Introduction TO SQL

Altibase Education Center

ALTIBASE

1. ALTIBASE CONCEPT

1. ALTIBASE HDB
2. 관계형 데이터모델
3. ISQL

1.1 ALTIBASE HDB

ALTIBASE HDB

Hybrid DBMS

비즈니스 변화

- 새로운 비즈니스 모델 등장
- 고객 중심의 환경
- 정보 흐름의 가속화
- 글로벌 경쟁 시대

IT 인프라 변화

- Memory 가격의 지속적인 하락
  - 대용량 Memory 탑재 서버 출현
- 통신 장비 속도의 지속적인 발전
  - M Byte → G Byte 전송 속도
  - 트랜잭션의 대용량화
- Mobile Device 보급률 확대

IT 환경 분석

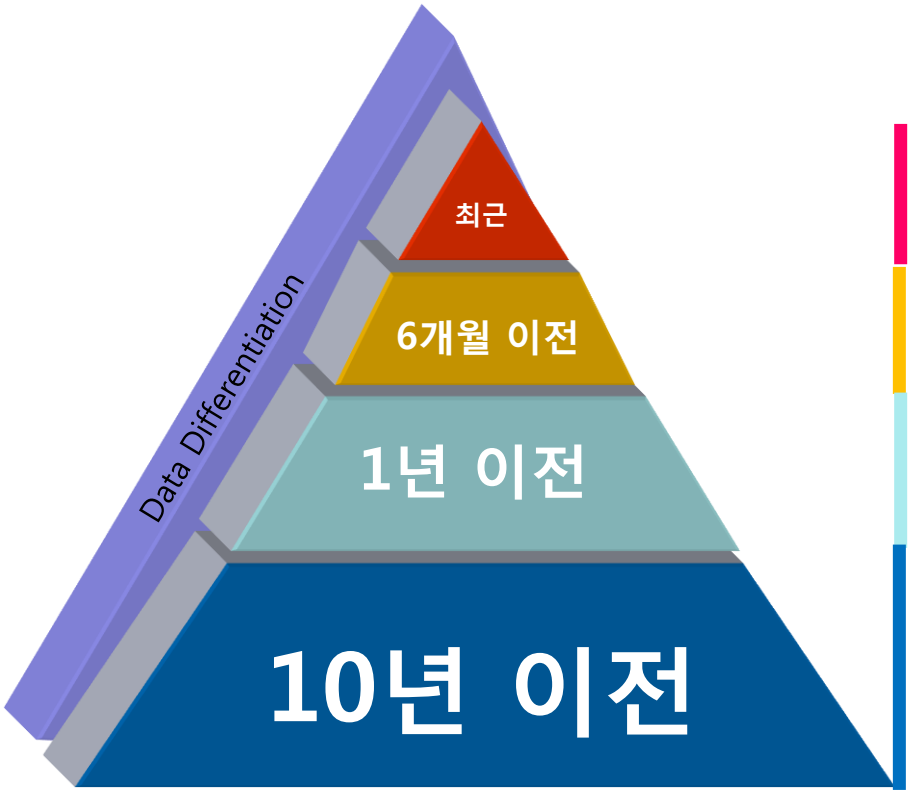
- 디지털화
- 유비쿼터스 컴퓨팅 환경
- 데이터 홍수

요구사항 변화

고성능 데이터 처리 **대용량 데이터 처리**

데이터 차별화

❖ 시간의 흐름에 따라 데이터의 속성은 변함



데이터 속성 변화	시스템 요구 사항
HOT DATA, 실시간 처리가 요구, 접근빈도 80%	
WARM DATA, 준 실시간 처리 요구, 접근빈도 10%	
TEPID DATA, 대용량 처리, 접근빈도 7%	
COLD DATA, 대용량 이력 보관용, 접근빈도 3%	

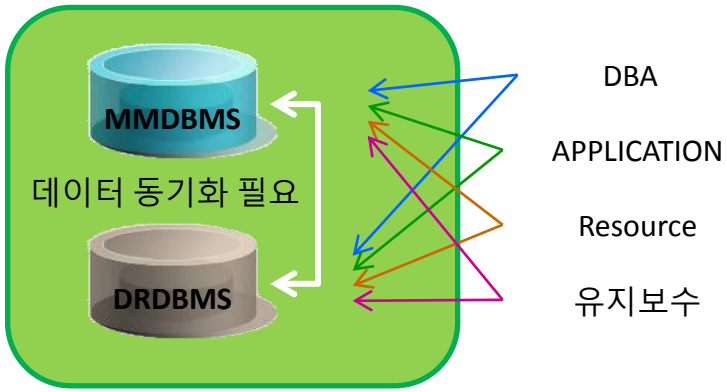
효율적인 데이터 관리를 위한
Data 차별화 필요
(Hot Data와 Cold Data 구분)

IDC 조사기록 : Worldwide Disk Storage System Forecast 2006-2010

데이터 차별화

❖ Hybrid DBMS를 통한 효율성 증대

일반적 MMDBMS의 적용(혼용 구조)



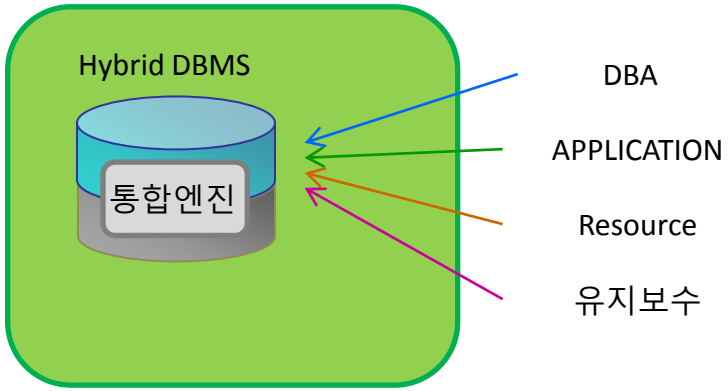
관리에서 운영, 구성적인 모든 비용과 노력들이 두 개의 DBMS를 전부 제어 해야만 함

관리적 비용 : DRDBMS + MMDBMS 비용 추가

운영적 비용 : DRDBMS + MMDBMS 비용 추가

구성적 비용 : DRDBMS + MMDBMS 비용 추가

Hybrid DBMS의 적용



메모리와 디스크 저장장치를 하나의 엔진에서 완벽하게 융합되어 제공함으로 효율성 극대화

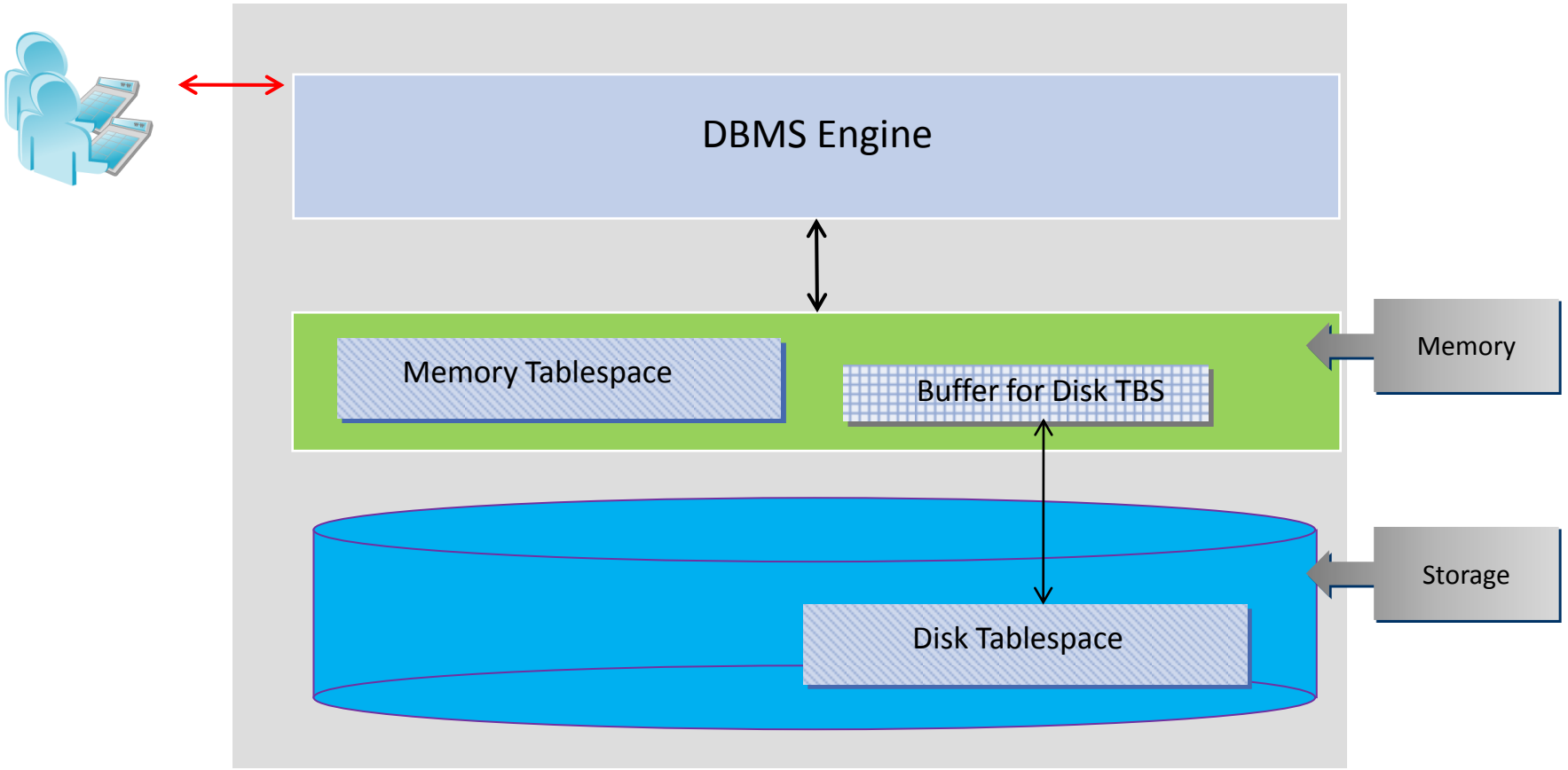
관리적 비용 : Hybrid DBMS만을 관리 [비용 ½ 절감]

운영적 비용 : Hybrid DBMS만을 관리 [비용 ½ 절감]

구성적 비용 : Hybrid DBMS만을 관리 [비용 ½ 절감]

Hybrid DBMS

❖ Hybrid DBMS 테이블스페이스 구조



ALTIBASE

1.2 관계형 데이터 모델

관계형 데이터 모델

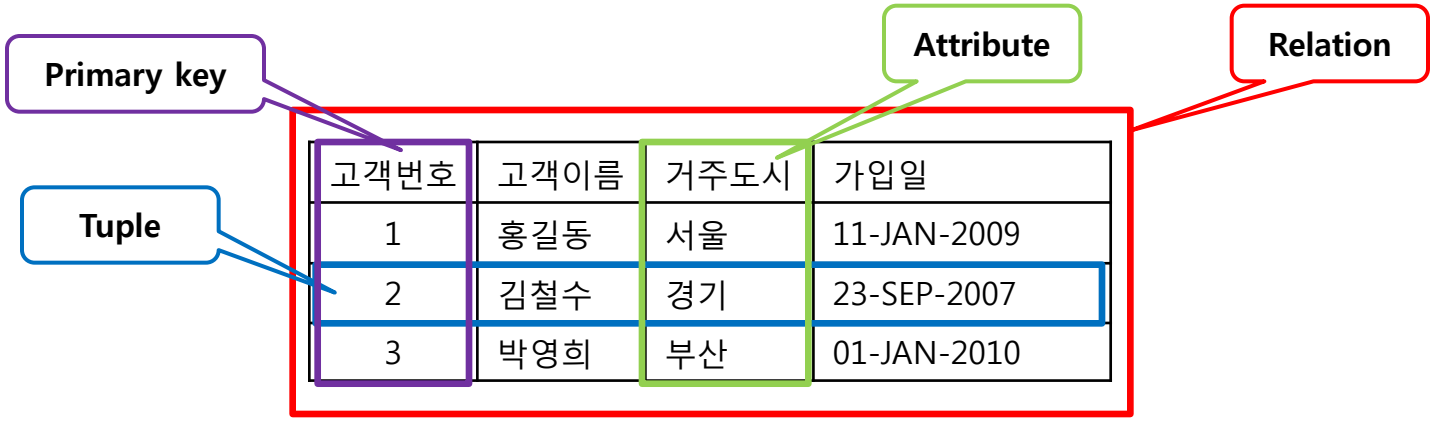
❖ 관계형 데이터베이스

➢ 관계형 데이터 모델

- DB 데이터를 이차원 테이블에 저장하고 테이블들간 관계를 정의하는 구조
- 데이터의 독립성, 일관성, 무결성을 보장
- 기본키와 이를 참조하는 외래키로 데이터 간의 관계를 표현
- ER(Entity Relationship)모델로 표현
- SQL(Structured Query Language)을 이용하여 DBMS와 통신

관계형 데이터 모델

- 관계형 데이터 모델에서 사용되는 객체



용어	개념	특징
Relation	테이블	각 열(Column)은 유일한 값을 가지며 행의 순서 무의미
Attribute	열(column)	속성의 이름은 모두 달라야 함 속성들간의 순서는 중요하지 않음
Tuple	행(row)	연관된 속성의 모임 파일 구조의 레코드와 같은 의미
Primary Key	유일한 식별자	테이블의 모든 행들을 구별하기 위해 사용
Domain	속성 값들의 집합	여러 개의 속성에서 공유 속성의 이름과는 달라도 무관

관계형 데이터 모델

SQL(Structured Query Language)

- 데이터베이스에 저장되는 데이터를 조작하고, 관리하며, 검색하기 위한 언어
- ;(세미콜론)으로 종료, 대소문자를 구분하지 않음

SQL 종류

- 데이터 정의어(Data Definition Language)
  - 메타 정보가 변경되는 SQL문, 객체를 생성,변경, 삭제하는 SQL문
  - CREATE, ALTER, DROP, TRUNCATE, RENAME, GRANT, REVOKE
- 데이터 조작어(Data Manipulation Language)
  - 데이터를 질의하거나 데이터를 변경하는 SQL문
  - AUTOCOMMIT이 OFF인 경우 명시적인 COMMIT, ROLLBACK이 필요
  - INSERT, DELETE, UPDATE, SELECT, MOVE, ENQUEUE, DEQUEUE, LOCK
- 데이터 제어어(Data Control Language)
  - 시스템, 세션 제어문
    - ALTER SYSTEM, ALTER SESSION
  - 트랜잭션 제어문
    - COMMIT, ROLLBACK, SAVEPOINT, SET TRANSACTION

ALTIBASE

1.3 ISQL

iSQL

iSQL

- ALTIBASE에 접속하여 질의 수행 및 결과를 조회할 수 있는 유틸리티
- DBA권한으로 ALTIBASE 구동 및 종료, 백업 및 복구 등 수행 가능
- $ALTIBASE_HOME/bin 에 위치

```
Shell::/home/alti2> isql –u sys –p manager –port 20300 –nls_use MS949 –s 127.0.0.1
----------------------------------------------------------------
   ALTIBASE Client Query utility.
   Release Version 6.1.1.0.10
   Copyright 2000, ALTIBASE Corporation or its subsidiaries.
   All Rights Reserved.
----------------------------------------------------------------
ISQL_CONNECTION = TCP, SERVER = 127.0.0.1, PORT_NO = 20300
iSQL>
```

iSQL

is 스크립트

- $ALTIBASE_HOME/bin 에 위치
- iSQL의 입력 시 옵션들을 생략하고 접속할 수 있도록 제공하는 스크립트

```
Shell::/home/alti2> cat $ALTIBASE_HOME/bin/is
#!/bin/sh

trap "" TSTP
${ALTIBASE_HOME}/bin/isql -s 127.0.0.1 -u sys -p manager $*
```

❖ iSQL 실행 시 입력 옵션

입력 옵션	설명
-s	ALTIBASE 서버가 위치한 IP를 지정
-u	ALTIBASE DB 사용자 명을 지정
-p	DB 사용자의 패스워드를 지정
-port	ALTIBASE Listen Port번호를 지정
-nls_use	ALTIBASE DB 생성 시 입력한 문자셋을 지정
-o	iSQL에서 실행한 결과를 저장할 파일명을 지정
-f	iSQL에서 수행할 질의 및 명령을 저장한 입력 파일명을 지정
-h	입력 옵션에 대한 도움말을 출력

❖ iSQL 실행 후 입력 옵션 (1)

옵션	설명
desc	테이블 구성 정보를 확인
@	지정된 파일명을 실행
!	OS 명령을 수행하고자 할 경우
h	수행된 질의의 목록을 확인
/	직전에 수행한 질의를 재 수행
ed	직전에 수행한 질의를 편집하고자 할 경우
autocommit	현재 세션의 autocommit 모드를 변경할 경우 (on/off)
spool [fileName]	spool 명령에 입력된 파일에 현재 수행 결과를 기록
show all	iSQL의 현재 설정 상태 및 사용자 명을 출력
show user	현재 iSQL로 서버에 접속중인 사용자 명을 출력

❖ iSQL 실행 후 입력 옵션 (2)

옵션	설명
colsize	칼럼 사이즈의 길이를 지정
linesize	하나의 레코드의 출력라인의 길이를 지정
pagesize	지정된 개수만큼 레코드 출력 후 칼럼 타이틀을 출력
heading	출력 결과에서 칼럼 타이틀을 보이거나 감추도록 설정
timing	실행한 질의의 수행 시간을 1/100 초 단위로 출력
vertical	칼럼들을 세로로 출력 한 라인에 (칼럼 명 : Value) 형태로 결과를 출력
foreignkeys	desc 명령으로 테이블 조회 시 참조키 정보를 출력 (on/off)

iSQL 사용 예 : 테이블 목록의 조회

```
iSQL> SELECT * FROM TAB;
USER_NAME        TABLE_NAME                  TYPE
-------------------------------------------------------------
SYSTEM_          STO_COLUMNS_                SYSTEM TABLE
SYSTEM_          STO_DATUMS_                 SYSTEM TABLE
SYSTEM_          STO_ELLIPSOIDS_             SYSTEM TABLE
SYSTEM_          STO_GEOCCS_                 SYSTEM TABLE
SYSTEM_          STO_GEOGCS_                 SYSTEM TABLE
….
….
iSQL>
```

❖ iSQL 사용 예 : 테이블 구성의 조회

```
iSQL> DESC t1;
[ TABLESPACE : SYS_TBS_MEM_DATA ]
[ ATTRIBUTE ]
------------------------------------------------------------------
NAME                    TYPE                  IS NULL
------------------------------------------------------------------
A                     INTEGER    FIXED        NOT NULL
[ INDEX ]
------------------------------------------------------------------
NAME                    TYPE      IS UNIQUE     COLUMN
------------------------------------------------------------------
__SYS_IDX_ID_102        BTREE     UNIQUE        A ASC
[ PRIMARY KEY ]
------------------------------------------------------------------
A
iSQL>
```

2. SELECT

1. SELECT 기초
2. 단일행 함수
3. 그룹함수 & 윈도우 함수
4. JOIN
5. 서브쿼리
6. SET 연산자
7. 계층 질의

ALTIBASE

2.1 SELECT 기초

데이터 검색

- **SELECT**

  데이터베이스에서 데이터를 검색하는 문장

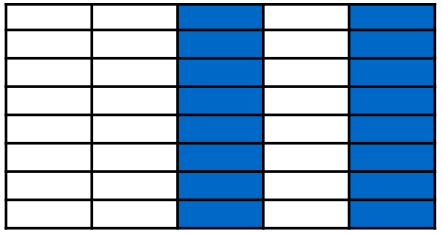
- **SELECT 문의 기능**

  - SELECTION

    테이블에서 일부의 행만 검색

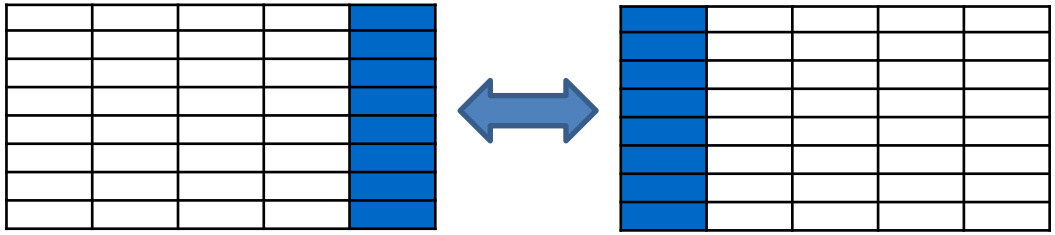
  - PROJECTION

    테이블에서 일부의 열만 검색



  - JOIN

    서로 다른 테이블을 연결하여 데이터를 함께 검색



데이터 검색

❖ 기본 SELECT 구문

```
SELECT * | { [DISTINCT | ALL] column | expression [alias], ... }
FROM table_name;
```

❖ 예제

- Employee 테이블의 모든 사원 정보를 검색

```
iSQL> SELECT *
    2  FROM  employee;
```

- Employee 테이블 중 사원이름과 급여를 검색

```
iSQL> SELECT ename, salary
    2  FROM  employee;
```

- 중복된 데이터 제거

```
iSQL> SELECT dno
    2  FROM  employee;

iSQL> SELECT DISTINCT dno
    2  FROM employee;
```

NULL

NULL

- 알 수 없는 값
- 0 또는 공백과 다름
- 테이블 생성시 NOT NULL 또는 PRIMARY KEY로 정의되지 않은 모든 타입의 칼럼은 NULL을 포함할 수 있음

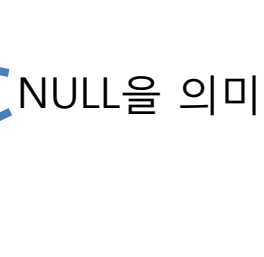
예제

- Employee 테이블의 급여 정보를 검색

```
iSQL> SELECT salary
    2  FROM  employee;
SALARY
--------------

1500000
2000000
1700000
500000

1200000
…
```



Alias

Alias

- 컬럼과 테이블의 별명을 지정
- 칼럼의 별명은 display 시 칼럼의 이름을 바꿔주고, 테이블의 별명은 join/subquery 등에서 사용

구문

```
SELECT column_name [AS] alias, ...
FROM table_name [AS] alias;
```

- AS는 생략 가능
- 공백, 특수문자를 포함하거나 대소문자를 구별할 때는 " ", ' ' 를 사용

예제

- ename 대신 Employee Name으로 검색

```
iSQL> SELECT ename AS "Employee name"
      2 FROM employee;
Employee name
-------------------------
EJJUNG
HJNO
-
```

Employee name으로 display

WHERE

SELECTION

- 테이블에서 필요한 행만 검색
- WHERE 절 사용

구문

```
SELECT column_name, ...
FROM table_name
WHERE conditions;
```

- conditions에는 칼럼이름, 표현식, 상수, 비교 연산자 등으로 구성

예제

- KSKIM 사원의 급여를 검색

```
iSQL> SELECT ename, salary
      2 FROM employee
      3 WHERE ename='KSKIM';
ENAME                 SALARY
---------------------------------------
KSKIM                 1800000
1 row selected.
```

WHERE

❖ 조건절 구성 항목

- 칼럼, 문자나 숫자 상수
- 연산자(+, -, *, /, =, >, < 등)
- LIKE, IN, BETWEEN, EXISTS, NOT
- IS NULL, IS NOT NULL
- 함수
- AND, OR, NOT
- ANY, ALL

❖ 상수의 표현

- 문자열

  대소문자를 구별하며, '(작은 따옴표)로 묶는다. ex) 'KSKIM'
- 날짜

  날짜 형식을 구별하며, '(작은 따옴표)로 묶는다.
  기본 형식은 'DD-MON-YYYY' ex) '05-MAY-2011'
- 숫자

  '(작은 따옴표)로 묶지 않고 숫자만 작성한다. ex) 1234

Operator

❖ 산술연산자

NUMBER, DATE 데이터에 대해 사용

산술 연산자	설명	사용 가능한 데이터 타입
+	더하기	NUMBER, DATE
-	빼기	NUMBER, DATE
*	곱하기	NUMBER
/	나누기	NUMBER

❖ 산술연산자 우선순위

- 곱셈과 나눗셈은 덧셈과 뺄셈 보다 우선순위가 높음
- 우선순위가 동일한 연산자는 왼쪽에서 오른쪽으로 계산
- 괄호()를 이용하여 우선순위를 조정

❖ 예제

- 사원이름, 급여 정보와 연봉을 함께 검색

```
iSQL> SELECT ename, salary, salary * 12
      2 FROM employee;
```

Operator

산술연산자와 NULL

NULL에 연산을 수행하면 그 결과 값도 NULL이 리턴

예제

- 사원이름, 급여 정보와 연봉을 함께 검색

```
iSQL> SELECT ename, salary, salary * 12
    2 FROM employee;
ENAME       SALARY       SALARY*12
-------------------------------------
EJJUNG
HJNO        1500000      18000000
HSCHOI      2000000      24000000
KSKIM       1800000      21600000
SJKIM       2500000      30000000
HYCHOI      1700000      20400000
HJMIN        500000       6000000
…
```

salary가 NULL일 경우
salary*12도 NULL이 리턴

Operator

연결 연산자

- 문자열을 연결할 때는 || 기호를 사용
- 칼럼과 칼럼을 연결하거나 문자열과 칼럼들을 연결

예제

- 사원이름과 급여를 함께 검색

```
iSQL> SELECT ename||salary
      2 FROM employee;
ENAME||SALARY
------------------------------------------------------------------------
EJJUNG
HJNO            1500000
HSCHOI          2000000
…
```

연결된 1개의 칼럼

- 사원이름과 급여를 함께 검색(문자열 포함)

```
iSQL> SELECT ename||'의 급여는 '||salary
      2 FROM employee;
ENAME||' 의 급여는 '||SALARY
------------------------------------------------
EJJUNG      의 급여는
HJNO        의 급여는  1500000
…
```

문자열을 함께 연결

Operator

❖ 비교 연산자 (1)

비교 연산자	설명
=	같음
>	~보다 큼
>=	~보다 크거나 같음
<	~보다 작음
<=	~보다 작거나 같음
<>, !=	같지 않음

❖ 예제

➢ 급여가 1000000 이상인 사원을 검색

```
iSQL> SELECT ename, salary
      2 FROM employee
      3 WHERE salary >= 1000000;
```

➢ 4002 부서에서 일하지 않는 사원을 검색

```
iSQL> SELECT ename, salary
      2 FROM employee
      3 WHERE dno <> 4002;
```

Operator

❖ 비교 연산자 (2)

비교 연산자	설명
BETWEEN ~ AND ~	지정한 값을 포함하여 두 값 사이인가?
IN(*list*)	list에 같은 값이 있는가?
IS NULL	NULL 값인가?
LIKE	문자 패턴 비교

❖ BETWEEN AND

지정한 값 사이에 있는지를 비교. 지정한 값은 포함하여 판단

❖ 예제

➢ 급여가 1000000과 2000000 사이인 사원을 검색

```
iSQL> SELECT ename, salary
      2 FROM employee
      3 WHERE salary BETWEEN 1000000 AND 2000000;
```

Operator

IN

() 안에 지정된 목록의 값이 일치하는지를 비교

예제

- 4001 또는 4002 부서에서 일하는 사원을 검색

```
iSQL> SELECT ename, dno
      2 FROM employee
      3 WHERE dno IN (4001, 4002);
```

- KSKIM, YHBAE 와 CHLEE 사원을 검색

```
iSQL> SELECT ename
      2 FROM employee
      3 WHERE ename IN ('KSKIM', 'YHBAE', 'CHLEE');
```

Operator

IS NULL

NULL 값인지를 비교
= NULL로 비교하면 No rows selected.

예제

- 급여가 아직 확정되지 않은 사원을 검색

```
iSQL> SELECT ename, salary
      2 FROM employee
      3 WHERE salary = NULL;
ENAME                  SALARY
---------------------------------------
No rows selected.
```

```
iSQL> SELECT ename, salary
      2 FROM employee
      3 WHERE salary IS NULL;
ENAME                  SALARY
---------------------------------------
EJJUNG
JDLEE
DIKIM
3 rows selected.
```

Operator

LIKE

- 문자 패턴을 비교
- 문자 패턴 비교 시 사용 가능한 기호

기호	설명
_	하나의 문자를 대체
%	문자가 없거나 하나 이상의 문자를 대체

- ESCAPE
  %, _를 포함한 문자열을 비교 시 사용

예제

- 이름이 K로 시작하는 사원 정보를 검색

```
iSQL> SELECT ename
      2 FROM employee
      3 WHERE ename LIKE 'K%';
```

- 이름의 두 번째 문자가 A인 사원을 검색

```
iSQL> SELECT ename
      2 FROM employee
      3 WHERE ename LIKE '_A%';
```

Operator

- 이름에 _를 포함한 사원을 검색

```
iSQL> SELECT ename
      2 FROM employee
      3 WHERE ename LIKE '%\_%' ESCAPE '\';
```

ESCAPE과 함께 임의의 문자를 사용 가능

Operator

❖ 논리 연산자

두 개의 조건의 결과를 결합하여 하나의 조건으로 생성

논리 연산자	설명
AND	두 조건이 모두 TRUE이면 TRUE, 그렇지 않으면 FALSE.
OR	두 조건 중에 하나라도 TRUE이면 TRUE, 둘 다 FALSE이면 FALSE.
NOT	조건이 TRUE이면 FALSE, FALSE이면 TRUE

❖ 예제

➢ 급여가 1000000 이상이면서 4002 부서에서 일하는 사원을 검색

```
iSQL> SELECT ename, dno, salary
      2 FROM employee
      3 WHERE salary >= 1000000
      4 AND dno = 4002;
```

➢ 급여가 1000000 이상이거나 4002 부서에서 일하는 사원을 검색

```
iSQL> SELECT ename, dno, salary
      2 FROM employee
      3 WHERE salary >= 1000000
      4 OR dno = 4002;
```

Operator

- 급여가 1000000과 2000000 사이가 아닌 사원을 검색

```
iSQL> SELECT ename, salary
      2 FROM employee
      3 WHERE salary NOT BETWEEN 1000000 AND 2000000;
```

- 4001, 4002 부서에서 일하지 않는 사원을 검색

```
iSQL> SELECT ename, dno
      2 FROM employee
      3 WHERE dno NOT IN (4001, 4002);
```

- 이름에 A를 포함하지 않는 사원을 검색

```
iSQL> SELECT ename, dno
      2 FROM employee
      3 WHERE ename NOT LIKE '%A%';
```

ORDER BY

결과를 정렬

- 검색 결과를 정렬
- 내림차순, 오름차순으로 정렬이 가능

구문

```
SELECT column_name, …
FROM table_name
[WHERE conditions]
ORDER BY {column_name | alias | column_index} [ASC | DESC];
```

- ASC(기본값)는 오름차순, DESC는 내림차순으로 정렬

예제

- 사원정보를 사원이름 순으로 내림차순으로 정렬하여 검색

```
iSQL> SELECT ename
      2 FROM employee
      3 ORDER BY ename DESC;
ENAME
------------------------
YHBAE
…
DIKIM
CHLEE
```

ORDER BY

- 사원정보를 사원이름 순으로 내림차순으로 정렬하여 검색(alias 사용)

```
iSQL> SELECT ename AS employee_name
      2 FROM employee
      3 ORDER BY employee_name DESC;
ENAME
------------------------
YHBAE
…
DIKIM
CHLEE
```

- 사원정보를 사원이름 순으로 내림차순으로 정렬하여 검색 (column_index 사용)

```
iSQL> SELECT ename
      2 FROM employee
      3 ORDER BY 1 DESC;
ENAME
------------------------
YHBAE
…
DIKIM
CHLEE
```

ORDER BY

- 사원정보를 부서번호의 오름차순 순으로 정렬하고 부서번호가 같다면 사원 이름의 내림차순 순으로 정렬

```
iSQL> SELECT dno, ename, salary
      2 FROM employee
      3 ORDER BY dno, ename DESC;

DNO          ENAME                     SALARY
-------------------------------------------------
1001         JHCHOI                   2300000
1001         HSCHOI                   2000000
1002         KWKIM                     980000
1002         HYCHOI                   1700000
1003         YHBAE                    4000000
1003         MSKIM                    2750000
…
4002         MYLEE                    1890000
4002         KMKIM                    1800000
4002         HJMIN                     500000
4002         DIKIM
             HJNO                     1500000
20 rows selected.
```

LIMIT

결과 집합을 제한

- 검색 결과의 일부분만 반환

구문

```
SELECT column_name, …
FROM table_name
[WHERE conditions]
[ORDER BY column_name,..]
LIMIT [start_index ,] row_count;
```

- start index 생략 시 첫 번째 행부터 시작

예제

- 사원 중 다섯 번째부터 3명의 사원만 검색

```
iSQL> SELECT *
      2 FROM employee
      3 LIMIT 5, 3;
```

- 급여를 내림차순으로 정렬한 후 상위 5명만 검색

```
iSQL> SELECT ename
      2 FROM employee
      3 ORDER BY salary DESC
      4 LIMIT 5;
```

PIVOT

PIVOT

- PIVOT이란 ROW 형태로 주어진 DATA를 Column 형태로 보여주는 쿼리

구문

```
SELECT column_name, …
FROM table_name
PIVOT (pivot_clause
        pivot_for_clause
        pivot_in_clause)
[WHERE conditions]
```

- Pivot_clause

  Aggregate 되는 컬럼을 정의하는 부분으로 7가지(COUNT, MIN, MAX, SUM, AVG, STDDEV, VARIANCE) Aggregate Function 만 올수 있다.

- Pivot_for_clause

  Pivot 되는 컬럼을 정의하며 컬럼 이름만 올수 있다.

- Pivot_in_clause

  Pivot For 구문에 사용된 컬럼의 Filter 를 정의하며 Alias 를 지정할수 있다.

PIVOT

❖ 예제

➢ 부서별로 PM, PL, MANAGER가 몇 명인지를 확인하는 내용을 PIVOT 구문을 사용하여 검색

```
iSQL> SELECT *  FROM
    2             (SELECT e.emp_job, d.dname, e.eno
    3              FROM employee e, department d
    4              WHERE e.dno = d.dno)
    5  PIVOT (COUNT(eno)
    6         FOR emp_job
    7         IN ('PM', 'PL', 'MANAGER') );

DNAME                                'PM'          'PL'          'MANAGER'
-------------------------------------------------------------------------
RESEARCH DEVELOPMENT DEPT 1            0             0                1
RESEARCH DEVELOPMENT DEPT 2            1             0                0
SOLUTION DEVELOPMENT DEPT              1             0                0
QUALITY ASSURANCE DEPT                 1             0                0
CUSTOMER SUPPORT DEPT                  0             1                0
PRESALES DEPT                          0             1                0
MARKETING DEPT                         0             0                1
BUSINESS DEPT                          0             0                1
8 rows selected.
```

PIVOT

❖ 유의사항

- 중복 Alias 를 허용하지 않는다.
- Pivot_clause에 Aggregate Function 만 올수 있다.
- Pivot_for 에는 순수 column 만 올수 있다.
- Pivot_for 의 항목 개수와 Pivot_in의 sublist 의 개수가 같아야 한다.

ALTIBASE

2.2 단일 행 함수

함수

❖ 함수

- 입력 값에 대해 작업을 수행하여 결과 값을 리턴
- SELECT, WHERE, ORDER BY, START WITH 절에서 사용 가능
- 중첩해서 사용 가능

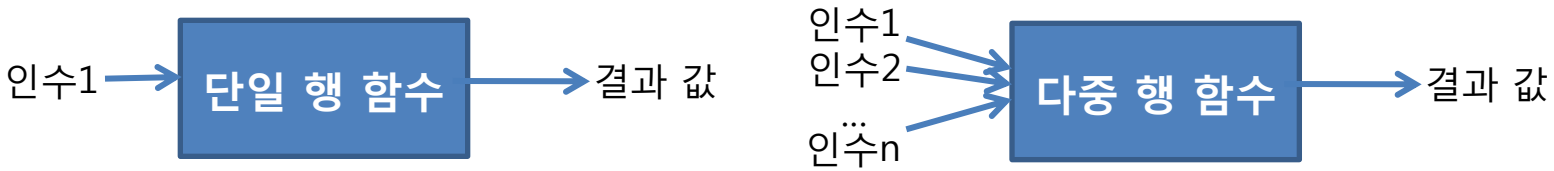
❖ 함수의 기능

- 데이터의 계산 수행
- 출력 결과를 조작
- 날짜 및 숫자 형식을 지정
- 칼럼의 데이터 유형 변경

함수

❖ 함수의 유형

- 단일 행 함수 : 하나의 입력 값에 하나의 결과 값을 반환
- 다중 행 함수 : 여러 개의 입력 값에 하나의 결과 값을 반환(그룹함수)



❖ 단일 행 함수의 유형

- 문자 : 문자를 입력하면 문자 또는 숫자를 반환
- 숫자 : 숫자를 입력하면 숫자를 반환
- 날짜 : Date 데이터의 값을 연산 , Date 값 혹은 숫자를 반환
- 변환 : 값의 데이터 타입을 변환
- 기타

함수

❖ 단일 행 함수 구문

function_name (column | expression [arg1, arg2,...])

❖ 예제

➢ 사원이름을 소문자로 변경

```
iSQL> SELECT ename, LOWER(ename) FROM employee;
ENAME           LOWER(ENAME)
-----------------------------------
EJJUNG          ejjung
HJNO            hjno
HSCHOI          hschoi
…
```

➢ 10000을 19로 나누었을 때 나머지를 검색

```
iSQL> SELECT MOD(10000,19) FROM dual;
MOD(10000,19)
----------------
6
1 row selected.
```

dual은 하나의 칼럼만 가지고 있는 dummy 테이블. 주로 일시적인 연산에 사용

문자함수

❖ 문자함수(1)

함수	결과	설명
ASCII('A')	65	ASCII 코드 값을 반환
CHAR_LENGTH ('ALTIBASE V5') CHARACTER_LENGTH('ALTIBASE V5') LENGTH('ALTIBASE V5')	11	문자열의 길이를 반환
CHR(65)	A	ASCII 코드 값을 받아서 해당하는 문자로 반환
CONCAT('ALTIBASE', 'V5')	ALTIBASEV5	첫 번째 문자와 두 번째 문자를 연결
INITCAP('altibase V5')	Altibase V5	단어의 첫 문자를 대문자, 나머지 문자를 소문자로 반환
INSTR('ALTIBASE V5', 'V5') POSITION('ALTIBASE V5', 'V5')	10	문자열을 찾아 그 위치를 반환
LOWER('ALTIBASE V5')	altibase v5	소문자로 변환
LPAD(12000, 8, '*')	***12000	첫 번째 문자가 두 번째 지정한 숫자길이가 되도록 왼쪽에서부터 세 번째 문자를 삽입하여 반환
LTRIM('ALTIBASE V5','ALTIBASE ')	V5	첫 번째 문자열이 두 번째 문자열로 시작할 경우 두 번째 문자열을 삭제한 후 반환
OCTET_LENGTH('ALTIBASE HDB') LENGTHB('ALTIBASE HDB')	12	입력한 문자열의 바이트 크기를 반환

문자함수

❖ 문자함수(2)

함수	결과	설명
REPLACE2('ALTIBASE V5', 'ALTIBASE‘ ,'A.')	A.V5	첫 번째 문자열에 포함된 두 번째 문자열을 세 번째 문자열로 변환
REPLICATE('Alti', 3)	AltiAltiAlti	문자열을 지정한 숫자만큼 반복하여 반환
RPAD(12000, 8, '*')	12000***	첫 번째 문자가 두 번째 지정한 숫자길이가 되도록 오른쪽에서부터 세 번째 문자를 삽입하여 반환
RTRIM('ALTIBASE V5', 'V5')	ALTIBASE	첫 번째 문자열이 두 번째 문자열로 끝날 경우 두 번째 문자열을 삭제한 후 반환
SIZEOF('ALTIBASE V5')	11	문자열 혹은 열에 할당된 크기를 반환 테이블 생성 시 정의된 칼럼 크기를 반환
SUBSTR('ALTIBASE V5', 1, 4)	ALTI	문자열의 일부를 반환. 첫 번째 문자부터 4개의 문자열을 반환
TRANSLATE('ALTIBASE V4', '4', '5')	ALTIBASE V5	첫 번째 문자열에 포함되어 있는 두 번째 문자를 모두 세 번째 문자로 변환
TRIM('ALTIBASE V5', 'A')	LTIBASE V5	첫 번째 문자열이 두 번째 문자열로 시작하거나 끝날 경우 두 번째 문자열을 삭제한 후 반환
UPPER('ALTIBASE V5')	ALTIBASE V5	대문자로 변환
REVERSE_STR('ALTIBASE')	ESABITLA	문자열을 거꾸로 반환

문자함수

❖ 예제

➢ 이름이 kskim인 사람의 정보를 검색

```
iSQL> SELECT ename, INITCAP(ename), SUBSTR(ename,2,3), REVERSE_STR(ename), INSTR(ename,'K')
      2 FROM employee
      3 WHERE LOWER(ename) = 'kskim';
ENAME   INITCAP(ENAME)   SUBSTR(ENAME,2,3) REVERSE_STR(ENAME) INSTR(ENAME,'K')
------------------------------------------------------------------------------
KSKIM       Kskim           SKI              MIKSK                 1
```

➢ emp_job에 여러 가지 문자함수를 사용

```
iSQL> SELECT emp_job, SIZEOF(emp_job), LENGTH(emp_job), RPAD(emp_job,8,'*'),
      2 TRANSLATE(emp_job,'E','Z')
      3 FROM employee;
EMP_JOB   SIZEOF(EMP_JOB) LENGTH(EMP_JOB) RPAD(EMP_JOB,8,'*') TRANSLATE(EMP_JOB,'E','Z')
-------------------------------------------------------------------------------------
CEO           15              3               CEO*****              CZO
DESIGNER      15              8               DESIGNER              DZSIGNZR
ENGINEER      15              8               ENGINEER              ZNGINZZR
...
```

숫자함수

❖ 숫자함수(1)

함수	결과	설명
ABS(-1234.56)	1234.56	절대값
ACOS(0.3)	1.266104	ACOS
ASIN(0.3)	0.304693	ASIN
ATAN(0.3)	0.291457	ATAN
ATAN2(0.3, 0.2)	0.982794	ATAN2
CEIL(99.9)	100	올림. 소수점을 기준
COS(180 * 3.14159265359/180)	-1	COS
COSH(0)	1	COSH
EXP(2.4)	11.023176	e^n (e는 2.71828183...)
FLOOR(-99.9)	-100	내림. 소수점을 기준
LN(2.4)	0.875469	자연로그

숫자함수

❖ 숫자함수(2)

함수	결과	설명
LOG(10, 100)	2	log 함수
MOD(10, 3)	1	첫 번째 인자값을 두 번째 인자값으로 나눈 나머지
POWER(3, 2)	9	첫 번째 인자값을 두 번째 인자값만큼 반복해서 곱한 값. 지수
RANDOM(2)		random 값을 반환. 입력한 숫자가 같으면 같은 값을 반환
ROUND(123.9994, 3)	123.999	반올림. 두 번째 숫자가 반올림할 소수점 자릿수를 의미
SIGN(15)	1	부호를 리턴. 0이면 0, 양수면 1, 음수면 -1을 리턴
SIN(30 * 3.14159265359/180)	0.5	SIN
SINH(1)	1.175201	SINH
SQRT(10)	3.162278	제곱근
TAN(135 * 3.14159265359/180)	-1	TAN
TANH(0.5)	0.462117	TANH
TRUNC(15.79, 1)	15.7	버림. 두 번째 숫자가 버림 할 소수점 자릿수를 의미

숫자함수

❖ 예제

- salary를 12로 나누었을 때 나머지를 질의

```
iSQL> SELECT ename, salary, MOD(salary,12)
      2 FROM employee;
ENAME                 SALARY      MOD(SALARY,12)
------------------------------------------------
EJJUNG
HJNO                  1500000                  0
HSCHOI                2000000                  8
KSKIM                 1800000                  0
SJKIM                 2500000                  4
…
```

- 123.673 숫자에 대해 여러 가지 숫자함수를 사용

```
iSQL> SELECT ROUND(123.673,2), TRUNC(123.673,-1), CEIL(123.673), FLOOR(123.673), SQRT(123.673)
      2 FROM dual;
ROUND(123.673,2) TRUNC(123.673,-1) CEIL(123.673) FLOOR(123.673) SQRT(123.673)
-------------------------------------------------------------------------------
123.67           120               124           123             11.1208362994876
```

숫자함수

- random 함수 사용

```
iSQL> SELECT RANDOM(100)
      2 FROM dual;
RANDOM(100)
--------------
677741240
iSQL> SELECT RANDOM(200)
      2 FROM dual;
RANDOM(200)
--------------
331330603
iSQL> SELECT RANDOM(100)
      2 FROM dual;
RANDOM(100)
--------------
677741240
```

날짜함수

❖ 날짜함수

함수	결과	설명
SYSDATE	28-APR-2011	시스템의 날짜를 반환
ADD_MONTHS(SYSDATE,5)	28-SEP-2011	날짜에 월을 더함
DATEADD(SYSDATE, -30, 'DAY')	29-MAR-2011	날짜에 세 번째 포맷을 두 번째 숫자만큼 더함. -30일을 더함
DATEDIFF(SYSDATE, '01-JAN-11', 'DAY')	-117	두 번째 날짜에서 첫 번째 날짜를 포맷만큼 뺀 수를 반환 01-JAN-11에서 SYSDATE만큼 며칠이 지났는지를 뺌
DATENAME(SYSDATE,'DAY')	THURSDAY	날짜에서 월/요일 이름을 추출. MONTH는 월을, DAY는 요일을 나타냄
EXTRACT(SYSDATE,'MONTH') DATEPART(SYSDATE,'MONTH')	4	날짜에서 주어진 포맷을 추출
LAST_DAY(SYSDATE)	30-APR-2011	날짜에서 해당 월의 마지막 날짜를 반환
MONTHS_BETWEEN(SYSDATE, '01-JAN-11')	3.8976769..	첫 번째 날짜에서 두 번째 날짜를 뺀 값을 개월 수로 반환
NEXT_DAY(SYSDATE,'SUNDAY')	01-MAY-2011	날짜 이후에 돌아오는 요일의 날짜를 반환
ROUND(SYSDATE,'MONTH')	01-MAY-2011	날짜에서 두 번째 포맷에 맞춰 반올림
TRUNC(SYSDATE,'MONTH')	01-APR-2011	날짜에서 두 번째 포맷에 맞춰 버림

날짜함수

❖ 예제

➢ 현재 시스템의 날짜(SYSDATE)와 SYSDATE에 대한 여러 가지 날짜함수를 사용

```
iSQL> SELECT SYSDATE, DATEADD(SYSDATE, 2, 'WEEK'), DATEDIFF(SYSDATE, '01-APR-11', 'DAY')
      2 FROM dual;
SYSDATE      DATEADD(SYSDATE, 2, 'WEEK') DATEDIFF(SYSDATE, '01-APR-11', 'DAY')
------------------------------------------------------------------------------
29-APR-2011                  13-MAY-2011                  -28
```

```
iSQL> SELECT EXTRACT(SYSDATE,'QUARTER'), DATENAME(SYSDATE,'DAY'),
      2 DATENAME(SYSDATE, 'DY')
      3 FROM dual;
EXTRACT(SYSDATE,'QUARTER') DATENAME(SYSDATE,'DAY')  DATENAME(SYSDATE, 'DY')
-------------------------------------------------------------------------
2                          FRIDAY                   FRI
```

```
iSQL> SELECT ROUND(SYSDATE,'YEAR'), ROUND(SYSDATE), TRUNC(SYSDATE, 'DAY'),
      2 TRUNC(SYSDATE)
      3 FROM dual;
ROUND(SYSDATE,'YEAR') ROUND(SYSDATE) TRUNC(SYSDATE, 'DAY') TRUNC(SYSDATE)
-------------------------------------------------------------------------
01-JAN-2011           30-APR-2011    29-APR-2011           29-APR-2011
```

날짜함수

❖ 예제

➢ 입사일(join_date), 입사일 6개월 이후의 날짜, 입사한지 몇 개월이 지났는지 검색

```
iSQL> SELECT ename, join_date, ADD_MONTHS(join_date, 6),
      2 MONTHS_BETWEEN(SYSDATE, join_date)
      3 FROM employee;
ENAME    JOIN_DATE     ADD_MONTHS(JOIN_DATE,6)   MONTHS_BETWEEN(SYSDATE, JOIN_DATE)
--------------------------------------------------------------------------------
HJNO     18-NOV-1999   18-MAY-2000               137.349825288486
HSCHOI   11-JAN-2000   11-JUL-2000               135.575631740099
…
```

➢ 입사일 이후 돌아오는 월요일, 입사일 해당 월의 마지막 날짜를 검색

```
iSQL> SELECT ename, join_date, NEXT_DAY(join_date, 'MONDAY'), LAST_DAY(join_date)
      2 FROM employee;
ENAME     JOIN_DATE     NEXT_DAY(JOIN_DATE, 'MONDAY') LAST_DAY(JOIN_DATE)
------------------------------------------------------------------------
HJNO            18-NOV-1999   22-NOV-1999               30-NOV-1999
HSCHOI          11-JAN-2000   17-JAN-2000               31-JAN-2000
```

변환함수

❖ 변환함수

함수	결과	설명
TO_BIN(10)	1010	숫자를 2진수 형태로 변환
TO_CHAR(SYSDATE,'HH:MI:SS')	09:41:01	날짜나 숫자를 지정한 포맷을 적용하여 문자로 변환
TO_DATE('110430','YYMMDD')	30-APR-2011	문자를 날짜로 변환
TO_HEX(100)	64	숫자를 16진수 형태로 변환
TO_NUMBER('$1000','$9999')	1000	문자를 지정한 포맷을 적용하여 숫자로 변환
TO_OCT(100)	144	숫자를 8진수 형태로 변환
ASCIISTR('알티')	₩C54C₩D2F0	문자열을 ASCII 문자열로 반환. ASCII가 아닌 문자는 '₩xxxx'와 같이 UTF-16 코드로 표현
CONVERT('ALTI','MS949','UTF8')	ALTI	두 번째 캐릭터 셋의 문자를 세 번째 캐릭터 셋으로 변환
UNISTR('\C54C\D2F0')	알티	문자를 지정한 캐릭터 셋으로 변환

변환함수

❖ 예제

➢ '2011-05-01'은 입사일 후 며칠 지났는지 검색

```
iSQL> SELECT ename, join_date, TO_DATE('2011-05-01','YYYY-MM-DD') - join_date
      2 FROM employee;
ENAME      JOIN_DATE    TO_DATE('2011-05-01','YYYY-MM-DD') - JOIN_DATE
------------------------------------------------------------------------------
HJNO         18-NOV-1999                          4182
HSCHOI       11-JAN-2000                          4128
…
```

(날짜 - 날짜) 연산은 첫 번째 날짜에서 두 번째 날짜를 뺀 day 값을 반환

➢ 숫자 100 에 대해 여러 가지 진법 값을 반환

```
iSQL> SELECT 100, TO_BIN(100), TO_OCT(100), TO_HEX(100)
      2 FROM dual;
100          TO_BIN(100)      TO_OCT(100)    TO_HEX(100)
-------------------------------------------------------
100          1100100               144                64
```

기타함수

❖ 기타함수

함수	결과	설명
CASE2(a, b, c)		a값이 참이면 b를 반환. 거짓이면 c를 반환
DECODE(a, b, c, d)		a값이 b이면 c를 반환. b가 아니면 d를 반환.b를 생략하면 NULL을 반환
DUMP('A')	Type=CHAR(KSC5601) Length=3: 1,0,65	자료형, 길이, 메모리 내용 출력
GREATEST(10,20,30)	30	인자값 중 가장 큰 값을 출력
LEAST(10,20,30)	10	인자값 중 가장 작은 값을 출력
NVL(salary,0)		salary의 값이 NULL이면 0로 대체
NVL2(salary,salary,0)		salary의 값이 NULL이 아니면 salary를 출력. NULL이면 0를 출력
DIGEST('alti', 'SHA-1')	A6769CDF8D5EE5D33937A39F1...	SHA-1 알고리즘으로 변환한 문자 출력
ROWNUM		테이블이나 조인된 뷰의 레코드 번호를 정수 형태로 반환

기타함수

❖ 예제

➢ 급여가 2000000 보다 크면 1 그렇지 않으면 2를 반환

```
iSQL> SELECT ename, salary, CASE2(salary>2000000, 1, 2)
      2 FROM employee;
ENAME                 SALARY      CASE2(SALARY>2000000, 1, 2)
------------------------------------------------------------
KSKIM                 1800000           2
SJKIM                 2500000           1
HYCHOI                1700000           2
…
```

➢ dno가 4001이면 salary*1.1 , 4002이면 salary*1.2 그렇지 않으면 salary를 반환

```
iSQL> SELECT ename, dno, salary,
      2 DECODE(dno, 4001, salary*1.1, 4002, salary*1.2, salary) AS new_sal
      3 FROM employee;
ENAME                 DNO         SALARY      NEW_SAL
-----------------------------------------------------
DIKIM                 2001        1400000     1400000
CHLEE                 4001        1900000     2090000
KMKIM                 4002        1800000     2160000
…
```

기타함수

- 급여가 NULL일 경우 0 으로 반환

```
iSQL> SELECT ename, salary, NVL(salary, 0)
      2 FROM employee;
ENAME                  SALARY       NVL(SALARY,0)
-------------------------------------------------
EJJUNG                                         0
HJNO                    1500000        1500000
HSCHOI                  2000000        2000000
…
```

```
iSQL> SELECT ename, salary, NVL2(salary, salary, 0)
      2 FROM employee;
ENAME                  SALARY       NVL2(SALARY, SALARY, 0)
-------------------------------------------------------------
EJJUNG                                         0
HJNO                    1500000        1500000
HSCHOI                  2000000        2000000
…
```

ALTIBASE

2.3 그룹함수 & 윈도우 함수

그룹함수

❖ 그룹함수

➢ 여러 행에 대해 grouping 하여 결과를 계산

함수	설명
MIN	최소값
MAX	최대값
SUM	합계
AVG	평균
STDDEV	표준편차
VARIANCE	분산
COUNT	행의 수

- COUNT(*)를 제외한 모든 그룹함수는 NULL을 제외하고 계산

그룹함수

❖ 예제

- 사원들의 최소 급여, 최대 급여, 평균 급여, 급여의 합, 사원 수를 검색

```
iSQL> SELECT MIN(salary), MAX(salary), AVG(salary), SUM(salary), COUNT(*)
      2 FROM employee;
MIN(SALARY) MAX(SALARY) AVG(SALARY) SUM(SALARY) COUNT
------------------------------------------------------------------------
500000      4000000     1836647.06  31223000    20
```

- 평균 급여 (NULL을 포함하여 계산)

```
iSQL> SELECT AVG(NVL(salary,0))
      2 FROM employee;
AVG(NVL(SALARY,0))
--------------------
1561150
```

- 사원의 수

```
iSQL> SELECT COUNT(salary), COUNT(NVL(salary,0)), COUNT(*)
      2 FROM employee;
COUNT(SALARY)         COUNT(NVL(SALARY,0)) COUNT
------------------------------------------------------------------
17                    20                   20
```

GROUP BY

- **데이터를 grouping**
  - 데이터를 그룹으로 나눈 후 그룹 함수를 적용
- **구문**

```
SELECT column_name, group_function(), …
FROM table_name
[WHERE conditions]
[GROUP BY grouping_expression]
ORDER BY {column_name|alias|column_index} [ASC | DESC]
LIMIT [start_index ,] row_count;
```

- **예제**
  - 부서번호 별로 급여의 합, 급여의 평균을 검색

```
iSQL> SELECT dno, SUM(salary), AVG(salary)
    2 FROM employee
    3 GROUP BY dno;
DNO         SUM(SALARY)  AVG(SALARY)
---------------------------------------
1001        4300000      2150000
1002        2680000      1340000
1003        9753000      2438250
…
```

GROUP BY

여러 칼럼을 GROUP BY

- 그룹 내 그룹
- 첫 번째 칼럼을 먼저 grouping하고, 그 그룹을 두 번째 칼럼을 기준으로 또 grouping을 수행

예제

- 부서번호, 직무 별로 급여의 합, 급여의 평균을 검색

```
iSQL> SELECT dno, emp_job, SUM(salary), AVG(salary)
    2 FROM employee
    3 GROUP BY dno, emp_job
    4 ORDER BY 1, 2;
DNO         EMP_JOB          SUM(SALARY) AVG(SALARY)
------------------------------------------------------------
1001        ENGINEER         2000000     2000000
1001        MANAGER          2300000     2300000
1002        PM                980000      980000
1002        PROGRAMMER       1700000     1700000
1003        PM               2003000     2003000
1003        PROGRAMMER       4000000     4000000
1003        WEBMASTER        3750000     1875000
2001        PM               1400000     1400000
3001        PL               1800000     1800000
…
```

HAVING

❖ 그룹의 제한

- WHERE 절을 사용하여 그룹을 제한할 수 없다.

```
iSQL> SELECT dno, SUM(salary)
      2 FROM employee
      3 WHERE SUM(salary) > 2000000
      4 GROUP BY  dno;
[ERR-31061 : An aggregate function is not allowed here.
```

- HAVING 절을 이용하여 그룹을 제한할 수 있다.

```
iSQL> SELECT dno, SUM(salary)
      2 FROM employee
      3 GROUP BY  dno
      4 HAVING SUM(salary) > 2000000;
```

- 구문

```
SELECT column_name, group_function(), …
FROM table_name
[WHERE conditions]
[GROUP BY grouping_expression]
[HAVING group_condition]
[ORDER BY column_name]
LIMIT [start_index ,] row_count;
```

HAVING

❖ 예제

➢ 급여의 최대 값이 2000000이 넘는 부서를 검색

```
iSQL> SELECT dno, MAX(salary)
      2 FROM employee
      3 GROUP BY dno
      4 HAVING MAX(salary) > 2000000;
DNO           MAX(SALARY)
---------------------------
1001          2300000
1003          4000000
3002          2500000
3 rows selected.
```

WINDOWS FUNCTION

❖ 윈도우 함수

- 행과 행의 관계를 정의하거나, 행과 행을 비교, 연산하는 함수
- 그룹 함수와는 다르게 결과가 줄어들지 않음
- ALTIBASE에서는 윈도우 함수 중 그룹 별 집계함수와 순위함수를 제공

❖ 종류

- 집계함수

  SUM(), AVG(), MAX(), MIN(), COUNT(), STDDEV(), VARIANCE()

- 순위함수

  RANK(), DENSE_RANK(), ROW_NUMBER()

❖ 구문

```
SELECT window_function([argument]) OVER (
            [PARTITION BY  value expression1 ORDER BY value expression2 [ASC | DESC] ]
        )
FROM table_name;
```

WINDOWS FUNCTION

❖ 집계함수 예제

➢ 사원 정보와 사원이 속한 부서의 급여의 합, 평균, 최대, 최소 값을 검색

```
iSQL> SELECT  ename, dno, salary,
      2 SUM(salary) OVER (PARTITION BY dno ) sum_dno_sal,
      3 AVG(salary) OVER (PARTITION BY dno) avg_dno_sal,
      4 MAX(salary) OVER (PARTITION BY dno) max_dno_sal,
      5 MIN(salary) OVER (PARTITION BY dno) min_dno_sal
      6 FROM employee;
ENAME      DNO    SALARY    SUM_DNO_SAL AVG_DNO_SAL MAX_DNO_SAL MIN_DNO_SAL
------------------------------------------------------------------------
HSCHOI     1001   2000000   4300000     2150000     2300000     2000000
JHCHOI     1001   2300000   4300000     2150000     2300000     2000000
KWKIM      1002   980000    2680000     1340000     1700000      980000
HYCHOI     1002   1700000   2680000     1340000     1700000      980000
JHSEOUNG   1003   1000000   9753000     2438250     4000000     1000000
MSKIM      1003   2750000   9753000     2438250     4000000     1000000
KCJUNG     1003   2003000   9753000     2438250     4000000     1000000
YHBAE      1003   4000000   9753000     2438250     4000000     1000000
DIKIM      2001   1400000   1400000     1400000     1400000     1400000
KSKIM      3001   1800000   1800000     1800000     1800000     1800000
SJKIM      3002   2500000   2500000     2500000     2500000     2500000
…
20 rows selected.
```

WINDOWS FUNCTION

❖ 순위함수 예제

➢ 사원 정보를 급여가 높은 순서 부터 순위함수를 이용하여 검색

```
iSQL> SELECT  eno, ename,  salary,
      2 RANK() OVER (ORDER BY salary DESC )  RANK,
      3 DENSE_RANK() OVER (ORDER BY salary DESC)  DENSE_RANK,
      4 ROW_NUMBER() OVER (ORDER BY salary DESC)  ROW_NUMBER
      5 FROM employee;

ENO          ENAME        SALARY     RANK     DENSE_RANK    ROW_NUMBER
-------------------------------------------------------------------------
10           YHBAE        4000000      1           1             1
11           MSKIM        2750000      2           2             2
5            SJKIM        2500000      3           3             3
16           JHCHOI       2300000      4           4             4
14           KCJUNG       2003000      5           5             5
3            HSCHOI       2000000      6           6             6
18           CHLEE        1900000      7           7             7
12           MYLEE        1890000      8           8             8
19           KMKIM        1800000      9           9             9
4            KSKIM        1800000      9           9             10
6            HYCHOI       1700000     11          10             11
2            HJNO         1500000     12          11             12
…
20 rows selected.
```

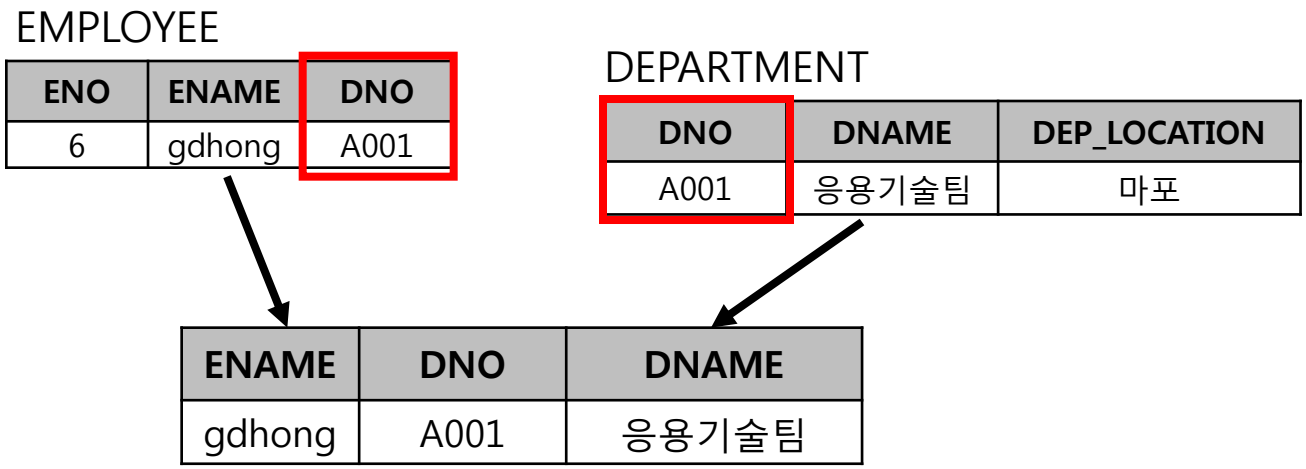
ALTIBASE

2.4 JOIN

JOIN

❖ 개념

두 개 이상의 테이블들을 연결하여 데이터를 출력



JOIN

❖ 종류

➢ JOIN 조건으로 사용되는 연산자에 따른 분류

JOIN	설명
EQUI JOIN	두 테이블 간의 칼럼 값들이 서로 일치하는 경우 JOIN 조건으로 '=' 연산자를 사용
NON EQUI JOIN	두 테이블 간의 칼럼 값들이 서로 일치하지 않는 경우 JOIN 조건으로 'BETWEEN ~ AND' 등의 범위 비교 연산자를 사용

➢ FROM 절의 JOIN 형태에 따른 분류

JOIN	설명
INNER JOIN	JOIN 조건에서 값이 일치하는 행만 반환
OUTER JOIN	JOIN 조건에서 값이 일치하지 않더라도 행을 반환

JOIN

EQUI JOIN

- 두 테이블 간의 칼럼 값들이 서로 정확하게 일치하는 경우에 사용
- WHERE 절에 '=' 연산자를 사용해서 비교

구문

```
SELECT table1.column_name, table2.column_name, …
FROM table1, table2
WHERE table1.column_name = table2.column_name;
```

예제

- 사원이 어떤 부서에서 근무하는지를 검색

```
iSQL> SELECT employee.eno, employee.ename, department.dno, department.dname
      2 FROM employee, department
      3 WHERE employee.dno = department.dno;
```

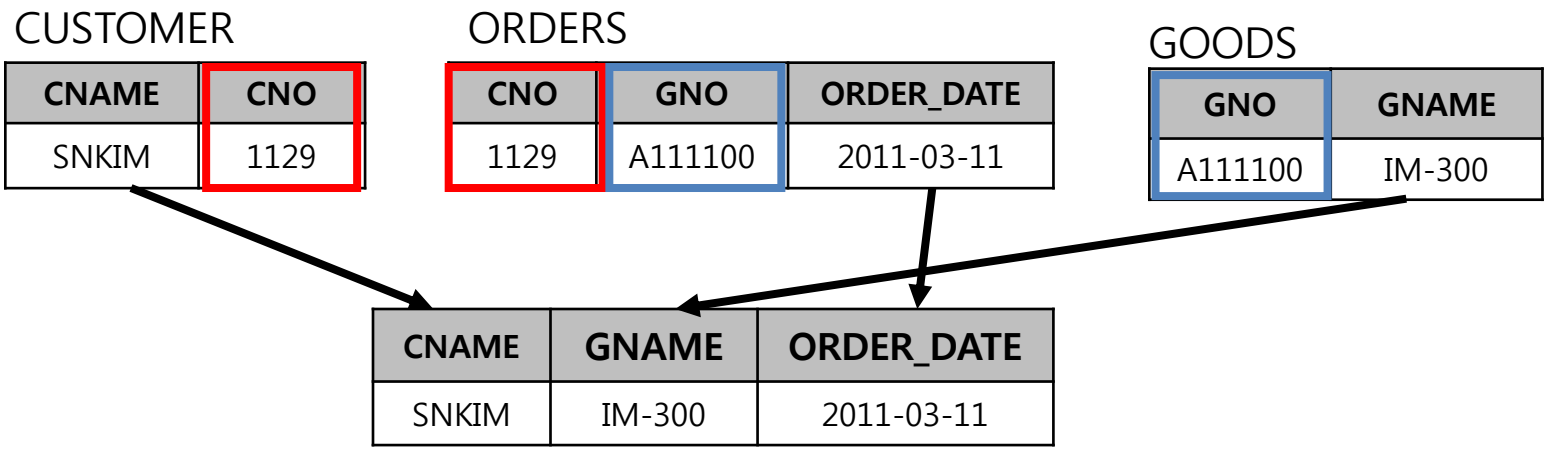
- 테이블 별칭을 통한 검색

```
iSQL> SELECT e.eno, e.ename, d.dno, d.dname
      2 FROM employee e, department d
      3 WHERE e.dno = d.dno;
```

JOIN

❖ N개의 테이블을 JOIN

- ➢ 최소 (N-1) 개의 JOIN 조건이 필요



❖ 예제

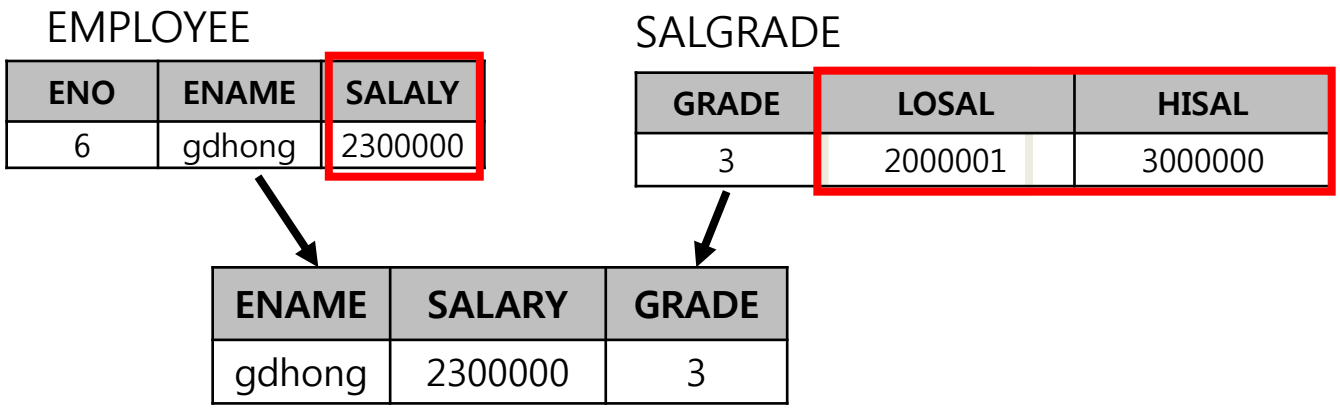
- ➢ 고객이 주문한 상품에 대한 정보를 조회

```
iSQL> SELECT c.cname, g.gname, o.orders_date
      2 FROM customer c, orders o, goods g
      3 WHERE c.cno = o.cno
      4 AND o.gno = g.gno
      5 ORDER BY c.cname;
```

JOIN

NON EQUI JOIN

- 두 테이블 간의 칼럼 값들이 서로 일치하지 않는 경우에 사용
- WHERE 절에 BETWEEN AND, >, <, >=, <= 등의 비교 연산자를 사용 구문



예제

- 사원이 받는 급여의 등급을 조회

```
iSQL> SELECT e.ename, e.salary, s.grade
      2 FROM employee e, salgrade s
      3 WHERE e.salary BETWEEN s.losal AND s.hisal;
```

JOIN

INNER JOIN

- JOIN 조건에서 값이 일치하는 행만 반환
- ON 절을 사용하여 JOIN 조건을 명시

구문

```
SELECT table1.column_name, table2.column_name, …
FROM table1 [INNER] JOIN table2
ON table1.column_name = table2.column_name;
```

예제

```
iSQL> SELECT e.eno, e.ename, d.dno, d.dname
      2 FROM employee e, department d
      3 WHERE e.dno = d.dno;
```

```
iSQL> SELECT e.eno, e.ename, d.dno, d.dname
      2 FROM employee e INNER JOIN department d
      3 ON e.dno = d.dno;
```

JOIN

❖ OUTER JOIN

- JOIN 조건을 만족하지 않는 값도 반환

EMPLOYEE

ENO	ENAME	DNO
6	gdhong	

DEPARTMENT

DNO	DNAME	DEP_LOCATION
A001	응용기술팀	마포

ENAME	DNO	DNAME
gdhong		

❖ 구문

```
SELECT table1.column_name, table2.column_name, …
FROM table1 {LEFT|RIGHT|FULL} OUTER JOIN table2
ON table1.column_name = table2.column_name;
```

- LEFT OUTER JOIN
  왼쪽 테이블을 기준으로 해서 조인조건에 만족하지 않는 값까지 반환
- RIGHT OUTER JOIN
  오른쪽 테이블을 기준으로 해서 조인조건에 만족하지 않는 값까지 반환

JOIN

- FULL OUTER JOIN

  양쪽 테이블을 기준으로 해서 조인조건에 만족하지 않는 값까지 반환

❖ 예제

- 사원이 어떤 부서에서 근무하는지를 검색
  부서를 아직 배정받지 못한 사원정보까지 함께 검색

```
iSQL> SELECT e.eno, e.ename, d.dno, d.dname
      2 FROM employee e LEFT OUTER JOIN department d
      3 ON e.dno = d.dno;
```

```
iSQL> SELECT e.eno, e.ename, d.dno, d.dname
      2 FROM department d RIGHT OUTER JOIN employee e
      3 ON e.dno = d.dno;
```

- 사원이 어떤 부서에서 근무하는지 검색
  부서를 아직 배정받지 못한 사원 및 사원이 아직 없는 부서정보도 함께 검색

```
iSQL> SELECT e.eno, e.ename, d.dno, d.dname
      2 FROM department d FULL OUTER JOIN employee e
      3 ON e.dno = d.dno;
```

JOIN

ORACLE STYLE OUTER JOIN

- 오라클 데이터베이스와 응용프로그램을 ALTIBASE HDB로 좀더 쉽게 포팅할수 있도록, 오라클 스타일의 OUTER JOIN 연산자를 지원

예제

- 사원이 어떤 부서에서 근무하는지 검색
  부서를 아직 배정받지 못한 사원정보까지 함께 검색

```
iSQL> SELECT e.eno, e.ename, d.dno, d.dname
      2 FROM employee e LEFT OUTER JOIN department d
      3 ON e.dno = d.dno;
```

```
iSQL> SELECT e.eno, e.ename, d.dno, d.dname
      2 FROM employee e , department d
      3 WHERE e.dno = d.dno(+);
```

- 사원정보가 없는 부서정보까지 함께 검색

```
iSQL> SELECT e.eno, e.ename, d.dno, d.dname
      2 FROM department d LEFT OUTER JOIN employee e
      3 ON e.dno = d.dno;
```

```
iSQL> SELECT e.eno, e.ename, d.dno, d.dname
      2 FROM department d , employee e
      3 WHERE e.dno(+) = d.dno;
```

JOIN

❖ 유의사항

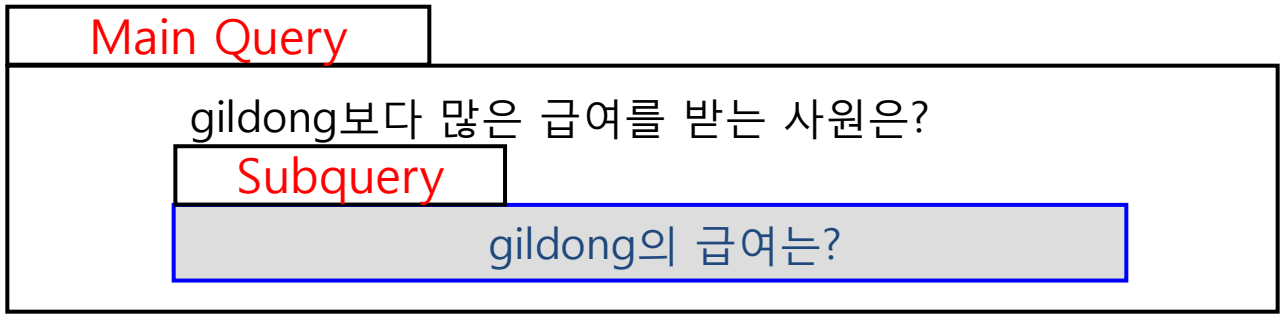
- JOIN 조건이 잘못되거나 생략했을 경우 cartesian product가 발생한다.
  (각 테이블의 행의 수를 곱한 만큼 결과 집합이 생성)
- INNER JOIN 사용 시 ON 절은 반드시 작성해야 한다.
- NATURAL JOIN, CROSS JOIN, USING 절을 이용한 JOIN은 지원하지 않는다.

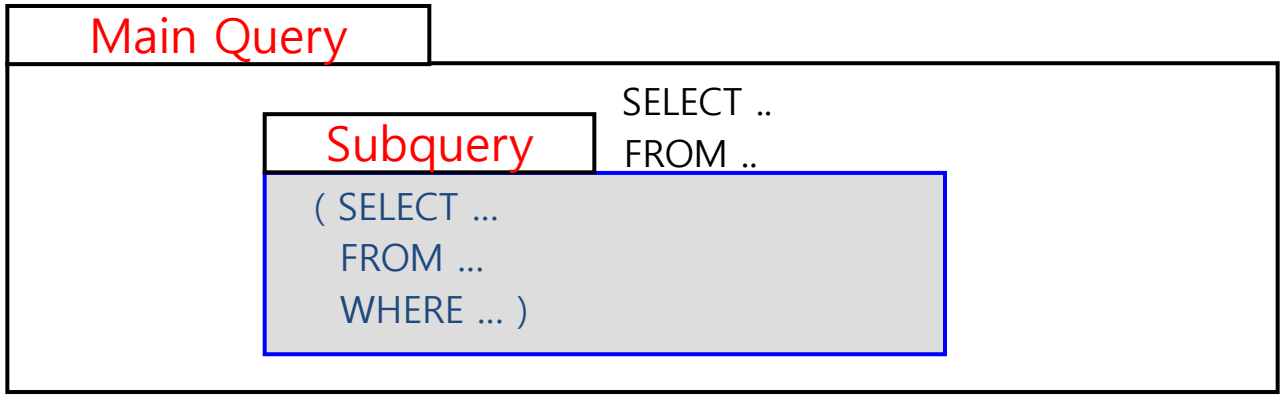
2.5 SUBQUERY

SUBQUERY

❖ 개념

- 하나의 SQL문 안에 포함되어있는 또 다른 SQL문
- main query가 실행되기 전에 먼저 실행
- subquery에서는 main query의 모든 칼럼을 참조할 수 있지만
  main query에서는 subquery의 칼럼을 참조할 수 없음





SUBQUERY

❖ 구문

```
SELECT select_list
FROM table_name
WHERE 표현식 연산자 (SELECT select_list
                     FROM table_name
                     WHERE 조건식);
```

❖ 예제

➢ gildong보다 급여를 많이 받는 사원을 출력

```
iSQL> SELECT ename, salary
    2 FROM employee
    3 WHERE salary > (SELECT salary
    4                 FROM employee
    5                 WHERE ename = 'gildong') ;
```

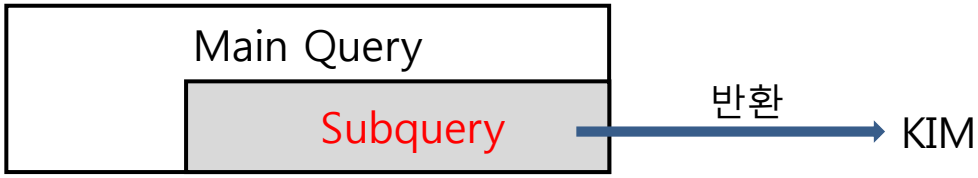
➢ LA에서 일하는 사원을 출력

```
iSQL> SELECT ename, dno
    2 FROM employee
    3 WHERE dno = (SELECT dno
    4              FROM department
    5              WHERE dep_location= 'LA') ;
```

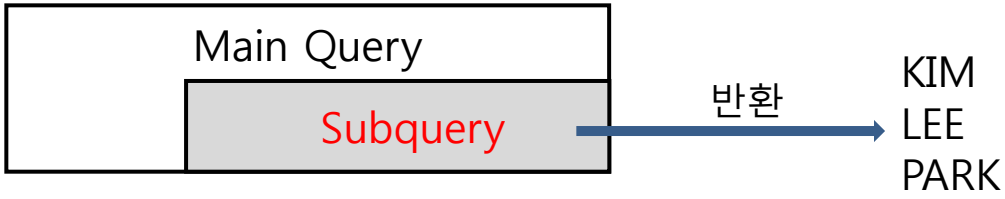
SUBQUERY

subquery 유형

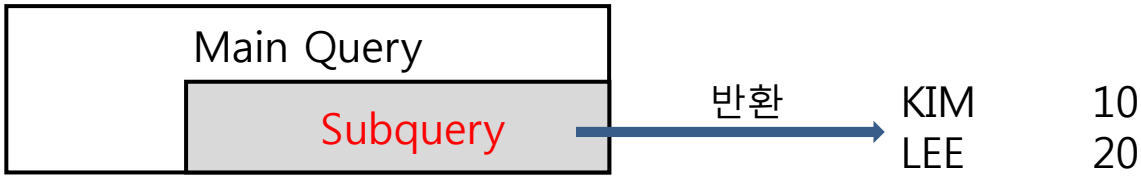
- 단일 행 subquery
  subquery에서 한 행만 반환



- 다중 행 subquery
  subquery에서 여러 행을 반환



- 다중 열 subquery
  subquery에서 여러 열을 반환



SUBQUERY

subquery 유형

- subquery에서 한 행만 반환
- 단일 행 비교연산자를 사용

연산자	의미
=	같음
>	보다 큼
>=	크거나 같음
<	보다 작음
<=	작거나 같음
<>, !=	같지 않음

예제

- 20번 사원과 같은 부서에서 일하는 사원을 출력

```
iSQL> SELECT ename, dno
      2 FROM employee
      3 WHERE dno = (SELECT dno
      4                    FROM employee
      5                    WHERE eno = 20) ;
```

SUBQUERY

- 20번 사원과 같은 부서에서 일하면서 17번 사원보다 급여를 많이 받는 사원을 출력

```
iSQL> SELECT ename, dno, salary
      2 FROM employee
      3 WHERE dno = (SELECT dno
      4                FROM employee
      5                WHERE eno = 20)
      6 AND salary  > (SELECT salary
      7                FROM employee
      8                WHERE eno=17);
```

- 급여를 가장 많이 받는 사원을 출력
  subquery 내에 그룹함수를 사용

```
iSQL> SELECT ename, salary
      2 FROM employee
      3 WHERE salary = (SELECT max(salary)
      4                 FROM employee);
```

SUBQUERY

- 평균급여가 4002번 부서의 평균급여보다 많은 부서번호를 출력
  HAVING 절에서 subquery를 사용

```
iSQL> SELECT dno, AVG(salary)
    2 FROM employee
    3 GROUP BY dno
    4 HAVING AVG(salary) > (SELECT AVG(salary)
    5                       FROM employee
    6                       WHERE dno=4002);
```

SUBQUERY

❖ 다중 행 subquery

- ➢ subquery에서 여러 행을 반환
- ➢ 다중 행 비교연산자를 사용

연산자	의미
IN	subquery 결과에 존재하는 임의의 값과 일치
ANY	subquery 결과에 존재하는 어느 하나의 값이라도 만족 >ANY : 최소값보다 큰 경우 <ANY : 최대값보다 작은 경우
ALL	subquery 결과에 존재하는 모든 값에 만족 >ALL : 최대값보다 큰 경우 <ALL : 최소값보다 작은 경우 NULL을 포함할 경우 No rows selected.
EXISTS	subquery 결과를 만족하는 값이 존재하는지 여부를 확인

❖ 예제

- ➢ 1003부서에서 근무하고 있는 사원들을 출력

```
iSQL> SELECT dno, ename, salary
    2 FROM employee
    3 WHERE salary = (SELECT salary
    4                 FROM employee
    5                 WHERE dno = 1003);
[ERR-31002 : A single-row subquery returns more than one row.]
```

SUBQUERY

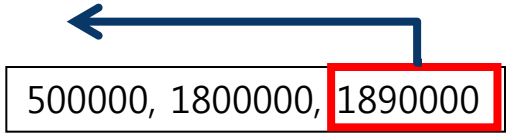
- 1003부서에서 근무하고 있는 사원들을 출력

```
iSQL> SELECT dno, ename, salary
    2 FROM employee
    3 WHERE salary IN (SELECT salary
    4                  FROM employee
    5                  WHERE dno = 1003);
DNO         ENAME                    SALARY
-----------------------------------------------
1003        YHBAE                    4000000
1003        MSKIM                    2750000
1003        KCJUNG                   2003000
1003        JHSEOUNG                 1000000
4 rows selected.
```

SUBQUERY

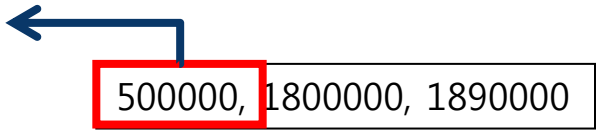
- 4002번 부서에서 근무하는 사원 중 급여를 가장 많이 받는 사람보다 급여를 적게 받는 사원들을 출력

```
iSQL> SELECT dno, ename, salary
      2 FROM employee
      3 WHERE salary <ANY (SELECT salary
      4                    FROM employee
      5                    WHERE dno=4002);
```



- 4002번 부서에서 근무하는 사원 중 급여를 가장 적게 받는 사람보다도 급여를 적게 받는 사원들을 출력

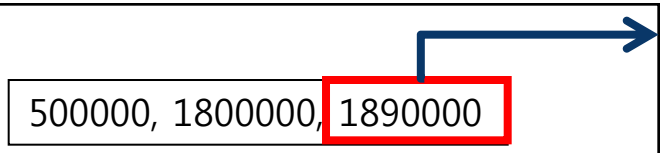
```
iSQL> SELECT dno, ename, salary
      2 FROM employee
      3 WHERE salary <ALL (SELECT salary
      4                    FROM employee
      5                    WHERE dno=4002);
```



SUBQUERY

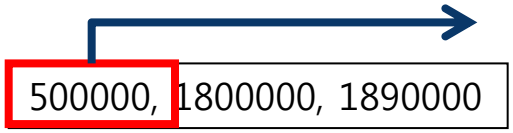
- 4002번 부서에서 근무하는 사원 중 급여를 가장 많이 받는 사람보다 급여를 많이 받는 사원들을 출력

```
iSQL> SELECT dno, ename, salary
      2 FROM employee
      3 WHERE salary >ALL (SELECT salary
      4                    FROM employee
      5                    WHERE dno=4002) ;
```



- 4002번 부서에서 근무하는 사원 중 급여를 가장 적게 받는 사람보다도 급여를 많이 받는 사원들을 출력

```
iSQL> SELECT dno, ename, salary
      2 FROM employee
      3 WHERE salary >ANY (SELECT salary
      4                    FROM employee
      5                    WHERE dno=4002) ;
```



SUBQUERY

다중 열 subquery

- subquery에서 여러 칼럼의 데이터를 반환
- ()와 IN 연산자를 이용하여 비교

예제

- 13번 사원과 같은 부서이면서 같은 직무를 하는 사람과 15번 사원과 같은 부서이면서 같은 직무를 하는 사원을 출력

```
iSQL> SELECT eno, ename, dno, emp_job
    2 FROM employee
    3 WHERE (dno, emp_job) IN (SELECT dno, emp_job
    4                          FROM employee
    5                          WHERE eno IN (13, 15)) ;
ENO         ENAME            DNO          EMP_JOB
------------------------------------------------------------
11          MSKIM            1003         WEBMASTER
13          KWKIM            1002         PM
15          JHSEOUNG         1003         WEBMASTER
3 rows selected.
```

SUBQUERY

- 13, 15번 사원과 같은 부서이면서 13, 15번 사원과 같은 직무를 하는 사원을 출력

```
iSQL> SELECT eno, ename, dno, emp_job
    2 FROM employee
    3 WHERE dno IN (SELECT dno
    4                 FROM employee
    5                 WHERE eno IN (13, 15))
    6 AND emp_job IN (SELECT emp_job
    7                   FROM employee
    8                   WHERE eno IN (13,15));

ENO         ENAME           DNO         EMP_JOB
------------------------------------------------------------
13          KWKIM           1002        PM
11          MSKIM           1003        WEBMASTER
14          KCJUNG          1003        PM
15          JHSEOUN         1003        WEBMASTER
4 rows selected.
```

SUBQUERY

scalar subquery

- SELECT 절에서 사용하는 subquery
- main query의 row 수만큼 반복되어 실행
- 조건에 만족하는 subquery의 데이터가 없을 경우 NULL을 반환
- 하나의 행, 열만 출력해야 함

예제

- 사원정보와 각 사원이 속한 부서의 평균 급여를 함께 출력

```
iSQL> SELECT ename, salary,
      2         (SELECT AVG(salary) FROM employee WHERE dno = e.dno) avg_sal
      3 FROM employee e;
ENAME           SALARY        AVG_SAL
------------------------------------------------
HSCHOI          2000000       2150000
KSKIM           1800000       1800000
SJKIM           2500000       2500000
HYCHOI          1700000       1340000
HJMIN            500000       1396666.67
…
```

SUBQUERY

inline view

- FROM 절에서 사용하는 subquery
- 동적인 view
- main query에서 inline view의 칼럼 사용 가능
  (subquery에서는 사용 불가)

예제

- 사원정보와 각 사원이 속한 부서의 평균 급여를 함께 출력

```
iSQL> SELECT e.ename, e.salary, a.avg_sal
    2 FROM employee e, (SELECT dno, AVG(salary) avg_sal FROM employee GROUP BY dno) a
    3 WHERE e.dno = a.dno;
ENAME              SALARY       AVG_SAL
-----------------------------------------
HSCHOI             2000000      2150000
JHCHOI             2300000      2150000
HYCHOI             1700000      1340000
KWKIM               980000      1340000
YHBAE              4000000      2438250
…
```

SUBQUERY

❖ 유의사항

- subquery는 ( )로 묶는다.
- subquery에 ORDER BY 절은 사용할 수 없다.

```
iSQL> SELECT ename, salary
    2 FROM employee
    3 WHERE salary > (SELECT salary
    4                 FROM employee
    5                 WHERE ename = 'gildong'
    6                 ORDER BY salary) ;
```

- 다중 행 subquery는 다중 행 연산자를 사용한다.
- scalar subquery는 main query의 레코드 수만큼 반복되어 실행되므로 성능저하가 일어날 수 있다.

ALTIBASE

2.6 SET 연산자

SET Operator

❖ 개념

여러 개의 질의 결과를 집합 연산을 수행하여 하나의 결과 집합으로 반환

❖ 종류

SET Operator	설명
UNION	합집합을 출력. 교집합 부분은 한번만 출력
UNION ALL	합집합을 출력. 교집합 부분은 반복 출력
MINUS	차집합을 출력
INTERSECT	교집합을 출력

❖ 구문

SELECT *statement1*
{UNION | UNION ALL | MINUS | INTERSECT}
SELECT *statement2*;

SET Operator

UNION

- 합집합
- 첫 번째 검색문과 두 번째 검색문의 결과를 모두 출력
- 동일한 검색 결과가 있을 경우 한 번만 출력
- 결과 순서는 보장하지 않음

예제

customer_co 테이블과 customer_pe 테이블의 모든 name 정보를 얻어온다.
만약 name이 중복되면 한번만 출력

iSQL> SELECT name FROM customer_co;

```
NAME
----------
ALTIBASE
KIM
```

iSQL> SELECT name FROM customer_pe;

```
NAME
----------
LEE
KIM
```

```
iSQL> SELECT name FROM customer_co
    2 UNION
    3 SELECT name FROM customer_pe;
NAME
--------------
ALTIBASE
KIM
LEE
```

SET Operator

UNION ALL

- 첫 번째 검색문과 두 번째 검색문의 결과를 모두 출력
- 동일한 검색 결과가 있을 경우 반복해서 출력
- 첫 번째 결과 집합을 출력 후 두 번째 결과 집합을 출력

예제

customer_co 테이블과 customer_pe 테이블의 모든 name 정보를 얻어온다.

```
iSQL> SELECT name FROM customer_co;
NAME
----------
KIM
ALTIBASE
```

```
iSQL> SELECT name FROM customer_pe;
NAME
----------
LEE
KIM
```

```
iSQL> SELECT name FROM customer_co
    2 UNION ALL
    3 SELECT name FROM customer_pe;
NAME
--------------
KIM
ALTIBASE
LEE
KIM
```

SET Operator

MINUS

- 차집합
- 첫 번째 검색결과에서 두 번째 검색 결과를 제외한 결과를 출력

예제

customer_pe 테이블에 존재하지 않는 customer_co 테이블의 name 정보를 얻어 온다.

```
iSQL> SELECT name FROM customer_co;
NAME
----------
KIM
ALTIBASE
```

```
iSQL> SELECT name FROM customer_pe;
NAME
----------
LEE
KIM
```

```
iSQL> SELECT name FROM customer_co
    2 MINUS
    3 SELECT name FROM customer_pe;
NAME
--------------
ALTIBASE
```

SET Operator

INTERSECT

- 교집합
- 첫 번째 검색 결과와 두 번째 검색 결과 중 중복된 검색 결과를 출력

예제

customer_pe 테이블에도 존재하고 customer_co 테이블에도 존재하는 name 정보를 얻어온다.

```
iSQL> SELECT name FROM customer_co;
NAME
----------
KIM
ALTIBASE
```

```
iSQL> SELECT name FROM customer_pe;
NAME
----------
LEE
KIM
```

```
iSQL> SELECT name FROM customer_co
    2 INTERSECT
    3 SELECT name FROM customer_pe;
NAME
--------------
KIM
```

SET Operator

❖유의사항

- 첫 번째 검색 column의 수와 두 번째 검색 column의 수는 같아야 한다.

```
iSQL> SELECT name, cust_no FROM customer_co
      2 INTERSECT
      3 SELECT name FROM customer_pe;
[ERR-31062 : Mismatched number of expressions in the target lists
of SELECT statements for a SET query]
```

- 첫 번째 검색 column의 타입과 두 번째 검색 column의 타입은 같아야 한다.

```
iSQL> SELECT cust_no FROM customer_co
      2 INTERSECT
      3 SELECT name FROM customer_pe;
[ERR-21011 : Invalid literal]
```

- ORDER BY절은 마지막에 위치해야 한다.

```
iSQL> SELECT name, cust_no FROM customer_co
      2 ORDER BY 1
      3 INTERSECT
      4 SELECT name FROM customer_pe;
```

- UNION ALL을 제외한 SET Operator 사용 시 집합 연산을 위한 비용이 들기 때문에 성능이 느려질 수 있다.

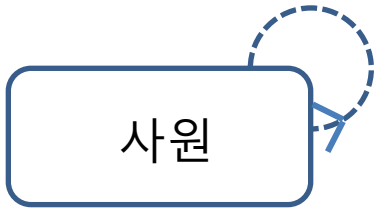
ALTIBASE

2.7 계층질의

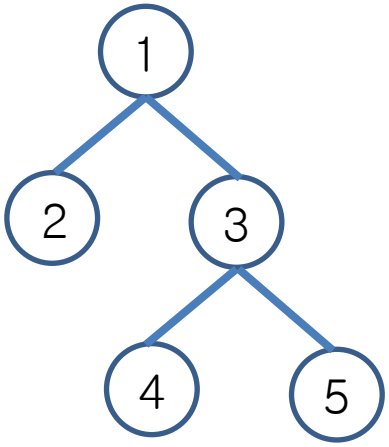
Hierarchical Query

개념

동일 테이블에 계층적으로 상위와 하위 데이터가 있을 때 계층적 조건을 만족하는 행들을 검색



1. 데이터 모델



2. 계층형 구조

eno	mgr
1	
2	1
3	1
4	3
5	3

3. 테이블

Hierarchical Query

❖ 구문

```
SELECT select_list
FROM table_name
WHERE condition
START WITH condition
CONNECT BY condition [IGNORE LOOP];
```

- START WITH 절
  - 계층 구조 전개의 시작 위치를 지정
  - 조건에 만족하는 모든 행들을 루트로 사용
- CONNECT BY 절
  - 계층 구조의 부모 행들과 자식 행들간의 관계를 식별하는 조건을 명시
  - 이전에 검색된 행과 현재 행을 구분하기 위해 PRIOR 연산자를 사용
- IGNORE LOOP
  - 계층 구성에서 loop이 발생될 경우 loop을 발생시키는 행을 질의의 결과 집합에 추가하지 않음(기본적으로 loop이 발생되면 error 발생)

Hierarchical Query

PRIOR

- 이전에 검색된 행과 현재 행을 구분하기 위해서 사용
- CONNECT BY, select_list, WHERE절 에서 사용 가능
- PRIOR 하위 = 상위
  - 상위 데이터에서 하위 데이터 방향으로 전개

```
iSQL> SELECT  ename, eno, mgr
      2 FROM employee;

ENAME      ENO        MGR
-------------------------
KIM        1
LEE        2          1
PARK       3          1
CHOI       4          3
HAN        5          3
SEO        6          2
JOON       7          2
```

```
iSQL> SELECT ename, eno, mgr
      2 FROM employee
      3 START WITH mgr IS NULL
      4 CONNECT BY PRIOR eno = mgr;

ENAME      ENO        MGR
-------------------------
KIM        1
LEE        2          1
SEO        6          2
JOON       7          2
PARK       3          1
CHOI       4          3
HAN        5          3
```

Hierarchical Query

PRIOR

- PRIOR 상위 = 하위
  - 하위 데이터에서 상위 데이터 방향으로 전개

```
iSQL> SELECT ename, eno, mgr
      2 FROM employee;
ENAME      ENO        MGR
--------------------------
KIM        1
LEE        2          1
PARK       3          1
CHOI       4          3
HAN        5          3
SEO        6          2
JOON       7          2
```

```
iSQL> SELECT ename, eno, mgr
      2 FROM employee
      3 START WITH eno = 7
      4 CONNECT BY PRIOR mgr = eno;
ENAME      ENO        MGR
--------------------------
JOON       7          2
LEE        2          1
KIM        1
```

Hierarchical Query

LEVEL

- 질의 결과가 root 데이터(가장 처음에 전개)이면 1, 그 하위 데이터는 2로 하여 leaf 데이터까지 1씩 증가시키는 pseudo column
- select_list, WHERE, ORDER BY, GROUP BY, HAVING 절에서 사용 가능

```
iSQL> SELECT ename, eno, mgr, level
      2 FROM employee
      3 START WITH mgr IS NULL
      4 CONNECT BY PRIOR eno = mgr;
ENAME    ENO        MGR        LEVEL
------------------------------------
KIM      1                     1
LEE      2          1          2
SEO      6          2          3
JOON     7          2          3
PARK     3          1          2
CHOI     4          3          3
HAN      5          3          3
```

```
iSQL> SELECT ename, eno, mgr, level
      2 FROM employee
      3 START WITH eno = 7
      4 CONNECT BY PRIOR mgr = eno;
ENAME      ENO        MGR        LEVEL
------------------------------------
JOON       7          2          1
LEE        2          1          2
KIM        1                     3
```

Hierarchical Query

IGNORE LOOP

- 데이터 전개 도중 이미 나타났던 동일한 데이터가 다시 나타나면 loop 발생
  A loop in hierarchical query detected. 오류발생
- loop 발생하더라도 오류 없이 이후 데이터는 전개하지 않음

```
iSQL> SELECT * FROM employee;
ENAME          ENO            MGR
------------------------------
KIM            1              7
…
```

```
iSQL> SELECT ename, eno, mgr, level
      2 FROM employee
      3 START WITH eno = 1
      4 CONNECT BY PRIOR eno = mgr;
ENAME       ENO       MGR       LEVEL
-------------------------------------
KIM         1         7         1
LEE         2         1         2
SEO         6         2         3
JOON        7         2         3
[ERR-311A4 : A loop in hierarchical
query detected.]
```

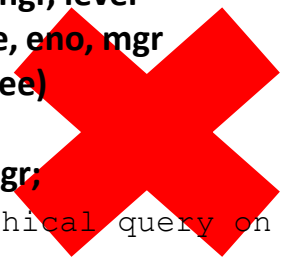
```
iSQL> SELECT ename, eno, mgr, level
      2 FROM employee
      3 START WITH eno = 1
      4 CONNECT BY PRIOR eno = mgr IGNORE LOOP;
ENAME       ENO       MGR       LEVEL
-------------------------------------
KIM         1         7         1
LEE         2         1         2
SEO         6         2         3
JOON        7         2         3
PARK        3         1         2
CHOI        4         3         3
HAN         5         3         3
```

Hierarchical Query

유의사항

- START WITH 절은 CONNECT BY 절이 없는 경우 사용할 수 없음
- START WITH, CONNECT BY 절에는 subquery를 포함할 수 없음
- inline view, view에 대해 hierarchical query는 사용할 수 없음

```
iSQL> SELECT ename, eno, mgr, level
    2 FROM (SELECT ename, eno, mgr
    3         FROM employee)
    4 START WITH eno = 1
    5 CONNECT BY eno = mgr;
[ERR-311A1 : A hierarchical query on neither a created view nor
inline view is allowed
```



ALTIBASE

3. 데이터 변경 (DML)

1. INSERT
2. UPDATE
3. DELETE
4. MOVE

3.1 INSERT

데이터 변경

DML (Data Manipulation Language)

데이터 조작어로 칭하며, 새로운 행의 삽입 및 기존 행의 삭제, 기존 칼럼의 변경과 이동 등을 가능하게 하는 SQL

DML 문의 종류 및 기능

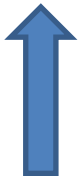
종류	기능
INSERT	데이터베이스의 특정 테이블에 새로운 레코드를 삽입
UPDATE	데이터베이스의 특정 테이블에서 조건에 해당하는 레코드를 찾아 명시한 칼럼들의 값을 변경
DELETE	특정 테이블에서 원하는 조건의 레코드를 찾아 삭제
MOVE	원천 테이블에서 원하는 조건의 레코드를 찾아 대상 테이블로 이동

INSERT

INSERT

테이블에 새로운 레코드를 삽입

고객번호	고객이름	거주도시	가입일
1	홍길동	서울	11-JAN-2009
2	김철수	경기	23-SEP-2007
3	박영희	부산	01-JAN-2010



INSERT

4	장동건	인천	18-JUL-2011

INSERT

❖ 기본 INSERT 구문

```
INSERT INTO table_name[(column_name,..)]
VALUES (value, ..);
```

❖ 예제

➢ 고객 테이블에 데이터를 입력

```
iSQL> INSERT INTO customer
      2 VALUES ( 8001011212123, 'HJKIM',
      3  'STUDENT', '025282222', 'F', '0101', '150763',
      4 '서울영등포구 여의도동 63 대한생명빌딩');
1 row inserted.
```

➢ 주문 테이블에 데이터를 입력

```
iSQL> INSERT INTO orders(ono, order_date, eno, cno, gno, qty, arrival_date, processing)
      2 VALUES (11290012, TO_DATE ('2000/11/29 01:17:00', 'YYYY/MM/DD HH:MI:SS'),
      3 12, 7610011000001, 'E111100001', 1000,
      4 TO_DATE('2000/12/01 09:10:00', 'YYYY/MM/DD HH:MI:SS'), 'D');
1 row inserted.
```

Subquery를 이용한 INSERT

Subquery를 이용한 INSERT

INSERT
INTO *table_name*[(*column_name*,..)]
SELECT *column_name*, …
FROM *table_name*
[**WHERE** *conditions*]
[**ORDER BY** *column_name*,..]
LIMIT [*start_index* ,] *row_count*;

예제

- 주문 테이블 중 processing='D' 의 조건을 만족하는 레코드를 찾아 delayed_processing 테이블에 입력

```
iSQL> DESC delayed_processing
-------------------------------------------------
NAME                TYPE                        IS NULL
-------------------------------------------------
CNO                 CHAR(14)          FIXED
ORDER_DATE          DATE              FIXED
iSQL> INSERT INTO delayed_processing
      2 SELECT cno, order_date
      3 FROM orders
      4 WHERE processing = 'D';
1 row inserted.
```

INSERT시 DEFAULT 값 사용

DEFAULT

```
INSERT INTO table_name[(column_name)]
VALUES (DEFAULT | value, ..)| DEFAULT VALUES;
```

예제

- 고객 테이블에 DEFAULT 값을 이용해 입력

```
iSQL> INSERT INTO customer
    2 VALUES (7111111431202, 'DJKIM' , 'DESIGNER' , '023442542' , DEFAULT, '1111',
    3         135010,  'Jigu Bank');
1 row inserted.
```

- t1테이블의 전체 칼럼들에 DEFAULT 값을 입력

```
iSQL> CREATE TABLE t1 (c1 INTEGER DEFAULT 0, C2 DATE DEFAULT SYSDATE);
Create seccess.
iSQL> INSERT INTO t1 DEFAULT VALUES;
1 row inserted
iSQL> SELECT * FROM t1;
C1          C2
-----------------------------------
0           2011/06/09 17:31:55
```

INSERT시 주의사항

❖ 주의사항

- 칼럼 이름을 명시하지 않았을 경우 테이블을 생성할 때 테이블의 칼럼의 개수보다 더 많은 데이터를 입력하거나 적은 데이터를 입력할 경우 오류 발생
- 칼럼 이름을 명시하지 않았을 경우 테이블을 생성할 때 나열한 칼럼 순서대로 입력
- 칼럼 이름을 명시한 경우 칼럼의 개수와 삽입할 값들의 개수는 동일해야 하며 호환 가능한 데이터 형이어야 함
- 일부 칼럼은 명시하고 일부 칼럼을 명시하지 않을 경우에 명시하지 않는 칼럼에 정의된 DEFAULT 값이 삽입되거나 NULL이 삽입
- 동일한 테이블을 이용하여 INSERT ~ SELECT 구문을 사용 가능
- NOT NULL 제약조건이 명시된 칼럼에 NULL값을 입력할 수 없음

3.2 UPDATE

UPDATE

UPDATE

테이블에서 명시한 칼럼들의 데이터를 변경

고객번호	고객이름	거주도시	가입일
1	홍길동	서울	11-JAN-2009
2	김철수	경기	23-SEP-2007
3	박영희	부산	01-JAN-2010

UPDATE

고객번호	고객이름	거주도시	가입일
1	홍길동	서울	11-JAN-2009
2	김철수	인천	23-SEP-2007
3	박영희	부산	01-JAN-2010

UPDATE

기본 UPDATE 구문

```
UPDATE table_name
SET column_name = value, ...
[WHERE conditions];
```

예제

- 사원 테이블에서 KMLEE의 급여를 변경

```
iSQL> UPDATE employee
      2 SET salary = 2500000
      3 WHERE ename = 'KMLEE';
1 row updated.
```

- 사원 테이블에서 모든 사원들의 급여를 일괄적으로 7%씩 인상

```
iSQL> UPDATE employee
      2 SET salary = salary * 1.07;
20 rows updated.
```

- 사원 테이블에서 KMKIM의 직책과 급여를 변경

```
iSQL> UPDATE employee
      2 SET emp_job = 'PM', salary = 3500000
      3 WHERE ename = 'KMKIM';
1 row updated.
```

UPDATE

다중 칼럼 수정

```
UPDATE table_name
SET (column_name, column_name, ...) = (value, value, ...)
[WHERE conditions];
```

예제

- 고객테이블에서 KSKIM 고객의 직업과 주소를 변경

```
iSQL> UPDATE customer
      2 SET (cus_job, address) = ('BANKER', 'JungGu Pusan')
      3 WHERE cname = 'KSKIM';
1 row updated.
```

UPDATE

SET 절에 subquery를 갖는 UPDATE

```
UPDATE table_name
SET column_name =  (SELECT statement …)
 [WHERE conditions];
```

예제

- 보너스 테이블에서 10번 사원의 직책을 PM으로 변경하고, bonus를 평균에서 10%인상한 값으로 변경

```
iSQL> UPDATE bonuses
      2 SET emp_job = 'PM',
      3 bonus = (SELECT 1.1 * AVG(bonus) FROM bonuses)
      4 WHERE eno = 10;
1 row updated.
```

UPDATE

❖ WHERE 절에 subquery를 갖는 데이터 수정

```
UPDATE table_name
SET column_name =  value, …
 WHERE column_name 연산자 (SELECT statement …);
```

❖ 예제

➢ 주문테이블에서 MYLEE 사원이 받은 주문량을 50개씩 줄임

```
iSQL> UPDATE orders
      2 SET qty = qty – 50
      3 WHERE eno  =  (SELECT  eno
                       FROM employee
                       WHERE ename = 'MYLEE');
12 row updated.
```

UPDATE 시 주의사항

❖ 주의사항

- UPDATE 사용 시 WHERE절을 생략할 경우, 명시한 테이블의 전체 행을 변경
- 같은 칼럼을 두 번 이상 사용 불가능
- SET 절에 subquery 사용 시 = 연산자로 이용해야하며 subquery에서 2 개 이상의 데이터를 리턴하면 에러 발생
- NOT NULL 제약조건이 있는 칼럼을 NULL 로 변경할 수 없음

ALTIBASE

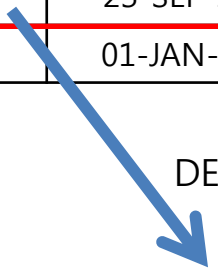
3.3 DELETE

DELETE

DELETE

테이블에서 레코드를 삭제

고객번호	고객이름	거주도시	가입일
1	홍길동	서울	11-JAN-2009
2	김철수	경기	23-SEP-2007
3	박영희	부산	01-JAN-2010



고객번호	고객이름	거주도시	가입일
1	홍길동	서울	11-JAN-2009
3	박영희	부산	01-JAN-2010

DELETE

❖ 기본 DELETE 구문

```
DELETE [FROM] table_name
[WHERE conditions];
```

❖ 예제

➢ 주문 테이블의 전체 데이터를 삭제

```
iSQL> DELETE FROM orders;
1 row deleted.
```

➢ 물품 테이블에서 물품명이 'IM-300'인 레코드를 삭제

```
iSQL> DELETE FROM goods
      2 WHERE gname = 'IM-300';
1 row deleted.
```

DELETE

❖ WHERE절에 subquery를 갖는 데이터 삭제

DELETE [**FROM**] *table_name*
[**WHERE** *column_name* 연산자 (**SELECT** *statement* …);

❖ 예제

➢ 주문 테이블에서 KMKIM 사원이 받은 주문을 삭제

```
iSQL> DELETE FROM orders
      2 WHERE eno = (SELECT eno FROM employee
      3              WHERE ename = 'KMKIM');
9 rows deleted.
```

LIMIT

LIMIT

- 삭제되는 행의 수를 제한
- DELETE 구문의 맨 끝에 사용

구문

```
DELETE [FROM] table_name
[WHERE conditions]
LIMIT [start_index ,] row_count;
```

예제

- 사원테이블에서 부서명이 'PRESALES DEPT'인 데이터중 한건을 삭제

```
iSQL> DELETE FROM employee
    2 WHERE dno = (SELECT dno
    3              FROM department
    4              WHERE dname = 'PRESALES DEPT')
    5 LIMIT 1;
Delete success.
```

DELETE 시 주의사항

DELETE시 주의사항

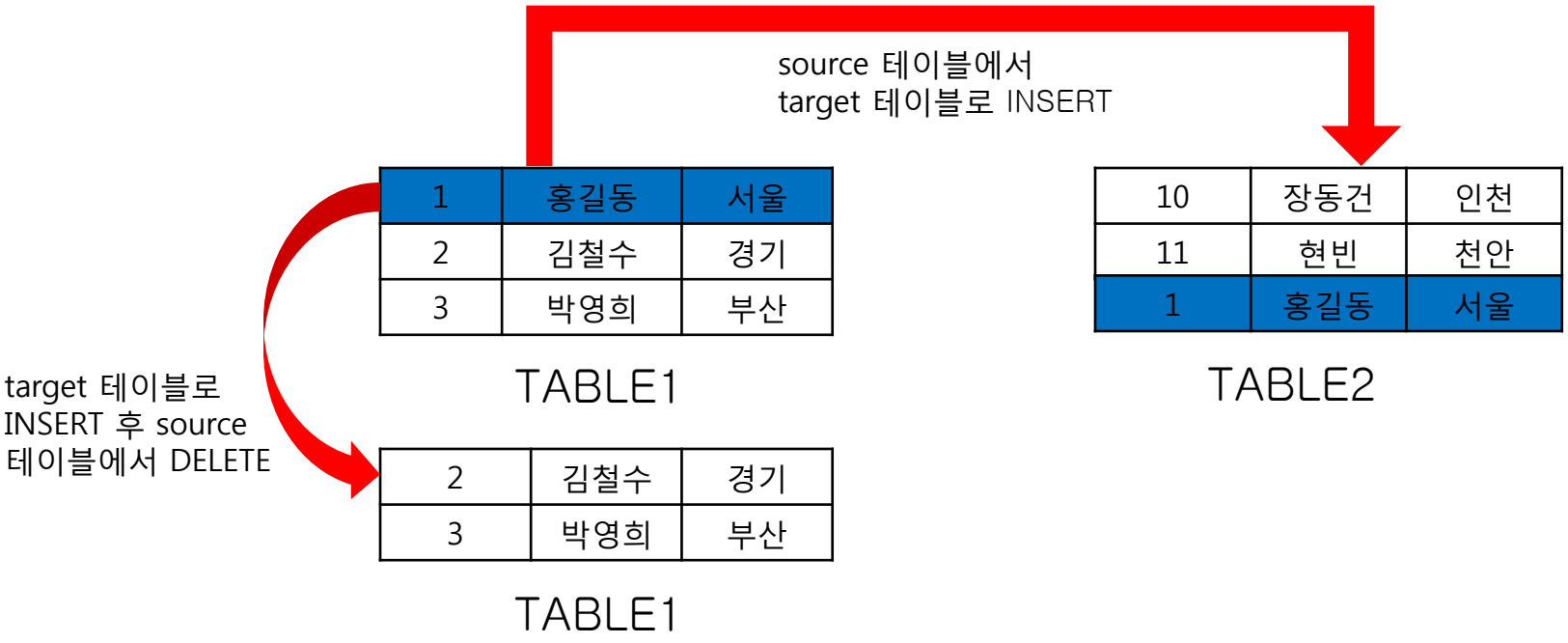
- WHERE 조건절을 생략할 경우 테이블의 모든 레코드를 삭제함
- 대량의 데이터를 DELETE 시 LIMIT을 사용하여 작업 단위를 나눌 수 있음

3.4 MOVE

MOVE

MOVE

- source 테이블에서 원하는 조건의 레코드를 찾아 target 테이블로 이동
- Target 테이블에 INSERT + source 테이블에 DELETE 수행



MOVE

기본 MOVE 구문

```
MOVE INTO target_table_name
FROM source_table_name
[WHERE conditions];
```

예제

- t1테이블에서 t2테이블로 전체 레코드를 이동

```
iSQL> MOVE INTO t2
      2 FROM t1;
10 rows moved.
```

- t1테이블에서 t2테이블로 c1=10 인 레코드만 이동

```
iSQL> MOVE INTO t2
      2 FROM t1
      3 WHERE c1 = 10;
10 rows moved.
```

MOVE

❖ 특정 칼럼만 이동

```
MOVE INTO target_table_name[(column_name,..)]
FROM  source_table_name [(column_name,..)]
[WHERE conditions];
```

❖ 예제

➢ t2 테이블의 i2=4를 만족하는 데이터 중 i1과 i2 칼럼의 데이터만 t1으로 이동

```
iSQL> MOVE INTO t1(i1, i2)
      2 FROM t2(i1, i2)
      3 WHERE T2.I2 = 4;
2 rows moved
```

➢ T2 테이블의 (i1, i2, i3) 칼럼의 데이터를 t1으로 이동

```
iSQL> MOVE INTO t1
      2 FROM t2(i1, i2, i3);
5 rows moved
```

LIMIT

LIMIT

- MOVE 구문으로 이동하는 레코드의 수를 제한
- MOVE 구문의 맨 끝에 사용

```
MOVE INTO target_table_name
FROM  source_table_name
[WHERE conditions]
LIMIT [start_index ,] row_count;
```

예제

- 고객 테이블에서 고객번호가 7001011111111과 7912319999999 사이인 레코드 중 3건만 customer_new 테이블로 이동

```
iSQL> MOVE INTO customer_new
      2 FROM customer
      3 WHERE cno BETWEEN 7001011111111 AND 7912319999999
      4 LIMIT 3;
3 rows moved.
```

MOVE 시 주의사항

❖ 주의사항

- WHERE 조건절을 생략했을 경우 테이블의 모든 행이 이동
- FROM절에서 테이블의 칼럼명을 생략할 경우, 전체 칼럼이 이동
- 칼럼의 이름을 명시할 경우 명시하지 않은 칼럼에는 NULL값이 들어감
  명시되지 않은 칼럼이 NOT NULL 제약조건인 경우는 에러 발생
- Source 테이블과 Target 테이블은 동일 테이블일 수 없음
- 두 테이블간의 칼럼 개수가 같아야 하며, 데이터 타입도 호환 가능해야 함

ALTIBASE

4. 트랜잭션 관리

1. COMMIT / ROLLBACK / SAVEPOINT

4.1 COMMIT / ROLLBACK / SAVEPOINT

트랜잭션 관리

❖ 트랜잭션의 정의

- ➢ 트랜잭션이란 하나 이상의 SQL로 이루어진 논리적인 작업단위
- ➢ 데이터베이스의 동시성을 제어하고 데이터의 일관성을 유지

❖ 트랜잭션의 특징

- ➢ ACID

  정상적인 트랜잭션의 경우 데이터베이스 무결성을 유지하기 위해서 ACID 특성을 만족시켜야 함

  - Atomicity

    트랜잭션 내의 모든 문장이 반영(COMMIT)되거나, 철회(ROLLBACK)되어야 함
  - Consistency

    트랜잭션으로 인해 데이터베이스의 무결성이 깨지지 않아야 함
  - Isolation

    한 개의 트랜잭션이 다른 트랜잭션의 영향을 받지 않아야 함
  - Durability

    완료된 트랜잭션은 어떤 상황에서도 영구적으로 유지되어야 함

COMMIT

COMMIT

지금까지 트랜잭션 안에서 수행한 모든 SQL 문의 결과를 데이터베이스에 영구적으로 반영하면서 해당 트랜잭션을 종료시키는 구문

구문

```
COMMIT;
```

예제

- HSCHOI 사원의 급여를 4000000으로 변경한 후 COMMIT을 수행

```
iSQL> UPDATE employee
      2 SET salary = 4000000
      3 WHERE ename = 'HSCHOI';
iSQL> COMMIT;
iSQL> SELECT ename, salary
      2 FROM employee
      3 WHERE ename = 'HSCHOI';
ENAME           SALARY
----------------------------
HSCHOI          4000000
```

ROLLBACK

ROLLBACK

지금까지 트랜잭션 안에서 수행한 모든 SQL 문들을 취소시키고, 데이터를 트랜잭션 수행 이전 상태로 복원

구문

```
ROLLBACK;
```

예제

- HSCHOI 사원의 급여를 변경한 후 ROLLBACK구문을 사용해 변경을 취소

```
iSQL> SELECT ename, salary FROM employee
      2 WHERE ename = 'HSCHOI';
ENAME            SALARY
-----------------------------
HSCHOI           4000000
iSQL> UPDATE employee
      2 SET salary = 4500000
      3 WHERE ename = 'HSCHOI';
iSQL> ROLLBACK;
iSQL> SELECT ename, salary FROM employee
      2 WHERE ename = 'HSCHOI';
ENAME            SALARY
-----------------------------
HSCHOI           4000000
```

SAVEPOINT

SAVEPOINT

- 하나의 트랜잭션을 여러 개의 부분으로 나누어 저장점을 표시
- ROLLBACK 구문을 이용하여 해당 부분까지만 취소가 가능

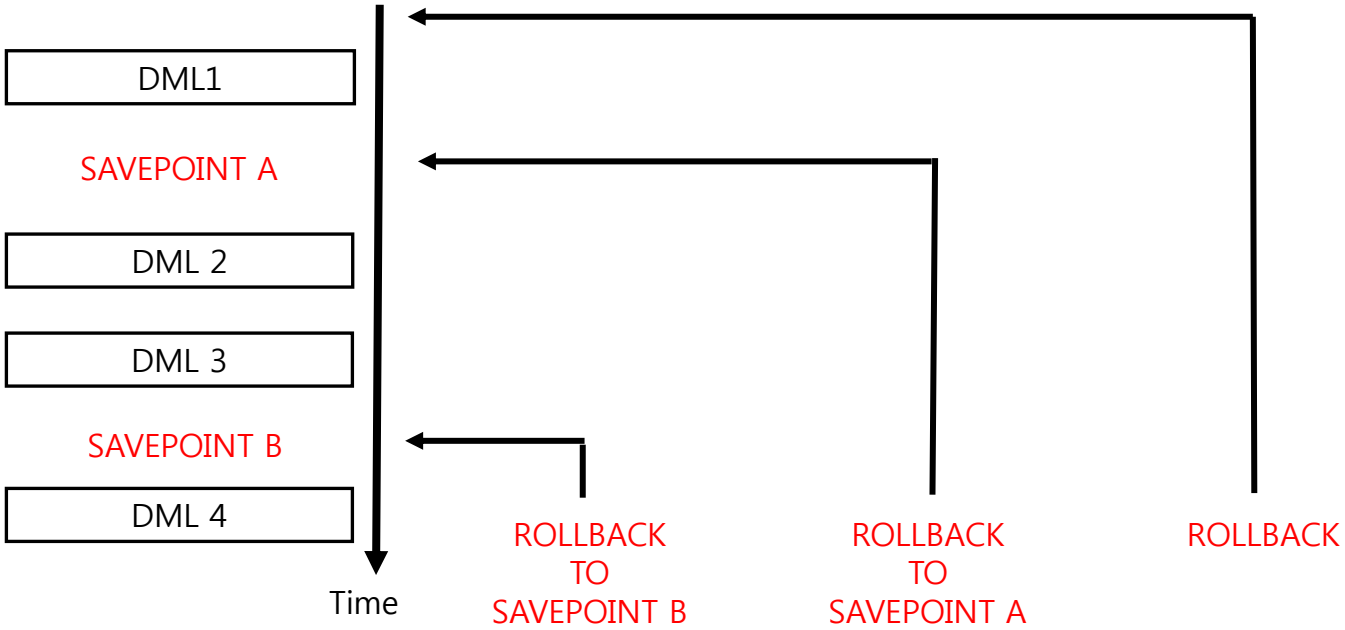
구문

SAVEPOINT *savepoint_name*;

ROLLBACK TO SAVEPOINT *savepoint_name*;

SAVEPOINT

ROLLBACK과 SAVEPOINT

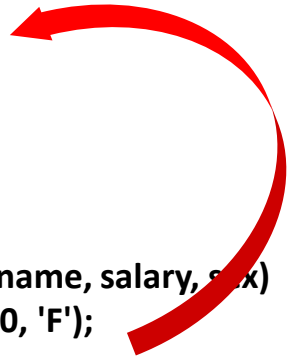


SAVEPOINT

❖ 예제

사원 테이블을 변경할 때마다 저장점을 저장하고 저장점으로 트랜잭션을 되돌림

```
iSQL> UPDATE employee
      2 SET salary = 2300000
      3 WHERE ename = 'HSCHOI';
1 row updated.
iSQL> SAVEPOINT save1;
iSQL> DELETE FROM employee
      2 WHERE ename = 'KMKIM';
1 row deleted.
iSQL> SAVEPOINT save2;
iSQL> INSERT INTO employee(eno, ename, salary, sex)
      2 VALUES(21, 'MSJUNG', 3000000, 'F');
1 row inserted
iSQL> ROLLBACK TO SAVEPOINT save1;
iSQL> SELECT * FROM employee
      2 WHERE ename IN ('HSCHOI', 'KMKIM', 'MSJUNG');
ENO         ENAME         SALARY          SEX
-----------------------------------------
20          HSCHOI        2300000         F
21          KMKIM         1000000         M
2 rows selected.
```



트랜잭션 관리

❖ 데이터 일관성

- ➢ 트랜잭션이 진행 중일 때 자신의 트랜잭션에서는 변경 이후 데이터를 조회
- ➢ 트랜잭션이 진행 중일 때 다른 트랜잭션에서는 변경 이전 데이터를 조회
- ➢ 트랜잭션을 COMMIT하면 다른 트랜잭션에서도 변경 이후 데이터를 조회

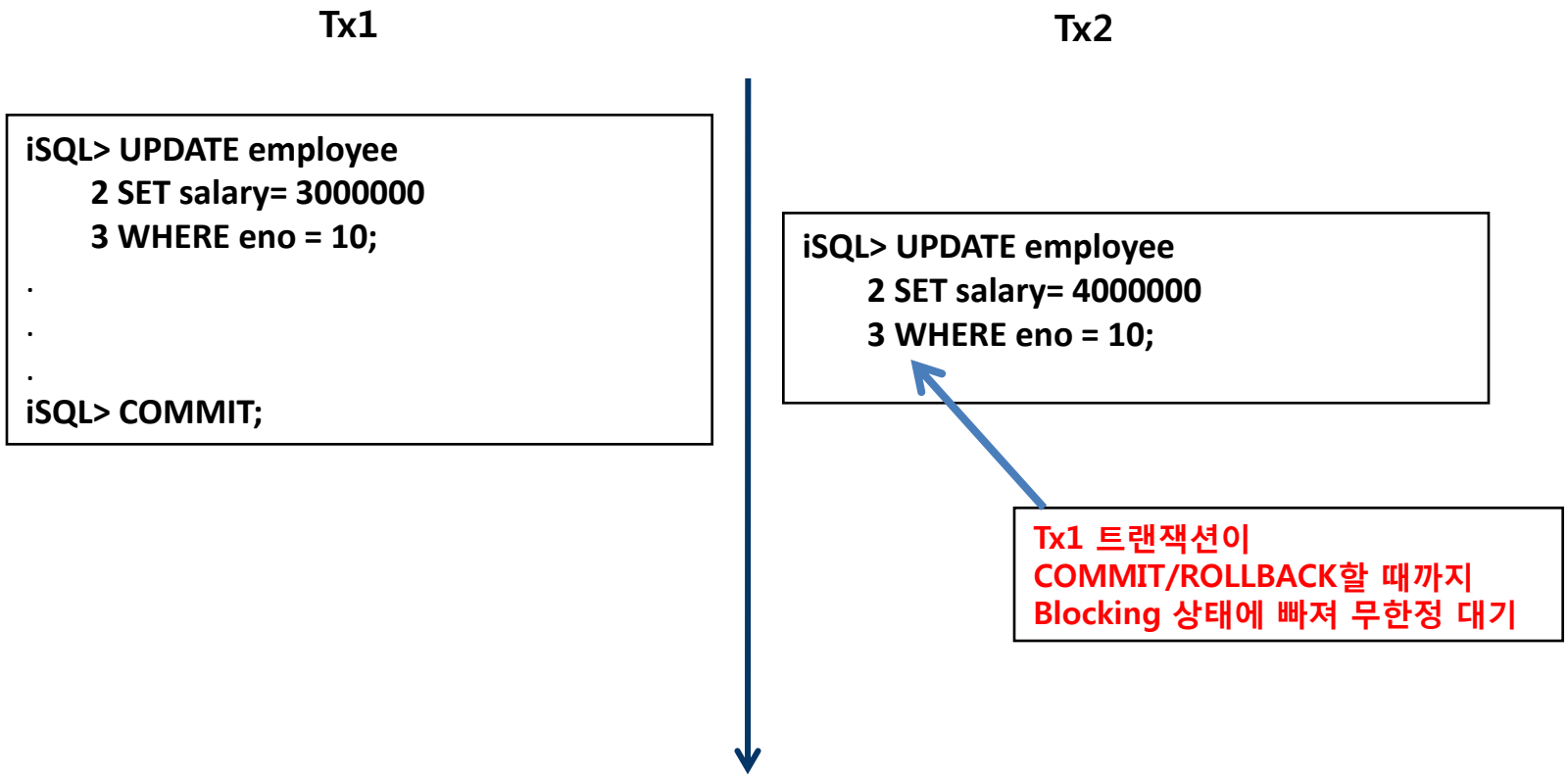
```
iSQL> SELECT eno, salary FROM employee
      2 WHERE eno = 10;
ENO         SALARY
---------------
10          2000000
iSQL> UPDATE employee
      2 SET salary= 3000000
      3 WHERE eno = 10;
iSQL> SELECT eno, salary FROM employee
      2 WHERE eno = 10;
ENO         SALARY
---------------
10          3000000
iSQL> COMMIT;
```

```
iSQL> SELECT eno, salary FROM employee
      2 WHERE eno = 10;
ENO         SALARY
---------------
10          2000000
```

```
iSQL> SELECT eno, salary FROM employee
      2 WHERE eno = 10;
ENO         SALARY
---------------
10          3000000
```

트랜잭션 관리

- 한 트랜잭션이 변경 중인 데이터를 다른 트랜잭션이 변경할 수 없음



트랜잭션 관리

❖ 트랜잭션 유의사항

- ALTIBASE는 기본적으로 AUTOCOMMIT 모드로 동작하며 DML 수행 시 자동으로(암묵적으로) COMMIT
- DDL 문은 수행이 완료됨과 동시에 자동으로 COMMIT이 수행
- NON-AUTOCOMMIT 모드에서만 SAVEPOINT가 유효함

❖ AUTOCOMMIT 모드 변경

- iSQL

```
iSQL> AUTOCOMMIT OFF;
iSQL> AUTOCOMMIT ON;
```

- 현재세션

```
iSQL> ALTER SESSION SET AUTOCOMMIT = FALSE;
iSQL> ALTER SESSION SET AUTOCOMMIT = TRUE;
```

5. ALTIBASE 객체

1. 객체종류
2. 테이블스페이스 & 사용자
3. 테이블
4. 인덱스
5. 뷰
6. 시퀀스
7. 시노님
8. 트리거
9. 큐

ALTIBASE

5.1 객체 종류

객체 종류

ALTIBASE HDB 객체

- 데이터베이스 객체는 스키마 객체와 비 스키마 객체로 구분함
- 스키마 객체

  스키마에 포함되어 특정 사용자에 의해 관리되는 객체
- 스키마 객체의 종류

종류	설명
Constraint	데이터의 정합성을 보장하기 위한 제약조건
Index	질의 성능 향상을 위해 사용되는 물리적인 저장 구조
Sequence	순차적으로 증감하는 유일한 숫자값을 자동으로 생성
Synonym	객체에 정의한 별칭
Table	행과 열로 구성된 2차원 저장 구조
Procedure/Function	절차적 질의 처리를 제공하는 객체
View	논리적 가상 테이블(logical table)
Trigger	DML 발생시 DBMS에서 절차적 질의 처리를 자동 수행
Queue	메시지를 저장하는 큐 테이블 구조의 객체

객체 종류

- 비 스키마 객체

  특정 스키마에 포함되지 않고 전체 데이터베이스 수준에서 관리되는 객체
- 비 스키마 객체의 종류

종류	설명
Replication	트랜잭션 로그를 네트워크를 통해 전송하여 실시간 데이터 복제 기능을 수행
User	스키마의 구성 단위
Tablespace	가장 큰 논리적 데이터 저장 구조
Directory	저장 프로시저에서 파일 처리를 위해 사용하는 객체

객체 종류

객체 이름 생성 규칙

- 객체들은 한 사용자 내에서 유일한 이름을 사용
- 최대 40바이트까지 사용 가능
- A-Z, a-z, 0-9, _, $ 만을 사용
- ALTIBASE의 예약어는 사용할 수 없음
- 첫 글자는 반드시 문자나 _ 로 사용해야 함
- 대소문자를 구별하거나, 특수문자 사용할 경우 큰따옴표를 이용하여 표현

  ex) CREATE TABLE "support@altibase.com" …

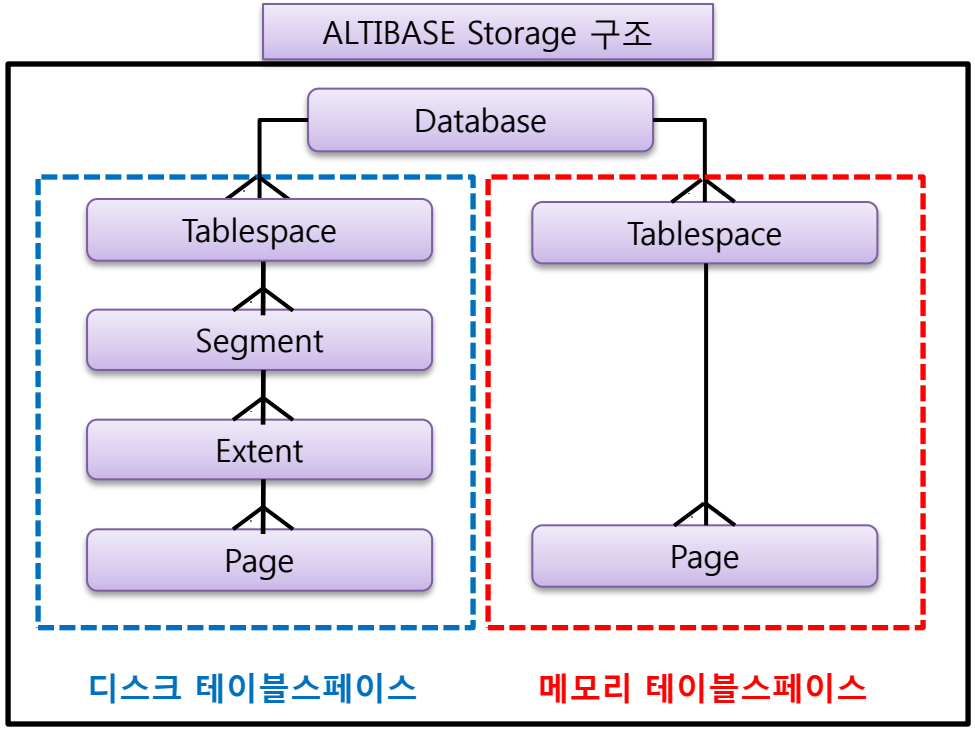
ALTIBASE

5.2 테이블스페이스 & 사용자

ALTIBASE Storage

❖ 스토리지 구조

- 하나의 데이터베이스는 한 개 이상의 테이블스페이스로 구성되며, 하나의 테이블스페이스는 다수의 세그먼트 또는, 다수의 페이지로 구성됨



- 메모리 테이블스페이스
  - 32K 크기의 페이지들로 구성
- 디스크 테이블스페이스
  - 다수의 세그먼트로 구성
  - 세그먼트는 다수의 익스텐트로 구성
  - 익스텐트는 8K 크기의 페이지 64개로 구성됨(512K)

테이블스페이스 개념

- **테이블스페이스 (Tablespace / TBS)**
  - 데이터베이스를 구성하는 최상위 논리적인 구조
  - 테이블, 인덱스 등의 데이터베이스 객체들이 저장되는 논리적인 저장소
  - 데이터베이스 운영을 위해 기본적으로 하나 이상의 테이블스페이스가 필요

테이블스페이스 종류

ALTIBASE에서 제공하는 테이블스페이스

데이터 속성에 따른 분류

- 메모리 테이블스페이스(Memory Tablespace)
- 디스크 테이블스페이스(Disk Tablespace)

생성시점에 따른 분류

- 시스템 테이블스페이스(System Tablespace)

사용자	테이블스페이스 종류
시스템	SYSTEM DICTIONARY TABLESPACE SYSTEM UNDO TABLESPACE
일반 사용자, SYS	SYSTEM MEMORY DEFAULT TABLESPACE SYSTEM DISK DEFAULT TABLESPACE SYSTEM DISK TEMPORARY TABLESPACE

- 사용자 테이블스페이스(User Tablespace)
  - 사용자의 필요에 따라 선택적으로 생성
  - 임시 TBS, 데이터 TBS(메모리 TBS, 휘발성 TBS, 디스크 TBS)

메모리 테이블스페이스 생성

❖ 메모리 테이블스페이스

- ➢ 데이터를 메모리에 저장하여, 모든 트랜잭션 처리를 메모리 상에서 처리
- ➢ 체크포인트 시에 물리적인 파일(checkpoint image file)에 저장
- ➢ DB 구동 시에 모든 데이터를 하드디스크에 저장된 물리적인 파일로부터 읽어서 메모리로 업로드 하여 사용

❖ 구문(기본)

CREATE MEMORY [**DATA**] **TABLESPACE** *tablespace_name*
SIZE *size* (**K** | **M** | **G**) ;

❖ 예제

- ➢ 초기 사이즈가 512M인 메모리 테이블스페이스를 생성

```
iSQL> CREATE MEMORY TABLESPACE test_mem
      2 SIZE 512M ;
Create success.
```

```
iSQL> CREATE MEMORY TABLESPACE test_mem
      2 SIZE 510M ;
[ERR-110EE : The initial size of the tablespace should be multiple of
expand chunk size ( EXPAND_CHUNK_PAGE_COUNT * PAGE_SIZE(32K) = 4096K )]
```

메모리 테이블스페이스는 기본적으로 4M 단위로 생성 및 확장 가능함

메모리 테이블스페이스 생성

❖구문(자동확장 추가)

```
CREATE MEMORY [DATA] TABLESPACE tablespace_name
SIZE size (K | M | G)
[AUTOEXTEND [ON [NEXT size] [MAXSIZE size] | OFF) ]
```

❖예제

➢ 초기 사이즈가 512M이고, 128M 단위로 자동 확장 가능한 최대 크기가 2G 인 메모리 테이블스페이스를 생성

```
iSQL> CREATE MEMORY TABLESPACE test_mem
    2 SIZE 512M
    3 AUTOEXTEND ON NEXT 128M MAXSIZE 2G ;
Create success.
```

➢ 초기 사이즈가 512M 이고, 자동확장을 하지 않는 메모리 테이블스페이스를 생성

```
iSQL> CREATE MEMORY TABLESPACE test_mem
    2 SIZE 512M
    3 AUTOEXTEND OFF ;
Create success.
```

메모리 테이블스페이스 생성

❖구문(체크포인트 경로 추가)

```
CREATE MEMORY [DATA] TABLESPACE tablespace_name
SIZE size (K | M | G)
[AUTOEXTEND [ON [NEXT size] [MAXSIZE size] | OFF) ]
[CHECKPOINT PATH 'path' [SPLIT EACH size]] ;
```

❖예제

➢ 초기 사이즈가 512M이고, 최대 1G까지 128M 단위로 자동 확장 가능한 메모리 테이블스페이스를 생성 (체크포인트 이미지 파일은 다중화를 위해 3개의 디렉토리에 나누어 저장)

```
iSQL> CREATE MEMORY TABLESPACE test_mem
    2 SIZE 512M
    3 AUTOEXTEND ON NEXT 128M MAXSIZE 1G
    4 CHECKPOINT PATH '/dbs/path1', '/dbs/path2', '/dbs/path3' ;
Create success.
```

```
iSQL> CREATE MEMORY TABLESPACE test_mem
    2 SIZE 512M
    3 AUTOEXTEND ON NEXT 128M MAXSIZE 1G
    4 CHECKPOINT PATH '/dbs/path1', '/dbs/path2', '/dbs/path3'
    5 SPLIT EACH 256M ;
Create success.
```

휘발성 테이블스페이스 생성

❖ 휘발성 테이블스페이스

- 메모리 테이블스페이스와 동일한 구조의 테이블스페이스
- 체크포인트를 하지 않고, 리두 로그를 기록하지 않음

❖ 구문

```
CREATE VOLATILE [DATA] TABLESPACE tablespace_name
SIZE size (K | M | G)
[AUTOEXTEND [ON [NEXT size][MAXSIZE size] | OFF) ] ;
```

❖ 예제

- 초기 사이즈가 512M이고, 최대 1G까지 128M 단위로 자동 확장 가능한 휘발성 테이블스페이스를 생성

```
iSQL> CREATE VOLATILE DATA  TABLESPACE test_mem
      2 SIZE 512M
      3 AUTOEXTEND ON NEXT 128M MAXSIZE 1G ;
Create success.
```

디스크 테이블스페이스 생성

❖ 디스크 테이블스페이스

- ➢ 모든 데이터가 디스크에 저장되는 테이블스페이스
- ➢ 물리적으로 데이터 파일로 구성되고, 논리적으로 세그먼트, 익스텐트, 페이지로 구성

❖ 구문

```
CREATE [DISK] [DATA] TABLESPACE tablespace_name
DATAFILE 'datafile_name' ;
```

❖ 예제

- ➢ 기본 경로에 데이터 파일 test01.dbf 를 생성하는 test_disk 테이블스페이스를 생성

```
iSQL> CREATE TABLESPACE test
      2 DATAFILE 'test01.dbf';
Create success.
```

디스크 테이블스페이스 생성

❖구문(자동확장 추가)

```
CREATE [DISK] [DATA] TABLESPACE tablespace_name
DATAFILE 'datafile_name '
[SIZE size (K | M | G) ] [REUSE]
[AUTOEXTEND [ON [NEXT size][MAXSIZE size] | OFF)];
```

❖예제

➢ 데이터 파일 test01.dbf, test02.dbf, test03.dbf 로 구성된 100MB의 test_disk 테이블스페이스를 생성(자동확장 하지 않음)

```
iSQL> CREATE TABLESPACE test_disk
      2 DATAFILE 'test01.dbf', 'test02.dbf',  'test03.dbf'
      3 SIZE 100M AUTOEXTEND OFF;
Create success.
```

➢ 데이터 파일 test01.dbf, test02.dbf, test03.dbf 로 구성되고, 초기크기가 100MB, 2G까지 자동 확장하는 test_disk 테이블스페이스를 생성

```
iSQL> CREATE TABLESPACE test_disk
      2 DATAFILE 'test01.dbf', 'test02.dbf',  'test03.dbf'
      3 SIZE 100M AUTOEXTEND ON NEXT 10M MAXSIZE 2G ;
Create success.
```

임시 테이블스페이스 생성

❖ 임시 테이블스페이스

- 디스크 데이터에 대한 질의 수행 중 생성되는 임시 결과를 저장하기 위한 테이블스페이스
- 트랜잭션이 종료하는 시점에 해당 질의가 남긴 모든 데이터들은 사라짐

❖ 구문

```
CREATE TEMPORARY TABLESPACE   tablespace_name
TEMPFILE 'tempfile_name '
[SIZE size (K | M | G) ] [REUSE]
[AUTOEXTEND [ON [NEXT size][MAXSIZE size] | OFF) ] ;
```

❖ 예제

- tbs.temp로 구성된 test_temp 임시 테이블스페이스를 생성. 임시 파일의 크기는 10M이고, 5M 크기로 자동 확장.

```
iSQL> CREATE TEMPORARY TABLESPACE test_temp
      2 TEMPFILE 'tbs.temp'
      3 SIZE 10M AUTOEXTENED ON NEXT 5M ;
Create success.
```

테이블스페이스 변경

❖ 테이블스페이스 변경

- ALTER TABLESPACE 구문으로 테이블스페이스에 데이터파일 추가/삭제, 자동확장 설정, 최대크기 등에 대해서 변경이 가능

❖ 구문(자동확장 추가)

```
ALTER TABLESPACE tablespace_name
{ [ ADD | DROP ] [ DATAFILE | TEMPFILE ] ...
  [ ALTER [ DATAFILE | TEMPFILE ] file_name SIZE ...
          [ AUTOEXTEND [ON [NEXT size][MAXSIZE size] | OFF) ]
};
```

❖ 예제

- test_disk 테이블스페이스에 64 MB의 데이터 파일 test01.dbf를 추가(공간이 더 필요할 때는 500K 씩 파일이 자동확장)

```
iSQL> ALTER TABLESPACE test_disk
      2 ADD DATAFILE 'test01.dbf' SIZE 64M
      3 AUTOEXTEND ON NEXT 500K;
Alter success.
```

테이블스페이스 변경

- test_disk 디스크 테이블스페이스가 자동확장을 하지 않도록 변경

```
iSQL> ALTER TABLESPACE test_disk
      2 ALTER DATAFILE ' test01.dbf' AUTOEXTEND OFF;
Alter success.
```

- test_disk 테이블스페이스의 데이터파일 test01.dbf 를 삭제하시오.

```
iSQL> ALTER TABLESPACE test_disk
      2 DROP DATAFILE ‘test01.dbf‘;
Alter success.
```

테이블스페이스 삭제

❖ 테이블스페이스 삭제

- 데이터베이스에서 테이블스페이스를 제거함
- 시스템 테이블스페이스는 삭제할 수 없음

❖ 구문

```
DROP TABLESPACE tablespace_name
[INCLUDING CONTENTS]
[ AND DATAFILES | CASCADE CONSTRAINTS] ;
```

❖ 예제

- 메모리 테이블스페이스 test_mem을 삭제

```
iSQL> DROP TABLESPACE test_mem;
Drop success.
```

- 디스크 테이블스페이스 test_disk의 모든 객체, 데이터 파일들과 함께 테이블스페이스를 삭제

```
iSQL> DROP TABLESPACE test_disk
    2 INCLUDING CONTENTS AND DATAFILES;
Drop success.
```

사용자 개념

- **USER**
  - 스키마를 구성하는 단위
  - DB 생성 초기에는 시스템 관리자인 SYSTEM_ 와 SYS 사용자만 존재
  - 일반 스키마를 구축하기 위해서는 일반 사용자를 생성
  - 사용자 종류
    - SYSTEM_
      - 메타 테이블의 소유자
    - SYS
      - DBA로 모든 권한을 가지며, 시스템 수준의 모든 작업을 수행함
    - 일반 사용자
      - CREATE 구문을 통해 생성된 사용자로 자신이 소유한 스키마 객체에 대한 권한을 가짐

사용자 개념

비밀번호 규칙

- 객체 이름과 유사한 규칙
- 지정할 수 있는 최대 크기는 운영체제에 따라 다르며 8~40자 사이
  - Solaris10, Windows XP 이후 : 40자
  - Solaris 2.8 이후, Windows NT : 11자
  - 그 외 운영체제 : 8자
- 최대 크기보다 많이 입력된 경우, 이후 문자는 무시함

사용자 생성

사용자 생성

- CREATE 구문을 이용하여 생성하며, 사용자 생성 시 비밀번호를 지정하고 테이블스페이스를 지정할 수 있음
- 최대 크기보다 많이 입력된 경우, 이후 문자는 무시함

구문(기본)

```
CREATE USER user_name
IDENTIFIED BY password;
```

예제

- 사용자 명이 alti 암호가 altibase인 사용자를 생성

```
iSQL> CREATE USER alti
    2  IDENTIFIED BY altibase;
Create success.
```

사용자 생성

구문(테이블스페이스 추가)

```
CREATE USER user_name
IDENTIFIED BY password
[DEFAULT TABLESPACE tablespace_name]
[TEMPORARY TABLESPACE tablespace_name
[ACCESS tablespace_name ON|OFF];
```

예제

- 사용자 명이 alti, 암호가 altibase 인 사용자가 SYS_TBS_MEM_DATA 테이블스페이스에 대해 사용 권한을 갖도록 생성

```
iSQL> CREATE USER alti
    2  IDENTIFIED BY altibase
    3  ACCESS sys_tbs_mem_data ON;
Create success.
```

사용자 생성

- 사용자 명이 alti, 암호는 altibase인 사용자가 default tablespace로 test_disk를, temporary tablespace로 SYS_TBS_DISK_TEMP를 사용하며, test_mem 테이블스페이스에 대해 사용권한을 갖도록 생성

```
iSQL> CREATE USER alti
    2 IDENTIFIED BY altibase
    3 DEFAULT TABLESPACE test_disk
    4 TEMPORARY TABLESPACE sys_tbs_disk_temp
    5 ACCESS test_mem ON;
Create success.
```

사용자 변경

❖ 사용자 변경

- ➢ 사용자의 암호, 기본 테이블스페이스 / 임시 테이블스페이스/ 테이블스페이스 사용권한을 변경

❖ 구문

ALTER USER *user_name*
[**IDENTIFIED BY** *password*]
[**DEFAULT TABLESPACE** *tablespace_name*]
[**TEMPORARY TABLESPACE** ***t****ablespace_name*
[**ACCESS** *tablespace_name* **ON/OFF**];

❖ 예제

- ➢ alti 사용자의 비밀번호를 edu 로 변경

```
iSQL> ALTER USER alti
      2 IDENTIFIED BY edu;
Alter success.
```

사용자 변경

- alti 사용자의 default tablespace를 sys_tbs_disk_data로 변경

```
iSQL> ALTER USER alti
    2 DEFAULT TABLESPACE sys_tbs_disk_data;
Alter success.
```

- alti 사용자가 sys_tbs_mem_data에 접근하지 못하도록 변경

```
iSQL> ALTER USER alti
    2 ACCESS sys_tbs_mem_data OFF;
Alter success.
```

사용자 삭제

사용자 삭제

- 데이터베이스에 명시된 사용자를 제거함

구문

```
DROP USER user_name
[CASCADE];
```

예제

- 사용자 alti와 사용자가 소유한 모든 객체들을 삭제

```
iSQL> DROP USER alti
    2 CASCADE;
Drop success.
```

ALTIBASE

5.3 테이블

테이블 개념

❖ 테이블 정의

- 데이터를 저장하기 위한 가장 기본적인 논리적 데이터 저장 구조
- 열(column)과 행(row)으로 구성됨
- 관계형 데이터베이스 시스템에서 가장 중요한 객체

❖ 테이블 종류

- 메모리 테이블
  - 데이터를 메모리에 적재 후, 모든 트랜잭션 처리를 메모리 상에서 진행함
  - 디스크 I/O가 거의 발생하지 않음

- 디스크 테이블
  - 데이터를 디스크에 적재 후, 일부의 데이터를 메모리 영역(버퍼)에 올려서 사용
  - 데이터 변경이나 조회 시 디스크 I/O가 발생할 수 있음

테이블 생성

❖ 구문(기본)

```
CREATE TABLE table_name
( column_name datatype [DEFAULT expr] [column_constraint],
...
[table_constraint] ) ;
```

❖ 예제

```
iSQL> CREATE TABLE emp
    2 (ename VARCHAR(10),
    3  age NUMBER,
    4 phone VARCHAR(20));
Create success.
```

테이블 개념

❖ 데이터 타입

분류	데이터 타입		설명
문자형	CHAR		고정 길이 문자형 데이터 타입. 최대 32KB
	NCHAR		고정 길이 유니코드 문자형 데이터 타입
	VARCHAR		가변 길이 문자형 데이터 타입. 최대 32KB
	NVARCHAR		가변 길이 유니코드 문자형 데이터 타입
숫자형	Native Type	SMALLINT	2Bytes 크기의 정수형 데이터 타입
		INTEGER	4Bytes 크기의 정수형 데이터 타입
		BIGINT	8Bytes 크기의 정수형 데이터 타입
		REAL	C의 FLOAT과 동일한 데이터 타입. 4Bytes크기
		DOUBLE	8 bytes 크기의 부동 소수점 데이터 타입
	Non-Native Type	NUMERIC	고정 소수점 데이터 타입
		NUMBER	NUMERIC 데이터 타입과 동일 precision, scale 아무것도 안주면 FLOAT와 동일
		DECIMAL	NUMERIC 데이터 타입과 동일
		FLOAT	부동 소수점 데이터 타입 precision만 지정 가능

테이블 개념

❖ 데이터 타입

분류	데이터 타입	설명
날짜형	DATE	8Bytes 마이크로 초까지 저장/검색 가능. HH는 HH24와 동일
이진형	BIT	0과 1로만 이루어진 고정 길이 이진 데이터 타입 최대 크기 131068
	BYTE	고정 길이 이진 데이터 타입 최대 32KB
	NIBBLE	가변 길이 이진 데이터 타입 최대 크기 254
대용량	BLOB	대용량 이진형 데이터 타입 최대 2GB
	CLOB	대용량 문자형 데이터 타입 최대 2GB
공간형	GEOMETRY	공간형 데이터 타입. 최대100MB

데이터 타입

❖ 문자형 데이터 타입

➢ CHAR

- 명시된 크기만큼 고정 길이를 가지는 문자형 데이터 타입
- 명시된 크기보다 입력 값의 크기가 작을 경우 뒷부분은 공백으로 채워짐

CHAR [(*size*)]

iSQL> CREATE TABLE emp (ename CHAR(20));

➢ VARCHAR

- 명시된 크기 내에서 가변 길이를 가지는 문자형 데이터 타입
- 명시된 크기는 최대 저장 가능한 문자열의 길이

VARCHAR [(*size*)]

iSQL> CREATE TABLE emp (ename VARCHAR(50));

데이터 타입

- NCHAR
  - 명시된 크기만큼 고정 길이를 가지는 문자형 데이터 타입
  - 칼럼의 문자 하나당 크기는 내셔널 캐릭터 셋의 설정에 따라서 다르게 저장됨

NCHAR [(*size*)]

- NVARCHAR
  - 명시된 크기 내에서 가변 길이를 가지는 문자형 데이터 타입
  - 칼럼의 문자 하나당 크기는 내셔널 캐릭터 셋의 설정에 따라서 다르게 저장됨

NVARCHAR [(*size*)]

데이터 타입

❖ 숫자형 데이터 타입 (Native Type)

- BIGINT
  - 8 바이트 크기의 정수형 데이터 타입
  - C의 long(64 bit) 이나 long long(32 bit)과 동일한 데이터 타입

```
BIGINT

iSQL> CREATE TABLE emp ( eno BIGINT) ;
```

- DOUBLE
  - 8 바이트 크기의 부동 소수점형 타입 (C의 double과 동일함)

```
DOUBLE

iSQL> CREATE TABLE emp ( price DOUBLE) ;
```

- INTEGER
  - 4 바이트 크기의 정수형 데이터 타입 (C의 int와 동일함)

```
INTEGER

iSQL> CREATE TABLE emp ( eno INTEGER) ;
```

데이터 타입

- SMALLINT
  - 2바이트 크기의 정수형 데이터 타입
  - C의 short와 동일한 데이터 타입

```
SMALLINT

iSQL> CREATE TABLE t1 ( c1 SMALLINT) ;
```

- REAL
  - 4 바이트 크기의 부동 소수점형 타입
  - C의 float 와 동일한 데이터 타입

```
REAL

iSQL> CREATE TABLE t1 ( c1 REAL) ;
```

데이터 타입

❖ 숫자형 데이터 타입 (Non Native Type)

➢ NUMERIC

- Precision과 scale을 가지는 숫자형 데이터 타입으로 precision 만큼의 유효 숫자와 scale 만큼의 소수점 이하 정밀도를 가지는 고정 소수점형
- precision과 scale이 모두 생략되면 precision은 38, scale은 0인 정수를 표현하는 형식인 고정소수점 으로 사용

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

iSQL> CREATE TABLE t1 (c1 NUMERIC(5,0)) ;

➢ FLOAT

- -1E+120에서 1E+120까지의 부동 소수점 숫자 데이터 타입

FLOAT [(*precision*)]

➢ DECIMAL

- NUMERIC 데이터 타입과 동일한 데이터 타입

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

데이터 타입

- NUMBER
  - NUMERIC type의 alias형으로 precision과 scale이 명시되지 않으면 FLOAT과 동일하게 취급

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

iSQL> CREATE TABLE t1 (c1 NUMBER(10,2)) ;

데이터 타입

❖ 날짜형 데이터 타입

- DATE
  - 날짜를 표현하는 데이터 타입
  - 8 바이트 크기를 가짐
  - Microsec까지 표현 가능

DATE

iSQL> CREATE TABLE emp (birth DATE) ;

❖ 이진 데이터 타입

- BYTE
  - 명시된 크기만큼 고정된 길이를 가지는 이진 데이터 타입(16진수로 표현)
  - 1바이트는 2개의 문자를 입력할 수 있음
    (ex. BYTE(1) => 00~FF)

BYTE [(*size*)]

iSQL> CREATE TABLE orders (gno BYTE(2)) ;

데이터 타입

- NIBBLE
  - 명시된 크기만큼 가변 길이를 가지는 이진 데이터 타입 (16진수로 표현)
  - 칼럼의 크기는 기본값으로 한 개의 문자 크기이며, 최대 255 크기까지 허용
  - BYTE와 달리 명시된 size 만큼의 문자만을 입력 가능
    (ex. NIBBLE(1) => 0~F)

NIBBLE [(*size*)]

- BIT
  - 0과 1로만 이루어진 고정 길이를 갖는 이진 데이터 타입
  - 기본값으로 1 bit이며, 최대 길이는 131068 bit

BIT [(*size*)]

- VARBIT
  - 0과 1로만 이루어진 가변 길이를 갖는 이진 데이터 타입
  - 기본값으로 1bit이며, 최대 길이는 131068 bit (128KB)

VARBIT [(*size*)]

iSQL> CREATE TABLE emp (code VARBIT(10)) ;

데이터 타입

LOB 데이터 타입

- 대용량 데이터를 저장할 수 있는 데이터 타입
- 최대 2G까지 저장
- 이진 데이터를 저장하는 BLOB과 문자열 데이터를 저장하는 CLOB으로 구분
- BLOB
  - 이진형 데이터를 저장하기 위한 타입으로 2G 크기 내에서 가변 길이를 가지는 이진형 데이터 타입

```
BLOB
```

- CLOB
  - 문자형 대용량 데이터를 저장하기 위한 것으로, 2GB 크기 내에서 가변 길이를 가지는 문자형 데이터 타입

```
CLOB
```

- 예제

```
iSQL> CREATE TABLE emp ( address CLOB, image BLOB) ;
```

데이터 타입 유의사항

❖ 숫자형 데이터타입

가능하면 SMALLINT, INTEGER, BIGINT, REAL, DOUBLE등 native type으로 지정을 권장

=> 데이터 처리시 변환 비용에 따른 overhead를 줄일 수 있음.

저장 공간의 효율성이 좋음

❖ LOB 타입 제약사항

- NON-AUTOCOMMIT MODE로 수행하지 않으면 오류 발생
  - “Connection is in autocommit mode. One can not operate on LOB datas with autocommit mode on”
- 프로시저나 트리거에서 사용할 수 없음
- Temp Tablespace에서 사용할 수 없음
- 인덱스를 생성할 수 없음

제약조건

❖ Constraints in ALTIBASE

제약조건	설명
PRIMARY KEY	PRIMARY KEY 값은 테이블 내에서 유일해야 하며, NULL 값을 가질 수 없다. 한 테이블 내에 정의 가능한 PRIMARY KEY 개수는 하나이다.
UNIQUE	UNIQUE 값은 테이블 내에서 유일해야 한다. NULL 값은 포함 가능하다.
FOREIGN KEY	다른 테이블의 PRIMARY KEY/UNIQUE 값을 참조하는 외래키를 정의한다.
NOT NULL/NULL	NOT NULL : 해당 칼럼에 NULL 값을 가질 수 없다. NULL : 해당 칼럼에 NULL 값을 가질 수 있다.
DEFAULT	칼럼 값을 명시해 주지 않을 경우 기본으로 저장될 값을 명시. 지정되지 않으면 기본 값은 NULL로 명시된다.

※ CHECK 제약조건은 제공하지 않는다.

테이블 생성

구문(제약조건)

- column_level

```
CREATE TABLE table_name
( column_name datatype [DEFAULT expr] [CONSTRAINT constraint_name] constraint_type,
...
) ;
```

- table_level

```
CREATE TABLE table_name
( column_name datatype [DEFAULT expr] ,
 ...
 [CONSTRAINT constraint_name] constraint_type (column_name),
 ...
) ;
```

- NOT NULL, NULL, DEFAULT 제약조건은 column_level 로만 정의 가능
- 복합 칼럼으로 구성되는 제약조건은 table_level로만 정의 가능

제약조건

❖ 예제

- Primary key

```
iSQL> CREATE TABLE t1(c1 INTEGER PRIMARY KEY, c2 CHAR(10));
Create success.
iSQL> DESC t1
[ TABLESPACE : SYS_TBS_MEM_DATA ]
[ ATTRIBUTE ]
------------------------------------------------------------------
NAME                            TYPE                          IS NULL
------------------------------------------------------------------
C1                              INTEGER          FIXED        NOT NULL
C2                              CHAR(10)         FIXED
[ INDEX ]
------------------------------------------------------------------
NAME                            TYPE     IS UNIQUE     COLUMN
------------------------------------------------------------------
__SYS_IDX_ID_122                BTREE    UNIQUE        C1 ASC
[ PRIMARY KEY ]
------------------------------------------------------------------
C1
```

제약조건

- Foreign Key

```
iSQL> CREATE TABLE t2(c1 INTEGER PRIMARY KEY, c3 CHAR(10));
Create success.
iSQL> CREATE TABLE t1 (c1 INTEGER, c2 CHAR(10),
      2 FOREIGN KEY(c1) REFERENCES t2(c1));
Create success.
```

```
iSQL> CREATE TABLE t1 (c1 INTEGER, c2 CHAR(10),
      2 FOREIGN KEY(c1) REFERENCES t2(c1));
[ERR-31011 : TABLE NOT FOUND]
iSQL> CREATE TABLE t2(c1 INTEGER, c3 CHAR(10));
Create success.
iSQL> CREATE TABLE t1 (c1 INTEGER, c2 CHAR(10),
      2 FOREIGN KEY(c1) REFERENCES t2(c1));
[ERR-31049 : Unable to find referenced constraint]
```

참조할 테이블 T2 가 없어서 오류 발생

참조할 테이블 T2 에 기본 키가 없어서 오류 발생

제약조건

- Unique / Default

```
iSQL> CREATE TABLE t1(c1 INTEGER UNIQUE, c2 CHAR(10) DEFAULT 'ALTIBASE');
Create success.
iSQL> DESC t1
[ TABLESPACE : SYS_TBS_MEM_DATA ]
[ ATTRIBUTE ]
------------------------------------------------------------------
NAME                          TYPE                          IS NULL
------------------------------------------------------------------
C1                            INTEGER          FIXED
C2                            CHAR(10)         FIXED
[ INDEX ]
------------------------------------------------------------------
NAME                          TYPE      IS UNIQUE      COLUMN
------------------------------------------------------------------
__SYS_IDX_ID_125              BTREE     UNIQUE         C1 ASC
T1 has no primary key

iSQL> INSERT INTO t1(c1) VALUES(1);
1 row inserted.
iSQL> SELECT * FROM t1;
C1          C2
---------------------------
1           ALTIBASE
1 row selected.
```

제약조건

- NULL / NOT NULL

```
iSQL> CREATE TABLE t1 (c1 INTEGER NULL, c2 CHAR(10) NOT NULL);
Create success.
iSQL> DESC t1
[ TABLESPACE : SYS_TBS_MEM_DATA ]
[ ATTRIBUTE ]
------------------------------------------------------------------
NAME                                TYPE                          IS NULL
------------------------------------------------------------------
C1                                  INTEGER          FIXED
C2                                  CHAR(10)         FIXED        NOT NULL
T1 has no index
T1 has no primary key

iSQL> INSERT INTO t1(c2) VALUES('ALTIBASE');
1 row inserted.
iSQL> SELECT * FROM t1;
C1          C2
---------------------------
            ALTIBASE
1 row selected.
iSQL> INSERT INTO t1(c1) VALUES(1);
[ERR-31074 : Unable to insert(or update) NULL into NOT NULL column.]
```

C2 칼럼에는 NULL이 들어 갈수 없어서 오류 발생

테이블 생성

예제

- column_level 제약조건 생성

```
iSQL> CREATE TABLE emp(
      2 eno INTEGER CONSTRAINT emp_pk PRIMARY KEY,
      3 ename VARCHAR(10) NOT NULL,
      4  age NUMBER DEFAULT 1,
      5 dno INTEGER REFERENCES dept(dno));
Create success.
iSQL> SET FOREIGNKEYS ON
iSQL> DESC emp;
-------------------------------------------------------------------
NAME                    TYPE                          IS NULL
-------------------------------------------------------------------
ENO                     INTEGER          FIXED        NOT NULL
ENAME                   VARCHAR(10)      FIXED        NOT NULL
AGE                     FLOAT            FIXED
DNO                     SMALLINT         FIXED
[ PRIMARY KEY ]
-------------------------------------------------------------------
ENO
[ FOREIGN KEYS ]
-------------------------------------------------------------------
* __SYS_CON_ID_411                        * __SYS_IDX_ID_405
( DNO )                          --->   SYS.DEPT ( DNO )
```

테이블 생성

- table_level 제약조건 생성

```
iSQL> CREATE TABLE emp(
      2 eno INTEGER,
      3 ename VARCHAR(10) NOT NULL,
      4  age NUMBER DEFAULT 1,
      5 dno INTEGER,
      6 CONSTRAINT emp_pk PRIMARY KEY(eno),
      7 CONSTRAINT emp_fk FOREIGN KEY (dno) REFERENCES dept(dno));
Create success.
iSQL> SET FOREIGNKEYS ON
iSQL> DESC emp;
------------------------------------------------------------------
NAME                    TYPE                          IS NULL
------------------------------------------------------------------
ENO                     INTEGER          FIXED        NOT NULL
ENAME                   VARCHAR(10)      FIXED        NOT NULL
AGE                     FLOAT            FIXED
DNO                     SMALLINT         FIXED
[ PRIMARY KEY ]
------------------------------------------------------------------
ENO
[ FOREIGN KEYS ]
------------------------------------------------------------------
* EMP_FK                                  * __SYS_IDX_ID_405
( DNO )                          --->    SYS.DEPT ( DNO )
```

테이블 생성

- 사원번호, 사원이름, 부서번호, 성별, 생일을 칼럼으로 가지는 테이블을 생성

```
iSQL> CREATE TABLE employee(
      2 eno INTEGER ,
      3 ename CHAR(20) ,
      4 dno INTEGER,
      5 sex CHAR(1) ,
      6 birth DATE ) ;
Create success.
```

- 주문번호, 주문일자, 판매사원, 고객주민번호, 상품번호, 주문수량을 칼럼으로 갖는 orders 테이블을 생성. 주문번호와 주문일자를 기본키로 생성

```
iSQL> CREATE TABLE orders(
      2 ono INTEGER,
      3 order_date DATE,
      4 eno INTEGER NOT NULL,
      5 cno CHAR(14) NOT NULL,
      6 gno INTEGER NOT NULL,
      7 qty INTEGER DEFAULT 1,
      8 PRIMARY KEY(ono, order_date));
Create success.
```

테이블 생성

❖ 구문(추가 구문)

```
CREATE TABLE table_name
( column_name datatype [DEFAULT expr] [column_constraint],
……
[table_constraint] )
[MAXROWS integer ]
[TABLESPACE  tablespace_name ]
[AS  subquery ];
```

❖ 예제

- column_level 제약조건 생성

```
iSQL> CREATE TABLE employee(
      2 eno INTEGER PRIMARY KEY,
      3 ename CHAR(20) NOT NULL,
      4 dno INTEGER,
      5 sex CHAR(1) DEFAULT 'M' NOT NULL,
      6 birth DATE )
      7 TABLESPACE edu_mem ;
Create success.
```

테이블 생성

- employee 테이블을 edu_mem 테이블스페이스에 생성하고 최대 row 수는 1000000 으로 지정

```
iSQL> CREATE TABLE employee(
      2 eno INTEGER PRIMARY KEY,
      3 ename CHAR(20) NOT NULL,
      4 dno INTEGER,
      5 sex CHAR(1) DEFAULT 'M' NOT NULL,
      6 birth DATE )
      7 MAXROWS 1000000
      8 TABLESPACE edu_mem;
Create success.
```

- employee 테이블에서 부서 번호가 10 인 조건을 만족하는 데이터를 가진 emp_dept_10 테이블을 생성

```
iSQL> CREATE TABLE  emp_dept_10
      2 AS
      3 SELECT *
      4 FROM employee
      5 WHERE dno = 10 ;
Create success.
```

테이블 생성

❖ 주의 사항

- 칼럼 크기 지정 시 최대 허용 크기를 넘거나 최소 크기 보다 작으면 오류 발생
- PRIMARY KEY는 한 테이블에 2개 이상 존재할 수 없음
- 참조 제약조건의 경우 FOREIGN KEY와 PRIMARY KEY의 칼럼 개수는 동일해야 함
- 참조 제약조건의 경우 FOREIGN KEY와 PRIMARY KEY의 칼럼 데이터 타입은 동일해야 함
- 한 테이블에 PRIMARY KEY 또는 UNIQUE 의 총합이 32개를 넘을 수 없음
- CREATE TABLE AS SELECT의 경우 칼럼 명을 명시하였다면 그 개수는 검색 대상에 명시한 칼럼 개수와 동일해야 함
- CREATE TABLE AS SELECT의 경우 CREATE TABLE 문에 칼럼 명을 명시하지 않고 검색 대상에 표현식을 사용한 경우 반드시 생성할 테이블의 칼럼 명으로 사용하기 위해 별명(alias)이 존재해야 함

테이블 생성

❖ 고려사항

- Tablespace 와 데이터

Memory Tablespace	고성능 데이터
Disk Tablespace	대용량 데이터
Volatile tablespace	logging이 필요 없는 고성능 데이터

- 테이블 생성 시 tablespace 절 지정하여 데이터 특성에 맞게 테이블을 생성
  - Ex) 최신 일주일 동안의 데이터를 자주 access 한다고 하면, 최신 일주일 데이터는 memory table로, 일주일이 지난 history 성 데이터는 disk table로 생성
- 동일 테이블을 디스크, 메모리 테이블스페이스에 나눠서 생성할 수 없음

테이블 변경

구문(추가 구문)

```
ALTER TABLE table_name
{
        ADD [ COLUMN ] ( column_name data_type ) |
        ALTER [ COLUMN ] ( column_name {
                SET DEFAULT | DROP DEFAULT  | NULL | NOT NULL } |
        MODIFY COLUMN  ( column_name data_type ) |
        DROP [ COLUMN ] ( column_name ) |
        ADD table_level_constraint |
        DROP {CONSTRAINT constraint_name | PRIMARY KEY | UNIQUE(column_name)} |
        RENAME COLUMN column_name TO new_column_name |
        MAXROWS |
        ALL INDEX [ENABLE | DISABLE] |
        COMPACT
};
```

예제

- orders 테이블에 주문상태 칼럼을 추가

```
iSQL> ALTER TABLE orders
    2 ADD COLUMN (processing CHAR(1) DEFAULT '0');
Alter success.
```

테이블 변경

- orders 테이블의 주문상태 칼럼의 크기를 CHAR(2)로 변경

```
iSQL> ALTER TABLE orders
      2 MODIFY COLUMN (processing CHAR(2));
Alter success.
```

- orders 테이블의 주문상태 칼럼의 이름을 변경

```
iSQL> ALTER TABLE orders
      2 RENAME COLUMN processing TO process;
Alter success.
```

- orders 테이블의 주문상태 칼럼을 삭제

```
iSQL> ALTER TABLE orders
      2 DROP COLUMN process;
Alter success.
```

테이블 COMPACT

❖ 테이블 COMPACT

- ➢ 데이터가 없는 빈 페이지들에 대하여 페이지를 반환
- ➢ 메모리 / 휘발성 테이블과 큐만 지원

❖ 구문

```
ALTER TABLE table_name
COMPACT ;
```

❖ 예제

- ➢ 데이터가 삭제된 후에 orders 테이블에 할당되어 있는 빈 공간을 반환

```
iSQL> ALTER TABLE orders
      2 COMPACT;
Alter success.
```

테이블 TRUNCATE

테이블 TRUNCATE

- 명시된 테이블에서 모든 데이터를 삭제하고
- 삭제된 데이터는 취소할 수 없음

구문

TRUNCATE TABLE *table_name* **;**

예제

- orders 테이블의 모든 데이터를 삭제

```
iSQL> SELECT COUNT(*)  FROM orders;
COUNT
----------------------
1000
iSQL> TRUNCATE TABLE orders;
Truncate success.
iSQL> SELECT COUNT(*)  FROM orders;
COUNT
----------------------
0
```

테이블 삭제

테이블 DROP

- 테이블을 삭제

구문

```
DROP TABLE table_name
[CASCADE [CONSTRAINTS] ] ;
```

예제

- orders 테이블을 삭제

```
iSQL> DROP TABLE orders;
Drop success.
```

- dept 테이블에 의해 참조되는 emp 테이블을 삭제

```
iSQL> DROP TABLE emp;
[ERR-3102A : A foreign key constraint that depends on the table or column exists.]
iSQL> DROP TABLE emp
    2 CASCADE CONSTRAINTS;
Drop success.
```

테이블 RENAME

테이블 RENAME

- 테이블의 이름 변경

구문

```
RENAME table_name TO new_table_name ;
```

예제

- employee 테이블의 이름을 emp로 변경

```
iSQL> RENAME employee TO emp;
Rename success.
iSQL> SELECT ename
    2 FROM emp
    3 LIMIT 1;
ENAME
------------------------
EJJUNG
```

```
iSQL> ALTER TABLE employee
    2 RENAME TO emp;
Alter success.
```

ALTIBASE

5.4 인덱스

인덱스 개념

INDEX

- 질의문의 성능 향상을 위해 테이블과는 별도로 저장되는 객체
- Index 대상 칼럼 값을 sorting 하여 저장
- Unique, Primary Key로 지정한 칼럼은 내부적으로 Unique Index가 생성
- 메모리 테이블의 인덱스는 메모리에, 디스크 테이블의 인덱스는 디스크에 생성
- 테이블에 대해 물리적, 논리적으로 독립적인 객체이기 때문에 테이블에 관계없이 삭제, 수정이 가능
- 테이블의 레코드가 수정되면 해당 인덱스도 수정이 됨
- 디스크 인덱스는 질의 및 인덱스 저장시 디스크 I/O비용을 줄이기 위해 테이블과 별도의 디스크에 분리해서 저장하는 것을 권장

인덱스 개념

❖ 인덱스 종류

➢ 인덱스 속성

- Unique Index
  - 인덱스 칼럼에 중복을 허용하지 않는 인덱스
- Non-Unique Index
  - 인덱스 칼럼에 중복 값을 허용하는 인덱스
  - UNIQUE 옵션 생략 시 기본적으로 생성이 되는 인덱스
- Composite Index
  - 여러 개의 칼럼 들로 구성된 인덱스
- Single Index
  - 하나의 칼럼으로 구성된 인덱스

➢ 인덱스 저장 위치

- Memory Index
  - Memory Table에 대한 Index
  - 실제 Table의 데이터에 대한 pointer(16bytes) 만 저장
- Disk Index
  - Disk Table에 대한 Index
  - 칼럼의 값과 테이블의 레코드 주소 값이 저장

인덱스 생성

❖ 구문

```
CREATE [UNIQUE ] INDEX index_name
ON table_name ( column_name [ ASC | DESC ], … ) ;
```

❖ 예제

- employee 테이블의 salary 칼럼에 오름차순의 인덱스 생성

```
iSQL> CREATE INDEX emp_idx1
      2 ON employee (salary ASC) ;
Create success.
```

- employee 테이블의 ename 칼럼에 unique 인덱스 생성

```
iSQL> CREATE UNIQUE INDEX emp_idx2
      2 ON employee (ename) ;
Create success.
```

- employee 테이블에 dno, emp_job 칼럼에 composite 인덱스 생성

```
iSQL> CREATE UNIQUE INDEX emp_idx3
      2 ON employee (dno, emp_job) ;
Create success.
```

인덱스 생성

구문 (추가)

```
CREATE [ UNIQUE ] INDEX index_name
ON table_name ( column_name [ ASC | DESC ], … )
[TABLESPACE tablespace_name ]
[NOPARALLEL | PARALLEL parallel_factor ] ;
```

예제

- employee 테이블의 salary 칼럼에 4개의 CPU를 사용하도록 병렬옵션을 지정하여 test_mem 테이블스페이스에 인덱스를 생성

```
iSQL> CREATE INDEX emp_idx1
      2 ON employee (salary ASC)
      3 TABLESPACE test_mem
      4 PARALLEL 4;
Create success.
```

인덱스 변경

인덱스 변경

- 인덱스를 재 구축 하거나 인덱스의 이름을 변경

구문

```
ALTER  INDEX index_name
[REBUILD]
[RENAME TO new_ index_name ];
```

예제

- 인덱스 t1_idx1을 재 구축

```
iSQL> ALTER INDEX t1_idx1
      2 REBUILD ;
Alter success.
```

- 인덱스 t1_idx1의 이름을 t1_idx로 변경

```
iSQL> ALTER INDEX t1_idx1
      2 RENAME TO t1_idx;
Alter success.
```

인덱스 삭제

인덱스 DROP

- 인덱스를 삭제

구문

```
DROP INDEX index_name ;
```

예제

- 인덱스 t1_idx1를 삭제

```
iSQL> DROP INDEX t1_idx1;
Drop success.
```

ALTIBASE

5.5 뷰

뷰 개념

View

- 하나 이상의 테이블에서 데이터의 부분 집합을 논리적으로 표시
- 실제 데이터가 저장되지 않는 가상 테이블

사용 목적

- 데이터 access를 제한하기 위해 사용
- 복잡한 질의를 쉽게 작성하기 위해 사용

ALTIBASE View의 특징

- View를 통해 조회만 가능
  - DML(INSERT, UPDATE, DELETE)이 가능한 Updatable View는 제공하지 않음
- View의 부연질의에 대한 제한 사항
  - 검색 대상 표현식의 개수는 최대 1024개
  - CURRVAL, NEXTVAL 의사열을 사용할 수 없음

뷰 생성

❖ 구문

CREATE [OR REPLACE] [[NO] FORCE] VIEW *view_name*
[(*alias_name*)]
AS
  sub_query
[**WITH READ ONLY**];

❖ 예제

➢ 사원 테이블에서 각 부서의 평균 월급을 부서별로 질의하는 뷰를 생성

```
iSQL> CREATE VIEW avg_sal
      2 AS
      3 SELECT dno, AVG(salary) emp_avg_sal
      4 FROM employee
      5 GROUP BY dno;
Create success.
iSQL> SELECT * FROM avg_sal ;
DNO          EMP_AVG_SAL
-----------------------------------
1001         2150000
1002         1340000
...
```

뷰 삭제

뷰 삭제

- 뷰를 삭제하며 base table은 삭제되지 않음

구문

```
DROP VIEW view_name ;
```

예제

- 뷰 avg_sal 을 삭제

```
iSQL> DROP VIEW avg_sal;
Drop success.
iSQL> SELECT * FROM avg_sal ;
[ERR-31031 : Table not found
iSQL> SELECT ename FROM employee;
ENAME
------------------------
EJJUNG
HJNO
HSCHOI
…
```

ALTIBASE

5.6 시퀀스

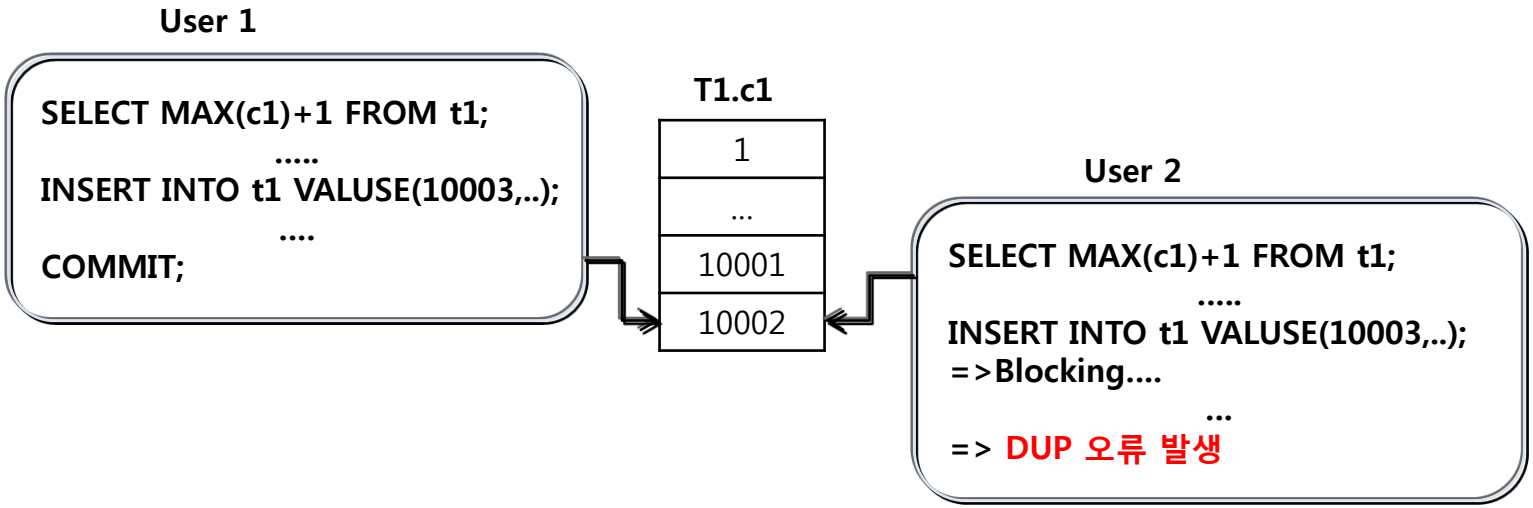
시퀀스 개념

- **SEQUENCE**
  - 연속적인 숫자를 생성하는 객체
  - PRIMARY KEY 칼럼의 유일 값을 생성하는데 주로 사용
  - 트랜잭션과 무관한 객체. ROLLBACK 등으로 인해 값이 복구되지 않음
  - 시퀀스는 테이블과 독립적이므로 테이블에 대한 블로킹 현상이 없음
  - 메모리에 일정 개수를 캐시 해놓기 때문에 성능이 빠름
  - .NEXTVAL 을 먼저 사용한 후에 .CURRVAL 을 사용할 수 있음

시퀀스 개념

❖ 시퀀스를 사용하지 않을 때의 문제점

- ➢ 유일 값을 얻어올 때 테이블 데이터를 조회 (ex. MAX(column)+1) 하는 방법은 블로킹 현상이 발생



시퀀스 생성

구문

CREATE SEQUENCE *sequence_name*
[**START WITH** *start_value*]
[**INCREMENT BY** *increment_value*]
[{**MAXVALUE** *max_value* | **NOMAXVALUE**}] [{**MINVALUE** *min_value* **| NOMINVALUE**}]
[{**CYCLE** | **NOCYCLE**}]
[{**CACHE** *n* | **NOCACHE**}] ;

예제

- 10 부터 시작해서 2씩 증가하고, 최소값이 0, 최대값이 무한대인 시퀀스 seq1을 생성

```
iSQL> CREATE SEQUENCE seq1
      2 START WITH 10
      3 INCREMENT BY 2
      4 MINVALUE 0 NOMAXVALUE;
Create success.
```

시퀀스 생성

- 시퀀스 seq1을 이용하여 t1 테이블에 시퀀스를 적용

```
iSQL> INSERT INTO t1 VALUES(seq1.NEXTVAL);
1 row inserted.
iSQL> INSERT INTO t1 VALUES(seq1.NEXTVAL);
1 row inserted.
iSQL> SELECT * FROM t1;
C1
--------
10
12
2 rows selected.
iSQL> SELECT seq1.CURRVAL FROM dual;
SEQ1.CURRVAL
------------------------
12
2 rows selected.
```

시퀀스 변경

시퀀스 변경

- 시퀀스 생성된 이후이므로 START WITH 값은 변경할 수 없음

구문

```
ALTER SEQUENCE  sequence_name
[INCREMENT BY increment_value ]
[{MAXVALUE max_value | NOMAXVALUE}] [{MINVALUE min_value | NOMINVALUE}]
[{CYCLE | NOCYCLE}]
[{CACHE n | NOCACHE}];
```

예제

- 시퀀스 seq1 의 증가값을 10으로 최대값을 100으로 변경

```
iSQL> ALTER SEQUENCE seq1
      2 INCREMENT BY 10
      3 MAXVALUE 100;
Alter success.
```

시퀀스 삭제

구문

```
DROP SEQUENCE sequence_name ;
```

예제

- 시퀀스 seq1을 삭제

```
iSQL> DROP SEQUENCE seq1;
Drop success.
```

ALTIBASE

5.7 시노님

시노님 개념

- **Synonym**
  - 테이블, 뷰, 시퀀스, 저장 프로시저 및 저장 함수, 시노님에 대한 별칭을 정의하는 기능

- **Synonym을 사용하는 경우**
  - 객체의 원래 이름과 소유한 사용자의 이름을 숨기고 싶은 경우
  - SQL 사용을 단순화하고자 하는 경우
  - 사용자 변경에 따른 응용프로그램의 변경을 최소화하고자 하는 경우

시노님 생성

구문

CREATE [PUBLIC] SYNONYM *synonym_name*
FOR *object_name* ;

예제

- 테이블 employee에 대한 시노님 emp를 생성하여 시노님을 통해 DML을 수행

```
iSQL> CREATE SYNONYM emp FOR employee;
Create success.
iSQL> SELECT ename FROM emp;
ENAME
------------------------
EJJUNG
HJNO
HSCHOI
KSKIM
…
```

시노님 생성

- User1이 소유한 salgrade 테이블에 대해 salgrade 이름의 public 시노님을 생성하여 시노님을 통해 SELECT를 수행

```
iSQL> CONNECT sys/manager
iSQL> CREATE  PUBLIC SYNONYM salgrade FOR user1.salgrade;
Create success.
iSQL> SELECT grade FROM salgrade;
GRADE
-------------------------
1
2
3
4
…
```

시노님 삭제

시노님 삭제

- 시노님을 삭제, 원본 객체는 삭제되지 않음

구문

DROP [PUBLIC] SYNONYM *synonym_name* ;

예제

- 시노님 emp를 삭제

```
iSQL> DROP SYNONYM emp;
Drop success.
```

- public 시노님 salgrade를 삭제

```
iSQL> CONNECT sys/manager
iSQL> DROP PUBLIC SYNONYM salgrade;
Drop success.
```

ALTIBASE

5.8 트리거

트리거 개념

TRIGGER

- 데이터의 변경 사항이 발생할 때, DBMS에서 자동으로 실행되는 객체

Trigger 구성 요소

- Trigger 이벤트
  - Trigger 수행을 발생시키는 SQL문
  - INSERT, UPDATE, DELETE, MOVE
- Trigger 조건
  - Trigger 이벤트가 발생하고 Trigger를 수행시킬 조건
- Trigger 액션
  - Trigger 조건이 TRUE일 때 Trigger가 수행하는 저장 프로시저 내용

트리거 생성

❖ 구문

```
CREATE TRIGGER trigger_name
{BEFORE | AFTER}  trigger_event  ON table_name
[REFERENCING [OLD | NEW] {ROW} {AS} alias_name ]
   trigger_action
```

- Trigger_event ::=

```
{INSERT | DELETE | UPDATE [ OF column_name, … ]}
```

- Trigger_action ::=

```
[{FOR EACH [ROW[WHEN (condition)] | STATEMENT ]}]
psm_body
```

  - psm_body ::=

```
{AS | IS}
   [declare_section]
BEGIN
   [statement]
[EXCEPTION exception_handler]
END;
```

트리거 생성

예제

- t1 테이블 레코드 정보가 삭제될 때 c1, c2 칼럼의 기존 값을 참조하여 t2 테이블에 INSERT하여 삭제되는 행에 대해 추적하는 트리거를 생성

```
iSQL> CREATE TRIGGER del_trigger
      2 AFTER DELETE ON t1
      3 REFERENCING OLD ROW old_row
      4 FOR EACH ROW
      5 AS
      6 BEGIN
      7    INSERT INTO t2 VALUES(old_row.c1, old_row.c2);
      8 END;
/
Create success.
```

트리거 생성

```
iSQL> SELECT * FROM t1;
C1          C2
------------------------
1           1
2           2
3           3
4           4
5           5
5 rows selected.
iSQL> SELECT * FROM t2;
C1          C2
------------------------
No rows selected.
```

```
iSQL> DELETE FROM t1
    2 WHERE c1 > 3;
2 rows deleted.
iSQL> SELECT * FROM t1;
C1          C2
---------------------------
1           1
2           2
3           3
3 rows selected.
iSQL> SELECT * FROM t2;
C1          C2
---------------------------
4           4
5           5
2 rows selected.
```

트리거 생성

❖ 고려 사항

- Trigger Event
  - DML 이벤트를 OR 연산을 이용해서 여러 개 나열할 수 없기 때문에 각각의 DML 이벤트에 대한 Trigger를 별도로 생성해야 함
- DDL Trigger는 지원하지 않음
- Before Update Trigger는 제공하지 않음
- Trigger 대상 테이블에 LOB 칼럼이 있으면 오류 발생

트리거 변경

트리거 변경

- 트리거를 활성화/비활성화 시키거나 재 컴파일을 수행

구문

ALTER TRIGGER *trigger_name*
{**ENABLE** | **DISABLE** | **COMPILE** };

예제

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

트리거 변경

```
iSQL> SELECT * FROM t1;
C1          C2
----------------------------
1           1
2           2
3           3
iSQL> DELETE FROM t1
    2 WHERE c1 < 3;
2 rows deleted.
iSQL> SELECT * FROM t1;
C1          C2
----------------------------
3           3
1 row selected.
iSQL> SELECT * FROM t2;
C1          C2
----------------------------
4           4
5           5
2 rows selected.
```

트리거 삭제

구문

```
DROP TRIGGER trigger_name;
```

예제

- 트리거 del_trigger를 삭제

```
iSQL> DROP TRIGGER del_trigger;
Drop success.
```

ALTIBASE

5.9 큐

큐 개념

QUEUE

- 메시지 큐 기능을 이용하여 데이터베이스와 사용자 프로그램간의 비동기 데이터 통신을 지원하는 객체
- 큐 테이블은 다른 데이터베이스 테이블과 마찬가지로 DDL과 DML로 제어할 수 있음

QUEUE 테이블 구조

Column name	Type	Length	Default	Description
MSGID	BIGINT	8	-	메시지 식별자
CORRID	INTEGER	4	0	사용자가 지정한 메시지 식별자
MESSAGE	VARCHAR	Message length	-	메시지
ENQUEUE_TIME	DATE	8	SYSDATE	메시지 ENQUEUE 시간

큐 생성

❖ 구문

CREATE QUEUE *queue_name(size)*
[**MAXROWS** *count*];

❖ 예제

➢ 메시지의 길이가 최대 40이고, 레코드 개수가 1,000,000인 q1 큐를 생성

```
iSQL> CREATE QUEUE q1(40)
      2 MAXROWS 1000000;
Create success.
iSQL> DESC q1;
--------------------------------------------------------------------
NAME                    TYPE                          IS NULL
--------------------------------------------------------------------
MSGID                   BIGINT          FIXED         NOT NULL
MESSAGE                 VARCHAR(40)     FIXED
CORRID                  INTEGER         FIXED
ENQUEUE_TIME            DATE            FIXED
…
[ PRIMARY KEY ]
--------------------------------------------------------------------
MSGID
```

메시지 삽입

ENQUEUE

- 큐에 메시지를 삽입함
- INSERT구문과 유사한 구조이며 INTO 절 이후에 하나 이상의 큐 칼럼 명을 명시

구문

ENQUEUE INTO *queue_name (column_name,..*)
VALUES ({**DEFAULT** | *value* }) **;**

예제

- ‘This is a message’ 메시지를 correlation id 1로 지정하여 q1에 삽입

```
iSQL> ENQUEUE INTO q1(message, corrid)
      2 VALUES (‘This is a message’, 1);
1 row inserted.
```

메시지 인출

DEQUEUE

- WHERE 절 조건에 맞는 메시지를 얻어오고 해당 메시지를 삭제

구문

```
DEQUEUE queue_column_list
FROM  queue_name
[WHERE condition ]
[FIFO | LIFO] [WAIT value ] ;
```

예제

- 큐 q1에서 correlation id 가 1인 메시지를 인출

```
iSQL> DEQUEUE message, corrid
      2 FROM q1
      3 WHERE corrid=1 ;
MESSAGE                                          CORRID
-------------------------------------------------------
This is a message                                     1
1 row selected.
```

큐 변경

큐 변경

- 데이터가 없는 빈 페이지들에 대하여 페이지를 반환

구문

```
ALTER QUEUE queue_name
COMPACT;
```

예제

- 큐 q1이 사용하고 있던 메모리 공간 중에서 q1의 메시지가 삭제 된 후 반환되지 않은 부분을 반환

```
iSQL> ALTER QUEUE q1
      2 COMPACT;
Alter success.
```

큐 삭제

구문

```
DROP QUEUE queue_name;
```

예제

- 큐 q1을 삭제

```
iSQL> DROP QUEUE q1;
Drop success.
```

ALTIBASE

6. 권한관리

권한 관리

❖ 권한 관리

사용자가 데이터베이스의 객체 및 데이터에 접근하기 위해서는 권한이 필요

❖ 권한의 종류

- ➢ 시스템 접근 권한
  - 모든 스키마 객체들을 관리할 수 있는 권한
  - 데이터베이스에 특정 작업을 수행할 수 있는 권한
  - 주로 DBA가 관리
- ➢ 객체 접근 권한
  - 다른 사용자가 소유한 객체에 대해 접근할 수 있는 권한
  - 객체를 소유한 사용자만이 권한을 부여

권한 관리

❖ 권한 종류

➢ 시스템 접근 권한

시스템 권한	이름
DATABASE	ALTER SYSTEM
INDEXES	CREATE ANY INDEX
	ALTER ANY INDEX
	DROP ANY INDEX
PROCEDURES	CREATE PROCEDURE
	CREATE ANY PROCEDURE
	ALTER ANY PROCEDURE
	DROP ANY PROCEDURE
	EXECUTE ANY PROCEDURE
SEQUENCES	CREATE SEQUENCE
	CREATE ANY SEQUENCE
	ALTER ANY SEQUENCE

권한 관리

❖ 권한 종류

➢ 시스템 접근 권한

시스템 권한	이름
SEQUENCES	DROP ANY SEQUENCE
	SELECT ANY SEQUENCE
SESSIONS	CREATE SESSION
TABLES	CREATE TABLE
	CREATE ANY TABLE
	ALTER ANY TABLE
	DELETE ANY TABLE
	DROP ANY TABLE
	INSERT ANY TABLE
	LOCK ANY TABLE
	SELECT ANY TABLE
	UPDATE ANY TABLE

권한 관리

❖ 권한 종류

➢ 시스템 접근 권한

시스템 권한	이름
USERS	CREATE USER
	ALTER USER
	DROP USER
SESSIONS	CREATE SESSION
VIEWS	CREATE VIEW
	CREATE ANY VIEW
	DROP ANY VIEW
PRIVILEGES	GRANT ANY PRIVILEGES
TRIGGER	CREATE TRIGGER
	CREATE ANY TRIGGER
	ALTER ANY TRIGGER
	DROP ANY TRIGGER

권한 관리

❖ 권한 종류

- ➢ 객체 접근 권한

Object privilege	Table	Sequence	PSM	View
ALTER	O	O		
DELETE	O			
EXECUTE			O	
INDEX	O			
INSERT	O			
REFERENCES	O			
SELECT	O	O		O
UPDATE	O			

GRANT(시스템 접근 권한)

GRANT(시스템 접근권한 부여)

특정 사용자에게 시스템 접근 권한을 부여

구문

```
GRANT {system_privilege,.. | ALL PRIVILEGES}
TO {user_name,.. | PUBLIC};
```

예제

- 사용자 user1에게 시스템 접근 권한을 부여

```
iSQL> CREATE TABLE altitest.t1
      2 (c1 INTEGER PRIMARY KEY, c2 CHAR(14));
[ERR-311B1 : The user should have CREATE_ANY_TABLE privilege(s) to execute
this statement.]
iSQL> CONNECT sys/manager
iSQL> GRANT CREATE ANY TABLE
      2 TO user1;
iSQL> CONNECT user1/user1
iSQL> CREATE TABLE altitest.t1
      2 (c1 INTEGER PRIMARY KEY, c2 CHAR(14));
Create success.
```

GRANT(객체 접근 권한)

GRANT(객체 접근권한 부여)

특정 사용자에게 자신이 소유한 객체에 대한 접근 권한을 부여

구문

```
GRANT {object_privilege, ... | ALL PRIVILEGES}
ON object_name
TO {user_name, ... | PUBLIC};
```

예제

- user1이 user2에게 객체 접근 권한을 부여

```
iSQL> CONNECT user2/user2
iSQL> INSERT INTO user1.usr1_tbl
    2 VALUES(1, TO_CHAR(SYSDATE, 'YYYYMMDDHHMISS'));
[ERR-311B1 : The user should have INSERT_ANY_TABLE privilege(s) to
execute this statement.]
iSQL> CONNECT user1/user1
iSQL> GRANT INSERT
    2 ON usr1_tbl
    3 TO user2;
iSQL> CONNECT user2/user2
iSQL> INSERT INTO user1.usr1_tbl
    2 VALUES(1, TO_CHAR(SYSDATE, 'YYYYMMDDHHMISS'));
1 row inserted.
```

REVOKE(시스템 접근 권한)

REVOKE(시스템 권한 회수)

특정 사용자 부여된 시스템 접근 권한을 회수

구문

```
REVOKE {system_privilege,... | ALL PRIVILEGES}
FROM {user_name, ... | PUBLIC};
```

예제

- 사용자 user1으로부터 시스템 접근 권한을 회수

```
iSQL> CONNECT sys/manager
iSQL> REVOKE CREATE ANY TABLE
    2 FROM user1;
iSQL> CONNECT user1/user1
iSQL> CREATE TABLE altitest.t1
    2 (c1 INTEGER PRIMARY KEY, c2 CHAR(14));
[ERR-311B1 : The user should have CREATE_ANY_TABLE privilege(s) to
execute this statement.]
```

REVOKE(객체 접근 권한)

REVOKE(객체 권한 회수)

특정 사용자 에게 부여된 객체 접근 권한을 회수

구문

```
REVOKE {object_privilege,... | ALL PRIVILEGES}
ON object_name
FROM {user_name, ... | PUBLIC};
```

예제

- 사용자 user2로부터 객체 접근 권한을 회수

```
iSQL> CONNECT user1/user1
iSQL> REVOKE INSERT
      2 ON usr1_tbl
      3 FROM user2;
iSQL> CONNECT user2/user2
iSQL> INSERT INTO user1.usr1_tbl
      2 VALUES (1, TO_CHAR(SYSDATE, 'YYYYMMDDHHMISS'));
[ERR-311B1 : The user should have INSERT_ANY_TABLE privilege(s) to
execute this statement.]
```

WITH GRANT OPTION

❖ WITH GRANT OPTION

자신이 부여 받은 객체 권한을 다른 사용자에게 권한을 재부여할 수 있게 하며 REVOKE 시에 연쇄적으로 회수

❖ 구문(옵션추가)

GRANT {*object_privilege, ...* | **ALL PRIVILEGES**}
ON *object_name*
TO {*user_name, ...* | **PUBLIC**}
WITH GRANT OPTION;

WITH GRANT OPTION

예제

- User1 사용자가 user2에게 객체 접근 권한을 WITH GRANT OPTION 옵션과 함께 부여하고, user2가 user1의 객체 접근 권한을 user3에게 부여

```
iSQL> CONNECT user1/user1;
iSQL> GRANT SELECT, DELETE
      2 ON emp
      3 TO user2
      4 WITH GRANT OPTION;
Grant success.
iSQL> CONNECT user2/user2;
Connect success.
iSQL> GRANT SELECT, DELETE
      2 ON user1.emp
      3 TO user3;
Grant success.
iSQL> CONNECT user3/user3;
iSQL> DELETE FROM user1.emp
      2 WHERE eno = 12;
1 row deleted.
iSQL> SELECT COUNT(*) FROM user1.emp;
COUNT
---------------------------------------
20
```