

# SQL 강의

## – Introduction to Database

### 목 차

---

1. 데이터베이스 소개
2. 관계형 모델

# 데이터베이스 개요

# 데이터베이스

## ❑ Database (DB)

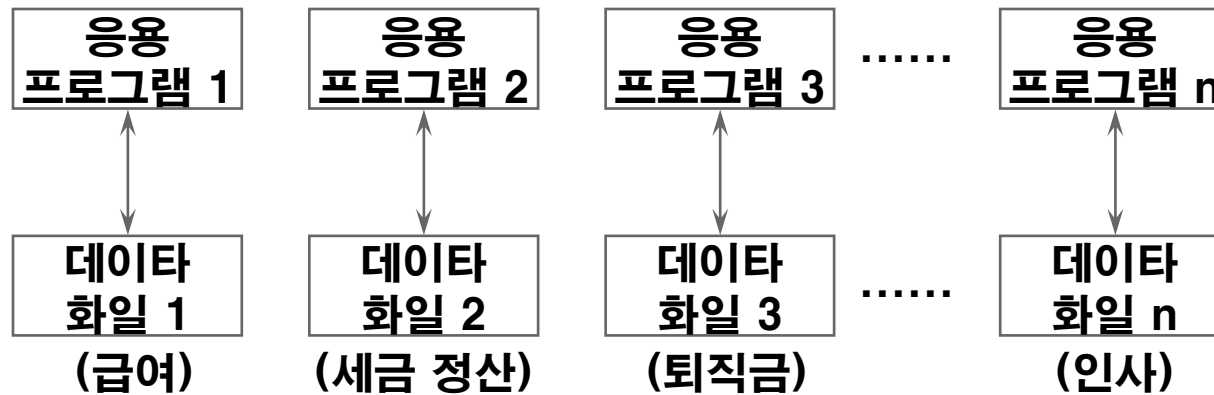
- 넓은 의미로 데이터가 모여 있는 것 자체를 의미
- 한 조직의 여러 응용 시스템들이 **공용(Shared)**하기 위해 **통합(Integrated)**, **저장(Stored)**한 데이터의 집합
- 전화번호부, 장부 등도 DB라고 할 수 있음



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

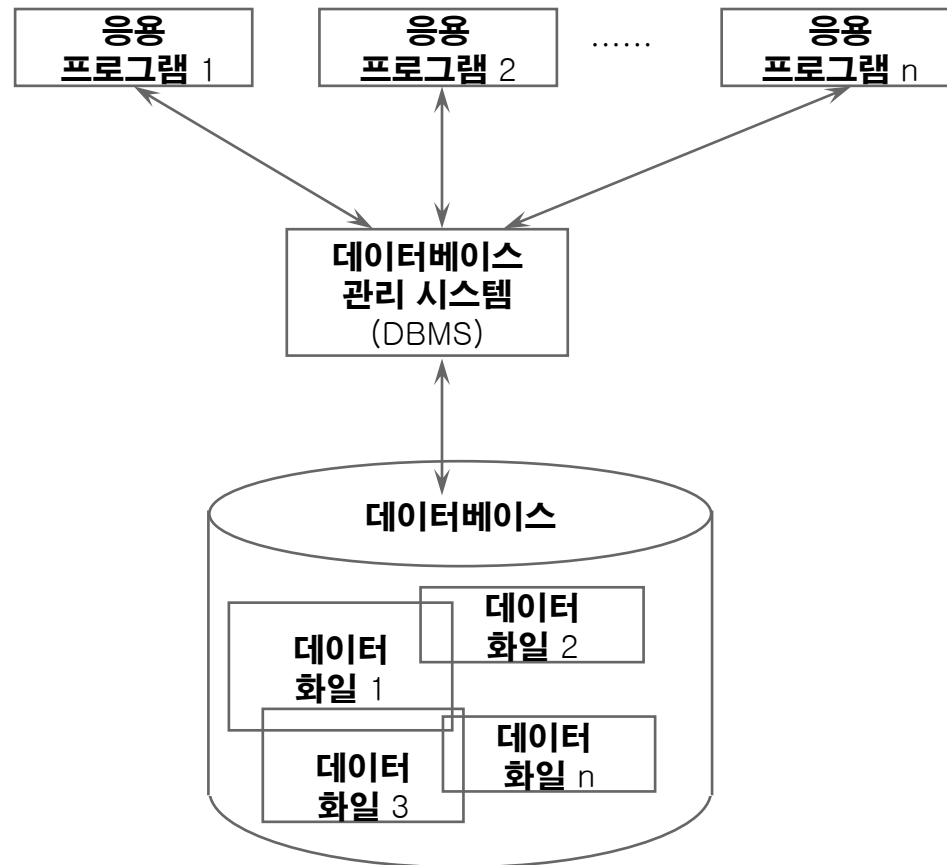
# DBMS의 목적

- ❑ 왜 **Database**관리를 위한 별도의 시스템이 필요한가?
- ❑ 각 응용 프로그램이 파일 시스템 등의 저장소를 이용하여 직접 **Database** 관리 프로그램을 짜면 더 효율적이고, 응용 프로그램에 적합하게 제작할 수 있지 않을까?



# DBMS

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



# DBMS의 장단점

---

## □ 장점

- 데이터 중복(redundancy)의 최소화
- 데이터의 공유(sharing)
- 일관성(consistency) 유지
- 무결성(integrity) 유지
- 보안(security) 보장

## □ 단점

- 비용: H/W, DBMS, 운영비, 교육비, 개발비

# DBMS 제품

---

## □ 상용 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

# 관계형 모델



# 관계형(Relational) 모델

## □ 핵심 용어

- 데이터베이스(Database): Table의 집합
- Table : Row의 집합
- Row : Table의 행
- Column : 테이블의 열
- Domain : 특정 Column이 가질 수 있는 값의 집합

**Table**

학번	이름	학년	학과
1	정성진	1	컴퓨터
2	박현진	2	수학
3	홍길동	4	물리
...	...	...	...

**Column**

**Row**

**Domain(학년) = {1, 2, 3, 4}**

The diagram illustrates a relational table with four columns: '학번' (Student ID), '이름' (Name), '학년' (Grade), and '학과' (Department). The first three rows contain specific data, and the fourth row shows ellipses to indicate more rows. Annotations include: 'Column' with arrows pointing to each of the four column headers; 'Row' with arrows pointing to the first and second data rows; and 'Domain(학년) = {1, 2, 3, 4}' with an arrow pointing to the '학년' column, specifically highlighting the values 1, 2, 3, and 4.

# 관계형 모델의 특징

---

## ❑ Column의 값은 원자값(atomic)이어야 함

- 값의 집합, Multivalue는 가질 수 없음

## ❑ Schema

- 데이터베이스의 구조를 정의

## ❑ Null

- 값이 지정되지 않았음을 의미하는 특별한 값
- 모든 Domain은 Null값을 포함.(기본적으로 모든 Column에 Null을 저장할 수 있다는 의미)

## ❑ Key

- 테이블에는 동일한 Row가 존재할 수 없음.
- 하나의 Row를 다른 Row와 구별하기 위한 키(Key)가 필요

# Primary Key (PK)

---

## ❑ Primary Key (PK) : 기본키

- Table에 Row를 구분하기 위하여 사용하는 기본 키
- 하나의 Column, 또는 Column의 집합(복합키) 가능
- Table 생성시 정의됨
- Table에는 동일한 PK를 지닌 Row가 존재할 수 없음

# Primary Key 예

## □ 학생

- 기본키: 학번

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)
- 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

- 다른 Table의 기본키를 참조하는 Column
- Table 간의 관계를 나타내기 위하여 사용
- Null 가능 (참조되지 않음 의미)

EMP							
PK		FK		FK			
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

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

# Integrity Constraint(무결성 제약)

---

## ❑ 개체 무결성 (Entity Integrity)

- 기본키의 값은 Null이 될 수 없다. 유일해야 한다.

## ❑ 참조 무결성 (Referential Integrity)

- 외래키의 값은 참조된 릴레이션의 기본키 값이거나 Null이다.

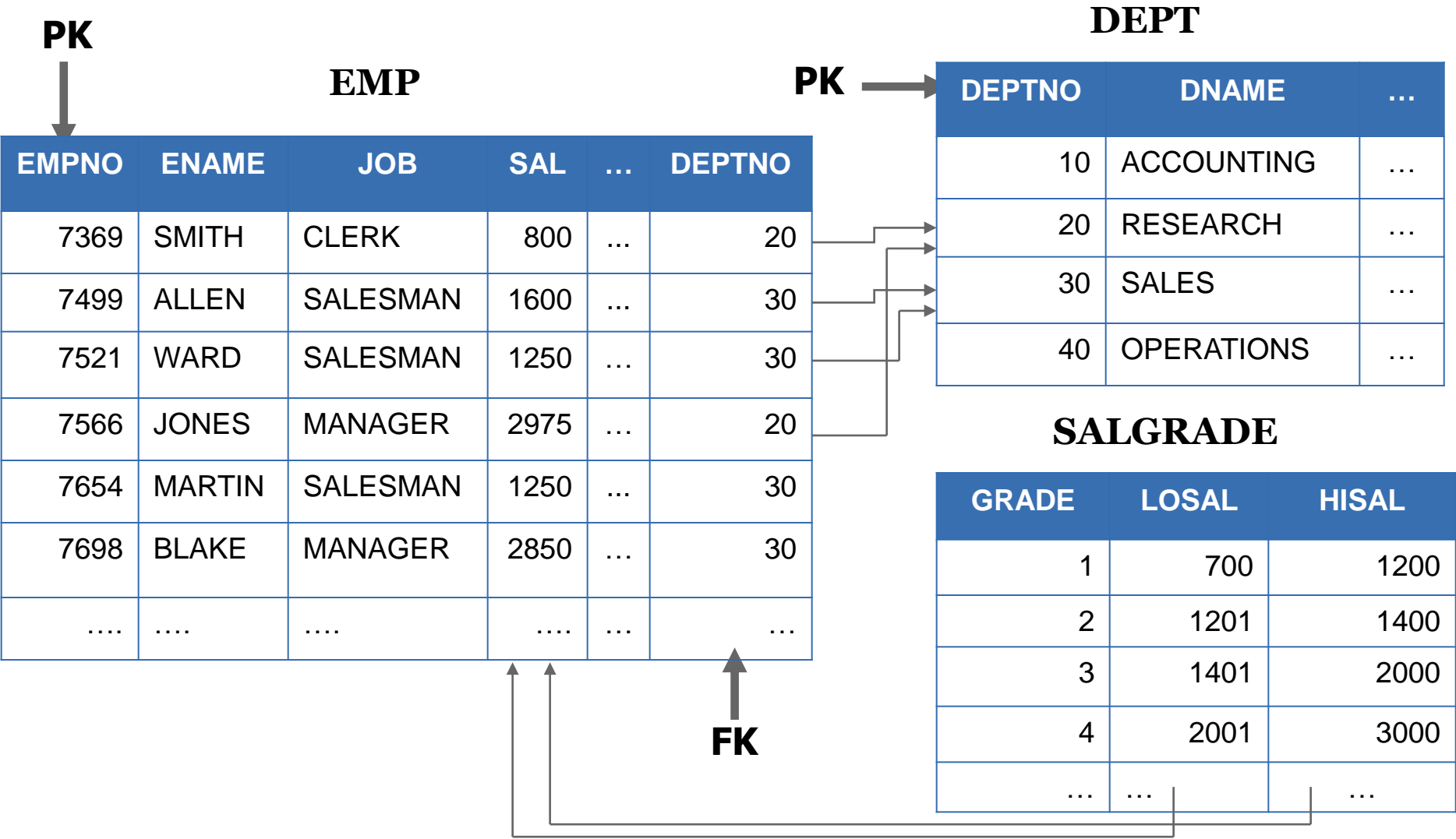
# SQL (Structured Query Language)

---

## ❑ 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**

# 실습 스키마 구조





# SQL 강의 #2

## – SQL Query I

### 목 차

---

1. 기본 SELECT
2. SINGLE-ROW Function

# 기본 SELECT

# SELECT

---

❑ 데이터베이스에서 원하는 데이터를 검색하는 것

❑ Syntax

– **SELECT** [DISTINCT] 컬럼\_리스트

**FROM** 테이블\_리스트

[**WHERE** 조건]

[**GROUP BY** 컬럼\_리스트 [**HAVING** 조건]]

[**ORDER BY** 컬럼\_리스트 [ASC | DESC]];

## 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

[illegible]

# 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: 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

# 산술연산

## □ 기본적인 산술연산 사용 가능

- +, -, \*, /, 괄호 등
- 우선순위: \* / , + -
- 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** 컬럼에는 사용할 수 없음
- ❑ **NULL**을 포함한 산술연산의 결과는 **NULL**
  - `SELECT sal, comm, (sal+comm)*12 FROM emp;`
- ❑ **NVL(expr1, expr2)**
  - expr1이 NULL이면 expr2를 출력한다.
  - 데이터타입이 호환 가능 해야 함.
  - `SELECT sal, comm, (sal+NVL(comm,0))*12 연봉 FROM emp;`



# Column Alias

- ❑ 컬럼의 제목을 변경
- ❑ AS는 생략 가능
- ❑ 큰따옴표(“ ”)을 사용하여 alias내에 공백이나 특수문자를 포함할 수 있다.
- ❑ 형태
  - SELECT ename name FROM emp;
  - SELECT ename as name 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

## || 연산자

### ❑ 문자열 결합(Concatenation) 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?

# WHERE

---

## □ 다음 조건을 만족하는 쿼리를 작성하시오.

- 급여가 3000 이상인 직원의 사번, 이름, 급여를 조회하시오.
- 이름이 WARD인 직원과 동일한 급여를 받는 직원의 사번, 이름, 급여를 조회하시오.
- 이름이 WARD인 직원과 동일한 부서에 속한 직원의 사번, 이름, 부서 번호를 조회하시오.
- 10, 30번 부서에 속한 직원의 사번, 이름, 부서번호를 조회하시오.
- 급여가 3000과 5000 사이를 받는 직원의 사번, 이름, 급여를 조회하시오.
- 82년 1월 1일부터 83년 12월 31일 사이에 입사한 직원의 이름, 입사일을 조회하시오.
- 커미션을 받는 직원의 이름, 커미션을 조회하시오.
- 직무가 SALESMAN이 아닌 직원의 이름, 직무를 조회하시오.
- 직무가 MANAGER와 SALESMAN이 아닌 직원의 이름, 직무를 조회하시오.

## ❑ Wildcard를 이용한 문자열 매칭

### ❑ Wildcard

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

# WHERE

---

## □ 다음 조건을 만족하는 쿼리를 작성하시오.

- 직무에 'ER' 이 포함된 직원의 이름과 직무를 조회하시오.
- 이름이 'S'로 시작하는 직원의 이름을 조회하시오.
- 직무에 'ER' 이 포함되어 있으면서 이름이 'S'로 시작하는 직원의 이름과 직무를 조회하시오.
- 81년도에 입사한 직원의 이름과 입사일을 조회하시오.
- 12월에 입사한 직원의 이름과 입사일을 조회하시오.

## ORDER BY

---

❑ 주어진 컬럼 리스트의 순서로 결과를 정렬

❑ 결과 정렬 방법

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

❑ 여러 컬럼 정의 가능

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

---

# **SINGLE ROW FUNCTION**



# SQL Function

---

## ❑ Single-Row Function : 하나의 Row를 입력으로 받는 함수

- 숫자함수
- 문자함수
- 날짜함수
- 기타함수

## ❑ Aggregation Function: 집합함수 : max(), min(), sum(), avg(), count(), count(\*)

예) `select max(sal), min(sal), sum(sal), avg(sal), count(sal), count(*) from emp;`

# 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
SUBSTR(s, m, n)	부분 문자열, m번째부터 길이 n인 문자열 반환
TRANSLATE(s, from, to)	s에서 from 문자열의 각 문자를 to문자열의 각 문자로 변환
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
MOD(m, n)	나머지	MOD(13,2)	1
ROUND(m, n)	소수점아래 n자리까지 반올림	ROUND(4.567,2)	4.57
TRUNC(m, n)	소수점아래 n자리미만 버림	TRUNC(4.567,2)	4.56

# Date 함수

Function	Purpose
SYSDATE	현재 날짜 시간 반환

## 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

[illegible]

# SQL 강의

## – SQL Query II

### 목 차

---

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

---

**JOIN**

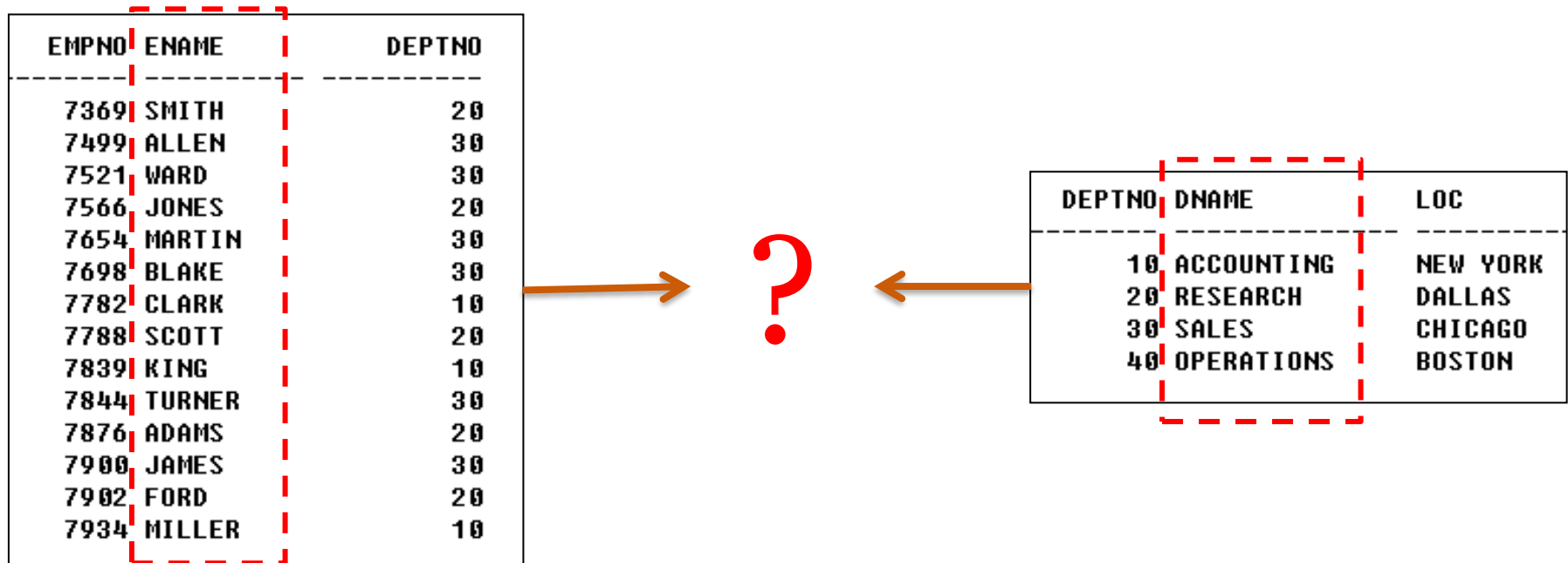


# Join

❑ 둘 이상의 테이블을 합쳐서 하나의 큰 테이블로 만드는 것

❑ 필요성

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




# 카티전 프로젝트

## ❑ 두 테이블에서 그냥 결과를 선택하면?

- SELECT ename, dname from emp, dept
- 결과: 두 테이블의 행들의 가능한 모든 쌍이 추출됨  
(Cartesian Product)

## ❑ Cartesian Product를 막기 위해서는

올바른 Join조건을 WHERE 절에 설정해야 함.



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

...

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(FK) = t2.col3(PK)
```

## □ 설명

- FROM 절에 필요로 하는 테이블을 모두 적는다.
- 컬럼 이름(FK, PK)의 모호성을 피하기 위해 Table 이름에 Alias 사용
- 적절한 Join 조건을 Where 절에 부여 (일반적으로 테이블 개수 -1 개의 조인 조건이 필요)

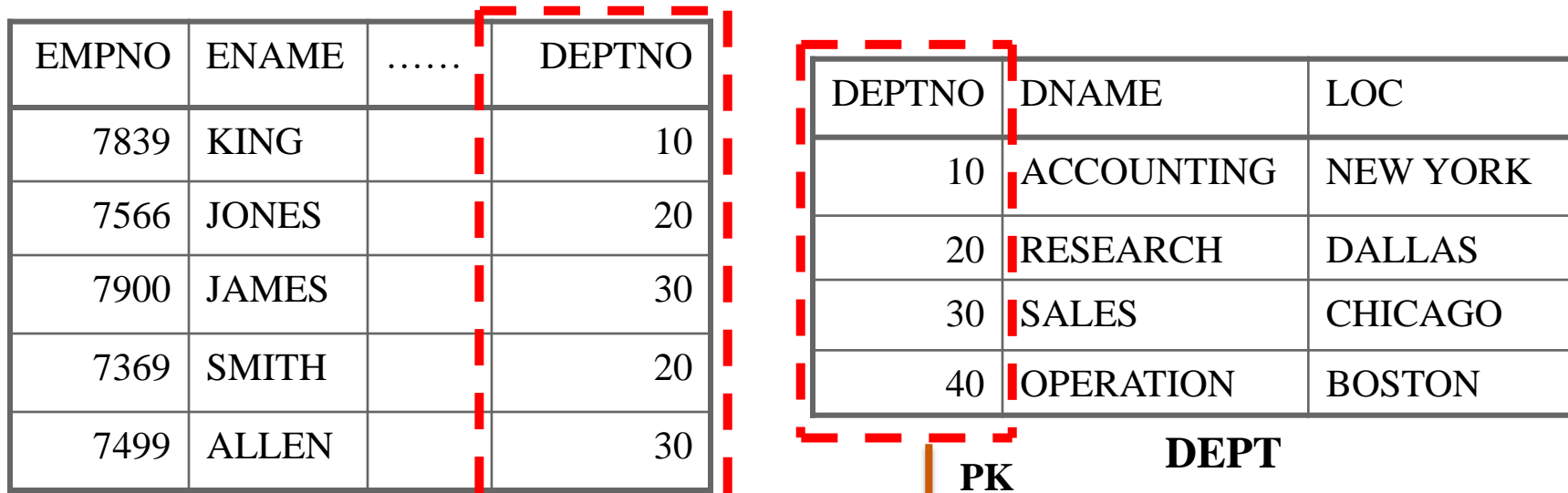
# Join 종류

---

## □ 용어

- Cross Join (Cartesian Product): 모든 가능한 쌍이 출력됨
- Inner Join: Join 조건을 만족하는 Row만 출력됨
- Outer Join: Join 조건을 만족하지 않는 Row도 출력됨
- Self Join: 자기 자신과 조인

# Equi-Join



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	SALES	DALLAS
7499	ALLEN		30	30	SALES	CHICAGO

**SELECT \* FROM EMP, DEPT**  
**WHERE EMP.DEPTNO = DEPT.DEPTNO**

# Outer Join

---

## □ 정의

- Join 조건을 만족하지 않는 데이터까지 출력할 때 사용하는 조인
  - LEFT OUTER JOIN은 왼쪽 테이블(dept)의 모든 데이터를 보겠다는 의미
- ```
SELECT *  
FROM dept d LEFT OUTER JOIN emp e  
ON d.deptno = e.deptno;
```

# Outer Join

---

## ❑ 문제

- 부서 정보가 없는 직원을 등록하고 부서에 속하지 않은 직원까지 조회하는 조인을 작성하시오.
- `INSERT INTO EMP VALUES(8000, 'DOOLY', 'CLERK', 7698, '2021-01-01', 1300.00, NULL, NULL);`

# Outer Join

---

- RIGHT OUTER JOIN은 오른쪽 테이블(dept)의 모든 데이터를 보겠다는 의미

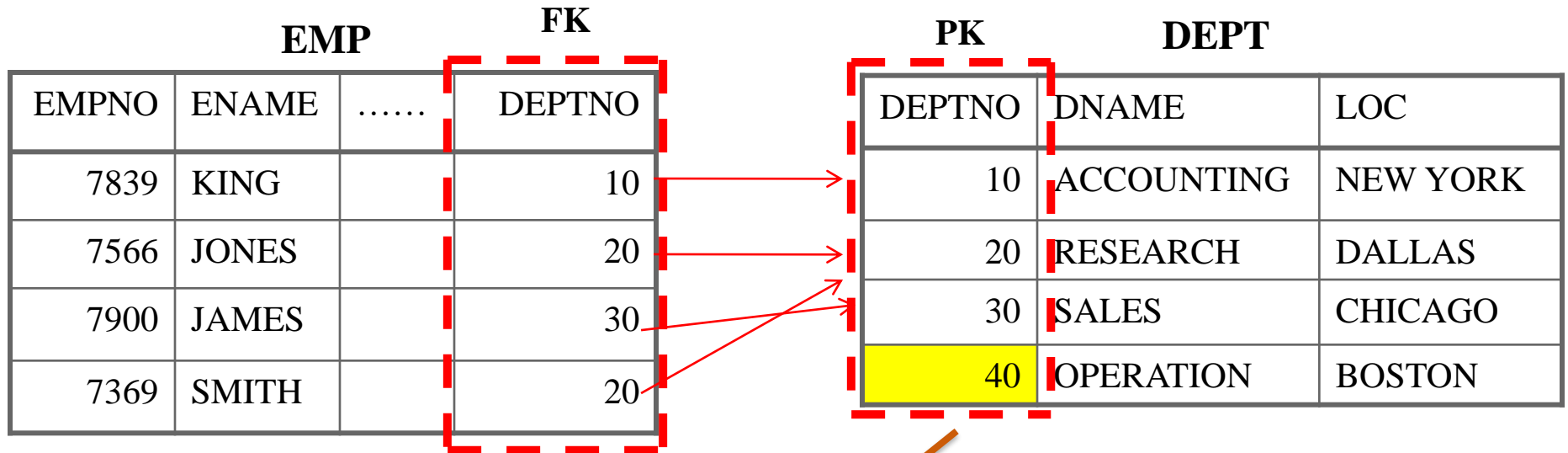
SELECT \*

FROM emp e RIGHT OUTER JOIN dept d

ON e.deptno = d.deptno;



# Outer Join



| EMPNO | ENAME | SELECT * FROM EMP E<br>RIGHT OUTER JOIN DEPT D<br>ON E.DEPTNO = D.DEPTNO |    |    |  | DNAME      | LOC      |
|-------|-------|--------------------------------------------------------------------------|----|----|--|------------|----------|
| 7839  | KING  |                                                                          |    |    |  | ACCOUNTING | NEW YORK |
| 7566  | JONES |                                                                          |    |    |  | 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.MGR = E2.EMPNO
```

| 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 |

---

# **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

# 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 급여?

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



Error!

# 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.

### ❑ 문제

- 부서 이름 별 평균 급여를 출력하시오.



# HAVING 절

❑ Aggregation 결과에 대해 다시 condition을 검사할 때

❑ 일반적인 오류

– 평균 월급이 2000 이상인 부서는?

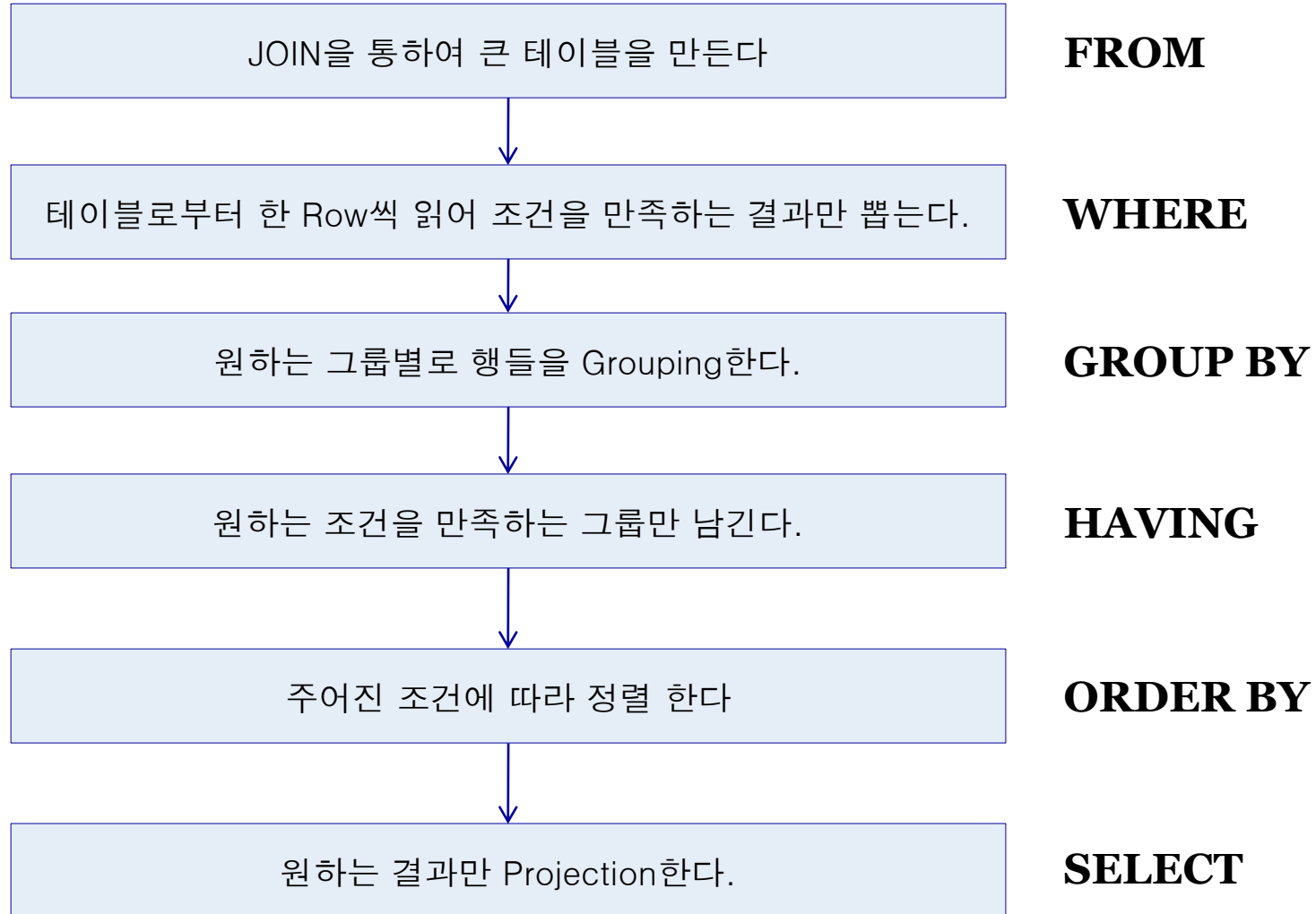
```
SELECT deptno, AVG(sal)
FROM emp
WHERE AVG(sal) > 2000
GROUP BY deptno;
```

❑ 주의

– WHERE 절은 Aggregation 이전, HAVING 절은 Aggregation 이후의 filtering

– Having절에는 Group by에 참여한 컬럼이나 Aggregate 함수만 사용가능

# 단일 SQL 문 실행 순서



---

**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 ROW인 경우

```
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');
```

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

- ❑ 'CHICAGO'지역에서 근무하는 직원의 정보를 조회하시오.

# SQL 강의 #4

## - DDL

### 목 차

---

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

---

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  
(DELETE)

# 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 |
| ...    | ...   | ...    | ...        |
|        |       |        |            |

# 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

# 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<br>(p:38, s:-84~127, 21Byte)<br>자리수 지정 없으면 NUMBER(38)<br>INT, |
| DATE           | 고정길이 날짜+시간, 7Byte                                                                          |

## □ 참고

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

# ALTER TABLE

---

## ❑ 컬럼 추가

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

## ❑ 컬럼 수정

- ALTER TABLE book **ALTER COLUMN** 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;

### ❑ RENAME

- **RENAME** book **TO** article;

# 제약조건

---

## □ 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 DBMS가 자동 부여



## 제약조건 (1/3)

### □ 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)

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

## 제약조건 (2/3)

### ❑ PRIMARY KEY

- NOT NULL + UNIQUE
- 테이블 당 하나만 나올 수 있음
- 복합 컬럼에 대해서 정의 가능

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

### ❑ CHECK

- 임의의 조건 검사.

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

## 제약조건 (3/3)

### □ FOREIGN KEY

- 참조 무결성 제약
- 일반적으로 REFERENCE 테이블의 PK를 참조
- REFERENCE 테이블에 없는 값은 삽입 불가

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

# 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

```
ALTER TABLE book DROP CONSTRAINT c_emp_u ;
```

# CASCADE CONSTRAINT

---

- ❑ 제약조건이 걸려있는 테이블이나 컬럼은 삭제 시 에러 발생
- ❑ 컬럼이나 테이블 DROP할 때 관련 제약조건도 함께 삭제 할 때

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

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

# SQL 강의 #5

## – DML

### 목 차

---

1. INSERT, DELETE, UPDATE
2. Transaction Control

# Data Manipulation Language

---

## □ 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);
```



# 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**

- SYSDATE : Current date and time.

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

### ❑ **DELETE** 와 **TRUNCATE**의 차이점

- Delete는 Rollback 가능 but 대량의 log 등을 유발하므로 Truncate보다 느림

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

---

**TRANSACTION**

---

# Transaction

---

## □ 정의:

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

# Transaction in ORACLE

---

## □ 구성

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

## □ 트랜잭션 정의

- 시작
  - 첫 DML이 시작되면 트랜잭션 시작
- 명시적 종료: COMMIT / ROLLBACK
- 묵시적 종료
  - DDL, DCL 등이 수행될때 (automatic commit)
  - 시스템 오류 (automatic rollback)

# SQL 강의 #6

## – Other DB Objects

### 1. Sequence

---

**SEQUENCE**

---



# 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]
```

- ❑ **INCREMENT BY n:**번호 간격 (기본 : 1)
- ❑ **START WITH n:** 시작번호(기본 : 1)

# Sequence 사용하기

## ❑ Pseudo Columns

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

## ❑ Example

```
DROP SEQUENCE seq_empno;
```

```
CREATE SEQUENCE seq_empno  
INCREMENT BY 1  
START WITH 8001;
```

```
SELECT * FROM EMP;
```

```
INSERT INTO EMP VALUES(SEQ_EMPNO.NEXTVAL, 'DOOLY', 'CLERK',  
7698, '2021-01-01', 1300.00, NULL, 30);
```

```
SELECT * FROM EMP;
```

## Other Syntax

---

❑ **DROP SEQUENCE** *sequence\_name*;