# COS221
# L15 an L16 - SQL Database Programming
(Chapter 13 in Edition 6 and Chapter 10 in Edition 7)

Linda Marshall

27 and 30 March 2023

## Accessing data from application programs

Two categories of application programmes exist:

- **Web-based applications** - An application of the *three-tier architecture*
  - Web interface the client
  - Application program, the middle layer, implements the business logic
  - Database server forms the bottom layer.

  An application does not necessarily link to a single database, it may link to multiple database systems.
- **Canned applications**
  - SQL Workbench is an example of an interactive interface.
  - A file of SQL commands can be presented to the database system.

## Accessing data from application programs - *Web-based applications*

Typical sequence of interaction with the database (e.g. access to Wheatley):

- establish or open a connection to the database server
  - URL
  - login account name
  - password for database access
- program interacts with the database by submitting queries, updates, and other database commands.
- program terminates or closes the connection after using the access.

## Accessing data from application programs - *Canned applications*

**Impedence mismatch** occurs when there is a difference between the programming language model and the database model. This occurs when:

- the programming language type does not correspond to the database type. Need to bind types between the programming language and the database.
- mapping between database set/multiset and a data structure in the programming language

## Accessing data from application programs - *Canned applications*

Approaches to include database interactions in application programs:

- **Embed database commands in a general-purpose programming language** - database statements are embedded into the host programming language, but they are identified by a special prefix and scanned by a pre-compiler.
- **Use a library of database functions or classes** - makes use of an API to access the database. For example, there could be functions to connect to a database, execute a query, execute an update, and so on.
- **Design a new language** - a database programming language is designed from scratch to be compatible with the database model and query language.

## Canned Approach 1 - Embed database commands in a general-purpose programming language

- In this embedded approach, the programming language is called the host language.
- Examples of host languages include C/C++, Java or C#
  – Embedded SQL's host language is C/C++
  – Dynamic SQL
  – SQLJ's host language is Java

## Canned Approach 1 - Embed database commands in a general-purpose programming language - Embedded SQL in C/C++

Embedded SQL statements include data or constraint definitions, queries, updates, or view definitions.
Defining tuples

- **Identifying Embedded SQL statements**
  – Prefix SQL statements with EXEC SQL for the preprocessor (or precompiler) to separate SQL statements from the host language code.
  – The SQL statements within a program are terminated by a matching END-EXEC or by a semicolon (;).
- **Shared variables** - used in both the C program and the embedded SQL statements - prefixed by a colon (:)
  – (:) distinguishes program variable names from the names of database schema constructs such as column names and table names.
  – (:) allows program variables to have the same names as attribute names
  – COMPANY database (EMPLOYEE and DEPARTMENT) program variables declared to match the types of the database attributes that the program will process.
- **Data types**
  – The SQL types INTEGER, SMALLINT, REAL, and DOUBLE are mapped to the C types long, short, float, and double, respectively
  – Although `varchar` is not a standard C data type, it is permitted when C is used for SQL database programming

## Canned Approach 1 - Embed database commands in a general-purpose programming language - Embedded SQL in C/C++

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

**Figure 10.1**
C program variables used in the embedded SQL examples E1 and E2.

- Line 1 and 7 – embedded SQL commands. (tell precompiler about shared variables)
- Lines 2 through 5 – regular C program declarations
- Line 6 – SQLCODE and SQLSTATE are used to communicate errors and exception conditions

  SQLCODE - integer variable.
  For every command executed, DBMS returns a value in SQLCODE. i.e
  0 - statement executed successfully
  100 - no more data available in query result
  $<$ 0 - indicates some error has occurred

  SQLSTATE - String of five characters.
  '00000' $=$ no error or exception
  Other values indicate various errors or exceptions.
  For example, '02000' indicates 'no more data'

## Canned Approach 1 - Embed database commands in a general-purpose programming language - Embedded SQL in C/C++

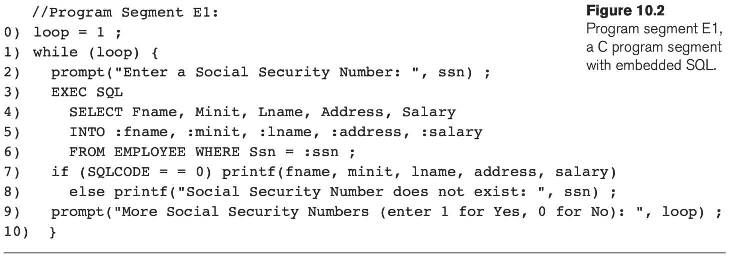
**Making database connections**

- Connecting to the database

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

  `SET CONNECTION <connection name> ;`
- Terminate connection

  `DISCONNECT <connection name> ;`



```
 //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 10.2**
Program segment E1, a C program segment with embedded SQL.

## Canned Approach 1 - Embed database commands in a general-purpose programming language - Embedded SQL in C/C++

**Processing query results using cursors**

- A cursor is a variable that refers to a single tuple from a query result that retrieves a collection of tuples.
- A cursor is used to loop over the query result, one record at a time.
- General cursor options

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

## Canned Approach 1 - Embed database commands in a general-purpose programming language - Embedded SQL in C/C++

### Processing query results using cursors

**Figure 10.3**
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 ;
```

## Canned Approach 1 - Embed database commands in a general-purpose programming language - Embedded SQL in C/C++

### Processing query results using cursors

- OPEN CURSOR is used fetches the query result and sets the cursor pointer to before the first entry
- FETCH moves the cursor to the next row in the result of the query and copies attribute values into the C program variables specified in the FETCH command by an INTO clause
- CLOSE CURSOR is issued to indicate that we are done with processing the result of the query associated with that cursor
- FOR UPDATE OF lists the names of any attributes that will be updated by the program. If rows are to be deleted, FOR UPDATE must be added without specifying any attributes
- WHERE CURRENT OF ¡cursor name¿ specifies the current tuple referenced by the cursor that is to be deleted or updated

## Canned Approach 1 - Embed database commands in a general-purpose programming language - Dynamic SQL

- Moves the specification of queries from compile-time to run-time.
- Is necessary when the user is allowed to specify their own query
- Updates are relatively easy to include, dynamic retrievals are a little more tricky due to not knowing the attributes or their types when writing the program. Therefore, a complex data structure is required to manage the results of a query.

## Canned Approach 1 - Embed database commands in a general-purpose programming language - SQLJ in Java

SQLJ's host language is Java and is the standard adopted by several vendors for embedding SQL in Java. SQLJ translates SQL into Java that are executed through the JDBC interface to an Oracle DBMS. It is therefore necessary to install a JDBC driver when using SQLJ.

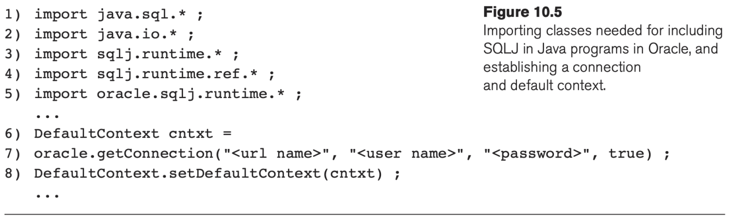
- Import several class libraries

```
import java.sql.* ;
import java.io.* ;
import sqlj.runtime.* ;
```

- Uses exceptions for error handling
  SQLException is used to return errors or exception conditions

# Canned Approach 1 - Embed database commands in a general-purpose programming language - SQLJ in Java

- Establishing a connection



```
import java.sql.* ;
import java.io.* ;
import sqlj.runtime.* ;
import sqlj.runtime.ref.* ;
import oracle.sqlj.runtime.* ;
...
DefaultContext cntxt =
oracle.getConnection("<url name>", "<user name>", "<password>", true) ;
DefaultContext.setDefaultContext(cntxt) ;
...
```

**Figure 10.5**
Importing classes needed for including SQLJ in Java programs in Oracle, and establishing a connection and default context.

- Java program variables

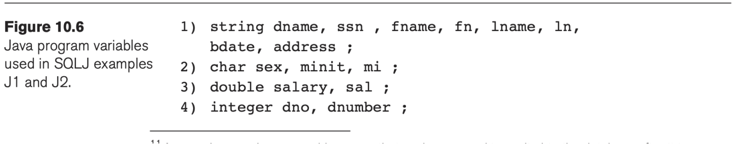


**Figure 10.6**
Java program variables used in SQLJ examples J1 and J2.

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

## Canned Approach 1 - Embed database commands in a general-purpose programming language - SQLJ in Java

- Accessing the data

```
//Program Segment J1:
ssn = readEntry("Enter a Social Security Number: ") ;
try {
  #sql { SELECT Fname, Minit, Lname, Address, Salary
    INTO :fname, :minit, :lname, :address, :salary
    FROM EMPLOYEE WHERE Ssn = :ssn} ;
} catch (SQLException se) {
    System.out.println("Social Security Number does not exist: " + ssn) ;
    Return ;
  }
System.out.println(fname + " " + minit + " " + lname + " " + address
  + " " + salary)
```

**Figure 10.7**
Program segment J1, a Java program segment with SQLJ.

## Canned Approach 1 - Embed database commands in a general-purpose programming language - SQLJ in Java

- Processing query results with iterators
  An iterator is an object associated with a set/multiset, the query result.
  There are two types of iterators:
  - **Named iterator** - Associated with a query result by listing attribute names and types in query result
  - **Positional iterator** - Lists only attribute types in query result. Makes use of FETCH.

## Canned Approach 1 - Embed database commands in a general-purpose programming language - SQLJ in Java

### Named iterator

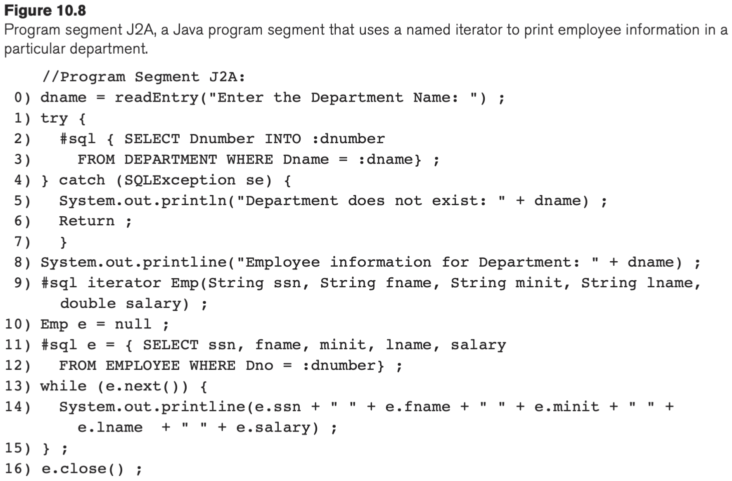


**Figure 10.8**
Program segment J2A, a Java program segment that uses a named iterator to print employee information in a particular department.

```
//Program Segment J2A:
dname = readEntry("Enter the Department Name: ") ;
try {
  #sql { SELECT Dnumber INTO :dnumber
    FROM DEPARTMENT WHERE Dname = :dname} ;
} catch (SQLException se) {
  System.out.println("Department does not exist: " + dname) ;
  Return ;
  }
System.out.printline("Employee information for Department: " + dname) ;
#sql iterator Emp(String ssn, String fname, String minit, String lname,
  double salary) ;
Emp e = null ;
#sql e = { SELECT ssn, fname, minit, lname, salary
  FROM EMPLOYEE WHERE Dno = :dnumber} ;
while (e.next()) {
  System.out.printline(e.ssn + " " + e.fname + " " + e.minit + " " +
    e.lname  + " " + e.salary) ;
} ;
e.close() ;
```

## Canned Approach 1 - Embed database commands in a general-purpose programming language - SQLJ in Java

### Positional iterator

**Figure 10.9**
Program segment J2B, a Java program segment that uses a positional iterator to print employee information in a particular department.

```
//Program Segment J2B:
dname = readEntry("Enter the Department Name: ") ;
try {
 #sql { SELECT Dnumber INTO :dnumber
  FROM DEPARTMENT WHERE Dname = :dname} ;
} catch (SQLException se) {
 System.out.println("Department does not exist: " + dname) ;
 Return ;
 }
System.out.printline("Employee information for Department: " + dname) ;
#sql iterator Emppos(String, String, String, String, double) ;
Emppos e = null ;
#sql e = { SELECT ssn, fname, minit, lname, salary
 FROM EMPLOYEE WHERE Dno = :dnumber} ;
#sql { FETCH :e INTO :ssn, :fn, :mi, :ln, :sal} ;
while (!e.endFetch()) {
 System.out.printline(ssn + " " + fn + " " + mi + " " + ln + " " + sal) ;
 #sql { FETCH :e INTO :ssn, :fn, :mi, :ln, :sal} ;
} ;
e.close() ;
```

## Canned Approach 2 - Using a library of database functions or classes

Till now, the programming has been static. The query text has been fixed and cannot be changed. Libraries (Application Programming Interface - API) allow for dynamic database querying.

Two function call interfaces will be discussed:

- SQL Call Level Interface (SQL/CLI), part of the SQL standard – provides a standardisation of the Open DataBase Connection (ODBC) library
- JDBC$^{TM}$, an interface for accessing the database from Java

## Canned Approach 2 - Using a library of database functions or classes - SQL/CLI

**SQL/CLI and ODBC using C as the host language** SQL statements are dynamically created and passed as string parameters in the function calls.
There is a need to keep track of the information about host program interactions with the database. This is done using 4 records which are defined as C structs and are accessed through a C pointer (handle) to the struct.

- Environment record - Tracks one or more database connections and sets environment information
- Connection record - Keeps track of information needed for a particular database connection
- Statement record - Keeps track of the information needed for one SQL statement
- Description record - Keeps track of information about tuples or parameters

## Canned Approach 2 - Using a library of database functions or classes - SQL/CLI

**SQL/CLI and ODBC using C as the host language**

To create a record and return its handle, the following SQL/CLI function is used:

`SQLAllocHandle(<handle_type>, <handle_1>, <handle_2>)`

- `<handle_type>` indicates the type of record being created. The possible values for this parameter are the keywords SQL_HANDLE_ENV, SQL_HANDLE_DBC, SQL_HANDLE_STMT, or SQL_HANDLE_DESC, for an environment, connection, statement, or description record, respectively.
- `<handle_1>` indicates the container within which the new handle is being created.
- `<handle_2>` is the pointer to the newly created record of type `<handle_type>`.

## Canned Approach 2 - Using a library of database functions or classes - SQL/CLI

```
//Program CLI1:
#include sqlcli.h ;
void printSal() {
SQLHSTMT stmt1 ;
SQLHDBC con1 ;
SQLHENV env1 ;
SQLRETURN ret1, ret2, ret3, ret4 ;
ret1 = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env1) ;
if (!ret1) ret2 = SQLAllocHandle(SQL_HANDLE_DBC, env1, &con1) else exit ;
if (!ret2) ret3 = SQLConnect(con1, "dbs", SQL_NTS, "js", SQL_NTS, "xyz",
    SQL_NTS) else exit ;
if (!ret3) ret4 = SQLAllocHandle(SQL_HANDLE_STMT, con1, &stmt1) else exit ;
SQLPrepare(stmt1, "select Lname, Salary from EMPLOYEE where Ssn = ?",
    SQL_NTS) ;
prompt("Enter a Social Security Number: ", ssn) ;
SQLBindParameter(stmt1, 1, SQL_CHAR, &ssn, 9, &fetchlen1) ;
ret1 = SQLExecute(stmt1) ;
if (!ret1) {
    SQLBindCol(stmt1, 1, SQL_CHAR, &lname, 15, &fetchlen1) ;
    SQLBindCol(stmt1, 2, SQL_FLOAT, &salary, 4, &fetchlen2) ;
    ret2 = SQLFetch(stmt1) ;
    if (!ret2) printf(ssn, lname, salary)
      else printf("Social Security Number does not exist: ", ssn) ;
    }
}
```

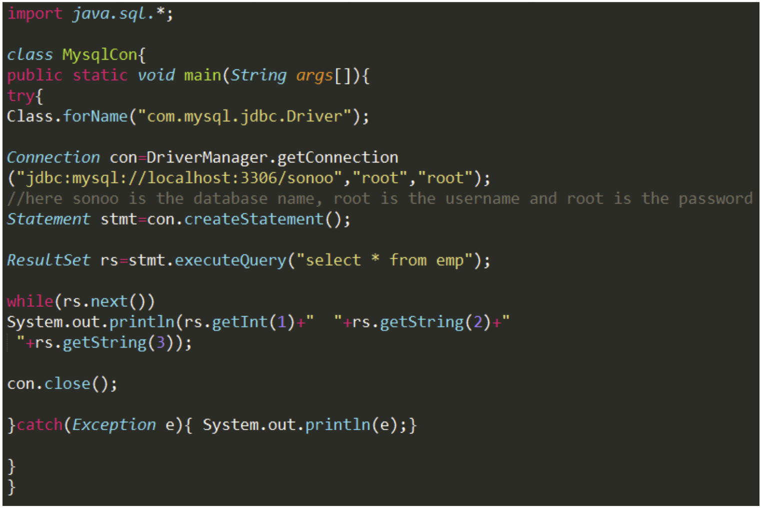
**Figure 10.10**
Program segment CLI1, a C program segment with SQL/CLI.

## Canned Approach 2 - Using a library of database functions or classes - JDBC

**JDBC - SQL Function Calls for Java Programming** Java program with JDBC function calls can access any RDBMS that has a JDBC driver available.

- Import the JDBC class libraries, which are called java.sql.*.
- Single Java program can connect to several different databases. Databases are referCred to as data sources accessed by the Java program
- Class.forName("com.mysql.jdbc.Driver"); Loads a JDBC driver explicitly (for mysql).

## Canned Approach 2 - Using a library of database functions or classes - JDBC



```
import java.sql.*;

class MysqlCon{
public static void main(String args[]){
try{
Class.forName("com.mysql.jdbc.Driver");

Connection con=DriverManager.getConnection
("jdbc:mysql://localhost:3306/sonoo","root","root");
//here sonoo is the database name, root is the username and root is the password
Statement stmt=con.createStatement();

ResultSet rs=stmt.executeQuery("select * from emp");

while(rs.next())
System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"
 "+rs.getString(3));

con.close();

}catch(Exception e){ System.out.println(e);}

}
}
```

More information is available from: - https://docs.oracle.com/javase/tutorial/jdbc/

## Canned Approach 3 - Design a new language

Additions are made to SQL. There are two basic additions:

- the concept of **Stored Procedures** (or Functions) - Persistent Stored Modules (SQL/PSM); and
- the inclusion of **constructs for selection and looping** in SQL.

## Design a new language

**Stored Procedures**

```
CREATE PROCEDURE <procedure name> (<parameters>)
<local declarations>
<procedure body> ;
```

```
CREATE FUNCTION <function name> (<parameters>)
                                  RETURNS <return type>
<local declarations>
<function body> ;
```

Calling a procedure or function:

```
CALL <procedure or function name> (<argument list>) ;
```

The procedure does not have to be defined in the DBMS, it can be in a “typical” programming language.

```
CREATE PROCEDURE <procedure name> (<parameters>)
  LANGUAGE <programming language name>
  EXTERNAL NAME <file path name> ;
```

## Canned Approach 3 - Design a new language

**Selection and Looping**

```
IF <condition> THEN <statement list>
ELSEIF <condition> THEN <statement list> ...
ELSEIF <condition> THEN <statement list>
ELSE <statement list>
END IF
```

```
WHILE <condition> DO <statement list>
END WHILE  REPEAT
<statement list> UNTIL <condition>
END REPEAT
```

```
FOR <loop name> AS <cursor name> CURSOR FOR <query> DO
<statement list>
END FOR
```

Breaking out of the loop is done by using the `LEAVE <loop name>` statement.

## (Dis)Advantages of the Approaches

- **Embed database commands in a general-purpose programming language**
  – *Advantage* - Query text checked for syntax errors and validated against database schema at compile time
  – *Disadvantage* - The query cannot be change at runtime. This means that for complex applications where queries have to be generated at runtime, the library of database functions approach is more suitable
- **Using a library of database functions or classes**
  – *Advantage* - More flexibility as queries can be generated at runtime.
  – *Disadvantages* - More complex programming and there is no checking of syntax done at compile time.
- **Design a new language**
  – *Advantage* - Does not suffer from the impedance mismatch problem, programming language and database types are the same
  – *Disadvantage* - Programmers must learn a new language which may differ between DBMS vendors