

---

# Oracle SQL & JDBC 프로그래밍

# - Introduction to Database

## 목 차

---

1. 데이터베이스 소개
2. 관계형 모델
3. 오라클 소개 및 SQL\*PLUS 사용법

## Reference

---

### □ 참고 URL

- 오라클 공식 문서: <http://www.oracle.com/pls/db102/homepage>
- 오라클 10g 튜토리얼:  
<http://st-curriculum.oracle.com/tutorial/DBXETutorial/index.htm>오라클 강좌:  
<http://www.oracleclub.com/oracle/lecture/LectureHTML.jsp?lectureType=INSTALL>

### □ 참고서적

- (열혈강의) 10g로 시작하는 오라클 SQL & PL/SQL, 김태근저, FREELEC

---

# 데이터베이스 개요

# 데이터베이스

---

## ❑ Database (DB)

- 한 조직의 여러 응용 시스템들이 **공용(Shared)**하기 위해  
**통합(Integrated)**, **저장(Stored)**한 **운영 데이터(Operational data)**의 집합

## ❑ 특징

- 컴퓨터 시스템과 무관
- 데이터의 구조적 집합 (종이, 장부 등도 DB라고 할 수 있음)
- 일반적으로 컴퓨터 시스템을 이용하여 구축하여둔 데이터의 집합을 의미
- 데이터 모델에 따라 데이터베이스의 구조는 달라질 수 있음



# 데이터베이스 관리 시스템

---

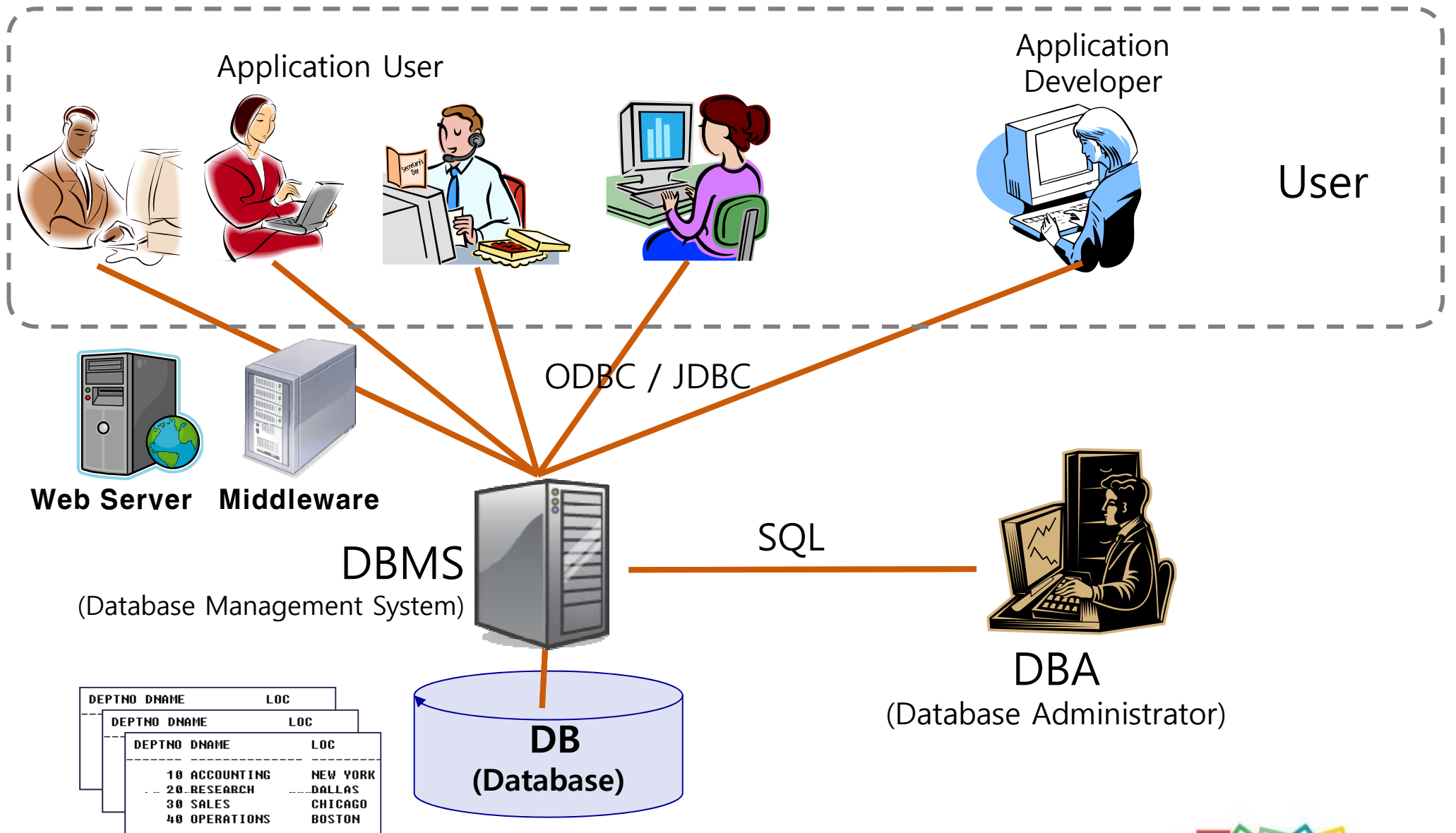
## ❑ Database Management System (DBMS) : DB관리를 위한 컴퓨터 시스템

- 전사적인 정보 관리
- 관련된 데이터의 집합
- 데이터에 접근하는 프로그램 집합
- 효율적이고 편한 사용을 위한 환경

## ❑ 데이터베이스 응용의 예:

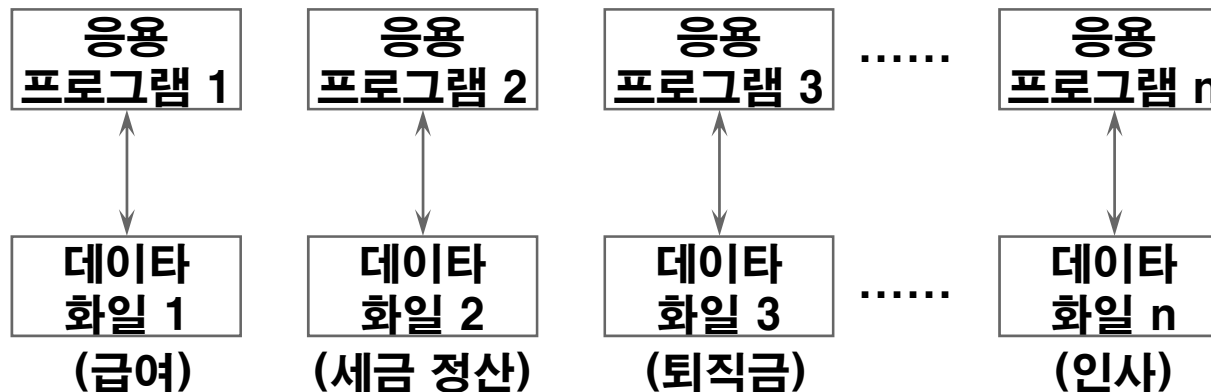
- Banking: all transactions
- Airlines: reservations, schedules
- Universities: registration, grades
- Sales: customers, products, purchases
- Online retailers: order tracking, customized recommendations
- Manufacturing: production, inventory, orders, supply chain
- Human resources: employee records, salaries, tax deductions

# Database System



## DBMS의 목적

- ❑ 왜 **Databae**관리를 위한 별도의 시스템이 필요한가?
- ❑ 파일 시스템 등의 저장소를 이용하여 직접 **Database** 관리 프로그램을 짜면 더 효율적이고, 응용 프로그램에 적합하게 제작할 수 있지 않을까?
- ❑ 초기 데이터 응용에서는 프로그래머가 직접 모든 프로그램을 작성
  - OS의 파일 시스템 등을 이용





## 파일 시스템의 문제점 (1/2)

---

### ❑ 데이터의 **중복(Redundancy)** 와 일관성(**Consistency**) 문제

- Multiple file formats, duplication of information in different files

### ❑ 데이터 접근의 어려움

- 각 작업마다 별도의 프로그램 작성
- 각각 별도의 방법이 필요할 수 있음

### ❑ 데이터 종속성 (**Dependency**)

- 데이터의 포맷이나 접근 방법 등이 프로그램 코드에 종속됨.
- 프로그램의 변경이나 데이터 형태, 종류 등의 변경이 불가능

### ❑ 데이터 독립성 (**Isolation**)

- 여러 프로그램에서 동시에 데이터를 수정하면?
- 하나의 수정 작업이 다른 작업에 영향을 줄 수 있음

## 파일 시스템의 문제점 (2/2)

---

### ❑ 변경의 원자성(Atomicity) 문제

- 일련의 작업 중 시스템의 failure가 발생하면??
- 예) 계좌이체 중 내 계좌에서 돈이 나갔는데, 다른 계좌에 가기 전에 정전이 일어난다면?

### ❑ 동시 사용성(Concurrency) 제어 문제

- 동시에 일련의 작업들이 이루어질 경우 올바른 수행을 보장할 수 있는가? (일관성에 문제)
- 예) 두 명이 동시에 한계좌에서 돈을 인출하려고 하면?

### ❑ 데이터 무결성 (Integrity) 문제

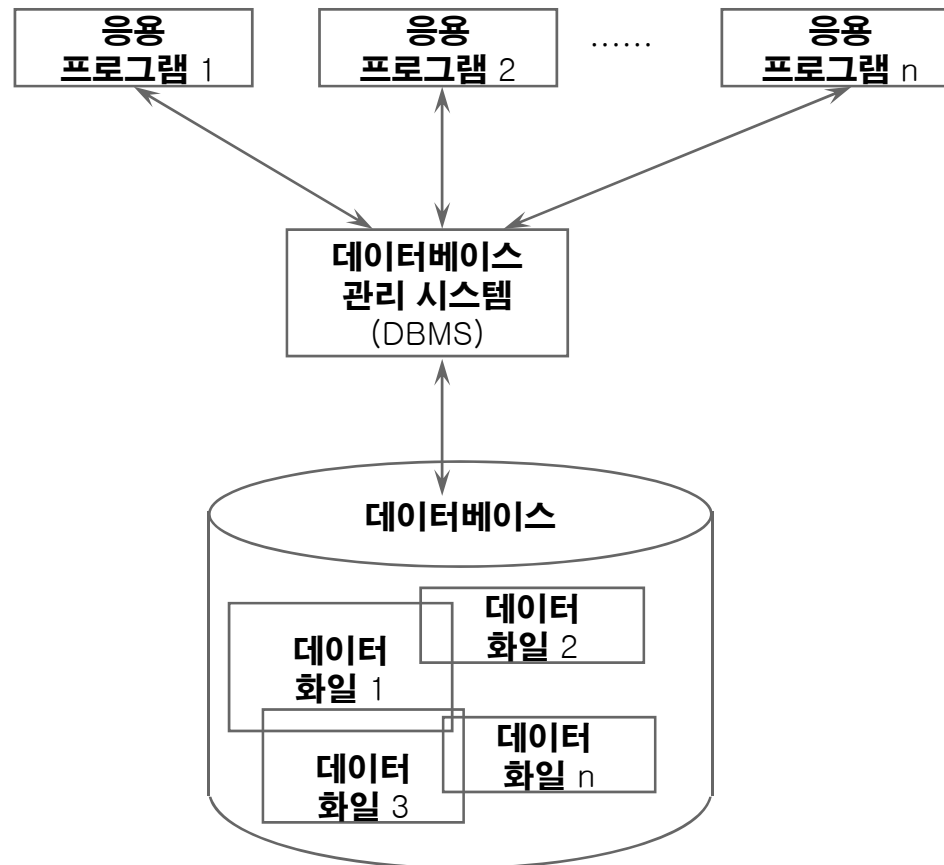
- Integrity constraints (예.  $\text{account balance} > 0$ ) 가 프로그램 코드 속에 기술
  - 프로그램 코드를 복잡하게 만들고 유지 보수를 어렵게 함
- 제약조건 변경이나 추가 등이 힘들다.

### ❑ 보안

- 보안을 보장하기 힘들: 다양한 파일, 다양한 접근 경로, 다양한 프로그램의 이용

# DBMS

- ❑ 데이터의 종속성과 중복성의 문제 해결
- ❑ 데이터베이스를 공유할 수 있도록 관리하는 시스템



# DBMS의 장단점

---

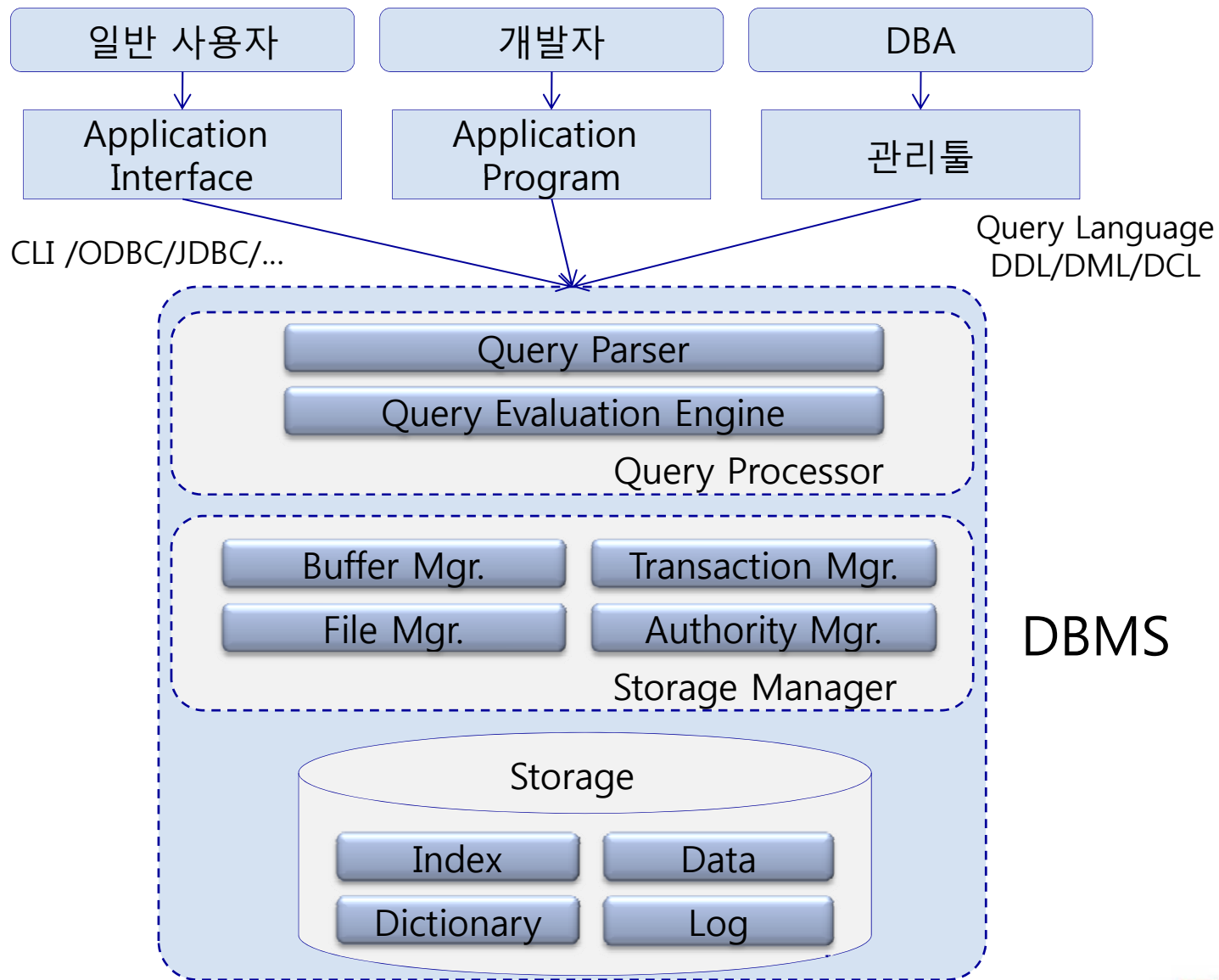
## □ 장점

- 데이터 중복(redundancy)의 최소화
- 데이터의 공유(sharing)
- 일관성(consistency) 유지
- 무결성(integrity) 유지
- 보안(security) 보장
- 표준화(standardization) 용이
- 전체 데이터 요구의 조정

## □ 단점

- 비용: H/W, DBMS, 운영비, 교육비, 개발비
- 프로그램의 복잡화
- 성능상의 오버헤드

# DBMS 구조



DBMS

# Data Model

---

## ❑ Relational(관계형) model

- 가장 널리 이용됨. 대표적 Data Model
- Relation (Table)에 기반한 모델, 사용 편리, 성능 우수
- Oracle, IBM DB2, MS-SQL Server 등 대부분 DBMS가 RDBMS

## ❑ Entity-Relationship(E-R) data model

- 데이터베이스 설계에 주로 이용됨

## ❑ Object-based data models (Object-oriented and Object-relational)

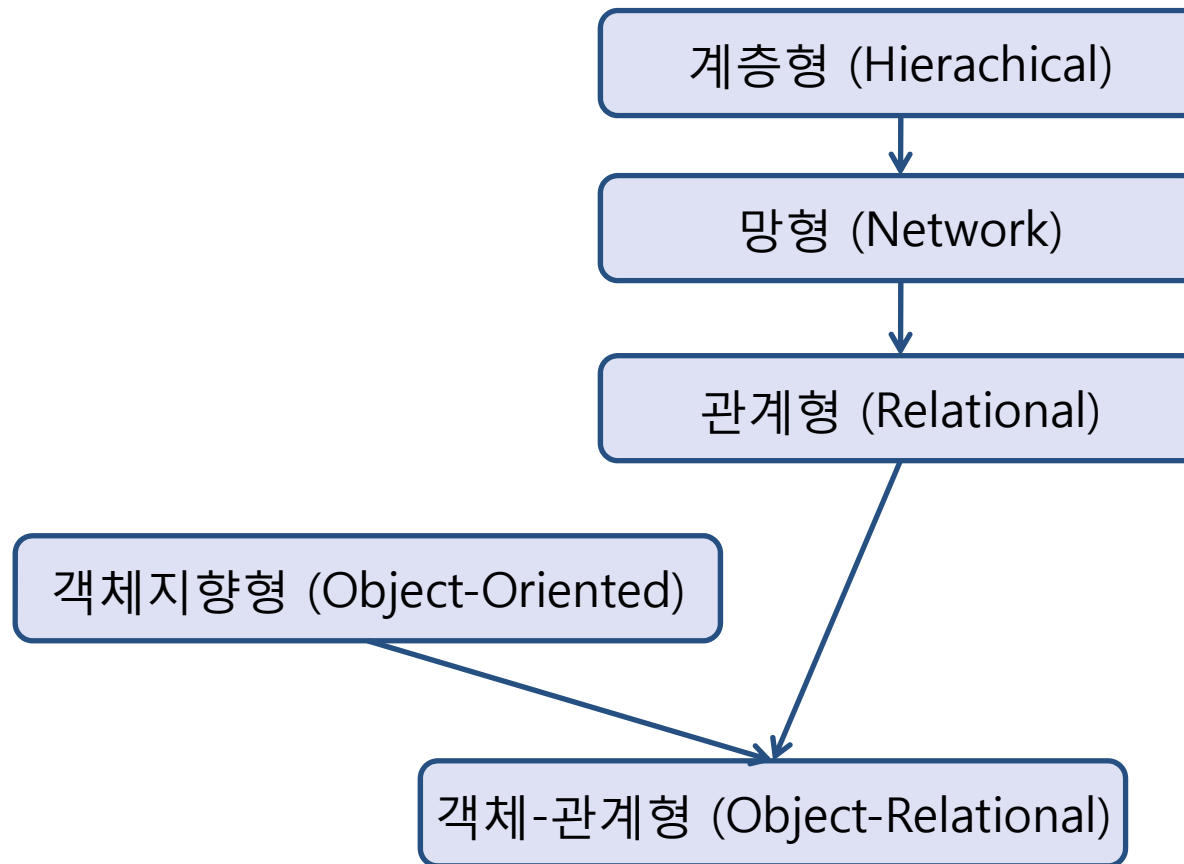
- RDBMS의 한계점 극복 위해 제안
- OO-DBMS의 장점이 부각되자, 대형 RDBMS Vendor들이 OR-DBMS 발표
- 현재 대부분의 RDBMS는 실질적으로 ORDBMS

## ❑ 기타

- Network(네트워크) model , Hierarchical(계층) model: RDBMS이전에 주로 사용되었던 모델.

## Data Model 역사

---



# 데이터베이스 역사

---

## ❑ 1950s and early 1960s:

- 펀치카드, 테이프(순차적 접근만 가능)등을 이용한 데이터 처리
- File System

## ❑ Late 1960s and 1970s:

- HDD 사용
- Network & hierarchical data models
- Ted Codd가 relational data model 정의
  - ACM Turing Award 수상
  - IBM Research: System R prototype
  - UC Berkeley: Ingres prototype
- 고성능 트랜잭션 처리



## 데이터베이스 역사 (계속)

---

### ❑ 1980s:

- RDBMS가 상용화 됨 (Oracle)
  - SQL이 표준으로 제정됨
- 병렬, 분산 Database
- OO-DBMS 소개됨

### ❑ 1990s:

- OR-DBMS
- 대규모 의사결정 시스템 및 Data Mining 응용
- 수 TB 규모의 데이터웨어하우스
- 웹 시스템과 결합

### ❑ 2000s:

- XML & XQuery, 스트림 처리, GIS, Grid, 내장형, Real-Time DB, 자동화된 DB 관리

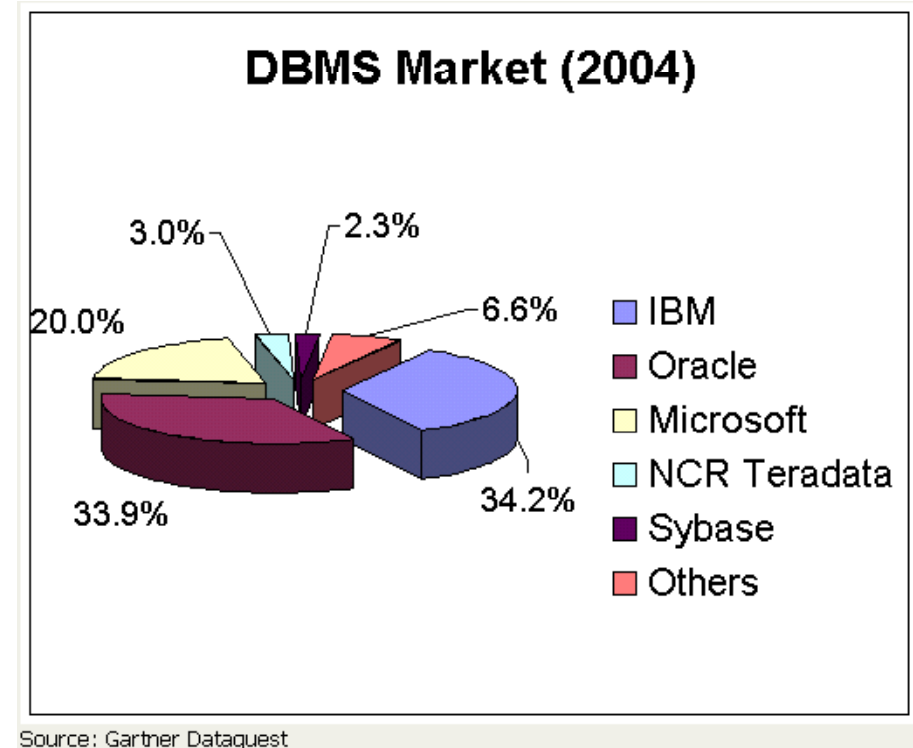
# DBMS 제품

## □ BIG 3

- Oracle: RDBMS 최초 상용화, RDBMS 시장 점유율 가장 높음 (국내 점유율 특히 높음)
- IBM DB2: RDBMS 최초개발, 메인프레임 등에서 점유율 높음
- MS-SQL Server: Sybase 코드에 기반

## □ 기타

- Teradata, Informix, Sybase
- MySQL, PostgreSQL, Firebird, Cubrid
- Main-Memory(Real-time) DB : Altibase, TimesTen
- Embedded DB: SQLite, BerkeleyDB



# DBMS 선정시 고려사항

---

## □ 기술적 요인

- DBMS에 사용되고 있는 데이터 모델
- DBMS가 지원하는 사용자 인터페이스
- 프로그래밍 언어
- 응용 개발 도구
- 저장 구조
- 성능
- 접근 방법...

## □ 경제적 요인

- 소프트웨어와 하드웨어 구입 비용
- 유지 보수 비용
- 직원들의 교육 지원 등

# DBMS 성능

---

## □ 현재 DB 성능의 한계는?











- 과연 현재 사용 시스템의 성능은 어느 정도인가?
- 가격은 어느 정도인가?

## □ DBMS 벤치마크 사이트: <http://www.tpc.org>

- 데이터베이스 시스템의 성능 벤치마크
- DBMS + H/W System + Middle Ware System ...
- 분류
  - TPC-C: 트랜잭션 시스템 (OLTP)  
2007년 5월 현재 1위: 100억원, 분당 400만 트랜잭션
  - TPC-H: 의사결정 시스템 (OLAP)

# Top Ten TPC-C by Performance

**Version 5 Results** As of 28-May-2007 4:55 AM [GMT]

Rank	Company	System	tpmC	Price/tpmC	System Availability	Database	Operating System	TP Monitor	Date Submitted	Cluster
1		HP Integrity Superdome-Itanium2/1.6GHz/24MB iL3	4,092,799	2.93 US \$	08/23/07	Oracle Database 10g R2 Enterprise Edt w/Partitioning	HP-UX 11i v3	BEA Tuxedo 8.0	02/27/07	N
2		IBM System p5 595	4,033,378	2.97 US \$	01/22/07	IBM DB2 9	IBM AIX 5L V5.3	Microsoft COM+	01/22/07	N
3		IBM eServer p5 595	3,210,540	5.07 US \$	05/14/05	IBM DB2 UDB 8.2	IBM AIX 5L V5.3	Microsoft COM+	11/18/04	N
4		IBM System p 570	1,616,162	3.54 US \$	11/21/07	IBM DB2 Enterprise 9	IBM AIX 5L V5.3	Microsoft COM+	05/21/07	N
5		IBM eServer p5 595	1,601,784	5.05 US \$	04/20/05	Oracle Database 10g Enterprise Edition	IBM AIX 5L V5.3	Microsoft COM+	04/20/05	N
6		PRIMEQUEST 540 16p/32c	1,238,579	3.94 US \$	12/15/06	Oracle Database 10g Enterprise Edition	Red Hat Enterprise Linux AS 4.0	BEA Tuxedo 8.1	11/30/06	N
7		HP Integrity Superdome ? Itanium2/1.6 GHz-64p/64c	1,231,433	4.82 US \$	06/05/06	Microsoft SQL Server 2005 Enterprise Edt SP1	Microsoft Windows Server 2003 Datacenter Ed.(64-bit)SP1	Microsoft COM+	11/28/05	N
8		HP Integrity rx5670 ClusterItanium2/1.5 GHz-64p/6	1,184,893	5.52 US \$	04/30/04	Oracle Database 10g Enterprise Edition	Red Hat Enterprise Linux AS 3	BEA Tuxedo 8.1	12/08/03	Y
9		IBM eServer pSeries 690 Model 7040-681	1,025,486	5.43 US \$	08/16/04	IBM DB2 UDB 8.1	IBM AIX 5L V5.2	Microsoft COM+	02/17/04	N
10		IBM System p5 570 Model 9117-570	1,025,169	4.42 US \$	05/31/06	IBM DB2 UDB 8.2	IBM AIX 5L V5.3	Microsoft COM+	02/14/06	N



## HP Integrity Superdome - Itanium2/1.6GHz/24MB iL3 - 64p/128c

TPC-C Revision 5.8

Report Date:  
February 27, 2007

Total System Cost

TPC Throughput

Price/Performance

Availability Date

**USD \$11,987,716**

**4,092,799 tpmC**

**USD \$2.93/tpmC**

**August 23, 2007**

Server Processors/Cores/Threads

Database Manager

Operating System

Other Software

Number of Users

64/128/256  
Intel Itanium2 1.6GHz

Oracle Database 10g Release 2  
Enterprise Edition with  
Partitioning

HP-UX 11i v3

TUXEDO 8.0

3,226,200

### Server

Client 1



1000 Base T



10GbE

6 Procurve Switches



Client 95



Fiber Channel



**126 HP StorageWorks MSA 1500**

- 504 HP StorageWorks Enclosures MSA30

- 4,200 36GB 15K RPM Disk Drives

- 2,856 73GB 15K RPM Disk Drives

### HP Integrity Superdome

Itanium2/1.6GHz/24MB iL3 - 64p/128c  
2,048GB Memory

# OLTP vs. OLAP

---

## ❑ OLTP (Online Transaction Processing)

- 단순한 검색, 삽입, 삭제, 수정 등의 작업이 섞인 트랜잭션 처리
- 상대적으로 단순한 트랜잭션을 얼마나 빠르게 많이 처리하느냐가 관건
- 예) Banking System, Online Shopping Mall

## ❑ OLAP (Online Analytic Processing)

- 대규모의 데이터를 분석하는 작업
- 대규모 데이터에서 복잡한 질의를 얼마나 빠르게 처리하느냐가 관건
- 데이터의 수정이나 삭제 등은 상대적으로 덜 중요함
- 예) Data Warehouse, Decision Support System, Data Mining

# 고급 데이터베이스 응용 (1)

---

## □ Data Warehouse

- 데이터 분석을 위한 거대한 데이터 보관소
- 다수의 데이터 소스로부터 분석에 필요한 자료를 미리 가공하여 수집
- 예) Walmart의 각 지사별 판매 정보를 분석을 위해 하나의 저장소에 모아둠.  
월별, 지역별 판매량 및 판매 추이 등의 분석에 사용 → 의사결정 시스템

## □ Data Mining

- 대규모의 데이터로부터 데이터의 패턴이나 규칙등의 유용한 정보를 자동으로 발견하는 응용 및 방법 (Knowledge Discovery)
- 예) 백화점 판매 정보로부터 '40대 강남 거주 기혼 여성'들이 "PRADA"가방을 많이 구입하는 것을 발견해냄 → Target Marketing 등에 활용 가능



## 고급 데이터베이스 응용 (2)

---

### ❑ ERP (Enterprise Resource Planning : 전사적 자원관리)

- 기업 활동에 필요한 전사적인 정보 관리 응용 프로그램의 집합
- 제조, Supply Chain 관리(SCM), 회계, 재무, 인사, 고객-관계 관리(CRM), 데이터웨어하우스 등의 SW 포함
- 기업 프로세스 전반에 밀접하게 관련됨.
- 대표적인 Vendor: SAP, Oracle

### ❑ Embedded Database System

- 내장 기기를 위한 초소형 데이터베이스, DBMS의 일부 기능 제거, 최적화
- 컴퓨팅 파워, 배터리, 저장공간 등의 제약조건

### ❑ Main-Memory (Real-time) Database System

- Performance를 극대화하기 위해 Main-Memory만을 사용하여 DB 응용 처리
- 일반 상용DBMS와 상호 보완적으로 특수한 용도에서 사용

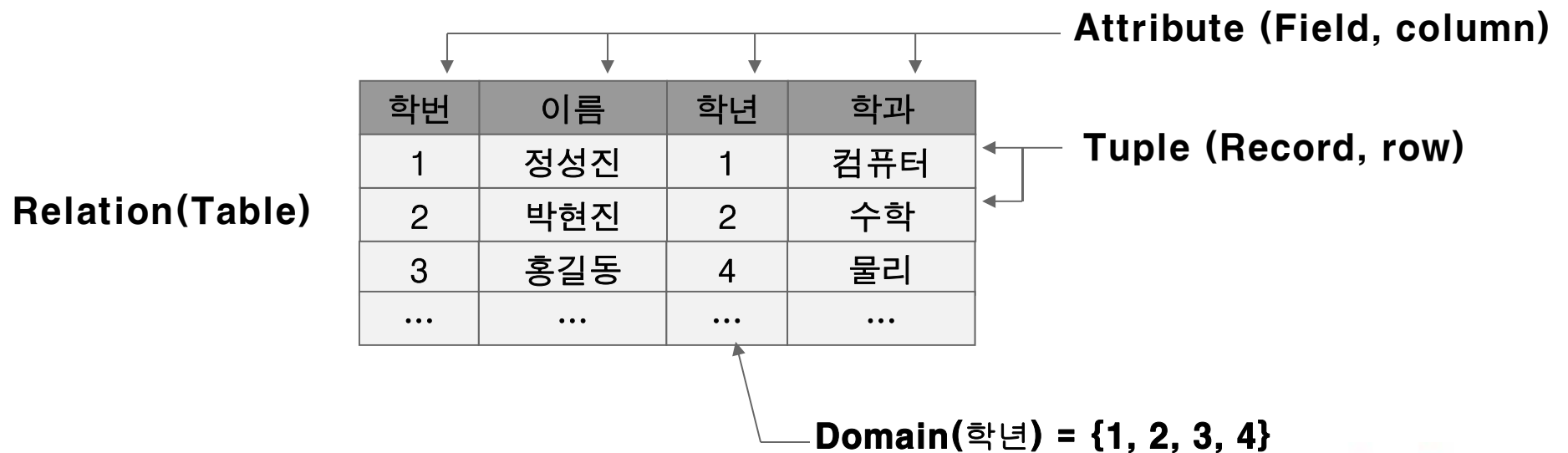
---

# 관계형 모델

# 관계형(Relational) 모델

## □ Key Term

- 데이터베이스(Database): Relation의 집합
- 릴레이션(Relation): Tuple의 집합
- 튜플(Tuple): 테이블의 Row
- 애트리뷰트(Attribute): 릴레이션의 특정 컬럼(열)
- 도메인(Domain): 특정 Attribute가 가질 수 있는 값의 집합



## Table vs. Relation

---

Table-oriented	Set-oriented	Record-oriented
Table	Relation	Record-type, file
Row	Tuple	Record
Column	Attribute	Field

---

### ❑ Relation은 수학적 개념:

- 튜플간, 애트리뷰트 간 순서가 없음.
- 동일한 튜플이 존재할 수 없음(집합이므로)

### ❑ 실제 **DBMS**의 테이블은 순서, 중복 등이 존재

## 관계형 모델의 특징

---

### ❑ Attribute의 값은 원자값(atomic)이어야 함

- 값의 집합, 복합value, Multivalue 등은 가질 수 없음

### ❑ Schema

- 데이터베이스의 구조를 정의
- specifies name of relation, plus name and type of each column

### ❑ Null

- 값이 지정되지 않았음을 의미하는 특별한 값
- 모든 Domain은 Null값을 포함.

### ❑ Key

- 릴레이션에는 동일한 튜플이 존재할 수 없음.
- 하나의 튜플을 다른 튜플과 구별하기 위한 키(Key)가 필요

# Primary Key (PK)

---

## □ Primary Key (PK) : 기본키

- 릴레이션에서 튜플을 구분하기 위하여 사용하는 기본 키
- 하나의 애트리뷰트, 또는 애트리뷰트의 집합(복합키) 가능
- 관리자에 의해 릴레이션 생성시 정의됨 (자동으로 Index생성됨)
- 동일한 PK를 지닌 레코드가 존재할 수 없음

## □ 기타

- Candidate Key (후보키): 튜플을 식별할 수 있는 최소한의 애트리뷰트 집합
  - 하나의 릴레이션에는 PK가 될 수 있는 키가 여러 개 있을 수 있음
  - 유일성과 최소성이 있으면 candidate key가 될 수 있음.
- Alternative Key (대체키): 후보키 중 기본키가 아닌것
- Super key (수퍼키): 유일성만 가지고 최소성을 가지지 않은 키
- Composite key (복합키): 둘 이상의 애트리뷰트가 하나의 Key를 이루는 경우

# Primary Key 예

## □ 학생

- 기본키: 학번
- 후보키: 기본키와 동일
- 수퍼키: (학번, 이름), (학번, 학년)...
- 주민등록번호 등이 있다면 후보키가 될 수 있음

학생  
(STUDENT)

<u>학번</u> (Sno)	이름 (Sname)	학년 (Year)	학과 (Dept)
100	나 수 영	4	컴퓨터
200	이 찬 수	3	전기
300	정 기 태	1	컴퓨터
400	송 병 길	4	컴퓨터

## □ 등록

- 기본키: (학번, 과목번호)
- 후보키: 기본키와 동일
- 학번이나 과목번호 만으로는 키가 되지 못함
- 등록번호와 같이 별도의 단일키를 추가하여 PK로 지정할 수도 있음.

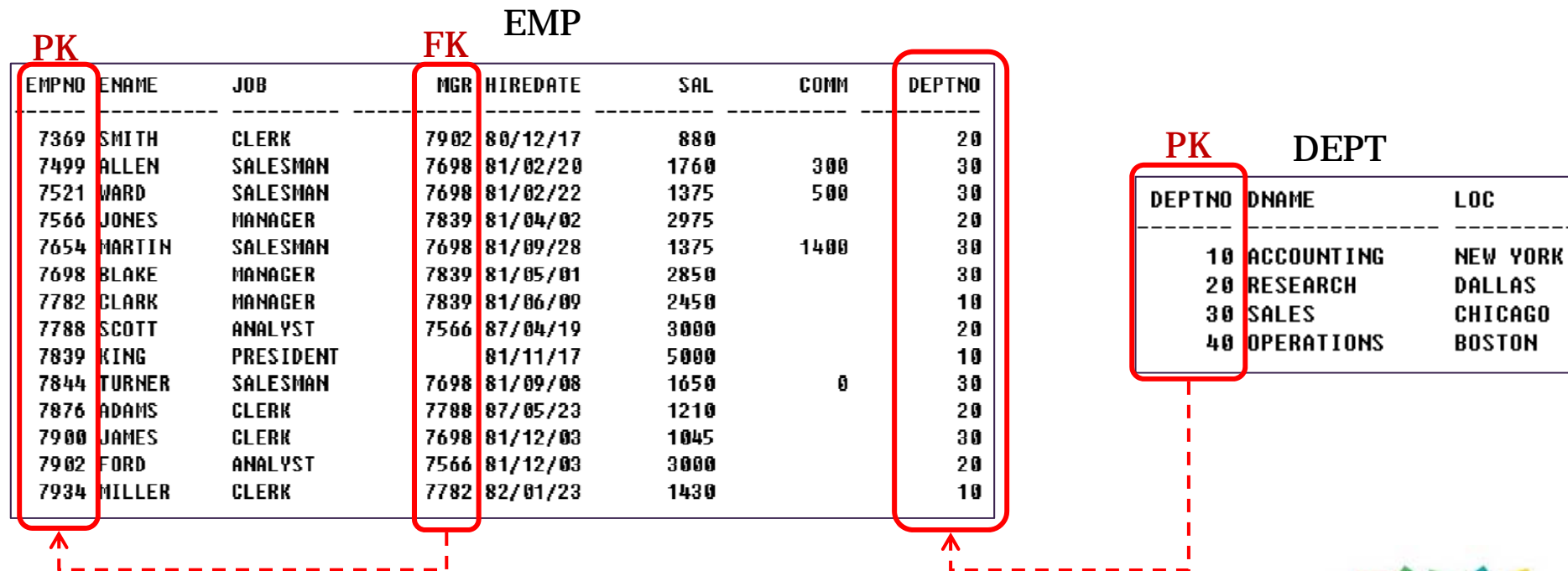
등록  
(ENROL)

<u>학번</u> (Sno)	<u>과목번호</u> (Cno)	성적 (Grade)
100	C413	A
100	E412	A
200	C123	B
300	C312	A

# Foreign Key

## Foreign Key(FK): 외래키

- 기본키를 참조하는 애트리뷰트
- 다른 릴레이션의 튜플을 대표
- 릴레이션 간의 관계를 나타내기 위하여 사용
- Null 가능 (참조되지 않음 의미)





## Integrity Constraint(**무결성 제약**)

---

### □ 개체 무결성 (**Entity Integrity**)

- 기본키의 값은 Null이 될 수 없다.

### □ 참조 무결성 (**Referential Integrity**)

- 외래키의 값은 참조된 릴레이션의 기본키 값이거나 Null이다.

# 관계형 모델의 연산

---

## ❑ Relational Algebra

- select:  $\sigma$
- project:  $\Pi$
- union:  $\cup$
- set difference:  $-$
- Cartesian product:  $\times$
- rename:  $\rho$

## ❑ Structured Query Language (SQL)

# SQL (Structured Query Language)

---

## ❑ SQL Categories

- Query: **SELECT**
- DML(Data Manipulation Language): **INSERT, UPDATE, DELETE, (MERGE)**
- Transaction Control: **COMMIT, ROLLBACK, (SAVEPOINT)**
- DDL (Data Definition Language): **CREATE, DROP, TRUNCATE, ALTER, (RENAME)**
- DCL (Data Control Language): **GRANT, REVOKE**

## ❑ Extension

- **PL/SQL**: Oracle
- Transact-SQL (T-SQL): Microsoft, Sybase
- SQL PL: IBM

# Relational Model의 장단점

---

## □ 장점

- 모델이 단순: 모든 것이 테이블
- 사용이 쉽다: 어떻게보다는 무엇이 필요한가에 집중
- 성능이 우수하다: 다양한 질의 최적화 기술, 오랜 기술 축적
- 표준화가 잘 되어 있다: SQL

## □ 단점

- 표현의 한계: 모든 것이 테이블???
- 성능: 과도한 테이블화로 많은 JOIN이 필요.
  - 반정규화, 비정규화

---

# ORACLE 기본 사용법

# Oracle

---

## ❑ 객체-관계 데이터 모델을 지원하는 관계 데이터베이스 시스템

### ❑ History

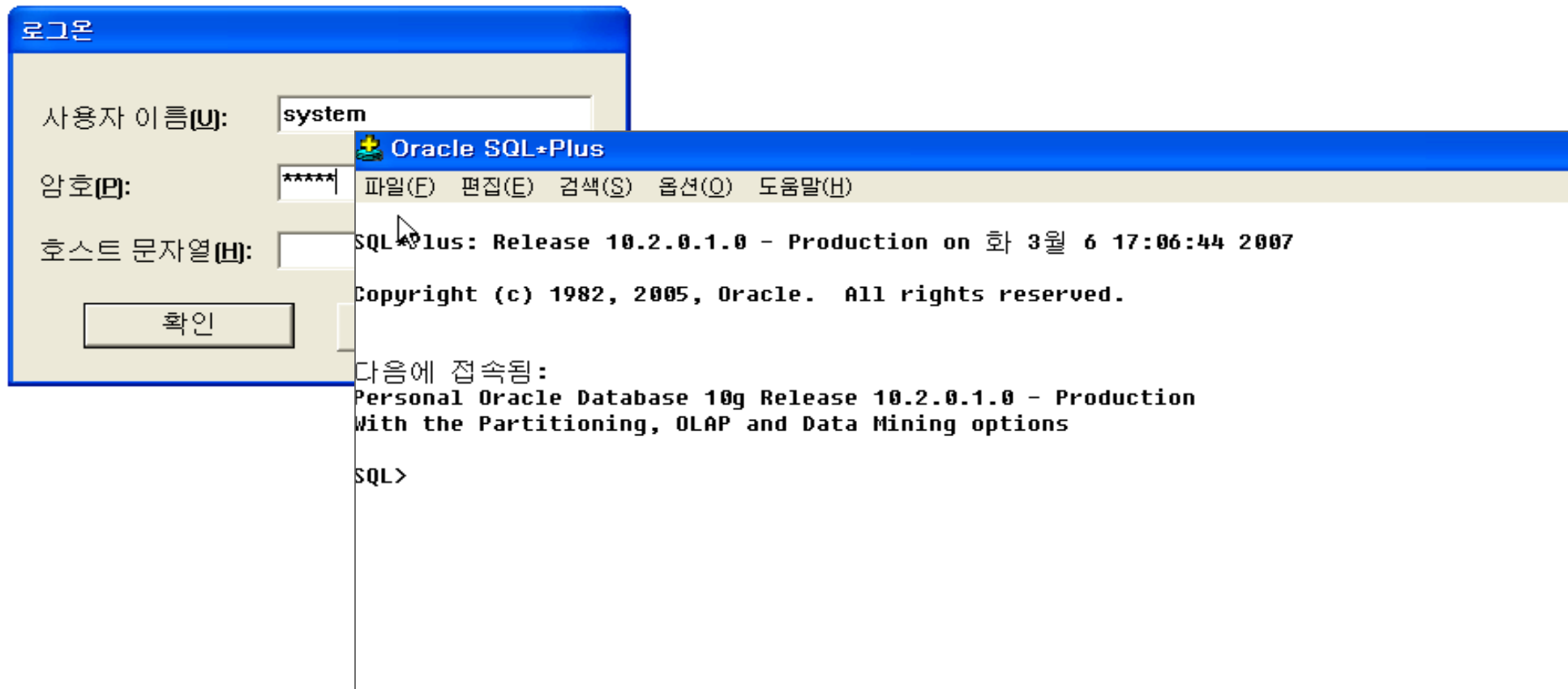
- 1979: Oracle V2 - 최초의 상용 RDBMS (basic SQL)
- 1992: Oracle 7
- 1999: Oracle 8i - 객체 관계형 데이터 모델과 인터넷 관련 기능
- 2001: Oracle 9i - XML과 애플리케이션 서버 기능
- 2003: Oracle 10g - Grid 컴퓨팅을 위해 설계

# SQL\*PLUS

❑ 시작메뉴>오라클>응용프로그램개발>SQL\*PLUS

❑ Command Line

– sqlplus [user/[passwd]]



# iSQL\*PLUS

- ❑ SQL\*PLUS의 인터넷 버전 (별도의 서비스가 실행중이어야함)
- ❑ <http://localhost:5560/isqlplus>





## 작업 영역

SQL, PL/SQL 및 SQL\*Plus 문을 입력하십시오.

지우기

```
select * from tab;
```

I

실행

스크립트 로드

스크립트 저장

취소

TNAME	TABTYPE	CLUSTERID
SYSCATALOG	SYNONYM	
CATALOG	SYNONYM	
TAB	SYNONYM	
COL	SYNONYM	
TABQUOTAS	SYNONYM	

## 오라클의 시동과 종료

### ❑ 제어판 > 관리도구 > 서비스 에서 관련 서비스를 시작/종료

NT LM Security Support Provider	명명...	수동	로컬 시스템
NVIDIA Display Driver Service	Prov...	시작됨 자동	로컬 시스템
Office Source Engine	업데	수동	로컬 시스템
OracleDBConsoleorcl		수동	로컬 시스템
OracleJobSchedulerORCL		사용 안 함	로컬 시스템
OracleOraDb10g_home1iSQL*Plus	iSQ...	수동	로컬 시스템
OracleOraDb10g_home1TNSListener		수동	로컬 시스템
OracleServiceORCL		수동	로컬 시스템
Performance Logs and Alerts	이미...	수동	네트워크 서비스
Plug and Play	사용...	시작됨 자동	로컬 시스템
Portable Media Serial Number Service	Retri...	수동	로컬 시스템
Print Spooler	나중...	시작됨 자동	로컬 시스템
Protected Storage	개인...	시작됨 자동	로컬 시스템
QoS RSVP	QoS...	수동	로컬 시스템

# SCOTT 스키마 사용하기

---

## ❑ SCOTT 스키마: 오라클 실습용 데이터베이스 스키마

### ❑ 사용법

- SQL\*PLUS 실행
- System 계정으로 로그인 (System / Manager)
- SCOTT계정 Lock 제거
  - **ALTER USER SCOTT UNLOCK**
- SCOTT계정 암호 변경
  - **ALTER USER SCOTT IDENTIFIED BY xxxxxx**
- SCOTT 으로 접속 가능
- SCOTT 접속
  - **CONN SCOTT**

## SCOTT 스키마 살펴보기

---

### ❑ 스키마 속의 테이블 / **Object** 전체 확인하기

- SELECT \* FROM tab;

### ❑ 각 테이블의 구조 확인

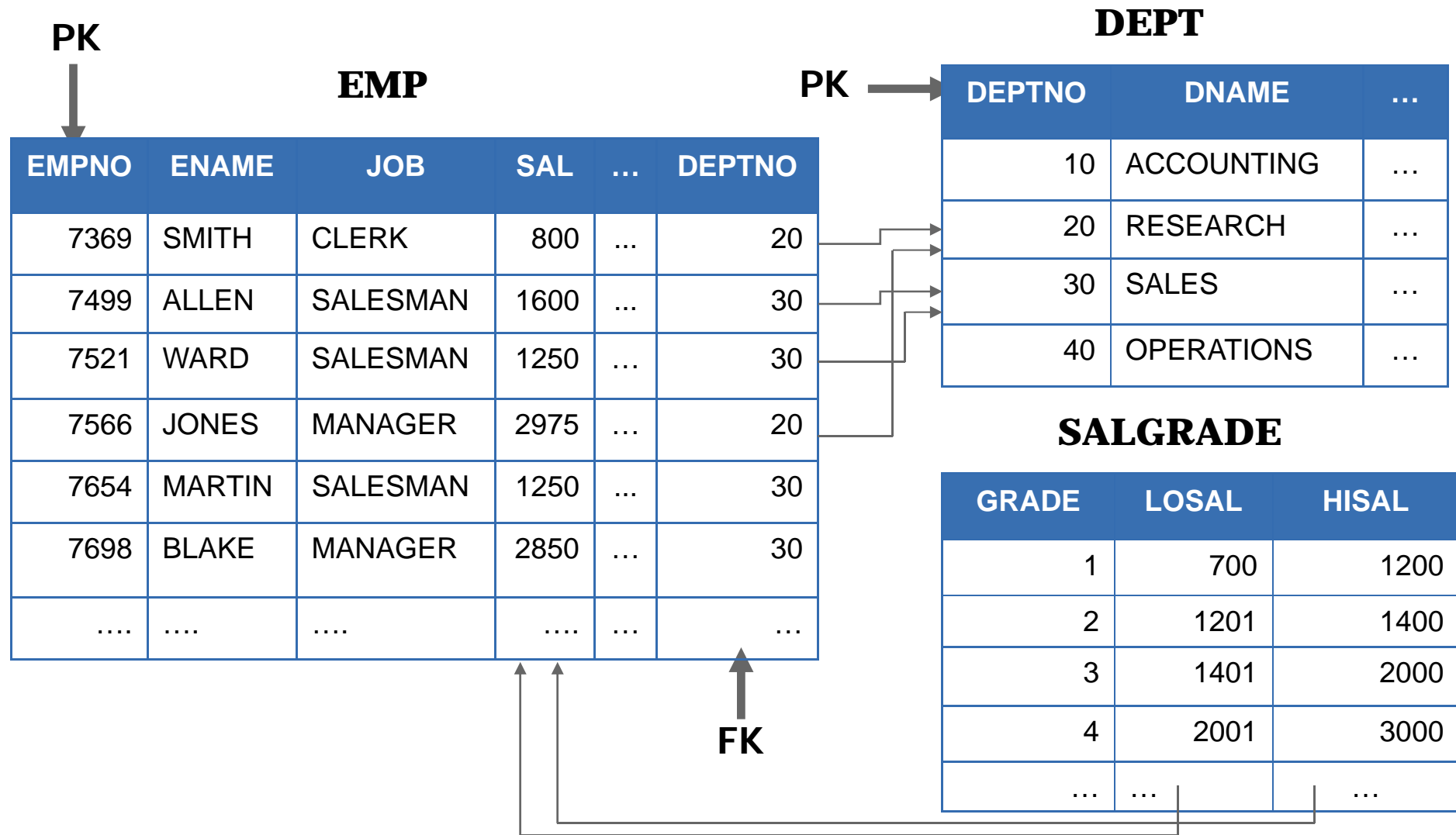
- DESC 테이블이름

- desc emp
- desc dept
- desc bonus
- desc salgrade

### ❑ 테이블 내용 확인하기

- SELECT \* FROM 테이블이름;
- SELECT \* FROM emp;
- SELECT \* FROM dept;
- SELECT \* FROM salgrade;

# SCOTT 스키마 구조



### □ SQL\*Plus에서 SQL 문장 작성 시 유의 사항

- SQL은 필요에 따라 여러줄로 입력할 수 있다. (종료는 ; 을 입력해야함)
- 모든 SQL문은 ‘;’으로 끝난다.
- SQL문은 대소문자를 구별하지 않는다.
- 데이터는 대소문자를 구별한다.
- 문자열은 ‘ ’ 를 이용하여 입력

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

- 최소한 SELECT절과 FROM절이 있어야 SELECT 문장이 가능하다.
- 컬럼제목: column 명이 대문자로 표시된다.
- 기본 정렬: 숫자는 오른쪽 정렬, 문자열.날짜 등은 왼쪽 정렬

## 유용한 SQL\*PLUS 명령 (1)

---

### ❑ **SELECT** 결과 출력시 가로/세로 길이 설정

- set pagesize 숫자
- set linesize 숫자

### ❑ 문자열 컬럼의 최대 출력길이 지정

- COLUMN 컬럼이름 FORMAT A숫자

### ❑ 마지막 수행한 **SQL** 문 재실행

- RUN
- /

### ❑ 마지막 수행한 **SQL** 문 표시

- LIST

### ❑ 수행한 **SQL**문의 특정 부분 문자열 치환

- c/변경대상문자열/변경할문자열

## 유용한 SQL\*PLUS 명령 (2)

---

### ❑ 파일 에디터로 **SQL** 편집

- ED 파일이름

### ❑ **SQL** 파일 수행

- @파일이름;

### ❑ **SQL**문 저장

- SAVE *파일이름*

### ❑ 실행내용 **Spooling**

- SPOOL *파일이름*
- SPOOL OFF



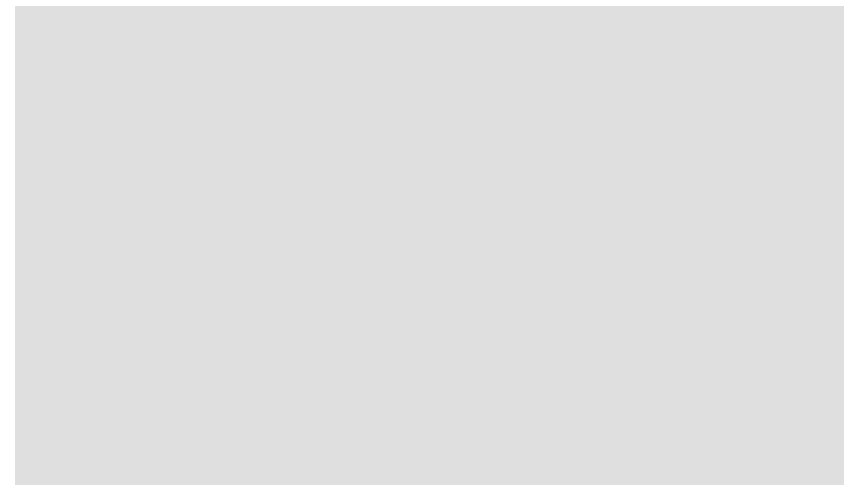
---

# - SQL Query I

## 목 차

---

1. 기본 SELECT
2. SINGLE-ROW Function



---

# 기본 SELECT

# SELECT

---

❑ 데이터베이스에서 원하는 데이터를 검색, 추출

## ❑ Syntax

- **SELECT** [ALL | DISTINCT] 열\_리스트  
[**FROM** 테이블\_리스트]  
[**WHERE** 조건]  
[**GROUP BY** 열\_리스트 [**HAVING** 조건]]  
[**ORDER BY** 열\_리스트 [ASC | DESC]];

## ❑ 기능

- Projection: 원하는 컬럼 선택
- Selection: 원하는 튜플 선택
- Join: 두개의 테이블 결합
- 기타: 각종 계산, 정렬, 요약(Aggregation)

# SELECT의 기능

## Projection

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	80/12/17	880		20
7499	ALLEN	SALESMAN	7698	81/02/20	1760	300	30
7521	WARD	SALESMAN	7698	81/02/22	1375	500	30
7566	JONES	MANAGER	7839	81/04/02	2975		20
7654	MARTIN	SALESMAN	7698	81/09/28	1375	1400	30
7698	BLAKE	MANAGER	7839	81/05/01	2850		30
7782	CLARK	MANAGER	7839	81/06/09	2450		10
7788	SCOTT	ANALYST	7566	87/04/19	3000		20
7839	KING	PRESIDENT		81/11/17	5000		10
7844	TURNER	SALESMAN	7698	81/09/08	1650	0	30
7876	ADAMS	CLERK	7788	87/05/23	1210		20
7900	JAMES	CLERK	7698	81/12/03	1045		30
7902	FORD	ANALYST	7566	81/12/03	3000		20
7934	MILLER	CLERK	7782	82/01/23	1430		10

Selection

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

Join


# 기본 SELECT

---

## □ 형식

- **SELECT \*|{[DISTINCT] column|expression [alias],...}**  
**FROM table;**

## □ 내용

- \*: 모든 컬럼 반환
- DISTINCT: 중복된 결과 제거
- SELECT 컬럼명: Projection
- FROM: 대상 테이블
- ALIAS: 컬럼 이름 변경
- Expression: 기본적인 연산 및 함수 사용 가능

## SELECT 실습

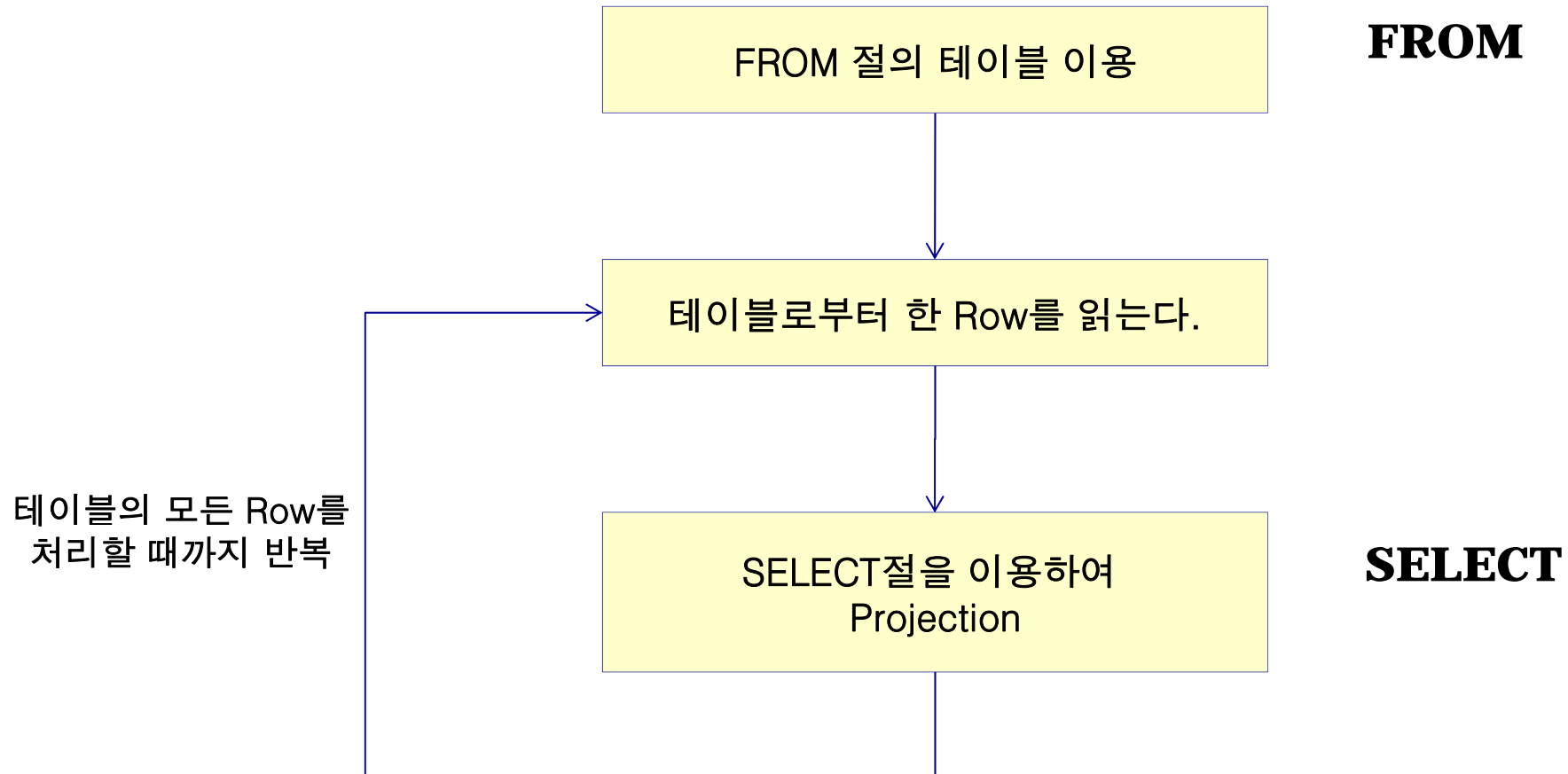
- ❑ SELECT \* FROM emp;
- ❑ SELECT ename FROM emp;
- ❑ SELECT ename, job FROM emp;
- ❑ SELECT ename 이름 FROM emp;

```
SQL> select * from emp;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	80/12/17	800		20
7499	ALLEN	SALESMAN	7698	81/02/20	1600	300	30
7521	WARD	SALESMAN	7698	81/02/22	1250	500	30
7566	JONES	MANAGER	7839	81/04/02	2975		20
7654	MARTIN	SALESMAN	7698	81/09/28	1250	1400	30
7698	BLAKE	MANAGER	7839	81/05/01	2850		30
7782	CLARK	MANAGER	7839	81/06/09	2450		10
7788	SCOTT	ANALYST	7566	87/04/19	3000		20
7839	KING	PRESIDENT		81/11/17	5000		10
7844	TURNER	SALESMAN	7698	81/09/08	1500	0	30
7876	ADAMS	CLERK	7788	87/05/23	1100		20
7900	JAMES	CLERK	7698	81/12/03	950		30
7902	FORD	ANALYST	7566	81/12/03	3000		20
7934	MILLER	CLERK	7782	82/01/23	1300		10

## SELECT, FROM 절 처리방법

---



## 산술연산

---

### □ 기본적인 산술연산 사용 가능

- +, -, \*, /, 부호, 괄호 등
- 우선순위: 부호, \* / , + -
- 컬럼 이름, 숫자
- 예
  - SELECT ename, (sal+200) \* 12 FROM emp;
  - SELECT ename, -sal \* 10 FROM emp;

```
SQL> SELECT ename, (sal+200) * 12 FROM emp;
```

ENAME	(SAL+200)*12
SMITH	12000
ALLEN	21600
WARD	17400
JONES	38100
MARTIN	17400
BLAKE	36600



# NULL

---

- ❑ 아무런 값도 정해지지 않았음을 의미
- ❑ 어떠한 데이터타입에도 사용가능
- ❑ **NOT NULL**이나 **Primary Key** 속성에는 사용할 수 없음
- ❑ **NULL**을 포함한 산술식은 **NULL**
  - SELECT sal, comm, (sal+comm)\*12 FROM emp;
- ❑ **NVL(expr1, expr2)**
  - expr1이 NULL이면 expr2를 출력한다.
  - 데이터타입이 호환 가능 하여야 함.
  - SELECT sal, comm, (sal+NVL(comm,0))\*12 FROM emp;

## Column Alias

- ❑ 컬럼의 제목을 변경
- ❑ 큰따옴표(“ ”)을 사용하여 **alias**내에 공백이나 특수문자를 포함할 수 있다.
- ❑ 형태
  - SELECT ename name FROM emp;
  - SELECT ename as name FROM emp;
  - SELECT ename “as” FROM emp;
  - SELECT (sal + comm) “Annual Salary” FROM emp;

```
SQL> select empno no, ename as name, job "to do" from emp;
```

NO	NAME	to do
7369	SMITH	CLERK
7499	ALLEN	SALESMAN
7521	WARD	SALESMAN
7566	JONES	MANAGER
7654	MARTIN	SALESMAN
7698	BLAKE	MANAGER
7782	CLARK	MANAGER
7788	SCOTT	ANALYST
7839	KING	PRESIDENT
7844	TURNER	SALESMAN
7876	ADAMS	CLERK

## Literal

---

- ❑ **SELECT** 절에 사용되는 문자, 숫자, **Date** 타입 등의 상수
- ❑ **Date** 타입이나 문자열은 작은따옴표 ( ' ')로 둘러싸야 함
- ❑ 문자열 결합(**Concatunation**) 연산자: ||
- ❑ 예
  - SELECT ename, 1000, SYSDATE FROM emp;
  - SELECT 'Name is ' || ename || ' and no is ' || empno FROM emp;

```
SQL> SELECT 'Name is ' || ename || ' and no is ' || empno FROM emp;

'NAMEIS' || ENAME || 'ANDNOIS' || EMPNO
-----
Name is SMITH and no is 7369
Name is ALLEN and no is 7499
Name is WARD and no is 7521
Name is JONES and no is 7566
Name is MARTIN and no is 7654
Name is BLAKE and no is 7698
Name is CLARK and no is 7782
Name is SCOTT and no is 7788
```

# WHERE

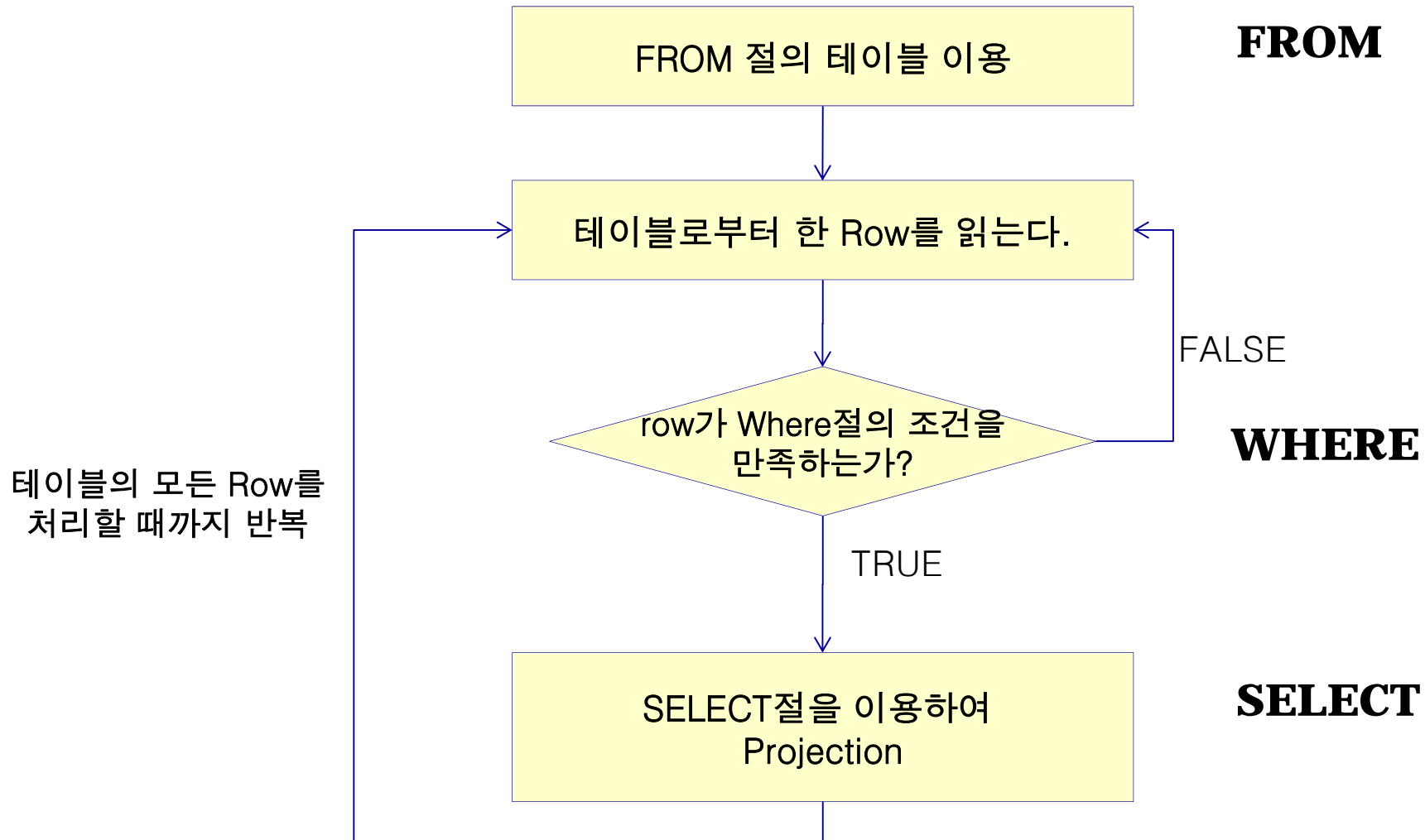
---

## ❑ 조건을 부여하여 만족하는 **ROW Selection**

### ❑ 연산자

- =, !=, >, <, <=, >=
- IN : 집합에 포함되는가?
- BETWEEN a AND b : a 와 b 사이?
- LIKE: 문자열 부분 검색
- IS NULL, IS NOT NULL: NULL인지 검색
- AND, OR: 둘다 만족? 둘 중 하나만 만족?
- NOT: 만족하지 않음?
- ANY, ALL : 집합 중 어느한열, 집합 중 모든 열 (다른 비교연산자와 함께 사용)
- EXIST: 결과 Row가 하나라도 있나? (subquery에서)

## WHERE 절 처리 방법



## LIKE연산

---

### ❑ Wildcard를 이용한 문자열 부분 매칭

#### ❑ Wildcard

- % : 임의의 길이의 문자열 (공백 문자 가능)
- \_ : 한 글자

#### ❑ Escape

- **ESCAPE** 뒤의 문자열로 시작하는 문자는 Wildcard가 아닌 것으로 해석

#### ❑ 예

- `ename LIKE 'KOR%'` : 'KOR'로 시작하는 모든 문자열(KOR가능)
- `ename LIKE 'KOR_'` : 'KOR'다음에 하나의 문자가 오는 모든 문자열
- `ename LIKE 'KOR/%%' ESCAPE '/'` : 'KOR%'로 시작하는 모든 문자열

## 연산자 우선 순위

---

- ① **Arithmetic operators**
- ② **Concatenation operator**
- ③ **Comparison conditions**
- ④ **IS[NOT] NULL, LIKE, [NOT] IN**
- ⑤ **[NOT] BETWEEN**
- ⑥ **NOT logical condition**
- ⑦ **AND logical condition**
- ⑧ **OR logical condition**

## 논리 연산자의 결과값

### □ NULL을 주의:

- NULL이 있으면 기본적으로 NULL, 확실히 답이 나오는 경우만 계산 가능

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



## ORDER BY

---

### ❑ 주어진 컬럼 리스트의 순서로 결과를 정렬

### ❑ 결과 정렬 방법

- ASC : 오름차순 (작은값→큰값) (default)
- DESC: 내림차순(큰값→작은값)

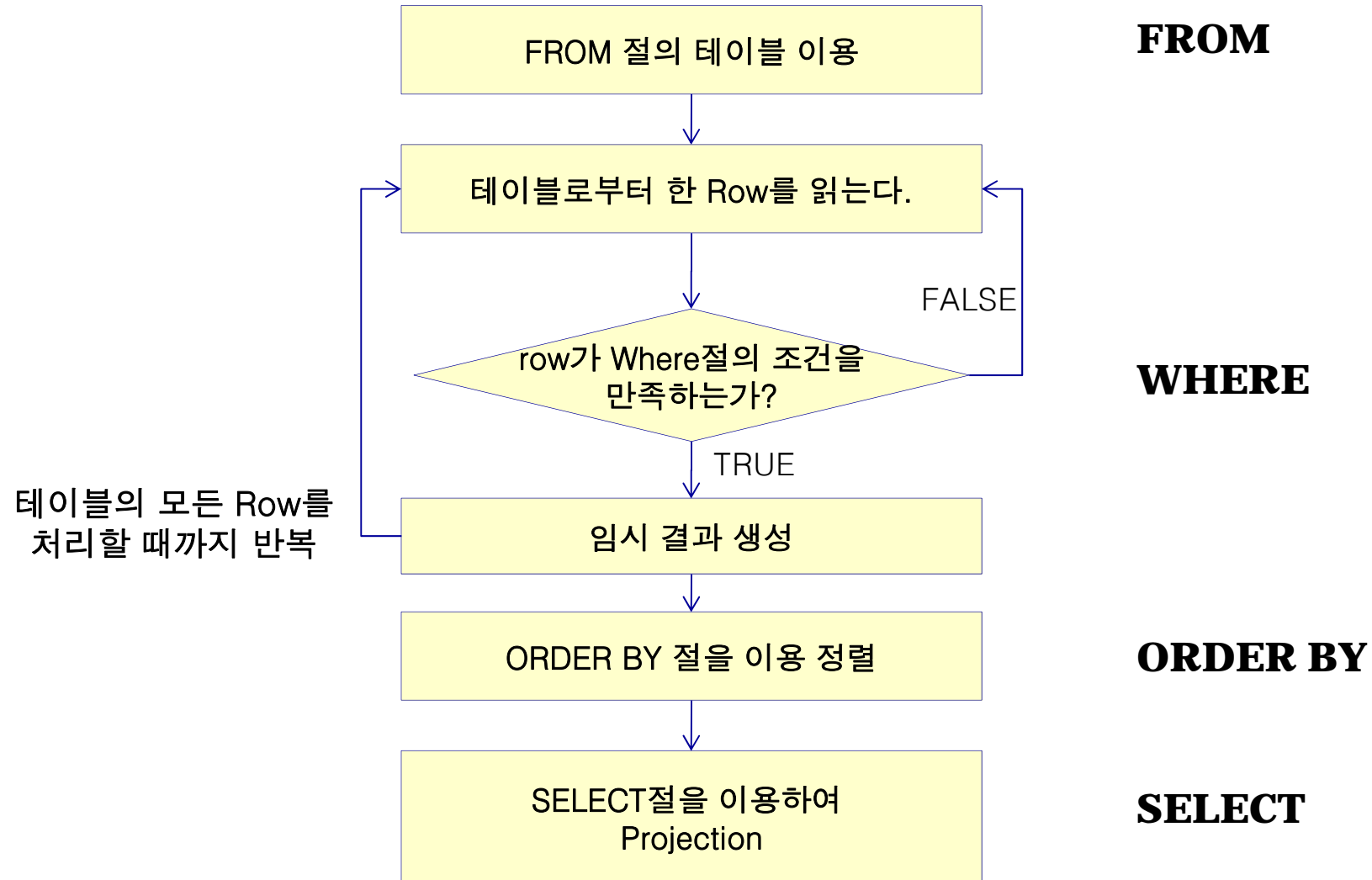
### ❑ 여러 컬럼 정의 가능

- 첫번째 컬럼이 같으면 두번째 컬럼으로, 두번째 컬럼도 같으면...

### ❑ 컬럼 이름대신 **Alias**, **expr**, **SELECT** 절상에서의 순서(1, 2, 3...) 도 사용가능

- 예) `SELECT * FROM emp WHERE deptno, sal DESC`
  - 부서번호순으로 정렬하고, sal가 높은 사람부터 출력하시오

## ORDER BY절 처리 방법



---

# SINGLE ROW FUNCTION

# SQL Function

---

## ❑ **Single-Row Function** : 하나의 **Row**를 입력으로 받는 함수

- 숫자함수
- 문자함수
- 날짜함수
- 변환함수
- 기타함수

## ❑ **Aggregation Function**: 집합함수

## ❑ **Analytic Function**: 분석함수

## ❑ **Regular Expression**: 정규표현식 (Oracle 10g 이상)

## 문자열 함수

Function	설명
CONCAT(s1,s2)	문자열 결합
INITCAP(s)	첫글자만 대문자로 변경
LOWER(s)	소문자로 변경
UPPER(s)	대문자로 변경
LPAD(s1,n,s2)	문자열의 왼쪽 채움 (길이:n, 채움문자 s2)
RPAD(s1,n,s2)	문자열 오른쪽 채움 (길이:n, 채움문자 s2)
LTRIM(s,c)	문자열 왼쪽 c문자열 제거
RTRIM(s,c)	문자열 오른쪽 c문자열 제거
CHR(n)	ASCII값이 n인 문자 반환
REPLACE(s,p,r)	문자열 치환, S속의 p문자열을 r로 치환
SUBSTR(s,m,n)	부분 문자열, m번째부터 길이 n인 문자열 반환
TRANSLATE(s,from,to)	s에서 from 문자열의 각 문자를 to문자열의 각 문자로 변환
ASCII(s)	ASCII값 반환
INSTR(s1,s2,m,n)	문자열 검색, s1의 m번째부터 s2 문자열이 나타나는 n번째 위치 반환
LENGTH(s)	문자열 길이 반환

## 문자열 함수 예

### □ 대소문자 변환

Function	Result
LOWER('Database system')	database system
UPPER('Database system')	DATABASE SYSTEM
INITCAP('Database system')	Database System

### □ 문자열 조작

함수	결과
CONCAT('Data', 'Base')	DataBase
SUBSTR('Database',2,4)	atab
LENGTH('database')	8
INSTR('Database', 'b')	5
LPAD(salary,10,'*')	*****24000
RPAD(salary, 10, '*')	24000*****
TRIM('#' FROM '##Database###')	Database

## 숫자 함수

Function	설명	Example	Result
ABS(n)	절대값	ABS(-5)	5
CEIL(n)	n 보다 크거나 같은 최소 정수	CEIL(-2.4)	-2
FLOOR(n)	n 보다 작거나 같은 최대 정수	FLOOR(-2.4)	-3
MOD(m,n)	나머지	MOD(13,2)	1
POWER(m,n)	m의 n승	POWER(2,3)	8
ROUND(m,n)	소수점아래 n자리까지 반올림	ROUND(4.567,2)	4.57
TRUNC(m,n)	소수점아래 n자리미만 버림	TRUNC(4.567,2)	4.56
SIGN(n)	부호 (1, 0, -1)	SIGN(-10)	-1

# Date 타입

---

## □ Oracle의 Date Type

- century, year, month, day, hours, minutes, seconds 등 포함한 내부 표현 (7Bytes)
- Date Format에 따라 출력, 입력됨
- 기본 Date Format: 'RR/MM/DD' or 'DD-MON-RR'
  - RR은 Y2K고려 2자리 년도 표기 (00~49:2000년대 / 50~99: 1900년대)
- 포맷 확인
  - `SELECT value FROM nls_session_parameters WHERE parameter = 'NLS_DATE_FORMAT';`



## Date 함수

---

Function	Purpose
ADD_MONTHS(d,n)	d날짜에 n달 더함
LAST_DAY(d)	d의 달의 마지막 날
MONTHS_BETWEEN(d1,d2)	d1, d2사이의 달 수
NEW_TIME(d,z1,z2)	z1타임존의 d에서 z2타임존의 날짜 생성
NEXT_DAY(d,day)	d날 후의 첫 day요일의 날짜
ROUND(d,fmt)	fmt에 따른 날짜반올림
TRUNC(d,fmt)	fmt에 따른 날짜반올림
SYSDATE	현재 날짜 시간 반환

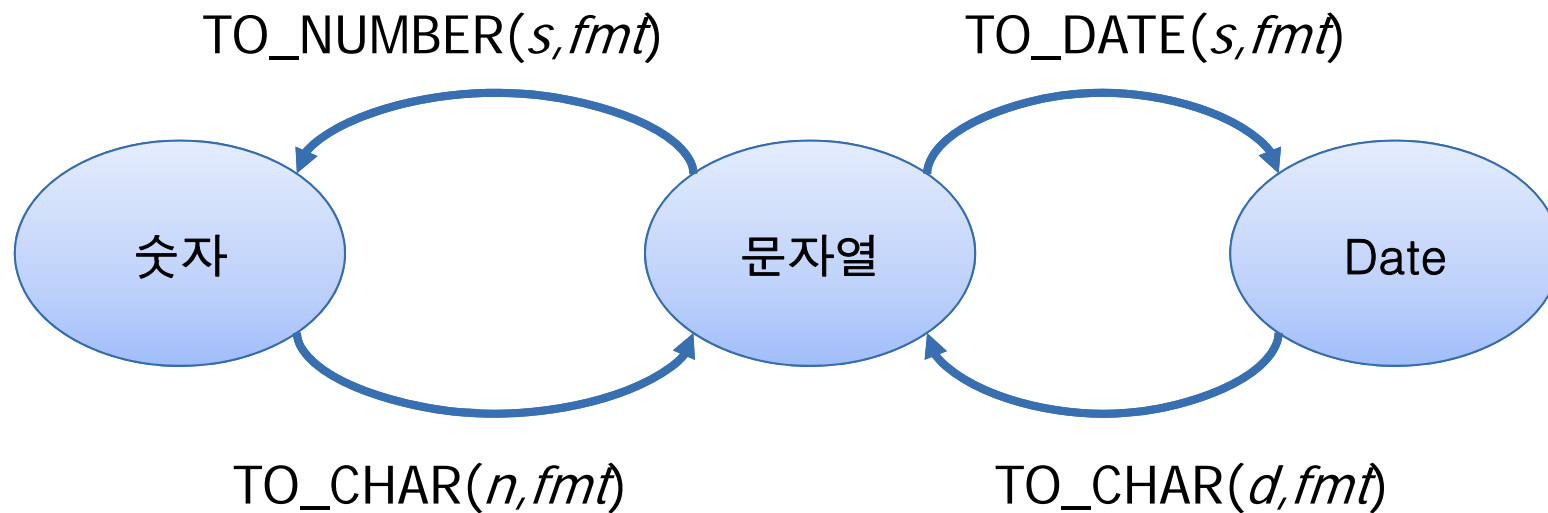
## Date 함수 예

FUNCTION	RESULT
MONTHS_BETWEEN ('01-SEP-95','11-JAN-94')	19.677419
ADD_MONTHS ('11-JAN-94',6)	'11-JUL-94'
NEXT_DAY ('01-SEP-95','FRIDAY')	'08-SEP-95'
LAST_DAY('01-FEB-95')	'28-FEB-95'

현재 날짜를 '25-JUL-95' 가정	
ROUND(SYSDATE,'MONTH')	01-AUG-95
ROUND(SYSDATE , 'YEAR')	01-Jan-96
TRUNC(SYSDATE , 'MONTH')	01-JUL-95
TRUNC(SYSDATE , 'YEAR')	01-JAN-95

## 변환 함수

- ❑ 묵시적 변환: 변환함수 없이도 어느정도는 자동으로 변환됨
- ❑ 자동으로 변환되지 않을때는 명시적인 변환 함수 사용



## Date 변환 포맷

### □ 예제는 **American** 포맷인 경우

fmt	Description	Example
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

fmt	Description	Example
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,YY Y	1,2,3-digit year.	
YYYY, SYYYYY	4-digit year. "S" prefixes BC dates with "-".	

## Date 변환 포맷 (계속)

fmt	Description	Example
YEAR, SYEAR	Year, spelled out. "S" prefixes BC dates with "-".	
RR	<p>Given a year with 2 digits.</p> <p><i>Returns a year in the next century if the year is &lt;50 and the last 2 digits of the current year are ≥50.</i></p> <p><i>Returns a year in the preceding century if the year is ≥50 and the last 2 digits of the current year are &lt;50.</i></p>	
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.	

## 숫자 변환 포맷

fmt	Description	Example
9	숫자	99999
0	강제로 0 출력	09999
,	지정된 위치에 ,	99,999
.	소수점	999.99
\$	\$ 마크	\$99999
FM	앞부분의 채움 문자(공백) 없음.	FM90.9
L	Local 화폐 단위	L99,999
MI	음수에 - 부호	99999MI
PR	음수에 괄호	99999PR
RN	로마자 (대소문자 따라 다름)	RN rn
S	부호 기호	S99,999
X	16진수	XXX xxx

## 기타 함수

---

### □ NULL 관련

- NVL(*expr1*, *expr2*): *expr1*이 NULL이면 *expr2*, 아니면 *expr1*
- NVL2 (*expr1*, *expr2*, *expr3*): *expr1*이 NOT NULL이면 *expr2*, 아니면 *expr3*
- NULLIF (*expr1*, *expr2*): 두 식이 같으면 NULL 아니면 *expr1*
- COALESCE(*expr1*, *expr2*,...*exprN*): 첫 NOT NULL인 식, 없으면 *expN*

### □ 데이터 타입에 주의

### □ 예

- SELECT *ename*, NVL(TO\_CHAR(*mgr*), 'No Manager')  
FROM emp

# Condition Expression

---

## ❑ CASE

```
– SELECT ename, job, sal, CASE job WHEN 'CLERK' THEN 1.10*sal  
                                WHEN 'MANAGER' THEN 1.15*sal  
                                WHEN 'PRESIDENT' THEN 1.20*sal  
                                ELSE sal END   REVISED_SALARY  
  
FROM emp;
```

## ❑ DECODE

```
– SELECT ename, job, sal, DECODE(job, 'CLERK', 1.10*sal,  
                                'MANAGER', 1.15*sal,  
                                'PRESIDENT', 1.20*sal,  
                                sal) REVISED_SALARY  
  
FROM emp;
```



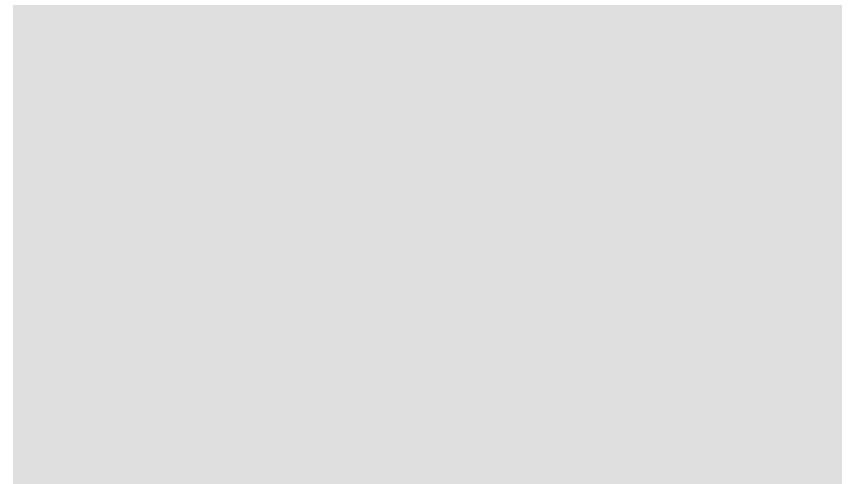
---

# - SQL Query II

## 목 차

---

1. Join (Inner, Outer, Natural)
2. Group & Aggregation
3. Subquery



---

# JOIN

# Join

❑ 둘 이상의 테이블을 합쳐서 하나의 큰 테이블로 만드는 방법

❑ 필요성

- 관계형 모델에서는 데이터의 일관성이나 효율을 위하여 데이터의 중복을 최소화 (정규화)
- Foreign Key를 이용하여 참조
- 정규화된 테이블로부터 결합된 형태의 정보를 추출할 필요가 있음
- 예) 직원의 이름과 직원이 속한 부서명을 함께 보고 싶으면???

EMPNO	ENAME	DEPTNO
7369	SMITH	20
7499	ALLEN	30
7521	WARD	30
7566	JONES	20
7654	MARTIN	30
7698	BLAKE	30
7782	CLARK	10
7788	SCOTT	20
7839	KING	10
7844	TURNER	30
7876	ADAMS	20
7900	JAMES	30
7902	FORD	20
7934	MILLER	10

?

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

## 카티전 프로덕트

### ❑ 두 테이블에서 그냥 결과를 선택하면?

- SELECT ename, dname from emp, dept
- 결과: 두 테이블의 행들의 가능한 모든 쌍이 추출됨
- 일반적으로 사용자가 원하는 결과가 아님.

ENAME	DNAME
SMITH	ACCOUNTING
ALLEN	ACCOUNTING
WARD	ACCOUNTING
JONES	ACCOUNTING
MARTIN	ACCOUNTING
BLAKE	ACCOUNTING
CLARK	ACCOUNTING
SCOTT	ACCOUNTING

### ❑ Cartesian Product

$$X \times Y = \{(x, y) | x \in X \text{ and } y \in Y\}$$

### ❑ Cartesian Product를 막기 위해서는

올바른 Join조건을 WHERE 절에 부여 해야 함.

ALLEN	OPERATIONS
WARD	OPERATIONS
JONES	OPERATIONS
MARTIN	OPERATIONS
BLAKE	OPERATIONS
CLARK	OPERATIONS
SCOTT	OPERATIONS
KING	OPERATIONS
TURNER	OPERATIONS
ADAMS	OPERATIONS
JAMES	OPERATIONS
FORD	OPERATIONS
MILLER	OPERATIONS

56 개의 행이 선택되었습니다.

# Simple Join

---

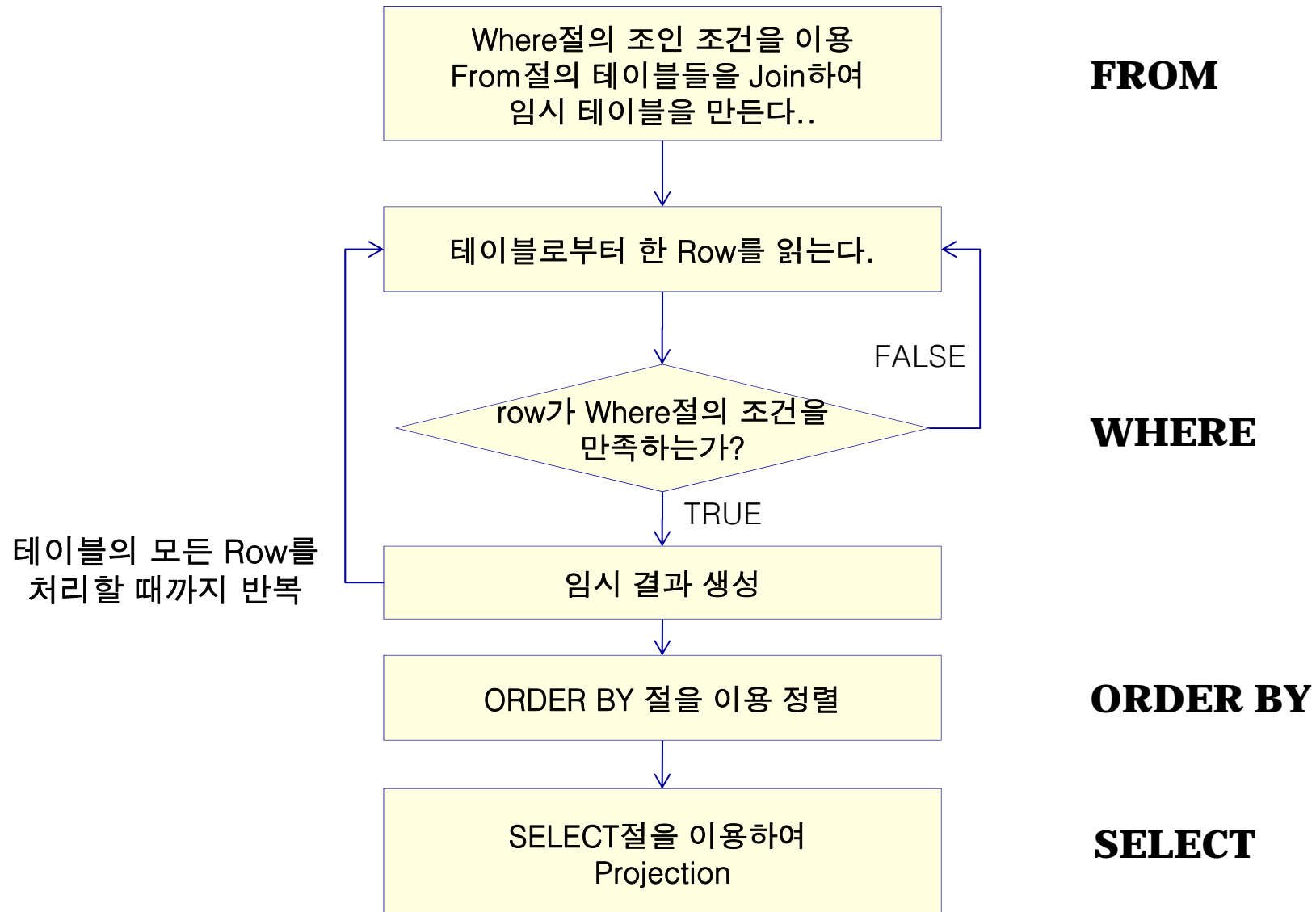
## □ Syntax

```
SELECT t1.col1, t1.col2, t2.col1 ...  
FROM Table1 t1, Table2 t2  
WHERE t1.col3 = t2.col3
```

## □ 설명

- FROM 절에 필요로 하는 테이블을 모두 적는다.
- 컬럼 이름의 모호성을 피하기 위해(어느 테이블에 속하는지 알 수 없음)이 있을 수 있으므로 Table 이름에 Alias 사용 (테이블 이름으로 직접 지칭 가능)
- 적절한 Join 조건을 Where 절에 부여 (일반적으로 테이블 개수 -1 개의 조인 조건이 필요)
- 일반적으로 PK와 FK간의 = 조건이 붙는 경우가 많음

## Join 처리 방법



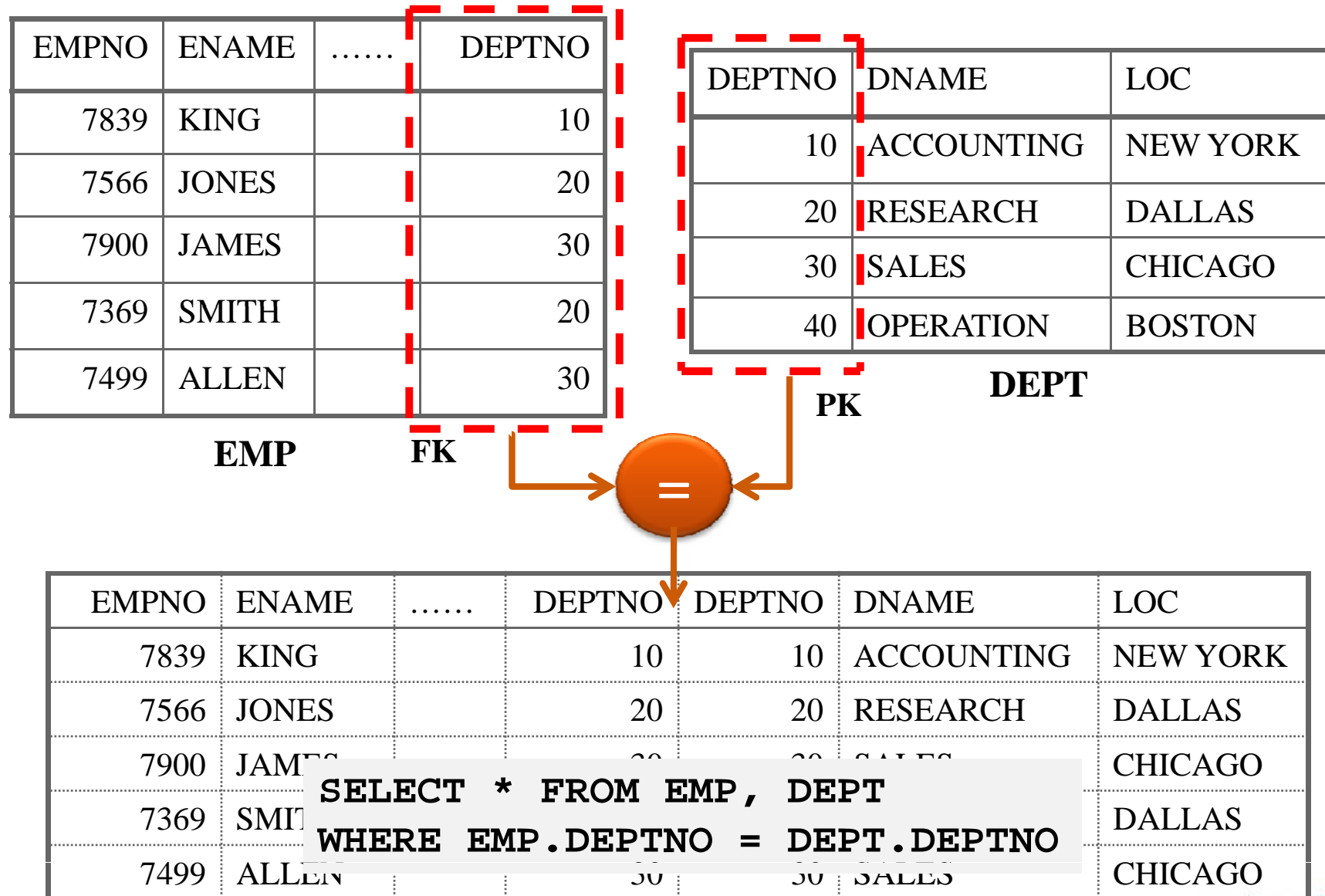
## Join 종류

---

### □ 용어

- Cross Join (Cartesian Product): 모든 가능한 쌍이 나타남
- Inner Join: Join 조건을 만족하는 튜플만 나타남
- Outer Join: Join 조건을 만족하지 않는 튜플 (짜이 없는 튜플)도 null과 함께 나타남
- Theta Join: 조건(theta)에 의한 조인
- Equi-Join: Theta Join & 조건이 Equal (=)
- Natural Join: Equi-join & 동일한 Column명 합쳐짐.
- Self Join: 자기 자신과 조인

# Equi-Join





# Theta Join

## 정의

- 임의의 조건을 Join 조건으로 사용가능
- Non-Equi Join이라고도 함

## 예

```
SELECT e.ename, e.sal, s.grade
FROM emp e, salgrade s
WHERE e.sal BETWEEN s.losal AND s.hisal
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	ENAME	SAL	GRADE
7369	SMITH	CLERK	7902	80/12/17	800	SMITH	800	1
7499	ALLEN	SALESMAN	7698	81/02/20	1600	JAMES	950	1
7521	WARD	SALESMAN	7698	81/02/22	1250	ADAMS	1100	1
7566	JONES	MANAGER	7839	81/04/02	2975	WARD	1250	2
7654	MARTIN	SALESMAN	7698	81/09/28	1250	MARTIN	1250	2
7698	BLAKE	MANAGER	7839	81/05/01	2850	MILLER	1300	2
7782	CLARK	MANAGER	7839	81/06/09	2450	TURNER	1500	3
7788	SCOTT	ANALYST	7566	87/04/19	3000	ALLEN	1600	3
7839	KING	PRESIDENT		81/11/17	5000	CLARK	2450	4
7844	TURNER	SALESMAN	7698	81/09/08	1500	BLAKE	2850	4
7876	ADAMS	CLERK	7788	87/05/23	1100	JONES	2975	4
7900	JAMES	CLERK	7698	81/12/03	950	SCOTT	3000	4
7902	FORD	ANALYST	7566	81/12/03	3000	FORD	3000	4
7934	MILLER	CLERK	7782	82/01/23	1300	KING	5000	5

GRADE	LOSAL	HISAL
1	700	1200
2	1201	1400
3	1401	2000
4	2001	3000
5	3001	9999

# Outer Join

---

## □ 정의

- Join 조건을 만족하지 않는 (짜이 없는 ) 튜플의 경우 Null을 포함하여 결과를 생성
- 모든 행이 결과 테이블에 참여

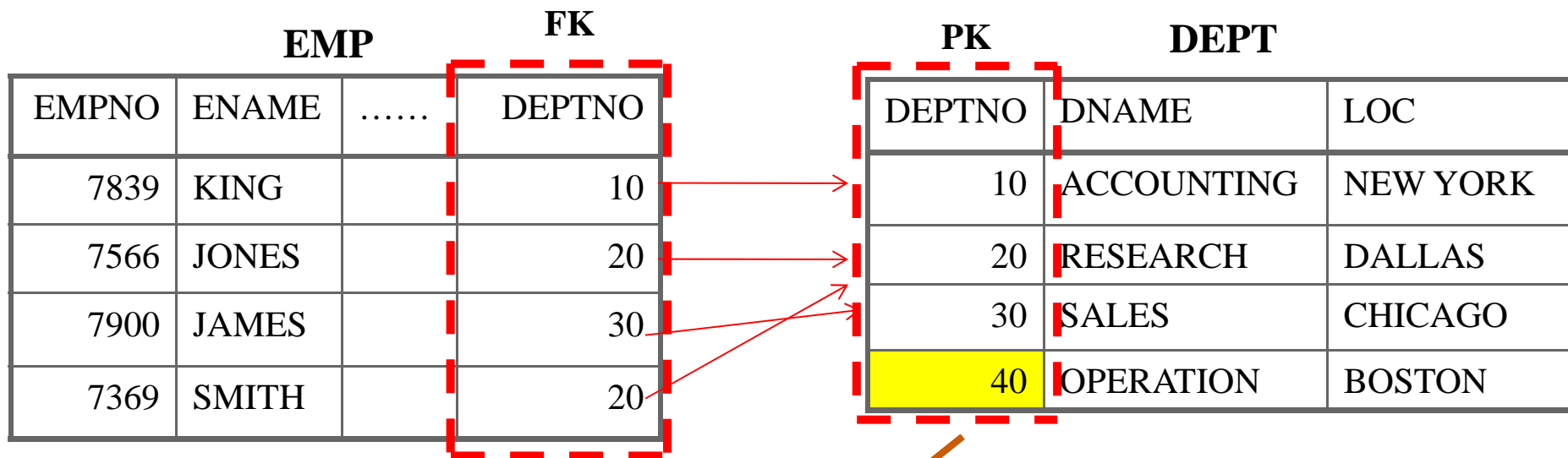
## □ 종류

- Left Outer Join: 왼쪽의 모든 튜플은 결과 테이블에 나타남
- Right Outer Join: 오른쪽의 모든 튜플은 결과 테이블에 나타남
- Full Outer Join: 양쪽 모두 결과 테이블에 참여

## □ 표현 방법

- NULL이 올 수 있는 쪽 조건에 (+)를 붙인다.

## Outer Join



EMPNO	ENA	SELECT * FROM EMP, DEPT				
7839	KING	WHERE EMP.DEPTNO (+) = DEPT.DEPTNO				
7566	JONES		20	20	RESEARCH	DALLAS
7900	JAMES		30	30	SALES	CHICAGO
7369	SMITH		20	20	RESEARCH	DALLAS
7499	ALLEN		30	30	SALES	CHICAGO
				40	OPERATION	BOSTON

## Self Join

- 자기자신과 Join
- Alias를 사용할 수 밖에 없음

```
SELECT * FROM EMP E1, EMP E2
WHERE E1.EMPNO = E2.MGR
```

PK		EMP		FK	
EMPNO	ENAME	MGR	.....		
7839	KING				
7566	JONES	7839			
7900	JAMES	7698			
7369	SMITH	7902			
7499	ALLEN	7698			

EMPNO	ENAME	MGR	.....	EMPNO	ENAME
7566	JONES	7839		7839	KING
7900	JAMES	7698		7698	BLAKE
7369	SMITH	7902		7902	FORD
7499	ALLEN	7698		7698	BLAKE

### ❑ From절에서 바로 **Join**을 명시적으로 정의

```
SELECT table1.column, table2.column  
FROM table1  
[CROSS JOIN table2] |  
[NATURAL JOIN table2] |  
[JOIN table2 USING (column_name)] |  
[JOIN table2  
ON(table1.column_name = table2.column_name)] |  
[LEFT|RIGHT|FULL OUTER JOIN table2  
ON (table1.column name = table2.column name)];
```

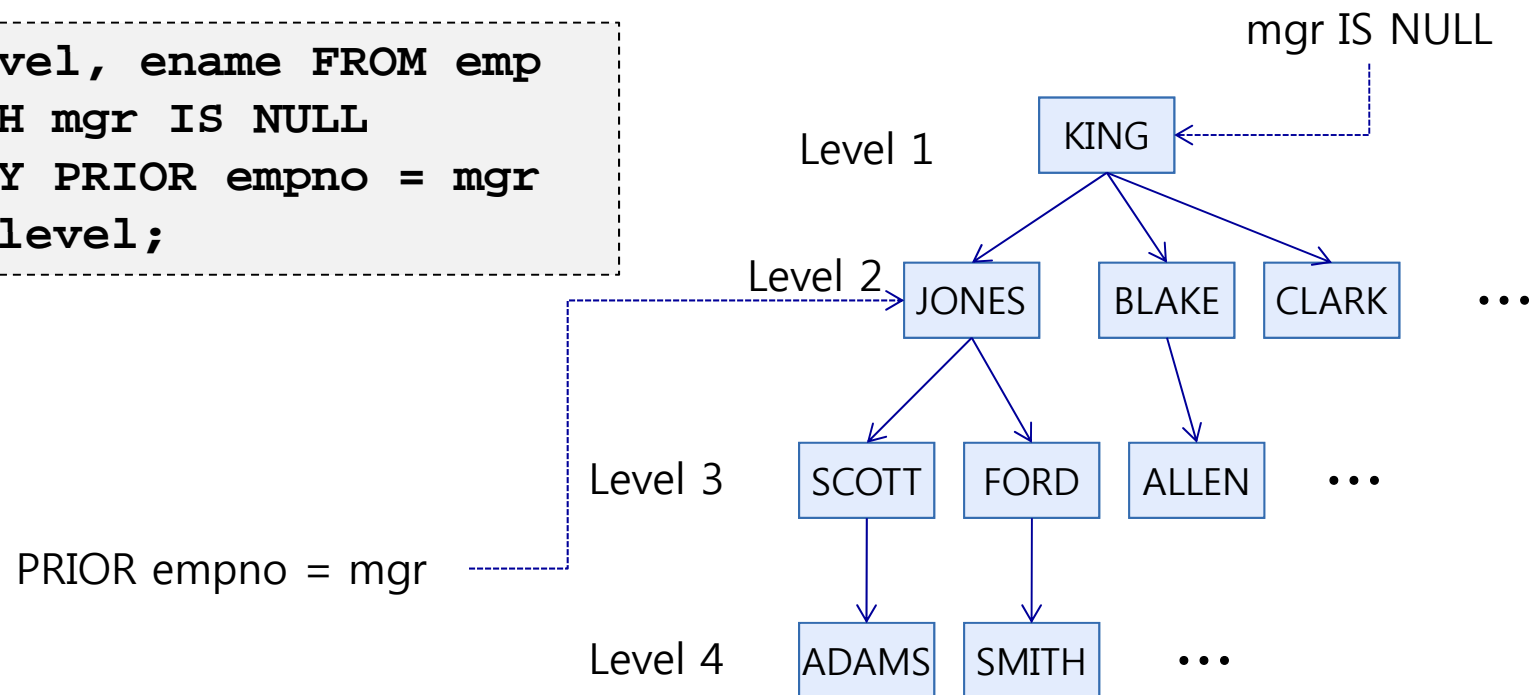
### ❑ 예

- SELECT \* FROM emp **NATURAL JOIN** dept;
- SELECT \* FROM emp **JOIN** dept **USING** (deptno);
- SELECT \* FROM emp **JOIN** dept **ON** emp.deptno = dept.deptno;
- SELECT \* FROM emp **RIGHT OUTER JOIN** dept **ON** (emp.deptno = dept.deptno);

## Hierarchical Query (ORACLE)

- ❑ 트리 형태 구조를 추출하기 위한 질의
- ❑ **START WITH (ROOT조건), CONNECT BY PRIOR (연결조건)**
- ❑ **LEVEL:** 트리의 레벨을 나타내는 **Pseudo Column**

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



---

# GROUP & AGGREGATION

## Aggregate Function (집계함수)

---

❑ 여러행으로부터 하나의 결과값을 반환

❑ 종류

- AVG
- COUNT
  - COUNT(\*): number of rows in table (NULL도 count된다)
  - COUNT(expr): non-null value (NULL은 빠진다)
  - COUNT(DISTINCT expr): distinct non-null
- MAX
- MIN
- SUM
- STDDEV
- VARIANCE



# Aggregate Function

```
SELECT sal FROM emp;
```

SAL
800
1600
1250
2975
1250
2850
2450
3000
5000
1500
1100
950
3000
1300

```
SELECT AVG(sal) FROM emp;
```

AVG(SAL)
2073.21429

## 일반적인 오류

---

### ❑ 부서별 평균 연봉?

```
SELECT deptno, AVG(sal) FROM emp;
```

### ❑ 주의

- 집계함수의 결과는 한 row만 남게 된다.
- deptno는 하나의 row에 표현될 수 없다.
- 부서별과 같은 내용이 필요할 때는 **Group by**절 사용

## GROUP BY

```
SELECT deptno, sal  
FROM emp  
ORDER BY deptno;
```

DEPTNO	SAL
10	2450
10	5000
10	1300
20	2975
20	3000
20	1100
20	800
20	3000
30	1250
30	1500
30	1600
30	950
30	2850
30	1250

```
SELECT deptno, AVG(sal)  
FROM emp  
GROUP BY deptno  
ORDER BY deptno;
```

DEPTNO	AVG(SAL)
10	2916.66667
20	2175
30	1566.66667

## 일반적인 오류

---

### ❑ 부서별 월급에서 부서명도 출력?

```
SELECT deptno, dname, AVG(sal)
FROM emp
GROUP BY deptno
ORDER BY deptno;
```

- 비록 부서번호에 따라 부서명은 하나로 결정될 수 있지만, dname은 grouping에 참여하지 않았으므로 하나의 row로 aggregate될 수 있다고 볼 수 없음

### ❑ 주의

- SELECT 의 Col 리스트에는 Group by에 참여한 필드나 aggregate 함수만 올 수 있다.
- Group by 이후에는 Group by에 참여한 필드나 aggregate 함수만 남아있는 셈
  - HAVING, ORDER BY 도 마찬가지

## HAVING 절

---

❑ **Aggregation** 결과에 대해 다시 **condition**을 검사할 때

❑ 일반적인 오류

- 평균 월급이 2000 이상인 부서는?

```
SELECT deptno, AVG(sal)
FROM emp
WHERE AVG(sal) > 2000
GROUP BY deptno;
```

❑ 주의

- WHERE 절은 Aggregation 이전, HAVING 절은 Aggregation 이후의 filtering
- Having절에는 Group by에 참여한 컬럼이나 Aggregate 함수만 사용가능

## 단일 SQL 문 실행 순서

---



## 단일 SQL 작성법

---

- ① 최종 출력될 정보에 따라 원하는 컬럼 **SELECT** 절에 추가
- ② 원하는 정보를 가진 테이블들을 **FROM** 절에 추가
- ③ **WHERE**절에 알맞은 **Join** 조건 추가
- ④ **WHERE**절에 알맞은 검색 조건 추가
- ⑤ 필요에 따라 **GROUP BY, HAVING** 등을 통해 **Grouping**
- ⑥ 정렬 조건 **ORDER BY**에 추가

## ROLLUP & CUBE

- ❑ ROLLUP (A, B): group by (A, B) & group by (A) & ALL
- ❑ CUBE(A, B): group by (A,B) & group by (A) & group by (B) & ALL
- ❑ GROUPING(expr): expr에 의해 그룹핑 된 경우만 1반환

```
SELECT loc, yr, sum(sales)
FROM salesdata
GROUP BY ROLLUP(loc, yr);
```

```
SELECT loc, yr, sum(sales)
FROM salesdata
GROUP BY CUBE(loc, yr);
```

**ROLLUP(지역,연도)**

지역별	년도별	SUM(매출)
서울	2000	100
서울	2001	200
서울	2002	300
서울		600
부산	2000	100
부산	2001	150
부산	2002	150
부산		400
		1000

**CUBE(지역,연도)**

지역별	년도별	SUM(매출)
서울	2000	100
서울	2001	200
서울	2002	300
서울		600
부산	2000	100
부산	2001	150
부산	2002	150
부산		400
	2000	200
	2001	350
	2002	450
		1000



---

# SUBQUERY

## Subquery

---

□ 하나의 **SQL** 질의문 속에 다른 **SQL** 질의문이 포함되어 있는 형태

□ 예) '**SCOTT**'보다 월급이 많은 사람의 이름은?

– 월급이 많은 사람의 이름?

- SELECT ename FROM emp WHERE sal > ???

– '**SCOTT**'의 월급?

- SELECT sal FROM emp WHERE ename='SCOTT'

```
SELECT ename
FROM emp
WHERE sal > ( SELECT sal
               FROM emp
               WHERE ename = 'SCOTT' )
```

## Single-Row Subquery

---

- ❑ Subquery의 결과가 한 **ROW**인 경우
- ❑ Single-Row Operator 사용해야 함: = , > , >=, < , <=, <>

```
SELECT  ename, sal, deptno
FROM emp
WHERE ename = (SELECT MIN(ename) FROM emp);
```

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

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

## Multi-Row Query

❑ Subquery의 결과가 둘 이상의 Row

❑ Multi-Row에 대한 연산을 사용해야 함: ANY, ALL, IN, EXIST...

```
SELECT  ename, sal, deptno
FROM    emp
WHERE   ename = (SELECT  MIN(ename)
                  FROM    emp GROUP BY deptno);
```



```
SELECT  ename, sal, deptno
FROM    emp
WHERE   ename IN (SELECT  MIN(ename)
                  FROM    emp GROUP BY deptno);
```



```
SELECT  ename, sal, deptno
FROM    emp
WHERE   ename = ANY (SELECT  MIN(ename)
                     FROM    emp GROUP BY deptno);
```



## Correlated Query

---

❑ **Outer Query와 Inner Query**가 서로 연관되어 있음

❑ 해석방법

- Outer query의 한 Row를 얻는다.
- 해당Row를 가지고 Inner Query를 계산한다.
- 계산 결과를 이용 Outer query의 WHERE절을 evaluate
- 결과가 참이면 해당 Row를 결과에 포함시킨다.

```
SELECT ename, sal, deptno
FROM    emp outer
WHERE   sal > (SELECT AVG(sal)
               FROM    emp
               WHERE   deptno = outer.deptno);
```

## 예제

---

- 각 부서별로 최고급여를 받는 사원을 출력하시오.

```
SELECT deptno, empno, ename, sal
FROM emp
WHERE (deptno,sal) IN (SELECT deptno, max(sal)
                      FROM emp GROUP BY deptno);
```

```
SELECT e.deptno, e.empno, e.ename, e.sal
FROM emp e,(SELECT s.deptno, max(s.sal)  msal
             FROM emp s GROUP BY deptno) m
WHERE e.deptno = m.deptno AND e.sal = m.msal;
```

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

## Top-K Query (ORACLE)

---

❑ **ROWNUM**: 질의의 결과에 가상으로 부여되는 Oracle의 Pseudo Column

❑ **Top-K Query**: 조건을 만족하는 상위 k개의 결과를 빨리 얻기

- 81년도에 입사한 사람 중 월급이 가장 많은 3명은 누구인가?

```
SELECT rownum, ename, sal
FROM emp
WHERE hiredate like '81%' AND rownum < 4
ORDER BY sal DESC;
```

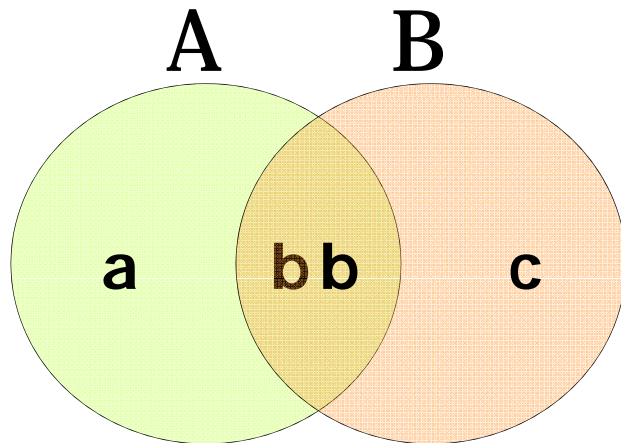


```
SELECT rownum, ename, sal
FROM (SELECT *
      FROM emp
      WHERE hiredate like '81%'
      ORDER BY sal DESC)
WHERE rownum < 4;
```



# SET Operator

- ❑ 두 질의의 결과를 가지고 집합 연산
- ❑ **UNION, UNION ALL, INTERSECT, MINUS**



- $A \cup B = \{a, b, c\}$
- $A \cup ALL B = \{a, b, b, c\}$
- $A \cap B = \{b\}$
- $A - B = \{a\}$

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



## RANK 관련 함수

```
SELECT sal, ename,  
       RANK() OVER (ORDER BY sal DESC) AS rank,  
       DENSE_RANK() OVER (ORDER BY sal DESC) AS dense_rank,  
       ROW_NUMBER() OVER (ORDER BY sal DESC) AS row_number,  
       rownum AS "rownum"  
FROM emp;
```



SAL	ENAME	RANK	DENSE_RANK	ROW_NUMBER	rownum
5000	KING	1	1	1	9
3000	FORD	2	2	2	13
3000	SCOTT	2	2	3	8
2975	JONES	4	3	4	4
2850	BLAKE	5	4	5	6
2450	CLARK	6	5	6	7
1600	ALLEN	7	6	7	2
1500	TURNER	8	7	8	10
1300	MILLER	9	8	9	14
1250	WARD	10	9	10	3
1250	MARTIN	10	9	11	5
1100	ADAMS	12	10	12	11
950	JAMES	13	11	13	12
800	SMITH	14	12	14	1

# - DDL & DCL

---

## 목 차

---

1. DDL
  1. Create Table, Drop Table , Truncate Table, Alter Table
  2. Data Type
  3. Constraint (NOT NULL, DEFAULT, CHECK, REFERENCE)
2. DCL

---

## Data Definition Language

# DDL

- ❑ **CREATE TABLE:** 테이블 생성
- ❑ **ALTER TABLE:** 테이블 관련 변경
- ❑ **DROP TABLE:** 테이블 삭제
- ❑ **RENAME:** 이름 변경
- ❑ **TRUNCATE:** 테이블의 모든 데이터 삭제
- ❑ **COMMENT:** 테이블에 설명 추가

## 테이블 생성

❑ CREATE TABLE문 이용

❑ 테이블이름, 컬럼 이름, 데이터 타입 등 정의

```
CREATE TABLE book (  
    bookno NUMBER(5),  
    title VARCHAR2(50),  
    author VARCHAR2(10),  
    pubdate DATE  
);
```



bookno	title	author	pubdate
1	토지	박경리	2005-03-12
2	슬램덩크	다케이코	2006-04-05
...	...	...	...

## Subquery를 이용한 테이블 생성

---

- ❑ Subquery의 결과와 동일한 테이블 생성됨
- ❑ 질의 결과 레코드들이 포함됨
- ❑ NOT NULL 제약 조건 만 상속됨

```
CREATE TABLE empSALES  
AS  
    SELECT * FROM emp  
    WHERE job = 'SALES';
```

# Naming Rules

---

## □ 테이블, 컬럼, ... 등의 이름 명명 규칙

- 문자로 시작
- 30자 이내
- A-Z, a-z, 0-9, \_, \$, #
- 같은 유저가 소유한 다른 Object의 이름과 겹치지 않아야함  
(다른 유저 소유의 object와는 같을 수도 있음)
- 오라클 예약어는 사용할 수 없음

### □ User Tables:

- Are a collection of tables created and maintained by the user
- Contain user information

### □ Data Dictionary:

- Is a collection of tables created and maintained by the Oracle Server
- Contain database information



## 기본 데이터 타입

Data type	Description
VARCHAR2(size)	가변길이 문자열 (최대 4000byte)
CHAR(size)	고정길이 문자열 (최대 2000byte)
NUMBER(p,s)	가변길이 숫자. 전체 p자리 중 소수점 이하 s자리 (p:38, s:-84~127, 21Byte) 자리수 지정 없으면 NUMBER(38) INT,
DATE	고정길이 날짜+시간, 7Byte

### □ 참고

- VARCHAR2와 CHAR의 차이점
- INT, FLOAT 등의 ANSI Type도 내부적으로 NUMBER(38)로 변환됨

Data type	Description
NCHAR(size)	national character set에 따라 결정되는 size 만큼의 고정길이 character data로 최대 2000byte까지 가능. 디폴트는 1 character.
NVARCHAR2 (size)	national character set에 따라 결정되는 size 만큼의 가변길이 character data로 최대 4000 byte까지 가능하며 반드시 길이를 정해 주어야 함.
LONG	가변 길이 character data로 최대 2 gigabyte까지 가능.
RAW (size)	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 만을 저장.
INTERVAL DAY TO SECOND	두 datetime 값의 차이를 DAY, HOUR, MINUTE, SECOND 까지 저장.

## ALTER TABLE

---

### □ 컬럼 추가

- ALTER TABLE book **ADD** (pubs VARCHAR2(50));

### □ 컬럼 수정

- ALTER TABLE book **MODIFY** (title VARCHAR2(100));

### □ 컬럼 삭제

- ALTER TABLE book **DROP** author;

### □ UNUSED 컬럼

- ALTER TABLE book **SET UNUSED** (author);  
ALTER TABLE book **DROP UNUSED COLUMNS**;

## 기타 테이블 관련 명령

---

### □ 테이블 삭제

- DROP TABLE book;

### □ 데이터 삭제

- TRUNCATE TABLE book;

### □ Comment

- COMMENT ON TABLE book IS 'this is comment';

### □ RENAME

- RENAME book TO article;

### □ 주의:

- ROLLBACK의 대상이 아님!

### □ Constraint

- Database 테이블 레벨에서 특정한 규칙을 설정해둠
- 예상치 못한 데이터의 손실이나 일관성을 어기는 데이터의 추가, 변경 등을 예방함

### □ 종류

- NOT NULL
- UNIQUE
- PRIMARY KEY
- FOREIGN KEY
- CHECK

## 제약조건 정의

---

### □ Syntax

- CREATE TABLE 테이블이름 (  
    컬럼이름 datatype [DEFAULT 기본값] [컬럼제약조건],  
    컬럼이름 datatype [DEFAULT 기본값] [컬럼제약조건],  
    ...  
    [테이블 제약조건] ...);
- 컬럼 제약조건: [CONSTRAINT 이름] constraint\_type
- 테이블제약조건: [CONSTRAINT 이름] constraint\_type(column,...)

### □ 주의

- 제약조건에 이름을 부여하지 않으면 Oracle이 Sys-Cn의 형태로 자동 부여

### □ NOT NULL

- NULL 값이 들어올 수 없음
- 컬럼형태로만 제약조건 정의할 수 있음 (테이블 제약조건 불가)

```
CREATE TABLE book (  
    bookno NUMBER(5) NOT NULL  
);
```

### □ UNIQUE

- 중복된 값을 허용하지 않음 (NULL은 들어올 수 있음)
- 복합 컬럼에 대해서도 정의 가능
- 자동적으로 인덱스 생성

```
CREATE TABLE book (  
    bookno NUMBER(5) CONSTRAINT c_emp_u UNIQUE  
);
```

## 제약조건 (2/3)

### □ PRIMARY KEY

- NOT NULL + UNIQUE (인덱스 자동 생성)
- 테이블 당 하나만 나올 수 있음
- 복합 컬럼에 대해서 정의 가능 (순서 중요)

```
CREATE TABLE book (  
    ssn1 NUMBER(6),  
    ssn2 NUMBER(7),  
    PRIMARY KEY (ssn1,ssn2)  
);
```

### □ CHECK

- 임의의 조건 검사. 조건식이 참이어야 변경 가능
- 동일 테이블의 컬럼만 이용 가능

```
CREATE TABLE book (  
    rate NUMBER CHECK (rate IN (1,2,3,4,5))  
);
```



### □ FOREIGN KEY

- 참조 무결성 제약
- 일반적으로 REFERENCE 테이블의 PK를 참조
- REFERENCE 테이블에 없는 값은 삽입 불가
- REFERENCE 테이블의 레코드 삭제 시 동작
  - **ON DELETE CASCADE:** 해당하는 FK를 가진 참조행도 삭제
  - **ON DELETE SET NULL:** 해당하는 FK를 NULL로 바꿈

```
CREATE TABLE book (  
    ...  
    author_id NUMBER(10),  
    CONSTRAINT c_book_fk FOREIGN KEY (author_id)  
    REFERENCE author(id)  
    ON DELETE SET NULL  
);
```

## ADD / DROP CONSTRAINTS

---

### □ 제약조건 추가

- **ALTER TABLE 테이블이름 ADD CONSTRAINT ...**
- NOT NULL은 추가 못함

```
ALTER TABLE emp ADD CONSTRAINT emp_mgr_fk  
FOREIGN KEY (mgr) REFERENCES emp(empno);
```

### □ 제약조건 삭제

- **ALTER TABLE 테이블이름 DROP CONSTRAINT 제약조건이름**
- PRIMARY KEY의 경우 FK 조건이 걸린 경우에는 CASCADE로 삭제해야함

```
ALTER TABLE book DROP CONSTRAINT c_emp_u;  
ALTER TABLE dept DROP PRIMARY KEY CASCADE;
```

## ENABLE/DISABLE CONSTRAINTS

---

### □ 제약조건 비활성화

- 제약조건 검사를 중지함
- CASCADE를 사용하여 의존되어 있는 다른 조건을 함께 중지시킬 수 있음
- 대규모 데이터 변경 등의 속도를 빠르게 함
- UNIQUE, PRIMARY KEY의 경우 인덱스 제거됨

```
ALTER TABLE emp DISABLE CONSTRAINT c_emp_pk CASCADE;
```

### □ 제약조건 활성화

- 중지되어 있던 제약조건 검사를 활성화함
- UNIQUE, PRIMARY KEY의 경우 인덱스 자동 생성

```
ALTER TABLE emp ENABLE CONSTRAINT c_emp_pk CASCADE;
```

## CASCADE CONSTRAINT

---

- ❑ 제약조건이 걸려있는 테이블이나 컬럼은 삭제시 에러 발생
- ❑ 컬럼이나 테이블 **DROP**할 때 관련 제약조건도 함께 삭제 할 때

```
ALTER TABLE emp DROP (deptno) CASCADE CONSTRAINT;
```

```
DROP TABLE emp CASCADE CONSTRAINT;
```

## □ Oracle이 관리하는 모든 정보를 저장하는 카탈로그

### □ 내용

- 모든 스키마 객체 정보, 스키마 객체의 공간 정보, 컬럼의 기본값, 제약조건 정보, 오라클 사용자 정보, 권한 및 롤 정보, 기타 데이터베이스 정보 ...

### □ Base-Table과 View로 구성됨

- VIEW의 Prefix
  - USER: 로그인한 사용자 레벨
  - ALL: 모든 사용자 정보
  - DBA: 관리자

### □ SYS scheme에 속함

### □ 주의

- DICTIONARY의 테이블이나 컬럼 이름은 모두 대문자 사용!

## Dictionary 예

---

### ❑ 모든 Dictionary 정보

```
SELECT * FROM DICTIONARY
```

### ❑ 사용자 스키마 객체 확인 (테이블)

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

### ❑ 제약조건 확인 (EMP 테이블의)

```
SELECT constraint_name, constraint_type, search_condition  
FROM user_constraints  
WHERE table_name = 'EMP';
```

### ❑ 제약조건 컬럼 확인

```
SELECT constraint_name, column_name  
FROM user_cons_columns  
WHERE table_name = 'EMP';
```

## Dictionary 예(2/2)

---

### ❑ 모든 사용자 확인

```
SELECT username, default_tablespace,  
temporary_tablespace FROM DBA_USERS;
```

---

## **Data Control Language**

# DCL



### □ Syntax

- 사용자 생성: `CREATE USER user IDENTIFIED BY passwd;`
- 비밀번호 변경: `ALTER USER user IDENTIFIED BY passwd;`
- 사용자 삭제: `DROP USER user [CASCADE];`

### □ 주의

- 일반적으로 DBA의 일
- 사용자를 생성하려면 `CREATE USER` 권한 필요
- 생성된 사용자가 Login하려면 `CREATE SESSION` 권한 필요
- 일반적으로 `CONNECT`, `RESOURCE`의 `ROLE`을 부여하면 일반사용자 역할을 할 수 있음

## 사용자 정보 확인

---

### □ 관련 Dictionary

- USER\_USERS: 현재 사용자 관련 정보
- ALL\_USERS: DB의 모든 사용자 정보
- DBA\_USERS: DB의 모든 사용자의 상세 정보 (DBA만 사용 가능)

```
SELECT * FROM USER_USERS;
```

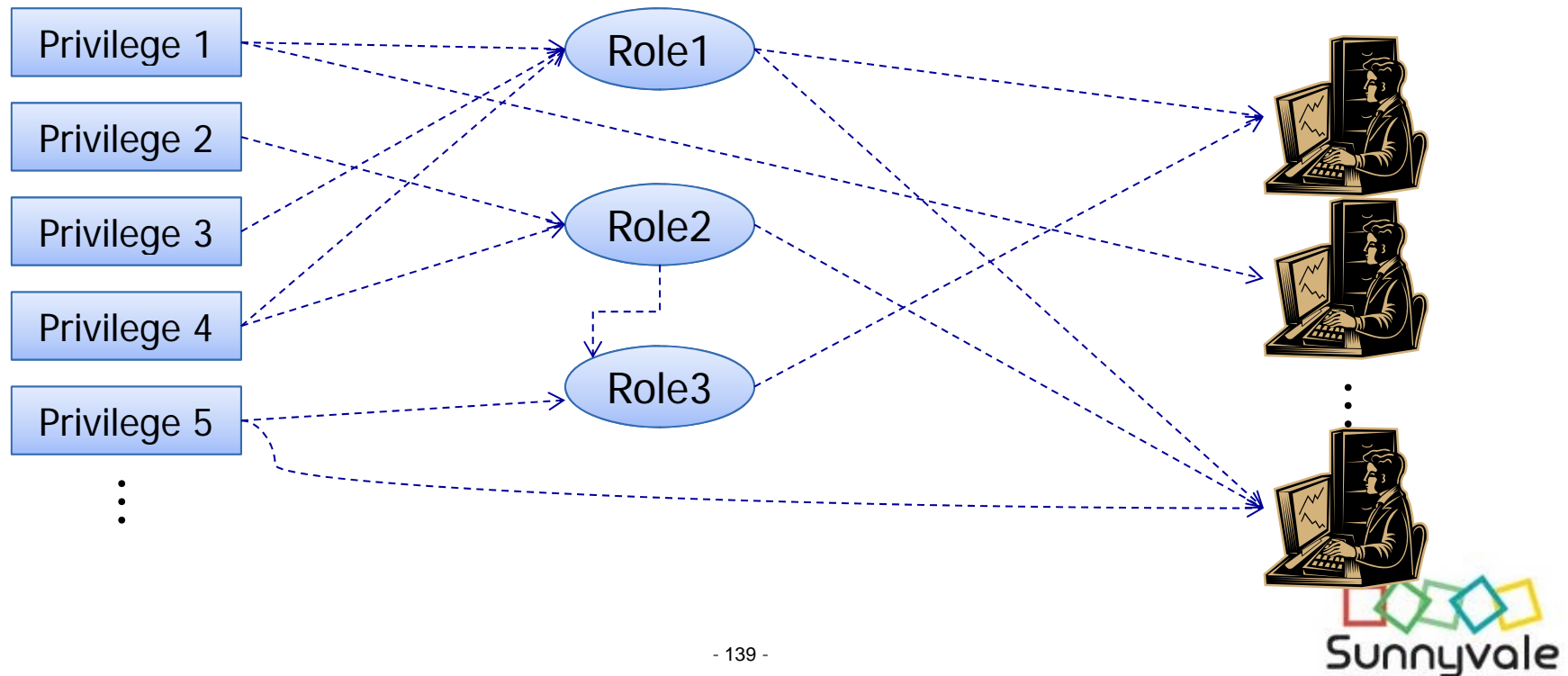
# 권한 (Privilege)과 룰(Role)

## □ 권한:

- 사용자가 특정 SQL문을 실행하거나 특정 정보에 접근할 수 있는 권리
- 종류: 시스템 권한(80여개) & 스키마 객체 권한

## □ 룰:

- 권한을 쉽게 관리하기 위하여 특정한 종류별로 묶어놓은 그룹



## GRANT / REVOKE

---

### ❑ 권한/롤을 부여하거나 회수

### ❑ 시스템 권한: 관리자로 수행 (**ADMIN OPTION/GRANT ANY PRIVILEGE** 권한)

```
GRANT create session TO user1;
```

```
REVOKE create session FROM user1;
```

### ❑ 스키마객체 권한

```
GRANT select ON emp TO user1;
```

```
REVOKE select ON emp FROM user1;
```

### ❑ WITH GRANT OPTION

- 해당 권한을 받은 사람이 다시 제3자에게 권한을 부여할 수 있도록 하는 옵션
- 권한을 REVOKE 하면 그 사람이 준 권한들도 함께 회수됨

```
GRANT select ON emp TO user2  
WITH GRANT OPTION;
```

## ROLE

---

### ❑ Role을 생성한 후 Role에 Privilege를 Grant하여 Role관리

- 주로 DBA작업

```
CREATE ROLE reviewer;  
GRANT select any table TO reviewer;  
GRANT create session, resource TO reviewer;
```

### ❑ 특정 Role을 사용자에게 Grant / Revoke

```
GRANT reviewer TO user3;
```

### □ 관련 Dictionary

- ROLE\_SYS\_PRIVS: System privileges granted to roles
- ROLE\_TAB\_PRIVS: Table privileges granted to roles
- USER\_ROLE\_PRIVS: Roles accessible by the user
- USER\_TAB\_PRIVS\_MADE: Object privileges granted on the user's object
- USER\_TAB\_PRIVS\_RECD: Object privileges granted to the user
- USER\_COL\_PRIVS\_MADE: Object privileges granted on the columns of the user's object
- USER\_COL\_PRIVS\_RECD: Object privileges granted to the user on specific columns
- USER\_SYS\_PRIVS: Lists system privileges granted to the user

```
SELECT * FROM USER_ROLE_PRIVS;  
SELECT * FROM ROLE_SYS_PRIVS;
```

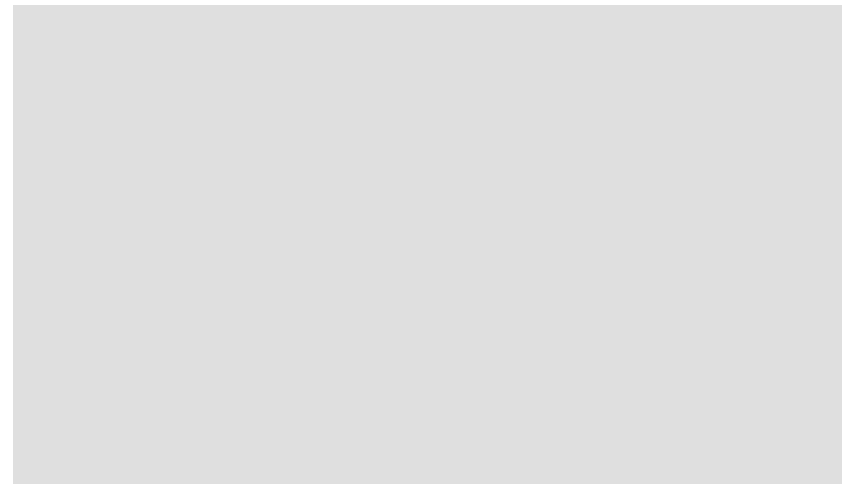
# - DML

---

## 목 차

---

1. INSERT, DELETE, UPDATE
2. Transaction Control



## □종류

- Add new row(s)
  - INSERT INTO 테이블이름 [(컬럼리스트)] VALUES ( 값리스트 );
- Modify existing rows
  - UPDATE 테이블이름 SET 변경내용 [WHERE 조건];
- Remove existing rows
  - DELETE FROM 테이블이름 [WHERE 조건];

## □트랜잭션의 대상

- 트랜잭션은 DML의 집합으로 이루어짐.



# INSERT

---

- ❑ 묵시적 방법: 컬럼 이름. 순서 지정하지 않음. 테이블 생성시 정의한 순서에 따라 값 지정

```
INSERT INTO dept  
VALUES (777, 'MARKETING', NULL);
```

- ❑ 명시적 방법: 컬럼 이름 명시적 사용. 지정되지 않은 컬럼 **NULL** 자동 입력

```
INSERT INTO dept(dname, deptno)  
VALUES ('MARKETING', 777);
```

- ❑ **Subquery** 이용: 타 테이블로부터 데이터 복사 (테이블은 이미 존재하여야 함)

```
INSERT INTO deptusa  
  SELECT deptno, dname  
  FROM dept  
  WHERE country = 'USA';
```

# UPDATE

---

## □ 조건을 만족하는 레코드를 변경

- 10번 부서원의 월급 100인상 & 수수료 0으로 변경

```
UPDATE emp
SET sal = sal + 100, comm = 0
WHERE deptno = 10;
```

## □ WHERE 절이 생략되면 모든 레코드에 적용

- 모든 직원의 월급 10%인상

```
UPDATE emp SET sal = sal * 1.1
```

## □ Subquery를 이용한 변경

- 담당업무가 'SCOTT'과 같은 사람들의 월급을 부서 최고액으로 변경

```
UPDATE emp SET sal = (SELECT MAX(sal) FROM emp)
WHERE job =
      (SELECT job FROM emp WHERE ename='SCOTT');
```

# DELETE

---

## □ 조건을 만족하는 레코드 삭제

- 이름이 'SCOTT'인 사원 삭제

```
DELETE FROM emp  
WHERE ename = 'SCOTT';
```

## □ 조건이 없으면 모든 레코드 삭제 (주의!)

- 모든 직원 정보 삭제

```
DELETE FROM emp;
```

## □ Subquery를 이용한 DELETE

- 'SALES'부서의 직원 모두 삭제

```
DELETE FROM emp  
WHERE deptno = (SELECT deptno FROM dept  
                WHERE dname = 'SALES');
```

## 참고

---

### ❑ 데이터 입력, 수정시 자주 사용되는 **Pseudo** 컬럼

- USER : Current user name.
- SYSDATE : Current date and time.
- ROWID : Location information of rows

```
INSERT INTO emp(eno, hiredate) VALUES (200, SYSDATE);
```

### ❑ **DEFAULT: default**값이 정의된 컬럼에 기본 값을 입력할 경우 사용할 수 있음

```
INSERT INTO book VALUES (200, 'Gems', DEFAULT);
```

### ❑ **DELETE** 와 **TRUNCATE**의 차이점

- Delete는 Rollback 가능 but 대량의 log 등을 유발하므로 Truncate보다 느림

### ❑ 모든 **DML**문은 **Integrity Constraint**를 어길 경우 에러 발생

---

# TRANSACTION

## □정의:

- DB에서 하나의 작업으로 처리되는 논리적 작업 단위
- DBMS의 Concurrency control과 Recovery에서 중요한 역할을 수행

## □ACID Property

- Atomicity: all or nothing. 하나의 단위로 처리되어야 함. (중간까지만 처리됨은 불가)
- Consistency: 데이터베이스의 일관성(무결성)을 깨지 않아야 함
- Isolation: 다른 transaction과 동시에 수행되더라도 독립적으로 영향을 받지 않아야 함
- Durability: 한번 수행 완료(commit)되면 영원히 반영되어 있어야 함 (시스템 crash에서라도)

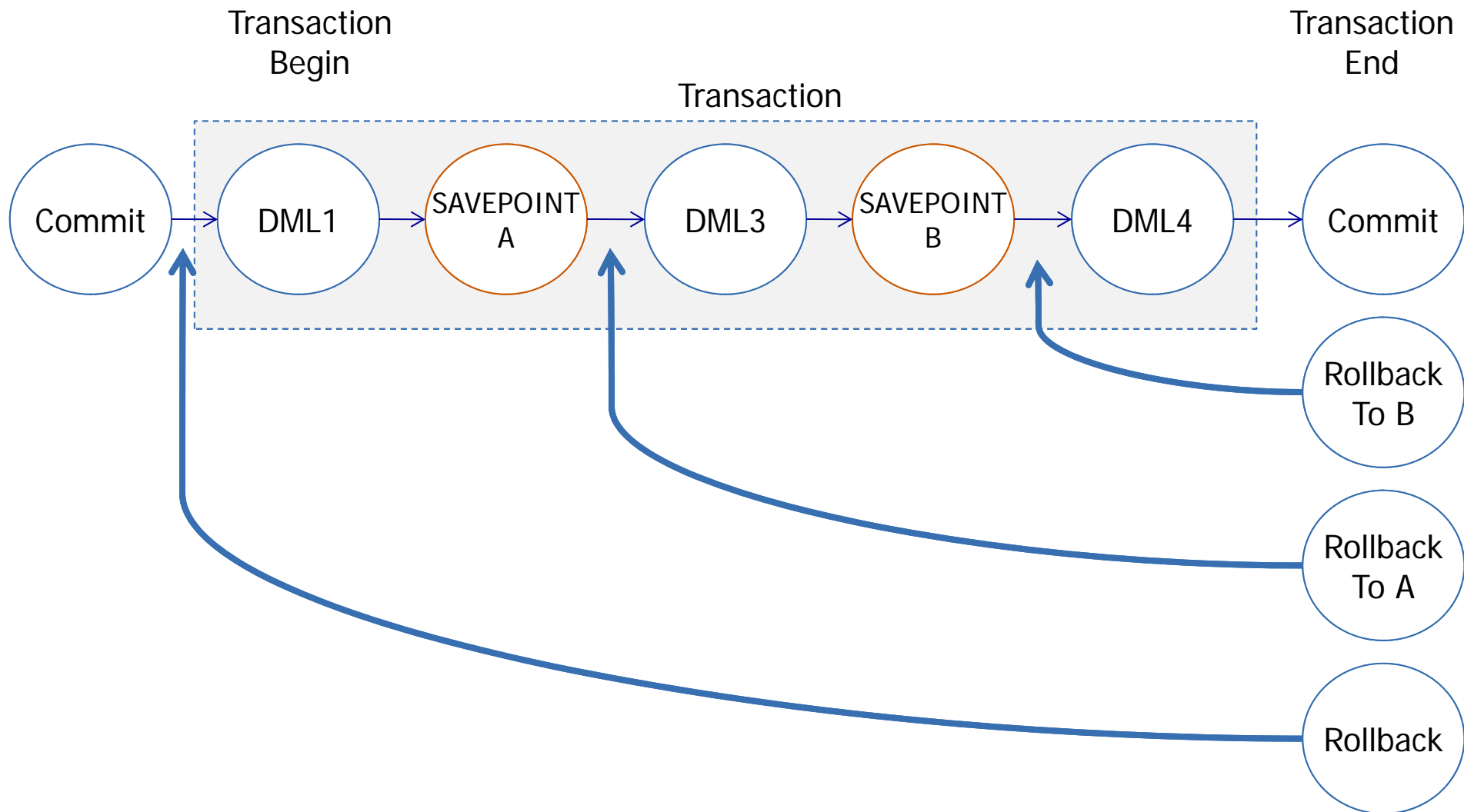
## □구성

- DML(INSERT, UPDATE, DELETE)의 집합
- DDL이나 DCL은 한 문장이 트랜잭션으로 처리됨

## □트랜잭션 정의

- 시작
  - 명시적인 트랜잭션 시작 명령 없음 (타 DBMS와의 차이점)
  - 첫 DML이 시작되면 트랜잭션 시작
- 명시적 종료: COMMIT / ROLLBACK
- 묵시적 종료
  - DDL, DCL 등이 수행될때 (automatic commit)
  - SQL\*PLUS등에서의 정상적 종료 (automatic commit)
  - 시스템 오류 (automatic rollback)

# Transaction Control





### ❑ Before Commit/Rollback

- 현재 사용자는 DML의 결과를 볼 수 있다.
- 다른 사용자는 현재 DML 결과를 볼 수 없다. (변경 이전 버전이 보임, Read Consistency)
- DML에 의해 변경된 모든 row는 Lock이 걸린다. (다른 트랜잭션에서 수정 불가)

### ❑ After Commit

- 변경이 영속적으로 DB에 반영됨. 이전 상태는 사라짐 (더 이상 Rollback 불가)
- 변경 결과를 모든 사용자가 볼 수 있음
- 모든 Lock이 풀림, 모든 Savepoint 사라짐

### ❑ After Rollback

- 모든 DML의 변경이 취소됨.
- 모든 Lock이 풀림, 모든 Savepoint 사라짐

# Lock in Oracle

## □ Lock

- Concurrent Transaction의 올바른 실행을 보장
- Oracle이 자동적으로 Lock을 관리

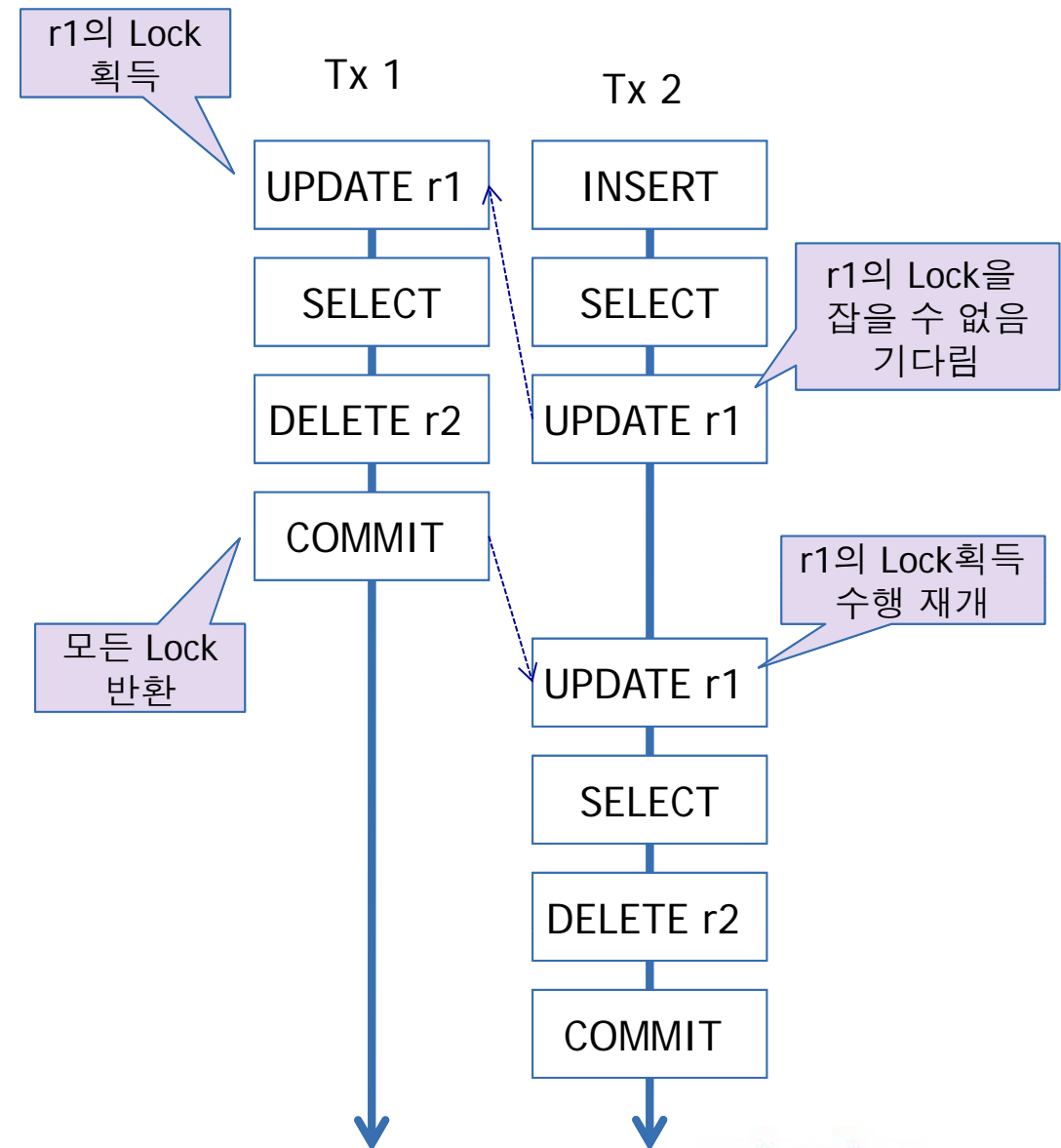
## □ Lock 종류

- DML: Table Share, Row Exclusive
  - 오라클이 자동적으로 Lock 획득 시도
- SELECT: No locks required
- DDL: Protects object definitions

## □ Locked 된 Row는

**COMMIT이나 ROLLBACK이**

**수행될 때까지 유지됨**



## □ Transaction Type

- SET TRANSACTION READ ONLY;
  - 읽기 전용 (Lock 필요없음)
  - Transaction-Level Read Consistency (트랜잭션 시작할때 봤던 값이 끝까지 나옴.)
- SET TRANSACTION READ WRITE; (default)
  - Statement-Level Read Consistency (Statement가 시작할때 값을 읽게 됨), WRITE 가능

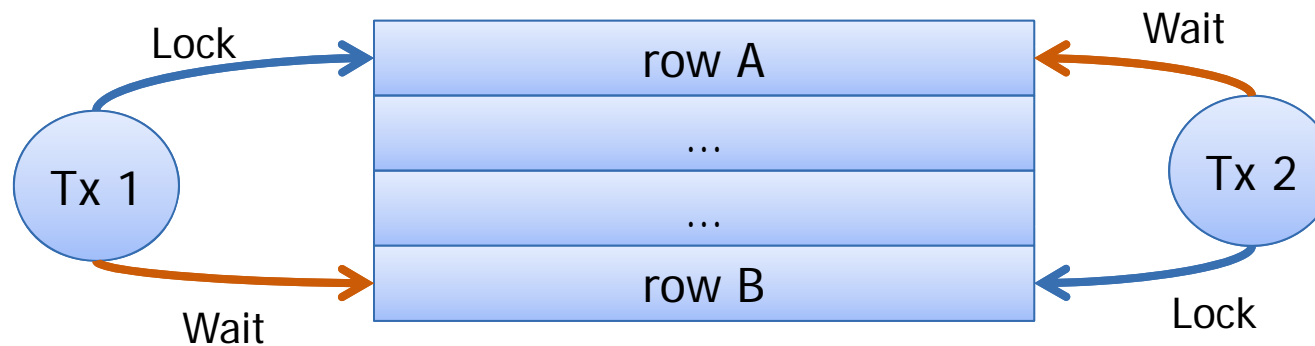
## □ Isolation Level

- SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
  - 나보다 늦게 시작한 트랜잭션이 먼저 수정 후 COMMIT한 row를 수정 하려면 ERROR 발생
  - 모든 충돌연산의 순서가 트랜잭션 순서와 동일해 짐.
- SET TRANSACTION ISOLATION LEVEL READ COMMITTED; (default)
  - 기본적인 lock기반 변경. commit된 데이터 변경 가능

# DEADLOCK

## ❑ Deadlock

- 둘 이상의 트랜잭션이 서로 상대방의 Lock을 순환 대기하여 어떤 트랜잭션도 더 이상 진행할 수 없는 상태
- Oracle이 주기적으로 자동 detect하여 에러를 돌려준다.



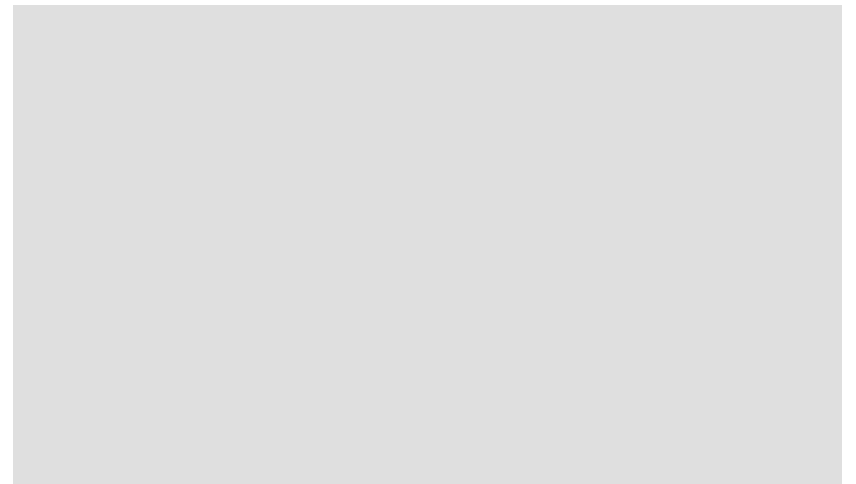
---

## **– Other DB Objects**

### **목 차**

---

1. Index
2. Sequence
3. Synonym
4. View
5. Trigger



---

# INDEX

## □ 장점

- 검색, 조인, 정렬...

## □ 단점

- 유지비용

## □ 언제?

- WHERE 절의 조건이나 Join에 자주 사용되는 컬럼
- 매우 큰 테이블에서 2~4%레코드만 선택될 때
- 값의 종류가 다양할 때 (예. 이름 vs. 성별?)
- 자주 변경되지 않을 때

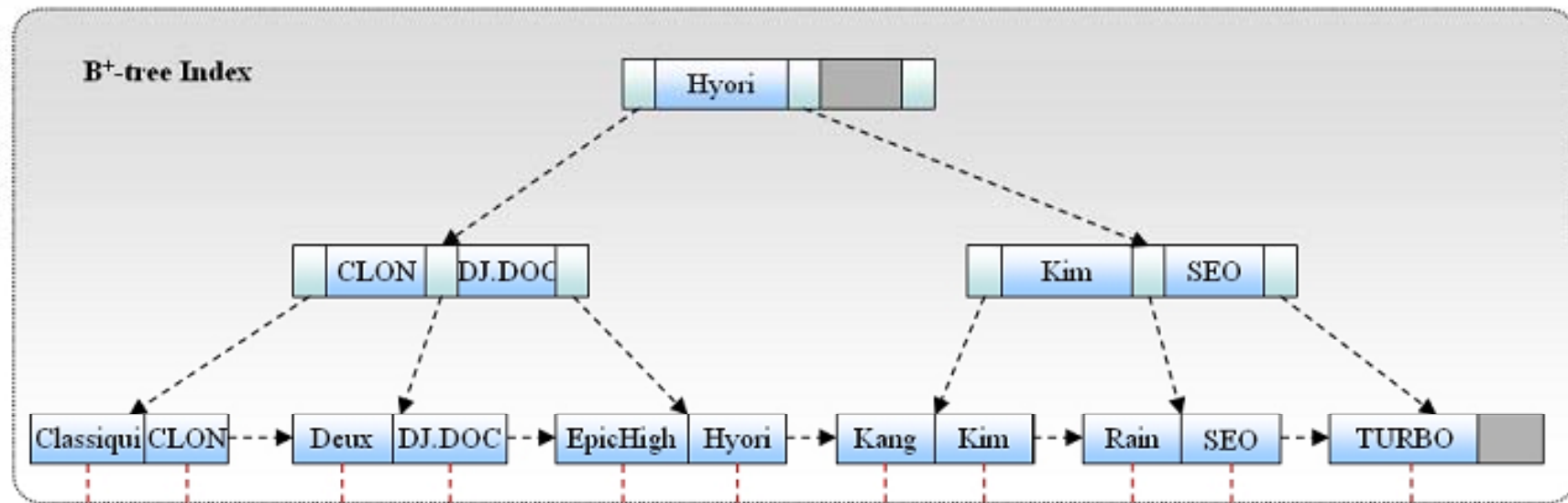
## □ Oracle Index

- B\*-Tree: 일반적 인덱스 (트리 기반)
- Bitmap 인덱스: 특수 용도 (OLAP 등에 사용)

## □ 구분

- Single 컬럼 vs. Composite 컬럼
  - Composite 컬럼: 둘 이상의 컬럼의 쌍에 인덱스가 걸림. 순서 중요!  
(이름, 부서)인덱스로 이름에 대한 검색은 처리할 수 있지만, 부서에 대한 검색은 처리할 수 없음
- Unique vs. Non-Unique
- Column Data vs. Function-Based
  - Function-Based: UPPER(name)과 같이 함수의 결과값으로 인덱스를 생성.  
WHERE UPPER(name)='AAA' 와 같은 질의를 빠르게 처리해줌
- Automatic-Created vs. User-Created
  - Automatic-Created: PK나 UNIQUE에 대해 index 자동 생성





Title	Artist	Genre	Year	Rate
I Know	SEO	2	1992	4.2
Fly High	EpicHigh	2	2005	3.5
Tonight	CLON	1	1998	2.7
10 Minutes	Hyori	3	2004	5.0
...	...	...	...	...
Color Your Soul	Classiqui	4	2005	4.1

**MUSIC Table**

# Syntax

---

- ❑ **CREATE [UNIQUE] INDEX *index\_name* ON *table\_name* (*column\_list...*);**

```
CREATE INDEX idx_emp_ename ON emp(ename);
```

- ❑ **DROP INDEX *index\_name*;**

```
DROP INDEX idx_emp_ename;
```

- ❑ **Dictionary**

- USER\_INDEXES
- USER\_IND\_COLUMNS

```
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';
```

## Hint Index

---

❑ 강제로 질의에서 특정 인덱스를 사용하도록 강요함

❑ **Oracle** 예

```
SELECT /* + index (employees EMP_EMAIL_UK) */ email  
FROM employees;
```

---

# SEQUENCE

### □ 자동 번호 생성기

### □ 용도

- Unique한 번호를 생성하고자 할 때 (PK 등을 위하여)
- 별도의 Concurrency나 Performance의 고려를 할 필요 없음

## Sequence 생성

---

```
CREATE SEQUENCE sequence_name  
  [INCREMENT BY n]  
  [START WITH n]  
  [{MAXVALUE n | NOMAXVALUE}]  
  [{MINVALUE n | NOMINVALUE}]  
  [{CYCLE | NOCYCLE}]  
  [{CACHE n | NOCACHE}];
```

- ❑ INCREMENT BY n: 번호 간격 (1)
- ❑ START WITH n: 시작번호(1)
- ❑ MAXVALUE n: 최대값 (최대 1027)
- ❑ MINVALUE n: 최소값 (최소 1)
- ❑ CACHE n: 오라클이 미리 생성해놓을 개수 (20)

# Sequence 사용하기

---

## ❑ Pseudo Columns

- CURRVAL: Sequence의 현재 값을 돌려준다.
- NEXTVAL: Sequence의 다음 값을 돌려주며, 현재값을 다음값으로 바꾼다. (increment)

## ❑ Example

```
INSERT INTO emp (empno, ename)  
VALUES (seq_empno.NEXTVAL, 'KIM');
```

```
SELECT seq_empno.CURRVAL FROM dual;
```

## Other Syntax

---

❑ **ALTER SEQUENCE** *sequence\_name*

...;

❑ **DROP SEQUENCE** *sequence\_name*;

❑ **Dictionary**

- USER\_SEQUENCES
- USER\_OBJECTS



---

# SYNONYM

□정의: 테이블, 뷰, 시퀀스 등의 이름의 **Alias**

□용도

- Alias
- 다른 사용자의 object를 바로 접근하기 위한 동의어

□Syntax

- CREATE [PUBLIC] SYNONYM *name* FOR *object*;
- DROP [PUBLIC] SYNONYM *name*;
- Dictionary
  - USER\_OBJECTS
  - USER\_SYNONYMS

## Example

**SQL> conn system**

hr 스키마에는 emp라는 object가 없다. (에러)

**SQL> SELECT \* FROM emp;**

scott.emp에 대한 synonym 생성

**SQL> CREATE SYNONYM emp FOR scott.emp;**

**SQL> SELECT \* FROM emp;**

synonym에 의해 scott.emp 직접 접근 가능  
물론, 적절한 권한 필요.

**SQL> DROP SYNONYM emp;**

---

# VIEW

□ 정의: a logical table based on one or tables or views.



□ 용도

- 보안강화
- 데이터 복잡성을 단순화
- 실제 테이블의 정의와 응용프로그램 분리
- 복잡한 질의 숨김

```
CREATE [OR REPLACE] [FORCE | NO FORCE] VIEW view_name  
    [(alias[,alias]...)]  
    AS Subquery  
    [WITH READ ONLY]  
    [WITH CHECK OPTION [CONSTRAINT constraint]];;
```

### □ 설명

- FORCE: Base 테이블이 없어도 무조건 Create
- WITH CHECK OPTION: View에 나타날 수 있는 행만 삽입/변경 가능

❑ **DROP VIEW *view\_name*;**

❑ **Dictionary**

- USER\_VIEWS
- USER\_OBJECTS

## View에 대한 변경

### □ 제약적으로 가능

- Simple View는 DML(DELETE, INSERT, UPDATE) 가능
- 제약조건을 만족시킬 수 없거나 원본 row를 유추할 수 없는 경우 등에는 변경이 불가
- View를 통한 변경이 원본 테이블에 반영이 가능한지 판단해야 함

```
CREATE OR REPLACE VIEW v1 AS  
  SELECT DISTINCT empno FROM emp;
```

```
CREATE OR REPLACE VIEW v1 AS  
  SELECT COUNT(empno) FROM emp;
```

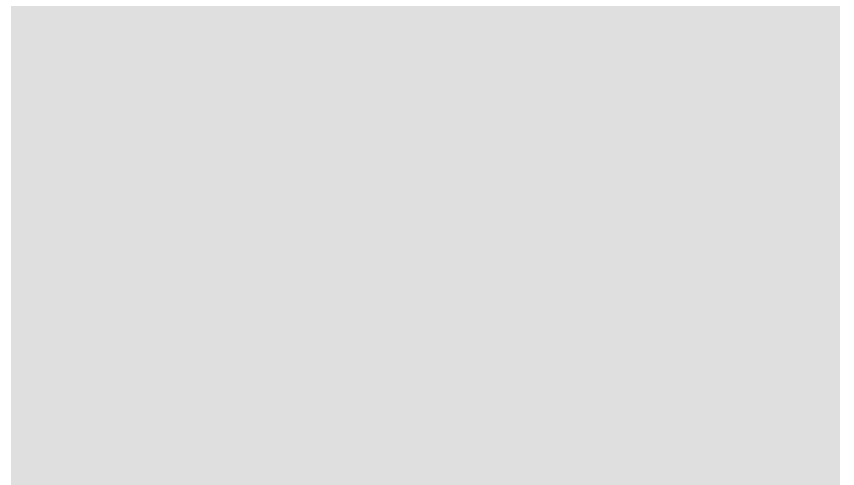
```
CREATE OR REPLACE VIEW v3 AS  
  SELECT * FROM emp, dept WHERE emp.deptno = dept.deptno;
```

```
CREATE OR REPLACE VIEW v4 AS  
  SELECT empno, sal * 2 AS dsal FROM emp;
```



---

**– JDBC**



---

# INTRODUCTION

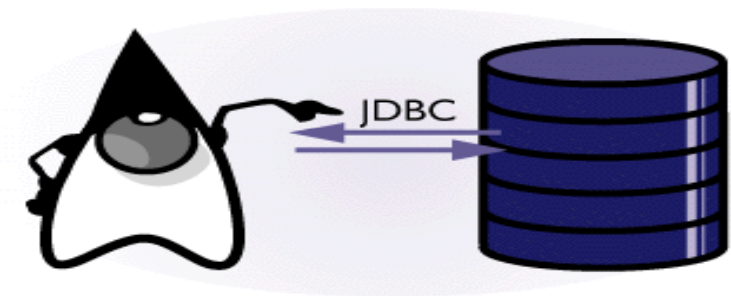
# JDBC

---

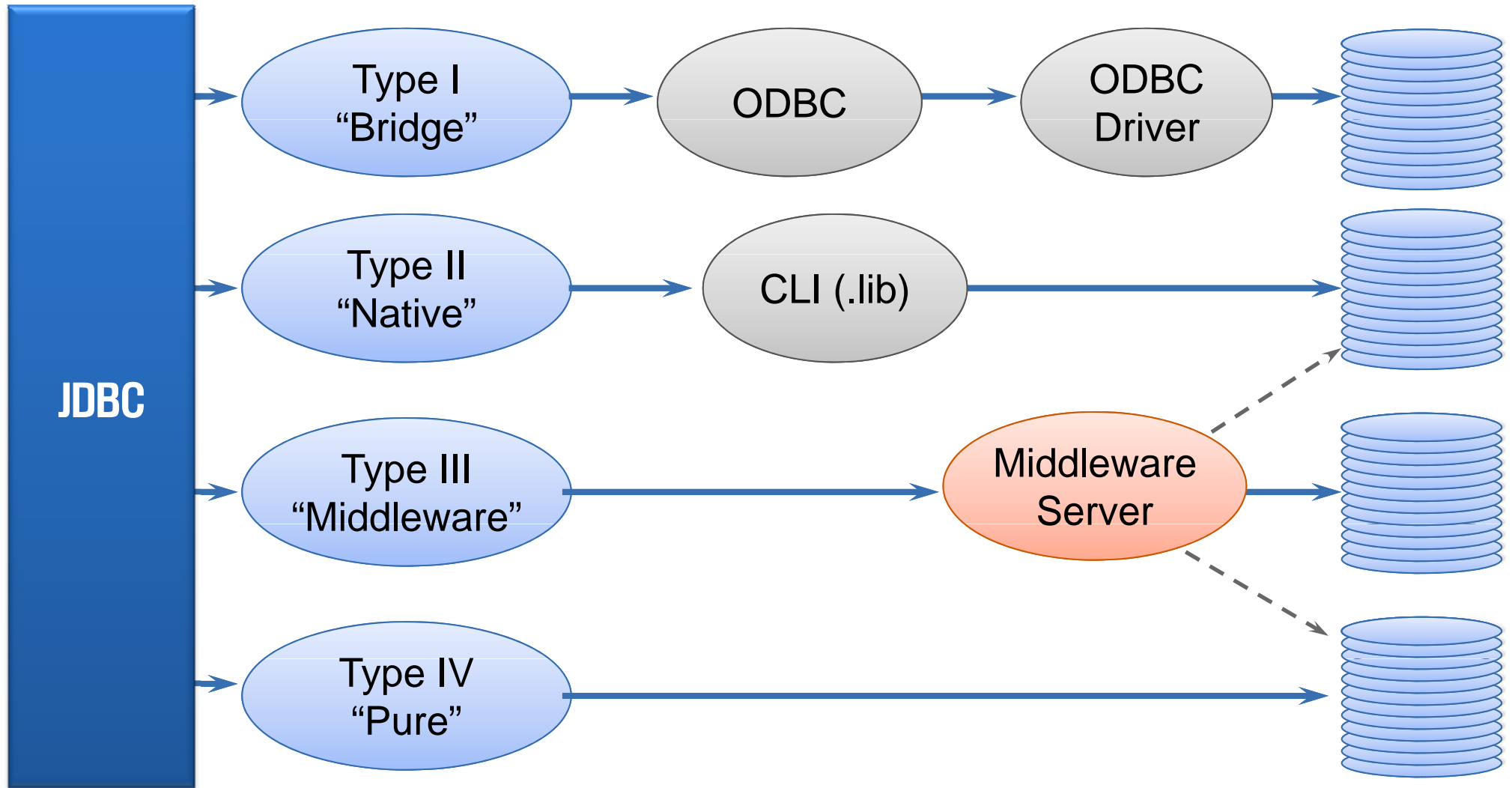
## □ JDBC(Java Database Connectivity)의 정의

- 자바를 이용한 데이터베이스 접속과 SQL 문장의 실행, 그리고 실행 결과로 얻어진 데이터의 핸들링을 제공하는 방법과 절차에 관한 규약
- 자바 프로그램내에서 SQL문을 실행하기 위한 자바 API
- SQL과 프로그래밍 언어의 통합 접근 중 한 형태

## □ 개발자를 위한 표준 인터페이스인 **JDBC API**와 데이터베이스 벤더, 또는 기타 써드파티에서 제공하는 드라이버(driver)



## JDBC Drivers 종류



### □ Client가 직접 Server에

#### 접속

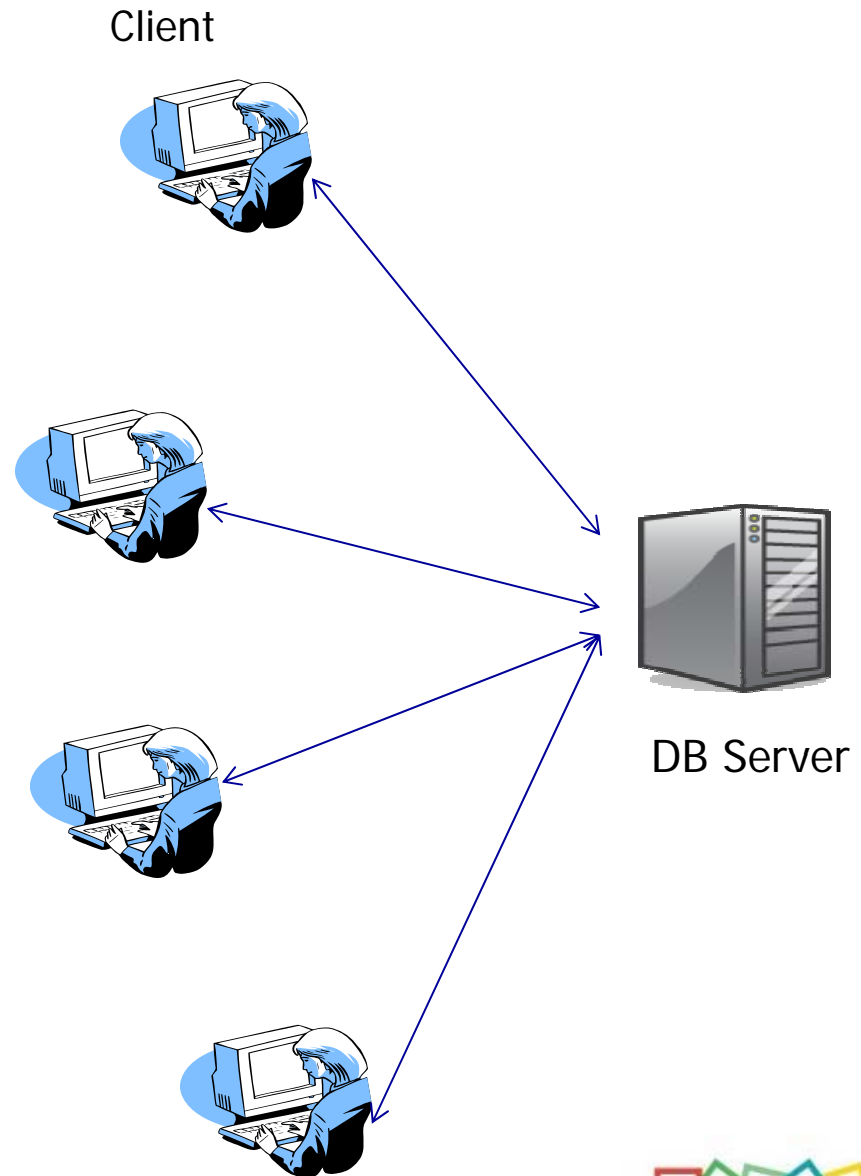
- 예) HTTP, email

### □ 장점

- 단순하다.
- 서버의 부담이 적다.

### □ 단점

- 클라이언트가 무겁다.
- 유지보수가 어렵다.
- 유연하지 못하다.
- 확장성 부족



## 3-Tier

### □ Client와 Server 사이에 Application 서버

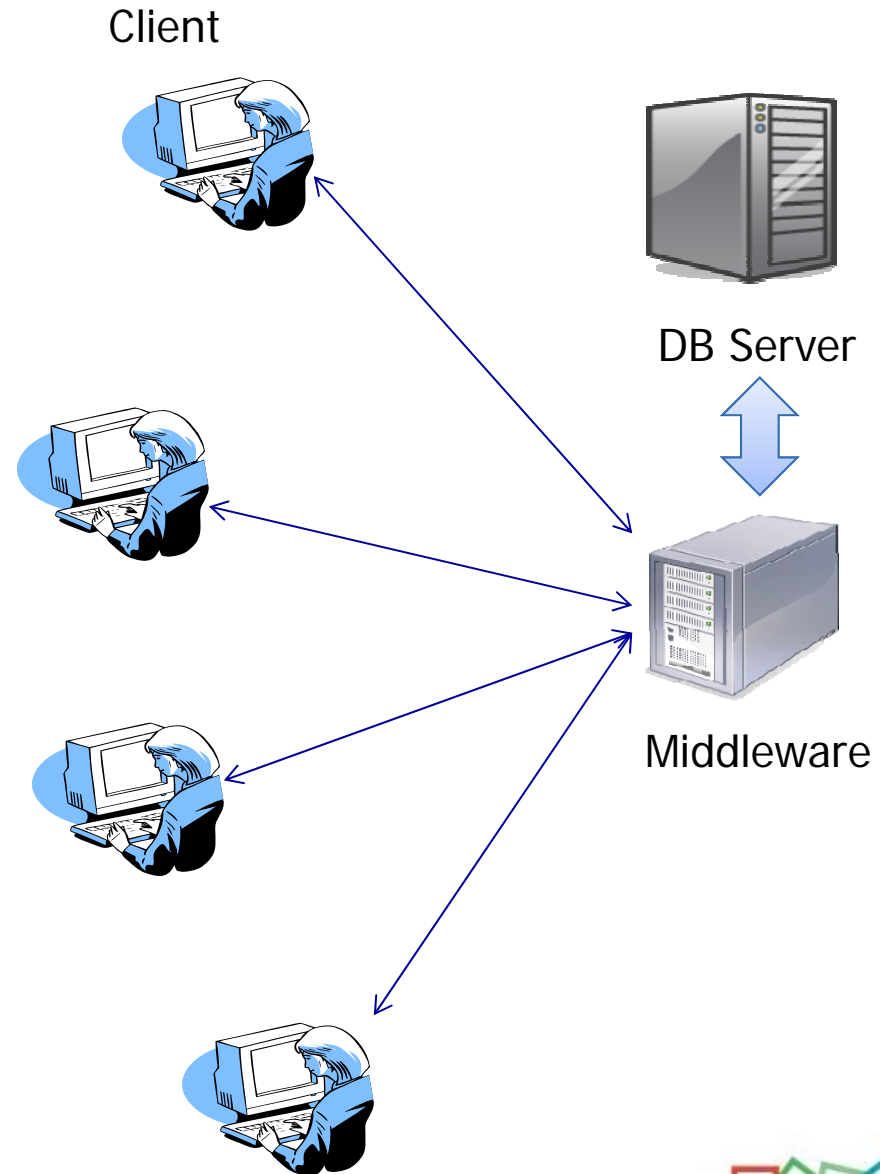
- 예) HTTP, email

### □ 장점

- 유연: 전체 영향 없이 부분 변경 가능
- DB 종류 변경 가능
- Presentation-Business logic-Data 분리
- 질의 Cache가능
- Proxy나 Firewall 등 가능
- Scalability: 확장성

### □ 단점

- 복잡하다. 비싸다. 성능(?)



## 환경 구성

---

### ■ JDK 설치

- <http://java.sun.com>

### ■ JSP 환경 구비

- <http://jakarta.apache.org/>

### ■ JDBC 드라이버 설치

- 오라클 JDBC 드라이버를 다운로드 받는다
  - [http://otn.oracle.com/software/tech/java/sqlj\\_jdbc/index.html](http://otn.oracle.com/software/tech/java/sqlj_jdbc/index.html)
  - ORACLE\_HOME/jdbc/lib/ 아래에도 있음
- JDBC 드라이버를 %JAVA\_HOME%\jre\lib\ext 에 복사
- 혹은 CLASSPATH에 추가

### ☐ Java API Reference

- <http://java.sun.com/javase/6/docs/api/>

### ☐ JDBC Tutorial

- <http://java.sun.com/docs/books/tutorial/jdbc/index.html>



---

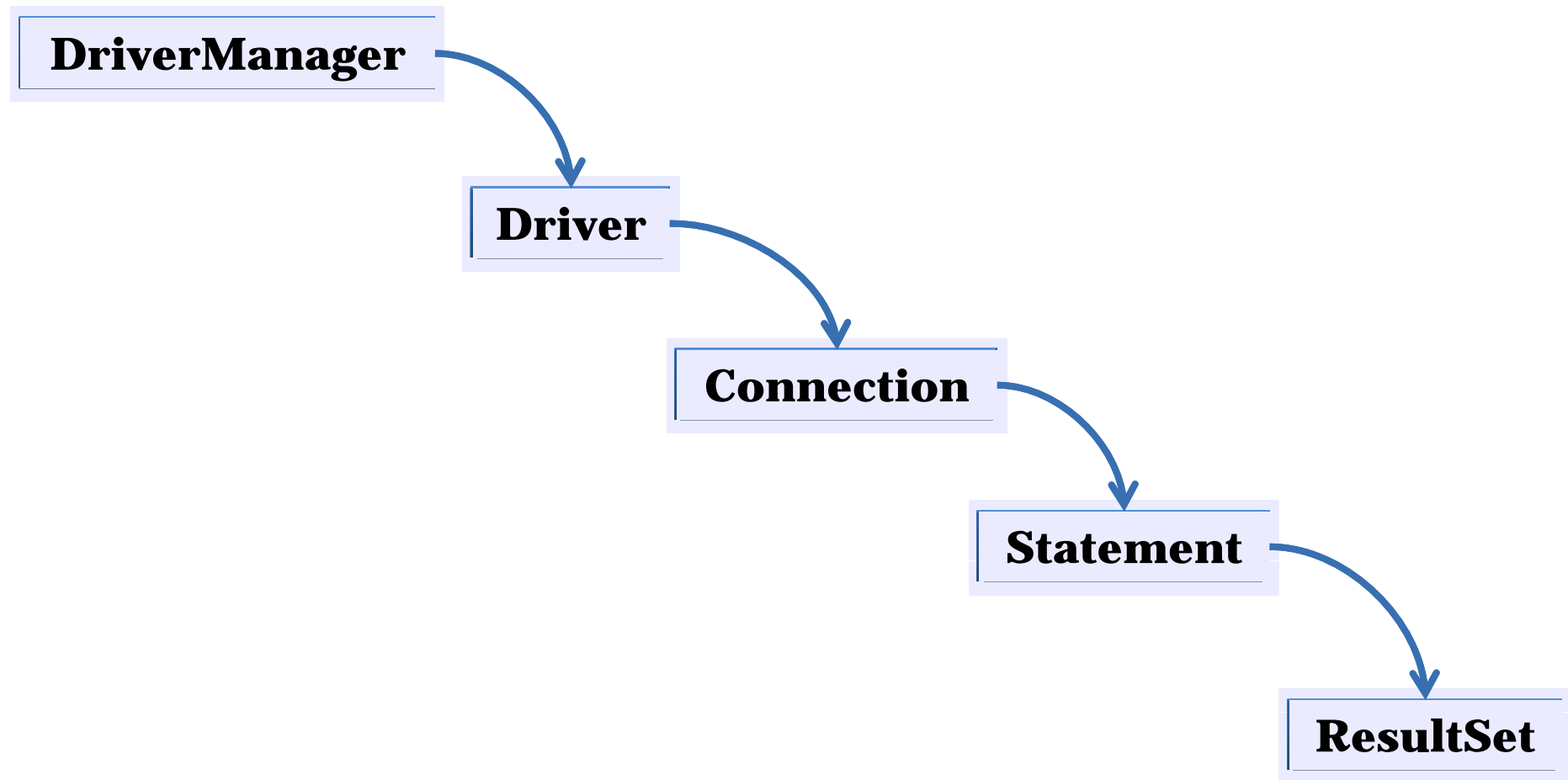
# JDBC 사용법

### □JDBC를 이용한 데이터베이스 연결 방법

- 1 단계 : `import java.sql.*;`
- 2 단계 : 드라이버를 로드 한다.
- 3 단계 : Connection 객체를 생성한다.
- 4 단계 : Statement 객체를 생성 및 질의 수행
- 5 단계 : SQL문에 결과물이 있다면 ResultSet 객체를 생성한다.
- 6 단계 : 모든 객체를 닫는다.

## JDBC Class Usage

---



### ■ **IMPORT**

- `import java.sql.*;`

### ■ **드라이버 로드**

- `Class.forName("oracle.jdbc.driver.OracleDriver");`

### ■ **Connection 얻기**

- `String dburl = "jdbc:oracle:thin:@localhost:1521:OID";`
- `Connection con = DriverManager.getConnection (dburl, ID, PWD);`

### ■ Statement 생성

- `Statement stmt = con.createStatement();`

### ■ 질의 수행

- `ResultSet rs = stmt.executeQuery("select ename from emp");`
  - any SQL  
SELECT
- 참고
  - `stmt.execute("query");`
  - `stmt.executeQuery("query");`
  - `stmt.executeUpdate("query");`
    - INSERT, UPDATE,  
DELETE

## 질의수행 예

### ❑ executeQuery(String sql) – select

```
Statement stmt = con.createStatement();  
ResultSet rs = stmt.executeQuery("SELECT * FROM emp");
```

### ❑ executeUpdate(String sql) –insert, update, delete

```
Statement stmt = con.createStatement();  
int updateCount = stmt.executeUpdate("INSERT INTO test VALUES (1)");
```

### ❑ execute(String sql) – SQL문 모를경우

```
Statement stmt = con.createStatement();  
if(stmt.execute(sql)){  
    ResultSet rs = stmt.getResultSet();  
    ...  
}else{  
    int rowCount = stmt.getUpdateCount();  
    ...  
}
```

### □ ResultSet으로 결과 받기

- `ResultSet rs = stmt.executeQuery("select ename from emp");`  
  `while (rs.next())`  
    `System.out.println(rs.getString("ename"));`

### □ Close

- `rs.close();`
- `stmt.close();`
- `con.close();`

## □ SQL문에 대한 결과를 얻는다.

- 이동함수: next, previous, first, last, beforeFirst, afterLast, relative, absolute
  - 기본: TYPE\_FORWARD\_ONLY & CONCUR\_READ\_ONLY  
(= next로만 이동 가능 & Read Only)
- 값 추출 함수: getXXX(숫자/이름) 데이터 타입에 따라 알맞게

	DEPTNO	DNAME	LOC
Cursor	10	ACCOUNTING	NEW YORK
	20	RESEARCH	DALLAS
	30	SALES	CHICAGO
	40	OPERATIONS	BOSTON

next()

getInt(1)    getString(2)    getString("LOC")



## 예제 1: JDBCTest.java

---

```
1  import java.sql.*;
2
3  class JDBCTest {
4      static{
5          try{
6              Class.forName("oracle.jdbc.driver.OracleDriver");
7          }catch(ClassNotFoundException cnfe){
8              cnfe.printStackTrace();
9          }
10     }
11
12     public static void main (String args []) throws SQLException {
13         String dburl = "jdbc:oracle:thin:@localhost:1521: ";
14         Connection con = DriverManager.getConnection (dburl, "scott" , "tiger");
15         Statement stmt = con.createStatement();
16
17         ResultSet rs = stmt.executeQuery("select ename from emp");
18
19         while (myResultSet.next())
20         {
21             System.out.println(rs.getString("ename"));
22         }
23
24         rs.close();
25         stmt.close();
26         con.close();
27     }
28 }
```

### □JDBC 2.0 이상

### □Scrollable RecordSet

- TYPE\_FORWARD\_ONLY (default) : FORWARD로만 이동가능. 속도 빠름
- TYPE\_SCROLL\_INSENSITIVE : 양방향이동 가능, 갱신 데이터 반영 안함
- TYPE\_SCROLL\_SENSITIVE : 양방향 이동 가능, 갱신된 데이터 반영

### □Updatable RecordSet

- CONCUR\_READ\_ONLY : 읽기 전용
- CONCUR\_UPDATABLE : 변경 가능

## Example

---

```
...  
stat = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,  
    ResultSet.CONCUR_UPDATABLE);  
ResultSet rs = stat.executeQuery("select empno, ename from emp");  
rs.moveToInsertRow();  
rs.updateInt(1,100);  
rs.updateString(2,"KIM");  
rs.insertRow();  
...  
rs.last();  
rs.updateString(3,"busan");  
rs.updateRow();  
...  
rs.absolute(1);  
rs.deleteRow();
```

### □컴파일 방법

- javac JDBCTest.java

### □실행 방법

- java JDBCTest

### □주의

- java JDK bin 디렉토리가 path에 잡혀 있어야 함
- CLASSPATH 설정 주의
- 클래스 이름 및 파일 이름의 대소문자 주의

## 주의:

---

# □에러시 해결방법

- 에러 메시지를 확인하자
- 대소문자가 틀렸나? (클래스 이름이나 파일 이름)
- JDBC는 제대로 찾고 있나?
- CLASSPATH나 PATH는 설정이 잘 되어 있나?
- 오라클은 제대로 켜져 있는가?
- SQL\*Plus로 접근이 가능한가?
- Listener 가 제대로 동작하는가? (telnet 127.0.0.1 1521 로 접속 시도해보자)
- OID나 ID/PWD는 올바른가?

## 과제 1.

❑ EMP 테이블의 결과를 다음과 같이 출력해보시오. (컬럼 이름 등은 문자열로 출력)

7369	SMITH	CLERK	7902	80/12/17	880		20
7499	ALLEN	SALESMAN	7698	81/02/20	1760	300	30
7521	WARD	SALESMAN	7698	81/02/22	1375	500	30
7566	JONES	MANAGER	7839	81/04/02	2975		20
7654	MARTIN	SALESMAN	7698	81/09/28	1375	1400	30
7698	BLAKE	MANAGER	7839	81/05/01	2850		30
7782	CLARK	MANAGER	7839	81/06/09	2450		10
7788	SCOTT	ANALYST	7566	87/04/19	3000		20
7839	KING	PRESIDENT		81/11/17	5000		10
7844	TURNER	SALESMAN	7698	81/09/08	1650	0	30
7876	ADAMS	CLERK	7788	87/05/23	1210		20
7900	JAMES	CLERK	7698	81/12/03	1045		30
7902	FORD	ANALYST	7566	81/12/03	3000		20
7934	MILLER	CLERK	7782	82/01/23	1430		10

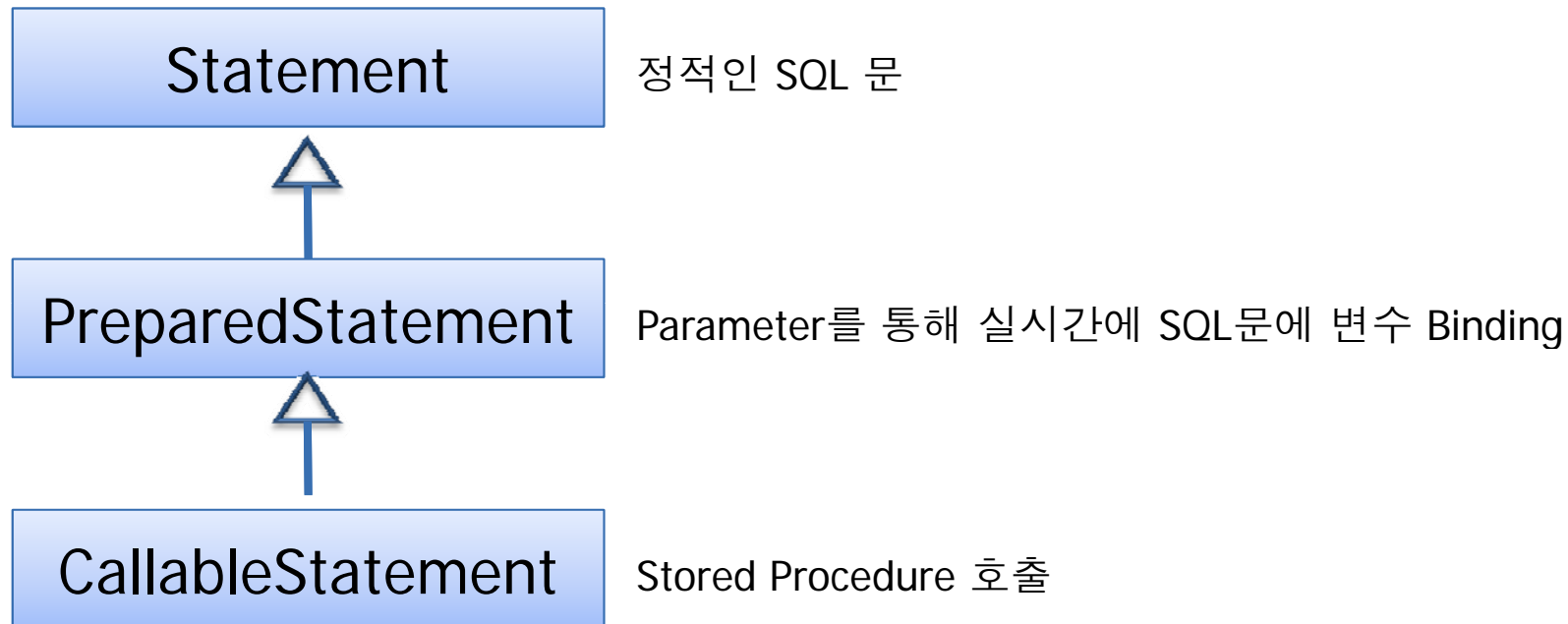
## ❑ 참고

- Java에서 Printf와 유사한 출력 (JDK1.5이상)
  - String.format → sprintf와 유사
  - System.out.format나 System.out.printf도 가능

## Statement 상속관계

### □ Statement를 상속받아서 각 클래스가 정의되어 있음

- 자식 클래스는 부모 클래스의 기능 + alpha



# PreparedStatement

---

## □ 수행 방법

- SQL 미리 준비 : 파싱, 실행 계획 저장
- 실시간에 Parameter 바인딩

## □ 바인딩

- 바인딩변수: ?
- setXXX 함수: 숫자 1부터

## □ 장점

- 효율성
- 보안 (SQL Injection)



## 예제2. PreparedStatement

---

```
...  
System.out.print("DEPT ID " + deptid + "=" );  
String sql = "SELECT dname, loc FROM dept WHERE deptno = ?";  
PreparedStatement pstmt = con.prepareStatement(sql);  
pstmt.setInt(1, deptid);  
ResultSet rs = pstmt.executeQuery();  
...
```

## 과제 2.

---

□사용자로부터 최소 봉급과 최대 봉급을 입력받아  
봉급이 이 범위 내에 속하는 사원의  
정보를 출력하는 프로그램을 작성하라.

# Transaction

---

## ❑ **con.setAutoCommit(true/false);**

- AutoCommit이 False이면 이하 DML은 Commit/Rollback등으로 Transaction 처리 하겠다는 의미

## ❑ **con.commit();**

## ❑ **con.rollback();**

# Savepoint

---

```
Statement stmt = conn.createStatement();
int rows = stmt.executeUpdate(
    "INSERT INTO TAB1 (COL1) VALUES ('AAA')");
...
// set savepoint
Savepoint svpt1 = conn.setSavepoint("SAVEPOINT_1");
rows = stmt.executeUpdate(
    "INSERT INTO TAB1 (COL1) VALUES ('AAA')");
...
conn.rollback(svpt1);
...
conn.commit();
```

### □ MetaData 추출

- 컬럼 이름, 컬럼의 데이터 타입, 널 허용여부 등...

```
rs = pstmt.executeQuery() ;
ResultSetMetaData rsmd = rs.getMetaData() ;
int numberColumn = rsmd.getColumnCount() ;
for(int i=0; i<numberColumn; i++) {
    String columnName = rsmd.getColumnName(i) ;
    String dbmsType = rsmd.getColumnTypeName(i) ;
    int isNull = rsmd.isNullable(i) ;
    ...
}
```

## □ DB Connection을 획득하고 반납하는 overhead 최소화

### □ 참고: DBCP

- JDBC 2.0부터 가능
- DataSource를 사용 JNDI의 lookup 이용
  - JNDI: Java Naming and Directory Interface
- DBCP (Database Connection Pool) API
  - Jakarta Commons 프로젝트의 일부
- 주로 톰캣 서버와 연동시 사용

