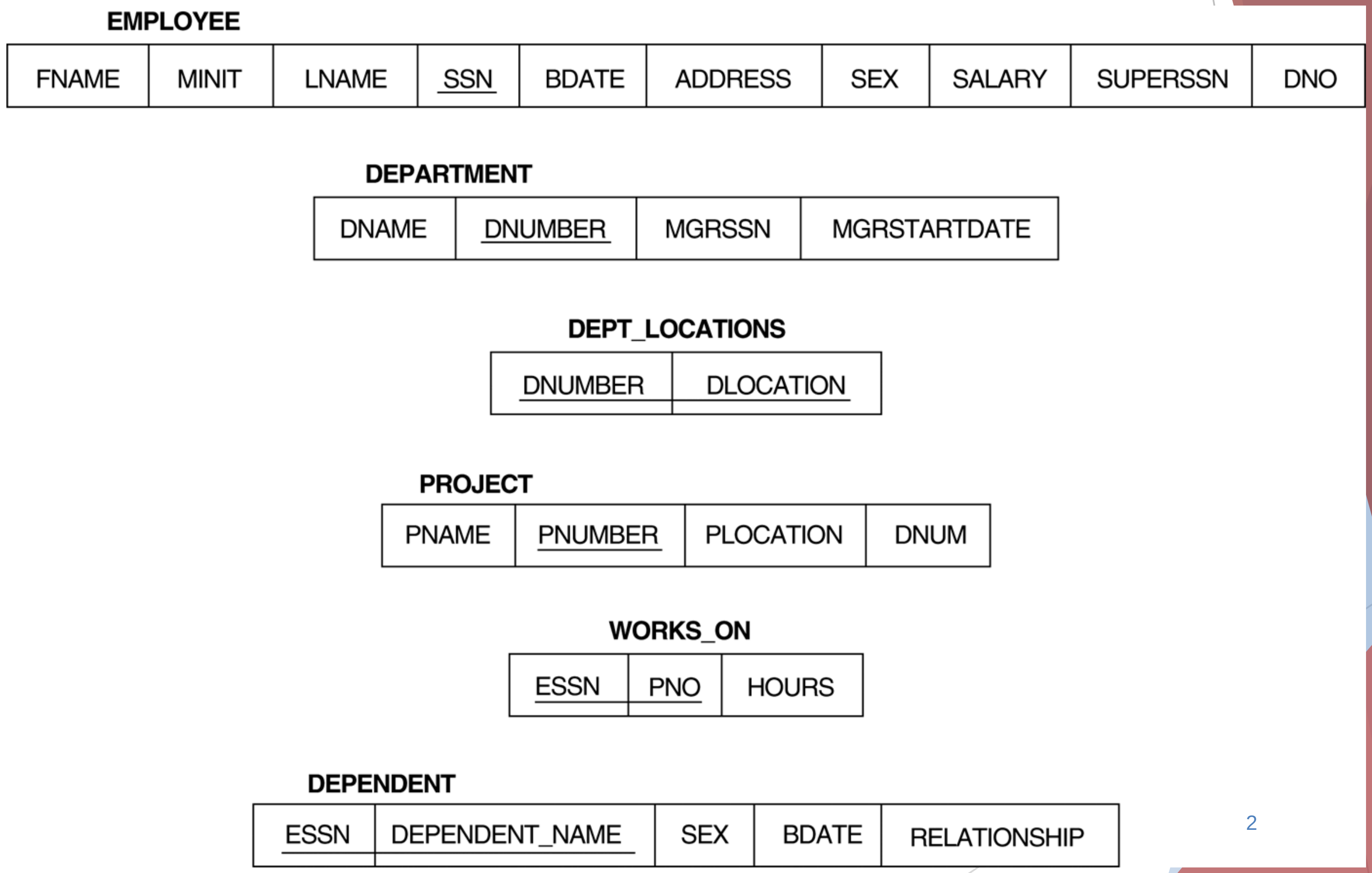
Lecture 8

SQL : Schema Definition, Constraints, and Queries and Views

Relational Database Schema



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

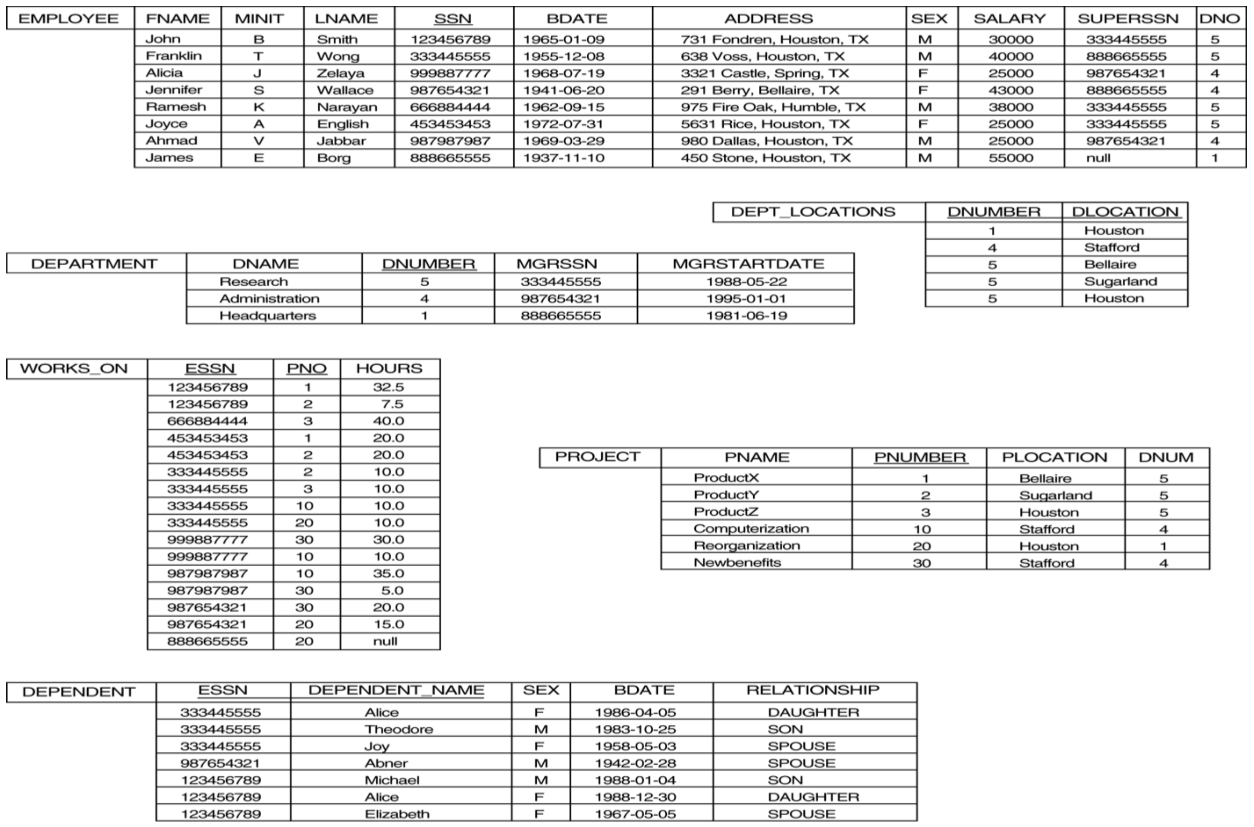
WORKS_ON

ESSN	PNO	HOURS

DEPENDENT

ESSN	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP

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

The general format of the SELECT statements

- 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>]
```

Projections (either project all attributes (*) or project only specefic attributes

Retrieval Queries in SQL

- <attribute list> is a list of attribute names whose values are to be retrieved by the query
- <table list> is a list of the relation names required to process the query
- <condition> is a conditional (Boolean) expression that identifies the tuples to be retrieved by the query
- We may use the comparison operators : =, <, >, ≤, ≥ , ≠
- We may use the Boolean operators AND, OR and NOT

Simple SQL Queries

Example of a simple query on one relation

- Query 0: Retrieve the birthdate and address of the employee whose name is 'John B. Smith'.

```
Q0:    SELECT    BDATE, ADDRESS
        FROM     EMPLOYEE
        WHERE    FNAME="John" AND  MINIT='B'
        AND      LNAME="Smith";
```

- The SELECT-clause specifies the projection attributes and the WHERE-clause specifies the selection condition

BDATE	ADDRESS
1965-1-9	171 …… street

- The result of the query may contain duplicate tuples

UNSPECIFIED WHERE-clause

- A *missing WHERE-clause* indicates no condition; hence, all tuples of the relations in the FROM-clause are selected
  - This is equivalent to the condition WHERE TRUE
- Query 1: Retrieve the SSN values for all employees.

```
Q1:     SELECT   SSN
         FROM    EMPLOYEE
```

SSN
12345678
12223334
…….

USE OF *

- To retrieve all the attribute values of the selected tuples, a * is used, which stands for *all the attributes*

Examples:

```
Q2:     SELECT    *
        FROM      EMPLOYEE
        WHERE      DNO=5
```

USE OF DISTINCT

- SQL does not treat a relation as a set; duplicate tuples can appear
- To eliminate duplicate tuples in a query result, the keyword **DISTINCT** is used
- For example, the result of Q3 may have duplicate SALARY values whereas Q3A does not have any duplicate values

```
Q3 :   SELECT   SALARY
       FROM     EMPLOYEE

Q3A:   SELECT   DISTINCT   SALARY
       FROM     EMPLOYEE
```

SALARY
1000
2000
1000
1500
…

SALARY
1000
2000
1500
…

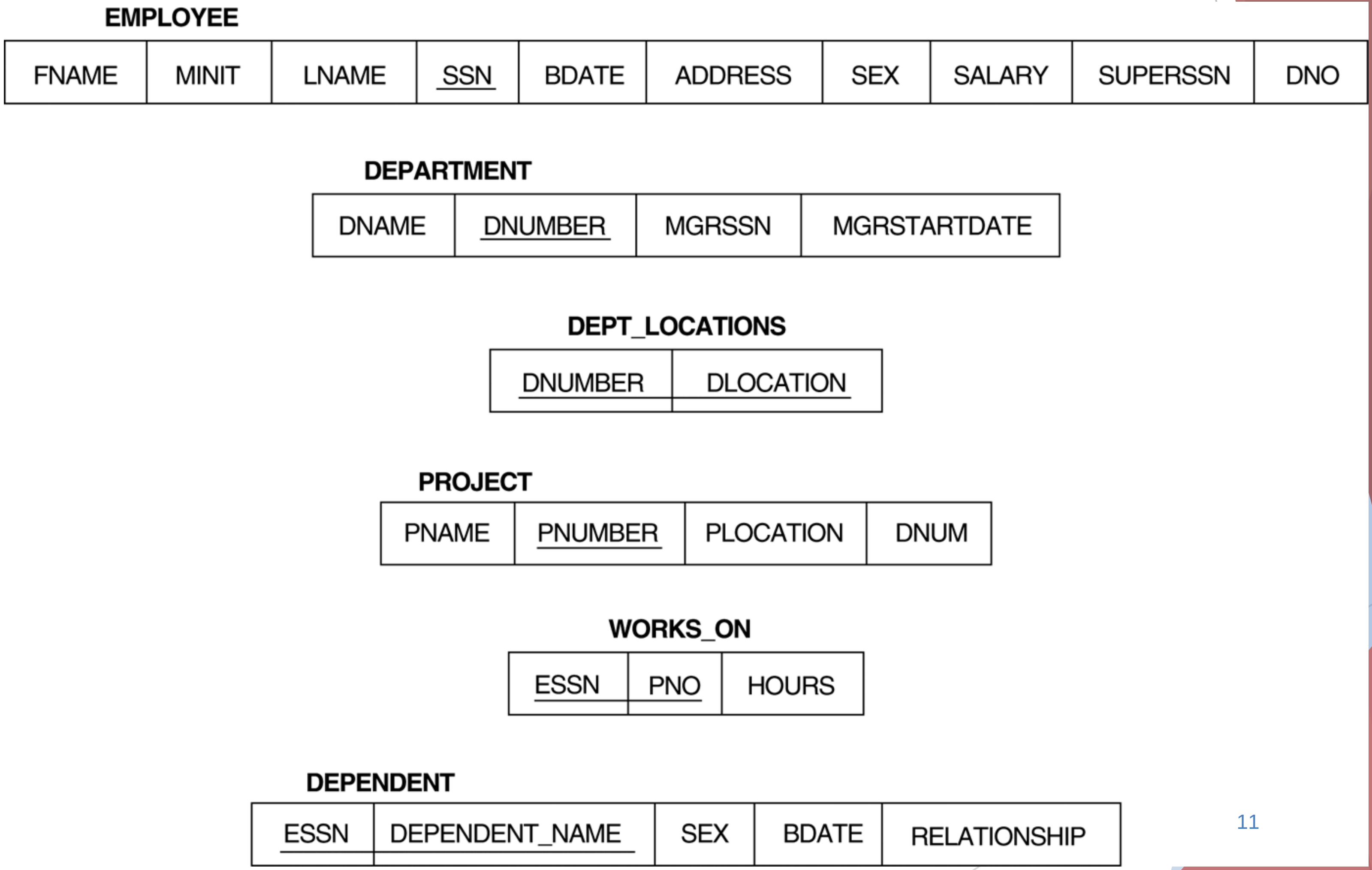
Simple SQL Queries

Query 4: Retrieve the name and address of all employees who work for the 'Research' department.

```
Q4:    SELECT    FNAME, LNAME, ADDRESS
       FROM      EMPLOYEE, DEPARTMENT
       WHERE     DNAME='Research' AND
                 DNUMBER=DNO
```

- (DNAME='Research') is a selection condition
- (DNUMBER=DNO) is a join condition

Relational Database Schema



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

Aliases, * and DISTINCT, Empty WHERE-clause

- In SQL, we can use the same name for two (or more) attributes as long as the attributes are in *different relations*
- A query that refers to two or more attributes with the same name must *qualify* the attribute name with the relation name by *prefixing* the relation name to the attribute name
- Example:

EMPLOYEE.LNAME, **DEPARTMENT.**DNAME

Simple SQL Queries

The above query may be re-written using

Relation name.attribute

```
Q4m:  SELECT   FNAME, LNAME, ADDRESS
      FROM     EMPLOYEE, DEPARTMENT
      WHERE    DNAME='Research' AND
      DEPARTMENT.DNUMBER= EMPLOYEE.DNO
```

Simple SQL Queries

Query 5: 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.

Q5:

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

- In Q5, there are two join conditions
- The join condition DNUM=DNUMBER relates a project to its controlling department
- The join condition MGRSSN=SSN relates the controlling department to the employee who manages that department

ALIASES

- Some queries need to refer to the same relation twice
  - In this case, *aliases* are given to the relation name
- Query 6: For each employee, retrieve the employee's name, and the name of his or her immediate supervisor.

Q6:

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

ALIASES

  - In Q6, the alternate relation names E and S are called *aliases* or *tuple variables* for the EMPLOYEE relation
  - We can think of E and S as two different *copies* of EMPLOYEE; E represents employees in role of *supervisees* and S represents employees in role of *supervisors*
- Can use the AS keyword to specify aliases

Q6m:

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

UNSPECIFIED WHERE-clause

- Example:

```
Q7:   SELECT   SSN, DNAME
      FROM     EMPLOYEE, DEPARTMENT
```

- If more than one relation is specified in the FROM-clause *and* there is no condition, then the *CARTESIAN PRODUCT* of tuples is selected
- It is extremely important not to overlook specifying any selection and join conditions in the WHERE-clause; otherwise, incorrect and very large relations may result

fname	mname	lname	SSN	…									
…													
….													

SET OPERATIONS

- SQL has directly incorporated some set operations
- There is a union operation (UNION), and in *some versions* of SQL there are set difference (MINUS) and intersection (INTERSECT) operations
- The resulting relations of these set operations are sets of tuples; *duplicate tuples are eliminated from the result*
- The set operations apply only to *union compatible relations*; the two relations must have the same attributes and the attributes must appear in the same order

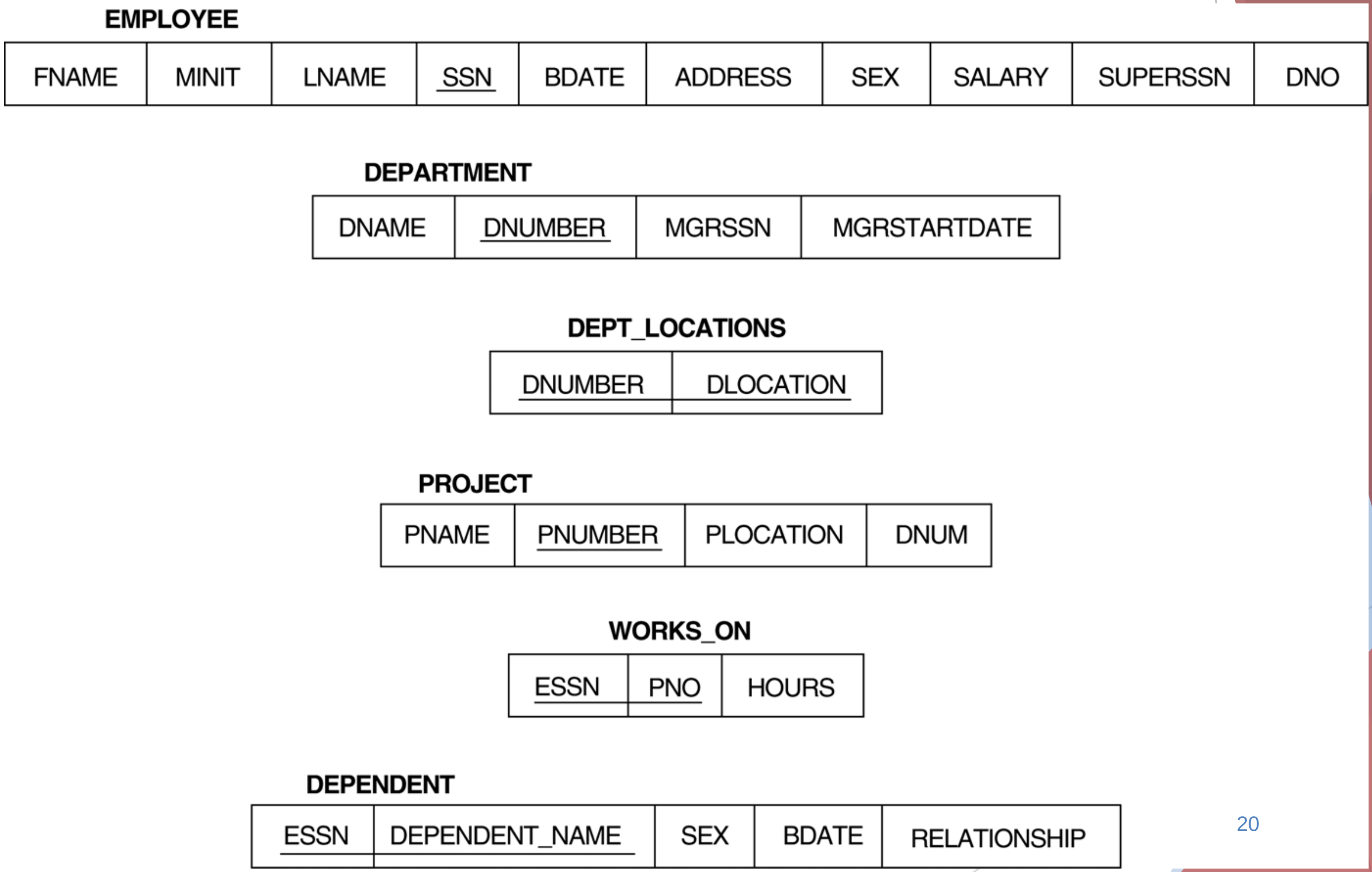
SET OPERATIONS

Query 8: 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.

```
Q8:     (SELECT    PNAME
         FROM      PROJECT, DEPARTMENT, EMPLOYEE
        WHERE      DNUM=DNUMBER AND
                   MGRSSN=SSN AND LNAME=”Smith�because)
        UNION
        (SELECT    PNAME
        FROM       PROJECT, WORKS_ON, EMPLOYEE
        WHERE      PNUMBER=PNO AND

                   ESSN=SSN AND LNAME=”Smith‟)
```

Relational Database Schema



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

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
- Query 4: Retrieve the name and address of all employees who work for the 'Research' department.

```
Q4m:  SELECT     FNAME, LNAME, ADDRESS
       FROM      EMPLOYEE
       WHERE     DNO IN
                ( SELECT    DNUMBER
                  FROM      DEPARTMENT
                  WHERE     DNAME="Research"
                 )
```

NESTING OF QUERIES

- 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

NESTING OF QUERIES

- 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 this example, the nested query is *not correlated* with the outer query
- 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*

CORRELATED NESTED QUERIES

- Query 9: Retrieve the name of each employee who has a dependent with the same first name as the employee.

```
Q9: SELECT      E.FNAME, E.LNAME
        FROM        EMPLOYEE AS E
        WHERE       E.SSN IN
                    (SELECT     ESSN
                      FROM       DEPENDENT
                      WHERE      ESSN=E.SSN AND Co-related here 'E.SSN'
                      E.FNAME=DEPENDENT_NAME)
```

CORRELATED NESTED QUERIES

- In Q9, the nested query has a different result in the outer query

Nested Queries are best when using aggregate functions. E.g : Select salary from EMP where SALARY > (select AVG(salary) from EMP

- 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. For example, Q9 may be written as in Q9A

Based on Gemini, The performance of one JOIN query is higher than nested query, but it depends on the DB engine, since some engines flatten nested queries to JOIN

```
Q9A:   SELECT     E.FNAME, E.LNAME
       FROM       EMPLOYEE E, DEPENDENT D
       WHERE      E.SSN=D.ESSN     AND
                  E.FNAME=D.DEPENDENT_NAME
```

Example

- Get employee numbers for employees who have the same salary as “John”

```
SELECT SSN
FROM    EMPLOYEE
WHERE  SALARY = ( SELECT SALARY
                  FROM EMPLOYEE
                  WHERE FNAME=“John”)
```

THE EXISTS FUNCTION

- EXISTS is used to check whether the result of a correlated nested query is empty (contains no tuples) or not
  - We can formulate Query 9 in an alternative form that uses EXISTS as Q9B

THE EXISTS FUNCTION

- Query 9: Retrieve the name of each employee who has a dependent with the same first name as the employee.

```
Q9B:  SELECT    FNAME, LNAME
      FROM      EMPLOYEE
      WHERE     EXISTS

               (SELECT  *
                FROM     DEPENDENT
                WHERE   SSN=ESSN
                AND
                FNAME=DEPENDENT_NAME)
```

THE EXISTS FUNCTION

- Query 10: Retrieve the names of employees who have no dependents.

```
Q10:    SELECT    FNAME, LNAME
        FROM      EMPLOYEE
        WHERE     NOT EXISTS

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

- In Q10, the correlated nested query retrieves all DEPENDENT tuples related to an EMPLOYEE tuple. If *none exist*, the EMPLOYEE tuple is selected

EXPLICIT SETS

- It is also possible to use an **explicit (enumerated) set of values** in the WHERE-clause rather than a nested query
- Query 11: Retrieve the social security numbers of all employees who work on project number 1, 2, or 3.

```
Q11:  SELECT    DISTINCT ESSN
      FROM  WORKS_ON
      WHERE     PNO IN  (1, 2, 3)
```

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*.
- Query 12: Retrieve the names of all employees who do not have supervisors.

```
Q12:   SELECT    FNAME, LNAME
       FROM      EMPLOYEE
       WHERE     SUPERSSN  IS  NULL
```

  - Note: If a join condition is specified, tuples with NULL values for the join attributes are not included in the result

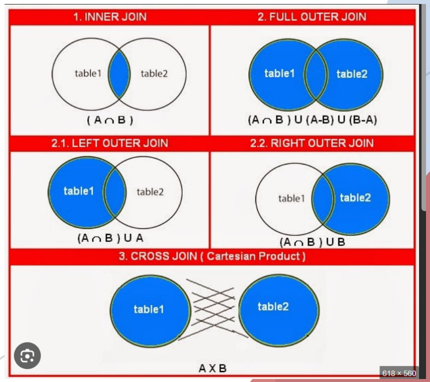
This is what happened in "Al-Naggar" queries where the query results were missing since joined objects in another table were null.
Solution: Use LEFT / RIGHT Joins to include all the table results you need even if null. This is specified in .NET orm with Nullable FK and objects.

Joined Relations Feature in SQL2

- Can specify a "joined relation" in the FROM-clause
  - Looks like any other relation but is the result of a join
  - Allows the user to specify different types of joins (regular "theta" JOIN, NATURAL JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN, CROSS JOIN, etc)

DEFINITIONS

- Theta JOIN: Produces all combinations of tuples from R and S that satisfies the join condition
- NATURAL JOIN (inner join) : As theta join but with equal conditions, and the joined attributes in S are not included
- LEFT OUTER JOIN: R left outer join S means keep all the tuples in R even if they are not matching the conditions.
- RIGHT OUTER JOIN: R right outer join S the same as above but keep all tuples of S



Joined Relations Feature in SQL2

- Examples:

Q13:

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

- can be written as:

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

E.FNAME	E.LNAME	S.FNAME	S.LNAME
Hoda	Mohamed	Aly	Ahmed
Aly	Ahmed		

Joined Relations Feature in SQL2

Examples:

```
Q4: SELECT   FNAME, LNAME, ADDRESS
    FROM     EMPLOYEE, DEPARTMENT
    WHERE    DNAME="Research" AND
             DNUMBER=DNO
```

- could be written as:

```
Q4: SELECT   FNAME, LNAME, ADDRESS
    FROM     (EMPLOYEE JOIN DEPARTMENT
              ON DNUMBER=DNO)
    WHERE     DNAME="Research"
```

- or as:

```
Q4: SELECT   FNAME, LNAME, ADDRESS
    FROM    (EMPLOYEE NATURAL JOIN   DEPARTMENT
             AS DEPT(DNAME, DNO, MSSN, MSDATE)
    WHERE    DNAME="Research"
```

Joined Relations Feature in SQL2

- Another Example: Q5 could be written as follows; this illustrates multiple joins in the joined tables

Q5:

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

AGGREGATE FUNCTIONS

- Include **COUNT, SUM, MAX, MIN, and AVG**
- Query 14: Find the maximum salary, the minimum salary, and the average salary among all employees.

```
Q14:  SELECT   MAX(SALARY),
               MIN(SALARY), AVG(SALARY)
               FROM  EMPLOYEE
```

- Some SQL implementations *may not allow more than one function* in the SELECT-clause

AGGREGATE FUNCTIONS

- Query 15: Find the maximum salary, the minimum salary, and the average salary among employees who work for the 'Research' department.

```
Q15:   SELECT    MAX(SALARY),
       MIN(SALARY), AVG(SALARY)
       FROM      EMPLOYEE, DEPARTMENT
       WHERE     DNO=DNUMBER AND
                 DNAME=”Research‟
```

AGGREGATE FUNCTIONS

- Queries 16 and 17: Retrieve the total number of employees in the company (Q16), and the number of employees in the 'Research' department (Q17).

```
Q16:  SELECT   COUNT (*)
      FROM     EMPLOYEE

Q17:  SELECT   COUNT (*)
      FROM     EMPLOYEE, DEPARTMENT
      WHERE    DNO=DNUMBER AND
               DNAME=”Research‟
```

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*

GROUPING

- Query 18: For each department, retrieve the department number, the number of employees in the department, and their average salary.

```
Q18:  SELECT    DNO, COUNT (*), AVG (SALARY)
      FROM      EMPLOYEE
      GROUPBY   DNO
```

Rule, Selecting Columns alongside Aggregate functions must be accompanied with GROUP BY for all columns, otherwise the database engine will not know how to display the selected column (attribute)

- In Q18, the 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

Understanging Grouping: When aggregate functions is used without grouping, it process all table and have 1 result, but when using GROUP BY, the processing for Aggregate function is across only each group of the specified attribute.
For e.g , the following table for query having 'GROUP BY DNO' : each set of equal DNO are grouped and processed together with the Aggregate function , displaying aggregate funcitons for each group independently

SSN	NAME	…	SALARY	DNO
123			1000	1
1234			2000	1
213			100	2
1324			200	2
4356			300	2
12567			500	3
7689			700	4

DNO	COUNT	AVERAGE
1	2	1500
2	3	200
….	….	…

GROUPING

- Query 19: For each project, retrieve the project number, project name, and the number of employees who work on that project.

```
Q19:  SELECT   PNUMBER, PNAME, COUNT (*)
      FROM     PROJECT, WORKS_ON
      WHERE    PNUMBER=PNO
      GROUPBY  PNUMBER, PNAME
```

  - In this case, the grouping and functions are applied after the joining of the two relations

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)

'Having' is a 'WHERE' clause applied on Aggregate function (the correct usage) after the grouping to filter the result with the specified condition

THE HAVING-CLAUSE

- Query 20: 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
GROUPBY   PNUMBER, PNAME
HAVING    COUNT (*) > 2
```

PNUMBER	PNAME
1	XX
1	XX
1	XX
2	YY

COUNT=3 > 2

COUNT=1 < 2

PNUMBER	PNAME	COUNT
1	XX	3

SUBSTRING COMPARISON

- The **LIKE** comparison operator is used to compare partial strings
- Two reserved characters are used: '%' (or '*' in some implementations) replaces an arbitrary number of characters, and '_' replaces a single arbitrary character

SUBSTRING COMPARISON

- Query 21: Retrieve all employees whose address is in Houston, Texas. Here, the value of the ADDRESS attribute must contain the substring 'Houston,TX‘ in it.

```
Q21:   SELECT    FNAME, LNAME
       FROM      EMPLOYEE
       WHERE     ADDRESS LIKE ”%Houston,TX%‟
```

SUBSTRING COMPARISON

- Query 22: 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), so the BDATE value is '_ _5_ _ _ _ _ _ _', with each underscore as a place holder for a single arbitrary character.

```
Q22:  SELECT   FNAME, LNAME
      FROM     EMPLOYEE
      WHERE    BDATE LIKE   "_ _5_ _ _ _ _ _ _"
```

- The LIKE operator allows us to get around the fact that each value is considered atomic and indivisible

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
- Query 23: Show the effect of giving all employees who work on the 'ProductX' project a 10% raise.

Q23:

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

ORDER BY

- The **ORDER BY** clause is used to sort the tuples in a query result based on the values of some attribute(s)
- Query 24: Retrieve a list of employees and the projects each works in, ordered by the employee's department, and within each department ordered alphabetically by employee last name.

```
Q24: SELECT       DNAME, LNAME, FNAME, PNAME
     FROM         DEPARTMENT, EMPLOYEE,
                        WORKS_ON, PROJECT
     WHERE        DNUMBER=DNO AND SSN=ESSN
                   AND PNO=PNUMBER
      ORDER BY   DNAME, LNAME
```

ORDER BY

- The default order is in ascending order of values
- We can specify the keyword **DESC** if we want a descending order; the keyword **ASC** can be used to explicitly specify ascending order, even though it is the default