JDBC

Java DataBase Connectivity

TOPICS-Outline…..

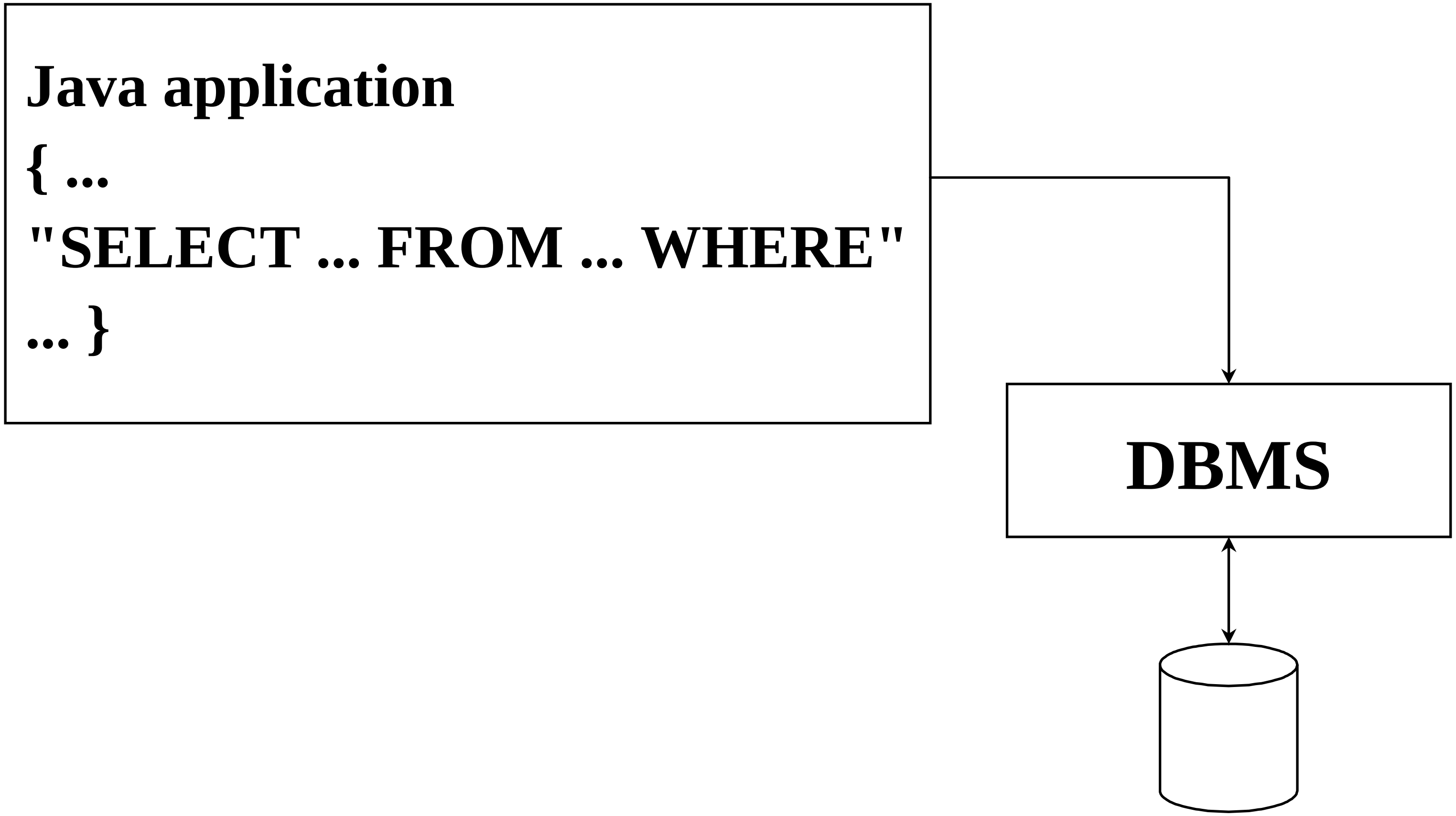
-What is JDBC?

-JDBC Driver and it's types…

-Different JDBC classes….

-How to connect a DBMS?

JDBC (Java DB Connectivity)



Java application

{ ...

"SELECT ... FROM ... WHERE"

... }

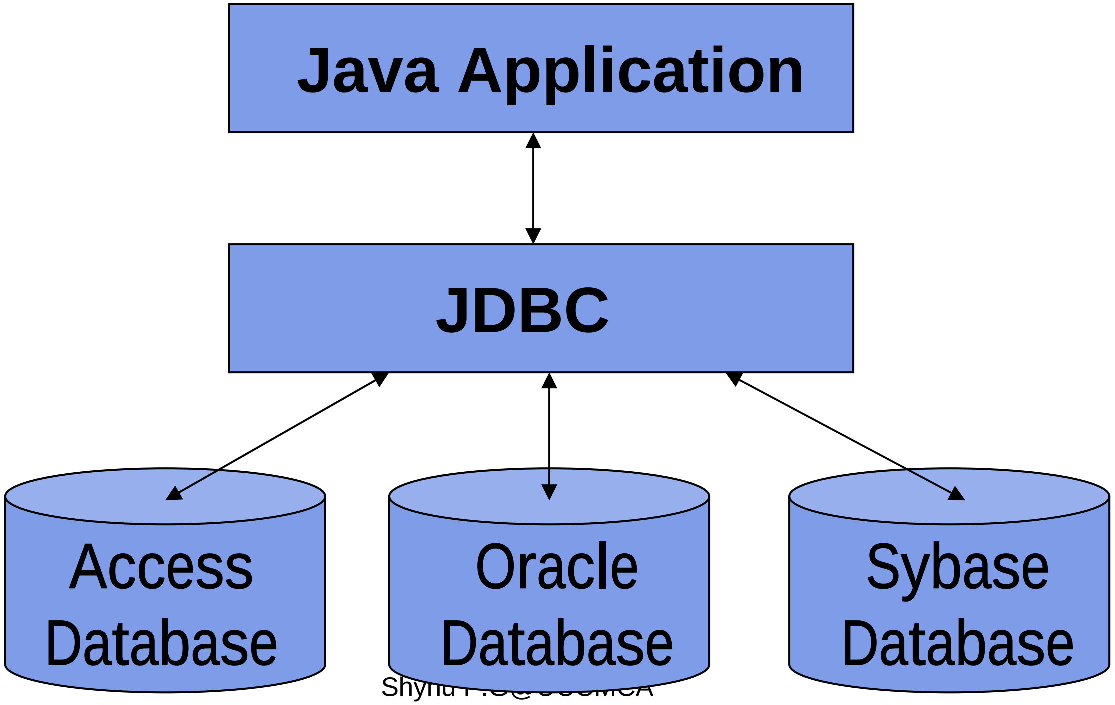
DBMS

What is JDBC?

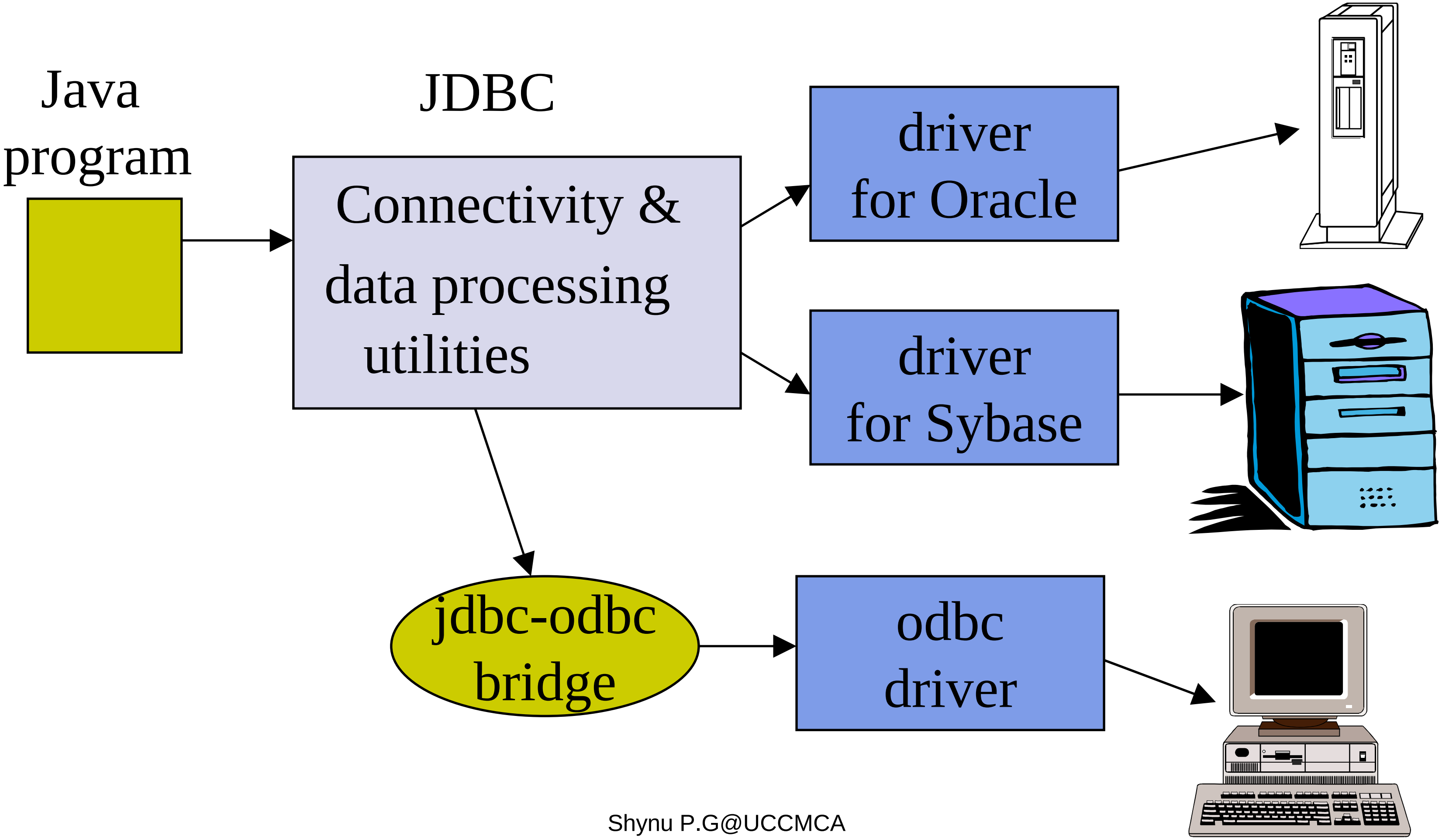
- “An API that lets you access virtually any tabular data source (relational/spreadsheet/flat files) from the Java programming language”
- JDBC library provides the means for executing SQL statements to access and operate on a relational database
- JDBC library is implemented in the java.sql package
  - Set of classes and interfaces that provide a uniform API for access to broad range of databases

Talking to Databases

- A JDBC based application is insulated from the characteristics of specific database engines



JDBC in Use



Java program

JDBC

Connectivity & data processing utilities

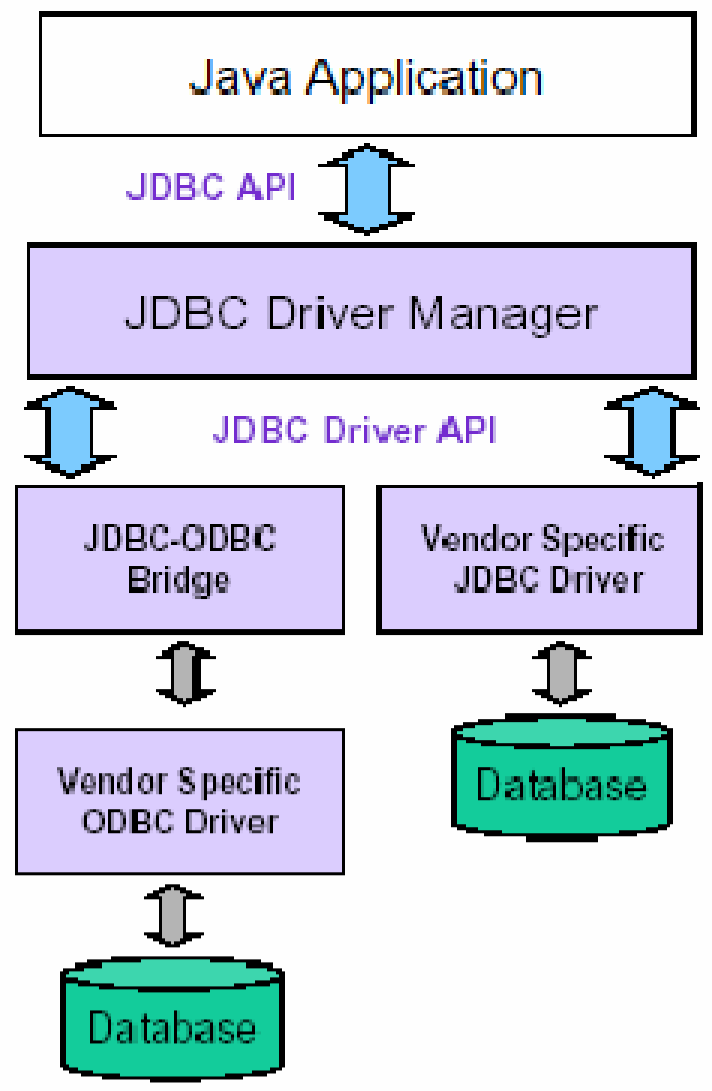
driver for Oracle

driver for Sybase

jdbc-odbc bridge

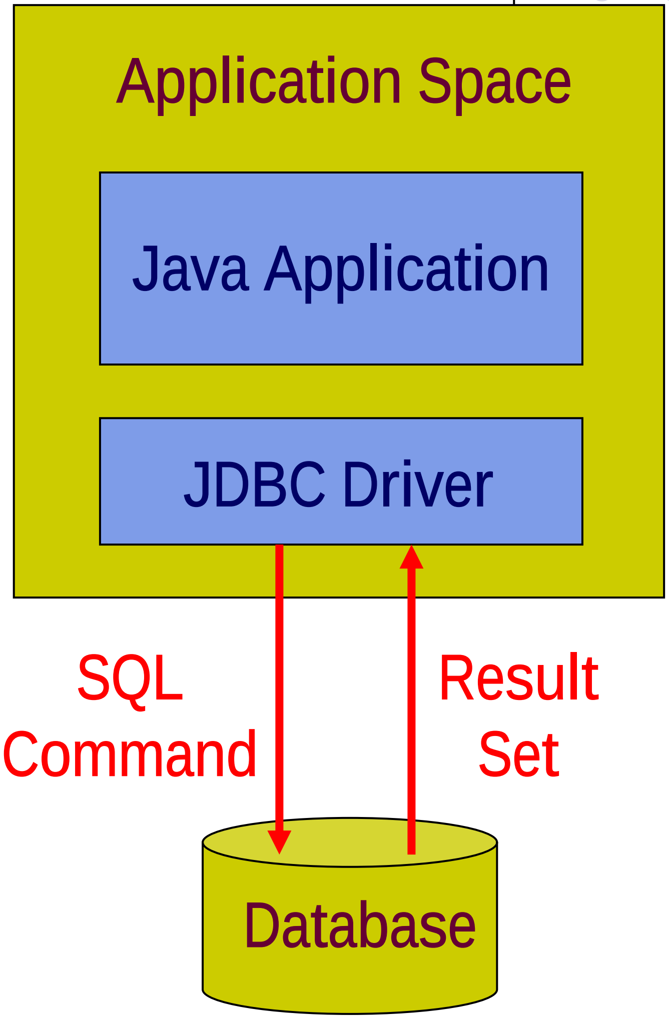
odbc driver

General Architecture



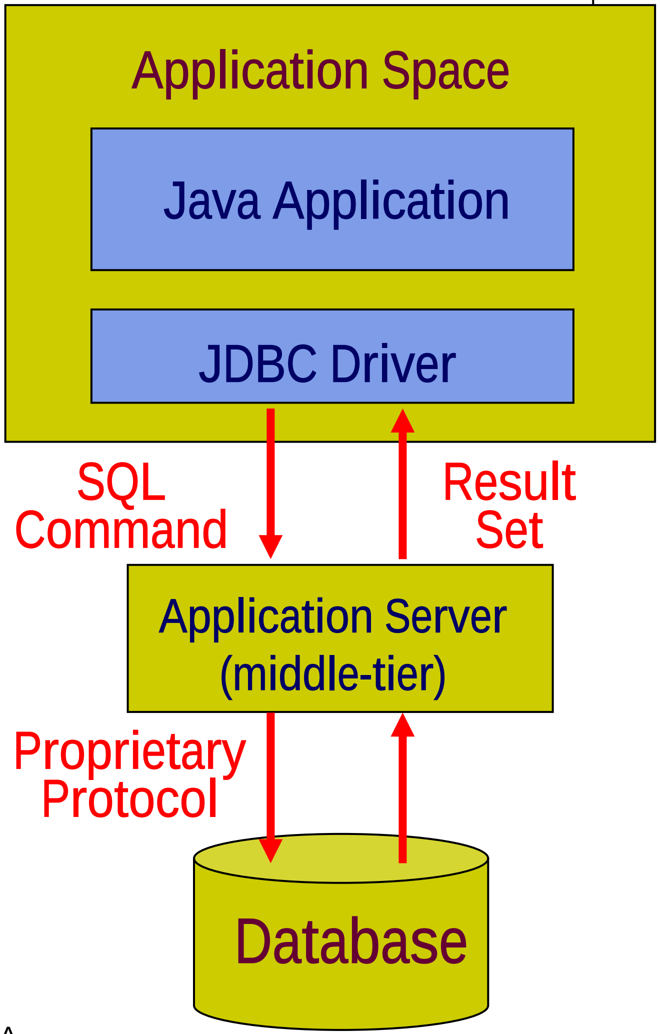
Two-Tier Database Access Model

- Java Application talks directly to the database
- Accomplished through the JDBC driver which sends commands directly to the database
- Results sent back directly to the application



Three-Tier Database Access Model

- **JDBC driver sends commands to a middle tier, which in turn sends commands to database.**
- **Results are sent back to the middle tier, which communicates them back to the application**

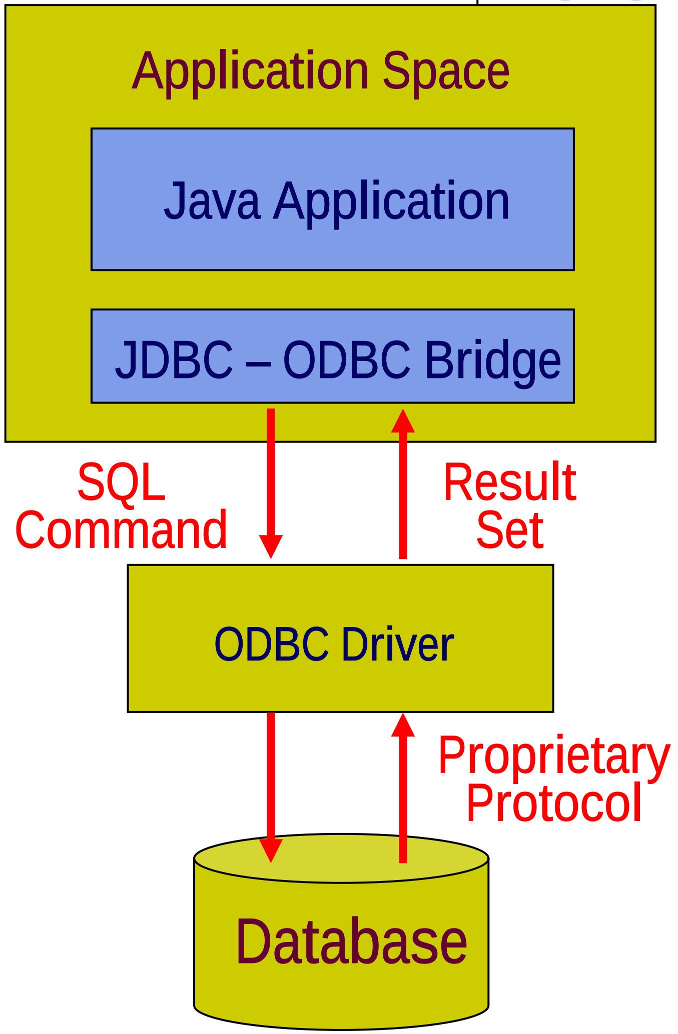


JDBC Driver Types

- JDBC-ODBC Bridge, plus ODBC driver (Type 1)
- Native-API, partly Java driver (Type 2)
- JDBC-net, pure Java driver (Type 3)
- Native-protocol, pure Java driver (Type 4)

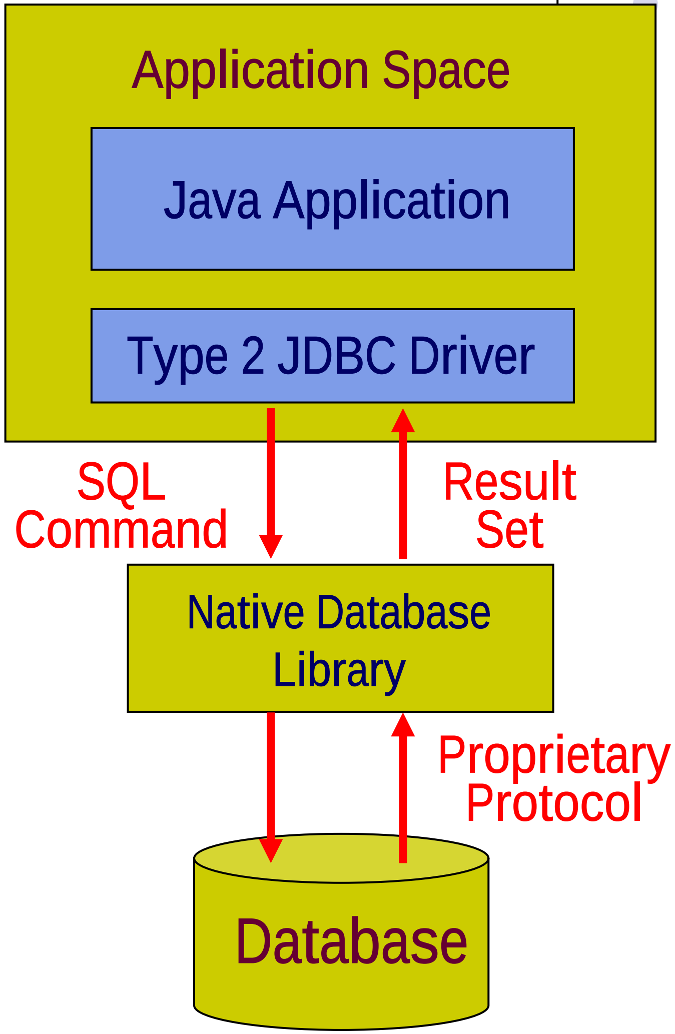
Type 1: JDBC-ODBC Bridge, Plus ODBC Driver

- This driver type is provided by Sun with JDK
- Provides JDBC access to databases through ODBC drivers
- ODBC driver must be configured for the bridge to work
- Only solution if no JDBC driver available for the DBMS



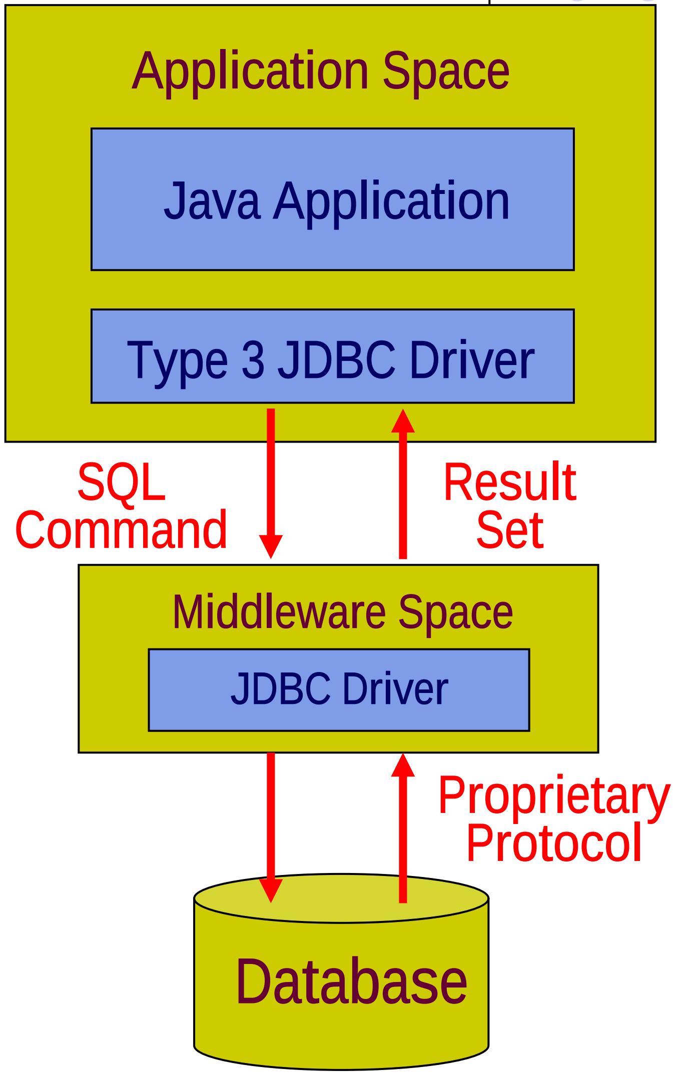
Type 2: Native-API, Partly Java Driver

- Native-API driver converts JDBC commands into DBMS-specific native calls
- Same restrictions as Type1 – must have some binary code loaded on its machine
- Directly interfaces with the database



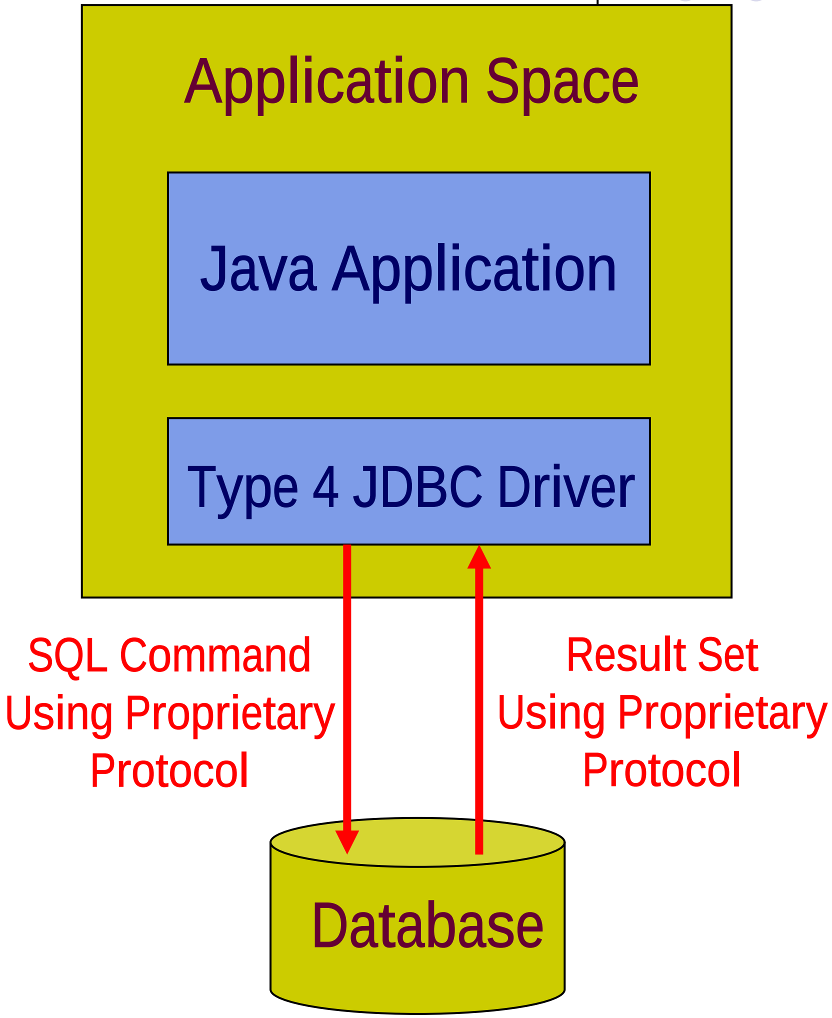
Type 3: JDBC-Net, Pure Java Driver

- **Translates JDBC calls into a database-independent network protocol and sent to a middleware server.**
- **This server translates this DBMS-independent protocol into a DBMS-specific protocol and sent to the database**
- **Results sent back to the middleware and routed to the client**



Type 4: Native-Protocol, Pure Java Driver

- **Pure Java drivers that communicate directly with the vendor's database**
- **JDBC commands converted to database engine's native protocol directly**
- **Advantage: no additional translation or middleware layer**
- **Improves performance**


Application Space
Java Application
Type 4 JDBC Driver
SQL Command Using Proprietary Protocol
Result Set Using Proprietary Protocol
Database

JDBC Concepts

- JDBC’s design is very similar to the design of ODBC
- Driver Manager
  - Loads database drivers, and manages the connection between the application and the driver
- Driver
  - Translates API calls into operations for a specific data source
- Connection
  - A session between an application and a database

JDBC Concepts (contd.)

- Statement
  - An SQL Statement to perform a query or update operation
- ResultSet
  - Logical set of columns and rows returned by executing an SQL statement (resulting tuples)
- Metadata
  - Information about returned data, the database and the driver

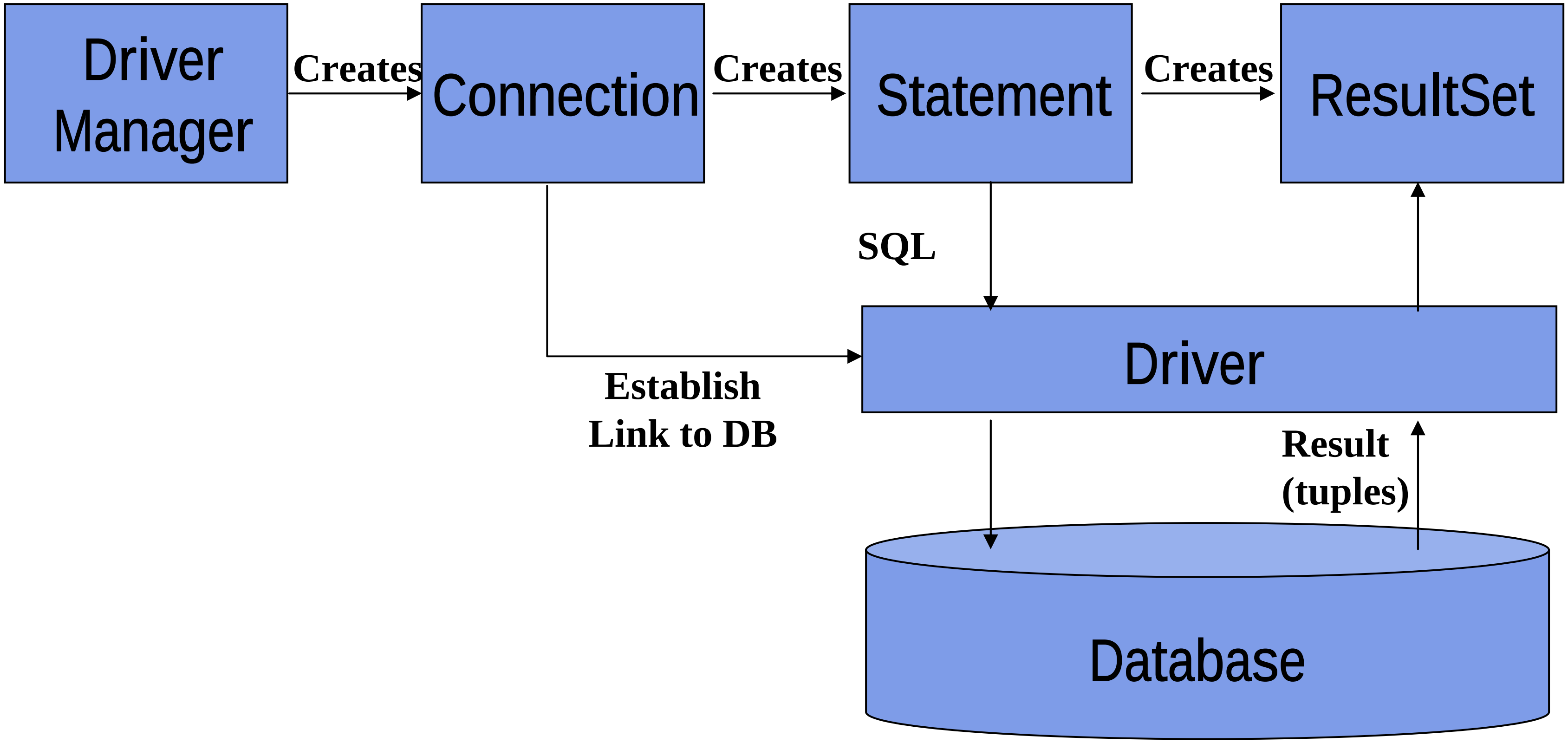
Basic steps to use a database in Java

1. Load DB-specific JDBC driver
2. Get a Connection object
3. Get a Statement object
4. Execute queries and/or updates
5. Read results
6. Read Meta-data (optional step)
7. Close Statement and Connection objects

More specifically….

- The following steps are executed for running a JDBC application
  - Import the necessary classes
  - Load the JDBC driver
  - Identify the database source
  - Allocate a “connection” object (create)
  - Allocate a “Statement” object (create)
  - Execute a query using the “Statement” object
  - Retrieve data from the returned “ResultSet” object
  - Close the “ResultSet” object
  - Close the “Statement” object
  - Close the “Connection” object

JDBC Component Interaction



1. Load DB-Specific Database Driver

- To manually load the database driver and register it with the DriverManager, load its class file
  - Class.forName(<database-driver>) (Dynamically loads a driver class)

```
try {
    // The driver has to be in the classpath.
    Class.forName("com.mysql.jdbc.Driver ");
   // Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");



}
catch (ClassNotFoundException cnfe){
     System.out.println("" + cnfe);
}
```

2. Get a Connection Object

- DriverManager class is responsible for selecting the database and creating the database connection
  - Using DataSource is a preferred means of getting a connection object

- **Eg:** Create the database connection as follows:

```
    try {
         Connection con =
    DriverManager.getConnection("jdbc:odbc:mydsn",“user", “pwd");
  //Establishes connection to database by obtaining
    Connection object with <protocol:jdbc driver:data source>
 }
     catch(SQLException sqle) {
         System.out.println("" + sqle);
    }
```

DriverManager & Connection

- java.sql.DriverManager
  - getConnection(String url, String user, String password) throws SQLException

- java.sql.Connection
  - Statement createStatement() throws SQLException
  - void close() throws SQLException
  - void setAutoCommit(boolean b) throws SQLException
  - void commit() throws SQLException
  - void rollback() throws SQLException

3. Get a Statement Object

- Create a Statement Object from Connection object
  - java.sql.Statement
    - ResultSet executeQuery(string sql)
    - int executeUpdate(String sql)
  - Example:
    - Statement statement = connection.createStatement();
- The same Statement object can be used for many, unrelated queries

4. Executing Query or Update

- From the Statement object, the 2 most used commands are
  - (a) QUERY (SELECT)
    - ResultSet rs = statement.executeQuery(“ SELECT * FROM customer_tbl”);
  - (b) ACTION COMMAND (UPDATE/DELETE)
    - int iReturnValue = statement.executeUpdate(“UPDATE manufacture_tbl SET name = ‘IBM' WHERE mfr_num = 19985678");

5. Reading Results

- Loop through ResultSet retrieving information
  - java.sql.ResultSet
    - boolean next()
    - xxx getXxx(int columnNumber)
    - xxx getXxx(String columnName)
    - void close()
- The iterator is initialized to a position before the first row
  - You must call next() once to move it to the first row

5. Reading Results (Continued)

- Once you have the ResultSet, you can easily retrieve the data by looping through it

    - while (rs.next()){
    - // Wrong this will generate an error
    - String value0 = rs.getString(0);
    - // Correct!
    - String value1 = rs.getString(1);
    - int value2 = rs.getInt(2);
    - int value3 = rs.getInt(“ADDR_PIN");
    - }

5. Reading Results (Continued)

- When retrieving data from the ResultSet, use the appropriate getXXX() method
  - getString()
  - getInt()
  - getDouble()
  - getObject()
- There is an appropriate getXXX method of each java.sql.Types datatype

6. Read ResultSet MetaData and DatabaseMetaData (Optional)

- Once you have the ResultSet or Connection objects, you can obtain the Meta Data about the database or the query
- This gives valuable information about the data that you are retrieving or the database that you are using
  - ResultSetMetaData rsMeta = rs.getMetaData();
  - DatabaseMetaData dbmetadata = connection.getMetaData();
    - There are approximately 150 methods in the DatabaseMetaData class.

ResultSetMetaData Example

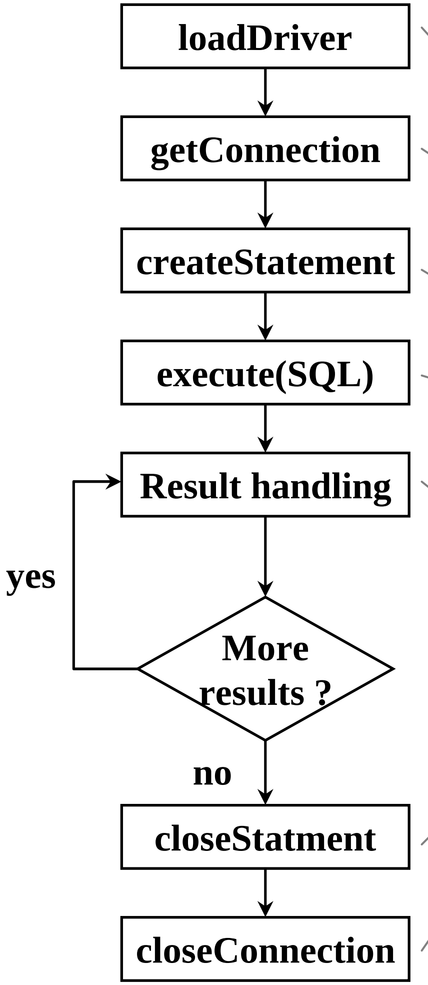
```
ResultSetMetaData meta = rs.getMetaData();
//Return the column count
int iColumnCount = meta.getColumnCount();

for (int i =1 ; i <= iColumnCount ; i++){
  System.out.println(“Column Name: " + meta.getColumnName(i));
  System.out.println(“Column Type" + meta.getColumnType(i));
  System.out.println("Display Size: " + meta.getColumnDisplaySize(i));
   …………………………
   ……………………………
}
```

7.Close connection

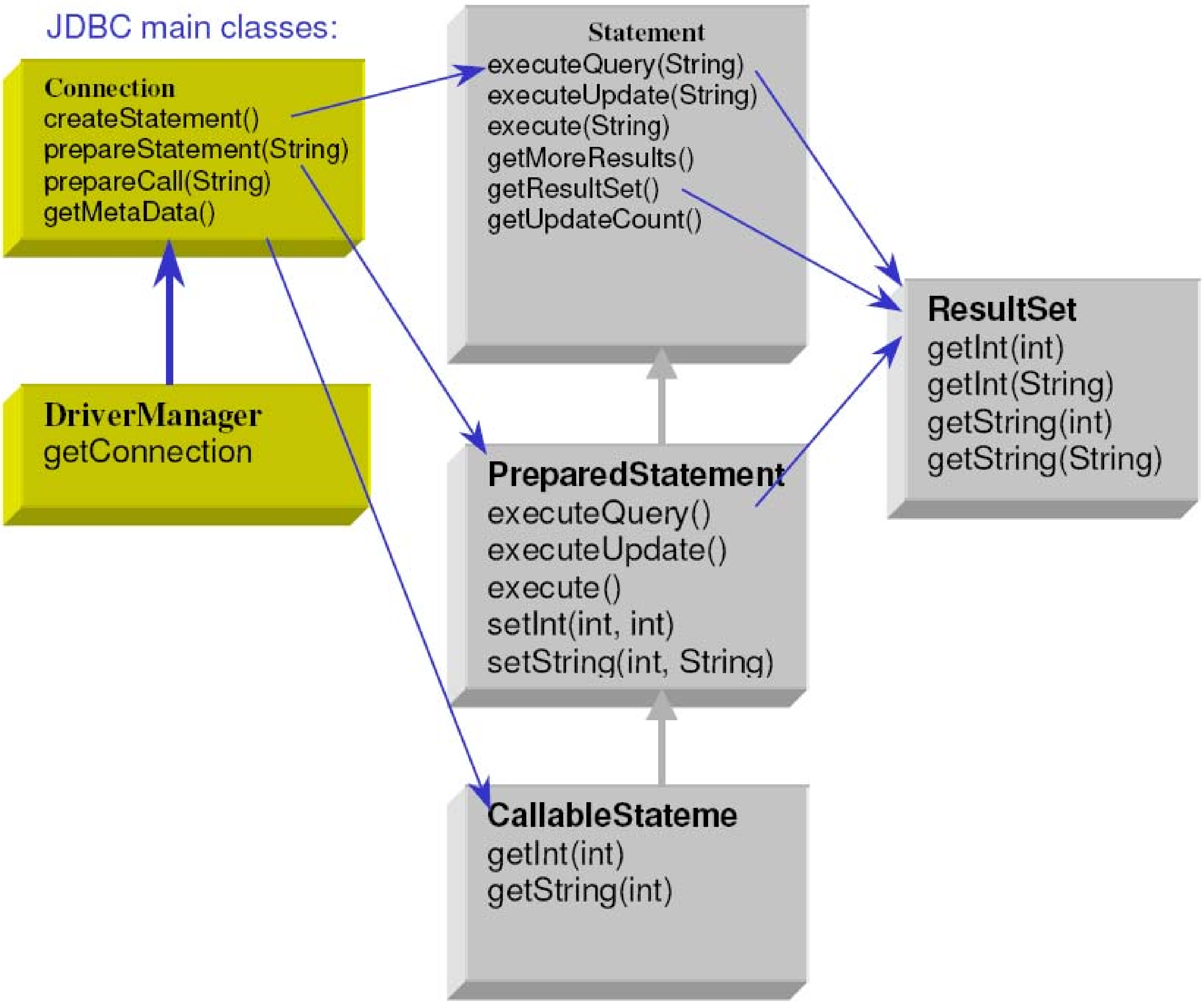
- rs.close(); //close resultset object
- statement.close();
- con.close();

A Simple JDBC application



```
import java.sql.*;
public class jdbctest {
  public static void main(String args[]){
     try{
       Class.forName(“sun.jdbc.odbc.JdbcOdbcDriver");
       Connection con = DriverManager.getConnection
          ("jdbc:odbc:DSN", "user", "passwd");
       Statement stmt = con.createStatement();
       ResultSet rs = stmt.executeQuery
          ("select name, number from pcmtable where number < 2");
       while(rs.next())
              System.out.println(rs.getString(1) + " (" + rs.getInt(2) + ")");
       stmt.close()
       con.close();
     } catch(Exception e){
       System.err.println(e);
}}}
```

JDBC main classes:



Connection
createStatement()
prepareStatement(String)
prepareCall(String)
getMetaData()

DriverManager
getConnection

Statement
executeQuery(String)
executeUpdate(String)
execute(String)
getMoreResults()
getResultSet()
getUpdateCount()

PreparedStatement
executeQuery()
executeUpdate()
execute()
setInt(int, int)
setString(int, String)

CallableStateme
getInt(int)
getString(int)

ResultSet
getInt(int)
getInt(String)
getString(int)
getString(String)

Driver Manager

- The DriverManager class is responsible for establishing connections to the data sources, accessed through the JDBC drivers
- JDBC database drivers are defined by classes that implement the “Driver” interface

Connection Object

- Creating a connection to a data source
- Connection object represents an established connection to a particular data source
- A connection object can also be used to query the data source (data and meta data)
- Different versions of getConnection() method contained in the DriverManager class that returns a connection object:
  - **Connection myconn = DriverManager.getConnection(source);**
  - **Connection myconn = DriverManager.getConnection(source, username, password);**
  - **Example**

System DSN name – ODBC data source

```
String mysource = “jdbc:odbc:technical_library”;
Connection myconn = DriverManager.getConnection(mysource);
```

Statement Object

- **Provides workspace for creating an SQL query, execute it, and retrieve the results that are returned**
- **Statement objects are created by calling the createStatement() method of a valid connection object**
- **Used to execute an SQL query by calling the executeQuery() method of Statement object**
- **The SQL query string is passed as argument to the executeQuery() method**
- **The result of executing the query is returned as on object of type "ResultSet"**

```
Statement  mystatement = myconn.createStatement();
ResultSet  myresults = mystatement.executeQuery("select * from authors");
```

- **JDBC Provides two other kinds of objects to execute SQL statement:**
  - **PreparedStatement -> extends Statement class**
  - **CallableStatement -> extends PreparedStatement class**

ResultSet Object

- The results of executing an SQL query are returned in the form of an object that implements the ResultSet interface
- ResultSet object contains a “cursor” that points to a particular record (called the current record)
- When the ResultSet object is created, the cursor points to the position immediately preceeding the first record
- Several methods available to navigate the ResultSet by moving the cursor
  - first(), last(), beforeFirst(), afterLast(), next(), previous(), etc. //returns true if the move is successful
  - isFirst() //whether you reached the beginning of the ResultSet
  - isLast() // whether you reached the end of the ResultSet

Accessing Data in a ResultSet

- We can retrieve the value of any column for the current row (specified by the cursor) by name or position
  - Using Name: authorNames.getString(“lastname”);

Name of the ResultSet

Method that returns the value of String

Name of the column or attribute

  - Using Position: authorNames.getString(2);

Second column in the row or tuple

  - Using the column position is a little bit faster
- Methods for Retrieving Column Data
  - getString(), getInt(), getShort(), getFloat(), getDouble(), getTime() etc.
- We can always use getString() method for numerical values if we are not going to do some computations
- Column names are NOT case sensitive

Scrollable Result Sets

- In JDBC1.0, result sets could be navigated in only one direction (forward) and starting at only one point (first row)
- Since JDBC 2.0, the cursor can be manipulated as if it were a array index
- Methods exist for reading both forward and backward, for starting from any row, and for testing the current cursor location.

JDBC 2.0 Navigation Methods for Scrollable Result Sets

boolean next ( )	Advances the cursor to the next row.
boolean previous ( )	Moves the cursor back one row.
boolean first ( )	Moves the cursor to the first row.
boolean last ( )	Moves the cursor to the last row.
void beforeFirst ( )	Moves the cursor before the first row, usually in anticipation of calling next ( )
void afterLast ( )	Moves the cursor after the last row, usually in anticipation of calling previous ( )
boolean absolute (int row)	Moves the cursor to the specified row. Specifying a negative number moves the cursor relative to the end of the result set;

JDBC 2.0 Navigation Methods for Scrollable Result Sets (contd.)

boolean isBeforeFirst ( )	True if the cursor is before the first row.
boolean isAfterLast ( )	True if the cursor is after the last row.
boolean isFirst ( )	True if the cursor is positioned on the first row.
boolean isLast ( )	True if the cursor is positioned on the last row.