

# Using JDBC to Access the Database

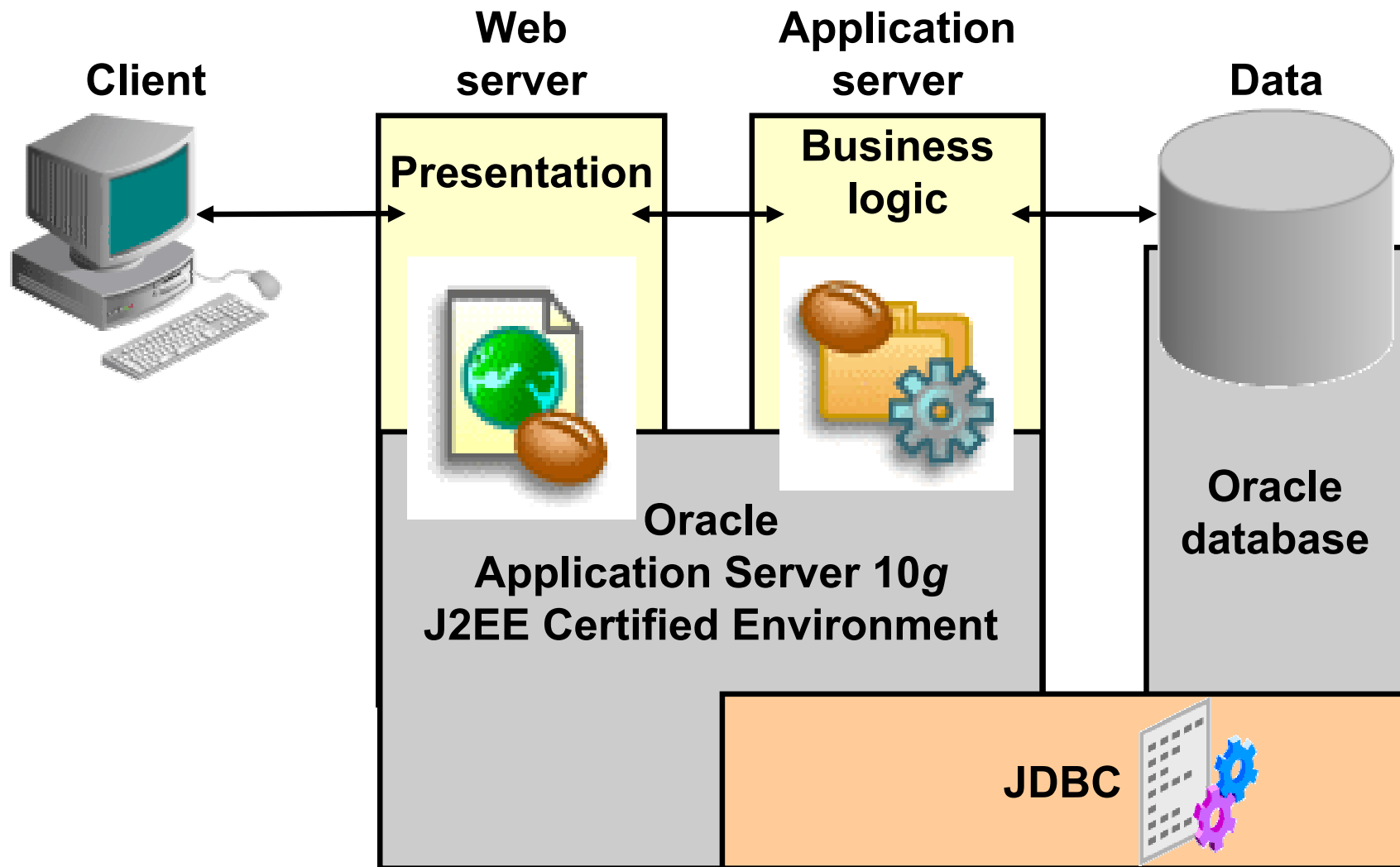
16

# Objectives

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

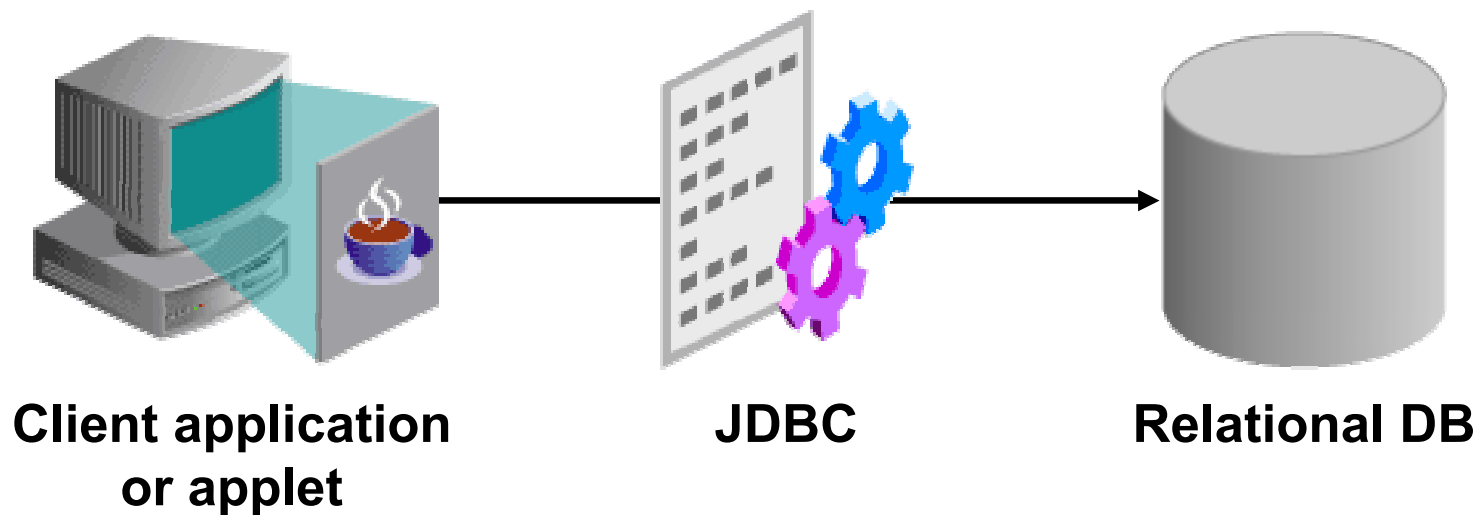
- **Describe how Java applications connect to the database**
- **Describe how Java database functionality is supported by the Oracle database**
- **Load and register a JDBC driver**
- **Connect to an Oracle database**
- **Follow the steps to execute a simple `SELECT` statement**
- **Map simple Oracle database types to Java types**

# Java, J2EE, and Oracle 10g



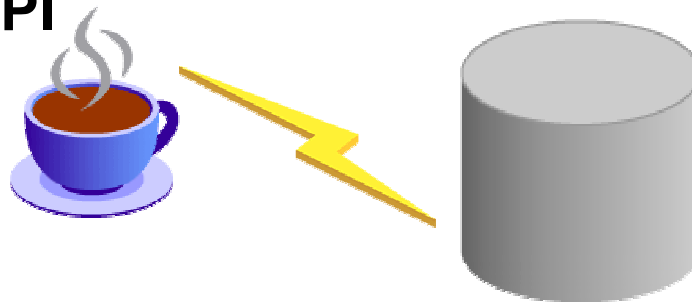
# Connecting to a Database with Java

Client applications, JSPs, and servlets use JDBC.



# What Is JDBC?

- **JDBC is a standard interface for connecting to relational databases from Java.**
  - The JDBC API includes Core API Package in `java.sql`.
  - JDBC 2.0 API includes 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 the Java 2, Standard Edition (J2SE).**

# Preparing the Environment

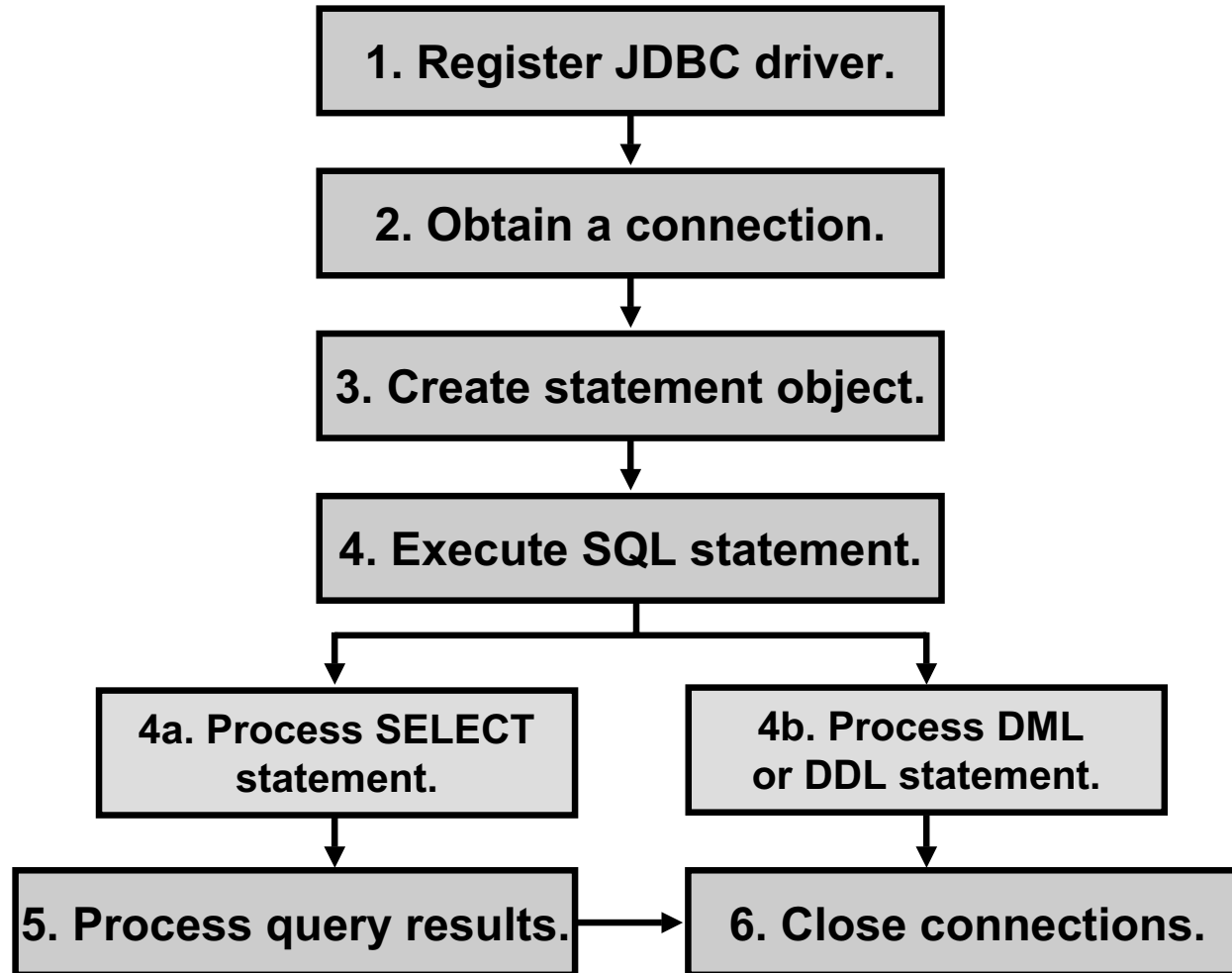
- **Set the CLASSPATH:**

```
[Oracle Home]\jdbc\lib\classes12.jar
```

- **Import JDBC packages:**

```
// 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

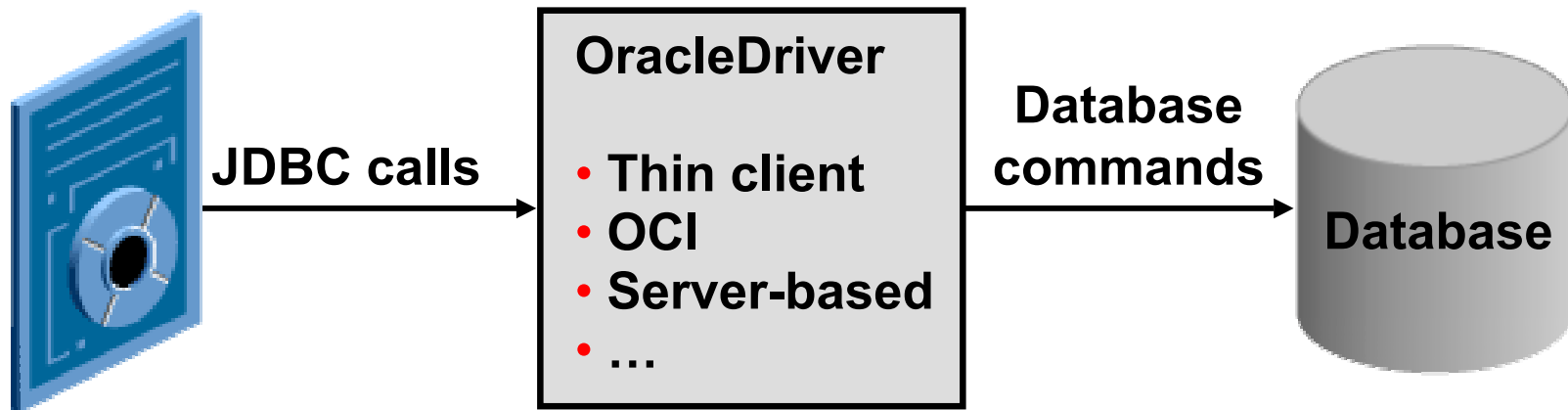


# Step 1: Registering the Driver

- **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>;`

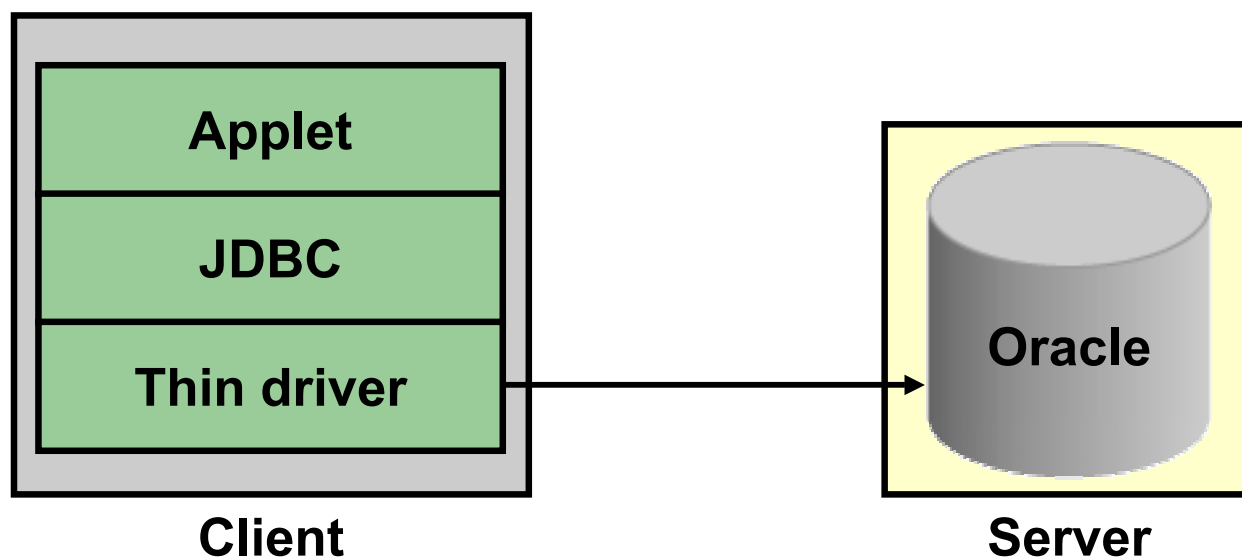
# Connecting to the Database

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



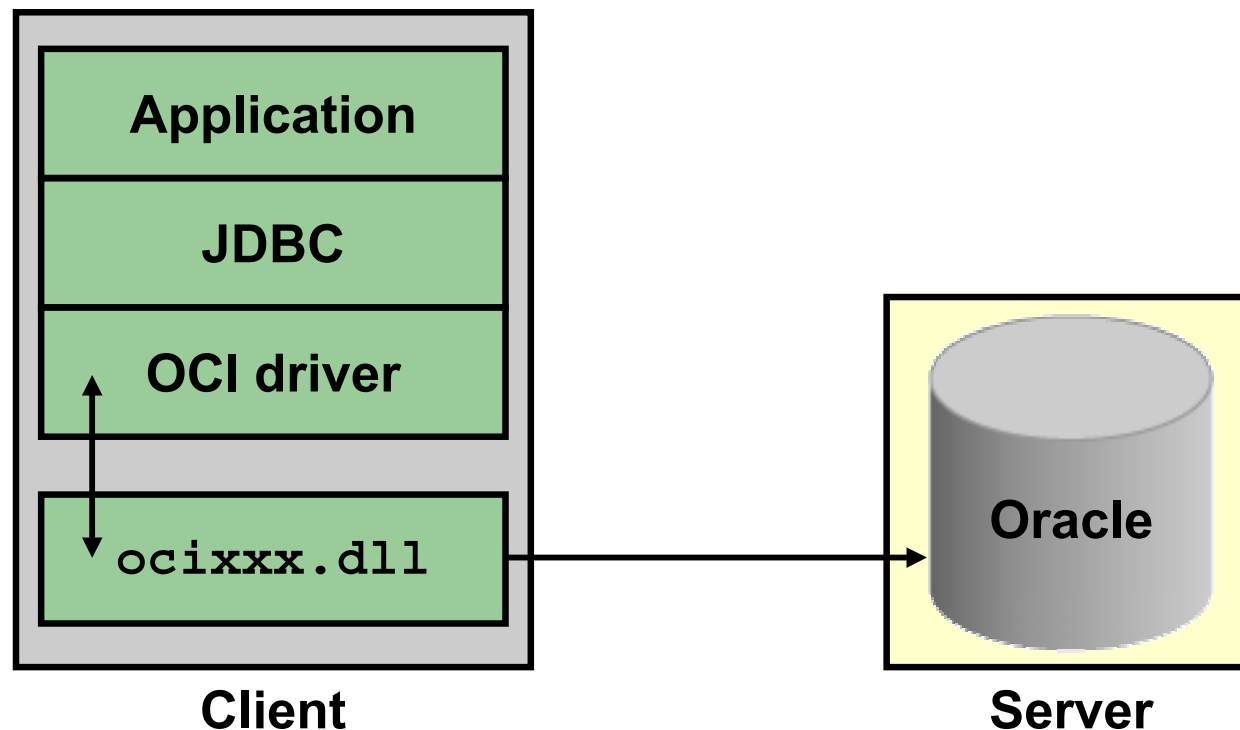
# Oracle JDBC Drivers: Thin Client Driver

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



# Oracle JDBC Drivers: OCI Client Drivers

- 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: Getting a Database Connection

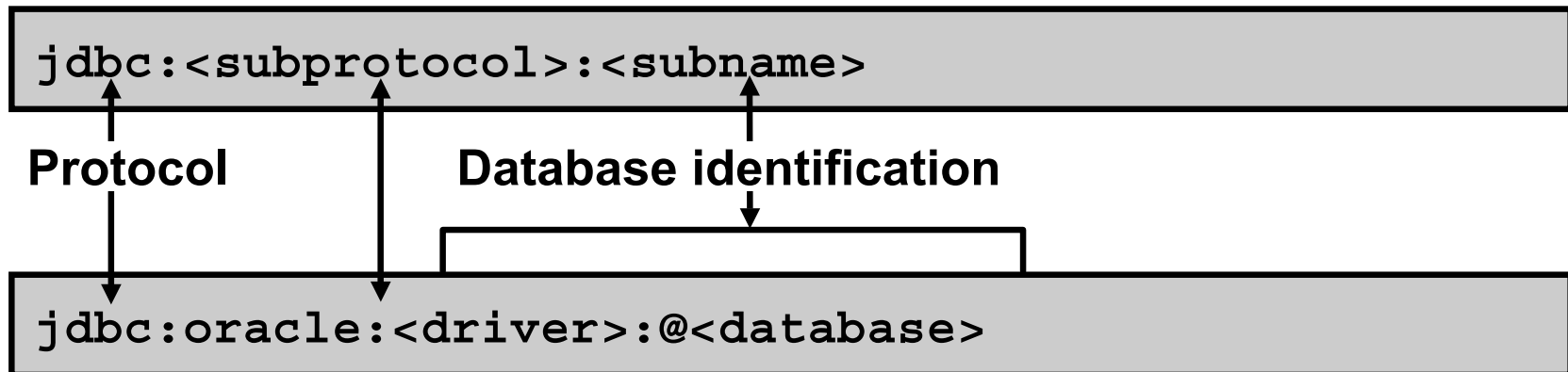
- In JDBC 1.0, use the `DriverManager` class, which provides overloaded `getConnection()` methods.
  - All connection methods require a JDBC URL to specify the connection details.
- Example:

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

- Vendors can provide different types of JDBC drivers.

# About 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 Oracle Thin JDBC driver:**
  - `jdbc:oracle:thin:@myhost:1521:ORCL`

# JDBC URLs with Oracle Drivers

- **Oracle Thin driver**

Syntax: `jdbc:oracle:thin:@<host>:<port>:<SID>`

Example: `"jdbc:oracle:thin:@myhost:1521:orcl"`

- **Oracle OCI driver**

Syntax: `jdbc:oracle:oci:@<tnsname entry>`

Example: `"jdbc:oracle:oci:@orcl"`

## Step 3: Creating a Statement

JDBC statement objects are created from the Connection instance:

- Use the `createStatement()` method, which provides a context for executing an 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");
```

# The 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)");
```

# 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 position in query, or column name.

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

# Step 6: Closing Connections

**Explicitly close a Connection, Statement, and ResultSet object 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();
...
```

# A 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();
    }
}
```

# 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()

# 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 */ }
}

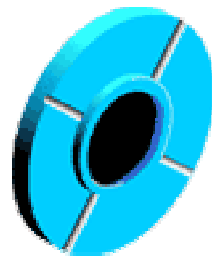
...
```

# Managing Transactions

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

# The PreparedStatement Object

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



# How to Create a PreparedStatement

1. Register the driver and create the database connection.
2. Create the PreparedStatement, 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 = ?");
```

# How to Execute a PreparedStatement

## 1. Supply values for the variables:

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

## 2. Execute the statement:

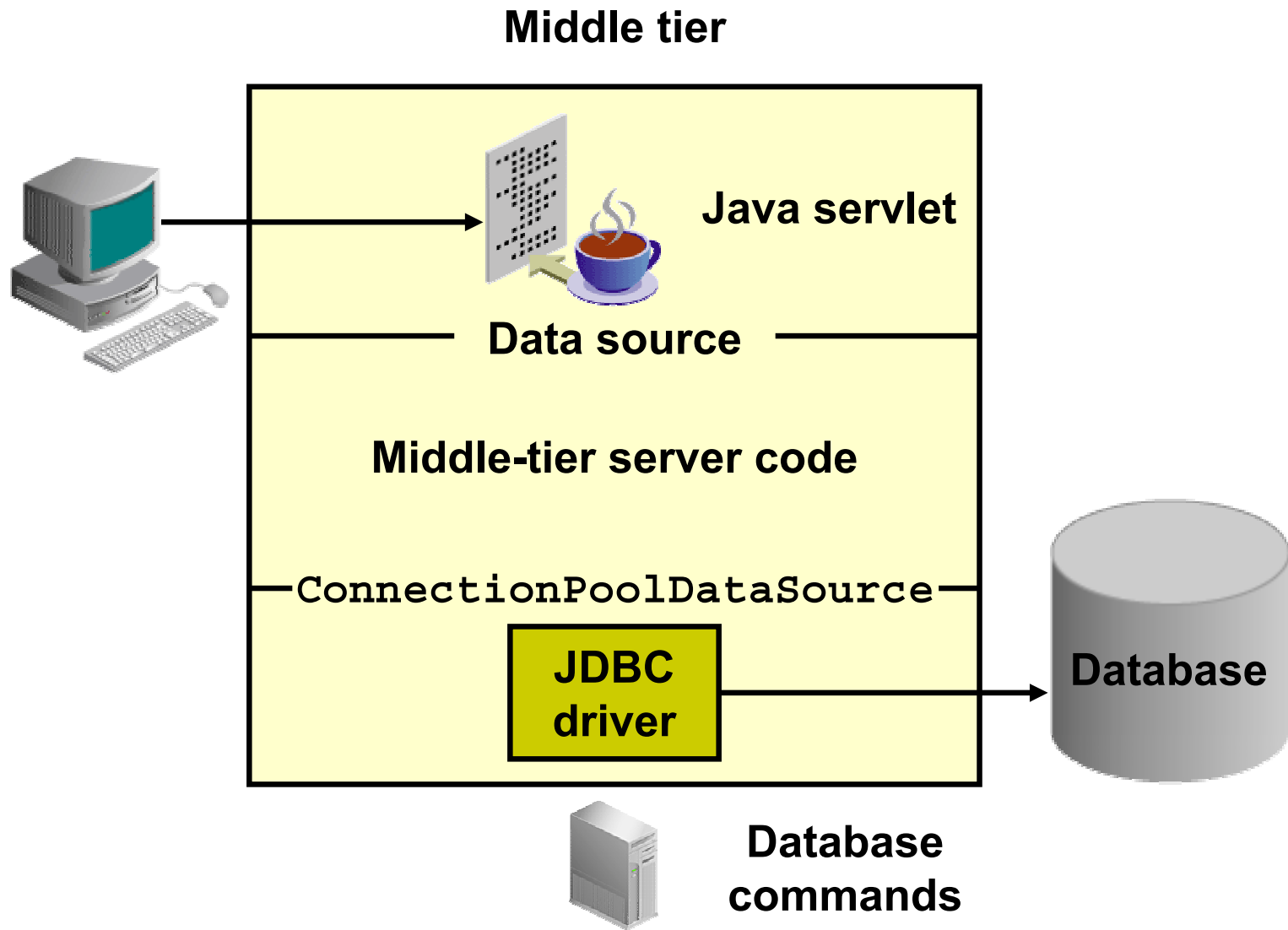
```
pstmt.executeQuery();  
pstmt.executeUpdate();
```

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

# Maximize 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 the J2SE.**
- **The steps for using SQL statements in Java are Register, Connect, Submit, and Close.**
- **SQL statements can throw exceptions.**
- **You can control default transactions behavior.**