SQL-99: Schema Definition, Basic Constraints, and Queries

Content

- Data Definition Language
  - Create, drop, alter
- Features Added in SQL2 and SQL-99
- Basic Structure and retrieval queries in SQL
- Set Operations
- Aggregate Functions
- Nested Sub-queries
- Derived Relations
- Views
- Modification of the Database (Data Manipulation Language)
- Joined Relations
- Embedded SQL, ODBC and JDBC

A Relational Database Language

- Most DBMSs provide the user a high-level declarative language interface

➔ the user has just to specifies "**what**" results he would like, leaving the "**how**" to the dbms.

- "Declarative" vs."Procedural"
  - SQL1/86 the first standard version
  - SQL2/92 the revised version
  - SQL3/99 introduces object oriented tools

Data Definition Language (DDL)

Allows the specification of not only a set of relations but also information about each relation, including:

- The schema for each relation.
- The domain of values associated with each attribute.
- Integrity constraints
- The set of indices to be maintained for each relations.
- Security and authorization information for each relation.
- The physical storage structure of each relation on disk.

Data Definition

- Used to CREATE, DROP, and ALTER the descriptions of the tables (relations) of a database

Create Table Construct

- An SQL relation is defined using the **create table** command:

  create table r (A_1 D_1, A_2 D_2, ..., A_n D_n,
  (integrity-constraint$_1$),
  ...,
  (integrity-constraint$_k$))

  - r is the name of the relation
  - each A_i is an attribute name in the schema of relation r
  - D_i is the data type of values in the domain of attribute A_i

Create Table

- Specifies a new base relation by giving it a name, and specifying each of its attributes and their data types (integer, float, decimal(i,j), char(n), varchar(n))
- A constraint NOT NULL may be specified on an attribute

```
CREATE TABLE   DEPARTMENT
      (        DNAME          VARCHAR(10)          NOT NULL,
               DNUMBER     INTEGER       NOT NULL,
               MGRSSN         CHAR(9),
               MGRSTARTDATE    CHAR(9)  );
```

Create Table

- In SQL2, one can use the CREATE TABLE command for specifying the primary key attributes, secondary keys, and referential integrity constraints (foreign keys).
- Key attributes can be specified via the PRIMARY KEY and UNIQUE phrases

```
CREATE TABLE   DEPT
(    DNAME      VARCHAR(10)      NOT NULL,
     DNUMBER  INTEGER    NOT NULL,
     MGRSSN     CHAR(9),
     MGRSTARTDATE CHAR(9),
     PRIMARY KEY (DNUMBER),
     UNIQUE (DNAME),
     FOREIGN KEY (MGRSSN) REFERENCES EMP
);
```

Create Table

- create table dep(. . .) as[query]
- create table employee(fname varchar(l5) not null default 'xxx', ssn char(9) not null, Dno int not null primary key ssn), foreign key(Dno) references department(D#);
- create table Departmentl(..., constraint pk_dep primary key(D#), constraint dept sk unique(Dname), . . .);
- create schema company authorization Adel;
- create table company.employee ....

DROP TABLE

- Used to remove a relation (base table) *and its definition*
- The relation can no longer be used in queries, updates, or any other commands since its description no longer exists
- Example:

DROP TABLE DEPENDENT;

ALTER TABLE

- Used to add an attribute to one of the base relations
- The new attribute will have NULLs in all the tuples of the relation right after the command is executed; hence, the NOT NULL constraint is not allowed for such an attribute
- Example:

ALTER TABLE EMPLOYEE ADD JOB VARCHAR(12);

- The database users must still enter a value for the new attribute JOB for each EMPLOYEE tuple. This can be done using the UPDATE command.

ALTERING

- **alter table:**
  - alter table company.employee add job varchar(12);
  - alter table company.employee drop address cascade;
- **altering attribute:**
  - alter table company.department alter location drop default.
  - alter table company.department alter location set default 'unknown'.
- **altering constraint:**
  - alter table company.department drop constraint dept_sk cascade.
  - alter table company.department add constraint DN unique(Dname).

Features Added in SQL2 and SQL-99

- **CREATE SCHEMA**
- **REFERENTIAL INTEGRITY OPTIONS**

CREATE SCHEMA

- Specifies a new database schema by giving it a name

Referential Integrity Options

- We can specify RESTRICT, CASCADE, SET NULL or SET DEFAULT on referential integrity constraints (foreign keys)

```
CREATE TABLE   DEPT
(     DNAME          VARCHAR(10)          NOT NULL,
      DNUMBER    INTEGER       NOT NULL,
      MGRSSN         CHAR(9),
      MGRSTARTDATE     CHAR(9),
      PRIMARY KEY (DNUMBER),
      UNIQUE (DNAME),
      FOREIGN KEY (MGRSSN) REFERENCES EMP
ON DELETE SET DEFAULT ON UPDATE CASCADE  );
```

Referential Integrity Options

```
CREATE TABLE   EMP
     (          ENAME          VARCHAR(30)  NOT NULL,
                ESSN    CHAR(9),
                BDATE DATE,
                DNO     INTEGER  DEFAULT 1,
                SUPERSSN       CHAR(9),
                PRIMARY KEY (ESSN),
                FOREIGN KEY (DNO) REFERENCES DEPT
           ON DELETE SET DEFAULT ON UPDATE CASCADE,
                FOREIGN KEY (SUPERSSN) REFERENCES EMP
           ON DELETE SET NULL ON UPDATE CASCADE  );
```

Additional Data Types in SQL2 and SQL-99

Has DATE, TIME, and TIMESTAMP data types

- **DATE:**
  - Made up of year-month-day in the format yyyy-mm-dd
- **TIME:**
  - Made up of hour:minute:second in the format hh:mm:ss
- **TIME(i):**
  - Made up of hour:minute:second plus i additional digits specifying fractions of a second
  - format is hh:mm:ss:ii...i
- **TIMESTAMP:**
  - Has both DATE and TIME components

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

Retrieval Queries in SQL (cont.)

- 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 (cont.)

- 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

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

Simple SQL Queries

- Basic SQL queries correspond to using the **SELECT**, **PROJECT**, and **JOIN** operations of the relational algebra
- All subsequent examples use the COMPANY database
- Example of a simple query on one relation
- Query 0: Retrieve the birthdate and address of the employee whose name is 'Badr A Arrouchi'.

```
Q0: SELECT    BDATE, ADDRESS
    FROM      EMPLOYEE
    WHERE     FNAME='Badr'
    AND       MINIT='A'
    AND       LNAME='Arrouchi'
```

```
select   BDate, Address
from     Employee
where    FName = "Badr"
And      Minit = "A"
And      LName = "Arrouchi"
```

Simple SQL Queries

- 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 (cont.)

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

```
Q1: SELECT   FNAME, LNAME, ADDRESS
    FROM     EMPLOYEE
     INNER JOIN DEPARTMENT on DNAME='Research'
    WHERE    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 (cont.)

- Query 2: For every project located in 'Riyadh', 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
    INNER JOIN DEPARTMENT on DNUM=DNUMBER
    INNER JOIN EMPLOYEE on MGRSSN=SSN
    WHERE     LOCATION='Riyadh'
```

  - 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
- **INNER JOIN EMPLOYEE S on 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 (cont.)

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

Q8: SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME
FROM EMPLOYEE AS E

- **INNER JOIN EMPLOYEE AS S on 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

- 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
INNER JOIN DEPARTMENT
on          DNO=DNUMBER
WHERE  DNAME='Research'
```

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 (Cont.)

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

```
Q4:  (SELECT  PNAME
      FROM    PROJECT
      INNER JOIN DEPARTMENT on DNUM=DNUMBER
      INNER JOIN EMPLOYEE on MGRSSN=SSN
      WHERE   LNAME='Jaber')
      UNION
      (SELECT  PNAME
      FROM    PROJECT
      INNER JOIN WORKS_ON  on ESSN=SSN
      INNER JOIN EMPLOYEE on PNUMBER=PNO
      WHERE   LNAME='Jaber')
```

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
                  FROMDEPARTMENT
                 WHERE DNAME='Research' )
```

Nesting Of Queries (cont.)

- 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

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

- 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 (cont.)

- In Q12, the nested query has a different result for each tuple 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,
       INNER JOIN DEPENDENT D  on E.SSN=D.ESSN
       WHERE      E.FNAME=D.DEPENDENT_NAME
```

Correlated Nested Queries (cont.)

- Most implementations of SQL do not have the operator contains
- The CONTAINS operator compares two sets of values , and returns TRUE if one set contains all values in the other set

*Division = $X \div Y = Z$
$schema(X) = schema(Y) \cup schema(Z)$

- **Query 3:** Retrieve the name of each employee who works on all (Division) 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 (cont.)

- 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 below

The Exists Function (cont.)

- 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 (cont.)

- 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

- Examples:

```
Q8: SELECT    E.FNAME, E.LNAME,
                S.FNAME, S.LNAME
    FROM      EMPLOYEE E
    INNER JOIN EMPLOYEE S
    on 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)
```

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

Joined Relations Feature in SQL2

- 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

- 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='Riyadh'
```

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.

count(salary): يرجع التكرار
count(*): يرجع بدون تكرار
max الباقي من وغيره لاتفعل (x)

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 (cont.)

- 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
     INNER JOIN  DEPARTMENT on DNO=DNUMBER
     WHERE     DNAME='Research'
```

Aggregate Functions (cont.)

- 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
INNER JOIN  DEPARTMENT
on DNO=DNUMBER
WHERE     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 (cont.)

- 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

Grouping (cont.)

- 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 (cont.)

- 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
        INNER JOIN  WORKS_ON
        on 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 (cont.)

- 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
INNER JOIN  WORKS_ON
on       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 (cont.)

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

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

Substring Comparison (cont.)

- Query 26: Retrieve all employees who were born during the 1980s. Here, '8' must be the 8th character of the string (according to our format for date), so the BDATE value is '_____8_', with each underscore as a place holder for a single arbitrary character.

Q26:
```
SELECT   FNAME, LNAME
FROM     EMPLOYEE
WHERE    BDATE LIKE '_____8_'
```

- 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
- **INNER JOIN WORKS_ON on SSN=ESSN**
- **INNER JOIN PROJECT on PNO=PNUMBER**

WHERE 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
        INNER JOIN  EMPLOYEE on DNUMBER=DNO
        INNER JOIN WORKS_ON on SSN=ESSN
        INNER JOIN PROJECT on PNO=PNUMBER
        ORDER BY  DNAME, LNAME
```

Order By (cont.)

- 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 (cont.)

- 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

Specifying Updates in SQL

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

INSERT

- In its simplest form, it is used to add one or more tuples to a relation
- Attribute values should be listed in the same order as the attributes were specified in the CREATE TABLE command

INSERT (cont.)

- Example:

U1: INSERT INTO EMPLOYEE
VALUES (‘Fahd','K','AlOtaibi', '653298653', '30-DEC-82',
‘159 street Dammam', 'M', 37000,'987654321', 4)

- An alternate form of INSERT specifies explicitly the attribute names that correspond to the values in the new tuple
- Attributes with NULL values can be left out
- Example: Insert a tuple for a new EMPLOYEE for whom we only know the FNAME, LNAME, and SSN attributes.

U1A: INSERT INTO EMPLOYEE (FNAME, LNAME, SSN)
VALUES ('Fahd', 'AlOtaibi', '653298653')

INSERT (cont.)

- Important Note: Only the constraints specified in the DDL commands are automatically enforced by the DBMS when updates are applied to the database
- Another variation of INSERT allows insertion of *multiple tuples* resulting from a query into a relation

INSERT (cont.)

- Example: Suppose we want to create a temporary table that has the name, number of employees, and total salaries for each department. A table DEPTS_INFO is created by U3A, and is loaded with the summary information retrieved from the database by the query in U3B.

```
U3A:CREATE TABLE  DEPTS_INFO
                  (DEPT_NAME   VARCHAR(10),
                   NO_OF_EMPS INTEGER,
                   TOTAL_SAL    INTEGER);
```

```
U3B:        INSERT INTO  DEPTS_INFO (DEPT_NAME,
                         NO_OF_EMPS, TOTAL_SAL)
            SELECT       DNAME, COUNT (*), SUM (SALARY)
            FROM         DEPARTMENT
            INNER JOIN  EMPLOYEE
            on          DNUMBER=DNO
            GROUP BY     DNAME ;
```

INSERT (cont.)

- Note: The DEPTS_INFO table may not be up-to-date if we change the tuples in either the DEPARTMENT or the EMPLOYEE relations after issuing U3B. We have to create a view to keep such a table up to date.

DELETE

- Removes tuples from a relation
- Includes a WHERE-clause to select the tuples to be deleted
- Tuples are deleted from only one table at a time (unless CASCADE is specified on a referential integrity constraint)
- A missing WHERE-clause specifies that all tuples in the relation are to be deleted; the table then becomes an empty table
- The number of tuples deleted depends on the number of tuples in the relation that satisfy the WHERE-clause
- Referential integrity should be enforced

DELETE (cont.)

- Examples:

```
U4A:    DELETE FROM     EMPLOYEE
        WHERE           LNAME='Badr'

U4B:    DELETE FROM     EMPLOYEE
        WHERE           SSN='123456789'

U4C:    DELETE FROM     EMPLOYEE
        WHERE           DNO  IN
                (SELECT     DNUMBER
                FROM DEPARTMENT
                WHERE       DNAME='Research')

U4D:    DELETE FROM     EMPLOYEE
```

UPDATE

- Used to modify attribute values of one or more selected tuples
- A WHERE-clause selects the tuples to be modified
- An additional SET-clause specifies the attributes to be modified and their new values
- Each command modifies tuples *in the same relation*
- Referential integrity should be enforced

UPDATE (cont.)

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

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

UPDATE (cont.)

- Example: 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')
```

- In this request, the modified SALARY value depends on the original SALARY value in each tuple
- The reference to the SALARY attribute on the right of = refers to the old SALARY value before modification
- The reference to the SALARY attribute on the left of = refers to the new SALARY value after modification