# JDBC: Priming

- To compile the APP, `javac` needs to know where the JDBC library is.
- To run the APP, `java` needs to know how to locate the database system in question.
  Mainly, this is setting up the CLASSPATH and LD_LIBRARY_PATH correctly for the local system.
- On PRISM,

  ```
  % source ~db2leduc/.cshrc
  ```

  will do it.



# JDBC: Establishing the Driver

The driver manages the type of data source (database system) with which the APP will be communicating via JDBC.

```
import java.net.*;
import java.sql.*;
...
// Register the driver with DriverManager.
Class.
  forName("COM.ibm.db2.jdbc.app.DB2Driver").
  newInstance();
```

# JDBC: The Connection

Which database is it?

```
// Conn. to the DBMS.
private Connection conDB;
// URL: Which database?
private String url;
...
// URL: This database.
url = "jdbc:db2:c3421m";
conDB = DriverManager.getConnection(url);
...
conDB.close();
```

Can throw a `COM.ibm.db2.jdbc.DB2Exception`.
Typically *one* connection per APP, *not* one per object!



# JDBC: "Talking" to the DB

1. Compose SQL in a string.
2. Prepare the SQL statement.
3. Execute the statement.
4. Walk through the resulting *cursor*.

# Building the SQL Query

A query is pure SQL in a Java string.

```
// The SQL text.
String            queryText = "";
// The query handle.
PreparedStatement querySt   = null;
// A cursor.
ResultSet         answers   = null;
...
queryText =
    "SELECT COUNT(*) as #custs"
  + "    FROM yrb_customer";
```



# Preparing & Executing

- Prepare the statement:

```
querySt =
    conDB.prepareStatement(queryText);
```

- Execute the statement:

```
answers = querySt.executeQuery();
```

Why two steps?

# Walk the Cursor

```
if (answers.next()) {
    int num_of_customers =
        answers.getInt("#custs");
    System.out.print("There are ");
    System.out.print(num_of_customers);
    System.out.println(
        " number of customers.");
} else {
    System.out.println(
        "There are no customers.");
}
```

Can we ask `answers` how many rows there are? *No.*



# Clean Up!

We're used to Java garbage collecting for us. However, this does not guarantee that these "objects" are deallocated when we are done with them on the DBMS side.

```
// Close the cursor.
answers.close();
...
// We're done with the handle.
querySt.close();
...
// Close the connection.
conDB.close();
```

# Cursors: Properties

- **scrollability**: Whether the cursor can move forward, backward, or to a specific row.
- **updatability**: Whether the cursor can be used to update or delete rows.
- **holdability**: Whether the cursor stays open after a `commit`.

Typically, a cursor is *not* scrollable unless declared so *and* provisions have been made.



# Cursors: Bad Habits

```
while (custCR.next()) {
    cid = custCR.getInt("cid");
    salesST.setInt(1, cid);
    salesCR = salesST.executeQuery();
    salesCR.next();
    sales = salesCR.getFloat("sales");
    System.out.print(cid);
    System.out.print(sales);
}
```

- Never use a cursor to do what could have been done instead in a query.
- In *procedural* versus *declarative*, go declarative!