# Embedded SQL

- Navathe 6th Edition

## 13.1 Database Programming: Techniques and Issues

- Most database systems have an **interactive interface** where these SQL commands can be typed directly into a monitor for execution by the database system.
- For example, in a computer system where the Oracle RDBMS is installed, the command SQLPLUS starts the interactive interface.
- The user can type SQL commands or queries directly over several lines, ended by a semicolon and the Enter key (that is, "`; <cr>`").
- Alternatively, a **file of commands** can be created and executed through the interactive interface by typing `@`<*filename*>.
- The system will execute the commands written in the file and display the results, if any.

## 13.1 Database Programming: Techniques and Issues

- **Application programs** or **database applications**, can be used to interact with the datatbase.
- We can access a database through an application program that implements a **Web interface**, for example, when making airline reservations or online purchases.
- Web database programming uses PHP, a scripting language that has recently become widely used.

## 13.1.1 Approaches to Database Programming

- Several techniques exist for including database interactions in application programs. The main approaches for database programming are the following:
- 1. **Embedding database commands in a general-purpose programming language.**
- In this approach, database statements are **embedded** into the host programming language, but 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 **preproccessor** 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 calls to the DBMS-generated code. This technique is generally referred to as **embedded SQL**.

## 13.1.1 Approaches to Database Programming

- 2. **Using a library of database functions.**
- A **library of functions** is made available to the host programming language for database calls.
- For example, there could be functions to connect to a database, execute a query, execute an update, and so on.
- The actual database query and update commands and any other necessary information are included as parameters in the function calls.
- This approach provides what is known as an **application programming interface** (**API**) for accessing a database from application programs.

## 13.1.1 Approaches to Database Programming

- 3. **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 fullfledged programming language.
- An example of this approach is Oracle’s PL/SQL.
- In practice, the first two approaches are more common, since many applications are already written in general-purpose programming languages but require some database access. The third approach is more appropriate for applications that have intensive database interaction.
- One of the main problems with the first two approaches is *impedance mismatch*, which does not occur in the third approach.

## 13.1.2 Impedance Mismatch

- **Impedance mismatch** is the term used to refer to the problems that occur because of differences between the database model and the programming language model.
- For example, the practical relational model has three main constructs: columns (attributes) and their data types, rows (also referred to as tuples or records), and tables (sets or multisets of records).
- The first problem that may occur is that the *data types of the programming language* differ from the *attribute data types* that are available in the data model. Hence, it is necessary to have a **binding** for each host programming language that specifies for each attribute type the compatible programming language types.
- A different binding is needed *for each programming language* because different languages have different data types.

## 13.1.2 Impedance Mismatch

- Another problem occurs because the results of most queries are sets or multisets of tuples (rows), and each tuple is formed of a sequence of attribute values.
- In the program, it is often necessary to access the individual data values within individual tuples for printing or processing.
- Hence, a binding is needed to map the *query result data structure*, which is 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 to 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 **iterator variable** is typically used to loop over the tuples in a query result.
- Individual values within each tuple are then extracted into distinct program variables of the appropriate type.

## 13.1.3 Typical Sequence of Interaction in Database Programming

- When a programmer or software engineer writes a program that requires access to a database, it is quite common for the program to be running on one computer system while the database is installed on another.
- A common architecture for database access is the client/server model, where a **client program** handles the logic of a software application, but includes some calls to one or more **database servers** to access or update the data.
- When writing such a program, a common sequence of interaction is the following:
- 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.

## 13.1.3 Typical Sequence of Interaction in Database Programming

- 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.
- 3. When the program no longer needs access to a particular database, it 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

## 13.2 Embedded SQL, Dynamic SQL, and SQLJ

- **13.2.1 Retrieving Single Tuples with Embedded SQL**
- **13.2.2 Retrieving Multiple Tuples with Embedded SQL Using Cursors**

## 13.2.1 Retrieving Single Tuples with Embedded SQL

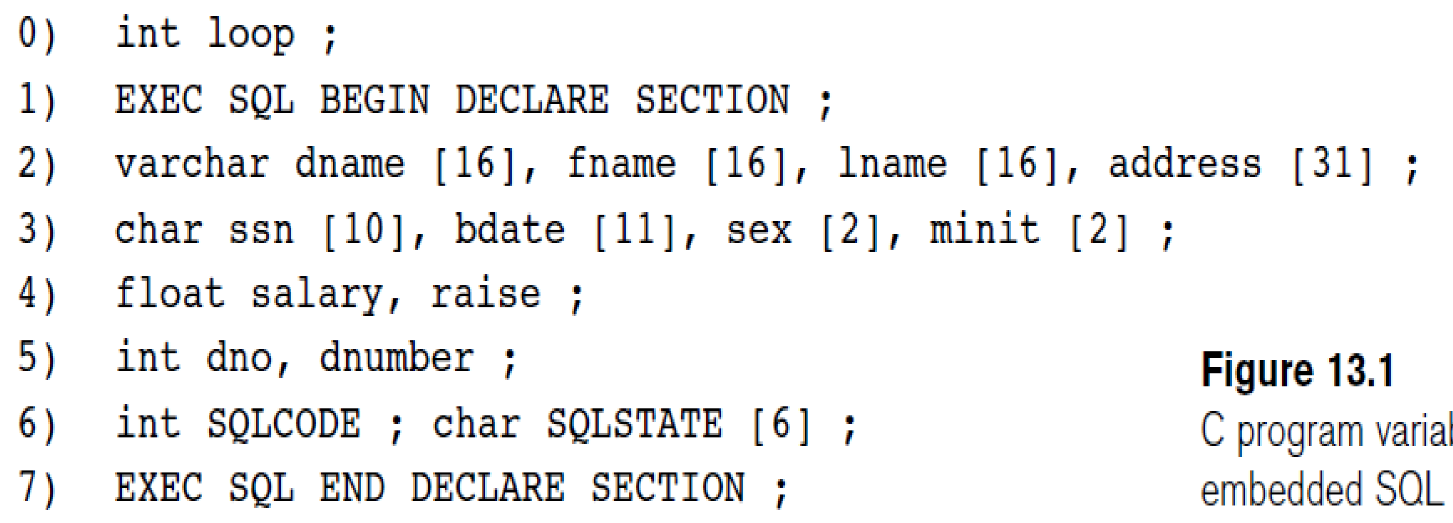
- When using C as the host language, an embedded SQL statement is distinguished from programming language statements by prefixing it with the keywords EXEC SQL so that a **preprocessor** (or **precompiler**) can separate embedded SQL statements from the host language code.
- The SQL statements within a program are terminated by a matching END-EXEC or by a semicolon (;).
- Within an embedded SQL command, we may refer to specially declared C program variables.
- These are called **shared variables** because they are used in both the C program and the embedded SQL statements.
- Shared variables are prefixed by a colon (:) *when they appear in an SQL statement*.
- This distinguishes program variable names from the names of database schema constructs such as attributes (column names) and relations (table names).

## 13.2.1 Retrieving Single Tuples with Embedded SQL

- It also allows program variables to have the same names as attribute names, since they are distinguishable by the colon (:) prefix in the SQL statement.
- Suppose that we want to write C programs to process the COMPANY database.
- We need to declare program variables to match the types of the database attributes that the program will process.
- The programmer can choose the names of the program variables; they may or may not have names that are identical to their corresponding database attributes.
- Shared variables are declared within a declare section in the program, as shown in Figure 13.1 (lines 1 through 7).

## 13.2.1 Retrieving Single Tuples with Embedded SQL

- 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 arrays 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 the character string itself.



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

**Figure 13.1**
C program variables used in the embedded SQL examples E1 and E2.

## 13.2.1 Retrieving Single Tuples with Embedded SQL

- Notice that the only embedded SQL commands in Figure 13.1 are lines 1 and 7, which tell the precompiler to take note of the C variable names between BEGIN DECLARE and END DECLARE because they can be included in embedded SQL statements— as long as they are preceded by a colon (:).
- Lines 2 through 5 are regular C program declarations.
- The C program variables declared in lines 2 through 5 correspond to the attributes of the EMPLOYEE and DEPARTMENT tables from the COMPANY database.
- The variables declared in line 6—SQLCODE and SQLSTATE—are used to communicate errors and exception conditions between the database system and the executing program.
- Line 0 shows a program variable loop that will not be used in any embedded SQL statement, so it is declared outside the SQL declare section.

## 13.2.1 Retrieving Single Tuples with Embedded SQL

- **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> ;

## 13.2.1 Retrieving Single Tuples with Embedded SQL

- **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 Figure 13.1 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 (records) are available in a query result.
- If SQLCODE < 0, this indicates some error has occurred.
- In some systems—for example, in the Oracle RDBMS— SQLCODE is a field in a record structure called 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 the following line:
- EXEC SQL include SQLCA ;

## 13.2.1 Retrieving Single Tuples with Embedded SQL

- In 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 values indicate various 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, whereas the codes returned in SQLCODE are not standardized but are defined by the DBMS vendor.
- Hence, it is generally better to use SQLSTATE because this makes error handling in the application programs independent of a particular DBMS.

## 13.2.1 Retrieving Single Tuples with Embedded SQL

- **Example of Embedded SQL Programming.**
- The first example to illustrate embedded SQL programming is a repeating program segment (loop) that takes as input a Social Security number of an employee and prints some information from the corresponding EMPLOYEE record in the database.
- The C program code is shown as program segment E1 in Figure 13.2.

```
//Program Segment E1:
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: ", ssn) ;
  prompt("More Social Security Numbers (enter 1 for Yes, 0 for No): ", loop) ;
  }
```

**Figure 13.2**
Program segment E1, a C program segment with embedded SQL.

## 13.2.1 Retrieving Single Tuples with Embedded SQL

- The program reads (inputs) an Ssn value and then retrieves the EMPLOYEE tuple with that Ssn from the database via the embedded SQL command.
- The **INTO** clause (line 5) specifies the program variables into which attribute values from the database record are retrieved.
- C program variables in the INTO clause are prefixed with a colon (:).
- The INTO clause can be used in this way only when the query result is a single record; if multiple records are retrieved, an error will be generated.
- Line 7 in E1 illustrates the communication between the database and the program through the special variable SQLCODE.
- If the value returned by the DBMS in SQLCODE is 0, the previous statement was executed without errors or exception conditions.

## 13.2.1 Retrieving Single Tuples with Embedded SQL

- Line 7 checks this and assumes that if an error occurred, it was because no EMPLOYEE tuple existed with the given Ssn; therefore it outputs a message to that effect (line 8).
- In E1 a *single record* is selected by the embedded SQL query (because Ssn is a key attribute of EMPLOYEE);. When a single record is retrieved, the programmer can assign its attribute values directly to C program variables in the INTO clause, as in line 5.
- In general, an SQL query can retrieve many tuples. In that case, the C program will typically go through the retrieved tuples and process them one at a time.
- The concept of a *cursor* is used to allow tuple-at-a-time processing of a query result by the host language program.

## 13.2.2 Retrieving Multiple Tuples with Embedded SQL Using Cursors

- We can think of a **cursor** as a pointer that points to a *single tuple* (*row*) from the result of a query that retrieves multiple tuples.

- The cursor is declared when the SQL query command is declared in the program.

- Later in the 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.

- This becomes the **current row** for the cursor.

- Subsequently, **FETCH** commands are issued in the program; each FETCH moves the cursor to the *next row* in the result of the query, making it the current row and copying its attribute values into the C (host language) program variables specified in the FETCH command by an INTO clause.

- The cursor variable is basically an **iterator** that iterates (loops) over the tuples in the query result—one tuple at a time.

## 13.2.2 Retrieving Multiple Tuples with Embedded SQL Using Cursors

- To determine when all the tuples in the result of the query have been processed, the communication variable SQLCODE (or, alternatively, SQLSTATE) is checked.
- If a FETCH command is issued that results in moving the cursor past the last tuple in the result of the query, a positive value (SQLCODE > 0) is returned in SQLCODE, indicating that no data (tuple) was found (or the string ‘02000’ is returned in SQLSTATE).
- The programmer uses this to terminate a loop over the tuples in the query result.
- In general, numerous cursors can be opened at the same time.
- A **CLOSE CURSOR** command is issued to indicate that we are done with processing the result of the query associated with that cursor.
- An example of using cursors to process a query result with multiple records is shown in Figure 13.3, where a cursor called EMP is declared in line 4.

## 13.2.2 Retrieving Multiple Tuples with Embedded SQL Using Cursors

**Figure 13.3**
Program segment E2, a C program segment that uses cursors with embedded SQL for update purposes.

```
//Program Segment E2:
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 = :dnumber
  FOR 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 EMPLOYEE
    set Salary = Salary + :raise
    where CURRENT OF EMP ;
  EXEC SQL FETCH from EMP into :ssn, :fname, :minit, :lname, :salary ;
  }
EXEC SQL CLOSE EMP ;
```

## 13.2.2 Retrieving Multiple Tuples with Embedded SQL Using Cursors

- The EMP cursor is associated with the SQL query declared in lines 5 through 6, but the query is not executed until the OPEN EMP command (line 8) is processed.
- The OPEN <cursor name> command executes the query and fetches its result as a table into the program workspace, where the program can loop through the individual rows (tuples) by subsequent FETCH <cursor name> commands (line 9).
- The program segment in E2 reads (inputs) a department name (line 0), retrieves the matching department number from the database (lines 1 to 3), and then retrieves the employees who work in that department via the declared EMP cursor.
- A loop (lines 10 to 18) iterates over each record in the query result, one at a time, and prints the employee name.
- The program then reads (inputs) a raise amount for that employee (line 12) and updates the employee's salary in the database by the raise amount that was provided (lines 14 to 16).
- This example also illustrates how the programmer can update database records.

## 13.2.2 Retrieving Multiple Tuples with Embedded SQL Using Cursors

- When a cursor is defined for rows that are to be modified (**updated**), 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.
- This is illustrated in line 7 of code segment E2.
- If rows are to be **deleted**, the keywords **FOR UPDATE** must be added without specifying any attributes.
- In the embedded UPDATE (or DELETE) command, the condition **WHERE CURRENT OF**<cursor name> specifies that the current tuple referenced by the cursor is the one to be updated (or deleted), as in line 16 of E2.
- Notice that declaring a cursor and associating it with a query (lines 4 through 7 in E2) does not execute the query; the query is executed only when the OPEN <cursor name> command (line 8) is executed.
- Also notice that there is no need to include the **FOR UPDATE OF** clause in line 7 of E2 if the results of the query are to be used *for retrieval purposes only* (no update or delete).

## 13.2.2 Retrieving Multiple Tuples with Embedded SQL Using Cursors

**General Options for a Cursor Declaration.** Several options can be specified when declaring a cursor. 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).
- If some of the tuples in the query result are to be updated, we need to specify FOR UPDATE OF <attribute list> and list the attributes that may be updated.
- If some tuples are to be deleted, we need to specify FOR UPDATE without any attributes listed.
- When the optional keyword SCROLL is specified in a cursor declaration, it is possible to position the cursor in other ways than for purely sequential access.

## 13.2.2 Retrieving Multiple Tuples with Embedded SQL Using Cursors

- A **fetch orientation** can be added to the FETCH command, whose value can be one of NEXT, PRIOR, FIRST, LAST, ABSOLUTE *i*, and RELATIVE *i*.
- In the latter two commands, *i* must evaluate to an integer value that specifies an absolute tuple position within the query result (for ABSOLUTE *i*), or a tuple position relative to the current cursor position (for RELATIVE *i*).
- The default fetch orientation, which we used in our examples, is NEXT.
- The fetch orientation allows the programmer to move the cursor around the tuples 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 of a FETCH command is as follows, with the parts in square brackets being optional:
- **FETCH** [ [ <fetch orientation> ] **FROM** ] <cursor name> **INTO** <fetch target list> ;

## 13.2.2 Retrieving Multiple Tuples with Embedded SQL Using Cursors

- The ORDER BY clause orders the tuples so that the FETCH command will fetch them in the specified order.
- It is specified in a similar manner to the corresponding clause for SQL queries.
- The last two options when declaring a cursor (INSENSITIVE and WITH HOLD) refer to transaction characteristics of database programs.

**END of CHAPTER 13**