NULL Values & Indicators

- INDICATORS are used to hold the status of variables and test for NULL values.
- An indicator is associated with each variable:
  - Short integer (2 bytes).
    - -1 : indicates NULL value
    - 0 : indicates valid data value.
  - Always *check* indicator when reading.
  - Always *set* indicator when writing.
- Note that each field in a struct is a separate variable. That is, a 4 field struct is associated with 4 indicators
- Indicators could be a struct or an array depending on the implementation

Host Data for Single Retrieval

Student(SID,Name,Major)

```
Struct {
   int sid;
   VARCHAR   student_name[UNAME_LEN];
   char      major[5];
} student_rec;

Struct {
   short     sid_ind;
   short     student_name_ind;
   short     major_ind;
} student_rec_ind;
```


Embedded SQL Single Retrieval

```
printf("\nEnter Student number (0 to quit): ");
gets(temp_char);
student_number = atoi(temp_char);

EXEC SQL WHENEVER NOT FOUND GOTO notfound;

EXEC SQL SELECT SID, Name, Major
   INTO :student_rec INDICATOR :student_rec_ind
   FROM STUDENT
   WHERE SID = :student_number;

display_student(student_rec, student_rec_ind);
notfound display_error();
```


ESQL Cursors

- If more than one tuples can be selected, then tuples must be processed one at a time by means of a cursor
  - This is similar to the record-at-a-time processing
- A cursor is a "pointer" to a tuple in a result of a query
  - Current tuple w.r.t. a cursor is the tuple pointed by the cursor
- DECLARE <cursor-name> CURSOR FOR <query>
  - It defines a query and associates a cursor with it
- OPEN <cursor-name> brings the query result from the DB and positions the cursor before the first tuple
- CLOSE cursor-name closes the named cursor and deletes the associated result table

Cursor Retrieval

- **Sequential Access:**
  FETCH <cursor-name> INTO <variable-list>;
  - copies into variables the current tuple and advances the cursor
  - Use SQLCODE, SQLSTATE, or WHENEVER NOT FOUND to detect end of result table
- **Random Access:** (positioning of cursor)
  FETCH **orientation**
  FROM <cursor-name> INTO <variable-list>;
  - where **orientation**: NEXT (default), PRIOR, LAST, ABSOLUTE <offset>, RELATIVE <offset>

Cursor Retrieval Example

```
EXEC SQL DECLARE st_cursor CURSOR FOR
    SELECT SID, Name, Major
    FROM STUDENT
    WHERE Major = 'CS' ;
EXEC SQL OPEN st_cursor;
While (1) {
   EXEC SQL WHENEVER NOT FOUND DO break;
   EXEC SQL FETCH st_cursor
       INTO :student_rec INDICATOR :student_rec_ind;
  display_student(student_rec, student_rec_ind);
 };
EXEC SQL CLOSE st_cursor;
```


Dynamic SQL Statements

- An SQL statement is passed to DBMS in the form of a string to be interpreted and executed
- EXECUTE IMMEDIATE
- PREPARE, EXECUTE, USING
  - create/drop table
  - insert, delete, update
- Dynamic DECLARE CURSOR, DESCRIBE, OPEN, FETCH
  - select statement
- RELEASE

Dynamic SQL: Prepare-Execute-Using

- It compiles an SQL statement with parameters indicated with "?"

```
char sqltxt[] = "DELETE FROM STUDENT WHERE id = ? AND name = ?";
EXEC SQL PREPARE delcmd FROM : sqltxt;
```

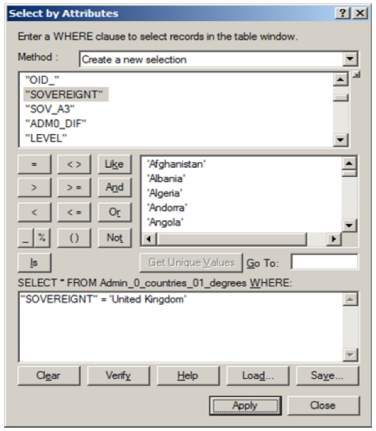
- USING statement allows the passing of parameters

```
cout << "Please Enter Student ID & Name to be deleted" << endl;
cin >> (char *) string_id;
cin >> (char *) student_name;
student_id=atoi(string_id);
EXEC SQL EXECUTE delcmd USING : student_id, : student_name;
```

- Release statement

```
EXEC SQL RELEASE delcmd;
```

SQL Builder



SQLJ: SQL-Java

- Semi-static version of embedded SQL in Java
- SQL statements are introduced with: *#sql*

```
#sql { delete from STUDENT where SID = :stid };
#sql { insert into STUDENT (SID) values (165) };
```

- *Iterator* object supports the notion of cursor

```
#sql iterator ST_Cursor (Integer Sid, String Name);
ST_Cursor stCursor;
#sql stCursor = { SELECT SID, Name  INTO :Sid, :Name
                  FROM STUDENT WHERE Major = 'none' };
while (stCursor.next()) {
  System.out.println(stCursor.Sid() + ", " + stCursor.Name()); }
stCursor.close();
```


SQLJ: Simple yet complete example

```
1.  import java.sql.*;      // you need this import for SQLException and other classes from JDBC
2.  import oracle.sqlj.runtime.Oracle;
3.  public class SingleRowQuery extends Base {
4.    public static void main(String[] args) {
5.      try {  connect();
6.             singleRowQuery(1);
7.      } catch (SQLException e) { e.printStackTrace(); }
8.    }
9.    public static void singleRowQuery(int id) throws SQLException {
10.     String fullname = null;
11.     String street = nul
12.     #sql {
13.       SELECT fullname,
14.         street INTO : OUT fullname,        //OUT is actually the default for INTO host variables
15.         : OUT street FROM customer WHERE ID = :id};
16.     System.out.println("Customer with ID = " + id);
17.     System.out.println();   System.out.println(fullname + " " + street);
18.   }
19. }
```


Database Programming Approaches

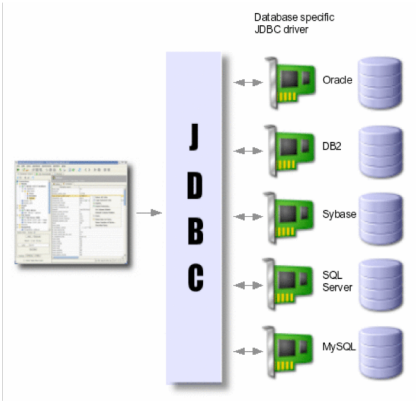
- Embedded commands:
  - Database commands are **embedded** in a general-purpose programming language
- Library of database functions:
  - Available to the host language for database calls; known as an **API** (Application Program Interface)
  - *e.g., JDBC, ODBC, PHP, Python*
- A brand new, full-fledged language
  - e.g., Oracle PL/SQL
  - **P**rocedural **L**anguage extensions to SQL

JDBC: An example of SQL API

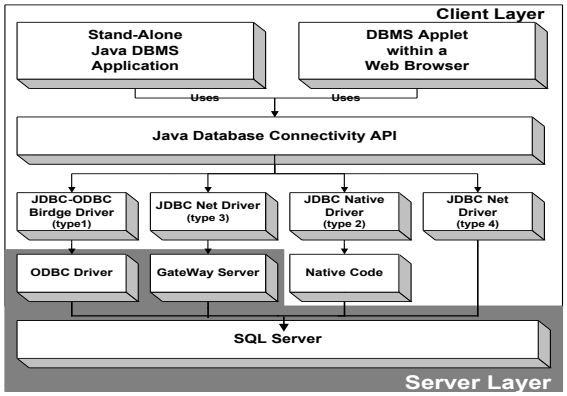
- JDBC resembles dynamic SQL, in which SQL statements are passed in the form of strings
- JDBC supports its own dialect of SQL
- An application program (Java applet) executes an SQL statement by submitting it to the JDBC driver manager
- Any database using can be accessed as long as an appropriate DBMS-specific driver exists, is loaded, and is registered with the driver manager:

```
import java.sql.*;
Class.forName("jdbc.driver_name");
```


JDBC Configuration



JDBC Drivers



JDBC Drivers

- Type 1: JDBC-ODBC bridge.
  This driver translates JDBC calls into ODBC calls.
- Type 2:
  Java JDBC Native Code. This partial Java driver converts JDBC calls into client API for the DBMS.
- Type 3: JDBC-Gateway.
  This pure Java driver connects to a database middleware server that in turn interconnects multiple databases and performs any necessary translations.
- Type 4:
  Pure Java JDBC. This driver connects directly to the DBMS.

Useful Links

- JDBC DRIVER
  - https://jdbc.postgresql.org/download.html
- JDBC API
  - https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/

Accessing a Database

- Open Connection:

```
Connection dbcon;
dbcon=
    DriverManager.getConnection(<"URL">,<"userId">,<"pwd">);
```

- E.g.,

```
import java.sql.*;
public class JavaDemo  {
private Connection dbcon;  private String username = "PittID", password = "PSNum";
  public JavaDemo() {
   Class.forName("org.postgresql.Driver");
   String url = "jdbc:postgresql://class3.cs.pitt.edu:5432/dbclass";
  dbcon = DriverManager.getConnection(url, username, password);
  …
```

- Close connection: dbcon.close();

Executing an SQL Statement

- Statement class: Execute SQL statements without parameters
  - Create statement object

```
Statement st;
 st = dbcon.createStatement();
```

  - Directly execute: Select, Update, Insert, Delete, DDL

```
st.executeQuery(<"sql-query">);
st.executeUpdate(<"sql-modification">);
```

- Example of an SQL modification

```
int numberrows = st.executeUpdate
("INSERT INTO STUDENT VALUES (123, 'J.J. Kay', 'CS' )");
```

  - Table can be prefixed by its schema, e.g., cs1555.STUDENT

Querying a database & Cursors

```
String fname = readString("Enter First Name: ");
String query1 = "SELECT SID, Name, Major FROM  STUDENT
                 WHERE Name  LIKE '" + fname + "  '"   ;
ResultSet res1 = st.executeQuery(query1);

int rsid; String rname, rmajor;
while (res1.next()) {
    rsid = res1.getInt("SID");
    rname = res1.getString("Name");
    rmajor = res1.getString(3);
    if (res1.wasNull()) {
      System.out.print(rsid+" "+rname+" NULL"); }
    else { System.out.print(rsid+" "+rname+" "+rmajor); }
};
```

getXXX(param)
XXX: valid SQL Type
param: name or index

wasNull() returns True if the last getXXX() value should be read as NULL

JDBC Example in PostgreSQL

```
package edu.pitt.cs;

import java.util.Properties;
import java.sql.*;

public class JavaDemo {
   public static void main(String args[]) throws
         SQLException, ClassNotFoundException {
      Class.forName("org.postgresql.Driver");
      String url = "jdbc:postgresql://localhost/postgres";
      Properties props = new Properties();
      props.setProperty("user", "postgres");
      props.setProperty("password", "password");
      Connection conn =
            DriverManager.getConnection(url, props);
```


JDBC Example in PostgreSQL

```
Statement st = conn.createStatement();
String query1 = "SELECT SID, Name, Major FROM
          CS1555.STUDENT WHERE Major='CS'";
ResultSet res1 = st.executeQuery(query1);
int rid; String rname, rmajor;
while (res1.next()) {
   rid = res1.getInt("SID");
   rname = res1.getString("Name");
   rmajor = res1.getString(3);
 If (res1.wasNull()) {
      System.out.println(rid + " " + rname + " " + "NULL");
   } else {
      System.out.println(rid + " " + rname + " " + rmajor);
   }
  }
}}
```