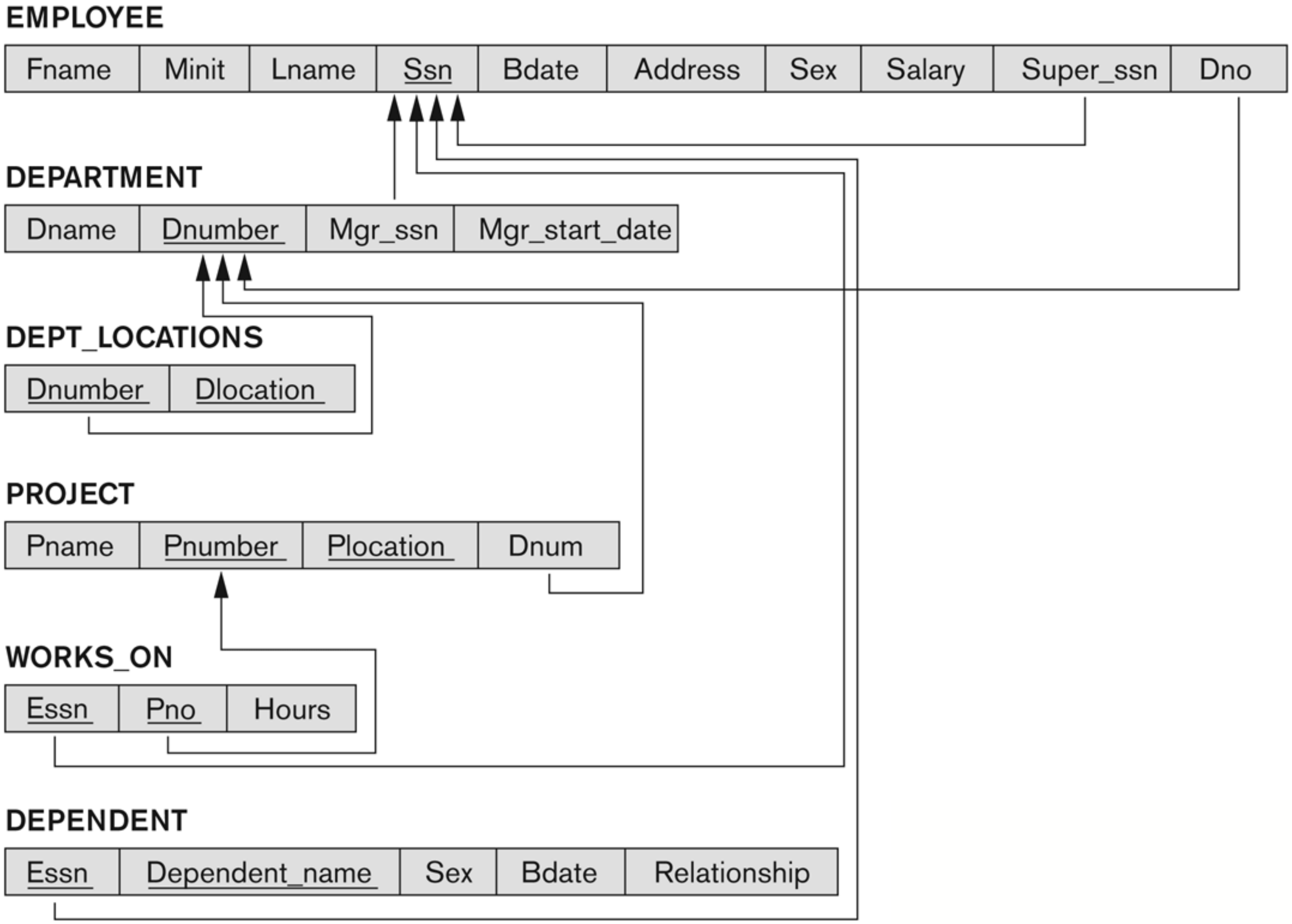
SQL - SELECT

COMPANY relational 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

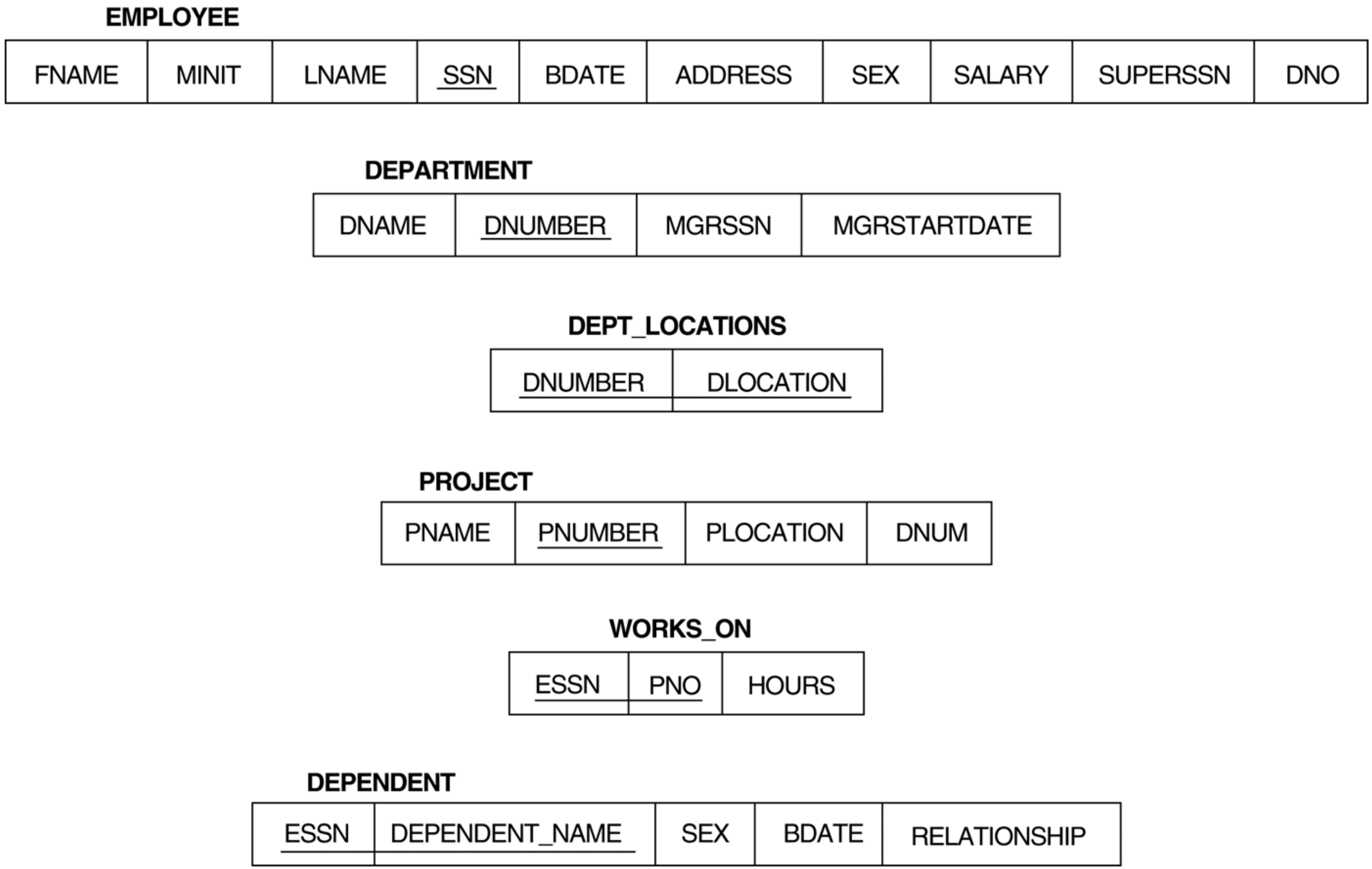
WORKS_ON

Essn	Pno	Hours

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship

Relational Database Schema



EMPLOYEE

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO

DEPARTMENT

DNAME	DNUMBER	MGRSSN	MGRSTARTDATE

DEPT_LOCATIONS

DNUMBER	DLOCATION

PROJECT

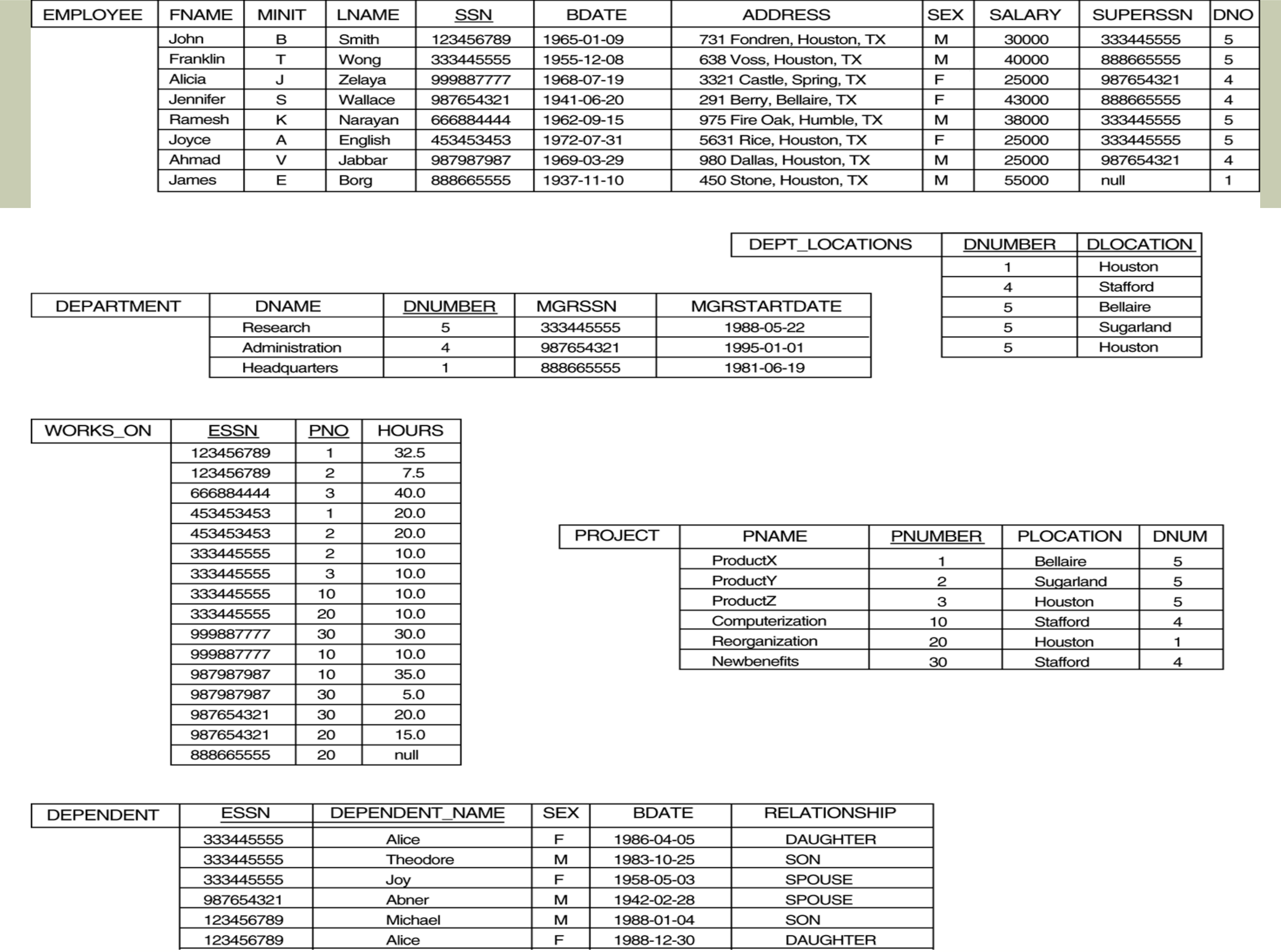
PNAME	PNUMBER	PLOCATION	DNUM

WORKS_ON

ESSN	PNO	HOURS

DEPENDENT

ESSN	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP



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

Retrieval Queries in SQL

- SQL has one basic statement for retrieving information from a database; the **SELECT** statement
  - This is *not the same as* the SELECT operation of the relational algebra
- Important distinction between SQL and the formal relational model:
  - SQL allows a table (relation) to have two or more tuples that are identical in all their attribute values
  - Hence, an SQL relation (table) is a **multi-set** (sometimes called a **bag**) of tuples; it is *not* a set of tuples
- SQL relations can be constrained to be sets by specifying PRIMARY KEY or UNIQUE attributes, or by using the DISTINCT option in a query

Retrieval Queries in SQL (contd.)

- A **bag** or **multi-set** is like a set, but an element may appear more than once.
  - Example: {A, B, C, A} is a bag. {A, B, C} is also a bag that also is a set.
  - Bags also resemble lists, but the order is irrelevant in a bag.
- Example:
  - {A, B, A} = {B, A, A} as bags
  - However, [A, B, A] is not equal to [B, A, A] as lists

Retrieval Queries in SQL (contd.)

- Basic form of the SQL SELECT statement is called a *mapping* or a SELECT-FROM-WHERE *block*

```
SELECT     <attribute list>
   FROM    <table list>
   WHERE   <condition>
```

  - <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

Simple SQL Queries

- Basic SQL queries correspond to using the following operations of the relational algebra:
  - SELECT
  - PROJECT
  - JOIN
- All subsequent examples use the COMPANY database

Simple SQL Queries (contd.)

- 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’
```

  - Similar to a SELECT-PROJECT pair of relational algebra operations:
    - The SELECT-clause specifies the projection attributes and the WHERE-clause specifies the selection condition
  - However, the result of the query may contain duplicate tuples

Simple SQL Queries (contd.)

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

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

  - Similar to a SELECT-PROJECT-JOIN sequence of relational algebra operations
  - (DNAME='Research') is a selection condition (corresponds to a SELECT operation in relational algebra)
  - (DNUMBER=DNO) is a join condition (corresponds to a JOIN operation in relational algebra)

Simple SQL Queries (contd.)

- Query 2: For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name, address, and birthdate.

```
Q2: SELECT    PNUMBER, DNUM, LNAME, BDATE, ADDRESS
    FROM      PROJECT, DEPARTMENT, EMPLOYEE
    WHERE     DNUM=DNUMBER AND MGRSSN=SSN
              AND PLOCATION='Stafford'
```

  - In Q2, 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, * 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

ALIASES

- Some queries need to refer to the same relation twice
  - In this case, *aliases* are given to the relation name
- Query 8: For each employee, retrieve the employee's name, and the name of his or her immediate supervisor.

```
Q8:  SELECT     E.FNAME, E.LNAME, S.FNAME, S.LNAME
 FROM        EMPLOYEE E S
 WHERE      E.SUPERSSN=S.SSN
```

  - In Q8, 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*

ALIASES (contd.)

- Aliasing can also be used in any SQL query for convenience
- Can also use the AS keyword to specify aliases

```
Q8: SELECT E.FNAME, E.LNAME,
S.FNAME, S.LNAME
    FROM   EMPLOYEE AS E,
EMPLOYEE AS S
    WHERE E.SUPERSSN=S.SSN
```

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 9: Retrieve the SSN values for all employees.

  -
    ```
    Q9:  SELECT   SSN
         FROM     EMPLOYEE
    ```
- If more than one relation is specified in the FROM-clause *and* there is no join condition, then the *CARTESIAN PRODUCT* of tuples is selected

UNSPECIFIED WHERE-clause (contd.)

- Example:

```
Q10:  SELECT  SSN, DNAME
      FROM    EMPLOYEE, DEPARTMENT
```

  - 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

USE OF *

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

```
Q1C:   SELECT   *
    FROM     EMPLOYEE
    WHERE    DNO=5

Q1D:   SELECT   *
    FROM     EMPLOYEE, DEPARTMENT
    WHERE    DNAME='Research' AND
DNO=DNUMBER
```

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 Q11 may have duplicate SALARY values whereas Q11A does not have any duplicate values

```
Q11:    SELECT   SALARY
FROM    EMPLOYEE
Q11A:   SELECT   DISTINCT SALARY
FROM    EMPLOYEE
```

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 (contd.)

- Query 4: 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.

```
Q4:     (SELECT    PNAME
         FROM      PROJECT, DEPARTMENT,
   EMPLOYEE
         WHERE     DNUM=DNUMBER AND
  MGRSSN=SSN AND LNAME='Smith')
         UNION
         (SELECT   PNAME
         FROM      PROJECT, WORKS_ON, EMPLOYEE
         WHERE     PNUMBER=PNO AND
                ESSN=SSN AND NAME='Smith')
```

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 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' )
```

NESTING OF QUERIES (contd.)

- 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 this 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) 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)
```

CORRELATED NESTED QUERIES (contd.)

- In Q12, the nested query has a different result in the outer query
- 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, Q12 may be written as in Q12A

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

CORRELATED NESTED QUERIES (contd.)

- The original SQL as specified for SYSTEM R also had a **CONTAINS** comparison operator, which is used in conjunction with nested correlated queries
  - This operator was *dropped from the language*, possibly because of the difficulty in implementing it efficiently
  - Most implementations of SQL do not have this operator
  - The CONTAINS operator compares *two sets of values*, and returns TRUE if one set contains all values in the other set
    - Reminiscent of the division operation of algebra

CORRELATED NESTED QUERIES (contd.)

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

```
Q3:  SELECT   FNAME, LNAME
     FROM     EMPLOYEE
     WHERE (      (SELECT  PNO
              FROM     WORKS_ON
              WHERE    SSN=ESSN)
                  CONTAINS
              (SELECT  PNUMBER
              FROM     PROJECT
              WHERE    DNUM=5) )
```

CORRELATED NESTED QUERIES (contd.)

- In Q3, the second nested query, which is *not correlated* with the outer query, retrieves the project numbers of all projects controlled by department 5
- The first nested query, which is correlated, retrieves the project numbers on which the employee works, which is *different for each employee tuple* because of the correlation

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 12 in an alternative form that uses EXISTS as Q12B

THE EXISTS FUNCTION (contd.)

- 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)
```

THE EXISTS FUNCTION (contd.)

- 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)
```

- In Q6, the correlated nested query retrieves all DEPENDENT tuples related to an EMPLOYEE tuple. If *none exist*, the EMPLOYEE tuple is selected
  - EXISTS is necessary for the expressive power of SQL

EXPLICIT SETS

- It is also possible to use an **explicit (enumerated) set of values** in the WHERE-clause rather than a nested query
- Query 13: Retrieve the social security numbers of all employees who work on project number 1, 2, or 3.

```
Q13:  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 14: Retrieve the names of all employees who do not have supervisors.

```
Q14:   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

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)

Joined Relations Feature in SQL2 (contd.)

- Examples:

```
Q8:     SELECT   E.FNAME, E.LNAME, S.FNAME, S.LNAME
        FROM       EMPLOYEE E S
        WHERE    E.SUPERSSN=S.SSN
```

- can be written as:

```
Q8:     SELECT   E.FNAME, E.LNAME, S.FNAME, S.LNAME
        FROM       (EMPLOYEE E LEFT OUTER JOIN
        EMPLOYEES ON  E.SUPERSSN=S.SSN)
```

Joined Relations Feature in SQL2 (contd.)

- Examples:

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

- could be written as:

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

- or as:

```
Q1:	SELECT	FNAME, LNAME, ADDRESS
	FROM	(EMPLOYEE NATURAL JOIN DEPARTMENT
		AS DEPT(DNAME, DNO, MSSN, MSDATE)
	WHERE	DNAME='Research’
```

Joined Relations Feature in SQL2 (contd.)

- Another Example: Q2 could be written as follows; this illustrates multiple joins in the joined tables

```
Q2:  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 15: Find the maximum salary, the minimum salary, and the average salary among all employees.

```
 Q15:  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 (contd.)

- Query 16: Find the maximum salary, the minimum salary, and the average salary among employees who work for the 'Research' department.

```
Q16:  SELECT     MAX(SALARY),
   MIN(SALARY), AVG(SALARY)
      FROM    EMPLOYEE, DEPARTMENT
      WHERE DNO=DNUMBER AND
   DNAME='Research'
```

AGGREGATE FUNCTIONS (contd.)

- Queries 17 and 18: Retrieve the total number of employees in the company (Q17), and the number of employees in the 'Research' department (Q18).

```
Q17:   SELECT   COUNT (*)
       FROM     EMPLOYEE


Q18:   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 (contd.)

- Query 20: For each department, retrieve the department number, the number of employees in the department, and their average salary.

```
Q20:   SELECT     DNO, COUNT (*), AVG (SALARY)
       FROM       EMPLOYEE
       GROUP BY   DNO
```

  - In Q20, 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

GROUPING (contd.)

- Query 21: For each project, retrieve the project number, project name, and the number of employees who work on that project.

```
Q21:   SELECT     PNUMBER, PNAME, COUNT (*)
       FROM       PROJECT, WORKS_ON
       WHERE      PNUMBER=PNO
       GROUP BY     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)

THE HAVING-CLAUSE (contd.)

- 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
```

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 (contd.)

- 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%'
```

SUBSTRING COMPARISON (contd.)

- 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), so the BDATE value is '_______5_', with each underscore as a place holder for a single arbitrary character.

```
Q26:   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
  - Hence, in SQL, character string attribute values are not atomic

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

```
Q27:   SELECT   FNAME, LNAME, 1.1*SALARY
       FROM     EMPLOYEE, WORKS_ON,
  PROJECT
       WHERE    SSN=ESSN AND PNO=PNUMBER
       AND PNAME='ProductX’
```

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 28: 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.

```
Q28:  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 (contd.)

- 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

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>]
```

Summary of SQL Queries (contd.)

- 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 first applying the WHERE-clause, then GROUP BY and HAVING, and finally the SELECT-clause

Recap 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>]
  ```

- There are three SQL commands to modify the database: **INSERT**, **DELETE**, and **UPDATE**