JEE :: Introduction to JDBC

TalentSprint

Licensed To Skill

Version 1.0.4

Introduction to JDBC

The content in this presentation is aimed at teaching learners to:

- Identify the types of Drivers
- List the Pros and Cons of JDBC Drivers
- Establish connection using Type-4 Driver
- Retrive/Insert/Update the data in/form Database

Introduction to JDBC

The content in this presentation is aimed at teaching learners to:

- Use different type of Statements
- Use different type of ResultSet
- Perform Batch Operations
- Get the Meta data of the Database and ResultSet

Introduction to JDBC

The content in this presentation is aimed at teaching learners to:

- Understanding PreparedStatement
- Flow of PreparedStatement working
- Programming Interaction

Introduction to JDBC

JDBC (Java DataBase Connectivity)

- A sub set of CLI (Call Level Interface) specification.
- Can Create a platform-neutral interface between Java applications and Databases.
- Contains standard functions required to connect and perform SQL operation on the DB
- Communicates with ODBC , DB native libraries , java socket connection to DB

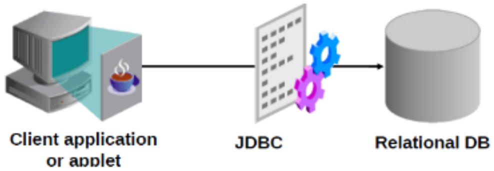
Introduction to JDBC

Why JDBC?

- To enable a java application to interact with aÂădatabase
- To provides a common base on which alternate interfaces and tools can be built

Introduction to JDBC

Why JDBC?



Introduction to JDBC

Driver Types

There are four types of JDBC drivers available in java for connecting to Database

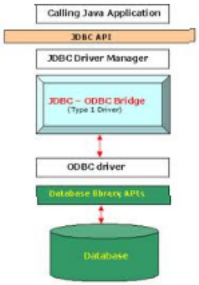
Type 1	JDBC-ODBC Bridge plus ODBC Driver
Type 2	Native API Partly Java Driver
Type 3	JDBC -Net pure Java Driver
Type 4	Native-Protocol Pure Java Driver

Introduction to JDBC

Type 1 - JDBC-ODBC Bridge plus ODBC Driver

- Translates the JDBC method calls into ODBC function calls.
- Included with the JDK in the sun.jdbc.odbc.JdbcOdbcDriver class.
- Not recommended for production use.

Introduction to JDBC



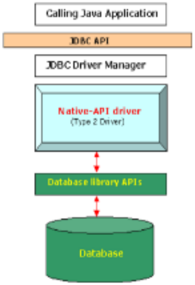
Introduction to JDBC

Type 2 - Native API Partly Java Driver

- Partly written in Java and partly in the native code. So, called Native API Partly Java Driver.
- Consists of drivers that communicate with databases servers in the serverâĂŹs native protocol.
- Implemented in a combination of binary code and Java.

Introduction to JDBC

- Installation is easier than installing both the JDBC-ODBC bridge and an ODBC driver.



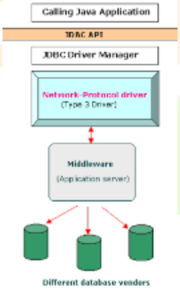
Introduction to JDBC

Type 3 - JDBC -Net pure Java Driver

- Communicate with a database access server using HTTP or SHTTP protocol and works for both the Internet and the Intranet.
- Translates the network protocol into a vendor specific database protocol
- Served from the web server are the best solution for the applets.

Introduction to JDBC

- Automatically installed on the userâĂŹs machine in a transparent manner.



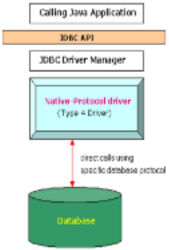
Introduction to JDBC

Type 4 - Native-Protocol Pure Java Driver

- A pure Java library that translates JDBC calls directly to a database-specific protocol.
- Written completely in Java and is hence platform independent.
- Installed inside the Java VirtualÂăMachine of the client.

Introduction to JDBC

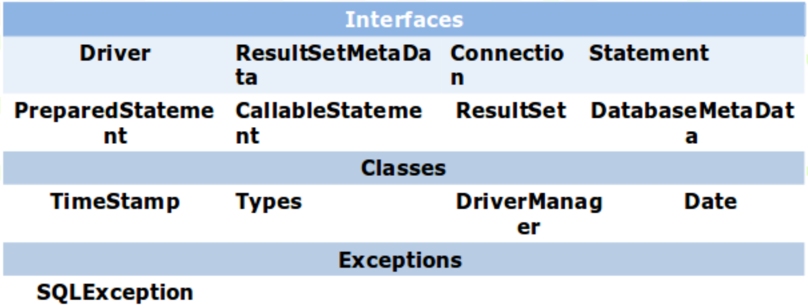
- Does not have the overhead ofÂăconversion of calls into ODBC or database API calls.
- Efficient for Intranet applications.



Introduction to JDBC

JDBC Specifications

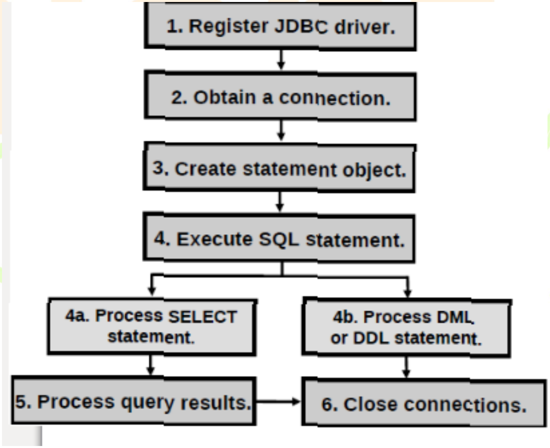
Consists of the following interfaces and classes:



Interfaces			
Driver	ResultSetMetaData	Connection	Statement
PreparedStatement	CallableStatement	ResultSet	DatabaseMetaData
Classes			
TimeStamp	Types	DriverManager	Date
Exceptions			
SQLException			

Introduction to JDBC

Steps to establish a connection



1. Register JDBC driver.
2. Obtain a connection.
3. Create statement object.
4. Execute SQL statement.
4a. Process SELECT statement.
4b. Process DML or DDL statement.
5. Process query results.
6. Close connections.

Introduction to JDBC

Steps to establish a connection

Step 1 - Load/Register JDBC Drivers

Before the driver manager can activate a driver, the driver must be registered manually by loading its class using the "Class" class **Example**

- JDBC-ODBC Bridge driver

  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

Introduction to JDBC

- Native-Protocol Pure Java Driver

  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

- Oracle

  Class.forName("oracle.jdbc.driver.OracleDriver");

- MySql

  Class.forName("com.mysql.jdbc.Driver");

Introduction to JDBC

Steps to establish a connection

Step 2 - Establish Connection using Driver Manager

- Select the database drivers
- Create a new database connection by calling the static method getConnection() of the DriverManager class

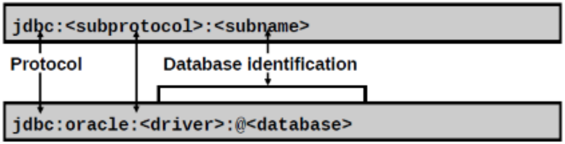
Introduction to JDBC

- This method takes
  - the database URL
  - a user name // Optional
  - password // Optional

Introduction to JDBC

JDBC URLs

- JDBC uses a URl-like string. The URL identifies.
- Database connection details, vary depending on the driver used.


jdbc:<subprotocol>:<subname>
Protocol
Database identification
jdbc:oracle:<driver>:@<database>

Introduction to JDBC

- jdbc:oracle:thin:@localhost:1521:orcl
  –For Oracle SE
- jdbc:oracle:thin:@localhost:1521:XE
  –For Oracle XE
- jdbc:mysql://localhost:3306/dbname
  –For MySql

Introduction to JDBC

Steps to establish a connection

Step 3 - Create the Statement

Create one of the following Statement object to send SQL statement to the database using Connection Object (con).

Statement executes a static SQL statement

```
Statement stmt= con.createStatement();
```

Introduction to JDBC

- PreparedStatement represents a precompiled SQL statement

  PreparedStatement ps=con.prepareStatement(
  query);

- CallableStatement executes SQL stored procedures

  CallableStatement cs=con.prepareCall(query);

Introduction to JDBC

Steps to establish a connection

Step 4 - Execute the Statement

The Statement interface provides three methods to execute SQL statements:

- Use executeQuery(String sql)for SELECT statements
  - Returns a ResultSet object for processing rows
- Use executeUpdate(String sql) for DML or DDL
  - Returns an int represents the row count for SQL Data Manipulation Language

Introduction to JDBC

- Use execute(String) for any SQL statement.
    - Returns a boolean value, such that
        - if the first result is aÂăResultSetÂăobject returns true
        - if it is an update count or there are no results returns false

Introduction to JDBC

Steps to establish a connection

Step 4a - Process SELECT Statement

Statement will returns the results of a query in a ResultSet object.

- Maintains a cursor pointing to its current row of data
- Provides following methods to retrieve column values
  - Use the next() method in loop to iterate through rows

Introduction to JDBC

- Use getXXX() methods to obtain column values by column position in query, or column name.

```
ResultSet rs=stmt.executeQuery("SELECT"+ "ename, empno
    FROM emp");
while(rs.next())
{
String s= rs.getString("ename");
int n = rs.getInt (" empno");
System.out.println(s+ "–" + n);
}
```

Introduction to JDBC

Steps to establish a connection

Step 4b - Submitting DDL / DML Statement

DDL Statement Create a table in a database from a JDBC program using the following lines of code:

```
String creatTable = “CREATE TABLE emp”+“(
    ename VARCHAR2(32),empno NUMBER)”;
stmt.executeUpdate(createTable);
```

Introduction to JDBC

DML Statement Insert values into a table in from a JDBC program using the following lines of code:

```
stmt.executeUpdate(“INSERT INTO emp”+ “
    VALUES(‘Ram’, ‘101’)”);
```

Introduction to JDBC

Steps to establish a connection

Step 5 - Closing Connection

- Explicitly close a Connection, Statement, and ResultSet object to release resources that are no longer needed.
- Protect the database from accidental changes.

```
rs.close() ;
stmt.close();
con.close();
```

Introduction to JDBC

Example 1 - An Example to create a Table in Database (DDL Statement)

```
void createDB() {
    try{
        String driver="oracle.jdbc.driver.OracleDriver";
        String url="jdbc:oracle:thin:@localhost:1521:orcl";
        Class.forName(driver);
        Connection con = DriverManager.getConnection(url,"
    scott","tiger");
```

Introduction to JDBC

```
        Statement stmt = con.createStatement();
        // Create the table Account Holder
        String query="CREATE TABLE AccountHolder(AcctNo
    INTEGER, Name VARCHAR(50), Address INTEGER
    VARCHAR(50), Balance FLOAT)";
        stmt.executeUpdate(query);
        // close the connection
        con.close();
    }
    catch(Exception ex) {
        System.out.println(ex.toString());
    }
}
```

Introduction to JDBC

Example 2 - An Example to insert data into a Table in Database (DML Statement)

```
void createDB() {
    try {
        String driver="oracle.jdbc.driver.OracleDriver";
        String url="jdbc:oracle:thin:@localhost:1521:orcl";
        Class.forName(driver);
        Connection con = DriverManager.getConnection(url,"
    scott","tiger");
        Statement stmt = con.createStatement();
```

Introduction to JDBC

```
        // Create the table Account Holder
        String query= "INSERT INTO AccountHolder VALUES
     (10015,'Asish','Commercial Street,Bangalore',1500000)";
        stmt.executeUpdate(query);
        // close the connection
        con.close();
     }
     catch(Exception ex) {
        System.out.println(ex.toString());
     }
  }
```

Introduction to JDBC

Example 3 - An Example to retirve data from Table in Database (SELECT Statement)

```
void createDB() {
    try {
        String driver="oracle.jdbc.driver.OracleDriver";
        String url="jdbc:oracle:thin:@localhost:1521:orcl";
        Class.forName(driver);
        Connection con = DriverManager.getConnection(url,"
    scott","tiger");
        Statement stmt = con.createStatement();
        // Select values into the Account Holder table
        ResultSet rs = stmt.executeQuery("SELECT * FROM
    ACCOUNTHOLDER");
```

Introduction to JDBC

```
        while(rs.next()) {
            System.out.println(" Acc No = "+rs.getString("
    AcctNo"));
            System.out.println(" Name = "+rs.getString("Name"))
    ;
            System.out.println(" Address = "+rs.getString("
    Address"));
            System.out.println(" Balance = "+rs.getString("
    Balance"));
        }
        rs.close() ;  // close the Result Set
        stmt.close();  // close the Statement
        con.close();  // close the connection
    }
    catch(Exception ex) {
        System.out.println(ex.toString());
    }
}
```

Introduction to JDBC

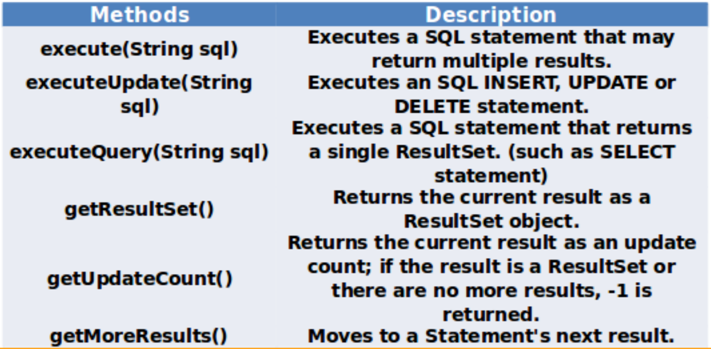
Executing SQL Statements

- To populate the database, update or delete the existing database information.
- Uses java.sql package
  - Statement Interface
  - PreparedStatement Interface
  - CallableStatement Interface

Introduction to JDBC

Statement Interface

Call createStatement() method of the Connection interface to create Statement Object



Methods	Description
execute(String sql)	Executes a SQL statement that may return multiple results.
executeUpdate(String sql)	Executes an SQL INSERT, UPDATE or DELETE statement.
executeQuery(String sql)	Executes a SQL statement that returns a single ResultSet. (such as SELECT statement)
getResultSet()	Returns the current result as a ResultSet object.
getUpdateCount()	Returns the current result as an update count; if the result is a ResultSet or there are no more results, -1 is returned.
getMoreResults()	Moves to a Statement's next result.

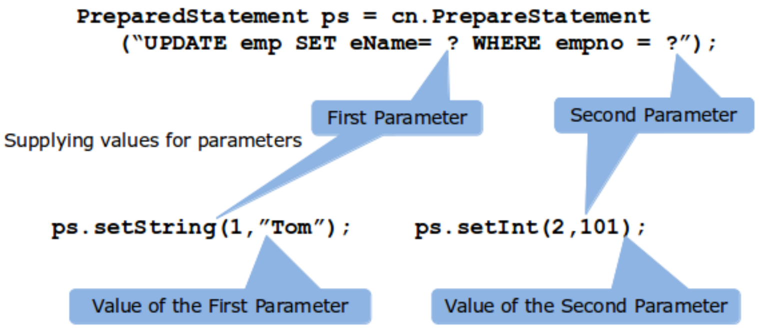
Introduction to JDBC

PreparedStatement Interface

- Extends the Statement interface
- Call prepareStatement() method of the Connection interface to create PreparedStatement Object
- Holds pre-compiled SQL statements
- Used to execute the SQL statement multiple times
- Enables us to retrieve, edit, or delete multiple records at a time.

Introduction to JDBC

PreparedStatement Interface



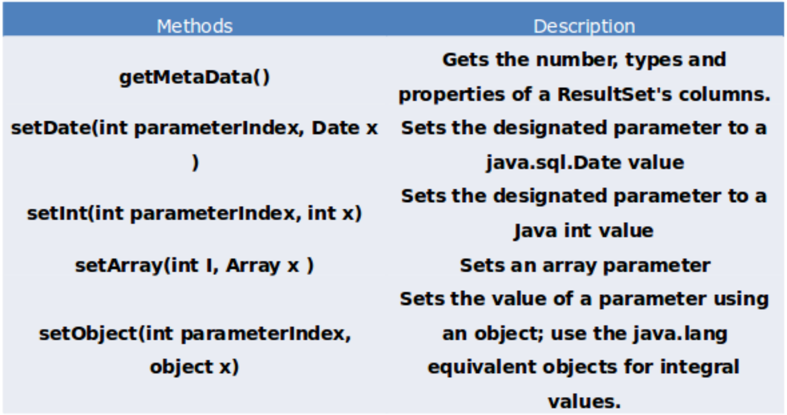
```
PreparedStatement ps = cn.PrepareStatement
   ("UPDATE emp SET eName= ? WHERE empno = ?");
```

Supplying values for parameters

```
ps.setString(1,"Tom");     ps.setInt(2,101);
```

Introduction to JDBC

PreparedStatement Interface



Methods	Description
getMetaData()	**Gets the number, types and properties of a ResultSet's columns.**
setDate(int parameterIndex, Date x)	**Sets the designated parameter to a java.sql.Date value**
setInt(int parameterIndex, int x)	**Sets the designated parameter to a Java int value**
setArray(int I, Array x)	**Sets an array parameter**
setObject(int parameterIndex, object x)	**Sets the value of a parameter using an object; use the java.lang equivalent objects for integral values.**

Introduction to JDBC

PreparedStatement Example

```
Statement stmt = con.createStatement();
String query="UPDATE emp SET eName =? WHERE empno =?
    ";
PreparedStatement ps=con.prepareStatement(query);
//Here 1 and 2 are the sequential number of values to be set.
ps.setString(1, "Tom");
ps.setInt(2,3);
ps.executeUpdate();
ResultSet rs = stmt.executeQuery("SELECT * FROM emp");
```

Introduction to JDBC

CallableStatement Interface

- Extends the PreparedStatement interface
- Have a call to a stored procedure
- May take IN parameters, OUT Parameters, INOUT parameters.
- Calling a stored procedure

Introduction to JDBC

- with no parameters

  {call procedure_name}

- Have a call to a stored procedure

  { call procedure_name [(?, ?, ?)]}

- May take IN parameters, OUT Parameters, INOUT parameters.

  {? = call procedure_name [(?, ?, ?,)]}

Introduction to JDBC

CallableStatement Interface

- To create a CallableStatement object.

  CallableStatement cstmt=cn.prepareCall(âĂIJ{call
      procedure_name}âĂİ);

- Passing IN parameters is done using setXXX() methods
- Each OUT parameter must be registered in a log file using registerOutParameter() method
- getXXX() functions OUT parameters are read into application

Introduction to JDBC

CallableStatement Interface

CallableStatement ct=cn.prepareCall(âĂIJ{call getTestData (?,?.?)}âĂİ);
ct.registerOutParameter(1,java.sql.Types.TINYINT); *// OUT*
ct.setFloat(2,1.34);*// IN*
ct.registerOutParameter(3,java.sql.Types.VARCHAR); *// INOUT*
ct.setString(3, âĂIJImranâĂİ); *// INOUT*
ct.executeQuery();
byte x=ct.getBytes(1); *// Getting IN*
String s=ct.getString(3); *// Getting INOUT*

Introduction to JDBC

JDBC Exception Handling

Common exception classes used in the JDBC API:

- java.sql.SQLException
  - Database access error or other errors
- java.sql.BatchUpdateException
  - Subclass ofÂăSQLExceptionÂă
  - An error occurs during a batch update operation

Introduction to JDBC

- java.sql.DataTruncation
  - A data values is unexpectedly truncated for some reasons
- java.sql.SQLWarning
  - database access warnings Âămay be retrieved from Connection, Statement and ResultSetÂăobjects

Introduction to JDBC

Batch Update Facility

- What is a batch update?
  - A set of multiple update statements, submitted to the database as a unit.
- Why?
  - It is more efficient to send multiple update.JDBC 2.0 API provides this batch update facility.

Introduction to JDBC

- How?

  addBatch() of Statement, to add a update statement to Batch.

  executeBatch() of Statement, submits a batch of commands to the database for execution.

Introduction to JDBC

Sample Batch Operation

```
Statement stmt = con.createStatement();
stmt.addBatch("INSERT INTO COFFEES VALUES('Amaretto',
    49, 9.99, 0, 0)");
stmt.addBatch("INSERT INTO COFFEES VALUES('Hazelnut',
    49, 9.99, 0, 0)");
stmt.addBatch("INSERT INTO COFFEES VALUES('
    Amaretto_decaf', 49, 10.99, 0, 0)");
stmt.addBatch("INSERT INTO COFFEES VALUES('
    Hazelnut_decaf', 49, 10.99, 0, 0)");
int [] updateCounts = stmt.executeBatch();
```

Introduction to JDBC

Meta data

- Data about a data is called metadata.
- Provides information about the Database/ResultSet.

Introduction to JDBC

- Two Interfaces provide following information:

DatabasesMetaData	Comprehensive information about the database as a whole.
ResultSetMetaData	Information about the types and properties of the columns in a ResultSet object.

Introduction to JDBC

DatabaseMetaData

```
DatabaseMetaData dbmd=
con.getMetaData();
```

Method's Description:

getDatabaseProductName() Returns the name of the database product.

Introduction to JDBC

getDatabaseProductVersion() Returns the version of the database product.

getUserName() Returns our user name as known tothe database.

getDriverName() Returns the name of the JDBC driver.

Introduction to JDBC

getDriverVersion() Returns the version of the JDBC driver.

getImportedKeys **(String catalog, String schema, String table)** Gets a description of the primary key columns that are referenced by a table's foreign key columns (the primary keys imported by a table).

Introduction to JDBC

ResultSetMetaData

ResultSetMetaData rsmd= rs.getMetaData();

Method's Description:

getColumnCount() Returns the number ofc olumns in this ResultSet.

Introduction to JDBC

getColumnName(int columnName) Gets a column's name.

getCoulmnType(int columnName) Retrieves a column's SQL type.

getTableName(int columnName) Gets a column's table name.

isCurrency(int columnName) Indicates whether the column is a cash value.

Introduction to JDBC

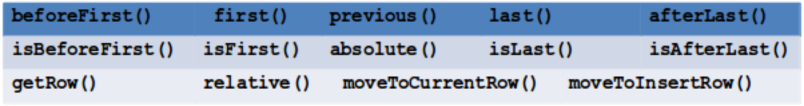
Scrollable ResultSets

- ResultSets have been used in a sequential manner using ResultSet.next()

Introduction to JDBC

- A new method in JDBC allowed to create scrollable and /or update the ResultSets.

  createStatement(**int** resultSetType,**int**
      resultSetConcurrency)



beforeFirst()	first()	previous()	last()	afterLast()
isBeforeFirst()	isFirst()	absolute()	isLast()	isAfterLast()
getRow()	relative()	moveToCurrentRow()	moveToInsertRow()	

Introduction to JDBC

Scrollable ResultSets

ResultSet Types

- TYPE_FORWARD_ONLY
  - This is the default type which allows only forward movement and columns can be generally read only once.
- TYPE_SCROLL_INSENSITIVE
  - Cursor is allowed to move backwards, forwards, and at random
  - Changes are invisible which indicates that the ResultSet is insensitive.

Introduction to JDBC

- TYPE_SCROLL_INSENSITIVE
  - Move backwards, forwards, and at random
  - Allows a dynamic view of data and changes are visible which also means, the ResultSet is sensitive

Introduction to JDBC