

Unit I – Part 2

JDBC API

Handout material
BCA 5th Sem.
2018-19

Prepared By: Dr. Sharon V. Mohtra
Assistant Professor,
Dept. of Computer Applications,
Christ College,
Rajkot



Content

- Introduction of JDBC
- Need for JDBC
- JDBC Architecture
- Types of JDBC Drivers
- JDBC API for Database Connectivity (java.sql package)
 - Connection Interface
 - Statement Interface
 - PreparedStatement Interface
 - CallableStatement Interface
 - ResultSet Interface
 - ResultSetMetaData Interface
 - DatabaseMetaData Interface
- Data types in JDBC
- Processing Queries
- Database Exception Handling
- Connecting with Databases (MySQL, Access, Oracle)

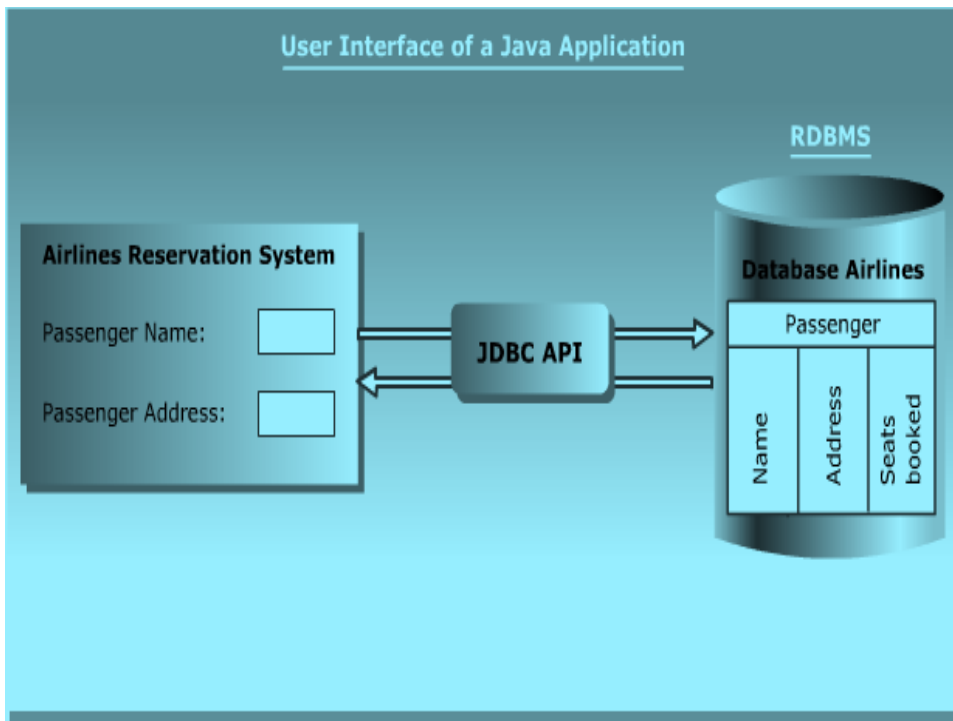


INTRODUCTION AND NEED FOR JDBC



Introduction

- **JDBC provides a standard library for accessing relational databases**
- API standardizes
 - Way to establish connection to database
 - Approach to initiating queries
 - Method to create stored (parameterized) queries
 - The data structure of query result (table)
- Determining the number of columns
- Looking up metadata, etc.
- API does *not* standardize SQL syntax
 - JDBC is not embedded SQL
- JDBC class located in java.sql package
- **Note: JDBC is not officially an acronym; unofficially, “Java Database Connectivity” is commonly used**



What is JDBC?

- Java Database Connectivity (JDBC) is an industry **Java standard** that provides the **interface (database-independent connectivity)** for connecting from **Java** to **relational databases** and other tabular data sources, such as spreadsheets or flat files.
- The JDBC standard is defined by **Sun Microsystems** and implemented through the standard `java.sql` interfaces. This allows individual providers to implement and extend the standard with their own JDBC drivers.
- JDBC is based on the X/Open SQL Call Level Interface (CLI) (embeds SQL code into host program).
- JDBC 4.0 complies with the SQL 2003 standard.
- JDBC 4.1, is specified by a maintenance release 1 of JSR 221 and is included in Java SE 7. The latest version, JDBC 4.2, is specified by a maintenance release 2 of JSR 221 and is included in Java SE 8.



Need for JDBC

- Before APIs like JDBC and ODBC, database connectivity was tedious
 - Each database vendor provided a function library for accessing their database
 - The connectivity library was proprietary.
 - If the database vendor changed for the application, the data access portions had to be rewritten
 - If the application was poorly structured, rewriting its data access might involve rewriting the majority of the application
 - The costs incurred generally meant that application developers were stuck with a particular database product for a given application



Versions

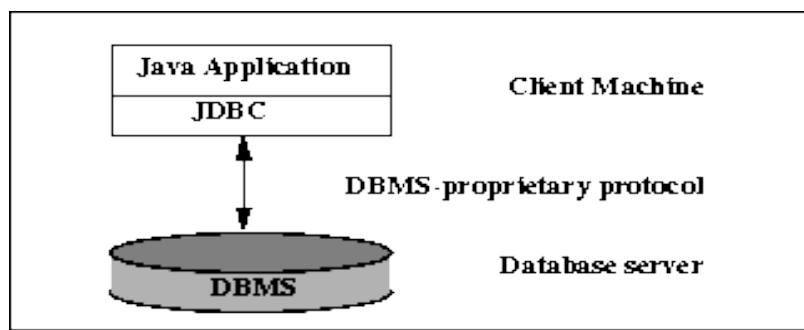
- Sun Microsystems released JDBC as part of JDK 1.1 on February 19, 1997 (Java Standard Edition).
- Starting with version 3.1, JDBC has been developed under the Java Community Process.
- **JSR 54** => JDBC 3.0 (included in J2SE 1.4),
- **JSR 114** => the JDBC Rowset additions,
- **JSR 221** => JDBC 4.0 (in Java SE 6),
- **JSR 221** => JDBC 4.1 (in Java SE 7),
- **JSR 221** => JDBC 4.2 (in Java SE 8),
- The latest version, is **JDBC 4.3**, is specified by a maintenance release 3 of **JSR 221** and is included in **Java SE 9**.

JDBC ARCHITECTURE

JDBC Architecture: Two Tier

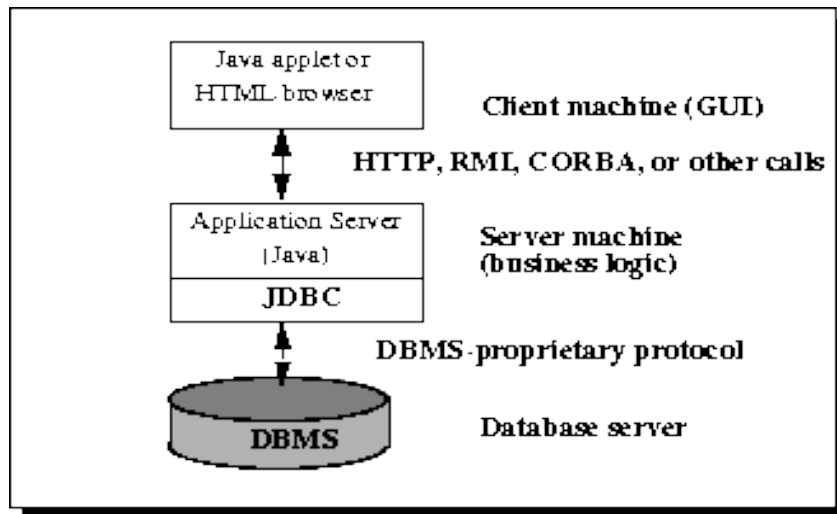
- The JDBC API supports both two-tier and three-tier processing models for database access.

Two-tier Architecture for Data Access



JDBC Architecture: Three Tier

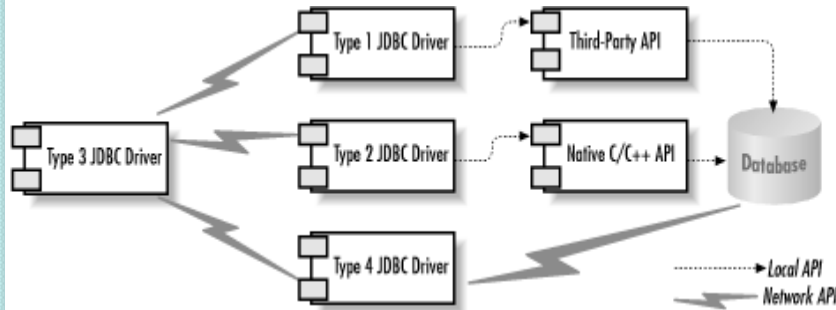
Three-tier Architecture for Data Access



TYPES OF JDBC DRIVERS

JDBC Drivers

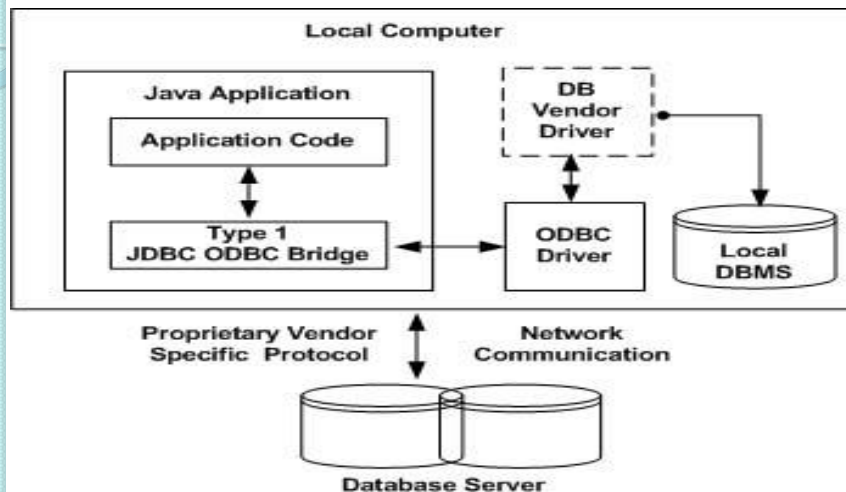
- JDBC technology allows you to use the Java programming language to exploit "Write Once, Run Anywhere" capabilities for applications that require access to enterprise data. With a JDBC technology-enabled driver, you can connect all corporate data even in a heterogeneous environment.
- Types are:
 - Type 1 : **JDBC bridge driver**
 - Type 2 : **Native API (part Java driver)**
 - Type 3 : **Network protocol (Middleware driver)**
 - Type 4 : **Native protocol (pure Java driver)**



Type 1: JDBC bridge driver

- This type uses bridge technology to connect a Java client to a third-party API such as Oracle DataBase Connectivity (ODBC).
- Sun's JDBC-ODBC bridge is an example of a Type 1 driver. These drivers are implemented using native code.
- In a Type 1 driver, a JDBC bridge is used to access ODBC drivers installed on each client machine. Using ODBC requires configuring on your system a Data Source Name (DSN) that represents the target database.
- Initially this was a useful driver because most databases only supported ODBC access but now this type of driver is recommended only for experimental use or when no other alternative is available.
- The JDBC-ODBC bridge that comes with JDK 1.2 is a good example of this kind of driver.

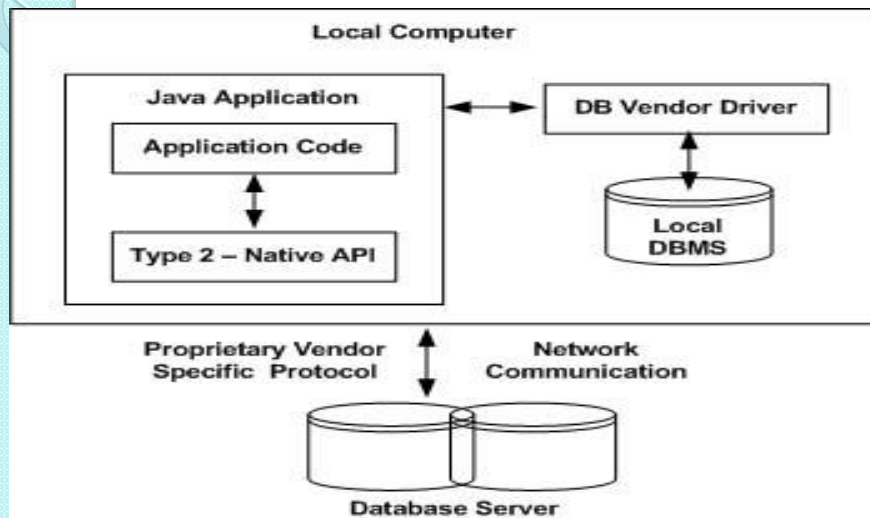
Type I



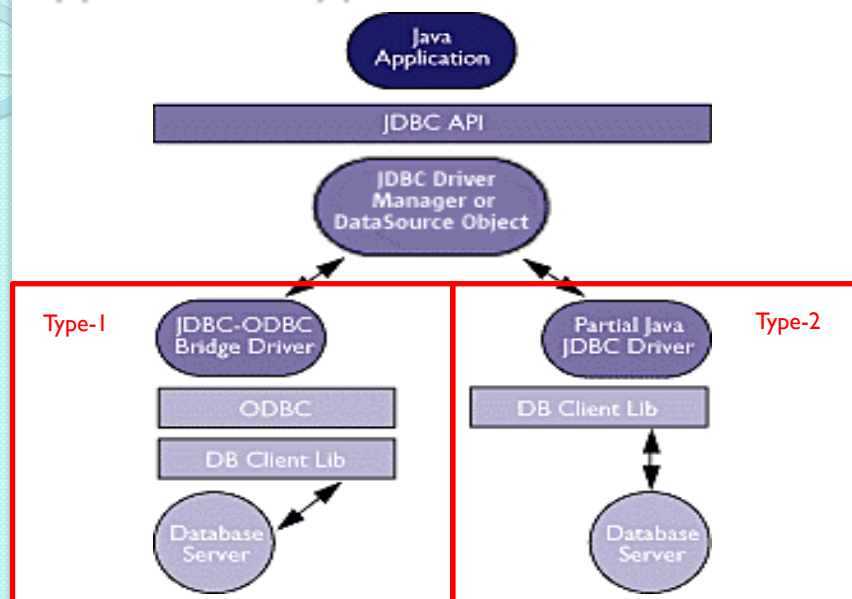
Type 2: Native API (part Java driver)

- This type of driver wraps a native API with Java classes. The Oracle Call Interface (OCI) driver is an example of a Type 2 driver. Because a Type 2 driver is implemented using local native code, it is expected to have better performance than a pure Java driver.
- In a Type 2 driver, JDBC API calls are converted into native C/C++ API calls which are unique to the database. These drivers typically provided by the database vendors and used in the same manner as the JDBC-ODBC Bridge, the vendor-specific driver must be installed on each client machine.
- If we change the Database we have to change the native API as it is specific to a database and they are mostly obsolete now but you may realize some speed increase with a Type 2 driver, because it eliminates ODBC's overhead.
- The Oracle Call Interface (OCI) driver is an example of a Type 2 driver.

Type 2



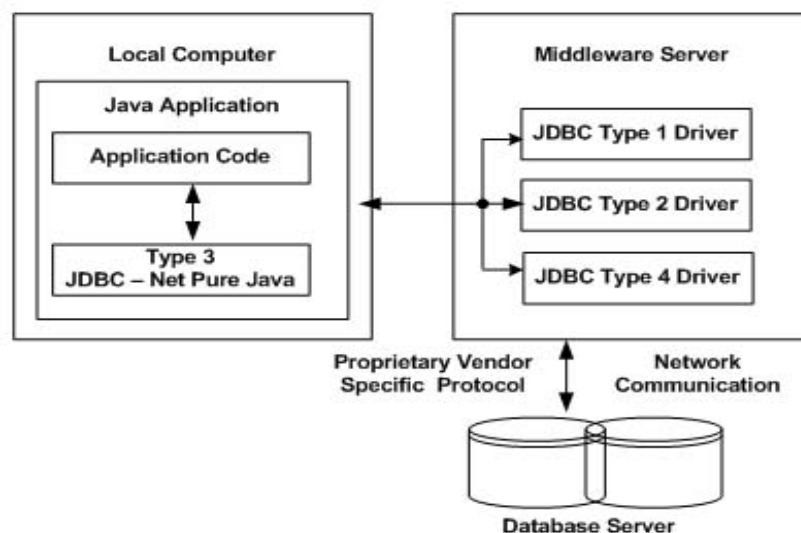
Type 1 and Type 2 drivers



Type 3: Network protocol (pure Java driver)

- This type of driver communicates using a network protocol to a middle-tier server. The middle tier in turn communicates to the database. Oracle does not provide a Type 3 driver. They do, however, have a program called Connection Manager that, when used in combination with Oracle's Type 4 driver, acts as a Type 3 driver in many respects.
- In a Type 3 driver, a three-tier approach is used to accessing databases. The JDBC clients use standard network sockets to communicate with an middleware application server. The socket information is then translated by the middleware application server into the call format required by the DBMS, and forwarded to the database server.
- This kind of driver is extremely flexible, since it requires no code installed on the client and a single driver can actually provide access to multiple databases.
- You can think of the application server as a JDBC "proxy," meaning that it makes calls for the client application. As a result, you need some knowledge of the application server's configuration in order to effectively use this driver type.
- Your application server might use a Type 1, 2, or 4 driver to communicate with the database, understanding the nuances will prove helpful.

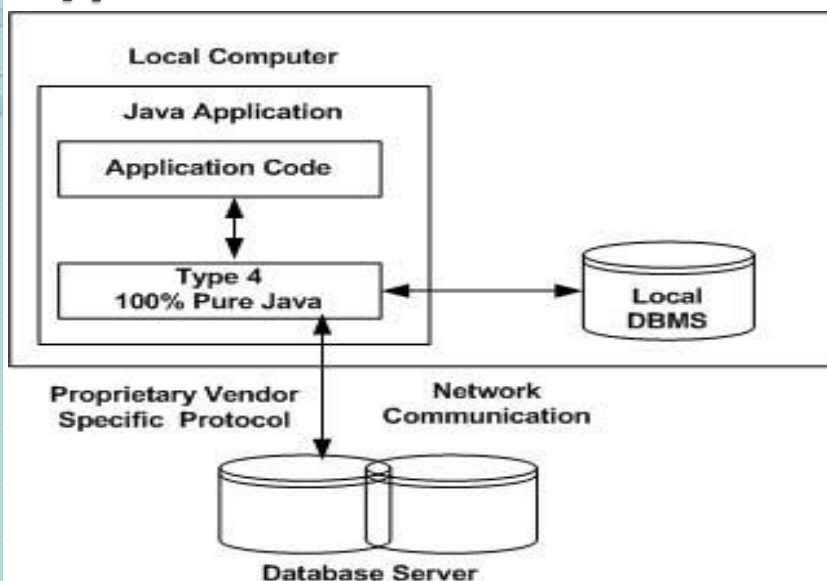
Type 3



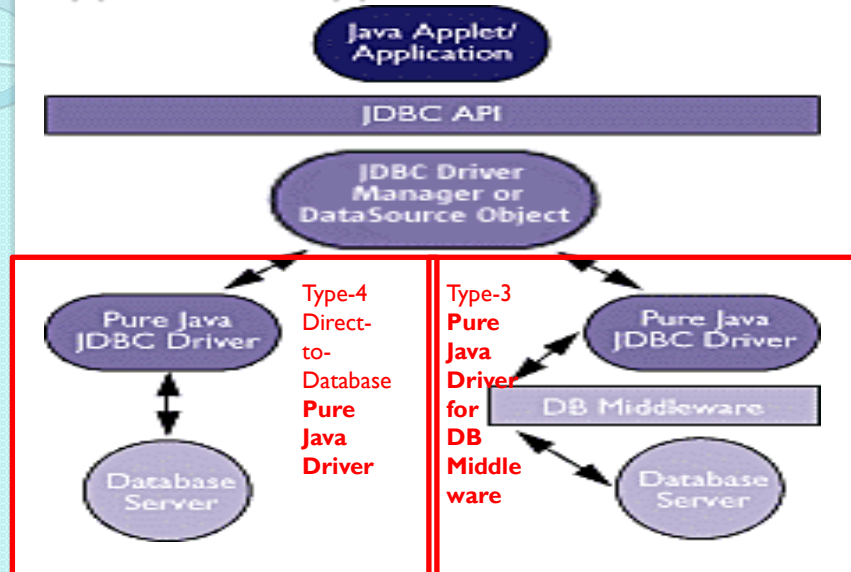
Type 4: Native protocol (pure Java driver)

- This type of driver, written entirely in Java, communicates directly with the database. No local native code is required. Oracle's thin driver is an example of a Type 4 driver.
- In a Type 4 driver, it is a pure Java-based driver that communicates directly with vendor's database through socket connection. This is the highest performance driver available for the database and is usually provided by the vendor itself.
- This kind of driver is extremely flexible; you don't need to install special software on the client or server. Further, these drivers can be downloaded dynamically.
- MySQL's Connector/J driver is a Type 4 driver.

Type 4



Type 3 and Type 4 drivers



JDBC API

- The JDBC API provides a call-level API for SQL-based database access.
- The JDBC classes are contained in the Java package `java.sql` and `javax.sql`.

Classes and Interfaces

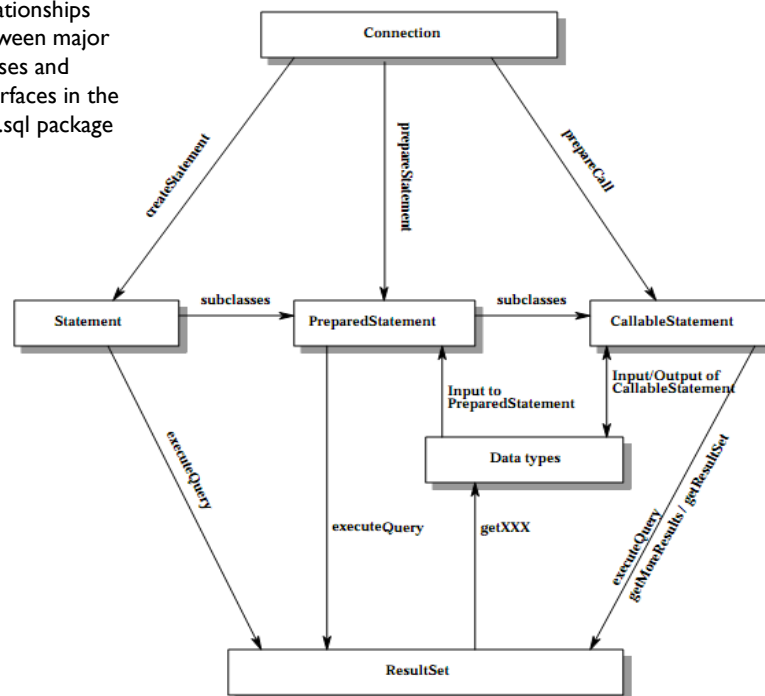
- The following classes and interfaces make up the JDBC API.
- **The java.sql Package**
- The core JDBC API is contained in the package java.sql.
- The classes and interface in java.sql are listed below. Classes are in bold type; interfaces are in standard type.

- java.sql.Array
- **java.sql.BatchUpdateException**
- java.sql.Blob
- java.sql.CallableStatement
- java.sql.Clob
- java.sql.Connection
- **java.sql.DataTruncation**
- java.sql.DatabaseMetaData
- **java.sql.Date**
- java.sql.Driver
- **java.sql.DriverManager**
- **java.sql.DriverPropertyInfo**
- java.sql.ParameterMetaData
- java.sql.PreparedStatement
- java.sql.Ref
- java.sql.ResultSet
- java.sql.ResultSetMetaData
- java.sql.Savepoint
- java.sql.SQLData
- **java.sql.SQLException**
- java.sql.SQLInput
- java.sql.SQLOutput
- java.sql.SQLPermission
- **java.sql.SQLWarning**
- java.sql.Statement
- java.sql.Struct
- **java.sql.Time**
- **java.sql.Timestamp**
- **java.sql.Types**

JDBC 3.0 API

- java.sql.Blob
- java.sql.CallableStatement
- java.sql.Clob
- java.sql.Connection
- java.sql.DatabaseMetaData
- **java.sql.ParameterMetaData**
- java.sql.PreparedStatement
- java.sql.Ref
- java.sql.ResultSet
- **java.sql.Savepoint**
- java.sql.SQLInput
- java.sql.SQLOutput
- java.sql.Statement
- java.sql.Types


Relationships between major classes and interfaces in the java.sql package





Using JDBC API

- The JDBC API classes and interfaces are available in the `java.sql` and the `javax.sql` packages.
- The commonly used classes and interfaces in the JDBC API are:
 - `DriverManager` class: Loads the driver for a database.
 - `Driver` interface: Represents a database driver. All JDBC driver classes must implement the `Driver` interface.
 - `Connection` interface: Enables you to establish a connection between a Java application and a database.

- 
- `Statement`: It executes SQL statements for particular connection and retrieve the results
 - `PreparedStatement`: It allows the programmer to create prepared SQL statements
 - `CallableStatement`: It executes stored procedures
 - `ResultSet` interface: Represents the information retrieved from a database.
 - `SQLException` class: Provides information about the *exceptions* that occur while interacting with databases.




Connection interface

- The Connection interface used to connect java application with particular database.
- After creating the connection with database we can execute SQL statements for that particular connection using object of Connection and retrieve the results.
- The interface has few methods that makes changes to the database temporary or permanently.
- The some methods are as given as follows:



Methods

- **void close()**
- This method frees an object of type Connection from database and other JDBC resources.
- **void commit()**
- This method makes all the changes made since the last commit or rollback permanent. It throws SQLException.
- **Statement createStatement()**
- This method creates an object of type Statement for sending SQL statements to the database. It throws SQLException.
- **boolean isClosed()**
- Return true if the connection is close else return false.

- 
- **CallableStatement prepareCall(String s)**
 - This method creates an object of type `CallableStatement` for calling the stored procedures from database. It throws `SQLException`.
 - **PreparedStatement prepareStatement(String s)**
 - This method creates an object of type `PreparedStatement` for sending dynamic (with or without IN parameter) SQL statements to the database. It throws `SQLException`.
 - **void rollback()**
 - This method undoes all changes made to the database.




Statement Interface

- The Statement interface is used for to execute a static query.
- It's a very simple and easy so it also calls a "**Simple Statement**".
- The statement interface has several methods for execute the SQL statements and also get the appropriate result as per the query sent to the database.
- Some of the most common methods are as follows:



Methods

- **void close()**
- This method frees an object of type Statement from database and other JDBC resources.
- **boolean execute(String s)**
- This method executes the SQL statement specified by s. The `getResultSet()` method is used to retrieve the result.
- **ResultSet getResultSet()**
- This method retrieves the ResultSet that is generated by the `execute()` method.

- 
- **ResultSet executeQuery(String s)**
 - This method is used to execute the SQL statement specified by s and returns the object of type ResultSet.
 - **int getMaxRows()**
 - This method returns the maximum number of rows those are generated by the `executeQuery()` method.
 - **int executeUpdate(String s)**
 - This method executes the SQL statement specified by s. The SQL statement may be a SQL insert, update and delete statement.



The Prepared Statement Interface

- The Prepared Statement interface is used to execute a dynamic query (parameterized SQL statement) with IN parameter.
- Subinterface of Statement interface.
- Improves performance
- Compiled only once
- Faster execution



Methods

- **void close()**
 - This method frees an object of type Prepared Statement from database and other JDBC resources.
- **boolean execute()**
 - This method executes the dynamic query in the object of type **Prepared Statement**. The **getResult()** method is used to retrieve the result.
- **ResultSet executeQuery()**
 - This method is used to execute the dynamic query in the object of type **Prepared Statement** and returns the object of type **ResultSet**.

- **int executeUpdate()**
 - This method executes the SQL statement in the object of type **PreparedStatement**. The SQL statement may be a SQL insert, update and delete statement.
- **ResultSetMetaData getMetaData()**
 - The ResultSetMetaData means a data about the data of ResultSet. This method retrieves an object of type ResultSetMetaData that contains information about the columns of the ResultSet object that will be return when a query is executed.
- **int getMaxRows()**
 - This method returns the maximum number of rows those are generated by the executeQuery() method.

Example

```
PreparedStatement pstmt = null;  
try { String SQL = "Update Employees SET  
    age = ? WHERE id = ?";  
    pstmt = conn.prepareStatement(SQL); ... }  
catch (SQLException e) { ... }  
finally { ...  
    pstmt.close(); }
```



The Callable Statement Interface

- It extends `PreparedStatement`, used to execute SQL stored procedures.
- JDBC provides a stored procedure SQL escape that allows stored procedures to be called in a standard way for all RDBMS's. This escape syntax has one form that includes a result parameter and one that does not. If used, the result parameter must be registered as an OUT parameter. The other parameters may be used for input, output or both. Parameters are referred to sequentially, by number. The first parameter is 1.
- `{?= call [, ...]}`
`{call [, ...]}`



Callable - II

- IN parameter values are set using the set methods inherited from `PreparedStatement`. The type of all OUT parameters must be registered prior to executing the stored procedure; their values are retrieved after execution via the get methods provided.
- A Callable statement may return a `ResultSet` or multiple `ResultSets`. Multiple `ResultSets` are handled using operations inherited from `Statement`.
- For maximum portability, a call's `ResultSets` and update counts should be processed prior to getting the values of output parameters.




Parameter Description


- **IN**
 - A parameter whose value is unknown when the SQL statement is created. You bind values to IN parameters with the `setXXX()` methods.
- **OUT**
 - A parameter whose value is supplied by the SQL statement it returns. You retrieve values from the OUT parameters with the `getXXX()` methods.
- **INOUT**
 - A parameter that provides both input and output values. You bind variables with the `setXXX()` methods and retrieve values with the `getXXX()` methods.



Methods

- **getBigDecimal**(int paraIndex, int sqltype) Get the value of a NUMERIC parameter as a `java.math.BigDecimal` object.
- **getBoolean**(int) Get the value of a BIT parameter as a Java boolean.
- **getByte**(int) Get the value of a TINYINT parameter as a Java byte.
- **getBytes**(int) Get the value of a SQL BINARY or VARBINARY parameter as a `Java byte[]`
- **getDate**(int) Get the value of a SQL DATE parameter as a `java.sql.Date` object
- **getDouble**(int) Get the value of a DOUBLE parameter as a Java double.
- **getFloat**(int) Get the value of a FLOAT parameter as a Java float.
- **getInt**(int) Get the value of an INTEGER parameter as a Java int.

- 
- **getLong(int)** Get the value of a BIGINT parameter as a Java long.
 - **getObject(int)** Get the value of a parameter as a Java object.
 - **getShort(int)** Get the value of a SMALLINT parameter as a Java short.
 - **getString(int)** Get the value of a CHAR, VARCHAR, or LONGVARCHAR parameter as a Java String
 - **getTime(int)** Get the value of a SQL TIME parameter as a java.sql.Time object.
 - **getTimestamp(int)** Get the value of a SQL TIMESTAMP parameter as a java.sql.Timestamp object.

- 
- **registerOutParameter(int parameterIndex, int sqlType)** Before executing a stored procedure call, you must explicitly call registerOutParameter to register the java.sql.Type of each out parameter.
 - **registerOutParameter(int parameterIndex, int sqlType, int scale)** Use this version of registerOutParameter for registering Numeric or Decimal out parameters.
 - **wasNull()** An OUT parameter may have the value of SQL NULL; wasNull reports whether the last value read has this special value.



Example

```
CallableStatement cstmt = null;
try { String SQL = "{call getEmpName (?,
    ?)}";
    cstmt = conn.prepareCall (SQL); ... } catch
    (SQLException e) { ... }
finally { cstmt.close(); }
```



° RESULTSET INTERFACE



ResultSet Interface

- The ResultSet objects contain the results that are returned after the execution of SQL statement from the database.
- The ResultSet object maintains a cursor that points to the current row of results.
- It has the following common methods to get results from the ResultSet object.



Example

- `Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);`
- `ResultSet rs = stmt.executeQuery("SELECT a, b FROM TABLE2");`
- `// rs will be scrollable, will not show changes made by others, // and will be updatable`

Fields

Modifier and Type	Field and Description
static int	CLOSE_CURSORS_AT_COMMIT The constant indicating that open ResultSet objects with this holdability will be closed when the current transaction is committed.
static int	CONCUR_READ_ONLY The constant indicating the concurrency mode for a ResultSet object that may NOT be updated.
static int	CONCUR_UPDATABLE The constant indicating the concurrency mode for a ResultSet object that may be updated.
static int	FETCH_FORWARD The constant indicating that the rows in a result set will be processed in a forward direction; first-to-last.
static int	FETCH_REVERSE The constant indicating that the rows in a result set will be processed in a reverse direction; last-to-first.
static int	FETCH_UNKNOWN The constant indicating that the order in which rows in a result set will be processed is unknown.
static int	HOLD_CURSORS_OVER_COMMIT The constant indicating that open ResultSet objects with this holdability will remain open when the current transaction is committed.
static int	TYPE_FORWARD_ONLY The constant indicating the type for a ResultSet object whose cursor may move only forward.
static int	TYPE_SCROLL_INSENSITIVE The constant indicating the type for a ResultSet object that is scrollable but generally not sensitive to changes to the data that underlies the ResultSet.
static int	TYPE_SCROLL_SENSITIVE The constant indicating the type for a ResultSet object that is scrollable and generally sensitive to changes to the data that underlies the ResultSet.

Method

Modifier and Type	Method and Description
boolean	absolute (int row)Moves the cursor to the given row number in this ResultSet object.
void	afterLast ()Moves the cursor to the end of this ResultSet object, just after the last row.
void	beforeFirst ()Moves the cursor to the front of this ResultSet object, just before the first row.
void	cancelRowUpdates ()Cancels the updates made to the current row in this ResultSet object.
void	clearWarnings ()Clears all warnings reported on this ResultSet object.
void	close ()Releases this ResultSet object's database and JDBC resources immediately instead of waiting for this to happen when it is automatically closed.
void	deleteRow ()Deletes the current row from this ResultSet object and from the underlying database.
int	findColumn (String columnLabel)Maps the given ResultSet column label to its ResultSet column index.
boolean	first ()Moves the cursor to the first row in this ResultSet object.
Array	getArray (int columnIndex)Retrieves the value of the designated column in the current row of this ResultSet object as an Array object in the Java programming language.
Array	getArray (String columnLabel)Retrieves the value of the designated column in the current row of this ResultSet object as an Array object in the Java programming language.

InputStream	getAsciiStream (int columnIndex)Retrieves the value of the designated column in the current row of this ResultSet object as a stream of ASC II characters.
InputStream	getAsciiStream (String columnLabel)Retrieves the value of the designated column in the current row of this ResultSet object as a stream of ASCII characters.
BigDecimal	getBigDecimal (int columnIndex)Retrieves the value of the designated column in the current row of this ResultSet object as a java.math.BigDecimal with full precision.
BigDecimal	getBigDecimal (int columnIndex, int scale) Deprecated.
BigDecimal	getBigDecimal (String columnLabel)Retrieves the value of the designated column in the current row of this ResultSet object as a java.math.BigDecimal with full precision.
BigDecimal	getBigDecimal (String columnLabel, int scale) Deprecated.
InputStream	getBinaryStream (int columnIndex)Retrieves the value of the designated column in the current row of this ResultSet object as a stream of uninterpreted bytes.
InputStream	getBinaryStream (String columnLabel)Retrieves the value of the designated column in the current row of this ResultSet object as a stream of uninterpreted bytes.
Blob	getBlob (int columnIndex)Retrieves the value of the designated column in the current row of this ResultSet object as a Blob object in the Java programming language.
Blob	getBlob (String columnLabel)Retrieves the value of the designated column in the current row of this ResultSet object as a Blob object in the Java programming language.
boolean	getBoolean (int columnIndex)Retrieves the value of the designated column in the current row of this ResultSet object as a boolean in the Java programming language.
boolean	getBoolean (String columnLabel)Retrieves the value of the designated column in the current row of this ResultSet object as a boolean in the Java programming language.
byte	getByte (int columnIndex)Retrieves the value of the designated column in the current row of this ResultSet object as a byte in the Java programming language.
byte	getByte (String columnLabel)Retrieves the value of the designated column in the current row of this ResultSet object as a byte in the Java programming language.
byte[]	getBytes (int columnIndex)Retrieves the value of the designated column in the current row of this ResultSet object as a byte array in the Java programming language.
byte[]	getBytes (String columnLabel)Retrieves the value of the designated column in the current row of this ResultSet object as a byte array in the Java programming language.

Reader	getCharacterStream (int columnIndex)Retrieves the value of the designated column in the current row of this ResultSet object as a java.io.Reader object.
Reader	getCharacterStream (String columnLabel)Retrieves the value of the designated column in the current row of this ResultSet object as a java.io.Reader object.
Clob	getClob (int columnIndex)Retrieves the value of the designated column in the current row of this ResultSet object as a Clob object in the Java programming language.
Clob	getClob (String columnLabel)Retrieves the value of the designated column in the current row of this ResultSet object as a Clob object in the Java programming language.
int	getConcurrency ()Retrieves the concurrency mode of this ResultSet object.
String	getCursorName ()Retrieves the name of the SQL cursor used by this ResultSet object.
Date	getDate (int columnIndex)Retrieves the value of the designated column in the current row of this ResultSet object as a java.sql.Date object in the Java programming language.
Date	getDate (int columnIndex, Calendar cal)Retrieves the value of the designated column in the current row of this ResultSet object as a java.sql.Date object in the Java programming language.
Date	getDate (String columnLabel)Retrieves the value of the designated column in the current row of this ResultSet object as a java.sql.Date object in the Java programming language.
Date	getDate (String columnLabel, Calendar cal)Retrieves the value of the designated column in the current row of this ResultSet object as a java.sql.Date object in the Java programming language.
double	getDouble (int columnIndex)Retrieves the value of the designated column in the current row of this ResultSet object as a double in the Java programming language.
double	getDouble (String columnLabel)Retrieves the value of the designated column in the current row of this ResultSet object as a double in the Java programming language.
int	getFetchDirection ()Retrieves the fetch direction for this ResultSet object.
int	getFetchSize ()Retrieves the fetch size for this ResultSet object.
float	getFloat (int columnIndex)Retrieves the value of the designated column in the current row of this ResultSet object as a float in the Java programming language.
float	getFloat (String columnLabel)Retrieves the value of the designated column in the current row of this ResultSet object as a float in the Java programming language.
int	getHoldability ()Retrieves the holdability of this ResultSet object

int	getInt (int columnIndex)Retrieves the value of the designated column in the current row of this ResultSet object as an int in the Java programming language.
int	getInt (String columnLabel)Retrieves the value of the designated column in the current row of this ResultSet object as an int in the Java programming language.
long	getLong (int columnIndex)Retrieves the value of the designated column in the current row of this ResultSet object as a long in the Java programming language.
long	getLong (String columnLabel)Retrieves the value of the designated column in the current row of this ResultSet object as a long in the Java programming language.
ResultSetMetaData	getMetaData ()Retrieves the number, types and properties of this ResultSet object's columns.
Reader	getNCharacterStream (int columnIndex)Retrieves the value of the designated column in the current row of this ResultSet object as a java.io.Reader object.
Reader	getNCharacterStream (String columnLabel)Retrieves the value of the designated column in the current row of this ResultSet object as a java.io.Reader object.
NClob	getNClob (int columnIndex)Retrieves the value of the designated column in the current row of this ResultSet object as a NClob object in the Java programming language.
NClob	getNClob (String columnLabel)Retrieves the value of the designated column in the current row of this ResultSet object as a NClob object in the Java programming language.
String	getString (int columnIndex)Retrieves the value of the designated column in the current row of this ResultSet object as a String in the Java programming language.
String	getString (String columnLabel)Retrieves the value of the designated column in the current row of this ResultSet object as a String in the Java programming language.
Object	getObject (int columnIndex)Gets the value of the designated column in the current row of this ResultSet object as an Object in the Java programming language.
<T> T	getObject (int columnIndex, Class<T> type)Retrieves the value of the designated column in the current row of this ResultSet object and will convert from the SQL type of the column to the requested Java data type, if the conversion is supported.
Object	getObject (int columnIndex, Map<String,Class<?>> map)Retrieves the value of the designated column in the current row of this ResultSet object as an Object in the Java programming language.
Object	getObject (String columnLabel)Gets the value of the designated column in the current row of this ResultSet object as an Object in the Java programming language.
<T> T	getObject (String columnLabel, Class<T> type)Retrieves the value of the designated column in the current row of this ResultSet object and will convert from the SQL type of the column to the requested Java data type, if the conversion is supported.
Object	getObject (String columnLabel, Map<String,Class<?>> map)Retrieves the value of the designated column in the current row of this ResultSet object as an Object in the Java programming language.
Ref	getRef (int columnIndex)Retrieves the value of the designated column in the current row of this ResultSet object as a Ref object in the Java programming language.
Ref	getRef (String columnLabel)Retrieves the value of the designated column in the current row of this ResultSet object as a Ref object in the Java programming language.
int	getRow ()Retrieves the current row number.
RowId	getRowId (int columnIndex)Retrieves the value of the designated column in the current row of this ResultSet object as a java.sql.RowId object in the Java programming language.
RowId	getRowId (String columnLabel)Retrieves the value of the designated column in the current row of this ResultSet object as a java.sql.RowId object in the Java programming language.

short	getShort (int columnIndex)Retrieves the value of the designated column in the current row of this ResultSet object as a short in the Java programming language.
short	getShort (String columnLabel)Retrieves the value of the designated column in the current row of this ResultSet object as a short in the Java programming language.
SQLXML	getSQLXML (int columnIndex)Retrieves the value of the designated column in the current row of this ResultSet as a java.sql.SQLXML object in the Java programming language.
SQLXML	getSQLXML (String columnLabel)Retrieves the value of the designated column in the current row of this ResultSet as a java.sql.SQLXML object in the Java programming language.
Statement	getStatement ()Retrieves the Statement object that produced this ResultSet object.
String	getString (int columnIndex)Retrieves the value of the designated column in the current row of this ResultSet object as a String in the Java programming language.
String	getString (String columnLabel)Retrieves the value of the designated column in the current row of this ResultSet object as a String in the Java programming language.
Time	getTime (int columnIndex)Retrieves the value of the designated column in the current row of this ResultSet object as a java.sql.Time object in the Java programming language.
Time	getTime (int columnIndex, Calendar cal)Retrieves the value of the designated column in the current row of this ResultSet object as a java.sql.Time object in the Java programming language.
Time	getTime (String columnLabel)Retrieves the value of the designated column in the current row of this ResultSet object as a java.sql.Time object in the Java programming language.
Time	getTime (String columnLabel, Calendar cal)Retrieves the value of the designated column in the current row of this ResultSet object as a java.sql.Time object in the Java programming language.
Timestamp	getTimestamp (int columnIndex)Retrieves the value of the designated column in the current row of this ResultSet object as a java.sql.Timestamp object in the Java programming language.
Timestamp	getTimestamp (int columnIndex, Calendar cal)Retrieves the value of the designated column in the current row of this ResultSet object as a java.sql.Timestamp object in the Java programming language.
Timestamp	getTimestamp (String columnLabel)Retrieves the value of the designated column in the current row of this ResultSet object as a java.sql.Timestamp object in the Java programming language.
Timestamp	getTimestamp (String columnLabel, Calendar cal)Retrieves the value of the designated column in the current row of this ResultSet object as a java.sql.Timestamp object in the Java programming language.

int	getType() Retrieves the type of this ResultSet object.
InputStream	getUnicodeStream(int columnIndex) Deprecated. use <i>getCharacterStream</i> in place of <i>getUnicodeStream</i>
InputStream	getUnicodeStream(String columnLabel) Deprecated. use <i>getCharacterStream</i> instead
URL	getURL(int columnIndex) Retrieves the value of the designated column in the current row of this ResultSet object as a java.net.URL object in the Java programming language.
URL	getURL(String columnLabel) Retrieves the value of the designated column in the current row of this ResultSet object as a java.net.URL object in the Java programming language.
SQLWarning	getWarnings() Retrieves the first warning reported by calls on this ResultSet object.
void	insertRow() Inserts the contents of the insert row into this ResultSet object and into the database.
boolean	isAfterLast() Retrieves whether the cursor is after the last row in this ResultSet object.
boolean	isBeforeFirst() Retrieves whether the cursor is before the first row in this ResultSet object.
boolean	isClosed() Retrieves whether this ResultSet object has been closed.
boolean	isFirst() Retrieves whether the cursor is on the first row of this ResultSet object.
boolean	isLast() Retrieves whether the cursor is on the last row of this ResultSet object.
boolean	last() Moves the cursor to the last row in this ResultSet object.

void	moveToCurrentRow() Moves the cursor to the remembered cursor position, usually the current row.
void	moveToInsertRow() Moves the cursor to the insert row.
boolean	next() Moves the cursor forward one row from its current position.
boolean	previous() Moves the cursor to the previous row in this ResultSet object.
void	refreshRow() Refreshes the current row with its most recent value in the database.
boolean	relative(int rows) Moves the cursor a relative number of rows, either positive or negative.
boolean	rowDeleted() Retrieves whether a row has been deleted.
boolean	rowInserted() Retrieves whether the current row has had an insertion.
boolean	rowUpdated() Retrieves whether the current row has been updated.
void	setFetchDirection(int direction) Gives a hint as to the direction in which the rows in this ResultSet object will be processed.
void	setFetchSize(int rows) Gives the JDBC driver a hint as to the number of rows that should be fetched from the database when more rows are needed for this ResultSet object.
void	updateArray(int columnIndex, Array x) Updates the designated column with a java.sql.Array value.
void	updateArray(String columnLabel, Array x) Updates the designated column with a java.sql.Array value.
void	updateAsciiStream(int columnIndex, InputStream x) Updates the designated column with an ascii stream value.
void	updateAsciiStream(int columnIndex, InputStream x, int length) Updates the designated column with an ascii stream value, which will have the specified number of bytes.
void	updateAsciiStream(int columnIndex, InputStream x, long length) Updates the designated column with an ascii stream value, which will have the specified number of bytes.
void	updateAsciiStream(String columnLabel, InputStream x) Updates the designated column with an ascii stream value.
void	updateAsciiStream(String columnLabel, InputStream x, int length) Updates the designated column with an ascii stream value, which will have the specified number of bytes.
void	updateAsciiStream(String columnLabel, InputStream x, long length) Updates the designated column with an ascii stream value, which will have the specified number of bytes.
void	updateBigDecimal(int columnIndex, BigDecimal x) Updates the designated column with a java.math.BigDecimal value.
void	updateBigDecimal(String columnLabel, BigDecimal x) Updates the designated column with a java.sql.BigDecimal value.

void	updateBinaryStream (int columnIndex, InputStream x)	Updates the designated column with a binary stream value.
void	updateBinaryStream (int columnIndex, InputStream x, int length)	Updates the designated column with a binary stream value, which will have the specified number of bytes.
void	updateBinaryStream (int columnIndex, InputStream x, long length)	Updates the designated column with a binary stream value, which will have the specified number of bytes.
void	updateBinaryStream (String columnLabel, InputStream x)	Updates the designated column with a binary stream value.
void	updateBinaryStream (String columnLabel, InputStream x, int length)	Updates the designated column with a binary stream value, which will have the specified number of bytes.
void	updateBinaryStream (String columnLabel, InputStream x, long length)	Updates the designated column with a binary stream value, which will have the specified number of bytes.
void	updateBlob (int columnIndex, Blob x)	Updates the designated column with a java.sql.Blob value.
void	updateBlob (int columnIndex, InputStream inputStream)	Updates the designated column using the given input stream.
void	updateBlob (int columnIndex, InputStream inputStream, long length)	Updates the designated column using the given input stream, which will have the specified number of bytes.
void	updateBlob (String columnLabel, Blob x)	Updates the designated column with a java.sql.Blob value.
void	updateBlob (String columnLabel, InputStream inputStream)	Updates the designated column using the given input stream.
void	updateBlob (String columnLabel, InputStream inputStream, long length)	Updates the designated column using the given input stream, which will have the specified number of bytes.
void	updateBoolean (int columnIndex, boolean x)	Updates the designated column with a boolean value.
void	updateBoolean (String columnLabel, boolean x)	Updates the designated column with a boolean value.

void	updateByte (int columnIndex, byte x)	Updates the designated column with a byte value.
void	updateByte (String columnLabel, byte x)	Updates the designated column with a byte value.
void	updateBytes (int columnIndex, byte[] x)	Updates the designated column with a byte array value.
void	updateBytes (String columnLabel, byte[] x)	Updates the designated column with a byte array value.
void	updateCharacterStream (int columnIndex, Reader x)	Updates the designated column with a character stream value.
void	updateCharacterStream (int columnIndex, Reader x, int length)	Updates the designated column with a character stream value, which will have the specified number of bytes.
void	updateCharacterStream (int columnIndex, Reader x, long length)	Updates the designated column with a character stream value, which will have the specified number of bytes.
void	updateCharacterStream (String columnLabel, Reader reader)	Updates the designated column with a character stream value.
void	updateCharacterStream (String columnLabel, Reader reader, int length)	Updates the designated column with a character stream value, which will have the specified number of bytes.
void	updateCharacterStream (String columnLabel, Reader reader, long length)	Updates the designated column with a character stream value, which will have the specified number of bytes.
void	updateClob (int columnIndex, Clob x)	Updates the designated column with a java.sql.Clob value.
void	updateClob (int columnIndex, Reader reader)	Updates the designated column using the given Reader object.
void	updateClob (int columnIndex, Reader reader, long length)	Updates the designated column using the given Reader object, which is the given number of characters long.
void	updateClob (String columnLabel, Clob x)	Updates the designated column with a java.sql.Clob value.
void	updateClob (String columnLabel, Reader reader)	Updates the designated column using the given Reader object.
void	updateClob (String columnLabel, Reader reader, long length)	Updates the designated column using the given Reader object, which is the given number of characters long.

void	updateDate (int columnIndex, Date x)Updates the designated column with a java.sql.Date value.
void	updateDate(String columnLabel, Date x)Updates the designated column with a java.sql.Date value.
void	updateDouble (int columnIndex, double x)Updates the designated column with a double value.
void	updateDouble(String columnLabel, double x)Updates the designated column with a double value.
void	updateFloat (int columnIndex, float x)Updates the designated column with a float value.
void	updateFloat(String columnLabel, float x)Updates the designated column with a float value.
void	updateInt (int columnIndex, int x)Updates the designated column with an int value.
void	updateInt(String columnLabel, int x)Updates the designated column with an int value.
void	updateLong (int columnIndex, long x)Updates the designated column with a long value.
void	updateLong(String columnLabel, long x)Updates the designated column with a long value.
void	updateNCharacterStream (int columnIndex, Reader x)Updates the designated column with a character stream value.
void	updateNCharacterStream (int columnIndex, Reader x, long length)Updates the designated column with a character stream value, which will have the specified number of bytes.
void	updateNCharacterStream(String columnLabel, Reader reader)Updates the designated column with a character stream value.
void	updateNCharacterStream(String columnLabel, Reader reader, long length)Updates the designated column with a character stream value, which will have the specified number of bytes.
void	updateNClob (int columnIndex, NClob nClob)Updates the designated column with a java.sql.NClob value.
void	updateNClob (int columnIndex, Reader reader)Updates the designated column using the given Reader The data will be read from the stream as needed until end-of-stream is reached.
void	updateNClob (int columnIndex, Reader reader, long length)Updates the designated column using the given Reader object, which is the given number of characters long.
void	updateNClob(String columnLabel, NClob nClob)Updates the designated column with a java.sql.NClob value.
void	updateNClob(String columnLabel, Reader reader)Updates the designated column using the given Reader object.
void	updateNClob(String columnLabel, Reader reader, long length)Updates the designated column using the given Reader object, which is the given number of characters long.

void	updateNString (int columnIndex, String nString)Updates the designated column with a String value.
void	updateNString(String columnLabel, String nString)Updates the designated column with a String value.
void	updateNull (int columnIndex)Updates the designated column with a null value.
void	updateNull(String columnLabel)Updates the designated column with a null value.
void	updateObject (int columnIndex, Object x)Updates the designated column with an Object value.
void	updateObject (int columnIndex, Object x, int scaleOrLength)Updates the designated column with an Object value.
void	updateObject(String columnLabel, Object x)Updates the designated column with an Object value.
void	updateObject(String columnLabel, Object x, int scaleOrLength)Updates the designated column with an Object value.
void	updateRef (int columnIndex, Ref x)Updates the designated column with a java.sql.Ref value.
void	updateRef(String columnLabel, Ref x)Updates the designated column with a java.sql.Ref value.
void	updateRow ()Updates the underlying database with the new contents of the current row of this ResultSet object.
void	updateRowId (int columnIndex, RowId x)Updates the designated column with a RowId value.
void	updateRowId(String columnLabel, RowId x)Updates the designated column with a RowId value.
void	updateShort (int columnIndex, short x)Updates the designated column with a short value.
void	updateShort(String columnLabel, short x)Updates the designated column with a short value.
void	updateSQLXML (int columnIndex, SQLXML xmlObject)Updates the designated column with a java.sql.SQLXML value.
void	updateSQLXML(String columnLabel, SQLXML xmlObject)Updates the designated column with a java.sql.SQLXML value.
void	updateString (int columnIndex, String x)Updates the designated column with a String value.
void	updateString(String columnLabel, String x)Updates the designated column with a String value.
void	updateTime (int columnIndex, Time x)Updates the designated column with a java.sql.Time value.
void	updateTime(String columnLabel, Time x)Updates the designated column with a java.sql.Time value.
void	updateTimestamp (int columnIndex, Timestamp x)Updates the designated column with a java.sql.Timestamp value.
void	updateTimestamp(String columnLabel, Timestamp x)Updates the designated column with a java.sql.Timestamp value.
boolean	wasNull ()Reports whether the last column read had a value of SQL NULL.



Interface DatabaseMetaData

- This interface is implemented by driver vendors to let users know the capabilities of a Database Management System (DBMS/RDBMS) in combination with the driver based on JDBC™ technology ("JDBC driver").
- Different relational DBMSs often support different features, implement features in different ways using different data types.
- In addition, a driver may implement a feature on top of what the DBMS offers.
- Information returned by methods in this interface applies to the capabilities of a particular driver and a particular DBMS working together.



Methods

- **boolean allProceduresAreCallable()**
- Retrieves whether the current user can call all the procedures returned by the method `getProcedures`.
- **boolean allTablesAreSelectable()**
- Retrieves whether the current user can use all the tables returned by the method `getTables` in a `SELECT` statement.
- **Connection getConnection()**
- Retrieves the connection that produced this metadata object.



Methods

- **ResultSet getClientInfoProperties()**
- Retrieves a list of the client info properties that the driver supports.
- **ResultSet getColumnPrivileges(String catalog, String schema, String table, String columnNamePattern)**
- Retrieves a description of the access rights for a table's columns.
- **ResultSet getColumns(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern)**
- Retrieves a description of table columns available in the specified catalog.



Methods

- **int getDatabaseMajorVersion()**
- Retrieves the major version number of the underlying database.
- **int getDatabaseMinorVersion()**
- Retrieves the minor version number of the underlying database.
- **String getDatabaseProductName()**
- Retrieves the name of this database product.
- **String getDatabaseProductVersion()**
- Retrieves the version number of this database product.



Methods

- **int getDriverMajorVersion()**
 - Retrieves this JDBC driver's major version number.
- **int getDriverMinorVersion()**
 - Retrieves this JDBC driver's minor version number.
- **String getDriverName()**
 - Retrieves the name of this JDBC driver.
- **String getDriverVersion()**
 - Retrieves the version number of this JDBC driver as a String.
- **String getURL()**
 - Retrieves the URL for this DBMS.
- **String getUsername()**
 - Retrieves the user name as known to this database.



Interface ResultSetMetaData

- An object that can be used to get information about the types and properties of the columns in a ResultSet object.
- The following code fragment creates the ResultSet object rs, creates the ResultSetMetaData object rsmd, and uses rsmd to find out how many columns rs has and whether the first column in rs can be used in a WHERE clause.
 - `ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM TABLE2");`
 - `ResultSetMetaData rsmd = rs.getMetaData();`
 - `int numberOfColumns = rsmd.getColumnCount();`
 - `boolean b = rsmd.isSearchable(1);`



Methods

- **int getColumnCount()**
 - Returns the number of columns in this ResultSet object.
- **String getColumnLabel(int column)**
 - Gets the designated column's suggested title for use in printouts and displays.
- **String getColumnName(int column)**
 - Get the designated column's name.
- **int getColumnType(int column)**
 - Retrieves the designated column's SQL type.
- **String getColumnName(int column)**
 - Retrieves the designated column's database-specific type name.



Methods

- **boolean isCurrency(int column)**
 - Indicates whether the designated column is a cash value.
- **boolean isDefinitelyWritable(int column)**
 - Indicates whether a write on the designated column will definitely succeed.
- **int isNullable(int column)**
 - Indicates the nullability of values in the designated column.
- **boolean isReadOnly(int column)**
 - Indicates whether the designated column is definitely not writable.
- **boolean isSearchable(int column)**
 - Indicates whether the designated column can be used in a where clause.



Methods

- **int** **getPrecision(int column)**
 - Get the designated column's specified column size.
- **int** **getScale(int column)**
 - Gets the designated column's number of digits to right of the decimal point.
- **String** **getTableName(int column)**
 - Gets the designated column's table name.
- **boolean** **isAutoIncrement(int column)**
 - Indicates whether the designated column is automatically numbered.
- **boolean** **isCaseSensitive(int column)**
 - Indicates whether a column's case matters.



Methods

- **boolean** **isSigned(int column)**
 - Indicates whether values in the designated column are signed numbers.
- **boolean** **isWritable(int column)**
 - Indicates whether it is possible for a write on the designated column to succeed.

Data types in JDBC

- Java has a basic data types, for example, int, long, float, double, string.
- Database systems also have a data type system, like int, char, varchar, text, blob, clob.
- The JDBC driver can convert the Java data type to the appropriate database type to and from the database.
- The mapping is used when calling the setXXX() method from the PreparedStatement or CallableStatement object or the ResultSet.updateXXX()/getXXX() method.

SQL	JDBC/Java	setXXX	getXXX	updateXXX
VARCHAR	java.lang.String	setString	getString	updateString
CHAR	java.lang.String	setString	getString	updateString
LONGVARCHAR	java.lang.String	setString	updateString	
BIT	boolean	setBoolean	getBoolean	updateBoolean
NUMERIC	java.math.BigDecimal	setBigDecimal	getBigDecimal	updateBigDecimal
TINYINT	byte	setByte	getByte	updateByte
SMALLINT	short	setShort	getShort	updateShort
INTEGER	int	setInt	getInt	updateInt
BIGINT	long	setLong	getLong	updateLong

REAL	float	setFloat	getFloat	updateFloat
FLOAT	float	setFloat	getFloat	updateFloat
DOUBLE	double	setDouble	getDouble	updateDouble
VARBINARY	byte[]	setBytes	getBytes	updateBytes
BINARY	byte[]	setBytes	getBytes	updateBytes
DATE	java.sql.Date	setDate	getDate	updateDate
TIME	java.sql.Time	setTime	getTime	updateTime
TIMESTAMP	java.sql.Timestamp	setTimestamp	getTimestamp	updateTimestamp
CLOB	java.sql.Clob	setClob	getClob	updateClob
BLOB	java.sql.Blob	setBlob	getBlob	updateBlob
ARRAY	java.sql.Array	setARRAY	getARRAY	updateARRAY
REF	java.sql.Ref	setRef	getRef	updateRef
STRUCT	java.sql.Struct	setStruct	getStruct	updateStruct

Processing queries with JDBC

- In general, to process any SQL statement with JDBC, you follow these steps:
 - Establishing a connection.
 - Create a statement.
 - Execute the query.
 - Process the ResultSet object.
 - Close the connection.



Database Exception Handling

- When JDBC encounters an error during an interaction with a data source or a database connection, it throws an instance of `SQLException` as opposed to `Exception`.
- The `SQLException` instance contains the following information that can help determine the cause of the error:
 - A description of the error. Retrieve the `String` object that contains this description by calling the method `SQLException.getMessage`.
 - A `SQLState` code. These codes and their respective meanings have been standardized by ISO/ANSI and Open Group (X/Open), although some codes have been reserved for database vendors to define for themselves. This `String` object consists of five alphanumeric characters. Retrieve this code by calling the method `SQLException.getSQLState`.



Database Exception Handling-II

- An error code. This is an integer value identifying the error that caused the `SQLException` instance to be thrown. Its value and meaning are implementation-specific and might be the actual error code returned by the underlying data source. Retrieve the error by calling the method `SQLException.getErrorCode`.
- A cause. A `SQLException` instance might have a causal relationship, which consists of one or more `Throwable` objects that caused the `SQLException` instance to be thrown. To navigate this chain of causes, recursively call the method `SQLException.getCause` until a null value is returned.
- A reference to any *chained* exceptions. If more than one error occurs, the exceptions are referenced through this chain. Retrieve these exceptions by calling the method `SQLException.getNextException` on the exception that was thrown.

Example

```
public static void
printSQLException(SQLException
ex) {
    for (Throwable e : ex) {
        if (e instanceof SQLException) {
            if (ignoreSQLException(
                ((SQLException)e).
                getSQLState()) == false)
            {
                e.printStackTrace(System.err);
                System.err.println("SQLState: " +
                    ((SQLException)e).getSQLState());
                System.err.println("Error
                Code: " +
                    ((SQLException)e).getErrorCode());
                System.err.println("Message: " +
                    e.getMessage());
                Throwable t =
                    ex.getCause();
                while(t != null) {
                    System.out.println("Cause: " + t);
                    t = t.getCause();
                }
            }
        }
    }
}
```

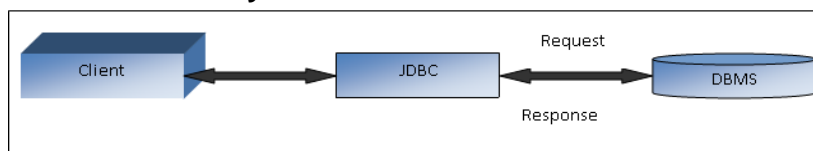
CONNECTING WITH DATABASES (MYSQL, ACCESS, ORACLE)

I. MS-Access (JDBC-ODBC) Connectivity

- Working with leaders in the database field, Sun developed a single API for database access—JDBC.
- As part of this process, they kept three main goals in mind:
 - JDBC should be a SQL-level API.
 - JDBC should capitalize on the experience of existing database APIs.
 - JDBC should be simple.
- JDBC API makes connection between Java application or Applet and Database Management System.
- An application which is written to access DBMS are vendor specific that means, an application which can access the DBMS of one vendor cannot access the DBMS of another vendor.

JDBC-ODBC Connectivity

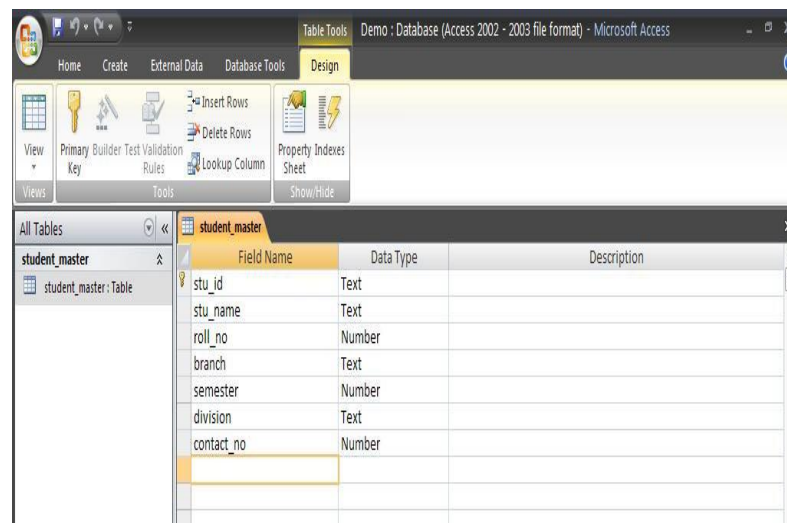
- Microsoft has developed a standard called ODBC (Open Database Connectivity) which is free from any vendor specific DBMS structure.
- A client using ODBC API can send SQL statements to database server and get the result.
- A JDBC client does not direct link to the DBMS server.
- A JDBC makes a use of ODBC to connect with database.
- The bridge between Java program and database is known as a JDBC - ODBC Driver.



Types of Connections

- DSN Oriented Connection
 - A layer of Data Source Name (DSN) is kept between the Java Program and the actual Database.
 - DSN Less Connection
 - A direct connection to the Database is managed here by directly specifying the driver path while preparing the connection.
- I. DSN Oriented Connection
- Here, first we need to create a DSN for our database by selecting the appropriate driver for it.
 - Then we need to specify the DSN path as url while preparing the connection.
 - The steps to be followed to create the DSN are as furnished in the subsequent steps in the form of snapshots.

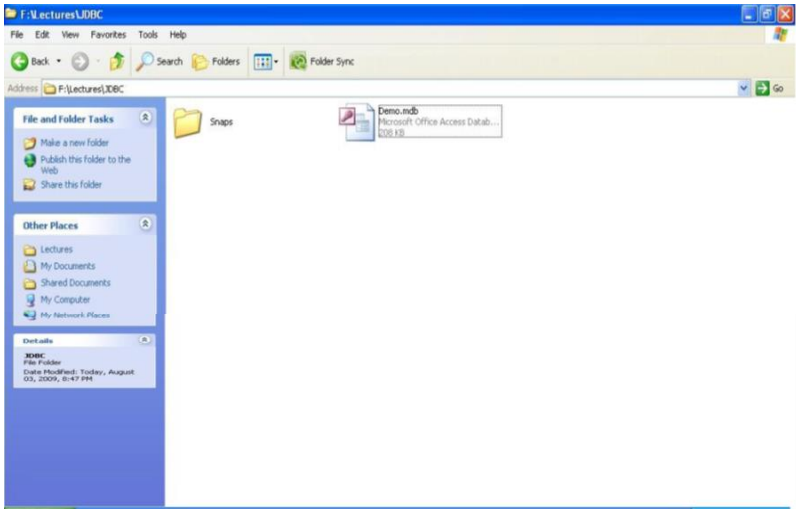
I. Create Ms Access Database



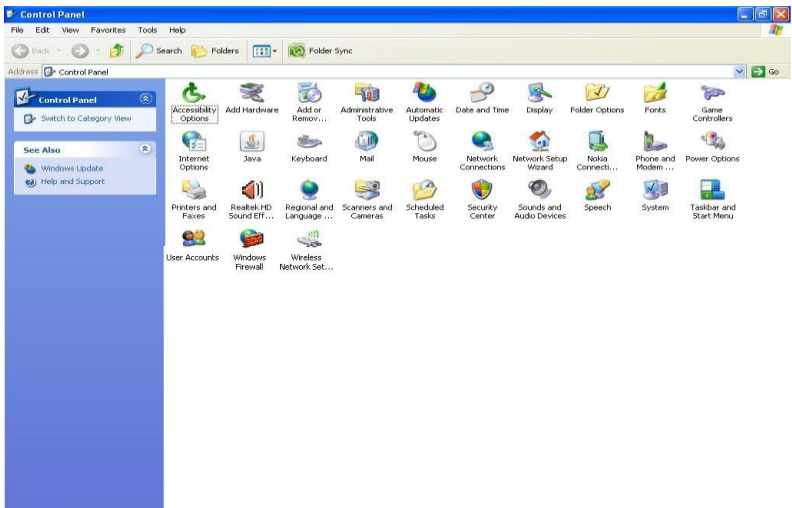
The screenshot shows the Microsoft Access application window titled "Demo : Database (Access 2002 - 2003 file format) - Microsoft Access". The ribbon is set to "Design" under the "Table Tools" group. The "Design" ribbon includes tabs for "Views", "Tools", and "Show/Hide". The "Tools" tab is active, showing options like "Primary", "Builder", "Test", "Validation", "Rules", "Lookup Column", "Insert Rows", "Delete Rows", "Property", "Indexes", and "Sheet". The "Table Design" grid is visible, showing the structure of the "student_master" table. The grid has three columns: "Field Name", "Data Type", and "Description". The fields listed are: "stu_id" (Text), "stu_name" (Text), "roll_no" (Number), "branch" (Text), "semester" (Number), "division" (Text), and "contact_no" (Number). The "roll_no" field is highlighted with a yellow background.

Field Name	Data Type	Description
stu_id	Text	
stu_name	Text	
roll_no	Number	
branch	Text	
semester	Number	
division	Text	
contact_no	Number	

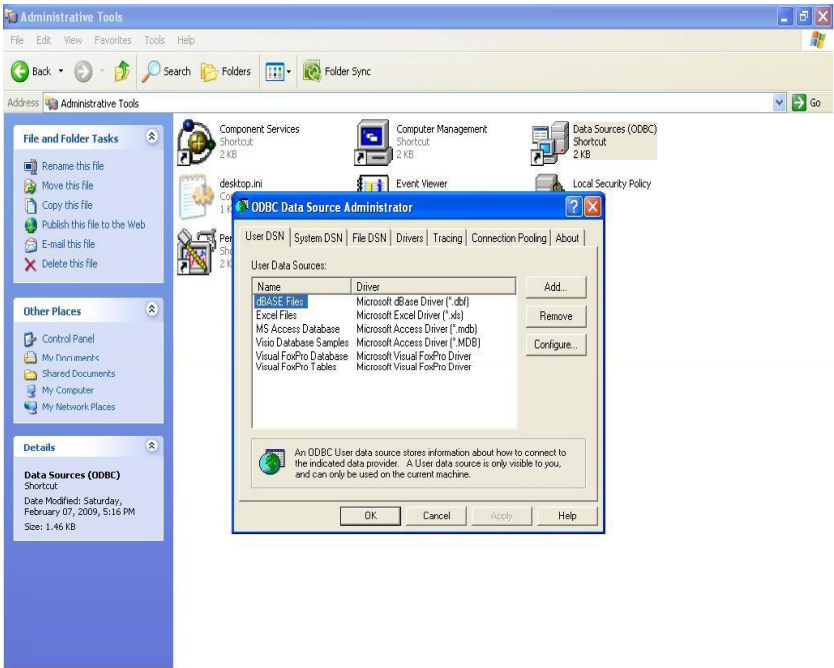
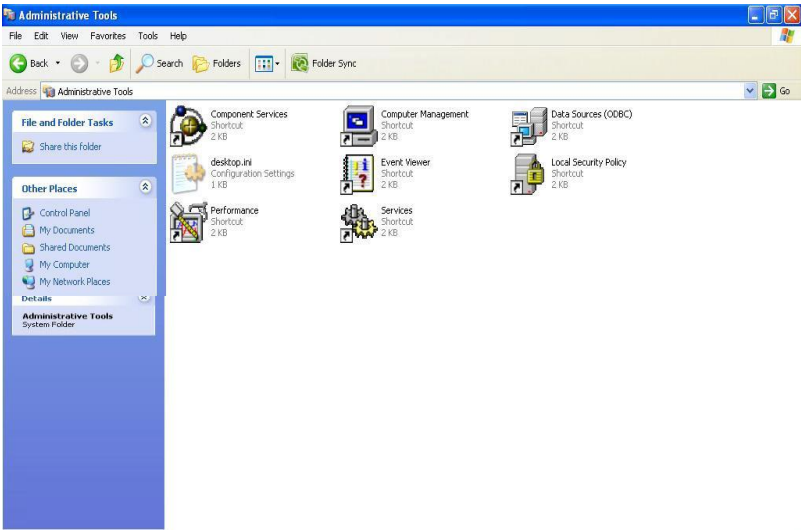
2. Browse to the location of the database(*.mdb, *.accdb)



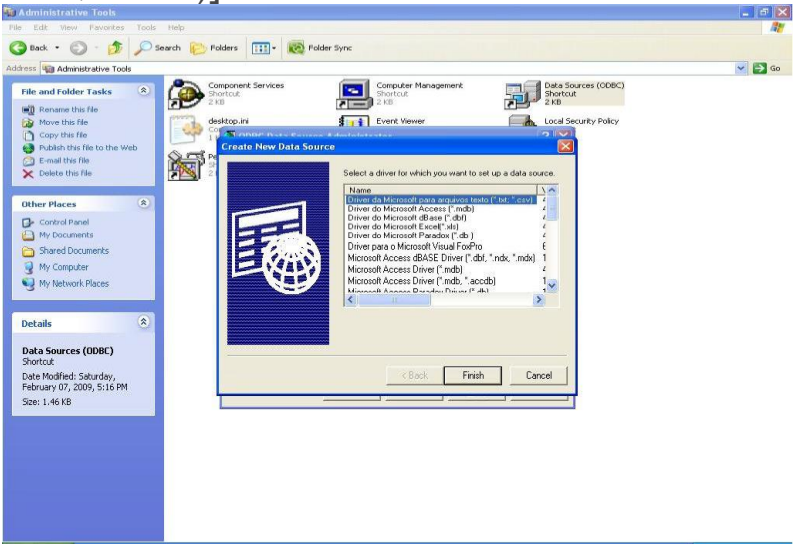
3. Go to Control Panel



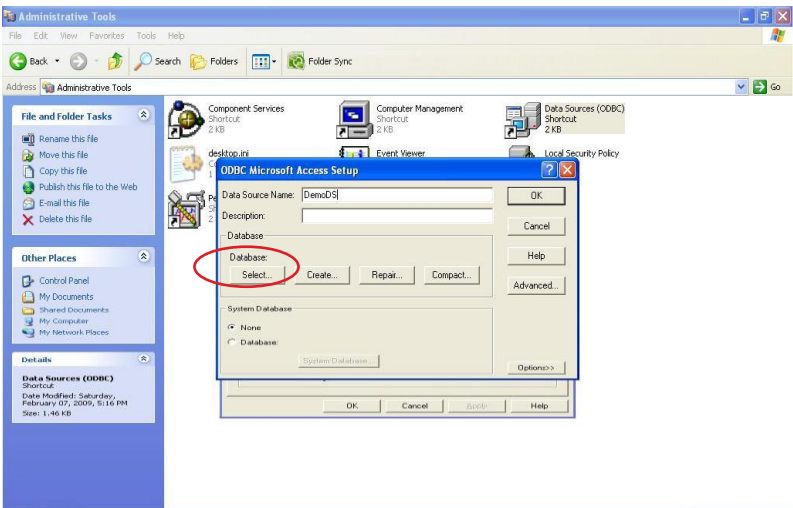
4.Administrative Tools -> Data Sources (ODBC)



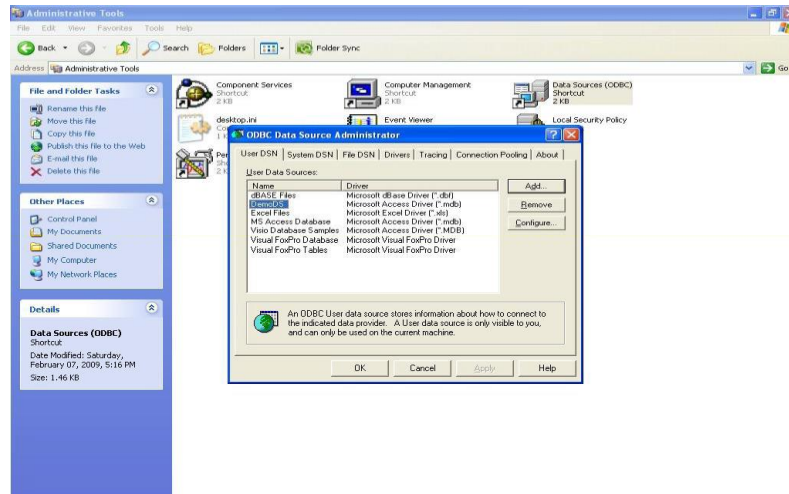
- 5. Create New Data Source
- 6. Select Microsoft Access Driver (*.mdb) [(*.mdb,*.accdb)]



- 7. DSN Setup
- 8. Select Database



9. DSN Created Successfully.



II. DSN Less Connection

- Here, we need not to create and connect to any DSN to have the JDBC connectivity with our database.
- Instead of that, we directly specify the drivers we want to use and the fully qualified path of our database while preparing the connection.
- For E.g., in the code we need to have the following connection string...

```
Connection con =  
DriverManager.getConnection("jdbc:odbc:Driver={  
Microsoft Access Driver  
(*.*.mdb)};DBQ=D:\\JDBC\\Student.mdb");
```




Creating JDBC Application:


- There are six steps involved in building a JDBC application:
- Step 1. Import the packages:
- This requires that you include the packages containing the JDBC classes needed for database programming. Most often, using `import java.sql.*` will suffice as follows:
- `//STEP 1. Import required packages`
- `import java.sql.*;`



Step 2. Register the JDBC driver:

- This requires that you initialize a driver so you can open a communications channel with the database. Following is the code snippet to achieve this:
- `//STEP 2: Register JDBC driver`
- `Class.forName("com.mysql.jdbc.Driver");`

- 
- **Step 3. Open a connection:**
 - This requires using the `DriverManager.getConnection()` method to create a `Connection` object, which represents a physical connection with the database as follows:
 - `//STEP 3: Open a connection`
 - `// Database credentials`
 - `static final String USER = "username";`
 - `static final String PASS = "password";`
 - `System.out.println("Connecting to database...");`
 - `conn = DriverManager.getConnection(DB_URL,USER,PASS);`
 - **Step 4. Create a Statement:**
 - This requires using an object of type `Statement` or `PreparedStatement` for building and submitting an SQL statement to the database as follows:
 - `System.out.println("Creating statement...");`
 - `stmt = conn.createStatement();`

- 
- **Step 5. Execute a query:**
 - `//STEP 5: Execute a query`
 - `String sql;`
 - `sql = "SELECT id, first, last, age FROM Employees";`
 - `ResultSet rs = stmt.executeQuery(sql);`
 - If there is an SQL `UPDATE`, `INSERT` or `DELETE` statement required, then following code snippet would be required:
 - `//STEP 5: Execute a query`
 - `System.out.println("Creating statement...");`
 - `stmt = conn.createStatement();`
 - `String sql;`
 - `sql = "DELETE FROM Employees";`
 - `ResultSet rs = stmt.executeUpdate(sql);`



Step 6. Extract data from result set:

- This step is required in case you are fetching data from the database. You can use the appropriate `ResultSet.getXXX()` method to retrieve the data from the result set as follows:

```
//STEP 6: Extract data from result set
while(rs.next()){
//Retrieve by column name or no
int id = rs.getInt("id"); // rs.getInt(1); where 1=column number
int age = rs.getInt("age");
String first = rs.getString("first");
String last = rs.getString("last");
//Display values
System.out.print("ID: " + id);
System.out.print(", Age: " + age);
System.out.print(", First: " + first);
System.out.println(", Last: " + last);
}
```



Step 7. Handle SQLException and Clean up the environment:

- You should explicitly close all database resources versus relying on the JVM's garbage collection as follows:

```
//STEP 7: Handle SQLException and Clean-up environment
rs.close();
stmt.close();
conn.close();

try
{
.....
}
catch(SQLException se){...}
```

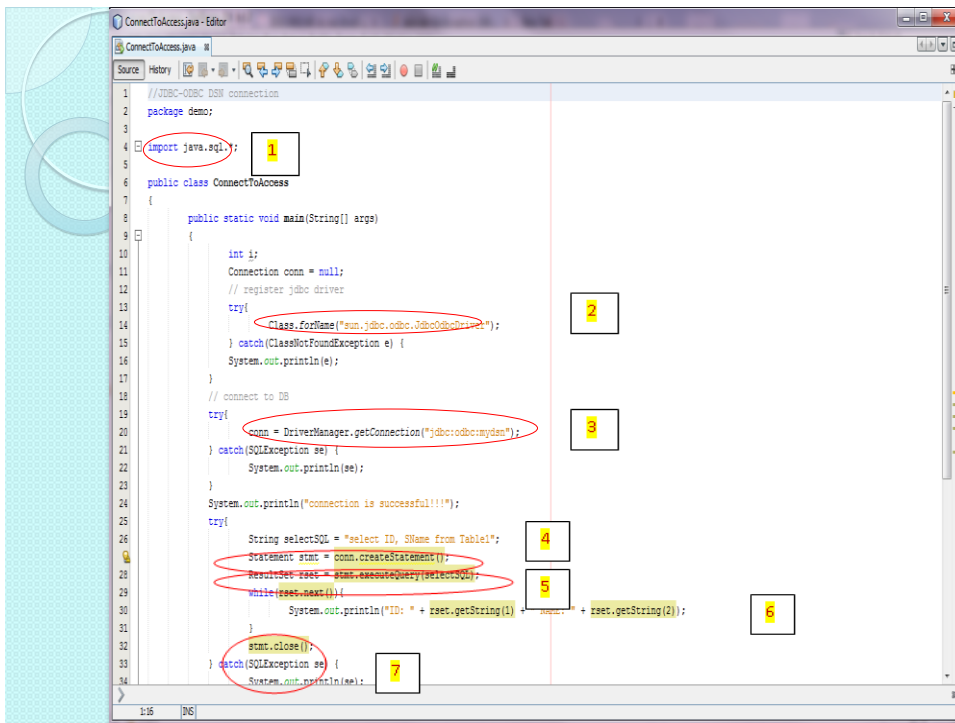
JDBC Programs

I. Program for JDBC-ODBC DSN connection using Statement.

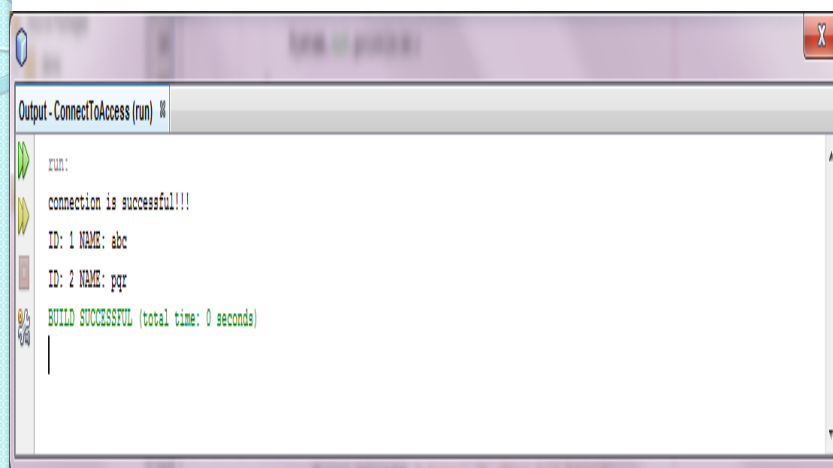
- Note: Save it like a simple Java Program as eg. ConnectToAccess.java

```
package demo;
import java.sql.*;
public class ConnectToAccess
{
    public static void main(String[] args)
    {
        int i;
        Connection conn = null;
        // register jdbc driver
        try{
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        }
        catch(ClassNotFoundException e) {
            System.out.println(e);
        }
        // connect to DB
        try{
            conn =
            DriverManager.getConnection("jdbc:odbc:mydb");
        } catch(SQLException se) {
            System.out.println(se);
        }

        System.out.println("connection is
        successful!!!");
        try{
            String selectSQL = "select
            ID, NAME from Table1";
            Statement stmt =
            conn.createStatement();
            ResultSet rset =
            stmt.executeQuery(selectSQL);
            while(rset.next()){
                System.out.println("ID:" +
                rset.getString(1) + " NAME:" +
                rset.getString(2);
            }
            stmt.close();
        } catch(SQLException se) {
            System.out.println(se);
        }
    }
}
```



Output: Console window(NetBeans)



2. Program for JDBC-ODBC DSN less connection

```
//JDBC-ODBC Dsn less connection
package demo;

import java.sql.*;

public class ConnectNoDSN
{
    public static void main(String[] args)
    {
        int i;
        Connection conn = null;
        // register jdbc driver
        try{
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        } catch(ClassNotFoundException e) {
            System.out.println(e);
        }
    }
}
```

```
// connect to DB
try{
    String myDB ="jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)};DBQ=C:/Users/jesussaves/Desktop/mydb.mdb";
    conn = DriverManager.getConnection(myDB,"","");
} catch(SQLException se) {
    System.out.println(se);
}
System.out.println("connection is successful!!!");
try{
    String selectSQL = "select ID, SName from Table1 ";
    Statement stmt = conn.createStatement();
    ResultSet rset = stmt.executeQuery(selectSQL);
    while(rset.next()){
        System.out.println("ID: " + rset.getString(1) + " NAME: " +
rset.getString(2));
    }
    stmt.close();
} catch(SQLException se) {
    System.out.println(se);
}
}
```

The IDE interface shows the source code editor with line numbers, a toolbar, and a status bar at the bottom indicating the current line is 24 of 34.

2. JDBC-MySQL Connectivity

- Use XAMPP or any other web server solution package to create MySQL database.
- A Driver file is required for JDBC-MySQL Connection which is downloaded beforehand. Eg. mysql-connector-java-5.0.5.jar
- Perform the connectivity code in the code editor of NetBeans IDE.

° PREPARED STATEMENT EXAMPLE USING MYSQL

```
package app;
import java.sql.*;
public class SelectRecords{
    public static void main(String[] args) {
        System.out.println("Select Records Example by using the
Prepared Statement!");
        Connection con = null;
        int count = 0;
        try{
            Class.forName("com.mysql.jdbc.Driver");
            con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/mydb","root","
");
            try{
                String sql = "SELECT rno,sname FROM
table1 WHERE rno >= ?";
                PreparedStatement prest =
con.prepareStatement(sql);
                prest.setInt(1,2);
                //prest.setInt(2,3);
```

```

        ResultSet rs = prest.executeQuery();
        while (rs.next()){
            String name = rs.getString(2);
            int no = rs.getInt(1);
            count++;
            System.out.println(no + "\t" + " - " +
+ name);
        }
        System.out.println("Number of records: " +
count);

        prest.close();
        con.close();
    }
    catch (SQLException s){
        System.out.println("SQL statement is not
executed!");
    }
}
catch (Exception e){
    e.printStackTrace();
}
}
}

```

```
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

//In This call we use only interfaces to refere objects.JDBC Objects class exist in
mysql jar file
public class CallableStatementExample {
    public static void main(String[] args) {
        // jdbc:mysql is protocol by which can be requested to connect with
        // database and 3306 is port number where my server can listen request
        String dbUrl = "jdbc:mysql://localhost:3306/mydb";// mydb is
        // database in mysql
        Connection conn = null;
        CallableStatement stmt = null;// this is CallableStatement reference
        // variable
        try {
            // STEP 2: Register JDBC driver
            Class.forName("com.mysql.jdbc.Driver");

            // STEP 3: Open a connection
            System.out.println("Connecting to database...");
            conn = DriverManager.getConnection(dbUrl, "root", ""); //any given passwd
```



CALLABLESTATEMENT EXAMPLE USING MYSQL



Create procedure in MySQL: getSName

```
DELIMITER $$  
  
DROP PROCEDURE IF EXISTS `getSName` $$  
CREATE PROCEDURE `getSName`(IN rnoI INT, OUT nameI  
VARCHAR(25))  
BEGIN  
  
SELECT name INTO nameI  
FROM tableI  
WHERE rno = rnoI;  
  
END $$  
  
DELIMITER ;
```



```

package app;
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
//In This call we use only interfaces to refere objects. JDBC Objects class exist in
//mysql jar file
public class CallableStatementExample {
    public static void main(String[] args) {
        // jdbc:mysql is protocol by which can be requested to connect with
        // database and 3306 is port number where my server can listen request
        String dbUrl = "jdbc:mysql://localhost:3306/mydb";// mydb is
        // database in mysql
        Connection conn = null;
        CallableStatement stmt = null;// this is CallableStatement reference
        // variable
        try {
            // STEP 2: Register JDBC driver
            Class.forName("com.mysql.jdbc.Driver");

            // STEP 3: Open a connection
            System.out.println("Connecting to database...");
            conn = DriverManager.getConnection(dbUrl, "root", ""); //any given passwd

```

```

// STEP 4: Execute a query
        System.out.println("Creating statement...");
        String sql = "{call getSName (?, ?)}";
        stmt = conn.prepareCall(sql);

        // Bind IN parameter first, then bind OUT
        parameter
        int no = 1;
        stmt.setInt(1, no); // This would set ID as 102
        // Because second parameter is OUT so
        register it
        stmt.registerOutParameter(2,
        java.sql.Types.VARCHAR);

        // Use execute method to run stored
        procedure.
        System.out.println("Executing stored
        procedure...");
        stmt.execute();

        // Retrieve employee name with getXXX
        method
        String sname = stmt.getString(2);
        System.out.println("Name with Roll no.:" +
        no + " is " + sname);
        stmt.close();
        conn.close();
    } catch (SQLException se) {

        // Handle errors for JDBC
        se.printStackTrace();
    } catch (Exception e) {
        // Handle errors for Class.forName
        e.printStackTrace();
    } finally {
        // finally block used to close resources
        try {
            if (stmt != null)
                stmt.close();
        } catch (SQLException se2) {
            // nothing we can do
        }
        try {
            if (conn != null)
                conn.close();
        } catch (SQLException se) {
            se.printStackTrace();
        }
        // end try
    } // end finally try
    System.out.println("Exited....!");
} // end main

```

3. JDBC-Oracle Connectivity Steps

1. Login to Oracle server using SQL Plus client.
 - a. Username and Password, hostString of Oracle Account ----- Eg. tybca01 , 123456, orcl or orcl.christcc.com
 - b. Create appropriate tables with given constraints and relationships in Oracle.
 - c. Insert records if required.
2. Have a JDBC-Oracle Driver.
 - Filename: Ojdbc6.jar
3. Start Netbeans->New Java Project->OracleJDBCEX->Finish.
 - a. Right Click on Source Packages->New package->app
 - b. Right Click on app->New Java Class->OrclJDBCEX
 - c. Write the connectivity code with the following driver class and connection string:

Driver class

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

Oracle Connection string

```
con = DriverManager.getConnection("jdbc:oracle:thin:@hpserver.christcc.com:1521:orcl","username","password");
```

- d. Change the select/insert/update/delete query as given in the Example. e. If given use Swing UI-JDBC-Oracle program code or just print on the console using `System.out.println()`.

New Connection Wizard

Customize Connection

Driver Name: Oracle Thin (Service ID (SID))

Host: hpserver.christcc.com Port: 1521

Service ID (SID): orcl

User Name: sharon

Password:

☐ Remember password

Connection Properties Test Connection

JDBC URL: jdbc:oracle:thin:@hpserver.christcc.com:1521:orcl

Connection Succeeded.

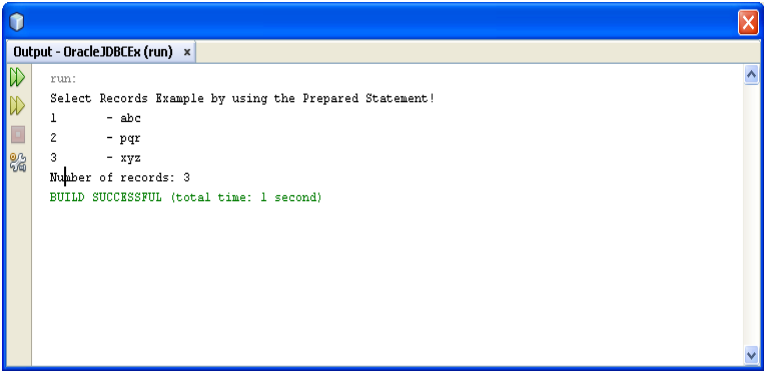
< Back Next > Finish Cancel Help

```


1 package app;
2 import java.sql.*;
3 public class SelectRecords {
4     public static void main(String[] args) {
5         System.out.println("Select Records Example by using the Prepared Statement!");
6         Connection con = null;
7         int count = 0;
8         try {
9             Class.forName("oracle.jdbc.driver.OracleDriver"); //driver
10            con = DriverManager.getConnection("jdbc:oracle:thin:@hpserver.christcc.com:1521:orcl", "sharon", "sharon");
11            try {
12                String sql = "SELECT rno,sname FROM stud";
13                Statement prest = con.createStatement();
14                //prest.setInt(1,2);
15                //prest.setInt(2,3);
16                ResultSet rs = prest.executeQuery(sql);
17                while (rs.next()) {
18                    String name = rs.getString(2);
19                    int no = rs.getInt(1);
20                    count++;
21                    System.out.println(no + "\t" + "-" + name);
22                }
23                System.out.println("Number of records: " + count);
24                prest.close();
25                con.close();
26            } catch (SQLException s) {
27                System.out.println("SQL statement is not executed!");
28            }
29        } catch (Exception e) {
30            e.printStackTrace();
31        }
32    }
33 }

```

Output



PREPAREDSTATEMENT EXAMPLE IN ORACLE



JDBC Oracle Program

```

package app;
import java.sql.*;
public class UpdateRecords {
    // JDBC driver name and database URL
    static final String JDBC_DRIVER = "oracle.jdbc.driver.OracleDriver";
    static final String DB_URL =
"jdbc:oracle:thin:@hpserver.christcc.com:1521:orcl";

    // Database credentials
    static final String USER = "sharon";//username
    static final String PASS = "shar2084";//pwd

    public static void main(String[] args) {
        Connection conn = null;
        PreparedStatement stmt = null;
        try{
            //STEP 2: Register JDBC driver
            Class.forName(JDBC_DRIVER);

            //STEP 3: Open a connection
            System.out.println("Connecting to database...");
            conn = DriverManager.getConnection(DB_URL,USER,PASS);

```



```

//STEP 4: Execute a query
    System.out.println("Creating statement...");
    String sql = "UPDATE stud set sname=? WHERE
rno=?";
    stmt = conn.prepareStatement(sql);

    //Bind values into the parameters.
    stmt.setString(1, "oracle");// This would set name
    stmt.setInt(2, 4); // This would set rno

    // Let us update age of the record with ID = 102;
    int rows = stmt.executeUpdate();
    System.out.println("Rows impacted : " + rows );

    // Let us select all the records and display them.
    sql = "SELECT rno, sname FROM stud";
    stmt = conn.prepareStatement(sql);
    ResultSet rs = stmt.executeQuery(sql);

    //STEP 5: Extract data from result set
    while(rs.next()){
        //Retrieve by column name
        int id = rs.getInt("rno");
        String first = rs.getString("sname");
        //Display values
        System.out.print("ID: " + id);
        System.out.print(", Name: " + first);

        }
    //STEP 6: Clean-up environment
    rs.close();
    stmt.close();
    conn.close();
}catch(SQLException se){
    //Handle errors for JDBC
    se.printStackTrace();
}catch(Exception e){
    //Handle errors for Class.forName
    e.printStackTrace();
}finally{
    //finally block used to close resources
    try{
        if(stmt!=null)
            stmt.close();
    }catch(SQLException se2){
    }// nothing we can do
    try{
        if(conn!=null)
            conn.close();
    }catch(SQLException se){
        se.printStackTrace();
    }//end finally try
    }//end try
    System.out.println("Exited!");
} //end main
} //end JDBCExample

```

CALLABLESTATEMENT EXAMPLE IN ORACLE

Create Procedure in Oracle: getSName

```

CREATE OR REPLACE PROCEDURE
SYSTEM.getSName
(s_name IN VARCHAR, rname OUT
VARCHAR) AS
BEGIN
    select sname into rname from
stud_details where sname like s_name;
END;

SELECT
sm.rno,sd.sname,sm.mark1,sm.mark2,sm.
mark3,sm.total,sm.per,sm.grade
FROM stud_marksheet sm left join
stud_details sd on sm.rno = sd.rno
where sd.sname = 'sharon';

DELIMITER @@
CREATE OR REPLACE procedure
SYSTEM."GETSNAME"
(s_name IN VARCHAR2,
r_no OUT NUMBER,
rname OUT VARCHAR2,
m1 OUT NUMBER,
m2 OUT NUMBER,
m3 OUT NUMBER,
tot OUT NUMBER,
per1 OUT NUMBER,
gr OUT VARCHAR2)
is
begin
    SELECT sm.rno into r_no,sd.sname into
rname,sm.mark1 into m1,sm.mark2 into
m2,sm.mark3 into m3,sm.total into
tot,sm.per into per1,sm.grade into gr
FROM stud_marksheet sm left join
stud_details sd on sm.rno = sd.rno
where sd.sname = s_name;

end; @@
DELIMITER ;

```

JDBC Oracle Program

```
package app;
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
```

//In This call we use only interfaces to refere objects.JDBC Objects class exist in mysql jar file

```
public class CallableStatementExample {
    public static void main(String[] args) throws SQLException {
        // jdbc:mysql is protocol by which can be requested to connect with
        // database and 3306 is port number where my server can listen request
        //String dbUrl = "jdbc:mysql://localhost:3306/mydb";// mydb is
        // database in mysql
        Connection conn = null;
        CallableStatement stmt = null;// this is CallableStatement reference
        PreparedStatement st = null;
        // variable
        try {
            // STEP 2: Register JDBC driver
            Class.forName("oracle.jdbc.OracleDriver");//com.mysql.jdbc.Driver");
```

```
// STEP 3: Open a connection
//System.out.println("Connecting to database...");
//conn = DriverManager.getConnection(dbUrl,"root","");//any given passwd


    conn =
    DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","system","oracle");//"jdbc:oracle:thin:@hpserver.christcc.com:1521:orcl","tybca_020","123456");
    String sql1 = "{call getSName(?,?)}";
    stmt = conn.prepareCall(sql1);
    //int no = 1;
    String nm = "sharon";//txt_roll_name.getText();
    stmt.setString(1,nm);//nm);// This would set ID as 102
    // Because second parameter is OUT so register it
    stmt.registerOutParameter(2,java.sql.Types.VARCHAR);

    int b = stmt.executeUpdate();

    if(b>0)
    {
        //stmt.close();

        String sname = stmt.getString(2);

        String sql = "SELECT sm.rno,sd.sname,sm.mark1,sm.mark2,sm.mark3,sm.total,sm.per,sm.grade
" +
        "FROM stud_marksheet sm left join stud_details sd on sm.rno = sd.rno" +
        "where sd.sname = 'sharon';";//+ sname.toString() + "";
```



```

st = conn.prepareStatement(sql);
    //prest.executeQuery(sql);
    //prest.setString(1,
txt_rol_name.getText());
    ResultSet rs = st.executeQuery();
    while(rs.next()){
        //tableModel.addRow(new
Object[]{new Integer(rs.getInt(1)),
rs.getString(2), rs.getInt(3), rs.getInt(4),
rs.getInt(5), new Float(rs.getInt(6)), new
Float(rs.getInt(7))+"%", rs.getString(8)});
        System.out.println("Name with
Name : " + sname);
    }
    }
    else
        System.out.println("Procedure
not executed");
    stmt.close();
    conn.close();
} catch (SQLException se) {
    // Handle errors for JDBC
    se.printStackTrace();
} catch (Exception e) {
    // Handle errors for Class.forName
    e.printStackTrace();
} finally {
    // finally block used to close
resources
    try {
        if (stmt != null)
            stmt.close();
    } catch (SQLException se2) {
        // nothing we can do
    }
    try {
        if (conn != null)
            conn.close();
    } catch (SQLException se) {
        se.printStackTrace();
    }
    // end try
} // end finally try
System.out.println("Exited....!");
} // end main
}

```