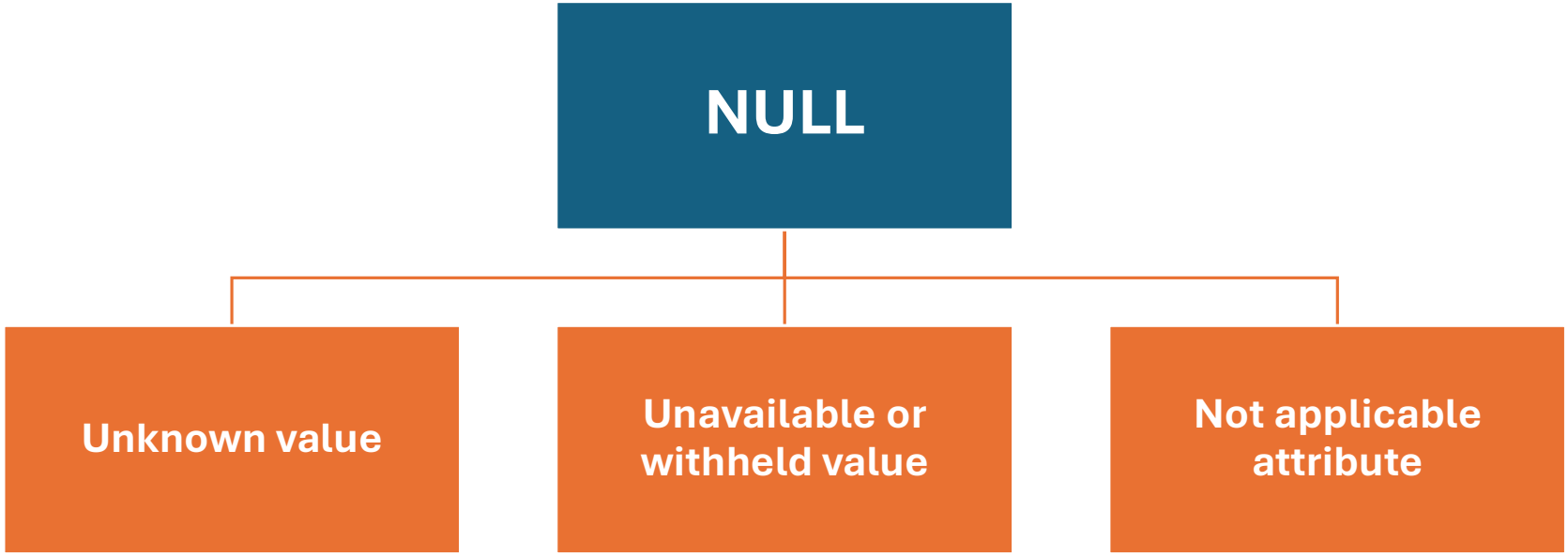
Lecture 28

Advanced SQL

Comparisons Involving NULL and Three-Valued Logic



- Each individual NULL value considered to be different from every other NULL value
- **SQL uses a three-valued logic: TRUE, FALSE, and UNKNOWN**

Logical Connectives in Three –Valued Logic

(a)

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

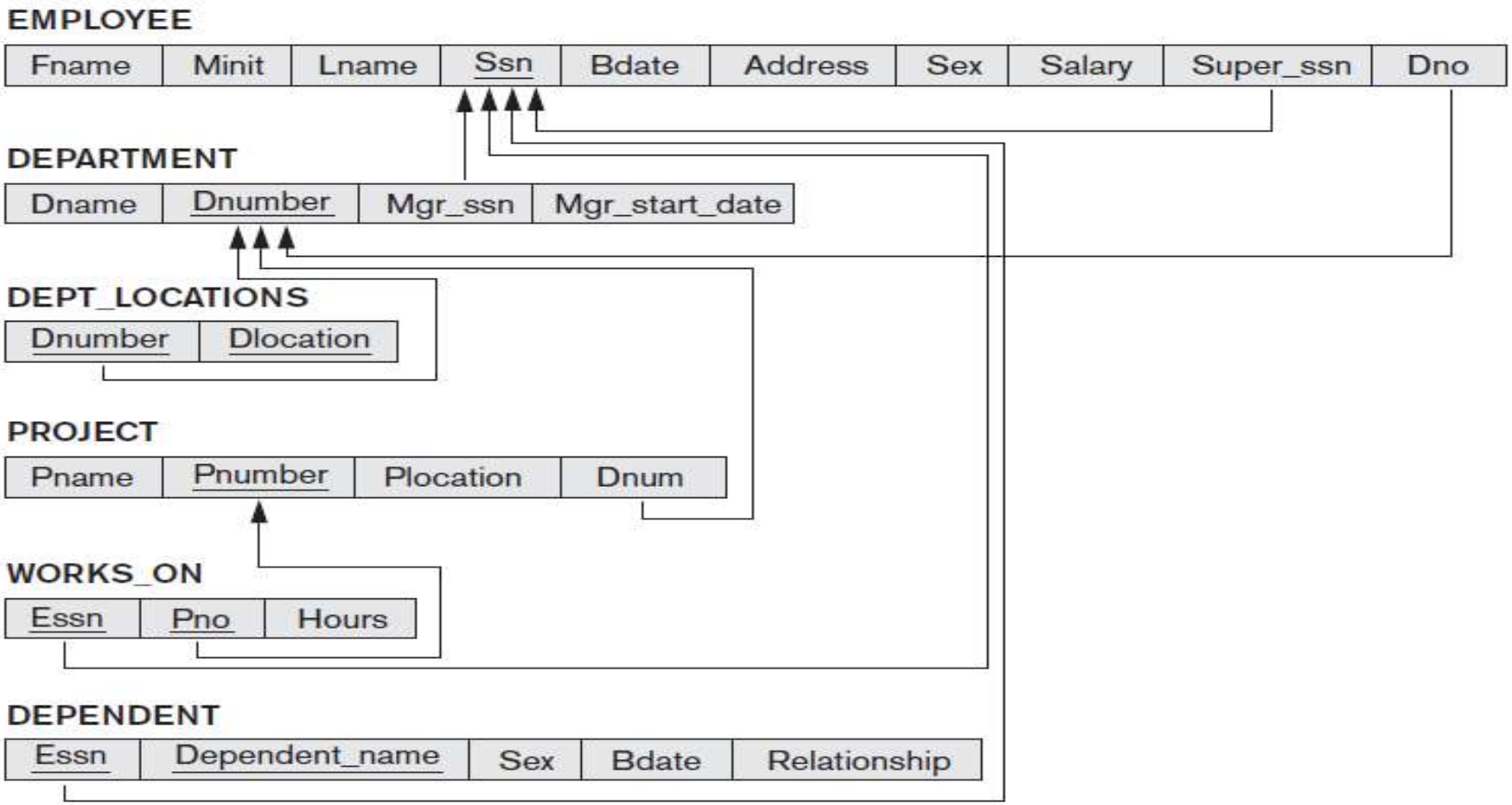
(b)

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

(c)

NOT	
TRUE	FALSE
FALSE	TRUE
UNKNOWN	UNKNOWN

COMPANY Database



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**

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

- **Query:**

```
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
- Nested Queries generally return a table (relation)
- **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*

Example - Nested Queries

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.

Query:

```
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' );
```

Selects the project numbers of projects that have an employee with last name 'Smith' involved as manager

Selects the project numbers of projects that have an employee with last name 'Smith' involved as worker

Example - Nested Queries (Cont.)

Select the Essns of all employees who work on the same project and hours as some project that employee 'John Smith' (whose Ssn = '123456789') works on.

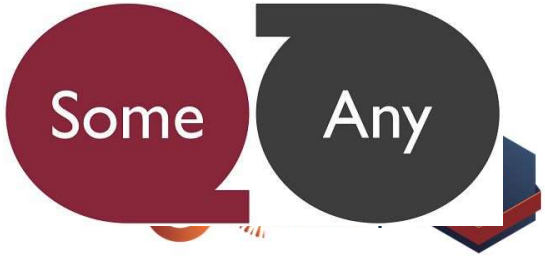
Query:

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

Parentheses are important to compare combination of Pno and Hours attributes

Comparison Operators to Compare a single value – ANY & ALL

- **= ANY (or = SOME) operator:** Returns TRUE if the value *v* is equal to some value in the set *V* (equivalent to IN)
- **ALL operator:** (*v* > ALL *V*) returns TRUE if the value *v* is greater than *all* the values in the set (or multiset) *V*.
- Other operators that can be combined with ANY (or SOME) and ALL: **>, >=, <, <=, and <>**



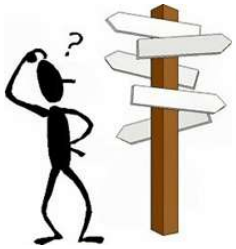
Example - ALL Operator

Return the names of employees whose salary is greater than the salary of all the employees in department 5

Query:

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

Ambiguity in Nested Queries



- Possible ambiguity among attribute names if attributes of the same name exist—one in a relation in the FROM clause of the outer query, and another in a relation in the FROM clause of the nested query.
- **Thumb Rule:** reference to an unqualified attribute refers to the relation declared in the innermost nested query.

Example - Ambiguity in Nested Queries

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

Query:

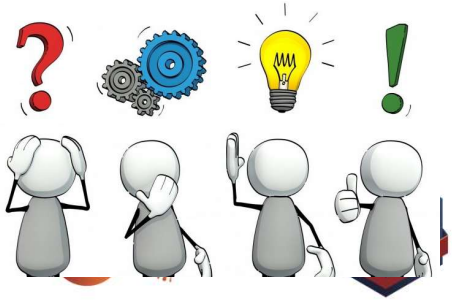
```
SELECT E.Fname, E.Lname
FROM EMPLOYEE AS E
WHERE E.Ssn IN ( SELECT Essn
                ROM DEPENDENT AS D
                ependent_name AND E.Sex = Sex);
```

- We must qualify E.Sex because it refers to the Sex attribute of EMPLOYEE from the outer query, and DEPENDENT also has an attribute Sex.
- Here, the unqualified references to Sex in the nested query, refer to the Sex attribute of DEPENDENT

No need to qualify Fname and Ssn of EMPLOYEE if they appeared in the nested query because the DEPENDENT relation does not have Fname and Ssn attributes, so there is no ambiguity.

Correlated Nested Queries

- Whenever a condition in the WHERE clause of a nested query references some attribute of a relation declared in the outer query, the two queries are said to be **correlated**.
- In a correlated query, the nested query is *evaluated once for each tuple (or combination of tuples) in the outer query*



Example - Correlated Nested Queries

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

Query:

```
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);
```

- For *each* EMPLOYEE tuple, evaluate the nested query, which retrieves the Essn values for all DEPENDENT tuples with the same sex and name as that EMPLOYEE tuple.
- If the Ssn value of the EMPLOYEE tuple is *in* the result of the nested query, then select that EMPLOYEE tuple

Nested Queries (Cont.)

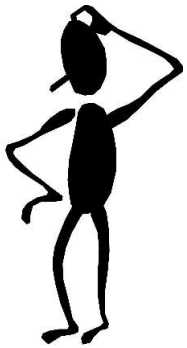
Retrieve the name of each employee who has a dependent with the same first name and is the same sex as the employee.

```
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);
```

==========

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

EXISTS Function



- **EXISTS function:** Check whether the result of a correlated nested query is empty or not
- EXISTS and NOT EXISTS are typically used in conjunction with a correlated nested query
- **EXISTS(Q):** returns TRUE if there is at least one tuple in the result of the nested query Q, and it returns FALSE otherwise.
- **NOT EXISTS(Q):** returns TRUE if there are no tuples in the result of nested query Q, and it returns FALSE otherwise.

Example – EXISTS

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

- For each EMPLOYEE tuple, evaluate the nested query, which retrieves all DEPENDENT tuples with the same Essn, Sex, and Dependent_name as the EMPLOYEE tuple.
- If at least one tuple EXISTS in the result of the nested query, then select that EMPLOYEE tuple.

```
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);
```

==========

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

Example – NOT EXISTS

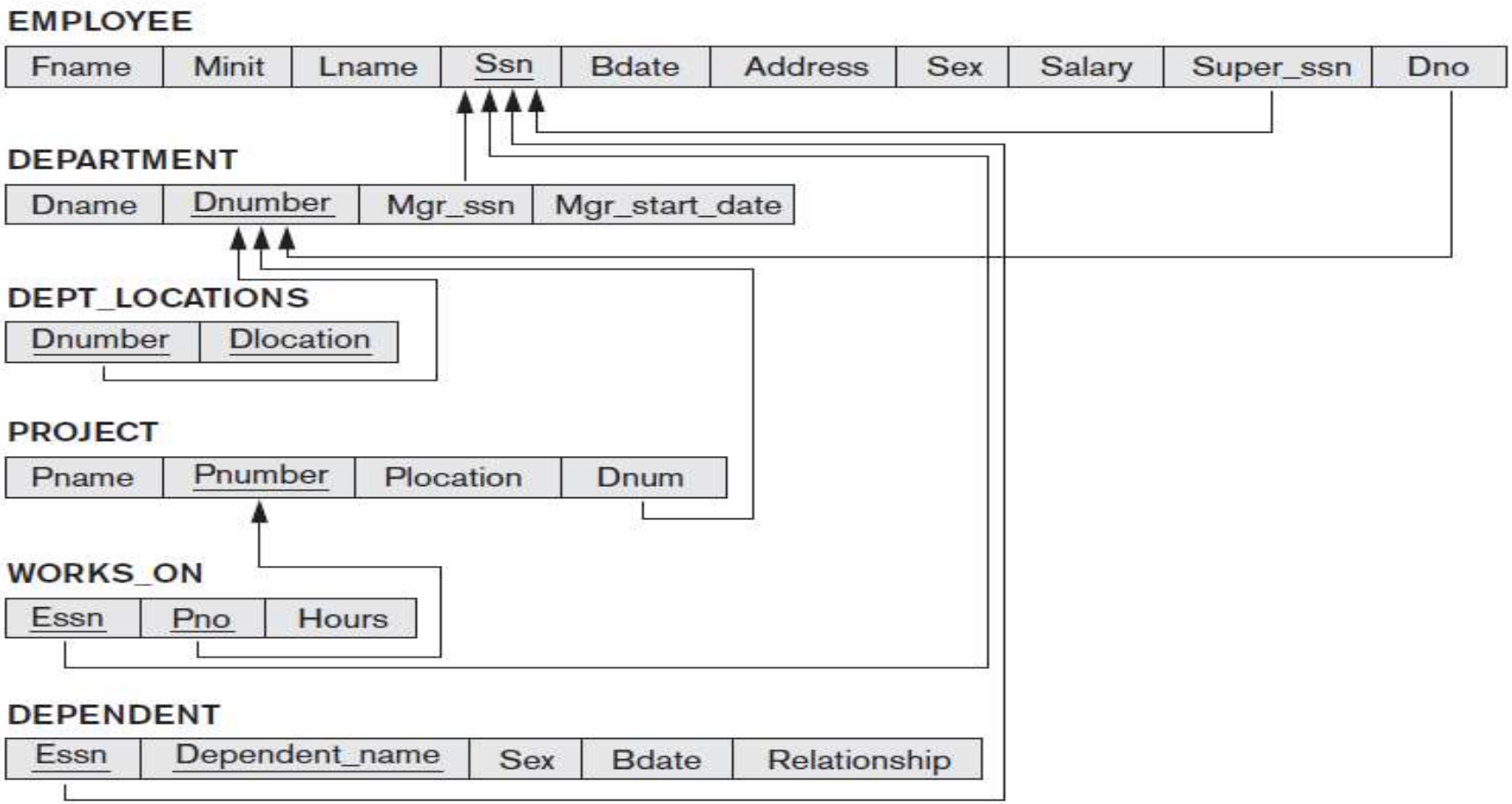
Retrieve the names of employees who have no dependents.

Query:

```
SELECT Fname, Lname

FROM EMPLOYEE

WHERE NOT EXISTS ( SELECT *

                   FROM DEPENDENT WHERE Ssn=Essn );
```

- For each EMPLOYEE tuple, the correlated nested query selects all DEPENDENT tuples whose Essn value matches the EMPLOYEE Ssn.
- If the result is empty, no dependents are related to the employee, so we select that EMPLOYEE tuple and retrieve its Fname and Lname.

COMPANY Database



Practice Drill

1. List the names of managers who have at least one dependent using EXISTS and NOT EXISTS functions

2. Retrieve the name of each employee who works on all the projects controlled by department number 5 using EXISTS and NOT EXISTS functions

Solution – Practice Drill

1. **List the names of managers who have at least one dependent**

```
SELECT Fname, Lname
FROM EMPLOYEE
WHERE EXISTS ( SELECT *
                FROM DEPENDENT
                WHERE Ssn=Essn )

AND EXISTS ( SELECT *
                FROM DEPARTMENT WHERE Ssn=Mgr_ssn );
```

Selects all DEPENDENT tuples related to an EMPLOYEE

Selects all DEPARTMENT tuples managed by the EMPLOYEE

Solution – Practice Drill

2. **Retrieve the name of each employee who works on all the projects controlled by department number 5**

```
SELECT Fname, Lname
FROM EMPLOYEE
WHERE NOT EXISTS ( ( SELECT Pnumber
                     FROM PROJECT
                     WHERE Dnum=5)
                     EXCEPT ( SELECT Pno
                              FROM WORKS_ON
                     WHERE Ssn=Essn) )
```

Selects all projects controlled by department 5 (not correlated to outer query)

Selects all projects that the particular employee being considered works on (correlated to outer query)

If the set difference of the first subquery result MINUS (EXCEPT) the second subquery result is empty, it means that the employee works on all the projects and is therefore selected.

Solution – Practice Drill

2. **Retrieve the name of each employee who works on all the projects controlled by department number 5**

```
SELECT Lname, Fname
FROM EMPLOYEE
WHERE NOT EXISTS ( SELECT * FROM WORKS_ON B
                   WHERE ( B.Pno IN ( SELECT Pnumber
                                      FROM PROJECT WHERE Dnum=5 )
AND NOT EXISTS ( SELECT * FROM WORKS_ON C
WHERE C.Essn=Ssn AND C.Pno=B.Pno )));
```

UNIQUE Function

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

```
SELECT FName, LName
FROM EMPLOYEE
WHERE UNIQUE (SELECT ESsn
              FROM WORKS_ON
              WHERE WORKS_on.ESsn = EMPLOYEE.Ssn);
```

Retrieves the names of all employees who work on only one project

Explicit Sets in WHERE Clause

Can use explicit set of values in WHERE clause

Example: Retrieve the Social Security numbers of all employees who work on project numbers 1, 2, or 3.

Query:

```
SELECT DISTINCT Essn
FROM WORKS_ON
WHERE Pno IN (1, 2, 3);
```

Renaming of Attributes

Rename any attribute that appears in the result of a query using "AS" qualifier followed by desired new name

Example:

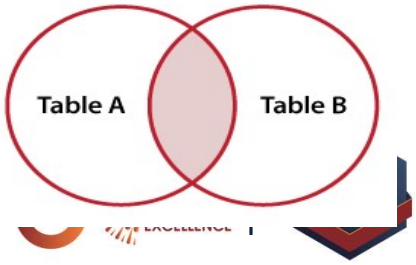
SELECT E.Lname **AS** Employee_name, S.Lname **AS** Supervisor_name

FROM EMPLOYEE **AS** E, EMPLOYEE **AS** S

WHERE E.Super_ssn=S.Ssn;

Joined Tables

- **Joined Table:** Permits users to specify a table resulting from a join operation in the FROM clause of a query
- The attributes of such a table are all the attributes of the first table followed by all the attributes of the second table.
- **The default type of join in a joined table is called an inner join**, where a tuple is included in the result only if a matching tuple exists in the other relation.



Example - Joined Tables

Retrieve the name and address of every employee who works for the 'Research' department.

Query:

SELECT Fname, Lname, Address

FROM (EMPLOYEE **JOIN** DEPARTMENT **ON** Dno=Dnumber)

WHERE Dname='Research';

Natural Join

- **NATURAL JOIN** on two relations R and S
  - No join condition specified
  - **Implicit EQUIJOIN condition for each pair of attributes with same name from R and S**

- It is possible to rename the attributes so that they match, if the names of the join attributes are not the same in the base relations.

Example – Natural join

Retrieve the name and address of every employee who works for the 'Research' department.

Query:

SELECT Fname, Lname, Address

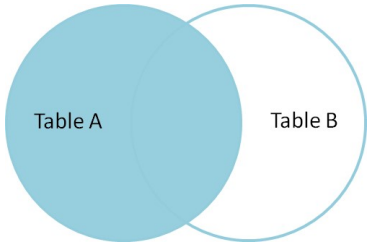
FROM (EMPLOYEE **NATURAL JOIN**

(DEPARTMENT **AS** DEPT (Dname, Dno, Mssn, Msdate)))

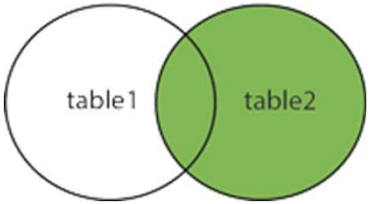
WHERE Dname='Research';

Outer Join

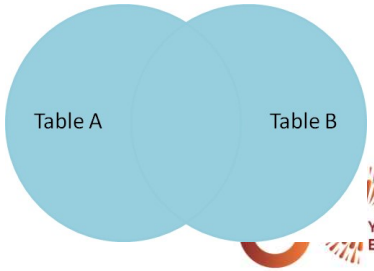
- **LEFT OUTER JOIN**
  - Every tuple in left table must appear in result
  - If no matching tuple, padded with NULL values for attributes of right table



- **RIGHT OUTER JOIN**
  - Every tuple in right table must appear in result
  - If no matching tuple, padded with NULL values for the attributes of left table



- **FULL OUTER JOIN**
- **CROSS JOIN** – for Cartesian Product

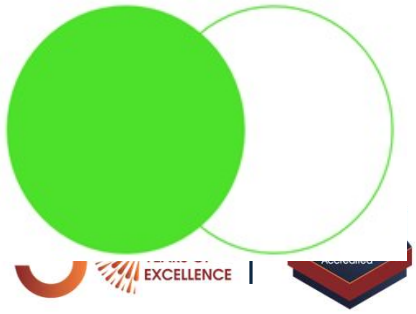


Example – Outer Join

List employees who have a supervisor and for those who do not have supervisor indicate a NULL value

Query:

```
SELECT E.Lname AS Employee_name, S.Lname AS Supervisor_name
FROM (EMPLOYEE AS E LEFT OUTER JOIN EMPLOYEE AS S
ON E.Super_ssn=S.Ssn);
```



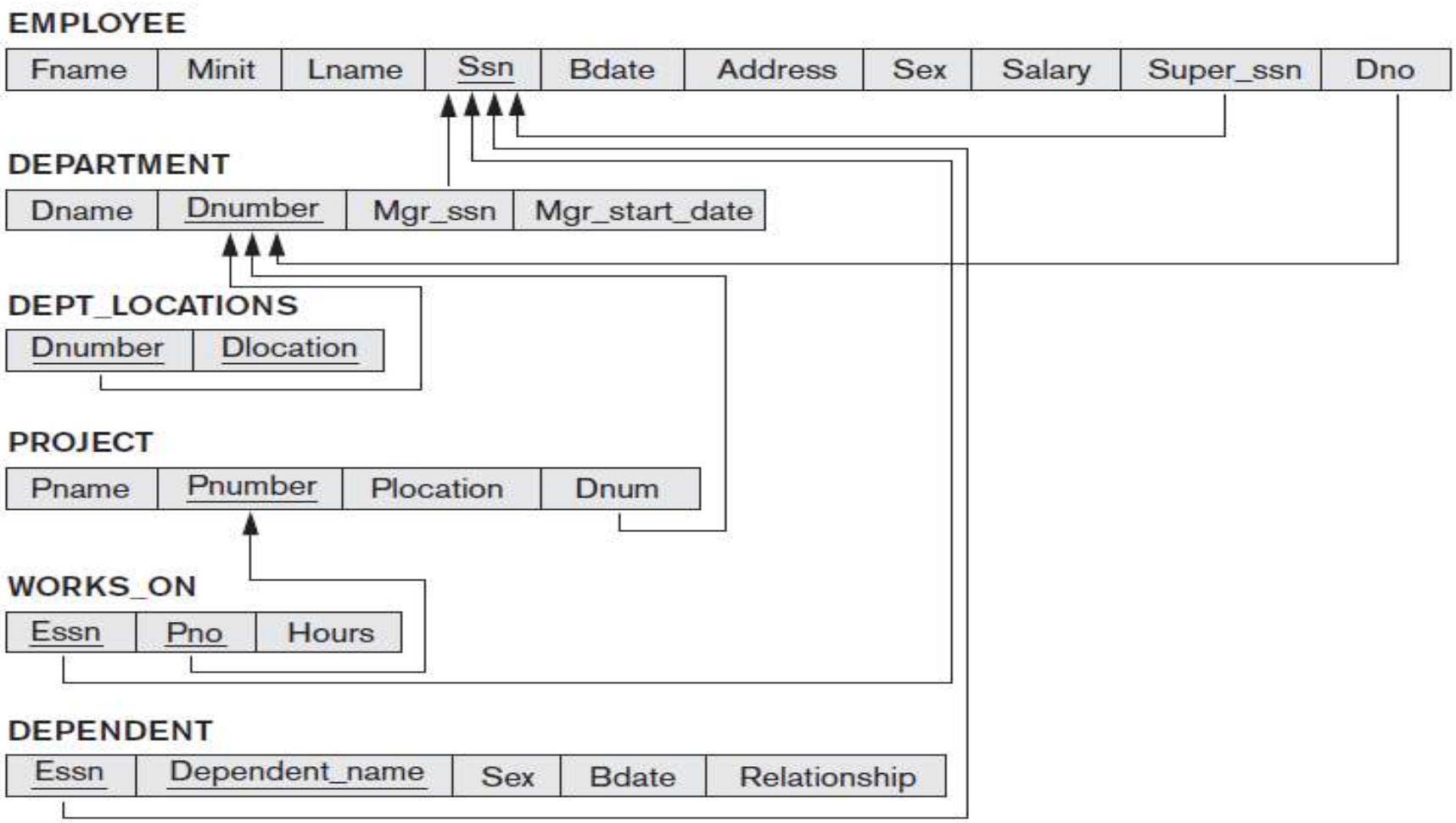
Nesting with Join Operation

For every project located in ‘Stafford’, list the project number, the controlling department number, and the department manager’s last name, address, and birth date.

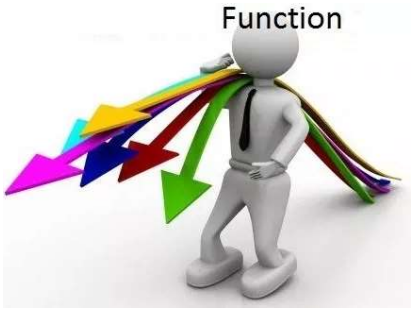
Query:

```
SELECT Pnumber, Dnum, Lname, Address, Bdate
FROM ((PROJECT JOIN DEPARTMENT ON Dnum=Dnumber)
                    JOIN EMPLOYEE ON Mgr_ssn=Ssn)
        WHERE Plocation='Stafford';
```

COMPANY Database



Aggregate Functions



- Used to summarize information

  from multiple tuples into a single-tuple summary

- **Grouping**
  - Create subgroups of tuples before summarizing
- **Built-in aggregate functions:** **COUNT**, **SUM**, **MAX**, **MIN**, and **AVG** **(NULL values discarded when aggregate functions are applied to a particular column)**
- **Functions can be used in the SELECT clause or in a HAVING clause**

Example - Aggregate Functions

- Find the sum of the salaries of all employees, the maximum salary, the minimum salary and the average salary.

Aggregate Functions

Query

SELECT SUM (Salary), **MAX** (Salary), **MIN** (Salary), **AVG** (Salary)

FROM EMPLOYEE;

Example - Aggregate Functions (Cont.)

- 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.

Aggregate Functions

Query

SELECT **SUM** (Salary), **MAX** (Salary), **MIN** (Salary), **AVG** (Salary)

FROM (EMPLOYEE **JOIN** DEPARTMENT **ON** Dno=Dnumber)

WHERE Dname='Research';

Example - Aggregate Functions (Cont.)

- Retrieve the number of employees in the 'Research' department

Aggregate Functions

Refers to the rows

Query

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

Example - Aggregate Functions (Cont.)

- Count the number of distinct salary values in the database.

Aggregate Functions
(will not count tuples with NULL)

Query

SELECT COUNT (**DISTINCT** Salary)

FROM EMPLOYEE;

Eliminates duplicate values

Example - Aggregate Functions (Cont.)

- Retrieve the names of all employees who have two or more dependents

Query

SELECT Lname, Fname

FROM EMPLOYEE

WHERE (**SELECT** **COUNT** (*)

FROM DEPENDENT

WHERE Ssn=Essn) >= 2;

Aggregate Functions

Counts the number of dependents that each employee has; if this is greater than or equal to two, the employee tuple is selected

The GROUP BY and HAVING Clauses

- **Partition** relation into subsets of tuples
  - Based on **grouping attribute(s)**
  - Apply function to each such group independently
- **GROUP BY** clause: Specifies grouping attributes
- **If NULLs** exist in grouping attribute
  - Separate group created for all tuples with

    a NULL value in grouping attribute
- **HAVING** clause
  - Provides a condition on the summary information

Example

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

Query:

SELECT Dno, **COUNT** (*), **AVG** (Salary)

FROM EMPLOYEE

GROUP BY Dno;

SELECT clause includes only the grouping attribute and the aggregate functions to be applied on each group of tuples.

Result

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

Grouping EMPLOYEE tuples by the value of Dno

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

Result of Q24

Example

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

Query:

SELECT Pnumber, Pname, **COUNT** (*)

FROM PROJECT, WORKS_ON

WHERE Pnumber=Pno

GROUP BY Pnumber, Pname;

Example

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

Query:

SELECT Pnumber, Pname, **COUNT** (*)

FROM PROJECT, WORKS_ON

WHERE Pnumber=Pno

GROUP BY Pnumber, Pname

HAVING COUNT (*) > 2;

WHERE clause limit the *tuples* to which functions are applied, the HAVING clause serves to choose *whole groups*

Result

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

After applying the WHERE clause but before applying HAVING

These groups are not selected by the HAVING condition

Result (Cont.)

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

After applying the HAVING clause condition

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

Result of Q26
(Pnumber not shown)

Example

- For each project, retrieve the project number, the project name and the number of employees from department 5 who work on the project.

Query:

SELECT Pnumber, Pname, **COUNT** (*)

FROM PROJECT, WORKS_ON, EMPLOYEE

WHERE Pnumber=Pno **AND** Ssn=Essn **AND** Dno=5

GROUP BY Pnumber, Pname;

Example

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

Query:

SELECT Dname, **COUNT** (*)

FROM DEPARTMENT, EMPLOYEE

WHERE Dnumber=Dno **AND** Salary>40000

GROUP BY Dname

HAVING COUNT (*) > 5;

It will select only departments that have more than five employees *who each earn more than $40,000*. The tuples are already restricted to employees who earn more than $40,000 *before* the function in the HAVING clause is applied.

Example (Cont.)

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

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.

Summarization of SQL Queries & Execution Sequence

5. SELECT <attribute and function list>

1. FROM <table list>

2. [**WHERE** <condition>]

3. [**GROUP BY** <grouping attribute(s)>]

4. [**HAVING** <group condition>]

6. [**ORDER BY** <attribute list>];

DROP Command

- Used to drop **named schema elements**, such as tables, domains, or constraint
- Drop behavior options:
  - **CASCADE and RESTRICT**

Removes the COMPANY database schema and all its tables, domains, and other elements

- **Example:**
  - DROP SCHEMA COMPANY **CASCADE**;

DROP Command Example

- **DROP SCHEMA** COMPANY **RESTRICT**

schema is dropped only if it has *no elements* in it; otherwise, the DROP command will not be executed

not only deletes all the records in the table if successful, but also removes the *table definition* from the catalog

- **DROP TABLE** DEPENDENT **CASCADE**

Thanks!!