JDBC

JDBC

- **JDBC** is a Java API for communicating with database systems supporting SQL.

- JDBC supports a variety of features for **querying** and **updating** data, and for retrieving query results.
- JDBC also supports **metadata retrieval**, such as querying about relations present in the database and the names and types of relation attributes.
- Model for communicating with the database:
  - Open a **connection**
  - Create a “statement” object
  - Execute queries using the **Statement** object to send queries and fetch results
  - Exception mechanism to handle errors

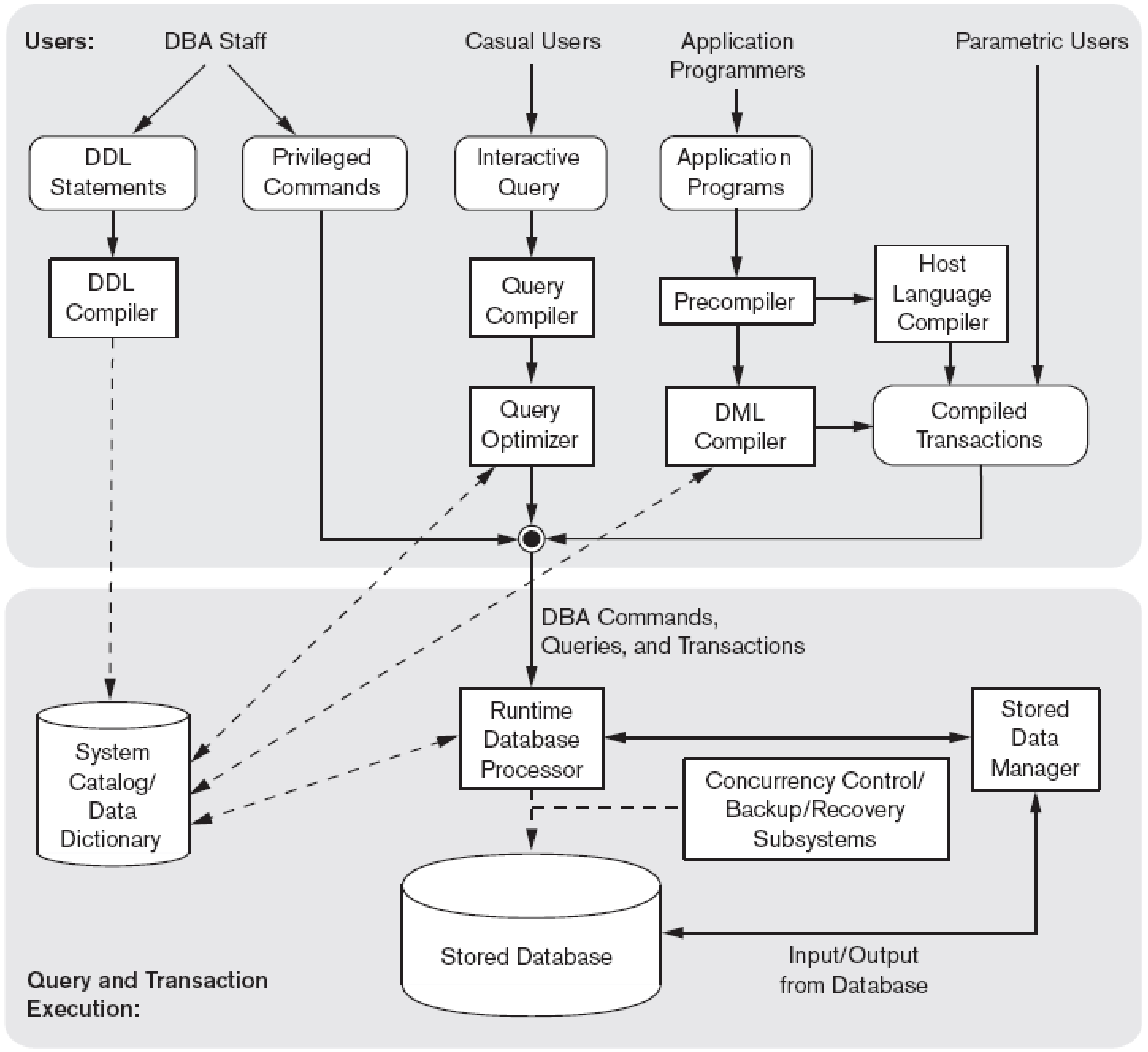
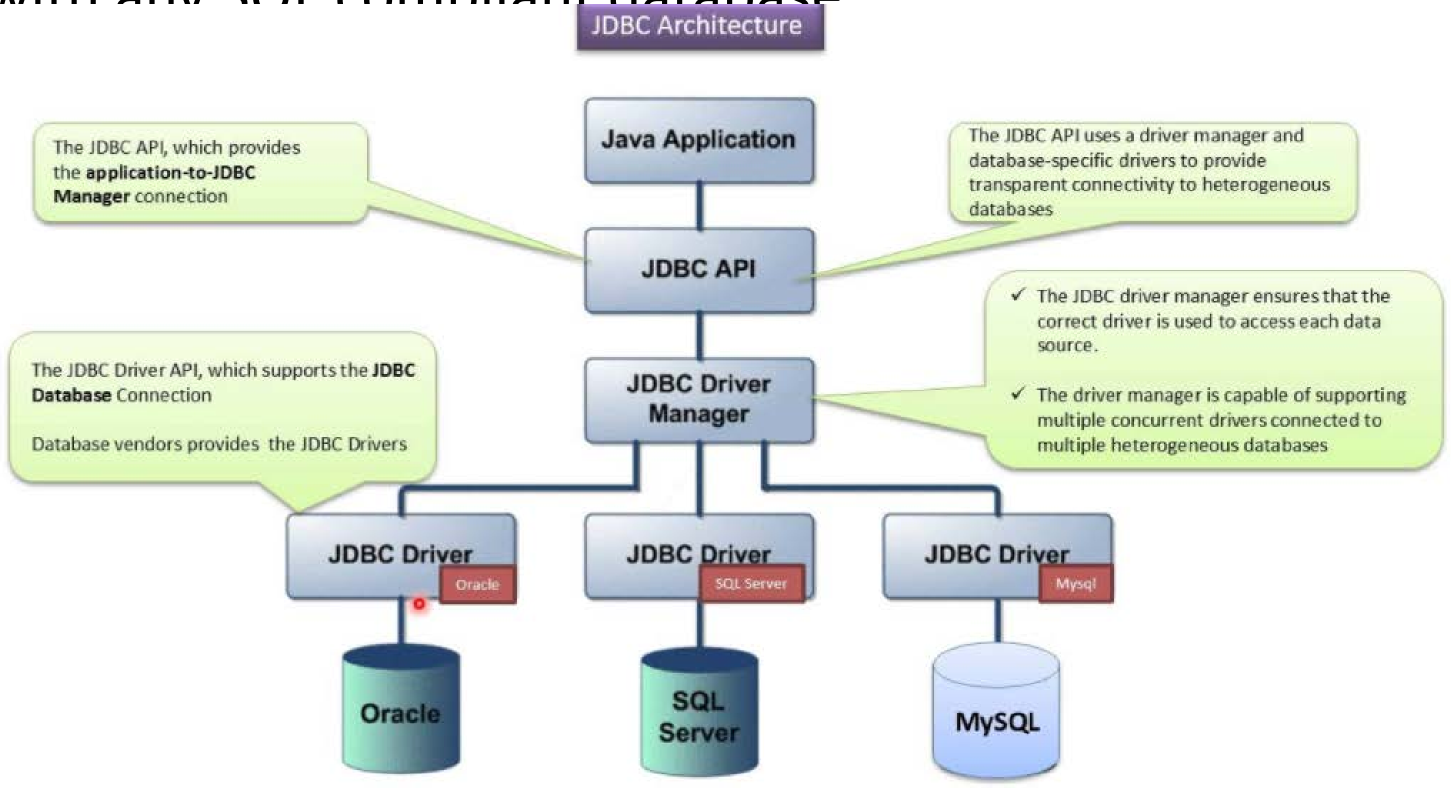


Figure 2.3
Component modules of a DBMS and their interactions.

What is JDBC?

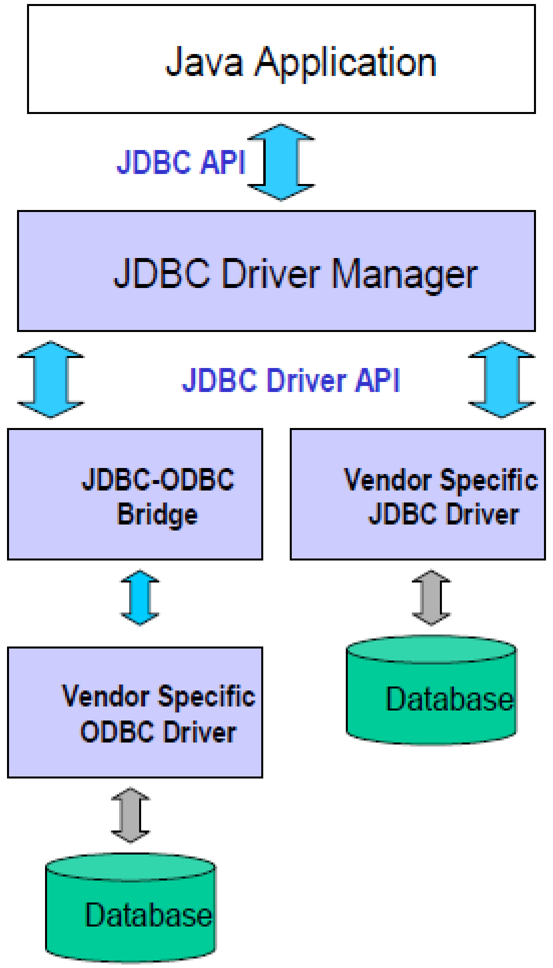
- JDBC stands for Java Database Connectivity and provides a set of **Java API** for accessing the relational databases from Java program.
  - These Java APIs enables Java programs to execute SQL statements and interact with any SQL compliant database


JDBC Architecture
Java Application
JDBC API
JDBC Driver Manager
JDBC Driver
Oracle
JDBC Driver
SQL Server
JDBC Driver
Mysql
Oracle
SQL Server
MySQL
The JDBC API, which provides the **application-to-JDBC Manager** connection
The JDBC API uses a driver manager and database-specific drivers to provide transparent connectivity to heterogeneous databases
✓ The JDBC driver manager ensures that the correct driver is used to access each data source.
✓ The driver manager is capable of supporting multiple concurrent drivers connected to multiple heterogeneous databases
The JDBC Driver API, which supports the **JDBC Database** Connection
Database vendors provides the JDBC Drivers

JDBC

- **JDBC consists of two parts:**
  1. JDBC API, a purely Java-based API
  2. JDBC Driver Manager ,which communicates with vendor-specific drivers that perform the real communication with the database.

  - Point: translation to vendor format is performed on the client
    - No changes needed to server
    - Driver (translator) needed on client



Seven Basic Steps in using JDBC

1. **Load the driver**
2. **Define the Connection URL**
3. **Establish the Connection**
4. **Create a Statement object**
5. **Execute a query**
6. **Process the results**
7. **Close the connection**

Why JDBC?

- to communicate with the different DBMS from JDBC, we need to use a *driver* to isolate the specific features of the DBMS and its communication protocol.

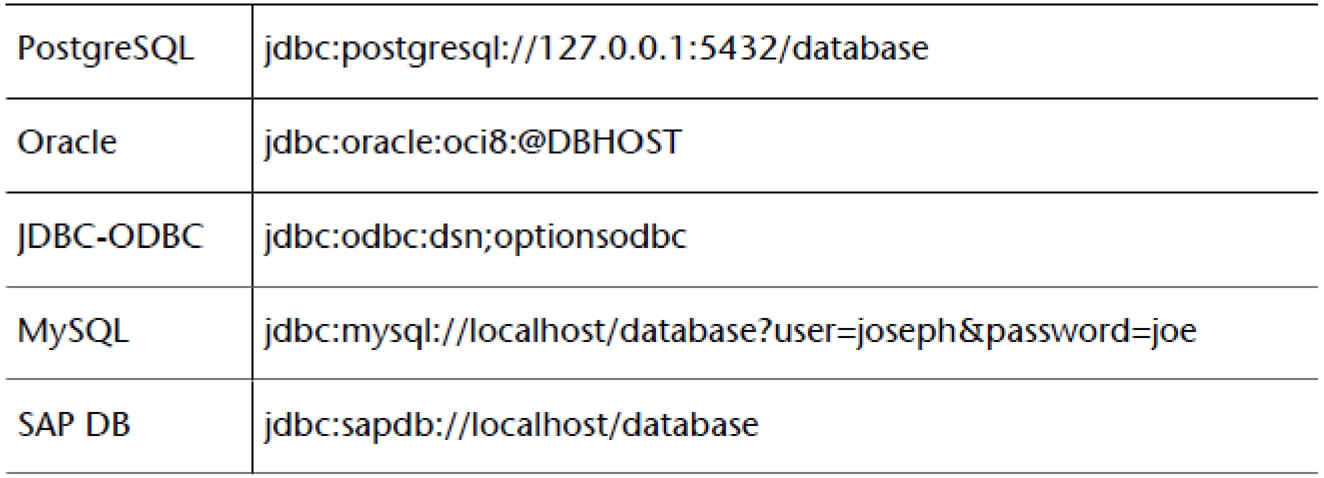
Step 1: Load the Driver

- To use a JDBC driver, we must first register it in the JDBC DriverManager.
  - This is usually done by loading the driver class using the forName method of the class called Class.
- **try {**

  ```
  Class.forName("com.mysql.jdbc.Driver");
  Class.forName("oracle.jdbc.driver.OracleDriver");
  } catch (ClassNotFoundException cnfe) {
  System.out.println("Error loading driver: " + cnfe);
  ```
- **}**

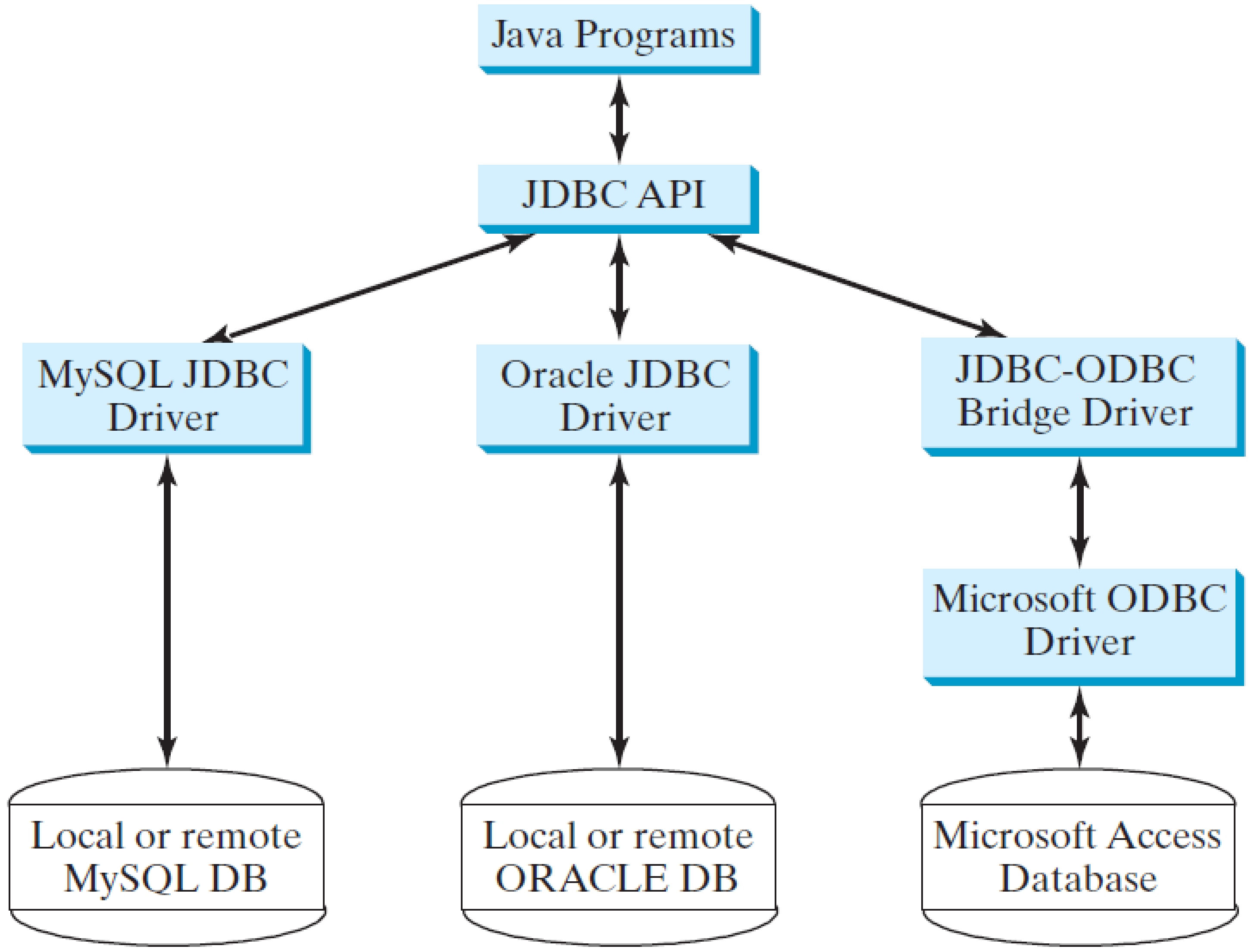
Step 2: Define the Connection URL

- To identify a given connection to a database, DBMS use a URL (Universal Resource Locator) address format. This address usually takes the form:
  - jdbc:driver:database

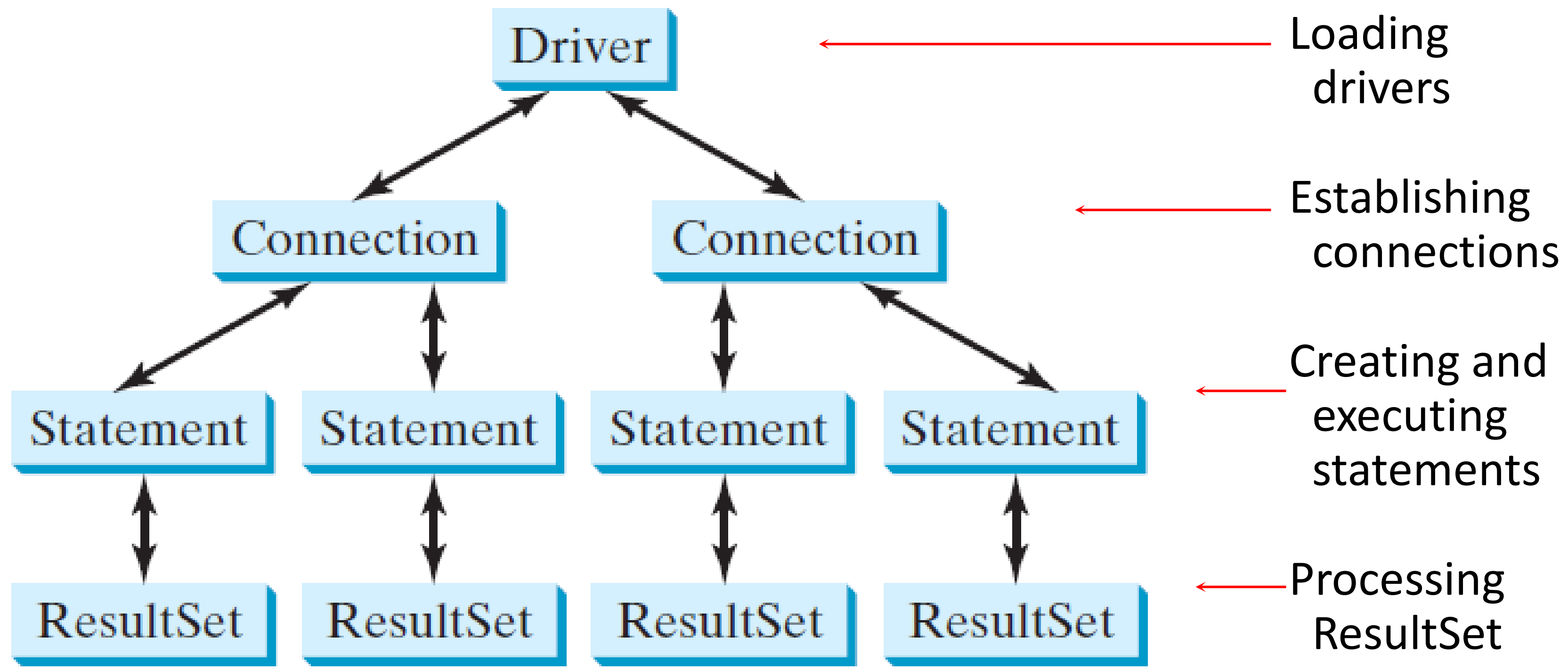


PostgreSQL	jdbc:postgresql://127.0.0.1:5432/database
Oracle	jdbc:oracle:oci8:@DBHOST
JDBC-ODBC	jdbc:odbc:dsn;optionsodbc
MySQL	jdbc:mysql://localhost/database?user=joseph&password=joe
SAP DB	jdbc:sapdb://localhost/database

The Architecture of JDBC



The JDBC Interfaces



Developing JDBC Programs

Loading drivers

Establishing connections

Creating and executing statements

Processing ResultSet

Statement to load a driver:

```
Class.forName("JDBCDriverClass");
```

A driver is a class. For example:

Database	Driver Class	Source
Access	sun.jdbc.odbc.JdbcOdbcDriver	Already in JDK
MySQL	com.mysql.jdbc.Driver	Website
Oracle	oracle.jdbc.driver.OracleDriver	Website

The JDBC-ODBC driver for Access is bundled in JDK.

MySQL driver class is in mysqljdbc.jar

Oracle driver class is in classes12.jar

To use the MySQL and Oracle drivers, you have to add mysqljdbc.jar and classes12.jar in the classpath using the following DOS command on Windows:

```
classpath=%classpath%;c:\book\mysqljdbc.jar;c:\book\classes12.jar
```

Developing JDBC Programs

Loading drivers

Establishing connections

Creating and executing statements

Processing ResultSet

```
Connection connection = DriverManager.getConnection(databaseURL);
```

Database	URL Pattern
Access	jdbc:odbc:dataSource
MySQL	jdbc:mysql://hostname/dbname
Oracle	jdbc:oracle:thin:@hostname:port#:oracleDBSID

Examples:

For Access:

```
Connection connection = DriverManager.getConnection
  ("jdbc:odbc:ExampleMDBDataSource");
```

For MySQL:

```
Connection connection = DriverManager.getConnection
  ("jdbc:mysql://localhost/test");
```

For Oracle:

```
Connection connection = DriverManager.getConnection
  ("jdbc:oracle:thin:@liang.armstrong.edu:1521:orcl", "scott", "tiger");
```

Developing JDBC Programs

Loading drivers

Establishing connections

Creating and executing statements

Processing ResultSet

Creating statement:

```
    Statement statement = connection.createStatement();
```

Executing statement (for update, delete, insert):

```
    statement.executeUpdate
      ("create table Temp (col1 char(5), col2 char(5))");
```

Executing statement (for select):

```
   // Select the columns from the Student table
   ResultSet resultSet = statement.executeQuery
    ("select firstName, mi, lastName from Student where lastName "
      + " = 'Smith'");
```

Developing JDBC Programs

Loading drivers

Establishing connections

Creating and executing statements

Processing ResultSet

Executing statement (for select):

```
    // Select the columns from the Student table
    ResultSet resultSet = stmt.executeQuery
      ("select firstName, mi, lastName from Student where lastName "
        + " = 'Smith'");
```

Processing ResultSet (for select):

```
    // Iterate through the result and print the student names
    while (resultSet.next())
      System.out.println(resultSet.getString(1) + " " + resultSet.getString(2)
        + ". " + resultSet.getString(3));
```

Simple JDBC Example

```
import java.sql.*;
public class SimpleJdbc {
  public static void main(String[] args)
      throws SQLException, ClassNotFoundException {
    // Load the JDBC driver
    Class.forName("com.mysql.jdbc.Driver");
    System.out.println("Driver loaded");

    // Establish a connection
    Connection connection = DriverManager.getConnection
      ("jdbc:mysql://localhost/test");
    System.out.println("Database connected");

    // Create a statement
    Statement statement = connection.createStatement();

    // Execute a statement
    ResultSet resultSet = statement.executeQuery
      ("select firstName, mi, lastName from Student where lastName "
        + " = 'Smith'");

    // Iterate through the result and print the student names
    while (resultSet.next())
      System.out.println(resultSet.getString(1) + "\t" +
        resultSet.getString(2) + "\t" + resultSet.getString(3));

    // Close the connection
    connection.close();
  }
}
```

JDBC Code

```
public static void JDBCexample(String dbid, String userid, String passwd)
    {
        try {
            Class.forName ("oracle.jdbc.driver.OracleDriver");
            Connection conn = DriverManager.getConnection(
                  "jdbc:oracle:thin:@db.yale.edu:2000:univdb", userid, passwd);
          Statement stmt = conn.createStatement();
            … Do Actual Work ….
          stmt.close();
          conn.close();
        }
        catch (SQLException sqle) {
          System.out.println("SQLException : " + sqle);
        }
    }
```

JDBC Code (Cont.)

- Update to database

  -
    ```
    try {
        stmt.executeUpdate (
            "insert into instructor values(’77987’, ’Kim’, ’Physics’, 98000)");
    } catch (SQLException sqle)
    {
       System.out.println("Could not insert tuple. " + sqle);
    }
    ```
- Execute query and fetch and print results

```
 ResultSet rset = stmt.executeQuery (
                       "select dept_name, avg (salary)
                        from instructor
                        group by dept_name");
while (rset.next()) {
     System.out.println(rset.getString("dept_name") + " " +
                            rset.getFloat(2));
}
```

JDBC Code Details

- Getting result fields:
  - **rs.getString("dept_name") and rs.getString(1) equivalent if dept_name is the first argument of select result.**

- Dealing with Null values
  - **int a = rs.getInt("a");**

    if (rs.wasNull()) Systems.out.println("Got null value");

Prepared Statement

- PreparedStatement pStmt = conn.prepareStatement(
  ```
                                        "insert into instructor values(?,?,?,?)");
  pStmt.setString(1, "88877");      pStmt.setString(2, "Perry");
  pStmt.setString(3, "Finance");   pStmt.setInt(4, 125000);
  pStmt.executeUpdate();
  pStmt.setString(1, "88878");
  pStmt.executeUpdate();
  ```
- For queries, use pStmt.executeQuery(), which returns a ResultSet
- WARNING: always use prepared statements **when taking an input from the user** and adding it to a query
  - **NEVER create a query by concatenating strings which you get as inputs**
  - "insert into instructor values(' " + ID + " ', ' " + name + " ', " +
    " ' + dept_name + " ', " ' balance + ")"
  - What if name is "D'Souza"?

SQL Injection

- Suppose query is constructed using
  - "select * from instructor where name = ’" + name + "’"
- Suppose the user, instead of entering a name, enters:
  - X’ or ’Y’ = ’Y
- then the resulting statement becomes:
  - "select * from instructor where name = ’" + "X’ or ’Y’ = ’Y" + "’"
  - which is:
    - select * from instructor where name = ’X’ or **’Y’ = ’Y’**
  - User could have even used
    - X’; update instructor set salary = salary + 10000; --
- Prepared statement internally uses:
  - **Always use prepared statements, with user inputs as parameters**

Example JDBC

SelectSimple.java

UpdateSimple.jav
a

SelectPrepared.jav
a