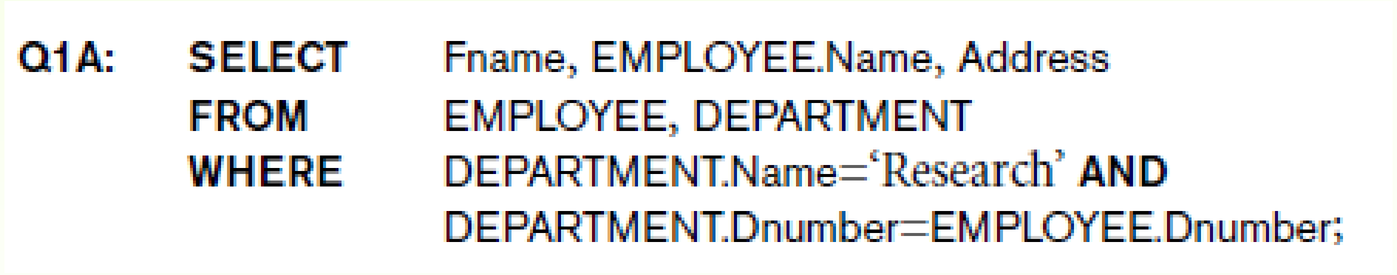
- SQL (more)

Ambiguous Attribute Names

- Same name can be used for two (or more) attributes (attributes are in different relations)
  - Must **qualify** the attribute name with the relation name to prevent ambiguity. Assume Dno and Lname of EMPLOYEE are Dnumber and Name. Dname of DEPT is Name.



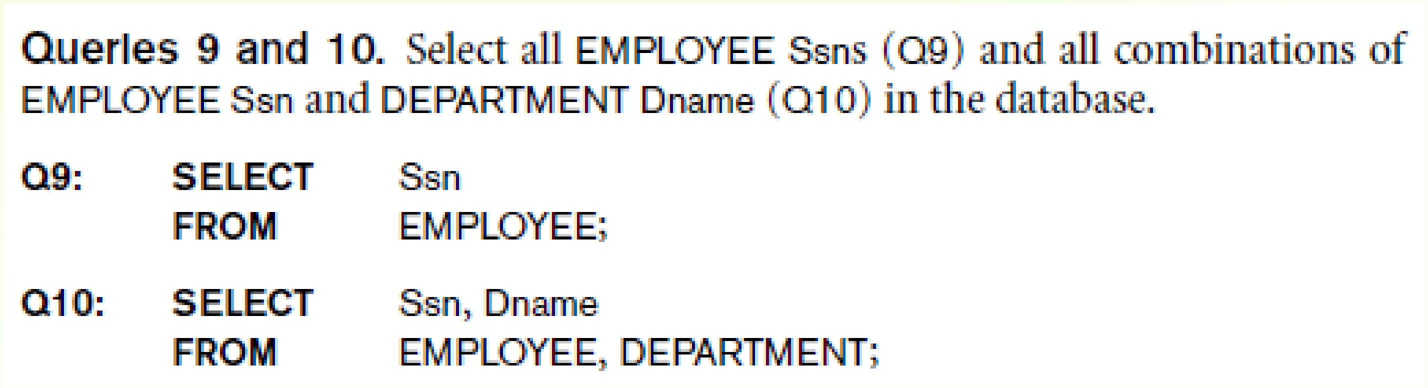
```
Q1A:    SELECT    Fname, EMPLOYEE.Name, Address
        FROM      EMPLOYEE, DEPARTMENT
        WHERE     DEPARTMENT.Name='Research' AND
                  DEPARTMENT.Dnumber=EMPLOYEE.Dnumber;
```

Aliasing, Renaming, and Tuple Variables

- **Aliases** or **tuple variables**
  - Declare alternative relation names E and S, called alias or tuple variable for EMPLOYEE
- Examples:
  - `EMPLOYEE AS E(Fn, Mi, Ln, Ssn, Bd, Addr, Sex, Sal, Sssn, Dno)`
  - `FROM EMPLOYEE E`
  - `FROM EMPLOYEE E, EMPLOYEE S`

Unspecified WHERE Clause and Use of the Asterisk

- Missing `WHERE` clause
  - Indicates no condition on tuple selection
- `CROSS PRODUCT`
  - All possible tuple combinations



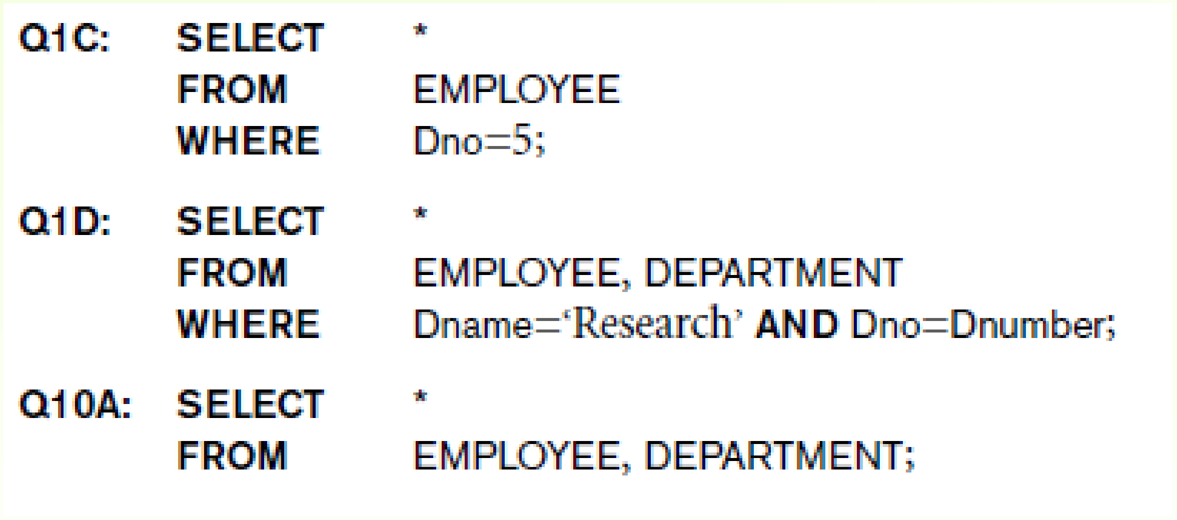
Queries 9 and 10. Select all EMPLOYEE Ssns (Q9) and all combinations of EMPLOYEE Ssn and DEPARTMENT Dname (Q10) in the database.

```
Q9:     SELECT   Ssn
        FROM     EMPLOYEE;

Q10:    SELECT   Ssn, Dname
        FROM     EMPLOYEE, DEPARTMENT;
```

Unspecified WHERE Clause and Use of the Asterisk (cont'd.)

- Specify an asterisk (*) in select-clause
  - Retrieve all the attribute values of the selected tuples



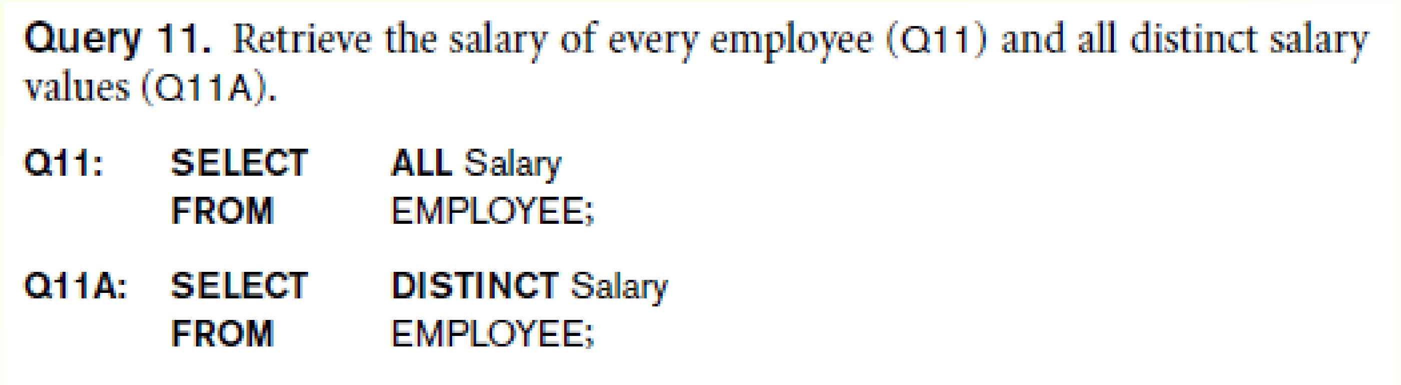
```
Q1C:   SELECT   *
       FROM     EMPLOYEE
       WHERE    Dno=5;

Q1D:   SELECT   *
       FROM     EMPLOYEE, DEPARTMENT
       WHERE    Dname='Research' AND Dno=Dnumber;

Q10A:  SELECT   *
       FROM     EMPLOYEE, DEPARTMENT;
```

Tables as Sets in SQL

- SQL does not automatically eliminate duplicate tuples in query results
- Use the keyword **`DISTINCT`** in the `SELECT` clause
  - Only distinct tuples should remain in the result

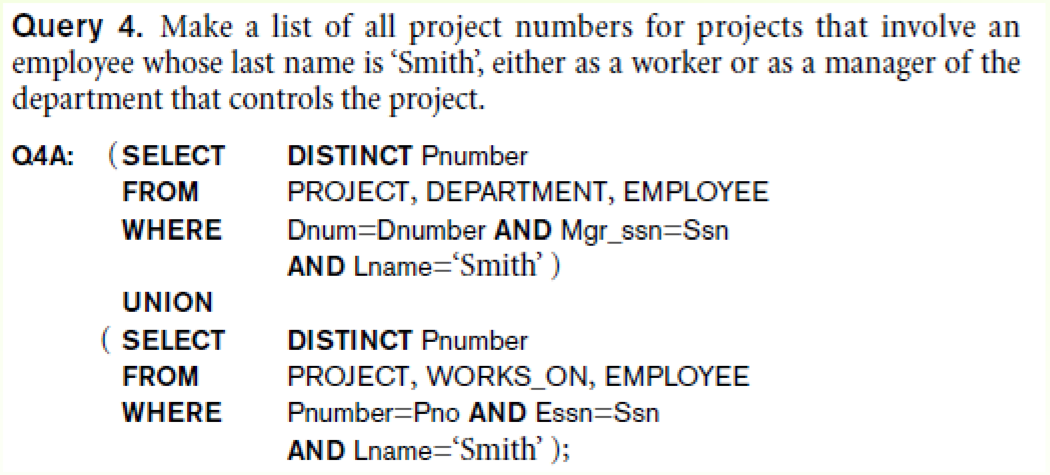

Query 11. Retrieve the salary of every employee (Q11) and all distinct salary values (Q11A).

```
Q11:    SELECT    ALL Salary
        FROM      EMPLOYEE;

Q11A:   SELECT    DISTINCT Salary
        FROM      EMPLOYEE;
```

Tables as Sets in SQL (cont'd.)

- Set operations
  - `UNION`, **`EXCEPT`** (difference), **`INTERSECT`**
  - Corresponding multiset operations: `UNION ALL`, `EXCEPT ALL`, `INTERSECT ALL`)



Query 4. Make a list of all project numbers for projects that involve an employee whose last name is 'Smith', either as a worker or as a manager of the department that controls the project.

```
Q4A:  ( SELECT    DISTINCT Pnumber
        FROM      PROJECT, DEPARTMENT, EMPLOYEE
        WHERE     Dnum=Dnumber AND Mgr_ssn=Ssn
                  AND Lname='Smith' )
        UNION
      ( SELECT    DISTINCT Pnumber
        FROM      PROJECT, WORKS_ON, EMPLOYEE
        WHERE     Pnumber=Pno AND Essn=Ssn
                  AND Lname='Smith' );
```

Substring Matching & Arithmetic Operators

LIKE comparison operator: Used for string **pattern matching**

- % replaces an arbitrary number of zero or more chars
- 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** '\'
- underscore (_) replaces a single character

Std arithmetic: Addition (+), subtraction (–), multiplication (*), and division (/)

BETWEEN comparison operator: Find names of all instructors with salary b/w $90,000 and $100,000 (≥$90,000 & ≤$100,000)

select *name*

from *instructor*

where *salary* **between** 90000 **and** 100000

Ordering **Display of Tuples**

- Use **`ORDER BY`** clause
- List in alphabetic order the names of all instructors:

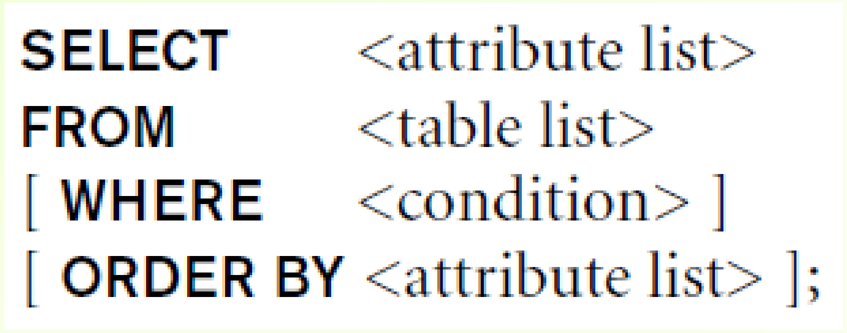
  select distinct *name*

  from *instructor*

  order by *name ASC;*

- **`DESC`** to see result in a descending order of values, keyword **`ASC`** to specify ascending order **explicitly**
  - `ORDER BY D.Dname DESC, E.Lname ASC, E.Fname ASC`

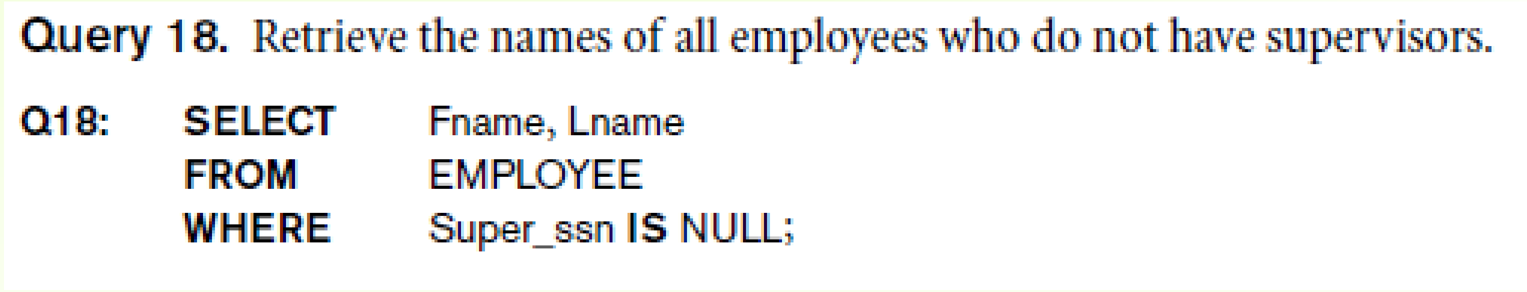
Discussion and Summary of Basic SQL Retrieval Queries



```
SELECT      <attribute list>
FROM        <table list>
[ WHERE     <condition> ]
[ ORDER BY <attribute list> ];
```

Comparisons Involving NULL and Three-Valued Logic (cont'd.)

- SQL allows queries that check whether an attribute value is `NULL`
  - `IS` or `IS NOT NULL`



Query 18. Retrieve the names of all employees who do not have supervisors.

```
Q18:    SELECT    Fname, Lname
        FROM      EMPLOYEE
        WHERE     Super_ssn IS NULL;
```

Nested Queries, Tuples, and Set/Multiset Comparisons

- **Nested queries**
  - Complete select-from-where blocks within WHERE clause of another query
  - **Outer query**
- Comparison operator `IN`
  - Compares value *v* with a set (or multiset) of values *V*
  - Evaluates to `TRUE` if *v* is one of the elements in *V*

Nested Queries (cont'd.)

```
Q4A:    SELECT      DISTINCT Pnumber
        FROM        PROJECT
        WHERE       Pnumber IN
                    ( SELECT        Pnumber
                      FROM          PROJECT, DEPARTMENT, EMPLOYEE
                      WHERE         Dnum=Dnumber AND
                                    Mgr_ssn=Ssn AND Lname='Smith' )
                    OR
                    Pnumber IN
                    ( SELECT        Pno
                      FROM          WORKS_ON, EMPLOYEE
                      WHERE         Essn=Ssn AND Lname='Smith' );
```

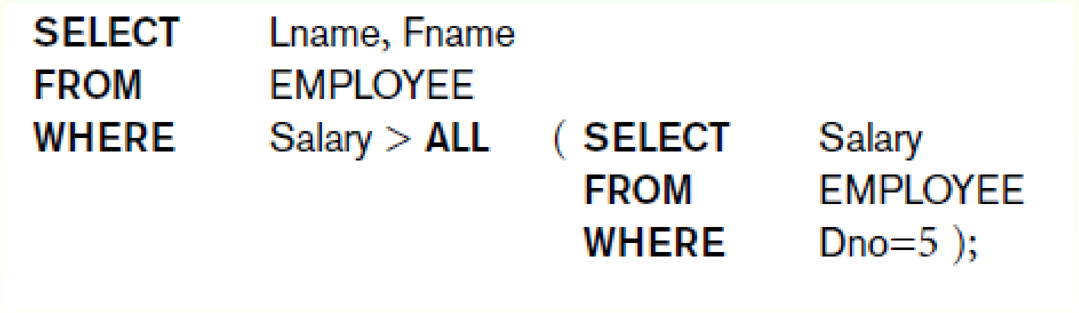
Nested Queries (cont'd.)

- Use tuples of values in comparisons
  - Place them within parentheses

```
SELECT      DISTINCT Essn
FROM        WORKS_ON
WHERE       (Pno, Hours) IN ( SELECT      Pno, Hours
                              FROM        WORKS_ON
                              WHERE       Essn='123456789' );
```

Nested Queries (cont'd.)

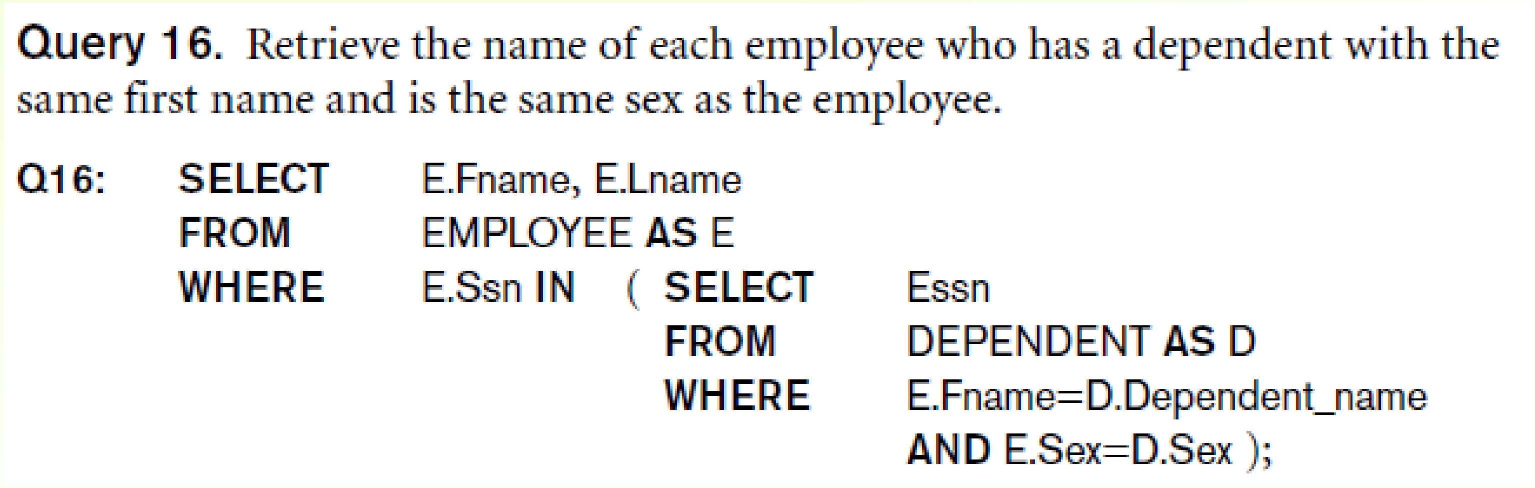
- Use other comparison operators to compare a single value *v*
  - `= ANY` (or `= SOME`) operator
    - Returns `TRUE` if the value *v* is equal to some value in the set *V* and is hence equivalent to `IN`
  - Other operators that can be combined with `ANY` (or `SOME`): `>`, `>=`, `<`, `<=`, and `<>`



```
SELECT    Lname, Fname
FROM      EMPLOYEE
WHERE     Salary > ALL    ( SELECT    Salary
                            FROM      EMPLOYEE
                            WHERE     Dno=5 );
```

Nested Queries (cont'd.)

- Avoid potential errors and ambiguities
  - Create tuple variables (aliases) for all tables referenced in SQL query



Query 16. Retrieve the name of each employee who has a dependent with the same first name and is the same sex as the employee.

```
Q16:    SELECT    E.Fname, E.Lname
        FROM      EMPLOYEE AS E
        WHERE     E.Ssn IN  ( SELECT    Essn
                              FROM      DEPENDENT AS D
                              WHERE     E.Fname=D.Dependent_name
                                        AND E.Sex=D.Sex );
```

Correlated Nested Queries

- **Correlated** nested query
  - Evaluated once for each tuple in the outer query (Q16)

```
SELECT E.Fname, E.Lname
FROM EMPLOYEE E, DEPENDENT D
WHERE E.ssn=D.Essn and E.sex=D.sex and
    E.Fname=D.Dependent_name
```

The EXISTS and UNIQUE

- `EXISTS` function: Check if result is empty or not

```
SELECT E.Fname, E.Lname
FROM EMPLOYEE E, DEPENDENT D
WHERE EXISTS (select * from DEPENDENT D
                             where E.ssn=D.Essn and
   E.sex=D.sex and E.Fname=D.Dependent_name);
```

- `NOT EXISTS: emp w/o dependents`

```
SELECT E.Fname, E.Lname
FROM EMPLOYEE E
WHERE NOT EXISTS (select * from DEPENDENT
                             where ssn=Essn);
```

- SQL function `UNIQUE(Q)` : Returns `TRUE` if there are no duplicate tuples in the result of query Q

Set Operations

- SECTION(course_id, course_name, credit, prof, year, sem)
- Find courses that ran in Fall 2009 or in Spring 2010

  (**select** *course_id* **from** *section* **where** *sem*= ‘Fall’ **and** *year* = 2009)

  union

  (**select** *course_id* **from** *section* **where** *sem*= ‘Spring’ **and** *year* = 2010)
- Find courses that ran in Fall 2009 but not in Spring 2010

  (**select** *course_id* **from** *section* **where** *sem*= ‘Fall’ **and** *year* = 2009)

  except

  (**select** *course_id* **from** *section* **where** *sem*= ‘Spring’**and** *year* = 2010)
- Find courses that ran in Fall 2009 and in Spring 2010

  (**select** *course_id* **from** *section* **where** *sem*= ‘Fall’**and** *year* = 2009)

  intersect

  (**select** *course_id* **from** *section* **where** *sem*= ‘Spring’**and** *year* = 2010)

COMPANY Database Schema and state--Figure 5.5 (from Elmasri/Navathe)

All following examples use the COMPANY database

EMPLOYEE

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO

DEPARTMENT

DNAME	DNUMBER	MGRSSN	MGRSTARTDATE

DEPT_LOCATIONS

DNUMBER	DLOCATION

PROJECT

PNAME	PNUMBER	PLOCATION	DNUM

WORKS_ON

ESSN	PNO	HOURS

DEPENDENT

ESSN	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP

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

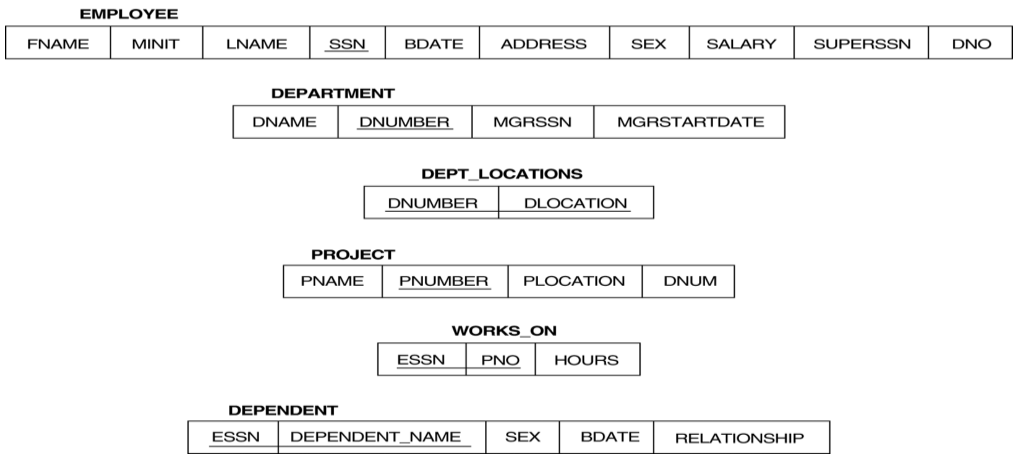
DEPARTMENT	DNAME	DNUMBER	MGRSSN	MGRSTARTDATE
	Research	5	333445555	1988-05-22
	Administration	4	987654321	1995-01-01
	Headquarters	1	888665555	1981-06-19

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

SQL Queries



EMPLOYEE

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO

DEPARTMENT

DNAME	DNUMBER	MGRSSN	MGRSTARTDATE

DEPT_LOCATIONS

DNUMBER	DLOCATION

PROJECT

PNAME	PNUMBER	PLOCATION	DNUM

WORKS_ON

ESSN	PNO	HOURS

DEPENDENT

ESSN	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP

- SQL allows queries that check whether an attribute value is `NULL`
  - `IS` or `IS NOT NULL`

Query 18. Retrieve the names of all employees who do not have supervisors.

```
Q18:    SELECT    Fname, Lname
        FROM      EMPLOYEE
        WHERE     Super_ssn IS NULL;
```

The EXISTS and UNIQUE

- EXISTS function: Check if result is empty or not

```
SELECT E.Fname, E.Lname
FROM EMPLOYEE E, DEPENDENT D
WHERE EXISTS (select * from DEPENDENT D
                              where E.ssn=D.Essn and
   E.sex=D.sex and E.Fname=D.Dependent_name);
```

- NOT EXISTS: emp w/o dependents

```
SELECT E.Fname, E.Lname
FROM EMPLOYEE E
WHERE NOT EXISTS (select * from DEPENDENT
                              where ssn=Essn);
```

- SQL function UNIQUE(Q): Returns TRUE if there are no duplicate tuples in the result of query Q

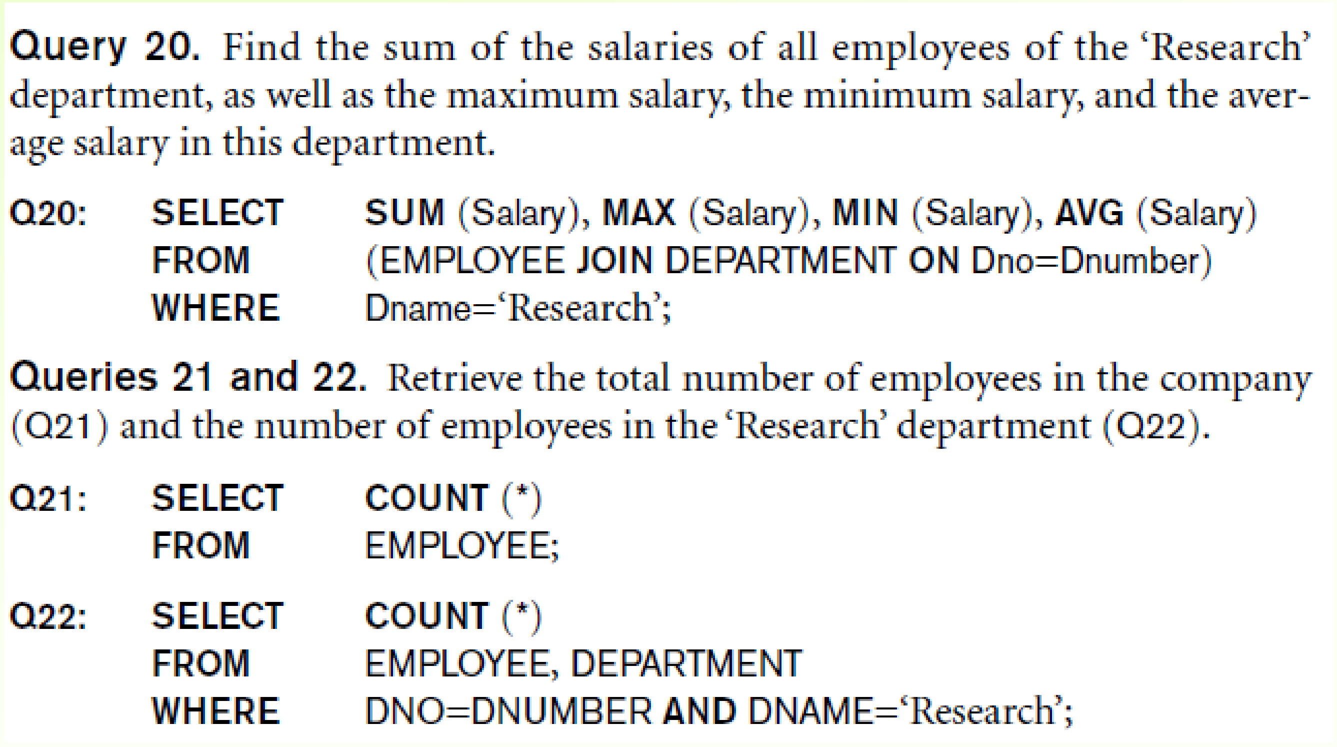
Aggregate fcns

Sum of salaries of all employees , max salary, min salary and avg salary

```
SELECT sum(salary), max(salary),
   min(salary), avg(salary)
FROM EMPLOYEE;
```

Aggregate Functions in SQL

- NULL values discarded when aggregate functions are applied to a particular column



Query 20. Find the sum of the salaries of all employees of the 'Research' department, as well as the maximum salary, the minimum salary, and the average salary in this department.

```
Q20:    SELECT    SUM (Salary), MAX (Salary), MIN (Salary), AVG (Salary)
        FROM      (EMPLOYEE JOIN DEPARTMENT ON Dno=Dnumber)
        WHERE     Dname='Research';
```

Queries 21 and 22. Retrieve the total number of employees in the company (Q21) and the number of employees in the 'Research' department (Q22).

```
Q21:    SELECT    COUNT (*)
        FROM      EMPLOYEE;

Q22:    SELECT    COUNT (*)
        FROM      EMPLOYEE, DEPARTMENT
        WHERE     DNO=DNUMBER AND DNAME='Research';
```

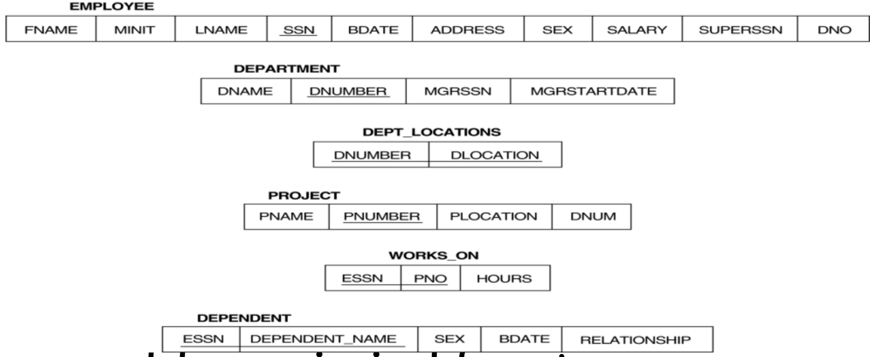
COUNT

- Count the number of distinct salary values

```
SELECT count(distinct salary)
FROM EMPLOYEE;
```

- NAMES OF ALL EMPLOYEES with 2 or more dependents

```
SELECT Lname, Fname
FROM EMPLOYEE
WHERE (select count(*)
            from dependent where ssn=Essn) >= 2;
```


EMPLOYEE
FNAME | MINIT | LNAME | SSN | BDATE | ADDRESS | SEX | SALARY | SUPERSSN | DNO
DEPARTMENT
DNAME | DNUMBER | MGRSSN | MGRSTARTDATE
DEPT_LOCATIONS
DNUMBER | DLOCATION
PROJECT
PNAME | PNUMBER | PLOCATION | DNUM
WORKS_ON
ESSN | PNO | HOURS
DEPENDENT
ESSN | DEPENDENT_NAME | SEX | BDATE | RELATIONSHIP

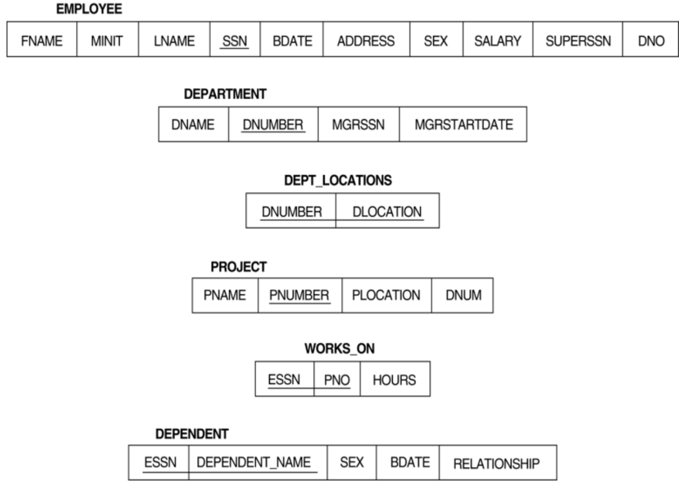
Sample Queries on COMPANY

- Query 25: Retrieve all employees whose address is in Houston, Texas. Here, the value of the ADDRESS attribute must contain the substring 'Houston,TX' in it.

```
Q25:        SELECT  FNAME, LNAME
            FROM    EMPLOYEE
            WHERE ADDRESS LIKE  '%Houston,TX%'
```

- Query 26: Retrieve all employees who were born during the 1950s.
  - *Here, '5' must be the 8th character of the string (according to our format for date),*

```
Q26:        SELECT  FNAME, LNAME
            FROM            EMPLOYEE
            WHERE   BDATE LIKE      '_______5_'
```



Sample Queries on COMPANY

- **Query 1: Retrieve the name and address of all employees who work for the 'Research' department.**

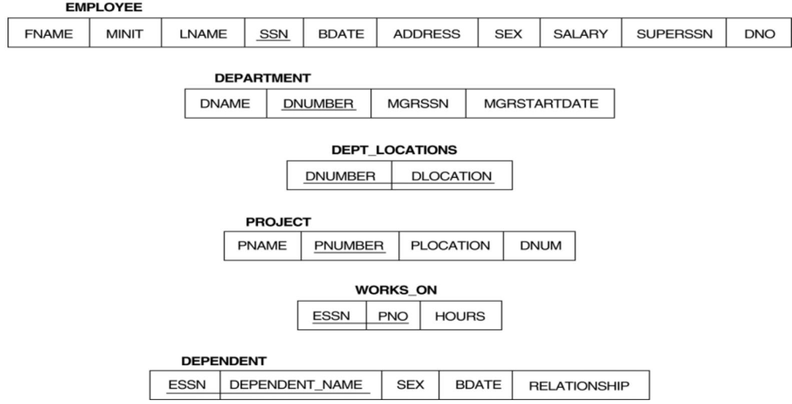
```
Q1:  SELECT   FNAME, LNAME, ADDRESS
     FROM     EMPLOYEE
     WHERE    DNO IN  (SELECT  DNUMBER
                       FROM    DEPARTMENT
                       WHERE   DNAME='Research' )
```

- 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) of the outer query
- **Query 12: Retrieve the name of each employee who has a dependent with the same first name as the employee.**

```
Q12: 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)
```

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.

```
Q12A:        SELECT E.FNAME, E.LNAME
             FROM     EMPLOYEE E, DEPENDENT D
             WHERE    E.SSN=D.ESSN AND
                      E.FNAME=D.DEPENDENT_NAME
```


EMPLOYEE
FNAME MINIT LNAME SSN BDATE ADDRESS SEX SALARY SUPERSSN DNO
DEPARTMENT
DNAME DNUMBER MGRSSN MGRSTARTDATE
DEPT_LOCATIONS
DNUMBER DLOCATION
PROJECT
PNAME PNUMBER PLOCATION DNUM
WORKS_ON
ESSN PNO HOURS
DEPENDENT
ESSN DEPENDENT_NAME SEX BDATE RELATIONSHIP

Sample Queries on COMPANY

- Query 12: Retrieve the name of each employee who has a dependent with the same first name as the employee.

```
Q12B:       SELECT  FNAME, LNAME
            FROM    EMPLOYEE
            WHERE EXISTS  (SELECT       *
                          FROM          DEPENDENT
                          WHERE         SSN=ESSN AND
                               FNAME=DEPENDENT_NAME)
```

- Query 6: Retrieve the names of employees who have no dependents.

```
Q6:  SELECT FNAME, LNAME
     FROM   EMPLOYEE
     WHERE NOT EXISTS  (SELECT *
                       FROM          DEPENDENT
                       WHERE         SSN=ESSN)
```

EMPLOYEE

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO

DEPARTMENT

DNAME	DNUMBER	MGRSSN	MGRSTARTDATE

DEPT_LOCATIONS

DNUMBER	DLOCATION

PROJECT

PNAME	PNUMBER	PLOCATION	DNUM

WORKS_ON

ESSN	PNO	HOURS

DEPENDENT

ESSN	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP

Sample Queries on COMPANY

- Query 3: Retrieve the name of each employee who works all the projects on controlled by department number 5.

  Q3 (rephrase) select each employee s.t. there does not exist a project controlled by dept.#5 that the employee that the employee does not work on.

```
SELECT FNAME, LNAME
FROM   EMPLOYEE
WHERE NOT EXIST
            (SELECT *
             FROM WORKS_ON B
             WHERE (B.PNO IN (SELECT PNUMBER
                              FROM PROJECT
                              WHERE DNUM=5))
                     AND
                     NOT EXIST (SELECT *
                                FROM WORKS_ON C
                                WHERE SSN=C.ESSN and
                                      C.PNO=B.PNO) );
```

- **Query 7: List the names of managers who have at least one dependent.**

```
Q7:  SELECT FNAME, LNAME
           FROM       EMPLOYEE
           WHERE  EXIST(SELECT *
                             FROM    DEPENDENT
                             WHERE   SSN=ESSN)
                      AND
                      EXIST(SELECT *
                            FROM DEPARTMENT
                            WHERE SSN=MGRSSN) )
```

Rewrite this query using only a single nested query or no nested query.

Query 22: 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.

```
Q22:     SELECT PNUMBER, PNAME, COUNT(*)
             FROM     PROJECT, WORKS_ON
             WHERE    PNUMBER=PNO
             GROUP BY         PNUMBER, PNAME
             HAVING COUNT (*) > 2
```

Updates in COMPANY DB

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

```
U5:          UPDATE        PROJECT
             SET     PLOCATION = 'Bellaire',
                     DNUM = 5
             WHERE         PNUMBER=10
```

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

```
U6:          UPDATE EMPLOYEE
             SET SALARY = SALARY *1.1
             WHERE DNO  IN (SELECT        DNUMBER
                           FROM           DEPARTMENT
                           WHERE          DNAME='Research')
```

Grouping: The GROUP BY and HAVING Clauses

- **Partition** relation into subsets of tuples
  - Based on **grouping attribute(s)**
  - Apply function to each such group independently
- If NULLs exist in grouping attribute
  - Separate group created for all tuples with a NULL value in grouping attribute

GROUP BY clause

- Specifies grouping attributes
- For each dept, retrieve dept no, number of employees in dept, and their avg salary

```
SELECT Dno, COUNT(*), AVG(Salary)
FROM Employee
GROUP BY Dno;
```

- Each group has same value for **grouping attribute** Dno
- For each project, retrieve project no, project name, number of emps who work on that project (shows use of join cond with group by)

```
SELECT Pnumber, Pname, COUNT(*)
FROM PROJECT, WORKS_ON
WHERE Pnumber=Pno
GROUP BY Pnumber, Pname
```

GROUP BY and HAVING

- For each project on which more than 2 emps work, retrieve project no, project name, number of emps who work on that project

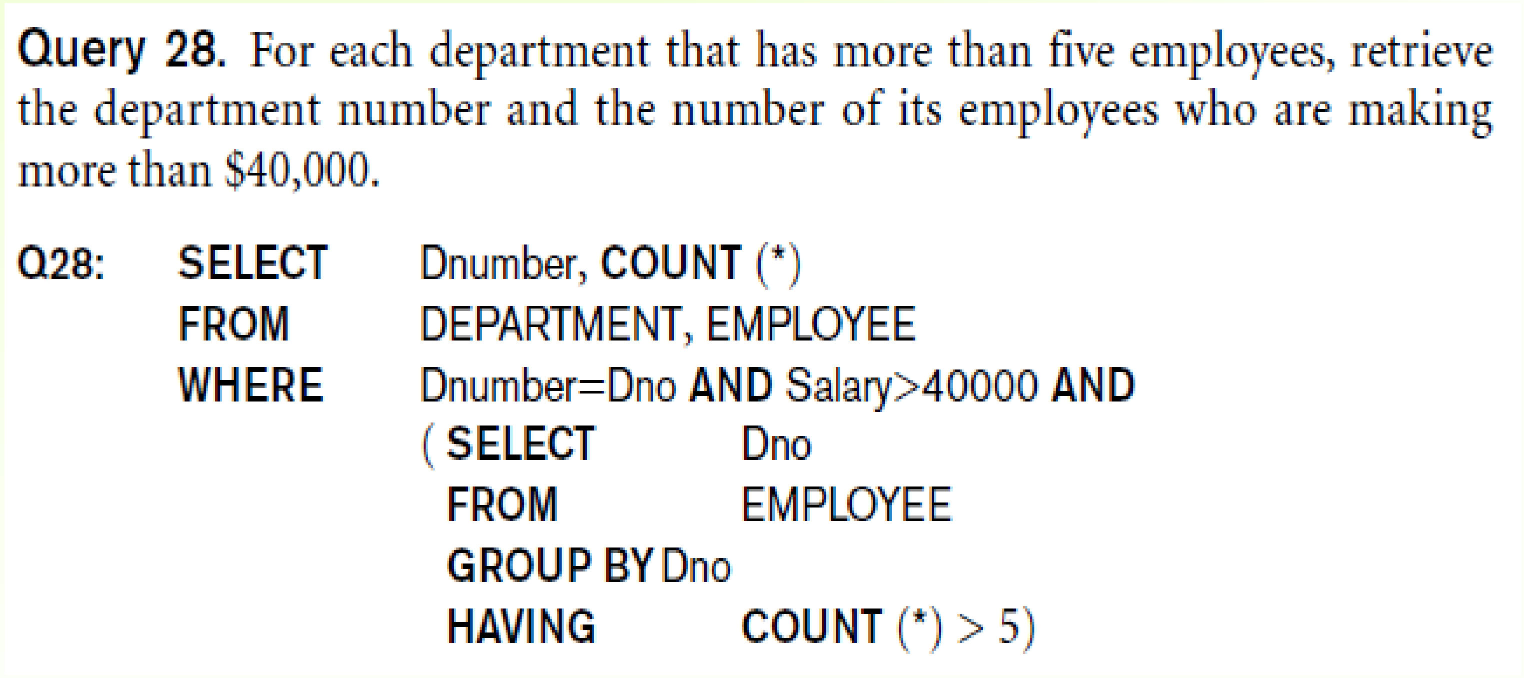
```
SELECT Pnumber, Pname, COUNT(*)
FROM PROJECT, WORKS_ON
WHERE Pnumber=Pno
GROUP BY Pnumber, Pname
HAVİNG COUNT(*) > 2;
```

- For each project, retrieve project no, project name, number of emps from dept 5 who work on that project

```
SELECT Pnumber, Pname, COUNT(*)
FROM PROJECT, WORKS_ON, EMPLOYEE
WHERE Pnumber=Pno AND ssn=Essn and Dno=5
GROUP BY Pnumber, Pname;
```

GROUP BY and HAVING

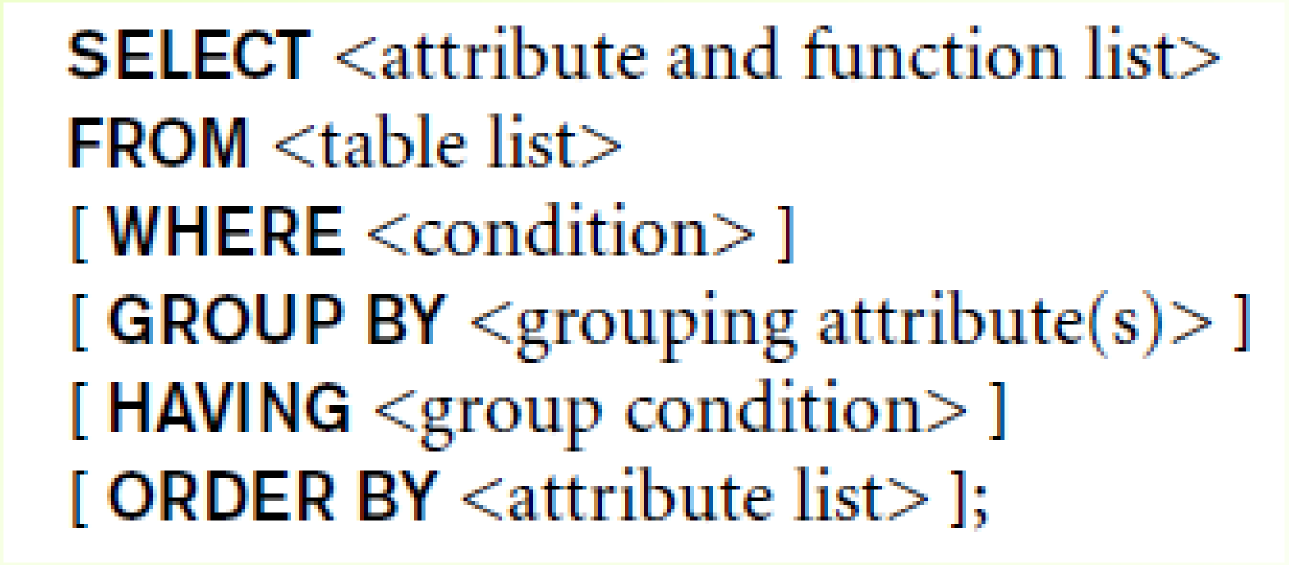
- **`HAVING`** clause
  - Provides a condition on the summary information
  - Problem with following query?



Query 28. For each department that has more than five employees, retrieve the department number and the number of its employees who are making more than $40,000.

```
Q28:   SELECT     Dnumber, COUNT (*)
       FROM       DEPARTMENT, EMPLOYEE
       WHERE      Dnumber=Dno AND Salary>40000 AND
                  ( SELECT        Dno
                    FROM          EMPLOYEE
                    GROUP BY Dno
                    HAVING        COUNT (*) > 5)
```

General Form of SQL Queries



```
SELECT <attribute and function list>
FROM <table list>
[ WHERE <condition> ]
[ GROUP BY <grouping attribute(s)> ]
[ HAVING <group condition> ]
[ ORDER BY <attribute list> ];
```

Evaluation of SQL Queries

- The order of execution:

  FROM ➔ WHERE ➔ GROUP BY ➔ HAVING ➔ SELECT ➔ UNION ➔ORDER BY

- SQL ile Sorgu Yazma:
  - Sonraki sayfadaki sorguları yazalım:

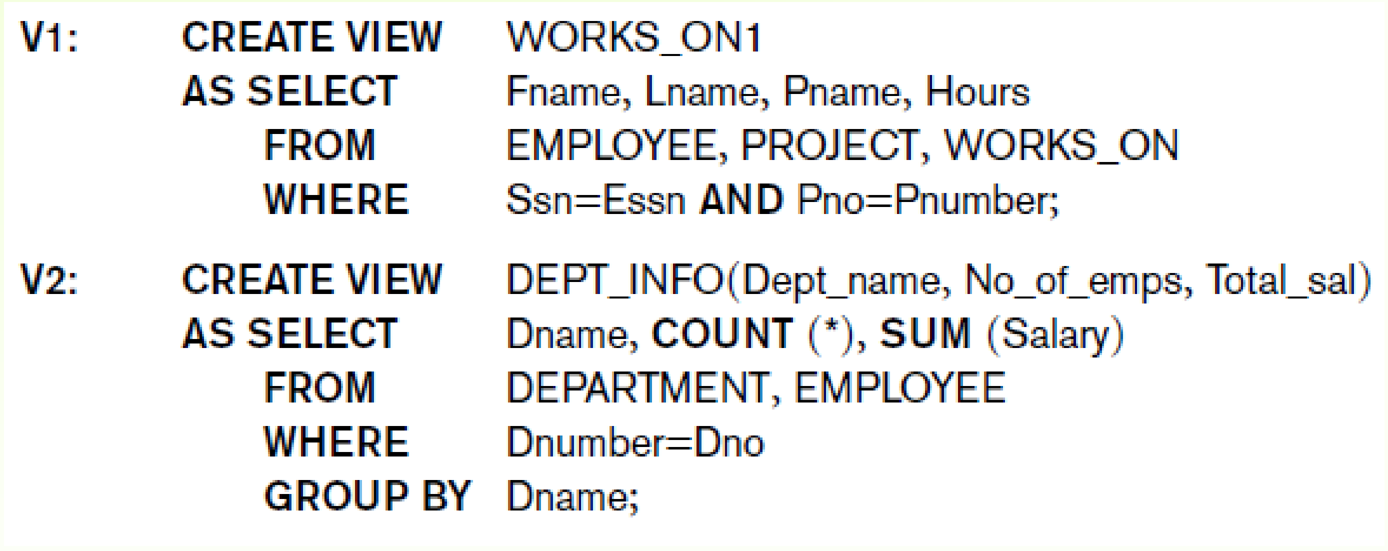
VT Şeması ve sorgular

- Ürün(<u>**üKodu**</u>, üAdı, birFiyatı, stokMiktarı)
- Bileşen(<u>**bKodu**</u>, bAdı, açıklama, stokMiktarı, sNo)
- Satıcı(<u>**sNo**</u>, sAdı, sAdresi, sTelNo)
- Gerekli(<u>**üKodu**</u>, <u>**bKodu**</u>, miktar)

- 1) Kaya adlı satıcının sattığı bileşenler
- 2) Kaya adlı satıcının toplam kaç çeşit bileşen sattığı
- 3) Kaya adlı satıcının sattığı bileşenlerin stok miktarları toplamı
- 4) stokmiktarı en az 10 olan ürünlerin adı
- 5) stokmiktarı 10 ‘dan az olan bileşenler ve onu sağlayan satıcı ad ve telno
- 6) Bilgisayar ürünü hangi bileşenlerden oluşuyor

Specification of Views in SQL

- **`CREATE VIEW`** command
  - Give table name, list of attribute names, and a query to specify the contents of the view



```
V1:     CREATE VIEW   WORKS_ON1
        AS SELECT     Fname, Lname, Pname, Hours
           FROM       EMPLOYEE, PROJECT, WORKS_ON
           WHERE      Ssn=Essn AND Pno=Pnumber;

V2:     CREATE VIEW   DEPT_INFO(Dept_name, No_of_emps, Total_sal)
        AS SELECT     Dname, COUNT (*), SUM (Salary)
           FROM       DEPARTMENT, EMPLOYEE
           WHERE      Dnumber=Dno
           GROUP BY   Dname;
```

Specification of Views in SQL (cont'd.)

- Specify SQL queries on a view
- View always up-to-date
  - Responsibility of the DBMS and not the user
- **`DROP VIEW`** command
  - Dispose of a view

The DROP Command

- Used to drop named schema elements, such as tables, domains, or constraint
- Drop behavior options:
  - `CASCADE : all elements are removed`
  - `RESTRICT : only if it has no elements in it`
- Example:
  - `DROP SCHEMA COMPANY CASCADE;`
  - `DROP TABLE DEPENDENT CASCADE;`

The ALTER Command

- **Alter table actions** include:
  - Adding or dropping a column (attribute)
  - Changing a column definition
  - Adding or dropping table constraints
- Example:
  - `ALTER TABLE COMPANY.EMPLOYEE ADD COLUMN Job VARCHAR(12);`
- To drop a column
  - Choose either `CASCADE` or `RESTRICT`

The ALTER Command (cont’d.)

- Change constraints specified on a table
  - Add or drop a named constraint

```
ALTER TABLE COMPANY.EMPLOYEE
DROP CONSTRAINT EMPSUPERFK CASCADE;
```

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>**
FROM	**<table list>**
[WHERE	**<condition>]**
[GROUP BY	**<grouping attribute(s)>]**
[HAVING	**<group condition>]**
[ORDER BY	**<attribute list>]**

- 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
- The order of execution:

  FROM ➔ WHERE ➔ GROUP BY ➔ HAVING ➔ SELECT ➔ UNION ➔ORDER BY