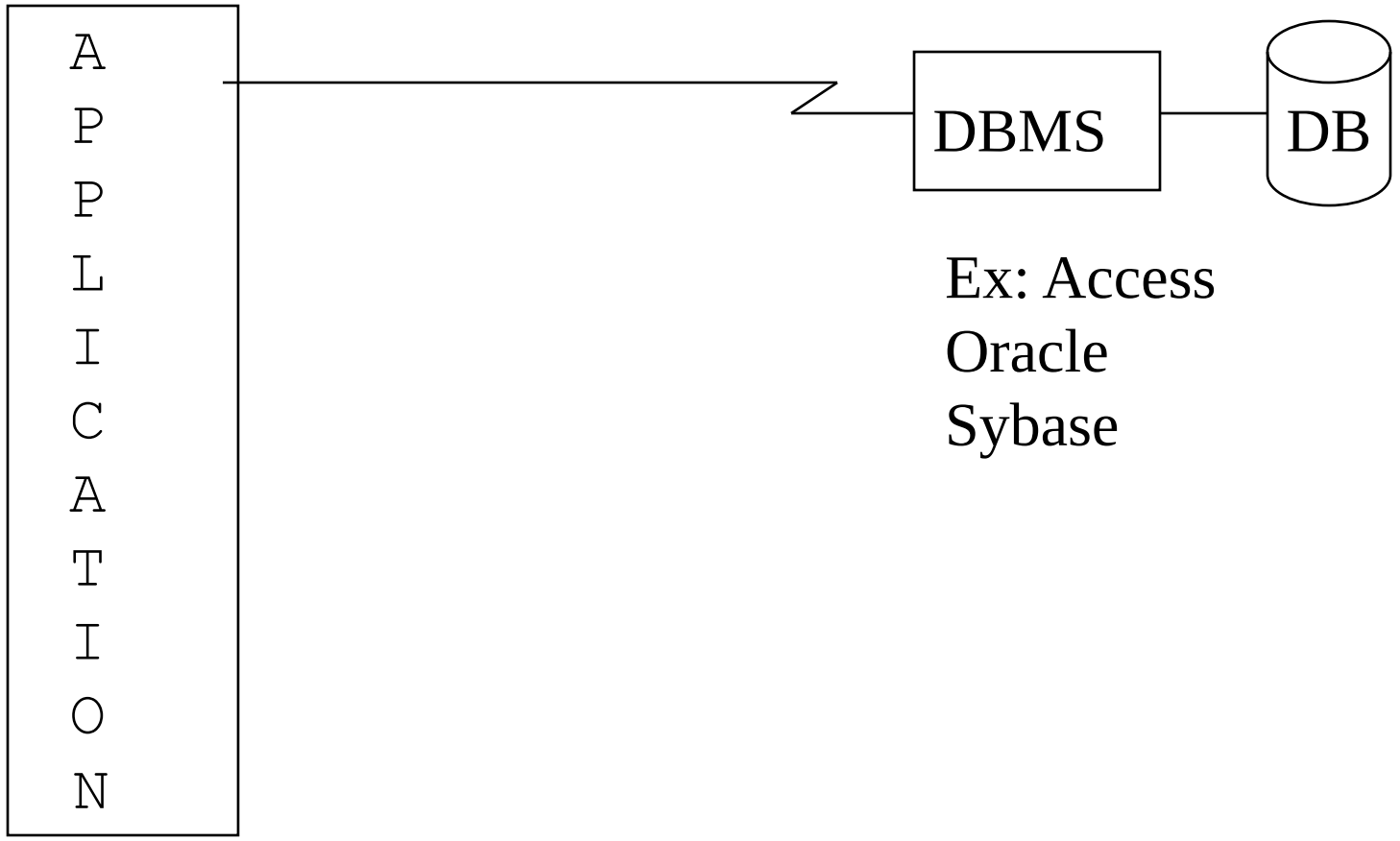
JDBC

By:
Dr. Deepak Kumar Sharma
Asst. Professor (SG)

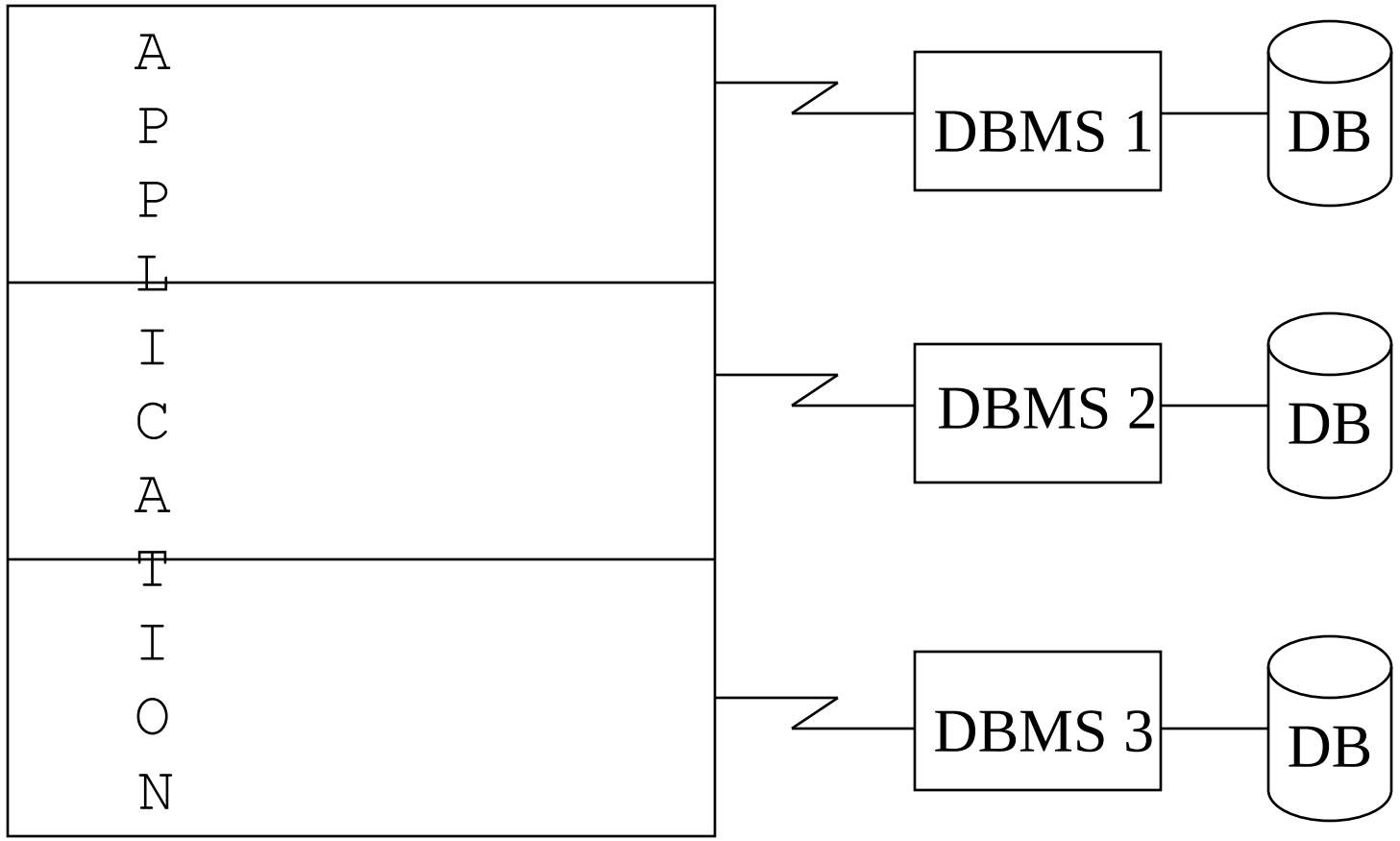
Introduction

- Most popular form of database system is the relational database system.
- Examples: MS Access, Sybase, Oracle, MySQL.
- Structured Query Language (SQL) is used among relational databases to construct queries.

Simple Database Application



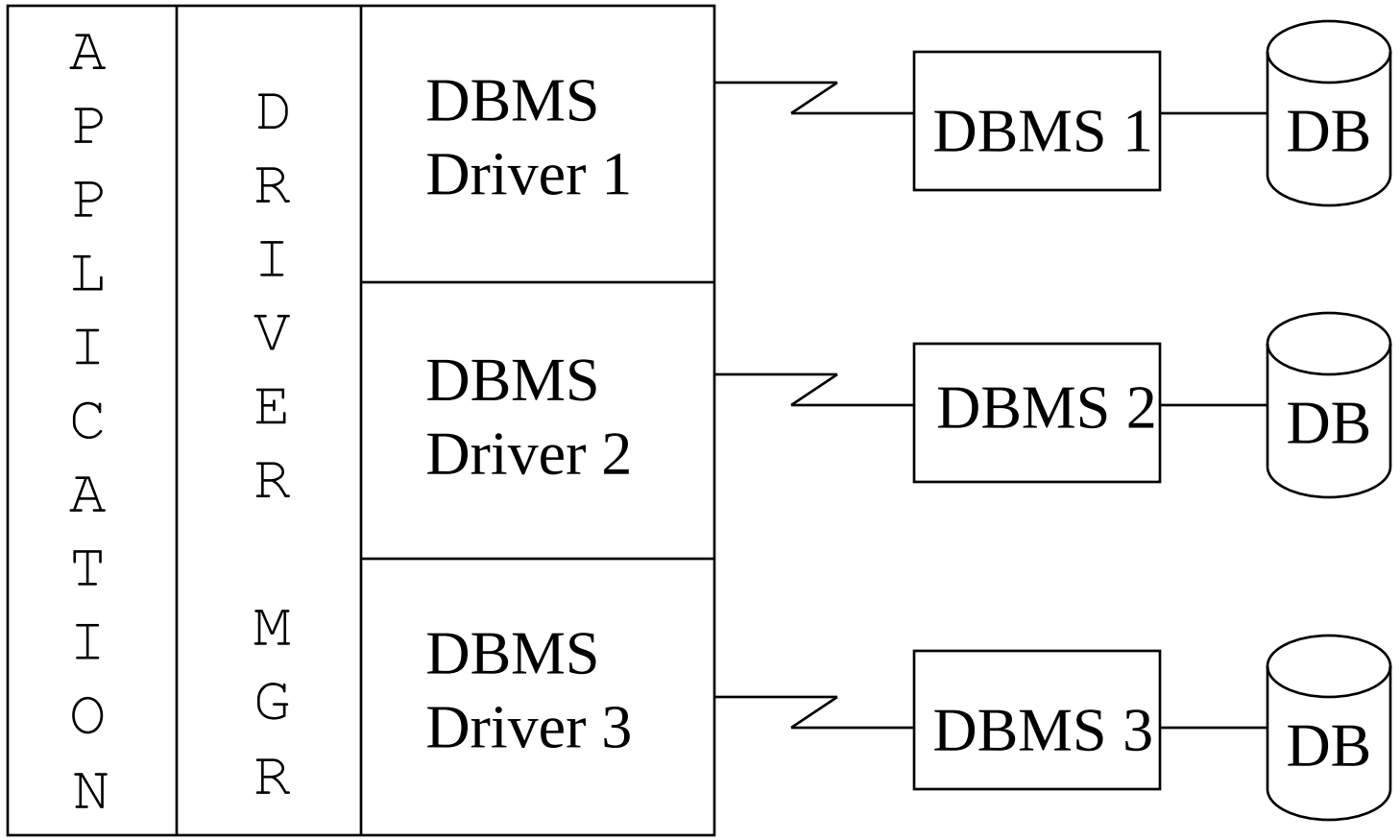
Multi-Databases



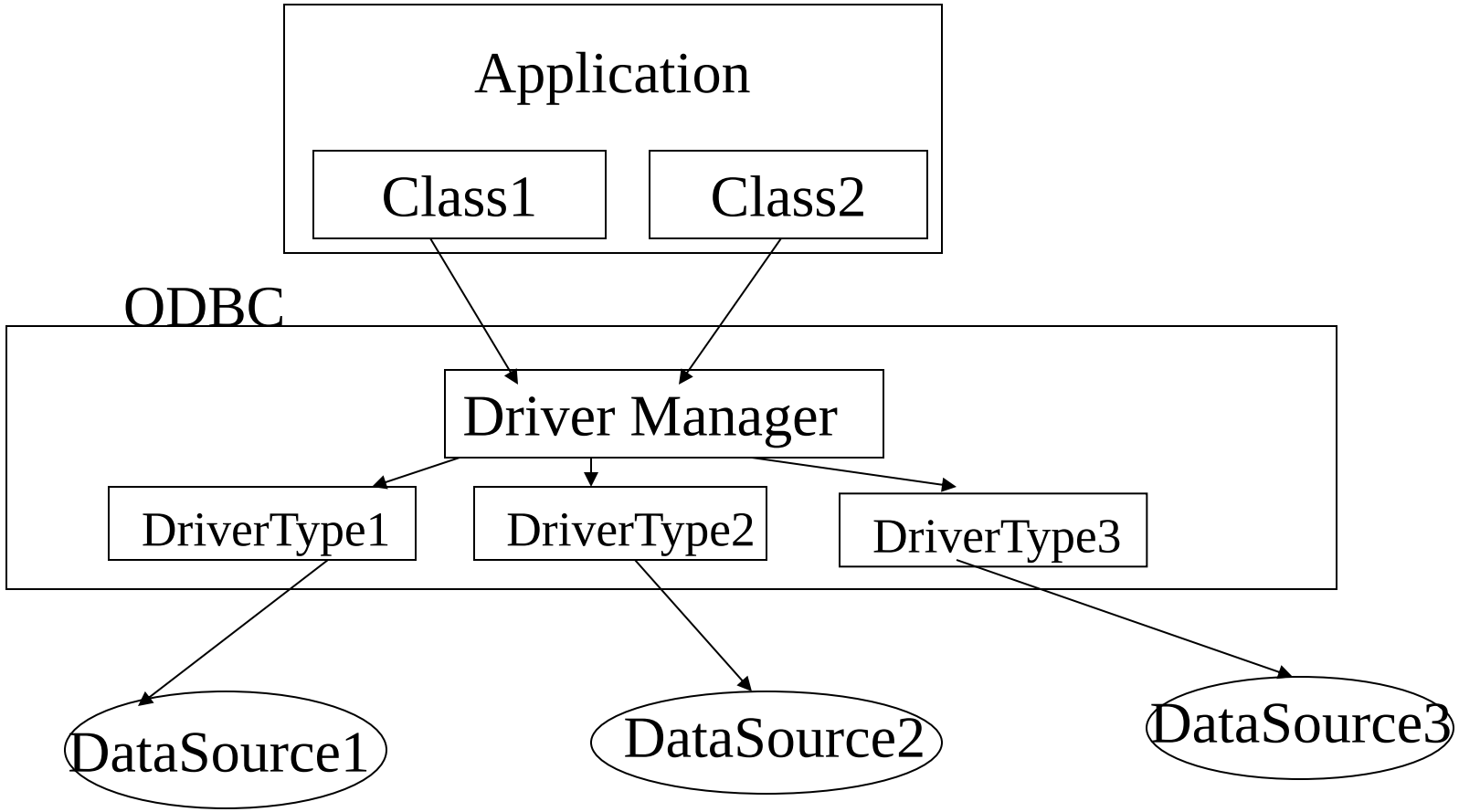
Open Database Connectivity (ODBC) Standard

- ODBC standard is an interface by which application programs can access and process SQL databases in a DBMS-independent manner. It contains:
- A **Data Source** that is the database, its associated DBMS, operating system and network platform
- A **DBMS Driver** that is supplied by the DBMS vendor or independent software companies
- A **Driver Manager** that is supplied by the vendor of the O/S platform where the application is running

Standard Access to DB



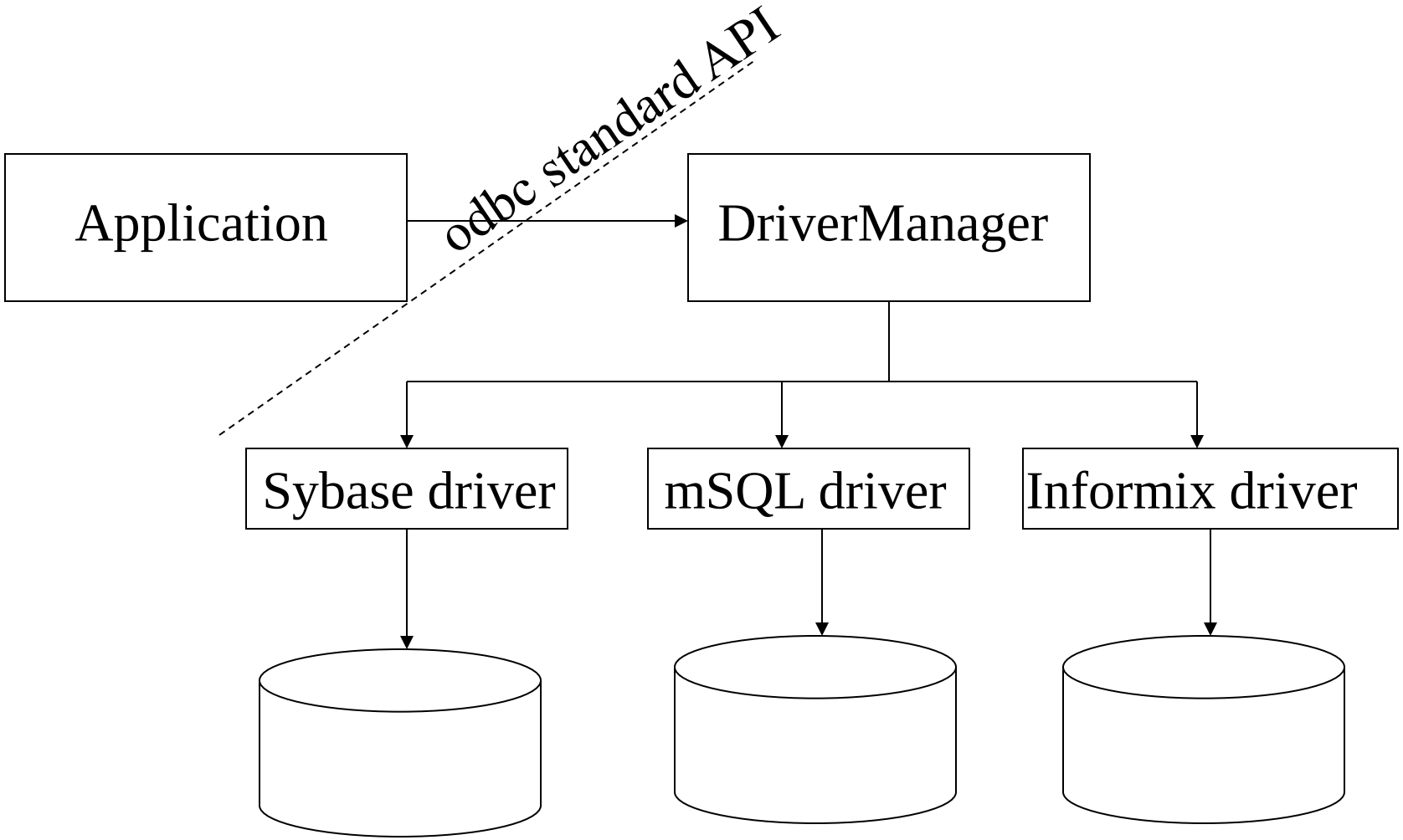
ODBC Architecture



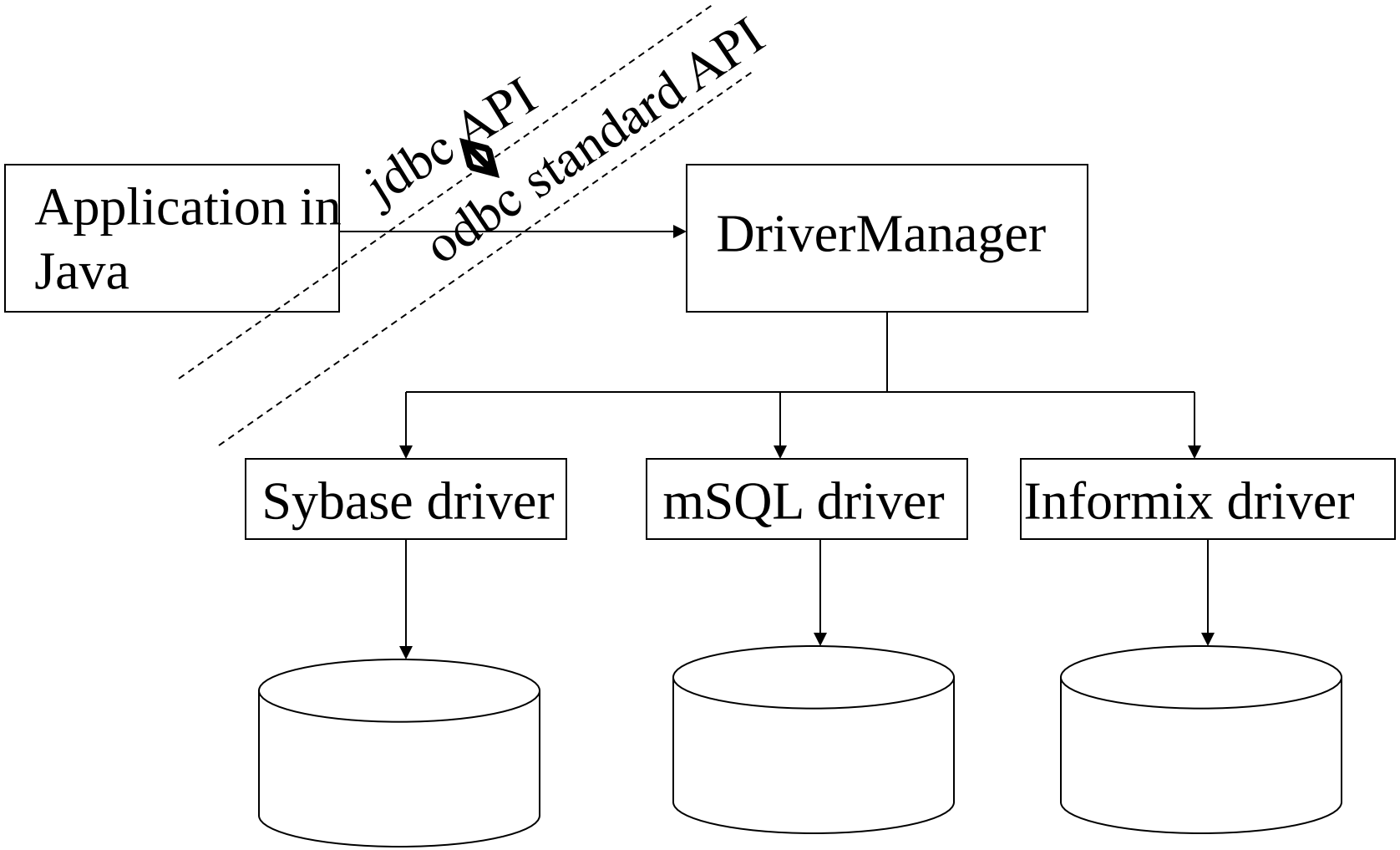
ODBC Interface

- It is a system independent interface to database environment that requires an ODBC driver to be provided for each database system from which you want to manipulate data.
- The database driver bridges the differences between your underlying system calls and the ODBC interface functionality.

An Example


odbc standard API
Application
DriverManager
Sybase driver
mSQL driver
Informix driver

Application in Java



Java Support for ODBC : JDBC

- When applications written in Java want to access data sources, they use classes and associated methods provided by Java DBC (JDBC) API.
- JDBC is specified as an “interface”.
- An interface in Java can have many “implementations”.
- So it provides a convenient way to realize many “drivers”

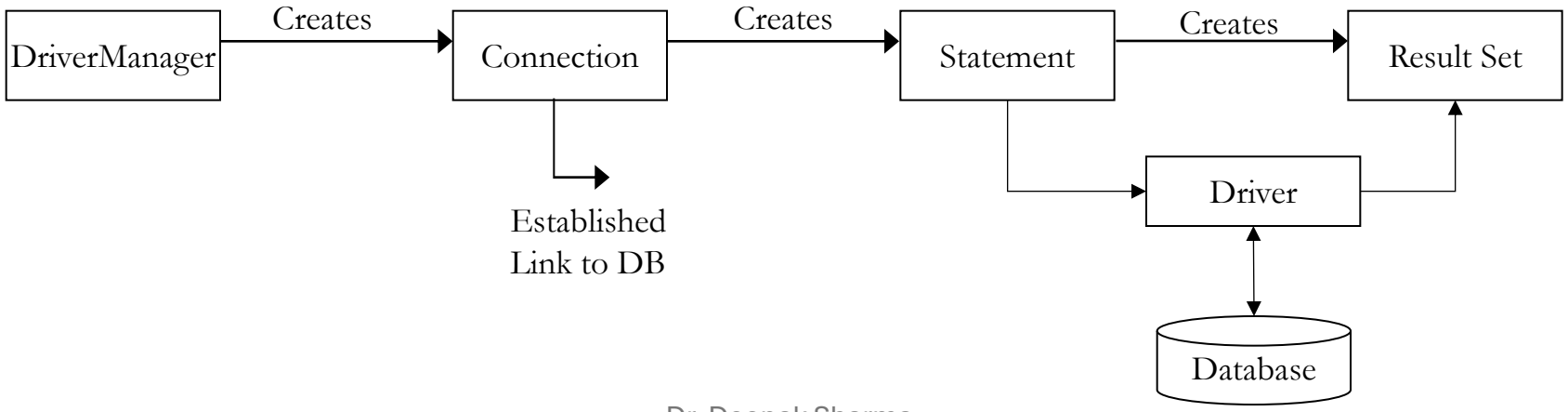
Data Source and Driver

- Data source is the data base created using any of the common database applications available.
- Your system should have the driver for the database you will be using.
- For example your Windows system should have the MS Access Driver.
- There are a number of JDBC drivers available.
  - Information on installing them is available at :
    - http://industry.java.sun.com/products/jdbc/drivers

JDBC

Conceptual Components

- **Driver Manager:** Loads database drivers and manages connections between the application and the driver
- **Driver:** Translates API calls into operations for specific database
- **Connection:** Session between application and data source
- **Statement:** SQL statement to perform query or update
- **Metadata:** Information about returned data, database, & driver
- **Result Set:** Logical set of columns and rows of data returned by executing a statement



JDBC Classes

Java supports DB facilities by providing classes and interfaces for its components

- **DriverManager**
- **Connection**
- **Statement**
- **ResultSet**

Driver Manager Class

- Provides static, “factory” methods for creating objects implementing the **connection** interface.
  - Factory methods create objects on demand
- It contains all the appropriate methods to register and deregister the database driver class and to create a connection between a Java application and the database.

Connection interface

- Connection class represents a session with a specific data source.
- Connection object establishes connection to a data source, allocates statement objects, which define and execute SQL statements.
- Connection can also get info (metadata) about the data source.

Example:

```
Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/mydb","root","root");
```

Statement interface

- Statement interface is implemented by the connection object.
- Statement object provides the workspace for SQL query, executing it, and retrieving returned data.
- SELECT {what} FROM {table name} WHERE {criteria} ORDER BY {field}
- Queries are embedded as strings in a Statement object.
- Types: Statement, PreparedStatement, CallableStatement

Example:

```
Statement stmt=con.createStatement();
String q="delete from student where name='raj' ";
stmt.executeUpdate(q);
```

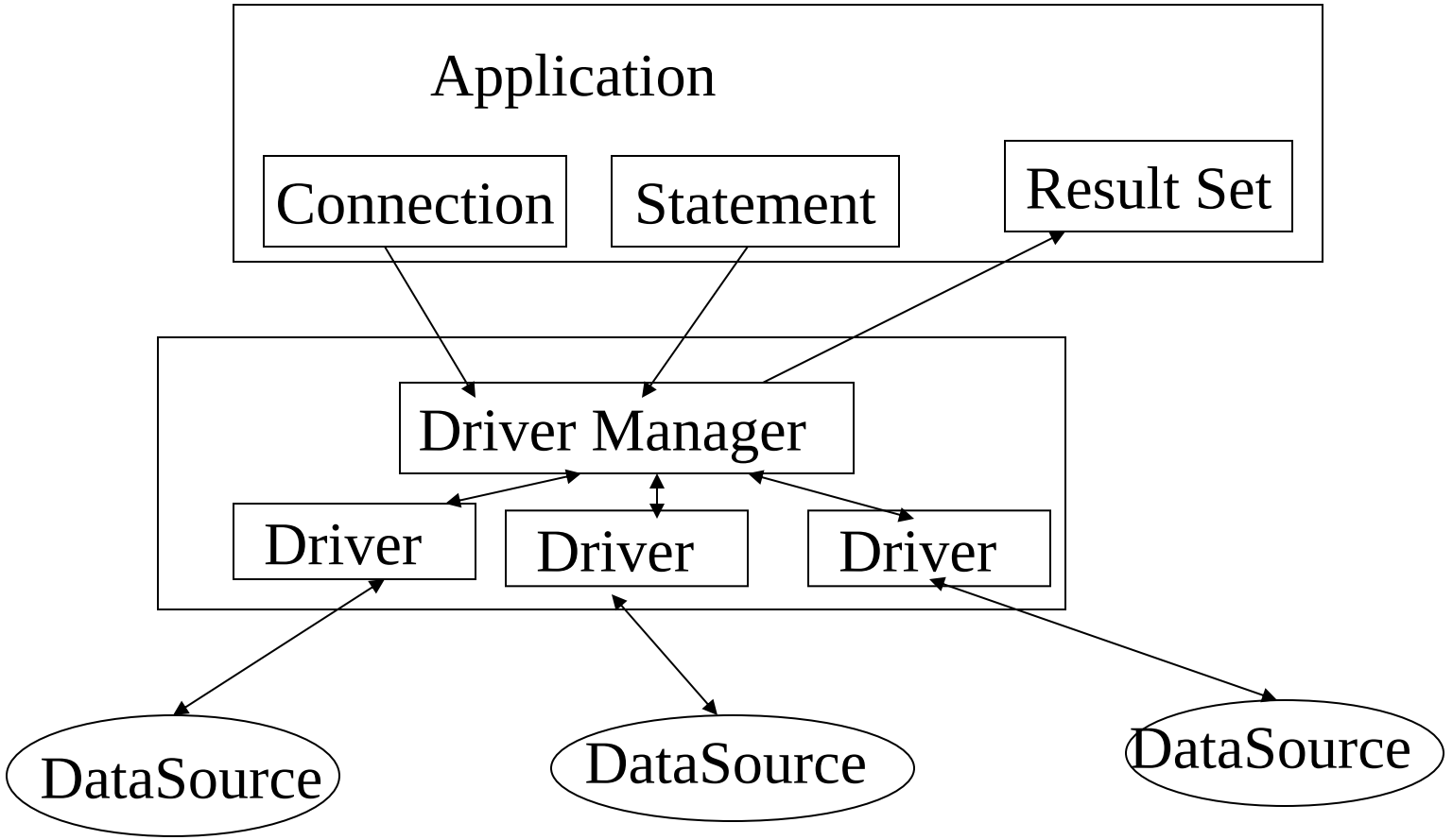
ResultSet interface

- Results are returned in the form of an object implementing the ResultSet interface.
- You may extract individual columns, rows or cell from the ResultSet using the metadata.

Example:

```
String q="Select * from student";

ResultSet rs=stmt.executeQuery(q);
```

JDBC Application Architecture



Prerequisites

Before connecting MySQL with Java, ensure the following:

- MySQL Server is installed and running.
- MySQL Connector/J (JDBC driver for MySQL) is downloaded.
  - (https://dev.mysql.com/downloads/connector/j/)
- Java Development Kit (JDK) is installed.
- A database and table exist in MySQL (or you can create them via Java).

JDBC Programming Steps

- Import necessary packages; Ex: import java.sql.*;
- Load JDBC driver(driver should have been installed)
  - Download the latest MySQL JDBC driver e.g., mysql-connector-java-8.0.30.jar
  - For IDE: Can be added directly by adding connector (.jar) in java build path.
  - Manual Addition (Using IDE)
    - In Eclipse/IntelliJ, right-click the project → Build Path → Configure Build Path → Add External JARs → Select the downloaded .jar file.
- Establish a Connection

  Use java.sql package classes to connect to MySQL
- Data source and its location should have been registered.
- Allocate Connection object, Statement object and ResultSet object
- Execute query using Statement object
- Retrieve data from ResultSet object
- Close Connection object.

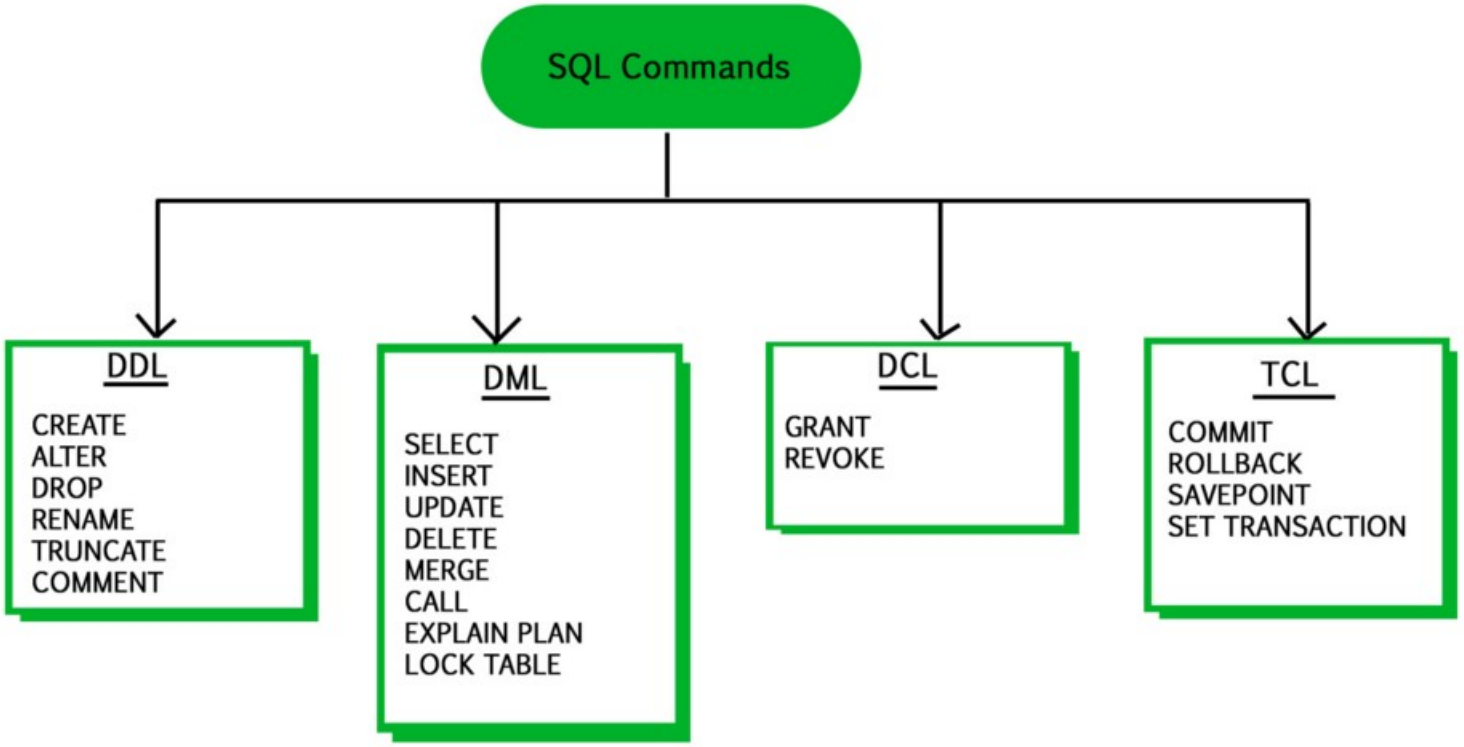
Executing Queries

Methods

- Two primary methods in statement interface used for executing Queries
  - executeQuery : Used to retrieve data from a database
  - executeUpdate: Used for creating, updating & deleting data
- executeQuery used to retrieve data from database
  - Primarily uses Select commands
  - It retuns ResultSet Object
- executeUpdate used for creating, updating & deleting data
  - SQL should contain Update, Insert or Delete commands
  - It returns no of rows affected

Query Type	Recommended Method	Example
SELECT (returns data)	stmt.executeQuery(sql)	ResultSet rs = stmt.executeQuery("SELECT * FROM users");
INSERT/UPDATE/DELETE (modifies data)	stmt.executeUpdate(sql)	int rowsAffected = stmt.executeUpdate("INSERT INTO users VALUES (...)")

SQL Commands Overview



Executing Queries

Data Definition Language (DDL)

- Data definition language queries use executeUpdate
- Syntax: int executeUpdate(String sqlString) throws SQLException
  - It returns an integer which is the number of rows updated
  - sqlString should be a valid String else an exception is thrown
- Example 1: Create a new table

```
Statement statement = connection.createStatement();
String sqlString =
"Create Table Catalog"
+ "(Title Varchar(256) Primary Key Not Null,"+
+ "LeadActor Varchar(256) Not Null, LeadActress Varchar(256) Not Null,"
+ "Type Varchar(20) Not Null, ReleaseDate Date Not NULL )";
Statement.executeUpdate(sqlString);
```

  - executeUpdate returns a zero since no row is updated

Executing Queries

DDL (Example)

- Example 2: Update table

```
Statement statement = connection.createStatement();
String sqlString =
 "Insert into Catalog"
+ "(Title, LeadActor, LeadActress, Type, ReleaseDate)"
+ "Values('Gone With The Wind', 'Clark Gable', 'Vivien Liegh',"
+ "'Romantic', '02/18/2003' "
Statement.executeUpdate(sqlString);
```

  - executeUpdate returns a 1 since one row is added

Executing Queries

Data Manipulation Language (DML)

- Data definition language queries use executeQuery
- Syntax

  ResultSet executeQuery(String sqlString) throws SQLException

  - It returns a ResultSet object which contains the results of the Query
- Example 1: Query a table

```
Statement statement = connection.createStatement();
String sqlString = "Select Catalog.Title, Catalog.LeadActor, Catalog.LeadActress," +
                   "Catalog.Type, Catalog.ReleaseDate From Catalog";
ResultSet rs = statement.executeQuery(sqlString);
```

ResultSet

Definition

- ResultSet contains the results of the database query that are returned
- Allows the program to scroll through each row and read all columns of data
- ResultSet provides various access methods that take a column index or column name and returns the data
  - All methods may not be applicable to all resultsets depending on the method of creation of the statement.
- When the executeQuery method returns the ResultSet the cursor is placed before the first row of the data
  - Cursor refers to the set of rows returned by a query and is positioned on the row that is being accessed
  - To move the cursor to the first row of data next() method is invoked on the resultset
  - If the next row has a data the next() results true else it returns false and the cursor moves beyond the end of the data
- First column has index 1, not 0

Example: JDBC MySQL Connectivity

```
Steps to follow:

//add mysql connector (See Next slide)

1. create connection
2. create statement/Query
3. Execute statement/Query
4. Store the results in resultset (optional)
5. close connection

Install MySQL and create database and a table

//create database mydb
//use mydb
//create table Student( sapid int(10), name varchar(30), cgpa
decimal(5,2))
```

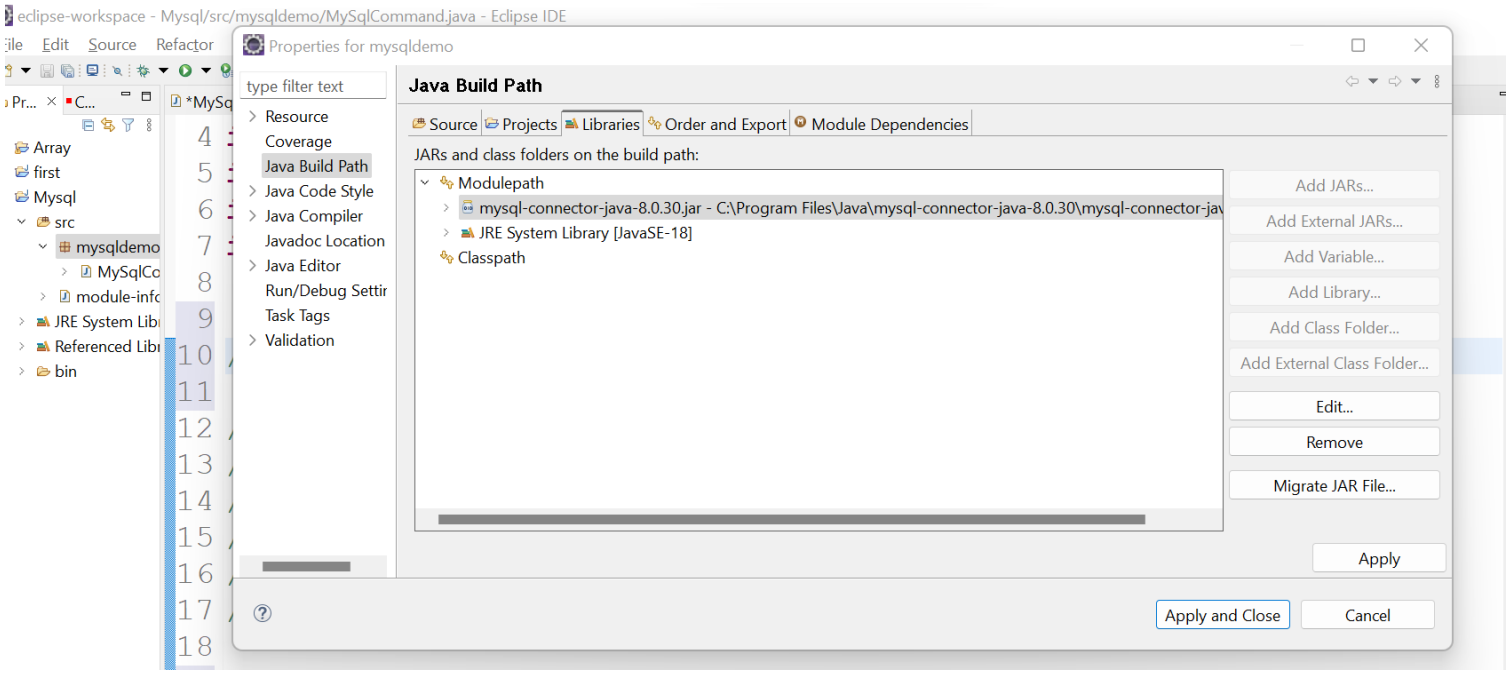
Load Java DriverManager class

```
Class.forName(" com.mysql.jdbc.driver ");
```

OR

Steps in Eclipse:

- Add MySql connector Jar file (mysql-connector-java-8.0.30)
  - Right click in your project->properties
  - Select Built Path
  - Then select Add External Jar



Example-1

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class MySqlCommand {
public static void main(String[] args) throws SQLException
{

//1. create connection
Connection
con=DriverManager.getConnection("jdbc:mysql://localhost:3306/mydb","root","root");

//2. create statement/Query
Statement stmt=con.createStatement();
String q="insert into student values(50002561,'Ranjit',7.5)";

//String q="update student set cgpa=8.9 where name=`Ranjit'";
//String q="delete from student where name='raj'";

//3. Execute statement/Query
stmt.executeUpdate(q);
con.close();
System.out.println("Query Executed");

}
}
```

O/P:
Query Executed

Example-2

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class MySqlCommand {
public static void main(String[] args) throws SQLException
{
//1. create connection
Connection
con=DriverManager.getConnection("jdbc:mysql://localhost:3306/mydb","root","root");
//2. create statement/Query
Statement stmt=con.createStatement();
//String q="insert into student values(50002561,'Ranjit',7.5)";

//4. store the results in ResultSet
String q="Select * from student";
ResultSet rs=stmt.executeQuery(q);
while(rs.next())
{
/*int sapid=rs.getInt("sapid");
String name=rs.getString("name");
float cgpa= rs.getFloat("cgpa");*/
System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getFloat(3));
}
con.close();
System.out.println("Query Executed");
}
}
```

50001364	Aman	8.0
50001366	Rahul	9.0
50002561	Ranjit	7.5

Summary of important SQL Commands

- SELECT - extracts data from a database
- UPDATE - updates data in a database
- DELETE - deletes data from a database
- INSERT INTO - inserts new data into a database
- CREATE DATABASE - creates a new database
- ALTER DATABASE - modifies a database
- CREATE TABLE - creates a new table
- ALTER TABLE - modifies a table

Prepared Statement

- The PreparedStatement interface is a subinterface of Statement.
- It is used to execute parameterized query.

Example: String sql="insert into emp values(?,?,?)";

- we are passing parameter (?) for the values.
- Its value will be set by calling the setter methods of PreparedStatement.

Why?

Improves performance: The performance of the application will be faster if you use PreparedStatement interface because query is compiled only once.

PreparedStatement

To create instance:

```
public PreparedStatement prepareStatement(String query)throws SQLException{}
```

Methods of PreparedStatement interface

Method	Description
public void **setInt**(int paramIndex, int value)	sets the integer value to the given parameter index.
public void **setString**(int paramIndex, String value)	sets the String value to the given parameter index.
public void **setFloat**(int paramIndex, float value)	sets the float value to the given parameter index.
public void **setDouble**(int paramIndex, double value)	sets the double value to the given parameter index.
public int executeUpdate()	executes the query. It is used for create, drop, insert, update, delete etc.
public ResultSet executeQuery()	executes the select query. It returns an instance of ResultSet.

PreparedStatement

How to execute the query using PreparedStatement

We need to follow 4 steps:

- Load driver & create an instance of Connection.
- Create PreparedStatement instance & SQL Query.
- Replace parameters with dynamic values.
- Execute query.

Example of PreparedStatement interface that inserts the record

Assume Table: create table emp(id number(10),name varchar2(50));

```
.....
Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/mydb","root","root");

PreparedStatement stmt=con.prepareStatement("insert into Emp values(?,?)");

stmt.setInt(1,101);//1 specifies the first parameter in the query
stmt.setString(2,"Ratan");

int i=stmt.executeUpdate();

System.out.println(i+" records inserted");

con.close();
}
}
```

Example- UPDATE using PreparedStatemt

```
package PreparedStmt;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class PreparedStmt {
public static void main(String[] args) throws SQLException {
Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/mydb","root","root");

//Create PreparedStatement instance & SQL Query
String sql = "UPDATE STUDENT SET NAME=?,sapid=? where cgpa>=?";
PreparedStatement pstmt = con.prepareStatement(sql);

//Replace paramenters with dynamic values
pstmt.setString(1, "TARUN");
pstmt.setInt(2, 50001234);
pstmt.setDouble(3, 8.0);

//Execute query
pstmt.executeUpdate();
```

```
50001234  TARUN  8.0
50001234  TARUN  9.0
50002561  Ranjit  7.5
Query Executed
```

```
PreparedStatement pstmt1 = con.prepareStatement("Select * from student");
ResultSet rs=pstmt1.executeQuery();
while(rs.next())
{
System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getFloat(3));
}
con.close();
System.out.println("Query Executed");
}}
```

Example of PreparedStatement interface that deletes the record

```
……..……
…..
PreparedStatement stmt=con.prepareStatement("delete from student where sapid=?");

stmt.setInt(1,  50001234);

int i=stmt.executeUpdate();

System.out.println(i+" records deleted");
…..
…..
```

Example of PreparedStatement interface that retrieve the records of a table

……..

PreparedStatement pstmt1 = con.prepareStatement("Select * from student");

ResultSet rs=pstmt1.executeQuery();

```
while(rs.next())
{
System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  " +
rs.getFloat(3));
  }
con.close();
```

….

Complete Code using Statement

```
package mysqldemo;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

//add mysql connector

//insert,update,delete
//1. create connection
//2. create statement/Query
//3. Execute statement/Query
//4. Store the results in resultset
//5. close connection

//create database mydb
//use mydb
//create table Student( sapid int(10),
name varchar(30), cgpa decimal(5))
```

```
public class MySqlCommand {
        public static void main(String[] args) throws SQLException
        {
        //1. create connection
        Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/mydb","root","root");

        //2. create statement/Query
        Statement stmt=con.createStatement();
        //String q="insert into student values(50002561,'Ranjit',7.5)";
        //String q="update student set cgpa=8.9 where name='raj'";
        //String q="delete from student where name='raj'";

        //3. Execute statement/Query
        //stmt.execute(q);

        //4. store the results in ResultSet
        String q="Select * from student";
        ResultSet rs=stmt.executeQuery(q);
        while(rs.next())
        {
                /*int sapid=rs.getInt("sapid");
                String name=rs.getString("name");
                float cgpa= rs.getFloat("cgpa");*/
                System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getFloat(3));
        }
        con.close();
        System.out.println("Query Executed");
        }
}
```

Complete Code using PreparedStatement

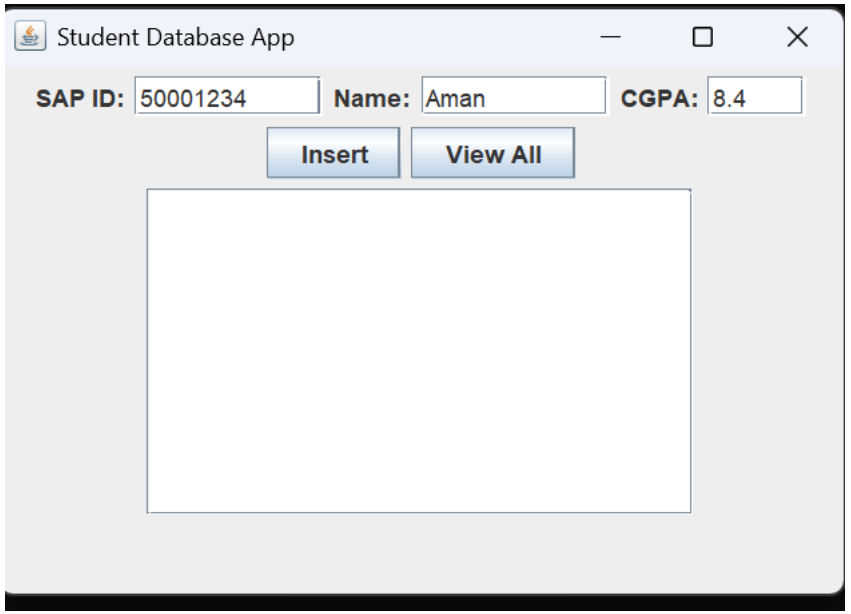
```
import java.sql.*;

public class MySqlCommand {
   public static void main(String[] args) throws SQLException {
      // 1. Create connection
      Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/mydb", "root", "root");

      // Example 1: Insert using PreparedStatement
      /*
      String insertQuery = "INSERT INTO student (sapid, name, cgpa) VALUES (?, ?, ?)";
      PreparedStatement pstmt = con.prepareStatement(insertQuery);
      pstmt.setInt(1, 50002561);
      pstmt.setString(2, "Ranjit");
      pstmt.setFloat(3, 7.5f);
      pstmt.executeUpdate();
      */

      // Example 2: Update using PreparedStatement
      /*
      String updateQuery = "UPDATE student SET cgpa = ? WHERE name = ?";
      PreparedStatement pstmt = con.prepareStatement(updateQuery);
      pstmt.setFloat(1, 8.9f);
      pstmt.setString(2, "raj");
      pstmt.executeUpdate();
      */  //Continue.....
```

Complete Code using PreparedStatement Cont..

```
// Example 3: Delete using PreparedStatement
    /*
    String deleteQuery = "DELETE FROM student WHERE name = ?";
    PreparedStatement pstmt = con.prepareStatement(deleteQuery);
    pstmt.setString(1, "raj");
    pstmt.executeUpdate();
    */

    // Example 4: Select using PreparedStatement
    String selectQuery = "SELECT * FROM student";
    PreparedStatement pstmt = con.prepareStatement(selectQuery);
    ResultSet rs = pstmt.executeQuery();

    // 4. Process the result
    while (rs.next()) {
       System.out.println(rs.getInt(1) + "  " + rs.getString(2) + "  " + rs.getFloat(3));
    }

    // 5. Close connection
    con.close();
    System.out.println("Query Executed");
  }
}
```

Swing application connected with MySQL

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.sql.*;

public class StudentApp extends JFrame {
    JTextField sapidField, nameField, cgpaField;
    JTextArea outputArea;
    JButton insertButton, viewButton;

    Connection conn;

    public StudentApp() {
        setTitle("Student Database App");
        setLayout(new FlowLayout());

        // Input Fields
        add(new JLabel("SAP ID:"));
        sapidField = new JTextField(10);
        add(sapidField);

        add(new JLabel("Name:"));
        nameField = new JTextField(10);
        add(nameField);

        add(new JLabel("CGPA:"));
        cgpaField = new JTextField(5);
        add(cgpaField);

        // Buttons
        insertButton = new JButton("Insert");
        viewButton = new JButton("View All");
        add(insertButton);
        add(viewButton);

        // Output Area
        outputArea = new JTextArea(10, 30);
        outputArea.setEditable(false);
        add(new JScrollPane(outputArea));

        // Button Actions
        insertButton.addActionListener(e -> insertRecord());
        viewButton.addActionListener(e -> viewRecords());

        // Database Connection
        connectToDatabase();

        setSize(400, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }
```



Swing application connected with MySQL Cont..

```
void connectToDatabase() {
    try {
        conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/mydb", "root", "root");
        outputArea.setText("Connected to database.\n");
    } catch (SQLException e) {
        outputArea.setText("Failed to connect to database.\n" + e.getMessage());
    }
}

void insertRecord() {
    try {
        String query = "INSERT INTO Student (sapid, name, cgpa) VALUES (?, ?, ?)";
        PreparedStatement pst = conn.prepareStatement(query);
        pst.setInt(1, Integer.parseInt(sapidField.getText()));
        pst.setString(2, nameField.getText());
        pst.setDouble(3, Double.parseDouble(cgpaField.getText()));
        pst.executeUpdate();
        outputArea.append("Record inserted successfully.\n");
    } catch (Exception e) {
        outputArea.append("Error inserting record: " + e.getMessage() + "\n");
    }
}
```

```
void viewRecords() {
        try {
            String query = "SELECT * FROM Student";
            PreparedStatement pst = conn.prepareStatement(query);
            ResultSet rs = pst.executeQuery();

            outputArea.setText("SAP ID\tName\tCGPA\n");
            while (rs.next()) {
                outputArea.append(
                    rs.getInt("sapid") + "\t" +
                    rs.getString("name") + "\t" +
                    rs.getDouble("cgpa") + "\n"
                );
            }
        } catch (Exception e) {
            outputArea.append("Error retrieving records: " + e.getMessage() + "\n");
        }
    }

    public static void main(String[] args) {
        new StudentApp();
    }
}
```