Module – III Introduction to SQL

- SQL data definition and data types
- Specifying constraints in SQL
- Basic retrieval queries in SQL
- Insert, update and delete statements in SQL
- Aggregate functions in SQL
- Group by and having clauses
- Database Programming: Issues and Techniques
- Embedded SQL

Overview

IBM SEQUEL (Structured English QUEry Language) language developed as part of System R project at the IBM San Jose Research Laboratory. Renamed as SQL (Structured Query Language)

- ANSI and ISO standard SQL:
- SQL-86 or SQL1
- SQL-89
- SQL-92 or SQL2
- SQL: 1999 (language name became Y2K compliant) or SQL3
- SQL: 2003

SQL is a comprehensive database language, it has statements for the data definition, updateand query. It has facilities for defining views on the database, for specifying security and authorization, for defining integrity constraints and for specifying transaction controls. Commercial systems offer most, if not all, SQL2 features, plus varying feature sets from later standards and special proprietary features.

Data Definition, Constraints, and Schema changes in SQL2

- ‘table’ for relation
- ‘row’ for tuple
- ‘column’ for attribute

Schema and Catalog concepts in SQL2

The concepts of relational database schema were incorporated into SQL2 in order to group together tables and other constructs that belong to the same database application. A schema is identified by a schema name and includes an authorization identifier. Authorization identifier indicates the user or account who owns the schema It also includes the descriptors for each element in the schema.

Schema and Catalog concepts in SQL2

Schema elements include tables, constraints, views, domains and other constructs (like authorization grants that describe the schema. Elements of schema can be defined at the time of creation of schema or later.

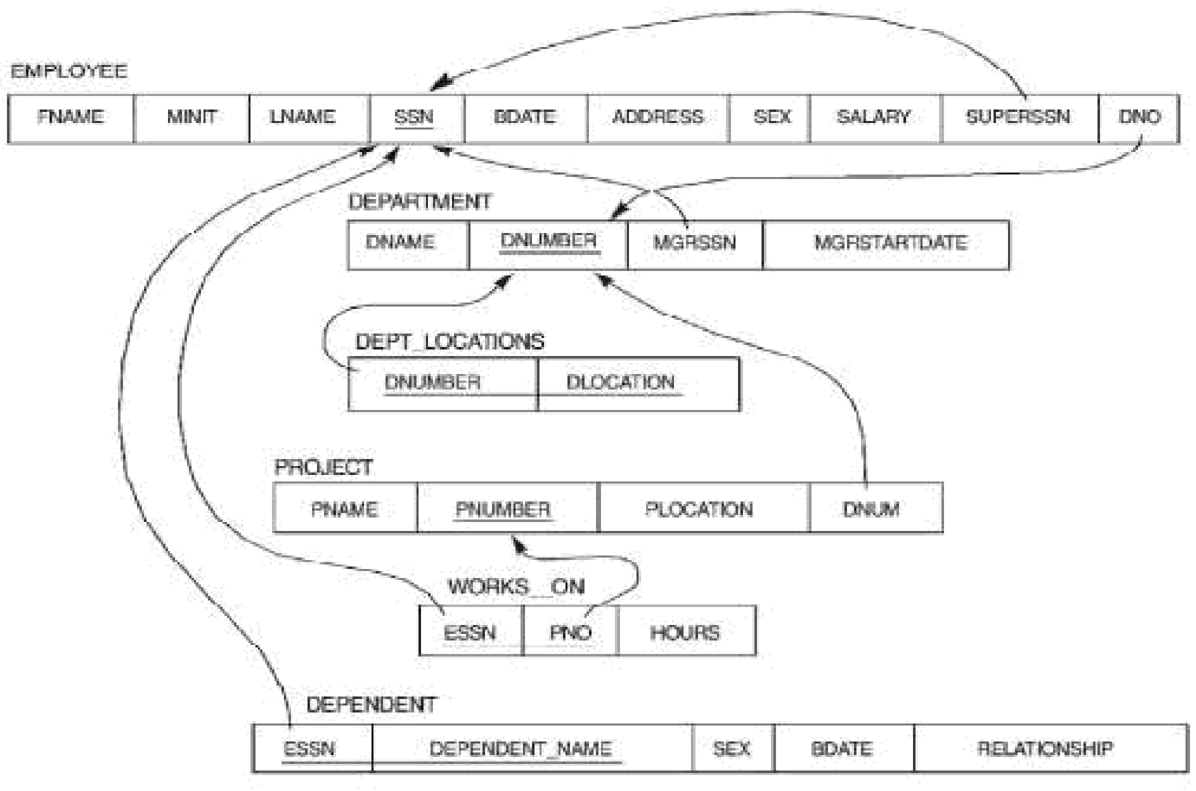
- CREATE SCHEMA
- Specifies a new database schema by giving it a name. Example:
- create schema company authorization KUMAR;
- All users are not authorized to create schemas and its elements
- The privileges to create the schema, tables and other construct can explicitly be granted to the relevant users by DBA.
- SQL2 uses the concept of catalog.
- Catalog is a named collection of schemas in an SQL environment.
- A catalog contains a special schema called INFORMATION_SCHEMA, which provides information on all the schemas in the catalog and all the element descriptors of all the schema in the catalog to authorized users.
- Integrity constraints can be defined for the relations and between the relations of same schema.
- Schemas within the same catalog can also share certain elements such as domain definitions.

CREATE TABLE Command in SQL

- The CREATE TABLE command is used to specify a new base relation by giving its name, and specifying each of its attributes and constraints.
- The attributes are specified first by giving their name, their data types and constraints for the attributes like NOT NULL, CHECKS etc. specified on each attribute.
- The key, entity integrity and referential integrity constraints can be specified within the CREATE TABLE command after the attributes declaration, or can be added later using the ALTER command.
- An SQL relation is defined using the create table command:

```
CREATE TABLE R(A1 D1, A2 D2, ..., An Dn,
        (integrity-constraint_1),
        ...,
        (integrity-constraint_k))
```

- R is the name of the relation
- Each Ai is an attribute name in the schema of relation R
- Di is the data type of values in the domain of attribute Ai



Relations in Company Schema

CREATE TABLE EMPLOYEE
(FNAME VARCHAR (15) **NOT NULL**,
MINIT CHAR,LNAME VARCHAR (15) **NOT**
NULL,
SSN CHAR (9), BDATE DATE,
ADDRESS VARCHAR2 (30), GENDER
CHAR,SUPERSSN CHAR (9),
DNO INT **NOT**
NULL,**PRIMARY**
KEY (SSN),
FOREIGN KEY (SUPERSSN) REFERENCES EMPLOYEE (SSN),
FOREIGN KEY (DNO) REFERENCES DEPARTMENT (DNUMBER));

CREATE TABLE DEPARTMENT
(DNAME VARCHAR2 (15) **NOT**
NULL,DNUMBER INT **NOT**
NULL,

MGRSSN CHAR (9) **NOT**
NULL,MGRSTARTDATE
DATE, **PRIMARY KEY**
(DNUMBER), **UNIQUE**
(DNAME),
FOREIGN KEY (MGRSSN) REFERENCES EMPLOYEE (SSN));

CREATE TABLE LOCATIONS
(DNUMBER INT **NOT NULL**,
DLOCATION VARCHAR2 (15) **NOT**
NULL,MGRSTARTDATE DATE,
PRIMARY KEY (DNUMBER, DLOCATION),
FOREIGN KEY (DNUMBER) REFERENCES DEPARTMENT);

CREATE TABLE PROJECT
(PNAME VARCHAR2 (15) **NOT**
NULL,PNUMBER INT **NOT**
NULL,
PLOCATION VARCHAR2 (15) **NOT**
NULL,DNUM INT **NOT NULL**,
PRIMARY KEY (PNUMBER),
UNIQUE (PNAME),
FOREIGN KEY (DNUM) REFERENCES DEPARTMENT (DNUMBER));

CREATE TABLE WORKS_ON
(ESSN CHAR (9) **NOT**
NULL,PNO INT **NOT**
NULL,
HOURS DECIMAL (3,1) **NOT**
NULL,DNO INT **NOT NULL**,
PRIMARY KEY (ESSN, PNO),
FOREIGN KEY (ESSN) REFERENCES EMPLOYEE (SSN),
FOREIGN KEY (PNO) REFERENCES PROJECT (PNUMBER));

CREATE TABLE DEPENDENT
(ESSN CHAR (9) **NOT NULL**,
DEPENDENT_NAME VARCHAR2 (15) **NOT NULL**, GENDER CHAR,
BDATE DATE,ADDRESS VARCHAR2 (30),
RELATIONSHIP VARCHAR2 (8),
PRIMARY KEY (ESSN, DEPENDENT_NAME),
FOREIGN KEY (ESSN) REFERENCES EMPLOYEE (SSN));

- Schema name can be explicitly attached with the relation name separated by period.

- CREATE TABLE COMPANY. EMPLOYEE…….
- EMPLOYEE table becomes part of Schema COMPANY

Attribute Data types and Domains in SQL

  - Numeric
  - Character
  - Bit-string
  - Boolean
  - Date
  - Time
  - Timestamp
  - Interval

- **Numeric:** integer number of various sizes and floating numbers of various precision
  - **INTEGER or INT:** Integer (a finite subset of the integers that is machine-dependent)without any decimal part.
  - **SMALLINT**: Small integer (a machine-dependent subset of the integer domain type)without any decimal part.
  - **REAL, DOUBLE** precision: Floating point and double-precision floating point numbers, with machine-dependent precision.
  - **FLOAT (n):** Floating point number, with user-specified precision of at least *n* digits.
  - **NUMERIC(i, j) or DECIMAL(I, j) or DEC(I, j)**: Fixed point number, with user-specifiedprecision of *i* as total no. of digits, with j digits to the right of decimal point.

- **Character:** consists of sequence of character either of fixed length or varying length, default length of character string is 1
  - **CHAR(n) or CHARACTER(n):** Fixed length character string, with user-specified lengthn.
  - **VARCHAR(n) or CHAR VARYING(n) or CHARACTER VARYING(N):** Variable lengthcharacter strings, with user-specified maximum length n.
  - For fixed length strings **(CHARACTER)**, a shorter string is padded with blank spaceswhich are ignored at time of comparison (lexicographic order)
  - **CHARACTER LARGE OBJECT (CLOB)** for large text values (documents)

- **Bit-string:** consists of sequence of bits, either fixed length or varying length, default length of bit string is 1. BLOB is also available to specify columns that have large binary values, such as images.

- **Boolean:** it has three values TRUE, FALSE or UNKNOWN (for null)

- **Date:** It has ten positions, in the form YYYY-MM-DD, components are YEAR, MONTH, DAY

- **Time:** it has at least eight positions, in the form HH:MM:SS, components are HOUR, MINUTE, SECOND
    - Only valid date & time is allowed, comparison operators can be used.
    - 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:** A timestamp includes the DATE and TIME fields, plus a minimum of six positions for decimal fractions of seconds and an optional WITH TIME ZONE qualifier.
    - **TIME WITH TIME ZONE**: this data type includes an additional six positions for specifying the displacement from the standard universal time zone, which is in the range +13:00 to 12:59 in the units of HOURS:MINUTES
    - If **WITH TIME ZONE is not included**, the default is the local time zone for the SQL session.

- **INTERVAL**: Interval data type specifies an interval – a relative value that can be used toincrement or decrement an absolute value of a date, time or timestamp.
    - Intervals are qualified to be either YEAR/MONTH or DAY/TIME intervals.
    - It can be positive or negative when added to or subtracted from an absolute value, the result is an absolute value.

Domains in SQL

- A domain can be declared and the domain can be used with several attributes.
- CREATE DOMAIN ENO_TYPE AS CHAR(9)
- ENO_TYPE can be used in place of CHAR(9) with SSN, ESSN, MGRSSN.
- Data type of domain can be changed that will be reflected for the numerous attributes in the schema and improves the schema readability.

Specifying Basic constraints in SQL

- Attribute constraints
- Attributes defaults
- Key constraints
- Referential integrity constraints

Specifying Attribute constraints & Attribute defaults

- A constraint **NOT NULL** may be specified on an attribute if null can not be permitted for thatattribute.
- A **DEFAULT** clause is used to declare a default value for an attribute in absence of actualvalue.
    - Whenever an explicit value is not provided for the attribute, default value will be appended.
- A clause **CHECK** is used to restrict the domain values for an attribute.

```
CREATE TABLE EMPLOYEE
                    (
                    ENO VARCHAR2 (6) CHECK (ENO LIKE 'E%'),
                    ENAME VARCHAR2 (10) CHECK (ENAME = UPPER
                    (ENAME)),ADDRESS VARCHAR2 (10) NOT NULL,
                    STATE VARCHAR2 (10) CHECK (STATE IN ('KARNATAKA',
                    'KERALA')),
                    EMP_TYPE CHAR (1) DEFAULT 'F'
                    );
```

Specifying Key constraints

- Constraints can be specified on a table, including keys and referential integrity.
- The **PRIMARY KEY** clause specifies the single (or composite) primary key constraint.

```
….DNUM INT PRIMARY KEY….
….PRIMARY KEY(ESSN, PNUM)….
```

- A **UNIQUE** clause is used to specify alternate keys.

```
….DNAME CHAR(9) NOT NULL UNIQUE….
```

```
CREATE TABLE EMPLOYEE
                    (   SSN CHAR(9),
                        FNAME   VARCHAR2(15)   NOT
                        NULL, LNAME VARCHAR2(15)
                        NOT NULL, BDATE DATE,
                        ADDRESS
                        VARCHAR2(30),
                        PRIMARY KEY (SSN)
                    );
```

Specifying Referential integrity constraints

- A clause FOREIGN KEY can be specified for the foreign key constraint to implement therelationship between the relations i.e. referential integrity.

- A referential integrity can be violated when tuples are inserted, deleted or foreign key valueis updated.
- We can specify CASCADE, SET NULL or SET DEFAULT on referential integrity constraints(foreign keys).
- An option must be qualified with either ON DELETE or ON UPDATE.
- Possible options for referential triggered actions:
  - ON DELETE SET DEFAULT
  - ON DELETE SET NULL
  - ON DELETE CASCADE
  - ON UPDATE CASCADE
  - ON UPDATE SET DEFAULT
  - ON UPDATE SET NULL

CREATE TABLE EMPLOYEE
(FNAME VARCHAR2(15) **NOT NULL**, LNAME VARCHAR2(15) **NOT NULL**, SSN CHAR (9), ENO CHAR (5) ,
BDATE DATE, ADDRESS VARCHAR2(30), SUPERSSN CHAR (9),
DNO INT **NOT NULL**,
PRIMARY KEY (SSN),
FOREIGN KEY (SUPERSSN) **REFERENCES** EMPLOYEE (SSN) **ON DELETE SET DEFAULT ON UPDATECASCADE,**
FOREIGN KEY (DNO) **REFERENCES** DEPARTMENT (DNUMBER) **ON DELETE CASCADE ON UPDATECASCADE**);

Naming the constraints

- A constraint name is used to identify a particular constraint in case the constraint must be dropped or modified later.
- Giving names to constraints is optional.
- Name of the constraint should be unique within a schema.

```
CREATE TABLE EMPLOYEE
        (ENO VARCHAR2(10) CONSTRAINT PK_EMPLOYEE
        PRIMARY KEY,ENAME VARCHAR2(30),
        ADDRESS VARCHAR2(100) NOT NULL,
        STATE VARCHAR2(10), DT_OF_JOINING DATE,
        CONSTRAINT CHK_ENO CHECK (ENO LIKE 'E%'),
        CONSTRAINT CHK_ENAME CHECK (ENAME=UPPER(ENAME)),
        CONSTRAINT CHK_STATE CHECK (STATE IN
        ('KARNATAKA','KERALA')),
        CONSTRAINT FK_DNO_DEPT FOREIGN KEY (DNO) REFERENCES
        DEPARTMENT);
```

Schema Change Statements in

SQLDrop commands

- **DROP** command is used to remove an element & its definition
- **DROP SCHEMA**: To drop schema
- **DROP TABLE**: To drop table
- The relation (or schema) can no longer be used in queries, updates, or any other commands since its description no longer exists
- There are two DROP behavior options **CASCADE** and **RESTRICT**
- A **CASCADE** option is used to remove schema and its all tables, views and all other elements

Examples:

- DROP SCHEMA COMPANY CASCADE;
  - Schema and its all element are dropped.
- DROP TABLE EMPLOYEE CASCADE;
  - Table and its all element are dropped.
- If RESTRICT option is used instead of CASCADE
  - A schema is dropped only if it has no elements, otherwise error will be shown.
  - A table is dropped only if it is not referenced in any constraint by any other table.

ALTER command

- The definition of a base table can be changed by using ALTER TABLE command.
- The various possible options include
  - adding a column
  - dropping a column
  - changing the definition of column
  - adding and dropping the table constraints
- ALTER TABLE command is 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 CHAR(12);
- The database users must enter a value for the new attribute JOB for each EMPLOYEE tuple.

- This can be done using the UPDATE command or by DEFAULT clause.
- ALTER TABLE command is also used to drop an attribute from one of the base relations.
- To drop a column, an option CASCADE or RESTRICT should be chosen for drop behaviour.
- ALTER TABLE EMPLOYEE DROP JOB;
- If CASCADE, all relations and views that reference the column are dropped automaticallyfrom the schema along with the column.
- ALTER TABLE command is also used to modify an attribute of one of the base relations.
- ALTER TABLE EMPLOYEE ALTER MGRSSN DROP DEFAULT;
- ALTER TABLE EMPLOYEE ALTER MGRSSN SET DEFAULT 999;

Adding or dropping constraints

- ALTER TABLE EMPLOYEE DROP CONSTRAINT FK_SUPERSSN CASCADE;

Structure of Basic Queries in SQL (The SELECT_FROM_WHERE)

- Basic form of the SQL SELECT statement is called 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

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

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

$\Pi_{\text{BDATE, ADDRESS}}(\sigma_{\text{FNAME='John' AND MINIT='B' AND LNAME='Smith'}}(\text{Employee}))$

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

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

```
    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)
- Retrieve FNAME, LNAME, ADDRESS is a project operation
- (DNUMBER=DNO) is a join condition (corresponds to a JOIN operation in relational algebra)

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.

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

- 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
- Retrieve PNUMBER, DNUM, LNAME, BDATE, ADDRESS is a project operation
- Project location is 'Stafford'

Ambiguous Attribute names, Aliasing, and Tuple Variables

- In SQL, we can use the same name for two (or more) attributes as long as the attributes arein 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.DNA
ME

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

```
SELECT FNAME, LNAME, ADDRESS
                    FROM EMPLOYEE, DEPARTMENT
```

```
                              WHERE DNAME='Research' AND
                                        EMPLOYEE.DNUMBER = DEPARTMENT.DNUMBER
```

Q1B

```
SELECT FNAME, LNAME, ADDRESS
                          FROM EMPLOYEE E, DEPARTMENT D
                              WHERE DNAME='Research' AND
                                        E.DNUMBER = D.DNUMBER;
```

- 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.

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

- The alternate relation names E and S are called aliases.
- E and S can be thought as two different copies of EMPLOYEE; E represents employees inrole of supervisees and S represents employees in role of supervisors

Basic Queries in SQL

- Aliasing can also be used in any SQL query for convenience and AS keyword can also beused to specify aliases

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

Unspecified WHERE-clause

- A missing WHERE-clause indicates no condition; hence, all tuples of the relations in theFROM-clause are selected

Query 9: Retrieve the SSN values for all employees.

```
                    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

Q1O
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 * (Asterisk)

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

Q1C: SELECT * FROM EMPLOYEE WHERE DNO=5;

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

Tables as Sets in SQL

- SQL usually treats a table not as a set but rather as a multiset, duplicate tuples can appear more than once in a table, and in the result of a query.
- SQL does not automatically eliminate duplicate tuples in the result of queries because:
    - Duplicate elimination is expensive.
    - The user may want to see the duplicates in the result of query.
    - For an aggregate function, elimination of tuples is not desired.
- An SQL table with a Primary key is restricted to being a set, since the key value must be distinct in each tuple.
- Keyword DISTINCT can be used in SELECT clause if duplicates have to be eliminated in thequery result.

SELECT SALARY FROM EMPLOYEE
or SELECT ALL SALARY FROM

EMPLOYEE SELECT DISTINCT

SALARY FROM EMPLOYEE

Set operations in SQL

- SQL has directly incorporated some set operations
    - union operation (UNION)
    - set difference (MINUS or EXCEPT)

  - intersection (INTERSECT)
- 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
- If duplicates have to be retained
  - union operation (UNION ALL)
  - set difference (MINUS ALL or EXCEPT ALL)
  - intersection (INTERSECT ALL)
- (All may not work in every tool)

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

```
(SELECT DISTINCT PNUMBER
                        FROM PROJECT, DEPARTMENT,
                            EMPLOYEE WHERE DNUM=DNUMBER
                            AND MGRSSN=SSN
                                      AND LNAME='Smith')
UNIO
N                       (SELECT DISTINCT
                            PNUMBER FROM
                            WORKS_ON, EMPLOYEE
                                      WHERE ESSN=SSN AND LNAME=' Smith')
```

Make a list of all project numbers for projects that involve an employee whose last name is
'Smith' as a worker and as a manager of the department that controls the project.

```
(SELECT DISTINCT PNUMBER
                        FROM PROJECT, DEPARTMENT,
                            EMPLOYEE WHERE DNUM=DNUMBER
                            AND MGRSSN=SSN
                                      AND LNAME='Smith')
INTERSE
CT                      (SELECT DISTINCT
                            PNUMBER FROM
                            WORKS_ON, EMPLOYEE
                                      WHERE ESSN=SSN AND LNAME=' Smith')
```

Make a list of all project numbers for projects that involve an employee whose last name is
'Smith' as a manager of the department that controls the project but not as a worker.

```
(SELECT DISTINCT PNUMBER
FROM PROJECT, DEPARTMENT,
    EMPLOYEE WHERE DNUM=DNUMBER
    AND MGRSSN=SSN
```

```
                              AND LNAME='Smith')
MINU
S                         (SELECT DISTINCT
                              PNUMBER FROM
                              WORKS_ON, EMPLOYEE
                                        WHERE ESSN=SSN AND LNAME=' Smith')
```

Substring Pattern Matching

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

Query 12: Retrieve all employees whose address is in Houston, Texas. (Here, the value of the ADDRESS attribute must contain the substring 'Houston,Texas'.)

```
SELECT FNAME, LNAME
                          FROM EMPLOYEE
                              WHERE ADDRESS
                              LIKE
                                        '%Houston,Texas%'
```

Query 12A: Retrieve all employees who were born during the 195Os. (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.

```
SELECT FNAME, LNAME
                          FROM EMPLOYEE
                              WHERE BDATE LIKE ' __ 195_';
                                        (or '%195_')
```

Arithmetic Operators

- The standard arithmetic operators '+', '-', '*', '/' (for addition, subtraction, multiplication, and division, respectively) can be applied to numeric values in an SQL query result.

Query 13: Show the resulting salaries if every employee working on the 'ProductX' project is
given a 10% raise.

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

Comparison Operators

Query 14: Retrieve all the employees in department 5 whose salary is between Rs. 30,000 andRs. 40,000.

```
SELECT *
                        FROM EMPLOYEE
                            WHERE (SALARY BETWEEN 30000 AND 40000) AND DNO = 5;
SELECT
*                       FROM EMPLOYEE
                           WHERE (SALARY >= 30000) AND (SALARY <= 40000) AND DNO = 5;
```

Ordering of Query results

- The **ORDER BY** clause is used to sort the tuples in a query result based on the values ofsome attribute(s)
- The default order is in ascending order of values.
- The keyword **DESC** can be used if descending order is required; the keyword ASC can beused to explicitly specify ascending order, even though it is the default

Query 15: 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, first name.

```
SELECT DNAME, LNAME, FNAME, PNAME
                        FROM DEPARTMENT, EMPLOYEE, WORKS_ON,
                                     PROJECTWHERE DNUMBER=DNO AND
                                     SSN=ESSN
                                              AND PNO=PNUMBER
                                                        ORDER BY DNAME, LNAME, FNAME;
```

- ORDER BY DNAME, LNAME, FNAME DESC
- ORDER BY DNAME, LNAME DESC, FNAME DESC
- ORDER BY DNAME DESC, LNAME, FNAME DESC

More Complex

QueriesNULLS in

SQL

- 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 fromother NULL values, so equality comparison is not appropriate.

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

```
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

Renaming Attributes

- Any attribute which appears in the result can be renamed by adding the qualifier ASfollowed by the desired new name.
- AS construct can be used for both attribute names and relation names and can be used inboth SELECT and FROM clauses.

```
Select ename as EMPNAME
                    from emp as EMPLOYEE
```

Nested Queries

- Some queries require that existing values in database be fetched and then used in a comparison condition.
- A complete SELECT query, called a nested query, can be specified within the WHERE-clauseof another query, called the outer query

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

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

- 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 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

Query 4A: 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.

```
SELECT DISTINCT PNUMBER FROM PROJECT WHERE PNUMBER IN
```

```
                    (SELECT PNUMBER
                       FROM PROJECT, DEPARTMENT, EMPLOYEE
                              WHERE DNUM=DNUMBER AND
                                     MGRSSN=SSNAND
                                     LNAME='Smith')
                    OR
                    PNUMBER
                    IN
                       (SELECT PNUMBER
                              FROM WORKS_ON,
                                     EMPLOYEEWHERE
                                     ESSN=SSN
                                            AND LNAME=' Smith')
```

- In addition of IN operator, a number of other comparison operator like:
- = (if nested query returns only single value)
- = ANY or =SOME
- >, <, <=, >=, <>
- ALL keyword can be used with all above operators.

```
SELECT FNAME, LNAME, ADDRESS
                    FROM EMPLOYEE
                       WHERE SALARY >
                       ALL
                              (SELECT SALARY
                                     FROM EMPLOYEE
                                            WHERE DNO=5)
```

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.
- A reference to an unqualified attribute refers to the relation declared in the innermost nested query

Query 16: Retrieve the name of each employee who has a dependent with the same name and same gender as the employee.

```
SELECT E.ENAME
FROM EMPLOYEE
AS EWHERE E.SSN
IN
                    (SELECT ESSN
                    FROM
                    DEPENDENT
                    WHERE E.GENDER =
                    D.GENDER AND
                    E.NAME=DEP_NAME)
```

Single Block Query: A query written with nested SELECT... FROM... WHERE... blocks and usingthe = or IN comparison operators can always be expressed as a single block query.

Q16A

```
SELECT E.ENAME
FROM EMPLOYEE E, DEPENDENT D
                    WHERE E.SSN = D.ESSN AND
                       E.ENAME = D.DEP_NAME
                       AND
                               E.GENDER = D.GENDER;
```

The Exists Function

- **EXISTS** is used to check whether the result of a correlated nested query is empty (contains no tuples) or not
- **EXISTS** and **NOT EXISTS** are usually used in conjunction with a correlated nested query
- **EXISTS** returns **TRUE** if there is at least one tuple in the result of the query, otherwise itreturns false.
- **NOT EXISTS** returns **TRUE** if there is no tuples in the result of the query, otherwise it returnsfalse.

Query 16B: Retrieve the name of each employee who has a dependent with the same name as the employee.

```
SELECT E.ENAME FROM EMPLOYEE E
                    WHERE
                       EXISTS
                       (SELECT *
                               FROM DEPENDENT D
                               WHERE E.SSN=D.ESSN
                               AND
                               E.ENAME=D.DEP_NAME
                               ANDE.GENDER =
                               D.GENDER)
```

Query 6: Retrieve the names of employees who have no dependents.

```
SELECT FNAME, LNAME
                    FROM EMPLOYEE
                       WHERE NOT
                       EXISTS(SELECT
                       *
                               FROM
DEPENDENT
WHERE
SSN=ESSN)
```

- 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

Q7. 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
(SELECT                              EXISTS
*
                                              FROM
                                              DEPARTMENT
                                              WHERE
                                              SSN=MGRSSN);
```

- 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 operator CONTAINS
- The CONTAINS operator compares two sets of values, and returns TRUE if one set containsall values in the other set (reminiscent of the division operation of algebra).

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

```
SELECT FNAME, LNAME FROM EMPLOYEE
                        WHERE ((SELECT PNO,
                               ESSNFROM
                               WORKS_ON
                               WHERE SSN=ESSN)
                        CONTAINS
                              (SELECT
                              PNUMBER
                              FROM PROJECT
```

```
                                WHERE
                                DNUM=5))
```

- 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 Explicit Sets in SQL

- 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 number1, 2, or 3.

```
SELECT DISTINCT ESSN
                    FROM WORKS ON
                       WHERE PNO IN
                       (1,2,3)
```

Renaming Attributes

Query 8A: For each employee, retrieve the employee's name, and the name of his or her
immediate supervisor.

```
SELECT E.NAME AS Supervisee_name,
                    S.NAME AS
Superviser_name,FROM EMPLOYEE  AS  E,
EMPLOYEE  AS  S
                       WHERE E.SUPERSSN=S.SSN
```

Joining Tables

- We can specify a "joined relation" in the FROM-clause and it looks like any other relation but is the result of a join
- It allows the user to specify different types of joins (regular THETA JOIN, NATURAL JOIN,LEFT OUTER JOIN, RIGHT OUTER JOIN, CROSS JOIN, etc)

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 DNUMBER=DNO AND DNAME='Research';
```

could be written as:

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

Further,

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

could be written as:

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

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

can be written as for outer join:

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

Q2 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.
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
- Some SQL implementations may not allow more than one function in the SELECT-clause

Query 19: Find the sum of the salary of all employees, the maximum salary, the minimumsalary, and the average salary among employees.

```
SELECT SUM(SALARY), MAX(SALARY), MIN(SALARY), AVG(SALARY) FROM
EMPLOYEE;
```

Query 20: Find the sum of the salary of all employees, the maximum salary, the minimum
salary, and the average salary among employees who work for the 'Research' department.

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

Or

```
  SELECT SUM(SALARY), MAX(SALARY), MIN(SALARY),
                                 AVG(SALARY)
                                  FROM (EMPLOYEE JOIN DEPARTMENT ON
                                                   DNO=DNUMBER)
                            WHERE DNAME= 'Research';
```

Query 21: Retrieve the total number of employees.

```
SELECT COUNT(*) FROM EMPLOYEE;
```

Query 22: Retrieve the total number of employees in the Research Department.

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

Query 23: Count the distinct salary values in the database.

```
SELECT COUNT(DISTINCT SALARY)
                          FROM EMPLOYEE;
```

Query 5: List the names of all employees with two or more dependents.

```
SELECT LNAME, FNAME
                          FROM EMPLOYEE
                              WHERE (SELECT COUNT (*) FROM
                                           DEPENDENTWHERE SSN = ESSN)
                                           >=2;
```

Grouping

- SQL has a GROUP BY-clause for specifying the grouping attributes, which must also appearin the SELECT-clause
- 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)
- Then aggregate function is applied to each subgroup independently

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

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

- The EMPLOYEE tuples are divided into groups--each group having the same value for thegrouping 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

Query 25: For each project, retrieve the project number, project name, and the number ofemployees who work on that project.

```
SELECT PNUMBER, PNAME, COUNT (*)
                          FROM PROJECT,
                              WORKS_ONWHERE
                              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 onindividual tuples)

Query 26: For each project on which more than two employees work, retrieve the projectnumber, project name, and the number of employees who work on that project.

```
SELECT PNUMBER, PNAME, COUNT (*)
                          FROM PROJECT,
                              WORKS_ONWHERE
                              PNUMBER=PNO
                                        GROUP BY PNUMBER,
                                                  PNAMEHAVING
                                                  COUNT (*) >2;
```

Query 27: For each project, retrieve the project number, project name, and the number

ofemployees from department 5 who work on that project.

```
SELECT PNUMBER, PNAME, COUNT (*)
                        FROM PROJECT, WORKS_ON, EMPLOYEE
```

```
                WHERE PNUMBER = PNO
                        AND SSN = ESSN AND DNO = 5
                                GROUP BY PNUMBER, PNAME;
```

Query 28: For each department that has more than five employees, retrieve the departmentnumber, and the number of its employees who are earning more than 40000.

```
SELECT DNUMBER, COUNT (*)
            FROM EMPLOYEE, DEPARTMENT
WHERE DNUMBER = DNO AND SALARY > 40000 AND
DNO IN(SELECT DNO FROM EMPLOYEE
                                GROUP BY DNO
                                HAVING COUNT
                                (*)>5)
                GROUP BY DNUMBER;
```

Summary of SQL Queries

- A query in SQL can consist of up to six clauses, but only the first two, SELECT and FROM, aremandatory. 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>]
```

- 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, andfinally the SELECT-clause.

Introduction to SQL Programming Techniques

Most database access is through software programs that implement database applications. The software is generally developed in a general-purpose programming language such as java, COBOL, C/C++ etc.

- Host Language: general-purpose programming language
- Data sub-language: SQL
- Database programming language: Special language developed for writing database applications.

Database Programming: Issues and Techniques

Most database systems have an interactive interface where these SQL commands can be typed directly giving input to database. The interactive interface is convenient for ad-hoc queries. But in practice, database interactions are executed through programs that are carefully designed and tested. These application programs or database applications are used as canned transactions by the end users. A database can be accessed through an application program thatimplements a Web Interface.

Approaches to Database Programming

Several techniques exist for including database interactions in application programs:

- Embedding database commands in a general-purpose programming language
- Using a library of database functions
- Designing a brand new language
- First two approaches are common but it suffers from the problem of impedance mismatch.
- Third approach is more appropriate for applications that have intensive databaseinteractions.

Embedded SQL

Embedding database commands in a general-purpose programming language:

Database statements are embedded into the host programming language, they are identified by a special prefix. For example, the prefix for embedded SQL is the string EXEC SQL, which precedes all SQL commands in a host language program. A precompiler or preprocessor scans the source program code to identify database statements and extract them for processing by the DBMS. They are replaced in the program by function call to the DBMS-generated code.

A library of database functions (Using a library of database functions)

These functions are made available to the host programming language for the database calls. For example, there could be functions to connect to a database, execute a query, execute an update etc. The actual database query, and update commands and any other necessary information are included as parameters in the function calls. This approach provides Application programming interface (API) for accessing a database from application programs.

A brand new language (Designing a brand new language)

A database programming language is designed from scratch to be compatible with the database model and query language. Additional programming structures such as loops and conditional statements are added to the database language to convert it into a full-fledged programming language

Impedance Mismatch

A problem occurs because of difference between the database model and the programming language model. For example the practical relational model has three main constructs: attributes and their data types, tuples and tables. The first problem that may occur is that the data type of the programming language differs from attribute data types in the data model. Hence it is necessary to have a binding for each host programming language that specifies for each attribute type and the compatible programming language types. Data types in C++ and java differ from the SQL data types. Another problem occurs because the results of most queries are sets or multisets of tuples, and each tuple is formed of sequence of attribute values.In the program it is often necessary to access individual data values within individual tuples for printing or processing. Hence a binding is needed to map the query result data structure, whichis a table, to an appropriate data structure in the programming language. A mechanism is needed to loop over the tuples in a query result in order to access a single tuple at a time and extract individual values from the tuple. The extracted attribute values are typically copied to appropriate program variables for further processing by the program. A cursor or iteratorvariable is used to loop over the tuples in a query result. Individual values within each tuple are typically extracted into distinct program variables of the appropriate type. Impedance mismatch is less of problem when a special database programming language is designed that uses the same data model and data types as the database model. One example of such a language is Oracle’s PL/SQL.
For object databases, the object data model is quite similar to the data model of the Java programming language. Thus impedance mismatch is greatly reduced when Java is used as host language for accessing a Java-compatible object database.

Typical sequence of interaction in Database programming

Generally, architecture for database access is the client-server model. A client program handlesthe logic of a software application but includes some calls to one or more database servers to access or update the data. Writing such an application, a common sequence of interaction is followed.

Step 1:

- When the client program requires access to a particular database, the program must first establish or open a connection to the database server.
- Typically, this involves specifying the Internet address (URL) of the machine where the database server is located, plus providing a login account name and password for database access.

Step 2:

- Once the connection is established, the program can interact with the database by submitting queries, updates and other database commands.
- In general most types of SQL statements can be included in an application program.

Step 3:

- When the program no longer needs access to a particular database, should terminate or close the connection to the database.
- A program can access multiple databases if needed.
- In some database programming approaches, only one connection can be active at a time, whereas in other approaches multiple connections can be established simultaneously.

Embedded SQL

SQL statements can be embedded in a general purpose programming language (host language) such as C, ADA, COBOL, or PASCAL. Most SQL statements including data or constraint definitions, queries, updates or view definitions can be embedded in the host language program.
Embedded SQL statement is distinguished from programming language by prefixing it with the keywords EXEC SQL so that preprocessor can separate embedded SQL statements from the host language code. The SQL statements can be terminated by a semicolon (;) or a matching END-EXEC.

Using C as the host programming language

Within an embedded SQL command, specially declared C program variables can be referred. These variables are called shared variables because these are used by both C and embedded SQL. Shared variables are prefixed by a colon(:) when they appear in an SQL Statement. This distinguishes program variables names from the names of database schema constructs such as attributes and relations. It allows program variables to have the same names as attribute names.
A few of the common bindings of C types to SQL types are as follows:

- The SQL types INTEGER, SMALLINT, REAL, and DOUBLE are mapped to the C types long, short, float, and double respectively.
- Fixed-length and varying-length strings (CHAR [i], VARCHAR [i]) in SQL can be mapped to array of characters (char [i+1], varchar [i+1]) in C that are one character longer than the SQL type because strings in C are terminated by a NULL character (\0),which is not part of character itself.

C program segment declaring variables

```
int loop;
EXEC SQL BEGIN DECLARE SECTION;
varchar fname [16], lname [16],  address  [31];
char ssn [11], bdate [10], minit [2];
float salary, raise;
int SQLCODE; char SQLSTATE
[6]; EXEC SQL END DECLARE
SECTION;
```

- The variables SQLCODE and SQLSTATE are used to communicate errors and exception conditions between the database system and the program.

Connecting to the Database

- The SQL command for establishing a connection to a database has the following form:

```
CONNECT TO<server name>AS<connection name>
AUTHORIZATION<user account name and
password>;
```

- In general, since a user or program can access several database servers, several connections can be established, but only one connection can be active at any point in time.
- The programmer or user can use the <connection name> to change from the currently active connection to a different one by using the following command:

SET CONNECTION <connection name>;

- Once a connection is no longer needed, it can be terminated by the following command:
  DISCONNECT<connection name>;

Communicating between the program and the DBMS

Using SQLCODE and SQLSTATE:

- The two special communication variables that are used by the DBMS to communicate exception or error conditions to the program are SQLCODE and SQLSTATE.
- The SQLCODE variable shown in the example is an integer variable.
- After each database command is executed, the DBMS returns a value in SQLCODE.
- A value of 0 indicates that the statement was executed successfully by the DBMS.
- If SQLCODE>0 (or, more specifically, if SQLCODE=100), this indicates that no more data areavailable in a query result.
- If SQL<0,this indicates some error has occurred.
- In some systems-for example, in the Oracle RDBMS-SQLCODE is field in record structurecalled SQLCA (SQL communication area), so it is referenced as SQLCA.SQLCODE.
- In this case, the definition of SQLCA must be included in the C program by including thefollowing line:

EXEC SQL includes SQLCA;

- In the later versions of the SQL standard, a communication variable called SQLSTATE was added, which is a string of five characters.
- A value of ‘00000’ in SQLSTATE indicates no error or exception; other variables indicatesvarious errors or exceptions.
- For example, ‘02000’ indicates ‘no more data’ when using SQLSTATE.
- Currently, both SQLSTATE and SQLCODE are available in the SQL standard.
- Many of the error and exception codes returned in SQLSTATE are supposed to be standardized for all SQL vendors and platforms, where the codes returned in SQLCODE are not standardized but are defined by the DBMS vendor.
- Hence is better to use SQLSTATE because this makes error handling in the application programs independent of a particular DBMS.

Example of Embedded SQL

Retrieving single tuple with Embedded SQL:

- Program segment reads a ssn of an employee and prints some information from corresponding EMPLOYEE record in the database.
- INTO clause specifies the program variables into which attribute values from the databaseare retrieved.
- SQLCODE is used for communication between database and program.
- If SQLCODE returns 0, the statement is executed without errors or exception

C program segment declaring variables

```
loop =1;
while (loop)
{
                    prompt (“Enter a Social Security Number:”, ssn);
EXEC SQL
    select Fname, Minit, Lname, Address, Salary
    into :fname, :minit, :lname, :address, :salary
    from EMPLOYEE where Ssn = :ssn;
    if (SQLCODE ==0) printf (fname, minit, lname, address, salary)
    else printf (“Social Security Number does not exist);
prompt (“More Social Security Number (enter 1 for yes, 0 for no):”, loop);
}
```

Retrieving multiple tuple with Embedded SQL using cursors:

- Cursor is declared when SQL command is declared in program
- An OPEN CURSOR command fetches the query result from the database and sets the cursor to a position before the first row in the result of the query.
- FETCH command moves the cursor to the next row in the result of the query, and copying its attribute values into C variables
- CLOSE CURSOR is issued to end the processing.
- The cursor variable is basically an iterator that iterates over the tuples in the query result one tuple at a time.
- To determine when all the tuples in the result of the query have been processed, the communication variable SQLCODE is checked.
- If a FETCH command is issued that result in moving the cursor till the last tuple in the result of the query, a positive value is returned in SQLCODE, indicating that no data was found.
- The programmer uses positive value of SQLCODE to terminate a loop over the tuple in the query result.
- In general, numerous cursors can be opened at the same time.
- When a cursor is defined for rows that are to be modified, we must add the clause FOR UPDATE OF in the cursor declaration and list the names of any attributes that will be updated by the program.

- If rows are to be deleted, the keywords FOR DELETE must be added without specifying anyattributes.
- If the result of the query is to be used for retrieval purposes only, there is no need to include the FOR UPDATE OF
- In the embedded UPDATE command, the condition WHERE CURRENT OF <cursor_name>specifies that the current tuple, referenced by the cursor is the one to be updated.

```
Prompt ("Enter the Department Name:", dname);
EXEC SQL
                        select Dnumber into :dnumber
                        from DEPARTMENT where Dname =
:dname;EXEC SQL DECLARE EMP CURSOR FOR
                        select Ssn, Fname, Minit, Lname, Salary
                        from    EMPLOYEE    where    Dno    =
                        :dnumberFOR UPDATE OF Salary;
EXEC SQL OPEN EMP
EXEC  SQL  FETCH  from  EMP  into  :ssn,  :fname,  :minit,  :lname,
:salary;while (SQLCODE == 0)
{
printf ("Employee Name is:", Fname, Minit,
Lname);prompt ("Enter the raise amount:", raise);
EXEC SQL
                            update
EMPLOYEEset Salary = Salary +:raise
where CURRENT OF EMP;
EXEC SQL FETCH from EMP into :ssn, :fname, :minit, :lname, :salary;
}
EXEC SQL CLOSE EMP;
```

- The general form of a cursor declaration is as follows:

DECLARE <cursor_name> [INSENSITIVE] [SCROLL] CURSOR
[WITH HOLD] FOR <query
specification>[ORDER BY <ordering
specification>]
[FOR READ ONLY | FOR UPDATE [OF <attribute list>]];

- The default is that the query is for retrieval purposes (FOR READ ONLY)
- When the optional keyword SCROLL is specified in a cursor declaration, it is possible toposition the cursor in other ways than for purely sequential access.
- A fetch orientation can be added to the FETCH command, whose value can be one of NEXT,PRIOR, FIRST, LAST, ABOSULTE i, and RELATIVE i.

- i must evaluate to an integer value that specifies an absolute tuple position or a tuple position relative to the current cursor position respectively.
- The default fetch orientation is NEXT.
- The fetch orientation allows the programmer to move the cursor around the tuple in the query result with greater flexibility, providing random access by position or access in reverse order.
- When SCROLL is specified on the cursor, the general form a FETCH command is as follows, with the parts in square brackets being optional:

FETCH [[<fetch orientation>] FROM] <cursor name> INTO <fetch target list>;

- The ORDER BY clause orders the tuples so that the FETCH command will fetch them in the specified order.

Question Bank

1. What are the different attribute data types and domains in SQL? Explain.
2. Explain the following commands in SQL, with examples: DROP, CREATE, ALTER, UPDATE,ROLLBACK, CHECK, EXISTS/NOT EXISTS
3. Explain the various constraints in SQL with syntax and examples.
4. Discuss the INSERT, DELETE and UPDATE statements in SQL with examples.
5. Explain JOIN operations in SQL with syntax and examples.
6. Explain Nested Queries with examples.
7. Explain the usage of GROUP BY and HAVING clause with syntax and examples.
8. List and explain all the aggregate functions in SQL with syntax and suitable examples.
9. Questions on Formulation of SQL queries, given any database.
10. Write a note on issues and techniques of Database Programming.
11. Explain the concept of Embedded SQL with suitable examples.