MODULE 3
SQL :ADVANCE QUERIES

3.1 Data Definition, Constraints, and Schema Changes in SQL2

- *Structured Query Language (SQL)* was designed and implemented at IBM Research.
- Created in late 70‘s, under the name of SEQUEL
- A standard version of SQL (ANSI 1986), is called SQL86 or SQL1.
- A revised version of standard SQL, called SQL2 (or SQL92).
- SQL are going to be extended with objectoriented and other recent database concepts.
- Consists of
    - A Data Definition Language (DDL) for declaring database schemas
    - Data Manipulation Language (DML) for modifying and querying database instances
- In SQL, relation, tuple, and attribute are called *table, row*, and *columns* respectively.
- The SQL commands for data definition are *CREATE, ALTER*, and *DROP*.
- The *CREATE TABLE* Command is used to specify a new table by giving it a name and specifying its attributes (columns) and constraints.
- Data types available for attributes are:
    - *Numeric* integer, real (formated, such as *DECIMAL(10,2)*)
    - *CharacterString* fixedlength and varyinglength
    - *BitString* fixedlength, varyinglength
    - *Date* in the form YYYYMMDD
    - *Time* in the form HH:MM:SS
    - *Timestamp* includes both the *DATE* and *TIME* fields
    - *Interval* to increase/decrease the value of date, time, or timestamp

3.2 Basic Queries in SQL

- SQL allows a table (relation) to have two or more tuples that are identical in all their attributes values. Hence, an **SQL table** is not a set of tuple, because a set does not allow two identical members; rather it is a multiset of tuples.
- A basic query statement in SQL is the *SELECT* statement.
- The *SELECT* statement used in SQL has no relationship to the *SELECT* operation of relational algebra.

The SELECT Statement

The syntax of this command is:

SELECT <attribute list>
FROM <table list>
WHERE <Condition>;

Some example:

Query 0: Retrieve the birthday and address of the employee(s) whose name is _John B. Smith‘

Q0: SELECT BDATE, ADDRESS

FROM EMPLOYEE

WHERE FNAME = _John‘ AND MINIT =_B‘ AND LNAME = _SMITH‘

Query 1: Retrieve the name and address of all employee who work for the _Research‘ Dept.

Q1: SELECT FNAME, LNAME, ADDRESS

FROM EMPLOYEE, DEPARTMENT

WHERE DNAME = _Research‘ AND DNUMBER = DNO

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, ADDRESS, BDATE

FROM PROJECT, DEPARTMENT, EMPLOYEE

WHERE DNUM=DNUMBER AND MGRSSN=SSN AND PLOCATION = _Stafford‘

Dealing with Ambiguous Attribute Names and Renaming (Aliening)

Ambiguity in the case where attributes are same name need to qualify the attribute using DOT separator

e.g., WHERE DEPARTMENT.DNUMBER=EMPLOYEE.DNUMBER

More

Ambiguity in the case of queries that refer to the same relation twice

Query 8: For each employee, retrieve the employee‘s first and last name and the first and last name of his or her immediate superviso

Unspecified WHEREClause and Use of Asterisk (*)

A missing WHEREclause indicates no conditions, which means all tuples are selected In

case of two or more table, then all possible tuple combinations are selected

Example: Q10: Select all EMPLOYEE SSNs , and all combinations of EMPLOYEE SSN and DEPARTMENT DNAME

```
SELECT   SSN, DNAME

FROM       EMPLOYEE, DEPARTMENT
```

More

To retrieve all the attributes, use * in SELECT clause

Retrieve all employees working for Dept. 5

```
SELECT *

FROM    EMPLOYEE

WHERE DNO=5
```

Substring Comparisons, Arithmetic Operations, and Ordering

- like, binary operator for comparing strings
- %, wild card for strings
- _, wild card for characters
- ||, concatenate operation for strings

(name like '%a_') is true for all names having _a' as second letter from the end.

- Partial strings are specified by using '
- SELECT FNAME, LNAME
- FROMEMPLOYEE
  WHERE FNAME LIKE '%Mc%';

- In order to list all employee who were born during 1960s we have the followings:
- SELECT FNAME, LN

- SQL also supports addition, subtraction, multiplication and division (denoted by +, , *, and /, respectively) on numeric values or attributes with numeric domains.

Examples: Show the resulting salaries if every employee working on the 'ProductX' project is given a 10 percent raise.

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

Retrieve all employees in department number 5 whose salary between $30000 and $40000.

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

It is possible to order the tuples in the result of a query.

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

The default order is in ascending order, but user can specify

```
ORDER BY DNAME DESC, LNAME ASC, FNAME, ASC;
```

Tables as Sets in SQL

SQL treats table as a **multiset,** which means duplicate tuples are OK

SQL does not delete duplicate because Duplicate elimination is an expensive operation (sort and delete) user may be interested in the result of a query in case of aggregate function, we do not want to eliminate duplicates

To eliminate duplicate, use DISTINCT

examples

Q11: Retrieve the salary of every employee , and (Q!2) all distinct salary values

Q11: SELECT ALL SALARY

FROM EMPLOYEE

Q12: SELECT DISTINCT SALARY

FROM EMPLOYEE

3.3 More Complex SQL Queries

Complex SQL queries can be formulated by composing nested SELECT/FROM/WHERE clauses within the WHEREclause of another query

Example: Q4: Make a list of Project numbers for projects that involve an employee whose last name is _Smith‘, either as a worker or as a manger of the department that controls the project

```
Q4      SELECT DISTINCT PNUMBER
        FROM PROJECT
       WHERE PNUMBER IN (SELECT PNUMBER
                         FROM PROJECT, DEPARTMENT, EMPLOYEE
                         WHERE DNUM=DNUMBER AND MGRSSN=SSN AND
LNAME=_Smith‘
 OR    PNUMBER IN (SELECT PNO
                   FROM WORKS_ON, EMPLOYEE
                   WHERE ESSN=SSN AND LNAME=_Smith‘)
```

IN operator and set of unioncompatible tuples

Example:

```
SELECT DISTINCT ESSN
FROM WORKS_ON
WHERE (PNO, HOURS) IN (SELECT PNO, HOURS
                       FROM   WORKS_ON
                       WHERE SSN=_123456789‘
```

ANY, SOME and >, <=,<>,etc.

The keyword ALL

In addition to the IN operator, a number of other comparison operators can be used to compare a single value v to a set of multiset V.

ALL V returns TRUE if v is greater than all the value in the set

Select the name of employees whose salary is greater than the salary of all the employees in department 5

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

Ambiguity in nested query

```
SELECT E.FNAME, E.LNAME
FROM EMPLOYEE AS E
WHERE E.SSN IN (SELECT ESSN
                FROM DEPENDENT
                 WHERE  ESSN=E.SSN  AND  E.FNAM=DEPENDENT_NAME   AND
SEX=E.SEX
```

Correlated Nested Query

Whenever a condition in the WHEREclause of a nested query references some attributes 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.

In general, any nested query involving the = or comparison operator IN can always be rewritten as a single block query

```
SELECT E.FNAME, E.LNAME

FROM EMPLOYEE E, DEPENDENT D

WHERE E.SSN=D.ESSN AND E.SEX=D.SEX AND E.FNAME =D.DEPENDENT=NAME
```

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)

In Q12, the nested query has a different result for each tuple in the outer query.

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).

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))

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 AND UNIQUE FUNCTIONS IN SQL

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.

Query 12: Retrieve the name of each employee who has a dependent with the same first name as the employee.

```
SELECT E.FNAME, E.LNAME
FROM EMPLOYEE E
WHERE EXISTS (SELECT *
                    FROM DEPENDENT
WHERE E.SSN=ESSN AND SEX=E.SEX AND E.FNAME=DEPENDENT_NAME
```

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 AND NULLS IN SQL

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

Retrieve SSNs of all employees who work on project number 1,2,3

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

Null example

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 .

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

Retrieve the name and address of every employee who works for _Search‘ department

```
SELECT FNAME, LNAME, ADDRESS
FROM (EMPLOYEE JOIN DEPARTMENT ON DNO=DNUMBER)
WHERE DNAME=_Search‘
```

Aggregate Functions

Include COUNT, SUM, MAX, MIN, and AVG

Query 15: Find the sum of the salaries of all employees the _Research‘ dept, and the max salary, the min salary, and average:

```
SELECT SUM(SALARY), MAX(SALARY), MIN(SALARY) AVG(SALARY)

FROM EMPLOYEE

WHERE DNO=FNUMBER AND DNAME=_RSEARCH‘
```

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'
```

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'
```

Example of 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 BYclause for specifying the grouping attributes, which must also appear in the SELECTclause

For each project, select the project number, the project name, and the number of employees who work on that projet

```
SELECT PNUMBER, PNAME, COUNT(*)

FROM PROJECT, WORKS_ON

WHERE PNUMBER=PNO

GROUP BY PNUMBER, PNAME
```

In Q20, the EMPLOYEE tuples are divided into groupseach 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 SELECTclause 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 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 HAVINGCLAUSE:

Sometimes we want to retrieve the values of these functions for only those groups that satisfy certain conditions. The HAVINGclause is used for specifying a selection condition on groups (rather than on individual tuples)

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
```

SQL The Relational Database Standard

3.1 Update Statements in SQL

The Insert Command

INSERT INTO EMPLOYEE

VALUES (_Richard‘,‘K‘,‘Marini‘,653298653‘,‘30dec52‘,98 Oak Forest, Katy, TX‘,‘M‘,37000,‘987654321‘,4)

More on Insert

Use explicit attribute names:

INSERT INTO EMPLOYEE (FNAME, LNAME,SSN)

VALUES (_Richard‘,‘Marini‘, _653298653‘

The DELECT Command

DELETE FROM EMPLOYEE

WHERE LNAME=_Brown‘

The UPDATE Command

Used to modify values of one or more selected tuples

Change the location and controlling department number of project number 10 to _Bellaire‘ and 5 respectively

UPDATE PROJECT

SET PLOCATION = _Bellaire‘, DNUM=5

Where PNUMBER=10;

3.2 Views in SQL

A view refers to a single table that is derived from other tables

```
CREATE VIEW WORKS_ON1
AS SELECT FNAME, LNAME, PNAME, HOURS
```

FROM EMPLOYEE, PROJECT, WORKS_ON WHERE SSN=ESSN AND

PNO=PNUMBER More on View

CREATE VIEW DEPT_INFO(DEPT_NAME, NO_OF_EMPLS, TOTAL_SAL)

AS SELECT DNAME, COUNT(*), SUM(SALARY)

FROM DEPARTMENT, EMPLOYEE

WHERE DNUMBER=DNO

GROUP BY DNAME

More on view

Treat WORKS_ON1 like a base table as follows

SELECT FNAME, LNAME

FROM WORKS_ON1

WHERE PNMAE=_PROJECTX‘

Main advantage of view:

Simplify the specification of commonly used queries

More on View

A View is always up to date;

A view is realized at the time we specify(or execute) a query on the view

DROP VIEW WORKS_ON1

Updating of Views

Updating the views can be complicated and ambiguous

In general, an update on a view on defined on a single table w/o any aggregate functions can be mapped to an update on the base table

More on Views

We can make the following observations:

A view with a single defining table is updatable if we view contain PK or CK of the base table

View on multiple tables using joins are not updatable

View defined using grouping/aggregate are not updatable

Specifying General Constraints

Users can specify certain constraints such as semantics constraints

CREATE ASSERTION SALARY_CONSTRAINT

CHECK (NOT EXISTS (SELECT * FROM EMPLOYEE E, EMPLOYEE M, DEPARTMENT D

WHERE E.SALARY > M. SALARY **AND** E.DNO=D.NUMBER **AND** D.MGRSSN=M.SSN))

3.3 Additional features

Granting and revoking privileges

Embedding SQL statements in a general purpose languages (C, C++, COBOL, PASCAL)

SQL can also be used in conjunction with a general purpose programming language, such as PASCAL, COBOL, or PL/I. The programming language is called the host language. The embedded SQL statement is distinguished from programming language statements by prefixing it with a special character or command so that a preprocessor can extract the SQL statements. In PL/I

the keywords EXEC SQL precede any SQL statement. In some implementations, SQL statements are passed as parameters in procedure calls. We will use PASCAL as the host programming language, and a "$" sign to identify SQL statements in the program. Within an embedded SQL command, we may refer to program variables, which are prefixed by a "%" sign. The programmer should declare program variables to match the data types of the database attributes that the program will process.These program variables may or may not have names that are identical to their corresponding attributes.

Example: Write a program segment (loop) that reads a social security number and prints out some information from the corresponding EMPLOYEE tuple.

```
        readln(SOC_SEC_NUM);
        $SELECT FNAME, MINIT, LNAME, SSN,
BDATE, ADDRESS, SALARY
          INTO %E.FNAME, %E.MINIT, %E.LNAME, %E.SSN,
                    %E.BDATE, %E.ADDRESS, %E.SALARY
          FROM EMPLOYEE
          WHERE SSN=%SOC_SEC_NUM ;
        writeln( E.FNAME, E.MINIT, E.LNAME,
E.SSN, E.BDATE, E.ADDRESS, E.SALARY);
        writeln('more social security numbers (Y or N)? ');
        readln(LOOP)
        end;
```

In E1, a single tuple is selected by the embedded SQL query; that is why we are able to assign its attribute values directly to program variables. In general, an SQL query can retrieve many tuples. The concept of a cursor is used to allow tupleatatime processing by the PASCAL programCURSORS: We can think of a cursor as a pointer that points to a single tuple (row) from the result of a query.The cursor is declared when the SQL query command is specified. A subsequent OPEN cursor command fetches the query result and sets the cursor to a position before the first row in the result of the query; this becomes the current row for the cursor. Subsequent FETCH commands in the program advance the cursor to the next row and copy its attribute values into PASCAL program variables specified in the FETCH command. An implicit variable SQLCODE communicates to the program the status of SQL embedded commands. An SQLCODE of 0 (zero) indicates successful execution. Different codes are returned to indicate exceptions and errors. A special END_OF_CURSOR code is used to terminate a loop over the tuples in a query result. A CLOSE cursor command is issued to indicate that we are done with
the result of the query When a cursor is defined for rows that are to be updated the clause FOR UPDATE OF must be in the cursor declaration, and a list of the names of any attributes that will be updated follows.The condition WHERE CURRENT OF cursor specifies that the current tuple is the one to be updated (or deleted)

Example: Write a program segment that reads (inputs) a department name, then lists the names of employees who work in that department, one at a time. The program reads a raise amount for each employee and updates the employee's salary by that amount.

E2:
```
writeln('enter the department name:'); readln(DNAME);
$SELECT DNUMBER INTO %DNUMBER
  FROM DEPARTMENT
  WHERE DNAME=%DNAME;
$DECLARE EMP CURSOR FOR
                                SELECT SSN, FNAME, MINIT, LNAME, SALARY
                                FROM EMPLOYEE
                                WHERE DNO=%DNUMBER
  FOR UPDATE OF SALARY;
$OPEN EMP;
$FETCH EMP INTO %E.SSN, %E.FNAME, %E.MINIT,
                                                %E.LNAME, %E.SAL;
```

```
while SQLCODE = 0 do
                    begin
writeln('employee name: ', E.FNAME, E.MINIT, E.LNAME);
writeln('enter raise amount: '); readln(RAISE);
$UPDATE EMPLOYEE SET SALARY = SALARY +
        %RAISE WHERE CURRENT OF EMP;
$FETCH EMP INTO %E.SSN, %E.FNAME, %E.MINIT,
                                              %E.LNAME, %E.SAL;
                    end;
        $CLOSE CURSOR EMP;
```

3.4 Database Programming

- Objective:
  - To access a database from an application program (as opposed to interactive interfaces)
- Why?
  - An interactive interface is convenient but not sufficient
    - A majority of database operations are made thru application programs (increasingly thru web applications)
- Embedded commands:
  - Database commands are embedded in a general-purpose programming language
- Library of database functions:
  - Available to the host language for database calls; known as an *API*
    - *API* standards for Application Program Interface
- A brand new, full-fledged language
  - Minimizes impedance mismatch

Impedance Mismatch

- Incompatibilities between a host programming language and the database model, e.g.,
  - type mismatch and incompatibilities; requires a new binding for each language
  - set vs. record-at-a-time processing
    - need special iterators to loop over query results and manipulate individual values
- Client program *opens a connection* to the database server
- Client program *submits queries to and/or updates* the database
- When database access is no longer needed, client program *closes (terminates) the connection*

3.5 Embedded SQL

- Most SQL statements can be embedded in a general-purpose *host* programming language such as COBOL, C, Java

- An embedded SQL statement is distinguished from the host language statements by enclosing it between EXEC SQL or EXEC SQL BEGIN and a matching END-EXEC or EXEC SQL END (or semicolon)
  - Syntax may vary with language
  - *Shared variables* (used in both languages) usually prefixed with a colon (:) in SQL
- Variables inside DECLARE are shared and can appear (while prefixed by a colon) in SQL statements
- SQLCODE is used to communicate errors/exceptions between the database and the program

```
int loop;

EXEC SQL BEGIN DECLARE SECTION;

        varchar dname[16], fname[16], …;

        char ssn[10], bdate[11], …;

        int dno, dnumber, SQLCODE, …;

EXEC SQL END DECLARE SECTION;
```

- Connection (multiple connections are possible but only one is active) CONNECT TO server-name AS connection-name

  AUTHORIZATION user-account-info;

- Change from an active connection to another one SET CONNECTION connection-name;
- Disconnection

  DISCONNECT connection-name;

```
loop = 1;

while (loop) {

        prompt (—Enter SSN: —, ssn);

        EXEC SQL
```

```
select FNAME, LNAME, ADDRESS, SALARY
into :fname, :lname, :address, :salary
from EMPLOYEE where SSN == :ssn;
if (SQLCODE == 0) printf(fname, …);
else printf(—SSN does not exist: —, ssn);
prompt(—More SSN? (1=yes, 0=no): —, loop);
END-EXEC
```

- }A cursor (iterator) is needed to process multiple tuples
- FETCH commands move the cursor to the *next* tuple
- CLOSE CURSOR indicates that the processing of query results has been completed
- Objective:

3.6 Dynamic SQL

Composing and executing new (not previously compiled) SQL statements at run-time

- a program accepts SQL statements from the keyboard at run-time
- a point-and-click operation translates to certain SQL query
- Dynamic update is relatively simple; dynamic query can be complex
  - because the type and number of retrieved attributes are unknown at compile time

```
EXEC SQL BEGIN DECLARE SECTION;
varchar sqlupdatestring[256];
EXEC SQL END DECLARE SECTION;
…prompt (—Enter update command:—, sqlupdatestring);
EXEC SQL PREPARE sqlcommand FROM :sqlupdatestring;
```

- EXEC SQLSQLJ: a standard for embedding SQL in Java
- An SQLJ translator converts SQL statements into Java
  - These are executed thru the *JDBC* interface
- Certain classes have to be imported
  - E.g., java.sql

EXECUTE sqlcommand;

- *Environment record*:
  - Keeps track of database connections
- *Connection record*:
  - Keep tracks of info needed for a particular connection
- *Statement record*:
  - Keeps track of info needed for one SQL statement
- *Description record*:
  - Keeps track of tuples
- Load SQL/CLI libraries
- Declare record handle variables for the above components (called: SQLHSTMT, SQLHDBC, SQLHENV, SQLHDEC)
- Set up an environment record using SQLAllocHandle
- Set up a connection record using SQLAllocHandle
- Set up a statement record using SQLAllocHandle
- Prepare a statement using SQL/CLI function SQLPrepare
- Bound parameters to program variables
- Execute SQL statement via SQLExecute
- Bound query columns to a C variable via SQLBindCol
- Use SQLFetch to retrieve column values into C variables

3.7 Database stored procedures and SQL/PSM

- Persistent procedures/functions (modules) are stored locally and executed by the database server
  - As opposed to execution by clients
- Advantages:
  - If the procedure is needed by many applications, it can be invoked by any of them (thus reduce duplications)
  - Execution by the server reduces communication costs
  - Enhance the modeling power of views
- Disadvantages:
  - Every DBMS has its own syntax and this can make the system less portable
- A stored procedure

  CREATE PROCEDURE procedure-name

  (params) local-declarations

  procedure-body;
- A stored function

CREATE FUNCTION fun-name (params) RETRUNS return-type

local-declarations

function-body;

- Calling a procedure or function

  CALL procedure-name/fun-name (arguments);

- SQL/PSM:
  - Part of the SQL standard for writing persistent stored modules
- SQL + stored procedures/functions + additional programming constructs
  - E.g., branching and looping statements
  - Enhance the power of SQL

CREATE FUNCTION DEPT_SIZE (IN deptno INTEGER)

RETURNS VARCHAR[7]

DECLARE TOT_EMPS INTEGER;

SELECT COUNT (*) INTO TOT_EMPS

FROM SELECT EMPLOYEE WHERE DNO = deptno;

IF TOT_EMPS > 100 THEN RETURN ―HUGE‖

ELSEIF TOT_EMPS > 50 THEN RETURN ―LARGE‖

ELSEIF TOT_EMPS > 30 THEN RETURN ―MEDIUM‖

ELSE RETURN ―SMALL‖

ENDIF;

Questions

1. List the approaches to DB Programming. Main issues involved in DB Programming?
2. What is Impedance Mismatch problem? Which of the three programming approaches minimizes this problem
3. How are Triggers and assertions defined in SQL?Explain
4. A explain the syntax of a SELECT statement in SQL.write the SQL query for the following relation algebra expression.
5. Explain the drop command with an example
6. How is a view created and dropped? What problems are associated with updating of views?
7. What is embedded SQL? With an example explain how would you Connect to a database, fetch records and display. Also explain the concept of stored procedure in brief.
8. Explain insert, delete and update statements in SQL with example.
9. Write a note on aggregate functions in SQL with examples.