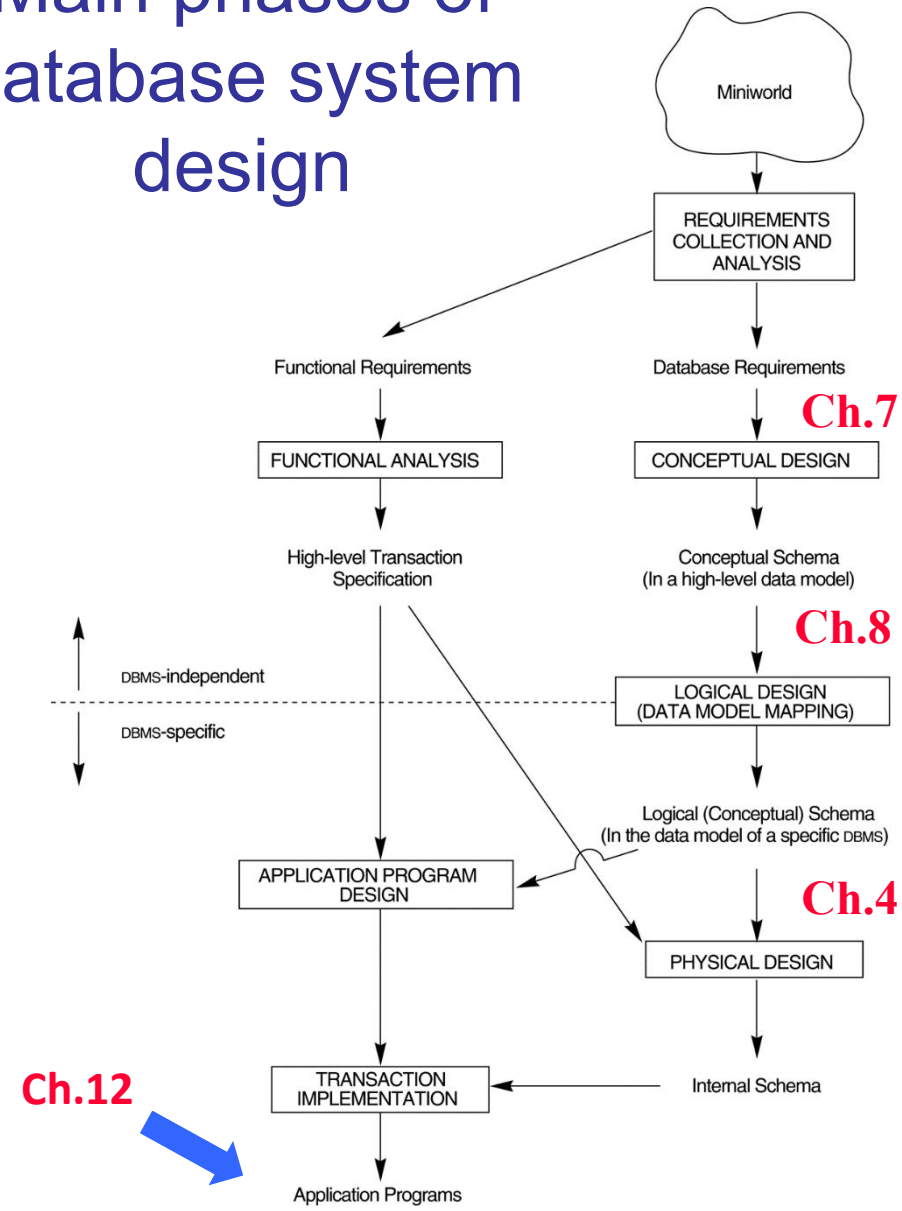
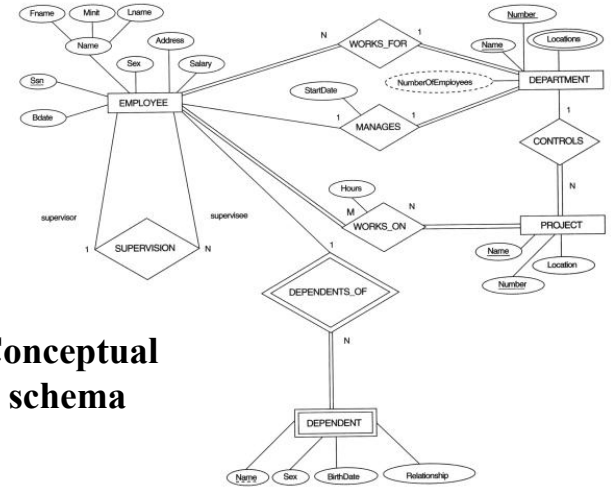
Chapter 12
SQL Application Programming Techniques Using C and Java

Main phases of database system design



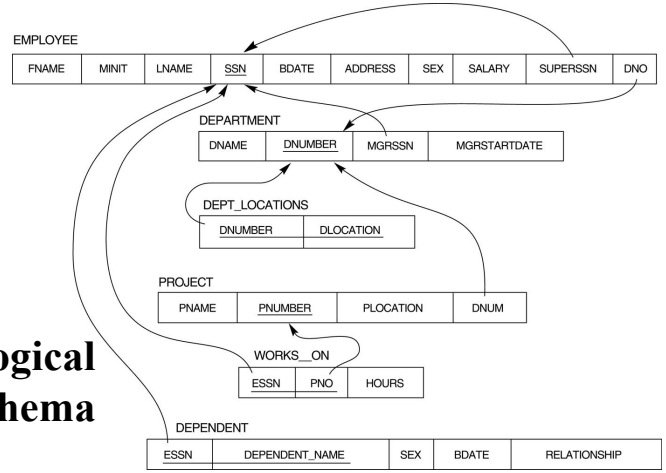
Data Requirements

Conceptual design ↓ (ER Model)



Conceptual schema

Logical design ↓ (Relational Model)



Logical schema

Physical design ↓

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

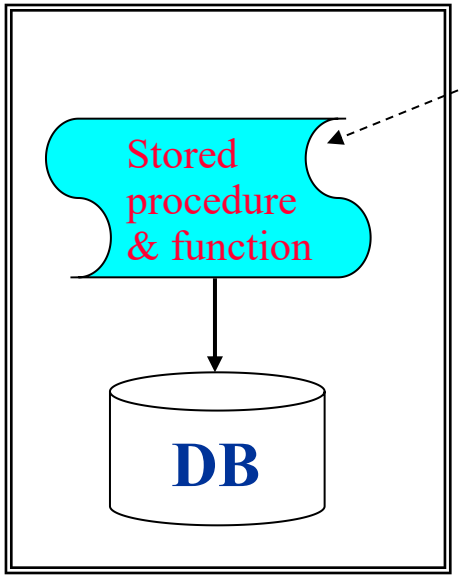
Construction and Operation

Ch. 12: Database programming
- embedded/dynamic SQL
- function call
- stored procedure and function

construct

Define DB
Load data

(SQL)



DB System

AP
…

SQL
Interactive

SELECT …
FROM …
WHERE …

Ch. 4: SQL
- **Data Definition Language**
  > base table, view

Ch. 5: SQL
- **Data Manipulation Language**
  > Query: SELECT
  > Update: INSERT, DELETE, UPDATE

Objectives

To access a database from an application program (as opposed to interactive interfaces)

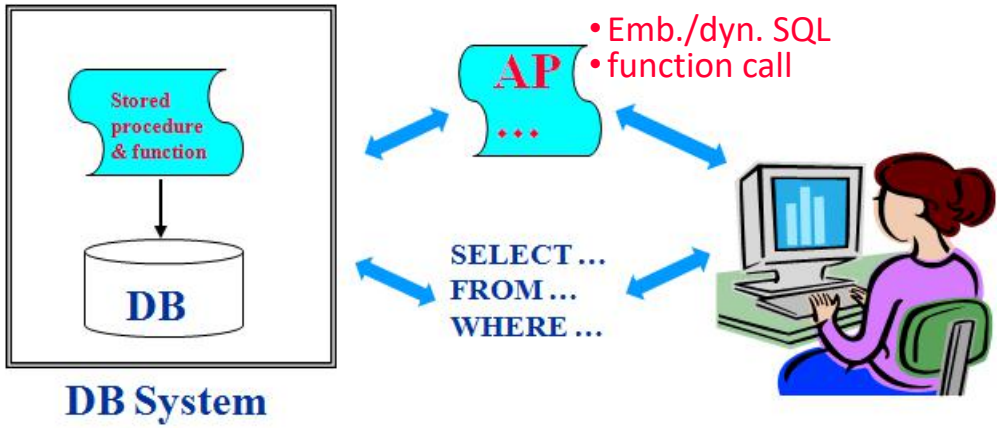
12.1 Database Programming

12.2 Embedded/dynamic SQL and SQLJ

12.3 Functions Calls, SQL/CLI and JDBC

12.4 Stored Procedures, SQL/PSM

12.5 Comparing the Three Approaches

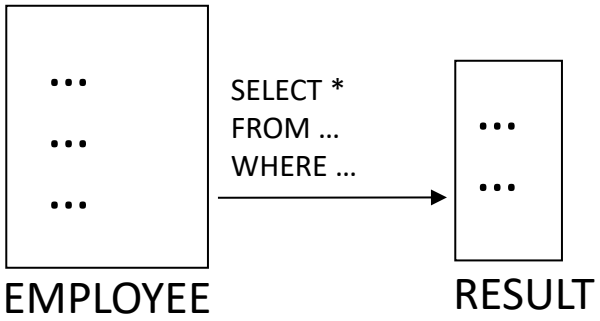


Database Programming Approaches

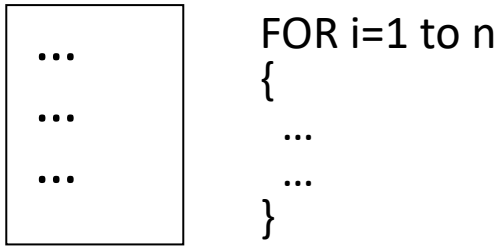
- Embedded SQL
  - SQL statements are embedded in a general-purpose programming language, e.g., C, Java, Pascal
- Call Level Interface (CLI)
  - library of database functions
  - available to the host language for database calls; known as an *API*
- A brand new, full-fledged language
  - stored procedures and functions
  - minimizes impedance mismatch

Impedance Mismatch

- Incompatibilities between a host programming language and the database model, e.g.,
  - type mismatch and incompatibilities
    - DATE, TIME, TIMESTAMP, etc.
    - requires a new binding for each language
  - Set-at-a-time vs. record-at-a-time processing
    - need special iterators to loop over query results and manipulate individual values



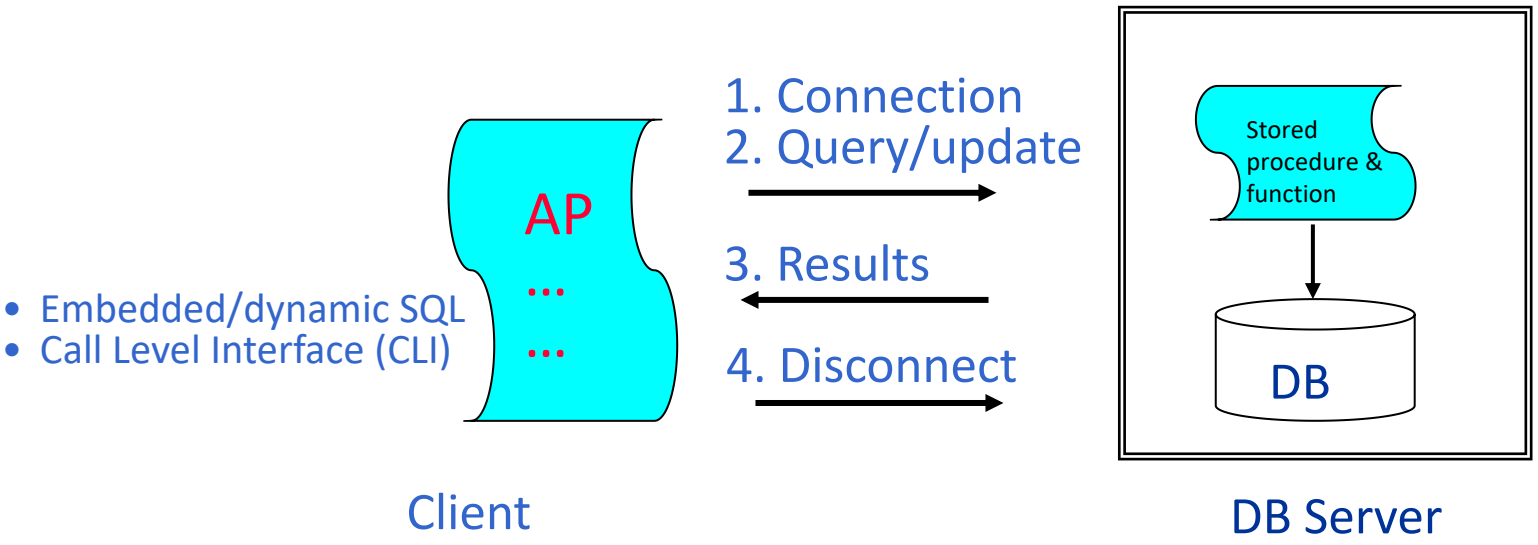
SQL



C, Java

Steps in Database Programming

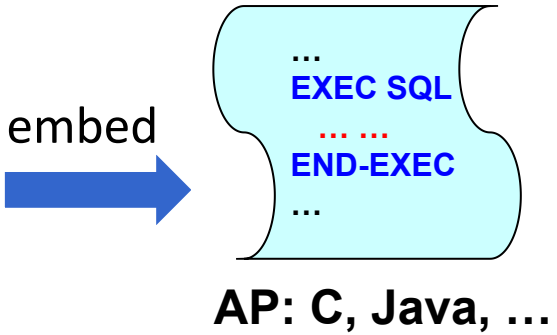
1. Client program opens a connection to the database server
2. Client program submits queries to and/or updates the database
3. When database access is no longer needed, client program terminates the connection



Embedded SQL

- Most SQL statements can be embedded in a general-purpose host programming language such as C, Java, Pascal
- An embedded SQL statement is distinguished from the host language statements by **EXEC SQL** and a matching **END-EXEC** (or ";" semicolon)
  - **shared variables** (used in both languages) usually prefixed with a colon (:) in SQL

```
EXEC SQL
   select FNAME, LNAME, ADDRESS, SALARY
   into :fname, :lname, :address, :salary
   from EMPLOYEE where SSN = :ssn
END-EXEC
```



Example: Variable Declaration in Language C

- 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     lname[16], fname[16], …;
        char        ssn[10], bdate[40], …;
        int         dno, salary;
        int         SQLCODE, …;
EXEC SQL END DECLARE SECTION;
```

```
…
EXEC SQL
   select FNAME, LNAME, ADDRESS, SALARY
   into :fname, :lname, :address, :salary
   from EMPLOYEE where SSN == :ssn;
…
```

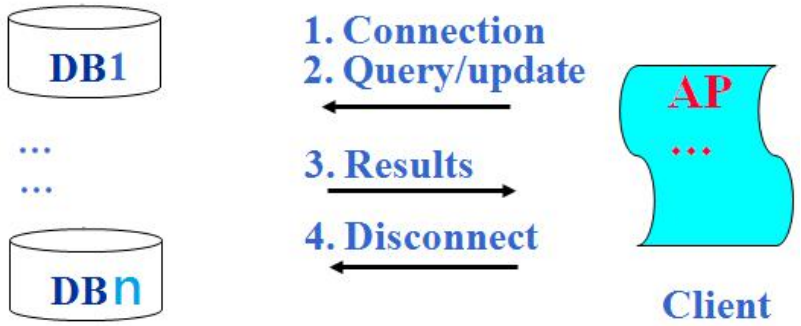
SQL Commands for Connecting to a Database

- 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;



Example: Embedded SQL in C

```
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, lname, ...);
    else printf("SSN does not exist: ", ssn);
    prompt("More SSN? (1=yes, 0=no): ", loop);
}
```

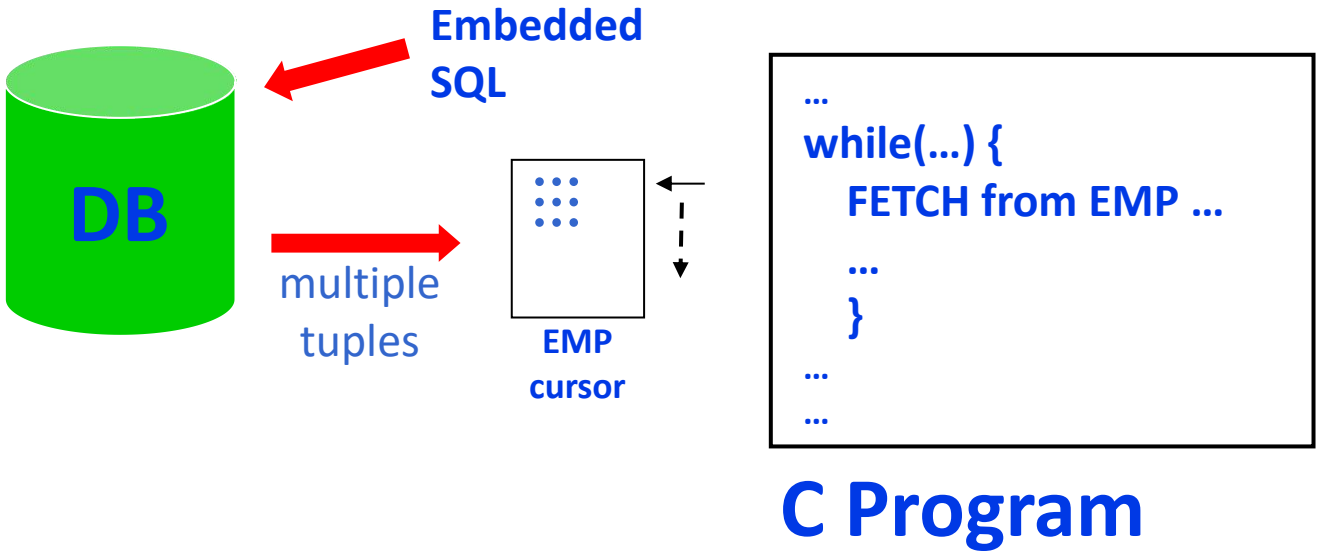
← Shared variables

← Check success or not

← Shared variables

Cursor for Multiple Tuples

- 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



Program segment E2, a C program segment that uses cursors with embedded SQL for update purposes.

```
//Program Segment E2:
0)   prompt("Enter the Department Name: ", dname) ;
1)   EXEC SQL
2)      select DNUMBER into :dnumber
3)      from DEPARTMENT where DNAME = :dname ;
4)   EXEC SQL DECLARE EMP CURSOR FOR
5)      select SSN, FNAME, MINIT, LNAME, SALARY
6)      from EMPLOYEE where DNO = :dnumber
7)      FOR UPDATE OF SALARY ;
8)   EXEC SQL OPEN EMP ;
9)   EXEC SQL FETCH from EMP into :ssn, :fname, :minit, :lname, :salary ;
10)  while (SQLCODE == 0) {
11)     printf("Employee name is:", fname, minit, lname)
12)     prompt("Enter the raise amount: ", raise) ;
13)     EXEC SQL
14)        update EMPLOYEE
15)        set SALARY = SALARY + :raise
16)        where CURRENT OF EMP ;
17)     EXEC SQL FETCH from EMP into :ssn, :fname, :minit, :lname, :salary ;
18)     }
19)  EXEC SQL CLOSE EMP ;
```

// declare a cursor EMP

Fetch

Fetch

close

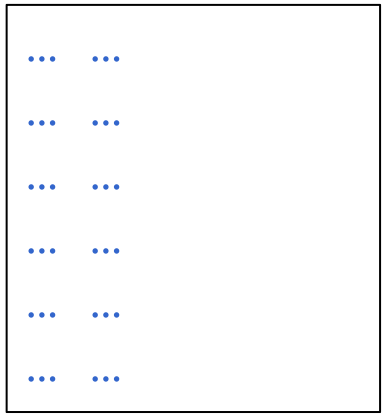
Embedded SQL in Java

- SQLJ
  - a standard for embedding SQL in Java
- An SQLJ translator converts SQL statements into Java (to be executed thru the *JDBC* interface)
- Certain classes, e.g., java.sql have to be imported

```
//Program Segment E2:
0)   prompt("Enter the Department Name: ", dname) ;
1)   EXEC SQL
2)      select DNUMBER into :dnumber
3)      from DEPARTMENT where DNAME = :dname ;
4)   EXEC SQL DECLARE EMP CURSOR FOR
5)      select SSN, FNAME, MINIT, LNAME, SALARY
6)      from EMPLOYEE where DNO = :dnumber
7)      FOR UPDATE OF SALARY ;
8)   EXEC SQL OPEN EMP ;
9)   EXEC SQL FETCH from EMP into :ssn, :fname, :minit, :lname, :salary ;
10)  while (SQLCODE == 0) {
11)     printf("Employee name is:", fname, minit, lname)
12)     prompt("Enter the raise amount: ", raise) ;
13)     EXEC SQL
14)        update EMPLOYEE
15)        set SALARY = SALARY + :raise
16)        where CURRENT OF EMP ;
17)     EXEC SQL FETCH from EMP into :ssn, :fname, :minit, :lname, :salary ;
18)     }
19)  EXEC SQL CLOSE EMP ;
```

Java + Embedded SQL

SQLJ translator



Java

Importing classes needed for including SQLJ in JAVA programs in ORACLE, and establishing a connection and default context.

```
1)   import java.sql.* ;
2)   import java.io.* ;
3)   import sqlj.runtime.* ;
4)   import sqlj.runtime.ref.* ;
5)   import oracle.sqlj.runtime.* ;

     ...
6)   DefaultContext cntxt =
7)      oracle.getConnection("<url name>", "<user name>", "<password>", true) ;
8)   DefaultContext.setDefaultContext(cntxt) ;

     ...
```

JAVA program variables used in SQLJ examples J1 and J2.

```
1)   string dname, ssn , fname, fn, lname, ln, bdate, address  ;
2)   char sex, minit, mi ;
3)   double salary, sal ;
4)   integer dno, dnumber ;
```

Program segment J1:
A JAVA program segment with SQLJ.

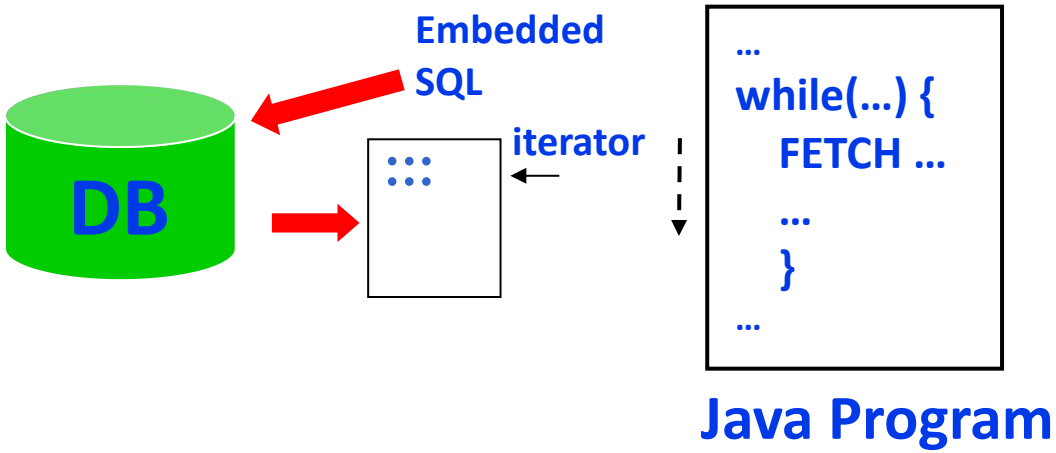
```
//Program Segment J1:
1)   ssn = readEntry("Enter a Social Security Number: ") ;
2)   try {
3)      #sql{select FNAME, MINIT, LNAME, ADDRESS, SALARY
4)         into :fname, :minit, :lname, :address, :salary
5)         from EMPLOYEE where SSN = :ssn} ;
6)   } catch (SQLException se) {
7)      System.out.println("Social Security Number does not exist: " + ssn) ;
8)      Return ;
9)      }
10)  System.out.println(fname + " " + minit + " " + lname + " " + address + " " +
     salary)
```

Embedded SQL

Multiple Tuples in SQLJ

- SQLJ supports two types of iterators:
  - ***named iterator***: associated with a query result
  - ***positional iterator***: lists only attribute types in a query result
- A **FETCH** operation retrieves the next tuple in a query result:

fetch iterator-variable **into** program-variable



A JAVA program segment that uses a named iterator to print employee information in a particular department.

```
//Program Segment J2A:
0)   dname = readEntry("Enter the Department Name: ") ;
1)   try {
2)      #sql{select DNUMBER into :dnumber
3)          from DEPARTMENT where DNAME = :dname} ;
4)   }  catch (SQLException se) {
5)      System.out.println("Department does not exist: " + dname) ;
6)      Return ;
7)      }
8)   System.out.printline("Employee information for Department: " + dname) ;
9)   #sql iterator Emp(String ssn, String fname, String minit, String lname,
     double salary) ;
10)  Emp e = null ;
11)  #sql e = {select ssn, fname, minit, lname, salary
12)             from EMPLOYEE where DNO = :dnumber} ;
13)  while (e.next()) {
14)     System.out.printline(e.ssn + " " + e.fname + " " + e.minit + " " +
           e.lname + " " + e.salary) ;
15)     } ;
16)  e.close() ;
```

A named iterator

Retrieve from DB

Process results

A JAVA program segment that uses a positional iterator to print employee information in a particular department.

```
//Program Segment J2B:
0)   dname = readEntry("Enter the Department Name: ") ;
1)   try {
2)      #sql{select DNUMBER into :dnumber
3)          from DEPARTMENT where DNAME = :dname} ;
4)   } catch (SQLException se) {
5)      System.out.println("Department does not exist: " + dname) ;
6)      Return ;
7)      }
8)   System.out.printline("Employee information for Department: " + dname) ;
9)   #sql iterator Emppos(String, String, String, String, double) ;
10)  Emppos e = null ;
11)  #sql e ={select ssn, fname, minit, lname, salary
12)     from EMPLOYEE where DNO = :dnumber} ;
13)  #sql {fetch :e into :ssn, :fn, :mi, :ln, :sal} ;
14)  while (!e.endFetch()) {
15)     System.out.printline(ssn + " " + fn + " " + mi + " " + ln + " " + sal) ;
16)     #sql {fetch :e into :ssn, :fn, :mi, :ln, :sal} ;
17)     } ;
18)  e.close() ;
```

positional iterator

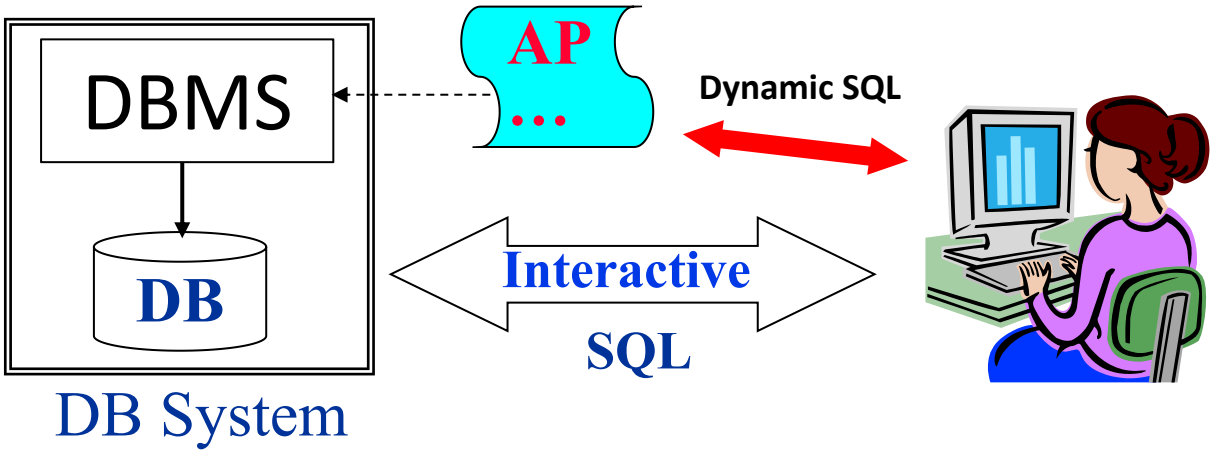
Retrieve from DB

close

Dynamic SQL

- Objective:

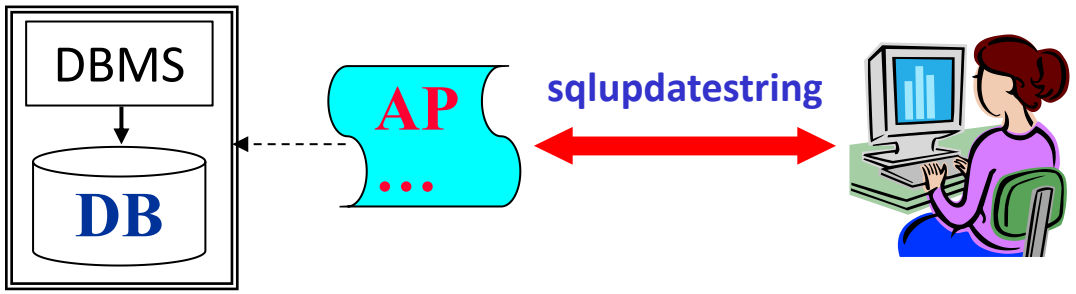
  executing new (not previously compiled) SQL statements at run-time
  - a program accepts SQL statements from the keyboard at run-time
  - a point-and-click operation translates to certain SQL query
- Dynamic update is relatively simple; dynamic query can be complex
  - because the type and number of retrieved attributes are unknown at compile time



Specifying Queries at Runtime

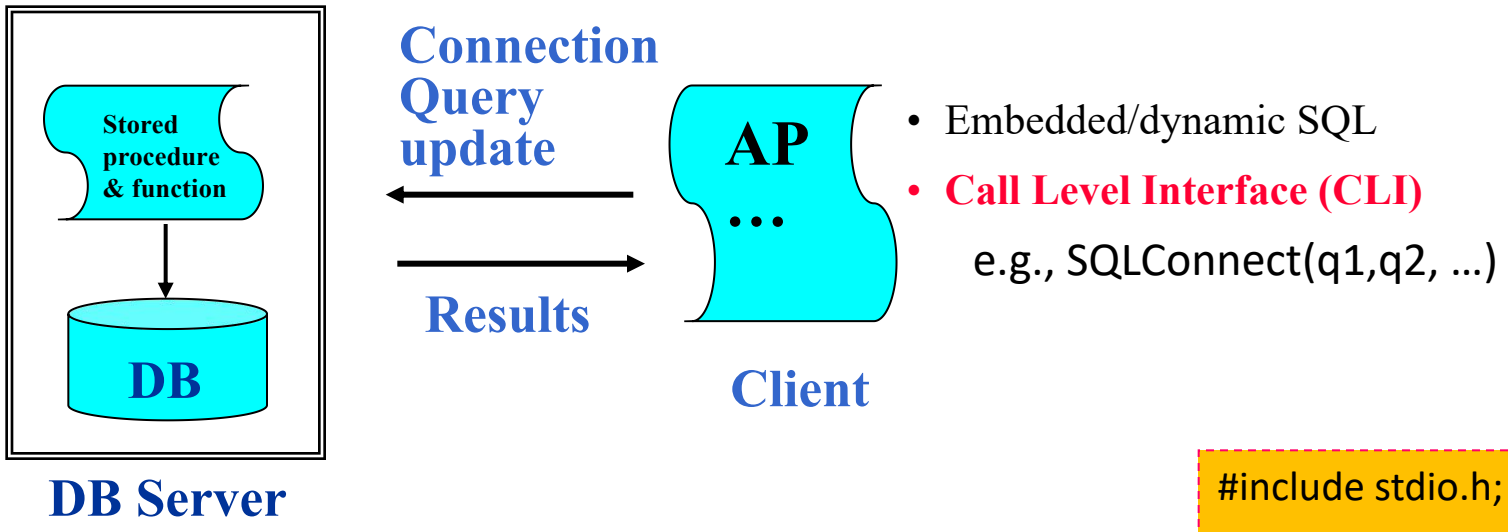
Program segment E3:
A C program segment that uses dynamic SQL for updating table.

```
0.  EXEC SQL BEGIN DECLARE SECTION;
1.      varchar sqlupdatestring[256];
2.  EXEC SQL END DECLARE SECTION;
…   …
7.  prompt ("Enter update command:", sqlupdatestring);
8.  EXEC SQL PREPARE sqlcommand FROM :sqlupdatestring;
9.  EXEC SQL EXECUTE sqlcommand;
```



Database Programming with Functional Calls

- Embedded SQL provides static database programming
- API: dynamic database programming with a library of functions
  - advantage: no preprocessor needed (thus more flexible)
  - drawback: SQL syntax checks to be done at run-time




```
#include stdio.h;
…
scanf("No.: %5d", MyNum);
```

SQL/CLI Call Level Interface

- A part of the SQL standard
- Provides easy access to several databases within the same program
- Certain libraries (e.g., **sqlcli.h** for C) have to be installed and available
- SQL statements are dynamically created and passed as string parameters in the calls

No preprocessor needed

Call Level Interface:

```
#include sqlcli.h;
 …

SQLConnect(p1,p2, …);
 SQLExecute(stmt);
 …
```

C program

Embedded SQL:

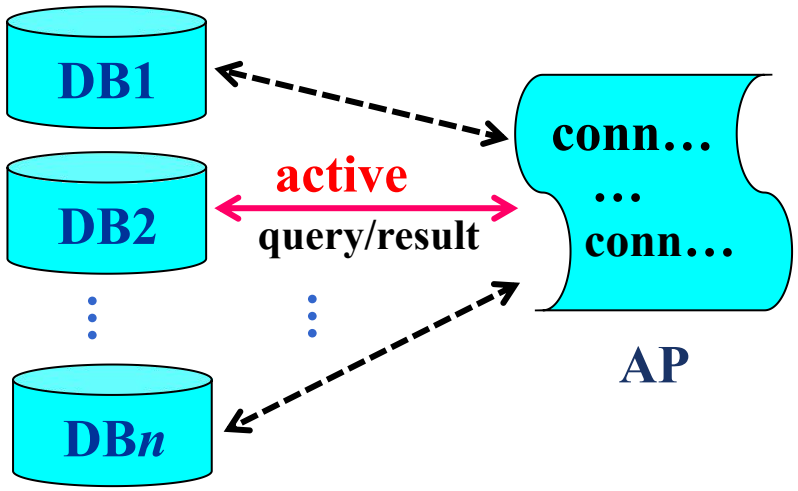
```
…
EXEC SQL
    SELECT SSN …
END-EXEC
…
```

C program

Need preprocessor

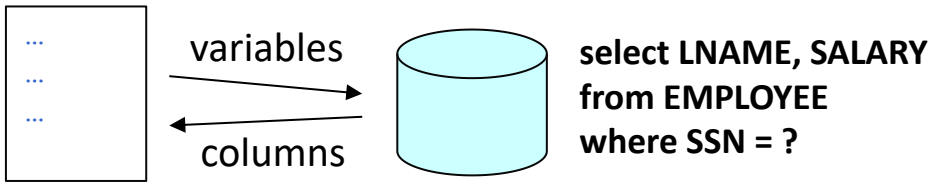
Components of SQL/CLI

- 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



Steps in C and SQL/CLI Programming

1. Load SQL/CLI libraries
2. Declare record handle variables for the above components (called: SQLHENV, SQLHDBC, SQLHSTMT, SQLHDEC)
3. Set up an **environment record** using **SQLAllocHandle**
4. Set up a **connection record** using **SQLAllocHandle**
5. Set up a **statement record** using **SQLAllocHandle**
6. **Prepare a statement** using SQL/CLI function **SQLPrepare**
7. **Bind parameters** to program variables via **SQLBindParameter**
8. **Execute SQL** statement via **SQLExecute**
9. **Bind columns** in a query to a C variable via **SQLBindCol**
10. **Use SQLFetch** to retrieve column values into C variables



Program segment CLI1:
A C program segment with SQL/CLI.

```
//Program CLI1:
0)   #include sqlcli.h ;
1)   void printSal() {
2)   SQLHSTMT stmt1 ;
3)   SQLHDBC con1 ;
4)   SQLHENV env1 ;
5)   SQLRETURN ret1, ret2, ret3, ret4 ;
6)   ret1 = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env1) ;
7)   if (!ret1) ret2 = SQLAllocHandle(SQL_HANDLE_DBC, env1, &con1) else exit ;
8)   if (!ret2) ret3 = SQLConnect(con1, "dbs", SQL_NTS, "js", SQL_NTS, "xyz", SQL_NTS)
else exit ;
9)   if (!ret3) ret4 = SQLAllocHandle(SQL_HANDLE_STMT, con1, &stmt1) else exit ;
10)  SQLPrepare(stmt1, "select LNAME, SALARY from EMPLOYEE where SSN = ?", SQL_NTS) ;
11)  prompt("Enter a Social Security Number: ", ssn) ;
12)  SQLBindParameter(stmt1, 1, SQL_CHAR, &ssn, 9, &fetchlen1) ;
13)  ret1 = SQLExecute(stmt1) ;
14)  if (!ret1) {
15)     SQLBindCol(stmt1, 1, SQL_CHAR, &lname, 15, &fetchlen1) ;
16)     SQLBindCol(stmt1, 2, SQL_FLOAT, &salary, 4, &fetchlen2) ;
17)     ret2 = SQLFetch(stmt1) ;
18)     if (!ret2) printf(ssn, lname, salary)
19)        else printf("Social Security Number does not exist: ", ssn) ;
20)     }
21)  }
```

declaration

Setup records

Prepare query

Bind parameters

Bind output

Process results

Program segment CLI2, a C program segment that uses SQL/CLI for a query with a collection of tuples in its result.

```
//Program Segment CLI2:
0)   #include sqlcli.h ;
1)   void printDepartmentEmps() {
2)   SQLHSTMT stmt1 ;
3)   SQLHDBC con1 ;
4)   SQLHENV env1 ;
5)   SQLRETURN ret1, ret2, ret3, ret4 ;
6)   ret1 = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env1) ;
7)   if (!ret1) ret2 = SQLAllocHandle(SQL_HANDLE_DBC, env1, &con1) else exit ;
8)   if (!ret2) ret3 = SQLConnect(con1, "dbs", SQL_NTS, "js", SQL_NTS, "xyz", SQL_NTS)
else exit ;
9)   if (!ret3) ret4 = SQLAllocHandle(SQL_HANDLE_STMT, con1, &stmt1) else exit ;
10)  SQLPrepare(stmt1, "select LNAME, SALARY from EMPLOYEE where DNO = ?", SQL_NTS) ;
11)  prompt("Enter the Department Number: ", dno) ;
12)  SQLBindParameter(stmt1, 1, SQL_INTEGER, &dno, 4, &fetchlen1) ;
13)  ret1 = SQLExecute(stmt1) ;
14)  if (!ret1) {
15)      SQLBindCol(stmt1, 1, SQL_CHAR, &lname, 15, &fetchlen1) ;
16)      SQLBindCol(stmt1, 2, SQL_FLOAT, &salary, 4, &fetchlen2) ;
17)      ret2 = SQLFetch(stmt1) ;
18)      while (!ret2) {
19)          printf(lname, salary) ;
20)          ret2 = SQLFetch(stmt1) ;
21)          }
22)      }
23)  }
```

declaration

Setup records

Prepare query

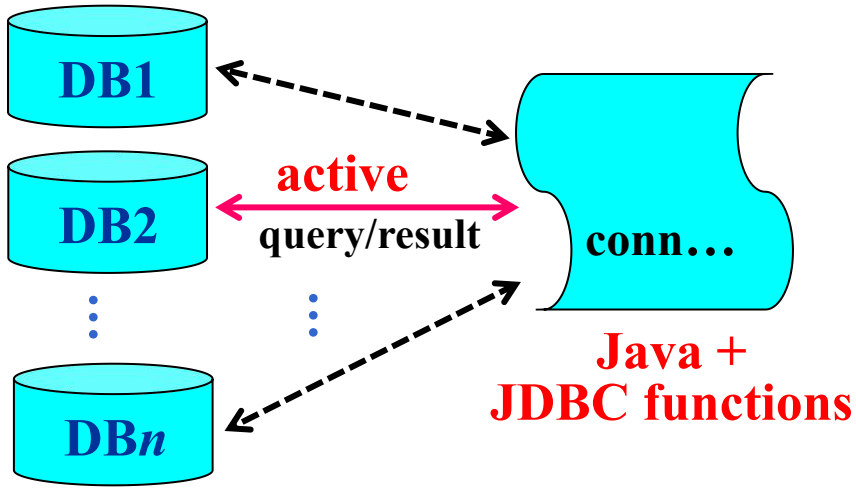
Bind parameters

Bind output

Iteratively process results

Java Database Connectivity

- JDBC: SQL connection function calls for Java programming
- A Java program with JDBC functions can access any relational DBMS that has a JDBC driver
- JDBC allows a program to connect to several databases (known as *data sources*)



Steps in JDBC Database Access

1. Import JDBC library **(java.sql.*)**
2. Load JDBC driver: **Class.forName("oracle.jdbc.driver.OracleDriver")**
3. Define appropriate variables
4. Create a **connect** object (via **getConnection**)
5. Create a **statement** object from the **Statement class**:
   - **(1) PreparedStatment**
   - **(2) CallableStatement**
6. Identify statement parameters (to be designated by question marks)
7. Bind **parameters** to **program variables**
8. Execute SQL statement (referenced by an object) via JDBC's **executeQuery**
9. Process query results (returned in an object of type **ResultSet**)
   - **ResultSet** is a 2-dimentional table

Program segment JDBC1:
A JAVA program segment with JDBC.

```
//Program JDBC1:
0)    import java.io.* ;
1)    import java.sql.*
      …
2)    class getEmpInfo {
3)       public static void main (String args []) throws SQLException, IOException {
4)       try { Class.forName("oracle.jdbc.driver.OracleDriver")
5)       } catch (ClassNotFoundException x) {
6)          System.out.println ("Driver could not be loaded") ;
7)          }
8)       String dbacct, passwrd, ssn, lname ;
9)       Double salary ;
10)      dbacct = readentry("Enter database account:") ;
11)      passwrd = readentry("Enter pasword:") ;
12)      Connection conn = DriverManager.getConnection
13)         ("jdbc:oracle:oci8:" + dbacct + "/" + passwrd) ;
14)      String stmt1 = "select LNAME, SALARY from EMPLOYEE where SSN = ?" ;
15)      PreparedStatement p = conn.prepareStatement(stmt1) ;
16)      ssn = readentry("Enter a Social Security Number: ") ;
17)      p.clearParameters() ;
18)      p.setString(1, ssn) ;
19)      ResultSet r = p.executeQuery() ;
20)      while (r.next()) {
21)         lname = r.getString(1) ;
22)         salary = r.getDouble(2) ;
23)         system.out.println(lname + salary) ;
24)      } }
25)   }
```

declaratio n

connect

Prepare query

Bind parameter

process query results

Program segment JDBC2, a JAVA program segment that uses JDBC for a query with a collection of tuples in its result.

```
//Program Segment JDBC2:
0)   import java.io.* ;
1)   import java.sql.*
     …
2)   class printDepartmentEmps {
3)      public static void main (String args []) throws SQLException, IOException {
4)      try { Class.forName("oracle.jdbc.driver.OracleDriver")
5)      } catch (ClassNotFoundException x) {
6)         System.out.println ("Driver could not be loaded") ;
7)         }
8)      String dbacct, passwrd, lname ;
9)      Double salary ;
10)     Integer dno ;
11)     dbacct = readentry("Enter database account:") ;
12)     passwrd = readentry("Enter pasword:") ;
13)     Connection conn = DriverManager.getConnection
14)        ("jdbc:oracle:oci8:" + dbacct + "/" + passwrd) ;
15)     dno = readentry("Enter a Department Number: ") ;
16)     String q = "select LNAME, SALARY from EMPLOYEE where DNO = " +
        dno.tostring() ;
17)     Statement s = conn.createStatement() ;
18)     ResultSet r = s.executeQuery(q) ;
19)     while (r.next()) {
20)        lname = r.getString(1) ;
21)        salary = r.getDouble(2) ;
22)        system.out.println(lname + salary) ;
23)     }  }
24)  }
```

declaration

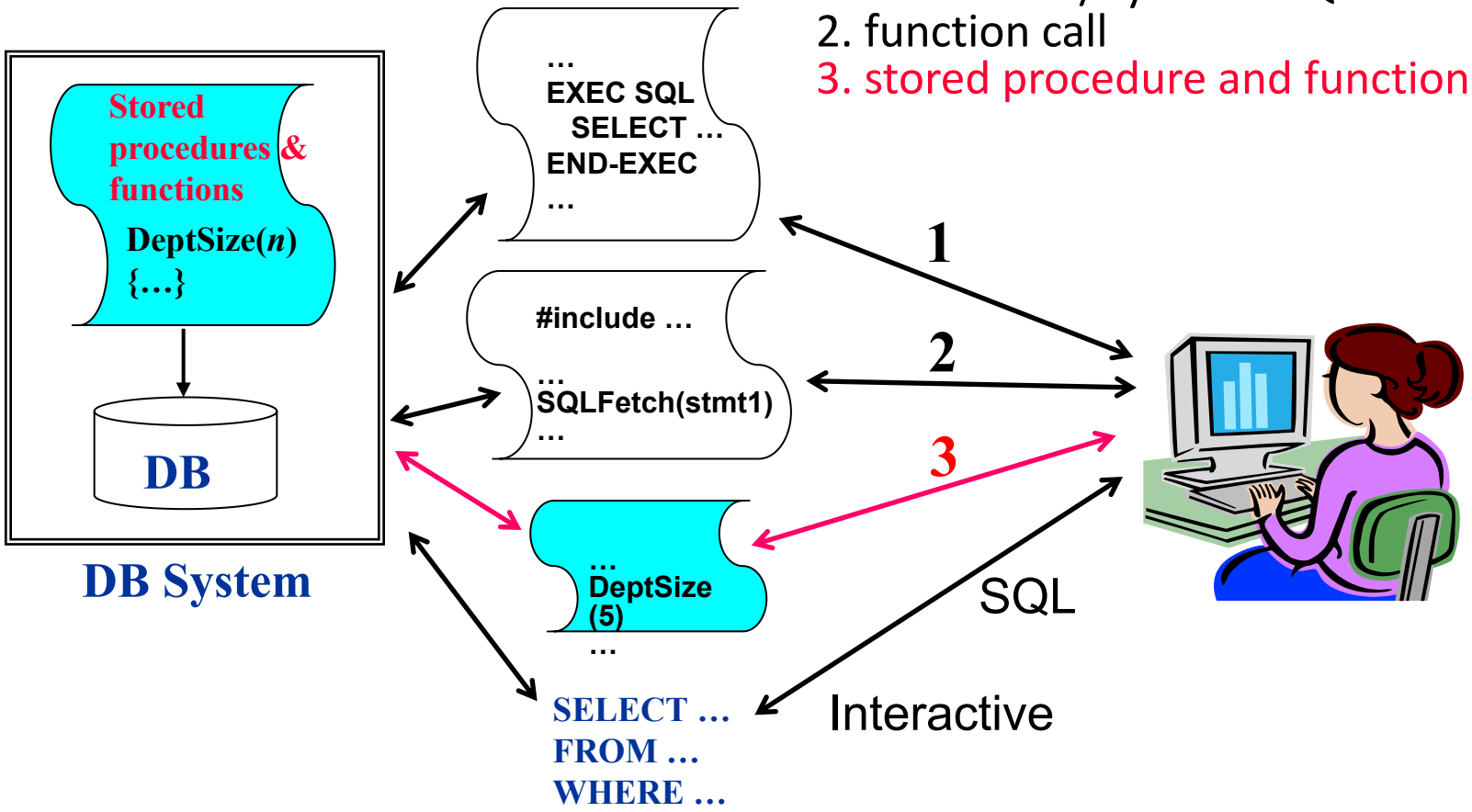
connect

prepare query

process query results

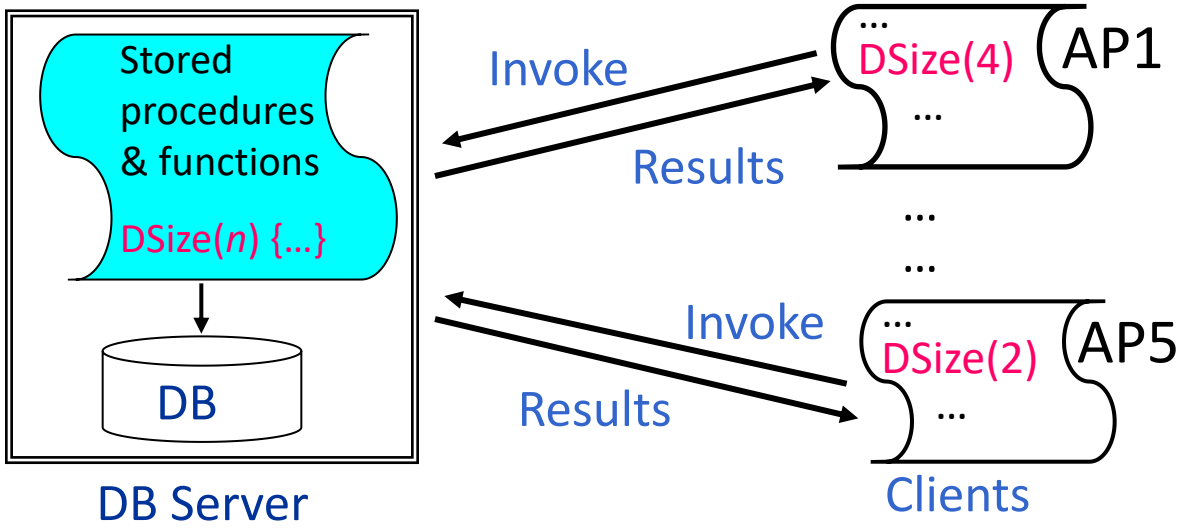
Database Programming

Database programming
1. embedded/dynamic SQL
2. function call
3. stored procedure and function



Database Stored Procedures

- 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



Stored Procedure Constructs

- A stored procedure

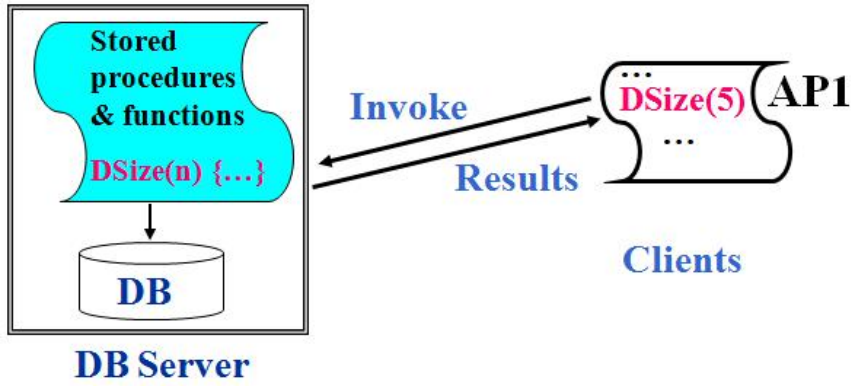
  CREATE PROCEDURE procedure-name (params)
  local-declarations
  procedure-body;

- A stored function

  CREATE FUNCTION fun-name (params) **RETRUNS** return-type
  local-declarations
  function-body;

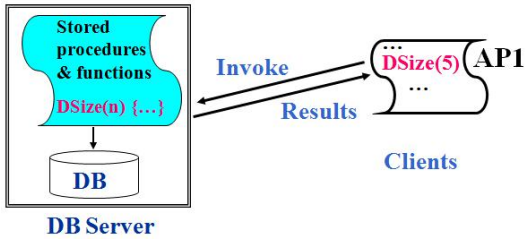
- Calling a procedure or function

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



SQL Persistent Stored Modules

- SQL/PSM
  - part of the SQL standard for writing persistent stored modules (PSM)
- SQL + stored procedures/functions + additional programming constructs
  - e.g., branching and looping statements
  - enhance the power of SQL



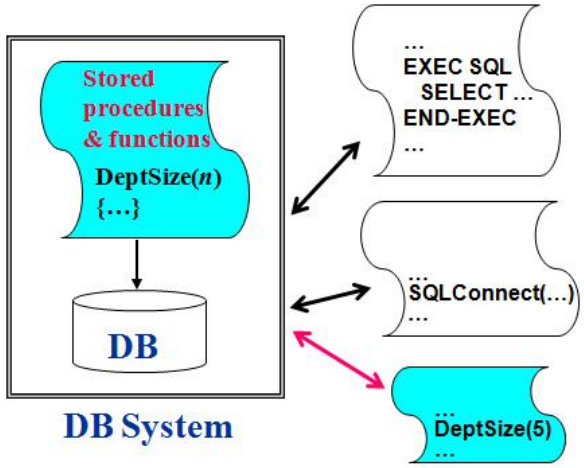

```
//Function PSM1:
0)   CREATE FUNCTION DeptSize(IN deptno INTEGER)
1)   RETURNS VARCHAR [7]
2)   DECLARE NoOfEmps INTEGER ;
3)   SELECT COUNT(*) INTO NoOfEmps
4)   FROM EMPLOYEE WHERE DNO = deptno ;
5)   IF NoOfEmps > 100 THEN RETURN "HUGE"
6)      ELSEIF NoOfEmps > 25 THEN RETURN "LARGE"
7)      ELSEIF NoOfEmps > 10 THEN RETURN "MEDIUM"
8)      ELSE RETURN "SMALL"
9)   END IF ;
```

SQL

Additional program control

Comparing the Three Approaches

- **Embedded SQL**
  - **Advantages**
    - Syntax errors are checked at compile time
    - Program is more readable
  - **Disadvantage**
    - Loss of flexibility in changing the query at runtime
- **Library of Function Calls**
  - **Advantage**
    - Flexibility: query can be generated at runtime
  - **Disadvantage**
    - Complex programming: programmer needs to check for runtime errors
- **Database Programming Language**
  - **Advantage**
    - No impedance mismatch problem
  - **Disadvantage**
    - Need to learn a new language



Summary

- A database may be accessed via
  - an interactive interface
  - application programs