Java Database Connectivity (JDBC)

`java.sql` and `javax.sql`

What is JDBC?

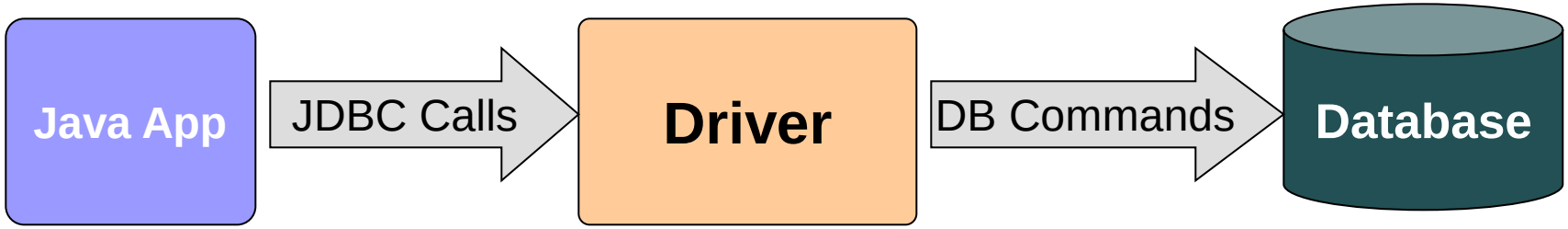
- Standard Java API for accessing relational database
  - Hides database specific details from application
- Part of J2SE

JDBC API

- **Defines a set of Java Interfaces, which are implemented by vendor-specific JDBC Drivers**
  - Applications use this set of Java interfaces for performing database operations
- **Majority of JDBC API is located in `java.sql` package**
  - DriverManager, Connection, ResultSet, DatabaseMetaData, ResultSetMetaData, PreparedStatement, CallableStatement and Types
- **Other advanced functionality exists in the `javax.sql` package**
  - DataSource

JDBC Driver

- Is an interpreter that translates JDBC method calls to vendor-specific database commands



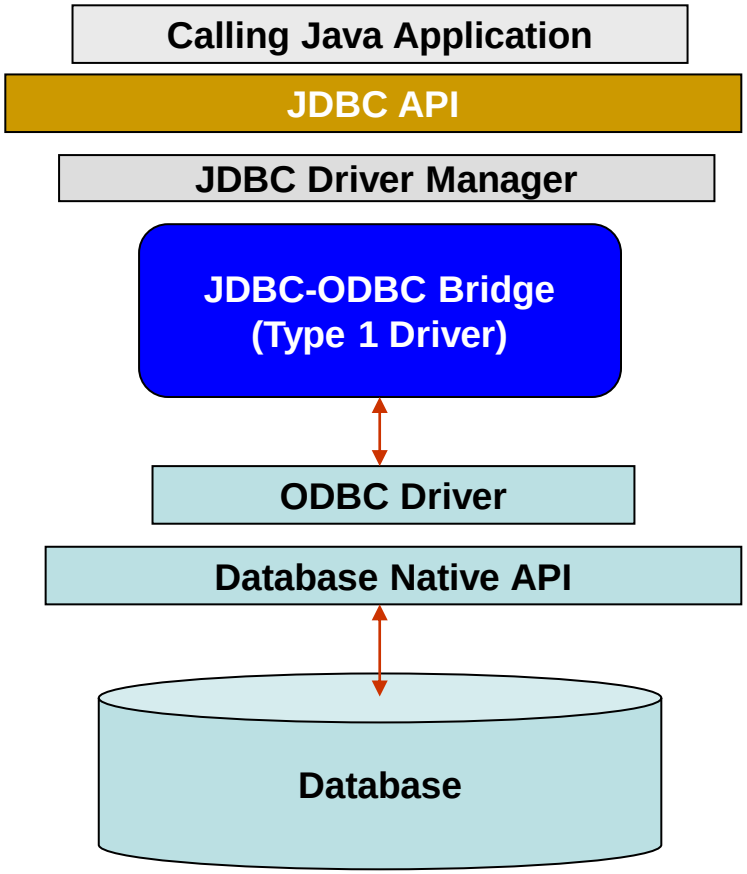
- Database specific implementation of JDBC interfaces
  - A JDBC driver implements the interfaces of java.sql package
  - Every database server has corresponding JDBC driver(s)

JDBC Driver

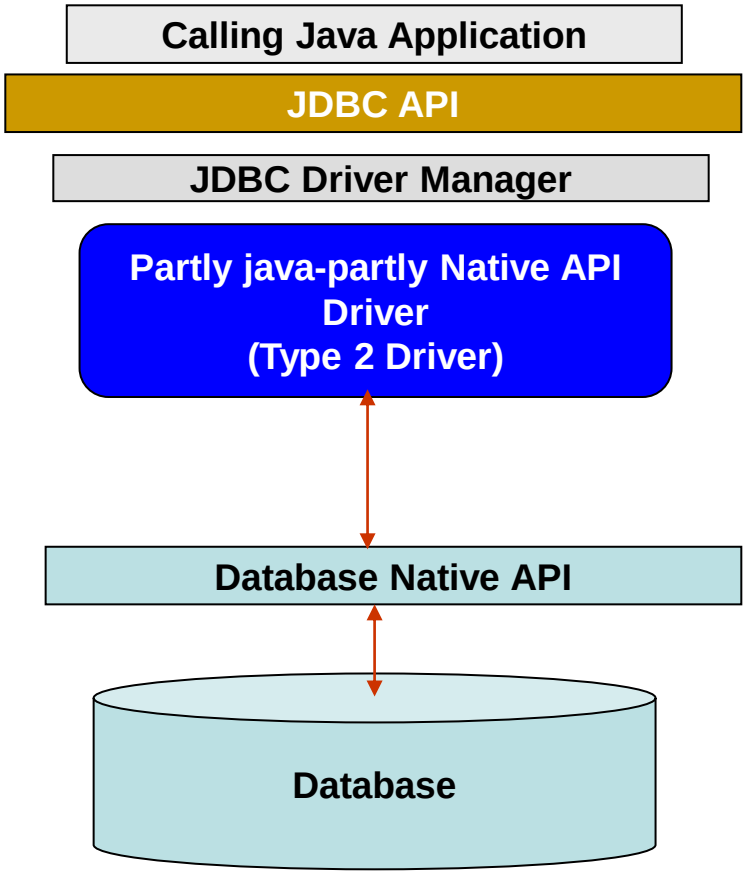
- Types of JDBC Driver

1. Type 1 Driver - the JDBC-ODBC bridge
2. Type 2 Driver – partly java-partly Native-API Driver
3. Type 3 driver - the Network-Protocol Driver
4. Type 4 - the Pure Java Driver

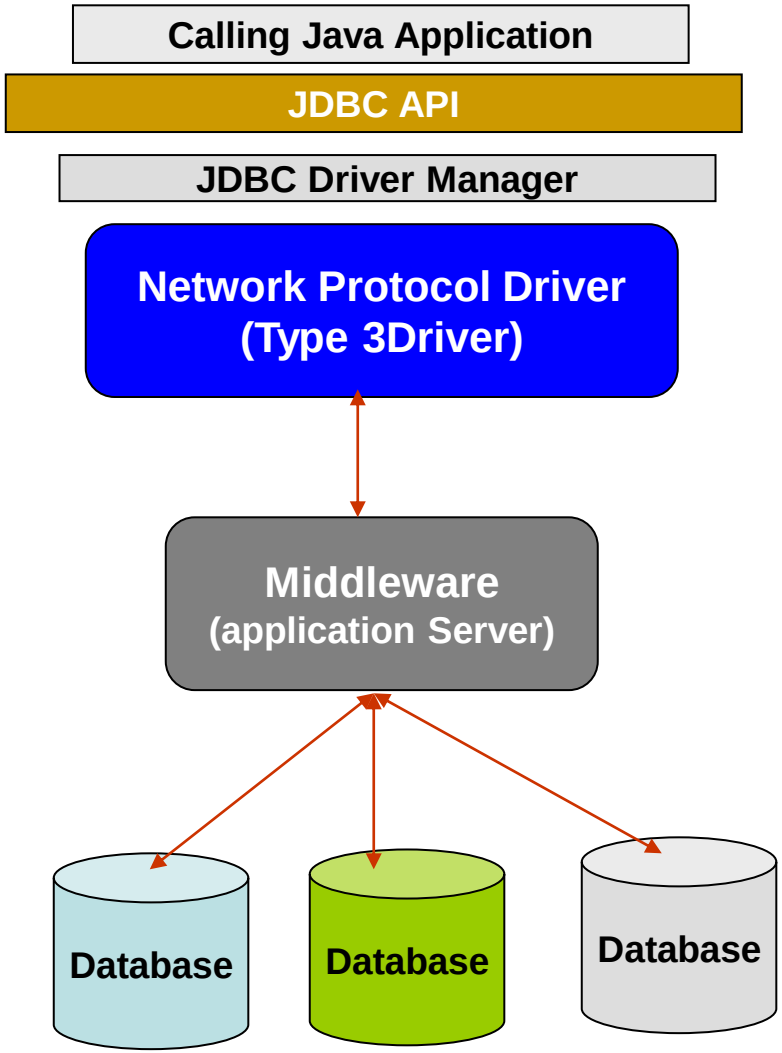
Type1 Driver



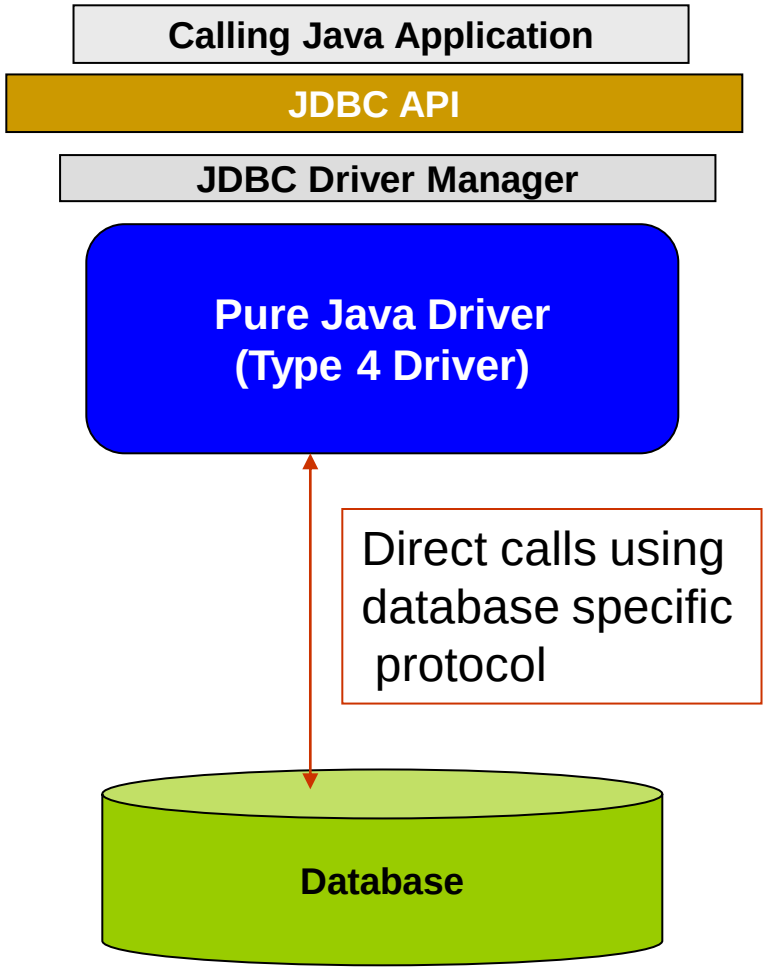
Type2 Driver



Type3 Driver



Type3 Driver



JDBC URLs

- Used to make a connection to the database
  - Can contain server, port, protocol etc…
- Syntax

```
jdbc:subprotocol_name:driver_dependant_dbname
```

- URL for Oracle thin driver

```
jdbc:oracle:thin:@machinename:1521:dbname
```

- URL for Pointbase

```
jdbc:mysql://localhost:3306/sampledb
```

JDBC URLs

- JDBC-ODBC driver (Type1)

```
jdbc:odbc:DSN
```

- OCI driver (Type2)

```
jdbc:oracle:oci8:@<TNSNAMES entry>
```

JDBC API

JDBC API

1. Important Interfaces of “java.sql” package
    1. Connection
    2. Statement
    3. PreparedStatement
    4. CallableStatement
    5. ResultSet
    6. ResultSetMetaData
    7. DatabaseMetaData
2. Important Classes of “java.sql” package
    1. DriverManager
    2. Types

Connection Interface

- Important methods

Return type	method
Statement	createStatement()
Statement	createStatement(int resultSetType,int resultSetConcurrency)
PreaparedStatement	prepareStatement(String sql)
PreparedStatement	prepareStatement(String sql,int resultSetType,int resultSetConcurrency)
CallableStatement	prepareCall(String sql);
CallableStatement	prepareCall(String sql, int resultSetType, int resultSetConcurrency)
void	close()
void	commit()
void	setAutocomit(boolean b)
void	rollback();

Statement interface

- The object used for executing a static SQL statement and obtaining the results produced by it.

Return type	method
ResultSet	executeQuery(String sql)
int	executeUpdate(String sql)
boolean	execute(String sql)
ResultSet	getResultSet()
void	close()
void	addBatch();
int[]	executeBatch()

PreparedStatement interface

- PreparedStatement extends Statement
- An object of PreparedStatement represents a precompiled SQL statement

Return type	method
ResultSet	executeQuery()
int	executeUpdate(l)
boolean	execute()
ResultSetMetaData	getMetaData()
void	close()
In addition to the above methods all setXXX methods are present in the PreparedStatement which are used to set values to the place holders during run time e.g. setString(int parameterindex,String value);	

CallableStatement interface

- CallableStatement extends PreparedStatement
- The interface used to execute SQL stored procedures and functions. JDBC provides a stored procedure SQL escape syntax that allows stored procedures to be called in a standard way for all RDBMSs
- In addition to the inherited methods ,it has following types of methods

Return type	method
void	registerOutParameter(int parameterIndex, int sqlType) This method has three overloaded forms
String	getString(int parameterIndex)
int	getInt(int parameterIndex)
double	getDouble(int parameterIndex)
long	getLong(int parameterIndex)
and Other getXXX methods …… (ref API doc)	

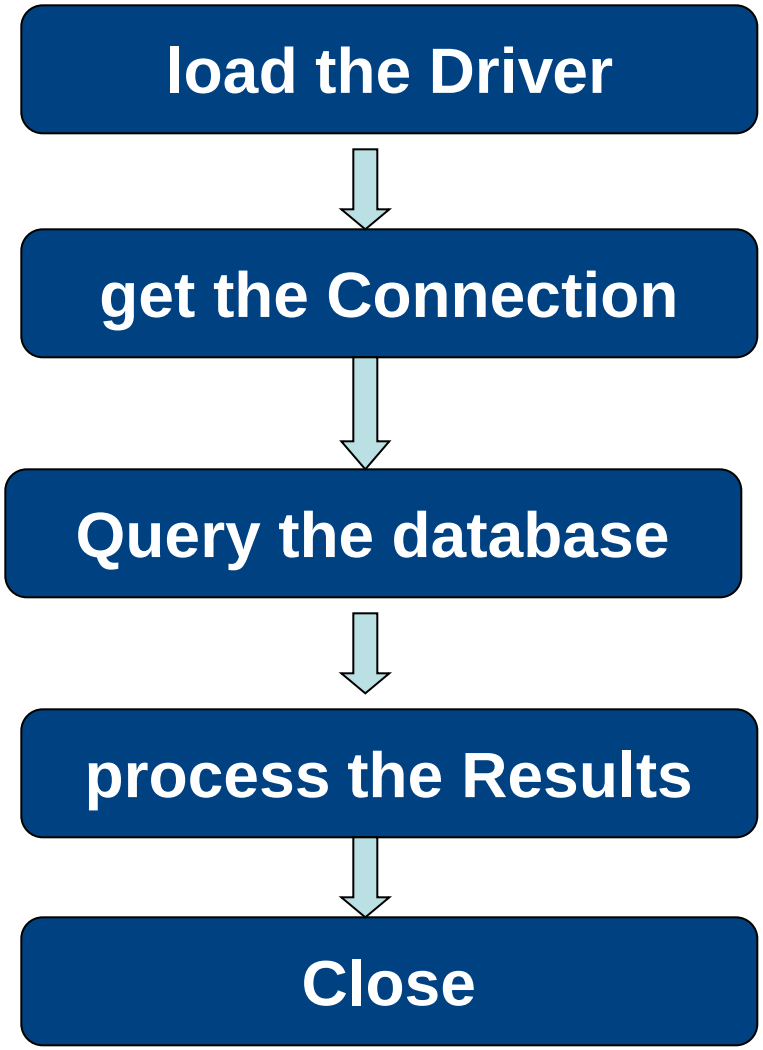
DriverManager class

- DriverManager Object provides the basic service for managing a set of JDBC drivers.

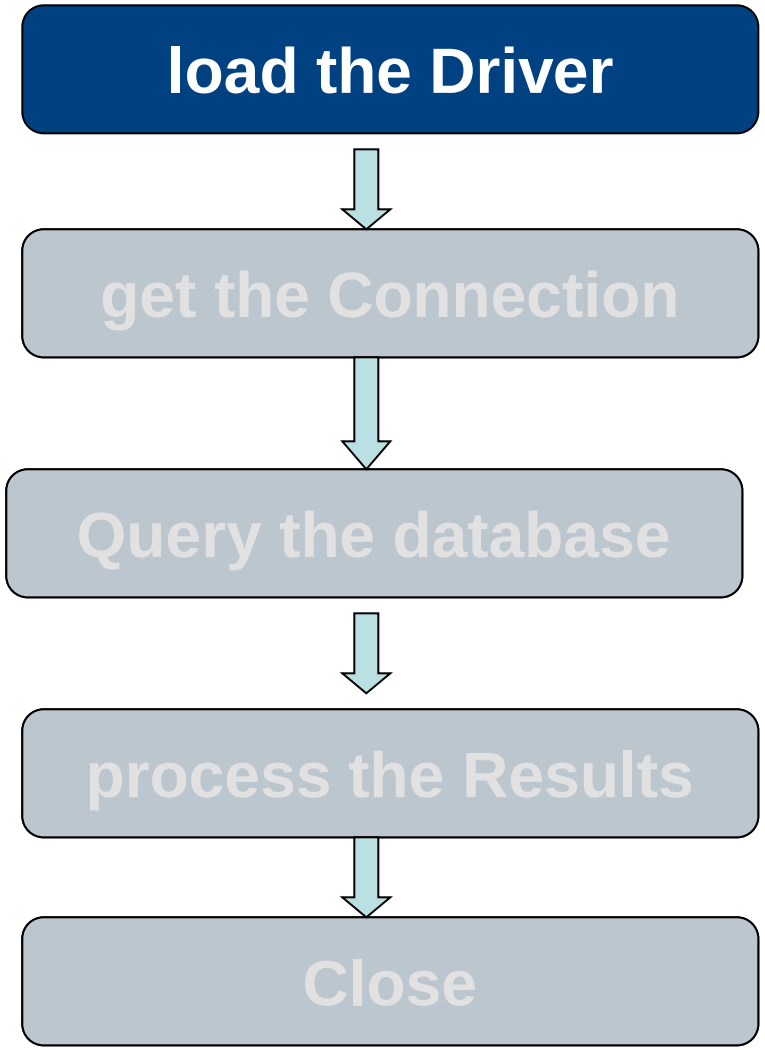
Return type	method
static void	registerDriver(Driver driver)
static Connection	getConnection(String url)
static Connection	getConnection(String url, Properties info)
static Connection	getConnection(String url, String uid, String pwd)
static void	deregisterDriver(Driver driver)

Step By Step Usage of JDBC API

Steps for JDBC



Steps for JDBC



Loading the Driver

- Register the Driver

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

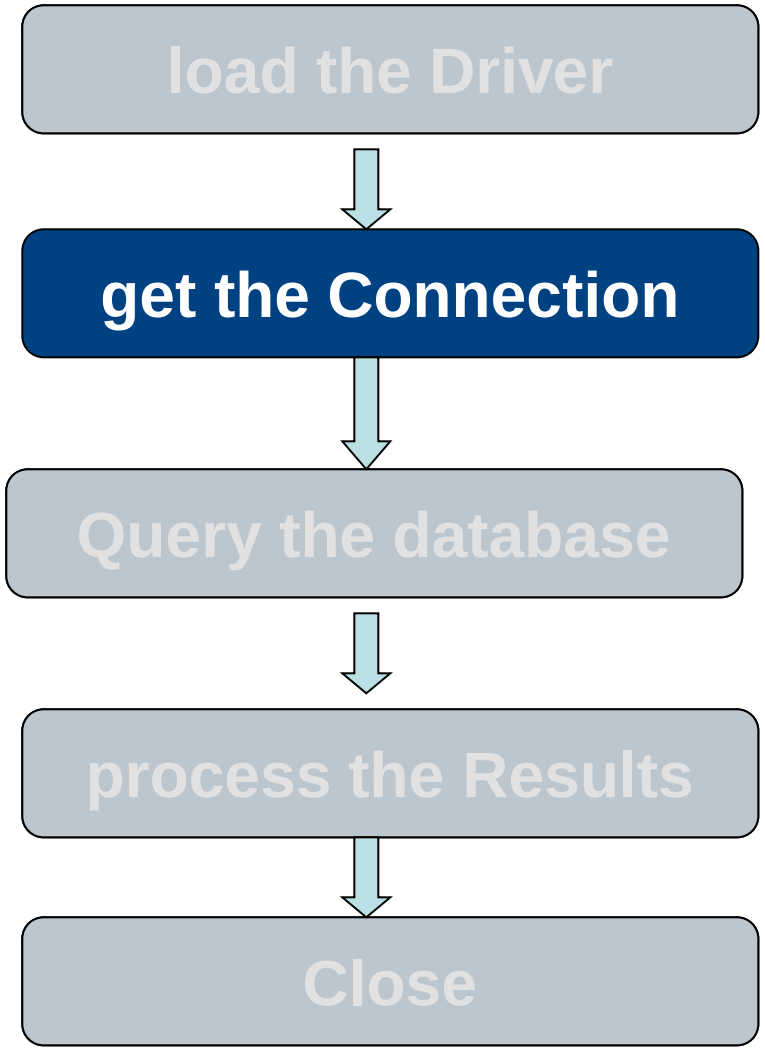
```
Class.forName( ″sun.jdbc.odbc.JdbcOdbcDriver");
```

OR

```
DriverManager.registerDriver(
            "oracle.jdbc.driver.OracleDriver");
```

```
DriverManager.registerDriver (
                    ″sun.jdbc.odbc.JdbcOdbcDriver");
```

Steps for JDBC



Connecting to the Database

- Connect to the database

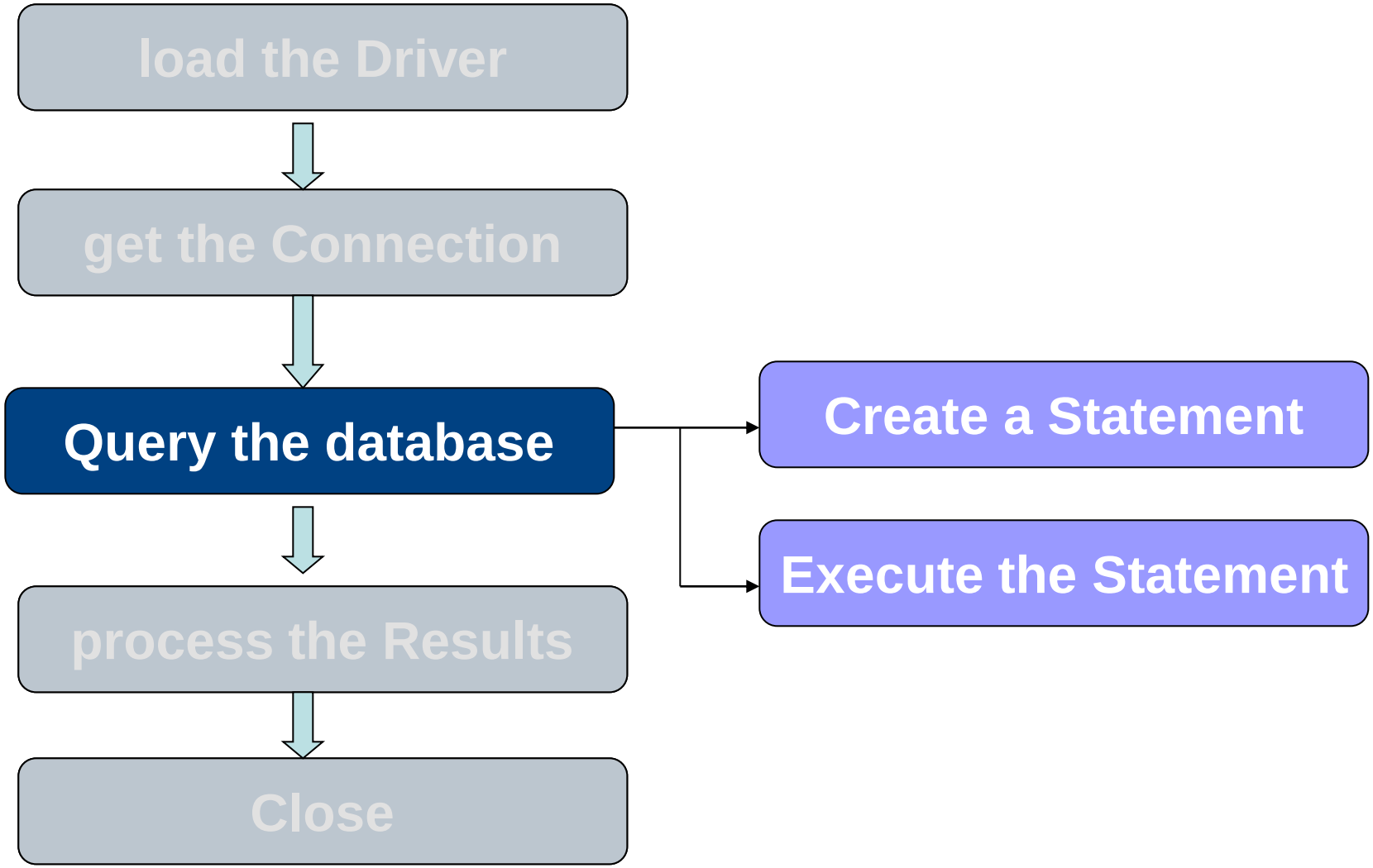
```
Connection Conn =
DriverManager.getConnection(jdbc_url,uid,pwd);
```

- Example code

```
Connection conn =
DriverManager.getConnection(jdbc:odbc:dsn,"scott","tiger");
```

```
Connection conn = DriverManager.getConnection
("jdbc:oracle:thin:@hostname:1521:orcl","scott","tiger");
```

Steps for JDBC



The Statement Object

- A ***`Statement`*** object sends your SQL statement to the database

- You need an active connection to create a JDBC ***`statement`***

- ***`Statement`*** has three methods to execute a SQL statement:
  - `executeQuery() for` QUERY statements
  - `executeUpdate()` for INSERT, UPDATE, DELETE, or DDL statements
  - `execute()` for either type of statement

Query the Database

1. Create an empty statement object

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

2. Execute the statement

```
ResultSet rset = stmt.executeQuery(statement);
int count = stmt.executeUpdate(statement);
boolean isquery = stmt.execute(statement);
```

Querying the Database: Examples

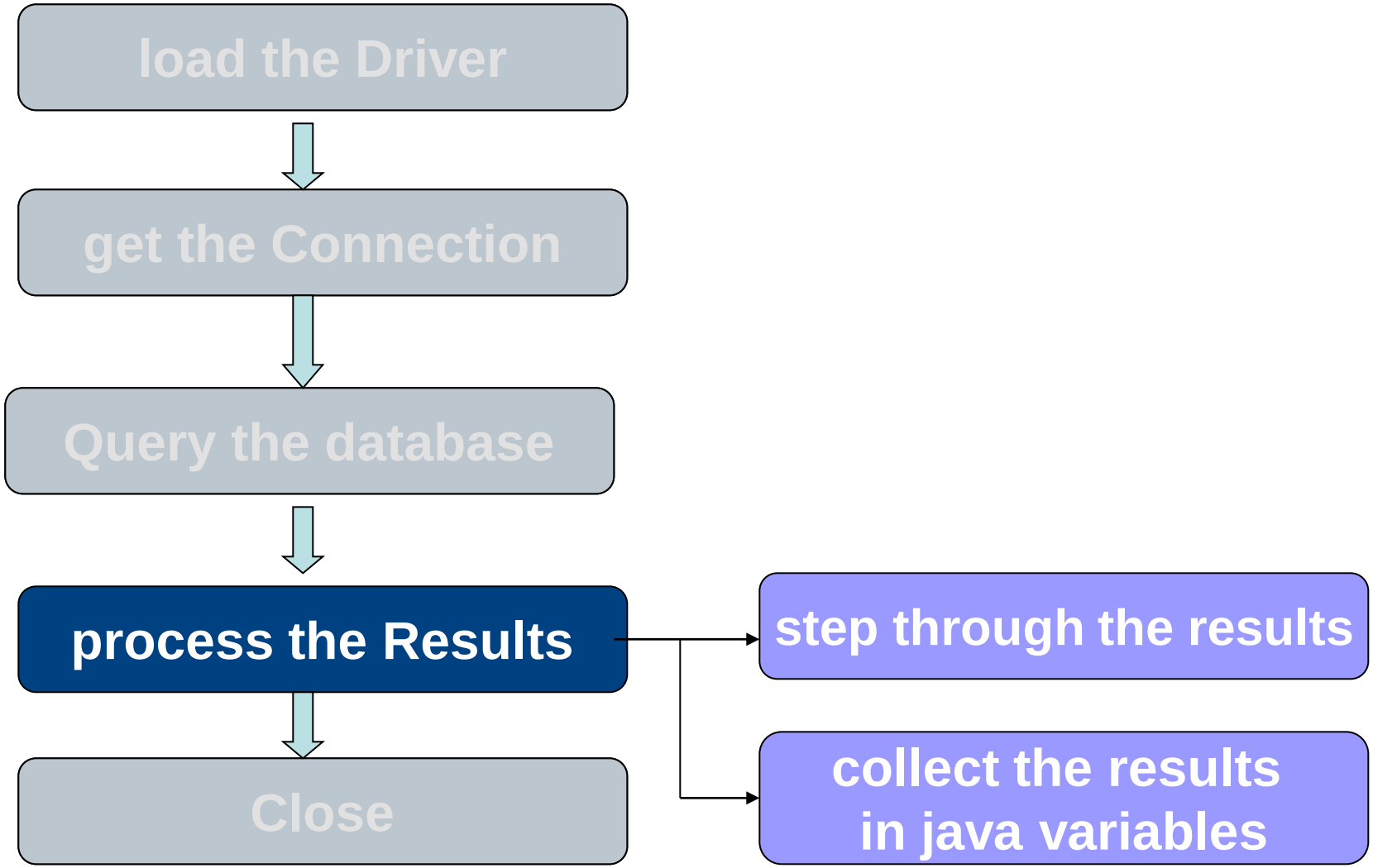
- Execute a select statement

```
Statement stmt = conn.createStatement();
ResultSet rset = stmt.executeQuery
  ("select Ename, Dept from EMP");
```

- Execute a delete statement

```
Statement stmt = conn.createStatement();
int rowcount = stmt.executeUpdate
  ("delete from EMP
      where ID = 2719");
```

Steps for JDBC



The ResultSet Object

- JDBC returns the results of a query in a **`ResultSet`** object
- A **`ResultSet`** maintains a cursor pointing to its current row of data
- Use `next()` to step through the result set row by row
- `getString()`, `getInt()`, and so on assign each value to a Java variable

How to Process the Results

1. Step through the result set

```
while (rset.next()) { … }
```

2. Use `getXXX()` to get each column value

```
String val =
rset.getString(colname);
```

```
String val =
rset.getString(colIndex);
```

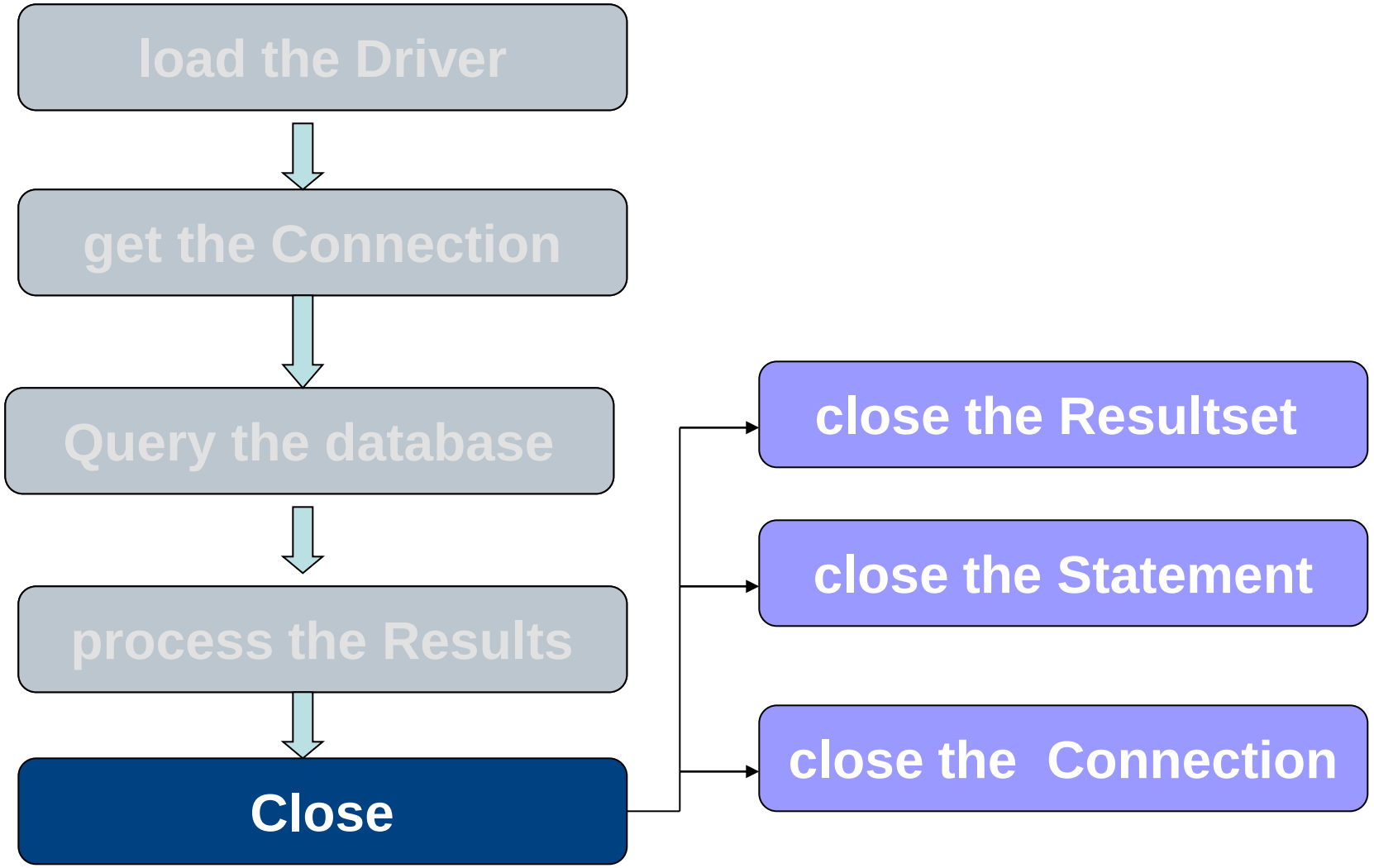
```
while (rset.next()) {
  String name = rset.getString ("ENAME");
  String supervisor = rset.getString ("DEPT");
  … // Process or display the data
}
```

How to Handle SQL Null Values

- Java primitive types cannot have `null` values
- Do not use a primitive type when your query might return a SQL null
- Use `ResultSet.wasNull()` to determine whether a column has a null value

```
while (rset.next()) {
   String year = rset.getString("YEAR");
   if (rset.wasNull() {
      … // Handle null value}
…}
```

Steps for JDBC



How to Close the Connection

1. Close the **ResultSet** object

```
rset.close();
```

2. Close the **Statement** object

```
stmt.close();
```

3. Close the **connection** (not necessary for server-side driver)

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

Connecting to Database(Type1)

```
import java.sql.*;
class  UsingType1
{
    public static void main(String[] args) throws Exception
    {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

            System.out.println("Driver is loaded...");

            Connection con = DriverManager.getConnection("jdbc:odbc:datasource");
            Statement stmt= con.createStatement();
            ResultSet rs = stmt.executeQuery("select * from pemp");
    while (rs.next())
    {
            int id= rs.getInt(1);
            String name=rs.getString(2);
            int sal = rs.getInt(3);

            System.out.println(id+" "+name+" "+sal);
    }
    }
}
```

Connecting to Database(Type2)

```
import java.sql.*;
class  UsingType2
{
    public static void main(String[] args) throws Exception
    {
        Class.forName(“oracle.jdbc.driver.OracleDriver");

        System.out.println("Driver is loaded...");

        Connection con =
        DriverManager.getConnection("jdbc:oracle:oci8:@“,”scott”,”tiger”);
        Statement stmt= con.createStatement();
        ResultSet rs = stmt.executeQuery("select * from pemp");
    while (rs.next())
        {
            int id= rs.getInt(1);
            String name=rs.getString(2);
            int sal = rs.getInt(3);
            System.out.println(id+" "+name+" "+sal);
    }
    }
}
```

Connecting to Database(Type4)

```
import java.sql.*;
class  UsingType4
{
    public static void main(String[] args) throws Exception
    {
        Class.forName(“oracle.jdbc.driver.OracleDriver");

        System.out.println("Driver is loaded...");

        Connection con =
        DriverManager.getConnection("jdbc:oracle:thin:@hostname:1521:ocl“,
                                    ”scott”,”tiger”);
        Statement stmt= con.createStatement();
        ResultSet rs = stmt.executeQuery("select * from pemp");
    while (rs.next())
        {
            int id= rs.getInt(1);
            String name=rs.getString(2);
            int sal = rs.getInt(3);
            System.out.println(id+" "+name+" "+sal);
    }
    }
}
```

The `DatabaseMetaData` Object

- The `Connection` object can be used to get a `DatabaseMetaData` object
- This object provides more than 100 methods to obtain information about the database

How to Obtain Database Metadata

1. Get the `DatabaseMetaData` object

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

2. Use the object's methods to get the metadata

```
DatabaseMetaData dbmd = conn.getMetaData();
String  s1 = dbmd getURL();
String  s2 = dbmd.getSQLKeywords();
boolean b1 = dbmd.supportsTransactions();
boolean b2 = dbmd.supportsSelectForUpdate();
```

The `ResultSetMetaData` Object

- The `ResultSet` object can be used to get a `ResultSetMetaData` object

- `ResultSetMetaData` object provides metadata, including:
  - Number of columns in the result set
  - Column type
  - Column name

How to Obtain Result Set Metadata

1. Get the `ResultSetMetaData` object

```
ResultSetMetaData rsmd = rset.getMetaData();
```

2. Use the object's methods to get the metadata

```
ResultSetMetaData rsmd = rset.getMetaData();
for (int i = 1; i <= rsmd.getColumnCount(); i++) {
  String colname = rsmd.getColumnName(i);
  int coltype = rsmd.getColumnType(i);
  …
}
```

Mapping Database Types to Java Types

`ResultSet` maps database types to Java Types types.

```
ResultSet rset = stmt.executeQuery
   ("select ID, DATE_OF_JOIN, SUPERVISOR
   from EMP");

int id = rset.getInt(1);
Date rentaldate = rset.getDate(2);
String status = rset.getString(3);
```

Col Name	Type
ID	NUMBER
DATE_OF_JOIN	DATE
SUPERVISOR	VARCHAR2

The `PreparedStatement` Object

- A `PreparedStatement` object holds precompiled SQL statements
- Use this object for statements you want to execute more than once
- A prepared statement can contain variables that you supply each time you execute the statement

How to Create a Prepared Statement

1. Register the driver and create the database connection
2. Create the prepared statement, identifying variables with a question mark (?)

```
PreparedStatement pstmt =
  conn.prepareStatement("update STUDENT
  set SUPERVISOR = ? where ID = ?");
```

```
PreparedStatement pstmt =
  conn.prepareStatement("select SUPERVISOR from
  STUDENT where ID = ?");
```

How to Execute a Prepared Statement

1. Supply values for the variables

```
pstmt.setXXX(index, value);
```

2. Execute the statement

```
pstmt.executeQuery();
pstmt.executeUpdate();
```

```
PreparedStatement pstmt =
  conn.prepareStatement("update STUDENT
  set SUPERVISOR = ? Where ID = ?");
pstmt.setString(1, "OUT");
pstmt.setInt(2, id);
pstmt.executeUpdate();
```

The `CallableStatement` Object

- A `CallableStatement` object holds parameters for calling stored procedures
- A callable statement can contain variables that you supply each time you execute the call
- When the stored procedure returns, computed values (if any) are retrieved through the `CallableStatement` object

How to Create a CallableStatement

- Register the driver and create the database connection

- Create the callable statement, identifying variables with a question mark (?)

```
CallableStatement cstmt =
   conn.prepareCall("{call " +
   ADDITEM +
   "(?,?,?)}");
  cstmt.registerOutParameter(2,Types.INTEGER);
  cStmt.registerOutParameter(3,Types.DOUBLE);
```

How to Execute a CallableStatement

1. Set the input parameters

```
cstmt.setXXX(index, value);
```

2. Execute the statement

```
cstmt.execute(statement);
```

3. Get the output parameters

```
var = cstmt.getXXX(index);
```

ResultSet Object in JDBC 2.1

ResultSet in JDBC 2.1

- Before JDBC 2.1 the ResultSet Object was forward-only scrollable
  - we could traverse the ResultSet only using next() method
- JDBC 2.1 provides more flexible means of accessing ResulSets
- Enhancements of JDBC 2.1 are:
  1. Scrollable ResultSets
  2. Scroll sensitivity
  3. Updatable ResultSets

Scrollable ResultSets

- ResultSet objects have the ability to move the cursor backwards

- Support absolute positioning of the cursor at a particular row in the ResultSet

- The Three types of ResultSets (**`java.sql.ResultSet`**)
  1. **TYPE_FORWARD_ONLY**
     - supports forward scrolling only
  2. **TYPE_SCROLL_INSENSITIVE**
     - scrollable but generally not sensitive to changes made by others.
  3. **TYPE_SCROLL_SENSITIVE**
     1. scrollable and generally sensitive to changes made by others.

Scrollable ResultSet

- **Important methods for Scrollable ResultSet**

```
public boolean isBeforeFirst() throws SQLException
public boolean isAfterLast() throws SQLException
public boolean isFirst() throws SQLException
public boolean isLast() throws SQLException

public void beforeFirst() throws SQLException
public void afterLast() throws SQLException

public void first() throws SQLException
public void last() throws SQLException
public void absolutet(int row) throws SQLException
public void relative(int row) throws SQLException
public void previous() throws SQLException
```

Updatable ResultSet

- By Default the ResultSet (s) are read only
  - contents of the resultset cannot be changed
- JDBC 2.1 also introduces Updatable resultsets
  - updating a resultset also updates the original data corresponding to the resultset
- The java.sql.ResultSet interface specifies two constants
  1. CONCUR_READ_ONLY
     - we cannot insert, update or delete rows in the resultset
  2. CONCUR_UPDATABLE
     - we can insert, update or delete rows in the resultset

Updatable ResultSet

- Updating a row
  - **`java.sql.ResultSet`** interface provides a set of ***`updateXXX()`*** methods to update the current row in the table
    - e.g. updateDouble(), updateFloat() etc
- Deleting a row
  - ***`deleteRow()`*** method can be used to delete the current row which also deletes the row in the database
- Inserting a row
  - **`moveToInsertRow()`** and then **`updateXXX()`** `and` finally **`insertRow();`**

Batch Update

- Support from JDBC 2.1 Specification
- Allows multiple update statements (INSERT,UPDATE,DELETE) in single request to the database
- Batching large number of statements results in significant performance gain
- Two important methods of Statement interface for batch update
  - addBatch(String SQL statement)
  - executeBatch()

Using Transactions

- The server-side driver does not support autocommit mode
- With other drivers:
  - New connections are in autocommit mode
  - Use *conn*`.setAutoCommit(false)` to turn autocommit off
- To control transactions when you are not in autocommit mode:
  - `conn.commit()`: Commit a transaction
  - `conn.rollback()`: Roll back a transaction

Prepared & Callable Statements

What Are They?

- PreparedStatement
  - SQL is sent to the database and compiled or prepared beforehand
- CallableStatement
  - Executes SQL Stored Procedures

PreparedStatement

- The contained SQL is sent to the database and compiled or prepared beforehand
- From this point on, the prepared SQL is sent and this step is bypassed. The more dynamic Statement requires this step on every execution.
- Depending on the DB engine, the SQL may be cached and reused even for a different PreparedStatement and most of the work is done by the DB engine rather than the driver

PreparedStatement cont.

- A PreparedStatement can take IN parameters, which act much like arguments to a method, for column values.

- PreparedStatements deal with data conversions that can be error prone in straight ahead, built on the fly SQL handling quotes and dates in a manner transparent to the developer

CallableStatement

- The interface used to execute SQL stored procedures
- A stored procedure is a group of SQL statements that form a logical unit and perform a particular task
- Stored procedures are used to encapsulate a set of operations or queries to execute on a database server.

This is test