

# Contents

Oracle 9i  (SQL, PL/SQL)

## Chapter 01 Introduction

---

Database System	»» 1-01
Database Schema	»» 1-04
Relational Data Model	»» 1-05
Integrity Constraints(IC)	»» 1-07
Relational Query Languages	»» 1-09
SQL	»» 1-10
SQL*Plus Login	»» 1-11
Describe Tables	»» 1-12
Demo Tables	»» 1-13

## Chapter 02 Basic Query

---

Basic Select Syntax	»» 2-01
Basic SELECT Example	»» 2-03
Arithmetic Expressions	»» 2-04
Null Value	»» 2-06
NVL Function	»» 2-08
Column Alias	»» 2-10
Concatenation Operator	»» 2-12
Literals	»» 2-14
Duplicate Values	»» 2-16
CASE	»» 2-18
WHERE Clause	»» 2-19
Comparison Operators	»» 2-21
Logical Operators	»» 2-27
ORDER BY Clause	»» 2-30
SQL*Plus Commands	»» 2-32
SQL Buffer Commands	»» 2-33
SQL*Plus File Commands	»» 2-35
For Lab	»» 2-36

---

## *Chapter 03* **Single—Row SQL Functions**

---

Functions	»» 3–01
SQL Functions	»» 3–02
Single—Row Functions	»» 3–04
Main Data Types	»» 3–05
Number Functions	»» 3–06
Character Functions	»» 3–07
Date Functions	»» 3–10
Date Arithmetic Operations	»» 3–11
Conversion Functions	»» 3–12
Format Model	»» 3–13
Format Elements for Number	»» 3–14
Format Elements for Date	»» 3–15
RR Format	»» 3–18
Miscellaneous Functions	»» 3–19
Nesting Single—Row Functions	»» 3–21

## *Chapter 04* **Advanced Query(Join)**

---

Cartesian Product	»» 4–01
Table Join	»» 4–02
Table Join Types	»» 4–03
Table Alias	»» 4–04
Equijoin	»» 4–06
Non—equijoin	»» 4–08
Outer Join	»» 4–10
Self Join	»» 4–12

## *Chapter 05* **Advanced Query(Aggregation)**

---

Aggregate Function	»» 5–01
GROUP BY Clause	»» 5–04
Incorrect Example for GROUP BY	»» 5–06
HAVING Clause	»» 5–07

ROLLUP,CUBE Operators	»» 5-09
ROLLUP Operator	»» 5-10
CUBE Operator	»» 5-11
GROUPING Function	»» 5-12
GROUPING SETS	»» 5-14

## *Chapter 06* **Advanced Query(Subquery, Set Operation)**

---

Subquery	»» 6-01
Nested Subquery	»» 6-02
Single-Row Subquery	»» 6-03
Multiple-Row Subquery	»» 6-04
Subquery in the FROM Clause	»» 6-05
Subquery in the HAVING Clause	»» 6-06
Correlated Subquery	»» 6-07
Set Operator	»» 6-08
Hierarchical Query	»» 6-11

## *Chapter 07* **SQL\*Plus Commands**

---

SQL and SQL*Plus	»» 7-01
SPOOL Command	»» 7-02
Set Command	»» 7-03
System Variables	»» 7-04
COLUMN Command	»» 7-08
BREAK Command	»» 7-11
Substitution Variables	»» 7-12
Define User Variables	»» 7-15
Declare Bind Variables	»» 7-16

## *Chapter 08* **Create Table**

---

Schema Objects	»» 8-01
Create Table	»» 8-02

Naming Tables and Columns	»» 8-05
Column Data Types	»» 8-06
Data Integrity	»» 8-07
Integrity Constraints	»» 8-08
Oracle ICs	»» 8-09
Define ICs	»» 8-10
NOT NULL Constraint	»» 8-12
UNIQUE Constraint	»» 8-13
PRIMARY KEY Constraint	»» 8-14
FOREIGN KEY Constraint	»» 8-15
CHECK Constraint	»» 8-19
S_DEPT Table Creation	»» 8-20
S_EMP Table Creation	»» 8-21
Create Tables by Subquery	»» 8-22
Data Dictionary	»» 8-24
Contents of Data Dictionary	»» 8-25
How to Use the Data Dictionary	»» 8-26
Data Dictionary Views	»» 8-27
Query Data Dictionary	»» 8-28
Script Using Data Dictionary	»» 8-30

## *Chapter 09* **Data Manipulation**

---

Data Manipulation Language	»» 9-01
Insert	»» 9-02
Update	»» 9-08
Merge	»» 9-11
Delete	»» 9-13
Returning Clause	»» 9-16
CHAR Data Types	»» 9-17
VARCHAR2 Data Types	»» 9-18
Date Time Data Types	»» 9-19
Date Time Functions	»» 9-20

## Chapter 10 Transaction Control

Transaction Control Statements	»»10-01
Transaction	»»10-02
Transaction Control	»»10-04
Transaction Control Statements	»»10-05
State of the Data During Transaction	»»10-07
State of Data After Transaction	»»10-08
Set Transaction	»»10-09

## Chapter 11 Data Definition

Some of DDL Statements	»»11-01
Alter Tables	»»11-02
Alter Tables for Columns	»»11-03
Add Columns	»»11-04
Modify Columns	»»11-05
Drop Columns	»»11-06
Unused Columns	»»11-07
Alter Tables for Constraints	»»11-08
Add Constraints	»»11-09
Drop Constraints	»»11-10
Enable Constraints	»»11-11
Disable Constraints	»»11-12
Drop Table	»»11-13
Rename	»»11-15
Truncate	»»11-17
Comment	»»11-19

## Chapter 12 Other DB Objects

Oracle Main DB Objects	»»12-01
View	»»12-02
Usages of View	»»12-03
CREATE VIEW	»»12-04

Types of View	»»12-07
WITH READ ONLY	»»12-10
WITH CHECK OPTION	»»12-11
DROP VIEW	»»12-12
Dictionary for Views	»»12-13
INDEX	»»12-14
Types of Index	»»12-15
CREATE INDEX	»»12-16
DROP INDEX	»»12-17
Dictionary for Indexes	»»12-18
Needs for the Index	»»12-19
Sequence	»»12-20
CREATE SEQUENCE	»»12-21
Using Sequences	»»12-22
ALTER SEQUENCE	»»12-23
DROP SEQUENCE	»»12-24
Dictionary for Sequences	»»12-25
Synonym	»»12-26
CREATE/DROP SYNONYM	»»12-27
Dictionary for Synonyms	»»12-28

## *Chapter 13* **Data Control**

---

Database User	»»13-01
Manage User	»»13-02
Dictionary for Users	»»13-03
Privileges	»»13-04
System Privileges	»»13-05
Control System Privileges	»»13-06
WITH ADMIN OPTION	»»13-09
Object Privileges	»»13-10
WITH GRANT OPTION	»»13-12
Role	»»13-13
Dictionary for Privileges	»»13-16

*Chapter 14* **PL/SQL Program(Introduction)**

PL/SQL	»»14-01
Advantages of PL/SQL	»»14-02
PL/SQL Block Structure	»»14-03
PL/SQL Execution Structure	»»14-04
PL/SQL Category	»»14-05
SQL*Plus Commands	»»14-06
PL/SQL Block Debugging	»»14-08
Anonymous Block	»»14-09
Stored Program	»»14-10
Stored Procedure	»»14-11
Stored Function	»»14-14
Usages of Stored Function	»»14-17
Dictionary for Stored Program	»»14-18

*Chapter 15* **PL/SQL Program(Variables, SQL, PL)**

Declare Variables	»»15-01
Scalar Data Type	»»15-02
Composite Data Type	»»15-04
TABLE Type Variables	»»15-05
RECORD Type Variables	»»15-06
%TYPE Attribute	»»15-07
%ROWTYPE Attribute	»»15-08
Assign Values to Variables	»»15-09
Scope Rules	»»15-10
Operators in PL/SQL	»»15-11
Functions in PL/SQL	»»15-12
SQL in PL/SQL	»»15-13
SELECT in PL/SQL	»»15-14
Exceptions from SELECT	»»15-16
DML in PL/SQL	»»15-18
Transaction Control in PL/SQL	»»15-20
IF Statement	»»15-21

Basic Loop	»»15-24
WHILE Loop	»»15-26
FOR Loop	»»15-28
Nested Loop	»»15-30
Nested Block	»»15-31
Comments	»»15-32

## Chapter 16 PL/SQL Program(Cursor, Exception)

Cursor	»»16-01
Explicit Cursor	»»16-02
Cursor Operation	»»16-03
Cursor For Loop	»»16-05
Where Current Of	»»16-06
Parameterized Cursor	»»16-07
Exception	»»16-08
Exception Types	»»16-09
Exception Handlers	»»16-10
Predefined Exceptions	»»16-11
Non-Predefined Exceptions	»»16-13
User-Defined Exceptions	»»16-15
Error Functions	»»16-17

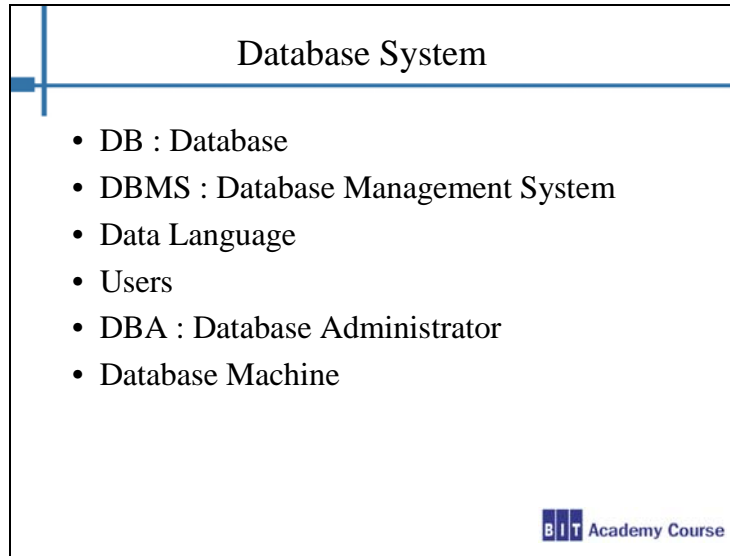
## 부 록

Oracle Server 설치	»»부록-1
분산 DB 실습	»»부록-2



# Introduction

- Database System의 개념과 구성 요소
- Relational Data Model의 용어
- 실습용 Table 소개

**Database의 정의**

어느 한 조직의 여러 응용 시스템들이 공용할 수 있도록 통합, 저장된 운영 data의 집합(이석호 교수)

**Database의 특징**

- 실시간 접근성(Real-time accessibility) : 사용자의 요구에 대한 즉각적인 응답(Response)
- 계속적인 변화(Continuous evolution) : 삽입, 삭제, 갱신 작업이 수시로 발생
- 동시 공유(Concurrent sharing) : 여러 사용자가 동시에 자기가 원하는 data에 접근 가능
- 내용에 의한 참조(Content reference) : 물리적 주소가 아닌 data에 대한 참조

**DBMS의 정의**

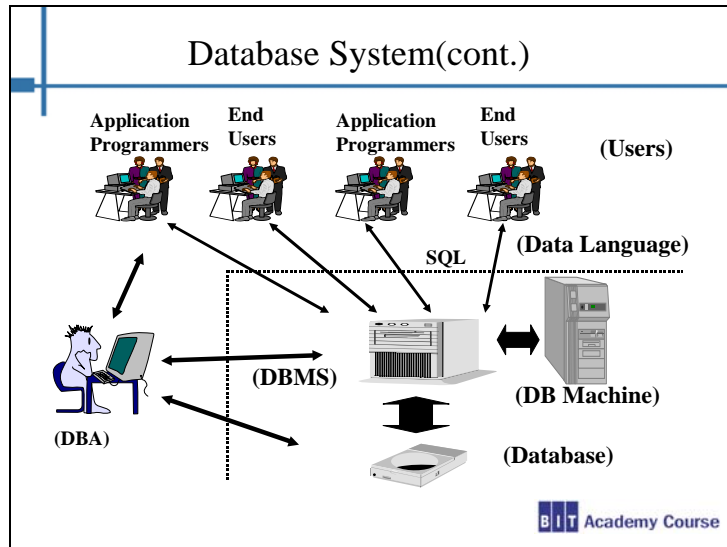
사용자와 Database 사이에 위치하여 사용자의 요구에 따라 Database를 조작하고 제어하는 기능을 제공하는 소프트웨어

**DBMS의 장점**

- data의 독립성 및 중복 최소화
- 응용 프로그램의 개발 시간 단축
- data의 무결성과 보안 보장
- 표준화되고 일관된 data 관리 가능
- data 동시 사용가능
- data 회복 가능

**DBMS의 단점**

- 시스템 자원 요구로 운영비 증대
- 고급 프로그래밍 필요로 자료 처리의 복잡화
- 장애 발생 대비를 위한 복잡한 Back up과 Recovery 작업 필요



### DBMS의 기능

- 정의 기능 : Database의 논리적, 물리적 구조를 정의할 수 있는 기능
- 조작 기능 : 사용자가 Database 내의 data를 조작할 수 있도록 하기 위한 기능
- 제어 기능 : Database가 항상 정확하고 올바른 data를 유지하도록 하기 위한 기능

### Data Language

Database를 정의,조작,제어하기 위하여 사용자와 Database 시스템 간에 사용하는 통신 수단으로 SQL이 하나의 예이다.

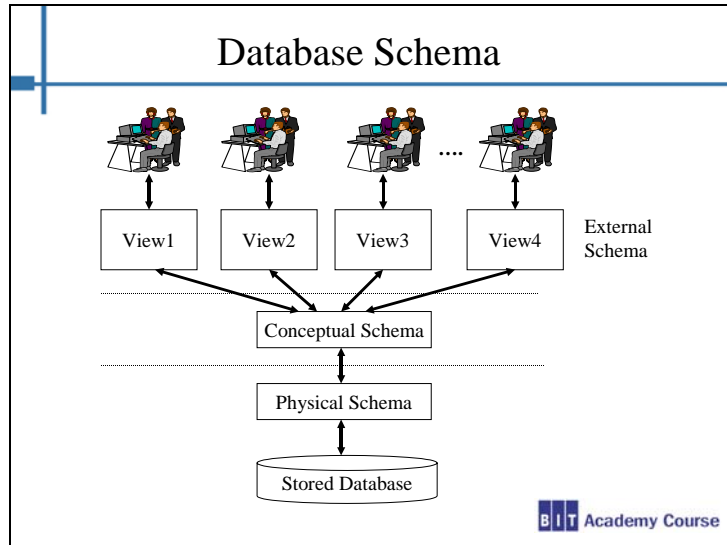
- data 정의어(DDL)
- data 조작어(DML)
- data 제어어(DCL)

### DBA의 역할

- Database 설계와 운영 : Database 구성 요소 결정, schema 정의, 저장 구조와 접근 방법 설정, 보안 및 권한 부여 정책 결정, 백업, 회복 절차 수립 등의 작업 수행
- 행정 및 불평 해결 : 사용자의 요구를 받아 분석하고 불만을 해소
- 시스템 감시 및 성능 분석 : 시스템 이용도, 병목 현상, 이용 패턴, data 사용 추세, 각종 통계 등의 분석 작업 수행

### DB Machine

- Database 시스템의 성능을 향상시키기 위해 사용하는 후위 컴퓨터
- 대용량의 data에 대한 빠른 처리를 위해 사용됨



## 스키마(Schema)

Database의 논리적 정의

### 3단계 schema

- External schema
  - 각 사용자의 입장에서 본 Database의 구조
  - 사용자마다 서로 다른 Database schema를 가짐
  - 개념 schema에 대한 서브 schema
- Conceptual schema
  - 조직 전체의 입장에서 본 Database 구조
  - 한 개의 schema만 존재하며, 서로 다른 사용자가 공유
  - data 객체(개체, 관계), 제약조건에 대한 명세를 유지
- Physical schema
  - 저장 장치의 입장에서 본 Database 구조
  - 각 data 객체의 저장 구조를 표현함
  - 내부 레코드의 형식
  - 인덱스의 유무
  - 저장 data 항목의 표현 방법

## Relational Data Model

- A data model is a collection of concepts for describing data.
- A schema is a description of a particular collection of data, using the a given data model.
- The relational model of data is the most widely used model today.
  - Main concept: relation, basically a table with rows and columns.
  - Every relation has a schema, which describes the columns, or fields.

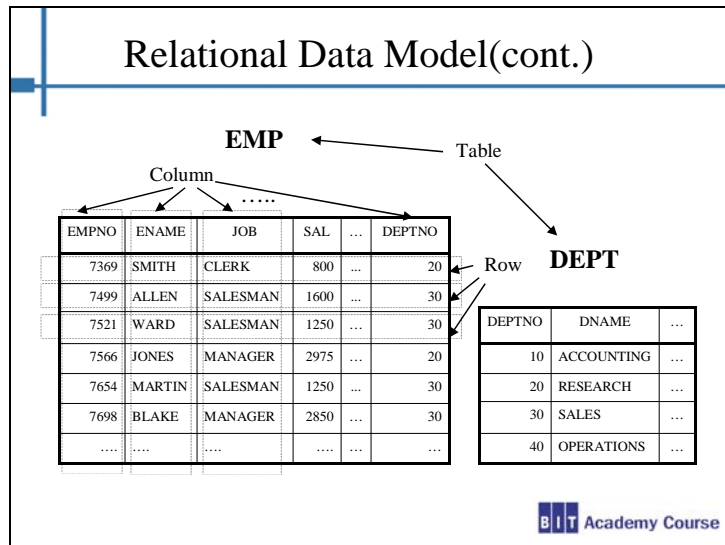
BIT Academy Course

### Database의 유형

- Hierarchical Database
- Network Database
- Relational Database
- Object-Oriented Database

### RDBMS(관계형 Database 관리 시스템)

- E.F.Codd 박사가 제안한 관계형 모델은 Oracle을 포함한 여러 Relational Database 관리 시스템의 기반이 되었다.



#### Relational Database의 정의

Relation 또는 2차원 Table을 사용하여 정보 저장

- Relation = Table
- Attribute = Column
- Tuple = Row

## Integrity Constraints(IC)

- IC : condition that must be true for any instance of the database.
  - ICs are specified when schema is defined.
  - ICs are checked when relations are modified.
- A legal instance of a relation is one that satisfies all specified ICs.
  - DBMS should not allow illegal instances.
- If the DBMS checks ICs, stored data is more faithful to real-world meaning.
  - Avoids data entry errors, too.

BIT Academy Course

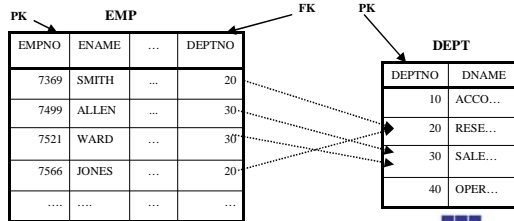
무결성 제약조건(Integrity Constraints)이란 기업 내의 실제 업무 규칙(Business Rule)을 표현한 것이다.

Primary Key(PK)와 Foreign Key(FK)는 가장 일반적인 무결성 제약 조건이다.



## Integrity Constraints(IC)(cont.)

- Entity Integrity : Primary key must be...
  - Unique
  - Not null
- Referential Integrity : Foreign key must be one of the referenced values.



BIT Academy Course

### 개체 무결성(Entity Integrity)

- table은 중복된 row를 가질 수 없으며 모든 table은 각각의 row를 유일하게 식별할 수 있는 column의 집합을 가진다. 이러한 column의 집합 중에서 주요한 것을 Primary Key(PK)로 정의한다.
- Primary Key의 값은 항상 유일(Unique)하며 널(Null)을 허용해서는 안된다.
- DEPT table의 DEPTNO column과 EMP table의 EMPNO column이 Primary Key이다.

### 참조 무결성(Referential Integrity)

- table들은 Foreign Key(FK)를 통하여 서로 연결되어 있다. Foreign Key는 다른 table 또는 자신 table의 Primary Key 값을 참조하기 위해 복사하여 가지고 있는 column을 말한다.
- 참조 무결성이 지켜지기 위해서는 Foreign Key column의 값은 참조하는 Primary Key column 값 중의 하나이거나 Null 이어야 한다.
- EMP table의 DEPTNO column이 Foreign Key이다.

무결성 제약은 DBMS 시스템이 자동으로 수행한다.

## Relational Query Languages

- A major strength of the relational model: supports simple, powerful querying of data.
- Queries can be written intuitively, and the DBMS is responsible for efficient evaluation.
- SQL Query Language.

BIT Academy Course

### SQL(Structured Query Language)

- 비 절차식 언어
- 1970년 대에 IBM의 SYSTEM R 프로젝트를 통해 개발
- 표준화 작업
  - SQL-86
  - SQL-89(minor revision)
  - SQL-92(major revision, current standard)
  - SQL-99(major extensions)

## SQL

- Data Manipulation Language
  - SELECT, INSERT, UPDATE, DELETE, MERGE
- Data Definition Language
  - CREATE, ALTER, DROP, RENAME, TRUNCATE
- Data Control Language
  - GRANT, REVOKE
- Transaction Control Language
  - COMMIT, ROLLBACK, SAVEPOINT

BIT Academy Course

### SQL

- Database server와 통신하기 위한 명령언어
- RDBMS를 사용하기 위해 ANSI에서 책정한 표준언어
- DBMS 제품별로 SQL에 대한 추가 및 확장

### PL/SQL

- 응용프로그램의 logic을 추가하여 SQL을 확장한 Oracle의 절차적인 언어
- 3세대 언어로 IF 문장이나 LOOP 문장을 통해 프로그램의 흐름을 제어할 수 있으며 SQL 문장을 사용하여 data 조작 가능
- Client가 PL/SQL 블록 실행을 요청하면 Oracle server는 Block 전체를 실행한 후 결과를 return하므로 한 번의 Oracle server 호출로 다량의 SQL과 logic을 구사할 수 있다는 장점이 있다.

### SQL\*Plus

SQL 및 PL/SQL 문장을 인식하고 실행시켜 주는 Oracle의 Tool로서 SQL, PL/SQL 등을 직접 입력하여 Oracle server로 보내 실행하게 한 후 결과를 받아본다.

## SQL\*Plus Login

- Windows Environment
  - Double click SQL\*Plus icon.
  - Enter user name and password.
- Command Line
  - sqlplus [user/[passwd]]
- Logout from SQL\*Plus
  - Close SQL\*Plus window.
  - SQL> EXIT;

BIT Academy Course

아래 위치에서 SQL\*Plus를 실행한다.

시작 -> 프로그램 -> Oracle-OraHome90 -> Application Development -> SQL Plus

아래의 계정으로 Database에 접속한다.

사용자 이름	scott
암호	tiger
호스트 스트링	

사용자 이름	scott/tiger
암호	
호스트 스트링	

접속이 성공하면 SQL\*Plus 프롬프트가 나타난다.

```
Connected to:
Enterprise Oracle9i Release 9.0.1.1.1 - Productio
With the Partitioning option
JServer Release 9.0.1.1.1 - Production
SQL>
```

접속을 끊고 SQL\*Plus를 종료하기 위해서는 EXIT 명령을 실행한다.

```
SQL> EXIT;
```

Describe Tables

- SQL\*Plus command
  - DESC[RIBE]
- SQL> DESC *name*
  - *name* : Table, View or Synonym name

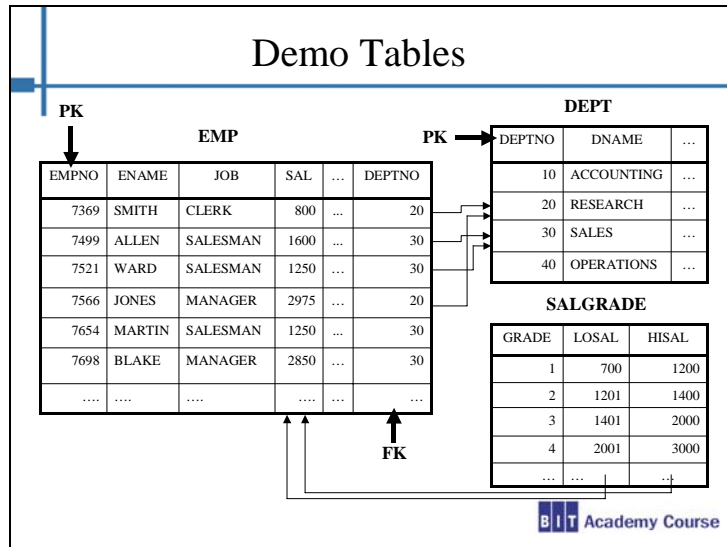
BIT Academy Course

SQL\*Plus 명령어는 맨 뒤에 세미콜론(;)을 붙이지 않아도 실행된다. (SQL 문장은 끝에 세미콜론을 붙여준다.)

DESC 명령의 결과는 column명, data type, column의 NULL 허용여부에 대한 정보를 보여준다.

```
SQL> DESC dept
```

이름	널?	유형
-----		
DEPTNO	NOT NULL	NUMBER(2)
DNAME		VARCHAR2(14)
LOC		VARCHAR2(13)



### 실습에 사용될 table

- DEPT : 부서 정보를 저장. Primary Key는 DEPTNO column.
- EMP : 사원 정보를 저장. Primary Key는 EMPNO column. DEPTNO column은 DEPT table의 Primary Key인 DEPTNO column을 참조하는 Foreign Key.
- SALGRADE : 급여 등급 정보를 저장. 급여의 범위에 따라 1,2,3... 등급을 정하고 있으며 이 table에는 Primary Key, Foreign Key가 없다.
- 특정 사원의 부서명이나 부서의 위치 정보를 알기 위해서는 Foreign Key인 DEPTNO column 값에 해당하는 DEPT table 정보를 검색해야 한다.
- 특정 사원의 급여가 어떤 등급인지 확인하려면 SAL column 값이 SALGRADE table의 LOSAL과 HISAL column 값 사이에 있는 Grade를 찾아야 한다.

SQL> DESC dept

SQL> DESC emp

SQL> DESC salgrade

# Basic Query

- SELECT문장의 기본적인 활용
- WHERE절을 이용하여 row를 제한
- ORDER BY 절을 이용한 data 정렬
- SQL\*Plus의 Buffer와 file관련 명령어

## Basic Select Syntax

```
SELECT * | {[ALL | DISTINCT] column / expr [alias],...}
FROM table
```

- SELECT clause and FROM clause are mandatory.
- SELECT clause : specifies the columns to be displayed.
- FROM clause : specifies the table contains columns written on the SELECT clause.
- SQL statements are not case sensitive.

BIT Academy Course

SELECT 문장은 Table에서 data를 검색하기 위한 문장으로 응용 프로그램 작성 시 가장 많이 사용하는 문장이다.

최소한 SELECT절과 FROM절이 있어야 SELECT 문장이 가능하다.

SQL\*Plus에서 SQL 문장 작성 시 유의 사항

- 모든 SQL 문장은 세미콜론(;)으로 끝낸다.
- SQL 문장은 한 줄로 입력하거나 여러 줄로 보기 좋게 나누어 입력한다.
- SQL 문장은 대소문자를 가리지 않는다. (Case insensitive)
- data 값은 대소문자를 가린다. (Case sensitive)



## Basic Select Syntax(cont.)

- *\** : selects all columns from table listed in the FROM clause.
- *ALL* : returns all rows selected. The default is *ALL*.
- *DISTINCT* : returns only one copy of each set of duplicate rows selected.
- *column* : selects specified column from table listed in the FROM clause.
- *expr* : selects an expression. An expression is a combination of one or more values, operators, and SQL functions that evaluate to a value.
- *alias* : provides a different name for the column expression and causes the alias to be used in the column heading.

 Academy Course

### Default Column Heading

column 명이 대문자로 display된다.

### Default Data justification

Number 값은 right-justified, Character와 Date 값은 left-justified 된다.

## Basic SELECT Example

- Display all employees information.

```
SQL> SELECT *
      2 FROM emp;
```

BIT Academy Course

SCOTT/TIGER로 접속하여 다음을 실습해 보시오.

```
SQL> SELECT ename FROM emp;
```

```
SQL> SET pagesize 1000
```

( \*1000 줄을 한 페이지로 설정하는 SQL\*Plus 명령 실행)

```
SQL> /
```

( \* SQL\*Plus의 buffer에 들어있는 SQL 문장을 다시 실행)

```
SQL> select ename from emp;
```

```
SQL> Select Ename From Emp;
```

```
SQL> SELECT ename
      FROM emp;
```

```
SQL> SELECT empno, ename FROM emp;
```

```
SQL> SELECT ename, empno FROM emp;
```

```
SQL> SELECT ename, ename FROM emp;
```

```
SQL> SELECT * FROM emp;
```

Arithmetic Expressions		
	Operator	Purpose
1	+ -	Denotes a positive or negative expression. These are unary operators.
2	* /	Multiplies, divides. These are binary operators.
3	+ -	Adds, subtracts. These are binary operators.

BIT Academy Course

연산자의 우선순위는 위의 표에서 1 → 2 → 3 순서이다.

우선순위가 높은 연산 먼저 수행하며, 같은 우선순위의 연산자들은 왼쪽에서 오른쪽으로 순서대로 계산해 나간다.

괄호를 사용하여 우선순위를 조정할 수 있다.

## Arithmetic Expressions(cont.)

- Display all employees name and annual salary.

```
SQL> SELECT ename, sal + 100
2 FROM emp;
```

BIT Academy Course

산술연산자의 우선순위를 확인하기 위한 문장들이다. 다음을 실행해 보시오.

```
SQL> SELECT sal, -sal FROM emp;
```

```
SQL> SELECT sal, sal*1.1 FROM emp;
```

```
SQL> SELECT sal, comm, sal + comm FROM emp;
```

```
SQL> SELECT sal, -sal + 100 * -2 FROM emp;
```

```
SQL> SELECT sal, (-sal + 100) * -2 FROM emp;
```

모든 사원의 연봉을 출력하기 위한 문장들이다. 다음을 실행해 보시오.

```
SQL> SELECT empno, ename, sal, sal*12 FROM emp;
```

```
SQL> SELECT empno, ename, sal, sal*12 + comm FROM emp;
```

```
SQL> SELECT empno, ename, sal, sal + comm * 12 FROM emp;
```

```
SQL> SELECT empno, ename, sal, (sal + comm) * 12 FROM emp;
```

## Null Value

- Use a null when the actual value is not known or when a value would not be meaningful.
- Do not use null to represent a value of zero or space, because they are not equivalent.
- Any arithmetic expression containing a null always evaluates to null.
- Nulls can appear in columns of any data type that are not restricted by NOT NULL or PRIMARY KEY integrity constraints.

BIT Academy Course

Null이란 아직 값을 알 수 없는 상태 또는 의미가 없는 상태를 표현한다.

Null은 0이나 space 등과 는 다르다.

산술 연산 수식에 Null 인 값이 하나라도 포함되어 있다면 결과값은 항상 Null이 된다.

NOT NULL이나 PRIMARY KEY 제약 조건을 갖는 column은 Null일 수 없다.

Oracle Database에서 Null인 column은 Length가 0으로 data를 위한 물리적 공간을 차지하지 않는다.

## Null Value(cont.)

- Display all employees salary, commission, and annual salary.

```
SQL> SELECT sal, comm, (sal+comm)*12
2 FROM emp;
```

BIT Academy Course

다음 문장을 실행해 보시오. Column이 Null인 경우 어떻게 표현되는가? Comm 값이 있는 사원은 어떤 사원들인가? 사번 7844인 사원의 커미션은 얼마인가?

```
SQL> SELECT empno, job, comm FROM emp;
```

(\* Null인 값은 비어있는 것으로 표현된다. JOB이 SALESMAN인 사원들에게만 커미션이 적용되며, 사번 7844인 사원의 커미션은 0이다.)

다음은 사원들의 연봉을 계산하는 문장이다. Comm 값이 Null인 경우 연봉은 얼마인가? 연봉 계산한 수식의 column heading은 어떻게 나타나는가?


```
SQL> SELECT sal, comm, (sal+comm)*12 FROM emp;
```

(\* comm 값이 Null인 row의 경우 (sal+comm)\*12를 한 결과도 모두 Null이 된다. 또한, expression 전체가 column heading으로 나타난다.)

## NVL Function

NVL (expr1, expr2)

- If expr1 is null, returns expr2.
- If expr1 is not null, returns expr1.
- The data type of the return value is always the same as the data type of expr1.

 BIT Academy Course

data의 Null 값을 처리하는 function에 NVL, NVL2, NULLIF, COALESC 등이 있다. (3-20참조)

## NVL Function(cont.)

- Example
  - NVL(hiredate, '02/12/31')
  - NVL(job, 'No Job')
  - NVL(sal, 1000)

```
SQL> SELECT sal, comm, (sal+NVL(comm,0))*12
2 FROM emp;
```

BIT Academy Course

다음은 사번과 커미션을 출력하면서 커미션이 없는 사원의 경우 Null이 아니라 0으로 출력하도록 하는 문장이다. 실행해 보시오.

```
SQL> SELECT empno, ename, NVL(comm, 0) comm FROM emp;
```

다음은 매니저가 없는, 즉 최고 직급의 사원인 경우 'No Manager'라고 출력되도록 하는 문장이다. 실행하여 Error 메시지를 적어보고 Error가 나는 이유를 설명하시오.

```
SQL> SELECT NVL(mgr, 'No Manager') FROM emp;
(* Error 발생)
```



## Column Alias

- use a column alias to label the preceding expression in the select list so that the column is displayed with a new heading.
- The alias effectively renames the select list item for the duration of the query.
- The alias can be used in the ORDER BY clause, but not other clauses in the query.
- The AS keyword is optional.

BIT Academy Course

alias는 SELECT 절에 expression을 사용할 때 유용하다.

column명이나 expression 바로 뒤에 명시하거나 column명과 alias사이에 AS를 끼워 넣어 사용한다.

Double Quotation(" ")을 사용하여 alias내에 공백이나 특수문자를 포함할 수 있다.

## Column Alias(cont.)

- Example

```
SQL> SELECT (sal+NVL(comm,0))*12 AS ANNUAL
2 FROM emp;
```

```
SQL> SELECT (sal + comm) "Annual Salary"
2 FROM emp;
```

BIT Academy Course

column alias를 사용한 문장들이다. column heading이 어떻게 나타나는지를 기록하고, Error가 나는 문장에 대해서는 이유를 설명하시오.

```
SQL> SELECT sal*12 annual_salary FROM emp;
```

```
SQL> SELECT sal*12 Annual_Salary FROM emp;
```

```
SQL> SELECT sal*12 Annual Salary FROM emp;
(* Error 발생)
```

```
SQL> SELECT sal*12 "Annual Salary" FROM emp;
```

```
SQL> SELECT sal*12 AS "Annual Salary" FROM emp;
```

### Concatenation Operator

Operator	Purpose
	Concatenates character strings.

- The result of concatenating two character strings is another character string.
- Although Oracle treats zero-length character strings as nulls, concatenating a zero-length character string with another operand always results in the other operand.

BIT Academy Course

연결연산자(||)는 character string들을 연결하여 하나의 결과 character string을 만들어 낸다.

character string에 Null string을 연결시키면 원래의 character string 그대로가 된다.

## Concatenation Operator(cont.)

- Example

```
SQL> SELECT ename || job  
2 FROM emp;
```

```
SQL> SELECT dname || loc  
2 FROM dept;
```

BIT Academy Course

다음은 실습해 보시오.

```
SQL> SELECT empno || ename FROM emp;
```

```
SQL> SELECT empno || ename || hiredate FROM emp;
```

(\* number나 date 값은 default 형태의 character 값으로 자동 변환된 후 연결된다.)

## Literals

- The terms **literal** and **constant value** are synonymous and refer to a fixed data value.
- For example, 'SMITH', and '101' are all character literals; 100 is a numeric literal.
- Character literals are enclosed in single quotation marks, which enable Oracle to distinguish them from schema object names.

BIT Academy Course

Literal은 상수 값을 의미한다.

Character literal은 작은 따옴표로 묶어서, Number literal은 따옴표 없이 그냥 써서 표현한다.

Character literal을 작은 따옴표로 묶어 주어야 Oracle server가 keyword나 object 이름과 구별할 수 있다.

## Literals(cont.)

- Example

```
SQL> SELECT 'Emp# of ' || ename || ' is' || empno
2 FROM emp;
```

```
SQL> SELECT dname || ' is located at ' || loc
2 FROM dept;
```

BIT Academy Course

다음을 실습해 보시오.

```
SQL> SELECT ename || ' ' || sal FROM emp;
```

```
SQL> SELECT ename || ' is working as a ' || job FROM emp;
```

FROM 절의 table을 바꿔가며 literal을 출력해 보시오. 결과 row의 개수는 table의 row 수와 같다.

```
SQL> SELECT 'Korea Fighting' FROM emp;
```

(\* Korea Fighting 이라는 literal이 14 번 출력된다.)

```
SQL> SELECT 'Korea Fighting' FROM dept;
```

(\* Korea Fighting 이라는 literal이 4 번 출력된다.)

literal이나 literal들의 연산결과를 출력해 볼 때는 sys사용자 소유의 dual이라는 dummy table을 활용한다.

```
SQL> SELECT 'Korea Fighting' FROM dual;
```

```
SQL> SELECT 10 + 20 FROM dual;
```

```
SQL> SELECT 'Red' || ' ' || 'Devil' FROM dual;
```

dual table을 이용하여 server의 현재 시각이나 현재 접속중인 DB 사용자를 조회해 볼 수 있다.

```
SQL> SELECT sysdate, user FROM dual;
```

## Duplicate Values

- ALL
  - returns all rows selected, including all copies of duplicates. The default is ALL.
- DISTINCT
  - returns only one copy of each set of duplicate rows selected.
  - use the DISTINCT keyword right after the SELECT keyword for eliminating duplicate values.

BIT Academy Course

ALL이 default로 조회된 결과를 모두 return 해준다.

DISTINCT는 조회 결과 중 중복된 row들은 하나의 결과만 return하도록 해준다.

## Duplicate Values(cont.)

- Example

```
SQL> SELECT ALL job
      2 FROM emp;
```

```
SQL> SELECT DISTINCT job
      2 FROM emp;
```

```
SQL> SELECT DISTINCT deptno, job
      2 FROM emp;
```

BIT Academy Course

다음 문장들을 실행하고 결과를 비교하십시오.

```
SQL> SELECT job FROM emp;
(? rows selected)
SQL> SELECT DISTINCT job FROM emp;
(? rows selected)
SQL> SELECT deptno FROM emp;
(? rows selected)
SQL> SELECT DISTINCT deptno FROM emp;
(? rows selected)
```

다음은 여러 column에 대한 중복 값을 제거하는 문장이다. 결과를 비교하십시오.

```
SQL> SELECT deptno, job FROM emp;
(? rows selected)
SQL> SELECT DISTINCT deptno, job FROM emp;
(? rows selected)
SQL> SELECT DISTINCT job,deptno FROM emp;
(? rows selected)
```



## CASE

```
CASE expr1 WHEN expr2 THEN expr3
      [WHEN expr4 THEN expr5
       .....
       ELSE expr6]
END
```

- CASE expressions are like IF-THEN-ELSE logic.
- IF *expr1* is equal to *expr2* then returns *expr3*, else goes to the next WHEN... THEN.
- IF there is no equal expression in WHEN...THENS, then returns *expr6* in ELSE. IF you omit the ELSE expression, then returns NULL.

BIT Academy Course

CASE는 IF-THEN-ELSE와 비슷한 logic을 제공한다.

각 부서별로 실적에 따라 급여를 다르게 인상하고자 한다. 10번과 20번 부서는 각각 10%, 20% 인상을 하고 나머지 부서는 동결할 경우의 급여를 CASE를 써서 출력하시오.

```
SQL> SELECT ename, CASE deptno WHEN 10 THEN sal * 1.1
      WHEN 20 THEN sal * 1.2
      ELSE sal END new_sal
      FROM emp;
```

## WHERE Clause

```
SELECT select_list
FROM table
[WHERE conditions];
```

- WHERE clause
  - restricts the rows selected to those that satisfy one or more *conditions*.
  - If you omit WHERE clause, Oracle returns all rows from the *table*.
- *Conditions*
  - specifies a combination of one or more expressions and logical operators that evaluates to either TRUE, FALSE, or NULL.

BIT Academy Course

WHERE 절을 사용하지 않으면 FROM 절에 명시된 table의 모든 row를 조회하게 된다.

Table 내의 특정 row만 선택하고 싶을 때 WHERE 절에 조건식(condition)을 써준다.

Oracle server는 Table의 row를 하나씩 읽어 WHERE 절의 조건식을 평가하여 TRUE로 만족하는 것만을 선택한다.

Condition을 평가한 결과는 TRUE, FALSE, NULL 중의 하나이다.

## WHERE Clause(cont.)

- Example

```
SELECT  dname
FROM    dept
WHERE   dname = 'SALES';
```

```
SELECT  ename, sal
FROM    emp
WHERE   sal BETWEEN 1500 AND 2000;
```

BIT Academy Course

condition 내에서 character와 date값의 literal은 작은 따옴표로 싸주고, number 값은 그냥 써준다.

조건식에 써주는 Character값은 대소문자를 구분한다(case sensitive).

Date값은 현재 session의 nls\_date\_format에 맞춰 표현해 준다.

## Comparison Operators

Operator	Purpose
=	Equal to
<>	Not equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to

BIT Academy Course

비교 연산자 중에서 같지 않음을 나타내는 <> 연산자는 모든 운영체제에서 사용 가능하며, 운영 체제 종류에 따라 != 나 ^= 도 사용될 수 있다.

## Comparison Operators(cont.)

- Example

```
SELECT  dname
FROM    dept
WHERE   deptno = 30;
```

```
SELECT  ename, sal
FROM    emp
WHERE   sal >= 1500;
```

BIT Academy Course

**사번이 7788인 사원의 이름과 급여를 출력하시오.**

```
SQL> SELECT ename, sal FROM emp WHERE empno = 7788;
```

**급여가 3000 이 넘는 직종을 출력하시오.**

```
SQL> SELECT job FROM emp WHERE sal > 3000;
```

**PRESIDENT 를 제외한 사원들의 이름과 직종을 출력하시오.**

```
SQL> SELECT ename, job FROM emp WHERE job <> 'PRESIDENT';
```

**BOSTON 지역에 있는 부서의 번호와 이름을 출력하시오.**

```
SQL> SELECT deptno, dname FROM dept WHERE loc = 'BOSTON';
```

**Date 값에 대해 조건을 줄 때는 현재 session의 NLS\_DATE\_FORMAT에 맞춰 주도록 한다.**

```
SQL> SELECT value FROM nls_session_parameters
WHERE parameter = 'NLS_DATE_FORMAT';
```

```
SQL> ALTER SESSION SET nls_date_format = 'DD-MON-RR';
```

```
SQL> SELECT empno, ename FROM emp WHERE hiredate >= '01-JAN-82';
```

```
SQL> ALTER SESSION SET nls_date_format = 'RR/MM/DD';
```

```
SQL> SELECT empno, ename FROM emp WHERE hiredate >= '82/01/01';
```

## Comparison Operators(cont.)

Operator	Purpose
IN ( <i>list</i> )	Equal to any member of <i>list</i> .
ANY ( <i>list</i> )	Compares a value to each value in a <i>list</i> . Must be preceded by =, !=, >, <, <=, >=.
ALL ( <i>list</i> )	Compares a value to every value in a <i>list</i> . Must be preceded by =, !=, >, <, <=, >=.
EXISTS ( <i>subquery</i> )	TRUE if a <i>subquery</i> returns at least one row.

 BIT Academy Course

ANY와 ALL 연산자의 앞에 비교연산자가 반드시 함께 쓰여야 한다.

IN 연산자는 =ANY 연산자와 같은 결과이다.

NOT IN 연산자는 <>ALL 연산자와 같은 결과이다.

EXISTS연산자의 사용 예: 6-7 참조

## Comparison Operators(cont.)

- Example

```
SELECT  dname
FROM    dept
WHERE   deptno IN (10, 20);
```

```
SELECT  ename
FROM    emp
WHERE   job =ANY ('ANALYST', 'CLERK');
```

```
SELECT  ename
FROM    emp
WHERE   sal >ALL (2000, 3000);
```

BIT Academy Course

**BOSTON이나 DALLAS 에 위치한 부서를 출력하시오.**

```
SQL> SELECT dname, loc FROM dept WHERE loc IN('BOSTON', 'DALLAS');
```

```
SQL> SELECT dname, loc FROM dept WHERE loc =ANY('BOSTON', 'DALLAS');
```

**30, 40번 부서에 속하지 않는 직원들을 출력하시오.**

```
SQL> SELECT ename,deptno FROM emp WHERE deptno NOT IN (30,40);
```

```
SQL> SELECT ename,deptno FROM emp WHERE deptno <>ALL (30,40);
```

**DALLAS의 20번 부서, 또는 CHICAGO의 30번 부서를 출력하시오.**

```
SQL> SELECT * FROM dept WHERE (deptno,loc) IN
      ((20, 'DALLAS'), (30, 'CHICAGO'));
```

Comparison Operators(cont.)	
Operator	Purpose
BETWEEN <i>a</i> AND <i>b</i>	greater than or equal to <i>a</i> and less than or equal to <i>b</i> .
<i>a</i> LIKE <i>b</i> [ESCAPE ' <i>c</i> ']	TRUE if <i>a</i> does match the pattern <i>b</i> . ' <i>%</i> ' and ' <i>_</i> ' are wildcard characters within <i>b</i> . A wildcard character is treated as a literal if preceded by the escape character ' <i>c</i> '
IS NULL	Null test

BIT Academy Course

Wildcard '*%*'는 0개 이상의 문자를 대표한다.

Wildcard '*\_*'는 1개의 문자를 대표한다.

Wildcard 문자를 일반 문자로 사용하고 싶을 때 ESCAPE 문자를 사용한다. ESCAPE 문자 바로 뒤에 사용된 wildcard 문자는 일반 문자로 인식된다.

Column의 Null여부를 판단할 때는 반드시 'IS NULL' 혹은 'IS NOT NULL' 연산자를 사용해야 한다.



Comparison Operators(cont.)

- Example

SELECT    ename  
FROM       emp  
WHERE     hiredate BETWEEN '82/01/01'  
          AND '82/12/31';

SELECT    dname  
FROM       dept  
WHERE     dname LIKE 'A%';

SELECT    ename  
FROM       emp  
WHERE     comm IS NULL;

BIT Academy Course

급여가 2000에서 3000 사이인 사원을 출력하시오.

```
SQL> SELECT ename, job, sal FROM emp WHERE sal BETWEEN 2000 AND 3000;
```

이름이 A 로 시작되는 사원을 출력하시오.

```
SQL> SELECT ename FROM emp WHERE ename LIKE 'A%';
```

사번이 8번으로 끝나는 사원을 출력하시오.

```
SQL> SELECT empno, ename FROM emp WHERE empno LIKE '%8';
```

82년도에 입사한 사원을 출력하시오.

```
SQL> SELECT ename, hiredate FROM emp WHERE hiredate LIKE '82%';
```

커미션 지급 대상인 사원을 출력하시오.

```
SQL> SELECT ename, comm FROM emp WHERE comm IS NOT NULL;
```

부서명에 X\_Y 가 포함되어 있는 부서를 출력하시오.

```
SQL> SELECT dname, loc FROM dept WHERE dname LIKE '%X/_Y%' ESCAPE '/';
```

COL 값이 아래와 같을 때 각 LIKE 연산의 결과를 예측하시오.

COL	col LIKE '%X_Y%'	col LIKE 'X_Y%'	col LIKE '%X\_Y%' ESCAPE '\'
XY			
aXY			
XaY			
X_Y			
aXbYc			
aX_Yc			

## Logical Operators

- A logical operator combines the results of two component conditions to produce a single result.

1	Arithmetic operators
2	Comparison operators
3	NOT
4	AND
5	OR

BIT Academy Course

논리연산자를 이용하여 복잡한 질의 조건을 줄 수 있으며, 전체 조건식의 연산 결과가 TRUE인 row만 선택된다.

우선순위는 아래 표의 번호 순서대로이며, 괄호를 이용하여 우선순위를 조정할 수 있다.

우선순위	연산자
1	+, - (Unary)
2	*, /
3	+, -, (Binary)
4	=, <>, <, >, <=, >=, IS NULL, LIKE, BETWEEN, IN
5	NOT
6	AND
7	OR

## Logical Operators(cont.)

NOT	TRUE	FALSE	NULL
	FALSE	TRUE	NULL

AND	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

OR	TRUE	FALSE	NULL
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

SQL의 논리 연산 진리표에는 Null 인 경우의 수가 추가 된다.

## Logical Operators(cont.)

- Example

```
SELECT  ename, job
FROM    emp
WHERE   NOT job = 'ANALYST';
```

```
SELECT  ename, sal, deptno
FROM    emp
WHERE   sal > 2500 AND deptno = 20;
```

```
SELECT  deptno, dname
FROM    dept
WHERE   deptno = 10 OR deptno = 20;
```

 BIT Academy Course

아래 두 문장의 결과를 비교해 보고 차이점을 설명하시오.

```
SQL> SELECT ename, comm FROM emp WHERE comm IS NULL;
```

```
SQL> SELECT ename, comm FROM emp WHERE comm = NULL;
```

직종이 CLERK 인 사원 중에서 급여가 1000 이상인 사원을 출력하시오.

```
SQL> SELECT ename, job, sal FROM emp WHERE job = 'CLERK' AND sal >= 1000;
```

commission을 받는 사원을 출력하시오.

```
SQL> SELECT ename FROM emp WHERE comm IS NOT NULL;
```

```
SQL> SELECT ename FROM emp WHERE NOT comm IS NULL;
```

10번 부서와 20번 부서에 속한 사원을 출력하시오.

```
SQL> SELECT ename, deptno FROM emp WHERE deptno = 10 OR deptno = 20;
```

```
SQL> SELECT ename, deptno FROM emp WHERE deptno IN (10, 20);
```

10번과 20번 부서에 속하지 않는 사원의 이름과 부서번호를 출력하시오.

```
SQL> SELECT ename, deptno FROM emp WHERE deptno <> 10 AND deptno <> 20;
```

```
SQL> SELECT ename, deptno FROM emp WHERE deptno NOT IN (10,20);
```

급여가 2000에서 3000 사이인 사원을 출력하시오.

```
SQL> SELECT ename, job, sal FROM emp WHERE sal >= 2000 AND sal <= 3000;
```

```
SQL> SELECT ename, job, sal FROM emp WHERE sal BETWEEN 2000 AND 3000;
```

## ORDER BY Clause

```
SELECT select_list
FROM table
[WHERE conditions]
[ORDER BY column {,column} [ASC|DESC]];
```

- orders rows returned by the SELECT statements.
- ASC and DESC specify either ascending or descending order. ASC is the default.
- Oracle sorts nulls following all others in ascending order and preceding all others in descending order.

BIT Academy Course

ORDER BY 절에 정렬의 기준이 되는 column을 여러 개 명시할 수 있다. Oracle은 첫 번째 column으로 정렬한 다음, 그 column 값이 같은 row들에 대해서는 두 번째 column값으로 정렬을 해준다.

오름차 순(ASC) 정렬이 default이며, 내림차 순으로 정렬하고자 할 때는 DESC 라는 옵션을 명시해 준다.

ORDER BY 절에 column 명 대신 positional notation을 사용할 수도 있다. Position number는 SELECT절의 column 순서를 의미한다.

## ORDER BY Clause(cont.)

- Example

```
SELECT dname
FROM dept
ORDER BY dname;
```

```
SELECT ename, sal
FROM emp
ORDER BY sal DESC, ename;
```

BIT Academy Course

Null 값은 오름차순의 경우 맨 마지막에, 내림차순의 경우 맨 처음에 display된다.

```
SQL> SELECT ename, comm FROM emp ORDER BY comm;
```

```
SQL> SELECT ename, comm FROM emp ORDER BY comm DESC;
```

급여가 적은 사원부터 출력하시오.

```
SQL> SELECT ename, sal FROM emp ORDER BY sal;
```

급여가 많은 사원부터 출력하시오.

```
SQL> SELECT ename, sal FROM emp ORDER BY sal DESC;
```

급여가 많은 사원부터 출력하되 급여가 같은 경우 이름 순서대로 출력하시오.

```
SQL> SELECT ename, sal FROM emp ORDER BY sal DESC, ename;
```

급여가 많은 사원부터 출력하되 급여가 같은 경우 이름 순서대로 출력하시오. (ORDER BY 절에 숫자 사용)

```
SQL> SELECT ename, sal FROM emp ORDER BY 2 DESC, 1;
```

SELECT 절에 나타나지 않은 column에 대해서도 정렬이 가능하다.

```
SQL> SELECT ename FROM emp ORDER BY sal DESC;
```

```
SQL> SELECT sal, ename FROM emp ORDER BY sal DESC;
```

SQL\*Plus Commands

- Three kinds of commands at the command prompt
  - SQL commands
  - PL/SQL blocks
  - SQL\*Plus commands
- SQL buffer stores recently entered SQL command or PL/SQL block
  - SQL\*Plus commands are not stored in the SQL buffer.
- SQL\*Plus commands are not case sensitive.

BIT Academy Course

SQL 문장을 실행할 때는 문장의 끝에 세미콜론(;)을 붙여주지만, SQL\*Plus 명령어는 맨 뒤에 세미콜론(;)을 붙이지 않아도 실행된다.

SQL\*Plus 명령어는 대소문자를 구분하지 않는다.

SQL\*Plus 명령어는 명령어의 줄임말을 제공한다.

SQL\*Plus 틀에는 SQL Buffer가 있는데 가장 최근에 입력된 SQL 문장 또는 PL/SQL Block 이 저장되어 있다. 이 때, 세미콜론(;)은 /로 변경되어 저장된다.

Buffer의 내용을 보거나 변경하거나 저장할 수 있으며, 다른 파일의 내용을 Buffer로 불러들이는 것도 가능하다.

## SQL Buffer Commands

Command	Purpose
A[PPEND] <i>text</i>	adds <i>text</i> at the end of a line
C[HANGE] / <i>old/new</i>	changes <i>old</i> to <i>new</i> in a line
C[HANGE] / <i>text</i>	deletes <i>text</i> from a line
CL[EAR] BUFF[ER]	deletes all lines
DEL	deletes the current line
DEL <i>n</i>	deletes line <i>n</i>
DEL *	deletes the current line
DEL <i>n</i> *	deletes line <i>n</i> through the current line
DEL LAST	deletes the last line
DEL <i>m n</i>	deletes a range of lines ( <i>m</i> to <i>n</i> )
DEL * <i>n</i>	deletes the current line through line <i>n</i>

 BIT Academy Course

다음 문장들을 실행하여 결과를 확인하시오.

```
SQL> SELECT ename FROM emp;
```

```
SELECT ename FROM emp
```

```
      *
```

1행에 오류:

ORA-00923: FROM 키워드가 있어야할 곳에 없습니다.

```
SQL> LIST
```

```
      1* SELECT ename FROM emp
```

```
SQL> 1 ename FROM emp
```

```
SQL> INPUT WHERE deptno = 10
```

```
SQL> LIST
```

```
      1 ename FROM emp
```

```
      2* WHERE deptno = 10
```

```
SQL> 0 SELECT
```

```
      1 SELECT
```

```
      2 ename FROM emp
```

```
      3* WHERE deptno = 10
```

(다음 페이지에 계속)



## SQL Buffer Commands(cont.)

Command	Purpose
I[INPUT]	adds one or more lines
I[INPUT] <i>text</i>	adds a line consisting of <i>text</i>
L[IST]	lists all lines in the SQL buffer
L[IST] <i>n</i>	lists line <i>n</i>
L[IST] *	lists the current line
L[IST] <i>n</i> *	lists line <i>n</i> through the current line
L[IST] LAST	lists the last line
L[IST] <i>m n</i>	lists a range of lines ( <i>m</i> to <i>n</i> )
L[IST] * <i>n</i>	lists the current line through line <i>n</i>
<i>n</i>	lists line <i>n</i>
<i>n text</i>	Replaces line <i>n</i> to the <i>text</i>

BIT Academy Course

(이전 페이지에서 계속)

```
SQL> LIST 1
      1* SELECT
SQL> APPEND ename FROM emp
      1* SELECT  ename FROM emp
SQL> LIST
      1 SELECT  ename FROM emp
      2 ename FROM emp
      3* WHERE deptno = 10
SQL> LIST 2
      2* ename FROM emp
SQL> DEL
SQL> LIST
      1 SELECT  ename FROM emp
      2* WHERE deptno = 10
SQL> c/10/20
      2* WHERE deptno = 20
SQL> CL BUFF
buffer 소거되었다.
SQL> LIST
```

## SQL\*Plus File Commands

Command	Purpose
ED[IT] [ <i>file_name.ext</i> ]	Invokes a host operating system text editor on the contents of the specified file or on the contents of the buffer.
SAV[E] <i>file_name.ext</i> [CRE[ATE]REP[LACE]]APP [END]]	Saves the contents of the SQL buffer in a host operating system file.
GET <i>file_name.ext</i> [LIST]NOL[IST]]	Loads a host operating system file into the SQL buffer.
RUN or /	Run (or re-run) the current SQL command or PL/SQL block in buffer.
START <i>file_name</i> or @ <i>file_name</i>	Run a command file containing SQL commands, PL/SQL blocks, and/or SQL*Plus commands.

 BIT Academy Course

다음 문장들을 실행하여 결과를 확인하십시오.

```
SQL> SELECT * FROM emp;
```

```
SQL> ED
```

(\* 편집기가 buffer의 내용을 열어 준다. 이 때 세미콜론(;)이 없어지고 맨 마지막 줄에 슬래쉬(/)가 붙어있는 것을 볼 수 있다. 두 번째 줄에 WHERE deptno=10 을 추가한다.)

```
SQL> /
```

```
SQL> SELECT * FROM emp;
```

```
(* 14 rows selected)
```

```
SQL> SAVE s1
```

```
SQL> ED s1
```

(\* WHERE sal > 2000 추가하고 저장)

```
SQL> / (*14 rows selected)
```

```
SQL> @ s1 (* 6 rows selected)
```

```
SQL> / (* 6 rows selected)
```

```
SQL> ED s1
```

(\*2000을 3000으로 변경하고 저장)

```
SQL> / (* 6 rows selected)
```

```
SQL> L (* 2000 확인)
```

```
SQL> GET s1 (* 3000 확인)
```

```
SQL> / (* 1 row selected)
```

For Lab

- Build Summit Sporting Goods DB

BIT Academy Course

실습용 table과 data를 만들기 위해 다음과 같이 실행하시오.

```
SQL> conn system/manager
(* system 사용자로 접속한다.)
SQL> CREATE USER testxx IDENTIFIED BY testxx;
(* xx 에는 각자 고유한 번호를 부여한다.)
SQL> GRANT resource, connect TO testxx;

SQL> conn testxx/testxx
SQL> ALTER SESSION SET nls_date_format = 'DD-MON-RR';
SQL> @ORACLE_HOME\WOCI\WSAMPLES\WSUMMIT2.sql
(* 각자의 ORACLE_HOME 위치를 확인하여 그 path를 적어준다. 부록 1 참조)

SQL> SELECT count(*) FROM s_emp;
(* 25 개가 나와야 한다.)
SQL> SELECT count(*) FROM s_dept;
(* 12 개가 나와야 한다.)

SQL> ALTER SESSION SET nls_date_format = 'RR/MM/DD';
```

# **Single-row SQL Functions**

- Number Functions
- Character Functions
- Data Functions
- Conversion Functions
- 기타 Functions

## Functions

- Functions manipulate data items and return a result.
- Format of Functions

`function(argument, argument, ...)`

  - Functions operate on zero, one, two, or more arguments.
- Two types of functions
  - SQL Functions
  - User-Defined Functions

BIT Academy Course

Function이란 0개 이상의 argument값을 처리하여 하나의 결과값을 return하는 logic이다.

Function에는 Oracle에 미리 정의되어 있는 SQL Function과 사용자가 필요에 따라 만들어서 사용하는 User-Defined Function 두 종류가 있다.

## SQL Functions

- They are built into Oracle and used in SQL statements.
- If data type of an argument is not the same as expected, Oracle implicitly converts the argument to the expected data type before performing the SQL function.
- If a null argument used, the SQL function automatically returns null.
  - Except for CONCAT, DECODE, NVL, and REPLACE.

BIT Academy Course

SQL Function 호출 시 사용한 argument의 data type이 정의된 값과 다를 경우 Oracle은 자동 변환을 시도한다.

대부분의 SQL Function에서 argument값이 NULL인 경우 결과값은 NULL이다.

## SQL Functions(Cont.)

- Single-Row Functions
  - return a single result row for every row of a queried table.
- Aggregate Functions
  - return a single row based on groups of rows, rather than on single rows.

BIT Academy Course

하나의 row 당 하나의 결과가 return되는 함수를 Single-Row Function이라 한다.

여러 row당 하나의 결과가 return되는 함수를 Aggregate Function이라 한다.

## Single-Row Functions

- Functions are grouped by the data types of their arguments and their return values.
  - Number Functions
  - Character Functions
    - Return character values
    - Return number values
  - Date Functions
  - Conversion Functions
  - Miscellaneous Functions



## Main Data Types

Data type	Description	Length/Default
NUMBER( <i>p,s</i> )	Variable-length numeric data. Maximum precision <i>p</i> and/or scale <i>s</i> is 38.	Variable for each row and the maximum space for a column is 21 bytes. Default is 38.
CHAR( <i>size</i> )	Fixed-length character data of length <i>size</i> bytes.	Fixed for every row and maximum <i>size</i> is 2000 bytes. Default <i>size</i> is 1 byte.
VARCHAR2( <i>size</i> )	Variable-length character data.	Variable for each row, up to 4000 bytes per row. A maximum <i>size</i> must be specified.
DATE	Fixed-length date and time data, ranging from January 1, 4712 BCE to December 31, 9999 CE ("A.D.")	Fixed at 7 bytes for each row in the table. Default format is a string (such as RR/MM/DD) specified by NLS_DATE_FORMAT parameter.

Number Functions		
Function	Purpose	Example
ABS( <i>n</i> )	Returns the absolute value of <i>n</i> .	ABS(-5)=>5
CEIL( <i>n</i> )	Returns smallest integer greater than or equal to <i>n</i> .	CEIL(5.6)=>6
FLOOR( <i>n</i> )	Returns largest integer equal to or less than <i>n</i> .	FLOOR(5.6)=>5
MOD( <i>m</i> , <i>n</i> )	Returns remainder of <i>m</i> divided by <i>n</i> . Returns <i>m</i> if <i>n</i> is 0.	MOD(13,2)=>1
POWER( <i>m</i> , <i>n</i> )	Returns <i>m</i> raised to the <i>n</i> th power	POWER(2,3)=>8
ROUND( <i>m</i> , <i>n</i> )	Returns <i>m</i> rounded to <i>n</i> places right of the decimal point.	ROUND(5.6)=>6
TRUNC( <i>m</i> , <i>n</i> )	Returns <i>m</i> truncated to <i>n</i> decimal places.	TRUNC(5.6)=>5
SIGN( <i>n</i> )	If <i>n</i> <0, the function returns -1. If <i>n</i> =0, the function returns 0. If <i>n</i> >0, the function returns 1.	SIGN(-10)=>-1

BIT Academy Course

ROUND, TRUNC는 첫 번째 argument를 소수점 아래 두 번째 argument자리까지 표현한다.

```
SQL> SELECT salary, ROUND(salary, -3), TRUNC(salary, -3) FROM s_emp;
```

ROUND, TRUNC 함수의 두 번째 argument를 생략하면 default로 0 이다.

```
SQL> SELECT ROUND(45.925), ROUND(45.925, 0), TRUNC(45.925),
        TRUNC(45.925, 0)
FROM dual;
```

FLOOR, CEIL 함수는 argument가 1개이며, TRUNC나 ROUND로 바꾸어 표현이 가능하다.

```
SQL> SELECT FLOOR(45.925), CEIL(45.925) FROM dual;
SQL> SELECT TRUNC(45.925), ROUND(45.925) FROM dual
```

다음은 실습해 보시오.

```
SQL> SELECT MOD(10, 3), MOD(10, -3), MOD(-10, 3), MOD(45.925, 10)
        FROM dual;
SQL> SELECT ABS(-15), ABS(15) FROM dual;
SQL> SELECT SIGN(-15), SIGN(15) FROM dual;
SQL> SELECT SIN(3.141592/2) FROM dual;
SQL> SELECT EXP(4) FROM dual;
(* e = 2.71828183 ... )
```

Character Functions	
Function	Purpose
CONCAT( <i>s1</i> , <i>s2</i> )	Returns <i>s1</i> concatenated with <i>s2</i> .
INITCAP( <i>s</i> )	Returns <i>s</i> , with the first letter of each word in uppercase, all other letters in lowercase.
LOWER( <i>s</i> )	Returns <i>s</i> , with all letters lowercase.
UPPER( <i>s</i> )	Returns <i>s</i> , with all letters uppercase.
LPAD( <i>s1</i> , <i>n</i> , <i>s2</i> )	Returns <i>s1</i> , left-padded to length <i>n</i> with the sequence of characters in <i>s2</i> .
RPAD( <i>s1</i> , <i>n</i> , <i>s2</i> )	Returns <i>s1</i> , right-padded to length <i>n</i> with the sequence of characters in <i>s2</i> .
LTRIM( <i>s</i> , <i>c</i> )	Removes characters from the left of <i>s</i> , with all the leftmost characters that appear in <i>c</i> removed.
RTRIM( <i>s</i> , <i>c</i> )	Removes characters from the right of <i>s</i> , with all the rightmost characters that appear in <i>c</i> removed.

BIT Academy Course

다음 문장들을 실행하여 결과를 확인하시오.

```
SQL> connect testxx/testxx
SQL> SET LINESIZE 120
SQL> SELECT CONCAT('Oracle','DBMS'), INITCAP('Oracle DBMS'),
        LOWER('Oracle DBMS'), UPPER('Oracle DBMS')
        FROM dual;
SQL> SELECT LPAD('Oracle DBMS', 13, 'x'), RPAD('Oracle DBMS', 13, 'x')
        FROM dual;
SQL> SELECT LPAD(last_name, 15)||' '||RPAD(title,20) FROM s_emp;
SQL> SELECT last_name , LTRIM(last_name, 'Ma'), RTRIM(last_name, 'le')
        FROM s_emp;

SQL> SELECT 'The title for ' || INITCAP(first_name) || ' ' ||
        UPPER(last_name) || ' is ' || LOWER(title)
        || '.' "EMPLOYEE'S JOB STATUS"
        FROM s_emp;
```

DB 상의 data가 대소문자를 구별하므로 저장된 정확한 형태를 알지 못하는 경우 Function을 이용하여 data를 변형시킨 후 비교하기도 한다.

```
SQL> SELECT last_name, salary FROM s_emp WHERE title = 'PRESIDENT';
SQL> SELECT last_name, salary FROM s_emp WHERE UPPER(title) = 'PRESIDENT';
```

**Oracle 9i 第8章 (SQL, PL/SQL) >>>**

```
SQL> SELECT last_name, salary FROM s_emp
      WHERE LOWER(title) = 'president';
SQL> SELECT last_name, salary FROM s_emp
      WHERE INITCAP(title) = 'President';
```

## Character Functions(Cont.)

Function	Purpose
CHR( <i>n</i> )	Returns the character having the binary equivalent to <i>n</i>
REPLACE( <i>s,p,r</i> )	Returns <i>s</i> with every occurrence of <i>p</i> replaced with <i>r</i>
SUBSTR( <i>s,m,n</i> )	Returns a portion of <i>s</i> , beginning at character <i>m</i> , <i>n</i> characters long
TRANSLATE( <i>s,from,to</i> )	Returns <i>s</i> with all occurrences of each character in <i>from</i> replaced by its corresponding character in <i>to</i>
ASCII( <i>s</i> )	Returns the decimal representation in the database character set of the first character of <i>s</i>
INSTR( <i>s1,s2,m,n</i> )	Searches <i>s1</i> beginning with its <i>m</i> th character for the <i>n</i> th occurrence of <i>s2</i> and returns the position of the character in <i>s1</i> that is the first character of this occurrence
LENGTH( <i>s</i> )	Returns the length of <i>s</i> in characters

BIT Academy Course

다음은 실습해 보시오.

```
SQL> SELECT CHR(79) || CHR(114) || CHR(97) || CHR(99) || CHR(108) || CHR(101)
FROM dual;
SQL> SELECT ASCII('O'),ASCII('r'),ASCII('a') FROM dual;
SQL> SELECT REPLACE('Oracle DB System','DB','Database') FROM dual;
SQL> SELECT SUBSTR('Oracle DB System',2,4) FROM dual;
SQL> SELECT TRANSLATE('Oracle DBMS','ABCD','1234') FROM dual;
SQL> SELECT INSTR('Oracle DBMS','a') FROM dual;
SQL> SELECT LENGTH('Oracle DBMS') FROM dual;
SQL> SELECT SUBSTR(first_name, 1,3) FROM s_emp;
SQL> SELECT SUBSTR(first_name, 1) FROM s_emp;
SQL> SELECT SUBSTR(first_name, -5, 3) FROM s_emp;
```

LENGTH는 날짜 값의 경우 문자열로 display이되는 길이를 return한다.

```
SQL> SELECT LENGTH(last_name), LENGTH(start_date) FROM s_emp;
```

INSTR 함수는 지정된 문자가 문자열 내에 없을 때는 0을 return한다.

```
SQL> SELECT INSTR(title, 'VP'), INSTR(title, '%') FROM s_emp;
```

Number data의 경우 character string으로 자동 변환된 후에 처리된다.

```
SQL> SELECT CONCAT(title, salary) FROM s_emp;
SQL> SELECT INSTR(salary,'00') FROM s_emp;
SQL> SELECT LPAD(salary, 10, 'W') FROM s_emp;
SQL> SELECT LENGTH(salary) FROM s_emp;
```

Date Functions	
Function	Purpose
ADD_MONTHS( <i>d,n</i> )	Returns the date <i>d</i> plus <i>n</i> months
LAST_DAY( <i>d</i> )	Returns the date of the last day of the month that contains <i>d</i>
MONTHS_BETWEEN( <i>d1,d2</i> )	Returns number of months between dates <i>d1</i> and <i>d2</i>
NEW_TIME( <i>d,z1,z2</i> )	Returns the date and time in time zone <i>z2</i> when date and time in time zone <i>z1</i> are <i>d</i>
NEXT_DAY( <i>d,day</i> )	Returns the date of the first weekday named by <i>day</i> that is later than the date <i>d</i>
ROUND( <i>d,fmt</i> )	Returns <i>d</i> rounded to the unit specified by the format model <i>fmt</i>
TRUNC( <i>d,fmt</i> )	Returns <i>d</i> with the time portion of the day truncated to the unit specified by the format model <i>fmt</i>
SYSDATE	Returns the current date and time

BIT Academy Course

**다음은 실습해 보시오.**

```
SQL> SELECT ADD_MONTHS(sysdate, 5), ADD_MONTHS(sysdate, -5) FROM dual;
SQL> SELECT LAST_DAY('03/01/01'), LAST_DAY('03/02/01') FROM dual;
SQL> SELECT MONTHS_BETWEEN('03/01/01', '03/07/01') FROM dual;
SQL> SELECT TO_CHAR(NEW_TIME(TO_DATE('03:10:30', 'HH24:MI:SS'),
    'EST', 'GMT'), 'HH24:MI:SS') FROM dual;
SQL> SELECT NEXT_DAY(sysdate, '일요일') FROM dual;
SQL> ALTER SESSION SET NLS_DATE_LANGUAGE='AMERICAN';
(* Date 값에 사용되는 언어를 바꿔준다.)
SQL> SELECT NEXT_DAY(sysdate, 'SUN') FROM dual;

SQL> SELECT ROUND(TO_DATE('03/07/16'), 'YEAR') FROM dual;
SQL> SELECT TRUNC(TO_DATE('03/07/16'), 'YEAR') FROM dual;

SQL> SELECT ROUND(TO_DATE('03/07/16'), 'MONTH') FROM dual;
SQL> SELECT TRUNC(TO_DATE('03/07/16'), 'MONTH') FROM dual;

SQL> SELECT ROUND(TO_DATE('03/07/16'), 'DAY') FROM dual;
SQL> SELECT ROUND(TO_DATE('03/07/17'), 'DAY') FROM dual;
SQL> SELECT TRUNC(TO_DATE('03/07/16'), 'DAY') FROM dual;
SQL> SELECT TRUNC(TO_DATE('03/07/17'), 'DAY') FROM dual;

SQL> SELECT sysdate FROM dual;
```

## Date Arithmetic Operations

- $\text{DATE} + d \Rightarrow \text{DATE}$
- $\text{DATE} - d \Rightarrow \text{DATE}$
- $\text{DATE} - \text{DATE} \Rightarrow d$
- $\text{DATE} + d/24 \Rightarrow \text{DATE}$

BIT Academy Course

**오늘 현재 날짜 5일 뒤의 날짜와 5일 전의 날짜를 출력하시오.**

```
SQL> SELECT sysdate + 5, sysdate - 5 FROM dual;
```

**성이 Biri 인 사원의 근무 일수를 출력하시오.**

```
SQL> SELECT sysdate - start_date FROM s_emp
      WHERE last_name = 'Biri';
```

(\* 조회 결과 정수 부분을 근무일수가 되며 소수부분은 시간으로 나타낼 수 있다.

만약 4479.45341가 return이 되었다면 4479 + 0.45341에서 소수부분에 24를 곱하면,  
 24시간 \* 0.45341 = 10.88184 즉, 10시간 + 0.88184 시간이다. 소수부분에 60을 곱하면,  
 60분 \* 0.88184 = 52.9104 즉, 52분 + 0.9104 분이다.

결국, 약 4479 일 10시간 53분 정도라고 환산해 볼 수 있다.)

**근무한 지 100000 시간이 넘은 사원을 출력하시오.**

```
SQL> SELECT last_name FROM s_emp
      WHERE (sysdate - start_date) * 24 > 100000;
```

**모든 사원에 대해 근무한 지 몇 주가 지났는지를 출력하시오.**

```
SQL> SELECT (sysdate - start_date)/7 weeks FROM s_emp;
```

## Conversion Functions

- convert a value from one data type to another.

Function	Purpose
TO_CHAR( <i>d</i> , <i>fmt</i> )	Converts <i>d</i> of DATE data type to a value of VARCHAR2 data type in the format specified by the date format <i>fmt</i> .
TO_CHAR( <i>n</i> , <i>fmt</i> )	Converts <i>n</i> of NUMBER data type to a value of VARCHAR2 data type, using the optional number format <i>fmt</i> .
TO_DATE( <i>s</i> , <i>fmt</i> )	Converts <i>s</i> of CHAR or VARCHAR2 data type to a value of DATE data type. The <i>fmt</i> is a date format specifying the format of <i>s</i> .
TO_NUMBER( <i>s</i> , <i>fmt</i> )	Converts <i>s</i> , a value of CHAR or VARCHAR2 data type containing a number in the format specified by the optional format model <i>fmt</i> , to a value of NUMBER data type.



## Format Model

- A character literal that describes the format of DATE or NUMBER data stored in a character string.
- A format model is used as an argument of the TO\_CHAR and TO\_DATE functions.
  - To specify the format for Oracle to use to return a value from the database.
  - To specify the format for a value you have specified for Oracle to store in the database.
- A format model does not change the internal representation of the value in the database.

BIT Academy Course

Format model은 character string이므로 작은 따옴표로 반드시 묶어준다.

Format Elements for Number		
<i>fmt</i>	Description	Example
9	Returns value with the specified number of digits	99999
0	Returns zeros.	(09999
,	Returns a comma in the specified position.	99,999
.	Returns a decimal point (.) in the specified position.	999.99
\$	Returns value with a leading dollar sign.	\$99999
FM	Returns a value with no leading or trailing blanks.	FM90.9
L	Returns in the specified position the local currency symbol.	L99,999
MI	Returns negative value with a trailing minus sign (-).	99999MI
PR	Returns negative value in <angle brackets>.	99999PR
RN	Returns a value as Roman numerals .	RN m
S	Returns sign - or +.	S99,999
X	Returns the hexadecimal value.	XXX xxx

BIT Academy Course

다음과 같이 여러 가지 형식을 사용하여 TO\_CHAR 함수를 실습해 보시오.

```
SQL> SELECT TO_CHAR(salary, '99,999'),TO_CHAR(salary, '099,999') ,
           TO_CHAR(salary, 'FM99,999'),TO_CHAR(salary, '99,999.0')
FROM s_emp;

SQL> SELECT TO_CHAR(-12345, '99,999MI'), TO_CHAR(-12345, '99,999PR')
FROM dual;

SQL> SELECT TO_CHAR(salary, 'RN'), TO_CHAR(salary, 'rn') FROM s_emp;
SQL> SELECT TO_CHAR(-12345,'S99,999'), TO_CHAR(12345,'S99,999')FROM dual;
SQL> SELECT TO_CHAR(salary, 'XXX'), TO_CHAR(salary, 'xxx') FROM s_emp;
```

현재 session의 Local Currency를 확인한 후 다음을 실습해 보시오.

```
SQL> SELECT value FROM nls_session_parameters
       WHERE parameter = 'NLS_CURRENCY';

SQL> ALTER SESSION SET NLS_CURRENCY='₩';

SQL> SELECT TO_CHAR(salary,'$99,999'), TO_CHAR(salary,'L99,999') FROM
s_emp;
```

TO\_NUMBER 함수는 character string이 number와 +, -로만 구성된 경우 format을 주지 않고도 변환이 가능하다.

```
SQL> SELECT TO_NUMBER('1234') FROM dual;
SQL> SELECT TO_NUMBER('-1234') FROM dual;
SQL> SELECT TO_NUMBER('+1234') FROM dual;
SQL> SELECT TO_NUMBER('$123,456', '$999,999') FROM dual;
```

## Format Elements for Date

<i>fmt</i>	Description	Range
SS	Second.	0 – 59
SSSSS	Seconds past midnight.	0 – 86399
MI	Minute.	0 – 59
HH, HH24	Hour of day.	0 – 12,23
AM,PM	Meridian indicator.	AM,PM
DD	Day of month.	1 – 31
DAY	Name of day.	SUNDAY – SATURDAY
DY	Abbreviated name of day.	SUN – SAT
D	Day of week.	1 – 7
DDD	Day of year.	1 – 366

BIT Academy Course

Format character는 최대 22자까지이다.

현재의 시각을 출력하시오.

```
SQL> SELECT TO_CHAR(sysdate, 'HH24"시" MI"분" SS"초"') FROM dual;
SQL> SELECT TO_CHAR(sysdate, 'HHAM'), TO_CHAR(sysdate, 'HHPM')
FROM dual;
```

오늘이 올해의 몇 번째 날인지를 출력하시오.

```
SQL> SELECT TO_CHAR(sysdate, 'DDD"일"') FROM dual;
```

오늘의 요일을 출력하시오.

```
SQL> SELECT TO_CHAR(sysdate, 'DAY DY') FROM dual;
```

사원들 입사 요일을 출력하시오.

```
SQL> SELECT TO_CHAR(start_date, 'DAY"x"') FROM s_emp;
(* 이 때, 요일의 이름은 고정된 9자리 string으로 변환되어 출력된다.)
SQL> SELECT TO_CHAR(start_date, 'FMDAY"x"') FROM s_emp;
(* 이 때, 요일의 이름은 공백문자를 제거한 가변길이 string으로 변환되어 출력된다.)
```

## Format Elements for Date (Cont.)

<i>fmt</i>	Description	Range
W	Week of month.	1 - 5
WW	Week of year.	1 - 53
MM	Two-digit numeric abbreviation of month.	1 - 12
MON	Abbreviated name of month.	JAN - DEC
MONTH	Name of month.	JANUARY - DECEMBER
Q	Quarter of year.	1 - 4
RM	Roman numeral month.	I - XII
AD,BC	AD, BC Indicator.	AD, BC
Y,YY,YYY	1,2,3-digit year.	
YYYY, SYYYY	4-digit year. "S" prefixes BC dates with "-".	

BIT Academy Course

오늘이 올해의 몇 번째 주의 몇 번째 날인지를 출력하시오.

```
SQL> SELECT TO_CHAR(sysdate, 'WW'주" D"일"') FROM dual;
```

사원들의 입사일을 출력하시오.

```
SQL> SELECT TO_CHAR(start_date, 'BC YYYY Q MM DD') FROM s_emp;
```

```
SQL> SELECT TO_CHAR(start_date, 'AD YY Q MON DD') FROM s_emp;
```

사원들 입사일의 월을 알파벳 전체 이름으로 출력하시오.

```
SQL> SELECT TO_CHAR(start_date, 'MONTH"월"') FROM s_emp;
```

(\* 이 때, 월의 이름은 고정된 9자리 string으로 변환되어 출력된다.)

```
SQL> SELECT TO_CHAR(start_date, 'FMMONTH"월"') FROM s_emp;
```

(\* 이 때, 월의 이름은 공백문자를 제거한 가변길이 string으로 변환되어 출력된다.)

TO\_DATE 함수를 사용하여 character string을 date 값으로 변환하시오.

```
SQL> SELECT TO_DATE('1966, 2, 8', 'YYYY, MM, DD') FROM dual;
```

```
SQL> SELECT TO_DATE('April', 'Month') FROM dual;
```

```
SQL> SELECT TO_DATE('03', 'YY') FROM dual;
```

```
SQL> SELECT TO_DATE('03', 'MM') FROM dual;
```

```
SQL> SELECT TO_DATE('03', 'DD') FROM dual;
```

## Format Elements for Date (Cont.)

<i>fmt</i>	Description
YEAR, SYEAR	Year, spelled out. "S" prefixes BC dates with "-".
RR	Given a year with 2 digits. <i>Returns a year in the next century if the year is &lt;50 and the last 2 digits of the current year are &gt;=50.</i> <i>Returns a year in the preceding century if the year is &gt;=50 and the last 2 digits of the current year are &lt;50.</i>
RRRR	Round year.
CC, SCC	One greater than the first two digits of a four-digit year; "S" prefixes BC dates with "-".
J	Julian day, the number of days since January 1, 4712 BC.
SP	Spelled number.
TH	Ordinal number.

BIT Academy Course

변환 함수를 사용하지 않으면 날짜 값은 NLS\_DATE\_FORMAT에 맞춰 출력된다.

SQL> COL value FORMAT A20(\*value라는 column을 20문자폭으로 출력하도록 설정한다.)

SQL> SELECT value FROM NLS\_SESSION\_PARAMETERS

WHERE parameter = 'NLS\_DATE\_FORMAT';

SQL> ALTER SESSION SET NLS\_DATE\_FORMAT = 'DD Month YYYY';

SQL> SELECT sysdate FROM dual;

SQL> ALTER SESSION SET NLS\_DATE\_FORMAT = 'RR/MM/DD';

SQL> SELECT sysdate FROM dual;

다음은 실습해 보시오.

SQL> SELECT TO\_CHAR(sysdate, '"오늘의 날짜" yyyy month dd') FROM dual;

(\* Format에 임의의 character string을 추가할 때는 이중 인용 부호(" ")로 묶어준다.)

SQL> SELECT TO\_CHAR(sysdate, 'CC YEAR MONTH DD') FROM dual;

SQL> SELECT TO\_CHAR(sysdate, 'J') FROM dual;

SQL> SELECT TO\_CHAR(sysdate, 'CCSP YEAR MONTH DAY') FROM dual;

SQL> SELECT TO\_CHAR(sysdate, 'FMCCSP YEAR MONTH DAY') FROM dual;

SQL> SELECT TO\_CHAR(start\_date, 'QSPTH') FROM s\_emp;

TO\_CHAR 함수에서 character string이 NLS\_DATE\_FORMAT에 맞춰져 있을 때는 format을 주지 않아도 변환이 가능하다.

SQL> SELECT TO\_DATE('66/02/08') FROM dual;

SQL> SELECT sysdate - TO\_DATE('66/02/08') FROM dual;

## RR Format

- Oracle provides RR format of year for solving some Y2K problems on DATE data type operations.

현재 연도	처리 연도	YY	RR
1950 - 1999	00 - 49	1900 - 1949	2000 - 2049
	50 - 99	1950 - 1999	1950 - 1999
2000 - 2049	00 - 49	2000 - 2049	2000 - 2049
	50 - 99	2050 - 2099	1950 - 1999

BIT Academy Course

현재가 2003년 일 때 다음 문장들을 실행하여 결과를 확인하시오.

```
SQL> SELECT TO_CHAR(TO_DATE('30/01/01','YY/MM/DD'),'YYYY') FROM dual ;
```

(\* 결과가 2030으로 나타난다.)

```
SQL> SELECT TO_CHAR(TO_DATE('30/01/01','RR/MM/DD'),'YYYY') FROM dual ;
```

(\* 결과가 2030으로 나타난다.)

```
SQL> SELECT TO_CHAR(TO_DATE('80/01/01','YY/MM/DD'),'YYYY') FROM dual ;
```

(\* 결과가 2080으로 나타난다.)

```
SQL> SELECT TO_CHAR(TO_DATE('80/01/01','RR/MM/DD'),'YYYY') FROM dual ;
```

(\* 결과가 1980으로 나타난다.)

현재의 시각을 1999년으로 설정한 후 위의 문장들을 실행하여 결과를 확인하시오.

## Miscellaneous Functions

Function	Purpose
GREATEST( <i>expr</i> ,...)	Returns the greatest of the list of <i>exprs</i> .
LEAST( <i>expr</i> ,...)	Returns the least of the list of <i>exprs</i>
USER	Returns the current Oracle user with the data type VARCHAR2.
VSIZE( <i>expr</i> )	Returns the number of bytes in the internal representation of <i>expr</i>
DECODE( <i>expr</i> , <i>search</i> <i>h</i> , <i>result</i> ,..., <i>default</i> )	If <i>expr</i> is equal to a <i>search</i> , Oracle returns the corresponding <i>result</i> . If no match is found, Oracle returns <i>default</i> , or, if <i>default</i> is omitted, returns null.

 BIT Academy Course

각 부서별로 실적에 따라 급여를 다르게 인상하고자 한다. 10번과 50번 부서는 각각 10%, 20% 인상을 하고 나머지 부서는 동결할 경우의 급여를 DECODE 함수를 써서 출력하시오.

```
SQL> SELECT DECODE(dept_id, 10, salary * 1.1, 50, salary * 1.2, salary)
        FROM s_emp;
```

다음은 실습해 보시오.

```
SQL> COL last_name FORMAT A20
```

```
SQL> COL first_name FORMAT A20
```

```
SQL> COL g_name FORMAT A20
```

```
SQL> COL l_name FORMAT A20
```

```
SQL> SET LINESIZE 100 (* 화면 출력을 100문자 폭으로 설정)
```

```
SQL> SELECT last_name, first_name, GREATEST(last_name, first_name) G_NAME,
        LEAST(last_name, first_name) L_NAME
        FROM s_emp;
```


```
SQL> SELECT VSIZE(last_name), VSIZE(start_date), VSIZE(salary)
        FROM s_emp;
```

```
SQL> SELECT user FROM dual;
```

```
SQL> CONN scott/tiger
```

```
SQL> SELECT user FROM dual;
```

Miscellaneous Functions(Cont.)	
Function	Purpose
NVL( <i>expr1</i> , <i>expr2</i> )	If <i>expr1</i> is NULL then returns <i>expr2</i> .
NVL2 ( <i>expr1</i> , <i>expr2</i> , <i>expr3</i> )	If <i>expr1</i> is NOT NULL then returns <i>expr2</i> , else returns <i>expr3</i> .
NULLIF ( <i>expr1</i> , <i>expr2</i> )	If <i>expr1</i> and <i>expr2</i> are equal then returns NULL, else returns <i>expr1</i> .
COALESCE( <i>expr1</i> , <i>expr2</i> ,... <i>exprN</i> )	Returns the first NOT NULL <i>expr</i> in the list.

 BIT Academy Course

다음은 실습해 보시오.

```
SQL> conn testxx/testxx
```

```
SQL> SELECT NVL(TO_CHAR(manager_id), 'No Manager') FROM s_emp;
```

```
SQL> SELECT NVL2(TO_CHAR(manager_id), 'Manager exists','No Manager')
FROM s_emp;
```

```
SQL> SELECT last_name, first_name,
NULLIF(last_name, GREATEST(last_name, first_name))
FROM s_emp;
```

```
SQL> SELECT last_name,
COALESCE(commission_pct, salary, (select avg(salary) from s_emp)) sal
FROM s_emp;
```



## Nesting Single-Row Functions

- Single-Row functions can be nested.
- Nested functions are evaluated from the most inner function to the outer ones.
- Example

```
SQL> SELECT TO_CHAR(NEXT_DAY(ADD_MONTHS
                        (hiredate, 3), '금요일'),
                        'fmDay, Month ddth, YYYY')
                        "End of OJT"
FROM emp;
```

BIT Academy Course

아래 두 문장의 실행 결과를 비교해 보시오.

```
SQL> SELECT last_name FROM s_emp WHERE INSTR(title, 'VP') > 0;
SQL> SELECT last_name FROM s_emp WHERE title LIKE '%VP%';
```

사원의 이름과 매니저 사번을 출력하시오. 단, 매니저가 없는 사원의 경우 'TOP'이라고 출력하시오.

```
SQL> COL manager FORMAT A20
SQL> SELECT id, NVL(TO_CHAR(manager_id), 'TOP') manager
FROM s_emp;
```

매월 1,3주 토요일은 휴무이다. 현재 월의 휴무 일을 출력하시오.

```
SQL> ALTER SESSION SET NLS_DATE_LANGUAGE='KOREAN';
SQL> SELECT DECODE(TO_CHAR(TRUNC(sysdate, 'MONTH'), 'D'),
                    7, TRUNC(sysdate, 'MONTH'),
                    NEXT_DAY(TRUNC(sysdate, 'MONTH'), '토요일')) FIRST,
                    DECODE(TO_CHAR(TRUNC(sysdate, 'MONTH'), 'D'),
                    7, TRUNC(sysdate, 'MONTH')+14,
                    NEXT_DAY(TRUNC(sysdate, 'MONTH'), '토요일')+14) THIRD
FROM dual;
```

# **Advanced Query (Join)**

- Cartesian Product
- Equijoin
- Non–equijoin
- Outer join
- Self join

## Cartesian Product

```
SELECT *
FROM emp, dept;
```

- If two tables in a join query have no join condition, Oracle returns their Cartesian product.
- Oracle combines each row of one table with each row of the other.
- Always include a join condition unless you need a Cartesian product.

BIT Academy Course

위와 같이 FROM 절에 두 개 혹은 그 이상의 table 명을 기술할 수 있다. 위 문장의 수행 결과 Cartesian Product가 발생하여 s\_emp 25 row \* s\_dept 12 row = 300 row의 결과가 나오게 된다.

대부분의 SELECT 문장에서는 위와 같이 두 table을 Cartesian Product 하지는 않는다. 예를 들어, s\_dept의 PK인 id와 s\_emp의 FK인 dept\_id 가 같은 row끼리만 업무적으로 의미 있는 data 이므로 WHERE 절에 두 table을 연관시키기 위한 condition을 기술해 준다. 이렇게 두 개 이상의 table 을 연관시키는 것을 Join이라고 하며 WHERE 절 내용을 Join condition이라고 한다.

```
SQL> SELECT * FROM s_emp, s_dept WHERE s_dept.id = s_emp.dept_id;
```

```
SQL> SELECT * FROM s_dept, s_emp WHERE s_dept.id = s_emp.dept_id;
```

## Table Join

```
SELECT table1.column, table2.column...  
FROM table1, table2  
WHERE table1.column = table2.column;
```

- If you need data from more than one table, table join is required.
- Usually you should write WHERE clause that contains join condition.
- The join condition is usually composed of PK and FK from each table.

BIT Academy Course

최종적으로 얻고자 하는 data가 여러 개의 table에 흩어져 있는 경우, 또는 최종적으로 얻고자 하는 data를 가져오기 위해 여러 table을 연관시켜야 하는 경우 Join을 사용한다.

대부분의 경우 한 table의 Primary Key와 다른 table의 Foreign Key를 사용하여 join condition을 작성하게 된다.

Join condition이 빠지면 Cartesian product가 발생하므로 유의하여야 한다.

## Table Join Types

- Oracle Term
  - Cartesian Product
  - Equijoin
  - Non-equijoin
  - Outer join
  - Self join
- ANSI Term
  - Cross join
  - Natural Join, Join Using, Join On
  - Right Outer Join, Left Outer Join, Full Outer Join

## Table Alias

```
SELECT ename, dname  
FROM emp, dept  
WHERE deptno = deptno; (Error)
```

- Oracle can't distinguish which table's column is used in the query if tables have the same column name.
- If you run the above query, Oracle will raise an error of ambiguous column.

BIT Academy Course

동일한 column (id)이 두 table에 모두 존재하면 아래와 같이 에러가 발생한다. Oracle이SELECT 문을 compile하다가 id column에 대해 어느 table의 column인지 판단할 수가 없기 때문이다.

```
SQL> SELECT * FROM s_dept, s_emp WHERE id = dept_id;  
SELECT * FROM s_dept, s_emp WHERE id = dept_id  
*  
ERROR at line 1:  
ORA-00918: column ambiguously defined
```

## Table Alias(cont.)

```
SELECT ename, dname
FROM emp, dept
WHERE emp.deptno = dept.deptno;
```

```
SELECT e.ename, d.dname
FROM emp e, dept d
WHERE e.deptno = d.deptno;
```

- You should write a table name in front of a column name in the join query if the tables have the same column name.
- You can use table alias for clearing which table has the column instead of table full name.

BIT Academy Course

column이 어느 table 소속인지를 명확히 하기 위해 아래와 같이 WHERE 절을 수정해 준다.

```
SQL> SELECT * FROM s_dept, s_emp WHERE id = dept_id;
(* Error발생 )
SQL> SELECT * FROM s_dept, s_emp WHERE s_dept.id = s_emp.dept_id;
```

복잡한 SELECT 문장을 작성하는 경우, 또는 table 명이 길어서 불편하거나, 또는 동일한 table을 두 번 이상 FROM 절에 사용하는 경우 등에는 table alias를 쓰도록 한다.

```
SQL> SELECT * FROM s_dept d, s_emp e WHERE d.id = e.dept_id;
```

## Equijoin

- A equijoin is a query retrieves data from more than two related tables using equality comparison operator in the join condition.
- N-1 join conditions are needed when N tables are related.

```
SELECT *  
FROM dept d, emp e  
WHERE d.deptno = e.deptno;
```

BIT Academy Course

Cartesian product의 예이다. 다음 문장들의 실행 결과를 비교하시오.

```
SQL> conn scott/tiger  
SQL> SELECT * FROM emp, dept;  
SQL> SELECT * FROM emp CROSS JOIN dept;  
(* SQL:1999 Syntax)
```

Equijoin의 예이다. 다음 문장들의 실행 결과를 비교하시오.

```
SQL> SELECT * FROM dept, emp  
WHERE dept.deptno = emp.deptno;  
  
SQL> SELECT * FROM dept NATURAL JOIN emp;  
(* SQL:1999 Syntax)  
  
SQL> SELECT * FROM dept JOIN emp USING(deptno);  
(* SQL:1999 Syntax)  
  
SQL> SELECT * FROM dept d JOIN emp e ON (d.deptno = e.deptno);  
(* SQL:1999 Syntax)
```



### Equijoin(cont.)

EMP				DEPT		
EMPNO	ENAME	.....	DEPTNO	DEPTNO	DNAME	LOC
7839	KING		10	10	ACCOUNTING	NEW YORK
7566	JONES		20	20	RESEARCH	DALLAS
7900	JAMES		30	30	SALES	CHICAGO
7369	SMITH		20	20	RESEARCH	DALLAS
7499	ALLEN		30	30	SALES	CHICAGO

BIT Academy Course

join condition 이외에 WHERE 절에 추가적인 condition을 AND나 OR 연산자를 이용하여 줄 수 있다.

31번 부서의 이름과 그 부서에 근무하는 사원의 이름을 출력하시오.

```
SQL> conn testxx/testxx
```

```
SQL> SELECT d.name, e.last_name FROM s_dept d, s_emp e
      WHERE d.id = e.dept_id AND d.id = 31;
```

1,2 번 지역의 이름과 그곳에 있는 부서의 이름을 출력하시오.

```
SQL> SELECT r.name, d.name FROM s_region r, s_dept d
      WHERE r.id = d.region_id AND (r.id = 1 OR r.id = 2);
```

Bunny Boot 라는 제품에 대한 주문번호, 주문 날짜, 수량을 출력하시오.

```
SQL> SELECT o.id, o.date_ordered, i.quantity, p.name
      FROM s_ord o, s_item i, s_product p
      WHERE o.id = i.ord_id AND i.product_id = p.id AND p.name =
      'Bunny Boot';
```

Asia 지역 고객의 이름과 전화번호, 담당 영업 사원의 이름을 출력하시오.

```
SQL> SELECT c.name, c.phone, e.last_name
      FROM s_customer c, s_region r, s_emp e
      WHERE c.region_id = r.id AND c.sales_rep_id = e.id AND r.name =
      'Asia';
```

## Non-equijoin

- If a join condition is containing something other than an equality comparison operator, the query is a non-equijoin.

```
SELECT *  
FROM emp e, salgrade s  
WHERE e.sal BETWEEN s.losal  
      AND s.hisal;
```

Equal(=)이외의 연산자를 사용하여 join condition을 작성한 경우 non-equijoin이라고 한다.

## Non-equijoin(cont.)

EMP				SALGRADE		
EMPNO	ENAME	SAL	.....	GRADE	LOSAL	HISAL
7839	KING	5000		1	700	1200
7566	JONES	2975		2	1201	1400
7900	JAMES	950		3	1401	2000
7369	SMITH	800		4	2001	3000
7499	ALLEN	1600		5	3001	9999

EMPNO	ENAME	SAL	.....	GRADE	LOSAL	HISAL
7839	KING	5000		5	3001	9999
7566	JONES	2975		4	2001	3000
7900	JAMES	950		1	700	1200
7369	SMITH	800		1	700	1200
7499	ALLEN	1600		3	1401	2000

BIT Academy Course

사원의 이름과 직종, 급여 그리고 급여의 등급을 출력하시오.

```
SQL> conn SCOTT/TIGER
SQL> SELECT e.ename, e.job, e.sal, s.grade
FROM emp e, salgrade s
WHERE e.sal BETWEEN s.losal AND s.hisal;
```

## Outer Join

- An outer join returns all rows that satisfy the join condition and those rows from one table for which no rows from the other satisfy the join condition.
- To write a query that performs an outer join of tables A and B and returns all rows from A, apply the outer join operator (+) to all columns of B in the join condition..

```
SELECT dname, ename  
FROM dept d, emp e  
WHERE d.deptno = e.deptno(+);
```

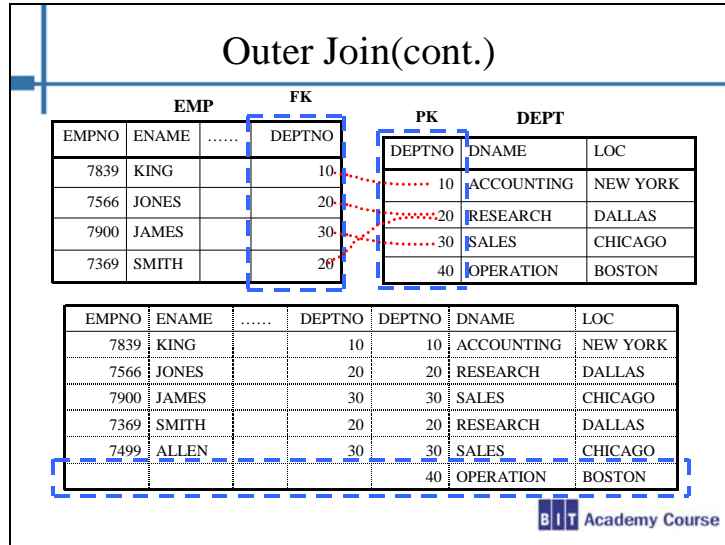
BIT Academy Course

Outer join operator는 하나의 SELECT 문 안에서 하나의 table에만 사용할 수 있으며 아래와 같은 구조를 갖는다.

```
SELECT    table.column, table.column  
FROM      table1.table2  
WHERE     table1.column(+) = table2.column;
```

(+) 연산자는 조인 조건의 필요한 곳에 붙이도록 한다. 어느 쪽에 붙이느냐 에 따라 의미가 변하므로 올바른 위치에 붙여야 한다.

위의 outer join 경우, table1에 matching 되는 row가 없는 table2의 모든 row에 대해서 Oracle 은 table1의 select list 값들을 모두 NULL로 return해준다.



다음 두 문장의 차이점을 설명하시오.

```
SQL> conn testxx/testxx
SQL> SELECT e.id, e.last_name, c.name FROM s_emp e, s_customer c
      WHERE e.id = c.sales_rep_id (+) ORDER BY e.id;
SQL> SELECT e.id, e.last_name, c.name FROM s_emp e, s_customer c
      WHERE e.id (+) = c.sales_rep_id ORDER BY e.id;
```

다음은 Outer join의 실습 예이다. 실행 결과를 비교해 보시오.

```
SQL> conn scott/tiger
SQL> SELECT ename, dname FROM emp e, dept d WHERE e.deptno (+)= d.deptno;
SQL> SELECT ename, dname FROM dept LEFT OUTER JOIN emp USING (deptno);
(* SQL:1999 Syntax)
SQL> INSERT INTO emp (empno, ename) VALUES (4444,user);
SQL> SELECT * FROM emp;
SQL> SELECT ename, dname FROM emp e, dept d WHERE e.deptno = d.deptno(+);
SQL> SELECT ename, dname FROM dept RIGHT OUTER JOIN emp USING (deptno);
(* SQL:1999 Syntax)
SQL> SELECT ename, dname FROM emp e, dept d
      WHERE e.deptno(+) = d.deptno(+);
SQL> SELECT ename, dname FROM dept FULL OUTER JOIN emp USING (deptno);
(* SQL:1999 Syntax)
```

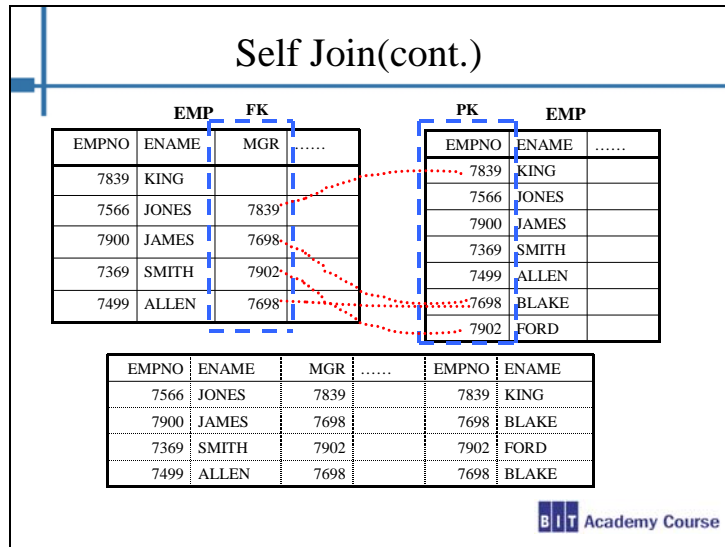
## Self Join

- A self join is a join of a table to itself.
- This table appears twice in the FROM clause and is followed by table aliases that qualify column names in the join condition.

```
SELECT emp.ename, mgr.ename  
FROM emp, emp mgr  
WHERE emp.mgr = mgr.empno;
```

BIT Academy Course

Self join의 경우 동일한 table 명이 2개 이상 사용되므로 반드시 table alias를 사용한다.



사원의 이름과 매니저의 이름을 출력하시오.

```
SQL> conn testxx/testxx
SQL> SELECT worker.last_name || ' works for ' || manager.last_name
FROM s_emp worker, s_emp manager
WHERE worker.manager_id=manager.id;
```

모든 사원의 ID, Last\_name, Salary, 매니저의 Last\_name, 소속부서의 Name을 출력하시오

```
SQL> SELECT e.id,e.last_name, e.salary, m.last_name, d.name
FROM s_emp e, s_emp m, s_dept d
WHERE e.manager_id=m.id(+) AND e.dept_id=d.id;
```

# **Advanced Query (Aggregation)**

- **Aggregate Functions**
- **Group By**
- **Having**
- **Rollup**
- **Cube**
- **Grouping**
- **Grouping Set**



## Aggregate Function

- You can divide rows into some groups and get summary information for each group by using aggregate functions.

EMPNO	ENAME	SAL	DEPTNO
7369	SMITH	800	20
7499	ALLEN	1600	30
7521	WARD	1250	30
7566	JONES	2975	20
7654	MARTIN	1250	30
7698	BLAKE	2850	30
7782	CLARK	2450	10
7788	SCOTT	3000	20
7839	KING	5000	10
7844	TURNER	1500	30
7876	ADAMS	1100	20
7900	JAMES	950	30
...	...	...	...



*Groups  
by deptno*

DEPTNO	SAL	...
10	5000	
10	2450	
20	1100	
20	800	
20	2975	
20	3000	
30	1500	
30	950	
30	1250	
30	1600	
30	1250	
30	2850	
...	...	...

## Aggregate Function(cont.)

- Aggregate functions return a single row based on groups of rows.
- A table is treated as one large group of information

Function	Purpose
COUNT(DISTINCT ALL * <i>expr</i> )	Returns the number of rows in the query.
SUM(DISTINCT ALL <i>n</i> )	Returns sum of values of <i>n</i> .
AVG(DISTINCT ALL <i>n</i> )	Returns average value of <i>n</i> .
MAX(DISTINCT ALL <i>expr</i> )	Returns maximum value of <i>expr</i> .
MIN(DISTINCT ALL <i>expr</i> )	Returns minimum value of <i>expr</i> .
STDDEV(DISTINCT ALL <i>x</i> )	Returns standard deviation of <i>x</i> , a number.
VARIANCE(DISTINCT ALL <i>x</i> )	Returns variance of <i>x</i> , a number.

## Aggregate Function(cont.)

- Example

```
SELECT MIN(ename), MAX(ename)
FROM emp;
```

```
SELECT AVG(sal), SUM(sal)
FROM emp
WHERE deptno IN (10,20);
```

BIT Academy Course

아래의 문장들을 실행하면서 결과를 확인하시오.

```
SQL> SELECT count(*) FROM s_emp;
```

```
(* 25 rows selected)
```

```
SQL> SELECT count(commission_pct) FROM s_emp;
```

```
(* 5 rows selected)
```

```
SQL> SELECT count(DISTINCT commission_pct) FROM s_emp;
```

```
(* 4 rows selected)
```

```
SQL> SELECT sum(salary) FROM s_emp;
```

```
(* 31377)
```

```
SQL> SELECT sum(DISTINCT salary) FROM s_emp;
```

```
(* 22582)
```

```
SQL> SELECT avg(commission_pct) FROM s_emp;
```

```
(* 13)
```

```
SQL> SELECT sum(commission_pct)/count(*) FROM s_emp;
```

```
(* 2.6)
```

```
SQL> SELECT sum(commission_pct)/count(commission_pct) FROM s_emp;
```

```
(* 13)
```

```
SQL> SELECT stddev(salary), variance(salary) FROM s_emp;
```

## GROUP BY Clause

- If you use the GROUP BY clause in a SELECT statement, Oracle divides the rows of a queried table into groups.
- Oracle applies the aggregate functions in the select list to each group of rows and returns a single result row for each group.
- If you omit the GROUP BY clause, Oracle applies aggregate functions in the select list to all the rows in the queried table.

 Academy Course

Aggregate Function은 row들의 집합에 대해 연산이 이루어지는데, 이러한 row들의 집합은 table 전체이거나 table의 일부 row일 수 있다.

## GROUP BY Clause(cont.)

- Example

```
SELECT job, MAX(sal), MIN(sal)
FROM emp
GROUP BY job;
```

```
SELECT deptno, SUM(sal), AVG(sal)
FROM emp
WHERE hiredate > '81/01/01'
GROUP BY deptno
ORDER BY 2;
```

BIT Academy Course

사원의 직무별로 급여의 평균과 최대, 최소값을 구하시오.

```
SQL> SELECT title, avg(salary), max(salary), min(salary)
      FROM s_emp GROUP BY title;
```

고객을 지역(region\_id)별로 나눈 다음 다시 국가(country)별로 나누어 명수를 구하시오.

```
SQL> SELECT region_id, country, count(*)
      FROM s_customer GROUP BY region_id, country;
```

Stock Clerk 직종의 사원들의 부서별 인원수를 출력하시오.

```
SQL> SELECT dept_id, COUNT(*) "Number" FROM s_emp
      WHERE title = 'Stock Clerk' GROUP BY dept_id;
```

신용등급별로 고객의 수를 출력하시오.

```
SQL> SELECT credit_rating, COUNT(*) "#Cust" FROM s_customer
      GROUP BY credit_rating;
```

직종별 최고 급여를 급여가 많은 직종부터 출력하시오.

```
SQL> SELECT title, max(salary) FROM s_emp
      GROUP BY title ORDER BY max(salary) DESC;
```

GROUP BY절에 positional notation이나 Column alias를 사용할 수 없다.

```
SQL> SELECT region_id, country, count(*)
      FROM s_customer
      GROUP BY 1, 2;
```

(\* Error 발생)

## Incorrect Example for GROUP BY

- All elements of the select list must be expressions from the GROUP BY clause, expressions containing aggregate functions, or constants.
- Example

```
SELECT job, ename, SUM(sal)
FROM emp
GROUP BY job;          (Error)
```

BIT Academy Course

GROUP BY clause가 포함된 질의의 SELECT clause에는 aggregate function과 상수, 그리고 GROUP BY clause에 나타난 column 들만을 명시할 수 있다.

### 잘못 사용한 예

```
SQL> SELECT region_id, COUNT(name) FROM s_dept;
      SELECT region_id, COUNT(name)
      *
ERROR at line 1:
ORA-00937: not a single-group group function
SQL> SELECT region_id, country, count(*)
      FROM s_customer GROUP BY region_id;
      SELECT region_id, country, count(*)
      *
ERROR at line 1:
ORA-00979: not a GROUP BY expression
```

### 수정한 질의의 예

```
SQL> SELECT region_id, COUNT(name)
      FROM s_dept
      GROUP BY region_id;
SQL> SELECT count(*)
      FROM s_customer
      GROUP BY region_id, country;
```

## HAVING Clause

- You use aggregate functions in the HAVING clause to eliminate groups from the output based on the results of the aggregate functions.
- Location of aggregate functions

```
SELECT column, aggregate_function
FROM table
[WHERE condition]
[GROUP BY group_by_expression]
[HAVING aggregate_condition]
[ORDER BY column];
```

BIT Academy Course

위 질의의 실행 순서는 다음과 같다.

- (1) FROM clause의 table에서 WHERE clause를 만족하는 row들을 찾는다.
- (2) (1)의 결과 row들을 GROUP BY clause에 따라 grouping한다.
- (3) (2)의 결과 group들에 대해 HAVING clause를 만족하는 group들을 찾는다.
- (4) (3)의 결과 group들에 대해 SELECT list의 expression을 구한다.
- (5) (4)의 결과를 ORDER BY clause에 따라 정렬한다.

## HAVING Clause(cont.)

- Example

```
SELECT job, AVG(sal) PAYROLL
FROM emp
WHERE deptno IN (10,20)
GROUP BY job
HAVING SUM(sal) > 5000
ORDER BY AVG(sal);
```

BIT Academy Course

**사원이 3명 이상인 직종에 대해 직종별 평균 급여를 출력하시오.**

```
SQL> SELECT title, AVG(salary) FROM s_emp
      GROUP BY title HAVING COUNT(*) > 3;
SQL> SELECT title, AVG(salary) FROM s_emp
      GROUP BY title;
```

**부서별로 2명 이상 근무하고 있는 직종 만을 출력하시오.**

```
SQL> SELECT dept_id, title
      FROM s_emp
      GROUP BY dept_id, title
      HAVING COUNT(*) > 1;
```



## ROLLUP, CUBE Operators

- Deliver aggregates and super-aggregates for expressions within a GROUP BY clause by using ROLLUP or CUBE operator.

Operator	Purpose
ROLLUP	returns a single row of summary for each group. You can use the ROLLUP operation to produce subtotal values.
CUBE	the selected rows based on the values of all possible combinations of expressions for each row, and returns a single row of summary information for each group. You can use the CUBE operation to produce cross-tabulation values.

## ROLLUP Operator


- Example

```
SELECT deptno, job, sum(sal)
FROM emp
GROUP BY ROLLUP(deptno, job);
.....
```

DEPTNO	JOB	SUM(SAL)
10	CLERK	1300
10	MANAGER	2450
10	PRESIDENT	5000
10		8750
20	ANALYST	6000
20	CLERK	1900
20	MANAGER	2975
20		10875
30	CLERK	950
30	MANAGER	2850
30	SALESMAN	5600
30		9400
		29025

*Regular rows* → (points to rows with deptno and job)

*Superaggregate rows* → (points to rows with empty job field and the grand total row)



다음을 실행해 보시오.

```
SQL > SELECT dept_id, title, sum(salary)
      FROM s_emp
      GROUP BY ROLLUP(dept_id, title);
```

### CUBE Operator

- Example

```
SELECT deptno, job, sum(sal)
FROM emp
GROUP BY CUBE(deptno, job);
```

*Regular rows* →

*Superaggregate rows* →

DEPTNO	JOB	SUM(SAL)
10	CLERK	1300
10	MANAGER	2450
10	PRESIDENT	5000
10		8750
20	ANALYST	6000
20	CLERK	1900
20	MANAGER	2975
20		10875
30	CLERK	950
30	MANAGER	2850
30	SALESMAN	5600
30		9400
	ANALYST	6000
	CLERK	4150
	MANAGER	8275
	PRESIDENT	5000
	SALESMAN	5600
		29025

DB Academy Course


다음을 실습해 보시오.

```
SQL > SELECT dept_id, title, sum(salary)
      FROM s_emp
      GROUP BY CUBE(dept_id, title);
```

## GROUPING Function

- applicable only in a SELECT statement that contains a GROUP BY extension, such as ROLLUP or CUBE.
- These operations produce superaggregate rows that contain null values representing the set of all values.

Function	Purpose
GROUPING( <i>expr</i> )	The function returns a value of 1 if the value of <i>expr</i> in the row is a null representing the set of all values. Otherwise, it returns zero.



다음을 실행해 보시오.

**SQL> COL title FORMAT A30**

```
SQL> SELECT dept_id, title, sum(salary),
        GROUPING(dept_id) g_dept, GROUPING(title) g_title
        FROM s_emp GROUP BY ROLLUP (dept_id, title);
```

```
SQL> SELECT dept_id, title, sum(salary),
        GROUPING(dept_id) g_dept, GROUPING(title) g_title
        FROM s_emp GROUP BY CUBE (dept_id, title);
```

**SQL> COL dept\_id FORMAT A10**

```
SQL> SELECT DECODE(GROUPING(dept_id), 1, 'All Depts', dept_id) AS dept_id,
        DECODE(GROUPING(title), 1, 'All Titles', title) AS title,
        SUM(salary) sumsal
        FROM s_emp GROUP BY ROLLUP (dept_id, title);
```

```
SQL> SELECT DECODE(GROUPING(dept_id), 1, 'All Depts', dept_id) AS dept_id,
        DECODE(GROUPING(title), 1, 'All Titles', title) AS title,
        SUM(salary) sumsal
        FROM s_emp GROUP BY CUBE(dept_id, title);
```

## GROUPING Function(cont.)

- Example

```
SELECT DECODE(GROUPING(deptno), 1, 'All Depts',
              deptno) AS deptno,
       DECODE(GROUPING(job), 1, 'All Jobs', job)
       AS job, SUM(sal) sumsal
FROM emp GROUP BY ROLLUP (deptno, job);
```

```
SELECT DECODE(GROUPING(deptno), 1, 'All Depts',
              deptno) AS deptno,
       DECODE(GROUPING(job), 1, 'All Jobs', job)
       AS job, SUM(sal) sumsal
FROM emp GROUP BY CUBE (deptno, job);
```

**B I T** Academy Course

DEPTNO	JOB	SUMSAL
10	CLERK	1300
10	MANAGER	2450
10	PRESIDENT	5000
10	All Jobs	8750
20	ANALYST	6000
20	CLERK	1900
20	MANAGER	2975
20	All Jobs	10875
30	CLERK	950
30	MANAGER	2850
30	SALESMAN	5600
30	All Jobs	9400
All Depts	All Jobs	29025

DEPTNO	JOB	SUMSAL
10	CLERK	1300
10	MANAGER	2450
10	PRESIDENT	5000
10	All Jobs	8750
20	ANALYST	6000
20	CLERK	1900
20	MANAGER	2975
20	All Jobs	10875
30	CLERK	950
30	MANAGER	2850
30	SALESMAN	5600
30	All Jobs	9400
All Depts	ANALYST	6000
All Depts	CLERK	4150
All Depts	MANAGER	8275
All Depts	PRESIDENT	5000
All Depts	SALESMAN	5600
All Depts	All Jobs	29025

## GROUPING SETS

- Define multiple groupings by using GROUPING SETS in the GROUP BY clause.
- Example

```
SELECT deptno, job, sum(sal)
FROM emp
GROUP BY
  GROUPING SETS ((deptno,job),(job),())
ORDER BY 1,2;
```

 Academy Course

다음 두 문장을 비교해 보시오.

```
SQL> COL dept_id FORMAT 999999
```

```
SQL> SELECT dept_id, title, sum(salary)
FROM s_emp
GROUP BY GROUPING SETS ((dept_id,title),(title),())
ORDER BY 1,2;
```

```
SQL> SELECT dept_id, title, sum(salary)
FROM s_emp
GROUP BY dept_id, title
UNION ALL
SELECT NULL, title, sum(salary)
FROM s_emp
GROUP BY title
UNION ALL
SELECT NULL, NULL, sum(salary)
FROM s_emp;
```

# Advanced Query (Subquery, Set Operation)

- Subquery
  - Single-Row subquery
  - Multiple-Row subquery
  - Inline View
  - Correlated subquery
- Set 연산
  - Union
  - Union All
  - Intersect
  - Minus
- Hierarchical Data 조회
  - Start With
  - Connect By
  - Prior
  - Level

## Subquery

- A query is an operation that retrieves data from one or more tables.
- A query nested within a SELECT statement is called a subquery.
- A subquery can contain another subquery. Oracle places no limit on the level of query nesting.
- To make your statements easier for you to read, always qualify the columns in a subquery with the name or alias of the table.



### Nested Subquery


- The inner query is executed before the main query.
- The result of the inner query is used by the main query.
- Example

```
SELECT ename
FROM emp
WHERE deptno
```

=

```
(SELECT deptno
FROM emp
WHERE ename = 'SCOTT')
```

*Main query**Inner query*



Subquery의 특징은 다음과 같다.

- subquery는 괄호로 묶어야 한다.
- subquery는 연산자의 오른쪽에 나타나야 한다.
- subquery는 여러 SQL 문장에서 사용 가능하다.
- subquery 실행 결과 row의 개수에 따라 operator의 종류를 맞게 써준다.

## Single-Row Subquery

- A single-Row subquery returns only one row from the inner query.
- A single-Row subquery uses single-row operators such as =, >, <, >=, <=.
- Example

```
SELECT ename, sal
FROM emp
WHERE sal < (SELECT AVG(sal)
             FROM emp);
```

BIT Academy Course

만약 한 개 이상의 값을 돌려주는 Subquery를 single-row operator로 비교하면 error가 발생한다. Subquery를 사용할 때는 리턴될 수 있는 값의 개수를 정확히 예측하고 그에 맞는 연산자를 사용해야 한다.

- Operator를 잘못 사용한 Subquery

```
SQL> SELECT last_name, salary, dept_id
      FROM s_emp
      WHERE last_name = (SELECT MIN(last_name)
                        FROM s_emp GROUP BY dept_id);
      WHERE last_name = (SELECT MIN(last_name)
                        *
                        ERROR at line 3:
                        ORA-01427: single-row subquery returns more than one row
```

- 수정한 Subquery

```
SQL> SELECT last_name, salary, dept_id
      FROM s_emp
      WHERE last_name IN(SELECT MIN(last_name)
                        FROM s_emp GROUP BY dept_id);
```

## Multiple-Row Subquery

- A multiple-row subquery returns more than one values from the inner query and must use multiple-row operators such as IN, ANY, ALL or EXISTS.
- Example

```
SELECT ename, sal
FROM emp
WHERE deptno IN (SELECT deptno
                  FROM emp
                  WHERE ename = 'SCOTT');
```

BIT Academy Course

각 부서별로 최고급여를 받는 사원을 출력하시오.(Nested Subquery 활용)

```
SQL> SELECT dept_id, id, last_name, salary
FROM s_emp
WHERE(dept_id,salary) IN(SELECT dept_id, max(salary)
FROM s_emp GROUP BY dept_id);
```

## Subquery in the FROM Clause

- You can use a subquery in the FROM clause.
- A subquery used in the FROM clause is called inline view.
- Example

```
SELECT e.ename, e.sal
FROM emp e, (SELECT deptno, MAX(sal) msal
             FROM emp GROUP BY deptno) g
WHERE e.deptno=g.deptno AND e.sal = g.msal;
```

BIT Academy Course

**각 부서별로 최고급여를 받는 사원을 출력하시오.(Inline View 활용)**

```
SQL> SELECT e.dept_id, e.id, e.last_name, e.salary
FROM s_emp e, (SELECT s.dept_id, max(s.salary) msal
FROM s_emp s GROUP BY dept_id) m
WHERE e.dept_id = m.dept_id AND e.salary = m.msal;
```

**급여를 많이 받는 순서대로 상위 3명을 출력하시오.**

```
SQL> SELECT rownum, last_name, salary
FROM (SELECT * FROM s_emp ORDER BY salary DESC)
WHERE rownum < 4;
```

## Subquery in the HAVING Clause

- The result of the inner subquery is used by the HAVING clause.
- Example

```
SELECT deptno, AVG(sal)
FROM emp
GROUP BY deptno
HAVING AVG(sal) > (SELECT AVG(sal)
                   FROM emp
                   WHERE deptno = 20);
```

## Correlated Subquery

- A subquery references a column from a table referred to in the parent statement is called correlated subquery.
- A correlated subquery is evaluated once for each row processed by the parent statement.
- Example

```
SELECT ename, sal
FROM emp e
WHERE e.sal = (SELECT MAX(sal)
               FROM emp
               WHERE deptno = e.deptno);
```

BIT Academy Course

Outer query의 table과 연결된 subquery를 Co-related Subquery 라고 하며 다음의 순서로 실행된다.

- (1) Outer query의 table에서 row를 하나 읽는다.
- (2) (1)에서 읽은 row의 data를 이용하여 Subquery를 수행한다.
- (3) (2)의 결과값으로 Outer query의 WHERE clause를 평가하여 row의 선택여부를 결정한다.
- (4) Outer query의 table에 row가 없을 때까지 (1) - (3)을 반복 수행한다.

**각 부서별로 최고급여를 받는 사원을 출력하시오. (Co-related Subquery 활용)**

```
SQL> SELECT dept_id, id, last_name, salary
      FROM s_emp e
      WHERE e.salary = (SELECT max(salary)
                        FROM s_emp WHERE dept_id=e.dept_id);
```

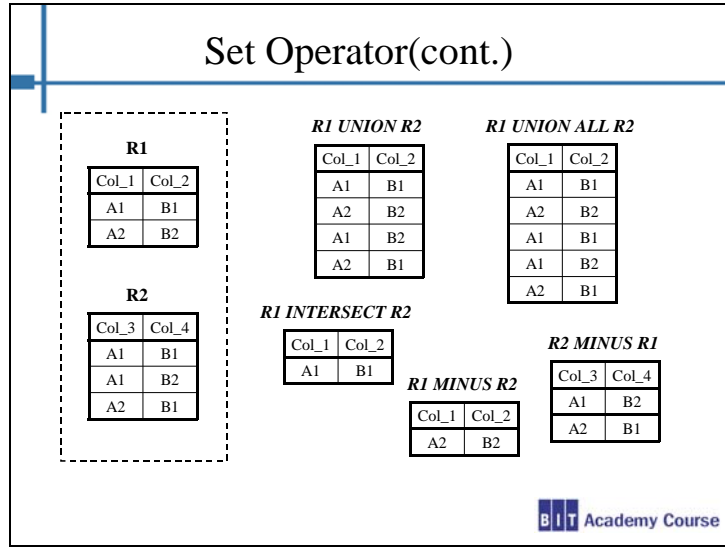
**사원이 한명이라도 있는 부서명을 출력하시오.**

```
SQL> SELECT name FROM s_dept d
      WHERE EXISTS(SELECT 1 FROM s_emp WHERE dept_id = d.id);
```

## Set Operator

- All set operators have equal precedence.

Operator	Purpose
UNION	All rows selected by either query.
UNION ALL	All rows selected by either query including duplicates.
INTERSECT	All distinct rows selected by both queries.
MINUS	All distinct rows selected by the first query but not the second.



SET Operation의 결과와 아래 Cartesian Product의 결과를 비교하시오.

R1 X R2

Col_1	Col_2	Col_3	Col_4
A1	B1	A1	B1
A1	B1	A1	B2
A1	B1	A2	B1
A2	B2	A1	B1
A2	B2	A1	B2
A2	B2	A2	B1

다음을 실행해 보시오. (5-14 실습 반복)

```
SQL> SELECT dept_id, title, sum(salary) FROM s_emp
      GROUP BY GROUPING SETS ((dept_id,title),(title),())
      ORDER BY 1,2;
```

```
SQL> SELECT dept_id, title, sum(salary) FROM s_emp GROUP BY dept_id, title
      UNION ALL
      SELECT NULL, title, sum(salary) FROM s_emp GROUP BY title
      UNION ALL
      SELECT NULL, NULL, sum(salary) FROM s_emp;
```



## Set Operator(cont.)

- The corresponding expressions in the select lists of the component queries of a compound query must match in number and data type.
- Example

```
SELECT ename FROM emp
UNION
SELECT dname FROM dept;
```

BIT Academy Course

회사의 사원이름과 고객이름을 출력하시오.

```
SQL> SELECT last_name NAME FROM s_emp
      UNION
      SELECT name FROM s_customer;
SQL> SELECT last_name NAME FROM s_emp
      UNION ALL
      SELECT name FROM s_customer;
```

고객을 담당하고 있는 사원의 사번과 이름을 출력하시오.

```
SQL> SELECT id, last_name FROM s_emp
      WHERE id IN (SELECT id FROM s_emp
                  INTERSECT
                  SELECT sales_rep_id FROM s_customer);
```

고객을 담당하고 있지 않은 사원의 사번과 이름을 출력하시오.

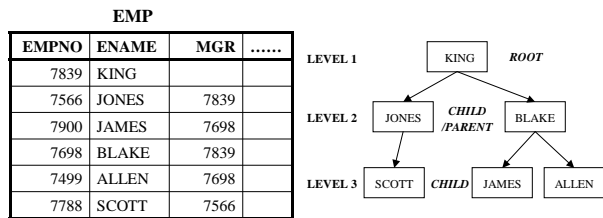
```
SQL> SELECT id, last_name FROM s_emp
      WHERE id IN (SELECT id FROM s_emp
                  MINUS
                  SELECT sales_rep_id FROM s_customer);
```

집합연산에서는 대응되는 data의 개수가 같고 type이 유사해야 한다.

```
SQL> SELECT id FROM s_emp
      UNION
      SELECT name FROM s_customer;
(* Error 발생)
```

## Hierarchical Query

- If a table contains hierarchical data, you can select rows in a hierarchical order using the hierarchical query clause.



## Hierarchical Query(cont.)

- You should use START WITH, CONNECT BY clauses and PRIOR operator.
- For each row returned by a hierarchical query, the LEVEL pseudo column returns 1 for a root node, 2 for a child of a root, and so on.
- Example

```
SELECT level, ename
FROM emp
START WITH mgr IS NULL
CONNECT BY PRIOR empno = mgr
ORDER BY level;
```

BIT Academy Course

### START WITH :

첫 번째로 조회할 row의 condition을 명시하며, 첫 번째로 조회된 row들의 level은 1이 되며 root node가 된다. 즉, mgr IS NULL 이라는 condition은 회사의 사장을 조회하는 것이다.

### CONNECT BY :

다음 번에 조회할 row의 조건을 명시한다. PRIOR operator는 직전 level의 row를 의미한다. 즉, PRIOR empno = mgr 이란 전 level의 empno가 다음 찾을 row의 mgr과 같은row들을 찾는 것이다.

이와 같은 Hierarchical query는 더 이상 CONNECT BY clause를 만족하는 다음 level의 row들이 존재하지 않을 때까지 진행된다.

다음은 level에 따라 data를 indentation 하는 문장이다. 실행해 보시오.

```
SQL> SELECT RPAD(' ',level*2,' ')||last_name Employee
FROM s_emp
START WITH manager_id IS NULL
CONNECT BY PRIOR id = manager_id;
```

# **SQL\*Plus Commands**

- **Spool**
- **System Variables**
- **Column**
- **Break**
- **Substitution Variables**
- **Bind Variables**

## SQL and SQL\*Plus

- SQL
  - SQL stands for Structured Query Language.
  - A standard statement sending to Oracle for needed data.
  - SQL is statement for definition, manipulation and control of database.
- SQL\*Plus
  - SQL\*Plus is a easy tool made by Oracle.
  - SQL\*Plus enables you to manipulate SQL commands and PL/SQL blocks, and to perform many additional tasks as well.

## SPOOL Command

- SQL\*Plus stores all information displayed on the screen after you enter the SPOOL command in the file you specify.

SPO[OL] [filename[.ext]]|OFF|OUT

BIT Academy Course

SQL\*Plus 등의 화면에 출력되는 내용을 파일로 저장하기 위해서 SPOOL 명령을 사용한다.

```
SQL> spool res.txt
SQL> SELECT * FROM s_emp;
SQL> spool off
SQL> ed res.txt
```

SPOOL 명령을 할 때 파일명의 확장자를 따로 주지 않으면 디폴트로 .lst가 붙게 된다.

SPOOL OFF 명령까지의 화면 내용을 파일에 기록한다.

SPOOL OUT 명령은 기록된 파일의 내용을 시스템의 기본 프린터로 보내 출력하게 한다.

## Set Command

- SET command
  - Controls environments of the current session.

`SET system_variable value`
- LOGIN.sql
  - Use this file to set up your SQL\*Plus environment and reuse those settings with each session.
  - Whenever SQL\*Plus is started, it automatically searches for your LOGIN file and runs the commands.
  - LOGIN.sql file can contain any SQL commands, PL/SQL blocks, or SQL\*Plus commands.

BIT Academy Course

client program을 작동시켜 Oracle과 connection을 맺게 되면 하나의 session이 시작된다. Client program을 수행 종료하거나 connection을 끊게 되면 session은 끝나게 된다. SQL\*Plus 도 client program의 하나이다.

SQL\*Plus를 구동할 때마다 적용하고 싶은 내용을 login.sql 파일에 작성해 둔다. login.sql 파일의 위치는 SQL\*Plus icon의 시작위치이다.

SQL\*Plus의 시작위치에 아래와 같이 login.sql 파일 작성한 뒤 SQL\*Plus를 종료하고 다시 띄워 차이 점을 확인한다. 테스트 뒤에 원상 복구한다.

```
SQL> ed login.sql
SET TIME ON
```

System Variables	
Variable	Description
ARRAY[SIZE] {15  <i>n</i> }	Sets the number of rows--called a <i>batch</i> --that SQL*Plus will fetch from the database at one time. Valid values are 1 to 5000.
AUTO[COMMIT] {OFF ON IMM[EDIATE]  <i>n</i> }	Controls when Oracle commits pending changes to the database.
AUTOT[RACE] {OFF ON TRACE[ONLY]} [EXP[LAIN]] [STAT[ISTICS]]	Displays a report on the execution of successful SQL DML statements (SELECT, INSERT, UPDATE or DELETE). The report can include execution statistics and the query execution path.
COLSEP {_  <i>text</i> }	Sets the text to be printed between SELECTed columns.
ECHO {OFF ON}	Controls whether the START command lists each command in a command file as the command is executed.

 Academy Course

다음은 실습해 보시오.

```
SQL> SET COLSEP *
SQL> SELECT * FROM s_dept;
SQL> SET COLSEP " "
SQL> /
```



## System Variables(cont.)

Variable	Description
FEED[BACK] { <u>6</u> <i>n</i>  OFF ON}	Displays the number of records returned by a query when a query selects at least <i>n</i> records.
HEA[DING] {OFF  <u>ON</u> }	Controls printing of column headings in reports
LIN[ESIZE] { <u>80</u> <i>n</i> }	Sets the total number of characters that SQL*Plus displays on one line before beginning a new line.
LONG { <u>80</u> <i>n</i> }	Sets maximum width (in bytes) for displaying LONG, CLOB and NCLOB values.
PAGES[IZE] { <u>24</u> <i>n</i> }	Sets the number of lines in each page.
PAU[SE] { <u>OFF</u>  ON  <i>text</i> }	Allows you to control scrolling of your terminal when running reports.

BIT Academy Course

다음은 실습해 보시오.

```
SQL> SELECT * FROM s_dept;
(* 12 개의 행이 선택되었습니다. => FEEDBACK 메시지 확인)
SQL> SET FEEDBACK OFF
SQL> /
(* FEEDBACK 메시지 없음)
SQL> SET FEEDBACK ON
SQL> /
(* 12 개의 행이 선택되었습니다. => FEEDBACK 메시지 확인)
SQL> SET FEEDBACK 15
SQL> /
(* FEEDBACK 메시지 없음)
SQL> SET HEADING OFF
SQL> /
SQL> SELECT * FROM s_emp;
SQL> SET LINESIZE 60
SQL> /
SQL> SET LINESIZE 80
```

System Variables(cont.)	
Variable	Description
SERVEROUT[PUT] {OFF ON}	Controls whether to display the output (that is, DBMS_OUTPUT.PUT_LINE) of stored procedures or PL/SQL blocks in SQL*Plus.
SQLP[ROMPT] {SQL> text}	Sets the SQL*Plus command prompt.
SUF[IX] {SQL text}	Sets the default file extension that SQL*Plus uses in commands that refer to command files.
TI[ME] {OFF ON}	Controls the display of the current time.
VER[IFY] {OFF ON}	Controls whether SQL*Plus lists the text of a SQL statement or PL/SQL command before and after SQL*Plus replaces substitution variables with values.

다음은 실습해 보시오.

```
SQL> SET PAUSE ON
SQL> SELECT last_name FROM s_emp;
SQL> SET PAGESIZE 24
SQL> /
SQL> SET PAUSE OFF
SQL> /
SQL> SET SQLPROMPT Testxx>
Testxx> SET TIME ON
09:40:50 Testxx> SET TIME OFF
Testxx> SET SQLPROMPT "SQL> "
```

## System Variables(cont.)

- Option -> Environment
  - You can check and set the system variables with SQL\*Plus windows menu.
- **SHOW *system\_variable***
  - Represents any system variable set by the SET command.
- **SHOW ALL**
  - Lists the settings of all SHOW options.

BIT Academy Course

다음은 실습해 보시오.

```
SQL> SHOW HEADING
heading OFF
SQL> SET HEADING ON
SQL> SHOW HEADING
heading ON
SQL> SHOW ALL
```

## COLUMN Command

- Specify display attributes for a given column.
- COL[UMN] [{Column|alias} *[option]*]

Option	Purpose
CLE[AR]	Resets the display attributes for the column to default values
FOR[MAT] <i>format</i>	Specifies the display format of the column
HEA[DING] <i>text</i>	Defines a column heading
JUS[TIFY]{L[EFT] C[ENTER] C[ENTRE] R[IGHT]}	Aligns the heading
NUL[L] <i>text</i>	Controls the text SQL*Plus displays for null values in the given column
ON/OFF	Controls the status of display attributes for a column

BIT Academy Course

다음은 실습해 보시오.

```
SQL> COLUMN last_name HEADING 'Employee|Name' FORMAT A15
SQL> SELECT last_name, salary FROM s_emp;
(* last_name column의 heading이 변경됨)
SQL> COL last_name FORMAT A5
SQL> /
(* last_name column의 폭이 좁아 다음 줄까지 디스플레이 됨)
SQL> COL last_name CLEAR
SQL> /
(* last_name column의 heading과 폭이 원상 복구됨)
SQL> COL last_name JUSTIFY RIGHT
SQL> COL salary JUSTIFY CENTER
SQL> /
(* column heading의 위치가 변경됨)
SQL> COL commission_pct NULL '***'
SQL> SELECT last_name, commission_pct FROM s_emp;
(* column 값이 NULL인 부분에 ***표가 나타남.)
SQL> COL commission_pct OFF
SQL> /
(* column 값이 NULL인 부분에 초기 설정대로 space가 나타남.)
```

## COLUMN Command(cont.)

- Format for Character and Date
  - To change the width of a character or date data type to *n*.
  - Example : COLUMN column FORMAT A10
- Format for Number

Element	Description
9	Number of "9"s specifies number of significant digits returned
0	Displays a leading zero
\$	Prefixes value with dollar sign
L	Displays the local currency symbol in this position.
,	Displays a comma in this position.
.	Displays a period (decimal point) in this position

 BIT Academy Course

다음을 실습해 보시오.

```
SQL> COLUMN salary JUSTIFY LEFT FORMAT $99,990.00
```

```
SQL> SELECT last_name, salary FROM s_emp;
```

(\* salary 값의 표현과 heading 위치가 변경됨)

```
SQL> COLUMN salary FORMAT L099,990.00
```

```
SQL> /
```

(\* salary 값의 표현이 변경됨)

```
SQL> COLUMN salary FORMAT 990
```

```
SQL> /
```

(\* salary 값이 #####로 표현된 것이 많음. 결과 값이 format에서 제공하는 자리 수를 초과하면 # 문자열을 출력한다.)

```
SQL> COLUMN salary FORMAT $99,990.00
```

```
SQL> /
```

(\* salary column의 data가 다 보임)

## COLUMN Command(cont.)

- Listing and Resetting Column display attributes.

Command	Purpose
COLUMN <i>column_name</i>	To list the current display attributes for a given column
COLUMN	To list the current display attributes for all columns
COLUMN <i>column_name</i> CLEAR	To reset the display attributes for a column to their default values
CLEAR COLUMNS	To reset the attributes for all columns

BIT Academy Course

다음은 실습해 보시오.

```
SQL> COLUMN
      COLUMN salary ON
      FORMAT $99,990.00

      COLUMN last_name ON
      JUSTIFY right
SQL> COLUMN last_name
      COLUMN last_name ON
      JUSTIFY right
SQL> COLUMN last_name CLEAR
SQL> COLUMN
      COLUMN salary ON
      FORMAT $99,990.00
SQL> CLEAR COLUMNS
columns 소거되었습니다.
```

## BREAK Command

- To suppress duplicate values by default in the column or expression.

– BRE[AK] [ON *break\_column*]

– Example :

```
BREAK ON title SKIP 1 ON REPORT
```

– BREAK

– CLEAR BREAKS

BIT Academy Course

Break의 효과를 갖기 위해서는 SELECT 문장 작성시 반드시 ORDER BY clause에 Break column을 써주도록 한다.

```
SQL> SELECT dept_id, last_name, salary FROM s_emp;
```

```
SQL> BREAK ON dept_id SKIP 1
```

```
SQL> /
```

(\* 주로 줄바꿈 효과만 나타남, 43번 부서 정보 중복 제거됨)

```
SQL> SELECT dept_id, last_name, salary FROM s_emp
        ORDER BY dept_id;
```

(\* 각 부서별로 정렬된 data에 대해 중복값 제거 및 줄바꿈 효과가 나타남)

```
SQL> BREAK
```

```
break on dept_id skip 1 nodup
```

```
SQL> BREAK ON dept_id SKIP 1 ON REPORT
```

```
SQL> BREAK
```

```
break on report nodup
```

```
on dept_id skip 1 nodup
```

```
SQL> CLEAR BREAKS
```

## Substitution Variables

- user variable name preceded by one or two ampersands (&).
- You can use substitution variables anywhere in SQL and SQL\*Plus commands.
- When SQL\*Plus encounters an undefined substitution variable in a command, SQL\*Plus prompts you for the value.



## Substitution Variables(cont.)

- Example

```
SQL> SET VERIFY ON
SQL> SELECT &select_list
      2 FROM &table;
Enter value for select_list: ename, sal
old 1: SELECT &select_list
new 1: SELECT ename, sal
Enter value for table: emp
old 2: FROM &table
new 2: FROM emp
```

BIT Academy Course

substitution 변수를 사용하여 data 조회하는 명령 파일을 작성해 보시오.

```
SQL> ed sub1
SET VERIFY ON
SELECT &select_list
FROM &table;
SET VERIFY OFF
SQL> @sub1
Enter value for select_list: last_name, salary
old 1: SELECT &select_list
new 1: SELECT last_name, salary
Enter value for table: s_emp
old 2: FROM &table
new 2: FROM s_emp
```

명령 파일을 실행하면서 argument를 이용하여 substitution 변수에 값을 전달할 수 있다.

```
SQL> ed sub1
SET VERIFY ON
SELECT &1
FROM &2;
SET VERIFY OFF
SQL> @sub1 "last_name, salary" s_emp
```

## Substitution Variables(cont.)

- use PROMPT and ACCEPT to customize the prompts for values SQL\*Plus automatically generates for substitution variables.

```
ACCEPT VariableName [Datatype]
[FORMAT] [PROMPT text] [HIDE]
```

BIT Academy Course

substitution 변수를 사용하여 data 조회하는 명령 파일을 작성해 보시오.

```
SQL> ed sub2
SET VERIFY ON
ACCEPT select_list PROMPT '조회할 칼럼 : '
ACCEPT table PROMPT '조회할 테이블 : '
SELECT &select_list
FROM &table;
SET VERIFY OFF
SQL> @sub2
조회할 칼럼 : last_name, salary
조회할 테이블 : s_emp
old 1: SELECT &select_list
new 1: SELECT last_name, salary
old 2: FROM &table
new 2: FROM s_emp
```

substitution 변수에 값을 지정해 보시오.

```
SQL> ACCEPT p_sal NUMBER PROMPT 'Salary : '
SQL> ACCEPT pswd CHAR PROMPT 'Password : ' HIDE
```

## Define User Variables

- You can define user variables for repeated use in a single command file.

```
DEFINE VariableName = Value
DEFINE VariableName
DEFINE
UNDEFINE VariableName
```

BIT Academy Course

User 변수를 선언해 보시오.

```
SQL> DEFINE p_dname = sales
SQL> DEFINE p_dname
SQL> ed sub3
SET ECHO OFF
SET VERIFY OFF
SELECT *
FROM s_dept
WHERE name = '&p_dname';
SQL> @sub3
SQL> UNDEFINE p_dname
SQL> DEFINE p_d_dname
```

\_editor 변수의 값을 변경해 보시오.

```
SQL> DEFINE _editor
(* 메모장이 연결되어 있음)
SQL> DEFINE _editor=write.exe
SQL> ed sub3
(* 워드패드 실행됨)
SQL> DEFINE _editor = notepad.exe
```

## Declare Bind Variables

- Declares a bind variable that can then be referenced in PL/SQL.
- Bind variables may be used as parameters to stored procedures, or may be directly referenced in anonymous PL/SQL blocks.

```
VARIABLE Name [NUMBER |  
              CHAR | CHAR(n) | VARCHAR2(n)]
```

 Academy Course

Bind 변수를 선언해 보시오.

```
SQL> VARIABLE a NUMBER
```

```
SQL> VARIABLE a
```

```
SQL> VARIABLE
```

## Declare Bind Variables(cont.)

- To display the value of a bind variable created with VARIABLE, use the PRINT command.

```
PRINT VariableName
```

BIT Academy Course

Bind 변수를 값을 출력해 보시오.

```
SQL> PRINT a
```

```
SQL> BEGIN :a := 10; END;
```

```
/
```

```
SQL> PRINT a
```

# Create Table

- **Table 생성**
  - CREATE TABLE
  - Subquery를 이용한 생성
- **Column의 data type과 특징**
  - CHAR, VARCHAR2, NUMBER, DATE
  - ROWID, ROWNUM
- **Constraint의 정의**
  - 정의 방법 : Column 레벨, Table 레벨
  - 종류 : PRIMARY KEY, UNIQUE, FOREIGN KEY, CHECK, NOT NULL
- **Data Dictionary**
  - DICTIONARY VIEW의 종류
  - Dynamic Script 작성

## Schema Objects

- A schema is a collection of schema objects.
- In Oracle, associated with each database user is a *schema*.
- Examples of schema objects include tables, views, sequences, synonyms, indexes, clusters, database links, snapshots, procedures, functions, and packages.

BIT Academy Course

### 사용자(User) :

database에 connect 하여 object들을 접근하기 위해 생성된 것.

### 스키마(Schema) :

특정 user 소유로 만들어진 object들의 집합.

## Create Table

- Tables are the basic unit of data storage in an Oracle database.
- You should define a table with a table name and set of columns.
- To create a relational table in your own schema, you must have CREATE TABLE system privilege and either space quota on the tablespace or UNLIMITED TABLESPACE system privilege.

BIT Academy Course

Oracle은 Database의 공간을 Tablespace 라는 논리적인 공간으로 분할하여 관리한다.

SYSTEM 이라는 이름의 Tablespace는 Database 생성시 항상 만들어지는 기본 공간으로 sys 사용자 소유의 Dictionary table들이 저장된다.



## Create Table(cont.)

- Syntax

```
CREATE TABLE [schema.]table_name
  (column datatype [DEFAULT expr]
    [column_constraints],
    ..... ,
    [table_constraints] );
```

- schema : owner user
- datatype : column data type
- column\_constraints, table\_constraints : Integrity constraints

BIT Academy Course

Table을 생성하려면 데이터 정의어(DDL)명령 중 하나인 CREATE TABLE명령을 사용한다.

Table을 만들기 위해서는 CREATE TABLE 권한과 객체를 생성할 수 있는 저장장소가 있어야 한다.

DBA는 user에게 권한을 주는 데이터 조작어(DCL) 명령을 사용한다.

다른 user의 table을 참조할 수 있으려면 참조되는 테이블은 동일한 database에 있어야 한다.

참조되는 table이 제약조건을 만드는 user의 소유가 아니라면 소유자의 이름(Schema)이 제약조건에서 참조되는 table 이름 앞에 붙여져야 한다.

DEFAULT 옵션을 이용하여 column의 default값을 지정할 수 있다. 새로운 row 입력 시 해당 column에 값이 입력되지 않으면 default옵션에 의해 설정된 값이 입력된다.

### Create Table(cont.)

- Example

```
CREATE TABLE cre_tab1
( db_user VARCHAR2(30) DEFAULT USER,
  issue_date DATE DEFAULT SYSDATE,
  type_operation NUMBER(3));
```
- Default : Specify a default value for a column during an insert.

BIT Academy Course

cre\_tab1이라는 이름의 table을 생성하시오.

```
SQL> conn testxx/testxx
SQL> CREATE TABLE cre_tab1
      (db_user VARCHAR2(30) DEFAULT USER,
       issue_date DATE DEFAULT SYSDATE,
       type_operation NUMBER(3));
```

tab이라는 Dictionary를 조회하여 생성된 table을 확인하시오.

```
SQL> SELECT * FROM tab;
(* 현재 user가 소유하고 있는 table 리스트가 출력됨. cre_tab1 테이블명이 CRE_TAB1로 보임.)
```

SQL문장의 Object 이름은 case insensitive하다.

```
SQL> SELECT * FROM cre_tab1;
SQL> SELECT * FROM Cre_Tab1;
(* FROM 절에 대소문자를 섞어도 같은 table을 조회한다.)
```

## Naming Tables and Columns

- Names must begin with a letter and can be 30 characters at a maximum size.
- Names must contain only A-Z, a-z, 0-9, \_, \$, #.
- Names must not be the same name of another objects owned by the same user.
- Names must not be an Oracle reserved word.

BIT Academy Course

대소문자를 섞어 table 명을 만들고자 할 때는 ` `로 묶어준다.

```
SQL> CREATE TABLE "Cre_tab2"(a NUMBER, b CHAR);
```

```
SQL> SELECT * FROM cre_tab2;
```

```
SELECT * FROM cre_tab2
```

```
      *
```

```
ERROR at line 1:
```

```
ORA-00942: table or view does not exist
```

(\* Error가 발생하는 이유를 적으시오.)

```
SQL> SELECT * FROM tab;
```

(\* Cre\_tab2 테이블 명이 대소문자 구별되어 보임.)

```
SQL> SELECT * FROM "Cre_tab2";
```

선택된 레코드가 없습니다.

Oracle은 ROWID 라는 Pseudocolumn을 제공한다.

```
SQL> SELECT rowid FROM s_dept;
```

(\* ROWID는 Oracle DB에 저장되는 모든 row의 물리적 위치값으로 '000000FFBBBBBBSSSS' 구조를 갖는다.)

Oracle은 ROWNUM 이라는 Pseudocolumn을 제공한다.

```
SQL> SELECT rownum, name FROM s_dept;
```

(\* ROWNUM은 WHERE 조건을 만족하는 row의 순서 값을 갖는다.)

Column Data Types		
Data type	Description	Length/Default
NUMBER( <i>p,s</i> )	Variable-length numeric data. Maximum precision <i>p</i> and/or scale <i>s</i> is 38.	Variable for each row and the maximum space for a column is 21 bytes. Default is 38.
CHAR( <i>size</i> )	Fixed-length character data of length <i>size</i> bytes.	Fixed for every row and maximum size is 2000 bytes. Default size is 1 byte.
VARCHAR2( <i>size</i> )	Variable-length character data.	Variable for each row, up to 4000 bytes per row. A maximum size must be specified.
DATE	Fixed-length date and time data, ranging from January 1, 4712 BCE to December 31, 9999 CE ("A.D.")	Fixed at 7 bytes for each row in the table. Default format is a string (such as RR/MM/DD) specified by NLS_DATE_FORMAT parameter.

BIT Academy Course

그 밖에도 아래와 같은 column의 data type이 있다.

Data type	Description
NCHAR( <i>size</i> )	national character set에 따라 결정되는 <i>size</i> 만큼의 고정길이 character data로 최대 2000byte까지 가능. 디폴트는 1 character.
NVARCHAR2 ( <i>size</i> )	national character set에 따라 결정되는 <i>size</i> 만큼의 가변길이 character data로 최대 4000 byte까지 가능하며 반드시 길이를 정해 주어야 함.
LONG	가변 길이 character data로 최대 2 gigabyte까지 가능.
RAW ( <i>size</i> )	가변 길이 raw binary data로 최대 2000 까지 가능하며 반드시 길이를 주어야 함.
LONG RAW	가변길이 raw binary data로 최대 2 gigabyte까지 가능.
BLOB	Binary data로 4 gigabyte까지 가능.
CLOB	Single-byte character data로 4 gigabyte까지 가능.
NCLOB	national character set까지 포함한 모든 character data로 4 gigabyte까지 가능.
BFILE	외부 파일로 저장된 binary data로 4 gigabyte까지 가능.
ROWID	Row의 물리적 주소를 나타내는 binary data로 extended rowid 는 10 byte, restricted rowid는 6 byte 길이.
TIMESTAMP	Date값을 미세한 초 단위까지 저장. NLS_TIMESTAMP_FORMAT 형식으로 처리.
INTERVAL YEAR TO MONTH	두 datetime 값의 차이에서 YEAR와 MONTH값 만큼 저장.
INTERVAL DAY TO SECOND	두 datetime 값의 차이를 DAY, HOUR, MINUTE, SECOND 까지 저장.

## Data Integrity

- predefined set of rules on data, as determined by the database administrator or application developer.
- Type of Data Integrity
  - Nulls
  - Unique
  - Primary Key
  - Referential Integrity
  - Complex Integrity Checking
- Most of these rules are defined using integrity constraints or database triggers in Oracle.

## Integrity Constraints

- An **integrity constraint(IC)** is a rule that restricts the values for one or more columns in a table.
- You can define integrity constraints to enforce the business rules you want to associate with the information in a database.
- Oracle uses integrity constraints to prevent invalid data entry into the base tables of the database.
- If any of the results of a DML statement execution violate an integrity constraint, Oracle rolls back the statement and returns an error.

## Oracle ICs

Constraint	Description
NOT NULL	requires a column of a table contain no null values.
UNIQUE	requires that every value in a column or set of columns be unique. No two rows of a table have duplicate values in a specified column or set of columns.
PRIMARY KEY	the unique identifier of the row. No two rows of a table have duplicate values in the specified column or set of columns and the primary key columns do not allow nulls.
FOREIGN KEY	requires that for each row of a table, the value in the foreign key matches a value in a parent key.
CHECK	enforce very specific or sophisticated integrity rules by specifying a check condition.

## Define ICs

- ICs can be defined at one of two levels.
  - Column constraint level
    - References a single column.
    - Defined within the owning column definition.
    - Any type of integrity constraints can be defined.
  - Table constraint level
    - References one or more columns.
    - Defined separately from the definitions of the columns.
    - Any type of integrity constraints can be defined except NOT NULL.



## Define ICs(cont.)

- Column constraint level

```
column [CONSTRAINT constraint_name] constraint_type
```

- Table constraint level

```
columns, ...(column definitions),  
[CONSTRAINT constraint_name] constraint_type(column,...)
```

- If you don't name a constraint, Oracle generates a name like a SYS-Cn format.

BIT Academy Course

constraint를 정의할 때 이름을 정해주지 않으면 Oracle이 SYS-Cn의 형식으로 생성해 준다.

User가 해당 constraint에 위배되는 명령을 요청했을 때, Oracle은 constraint 이름과 함께 error를 내보낸다.

## NOT NULL Constraint

- Specifies that a column cannot contain null values.
- To satisfy this constraint, every row in the table must contain a value for the column.
- This can be specified only at the column level.

```
CREATE TABLE s_dept  
(name VARCHAR2(25) NOT NULL,  
.....)
```

```
CREATE TABLE s_emp  
(last_name VARCHAR2(25)  
CONSTRAINT s_emp_last_name_nn NOT NULL,  
.....)
```

## UNIQUE Constraint

- No two rows in the table can have the same value for the unique key.
- The unique key can be composed a column or combination of columns.
- The unique key can contain nulls.
- A composite unique key cannot have more than 32 columns
- Oracle automatically creates unique index for the unique columns.

```
CREATE TABLE s_emp
(userid VARCHAR2(8)
CONSTRAINT s_emp_userid_uk UNIQUE,
.....)
```

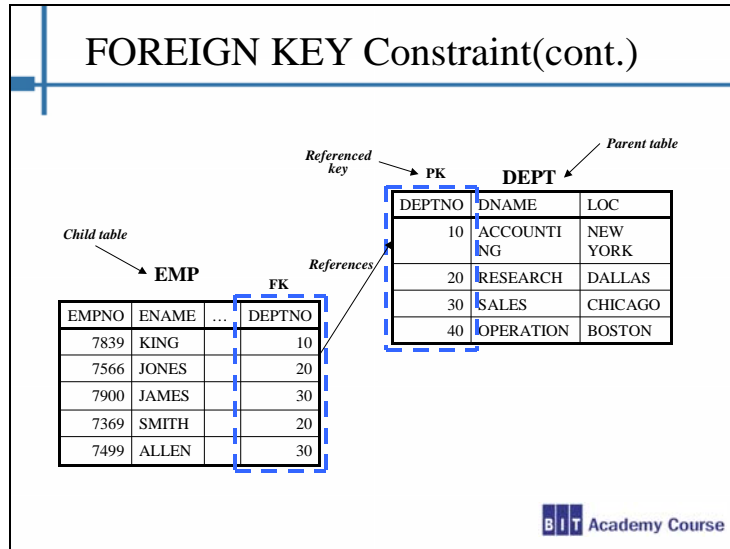
## PRIMARY KEY Constraint

- No primary key value can appear in more than one row in the table.
- A table can have only one primary key.
- No column that is part of the primary key can contain a null.
- A composite primary key cannot have more than 32 columns.
- Oracle automatically creates unique index for the primary key columns.

```
CREATE TABLE s_emp
(id NUMBER(7)
 CONSTRAINT s_emp_id_pk PRIMARY KEY,
.....)
```

## FOREIGN KEY Constraint

- The table containing the foreign key is called the child table, and the table containing the referenced key is called the parent table.
- The referenced UNIQUE or PRIMARY key constraint on the parent table must already be defined.
- The foreign key and the referenced key can be in the same table. In this case, the parent and child tables are the same.
- You can define multiple foreign keys in a table.



Referential Integrity에는 referenced value가 delete 되거나 update 될 때 dependent value를 어떻게 유지할 것인지에 따라 다음과 같은 rule의 종류가 있다.

Rule	Description
Restrict	Referenced value에 대해 delete나 update를 허락하지 않음(DML문장의 끝이나 Transaction의 끝에 check)
Set to Null	Referenced value에 대한 delete나 update를 하면서 모든 dependent value를 NULL로 설정함
Set to Default	Referenced value에 대한 delete나 update를 하면서 모든 dependent value를 Default value로 설정함
Cascade	Referenced value가 update 될 때 dependent value를 같은 값으로 설정하고, delete 될 때는 dependent row들을 함께 delete 함.
No Action	Referenced value에 대해 delete나 update를 허락하지 않음

## FOREIGN KEY Constraint(cont.)

- REFERENCES

- Designates the current column as the foreign key and identifies the parent table and the column or combination of columns that make up the referenced key.
- The corresponding columns of the referenced key and the foreign key must match in number and datatypes.

- Example

```
CREATE TABLE s_emp
  (dept_id NUMBER(7)
    CONSTRAINT s_emp_dept_id_fk
    REFERENCES s_dept(id),
    .....)
```

## FOREIGN KEY Constraint(cont.)

- ON DELETE
  - Determines how Oracle automatically maintains referential integrity if you remove a referenced primary or unique key value.
  - CASCADE specifies that Oracle removes dependent foreign key values.
  - SET NULL specifies that Oracle converts dependent foreign key values to NULL.
- Example

```
CREATE TABLE s_item
(ord_id NUMBER(7)
 CONSTRAINT s_item_ord_id_fk
 REFERENCES s_ord(id) ON DELETE CASCADE,
.....)
```



## CHECK Constraint

- To satisfy the constraint, each row in the table must make the condition either TRUE or NULL.
- The condition of a CHECK constraint can refer to any column in the table, but it cannot refer to columns of other tables.

```
CREATE TABLE s_emp
  (commission_pct NUMBER(4, 2)
   CONSTRAINT s_emp_commission_pct_ck CHECK
     (commission_pct IN (10, 12.5, 15, 17.5, 20)),
   .....)
```

## S\_DEPT Table Creation

- Example

```
CREATE TABLE s_dept
(id          NUMBER(7)
 CONSTRAINT s_dept_id_pk  PRIMARY KEY,
 name       VARCHAR2(25)
 CONSTRAINT s_dept_name_nn NOT NULL,
 region_id  NUMBER(7)
 CONSTRAINT s_dept_region_id_fk
 REFERENCES s_region (id),
 CONSTRAINT s_dept_name_region_id_uk
 UNIQUE (name, region_id) ;
```

## S\_EMP Table Creation

- Example

```
CREATE TABLE s_emp
(id          NUMBER(7)
   CONSTRAINT s_emp_id_pk PRIMARY KEY,
last_name    VARCHAR2(25)
   CONSTRAINT s_emp_last_name_nn NOT NULL,
first_name   VARCHAR2(25),
userid       VARCHAR2(8),
start_date   DATE,
comments     VARCHAR2(255),
manager_id   NUMBER(7),
title        VARCHAR2(25),
dept_id      NUMBER(7),
salary       NUMBER(11, 2),
commission_pct NUMBER(4, 2),
   CONSTRAINT s_emp_userid_uk UNIQUE (userid),
   CONSTRAINT s_emp_commission_pct_ck
   CHECK (commission_pct IN (10, 12.5, 15, 17.5, 20)));
```

## Create Tables by Subquery

- You can insert the rows returned by the subquery into the table upon its creation.
- The number of columns in the table must equal the number of expressions in the subquery.
- The column definitions can specify only column names, default values, and integrity constraints, not data types.
- You cannot define a referential integrity constraint in a CREATE TABLE statement that contains AS subquery. So, you must create the table without the constraint and then add it later with an ALTER TABLE statement.

## Create Tables by Subquery(cont.)

- Syntax

```
CREATE TABLE table_name
[column (, column...)]
AS SubQuery ;
```

- Example

```
CREATE TABLE emp_41
AS SELECT id, last_name, userid, start_date
FROM s_emp
WHERE dept_id = 41;
```

BIT Academy Course

Create table의 Column수와 Subquery의 select\_list 수가 같아야 한다.  
Subquery에 있는 기존 table에서 새로 만들어지는 table로 NOT NULL 제약조건만 상속된다.

다른 table의 구조만 복사하고 data는 가져오지 않을 때는 다음과 같이 Subquery를 활용한다.

```
SQL> CREATE TABLE history
      AS SELECT * FROM s_emp
      WHERE 1 = 0;

SQL> DESC history

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

## Data Dictionary

- One of the most important parts of an Oracle database is its *data dictionary*.
- The data dictionary is a **read-only** set of tables that provides information about its associated database.
- The data dictionary is structured in tables and views.
- A database's data dictionary consists of
  - base tables
  - user-accessible views
- The Oracle user SYS owns all base tables and user-accessible views of the data dictionary.

 Academy Course

## Contents of Data Dictionary

- the definitions of all schema objects in the database.
- how much space has been allocated for, and is currently used by the schema objects.
- default values for columns.
- integrity constraint information.
- the names of Oracle users.
- privileges and roles each user has been granted.
- auditing information, such as who has accessed or updated various schema objects.
- other general database information.

 Academy Course

**Database Data의 종류**

- Table
  - 사용자가 만들고 사용하는 table
  - 사용자가 입력한 정보를 저장
- Data Dictionary
  - Oracle 서버가 만들고 관리하는 table
  - Database에 대한 정보를 저장

### How to Use the Data Dictionary

- Oracle accesses the data dictionary to find information about users, schema objects, and storage structures.
- Oracle modifies the data dictionary every time that a data definition language (DDL) statement is issued.
- Any Oracle user can use the data dictionary as a read-only reference for information about the database.

## Data Dictionary Views

- The data dictionary consists of sets of views. a set consists of three views containing similar information and distinguished from each other by their prefixes:

Prefix	Scope
USER	user's view (what is in the user's schema)
ALL	expanded user's view (what the user can access)
DBA	database administrator's view (what is in all users' schemas)

BIT Academy Course

모든 사용자는 USER\_, ALL\_ 이름의 Dictionary view를 조회 할 수 있다.

```
SQL> SELECT * FROM user_users;
(* testxx 사용자만 확인 가능)
SQL> SELECT * FROM all_users;
```

사용자의 권한 정도에 따라 조회할 수 있는 Dictionary view의 종류가 다르다.

```
SQL> SELECT * FROM dba_users;
SELECT * FROM dba_users
*
ERROR at line 1:
ORA-00942: table or view does not exist
SQL> conn system/manager
SQL> SELECT * FROM dba_users;
```



## Query Data Dictionary

- Example

- “DICTIONARY” view has all dictionary views database users can access.

```
SELECT *
FROM DICTIONARY;
```

- Display all table names the database user owns.

```
SELECT object_name
FROM user_objects
WHERE object_type = 'TABLE';
```

“DICTIONARY” view를 이용하여 적절한 Dictionary를 찾도록 한다.

```
SQL> conn testxx/testxx
SQL> SELECT table_name FROM dictionary
      WHERE table_name LIKE '%USER%';
SQL> SELECT table_name FROM dictionary
      WHERE table_name LIKE '%PRIV%';
SQL> SELECT table_name FROM dictionary
      WHERE table_name LIKE '%TABLE%'
      OR table_name LIKE '%COLUMN%';
SQL> SELECT table_name FROM dictionary
      WHERE table_name LIKE '%CONS%';
SQL> SELECT table_name FROM dictionary
      WHERE table_name LIKE '%AUDIT%';
SQL> SELECT table_name FROM dictionary
      WHERE table_name LIKE '%IND%';
```

## Query Data Dictionary(cont.)

- Views related to constraints
  - USER\_CONSTRAINTS
  - USER\_CONS\_COLUMNS

- Example

```
SELECT      c2.column_name, c1.constraint_name,
            c1.constraint_type, c1.search_condition,
            c1.r_constraint_name
FROM        user_constraints c1, user_cons_columns c2
WHERE       c1.constraint_name = c2.constraint_name
AND         c1.table_name = 'S_EMP'
ORDER BY   1;
```

BIT Academy Course

Dictionary 조회 시 Object의 이름을 대문자로 조회하도록 한다.

다음은 Dictionary view를 통해 s\_emp table에 만들어진 constraint를 조회하는 내용이다.

```
SQL> SELECT constraint_name, constraint_type,
           search_condition, r_constraint_name
FROM user_constraints
WHERE table_name = 's_emp';
```

선택된 레코드가 없습니다.

```
SQL> c/s_emp/S_EMP/
SQL> /
```

```
SQL> COL constraint_name FORMAT A25
SQL> COL search_condition FORMAT A42
SQL> COL r_constraint_name FORMAT A18
SQL> SET LINESIZE 120
SQL> /
SQL> SELECT c2.column_name, c1.constraint_name,
           c1.constraint_type, c1.search_condition,
           c1.r_constraint_name
FROM user_constraints c1, user_cons_columns c2
WHERE c1.constraint_name = c2.constraint_name
AND c1.table_name = 'S_EMP'
ORDER BY 1;
```

## Script Using Data Dictionary

- You can build useful dynamic scripts using current information within data dictionary.
- Example

```
SELECT 'DROP TABLE ' || object_name ||
      ' CASCADE CONSTRAINTS;'
FROM user_objects
WHERE object_type = 'TABLE';
```

BIT Academy Course

다음은 소유한 모든 table을 drop하기 위한 dynamic script의 예이다.

```
SQL> ed drop
```

```
SET ECHO OFF
SET FEEDBACK OFF
SET PAGESIZE 0
SPOOL drop_tbl.sql
SELECT 'DROP TABLE ' || object_name || ' CASCADE CONSTRAINTS;'
FROM user_objects
WHERE object_type = 'TABLE';
SPOOL OFF
REM @ drop_tbl
SET FEEDBACK ON
SET PAGESIZE 24
SET ECHO ON
```

```
SQL> @drop
```

```
SQL> ed drop_tbl
```

( \* drop\_tbl.sql 파일의 내용을 확인한다. )



# Data Manipulation

- **Data Manipulation Language**
  - INSERT
  - UPDATE
  - DELETE
  - MERGE
  - RETURNING
- **Data Types**
  - Char
  - Varchar2
  - Date Time

## Data Manipulation Language

- Data manipulation language (DML) statements query and manipulate data in existing schema objects.
- These statements do not implicitly commit the current transaction.
  - SELECT
  - INSERT
  - UPDATE
  - MERGE
  - DELETE

## Insert

- To add rows to a table.
- To insert rows into a table, the table must be in your own schema or you must have INSERT privilege on the table.
- Syntax

```
INSERT INTO table_name [(column [, column...]) ]  
{VALUES (value[, value...])|Subquery};
```
- Example

```
INSERT INTO dept  
VALUES (50, 'Local Branch', 'Korea');
```

BIT Academy Course

character와 date 값은 단일 따옴표(')로 둘러싼다.

모든 column의 값을 갖는 row을 삽입할 때는 INSERT clause에 column list를 주지 않고 table에 정의된 column 순서에 따라 VALUES clause에 값을 나열하면 된다.

일부 column 값만 있는 row을 삽입할 때는 INSERT clause에 column list를 주고 VALUES clause에 해당 column 값만 주면 된다.

다음 INSERT 실행 시 발생하는 Error의 이유를 설명하시오.

```
SQL> CREATE TABLE test_con  
      (a NUMBER, b CHAR(2), c DATE NOT NULL,  
       CONSTRAINT test_con_pk PRIMARY KEY(a),  
       CONSTRAINT test_con_ck CHECK(b IN('AA','BB')));  
SQL> INSERT INTO test_con VALUES(10, 'AA', sysdate);  
SQL> INSERT INTO test_con VALUES(10, 'AA', sysdate);  
(* Error 발생)  
SQL> INSERT INTO test_con VALUES(20, 'BB', sysdate);  
SQL> INSERT INTO test_con VALUES(NULL, 'AA', sysdate);  
(* Error 발생)  
SQL> INSERT INTO test_con VALUES(30, 'AA', NULL);  
(* Error 발생)  
SQL> INSERT INTO test_con VALUES(30, 'CC', sysdate);  
(* Error 발생)
```

## Insert(cont.)

- After inserting a row, you can query data with SELECT statement.
- Example

```
SELECT *  
FROM dept  
WHERE deptno = 50;
```

## Insert(cont.)

- Two way to insert a row with NULL values.
- Implicit method : Omit column names from the column list
  - Example
- Explicit method : Specify the NULL keyword in the VALUES clause.

```
INSERT INTO dept (deptno, dname)
VALUES (60, 'HQs');
```

- Example

```
INSERT INTO dept
VALUES (60, 'HQs', NULL);
```

BIT Academy Course

row를 insert하고 결과를 확인하는 실습이다. Error 발생의 이유를 설명하시오.

```
SQL> SELECT * FROM s_dept;
```

(\* 현재의 row들을 확인)

```
SQL> INSERT INTO s_dept VALUES(80, 'Branch');
```

(\* Error발생.)

```
SQL> INSERT INTO s_dept(id, name) VALUES(80, 'Branch');
```

```
SQL> INSERT INTO s_dept VALUES(90, 'HQs', NULL);
```

```
SQL> SELECT * FROM s_dept WHERE id IN (80,90);
```

(\* REGION\_ID column의 값이 NULL인 것을 확인)



### Insert(cont.)

- Insert using pseudocolumns and function.
  - USER : Current user name.
  - SYSDATE : Current date and time.
  - ROWID : Location information of rows.
- Example

```
INSERT INTO emp (empno, ename, hiredate)
VALUES (9000, 'Bit', SYSDATE);
```

BIT Academy Course

pseudocolumn을 이용하여 insert를 실습해 보시오.

```
SQL> INSERT INTO s_emp(id, last_name, userid, start_date)
VALUES (50, 'Bit', 'BIT', SYSDATE);
SQL> INSERT INTO s_emp(id, last_name, userid, start_date)
VALUES(60, 'Bit', USER,TO_DATE('2003, June, 23', 'YYYY, Month, DD'));
SQL> SELECT * FROM s_emp WHERE id IN (50,60);
```

column에 DEFAULT 값이 설정된 경우의 insert를 실습해 보시오.

```
SQL> CREATE TABLE def_tab
(a NUMBER DEFAULT 10, b VARCHAR2(10) DEFAULT 'x', c VARCHAR2(10));
SQL> INSERT INTO def_tab VALUES (DEFAULT, DEFAULT,DEFAULT);
SQL> SELECT * FROM def_tab;
SQL> INSERT INTO def_tab VALUES (DEFAULT, DEFAULT,'aaaaaa');
SQL> SELECT * FROM def_tab;
SQL> INSERT INTO def_tab (c) VALUES ('bbbbbb');
SQL> SELECT * FROM def_tab;
SQL> INSERT INTO def_tab (a, c) VALUES (20, 'cccccc');
SQL> SELECT * FROM def_tab;
```

## Insert(cont.)

- Copy data from another tables by using subquery.
- If the subquery selects no rows, Oracle inserts no rows into the table.
- Example

```
INSERT INTO emp_history
SELECT empno, ename, salary, sysdate
FROM emp;
```

BIT Academy Course

다음은 Subquery를 이용하여 table을 생성하고 data를 입력하는 실습이다.

```
SQL> CREATE TABLE test AS SELECT * FROM s_emp;
SQL> SELECT COUNT(*) FROM test;
SQL> INSERT INTO test SELECT * FROM s_emp;
SQL> / (여러 번 반복)
SQL> SELECT COUNT(*) FROM test;

SQL> CREATE TABLE s_dept_h (id, name, rid)
      AS SELECT id, name, rowid FROM s_dept WHERE 1=0;
SQL> INSERT INTO s_dept_h
      SELECT id, name,rowid FROM s_dept WHERE id < 30;
SQL> SELECT * FROM s_dept_h;

SQL> CREATE TABLE s_emp_h (name, salary)
      AS SELECT last_name, salary FROM s_emp WHERE 1=0;
SQL> INSERT INTO s_emp_h VALUES
      ((SELECT last_name FROM s_emp WHERE rownum < 2),
       (SELECT salary FROM s_emp WHERE rownum < 2));
SQL> SELECT * FROM s_emp_h;
```

## Insert(cont.)

- Oracle returns error if you attempt to insert a data with a value that violates an integrity constraint.
- Example

```
SQL > INSERT INTO emp (empno, ename, deptno)
      VALUES (8000,'SCOTT', 80);

INSERT INTO emp (empno, ename, deptno)
      VALUES (8000,'SCOTT', 80);
*
ERROR at line 1:
ORA-02291: integrity constraint (SCOTT.FK_DEPTNO) violated –
parent key not found
```

## Update

- To change existing values in a table.
- To update values in a table, the table must be in your own schema or you must have UPDATE privilege on the table.
- Syntax

```
UPDATE {table_name | subquery }  
SET column= value[,column= value] [WHERE condition];
```
- Example

```
UPDATE s_emp SET dept_id =10  
WHERE id = 2;
```

BIT Academy Course

WHERE clause가 생략된 UPDATE 문장은 해당 table의 모든 row를 변경하므로 주의해서 사용해야 한다.

```
SQL> SELECT COUNT(*) FROM test WHERE commission_pct = 10;  
SQL> UPDATE test SET commission_pct =10;  
SQL> SELECT COUNT(*) FROM test WHERE commission_pct = 10;
```

column에 DEFAULT 값이 설정된 경우의 update 를 실습해 보시오.

```
SQL> SELECT * FROM def_tab;  
SQL> UPDATE def_tab SET b = 'y';  
SQL> SELECT * FROM def_tab;  
  
SQL> COL column_name FORMAT A30  
SQL> COL data_default FORMAT A30  
SQL> SELECT column_name, data_default FROM user_tab_columns  
WHERE table_name = 'DEF_TAB' ;  
SQL> UPDATE def_tab SET a = DEFAULT, b = DEFAULT WHERE c = 'cccccc';  
SQL> SELECT * FROM def_tab;
```

## Update(cont.)

- Update using subquery.
- Example

```
UPDATE (SELECT * FROM emp WHERE deptno=10)
SET sal = sal * 1.1;
```

```
UPDATE emp
SET deptno = (SELECT deptno FROM dept
              WHERE dname = 'ACCOUNTING'),
    sal = (SELECT MAX(sal) FROM emp)
WHERE empno = 7788;
```

BIT Academy Course

다음 두 문장의 실행결과가 같다.

```
SQL> UPDATE (SELECT * FROM emp WHERE deptno=10)
      SET sal = sal * 1.1;
```

```
SQL> UPDATE emp SET sal = sal * 1.1
      WHERE deptno = 10;
```

다음은 subquery를 이용하여 table을 생성하고 변경하는 실습이다.

```
SQL> CREATE TABLE upt_emp(name, sal, did)
      AS SELECT last_name, salary, dept_id FROM s_emp;
```

```
SQL> SELECT * FROM upt_emp;
```

```
SQL> UPDATE(SELECT * FROM upt_emp WHERE did=45) SET sal = 10;
```

```
SQL> SELECT * FROM upt_emp;
```

```
SQL> UPDATE upt_emp d
      SET sal = (SELECT MAX(salary) FROM s_emp WHERE dept_id = d.did);
```

```
SQL> SELECT * FROM upt_emp;
```

## Update(cont.)

- Oracle returns error if you attempt to change a data with a value that violates an integrity constraint.
- Example

```
SQL> UPDATE emp SET deptno=99 WHERE deptno =10;

      UPDATE emp SET deptno=99 WHERE deptno =10
      *
      ERROR at line 1:
      ORA-02291: integrity constraint (SCOTT.FK_DEPTNO)
      violated parent key not found
```

## Merge

- To update or insert a row conditionally into a table.
- To merge data into a table, the table must be in your own schema or you must have UPDATE and INSERT privilege on the table.
- Syntax

```

MERGE INTO table_name alias
      USING (table|view|subquery) alias
      ON (join condition)
      WHEN MATCHED THEN
          UPDATE SET col1 = val1[, col2 = val2...]
      WHEN NOT MATCHED THEN
          INSERT (col_list) VALUES (val_list);
  
```

BIT Academy Course

**MERGE는 UPDATE와 INSERT를 결합한 문장이다.**

- INTO clause : data가 update 되거나 insert될 table 이름
- USING clause : 대상 table의 data와 비교한 후 update 또는 insert할 때 사용할 data의 source.
- ON clause : update나 insert를 하게 될 condition으로, 해당 condition을 만족하는 row가 있으면 WHEN MATCHED 이하를 실행하게 되고, 없으면 WHEN NOT MATCHED 이하를 실행하게 된다.
- WHEN MATCHED : ON clause의 조건이 TRUE인 row에 수행할 내용
- WHEN NOT MATCHED : ON clause의 조건에 맞는 row가 없을 때 수행할 내용

## Merge(cont.)

- Example

```
MERGE INTO emp_history eh
  USING emp e
  ON (e.empno = eh.empno)
  WHEN MATCHED THEN
    UPDATE SET eh.salary = e.sal
  WHEN NOT MATCHED THEN
    INSERT VALUES (e.empno, sysdate, sal);
```

BIT Academy Course

다음은 s\_emp의 data를 s\_emp\_h에 merge 하는 실습이다.

```
SQL> DROP TABLE s_emp_h;
SQL> CREATE TABLE s_emp_h(id, last_name, salary, before_upt_date,
  last_upt_date)
  AS SELECT id, last_name, salary, sysdate, sysdate FROM s_emp WHERE 1=0;
SQL> COL last_name FORMAT A30
SQL> SET LINESIZE 120
SQL> ALTER SESSION SET nls_date_format = 'YY/MM/DD HH24:MI:SS' ;
SQL> SELECT * FROM s_emp_h;
SQL> MERGE INTO s_emp_h h USING s_emp e ON (e.id = h.id)
  WHEN MATCHED THEN
    UPDATE SET h.salary = e.salary, before_upt_date = last_upt_date,
      last_upt_date = sysdate
  WHEN NOT MATCHED THEN
    INSERT VALUES (e.id, e.last_name, e.salary, sysdate, sysdate);
SQL> SELECT * FROM s_emp_h;
(* before_upt_date, last_upt_date 값이 동일하다.)
SQL> INSERT INTO s_emp (id, last_name, salary) VALUES(30, user,9999);

SQL> MERGE..... (위와 동일한 문장 반복)
SQL> SELECT * FROM s_emp_h;
(* before_upt_date, last_upt_date 값이 다르고, 30번 사번 정보가 추가되었다.)

SQL> ALTER SESSION SET nls_date_format = 'YY/MM/DD' ;
```



## Delete

- To remove rows from a table.
- To delete rows from a table, the table must be in your own schema or you must have DELETE privilege on the table.
- Syntax

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

- Example

```
DELETE FROM emp
WHERE hiredate <
  TO_DATE('1980,01,01','YYYY,MM,DD');
```

BIT Academy Course

WHERE clause가 생략된 DELETE 문장은 해당 table의 모든 row를 삭제하므로 주의해서 사용해야 한다.

Test table의 data를 모두 삭제했다가 취소해 보시오.

```
SQL> SELECT COUNT(*) FROM test;
SQL> DELETE FROM test;
SQL> SELECT COUNT(*) FROM test;
SQL> ROLLBACK;
(*delete를 취소한다.)
SQL> SELECT COUNT(*) FROM test;
```

## Delete(cont.)

- Oracle returns error if you attempt to change a data with a value that violates an integrity constraint.
- Example

```
SQL> DELETE FROM dept WHERE deptno = 10;
      DELETE FROM dept WHERE deptno = 10
      *
      ERROR at line 1:
      ORA-02292: integrity constraint (SCOTT.FK_DEPTNO)
      violated - child record found
```

BIT Academy Course

다음은 Foreign Key Constraint에 위배되는 DML 문장에 대한 실습이다.

```
SQL> CREATE TABLE cons1(a NUMBER PRIMARY KEY);
SQL> INSERT INTO cons1 VALUES (10);
SQL> CREATE TABLE cons2(b NUMBER, CONSTRAINT cons2_fk
      FOREIGN KEY(b) REFERENCES cons1(a));
SQL> INSERT INTO cons2 VALUES (10);
SQL> INSERT INTO cons2 VALUES (20);
(* Error 내용을 적고 그 이유를 설명하시오.)
SQL> UPDATE cons2 SET b = 20 WHERE b = 10;
(* Error 내용을 적고 그 이유를 설명하시오.)
SQL> DELETE FROM cons1 WHERE a = 10;
(* Error 내용을 적고 그 이유를 설명하시오.)
```

다음은 ON DELETE SET NULL 옵션에 대한 실습이다.

```
SQL> ALTER TABLE cons2 DROP CONSTRAINT cons2_fk;
SQL> ALTER TABLE cons2 ADD CONSTRAINT cons2_fk FOREIGN KEY (b)
      REFERENCES cons1(a) ON DELETE SET NULL;
SQL> DELETE FROM cons1;
SQL> SELECT * FROM cons1;
SQL> SELECT * FROM cons2;
```

다음은 ON DELETE CASCADE 옵션에 대한 실습이다.

```
SQL> ROLLBACK;  
SQL> SELECT * FROM cons1;  
SQL> SELECT * FROM cons2;  
SQL> ALTER TABLE cons2 DROP CONSTRAINT cons2_fk;  
SQL> ALTER TABLE cons2 ADD CONSTRAINT cons2_fk FOREIGN KEY (b)  
    REFERENCES cons1(a) ON DELETE CASCADE;  
SQL> DELETE FROM cons1;  
SQL> SELECT * FROM cons1;  
SQL> SELECT * FROM cons2;
```

## Returning Clause

- retrieves the rows affected by the INSERT, UPDATE or DELETE statement.
- to return column expressions and ROWIDs, and store them in output bind variables.

RETURNING *expr* INTO *data\_item*

 Academy Course

RETURNING clause를 실습해 보시오.

```
SQL> VARIABLE a NUMBER
```

```
SQL> VARIABLE b CHAR
```

```
SQL> VARIABLE c NUMBER
```

```
SQL> INSERT INTO s_dept VALUES (100, 'X', 2)
```

```
RETURNING id, name, region_id INTO :a, :b, :c;
```

```
SQL> PRINT a
```

```
SQL> PRINT b
```

```
SQL> PRINT c
```

```
SQL> UPDATE s_dept SET name = 'Y', region_id = 2 WHERE id = 100
```

```
RETURNING id, name, region_id INTO :a, :b, :c;
```

```
SQL> PRINT a
```

```
SQL> PRINT b
```

```
SQL> PRINT c
```

```
SQL> DELETE FROM s_dept WHERE id = 100
```

```
RETURNING id, name, region_id INTO :a, :b, :c;
```

```
SQL> PRINT a
```

```
SQL> PRINT b
```

```
SQL> PRINT c
```

## CHAR Data Types

- The CHAR datatype stores **fixed**-length character strings.
  - specify a string length between 1 and 2000 for the CHAR column width. (The default is 1.)
  - When you insert or update a row in the table, the value for the CHAR column has the fixed length.
  - If you give a shorter value, the value is blank-padded to the fixed length.
  - If you give a longer value with trailing blanks, blanks are trimmed from the value to the fixed length.
  - If a value is too large, Oracle returns an error.
  - Oracle compares CHAR values using blank-padded comparison semantics.

## VARCHAR2 Data Types

- The VARCHAR2 datatype stores variable-length character strings.
  - specify a maximum string length between 1 and 4000 for the VARCHAR2 column.
  - For each row, Oracle stores each value in the column as a variable-length field.
  - Oracle compares VARCHAR2 values using nonpadded comparison semantics.

 Academy Course

다음은 CHAR와 VARCHAR2 type에 대한 실습이다. 각 문장의 수행 결과를 예측하시오.

```
SQL> CREATE TABLE test_char
```

```
  (id NUMBER,
```

```
   a CHAR(2),
```

```
   b VARCHAR2(2));
```

```
SQL> INSERT INTO test_char VALUES(1, 'x','x'); (x 뒤에 공백 없이)
```

```
SQL> INSERT INTO test_char VALUES(2, 'x ','x '); (x 뒤에 공백 하나씩)
```

```
SQL> SELECT * FROM test_char WHERE a = 'x'; (x 뒤에 공백 없이)
```

(\* 결과를 적으시오)

```
SQL> SELECT * FROM test_char WHERE a = 'x '; (x 뒤에 공백 하나)
```

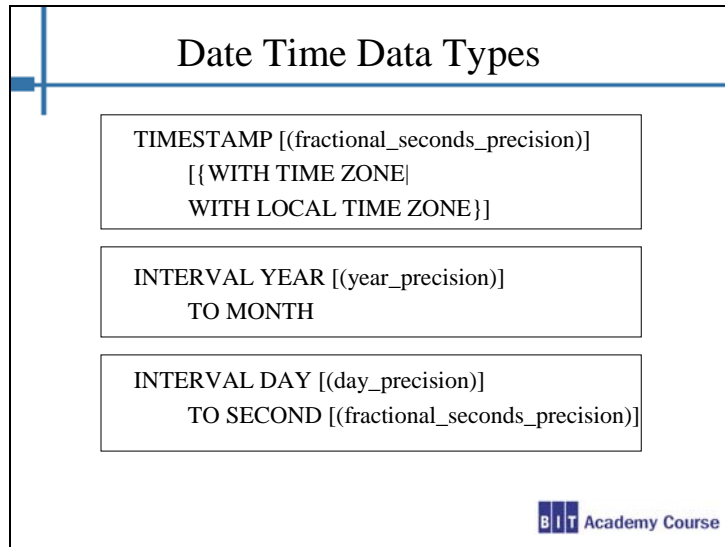
(\* 결과를 적으시오)

```
SQL> SELECT * FROM test_char WHERE b = 'x'; (x 뒤에 공백 없이)
```

(\* 결과를 적으시오)

```
SQL> SELECT * FROM test_char WHERE b = 'x '; (x 뒤에 공백 하나)
```

(\* 결과를 적으시오)



**TIMESTAMP** : DATE type의 값에 미세한 초 단위 정보까지 포함한 data type.

**TIMESTAMP WITH TIME ZONE** : TIMESTAMP type의 값에 UTC(Coordinated Universal Time) 기준 시간으로부터의 offset 값을 포함한 data type.

**TIMESTAMP WITH LOCAL TIME ZONE** : TIMESTAMP WITH TIME ZONE type으로, database time zone에 맞춰 저장되었다가 session time zone에 맞춰 조회해 볼 수 있는 type.

**INTERVAL YEAR TO MONTH** : 두 날짜의 차이 값을 YEAR와 MONTH로 표현하는 type.

**INTERVAL DAY TO SECOND** : 두 날짜의 차이 값을 DAY, HOUR, MINUTE, SECOND로 표현하는 type.

### Date Time type의 column을 다루는 실습을 해보시오.

```
SQL> SELECT value FROM nls_session_parameters;
        WHERE parameter = 'NLS_TIMESTAMP_FORMAT';

SQL> CREATE TABLE time_tab (a TIMESTAMP, b INTERVAL YEAR TO MONTH,
                           c INTERVAL DAY TO SECOND);

SQL> INSERT INTO time_tab VALUES(sysdate, INTERVAL '6' MONTH,
        INTERVAL '30' DAY);

SQL> INSERT INTO time_tab VALUES(sysdate+INTERVAL '2' MONTH,
        INTERVAL '10-6' YEAR TO MONTH, INTERVAL '10 12:10:20' DAY TO SECOND);

SQL> COL a FORMAT A30
SQL> COL b FORMAT A30
SQL> COL c FORMAT A30
SQL> SELECT * FROM time_tab;
SQL> SELECT a, a+b b, a+c c FROM time_tab;
```

## Date Time Functions

Function	Purpose
TZ_OFFSET	Returns the time zone offset from the UTC
CURRENT_DATE	Returns the current date in the session time zone
CURRENT_TIME STAMP	Returns the current date and time in the session time zone WITH TIME ZONE
LOCALTIME STAMP	Returns the current date and time in the session time zone
DBTIMEZONE	Returns the database time zone
SESSIONTIME ZONE	Returns the current session time zone



## Date Time Functions(cont.)

Function	Purpose
EXTRACT	Extracts and returns the specified datetime field
FROM_TZ	Converts a TIMESTAMP value to a TIMESTAMP WITH TIME ZONE value
TO_TIMESTAMP	Converts a string to a TIMESTAMP data type
TO_TIMESTAMP_TZ	Converts a string to a TIMESTAMP WITH TIME ZONE data type
TO_YMINTERVAL	Converts a string to an INTERVAL YEAR TO MONTH data type

### Date Time Functions(cont.)

```
SELECT TZ_OFFSET('Asia/Seoul') FROM dual;
```

```
SELECT DBTIMEZONE, SESSIONTIMEZONE  
FROM dual;
```

```
SELECT EXTRACT (YEAR FROM hiredate)  
FROM emp;
```

```
SELECT TO_TIMESTAMP('2003/07/01 14:00:00',  
                    'YYYY/MM/DD HH24:MI:SS')  
FROM dual;
```

BIT Academy Course

Date time function을 사용하는 실습을 해보시오.

```
SQL> DESC v$timezone_names
SQL> SELECT * FROM v$timezone_names WHERE tzname LIKE '%Seoul%';
SQL> SELECT TZ_OFFSET('Asia/Seoul') FROM dual;

SQL> SELECT CURRENT_DATE, CURRENT_TIMESTAMP, LOCALTIMESTAMP
FROM dual;

SQL> SELECT SESSIONTIMEZONE, DBTIMEZONE FROM dual;
SQL> SELECT start_date,
EXTRACT(YEAR FROM start_date), EXTRACT(MONTH FROM start_date)
FROM s_emp;
SQL> SELECT EXTRACT(DAY FROM start_date), EXTRACT(HOUR FROM start_date)
FROM s_emp;
(* Error 발생)
SQL> SELECT a, EXTRACT(DAY FROM a), EXTRACT(HOUR FROM a)
FROM time_tab;
SQL> SELECT FROM_TZ(TIMESTAMP '2003-08-31 09:30:10', '3:00') FROM dual;
SQL> SELECT FROM_TZ(a, '3:00') FROM time_tab;
SQL> SELECT TO_TIMESTAMP('2003/08/31 09:30:10') FROM dual;
SQL> SELECT TO_TIMESTAMP_TZ('2003/08/31 09:30:10 +9:00') FROM dual;
SQL> SELECT start_date, start_date + TO_YMINTERVAL('1-6') FROM s_emp;
```

# **Transaction Control**

- **Transaction Control**
  - COMMIT
  - SAVEPOINT
  - ROLLBACK
  - Set Transaction

## Transaction Control Statements

- Transaction control statements manage changes made by DML statements.
  - COMMIT
  - ROLLBACK
  - SAVEPOINT
  - SET TRANSACTION

## Transaction

- A transaction is a logical unit of work that comprises one or more SQL statements executed by a single user.
- Example
  - In a bank database, the transaction of transferring money from A account to B account might consist of three separate operations: decrease the A account, increase the B account, and record the transaction in the transaction journal.

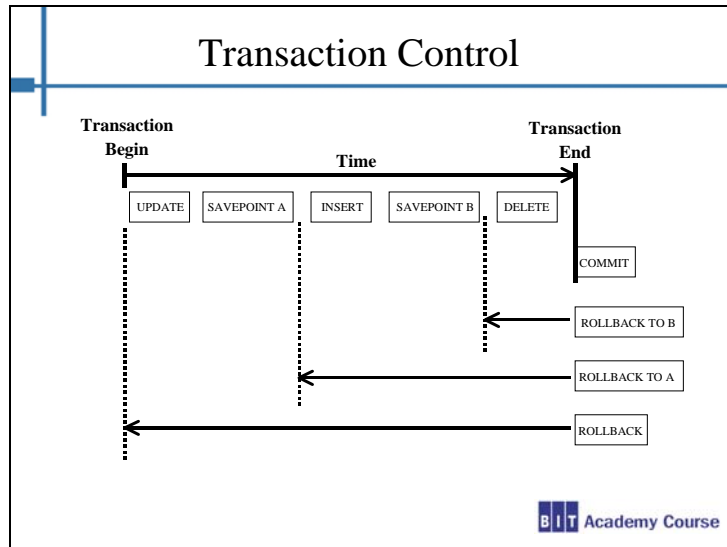
BIT Academy Course

**트랜잭션(Transaction)은 다음의 4가지 속성을 만족해야 한다.**

- 원자성(Atomicity) : 트랜잭션은 연산들을 전부 실행하든지 전혀 실행하지 않아야지 일부만 실행해서는 안된다. All or Nothing
- 일관성(Consistency) : 트랜잭션이 성공적으로 실행되면 데이터베이스 상태는 모순되지 않고 일관된 상태가 된다.
- 격리성(Isolation) : 트랜잭션 실행 도중의 연산 결과는 다른 트랜잭션에서 접근할 수 없다.
- 영속성(Durability) : 트랜잭션이 성공적으로 완료되면 그 결과는 영속적이다.

## Transaction(cont.)

- A transaction begins with the user's first executable SQL statement.
  - DML
  - DDL, DCL : One statement is one transaction
- A transaction ends when it is explicitly committed or rolled back by that user.
  - COMMIT
  - ROLLBACK
  - DDL,DCL :Automatic Commit
  - Abnormal termination, system failure : Automatic Rollback
  - Normal exit from the SQL\*Plus : Automatic Commit



## Transaction Control Statements

Command	Description
COMMIT	To end your current transaction and make permanent all changes performed in the transaction. This statement also erases all savepoints in the transaction and releases the transaction's locks.
ROLLBACK	To undo work done in the current transaction. This statement also erases all savepoints in the transaction and releases the transaction's locks.
SAVEPOINT <i>x</i>	To identify a point <i>x</i> in a transaction to which you can later roll back.
ROLLBACK TO SAVEPOINT <i>x</i>	Rolls back the current transaction to the specified savepoint <i>x</i> .



## Transaction Control Stmt(cont.)

- Example

```
SQL> SELECT deptno FROM emp WHERE empno = 7788;
DEPTNO
-----
      20
SQL> UPDATE emp SET deptno = 30 WHERE empno = 7788;
1 row updated.
SQL> SAVEPOINT a;
Savepoint created.
SQL> DELETE FROM emp;
14 rows deleted.
SQL> ROLLBACK TO a;
Rollback complete.
SQL> SELECT deptno FROM emp WHERE empno = 7788;
DEPTNO
-----
      30
SQL> ROLLBACK;
```

## State of the Data During Transaction

- Before COMMIT or ROLLBACK
  - The affected rows are locked, so other users can't change the same rows.
  - All the changes can be cancelled.
  - The current user session can see the results of DML statements.
  - Other users and even other sessions of the same user can't see the results made by the current user session.

BIT Academy Course


Oracle은 DML 문장을 실행할 때 먼저 변경 대상 row에 Lock을 획득한 후, ROLLBACK SEGMENT에 data의 Before image를 기록한 다음, Data block의 data를 변경한다.

DML 문장으로 변경된 row들은 row level의 Lock이 걸리게 되며, 다른 user들이 동일한 row를 변경하지 못하도록 막아준다. 물론 다른 user들은 같은 table의 다른 row들은 변경할 수 있다. 만일 다른 user가 이미 Lock 이 걸린 row를 변경하고자 한다면 해당 row에 대한 Lock 이 풀릴 때까지 Waiting해야 한다. Transaction 중에 확보된 Lock들은 COMMIT이나 ROLLBACK 명령으로 풀리게 되고 Waiting하던 다음 user가 Lock을 확보하게 된다.

ROLLBACK SEGMENT는 Transaction 진행 중 다른 user가 변경된 data를 조회하고자 할 때 consistent 한 data를 보여주기 위해 사용된다. 또한, Transaction이 Rollback 되면 이 정보를 이용하여 data block을 이전 상태로 복구해 준다.

## State of Data After Transaction

- After COMMIT
  - Changed data is made permanent in the DB.
  - All locks on rows are released, so other users can change the same rows.
  - All users can see the results.
  - All savepoints are erased.
- After ROLLBACK
  - All changes of data is cancelled.
  - All locks on rows are released, so other users can change the same rows.
  - All savepoints are erased.

 Academy Course

SQL\*Plus를 2개 띄워 각각 testxx/testxx로 접속한 후 아래 문장들을 번호 순서대로 테스트하면서 각 문장의 실행 결과를 적으시오.

SESSION #1		SESSION #2	
1	SELECT * FROM tr1;	2	CREATE TABLE tr1 ( a NUMBER);
3	/	4	INSERT INTO tr1 VALUES (10);
5	/	6	SELECT * FROM tr1;
		7	COMMIT;
8	/		
9	UPDATE tr1 SET a = 20;	10	SELECT * FROM tr1;
11	SELECT * FROM tr1;	12	UPDATE tr1 SET a = 30;
13	COMMIT;		
14	SELECT * FROM tr1;	15	SELECT * FROM tr1;
16	/	17	COMMIT;
18	/		

## Set Transaction

- To establish the current transaction's property.

```
SET TRANSACTION READ ONLY;  
SET TRANSACTION READ WRITE;  
SET TRANSACTION  
    ISOLATION LEVEL SERIALIZABLE;  
SET TRANSACTION  
    ISOLATION LEVEL READ COMMITTED;
```

## Set Transaction(cont.)

- READ ONLY
  - establishes the current transaction as a read-only transaction.
  - This clause established transaction-level read consistency.
- READ WRITE
  - establishes the current transaction as a read-write transaction.
  - This clause established statement-level read consistency, which is the *default*.

Transaction의 속성을 변경해 보시오.

```
SQL> SET TRANSACTION READ ONLY;
```

```
SQL> UPDATE s_emp SET salary = salary + 10;
```

(\* Error 발생. 그 이유를 설명하시오.)

## Set Transaction(cont.)

- Transaction Isolation level
  - **SERIALIZABLE** : If a serializable transaction contains data manipulation language(DML) that attempts to update any resource that may have been updated in a transaction uncommitted at the start of the serializable transaction, then the DML statement fails.
  - **READ COMMITTED** : The *default* Oracle transaction behavior. If the transaction contains DML that requires row locks held by another transaction, then the DML statement waits until the row locks are released.

BIT Academy Course

Transaction의 Isolation level을 변경하여 실습해 보시오.

SQL\*Plus를 2개 띄워 각각 testxx/testxx로 접속한 후 아래 문장들을 번호 순서대로 테스트하면서 각 문장의 실행 결과를 적으시오.

SESSION #1		SESSION #2	
1	SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;	2	SELECT salary FROM s_emp WHERE id IN (1,2);
3	SELECT salary FROM s_emp WHERE id IN (1,2);	4	UPDATE s_emp SET salary = 0 WHERE id = 1;
		5	COMMIT;
6	/		
7	UPDATE s_emp SET salary = 10 WHERE id =2;		
8	UPDATE s_emp SET salary = 10 WHERE id = 1;		
9	SELECT salary FROM s_emp WHERE id IN (1,2);	10	SELECT salary FROM s_emp WHERE id IN (1,2);
11	ROLLBACK;		

# Data Definition

- **Alter Table**
  - Column Add/Modify/Drop
  - Constraint Add/Drop/Enable/Disable
- **기타 DDL 명령어**
  - Drop Table
  - Rename
  - Truncate Table
  - Comment

## Some of DDL Statements

Statement	Purpose
CREATE TABLE	To create a relational table, the basic structure to hold user data.
ALTER TABLE	To alter the definition of a table.
DROP TABLE	To remove a table and all its data from the database.
TRUNCATE	To remove all rows from a table and reset the STORAGE parameters to the values when the table or cluster was created. Deleting rows with the TRUNCATE statement can be more efficient.
RENAME	To rename a table, view, sequence, or private synonym for a table, view, or sequence.
COMMENT	To add a comment about a table, view, materialized view, or column into the data dictionary.

 BIT Academy Course

### DDL(Data Definition Language)

database 구조를 만들고, 갱신하고, 삭제하는 SQL 명령의 부분집합으로 이 명령은 즉각적으로 database에 영향을 미치며, 데이터 사전(Data Dictionary)에 정보를 기록한다.



## Alter Tables

- To alter the definition of a table.
- The table must be in your own schema, or you must have ALTER privilege on the table, or you must have ALTER ANY TABLE system privilege.
- You can add, modify and drop columns using ALTER TABLE statement.
- You can add, drop, enable, disable constraints using ALTER TABLE statement.

## Alter Tables for Columns

- Syntax

```
ALTER TABLE table_name  
ADD (column datatype [DEFAULT expr]  
      [column_constraints], ..... );
```

```
ALTER TABLE table_name  
MODIFY (column datatype [DEFAULT expr]  
        [column_constraints], ..... );
```

```
ALTER TABLE table_name  
SET UNUSED (column );
```

```
ALTER TABLE table_name  
DROP (column )|UNUSED COLUMNS;
```

## Add Columns

- If you add a column, the initial value of each row for the new column is null.
- You cannot add a column with a NOT NULL constraint if table has any rows.
- Example

```
ALTER TABLE dept  
ADD (comments VARCHAR2(30));
```

새로 추가하는 column은 그 table의 마지막에 추가된다.

## Modify Columns

- Modifies the definition of an existing column.
- You can change any column's data type or decrease any column's size if all rows for the column contain nulls.
- You can always increase the size of a character or raw column or the precision of a numeric column, whether or not all the columns contain nulls.
- Example

```
ALTER TABLE dept  
MODIFY (comments VARCHAR2(255));
```

## Drop Columns

- Removes the column descriptor and the data associated with the target column from each row in the table.
- All indexes defined on any of the target columns are also dropped.
- All constraints that reference a target column are removed.
- Example

```
ALTER TABLE dept  
DROP (comments );
```

BIT Academy Course

다음은 column을 추가, 길이 변경, 삭제하는 실습이다.

```
SQL> CREATE TABLE alter1(a NUMBER, b CHAR);
```

```
SQL> DESC alter1
```

```
SQL> INSERT INTO alter1 VALUES(10,'x');
```

```
SQL> INSERT INTO alter1 VALUES(10,'y');
```

```
SQL> SELECT * FROM alter1;
```

```
SQL> ALTER TABLE alter1 ADD c DATE default sysdate;
```

```
SQL> SELECT * FROM alter1;
```

```
SQL> SELECT length(a), length(b), length(c) FROM alter1;
```

```
SQL> ALTER TABLE alter1 MODIFY b CHAR(4);
```

```
SQL> SELECT length(a), length(b), length(c) FROM alter1;
```

```
SQL> ALTER TABLE alter1 DROP (b) ;
```

```
SQL> SELECT * FROM alter1;
```

## Unused Columns

- SET UNUSED : By marking them to be dropped at a future time when the demand on system resources is less.
- DROP UNUSED COLUMNS : Removes from the table all columns currently marked as unused.
- Example

```
ALTER TABLE dept
SET UNUSED (comments);
```

```
ALTER TABLE dept
DROP UNUSED COLUMNS;
```

BIT Academy Course

다음은 column을 unused로 변경했다가 삭제하는 실습이다.

```
SQL> DESC alter1
```

```
SQL> ALTER TABLE alter1 SET UNUSED(c);
```

```
SQL> DESC alter1
```

```
SQL> SELECT * FROM alter1;
```

```
SQL> ALTER TABLE alter1 DROP UNUSED COLUMNS;
```

```
SQL> DESC alter1
```

## Alter Tables for Constraints

- Syntax

```
ALTER TABLE table_name  
ADD [CONSTRAINT constraint_name]  
constraint_type (column);
```

```
ALTER TABLE table_name  
DROP PRIMARY KEY | UNIQUE (column) |  
CONSTRAINT constraint [CASCADE];
```

```
ALTER TABLE table_name  
DISABLE | ENABLE CONSTRAINT  
constraint_name [CASCADE];
```

## Add Constraints

- Add all types of constraints except NOT NULL that must be specified only in the ALTER TABLE MODIFY clause.
- If PRIMARY KEY or UNIQUE constraint is added, Oracle automatically builds a unique index on the column or columns related to the constraints.
- Example

```
ALTER TABLE emp  
ADD CONSTRAINT emp_mgr_fk  
FOREIGN KEY (mgr)  
REFERENCES emp(empno);
```



## Drop Constraints

- Drops an integrity constraint from the database.
- Oracle stops enforcing the constraint and removes it from the data dictionary.
- You cannot drop a UNIQUE or PRIMARY KEY constraint that is part of a referential integrity constraint without also dropping the foreign key.
- To drop the referenced key and the foreign key together, use the CASCADE clause.
- Example

```
ALTER TABLE dept  
DROP PRIMARY KEY CASCADE;
```

## Enable Constraints

- To enable or disable any integrity constraint, you must have defined the constraint.
- You cannot enable a referential integrity constraint unless the referenced unique or primary key constraint is already enabled.
- If PRIMARY KEY or UNIQUE constraint is enabled, Oracle automatically builds an unique index.
- Example

```
ALTER TABLE emp  
ENABLE PRIMARY KEY;
```

## Disable Constraints

- Disables the integrity constraint.
- Disabled integrity constraints appear in the data dictionary along with enabled constraints.
- If PRIMARY KEY or UNIQUE constraint is disabled, Oracle automatically drops an unique index on the column or columns related to the constraints.
- Example

```
ALTER TABLE emp DISABLE PRIMARY KEY;
```

BIT Academy Course

data 상태가 만족되어야 제약조건이 추가, 또는 enable될 수 있음을 보여주는 실습이다.

```
SQL> CREATE TABLE alter2(a NUMBER, b CHAR, c DATE);
SQL> INSERT INTO alter2 VALUES(10,'x', sysdate);
SQL> INSERT INTO alter2 VALUES(10,'y', sysdate);
SQL> ALTER TABLE alter2 ADD CONSTRAINT alter2_a_pk PRIMARY KEY(a);
(* Error 발생. 이유를 설명하시오.)
SQL> SELECT * FROM alter2 ORDER BY a;
SQL> DELETE FROM alter2 WHERE a=10 AND b='y';

SQL> ALTER TABLE alter2 ADD CONSTRAINT alter2_a_pk PRIMARY KEY (a);
SQL> INSERT INTO alter2 VALUES(10,'y',sysdate);
(* Error 발생. 이유를 설명하시오.)

SQL> ALTER TABLE alter2 DISABLE CONSTRAINT alter2_a_pk;
SQL> INSERT INTO alter2 VALUES(10,'y',sysdate);
SQL> /
SQL> ALTER TABLE alter2 ENABLE CONSTRAINT alter2_a_pk;
(* Error 발생. 이유를 설명하시오.)

SQL> DELETE FROM alter2 WHERE a=10 AND b='y';
SQL> ALTER TABLE alter2 ENABLE CONSTRAINT alter2_a_pk;
```

## Drop Table

- To remove a table and all its data from the database as if the rows were deleted.
- Drops all the table's indexes.
- Oracle invalidates all dependent objects of the table, but does not drop them.
- CASCADE CONSTRAINT drops all referential integrity constraints that refer to primary and unique keys in the dropped table.
- The table must be in your own schema or you must have the DROP ANY TABLE system privilege.

## DROP Table(cont.)

- Syntax

DROP TABLE [schema].table\_name  
[CASCADE CONSTRAINT];
- Example

DROP TABLE history;

BIT Academy Course

다음을 실행하면서 Error가 발생하는 이유를 설명하시오.

```
SQL> SELECT * FROM cons1;
```

```
SQL> SELECT * FROM cons2;
```

```
SQL> DROP TABLE cons1;
```

(\* Error 발생. 그 이유를 설명하시오.)

```
SQL> DROP TABLE cons1 CASCADE CONSTRAINTS;
```

```
SQL> DROP TABLE cons2;
```

## Rename

- To rename a table, view, sequence, or private synonym for a table, view, or sequence.
- Oracle automatically transfers integrity constraints, indexes, and grants on the old object to the new object.
- Oracle invalidates all objects that depend on the renamed object, such as views, synonyms, and stored procedures and functions that refer to a renamed table.
- The object must be in your own schema.

### Rename(cont.)

- Syntax

```
RENAME old_name TO new_name;
```
- Example

```
RENAME emp TO employees;
```

BIT Academy Course

table의 이름을 변경해 보시오.

```
SQL> RENAME alter1 TO alter2;
```

(\* Error 발생)

```
SQL> RENAME alter1 TO alter4;
```

```
SQL> DESC alter1
```

```
SQL> DESC alter4
```

```
SQL> RENAME alter4 TO alter1;
```

## Truncate

- To remove all rows from a table and reset the `STORAGE` parameters to the values when the table was created.
- You cannot roll back a `TRUNCATE` statement.
- Deleting rows with the `TRUNCATE` statement can be more efficient than dropping and re-creating a table.
- The table or cluster must be in your schema or you must have `DROP ANY TABLE` system privilege.



## Truncate(cont.)

- Syntax

TRUNCATE TABLE table\_name;
- Example

TRUNCATE TABLE history;

BIT Academy Course

**DELETE는 ROLLBACK이 가능하나 TRUNCATE는 ROLLBACK이 불가능함을 확인하시오.**

```
SQL> SELECT count(*) FROM alter2;
SQL> DELETE FROM alter2;
SQL> SELECT count(*) FROM alter2;
SQL> ROLLBACK;
SQL> SELECT count(*) FROM alter2;
(* Rollback이 되어 data가 복구된다.)
```

```
SQL> TRUNCATE TABLE alter2;
SQL> SELECT count(*) FROM alter2;
SQL> ROLLBACK;
SQL> SELECT count(*) FROM alter2;
(* Rollback이 불가능하며 data는 영구히 손실된다.)
```

## Comment

- To add a comment about a table or column into the data dictionary.
- You can view the comments on a particular table or column by querying the data dictionary views.
  - USER\_TAB\_COMMENTS
  - DBA\_TAB\_COMMENTS
  - ALL\_TAB\_COMMENTS
  - USER\_COL\_COMMENTS
  - DBA\_COL\_COMMENTS
  - ALL\_COL\_COMMENTS


## Comment(cont.)

- Syntax

```
COMMENT ON TABLE table_name |  
                COLUMN table_name.column  
IS 'add comment text';
```
- Example

```
COMMENT ON TABLE emp  
IS 'Employee Information';
```

```
COMMENT ON COLUMN  
emp.ename IS 'Employee Name';
```



table과 column에 주석을 달고 확인해 보시오.

```
SQL> SELECT comments FROM user_tab_comments  
      WHERE table_name = 'S_EMP';  
SQL> COMMENT ON TABLE s_emp IS 'This table is for employee information';  
SQL> SELECT comments FROM user_tab_comments  
      WHERE table_name = 'S_EMP';  
  
SQL> COL table_name FORMAT A20  
SQL> COL column_name FORMAT A20  
SQL> COL comments FORMAT A35  
SQL> SELECT * FROM user_col_comments  
      WHERE table_name = 'S_EMP';  
SQL> COMMENT ON COLUMN s_emp.last_name IS 'Employee Last Name'  
SQL> SELECT * FROM user_col_comments  
      WHERE table_name = 'S_EMP';
```

# Other DB Objects

- **VIEW**

- CREATE/DROP VIEW
- WITH CHECK OPTION
- WITH READ ONLY
- ALTER VIEW

- **INDEX**

- Index의 종류
- CREATE (UNIQUE) INDEX, DROP INDEX
- SQL Trace
- Index가 필요한 경우

- **Sequence**

- CREATE/DROP SEQUENCE
- NEXTVAL
- CURRVAL

- **Synonym**

- CREATE/DROP
- Public Synonym

## Oracle Main DB Objects

Object	Purpose
VIEW	A logical table based on one or more tables or views.
INDEX	A schema object that contains an entry for each value that appears in the indexed column(s) of the table or cluster and provides direct, fast access to rows.
SEQUENCE	A schema object that generate unique sequential values.
SYNONYM	A synonym is an alias for a table, view, sequence, or program unit.

## View

- View is a logical table based on one or more tables or views.
- A view contains no data itself.
- The tables upon which a view is based are called **base tables**.
- Oracle stores a view's definition in the data dictionary as the text of the query that defines the view.
- To create a view in your own schema, you must have CREATE VIEW system privilege.

## Usages of View

- to provide an additional level of table security by restricting access to a predetermined set of rows and/or columns of a table.
- to hide data complexity and to simplify commands for the user.
- to present the data in a different perspective from that of the base table.
- to isolate applications from changes in definitions of base tables.
- to save complex queries.

## CREATE VIEW

- Syntax

```
CREATE [OR REPLACE] [FORCE|NO FORCE]
VIEW view_name [(alias[,alias]...)]
AS Subquery
[WITH READ ONLY]
[WITH CHECK OPTION [CONSTRAINT constraint];
```



## CREATE VIEW(Cont.)

Paramter	Purpose
OR REPLACE	re-creates the view if it already exists. You can use this clause to change the definition of an existing view without dropping, re-creating.
FORCE	creates the view regardless of whether the view's base tables or the referenced object types exist or the owner of the schema containing the view has privileges on them.
NO FORCE	creates the view only if the base tables exist and the owner of the schema containing the view has privileges on them. This is the <i>default</i> .
WITH READ ONLY	specifies that no delete, inserts, or updates can be performed through the view.
WITH CHECK OPTION	specifies that inserts and updates performed through the view must result in rows that the view query can select.
CONSTRAINT <i>constraint</i>	assigns the name of the CHECK OPTION constraint.

BIT Academy Course

Base table이 없는 상태에서 View를 만들기 위해서는 Force option을 사용한다.

```
SQL> CREATE VIEW v_x
      AS SELECT * FROM s_ddd;
```

(\* Error 발생. 그 이유를 설명하시오.)

```
SQL> CREATE FORCE VIEW v_x
      AS SELECT * FROM s_ddd;
```

(\* Warning 발생. 그 이유를 설명하시오.)

```
SQL> DESC v_x
```

## CREATE VIEW(Cont.)

- Example

```
SQL> CREATE VIEW emp20
      AS SELECT empno, ename, sal
      FROM emp WHERE deptno = 20;

SQL> DESC emp20

SQL> SELECT * FROM emp20;

SQL> CREATE OR REPLACE emp20
      (eno, name, payroll)
      AS SELECT empno, ename, sal
      FROM emp WHERE deptno = 20;
```

BIT Academy Course

고객을 담당하고 있는 영업사원 View를 생성해 보시오.

```
SQL> CREATE VIEW v_sales_rep
      AS SELECT id, last_name, dept_id FROM s_emp
      WHERE title = 'Sales Representative';
```

영업사원이 담당하는 고객정보 View를 생성해 보시오.

```
SQL> CREATE VIEW v_emp_cust
      AS SELECT e.last_name, c.name, e.dept_id FROM v_sales_rep e, s_customer c
      WHERE e.id = c.sales_rep_id;
```

```
SQL> COL name FORMAT A40

SQL> SELECT * FROM v_emp_cust;

SQL> SELECT * FROM v_emp_cust ORDER BY 1;
```

중복 값을 제거하여 출력해 보시오.

```
SQL> BREAK ON last_name ON dept_id
(* last_name, dept_id 의 중복 값을 제거하고 display해준다.)

SQL> SELECT dept_id, last_name, name FROM v_emp_cust ORDER BY 1;
```

## Types of View

- Simple View
  - Based on only one table and doesn't have any functions or groups of data in select list.
  - Update, Insert, Delete operations are permitted.
- Complex View
  - Based on many tables and has functions or groups of data in select list.
  - Update, Insert, Delete operations aren't permitted.

BIT Academy Course

Simple View에서는 DML이 가능하다.

View가 다음을 포함하는 경우 row를 Delete할 수 없다.

- Join condition, Aggregate Function, GROUP BY clause, DISTINCT

View가 다음을 포함하는 경우 row를 Update할 수 없다.

- 위의 조건, expression으로 정의된 Column

View가 다음을 포함하는 경우 row를 Insert할 수 없다.

- 위의 조건, View로 선택되지 않은 NOT NULL Column

User 마다 다른 data를 보여주는 View를 만들어 실험해 본다.

```
SQL> DELETE FROM s_emp WHERE userid=user;
SQL> CREATE VIEW myteam
      AS SELECT id, last_name, userid, dept_id FROM s_emp
         WHERE dept_id IN (SELECT dept_id FROM s_emp WHERE userid = user);
SQL> INSERT INTO myteam VALUES (40, 'xxxxx', user, 31);
SQL> INSERT INTO myteam VALUES (41, 'yyyyy', 'SCOTT', 41);
SQL> SELECT * FROM myteam;
(* 조회 결과를 기록하시오.)
SQL> GRANT SELECT ON myteam TO SCOTT;
SQL> conn SCOTT/TIGER
SQL> SELECT * FROM testxx.myteam;
(* 조회 결과를 기록하고 위의 결과와 비교해 보시오.)
```

## Types of View(Cont.)

- Updatable join view
  - Based two or more base tables or views.
  - Update, Insert, Delete operations are permitted.
  - Query `USER_UPDATABLE_COLUMNS` to see whether the columns in a join view are updatable.

Updatable Join View에 대해 DML 문장을 실행해 보시오.

```
SQL> conn testxx/testxx
SQL> CREATE OR REPLACE VIEW dept_emp (did, dname, eid, ename)
AS SELECT d.id, d.name, e.id, e.last_name
FROM s_dept d, s_emp e
WHERE d.id = e.dept_id;
SQL> SELECT * FROM dept_emp;
SQL> COL column_name FORMAT A20
SQL> SELECT column_name, updatable, insertable, deletable
FROM user_updatable_columns
WHERE table_name = 'DEPT_EMP';
SQL> INSERT INTO dept_emp (eid, ename) VALUES (100, user);
SQL> SELECT * FROM s_emp WHERE id = 100;
SQL> SELECT * FROM dept_emp WHERE eid = 100;
SQL> UPDATE s_emp SET dept_id = 10 WHERE id = 100;
SQL> UPDATE dept_emp SET ename = 'XXX' WHERE eid = 100;
SQL> SELECT * FROM dept_emp;
```

## Types of View(Cont.)

- Inline view
  - It's not a schema object.
  - It is a subquery with an alias (correlation name) that you can use like a view within a SQL statement.

BIT Academy Course

Inline view를 이용한 DML 문장을 실습해 보시오.

```
SQL> INSERT INTO(SELECT id, last_name, dept_id FROM s_emp
WHERE dept_id=31)
VALUES(60, user,31);
SQL> INSERT INTO(SELECT id, last_name, dept_id FROM s_emp
WHERE dept_id=31)
VALUES(61, user,32);
SQL> UPDATE(SELECT id, last_name, dept_id FROM s_emp WHERE dept_id=31)
SET last_name = 'xxxx' WHERE id = 60;
SQL> UPDATE(SELECT id, last_name, dept_id FROM s_emp WHERE dept_id=31)
SET last_name = 'xxxx' WHERE id = 1;
(* 0 rows updated)
SQL> UPDATE(SELECT id, last_name, dept_id FROM s_emp WHERE dept_id=31)
SET salary = 10 WHERE id = 60;
(* Error발생. 그 이유를 설명하시오.)
SQL> SELECT id, dept_id FROM s_emp WHERE dept_id=32 OR id IN (60,61);
SQL> DELETE FROM(SELECT id FROM s_emp WHERE dept_id=32);
SQL> DELETE FROM(SELECT id FROM s_emp WHERE dept_id=32)
WHERE id = 1;
SQL> DELETE FROM(SELECT id FROM s_emp WHERE dept_id=32)
WHERE id IN(60,61);
```

## WITH READ ONLY

- Specifies that no delete, inserts, or updates can be performed through the view.
- Example

```
CREATE OR REPLACE VIEW emp20
(id, name, title)
AS SELECT empno, ename, job
FROM emp
WHERE deptno = 20
WITH READ ONLY;
```

 Academy Course

WITH READ ONLY view에 대한 실습을 진행하면서 Error 발생의 이유를 적으시오.

```
SQL> CREATE TABLE s_eee
      AS SELECT id, last_name, dept_id FROM s_emp;
SQL> CREATE OR REPLACE VIEW s_eee_v41 (empno, empname,deptno)
      AS SELECT id, last_name, dept_id
      FROM s_eee WHERE dept_id = 41
      WITH READ ONLY;
SQL> UPDATE s_eee_v41 SET empname = 'XXXX';
(* Error 발생)
SQL> CREATE OR REPLACE VIEW s_eee_v41 (empno, empname,deptno)
      AS SELECT id, last_name, dept_id
      FROM s_eee WHERE dept_id = 41;
SQL> UPDATE s_eee_v41 SET empname = 'XXXX';
SQL> SELECT * FROM s_eee_v41;
SQL> SELECT * FROM s_eee;
```

## WITH CHECK OPTION

- Specifies that inserts and updates performed through the view must result in rows that the view query can select.
- Example

```
CREATE OR REPLACE VIEW emp20
AS SELECT *
FROM emp
WHERE deptno = 20
WITH CHECK OPTION
CONSTRAINT emp20_ck;
```

BIT Academy Course

WITH CHECK OPTION view에 대한 실습을 진행하면서 Error 발생의 이유를 적으시오.

```
SQL> CREATE OR REPLACE VIEW s_eee_v41
AS SELECT id, last_name, dept_id
FROM s_eee WHERE dept_id = 41
WITH CHECK OPTION CONSTRAINT s_eee_v41_ck;
SQL> SELECT * FROM s_eee_v41;
SQL> UPDATE s_eee_v41 SET dept_id = 31;
(* Error 발생)
SQL> INSERT INTO s_eee_v41 VALUES ( 1000, 'AAAA', 1000);
(* Error 발생)
SQL> INSERT INTO s_eee_v41 VALUES ( 2000, 'AAAA', 41);
SQL> SELECT * FROM s_eee_v41;
```

WITH CHECK OPTION 을 가진 inline view에 대한 실습이다.

```
SQL> INSERT INTO (SELECT id, last_name, dept_id FROM s_emp
WHERE dept_id=31 WITH CHECK OPTION) VALUES (62, user,32);
(* Error발생. 그 이유를 설명하시오.)
SQL> UPDATE(SELECT id, last_name, dept_id FROM s_emp
WHERE dept_id=31 WITH CHECK OPTION) SET last_name='yyyy' WHERE id= 60;
SQL> UPDATE(SELECT id, last_name, dept_id FROM s_emp
WHERE dept_id=31 WITH CHECK OPTION) SET dept_id = 32;
(* Error발생. 그 이유를 설명하시오.)
```

## DROP VIEW

- To remove a view or an object view from the database.
- Oracle deletes only the definition information of the VIEW on Dictionary, not dropping base table of the VIEW.
- Syntax

```
DROP VIEW view_name;
```

```
SQL> DROP VIEW myteam;
```



## Dictionary for Views

- USER\_VIEWS
- USER\_OBJECTS
- Example

```
SELECT object_name, created, status
FROM   user_objects
WHERE  object_type = 'VIEW';
```

```
SELECT view_name, text
FROM   user_views;
```

BIT Academy Course

USER\_VIEWS에서 View에 대한 정의 내용을 조회해 본다.

```
SQL> SET LONG 1000
```

```
SQL> SELECT view_name, text FROM user_views;
```

USER\_OBJECTS의 status는 Base Table의 상태에 따라 변화한다. Base Table에 대한 변화가 있게 되면 그에 dependent한 View는 INVALID 상태로 바뀌고, View에 대한 질의가 수행되는 순간 Oracle 은 View를 다시 컴파일하여 VALID 한 경우 status를 바꾸어 놓고 질의를 수행한다.

다음은 base table을 변화시키면서 view의 status를 추적하는 실습이다.

```
SQL> SELECT object_name, created, status
      FROM user_objects
      WHERE object_type = 'VIEW' AND object_name = 'S_EEE_V41';
```

```
SQL> SAVE v1
```

```
SQL> ALTER TABLE s_eee ADD (x NUMBER);
```

```
SQL> @v1
```

(\* status가 INVALID 상태로 바뀐 것을 확인)

```
SQL> SELECT * FROM s_eee_v41;
```

```
SQL> @v1
```

(\* status가 VALID 상태로 바뀐 것을 확인)

```
SQL> ALTER TABLE s_eee DROP (x);
```

```
SQL> @v1
```

(\* status가 INVALID 상태로 바뀐 것을 확인)

INDEX

- Optional structures associated with tables.
- Create to increase the performance of data retrieval.
- Indexes are the primary means of reducing disk I/O when properly used.
- Once created, an index is automatically maintained and used by Oracle.
- Indexes are logically and physically independent of the data.
- Oracle uses B\*-tree indexes that are balanced to equalize access times to any row.

BIT Academy Course

index는 data에 대한 조회 속도를 높이기 위해 만든다.

Data에 대한 처리 요청 시 Oracle은 효율적인 처리를 위해 필요한 index들을 찾아 사용한다.

index는 특정한 row를 찾거나 일정한 range의 row를 찾을 때 유용하다.

index는 만들고 나면 Oracle 서버가 자동적으로 사용하고 유지보수 한다. Data에 대한 update, insert, delete를 요청하면 연관 있는 index에 자동으로 반영해 준다.

index는 table과는 독립적이므로 index를 drop해도 table의 data에는 영향을 주지 않는다. 다만, 그 table에 대한 DML처리 속도가 달라질 수는 있다.

## Types of Index

Type	Description
Single column ⇔ Composite	A composite index is an index that you create on multiple columns in a table.
Unique ⇔ Nonunique	Unique indexes guarantee that no two rows of a table have duplicate values in the columns that define the index.
Column data ⇔ Function-Based	Function-Based index is index on functions and expressions that involve one or more columns in the table being indexed.
Automatic created ⇔ User create	Oracle enforces UNIQUE or PRIMARY KEY integrity constraints by automatically defining a unique index on the column or columns.

## CREATE INDEX

- Syntax

```
CREATE [UNIQUE] INDEX index_name  
ON table_name (Column|Expr[,Column|Expr]...);
```
- Example

```
CREATE INDEX emp_ename_ind  
ON emp (ename);
```

BIT Academy Course

Unique index를 생성해 보시오.

```
SQL> CREATE TABLE s_emp_i AS SELECT * FROM s_emp;  
SQL> CREATE UNIQUE INDEX s_emp_i_id_ind ON s_emp_i(id);  
SQL> CREATE UNIQUE INDEX s_emp_i_last_name_ind  
ON s_emp_i(last_name);  
(* Error 발생. 그 이유를 설명하시오)
```

composite index를 생성해 보시오.

```
SQL> CREATE INDEX s_emp_i_last_name_dept_id_ind  
ON s_emp_i(dept_id, last_name);  
SQL> CREATE INDEX s_emp_i_dept_id_last_name_ind  
ON s_emp_i(last_name, dept_id);
```

Function-based index를 생성해 보시오.

```
SQL> CREATE INDEX s_emp_i_last_name_ind  
ON s_emp_i UPPER(last_name);
```

## DROP INDEX

- To remove an index from the database.
- The index must be in your own schema or you must have the DROP ANY INDEX system privilege.

- Syntax

```
DROP INDEX schema.index_name;
```

- Example

```
DROP INDEX emp_ename_ind;
```

### Dictionary for Indexes

- USER\_INDEXES
- USER\_IND\_COLUMNS
- Example

```
SELECT  t.index_name, t.uniqueness,
        c.column_name, c.column_position
FROM    user_indexes t, user_ind_columns c
WHERE   c.index_name = t.index_name
AND     t.table_name = 'EMP';
```

BIT Academy Course

다음은 Dictionary에서 S\_EMP table에 만들어진 index를 조회하는 내용이다.

```
SQL> COL index_name FORMAT A20
SQL> COL column_name FORMAT A30
SQL> SET LINESIZE 100
SQL> SELECT t.index_name, t.uniqueness,
           c.column_name, c.column_position
FROM   user_indexes t, user_ind_columns c
WHERE  c.index_name = t.index_name
AND    t.table_name = 'S_EMP';
```

### Needs for the Index

- Frequently used columns in WHERE clause or join conditions.
- Very large table but less than 2 – 4 % of rows are retrieved.
- Columns has a wide range of values or a large number of null values.
- Not frequently updated table.

BIT Academy Course

Oracle server가 조회 시 index를 사용하는지 알아보기 위해 Execution Plan을 Trace해본다.

```
SQL> conn system/manager
```

```
SQL> @ORACLE_HOME\WRDBMS\ADMIN\WUTLXPAN
```

(\* Oracle Optimizer의 Execution Plan 정보를 넣어들 PLAN\_TABLE 생성)

```
SQL> SET AUTOTRACE TRACE EXP
```

(\* SQL\*Plus에서 실행문 마다 Execution Plan을 보여주도록 설정)

```
SQL> CREATE TABLE exp_table AS SELECT * FROM testxx.s_emp;
```

```
SQL> SELECT * FROM exp_table WHERE last_name = 'Menchu';
```

(\* Plan이 FULL TABLE SCAN으로 나타남)

```
SQL> CREATE INDEX last_name_ind ON exp_table(last_name);
```

```
SQL> SELECT * FROM exp_table WHERE last_name = 'Menchu';
```

(\* Plan이 INDEX 를 활용하는 것으로 나타남)

```
SQL> SELECT * FROM exp_table;
```

(\* Plan이 FULL TABLE SCAN으로 나타남)

## Sequence

- A **sequence** is a database object from which multiple users may generate unique integers .
- Use sequences to automatically generate primary key or unique column values.
- Sequence numbers are generated independently of tables, so the same sequence can be used for one or for multiple tables.
- It is possible that individual sequence numbers will appear to be skipped, because they were generated and used in a transaction that ultimately rolled back.

 Academy Course

Sequence는 Primary Key 값을 자동으로 발생시킬 목적으로 생성한다.

Sequence는 table 또는 column과는 독립적으로 생성, 삭제된다.



## CREATE SEQUENCE

- Syntax

```
CREATE SEQUENCE sequence_name
  [INCREMENT BY n]
  [START WITH n]
  [{MAXVALUE n | NOMAXVALUE}]
  [{MINVALUE n | NOMINVALUE}]
  [{CYCLE | NOCYCLE}]
  [{CACHE n | NOCACHE}]
```

- Example

```
CREATE SEQUENCE seq_dept_no
  START WITH 100
  INCREMENT BY 10;
```

BIT Academy Course

### INCREMENT BY n

생성되는 Sequence번호의 간격을 정수 n으로 정의한다.  
옵션이 생략되면 Sequence는 1씩 증가한다.

### START WITH n

생성되는 첫 번째 Sequence 번호를 정의한다.  
옵션이 생략되면 sequence는 1부터 시작한다.

### MAXVALUE n

생성 가능한 Sequence의 최대값을 정의한다.  
NOMAXVALUE가 default이며 최대값은 10의 27승이다.

### MINVALUE n

생성 가능한 Sequence의 최소값을 정의한다.  
디폴트가 NOMINVALUE이며 최소값은 1이다.

### CACHE n

Oracle server가 cache에 미리 Sequence를 생성해 놓는 개수를 정의한다.  
Default가 20이다.  
cache하지 않으려면 NOCACHE 옵션을 사용한다.

## Using Sequence

- Refer to sequence values in SQL statements with these pseudocolumns:

Column	Purpose
CURRVAL	returns the current value of a sequence
NEXTVAL	increments the sequence and returns the next value

- Example

```
INSERT INTO dept (deptno, dname, loc)
VALUES (seq_dept_no.NEXTVAL, 'Training', 'Seoul');
```

```
SELECT seq_dept_no.CURRVAL
FROM dual;
```

**BIT** Academy Course

Sequence를 이용하여 Insert, Update에 활용해 보시오.

```
SQL> conn testxx/testxx
SQL> CREATE SEQUENCE s1;
SQL> SELECT s1.NEXTVAL FROM dual;
SQL> COL a FORMAT 99999
SQL> CREATE TABLE seq_tab1(a NUMBER);

SQL> INSERT INTO seq_tab1 VALUES(s1.NEXTVAL);
SQL> SELECT * FROM seq_tab1;
SQL> INSERT INTO seq_tab1 SELECT s1.NEXTVAL FROM dual;
SQL> SELECT * FROM seq_tab1;
SQL> UPDATE seq_tab1 SET a = s1.NEXTVAL WHERE a = 2;
SQL> SELECT * FROM seq_tab1;
```

다음은 Sequence를 사용할 수 없는 예이다.

```
SQL> CREATE TABLE seq_tab(a NUMBER DEFAULT s1.NEXTVAL);
(* Error 발생)
SQL> SELECT DISTINCT s1.NEXTVAL FROM dual;
(* Error 발생)
SQL> SELECT SUM(salary)FROM s_emp GROU BY s1.NEXTVAL;
(* Error 발생)
```

## ALTER SEQUENCE

- To change the increment, minimum and maximum values, cached numbers, and behavior of an existing sequence.
- This statement affects only future sequence numbers.
- Syntax

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

BIT Academy Course

다음을 실행하면서 실행 결과와 Error의 이유를 적으시오.

```
SQL> CREATE SEQUENCE s2
      START WITH 5 INCREMENT BY 10 CACHE 2;
SQL> SELECT s2.CURRVAL FROM dual;
(* Error 발생)
SQL> CREATE TABLE seq_tab2(a NUMBER, b CHAR);
SQL> INSERT INTO seq_tab2 VALUES (s2.NEXTVAL, 'A');
SQL> SELECT s2.CURRVAL FROM dual;
SQL> SELECT * FROM seq_tab2;
SQL> INSERT INTO seq_tab2 VALUES(s2.NEXTVAL, 'B');
SQL> SELECT s2.CURRVAL FROM dual;
SQL> SELECT s2.NEXTVAL FROM dual;
SQL> INSERT INTO seq_tab2 VALUES (s2.NEXTVAL, 'C');
SQL> SELECT * FROM seq_tab2;
SQL> ALTER SEQUENCE s2
      MINVALUE 0 MAXVALUE 40 CYCLE;
SQL> INSERT INTO seq_tab2 VALUES (s2.NEXTVAL, 'D');
SQL> SELECT * FROM seq_tab2;
SQL> INSERT INTO seq_tab2 VALUES (s2.NEXTVAL, 'X');
SQL> / ( 네 번 실행)
SQL> SELECT * FROM seq_tab2;
```

## DROP SEQUENCE

- To remove a sequence from the database.
- You can also use this statement to restart a sequence by dropping and then re-creating it.
- The sequence must be in your own schema or you must have the DROP ANY SEQUENCE system privilege.
- Syntax

```
DROP SEQUENCE schema.sequence_name;
```

BIT Academy Course

Sequence가 삭제된 후에는 참조할 수 없다.

```
SQL> DROP SEQUENCE s2;
```

```
SQL> INSERT INTO seq_tab2 VALUES (s2.NEXTVAL, 'X');
```

(\* Error 발생)

## Dictionary for Sequences

- USER\_SEQUENCES
- USER\_OBJECTS
- Example

```
SELECT object_name
FROM   user_objects
WHERE  object_type = 'SEQUENCE';
```

```
SELECT sequence_name, min_value,
       max_value, increment_by,
       last_number
FROM   user_sequences;
```

BIT Academy Course

Dictionary에서 Sequence에 대한 정보를 조회해 보시오.

```
SQL> CREATE SEQUENCE s3;
```

```
SQL> SELECT object_name
       FROM   user_objects
       WHERE  object_type = 'SEQUENCE';
```

```
SQL> SELECT sequence_name, min_value, max_value, increment_by,
       last_number
       FROM   user_sequences
       WHERE  sequence_name='S3';
```

```
SQL> SELECT s3.NEXTVAL FROM dual;
```

```
SQL> SELECT sequence_name, min_value, max_value,
       increment_by, last_number
       FROM   user_sequences
       WHERE  sequence_name='S3';
```

(\* last\_number는 Oracle server가 다음에 발행할 값을 의미한다.)

## Synonym

- A synonym is an alias for a table, view, sequence, or program unit.
- A synonym can be public or private.
- A synonym is not a schema object, but a direct reference to a schema object used for :
  - mask the real name and owner of a schema object.
  - provide public access to a schema object.
  - provide location transparency for tables, views, or program units of a remote database.
  - simplify the SQL statements for database users.

## CREATE/DROP SYNONYM

- Syntax

```
CREATE [PUBLIC] SYNONYM
        synonym FOR object;
```

```
DROP [PUBLIC] SYNONYM synonym;
```

- Example

```
CREATE SYNONYM emp FOR scott.emp;
DROP SYNONYM emp;
```

BIT Academy Course

Public synonym은 권한이 있는 user만이 만들 수 있으며, 전체 user들이 사용할 수 있다.

Synonym을 만들어 data를 조회해 보시오.

```
SQL> conn system/manager
SQL> SELECT * FROM s_emp;
(* Error 발생)
SQL> SELECT * FROM testxx.s_emp;
(* system user는 SELECT ANY TABLE 권한을 가지고 있으므로 성공)
SQL> CREATE SYNONYM s_emp FOR testxx.s_emp;
SQL> SELECT * FROM s_emp;
SQL> CREATE TABLE s_emp (a number);
(* Error 발생)
```

Base table의 이름이 바뀌면 Synonym은 더 이상 사용할 수 없게 된다.


```
SQL> conn testxx/testxx
SQL> RENAME s_emp TO e;
SQL> conn system/manager
SQL> SELECT * FROM s_emp;
(* Error 발생)
SQL> conn testxx/testxx
SQL> RENAME e TO s_emp;
```

### Dictionary for Synonyms

- USER\_OBJECTS
- USER\_SYNONYMS
- Example

```
SELECT object_name, status  
FROM user_objects  
WHERE object_type = 'SEQUENCE';
```

```
SELECT * FROM user_synonyms;
```



Synonym에 대한 정보를 담고 있는 Dictionary View를 찾아 조회해 보시오.

```
SQL> conn system/manager  
SQL> SELECT * FROM dictionary  
WHERE table_name LIKE '%SYNONYM%';  
  
SQL> COL object_name FORMAT A30  
SQL> SELECT object_name, status FROM user_objects  
WHERE object_type = 'SEQUENCE';  
  
SQL> COL synonym_name FORMAT A20  
SQL> COL table_owner FORMAT A20  
SQL> COL table_name FORMAT A20  
SQL> COL db_link FORMAT A20  
SQL> SELECT * FROM user_synonyms;
```

DUAL이 Public Synonym임을 확인하시오.

```
SQL> COL owner FORMAT A20  
SQL> SET linesize 120  
SQL> SELECT * FROM dba_synonyms  
WHERE synonym_name = 'DUAL' ;
```



# Data Control

- Oracle Database의 User와 Schema
  - CREATE USER
  - ALTER USER
- System Privilege, Object Privilege  
종류 및 제어 방법
  - GRANT, REVOKE
  - WITH ADMIN OPTION
  - WITH GRANT OPTION
- Role의 활용
  - CREATE ROLE
- 관련 Dictionary
  - ROLE\_SYS\_PRIVS
  - ROLE\_TAB\_PRIVS
  - USER\_ROLE\_PRIVS
  - USER\_TAB\_PRIVS\_MADE
  - USER\_TAB\_PRIVS\_RECD
  - USER\_COL\_PRIVS\_RECD

## Database User

- A user is a name defined in the database that can connect to and access objects.
- A schema is a named collection of objects associated with a particular user.
- A user is associated only with the schema of the same name.
- A session is a specific connection of a user to an Oracle instance via a user process.
- Multiple sessions can be created and exist concurrently for a single Oracle user using the same username.

BIT Academy Course

Oracle Database 생성 시 SYS, SYSTEM user는 기본적으로 만들어진다.

SYS는 Data Dictionary table의 소유자이며, SYSTEM은 각종 Dictionary view의 소유자이다.

각 user는 동일한 이름의 schema 소유자로서 해당 schema의 모든 object들에 접근할 수 있다.

## Manage User

- To create and configure a database user, you must have CREATE USER system privilege.
- A user must have CREATE SESSION system privilege to log on to Oracle.
- Syntax

```
CREATE USER user IDENTIFIED BY passwd;  
ALTER USER user IDENTIFIED BY new_passwd;  
DROP USER user [CASCADE];
```
- Example

```
CREATE USER userxx IDENTIFIED BY userxx;
```

BIT Academy Course

user를 생성하고 privilege를 주지 않으면 아무런 일도 할 수 없다.

```
SQL> conn system/manager  
SQL> CREATE USER userxx IDENTIFIED BY userxx;  
(* xx 는 각자 고유번호 부여)  
SQL> conn userxx/userxx  
ERROR:  
ORA-01045: user USERXX lacks CREATE SESSION privilege; logon denied
```

user는 DBA가 초기화한 password를 갖게 되며 ALTER USER 명령을 써서 각자의 password를 관리한다.

testxx 사용자의 password를 변경해 보시오.

```
SQL> conn testxx/testxx  
SQL> ALTER USER testxx IDENTIFIED BY xx;  
SQL> conn testxx/xx  
  
SQL> ALTER USER testxx IDENTIFIED BY testxx;
```

## Dictionary for Users

- USER\_USERS
  - Information about the current user
- ALL\_USERS
  - Information about all users of the database
- DBA\_USERS
  - Information about all users of the database
- Example

```
SELECT username FROM user_users;
```

BIT Academy Course

모든 user는 USER\_USERS, ALL\_USERS dictionary view를 조회할 수 있다.  
DBA 권한을 가진 user 만이 DBA\_USERS dictionary view를 조회할 수 있다.

현재의 database 내에 생성된 user를 조회하시오.

```
SQL> conn testxx/testxx
SQL> SELECT username FROM user_users;
SQL> SELECT username FROM all_users;
SQL> SELECT username FROM dba_users;
(* Error 발생)
SQL> conn system/manager
SQL> /
(* system user가 DBA 권한을 가지고 있어 조회 성공)
```

## Privileges

- A privilege is a right to execute a particular type of SQL statement or to access another user's object.
- Users should be granted to related privileges required for tasks.
- Types of Privileges
  - system privileges
  - schema object privileges

## System Privileges

- A system privilege is the right to perform a particular action.
- There are over 80 distinct system privileges.
- Example
  - CREATE SESSION
  - CREATE TABLE
  - SELECT ANY TABLE
  - EXECUTE ANY PROCEDURE
  - DROP ANY TABLE

## Control System Privileges

- To grant system privileges and roles to users and roles.
- Only users who have been granted a specific system privilege with the ADMIN OPTION or users with the GRANT ANY PRIVILEGE system privilege can grant or revoke system privileges to users and roles.
- To grant a role, you must either have been granted the role with the ADMIN OPTION or have been granted the GRANT ANY ROLE system privilege, or you must have created the role.
- GRANT statement is one transaction.

## Control System Privileges(Cont.)

- Syntax

```
GRANT {system_priv | role [, system_priv | role ...]}
TO {user [,user...] | role | PUBLIC }
[WITH ADMIN OPTION];
```

```
REVOKE {system_priv | role[, system_priv | role...]}
FROM {user [, user...] | role | PUBLIC};
```

- Example

```
GRANT create session TO userxx;
```

```
REVOKE create session FROM userxx;
```

BIT Academy Course

Oracle에 Session을 맺을 수 있는 Privilege를 준다.

```
SQL> conn system/manager
SQL> GRANT create session TO userxx;
SQL> conn userxx/userxx
(* Oracle에 접속 성공)
```

```
SQL> CREATE TABLE x (a NUMBER);
(* Error 발생. 그 이유를 설명하시오.)
```

userxx 사용자에게 connect, resource라는 role을 부여하면 table 생성 등을 할 수 있다.

```
SQL> conn system/manager
SQL> GRANT connect, resource TO userxx;
SQL> conn userxx/userxx
SQL> CREATE TABLE x (a NUMBER);
(* 성공)
```

GRANT, REVOKE 문장은 실행 즉시 효력을 갖는다.

```
SQL> conn system/manager
SQL> GRANT select any table TO userxx;
SQL> conn userxx/userxx
SQL> SELECT * FROM scott.emp;
SQL> SELECT * FROM testxx.s_emp;
```

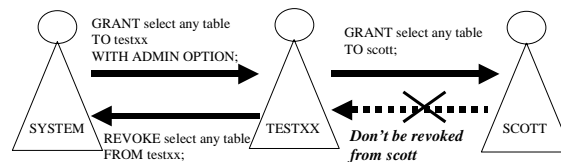


**Oracle 9i 실행(SQL, PL/SQL) >>>**

```
SQL> conn system/manager
SQL> REVOKE select any table FROM userxx;
SQL> conn userxx/userxx
SQL> SELECT * FROM scott.emp;
(* Error 발생)
```

## WITH ADMIN OPTION

- Allows the grantee to grant the system privileges to other users and roles.
- System privileges granted with this option are not revoked when the grantor's privilege is revoked.



BIT Academy Course

WITH ADMIN OPTION에 대한 실습이다. 각 문장의 수행 결과를 설명하시오.

SESSION #1		SESSION #2	
1	Conn system/manager	2	Conn testxx/testxx
3	CREATE TABLE t (a NUMBER); INSERT INTO t VALUES (10);		
4	GRANT select any table TO testxx WITH ADMIN OPTION;	5	SELECT * FROM system.t;
		6	GRANT select any table TO scott;
7	REVOKE select any table FROM testxx;	8	SELECT * FROM system.t; Conn scott/tiger SELECT * FROM system.t;
9	GRANT select any table TO testxx,scott WITH ADMIN OPTION;	10	REVOKE select any table FROM testxx; Conn testxx/testxx SELECT * FROM system.t;
11	REVOKE select any table FROM scott;	12	Conn scott/tiger SELECT * FROM system.t;

## Object Privileges

- A schema object privilege is a privilege or right to perform a particular action on a *specific* table, view, sequence, procedure, function, or package.
- Different object privileges are available for different types of schema objects.
- A schema object and its synonym are equivalent with respect to privileges.

## Object Privileges(Cont.)

- Syntax

```
GRANT {object_priv [, object_priv ...] | ALL}
ON object TO {user [, user...] | role | PUBLIC}
[WITH GRANT OPTION];
```

```
REVOKE {object_priv [, object_priv...] | ALL}
ON object FROM {user [, user ...] | role | PUBLIC};
```

- Example

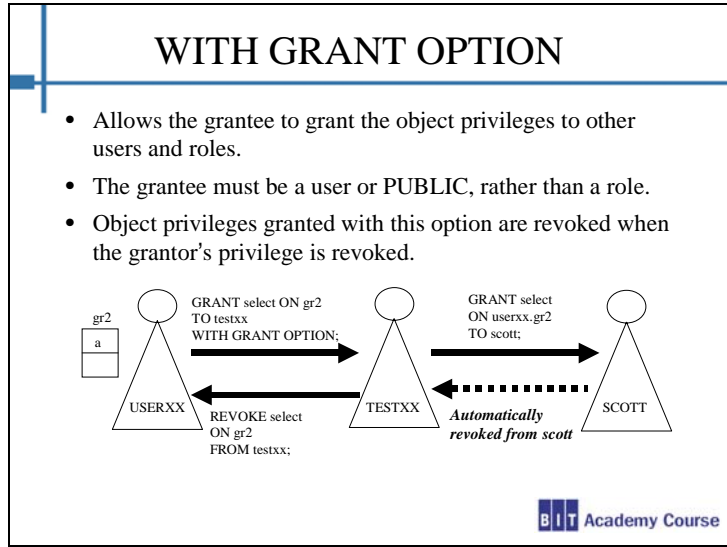
```
GRANT select ON emp TO userxx;
```

```
REVOKE select ON emp FROM userxx;
```

 BIT Academy Course

SQL\*Plus를 2개 띄워 접속한 후 아래 문장들을 번호 순서대로 테스트하면서 각 문장의 실행 결과를 적으시오.

SESSION #1		SESSION #2	
1	Conn userxx/userxx	2	Conn testxx/testxx
3	CREATE TABLE gr1 (a NUMBER); INSERT INTO gr1 VALUES (10);	4	SELECT * FROM userxx.gr1;
5	GRANT select ON gr1 TO testxx;	6	/
		7	DELETE FROM userxx.gr1;
8	GRANT all ON gr1 TO testxx;	9	/
10	REVOKE all ON gr1 FROM testxx;	11	SELECT * FROM userxx.gr1;
12	SELECT * FROM gr1;	13	COMMIT;
14	/		

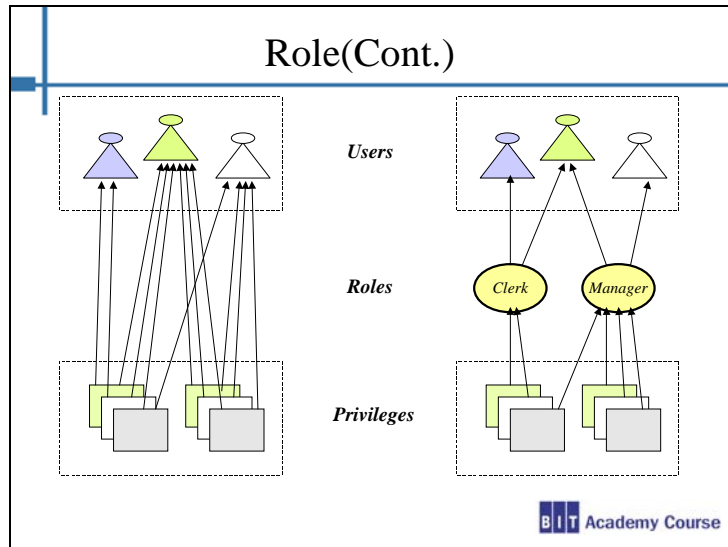


WITH GRANT OPTION에 대한 실습이다. 각 문장의 수행 결과를 설명하시오.

SESSION #1		SESSION #2	
1	Conn userxx/userxx	2	Conn testxx/testxx
3	CREATE TABLE gr2 (a NUMBER); INSERT INTO gr2 VALUES (10);	4	SELECT * FROM userxx.gr2;
5	GRANT select ON gr2 TO testxx WITH GRANT OPTION;	6	/ GRANT select ON userxx.gr2 TO scott;
		7	Conn scott/tiger SELECT * FROM userxx.gr2;
8	REVOKE select ON gr2 FROM scott;	9	SELECT * FROM userxx.gr2;
		10	Conn testxx/testxx SELECT * FROM userxx.gr2
11	REVOKE select ON gr2 FROM testxx;	12	SELECT * FROM userxx.gr2;
		13	Conn scott/tiger SELECT * FROM userxx.gr2;

## Role

- Oracle provides for easy and controlled privilege management through roles.
- Roles are named groups of related privileges.
- A role can be granted system or schema object privileges.
- A role can be granted to other roles.
- Any role can be granted to any database user.



## Role(Cont.)

- Usage

```
CREATE ROLE role_name;
GRANT privs TO role_name;
GRANT role_name TO user;
```

- Example

```
CREATE ROLE mgr;
GRANT select any table TO mgr;
GRANT mgr TO userxx;
```

BIT Academy Course

role을 생성하여 user에게 grant, revoke 해 보시오.

```
SQL> conn system/manager
SQL> CREATE ROLE mgr;
SQL> GRANT select any table, update any table TO mgr;
SQL> GRANT mgr TO userxx;
```

```
SQL> conn userxx/userxx
SQL> SELECT * FROM testxx.s_dept;
SQL> UPDATE testxx.s_emp SET salary = 0;
SQL> ROLLBACK;
```

```
SQL> conn system/manager
SQL> REVOKE mgr FROM userxx;
```


```
SQL> conn userxx/userxx
SQL> SELECT * FROM testxx.s_dept;
```



### Dictionary for Privileges

- **ROLE\_SYS\_PRIVS**
  - System privileges granted to roles
- **ROLE\_TAB\_PRIVS**
  - Table privileges granted to roles
- **ROLE\_ROLE\_PRIVS**
  - Roles which are granted to roles
- **USER\_ROLE\_PRIVS**
  - Roles granted to current user
- **Example**

```
SELECT * FROM user_role_privs;
```



userxx가 부여받은 Role을 조회한다.

```
SQL> conn userxx/userxx
```

```
SQL> SELECT * FROM user_role_privs;
```

CONNECT, RESOURCE 라는 Role에 부여된 system privilege를 조회한다.

```
SQL> SELECT * FROM role_sys_privs
WHERE role IN('CONNECT', 'RESOURCE');
```

다음은 Synonym을 사용하는 실습이다. 결과를 적으면서 따라 해 보시오.

SESSION #1		SESSION #2	
1	Conn userxx/userxx	2	Conn testxx/testxx
		3	CREATE SYNONYM gr2 FOR userxx.gr2;
		4	SELECT * FROM gr2;
5	GRANT update(a) ON gr2 TO testxx;	6	SELECT * FROM gr2; UPDATE gr2 SET a = 200;
7	GRANT select ON gr2 TO testxx;	8	SELECT * FROM gr2;
9	SELECT * FROM gr2;	10	COMMIT;
11	/		
12	REVOKE all ON gr2 FROM testxx;	13	SELECT * FROM gr2;

## Dictionary for Privileges(cont.)

- USER\_TAB\_PRIVS\_MADE
  - All grants on objects owned by the user
- USER\_COL\_PRIVS\_MADE
  - All grants on columns of objects owned by the user
- USER\_TAB\_PRIVS\_RECD
  - Grants on objects for which the user is the grantee
- USER\_COL\_PRIVS\_RECD
  - Grants on columns for which the user is the grantee
- Example

```
SELECT * FROM user_tab_privs_made
WHERE table_name = 'S_EMP';
```

BIT Academy Course

주고 받은 object privilege를 조회한다.

```
SQL> conn testxx/testxx
SQL> SELECT * FROM user_tab_privs_made;
SQL> SELECT * FROM user_col_privs_made;
SQL> GRANT select ON s_emp TO userxx WITH GRANT OPTION;
SQL> GRANT update(salary) ON s_emp TO userxx;
SQL> SELECT * FROM user_tab_privs_made;
SQL> SELECT * FROM user_col_privs_made;

SQL> conn userxx/userxx
SQL> SELECT * FROM user_tab_privs_recd;
SQL> SELECT * FROM user_col_privs_recd;
```



# PL/SQL Program

## (Introduction)

- PL/SQL Program의 장점
- PL/SQL Program의 종류
  - ANONYMOUS Block
  - Stored Procedure
  - Stored Function

## PL/SQL

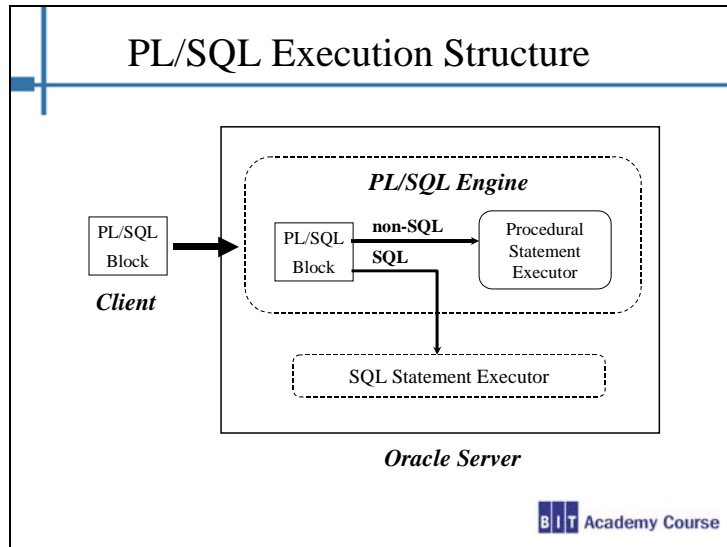
- PL/SQL is Oracle's procedural language extension to SQL.
- It extends SQL with flow control and other statements that make it possible to write complex programs in it.
- PL/SQL program units can be defined and executed as procedures, functions, and packages.
- The *PL/SQL engine* is the tool you use to define, compile, and execute PL/SQL program units.

## Advantages of PL/SQL

- Oracle must process SQL statements one at a time.
- But, with PL/SQL, an entire block of statements can be sent to Oracle at one time.
- This can reduce communication between your application and Oracle.
- Especially if your application is database intensive, using PL/SQL can be effective.

## PL/SQL Block Structure

- PL/SQL is a *block-structured* language.
- The basic units are logical blocks, which can contain any number of nested sub-blocks.
  - [DECLARE]      - *Optional*
    - Variable, constant, Cursor, user - defined exception
  - BEGIN              - *Mandatory*
    - SQL statement
    - PL/SQL control statement
  - [EXCEPTION] - *Optional*
    - Error handling
  - END;                - *Mandatory*



Oracle server에는 PL/SQL Block을 처리할 수 있는 엔진이 장착되어 있어 절차적 문장은 Procedural Statement Executor에서 처리를 하고, Block 안에서 사용된 SQL 문장은 SQL Statement Executor로 보내어 data 처리를 하도록 한다.

## PL/SQL Category

- Anonymous blocks
  - It appears within your application.
  - It is not named or stored in the database
  - Oracle compiles the PL/SQL block and places the compiled version in the shared pool of the SGA, but does not store the source code or compiled version.
- Stored programs
  - It is created and stored in the database as a schema object.
  - It can be called by name from an application.
  - Once created and compiled, it is a named object that can be executed without recompiling.



## SQL\*Plus Commands

- **ACCEPT**
  - send messages to the end user and accept values as end-user input.
- **VARIABLE**
  - Declares a bind variable that can then be referenced in PL/SQL.
- **PRINT**
  - Displays the current value of bind variables.
- **EXECUTE**
  - Executes a single PL/SQL statement.

## SQL\*Plus Commands(cont.)

- ACCEPT
  - ACCEPT *variable* [*datatype*] PROMPT *text* [HIDE]
  - DEF[INE] [*variable*][*variable* = *text*]
- VARIABLE
  - VAR[iable] [*variable*] [NUMBER|CHAR|CHAR (*n*)  
|VARCHAR2 (*n*)|NVARCHAR2 (*n*)]
- PRINT
  - PRI[NT] [*variable* ...]
- EXECUTE
  - EXEC[UTE] *statement*

## PL/SQL Block Debugging

- Anonymous block
  - Errors are displayed on the screen.
- Stored program
  - Oracle saves error messages occurred during compiling stored procedures and functions into USER\_ERRORS dictionary table.

```
SQL> SHOW ERRORS
SQL> SELECT *
      FROM user_errors;
```

BIT Academy Course

Error가 났을 때 LINE 번호와 COLUMN 번호, ERROR 내용을 확인한 후 source를 고친다.

```
SQL> SHOW ERRORS
```

```
LINE/COL ERROR
```

```
-----
8/13          PL/SQL: SQL Statement ignored
8/20          PLS-00201: 'PRODUCT_ID.PRICE' ....
```

## Anonymous Block

- Example

```
SQL> EDIT anony
ACCEPT p_sal PROMPT 'Enter the salary:'
VARIABLE g_annual_sal NUMBER
DECLARE
  v_sal NUMBER := &p_sal;
BEGIN
  :g_annual_sal := &p_sal*12;
END;
/
PRINT g_annual_sal
SQL> @anony
```

*SQL\*Plus commands* (points to ACCEPT, VARIABLE, PRINT, and SQL> @anony)

*PL/SQL Block* (points to the DECLARE...BEGIN...END; section)

BIT Academy Course

다음과 같이 Anonymous Block을 작성하여 실행해 보시오.

```
SQL> EDIT anony
ACCEPT p_sal PROMPT 'Enter the salary:'
VARIABLE g_annual_sal NUMBER
DECLARE
  v_sal NUMBER := &p_sal;
BEGIN
  :g_annual_sal := &p_sal*12;
END;
/
PRINT g_annual_sal
SQL> @anony
```

## Stored Program

- Types
  - Stored Procedure
  - Stored Function
  - Package
- *Procedures and functions* are identical, except that functions always **return a single value** to the caller, while procedures do not return a value to the caller.
- *Packages* provide a method of encapsulating and storing related procedures, functions, and other package constructs together as a unit in the database.

## Stored Procedure

- Syntax

```
CREATE OR REPLACE PROCEDURE name
    [(Parameter, ...)]
IS
    PL/SQL Block;
```

- Parameter Syntax

```
parameter_name [IN | OUT | IN OUT] datatype
    [{ := | DEFAULT} expr]
```

BIT Academy Course

### Parameter

- 실행환경과 Subprogram 사이의 값을 전달
- 종류
  - IN : 실행환경에서 procedure로 값 전달  
(Default)
  - OUT : procedure에서 실행환경으로 값을 전달
  - IN OUT : 실행환경에서 procedure로 값을 전달하고 procedure에서 실행환경으로 값을 전달

## Stored Procedure(cont)

- Execute procedures in the SQL\*Plus.
- Put the procedure name with argument values at the SQL prompt.
- Example

```
SQL> EXEC change_salary (1,10000);
```

## Stored Procedure(cont.)

- Example

```
SQL> EDIT p1
CREATE OR REPLACE PROCEDURE change_salary
(p_id IN NUMBER, p_new_sal IN NUMBER)
IS
BEGIN
    UPDATE s_emp
    SET salary = p_new_sal
    WHERE id = p_id;
    COMMIT;
END change_salary ;
/
SQL> @p1
SQL> EXEC change_salary (1,10000);
```

BIT Academy Course

아래와 같이 change\_salary procedure를 생성하고 실행해 보시오.

```
SQL> conn testxx/testxx
SQL> EDIT p1
CREATE OR REPLACE PROCEDURE change_salary
(p_id IN NUMBER, p_new_sal IN NUMBER)
IS
BEGIN
    UPDATE s_emp
    SET salary = p_new_sal
    WHERE id = p_id;
    COMMIT;
END change_salary ;
/
SQL> @p1
(* Error 발생 시 SHOW ERRORS 명령으로 확인하여 수정한 후 다시 생성)

SQL> SELECT id, salary FROM s_emp WHERE id = 1;
(* 현재의 salary 확인)
SQL> EXEC change_salary(1, 5000);
SQL> SELECT id, salary FROM s_emp WHERE id = 1;
(* 변경된 salary 확인)
```



## Stored Function

- Syntax

```
CREATE OR REPLACE FUNCTION name  
    [(Parameter, ...)]  
    RETURN datatype  
  
IS  
    PL/SQL Block;
```

- Function can be finished successfully after execute a return command.
- There is no needs for OUT or IN OUT mode parameters because of a RETURN value.

## Stored Function(cont)

- Execute functions in the SQL\*Plus.
- Define a bind variable for a return value from the function.
- Put the function name with argument values at the SQL prompt.
- Example

```
SQL> VARIABLE g_sal NUMBER  
SQL> EXEC :g_sal := get_salary (1);  
SQL> PRINT g_sal
```

## Stored Function(cont.)

- Example

```
SQL> EDIT f1
CREATE OR REPLACE FUNCTION get_salary
  (p_id IN NUMBER) RETURN NUMBER
IS
  v_sal NUMBER;
BEGIN
  SELECT salary INTO v_sal
    FROM s_emp WHERE id = p_id;
  RETURN v_sal;
END get_salary ;
/
SQL> @f1
SQL> VARIABLE g_sal NUMBER
SQL> EXEC :g_sal := get_salary (1);
SQL> PRINT g_sal
```

 Academy Course

아래와 같이 get\_salary function을 생성하고 실행해 보시오.

```
SQL> EDIT f1
CREATE OR REPLACE FUNCTION get_salary
  (p_id IN NUMBER)
  RETURN NUMBER
IS
  v_sal NUMBER;
BEGIN
  SELECT salary
    INTO v_sal
    FROM s_emp
    WHERE id = p_id;
  RETURN v_sal;
END get_salary ;
/
SQL> @f1
(* Error 발생 시 SHOW ERRORS 명령으로 확인하여 수정한 후 다시 생성)

SQL> VARIABLE g_sal NUMBER
SQL> EXEC :g_sal := get_salary (1);
SQL> PRINT g_sal
```

## Usages of Stored Function

- Example

```
SQL> SELECT get_salary(2)
       FROM dual;
SQL> SELECT *
       FROM s_emp
       WHERE salary > get_salary(2);
SQL> INSERT INTO s_emp (id, last_name, salary)
       VALUES (100, USER, get_salary(2));
```

Dictionary for Stored Program

- USER\_OBJECTS
- USER\_SOURCE
- Example

```
SELECT object_name, object_type, status
FROM user_objects
WHERE object_type IN
      ('PROCEDURE','FUNCTION');
```

BIT Academy Course

dictionary view를 이용하여 생성한 Stored program의 상태를 조회해 보시오.

```
SQL> SELECT object_name, object_type, status
      FROM user_objects
      WHERE object_type IN ('PROCEDURE','FUNCTION');
```

dictionary view를 이용하여 생성한 Stored program의 source를 조회해 보시오.

```
SQL> SET long 1000
SQL> SELECT text      FROM user_source
      WHERE name = 'CHANGE_SALARY'
      ORDER BY line;
```

Procedure나 Function을 호출할 때 필요한 parameter를 확인하시오.

```
SQL> desc change_salary
SQL> desc get_salary
```

# PL/SQL Program

(Variables, SQL, PL)

- 변수, 상수 선언

- Scalar
- Table
- Record

- SQL 문장

- DML
- Transaction Control 문장

- IF 문

- IF THEN ELSE IF THEN ELSE END IF;

- LOOP 문

- LOOP END LOOP
- WHILE LOOP END LOOP
- FOR IN LOOP END LOOP

## Declare Variables

- Syntax

```
Identifier [CONSTANT] Datatype [NOT NULL]  
[:= | DEFAULT expr];
```

- Follow Oracle's naming rule.
- One variable or constant per one line.
- Define initial values for the NOT NULL variables or constant using assignment operator (:=) or DEFAULT keyword.

## Scalar Data Type

- A *scalar* type has no internal components.
  - BINARY\_INTEGER
  - NUMBER (*p,s*)
  - CHAR [(*length*)]
  - LONG
  - LONG RAW
  - VARCHAR2(*length*)
  - DATE
  - BOOLEAN



## Scalar Data Type(cont.)

- Example : Define variables.
  - v\_gender            CHAR(1);
  - v\_count            BINARY\_INTEGER:=0;
  - v\_total\_sal        NUMBER(9,2):=0;
  - v\_order\_date       DATE:=SYSDATE+7;
  - v\_tax\_rate          NUMBER(3,2):=8.25;
  - v\_valid            BOOLEAN NOT NULL:=TRUE;
- Example : Define constants.
  - c\_female           CONSTANT CHAR(1) := 'F';
  - c\_male             CONSTANT CHAR(1) DEFAULT 'M';
  - c\_limit            CONSTANT BINARY\_INTEGER := 10;
  - c\_current\_date     CONSTANT DATE := SYSDATE;

 BIT Academy Course

변수에 초기값을 주지 않으면 NULL이다.

## Composite Data Type

- A *composite* type has internal components that can be manipulated individually.
  - TABLE
    - Collection type like one-dimensional array.
    - It is unbounded so, the size can increase dynamically.
    - It can have nonconsecutive subscripts.
  - RECORD
    - It is a group of related data items stored in *fields*, each with its own name and datatype.

## TABLE Type Variables

- Syntax

```
DECLARE
  TYPE type_name IS TABLE OF datatype
                        [NOT NULL]
                        INDEX BY BINARY_INTEGER;
  identifier type_name;
```

- Example

```
DECLARE
  TYPE name_table IS TABLE OF VARCHAR2(25)
                        INDEX BY BINARY_INTEGER;
  v_emp_name      name_table;
  v_cust_name     name_table;
```

## RECORD Type Variables

- Syntax

```
DECLARE
  TYPE type_name IS RECORD
    (field1 datatype [NOT NULL {:=|DEFAULT} expr],
     (field2 datatype [NOT NULL {:=|DEFAULT} expr], ...);
  identifier type_name;
```

- Example

```
DECLARE
  TYPE emp_record IS RECORD
    (last_name VARCHAR2(25),
     salary    NUMBER(11,2));
  v_employee emp_record;
```

## %TYPE Attribute

- Provides the datatype of a variable or database column.
- Example

```
DECLARE
  v_last_name      s_emp.last_name%TYPE;
  v_salary          s_emp.salary%TYPE;
  v_counter         NUMBER(3);
  v_low_counter     v_counter%TYPE:=0;
```

BIT Academy Course

다음을 실행하십시오.

```
SQL> ed a1
DECLARE
    v_last_name s_emp.last_name%TYPE;
    v_first_name s_emp.first_name%TYPE;
BEGIN
    SELECT last_name, first_name
    INTO v_last_name, v_first_name
    FROM s_emp
    WHERE id = 1;

    DBMS_OUTPUT.PUT_LINE(v_last_name);
    DBMS_OUTPUT.PUT_LINE(v_first_name);
END;
/
SQL> SET SERVEROUTPUT ON
SQL> @a1
```

## %ROWTYPE Attribute

- Provides a record type that represents a row in a table.
- Example

```
DECLARE
  v_dept      s_dept%ROWTYPE;
  v_emp       s_emp%ROWTYPE;
```

 Academy Course

다음을 실행해 보시오.

```
SQL> ed a2
DECLARE
    employee_record  s_emp%ROWTYPE;
BEGIN
    SELECT *
    INTO employee_record
    FROM s_emp
    WHERE id = 1;

    DBMS_OUTPUT.PUT_LINE(employee_record.last_name);
    DBMS_OUTPUT.PUT_LINE(employee_record.first_name);
    DBMS_OUTPUT.PUT_LINE(employee_record.salary);
END;
/
SQL> @a2
```

## Assign Values to Variables

- Use Assignment Operator.

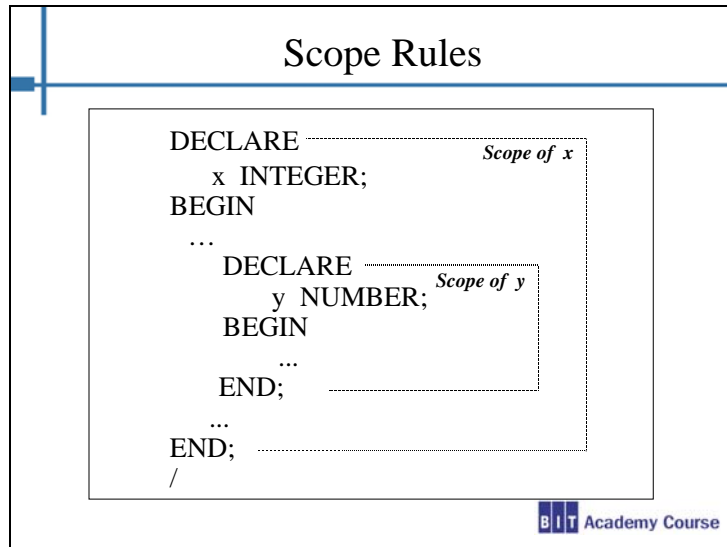
– Example

```
v_counter := 10;  
v_emp_name(2) := 'SCOTT';  
v_employee.salary := 2000;
```

- Select or fetch database values into variables.

– Example

```
SELECT last_name, salary  
INTO v_last_name, v_salary  
FROM s_emp  
WHERE id = 1;
```



위에서 x 변수는 전체 block에 걸쳐 사용 가능한 변수이다. 그러나, y변수는 안쪽 block에서만 사용이 가능한 변수이다.

아래에서 변수 x의 값을 예측하시오.

```
SQL> ed a3
DECLARE
    x NUMBER := 10;
BEGIN
    DBMS_OUTPUT.PUT_LINE(x);  -- (1)
    x := x + 10;
    DECLARE
        x NUMBER := 100;
    BEGIN
        DBMS_OUTPUT.PUT_LINE(x);  -- (2)
        x := x + 100;
    END;
    DBMS_OUTPUT.PUT_LINE(x);  -- (3)
END;
/
SQL> @a3
```



## Operators in PL/SQL

- All arithmetic , comparison operators and logical operators in SQL statements are allowed in PL/SQL blocks.
- You can use a exponentiation operator (\*\*).

BIT Academy Course

**다음**을 실행해 보세요.

```
SQL> ed a4
DECLARE
    x NUMBER;
    y VARCHAR2(100);
    z DATE := sysdate;
BEGIN
    x := 1 + 3 * 2;
    DBMS_OUTPUT.PUT_LINE(x);
    x := (1 + 3) * 2;
    DBMS_OUTPUT.PUT_LINE(x);
    x := x ** 2;
    DBMS_OUTPUT.PUT_LINE(x);
    y := 'ABC' || 'DEF';
    DBMS_OUTPUT.PUT_LINE(y);
    y := y || y;
    DBMS_OUTPUT.PUT_LINE(y);
    z := z + 7;
    DBMS_OUTPUT.PUT_LINE(z);
END;
/
SQL> @a4
```

## Functions in PL/SQL

- Available Functions
  - Single-Row Number Functions
  - Single-Row Character Functions
  - Single-Row Date Functions
  - Single-Row Data Type Conversion Functions
- Not available Functions
  - GREATEST, LEAST
  - Aggregate Functions : SUM, AVG, MIN, MAX, COUNT etc.

## SQL in PL/SQL

- PL/SQL supports Data Manipulation and Transaction Control Languages.
- PL/SQL doesn't support Data Definition Languages such as CREATE TABLE and DROP TABLE.
- PL/SQL doesn't support Data Control Languages such as GRANT and REVOKE.

BIT Academy Course

PL/SQL block에 DDL, DCL 문장을 사용하면 Error가 발생한다.

```
SQL> BEGIN
2   CREATE TABLE xxx ( a NUMBER);
3 END;
4 /
(* Error 발생)
```

## SELECT in PL/SQL

- Query data from the database by using SELECT.

```
SELECT  select_list  
INTO    variable_name | record_name  
FROM    table  
WHERE   condition ;
```

- INTO clause is necessary.
- Only one row should be returned.
- All SELECT syntax can be used.
- SQL statement should be ended with semicolon (;).

## SELECT in PL/SQL(cont.)

- Example

```
CREATE OR REPLACE PROCEDURE emp_info
    (p_id s_emp.id%TYPE)
IS
    v_last_name  s_emp.last_name%TYPE;
    v_salary     s_emp.salary%TYPE;
BEGIN
    SELECT  last_name, salary
    INTO    v_last_name, v_salary
    FROM    s_emp
    WHERE   id = p_id;
    .....
END emp_info;
/
```

BIT Academy Course

다음을 실습해 보시오.

```
SQL> ed p2
CREATE OR REPLACE PROCEDURE emp_info
    (p_id  s_emp.id%TYPE)
IS
    v_last_name  s_emp.last_name%TYPE;
    v_salary     s_emp.salary%TYPE;
BEGIN
    SELECT  last_name, salary
    INTO    v_last_name, v_salary
    FROM    s_emp
    WHERE   id = p_id;

    DBMS_OUTPUT.PUT_LINE(v_last_name);
    DBMS_OUTPUT.PUT_LINE(v_salary);
END emp_info;
/
SQL> @p2
SQL> EXEC emp_info (1);
SQL> EXEC emp_info (99999);
```

## Exceptions from SELECT

- SELECT Exceptions
  - Only one row should be returned with SELECT statement in PL/SQL.
  - IF more over one rows or none of rows are returned, exceptions are occurred.
- Exceptions related SELECT:
  - TOO\_MANY\_ROWS (ORA -01422)
  - NO\_DATA\_FOUND (ORA-01403)

BIT Academy Course

PL/SQL block에서 SELECT 문장을 사용하게 될 때는 반드시 EXCEPTION 처리를 해준다.

```
SQL> ed a5
SET VERIFY OFF
ACCEPT p_dept_id PROMPT '부서번호 : '

DECLARE
    v_sal          s_emp.salary%TYPE;
    v_last_name    s_emp.last_name%TYPE;
BEGIN
    SELECT last_name, salary
    INTO   v_last_name, v_sal
    FROM   s_emp
    WHERE  dept_id = &p_dept_id;

    DBMS_OUTPUT.PUT_LINE(v_last_name || ' ' || v_sal);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE(' No data for the ' || &p_dept_id);
    WHEN TOO_MANY_ROWS THEN
        DBMS_OUTPUT.PUT_LINE(' Too many data for the ' || &p_dept_id);
END;
/
```

```
SQL> @a5
```

```
부서번호 : 31
```

```
SQL> @a5
```

```
부서번호 : 100
```

## DML in PL/SQL

- Example

```
CREATE OR REPLACE FUNCTION dml_emp
  (p_id s_emp.id%TYPE) RETURN NUMBER
IS
  v_emp s_emp%ROWTYPE ;
BEGIN
  INSERT INTO s_emp(id, last_name)
  VALUES (p_id, USER);
  UPDATE s_emp SET salary = 2000 WHERE id = p_id;
  SELECT * INTO v_emp
  FROM s_emp WHERE id = p_id;
  RETURN (v_emp.salary);
END dml_emp;
```

 Academy Course

dml\_emp function을 만들어 실행시켜 보시오.

```
SQL> ed f2
CREATE OR REPLACE FUNCTION dml_emp
  (p_id s_emp.id%TYPE)
  RETURN NUMBER
IS
  v_emp s_emp%ROWTYPE ;
BEGIN
  INSERT INTO s_emp(id, last_name) VALUES (p_id, USER);
  UPDATE s_emp SET salary = 2000 WHERE id = p_id;

  SELECT *
  INTO v_emp
  FROM s_emp
  WHERE id = p_id;

  RETURN (v_emp.salary);
END dml_emp;
/
SQL> @f2
SQL> VARIABLE g_sal NUMBER
SQL> EXEC :g_sal := dml_emp(77);
```



```
SQL> PRINT g_sal
```

```
SQL> SELECT * FROM s_emp WHERE id = 77;
```

```
SQL> ROLLBACK;
```

```
SQL> SELECT * FROM s_emp WHERE id = 77;
```

## Transaction Control in PL/SQL

- Example

```
BEGIN
  INSERT INTO temp(num_col1, num_col2, char_col)
    VALUES (1, 1, 'ROW1');
  SAVEPOINT a;
  INSERT INTO temp(num_col1, num_col2, char_col)
    VALUES (2, 2, 'ROW2');
  SAVEPOINT b;
  INSERT INTO temp(num_col1, num_col2, char_col)
    VALUES (3, 3, 'ROW3');
  SAVEPOINT c;
  ROLLBACK TO SAVEPOINT b;
  COMMIT;
END;
/
```

 Academy Course

Transaction control 문장을 실습해 보시오.

```
SQL> CREATE TABLE temp
      (num_col1 NUMBER,
       num_col2 NUMBER,
       char_col char(4));

SQL> ed a6
BEGIN
      INSERT INTO temp(num_col1, num_col2, char_col)
        VALUES (1, 1, 'ROW1');
      SAVEPOINT a;
      INSERT INTO temp(num_col1, num_col2, char_col)
        VALUES (2, 2, 'ROW2');
      SAVEPOINT b;
      INSERT INTO temp(num_col1, num_col2, char_col)
        VALUES (3, 3, 'ROW3');
      SAVEPOINT c;

      ROLLBACK TO SAVEPOINT b;
      COMMIT;

END;
/
SQL> @a6
SQL> SELECT * FROM temp;
```

## IF Statement

- Syntax

```
IF condition THEN
    statements
[ELSIF condition THEN
    statements]
[ELSE
    statements]
END IF;
```

## IF Statement(cont.)

- Example

```
CREATE OR REPLACE FUNCTION calc_num
(p_num IN NUMBER) RETURN NUMBER
IS
BEGIN
    IF p_num > 10 THEN
        RETURN (2 * p_num);
    ELSIF p_num > 5 THEN
        RETURN (1.5 * p_num);
    ELSE RETURN (p_num);
    END IF;
END calc_num;
/
```

 Academy Course

calc\_num function을 생성하여 IF 문장을 실습하시오.

```
SQL> ed f3
CREATE OR REPLACE FUNCTION calc_num
    (p_num IN NUMBER)
    RETURN NUMBER
IS
BEGIN
    IF p_num > 10 THEN
        RETURN (3 * p_num);
    ELSIF p_num > 5 THEN
        RETURN (2 * p_num);
    ELSE
        RETURN (p_num);
    END IF;
END calc_num;
/
SQL> @f3
SQL> VARIABLE num NUMBER
SQL> EXEC :num := calc_num(16)
SQL> PRINT num
SQL> EXEC :num := calc_num(6)
SQL> PRINT num
```

```
SQL> EXEC :num := calc_num(0)
SQL> PRINT num
```

## Basic Loop

- Syntax

```
LOOP
  statement1;
  statement2;
  ...
  EXIT [WHEN condition];
END LOOP;
```

- To quit the loop, use EXIT statement.

## Basic Loop(cont.)

- Example

```

DECLARE
    v_counter NUMBER := 1;
BEGIN
    LOOP
        DBMS_OUTPUT.PUT_LINE(v_counter);
        EXIT WHEN (v_counter = 10);
        v_counter := v_counter + 1;
    END LOOP;
END;
/

```

BIT Academy Course

Basic loop를 실습해 보시오.

```

SQL> ed a7
DECLARE
    v_counter NUMBER := 1;
BEGIN
    LOOP
        DBMS_OUTPUT.PUT_LINE(v_counter);
        EXIT WHEN (v_counter = 10);
        v_counter := v_counter + 1;
    END LOOP;
END;
/
SQL> @a7

```

## WHILE Loop

- Iterates loop during the condition is TRUE.
- Syntax

```
WHILE condition LOOP  
    statement1;  
    statement2;  
    ...  
END LOOP;
```



## WHILE Loop(cont.)

- Example

```

DECLARE
    v_counter NUMBER := 1;
BEGIN
    WHILE (v_counter <= 10) LOOP
        DBMS_OUTPUT.PUT_LINE(v_counter);
        v_counter := v_counter + 1;
    END LOOP;
END;
/

```

BIT Academy Course

While loop를 실습해 보시오.

```

SQL> ed a8
DECLARE
    v_counter NUMBER := 1;
BEGIN
    WHILE (v_counter <= 10) LOOP
        DBMS_OUTPUT.PUT_LINE(v_counter);
        v_counter := v_counter + 1;
    END LOOP;
END;
/
SQL> @a8

```

## FOR Loop

- Iterates over a specified range of integers.
- Syntax

```
FOR index IN [REVERSE]
  lower_bound .. upper_bound LOOP
  statement1;
  statement2;
  ...
END LOOP;
```

- You need not explicitly declare the loop counter.
- The loop counter is defined only within the loop.
- You cannot change the index value in the loop.

## FOR Loop(cont.)

- Example

```
BEGIN
  FOR i IN 1..10 LOOP
    DBMS_OUTPUT.PUT_LINE(i);
  END LOOP;
END;
/
```

BIT Academy Course

For loop를 실습해 보시오.

```
SQL> ed a9
BEGIN
    FOR i IN 1..10 LOOP
        DBMS_OUTPUT.PUT_LINE(i);
    END LOOP;
END;
/
SQL> @a9
```

index 변수는 수정이 불가능하다.

```
SQL> ed a9
BEGIN
    FOR i IN 1..10 LOOP
        DBMS_OUTPUT.PUT_LINE(i);
        i := 5;
    END LOOP;
END;
/
SQL> @a9
```

## Nested Loop

- Example

```
.....
<<L1>>LOOP
  DBMS_OUTPUT.PUT_LINE('BEGIN LOOP1');
  <<L2>>LOOP
    DBMS_OUTPUT.PUT_LINE('BEGIN LOOP2');
    IF v_num > 10 THEN EXIT "L1";
    ELSE EXIT "L2"; END IF;
    DBMS_OUTPUT.PUT_LINE('END LOOP2');
  END LOOP;
  DBMS_OUTPUT.PUT_LINE('END LOOP1');
  EXIT;
END LOOP;
```

BIT Academy Course

중첩된 loop에 label을 사용하는 다음의 실습을 해보시오.

```
SQL> ed a10
DECLARE
    v_num NUMBER := &p_num;
BEGIN
<<L1>>LOOP
    DBMS_OUTPUT.PUT_LINE('BEGIN LOOP1');
    <<L2>>LOOP
        DBMS_OUTPUT.PUT_LINE('BEGIN LOOP2');
        IF v_num > 10 THEN EXIT "L1";
        ELSE EXIT "L2";
        END IF;
        DBMS_OUTPUT.PUT_LINE('END LOOP2');
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('END LOOP1');
    EXIT;
END LOOP;
END;
/
SQL> @a10
Enter value for p_num: 1
SQL> @a10
Enter value for p_num: 20
```

## Nested Block

- Example

```
BEGIN
  DBMS_OUTPUT.PUT_LINE('BEGIN 1');
  BEGIN
    DBMS_OUTPUT.PUT_LINE('BEGIN 2');
    BEGIN
      DBMS_OUTPUT.PUT_LINE('BEGIN 3');
      GOTO B3;
    END;
    DBMS_OUTPUT.PUT_LINE('END 2');
  END;
  <<B3>> DBMS_OUTPUT.PUT_LINE('END 1');
END;
/
```

BIT Academy Course

중첩된 loop에 label을 사용하는 다음의 실습을 해보시오.

```
SQL> ed a11
DECLARE
    v_num NUMBER := &p_num;
BEGIN
    DBMS_OUTPUT.PUT_LINE('BEGIN 1');
    BEGIN
        DBMS_OUTPUT.PUT_LINE('BEGIN 2');
        BEGIN
            DBMS_OUTPUT.PUT_LINE('BEGIN 3');
            IF v_num > 10 THEN GOTO B3;
            ELSE GOTO B2;
            END IF;
            DBMS_OUTPUT.PUT_LINE('END 3');
        END;
        <<B2>>DBMS_OUTPUT.PUT_LINE('END 2');
    END;
    <<B3>>DBMS_OUTPUT.PUT_LINE('END 1');
END;
/
SQL> @a11
Enter value for p_num: 1 (또는 20)
```

## Comments

- The PL/SQL compiler ignores comments.
- Add comments to your program promotes readability and aids understanding.
- PL/SQL supports two comment styles
  - single-line : --
  - multi-line : /\* ... \*/

# PL/SQL Program

(Cursor, Exception)

- **CURSOR**

- Declare, Open, Fetch, Close
- Cursor For Loop
- Where Current Of
- Parameterized Cursor

- **Exception**

- Predefined
- Non-Predefined
- User Defined
- SQLCODE, SQLERRM

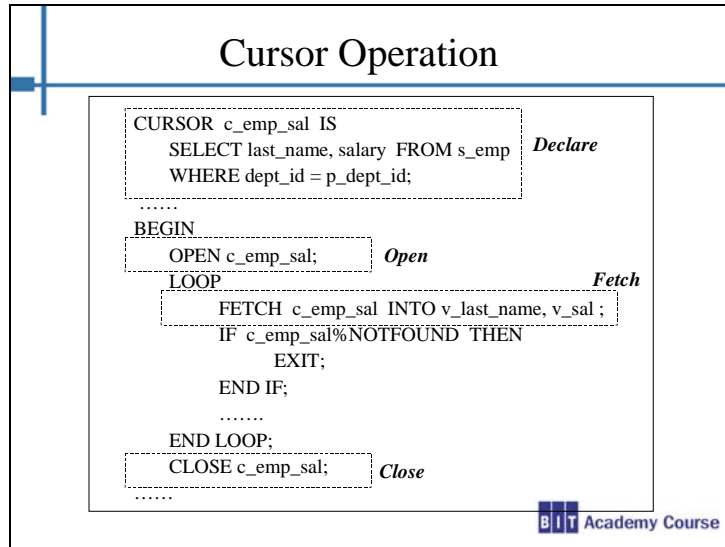
## Cursor

- A cursor is a handle or name for a private SQL area.
- Implicit Cursor
  - Automatically created and used by Oracle for all SQL statements.
- Explicit Cursor
  - Declared by programmers.
  - Used to query multi-rows.



## Explicit Cursor

- Cursor Operation Step
  - 1. DECLARE
  - 2. OPEN
  - 3. FETCH
  - 4. CLOSE
- Cursor Attributes
  - %ISOPEN
  - %NOTFOUND
  - %FOUND
  - %ROWCOUNT



Simple Loop를 사용하여 Cursor를 사용하는 실습을 해보시오.

```

SQL> ed p3
CREATE OR REPLACE PROCEDURE p_emp_info
  (p_dept_id   s_emp.dept_id%TYPE)
IS
  CURSOR c_emp_sal IS
    SELECT last_name, salary
    FROM s_emp
    WHERE dept_id = p_dept_id;
  v_sal          s_emp.salary%TYPE;
  v_last_name    s_emp.last_name%TYPE;
BEGIN
  OPEN c_emp_sal;
  LOOP
    FETCH c_emp_sal INTO v_last_name, v_sal ;
    IF c_emp_sal%NOTFOUND THEN
      EXIT;
    END IF;
    DBMS_OUTPUT.PUT_LINE(v_last_name || ' ' || v_sal);
  END LOOP;
  CLOSE c_emp_sal;
END;

```

*Oracle 9i 실행(SQL, PL/SQL) >>>*

```
/
SQL> @p3
SQL> EXEC p_emp_info(31);
```

## Cursor For Loop

```

CURSOR c_emp_sal IS                                Declare
  SELECT last_name, salary FROM s_emp
  WHERE dept_id = p_dept_id;
BEGIN
  FOR emp_rec IN c_emp_sal LOOP                    Open
    DBMS_OUTPUT.PUT_LINE                          Fetch
      (emp_rec.last_name);
  END LOOP;                                         Close
  .....

```

BIT Academy Course

Cursor For Loop를 사용하여 Cursor를 사용하는 실습을 해보시오.

```

SQL> ed p4
CREATE OR REPLACE PROCEDURE p_emp_info
  (p_dept_id s_emp.dept_id%TYPE)
IS
  CURSOR c_emp_sal IS
    SELECT last_name, salary
    FROM s_emp
    WHERE dept_id = p_dept_id;
BEGIN
  FOR emp_rec IN c_emp_sal LOOP
    DBMS_OUTPUT.PUT_LINE(emp_rec.last_name || ' ' || emp_rec.salary);
  END LOOP;
END;
/
SQL> @p4
SQL> EXEC p_emp_info(31);

```

## Where Current Of

```
CURSOR c_emp_sal IS
  SELECT last_name, salary FROM s_emp
         FOR UPDATE;

BEGIN
  FOR emp_rec IN c_emp_sal LOOP
    UPDATE s_emp
      SET salary = salary * 1.2
      WHERE CURRENT OF c_emp_sal;
    .....
  
```

BIT Academy Course

Where Current Of를 사용하여 Cursor를 사용하는 실습을 해보시오.

```
SQL> ed a12
DECLARE
  CURSOR c_emp_sal IS
    SELECT last_name, salary
    FROM s_emp
    FOR UPDATE;

BEGIN
  FOR emp_rec IN c_emp_sal LOOP
    DBMS_OUTPUT.PUT_LINE(emp_rec.last_name||' '|| emp_rec.salary);
    IF emp_rec.salary < 2000 THEN
      UPDATE s_emp SET salary = salary * 1.2
        WHERE CURRENT OF c_emp_sal;
    ELSE
      UPDATE s_emp SET salary = salary * 1.1
        WHERE CURRENT OF c_emp_sal;
    END IF;
  END LOOP;

END;
/
SQL> @a12
SQL> SELECT last_name, salary FROM s_emp;
```

## Parameterized Cursor

- Syntax

```
CURSOR cursor_name [(Parameter_name data type, ...)]
IS subquery;
```

- Example

```
CURSOR c_dept IS
    SELECT * FROM s_dept;
CURSOR c_emp_sal (p_dept_id s_emp.dept_id%TYPE) IS
    SELECT last_name, salary FROM s_emp
    WHERE dept_id = p_dept_id;
BEGIN
    FOR dept IN c_dept LOOP
        FOR emp_record IN c_emp_sal (dept.id) LOOP
```

BIT Academy Course

Parameter를 이용한 Cursor를 사용하는 실습을 해보시오.

```
SQL> ed a13
DECLARE
    CURSOR c_dept IS
        SELECT *
        FROM s_dept;
    CURSOR c_emp_sal (p_dept_id s_emp.dept_id%TYPE) IS
        SELECT last_name, salary
        FROM s_emp
        WHERE dept_id = p_dept_id;
BEGIN
    FOR dept IN c_dept LOOP
        DBMS_OUTPUT.PUT_LINE('=====');
        DBMS_OUTPUT.PUT_LINE(dept.id || ' :: ' || dept.name);
        FOR emp_record IN c_emp_sal (dept.id) LOOP
            DBMS_OUTPUT.PUT_LINE(emp_record.last_name ||
                                ' ' || emp_record.salary);
        END LOOP;
    END LOOP;
END;
/
SQL> @a13
```

## Exception

- A warning or error condition in PL/SQL.
- Exceptions can be internally defined or user defined.
- When an exception is raised normal execution stops and control transfers to the exception-handling part.
- Exception handlers can be written to handle any internal or user-defined exception.
- Every Oracle error has a number, but exceptions must be handled by name.

## Exception Types


- **Predefined**
  - Common Oracle errors defined as exceptions already.
- **Non-Predefined**
  - Oracle errors defined as exceptions by the pragma `EXCEPTION_INIT` to associate exception names with Oracle error codes.
- **User-Defined**
  - Logical errors declared as exceptions.
  - Must be raised explicitly by `RAISE` statements.



## Exception Handlers

- Syntax

```
EXCEPTION
  WHEN expr1 [OR expr2 . . .] THEN
    statement1;
    statement2;
    ...
  [WHEN expr3 [OR expr4 . . .] THEN
    statement1;
    statement2;
    ...]
  [WHEN OTHERS THEN
    statement1;
    statement2;
    ...]
```

 BIT Academy Course

Error 발생하면 프로그램의 control은 더 이상 block을 수행하지 않고 EXCEPTION 처리부로 넘어간다.

WHEN clause를 차례로 비교하다가 처리되지 않은 Error는 마지막 WHEN OTHERS clause에서 처리된다.

해당 Error에 대한 exception handler가 실행되고 나면 해당 block이 종료된다.

Error가 발생했더라도 exception 처리가 되었다면 PL/SQLblock 전체는 정상적으로 완료된다.

## Predefined Exceptions

- Example
  - NO\_DATA\_FOUND (ORA-01403)
  - TOO\_MANY\_ROWS (ORA-01422)
  - INVALID\_CURSOR (ORA-01001)
  - ZERO\_DIVIDE (ORA-01476)
  - DUP\_VAL\_ON\_INDEX (ORA-00001)

## Predefined Exceptions(cont.)

- Example

```
.....  
SELECT last_name, salary  
INTO v_last_name, v_salary  
FROM s_emp;  
EXCEPTION  
  WHEN NO_DATA_FOUND THEN  
    ROLLBACK;  
  .....  
  WHEN TOO_MANY_ROWS THEN  
    .....  
END;
```

## Non-Predefined Exceptions

1. Declare exception names in the declare part.

```
exception_name EXCEPTION;
```

2. Associate exception names with Oracle error codes by using PRAGMA EXCEPTION\_INIT.

```
PRAGMA EXCEPTION_INIT
(exception_name, error_number);
```

3. Reference exception names in the exception handling part.

BIT Academy Course

Predefined exception도 아래와 같이 다른 이름으로 재정의하여 사용할 수 있다.

```
SQL> ed ex1
DECLARE
    v_emp_record          s_emp%ROWTYPE;
    e_too_many_rows      EXCEPTION;
    PRAGMA EXCEPTION_INIT(e_too_many_rows, -1422);
BEGIN
    SELECT *
    INTO   v_emp_record
    FROM   s_emp;
EXCEPTION
    WHEN e_too_many_rows THEN
        DBMS_OUTPUT.PUT_LINE('There are too many rows');
END;
/
SQL> @ex1
```

## Non-Predefined Exceptions(cont.)

- Example

```
DECLARE
    e_child_exists EXCEPTION;
    PRAGMA EXCEPTION_INIT
        (e_child_exists, -2292);
BEGIN
    . . .
    EXCEPTION
        WHEN e_child_exists THEN
            DBMS_OUTPUT.PUT_LINE(...);
        . . .
END;
```

 Academy Course

DELETE, INSERT를 각각 실행하여 다른 종류의 Exception을 발생시켜 보시오.

```
SQL> CREATE TABLE ex1 (a NUMBER PRIMARY KEY);
SQL> INSERT INTO ex1 VALUES(10);
SQL> CREATE TABLE ex2 (b NUMBER REFERENCES ex1(a));
SQL> INSERT INTO ex2 VALUES(10);
SQL> COMMIT;
SQL> ed ex2
DECLARE
    e_child_exists EXCEPTION;
    e_no_parent      EXCEPTION;
    PRAGMA EXCEPTION_INIT (e_child_exists, -2292);
    PRAGMA EXCEPTION_INIT (e_no_parent, -2291);

BEGIN
    DELETE FROM ex1;
    -- INSERT INTO ex2 VALUES(30);

EXCEPTION
    WHEN e_child_exists THEN
        DBMS_OUTPUT.PUT_LINE('Referential integrity constraint violated.');

WHEN e_no_parent THEN



DBMS_OUTPUT.PUT_LINE('Foreign key constraint violated.');



END;



/



SQL> @ex2


```

## User-Defined Exceptions

1. Declare exception names in the declare part.

```
exception_name EXCEPTION;
```

2. Raise an exception explicitly by using RAISE statement.

```
RAISE exception_name;
```

3. Reference exception names in the exception handling part.

## User-Defined Exceptions(cont.)

- Example

```
DECLARE
    e_too_less_sal EXCEPTION;
    . . .
BEGIN
    . . .
    RAISE e_too_less_sal;
    . . .
EXCEPTION
    WHEN e_too_less_sal THEN
        . . .
END;
```

BIT Academy Course

아래 내용대로 User defined Exception을 발생시켜 보시오.

```
SQL> ed ex3
DECLARE
    e_too_less_sal    EXCEPTION;
    v_sal             s_emp.salary%TYPE;
BEGIN
    SELECT salary
    INTO v_sal
    FROM s_emp
    WHERE id = 23;

    IF v_sal < 2000 THEN
        RAISE e_too_less_sal;
    END IF;

    DBMS_OUTPUT.PUT_LINE('LAST STATEMENT');
EXCEPTION
    WHEN e_too_less_sal THEN
        DBMS_OUTPUT.PUT_LINE('Salary Increase is needed');
END;
/
SQL> @ex3
```

## Error Functions

- **SQLCODE**
  - Returns error number.
- **SQLERRM**
  - Returns error messages.



## Error Functions(cont.)

- Example

```
EXCEPTION
  WHEN e_child_exists THEN
    DBMS_OUTPUT.PUT_LINE(
      SQLCODE || ' : ' || SQLERRM);
  WHEN e_no_parent THEN
    DBMS_OUTPUT.PUT_LINE(
      SQLCODE || ' : ' || SQLERRM);
END;
```

 Academy Course

SQLCODE, SQLERRM을 사용하여Exception을 처리해 보시오.

```
SQL> ed ex2
DECLARE
    e_child_exists    EXCEPTION;
    e_no_parent       EXCEPTION;
    PRAGMA EXCEPTION_INIT (e_child_exists, -2292);
    PRAGMA EXCEPTION_INIT (e_no_parent, -2291);

BEGIN
    DELETE FROM ex1;
    -- INSERT INTO ex2 VALUES(30);

EXCEPTION
    WHEN e_child_exists THEN
        DBMS_OUTPUT.PUT_LINE(SQLCODE || ' : ' || SQLERRM);
    WHEN e_no_parent THEN
        DBMS_OUTPUT.PUT_LINE(SQLCODE || ' : ' || SQLERRM);

END;
/
SQL> @ex2
```

# 부 록

- 
1. Oracle Server 설치
  2. 분산 DB 실습

## 1. Oracle Server 설치

### 1.1 설치 과정에서의 유의사항

CD를 넣으면 installer가 자동으로 구동되며(또는 setup.exe를 실행) Default 설정대로 Next 버튼을 눌러 각자의 PC에 DBMS를 설치한다. 이 때, 오라클 제품이 설치될 위치를 확인하여 둔다.

예) C: \WOracle\Wora90 또는 D: \WOracle\Wora90

또한, 아래와 같이 입력을 받는 단계에서는 각자의 PC에 만들어질 DB와 Oracle Service ID 이름을 정해서 넣어준다. 이 때, xx는 각자의 고유 번호로 한다.

예) Global Database Name : BITxx  
Service Identifier : BITxx

DBMS 설치와 함께 테스트용 DB가 설치된다. 이 때, SYS,SYSTEM 사용자가 생성되는데 password를 default로 설정하여 만든다.

예) SYS : change\_on\_install  
SYSTEM : manager

### 1.2 설치 후 Registry 확인

시작 -> 실행 -> regedit

내컴퓨터 -> HKEY\_LOCAL\_MACHINE -> SOFTWARE -> Oracle -> HOME0

아래 문자열의 값을 기록하시오.

NLS\_LANG :  
ORACLE\_HOME :  
ORACLE\_SID :

### 1.3 설치 후 Service 확인

Oracle DBMS가 설치되고 나면 관련 Service들이 생성된다.

시작 -> 설정 -> 제어판 -> 관리도구 -> 서비스 -> OracleOraHome90TNSListener

시작 -> 설정 -> 제어판 -> 관리도구 -> 서비스 -> OracleService(BITxx)

## 2. 분산 DB 실습

### 2.1 TNS alias 등록

#### 2.1.1 상대 DB 서버의 TNS 정보 확인

연결하고 싶은 DB 서버 ORACLE\_HOMEnetwork\admin\tnsnames.ora를 열어  
(notepad로 연결) HOST,PORT,SERVICE\_NAME을 확인한다.

```
예) (DESCRIPTION =  
      (ADDRESS_LIST =  
        (ADDRESS = (PROTOCOL = TCP)(HOST = teacher)(PORT = 1521))  
      )  
      (CONNECT_DATA =  
        (SERVICE_NAME = BIT01)  
      )  
    )
```

#### 2.1.2 상대 DB 서버의 정보를 내 PC에 등록

내 PC의 ORACLE\_HOMEnetwork\admin\tnsnames.ora에 상대 DB를 지칭하는 TNS  
alias를 추가한다

```
예) TNS01 = (DESCRIPTION =  
      (ADDRESS_LIST =  
        (ADDRESS = (PROTOCOL = TCP)(HOST = teacher)(PORT = 1521))  
      )  
      (CONNECT_DATA =  
        (SERVICE_NAME = BIT01)  
      )  
    )
```

#### 2.1.3 상대 DB 서버에 접속

– 연결할 서버의 사용자/암호 확인

예) test01/test01

– SQL\*Plus를 띄워 TNS alias를 사용하여 상대 DB 서버에 접속

예) 사용자 : test01

암호 : test01

호스트 : TNS01

### 2.2 데이터베이스 링크 생성 및 분산 트랜잭션 테스트

```
SQL> conn test01/test01@tns01
```

```
SQL> CREATE TABLE table01 ( a number);
SQL> INSERT INTO table01 VALUES(10);
SQL> COMMIT;

SQL> conn testxx/testxx
SQL> CREATE DATABASE LINK link01
CONNECT TO test01 IDENTIFIED BY test01
USING 'TNS01';
SQL> SELECT * FROM user_db_links;

SQL> SELECT * FROM s_emp;
SQL> SELECT * FROM s_emp@link01;

SQL> SELECT * FROM table01@link01;
SQL> CREATE SYNONYM table01 FOR table01@link01;
SQL> SELECT * FROM table01;

SQL> INSERT INTO table01 VALUES(20);
SQL> UPDATE s_emp SET salary = 0;
SQL> COMMIT;

SQL> DROP DATABASE LINK link01;
SQL> SELECT * FROM table01;
```