16

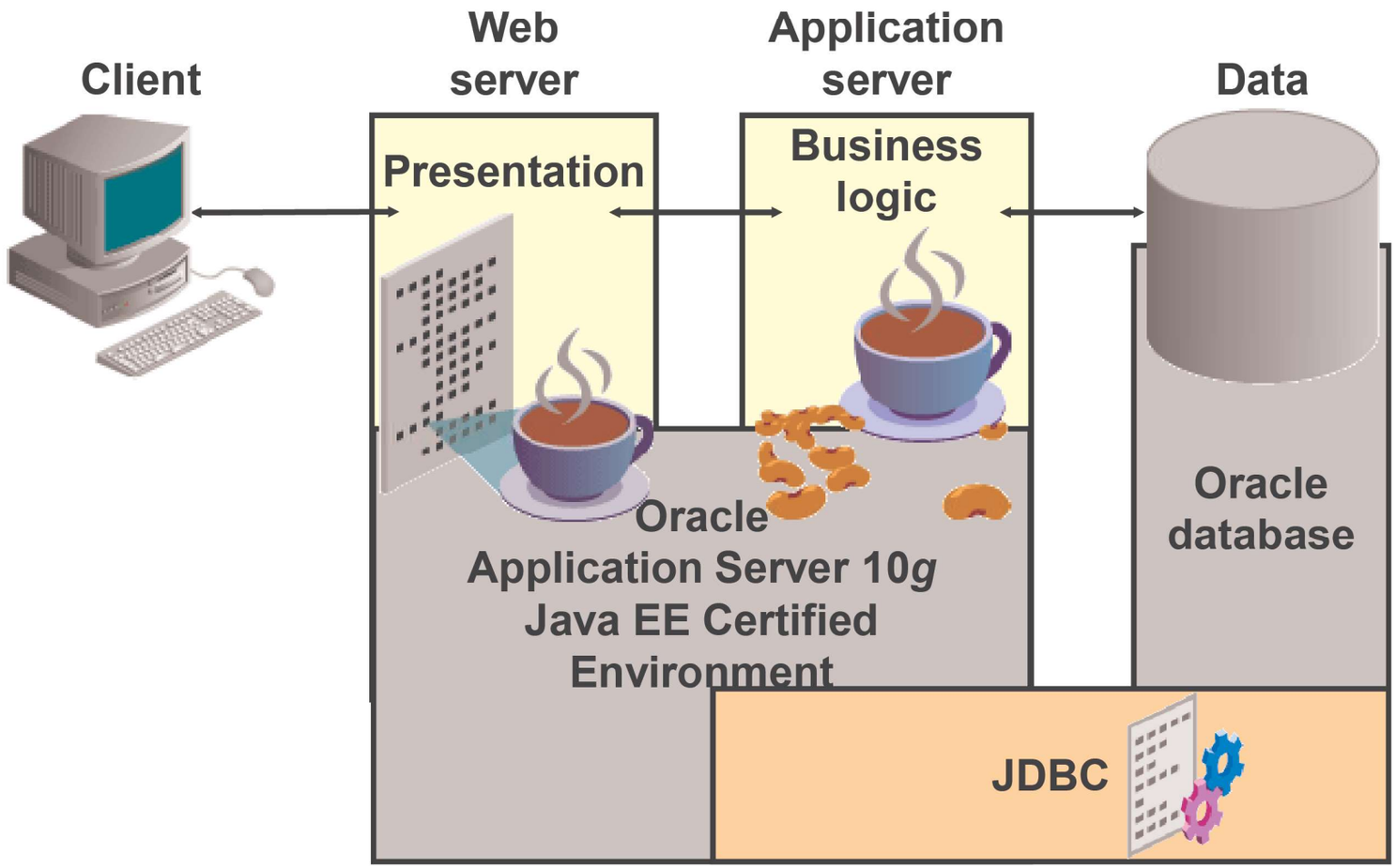
Using JDBC to access Database

Objectives

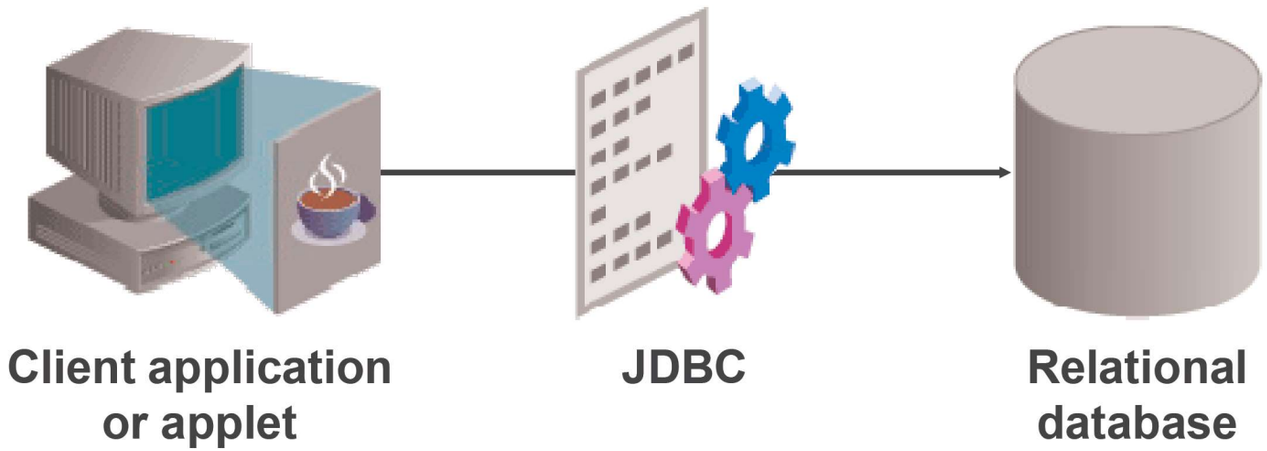
After completing this lesson, you should be able to do the following:

- Describe how Java code connects to the database
- Describe how Java database functionality is supported by the Oracle database
- Load and register a JDBC driver and Connect to an Oracle database
- Perform a simple `SELECT` statement
- Map simple Oracle database types to Java types

Java, Java EE, and Oracle 10*g*

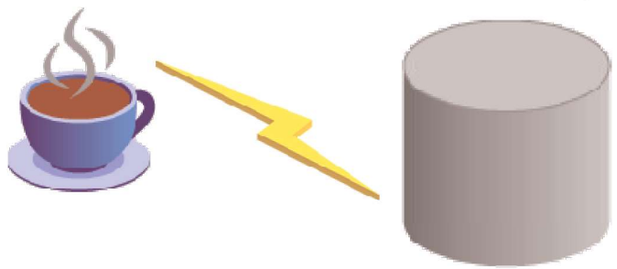


Client applications, JSPs, and servlets use JDBC.



Java Database Connectivity (JDBC)

- JDBC is a standard API for connecting to relational databases from Java.
  - The JDBC API includes the Core API Package in `java.sql`.
  - JDBC 2.0 API includes the Optional Package API in `javax.sql`.
  - JDBC 3.0 API includes the Core API and Optional Package API.



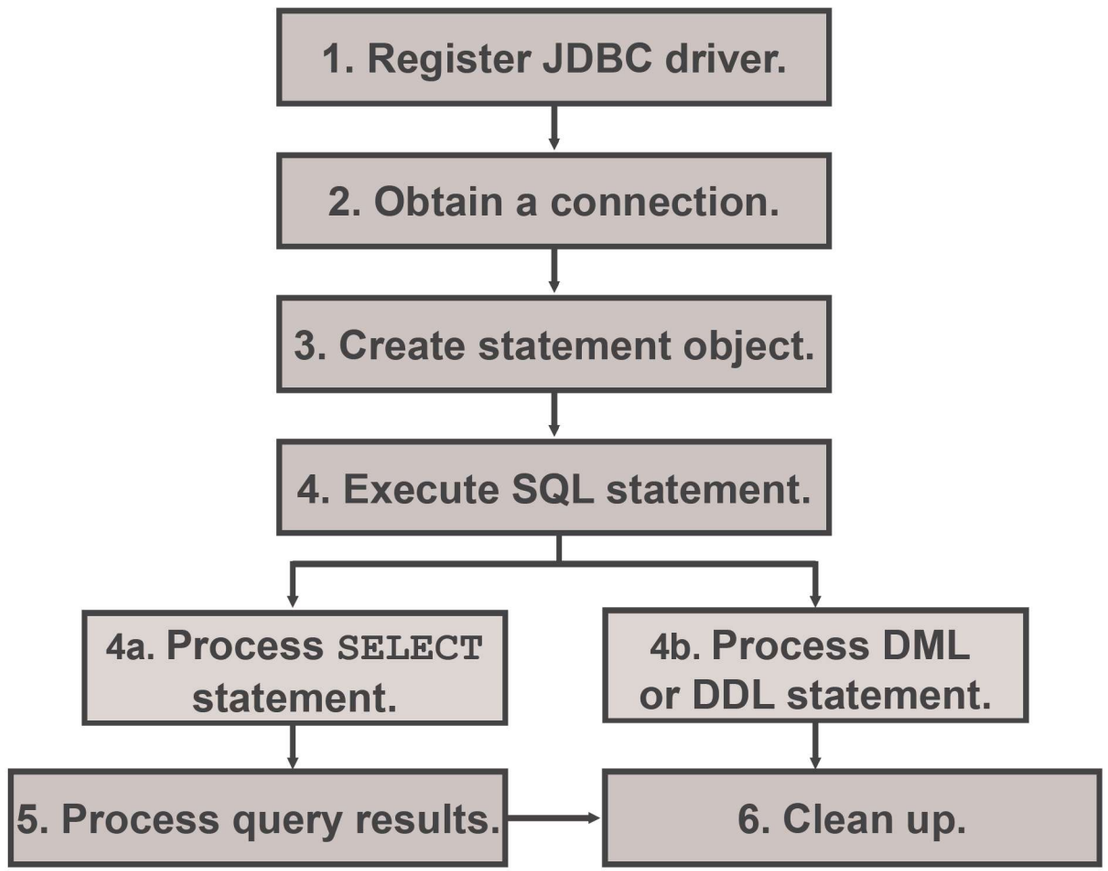
- Include the Oracle JDBC driver archive file in the `CLASSPATH`.
- The JDBC class library is part of Java 2, Standard Edition (J2SE).

Preparing the Environment

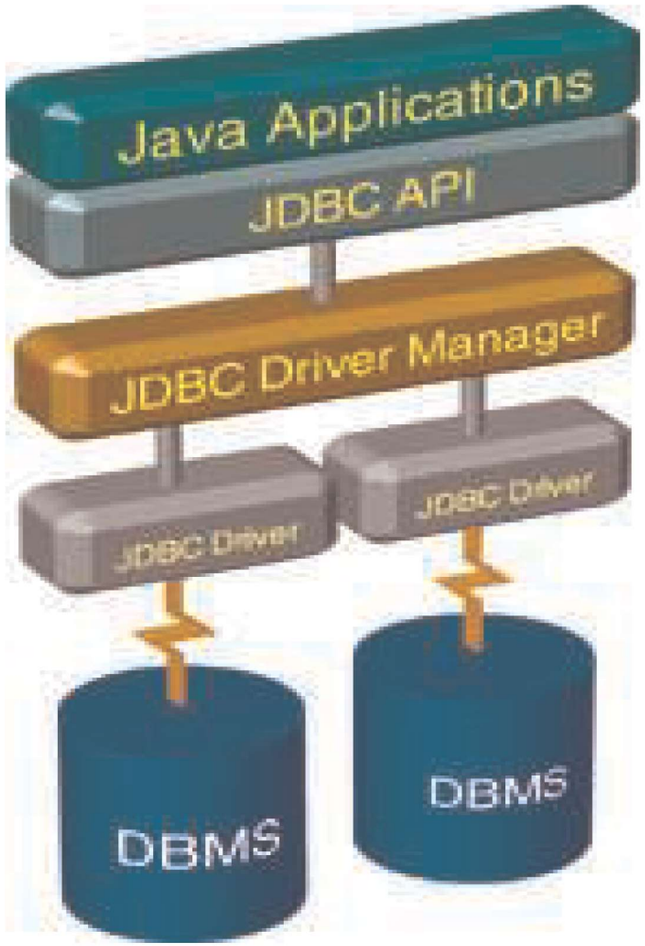
- Import the JDBC packages:
- Include the JDBC driver classes in the classpath settings.

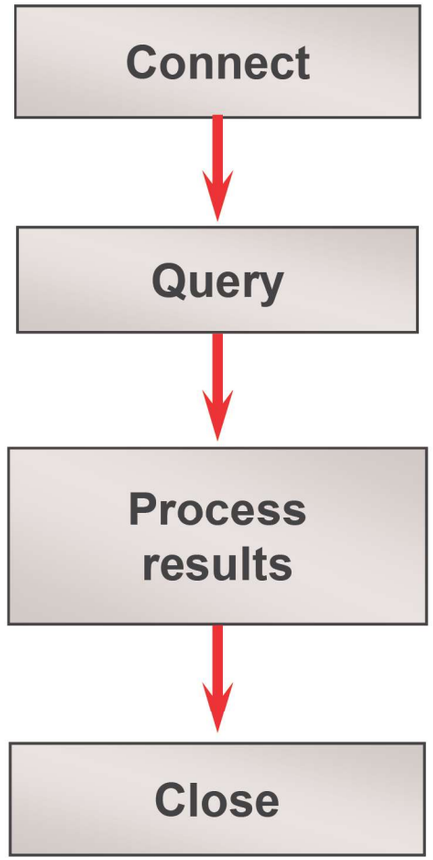
```
// Standard packages
import java.sql.*;
import java.math.*; // optional
// Oracle extension to JDBC packages
import oracle.jdbc.*;
import oracle.sql.*;
```

Steps for Using JDBC to Execute SQL Statements

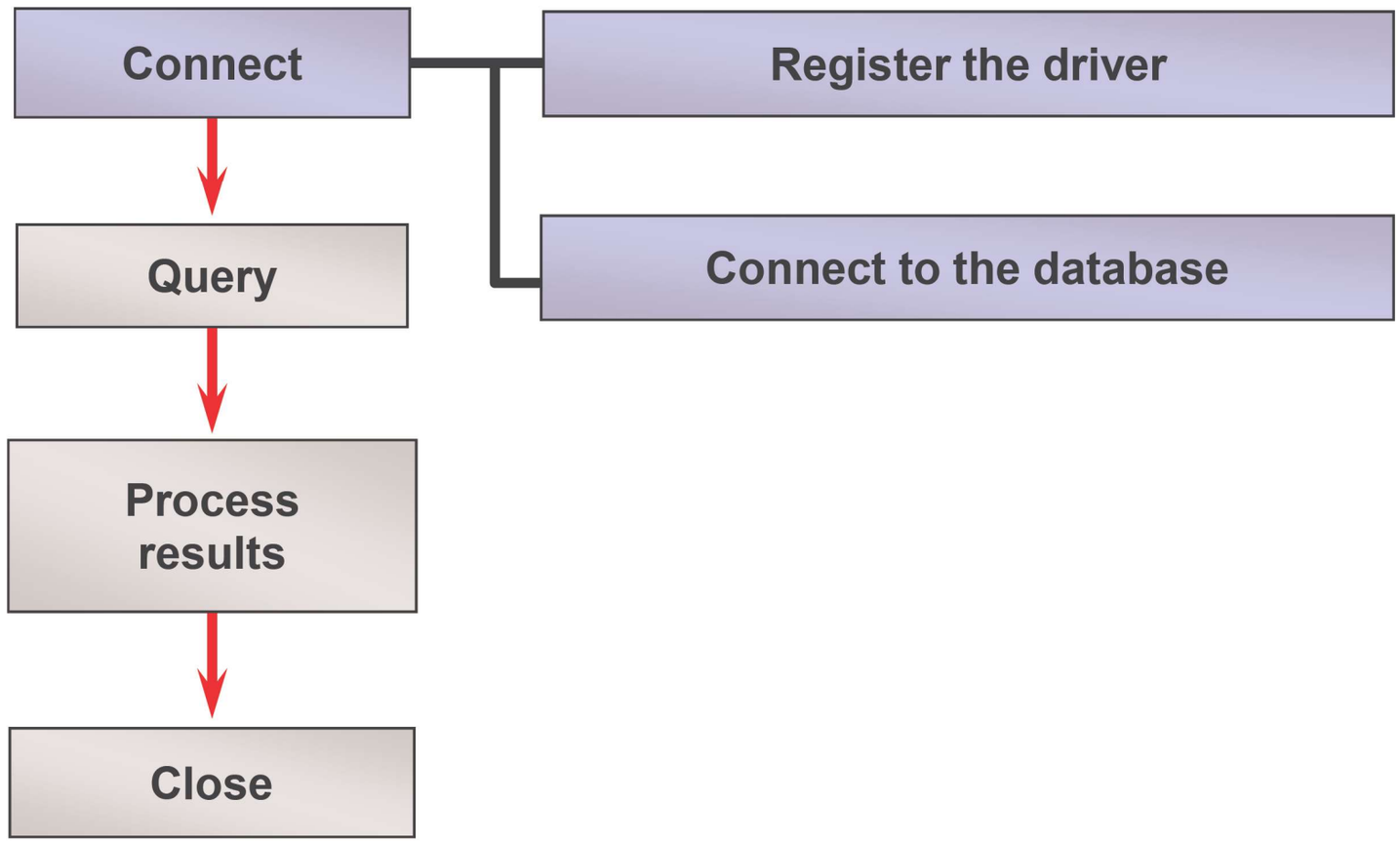


Architecture and Querying With JDBC





Stage 1: Connect

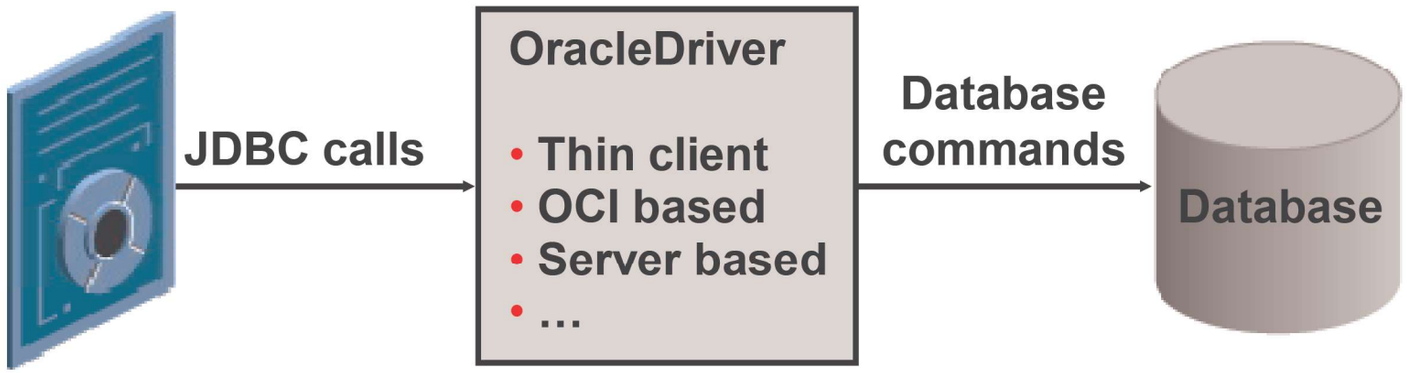


Step 1: Registering the Driver

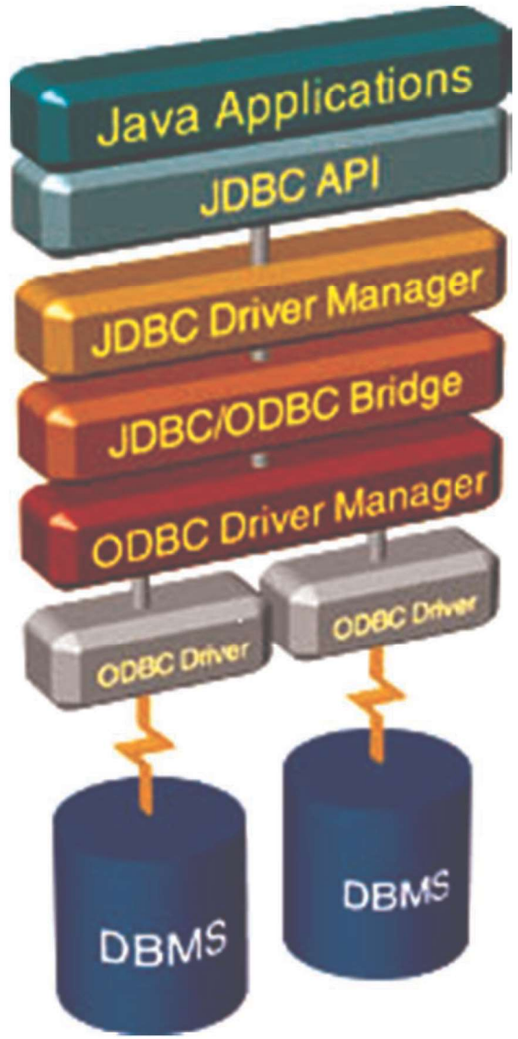
- Register the driver in the code:
  - `DriverManager.registerDriver (new oracle.jdbc.OracleDriver());`
  - `Class.forName ("oracle.jdbc.OracleDriver");`
- Register the driver when launching the class:
  - `java -D jdbc.drivers = oracle.jdbc.OracleDriver <ClassName>;`

Connecting to the Database

- Using the `oracle.jdbc.driver` package, Oracle provides different drivers to establish a connection to the database.



1. JDBC-ODBC Bridge Driver (Type I Driver)



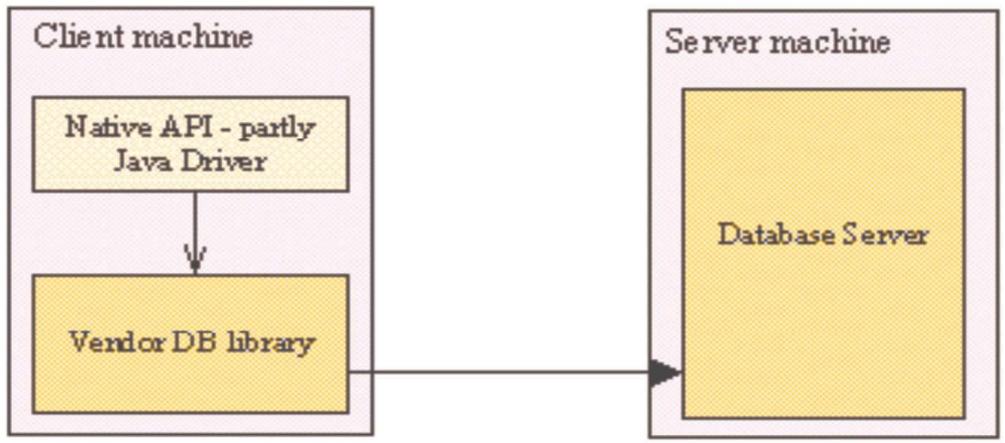
Why not use ODBC driver directly?

- ODBC was developed using C. Accesing ODBC directly from a Java program could be problematic due to usage of pointers and programming constructs.
- Rewriting ODBC in Java could be time consuming.

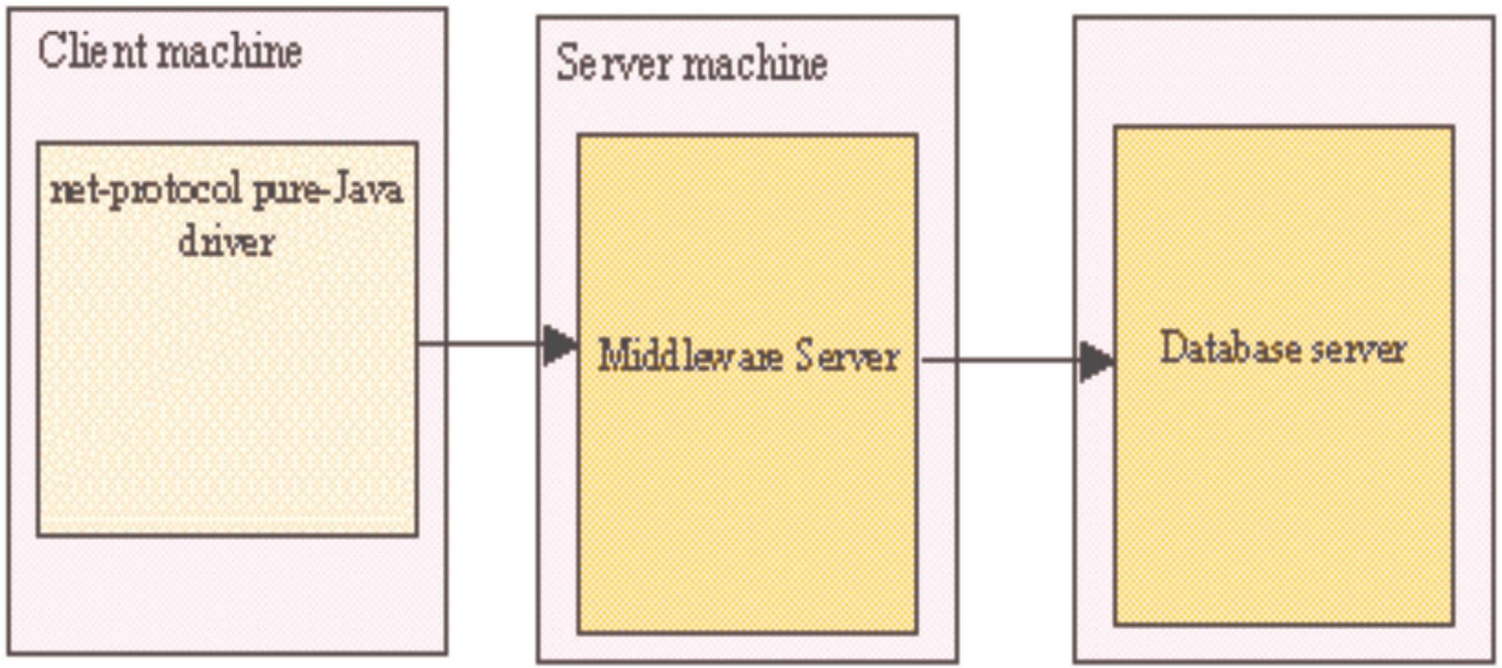
Native-API Partly Java Driver – Partly Java. It talks to database using server's native protocol

Ex: DB2 Driver uses IBM Database protocol

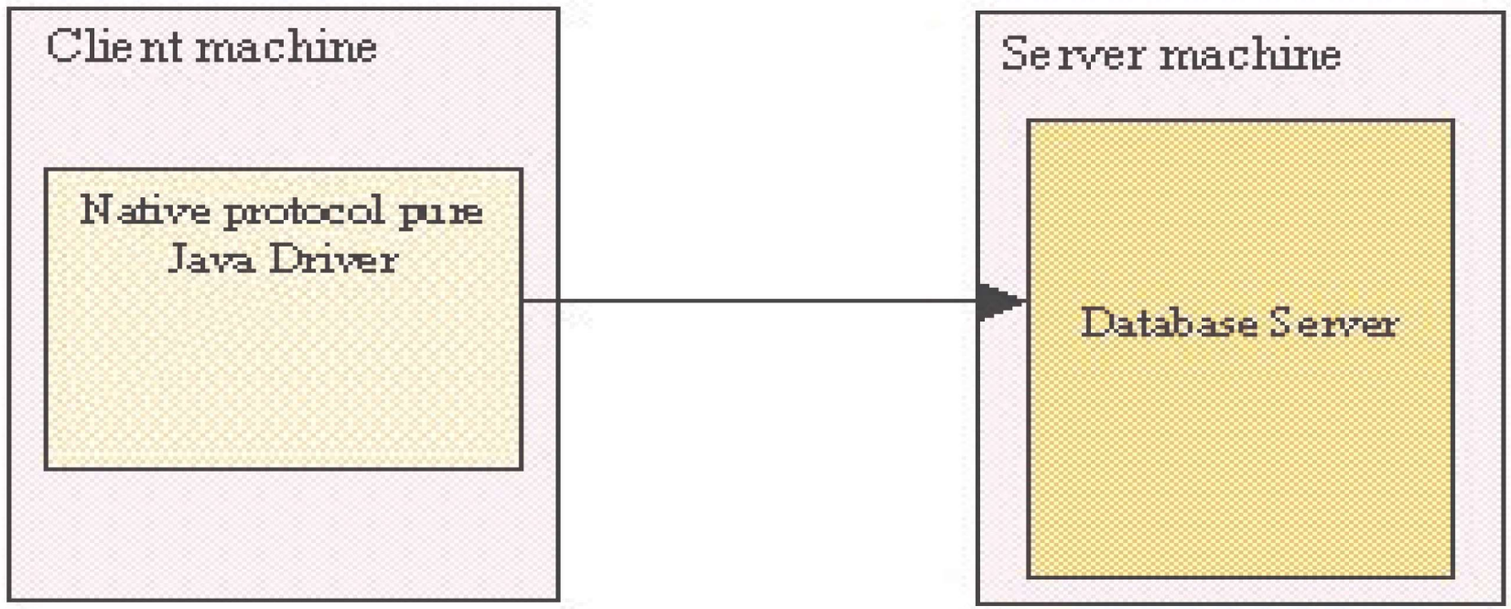
Oracle Driver uses SQLNet protocol.



JDBC-Net Pure Java Driver – uses standard protocol (ex: HTTP). It communicate to a database access server which translates to database specific protocol. Ex: IDS JDBC Driver

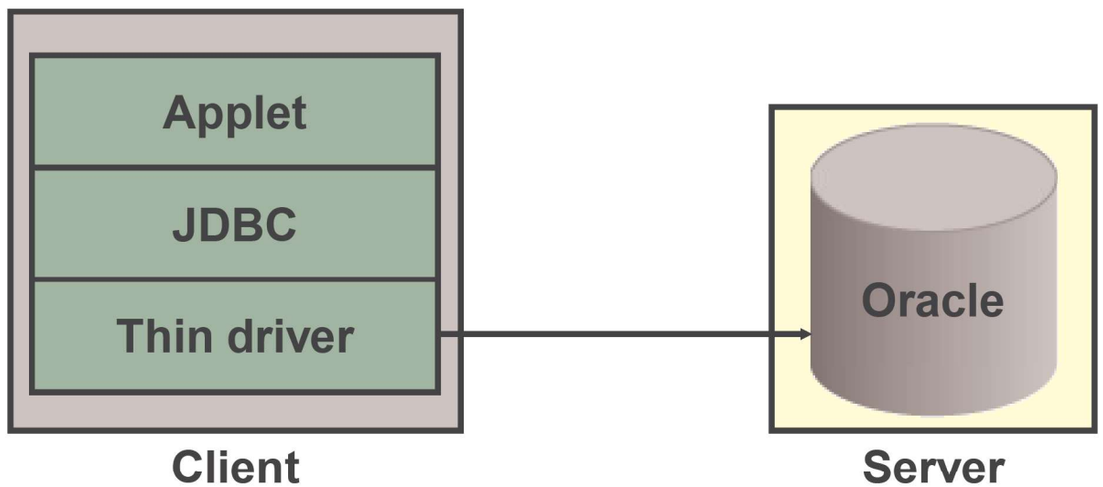


- **Native-Protocol Pure Java Driver** – It uses database
- specific protocol. Ex: MM.MySQL



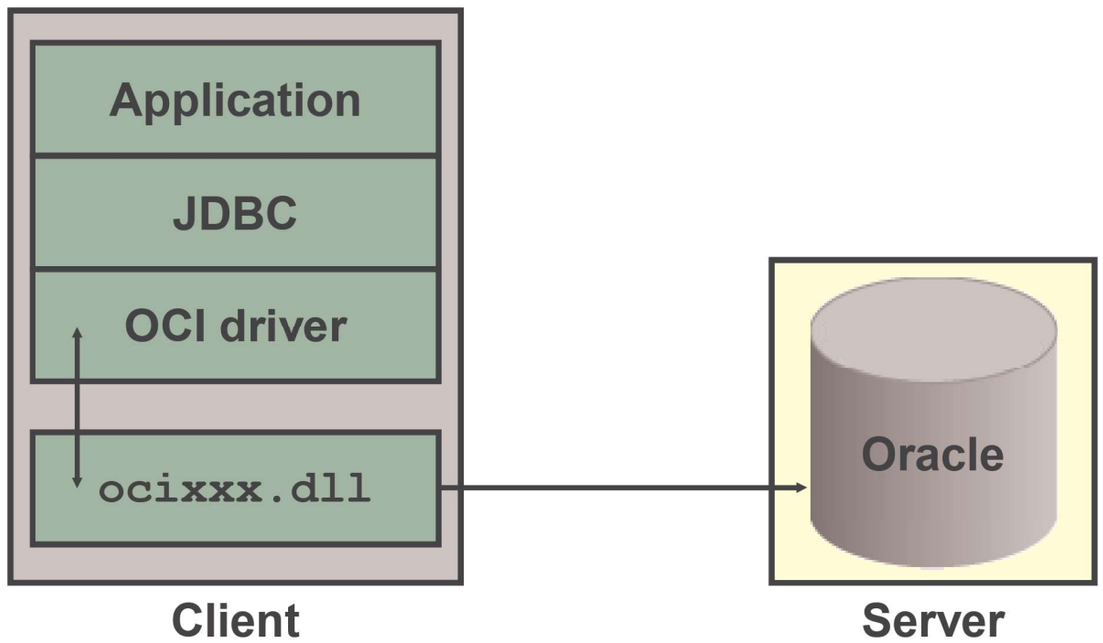
Oracle JDBC Drivers: Thin-Client Driver

- Is written entirely in Java
- Must be used by applets



Oracle JDBC Drivers: OCI Client Driver

- Is written in C and Java
- Must be installed on the client



Choosing the Right Driver

Type of Program	Driver	
Applet	Thin	
Client application	Thin	OCI
EJB, servlet (on the middle tier)	Thin	
	OCI	
Stored procedure	Server side	

Step 2: Obtaining a Database Connection

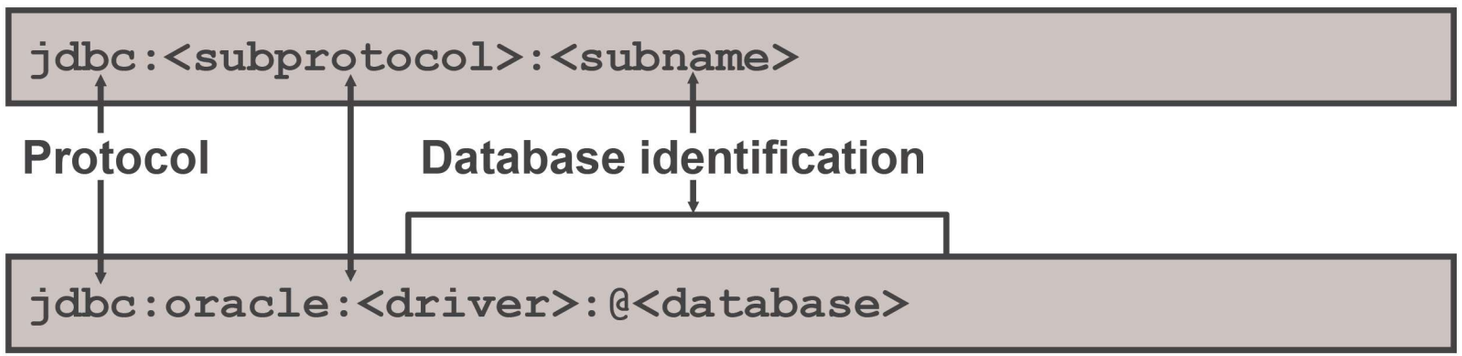
- In JDBC 1.0, use the `DriverManager` class, which provides overloaded `getConnection()` methods.
  - All connection methods require a JDBC URL to specify the connection details.
- Example:

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

- Vendors can provide different types of JDBC drivers.

JDBC URLs

- JDBC uses a URL-like string. The URL identifies:
  - The JDBC driver to use for the connection
  - Database connection details, which vary depending on the driver used



- Example using the Oracle JDBC Thin driver:
  - `jdbc:oracle:thin:@myhost:1521:ORCL`

JDBC URLs with Oracle Drivers

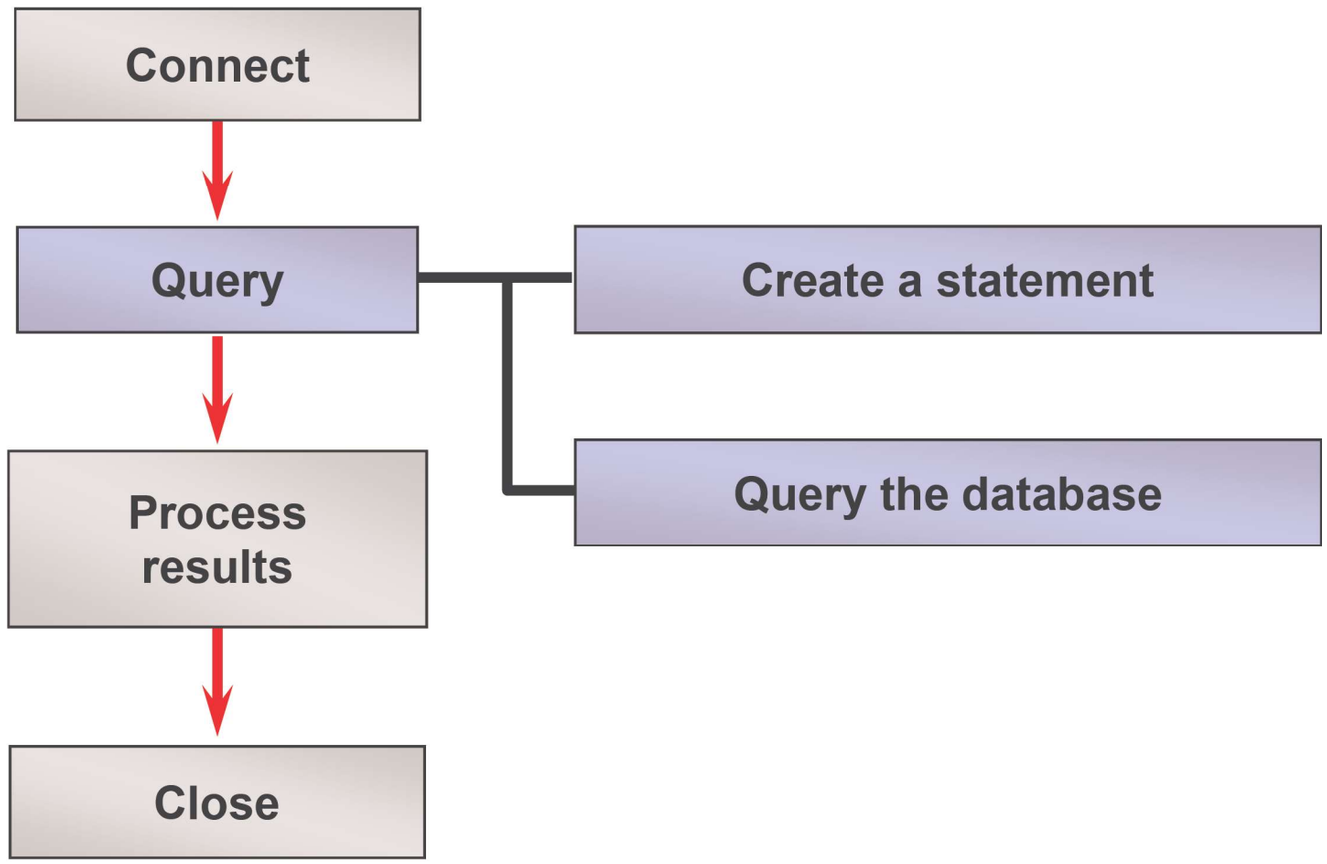
- Oracle JDBC Thin driver

```
Syntax:   jdbc:oracle:thin:@<host>:<port>:<SID>
Example:  "jdbc:oracle:thin:@myhost:1521:orcl"
```

- Oracle JDBC OCI driver

```
Syntax:   jdbc:oracle:oci:@<tnsname entry>
Example:  'jdbc:oracle:oci:@orcl"
```

Stage 2: Query



How to 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);
```

Step 3: Creating a Statement

JDBC statement objects are created from the `Connection` instance:

- Use the `createStatement()` method, which provides a context for executing a SQL statement.
- Example:

```
Connection conn =
DriverManager.getConnection(
       "jdbc:oracle:thin:@myhost:1521:ORCL",
       "scott","tiger");
Statement stmt = conn.createStatement();
```

Using the `Statement` Interface

The `Statement` interface provides three methods to execute SQL statements:

- Use `executeQuery(String sql)` for `SELECT` statements.
  - Returns a `ResultSet` object for processing rows
- Use `executeUpdate(String sql)` for DML or DDL.
  - Returns an `int`
- Use `execute(String)` for any SQL statement.
  - Returns a `boolean` value

Step 4a: Executing a Query

Provide a SQL query string, without semicolon, as an argument to the `executeQuery()` method.

- Returns a `ResultSet` object:

```
Statement stmt = null;
ResultSet rset = null;
  stmt = conn.createStatement();
  rset = stmt.executeQuery
      ("SELECT ename FROM emp");
```

Querying the Database: Examples

Execute a select statement

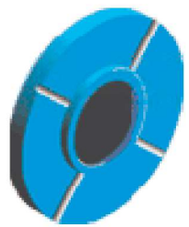
```
Statement stmt = conn.createStatement();
ResultSet rset = stmt.executeQuery
  ("select RENTAL_ID, STATUS from ACME_RENTALS");
```

Execute a delete statement

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

`ResultSet` Object

- The JDBC driver returns the results of a query in a `ResultSet` object.
- `ResultSet`:
  - Maintains a cursor pointing to its current row of data
  - Provides methods to retrieve column values



Step 4b: Submitting DML Statements

1. Create an empty statement object.

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

2. Use `executeUpdate` to execute the statement.

```
int count = stmt.executeUpdate(SQLDMLstatement);
```

Example:

```
Statement stmt = conn.createStatement();
int rowcount = stmt.executeUpdate
  ("DELETE FROM order_items
      WHERE order_id = 2354");
```

Step 4b: Submitting DDL Statements

1. Create an empty statement object.

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

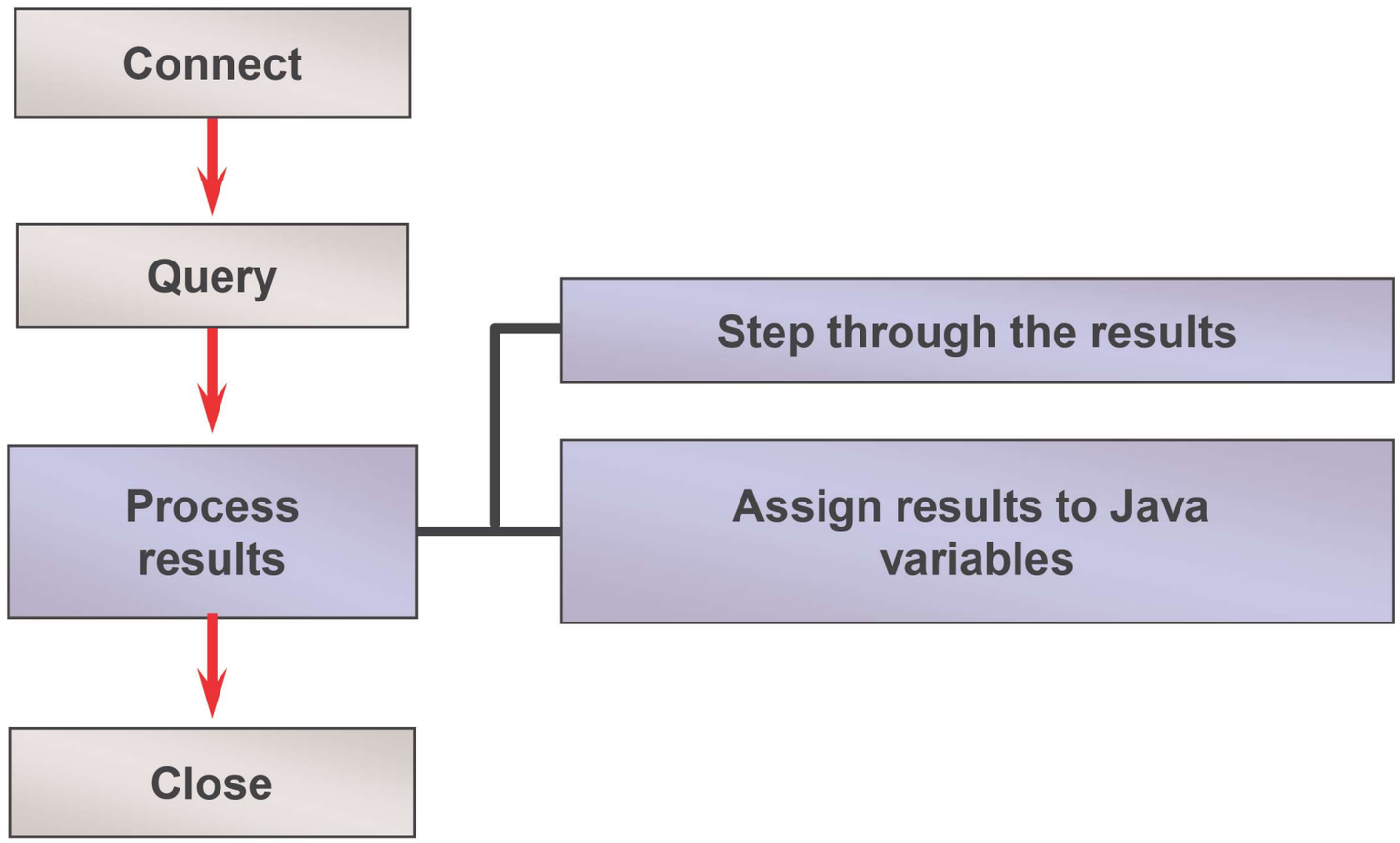
2. Use `executeUpdate` to execute the statement.

```
int count = stmt.executeUpdate(SQLDDLstatement);
```

Example:

```
Statement stmt = conn.createStatement();
int rowcount = stmt.executeUpdate
  ("CREATE TABLE temp (col1 NUMBER(5,2),
      col2 VARCHAR2(30)");
```

Stage 3: Process the Results



How to 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
}
```

Step 5: Processing the Query Results

The `executeQuery()` method returns a `ResultSet`.

- Use the `next()` method in loop to iterate through rows.
- Use `getXXX()` methods to obtain column values by column name or by column position in query.

```
stmt = conn.createStatement();
rset = stmt.executeQuery(
          "SELECT ename FROM emp");
while (rset.next()) {
    System.out.println
(rset.getString("ename"));
}
```

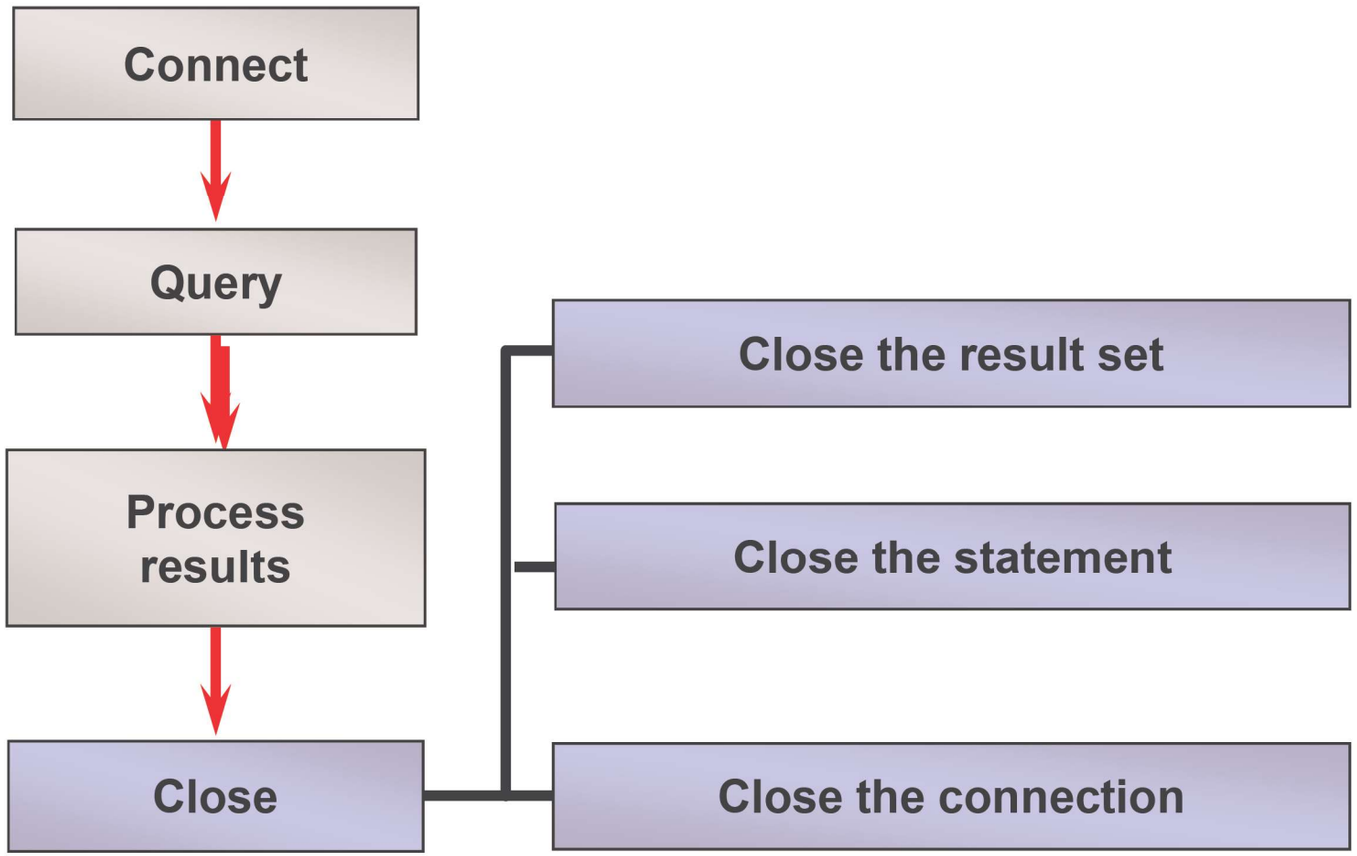
Mapping Database Types to Java Types

`ResultSet` maps database types to Java types:

```
ResultSet rset = stmt.executeQuery
  ("SELECT empno, hiredate, job
  FROM emp");
while (rset.next()){
int id = rset.getInt(1);
Date hiredate = rset.getDate(2);
String job = rset.getString(3);
```

Column Name	Type	Method
empno	NUMBER	getInt()
hiredate	DATE	getDate()
job	VARCHAR2	getString()

Stage 4: Close



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();
```

Step 6: Cleaning Up

Explicitly close the `Connection`, `Statement`, and `ResultSet` objects to release resources that are no longer needed.

- Call their respective `close()` methods:

```
Connection conn = ...;
Statement stmt = ...;
ResultSet rset = stmt.executeQuery(
          "SELECT ename FROM emp");
...
// clean up
rset.close();
stmt.close();
conn.close();
...
```

Basic Query Example

```
import java.sql.*;
  class TestJdbc {
    public static void main (String args [ ]) throws
SQLException {
      DriverManager.registerDriver (new
oracle.jdbc.OracleDriver());
      Connection conn = DriverManager.getConnection
     ("jdbc:oracle:thin:@myHost:1521:ORCL","scott",
"tiger");
      Statement stmt = conn.createStatement ();
      ResultSet rset = stmt.executeQuery
      ("SELECT ename        FROM emp");
      while (rset.next ())
              System.out.println (rset.getString
("ename"));
      rset.close();
      stmt.close();
      conn.close();
      }
}
```

Handling an Unknown SQL Statement

1. Create an empty statement object.

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

2. Use `execute` to execute the statement.

```
boolean isQuery = stmt.execute(SQLstatement);
```

3. Process the statement accordingly.

```
if (isQuery) { // was a query - process results
  ResultSet r = stmt.getResultSet(); ...
}
else { // was an update or DDL - process result
 int count = stmt.getUpdateCount(); ...
}
```

Handling Exceptions

- SQL statements can throw a `java.sql.SQLException`.
- Use standard Java error-handling methods.

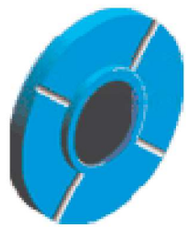
```
 try  {
     rset = stmt.executeQuery("SELECT empno,
       ename FROM emp");
       }
       catch (java.sql.SQLException e)
       { ... /* handle SQL errors */ }
...
finally {  // clean up
 try { if (rset != null) rset.close(); }
       catch (Exception e)
        { ... /* handle closing errors */ }
...
```

Transactions with JDBC

- By default, connections are in autocommit mode.
- Use `conn.setAutoCommit(false)` to disable autocommit.
- To control transactions when you are not in autocommit mode, use:
  - `conn.commit()` to commit a transaction
  - `conn.rollback()` to roll back a transaction
- Closing a connection commits the transaction even with autocommit disabled.

`PreparedStatement` Object

- A prepared statement prevents reparsing of SQL statements.
- Use the `PreparedStatement` object for statements that you want to execute more than once.
- A prepared statement can contain variables that you supply each time you execute the statement.



Creating a `PreparedStatement` Object

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

```
PreparedStatement pstmt =
  conn.prepareStatement
("UPDATE emp SET ename = ? WHERE empno = ?");
```

```
PreparedStatement pstmt =
  conn.prepareStatement
("SELECT ename FROM emp WHERE empno = ?");
```

Executing a `PreparedStatement` Object

1. Supply values for the variables.

```
pstmt.setXXX(Occurance of PlaceHolder, value);
```

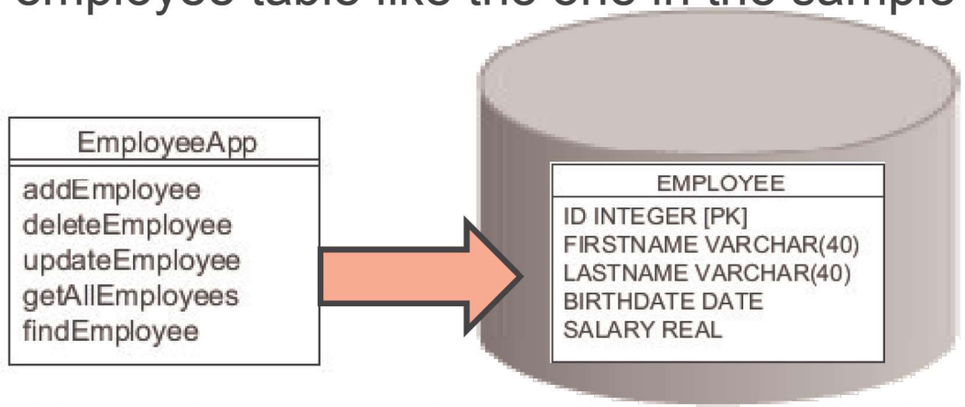
2. Execute the statement.

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

```
int empNo = 3521;
PreparedStatement pstmt =
  conn.prepareStatement("UPDATE emp
  SET ename = ? WHERE empno = ? ");
pstmt.setString(1, "DURAND");
pstmt.setInt(2, empNo);
pstmt.executeUpdate();
```

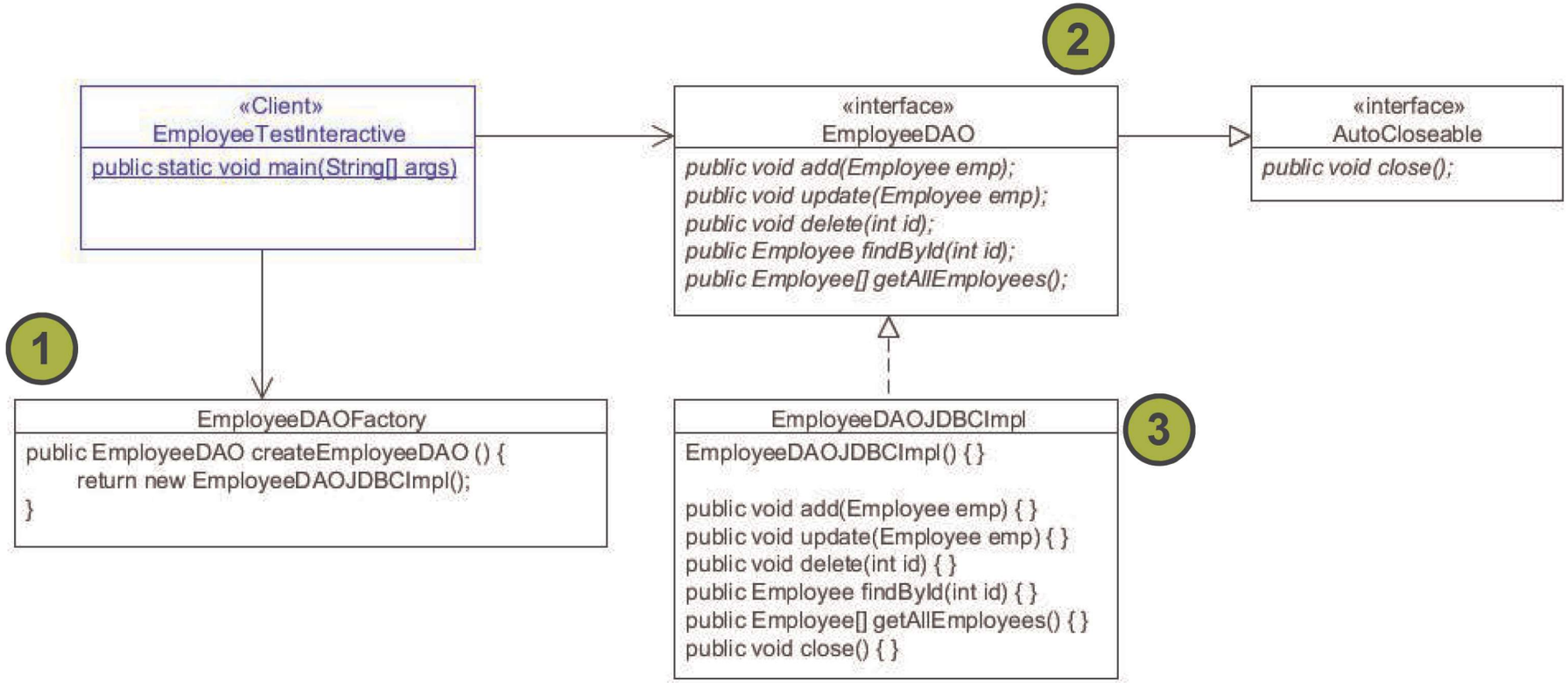
Data Access Objects

Consider an employee table like the one in the sample JDBC code.

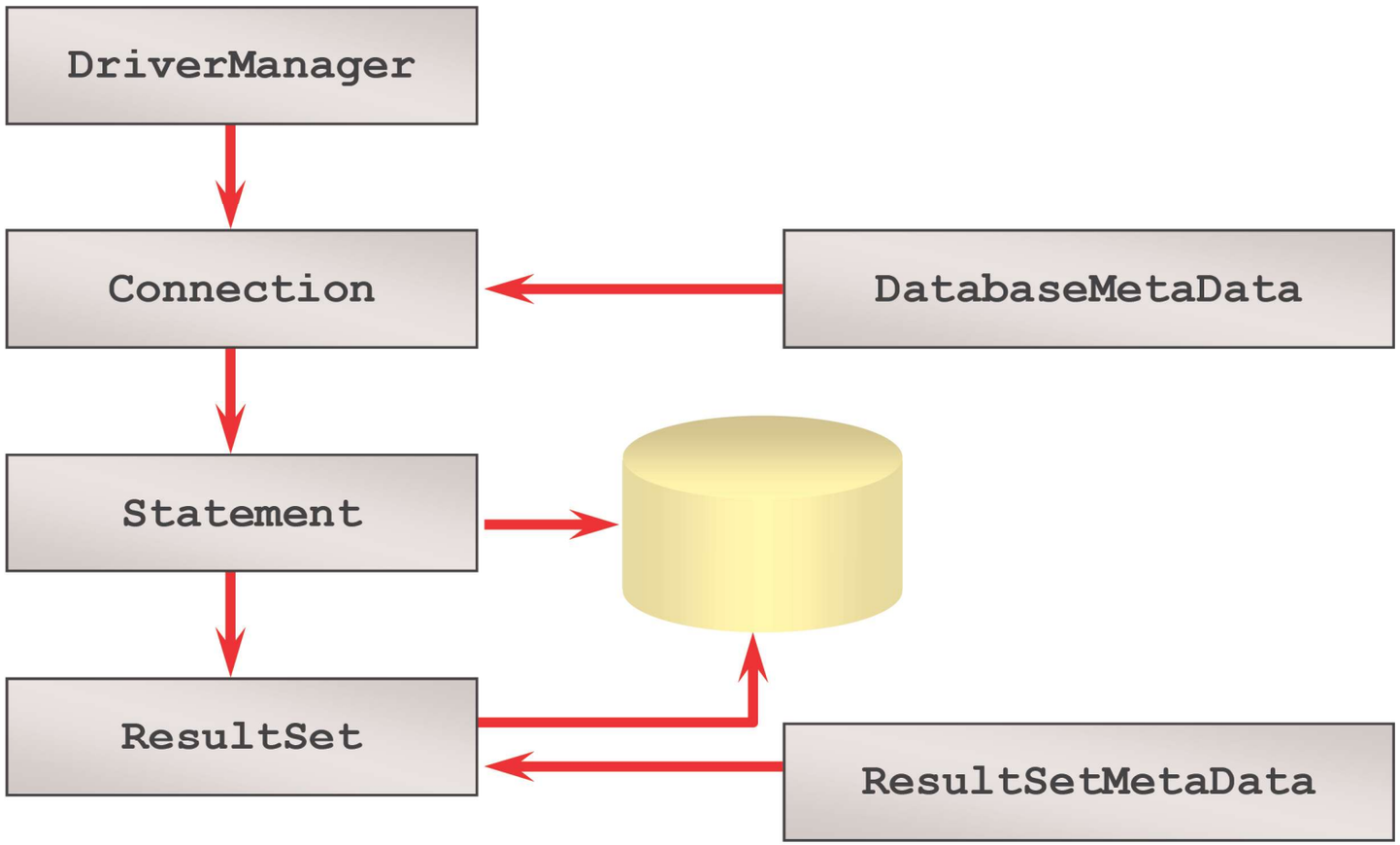


- By combining the code that accesses the database with the "business" logic, the data access methods and the Employee table are tightly coupled.
- Any changes to the table (such as adding a field) will require a complete change to the application.
- Employee data is not encapsulated within the example application.

The Data Access Object Pattern



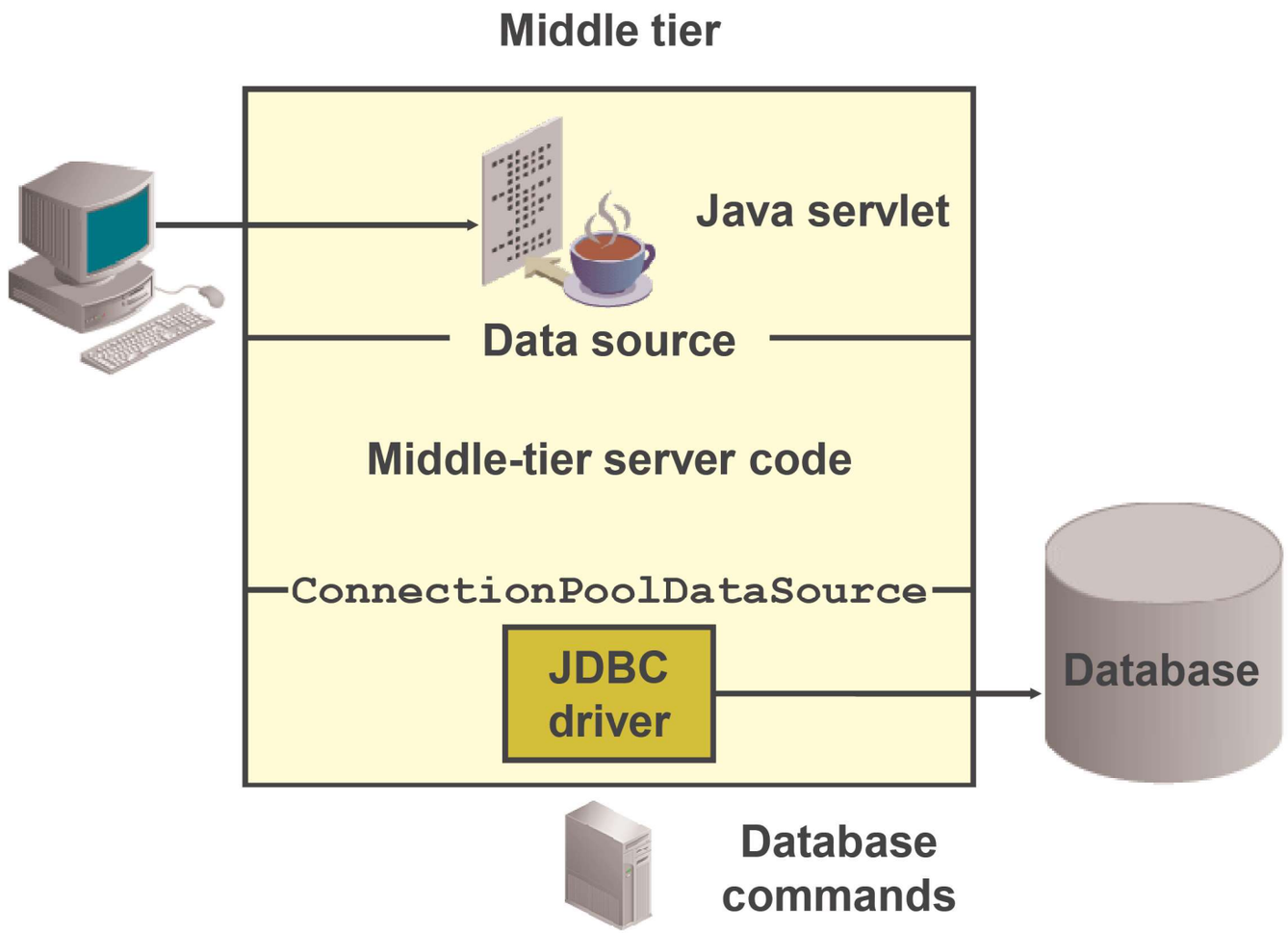
Summary of JDBC Classes



Maximizing Database Access

- Use connection pooling to minimize the operation costs of creating and closing sessions.
- Use explicit data source declaration for physical reference to the database.
- Use the `getConnection()` method to obtain a logical connection instance.

Connection Pooling



Summary

In this lesson, you should have learned the following:

- JDBC provides database connectivity for various Java constructs, including servlets and client applications.
- JDBC is a standard Java interface and part of J2SE.
- The steps for using SQL statements in Java are Register, Connect, Submit, and Close.
- SQL statements can throw exceptions.
- You can control the behavior of default transactions.

Practice : Overview

This practice covers the following topics:

- Setting up the Java environment for JDBC
- Adding JDBC components to query the database