7. SQL

(Structured Query Language)

SQL

- Initially, SEQUEL language (System R) project at IBM
- Three relational languages : QUEL, QBE, SQL
- (Later renamed) Structured Query Language (SQL)
- Currently, Standard Relational database Language
  - Users need to be less concerned to migrate to other types of relational DBMS
- Very-High-level language
  - **Declarative** (= Non-procedural) language
  - Users do not need to specify how (= what order of) to execute query operations.
  - Say "what to do" rather than "how to do"
  - Relational algebra is a procedural language.

SQL

- Table (= relation), row (= tuple), column (= attribute)

- A table consists of :
  - Base Table
  - View (= Virtual table)

- Basic commands;
  - Data Definition Language (DDL)
  - Data Manipulation Language (DML)
  - Data Control Language (DCL)

- DDL is used for defining schemas, tables, and views.
  - CREATE, DROP, ALTER

- DML is used for retrieving and modifying tuples in a table.
  - SELECT (FROM WHERE)
  - INSERT, DELETE, UPDATE

CREATE TABLE

- A table is defined using **CREATE TABLE** command:

CREATE TABLE R
(A_1 : D_1
A_2 : D_2
. . . . .
A_n : D_n)
Constraints)

- R : relation name
- A_i : attribute name
- D_i : domain (=data type) of A_i
- Constraints : Integrity Constraints

- A relation defined by CREATE TABLE is called a "**base table**".
  - The relation (and its tuples) are "physically stored".
  - All created tables are initially "empty".
  - Attributes are ordered as they are specified in CREATE TABLE

CREATE TABLE : Attribute Data Types

- System Defined
  - Numeric : **INT**, **SMALLINT**, **FLOAT**(or **REAL**), . . .
  - Character String : **CHAR(n)**, **VARCHAR(n)**, **CLOB**, . .
  - Bit String : **BIT(n)**, **BIT VARYING(n)**, **BLOB**, . .
  - Boolean : **TRUE**, **FALSE**, **UNKOWN**
  - DATE : **YEAR**, **MONTH**, **DAY**, . . .
  - . . . . . .

- User Defined
  - It is possible for users to specify domain of attribute directly;
  - For example, we can create domain "SSN-TYPE";

    CREATE DOMAIN SSN-TYPE **AS CHAR(9)**
  - We can use this SSN-TYPE for defining attributes SSN, Super_SSN, Mger-SSN, . . .;
  - What if we change later SSN data type to another one?

CREATE TABLE : Constraints

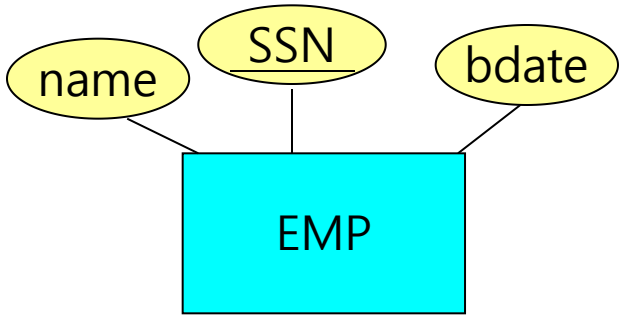
- Key Integrity
  - Primary key and keys can be specified by **PRIMARY KEY** and **UNIQUE**, respectively

- Entity Integrity
  - This must be specified by **NOT NULL** on **PRIMARY KEY**
  - **NOT NULL** also optionally can be specified on other attributes.
    : For example, employee names, phone may be specified by **non null**; What if new employee's name is unknown?

- Referential Integrity
  - This is specified by **FOREIGN KEY REFERENCES**
  - SQL's default action is to **reject** the operation that violates referential integrity violation
  - SQL also provides **user specified trigger** action

CREATE TABLE : Key/Entity Constraints

ER :



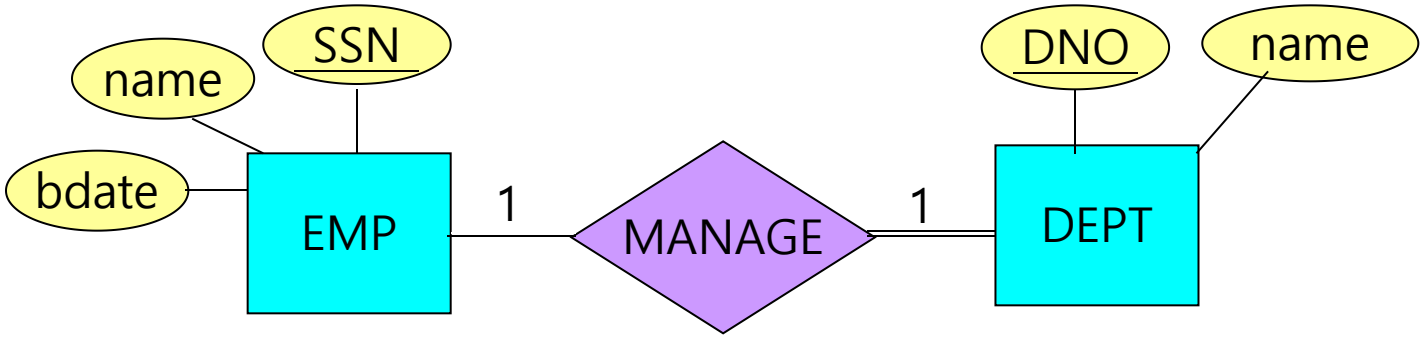
Relation :

EMP (SSN, name, bdate)

Table :

```
CREATE TABLE   EMP
( SSN     CHAR(9)        NOT NULL
  name    VARCHAR(10)    NOT NULL
  bdate   DATE
  PRIMARY KEY (SSN)
  UNIQUE (name)
  )
```

CREATE TABLE : Referential Constraints



EMP (SSN, name, bdate)

DEPT (DNO, name, Mgr_SSN)

```
CREATE TABLE   EMP
( SSN    CHAR(9)    NOT NULL
  name   CHAR(10)   NOT NULL
  bdate  DATE
  PRIMARY KEY (SSN)
  UNIQUE (name)   )
```

```
CREATE TABLE   DEPT
(DNO       INT          NOT NULL
 name      CHAR(10)
 Mgr_SSN   CHAR(9)      NOT NULL
 PRIMARY KEY(DNO),
 FOREIGN KEY(Mgr-SSN) REFERENCES EMP(SSN)
```

CREATE TABLE : Default/DOMAIN/CHECK

- **DEFAULT**
  - Default value is included in new tuple if explicit value is not provided
  - For example, default manager for new department is;

```
CREATE TABLE    DEPT
        Mgr_SSN   CHAR(9)   DEFAULT 999999999
```

  - If no default value is specified, the default DEFAULT value is NULL.

- **CHECK**
  - We can restrict domain to specific values;
  - For example, department numbers are between 1 and 20;

```
DNO INT . . .CHECK (DNO > 0) AND (DNO < 21)
```

  - CHECK can be also used with **CREATE DOMAIN**

```
CREATE DOMAIN DNO AS INT
DNO INT . . .CHECK (DNO > 0) AND (DNO < 21)
```

CREATE DOMAIN / DEFAULT

```
CREATE DOMAIN SSN-TYPE AS CHAR(9)
CREATE TABLE   EMP
 ( SSN   SSN-TYPE      NOT NULL
   name  CHAR(10)     NOT NULL
   bdate  DATE
   PRIMARY KEY (SSN)
   UNIQUE (name)   )
```

```
CREATE TABLE   DEPT
( DNO          INT          NOT NULL
  name         CHAR(10)  NOT NULL
  Mgr-SSN     SSN-TYPE
  PRIMARY KEY(DNO),
  UNIQUE(name),
  FOREIGN KEY(Mgr-SSN) REFERENCES EMP(SSN)
```

- What if we change later SSN into another type?

CREATE TABLE : Trigger

- SQL provides the following "Referential Trigger Action";

Action (Referencing)	Event (Referenced)
CASCADE **SET NULL** **SET DEFAULT**	**ON DELETE** **ON UPDATE**

- Total 6 trigger actions possible; For example:

(1) **ON UPDATE CASCADE**

: Change the value of referencing FK to the updated (new) PK value for all the referencing tuples;

(2) **ON DELETE CASCADE**

: Delete all the referencing tuples;

(3) **ON UPDATE SET DEFAULT**

(4) **ON DELETE SET DEFAULT**

: The value of FK in referencing tuples is changed to default value;

CREATE TABLE : Trigger

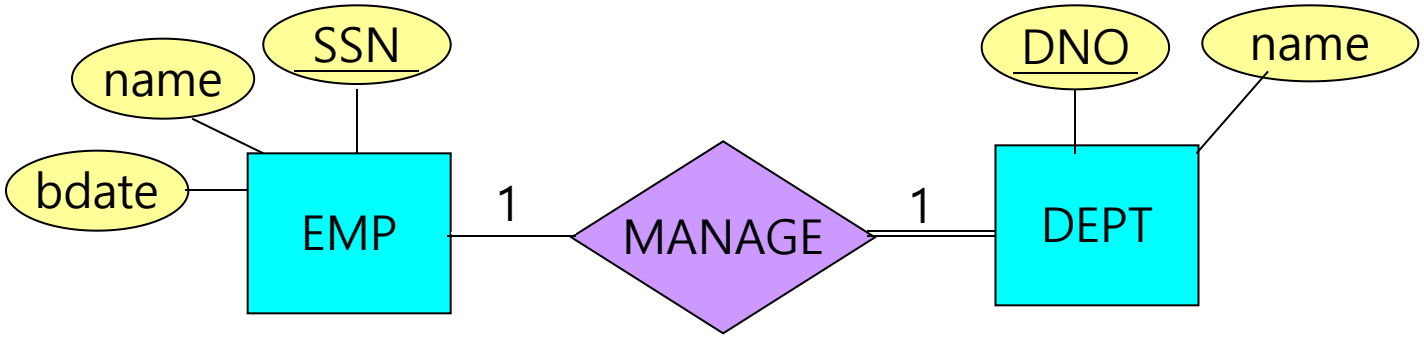
(5) **ON UPDATE SET NULL**

(6) **ON DELETE SET NULL**

: The value of FK in referencing tuples is changed to NULL value;

- Note : This **SET NULL** option is not allowed if the FK in referencing relation is a part of its own primary key.
- It is the responsibility of database designer to specify appropriate trigger actions.
- In general, **CASCADE** is useful for following "relationship" relations.
  - binary relationships (i.e., WORK-ON)
  - weak entity types (i.e., DEPENDENT)
  - multi-valued attributes (i.e., DEPT_LOCATIONS)

CREATE TABLE : Trigger



EMP (SSN, name, DNO)

DEPT (DNO, name, Mgr-SSN)

```
CREATE TABLE   DEPT
( DNO          INT              NOT NULL
  name         CHAR(10)
  Mgr-SSN      CHAR(9)          NOT NULL
  PRIMARY KEY(DNO),
  FOREIGN KEY(Mgr-SSN) REFERENCES
        EMP(SSN)
          ON DELETE    SET NULL
          ON UPDATE    CASCADE )
```

```
CREATE TABLE   EMP
( SSN   CHAR(9)      NOT NULL
  name  CHAR(10)     NOT NULL
  bdate DATE
   PRIMARY KEY (SSN)
   UNIQUE (name)      )
```

CREATE TABLE : Trigger

COURSE

CID	name
CS200	OS
CS250	DB
CS300	PL

Referenced:
PK : CID

ENROLL

CID	SID	credit
CS200	12345	3
CS200	23456	3
CS300	23456	4
CS250	23456	3
CS300	45678	3

Referencing:
FK : CID and SID
PK : (CID, SID)

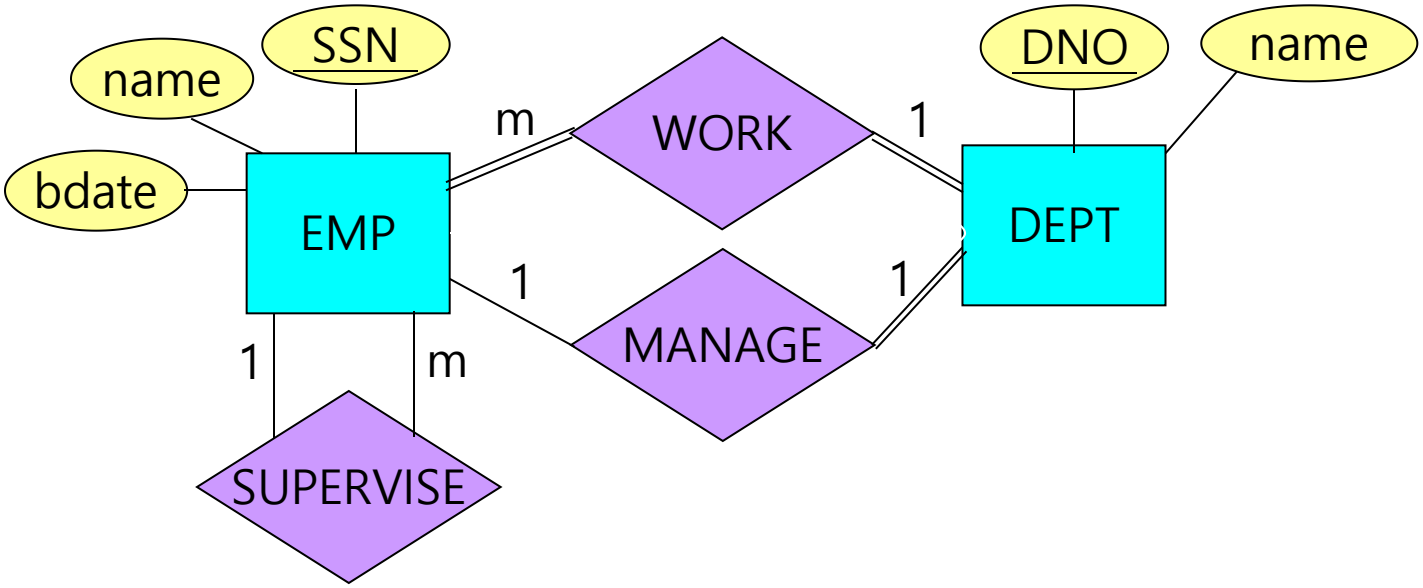
STUDENT

SID	name	age
12345	Bob	22
23456	An	18
34567	Jim	30
45678	Eve	27

Referenced:
PK : SID

- **CASCADE ON DELETE** STUDENT
- **CASCADE ON UPDATE** COURSE
- **SET NULL ON DELETE** COURSE : This is not allowed; Why??

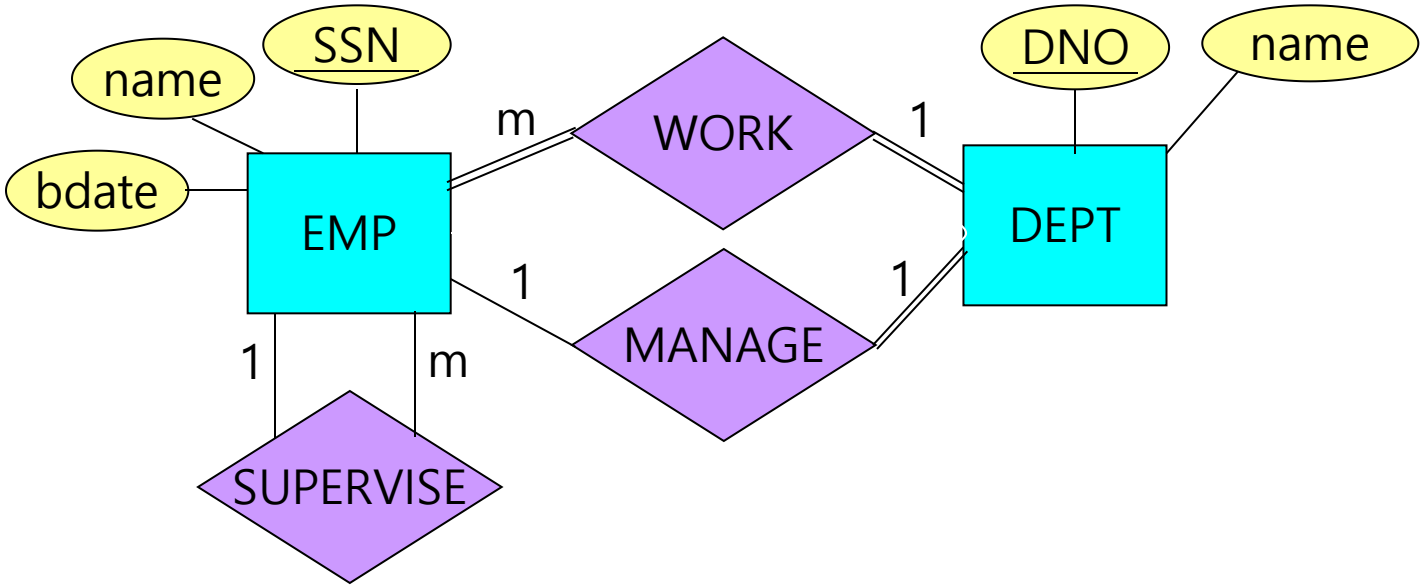
CREATE TABLE : Exercise



- Convert the above ER schema into relational schema;

- Create tables by using SQL; (Specify all constraints/triggers)

CREATE TABLE



EMP (SSN, name, bdate, DNO, Super-SSN)

DEPT (DNO, name, Mgr-SSN)

CREATE TABLE : Exercise

- **CREATE TABLE** DEPT
  (DNO INTEGER **NOT NULL**
  name VARCHAR(10)
  Mgr-SSR CHAR(9) **NOT NULL**
  PRIMARY KEY (DNO),
  UNIQUE (name),
  FOREIGN KEY (MgrSSN) **REFERENCES** EMP(SSN)
  ON DELETE SET DEFAULT
  ON UPDATE CASCADE)

  CREATE TABLE EMP
  (SSN CHAR(9) **NOT NULL**
  name CHAR(10) **NOT NULL**
  bdate DATE
  DNO INTEGER
  Super-SSN CHAR(9)
  PRIMARY KEY (SSN**)**
  FOREIGN KEY (DNO**) REFERENCES** DEPT(DNO)
  ON DELETE SET DEFAULT
  ON UPDATE CASCADE,
  FOREIGN KEY (Super-SSN**) REFERENCES** EMP(SSN)
  ON DELETE SET NULL
  ON UPDATE CASCADE)

DROP TABLE

- Used to remove a relation (base table) *and its definition*
- The dropped table can no longer be used in queries, updates, or any other commands since it does no longer exist.
- Two DROP options: **CASCADE** and **RESTRICT**
- Example:

DROP TABLE DEPENDENT **RESTRICT**

- This table is removed only if it is not referenced in any constraints or view. Otherwise, the DROP command will not be executed.
- For example, by foreign keys in another tables or views

DROP TABLE DEPENDENT **CASCADE**

- All constraints and views that reference this table are also removed automatically

ALTER TABLE

- Used to add or drop columns and change column definition to an existing table.

- Example:

ALTER TABLE EMP **ADD COLUMN** hobby **CHAR(12)**

- The new column 'hobby' will have NULLs in all tuples if no default value is specified.
- Users must enter a value for 'hobby' column by using **UPDATE** command.

ALTER TABLE EMP **DROP COLUMN** bdate **CASCADE**

- All constraints and views that 'bdate' column are also dropped automatically.
- If **RESTRICT** is used, this command is executed only if no views or constraints reference 'bdate' column.

Retrieval: Queries

- SQL has one basic statement for retrieving information from a database; the **SELECT** command.

- **SELECT** is not the same as the SELECT(σ) operation of the relational algebra

- Important difference between SQL and relational model;

  - SQL allows a table to have two or more duplicate tuples
  - Thus, an SQL relation (table) is a ***multi-set*** of tuples (= called a **bag**); it is not a set of tuples.

- SQL relations can be constrained to be **a set** by specifying:

  - **PRIMARY KEY** or **UNIQUE** attributes, or
  - **DISTINCT** option in a query

Retrieval Queries

- Basic form of the Retrieval Queries:

```
SELECT   <attribute list>
FROM     <table list>
WHERE    <condition>
```

  - <attribute list> is a list of attribute names whose values are to be retrieved by the query
  - <table list> is a list of the relation names to process query
  - <condition> is a conditional (Boolean) expression that identifies the tuples to be retrieved by the query

Figure 3.6
One possible database state for the COMPANY relational database schema.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

Simple Query : Single Table

- Retrieve the birthdate and address of employees whose sex is 'male' and whose salary is greater than 30,000.

```
SELECT    bdate, address
FROM      EMP
WHERE     (sex = 'male') AND (salary > 30000)
```

- **SELECT**-clause specifies **PROJECT**(π);

  WHERE-clause specifies **SELECT** (σ)

(참고: SQL의 SELECT는 relational algebra의 σ와 다름; π 를 의미)

- This SQL expression is equivalent to:

$$\pi_{\text{bdate, address}} (\sigma_{(\text{salary} > 30000)\ \text{AND}\ (\text{sex} = \text{'male'})} \text{EMP})$$

Optional WHERE / Use of *

- 'WHERE' is optional; That means "no condition" is applied; Thus, all tuples of the table are retrieved.

```
SELECT    SSN
FROM      EMP
```

This is equivalent to : π_{SSN} (EMP)

- To retrieve all the attribute values of the selected tuples, the symbol * is used, which means *all the attributes*

```
SELECT    *
FROM      EMP
WHERE     salary > 30000
```

This is equivalent to : $\sigma_{salary > 30000}$ (EMP)

SQL Allows Duplicated Tuples

- SQL allows duplicate tuples in a relation : **Multi-Set** of Tuples (= bag)

- Selection (σ_θ (R)) :
  If there are **m** copies of tuple **t** in **R** and **t** satisfies σ_θ, then there are **m** copies of **t** in σ_θ (**R**).

- Projection (π_A (R)) :
  For each copy of tuple **t** in **R**, there is a copy of tuple π_A (**t**) in π_A (**R**), where π_A (**t**) denotes the projection of the single tuple **t**.

- Cartesian Product (R_1 x R_2) :
  If there are **m1** copies of tuple **t1** in **R1** and **m2** copies of tuple **t2** in **R2**, there are **m1** x **m2** copies of the tuple $t_1 * t_2$ in **R1** x **R2**

DISTINCT

- SQL does not remove duplicate tuples in query result: Why?
  - Duplicates removal is expensive; (Sort and elimination cost!)
  - Users want to see duplicate tuples in the result of query.
  - When aggregate function (ie., sum, avg) is applied, we need to keep duplicated tuples.

- To remove duplicates in a query result, **DISTINCT** is used.

- Q1 may have duplicate values, but Q2 does not have any duplicates.

```
Q1 : SELECT    salary
     FROM      EMP

Q2:  SELECT    DISTINCT salary
     FROM      EMP
```

- What do think about the following query Q3?

```
Q3 : SELECT    DISTINCT SSN, salary
     FROM      EMP
```

Multiple Table Query

- If more than one tables are specified in the FROM-clause, then Cartesian Product (X) of tuples is selected.

- Retrieve all the combination of tuples from both employees and departments.

```
SELECT      *
FROM        EMP,  DEPT
```

- This is equivalent : EMP X DEPT

```
SELECT      SSN,  DNAME
FROM        EMP,  DEPT
```

- This is equivalent : $\pi_{SSN, DNAME}$ (EMP X DEPT)

Multiple Table Query

- Use a join to query data from more than one table. For example, we have a join from two tables

SELECT <attribute list>

FROM table1, table2

WHERE table1.A_i *op* table2.Bj

- Write the join condition in the WHERE clause;

  Ai and Bj : Join Attributes,

  op ∈ {=, ≠, >, ≥, <, ≤})

- Prefix the attribute name with the table name when the same attribute name appears in more than one table.

Multiple Table Query

- Retrieve SSN and age of employees who work for ‘research' dept.

```
SELECT    SSN, age
FROM      EMP, DEPT
WHERE     (DNAME = 'research') AND
          (DNO = DNUMBER)
```

- (DNO = DNUMBER) condition specifies the JOIN;

- This SQL expression is equivalent to :

(1) $\pi_{SSN,\ age}(\sigma_{(DNAME = 'research')}(EMP \underset{DNO = DNUMBER}{\bowtie} DEPT))$

or

(2) $\pi_{SSN,\ age}((EMP \underset{DNO = DNUMBER}{\bowtie} (\sigma_{(DNAME = 'research')} DEPT))$

Multiple Table Query

- For project name ‘product X', list the project number, controlling department number, and the department manager's salary.

```
SELECT    PNO, DNO, salary
FROM      PROJECT, DEPT, EMP
WHERE     (PNAME = ‘product X’)   AND
          (Mgr_SSN = SSN)    AND
          (DNUM = DNUMBER)
```

- Notice that there are *two* join conditions
  - (DNUM = DNUMBER) relates a project to its controlling department
  - (Mgr-SSN = SSN) relates the controlling department to the employee who manages that department

Rename Operation

- We can rename for both relation and attributes, taking the form;

  old name **AS** new name

- The **AS** clause can appear in both the **SELECT** and **FROM** clause.

```
SELECT    SSN, address AS addr
FROM      EMPLOYEE
WHERE     (age >= 50)
```

```
SELECT    SSN, address
FROM      EMPLOYEE AS EMP
WHERE     (age >= 50)
```

Attribute Name Conflicting

- Retrieve name and dept. phone of employees who work for each dept.

ENP(SSN, name, phone, DNO)

DEPT(DNO, name, phone)

```
SELECT    name, phone
FROM      EMP, DEPT
WHERE     DNO = DNO
```

Which name? which phone?

: Name conflicting!!

```
SELECT    EMP.name, DEPT.phone
FROM      EMP, DEPT
WHERE     EMP.DNO = DEPT.DNO
```

방식 1 :
Prefix에 Relation name
을 부쳐 구별

```
SELECT    e.name, d.phone
FROM      EMP AS e, DEPT AS d
WHERE     e.DNO = d.DNO
```

방식 2 :
Tuple variable로 구별
(뒷 장 참조!)

Use of Tuple Variables

- We can use **tuple** variables to refer any tables in SQL query;

- Retrieve name and dept. phone of employees who work for 'research' dept.

```
SELECT    e.name, d.phone
FROM      EMP AS e, DEPT AS d
WHERE     (d.name = 'research') AND
          (e.DNO = d.DNUMBER)
```

- The alternative table names e and d are called "tuple variables"

- This SQL expression is equivalent to :

```
SELECT    EMP.name, DEPT.phone
FROM      EMP, DEPT
WHERE     (DEPT.name = 'research') AND
          (EMP.DNO = DEPT.DNUMBER)
```

Use of Tuple Variables

- Tuple variables are very useful for comparing two tuples in the same relation.
- In this case, two tuple variables are given to the same relation.
- For each employee, retrieve the employee's SSN, and the name of his (or her) direct supervisor.

```
SELECT   e.SSN, s.name
FROM     EMP AS e, EMP AS s
WHERE    e.Super_SSN = s.SSN
```

- The alternative table names e and s are called "tuple variables"
- We can think of e and s as *two different copies* of EMPLOYEE;
  - e represents employees in role of 'workers'
  - s represents employees in role of 'bosses'

Use of Tuple Variables

- Retrieve SSN and names of all employees whose salary greater than Smith's salary.

```
SELECT    e.SSN, e.name
FROM      EMP AS e, EMP  AS s
WHERE     (e.salary > s.salary) AND
          (s.name = 'smith')
```

- We can also think of e and b as *two different copies* of EMPLOYEE;
  - e represents in the role of employees
  - s represents in the role of employee(s) with name = 'smith'

Use of Tuple Variables

- Retrieve the names of two employees (i.e., married) who share the same address.

```
SELECT    e1.name, e2.name
FROM      EMP AS e1, EMP  AS e2
WHERE     (e1.address = e2.address) AND
          (e1.name < e2.name)
```

- The second condition (e1.name < e2.name) means that the name of first employee precedes the name of second employee alphabetically.

- If this condition is omitted, tuple variables e1 and e2 both could refer to the same tuple. Thus, the query result produces each employee name paired with itself.

- Question: If use "<>" instead of "< ",' what will happen?

Substring Pattern Matching

- SQL provides **LIKE** comparison operator for pattern matching. We describe patterns by using two special characters.
  - Percent(%): The % matches any "substring".
  - Underscore(_): The _ matches any "character".
- Patterns are case sensitive; that is, upper case characters do not match lower case characters, or vice versa; Some examples are;
  - 'Blue%' matches any string beginning with 'Blue'.
  - '%idge% matches any string containing "idge" as substring. Examples: Blueridge, Ridgeriver, 'Rockbridge, Ridgeway, . .
  - '_ _ _' matches any string of exactly 3 characters.
  - '_ _ _%' matches any string of at least 3 characters.

Substring Pattern Matching

- Retrieve all employees whose address is in Houston, Texas.

```
SELECT    *
FROM      EMP
WHERE     Address LIKE '%Houston, Texas%'
```

- Retrieve all employees who were born during the 1970s.

```
SELECT    *
FROM      EMP
WHERE     Bdate LIKE '_ _ 7 _ _ _ _ _ _ _'
```

- Note that **NOT LIKE** also can be used in a similar way.

Arithmetic Operations

- We can use arithmetic operations in query; The standard operators +, -, *, and / can be applied to attributes with numeric data types;

- Show resulting salaries for every employee working for the research department is given 10% raise.

```
SELECT   e.SSN, 1.1 * e.salary AS increased_sal
FROM     EMP AS e  DEPT d
WHERE    (d.name = 'research') AND (DNO = d.DNUMBER)
```

- SQL provides **BETWEEN** operator to simplify comparison operators. For example, retrieve all employees whose salary is between 30,000 and 40,000. (that is, salary **>=** **AND** salary **<=** 40000)

```
SELECT    *
FROM      EMP
WHERE     salary BETWEEN 30000 AND 40000
```

Set Operators

- **UNION**, **INTERSECT**, **EXCEPT**
- Each of the above operations automatically removes duplicated tuples;
- To retain all duplicated tuples, use the corresponding operations **UNION ALL, INTERSECT ALL**, **EXCEPT ALL.**
- Suppose a tuple **t** occurs **m** times in R and **n** times in S;
  - R **UNION ALL** S : (**m + n**) times occurs.
  - R **INTERSECT ALL** S : **Min**(**m**, **n**) times occurs.
  - R **EXCEPT ALL** S : **Max**(**0**, **m – n**) times occurs.

Set Operators

- Retrieve SSN and name of employees with age > 40 or sex = 'male'.

```
SELECT    SSN, name
FROM      EMP
WHERE     age > 40
      UNION
SELECT    SSN, name
FROM      EMP
WHERE     sex = 'male'
```

- This is equivalent to :

```
SELECT    SSN, name
FROM      EMP
WHERE     (age > 40) OR (sex = 'male')
```

Set Operators

- Retrieve SSN of employees with any no dependents.

```
SELECT      SSN
FROM        EMP
      EXCEPT
SELECT      ESSN
FROM        DEPENDENT
```

Set Operators

- Retrieve all project numbers for projects that involve an employee whose name is 'Bob' as a worker, and as a manager of the department that controls the project.

```
SELECT    PNO
FROM      PROJECT, WORK-ON, EMP
WHERE     (PNUMBER = PNO) AND (ESSN = SSN)
          AND (name = 'Bob')
    INTERSECT
SELECT    PNO
FROM      PROJECT, DEPT, EMP
WHERE     (DNUM = DNUMBER) AND (Mgr_SSN = SSN)
          AND (name = 'Bob')
```

NULL Values

- NULL represents missing values; It has 3 different meanings :

  (1) Unknown (exists, but unknown):
  - Someone has 'blood-type', but currently is not known.

  (2) Unavailable (exists, but is withheld):
  - Someone has 'phone-number' but does not want it to be listed.

  (3) No applicable (undefined or non-existent):
  - 'spouse' may be NULL for unmarried someone.

- It is not possible to distinguish these 3 meanings: For example,
  Suppose many persons have "NULL value for phone-number";
  How can we interpret this?

NULL Values

- NULL values occur inevitably due to the following cases;
  - Outer Union
  - Outer Join
  - Insertion of tuples with unknown attributes

- Let X have the value NULL;

  (1) Any arithmetic operation with NULL returns "**NULL**".

  - X + 5, X – 5, X * 5, . . .

  (2) Any comparison with NULL returns "**UNKOWN**" :

  - X < 5, X = 5, X = NULL, X <> 5, . . .

- Note that "**UNKOWN**" is another truth value like TRUE and FALSE.

- SQL provides **IS NULL** (or **IS NOT NULL**) for dealing NULL values; "X IS NULL" returns "TRUE" if X has NULL value; It returns "FALSE" otherwise.

NULL Values

- SQL uses a **three**-valued logic: TRUE, FALSE, UNKNOWN

AND	T	F	U
T	T	F	U
F	F	F	F
U	U	F	U

OR	T	F	U
T	T	T	T
F	T	F	U
U	T	U	U

	NOT
T	F
F	T
U	U

- To understand how 3-valued logic works:
  Let T = 1, F = 0, U = ½,
  AND = MIN, OR = MAX, NOT(X) = 1 - X.

- Example:
  T AND (F OR NOT(U)) = MIN(1, MAX(0, (1 - ½)))
  = MIN(1, MAX(0, ½) = MIN(1, ½) = ½ (= U).

NULL Values

- Result of **WHERE** condition is treated as FALSE if it evaluates to UNKNOWN.

- Tuples that are evaluated to only TRUE are selected.
  - Thus, tuples with either FALSE or UNKNOWN values are not selected from the result.

- Example:

SSN	age	sex	salary
22222	null	null	60000

SELECT SSN
FROM EMP
WHERE (age < 50) **AND**
((sex = 'male') **OR** (salary > 50000))

  - Is this tuple selected? Answer: NO!

NULL Values

- (Funny) Example:

SSN	age	sex	salary
33333	40	male	null

```
SELECT   SSN
FROM     EMP
WHERE   (salary < 50000) OR (salary >= 50000)
```

  - Is this tuple selected? Answer: NO!

- In this case, we need to use **IS NULL** to check whether the salary attribute has NULL or not.

```
SELECT   SSN
FROM     EMP
WHERE   (salary < 50000) OR (salary >= 50000)
          OR (salary IS NULL)
```

  - Now, all tuples are selected!

Use of JOIN Operators

- SQL permits users to specify join operators in **FROM** clause; This is easier to understand than mixing together select and join conditions in **WHERE**;

```
SELECT   SSN, age
FROM     EMP JOIN DEPT ON DNO = DNUMBER
WHERE    DNAME = 'research'
```

This is equivalent to:

```
SELECT   SSN, age
FROM     EMP, DEPT
WHERE    (DNAME = 'research') AND (DNO = DNUMBER)
```

- SQL also provides various types of joins;
  - **JOIN**, **NATURAL JOIN**, **LEFT OUTER JOIN**, **RIGHT OUTER JOIN**, **CROSS JOIN**, . . .

Use of JOIN Operators

- In the case of **Natural Join**, if the name of join attributes are not the same, it is possible to rename the attributes so that they match.

- Consider the following query;

```
SELECT    Name, age
FROM      EMP NATURAL JOIN DEPARMENT
          AS DEPT(Dname, DNO, MgrSSN, Mdate)
WHERE     Dname = 'research'
```

- Not that DEPARTMENT is renamed as DEPT and Dnumber is also renamed as DNO to match the desired join attribute DNO in EMP relation.

Use of JOIN Operators

- In this join, only employees who have a superviser are included in query result;

```
SELECT    e.SSN, s.name
FROM      EMP AS e, EMP  AS s
WHERE     e.Super-SSN = s.SSN
```

- To include all employees in query result, the following outer join can be used;

```
SELECT    e.SSN, s.name
FROM      EMP AS e LEFT OUTER JOIN EMP AS s
          ON e.Super-SSN = s.SSN
```

- It is also possible to nest many join operators;

```
SELECT    SSN, age
FROM      (PROJECT JOIN WORK-ON ON PNO = PNUMBER)
          (EMP JOIN WORK-ON ON SSN = ESSN) )
WHERE     PNAME = 'laptop'
```

Nested Queries (= Subqueries)

- A query that is part of another query is called a **subquery** (or **nested query**);

- Subquery can have many nested subquries as we want;

- There are many other ways that subqury can be used;

(1) Subquery can return constant value(s), and this value can be compared with another value in WHERE clause;

(2) Subquery can return relation (a set of tuples) that can be used in various ways in WHERE clause;

(3) Subquery can have their relations appear in FROM clause;

Nested Queries

- A nested query can be specified within the **WHERE**-clause of another query, called the outer query
- Single query :

```
SELECT   SSN
FROM     EMP,  DEPT
WHERE    (DNAME = 'research') AND DNO = DNUM
```

- Nested Query : Having executed nested query returns 'research' number; Then, outer query returns SSN(s) only if its DNO is equal to result of nested query.

```
SELECT   SSN
FROM     EMP
WHERE    DNO =
    (SELECT   DNUM
     FROM     DEPT
     WHERE    DNAME = 'research')
```

Outer query : Select a EMP tuple if its DNO is equal to result of nested query

Nested query : research dept number

- Question: What if there are more than one research dept numbers?

Nested Queries : IN

- A nested query can be compared with outer query by using **IN** operator;

- (*v* **IN** *V*) compares a value *v* (in outer query) with the value(s) *V* (in nested query), and returns "true" if *v* is included in *V*. Otherwise, return "false";

```
SELECT   SSN
FROM     EMP
WHERE    DNO
```

Outer query : Select an EMP tuple if its DNO is included in result of nested query)

IN

```
SELECT   DNUM
FROM     DEPT
WHERE    DNAME = 'research'
```

Nested query : research dept number(s)

Nested Queries : IN

- SQL allows use of (sub)**tuples** of values in comparisons by placing them within parenthesis;

- Get SSNs of employees who work for the same (project, hours) on some project that employee with SSN = '123456789' works on;

```
SELECT    DISTINCT ESSN
FROM      WORK-ON
WHERE     (PNO, hours)
          IN
          SELECT    PNO, hours
          FROM      WORK-ON
          WHERE     ESSN = '123456789'
```

outer query : Select an EMP tuple if its (pno, hours) is included in result of nested query)

nested query : a set of (pno, hours) worked by '1234567'

Nested Queries : IN

- Get names of employees who work for the same (project, hours) on some project that employee with SSN = '123456789' works on;

```
SELECT   ename
FROM     EMP
WHERE    SSN
         IN
SELECT   ESSN
FROM     WORK-ON
WHERE    (PNO, hours)
         IN
SELECT   PNO, hours
FROM     WORK-ON
WHERE    ESSN = '123456789'
```

nested query : Select an EMP tuple name if its SSN is included in result of nested query

nested query : Select an EMP ESSN if its (pno, hours) is included in result of nested query)

nested query : a set of (pno, hours) worked by '1234567'

Nested Queries : NOT IN

- (*v* **NOT IN** *V*) compares a value *v* (in outer query) with the value(s) *V* (in nested query), and returns "true" if *v* is not included in *V*. Otherwise, return "false";

- Retrieve names of employees with no dependents.

```
SELECT    ename
FROM      EMP
WHERE     SSN
NOT IN
SELECT    ESSN
FROM      DEPENDENT
```

Outer query : Select an EMP tuple if its SSN is not included in result of nested query)

Nested query : SSN(s) with dependents

- Retrieve names of employees whose names are neither Smith nor Jones.

```
SELECT    ename
FROM      EMP
WHERE     ename
      NOT IN ('Smith', 'Jone')
```

Nested Queries : Use of Tuple Variables

- Nested Queries also can use tuple variables;

- Retrieve names of employees who have a dependent with the same sex as the employees.

```
SELECT    name
FROM      EMP AS e
WHERE     e.SSN
          IN
          SELECT    d.ESSN
          FROM      DEPENDENT AS d
          WHERE     e.sex = d.sex
```

outer query : Select a EMP tuple if its SSN is included in result of nested query.

nested query : Set of employee SSNs with the same sex as their dependents;

Nested Queries : SOME (= ANY)

- **SOME** (= **ANY**) can be used with {<, >, =, <=, >=, <>} operators:
  - For example, (*v* > **SOME** *V*) returns "true" if the value *v* is greater than **some** of the values in a set *V*.

- Get SSNs of employees whose salary is greater than salary of some employees in department 5:

```
SELECT   SSN
FROM     EMP
WHERE    salary
        > SOME
SELECT   salary
FROM     EMP
WHERE    DNO = 5
```

outer query : Select an EMP tuple if its salary is greater than some values in result of nested query

nested query : a set of all salaries of employees who work for DNO = 5

Example : Meaning of SOME

- Example :
  - ✓ (5 < **SOME** {0, 5, 7}) = true
  - ✓ (5 < **SOME** {1, 3, 5}) = false
  - ✓ (5 = **SOME** {0, 5, 7}) = true
  - ✓ (5 <> **SOME** {0, 5, 7}) = true
- 참조: (= **SOME**)은 **IN** 과 동일한 표현임. 반면에, (<> **SOME**)은 **NOT IN** 과 동일하지 않음.

Nested Queries : ALL

- **ALL** can be used with {<, >, =, <=, >=, <>} operators:
  - For example, (*v* **>** **ALL** *V*) returns "true" if the value *v* is greater than **all** the values in a set *V*.

- Get SSNs of employees whose salary is greater than salary of all the employees in department 5:

```
SELECT    SSN
FROM      EMP
WHERE     salary
        > ALL
          SELECT    salary
          FROM      EMP
          WHERE     DNO = 5
```

outer query : Select an EMP tuple if its salary is greater than all values in result of nested query

nested query : a set of all salaries of employees who work for DNO = 5

Example : Meaning of ALL

- Example :
  - ✓ (5 **< ALL** {0, 5, 6}) = false
  - ✓ (5 **> ALL** {0, 3, 4}) = true
  - ✓ (5 **= ALL** {4, 5, 7}) = false
  - ✓ (5 **<> ALL** {4, 6, 8}) = true

- 참조: (**<> ALL**)은 **NOT IN** 과 동일한 표현임. 반면에, (**= ALL**)은 **IN** 과 동일하지 않음.

Nested Queries : EXISTS

- **EXISTS**(Q) returns "true" if there is **at least one** tuple (= **not empty**) in the result of a nested query Q. Otherwise, it returns "false"

- Retrieve SSNs of employees who have dependent(s).

```
SELECT    SSN
FROM      EMP
WHERE
```

outer query : For each employee, if result of its related nested query is not empty, select that employee SSN.

EXISTS

```
SELECT    *
FROM      DEPENDENT
WHERE     SSN = ESSN
```

nested query : For each EMP tuple, select all dependent tuples whose ESSN matches SSN.

Nested Queries : EXISTS

- Retrieve names of employees who have a dependent with the same sex as the employees.

```
SELECT    name
FROM      EMP AS e
WHERE
    EXISTS
        ( SELECT    *
          FROM    DEPENDENT AS d
          WHERE   (e.SSN = d.ESSN)
                AND (e.sex = d.sex) )
```

outer query : For each employee tuple, if result of its related nested query is not empty, select that employee name.

nested query :Set of dependent tuples with the same sex as their employees;

Nested Queries : NOT EXISTS

- **NOT EXISTS**(Q) returns "true" if there **no tuple** (= **empty**) in the result of a nested query Q. Otherwise, it returns "false".

- Retrieve SSNs of employees who have no dependent.

```
SELECT    SSN
FROM      EMP
WHERE
```

outer query : For each employee tuple, if result of its related nested query is empty, select that employee SSN.

NOT EXISTS

```
SELECT    *
FROM      DEPENDENT
WHERE     SSN = ESSN
```

nested query : For each EMP tuple, select all dependent tuples whose ESSN matches SSN.

Nested Queries : EXISTS

- Retrieve names of employees whose sex are not the same with any dependent or with no dependent.

```
SELECT      name
FROM        EMP AS e
WHERE
    NOT EXISTS
        ( SELECT    *
          FROM      DEPENDENT AS d
          WHERE     (e.SSN = d.ESSN)
                AND (e.sex = d.sex) )
```

outer query : For each employee tuple, if result of its related nested query is not empty, select that employee name.

nested query :Set of dependent tuples with the same sex as their employees;

Nested Queries : NOT EXISTS

- Retrieve names of employees whose salary are greater than salary of all the employees with age < 25.

```
SELECT    e1.ename
FROM      EMP AS e1
WHERE
      NOT EXISTS
    SELECT      *
    FROM      EMP AS e2
    WHERE   e2.age < 25 AND
              e1.salary <= e2.salary
```

- Question: Suppose that age of all employees are >= 25; Then, what is the query result?

Division Query : NOT EXISTS and EXCEPT

- Retrieve name of each employee who work on **all** the projects controlled by department 5.

```
SELECT    name
FROM      EMP
WHERE
```

outer query : For each employee, if query 1 – query 2 is empty (that means, that employee works all projects by dept 5) then the answer is selected

NOT EXISTS

```
SELECT    Pnumber
FROM      PROJECT
WHERE     DUM = 5
```

nested query 1 : Select all project numbers controlled by dept 5

EXCEPT

```
SELECT    PNO
FROM      WORK-ON
WHERE     SSN = ESSN
```

nested query 2 : Select all project numbers performed currently by each employee

EXISTS and NOT EXISTS

- **INTERSECT** vs **EXISTS**

```
( SELECT   R.A, R.B
  FROM     R  )
    INTERSECT
( SELECT   S.A, S.B
  FROM     S)
```

=

```
SELECT   R.A, R.B
FROM     R
WHERE
   EXISTS
     ( SELECT   *
       FROM     S
       WHERE  R.A=S.A
            AND R.B=S.B )
```

- **EXCEPT** vs **NOT EXISTS**

```
( SELECT   R.A, R.B
  FROM     R  )
    EXCEPT
( SELECT   S.A, S.B
  FROM     S)
```

=

```
SELECT   R.A, R.B
FROM     R
WHERE
   NOT EXISTS
     ( SELECT   *
       FROM     S
       WHERE  R.A=S.A
            AND R.B=S.B )
```