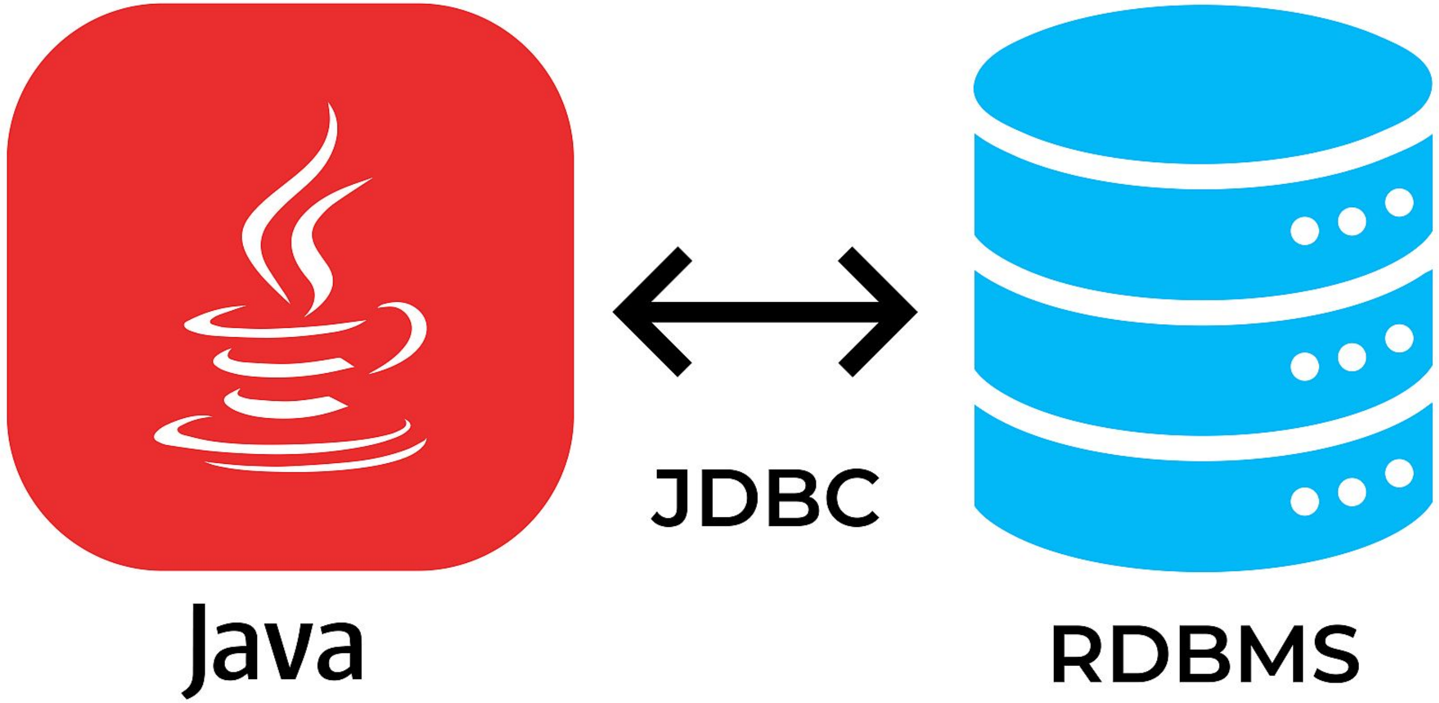
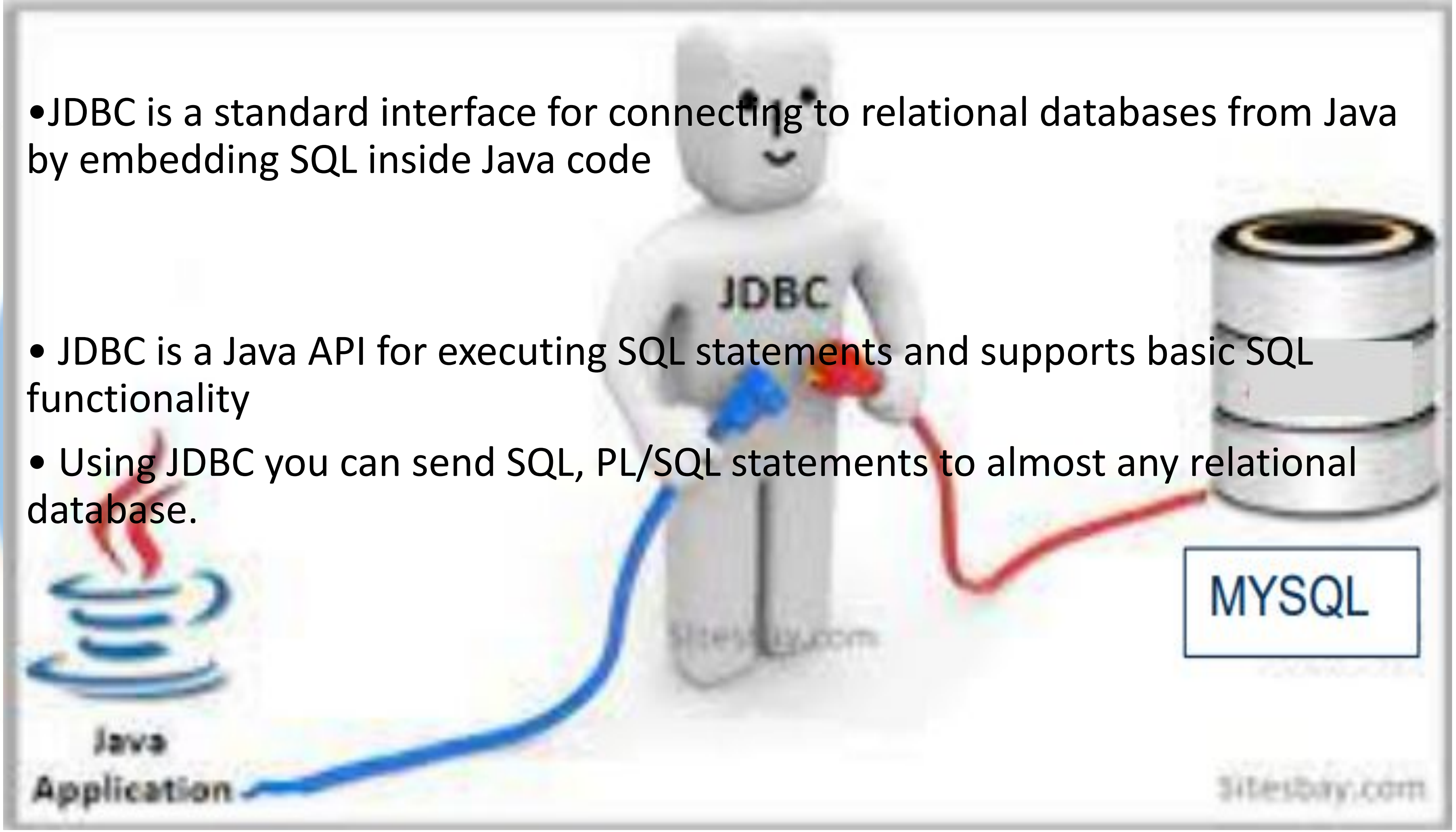
JDBC

Java DataBase Connectivity


Java
JDBC
RDBMS

Contents



•JDBC is a standard interface for connecting to relational databases from Java by embedding SQL inside Java code

• JDBC is a Java API for executing SQL statements and supports basic SQL functionality

• Using JDBC you can send SQL, PL/SQL statements to almost any relational database.

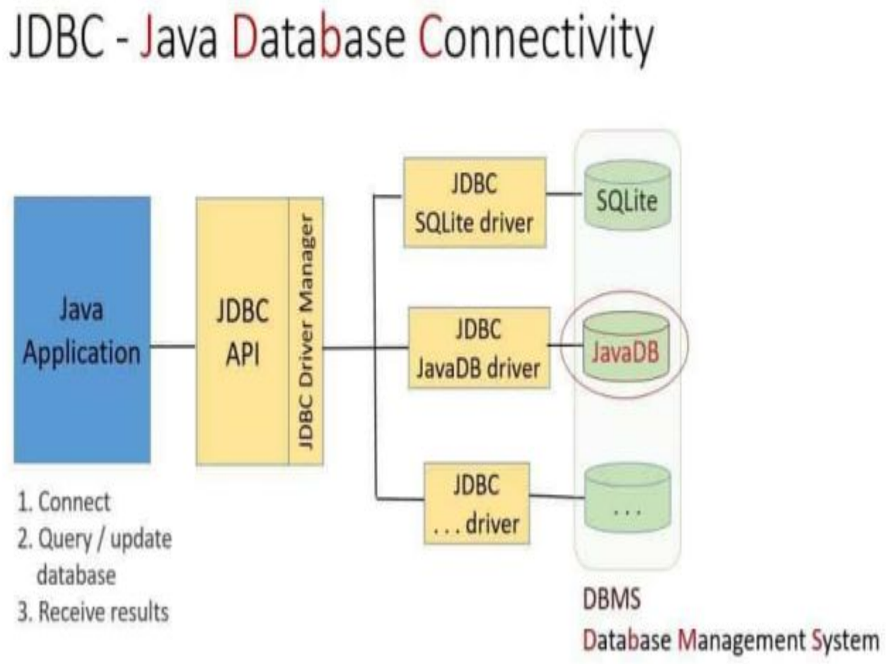
JDBC Architecture

- Application
  - Uses java.sql API to retrieve/query a database
- Database
  - A repository system for organizing data in a structured way
- Database Driver
  - A separate entity which provides interface between the Application and Database.

Types of JDBC Drivers

JDBC-ODBC Bridge driver (Bridge)

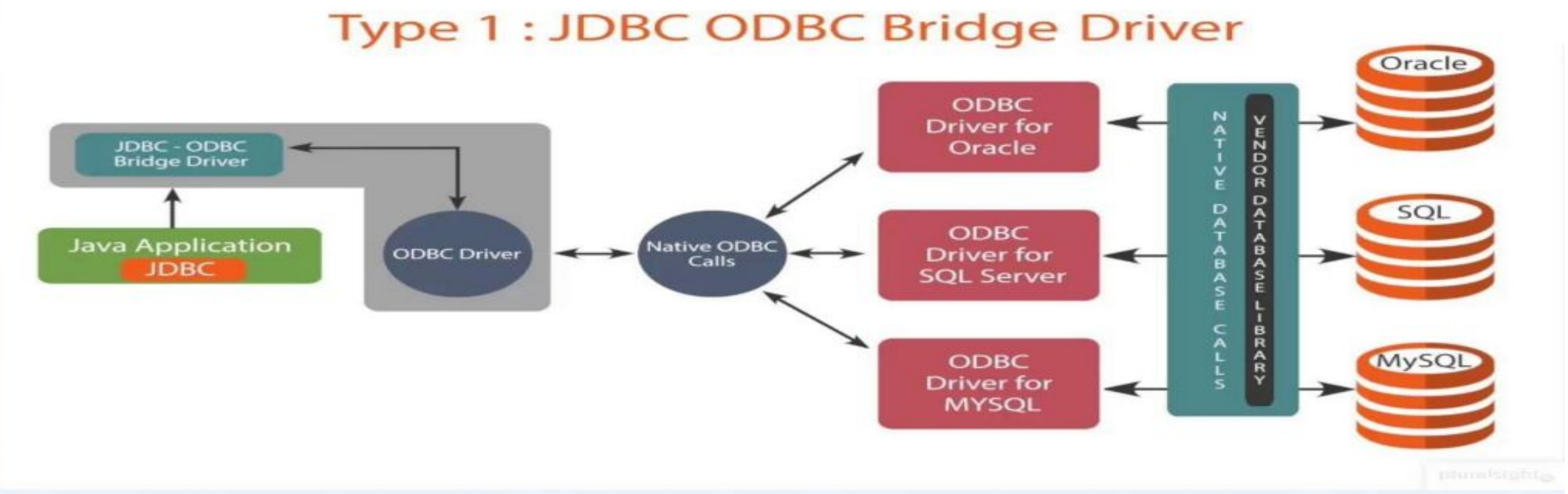
- Native-API/partly Java driver (Native)
- All Java/Net-protocol driver (Middleware)
- All Java/Native-protocol driver (Pure)



Type 1: JDBC-ODBC Bridge Driver

Translates all JDBC calls into ODBC calls and sends them to the ODBC driver

- Advantage
  - The JDBC-ODBC Bridge allows access to almost any database, since the database's ODBC drivers are already available



Disadvantage

– Type 1 drivers are not portable

– Performance

- very Slow

– Client requires ODBC installation

– Not good for Web

Type 2: Native-API/partly Java Driver

Converts JDBC calls into databasespecific calls

• The driver is specific to a particular database. Example: Oracle will have oracle native api.

• Advantage

– Better performance

- Less layers of communication and native drivers

• Disadvantage

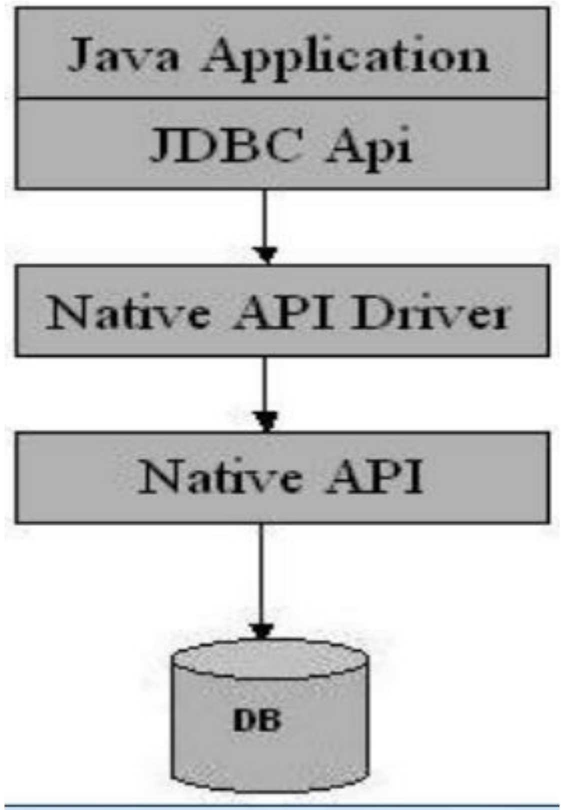
– Native API must be installed in the Client System

-hence cannot be used for internet

– Portability issue (not written in Java)

– Native driver's are database dependent

– Not thread safe



Type 3: All Java/Net-protocol Driver

Requests are passed through the network to the middle-tier server.

- The middle-tier translates the request to the database.

• Advantage

– Driver is server-based, so there is no need for any vendor database library to be present on client machines.

– Portable and suitable for web

– Portability, performance, and scalability can be optimized

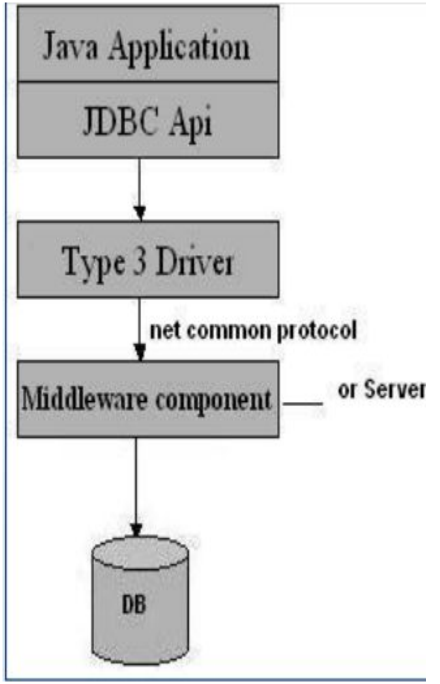
– Supports features such as caching, load balancing and advanced system administration such as logging and auditing

– access to multiple databases using one driver

• Disadvantage

– Requires another server application to install and maintain

– Traversing the recordset may take longer, since the data comes through the backend server



Type 4: Native-protocol/all-Java Driver

- Uses java networking libraries to communicate directly with the database server .
- **Advantage**

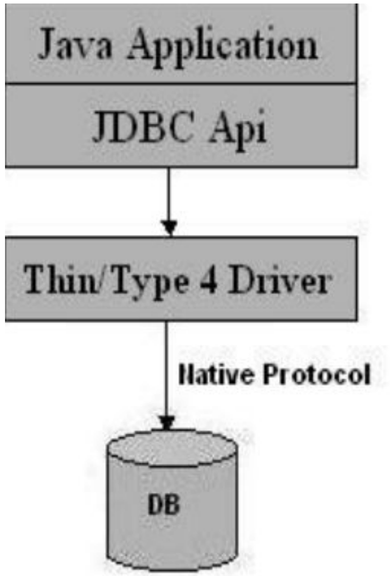
– Platform independent since written in Java

– Performance quite good

– No special software on the client or server

- **Disadvantage**

– Different driver for each database

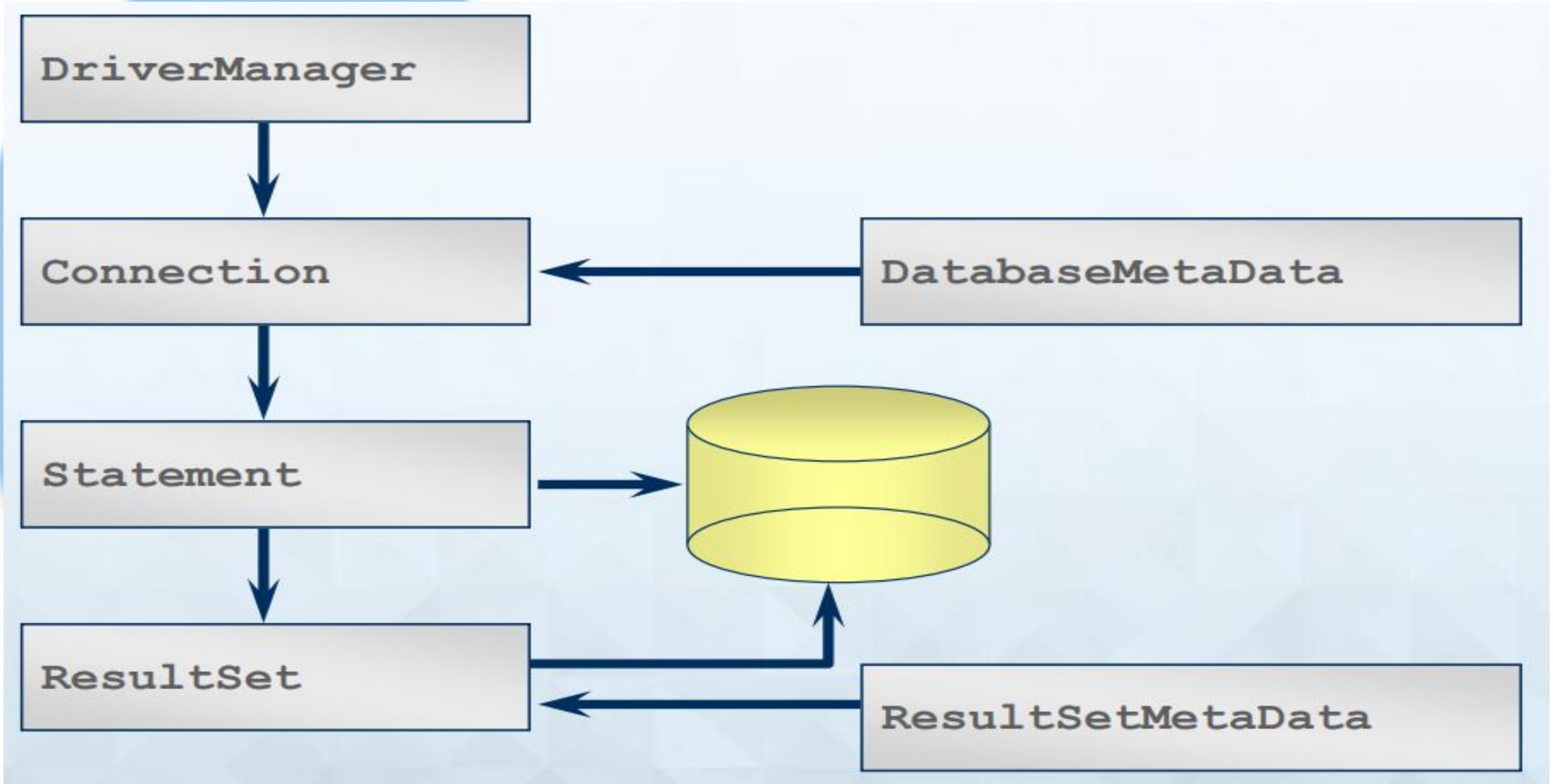


Java Database Connectivity Steps

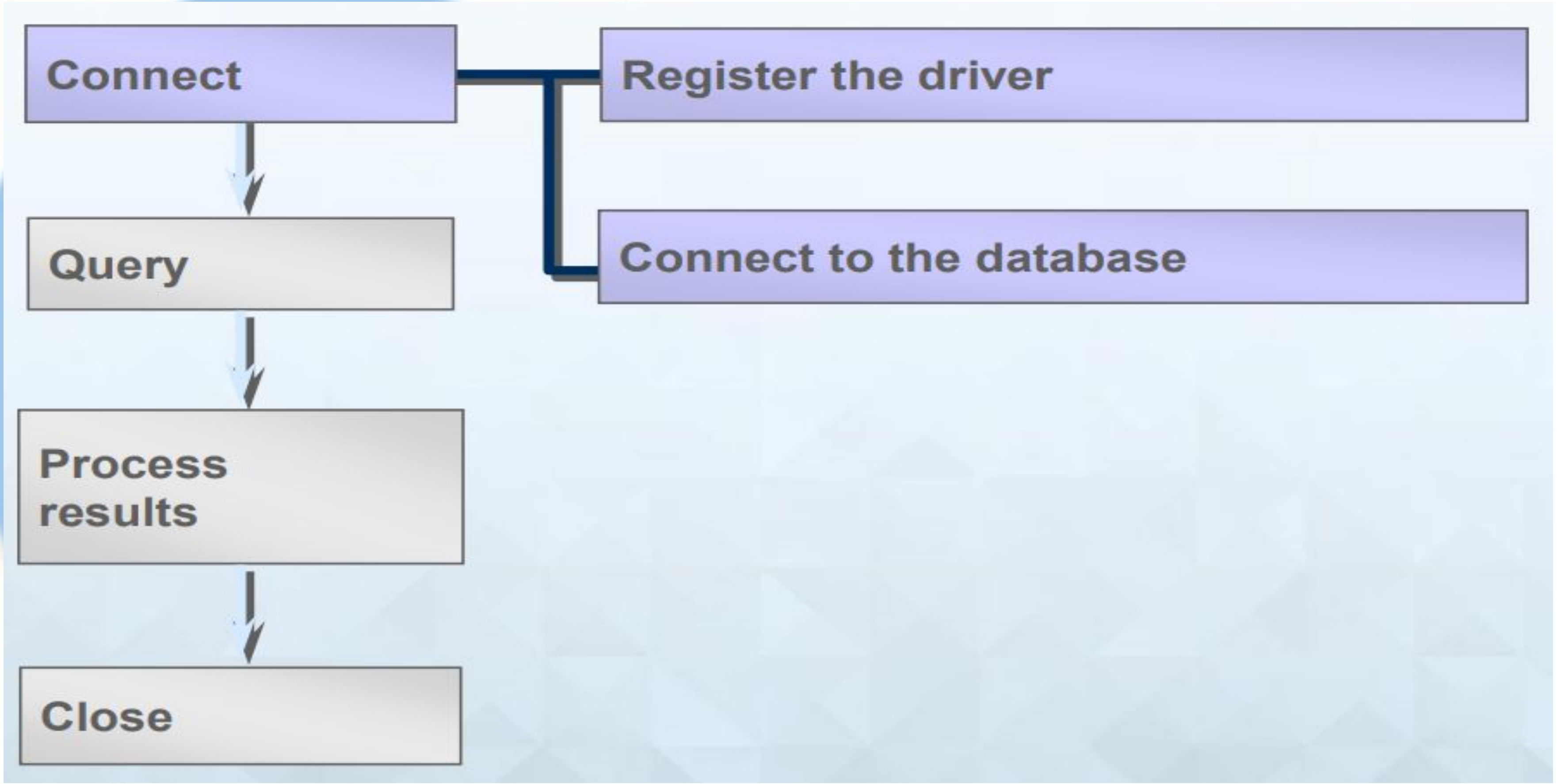
Java Database Connectivity Steps

- Import the java.sql package.
- Create a data source name using ODBC
- Register the Driver
- Establish a Connection to the database
- Create a Statement object
- Execute SQL Query statement(s)
- Retrieve the ResultSet Object
- Retrieve record/field data from ResultSet object for processing
- Close ResultSet Object
- Close Statement Object
- Close Connection Object

Java Database Connectivity Steps

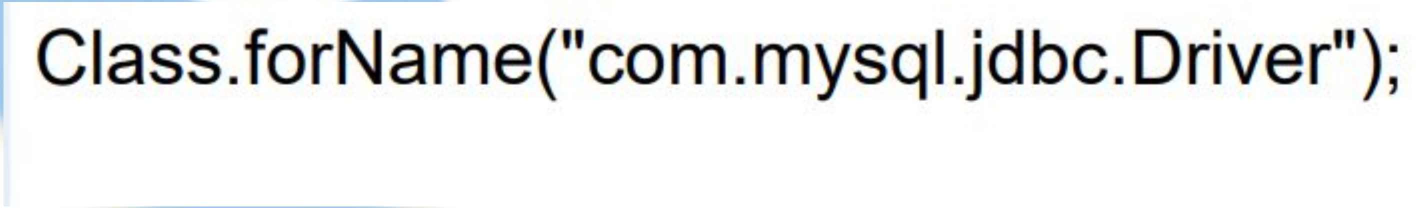


Stage1: Connection establishment



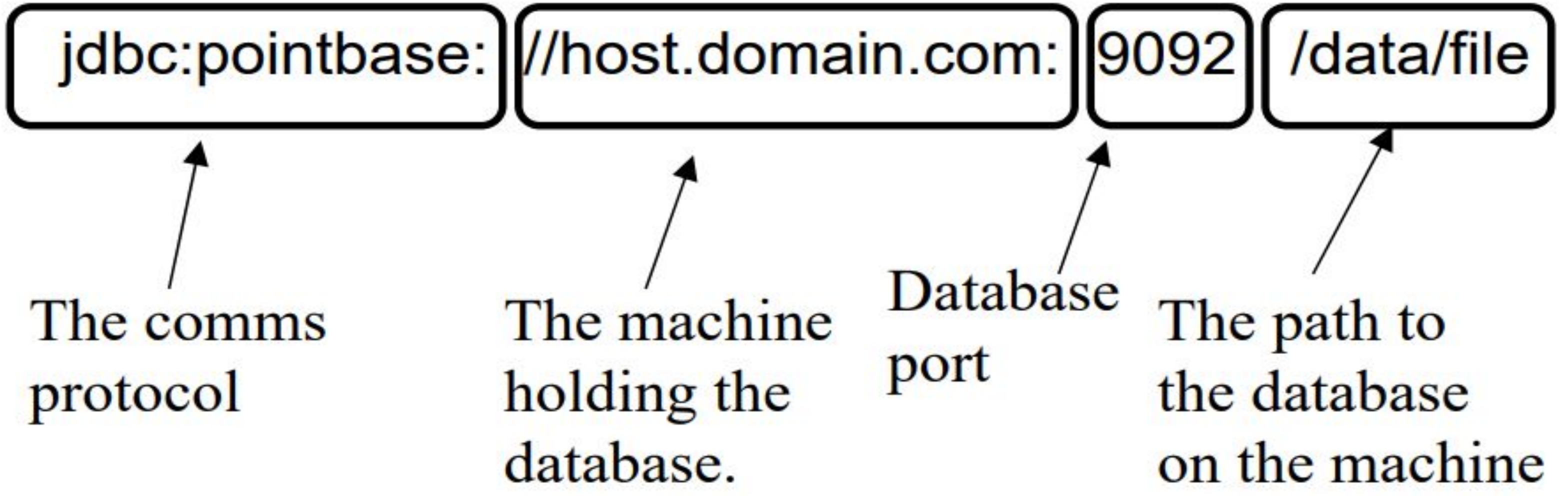
Register the Driver

- Load the driver class by calling Class.forName() with the Driver class name as an argument.
- The Driver class creates an instance of itself.
- The return type of the Class.forName(String ClassName) method is "Class". Class is a class in java.lang package.



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

Database URL



Eg: jdbc:mysql://localhost:3306/myDb

Establish a Connection

JDBC DriverManager class defines objects which can connect Java applications to a JDBC driver.

• The getConnection() method is used to establish a session/connection to a specific database

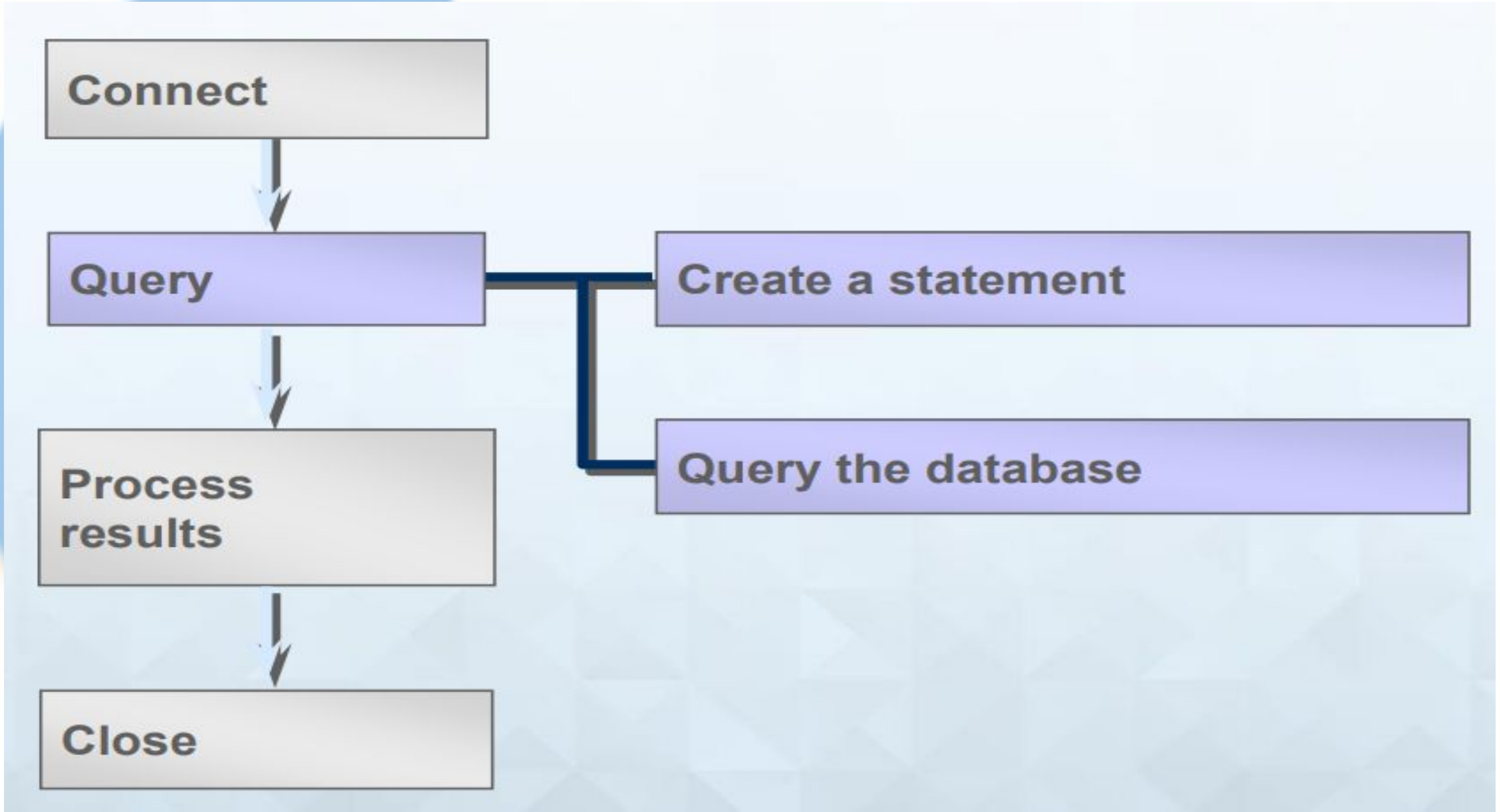
• An application can have one or more connections with a single database, or it can have many connections with different databases.

• A Connection object provides metadata i.e. information about the database, tables, and fields. It also contains methods to deal with transactions.

```
Connection conn =
DriverManager.getConnection(URL,userid,password);
```

```
Connection con=DriverManager.getConnection( "jdbc:
mysql://localhost:3306/mydb","root",“password123");
```

Stage 2: Query construction



Create a Statement

A Statement object sends your SQL statement to the database

- You need an active connection to create a JDBC statement

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

Types of Statement

- **Statement**
  - Execute simple sql queries without parameters.

```
Statement createStatement()
```

- **Prepared Statement**

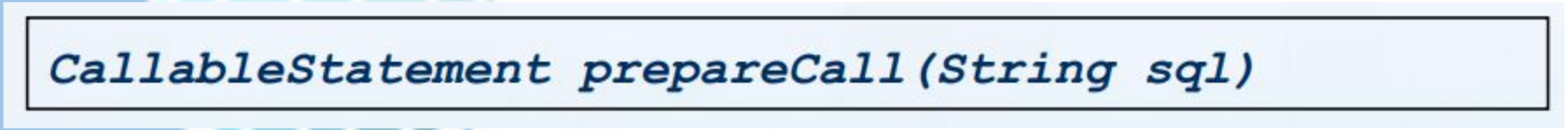
– Execute precompiled sql queries with or without parameters.

– PreparedStatement objects are precompiled SQL statements.

```
PreparedStatement prepareStatement(String sql)
```

Types of Statement

- **Callable Statement**
  - Execute a call to a database stored procedure.



```
CallableStatement prepareCall(String sql)
```

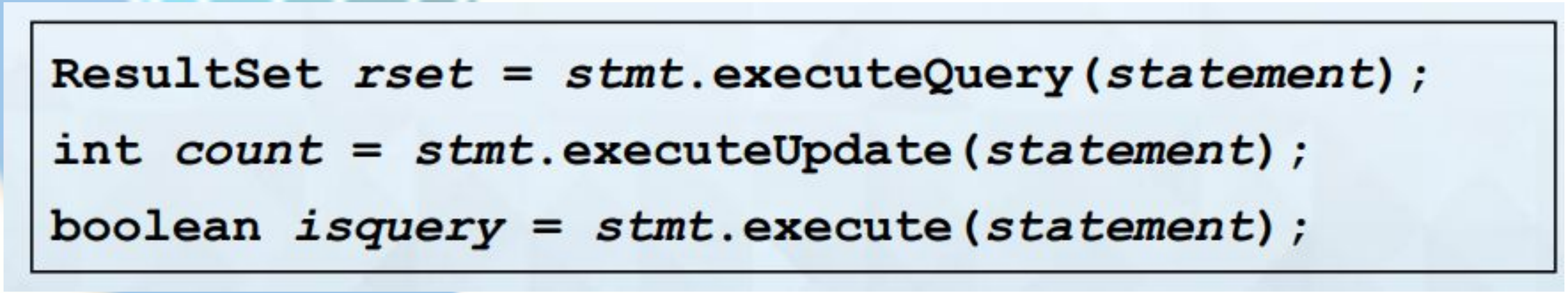
Query the Database

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



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

Query the Database: Examples

- Execute a select statement

```
Statement stmt = conn.createStatement();
ResultSet rset = stmt.executeQuery
   ("select RENTAL_ID, STATUS from ACME_RENTALS");
```

- Execute a delete statement

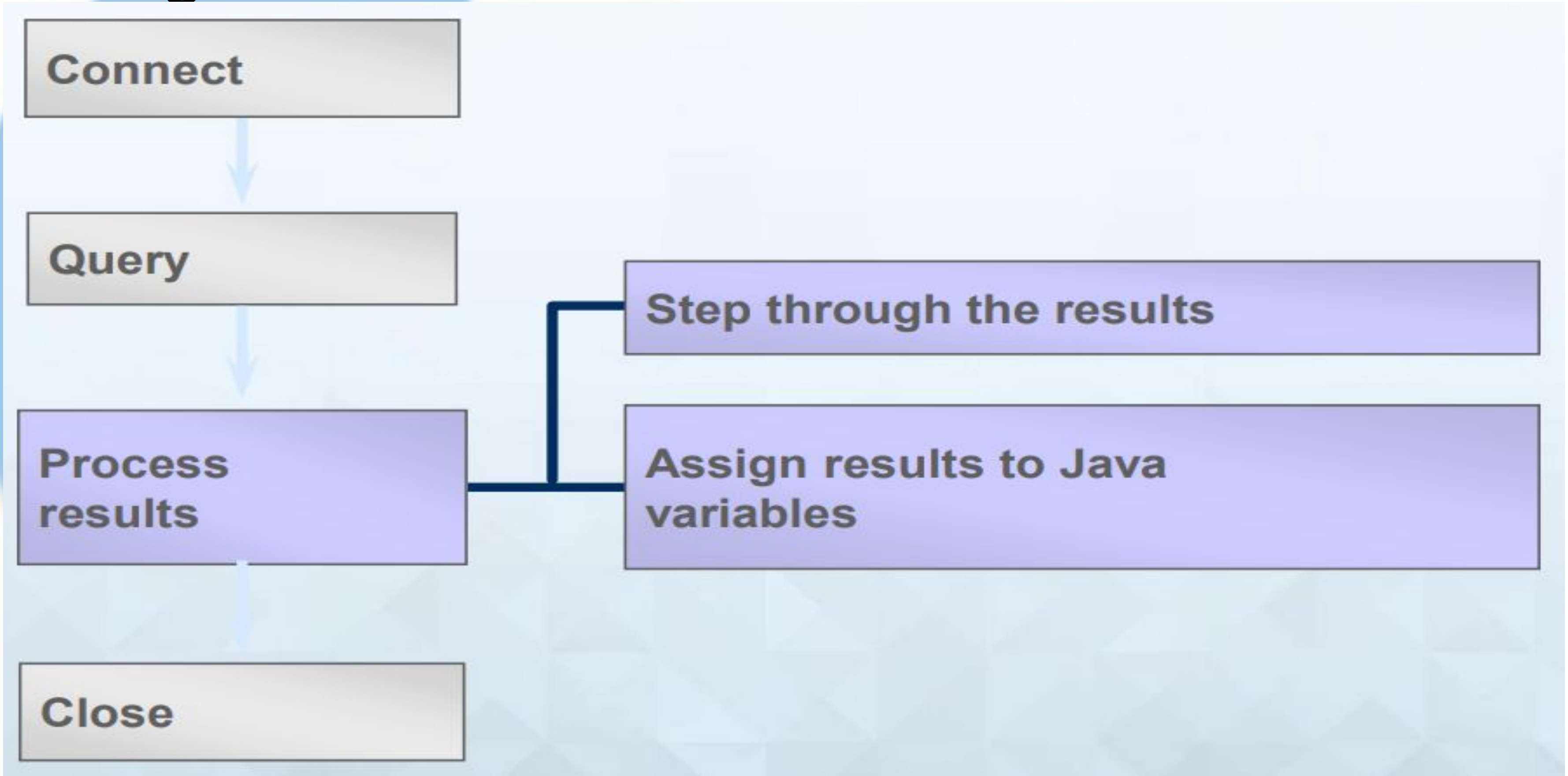
```
Statement stmt = conn.createStatement();
int rowcount = stmt.executeUpdate
   ("delete from ACME_RENTAL_ITEMS
       where rental_id = 1011");
```

Statement Object

- The `Statement` object provides a workspace where SQL queries can be created, executed, and results collected.
- e.g.

```
Statement st =
              conn.createStatement():
ResultSet rs = st.executeQuery(
        " select * from Authors" );
        :
st.close();
```

Stage 3: Process the Results



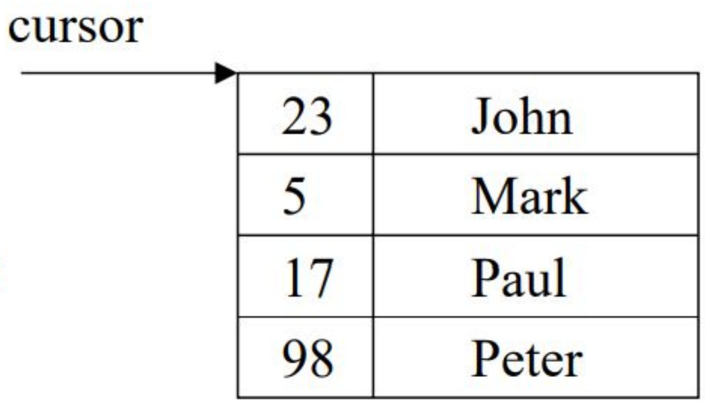
ResultSet Object

- Stores the results of a SQL query.
- A `ResultSet` object is similar to a ‘table’ of answers, which can be examined by moving a ‘pointer’ (cursor).

Accessing a ResultSet

- Cursor operations:
  - `first()`, `last()`, `next()`, `previous()`, etc.
- Typical code:

```
while( rs.next() ) {
  // process the row;
}
```



23	John
5	Mark
17	Paul
98	Peter

Accessing a ResultSet (Contd.)

- The `ResultSet` class contains many methods for accessing the value of a column of the current row
  - can use the column name or position
  - e.g. get the value in the lastName column:

```
rs.getString("lastName")
or rs.getString(2)
```

Accessing a ResultSet (Contd.)

- The ‘tricky’ aspect is that the values are SQL data, and so must be converted to Java types/objects.

- There are many methods for accessing/converting the data, e.g.
  - `getString()`, `getDate()`, `getInt()`, `getFloat()`, `getObject()`

ResultSet

- 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

Process the Results

- Step through the result set

```
while (rset.next()) { … }
```

- Use getXXX() to get each column value

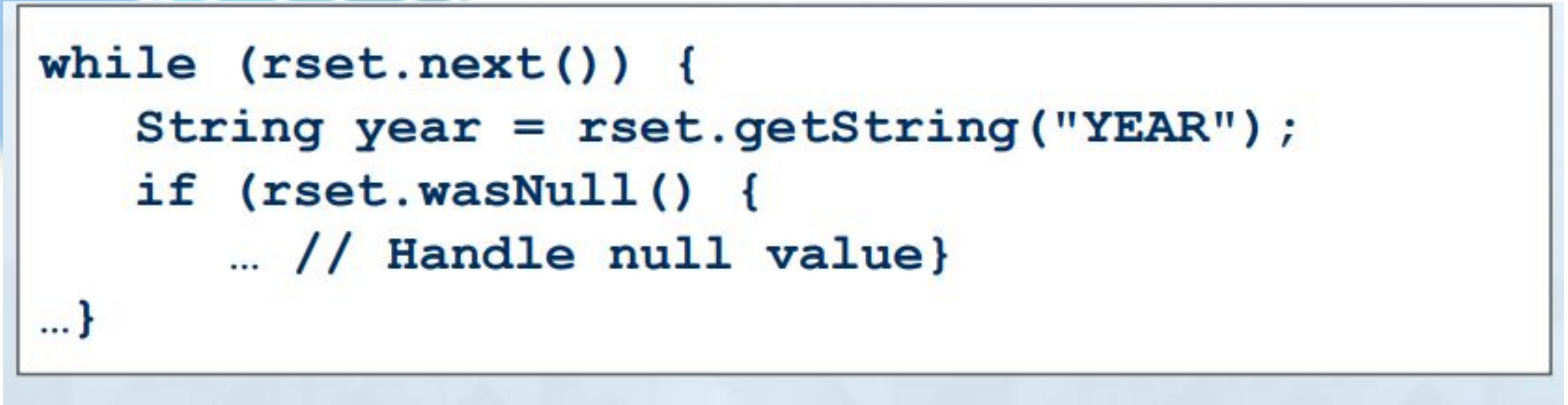
```
String val =
rset.getString(colname);
```

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

```
while (rset.next()) {
   String title = rset.getString("TITLE");
   String year = rset.getString("YEAR");
   … // Process or display the data
}
```

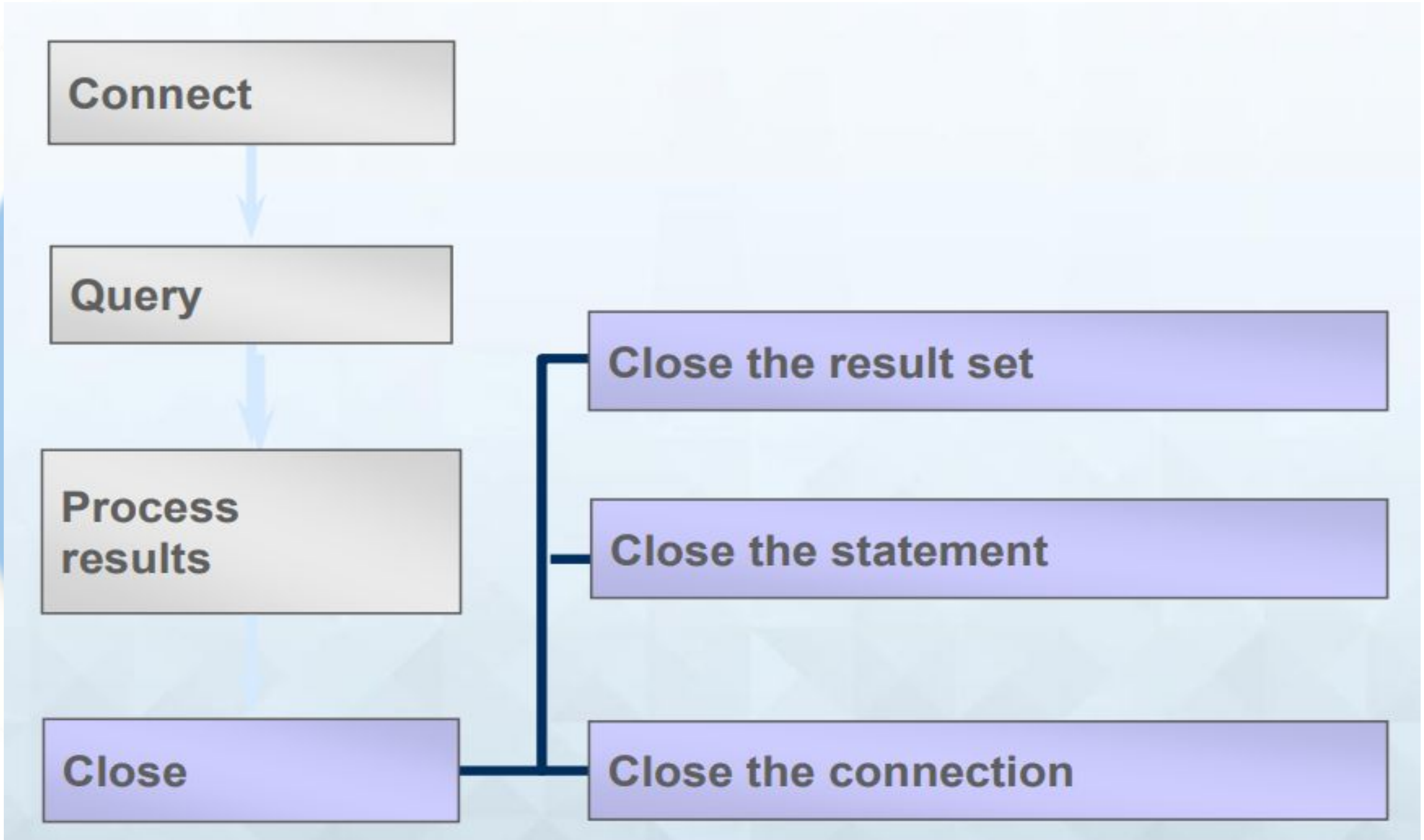
Handle SQL Null Values

- 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



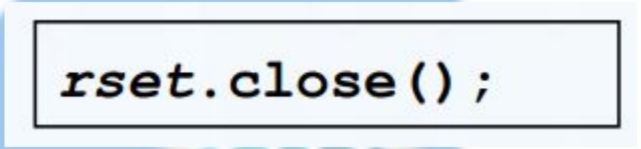
```
while (rset.next()) {
   String year = rset.getString("YEAR");
   if (rset.wasNull() {
      … // Handle null value}
…}
```

Stage 4: Close



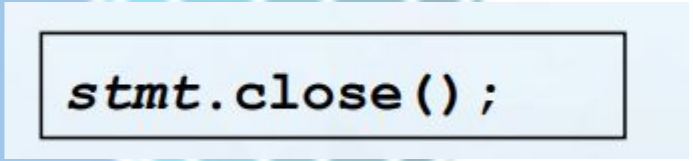
Close the Connection

- Close the ResultSet object



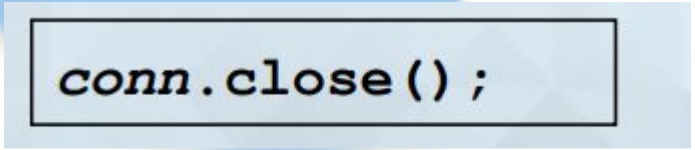
```
rset.close();
```

- Close the Statement object



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

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



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

PreparedStatement

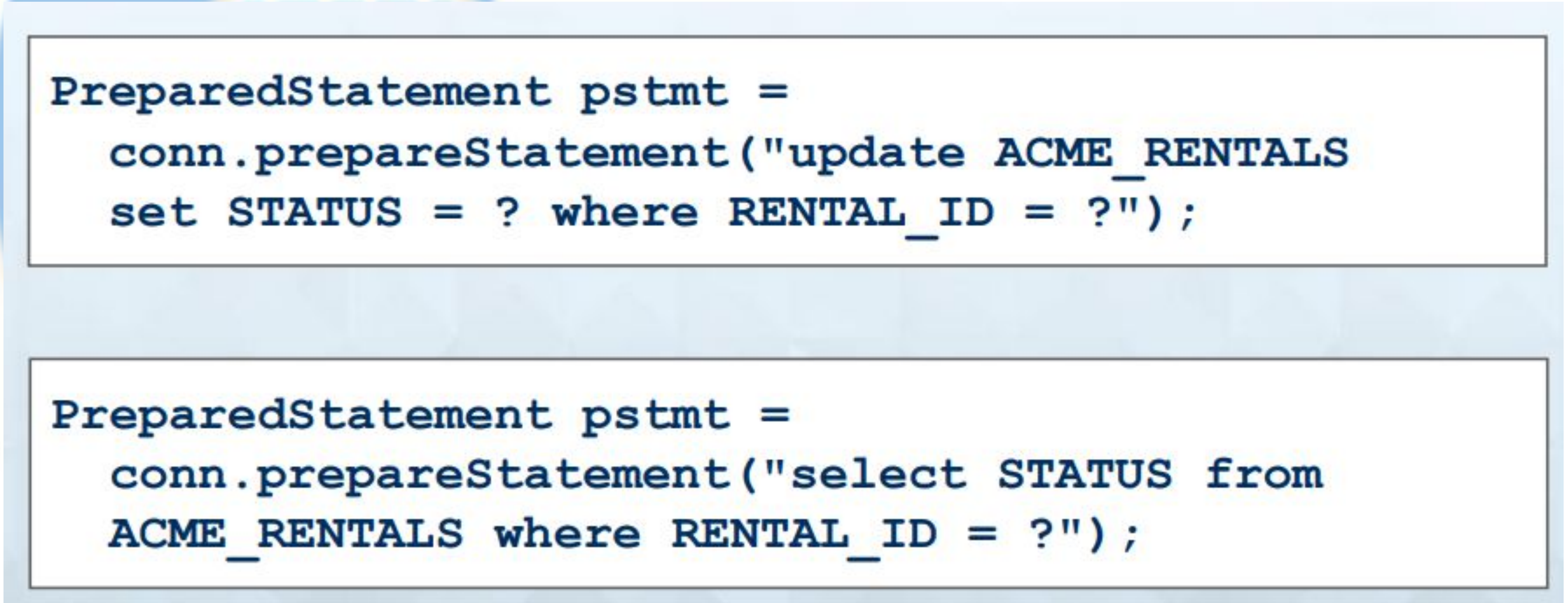
The PreparedStatement Object

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

Create a Prepared Statement

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



```
PreparedStatement pstmt =
   conn.prepareStatement("update ACME_RENTALS
   set STATUS = ? where RENTAL_ID = ?");
```

```
PreparedStatement pstmt =
   conn.prepareStatement("select STATUS from
   ACME_RENTALS where RENTAL_ID = ?");
```

Execute a Prepared Statement

- Supply values for the variables

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

- Execute the statement

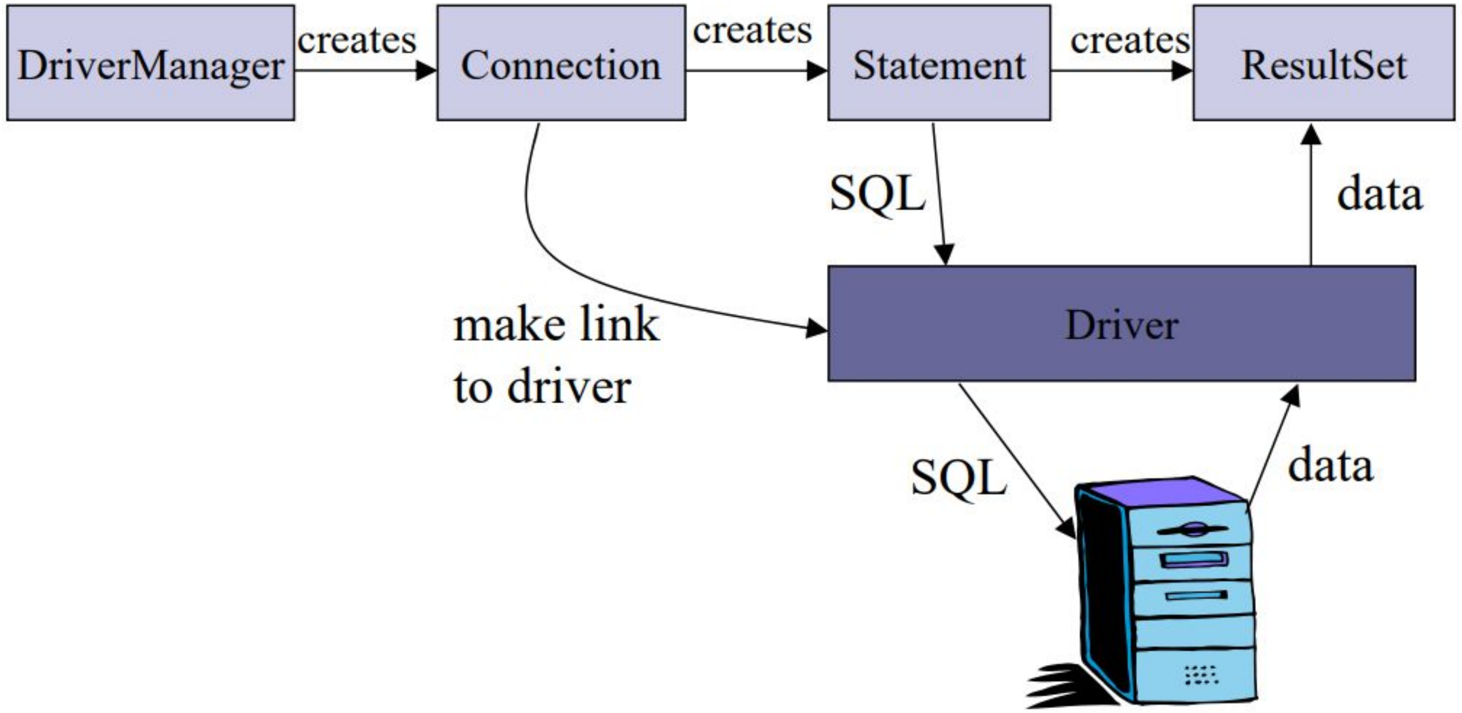
```
pstmt.executeQuery();
pstmt.executeUpdate();
```

```
PreparedStatement pstmt =
   conn.prepareStatement("update ACME_RENTALS
   set STATUS = ? where RENTAL_ID = ?");
pstmt.setString(1, "OUT");
pstmt.setInt(2, rentalid);
pstmt.executeUpdate();
```

JDBC Pseudo Code

- All JDBC programs do the following:
- Step 1) load the JDBC driver
- Step 2) Specify the name and location of the database being used
- Step 3) Connect to the database with a `Connection` object
- Step 4) Execute a SQL query using a `Statement` object
- Step 5) Get the results in a `ResultSet` object
- Step 6) Finish by closing the `ResultSet`, `Statement` and `Connection` objects

JDBC Diagram



Any questions?

Meta Data

- Meta data is the information *about* the database:
  - e.g. the number of columns, the types of the columns
  - meta data is the *schema* information

ID	Name	Course	Mark
007	James Bond	Shooting	99
008	Aj. Andrew	Kung Fu	1

← meta data

Code to add

```
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>8.0.25</version>
</dependency>
```

```
public static Persons[] fetchDb(){
                Persons m[]=null;
   try{
     Class.forName("com.mysql.cj.jdbc.Driver");
    Connection con=DriverManager.getConnection(
     "jdbc:mysql://localhost:3306/tyconsultDB","root","8438");
     Statement stmt=con.createStatement();

     ResultSet rs=stmt.executeQuery("select * from persons");
     ArrayList<Persons> list=new ArrayList<Persons>();
     while(rs.next()) {
     list.add(new Persons(rs.getInt(1),rs.getString(2),rs.getString(3),rs.getString(4),
                          rs.getString(5)));
       m=new Persons[list.size()];
       m= list.toArray(m);
        }

   }catch(Exception e){ System.out.println(e);}
   System.out.println("length is:" +m.length);
   for (int i = 0; i < m.length; i++) {
                System.out.println(m[i]);
               }
   return m;
   }
```