MYONGJI UNIVERSITY

Database

Lecture 4-1. Structured Query Language (Part I)

Spring 2024

Prof. Jik-Soo Kim, Ph.D.

E-mail: jiksoo@mju.ac.kr

Notes

- **Readings**
  - Chapter 3: Introduction to SQL (Database System Concepts 7th Edition)

History

- IBM Sequel language developed as part of System R project at the IBM San Jose Research Laboratory
- Renamed **Structured Query Language (SQL)**
- ANSI and ISO *standard* SQL:
  - SQL-86
  - SQL-89
  - SQL-92
  - SQL:1999
  - SQL:2003, …
- Commercial systems offer most, if not all, SQL-92 features plus varying feature sets from later standards and special proprietary features
  - not all examples here may work on your particular system (consult the user manuals)

Structured Query Language

- **Data-Definition Language (DDL)**
  - defining relation schemas, deleting relations, modifying relation schemas
- **Data-Manipulation Language (DML)**
  - querying information from the database, inserting tuples, deleting tuples, modifying tuples
- **Integrity** 무결성제약조건
  - DDL includes commands for specifying *integrity constraints*
  - **any updates that *violates* integrity constraints are disallowed !!**

Structured Query Language

- Transaction control (TCL)
  - commands for specifying the beginning and ending of transactions
- View definition
  - DDL includes commands for defining views
- Authorization
  - commands for specifying access rights to relations and views
- Embedded SQL and dynamic SQL
  - define how SQL statements can be embedded within general-purpose programming languages

Data Definition Language

- Allows the specification of not only a set of relations but also information about each relation including:
  - the **schema** for each relation
  - the **domain** of values associated with each attribute
  - **integrity constraints**

  - also, other information such as set of indices to be maintained for each relation, etc.

Domain Types in SQL

- **char(n)**: Fixed length character string with user-specified length *n*
- **varchar(n)**: Variable length character strings with user-specified maximum length *n*
- **int**: Integer (a finite subset of the integers that is machine-dependent)
- **smallint**: Small integer (a machine-dependent subset of the integer domain type)
- **numeric(p,d)**: Fixed point number with user-specified precision of *p* digits with *d* digits to the right of decimal point
  - e.g., numeric(3,1) allows 44.5 to be stores exactly, but not 444.5 or 0.32

    숫자3글자에 소수점1째자리까지
- **real, double precision**: Floating point and double-precision floating point numbers with machine-dependent precision
- **float(n)**: Floating point number with user-specified precision of at least *n* digits

Domain Types in SQL

- Each type may include a special value called *null*

- recommend you always use the *varchar* type instead of the char
  - storing "Avi" as type **char**(10) vs. type **varchar**(10)

- SQL also provides the **nvarchar** type to store *multilingual* data using the Unicode representation
  - many databases allow Unicode (in the UTF-8 representation) to be stored even in **varchar** types

Create Table Construct

- An SQL relation is defined using the **create table** command

  create table r (A_1 D_1, A_2 D_2, ..., A_n D_n,
  (integrity-constraint$_1$),
  ...,
  (integrity-constraint$_k$))

  – r is the name of the relation 중복X
  – each A_i is an attribute name in the schema of relation r
  – D_i is the data type of values in the domain of attribute A_i

- Example:

```
create table instructor (
        ID              char(5),
        name            varchar(20),
        dept_name       varchar(20),
        salary          numeric(8,2));
```

Integrity Constraints in Create Table

- **not null** primary key는 자동으로 not null이므로 생략함.
- **primary key** (A_1, ..., A_n)
- **foreign key** (A_m, ..., A_n) **references** *r*
- **SQL prevents any update to the database that violates an integrity constraint!**

Example:

```
create table instructor (
        ID            char(5),
        name          varchar(20) not null,
        dept_name     varchar(20),
        salary        numeric(8,2),
        primary key (ID),
        foreign key (dept_name) references department);
```

도메인 네임이 같으면 생략해도
되지만 이름이 같지않다면 명시할 것

primary key declaration on an attribute *automatically* ensures **not null** (**Entity Integrity Constraints**)

Examples of Relation Definitions

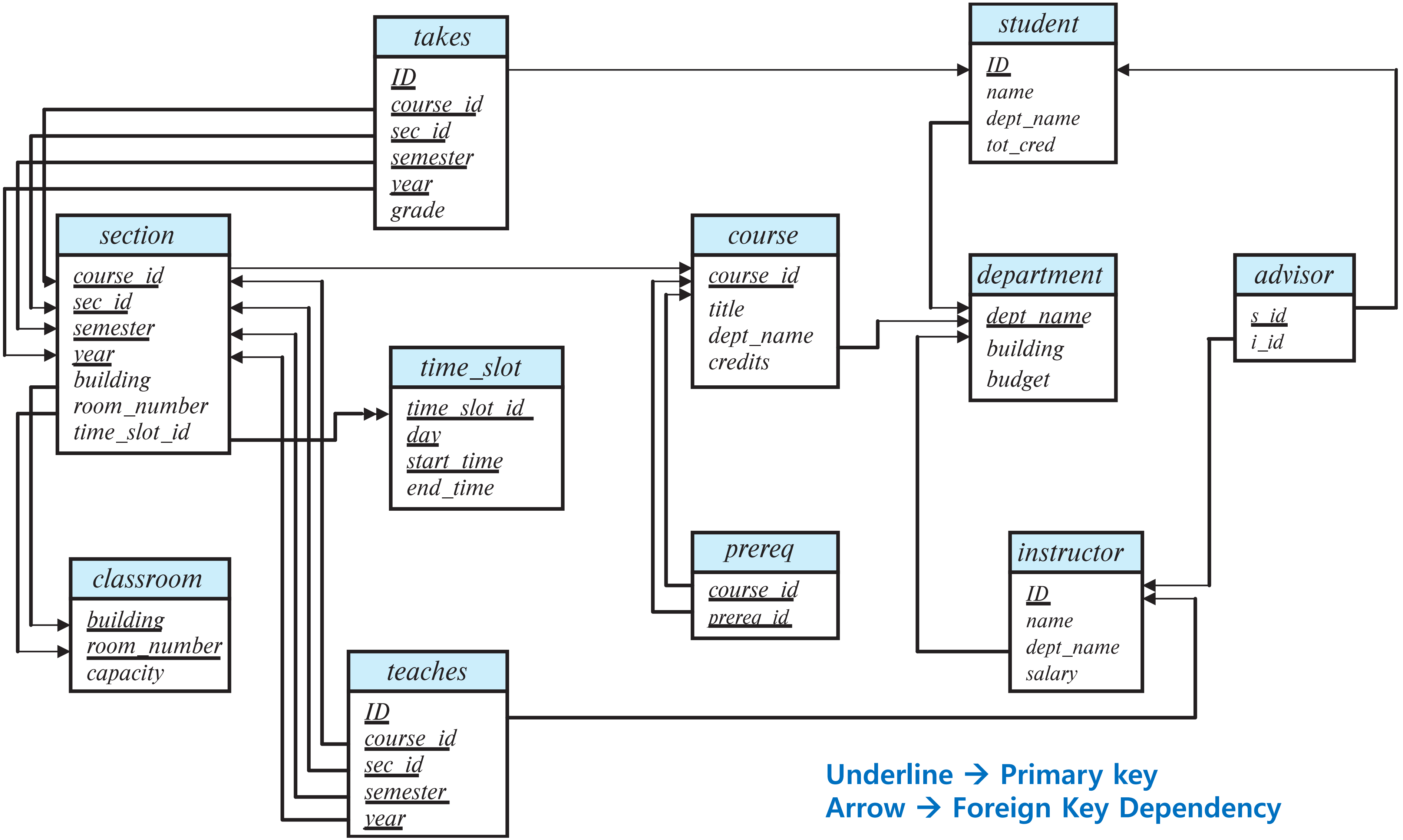
-
  ```
  create table student (
          ID              varchar(5),
          name            varchar(20) not null,
          dept_name       varchar(20),
          tot_cred        numeric(3,0),
          primary key (ID),
          foreign key (dept_name) references department);
  ```

-
  ```
  create table takes (
          ID              varchar(5),
          course_id       varchar(8),
          sec_id          varchar(8),
          semester        varchar(6),
          year            numeric(4,0),
          grade           varchar(2),
          primary key (ID, course_id, sec_id, semester, year) ,
          foreign key (ID) references  student,
          foreign key (course_id, sec_id, semester, year) references section);
  ```

Examples of Relation Definitions

-
  ```
  create table course (
          course_id       varchar(8),
          title           varchar(50),
          dept_name       varchar(20),
          credits         numeric(2,0),
          primary key (course_id),
          foreign key (dept_name) references department);
  ```

Schema Diagram for University Database



takes
ID
course_id
sec_id
semester
year
grade

student
ID
name
dept_name
tot_cred

section
course_id
sec_id
semester
year
building
room_number
time_slot_id

course
course_id
title
dept_name
credits

department
dept_name
building
budget

advisor
s_id
i_id

time_slot
time_slot_id
day
start_time
end_time

prereq
course_id
prereq_id

instructor
ID
name
dept_name
salary

classroom
building
room_number
capacity

teaches
ID
course_id
sec_id
semester
year

Underline → Primary key
Arrow → Foreign Key Dependency

Updates to tables

- **Drop Table** 테이블 자체를 지우기
  - remove a relation from an SQL database
  - **drop table** *r*

- **Alter Table**
  - change the schema of the database
  - **alter table** *r* **add** *A D*
    - where *A* is the name of the attribute to be added to relation *r* and *D* is the domain of *A*
    - all exiting tuples in the relation are assigned *null* for the new attribute
  - **alter table** *r* **drop** *A*
    - where *A* is the name of an attribute of relation *r*
    - dropping of attributes is not supported by many databases

되도록 alter table은 쓰지않는것이 좋음
처음 만들때 잘만들 것

Data Definition Languages

Updates to tables

- **Insert**
  - load data into the relation   테이블 스키마 목록순대로 넣어야된다.
  - e.g., **insert into** *instructor* **values** (‘10211’, ’Smith’, ’Biology’, 66000);

  작은 따옴표로 문자열(string) 나타냄

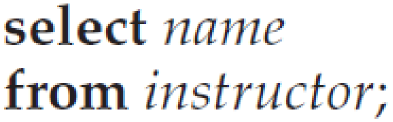
- **Delete**
  - delete tuples from a relation
  - e.g., **delete from** *student*

  → remove *all* tuples from the *student* relation

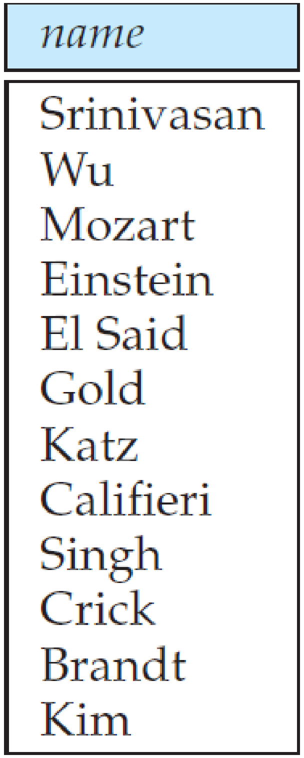
Data Manipulation Languages

Query the tables

- **Select**
  - the basic structure of an SQL *query* consists of three clauses: **select**, **from**, and **where**
  - the query takes as its input the relations listed in the **from** clause, operates on them as specified in the **where** and **select** clauses, and then produces a relation as the result



```
select name
from instructor;
```



name
Srinivasan
Wu
Mozart
Einstein
El Said
Gold
Katz
Califieri
Singh
Crick
Brandt
Kim

Data Manipulation Languages

DATA DEFINITION LANGUAGE – ORACLE CASE STUDY

Data Definition Language (DDL)

- Create Table, Drop Table, Truncate Table, Alter Table
- Data Type
- Constraint (NOT NULL, DEFAULT, CHECK, REFERENCE)

DDL 요약

- 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 TABLE**문 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
…	…	…	…

기본 데이터 타입

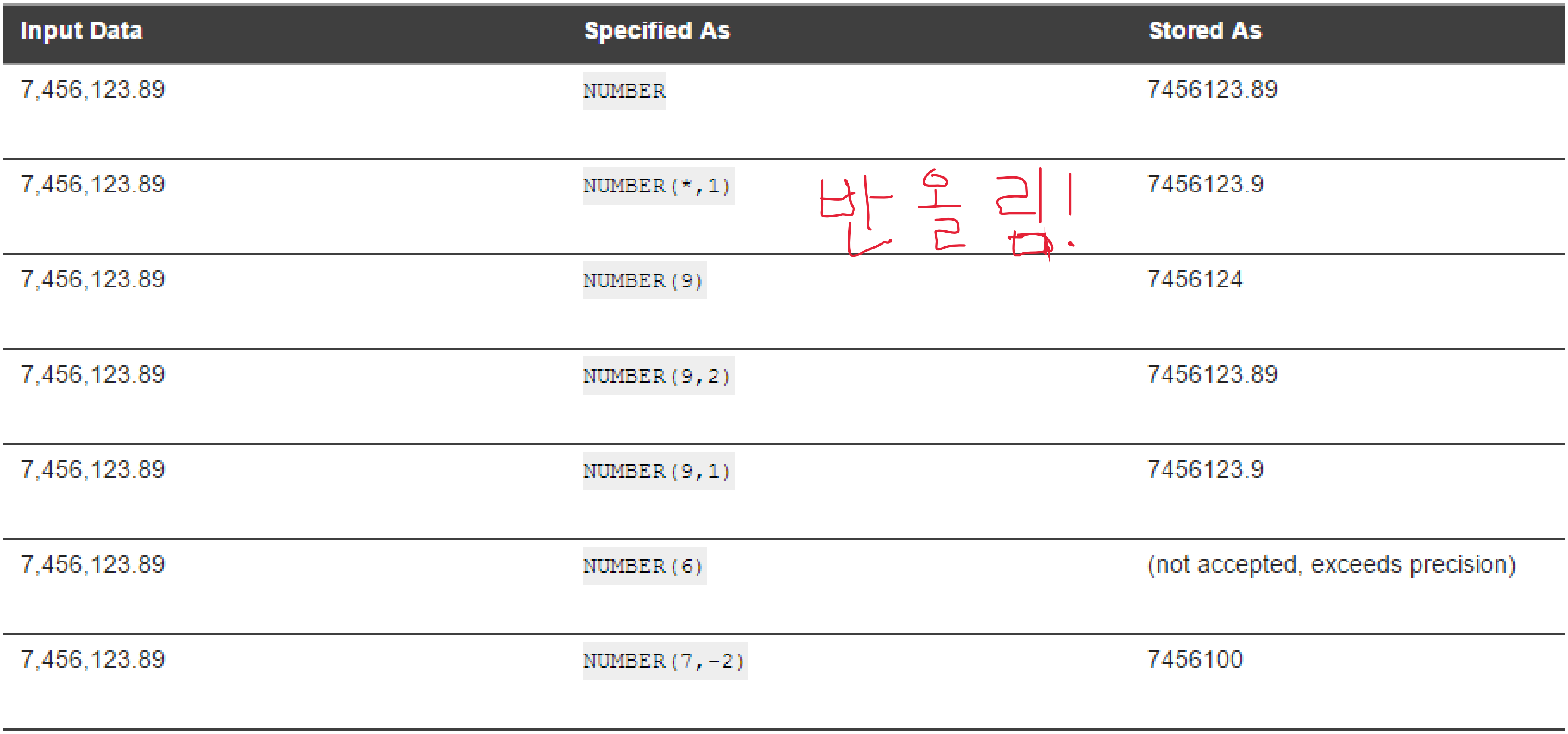
Data Type	Description
VARCHAR2(size)	가변길이 문자열 (최대 4000byte)
CHAR(size)	고정길이 문자열 (최대 2000byte)
NUMBER(p,s)	가변길이 숫자. 전체 p자리 중 소수점 이하 s자리 (p:38, s:-84~127, 21Bytes) 자리수 지정 없으면 NUMBER(38)
DATE	고정길이 날짜+시간, 각 필드별로 7Bytes (Year, Month, Day, …)

- 참고
  - VARCHAR2와 CHAR의 차이점 주의! char는 그메모리전체무조건 할당
  - INT, FLOAT 등의 ANSI Type도 내부적으로 NUMBER(38)로 변환됨
  - DATE = DATE+TIME의 결합형

[참고]Oracle Data Types:
https://docs.oracle.com/cd/B28359_01/server.111/b28318/datatype.htm

기본 데이터 타입

- NUMBER Datatype 활용 사례



Input Data	Specified As	Stored As
7,456,123.89	NUMBER	7456123.89
7,456,123.89	NUMBER(*,1)	7456123.9
7,456,123.89	NUMBER(9)	7456124
7,456,123.89	NUMBER(9,2)	7456123.89
7,456,123.89	NUMBER(9,1)	7456123.9
7,456,123.89	NUMBER(6)	(not accepted, exceeds precision)
7,456,123.89	NUMBER(7,-2)	7456100

반올림!

*** 만약 scale이 precision 보다 크면??**

→ Number of digits on the right of decimal point = Scale

→ Minimum number of zeroes right of decimal = Scale - Precision

Subquery를 이용한 테이블 생성

- **Subquery**의 결과와 동일한 테이블 생성됨
  - 질의 결과 레코드들이 포함됨
  - <u>NOT NULL 제약조건만 상속됨</u>

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

emp라는 테이블에서 직종이 '세일즈'만 뽑아서
empSALES라는 테이블을 만들어라.

- Create a table with the same *schema* as an existing table:
  create table *temp_account* **like** *account* 복제하는 것

TABLE 종류

- **User Tables**
  - a collection of tables created and maintained by the user
  - contain user information

- **Data Dictionary**
  - a collection of tables created and maintained by the Oracle Server
  - contain database information

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

기타 테이블 관련 명령

- 테이블 삭제
  - **DROP TABLE** book;

테이블 전체날리기
즉 schema + instane 전체날리기

- 데이터 삭제
  - **TRUNCATE TABLE** book;

(데이터)instane만 날리기
즉 schema는 존재

+a delete from book이랑 하는 일은 같다. (복구유무차이)

- Comment
  - **COMMENT ON TABLE** book **IS** ‘this is comment’;
- RENAME
  - **RENAME** book **TO** article;
- 주의:
  - **ROLLBACK의 대상이 아님(DDL)!**

즉, 실행하면 복구못함.

제약조건(Constraints)

- **Constraint**
  - Database 테이블 레벨에서 “특정한 규칙”을 설정함
  - 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
  - **테이블 제약조건**: [CONSTRAINT *이름*] constraint_type(column,..)

- 주의
  - 제약조건에 이름을 부여하지 않으면 Oracle이 Sys-C*n*의 형태로 자동 부여

제약조건: NOT NULL, UNIQUE

- **NOT NULL**
  - NULL 값이 들어올 수 없음
  - 컬럼 형태로만 제약조건 정의할 수 있음 (테이블 제약조건 불가)

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

- **UNIQUE**
  - 중복된 값을 허용하지 않음(NULL은 들어올 수 있음) null이 딱 한번 들어갈수 있음. (즉 null도 중복 불가)
  - 복합 컬럼에 대해서도 정의 가능
  - 자동적으로 인덱스 생성 검색할때 필요

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

제약조건: PRIMARY KEY, CHECK

- **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 (                ex)평점
        rate NUMBER CHECK (rate IN (1,2,3,4,5))
);
```

제약조건: FOREIGN KEY

- **FOREIGN KEY**
  - **참조 무결성(Referential Integrity)** 제약
  - 일반적으로 REFERENCE 테이블의 **PK**를 참조
  - REFERENCE 테이블에 없는 값은 삽입 불가
  - REFERENCE 테이블의 레코드 삭제 시 동작 두가지 가능
    - **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)
      REFERENCES author(id)
      ON DELETE SET NULL
);
```

확실히 명시해줄 것

ADD/DROP CONSTRAINTS

- 제약조건 추가
  - **ALTER TABLE** *테이블이름* **ADD CONSTRAINT** …
  - NOT NULL은 추가하지 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;
```

DATA MANIPULATION LANGUAGE

Data Manipulation Language (DML)

- SELECT
- DELETE
- INSERT
- UPDATE

BASIC STRUCTURE OF SQL QUERIES

SELECT

- 데이터베이스에서 원하는 데이터를 **검색, 추출**

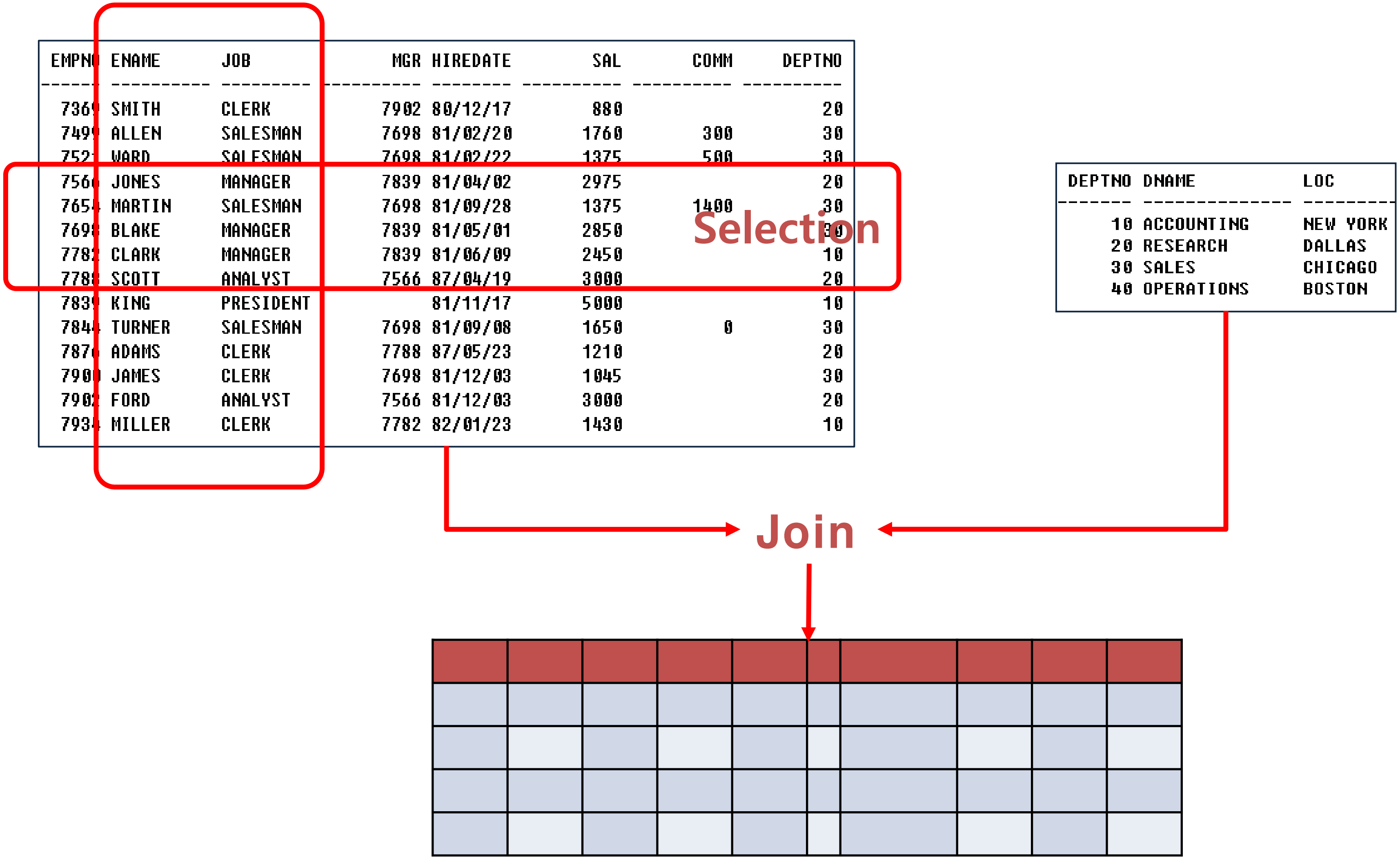
- Syntax

  -
    ```
    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**: 두 개 이상의 테이블 결합
  - 기타: 각종 계산, 정렬, 집계(**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
```

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

The SELECT Clause

- SQL is based on set and relational operations with certain modifications and enhancements
- A typical SQL query has the form:

select $A_1, A_2, \ldots, A_n$
from $r_1, r_2, \ldots, r_m$
where P

  - A_i represents an attribute
  - r_i represents a relation
  - P is a predicate

- This query is equivalent to the **relational algebra** expression

$$\prod_{A1,\ A2,\ \ldots,\ An}(\sigma_P\,(r_1 \text{ x } r_2 \text{ x } \ldots \text{ x } r_m))$$

- The result of an SQL query is a *relation*

The SELECT Clause

- The **select** clause lists the attributes desired in the result of a query
  - corresponds to the *projection* operation of the relational algebra

- Example: find the names of all instructors:

  select *name*
  from *instructor*

sql에선 중복제거 X

- NOTE: SQL names are *case insensitive* (i.e., you may use upper- or lower-case letters)
  - e.g., *Name* ≡ *NAME* ≡ *name*

The SELECT Clause

- SQL allows duplicates in relations as well as in query results
  - in practice, duplication elimination is time-consuming
- To force the elimination of duplicates, insert the keyword **distinct** after select

- Find the department names of all instructors and remove duplicates

distinct쓰면 중복제거하고 select됨.

```
select distinct dept_name
from instructor
```

- The keyword **all** specifies that duplicates should not be removed

```
select all dept_name
from instructor
```

The SELECT Clause

- An asterisk in the select clause denotes "all attributes"

  select *
  from *instructor*

- An attribute can be a literal with no **from** clause

  select '437'

  – results is a table with one column and a single row with value 437
  – can give the column a name using:

  select '437' **as** *FOO*

  rename

- An attribute can be a literal with **from** clause

  select 'A'
  from *instructor*

  – result is a table with one column and *N* rows (number of tuples in the *instructor* table) each row with value "A"

The SELECT Clause

- The **select** clause can contain *arithmetic* expressions involving +, –, ∗, / and operate on constants or attributes of tuples
  - the query:

    select *ID, name, salary/12*
    from *instructor*

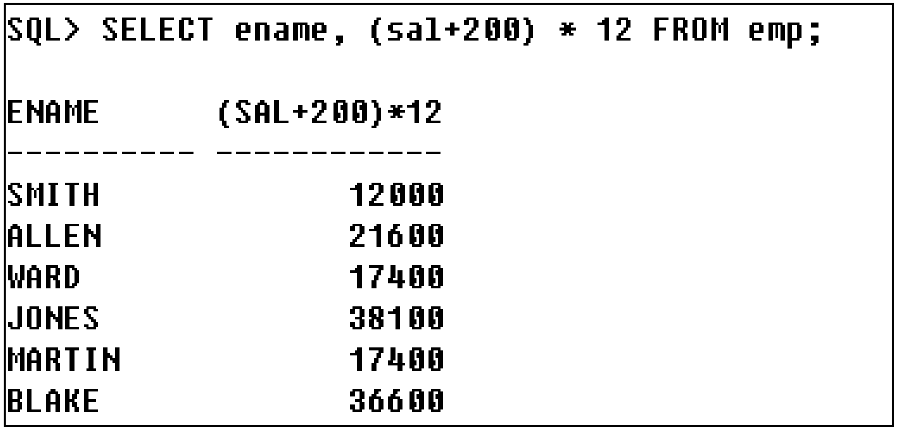
    would return a relation that is the same as the *instructor* relation, except that the value of the attribute *salary* is divided by 12

  - can rename "s*alary/12"* using the **as** clause:

    select *ID, name, salary/12* **as** *monthly_salary*

SELECT문에서의 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;

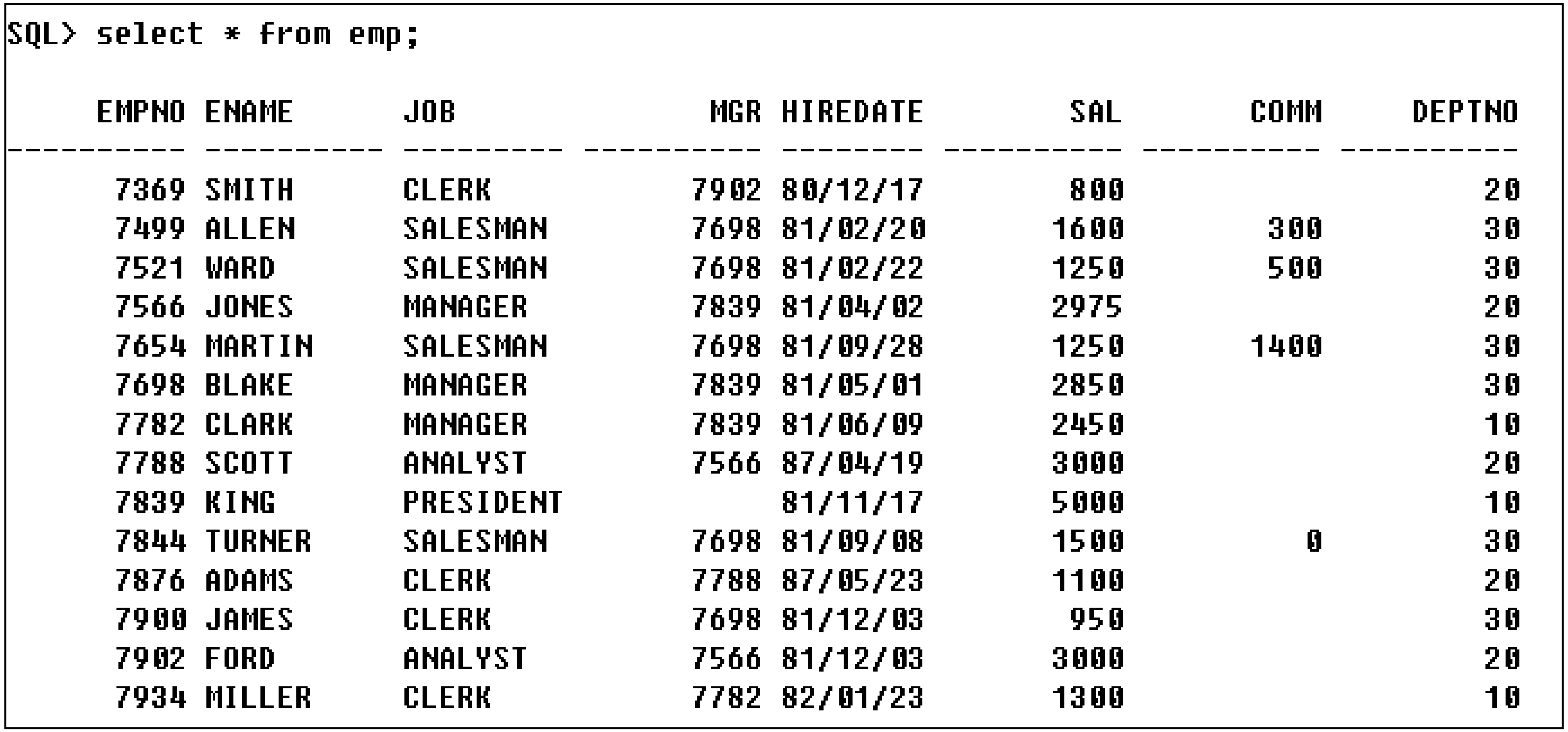


```
SQL> SELECT ename, (sal+200) * 12 FROM emp;

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

SELECT 예제

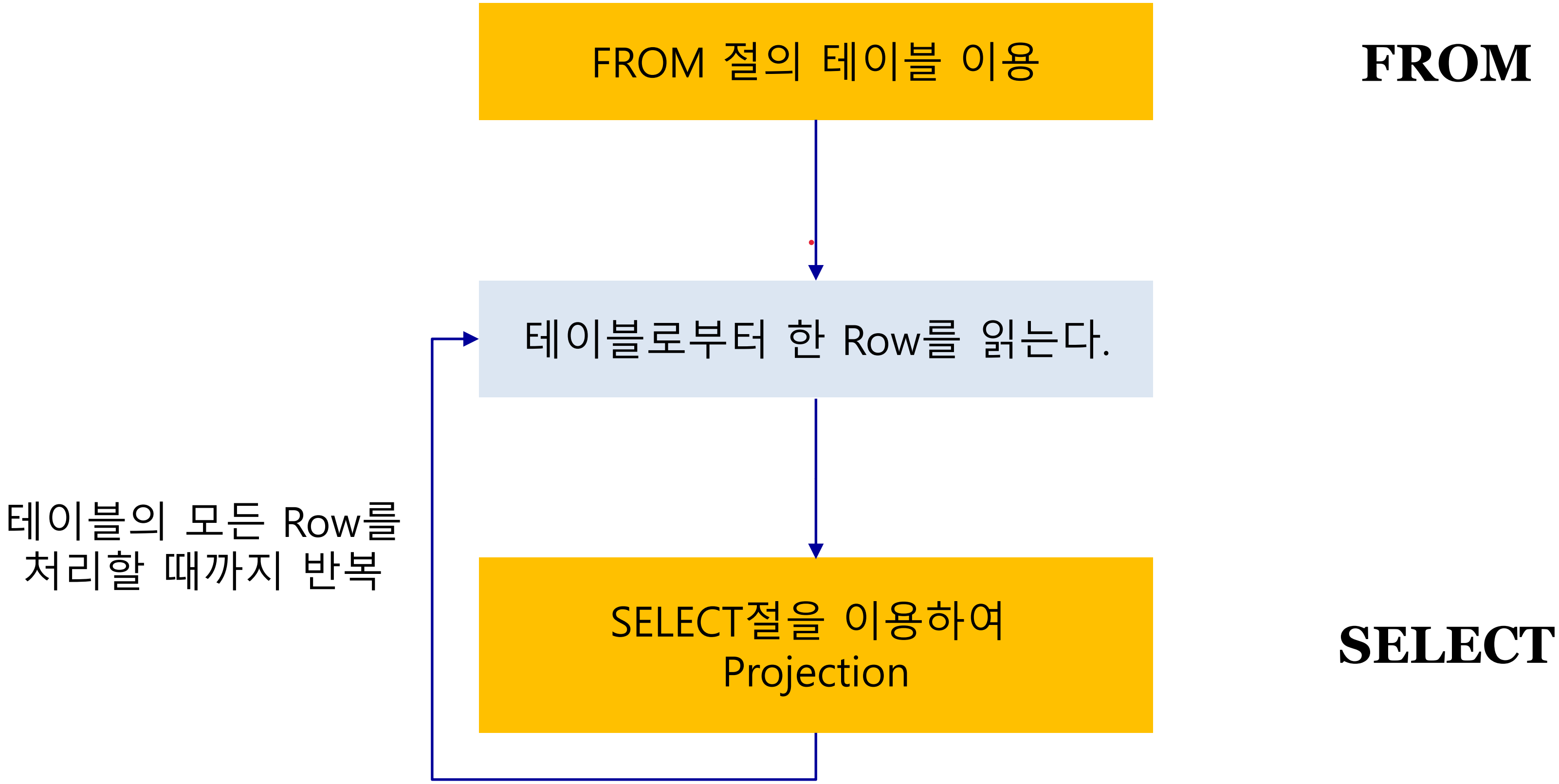
- SELECT * FROM emp;
- SELECT ename FROM emp;
- SELECT ename, job FROM emp;



```
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 처리 개념


FROM 절의 테이블 이용
테이블로부터 한 Row를 읽는다.
SELECT절을 이용하여 Projection
테이블의 모든 Row를 처리할 때까지 반복
FROM
SELECT

실제 모든 SQL이 이렇게 처리되는 것은 아닙니다. SQL의 처리 순서는 DBMS가 질의 최적화 과정을 통하여 결정합니다. 질의의 종류, 데이터의 분포 등에 따라 질의의 실제 순서는 달라질 수도 있습니다.

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: 둘 다 만족? 둘 중 하나만 만족?
  - NOT: 만족하지 않음?
  - ANY, ALL: 집합 중 어느 한열, 집합 중 모든 열 (다른 비교연산자와 함께 사용)
  - EXIST: 결과 Row가 하나라도 있나? (subquery에서)

The WHERE Clause

- The **where** clause specifies *conditions* that the result must satisfy
  - corresponds to the *selection predicate* of the relational algebra
- To find all instructors in "Comp. Sci." dept

  select *name*
  from *instructor*
  where *dept_name* = 'Comp. Sci.'

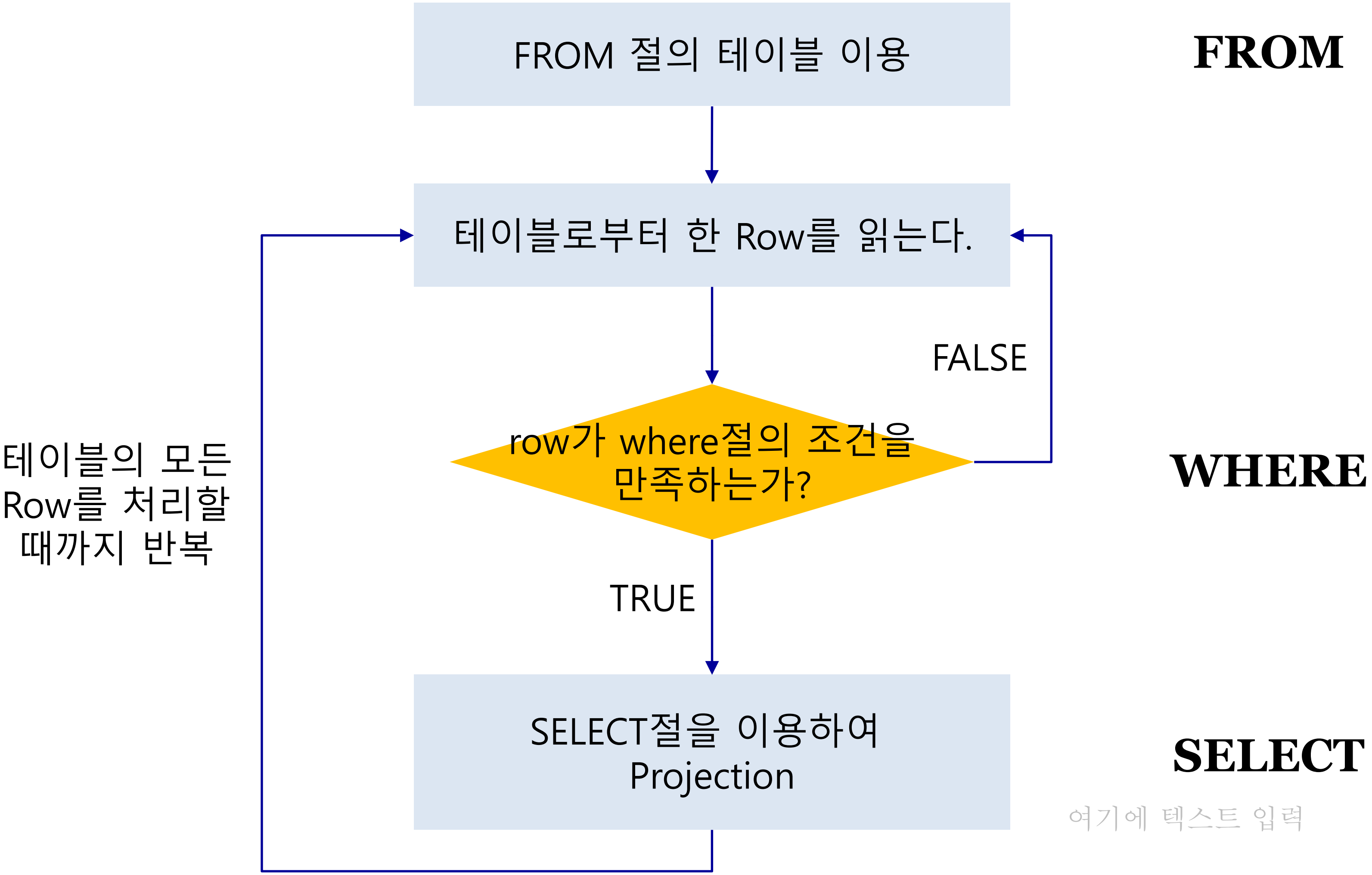
  where절은 vaule값을 나타내므로
  대소문자 구별할 것

- Comparison results can be combined using the logical connectives **and**, **or**, and **not**
  - to find all instructors in Comp. Sci. dept with salary > 80000

  select *name*
  from *instructor*
  where *dept_name* = 'Comp. Sci.' **and** *salary* > 80000

- Comparisons can be applied to results of arithmetic expressions

WHERE 절 처리 개념


FROM 절의 테이블 이용
FROM
테이블로부터 한 Row를 읽는다.
FALSE
row가 where절의 조건을 만족하는가?
WHERE
테이블의 모든 Row를 처리할 때까지 반복
TRUE
SELECT절을 이용하여 Projection
SELECT
여기에 텍스트 입력

실제 모든 SQL이 이렇게 처리되는 것은 아닙니다. SQL의 처리 순서는 DBMS가 질의 최적화 과정을 통하여 결정합니다. 질의의 종류, 데이터의 분포 등에 따라 질의의 실제 순서는 달라질 수도 있습니다.

The FROM Clause

- The **from** clause lists the relations involved in the query
  - corresponds to the *cartesian product* operation of the relational algebra (두 개 이상의 Relation들이 올 수 있음)

- Find the Cartesian product *instructor X teaches*

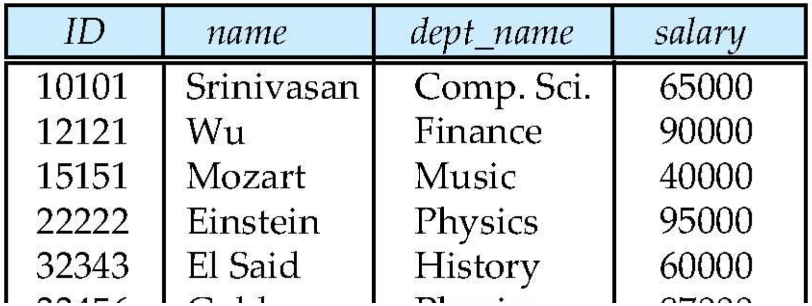
  select ∗
  from *instructor, teaches*

  - generates every possible *instructor – teaches* pair with all attributes from both relations
  - for common attributes (e.g., *ID*), the attributes in the resulting table are renamed using the relation's name (e.g., *instructor.ID*)

- Cartesian product is not very useful directly but useful when it is **combined with where-clause condition**

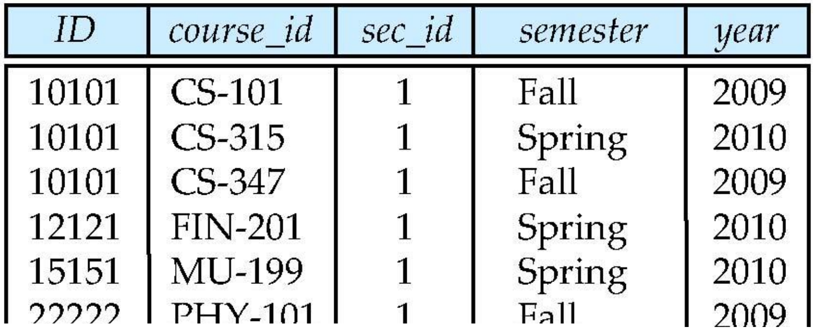
Cartesian Product

instructor

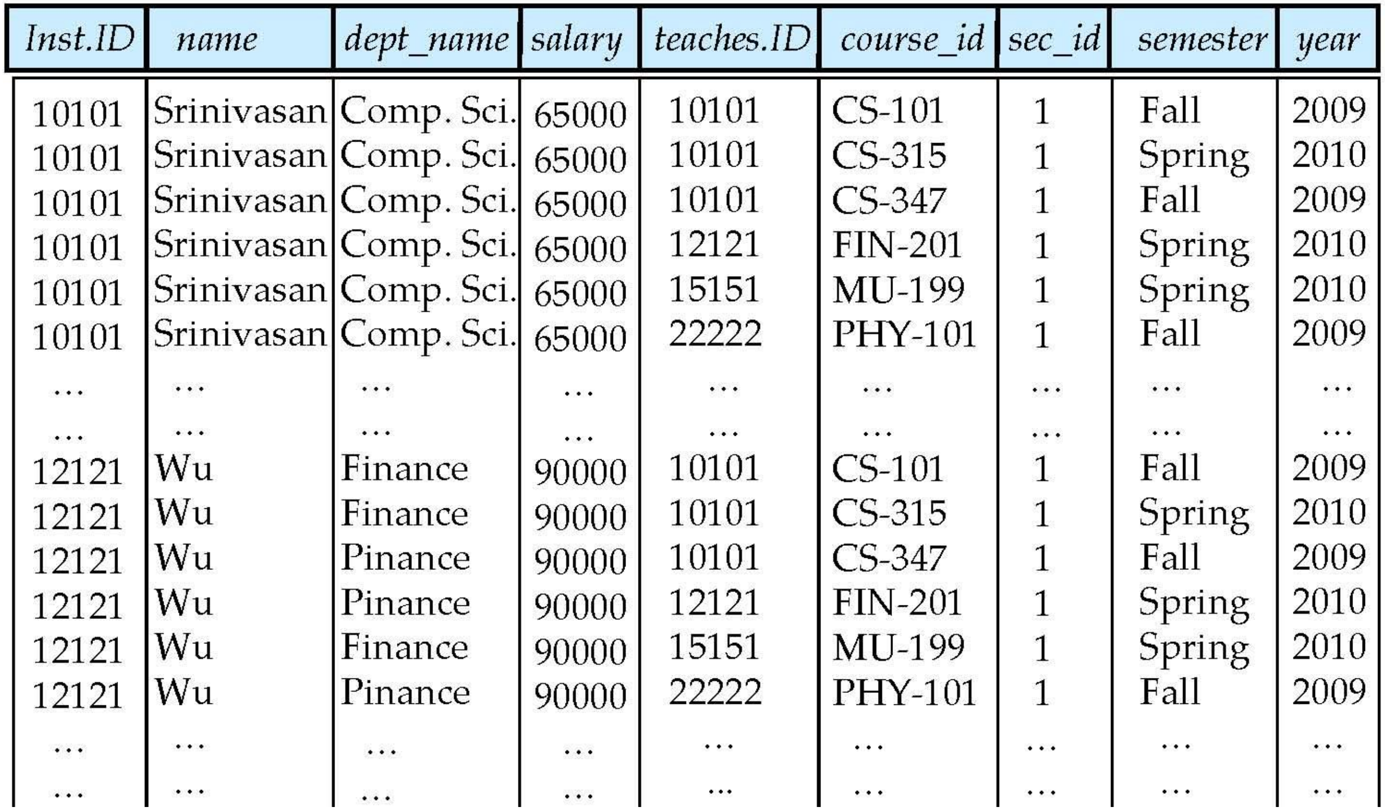


ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000

teaches



ID	course_id	sec_id	semester	year
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009



Inst.ID	name	dept_name	salary	teaches.ID	course_id	sec_id	semester	year
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	12121	FIN-201	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	15151	MU-199	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
12121	Wu	Finance	90000	10101	CS-101	1	Fall	2009
12121	Wu	Finance	90000	10101	CS-315	1	Spring	2010
12121	Wu	Pinance	90000	10101	CS-347	1	Fall	2009
12121	Wu	Pinance	90000	12121	FIN-201	1	Spring	2010
12121	Wu	Finance	90000	15151	MU-199	1	Spring	2010
12121	Wu	Pinance	90000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

Examples of select-from-where

- Find the *names* of all instructors who have taught some course and the *course_id*

  -
    ```
    select name, course_id
    from instructor, teaches
    where instructor.ID = teaches.ID
    ```

- Find the *names* of all instructors in the “Art” department who have taught some course and the *course_id*

  -
    ```
    select name, course_id
    from instructor, teaches
    where instructor.ID = teaches.ID and
               instructor. dept_name = 'Art'
    ```

Semantics of select-from-where

- In general, the meaning of an SQL query can be understood as follows:
  1) generate a Cartesian product of the relations listed in the **from** clause
  2) apply the predicates specified in the **where** clause on the result of Step 1
  3) for each tuple in the result of Step 2, output the attributes (or results of expressions) specified in the **select** clause

→ *What* the result of an SQL query should be, not How it should be executed!

ADDITIONAL OPERATIONS IN SQL QUERIES

The Rename Operation

- The SQL allows renaming relations and attributes using the **as** clause:

  old-name **as** *new-name*

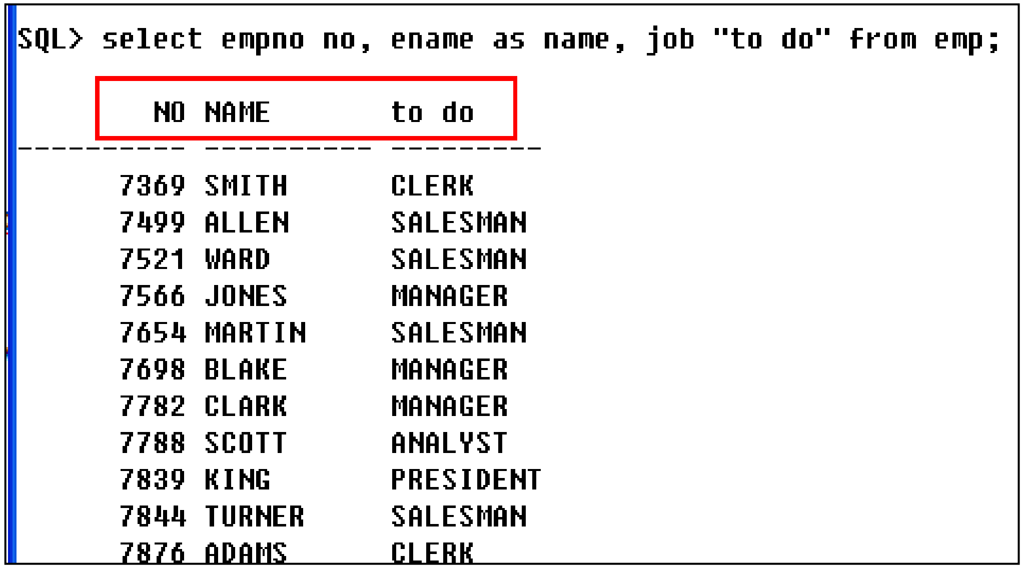
- Find the names of all instructors who have a higher salary than some instructor in 'Comp. Sci.'
  - **select distinct** *T.name*
    from *instructor* **as** *T*, *instructor* **as** *S*   임시로 T,S로 테이블가명 취급
    where *T.salary* > *S.salary* **and** *S.dept_name* = 'Comp. Sci.'

- Keyword **as** is *optional* and may be omitted   즉 as생략가능

  instructor **as** *T* ≡ *instructor T*

The Rename Operation

- 컬럼의 제목을 변경 가능 즉 큰따옴표시 string취급
- 큰따옴표(“ ”)를 사용하여 공백이나 특수문자를 포함할 수 있음
- 예)
  - SELECT ename name FROM emp;
  - SELECT ename as name FROM emp;
  - SELECT ename “as” FROM emp; as라는 테이블로 rename
  - SELECT (sal + comm) “Annual Salary” FROM emp; Annual Salary로 rename



```
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
```

String Operations

- A string-matching operator for comparisons on strings
  - **like** uses *patterns* that are described using two special characters
    - percent (%): The % character matches any *substring* 0개이상을 가지는 any string
    - underscore (_): The _ character matches any *character*

- Find the names of all instructors whose name includes the substring “dar”

```
select name
from instructor
where name like '%dar%'
```

- Match the string “100%”

```
like '100\%'  escape  '\'
```

in that above we use backslash (\) as the escape character

String Operations

- Wildcard를 이용한 문자열 부분 매칭
- Wildcard
  - % : 임의의 길이의 문자열(공백 문자 가능)
  - _ : 한 글자
- Escape
  - **ESCAPE** 뒤의 문자열로 시작하는 문자는 Wildcard가 아닌 것으로 해석
  - 모든 문자가 ESCAPE로서 가능

- 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

String Operations

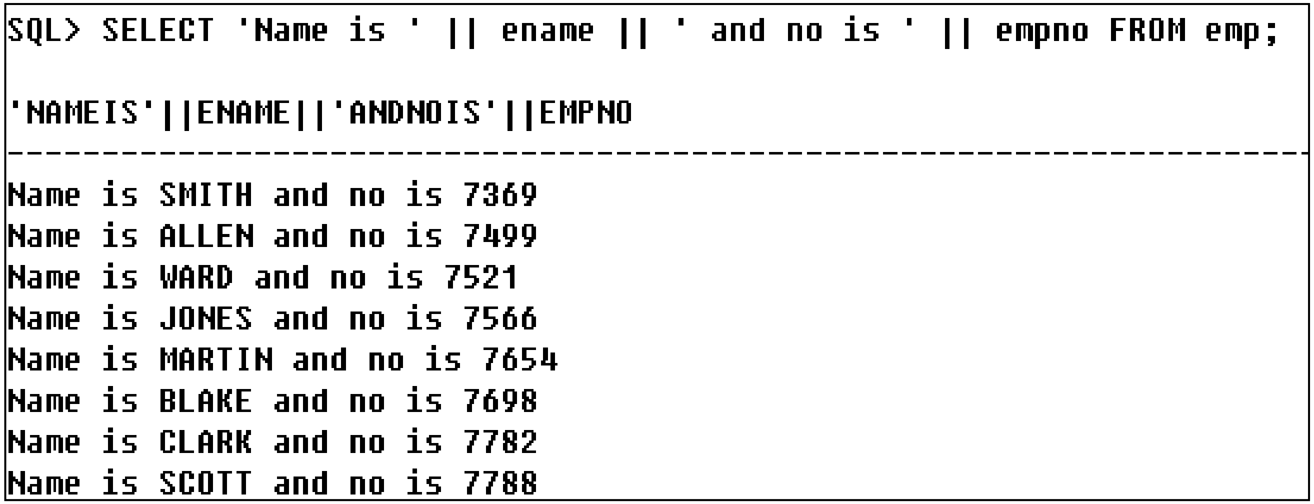
- Patterns are *case sensitive* 대소문자 구분한다.

- Pattern matching examples:
  - ‘Intro%’ matches any string beginning with “Intro”
  - ‘%Comp%’ matches any string containing “Comp” as a substring
  - ‘_ _ _’ matches any string of exactly three characters
  - ‘_ _ _ %’ matches any string of at least three characters

- SQL supports a variety of string operations such as
  - concatenation (using “||”)
  - converting from upper to lower case (and vice versa)
  - finding string length, extracting substrings, etc.

Literal

- SELECT 절에 사용되는 문자, 숫자, Date 타입 등의 상수
- Date 타입이나 문자열은 **작은따옴표 (‘ ‘)**로 둘러싸야 함
- 문자열 결합(Concatenation) 연산자: ||
- 예)
  - 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
```

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

ORDER BY

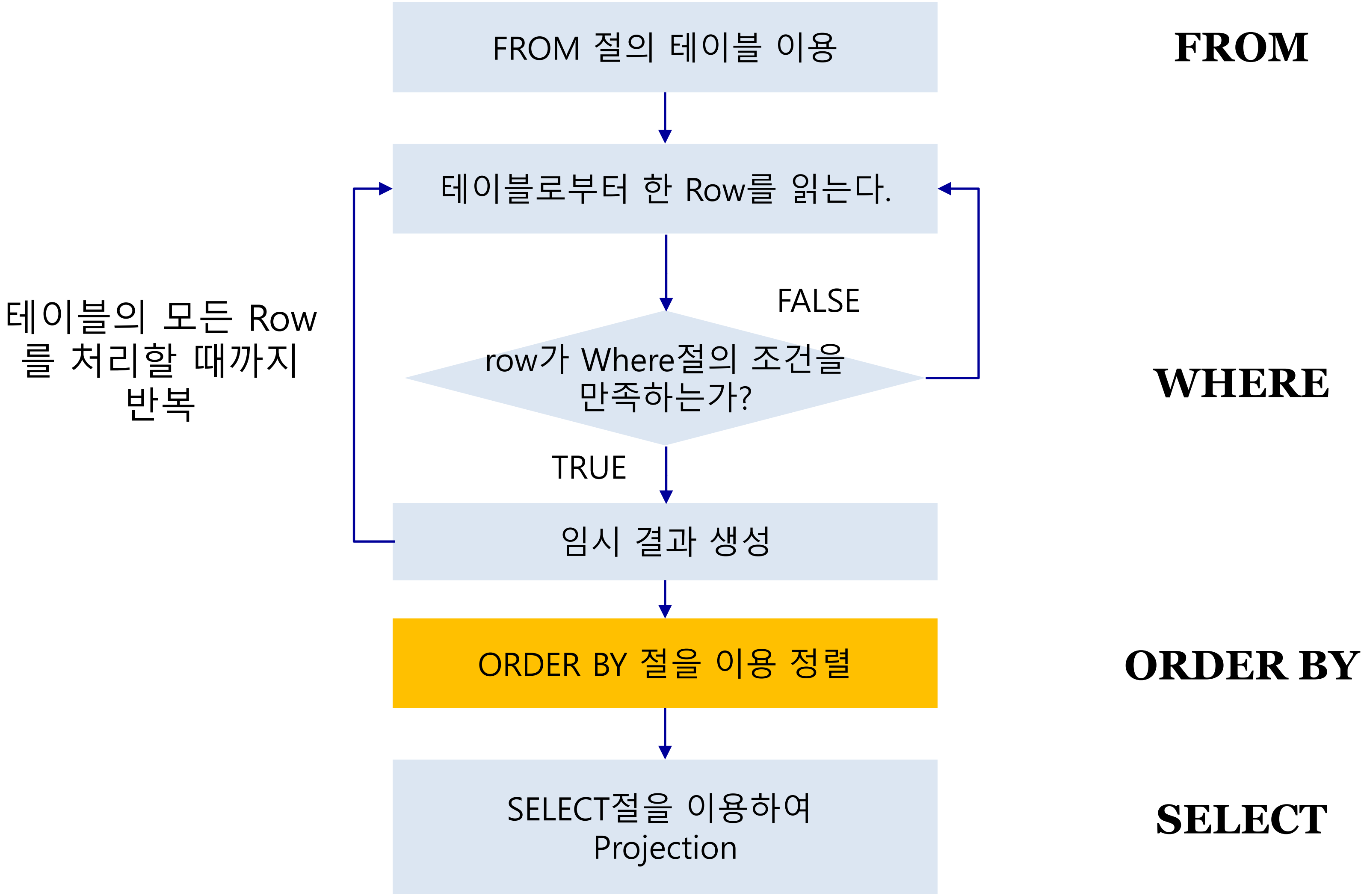
- 주어진 컬럼 리스트의 순서로 결과를 정렬

- 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 사용 가능
  - 예) SELECT * FROM emp **ORDER BY** deptno, sal **DESC**
    - 부서번호순으로 오름차순 정렬하고, sal가 높은 사람부터(내림차순) 출력

ORDER BY 절 처리 개념



FROM 절의 테이블 이용

테이블로부터 한 Row를 읽는다.

row가 Where절의 조건을 만족하는가?

FALSE

TRUE

임시 결과 생성

ORDER BY 절을 이용 정렬

SELECT절을 이용하여 Projection

테이블의 모든 Row를 처리할 때까지 반복

FROM

WHERE

ORDER BY

SELECT

실제 모든 SQL이 이렇게 처리되는 것은 아닙니다. SQL의 처리 순서는 DBMS가 질의 최적화 과정을 통하여 결정합니다. 질의의 종류, 데이터의 분포 등에 따라 질의의 실제 순서는 달라질 수도 있습니다.

ORDER BY

- List in alphabetic order the names of all instructors

```
select distinct name
from    instructor
order by name
```

- We may specify **desc** for descending order or **asc** for ascending order, for each attribute; ascending order is the default
  - Example: **order by** *name* **desc**

- Can sort on multiple attributes
  - Example: **order by** *dept_name, name*

Where Clause Predicates

- SQL includes a **between** comparison operator
- Example: Find the names of all instructors with salary between $90,000 and $100,000 (that is, ≥ $90,000 and ≤ $100,000)
  - **select** *name*
    from *instructor*
    where *salary* **between** 90000 **and** 100000 이상, 이하 포함!
- Tuple comparison
  - **select** *name*, *course_id*
    from *instructor*, *teaches*
    where (*instructor.ID*, *dept_name*) = (*teaches.ID*, 'Biology');

Duplicates

- In relations with duplicates, SQL can define how many copies of tuples appear in the result

- **Multiset** versions of some of the relational algebra operators – given multiset relations r_1 and r_2:

1. **$\sigma_\theta (r_1)$:** if there are c_1 copies of tuple t_1 in r_1, and t_1 satisfies selections σ_θ, then there are c_1 copies of t_1 in $\sigma_\theta (r_1)$.
2. **$\Pi_A (r)$:** for each copy of tuple t_1 in r_1, there is a copy of tuple $\Pi_A (t_1)$ in $\Pi_A (r_1)$ where $\Pi_A (t_1)$ denotes the projection of the single tuple t_1.
3. **$r_1 \times r_2$:** if there are c_1 copies of tuple t_1 in r_1 and c_2 copies of tuple t_2 in r_2, there are c_1 x c_2 copies of the tuple t_1. t_2 in r_1 x r_2

Duplicates

- Example: Suppose multiset relations r_1 (A, B) and r_2 (C) are as follows:

  $r_1 = \{(1,a), (2,a)\}$ $\quad$ $r_2 = \{(2), (3), (3)\}$

- Then $\Pi_B(r_1)$ would be {(a), (a)}, while $\Pi_B(r_1) \times r_2$ would be

  {(a,2), (a,2), (a,3), (a,3), (a,3), (a,3)} a,3이 중복이지만 그대로 뽑음.

- SQL duplicate semantics:

  select $A_1, A_2, \ldots, A_n$
  from $r_1, r_2, \ldots, r_m$
  where P

  is equivalent to the *multiset* version of the expression:

$$\prod_{A_1, A_2, \ldots, A_n} (\sigma_P (r_1 \times r_2 \times \ldots \times r_m))$$

THE END