4CCS1DBS – Database Systems

Structured Query Language (SQL)

Queries, Update, Delete

Recap: SQL from Last Week

- Data Definition Language (DDL):

CREATE SCHEMA <schema name>

CREATE TABLE (<attribute definition list>)

ALTER TABLE <table name> **ADD** <attribute definition>

DROP TABLE <table name>

DROP SCHEMA <schema name>

- Data Manipulation Language (DML):

SELECT <attribute list>
FROM <table list>
[**WHERE** <condition>]

Today

- **DML (Data Manipulation Language)** Commands
  - `SELECT` SQL Queries (continue with `SELECT`)
    - Review Set Operations
    - Math Expressions
    - Casting Types
  - `INSERT` Data
  - `UPDATE` Data
  - `DELETE` Data
  - Nested Queries
  - Types of Joins
  - Grouping and Aggregation Functions
- Assertions and Views (DDL Data Definition Language)

COMPANY Relational Database Schema

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

DEPT_LOCATIONS

Dnumber	Dlocation

PROJECT

Pname	Pnumber	Plocation	Dnum

WORKS_ON

Essn	Pno	Hours

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship

Figure 5.5
Schema diagram for the COMPANY relational database schema.

COMPANY Populated Database

EMPLOYEE	FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	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	1968-07-19	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

DEPT_LOCATIONS	DNUMBER	DLOCATION
	1	Houston
	4	Stafford
	5	Bellaire
	5	Sugarland
	5	Houston

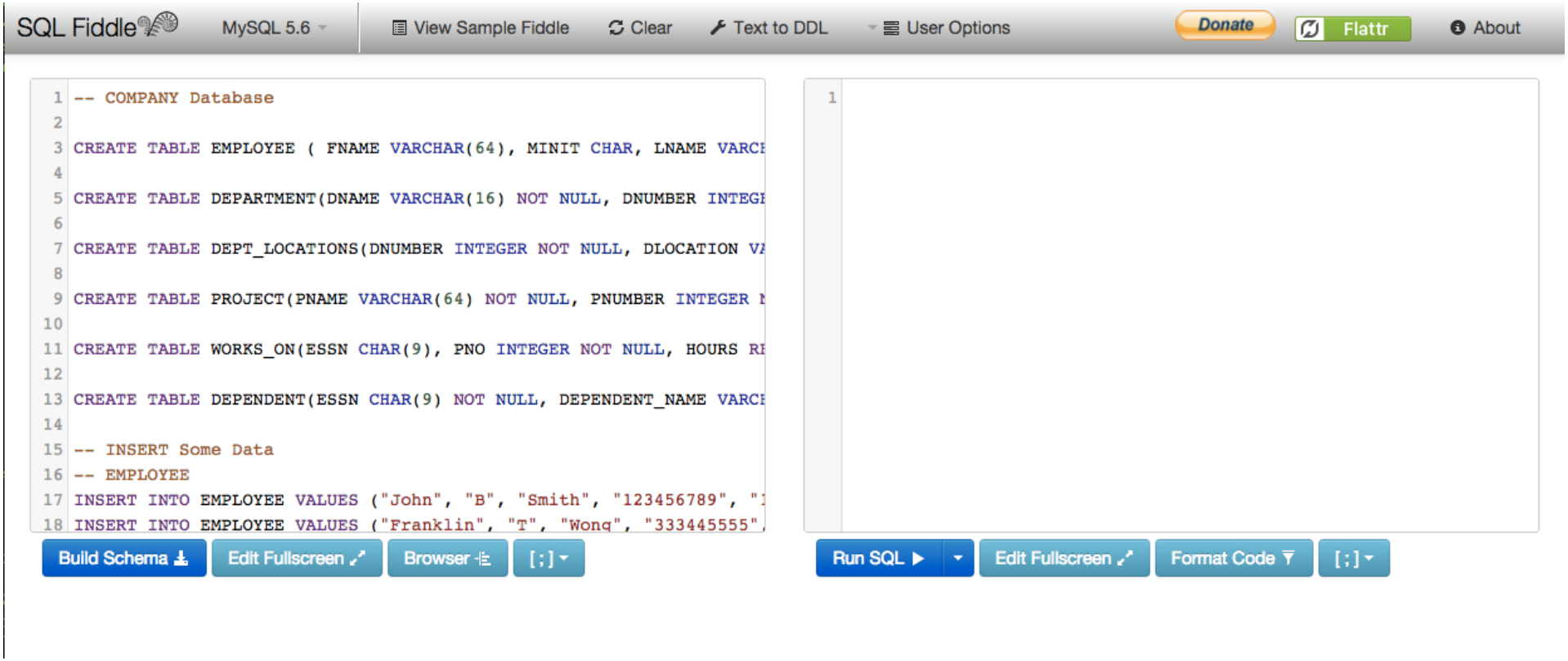
DEPARTMENT	DNAME	DNUMBER	MGRSSN	MGRSTARTDATE
	Research	5	333445555	1988-05-22
	Administration	4	987654321	1995-01-01
	Headquarters	1	888665555	1981-06-19

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

No NEED to RUN DB at HOME …

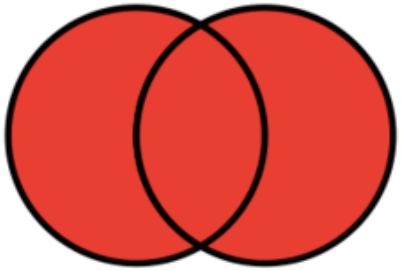


http://sqlfiddle.com/#!9/e02e29

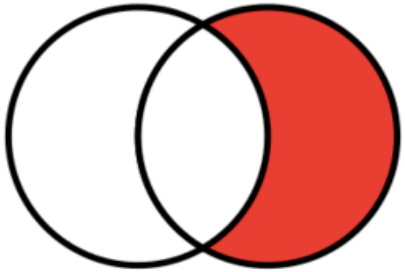
Review: Set Operations

- SQL has directly incorporated some set operations

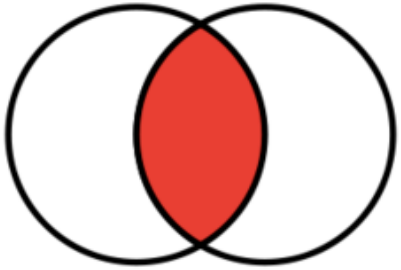
UNION
(set union)



MINUS/EXCEPT
(set difference)



INTERSECTION
(set intersection)



- Resulting relations of these set operations are sets of tuples — *duplicate tuples are eliminated from the result*
- Set operations apply only to *union compatible relations:*

1. Two relations must have the same attributes (names)

2. Each corresponding pair of attributes has the same domain.

EXERCISE: Set Operations — UNION

Make a list of all project numbers for projects that involve an employee whose last name is 'Smith' as a worker or as a manager of the department that controls the project. Which rows are retrieved?

```
(SELECT    PNUMBER
 FROM      PROJECT, DEPARTMENT, EMPLOYEE
 WHERE     DNUM=DNUMBER AND MGRSSN=SSN
           AND LNAME='Smith')
UNION
(SELECT    PNUMBER
 FROM      PROJECT, WORKS_ON, EMPLOYEE
 WHERE     PNUMBER=PNO AND ESSN=SSN
                AND LNAME='Smith')
```

EXERCISE: Set Operations — UNION

Make a list of all project numbers for projects that involve an employee whose last name is 'Smith' as a worker or as a manager of the department that controls the project. Which rows are retrieved?

```
(SELECT   PNUMBER
 FROM     PROJECT, DEPARTMENT, EMPLOYEE
 WHERE    DNUM=DNUMBER AND MGRSSN=SSN
          AND LNAME='Smith')
UNION
(SELECT   PNUMBER
 FROM     PROJECT, WORKS_ON, EMPLOYEE
 WHERE    PNUMBER=PNO AND ESSN=SSN
               AND LNAME='Smith')
```

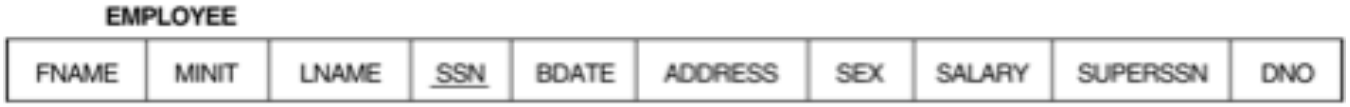
empty

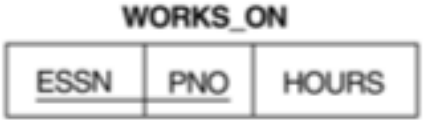
PNUMBER

1
2

Set Operations — EXCEPT (MINUS)

Example: List SSNs from all employees except those who are working on Project 1.





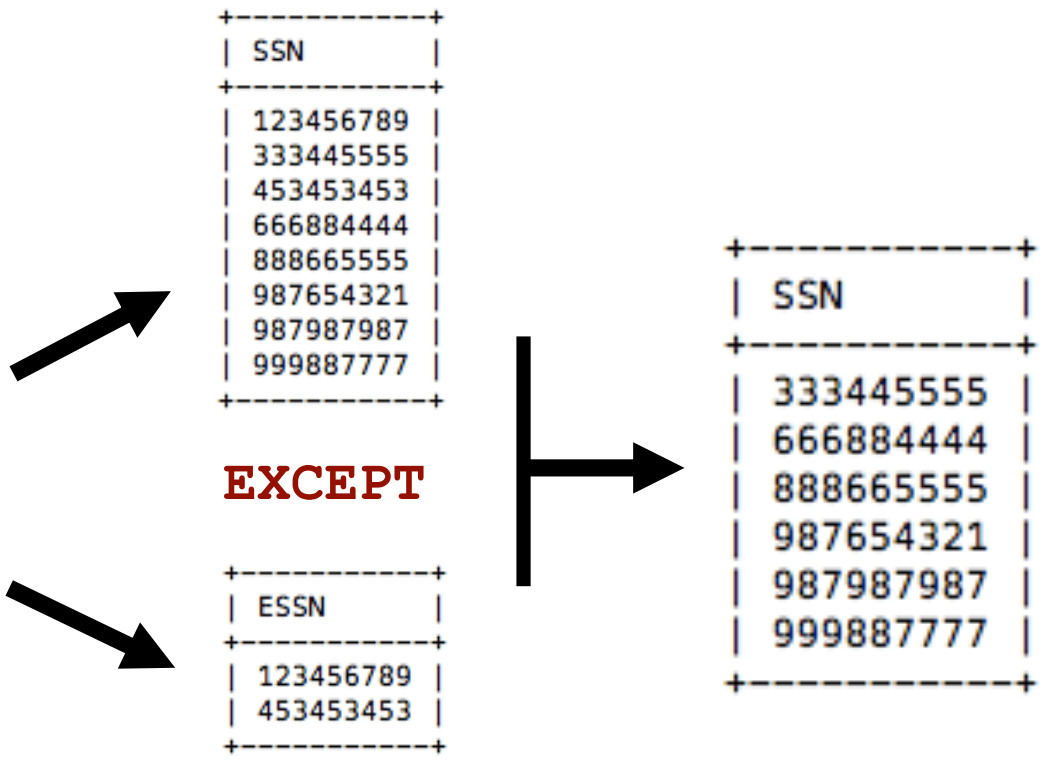

```
(SELECT    SSN
 FROM      EMPLOYEE)
EXCEPT
(SELECT    ESSN as SSN
 FROM      WORKS_ON
 WHERE     PNO=1)
```

selecting from WORKS_ON, so ESSN as SSN required

Set Operations — EXCEPT (MINUS)

Example: List SSNs from all employees except those who are working on Project 1.

```
(SELECT    SSN
 FROM      EMPLOYEE)
EXCEPT
(SELECT    ESSN as SSN
 FROM      WORKS_ON
 WHERE     PNO=1)
```



Arithmetic Operations

- The standard arithmetic operators **'+', '-'. '*', and '/'** (for addition, subtraction, multiplication, and division, respectively) can be applied to numeric values in an SQL query result

Show the effect of giving all employees who work on the 'ProductX' project a 10% raise.

```
SELECT FNAME, LNAME, 1.1*SALARY
FROM   EMPLOYEE, WORKS_ON, PROJECT
WHERE  SSN=ESSN AND PNO=PNUMBER AND
       PNAME='ProductX'
```

More on Arithmetic Expressions and IF() function

- Constants are allowed
- Note the use of **AS** to alias the results as an attribute
- **IF**(<condition>, <True Value>, <False Value>)
- **IF** with more than 2 values? See **CASE()**, or nested IF statements

```
SELECT FNAME, LNAME,(SALARY / 1000) AS SALARY_K, 1 AS ONE,
    IF(SALARY > 30000, True, False) AS IS_LOADED,
    IF(SUPERSSN IS NULL, "Boss", "Worker") AS ETYPE
FROM EMPLOYEE
```

FNAME	LNAME	SALARY_K	ONE	IS_LOADED	ETYPE
John	Smith	30.0000	1	0	Worker
Franklin	Wong	40.0000	1	1	Worker
Joyce	English	25.0000	1	0	Worker
Ramesh	Narayan	38.0000	1	1	Worker
James	Borg	55.0000	1	1	Boss
Jennifer	Wallace	43.0000	1	1	Worker
Ahmad	Jabbar	25.0000	1	0	Worker
Alicia	Zelaya	25.0000	1	0	Worker

Use of CAST()

- Convert the Data Type of an attribute using **CAST()**:
- **CAST**(<expression> **AS** <type>):
  - BINARY[(*N*)]
  - CHAR[(*N*)]
  - DATE
  - DATETIME
  - DECIMAL[(*M*[,*D*])]
  - SIGNED [INTEGER]
  - TIME
  - UNSIGNED [INTEGER]

```
SELECT FNAME, LNAME,
    CAST((SALARY / 1000) AS UNSIGNED) AS SALARY_K
FROM EMPLOYEE
```

fname	lname	SALARY_K
John	Smith	30
Franklin	Wong	40
Joyce	English	25
Ramesh	Narayan	38
James	Borg	55
Jennifer	Wallace	43
Ahmad	Jabber	25
Alicia	Zelaya	25

Manipulating Data in SQL

- Three SQL commands to modify the STATE of a database (part of the DML for SQL)
    1. **INSERT**
    2. **DELETE**
    3. **UPDATE**

- *They do not modify the SCHEMA of the database*
    - *What commands are used in this case?*
- *Note that **SELECT** is widely considered part of the DML because it clearly is not a DDL command.*

INSERT

- In its simplest form, it is used to add one or more tuples to a relation

```
INSERT INTO     <table name>
VALUES          <tuple>
```

- Attribute values should be listed in *the same order* as the attributes were specified in the **CREATE TABLE** command

```
INSERT INTO  EMPLOYEE
VALUES ('Richard','K','Marini’,'653298653’,
        '30-DEC-52’,'98 Oak Forest,Katy,TX’,
        'M', 37000,'987654321', 4)
```

INSERT (Specify Values)

- Alternate form of INSERT specifies explicitly the attribute names that correspond to the values in the new tuple

- *Left out attributes will be default value or NULL*

```
INSERT INTO     <table name> (<attribute list>)
VALUES          <tuple>
```

Example: Insert a tuple for a new EMPLOYEE for whom we only know the FNAME, LNAME, and SSN attributes.

```
INSERT INTO EMPLOYEE (FNAME, LNAME, SSN)
VALUES ('Richard', 'Marini', '653298653')
```

INSERT Multiple Values from a CREATE

- Another variation of INSERT allows insertion of *multiple tuples* resulting from a query into a relation

Example: We want to create a temporary table that has the employee last name, project name, and hours per week for each employee working on a project.

- *First create a table, WORKS_ON_INFO*

```
CREATE TABLE WORKS_ON_INFO (
  EMP_NAME              VARCHAR(15),
  PROJ_NAME             VARCHAR(15),
  HOURS_PER_WEEK        DECIMAL(3,1)
);
```

INSERT Multiple Values from a CREATE

Example: We want to create a temporary table that has the employee last name, project name, and hours per week for each employee working on a project.

- *Then load WORKS_ON_INFO with the results of a joined query:*

```
INSERT INTO WORKS_ON_INFO(EMP_NAME, PROJ_NAME, HOURS_PER_WEEK)
SELECT      E.Lname, P.Pname, W.Hours
FROM        PROJECT P, WORKS_ON W, EMPLOYEE E
WHERE       P.Pnumber = W.Pno AND W.Essn = E.Ssn;
```

INSERT Multiple Values from a CREATE

Example: We want to create a temporary table that has the employee last name, project name, and hours per week for each employee working on a project.

- *Then load WORKS_ON_INFO with the results of a joined query:*

```
INSERT INTO WORKS_ON_INFO (EMP_NAME, PROJ_NAME, HOURS_PER_WEEK)
SELECT      E.Lname, P.Pname, W.Hours
FROM        PROJECT P, WORKS_ON W, EMPLOYEE E
WHERE       P.Pnumber = W.Pno AND W.Essn = E.Ssn;
```

- *Values are mapped to Attributes in the Order they appear*

Using CREATE TABLE … AS

- It is also possible to do the previous two queries in one CREATE TABLE command (note that **AS** is used in two different ways in the following query):

```
CREATE TABLE    WORKS_ON_INFO AS
SELECT              E.Lname AS Emp_Name,
                    P.Pname AS Proj_Name,
                W.Hours AS Hours_per_week
FROM            PROJECT P, WORKS_ON W, EMPLOYEE E
WHERE           P.Pnumber=W.Pno AND W.Essn = E.Ssn;
```

new attributes: Emp_Name, Proj_Name, Hours_per_week

- *Note the use of the keyword **AS** to specify table/attribute names*

DELETE

- Removes tuples from a relation

```
DELETE FROM     <table name>
WHERE           <condition>
```

- Includes a WHERE-clause to select the tuples to be deleted
- Referential integrity should be enforced
- Tuples are deleted from only ***one table*** at a time (unless CASCADE is specified on a referential integrity constraint)
- A missing WHERE-clause specifies that ***all tuples*** in the relation are to be deleted; the table then becomes an empty table
- The number of tuples deleted depends on the number of tuples in the relation that satisfy the WHERE-clause

EXERCISE: DELETE

Which tuples do these queries delete?

- **Example 1**

```
DELETE FROM EMPLOYEE
WHERE LNAME='Brown';
```

- **Example 2**

```
DELETE FROM EMPLOYEE
WHERE SSN='123456789';
```

- **Example 3**

```
DELETE FROM      EMPLOYEE
WHERE DNO = 5;
```

- **Example 4**

```
DELETE FROM      EMPLOYEE;
```

UPDATE

- Used to modify attribute values of one or more selected tuples

```
UPDATE      <table name>
SET         <attribute>=<value>,…
WHERE       <condition>
```

- A WHERE-clause selects the tuples to be modified
- An additional SET-clause specifies the attributes to be modified and their new values
- Each command modifies tuples *in the same relation*
- Referential integrity should be enforced

EXERCISE: Update

Example: Change the location and controlling department number of project number 10 to 'Bellaire' and 5, respectively.

```
UPDATE      PROJECT
SET         PLOCATION = 'Bellaire',
            DNUM = 5
WHERE       PNUMBER=10;
```

How many rows are updated in this UPDATE in the Project table?

EXERCISE: Update

Example: Change the location and controlling department number of project number 10 to 'Bellaire' and 5, respectively.

```
UPDATE      PROJECT
SET         PLOCATION = 'Bellaire',
            DNUM = 5
WHERE       PNUMBER=10;
```

How many rows are updated in this UPDATE in the Project table?

1 Tuple

EXERCISE: Update

Example: Change the location and controlling department number of project number 10 to 'Bellaire' and 5, respectively. – What if WHERE forgotten?

```
UPDATE      PROJECT
SET         PLOCATION = 'Bellaire',
            DNUM = 5;
```

How many rows are updated in this UPDATE in the Project table?

UPDATE Example 2

Example: Give all employees in the 'Research' department a 10% raise in salary.

```
UPDATE      EMPLOYEE
SET         SALARY = SALARY*1.1
WHERE       DNO IN (SELECT    DNUMBER
                    FROM   DEPARTMENT
                    WHERE  DNAME='Research');
```

- Math Expression: the modified SALARY value depends on the original SALARY value in each tuple.
  - The reference to the SALARY attribute on the ***right*** of = refers to the ***old*** SALARY value before modification
  - The reference to the SALARY attribute on the ***left*** of = refers to the ***new*** SALARY value after modification

Nesting of Queries

- A complete SELECT query, called a *nested query*, can be specified within the WHERE-clause of another query, called the *outer query*
  - Many of the previous queries can be specified in an alternative form using nesting

Example: Retrieve the name and address of all employees who work for the 'Research' department.

```
SELECT      FNAME, LNAME, ADDRESS
FROM        EMPLOYEE, DEPARTMENT
WHERE       DNAME='Research' AND DNUMBER=DNO
```

Nesting of Queries — Equivalent Query using IN

- Using the comparison operator **IN:**

Compares a value *v* with a set of values *V* and returns TRUE if *v* is one of the elements in *V*.

Example: Retrieve the name and address of all employees who work for the 'Research' department.

```
SELECT      FNAME, LNAME, ADDRESS
FROM        EMPLOYEE
WHERE       DNO IN (SELECT DNUMBER
                    FROM DEPARTMENT
                    WHERE DNAME='Research');
```

Nesting of Queries — Explanation of IN

- The nested query selects the number of the 'Research' department
- The outer query select an EMPLOYEE tuple if its DNO value is in the result of either nested query
- **The comparison operator IN compares a value v with a set (or multi-set) of values V, and evaluates to TRUE if v is one of the elements in V**
- In general, we can have several levels of nested queries
- A reference to an *unqualified attribute* refers to the relation declared in the *innermost nested query*
- In previous example, the nested query is *not correlated* with the outer query

Correlated Nested Queries

- If a condition in the WHERE-clause of a *nested query* references an attribute of a relation declared in the *outer query*, the two queries are said to be ***correlated***
- The result of a correlated nested query is ***different*** for each tuple (or combination of tuples) of the relation(s) in the outer query.

Example: Retrieve the name of each employee who has a dependent with the same first name as the employee.

```
SELECT    E.FNAME, E.LNAME
FROM      EMPLOYEE AS E
WHERE     E.SSN IN  (SELECT ESSN
                     FROM DEPENDENT
                     WHERE ESSN=E.SSN AND
                     E.FNAME=DEPENDENT_NAME)
```

ESSN depends on E.FNAME

Correlated Nested Queries - Re-written

- A query written with nested SELECT... FROM... WHERE... blocks and using the = or IN comparison operators can ***always*** be expressed as a single block query.

```
SELECT      E.FNAME, E.LNAME
FROM        EMPLOYEE AS E
WHERE       E.SSN IN (SELECT    ESSN
                      FROM      DEPENDENT
                      WHERE     ESSN=E.SSN AND
                                E.FNAME=DEPENDENT_NAME)
```

is re-written as a single block query:

```
SELECT      E.FNAME, E.LNAME
FROM        EMPLOYEE E, DEPENDENT D
WHERE       E.SSN=D.ESSN AND
            E.FNAME=D.DEPENDENT_NAME
```

The EXISTS Function

- EXISTS is used to check whether the result of a correlated nested query is empty (contains no tuples) or not

Example: Retrieve the name of each employee who has a dependent with the same first name as the employee.

```
SELECT      FNAME, LNAME
FROM        EMPLOYEE E
WHERE       EXISTS (SELECT *
                    FROM DEPENDENT
                    WHERE     E.SSN=ESSN AND
                              E.FNAME=DEPENDENT_NAME)
```

Again re-written in a different form…

The EXISTS Function - NOT EXISTS

- NOT EXISTS is TRUE if there are NO tuples as a result of the query.

Example: Retrieve the names of employees who have no dependents.

```
SELECT      FNAME, LNAME
FROM        EMPLOYEE E
WHERE       NOT EXISTS(SELECT *
                       FROM   DEPENDENT
                       WHERE  E.SSN=ESSN)
```

ALL comparison operator

- Comparison operators to compare a ***single*** value (as an attribute) to a ***set*** or ***multiset*** (a nested query)

Example: Retrieve the names of employees whose salary is greater than the salary of all employees in department 5.

```
SELECT      LNAME, FNAME
FROM        EMPLOYEE
WHERE       SALARY > ALL(SELECT SALARY
                         FROM  EMPLOYEE
                         WHERE DNO=5)
```

NULLs in SQL Queries

- SQL allows queries that check if a value is **NULL** (missing or undefined or not applicable)
- SQL uses **IS** or **IS NOT** to compare NULLs because it considers each NULL value distinct from other NULL values, so *equality comparison is not appropriate*
- In join conditions, tuples with NULL values in these attributes are not included in result (i.e. DNUMBER = DNO, and both are NULL)

Example: Retrieve the names of all employees who do not have supervisors

```
SELECT      FNAME, LNAME
FROM        EMPLOYEE
WHERE       SUPERSSN IS NULL
```

Joined Relations — Using JOIN

- Using the JOIN keyword can specify “joined relations”
- Two joined relations look like any other relation
- Many types:
  - **JOIN** (regular “theta” join as you will see in Rel. Alg.)
  - **NATURAL JOIN**
  - **LEFT OUTER JOIN, LEFT JOIN**
  - **RIGHT OUTER JOIN, RIGHT JOIN**
  - **FULL OUTER JOIN, OUTER JOIN**
  - **INNER JOIN**
  - **CROSS JOIN**

Joined Relations — JOIN … ON

- SELECT … with a JOIN Condition in the WHERE clause:

```
SELECT DLOCATION, MGRSSN
FROM DEPARTMENT, DEPT_LOCATIONS
WHERE DNAME='Research' AND
DEPARTMENT.DNUMBER=DEPT_LOCATIONS.DNUMBER;
```

- Using JOIN … ON as an "*equi-join*"

```
SELECT DLOCATION, MGRSSN
FROM DEPARTMENT JOIN DEPT_LOCATIONS ON
DEPARTMENT.DNUMBER=DEPT_LOCATIONS.DNUMBER
WHERE  DNAME='Research';
```

- NATURAL JOIN

```
SELECT DLOCATION, MGRSSN
FROM DEPARTMENT NATURAL JOIN DEPT_LOCATIONS
WHERE  DNAME='Research';
```

Distinguishing between JOIN Functions

- **NATURAL JOIN** (same as JOIN):
  - *No join condition may be specified, implicit condition to join on attributes with the same name*
- **INNER JOIN**:
  - *Tuple is included in the result only if a matching tuple exists in the other relation (default type of JOIN)*
- **OUTER JOIN**:
  - *all matching tuples are returned (depending on type of OUTER JOIN):*
    - LEFT OUTER JOIN
    - RIGHT OUTER JOIN
    - FULL OUTER JOIN

EXERCISE: LEFT OUTER JOIN

- Previous query:

```
SELECT      E.FNAME, E.LNAME, S.FNAME, S.LNAME
FROM        EMPLOYEE E, EMPLOYEE S
WHERE       E.SUPERSSN=S.SSN
```

- How does changing it to a **LEFT OUTER JOIN** modify the results?

```
SELECT      E.FNAME, E.LNAME, S.FNAME, S.LNAME
FROM        (EMPLOYEE E LEFT OUTER JOIN
                        EMPLOYEE AS S
                        ON E.SUPERSSN=S.SSN)
```

OUTER JOIN Examples

- Previous query:

```
SELECT      E.FNAME, E.LNAME, S.FNAME, S.LNAME
FROM        EMPLOYEE E, EMPLOYEE S
WHERE       E.SUPERSSN=S.SSN
```

- How does changing it to a **LEFT OUTER JOIN** modify the results?

```
SELECT      E.FNAME, E.LNAME, S.FNAME, S.LNAME
FROM        (EMPLOYEE E LEFT OUTER JOIN
                        EMPLOYEE AS S
                        ON E.SUPERSSN=S.SSN)
```

Same as above but include this row… (Employees without supervisors)

James	Borg	NULL	NULL

EXERCISE: RIGHT OUTER JOIN

- Previous query:

```
SELECT      E.FNAME, E.LNAME, S.FNAME, S.LNAME
FROM        EMPLOYEE E, EMPLOYEE S
WHERE       E.SUPERSSN=S.SSN
```

- How does changing it to a **RIGHT OUTER JOIN** modify the results?

```
SELECT      E.FNAME, E.LNAME, S.FNAME, S.LNAME
FROM        (EMPLOYEE E RIGHT OUTER JOIN
                        EMPLOYEE AS S
                        ON E.SUPERSSN=S.SSN)
```

OUTER JOIN Examples

- Previous query:

```
SELECT      E.FNAME, E.LNAME, S.FNAME, S.LNAME
FROM        EMPLOYEE E S
WHERE       E.SUPERSSN=S.SSN
```

- How does changing it to a **RIGHT OUTER JOIN** modify the results?

```
SELECT      E.FNAME, E.LNAME, S.FNAME, S.LNAME
FROM        (EMPLOYEE E RIGHT OUTER JOIN
                        EMPLOYEE AS S
                        ON E.SUPERSSN=S.SSN)
```

Same as above but include non-matching tuples from RIGHT table…(Employees who are not supervising anyone)

NULL	NULL	John	Smith
…			

EXERCISE: FULL OUTER JOIN

- Previous query:

```
SELECT      E.FNAME, E.LNAME, S.FNAME, S.LNAME
FROM        EMPLOYEE E S
WHERE       E.SUPERSSN=S.SSN
```

- How does changing it to a **FULL OUTER JOIN** modify the results?

```
SELECT      E.FNAME, E.LNAME, S.FNAME, S.LNAME
FROM        (EMPLOYEE E FULL OUTER JOIN
                        EMPLOYEE AS S
                        ON E.SUPERSSN=S.SSN)
```

OUTER JOIN Examples

- Previous query:

```
SELECT      E.FNAME, E.LNAME, S.FNAME, S.LNAME
FROM        EMPLOYEE E S
WHERE       E.SUPERSSN=S.SSN
```

- How does changing it to a **FULL OUTER JOIN** modify the results?

```
SELECT      E.FNAME, E.LNAME, S.FNAME, S.LNAME
FROM        (EMPLOYEE E FULL OUTER JOIN
                        EMPLOYEE AS S
                        ON E.SUPERSSN=S.SSN)
```

Include both LEFT and RIGHT table's non-matching tuples
(Employees without supervisors and employees not being supervisors)

James	Bord	NULL	NULL
NULL	NULL	Joyce	English
...			

CROSS JOIN — (Cartesian Product)

- What does this query return?

```
SELECT      E.FNAME, E.LNAME, S.FNAME, S.LNAME
FROM        (EMPLOYEE E CROSS JOIN EMPLOYEE S)
```

- Equivalent to:

```
SELECT      E.FNAME, E.LNAME, S.FNAME, S.LNAME
FROM        EMPLOYEE E,EMPLOYEE S
```

- As well as:

```
SELECT      E.FNAME, E.LNAME, S.FNAME, S.LNAME
FROM        EMPLOYEE E JOIN EMPLOYEE S
```

Multiway JOINs

Example: For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name, address, and birthdate.

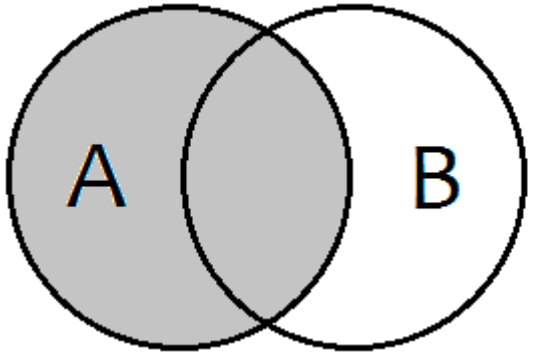
```
SELECT      PNUMBER, DNUM, LNAME, ADDRESS, BDATE
FROM            PROJECT, DEPARTMENT, EMPLOYEE
WHERE       DNUM=DNUMBER AND MGRSSN=SSN
                         AND PLOCATION='Stafford'
```

- Is equivalent to specifying a *multiway* join:

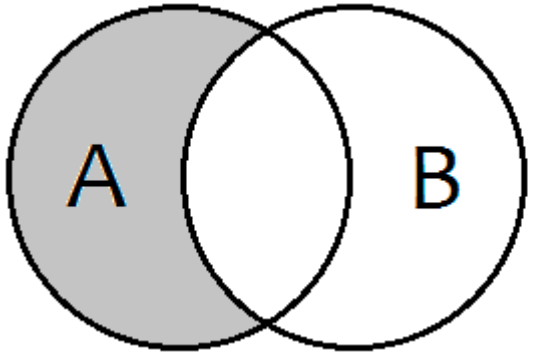
```
SELECT      PNUMBER, DNUM, LNAME, ADDRESS,BDATE
FROM        ((PROJECT JOIN DEPARTMENT ON  DNUM=DNUMBER)
                      JOIN EMPLOYEE ON  MGR_SSN=SSN)
WHERE PLOCATION='Stafford'
```

SQL JOINS

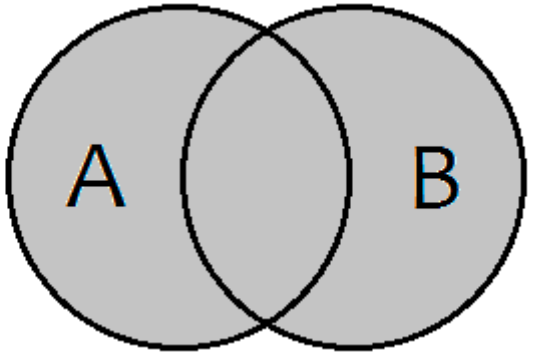
Source: jpromero.com



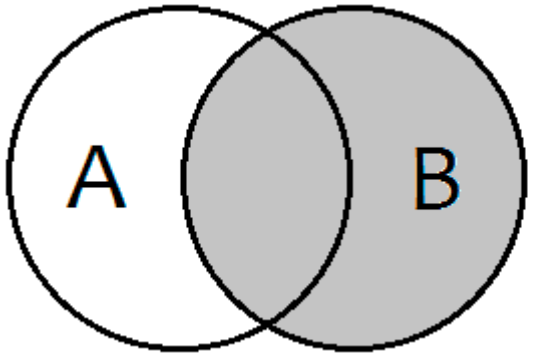

```
SELECT *
FROM TableA a
LEFT JOIN TableB b
ON a.Key = b.Key
```



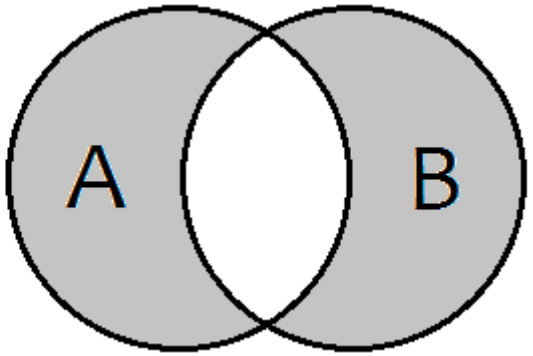

```
SELECT *
FROM TableA a
LEFT JOIN TableB b
ON a.Key = b.Key
WHERE b.Key IS NULL
```



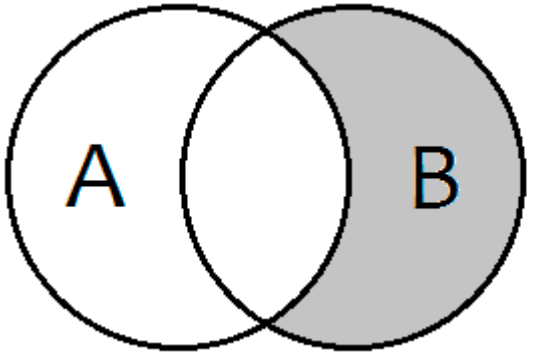

```
SELECT *
FROM TableA a
FULL OUTER JOIN TableB b
ON a.Key = b.Key
```



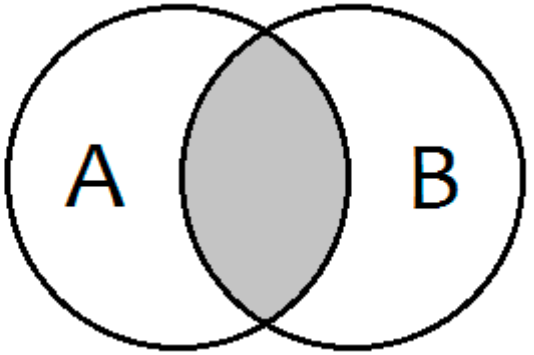

```
SELECT *
FROM TableA a
RIGHT JOIN TableB b
ON a.Key = b.Key
```




```
SELECT *
FROM TableA a
FULL OUTER JOIN TableB b
ON a.Key = b.Key
WHERE a.Key IS NULL
OR b.Key IS NULL
```




```
SELECT *
FROM TableA a
RIGHT JOIN TableB b
ON a.Key = b.Key
WHERE a.Key IS NULL
```




```
SELECT *
FROM TableA a
INNER JOIN TableB b
ON a.Key = b.Key
```

Aggregate Functions

Example: Find the maximum salary, the minimum salary, and the average salary among all employees.

```
SELECT      MAX(SALARY), MIN(SALARY), AVG(SALARY)
FROM        EMPLOYEE
```

```
+-------------+-------------+-------------+
| max(salary) | min(salary) | avg(salary) |
+-------------+-------------+-------------+
|       55000 |       25000 |  35125.0000 |
+-------------+-------------+-------------+
```

- Functions include **COUNT**, **SUM**, **MAX**, **MIN**, and **AVG**
- Some SQL implementations may not allow more than one function in the SELECT-clause

Aggregate Functions

Example: Find the maximum salary, the minimum salary, and the average salary among employees who work for the 'Research' department.

```
SELECT      MAX(SALARY), MIN(SALARY), AVG(SALARY)
FROM        EMPLOYEE, DEPARTMENT
WHERE       DNO=DNUMBER AND DNAME='Research'
```

Aggregate Functions - COUNT

Example: Retrieve the total number of employees in the company (Q1), and the number of employees in the 'Research' department (Q2).

```
Q1:    SELECT      COUNT (*)
       FROM        EMPLOYEE;

Q2:    SELECT      COUNT (*)
       FROM        EMPLOYEE, DEPARTMENT
                   WHERE DNO=DNUMBER AND DNAME='Research'
```

Aggregate Functions - COUNT

Example: Select the names of the all employees who have two or more dependents.

```
SELECT      LNAME, FNAME
FROM        EMPLOYEE
WHERE       (SELECT COUNT(*)
             FROM DEPENDENT
             WHERE  SSN=ESS) >= 2;
```

Note that when the result is one attribute and one tuple

*– that becomes a **SCALAR**.*

Aggregate Functions - COUNT

Example: Count the number of distinct salary values in the Employees table;

```
SELECT      COUNT (DISTINCT SALARY)
FROM        EMPLOYEE
```

Note that NULL values are not counted as part of the aggregate

EXERCISE: COUNT vs. COUNT(*)

What is the difference between these 2 queries?

```
SELECT      COUNT (SUPERSSN)
FROM        EMPLOYEE;
```

```
SELECT      COUNT (*)
FROM        EMPLOYEE;
```

Aggregate Functions - COUNT vs. COUNT(*)

What is the difference between these 2 queries?

```
SELECT      COUNT (SUPERSSN)
FROM              EMPLOYEE
```

7

```
SELECT      COUNT (*)
FROM              EMPLOYEE
```

8

COUNT(*) just counts rows

(rows with NULL values are also counted)

Grouping

- In many cases, we want to apply the aggregate functions to *subgroups of tuples* in a relation
- Each subgroup of tuples consists of the set of tuples that have the *same value* for the *grouping attribute(s)*
- The function is applied to each subgroup independently
- SQL has a **GROUP BY**-clause for specifying the grouping attributes, which *must also appear in the SELECT-clause*

```
SELECT      <attribute list, include grouping attributes>
FROM        <table list>
[WHERE      <condition>]
GROUP BY    <grouping attributes>
```

Grouping with Aggregate Functions

Example: For each department, retrieve the department number, the number of employees in the department, and their average salary.

```
SELECT      DNO, COUNT (*), AVG (SALARY)
FROM        EMPLOYEE
GROUP BY    DNO
```

- EMPLOYEE tuples are divided into groups - Each group having the same value for the grouping attribute **DNO**
- The COUNT and AVG functions are applied to each such group of tuples separately
- The SELECT-clause includes only the grouping attribute and the functions to be applied on each group of tuples
- A join condition can be used in conjunction with grouping

Grouping with Aggregate Functions

Example: For each department, retrieve the department number, the number of employees in the department, and their average salary.

```
SELECT      DNO, COUNT (*), AVG (SALARY)
FROM        EMPLOYEE
GROUP BY    DNO
```

Fname	Minit	Lname	Ssn	...	Salary	Super_ssn	Dno
John	B	Smith	123456789		30000	333445555	5
Franklin	T	Wong	333445555		40000	888665555	5
Ramesh	K	Narayan	666884444		38000	333445555	5
Joyce	A	English	453453453	...	25000	333445555	5
Alicia	J	Zelaya	999887777		25000	987654321	4
Jennifer	S	Wallace	987654321		43000	888665555	4
Ahmad	V	Jabbar	987987987		25000	987654321	4
James	E	Bong	888665555		55000	NULL	1

Dno	Count (*)	Avg (Salary)
5	4	33250
4	3	31000
1	1	55000

Result of Q24

Grouping with Aggregate Functions

Example: For each project, retrieve the project number, project name, and the number of employees who work on that project.

```
SELECT      PNAME, PNUMBER, COUNT(*)
FROM              PROJECT, WORKS_ON
WHERE       PNUMBER=PNO
GROUP BY    PNUMBER, PNAME
```

- The grouping and functions are applied ***after*** joining the two relations

Grouping with Aggregate Functions

```
SELECT      PNAME, PNUMBER, COUNT(*)
FROM        PROJECT, WORKS_ON
WHERE       PNUMBER=PNO
GROUP BY    PNUMBER, PNAME
```

Pname	Pnumber	...	Essn	Pno	Hours
ProductX	1		123456789	1	32.5
ProductX	1		453453453	1	20.0
ProductY	2		123456789	2	7.5
ProductY	2		453453453	2	20.0
ProductY	2		333445555	2	10.0
ProductZ	3		666884444	3	40.0
ProductZ	3		333445555	3	10.0
Computerization	10	...	333445555	10	10.0
Computerization	10		999887777	10	10.0
Computerization	10		987987987	10	35.0
Reorganization	20		333445555	20	10.0
Reorganization	20		987654321	20	15.0
Reorganization	20		888665555	20	NULL
Newbenefits	30		987987987	30	5.0
Newbenefits	30		987654321	30	20.0
Newbenefits	30		999887777	30	30.0

PNAME	PNUMBER	COUNT(*)
ProductX	1	2
ProductY	2	3
ProductZ	3	3
Computerization	10	3
Reorganization	20	2
Newbenefits	30	3

The HAVING-Clause

- Sometimes we want to retrieve the values of these functions for only those *groups that satisfy certain conditions*
- The **HAVING**-clause is used for specifying a selection condition on groups (rather than on individual tuples)

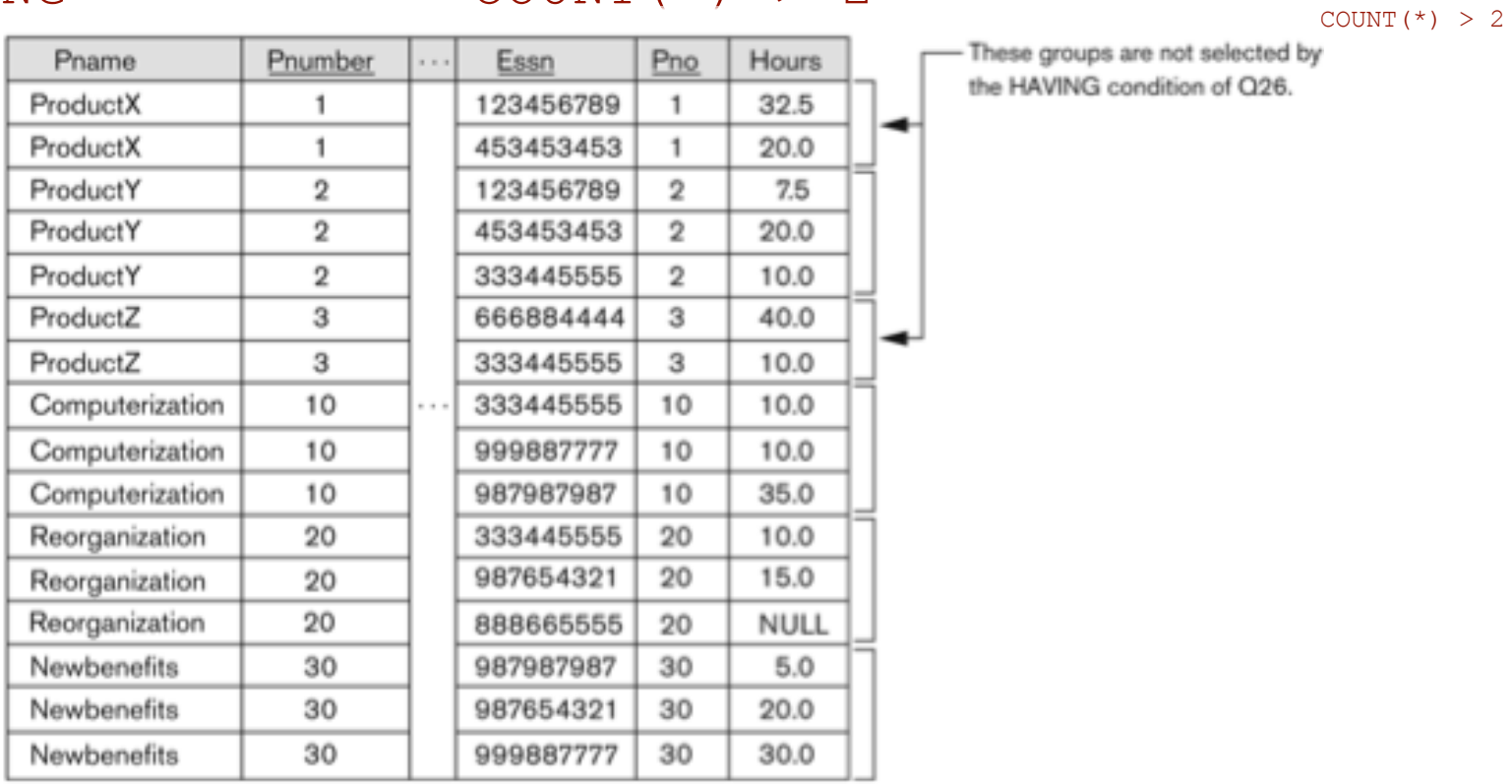
SELECT <attribute list, include **grouping attributes**>
FROM <table list>
[**WHERE** <condition>]
GROUP BY <**grouping attributes**>
HAVING <condition>

EXERCISE: What does this query?

```
SELECT      PNUMBER, PNAME, COUNT(*)
FROM        PROJECT, WORKS_ON
WHERE       PNUMBER=PNO
GROUP BY    PNUMBER, PNAME
HAVING      COUNT(*) > 2
```

The HAVING-Clause

```
SELECT      PNUMBER, PNAME, COUNT(*)
FROM        PROJECT, WORKS_ON
WHERE       PNUMBER=PNO
GROUP BY    PNUMBER, PNAME
HAVING      COUNT(*) > 2
```



Pname	Pnumber	...	Essn	Pno	Hours
ProductX	1		123456789	1	32.5
ProductX	1		453453453	1	20.0
ProductY	2		123456789	2	7.5
ProductY	2		453453453	2	20.0
ProductY	2		333445555	2	10.0
ProductZ	3		666884444	3	40.0
ProductZ	3		333445555	3	10.0
Computerization	10	...	333445555	10	10.0
Computerization	10		999887777	10	10.0
Computerization	10		987987987	10	35.0
Reorganization	20		333445555	20	10.0
Reorganization	20		987654321	20	15.0
Reorganization	20		888665555	20	NULL
Newbenefits	30		987987987	30	5.0
Newbenefits	30		987654321	30	20.0
Newbenefits	30		999887777	30	30.0

The HAVING-Clause

For each project *on which more than two employees work*, retrieve the project number, project name, and the number of employees who work on that project.

```
SELECT      PNUMBER, PNAME, COUNT(*)
FROM        PROJECT, WORKS_ON
WHERE       PNUMBER=PNO
GROUP BY    PNUMBER, PNAME
HAVING      COUNT(*) > 2
```

Pname	Pnumber	...	Essn	Pno	Hours
ProductY	2		123456789	2	7.5
ProductY	2		453453453	2	20.0
ProductY	2		333445555	2	10.0
Computerization	10		333445555	10	10.0
Computerization	10	...	999887777	10	10.0
Computerization	10		987987987	10	35.0
Reorganization	20		333445555	20	10.0
Reorganization	20		987654321	20	15.0
Reorganization	20		888665555	20	NULL
Newbenefits	30		987987987	30	5.0
Newbenefits	30		987654321	30	20.0
Newbenefits	30		999887777	30	30.0

Pname	Count (*)
ProductY	3
Computerization	3
Reorganization	3
Newbenefits	3

Result of Q26
(Pnumber not shown)

EXERCISE: The HAVING-Clause (contd.)

Example: Count the *total* number of employees whose salaries exceed $40,000 in each department, but only for the departments where more than five employees work.

```
SELECT          DNAME, COUNT(*)
FROM            DEPARTMENT, EMPLOYEE
WHERE           DNUMBER=DNO AND SALARY > 40000
GROUP BY        DNAME
HAVING          COUNT(*) > 5
```

Is this the correct query?

EXERCISE: The HAVING-Clause (contd.)

Example: Count the *total* number of employees whose salaries exceed $40,000 in each department, but only for the departments where more than five employees work.

departments with more than 5 ppl earning more than 40000
but we are interested in how many ppl earning more than 40000 are in large departments over 5 ppl

```
SELECT      DNAME, COUNT(*)
FROM        DEPARTMENT, EMPLOYEE
WHERE       DNUMBER=DNO AND SALARY > 40000
GROUP BY    DNAME
HAVING      COUNT(*) > 5
```

Incorrect query! It selects only the departments that have more *than five employees who each earn more than $40,000. The query is too limited.*

The HAVING-Clause (contd.)

Example: Count the *total* number of employees whose salaries exceed $40,000 in each department, but only for the departments where more than five employees work.

```
SELECT        DNAME, COUNT(*)
FROM          DEPARTMENT, EMPLOYEE
WHERE         DNUMBER=DNO AND SALARY > 40000
              AND DNO IN (SELECT DNO
                          FROM EMPLOYEE
                          GROUP BY DNO
                          HAVING    COUNT(*) > 5)
GROUP BY DNAME
```

Use a Nested Correlated Query – this nested query selects the DEPARTMENTS who's number of employees > 5.

Summary of SQL Queries

- A query in SQL can consist of up to six clauses, but only the first two, ***SELECT and FROM, are mandatory***. The clauses are specified in the following order:

```
SELECT      <attribute list, include any grouping attributes>
FROM        <table list>
[WHERE      <condition>]
[GROUP BY   <grouping attribute(s)>]
[HAVING     <grouping condition>]
[ORDER BY   <attribute list>]
```

Summary of SQL Queries

- The **SELECT**-clause lists the attributes or functions to be retrieved
- The **FROM**-clause specifies all relations (or aliases) needed in the query but not those needed in nested queries
- The **WHERE**-clause specifies the conditions for selection and join of tuples from the relations specified in the FROM-clause
- **GROUP BY** specifies grouping attributes
- **HAVING** specifies a condition for selection of groups
- **ORDER BY** specifies an order for displaying the result of a query

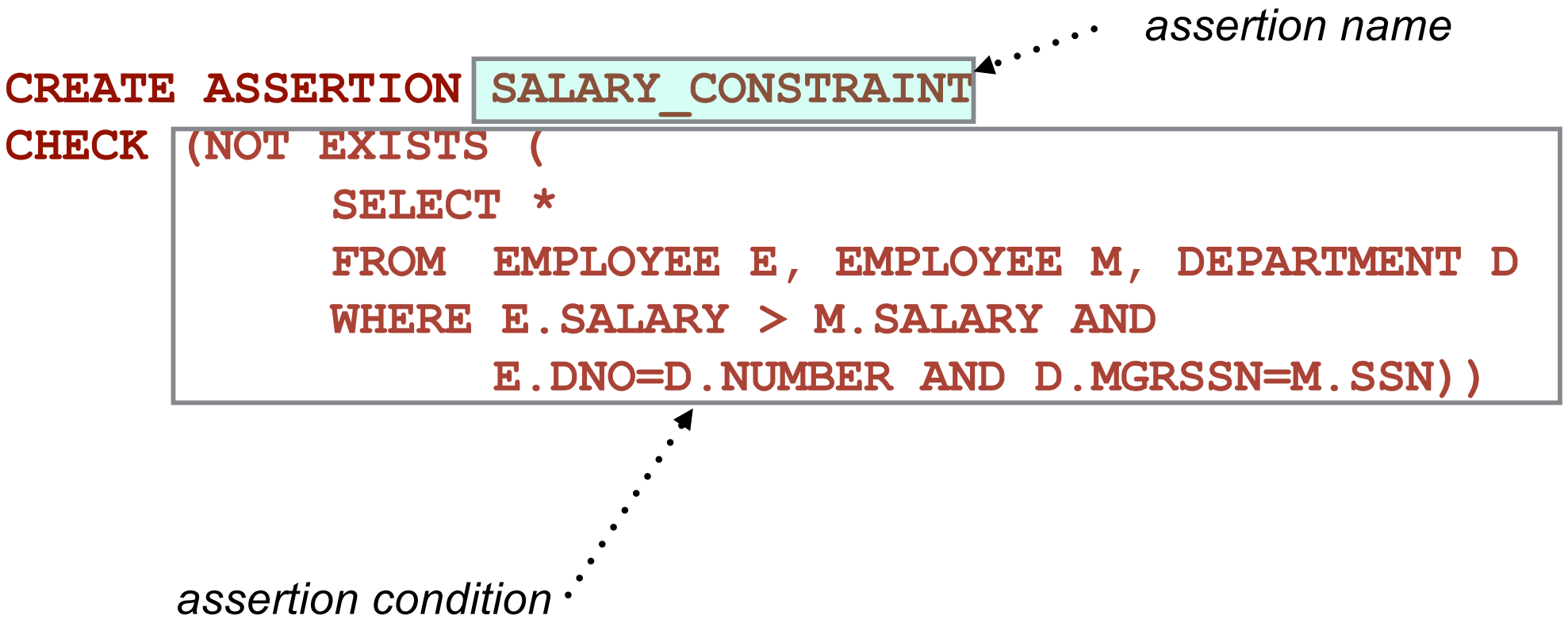
- A query is evaluated by:
  1. Including tables in the FROM clause
  2. Applying the conditions in the WHERE-clause
  3. Performing GROUP BY
  4. Applying conditions in the HAVING-clause
  5. Selecting attributes in the SELECT-clause
  6. Running ORDER BY on the resulting tuples

Constraints as Assertions

- General constraints: constraints that do not fit in the basic SQL categories
- Useful for ***Schema Assertions*** - Outside the scope of the built-in relational model constraints (primary and unique keys, entity integrity, referential integrity).
- Defines whether the State of the Database is VALID at any given point of time.
- **`CREATE ASSERTION,`** Components include:
  - A constraint name
  - Followed by a CHECK keyword
  - Followed by a condition clause
- Enforcing the Assertion is up to the Database Implementation – i.e. Rejecting a Query that will violate the CHECK ASSERTION.

CREATE ASSERTION Example

Example: The salary of an employee must not be greater than the salary of the manager of the department that the employee works for



Views in SQL

- A view is a "*virtual*" table that is derived from other tables
- Two ways they are implemented in implementation:
  - Query modification - *copy and paste queries*
  - View materialization *- short-term physical implementation*
- Limited for UPDATE operations. Unable to Update Views which are:
    - Derived from Multiple Tables with JOINs
    - Views defined with GROUP BY and aggregate functions. Allows full query operations
- A convenience for expressing certain operations
- Useful for security and authorization
- Prevents redundant storage of data

SQL View Example

Example: A "friendlier" view of WORKS_ON

- SQL command: `CREATE VIEW`

view name

```
CREATE VIEW WORKS_ON1 AS
SELECT FNAME, LNAME, PNAME, HOURS
FROM EMPLOYEE, PROJECT, WORKS_ON
WHERE SSN=ESSN AND PNO=PNUMBER
```

query to specify the contents of view

WORKS_ON1

Fname	Lname	Pname	Hours

view name option: specify attribute names

WORKS_ON1(FIRST_NAME, LAST_NAME, PROJECT, HOURS)

Using a Virtual Table (a View)

Example: A "friendlier" view of WORKS_ON

- We can specify SQL queries on a newly create table (view):

```
SELECT FNAME, LNAME
       FROM WORKS_ON1
       WHERE PNAME='ProductX';
```

- When no longer needed, a view can be dropped:

```
DROP WORKS_ON1;
```

- Dropping a View does NOT modify the data!

Important Take-Aways

- 6 Parts of SELECT Query and Execution Order
    - SELECT… FROM … WHERE… GROUP BY … HAVING… ORDER BY
- Modifying the STATE of a database
    - INSERT, UPDATE, DELETE
- Nested Queries in WHERE clause:
    - IN, EXISTS, ALL in conditions, correlated queries
- Types of Joins: INNER, OUTER, NATURAL JOINS
- GROUP BY… Aggregation Functions
- Why and How to use: CREATE ASSERTIONS
- Why and How to use: CREATE VIEW