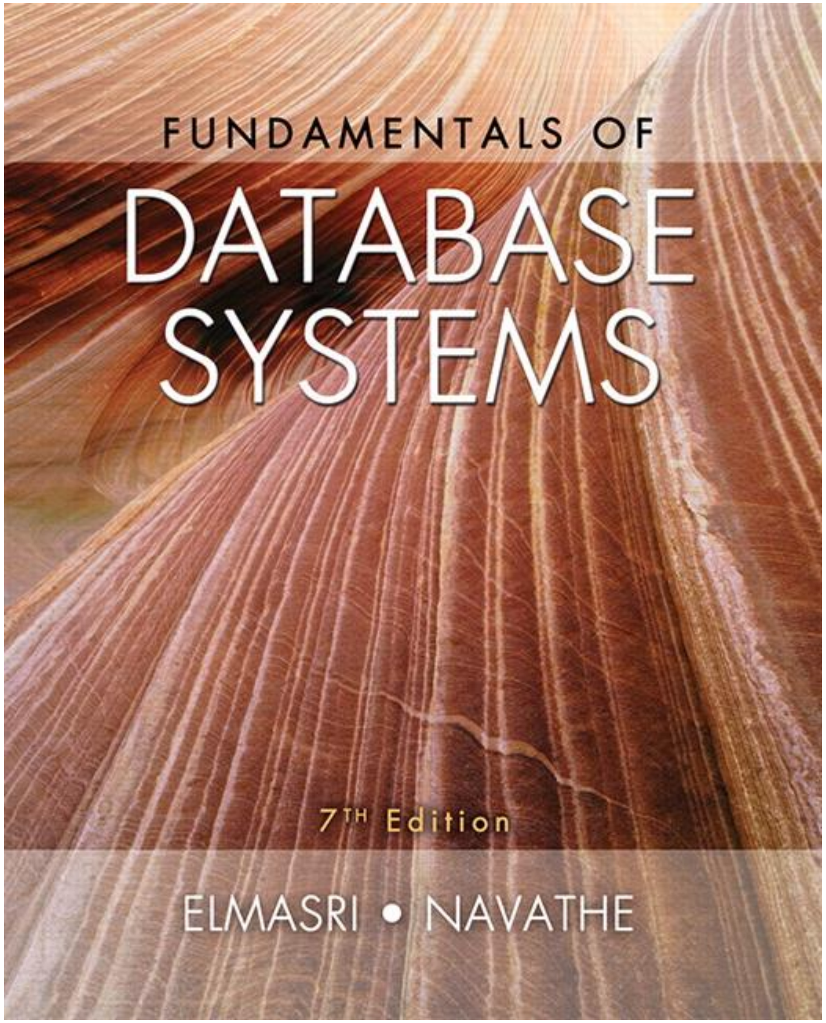

FUNDAMENTALS OF
DATABASE SYSTEMS
7TH Edition
ELMASRI • NAVATHE

CHAPTER 7

More SQL: Complex Queries, Triggers, Views, and Schema Modification

Chapter 7 Outline

- More Complex SQL Retrieval Queries
- Specifying Semantic Constraints as Assertions and Actions as Triggers
- Views (Virtual Tables) in SQL
- Schema Modification in SQL

More Complex SQL Retrieval Queries

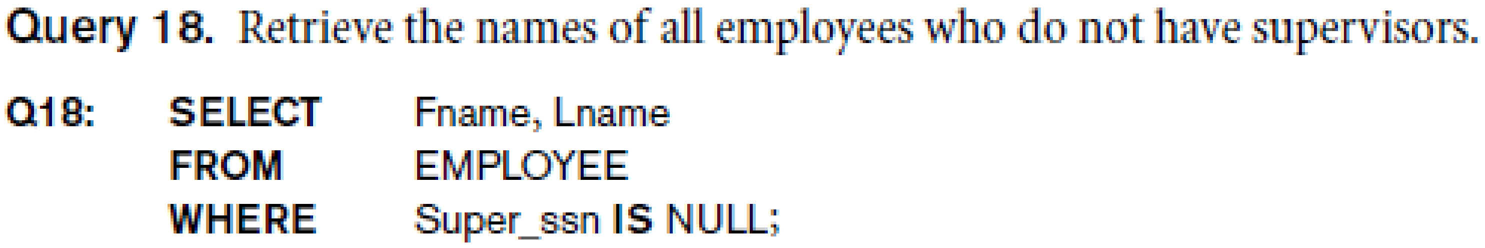
- Additional features allow users to specify more complex retrievals from database:
  - Nested queries, joined tables, and outer joins (in the FROM clause), aggregate functions, and grouping

Comparisons Involving NULL and Three-Valued Logic

- Meanings of `NULL`
  - **Unknown value**
  - **Unavailable or withheld value**
  - **Not applicable attribute**
- Each individual `NULL` value considered to be different from every other `NULL` value
- SQL uses a three-valued logic:
  - `TRUE`, `FALSE`, and `UNKNOWN` (like Maybe)
- **NULL = NULL comparison is avoided**

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`



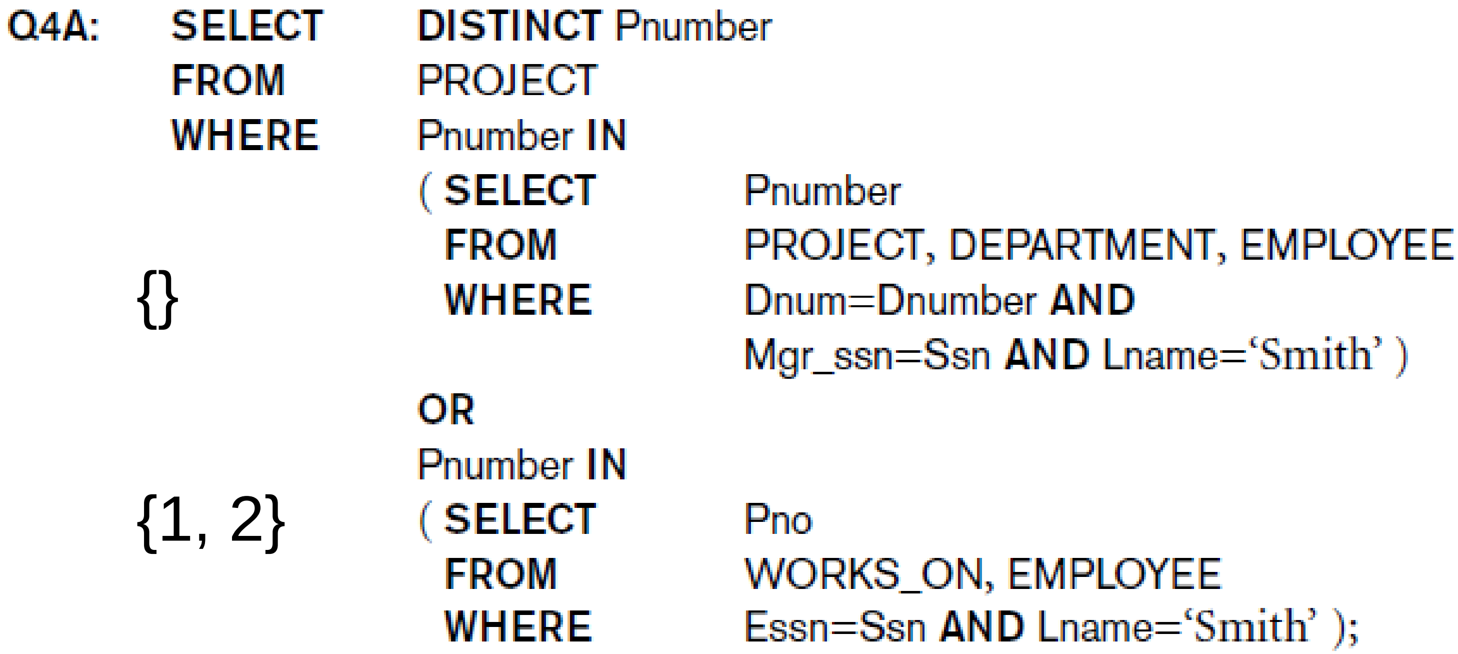
Query 18. Retrieve the names of all employees who do not have supervisors.

```
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 and nested subqueries**
- 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.)

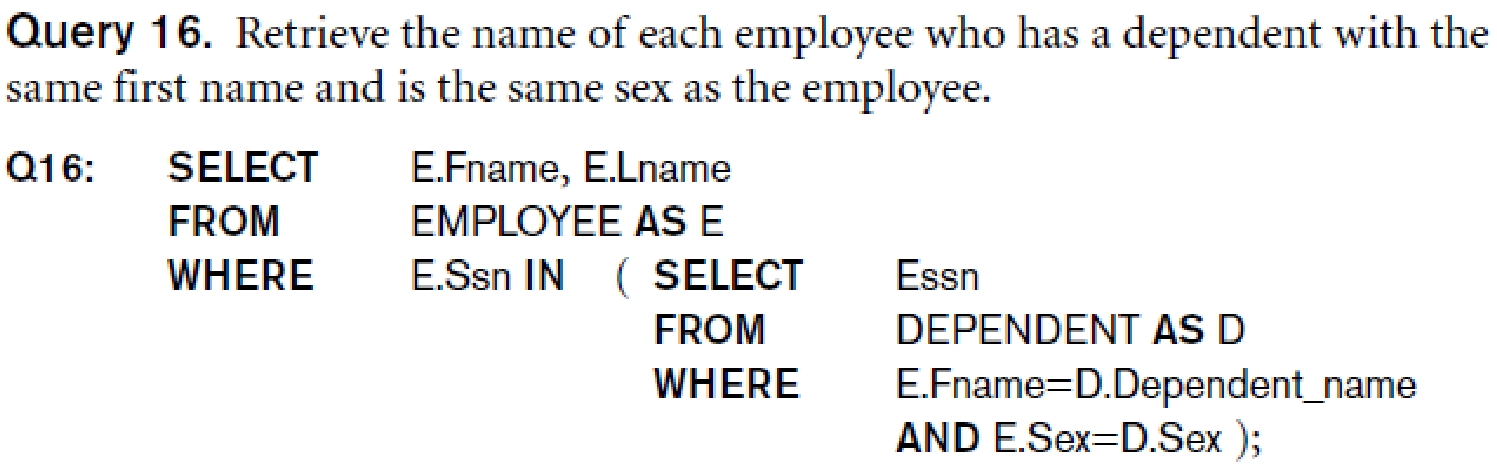


```
Q4A:    SELECT      DISTINCT Pnumber
        FROM        PROJECT
        WHERE       Pnumber IN
                    ( SELECT        Pnumber
                      FROM          PROJECT, DEPARTMENT, EMPLOYEE
      {}              WHERE         Dnum=Dnumber AND
                                    Mgr_ssn=Ssn AND Lname='Smith' )
                    OR
                    Pnumber IN
      {1, 2}        ( SELECT        Pno
                      FROM          WORKS_ON, EMPLOYEE
                      WHERE         Essn=Ssn AND Lname='Smith' );
```

{1, 2}

Nested Queries (cont'd.)

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

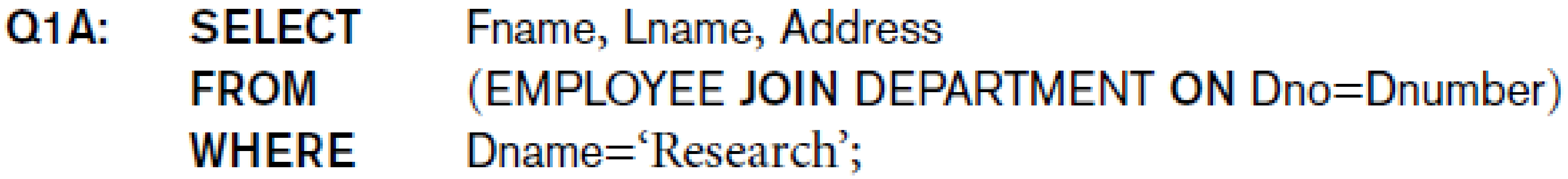


Query 16. Retrieve the name of each employee who has a dependent with the same first name and is the same sex as the employee.

```
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 );
```

Specifying Joined Tables in the FROM Clause of SQL

- **Joined table**
  - Permits users to specify a table resulting from a join operation in the FROM clause of a query
- The FROM clause in Q1A
  - Contains a single joined table. JOIN may also be called INNER JOIN



```
Q1A:    SELECT    Fname, Lname, Address
        FROM      (EMPLOYEE JOIN DEPARTMENT ON Dno=Dnumber)
        WHERE     Dname='Research';
```

FName	MInt	LName	Dno	Address	….	Dname	Dnumber	….
John	B	Smith	5	731 Fondern,..		Research	5	
Fraklin	T	Wong	5	638 Voss,..		Research	5	
Alicia	J	Zelaya	4	3321 Castle,..		Administration	4	
..			..					

Join

Different Types of JOINed Tables in SQL

- Specify different types of join
  - NATURAL JOIN
  - Various types of OUTER JOIN (LEFT, RIGHT, FULL)
- NATURAL JOIN on two relations R and S
  - No join condition specified
  - Is equivalent to an implicit EQUIJOIN condition for each pair of attributes with same name from R and S

NATURAL JOIN

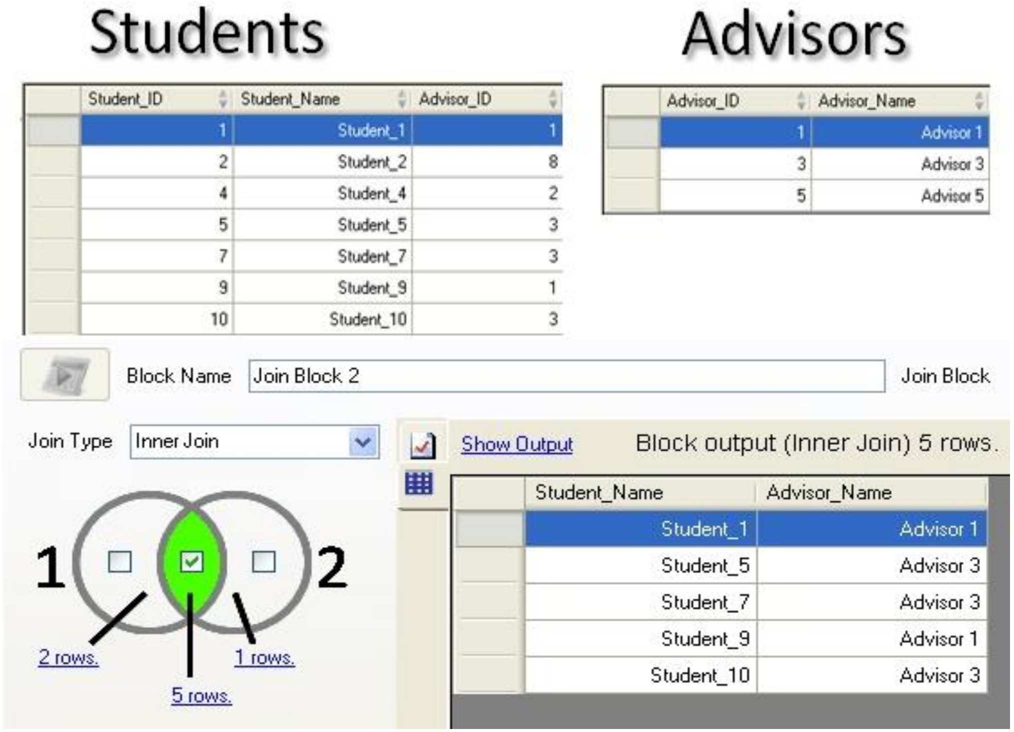
- Rename attributes of one relation so it can be joined with another using NATURAL JOIN:

```
Q1B:    SELECT    Fname, Lname, Address
        FROM      (EMPLOYEE NATURAL JOIN
                  (DEPARTMENT AS DEPT (Dname, Dno, Mssn,
                   Msdate)))
        WHERE      Dname='Research';
```

The above works with EMPLOYEE.Dno = DEPT.Dno as an implicit join condition

INNER and OUTER Joins

- INNER JOIN
  - Default type of join in a joined table
  - Tuple is included in the result only if a matching tuple exists in the other relation



Students

Student_ID	Student_Name	Advisor_ID
1	Student_1	1
2	Student_2	8
4	Student_4	2
5	Student_5	3
7	Student_7	3
9	Student_9	1
10	Student_10	3

Advisors

Advisor_ID	Advisor_Name
1	Advisor 1
3	Advisor 3
5	Advisor 5

Block Name: Join Block 2

Join Block

Join Type: Inner Join

Show Output

Block output (Inner Join) 5 rows.

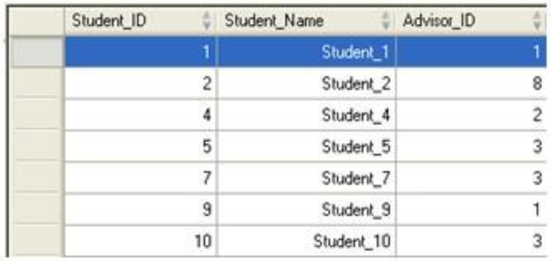
Student_Name	Advisor_Name
Student_1	Advisor 1
Student_5	Advisor 3
Student_7	Advisor 3
Student_9	Advisor 1
Student_10	Advisor 3

select Student_Name, Advisor_Name
from Students S INNER JOIN Advisors A ON S.Advisor_ID=A.Advisor_ID

INNER and OUTER Joins

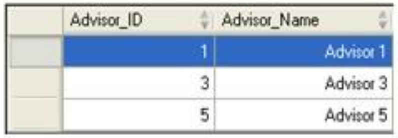
- LEFT OUTER JOIN
  - Every tuple in left table must appear in result
  - If no matching tuple
    - Padded with NULL values for attributes of right table

Students



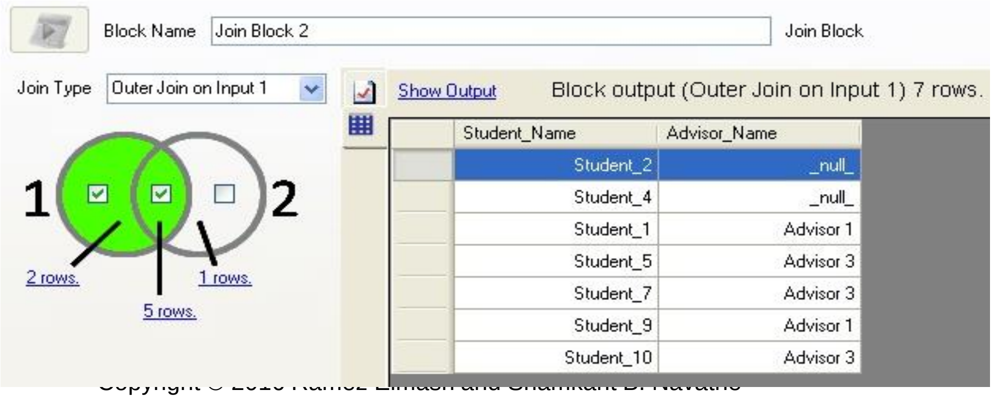
Student_ID	Student_Name	Advisor_ID
1	Student_1	1
2	Student_2	8
4	Student_4	2
5	Student_5	3
7	Student_7	3
9	Student_9	1
10	Student_10	3

Advisors



Advisor_ID	Advisor_Name
1	Advisor 1
3	Advisor 3
5	Advisor 5

```
select *
from
Students S
LEFT OUTER JOIN Advisors A
ON S.Advisor_ID=A.Advisor_ID
```



Block Name: Join Block 2

Join Block

Join Type: Outer Join on Input 1

Show Output

Block output (Outer Join on Input 1) 7 rows.

Student_Name	Advisor_Name
Student_2	_null_
Student_4	_null_
Student_1	Advisor 1
Student_5	Advisor 3
Student_7	Advisor 3
Student_9	Advisor 1
Student_10	Advisor 3

INNER and OUTER Joins

- LEFT OUTER JOIN
  - Every tuple in left table must appear in result
  - If no matching tuple
    - Padded with NULL values for attributes of right table
- RIGHT OUTER JOIN
  - Every tuple in right table must appear in result
  - If no matching tuple
    - Padded with NULL values for attributes of left table

Aggregate Functions in SQL

- Used to summarize information from multiple tuples into a single-tuple summary
- Built-in aggregate functions
  - **`COUNT`**, **`SUM`**, **`MAX`**, **`MIN`**, and **`AVG`**
- **Grouping**
  - Create subgroups of tuples before summarizing
- To select entire groups, `HAVING` clause is used
- Aggregate functions can be used in the `SELECT` clause or in a `HAVING` clause

Renaming Results of Aggregation

- Following query returns a single row of computed values from EMPLOYEE table:

```
Q19:        SELECT    SUM (Salary), MAX (Salary), MIN (Salary), AVG
                       (Salary)
             FROM      EMPLOYEE;
```

- The result can be presented with new names:

```
Q19A:        SELECT    SUM (Salary) AS Total_Sal, MAX (Salary) AS
                       Highest_Sal, MIN (Salary) AS Lowest_Sal, AVG
                       (Salary) AS Average_Sal
             FROM      EMPLOYEE;
```

Aggregate functions

- For example: get all employees whose salary is > than 30.
- Some advanced operations may address sets of tuples.
- For example: how many employees have a salary > than 30?
- SQL provides this functionality through aggregate functions.

```
select *
from Employee
where Salary > 30
```

Name	*Surname*	*Department*	*Supervisor*	*Salary*
John	White	1	2	36
Mark	Frank	1	3	46
~~Moan~~	~~Jones~~	~~2~~	~~1~~	~~27~~

Example

- Select the number of employees working at Department number 1.

Evaluating aggregate queries (1)

```
select          *
from Employee
where Department = 1
```

Name	*Surname*	*Department*	*Supervisor*	*Salary*
John	White	1	2	36
Mark	Frank	1	3	46

Evaluating aggregate queries (2)

```
select count(*) AS numberOfEmployees
from Employee
where Department = 1
```

Name	Surname	Department	Supervisor	Salary
John	White	1	2	36
Mark	Frank	1	3	46

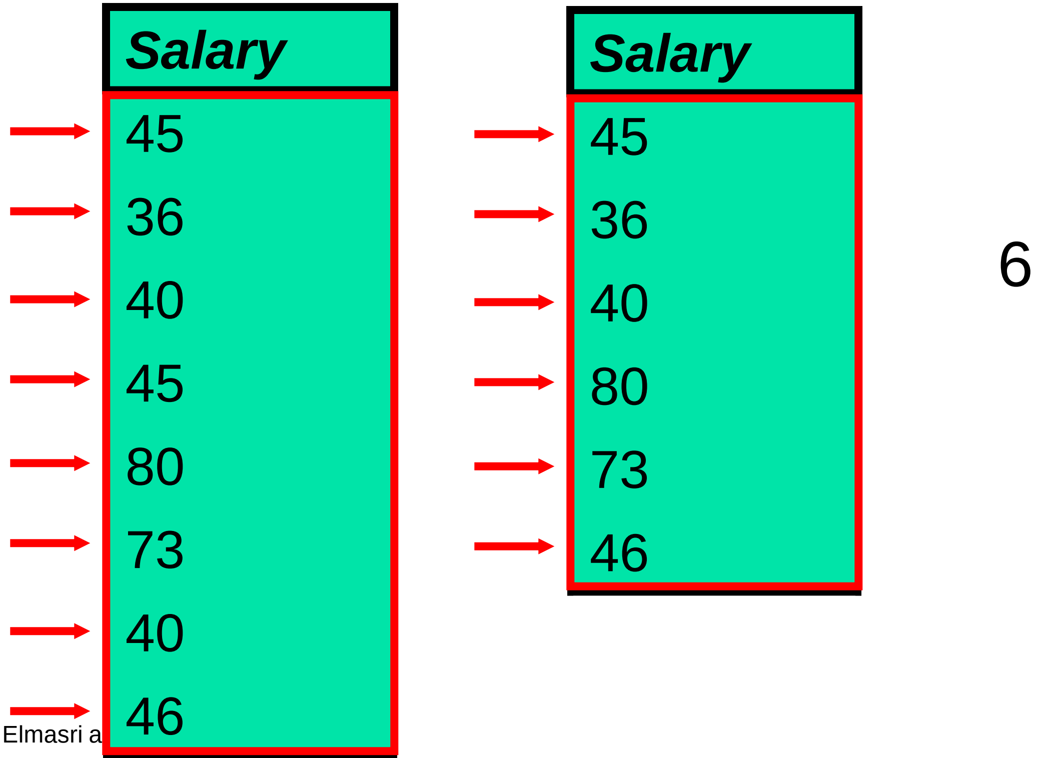
numberOfEmployees
2

Other standard aggregate functions

- count, sum, max, min, avg.
- Check the manual of the system you want to use for other options.

Target of the aggregate function

select count(distinct salary) AS numOfDistinctSalaries
from Employee



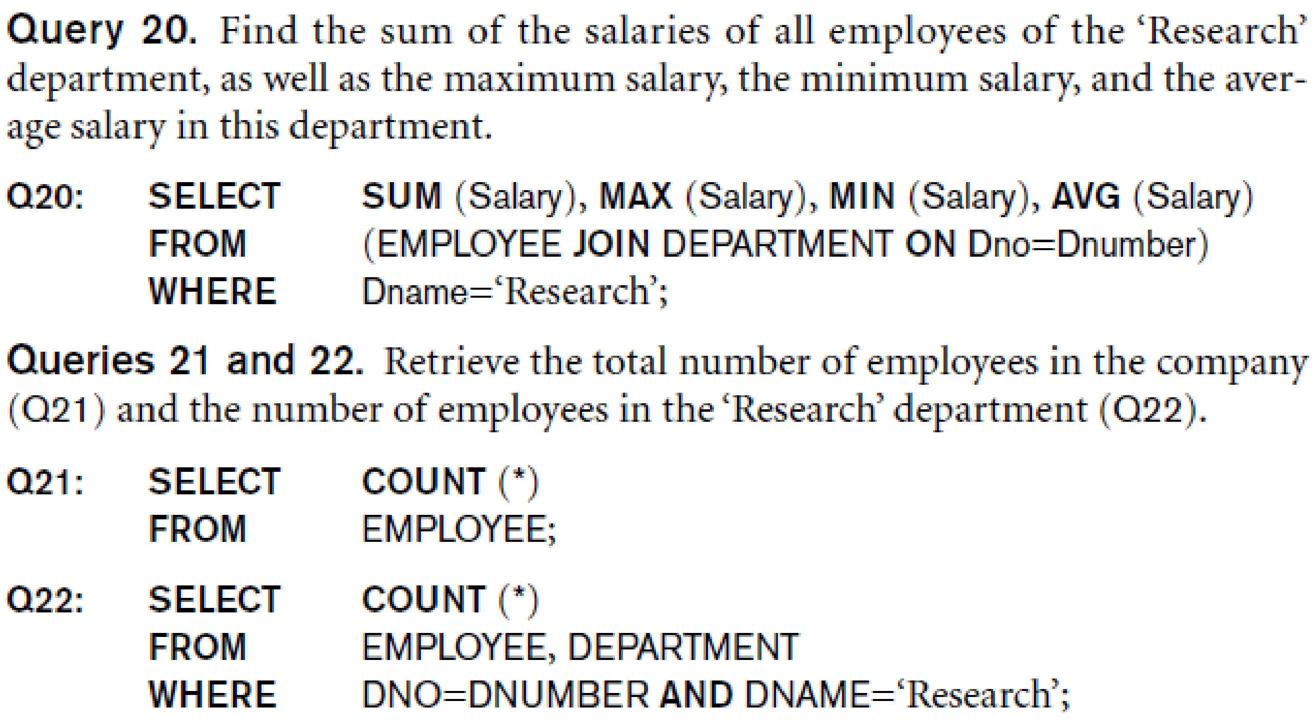
Salary
45
36
40
45
80
73
40
46

Salary
45
36
40
80
73
46

6

Aggregate Functions in SQL (cont'd.)

- NULL values are 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';
```

Grouping: The GROUP BY Clause

- **Partition** relation into subsets of tuples
  - Based on **grouping attribute(s)**
  - Apply function to each such group independently
- **`GROUP BY`** clause
  - Specifies grouping attributes
- COUNT (*) counts the number of rows in the group

Sum of salaries per department

select Department, Salary

from Employee

Department	*Salary*
1	45
2	36
1	40
3	45
4	80
4	73
1	40
2	46

Sum of salaries per department

```
select Department,        Salary
from Employee
GROUP BY Department
```

Department	***Salary***
1	45
1	40
1	40
2	36
2	46
3	45
4	80
4	73

Sum of salaries per department

```
select Department, sum(Salary)
from Employee
GROUP BY Department
```

Department	***Salary***
1	45
1	40
1	40
2	36
2	46
3	45
4	80
4	73

Department	
1	125
2	82
3	45
4	153

Sum of salaries per department

```
select Department, sum(Salary) AS allSalary
from Employee
GROUP BY Department
```

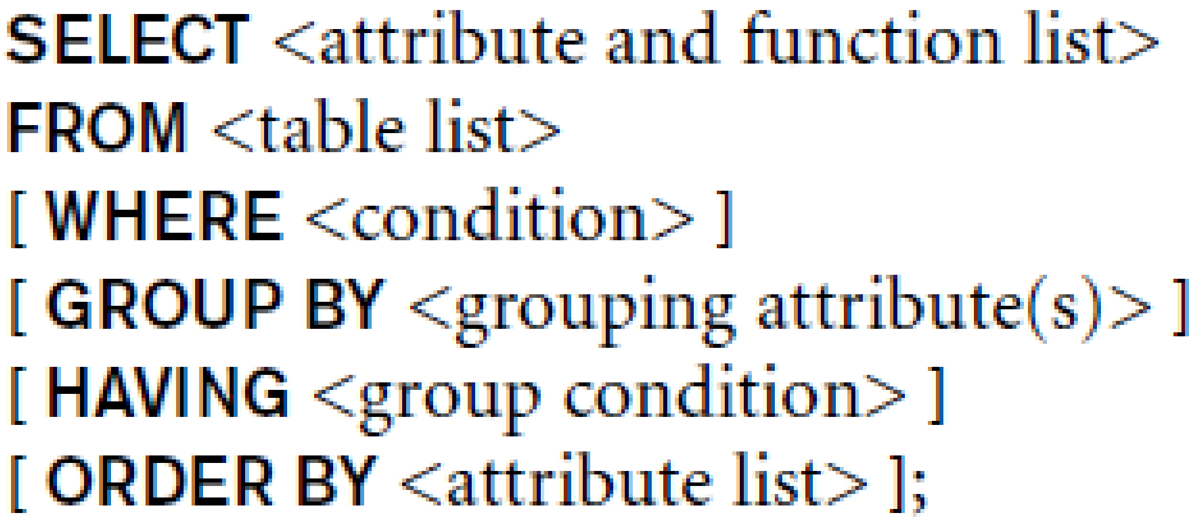
Department	***allSalary***
1	125
2	82
3	45
4	153

Predicates on groups

- **HAVING** clause
  - Provides a condition to select or reject an entire group:

```
select Department, sum(Salary)
from Employee
group by Department
HAVING sum(Salary) > 100
```

EXPANDED Block Structure 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> ];
```

Combining the WHERE and the HAVING Clause

- Consider the query: we want to count the *total* number of employees whose salaries exceed $40,000 in each department, but only for departments where more than five employees work.

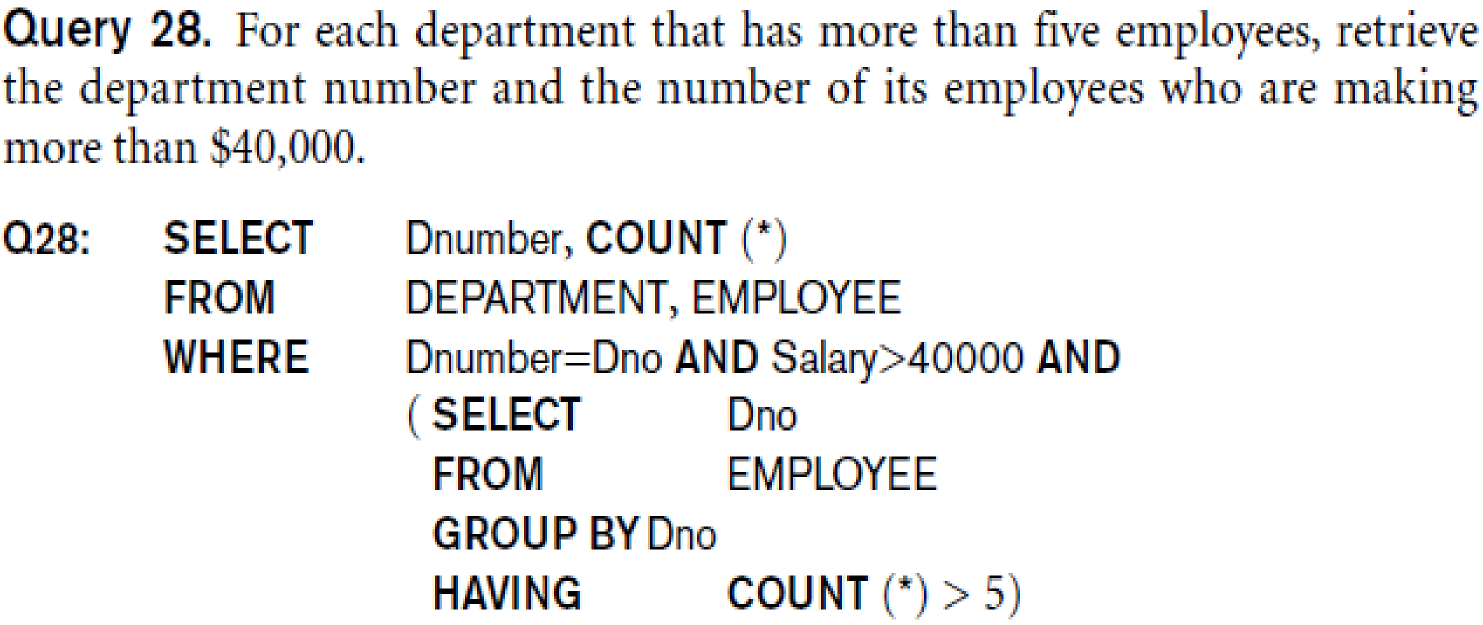
- INCORRECT QUERY:

```
SELECT      Dno, COUNT (*)
FROM        EMPLOYEE
WHERE       Salary>40000
GROUP BY    Dno
HAVING      COUNT (*) > 5;
```

Combining the WHERE and the HAVING Clause (continued)

Correct Specification of the Query:

- Note: the WHERE clause applies tuple by tuple whereas HAVING applies to entire group of tuples



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.

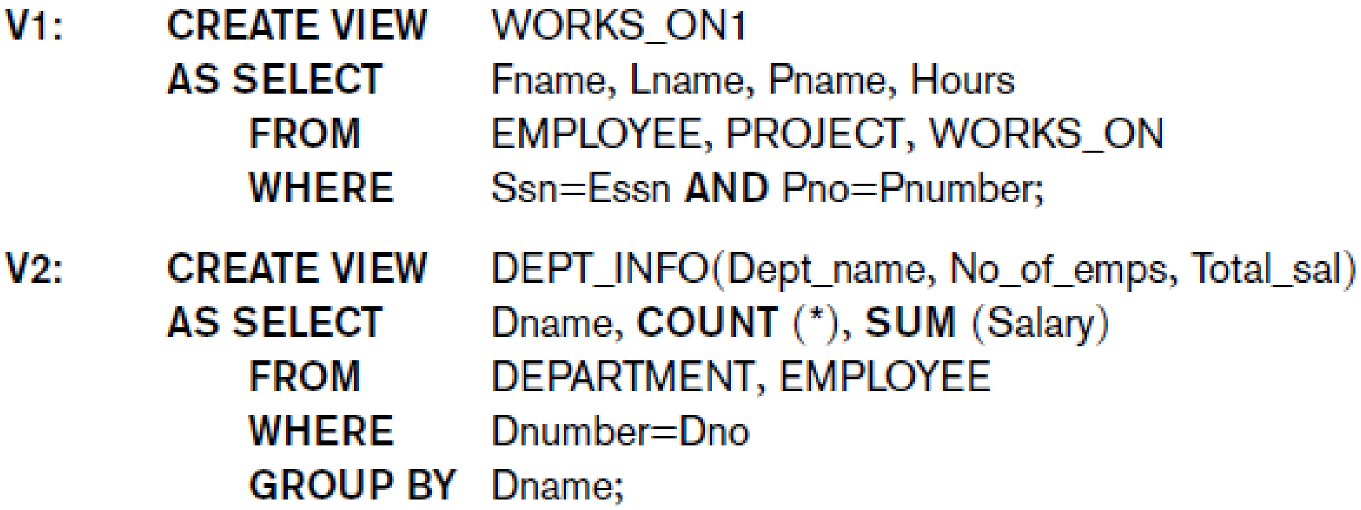
```
Q28:    SELECT    Dnumber, COUNT (*)
        FROM      DEPARTMENT, EMPLOYEE
        WHERE     Dnumber=Dno AND Salary>40000 AND
                  ( SELECT       Dno
                    FROM         EMPLOYEE
                    GROUP BY Dno
                    HAVING       COUNT (*) > 5)
```

Views (Virtual Tables) in SQL

- Concept of a view in SQL
  - Single table derived from other tables called the **defining tables**
  - Considered to be a virtual table that is not necessarily populated

Specification of Views in SQL

- **`CREATE VIEW`** command
- **View** – a single table derived from other tables.
- The tables can be **base tables** or other previously defined views.
  - Give table name, list of attribute names, and a query to specify the contents of the view
  - In V1, attributes retain the names from base tables. In V2, attributes are assigned names



```
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.)

- Once a View is defined, SQL queries can use the View relation in the FROM clause
- View is always up-to-date
    - Responsibility of the DBMS and not the user
- **`DROP VIEW`** command
    - Dispose of a view

The DROP Command

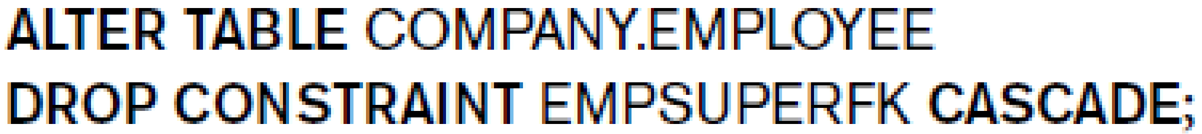
- `DROP` command
  - Used to drop named schema elements, such as tables, domains, or constraint
- Drop behavior options:
  - `CASCADE` and `RESTRICT`
- Example:
  - `DROP SCHEMA COMPANY CASCADE;`
  - This removes the schema and all its elements including tables, views, constraints, etc.

The ALTER table 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);`

Adding and Dropping Constraints

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



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

Summary

- Complex SQL:
  - Nested queries, joined tables (in the FROM clause), outer joins, aggregate functions, grouping
- Handling semantic constraints with `CREATE ASSERTION` and `CREATE TRIGGER`
- `CREATE VIEW` statement and materialization strategies
- Schema Modification for the DBAs using `ALTER TABLE , ADD and DROP COLUMN, ALTER CONSTRAINT` etc.