

**Oracle JDBC**

**CS 4400**

## Oracle JDBC

- JDBC is an application programming interface that enables database access in Java
- It consists of a set of classes and interfaces written in Java
- It allows the programmer to send SQL statements to a database server for execution and retrieve query results (for SQL queries)
- Provides for portability across database servers and hardware architectures

### Interaction with the database using JDBC

- Import the JDBC classes
- Load the JDBC drivers
- Connect to the database
- Interact with the database using JDBC
- Disconnect from the database

## Developing JDBC Applications

- Import the JDBC classes  

```
import java.sql.*;  
import oracle.jdbc.driver.*;  
import oracle.sql.*;
```
- Load the JDBC drivers  

```
DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver ());
```
- Connect to the database  

```
OracleConnection conn = (OracleConnection)  
DriverManager.getConnection("jdbc:oracle:oci8:@ccdb",user,pass);
```
- Interact with the database using JDBC
- Disconnect from the database  

```
conn.close();
```

## Creating JDBC Statements

- A JDBC statement object is used to send your SQL statement to the database server
- A JDBC statement object is associated with an open connection and not any single SQL statement
- By default, the Connection object automatically commits changes after executing each statement,
- If auto-commit has been disabled, then an explicit commit must be done to save the changes in the database
- JDBC provides three classes for sending SQL statements to the database server
  - Statement, used for SQL statements without parameters
  - PreparedStatement, used when the same statement with possible different parameters, is to be executed multiple times (it is pre-compiled and stored for future use)
  - CallableStatement, used for executing stored procedures

## Creating JDBC Statements

- Statement
  - The following code snippet uses the Connection object con to create a Statement object  
`Statement stmt = conn.createStatement ();`
  - No SQL statement is associated with it

## Creating JDBC Statements

- PreparedStatement
  - With this subclass of Statement, an SQL statement is provided with it when the PreparedStatement object is created and the SQL statement is passed to the database server right away, where it is compiled
  - The following code snippet uses the Connection object con to create a parameterized SQL statement with four input parameters  
PreparedStatement stmt = conn.prepareStatement( "insert into students values (?, ?, ?, ?)" );
  - Before a PreparedStatement can be executed, we will need to supply values for the parameters, which can be done by calling one of the setXXX methods defined in the class PreparedStatement  
stmt.setInt(1,id);  
stmt.setString(2,name);  
stmt.setString(3,city);  
stmt.setFloat(4,salary);
  - It can be executed with the following: stmt.executeUpdate();

## Creating JDBC Statements

- Executing Create/Insert/Update Statements
  - Create table  
stmt.executeUpdate("create table scores (" +  
"sid varchar2(5) not null," +  
"term varchar2(10) not null," +  
"lineno number(4) not null," +  
"compname varchar2(15) not null," +  
"points number(4) check(points >= 0))");

## Creating JDBC Statements

- Executing Create/Insert/Update Statements
  - Insert
 

```
PreparedStatement stmt2 = conn.prepareStatement(
    "insert into scores values (?, ?, ?, ?)"
    + cname + ", ?, ?)"
    );
```
  - Update
 

```
String query2 = "update scores set points = " + ns +
    " where sid = " + id + " and compname = " +
    cname + " and term = " + term_in + " and lineno = " +
    ls ;
stmt.executeUpdate(query2);
```
  - Delete
 

```
String query3 = "delete scores where sid = " + id +
    " and term = " + term_in + " and lineno = " + ls;
stmt.executeUpdate(query3);
```

## Creating JDBC Statements

- Executing Select statements
  - Use the method `executeQuery`, which returns its results as a `ResultSet` object

```
String query3 = "select points, compname " +  
"from scores " +  
"where term = '" + term_in + "' and " +  
"lineno = '" + ls + "' and " +  
"sid = ?";
```

```
PreparedStatement stmt2 = conn.prepareStatement(query3);
```

## Creating JDBC Statements

- Executing Select statements

```
stmt2.setString(1,studentid);
```

```
ResultSet rset3;
```

```
rset3 = stmt2.executeQuery();
```

```
while ( rset3.next() ) {
```

```
    points = rset3.getInt(1);
```

```
    if ( !rset3.isNull() ) {
```

```
        system.out.print("points = " + points);
```

```
    }
```

```
    comp_name = rset3.getString(2);
```

```
    if ( !rset3.isNull() ) {
```

```
        system.out.println("compname = " + comp_name);
```

```
    }
```

```
    }
```

## Creating JDBC Statements

- Executing Select statements
  - The bag of tuples resulting from the query are contained in the variable `rset3`, which is an instance of the `ResultSet`. The `ResultSet` provides a cursor, which can be used to access each row in the result. The cursor is initially set just before the first row and each invocation of the method `next` causes it to move to the next row and return `TRUE` if one exists and `FALSE` if there is no remaining row.
  - We can use the `getXXX` method of the appropriate type to retrieve the attributes of a row for example

```
id = rset.getInt(1);
name = rset.getString(2);
salary = rset.getFloat(3);
```

## Transactions and JDBC

- JDBC allows SQL statements to be grouped together into a single transaction
- Transaction control is performed by the Connection object, default mode is auto-commit, i.e., each SQL statement is treated as a transaction
- We can turn off the auto-commit mode with  
`con.setAutoCommit(false);`  
and turn it back on with  
`con.setAutoCommit(true);`
- Once auto-commit is off, no SQL statement will be committed until an explicit commit is invoked  
`con.commit();`  
At this point, all changes done by the SQL statements will be made permanent in the database

## Transactions and JDBC

- If we don't want certain changes to be made permanent, we can issue `con.rollback();`  
Any changes made since the last commit will be ignored - usually rollback is used in combination with Java's exception handling ability to recover from unpredictable errors.

- Example

```
con.setAutoCommit(false);
Statement stmt = con.createStatement();
stmt.executeUpdate("INSERT INTO scores VALUES ('12345','fall
2001',4400,'quiz 1', 0)" );
con.rollback();
stmt.executeUpdate("INSERT INTO scores VALUES ('12345','fall
2001',4400,'quiz 1',30)" );
con.commit();
con.setAutoCommit(true);
```

## Handling Errors with Exceptions

- Programs should recover and leave the database in a consistent state
- In Java, statements which are expected to "throw" an exception or a warning are enclosed in a try block
- If a statement in the try block throws an exception or warning, it can be caught in one of the corresponding catch statements

- Example

```
PreparedStatement stmt2 = conn.prepareStatement(
    "insert into scores values ('11111', 'fall 2001', 4400, 'quiz 1', 0)");
try{
    stmt2.executeUpdate();
} catch (SQLException e) {
    System.out.println("Score was not added! Error!");
    while (e != null) {
        System.out.println("Message : " + e.getMessage());
        e = e.getNextException();
    }
} }
```

### Simple JDBC Program

```
import java.sql.*;
import oracle.jdbc.driver.*;
import oracle.sql.*;
import java.math.BigDecimal;
import java.util.Map;
import java.io.*;

class simple {
    public static void main (String args [])
        throws SQLException, IOException {

        String user, pass;
        user = readEntry("userid : ");
        pass = readEntry("password: ");
```

### Simple JDBC Program

```
// Connect
DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver ());
OracleConnection conn = (OracleConnection)
DriverManager.getConnection("jdbc:oracle:oci8:@ccdb",
user,pass);
Statement stmt = conn.createStatement ();
ResultSet rset = stmt.executeQuery
("select distinct eno,ename,zip,hdate from employees");
while (rset.next ()) {
System.out.println(rset.getInt(1) + " " +
rset.getString(2) + " " +
rset.getInt(3) + " " +
rset.getDate(4));
}

conn.close();
}
```

### Simple JDBC Program

```
//readEntry function – to read input string
static String readEntry(String prompt) {
    try {
        StringBuffer buffer = new StringBuffer();
        System.out.print(prompt);
        System.out.flush();
        int c = System.in.read();
        while(c != '\n' && c != -1) {
            buffer.append((char)c);
            c = System.in.read();
        }
        return buffer.toString().trim();
    } catch (IOException e) {
        return "" ;
    }
}
```

## Compilation and Execution

- The following library information should be added to your .profile file on

```

ACMEX
export FMHOME=/usr/local/lib/frame
export ORACLE_HOME=/usr/local/oracle/8.1.7
export ORACLE_SID=ccdb
export LD_LIBRARY_PATH=$ORACLE_HOME/lib

export PATH="$PATH:$WRECKDIR/bin/X11:/usr/java1.2/bin/"
export PATH="$PATH:$FMHOME/bin:"
export PATH="$PATH:$ORACLE_HOME/bin:"

# JDBC specific
export CLASSPATH=
"$ORACLE_HOME/jdbc:$ORACLE_HOME/jdbc/lib/classes12.zip:
$ORACLE_HOME/jdbc/lib/nls_charset12.zip:"
export LD_LIBRARY_PATH=
"$LD_LIBRARY_PATH:$LD_LIBRARY_PATH/jdbc/lib"

```

- Compile and run your java program  
`javac my_jdbc-program.java`  
`java my_jdbc-program`