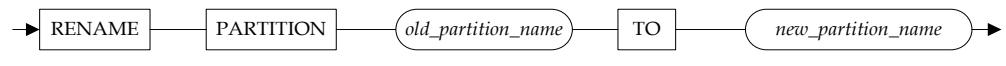
*drop\_table\_partition ::=*



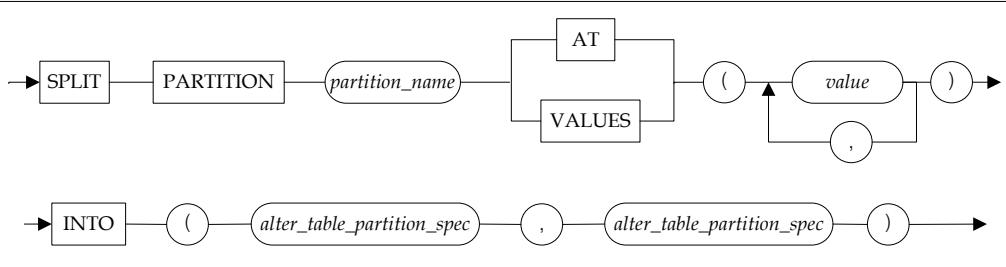
*merge\_table\_partition ::=*



*rename\_table\_partition ::=*



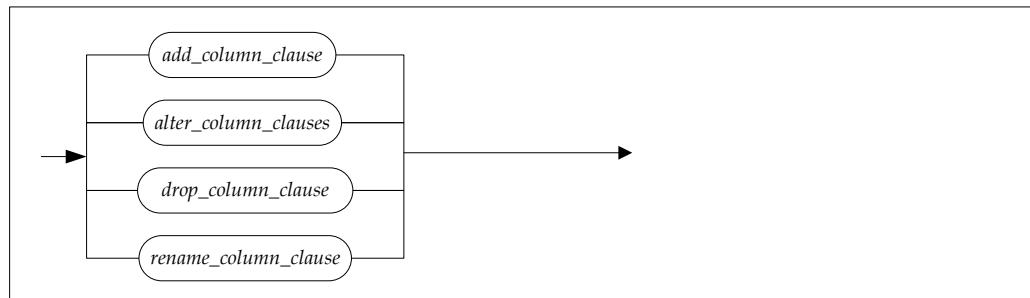
*split\_table\_partition ::=*



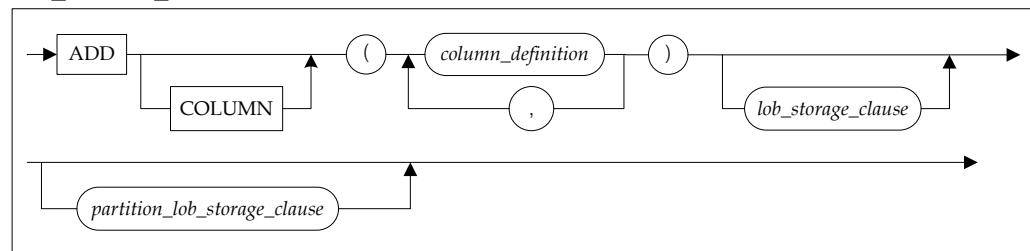
*truncate\_table\_partition ::=*



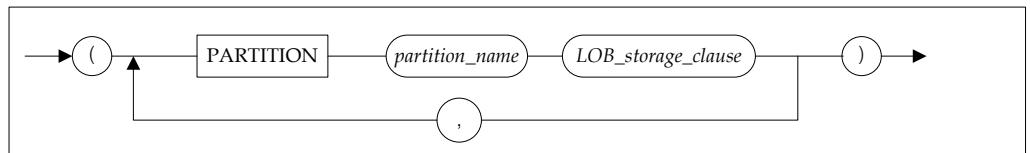
*column\_clauses*::=



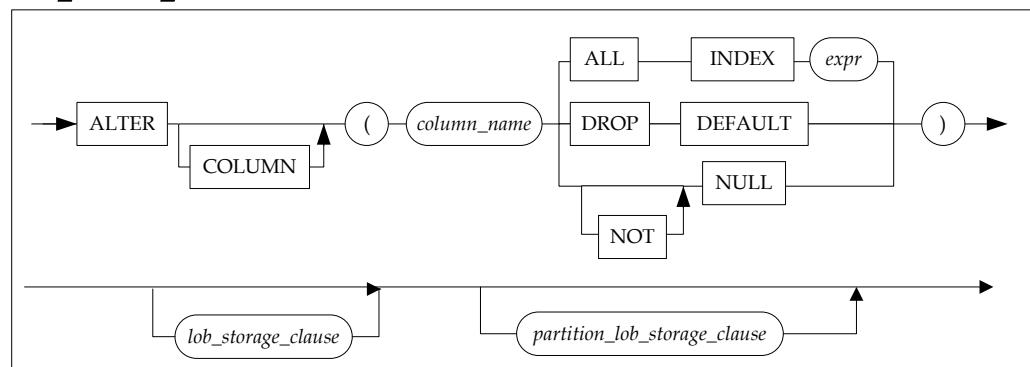
*add\_column\_clauses*::=



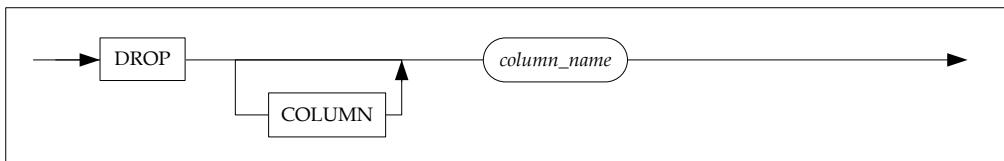
*partition\_lob\_storage\_clause* ::=



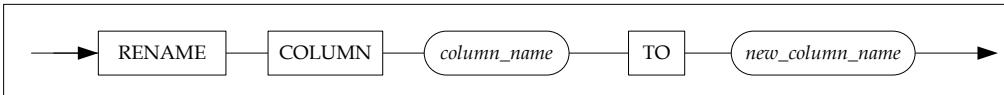
*alter\_column\_clause* ::=



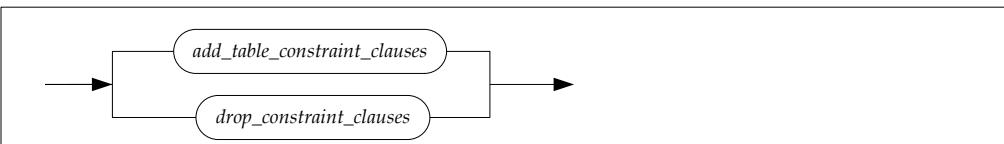
*drop\_column\_clause ::=*



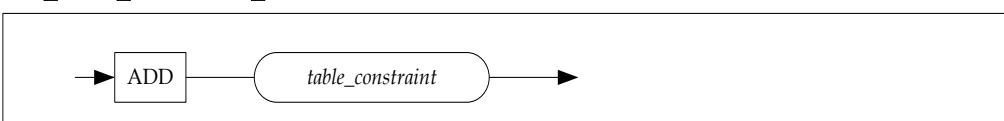
*rename\_column\_clause ::=*



*constraints\_clauses ::=*



*add\_table\_constraint\_clauses ::=*



## 전제 조건

SYS 사용자이거나 테이블이 현재 사용자의 스키마에 속하거나 테이블에 대해 ALTER 접근 권한을 가지거나 또는 ALTER ANY TABLE 시스템 권한을 가진 사용자만이 테이블 정의를 변경할 수 있다.

## 설명

명시된 테이블 정의를 변경하는 SQL 문으로 수행 후 메타 정보가 변경된다.

ALTER TABLE 구문으로 파티션드 테이블(partitioned table)의 속성을 변경할 수 있다. 관련 구문으로 추가(ADD), 병합(COALESCE), 삭제(DROP), 합병(MERGE), 이름변경(RENAMEN), 분할(SPLIT), 레코드 삭제(TRUNCATE)가 있다.

아래 표는 각 구문을 범위, 해시, 리스트 파티션에서 사용할 수 있는지 여부를 나타낸다.

	범위 파티션	리스트 파티션	해시 파티션
추가	X	X	○
병합	X	X	○
삭제	○	○	X
합병	○	○	X
이름 변경	○	○	○
분할	○	○	X
레코드 삭제	○	○	○

[표 3-1] 파티션 방법에 따른 연산

#### *user\_name*

변경될 테이블의 소유자 이름을 명시한다. 생략하면 알티베이스는 현재 세션에 연결된 사용자의 스키마에 속한 것으로 간주한다.

#### *tbl\_name*

변경될 테이블 이름을 명시한다.

#### *add\_table\_partition*

파티션드 테이블에 파티션을 추가하는 구문으로 해시 파티션드 테이블에 사용할 수 있다.

기존 테이블에 로컬 인덱스가 생성되어 있는 경우, 추가된 파티션에도 로컬 인덱스가 생성된다. 이 때 로컬 인덱스의 이름은 시스템에서 자동으로 생성되고, 테이블스페이스는 파티션의 테이블스페이스를 따른다.

#### *alter\_table\_partition\_spec*

파티션의 이름과 테이블스페이스를 명시할 수 있다.  
테이블스페이스는 생략이 가능하며 이 때에는 해당 테이블이 위치한 테이블스페이스에 같이 저장된다.

그리고 해당 테이블에 인덱스가 존재한다면 인덱스 파티션을 저장할 테이블스페이스를 지정할 수 있다.

#### *table\_partition\_description*

각 파티션에 테이블스페이스를 명시하거나 LOB 컬럼이 있는 경우 LOB 컬럼의 속성을 지정할 수 있다.

테이블스페이스 구문이 생략되면, 해당 테이블의 기본 테이블스페이스에 같이 저장된다. 마찬가지로 LOB 컬럼에 테이블스페이스 구문을 생략한 경우에는 해당 파티션의 테이블스페이스에 저장된다.

테이블스페이스 적용 방식에 관한 더 자세한 내용은 CREATE TABLE 구문의 *table\_partition\_description*을 참고한다.

#### *alter\_table\_index\_partition\_description*

SPLIT PARTITION, MERGE PARTITION, ADD PARTITION의 경우 새로운 파티션이 생성된다. 이 때, 테이블 파티션과 같이 생성되는 인덱스 파티션을 저장할 테이블스페이스를 지정할 수 있는 구문이다.

#### *coalesce\_table\_partition*

해시 파티션에만 적용할 수 있다. 해시 파티션을 병합하고 데이터를 재구성한다. 파티션을 병합하면 마지막 파티션부터 제거되고, 파티션의 이름도 마지막 파티션부터 제거된다.

#### *drop\_table\_partition*

파티션을 제거한다. 파티션에 있는 데이터와 함께 로컬 인덱스도 제거된다. 데이터를 삭제하지 않으려면, DROP 을 하기 전에 다른 파티션과 합병(MERGE)한다.

#### *merge\_table\_partition*

2 개의 파티션을 1 개의 파티션으로 합병한다. INTO 절에 합병된 새로운 파티션의 이름을 지정한다. 이름으로는 합병되는 2 개의 파티션 이름 중 하나 또는 해당 테이블에 존재하지 않는 파티션의 이름이 올 수 있다.

범위 파티션의 경우, 2 개의 파티션 중 더 큰 상한값을 갖는 파티션으로 합병된다.

리스트 파티션의 경우에는 2 개의 파티션이 갖는 파티션 키 값의 합집합을 갖는 파티션으로 합병된다.

기본 파티션과 합병할 경우에는 합병된 파티션의 도메인은 기본 파티션의 도메인으로 포함되어 기본 파티션만 남는다.

해당 테이블에 로컬 인덱스가 있는 경우에는 합병된 파티션의 로컬 인덱스가 삭제된다.

LOB 컬럼이 있는 경우에는 LOB 컬럼에 대한 속성을 따로 정의할 수 있다.

테이블스페이스을 명시하지 않을 경우의 테이블스페이스 적용 규칙은 새로 만들어진 파티션의 이름에 상관없이 테이블의 기본 테이블스페이스를 따른다.

#### *rename\_table\_partition*

파티션의 이름을 변경한다.

#### *split\_table\_partition*

하나의 파티션을 분리하여, 2 개의 파티션으로 나눈다.

AT 절은 범위 파티션에만 사용할 수 있으며 2 개의 파티션으로 나눌 기준값을 명시한다. 이 때 기준값은 바로 왼쪽 파티션의 기준값보다는 커야하고, SPLIT 하기 전의 파티션의 기준값보다는 작아야 한다.

VALUES 절은 리스트 파티션에만 사용할 수 있으며, 기존의 파티션 VALUE 리스트에서 SPLIT 하기를 원하는 값의 리스트를 명시할 수 있다. 이 때, VALUES 절에 올 수 있는 값은 기존 파티션 값의 리스트에 반드시 들어있는 값이어야 하며, 그 값의 리스트를 모두 포함할 수 없다.

INTO 절은 SPLIT 된 2 개의 파티션의 이름과 테이블스페이스 등을 지정할 수 있는 구문이다.

로컬 인덱스가 있는 경우 로컬 인덱스 파티션도 파티션과 같이 분리된다.

LOB 컬럼이 있는 경우 LOB 컬럼에 대한 속성을 따로 정의할 수 있다.

*truncate table partition*

해당 파티션안에 있는 모든 데이타를 삭제한다.

*add column clause*

테이블에 새로운 칼럼 추가한다.

*partition lob clause*

파티션드 테이블에 LOB 칼럼을 추가할 경우 해당 칼럼을 파티션별로 어떤 테이블스페이스에 저장할 것인지 정할 수 있다.

### *alter\_column\_clauses*

기존 칼럼 기본 값을 변경한다.

*drop\_column\_clause*

DROP [COLUMN]      칼럼 삭제

*rename\_column\_clause*

- RENAME COLUMN 칼럼 이름 변경
  - DROP CONSTRAINT 제약조건 삭제
  - DROP PRIMARY KEY 기본키 제약 삭제
  - DROP UNIQUE UNIQUE 제약 삭제
  - RENAME TO 테이블 이름 변경
  - MAXROWS 테이블 생성 시 지정된 최대 레코드의 개수를 변경 (CREATE TABLE 참고)

- ENABLE/DISABLE 해당 테이블(*tbl\_name*)의 모든 색인들을 비활성화 또는 활성화로 변경할 수 있다. 서버 재구동 시 또는 런타임 시 인덱스 빌딩 시간을 최소화<sup>1</sup> 하여 성능을 향상시킬 수 있다. 예를 들어, iLoader로 대량의 데이터를 데이터베이스에 적재 시 (또는 기존 테이블의 내용을 새 테이블로 이동 등) 데이터가 저장될 테이블에 인덱스가 많이 있는 경우, 인덱스에 대한 연산으로 인해 데이터 로딩에 많은 시간이 소요<sup>2</sup>되게 된다. 그러므로, 인덱스를 비활성화(disable)하고, 대량의 레코드 삽입 작업 후 나중에 인덱스를 활성화(enable)함으로써 인덱스 빌딩 시간을 단축시켜 성능을 향상시킬 수 있다.

#### *column\_definition*

- DEFAULT 새로운 칼럼 추가 시 DEFAULT 절을 명시하지 않으면 새로운 칼럼에 대한 각 행의 초기값은 NULL이다. DEFAULT를 명시한 경우, 알티베이스는 새로운 칼럼에 대한 기존 행을 칼럼 추가 시 명시한 DEFAULT 값으로 변경한다.
- TIMESTAMP TIMESTAMP 칼럼 추가

#### *column\_constraint*

새로운 칼럼에 대해 제약조건 추가

- NULL/NOT NULL 열에 null 값 허용 여부를 지정한다. null 값이 허용되지 않는 열은 기본값이 지정된 경우에만 ALTER TABLE을 사용하여 추가할 수 있다. 즉, 테이블에 새로 추가된 열은 null 값을 허용하거나 기본값이 지정되어 있어야 한다.
- SET PERSISTENT 자세한 내용은 CREATE INDEX 참고한다
- USING INDEX TABLESPACE *tablespace\_name* 제약 조건을 위해 생성되는 인덱스가 저장될 tablespace 를 지정한다.

\*. ALTER TABLE 문의 상당수의 *clause*들이 CREATE TABLE 문과 같은 기능을 가지고 있다. 그러한 절들에 대한 자세한 정보는 CREATE TABLE 을 참고 한다.

## 주의 사항

<sup>1</sup> 재 구동 시의 성능을 극대화 시키기 위해 Parallel Index Building 을 위한 구문과 서버를 정상으로 종료할 때 인덱스를 디스크에 반영하는 PERSISTENT 인덱스 구문을 사용할 수 있다.

<sup>2</sup> 데이터베이스에 대량의 데이터를 가진 테이블에 대해서 인덱스 생성시 여러 개의 인덱스가 있을 경우 인덱스의 개수에 비례한다. 그러나 인덱스 빌딩을 동시에(병렬로) 수행한다면 인덱스 빌딩 시간은 인덱스 중 수행 시간이 가장 큰 인덱스 하나의 빌딩 시간과 같아지므로 대량의 레코드 삽입 시 더 나은 성능을 얻을 수 있다.

이중화 대상 테이블일 경우 테이블의 정의를 변경할 수 없다.

파티션이 하나만 있으면 COALESCE/DROP TABLE PARTITION 을 사용할 수 없다.

해시 파티션드 테이블은 DROP PARTITION 과 MERGE PARTITION 을 사용할 수 없다. 대신 COALESCE PARTITION 구문을 이용한다. 또한 해시 파티션드 테이블은 SPLIT PARTITION 을 사용할 수 없다.

범위 파티션드 테이블은 병합할 파티션이 인접해야 한다.

다른 테이블에 의해 참조되는 기본키(PRIMARY KEY) 또는 유니크가 존재하면 변경할 수 없다.

칼럼의 추가 또는 삭제로 전체 테이블의 칼럼 수가 0 이 되거나 최대 허용 칼럼 수(1024 개)를 초과할 수 없다. 만약 VARCHAR 칼럼을 사용한 테이블이라면 최대 칼럼 허용 칼럼 수는 500 개이다.

기본키는 한 테이블에 2 개 이상 존재할 수 없다.

참조 제약의 경우 외래키(foreign key)와 참조키의 칼럼 개수와 자료형은 동일해야 한다.

한 테이블에 생성할 수 있는 최대 인덱스 개수는 64 개이며 기본키 또는 유니크의 개수의 총합이 32 개를 넘을 수 없다.

인덱스가 비활성화 된 상태에 있는 테이블에 대해  
SELECT/UPDATE/DELETE 문을 수행 시 다음과 같은 오류  
메시지를 볼 수 있다.

Index was disabled.

해결 방법은 위의 구문 중 ALTER TABLE ... ENABLE; 을 수행 후  
SELECT/UPDATE/DELETE 문을 수행한다.

---

## 제한 사항

ALTER TABLE ADD/DROP CONSTRAINT 문으로 기존 칼럼에  
TIMESTAMP constraint 를 추가/삭제할 수 없다.

TIMESTAMP 칼럼은 INSERT 또는 UPDATE 수행 시 기본값으로  
시스템 시간 값을 설정한다. 따라서 명시적으로 ALTIER TABLE  
SET/DROP DEFAULT 문을 이용하여 DEFAULT 를 설정/삭제할 수  
없다. CREATE TABLE 을 참고한다

---

## 예제

## 새로운 칼럼 추가/삭제

〈질의〉 테이블 book에 다음 칼럼들을 추가하라.

```
iSQL> ALTER TABLE book  
      ADD COLUMN (isbn CHAR(10) PRIMARY KEY,  
                  edition INTEGER DEFAULT 1);  
Alter success.
```

또는

```
iSQL> ALTER TABLE book  
      ADD COLUMN (isbn CHAR(10) CONSTRAINT const1  
                  PRIMARY KEY, edition INTEGER DEFAULT 1);  
Alter success.
```

〈질의〉 테이블 book의 isbn 칼럼을 삭제하라.

```
iSQL> ALTER TABLE book  
      DROP COLUMN isbn;  
Alter success.
```

〈질의〉 테이블 book에 TIMESTAMP 칼럼을 추가하라.

```
iSQL> ALTER TABLE book  
      ADD COLUMN (due_date TIMESTAMP);  
Alter success.
```

〈질의〉 테이블 book에 TIMESTAMP 칼럼을 삭제하라.

```
iSQL> ALTER TABLE book  
      DROP COLUMN due_date;  
Alter success.
```

## 기존 칼럼에 대한 제약 추가/삭제

〈질의〉 테이블 book의 기존 북넘버(bno)에 UNIQUE 제약을 추가하라.

```
iSQL> ALTER TABLE book  
      ADD UNIQUE(bno);  
Alter success.
```

또는

```
iSQL> ALTER TABLE book  
      ADD CONSTRAINT const1 UNIQUE(bno);  
Alter success
```

〈질의〉 테이블 book의 bno 칼럼의 UNIQUE 제약을 삭제하라.

```
iSQL> ALTER TABLE book  
      DROP UNIQUE(bno);  
Alter success.
```

〈질의〉 테이블 inventory에 다음 칼럼 추가 시 book 테이블의 isbn을 참조하는 제약 fk\_isbn을 추가하라.

```
iSQL> ALTER TABLE inventory
      ADD COLUMN(isbn CHAR(10) CONSTRAINT fk_isbn REFERENCES
book(isbn)) USING INDEX PARALLEL 4;
Alter success.
```

〈질의〉 테이블 inventory의 제약 fk\_isbn을 삭제하라.

```
iSQL> ALTER TABLE inventory
      DROP CONSTRAINT fk_isbn;
Alter success.
```

〈질의〉 테이블 book에 기본키 제약을 삭제하라.

```
iSQL> ALTER TABLE book
      DROP PRIMARY KEY;
Alter success.
```

〈질의〉 테이블 book의 기존 북넘버(bno)에 PRIMARY KEY 제약을 추가하고, 이때 인덱스는 시스템 고장이나 미디어 고장이 발생하더라도 사용할 수 있게(LOGGING) 하라.

```
iSQL> ALTER TABLE book
      ADD PRIMARY KEY (bno) USING INDEX PARALLEL 4;
Alter success.
```

또는

```
iSQL> ALTER TABLE book
      ADD PRIMARY KEY (bno) USING INDEX LOGGING
      PARALLEL 4;
Alter success.
```

〈질의〉 테이블 book의 기존 북넘버(bno)에 PRIMARY KEY 제약을 추가하고, 이때 인덱스는 NOLOGGING 옵션으로 생성하되 서버가 죽더라도 인덱스를 사용할 수 있게(FORCE) 하라.

```
iSQL> ALTER TABLE book
      ADD PRIMARY KEY (bno) USING INDEX NOLOGGING PARALLEL 4;
Alter success.
```

또는

```
iSQL> ALTER TABLE book
      ADD PRIMARY KEY (bno) USING INDEX NOLOGGING FORCE PARALLEL
      4;
Alter success.
```

〈질의〉 테이블 book의 기존 북넘버(bno)에 PRIMARY KEY 제약을 추가하고, 이때 인덱스는 NOLOGGING 옵션으로 생성하고 디스크에 반영하지 않게(NOFORCE) 하라.

```
iSQL> ALTER TABLE book
      ADD PRIMARY KEY (bno) USING INDEX NOLOGGING NOFORCE
      PARALLEL 4;
```

Alter success.

## 각 인덱스 파티션의 테이블스페이스 지정

〈질의〉 파티션드 테이블 T1 에 LOCALUNIQUE 제약을 갖는 I2  
컬럼을 추가하라.

```
iSQL> ALTER TABLE T1 ADD COLUMN  
(I2 INTEGER LOCALUNIQUE USING INDEX LOCAL  
(  
    PARTITION P1_LOCALUNIQUE ON P1 TABLESPACE TBS3,  
    PARTITION P2_LOCALUNIQUE ON P2 TABLESPACE TBS2,  
    PARTITION P3_LOCALUNIQUE ON P3 TABLESPACE TBS1  
)  
);
```

## 칼럼 이름 변경

임의의 테이블의 칼럼 이름을 변경할 때 사용한다. 새로운 칼럼  
이름은 그 테이블에 있는 다른 칼럼 이름과 같아서는 안 된다. 칼럼  
이름이 변경됐을 때, 예전 칼럼과 관련 된 인덱스 및 모든 제한  
조건은 새로운 칼럼이 승계한다.

예전 칼럼과 관련 된 PSM 이 존재하면 해당 PSM 은 invalid 한  
상태로 설정되므로, 만약 PSM 을 다시 사용할 시에는 사용자는  
새로운 칼럼 이름으로 PSM 을 변경할 필요가 있다.

〈질의〉 테이블 department 에서 칼럼 이름 dno 를 dcode 로  
변경하라.

```
iSQL> ALTER TABLE department  
      RENAME COLUMN dno TO dcode;  
Alter success.
```

## DEFAULT 설정/삭제

〈질의〉 테이블 employee 에서 sex 의 기본값을 'M'으로 설정하라.

```
iSQL> ALTER TABLE employee  
      ALTER (sex SET DEFAULT 'M');  
Alter success.
```

〈질의〉 테이블 employee 에서 sex 의 기본값 설정을 삭제하라.

```
iSQL> ALTER TABLE employee  
      ALTER (sex DROP DEFAULT);  
Alter success.
```

## 테이블 이름 변경

〈질의〉 테이블 book 의 이름을 ebook 으로 변경하라.

```
iSQL> RENAME book TO ebook;
```

```
Rename success.
```

또는

```
iSQL> ALTER TABLE book  
      RENAME TO ebook;  
Alter success.
```

## PERSISTENT 인덱스 변경

〈질의〉 테이블 book에 PRIMARY KEY 제약과 PERSISTENT 인덱스를 가진 칼럼 isbn을 추가하라.

```
iSQL> ALTER TABLE book  
      ADD COLUMN (isbn CHAR(10) CONSTRAINT const1  
      PRIMARY KEY SET PERSISTENT = ON, edition  
      INTEGER DEFAULT 1);  
Alter success.
```

〈질의〉 테이블 book에 PRIMARY KEY 제약을 가지고 있는 칼럼 isbn에 PERSISTENT 인덱스를 추가하라.

```
iSQL> ALTER TABLE book  
      ADD COLUMN (isbn CHAR(10) CONSTRAINT const1  
      PRIMARY KEY, edition INTEGER DEFAULT 1);  
Alter success.  
iSQL> ALTER TABLE book  
      DROP COLUMN isbn;  
Alter success.  
iSQL> ALTER TABLE book  
      ADD COLUMN (isbn CHAR(10) CONSTRAINT const1  
      PRIMARY KEY SET PERSISTENT = ON, edition  
      INTEGER DEFAULT 1);  
Alter success.
```

## MAXROWS 변경

〈질의〉 테이블 department에 최대 입력할 수 있는 레코드의 개수를 6개로 설정하라.

```
iSQL> ALTER TABLE department MAXROWS 6;  
Alter success.
```

## 인덱스 활성화/비활성화

〈질의〉 테이블 orders의 모든 인덱스를 비활성화하라.

```
iSQL> ALTER TABLE orders ALL INDEX DISABLE;  
Alter success.
```

## 파티션드 테이블 생성

〈질의〉 범위, 리스트, 해시 파티션드 테이블을 생성하라.

```

CREATE TABLE T1
(
    I1 INTEGER,
    I2 INTEGER
)
PARTITION BY RANGE(I1)
(
    PARTITION P1 VALUES LESS THAN (100),
    PARTITION P2 VALUES LESS THAN (200),
    PARTITION P3 VALUES DEFAULT
) TABLESPACE SYS_TBS_DISK_DATA;

CREATE TABLE T2
(
    I1 INTEGER,
    I2 INTEGER
)
PARTITION BY LIST (I1)
(
    PARTITION P1 VALUES (1,2,3,4),
    PARTITION P2 VALUES (5,6,7,8),
    PARTITION P3 VALUES DEFAULT
) TABLESPACE SYS_TBS_DISK_DATA;

CREATE TABLE T3
(
    I1 INTEGER
)
PARTITION BY HASH (I1)
(
    PARTITION P1,
    PARTITION P2
) TABLESPACE SYS_TBS_DISK_DATA;

```

## **ADD PARTITION**

〈질의〉 해시 파티션드 테이블에 새로운 파티션을 추가하라..

```
ALTER TABLE T3 ADD PARTITION P3;
```

## **COALESCE PARTITION**

〈질의〉 해시 파티션드 테이블의 파티션을 병합하라.(T3 에는 2 개의 해시 파티션만 남는다)

```
ALTER TABLE T3 COALESCE PARTITION;
```

## **DROP PARTITION**

〈질의〉 테이블 T1에서 파티션 P2를 삭제한다.

```
ALTER TABLE T1 DROP PARTITION P2;
```

## MERGE PARTITION

〈질의〉 테이블 T1에 남아있는 파티션인 P1, P3를 P\_1\_3이라는 새로운 이름을 갖는 파티션으로 병합한다.

```
ALTER TABLE T1 MERGE PARTITIONS P1, P3 INTO PARTITION P_1_3;
```

## RENAME PARTITION

〈질의〉 파티션 P1의 이름을 P1\_LIST로 변경한다.

```
ALTER TABLE T2 RENAME PARTITION P1 TO P1_LIST;
```

## SPLIT PARTITION

〈질의〉 범위 파티션인 T1에서 기본 파티션인 P3를 350을 기준으로 SPLIT 한다. 따라서 200 ~ 350의 범위를 갖는 P\_200\_350이라는 이름의 파티션 하나가 생성되고, 기존의 기본 파티션의 이름은 P\_OVER\_350으로 변경된다.

```
ALTER TABLE T1 SPLIT PARTITION P3  
AT ( 350 ) INTO ( PARTITION P_200_350, PARTITION  
P_OVER_350 );
```

〈질의〉 리스트 파티션의 경우에는 AT 대신 VALUES를 사용해서 SPLIT 한다.

```
ALTER TABLE T2  
SPLIT PARTITION P1_LIST VALUES ( 2, 4 )  
INTO  
(  
PARTITION P_2_4 TABLESPACE TBS1,  
PARTITION P_1_3 TABLESPACE TBS2  
);
```

## TRUNCATE PARTITION

〈질의〉 파티션 P5에 들어있는 모든 데이터를 삭제한다.

```
ALTER TABLE T1 TRUNCATE PARTITION P5;
```

## row\_movement\_clause 구문 이용

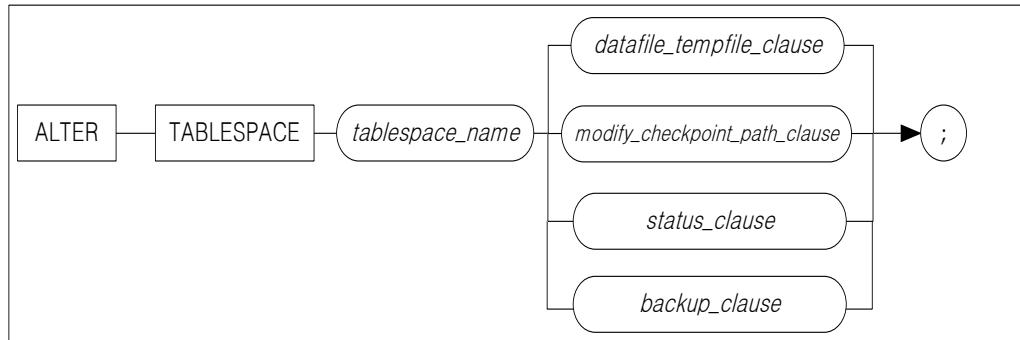
〈질의〉 테이블 T1은 반드시 파티션드 테이블이어야 한다. 논파티션드 테이블일 경우 에러가 발생한다.

```
ALTER TABLE T1 ENABLE ROW MOVEMENT;
```

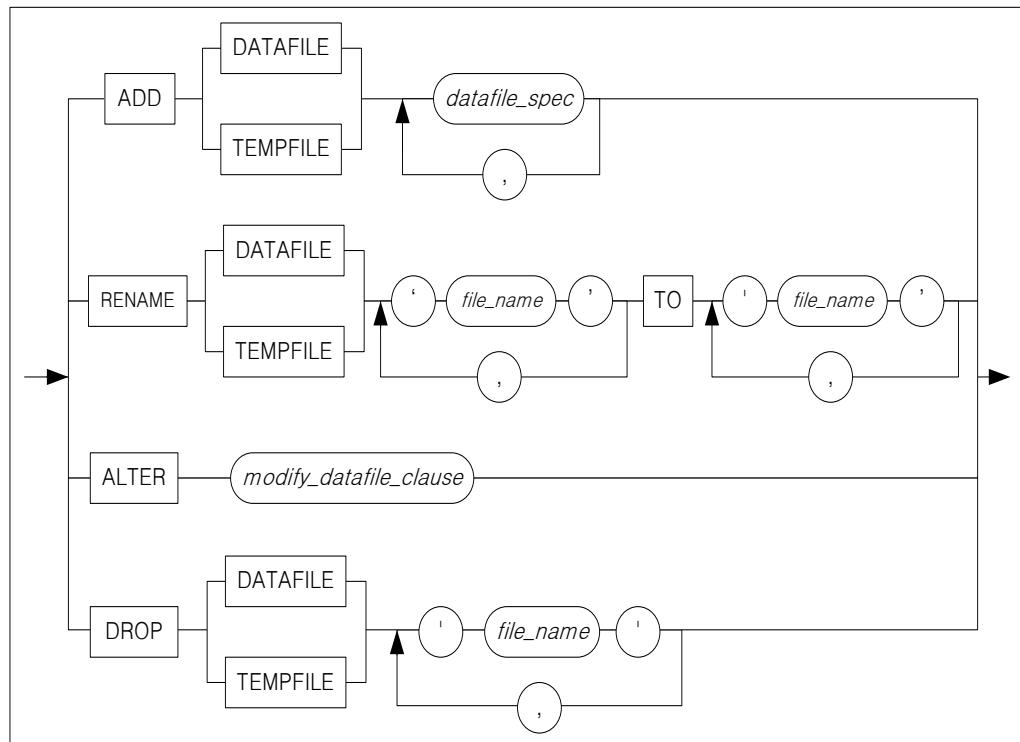
# ALTER TABLESPACE

## 구문

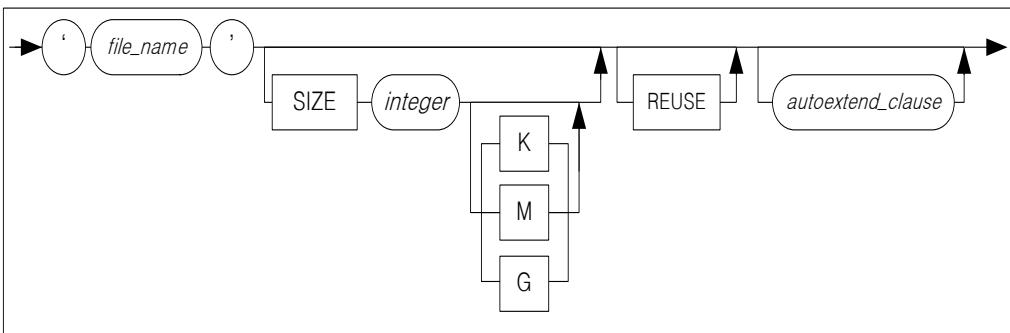
*alter\_tablespace ::=*



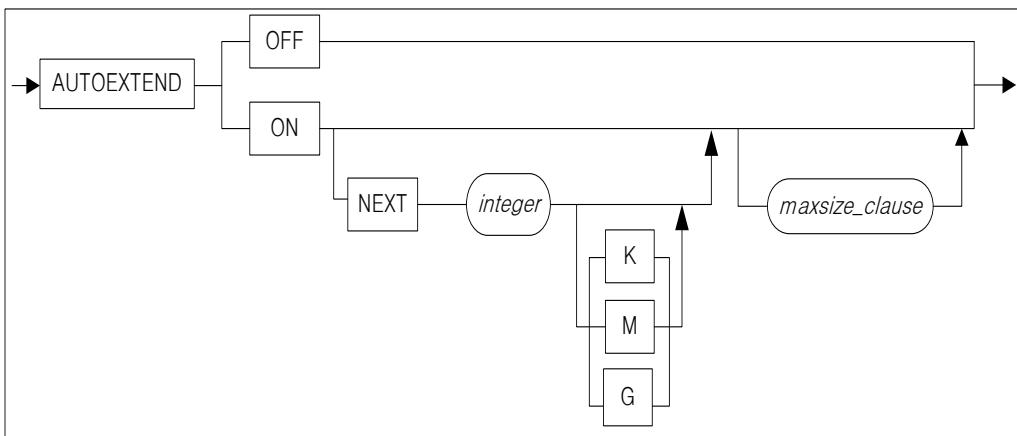
*datafile\_tempfile\_clause ::=*



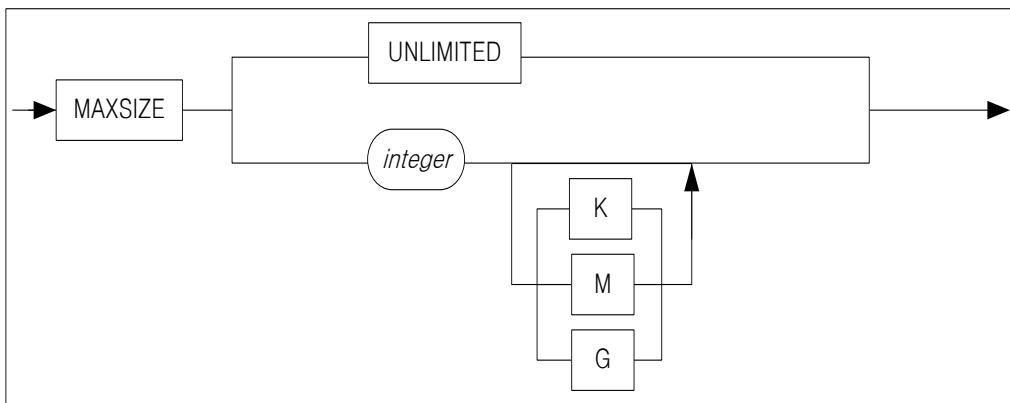
*datafile\_spec* ::=



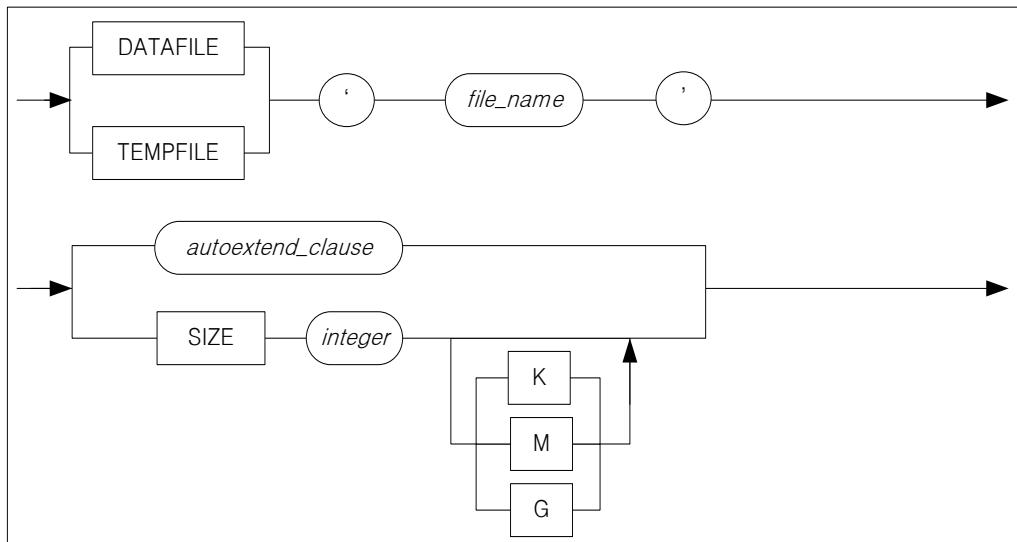
*autoextend\_clause* ::=



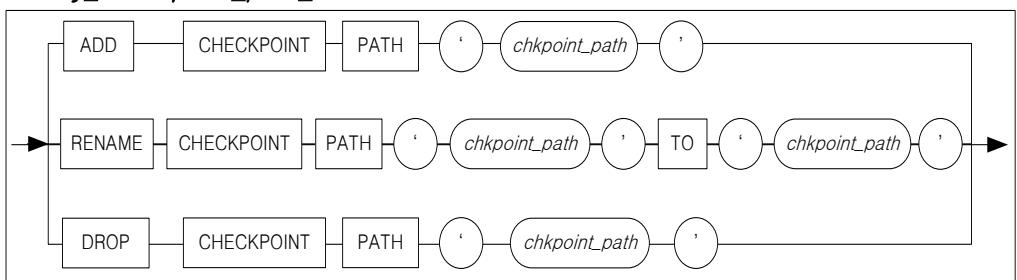
*maxsize\_clause* ::=



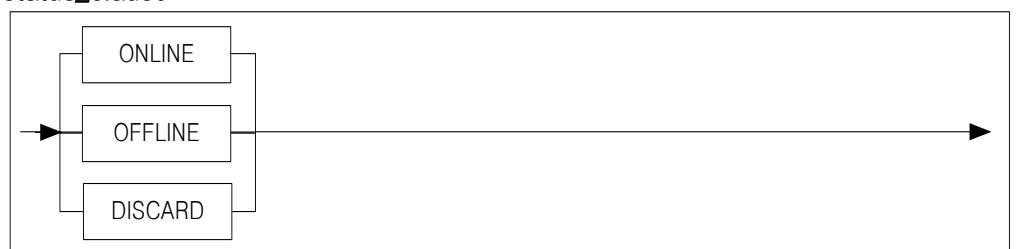
*modify\_datafile\_clause* ::=



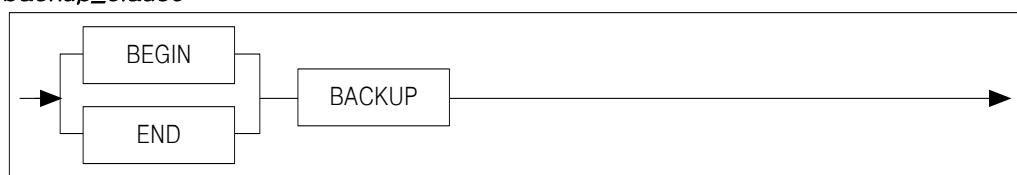
*modify\_checkpoint\_path\_clause* ::=



*status\_clause* ::=



*backup\_clause* ::=



---

## 전제 조건

SYS 사용자이거나 ALTER TABLESPACE 시스템 권한을 가진 사용자가 ALTER TABLESPACE 문의 모든 기능을 수행할 수 있다.

---

## 설명

ALTER TABLESPACE 문으로 디스크, 임시, 메모리, 휘발성 테이블스페이스의 정의를 변경한다. 또한 데이터 파일, 임시파일, 체크포인트 경로, 자동 확장 설정, 테이블스페이스 상태 등에 대해서도 변경할 수 있다.

*tablespace\_name*

변경될 테이블스페이스 이름을 명시한다.

*datafile\_tempfile\_clause*

데이터 파일 또는 임시 파일을 추가하거나 삭제, 변경하는 기능을 수행한다.

*datafile\_spec*, *maxsize\_clause*, *autoextend\_clause* 는 CREATE TABLESPACE 를 참고한다.

ADD DATAFILE | TEMPFILE 절

데이터 파일이나 임시 파일들을 해당 테이블스페이스에 추가할 수 있다.

RENAME DATAFILE | TEMPFILE 절

테이블스페이스에 속한 데이터 파일이나 임시 파일들을 새로운 이름으로 변경한다. 한번에 여러 개의 파일 이름을 변경할 수 있다. TO *file\_name* 의 새로운 파일 이름은 미리 생성되어 있어야 한다.

*modify\_datafile\_clause*

데이터 파일이나 임시 파일의 autoextend 속성, 파일 크기를 변경할 수 있다.

autoextend 의 속성을 변경하여 메모리와 휘발성 테이블스페이스의 자동확장 여부, 확장단위, 최대 크기를 변경할 수 있다.

DROP DATAFILE | TEMPFILE 절

데이터 파일이나 임시 파일들을 해당 테이블스페이스에서 제거한다. 한번에 하나 이상의 파일을 해당 테이블스페이스에서 제거할 수 있다. 이 절을 수행하더라도 운영체제상의 파일이 삭제되는 것은 아니므로 사용자가 별도의 관리를 해야 한다.

*modify\_checkpoint\_path\_clause*

체크포인트 이미지 경로의 추가 및 변경, 삭제를 한다. 체크포인트 이미지 경로 관련 연산들은 CONTROL 단계에서만 가능하다.

#### ADD CHECKPOINT PATH 절

메모리 테이블스페이스에 새로운 체크포인트 경로를 추가한다. DBA는 다른 체크포인트 경로 안에 존재하는 기존의 체크포인트 이미지들을 새로운 체크포인트 경로로 이동하는 작업을 수행해야 한다. 메모리 테이블스페이스를 로드할 때, 모든 체크포인트 경로에 대해서 체크포인트 이미지 파일을 검색하기 때문에, 체크포인트 이미지는 테이블스페이스의 체크포인트 경로 중 하나에 저장되어 있어야 한다.

새로운 체크포인트 경로를 추가한 후에 체크포인트가 발생하면, 체크포인트 이미지 파일을 새로운 체크포인트 경로까지 포함한 모든 체크포인트 경로에 골고루 분배한다.

DBA는 파일 시스템 상에서 추가할 체크포인트 경로를 생성하는 작업을 직접 수행한다. 그러나 해당 체크포인트 경로가 존재하지 않거나, 체크포인트 경로에 적절한 권한(permission)이 없으면 에러가 발생한다.

#### RENAME CHECKPOINT PATH 절

메모리 테이블스페이스의 기존 체크포인트 경로를 TO 이하 절로 변경한다. DBA는 파일 시스템 상에서 실제 체크포인트 경로의 이름을 변경하는 작업을 직접 수행한다. 그러나 해당 체크포인트 경로가 존재하지 않거나, 체크포인트 경로에 적절한 권한(permission)이 없으면 에러가 발생한다.

#### DROP CHECKPOINT PATH 절

메모리 테이블스페이스의 기존 체크포인트 경로를 삭제한다. DBA는 삭제된 체크포인트 경로 안에 존재하는 기존의 체크포인트 이미지들을 테이블스페이스에 남아있는 다른 체크포인트 경로로 이동하는 작업을 직접 수행한다. 메모리 테이블스페이스를 로드할 때, 모든 체크포인트 경로에 대해서 체크포인트 이미지 파일을 검색하기 때문에, 체크포인트 이미지는 테이블스페이스의 유효한 체크포인트 경로 중 하나에 저장되어 있어야 한다.

파일 시스템 상에서 실제 체크포인트 경로를 삭제하는 작업은 DBA가 직접 수행하여야 한다. 메모리 테이블스페이스에는 최소한 하나의 체크포인트 경로가 존재해야 한다. 만약 메모리 테이블스페이스에 남아있는 하나뿐인 유일한 체크포인트 경로를 제거하려고 하는 경우 에러가 발생한다.

#### *status\_clause*

디스크 테이블스페이스와 메모리 테이블스페이스의 상태를 ONLINE, OFFLINE, DISCARD로 전이한다.

## OFFLINE

디스크 테이블스페이스의 상태가 OFFLINE 인 경우  
테이블스페이스의 모든 데이터 페이지 내용이 데이터 파일에  
기록되며, 버퍼풀에서 무효화된다.

메모리 테이블스페이스는 데이터의 페이지 내용이 체크포인트 이미지  
파일에 기록되고, 페이지 메모리가 해제된다.

테이블스페이스의 모든 인덱스 메모리가 해제되며, 테이블에 생성된  
인덱스도 사용할 수 없다. 또한 해당 테이블스페이스에 속한  
테이블은 테이블스페이스가 ONLINE 상태로 전이할 때까지  
일시적으로 사용이 불가능한 상태가 된다.

## ONLINE

디스크 테이블스페이스의 모든 데이터 파일에 접근 가능하며, 해당  
테이블스페이스 안에 속한 테이블을 다시 사용할 수 있는 상태로  
된다.

메모리 테이블스페이스는 모든 데이터 페이지 메모리가 다시  
할당되며, 체크포인트 이미지 파일로부터 그 내용을 메모리 페이지로  
로드한다.

만약 참조 테이블스페이스가 OFFLINE 상태가 아닌 상태로  
존재한다면, 테이블스페이스의 ONLINE 연산은 실패할 수 있다.

참조 테이블스페이스란 디스크 테이블은 테이블이 소속된  
테이블스페이스와 해당 테이블과 연관된 인덱스, BLOB/CLOB 칼럼,  
파티션 테이블 등이 다른 테이블스페이스에 존재할 수 있다.

## DISCARD

STARTUP CONTROL 단계에 있는 디스크 테이블스페이스와 메모리  
테이블스페이스의 상태를 디스카드(DISCARD)로 전이한다.

디스카드 된 테이블스페이스의 테이블, 인덱스, BLOB/CLOB 칼럼에  
대해 사용이 불가능해진다. 또한 RESTART RECOVERY 와 REFINE  
DB 단계에서 DISCARD TABLESPACE 는 모두 무시한다.

한번 DISCARD 상태가 되면 DROP TABLESPACE 만 가능하며  
ONLINE 으로 될 수 없다.

## *backup clause*

디스크 또는 메모리 테이블스페이스의 데이터 파일을 복사하기 위해  
온라인 백업(핫 백업)의 시작과 완료를 명시하는 구문이다.

## BEGIN BACKUP

테이블스페이스를 구성하는 모든 데이터 파일들을 온라인 백업  
모드로 설정하기 위해 명시한다. 백업 중인 테이블스페이스는

트랜잭션 접근을 방해하지는 않는다.

사용자는 백업하기 전에 BEGIN BACKUP 을 사용해야 한다. 또한 사용자는 한 개 이상의 테이블스페이스를 동시에 온라인 백업 모드로 설정하여 온라인 백업을 진행할 수 있다. 그러나 디스크 임시 테이블스페이스는 온라인 백업 대상이 아니다.

END BACKUP

디스크 또는 메모리 테이블스페이스의 온라인 백업이 완료되었음을 명시한다. 사용자는 온라인 백업이 완료된 직후에 바로 END BACKUP 구문을 수행해야 한다.

---

## 주의사항

ALTER TABLESPACE 구문 중 데이터 파일 추가 작업과 속성 변경 작업은 온라인 모드에서 가능하고, 데이터 파일의 명칭변경은 단계별로 가능하다.

status\_clause 는 임시 테이블스페이스와 휘발성 테이블스페이스에는 사용할 수 없다.

---

## 예제

〈질의 1〉 64 MB 의 데이터 파일 tbs2.user 를 user\_disk\_tbs 테이블스페이스에 추가하고 더 큰 공간이 필요할 때 500 KB 크기의 파일이 자동으로 증가하게 하라.

```
iSQL> ALTER TABLESPACE user_disk_tbs  
      ADD DATAFILE '/tmp/tbs2.user' SIZE 64M  
      AUTOEXTEND ON NEXT 500K;  
Alter success.
```

〈질의 2〉 디스크 입출력 분산을 위해 '/home/path'를 user\_memory\_tbs 테이블스페이스에 추가하고, 확장 단위를 256M, 최대 크기를 1G 로 변경하라. (패스 추가 작업은 control 단계에서만 가능하고, 속성 변경은 서비스 단계에서 할 수 있다.)

```
iSQL(sysdba)>>>startup control;  
iSQL(sysdba)>>>ALTER TABLESPACE user_memory_tbs ADD CHECKPOINT  
      PATH '/home/path';  
Alter success.
```

```
iSQL> ALTER TABLESPACE user_memory_tbs ALTER AUTOEXTEND OFF;  
iSQL> ALTER TABLESPACE user_memory_tbs ALTER AUTOEXTEND ON NEXT  
      256M MAXSIZE 1G;  
Alter success.
```

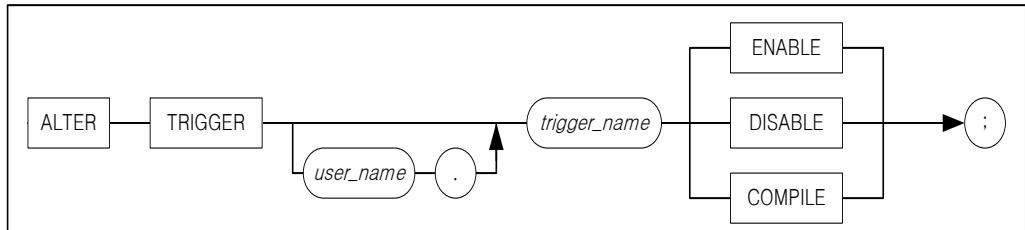
〈질의 3〉 user\_volatile\_tbs 테이블스페이스의 확장 단위를 256M,  
최대 크기를 1G로 변경하라.

```
iSQL> ALTER TABLESPACE user_volatile_tbs ALTER AUTOEXTEND ON NEXT  
256M MAXSIZE 1G;  
Alter success.
```

# ALTER TRIGGER

## 구문

*alter\_trigger ::=*



## 전제 조건

SYS 사용자이거나 트리거가 현재 사용자의 스키마에 속하거나 또는 ALTER ANY TRIGGER 시스템 권한을 가진 사용자만이 트리거를 변경할 수 있다.

## 설명

명시한 트리거의 작동을 가능, 불가능하게 하거나 컴파일 한다.

*user\_name*

변경될 트리거의 소유자 이름을 명시한다. 생략하면 알티베이스는 현재 사용자가 소유한 테이블에 트리거를 변경한다.

*trigger\_name*

변경될 트리거의 이름을 명시한다.

ENABLE

명시한 트리거의 작동을 사용 가능하게 한다.

DISABLE

명시한 트리거의 작동을 불가능하게 한다.

COMPILE

명시한 트리거의 유효성 여부에 관계 없이 명시적으로 컴파일 한다. 명시적 재컴파일은 수행 중에 트리거가 유효하지 않은 경우

암시적으로 컴파일 하는 부하를 제거할 수 있다.

---

## 예제

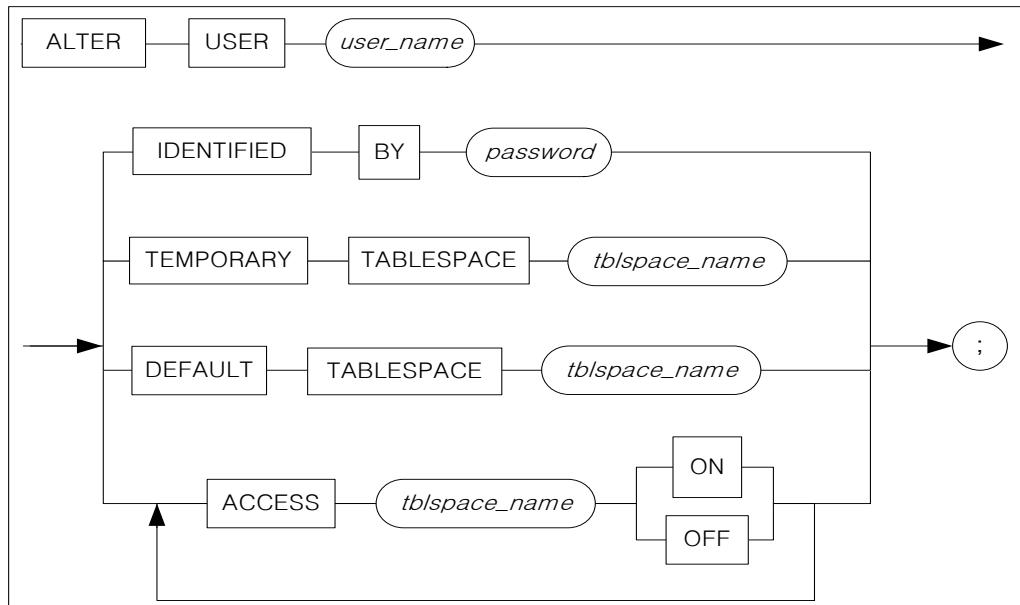
〈질의〉 다음 예는 명시한 트리거의 작동을 불가능하게 한다. 트리거 del\_trigger 는 CREATE TRIGGER 문의 예제를 참조한다.

```
iSQL> ALTER TRIGGER del_trigger DISABLE;  
Alter success.
```

# ALTER USER

## 구문

*alter\_user ::=*



## 전체 조건

SYS 사용자 이거나 ALTER USER 시스템 권한을 가진 사용자만이 사용자 정의를 변경할 수 있다. 그러나 현재 사용자의 암호 변경 시에는 권한 없이 암호를 변경할 수 있다.

## 설명

사용자의 암호를 변경하는 실행문이다.

IDENTIFIED 절

사용자의 새로운 암호를 명시한다.

이외 다른 기능들은 CREATE USER 문과 동일하므로 CREATE

USER 문을 참고한다.

---

## 주의사항

SYSDBA 모드로 접속가능한 SYS 사용자의 암호를 변경할 경우,  
ALTER USER 문으로 암호를 변경한 후 운영체제의 콘솔(유닉스 셸  
혹은 Dos 창)에서 altipasswd 를 실행하여 암호를 한번 더 변경해야  
한다. altipasswd 에 대한 자세한 내용은 *Utilities User's Manual* 을  
참고한다.

---

## 예제

〈질의〉 사용자 uare 의 암호를 abrose 로 변경하라.

```
iSQL> ALTER USER uare  
        IDENTIFIED BY abrose;  
Alter success.
```

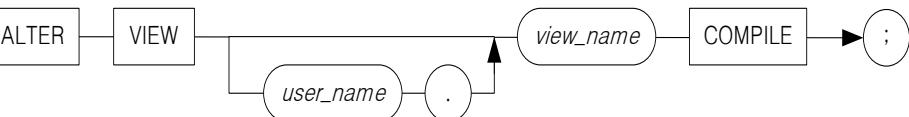
〈질의〉 사용자의 DEFAULT TABLESPACE 를 변경하라.

```
iSQL> ALTER USER uare  
        DEFAULT TABLESPACE uare_data;  
Alter success.
```

## ALTER VIEW

### 구문

*alter\_view ::=*



### 전제 조건

SYS 사용자이거나 뷰가 현재 사용자의 스키마에 속하거나 또는 ALTER ANY TABLE 시스템 권한을 가진 사용자만이 뷰를 변경할 수 있다.

### 설명

뷰가 유효하지 않을 때(invalid) 그 뷰를 재 컴파일 하는데 사용한다. 예를 들어, 뷰의 기본 테이블중에 하나가 ALTER TABLE 문에 의하여 변경된 경우 명시적으로 뷰를 재 컴파일 하는데 사용한다.

*user\_name*

재 컴파일 될 뷰의 소유자 이름을 명시한다. 생략하면 알티베이스는 현재 세션에 연결된 사용자의 스키마에 속한 것으로 간주한다.

*view\_name*

재 컴파일 될 뷰의 이름을 명시한다.

뷰를 재 컴파일 때 알티베이스는 뷰 생성문을 읽어와 다시 컴파일을 수행하므로 뷰 생성 시 발생할 수 있는 오류들이 ALTER VIEW 문 수행 시 발생할 수 있다.

CREATE VIEW 문에 FORCE 옵션을 포함한 경우 ALTER VIEW 문

수행 성공 후에도 뷰가 여전히 무효한 상태일 수 있다.

ALTER VIEW 문은 기존 뷰의 정의를 변경하지 않는다. 따라서, 뷰를 재 정의하려면 CREATE OR REPLACE VIEW 문을 사용해야만 한다.

---

## 예제

〈질의〉 다음은 기반 테이블 employee 의 정의를 변경 후 뷰 avg\_sal (CREATE VIEW 예제 참고)을 재 컴파일하라.

```
iSQL> ALTER TABLE employee
      ADD COLUMN (email VARCHAR(20));
Alter success.
iSQL> SELECT * FROM avg_sal;
[ERR-311BE: This view is invalid.
0001: SELECT * FROM AVG_SAL
          ^
          ^
]
iSQL> ALTER VIEW avg_sal COMPILE;
Alter success.
iSQL> SELECT * FROM avg_sal;
DNO  EMP_AVG_SAL
```

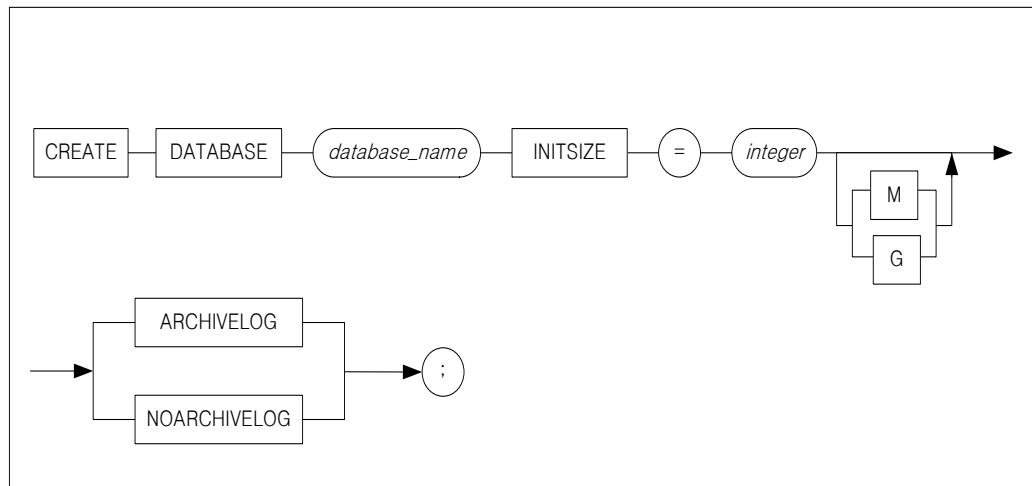
---

```
D001 2075750
C002 1660000
A001 2066666.67
C001 1576666.67
F001 1845000
6 rows selected.
```

# CREATE DATABASE

## 구문

*create\_database ::=*



## 전제 조건

데이터베이스 다단계 구동에서 PROCESS 단계에서만 수행할 수 있는 SQL 문으로 SYS 사용자가 -sysdba 관리자 모드에서만 수행할 수 있다.

## 설명

데이터베이스를 생성하는 구문이다. 기존 createdb 유ти리티를 대체하는 SQL 문이다. 데이터베이스 생성 시 딕셔너리 테이블스페이스(Dictionary Tablespace) 및 언두 테이블스페이스(Undo Tablespace), 임시 테이블스페이스(Temp Tablespace)등 다수의 시스템 테이블스페이스(System Tablespace)가 만들어진다. 생성되는 시스템 테이블스페이스는 시스템에 정의된 이름과 알터베이스 프로퍼티의 기본값을 갖는다. 그러나 데이터베이스 생성시에는 사용자 정의 테이블스페이스를 만들수 없고, 이후에 사용자가 추가할 수 있다.

### *database\_name*

생성할 데이터베이스 이름을 명시한다. 데이터베이스 이름을 명시할 경우 프로퍼티 파일에 DB\_NAME에 명시된 이름과 동일해야 하며 다른 경우에는 오류가 발생한다.

### INITSIZE 절

초기화 될 MEMORY DATABASE의 크기를 나타내며, 128M, 4G 등의 형식으로 사용할 수 있다. 데이터베이스 공간의 크기 단위는 기본값으로 MB(Mega Bytes)이다.

DISK DATABASE와 관련된 SYSTEM TABLESPACE도 CREATE DATABASE 문 수행 시에 생성된다. SYSTEM TABLESPACE에 대한 기본값들은 프로퍼티 파일에서 아래의 프로퍼티를 읽어 생성한다.

- SYS\_DATA\_TBS\_EXTENT\_SIZE
- SYS\_DATA\_TBS\_INIT\_SIZE
- SYS\_DATA\_TBS\_MAX\_SIZE
- SYS\_DATA\_TBS\_NEXT\_SIZE

### ARCHIVELOG | NOARCHIVELOG

데이터베이스 생성 시점에 archive log 또는 noarchive log 방식으로 운영할지를 명시한다.

ARCHIVELOG는 미디어 리커버리의 가능성에 대비하기 위한 절인 반면에 NOARCHIVE 절은 미디어 리커버리를 허용하지 않는다.

---

## 예제

〈질의〉 데이터베이스 이름이 mydb이고 크기가 10MB인 데이터베이스 파일을 생성하라.

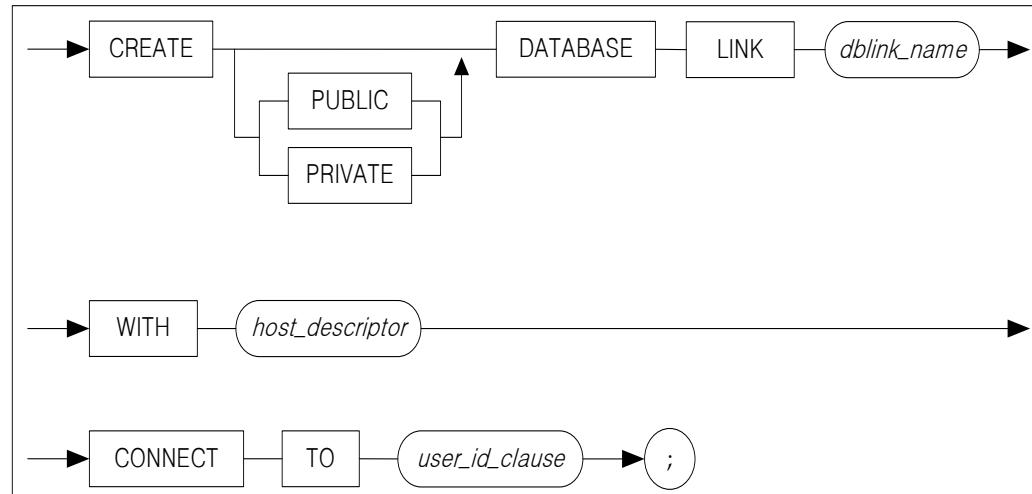
```
shell> is -sysdba
..
iSQL> STARTUP PROCESS;
Trying Connect to Altibase.. Connected with Altibase.

TRANSITION TO PHASE: PROCESS
Command execute success.
iSQL> CREATE DATABASE mydb INITSIZE=10M NOARCHIVELOG;
.
.
Create success.
```

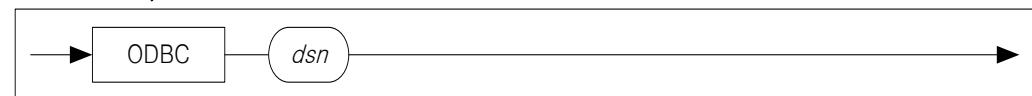
## CREATE DATABASE LINK

### 구문

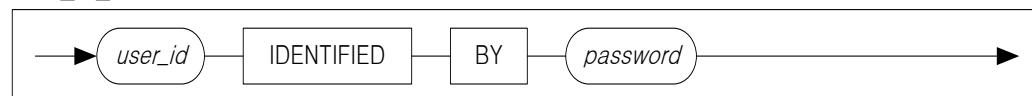
*create\_database\_link ::=*



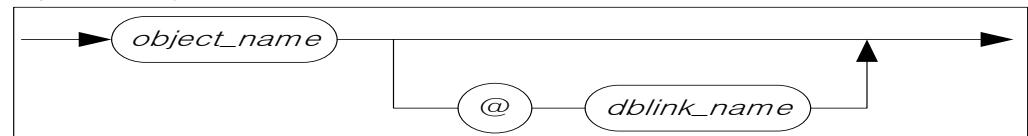
*host\_descriptor ::=*



*user\_id\_clause ::=*



*object\_descriptor ::=*



---

## 전제 조건

SYS 사용자이거나 CREATE DATABASE LINK 시스템 권한을 가진 사용자만이 데이터베이스 링크를 생성할 수 있다.

데이터베이스 링크는 ODBC를 사용하여 원격 서버와 접속하게 되므로, 서버에 ODBC 드라이버가 설치되어 있어야 하며, 원격 서버에 대한 DSN이 ODBC 환경 화일에 설정되어 있어야 한다. ODBC 설정에 대한 자세한 내용은 *Starting User's Manual* 을 참고한다.

---

## 설명

주어진 데이터베이스 링크 이름으로 새로운 데이터베이스 링크 객체를 생성한다.

데이터베이스 링크의 생성 구문으로 데이터베이스 링크를 생성할 경우 생성 후 바로 서비스를 시작할 수 있어, 별도의 서비스 시작|종료를 위한 명령을 필요로 하지 않는다.

PUBLIC|PRIVATE

생성할 데이터베이스 링크의 PUBLIC|PRIVATE 속성을 지정한다. PUBLIC으로 지정하면 생성된 데이터베이스 링크를 모든 사용자가 사용할 수 있으며, PRIVATE으로 지정하면 데이터베이스 링크를 생성한 사용자만 사용할 수 있다. 지정하지 않을 경우, PRIVATE으로 생성된다.

*dblink\_name*

생성할 데이터베이스 링크 이름을 명시한다.

*dsn*

ODBC를 통하여 접근할 때 ODBC의 DSN(Data Source Name) 이름을 지정한다.

*user\_id\_clause(user\_id/password)*

연결하고자 하는 원격 서버의 데이터베이스 사용자를 지정하는 것으로, 사용자 식별자 및 비밀번호를 지정한다. 여기에 지정된 사용자는 데이터베이스 링크로 접근할 때, 해당 객체에 대한 접근 권한이 부여되어 있어야 한다. 그렇지 않을 경우, 권한 관련 오류가 발생한다.

---

## 위치 표시자

DML 질의문에서 데이터베이스 링크를 사용하기 위해서 위치 표시자(Location descriptor)를 사용한다. 위치 표시자는 '@' 기호와 링크명으로 구성된다. 데이터베이스 링크의 위치 표시자는 FROM 구문에서만 사용가능하다.

위치 표시자가 결합된 객체 표시자(Object descriptor)는 object\_descriptor ::= 의 구문으로 사용할 수 있다. 데이터베이스 링크를 통해 원격 서버의 객체에 접속하도록 객체명과 데이터베이스 링크명을 지정한다.

### 전제 조건

사용하고자 하는 데이터베이스 링크가 PUBLIC 데이터베이스 링크이거나, 자신이 생성한 PRIVATE 데이터베이스 링크이어야 한다.

#### *object\_name*

원격 서버에 존재하는 객체의 이름을 명시한다. 이 객체는 원격 서버의 테이블 또는 뷰가 될 수 있다.

#### *dblink\_name*

사용할 데이터베이스 링크의 이름을 명시한다.

### 주의 사항

데이터베이스 링크를 사용할 때는 링크에 대한 위치표시자가 FROM 절에 명시되어야 한다. WHERE 절에 원격 서버의 객체에 대한 조건을 기술하고자 할 때는, WHERE 절에 직접 위치 표시자를 쓰지 않고 FROM 절의 해당 객체에 별명(alias)을 준 다음, WHERE 절 조건에서 별명을 사용해야 한다.

```
예제 1) SELECT * FROM emp@link1;
예제 2) SELECT * FROM dept d, emp@link1 AS e
WHERE d.dept_no = e.dept_no;
```

---

## REMOTE\_HINT

데이터베이스 링크를 사용할 때, 원격 서버에서 검색 대상이 되는 테이블들에 대한 조인(Join)을 처리하고 그 결과만을 가져오는 것이 효율적인 경우가 발생할 수 있다. 이렇게 원격 서버에서 자체적으로 처리하고자 하는 질의가 있을 때, EXEC\_REMOTE hint를 사용하여 원격 서버로 질의를 전달하여 원격서버에서 처리할 수 있다.

### 기능

원격 서버로 질의를 직접 전송하여 질의를 수행하게 한다.

## 사용방법

EXEC\_REMOTE 힌트는 주어진 질의의 대상이 모두 동일한 원격 서버의 객체들일 때 사용할 수 있다. 다음과 같은 경우에는 EXEC\_REMOTE 힌트를 적용할 수 없다.

- 지역(local) 서버의 객체가 포함되어 있다.
- 객체들의 데이터베이스 링크명이 다르다.
- 저장 프로시저나 시퀀스가 포함되어 있다.
- 주언어 변수를 사용한다.
- SUBQUERY에서 외부 컬럼을 참조한다.

위의 조건에 하나라도 해당되면 EXEC\_REMOTE 힌트는 무시되고, 해당되는 조건이 전혀 없을 때 힌트가 적용된다.

힌트를 사용하게 되는 구문의 객체 가운데 하나라도 다른 링크를 사용하거나 지역(Local) 서버의 객체가 포함되어 있다면, 이 힌트는 무시된다.

또한, Subquery 를 포함하고 상위 질의와 하위 질의가 모두 EXEC\_REMOTE 힌트를 사용한 경우에는, 상위 질의에 있는 힌트가 적용되고, 하위에 있는 EXEC\_REMOTE 힌트는 무시된다. 다시 말해, 상위 질의에 EXEC\_REMOTE 힌트가 없을 때, 하위 질의의 힌트가 적용된다.

```
예제 1) SELECT /*+EXEC_REMOTE*/ DISTINCT name  
FROM employee@link1;
```

```
예제 2) SELECT /*+EXEC_REMOTE*/ e.dno,  
COUNT(e.dno) CNT, AVG(salary)  
FROM employee@link1 e, department@link1 d  
WHERE e.dno = d.dno  
GROUP BY e.dno  
HAVING COUNT(e.dno) >= 3;
```

```
예제 3) SELECT /*+EXEC_REMOTE*/ e.ename, e.birth,  
(SELECT /*+EXEC_REMOTE*/  
SUM(salary)  
FROM employee@link1 sum  
FROM employee@link1 e,  
(SELECT /*+EXEC_REMOTE*/ dname, dno  
FROM department@link1 d  
WHERE e.dno = d.dno  
AND e.emp_job = 'SALESMAN'  
AND 10 >  
(SELECT /*+EXEC_REMOTE*/ count(*)  
FROM department@link1)  
ORDER BY e.eno DESC  
LIMIT 3;
```

---

## 예제

〈질의 1〉 DSN 이 altibase\_odbc 인 원격 데이터베이스 서버에  
데이터베이스 링크를 위한 계정으로 user1/user1 이 존재한다. 이 때,  
생성자 자신만이 사용할 수 있는 link1이라는 이름의 데이터베이스  
링크를 생성한다.

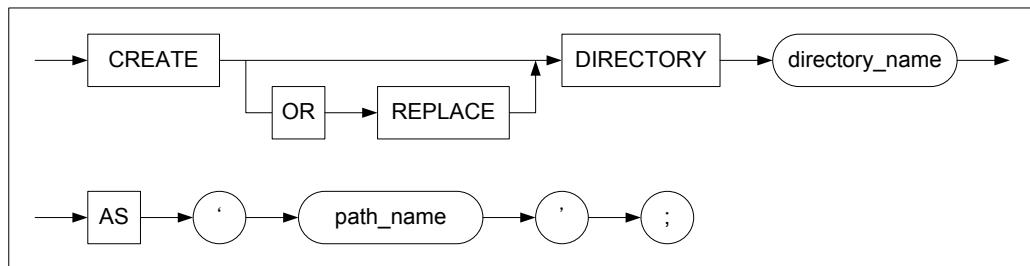
```
iSQL> CREATE PRIVATE DATABASE LINK link1  
      WITH ODBC altibase_odbc  
      CONNECT TO user1 IDENTIFIED BY user1;
```

〈질의 2〉 DSN 이 altibase\_odbc 인 원격 데이터베이스 서버에  
대하여 데이터베이스 링크를 위한 계정으로 user1/user1 이 존재한다.  
이 때, 시스템의 모든 사용자가 사용할 수 있는 link2라는 이름을  
가진 데이터베이스 링크를 생성한다.

```
iSQL> CREATE PUBLIC DATABASE LINK link2  
      WITH ODBC altibase_odbc  
      CONNECT TO user1 IDENTIFIED BY user1;
```

## CREATE DIRECTORY

### 구문



### 전제 조건

SYS 사용자 또는 CREATE ANY DIRECTORY 시스템 권한을 가진 사용자여야 한다.

### 설명

저장프로시저의 파일 제어 기능은 운영 체제의 텍스트 파일에 대한 읽기 및 쓰기 기능을 제공한다. 이 기능을 이용하여 사용자는 저장프로시저 실행에 대한 별도의 메시지 등을 파일에 남길 수도 있으며, 파일로 결과를 리포팅하거나 파일로부터 데이터를 읽어와 테이블에 삽입하는 등 다양한 작업을 수행할 수 있다.

이러한 저장프로시저 파일 제어 기능에서 사용하는 파일들을 저장할 디렉토리들은 CREATE DIRECTORY 문을 사용하여 데이터베이스 객체로 생성한다.

CREATE DIRECTORY 문으로 생성된 DIRECTORY 오브젝트의 소유자는 SYS이며 실제 생성한 사용자는 이 오브젝트에 대한 읽기/쓰기 권한(WITH GRANT OPTION 포함)을 부여받는다.

CREATE DIRECTORY 문은 SYS\_DIRECTORIES\_ 메타 테이블에 디렉토리 정보를 기록하며, 실제 운영 체제의 파일 시스템에 디렉토리를 생성하지는 않는다. 따라서 사용자는 실제 파일 시스템에 명시적으로 디렉토리를 생성해야 한다.

OR REPLACE

이미 존재하는 디렉토리를 대체하여 같은 이름의 새로운 디렉토리를 생성하기 위한 옵션이다. 실제 파일 시스템의 기존 디렉토리는 삭제되지 않는다.

*directory\_name*

데이터베이스 객체로 디렉토리 이름을 명시한다.

*path\_name*

운영 체제 파일 시스템 상의 절대 경로를 문자열 데이터 타입으로 명시한다.

예) '/home/altibase/altibase\_home/msg\_psm'

---

## 예제

〈질의〉 /home/altibase/altibase\_home/msg\_psm 디렉토리를 가리키는 alti\_dir1 디렉토리를 생성하라.

```
iSQL> create directory alti_dir1 as '/home/altibase/altibase_home/msg_psm';
Create success.
```

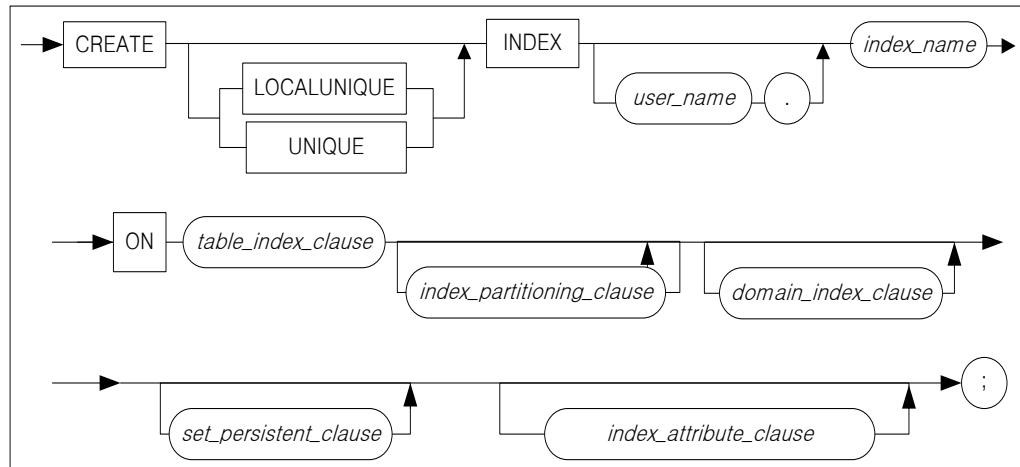
〈질의〉 /home/altibase/altibase\_home/msg\_result 디렉토리를 가리키는 alti\_dir1 디렉토리를 생성하라. 단 이미 alti\_dir1 이름의 디렉토리가 데이터베이스에 존재하는 경우 이를 대체하여 생성하라.

```
iSQL> create or replace directory alti_dir1 as
'/home/altibase/altibase_home/msg_result';
Create success.
```

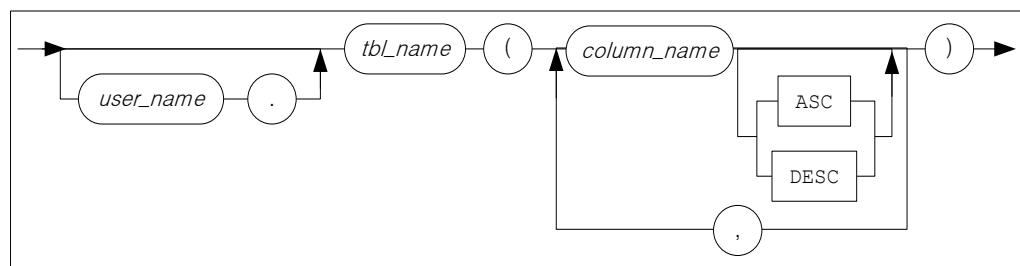
# CREATE INDEX

## 구문

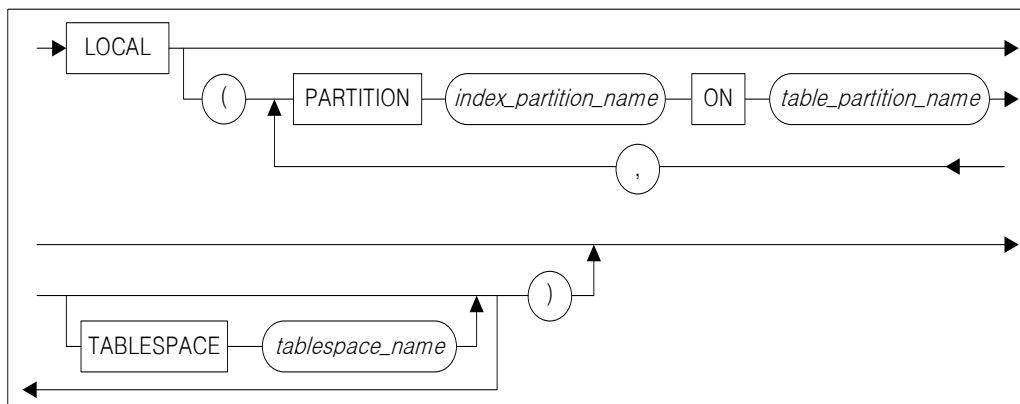
*create\_index ::=*



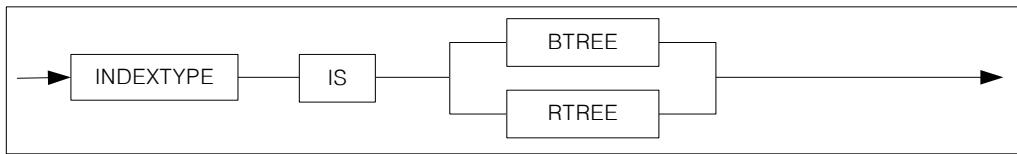
*table\_index\_clause ::=*



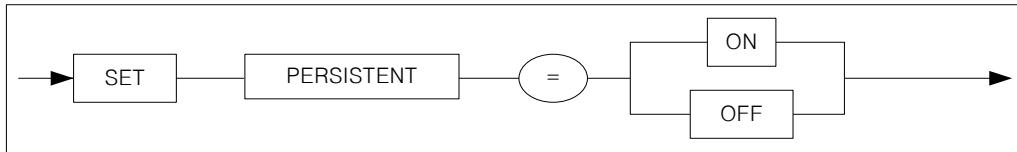
*index\_partitioning\_clause ::=*



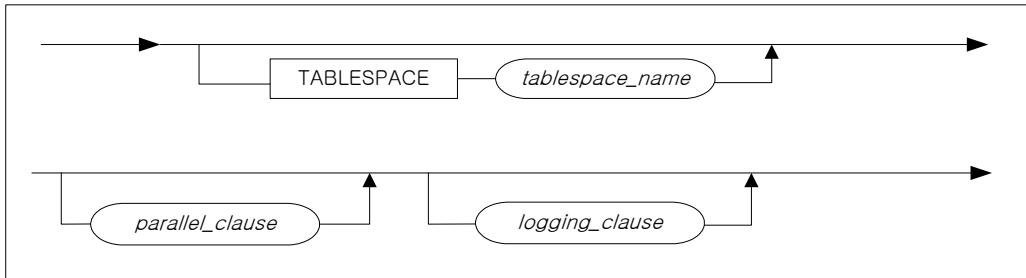
*domain\_index\_clause ::=*



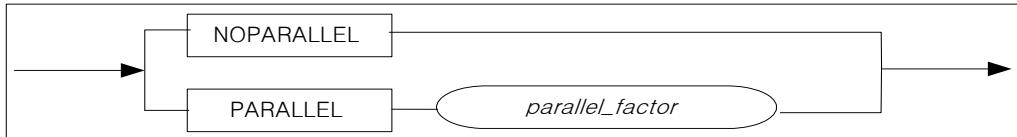
*set\_persistent\_clause ::=*



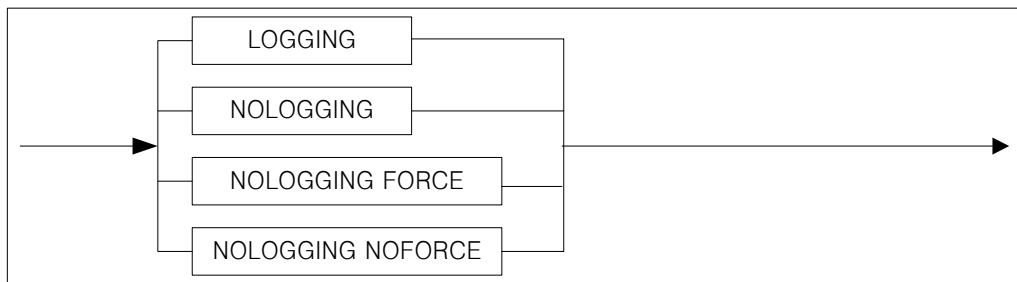
*index\_attribute\_clause ::=*



*parallel\_clause ::=*



*logging\_clause ::=*



## 전제 조건

SYS 사용자, CREATE INDEX 권한을 가진자 또는 색인될 테이블에 대한 인덱스 객체 접근 권한을 가진 사용자여야 한다.

## 설명

테이블의 명시된 칼럼에 대해 인덱스를 생성한다. 인덱스를 생성할 때 로컬 인덱스(local index)를 생성할 수 있으며, 로컬 유니크 속성을 부여할 수 있다.

파티션드 인덱스는 파티션 키와 인덱스 칼럼의 관계에 따라 프리픽스드 인덱스(prefixed-index)와 논프리픽스드 인덱스(non-prefixed index)로 구분된다. 프리픽스드 인덱스는 인덱스 파티션 키와 인덱스 칼럼의 왼쪽 컬럼이 같은 경우이며, 같지 않은 경우 논프리픽스드 인덱스이다.

*user\_name*

생성될 인덱스의 소유자 이름을 명시한다. 생략하면 알터베이스는 현재 세션에 연결된 사용자의 스키마에 인덱스를 생성한다.

*index\_name*

생성될 인덱스 이름을 명시한다.

UNIQUE

중복 값을 허용하지 않는다.

ASC/DESC

각 칼럼마다 정렬 방법을 결정할 수 있다.

*index\_partitioning\_clause*

파티션드 테이블의 각 파티션에 로컬 인덱스를 생성한다. 로컬 파티션 인덱스는 LOCAL 구문까지만 쓰는 경우와 각 테이블 파티션에 대해 생성할 인덱스 파티션을 직접 명시하는 경우로 구분된다.

LOCAL 구문까지만 쓰는 경우, 모든 테이블 파티션에 인덱스 파티션이 자동으로 생성되며 각 인덱스 파티션의 이름은 시스템 내부에서 자동으로 만들어진다. 인덱스 파티션 이름은 SYS\_IDX1, SYS\_IDX2, ... 와 같은 형태로 순차적으로 붙여진다.

테이블 파티션에 인덱스 파티션을 직접 명시하는 경우 부분 또는 전체에 명시할 수 있다. 부분적으로 명시한 경우 나머지 테이블 파티션에 대한 인덱스 파티션은 자동적으로 생성되며, 생성 방식은 상기와 동일하다.

BTREE

B+-Tree 인덱스이며, 범위 검색 시 유용하다. INDEXTYPE IS 절 생략 시 기본값이다.

RTREE

R-Tree 인덱스이며, 다차원의 데이터 처리 등을 할 때 유용하다.

#### SET PERSISTENT

서버가 정상으로 종료 시 메모리 인덱스를 디스크에 저장할 것인지 여부를 지정하는 옵션이다.

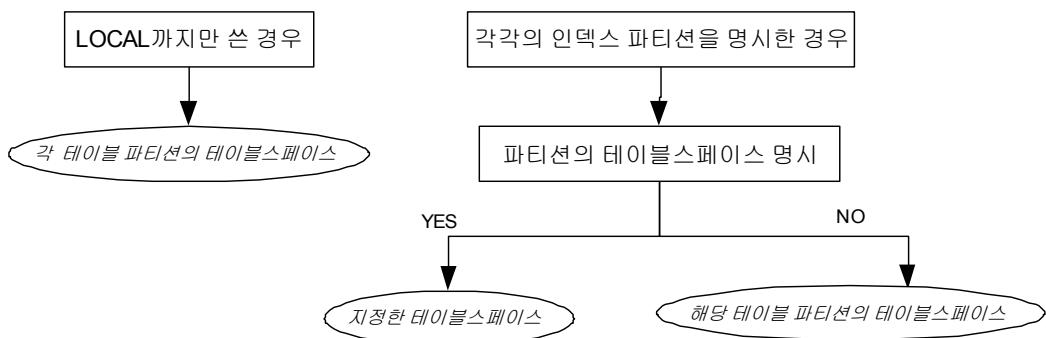
서버는 종료 시 메모리 인덱스를 저장하지 않고, 서버 구동 시 메모리 테이블스페이스의 모든 인덱스들을 재생성한다. 그러나 PERSISTENT 옵션을 ON으로 지정하면, 서버를 정상 종료할 때 메모리 인덱스를 특정 파일로 디스크에 저장한다. 그리고 서버를 구동할 때 저장된 인덱스 파일들을 읽음으로써 인덱스 재생성 시간을 절약할 수 있다.

기본값은 OFF이며, 서버를 종료할 때 인덱스를 디스크에 저장하기 위해서는 PERSTISTENT 옵션을 ON으로 지정한다.

#### *index\_attribute\_clause*

테이블스페이스를 지정한다. 이 구문을 생략한 경우 테이블의 기본 테이블스페이스에 인덱스 파티션이 생성된다.

테이블스페이스를 명시적으로 지정하지 않을 경우, 로컬 인덱스의 테이블스페이스 결정 방법은 아래 그림과 같다.



#### TABLESPACE 절

인덱스가 사용할 테이블스페이스 이름을 명시한다. 이 절을 생략하면 알터베이스는 인덱스를 포함한 스키마 소유자의 기본 테이블스페이스에 인덱스를 생성한다.

#### *parallel\_clause*

빠른 인덱스 생성을 위해서 parallel\_factor 를 줄 수 있다. 값은 생략 가능하며, 생략할 경우 altibase.properties에서 정의한 INDEX\_BUILD\_THREAD\_COUNT 값으로 인식된다.

parallel\_factor의 최대값은 CPU 개수 \* 2이며, 최대값을 초과해 설정된 경우도 시스템의 CPU 개수 \* 2 만큼 thread 가 생성된다. 0 또는 음의 값을 설정한 경우엔 오류 메시지가 나타난다.

#### *logging\_clause*

디스크 테이블은 로깅(LOGGING)과 노로깅(NOLOGGING) 구문을 사용해 제약조건을 위한 인덱스를 생성할 때 발생하는 로그를 기록하거나 생략할 수 있다. 기본 값은 로그를 기록하며, 인덱스를 생성할 때 로깅을 한다.

생성된 디스크 인덱스를 강제로 디스크에 저장할 것인지 여부를 선택하기 위해 FORCE 와 NOFORCE 옵션을 사용한다.

*logging\_clause*에 대한 자세한 내용은 [Administrator's Manual](#)의 데이터베이스 객체 및 권한 관리 중에서 인덱스 관리를 참고한다.

---

## 주의 사항

로컬 인덱스 생성시에는 인덱스의 테이블스페이스는 지정할 수 없으며, INDEXTYPE 으로 BTREE 만 지정할 수 있다.

NOLOGGING(FORCE/NOFORCE)으로 생성된 인덱스는 시스템이나 미디어 고장으로 인덱스의 일관성을 보장하기 어려울 수 있다. 이 경우 'The index is inconsistent.'라는 오류 메시지가 나온다. 이러한 오류를 해결하기 위해 일관성이 깨어진 인덱스를 찾아 DROP 한 후에 해당 인덱스를 다시 생성한다. 인덱스의 일관성은 V\$DISK\_BTREE\_HEADER 에서 확인할 수 있다.

LOB 칼럼에는 인덱스를 생성할 수 없다.

---

## 예제

〈질의 1〉 사원 테이블에서 다음 칼럼을 조인하여 인덱스 emp\_idx2 를 생성하라.

사원번호(eno): ASC  
부서번호(dno): ASC

```
iSQL> CREATE INDEX emp_idx2
    ON employee (eno ASC, dno ASC);
Create success.
```

〈질의 2〉 사원 테이블에서 다음 칼럼을 이용하여 유니크 인덱스 emp\_idx2 를 생성하라. 사원 테이블에 레코드가 전혀 없거나 칼럼 dno 내의 값이 unique 한 값만 존재 할 때 생성할 수 있다.

부서번호(dno): DESC

```
iSQL> CREATE UNIQUE INDEX emp_idx2
    ON employee (dno DESC);
Create success.
```

〈질의 3〉 테이블 employee에서 다음 칼럼을 이용하여 BTREE 인덱스 emp\_idx3를 생성하라. 이미 사원 테이블에 eno에 PRIMARY KEY가 생성되어 있는 상태이기 때문에 기본키 제약을 삭제한 후에 인덱스 emp\_idx3를 생성할 수 있다. 그렇지 않으면, [ERR-3104C: Duplicated index names. The specified index name already exists on the database.] 에러가 발생한다.

사원번호(eno): ASC

```
iSQL> ALTER TABLE employee  
        DROP PRIMARY KEY;  
Alter success.  
iSQL> CREATE INDEX emp_idx3  
        ON employee (eno ASC)  
        INDEXTYPE IS BTREE;  
Create success.
```

〈질의 4〉 테이블 employee에서 다음 칼럼을 이용하여 BTREE, PERSISTENT 인덱스를 가진 emp\_idx1을 생성하라.

사원번호(eno): ASC  
부서번호(dno): ASC

```
iSQL> CREATE INDEX emp_idx1  
        ON employee (eno ASC, dno ASC)  
        INDEXTYPE IS BTREE SET PERSISTENT = ON;  
Create success.
```

\* 인덱스가 PERSISTENT인지의 여부는 메타 테이블 SYSTEM\_.SYS\_INDICES\_에서 볼 수 있으며 (e.g. iSQL> DESC SYSTEM\_.SYS\_INDICES\_);, 칼럼 IS\_PERS는 그 색인이 persistent 모드이면 'T', non\_persistent인 경우 'F'로 나타난다.

〈질의 5〉 user\_data 테이블스페이스에 table\_user 테이블의 열 i1에 인덱스 idx1을 생성하라.

```
iSQL> CREATE INDEX idx1  
        ON table_user (i1)  
        TABLESPACE user_data;  
Create success.
```

〈질의 6〉 user\_data 테이블스페이스에 table\_user 테이블의 열 i1에 인덱스 idx2을 병렬 옵션으로 생성하라.

```
iSQL> CREATE INDEX idx1  
        ON table_user (i1)  
        TABLESPACE user_data PARALLEL 4;  
Create success.
```

〈질의 7〉 아래 쿼리는 product\_id를 기준으로 각 테이블 파티션에 맞는 로컬 인덱스를 자동으로 생성한다. 파티션의 이름은 자동 생성된다.

```
CREATE INDEX prod_idx ON products(product_id) LOCAL;
```

〈질의 8〉 아래 쿼리는 각각의 인덱스 파티션을 지정해서 로컬 인덱스를 생성한다.

```
CREATE INDEX prod_idx ON products(product_id)
LOCAL
(
    PARTITION p_idx1 ON p1 TABLESPACE tbs_disk1,
    PARTITION p_idx2 ON p2 TABLESPACE tbs_disk2,
    PARTITION p_idx3 ON p3 TABLESPACE tbs_disk3
);
```

〈질의 9〉 인덱스 파티션을 부분적으로 지정한 경우이다. 지정하지 않은 로컬 인덱스는 자동으로 생성된다.

```
CREATE INDEX prod_idx ON products(product_id)
LOCAL
(
    PARTITION p_idx1 ON p1 TABLESPACE tbs_disk1,
    PARTITION p_idx3 ON p3 TABLESPACE tbs_disk3
);
```

〈질의 10〉 테이블 employee에서 사원번호(eno)로 인덱스 idx1을 생성하되 시스템 고장이나 미디어 고장이 발생하더라도 사용할 수 있게(LOGGING) 하라.

```
iSQL> CREATE INDEX idx1
      ON employee (eno);
Create success.
```

또는

```
iSQL> CREATE INDEX idx1
      ON employee (eno) LOGGING ;
Create success.
```

〈질의 11〉 테이블 employee에서 다음 칼럼을 이용하여 인덱스 idx1을 NOLOGGING 옵션으로 생성하라(단, 인덱스 생성 후 시스템 고장이 발생하더라도 인덱스가 사용가능해야 함: FORCE).

```
사원번호 (eno) : ASC
부서번호 (dno) : ASC
iSQL> CREATE INDEX idx1
      ON employee (eno ASC, dno ASC)
      NOLOGGING;
Create success.
```

또는

```
사원번호 (eno) : ASC
부서번호 (dno) : ASC
iSQL> CREATE INDEX idx1
      ON employee (eno ASC, dno ASC)
      NOLOGGING FORCE;
Create success.
```

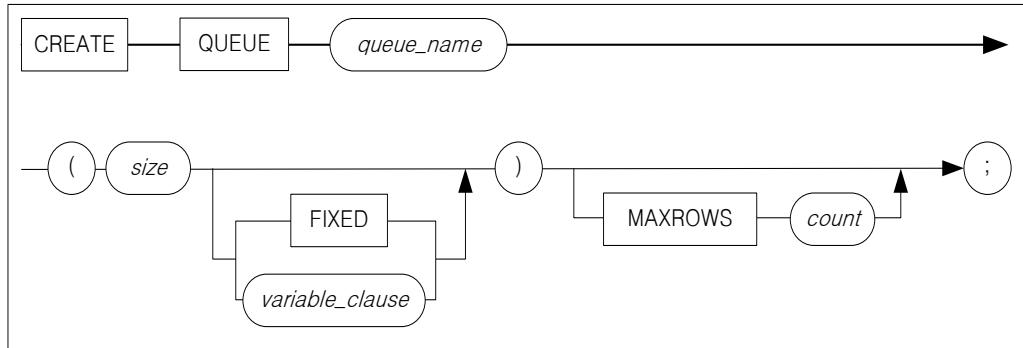
〈질의 12〉 테이블 employee에서 다음 칼럼을 이용하여 인덱스 idx1 을 NOLOGGING 옵션으로 생성하고, 디스크에 반영하지 않게(NOFORCE) 하라.

```
사원번호 (eno) : ASC  
부서번호 (dno) : ASC  
iSQL> CREATE INDEX idx1  
        ON employee (eno ASC, dno ASC)  
        NOLOGGING NOFORCE;  
Create success.
```

# CREATE QUEUE

## 구문

*create\_queue ::=*



## 설명

큐에 삽입 가능한 최대 메시지의 길이를 지정하여 큐를 생성하는 구문이다. 해당 큐 테이블의 최대 레코드 수도 지정할 수 있다.

*queue\_name ( size )*

큐의 이름과 함께 큐에 저장될 메시지의 최대 길이(byte)를 지정한다.  
사용가능한 최대 큐이름의 길이는 32 바이트 이다.

*FIXED | variable\_clause*

메시지의 저장 방식을 지정한다. (VARCHAR 데이터 타입 참고)

*MAXROWS count*

큐테이블에 저장할 최대 레코드 수를 지정한다.

## 주의사항

큐 생성시에 데이터베이스 내부적으로 “큐 이름”+“\_NEXT\_MSG\_ID”라는 명칭의 오브젝트가 생성된다. 따라서 생성하고자 하는 큐의 이름과 동일한 테이블이나 뷰, SEQUENCE, SYNONYM, 저장 프로시저, “큐 이름”+“\_NEXT\_MSG\_ID”등의 오브젝트가 이미 존재하는 경우에 CREATE QUEUE 문장은 에러를 리턴한다.

생성된 큐는 내부적으로 다음과 같은 구조를 가진다.

Column name	Type	Description
MSGID	INTEGER	메시지 식별자
MESSAGE	VARCHAR	메시지
CORRID	integer	사용자가 지정한 메시지 식별자
ENQUEUE_TIME	DATE	메시지 입력 시간

---

## 예제

〈질의〉 메시지의 길이가 최대 40이고, 레코드 개수가 1,000,000인 Q1이라는 이름의 큐를 생성하라.

```
CREATE QUEUE Q1(40) MAXROWS 1000000;
```

생성된 큐를 확인한다.

```
iSQL> desc Q1;  
[ TABLESPACE : SYS_TBS_MEM_DIC ]  
[ ATTRIBUTE ]
```

NAME	TYPE	IS NULL	
MSGID	BIGINT	FIXED	NOT NULL
MESSAGE	VARCHAR(40)	FIXED	
CORRID	INTEGER	FIXED	
ENQUEUE_TIME	DATE	FIXED	
[ INDEX ]			

NAME	TYPE	IS UNIQUE	COLUMN
_SYS_IDX_ID_123	BTREE	UNIQUE	MSGID ASC
[ PRIMARY KEY ]			

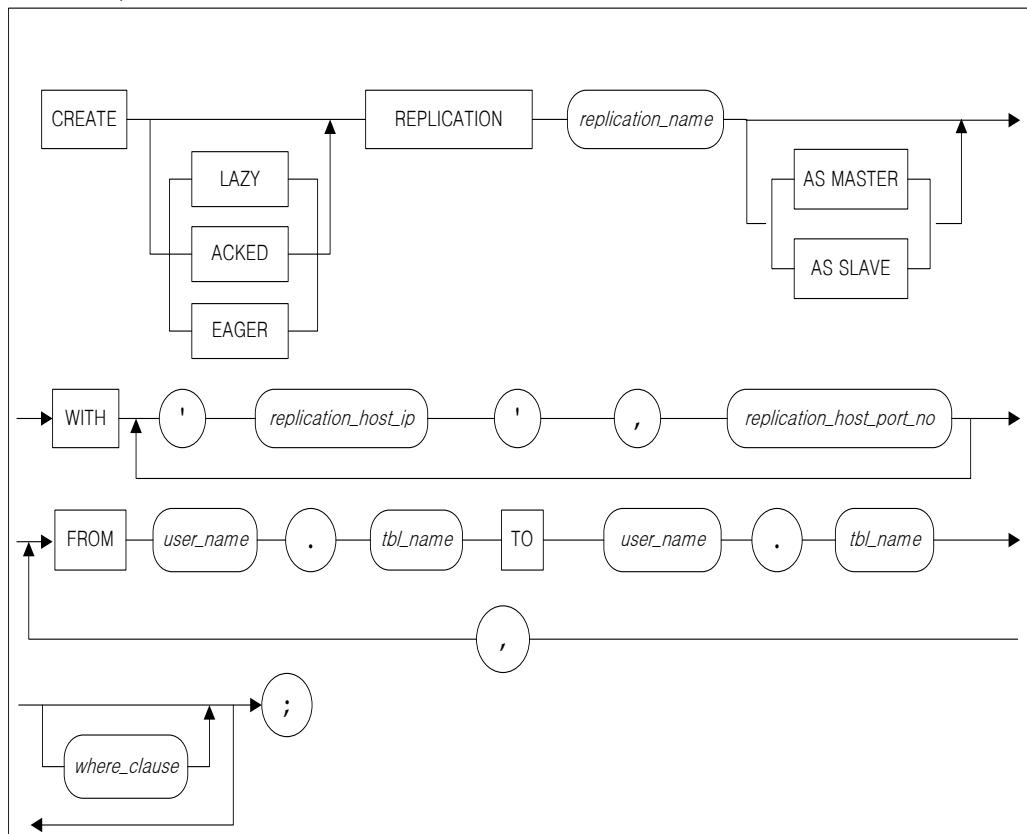
  

MSGID
-------

# CREATE REPLICATION

## 구문

*create\_replication ::=*



## 설명

이중화를 사용하기 위해 지역 서버에서 원격 서버로 연결을 설정한다.

이중화를 생성할 때 최대 32 개의 다른 원격 서버와 연결할 수 있으며, 테이블 단위로 1: 1 매칭만 가능하다.

이중화를 생성할 때 충돌을 해결하기 위해 구문에 as master 또는 as slave 를 지정하는 Master-Slave scheme 을 사용할 수 있다. 만약 지정하지 않았다면, REPLICATION\_UPDATE\_REPLACE 를 사용하는 방식과 동일하다. 자세한 내용은 *Replication User's*

*Manual* 을 참고한다.

#### *replication\_name*

이중화 이름을 명시하며, 지역 서버와 원격 서버가 동일해야 한다.

#### *replication\_host\_ip*

원격 서버의 IP 주소값을 입력한다.

#### *replication\_host\_port\_no*

원격 서버의 수신 쓰레드 포트번호를 입력한다. 알티베이스 프로퍼티 `REPLICATION_PORT_NO`를 참고한다.

#### *user\_name*

이중화 테이블의 소유자 이름을 명시한다.

#### *tbl\_name*

이중화 테이블 이름을 명시한다.

#### *where\_clause*

SELECT 구문의 `where_clause` 를 참고한다.

이중화 조건절은 WHERE `user_name.table_name.column_name`  
`{< | > | ◇ | >= | <= | = | !=} value [{AND | OR} ... ]` 사용하여  
조건에 해당하는 로그만 이중화 대상으로 한다.

---

## 주의 사항

### 선행조건

입력/수정/삭제시 충돌(confliction)이 발생하면 무시(discard)  
처리하고 에러 로그 파일에 남긴다.

\* 에러 로그 파일에 제외되는 경우

- deadlock: 원격서버에서 데드락으로 인해 이중화 트랜잭션이  
롤백될 경우에는 에러로그를 남기지 않는다.
- network error: 이중화 도중 네트워크(TCP/IP) 에러로 인한  
데이터 손실을 막을수 없다. 또한 에러 로그도 남기지 않는다.

이중화 작업 도중 발생한 에러에 대해 부분 롤백한다. 즉 여러 개의  
자료 입력 중 한 개의 중복 데이터가 있으면 중복 데이터만 취소하고  
나머지는 완료할 수 있다.

이중화 속도는 서비스 속도보다 매우 느릴 수 있다.

알티베이스 프로퍼티의 `AUTO_REMOVE_ARCHIVE_LOG = 1` 로  
세팅한 경우라도 완료되지 않은 이중화 작업 로그는 로그 파일에서

삭제되지 않는다. 만일 0 으로 세팅할 경우 아래의 5 번은 이중화 작업에 영향이 없다.

현재 이중화 송신 쓰레드가 작업하는 로그 파일과 현재 로그 파일의 차이가 알티베이스 프로퍼티의 최대로그파일 개수(REPLICATION\_MAX\_LOGFILE)보다 큰 시점에 체크 포인트가 발생하면 이중화 작업이 완료되지 않은 로그 파일도 삭제되며, 이중화 작업은 이전의 로그 파일은 무시하고 현재 로그 파일부터 다시 시작한다.

## 데이터 제약조건

복제되는 테이블에 대해 반드시 기본키가 존재해야 한다.

복제되는 테이블에 대해 기본키에 대한 수정이 없어야 한다.

양쪽 테이블의 칼럼 정보, 기본키, NOT NULL 정보가 동일해야 한다.

이중화를 위해 만들어진 테이블은 이중화 개시 이 후 ALTER/DROP 할 수 없다.

## 연결 제약조건

최대 이중화 연결은 하나의 알티베이스 데이터베이스에 32 개이다.

---

## 예제

〈질의〉 다음 조건을 만족하는 이중화 *rep1*을 생성하라.

지역 서버의 IP 주소가 192.168.1.60 이고 포트 번호가 25524, 원격 서버의 IP 주소가 192.168.1.12 이고 포트 번호가 35524 이라고 하자. 두 서버간의 employee 테이블과 department 테이블을 이중화 한다.

지역 서버의 경우 (IP: 192.168.60)

```
iSQL> CREATE REPLICATION rep1
      WITH '192.168.1.12',35524
      FROM sys.employee TO sys.employee,
           FROM sys.department TO sys.department;
Create success.
```

원격 서버의 경우 (IP: 192.168.1.12)

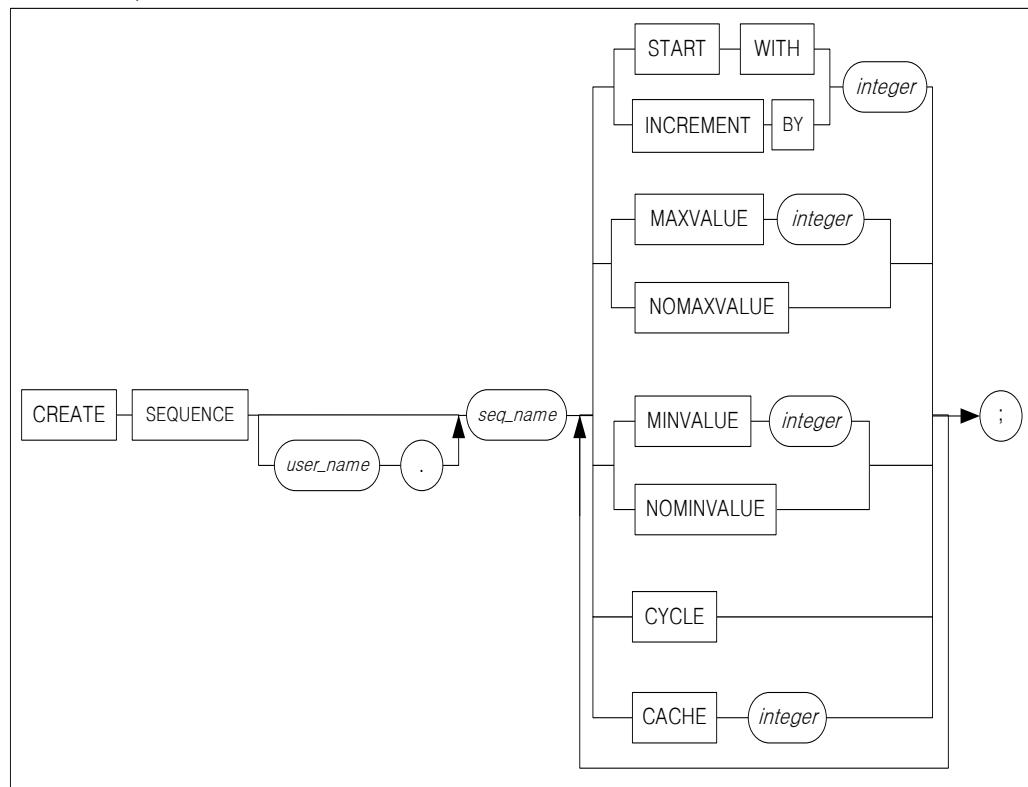
```
iSQL> CREATE REPLICATION rep1
      WITH '192.168.1.60',25524
      FROM sys.employee TO sys.employee,
           FROM sys.department TO sys.department;
```

Create success.

# CREATE SEQUENCE

## 구문

*create\_sequence ::=*



## 전제 조건

SYS 사용자 또는 CREATE SEQUENCE 시스템 권한을 가진 사용자여야 한다.

다른 사용자의 스키마에 시퀀스를 생성하려면 CREATE ANY SEQUENCE 권한을 가져야 한다.

## 설명

명시된 시퀀스명으로 새로운 시퀀스를 정의하여 시퀀스 번호를

자동으로 생성한다.

*user\_name*

생성될 시퀀스의 소유자 이름을 명시한다. 생략하면 알티베이스는 현재 세션에 연결된 사용자의 스키마에 시퀀스를 생성한다.

*seq\_name*

생성될 시퀀스 이름을 명시한다.

START WITH

시퀀스의 시작값

INCREMENT BY

시퀀스의 증감분

MAXVALUE

시퀀스의 최대값

MINVALUE

시퀀스의 최소값

CYCLE

시퀀스가 MAXVALUE 또는 MINVALUE 한계에 도달하면 해당 시퀀스의 추가 값을 할당 하기 위하여 오름차순 시퀀스인 경우는 minimum value 를 내림차순 시퀀스인 경우는 maximum value 를 생성한다.

CACHE

해당 시퀀스 값을 더 빠르게 액세스 하기 위하여 명시된 값 만큼 시퀀스 값을 메모리에 캐시한다. 캐시는 시퀀스를 처음 참조할 때 채워지며 다음에 시퀀스 값을 요청할 때마다 캐시 된 시퀀스에서 검색된다. 마지막 시퀀스를 사용한 이후의 시퀀스 요청은 시퀀스의 다른 캐시를 메모리로 가져온다. 시퀀스 생성시 기본값은 20

\* *sequence\_name*.CURRVAL 을 사용하기 위해서는 시퀀스 생성 후 *sequence\_name*.NEXTVAL 을 먼저 사용한 후 여야만 한다.

---

## 예제

〈질의〉 다음 SQL 문들을 이용하여 새로운 시퀀스를 정의하고 시퀀스 값과 정보를 확인해본다.

```
iSQL> CREATE TABLE seqtbl(i1 INTEGER);
Create success.
iSQL> CREATE OR REPLACE PROCEDURE proc1
AS
```

```

BEGIN
  FOR i IN 1 .. 10 LOOP
    INSERT INTO seqtbl VALUES(i);
  END LOOP;
END;
/
Create success.
iSQL> EXEC proc1;
Execute success.
iSQL> CREATE SEQUENCE seq1
  START WITH 13
  INCREMENT BY 3
  MINVALUE 0 NOMAXVALUE;
Create success.
-> 13 부터 시작해서 3씩 증가하고 최소값이 0, 최대값이 무한대인
seq1을 생성

```

```

iSQL> INSERT INTO seqtbl VALUES(seq1.NEXTVAL);
1 row inserted.
iSQL> INSERT INTO seqtbl VALUES(seq1.NEXTVAL);
1 row inserted.
iSQL> SELECT * FROM seqtbl;
SEQTBL.I1

```

---

```

1
2
3
4
5
6
7
8
9
10
13
16
12 rows selected.
iSQL> ALTER SEQUENCE sys.seq1
  INCREMENT BY 50
  MAXVALUE 100
  CYCLE;
Alter success.

```

-> seq1을 50씩 증가 시키되 최대값 100에 도달한 경우에는 다시 최소값부터 시작하여 주어진 조건에 맞게 시퀀스를 변경한다.

```

iSQL> INSERT INTO sys.seqtbl VALUES(seq1.NEXTVAL);
1 row inserted.
iSQL> INSERT INTO sys.seqtbl VALUES(seq1.NEXTVAL);
1 row inserted.
iSQL> INSERT INTO sys.seqtbl VALUES(seq1.NEXTVAL);
1 row inserted.
iSQL> INSERT INTO sys.seqtbl VALUES(seq1.NEXTVAL);

```

```
1 row inserted.  
iSQL> SELECT * FROM sys.seqtbl;  
SEQTBL.I1  
-----  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
13  
16  
66  
0  
50  
100  
16 rows selected.  
iSQL> SELECT seq1.CURRVAL FROM seqtbl;  
SEQ1.CURRVAL  
-----  
100  
. . .  
100  
16 rows selected.
```

→ 새 번호 생성을 위해 seq1의 현재 값을 볼 수 있다.

```
iSQL> UPDATE SEQTBL SET i1 = seq1.NEXTVAL;  
16 rows updated.  
→ 열 i1은 0으로 변경됨  
iSQL> SELECT seq1.CURRVAL FROM seqtbl;  
SEQ1.CURRVAL  
-----
```

```
0  
. . .  
0  
16 rows selected.
```

→ 새 번호 생성을 위해 seq1의 현재 값을 볼 수 있다.

```
iSQL> ALTER SEQUENCE seq1  
INCREMENT BY 2  
MAXVALUE 200  
CACHE 25;  
Alter success.
```

→ 빠른 액세스를 위해 명시된 값 (25 개) 만큼 시퀀스 값을 메모리에 캐시한다.

```
iSQL> CREATE OR REPLACE PROCEDURE proc2
AS
BEGIN
  FOR i IN 1 .. 30 LOOP
    INSERT INTO sqqtbl VALUES(seq1.NEXTVAL);
  END LOOP;
END;
/
Create success.
iSQL> EXEC proc2;
Execute success.
iSQL> SELECT * FROM seqtbl;
SEQTBL.I1
-----
0
50
100
0
50
100
0
50
100
0
50
100
0
50
100
0
50
100
0
2
4
6
8
10
12
14
.
.
.
58
60
46 rows selected.
```

```
iSQL> SELECT * FROM SEQ;
→ SYS 계정으로 데이터베이스에 접속한 경우 모든 sequence 들의 정보를 출력한다.
```

USER\_NAME

SEQUENCE_NAME	CURRENT_VALUE	
INCREMENT_BY		
MIN_VALUE	MAX_VALUE	CYCLE
CACHE_SIZE		
SYS		
SEQ1	0	2
0	200	YES
		25
1 row selected.		

〈질의〉 다음 SQL 문들을 이용하여 여러 계정에서 새로운 시퀀스를 정의하고 시퀀스 값과 정보를 확인해본다.

```
iSQL> CONNECT sys/manager;
Connect success.
iSQL> CREATE USER user1 IDENTIFIED BY user1;
Create success.
iSQL> CREATE USER user2 IDENTIFIED BY user2;
Create success.
iSQL> CONNECT user1/user1;
Connect success.
iSQL> CREATE SEQUENCE seq1 MAXVALUE 100 CYCLE;
Create success.
iSQL> CREATE SEQUENCE seq2;
Create success.
iSQL> SELECT * FROM SEQ;
→ user1 이 생성한 모든 sequence 들의 정보를 출력한다.
SEQUENCE_NAME          CURRENT_VALUE
INCREMENT_BY

MIN_VALUE      MAX_VALUE      CYCLE
CACHE_SIZE

SEQUENCE_NAME          CURRENT_VALUE
INCREMENT_BY

MIN_VALUE      MAX_VALUE      CYCLE
CACHE_SIZE

SEQ1           1             1
1              100            YES           20
SEQ2           1             1
1              9223372036854775806  NO            20
2 rows selected.
iSQL> CONNECT user2/user2;
Connect success.
iSQL> CREATE SEQUENCE seq1 INCREMENT BY -30;
Create success.
iSQL> CREATE SEQUENCE seq2 INCREMENT BY -10 MINVALUE -100;
Create success.
iSQL> CONNECT sys/manager;
Connect success.
iSQL> CREATE SEQUENCE seq2 START WITH 20 INCREMENT BY 30;
Create success.
iSQL> CREATE SEQUENCE seq3 CACHE 40;
Create success.
```

```
iSQL> SELECT * FROM SEQ;
-> SYS 계정으로 데이터베이스에 접속한 경우 모든 sequence 들의
정보를 출력한다.
```

USER\_NAME

SEQUENCE_NAME	CURRENT_VALUE	
INCREMENT_BY		
MIN_VALUE	MAX_VALUE	CYCLE
CACHE_SIZE		
SYS		
SEQ1	60	2
0	200	YES
SYS		
SEQ2	20	30
1	9223372036854775806	NO
SYS		
SEQ3	1	1
1	9223372036854775806	NO
USER1		
SEQ1	1	1
1	100	YES
USER1		
SEQ2	1	1
1	9223372036854775806	NO
USER2		
SEQ1	-1	-30
-9223372036854775806	-1	NO
USER2		
SEQ2	-1	-10
-100	-1	NO
7 rows selected.		

〈질의〉 다음 SQL 문을 이용하여 시퀀스 객체로부터 정보를 확인한다.

```
iSQL> select * from v$seq;
-> 생성되어 있는 시퀀스 객체로부터 정보를 읽어 들인다. Select * from seq 와
달리 다른 사람의 시퀀스 정보를 확인할수 있다. V$SEQ 는 성능 뷰에 해당하며,
Administrator's Manual 의 3 장 데이터 딕션어리의 성능뷰편을 참고한다.
```

V\$SEQ.SEQ\_OID      V\$SEQ.CURRENT\_SEQ      V\$SEQ.START\_SEQ

V\$SEQ.INCREMENT_SEQ	V\$SEQ.SYNC_INTERVAL	V\$SEQ.MAX_SEQ
V\$SEQ.MIN_SEQ	V\$SEQ.LAST_SYNC_SEQ	V\$SEQ.FLAG

34664	104	4
1	100	9223372036854775806
1	104	0
35604	21	1
1	20	9223372036854775806
1	21	0
36544	61	1
1	20	9223372036854775806

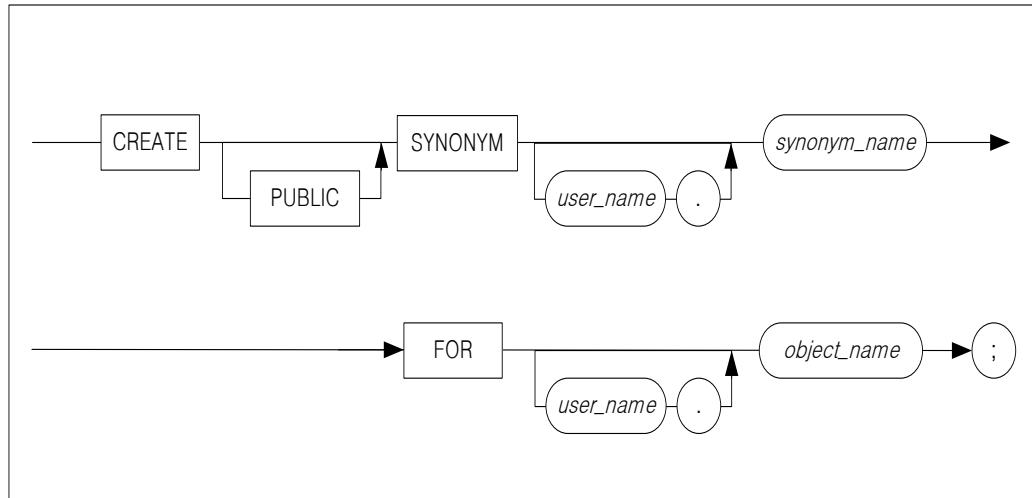
1	61	0
37484	1	1
1	20	9223372036854775806
1	1	0
38424	1	1
20	9223372036854775806	

4 rows selected.

# CREATE SYNONYM

## 구문

*create\_synonym ::=*



## 전제 조건

SYS 사용자 또는 CREATE SYNONYM 시스템 권한을 가진 사용자여야 한다.

다른 사용자의 스키마에 시노님을 생성하려면 CREATE ANY SYNONYM 권한을 가져야 한다.

PUBLIC 시노님을 위해서는 CREATE PUBLIC SYNONYM 권한을 가져야 한다.

## 설명

명시된 시노님명으로 새로운 시노님을 생성하는 SQL 문으로 시노님이란 다음 객체들에 대해 별칭을 정의하는 기능이다.

- 테이블
- 뷰
- 시퀀스

- 저장 프로시저 및 저장 함수
- 다른 시노님

생성한 시노님을 사용할 수 있는 SQL 문은 다음과 같다.

DML 문	DDL 문
SELECT	GRANT
INSERT	REVOKE
UPDATE	
DELETE	
MOVE	
LOCK TABLE	

#### PUBLIC 시노님과 PRIVATE 시노님

PUBLIC 시노님은 모든 사용자가 사용할 수 있는 시노님이며, PRIVATE 시노님은 해당 시노님을 소유한 사용자만 사용할 수 있는 시노님이다.

PUBLIC 시노님을 생성하기 위해서는 생성 시 PUBLIC 을 명시해야 하며, 명시하지 않을 경우 기본적으로는 PRIVATE 시노님을 생성한다.

#### *user\_name*

시노님 앞에 명시하는 사용자명은 시노님 소유자명이다.

PUBLIC 시노님을 생성하는 경우에는 소유자명을 명시하지 않는다.

PRIVATE 시노님을 생성하는 경우에는 소유자명을 명시할 수 있으며 명시하지 않을 경우 현재 CREATE SYNONYM 문을 수행하는 세션의 사용자명으로 설정된다.

#### *synonym\_name*

생성할 시노님명으로 동일한 이름의 테이블, 뷰, 시퀀스, 다른 시노님, 저장프로시저 및 저장함수가 존재할 경우에는 오류를 리턴한다. 시노님은 이를 객체와 동일한 이름 영역을 사용하므로 현재 스키마 내에서 유일한 이름을 명시해야 한다.

#### *FOR clause*

별칭 대상 객체 정보를 명시한다.

#### *user\_name*

별칭을 생성할 객체를 소유한 사용자명을 명시한다.

지정하지 않을 경우에는 현재 시노님을 생성하는 사용자명으로 간주한다.

#### *object\_name*

별칭을 생성할 객체명을 명시한다.

만약 해당하는 객체가 실제 데이터베이스 내에 존재하지 않는 객체라 하더라도 시노님 생성 시에 오류를 리턴하지 않고 시노님 생성은 성공한다.

---

## 시노님을 통한 DML 문 수행 권한

시노님을 사용하여 DML 문을 수행하기 위해서는 사용자가 해당 객체에 대해 수행 권한을 가지고 있어야 한다.

DML 문의 수행 권한은 시노님을 통하여 부여되더라도 시노님의 대상이 되는 객체에 직접적으로 부여되고 박탈된다.

따라서 DML 문 수행 시 권한 오류가 발생하는 경우 실제 객체에 대해 해당 권한이 사용자에게 부여되어 있는지 메타 테이블에서 확인해야 하며, 적절한 권한이 존재하지 않는 경우 실제 객체에 대해 직접 권한을 부여하여도 되고 시노님을 통해서 권한을 부여해도 된다.

해당 객체에 대해 이미 부여받은 권한이 있다면 시노님을 생성한 후 별도로 권한을 부여할 필요는 없다.

또한 시노님을 통하여 객체에 부여한 권한은 해당 시노님이 삭제되더라도 그대로 유지된다. 이는 시노님을 통하여 권한을 부여하더라도 시노님에 대하여 권한을 부여하는 것이 아니라 시노님 대상 객체에 권한이 부여되기 때문이다.

---

## 객체 이름 검색 우선 순위

SQL 문에 사용한 객체 이름의 검색 우선순위는 실제 존재하는 객체 즉, 테이블, 뷰, 시퀀스, 저장프로시저 및 저장함수인지 먼저 검사하고 존재하지 않을 경우 시노님 이름에서 객체 이름을 검색한다.

시노님 내에서는 PRIVATE 시노님이 PUBLIC 시노님보다 우선 순위가 높다.

예를 들면 SQL 문에서 명시한 객체 이름이 데이터베이스에 존재하는지 검사하는 순서는 다음과 같다.

### “SELECT \* FROM NAME” 인 경우

테이블 또는 뷰인지 검사한다.

존재하지 않는 경우 현재 세션에 접속한 사용자의 PRIVATE 시노님인지 검사한다.

존재하지 않는 경우 PUBLIC 시노님인지 검사한다.

## “SELECT \* FROM USER.NAME” 인 경우

테이블 또는 뷰인지 검사한다.

존재하지 않는 경우 현재 세션에 접속한 사용자의 PRIVATE 시노님인지 검사한다.

PUBLIC 시노님의 경우 소유자가 존재하지 않으므로 PUBLIC 시노님인지 검사하지는 않는다.

---

### 예제

〈질의〉 altibase 사용자가 소유한 dept 테이블에 대해 별칭으로 my\_dept라는 시노님을 현재 사용자 소유로 생성하고 DML 문을 수행한다.

```
iSQL> CONNECT altibase/altibase;
Connect success.
iSQL> CREATE TABLE dept
  2 (
  3 id integer,
  4 name char(10),
  5 location varchar(40),
  6 member integer
  7 );
Create success.
iSQL> GRANT INSERT ON dept TO mylee;
Grant success.
iSQL> GRANT SELECT ON dept TO mylee;
Grant success.
iSQL> CONNECT mylee/mylee;
Connect success.
iSQL> CREATE SYNONYM mylee.my_dept FOR altibase.dept;
Create success.
iSQL> INSERT INTO my_dept VALUES (1,'rndn1',NULL,4);
1 row inserted.
iSQL> SELECT * FROM my_dept;
MY_DEPT.ID  MY_DEPT.NAME  MY_DEPT.LOCATION
```

---

MY\_DEPT.MEMBER

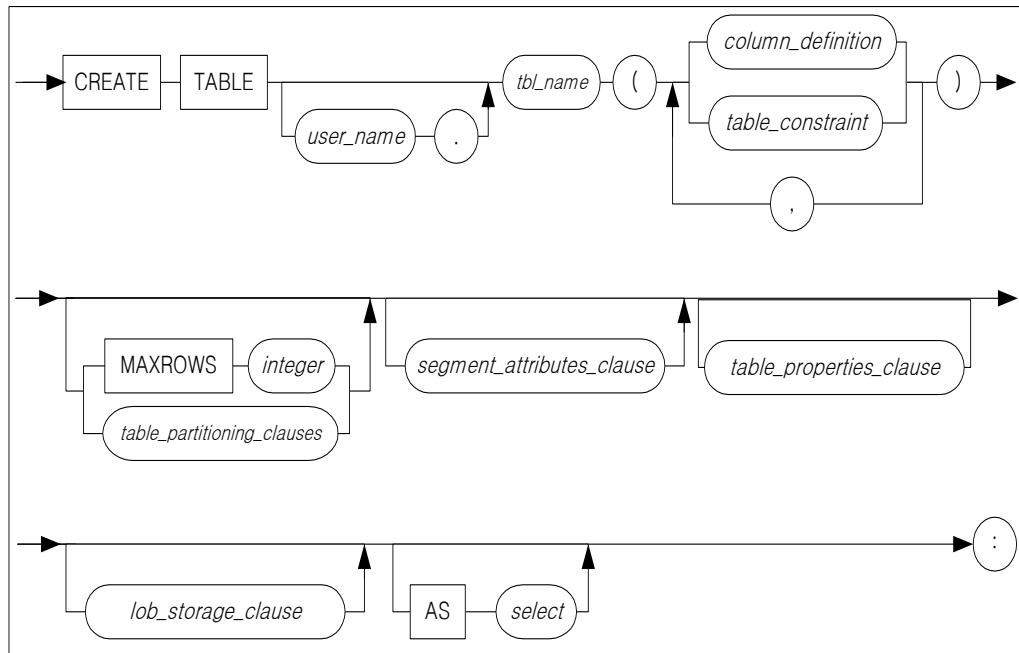
---

```
1          rndn1
4
1 row selected.
```

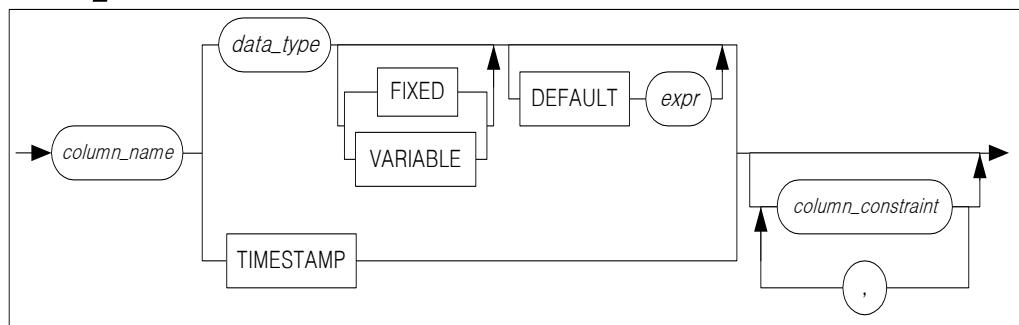
# CREATE TABLE

## 구문

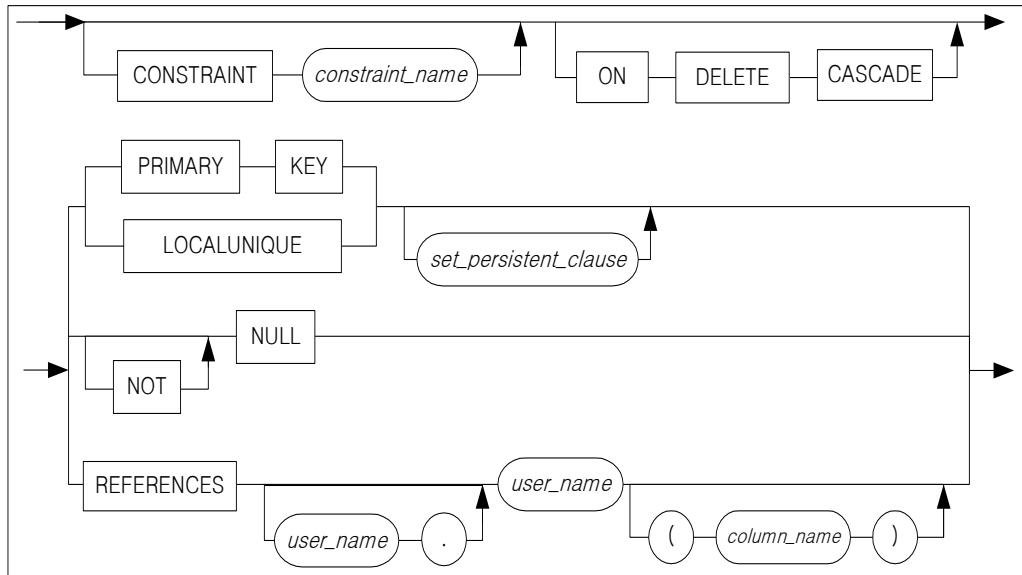
*create\_table ::=*



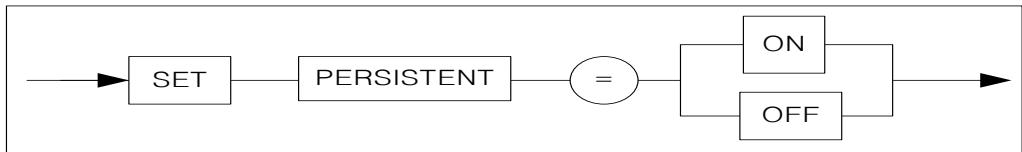
*column\_definition ::=*



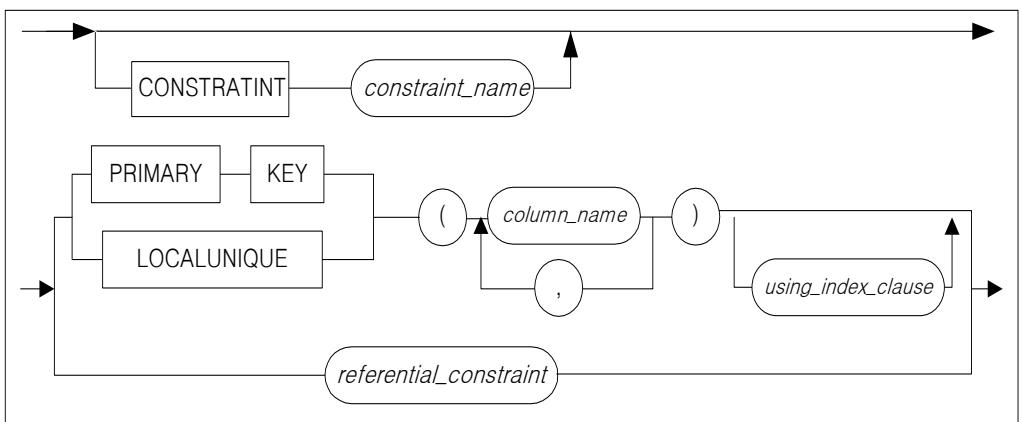
*column\_constraint ::=*



*set\_persistent\_clause ::=*



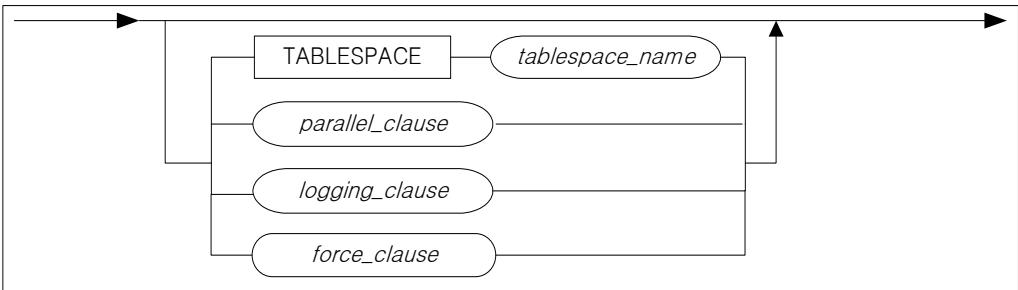
*table\_constraint ::=*



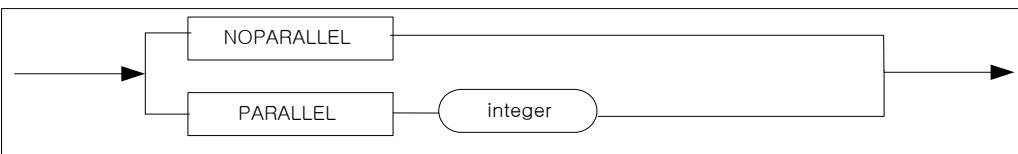
*using\_index\_clause ::=*



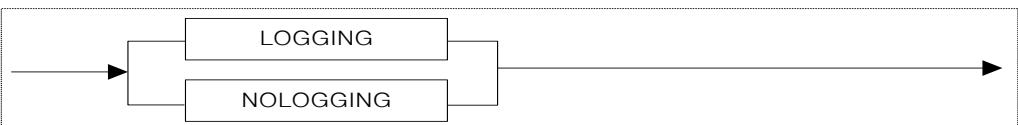
*index\_attribute\_clause ::=*



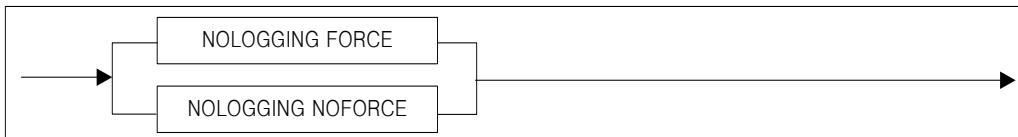
*parallel\_clause ::=*



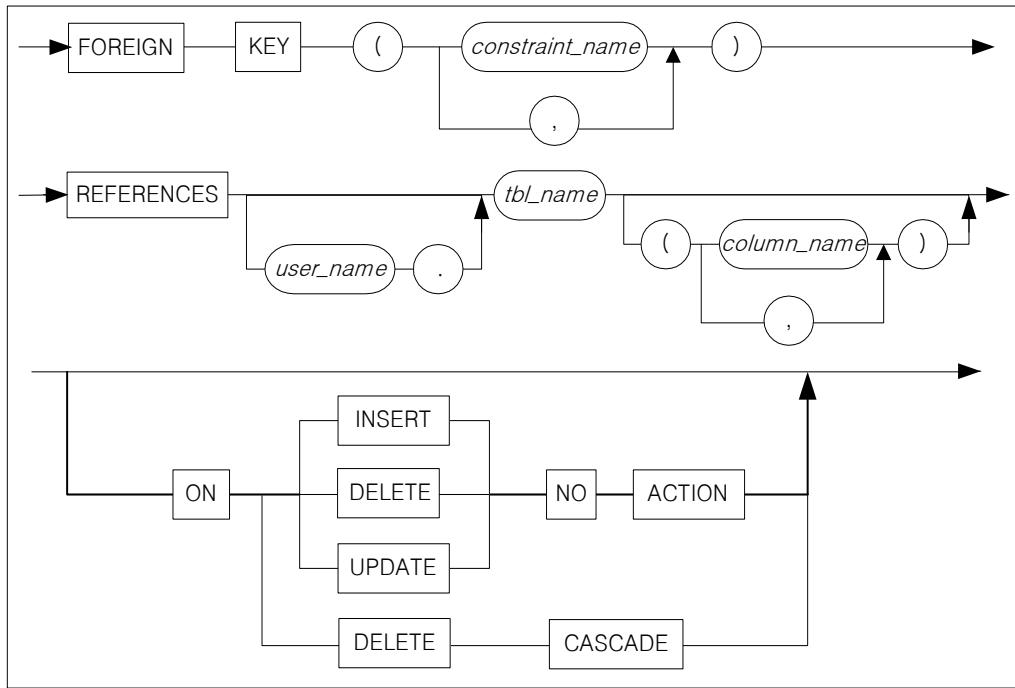
*logging\_clause ::=*



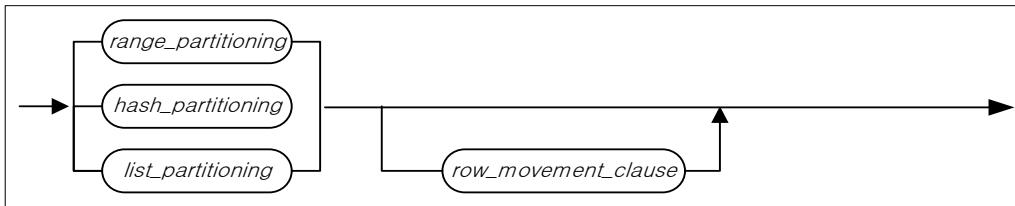
*force\_clause ::=*



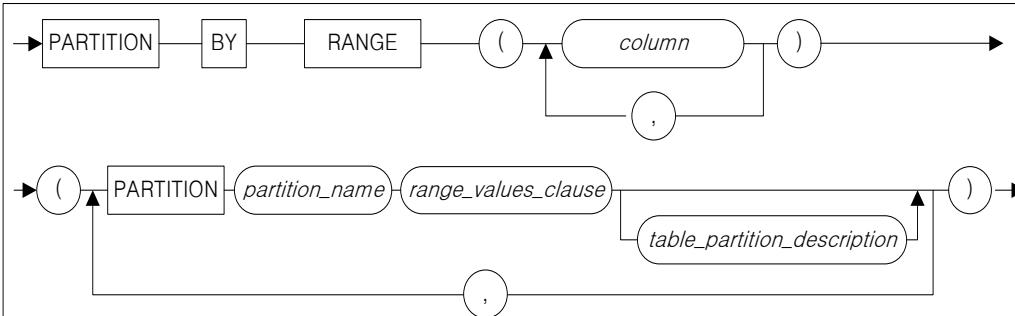
*referential\_constraint ::=*



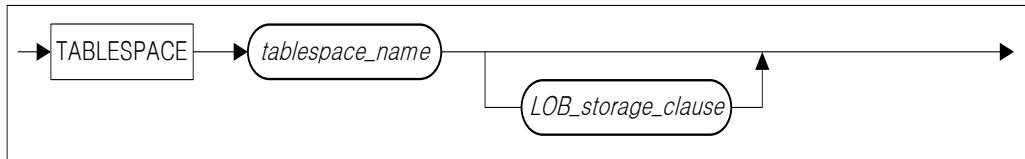
*table\_partitioning\_clause ::=*



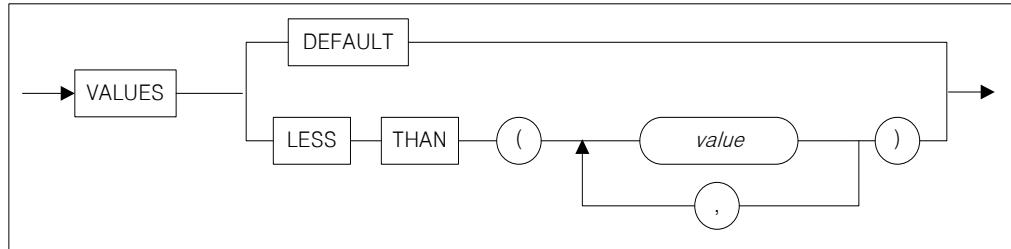
*range\_partitioning ::=*



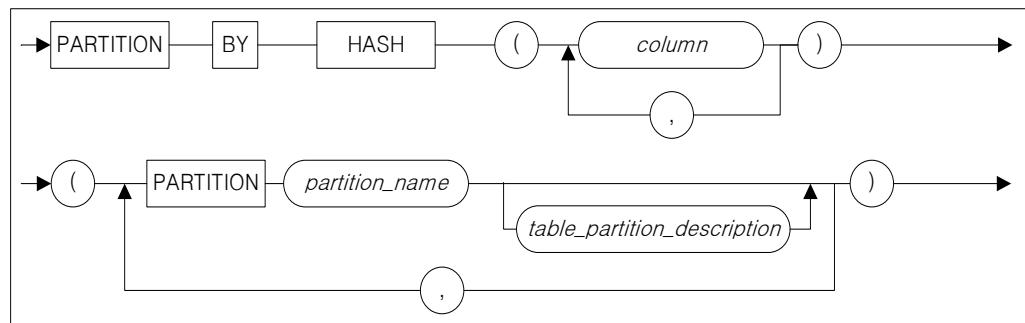
*table\_partition\_description* ::=



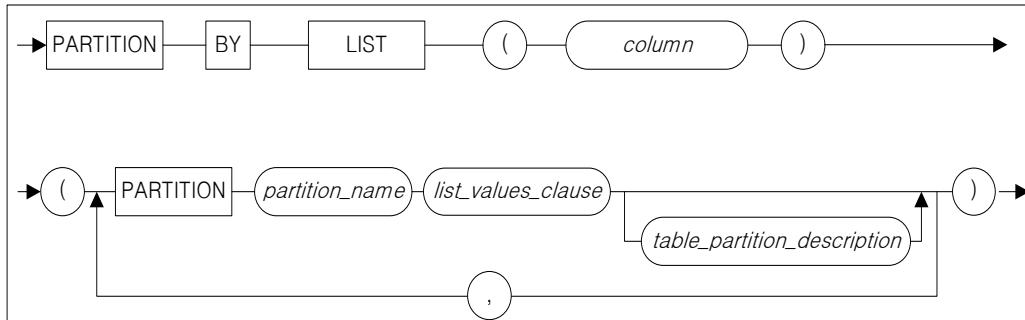
*range\_values\_clause* ::=



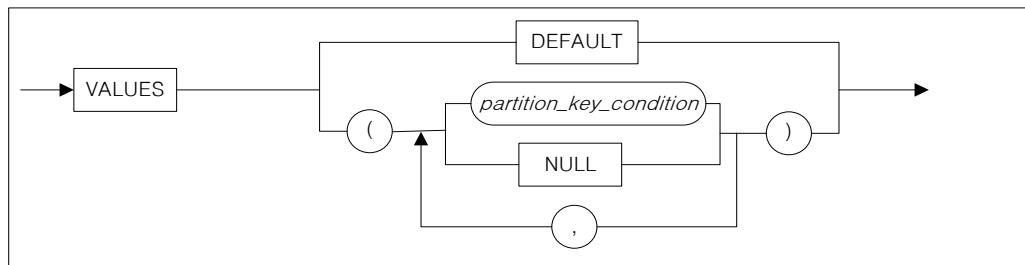
*hash\_partitioning* ::=



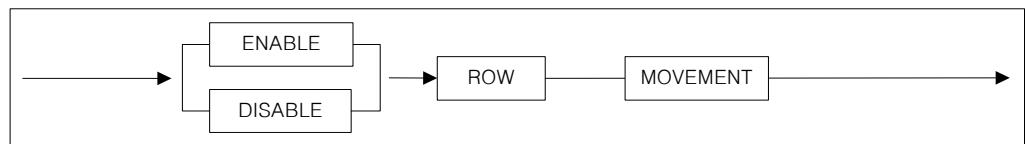
*list\_partitioning* ::=



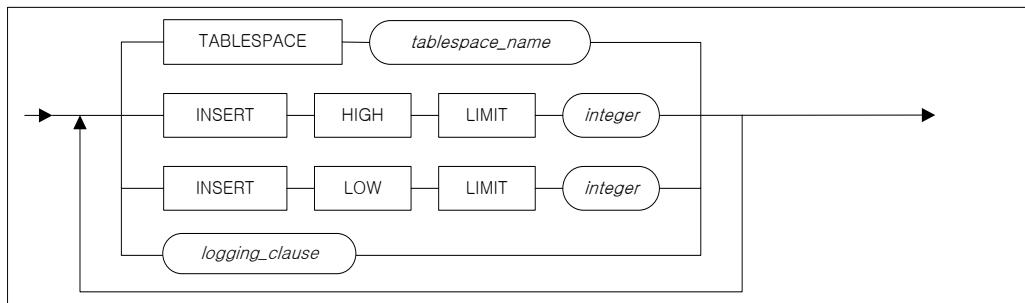
*list\_values\_clause ::=*



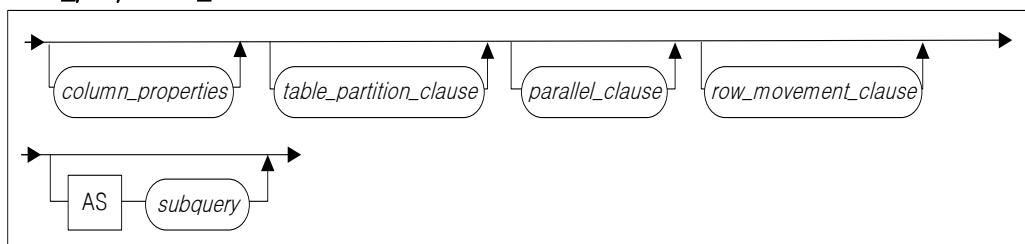
*row\_movement\_clause ::=*



*segment\_attributes\_clause ::=*



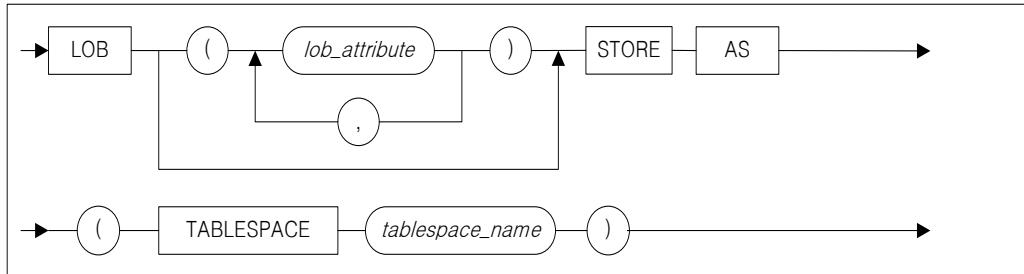
*table\_properties\_clause ::=*



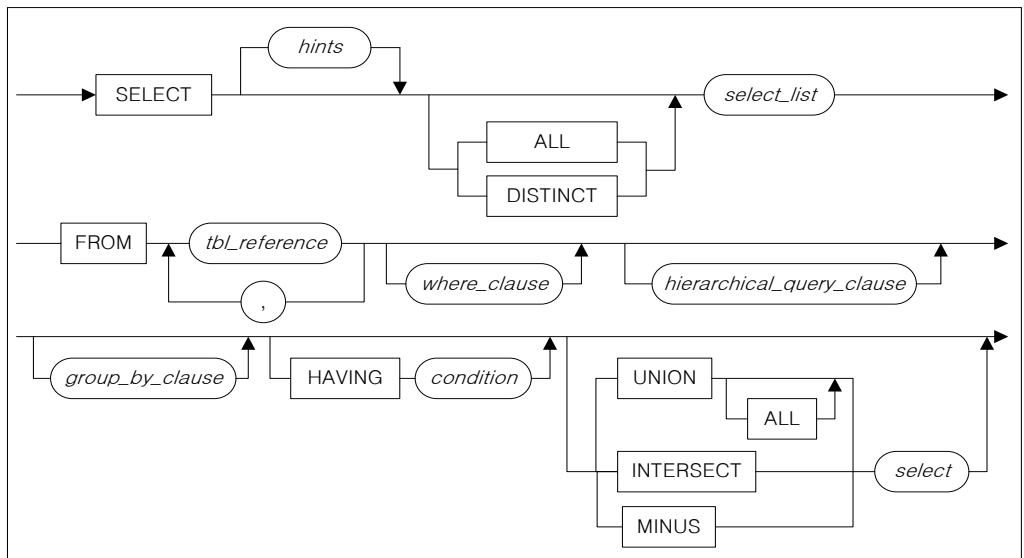
*lob\_storage\_clause ::=*



*lob\_storage ::=*



*select\_clause ::=*



---

## 전제 조건

SYS 사용자이거나 CREATE TABLE 시스템 권한을 가진 사용자만이 테이블을 생성할 수 있다.

---

## 설명

명시된 테이블 명으로 새로운 테이블을 생성한다.

*user\_name*

생성될 테이블 소유자 이름을 명시한다. 생략하면 알티베이스는 현재 세션에 연결된 사용자의 스키마에 테이블을 생성한다.

*tbl\_name*

생성될 테이블 이름을 명시한다.

#### *column definition*

- DEFAULT 칼럼에 DEFAULT 절을 명시하지 않으면 해당 칼럼에 대한 각 행의 초기값은 NULL이다. DEFAULT를 명시한 경우 칼럼에 대한 기존 행을 칼럼 추가 시 명시한 DEFAULT 값으로 변경한다.
- TIMESTAMP 문법상으로 데이터 타입처럼 지원한다. 예를 들어, CREATE TABLE 문에 칼럼 데이터 타입으로 TIMESTAMP를 명시한 경우 내부적으로 데이터 크기가 8Byte인 TIMESTAMP가 생성된다. 그러나 TIMESTAMP 칼럼은 시스템 값을 기본으로 하기 때문에 명시적으로 DEFAULT를 설정할 수 없다. 또한 TIMESTAMP 칼럼은 한 테이블에 하나만 생성할 수 있다.

#### *column constraint*

새로운 테이블을 생성할 때 칼럼에 대한 조건을 설정한다. 하나 이상의 칼럼 제약에 대해 명시적으로 제약 이름을 사용할 수 있으며, 파티션드 테이블인 경우에는 로컬 유니크 속성을 부여한다.

- PRIMARY KEY
- (LOCAL) UNIQUE
- (NOT) NULL
- 참조 무결성(referential integrity) 제약조건
- PERSISTENT: CREATE INDEX 참고
- TIMESTAMP 제약조건

#### PRIMARY KEY

기본키의 같은 테이블 내에서 유일해야 하며 기본키에 속하는 칼럼은 널(NULL) 값을 가질 수 없다. 한 테이블 내에 정의 가능한 기본키의 개수는 하나이며, 최대 32 개 칼럼에 대해 기본 키를 생성할 수 있다.

#### LOCALUNIQUE

각 지역 인덱스별로 UNIQUE를 만족하는 제약 조건이다.

#### UNIQUE

유니크를 정의하면 같은 값을 가지는 2 개 이상의 레코드가 존재할 수 없다. 같은 칼럼 또는 같은 칼럼의 조합에 대해 유니크와 기본키를 동시에 정의할 수 없고 2 개 이상의 유니크도 존재할 수 없다. 그러나 다른 칼럼 또는 칼럼들의 조합이 다른 경우엔 가능하다. 최대 17 개 칼럼에 대해 유니크를 생성할 수 있다.

#### NULL

해당 칼럼이 널 값을 가질 수 있다는 것을 의미한다.

## NOT NULL

---

해당 칼럼이 널 값을 가질 수 없다는 것을 의미한다.

## PERSISTENT 인덱스

---

Persistent 옵션을 사용하지 않은 경우 기본값으로 OFF 이다.

CREATE INDEX 문장을 참고한다.

### *table\_constraint*

한 칼럼 이상의 칼럼들에 대한 제약으로, 하나 이상의 칼럼 제약에 대해 명시적으로 다음의 제약조건을 사용할 수 있다.

- PRIMARY KEY
- LOCALUNIQUE
- 참조 무결성(referential integrity) 제약조건
- using\_index\_clause)

제약 조건을 위해 생성되는 인덱스가 저장될 테이블스페이스를 지정한다.

PRIMARY KEY, UNIQUE 또는 LOCALUNIQUE 제약을 설정할 경우, 생성되는 로컬 인덱스의 테이블스페이스를 각 인덱스 파티션 별로 지정할 수 있다. 자세한 설명은 CREATE INDEX 구문의 *index\_partitioning\_clause*를 참조한다.

### *parallel\_clause*

제약 조건을 위한 인덱스 생성을 빠르게 하기 위해서 PARALLEL 구문을 사용한다.

### *logging\_clause*

로깅(LOGGING)과 노로깅(NOLOGGING) 구문을 사용해 제약조건을 위한 인덱스를 생성할 때 발생하는 로그를 기록하거나 생략할 수 있다. 기본 값은 로그를 기록하며, 인덱스를 생성할 때 로깅을 한다.

### *force\_clause*

생성된 인덱스를 강제로 디스크에 저장할 것인지 여부를 선택하기 위해 포스(FORCE)와 노포스(NOFROCE) 옵션을 사용할 수 있다.

### *references\_constraint*

외래키를 정의한다. 외래키에 의해 참조되는 참조키(referenced key)는 그 테이블에서 유일(unique)하거나 기본키이다. 만약 참조키의 칼럼들이 명시되지 않은 경우 해당 테이블의 기본키가 자동으로 참조키가 된다.

## NO ACTION 참조 행동

---

Insert/Delete/Update 시에 피참조 테이블에 대한 무결성 검사를

연산 이후에 함으로 해서 연산을 정확히 수행할 수 있다. 예를 들어 다음과 같이 테이블을 생성하면, 부서 테이블에서 부서를 삭제한다고 해도 같은 부서 코드를 가진 사원 테이블의 부서 코드 삭제는 실패한다

```
CREATE TABLE EMPLOYEE (
    ENO INTEGER PRIMARY KEY,
    DNO INTEGER,
    NAME CHAR(10),
    FOREIGN KEY(DNO) REFERENCES
        DEPARTMENT(DNO) ON DELETE NO ACTION );
ON DELETE CASCADE
```

부모(Parent) 테이블의 행이 삭제되면 이 행과 연관된 외부 키 값을 가진 자식(child) 테이블들의 행도 모두 삭제된다. 예를 들어 다음과 같이 테이블을 생성하면 부서 테이블에서 행이 삭제될 경우 같은 부서 코드를 가진 사원테이블의 행도 모두 삭제된다.

```
CREATE TABLE EMPLOYEE (
    ENO INTEGER PRIMARY KEY,
    DNO INTEGER,
    NAME CHAR(10),
    FOREIGN KEY(DNO) REFERENCES
        DEPARTMENT(DNO) ON DELETE CASCADE );
```

MAXROWS

테이블에 입력할 수 있는 최대 레코드 개수를 지정한다.

#### *table\_partitioning\_clauses*

테이블을 생성할 때 범위 파티셔닝(range partitioning), 해시 파티셔닝(hash partitioning), 리스트 파티셔닝(list partitioning)으로 파티션드 테이블을 생성할 수 있다.

파티션드 테이블을 만들 때 *row\_movement\_clause*를 명시할 수 있다.

#### *range\_partitioning*

파티션 키 값의 범위를 기준으로 파티셔닝한다. 주로 DATE 자료형에 많이 사용된다. 사용자가 지정한 값으로 파티션이 결정되기 때문에 데이터의 고른 분포는 보장하지 못한다. 각각의 파티션에 상한값을 정하여 파티션의 범위를 결정할 수 있다.

그 외의 모든 값과 NULL은 기본 파티션(default partition)에 속하며, 기본 파티션 구문은 생략할 수 없다. 파티션 키로 컬럼의 리스트가 가능하다.

#### *table\_partition\_description*

테이블스페이스를 지정하거나, LOB 컬럼이 있을 경우 LOB 컬럼에 대해서 따로 속성을 정의할 수 있다.

테이블스페이스 구문이 생략되면, 파티션은 해당 테이블의 기본 테이블스페이스(default tablespace)에 함께 저장된다.

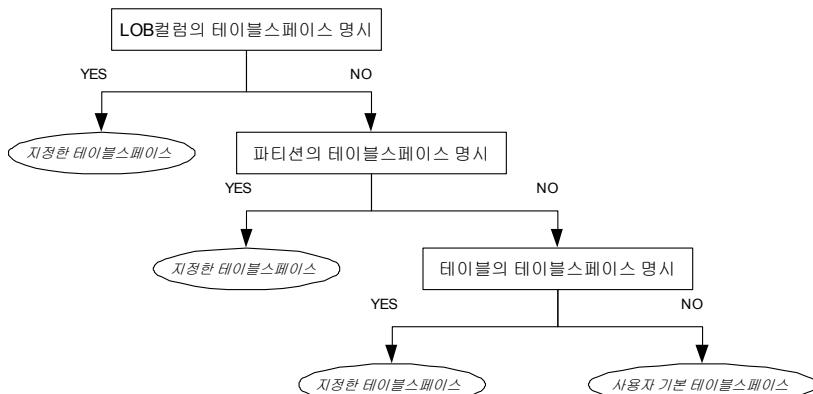
또한 LOB 컬럼에 테이블스페이스 구문을 생략한 경우에는 해당 파티션의 테이블스페이스에 저장된다.

예를 들면 다음과 같다. (사용자 기본 테이블스페이스는 tbs\_05라고 가정한다.)

```
CREATE TABLE print_media_demo
(
    product_id INTEGER,
    ad_photo BLOB,
    ad_print BLOB,
    ad_composite BLOB
)
PARTITION BY RANGE (product_id)
(
    PARTITION p1 VALUES LESS THAN (3000) TABLESPACE tbs_01
        LOB (ad_photo) STORE AS (TABLESPACE
        tbs_02),
    PARTITION p2 VALUES DEFAULT
        LOB (ad_composite) STORE AS (TABLESPACE
        tbs_03)
) TABLESPACE tbs_04;
```

파티션 p1은 테이블스페이스를 명시적으로 지정했으므로 tbs\_01에 저장된다. 그리고 해당 파티션의 ad\_photo 컬럼은 tbs\_02에 저장된다. 기본 파티션인 p2는 테이블스페이스를 지정하지 않았으므로 T1 테이블의 테이블스페이스인 tbs\_04에 저장된다. 만약 테이블의 테이블스페이스도 지정하지 않았다면 기본 테이블스페이스인 tbs\_05에 저장된다.

위의 설명을 그림으로 나타내면 다음과 같다.



#### range\_values\_clause

해당 파티션에 대한 상한값을 지정한다. 이 값은 다른 파티션의 범위와 겹치지 않아야 한다.

### *hash\_partitioning*

파티션 키 값의 해시 값을 기준으로 파티셔닝한다. 데이터의 고른 분산을 원하는 경우에 적합하며, 파티션 키로 컬럼의 리스트가 가능하다.

### *list\_partitioning*

리스트 파티셔닝은 값의 집합을 정의하여 파티셔닝한다. 기본 파티션은 생략할 수 없으며, 해당 컬럼의 모든 도메인(domain) 값을 갖고 있는 파티션이다.

특정 값의 집합으로 새로운 파티션을 명시할 경우 기본 파티션에서는 그 값이 제거된다. 각 파티션이 갖을 수 있는 값들은 서로 중복될 수 없기 때문이다. 파티션 키로 단일 컬럼만 올 수 있다.

### *list\_values\_clause*

각 리스트 파티션은 적어도 1 개 이상의 값을 가져야 한다. 다른 파티션 키의 값과 중복될 수 없다.

### *row\_movement\_clause*

파티션드 테이블을 업데이트하여 파티션 키 값이 바뀌어 다른 파티션으로 이동되어야 하는 경우 데이터에 대해서 자동적으로 이동시킬 것인지 아니면 에러를 발생시킬 것인지를 결정하는 구문이다. 이 절을 명시하지 않으면 DISABLE ROW MOVEMENT 로 설정된다.

### CREATE TABLE … AS SELECT

생성할 테이블의 칼럼 수와 검색 대상의 표현식 개수는 동일해야 한다. 생성할 칼럼의 데이터 타입은 명시할 수 없고, 자동으로 검색 결과에 따라 데이터 타입이 결정된다.

생성할 테이블의 칼럼 명을 명시하지 않을 경우엔 검색 칼럼 명이 생성할 테이블의 칼럼 명이 된다. 검색 대상이 표현식인 경우 별명이 존재해야 하고 그 이름이 칼럼명이 된다.

### *segment\_attributes\_clause*

테이블스페이스와 INSERT HIGH LIMIT, INSERT LOW LIMIT 을 지정하는 구문이다. 파티션드 테이블에 이 구문이 사용되면 INSERT HIGH LIMIT 과 INSERT LOW LIMIT 은 각각의 파티션에 똑같이 적용된다.

- TABLESPACE 절

테이블이 저장될 테이블스페이스를 지정하는 절이다. CREATE TABLE문 시 지정되면 이 후 ALTER TABLE문으로 변경할 수 없다. 이 절을 생략할 경우 사용자 생성 시 지정된 DEFAULT TABLESPACE가 해당 테이블을 위한 테이블스페이스로 지정된다. (사용자 생성 시 DEFAULT TABLESPACE를 생략한

경우 해당 테이블은 메모리에 생성된다.)

CREATE TABLE문 내에 unique key, primary key constraint를 명시하는 경우 이들을 위한 인덱스의 테이블스페이스는 테이블의 테이블스페이스와 동일하다.

- INSERT HIGH LIMIT 절  
페이지에 레코드를 삽입할 수 있는 공간의 비율을 정의한다.  
페이지에 이 비율의 공간만큼만 레코드 insert가 가능하며, 이 비율의 공간을 제외한 나머지 공간은 페이지에 이미 저장되어 있는 레코드를 갱신하기 위한 용도로 사용된다. 이 값은 페이지의 백분율로 표시된다.  
예를 들어 이 값을 80으로 명시된 테이블의 경우, 각 페이지 크기의 80%까지만 레코드를 입력할 수 있으며, 나머지 20%는 레코드의 갱신 용도로만 사용된다. 이 값은 디스크 기반의 테이블에 대해서만 의미가 있다.  
CREATE TABLE문시 지정되면, 이 후 ALTER TABLE문으로 변경할 수 없다. 명시할 수 있는 값은 1에서 100까지 정수이며 백분율을 의미한다. 명시하지 않을 경우 기본값은 90 이다. 이 값은 INSERT LOW LIMIT 보다 커야 한다. 이 값은 테이블을 위해 할당된 페이지에만 적용된다.
- INSERT LOW LIMIT 절  
한 페이지가 다시 레코드 삽입을 위해 사용될 수 있는 페이지 사용공간의 최소 비율을 나타낸다. INSERT HIGH LIMIT에 도달한 페이지에 대해서는 더 이상 레코드가 삽입되지 않고, 갱신, 삭제만 허용된다. 레코드 삭제로 인해 페이지의 사용중인 공간의 비율이 이 값 이하로 떨어지면 페이지에 다시 레코드 삽입이 허용된다.  
예를 들어, 이 값을 40으로 명시한 경우, 임의의 페이지의 사용 공간이 INSERT HIGH LIMIT에 명시된 값에 도달했다고 한다면, 페이지의 사용 공간이 39% 이하가 될 때까지 레코드 삽입을 하지 않는다. 사용된 공간이 40% 이하가 떨어져야 비로소 새로운 레코드를 다시 그 페이지에 삽입할 수 있다. 이 값은 디스크 기반의 테이블에 대해서만 의미가 있다.  
CREATE TABLE문 시 지정되면 이 후 ALTER TABLE문으로 변경할 수 없다. 명시할 수 있는 값은 1에서 100까지 정수이며 백분율을 의미한다. 명시하지 않을 경우 기본값은 40 이다. 이 값은 INSERT HIGH LIMIT 보다 작아야 한다. 이 값은 테이블을 위해 할당된 페이지에만 적용된다.

#### \* 참고

위의 두 절은 페이지 사용의 최적화를 위해 다음과 같은 형태로 함께 사용된다. (만약 INSERT HIGH LIMIT = 80, INSERT LOW LIMIT = 40 으로 지정할 경우)

테이블을 위해 할당된 각 페이지 크기의 20%는 기존 레코드에 대한 변경 연산을 위한 공간으로 예약되며, 이 공간을 제외한 나머지 페이지 크기의 80%가 될 때까지만 새로운 레코드들이 삽입된다.

페이지의 80%까지 레코드가 삽입되면 더 이상 어떠한 새로운 레코드도 페이지에 삽입될 수 없다. 이미 저장된 레코드에 대해서만 변경 연산이 가능하며 변경 연산은 예약해둔 20%의 빈 공간을 사용한다.

레코드의 삭제가 발생하여 사용중인 공간이 40% 이하가 되면 새로운 레코드를 다시 그 페이지에 삽입할 수 있다.

INSERT HIGH LIMIT 값과 INSERT LOW LIMIT 값을 이용하여 페이지의 공간을 사용하는 방법은 위와 같은 방법으로 계속 순환된다.

#### *LOB\_storage\_clause*

디스크 테이블은 LOB 칼럼 데이터를 LOB 칼럼이 속한 테이블과 별도의 테이블스페이스에 저장할 수 있다. 그러나 메모리 테이블은 별도로 저장할 수 없고 테이블과 동일한 테이블스페이스에만 저장할 수 있다.

---

## 주의 사항

칼럼 생성시 크기가 최대 허용 크기를 넘거나 최소 크기 보다 작으면 오류. 각 데이터 형마다 크기는 다르다.

기본키는 한 테이블에 2 개 이상 존재할 수 없다.

참조 제약조건의 경우 외래키와 참조키의 칼럼 개수는 동일해야 한다.

참조 제약조건의 경우 외래키와 참조키의 칼럼 데이터 타입은 동일해야 한다.

한 테이블에 생성할 수 있는 최대 인덱스 개수는 17 개이며, 기본키 또는 유니크 개수의 총합이 17 개를 넘을 수 없다.

CREATE TABLE AS SELECT 의 경우 칼럼 명을 명시하였다면 그 개수는 검색 대상에 명시한 칼럼 개수와 동일해야 한다.

CREATE TABLE AS SELECT 의 경우 CREATE TABLE 문에 칼럼 명을 명시하지 않고 검색 대상에 표현식을 사용한 경우 반드시 생성할 테이블의 칼럼 명으로 사용하기 위해 별명(alias name)이 존재해야 한다.

파티션드 테이블은 MAXROWS 구문을 지원하지 않는다.

범위 파티셔닝과 해시 파티셔닝의 파티셔닝 컬럼은 최대 32 개까지 지정할 수 있다.(인덱스 생성 시 인덱스 컬럼의 개수 제한과

동일하다.)

NOLOGGING(FORCE/NOFORCE)으로 생성된 인덱스는 시스템이나 미디어 고장으로 인덱스의 일관성을 보장하기 어려울 수 있다. 이 경우 'The index is inconsistent.'라는 오류 메시지가 나온다. 이러한 오류를 해결하기 위해 일관성이 깨어진 인덱스를 찾아 DROP 한 후에 해당 인덱스를 다시 생성한다. 인덱스의 일관성은 V\$DISK\_BTREE\_HEADER에서 확인할 수 있다.

로컬 인덱스의 테이블스페이스를 지정할 때, CREATE INDEX 구문과 마찬가지로 파티션드 인덱스의 테이블스페이스를 지정할 수 없다.

---

## 예제

### 테이블 생성

다음 테이블들을 생성하라. (부록: Schema 참조)

- 테이블 이름 – employee  
칼럼 – 사원번호, 사원이름, 직책, 전화번호, 부서번호, 월급,  
성별, 생일, 입사날짜, status

```
iSQL> CREATE TABLE employee(
    eno INTEGER PRIMARY KEY,
    ename CHAR(20) NOT NULL,
    emp_job CHAR(15),
    emp_tel NIBBLE(15),
    dno Byte(2),
    salary NUMBER(10,2) DEFAULT 0,
    sex CHAR(1) DEFAULT 'M' NOT NULL,
    birth Byte(2),
    join_date DATE,
    status CHAR(1) DEFAULT 'H');
Create success.
```

- 테이블 이름 – orders  
칼럼 – 주문번호, 주문일자, 판매사원, 고객주민번호,  
상품번호, 주문수량, 도착 예정일자, 주문상태

```
iSQL> CREATE TABLE orders(
    ono NIBBLE(10),
    order_date DATE,
    eno INTEGER NOT NULL,
    cno CHAR(14) NOT NULL,
    gno Byte(5) NOT NULL,
    qty INTEGER DEFAULT 1,
    arrival_date DATE,
```

```
processing CHAR(1) DEFAULT '0', PRIMARY KEY(ono, order_date));  
Create success.
```

- CREATE TABLE … AS SELECT  
다음 질의는 직원 테이블에서 부서 번호가 'c002'인 조건을 만족하는 데이터를 가진 테이블 dept\_c002를 생성한다.

```
iSQL> CREATE TABLE dept_c002  
      AS SELECT * FROM employee  
      WHERE dno = Byte'C002';  
Create success.
```

- 테이블 이름 – tbl\_timestamp  
칼럼 데이터 타입으로 TIMESTAMP를 사용하여 테이블 생성한다.

```
iSQL> CREATE TABLE tbl_timestamp(  
i1 TIMESTAMP CONSTRAINT const2 PRIMARY KEY,  
i2 INTEGER,  
i3 DATE,  
i4 Byte(8));  
Create success.
```

\* 테이블 tbl\_timestamp 의 정보는 다음과 같다.

```
iSQL> DESC tbl_timestamp;  
[ TABLESPACE: SYS_TBS_MEMORY ]  
[ ATTRIBUTE ]  
-----  
NAME          TYPE          IS  
NULL  
-----  
I1           TIMESTAMP     NOT  
NULL  
I2           INTEGER       FIXED  
I3           DATE         FIXED  
I4           Byte(8)      FIXED  
  
[ INDEX ]  
-----  
NAME          TYPE          IS UNIQUE  
COLUMN  
-----  
CONST2        BTREE        UNIQUE      I1 ASC  
  
[ PRIMARY KEY ]  
-----  
I1
```

명시적으로 Byte(8) 데이터 타입을 선언한 칼럼 i4 와 TIMESTAMP 데이터 타입 칼럼인 i1 을 구별하는 방법은 메타테이블 SYS\_CONSTRAINTS\_와 SYS\_CONSTRAINT\_COLUMNS\_를 조회함으로서 TIMESTAMP constraint

(\_\_SYS\_CON\_TIMESTAMP\_ID\_113) 칼럼을 확인할 수 있다.

\* 참고: TIMESTAMP 칼럼은 INSERT 나 UPDATE 수행 시,  
사용자가 TIMESTAMP 칼럼 값을 DEFAULT로 명시한 경우 각각  
시스템 시간값으로 INSERT 또는 UPDATE 된다.

```
iSQL> INSERT INTO tbl_timestamp VALUES(DEFAULT, 2, '02-FEB-01',
  Byte'A1111002');
  1 row inserted.
iSQL> UPDATE tbl_timestamp SET i1 = DEFAULT, i2 = 102, i3 = '02-FEB-02',
  i4 = Byte'B1111002' WHERE i2 = 2;
  1 row updated.
iSQL> SELECT * FROM tbl_timestamp;
TBL_TIMESTAMP.I1 TBL_TIMESTAMP.I2 TBL_TIMESTAMP.I3
TBL_TIMESTAMP.I4
-----
41EDEF14000ED8B8 102      2002/02/02 00:00:00 B1111002000000000
1 row selected.
```

TIMESTAMP 칼럼은 INSERT 나 UPDATE 수행 시, 사용자가  
TIMESTAMP 칼럼 값을 명시하지 않은 경우 각각 시스템  
시간값으로 INSERT 또는 UPDATE 된다.

```
iSQL> INSERT INTO tbl_timestamp(i2, i3, i4) VALUES(4, '02-APR-01',
  Byte'C1111002');
  1 row inserted.
iSQL> UPDATE tbl_timestamp SET i2=104, i3='02-APR-02', i4=BYTE'D1111002'
  WHERE i2=4;
  1 row updated.
```

(테이블 생성 시 칼럼 i3에 NOT NULL constraint를 명시하는  
경우에도 시스템 시간값으로 INSERT 된다.)

```
iSQL> SELECT * FROM tbl_timestamp;
TBL_TIMESTAMP.I1 TBL_TIMESTAMP.I2 TBL_TIMESTAMP.I3
TBL_TIMESTAMP.I4
-----
41EDEF14000ED8B8 102      2002/02/02 00:00:00 B1111002000000000
41EDEF66000AB65C 104      2002/04/02 00:00:00
D1111002000000000
2 rows selected.
```

## 지정 테이블스페이스에 테이블 생성(CREATE TABLESPACE 및 CREATE USER 예제 참조)

〈질의〉 테이블 소유자가 uare1 인 테이블 tbl1 을 생성하라. (사용자  
생성 시 디폴트 테이블스페이스가 지정 안된 경우)

```
iSQL> CONNECT uare1/rose1;
Connect success.
iSQL> CREATE TABLE tbl1(
  i1 INTEGER,
  i2 VARCHAR(3));
Create success.
```

\* 참고: 사용자 생성 시 디폴트 테이블스페이스가 지정 안된 경우  
메모리에 테이블이 생성 된다.

〈질의〉 사용자 생성 시 지정된 디폴트 테이블스페이스 user\_data 에  
다음 조건을 만족하는 테이블 book 과 inventory 를 생성하라.

book: 칼럼 - 책번호, 책이름, 저자, 판, 출판연도, 가격, 출판사코드 (테이블  
book 에 입력할 수 있는 최대 레코드 개수는 2 개 이다.)  
inventory: 칼럼 - 예약구독번호, 책번호, 상점코드, 구입날짜, 구입량, 지불여부

```
iSQL> CREATE TABLE book(
    isbn CHAR(10) CONSTRAINT const1 PRIMARY KEY SET PERSISTENT = ON,
    title VARCHAR(50),
    author VARCHAR(30),
    edition INTEGER DEFAULT 1,
    publishingyear INTEGER,
    price NUMBER(10,2),
    pubcode CHAR(4)) MAXROWS 2
TABLESPACE user_data;
Create success.
```

```
iSQL> CREATE TABLE inventory(
    subscriptionid CHAR(10) PRIMARY KEY,
    isbn CHAR(10) CONSTRAINT fk_isbn REFERENCES book(isbn),
    storecode CHAR(4),
    purchasedate DATE,
    quantity INTEGER,
    paid CHAR(1))
TABLESPACE user_data;
Create success.
```

또는

```
iSQL> CREATE TABLE inventory(
    subscriptionid CHAR(10),
    isbn CHAR(10),
    storecode CHAR(4),
    purchasedate DATE,
    quantity INTEGER,
    paid CHAR(1),
    PRIMARY KEY(subscriptionid),
    CONSTRAINT fk_isbn FOREIGN KEY(isbn) REFERENCES book(isbn))
TABLESPACE user_data;
Create success.
```

## 각 인덱스 파티션의 테이블스페이스를 지정

〈질의〉 I1 컬럼에 대한 UNIQUE 제약을 갖는 파티션드 테이블  
T1 을 생성하라.

```
CREATE TABLE T1
(
```

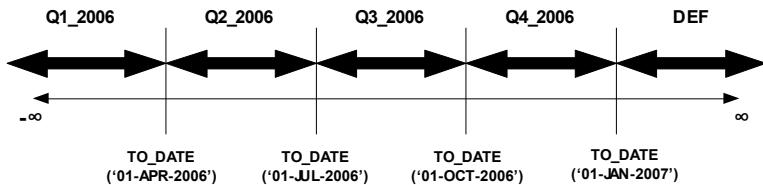
```

I1 INTEGER UNIQUE USING INDEX LOCAL
(
    PARTITION P1_UNIQUE
    ON P1 TABLESPACE TBS3,
    PARTITION P2_UNIQUE
    ON P2 TABLESPACE TBS2,
    PARTITION P3_UNIQUE
    ON P3 TABLESPACE TBS1
)
)
PARTITION BY RANGE (I1)
(
    PARTITION P1 VALUES LESS THAN (100),
    PARTITION P2 VALUES LESS THAN (200) TABLESPACE
    MEM_TBS1,
    PARTITION P3 VALUES DEFAULT TABLESPACE MEM_TBS2
) TABLESPACE SYS_TBS_DISK_DATA;

```

### 범위 파티셔닝(range partitioning)

〈질의 1〉 아래 그림과 같이 2006년의 각 분기별로 파티셔닝하여 range\_sales 테이블을 생성한다.



```

CREATE TABLE range_sales
(
    prod_id          NUMBER(6),
    cust_id          NUMBER,
    time_id          DATE,
)
PARTITION BY RANGE (time_id)
(
    PARTITION Q1_2006 VALUES LESS THAN (TO_DATE('01-APR-
2006')),
    PARTITION Q2_2006 VALUES LESS THAN (TO_DATE('01-JUL-
2006')),
    PARTITION Q3_2006 VALUES LESS THAN (TO_DATE('01-OCT-
2006')),
    PARTITION Q4_2006 VALUES LESS THAN (TO_DATE('01-JAN-
2007')),
    PARTITION DEF VALUES DEFAULT
) TABLESPACE SYS_TBS_DISK_DATA;

```

〈질의 2〉 TABLESPACE 구문을 지정하는 예제

```
CREATE TABLE T1
```

```

(
    I1          INTEGER,
    I2          INTEGER
)
PARTITION BY RANGE (I1)
(
    PARTITION P1 VALUES LESS THAN (100),
    PARTITION P2 VALUES LESS THAN (200) TABLESPACE TBS1,
    PARTITION P3 VALUES DEFAULT TABLESPACE TBS2
) TABLESPACE SYS_TBS_DISK_DATA;

```

〈질의 3〉 다중 컬럼을 파티션 키로 갖는 예제

```

CREATE TABLE T1
(
    I1          DATE,
    I2          INTEGER
)
PARTITION BY RANGE (I1, I2)
(
    PARTITION P1 VALUES LESS THAN (TO_DATE('01-JUL-2006'),
100),
    PARTITION P2 VALUES LESS THAN (TO_DATE('01-JAN-2007'),
200),
    PARTITION P3 VALUES DEFAULT
) TABLESPACE SYS_TBS_DISK_DATA;

```

〈질의 4〉 row\_movement\_clause 를 지정하는 예제

```

CREATE TABLE T1
(
    I1 INTEGER,
    I2 INTEGER
)
PARTITION BY LIST (I1)
(
    PARTITION P1 VALUES (100, 200),
    PARTITION P2 VALUES (150, 250),
    PARTITION P3 VALUES DEFAULT
) ENABLE ROW MOVEMENT TABLESPACE SYS_TBS_DISK_DATA;

```

## 리스트 파티셔닝(list partitioning)

〈질의〉 nls\_territory 컬럼의 값이 ‘CHINA’ 또는 ‘THAILAND’인 asia 파티션, ‘GERMANY’, ‘ITALY’, ‘SWITZERLAND’인 europe 파티션, ‘AMERICA’인 west 파티션, ‘INDIA’인 east 파티션, 그 외 나머지 같은 기본 파티션으로 파티셔닝 할 수 있도록 list\_customers 테이블을 생성한다.

```

CREATE TABLE list_customers
(
    customer_id    NUMBER(6),
    cust_first_name VARCHAR(20),

```

```

        cust_last_name  VARCHAR(20),
        nls_territory    VARCHAR(30),
        cust_email       VARCHAR(30)
    )
PARTITION BY LIST (nls_territory)
(
    PARTITION asia VALUES ('CHINA', 'THAILAND'),
    PARTITION europe VALUES ('GERMANY', 'ITALY', 'SWITZERLAND'),
    PARTITION west VALUES ('AMERICA'),
    PARTITION east VALUES ('INDIA'),
    PARTITION rest VALUES DEFAULT
) TABLESPACE SYS_TBS_DISK_DATA;

```

### 해시 파티셔닝(hash partitioning)

〈질의〉 product\_id 에 따라서 4 개의 해시 파티션으로 분할하여 테이블을 생성한다.

```

CREATE TABLE hash_products
(
    product_id          NUMBER(6),
    product_name        VARCHAR(50),
    product_description VARCHAR(2000)
)
PARTITION BY HASH (product_id)
(
    PARTITION p1,
    PARTITION p2,
    PARTITION p3,
    PARTITION p4
) TABLESPACE SYS_TBS_DISK_DATA;

```

〈질의〉 LOB 파일을 별도의 테이블스페이스에 저장하되, image1 은 테이블스페이스 LOB\_DATA1 에, image2 는 테이블스페이스 LOB\_DATA2 에 저장하는 테이블을 생성한다.

```

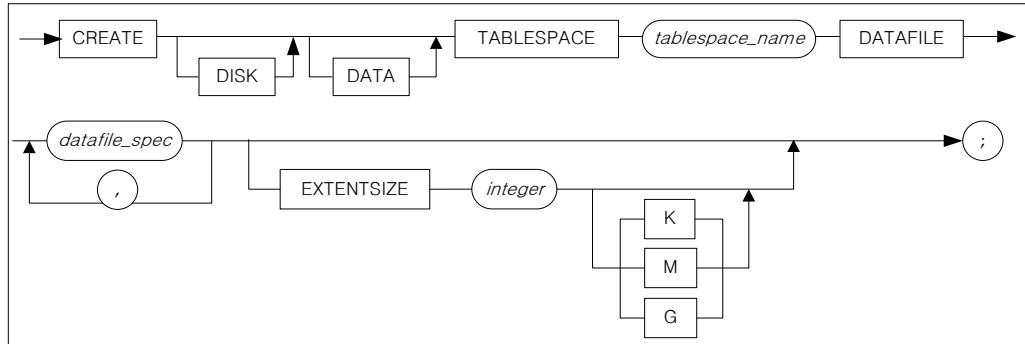
CREATE TABLE lob_products
(
    product_id      integer,
    image1          BLOB,
    image2          BLOB
) TABLESPACE SYS_TBS_DISK_DATA
LOB(image1) STORE AS ( TABLESPACE LOB_DATA1 )
LOB(image2) STORE AS ( TABLESPACE LOB_DATA2 );

```

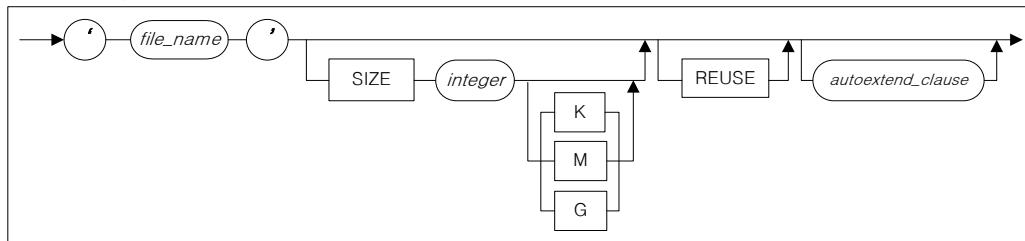
# CREATE DISK TABLESPACE

## 구문

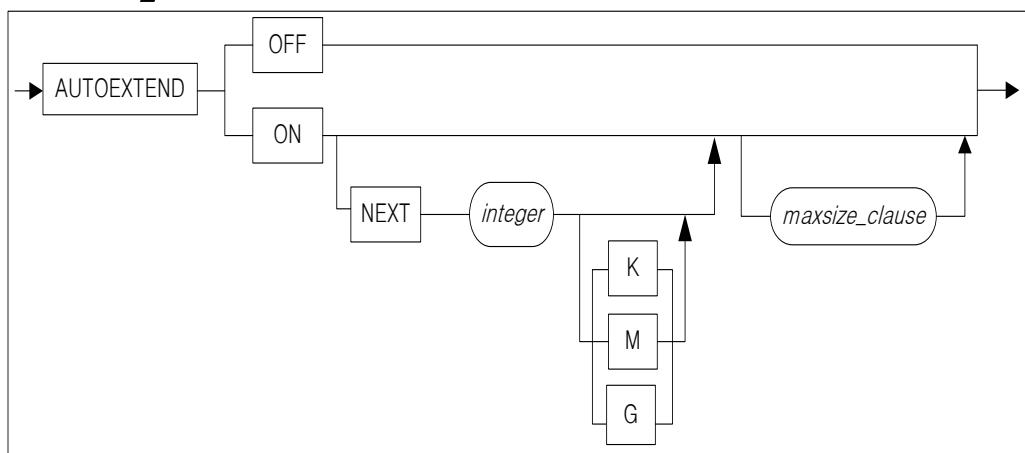
*create\_disk\_tablespace ::=*



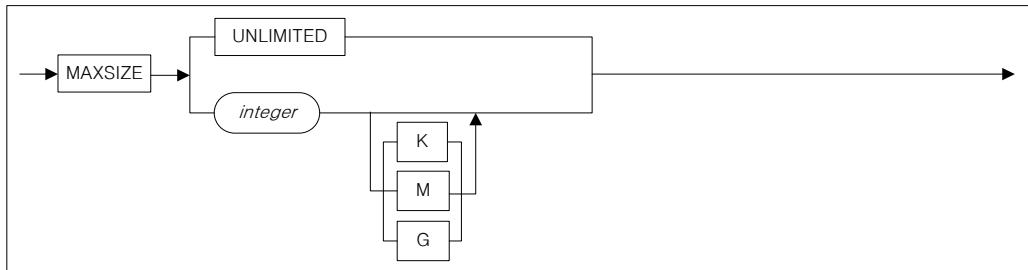
*datafile\_spec ::=*



*autoextend\_clause ::=*



*maxsize\_clause ::=*



## 전제 조건

SYS 사용자이거나 CREATE TABLESPACE 시스템 권한을 가진 사용자만이 테이블스페이스를 생성할 수 있다.

## 설명

데이터베이스 내에 영구적으로 데이터베이스 객체를 저장할 수 있는 디스크 테이블스페이스를 생성하는 구문이다.

이 구문에 의해 생성되는 테이블스페이스에는 테이블과 인덱스가 저장될 수 있다.

DISK

디스크 테이블스페이스를 생성한다. DISK 키워드가 없이 CREATE TABLESPACE 구문을 실행하여도 디스크 테이블스페이스가 생성된다.

DATA

사용자의 데이터가 저장될 테이블스페이스가 생성된다. DATA 키워드가 없이 CREATE TABLESPACE 구문을 실행하여도 데이터 테이블스페이스가 생성된다.

*tblspace\_name*

생성될 테이블스페이스 이름을 명시한다.

*DATAFILE datafile\_spec*

디스크 테이블스페이스를 구성하는 데이터 파일을 명시한다.

EXTENTSIZE 절

PAGE의 집합인 EXTENT의 SIZE를 명시하는 절이다. 생성 시

결정되며, 이후에는 변경할 수 없다.

SIZE 를 명시하지 않을 경우에는 기본 값으로 8 개 페이지의 크기(the size of page \* 8)로 결정되며, 페이지 크기의 배수로 해야 한다. 만약 EXTENT SIZE 가 페이지 크기의 배수로 지정되지 않는다면, 내부적으로 페이지 배수로 수정되어 처리된다.

EXTENT 의 SIZE 는 최소한 페이지 크기의 2 배가 되어야 한다. 예를 들어 페이지의 크기가 32K 라면 EXTENT 의 SIZE 는 최소한 64K 이상으로 지정해야 한다.

Kilobytes(K), Megabytes(M), 또는 Gigabytes(G) 단위로 크기를 명시할 수 있으며 이 단위를 명시하지 않을 경우 기본 단위는 K 이다.

#### *file\_name*

절대 경로를 이용하여 생성될 데이터 파일 이름을 명시한다.

#### SIZE 절

데이터 파일의 크기를 명시한다. 이 절을 명시하지 않으면, 기본 값은 100M 이다. (SYS\_DATA\_TBS\_INIT\_SIZE 에 명시된 값)

Kilobytes(K), Megabytes(M), 또는 Gigabytes(G) 단위로 크기를 명시할 수 있으며, 단위를 명시하지 않을 경우 기본 단위는 K 이다.

#### REUSE

기존 데이터 파일의 재사용 여부를 나타낸다. 명시한 이름을 가진 파일이 이미 존재한다면 반드시 REUSE 를 명시해야 한다. 그러나 존재하는 파일을 재사용할 때에는 기존 데이터가 소실되기 때문에 주의가 필요하다.

그러나 존재하지 않는 파일 이름에 대해 이 옵션을 사용할 경우 이 옵션은 무시되며, 새로운 파일을 생성한다.

#### *autoextend\_clause*

파일에 대하여 확장할 수 있는 최대 공간까지 자동 확장 여부를 명시하는 절이다.

#### ON

파일에 대한 자동 확장이 가능함을 나타낸다.

#### OFF

파일에 대한 자동 확장 불가능함을 나타낸다. 기본값이다.

#### NEXT

파일의 자동 확장 시 다음에 확장할 공간의 크기를 명시한다.

AUTOEXTEND 가 ON 상태이고, 이 값을 명시하지 않을 경우 기본값은 n 개의 EXTENT 크기(the size of extent \* n)가 된다. (ADD\_EXTENT\_NUM\_FROM\_SYSTEM\_TO\_TBS 에 명시된 값)

AUTOEXTEND 가 OFF 일 때는 값이 0 이다.

Kilobytes(K), Megabytes(M), 또는 Gigabytes(G) 단위로 크기를 명시할 수 있으며, 단위를 명시하지 않을 경우 기본 단위는 K 이다.

#### *maxsize\_clause*

파일을 자동 확장할 때 가능한 최대 공간의 크기를 명시한다.

Kilobytes(K), Megabytes(M), 또는 Gigabytes(G) 단위로 크기를 명시할 수 있으며, 단위를 명시하지 않을 경우 기본 단위는 K 이다.

AUTOEXTEND 가 ON 상태이고, 이 값을 명시하지 않을 경우 기본값은 UNLIMITED 이다. AUTOEXTEND 가 OFF 일 때의 값은 0 이다.

#### UNLIMITED

파일의 자동 확장할 수 있는 크기에 제한을 두고 싶지 않은 경우 명시한다.

---

## 예제

〈질의〉 다음은 3 개의 데이터 파일을 가진 user\_data 테이블스페이스를 생성한다.

```
iSQL> CREATE TABLESPACE user_data  
        DATAFILE '/tmp/tbs1.user' SIZE 10M,  
        '/tmp/tbs2.user' SIZE 10M,  
        '/tmp/tbs3.user' SIZE 10M;  
Create success.
```

〈질의〉 테이블스페이스를 구성하는 데이터 파일이 tbs.user 인 10MB 의 user\_data 테이블스페이스를 생성한다. (user\_data 테이블스페이스에 기록되는 테이블이나 인덱스는 tbs.user 파일에 저장된다.)

```
iSQL> CREATE TABLESPACE user_data DATAFILE '/tmp/tbs.user' SIZE 10M  
        AUTOEXTEND ON;  
Create success.
```

〈질의〉 User\_data 테이블스페이스를 생성한다. 더 큰 공간이 요구될 때 500K 크기의 데이터 파일이 최대 크기 100M 까지 자동 확장된다.

```
iSQL> CREATE TABLESPACE user_data  
        DATAFILE '/tmp/tbs.user' SIZE 500K REUSE  
        AUTOEXTEND ON NEXT 500K MAXSIZE 100M;  
Create success.
```

〈질의〉 테이블스페이스 user\_data 를 생성한다. 데이터 파일 tbs.user 는 자동확장하지 않는다.

```
iSQL> CREATE TABLESPACE user_data
      DATAFILE '/tmp/tbs.user' AUTOEXTEND OFF;
Create success.
```

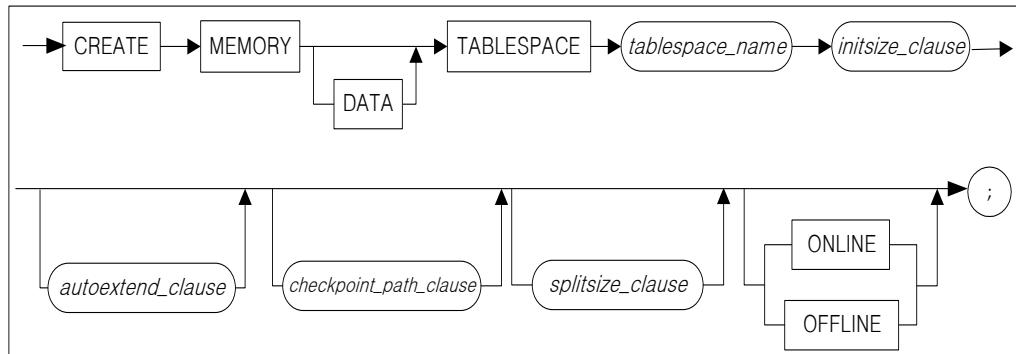
ID	테이블스페이스 종류	저장 공간	테이블스페이스 이름	생성 시점
0	SYSTEM DICTIONARY TABLESPACE	메모리	SYS_TBS_MEMDIC	CREATE DATABASE
1	SYSTEM MEMORY DEFAULT TABLESPACE	메모리	SYS_TBS_MEM_DATA	CREATE DATABASE
2	SYSTEM DISK DEFAULT TABLESPACE	디스크	SYS_TBS_DISK_DATA	CREATE DATABASE
3	SYSTEM UNDO TABLESPACE	디스크	SYS_TBS_DISK_UNDO	CREATE DATABASE
4	SYSTEM DISK TEMPORARY TABLESPACE	디스크	SYS_TBS_DISK_TEMP	CREATE DATABASE
5이상	USER MEMORY DATA TABLESPACE	메모리	사용자 지정	CREATE MEMORY DATA TABLESPACE
5이상	USER DISK DATA TABLESPACE	디스크	사용자 지정	CREATE DISK DATA TABLESPACE
5이상	USER DISK TEMPORARY TABLESPACE	디스크	사용자 지정	CREATE DISK TEMPORARY TABLESPACE
5이상	USER VOLATILE DATA TABLESPACE	메모리	사용자 지정	CREATE VOLATILE DATA TABLESPACE

[표 3-2] 테이블스페이스의 종류

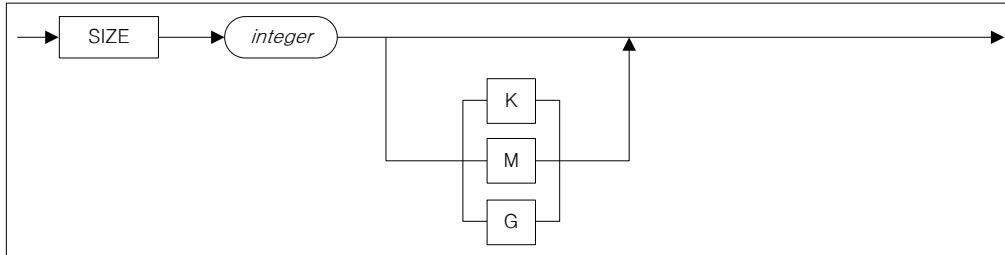
# CREATE MEMORY TABLESPACE

## 구문

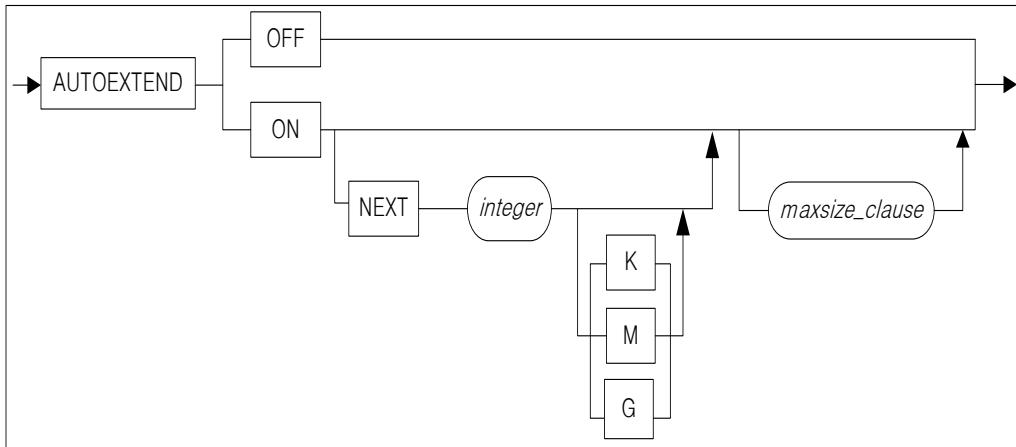
*create\_memory\_tablespace ::=*



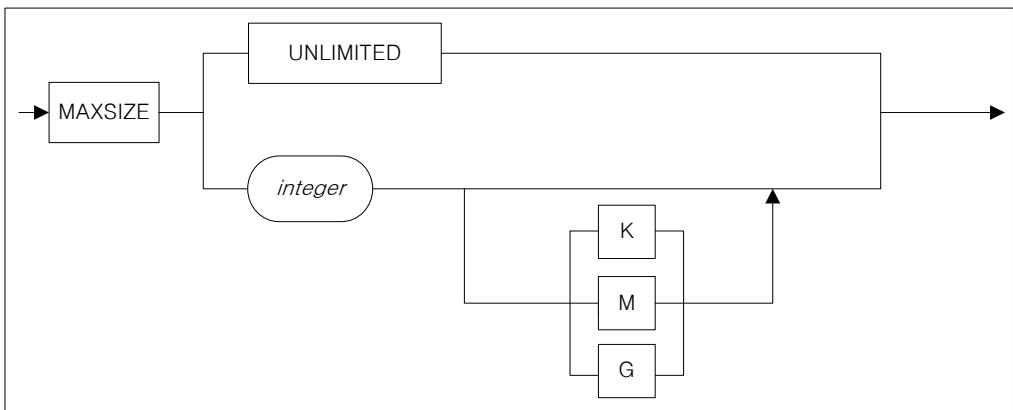
*initsize\_clause ::=*



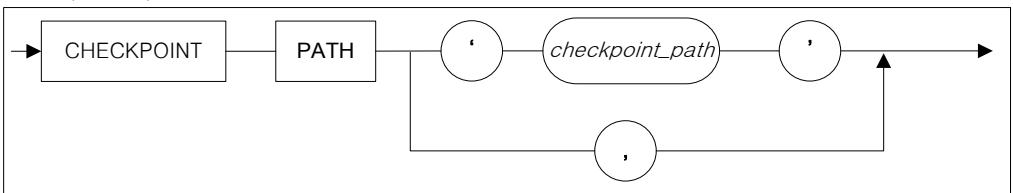
*autoextend\_clause ::=*



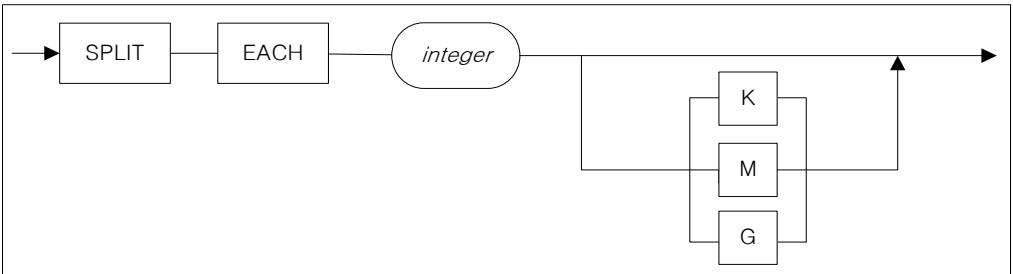
*maxsize\_clause ::=*



*checkpoint\_path\_clause ::=*



*splitsize\_clause ::=*



## 전제 조건

테이블스페이스는 SYS 사용자이거나 CREATE TABLESPACE 시스템 권한을 가진 사용자만 만들수 한다.

## 설명

데이터베이스 내에 데이터베이스 객체를 저장할 수 있는 메모리 테이블스페이스를 생성하는 구문이다.

이 구문으로 생성한 테이블스페이스에 메모리 테이블을 생성할 수 있다.

## MEMORY

메모리 테이블스페이스를 생성한다.

## DATA

사용자의 데이터를 저장할 테이블스페이스를 생성한다. DATA 키워드를 지정하지 않은 채로 CREATE TABLESPACE 구문을 수행하여도 기본적으로 데이터 테이블스페이스를 생성한다.

### *tablespace\_name*

생성될 테이블스페이스의 이름을 명시한다.

### *initsize\_clause*

생성될 테이블스페이스의 초기 크기를 지정한다.

## SIZE

테이블스페이스의 초기 크기를 명시한다. 크기는 Kilobytes(K), Megabytes(M), 또는 Gigabytes(G) 단위로 명시하며, 기본 단위는 K 이다.

메모리 테이블스페이스의 기본 확장 단위의 배수여야 한다. (EXPAND CHUNK PAGE COUNT 프로퍼티에 지정한 페이지 숫자 \* 메모리 테이블스페이스의 페이지 크기 (32KB)) 예를 들어 EXPAND CHUNK PAGE COUNT 를 128 로 지정했다면, 메모리 테이블스페이스의 기본 확장 크기는 128 \* 32KB 로 계산하여 값은 4MB 를 얻는다. 그러므로 SIZE 로 지정할 수 있는 크기는 4MB 의 배수여야 한다.

### *autoextend\_clause*

테이블스페이스의 자동 확장 여부를 명시한다.

## ON

자동 확장이 가능함을 나타낸다.

## OFF

자동 확장 불가능함을 나타내며, 기본값이다.

## NEXT

메모리 테이블스페이스의 기본 확장 단위의 배수여야 한다. (EXPAND CHUNK PAGE COUNT 프로퍼티에 지정한 페이지 숫자 \* 메모리 테이블스페이스의 페이지 크기 (32KB))

자동 확장 시 다음에 확장될 공간의 크기를 명시한다.

AUTOEXTEND 가 ON 이고 이 값을 명시하지 않을 경우 기본값은 EXPAND CHUNK PAGE COUNT 에 지정한 메모리 테이블스페이스의 기본 확장 단위 만큼 확장된다.

AUTOEXTEND 가 OFF 일 때는 값이 0 이다.

Kilobytes(K), Megabytes(M), Gigabytes(G) 단위로 명시하며, 기본 단위는 K 이다.

#### *maxsize\_clause*

자동 확장 시 확장할 수 있는 최대 공간의 크기를 명시한다.

AUTOEXTEND ON 상태이고 이 값을 명시하지 않을 경우 기본값은 UNLIMITED 이다.

AUTOEXTEND OFF 상태일 때 이 값은 0 이다. Kilobytes (K), Megabytes (M), Gigabytes (G) 단위로 크기를 명시할 수 있으며 이 단위를 명시하지 않을 경우 기본 단위는 K 이다.

#### UNLIMITED

자동 확장 크기에 제한을 두지 않을 경우 명시한다.

이 경우 시스템에 존재하는 모든 메모리 테이블스페이스의 크기를 합친 전체 크기가 MEM\_MAX\_DB\_SIZE 프로퍼티에 지정한 크기를 벗어나지 않는 한도 내에서 테이블스페이스가 자동확장된다.

#### *checkpoint\_path\_clause*

메모리 테이블스페이스에 저장된 데이터의 영속성을 보장하기 위해 데이터를 파일로 저장하여야 한다. 이러한 메모리 테이블스페이스의 데이터 파일을 체크포인트 이미지라고 한다.

checkpoint\_path 절은 체크포인트 이미지 파일이 저장될 체크포인트 경로(Path)들을 지정한다.

체크포인트 경로를 지정하지 않은 경우 MEM\_DB\_DIR 의 프로퍼티에 지정한 경로를 기본으로 한다.

#### *checkpoint\_path*

메모리 테이블스페이스의 체크포인트시 체크포인트 이미지가 저장되는 경로이다. 체크포인트 및 테이블스페이스 로딩시 디스크 입출력 비용을 분산할 수 있도록 다수의 경로가 지정될 수 있다.

#### *split\_each\_clause*

메모리 테이블스페이스의 크기가 운영체제에서 제공하는 최대 파일 크기를 초과하거나 입출력 비용을 분산할 경우 작은 파일로 분할된다. 이러한 분할의 크기는 사용자가 지정할 수 있으며, 지정하지 않을 경우 DEFAULT\_MEM\_DB\_FILE\_SIZE 프로퍼티에 지정된 값이 기본으로 사용된다.

Kilobytes(K), Megabytes(M), Gigabytes(G) 단위로 명시하며, 기본 단위는 K 이다.

## 예제

〈질의 1〉 초기 사이즈가 512M이고, 자동 확장하지 않는 사용자 정의 메모리 데이터 테이블스페이스를 생성한다. (체크포인트 이미지는 MEM\_DB\_DIR 프로퍼티에 지정된 경로에 저장된다. 분할될 체크포인트 이미지 파일의 크기는 DEFAULT\_MEM\_DB\_FILE\_SIZE 프로퍼티의 값을 따른다.)

```
iSQL> CREATE MEMORY DATA TABLESPACE user_data SIZE 512M;  
Create success.
```

〈질의 2〉 초기 사이즈가 512M이고, 128M 단위로 자동 확장 가능한<sup>1</sup> 사용자 정의 메모리 데이터 테이블스페이스를 생성한다. (체크포인트 이미지는 MEM\_DB\_DIR 프로퍼티에 지정된 경로에 저장된다. 분할될 체크포인트 이미지 파일의 크기는 DEFAULT\_MEM\_DB\_FILE\_SIZE 프로퍼티의 값을 따른다.)

```
iSQL> CREATE MEMORY DATA TABLESPACE user_data SIZE 512M  
AUTOEXTEND ON NEXT 128M;  
Create success.
```

〈질의 3〉 초기 사이즈가 512M이고, 최대 1G 까지 128M 단위로 자동 확장 가능한 사용자 정의 메모리 데이터 테이블스페이스를 생성한다. (체크포인트 이미지는 다중화를 위해 3 개의 디렉토리에 나누어 저장하고, 분할될 체크포인트 이미지 파일의 크기를 256M로 한다.)

```
iSQL> CREATE MEMORY DATA TABLESPACE user_data SIZE 512M  
AUTOEXTEND ON NEXT 128M MAXSIZE 1G CHECKPOINT PATH '/dbs/path1',  
'/dbs/path2', '/dbs/path3' SPLIT EACH 256M;  
Create success.
```

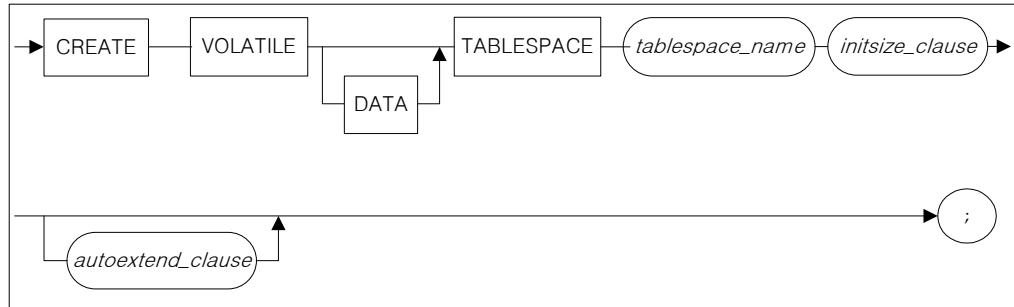
---

<sup>1</sup> 테이블스페이스의 최대 크기인 MAXSIZE절을 지정하지 않았으므로, 기본적으로 UNLIMITTED를 지정한 것과 같다. 이 경우 시스템에 존재하는 모든 메모리 테이블스페이스의 크기 총합이 MEM\_MAX\_DB\_SIZE 프로퍼티에 지정한 메모리 크기를 벗어나지 않는 한도 내에서 테이블스페이스의 확장이 이루어진다.

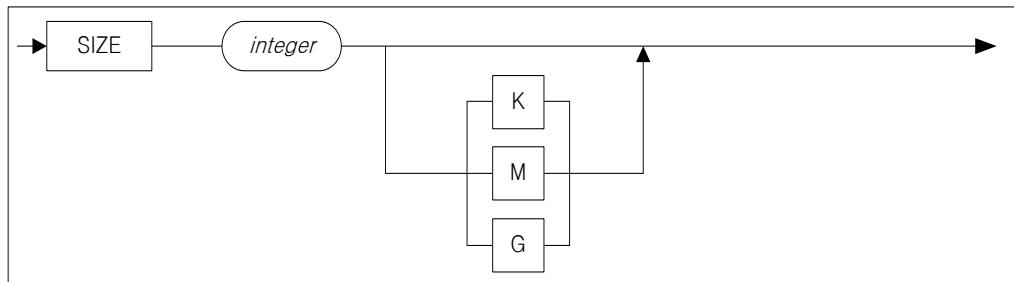
# CREATE VOLATILE TABLESPACE

## 구문

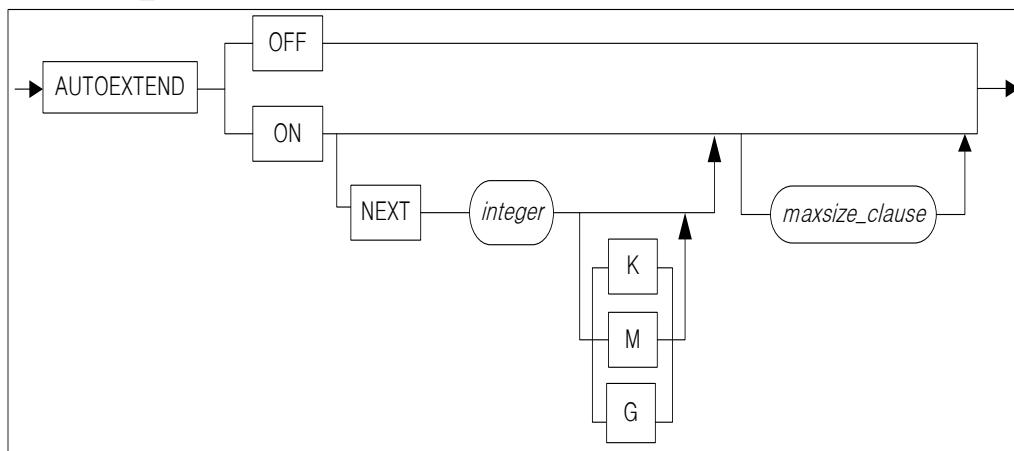
*create\_tablespace ::=*



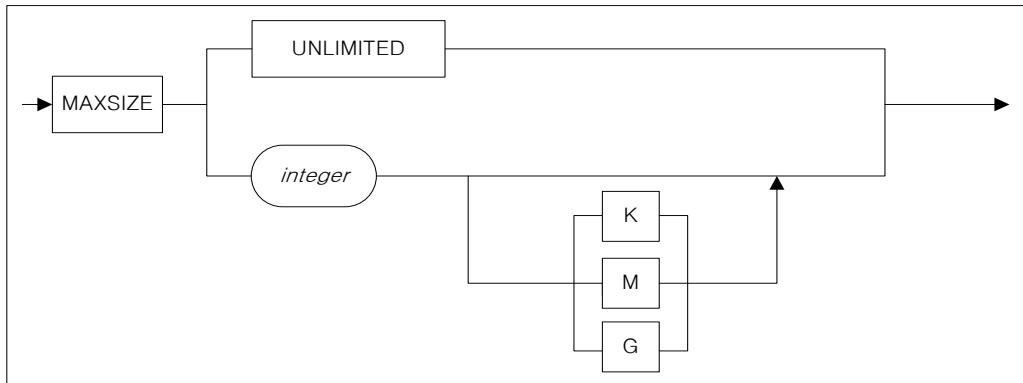
*initsize\_clause ::=*



*autoextend\_clause ::=*



*maxsize\_clause ::=*



## 전제 조건

테이블스페이스는 SYS 사용자이거나 CREATE TABLESPACE 시스템 권한을 가진 사용자만 만들수 한다.

## 설명

데이터베이스 내에 데이터베이스 객체를 저장할 수 있는 휘발성 테이블스페이스를 생성하는 구문이다.

이 구문으로 생성한 테이블스페이스에 휘발성 테이블을 생성할 수 있다.

VOLATILE

휘발성 테이블스페이스를 생성한다.

DATA

사용자의 데이터를 저장할 테이블스페이스를 생성한다. DATA 키워드를 지정하지 않은 채로 CREATE TABLESPACE 구문을 수행하여도 기본적으로 데이터 테이블스페이스를 생성한다.

*tablespace\_name*

생성될 테이블스페이스 이름을 명시한다.

*initsize\_clause*

생성될 테이블스페이스의 초기 크기를 지정한다.

SIZE

테이블스페이스의 초기 크기를 명시한다. 크기는 Kilobytes(K), Megabytes(M), 또는 Gigabytes(G) 단위로 명시하며, 기본 단위는 K 이다.

메모리 테이블스페이스의 기본 확장 단위의 배수여야 한다. (EXPAND CHUNK PAGE COUNT 프로퍼티에 지정한 페이지 숫자 \* 메모리 테이블스페이스의 페이지 크기 (32KB)) 예를 들어 EXPAND CHUNK PAGE COUNT 를 128 로 지정했다면, 메모리 테이블스페이스의 기본 확장 크기는 128 \* 32KB 로 계산하여 값은 4MB 를 얻는다. 그러므로 SIZE 로 지정할 수 있는 크기는 4MB 의 배수여야 한다.

#### *autoextend\_clause*

테이블스페이스의 자동 확장 여부를 명시한다.

ON

자동 확장이 가능함을 나타낸다.

OFF

자동 확장 불가능함을 나타낸다. 기본값이다.

NEXT

메모리 테이블스페이스의 기본 확장 단위의 배수여야 한다. (EXPAND CHUNK PAGE COUNT 프로퍼티에 지정한 페이지갯수 \* 메모리 테이블스페이스의 페이지 크기 (32KB))

자동 확장 시 다음에 확장될 공간의 크기를 명시한다.

AUTOEXTEND 가 ON 이고 이 값을 명시하지 않을 경우 기본값은 EXPAND CHUNK PAGE COUNT 에 지정한 메모리 테이블스페이스의 기본 확장 단위 만큼 확장된다.

AUTOEXTEND OFF 상태일 때 이 값은 0 이다. Kilobytes(K), Megabytes(M), 또는 Gigabytes(G) 단위로 크기를 명시할 수 있으며 이 단위를 명시하지 않을 경우 기본 단위는 K 이다.

#### *maxsize\_clause*

자동 확장 시 확장할 수 있는 최대 공간의 크기를 명시한다.

Kilobytes (K), Megabytes (M), 또는 Gigabytes (G) 단위로 크기를 명시할 수 있으며 이 단위를 명시하지 않을 경우 기본 단위는 K 이다.

AUTOEXTEND ON 상태이고 이 값을 명시하지 않을 경우 기본값은 UNLIMITED 이다. AUTOEXTEND OFF 상태일 때 이 값은 0 이다.

UNLIMITED

자동 확장 가능 크기에 제한을 두고 싶지 않을 경우 명시한다.

이 경우 시스템에 존재하는 모든 메모리 테이블스페이스의 크기를

합친 전체 크기가 MEM\_MAX\_DB\_SIZE 프로퍼티에 지정한 크기를 벗어나지 않는 한도 내에서 테이블스페이스가 자동확장된다.

---

## 예제

〈질의 1〉 초기 사이즈가 512M이고, 자동 확장하지 않는 사용자 정의 휘발성 데이터 테이블스페이스를 생성한다.

```
iSQL> CREATE VOLATILE DATA TABLESPACE user_data SIZE 512M;  
Create success.
```

〈질의 2〉 초기 사이즈가 512M이고, 128M 단위로 자동 확장가능한<sup>2</sup> 사용자 정의 휘발성 데이터 테이블스페이스를 생성한다.

```
iSQL> CREATE VOLATILE DATA TABLESPACE user_data SIZE 512M  
AUTOEXTEND ON NEXT 128M;  
Create success.
```

---

## 주의사항

휘발성 테이블에서는 LOB 타입을 사용할 수 없다.

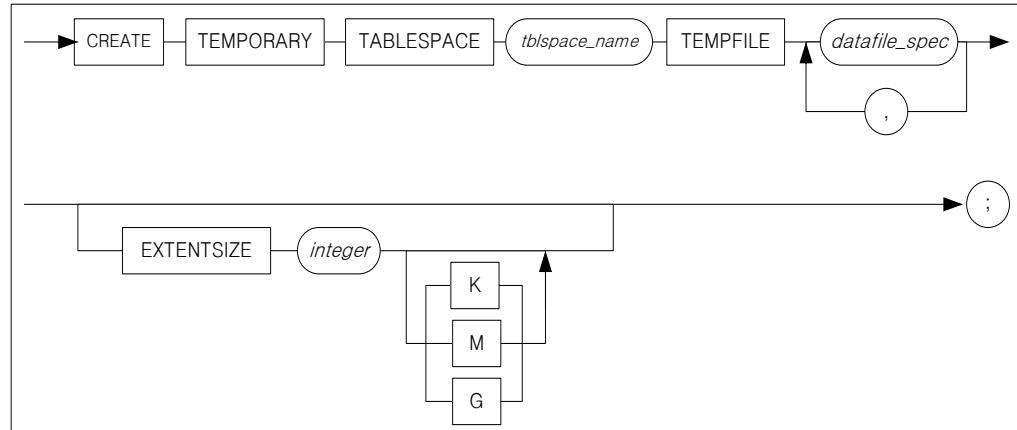
---

<sup>2</sup> 테이블스페이스의 최대 크기인 MAXSIZE절을 지정하지 않았으므로, 기본적으로 UNLIMITED를 지정한 것과 같다. 이 경우 시스템에 존재하는 모든 메모리 테이블스페이스의 크기의 총 합이 MEM\_MAX\_DB\_SIZE 프로퍼티에 지정한 메모리 크기를 벗어나지 않는 한도 내에서 테이블스페이스의 확장이 이루어진다.

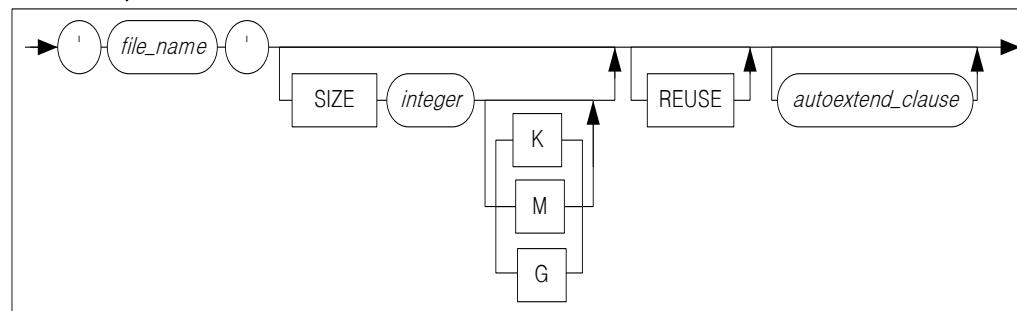
# CREATE TEMPORARY TABLESPACE

## 구문

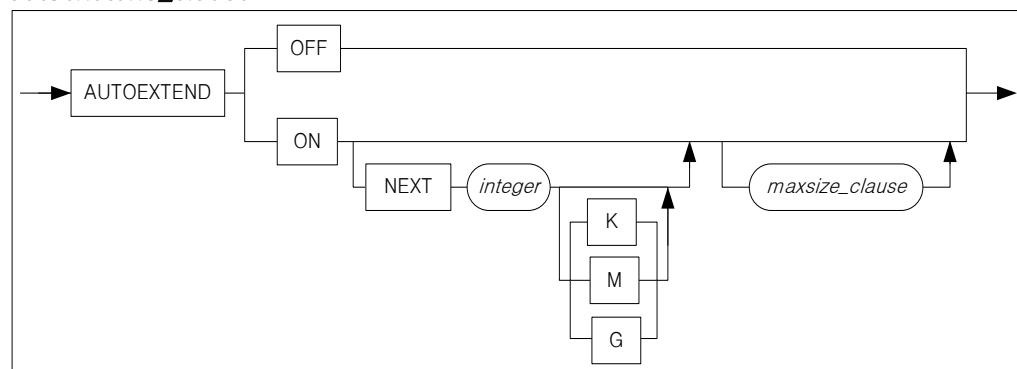
*create\_temporary\_tablespace ::=*



*datafile\_spec ::=*



*autoextend\_clause ::=*



---

## 전제 조건

SYS 사용자이거나 CREATE TABLESPACE 시스템 권한을 가진 사용자여야 한다.

---

## 설명

임의의 세션이 지속되는 동안 데이터베이스 객체를 저장하여 사용할 임시 테이블스페이스를 DISK 공간에 생성하는 구문이다.

데이터베이스 내에 영구적인 데이터베이스 객체를 저장하기 위해서는 CREATE TABLESPACE 문을 사용한다.

tblspace\_name

생성할 임시 테이블스페이스 이름을 명시한다.

TEMPFILE datafile\_space

임시 테이블스페이스를 구성하는 임시 파일을 명시하는 절이다.

---

## 예제

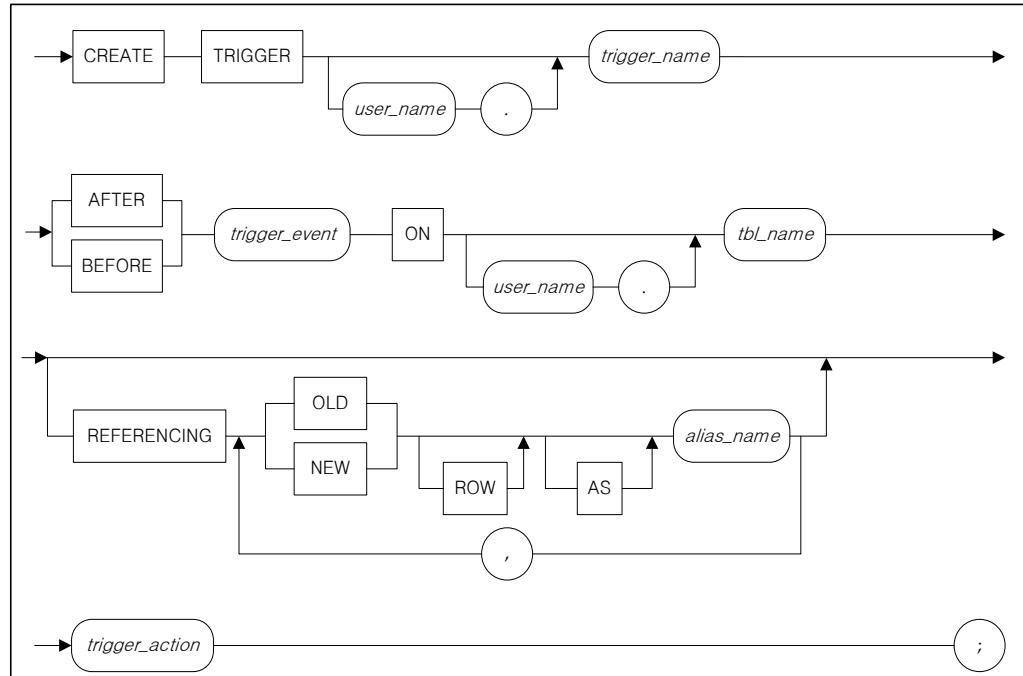
〈질의〉 임시 테이블스페이스를 구성하는 데이터 파일이 tbs.temp 인 5MB 의 temp\_data 테이블스페이스를 생성한다.

```
iSQL> CREATE TEMPORARY TABLESPACE temp_data
      TEMPFILE '/tmp/tbs.temp' SIZE 5M
      AUTOEXTEND ON;
Create success.
```

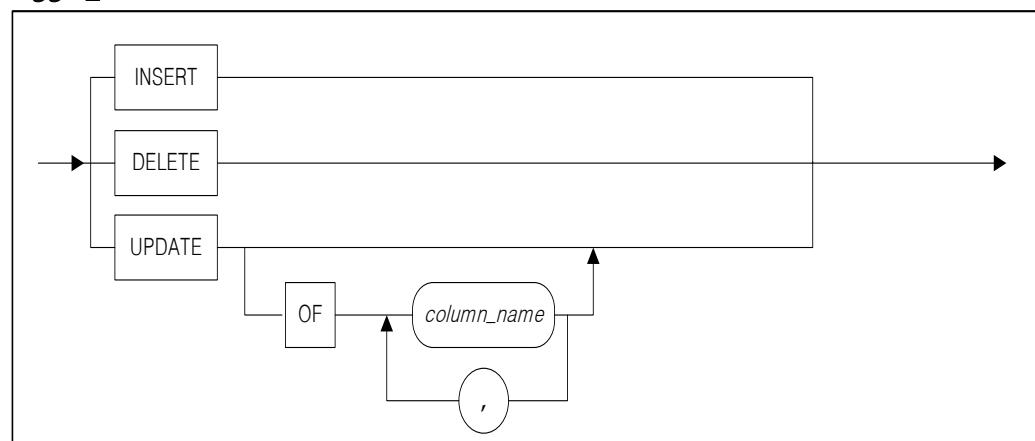
# CREATE TRIGGER

## 구문

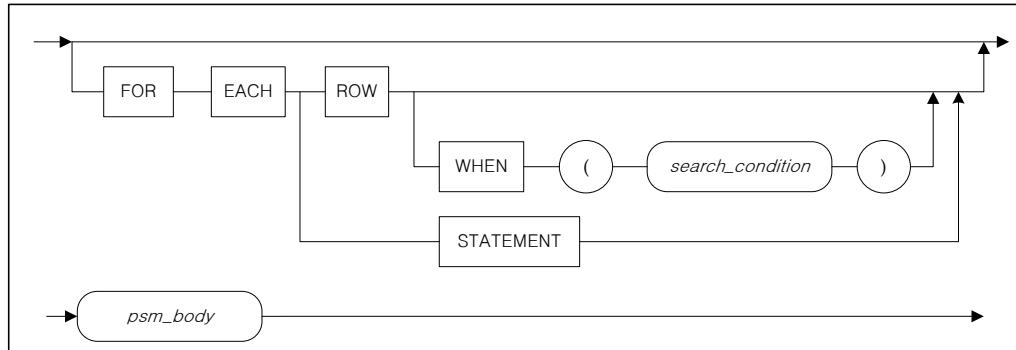
*create\_trigger ::=*



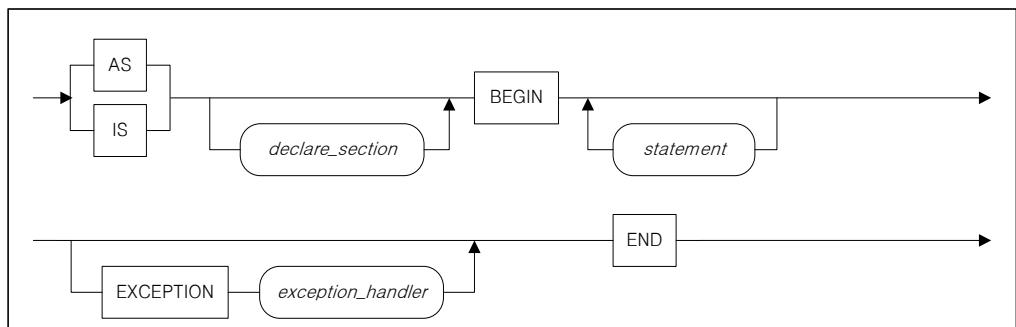
*trigger\_event ::=*



*trigger\_action ::=*



*psm\_body ::=*



---

## 전제 조건

트리거를 생성하기 위해 SYS 사용자이거나 CREATE TRIGGER 시스템 권한을 가져야 한다.

다른 사용자의 테이블에 트리거를 생성하려면 CREATE ANY TRIGGER 권한을 가져야 한다.

---

## 설명

명시된 이름으로 트리거를 생성한다.

*user\_name*

생성될 트리거의 소유자 이름을 명시한다. 생략하면 알티베이스는 현재 사용자가 소유한 테이블에 트리거를 생성한다.

*trigger\_name*

생성될 트리거의 이름을 명시한다. 해당 스키마에서 유일해야 한다.

#### AFTER

트리거 동작이 실행될 시기를 지정한다. *trigger\_event*가 실행된 후에 수행됨을 의미한다.

#### BEFORE

트리거 동작이 실행될 시기를 지정한다. *trigger\_event*가 실행되기 전에 수행됨을 의미한다.

#### *trigger\_event*

트리거를 동작시킬 테이블 데이터의 변형이다. 그러나 데이터베이스의 무결성을 위해 이중화에 의해 적용되는 테이블의 변경은 트리거 이벤트로 처리되지 않는다.

*trigger\_event*로 다음의 세 가지 DML을 지정할 수 있다.

- DELETE: 해당 테이블의 데이터를 삭제하는 DELETE 구문을 수행 시 트리거를 동작시킨다.
- INSERT: 해당 테이블의 데이터를 삽입하는 INSERT 구문을 수행 시 트리거를 동작시킨다.
- UPDATE: 해당 테이블의 데이터를 변경하는 UPDATE 구문을 수행 시 트리거를 동작시킨다. UPDATE 트리거 이벤트에 OF 구문을 사용할 경우 OF 구문에 명시된 컬럼이 변경될 경우에만 트리거를 동작시킨다. BEFORE UPDATE ROW 구문의 트리거는 현재 지원하지 않는다.

#### ON *table\_name*

트리거가 동작하기 위해 참조하는 테이블을 의미한다. 트리거는 *table\_name*에 정의된 테이블의 변화에 따라 반응하게 된다.

다음과 같은 테이블에 대해서는 트리거를 생성할 수 없다.

일반 테이블을 제외한 뷰, 시퀀스, 저장 프로시저와 같은 객체에는 트리거를 생성할 수 없다.

이중화에 포함되어 있는 테이블에는 트리거를 생성할 수 없다. 참고로, 트리거가 존재하는 테이블에 대한 리플리케이션은 생성할 수 있다.

#### REFERENCING 구문

트리거의 특성상 old row 와 new row 의 개념을 갖는다. 즉, 해당 테이블의 데이터가 변경되면, 특정 로우(row)의 변경으로 로우는 이전 값과 이후 값을 갖는다. 이러한 old row 및 new row 를 참조하기 위해 REFERENCING 절을 사용한다.

REFERENCING 구문은 다음과 같은 제약을 갖는다.

- REFERENCING 구문은 FOR EACH ROW 단위(granularity)에서만 사용할 수 있다.
- REFERENCING 구문은 *trigger\_action*에서 참조할 수 있도록 다음과 같은 RERERENCING을 가질 수 있다.

{OLD|OLD ROW|OLD ROW AS|OLD AS} alias\_name

변경되기 이전의 로우(row)를 의미한다. WHEN 구문 또는 트리거의 action body에서 참조할 수 있다. OLD referencing은 INSERT 트리거 이벤트에는 사용할 수 없다.

{NEW|NEW ROW|NEW ROW AS|NEW AS} alias\_name

변경된 후의 로우(row)를 의미하며, BEFORE TRIGGER에서는 변경되기 전의 로우를 가리킨다. DELETE 트리거 이벤트에는 사용할 수 없다. BEFORE TRIGGER에서 각 컬럼의 값을 변경할 수 있다.

*trigger\_action*

트리거 동작은 그 구문상 다음과 같은 세 가지 부분으로 구성된다.

- Action granularity: 트리거 동작의 수행 단위  
(FOR EACH {ROW|STATEMENT})
- Action When condition: 트리거 동작의 수행 조건  
(WHEN search\_condition)
- Action body: 트리거 동작의 내용 (psm\_body)

FOR EACH {ROW|STATEMENT}

트리거의 수행 단위인 동작 단위를 정의한다. 테이블의 데이터에 대한 변경이 발생할 때 트리거를 수행하는 단위가 된다. 기본값은 FOR EACH STATEMENT이며, 다음과 같은 구문을 사용한다.

- FOR EACH ROW: 트리거를 구동시킬 DML 이벤트 발생 시 매 row마다 트리거 action body를 수행하게 된다.  
REFERENCING 구문 또는 WHEN 조건 등을 기술하기 위해서는 반드시 FOR EACH ROW 구문을 사용하여야 한다.
- FOR EACH STATEMENT: 트리거를 구동시킬 DML 이벤트 발생 시 SQL문이 완료되는 시점에 한해 한 번 트리거가 동작하게 된다.

WHEN search\_condition

트리거가 수행될 조건을 의미한다. *search\_condition*이 TRUE인 경우에만 action body가 수행되며, WHEN 조건이 FALSE인 경우에는 action body가 수행되지 않는다. WHEN 조건이 명시되어 있지 않으면, 항상 TRUE로 인식되어 트리거 이벤트 발생 시 트리거 동작이 수행된다.

WHEN 조건을 사용하기 위해서는 다음과 같은 제약을 만족해야 한다.

- WHEN 조건은 반드시 FOR EACH ROW 단위일 경우에만 사용할 수 있다.
- WHEN 조건에는 REFERENCING에 정의된 alias\_name만을 사용할 수 있다.
- WHEN 조건에는 부연질의를 사용할 수 없다.
- WHEN 조건에는 저장 프로시저를 사용할 수 없다.

### *psm\_body*

트리거의 action body 를 의미하며, 트리거가 수행할 구문을 기술한다. 저장 프로시저의 블록 구문과 동일하게 사용할 수 있으며, 다음과 같은 제약을 만족하여야 한다.

트리거의 특성 및 개념 상 action body 를 위한 SQL statement 구문은 다음과 같은 것을 사용할 수 없다.

- Host 변수 및 저장 프로시저 변수를 사용할 수 없다.
- COMMIT, ROLLBACK 등과 같은 트랜잭션문을 사용할 수 없다.
- CONNECT 등과 같은 세션문을 사용할 수 없다.
- CREATE TABLE 등과 같은 스키마문을 사용할 수 없다.
- 저장 프로시저를 호출할 수 없다.
- 주기(cycle)가 발생하는 트리거는 생성할 수 없다.

저장 프로시저의 블록에 대한 자세한 설명은 *Stored Procedure User's Manual* 을 참조하기 바란다.

## 주의 사항

- **트리거의 수행 순서**  
트리거는 하나의 테이블에 대하여 정의하지 않거나, 하나 이상의 트리거를 정의할 수 있다.  
여러 개의 트리거가 정의되어 있을 때 트리거의 수행 순서는 일정하지 않으며, 우선 순위가 중요할 경우에는 여러 개의 트리거를 하나로 통합하여야 한다.
- **트리거의 수행 실패**  
트리거를 수행하던 도중 오류가 발생하면, 해당 트리거를 발생시킨 DML도 실패하게 된다.
- **트리거가 참조하는 테이블에 발생하는 DDL**  
DROP TABLE을 사용하여 데이터베이스에서 트리거 이벤트가 발생하는 테이블을 제거하면 해당 테이블에 대한 모든 트리거도 삭제된다.  
트리거 action body가 참조하는 테이블이 변경되거나 삭제될 경우 트리거는 제거되지 않는다. 이 때, 참조 테이블이 삭제되어 해당 트리거의 action body를 수행할 수 없는 경우

- 이벤트에 해당하는 테이블의 DML은 실패하며, action body를 수행할 수 있는 경우(참조 테이블의 변경)는 트리거를 암시적으로 재 컴파일하여 정상적으로 수행되게 된다.
- **트리거와 이중화**  
이중화로 인해 반영되는 테이블의 변경은 트리거에 영향을 주지 않는다.
  - **트리거와 LOB**  
트리거에서 LOB 타입을 사용할 수 없다. 임의로 사용할 경우, 데이터가 모두 널(NULL)로 보인다.

---

## 예제

〈질의〉 다음은 FOR EACH ROW 트리거되는 동작을 생성하여 트리거 명령문의 결과를 가지고 ono, cno, qty 및 arrival\_date 칼럼의 기존 값을 참조하여 orders 테이블 행에서 배달이 완료된 (processing='D') 주문에 대한 정보가 삭제될 때 log\_tbl에서 행을 작성하여 삭제되는 행에 대해 추적하는 예이다.

```
iSQL> CREATE TABLE orders(
    ono NIBBLE(10),
    cno CHAR(14),
    qty INTEGER,
    arrival_date DATE,
    sysdate DATE);
Create success.
```

```
iSQL> CREATE TABLE log_tbl(
    ono NIBBLE(10),
    cno CHAR(14),
    qty INTEGER,
    arrival_date DATE,
    sysdate DATE);
Create success.
```

```
iSQL> CREATE TRIGGER del_trigger
    AFTER DELETE ON orders
    REFERENCING OLD ROW old_row
    FOR EACH ROW
    AS BEGIN
        INSERT INTO log_tbl VALUES(old_row.ono, old_row.cno, old_row.qty,
        old_row.arrival_date, sysdate);
    END;
/
Create success.
```

```
iSQL> DELETE FROM orders WHERE processing = 'D';
2 rows deleted.
iSQL> SELECT * FROM log_tbl;
```

LOG_TBL.ONO	LOG_TBL.CNO	LOG_TBL.QTY
LOG_TBL.ARRIVAL_DATE		
<hr/>		
LOG_TBL.SYSDATE		
<hr/>		
0011290011	761001-1000001	1000
2005/03/21 10:07:12		2000/12/05 00:00:00
0011290100	700101-1001001	500
2005/03/21 10:07:12		2000/12/07 00:00:00
2 rows selected.		

〈질의〉 다음은 FOR EACH ROW 트리거되는 동작을 생성하여,  
SCORES 테이블에 레코드가 입력되기 되기 전 SCORE 가 없는  
레코드의 SCORE 를 0 점으로 변경한다.

```
iSQL> CREATE TABLE SCORES( ID INTEGER, SCORE INTEGER );
Create success.
iSQL> CREATE TRIGGER SCORES_TRIGGER
BEFORE INSERT ON SCORES
REFERENCING NEW ROW NEW_ROW
FOR EACH ROW
AS BEGIN
    IF NEW_ROW.SCORE IS NULL THEN
        NEW_ROW.SCORE := 0;
    END IF;
END;
/
Create success.
```

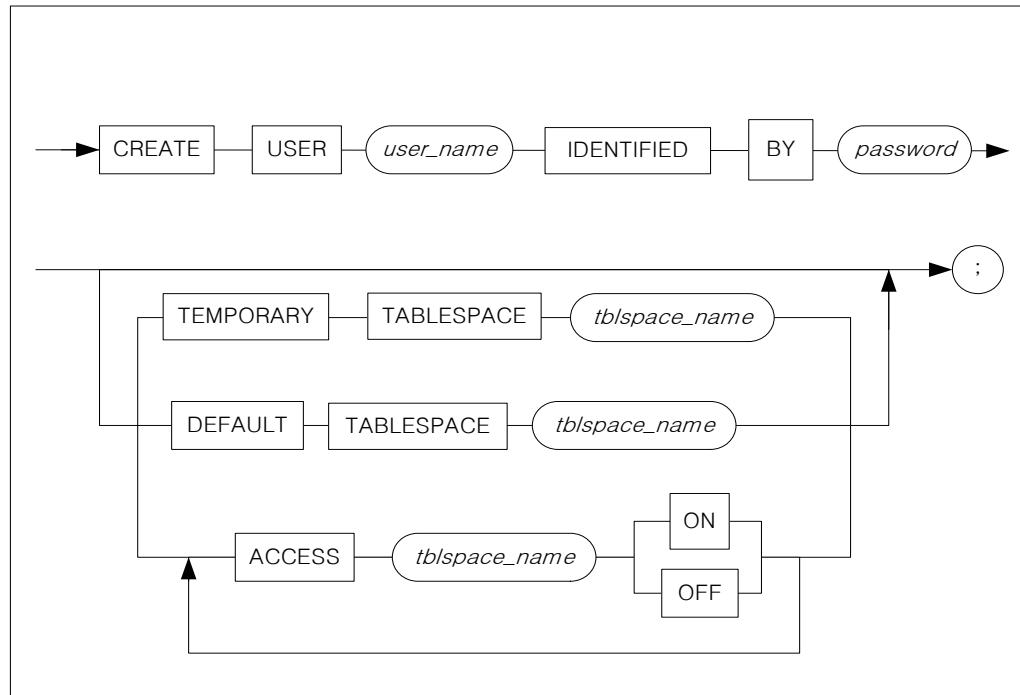
```
iSQL> INSERT INTO SCORES VALUES( 1, 20 );
1 row inserted.
iSQL> INSERT INTO SCORES VALUES( 5, NULL );
1 row inserted.
iSQL> INSERT INTO SCORES VALUES( 17, 75 );
1 row inserted.
```

```
iSQL> SELECT * FROM SCORES;
ID      SCORE
-----
1       20
5       0
17      75
3 rows selected.
```

# CREATE USER

## 구문

*create\_user ::=*



## 전제 조건

사용자를 생성하기 위해 SYS 사용자이거나 CREATE USER 시스템 권한을 가져야 한다.

## 설명

명시된 사용자 명과 암호로 새로운 사용자를 등록한다.

*user\_name*

생성될 사용자 이름을 명시한다. 사용자의 이름은 유일해야 한다.

IDENTIFIED BY *password*

알티베이스가 사용자를 암호로 인증하는 방법으로 사용자는 로그 온 시 암호를 명시해야만 한다.

사용자의 암호의 최대 크기는 운영체제에 따라 다르며 8~40 자 사이이다. Solaris10 과 Window XP 이후의 운영체제는 40 자까지, Solaris 2.8 이후의 운영체제와 Window NT 에서는 11 자, 그 외의 운영체제에서는 8 자까지로 지정할 수 있다. 만일 이보다 더 긴 문자열을 이용하여 사용자를 생성한 경우, 알티베이스는 최대크기 위반 오류를 발생시킨다.

사용자 이름과 암호를 정의하기 위한 규칙은 1 장 알티베이스 SQL 소개에서 SQL 개요 내의 ‘객체 이름 생성 규칙’을 참고한다.

#### TEMPORARY TABLESPACE 절

DISK 내 사용자의 테이블 연산 시 사용될 TEMPORARY TABLESPACE 의 DEFAULT TEMPORARY TABLESPACE 를 지정하는 기능이다.

명시하지 않으면 SYSTEM TEMPORARY TABLESPACE<sup>1</sup> 가 해당 사용자의 DEFAULT TEMPORARY TABLESPACE 로 지정된다. 사용자에게 임시 테이블스페이스를 할당하는데 사용된다.

일반적으로 TEMPORARY TABLESPACE 를 사용할 때는 SQL 문 내 하나 이상의 디스크 기반 테이블을 가지면 DISK TEMPORARY TABLESPACE 를 사용한다.

만약 모든 테이블들이 메모리에 존재하는 테이블이라면 연산 시 사용하는 공간도 모두 메모리이며, 힌트를 사용하지 않는다면 디스크는 사용하지 않는다.

TEMPORARY TABLESPACE 는 한 사용자에 하나만 지정할 수 있다.

#### DEFAULT TABLESPACE 절

사용자가 생성한 객체를 저장하여 사용할 디폴트 테이블스페이스를 명시한다. 이 절을 생략하면 객체들은 메모리에 저장된다.

DEFAULT TABLESPACE 는 한 사용자에 하나만 지정할 수 있다.

#### ACCESS 절

명시한 (*tablespace\_name*) 테이블스페이스의 사용 가능 여부를 명시하는 기능으로, ACCESS ON 으로 지정한 테이블스페이스에 대해서는 사용 권한을 부여 받는다. OFF 로 명시한 테이블스페이스에 대해서는 접근이 불가능하다.

---

<sup>1</sup> SYSTEM TEMPORARY TABLESPACE : 쿼리 수행 중에 발생되는 임시 데이터들을 저장하기 위해서 사용되며, 데이터 복구를 위한 로깅이 수행되지 않는다.

테이블스페이스에 대해 명시적으로 system privilege 를 부여 받지 않은 사용자의 경우 기본적으로 임의의 테이블스페이스에 대해서 접근을 불허한다.

---

## 제한 사항

한 사용자는 여러 DATA TABLESPACE 를 사용할 수 있다. 그러나 TEMPORARY TABLESPACE 는 하나만 사용할 수 있다.

SYSTEM UNDO TABLESPACE 에 대하여 사용자가 구문에 명시하여 접근할 수 없다. 사용자는 UNDO TABLESPACE 내에 테이블이나 인덱스 등을 생성할 수 없다. SYSTEM UNDO TABLESPACE 는 데이터베이스 내에 오직 하나만 존재 하며, 사용자가 생성하거나 삭제할 수 없다.

---

## 예제

〈질의〉 사용자 명이 uare1, 암호가 rose1 인 사용자를 생성하라.

```
iSQL> CREATE USER uare1 IDENTIFIED BY rose1;  
Create success.
```

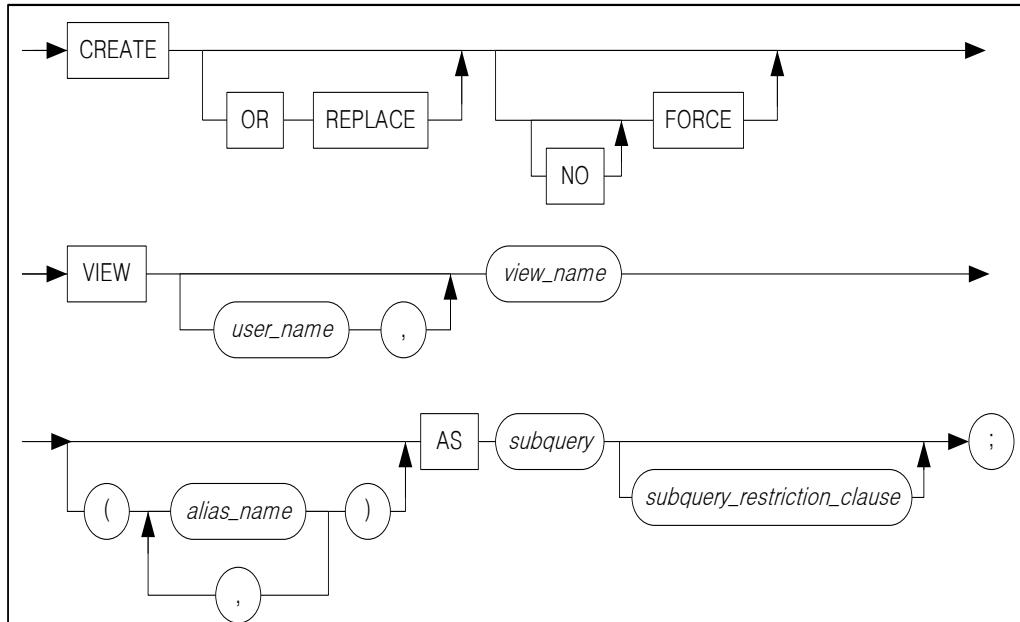
〈질의〉 사용자 이름이 uare4, 암호가 rose4 이며 사용자의 default tablespace 로 user\_data 를 temporary tablespace 로서 temp\_data 테이블스페이스를 사용하며, memory tablespace 인 SYS\_TBS\_MEMORY 에 대해 사용 권한을 가진 사용자를 생성하라.

```
iSQL> CREATE USER uare4  
        IDENTIFIED BY rose4  
        DEFAULT TABLESPACE user_data  
        TEMPORARY TABLESPACE temp_data  
        ACCESS SYS_TBS_MEMORY ON;  
Create success.
```

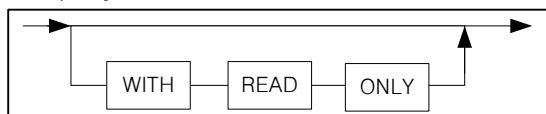
## CREATE VIEW

### 구문

*create\_view ::=*



*subquery\_restriction\_clause ::=*



### 전제 조건

뷰를 생성하기 위해 SYS 사용자 이거나 CREATE VIEW 시스템 권한을 가져야 한다.

다른 사용자의 스키마에 뷰를 생성하려면 CREATE ANY VIEW 권한을 가져야 한다.

뷰를 내포하고 있는 스키마 소유자는 테이블 또는 뷰로 부터 SELECT 하는데 필요한 권한을 가지고 있어야 한다.

## 설명

명시된 뷰<sup>3</sup>의 이름으로 새로운 뷰를 생성한다. 뷔의 이름은 데이터베이스 안에서 유일해야 한다. 즉, 예를 들어, sales 테이블과 sales 뷔가 같은 데이터베이스 안에 존재할 수 없다.

OR REPLACE

이미 생성되어 있는 뷔를 재 생성한다. 즉, 뷔를 제거한 후 재 생성할 필요 없이 기존 뷔의 정의를 대체하는 기능을 수행한다.

FORCE

뷰의 기본 테이블의 존재 여부, 뷔를 내포하고 있는 스키마 소유자의 권한에 관계없이 뷔를 생성한다.

따라서, 의미상 오류를 내포한 무효한 상태의 뷔가 생성될 수 있다. 이런 경우, 뷔에 대해 SELECT 문 수행 시 오류가 발생하므로 FORCE 옵션을 사용해 뷔를 생성한 후에는 뷔를 SELECT 해보거나 메타 테이블을 조회해 뷔의 상태를 확인해야 한다.

NO FORCE

뷰의 기본 테이블이 존재하고 뷔를 내포하고 있는 스키마 소유자가 권한을 가지고 있을 때만 뷔를 생성한다. (기본값)

*user\_name*

생성될 뷔의 소유자 이름을 명시한다. 생략하면 알티베이스는 현재 세션에 연결된 사용자의 스키마에 뷔를 생성한다.

*view\_name*

생성될 뷔의 이름을 명시한다.

*alias\_name*

뷰의 검색 대상이 표현식인 경우 이름(별명)을 명시해야 하며, 그 이름이 뷔의 칼럼 명이 된다. 별명은 유일해야 한다.

*subquery*

기본 테이블의 열과 행을 식별하는 부연질의를 명시한다.

WITH READ ONLY

테이블 또는 뷔가 읽기 전용임을 의미한다. 뷔에 대해서는 변경 연산 (INSERT, UPDATE, 또는 DELETE)을 수행할 수 없으므로 이 옵션을 명시하지 않아도 기본적으로 일기 전용 뷔를 생성한다.

---

<sup>3</sup> 뷔(view)란 실제 데이터는 존재하지 않고, 하나 또는 그 이상의 테이블 또는 뷔를 기반으로 한 논리적 테이블(logical table)을 의미한다. 뷔의 기반이 된 테이블을 기본 테이블(base table) 이라 한다.

## 뷰의 부연질의에 대한 제한 사항

검색 대상에 명시할 수 있는 표현식의 개수는 최대 1024 개이다.  
CURRVAL, NEXTVAL 의사열을 사용할 수 없다.  
LEVEL 의사열을 사용할 경우 반드시 별명을 명시해야 한다.

## 예제

### 뷰 생성하기

〈질의〉 다음 예제는 employee 테이블을 기반으로 한 이름이 avg\_sal 인 뷰를 생성한다. 뷰는 각 부서의 평균 월급을 부서별로 보여준다.

```
iSQL> CREATE VIEW avg_sal AS  
      SELECT DNO, AVG(salary) emp_avg_sal  
      — salary average of each department  
      FROM employee  
      GROUP BY dno;  
Create success.  
iSQL> SELECT * FROM avg_sal;  
AVG_SAL.DNO  AVG_SAL.EMP_AVG_SAL
```

```
A001 2066666.67  
C001 1576666.67  
C002 1660000  
D001 2075750  
F001 1845000
```

6 rows selected.

\* 뷰를 선언하는데 있어서 부연질의가 표현식 AVG(salary)에 대한 열 alias\_name (emp\_avg\_sal)을 사용하고 있기 때문에 뷰의 열 이름을 명시할 필요가 없다.

### 조인 뷰<sup>1</sup> 생성하기

〈질의〉 다음 뷰는 주문된 상품을 담당하고 있는 사원 번호, 사원 이름과 담당 고객의 이름을 보여준다.

```
iSQL> CREATE VIEW emp_cus AS  
      SELECT DISTINCT o.eno, e.ename, c.cname  
      FROM employee e, customer c, orders o  
      WHERE e.eno = o.eno AND o.cno = c.cno;
```

<sup>1</sup> 조인 뷰는 뷰의 부연질의에 조인을 내포하는 것을 의미한다.

Create success.  
iSQL> SELECT \* FROM emp\_cus;

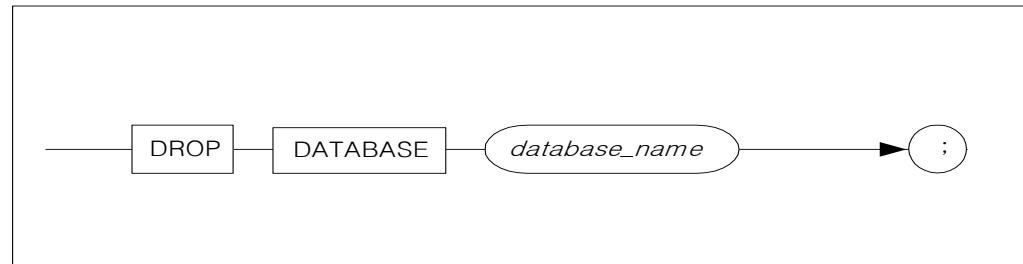
ENO	ENAME	CNAME
12	MYLEE	DJKIM
12	MYLEE	JHKIM
12	MYLEE	YSKIM
12	MYLEE	JHCHOI
12	MYLEE	DHCHO
12	MYLEE	KSKIM
12	MYLEE	BSYOUN
19	KMKIM	LSPARK
19	KMKIM	BSYOUN
19	KMKIM	CHLEE
19	KMKIM	JHPARK
19	KMKIM	SMCHO
19	KMKIM	DKKIM
20	DIKIM	DHKIM
20	DIKIM	DKKIM
20	DIKIM	YDPARK
20	DIKIM	LSPARK
20	DIKIM	DJKIM
20	DIKIM	IJLEE
20	DIKIM	JHKIM
20	DIKIM	YSKIM

21 rows selected.

# DROP DATABASE

## 구문

*drop\_database ::=*



## 전제 조건

데이터베이스 다단계 구동에서 PROCESS 단계에서만 수행할 수 있는 SQL 문으로 SYS 사용자가 -sysdba 관리자 모드에서만 수행할 수 있다.

## 설명

시스템에서 데이터베이스를 삭제하는 명령어이다.

*database\_name*

삭제할 데이터베이스 이름을 명시해야 한다.

이 명령어가 실행되면 해당 데이터베이스가 사용하고 있던 데이터 파일과 로그 파일, 로그 앱크 파일 등이 모두 삭제된다.

## 예제

〈질의〉 mydb라는 이름의 데이터베이스를 삭제하라.

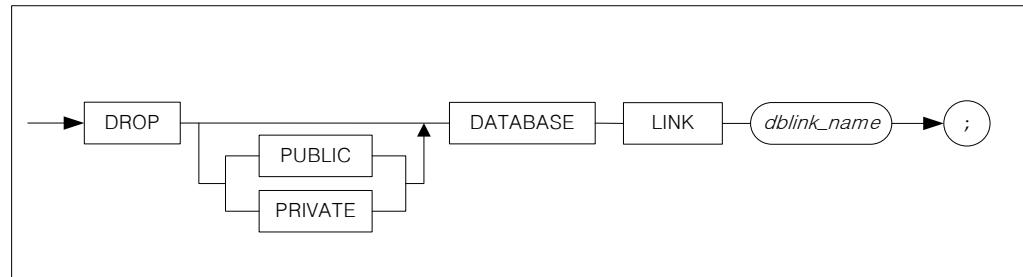
```
iSQL> DROP DATABASE mydb;  
Checking MRDB files  
[Ok] /home/ altibase_home/dbs/mydb-0-0 Exist,  
[Ok] /home/ altibase_home/dbs/mydb-1-0 Exist.  
Checking Log Anchor files
```

```
[Ok] /home /altibase_home/logs/loganchor0 Exist.  
[Ok] /home /altibase_home/logs/loganchor1 Exist.  
[Ok] /home /altibase_home/logs/loganchor2 Exist.  
Removing DRDB files  
Removing Log files  
Removing Log Anchor files  
Drop success.
```

# DROP DATABASE LINK

## 구문

*drop\_database\_link ::=*



## 전제 조건

SYS 사용자이거나 DROP DATABASE LINK 시스템 권한을 가진 사용자만이 데이터베이스 링크를 제거할 수 있다.

## 설명

명시된 이름을 가진 데이터베이스 링크를 제거한다.

*user\_name*

제거할 데이터베이스 링크가 PRIVATE 인 경우에는 생성한 사용자만 접근 가능한 상태이다. 그러므로 이를 제거하고자 하는 경우, 명시적으로 사용자 이름을 기술해야 한다.

*dblink\_name*

제거할 데이터베이스 링크 이름을 명시한다.

## 주의사항

현재 제거하고자 하는 데이터베이스 링크가 사용중인 상태라면, 제거할 수 없다. 데이터베이스 링크를 제거하기 위해서는 제거할 데이터베이스 링크에 대해 수행중인 질의가 없는 상태에서만 제거가 가능하며, 링크에 대해 질의 수행 중인 경우에는 오류가 발생한다.

---

## 예제

〈질의〉 사용자 user1 의 dblink1 이라는 이름으로 생성한 PRIVATE 데이터베이스 링크를 제거한다.

```
iSQL> DROP DATABASE LINK user1.dblink1;
```

# DROP DIRECTORY

## 구문

*drop\_directory ::=*



## 전제 조건

SYS 사용자이거나 DROP ANY DIRECTORY 시스템 권한을 가진 사용자여야 한다.

## 설명

디렉토리를 제거하는 구문으로, 실제 파일 시스템에서 디렉토리를 삭제하지는 않고 데이터베이스에서만 삭제한다.

*directory\_name*

제거할 디렉토리 이름을 명시한다.

## 예제

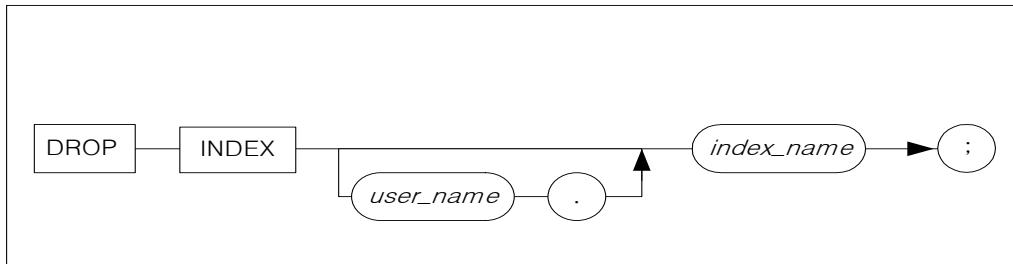
〈질의〉 디렉토리 alti\_dir1 삭제하라.

```
iSQL> DROP DIRECTORY alti_dir1;  
Drop success.
```

## DROP INDEX

### 구문

*drop\_index ::=*



### 전제 조건

SYS 사용자이거나 인덱스가 현재 사용자의 스키마에 속하거나 또는 DROP ANY INDEX 시스템 권한을 가진 사용자여야 한다.

### 설명

인덱스를 제거하는 구문이다.

*user\_name*

제거될 인덱스 소유자 이름을 명시한다. 생략하면 알티베이스는 현재 세션에 연결된 사용자의 스키마에 속한 것으로 간주한다.

*index\_name*

제거할 인덱스 이름을 명시한다.

### 예제

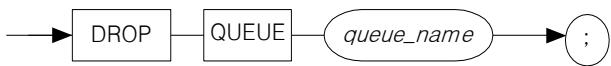
〈질의〉 인덱스 emp\_idx1 을 삭제하라.

```
iSQL> DROP INDEX emp_idx1;  
Drop success.
```

## DROP QUEUE

### 구문

*drop\_queue ::=*



### 설명

지정한 이름의 큐을 삭제한다. 이때 큐 테이블, 큐 테이블의 색인 .  
큐 테이블의 MSGID 를 위해서 생성되었던 SEQUENCE 등이 함께  
삭제된다.

### 예제

〈질의〉 Q1이라는 이름을 가지는 메시지 큐와 부속 오브젝트를 모두  
삭제하라.

DROP QUEUE Q1;

## DROP REPLICATION

### 구문

*drop\_replication ::=*



### 설명

이중화 작업을 제거하는 SQL 문이다.

이중화 개시(ALTER REPLICATION START)가 되어있을 경우  
사용할 수 없고, 이중화 종료(ALTER REPLICATION STOP) 후  
삭제할 수 있다.

*replication\_name*

제거할 리플리케이션 이름을 명시한다.

### 주의 사항

이중화 관련문은 SYS 사용자만 사용 가능하다.

이중화가 실행중일 경우 이중화를 제거할 수 없다.

### 예제

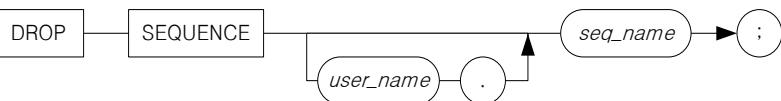
〈질의〉 이중화 rep1 을 삭제하라.

```
iSQL> DROP REPLICATION rep1;  
Drop success.
```

## DROP SEQUENCE

### 구문

*drop\_sequence ::=*



### 전제 조건

시퀀스를 삭제하기 위해 SYS 사용자이거나 시퀀스가 현재 사용자의 스키마이거나 또는 DROP ANY SEQUENCE 시스템 권한을 가져야 한다.

### 설명

명시된 시퀀스를 삭제한다.

*user\_name*

제거될 시퀀스 소유자 이름을 명시한다. 생략하면 알티베이스는 현재 세션에 연결된 사용자의 스키마에 속한 것으로 간주한다.

*seq\_name*

제거할 시퀀스 이름을 명시한다.

### 예제

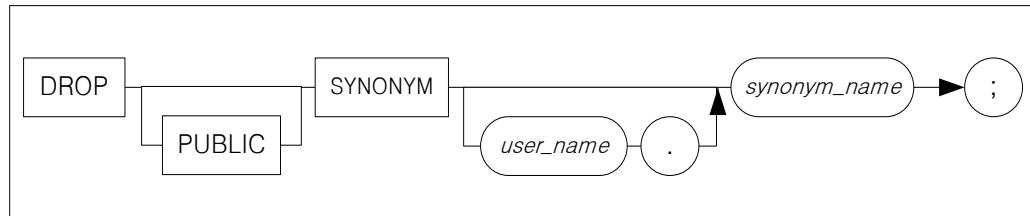
〈질의〉 시퀀스 seq1 을 삭제하라.

```
iSQL> DROP SEQUENCE seq1;  
Drop success.
```

## DROP SYNONYM

### 구문

*drop\_synonym ::=*



### 전제 조건

시노님을 삭제하기 위해 SYS 사용자이거나 시노님이 현재 사용자의 스키마에 속하거나 또는 DROP ANY SYNONYM 시스템 권한을 가져야 한다.

PUBLIC 시노님을 삭제하기 위해서는 DROP PUBLIC SYNONYM 시스템 권한을 가져야 한다.

### 설명

명시된 시노님을 삭제한다.

`PUBLIC`

PUBLIC 시노님을 삭제하기 위해서는 PUBLIC을 명시해야 하며, PUBLIC을 명시하지 않는 경우 명시한 이름의 PRIVATE 시노님을 삭제한다.

PUBLIC을 명시한 경우 `user_name`은 명시할 수 없다.

`user_name`

삭제할 시노님의 소유자 이름을 명시한다. 생략하면 알티베이스는 현재 세션에 연결된 사용자의 스키마에 속한 것으로 간주한다.

`synonym_name`

삭제할 시노님의 이름을 명시한다.

---

## 예제

〈질의〉 my\_dept 시노님을 삭제하라.

```
iSQL> DROP SYNONYM my_dept;  
Drop success.
```

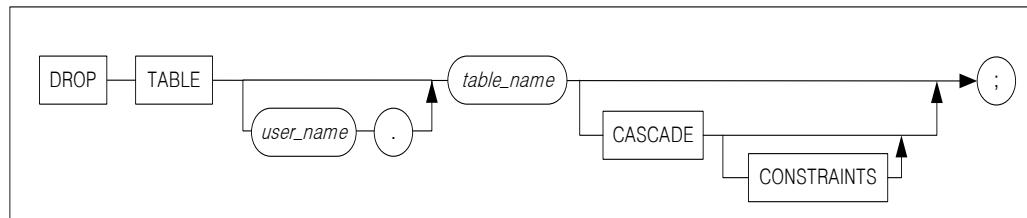
〈질의〉 PUBLIC 시노님인 dept 를 삭제하라.

```
iSQL> DROP PUBLIC SYNONYM dept;  
Drop success.
```

# DROP TABLE

## 구문

*drop\_table ::=*



## 전제 조건

테이블을 제거하기 위해 SYS 사용자이거나 테이블이 현재 사용자의 스키마에 속하거나 또는 DROP ANY TABLE 시스템 권한을 가져야 한다.

## 설명

명시된 테이블을 제거한다.

*user\_name*

제거될 테이블 소유자 이름을 명시한다. 생략하면 알터베이스는 현재 세션에 연결된 사용자의 스키마에 속한 것으로 간주한다.

*table\_name*

제거될 테이블 이름을 명시한다.

{CASCADE | CASCADED CONSTRAINTS}

테이블의 primary key, unique key를 참조하는 다른 테이블들의 referential integrity constraint도 함께 삭제한다

## 예제

〈질의〉 employee 테이블을 삭제하라.

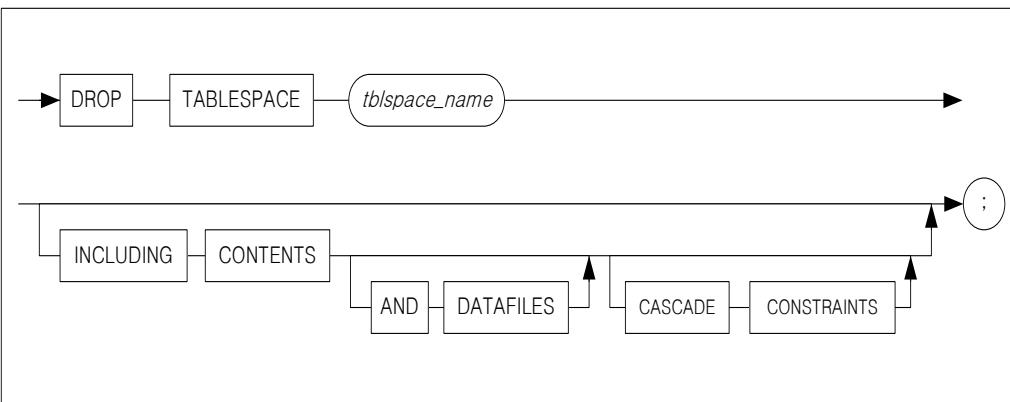
iSQL> `DROP TABLE employee;`

Drop success.

# DROP TABLESPACE

## 구문

*drop\_tablespace ::=*



## 전제 조건

SYS 사용자이거나 `DROP TABLESPACE` 시스템 권한을 가진 사용자여야 한다.

## 설명

데이터베이스에서 테이블스페이스를 제거한다.

*tblspace\_name*

제거할 테이블스페이스를 명시한다.

INCLUDING CONTENTS

테이블스페이스 내의 모든 내용을 삭제한다. 만약 테이블스페이스 내에 하나 이상의 객체가 존재한다면 반드시 이 절을 명시해야 한다. 그렇지 않은 경우 알터베이스는 오류를 발생시키고 `DROP TABLESPACE` 문의 수행은 실패한다.

AND DATAFILES

INCLUDING CONTENTS 절을 명시할 때 AND DATAFILES 절을 함께 명시하면 파일 시스템에서 테이블스페이스와 관련된 모든

파일을 삭제한다.

디스크 테이블스페이스일 경우에는 디스크 테이블스페이스의 모든 데이터 파일을 파일 시스템으로부터 삭제한다.

메모리 테이블스페이스일 경우에는 메모리 테이블스페이스의 모든 체크포인트 이미지 파일들을 파일 시스템으로부터 삭제한다. 단, 체크포인트 경로는 삭제되지 않는다.

또한 휘발성 테이블스페이스에 대해서는 AND DATAFILES 구문을 사용할 수 없다.

## CASCADE CONSTRAINTS

테이블스페이스 내에 존재하는 테이블들의 primary key, unique key 를 참조하는 테이블스페이스 밖에 있는 테이블들의 모든 referential integrity constraint 들도 함께 제거하려면 이 절을 명시해야 한다. 즉, 관련된 referential integrity constraint 가 존재하는 상태에서 이 절을 명시하지 않고 수행하면 알티베이스는 오류를 발생시키고 DROP TABLESPACE 문의 수행은 실패한다.

---

## 제한 사항

시스템 테이블스페이스(SYS\_TBS\_MEMDIC, ,  
SYS\_TBS\_MEMDATA, SYS\_TBS\_TEMP, SYS\_TBS\_UNDO)는  
삭제할 수 없다.

---

## 예제

〈질의 1〉 다음은 테이블스페이스 user\_data 를 제거한다.

```
iSQL> DROP TABLESPACE user_data;  
Drop success.
```

〈질의 2〉 다음은 디스크 테이블스페이스 user\_data 의 모든  
객체(object)와 데이터 파일들과 함께 테이블스페이스도 삭제한다.

```
iSQL> DROP TABLESPACE user_data INCLUDING CONTENTS AND  
DATAFILES;  
Drop success.
```

〈질의 3〉 다음은 메모리 테이블스페이스 user\_data 의 모든  
객체(object)와 데이터 파일들과 함께 테이블스페이스도 삭제한다.

```
iSQL> DROP TABLESPACE user_memory_tbs INCLUDING CONTENTS AND  
DATAFILES;  
Drop success.
```

〈질의 4〉 다음은 테이블스페이스 user\_data 의 모든 객체(object)와

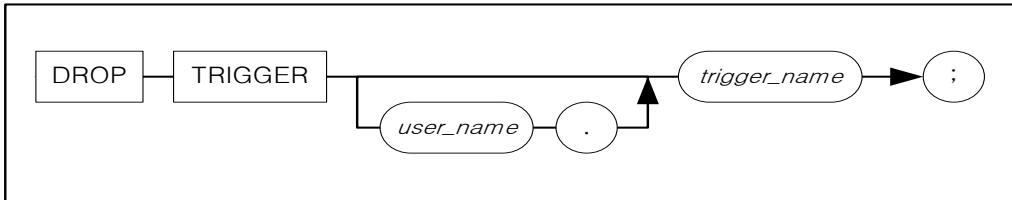
모든 테이블의 Primary Key 또는 Unique Key 를 참조하는 다른 테이블스페이스 테이블들의 모든 referential integrity constraints 들도 함께 user\_data 테이블스페이스를 삭제한다.

```
iSQL> DROP TABLESPACE user_data INCLUDING CONTENTS CASCADE  
CONSTRAINTS;  
Drop success.
```

# DROP TRIGGER

## 구문

*drop\_trigger ::=*



## 전제 조건

트리거를 제거하기 위해 SYS 사용자이거나 트리거가 현재 사용자의 스키마에 속하거나 또는 DROP ANY TRIGGER 시스템 권한을 가진 사용자여야 한다.

## 설명

데이터베이스에서 명시된 트리거를 제거한다.

*user\_name*

제거될 트리거의 소유자 이름을 명시한다. 생략하면 알터베이스는 현재 사용자의 스키마 내에 트리거를 제거한다.

*trigger\_name*

제거될 트리거의 이름을 명시한다.

## 예제

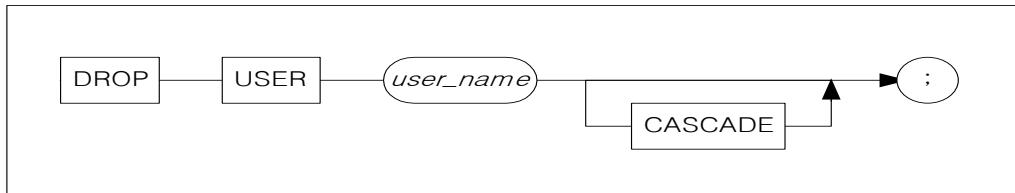
〈질의〉 트리거 *del\_trigger* 을 삭제하라. 트리거 *del\_trigger* 는 CREATE TRIGGER 문의 예제를 참조한다.

```
iSQL> DROP TRIGGER del_trigger;  
Drop success.
```

# DROP USER

## 구문

*drop\_user ::=*



## 전제 조건

사용자를 제거하기 위해 SYS 사용자이거나 DROP USER 시스템 권한을 가진 사용자여야 한다.

## 설명

데이터베이스에서 명시된 사용자를 제거한다.

*user\_name*

제거될 사용자 이름을 명시한다.

CASCADE 기능

데이터베이스 user 뿐만 아니라 그 user의 스키마를 모두 삭제한다. 또한 해당 user의 테이블이 갖는 primary key나 unique key를 참조하는 다른 테이블들의 referential integrity constraint들도 함께 삭제한다.

CASCADE가 생략된 경우 제거 될 사용자가 생성한 테이블 또는 시퀀스를 먼저 제거해야 한다.

## 예제

〈질의〉 사용자 uare1 을 삭제하라.

iSQL> `DROP USER uare1;`

Drop success.

〈질의〉 사용자 uare4 와 그것에 속한 모든 objects 를 삭제하라.

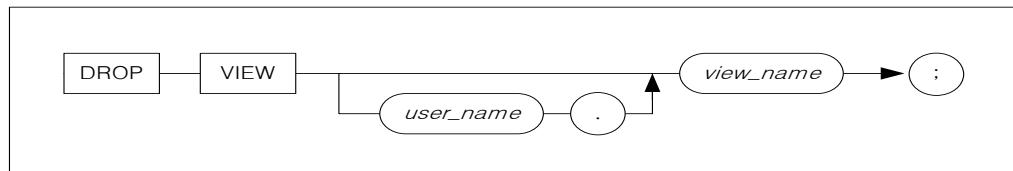
iSQL> DROP USER uare4 CASCADE;

Drop success.

## DROP VIEW

### 구문

*drop\_view ::=*



### 전제 조건

SYS 사용자이거나 뷰가 현재 사용자의 스키마에 속하거나 또는 DROP ANY VIEW 시스템 권한을 가진 사용자만이 뷰를 제거할 수 있다.

### 설명

데이터베이스에서 명시된 뷰를 제거한다.

*user\_name*

제거될 뷰의 소유자 이름을 명시한다. 생략하면 알터베이스는 현재 사용자의 스키마 내에 뷰를 제거한다.

*view\_name*

제거될 뷰의 이름을 명시한다.

### 예제

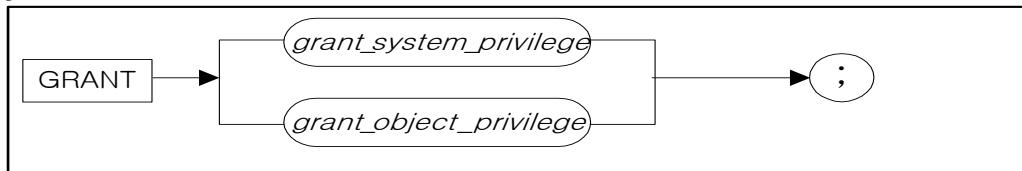
〈질의〉 다음은 뷰 avg\_sal을 제거한다.

```
iSQL> DROP VIEW avg_sal;  
Drop success.
```

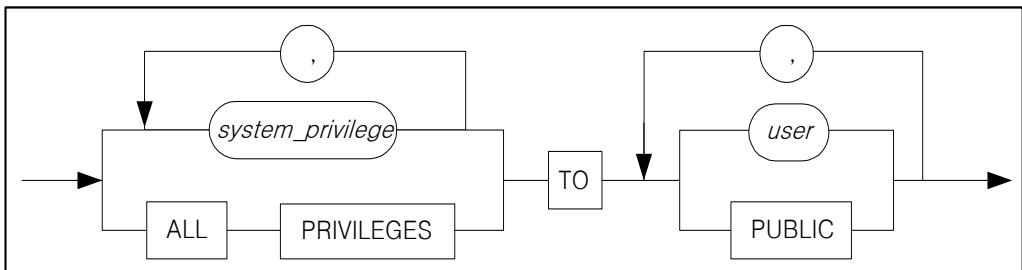
# GRANT

## 구문

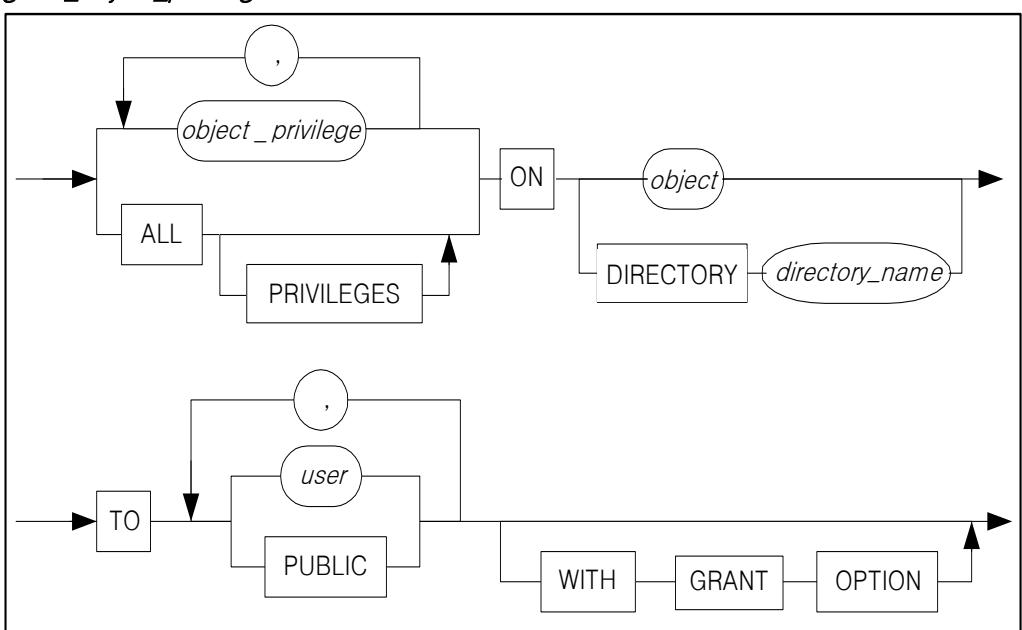
*grant ::=*



*grant\_system\_privilege ::=*



*grant\_object\_privilege ::=*



---

## 전제 조건

SYS 사용자이거나 GRANT ANY PRIVILEGES 시스템 권한이 있어야 시스템 권한을 부여할 수 있다.

SYS 사용자이거나 객체의 소유자이거나 객체의 소유자는 GRANT OPTION 과 더불어 객체 권한이 있어야 객체 권한을 부여할 수 있다.

---

## 설명

명시된 사용자에게 데이터베이스와 객체에 접근 하기 위한 권한들을 부여할 수 있다.

권한은 system privilege 와 object privilege 로 나누어 관리한다.

### *grant\_system\_privilege*

시스템 접근 권한은 일반적으로 DBA 가 관리를 하며, 데이터베이스에 특정한 작업을 수행하거나 모든 스키마에 있는 객체들을 관리할 수 있는 권한이다.

DDL 문, DCL 문을 수행 시 필요한 권한들이다.

### *grant\_object\_privilege*

객체 접근 권한은 객체의 소유자가 관리를 하며, 객체에 접근하고 조작할 수 있는 권한이다.

DML 문을 수행 시 필요한 권한들이다.

---

## 시스템 접근 권한 (System privilege)

시스템 접근 권한은 일반적으로 DBA 가 관리를 하며, 데이터베이스에 특정한 작업을 수행하거나 모든 스키마에 있는 객체들을 관리할 수 있는 권한이다.

### *system\_privilege*

부여될 시스템 접근 권한 (아래 테이블 참고)

ALL PRIVILEGES

시스템 접근 권한에 관한 모든 권한 부여

TO user

사용자 별 시스템 접근 권한 부여

TO PUBLIC

## 모든 사용자에게 시스템 접근 권한 부여

### 주의 사항

- \* SYS 사용자나 GANT ANY PRIVILEGES 권한을 가진 사용자는 모든 시스템 접근 권한을 다른 사용자에게 부여할 수 있다.
- \* SYS 사용자는 모든 시스템 접근 권한을 가진다.
- \* 시스템 접근 중 ANY 키워드는 모든 스키마에서 권한을 가진다. 예를 들어 SELECT ANY TABLE 권한은 데이터베이스 내에 있는 모든 테이블을 SELECT 할 수 있다.
- \* CREATE 권한은 객체를 생성하는 권한이며, 해당 객체를 삭제하는 권한도 포함한다.
- \* CREATE TABLE 권한은 인덱스를 생성하는 권한을 포함하며, 인덱스를 생성하는 권한은 시스템 접근 권한이 아니라 객체 접근 권한이다.
- \* 새로운 사용자를 생성할 때에는 기본적으로 CREATE SESSION, CREATE TABLE, CREATE SEQUENCE, CREATE SYNONYM, CREATE PROCEDURE, CREATE VIEW, CREATE TRIGGER, 그리고 CREATE DATABASE LINK 권한을 함께 부여한다.

```
iSQL> SELECT * FROM SYSTEM$_SYS_PRIVILEGES_ where PRIV_TYPE = 2;
```

알티베이스에서 지원하는 모든 시스템 접근 권한들에 대한 정보를 볼 수 있다.

알티베이스는 다음과 같은 총 51 개의 시스템 접근 권한을 지원한다.

PrivID	System privilege	Name	Purpose
1		ALL	사용불가
201	DATABASE	ALTER SYSTEM	알티베이스 관련 환경 설정을 동적으로 변경할 수 있다.
233		ALTER DATABASE	사용불가
234		DROP DATABASE	사용불가
250	DIRECTORY	CREATE ANY DIRECTORY	저장프로시저의 파일 제어를 위한 디렉토리를 생성한다.
251		DROP ANY DIRECTORY	저장프로시저의 파일 제어를 위한 디렉토리를 삭제한다.
202	INDEXES	CREATE ANY INDEX	Grantee의 스키마 뿐 아니라 다른 사용자의 스키마 내에서도 인덱스 생성이 가능하다 # create table 시 constraint에 의해 임의로

PrivID	System privilege	Name	Purpose
			생성되는 인덱스도 포함된다.
203		ALTER ANY INDEX	DB에 존재하는 모든 인덱스의 정의를 변경할 수 있다.
204		DROP ANY INDEX	DB에 존재하는 모든 인덱스를 삭제할 수 있다.
205	PROCEDURES	CREATE PROCEDURE	Grantee의 스키마 내에 Stored Procedure나 Function을 생성할 수 있다.
206		CREATE ANY PROCEDURE	Grantee의 스키마 뿐 아니라 다른 사용자의 스키마 내에 Stored Procedure나 Function을 생성할 수 있다.
207		ALTER ANY PROCEDURE	DB에 존재하는 모든 Stored Procedure나 Function을 재컴파일 할 수 있다.
208		DROP ANY PROCEDURE	DB에 존재하는 모든 Stored Procedure나 Function을 삭제할 수 있다.
209		EXECUTE ANY PROCEDURE	DB에 존재하는 모든 Stored Procedure나 Function을 실행할 수 있다.
210	SEQUENCES	CREATE SEQUENCE	Grantee의 스키마 내에 시퀀스를 생성할 수 있다.
211		CREATE ANY SEQUENCE	Grantee의 스키마 뿐 아니라 다른 사용자의 스키마 내에서도 시퀀스 생성이 가능하다.
212		ALTER ANY SEQUENCE	DB에 존재하는 모든 시퀀스의 정의를 변경할 수 있다.
213		DROP ANY SEQUENCE	DB에 존재하는 모든 시퀀스를 삭제할 수 있다.
214		SELECT ANY SEQUENCE	DB에 존재하는 모든 시퀀스를 조회할 수 있다.
215	SESSIONS	CREATE SESSION	User가 서버에 연결할 수 있다. 사용자명과 패스워드로 서버에 연결된다.
216		ALTER SESSION	의미 없음

PrivID	System privilege	Name	Purpose
245	SYNONYM	CREATE SYNONYM	자기 소유의 시노님을 생성할 수 있다.
246		CREATE PUBLIC SYNONYM	PUBLIC 시노님을 생성할 수 있다.
247		CREATE ANY SYNONYM	자신의 스키마 뿐 아니라 다른 사용자의 스키마 내에서도 PRIVATE 시노님을 생성할 수 있다.
248		DROP ANY SYNONYM	임의의 PRIVATE 시노님을 삭제할 수 있다.
249		DROP PUBLIC SYNONYM	PUBLIC 시노님을 삭제할 수 있다.
217	TABLES	CREATE TABLE	Grantee의 스키마 내에 테이블을 생성할 수 있다.
218		CREATE ANY TABLE	자신의 스키마 뿐 아니라 다른 사용자의 스키마 내에서도 테이블 생성이 가능하다.
219		ALTER ANY TABLE	DB에 존재하는 모든 테이블의 정의를 변경할 수 있다.
220		DELETE ANY TABLE	DB에 존재하는 모든 테이블의 레코드를 삭제 할 수 있다.
221		DROP ANY TABLE	DB에 존재하는 모든 테이블에 대해서 truncate(모든 레코드 삭제)나 drop(테이블 삭제)을 할 수 있다.
222		INSERT ANY TABLE	DB에 존재하는 모든 테이블에 새로운 레코드를 삽입할 수 있다.
223		LOCK ANY TABLE	DB에 존재하는 모든 테이블에 테이블 잠금을 할 수 있다.
224		SELECT ANY TABLE	DB에 존재하는 모든 테이블의 데이터를 검색 할 수 있다.
225		UPDATE ANY TABLE	DB에 존재하는 모든 테이블에 대해서 열들의 값을 변경할 수 있다.

PrivID	System privilege	Name	Purpose
235	TABLESPACES	CREATE TABLESPACE	테이블스페이스를 생성할 수 있다.
236		ALTER TABLESPACE	테이블스페이스 정의를 변경할 수 있다.
237		DROP TABLESPACE	테이블스페이스를 삭제할 수 있다.
238		MANAGE TABLESPACE	사용불가
241	TRIGGER	CREATE TRIGGER	새로운 트리거를 생성할 수 있다.
242		CREATE ANY TRIGGER	자신의 스키마 뿐 아니라 다른 사용자의 스키마 내에서도 트리거 생성이 가능하다.
243		ALTER ANY TRIGGER	DB에 존재하는 모든 트리거의 정의를 변경할 수 있다.
244		DROP ANY TRIGGER	DB에 존재하는 모든 트리거를 제거할 수 있다.
226	USERS	CREATE USER	새로운 사용자를 생성할 수 있다.
227		ALTER USER	모든 사용자의 암호를 변경할 수 있다.
228		DROP USER	사용자를 제거할 수 있다.
240		SYSDBA	사용불가
229	VIEWS	CREATE VIEW	Grantee의 스키마 내에 뷰를 생성할 수 있다.
230		CREATE ANY VIEW	자신의 스키마 뿐 아니라 다른 사용자의 스키마 내에서도 뷰 생성이 가능하다.
231		DROP ANY VIEW	DB에 존재하는 모든 뷰를 삭제 할 수 있다.
232	MISCELLANEOUS	GRANT ANY PRIVILEGES	모든 system privilege의 권한을 다른 유저에게 부여할 수 있다.

## 객체 접근 권한 (Object privilege)

객체 접근 권한은 객체의 소유자가 관리를 하며, 객체에 접근하고 조작할 수 있는 권한이다.

*object\_privilege*

부여 될 객체 접근 권한 (아래 테이블 참고)

ALL [PRIVILEGES]

객체에 대한 모든 객체 접근 권한을 정의한다.

ON object

해당 객체를 명시. 객체에는 테이블, 시퀀스, 저장 프로시저가 있다.

ON DIRECTORY directory\_name

저장 프로시저에서 제어하는 디렉토리에 대한 READ 또는 WRITE 권한을 부여할 디렉토리 이름을 명시한다.

TO user

해당 객체에 대한 객체 접근 권한을 부여 받는 사용자를 명시한다.

TO PUBLIC

모든 사용자에게 객체 접근 권한을 부여한다.

WITH GRANT OPTION

피수여자가 다른 사용자들(수여자가 생성한 사용자들 또는 피수여자가 생성한 사용자들)에게 객체 접근 권한을 부여할 수 있게 한다.

\* 비교: system privileges 중에서 GRANT ANY PRIVILEGE

### 주의 사항

\* 객체의 소유자란 객체를 생성한 사용자를 말한다.

\* 객체 접근 권한을 부여하기 위해서는 해당 객체의 소유자이거나, 객체 접근 권한을 WITH GRANT OPTION 으로 부여 받은 사용자이어야 한다.

\* 객체의 소유자는 자동적으로 해당 객체의 모든 객체 접근 권한을 가진다.

iSQL> SELECT \* FROM SYSTEM\$\_SYS\_PRIVILEGES\_ where PRIV\_TYPE = 1;  
=> 알티베이스에서 지원하는 모든 객체 접근 권한들에 대한 정보를 볼 수 있다.

알터베이스는 다음과 같은 객체 접근 권한을 지원한다.

PrivID	Object privileges	Table	Sequence	PSM	View	directory
101	ALTER	O	O			
102	DELETE	O				
103	EXECUTE			O		
104	INDEX	O				
105	INSERT	O				
106	REFERENCES	O				
107	SELECT	O	O		O	
108	UPDATE	O				
109	READ					O
110	WRITE					O

메타 테이블은 모든 사용자가 SELECT 권한을 가지며, ANY 키워드가 있는 객체 접근 권한을 가진 사용자가 메타 테이블에 대해 DML을 수행할 수 있다.

---

## 예제

### 시스템 접근 권한

〈질의 1〉 다음은 사용자 user5에게 EXECUTE ANY PROCEDURE, SELECT ANY TABLE, ALTER ANY SEQUENCE, INSERT ANY TABLE, SELECT ANY SEQUENCE 등의 시스템 접근 권한을 부여하여 질의를 수행한 결과이다.

```
iSQL> CREATE TABLE seqtbl(i1 INTEGER);
Create success.
iSQL> CREATE OR REPLACE PROCEDURE proc1
AS
BEGIN
  FOR i IN 1 .. 10 LOOP
    INSERT INTO seqtbl VALUES(i);
  END LOOP;
END;
/
Create success.
iSQL> CREATE USER uare5 IDENTIFIED BY rose5;
Create success.
iSQL> GRANT EXECUTE ANY PROCEDURE, SELECT ANY TABLE TO uare5;
Grant success.
iSQL> CONNECT uare5/rose5;
Connect success.
iSQL> EXEC sys.proc1;
Execute success.
iSQL> SELECT * FROM sys.seqtbl;
```

SEQTBL,I1

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
10 rows selected.  
iSQL> CONNECT sys/manager;  
Connect success.  
iSQL> CREATE SEQUENCE seq1  
    START WITH 13  
    INCREMENT BY 3  
    MINVALUE 0 NOMAXVALUE;  
Create success.  
iSQL> INSERT INTO seqtbl VALUES(seq1.NEXTVAL);  
1 row inserted.  
iSQL> INSERT INTO seqtbl VALUES(seq1.NEXTVAL);  
1 row inserted.  
iSQL> SELECT * FROM seqtbl;
```

SEQTBL,I1

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
13  
16  
12 rows selected.  
iSQL> GRANT ALTER ANY SEQUENCE, INSERT ANY TABLE, SELECT ANY  
SEQUENCE TO uare5;  
Grant success.  
iSQL> CONNECT uare5/rose5;  
Connect success.  
iSQL> ALTER SEQUENCE sys.seq1  
    INCREMENT BY 50  
    MAXVALUE 100  
    CYCLE;  
Alter success.  
iSQL> INSERT INTO sys.seqtbl VALUES(sys.seq1.NEXTVAL);  
1 row inserted.
```

```
iSQL> INSERT INTO sys.seqtbl VALUES(sys.seq1.NEXTVAL);
1 row inserted.
iSQL> INSERT INTO sys.seqtbl VALUES(sys.seq1.NEXTVAL);
1 row inserted.
iSQL> INSERT INTO sys.seqtbl VALUES(sys.seq1.NEXTVAL);
1 row inserted.
iSQL> SELECT * FROM sys.seqtbl;
SEQTBL.I1
-----
1
2
3
4
5
6
7
8
9
10
13
16
66
0
50
100
16 rows selected.
```

## 객체 접근 권한

〈질의 1〉 피수여자 uare6 가 WITH GRANT OPTION 으로  
객체(employee 테이블)의 SELECT 와 DELETE 권한을 부여 받은  
후 같은 권한을 부여 받은 다른 사용자 uare7 이 질의를 수행한  
결과이다.

```
iSQL> CREATE USER uare6 IDENTIFIED BY rose6;
Create success.
iSQL> GRANT CREATE USER TO uare6;
Grant success.
iSQL> @schema.sql
iSQL> GRANT SELECT, DELETE ON employee TO uare6 WITH GRANT
OPTION;
Grant success.
iSQL> CONNECT uare6/rose6;
Connect success.
iSQL> CREATE USER uare7 IDENTIFIED BY rose7;
Create success.
iSQL> GRANT SELECT, DELETE ON sys.employee TO uare7;
Grant success.
iSQL> CONNECT uare7/rose7;
Connect success.
iSQL> DELETE FROM SYS.employee WHERE eno = 12;
```

```
1 row deleted.  
iSQL> SELECT eno, ename FROM sys.employee WHERE eno = 12;  
ENO      ENAME  
-----  
No rows selected.  
iSQL> CONNECT sys/manager;  
Connect success.  
iSQL> CREATE USER uare8 IDENTIFIED BY rose8;  
Create success.  
iSQL> CONNECT uare6/rose6;  
Connect success.  
iSQL> GRANT SELECT, DELETE ON sys.employee TO uare8;  
Grant success.
```

〈= WITH GRANT OPTION 으로 생성된 피수여자(uare6)는 피수여자가 생성한 사용자(uare7) 뿐만 아니라 수여자(SYS)가 생성한 사용자(uare8)에게도

객체 접근 권한을 부여할 수 있다.

```
iSQL> CONNECT uare8/rose8;  
Connect success.  
iSQL> DELETE FROM sys.employee WHERE eno = 13;  
1 row deleted.
```

```
iSQL> SELECT eno, ename FROM sys.employee WHERE eno = 13;  
ENO      ENAME  
-----
```

No rows selected.

〈복합 질의〉 다음은 임의의 사용자에게 임의의 시스템 접근 권한, 객체 접근 권한을 부여한 후 각각의 권한을 해제하는 질의를 수행한 결과이다.

```
iSQL> CONNECT sys/manager;  
Connect success.  
iSQL> CREATE TABLE book(  
    isbn CHAR(10) PRIMARY KEY,  
    title VARCHAR(50),  
    author VARCHAR(30),  
    edition INTEGER DEFAULT 1,  
    publishingyear INTEGER,  
    price NUMBER(10,2),  
    pubcode CHAR(4));  
Create success.  
iSQL> CREATE TABLE inventory(  
    subscriptionid CHAR(10) PRIMARY KEY,  
    storecode CHAR(4),  
    purchasedate DATE,  
    quantity INTEGER,  
    paid CHAR(1));  
Create success.
```

```
iSQL> CREATE USER uare9 IDENTIFIED BY rose9;  
Create success.  
iSQL> GRANT ALL PRIVILEGES TO uare9;  
Grant success.
```

<= sys 가 uare9에게 모든 시스템 접근 권한을 부여한다.  
iSQL> GRANT REFERENCES ON book TO uare9 WITH GRANT OPTION;  
Grant success.  
<= uare9은 sys로부터 객체 book에 대해 REFERENCES 권한을  
부여 받을 뿐 아니라 WITH GRANT OPTION 으로 uare9은 다른  
사용자(uare10)에게 객체 book에 대해 REFERENCES 객체 권한을 부여할 수  
있다.

iSQL> CONNECT uare9/rose9;  
Connect success.

iSQL> INSERT INTO sys.book VALUES ('0070521824', 'Software Engineering',  
'Roger S. Pressman', 4, 1982, 100000, 'CHAU');  
1 row inserted.  
<= uare9이 sys의 객체인 book 테이블에 데이터를 입력한다.

iSQL> INSERT INTO sys.book VALUES ('0137378424', 'Database Processing',  
'David M. Kroenke', 6, 1972, 80000, 'PREN');  
1 row inserted.

iSQL> INSERT INTO sys.inventory VALUES('BORD000002', 'BORD', '12-Jun-  
2003', 6, 'N');  
<= uare9이 sys의 객체인 inventory 테이블에 데이터를 입력한다.

iSQL> INSERT INTO sys.inventory VALUES('MICR000001', 'MICR', '07-Jun-  
2003', 7, 'N');  
1 row inserted.

iSQL> SELECT \* FROM sys.book;  
<= uare9이 sys의 객체인 book 테이블을 검색한다.  
BOOK.ISBN BOOK.TITLE

---

BOOK.AUTHOR	BOOK.EDITION	BOOK.PUBLISHINGYEAR
BOOK.PRICE		

---

BOOK.PUBCODE

---

0070521824	Software Engineering		
Roger S. Pressman	4	1982	100000
CHAU			
0137378424	Database Processing		
David M. Kroenke	6	1972	80000
PREN			

2 rows selected.

iSQL> SELECT \* FROM sys.inventory;  
<= uare9이 sys의 객체인 inventory 테이블을 검색한다.  
INVENTORY.SUBSCRIPTIONID INVENTORY.STORECODE  
INVENTORY.PURCHASEDATE

---

INVENTORY.QUANTITY	INVENTORY.PAID
--------------------	----------------

---

```
BORD000002 BORD 2003/06/12 00:00:00
6          N
MICR000001 MICR 2003/06/07 00:00:00
7          N
2 rows selected.
```

```
iSQL> CREATE TABLE book(
    isbn CHAR(10) PRIMARY KEY,
    title VARCHAR(50),
    author VARCHAR(30),
    edition INTEGER DEFAULT 1,
    publishingyear INTEGER,
    price NUMBER(10,2),
    pubcode CHAR(4));
Create success.
```

```
iSQL> CREATE TABLE inventory(
    subscriptionid CHAR(10) PRIMARY KEY,
    isbn CHAR(10) CONSTRAINT fk_isbn REFERENCES book(isbn),
    storecode CHAR(4),
    purchasedate DATE,
    quantity INTEGER,
    paid CHAR(1));
Create success.
```

```
iSQL> CREATE USER uare10 IDENTIFIED BY rose10;
Create success.
<= uare9 은 sys 로부터 ALL PRIVILEGES 를 부여 받았으므로 다른 사용자를
생성할 수 있다.
```

```
iSQL> GRANT REFERENCES ON sys.book TO uare10;
Grant success.
<= uare9 이 sys 에게 받은 REFERENCES ... WITH GRANT OPTION 으로
uare10 에게 객체 sys.book 에게 REFERENCES 권한을 부여한다.
```

```
iSQL> GRANT ALTER ANY TABLE, INSERT ANY TABLE, SELECT ANY TABLE,
    DELETE ANY TABLE TO uare10;
Grant success.
<= GRANT ANY PRIVILEGES 를 부여 받은 uare9 이 다른 사용자(uare10)에게
시스템 권한을 부여한다.
```

```
iSQL> CONNECT uare10/rose10;
Connect success.
```

```
iSQL> ALTER TABLE sys.inventory
    ADD COLUMN (isbn CHAR(10) CONSTRAINT fk_isbn REFERENCES
    sys.book(isbn));
Alter success.
<= uare10 이 sys 의 객체 book 에 CONSTARINT CONDITION 을 부여한다. (by
ALTER ANY TABLE 과 REFERENCE 권한)
```

```
iSQL> INSERT INTO uare9.book VALUES('0471316156', 'JAVA and CORBA',
'Robert Orfali', 2, 1998, 50000, 'PREN');
```

1 row inserted.

<= uare10 이 uare9 의 객체 book 에 데이터를 입력한다. (by INSERT ANY TABLE 권한)

1 row inserted.

```
iSQL> INSERT INTO uare9.inventory VALUES('TOWE000001', '0471316156',
'TOWE', '01-Jun-2003', 5, 'N');
```

1 row inserted.

```
iSQL> INSERT INTO sys.book VALUES('053494566X', 'Working Classes',
'Robert Orfali', 1, 1999, 80000, 'WILE');
```

1 row inserted.

<= uare10 이 sys 의 객체 book 에 데이터를 입력한다. (by INSERT ANY TABLE 권한)

```
iSQL> INSERT INTO sys.inventory VALUES('MICR000005', 'WILE', '28-JUN-
1999', 8, 'N', '053494566X');
```

1 row inserted.

<= uare10 이 sys 의 객체 inventory 에 데이터를 입력한다. (by INSERT ANY TABLE 권한)

```
iSQL> SELECT * FROM uare9.book;
```

<= uare10 이 uare9 의 객체 book 의 데이터를 검색한다. (by SELECT ANY TABLE 권한)

BOOK.ISBN BOOK.TITLE

---

BOOK.AUTHOR	BOOK.EDITION	BOOK.PUBLISHINGYEAR
BOOK.PRICE		

---

BOOK.PUBCODE

---

0471316156 JAVA and CORBA

Robert Orfali 2 1998 50000

PREN

1 row selected.

```
iSQL> SELECT * FROM uare9.inventory;
```

<= uare10 의 SELECT ANY TALBE 권한으로 검색

INVENTORY.SUBSCRIPTIONID	INVENTORY.ISBN	INVENTORY.STORECODE
--------------------------	----------------	---------------------

---

INVENTORY.PURCHASEDATE	INVENTORY.QUANTITY	INVENTORY.PAID
------------------------	--------------------	----------------

---

TOWE000001 0471316156 TOWE

2003/06/01 00:00:00 5 N

1 row selected.

```
iSQL> SELECT * FROM sys.book;
```

<= uare10 의 SELECT ANY TALBE 권한으로 검색

BOOK.ISBN BOOK.TITLE

---

```
BOOK.AUTHOR          BOOK.EDITION BOOK.PUBLISHINGYEAR
BOOK.PRICE
```

---

```
BOOK.PUBCODE
```

---

```
0070521824 Software Engineering
Roger S. Pressman      4        1982      100000
CHAU
0137378424 Database Processing
David M. Kroenke       6        1972      80000
PREN
053494566X Working Classes
Robert Orfali         1        1999      80000
WILE
3 rows selected.
```

```
iSQL> SELECT * FROM sys.inventory;
INVENTORY.SUBSCRIPTIONID  INVENTORY.STORECODE
INVENTORY.PURCHASEDATE
```

---

```
INVENTORY.QUANTITY INVENTORY.PAID  INVENTORY.ISBN
```

---

```
BORD000002 BORD 2003/06/12 00:00:00
6           N
MICR000001 MICR 2003/06/07 00:00:00
7           N
MICR000005 WILE 1999/06/28 00:00:00
8           N 053494566X
3 rows selected.
```

```
iSQL> DELETE FROM uare9.inventory WHERE subscriptionid = 'TOWE000001';
1 row deleted.
```

```
iSQL> SELECT * FROM uare9.inventory;
INVENTORY.SUBSCRIPTIONID  INVENTORY.ISBN  INVENTORY.STORECODE
```

---

```
INVENTORY.PURCHASEDATE INVENTORY.QUANTITY INVENTORY.PAID
```

---

```
No rows selected.
```

```
iSQL> DELETE FROM sys.inventory WHERE subscriptionid = 'MICR000005';
1 row deleted.
```

```
iSQL> SELECT * FROM sys.inventory;
INVENTORY.SUBSCRIPTIONID  INVENTORY.STORECODE
INVENTORY.PURCHASEDATE
```

---

```
INVENTORY.QUANTITY INVENTORY.PAID  INVENTORY.ISBN
```

---

```
BORD000002 BORD 2003/06/12 00:00:00
6           N
MICR000001 MICR 2003/06/07 00:00:00
```

```
7          N  
2 rows selected.
```

```
iSQL> CONNECT uare9/rose9;  
Connect success.
```

```
iSQL> REVOKE ALTER ANY TABLE, INSERT ANY TABLE, SELECT ANY TABLE,  
DELETE ANY TABLE FROM uare10;  
Revoke success.
```

<= REVOKE ALL FROM uare10;을 대신 실행해도 현재 uare9에게서  
수여 받은 시스템 권한이 모두 해제된다.

```
iSQL> REVOKE REFERENCES ON sys.book FROM uare10 CASCADE  
CONSTRAINTS;  
Revoke success.
```

<= uare10의 REFERENCES 권한과 관련된 REFERENTIAL  
INTEGRITY CONSTRAINTS도 같이 해제한다.

```
iSQL> CONNECT sys/manager;  
Connect success.
```

```
iSQL> REVOKE ALL PRIVILEGES FROM uare9;  
Revoke success.
```

<= uare9의 모든 시스템 권한을 해제한다.

```
iSQL> REVOKE GRANT ANY PRIVILEGES FROM uare9;  
Revoke success.
```

<= uare9의 GRANT ANY PRIVILEGES를 해제한다.

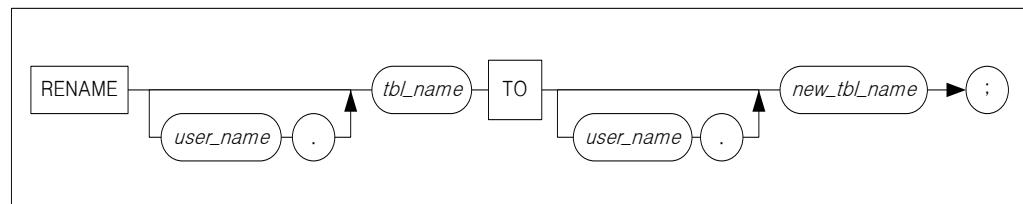
```
iSQL> REVOKE REFERENCES ON book FROM uare9;  
Revoke success.
```

<= uare9의 REFERENCES 권한을 해제한다.

## RENAME TABLE

### 구문

*rename ::=*



### 전제 조건

SYS 사용자이거나 테이블이 현재 사용자의 스키마이거나 또는 ALTER ANY TABLE 시스템 권한을 가져야 한다.

### 설명

명시된 객체(테이블, 시퀀스, 또는 뷰)의 이름을 새로운 이름으로 변경한다. 객체 이름만 변경되고 저장된 데이터는 유지된다.

*user\_name*

이름이 변경될 테이블 소유자 이름을 명시한다. 생략하면 알터베이스는 현재 세션에 연결된 사용자의 스키마에 속한 것으로 간주한다.

*old\_name*

기존 테이블, 시퀀스, 또는 뷰의 이름을 명시한다.

*new\_name*

기존 테이블, 시퀀스, 또는 뷔의 이름에 주어질 새로운 이름을 명시한다.

### 주의 사항

이중화 대상 테이블일 경우 테이블의 이름을 변경할 수 없다.

---

## 예제

〈질의〉 테이블 employee 의 이름을 emp1 으로 변경하라.

```
iSQL> RENAME employee TO emp1;  
Rename success.
```

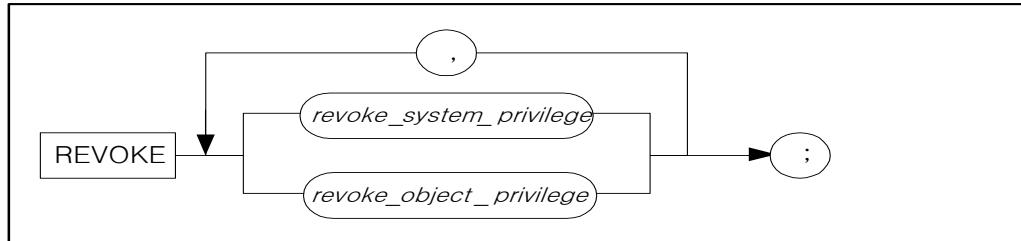
또는

```
iSQL> ALTER TABLE employee  
      RENAME TO emp1;  
Alter success.
```

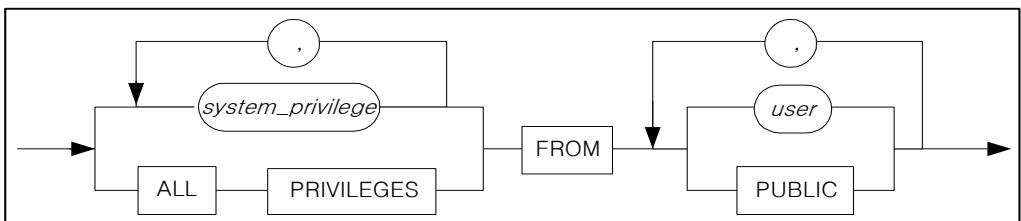
# REVOKE

## 구문

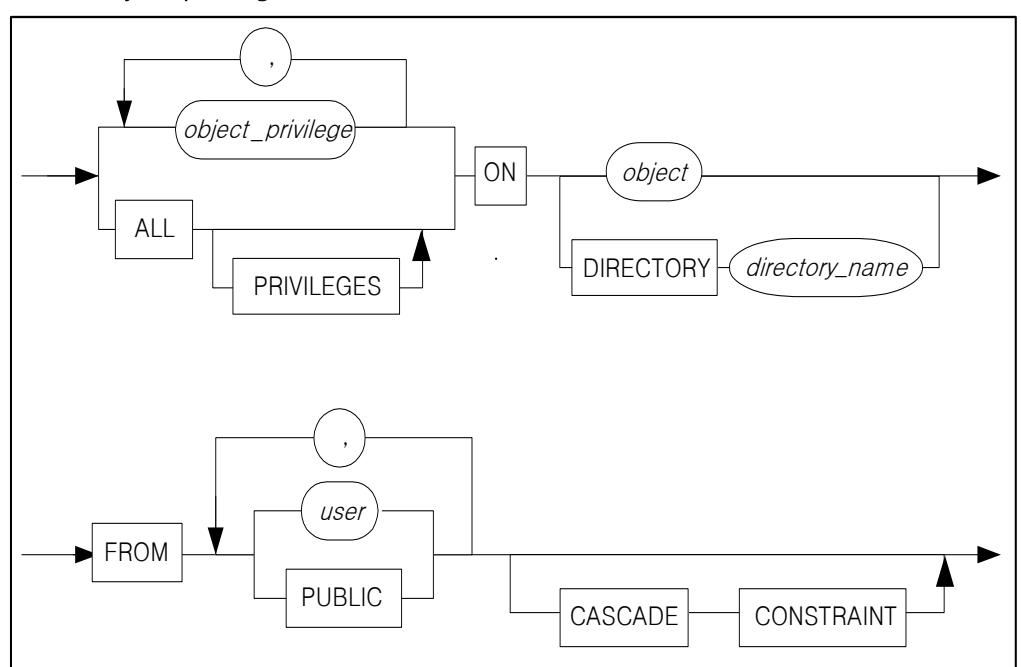
*revoke ::=*



*revoke\_system\_privilege ::=*



*revoke\_object\_privilege ::=*



---

## 전제 조건

SYS 사용자이거나 권한을 해제하려면 GRANT 명령으로 직접 부여되었던 권한들이어야 한다.

---

## 설명

명시된 사용자가 가진 시스템 접근 권한을 해제하거나 특정 객체에 대한 객체 접근 권한을 해제할 수 있다.

시스템 및 객체 접근 권한을 해제하려면 GRANT 명령으로 직접 부여되었던 권한들에 대해서만 해제할 수 있다.

### 시스템 접근 권한 (System privilege)

*system\_privilege*

GRANT 의 system privileges 테이블 참고

ALL PRIVILEGES

모든 시스템 접근 권한을 해제한다.

FROM user

시스템 접근 권한을 해제할 사용자를 명시한다.

FROM PUBLIC

모든 사용자로부터 시스템 접근 권한을 해제한다.

〈참고〉

\* ALL PRIVILEGES 키워드로 시스템 접근 권한을 부여했을 때, ALL PRIVILEGES 로 시스템 접근 권한을 해제 할 수 있다. 즉, 권한 해제 방법은 REVOKE ALL PRIVILEGES FROM *user* 로서 시스템 접근 권한을 모두 해제 할 수 있을 뿐만 아니라 REVOKE SELECT ANY TABLE FROM *user* 등을 이용해 각각의 system privilege 를 해제 할 수 있다.

\* PUBLIC 에게 시스템 접근 권한을 부여했을 때, PUBLIC 으로부터 시스템 접근 권한을 해제할 수 있다.

### 객체 접근 권한 (Object privilege)

*object\_privilege*

GRANT 의 object privileges 테이블 참고

ALL [PRIVILEGES]

사용자가 가지는 객체 접근 권한을 모두 해제한다.

ON object

해당 객체를 명시. 객체에는 테이블, 시퀀스, 저장 프로시저가 있다.

ON DIRECTORY directory\_name

명시한 디렉토리에 대한 객체 권한을 해제한다.

FROM user

해당 객체에 대한 객체 접근 권한을 해제 할 사용자를 명시한다.

FROM PUBLIC

모든 사용자로부터 객체 접근 권한을 해제한다.

#### CASCADE CONSTRAINTS

REFERENCES 권한 또는 ALL [PRIVILEGES]를 해제할 때 사용하는 옵션으로 사용자의 권한이 해제되면 관련된 모든 referential integrity constraints 도 함께 삭제한다.

\* ALL [PRIVILEGES]로 객체 접근 권한을 해제할 때, 사용자가 가지는 모든 객체 접근 권한이 해제된다. 즉, 시스템 접근 권한과 달리 ALL [PRIVILEGES] 키워드로 객체 접근 권한을 부여하지 않아도, 사용자가 가지는 객체 접근 권한들은 모두 해제된다. 예를 들어, GRANT SELECT ON *object* TO *user* 를 부여하여 질의 수행 후 REVOKE SELECT ON *object* FROM *user* 로서 권한 해제는 물론 REVOKE ALL ON *object* FROM *user* 로서 객체 접근 권한을 해제할 수 있다.

\* PUBLIC에게 객체 접근 권한을 부여했을 때, PUBLIC으로부터 객체 접근 권한을 해제할 수 있다.

---

## 예제

### 시스템 접근 권한

\* GRANT 예제 중 <복합 질의> 내용을 참고 한다.

### 객체 접근 권한

\* GRANT 예제 중 객체 접근 권한 <질의 1> 참고한다.

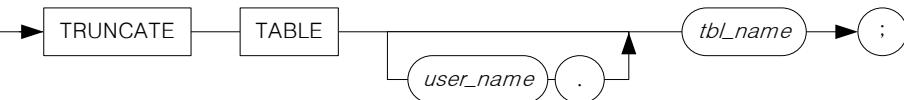
```
iSQL> CONNECT uare6/rose6;
Connect success.
iSQL> REVOKE SELECT, DELETE ON sys.employee
      FROM uare7, uare8;
Revoke success.
```

```
iSQL> CONNECT uare7/rose7;
Connect success.
iSQL> SELECT eno, ename FROM sys.employee WHERE eno = 15;
[ERR-311B1: You must have SELECT_ANY_TABLE privilege(s) for executing
this statement.]
      <= employee 테이블에 대한 SELECT 와 DELETE 권한 해제 후 SELECT
문을 수행 시 오류 메시지를 볼 수 있다.
```

# TRUNCATE TABLE

## 구문

*truncate ::=*



## 전제 조건

SYS 사용자이거나 테이블이 현재 사용자의 스키마이거나 DROP ANY TABLE 시스템 권한을 가진 사용자여야 한다.

## 설명

명시된 테이블의 모든 레코드를 삭제한다.

*user\_name*

레코드가 삭제될 테이블 소유자 이름을 명시한다. 생략하면 알터베이스는 현재 세션에 연결된 사용자의 스키마에 속한 것으로 간주한다.

*tbl\_name*

레코드가 삭제될 테이블 이름을 명시한다.

TRUNCATE 문을 수행한 경우는 해당 테이블에 할당된 모든 페이지를 free page로 반납하기 때문에 다른 테이블들이 해당 페이지들을 이용할 수 있다. 그러나 DELETE 문을 수행하여 해당 테이블의 모든 레코드를 삭제한 경우는 free page가 생기더라도 그 페이지를 데이터베이스 공간으로 다시 반납하지 않고 해당 테이블 내에서만 이용할 수 있기 때문에 메모리 사용량이 줄지 않는다.

DDL 이므로 이 구문을 성공적으로 수행한 후에는 rollback을 할 수 없다.

*table\_name* 에는 큐 테이블 명을 지정하여 ENQUE 된 메시지를 한꺼번에 삭제할 수 있다.

---

## 주의 사항

레코드의 삭제가 성공적으로 수행되었다면 삭제된 레코드를 복구할 수 없다. 그러나 수행 완료 전에 오류가 발생한 경우나 서버가 죽은 경우엔 rollback 이 된다.

---

## 예제

〈질의〉 테이블 employee 의 모든 데이터를 삭제하라.

```
iSQL> TRUNCATE TABLE employee;  
Truncate success.
```

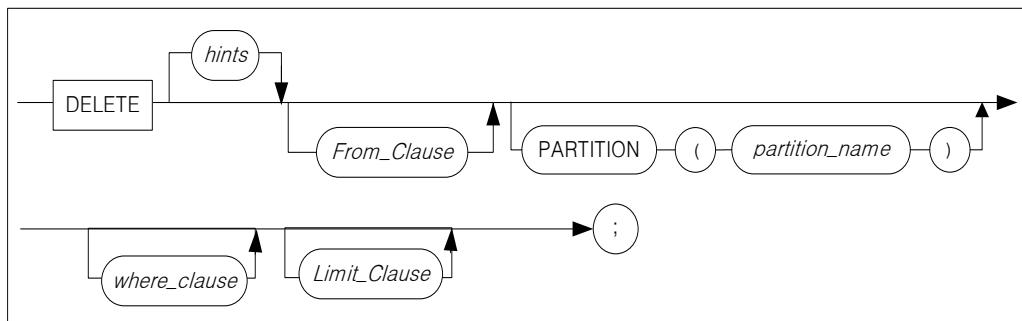
## 4. 데이터 조작어

이 장에서는 데이터를 액세스에 사용되는 DML 문장에 대해서 상세한 예를 들어서 설명하고 있다.

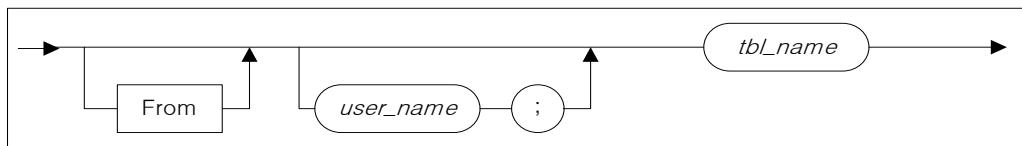
# DELETE

## 구문

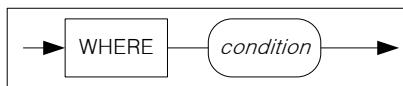
*delete ::=*



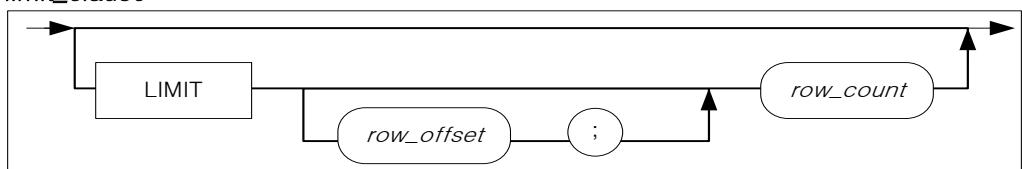
*from\_clause ::=*



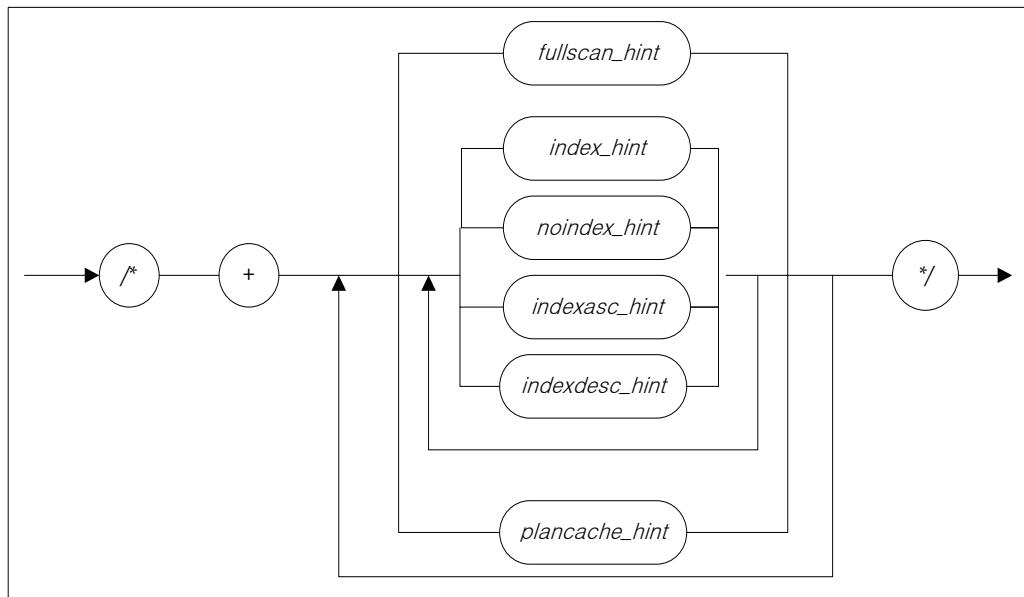
*where\_clause ::=*



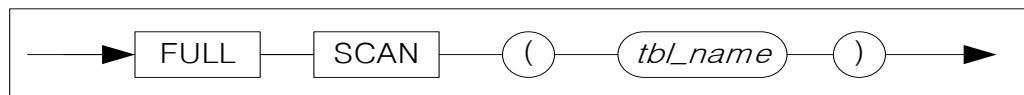
*limit\_clause ::=*



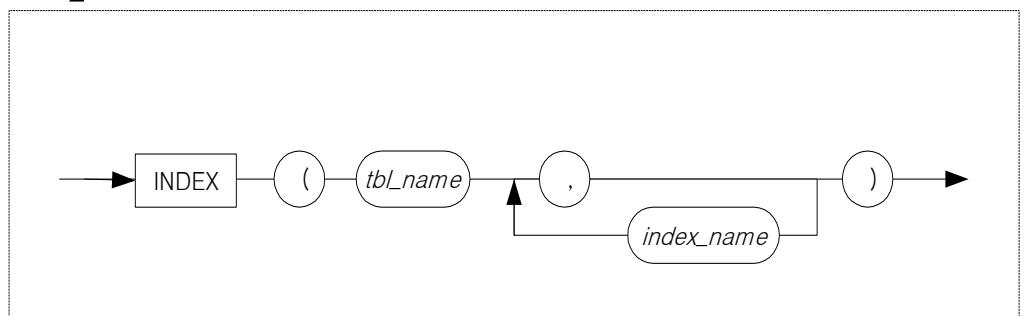
*hints* ::=



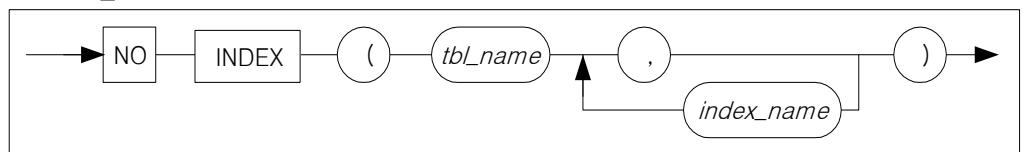
*fullscan\_hint* ::=



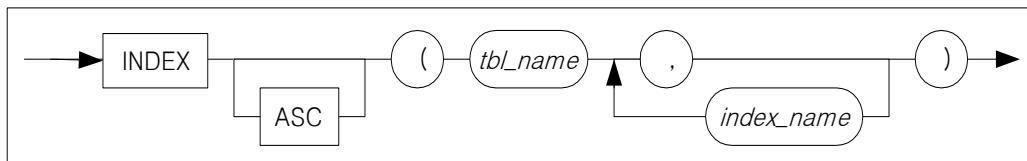
*index\_hint* ::=



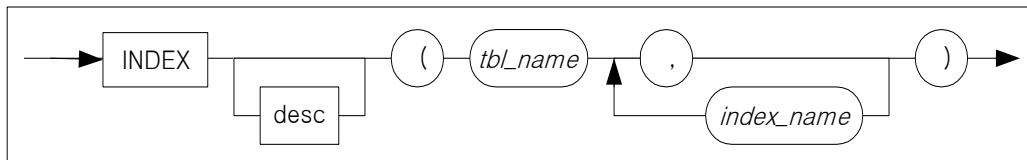
*noindex\_hint* ::=



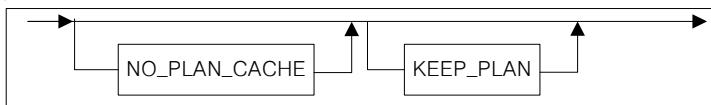
*indexasc\_hint* ::=



*indexdesc\_hint* ::=



*plancache\_hint* ::=



## 전제 조건

테이블의 레코드를 삭제하기 위해 SYS 사용자이거나 DELETE ANY TABLE 시스템 권한을 가진 사용자여야 한다.

## 설명

조건을 만족하는 레코드를 해당 테이블에서 삭제한다. 또한 특정 파티션에 있는 데이터를 삭제할 수 있다.

조건(WHERE)절은 검색(SELECT)의 조건과 동일하다.

조건절이 생략되었을 경우 모든 데이터를 삭제한다.

*user\_name*

삭제될 레코드의 테이블 소유자 이름을 명시한다. 생략하면 알터베이스는 현재 세션에 연결된 사용자의 스키마에 속한 것으로 간주한다.

*tbl\_name*

삭제될 레코드를 포함한 테이블 이름을 명시한다.

---

## HINTS 기능

알티베이스는 SQL 문 내에 주석을 달아 명령어들을 옵티마이저로 전달할 수 있다. 옵티마이저는 plan node를 선택하기 위해 이러한 힌트를 사용한다. +(plus sign)은 알티베이스가 주석을 hints로 변환하게 하며 주석 기호 바로 다음에 위치해야 한다(no space).

hints 부분에 문법 오류가 있어도 무시하고 검색문은 실행된다. 그러나 hints의 기능은 수행하지 않는다.

- FULL SCAN: 테이블에 이용 가능한 인덱스가 존재하더라도 인덱스를 사용하지 않고 테이블 전체를 검색한다.
- INDEX: 해당 테이블에 대한 접근 방법으로 나열된 인덱스에 대해서 index scan을 사용한다.
- NO INDEX: 해당 테이블에 대한 접근 방법으로 나열된 index scan을 사용하지 않는다.
- INDEX ASC: 해당 테이블에 대한 접근 방법으로 나열된 인덱스에 대해서 index scan을 사용하되, 오름 차순으로 탐색한다.
- INDEX DESC: 해당 테이블에 대한 접근 방법으로 나열된 인덱스에 대해서 index scan을 사용하되, 내림 차순으로 탐색한다.
- plancache\_hint : plan cache에서 사용할 수 있는 힌트는 NO\_PLAN\_CACHE와 KEEP\_PLAN이다. NO\_PLAN\_CACHE는 생성된 플랜을 캐시하지 않도록 하는 힌트이다. KEEP\_PLAN은 한 번 생성된 플랜이 참조하는 테이블의 통계 정보가 변경되더라도 플랜이 재생성되는 것을 방지하고 그대로 사용할 수 있도록 한다. KEEP\_PLAN은 direct-execute 수행 쿼리뿐 아니라 prepare-execute로 수행하는 쿼리에서도 사용이 가능하다.

---

## 예제

### 단순 데이터 삭제

〈질의〉 모든 주문을 삭제한다.

```
iSQL> DELETE FROM orders;  
30 rows deleted.
```

### 파티션 데이터 삭제

〈질의〉 T1 테이블의 P2 파티션을 삭제한다

```
iSQL> DELETE FROM T1 PRTITION (P2);
```

## 부 질의를 갖는 데이터 삭제

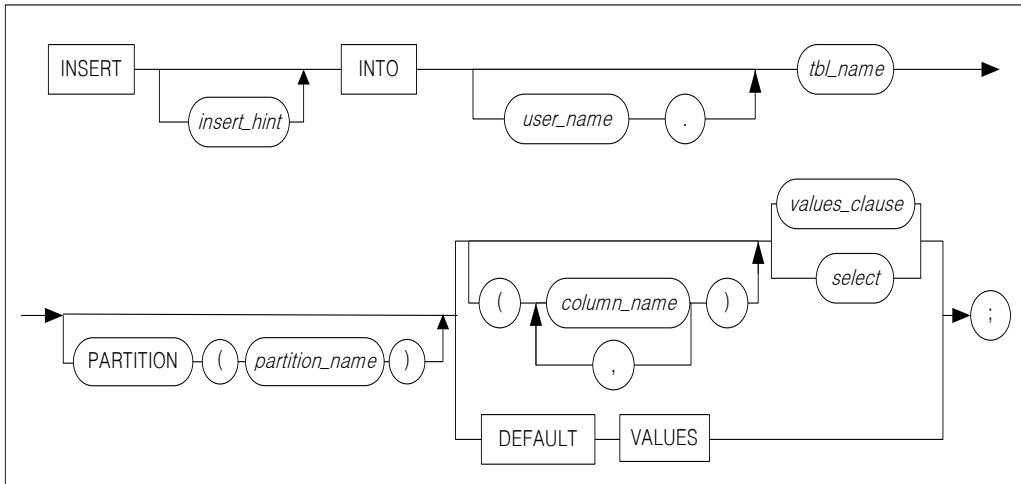
〈질의〉 KMKIM 직원이 받은 주문들을 삭제한다.

```
iSQL> DELETE  
      FROM orders  
     WHERE eno = (SELECT eno FROM employee  
                  WHERE ename = 'KMKIM');  
9 rows deleted.
```

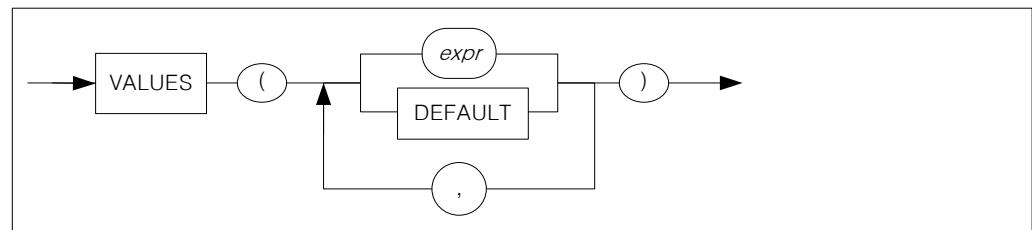
# INSERT

## 구문

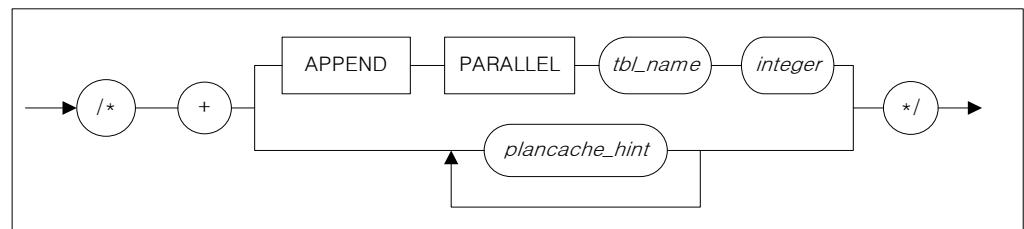
*insert ::=*



*values\_clause ::=*



*insert\_hints ::=*



## 전제 조건

테이블에 레코드를 삽입하기 위해 SYS 사용자이거나 INSERT ANY

TABLE 시스템 권한을 가져야 한다.

## 설명

명시한 테이블 또는 특정 파티션에 새로운 레코드를 삽입하는 것으로 만약 해당 테이블에 인덱스가 존재할 경우엔 인덱스 정보도 변경된다.

*user\_name*

레코드가 삽입될 테이블의 소유자 이름을 명시한다. 생략하면 알티베이스는 현재 세션에 연결된 사용자의 스키마에 속한 것으로 간주한다.

*tbl\_name*

레코드가 삽입될 테이블의 이름을 명시한다.

NULL

일부 칼럼은 명시하고 일부 칼럼을 명시하지 않을 경우에 명시하지 않는 칼럼에 기본값이 없으면 널이 삽입된다. ( TIMESTAMP 칼럼은 INSERT 수행 시 기본값으로 시스템 시간 값을 설정한다. 따라서 TIMESTAMP 칼럼 값을 명시하지 않을 경우 널이 아닌 시스템 시간 값이 삽입된다. )

VALUES 절에 명시적으로 널을 기술하면 널이 삽입된다.

DEFAULT

VALUES 절에 DEFAULT를 명시하면 해당 칼럼에 정의된 DEFAULT 값이 삽입된다.

전체 칼럼들에 대해 DEFAULT 값을 삽입하고자 할 경우엔 DEFAULT VALUES 절을 사용한다.

TIMESTAMP 칼럼에 DEFAULT를 명시하면 시스템 시간 값이 삽입된다.

INSERT SELECT

SELECT의 결과를 삽입하는 것으로 삽입할 테이블과 SELECT 할 테이블이 동일할 수도 있다.

삽입할 칼럼의 개수와 SELECT 한 칼럼의 개수는 동일해야 하며 호환 가능한 데이터 타입이어야 한다.

## INSERT\_HINTS 기능

- 알티베이스 SQL 구문 내에 주석을 달아 명령어들을 옵티마이저로 전달할 수 있다. 옵티마이저는 plan node를 선택하기 위해서 이러한 힌트를 사용한다. +(plus sign)는 알티베이스가 주석을 hints로 변환하게 하며 주석 기호 바로 다음에 빈 공간 없이 위치해야 한다.
- INSERT SQL 문 내에 주석을 달아 입력하는 방식을 결정할 수 있다. HINTS 기능은 ‘INSERT… INTO tbl\_name SELECT…’ 문장에서만 PARALLEL 사용이 가능하다. 그 외의 문장에서는 HINTS 기능이 무시된다.
- APPEND : direct-path INSERT 방식으로 수행된다.  
데이터가 입력될 때 페이지의 빈 공간을 찾아 들어가는 대신 새로운 페이지를 만들어 데이터를 입력한다.
- PARALLEL : 해당 테이블에 대한 parallel의 수행 여부를 설정하여 direct-path INSERT를 수행한다.  
우선 [RARALLEL DML MODE](#)에서 PARALLEL DML을 수행한다고 지정한 후, insert\_hint에서 PARALLEL의 정도를 (integer가 2 이상) 표시한다.  
만약 PARALLEL의 정도를 1로 표시하면, 대상 테이블의 PARALLEL 속성을 따른다.

---

## 주의 사항

명시한 칼럼의 개수와 삽입할 값들의 개수는 동일해야 하며 호환 가능한 데이터형 이어야 한다.

파티션을 지정할 경우 해당 파티션에 맞지 않는 값일 경우 입력할 수 없다.

테이블에 정의된 기본값을 생략할 경우, 널 값이 삽입된다. 단 해당 칼럼에 NOT NULL 조건이 없어야 한다.

direct-path INSERT에는 다음과 같은 제약조건이 존재한다.

- 대상 테이블은 디스크 테이블이며, LOB 칼럼은 포함할 수 없다. 또한 인덱스를 가질 수 없다.
- 대상 테이블은 이중화를 할 수 없다.
- 대상 테이블은 트리거나 참조 무결성을 갖을 수 없다.

---

## 예제

### 단순 데이터 입력 명령

〈질의〉 HJKIM 인 고객의 정보를 입력하기

```
iSQL> INSERT INTO customer VALUES ( '800101-1212123', 'HJKIM', 'STUDENT',
NIBBLE'025282222', 'F', Byte'0101', Byte'150763', '서울 영등포구 여의도동
63 대한생명빌딩');
1 row inserted.
```

〈질의〉 MSJUNG 인 고객의 정보중 사번, 이름, 성별만 입력하기

```
iSQL> INSERT INTO employee(eno, ename, sex) VALUES(21, 'MSJUNG', 'F');
1 row inserted.
```

## 복합 데이터 입력 명령

〈질의〉 지연중인 주문 테이블 자료의 고객 번호와 주문일을  
delayed\_processing 테이블에 복사하라.

```
iSQL> CREATE TABLE delayed_processing(
cno CHAR(14), order_date DATE);
Create success.
iSQL> INSERT INTO delayed_processing
SELECT cno, order_date
FROM orders
WHERE PROCESSING = 'D';
1 row inserted.
```

## 파티션에 데이터 입력 명령

```
CREATE TABLE T1 ( I1 INTEGER, I2 INTEGER )
PARTITION BY RANGE ( I1 )
(
    PARTITION P1 VALUES LESS THAN ( 300 ),
    PARTITION P2 VALUES LESS THAN ( 400 ),
    PARTITION P3 VALUES DEFAULT
) TABLESPACE SYS_TBS_DISK_DATA;

INSERT INTO T1 PARTITION ( P1 ) VALUES ( 123, 456 );
1 row inserted.
```

## direct-path INSERT 사용 명령

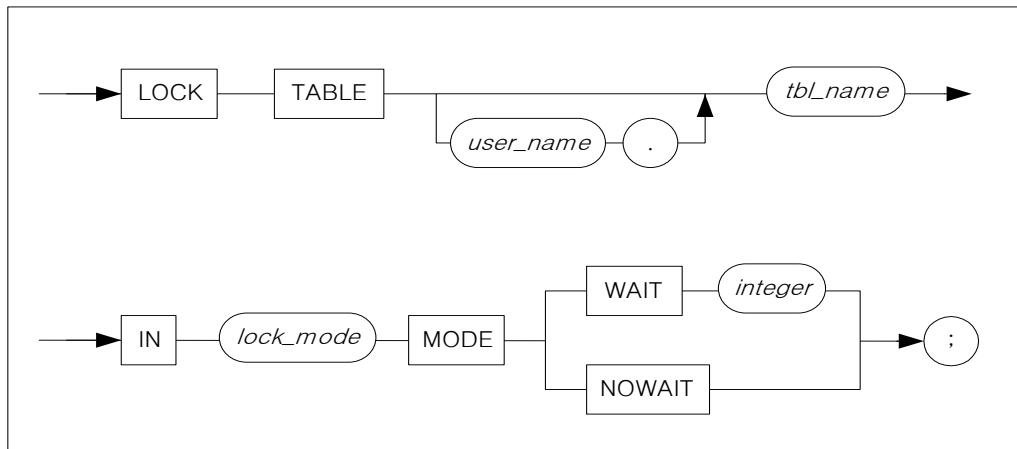
〈질의〉 T1 의 데이터 전부를 direct-path INSERT 방식으로 T2 에  
입력한다. 이 때 PARALLEL 의 정도는 5 로 한다.

```
INSERT /*+ APPEND PARALLEL(T2, 5) */ INTO T2 SELECT * FROM T1;
```

# LOCK TABLE

## 구문

*lock\_table ::=*



## 전제 조건

SYS 사용자이거나 테이블이 현재 사용자의 스키마이거나 LOCK ANY TABLE 시스템 권한을 가진 사용자여야 한다.

## 설명

특정한 모드내에서 테이블 자체에 잠금(lock table)을 거는 기능이다.  
잠금이 걸린 테이블은 트랜잭션이 반영되거나 롤백이 될 때까지 계속  
잠금을 유지한다.

*user\_name*

잠금이 걸릴 테이블의 소유자 이름을 명시한다. 생략하면  
알터베이스는 현재 세션에 연결된 사용자의 스키마에 속한 것으로  
간주한다.

*tbl\_name*

잠금이 걸릴 테이블의 이름을 명시한다.

```
LOCK TABLE [user_name.] {tbl_name} IN lock_mode MODE  
{wait_clause};
```

lock\_mode = ROW SHARE  
SHARE UPDATE  
ROW EXCLUSIVE  
SHARE ROW EXCLUSIVE  
SHARE  
EXCLUSIVE

*wait\_clause* = NOWAIT | WAIT n(INTEGER)

#### ROW SHARE

이 잠금이 걸린 테이블에 다른 트랜잭션에 의한 동시 접근을 허용한다. 그러나 다른 사용자들이 독점적인 접근을 위해 이 테이블에 EXCLUSIVE mode로 잠금을 거는 것은 금한다.

#### SHARE UPDATE

ROW SHARE 와 같은 뜻을 가진다.

#### ROW EXCLUSIVE

ROW SHARE 와 같다. 그러나 다른 트랜잭션이 SHARE mode로 잠금을 요청하는 것을 금한다. 데이터를 생성, 삽입하거나 또는 삭제하면 자동적으로 이 잠금을 얻는다.

#### SHARE ROW EXCLUSIVE

전체 테이블을 보는 것을 허용한다. 그러나 다른 트랜잭션들이 SHARE mode로 잠금을 요청하거나 행들을 생성하는 것을 금한다.

#### SHARE

다른 트랜잭션이 이 잠금이 걸린 테이블을 읽는 것은 허용하지만 생성하는 것은 금한다.

#### EXCLUSIVE

현재 트랜잭션이 이 잠금이 걸린 테이블을 읽거나 생성하는 것은 허용하지만 그 밖의 어떤 행위의 사용도 금한다.

#### *wait\_clause*

생략이 가능하고, 생략할 경우 행 단위 잠금이 걸리기까지 무한정 기다린다.

#### WAIT

행 단위 잠금(row lock)이 걸리기까지 n 초 만큼만 기다려 보고 획득 실패하면 에러로 처리한다.

#### NOWAIT

행 단위 잠금이 걸리기까지 기다리지 않고 이미 잠금이 걸렸다는 것을 다른 사용자들에게 바로 알려 예외로 처리한다.

SQL Statement	Mode of Table Lock	Lock Modes Permitted?				
		IS	IX	S	SIX	X
SELECT ... FROM <i>tbl_name</i> ...	IS	Y(IS)	Y(IX)	Y(S)	Y(SIX)	N(X)
INSERT INTO <i>tbl_name</i> ...	IX	Y(IX)	Y(IX)	N(SIX)	N(SIX)	N(X)
UPDATE <i>tbl_name</i> ...	IX	Y*(IX)	Y*(IX)	N(SIX)	N(SIX)	N(X)
DELETE FROM <i>tbl_name</i> ...	IX	Y*(IX)	Y*(IX)	N(SIX)	N(SIX)	N(X)
SELECT ... FROM <i>tbl_name</i> FOR UPDATE ...	IS	Y*(IX)	Y*(IX)	Y*(S)	Y*(SI X)	N(X)
LOCK TABLE <i>tbl_name</i> IN ROW SHARE MODE	IS	Y(IS)	Y(IX)	Y(S)	Y(SIX)	N(X)
LOCK TABLE <i>tbl_name</i> IN ROW EXCLUSIVE MODE	IX	Y(IX)	Y(IX)	N(SIX)	N(SIX)	N(X)
LOCK TABLE <i>tbl_name</i> IN SHARE MODE	S	Y(S)	N(SIX )	Y(S)	N(SIX)	N(X)
LOCK TABLE <i>tbl_name</i> IN SHARE ROW EXCLUSIVE MODE	SIX	Y(SIX)	N(SIX )	N(SIX)	N(SIX)	N(X)
LOCK TABLE <i>tbl_name</i> IN EXCLUSIVE MODE	X	N(X)	N(X)	N(X)	N(X)	N(X)

IS: row share (Intension share lock)  
IX: row exclusive (Intension exclusive lock)  
S: share  
SIX: share row exclusive (Share intentions exclusive lock)  
X: exclusive  
\* Y: 다른 트랜잭션에 의해 행 잠금 충돌이 일어나지 않은 경우, 그렇지 않으면 기다림이 발생한다.  
() 안은 1: 다른 트랜잭션에 의해 잠금 모드 전환이 허용되는 경우(Y), 현재 걸려있는 잠금 타입에 새로운 잠금을 걸었을 때의 잠금 변환이다.  
2: 다른 트랜잭션에 의해 잠금 모드 전환이 허용되지 않는 경우(N), 즉 한 트랜잭션에 의해서만 잠금을 요구 할 수 있는 경우, 그 트랜잭션에 새로운 잠금을 걸었을 때의 잠금 변환이다.

[표 4-1] Summary of Table Locks

## 예제

다음은 LOCK TABLE 과 SELECT 문이 사용 될 때 알티베이스가  
데이터 동시성, 무결성, 그리고 일관성을 어떻게 관리하는가를  
보여주는 예제이다.

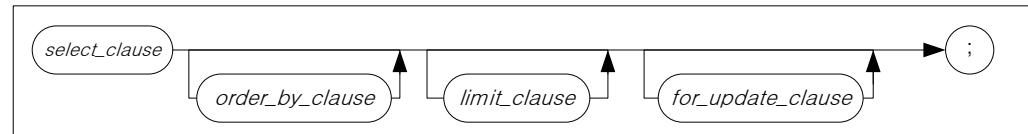
Transaction A	Time Point	Transaction B
iSQL> AUTOCOMMIT OFF; Set autocommit off success.		iSQL> AUTOCOMMIT OFF; Set autocommit off success.
	1	(request X lock on employee)  iSQL> LOCK TABLE employee IN EXCLUSIVE MODE; Command execute success.  (acquire X lock on employee)
iSQL> DROP TABLE employee; [ERR-11170: Trans exceeds lock wait time specified by user]	2	
	3	iSQL> UPDATE employee SET salary = 2500000 WHERE eno = 15; 1 row updated.
(request S lock on employee)  iSQL> LOCK TABLE employee IN SHARE MODE;  (request conflicts with the X lock already held by transaction B) wait wait wait	4	
	5	iSQL> COMMIT; Commit success.  (release X lock on employee)
(resume) Lock success. (acquire S lock on employee)  iSQL> SELECT salary FROM employee	6	

WHERE eno = 15; SALARY ----- 2500000 1 row selected.  (committed data is seen)		
iSQL> ROLLBACK; Rollback success.  (release S lock on employee)	7	
iSQL> LOCK TABLE employee IN EXCLUSIVE MODE; Lock success.  (acquire X lock on employee)	8	
		iSQL> SELECT SALARY FROM employee WHERE eno = 15; wait wait wait
iSQL> UPDATE employee SET eno = 30 WHERE eno = 15; 1 row updated.	10	
iSQL> COMMIT; Commit success. (release X loc on employee)	11	
	12	(resume) SALARY ----- 2500000 1 row selected.

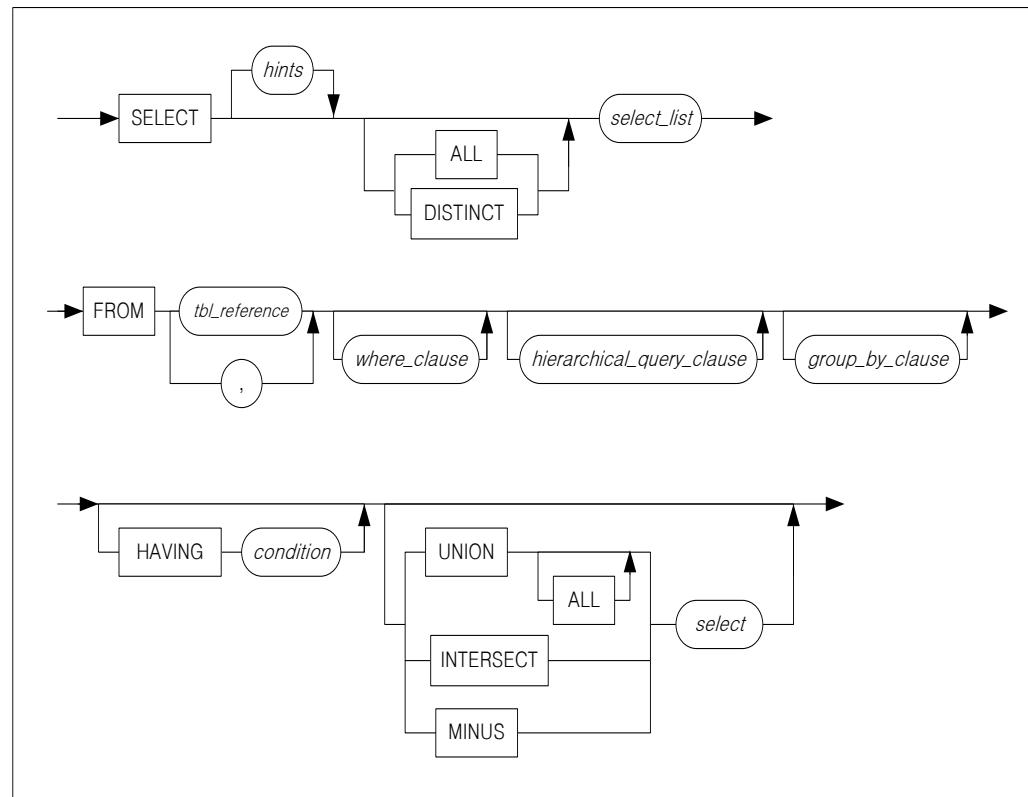
# SELECT

## 구문

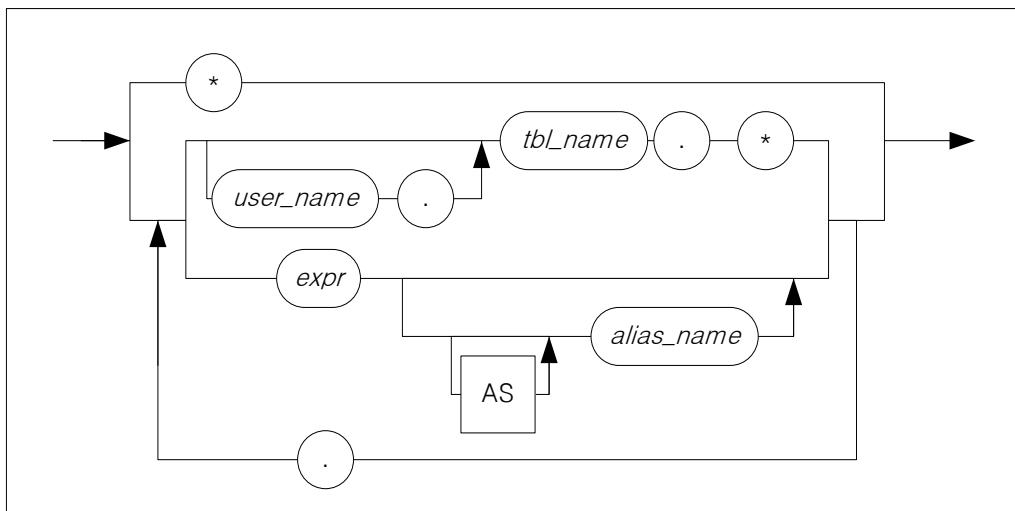
*select ::=*



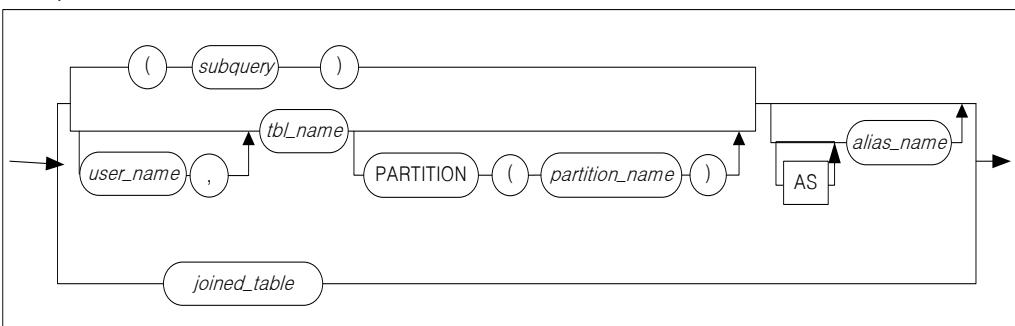
*select\_clause ::=*



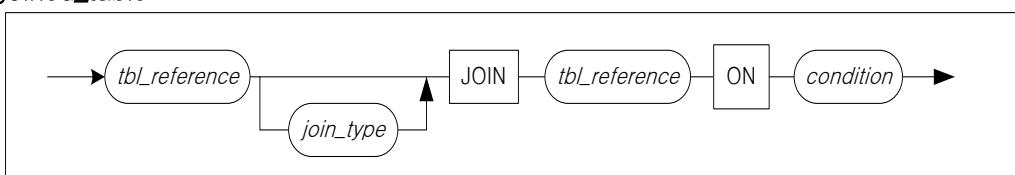
*select\_list ::=*



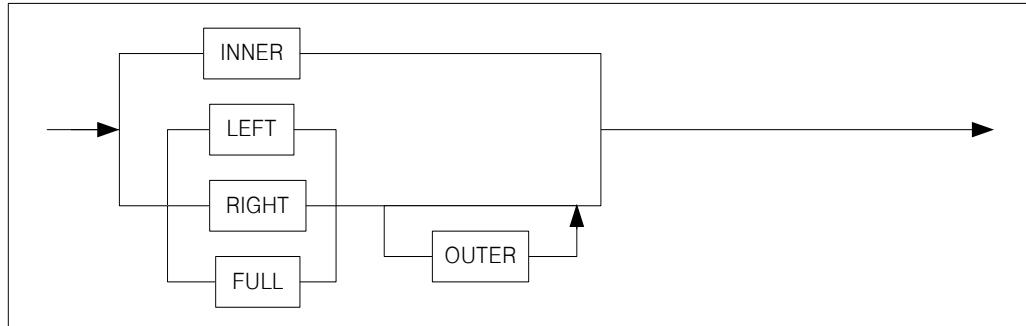
*tbl\_space ::=*



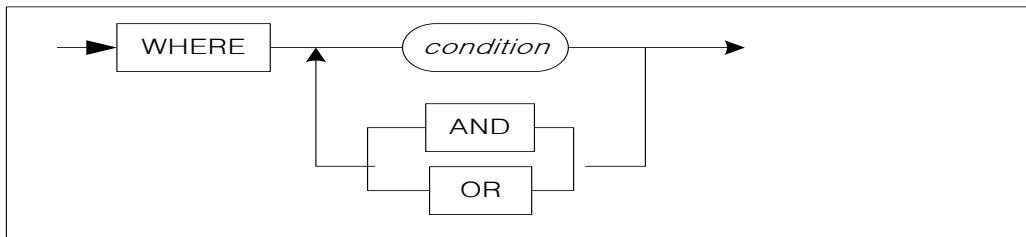
*joined\_table ::=*



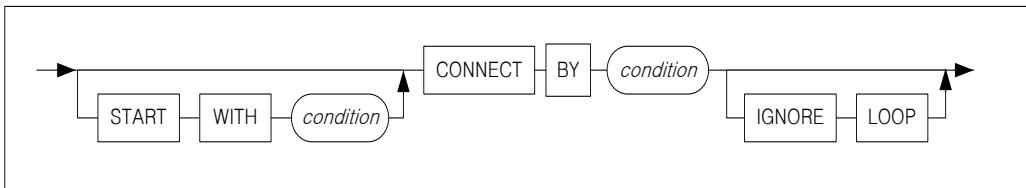
*join\_type* ::=



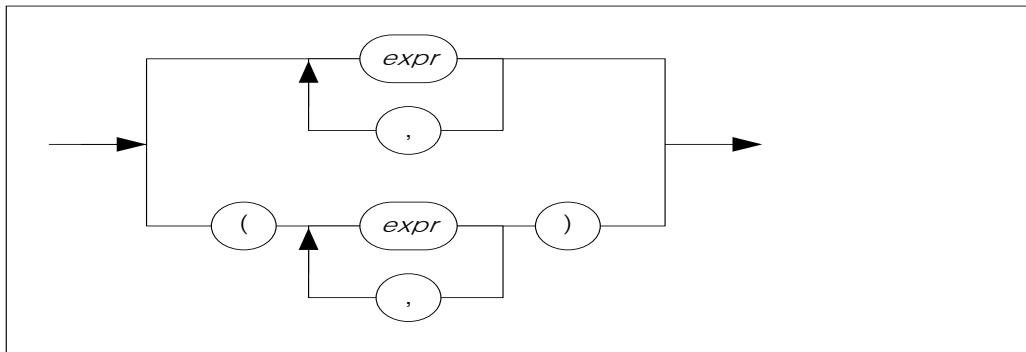
*where\_clause* ::=



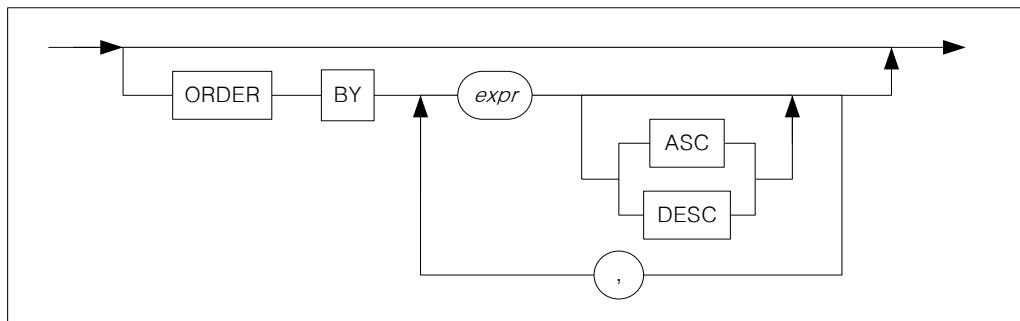
*hierarchical\_query\_clause* ::=



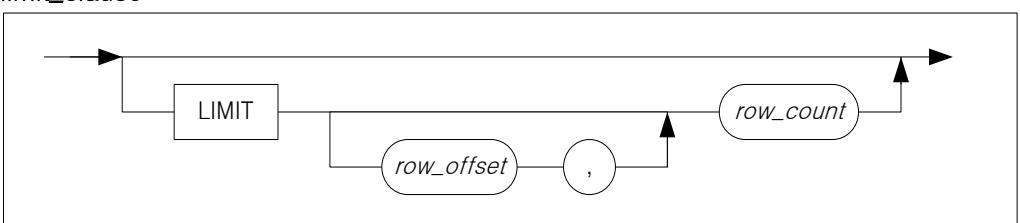
*expression\_list* ::=



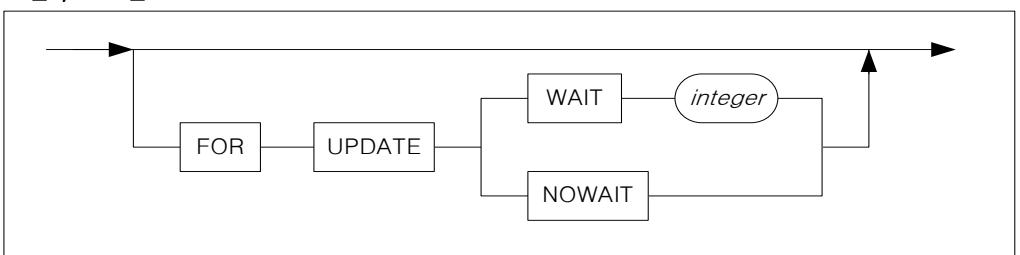
*order\_by\_clause ::=*



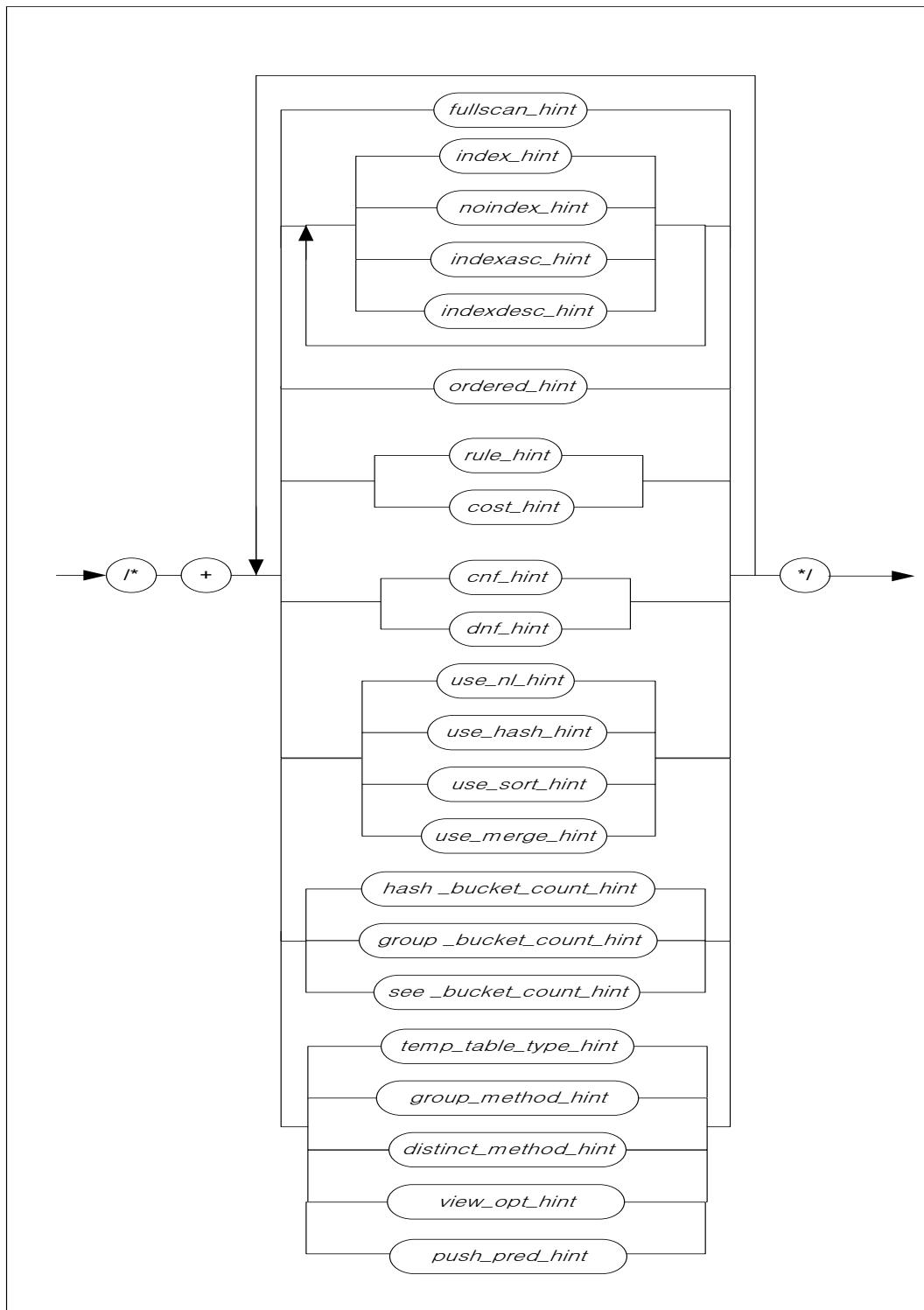
*limit\_clause ::=*



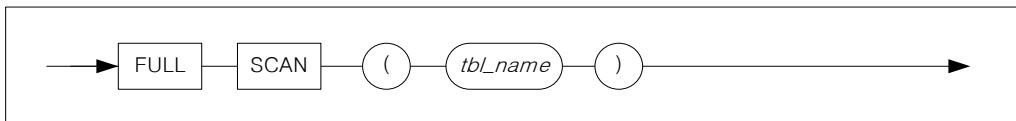
*for\_update\_clause ::=*



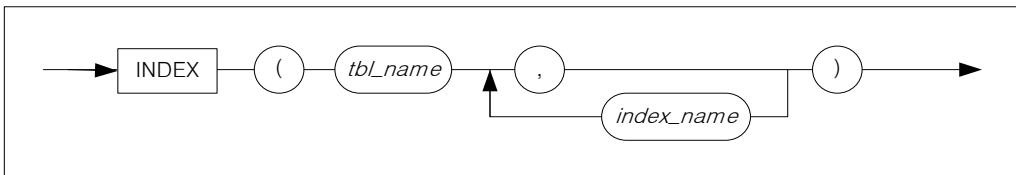
*hints* ::=



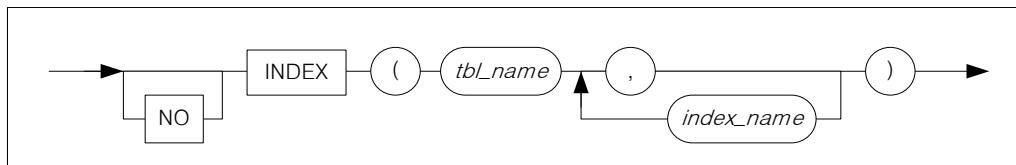
*fullscan\_hint* ::=



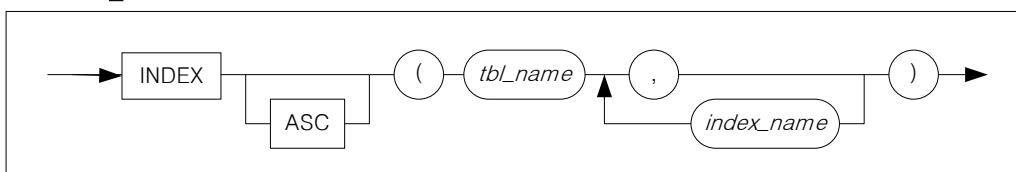
*index\_hint* ::=



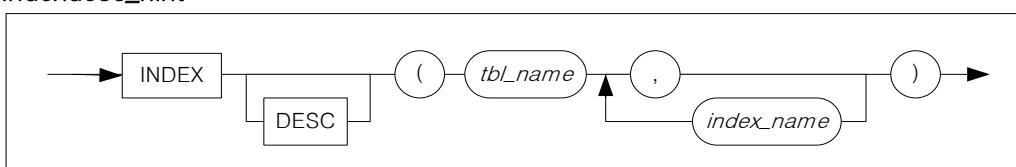
*noindex\_hint* ::=



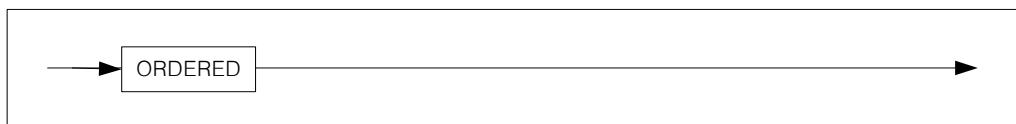
*indexasc\_hint* ::=



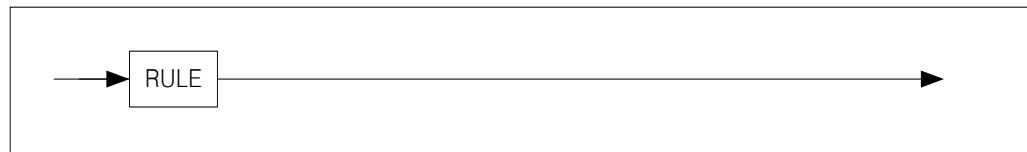
*indexdesc\_hint* ::=



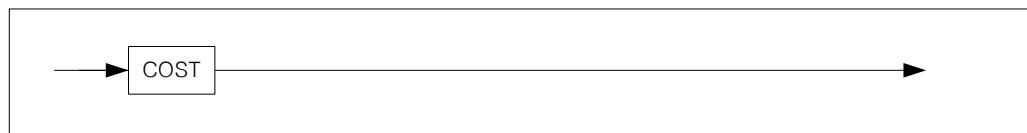
*ordered\_hint* ::=



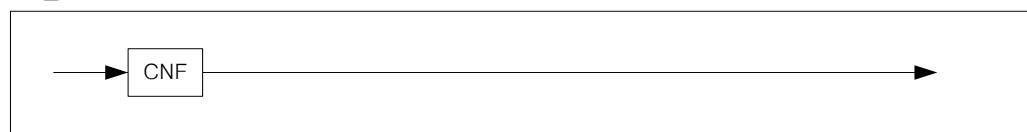
*rule\_hint ::=*



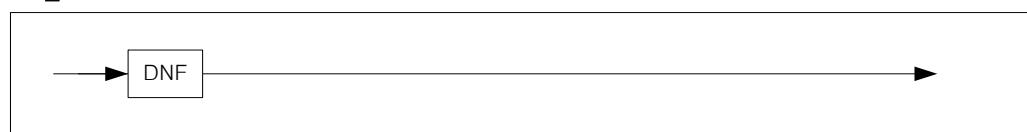
*cost\_hint ::=*



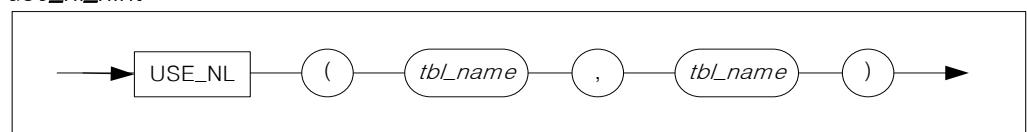
*cnf\_hint ::=*



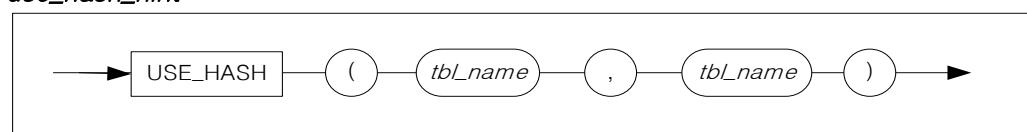
*dnf\_hint ::=*



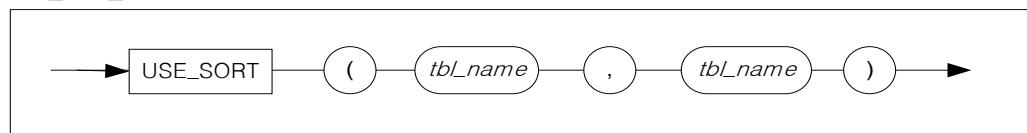
*use\_nl\_hint ::=*



*use\_hash\_hint ::=*



*use\_sort\_hint ::=*



*use\_merge\_hint ::=*



*hash\_bucket\_count\_hint ::=*



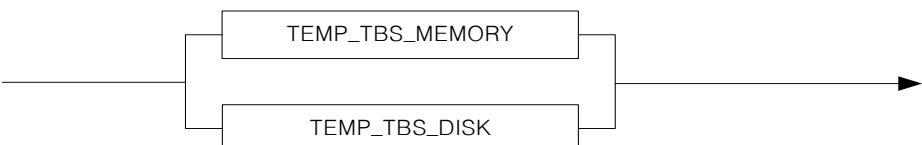
*group\_bucket\_count\_hint ::=*



*hash\_bucket\_count\_hint ::=*



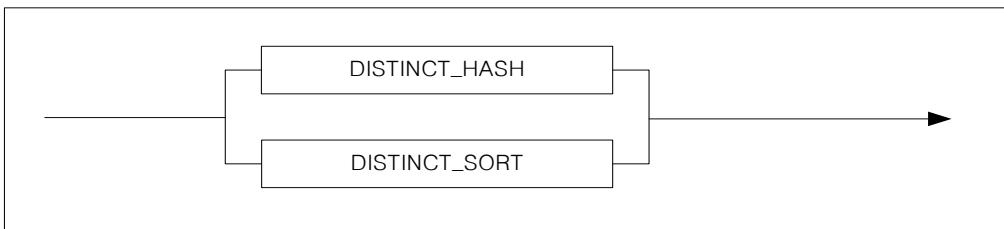
*temp\_table\_type\_hint ::=*



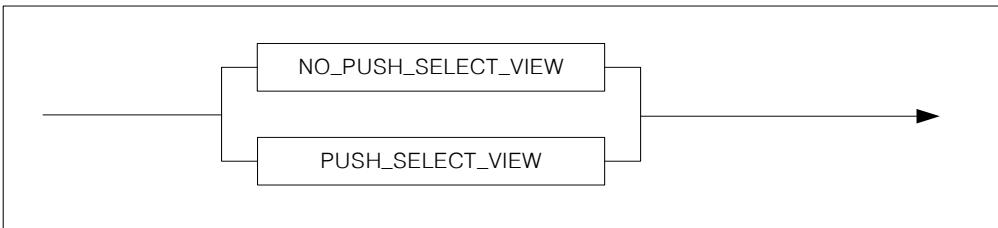
*group\_method\_hint ::=*



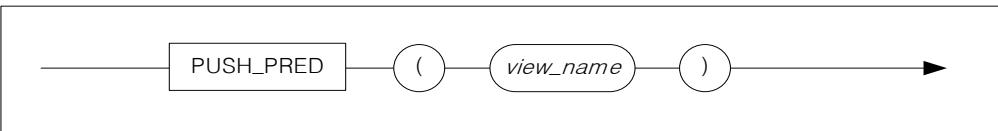
*distinct\_method\_hint* ::=



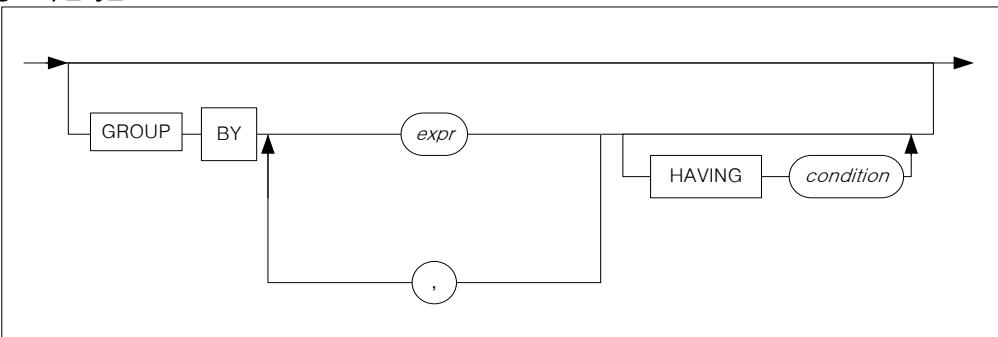
*view\_opt\_hint* ::=



*push\_pred\_hint* ::=



*group\_by\_clause* ::=



---

## 전제 조건

테이블이나 뷰의 기본 테이블에서 데이터를 검색하려면 SYS 사용자이거나 SELECT ANY TABLE 시스템 권한을 가져야 한다.

---

## 설명

## 검색 대상 (select\_list)

DISTINCT 를 명시할 경우 검색 시 중복된 레코드는 제거된다.

만약 GROUP BY 절이 존재한다면 상수, 그룹함수, GROUP BY 절에 명시된 표현식과 이들을 조합한 표현식 (이를 그룹 표현식이라고 하자)만 명시 가능하다.

그룹 함수가 하나 이상 존재할 경우 검색 대상의 다른 표현식도 GROUP BY 절에 존재하는 표현식의 조합이거나 그룹 함수 이어야 한다.

검색 바로 다음에 HINTS 를 명시할 수 있다.

## FROM 절

별명(alias\_name)을 두 번 이상 사용할 수 없다. 같은 테이블 명은 여러 번 사용할 수 있으나, 별명은 사용이 불가능하다.

별명을 명시하지 않고 같은 테이블 명을 두 번 이상 사용할 수 없다.

FROM 절에 사용되는 테이블 또는 뷰의 개수는 최대 32 개까지 허용한다.

## OUTER JOIN 기능

조인 조건을 만족하지 않는 데이터를 처리하기 위한 JOIN 의 확장 형태이다. (INNER) JOIN 이 두 테이블에 있는 키 값이 일치하는 데이터만 가져오는 것에 비해 OUTER JOIN 은 어느 한 쪽의 데이터를 모두 가져온다. 또한, 한 테이블의 행이 다른 테이블의 행에 대응되지 않을 때 반환된 결과 집합의 경우, 대응되지 않는 행을 갖고 있는 테이블로 확인된 모든 결과 집합 열에 대해 NULL 값이 제공된다.

## In-line View 기능

인라인 뷰는 SQL 문에서 사용 가능한 별칭을 사용하는 하위 질의이다. 인라인 뷰는 기본 질의의 FROM 절에 명명된 하위 질의를 사용하는 것과 유사하다.

## *where\_clause*

WHERE 구문의 condition 에 대한 설명은 9 장 [조건 연산자](#)를 참조한다.

## Hierarchical Query 절

계층적 질의(Hierarchical query)란 임의의 테이블이 계층적 데이터를 가지고 있다면 주어진 검색 조건을 갖는 루트 행에 대하여, 그 행과 그의 종속 행에 대한 계층적 조건을 만족하는 행들을 검색하는 기능이다.

## START WITH 절

---

계층적 질의의 루트로 사용될 행을 식별하는 조건을 명시한다.

이 조건을 만족하는 모든 행들을 루트로 사용한다.

이 절을 생략하면 알티베이스는 테이블에 있는 모든 행들을 루트 행들로 사용한다.

CONNECT BY 절이 없을 경우는 사용할 수 없으며, CONNECT BY 절이 있을 경우 생략이 가능하다.

부연질의(subquery)를 포함할 수 없다.

### **CONNECT BY 절**

---

계층 구조의 부모 행들과 자식 행들간의 관계를 식별하는 조건을 명시한다.

이전에 검색된 행과 현재 행을 구분하기 위해서 prior 연산자를 사용한다. 즉, 부모행을 언급하기 위해 반드시 prior 연산자를 사용해야만 한다.

Prior 연산자는 CONNECT BY 절이 사용된 질의문에서만 사용 가능하다.

Prior 연산자는 CONNECT BY 절을 사용하는 SELECT 문의 select\_list, WHERE 절, CONNECT BY 절에서 사용할 수 있다.

Prior 연산자 이후에는 함수를 포함한 열이나 그에 관한 condition 을 사용할 수 있다.

부연질의를 포함할 수 없다.

JOIN 을 사용할 수 없다.

WHERE 절 이후 ORDER BY, GROUP BY, HAVING 절 이전에 사용하여야 한다.

### **LEVEL**

---

루트 행의 LEVEL 을 1로 할 때, 루트 행으로 부터의 종속 행간의 계승 관계를 갖는 행의 숫자를 나타내는 의사열<sup>1</sup>이다. 즉, 루트의 자식 행의 LEVEL 은 2, 그것의 자식 행의 LEVEL 은 3 등이다.

WHERE 절, ORDER BY 절, GROUP BY 절, HAVING 절에서 사용 가능하다.

CONNECT BY 절이 없더라도, select\_list 절에서 사용할 수 있다.  
(e.g.: select level from t1;)

---

<sup>1</sup> 의사열(Pseudocolumn): 임의의 테이블의 열과 같으나 테이블에 실제 저장되는 것은 아니다. 의사열을 검색할 수 있으나 그 값을 삽입, 수정 또는 삭제할 수 없다. 이 외에 알티베이스는 CURRVAL 과 NEXTVAL 을 지원한다.

## IGNORE LOOP

---

계층 구조에서 loop 이 생성될 경우 알티베이스는 오류를 반환한다.  
(여기서 loop 이란 한 행이 다른 행의 부모 행도 되고 자식 행도 되는 경우를 말한다.) 단, IGNORE LOOP 이 명시되었을 경우, 질의 수행시 loop 이 생성되더라도 오류가 반환되지 않고, loop 을 발생시키는 행을 질의의 결과 집합에 추가하지 않는다.

### GROUP BY 절

명시된 표현식의 값이 같은 레코드들이 하나의 그룹 단위로 묶여진다.

WHERE *condition* 절을 사용하여 그룹을 제한할 수 없다.

HAVING *condition* 절을 사용하여 그룹을 제한할 수 있다.

HAVING 및 GROUP BY 절은 WHERE 절 다음에 넣으며,  
알티베이스는 그 명령문 순서대로 평가한다. 만약 ORDER BY 절을 사용할 경우엔 맨 마지막에 넣는다.

### HAVING condition 절

GROUP BY 절이 존재할 때만 사용할 수 있다.

HAVING 절에 사용하는 표현식은 상수, 그룹함수, 그룹 표현식만 가능하다.

Condition 의 구문 설명은 9 장 조건 연산자 참조

### UNION(ALL), INTERSECT, MINUS

6 장 집합 연산자 참조한다.

### ORDER BY 절

검색된 레코드들을 오름차순 (default) 또는 내림차순으로 재 배열하여 보여준다.

전체 검색문에 한번 사용 가능하고 부연질의에서는 사용할 수 없다.

표현식을 명시한 경우엔 표현식 연산 후 그 값에 대해 정렬하고 상수를 명시한 경우엔 검색 대상의 위치를 나타내는 것으로 검색 값에 대해 정렬 한다.

Set operators (UNION, INTERSECT 등)를 사용한 경우,  
위치(position) 또는 검색 대상의 별명만 사용 가능하다.

GROUP BY 가 존재할 경우 그룹 표현식만 사용 가능하다.

검색 대상에 DISTINCT 가 존재할 경우 ORDER BY 절에는 검색 대상의 표현식 또는 이들의 조합만 사용 가능하다.

### LIMIT 절

row offset: 검색하고자 하는 시작 레코드

row count: 검색하고자 하는 레코드의 개수. row count 값 만 지정하면 즉, LIMIT n 을 사용한 경우 상위 n 개 행만 가져온다.  
부연질의에서도 사용할 수 있다.

#### FOR UPDATE 절

현재 트랜잭션 진행이 끝날 때 까지 다른 사용자들이 행(row)을 잠그거나 수정할 수 없도록 선택된 행을 잠근다.

FOR UPDATE [wait\_clause] 에서 wait\_clause 는 생략 가능하다.

최 상위 SELECT 문에서만 지정 가능하다.

부연질의에는 지정이 불가능하다.

틀린 예: select eno from employee where (select eno from department for update);

DISTINCT, GROUP BY, aggregate(group) functions, set operators (UNION, INTERSECT 등)과 사용 할 수 없다.

---

## HINTS 기능

알티베이스는 SQL 문 내에 주석을 달아 명령어들을 옵티마이저로 전달할 수 있다. 옵티마이저는 plan node를 선택하기 위해 이러한 힌트를 사용한다. +(plus sign)은 알티베이스가 주석을 hints로 변환하게 하며 주석 기호 바로 다음에 빈 공간없이 위치해야 한다.

hints 부분에 문법 오류가 있어도 무시하고 검색문은 실행된다.  
그러나 hints 의 기능은 수행되지 않는다.

#### Table Access Method Hints

테이블 검색 방법에 대하여 지정할 수 있는 hints는 다음과 같다.  
뷰를 통하여 검색하는 경우, 뷰 내부에 사용된 테이블의 인덱스 이름을 힌트 구문에 사용하면 뷰에 대해서도 일반 테이블처럼 인덱스 힌트를 줄 수 있다.

- FULL SCAN: 테이블에 이용 가능한 인덱스가 존재하더라도 인덱스를 사용하지 않고 테이블 전체를 검색한다.
- INDEX: 해당 테이블에 대한 접근 방법으로 나열된 인덱스에 대해

#### Push Predicate Hints

뷰 외부의 WHERE 절의 조건 중 뷰와 관계된 조인 조건을 뷰 내부로 이동하여 처리한다.

- PUSH\_PRED ( view\_table ) : 뷰와 관계된 조인 조건을 뷰내부에서 먼저 처리되도록 하는 hint이다.

- INDEX ASC: 해당 테이블에 대한 접근 방법으로 나열된 인덱스에 대해서 index scan을 사용하되, 오름 차순으로 탐색한다.
- INDEX DESC: 해당 테이블에 대한 접근 방법으로 나열된 인덱스에 대해서 index scan을 사용하되, 내림 차순으로 탐색한다.
- NO INDEX: 해당 테이블에 대한 접근 방법으로 나열된 index scan을 사용하지 않는다.

#### Join Ordered Hints

- ORDERED: JOIN 순서를 OPTIMIZER가 결정하는 것이 아니라 FROM 절에 명시된 순서대로 한다. FROM 절의 순서대로 테이블 조인 순서를 결정하여 실행 계획 트리를 생성한다. ORDERDED를 명시하지 않은 경우, 기본적으로 FROM 절의 순서에 상관없이 optimizer에 의해서 테이블 조인 순서를 결정하여 실행 계획 트리를 생성한다.

#### Optimizer Mode Hints

- RULE: 질의문을 최적화하기 위하여 cost를 배제한 실행 계획 트리를 생성한다. 권장하지 않음.
- COST: 질의문을 최적화하기 위하여 cost를 고려한 실행 계획 트리를 생성한다. 권장

#### Normal Form Hints

- CNF: Conjunctive normal form은 최상위의 논리 연산자가 ‘And’이며 그 하위에 ‘Or’ 논리 연산자를 가지는 형태의 조건절이다. 질의문을 conjunctive normal form으로 변형하여 실행 계획 트리를 생성한다. 단, 조건절의 속성이 CNF 형태로 변경할 때 매우 많은 조건절을 생성할 경우에 CNF를 포기한다.
- DNF: Disjunctive normal form은 최상위의 논리 연산자가 ‘Or’이며 그 하위에 ‘And’ 논리 연산자를 가지는 형태의 조건절이다. Or 절에 의해서 나뉘어지는 조건절에 대해서 각각 인덱스를 사용할 기회를 가진다. 질의문을 disjunctive normal form으로 변형하여 실행 계획 트리를 생성한다. 단, or 절이 없는 경우 무시하며, 조건절의 속성이 DNF 형태로 변경할 때 매우 많은 조건절을 생성할 경우에 DNF를 포기한다.

#### Join Method Hints

- USE\_NL: 임의의 테이블 T1, T2를 조인 할 경우 nested loop join을 사용한다. T2, T1의 순서일 경우에는 해당 사항 없음.
- USE\_HASH: 임의의 테이블 T1, T2를 조인 할 경우 hash join을 사용한다. 단, hashing predicate이 존재하지 않는 경우에 nested loop join을 사용한다.

- USE\_SORT: 임의의 테이블 T1, T2를 조인 할 경우 sort join을 사용한다. 단, sorting predicate이 존재하지 않는 경우에 nested loop join을 사용한다.
- USE\_MERGE: 임의의 테이블 T1, T2를 조인 할 경우 sort merge join을 사용한다. 단, sorting predicate이 존재하지 않는 경우에 nested loop join을 사용한다.

#### Hash Bucket Size Hints

Hash join, group by, aggregation function, union, intersect, minus, distinct 등은 해싱 기법을 이용해 처리된다. 이 경우 레코드 건수에 따라 적절한 해시 버켓 수가 할당될 때 실행 처리 속도를 높일 수 있다.

비용 기반 최적화의 경우 내부적으로 테이블의 레코드 수에 따라 적절한 해시 버켓 수를 지정해 주지만, 사용자가 다음 힌트들을 사용해 임의로 해시 버켓 수를 조정하여 SQL 문을 튜닝할 수 있다.

HASH BUCKET COUNT: HASH, DISTINCT 노드의 해시 버켓 수 변경

GROUP BUCKET COUNT: GROUP-AGGREGATION, AGGREGATION 노드의 해시 버켓 수 변경

SET BUCKET COUNT: SET-INTERSECT, SET-DIFFERENCE 노드의 해시 버켓 수 변경

#### Temp Table Type Hints

Temp Table 은 중간 결과를 저장하는 임시 테이블이다. 이 임시테이블의 종류를 지정해줄수가 있다. 작은 양의 테이블이라면 메모리 임시 테이블이 유리하다.

- TEMP\_TBS\_MEMORY:  
중간 결과가 생성되면, Memory Temp Table에 저장
- TEMP\_TBS\_DISK:  
중간 결과가 생성되면, Disk Temp Table에 저장

#### Grouping Method Hints

Grouping 을 Sort-based 로 수행할지, Hash-based 로 수행할지를 결정할 수 있는 Hint 이다.

- GROUP\_HASH: Hash-based로 Grouping을 수행
- GROUP\_SORT: Sort-based로 Grouping을 수행

#### Distinction Method Hints

Distinction 을 Sort-based 로 수행할지, Hash-based 로 수행할지를 결정할 수 있는 Hint 이다

- DISTINCT\_HASH: Hash-based로 Distinction을 수행
- DISTINCT\_SORT: Sort-based로 Distinction을 수행

#### View Optimization Hints

뷰 최적화를 View Materialization 으로 수행할지, Push Selection 으로 수행할지를 결정할 수 있는 힌트이다.

- NO\_PUSH\_SELECT\_VIEW: View Materialization 기법으로 View 최적화
- PUSH\_SELECT\_VIEW: Push Selection 기법으로 View 최적화

#### Push Predicate Hints

뷰 외부의 WHERE 절의 조건 중 뷰와 관계된 조인 조건을 뷰 내부로 이동하여 처리한다.

PUSH\_PRED(view\_table) 힌트는 뷰와 관계된 조인 조건을 뷔 내부에서 먼저 처리되도록 하는 힌트이다.

## 질의 처리 제약

알티베이스는 SQL 질의 및 저장프로시저 수행에 있어 다음과 같은 제약을 가진다.

- 질의 처리에 사용되는 internal tuple의 개수는 최대 65536개 까지 사용할 수 있다.
- FROM절에 사용되는 테이블 또는 뷔의 개수는 최대 32개 까지 사용할 수 있다.
- WHERE, GROUP BY, ORDER BY 등과 같이 연산식이 사용 가능한 절에서 최대 32개까지의 테이블 또는 뷔를 사용할 수 있다.

위와 같은 제약을 위배하게 되면 다음과 같은 에러가 발생하게 된다.

- qpERR\_ABORT\_QTC\_TUPLE\_SHORTAGE  
: Too many DML statements in the stored procedure or too long query.
- qpERR\_ABORT\_QTC\_TOO\_MANY\_TABLES  
: Too many tables are referenced in a phrase

#### 내부 템플(Internal Tuple)의 개념

알티베이스는 질의 처리를 위해 템플 집합(Tuple set) 이라는 개념을 사용한다. 질의 처리 시 하나의 테이블 또는 연산 등을 처리하기 위하여 레코드 단위로 관리하게 되고 이를 내부 템플이라고 하며 이러한 템플은 최대 65536 개까지 사용할 수 있다.

질의 처리에 사용되는 튜플의 개수는 다음과 같은 필요에 의하여 사용하게 된다.

- 테이블 용 내부 튜플  
하나의 테이블(또는 뷰)에 대해 하나의 내부 튜플을 사용
- 임시 테이블 용 내부 튜플  
해싱 또는 정렬을 위해 생성되는 임시 테이블에 대해 하나의 내부 튜플을 사용
- 상수용 내부 튜플  
1, 'abc' 등과 같은 상수를 관리하기 위한 내부 튜플로 상수들의 크기 합이 4096 Bytes당 하나의 내부 튜플을 사용
- 호스트 변수용 내부 튜플  
호스트 변수 1024 개당 하나의 내부 튜플을 사용
- 연산 결과용 내부 튜플  
 $a1 + 3$  과 같이 +의 결과를 저장하기 위한 내부 튜플로 1024개의 연산 당 하나의 내부 튜플을 사용

위와 같은 기준으로 볼 때 사용하는 테이블의 개수가 n이라 하면 약  $3 * n$  개 이하의 internal tuple을 질의 처리에 사용한다.

주의할 점은 다음과 같이 뷰가 사용되는 경우 뷰 내부에서 사용되는 테이블 및 기타 내부 튜플 사용이 필요한 경우도 마찬가지로 내부 튜플이 필요하다.

예) SELECT \* FROM V1, V2 WHERE V1.i1 = V2.i1;

예를 들어, V1과 V2가 100개의 테이블로 구성된 뷰라고 하면 해당 질의의 처리를 위해서는 최소 202개 이상의 내부 튜플을 사용한다.

튜플 집합 내의 내부 튜플의 개수는 알티베이스가 질의 처리를 위해 사용하는 메모리 크기에 영향을 미치며, 기본 내부 튜플의 개수는 16개이고 질의 복잡도에 따라 최대 65536개까지 자동으로 동적 증가한다.

튜플에 대한 일부 정보는 실행 계획 트리에서 확인할 수 있다. 실행 계획 트리에 대한 자세한 내용은 *Administrator's Manual*을 참조한다.

---

## 예제

### 단순 검색 명령

<질의> 모든 사원의 이름, 고용일, 월급을 검색하라.

```
iSQL> SELECT ename, join_date, salary
      FROM employee;
      ENAME           JOIN_DATE        SALARY
```

```
SWNO
HJNO          1999/11/18 00:00:00  1500000
HSCHOI        2000/01/11 00:00:00  2000000
```

```
20 rows selected.
```

## 파티션을 사용한 검색

```
CREATE TABLE T1 (I1 INTEGER)
PARTITION BY RANGE (I1)
(
    PARTITION P1 VALUES LESS THAN (100),
    PARTITION P2 VALUES LESS THAN (200),
    PARTITION P3 VALUES DEFAULT
) TABLESPACE SYS_TBS_DISK_DATA;

INSERT INTO T1 VALUES (55);
INSERT INTO T1 VALUES (123);

SELECT * FROM T1 PARTITION (P1);
I1
-----
55
SELECT * FROM T1 PARTITION (P2);
I1
-----
123
SELECT * FROM T1 PARTITION (P3);
No rows selected.
```

## 검색 조건 사용

〈질의〉 월급이 100 만원 이하인 직원의 이름, 업무, 입사일, 월급을 월급 순서로 정렬하라.

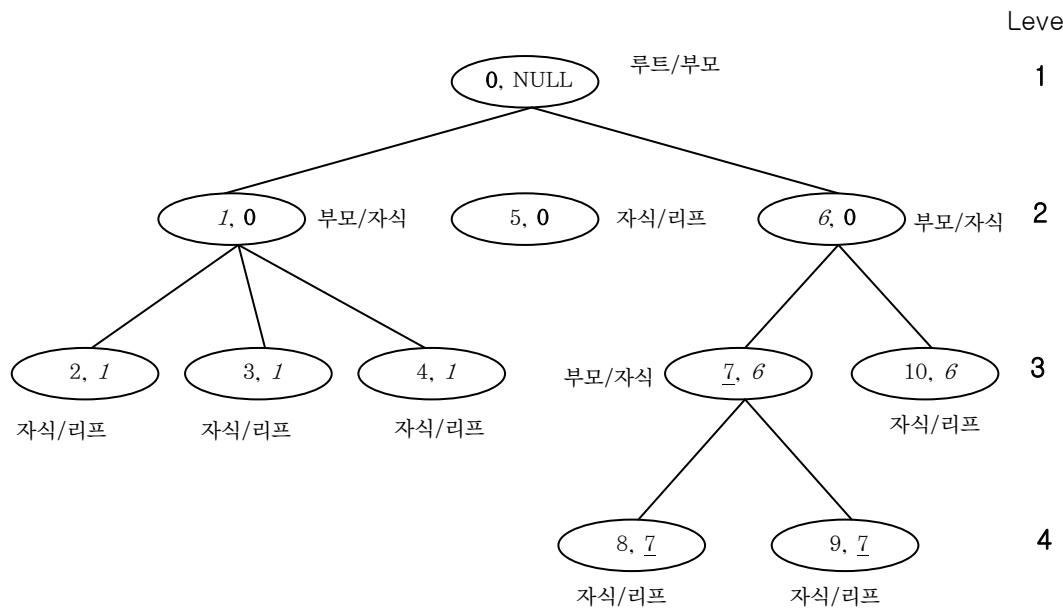
```
iSQL> SELECT ename, emp_job, join_date, salary
      FROM employee
     WHERE salary < 1000000
     ORDER BY 4 DESC;
    ENAME           EMP_JOB        JOIN_DATE       SALARY
    -----          -----          -----          -----
    KWKIM            CEO            980000
    HJMIN           MANAGER        2000/01/24 00:00:00  500000
2 rows selected.
```

## Hierachical query 사용 검색

〈질의〉 id 열의 값이 0 인 행을 루트로 하는 행들을 얻기 위한

계층적 질의문은 다음과 같다.

```
iSQL> CREATE TABLE hier_order(id INTEGER, parent INTEGER);
Create success.
iSQL> INSERT INTO hier_order VALUES(0, NULL);
1 row inserted.
iSQL> INSERT INTO hier_order VALUES(1, 0);
1 row inserted.
iSQL> INSERT INTO hier_order VALUES(2, 1);
1 row inserted.
iSQL> INSERT INTO hier_order VALUES(3, 1);
1 row inserted.
iSQL> INSERT INTO hier_order VALUES(4, 1);
1 row inserted.
iSQL> INSERT INTO hier_order VALUES(5, 0);
1 row inserted.
iSQL> INSERT INTO hier_order VALUES(6, 0);
1 row inserted.
iSQL> INSERT INTO hier_order VALUES(7, 6);
1 row inserted.
iSQL> INSERT INTO hier_order VALUES(8, 7);
1 row inserted.
iSQL> INSERT INTO hier_order VALUES(9, 7);
1 row inserted.
iSQL> INSERT INTO hier_order VALUES(10, 6);
1 row inserted.
iSQL> SELECT ID, parent, LEVEL
      FROM hier_order START WITH id = 0 CONNECT BY PRIOR id = parent
      ORDER BY level;
ID          PARENT      LEVEL
-----  -----
0                  1
6                  0          2
5                  0          2
1                  0          2
10                 6          3
4                  1          3
7                  6          3
3                  1          3
2                  1          3
8                  7          4
9                  7          4
11 rows selected.
```



[그림 4-1] 계층적 구조 데이터

〈질의〉 START WITH 절을 생략하여 테이블 내의 모든 행을 루트 행으로 사용하고 PRIOR id = parent 조건을 만족하는 질의이다.

```
iSQL> SELECT id, parent, level
  FROM hier_order CONNECT BY PRIOR id = parent ORDER BY id;
    ID      PARENT      LEVEL
-----
```

0		1
1	0	1
1	0	2
2	1	1
2	1	3
2	1	2
3	1	2
3	1	1
3	1	3
4	1	1
4	1	2
4	1	3
5	0	1
5	0	2
6	0	2
6	0	1
7	6	1
7	6	2
7	6	3
8	7	3
8	7	1
8	7	2
8	7	4

```

9      7      2
9      7      3
9      7      4
9      7      1
10     6      1
10     6      2
10     6      3
30 rows selected.

```

〈질의〉 다음은 질의 검색 시 LOOP 이 생성된 행을 제외한 결과  
집합을 얻기 위하여 IGNORE LOOP 절을 사용한 계층적 질의문이다.

```

iSQL> CREATE TABLE triple(
    num INTEGER,
    tri INTEGER,
    PRIMARY KEY(num, tri));
Create success.
iSQL> CREATE OR REPLACE PROCEDURE proc_tri
AS
    v1 INTEGER;
BEGIN
    FOR v1 IN 1 .. 1000 LOOP
        INSERT INTO triple VALUES(v1, v1 * 3);
    END LOOP;
    INSERT INTO triple VALUES(1, 1);
END;
/
Create success.
iSQL> EXEC proc_tri;
Execute success.
iSQL> SELECT num, tri, level
    FROM triple
   WHERE num < 3001
     START WITH num = 1
     CONNECT BY PRIOR tri = num
     IGNORE LOOP;

```

NUM	TRI	LEVEL
1	1	1
1	3	2
3	9	3
9	27	4
27	81	5
81	243	6
243	729	7
729	2187	8
1	3	1
3	9	2
9	27	3
27	81	4
81	243	5
243	729	6

```
729      2187      7  
15 rows selected.
```

## GROUP BY 를 이용한 검색

〈질의〉 전 직원의 부서 번호별 급여 평균을 검색하라.

```
iSQL> SELECT dno, AVG(salary) AS avg_sal  
— salary average of each department  
FROM employee  
GROUP BY dno;  
DNO  AVG_SAL  
-----  
A001  2066666.67  
C001  1576666.67  
C002  1660000  
D001  2075750  
F001  1845000
```

6 rows selected.

\* SELECT 목록의 열 중 그룹 함수에 없는 열은 모두 GROUP BY 절에 포함되어야 한다.

\* 칼럼의 제목을 주거나, 다른 제목을 사용하고 싶으면 위의 AS avg\_sal 처럼 칼럼의 이름 뒤에 사용하고 싶은 제목을 적어주면 된다. 칼럼 제목을 만들 때의 AS 는 생략가능하다.

\* -- (하이픈 두개)가 오면 그 줄의 이후 부분은 모두 주석(comment) 으로 처리된다.

〈질의〉 각 부서내에서 각 직위에 대해 지급되는 급여 총액을 출력하라. (여러 열에 GROUP BY 절을 사용)

```
iSQL> SELECT dno, emp_job, COUNT(emp_job) num_emp, SUM(salary)  
sum_sal  
FROM employee  
GROUP BY dno, emp_job;  
DNO  EMP_JOB      NUM_EMP      SUM_SAL  
-----  
CEO          1  
C002 DESIGNER    2          3800000  
D001 ENGINEER    3          6300000  
.  
.  
.  
12 rows selected.
```

〈질의〉 평균 급여가 1500000 원을 넘는 부서의 평균 급여를 출력하라.

```
iSQL> SELECT dno, AVG(salary)  
FROM employee
```

```

        WHERE AVG(salary) > 1500000
        GROUP BY dno;
[ERR-31061: An aggregate function is not allowed here.
0003:  WHERE AVG(SALARY) > 1500000
          ^
          ^
      ]

```

\* HAVING 절로 그룹을 제한하여 위의 오류를 다음과 같이 수정할 수 있다.

```
iSQL> SELECT dno, AVG(salary)
   FROM employee
   GROUP BY dno
   HAVING AVG(salary) > 1500000;
      DNO    AVG(SALARY)
-----
A001 2066666.67
C001 1576666.67
C002 1660000
D001 2075750
F001 1845000
5 rows selected.
```

〈질의〉 3 개 이상 주문된 상품번호와 그 상품들의 총 수를 출력하라.

```
iSQL> SELECT gno, COUNT(*)
   FROM orders
   GROUP BY gno
   HAVING COUNT(*) > 2;
      GNO      COUNT
-----
A111100002 3
C111100001 4
D111100008 3
E111100012 3
4 rows selected.
```

〈질의〉 12 월 한 달 동안 2 개 이상 주문된 상품번호와 그 상품들의 평균 주문양을 평균 주문양 순서대로 출력하라.

```
iSQL> SELECT gno, AVG(qty) month_avg
   FROM orders
   WHERE order_date BETWEEN '01-Dec-2000' AND '31-Dec-2000'
   GROUP BY gno
   HAVING COUNT(*) > 1
   ORDER BY AVG(qty);
      GNO      MONTH_AVG
-----
A111100002 35
D111100003 300
D111100004 750
C111100001 1637.5
D111100010 1750
D111100002 1750
```

```
E111100012 4233.33333  
D111100008 5500  
8 rows selected.
```

## ORDER BY를 이용한 검색

〈질의〉 모든 사원의 이름, 부서 번호 및 급여를 표시하고 부서 번호를 기준으로 정렬한 후 급여를 기준으로 해서 내림차순으로 출력하라. (ORDER BY 목록의 순서가 정렬 순서임.)

```
iSQL> SELECT ename, dno, salary  
      FROM employee  
      ORDER BY dno, salary DESC;  
ENAME          DNO    SALARY  
-----  
YHBAE          A001  4000000  
HYCHOI         A001  1700000  
HJMIN          A001  500000  
MSKIM          C001  2750000
```

20 rows selected.

〈질의〉 다음은 모든 사원의 이름 및 급여를 표시하고 부서 번호를 기준으로 정렬한 후 급여를 기준으로 해서 내림차순으로 출력하는 질의이다. (SELECT 목록에 없는 열을 기준으로 정렬할 수 있다.)

```
iSQL> SELECT ename, salary  
      FROM employee  
      ORDER BY dno, salary DESC;  
ENAME          SALARY  
-----  
YHBAE          4000000  
HYCHOI         1700000  
HJMIN          500000  
MSKIM          2750000
```

20 rows selected.

## 연산자 사용 검색

〈질의〉 재고 상품의 이름, 각 제품의 재고 값을 출력하라.

```
iSQL> SELECT gname, (stock*price) stock_price  
      FROM goods;  
GNAME          STOCK_PRICE  
-----  
IM-300          78000000  
IM-310          9800000
```

NT-H5000 27924000

.

.

.

30 rows selected.

## 별명(alias\_name)을 사용한 검색

〈질의〉 부서 위치의 별명(지역명)을 지정하여 검색하라.

```
iSQL> SELECT dname, 'District Name', dep_location
      FROM department;
      DNAME          'District Name'  DEP_LOCATION
      -----
      응용기술팀      District Name  마포
      엔진개발팀      District Name  여의도
      마케팅팀        District Name  강남
      기획관리팀      District Name  강남
      영업팀          District Name  신촌
      5 rows selected.
```

## LIMIT을 사용한 검색

〈질의〉 EMPLOYEE 테이블에서 사원 이름을 3 번째 레코드 부터 5 명만 출력하라.

```
iSQL> SELECT ename Name FROM employee LIMIT 3, 5;
      NAME
      -----
      HSCHOI
      KSKIM
      SJKIM
      HYCHOI
      HJMIN
      5 rows selected.
```

〈질의〉 관리자 테이블에서 첫 번째 레코드에 해당하는 사원의 이름과 급여를 출력하라.

```
iSQL> CREATE TABLE manager(
      mgr_no INTEGER PRIMARY KEY,
      mname VARCHAR(20),
      address VARCHAR(60));
      Create success.
iSQL> INSERT INTO manager VALUES(7, 'HJMIN', '44-25 YoulDo-dong
      Youngdungpo-gu Seoul, Korea');
      1 row inserted.
iSQL> INSERT INTO manager VALUES(8, 'JDLEE', '3101 N. Wabash Ave.
      Brooklyn, NY');
      1 row inserted.
iSQL> INSERT INTO manager VALUES(12, 'MYLEE', '130 Gongpyeongno Jung-
      gu Daegu, Korea');
      1 row inserted.
```

```
iSQL> SELECT ename, salary
      FROM employee
      WHERE eno = (SELECT mgr_no FROM manager LIMIT 1);
    ENAME          SALARY
    -----
HJMIN           500000
1 row selected.
```

### FOR UPDATE를 사용한 검색 명령

Transaction A	Time Point	Transaction B
iSQL> AUTOCOMMIT OFF; Set autocommit off success. (request X lock on employee)	1	iSQL> AUTOCOMMIT OFF; Set autocommit off success.
iSQL> LOCK TABLE employee IN EXCLUSIVE MODE; Lock success.		
(acquire X lock on employee)		
iSQL> SELECT ename FROM employee WHERE eno = 15; ENAME		iSQL> SELECT ename FROM employee WHERE eno = 15 FOR UPDATE;
----- -- JHSEOUNG 1 row selected.	2	(request conflicts with the X lock already held by transaction A)
		wait wait wait
iSQL> UPDATE employee SET ENO = 30 WHERE eno = 15; 1 row updated.	3	
iSQL> SELECT ename FROM employee		

```

        WHERE eno = 30;
ENAME
-----
-- 
JHSEOUNG
1 row selected.
iSQL> COMMIT;                                4
Commit success.
5      (resume)
ENAME
-----
No rows selected.

```

## HINTS를 사용한 검색

Table Access Method Hints (full scan, index scan, index ascending order scan, index descending order scan, no index scan)

다음은 사원들 중 모든 여사원의 번호, 이름, 직업을 검색하는 질의이다.

〈질의〉 `SELECT ENO, ENAME, EMP_JOB FROM EMPLOYEE WHERE SEX = 'F';`

예를 들어, 많은 수의 사원들이 있는 사원 테이블의 성별(SEX) column에 인덱스가 생성되어 있고, 이 열은 'M' 또는 'F' 값을 가지고 있다고 하자. 만약, 남자 직원과 여자 직원의 비율이 같다면 full scan으로 질의를 검색하는 것이 index scan으로 검색하는 것보다 더 빠를 것이다. 그러나, 만약 여자 직원의 비율이 상대적으로 남자 직원보다 적다면, index scan이 full table scan 보다 빠를 것이다. 즉, 서로 다른 두 개의 값만을 가지고 있는 열들이 있을 때, 조건에 맞는 질의를 검색하기 위해, optimizer는 각각의 값이 행들의 50% 내에서 나타난다고 가정해 cost\_based 접근 방식으로서 index scan 보다 full table scan을 선택해 수행한다. 아래의 질의들에서 access 회수를 비교하면 각각 20과 4인 것을 확인 할 수 있다.

〈질의〉 성별이 여자인 사원 번호, 이름, 직업을 출력하라. (full scan 이용)

```

iSQL> SELECT /*+ FULL SCAN(employee) */ eno, ename, emp_job
      FROM employee WHERE sex = 'F';
ENO      ENAME          EMP_JOB
-----  -----
2        HJNO            DESIGNER
10       YHBAE           PROGRAMMER
12       MYLEE           SALESMAN

```

```
16          JHCHOI          DESIGNER
4 rows selected.
```

---

```
PROJECT ( COLUMN_COUNT: 3, TUPLE_SIZE: 43 )
SCAN ( TABLE: EMPLOYEE, FULL SCAN, ACCESS: 20, SELF_ID: 2 )
```

---

〈질의〉 성별이 여자인 사원 번호, 이름, 직업을 출력하라. (index 이용)

```
iSQL> CREATE INDEX sex_index ON employee(sex);
Create success.
iSQL> SELECT /*+ INDEX(employee, sex_INDEX) use sex_index because
       there are few female employees */ eno, ename, emp_job
      FROM employee
     WHERE sex = 'F';
ENO        ENAME           EMP_JOB
-----  -----
2          HJNO            DESIGNER
10         YHBAE           PROGRAMMER
12         MYLEE           SALESMAN
16         JHCHOI          DESIGNER
4 rows selected.
```

---

```
PROJECT ( COLUMN_COUNT: 3, TUPLE_SIZE: 43 )
SCAN ( TABLE: EMPLOYEE, INDEX: SEX_INDEX, ACCESS: 4, SELF_ID: 2 )
```

---

```
iSQL> SELECT /*+ INDEX ASC ( t1, idx1, idx2, idx3 ) INDEX DESC ( t1, idx4 )
       */
       * FROM t1 WHERE t1.i1 = 1;
iSQL> SELECT /*+ INDEX DESC ( t1, idx1, idx2, idx3 ) */ * FROM t1 WHERE
       t1.i1
       = 1;
iSQL> SELECT /*+ NO INDEX ( t1, idx1, idx2, idx3 ) */ * FROM t1 WHERE t1.i1
       =
       1;
```

〈질의〉 1 월에서 3 월까지의 주문테이블을 상품번호칼럼인덱스로 검색하라.

```
create view orders as
select ono, order_date, eno, cno, gno, qty from orders_01
union all
select ono, order_date, eno, cno, gno, qty from orders_02
union all
select ono, order_date, eno, cno, gno, qty from orders_03;
create index order1_gno on orders_01(gno);
create index order2_gno on orders_02(gno);
create index order3_gno on orders_03(gno);
iSQL> select /*+ index( orders,
       orders1_gno, orders2_gno, orders3_gno ) */
```

```

2      ONO, GNO, QTY
3  from orders;
ONO          GNO          QTY
_____
11290007      A111100002 70
11290011      E111100001 1000
11290100      E111100001 500
12300002      C111100001 300
12300001      D111100004 1000
12100277      D111100008 2500
12300003      E111100002 900
12310001      A111100002 50
12300011      C111100001 1000
12300013      C111100001 5000
12310002      D111100008 10000
12310003      E111100009 1500
12300012      E111100012 1300
12300014      F111100001 800
14 rows selected.

PROJECT ( COLUMN_COUNT: 3, TUPLE_SIZE: 24 )
VIEW ( ORDERS, ACCESS: 14, SELF_ID: 6 )
PROJECT ( COLUMN_COUNT: 6, TUPLE_SIZE: 48 )
VIEW ( ACCESS: 14, SELF_ID: 5 )
BAG-UNION
PROJECT ( COLUMN_COUNT: 6, TUPLE_SIZE: 48 )
SCAN ( TABLE: ORDERS_01, INDEX: ORDERS1_GNO, ACCESS: ,
SELF_ID: 0 )
PROJECT ( COLUMN_COUNT: 6, TUPLE_SIZE: 48 )
SCAN ( TABLE: ORDERS_02, INDEX: ORDERS2_GNO, ACCESS: 4,
SELF_ID: 1 )
PROJECT ( COLUMN_COUNT: 6, TUPLE_SIZE: 48 )
SCAN ( TABLE: ORDERS_03, INDEX: ORDERS3_GNO, ACCESS: 7,
SELF_ID: 4 )

```

### Join Order Hints (ordered, optimized)

〈질의〉 주문된 상품을 담당하고 있는 사원 번호, 이름과 담당 고객의 이름을 출력하라. (employee 테이블과 customer 테이블을 조인하고, 그 결과를 orders 테이블과 조인)

```
iSQL> SELECT /*+ ORDERED */ DISTINCT o.eno, e.ename, c cname
  FROM employee e, customer c, orders o
 WHERE e.eno = o.eno AND o.cno = c.cno;
ENO          ENAME          CNAME
_____
12          MYLEE          YSKIM
12          MYLEE          DJKIM
12          MYLEE          BSYOUN
12          MYLEE          JHCHOI
```

```

12      MYLEE          KSKIM
12      MYLEE          DHCHO
12      MYLEE          JHKIM
19      KMKIM          CHLEE
19      KMKIM          JHPARK
19      KMKIM          BSYOUN
19      KMKIM          LSPARK
19      KMKIM          DKKIM
19      KMKIM          SMCHO
20      DIKIM          YSKIM
20      DIKIM          IJLEE
20      DIKIM          LSPARK
20      DIKIM          YDPARK
20      DIKIM          DHKIM
20      DIKIM          DKKIM
20      DIKIM          JHKIM
20      DIKIM          DJKIM
21 rows selected.

```

---

```

PROJECT ( COLUMN_COUNT: 3, TUPLE_SIZE: 48 )
DISTINCT ( ITEM_SIZE: 40, ITEM_COUNT: 21, BUCKET_COUNT: 1024,
ACCESS: 21, SELF_ID: 4, REF_ID: 3 )
JOIN
JOIN
SCAN ( TABLE: EMPLOYEE E, FULL SCAN, ACCESS: 20, SELF_ID: 1 )
SCAN ( TABLE: CUSTOMER C, FULL SCAN, ACCESS: 400, SELF_ID: 2 )
SCAN ( TABLE: ORDERS O, FULL SCAN, ACCESS: 12000, SELF_ID: 3 )

```

---

〈질의〉 주문된 상품을 담당하고 있는 사원 번호, 이름과 담당 고객의 이름을 출력하라. (FROM 절의 순서에 상관없이 optimizer에 의해서 테이블 조인 순서를 실행)

```
iSQL> SELECT DISTINCT o.eno, e.ename, c cname
  FROM employee e, customer c, orders o
 WHERE e.eno = o.eno AND o.cno = c.cno;
```

ENO	ENAME	CNAME
19	KMKIM	CHLEE
12	MYLEE	YSKIM
20	DIKIM	YSKIM
12	MYLEE	DJKIM
19	KMKIM	JHPARK
19	KMKIM	BSYOUN
12	MYLEE	BSYOUN
20	DIKIM	IJLEE
12	MYLEE	JHCHOI
12	MYLEE	KSKIM
19	KMKIM	LSPARK
20	DIKIM	LSPARK
12	MYLEE	DHCHO
20	DIKIM	YDPARK

```

20      DIKIM          DHKIM
20      DIKIM          DKKIM
19      KMKIM          DKKIM
19      KMKIM          SMCHO
12      MYLEE          JHKIM
20      DIKIM          JHKIM
20      DIKIM          DJKIM
21 rows selected.

```

---

```

PROJECT ( COLUMN_COUNT: 3, TUPLE_SIZE: 48 )
DISTINCT ( ITEM_SIZE: 40, ITEM_COUNT: 21, BUCKET_COUNT: 1024,
ACCESS: 21, SELF_ID: 4, REF_ID: 1 )
JOIN
JOIN
SCAN ( TABLE: CUSTOMER C, FULL SCAN, ACCESS: 20, SELF_ID: 2 )
SCAN ( TABLE: ORDERS O, INDEX: ODR_IDX2, ACCESS: 30, SELF_ID: 3 )
SCAN ( TABLE: EMPLOYEE E, INDEX: __SYS_IDX_ID_366, ACCESS: 30,
SELF_ID: 1 )

```

---

### Optimizer Mode Hints (rule, cost)

```
iSQL> SELECT /*+ RULE */ * FROM t1, t2 WHERE t1.i1 = t2.i1;
iSQL> SELECT /*+ COST */ * FROM t1, t2 WHERE t1.i1 = t2.i1;
```

### Normal Form Hints (CNF, DNF)

```
iSQL> SELECT /*+ CNF */ * FROM t1 WHERE i1 = 1 OR i1 = 2;
iSQL> SELECT /*+ DNF */ * FROM t1 WHERE i1 = 1 OR i1 = 2;
```

### Join Method Hints (nested loop, hash, sort, sort merge)

```
iSQL> SELECT /*+ USE_NL (t1,t2) */ * FROM t1, t2 WHERE t1.i1 = t2.i1;
iSQL> SELECT /*+ USE_HASH (t1,t2) */ * FROM t1, t2 WHERE t1.i1 = t2.i1;
iSQL> SELECT /*+ USE_SORT (t1,t2) */ * FROM t1, t2 WHERE t1.i1 = t2.i1;
iSQL> SELECT /*+ USE_MERGE (t1,t2) */ * FROM t1, t2 WHERE t1.i1 = t2.i1;
```

### Hash Bucket Size Hints (hash bucket count, group bucket count, set bucket count)

```
iSQL> SELECT /*+ HASH BUCKET COUNT (20) */ DISTINCT * FROM t1;
iSQL> SELECT * FROM t1 GROUP BY i1, i2;
iSQL> SELECT /*+ GROUP BUCKET COUNT (20) */ * FROM t1 GROUP BY i1,
i2;
iSQL> SELECT * FROM t1 INTERSECT SELECT * FROM t2;
iSQL> SELECT /*+ SET BUCKET COUNT (20) */ * FROM t1 INTERSECT
SELECT * FROM t2;
```

### Push Predicate Hints 를 이용한 검색

〈질의〉 1 월에서 3 월까지의 주문테이블에서 한번의 주문수량이  
10000 개이상인 고객의 명단과 상품번호를 구하라.

```

create view orders as
select ono, order_date, eno, cno, gno, qty from orders_01
union all
select ono, order_date, eno, cno, gno, qty from orders_02
union all
select ono, order_date, eno, cno, gno, qty from orders_03;

```

```
iSQL> select /*+ PUSH_PRED(orders) */ cname, gno
      2   from customer, orders
      3  where customer.cno = orders.cno
      4  and orders.qty >= 10000;
```

CNAME	GNO
KSKIM	D111100008

1 row selected.

```

PROJECT ( COLUMN_COUNT: 2, TUPLE_SIZE: 34 )
JOIN
SCAN ( TABLE: CUSTOMER, FULL SCAN, ACCESS: 20, SELF_ID: 2 )
FILTER
[ FILTER ]
AND
OR
ORDERS.QTY >= 10000
VIEW ( ORDERS, ACCESS: 1, SELF_ID: 8 )
PROJECT ( COLUMN_COUNT: 6, TUPLE_SIZE: 48 )
VIEW ( ACCESS: 1, SELF_ID: 7 )
BAG-UNION
PROJECT ( COLUMN_COUNT: 6, TUPLE_SIZE: 48 )
SCAN ( TABLE: ORDERS_01, INDEX: ODR1_IDX2, ACCESS: 3,
SELF_ID: 3 )
[ VARIABLE KEY ]
OR
AND
[ FILTER ]
AND
OR
PROJECT ( COLUMN_COUNT: 6, TUPLE_SIZE: 48 )
SCAN ( TABLE: ORDERS_02, INDEX: ODR2_IDX2, ACCESS: 4,
SELF_ID: 4 )
[ VARIABLE KEY ]
OR
AND
[ FILTER ]
AND
OR
PROJECT ( COLUMN_COUNT: 6, TUPLE_SIZE: 48 )
SCAN ( TABLE: ORDERS_03, INDEX: ODR3_IDX2, ACCESS: 7,
SELF_ID: 6 )
[ VARIABLE KEY ]
OR

```

AND  
[ FILTER ]  
AND  
OR

---

## OUTER JOIN 을 이용한 검색 기능

〈질의〉 다음은 모든 부서에 대한 부서 번호와 사원 이름을 출력하라.  
(사원이 전혀 없는 부서 번호 B001 도 출력된다.)

```
iSQL> INSERT INTO department VALUES('B001', 'Quality Assurance',  
'Jonglo', 22);  
1 row inserted.  
iSQL> SELECT d.dno, e.ename  
FROM department d LEFT OUTER JOIN employee e ON d.dno = e.dno  
ORDER BY d.dno;  
DNO    ENAME  
  
A001  HYCHOI  
A001  HJMIN  
A001  YHBAE  
B001  
C001  MSKIM  
C001  KWKIM  
C001  JHSEOUNG  
C002  HJNO  
C002  KMLEE  
C002  JHCHOI  
C002  DIKIM  
C002  CHLEE  
D001  HSCHOI  
D001  KSKIM  
D001  SJKIM  
D001  JDLEE  
D001  KCJUNG  
F001  MYLEE  
F001  KMKIM  
F001  DIKIM  
20 rows selected.
```

〈질의〉 다음은 모든 부서에 대한 부서 번호와 사원 이름을 출력하라.  
(부서에 소속이 되어 있지 않은 CEO 도 출력된다.)

```
iSQL> SELECT d.dno, e.ename  
FROM department d RIGHT OUTER JOIN employee e ON d.dno = e.dno  
ORDER BY d.dno;  
DNO    ENAME  
  
A001  HJMIN  
A001  HYCHOI  
A001  YHBAE
```

```
C001 JHSEOUNG  
C001 KWKIM  
C001 MSKIM  
C002 HJNO  
C002 JHCHOI  
C002 DIKIM  
C002 CHLEE  
C002 KMLEE  
D001 SJKIM  
D001 KCJUNG  
D001 KSKIM  
D001 HSCHOI  
D001 JDLEE  
F001 KMKIM  
F001 DIKIM  
F001 MYLEE  
SWNO
```

20 rows selected.

〈질의〉 부서의 위치와 상품을 모아 놓은 장소가 같은 곳의 부서 번호,  
부서 이름, 상품 번호를 출력하라.

```
iSQL> INSERT INTO department VALUES(BYTE'F002', 'headquarters', 'CE0002',  
100);
```

1 row inserted.

```
iSQL> SELECT d.dno, d.dname, g.gno  
FROM department d FULL OUTER JOIN goods g  
ON d.dep_location = g.goods_location;
```

DNO	DNAME	GNO
A111100001		
A111100002		
B111100001		
C111100001		
C111100002		
D111100001		
D111100002		
D111100003		
D111100004		
D111100005		
D111100006		
D111100007		
D111100008		
D111100009		
D111100010		
D111100011		
E111100001		
E111100002		
E111100003		
E111100004		
F002 headquarters	E111100005	
	E111100006	

1 row inserted.

DNO	DNAME	GNO
A111100001		
A111100002		
B111100001		
C111100001		
C111100002		
D111100001		
D111100002		
D111100003		
D111100004		
D111100005		
D111100006		
D111100007		
D111100008		
D111100009		
D111100010		
D111100011		
E111100001		
E111100002		
E111100003		
E111100004		
F002 headquarters	E111100005	
	E111100006	

```

E111100007
E111100008
E111100009
E111100010
E111100011
E111100012
E111100013
F111100001

A001 응용기술팀
D001 엔진개발팀
C002 기획관리팀
C001 마케팅팀
F001 영업팀
B001 Quality Assurance
36 rows selected.

```

### In-line View 를 이용한 검색 기능

〈질의〉 해당 부서의 평균 급여보다 급여를 많이 받는 모든 사원의 이름, 급여, 부서 번호 및 그 부서의 평균 급여를 출력하라.

```
iSQL> SELECT e.ename, e.salary, e.dno, v1.salavg
      FROM employee e, (SELECT dno, AVG(salary) salavg FROM employee
      GROUP BY dno) v1
     WHERE e.dno = v1.dno
       AND e.salary > v1.salavg;

```

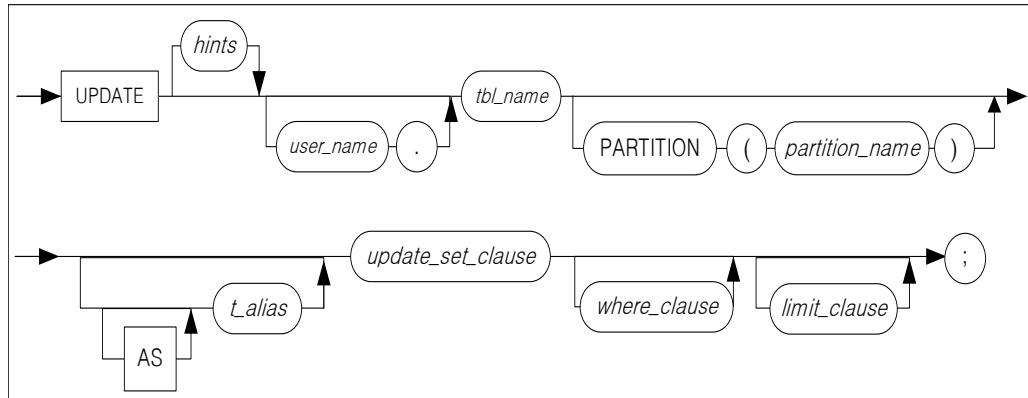
ENAME	SALARY	DNO	SALAVG
YHBAE	4000000	A001	2066666.67
MSKIM	2750000	C001	1576666.67
JHCHOI	2300000	C002	1660000
CHLEE	1900000	C002	1660000
SJKIM	2500000	D001	2075750
MYLEE	1890000	F001	1845000

6 rows selected.

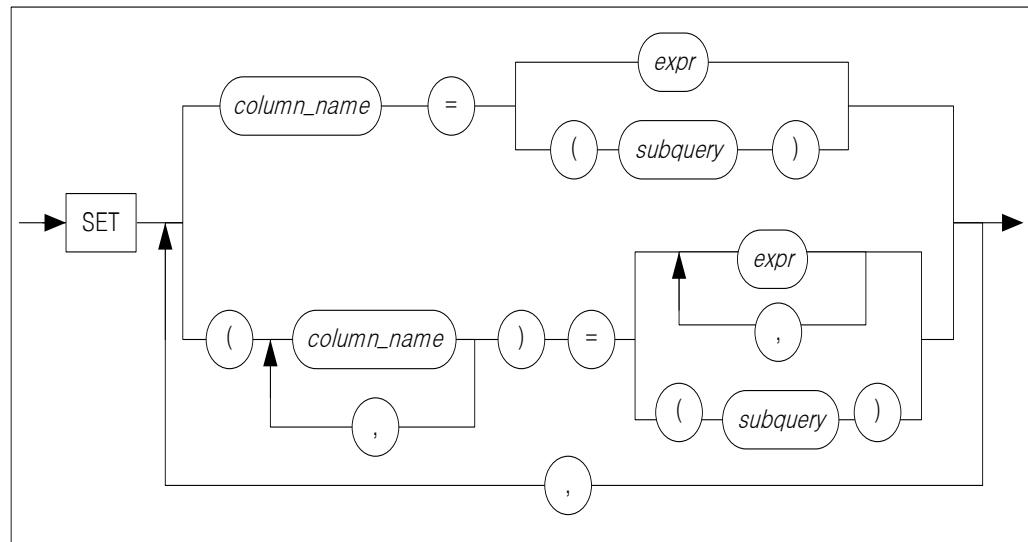
# UPDATE

## 구문

*update ::=*



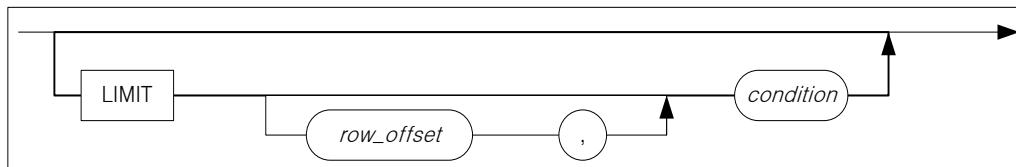
*update\_set\_clause ::=*



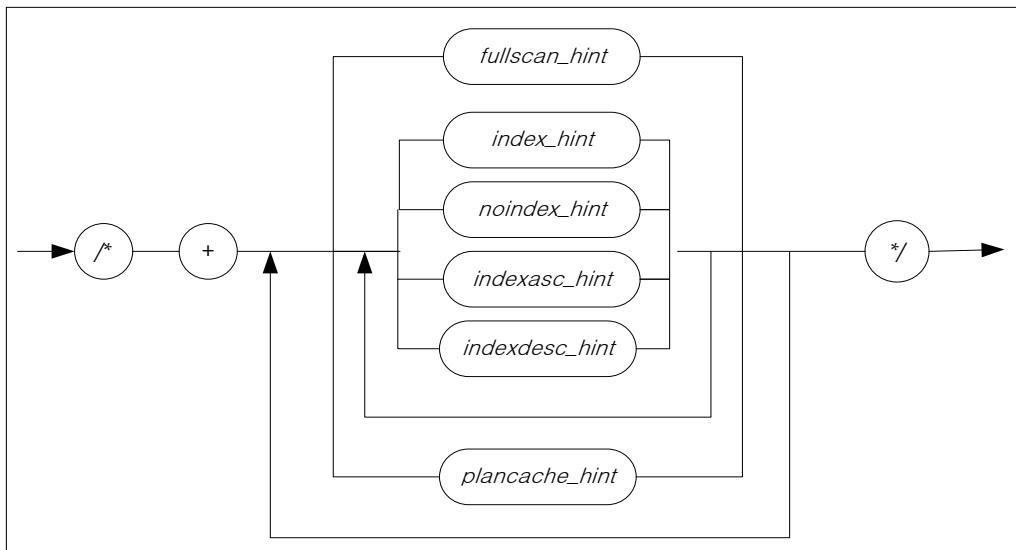
*where\_clause ::=*



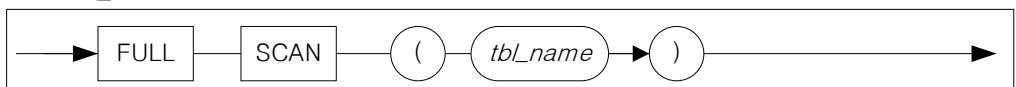
*limit\_clause ::=*



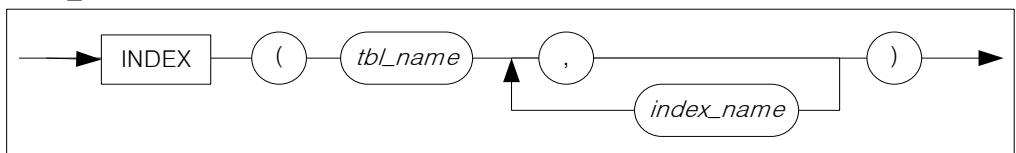
*hints ::=*



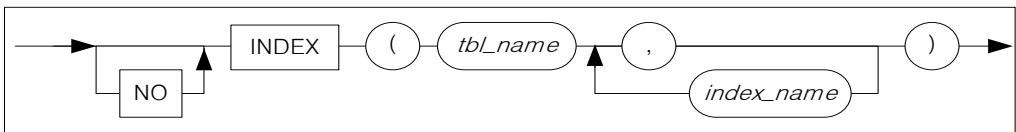
*fullscan\_hint ::=*



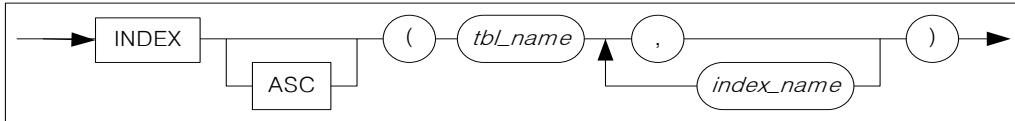
*index\_hint ::=*



*noindex\_hint ::=*



*indexasc\_hint ::=*



*indexdesc\_hint ::=*



## 전제 조건

테이블 또는 뷰의 기본 테이블 내의 값을 변경하려면 SYS 사용자이거나 UPDATE ANY TABLE 시스템 권한을 가져야 한다.

## 설명

조건을 만족하는 레코드를 찾아 명시한 칼럼들의 값을 변경한다.

파티션을 명시할 경우 해당 파티션에서 조건을 만족하는 레코드의 칼럼 값을 변경한다.

*user\_name*

레코드가 변경될 테이블의 소유자 이름을 명시한다. 생략하면 알터베이스는 현재 세션에 연결된 사용자의 스키마에 속한 것으로 간주한다.

*tbl\_name*

레코드가 변경될 테이블의 이름을 명시한다.

## HINTS 기능

알터베이스는 SQL 문 내에 주석을 달아 명령어들을 옵티마이저로 전달할 수 있다. 옵티마이저는 plan node를 선택하기 위해 이러한 힌트를 사용한다. +(plus sign)은 알터베이스가 주석을 힌트로 변환하게 하며 주석 기호 바로 다음에 위치해야 한다(no space).

- FULL SCAN: 테이블에 이용 가능한 인덱스가 존재하더라도 인덱스를 사용하지 않고 테이블 전체를 검색한다.

- INDEX: 해당 테이블에 대한 접근 방법으로 나열된 인덱스에 대해서 index scan을 사용한다.
- INDEX ASC: 해당 테이블에 대한 접근 방법으로 나열된 인덱스에 대해서 index scan을 사용하되, 오름 차순으로 탐색한다.
- INDEX DESC: 해당 테이블에 대한 접근 방법으로 나열된 인덱스에 대해서 index scan을 사용하되, 내림 차순으로 탐색한다.
- NO INDEX: 해당 테이블에 대한 접근 방법으로 나열된 index scan을 사용하지 않는다.
- hints 부분에 문법 오류가 있어도 무시하고 검색문은 실행된다. 그러나 hints의 기능은 수행하지 않는다.

SET 절에 부연 질의를 갖는 데이터 수정

- SET 절에 단지 한 열만을 명시하면, 부연 질의는 오로지 한 값만을 반환할 수 있다.
- SET 절에 여러 열들을 명시하면, 부연 질의는 명시된 열들의 개수 만큼 반환한다.
- SET 절에 부연 질의의 결과로서 반환되는 행이 없으면 널 값으로 할당된다.

TIMESTAMP 칼럼의 데이터 수정

- TIMESTAMP 칼럼은 UPDATE 수행 시 기본값으로 시스템 시간 값을 설정한다. 따라서 TIMESTAMP 칼럼의 데이터 수정 시 TIMESTAMP 칼럼 값을 명시하지 않을 경우 널이 아닌 시스템 시간 값으로 설정된다.
- TIMESTAMP 칼럼에 DEFAULT를 명시하면 시스템 시간 값으로 설정된다.

---

## 주의 사항

같은 칼럼을 두번 이상 사용할 수 없다.

파티셔닝 키 값이 수정되어 관련 데이터가 다른 파티션으로 이동해야 할 경우, 파티션드 테이블이 ENABLE ROW MOVEMENT 속성을 갖으면 이동이 가능하지만, 그렇지 않을 때에는 에러가 발생한다.

널 제약조건이 있는 칼럼에 널을 삽입하거나 널로 변경할 수 없다.

---

## 예제

### 단순 데이터 자료 수정

〈질의〉 이름이 JDLEE 인 직원의 월급을 2500000 으로 갱신하라.

```
iSQL> UPDATE employee  
SET salary = 2500000  
WHERE ename = 'KMLEE';  
1 row updated.
```

〈질의〉 전 직원의 월급을 7% 인상하라.

```
iSQL> UPDATE employee  
SET salary = salary * 1.07;  
20 rows updated.
```

## WHERE 절에 부연 질의를 갖는 데이터 수정

〈질의〉 MYLEE 직원이 받은 주문들의 수량을 50 개씩 빼라.

```
iSQL> UPDATE orders  
SET qty = qty - 50  
WHERE eno IN(  
    SELECT eno  
    FROM employee  
    WHERE ename = 'MYLEE');  
9 rows updated.
```

## 파티션드 테이블의 데이터 수정

```
iSQL> UPDATE T1 PARTITION(P1) SET l1 = 200;
```

## SET 절에 부연 질의를 갖는 데이터 수정

〈질의〉 다음 예제는 UPDATE 문의 구문상의 구조를 나타낸다.  
(UPDATE 문 내에 두 형태의 부연 질의로 열을 수정, 서로 관련된  
부연 질의, WHERE 절은 수정될 행을 제한)

```
iSQL> CREATE TABLE bonuses  
(eno INTEGER, bonus NUMBER(10, 2) DEFAULT 100, commission  
NUMBER(10, 2) DEFAULT 50);  
Create success.  
iSQL> INSERT INTO bonuses(eno)  
(SELECT e.eno FROM employee e, orders o  
WHERE e.eno = o.eno  
GROUP BY e.eno);  
3 rows inserted.  
iSQL> SELECT * FROM bonuses;  
BONUSES,ENO BONUSES,BONUS BONUSES,COMMISSION
```

---

12	100	50
19	100	50
20	100	50

3 rows selected.

```
iSQL> UPDATE bonuses  
SET eno = eno + 100, (bonus, commission) =
```

```

(SELECT 1.1 * AVG(bonus), 1.5 * AVG(commission) FROM bonuses)
WHERE eno IN
(SELECT eno
FROM orders
WHERE qty >= 10000);
1 row updated.
iSQL> SELECT * FROM bonuses;
BONUSES,ENO BONUSES,BONUS BONUSES,COMMISSION

```

---

12	100	50
20	100	50
119	110	75

3 rows selected.

\* 참고: WHERE 절에서 부연 질의의 결과가 널인 경우 어떠한 열도 영향을 받지 않으나 SET 절에서의 부연 질의 결과가 널인 경우는 널값으로 할당된다.

```

iSQL> UPDATE orders
SET qty = qty - 50
WHERE eno IN(
SELECT eno
FROM employee
WHERE ename = 'HHHHH');
No rows updated.
iSQL> UPDATE employee
SET dno =
(SELECT dno
FROM department
WHERE dep_location = 'LOCAL');
20 rows updated.
iSQL> WHERE eno = 12;
Syntax error.
iSQL> SELECT ename, dno
FROM employee
WHERE eno = 12;
ENAME          DNO

```

---

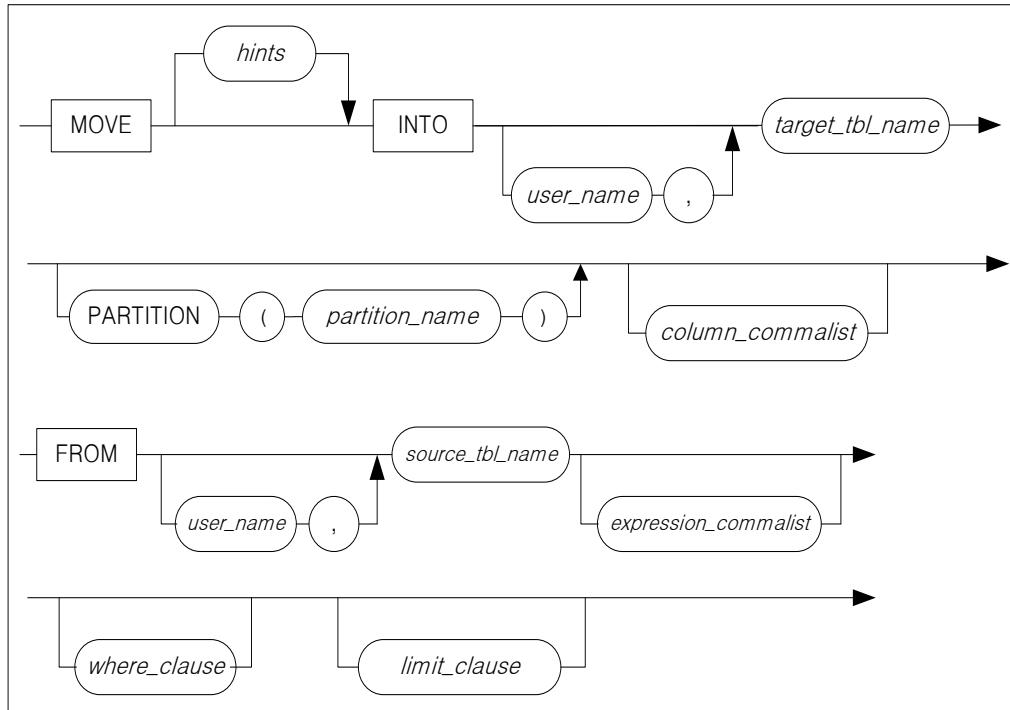
ENAME	DNO
MYLEE	

1 row selected.

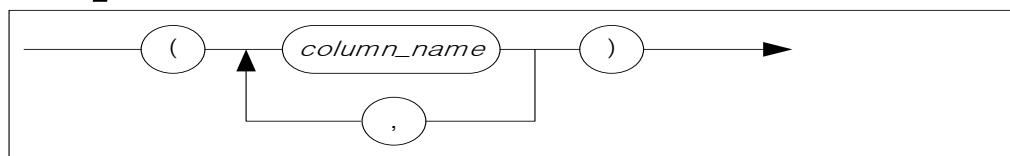
# MOVE

## 구문

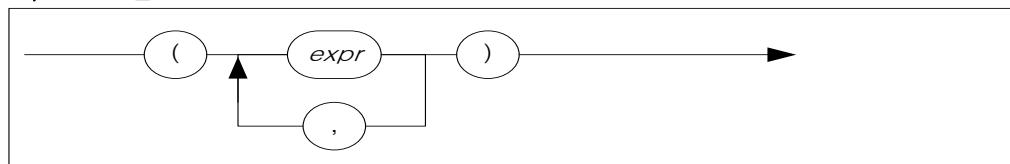
*move ::=*



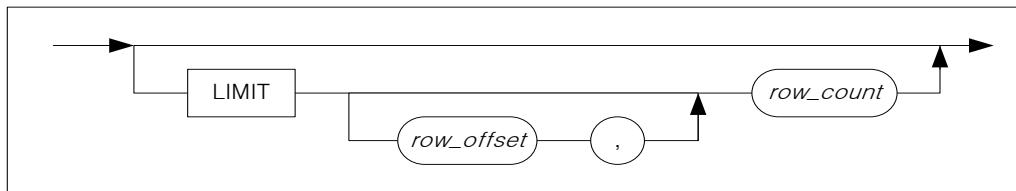
*column\_commalist ::=*



*expression\_commalist ::=*



*limit\_clause ::=*



## 전제 조건

테이블의 레코드를 이동(MOVE)하기 위해서는 테이블에 레코드를 삽입, 삭제할 수 있는 권한이 있어야 한다. 이동은 삽입과 삭제로 구분되기 때문이다.

테이블에 레코드를 삽입하기 위해 SYS 사용자이거나 INSERT ANY TABLE 시스템 권한을 가져야 하며, INTO 뒤의 테이블을 대상(target)으로 한다.

테이블의 레코드를 삭제하기 위해 SYS 사용자이거나 DELETE ANY TABLE 시스템 권한을 가진 사용자여야 하며, FROM 뒤의 테이블이 원천(source)에 해당된다.

## 설명

원천(source) 테이블에서 조건을 만족하는 레코드를 찾아 대상(target) 테이블로 이동한다. 또한 특정 파티션에 있는 데이터도 이동이 가능하다.

*hints*

from 절에 대한 hint를 제공한다. 이는 SELECT 구문의 hints와 동일하다.

*table\_name*

view나 메타 테이블이 아닌 순수한 table의 이름이다.

*column\_commalist*

테이블에 속하는 순수한 칼럼의 리스트이다.

*expression\_commalist*

expression으로 이루어진 리스트이다.

*where\_clause*

SELECT 구문의 where clause와 동일하다.

### *limit\_clause*

SELECT 구문의 limit\_clause 와 동일하다.

---

### 주의 사항

동일한 테이블에 관하여 MOVE 를 사용할 수 없다.

파티션을 지정할 경우 해당 파티션에 맞지 않는 값일 경우 입력할 수 없다.

---

### 예제

〈질의〉 T2 (I1, I2)에서 T2.I2=4 을 만족하는 레코드를 T1 (I1, I2)에 삽입하고 T2에서 삭제한다.

```
iSQL> MOVE INTO T1(I1, I2) FROM T2(I1, I2) WHERE T2.I2 = 4;
```

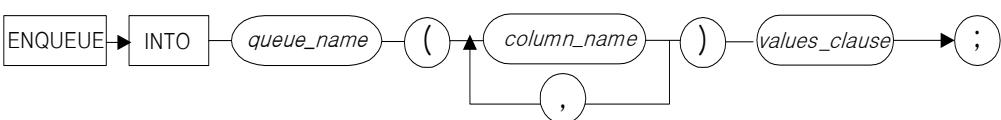
〈질의〉 T2 의 (I1, I2, I3)로 이루어진 레코드를 T1 에 삽입하고 T2에서 삭제한다. (T1 은 T2 의 (I1, I2, I3)과 대응되는 칼럼이 있어야 하며 칼럼 개수가 동일해야 함)

```
iSQL> MOVE INTO T1 FROM T2(I1, I2, I3);
```

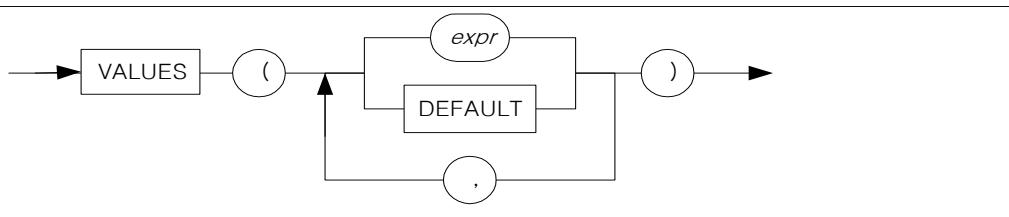
# ENQUEUE

## 구문

*enqueue ::=*



*values\_clause ::=*



## 설명

큐에 메시지를 삽입한다. ENQUEUE 구문은 INSERT 구문과 유사한 구조를 가지는데 INTO 절 이후에 반드시 하나 이상의 큐 칼럼 명을 나열해야 한다.

일반적인 경우에 사용자는 자신이 저장할 메시지만 지정하여 메시지를 입력하는데, 메시지를 구분하거나 분류하여 차별화 할 필요가 있을 때에는 Correlation id 를 명시적으로 지정하여 입력할 수 있다.

## 예제

〈질의〉 'This is a message'라는 메시지를 Q1이라는 메시지 큐에 입력하기.

```
ENQUEUE INTO Q1(message) VALUES ('This is a message');
```

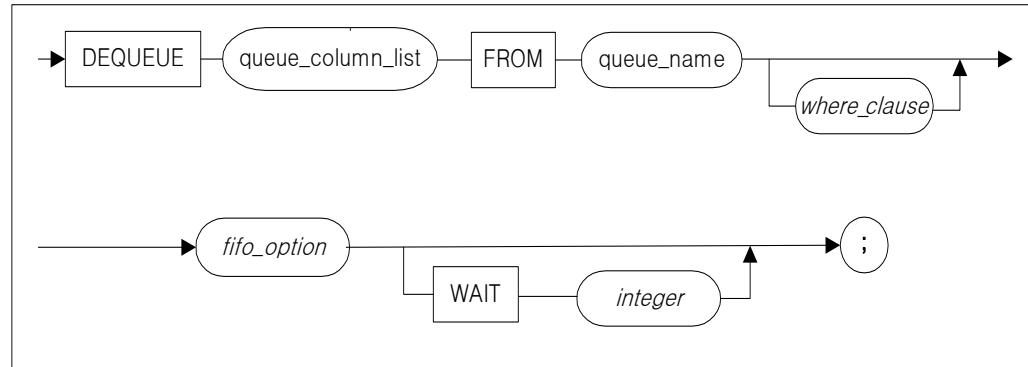
〈질의〉 'This is a message'라는 메시지를 237이라는 Correlation id로 Q1이라는 메시지 큐에 입력하기.

```
ENQUEUE INTO Q1(message,corrid) VALUES ('This is a message', 237);
```

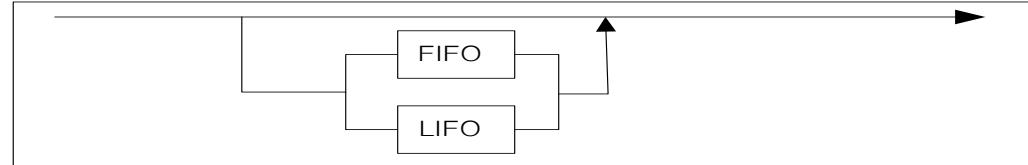
# DEQUEUE

## 구문

*dequeue ::=*



*fifo\_option ::=*



## 설명

DEQUEUE 구문은 *where\_clause* 절의 조건에 맞는 메시지를 얻어 오고 해당 메시지를 삭제한다.

*fifo\_option*

FIFO 옵션이 설정되어 있거나 옵션을 설정하지 않은 경우에는 조건에 맞는 메시지 중 가장 오래된 메시지를 얻어 오고, LIFO 옵션이 설정된 경우에는 가장 최신의 메시지를 얻어온다.

*WAIT integer*

DEQUEUE 문은 큐에 메시지가 없을 경우에 큐에 메시지가 ENQUEUE 될 때까지 대기하는데, 대기 시간은 WAIT 절에 설정된다. 대기 시간이 설정되지 않은 경우, 계속 대기한다.

---

## 주의사항

DEQUEUE 구문의 사용시에 다음과 같은 점에 주의해야 한다.

DEQUEUE 의 *queue\_column\_list* 에는 큐 테이블 내의 칼럼명만 지정 가능하다.

DEQUEUE 문은 SELECT 구문의 일부 특징을 가지고 있지만 DELETE 문의 FROM 절에는 단 하나의 QUEUE 이름만 지정이 가능하며, 두개 이상의 QUEUE 의 이름이나 테이블 이름이 오면 에러로 처리된다.

DEQUEUE 구문의 WHERE 절에는 부연질의(Subquery)가 올 수 없다.

---

## 예제

〈질의〉 메시지 큐 Q1에서 Correlation id 가 'From Seoul'인 메시지들을 모두 읽어 온다.

```
DEQUEUE MESSAGE, CORRID FROM Q1 WHERE CORRID='From Seoul';
```

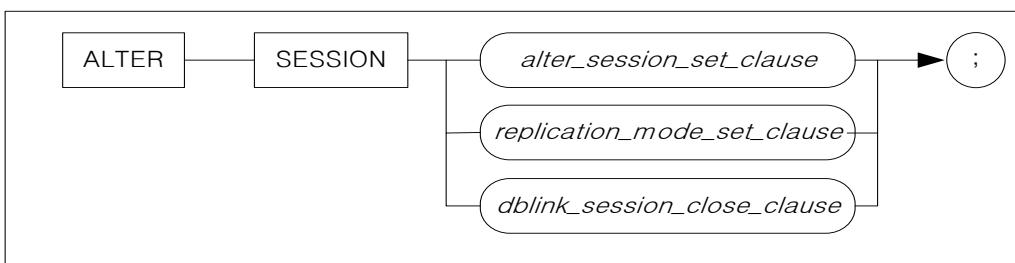
## 5. 데이터 제어어

이장에서는 SQL 문을 이용한 사용자 세션 제어와 트랜잭션 관리 방법 등에 대해서 설명하고 있다.

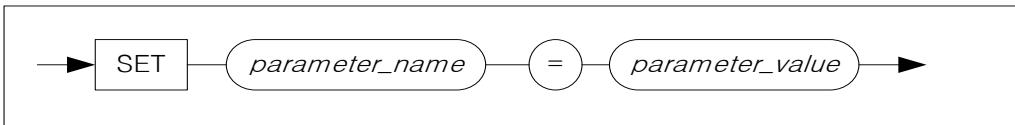
# ALTER SESSION

## 구문

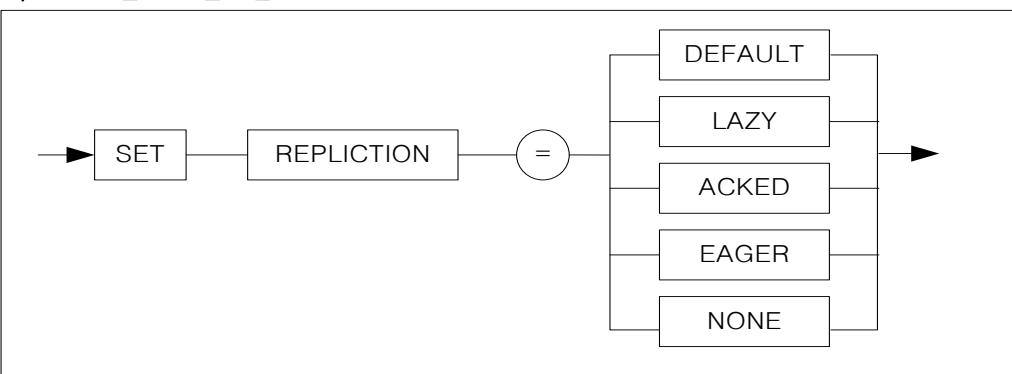
*alter\_session ::=*



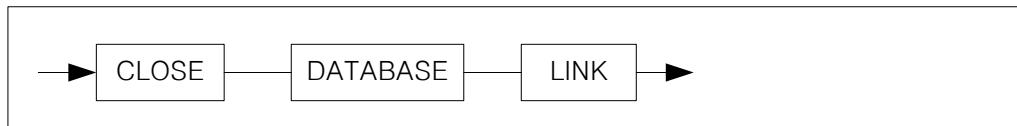
*alter\_session\_set\_clause ::=*



*replication\_mode\_set\_clause ::=*



*dblink\_session\_close\_clause ::=*



---

## 설명

현재 세션(Session)의 속성을 변경한다.

*alter\_session\_set\_clause*

*alter\_session\_set\_clause* 절의 parameter\_name과 parameter\_value에 관한 자세한 내용은 Starting User's Manual의 알티베이스 프로퍼티를 참조한다.

*replication\_mode\_set\_clause*

*replication\_mode\_set\_clause*는 현재 세션에서 수행하는 트랜잭션 이중화의 모드 속성을 설정하여 준다.

세션에서 DEFAULT는 이중화 모드를 설정하지 않은 것으로 모드를 설정하지 않으면 기본으로 설정된다. 그 외에는 현재 세션에서 수행하는 트랜잭션을 LAZY, ACKED, EAGER 모드로 설정한다. 그러나 NONE으로 지정할 경우 세션에서 수행하는 모든 DDL, DML, DCL 등이 이중화 대상에서 제외된다.

이중화 모드에 대하여 보다 자세한 내용은 *Replication User's Manual*을 참조한다.

*dblink\_session\_close\_clause*

사용자가 서버에 접속하면 서버에는 세션이 생성된다. 이 세션에서 데이터베이스 링크를 사용하면, 사용자에게 할당된 세션이 데이터베이스 링크 작업을 위하여 AltiLinker로의 데이터베이스 링크 세션을 생성하게 된다. 그리고 이 사용자의 세션이 종료될 때, 데이터베이스 링크 세션도 같이 종료된다. 그런데 데이터베이스 링크 작업을 한번 수행한 후, 이 사용자 세션을 종료하지 않은 채로 계속 유지하게 된다면, 불필요한 데이터베이스 링크 세션이 남아있게 된다.

이 때 이 명령을 사용하여 데이터베이스 링크 세션을 정리할 수 있다.

---

## 주의사항

ALTER SESSION SET REPLICATION 구문으로 이중화 모드를

EAGER로 설정하였을 때, 다음 사항을 주의해야 한다.

- 이중화 충돌(Replication Conflict)이 발생하여 커밋(Commit)이 실패했다는 메시지를 받게 되면, 사용자가 명시적으로 롤백(Rollback)을 해야 한다. 그렇지 않으면, 커밋을 할 수 없는 트랜잭션이 계속 진행되기 때문에 변경 작업을 반영할 수 없다.
- Autocommit 모드일 때, 트랜잭션이 이중화의 충돌로 실패할 경우, 자동적으로 롤백(Rollback)이 된다.
- 트랜잭션이 이중화 충돌에 의해 실패한 후 세션을 종료하는 경우, 자동적으로 롤백된다.
- ALTER SESSION CLOSE DATABASE LINK 명령은 사용자의 세션을 끊는 것이 아니라, 그 세션이 보유한 데이터베이스 링크 세션만 정리한다.

---

## 예제

〈질의〉 현재 세션의 이중화 모드를 변경한 후 DML을 수행하고, Replication Conflict 때문에 Commit이 실패하여 Rollback 한다.

```
iSQL> ALTER SESSION SET REPLICATION = EAGER;
Alter success.
iSQL> INSERT INTO TABLEA VALUES ( 2, 2 );
1 row inserted.
iSQL> COMMIT;
[ERR-110D2 : Transaction's commit was canceled by replication conflict.]
iSQL> ROLLBACK;
Rollback success.
```

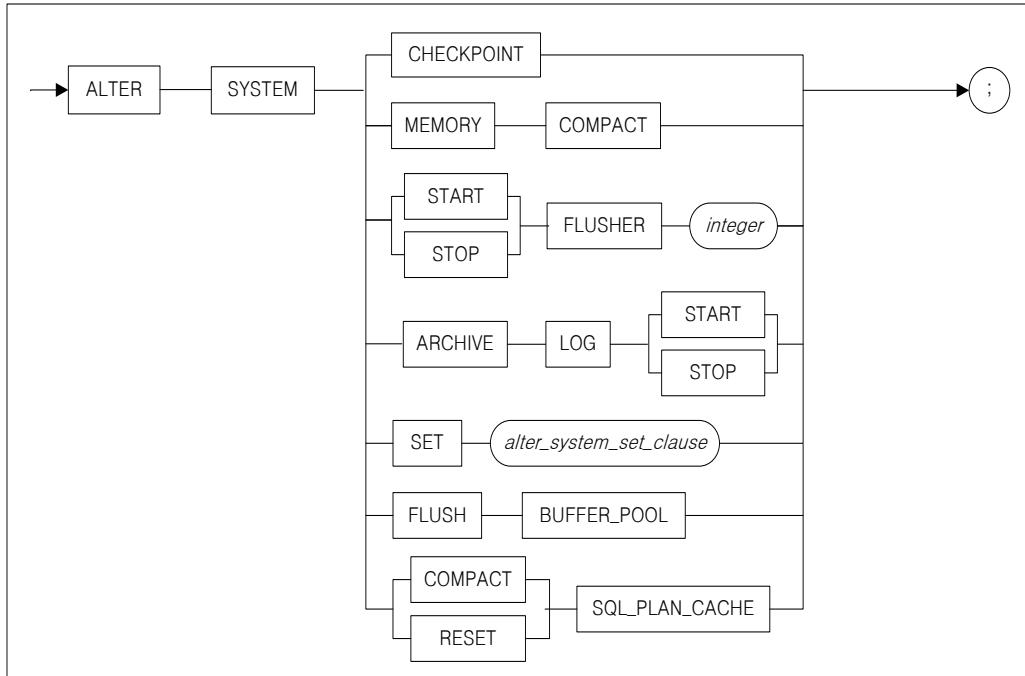
〈질의〉 현재 세션은 그냥두고 데이터베이스 링크 세션만 종료한다.

```
iSQL> ALTER SESSION CLOSE DATABASE LINK;
```

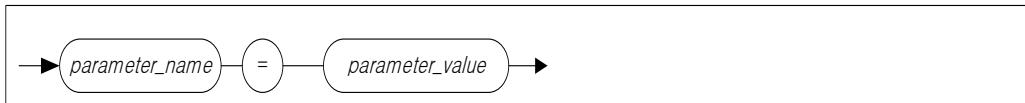
# ALTER SYSTEM

## 구문

*alter\_system ::=*



*alter\_system\_set\_clause ::=*



## 설명

알터베이스의 시스템 속성을 변경한다.

매개 변수(parameter)에 대한 자세한 내용은 *Starting User's Manual*의 알터베이스 프로퍼티를 참조한다.

START/STOP FLUSHER

알터베이스 플러셔를 구동하거나 정지시킨다.

ARCHIVE LOG START/STOP

START를 실행하면 아카이브 쓰레드가 시작되고, STOP하면 종료된다. 아카이브 모드일 경우에만 실행할 수 있다.

아카이브 모드의 실행 여부는 V\$LOG, V\$ARCHIVE 등에서 확인이 가능하다. 아카이브에 대한 자세한 내용은 [Admin User's Manual의 데이터베이스 모드](#)를 참조한다.

#### FLUSH BUFFER\_POOL

버퍼에 있는 모든 페이지를 디스크에서 내리고, 버퍼를 비운다.

이 구문은 sysdba 만 수행할 수 있으며, 사용할 때는 반드시 테스트 데이터베이스에서만 사용해야 한다. 구문을 수행할 경우 버퍼에 있는 모든 페이지가 삭제되기 때문에 다음에 수행되는 질의문은 처음 접근하는 모든 페이지에 대해 버퍼 미스(Buffer Miss)가 발생하기 때문이다.

---

## 예제

〈질의〉 플러셔 1 번을 중지시킨다.

```
iSQL> ALTER SYSTEM STOP FLUSHER 1;
```

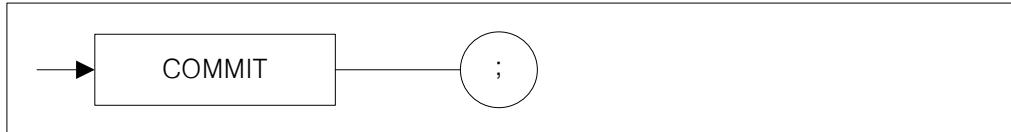
〈질의〉 아카이브 모드일 경우 아카이브 로그를 시작시킨다.

```
iSQL> ALTER SYSTEM ARCHIVE LOG START;
```

# COMMIT

## 구문

*commit ::=*



## 설명

현재의 트랜잭션을 명시적으로 반영한다.

자동반영(AUTOCOMMIT) 모드가 거짓(FALSE)일 때 사용할 수 있다.

## 주의 사항

자동반영 모드 시에 이 문장을 수행할 수 없다.

## 예제

```
iSQL> COMMIT; => 지금까지의 명령들을 모두 반영한다.  
Commit success.
```

# SAVEPOINT

## 구문

*savepoint ::=*



## 설명

저장점을 명시하여 트랜잭션 처리 과정에서 지금까지 실행된 트랜잭션을 임시로 저장한다. 즉, 트랜잭션 내에서 롤백하는 지점을 명시적으로 지정하는 명령어이다. AUTOCOMMIT 모드가 FALSE 일 때 유효하다.

## 주의 사항

자동반영(AUTOCOMMIT) 모드 일때 사용할 수 없다.

## 예제

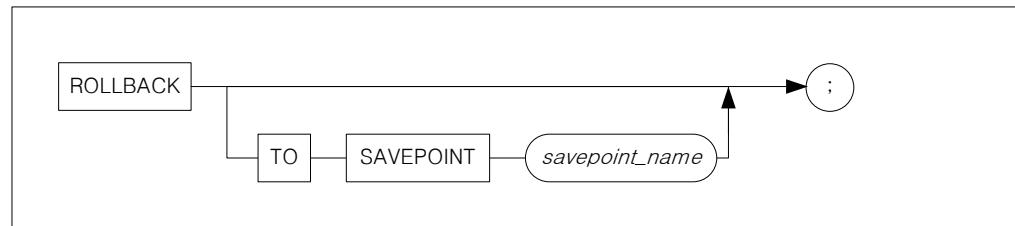
```
iSQL> AUTOCOMMIT OFF;
Set autocommit off success.
iSQL> CREATE TABLE savept(num INTEGER);
Create success.
iSQL> INSERT INTO savept VALUES(1);
1 row inserted.
iSQL> SAVEPOINT sp1;
Savepoint success.
=> 트랜잭션 처리 중 중간점을 표시하려면 SAVEPOINT 를
사용한다.(지금까지의 트랜잭션을 임시저장)
iSQL> INSERT INTO savept VALUES(2);
1 row inserted.
iSQL> SELECT * FROM savept;
SAVEPT.NUM
```

```
2
2 rows selected.
iSQL> ROLLBACK TO SAVEPOINT sp1;
Rollback success.
=> savepoint_name(sp1) 이 지정되어 있는 지점까지 트랜잭션을
    철회 시킨다.
iSQL> SELECT * FROM savept;
SAVEPT.NUM
-----
1
1 row selected.
iSQL> COMMIT;
Commit success.
```

# ROLLBACK

## 구문

*rollback ::=*



## 설명

현재의 트랜잭션을 명시적으로 반영한다. 현재의 트랜잭션을 모두 롤백하거나 저장점까지 부분 롤백한다. 현재의 트랜잭션을 롤백하기 위해 다음 명령어를 사용할 수 있다.

ROLLBACK;

또는

ROLLBACK TO SAVEPOINT *savepoint\_name*;

## 주의 사항

자동반영(AUTOCOMMIT) 모드 일때 사용할 수 없다.

## 예제

```
iSQL> AUTOCOMMIT OFF;
Set autocommit off success.
iSQL> UPDATE employee
SET salary = 2300000
WHERE ename = 'HSCHOI';
1 row updated.
iSQL> SAVEPOINT hachoi_sal;
Savepoint success.
iSQL> DELETE FROM employee
WHERE ename = 'KMKIM';
```

```

1 row deleted.
iSQL> SAVEPOINT kmkim_ret;
Savepoint success.
iSQL> INSERT INTO employee(eno, ename, salary, sex)
VALUES(21, 'MSJUNG', 3000000, 'F');
1 row inserted.
iSQL> SAVEPOINT msjung_join;
Savepoint success.
iSQL> UPDATE employee
SET salary = 2200000
WHERE ename = 'CHLEE';
1 row updated.
iSQL> SELECT ename, salary
FROM employee
WHERE ename = 'HSCHOI' OR ename = 'KMKIM'
OR ename = 'MSJUNG' OR ename = 'CHLEE';
ENAME          SALARY
-----
HSCHOI        2300000
MSJUNG        3000000
CHLEE         2200000
3 rows selected.

→ savepoint_name (msjung_join) 이 지정되어 있는 지점까지
트랜잭션을 철회 시킨다.

iSQL> SELECT ename, salary
FROM employee
WHERE ename = 'HSCHOI' OR ename = 'KMKIM'
OR ename = 'MSJUNG' OR ename = 'CHLEE';
ENAME          SALARY
-----
HSCHOI        2300000
MSJUNG        3000000
CHLEE         1900000
3 rows selected.

→ savepoint_name (kmkim_ret) 이 지정되어 있는 지점까지
트랜잭션을 철회 시킨다.

iSQL> SELECT ename, salary
FROM employee
WHERE ename = 'HSCHOI' OR ename = 'KMKIM'
OR ename = 'MSJUNG' OR ename = 'CHLEE';
ENAME          SALARY
-----
HSCHOI        2300000
CHLEE         1900000
2 rows selected.

[ERR-11016: ERR-11016: Unable to find the savepoint
iSQL> INSERT INTO employee(eno, ename, sex, join_date)
```

```
VALUES(22, 'MHJUNG', 'F', TO_DATE('2001-11-19 00:00:00', 'YYYY-MM-DD  
HH:MI:SS'));  
1 row inserted.  
iSQL> COMMIT;  
Commit success.
```

→ 첫 번째 DML 문 (DELETE 문)과 마지막 DML 문 (두 번째 INSERT 문)에 의해 수행된 모든 action 들이 명시적으로 반영된다. 모든 다른 SQL 문들은 COMMIT 전에 롤백되어졌으며 SAVEPOINT hachoi\_sal 은 더 이상 active 하지 않는다.

```
iSQL> SELECT eno, ename, salary FROM employee;  
ENO        ENAME          SALARY
```

ENO	ENAME	SALARY
3	HSCHOI	2300000
22	MHJUNG	0

.

.

.

20 rows selected.

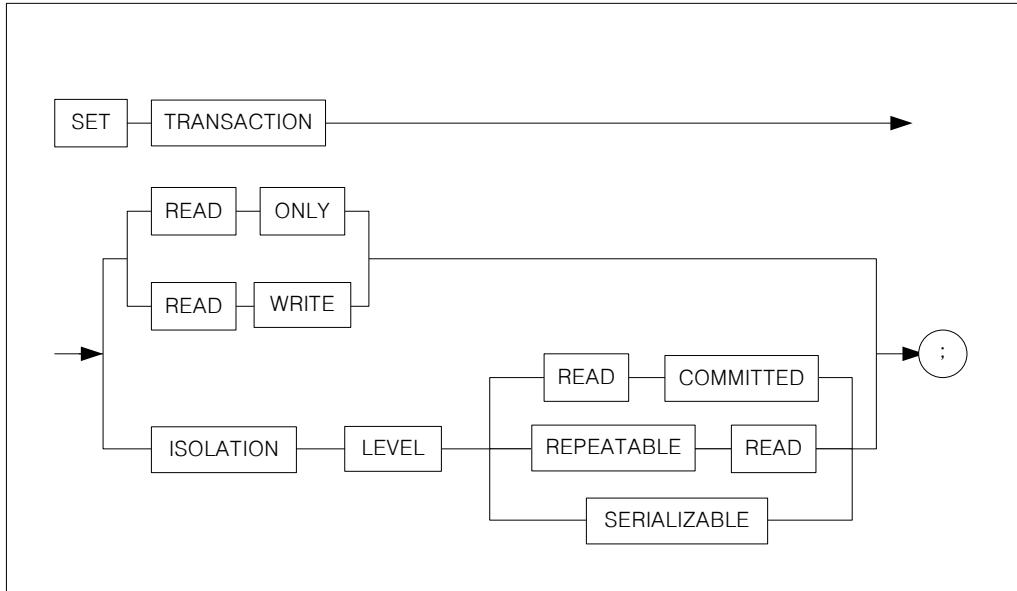
```
iSQL> COMMIT;
```

```
Commit success.
```

# SET TRANSACTION

## 구문

*set\_transaction ::=*



## 설명

현재 트랜잭션에 read only, read write 또는 isolation level 을 설정하기 위해 SET TRANSACTION 명령어를 사용할 수 있다. READ COMMITTED 와 SERIALIZABLE 모드는 행 수준 잠금과 다중버전 기법의 조합을 통하여 높은 수준의 데이터 일관성, 동시성 그리고 성능을 제공한다.

SET TRANSACTION 명령어에 의해 수행된 동작들은 다른 사용자들이나 다른 트랜잭션이 아닌 오로지 현재 트랜잭션에만 영향을 미친다.

현재 트랜잭션에 isolation level 을 설정하기위해 다음과 같은 속성을 선택, 사용할 수 있다.

READ COMMITTED

테이블 내의 commit 된 데이터에 대해 읽기를 허용하며, commit 되지 않은 데이터에 대해서는 이전 버전의 값을 읽을 수 있도록

동작한다. 알티베이스 서버에서 아무런 지정도 하지 않으면 read committed 로 동작한다.

#### REPEATABLE READ

트랜잭션에서 읽어간 데이터에 대해 잠금을 수행하여, 그 트랜잭션이 commit 되기 전까지 해당 데이터에 대한 다른 트랜잭션의 변경연산을 금지한다. 이러한 동작은 반복적으로 그 값을 다시 읽었을 때도 같은 값이 되돌려 질 것을 보장한다.

#### SERIALIZABLE

한번 SELECT 하여 가져간 모든 데이터에 공유잠금을 걸뿐만 아니라, 그 사이에 있는 모든 키 값에 대해서도 잠금을 걸게 된다. 즉, phantom 데이터에 대한 출현을 금지시켜, 트랜잭션의 독립성(isolation)을 보장한다.

SET TRANSACTION 명령어를 이용해 설정한 트랜잭션은 다음중 어느 하나가 발생할 때 끝난다.

- 사용자가 SAVEPOINT 절 없이 COMMIT 또는 ROLLBACK 문을 실행할 때
- 사용자가 DDL 문을 수행할 때
- 사용자가 알티베이스 접속을 해제
- 사용자 프로세스가 비정상적으로 종료되면 현재 트랜잭션은 데이터베이스에 반영되지 않는다.

---

### 주의 사항

현재 모드가 자동반영 모드일 경우에는 사용할 수 없다.

활성화된 트랜잭션이 있을 경우 사용할 수 없다.

---

### 예제

```
iSQL> AUTOCOMMIT OFF;  
Set autocommit off success.
```

```
iSQL> SET TRANSACTION ISOLATION LEVEL READ COMMITTED;  
Command execute success.
```

```
iSQL> SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;  
Command execute success.
```

Transaction A	Time Point	Transaction B
iSQL> AUTOCOMMIT OFF; Set autocommit off success.		iSQL> AUTOCOMMIT OFF; Set autocommit off success.
iSQL> SET TRANSACTION READ ONLY; Command execute success.	1	
iSQL> SELECT ename FROM employee WHERE eno = 20; ENAME ----- DIKIM 1 row selected.	2	
	3	iSQL> UPDATE EMPLOYEE SET ename = 'KBJUNG' WHERE eno = 20; 1 row updated.
iSQL> SELECT ename FROM employee WHERE eno = 20; ENAME ----- DIKIM 1 row selected.	4	
	5	iSQL> commit; Commit success.
iSQL> SELECT ename FROM employee WHERE eno = 20; ENAME ----- KBJUNG 1 row selected.	6	



# **Part III**



## 6. 집합 연산자

이장에서는 SQL 질의 실행결과 집합을 대상으로 사용할 수 있는  
집합 연산자들에 대해서 설명하고 있다.

## 합집합 (UNION)

### 구문

```
SELECT statement1 UNION SELECT statement2
```

### 설명

첫번째 검색문과 두번째 검색문의 결과를 모두 출력한다.

단, 동일한 검색 결과가 있을 경우 한번만 출력한다.

### 예제

〈질의〉 사원과 고객의 생일이 상반기(6 월 30 일 이전)인 사람들의 수를 출력하라.(중복된 결과는 한번만 출력한다.)

```
iSQL> SELECT COUNT(*) Num_birth  
      FROM employee  
      WHERE birth < Byte'0630'  
      UNION  
      SELECT COUNT(*)  
      FROM customer  
      WHERE birth < Byte'0630';  
      NUM_BIRTH
```

---

7

8

2 rows selected.

# 합집합 (UNION ALL)

## 구문

```
SELECT statement1 UNION ALL SELECT statement2
```

## 설명

첫번째 검색문과 두 번째 검색문의 결과를 모두 출력한다.

동일한 검색결과가 여러 번 있을 경우에도 모두 출력한다.

## 예제

\* UNION ALL 을 지정하면 중복된 데이터를 모두 가져오며, 지정하지 않고 UNION 만 사용하면 중복 데이터는 제거된다.

〈질의〉 사원의 생일이 상반기(6 월 30 일 이전)인 사번과 주문 테이블의 수량이 50 보다 작은 사번을 출력하라.

```
iSQL> SELECT eno  
      FROM employee  
      WHERE birth < Byte'0630'  
      UNION ALL  
      SELECT eno  
      FROM orders  
      WHERE qty < 50 ;  
      ENO
```

```
-----  
3  
7  
9  
10  
12  
15  
16  
20  
8 rows selected.
```

---

## 교집합

---

### 구문

```
SELECT statement1 INTERSECT SELECT statement2
```

### 설명

첫번째 검색문과 두 번째 검색문의 결과 중 동일한 검색 결과만 출력한다.

### 예제

〈질의〉 사원과 고객의 생일이 상반기(6 월 30 일 이전)인 사람들중 같은 이름을 가진 사람의 이름을 출력하라.

```
iSQL> SELECT ename
      FROM employee
     WHERE birth < Byte'0630'
INTERSECT
SELECT cname
      FROM customer
     WHERE birth < Byte'0630';
ENAME
-----
MYLEE
1 row selected.
```

# 차집합

## 구문

```
SELECT statement1 MINUS SELECT statement2
```

## 설명

첫번째 검색 결과에서 두 번째 검색 결과를 제외한 결과를 돌려준다.

## 예제

〈질의〉 주문된 상품들을 제외한 상품 번호를 출력하라.

```
iSQL> SELECT gno FROM goods  
MINUS  
SELECT gno FROM orders;  
GNO  
-----  
A111100001  
B111100001  
C111100002
```

.

.

.

14 rows selected.

## 우선순위 규칙

### 설명

계산 순서	연산자
1	모든 비교 연산자
2	NOT
3	AND
4	OR

괄호를 사용하여 우선순위 규칙보다 우선 적용할 수 있다.

### 예제

#### AND 연산자의 우선순위

〈질의〉 ENGINEER 이면서 급여가 1850000 원을 넘는 사원 또는 SALESMAN 인 사원의 이름, 직위, 급여를 출력하라.

```
iSQL> SELECT ename, emp_job, salary
   FROM employee
  WHERE emp_job = 'SALESMAN'
    OR emp_job = 'ENGINEER'
    AND salary >= 1850000;
      ENAME          EMP_JOB        SALARY
-----+
      HSCHOI        ENGINEER      2000000
      SJKIM         ENGINEER      2500000
      MYLEE        SALESMAN     1890000
      KMKIM        SALESMAN     1800000
      DIKIM        SALESMAN
5 rows selected.
```

#### 괄호를 사용하여 우선순위를 강제로 지정한다.

〈질의〉 ENGINEER 또는 SALESMAN 이면서 급여가 1850000 원을 넘는 사원의 이름, 직위, 급여를 출력하라.

```
iSQL> SELECT ename, emp_job, salary
   FROM employee
  WHERE (emp_job = 'SALESMAN'
    OR emp_job = 'ENGINEER')
    AND salary >= 1850000;
      ENAME          EMP_JOB        SALARY
-----+
      HSCHOI        ENGINEER      2000000
      SJKIM         ENGINEER      2500000
      MYLEE        SALESMAN     1890000
      KMKIM        SALESMAN     1800000
      DIKIM        SALESMAN
5 rows selected.
```

---

HSCHOI	ENGINEER	2000000
SJKIM	ENGINEER	2500000
MYLEE	SALESMAN	1890000

3 rows selected.



## 7. 함수

## 함수의 종류

다음은 알티베이스에서 제공하는 함수들에 대한 간단한 설명과 종류이다.

함수 범주	설명
그룹 함수	쿼리 결과 집합에 대한 단일 값을 생성한다.
숫자 함수	함수에 매개 변수로 제공되는 숫자 입력 값을 기반으로 하여 계산 작업을 수행하고 숫자 값을 반환한다.
문자 함수	문자열 입력 값에 대한 작업을 수행하고 문자열이나 숫자 값을 반환한다.
날짜 함수	날짜 및 시간 입력 값에 대한 작업을 수행하며 문자열, 숫자 또는 날짜와 시간 값을 반환한다.
변환 함수	입력 값(date, character 또는 number)에 대해 문자형, DATE 날짜 형식, 또는 NUMBER 데이터 형식으로 변환한다.
기타 함수	기타

함수 범주	종류
그룹 함수	AVG, COUNT, MAX, MIN, STDDEV, SUM, VARIANCE
숫자 함수	ABS, ACOS, ASIN, ATAN, ATAN2, CEIL, COS, COSH, EXP, FLOOR, LN, LOG, MOD, POWER, RANDOM, ROUND, SIGN, SIN, SINH, SQRT, TAN, TANH, TRUNC, BITAND, BITOR, BITXOR, BITNOT
문자 함수	숫자 값 반환 함수: ASCII, BINARY_LENGTH, CHAR_LENGTH(CHARACTER_LENGTH, LENGTH), DIGEST, INSTR(POSITION, INSTRB), OCTET_LENGTH(LENGTHB), SIZEOF 문자 값 반환 함수: CHR, CONCAT, DIGITS, INITCAP, LOWER, LPAD, LTRIM, REPLACE2, RPAD, RTRIM, SUBSTRB(SUBSTRING), TRANSLATE, TRIM, UPPER, REPLICATE, REVERSE_STR, STUFF
날짜 함수	ADD_MONTHS, DATEADD, DATEDIFF, DATENAME, EXTRACT(DATEPART), LAST_DAY, MONTHS_BETWEEN, NEXT_DAY, SYSDATE, SYSTIMESTAMP
변환 함수	BIN_TO_NUM, HEX_TO_NUM, OCT_TO_NUM, TO_BIN, TO_CHAR(datetime), TO_CHAR(number), TO_DATE, TO_HEX, TO_OCT, TO_NUMBER
암호화 함수	DESENCRYPT, DESDECRYPT
기타 함수	CASE2, CASE WHEN, DECODE, DUMP, GREATEST, LEAST, NVL, NVL2, SENDMSG, 중첩 함수

---

## 그룹 함수

쿼리 결과 집합에 대한 단일 값을 생성한다. 그룹 함수는 SELECT 목록이나 ORDER BY 또는 HAVING 절 안에 나타날 수 있다.

---

### AVG

#### 구문

**AVG ( [ALL | DISTINCT] expression)**

#### 설명

입력된 *expression*의 평균값을 구한다.

#### 예제

〈질의〉 상품 테이블의 가격 레코드 평균을 계산하여 출력하라.

```
iSQL> SELECT AVG(price) FROM goods;
AVG(PRICE)
-----
30406.173
1 row selected.
```

\* 칼럼의 값이 널인 것은 제외 된다.

---

### COUNT

#### 구문

**COUNT ( [ \* | [ALL | DISTINCT] expression ] )**

#### 설명

검색한 레코드의 수를 구한다.

#### 예제

〈질의〉 사원 테이블의 전체 레코드의 개수를 출력하라.

```
iSQL> SELECT COUNT(*) Rec_count FROM employee;
REC_COUNT
-----
```

```
20  
1 row selected.
```

〈질의〉 사원 테이블 생일 자료의 개수를 출력하라.

```
iSQL> SELECT COUNT(birth) Rec_count  
      FROM employee;  
REC_COUNT
```

```
13  
1 row selected.
```

\* 칼럼의 값이 널인 것은 카운트 되지 않는다.

---

## MAX

### 구문

**MAX ([ALL | DISTINCT] *expression*)**

### 설명

입력된 *expression*의 최대값을 구한다.

### 예제

〈질의〉 상품 테이블의 가격이 가장 비싼 값을 출력하라.

```
iSQL> SELECT MAX(price) FROM goods;  
MAX(PRICE)
```

```
100000  
1 row selected.
```

---

## MIN

### 구문

**MIN ([ALL | DISTINCT] *expression*)**

### 설명

입력된 *expression*의 최소값을 구한다.

### 예제

〈질의〉 상품 테이블의 가격이 가장 싼 값을 출력하라.

```
iSQL> SELECT MIN(price) FROM goods;
MIN(PRICE)
-----
966.99
1 row selected.
```

---

## STDDEV

### 구문

**STDDEV ([ALL | DISTINCT] expression)**

### 설명

STDDEV 는 *expression* 의 표준편차를 반환한다.

### 예제

〈질의〉 직원 테이블에서 급여의 표준편차를 구하라.

```
iSQL> SELECT STDDEV(salary) standard_deviation
      FROM employee;
STANDARD_DEVIATION
-----
797706.787
1 row selected.
```

---

## SUM

### 구문

**SUM ([ALL | DISTINCT] expression)**

### 설명

입력된 *expression* 의 합을 구한다.

### 예제

〈질의〉 상품 테이블의 모든 보관 수량의 합을 구하라.

```
iSQL> SELECT SUM(stock) FROM goods;
SUM(STOCK)
-----
379420
1 row selected.
```

---

## VARIANCE

### 구문

**VARIANCE ([ALL | DISTINCT] *expression*)**

### 설명

VARIANCE 는 *expression* 의 분산을 반환한다.

\* 시스템 호출에 의한 실수 연산의 오차 누적에 따라 반환 값의 오차가 발생할 수 있다.

### 예제

〈질의〉 직원 테이블에서 급여의 분산을 구하라.

```
iSQL> SELECT VARIANCE(salary) variance
      FROM employee;
VARIANCE
_____
6.3634E+11
```

## 숫자 함수

함수에 매개 변수로 제공되는 숫자 입력 값을 기반으로 하여 계산 작업을 수행하고 숫자 값을 반환한다.

### ABS

#### 구문

**ABS** (*number*)

#### 설명

입력된 값의 절대값을 돌려준다.

#### 예제

〈질의〉 세 종류의 숫자에 대한 ABS 함수의 결과를 출력하라.

```
iSQL> SELECT ABS(-1), ABS(0.0), ABS(1) FROM dual;
      ABS(-1)    ABS(0.0)    ABS(1)
      -----  -----
      1          0          1
1 row selected.
```

〈질의〉 상품 테이블에서 가장 비싼 품목의 가격과 가장 싼 품목의 가격 차이를 구하라.

```
iSQL> SELECT ABS(MIN(price) - MAX(price)) absolute_value FROM goods;
      ABSOLUTE_VALUE
      -----
      99033.01
1 row selected.
```

### ACOS

#### 구문

**ACOS** (*n*)

#### 설명

코사인 값이 지정한 FLOAT 또는 REAL 형식의 식인 라디안 단위의 각도를 0부터 pi 사이인 특정 각도 (라디안)로 반환한다. 아크코사인

이라고도 한다. 값이 이 범위에 속하지 않으면 0.000000 을 반환한다.  
=< ACOS(1)도 0

1 라디안 =  $180^\circ/\pi$

## 예제

〈질의〉

```
iSQL> SELECT ACOS(.3) Arc_Cosine FROM dual;
ARC_COSINE
_____
1.266104
1 row selected.
```

---

## ASIN

### 구문

**ASIN (n)**

### 설명

사인 값이 지정한 FLOAT 형식의 식인 라디안 단위의 각도를 –  
 $\pi/2$ 에서  $\pi/2$  사이인 라디안으로 반환한다. 아크사인이라고도 한다.  
*n* 값은 -1에서 1 사이인 FLOAT 형식의 식이다. 값이 이 범위에  
속하지 않으면 0.000000 을 반환한다.

## 예제

〈질의〉

```
iSQL> SELECT ASIN(.3) Arc_Sine FROM dual;
ARC_SINE
_____
0.304693
1 row selected.
```

---

## ATAN

### 구문

**ATAN (n)**

### 설명

탄젠트 값이 지정한 FLOAT 형식의 식인 라디안 단위의 각도를 – pi/2에서 pi/2 사이인 라디안으로 반환한다. 아크탄젠트라고도 한다.

## 예제

〈질의〉

```
iSQL> SELECT ATAN(.3) Arc_Tangent FROM dual;
ARC_TANGENT
_____
0.291457
1 row selected.
```

---

## ATAN2

### 구문

**ATAN2** (*n*, *m*)

### 설명

FLOAT 형식의 식인 *n*과 *m*의 아크탄젠트 값을 반환한다. 이것은 *n* / *m*의 아크탄젠트 값과 비슷하며 반환 값은 -pi에서 pi 사이인 라디안이다.

## 예제

〈질의〉

```
iSQL> SELECT ATAN2(.3, .2) Arc_Tangent2 FROM dual;
ARC_TANGENT2
_____
0.982794
1 row selected.
```

---

## CEIL

### 구문

**CEIL** (*number*)

### 설명

입력된 값 이상인 가장 작은 정수를 돌려준다.

## 예제

〈질의〉 지정한 숫자 식 (양수 99.9, 음수 -99.9) 이상인 최소 정수를 반환하라.

```
iSQL> SELECT CEIL(99.9), CEIL(-99.9) FROM dual;
CEIL(99.9)  CEIL(-99.9)
-----
100          -99
1 row selected.
```

〈질의〉 상품 테이블에서 가장 비싼 품목의 가격과 가장 싼 품목의 가격 차이를 구해 결과보다 큰 가장 작은 정수를 구하라.

```
iSQL> SELECT CEIL(ABS(MIN(price) – MAX(price))) Smallest_int FROM goods;
SMALLEST_INT
-----
99034
1 row selected.
```

---

## COS

### 구문

**COS (n)**

### 설명

FLOAT 형식의 식인 라디안 단위 각도의 삼각법 코사인을 라디안으로 반환하는 수치 연산 함수이다.

## 예제

〈질의〉

```
iSQL> SELECT COS(180 * 3.14159265359/180) Cos_of_180_degrees FROM
dual;
COS_OF_180_DEGREES
-----
-1
1 row selected.
```

---

## COSH

### 구문

**COSH (n)**

## 설명

입력한 식의 hyperbolic 코사인을 반환한다.

$$\text{COSH}(n) = (e^n + e^{-n})/2$$

## 예제

〈질의〉

```
iSQL> SELECT COSH(0) FROM dual;
```

```
COSH(0)
```

---

```
-----
```

```
1
```

```
1 row selected.
```

---

## EXP

### 구문

**EXP (n)**

## 설명

EXP 함수는 e 의 *n*승을 반환한다.  $e = 2.71828183\dots$

## 예제

〈질의〉

```
iSQL> SELECT EXP(2.4) FROM dual;
```

```
EXP(2.4)
```

---

```
-----
```

```
11.023176
```

```
1 row selected.
```

---

## FLOOR

### 구문

**FLOOR (*number*)**

## 설명

입력된 값 이하인 가장 큰 정수를 돌려준다

## 예제

〈질의〉 지정한 숫자 식 (양수 99.9, 음수 -99.9) 이하인 최대 정수를 반환하라.

```
iSQL> SELECT FLOOR(99.9), FLOOR(-99.9) FROM dual;  
FLOOR(99.9) FLOOR(-99.9)  
-----  
99      -100  
1 row selected.
```

〈질의〉 상품 테이블에서 가장 비싼 품목의 가격과 가장 싼 품목의 가격 차이를 구해 결과보다 작은 가장 큰 정수를 구하라.

```
iSQL> SELECT FLOOR(ABS(MIN(price) – MAX(price))) Largest_int FROM  
goods;  
LARGEST_INT  
-----  
99033  
1 row selected.
```

---

## LN

### 구문

**LN (n)**

### 설명

LN 함수는 0 보다 큰 수  $n$ 에 대하여,  $n$ 의 자연로그를 반환한다.

## 예제

〈질의〉

```
iSQL> SELECT LN(2.4) FROM dual;  
LN(2.4)  
-----  
0.875469  
1 row selected.
```

---

## LOG

### 구문

**LOG (m, n)**

## 설명

LOG 함수는 밑이  $m$ 인  $n$ 의 로그를 반환한다. 밑  $m$ 은 0과 1이 아닌 임의의 양수이어야 하고,  $n$ 은 임의의 양수이어야 한다.

## 예제

〈질의〉

```
iSQL> SELECT LOG(10, 100) FROM dual;
LOG(10, 100)
-----
2
1 row selected.
```

---

## MOD

### 구문

**MOD** (*number\_a*, *number\_b*)

### 설명

*number\_a* 를 *number\_b*로 나눈 나머지를 돌려준다.

## 예제

〈질의〉 10 을 3 으로 나눈 나머지를 구하라.

```
iSQL> SELECT MOD(10, 3) FROM dual;
MOD(10, 3)
-----
1
1 row selected.
```

〈질의〉 모든 사람의 급여의 합을 가장 적은 사람의 급여로 나눈 나머지를 구하라.

```
iSQL> SELECT MOD(SUM(salary), MIN(salary)) Remainder FROM employee;
REMAINDER
-----
223000
1 row selected.
```

---

## POWER

## 구문

**POWER (m, n)**

## 설명

POWER 함수는 *m* 의 *n*승을 반환한다. *m*과 *n*은 임의의 숫자일 수 있고, *m*이 음수라면 *n*은 정수라야 한다.

### 예제

〈질의〉

```
iSQL> SELECT POWER(3, 2) FROM dual;
POWER(3, 2)
_____
9
1 row selected.
```

---

## RANDOM

## 구문

**RANDOM (number)**

## 설명

의사 랜덤 정수형 값(pseudo random INTEGER value)을 반환한다. 반환 범위는 0 부터 INTEGER 형의 최대값이다.

*number*가 0 이 아닌 값이면 random seed 값을 주어진 *number*로 세팅한 다음, 의사 랜덤 정수를 돌려준다. 같은 seed 값을 가진 RANDOM 함수를 중복해서 부르면, 같은 값이 반환된다.

### 예제

〈질의〉

```
iSQL> SELECT RANDOM(0) FROM dual;
RANDOM(0)
_____
16838
1 row selected.
```

〈질의〉

```
iSQL> SELECT RANDOM(100) FROM dual;
RANDOM(100)
_____
12662
```

1 row selected.

---

## ROUND

### 구문

**ROUND ( *number\_a* [ , *number\_b* ] )**

### 설명

반올림 함수이다. *number\_a*를 소수점 아래 *number\_b* + 1 번째 자리에서 반올림하여 *number\_b* 번째 자리까지 반환한다.

### 예제

〈질의〉 다음 ROUND 함수로 표현한 두 개의 식의 결과를 출력하라:  
ROUND(123.9994, 3), ROUND(123.9995, 3)

```
iSQL> SELECT ROUND(123.9994, 3), ROUND(123.9995, 3) FROM dual;  
ROUND(123.9994, 3) ROUND(123.9995, 3)
```

---

```
123.999      124  
1 row selected.
```

〈질의〉 가장 싼 상품의 값을 정수값으로 반올림해서 출력하라.

```
iSQL> SELECT ROUND( MIN(price) ) FROM goods;  
ROUND( MIN(PRICE) )
```

---

```
967  
1 row selected.
```

\* 참고:

ROUND는 *number\_b*가 음수일 경우 데이터 형식에 상관없이 반올림한 값을 반환한다.

예제	결과
ROUND(748.58, -1)	750
ROUND(748.58, -2)	700
ROUND(748.58, -3)	1000

ROUND는 항상 값을 반환한다. *number\_b*가 음수이고 소수점 전의 자릿수보다 클 경우 ROUND는 0을 반환한다.

예제	결과
ROUND(748.58, -4)	0

---

## SIGN

### 구문

**SIGN (number)**

### 설명

*number*의 부호에 따라 다음의 결과를 돌려준다.

양수 일 때: 1, 음수일 때: -1, 그렇지 않으면 0을 돌려준다.

### 예제

〈질의〉

```
iSQL> SELECT SIGN(15), SIGN(0), SIGN(-15) FROM dual;
```

SIGN(15)	SIGN(0)	SIGN(-15)
1	0	-1

```
-----  
1      0      -1  
1 row selected.
```

〈질의〉 급여가 100 만원 보다 많으면 1, 적으면 -1, 그렇지 않으면 0을 출력하라.

```
iSQL> SELECT ename AS Employee_name, SIGN(salary-1000000) FROM  
employee;  
EMPLOYEE_NAME          SIGN(SALARY-1000000)
```

```
-----  
SWNO  
HJNO           1  
HSCHOI         1  
KSKIM          1  
SJKIM          1  
HYCHOI         1  
HJMIN         -1  
.  
.  
.  
20 rows selected.
```

---

## SIN

### 구문

**SIN (n)**

### 설명

특정 각도 (라디안)의 삼각 사인을 근사 숫자 (FLOAT) 식으로 반환합니다.

## 예제

〈질의〉

```
iSQL> SELECT SIN (30 * 3.14159265359/180) Sine_of_30_degrees FROM dual;
SINE_OF_30_DEGREES
-----
0.5
1 row selected.
```

---

## SINH

### 구문

**SINH (n)**

### 설명

입력한 식의 hyperbolic 사인을 반환한다.

$$\text{SINH}(n) = (\text{e}^n - \text{e}^{-n})/2$$

## 예제

〈질의〉

```
iSQL> SELECT SINH(1) Hyperbolic_sine_of_1 FROM dual;
HYPERBOLIC_SINE_OF_1
-----
1.175201
1 row selected.
```

---

## SQRT

### 구문

**SQRT (n)**

### 설명

SQRT 함수는  $n$ 의 제곱근을 반환한다.  $n$ 은 음수가 아니어야 한다.

## 예제

〈질의〉

```
iSQL> SELECT SQRT(10) FROM dual;  
SQRT(10)
```

```
-----  
3.162278  
1 row selected.
```

---

## TAN

### 구문

**TAN (n)**

### 설명

입력한 식의 탄젠트를 반환한다. 입력 인수는 FLOAT 또는 REAL 형식의 식이며 라디안 수로 해석된다.

### 예제

〈질의〉

```
iSQL> SELECT TAN (135 * 3.14159265359/180) Tangent_of_135_degrees  
FROM dual;  
TANGENT_OF_135_DEGREES
```

```
-----  
-1  
1 row selected.
```

---

## TANH

### 구문

**TANH (n)**

### 설명

입력한 식의 hyperbolic 탄젠트를 반환한다.

### 예제

〈질의〉

```
iSQL> SELECT TANH(.5) Hyperbolic_tangent_of_ FROM dual;  
HYPERBOLIC_TANGENT_OF_
```

```
0.462117  
1 row selected.
```

---

## TRUNC(number)

### 구문

```
TRUNC ( number_a [ , number_b ] )
```

### 설명

버림 함수이다.

*number\_a* 를 소수점아래 *number\_b* 번째 자리에서 버림한 float 타입 결과를 돌려준다.

*number\_b* 가 생략될 경우 0 으로 취급하여 소수점 아래 자리를 모두 버린다.

*number\_b* 가 음수일 경우 소수점 윗자리를 버려 0 으로 만든다.

### 예제

〈질의〉

```
iSQL> SELECT TRUNC(15.79, 1), TRUNC(15.79, -1) FROM dual;  
TRUNC(15.79, 1) TRUNC(15.79, -1)
```

---

```
15.7      10  
1 row selected.
```

〈질의〉 가장 싼 상품의 정수값을 출력하라.

```
iSQL> SELECT TRUNC(MIN(price)) FROM goods;  
TRUNC(MIN(PRICE))
```

---

```
966  
1 row selected.
```

---

## BITAND

### 구문

```
BITAND (bit_a, bit_b)
```

### 설명

bit\_a 와 bit\_a 의 AND 연산 결과를 돌려준다.

## 예제

〈질의〉

```
iSQL> SELECT TO_CHAR( BITAND( BIT'01010101', BIT'10101010' ) ) FROM
DUAL;
TO_CHAR( BITAND( BIT'01010101', BIT'1010
-----
00000000
1 row selected.
```

---

## BITOR

### 구문

**BITOR** (*bit\_a, bit\_b*)

### 설명

bit\_a 와 bit\_b 의 OR 연산 결과를 돌려준다.

## 예제

〈질의〉

```
iSQL> SELECT TO_CHAR( BITOR( BIT'01010101', BIT'10101010' ) ) FROM DUAL;
TO_CHAR( BITOR( BIT'01010101', BIT'10101
-----
11111111
```

---

## BITXOR

### 구문

**BITXOR** (*bit\_a, bit\_b*)

### 설명

bit\_a 와 bit\_b 의 XOR 연산 결과를 돌려준다.

## 예제

〈질의〉

```
iSQL> SELECT TO_CHAR( BITXOR( BIT'01010101', BIT'10101010' ) ) FROM
DUAL;
TO_CHAR( BITXOR( BIT'01010101', BIT'1010
_____
1111111
1 row selected.
```

---

## BITNOT

### 구문

**BITNOT** (*bit\_a*)

### 설명

*bit\_a* 의 NOT 연산 결과를 돌려준다.

### 예제

〈질의〉

```
iSQL> SELECT TO_CHAR( BITNOT( BIT'01010101' ) ) FROM DUAL;
TO_CHAR( BITNOT( BIT'01010101' ) )
_____
10101010
1 row selected.
```

---

## 문자 함수

문자열을 입력하면 문자 값이나 숫자 값을 반환한다.

### 숫자 값 반환 함수

ASCII, BINARY\_LENGTH, INSTR(POSITION, INSTRB),  
CHAR\_LENGTH(CHARACTER\_LENGTH, LENGTH),  
OCTET\_LENGTH(LENGTHB), SIZEOF

### 문자 값 반환 함수

CHR, CONCAT, DIGITS, INITCAP, LOWER, LPAD, LTRIM,  
REPLACE2, REPLICATE, REVERSE\_STR, RPAD, RTRIM, , STUFF,  
SUBSTRB, SUBSTRING, TRANSLATE, TRIM, UPPER

---

## ASCII

### 구문

**ASCII** (*char*)

### 설명

문자 식에서 가장 왼쪽 문자의 ASCII 코드 값을 반환한다.

### 예제

〈질의〉 문자 'A'를 ASCII 코드로 출력하라.

```
iSQL> SELECT ASCII('A') FROM dual;  
ASCII('A')
```

```
_____  
65  
1 row selected.
```

---

## BINARY\_LENGTH

### 구문

**BINARY\_LENGTH** (*char*)

### 설명

BLOB, Byte, NIBBLE 과 같은 이진 데이터 형의 길이를 반환한다.  
인자로 BLOB, Byte, NIBBLE 데이터 형의 값이 입력되어야 한다.

## 예제

〈질의〉 세 가지 이진 데이터 형의 값의 길이를 출력하라.

```
iSQL> CREATE TABLE T1 (I1 BLOB, I2 Byte(10), I3 NIBBLE(10) );
Create success.
```

```
iSQL> INSERT INTO T1 VALUES ( BLOB'3FD', Byte'123FD', NIBBLE'90BCD');
1 row inserted.
```

```
iSQL> SELECT BINARY_LENGTH (I1), BINARY_LENGTH (I2), BINARY_LENGTH
(I3) FROM T1;
BINARY_LENGTH (I1) BINARY_LENGTH (I2) BINARY_LENGTH (I3)
```

---

2	10	5
1 row selected.		

## CHAR\_LENGTH, CHARACTER\_LENGTH, LENGTH

### 구문

**CHAR\_LENGTH (char)**  
**CHARACTER\_LENGTH (char)**  
**LENGTH (char)**

### 설명

입력된 문자열(*char*)의 길이를 돌려준다.

한글 한글자는 2 바이트지만 길이는 1로 출력된다. 단, 알티베이스  
프로퍼티 파일의 국가설정 (NLS\_USE)을 영문(US7ASCII)으로  
설정했을 경우 한글 한글자의 길이는 2로 출력된다. 만일 국가설정을  
한글(KO16KSC5601)로 설정할 경우 한글 한글자도 1의 크기로  
표시된다.

## 예제

〈질의〉 Manager 의 주소 길이를 출력한다.

```
iSQL> CREATE TABLE manager(
mgr_no INTEGER PRIMARY KEY,
mname VARCHAR(20),
address VARCHAR(60));
```

Create success.

```
iSQL> INSERT INTO manager VALUES(1, 'JDLEE', '3101 N. Wabash Ave.  
Brooklyn, NY');  
1 row inserted.
```

```
iSQL> INSERT INTO manager VALUES(15, 'HJMIN', '서울 마포구 아현 1');  
1 row inserted.
```

```
iSQL> SELECT CHAR_LENGTH (address) FROM manager;  
CHAR_LENGTH (ADDRESS)
```

```
-----  
32  
18  
2 rows selected.
```

\* NLS\_USE = KO16KSC5601 인 경우

```
<결과>  
CHAR_LENGTH (ADDRESS)
```

```
-----  
32  
11  
2 rows selected.
```

---

## CHR

### 구문

**CHR** (integer)

### 설명

ASCII 코드값을 받아서 해당하는 문자로 변환하는 함수이다.

### 예제

<질의> ‘ALTIBASE’를 ASCII 코드값을 이용해서 출력하기

```
iSQL> SELECT CHR(65) || CHR(76) || CHR(84) || CHR(73) || CHR(66) ||  
CHR(65) || CHR(83) || CHR(69) mmdbms  
FROM dual;  
MMDBMS
```

```
-----  
ALTIBASE  
1 row selected.
```

<질의> 다음은 결과가 텍스트로 반환될 때, 줄을 바꿔서 고객의 이름, 전화번호, 우편번호, 주소 정보를 인쇄하기 위해 CHR(10)을

사용하는 예제이다.

```
iSQL> SELECT cname || CHR(10) || sex || '' || cus_job || CHR(10) || address  
cus_info  
FROM customer  
WHERE cno = '730828-1201145';  
CUS_INFO
```

---

```
CHLEE  
M ENGINEER  
부산 동구 수정 3  
1 row selected.
```

\* 참고:

제어 문자	값
탭	CHR(9)
줄 바꿈	CHR(10)
캐리지 리턴	CHR(13)

## CONCAT

### 구문

**CONCAT** (*char1, char2*)

### 설명

첫 번째 문자 값을 두 번째 문자 값에 연결한다.

연결 연산자 (||)와 동일하다.

### 예제

<질의>

```
iSQL> SELECT CONCAT(CONCAT(RTRIM(ename), "s job is ' ), emp_job ) Job  
FROM employee  
WHERE eno = 10;  
JOB
```

---

```
YHBAE's job is PROGRAMMER  
1 row selected.
```

## DIGITS

### 구문

## DIGITS (*number*)

### 설명

*number* 의 데이터 형에 따라서 길이를 다르게 하여 문자열로 반환한다.

SMALL INTEGER 일 때는 5 자리, INTEGER 일 때는 10 자리, BIG INTEGER 일 때는 19 자리로 문자열을 반환한다. 자리 수가 부족한 경우에는 0 으로 채운다.

### 예제

〈질의〉 세 가지 숫자 데이터 형의 종류에 따라서 문자열 길이를 맞추어 출력한다.

```
iSQL> CREATE TABLE T1 (I1 SMALLINT, I2 INTEGER, I3 BIGINT);
Create success.
```

```
iSQL> INSERT INTO T1 VALUES (357, 12, 5000);
1 row inserted.
```

```
iSQL> SELECT DIGITS(I1), DIGITS(I2), DIGITS(I3) FROM T1;
DIGITS(I1)  DIGITS(I2)  DIGITS(I3)
```

```
00357 0000000012 00000000000000005000
1 row selected.
```

---

## INITCAP

### 구문

## INITCAP (*char*)

### 설명

대문자 변환함수

주어진 문자열의 첫 문자를 대문자로 반환한다.

### 예제

〈질의〉 'the soap'의 첫 문자를 대문자로 출력하라.

```
iSQL> SELECT INITCAP ('the soap') Capital FROM dual;
CAPITAL
```

```
The soap
1 row selected.
```

## INSTR, POSITION

### 구문

**INSTR** (char, substring [, start [, occurrence]])  
**INSTRB** (char, substring [, start [, occurrence]])  
**POSITION** (char, substring [, start [, occurrence]])

### 설명

INSTR 함수는 char에 입력된 문자열에서 substring을 찾아서 그 위치를 반환한다. 탐색된 문자열이 없으면 0을 반환한다. INSTRB 함수는 지정한 문자의 위치를 문자 단위가 아닌 바이트 단위로 문자열을 반환한다.

start는 탐색을 시작하는 위치를 지정하며 기본값은 1이다. 음수이면 char의 맨 끝에서부터 찾기 시작하며, 0으로 지정하면 0을 반환한다. char의 길이보다 큰 값을 지정하면 에러를 발생한다.

occurrence는 탐색한 substring의 개수를 나타내며, 기본값은 1이다. occurrence가 1이면 처음 탐색된 substring의 위치를 반환한다. 0이거나, 탐색된 개수보다 더 큰 값으로 설정되면 함수의 결과는 0으로 나타난다. 음수로 설정하면 에러를 발생한다.

### 예제

〈질의〉 문자열 ‘CORPORATE FLOOR’에서 ‘OR’의 위치를, 앞에서 3 번째 문자부터 탐색을 시작하여 2 번째로 탐색된 문자열의 위치를 출력하라.

```
iSQL> SELECT INSTR ('CORPORATE FLOOR','OR', 3, 2) Instring FROM dual;
INSTRING
_____
14
1 row selected.
```

〈질의〉 문자열 ‘알티베이스 4 데이터베이스’에서 ‘베이’의 위치를 뒤에서 3 번째 문자부터 탐색을 시작하여 2 번째로 탐색된 문자열의 위치를 출력하라. (KO16KSC5601로 설정된 경우)

```
iSQL> SELECT INSTR ('알티베이스 4 데이터베이스','베이', 3, 2) Instring FROM
dual;
INSTRING
_____
11
1 row selected.
```

---

## LOWER

### 구문

**LOWER** (*char*)

### 설명

소문자 변환함수

입력된 문자를 소문자로 변환한 값을 돌려준다.

### 예제

〈질의〉 다음을 소문자로 변환하여 출력하라.

```
iSQL> SELECT LOWER('ONE PAGE PROPOSAL') Lowercase FROM dual;  
LOWERCASE
```

```
-----  
one page proposal  
1 row selected.
```

---

## LPAD

### 구문

**LPAD** (*char1, n [, char2]*)

### 설명

첫 번째 문자열 *char1*의 왼쪽부터 문자열 *char2*를 (연속적으로) 삽입하여 지정한 길이 *n* 만큼의 *char2*가 포함된 *char1*을 반환한다. *char2*의 기본값은 single blank이다. 만약, *char1*의 길이가 *n* 보다 길면 *char1*의 일부만(길이 *n*)을 반환한다. 단 *n*의 단위는 Byte가 아니라 문자 개수를 의미하므로 사용하는 언어에 따라 실제 문자열의 Byte 크기는 다를 수 있다.

### 예제

〈질의〉 “abc”라는 문자 식의 왼쪽부터 시작하여 “xyz”라는 문자열을 삽입하여 총 10 글자를 반환하는 예이다.

```
iSQL> SELECT LPAD('abc', 10, 'xyz') Lpad_ex FROM dual;  
LPAD_EX
```

```
-----  
xyzxyzxabc  
1 row selected.
```

---

## LTRIM

### 구문

**LTRIM** (*char1* [, *char2*])

### 설명

*char1*에 가장 왼쪽 문자부터 *char2*에 있는 문자들과 비교한다.

*char1*의 문자열이 *char2*의 문자와 같으면 삭제한다. 반복적으로 *char2*에 문자와 일치되는 문자가 없을 때까지 삭제한 결과를 출력한다.

*char2*는 기본적으로 single blank이며, 이 경우 *char1*의 앞(왼쪽)에 있는 빈칸을 모두 삭제한다.

### 예제

〈질의〉 문자열 'abaAabLEFT TRIM' 중 가장 왼쪽에 나타나 있는 a와 b를 제외한 문자열을 출력하라.

```
iSQL> SELECT LTRIM ('abaAabLEFT TRIM', 'ab') Ltrim_ex FROM dual;  
LTRIM_EX
```

```
-----  
AabLEFT TRIM  
1 row selected.
```

〈질의〉 고객의 주민등록번호 중에서 뒷자리만 출력하라.

```
iSQL> SELECT LTRIM (LTRIM(cno,'1234567890'), '-' ) FROM customer;  
LTRIM (LTRIM(CNO,'1234567890'), '-' )
```

```
-----  
1201145  
1345471  
1431202  
. . .  
20 rows selected.
```

---

## OCTET\_LENGTH, LENGTHB

### 구문

**OCTET\_LENGTH** (*char*)

### 설명

입력된 문자열(*char*)의 바이트 크기를 돌려준다.

항상 영문과 숫자는 1 바이트, 한글은 2 바이트를 사용한다.

## 예제

〈질의〉 '우리나라'에 할당된 길이를 출력하라. (K016KSC5601로 설정된 경우)

```
iSQL> SELECT OCTET_LENGTH('우리나라') FROM dual;  
OCTET_LENGTH('우리나라')
```

```
8  
1 row selected.
```

〈질의〉 Manager의 주소 길이를 출력하라.

```
iSQL> SELECT OCTET_LENGTH(address)  
FROM manager;  
OCTET_LENGTH(ADDRESS)
```

```
32  
18  
2 rows selected.
```

---

## REPLACE2

### 구문

**REPLACE2** (*char*, *string1*, [*string2*])

### 설명

*Char* 중 해당되는 *string1*은 *string2*로 치환을 시킨다. *string2*가 생략되거나 null인 경우 *string1*은 제거되고 공백 문자라면 *char* 중 해당되는 *string1*은 공백 문자로 치환된다. 만일 *string1*이 null이면 *char*가 반환된다.

TRANSLATE 함수가 대응하는 한 문자에 대해 하나씩 대체되는 것에 반해 REPLACE2 함수는 문자열을 제거시킬 뿐만 아니라 한 문자열을 다른 문자열로 대치한다.

## 예제

〈질의〉 DEPARTMENT 테이블 중에서 DNAME 을 [팀]이 아닌 [부문]으로 치환하라.

```
iSQL> SELECT REPLACE2(dname, '팀', '부문')  
FROM department;
```

```
REPLACE2(DNAME, '팀', '부문')
```

---

응용기술부문  
엔진개발부문  
마케팅부문  
기획관리부문  
영업부문  
5 rows selected.

〈질의〉 다음 예제는 abcdefghi 의 문자열 cde 를 xx 로 바꾼다.

```
iSQL> SELECT REPLACE2('abcdefghicde', 'cde', 'xx') FROM dual;  
REPLACE2('abcdefghicde', 'cde', 'xx')
```

---

abxxfghixx  
1 row selected.

---

## RPAD

### 구문

**RPAD** (*char1*, *n* [, *char2*])

### 설명

문자열 *char1*의 오른쪽에 지정한 길이 *n*이 될 때까지 문자열 *char2*를 (연속적으로) 삽입한 *char1*을 반환한다. *char2*의 기본값은 single blank 이다. 만약, *char1*의 길이가 *n* 보다 길면 *char1*의 일부만(길이 *n*)을 반환한다. 단 *n*의 단위는 Byte 가 아니라 문자 개수를 의미하므로 사용하는 언어에 따라 실제 문자열의 Byte 크기는 다를 수 있다.

### 예제

〈질의〉 “123”이라는 문자 식의 오른쪽에 “0” 문자열을 삽입하여 총 10 자리 숫자를 반환하는 예이다.

```
iSQL> SELECT TO_NUMBER(RPAD('123', 10, '0')) rpad_ex FROM dual;  
RPAD_EX
```

---

1230000000  
1 row selected.

---

## RTRIM

### 구문

## RTRIM (*char1* [, *char2*])

### 설명

*char1*에 가장 오른쪽 문자부터 *char2*에 있는 문자들과 비교한다.

*char1*의 문자열이 *char2*의 문자와 같으면 삭제한다. 반복적으로 *char2*에 문자와 일치되는 문자가 없을 때까지 삭제한 결과를 출력한다.

*char2*는 기본적으로 single blank이며, 이 경우 *char1*의 뒤(오른쪽)에 오는 빈칸을 모두 삭제한다.

### 예제

〈질의〉 문자열 'RIGHTTRIMbaAbab' 중 가장 오른쪽에 나타나 있는 a 와 b를 제외한 문자열을 출력하라.

```
iSQL> SELECT RTRIM ('RIGHTTRIMbaAbab', 'ab') rtrim_ex FROM dual;
RTRIM_EX
```

```
RIGHTTRIMbaA
1 row selected.
```

〈질의〉 고객의 주민등록번호 중에서 앞자리(생년월일)만 출력하라.

```
iSQL> SELECT RTRIM (RTRIM(cno,'1234567890'), '-' ) FROM customer;
RTRIM (RTRIM(CNO,'1234567890'), ' - ')
```

```
730828
771215
711111
.
.
.
20 rows selected.
```

---

## SIZEOF

### 구문

## SIZEOF (*string*)

### 설명

문자열 또는 열에 할당된 크기를 알 수 있다. 문자열은 CHAR, VARCHAR, 또는 숫자형 데이터 타입이 올 수 있으며 숫자형 데이터 타입에 대해서는 VARCHAR로 변환된 크기를 되돌려 준다.

OCTET\_LENGTH 함수가 입력된 문자열의 크기를 반환하는 것에 반해 SIZEOF 함수는 테이블 생성시 정의된 변수의 크기를 반환한다.

\* 참고: 숫자형 데이터 타입에 대해서 INTEGER, BIGINT, SMALLINT 에 대해서는 각각 20 을 반환하며, DECIMAL, FLOAT, NUMBER, NUMERIC, DOUBLE, REAL 등에 대해서는 각각 크기가 45 인 값을 반환한다.

## 예제

〈질의〉 '우리나라'에 할당된 길이를 출력하라. (K016KSC5601로 설정된 경우 한글은 한 문자당 2 바이트 길이가 반환된다.)

```
iSQL> SELECT SIZEOF('우리나라') FROM dual;  
SIZEOF('우리나라')
```

```
_____  
8  
1 row selected.
```

〈질의〉 테이블 dual 의 column x 에 할당된 길이를 출력하라.

```
iSQL> SELECT SIZEOF(x) FROM dual;  
SIZEOF(X)
```

```
_____  
1  
1 row selected.
```

---

## SUBSTR, SUBSTRING

### 구문

**SUBSTR** (*string*, *m* [, *n*])

### 설명

문자 조작 함수

문자 값의 위치 *m* 번째 문자 부터 시작해서 *n* 개의 지정된 길이의 문자열을 반환한다. (*m* 이 음수면 문자 값의 끝부터 세며 *n* 을 생략하면 문자열의 끝까지 모든 문자가 반환된다.)

문자열은 CHAR 또는 VARCHAR 데이터 타입이며, 반환되는 값 역시 문자열로서 같은 데이터 타입이다.

SUBSTRB 함수는 지정한 문자의 위치를 문자 단위가 아닌 바이트 단위로 문자열을 반환한다.

## 예제

〈질의〉 다음은 문자열 SALESMAN 의 1 번째 문자부터 시작해서 길이가 5 인 substring 을 반환하라.

```
iSQL> SELECT SUBSTR('SALESMAN', 1 ,5) Substring FROM dual;
SUBSTRING
_____
SALES
1 row selected.
```

〈질의〉 다음은 ABCDEFG 의 substring 을 반환한다.

```
iSQL> SELECT SUBSTR('ABCDEFG', -5 ,4) Substring FROM dual;
SUBSTRING
_____
CDEF
1 row selected.
```

〈질의〉 다음은 문자열 'ABCDEFG'에서 5 번째 바이트 부터 2 바이트 길이 만큼의 문자를 출력한다.

```
iSQL> SELECT SUBSTRB('ABCDEFG', 5, 2) Substring_with_bytes FROM dual;
SUBSTRING_WITH_BYTES
_____
EF
1 row selected.
```

---

## TRANSLATE

### 구문

**TRANSLATE** (char , string1, string2 )

### 설명

*String1* 의 각 문자에 대응하는 *string2* 의 각 문자로 교체되어 발생될 수 있는 char 를 반환한다. char 에 있는 문자들 중 *string1*에 있지 않은 문자들은 교체되지 않는다. *String1*은 *string2* 보다 더 많은 문자들을 가질수가 있다. 이런 경우, *string1* 끝에 나머지 문자들은 *string2* 와 대응하는 문자가 없다. 만약, 이러한 나머지 문자들이 char 에 있으면 그 문자들은 반환되는 값에서 제거된다. 해당하는 모든 문자를 제거하려면 *string2* 가 empty string 이면 가능하다.

### 예제

〈질의〉 stock 이 50000 개가 넘는 상품 이름 중에서 M 이 들어간 모델을 L 로 바꾸어라.

```
iSQL> SELECT TRANSLATE(gname, 'M', 'L')
```

```
FROM goods
WHERE stock > 50000;
TRANSLATE(GNAME, 'M', 'L')
```

---

```
TL-U200
L-190G
2 rows selected.
```

〈질의〉 문자열 중에서 대문자는 소문자로 변환하라.

```
iSQL> SELECT
TRANSLATE('0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789',
'ABCDEFIGHJKLMNOPQRSTUVWXYZ',
'abcdefghijklmnopqrstuvwxyz')
FROM dual;
TRANSLATE('0123456789ABCDEFGHIJKLMNOPQRS
```

---

```
0123456789abcdefghijklmnopqrstuvwxyz0123456789
1 row selected.
```

〈질의〉 다음 SQL 문은 문자열은 제거하고 숫자만 남는 라이센스 번호를 반환한다.

```
iSQL> SELECT TRANSLATE('3PQR334',
'0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ',
'0123456789') License
FROM dual;
LICENSE
```

---

```
3334
1 row selected.
```

---

## TRIM

### 구문

**TRIM (*char1* [, *char2*])**

### 설명

*Char1*에 문자열을 *char2*에 있는 문자들과 양 방향에서 비교한다.

*Char1*의 문자열이 *char2*의 문자와 같으면 삭제한다. 반복적으로 *char2*에 문자와 일치되는 문자가 없을 때까지 삭제한 결과를 출력한다.

*Char2*는 기본적으로 single blank 이며, 이 경우 *char1*의 앞(왼쪽), 뒤(오른쪽)에 있는 빈칸을 삭제한다.

### 예제

〈질의〉 문자열 'abbAaBbAbba' 중 가장 왼쪽과 오른쪽에 나타나 있는 a 와 b 를 제외한 문자열을 출력하라.

```
iSQL> SELECT TRIM ('abbAaBbAbba', 'ab') trim_ex FROM dual;
TRIM_EX
-----
AaBbA
1 row selected.
```

〈질의〉 사원의 이름과 입사월을 출력하라.

```
iSQL> SELECT ename, TRIM (join_date , '1234567890-')
  FROM employee;
ENAME          TRIM (JOIN_DATE , '1234567890-')
-----
SWNO
HJNO           NOV
HSCHOI         JAN
.
.
.
20 rows selected.
```

---

## UPPER

### 구문

**UPPER** (*char*)

### 설명

대문자 변환함수이다.

입력된 문자를 대문자로 변환한 값을 돌려준다.

### 예제

〈질의〉 문자열 Capital 을 대문자로 출력하라.

```
iSQL> SELECT UPPER('Capital') Uppercase FROM dual;
UPPERCASE
-----
CAPITAL
1 row selected.
```

---

## REPLICATE

### 구문

## **REPLICATE (*char*, *number*)**

### **설명**

*char* 를 지정한 *number* 만큼 반복하여 반환한다. *char* 에는 문자열, *number* 는 양수가 와야 한다. 만약 0 또는 음수가 입력되면 REPLICATE 함수는 null 을 반환한다.

### **예제**

〈질의〉 문자열 ‘KSKIM’을 3 회 반복하여 출력하라.

```
iSQL> SELECT REPLICATE ('KSKIM', 3) FROM dual;
REPLICATE ('KSKIM', 3)
```

---

```
KSKIMKSKIMKSKIM
1 row selected.
```

## **REVERSE\_STR**

### **구문**

## **REVERSE\_STR (*char*)**

### **설명**

인자로 받은 *char* 를 거꾸로 반환한다.

### **예제**

〈질의〉 문자열 ‘KSKIM’을 거꾸로 출력하라.

```
iSQL> SELECT REVERSE_STR ('KSKIM') FROM dual;
REVERSE_STR ('KSKIM')
```

---

```
MIKS
1 row selected.
```

〈질의〉 문자열 ‘알티베이스 4’를 거꾸로 출력하라. (KO16KSC5601 로 설정된 경우)

```
iSQL> SELECT REVERSE_STR ('알티베이스 4') FROM dual;
REVERSE_STR ('알티베이스 4')
```

---

```
4 스이베티알
1 row selected.
```

## STUFF

### 구문

**STUFF (char1, start, length, char2)**

### 설명

char1에서 start 위치의 문자부터 length 만큼 지우고, 그 부분에 char2를 삽입한다.

length에 0이 입력되면 삭제하지 않고 char2를 삽입한다. 이때 char2의 삽입 위치는 start가 가리키는 위치의 왼쪽이다. length가 char1의 길이보다 큰 값이 입력되면, 문자열의 끝까지 지운 후 char2를 삽입한다.

char2를 char1의 맨 뒤에 삽입하기 위해서는 start를 char1의 길이+1로 지정하고, length는 0 또는 양수로 지정해야 한다.

start 또는 length의 값이 음수이거나 start가 char1의 길이보다 2 이상 클 경우에는 에러를 발생한다.

### 예제

〈질의〉 STUFF 함수를 이용하여 ‘KDHONG’을 ‘KILDONG HONG’으로 변환하라.

```
iSQL> SELECT STUFF ('KDHONG', 2, 1, 'ILDONG ') FROM dual;  
STUFF ('KDHONG', 2, 1, 'ILDONG ')
```

```
-----  
KILDONG HONG  
1 row selected.
```

〈질의〉 char2의 문자열을 char1의 문자열 앞에 삽입하라.

```
iSQL> SELECT STUFF ('KDHONG', 1, 0, 'ILDONG ') FROM dual;  
STUFF ('KDHONG', 1, 0, 'ILDONG ')
```

```
-----  
ILDONG KDHONG  
1 row selected.
```

〈질의〉 char2의 문자열을 char1의 문자열 뒤에 삽입하라.

```
iSQL> SELECT STUFF ('KDHONG', 7, 0, 'ILDONG ') FROM dual;  
STUFF ('KDHONG', 7, 0, 'ILDONG ')
```

```
-----  
KDHONGILDONG  
1 row selected.
```

〈질의〉 length가 0인 경우, start의 왼쪽에 char2가 삽입된다.

```
iSQL> SELECT STUFF ('KDHONG', 2, 0, 'ILDONG ') FROM dual;
```

```
STUFF ('KDHONG', 2, 0, 'ILDONG ')
```

---

```
KILDONG DHONG
```

```
1 row selected.
```

〈질의〉 KO16KSC5601 로 설정된 경우

```
iSQL> SELECT STUFF ('알티베이스 0', 5, 1, '데이터베이스') FROM dual;
```

```
STUFF ('알티베이스 0', 5, 1, '데이터베이스')
```

---

```
알티베이스 데이터베이스 0
```

```
1 row selected.
```

---

## 날짜 함수

날짜 및 시간 입력 값에 대한 작업을 수행하며 문자열, 숫자 또는 날짜와 시간 값을 반환한다. 날짜형 데이터의 변환에 이용되는 데이터 형식 스트링은 본 매뉴얼 2 절의 날짜형 데이터 타입부분의 날짜형 데이터 형식절을 참조한다.

---

### ADD\_MONTHS

#### 구문

**ADD\_MONTHS** (*date*, *number*)

#### 설명

월 *number*를 더하여 날짜 *date*를 돌려준다. *number*는 어떤 정수라도 상관없다.

#### 예제

〈질의〉 사원중 MYLEE 의 입사일과 6 개월 후의 날짜를 출력하라.

```
iSQL> SELECT join_date, ADD_MONTHS(join_date, 6)
      FROM employee
     WHERE ename = 'MYLEE';
      JOIN_DATE          ADD_MONTHS(JOIN_DATE, 6)
```

```
1999/12/14 00:00:00  2000/06/14 00:00:00
1 row selected.
```

---

---

### DATEADD

#### 구문

**DATEADD** (*date*, *number*, *date\_field\_name*)

#### 설명

날짜 또는 시간 데이터 형인 *date*의 *date\_field\_name* 부분을 *number* 만큼 증가시킨다. *number* 가 소수일 경우에는 소수 부분은 버린 후에 적용한다.

*date\_field\_name* 이 'SECOND'일 경우에는 *number* 는 68 년

이내의 값이어야 하고, 'MICROSECOND'일 경우에는 number 는 30 일 이내의 값이어야 한다.

DATEADD 함수에서 사용할 수 있는 date\_field\_name 은 다음과 같다.

date field name	내용
CENTURY	현재 날짜(년 부분)에 100*number 만큼을 더한다.
YEAR	현재 날짜(년 부분)에 number 만큼을 더한다.
QUARTER	현재 날짜(월 부분)에 3*number 만큼을 더한다.
MONTH	현재 날짜(월 부분)에 number 만큼을 더한다.
WEEK	현재 날짜(일 부분)에 7*number 만큼을 더한다. (1월 1일부터 그 주 토요일까지가 그 해의 1주이다.)
DAY	현재 날짜(일 부분)에 number 만큼을 더한다.
HOUR	현재 날짜(시 부분)에 number 만큼을 더한다.
MINUTE	현재 날짜(분 부분)에 number 만큼을 더한다.
SECOND	현재 날짜(초 부분)에 number 만큼을 더한다.
MICROSECOND	현재 날짜(마이크로 초 부분)에 number 만큼을 더한다.

## 예제

〈질의〉 입사한지 40 일이 안된 사원의 수를 구하라.

```
iSQL> SELECT COUNT(*) FROM employee WHERE join_date >
      DATEADD (SYSDATE, -40, 'DAY');
      COUNT
      _____
      5
      1 row selected.
```

---

## DATEDIFF

## 구문

## DATEDIFF (startdate, enddate, date\_field\_name)

### 설명

enddate에서 startdate를 뺀 값(enddate-startdate)을 date\_field\_name으로 보여준다. enddate보다 startdate의 값이 더 크면 음수로 반환된다.

enddate와 startdate에서 지정한 date\_field의 값을 각각 구해서 빼는 것으로 DATEDIFF 함수의 결과값은 항상 정수이다.

사용 가능한 date\_field\_name은 다음과 같다.

date field name	내용
CENTURY	세기
YEAR	년
QUARTER	분기
MONTH	월
WEEK	주 (1월 1일부터 그 주 토요일까지가 그 해의 1주이다.)
DAY	일
HOUR	시
MINUTE	분
SECOND	초
MICROSECOND	마이크로 초

DATEDIFF 함수는 결과값의 범위를 한정하고 있다.

date\_field\_name이 'MICROSECOND'일 때는 enddate에서 startdate를 뺀 값이 30일 이내여야 한다. 그리고 초를 나타내는 'SECOND'일 경우에 한계 범위는 68년이다. 한계 범위를 초과하면 에러가 발생한다.

반환형은 BIG INTEGER이다.

### 예제

〈질의〉 2005년 8월 31일과 2005년 11월 30일 간의 개월 수의 차를 구하라.

```
iSQL> SELECT DATEDIFF ('31-AUG-2005', '30-NOV-2005', 'MONTH') FROM
dual;
DATEDIFF ('31-AUG-2005', '30-NOV-2005',
```

---

3

1 row selected.

## DATENAME

### 구문

**DATENAME (date, date\_field\_name)**

### 설명

지정한 date 의 date\_field\_name 에 맞는 이름을 반환한다.

사용 가능한 date\_field\_name 은 다음과 같다.

date field name	내용
MONTH	월의 이름. (전체 이름)
MON	월의 이름. (약자)
DAY	요일의 이름. (전체 이름)
DY	요일의 이름. (약자)

date\_field\_name 별 출력 결과는 다음과 같다. date\_field\_name 의 대소문자 형식에 따라서 결과값의 대소문자도 같은 형식으로 출력된다.

- MONTH  
JANUARY, FEBRUARY, MARCH, APRIL, MAY, JUNE,  
JULY, AUGUST, SEPTEMBER, OCTOBER, NOVEMBER,  
DECEMBER
- MON  
JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT,  
NOV, DEC
- DAY  
SUNDAY, MONDAY, TUESDAY, WEDNESDAY,  
THURSDAY, FRIDAY, SATURDAY
- DY  
SUN, MON, TUE, WED, THU, FRI, SAT

### 예제

〈질의〉 1980 년 12 월 28 일의 월의 이름을 구하라.

```
iSQL> SELECT DATENAME ('28-DEC-1980', 'Month') FROM dual;  
DATENAME ('28-DEC-1980', 'Month')
```

```
-----  
December  
1 row selected.
```

## EXTRACT, DATEPART

### 구문

**EXTRACT (date, date\_field\_name)**  
**DATEPART (date, date\_field\_name)**

### 설명

입력받은 date에서 date\_field\_name 부분만을 반환한다.

date field name	내용
CENTURY	세기
YEAR	년
QUARTER	분기
MONTH	월
WEEK	일년 중 몇 번째 주인지 나타냄. (1월 1일부터 그 주 토요일까지가 그 해의 1주이다.)
WEEKOFMONTH	그 달의 몇 번째 주인지 나타냄. (1일부터 그 주 토요일까지가 그 달의 1주이다.)
DAY	일
DAYOFYEAR	일년 중 몇 번째 날인지 나타냄.
DAYOFWEEK	해당 주의 몇 번째 날인지 나타냄. (일요일: 1)
HOUR	시
MINUTE	분
SECOND	초
MICROSECOND	마이크로 초

### 예제

〈질의〉 이름이 ‘KSKIM’인 사원이 입사한 분기를 구하라.

```
iSQL> SELECT DATEPART (join_date, 'QUARTER') FROM employee WHERE  
ename = 'KSKIM' ;  
DATEPART (join_date, 'QUARTER')
```

3

1 row selected.

## MONTHS\_BETWEEN

## 구문

**MONTHS\_BETWEEN (date1, date2)**

## 설명

date1에서 date2를 뺀 값을 개월 수로 반환한다. date1이 date2보다 작다면 음수가 된다. date1과 date2가 같은 달의 같은 날이거나, 다른 달의 같은 날 또는 서로 다른 달의 마지막 날이라면 정수가 반환된다. 이 경우에는 시간, 분, 초, 마이크로 초 등에 상관없이 무조건 정수가 반환된다.

반환되는 값이 정수가 아닐 경우에는 한 달을 31일로 간주하고, 시간, 분, 초, 마이크로 초 등을 고려해서 소수 부분을 계산한다.

반환형은 DOUBLE이다.

## 예제

〈질의〉 1995년 2월 2일에서 1995년 1월 1일을 뺀 값을 개월 수로 구하라.

```
iSQL> SELECT MONTHS_BETWEEN (TO_DATE('02-02-1995','MM-DD-YYYY'),  
TO_DATE('01-01-1995','MM-DD-YYYY') ) Months FROM DUAL;  
MONTHS  
-----  
1.03225806451613  
1 row selected.
```

---

## ROUND

## 구문

**ROUND (date [, date\_field\_name])**

## 설명

지정한 date\_field\_name에 맞춰서 반올림한 date를 반환한다. date\_field\_name의 기본값은 'DAY'이다.

사용 가능한 date\_field\_name은 다음과 같다.

date field name	내용
CENTURY	51년부터 다음 세기로 올림. (세기는 xxx1년부터 시작함)
YEAR	7월 1일부터 다음 해로 올림.
QUARTER	해당 분기의 두 번째 달의 16일부터

	올림.
MONTH	16일부터 다음 달로 올림.
WEEK	목요일부터 다음 주 일요일로 올림.
DAY	PM 12:00부터 다음 일로 올림.
HOUR	30분부터 다음 시로 올림.
MINUTE	30초부터 다음 분으로 올림.

## 예제

〈질의〉 1980년 12월 27일을 YEAR로 반올림하여 출력하라.

```
iSQL> SELECT ROUND ( TO_DATE('27-DEC-1980', 'DD-MON-YYYY'), 'YEAR')
   FROM dual;
ROUND ( TO_DATE('27-DEC-1980', 'DD-MON-Y
```

---

1981/01/01 00:00:00

1 row selected.

## LAST\_DAY

### 구문

**LAST\_DAY** (*date*)

### 설명

*date*을 포함하는 달의 마지막 일을 출력한다.

## 예제

〈질의〉 12월의 마지막 일을 출력하라.

```
iSQL> SELECT LAST_DAY(TO_DATE('15-DEC-2001')) FROM dual;
LAST_DAY(TO_DATE('15-DEC-2001'))
```

---

2001/12/31 00:00:00

1 row selected.

〈질의〉 사원들의 입사월 마지막 일을 출력하라.

```
iSQL> SELECT LAST_DAY(join_date ) FROM employee;
LAST_DAY(JOIN_DATE )
```

---

1999/11/30 00:00:00

2000/01/31 00:00:00

.

.

.

20 rows selected.

---

## NEXT\_DAY

### 구문

**NEXT\_DAY (date, char)**

### 설명

입력된 날짜(*date*) 이후에 알고 싶은 요일(*char*)의 날짜를 출력한다.

*Char* 입력은 요일을 영문으로 입력한다.

### 예제

〈질의〉 사원들의 입사일, 입사한 후 첫번째 일요일을 출력하라.

```
iSQL> SELECT join_date, NEXT_DAY(join_date, 'SUNDAY') First_sunday FROM  
employee;  
JOIN_DATE          FIRST_SUNDAY
```

---

```
1999/11/18 00:00:00  1999/11/21 00:00:00  
2000/01/11 00:00:00  2000/01/16 00:00:00  
. .  
.
```

20 rows selected.

---

## SYSDATE

### 구문

**SYSDATE**

### 설명

현재 시스템의 날짜를 출력한다.

### 예제

〈질의〉 시스템 날짜(현재 날짜)를 출력하라.

```
iSQL> SELECT SYSDATE System_Date FROM dual;  
SYSTEM_DATE
```

---

2005/01/20 09:49:33

1 row selected.

---

## SYSTIMESTAMP

### 구문

**SYSTIMESTAMP**

### 설명

현재 시스템의 날짜를 출력한다. SYSDATE 함수의 alias이며 time zone은 지원하지 않는다.

### 예제

〈질의〉 시스템 날짜(현재 날짜)를 출력하라.

```
iSQL> SELECT SYSTIMESTAMP FROM dual;
SYSTIMESTAMP
```

---

2005/01/20 09:49:33

1 row selected.

---

## TRUNC (date)

### 구문

**TRUNC (date [, 'fmt'])**

### 설명

버림 함수이다.

fmt에 명시된 단위까지 유효숫자로 date를 반환한다.

해당 fmt를 제외한 나머지 단위는 0으로 출력된다.

두번째 인자를 생략한 경우 날짜까지만 반환하고 시간 이하 단위는 0으로 출력된다.

### 예제

〈질의〉 다음 예는 시스템 시간의 시간을 버림한 결과를 돌려준다.

```
iSQL> SELECT TRUNC(SYSDATE) FROM DUAL;
〈결과〉
```

TRUNC(SYSDATE)

---

2005/07/19 00:00:00

1 row selected.

〈질의〉 다음 예는 날짜를 버림한 결과를 돌려준다.

iSQL> **SELECT TRUNC(TO\_DATE('2005-JUL-19','YYYY-MON-DD'), 'YEAR')**

New\_Year **FROM DUAL;**

〈결과〉

NEW\_YEAR

---

2005/01/01 00:00:00

1 row selected.

---

## 변환 함수

입력 값(date, character, INTEGER 또는 number)에 대해 문자형, DATE 날짜 형식, 또는 NUMBER 데이터 형식으로 변환한다.

---

### TO\_BIN

#### 구문

**TO\_BIN** (*integer*)

#### 설명

주어진 *integer*를 2 진수 형태로 변환한다. 기본적으로 알티베이스에서 사용되는 정수는 10 진수이다.

\* 반환된 결과의 데이터 형식은 문자형이며, SELECT 문에서만 사용할 수 있다.

#### 예제

〈질의〉 주어진 값을 2 진수 형태로 변환하라.

```
iSQL> SELECT TO_BIN(1000) FROM dual;
TO_BIN(1000)
```

```
111101000
1 row selected.
```

---

### BIN\_TO\_NUM

#### 구문

**BIN\_TO\_NUM** (*char*)

#### 설명

1 과 0 으로만 이루어진 문자형 인자 값을 10 진수로 변환한다.  
반환형은 정수형이다.

*char*에는 최대 32 개까지의 문자가 올 수 있다.

#### 예제

〈질의〉 주어진 2 진수를 10 진수로 변환하라.

```
iSQL> SELECT BIN_TO_NUM ('1010') FROM dual;  
BIN_TO_NUM ('1010')
```

```
-----  
10  
1 row selected.
```

---

## HEX\_TO\_NUM

### 구문

**HEX\_TO\_NUM (*char*)**

### 설명

0~9 와 A~F 로만 이루어진 문자형 인자 값을 10 진수로 변환한다.  
반환형은 정수형이다.

*char* 에는 최대 8 개까지의 문자가 올 수 있다.

### 예제

〈질의〉 주어진 16 진수를 10 진수로 변환하라.

```
iSQL> SELECT HEX_TO_NUM ('1A') FROM dual;  
HEX_TO_NUM ('1A')
```

```
-----  
26  
1 row selected.
```

---

## OCT\_TO\_NUM

### 구문

**OCT\_TO\_NUM (*char*)**

### 설명

0~7 로만 이루어진 문자형 인자 값을 10 진수로 변환한다. 반환형은 정수형이다.

*char* 에는 최대 11 개까지의 문자가 올 수 있다.

### 예제

〈질의〉 주어진 8 진수를 10 진수로 변환하라.

```
iSQL> SELECT OCT_TO_NUM ('71') FROM dual;
OCT_TO_NUM ('71')
```

```
57
1 row selected.
```

---

## TO\_CHAR(날짜형)

### 구문

**TO\_CHAR (date [, 'fmt'])**

### 설명

입력된 날짜를 date\_fmt 형식에 맞게 문자형으로 변환하여 출력한다. date\_fmt 는 date 의 날짜 형식을 지정한다. date\_fmt 가 생략되었을 경우 altibase.properties 에 있는 DEFAULT\_DATE\_FORMAT 형식을 따라야 한다. 이 형식의 기본값은 DD-MON-RRRR 이다. 날짜형 데이터의 변환에 이용되는 데이터 형식 스트링은 본 매뉴얼 2 절의 날짜형 데이터 타입부분의 날짜형 데이터 형식절을 참조한다.

### 예제

〈질의〉 사원의 입사일이 YYYY-MM-DD HH:MI:SS 형식으로 표시되도록 하라.

```
iSQL> SELECT ename, TO_CHAR(join_date, 'YYYY-MM-DD HH:MI:SS')
   Join_date FROM employee;
ENAME          JOIN_DATE
```

```
SWNO
HJNO           1999-11-18 00:00:00
HSCHOI         2000-01-11 00:00:00
.
.
.
20 rows selected.
```

---

## TO\_CHAR (숫자 변환)

### 구문

**TO\_CHAR (number [, number\_format])**

## 설명

입력된 숫자형 자료를 문자형으로 변환하여 출력한다. 이때 숫자형 데이터 표현 형식을 지정하여 출력 형식을 변경할 수도 있다.

## 예제

〈질의〉 다음 SQL 문은 문자열과 숫자를 함축적 변환을 사용하여 숫자로 해석한 후 TO\_CHAR 함수를 이용하여 문자형으로 변환하여 출력한다.

```
iSQL> SELECT TO_CHAR('01110' + 1) FROM dual;  
TO_CHAR('01110' + 1)
```

```
-----  
1111  
1 row selected.
```

〈질의〉 다음 SQL 문은 숫자를 다양한 형식의 문자열로 출력하는 예를 보여준다.

```
iSQL> SELECT TO_CHAR (123, '99999') FROM dual;  
TO_CHAR (123, '99999')
```

```
-----  
123  
1 row selected.
```

```
iSQL> SELECT TO_CHAR (123.4567, '999999') FROM dual;  
TO_CHAR (123.4567, '999999')
```

```
-----  
123  
1 row selected.
```

```
iSQL> SELECT TO_CHAR (1234.578, '9999.99') FROM dual;  
TO_CHAR (1234.578, '9999.99')
```

```
-----  
1234.58  
1 row selected.
```

```
iSQL> SELECT TO_CHAR (1234.578, '999.99999') FROM dual;  
TO_CHAR (1234.578, '999.99999')
```

```
-----  
#####  
1 row selected.
```

---

## TO\_DATE

## 구문

## **TO\_DATE (*char*[, '*fmt*'])**

### **설명**

TO\_DATE 는 CHAR, VARCHAR 데이터 타입의 *char*를 DATE 데이터 타입으로 변환한다. *fmt*는 *char*의 날짜 형식을 지정한다. *fmt*가 생략되었을 경우 'DD-MON-YY' 또는 'DD-MON-YYYY' 형식을 따라야 한다.

만약 년 또는 월의 날짜 형식을 명시하지 않은 경우 TO\_DATE 결과값은 실행 시 년 또는 월의 값으로 변환한다. 예를 들어, TO\_DATE(TO\_CHAR(sysdate, 'YYYY'), 'YYYY') 실행 시각이 2005/08/24 17:32:34 일 경우 TO\_DATE 의 결과값은 20050801 00:00:00 이 된다. TO\_DATE 수행 시 월의 날짜 형식을 명시하지 않았으므로 실행 시 월인 8 월이 결과값이 되고 일, 시, 분, 초는 초기값인 1 일 00 시 00 분 00 초가 결과값이 된다.

### **예제**

〈질의〉 2001년 11월 19일에 입사한 사원의 번호, 이름, 성별과 입사일을 입력하라.

```
iSQL> INSERT INTO employee(eno, ename, sex, join_date) VALUES(22,
  'MHJUNG', 'F', TO_DATE('2001-11-19 00:00:00', 'YYYY-MM-DD HH:MI:SS'));
1 row inserted.
```

〈질의〉 월의 날짜 형식을 명시하지 않은 경우 TO\_DATE 결과

```
iSQL> select to_char(to_date(to_char(sysdate, 'YYYY'), 'YYYY'), 'YYYYMMDD
HH24:MI:SS') from dual;
TO_CHAR(TO_DATE(TO_CHAR(SYSDATE,'YYYY'),
_____
20080501 00:00:00
1 row selected.
(SYSDATE = 20080502 13:17:20)
```

## **TO\_HEX**

### **구문**

## **TO\_HEX (*integer*)**

### **설명**

주어진 정수값을 16 진수 형태로 변환한다. 기본적으로 알티베이스에서 사용되는 정수는 10 진수이다.

\* 반환된 결과의 데이터 형식은 문자형이며, SELECT 문에서만 사용할 수 있다.

## 예제

〈질의〉 주어진 값을 16 진수 형태로 변환하라.

```
iSQL> SELECT TO_HEX(1000) FROM dual;
TO_HEX(1000)
_____
3E8
1 row selected.
```

---

## TO\_NUMBER

### 구문

**TO\_NUMBER (char [, number\_fmt] )**

### 설명

TO\_NUMBER 함수는 문자열 *char*를 숫자형 데이터형으로 변환한다. 이때 사용자는 원하는 숫자 출력 형식을 지정할 수 있다.

## 예제

〈질의〉 문자열 10000.00 을 숫자형식으로 변환하여 출력하라.

```
iSQL> UPDATE employee
SET salary = salary + TO_NUMBER( '10000.00')
WHERE ename = 'MSKIM';
1 row updated.
```

〈질의〉 다음의 예제는 주어진 문자열을 다양한 숫자형 출력 형식을 지정하여 출력하는 예를 보여준다.

```
iSQL> SELECT TO_NUMBER ( '0123.4500', '0990.9909' ) FROM dual;
TO_NUMBER ( '0123.4500', '0990.9909' )
_____
123.45
1 row selected.
```

```
iSQL> SELECT TO_NUMBER ( '$12,3.45-', '09,$0.00S' ) FROM dual;
TO_NUMBER ( '$12,3.45-', '09,$0.00S' )
_____
-123.45
1 row selected.
```

```
iSQL> SELECT TO_NUMBER ( '($183.5)', '$9,000.0PR' ) FROM dual;
```

```
TO_NUMBER ( '$183.5', '$9,000.0PR' )
```

---

```
-183.5
```

```
1 row selected
```

---

## TO\_OCT

### 구문

**TO\_OCT** (*integer*)

### 설명

주어진 *INTEGER*를 8 진수 형태로 변환한다. 기본적으로 알티베이스에서 사용되는 정수는 10 진수이다.

\* 반환된 결과의 데이터 형식은 문자형이며, SELECT 문에서만 사용할 수 있다.

### 예제

〈질의〉 주어진 값을 8 진수 형태로 변환하라.

```
iSQL> SELECT TO_OCT(1000) FROM dual;
```

---

```
TO_OCT(1000)
```

```
1750
```

```
1 row selected.
```

---

## 기타 함수

---

### CASE2

#### 구문

```
CASE2 (expression1, result1,  
[,expression2, result2,...]  
, [ default])
```

#### 설명

*expression* 이 참이면 *result* 값을 반환한다. 만약 거짓이면 *default* 값을 반환하고, *default*가 지정되어 있지 않으면 null 을 반환한다.

#### 예제

〈질의〉 월급여가 2000000 보다 크면 'HIGH'를, 1500000 보다 작으면 'LOW'을 출력하라.

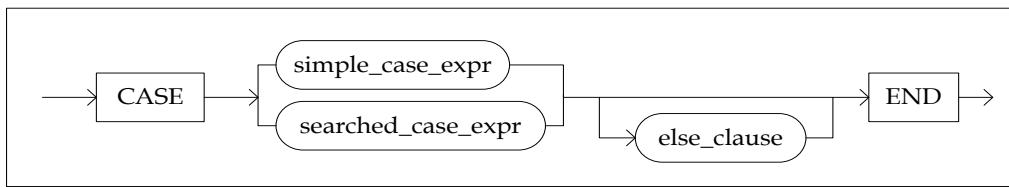
```
iSQL> SELECT ename, emp_job, salary, CASE2(salary > 2000000, 'HIGH',  
salary < 1500000, 'LOW') Sal FROM employee;
```

ENAME	EMP_JOB	SALARY	SAL
SWNO	CEO		
HJNO	DESIGNER	1500000	
HSCHOI	ENGINEER	2000000	
KSKIM	ENGINEER	1800000	
SJKIM	ENGINEER	2500000	HIGH
HYCHOI	PROGRAMMER	1700000	
HJMIN	MANAGER	500000	LOW
.			
.			
20 rows selected.			

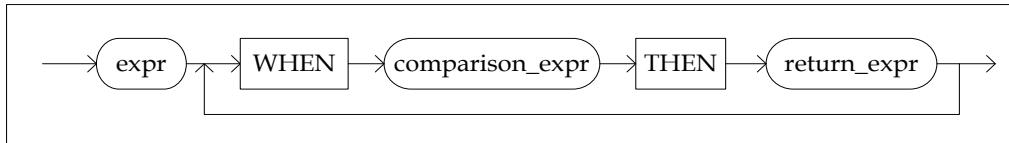
---

### CASE WHEN

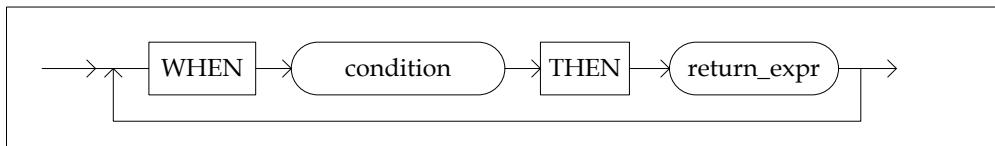
#### 구문



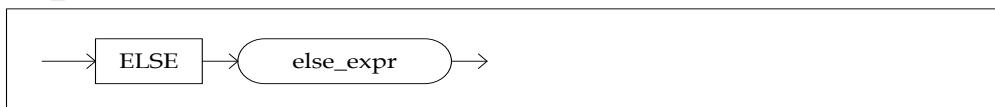
### *simple\_case\_expr*



### *searched\_case\_expr*



### *else\_clause*



## 설명

CASE2 함수와 동일한 기능을 하는 표현식으로 CASE WHEN 구문을 사용할 수 있다. 만약 *expr*이 참이면 *return\_expr* 값을 반환하고, 거짓이면 *else\_expr* 값을 반환한다. 또는 값이 없으면 널(NULL)을 반환한다.

*simple\_case\_expr*은 비교 조건이 동등 연산(=)인 경우에만 사용할 수 있고, *searched\_case\_expr*은 CASE2 함수와 같이 다양한 비교 연산을 할 수 있다.

## 예제

〈질의〉 C1 컬럼의 셋째 자리에 있는 한자리 값이 a 이면 aaaaa, b 이면 bbbbb, c 이면 ccccc 를 출력하라.

```

iSQL> create table test (c1 char(10));
iSQL> insert into test values('abcdefghijkl');
iSQL> select CASE substring(c1,3,1)
          WHEN 'a' THEN 'aaaaa'
          WHEN 'b' THEN 'bbbbbb'
          WHEN 'c' THEN 'ccccc'
      END

```

```
from test;

CASE SUBSTRING(C1,3,1)
-----
ccccc
1 row selected.
```

---

## DECODE

### 구문

```
DECODE (expression, search1, result1,
[, search2, result2,...]
[, default])
```

### 설명

DECODE 함수는 *expression* 을 각각의 *search* 값과 하나씩 순서대로 비교한 후 해독하여 *expression* 이 *search* 값과 동일하면 *result* 값을 반환한다. 만약 해당하는 *search* 값이 없으면 *default* 값을 반환하고, *default* 가 지정되어있지 않으면 null 을 반환한다.

### 예제

〈질의〉 i 가 null 이면 NULL, 1 이면 ONE, 2 이면 TWO 를 반환하라.

```
iSQL> CREATE TABLE t2(i NUMBER);
Create success.
iSQL> INSERT INTO t2 VALUES(NULL);
1 row inserted.
iSQL> INSERT INTO t2 VALUES(1);
1 row inserted.
iSQL> INSERT INTO t2 VALUES(2);
1 row inserted.
iSQL> INSERT INTO t2 VALUES(3);
1 row inserted.
iSQL> SELECT DECODE(i, NULL, 'NULL', 1, 'ONE', 2, 'TWO') Revised_i FROM
t2;
REVISED_I
-----
NULL
ONE
TWO

4 rows selected.
```

〈질의〉 emp\_job 이 ENGINEER 이면 급여를 10%, SALESMAN 이면

12%, MANAGER 이면 20% 인상하며, 그 외의 사원은 현재 급여를 출력하라.

```
iSQL> SELECT emp_job, salary,
      DECODE(RTRIM(emp_job, ' '),
      'ENGINEER', salary*1.1,
      'SALESMAN', salary*1.12,
      'MANAGER', salary*1.20,
      salary) Revised_salary
FROM employee;
EMP_JOB          SALARY      REVISED_SALARY
-----  -----
CEO                1500000    1500000
DESIGNER         2000000    2200000
ENGINEER         1800000    1980000
ENGINEER         2500000    2750000
PROGRAMMER       1700000    1700000
MANAGER          500000     600000
.
.
.
20 rows selected.
```

---

## DIGEST

### 구문

**DIGEST(expression1, algorithm\_name )**

### 설명

공개된 메시지 다이제스트 알고리즘을 사용하여 주어진 문자열에 대한 해쉬값을 Varchar 타입의 문자열로 반환한다. Altibase4 가 현재 지원하는 알고리즘은 SHA-1 이다.

### 예제

〈질의〉 예제 테이블의 특정 칼럼값을 SHA-1 알고리즘으로 변환한 문자열을 출력한다.

```
iSQL> SELECT DIGEST( 'I am a boy.' , 'SHA-1' ) FROM DUAL;
      DIGEST('I am a boy.', 'SHA-1')
```

```
A817613E0B781BBF01816F36A8B0DC7C98B2C0CC
1 ROW SELECTED. rows selected.
```

---

## DUMP

### 구문

**DUMP (expression)**

### 설명

입력된 자료를 분석하여 ‘자료형, 길이, 메모리 내용’의 형식으로 출력한다.

### 예제

〈질의〉 사원의 번호와 이름에 입력된 자료형에 관한 정보를 3 개만 출력하라.

```
iSQL> SELECT DUMP(eno) EMP_NUMBER, DUMP(ename) Name  
      FROM employee LIMIT 3;  
EMP_NUMBER
```

---

NAME

---

```
Type=INTEGER Length=4: 0,0,0,1  
Type=CHAR(ENGLISH) Length=22:  
0,20,83,87,78,79,32,32,32,32,32,32,32,32,32,32,32,32,32,32,32,32,32,32,32  
Type=INTEGER Length=4: 0,0,0,2  
Type=CHAR(ENGLISH) Length=22:  
0,20,72,74,78,79,32,32,32,32,32,32,32,32,32,32,32,32,32,32,32,32,32,32,32  
Type=INTEGER Length=4: 0,0,0,3  
Type=CHAR(ENGLISH) Length=22:  
0,20,72,83,67,72,79,73,32,32,32,32,32,32,32,32,32,32,32,32,32,32,32,32,32  
3 rows selected.
```

---

## GREATEST

### 구문

**GREATEST (expressions)**

### 설명

표현식들 중에서 가장 큰 값을 가진 문자열을 반환한다. 데이터 타입은 CHAR 또는 VARCHAR 이다.

### 예제

〈질의〉

```
iSQL> SELECT GREATEST('HARRY', 'HARRIOT', 'HAROLD') Greatest FROM
dual;
GREATEST
_____
HARRY
1 row selected.
```

---

## LEAST

### 구문

**LEAST(expressions)**

### 설명

표현식들 중에서 가장 작은 값을 가진 문자열을 반환한다. 데이터 타입은 CHAR 또는 VARCHAR 이다.

### 예제

〈질의〉

```
iSQL> SELECT LEAST('HARRY','HARRIOT','HAROLD') Least FROM dual;
LEAST
_____
HAROLD
1 row selected.
```

---

## ROWNUM

### 구문

**ROWNUM**

### 설명

의사 레코드 번호(pseudo rownum BIGINT value)를 정수 형태로 반환한다. 반환 범위는 1부터 BIGINT 형의 최대값이다.

레코드 번호는 테이블이나 조인된 뷰의 레코드 순서에 따라 부여되지만, ORDER BY, GROUP BY, HAVING 절에 의해 순서가 바뀔 수 있다.

SELECT 문에서 사용할 수 있으며 UPDATE, DELETE 등의

DML에서는 사용할 수 없다.<sup>4</sup>

## 예제

〈질의〉 사원 이름순으로 검색하여 앞에서 3명까지만 사원의 사번, 이름, 전화번호, 상태를 출력하라.

```
iSQL> SELECT eno, ename, emp_tel, status FROM employee WHERE  
ROWNUM < 4 ORDER BY ename;
```

ENO	ENAME	EMP_TEL	STATUS
1	EJJUNG	1195662365	R
2	HJNO	113654540	H
3	HSCHOI	162581369	H
3 rows selected.			

---

## NVL

### 구문

**NVL (expression1, expression2)**

### 설명

널값을 다른 수나 캐릭터로 변환하여 사용한다. 사용되는 데이터 타입은 DATE, CHAR 및 NUMBER이다. 표현식의 데이터 유형이 일치해야 한다.

## 예제

〈질의〉 사원의 이름과 급여를 출력한다. 급여 데이터가 없을 경우 'NOT'을 출력하라.

```
iSQL> SELECT ename, NVL(TO_CHAR(salary), 'NOT')  
FROM employee;  
ENAME          NVL(TO_CHAR(SALARY), 'NOT')
```

SWNO	NOT
HJNO	1500000
HSCHOI	2000000
.	
.	

---

<sup>4</sup> UPDATE, DELETE 문의 경우 LIMIT 구문으로 ROWNUM과 유사하게 사용할 수 있다.

iSQL> DELETE FROM employee LIMIT 1, 10 ;

20 rows selected.

---

## NVL2

### 구문

**NVL2** (*expression1*, *expression2*, *expression3*)

### 설명

*expression1* 이 NULL 아니면 NVL2 는 *expression2*를 반환하고,  
NULL 이면 *expression3*를 반환한다.

### 예제

〈질의〉 사원의 이름과 급여를 출력하되, 급여 데이터가 있을 경우는  
10% 인상된 급여를, 없을 경우는 'NOT'을 출력하라.

```
iSQL> SELECT ename, salary, NVL2(TO_CHAR(salary), TO_CHAR(salary * 1.1),  
'NOT') Nvl2_salary FROM employee;  
ENAME          SALARY  
-----  
NVL2_SALARY  
-----  
SWNO  
NOT  
HJNO          1500000  
1650000  
. . .  
20 rows selected.
```

---

## SENDMSG

### 구문

**SENDMSG** ( VARCHAR ipaddr,  
INTEGER port,  
VARCHAR msg,  
INTEGER ttl )

### 설명

사용자 메시지를 지정된 ip-address, port 에 Socket datagram 으로 전달한다. 일반 ip-address 를 입력하면 UDP datagram 을 전송하며, Multicast ip-address 를 입력하면 Multicast datagram 을 전송한다.

Multicast ip-address 는 예약된 Multicast group 을 제외하면 225.0.0.0 ~ 238.0.0.255 로 제한된다.

port 는 1025~65535 까지의 범위이다.

msg 는 최대 2048 바이트를 넘을 수 없다.

ttl 은 TimeToLive 의 준말로, Multicast 전송시 다음과 같이 전송 범위를 제한하는 값이다. 이는 0 ~ 255 까지의 범위이며 Multicast ip-addresses 로 메시지를 전송할 때 적용된다.

TTL	범위
0	호스트 내부로 제한, 인터페이스로 출력되지 않음.
1	동일 서브넷으로 제한, 라우터에는 포워딩하지 않음.
< 32	동일 사이트(SITE), 단체나 부서로 제한.
< 64	동일 지역(Region)으로 제한.
< 128	동일 대륙으로 제한.
< 255	무제한, 전세계.

반환값은 보낸 메시지의 길이(INTEGER)이다.

## 예제

〈질의〉 일반 ip-address 로 사용자 메시지를 보내는 예를 작성하라.  
(ttl 값은 무시됨)

```
iSQL> SELECT SENDMSG( '192.168.1.60', 12345, 'THIS IS MESSAGE', 1 ) FROM  
T1;  
SENDMSG( '192.168.1.60', 12345, 'THIS IS
```

```
-----  
15  
1 row selected.
```

〈질의〉 Multicast ip-address 로 사용자 메시지를 보내는 예를 작성하라. (ttl 값이 적용됨)

```
iSQL> SELECT SENDMSG( '226.0.0.37', 12345, 'THIS IS MESSAGE', 0 ) FROM  
T1;  
SENDMSG( '192.168.1.60', 12345, 'THIS IS
```

```
-----  
15  
1 row selected.
```

---

## **USER\_ID**

### **구문**

**USER\_ID()**

### **설명**

현재 접속한 사용자의 ID를 출력한다. 데이터 타입은 integer이다.

### **예제**

〈질의〉 현재 사용자의 테이블 정보를 조회하라.

```
iSQL> SELECT table_name FROM system_.sys_tables_ WHERE user_id =  
USER_ID();
```

TABLE_NAME
T_CO
T_DEPT
T_EMP

3 rows selected.

---

## **USER\_NAME**

### **구문**

**USER\_NAME()**

### **설명**

현재 접속한 사용자의 이름을 출력한다. 데이터 타입은 integer이다.

### **예제**

〈질의〉 현재 사용자의 이름을 조회하라.

```
iSQL> SELECT user_name(), user_id() FROM dual;
```

USER_NAME	USER_ID
SYS	2

1 row selected.

---

## SESSION\_ID

### 구문

SESSION\_ID()

### 설명

현재 접속한 사용자의 세션 식별자를 출력한다. 데이터 타입은 integer이다.

### 예제

〈질의〉 현재 사용자의 클라이언트 NLS 정보를 조회하라.

```
iSQL> SELECT client_nls FROM v$session WHERE id = SESSION_ID();
CLIENT_NLS
```

```
-----  
US7ASCII  
1 row selected.
```

---

## 중첩 함수

### 설명

단일 행 함수는 여러 번 중첩될 수 있다. 중첩 함수는 가장 안쪽부터 바깥쪽 순으로 계산 된다.

### 예제

〈질의〉 입사일로부터 여섯 달 경과 후 첫 번째 월요일의 날짜를 입사일을 기준으로 출력하라.

```
iSQL> SELECT TO_CHAR(NEXT_DAY(ADD_MONTHS(join_date, 6), 'MONDAY'),
  'DD-Mon-YYYY') Ater_six_months
  FROM employee
 ORDER BY join_date;
ATER_SIX_MONTHS
```

```
-----  
22-May-2000  
05-Jun-2000  
19-Jun-2000
```

20 rows selected.

---

## 암호화 함수

주어진 문자열을 암호화하거나 암호화 된 문자열을 해제하는 함수를 제공한다. 암호화/복호화 알고리즘은 DES(Data Encryption Standard)를 사용한다.

8 바이트 블록 암호화 알고리즘은 CBC(Cipher Block Chaining)을 사용한다.

---

## DESENCRYPT

### 구문

**DESENCRYPT** ( *varchar my\_string* , *varchar key\_string* )

### 설명

*my\_string* : 암호화 하기를 원하는 문자열을 입력한다. 이 문자열은 반드시 그 길이가 8의 배수 크기여야 한다.

*key\_string* : 암호화 키가 되는 문자열을 입력한다. 키 값의 최소 길이는 8이고, 9 이후의 문자열은 무시한다.

### 예제

아래 DESDECRYPT 예제를 참조한다.

---

## DESDECRYPT

### 구문

**DESDECRYPT** ( *varchar encrypted\_string* , *varchar key\_string* )

### 설명

*encrypted\_string* : 복호화 하기를 원하는 문자열을 입력한다. 이 문자열은 반드시 그 길이가 8의 배수 크기여야 한다.

*key\_string* : 암호화 키였던 문자열을 입력한다. 키 값의 최소 길이는 8이고, 9 이후의 문자열은 무시한다.

\* 주의: 암호화된 문자열을 화면에 출력하면 터미널 에뮬레이터

오류가 발생할 수 있다.

## 예제

〈질의 1〉 암호화한 텍스트를 테이블에 저장 및 복호화하여 출력하라.

```
iSQL> create table t1( encrypted_string varchar(40) );
Create success.
```

1) 암호화하여 저장하기

```
iSQL> insert into t1 values( desencrypt( 'A4 ALTIBASE Corporation.',
'altibase' ) );
1 row inserted.
```

2) 암호화된 문장이 들어가서 select 를 하면 알아볼 수 없다.

```
iSQL> select * from t1;
T1,ENCRYPTED_STRING
-----
Z\uf900\ub5b7\ub94c]ffff\ffff\ffffx\ffffXek
\ffff
1 row selected.
```

3) 암호화 할 때와 동일한 키를 이용하여 복호화해서 원문을 출력

```
iSQL> select desdecrypt(encrypted_string, 'altibase') from t1;
DESDECRYPT(ENCRYPTED_STRING, 'altibase')
-----
A4 ALTIBASE Corporation.
1 row selected.
```

〈질의 2〉 문자열의 8 배수 길이 제한 및 키의 8 이상 길이 제한을 없앤 PSM 을 사용하여 암호화 및 복호화 하라.

```
iSQL> create table t1( encrypted_string varchar(40) );
Create success.
```

1) 원문의 길이가 8 의 배수가 아니기 때문에 에러가 발생한다.

```
iSQL> insert into t1 values( desencrypt( 'Altibase Client Query utility.',
'altibase' ) );
[ERR-2100D : Invalid length of the data type]
```

2) 원문의 길이는 8 의 배수이나, key 의 길이가 8 보다 짧아서 에러가 발생한다.

```
iSQL> insert into t1 values( desencrypt( 'Altibase Client Query utility...',
'alti4' ) );
[ERR-2100D : Invalid length of the data type]
```

3) 원문을 8 의 배수로 맞춰서 rpad 를 하고, key 의 길이를 8 로 맞춰서 rpad 한 다음 암호화한다.

```
iSQL> create or replace function my_encrypt( input_string in varchar(100),
key_string
in varchar(40) )
```

```

2 return varchar(100)
3 as
4   encrypted_string varchar(100);
5   pieces_of_eight INTEGER;
6 begin
7   pieces_of_eight := ((FLOOR(LENGTH(input_string)/8 + .9)) * 8);
8
9   encrypted_string := desencrypt( RPAD( input_string, pieces_of_eight),
10                                 RPAD( key_string, 8, '#' ) );
11  return encrypted_string;
12 end;
13 /
Create success.

```

- 4) key 의 길이를 8 로 맞춰서 rpad 한 다음 복호화한다. 복호화 후 나온 원문을 trim 한다.

```

iSQL> create or replace function my_decrypt( input_string in varchar(100),
key_string
in varchar(40) )
2 return varchar(100)
3 as
4   decrypted_string varchar(100);
5 begin
6   decrypted_string := desdecrypt( input_string,
7                                 RPAD( key_string, 8, '#' ) );
8
9   return trim(decrypted_string);
10 end;
11 /
Create success.

```

- 5) 원문의 길이가 8 의 배수가 아님. my\_encrypt 함수로 암호화 하여 저장한다.

```

iSQL> insert into t1 values( my_encrypt( 'Altibase Client Query utility',
'altibase' ) );
1 row inserted.

```

- 6) my\_decrypt 함수로 복호화

```

iSQL> select my_decrypt( encrypted_string, 'altibase' ) from t1;
MY_DECRYPT( ENCRYPTED_STRING, 'altibase'

```

---

```

Altibase Client Query utility.
1 row selected.

```

```

iSQL> delete from t1;
1 row deleted.

```

- 7) key 의 길이가 8 이 아님. my\_encrypt 함수로 암호화 하여 저장

```

iSQL> insert into t1 values( my_encrypt( 'Altibase Client Query utility...', '
'alti4' ) );

```

1 row inserted.

8) my\_decrypt 함수로 복호화

```
iSQL> select my_decrypt( encrypted_string, 'alti4' ) from t1;  
MY_DECRYPT( ENCRYPTED_STRING, 'alti4' )
```

---

Altibase Client Query utility...

1 row selected.



## 8. 산술 연산자

이 장에서는 SQL에서 사용하는 산술연산자들에 대해서 자세히 설명한다.

## 종류

### 산술 연산자의 종류

다음은 알티베이스에서 제공하는 산술 연산자에 대한 간단한 설명과 종류이다.

산술 연산자 범주	설명
단항 연산자(+)	명시적으로 양수를 나타낸다
단항 연산자(-)	입력 number의 부호를 반전시킨다
사칙 연산자	숫자와 숫자의 계산 결과를
연결 연산자	두 문자열을 합친다.

---

## 단항 연산자

---

### 단항 연산자 ( + )

#### 구문

$+ number$

#### 설명

명시적으로 입력된 값을 양수로 표시한다.

---

### 단항 연산자 ( - )

#### 구문

$- number$

#### 설명

입력한 number의 부호를 반전시킨다.

---

## 사칙 연산자

---

### 덧셈

#### 구문

$number1 + number2$

#### 설명

숫자와 숫자의 덧셈 결과를 돌려준다.

---

### 뺄셈

#### 구문

$number1 - number2$

#### 설명

숫자와 숫자의 뺄셈 결과를 돌려준다.

---

### 곱셈

#### 구문

$number1 \times number2$

#### 설명

숫자와 숫자의 곱셈 결과를 돌려준다.

---

### 나눗셈

#### 구문

$number1 / number2$

## 설명

숫자와 숫자의 나눗셈 결과를 돌려준다.

## 산술 연산자를 이용한 날짜 계산

### 구문

```
date [ + | - ] number  
date - date  
date [ + | - ] day (i일 후/전: i)  
date [ + | - ] hour (i시간 후/전: i/24) )  
date [ + | - ] minute (i분 후/전: i/(24*60) )  
date [ + | - ] second (i초 후/전: i/(24*60*60))
```

### 설명

날짜 + 숫자 = 날짜: 날짜에 일 수를 더한다.

날짜 - 숫자 = 날짜: 날짜에서 일 수를 뺀다.

날짜 - 날짜 = 일 수: 한 날짜에서 다른 날짜를 뺀다.

날짜 + 일수 = 날짜: 날짜에 일수를 더한다.

날짜 - 일수 = 날짜: 날짜에 일수를 뺀다.

날짜 + 시간 = 날짜: 날짜에 시간을 더한다.

날짜 - 시간 = 날짜: 날짜에 시간을 뺀다.

날짜 + 분 = 날짜: 날짜에 분을 더한다.

날짜 - 분 = 날짜: 날짜에 분을 뺀다.

날짜 + 초 = 날짜: 날짜에 초를 더한다.

날짜 - 초 = 날짜: 날짜에 초를 뺀다.

### 예제

〈질의〉 부서 A001에 있는 모든 사원의 이름 및 근무한 주 수를 출력하라.

```
iSQL> SELECT ename, (SYSDATE-join_date)/7 Weeks  
      FROM employee  
     WHERE dno = Byte'A001';  
          ENAME           WEEKS
```

```
HYCHOI           227.772775
HJMIN           260.487061
YHBAE           263.201346
3 rows selected.
```

〈질의〉 현재 시간에서 10 분 후의 시간을 출력하라.

```
iSQL> SELECT SYSDATE + (10/(24*60)) After_10minutes FROM dual;
AFTER_10MINUTES
```

---

```
2005/01/20 09:59:34
1 row selected.
```

## 연결 연산자

### 구문

*char1 || char2*

### 설명

두개의 문자열 *char1*과 *char2*를 합친다.

### 예제

〈질의〉 사원 이름과 직책사이에 ‘ is a ’를 삽입하여 하나의 칼럼으로 출력하라.

```
iSQL> SELECT ename || ' is a ' || emp_job Concatenation  
      FROM employee;  
      CONCATENATION
```

---

SWNO	is a CEO
HJNO	is a DESIGNER
HSCHOI	is a ENGINEER

20 rows selected.

## 변환 연산자

### 구문

**CAST (*expression* AS *data\_type*)**

### 설명

*expression* 의 타입을 명시적으로 주어진 *data\_type* 으로 변환한다.  
(blob, clob 타입을 제외한 변환 가능한 모든 타입을 지원한다.)

### 예제

〈질의〉 스트링값을 double 타입으로 변환하라.

```
iSQL> SELECT CAST('3.14159265359' AS DOUBLE) PI FROM dual;
PI
-----
3.14159265359
1 row selected.
```

## 9. 조건 연산자

이장에서는 SQL 문에서 사용하는 조건 연산자들에 대해서 예를 들어서 자세히 설명하고 있다.

## 종류

### 논리 연산자의 종류

다음은 알티베이스에서 제공하는 논리 연산자에 대한 간단한 설명과 종류이다.

논리 연산자 범주	설명
부정 (NOT)	입력된 값의 반대 결과를 돌려준다.
논리곱 (AND)	두 입력 값을 논리곱 연산한 결과를 돌려준다.
논리합 (OR)	두 입력 값을 논리합 연산한 결과를 돌려준다.

### 단항 비교 종류

다음은 알티베이스에서 제공하는 단항 비교에 대한 간단한 설명과 종류이다.

단항 비교 범주	설명
구간(BETWEEN) 조건	비교 조건의 일종으로 일정 범위 내의 값인지 아닌지 검사한다.
IN 조건	IN 조건은 좌측의 표현식이 우측의 표현식 중 하나 이상과 같은지 검사한다. NOT IN 조건은 좌측의 표현식이 우측의 모든 표현식과 같지 않은지를 검사한다.
LIKE 조건	패턴 일치 검사 조건으로 문자열을 사용해 패턴 일치를 검사한다.
NULL 조건	표현식이 널 값인지 아닌지 검사한다.
EXISTS 조건	부연질의의 결과가 적어도 하나 이상 존재하는지 검사한다.
UNIQUE Predicate	부연질의의 결과가 오직 하나의 레코드인지 검사한다.

## 논리 연산자

### 부정 (NOT)

#### 구문

NOT condition

#### 설명

입력 *condition*의 반대 결과를 돌려준다.

Condition	TRUE	FALSE	UNKNOWN
결과	FALSE	TRUE	UNKNOWN

#### 예제

〈질의〉 생일이 상반기(6 월 30 일) 이전인 사원을 제외한 사원들의 이름, 부서, 생일을 출력하라.

```
iSQL> SELECT ename, dno, birth
      FROM employee
     WHERE NOT birth < Byte'0630';
          ENAME      DNO      BIRTH
-----+
        HJNO      C002    1219
        KSKIM      D001    0730
        HYCHOI     A001    0822
        JDLEE      D001    0726
        KWKIM      C001    1102
        DIKIM      C002    1026
6 rows selected.
```

### 논리곱 (AND)

#### 구문

*condition1 AND condition2*

#### 설명

*condition1*과 *condition2*를 논리곱 연산한 결과를 돌려준다.

논리곱 연산의 결과는 다음과 같다.

Condition1 Condition2	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

## 예제

〈질의〉 엔지니어이면서 급여가 2000000 원 이상인 직원의 이름, 급여, 입사년을 출력하라.

```
iSQL> SELECT ename, salary, join_date
  FROM employee
 WHERE emp_job = 'ENGINEER' AND salary >= 2000000;
    ENAME          SALARY      JOIN_DATE
-----+
 HSCHOI        2000000  2000/01/11 00:00:00
 SJKIM         2500000  1999/12/20 00:00:00
 2 rows selected.
```

## 논리합 (OR)

### 구문

*condition1 OR condition2*

### 설명

*condition1*과 *condition2*를 논리합 연산한 결과를 돌려준다.

논리합 연산의 결과는 다음과 같다.

Condition1 Condition2	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

## 예제

〈질의〉 재고 수량이 20000 이상이거나 단가가 100000 원 이상인 데이터를 출력하라.

```
iSQL> SELECT *
  FROM goods
```

```
WHERE stock > 20000 OR price >= 100000;
GOODS.GNO    GOODS.GNAME          GOODS.GOODS_LOCATION
GOODS STOCK
```

---

```
GOODS.PRICE
```

---

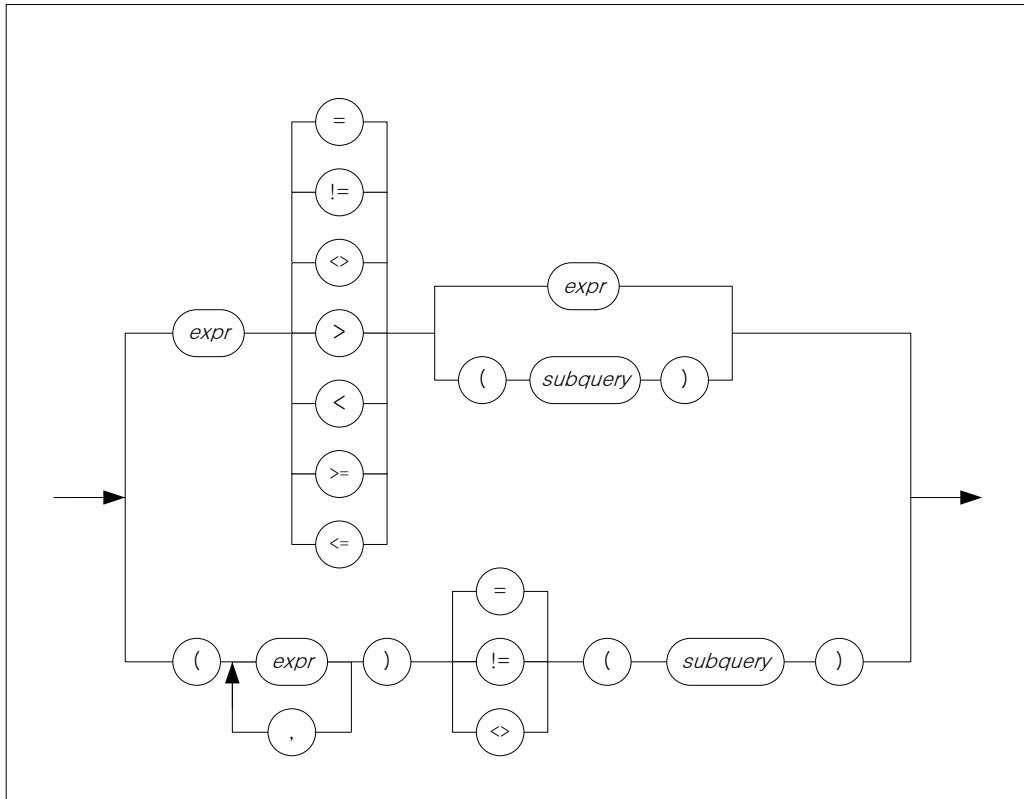
C111100001	IT-U950	FA0001	35000
7820.55			
D111100008	TM-U200	AC0006	61000
10000			
E111100004	M-190G	CE0001	88000
5638.76			
E111100012	M-U420	CE0003	43200
3566.78			
F111100001	AU-100	AC0010	10000
100000			

5 rows selected.

## 비교조건

### 구문

*simple\_comparison\_condition ::=*



### 설명

비교 조건은 비교 연산자를 기준으로 양쪽의 조건을 검토하여 참, 거짓을 판별하여 나타낸다.

좌측과 우측의 두 표현(expression)의 대소 비교 또는 동등 비교를 수행하는 조건절이다.

좌측이 둘 이상의 표현식(expression)인 경우 좌측의 표현식(expression) 수와 우측의 부연질의(subquery)의 대상 리스트의 표현식 수는 동일해야 한다.

우측의 부연질의 결과는 한 레코드여야 한다.

### 예제

〈질의〉 재고금액이 1 억원 이상인 상품의 품명, 보관수량, 원가, 재고금액을 출력하라. 재고금액은 보관수량 \* 원가이다.

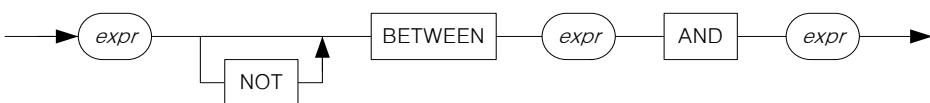
```
iSQL> SELECT gname, stock, price, stock*price stock_price
   FROM goods
 WHERE stock*price > 100000000;
      GNAME          STOCK       PRICE     STOCK_PRICE
-----+
IT-U950           35000    7820.55    273719250
TM-T88            10000    72000      720000000
TM-U950           8000     96200      769600000
.
.
.
11 rows selected.
```

## 단항 비교

### 구간 (BETWEEN) 조건

#### 구문

*between\_condition ::=*



#### 설명

비교 조건(comparison predicate)의 일종으로 일정 범위 내의 값인지 아닌지 검사한다. (지정한 값을 포함한다.)

'column1 between x1 and x2'는 'column1 >= x1 and column1 <= x2'와 논리적으로 동일하다.

#### 예제

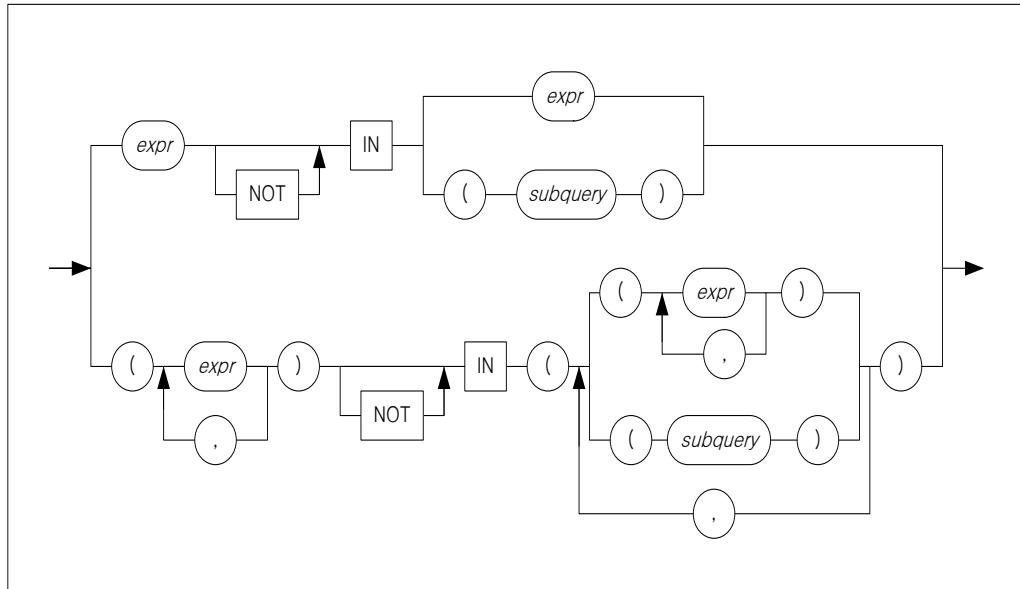
〈질의〉 재고금액이 100 만원 이상 1000 만원 이하인 상품의 품명, 보관수량, 원가, 재고금액을 출력하라. 재고금액은 보관수량 \* 원가이다.

```
iSQL> SELECT gname, stock, price, stock*price stock_price
      FROM goods
     WHERE stock*price BETWEEN 1000000 AND 10000000;
      GNAME          STOCK       PRICE      STOCK_PRICE
-----+-----+-----+-----+
      IM-310        100    98000    9800000
      IT-U200       1000   9455.21   9455210
      M-T245        900    2290.54   2061486
      M-180         1000   2300.55   2300550
      M-T153        900    2338.62   2104758
      M-T102       7890    966.99   7629551.1
      M-T500       5000   1000.54   5002700
      7 rows selected.
```

## IN 조건

### 구문

*in\_condition ::=*



### 설명

IN 조건은 ' $= ANY$ '와 동일한 predicate 으로 좌측의 표현식(expression)이 우측의 표현식(expression) 중 하나 이상과 같은지 검사한다. (지정된 목록 중의 값과 일치)

NOT IN 조건은 ' $\neq ALL$ '과 같은 predicate 으로 좌측의 표현식(expression)이 우측의 모든 표현식(expression)과 같지 않은지를 검사한다.

### 예제

〈질의〉 응용기술 팀 또는 마케팅 팀에서 일하고 있는 사원의 이름, 업무, 전화번호, 입사일을 출력하라.

```
iSQL> SELECT ename, emp_job, emp_tel, join_date  
FROM employee  
WHERE dno IN (BYTE'A001', Byte'C001');  
ENAME          EMP_JOB        EMP_TEL
```

---

JOIN\_DATE

---

HYCHOI

PROGRAMMER

0197853222

```

2000/09/09 00:00:00
HJMIN          MANAGER      0175221002
2000/01/24 00:00:00
YHBAE          PROGRAMMER   0167452000
2000/01/05 00:00:00
MSKIM          WEBMASTER    0114553206
2000/04/28 00:00:00
KWKIM          CEO          0187636550
JHSEOUNG       WEBMASTER    01195568840

```

6 rows selected.

\* 위 SQL 문에서 WHERE 절은 WHERE DNO = Byte'A001' or DNO = Byte'C001'을 의미한다.

〈질의〉 상품 C111100001 을 주문한 고객이름을 출력하라.

```

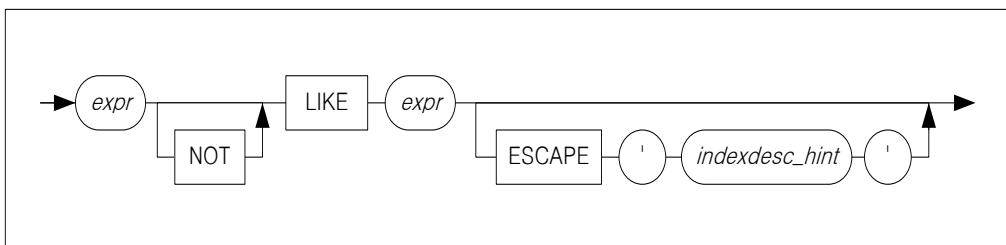
iSQL> SELECT DISTINCT customer.cname
      FROM customer
      WHERE customer.cno IN
        (SELECT orders.cno
         FROM orders
         WHERE orders.gno = Byte'C111100001');
CNAME
-----
YSKIM
DKKIM
IJLEE
CHLEE
4 rows selected.

```

## LIKE 조건

### 구문

*like\_condition ::=*



### 설명

패턴 일치 검사 조건으로 문자열은 ‘%’, 한 문자는 ‘\_’를 사용해 패턴을 검사한다. 패턴을 의미하는 특수 문자인 '%'와 '\_'를 특수 문자가 아닌 일반 문자로 패턴 일치를 검사할 경우 ESCAPE 절을 사용해 escape 문자를 정의하고 특수 문자 앞에 escape 문자를 덇붙인다.

## 예제

〈질의〉 성이 LEE 인 직원들의 사원번호, 이름, 부서번호, 전화번호 출력하라.

```
iSQL> SELECT eno, ename, dno, emp_tel  
      FROM employee  
     WHERE ename LIKE '%LEE%';  
ENO      ENAME          DNO    EMP_TEL  
-----  
 8       JDLEE        D001 0178829663  
 9       KMLEE        C002 0165293668  
12       MYLEE        F001 0174562330  
18       CHLEE        C002 01755231044  
4 rows selected.
```

〈질의〉 부서 이름에 밑줄( )이 있는 부서에 대한 모든 열을 출력하라.

```
iSQL> INSERT INTO department VALUES(BYTE'F002', 'HEAD_QUARTERS',  
      'YOUIDO', 100);  
1 row inserted.  
iSQL> SELECT * FROM department  
     WHERE dname LIKE '%₩_%' ESCAPE '₩';  
DEPARTMENT.DNO  DEPARTMENT.DNAME  
DEPARTMENT.DEP_LOCATION  DEPARTMENT.MGR_NO  
-----  
F002  HEAD_QUARTERS        YOUIDO      100  
1 row selected.
```

\* ESCAPE 옵션은 백슬래시 (\)를 ESCAPE 문자로 식별한다. 이 패턴에서 ESCAPE 문자가 밑줄( ) 앞에 있으므로 Altibase 는 밑줄을 리터럴로 해석한다.

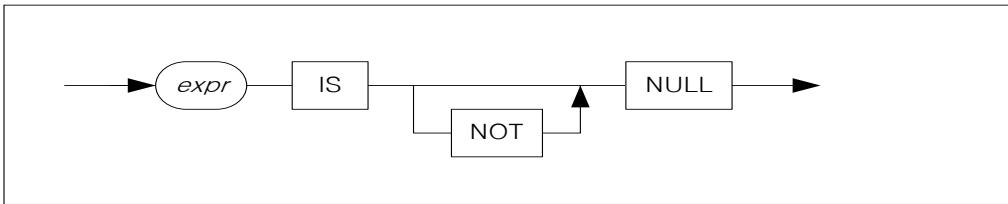
〈질의〉 이름의 두 번째 문자가 ‘S’인 모든 사원의 이름을 출력하라.

```
iSQL> SELECT ename  
      FROM employee  
     WHERE ename LIKE '_S%';  
ENAME  
-----  
HSCHOI  
KSKIM  
MSKIM  
3 rows selected.
```

## NULL 조건

### 구문

*null\_condition ::=*



### 설명

표현식(expression)이 널(NULL) 값인지 아닌지 검사한다.

### 예제

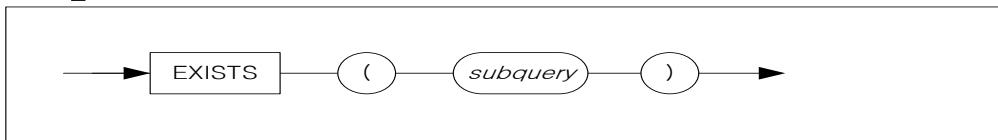
〈질의〉 생일이 입력 되지 않은 직원의 사원번호, 이름, 업무를 출력하라.

```
iSQL> SELECT eno, ename, emp_job  
      FROM employee  
     WHERE salary IS NULL;  
ENO      ENAME          EMP_JOB  
---  
1        SWNO            CEO  
8        JDLEE           MANAGER  
20       DIKIM           SALESMAN  
3 rows selected.
```

## EXISTS 조건

### 구문

*exists\_condition ::=*



### 설명

부연질의의 결과가 적어도 하나 이상 존재하는지 검사한다.

하나라도 존재하면 그 결과는 참(TRUE)을 돌려준다.

## 예제

〈질의〉 상품을 한 개 이상 주문한 고객번호를 출력하라. (먼저 부연질의에서는 주문 테이블에서 같은 고객번호이지만 다른 상품을 주문 하는 두 개의 row 를 찾는다. 즉, 한 고객이 한 개 이상의 상품을 주문한 것을 의미한다. 만약, 그러한 두개의 row 가 존재하면, EXISTS 의 값은 TRUE 이다. 이런 경우, 결과는 그 고객번호를 출력한다.)

```
iSQL> SELECT DISTINCT cno
  FROM orders a
 WHERE EXISTS
   (SELECT *
    FROM orders b
    WHERE a.cno = b.cno
      AND NOT(a.gno = b.gno));
CNO
-----
771215-1345471
761012-1220475
750625-1122143
740508-1332014
731231-1515123
730828-1201145
711111-1431202
700101-1001001
690209-1234567
9 rows selected.
```

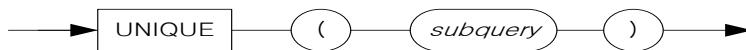
〈질의〉 모든 상품을 주문한 고객이름을 출력하라. (세 부분중 맨 아래 부분의 질의는 주문 테이블에서 고객이 주문한 상품들을 찾는 것이다. 가운데 부분은 그 고객이 주문하지 않은 상품을 찾는 것이다. 만약, 주문하지 않은 상품이 존재하지 않는다면 그 때의 고객의 이름을 출력한다.)

```
iSQL> SELECT customer cname
  FROM customer
 WHERE NOT EXISTS
   (SELECT *
    FROM goods
    WHERE NOT EXISTS
      (SELECT *
       FROM orders
       WHERE orders.cno = customer.cno
         AND orders.gno = goods.gno));
CNAME
-----
No rows selected.
```

## UNIQUE Predicate

### 구문

*unique\_condition ::=*



### 설명

부연질의의 결과가 오직 하나의 레코드인지 검사한다.

### 예제

〈질의〉 사원 중 'CHLEE'라는 이름이 한명만 있을 경우 사원들의 이름을 출력하라.

```
iSQL> SELECT ename
      FROM employee
      WHERE UNIQUE
        (SELECT *
         FROM employee
         WHERE ename = 'CHLEE');
ENAME
```

```
-----  
SWNO  
HJNO  
HSCHOI
```

20 rows selected.

〈질의〉 고객중 여성이 한명만 있을 경우 사원들의 이름을 출력하라.

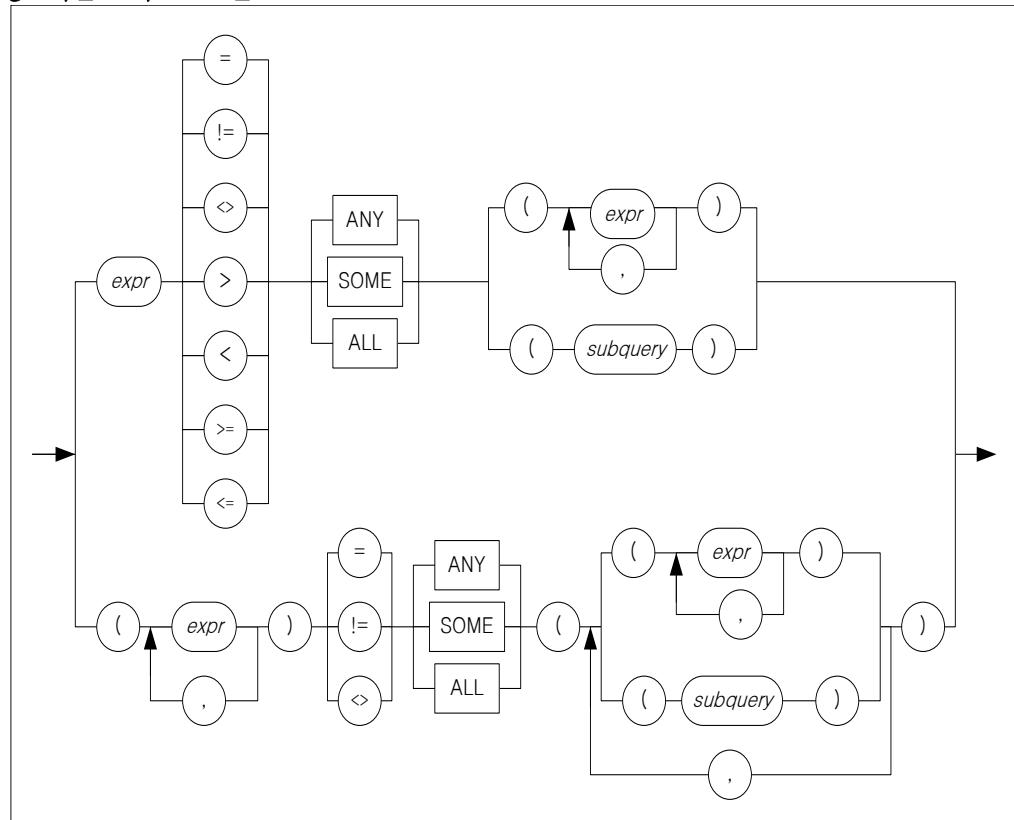
```
iSQL> SELECT ename
      FROM employee
      WHERE UNIQUE
        (SELECT *
         FROM customer
         WHERE SEX = 'F');
ENAME
```

```
-----  
No rows selected.
```

## 결과 비교 조건

### 구문

*group\_comparison\_condition ::=*



### 설명

- ANY / SOME  
표현식(expression) 또는 부연질의의 결과와 비교해 만족하는 값이 하나 이상 존재하면 참(TRUE)을 돌려준다.
- ALL  
표현식(expression) 또는 부연질의의 모든 결과와 비교해 만족하면 참(TRUE)을 돌려준다.

### 예제

〈질의〉 LEE 성을 가진 사원의 입사한 후 60 일 이후 받은 주문의 주문번호, 주문일자, 주문상태를 출력하라.

iSQL> SELECT ono, order\_date, processing

```
FROM orders
WHERE order_date < ANY
  (SELECT (join_date+60)
   FROM employee
   WHERE ename LIKE '%LEE%');

ONO          ORDER_DATE      PROCESSING
-----
0011290007    2000/11/29 00:00:00  C
0011290011    2000/11/29 00:00:00  D
0011290100    2000/11/29 00:00:00  D
0012100277    2000/12/10 00:00:00  C
0012300001    2000/12/01 00:00:00  P
5 rows selected.
```

# **Part IV**



## A. 부록 : 스키마

이 부록은 본 매뉴얼에서 전반적으로 사용된 예에 대한 스키마 및 기본 데이터에 대한 참조 정보이다.

## 예제 테이블 정보

### 목적

알티베이스 SQL 문법과 기능 설명을 위한 예제 테이블에 대한 정보를 제공한다.

### 제공 스크립트 파일

스키마 생성파일은 \$ALTIABSE\_HOME/sample/schema.sql 파일로 제공된다. 이 파일은 본 매뉴얼에서 사용된 테이블을 생성하고 예제 데이터를 삽입하는 파일이다. 따라서 본 매뉴얼에 기술되어 있는 예제를 실행하고자 한다면 제공된 파일을 수행후 예제를 따라해 볼 수 있다.

### 스키마

기 능: 고객, 주문관리

테이블: 사원, 부서, 고객, 주문, 상품

### 사원(employee) 테이블

기본 키: 사원번호(eno)

레코드 크기: 20 개

칼럼명	데이터 타입	설명	기타
eno	INTEGER	사원번호	PRIMARY KEY
ename	CHAR(20)	사원이름	NOT NULL
emp_job	CHAR(15)	직책	NULL
emp_tel	NIBBLE(15)	전화번호	NULL
dno	BYTE(2)	부서번호	NULL, INDEX, ASC
salary	NUMBER(10,2)	월급	NULL, DEFAULT 0
sex	CHAR(1)	성별	NOT NULL, 'M' or 'F', DEFAULT 'M'
birth	BYTE(2)	생일	NULL
join_date	DATE	입사날짜	NULL

status	CHAR(1)	지위	DEFAULT 'H'
--------	---------	----	-------------

### 부서(department) 테이블

기본 키: 부서번호(dno)

레코드 크기: 5 개

칼럼명	데이터 타입	설명	기타
dno	BYTE(2)	부서번호	PRIMARY KEY
dname	CHAR(20)	부서명	NOT NULL
dep_location	CHAR(9)	부서위치	NULL
mgr_no	INTEGER	관리자번호	NULL, BTREE INDEX, ASC

### 고객(customer) 테이블

기본 키: 주민등록번호(cno)

레코드 크기: 20 개

칼럼명	데이터 타입	설명	기타
cno	CHAR(14)	주민등록번호	PRIMARY KEY
cname	CHAR(20)	고객이름	NOT NULL
cus_job	CHAR(20)	직업	NULL
cus_tel	NIBBLE(15)	전화번호	NOT NULL
sex	CHAR(1)	성별	NOT NULL, 'M' or 'F', DEFAULT 'M'
birth	BYTE(2)	생일	NULL
post	BYTE(3)	우편번호	NULL
address	CHAR(60)	주소	NULL

### 주문(orders) 테이블

기본 키: 주문일자 + 주문번호 (ono, order\_date)

레코드 크기: 30 개

칼럼명	데이터 타입	설명	기타
ono	NIBBLE(10)	주문번호	PRIMARY KEY
order_date	DATE	주문일자	PRIMARY KEY
eno	INTEGER	판매사원	NOT NULL, BTREE INDEX, ASC
cno	CHAR(14)	고객주민번호	NOT NULL,

			BTREE INDEX, DESC
gno	BYTE(5)	상품번호	NOT NULL, INDEX, ASC
qty	INTEGER	주문수량	NULL, DEFAULT 1
arrival_date	DATE	도착예정일자	NULL
processing	CHAR(1)	주문상태	NULL, O: ORDER, R: PREPARE, D: DELIVERY, C: COMPLETE, DEFALT 'O'

### 상품(goods) 테이블

기본 키: 상품번호(gno)

레코드 크기: 30 개

칼럼명	데이터 타입	설명	기타
gno	BYTE(5)	상품번호	PRIMARY KEY
gname	CHAR(20)	상품이름	NOT NULL, UNIQUE
goods_location	CHAR(9)	보관위치	NULL
stock	INTEGER	보관수량	NULL, DEFAULT 0
price	NUMERIC(10,2)	원가	NULL

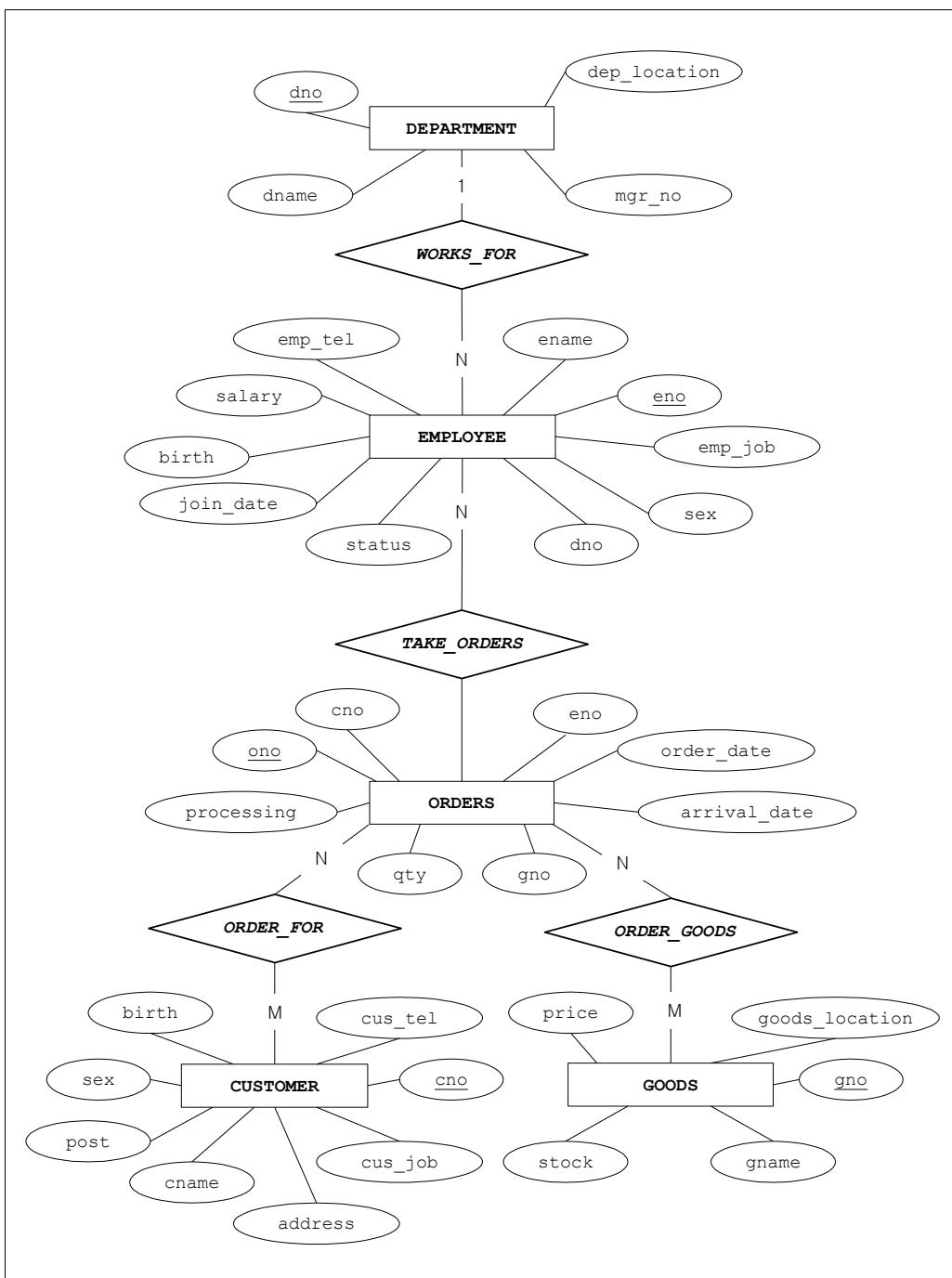
### dual 테이블

레코드 크기: 1 개

칼럼명	데이터 타입	설명	기타
x	CHAR(1)		

## E-R 다이어그램과 샘플 데이터

### E-R 다이어그램



## 샘플 데이터

### 사원테이블

```
iSQL> SELECT * FROM employee;  
EMPLOYEE,ENO EMPLOYEE,ENAME EMPLOYEE,EMP_JOB EMPLOYEE,EMP_TEL
```

```
EMPLOYEE,DNO EMPLOYEE,SALARY EMPLOYEE,SEX EMPLOYEE,BIRTH
```

```
EMPLOYEE,JOIN_DATE EMPLOYEE,STATUS
```

1	SWNO	CEO	01195662365
	M	R	
2	HJNO	DESIGNER	0113654540
C002	1500000	F 1219	
1999/11/18	00:00:00	H	
3	HSCHOI	ENGINEER	0162581369
D001	2000000	M 0226	
2000/01/11	00:00:00	H	
4	KSKIM	ENGINEER	0182563984
D001	1800000	M 0730	
	H		
5	SJKIM	ENGINEER	01145582310
D001	2500000	M	
1999/12/20	00:00:00	H	
6	HYCHOI	PROGRAMMER	0197853222
A001	1700000	M 0822	
2000/09/09	00:00:00	H	
7	HJMIN	MANAGER	0175221002
A001	500000	M 0417	
2000/01/24	00:00:00	H	
8	JDLEE	MANAGER	0178829663
D001		M 0726	
1999/11/29	00:00:00	H	
9	KMLEE	PLANER	0165293668
C002	1200000	M 0102	
2000/06/14	00:00:00	H	
10	YHBAE	PROGRAMMER	0167452000
A001	4000000	F 0213	
2000/01/05	00:00:00	H	
11	MSKIM	WEBMASTER	0114553206
C001	2750000	M	
2000/04/28	00:00:00	H	
12	MYLEE	SALESMAN	0174562330
F001	1890000	F 0211	
1999/12/14	00:00:00	H	
13	KWKIM	CEO	0187636550
C001	980000	M 1102	
	H		
14	KCJUNG	CEO	0197664120

```

D001 2003000 M
      H
15       JHSEOUNG WEBMASTER      01195568840
C001 1000000 M 0514
      H
16       JHCHOI    DESIGNER      0195562100
C002 2300000 F 0509
      H
17       DIKIM     CEO          0165293886
C002 1400000 M 1026
2000/05/07 00:00:00 H
18       CHLEE     PLANER      01755231044
C002 1900000 M
2000/10/30 00:00:00 H
19       KMKIM     SALESMAN    0185698550
F001 1800000 M
2000/11/18 00:00:00 H
20       DIKIM     SALESMAN    01154112366
F001     M
2000/11/18 00:00:00 H
selected row count [20]

```

## 부서테이블

```

iSQL> SELECT * FROM department;
DEPARTMENT,DNO      DEPARTMENT,DNAME           DEPARTMENT,DEP_LOCATION
DEPARTMENT,MGR_NO

A001 응용기술팀      마포      1
D001 엔진개발팀      여의도    10
C001 마케팅팀        강남      9
C002 기획관리팀      강남      15
F001 영업팀          신촌      9
5 rows selected.

```

## 고객테이블

```

iSQL> SELECT * FROM customer;
CUSTOMER,CNO      CUSTOMER,CNAME           CUSTOMER,CUS_JOB
CUSTOMER,CUS_TEL      CUSTOMER,SEX  CUSTOMER,BIRTH  CUSTOMER,POST
CUSTOMER,ADDRESS

730828-1201145 CHLEE      ENGINEER
0514685282          M 0828 601033
부산 동구 수정 3
771215-1345471 YSKIM      DOCTOR
0232421211          M 1215 121011
서울 마포구 아현 1
711111-1431202 DJKIM      DESIGNER
023442542           M 1111 135010

```

서울 강남구 논현동  
 720305-2101114 JHPARK ENGINEER  
 022326393 F 0305 121758  
 서울 마포구 공덕2 제일빌딩  
 761012-1220475 BSYOUN WEBMASTER  
 0233452141 M 1012 121021  
 서울 마포구 공덕1  
 690209-1234567 IJLEE WEBPD  
 025743215 M 0209 136751  
 서울 성북구 돈암2 한신아파트  
 731225-2402221 JHCHOI PLANER  
 023143366 F 1225 156772  
 서울 동작구 사당2 극동아파트  
 730801-1101115 HYCHOI PD  
 024721114 M 0801 135747  
 서울 강남구 신사 성도빌딩  
 600211-2417214 MYLEE DESIGNER  
 0512543734 F 0211 600033  
 부산 중구 광복3  
 620815-1724174 KSKIM  
 0516232256 M 0815 608703  
 부산 남구 대연3 부산시차량등록사업소  
 700101-1001001 LSPARK MANAGER  
 027664545 M 0101 142704  
 서울 강북구 미아9 성바오로딸수도회  
 670905-2101013 DHCHO BANKER  
 023343214 F 0905 152761  
 서울 구로구 구로1 구로주공아파트  
 791230-2114547 YDPARK ENGINEER  
 022320119 F 1230 153600  
 서울 금천구 서울구로우체국사서함  
 740508-1332014 DHKIM BANKER  
 024720112 M 0508 135740  
 서울 강남구 삼성1 강남병원  
 750625-1122143 DKKIM MANAGER  
 0518064398 M 0625 606796  
 부산 영도구 동삼2 한국해양대학교  
 781225-1333044 SMCHO PLANER  
 027544147 M 1225 157703  
 서울 강서구 화곡6 강서의료보험조합  
 761001-1000001 JHKIM  
 023543541 M 1001 157717  
 서울 강서구 등촌2 국군수도통합병원  
 740419-2146506 JHKIM ENGINEER  
 024560207 F 0419 138701  
 서울 송파구 가락1 가락동농수산물시장  
 731231-1515123 DJKIM  
 022371234 M 1231 138742  
 서울 송파구 신천 서울시교통회관건물  
 700405-2321123 DKHAN WEBMASTER  
 024560002 F 0405 135757  
 서울 강남구 삼성1 종합무역센타빌딩

20 rows selected.

## 주문테이블

```
iSQL> SELECT * FROM orders;
```

ORDERS.ONO	ORDERS.ORDER_DATE	ORDERS.ENO	ORDERS.CNO
------------	-------------------	------------	------------

ORDERS.GNO	ORDERS.QTY	ORDERS.ARRIVAL_DATE	ORDERS.PROCESSING	
0011290007		2000/11/29 00:00:00	12	711111-1431202
A111100002	70	2000/12/02 00:00:00	C	
0011290011		2000/11/29 00:00:00	12	761001-1000001
E111100001	1000	2000/12/05 00:00:00	D	
0011290100		2000/11/29 00:00:00	19	700101-1001001
E111100001	500	2000/12/07 00:00:00	D	
0012100277		2000/12/10 00:00:00	19	761012-1220475
D111100008	2500	2000/12/12 00:00:00	C	
0012300001		2000/12/01 00:00:00	19	730828-1201145
D111100004	1000	2001/01/02 00:00:00	P	
0012300002		2000/12/29 00:00:00	12	771215-1345471
C111100001	300	2001/01/02 00:00:00	P	
0012300003		2000/12/29 00:00:00	20	740508-1332014
E111100002	900	2001/01/02 00:00:00	P	
0012300004		2000/12/30 00:00:00	20	750625-1122143
D111100002	1000	2001/01/02 00:00:00	P	
0012300005		2000/12/30 00:00:00	19	720305-2101114
D111100008	4000	2001/01/02 00:00:00	P	
0012300006		2000/12/30 00:00:00	20	791230-2114547
A111100002	20	2001/01/02 00:00:00	P	
0012300007		2000/12/30 00:00:00	12	731225-2402221
D111100002	2500	2001/01/02 00:00:00	P	
0012300008		2000/12/30 00:00:00	20	700101-1001001
D111100011	300	2001/01/02 00:00:00	P	
0012300009		2000/12/30 00:00:00	20	731231-1515123
D111100003	500	2001/01/02 00:00:00	P	
0012300010		2000/12/30 00:00:00	19	781225-1333044
D111100010	2000	2001/01/02 00:00:00	P	
0012300011		2000/12/30 00:00:00	20	750625-1122143
C111100001	1000	2001/01/02 00:00:00	P	
0012300012		2000/12/30 00:00:00	12	711111-1431202
E111100012	1300	2001/01/02 00:00:00	P	
0012300013		2000/12/30 00:00:00	20	690209-1234567
C111100001	5000	2001/01/02 00:00:00	P	
0012300014		2000/12/30 00:00:00	12	670905-2101013
F111100001	800	2001/01/02 00:00:00	P	
0012310001		2000/12/31 00:00:00	20	750625-1122143
A111100002	50	2000/12/09 00:00:00	O	
0012310002		2000/12/31 00:00:00	12	620815-1724174
D111100008	10000	2001/01/03 00:00:00	O	
0012310003		2000/12/31 00:00:00	20	740419-2146506
E111100009	1500	2001/01/03 00:00:00	O	
0012310004		2000/12/31 00:00:00	19	761012-1220475

E111100010	5000	2001/12/08 00:00:00	O	
0012310005		2000/12/31 00:00:00	20	740508-1332014
E111100007	940	2001/01/03 00:00:00	O	
0012310006		2000/12/31 00:00:00	20	771215-1345471
D111100004	500	2001/01/03 00:00:00	O	
0012310007		2000/12/31 00:00:00	12	731231-1515123
E111100012	1400	2001/01/03 00:00:00	O	
0012310008		2000/12/31 00:00:00	19	730828-1201145
D111100003	100	2001/01/03 00:00:00	O	
0012310009		2000/12/31 00:00:00	12	761012-1220475
E111100013	500	2001/01/03 00:00:00	O	
0012310010		2000/12/31 00:00:00	20	690209-1234567
D111100010	1500	2001/01/03 00:00:00	O	
0012310011		2000/12/31 00:00:00	19	750625-1122143
E111100012	10000	2001/01/03 00:00:00	O	
0012310012		2000/12/31 00:00:00	19	730828-1201145
C111100001	250	2001/01/03 00:00:00	O	

30 rows selected.

## 상품테이블

```
iSQL> SELECT * FROM goods;
GOODS.GNO  GOODS.GNAME          GOODS.GOODS_LOCATION  GOODS STOCK
```

---

### GOODS.PRICE

---

A111100001	IM-300	AC0001	1000
	78000		
A111100002	IM-310	DD0001	100
	98000		
B111100001	NT-H5000	AC0002	780
	35800		
C111100001	IT-U950	FA0001	35000
	7820.55		
C111100002	IT-U200	AC0003	1000
	9455.21		
D111100001	TM-H5000	AC0004	7800
	12000		
D111100002	TM-T88	BF0001	10000
	72000		
D111100003	TM-L60	BF0002	650
	45100		
D111100004	TM-U950	DD0002	8000
	96200		
D111100005	TM-U925	AC0005	9800
	23000		
D111100006	TM-U375	EB0001	1200
	57400		
D111100007	TM-U325	EB0002	20000
	84500		
D111100008	TM-U200	AC0006	61000

```
10000
D111100009 TM-U300          DD0003 9000
50000
D111100010 TM-U590          DD0004 7900
36800
D111100011 TM-U295          FA0002 1000
45600
E111100001 M-T245          AC0007 900
2290.54
E111100002 M-150           FD0001 4300
7527.35
E111100003 M-180           BF0003 1000
2300.55
E111100004 M-190G          CE0001 88000
5638.76
E111100005 M-U310          CE0002 11200
1450.5
E111100006 M-T153          FD0002 900
2338.62
E111100007 M-T102          BF0004 7890
966.99
E111100008 M-T500          EB0003 5000
1000.54
E111100009 M-T300          FA0003 7000
3099.88
E111100010 M-T260          AC0008 4000
9200.5
E111100011 M-780           AC0009 9800
9832.98
E111100012 M-U420          CE0003 43200
3566.78
E111100013 M-U290          FD0003 12000
1295.44
F111100001 AU-100           AC0010 10000
100000
30 rows selected.
```

## DUAL 테이블

```
iSQL> SELECT * FROM dual;
```

```
DUAL.X
```

---

```
X
```

```
selected row count [1]
```

## Altibase 객체들의 최대값

아래의 표는 Altibase 객체들에 대한 최대 값을 정리한 내용으로 다음과 같다.

구분	객체의 최대값	비고
DB_NAME 길이	128	
OBJECT 길이	40	
테이블스페이스 개수	64K (시스템 TBS 포함)	DB에 대한 TBS 최대 개수
데이터 파일 개수	1,024	TBS에 대한 데이터 파일 최대 개수
	4,294,967,295 ( $2^{32}-1$ )이다. 그러나 TBS가 64K이고, TBS당 데이터 파일의 개수가 1,024로 제한되기 때문에 실제는 67,108,864 (64K * 1024)이다.	DB에 대한 데이터 파일 최대 개수
사용자 개수	2,147,483,638(시스템 사용자 포함)	DB에 대한 사용자 최대 개수
테이블 개수	2,097,151(메타 테이블 포함)	DB에 대한 테이블 최대 개수
인덱스 개수	64	테이블에 대한 인덱스 최대 개수
컬럼 개수	1,024	테이블에 대한 컬럼 최대 개수
	32	인덱스에 대한 컬럼 최대 개수
로우 개수	무한 개수	테이블에 대한 로우 최대 개수
파티션 개수	2,147,483,638(시스템 전체)	테이블에 대한 파티션 최대 개수
제약사항 개수	2,147,483,638(시스템 전체)	칼럼에 대한 제약사항 최대 개수

## 찾아보기

7

객체 이름 생성 규칙	6
객체 접근 권한	218
결과 비교 조건	419
공간 데이터 타입	56
구간 (BETWEEN) 조건	412
그룹 함수	327

L

날짜 함수	364
날짜형 데이터 타입	35
날짜형 데이터 형식	28, 35
논리 연산자	407
논리곱	407
논리연산자의 종류	406
논리합	408

M

단항 연산자	399
데이터 정의어(DDL)	9
데이터 제어어(DCL)	11
데이터 조작어(DML)	10
데이터 형식 변환	17
데이터형의 종류	14
데이터 타입 변환과 호환	16
디렉토리	6

O

문자 함수	346
문자형 데이터 타입	19

R

변환 연산자	404
--------	-----

변환 함수 ..... 374

부연질의 ..... 4

부정 ..... 407

뷰 ..... 6

비교조건 ..... 410

S

사용자 ..... 6

사칙연산자 ..... 400

산술 연산자의 종류 ..... 398

숫자 함수 ..... 331

숫자형 데이터 타입 ..... 22

시노님 ..... 6

시스템 접근 권한 ..... 213

시스템 제어문 ..... 11

시퀀스 ..... 6

T

알티베이스 객체 ..... 6

암호화 함수 ..... 392

연결 연산자 ..... 403

연산자 우선순위 ..... 322

예약어 ..... 7

이중화 ..... 6

이진 데이터 타입 ..... 48

인덱스 ..... 6

X

작업 제어문 ..... 11

저장 프로시저 ..... 6

제약조건 ..... 6

주석 ..... 5

중첩 함수 ..... 391

지원 언어 .....	5
집합 연산자 .....	317
<b>E</b>	
테이블 .....	6
테이블스페이스 .....	6
트랜잭션 제어문 .....	11
트리거 .....	6
<b>S</b>	
함수 .....	326
합집합 .....	318
<b>A</b>	
ABS function .....	331
ACOS function .....	331
ADD TABLE 절 .....	68
ADD_MONTHS function .....	364
aggregate functions .....	327
ALL PRIVILEGES 절 .....	218
ALTER REPLICATION statement .....	67
ALTER SEQUENCE statement .....	72
ALTER SESSION statement .....	300
ALTER SYSTEM statement .....	303
ALTER TABLE statement .....	75
ALTER TABLESPACE statement .....	92
ALTER TRIGGER statement .....	100
ALTER VIEW statement .....	104
Altibase object .....	6
AND condition .....	407
AS SELECT .....	152
ASC 절 .....	117
ASCII function .....	346
ASIN function .....	332
ATAN function .....	332
ATAN2 function .....	333
ATLER DATABASE statement .....	59
ATLER INDEX statement .....	65
ATLER USER statement .....	102
autoextend_clause .....	164
AVG function .....	327
<b>B</b>	
BETWEEN condition .....	412
BIGINT datatype .....	22
BIN_TO_NUM function .....	374
BINARY function .....	346
BIT datatype .....	50
BITAND .....	343
BITNOT .....	345
BITOR .....	344
BITXOR .....	344
BLOB datatype .....	53
BTREE .....	117
BYTE datatype .....	48
<b>C</b>	
CACHE 절 .....	73, 130
CASCADE .....	150, 203, 209
CASCADE CONSTRAINTS .....	232
CASE WHEN function .....	381
CASE2 function .....	381
cast .....	404
CBC(Cipher Block Chaining) .....	392
CEIL function .....	333
CHAR datatype .....	19
CHAR_LENGTH function .....	347
character functions .....	346
CHARACTER_LENGTH function .....	347
CHR function .....	348
CLOB datatype .....	54
CNF hint .....	265

comment.....	5
COMMIT statement .....	305
comparison conditions .....	410
CONCAT function .....	349
concatenation operator .....	403
CONNECT BY 절 .....	262
constraint.....	6
conversion functions .....	374
COS function.....	334
COSH function .....	334
COST hint .....	265
COUNT function .....	327
CREATE DATABASE LINK statement .....	108
CREATE DATABASE statement.....	106
CREATE DIRECTORY statement .....	113
CREATE INDEX statement .....	115
CREATE REPLICATION statement .....	125
CREATE SEQUENCE statement .....	129
CREATE SYNONYM statement.....	137
CREATE TABLE statement.....	141
CREATE TABLESPACE statement .....	162
CREATE TEMPORARY TABLESPACE statement .....	167
CREATE TRIGGER statement .....	178
CREATE USER statement .....	185
CREATE VIEW statement .....	188
CYCLE.....	73, 130
<b>D</b>	
DATE datatype .....	35
date format model.....	28, 35
DATEADD function .....	364
DATEDIFF function .....	365
DATENAME function .....	367
DATEPART function .....	368
datetime functions .....	364
DECIMAL datatype .....	22
DECODE function.....	383
DEFAULT 절.....	87
DELETE statement.....	238
DEQUEUE statement .....	297
DES(Data Encryption Standard) .....	392
DESC 절 .....	117
DESDECRYPT .....	392
DESENCRYPT .....	392
DIGEST function .....	384
DIGITS function.....	350
directory.....	6
DISABLE .....	83
DNF hint.....	265
DOUBLE datatype .....	23
DROP 절 .....	82
DROP CONSTRAINT 절 .....	82
DROP DATABASE LINK statement .....	194
DROP DATABASE statement .....	192
DROP DIRECTORY statement.....	196
DROP INDEX statement.....	197
DROP PRIMARY KEY.....	82
DROP QUEUE statement .....	198
DROP REPLICATION statement.....	199
DROP SEQUENCE statement .....	200
DROP SYNONYM statement .....	201
DROP TABLE 절 .....	68
DROP TABLE statement .....	203
DROP TABLESPACE statement .....	205
DROP TRIGGER statement.....	208
DROP UNIQUE .....	82
DROP USER statement .....	209
DROP VIEW statement .....	211

DUMP function ..... 385

## E

ENABLE ..... 83  
encryption function ..... 392  
ENQUEUE statement ..... 296  
EXCLUSIVE lock mode ..... 248  
execution plan tree ..... 4  
EXISTS ..... 416  
EXP function ..... 335  
EXTENTSIZE 절 ..... 163  
EXTRACT function ..... 368

## F

FLOAT datatype ..... 23  
FLOOR function ..... 335  
FOR UPDATE 절 ..... 264  
FORCE ..... 189  
force\_clause ..... 149  
FULL SCAN ..... 289  
FULL SCAN hint ..... 241, 264  
functions ..... 326

## G

Geometry Datatype ..... 56  
GRANT statement ..... 212  
GREATEST function ..... 385  
GROUP BUCKET COUNT hint ..... 266  
GROUP BY 절 ..... 263

## H

HASH BUCKET COUNT hint ..... 266  
HAVING 조건 ..... 263  
HEX\_TO\_NUM function ..... 375  
Hierarchical query 절 ..... 261  
HINTS ..... 241, 264, 289

## I

IGNORE LOOP ..... 263  
IN ..... 413  
INCREMENT BY 절 ..... 73, 130  
index ..... 6  
INDEX ASC hint ..... 241, 265  
INDEX DESC hint ..... 241, 265  
INDEX hint ..... 241, 264  
index\_attribute\_clause ..... 118  
index\_partitioning\_clause ..... 117  
INITCAP function ..... 350  
In-line view ..... 261  
INSERT statement ..... 243  
INSTR function ..... 351  
INTEGER datatype ..... 24  
Internal Tuple ..... 268  
INTERSECT 집합 연산자 ..... 263  
INTERSECT set operator ..... 320  
ISOLATION LEVEL 절 ..... 311

## K

keyword ..... 7  
KO16KSC5601 ..... 5

## L

LAST\_DAY function ..... 370  
LEAST function ..... 386  
LENGTH function ..... 347  
LENGTHB function ..... 354  
LEVEL ..... 262  
LIKE ..... 414  
LIMIT 절 ..... 263  
LN function ..... 336  
LOB 데이터 타입 ..... 52  
LOB(Large Object) ..... 52

LOCALUNIQUE	148
LOCK TABLE statement	247
LOG function	336
logging_clause	149
logical conditions	407
LOWER function	352
LPAD function	352
LTRIM function	353

## M

MAX function	328
MAXROWS	83, 150
maxsize_clause	165
MAXVALUE 절	73, 130
MIN function	328
MINUS 집합 연산자	263
MINUS set operator	321
MINVALUE 절	73, 130
MOD function	337
modify_checkpoint_path_clause	95
MONTHS_BETWEEN function	369
MS949	5

## N

Naming Rule	6
NEXT_DAY function	371
NIBBLE datatype	49
NO ACTION	150
NO FORCE	189
NO INDEX hint	241, 265
NOT condition	407
NOT NULL	83, 149
NOWAIT	248
NULL	83
NULL	16
NULL	149

NULL	416
NUMBER datatype	25
number functions	331
NUMERIC datatype	26
NVL function	387
NVL2 function	388

## O

object privilege	213, 218
OCT_TO_NUM function	375
OCTET_LENGTH function	354
operators	398
OR condition	408
OR REPLACE	189
ORDER BY 절	263
ORDERED hint	265
OUTER JOIN 절	261

## P

parallel_clause	118, 149
PERSISTENT 인덱스 변경	66
PERSISTENT 절	83, 118
PERSISTENT INDEX	149
POSITION function	351
POWER function	337
PRIMARY KEY	148
prior 연산자	262
private synonym	138
public synonym	138
PUSH_PRED hint	265

## Q

QUICKSTART	68
------------	----

## R

RANDOM function	338
-----------------	-----

READ COMMITTED .....	311	set operators .....	317
READ ONLY .....	311	SET TRANSACTION statement .....	311
READ WRITE.....	311	SHARE lock mode .....	248
REAL datatype.....	27	SHARE ROW EXCLUSIVE lock mode .....	248
RENAME COLUMN 절.....	82	SHARE UPDATE lock mode .....	248
RENAME TABLE statement.....	228	SHIFT-JIS .....	6
RENAME TO 절 .....	83	SIGN function.....	340
REPEATABLE READ .....	312	SIN function.....	340
REPLACE2 function .....	354	SINH function.....	341
REPLICATE function .....	361	SIZEOF function.....	357
replication .....	6	SMALLINT datatype.....	27
reserved word .....	7	SQL .....	4
REVERSE_STR function .....	361	SQL 분류 .....	9
REVOKE statement .....	230	SQRT function.....	341
ROLLBACK statement .....	308	START .....	68
ROUND .....	369	START FLUSHER .....	303
ROUND function .....	339	START WITH 절 .....	130, 261
ROW EXCLUSIVE lock mode .....	248	STDDEV function .....	329
ROW SHARE lock mode.....	248	STOP .....	68
row_movement_clause .....	152	STOP FLUSHER .....	303
ROWNUM .....	386	stored procedure.....	6
RPAD function.....	355	string functions .....	346
RTREE .....	117	STUFF function .....	362
RTRIM function.....	356	subquery.....	4
RULE hint .....	265	SUBSTR function .....	357
<b>S</b>		SUBSTRB function .....	358
SAVEPOINT statement .....	306	SUBSTRING function .....	357
SELECT statement.....	252	SUM function .....	329
select_list.....	261	SYNC .....	67
SENDMSG.....	388	synonym .....	6
sequence.....	6	SYSDATE function .....	371, 372
SERIALIZABLE.....	312	system privilege .....	213
SET 절 .....	290		
SET BUCKET COUNT hint.....	266		

## T

table .....	6
table_constraint.....	149
table_partitioning_clauses .....	150
tablespace .....	6
TAN function.....	342
TANH function .....	342
TIMESTAMP constraint .....	84
TO_NUMBER function .....	379
TO_BIN function .....	374
TO_CHAR(datetime) function .....	376
TO_CHAR(number) function.....	376
TO_DATE function.....	377
TO_HEX function .....	378
TO_OCT function.....	380
TRANSLATE function .....	358
trigger.....	6
TRIM function .....	359
TRUNC(date) function.....	372
TRUNC(number) function.....	343
TRUNCATE TABLE statement.....	234
Tuple set .....	268

## U

UNION ALL set operator .....	319
UNION set operator .....	318
UNION(ALL) 집합 연산자 .....	263
UNIQUE .....	148, 418
UNIQUE 절 .....	117

UPDATE statement.....	287, 293
UPPER function .....	360
US7ASCII .....	5
USE_HASH hint .....	266
USE_MERGE hint.....	266
USE_NL hint .....	266
USE_SORT hint .....	266
user.....	6
USER_ID .....	390
USER_NAME .....	390
using_index_clause .....	149
UTF8 .....	6

## V

VARBIT datatype .....	51
VARCHAR datatype.....	20
VARCHAR FIXED .....	20
VARCHAR VARIABLE .....	20
VARIANCE function.....	330
view .....	6

## W

WAIT .....	248
where_clause .....	68, 261
WITH GRANT OPTION.....	218
WITH READ ONLY .....	189

## Z

ZHS16CGB231280.....	6
ZHT16BIG5 .....	6