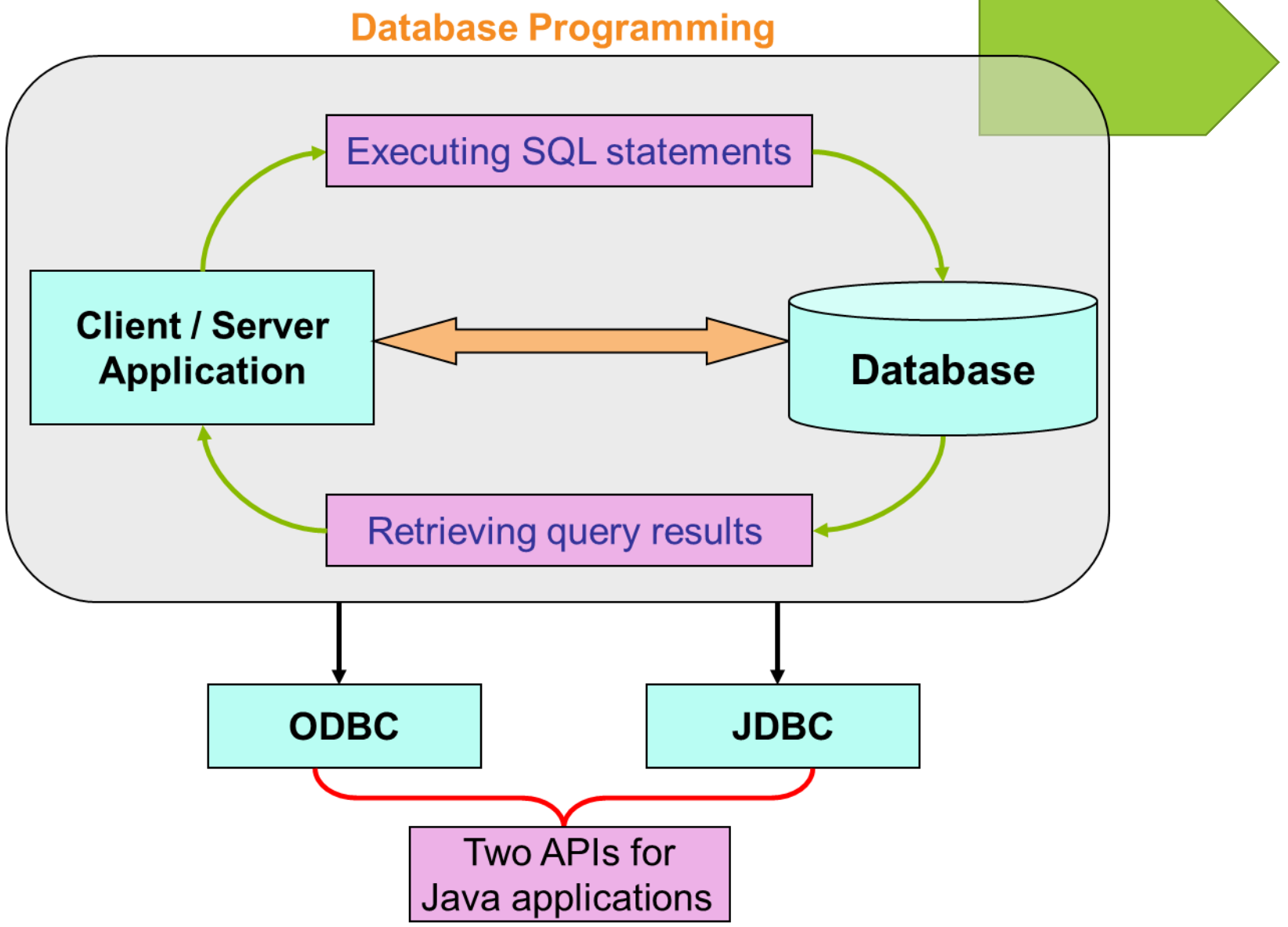
Java.sql package

By: **RAJASHEKAR G.S**
Java Full Stack Trainer

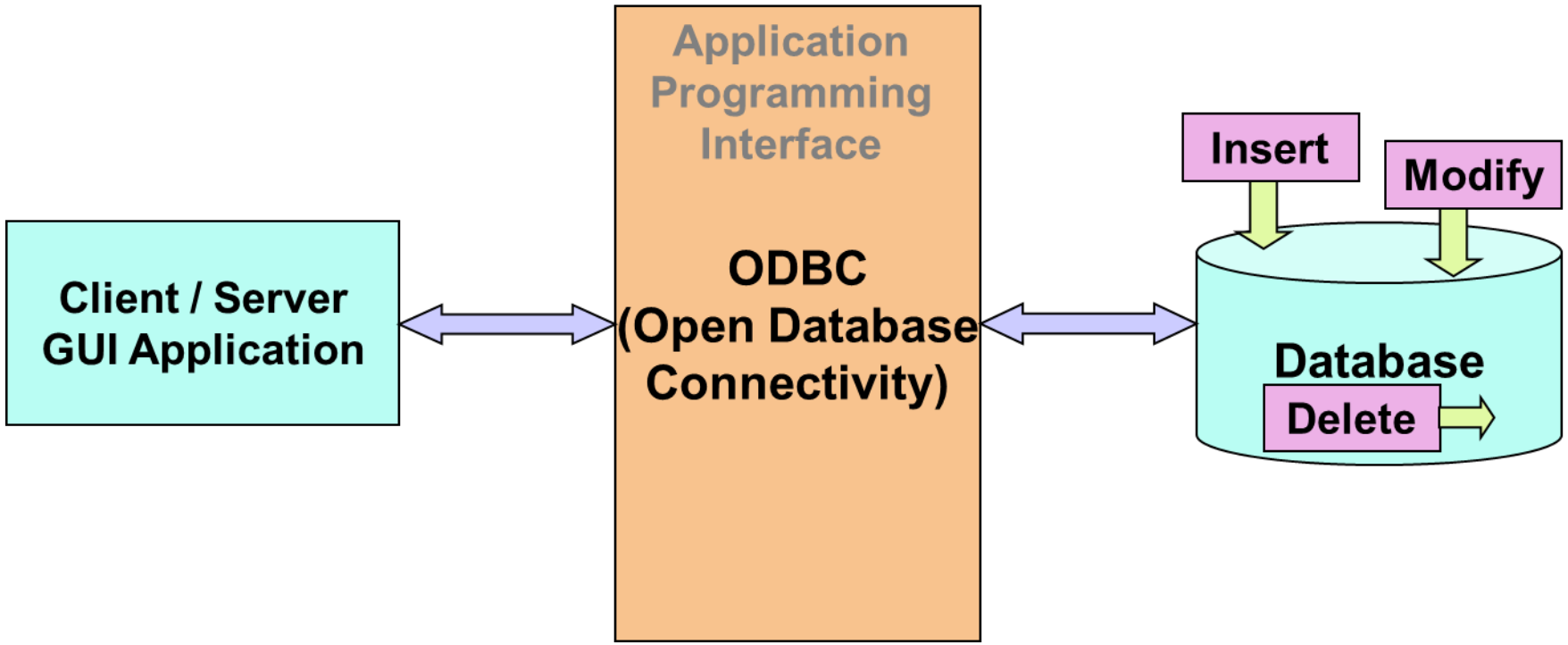
Introduction

- *Discuss Java Database Connectivity (JDBC)*
- *Describe the java.sql package in brief*
- *Discuss types of JDBC drivers*
- *Explain the anatomy of a JDBC program*
- *Describe the PreparedStatement interface*
- *Use ResultSet Interface*

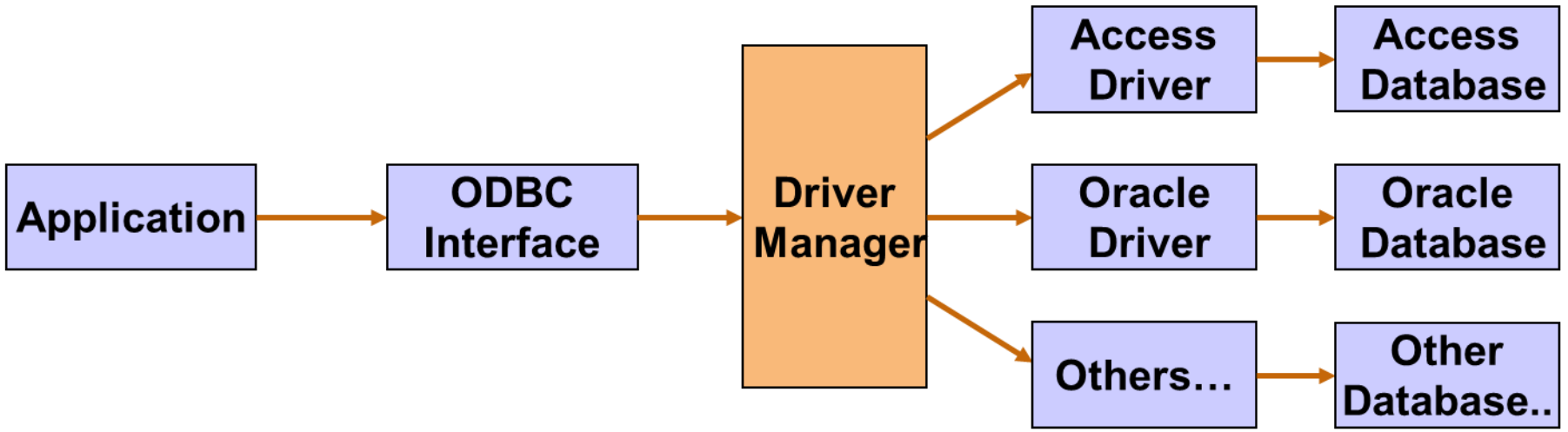
Database


Database Programming
Executing SQL statements
Client / Server Application
Database
Retrieving query results
ODBC
JDBC
Two APIs for Java applications

ODBC 3-1


Application Programming Interface
ODBC (Open Database Connectivity)
Client / Server GUI Application
Insert
Modify
Database
Delete

ODBC 3-2



ODBC Connection

ODBC

- ODBC API was the database API to connect and execute the query with the database prior to JDBC.
- But ODBC API uses ODBC driver which is written in C language i.e. platform dependent and unsecured.
- So Java defined its own API – JDBC API that uses JDBC drivers written in Java Language.

What is API ?

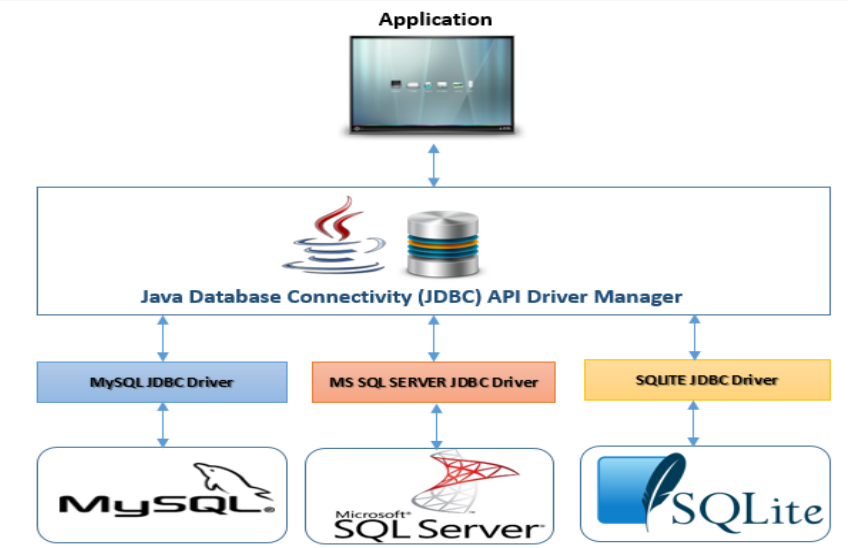
Application Programming Interface is a document that contains description of all the features of a Product or Software.

It represents Classes and Interfaces that Software programs can follow to communicate with each other.

An API can be created for Applications, Libraries, Operating System etc.

JDBC

- JDBC is a Java-based data access technology (Java Standard Edition platform) from Oracle Corporation.
- This technology is an API for the Java programming language that defines how a client may access a database.
- It provides methods for querying and updating data in a database.



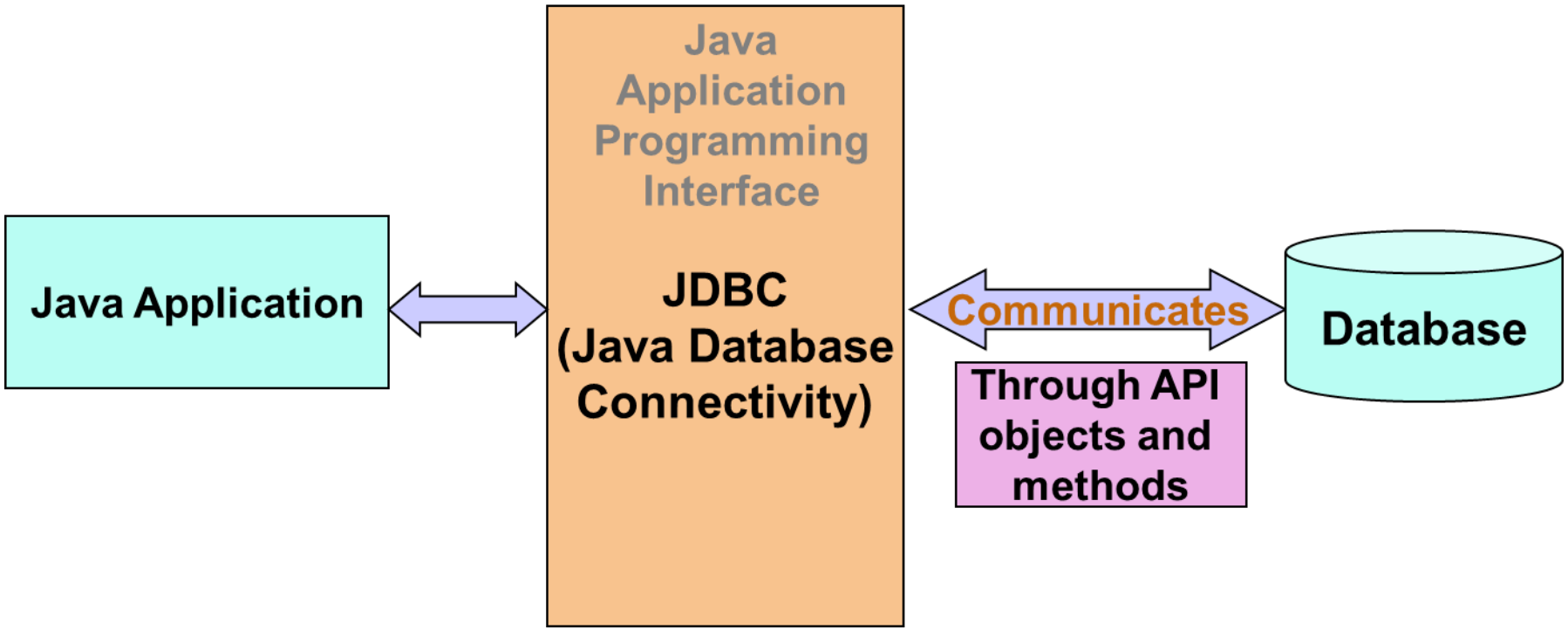
JDBC History

- Before JDBC, ODBC API was used to connect and execute query to the database.
- But ODBC API uses ODBC driver that is written in C language which is platform dependent and unsecured.
- That is why Sun Micro System has defined its own API (JDBC API) that uses JDBC driver written in Java language.
- Sun Microsystems released JDBC as part of JDK 1.1 on February 19, 1997.
- The JDBC classes are contained in the Java package java.sql and javax.sql
- The latest version, JDBC 4.3, and is included in Java SE 17.

API

- The Java API is the set of classes included with the Java Development Environment.
- These classes are written using the Java language and run on the JVM.
- The Java API includes everything from collection classes to GUI classes.
- JDBC is also an API.

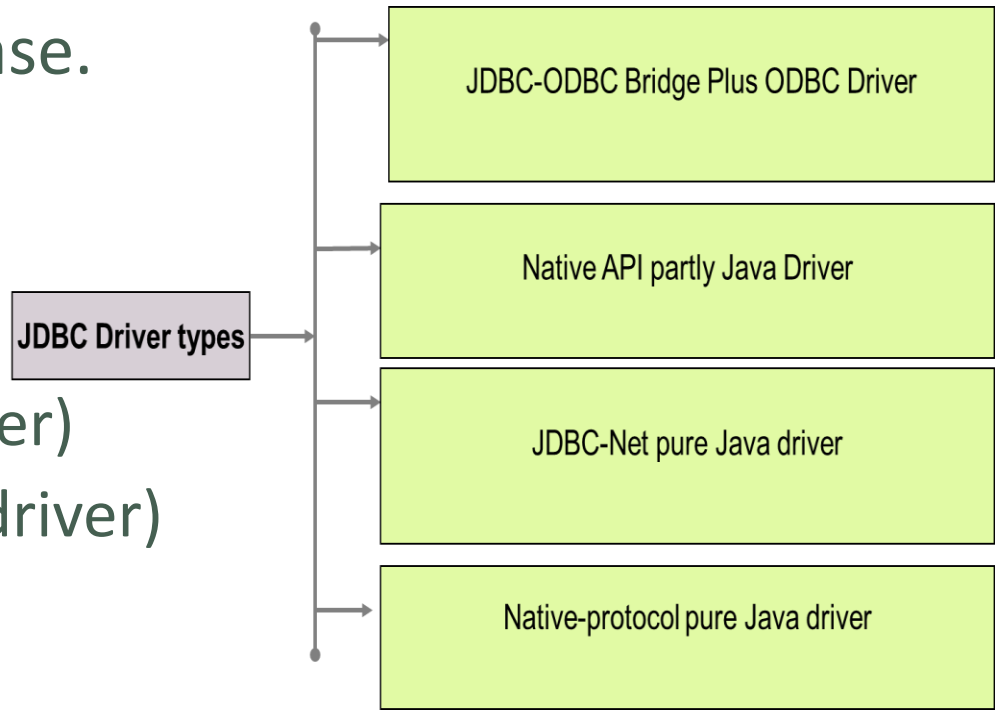
JDBC


Java Application
Java Application Programming Interface
JDBC (Java Database Connectivity)
Communicates
Through API objects and methods
Database

JDBC Driver

- JDBC Driver is a software component that enables java application to interact with the database.

- There are 4 types of JDBC drivers:
  - Type 1: JDBC-ODBC bridge driver
  - Type 2: Native-API driver (partially java driver)
  - Type 3: Network Protocol driver (fully java driver)
  - Type 4: Thin driver (fully java driver)

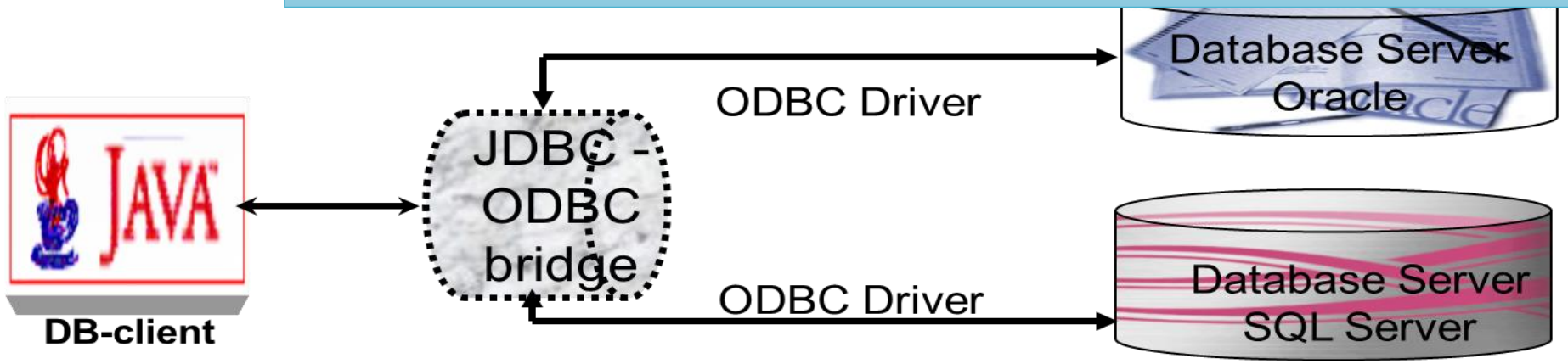


JDBC Driver Types (1)

- JDBC-ODBC Bridge plus ODBC Driver

- ODBC is not readily convertible to Java
- Sun provides a bridge driver to access ODBC data sources from JDBC
- This is called the JDBC-ODBC Bridge plus ODBC driver

JDBC bridge is used to access ODBC drivers installed on each client machine. For example, JDBC-ODBC Bridge driver in JDK 1.2.



JDBC Driver Types (2)

- Native API partly-Java Driver

- JDBC calls are converted into calls on the client API for DBMS
- This driver uses JavaNativeInterface(JNI) that calls the local database APIs
- The Native APIs partly-Java driver calls the Native Database Library that accesses the database
- This driver like the ODBC driver needs binary code on the client machine

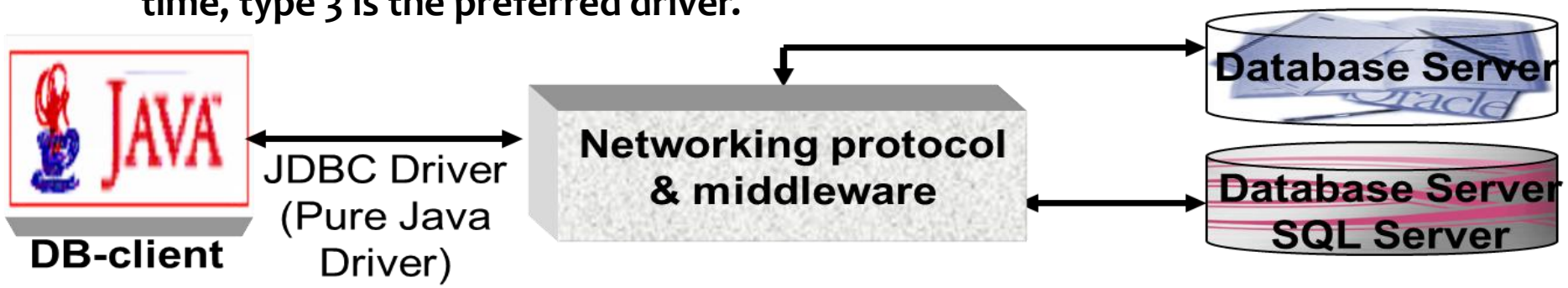
These APIs are vendor specific and vendor provided driver is required to be installed.
Eg:Oracle Call Interface (OCI) driver.

JDBC Driver Types (3)

- JDBC-Net pure Java driver

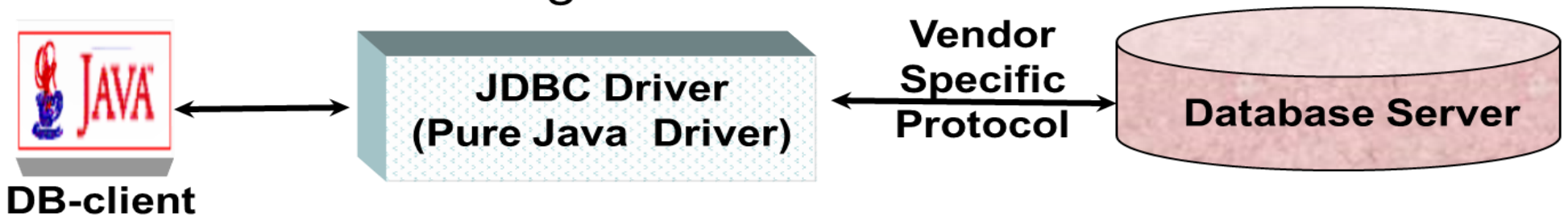
- Uses a networking protocol and middleware to communicate with the server
- Server then translates the messages communicated to DBMS specific function calls
- Specific protocol used depends on the vendor
- No need for client installation
- Allows access to multiple back-end databases

- **If your Java application is accessing multiple types of databases at the same time, type 3 is the preferred driver.**

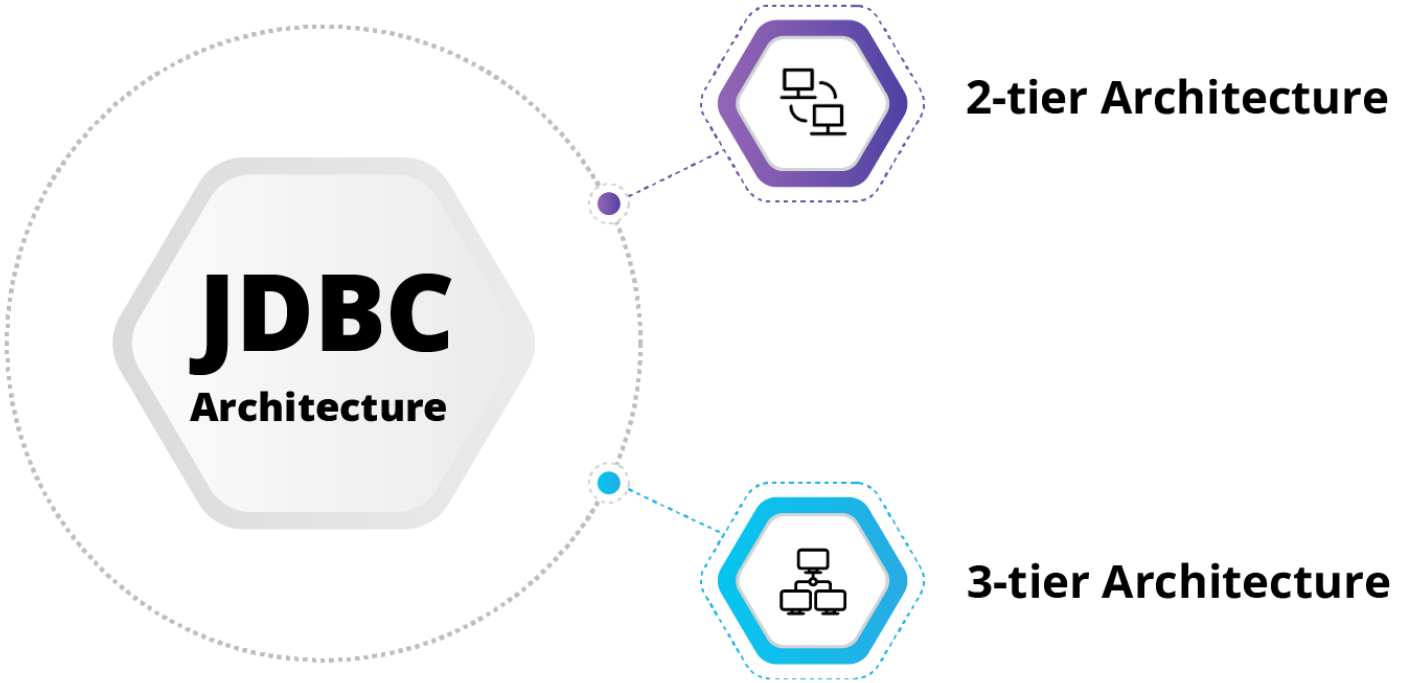


JDBC Driver Types (4)

- **If you are accessing one type of database, such as Oracle, Sybase, or IBM, the preferred driver type is 4.**
  - Native-protocol pure Java driver
    - 100% Java enabled and does not use CLI libraries
    - Capable of communicating directly with the database
    - Converts JDBC calls into network protocols such as TCP/IP and other proprietary protocols used by DBMS directly
    - Since many of these protocols are proprietary, the database vendors themselves will be the primary source of usage

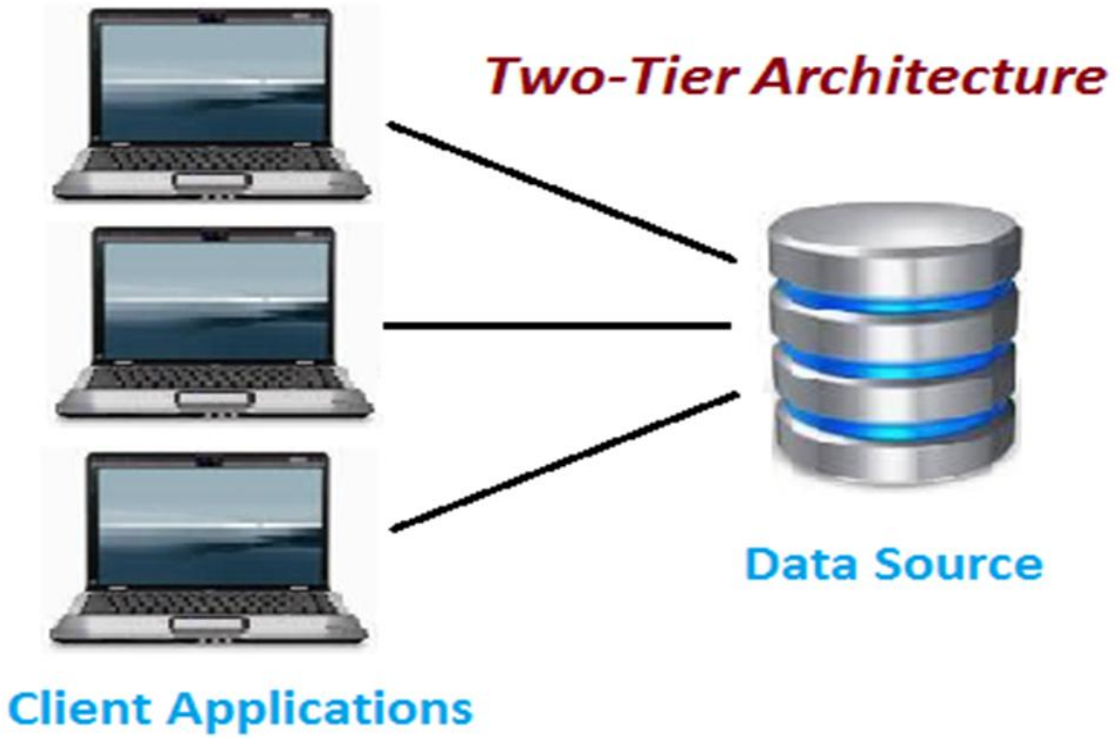


Types of JDBC Architecture



Two-Tier Architecture

- The two-tier is based on Client Server architecture.
- The two-tier architecture is like client server application.
- The direct communication takes place between client and server.
- There is no intermediate between client and server.
- Tight coupling allows 2 tiered application run faster.



Two-Tier Architecture

Pros:

- Easy to maintain and modification is bit easy
- Communication is faster

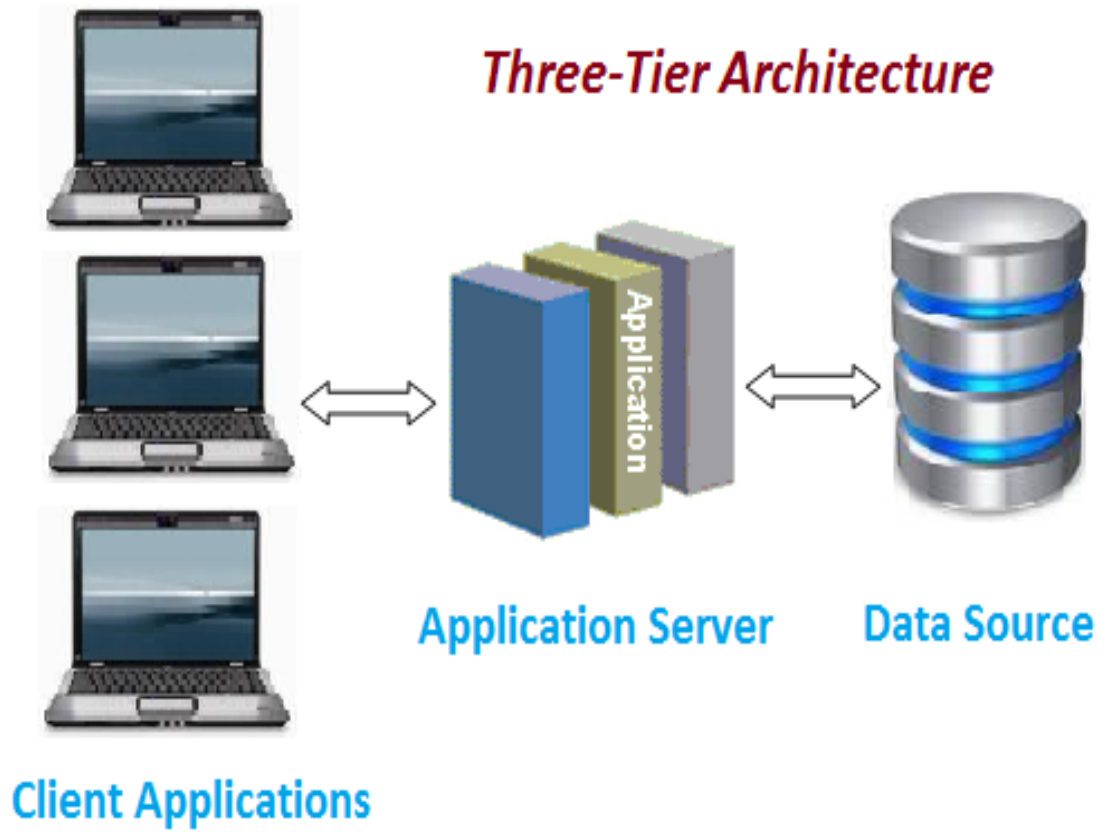
Cons:

- In two tier architecture application performance will be degrade upon increasing the users.
- Cost-ineffective

In this architecture, the application or Java program communicates directly with the database

Three-Tier Architecture

- Three-tier architecture typically comprise a **presentation tier**, a **business** or **data access** tier, and a data tier.
- Three layers in the three tier architecture are as follows:
  1) Client layer
  2) Business layer
  3) Data layer



Examples of three-tier architecture include websites and registration forms.

3- Tier Architecture

1) Client layer:

- It is also called as Presentation layer which contains UI part of our application. This layer is used for the design purpose where data is presented to the user or input is taken from the user. For example designing registration form which contains text box, label, button etc.

2) Business layer:

- In this layer all business logic written like validation of data, calculations, data insertion etc. This acts as a interface between Client layer and Data Access Layer. This layer is also called the intermediary layer helps to make communication faster between client and data layer.

3) Data layer:

- In this layer actual database is comes in the picture. Data Access Layer contains methods to connect with database and to perform insert, update, delete, get data from database based on our input data.

3- Tier Pros & Cons

Pros

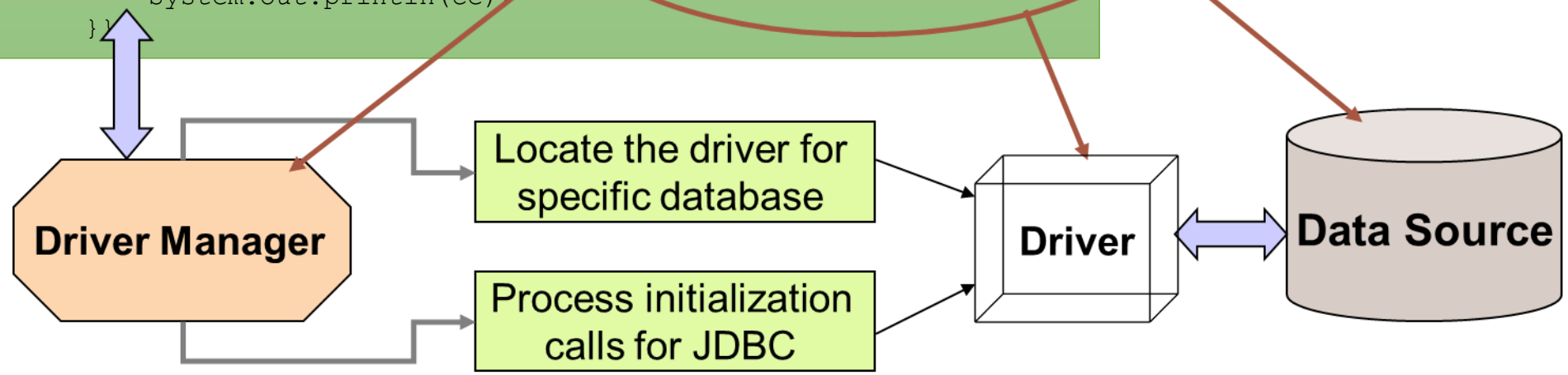
- High performance, lightweight persistent objects
- Scalability – Each tier can scale horizontally
- Performance – Because the Presentation tier can cache requests, network utilization is minimized, and the load is reduced on the Application and Data tiers.
- High degree of flexibility in deployment platform and configuration
- Better Re-use
- Improve Data Integrity
- Improved Security – Client is not direct access to database.
- Easy to maintain and modification is bit easy, won't affect other modules
- In three tier architecture application performance is good.

Cons

- Increase Complexity/Effort

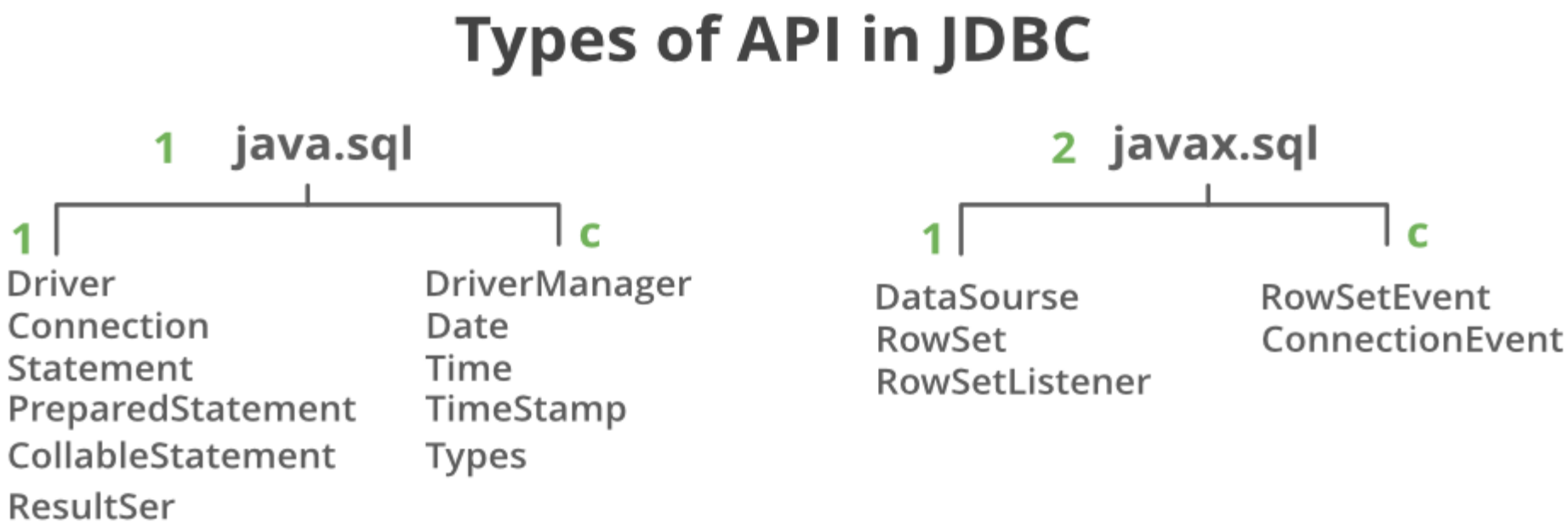
JDBC Components

```
import java.sql.*;
import java.util.*;
class Jdbctest2  {
          public static void main(String args[]) {
          try  {
                    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
          } catch(ClassNotFoundException ce) {
          System.out.println(ce);
          }
          try  {
          String url = "jdbc:odbc:test";
          Connection con = DriverManager.getConnection(url);
          Statement s = con.createStatement();
          ResultSet rs = s.executeQuery("select hiredate,sum(salary)
          from friends group by hiredate");
          while(rs.next()) {
                    System.out.print(rs.getDate(1) +"\t" );
                    System.out.print(rs.getInt(2) +"\t" );
                    System.out.println(" ");
          }
          } catch(SQLException ce) {
              System.out.println(ce);
          }}
}
```



Java.sql / javax.sql package

- The main two packages namely as follows: **java.sql package** and **javax.sql** package.



Popular Classes and Interfaces of JDBC API

Class/Interface	Description
DriverManager class	Manages JDBC drivers and provides methods to establish connections to databases.
Blob class	Represents Binary Large Object (BLOB) data stored in a database.
Clob class	Represents Character Large Object (CLOB) data stored in a database.
Types class	Provides constants that represent the SQL types supported by JDBC.
Driver interface	Represents a JDBC driver and provides methods to connect to a database.
Connection interface	Represents a connection to a database and provides methods for executing SQL statements and managing transactions.
Statement interface	Represents a general SQL statement and provides methods for executing queries and updates.
PreparedStatement interface	Extends the Statement interface and provides methods for executing parameterized SQL statements.
CallableStatement interface	Extends the PreparedStatement interface and provides methods for executing stored procedures.
ResultSet interface	Represents the result set of a database query and provides methods to retrieve and process the data.
ResultSetMetaData interface	Provides methods to retrieve metadata about the columns in a ResultSet.
DatabaseMetaData interface	Provides methods to retrieve metadata about the database, such as supported features and capabilities.
RowSet interface	Represents a disconnected set of rows from a ResultSet. It provides methods to work with data in an offline mode.

java.sql package 3-1

Interface Name	Description
CallableStatement	This contains methods that are used for execution of SQL stored procedures.
Connection	This is used to maintain and monitor database sessions. Data access can also be controlled using the transaction locking mechanism.
DatabaseMetaData	This interface provides database information such as the version number, names of the tables and functions supported.
Driver	This interface is used to create Connection objects.
PreparedStatement	This is used to execute pre-compiled SQL statements.
ResultSet	This interface provides methods for retrieving data returned by an SQL statement.
ResultSetMetaData	This interface is used to collect the meta data information associated with the last ResultSet object.
Statement	It is used to execute SQL statements and retrieve data into the ResultSet.

java.sql package 3-2

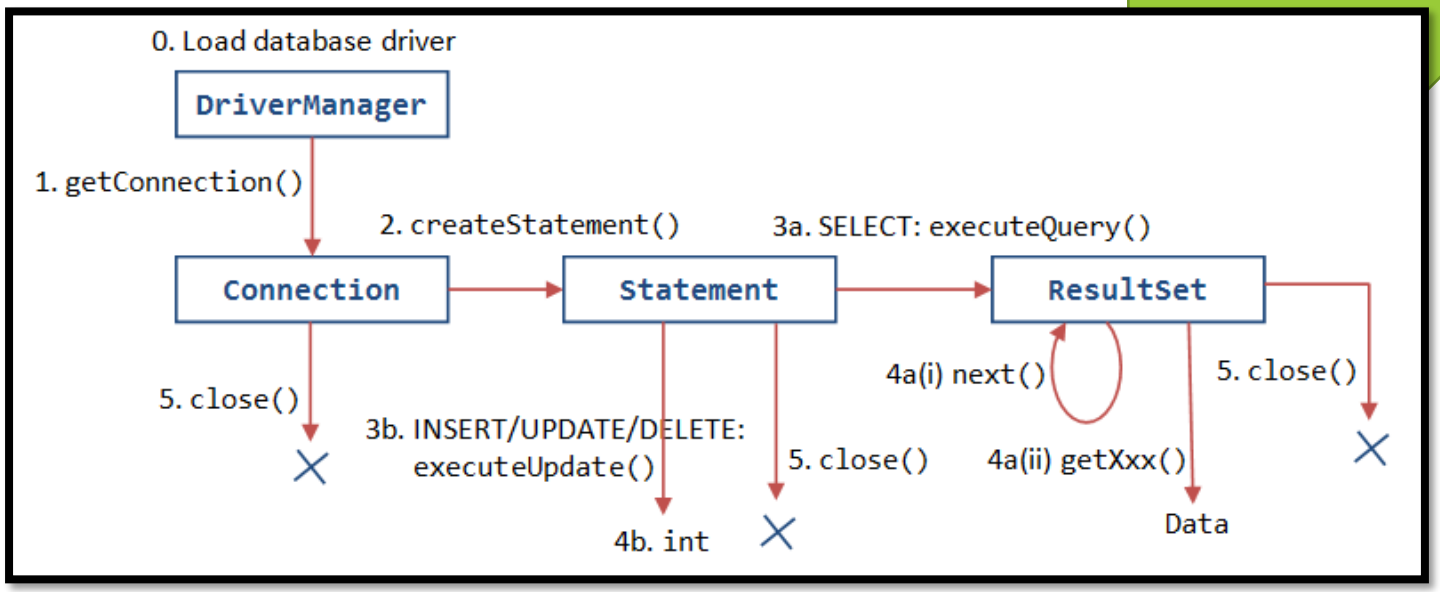
Class Name	Description
Date	This class contains methods for performing conversion of SQL date formats to Java Date formats.
DriverManager	This class is used to handle loading and unloading of drivers and establish connection with the database.
DriverPropertyInfo	The methods in this class are used to retrieve or insert driver properties.
Time	This class provides formatting and parsing operations for Time values.
TimeStamp	This provides precision to Java Date object by adding a Nanosecond field.
Types	This class has no methods. This class defines the constants used to identify the SQL types.

java.sql package 3-3

- *Introducing java.sql package.*

```
…
try{
        fooBar();
catch(SQLException ex)
{
        //If an SQLException is thrown, we'll end up here.
        //Output is the error message, SQLstate and vendor
code.
        System.out.println("An SQLException was caught!");
        System.out.println("Message: " + ex.getMessage());
        System.out.println("SQLState: " + ex.getSQLState());
        System.out.println("Vendor Code: " +
ex.getErrorCode());
}
…
```

Steps for Database Access



Begin

Import the java.sql package

Load and register the driver

Create a Connection object

Create a statement object

Execute the statement

Close ResultSet object

Close Statement object

Close the Connection

End

Registering the Driver

The forName() method of Class class is used to register the driver class.

Class.forName("com.mysql.cj.jdbc.Driver");

Class.forName("oracle.jdbc.driver.OracleDriver”)

Creating Connection Object

The getConnection() method of DriverManager class is used to establish connection with the database.

```
Connection con=DriverManager.getConnection(
"jdbc:mysql://localhost:3306","root","password");
Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:orcl","system","oracle");
```

Creating Statement Object

The createStatement() method of Connection interface is used to create statement.

The object of statement is responsible to execute queries with the database.

Statement stmt=con.createStatement();

Execute Query

The executeQuery() method of Statement interface is used to execute queries to the database.

This method returns the object of ResultSet that can be used to get all the records of a table.

- ResultSet rs=stmt.executeQuery("select * from emp");
  while(rs.next()) {
  System.out.println(rs.getInt(1)+" "+rs.getString(2)); }

Closing Connection

By closing connection object statement and ResultSet will be closed automatically.

The close() method of Connection interface is used to close the connection.

con.close();

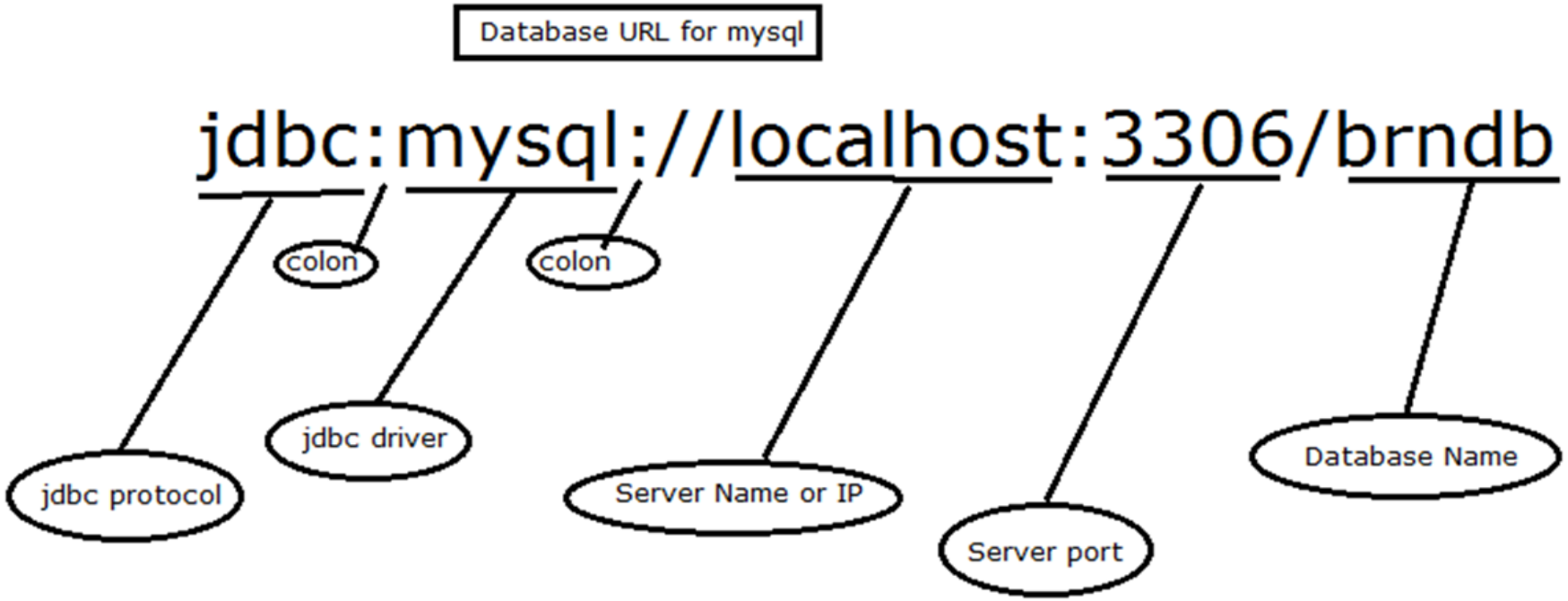
Connecting to the MySQL Database

- Driver class: com.mysql.jdbc.Driver.
- Connection URL: jdbc:mysql://localhost:3306/db_name
- Username: The default username for the mysql database is root.
- Password: Given by the user at the time of installing the mysql database
- Example:

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

```
"jdbc:oracle:thin:@localhost:1521:orcl","system","oracle"
```

Database URL for MYSQL



"jdbc:oracle:thin:@localhost:1521:orcl","system","oracle"

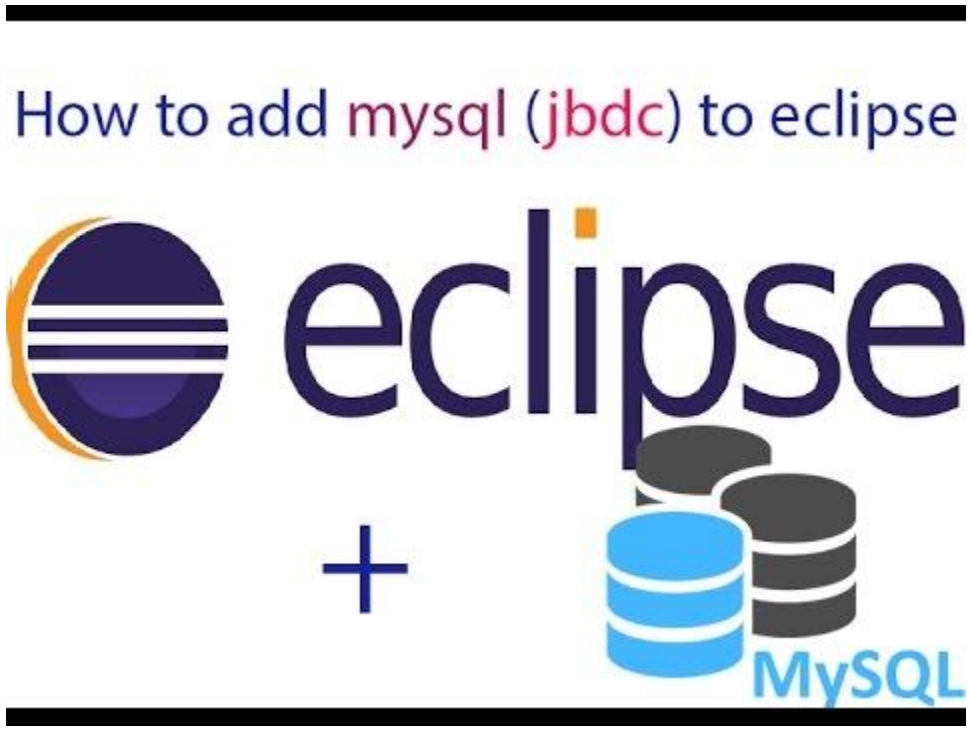
Oracle Service

Loading the .jar

- Download the **MySQL connector.jar** from mysql.com
- Paste the mysqlconnector.jar in the lib folder of source directory.
- Set the classpath

- Add jar file to the build path in the Eclipse IDE

Ojdbc14.jar for Oracle



DriverManager class

- The DriverManager class acts as an interface between user and drivers.
- It keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver.

- **Connection con = null;**

con=DriverManager.getConnection(
"jdbc:mysql://localhost:3306/dbname","root","password");

- **DriverManager.registerDriver().**

"jdbc:oracle:thin:@localhost:1521:orcl","system","oracle"

Connection Interface

- A Connection is the session between Java application and database.
- The Connection interface is a factory of Statement and PreparedStatement.
- Object of Connection can be used to get the object of Statement and PreparedStatement.

Methods of Connection interface

- **public Statement createStatement():** creates a statement object that can be used to execute SQL queries.
- **public void commit():** saves the changes made since the previous commit/rollback permanent.
- **public void close():** closes the connection and Releases a JDBC resources immediately.

Statement Interface

- The Statement interface provides methods to execute queries with the database.
- The statement interface is a factory of ResultSet.
- It provides factory method to get the object of ResultSet.

Methods of Statement interface

- **public ResultSet executeQuery(String sql):** is used to execute SELECT query. It returns the object of ResultSet.
- **public int executeUpdate(String sql):** is used to execute specified query, it may be create, drop, insert, update, delete etc.
- **public boolean execute(String sql):** is used to execute queries that may return multiple results.

Statement Interface Example

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

int result=stmt.executeUpdate("delete from table where id=xy");

System.out.println(result+" records affected");

con.close();
```

ResultSet interface

ResultSet interface represents the result set of a database query.

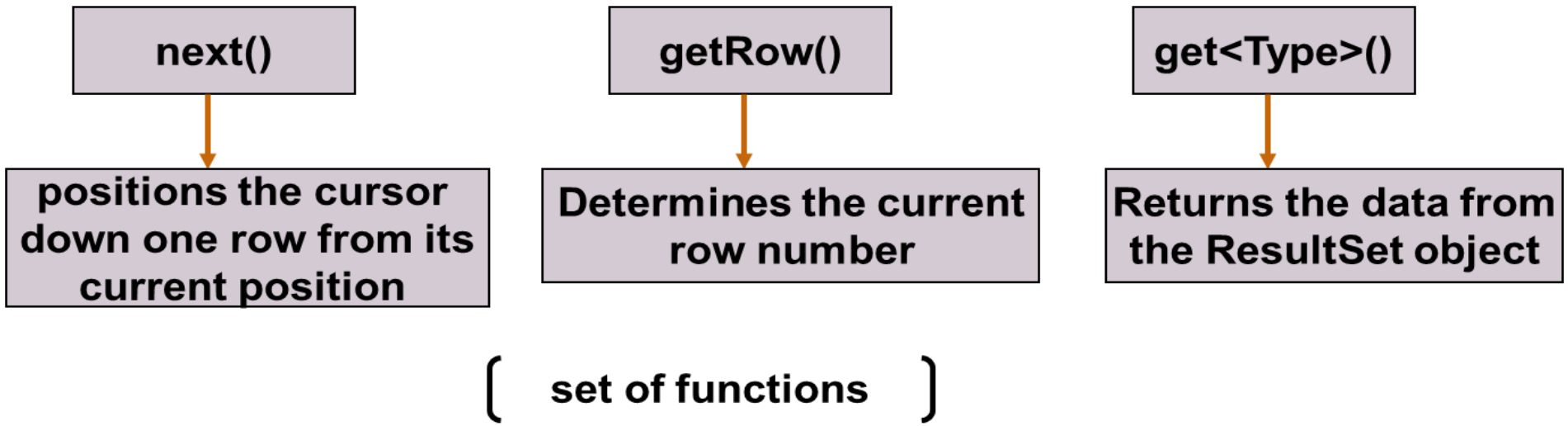
The object of ResultSet maintains a cursor pointing to a particular row of data.

Initially, cursor points to before the first row.

The term "result set" refers to the row and column data contained in a ResultSet object.

Working with ResultSets

- ResultSet objects are entirely dependent on the Statement and Connection objects.
- Each time an SQL statement is executed, the resultset is overwritten with new results.
- The ResultSet object gets automatically closed when the related Statement is closed.



Methods of ResultSet interface

- **public boolean next():** is used to move the cursor to the one row next from the current position.
- **public boolean previous():**is used to move the cursor to the one row previous from the current position.
- **public boolean first():** is used to move the cursor to the first row in result set object.
- **public boolean last():**is used to move the cursor to the last row in result set object.

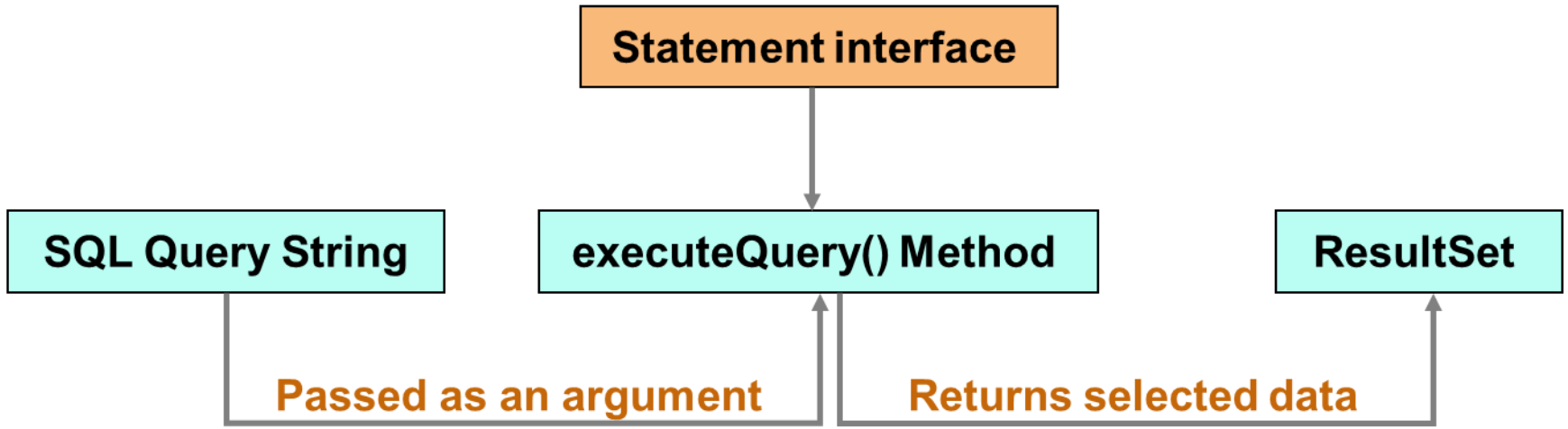
Methods of ResultSet interface

- **public int getInt(int columnIndex):** is used to return the data of specified column index of the current row as int.
- **public int getInt(String columnName):**is used to return the data of specified column name of the current row as int.
- **public String getString(int columnIndex):** is used to return the data of specified column index of the current row as String.
- **public String getString(String columnName):** is used to return the data of specified column name of the current row as String.

ResultSet Interface Example

```
ResultSet rs=stmt.executeQuery("select * from table");

 //getting the record of 3rd row
rs.absolute(3);
System.out.println(rs.getString(1)+" "+rs.getString(2)+" "+rs.getString(3));

 con.close();
```

Using SQL 5-1



In SQL, query can be written as

```
SELECT name, email, phone FROM colleagues;
```

for JDBC the Java code to execute the above query would be

```
String str = "SELECT emp_id, lname, fname FROM colleagues";
Statement stmt = con.createStatement();
ResultSet rset = stmt.executeQuery(str);
```

Setting Up MySQL JDBC Development Environment

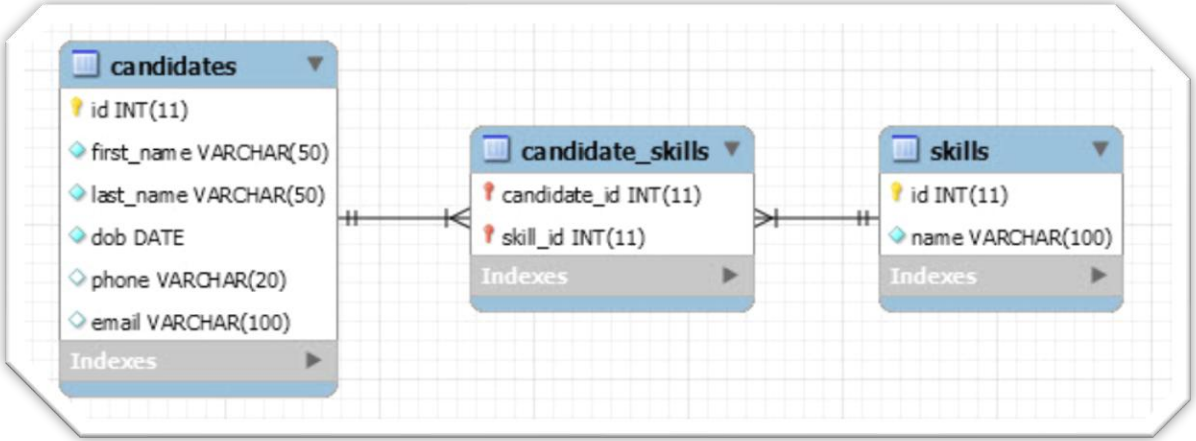
- Download and install JDK.
- Download and install NetBeans / Eclipse IDE.
- Download MySQL Connector/J /ojdbc14.jar and configure it to build path in Eclipse.

Connecting & Querying Data From MySQL Using JDBC

```
import java.sql.*;

public class JdbcMySQLDemo {
   // Database credentials
   private static final String URL = "jdbc:mysql://localhost:3306/companydb";
   private static final String USER = "root";
   private static final String PASSWORD = "yourpassword";

   public static void main(String[] args)  {
     Connection conn = null;
      Statement stmt = null;
      ResultSet rs = null;

try {
         // Step 1: Load MySQL JDBC Driver (optional for JDBC 4.0+, but safer)
         Class.forName("com.mysql.cj.jdbc.Driver");
         System.out.println(" ✅ MySQL Driver Loaded Successfully.");

         // Step 2: Establish Connection
         conn = DriverManager.getConnection(URL, USER, PASSWORD);
         System.out.println(" ✅ Connected to Database Successfully.");
```


```
      // Step 3: Create Statement
         stmt = conn.createStatement();

         // Step 4: Execute Query
         String sql = "SELECT id, name, department, salary FROM employees";
         rs = stmt.executeQuery(sql);

       System.out.println("------ Employee Records ------");

         // Step 5: Process ResultSet
         while (rs.next()) {
            int id = rs.getInt("id");
            String name = rs.getString("name");
            String department = rs.getString("department");
            double salary = rs.getDouble("salary");

            System.out.printf("ID: %d | Name: %s | Department: %s | Salary:
%.2f%n",id, name, department, salary);          }
}    catch (Exception e) {
         System.out.println(" ❌ Database operation failed.");
         e.printStackTrace();
       } finally {
         // Step 6: Close resources manually
try {
      if (rs != null) rs.close();   if (stmt != null) stmt.close();  if (conn != null)
conn.close();
   } catch (Exception e) {
      e.printStackTrace();
   }
   System.out.println(" ✅ Resources Closed."); } } }
```

MySQL JDBC sample database - mysqljdbc



The sample database includes three tables:

- **candidates:** stores candidate data such as first name, last name, date of birth (dob), phone, and email.
- **skills:** stores skills of the candidates
- **candidate_skills:** the pivot table links between candidates & skills

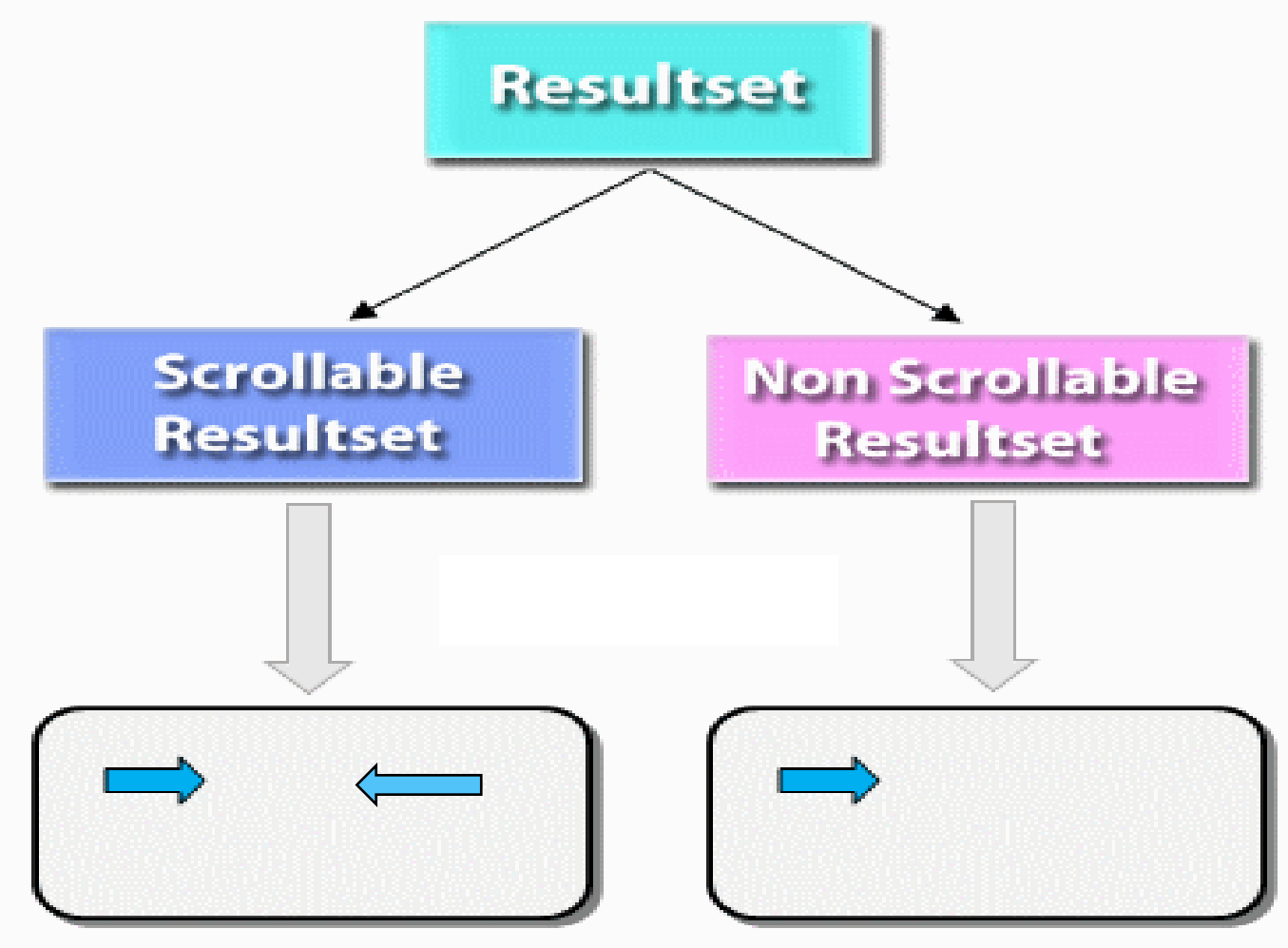
Resultset in JDBC

In Jdbc ResultSet Interface are classified into two type;.

- Non-Scrollable ResultSet in JDBC
- Scrollable ResultSet

- By default a ResultSet Interface is Non-Scrollable.
- In non-scrollable ResultSet we can move only in forward direction, but not in Backward Direction,
- If you want to move in backward direction use Scrollable Interface.

Types of Resultset



Difference between Scrollable ResultSet and Non-Scrollable ResultSet

Non-Scrollable ResultSet	Scrollable ResultSet
Cursor move only in forward direction	Cursor can move both forward and backward direction
Slow performance, If we want to move nth record then we need to n+1 iteration	Fast performance, directly move on any record.
Non-Scrollable ResultSet cursor can not move randomly	Scrollable ResultSet cursor can move randomly

Create Scrollable ResultSet

- To create a Scrollable ResultSet, create Statement object with two parameters.

Syntax :

Statement stmt=con.CreateStatement(param1, param2);

// parm1 type and param2 mode

Scrollable ResultSet – Type

- These type and mode are predefined in ResultSet Interface of Jdbc like below which is static final.

Type	Description
ResultSet.TYPE_FORWARD_ONLY	The cursor can only move forward in the result set.
ResultSet.TYPE_SCROLL_INSENSITIVE	The cursor can scroll forward and backward, and the result set is not sensitive to changes made by others to the database that occur after the result set was created.
ResultSet.TYPE_SCROLL_SENSITIVE.	The cursor can scroll forward and backward, and the result set is sensitive to changes made by others to the database that occur after the result set was created.

```
Statement stmt=con.CreateStatement(ResutlSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);
```

Scrollable ResultSet – Mode & Methods

Concurrency	Description
ResultSet.CONCUR_READ_ONLY	Creates a read-only result set. This is the default
ResultSet.CONCUR_UPDATABLE	Creates an updateable result set.

Below all methods are used for move the cursor in Scrollable ResultSet.

afterLast: Used to move the cursor after last row.
BeforeFirst: Used to move the cursor before first row.
previous: Used to move the cursor backward.
first: Used to move the cursor first at row.
last: Used to move the cursor at last row.
absolute: Moves the cursor to the specified row.
relative: Moves the cursor the given number of rows forward or backward.

The executeUpdate( ) Method

- The executeUpdate( ) method is used to insert, update, and delete rows in the tables.
- It is used to execute DML Commands.
- It returns an integer value that reports the number of rows affected by the SQL statement.

```
int       rslt = 0;
Statement stmt = null;
try {
  stmt = conn.createStatement(  );
  rslt = stmt.executeUpdate("delete person");
  . . .
}
```

Using SQL 5-4

Statement	Syntax	Description
ALTER TABLE <tableName>	ALTER TABLE emp ADD (emp_email char(20))	A new column called emp_email is added to the table emp of character size 20.
INSERT INTO <tableName>	INSERT INTO emp VALUES ('Jennifer','E004',32,'jenni @abc.com')	The values specified here are inserted into the table emp.
DELETE FROM <tablename>	DELETE FROM emp WHERE emp_name = 'Bradman'	Deletes all the rows with the emp_name equal to 'Bradman' from the emp table.
DROP TABLE <tableName>	DROP TABLE emp	The table named 'emp' is permanently removed from the database.

Using SQL Insert

```
/** This is a main method. */
    public static void main(final String [] args) {
    try {
        Class.forName(" com.mysql.cj.jdbc.Driver ");
```

```
ResultSet rs = s.executeQuery(str1);
while (rs.next()) {
            System.out.print(rs.getString(1) + "\t");
            System.out.print(rs.getInt(2) + "\t");
            System.out.println(" ");
    }
    } catch (SQLException ce) {
        System.out.println(ce);
    }
    }
}
```

```
        String str1 = "select name, avg(salary) from
friends"
                        + " group by name";
```

```
/** This is a main method. */
    public static void main(final String [] args) {
    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
    } catch (ClassNotFoundException ce) {
        System.out.println(ce);
    }
    try {
        String url = "jdbc:odbc:test";
        String str = "INSERT INTO
friends(name,address,salary)" +
"VALUES('Jessica','Alaska',25690)";
        Connection con = DriverManager.getConnection(url);
        Statement s = con.createStatement();
        int rowcount = s.executeUpdate(str);
        String str1 = "select name, avg(salary) from
friends"
                        + " group by name";
```

Using SQL 5-5

```
public static void main(final String [] args) {
    Connection con;
    Statement stmt;
    String url;
    String sql;
     con = DriverManager.getConnection(url);
           sql = "Update friends set address=\'Adelaide\'
where " + "rtrim(name) like \'Mike\'; ";
           System.out.println(" ");
           stmt = con.createStatement();
           stmt.executeUpdate(sql);
           con.close();
           System.out.println("Record for Mike updated");
       } catch (SQLException ce) {
           System.out.println(ce);
       }
       }
 }
```

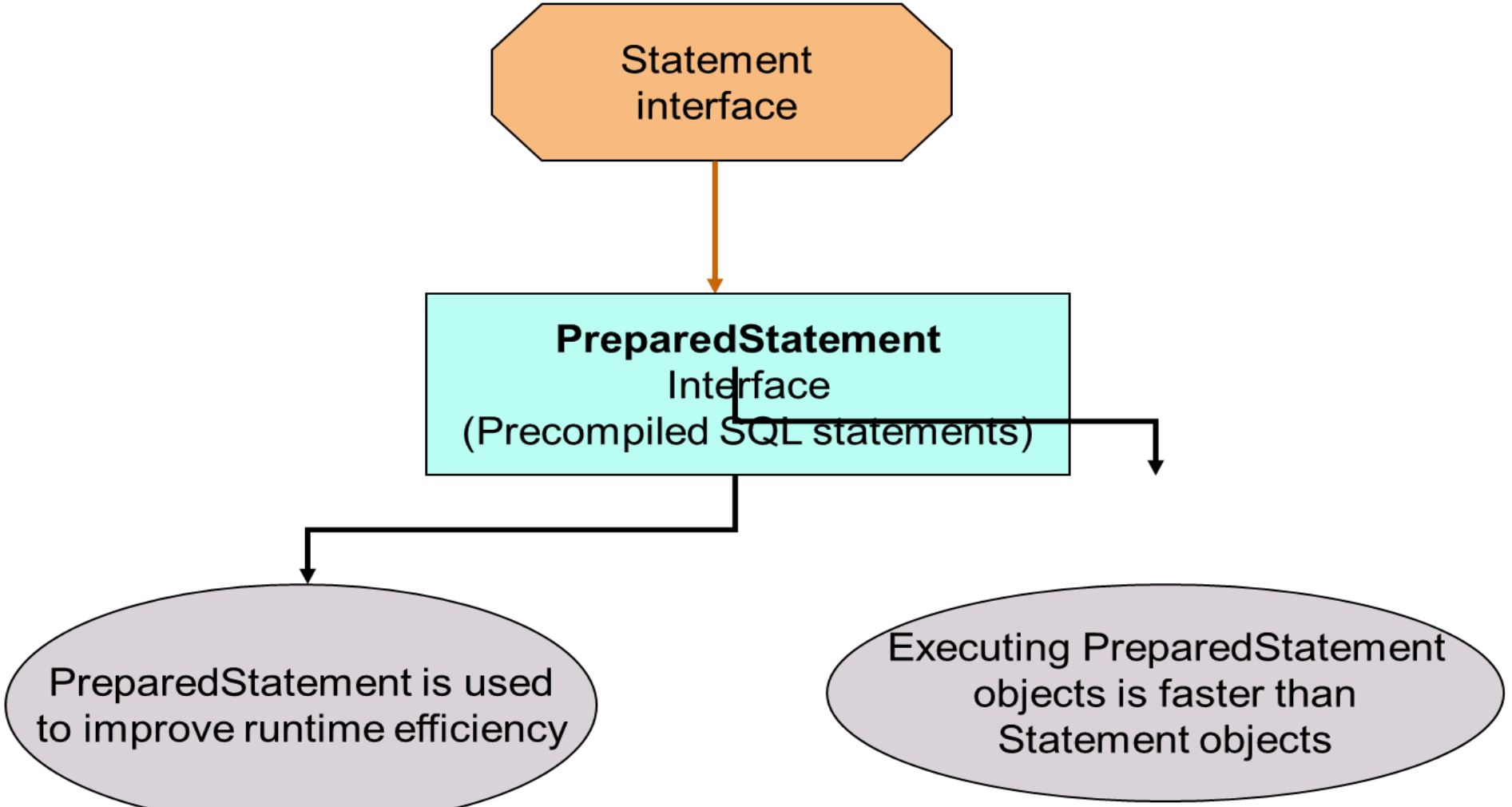
PreparedStatement Interface

- The PreparedStatement interface is a sub interface of Statement.
- If the sql command is same then actually no need to compiling it for each time before it is executed.
- So the performance of an application will be Increased.
- In this case **PreparedStatement** is used.
- It accepts values to query at runtime.
- It is used to execute parameterized query.
- Example of parameterized query: –

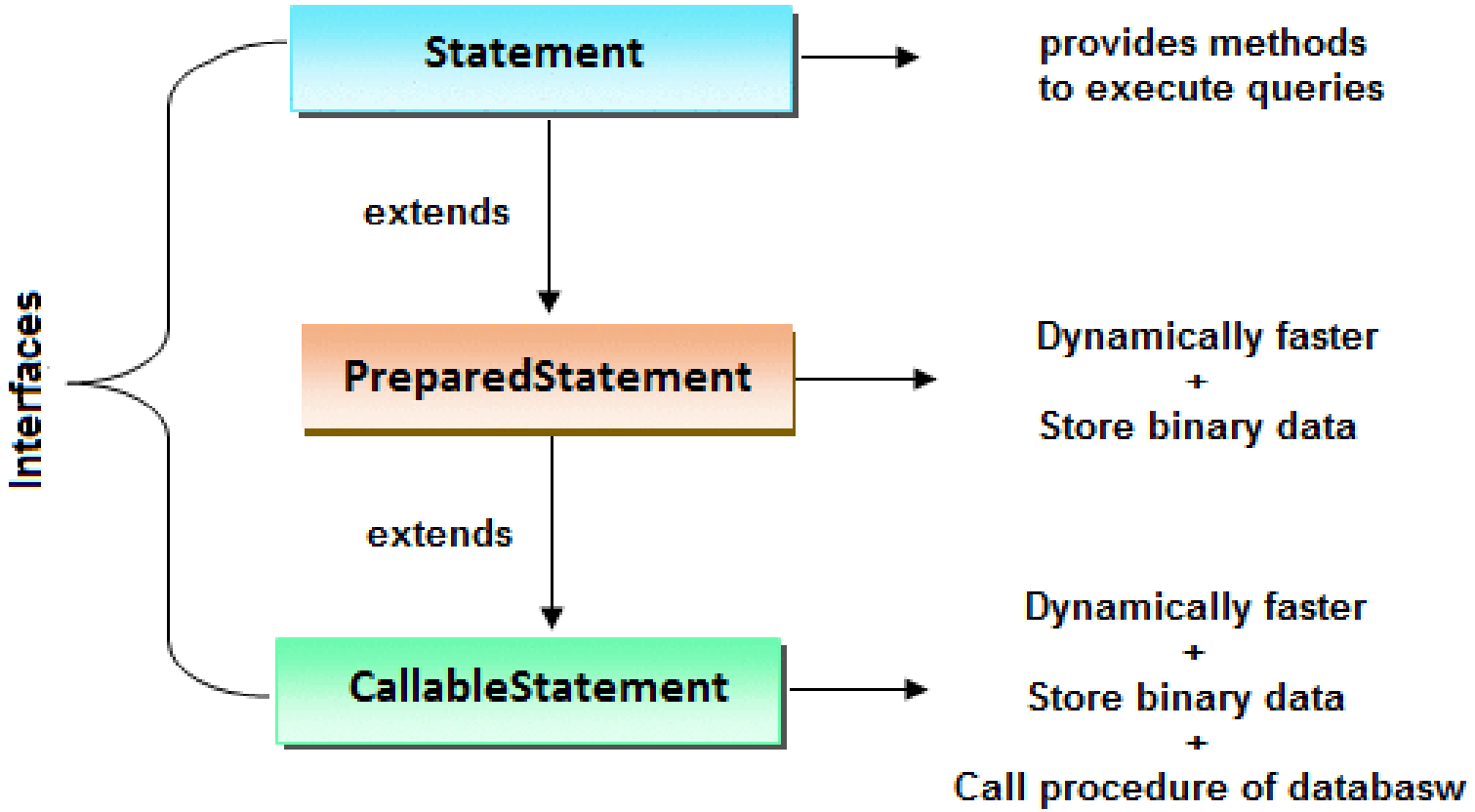
String sql="insert into emp values(?,?,?)";

Prepared Statement interface 3-1

The PreparedStatement interface allows us to create pre-compiled SQL statements and gives the option of specifying the parameters for the statement at a later stage



PreparedStatement

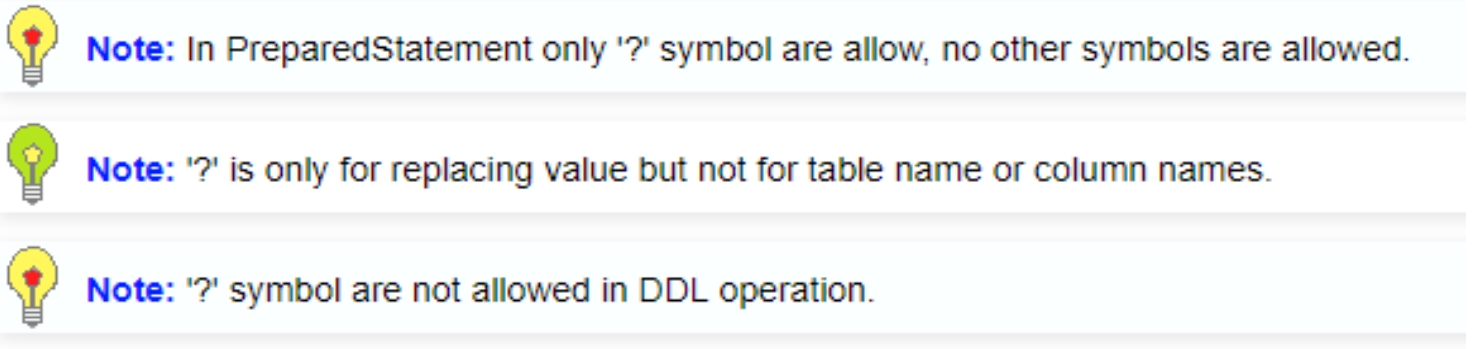


Difference between PreparedStatement and Statement

Statement	PreparedStatement
Statement interface is slow because it compile the program for each execution	PreparedStatement interface is faster, because its compile the command for once.
We can not use ? symbol in sql command so setting dynamic value into the command is complex	We can use ? symbol in sql command, so setting dynamic value is simple.
We can not use statement for writing or reading binary data (picture)	We can use PreparedStatement for reading or writing binary data.

Why use '?' symbol in PreparedStatement

- To pre-compile a command only syntax of the command is required.
- So we can use '?' symbol for value in the command.
- '?' symbol is called parameter or replacement operator or place-resolution operator.



Methods of PreparedStatement

- **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.

Methods of PreparedStatement

- **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 Interface Example

```
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");
```

Note that you can use any statement such as SELECT, INSERT, DELETE, etc with PreparedStatement interface.

SQL injection

- SQL injection is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database.
- It generally allows an attacker to view data that they are not normally able to retrieve.
- This might include data belonging to other users, or any other data that the application itself is able to access.
- In many cases, an attacker can modify or delete this data, causing persistent changes to the application's content or behaviour.
- In some situations, an attacker can escalate an SQL injection attack to compromise the underlying server or other back-end infrastructure, or perform a denial-of-service attack.

Callable Statement

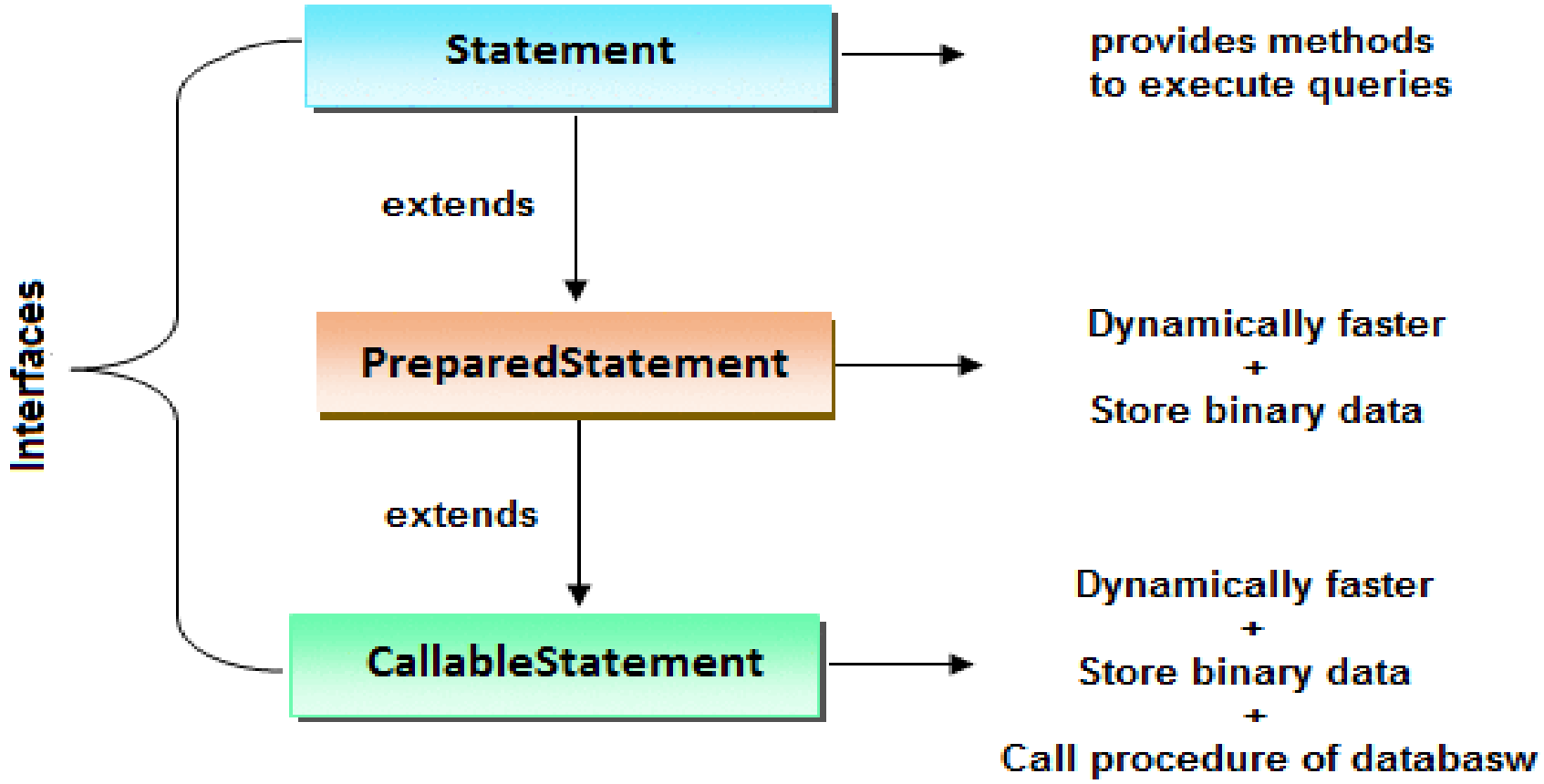
- Callable statement is used to call Stored Procedures and functions.
- We can have business logic on the database by the use of stored procedures and functions.
- It will make performance better because they are precompiled.

- Eg: To get the total experience of an employee based on DOJ.

Callablestatement in Jdbc

- To call the procedures and functions of a database, CallableStatement interface is used.
- CallableStatement is a derived Interface of preparedStatement.
- It has one additional feature over PreparedStatement that is calling procedures and function of a database.
- Because of CallableStatement is inherited from PreparedStatement all the features of PreparedStatement are also available with CallableStatement.

Callable Statement



Stored Procedures & Functions

- We can have business logic on the database by the use of stored procedures and functions.
- It will make the performance better because these are precompiled.

Example:

- Suppose you need the get the age of the employee based on the date of birth, you may create a function that receives date as the input and returns age of the employee as the output.

Stored Procedures vs Functions

Stored Procedure	Function
is used to perform business logic.	is used to perform calculation.
must not have the return type.	must have the return type.
may return 0 or more values.	may return only one values.
We can call functions from the procedure.	Procedure cannot be called from function.
Procedure supports input and output parameters.	Function supports only input parameter.
Exception handling using try/catch block can be used in stored procedures.	Exception handling using try/catch can't be used in user defined functions.

Create object of CallableStatement

- The **prepareCall()** method of Connection interface returns the instance of CallableStatement.
- To create a reference of CallableStatement we have two syntaxes one with command and the other is with calling procedure or function.

Syntax of prepareCall() method

```
public CallableStatement prepareCall("{ call procedurename(?,?...?)}");
```

Syntax

```
CallableStatement cstmt=con.prepareCall("sql command");
```

Syntax of prepareCall() method

Syntax of prepareCall() method

```
public CallableStatement prepareCall("{ call procedurename(?,?...?)}");
```

Syntax

```
CallableStatement cstmt=con.prepareCall("sql command");
```

Syntax

```
CallableStatement cstmt=con.prepareCall("{call procedure name or function name}");
```

Syntax

```
CallableStatement stmt=con.prepareCall("{call myprocedure(?,?)}");
```

The general syntax of calling a stored procedure

{?= call procedure_name(param1,param2,...)}

- Wrap the stored procedure call within braces ({}).
- If the stored procedure returns a value, you need to add the question mark and equal (?=) before the call keyword.
- If a stored procedure does not return any values, you just omit the ?= sign.
- In case the stored procedure accepts any parameters, you list them within the opening and closing parentheses after the stored procedure's name

Examples of using the syntax for calling stored procedures in different contexts

Syntax	Stores Procedures
{ call procedure_name() }	Accept no parameters and return no value
{ call procedure_name(?,?) }	Accept two parameters and return no value
{?= call procedure_name() }	Accept no parameter and return value
{?= call procedure_name(?) }	Accept one parameter and return value

JDBC MySQL stored procedure example -1

- Open a connection to MySQL server by creating a new Connection object.
- Then, prepare a stored procedure call and create a CallableStatement object by calling prepareCall() method of the Connection object.

```
String query = "{CALL get_candidate_skill(?)}";
CallableStatement stmt = conn.prepareCall(query)
```

- Next, pass all the parameters to the stored procedure. In this case, the get_candidate_skill stored procedure accepts only one IN parameter.

```
stmt.setInt(1, candidateId);
```

JDBC MySQL stored procedure example -2

- After that, execute the stored procedure by calling the executeQuery() method of the CallableStatement object.
- It returns a result set in this case.

ResultSet rs = stmt.executeQuery();

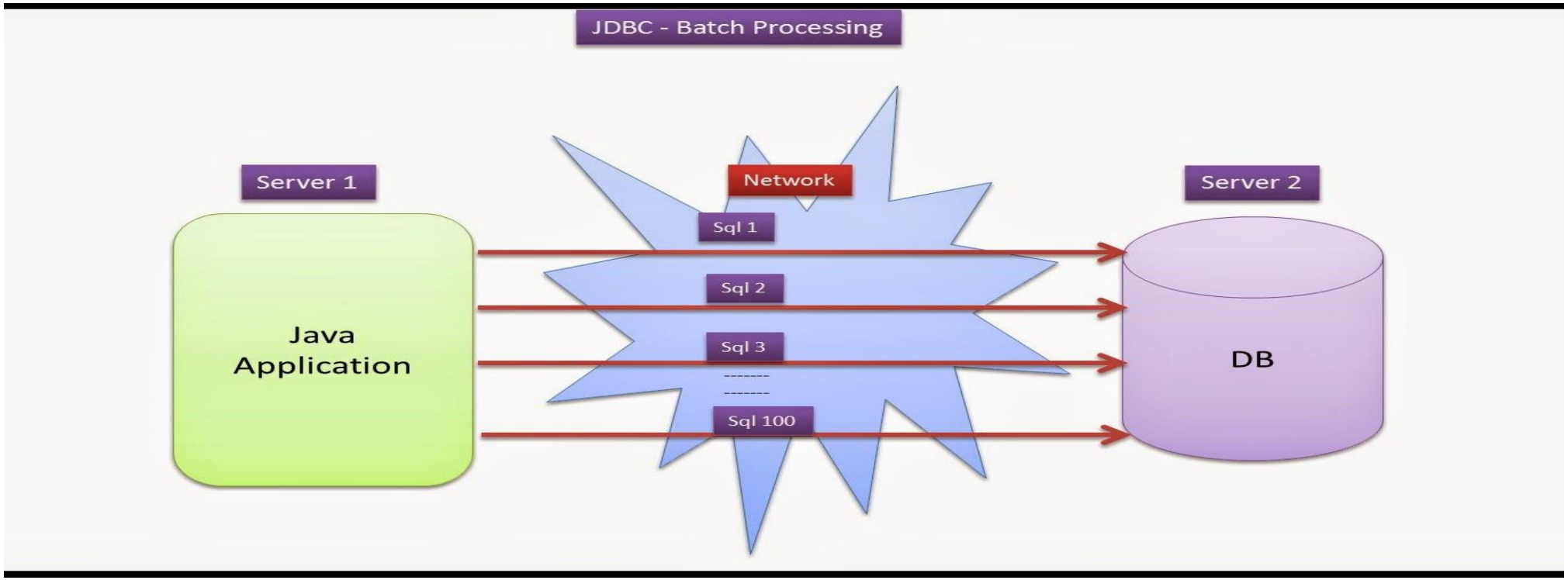
- Finally, traverse the ResultSet to display the results.

Callable Statement Example

```
String query = "{ call get_candidate_skill(?) }";
        ResultSet rs;

        try {
            CallableStatement stmt = conn.prepareCall(query)) {

           stmt.setInt(1, candidateId);

           rs = stmt.executeQuery();
           while (rs.next()) {
              System.out.println(String.format("%s - %s",
                    rs.getString("first_name") + " "
                    + rs.getString("last_name"),
                    rs.getString("skill")));
           }
```

Batch Processing in Jdbc

- Instead of executing a single query, we can execute a batch (group) of queries using batch processing. It makes the performance fast.
- The **java.sql.Statement** and **java.sql.PreparedStatement** interfaces provide methods for batch processing.



Why need of Batch Processing ?

- In a Jdbc program if multiple sql operations are there, then each operation is individually transferred to the database.
- This approach will increase the number of round trips between java program (application) and database.
- If the number of **round trips** is increased between an application and database, then it will reduce the performance of an application.
- To overcome these problems we use Batch Processing.

Advantage:

- Increase the performance of an application.

Methods of Batch Processing

- These methods are given by Statement Interface.

Method	Description
void addBatch(String query)	It adds query into batch.
int[] executeBatch()	It executes the batch of queries.

- In Batch Processing only **non-select** operations are allowed select operation is not allowed.
- If any operation failed in batch processing then **java.sql.BatchUpdate** Exception will be thrown.
- When we want to cancel all the operation of the batch when one operation failed then apply batch processing with transaction management.

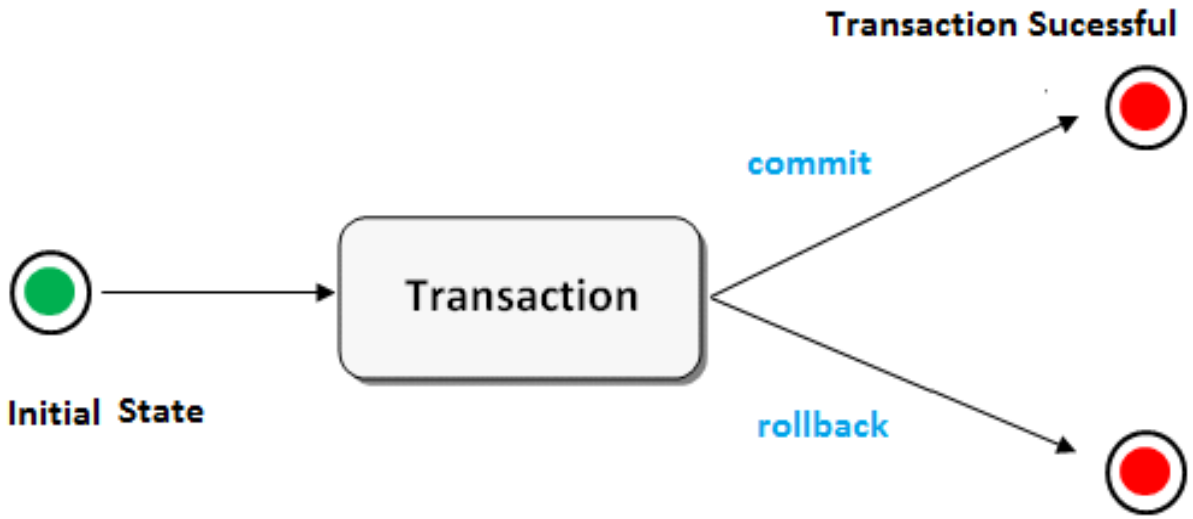
Batch Processing - Example

```
//create batch
stmt.addBatch("insert into
student values(901,'PQR',788)");
stmt.addBatch("update emp_info
set esal=8888 where eno=1012");
stmt.addBatch("delete from
customer where custid=111");

//disabl auto-commit mode
con.setAutoCommit(false);
```

```
try {
 int i[]=stmt.executeBatch();
 con.commit();
 System.out.println("batch is successfully
executed"); }
catch (Exception e)
{
Try {
con.rollback();
System.out.println("batch is failed")  }
```

Transaction Management in JDBC

- A transaction is a group of operation used to performed one task.
- If all operations in the group are success then the task is finished and the transaction is successfully completed.
- If any one operation in the group is failed then the task is failed and the transaction is failed.

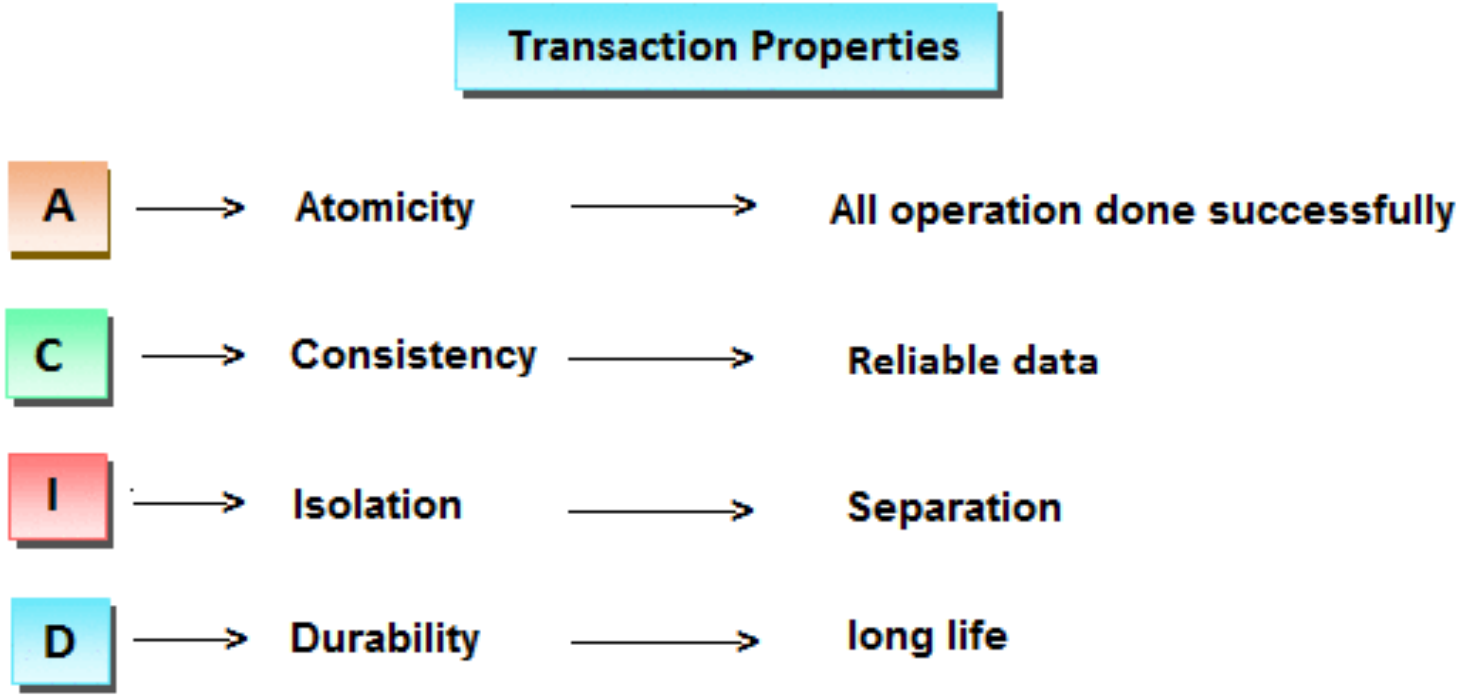


Transaction - Example

- Suppose a movie ticket booking at online is a transaction. This task contains four operation.
  - Verify the seats
  - Reserve the seats
  - Payment
  - Issue tickets
- If all the above four operations are done successfully then a transaction is finished successfully.
- In the middle, if any one operation is failed then all operation are canceled and finally a transaction is failed.

Properties of Transaction managements

- Every transaction follows some transaction properties these are called ACID properties.



ACID

- **Atomicity:** Atomicity of a transaction is nothing but in a transaction either all operations can be done or all operation can be undone, but some operations are done and some operation are undone should not occur.
- **Consistency:** Consistency means, after a transaction completed with successful, the data in the datastore should be a reliable data this reliable data is also called as consistent data.
- **Isolation:** Isolation means, if two transaction are going on same data then one transaction will not disturb another transaction.
- **Durability**: Durability means, after a transaction is completed the data in the data store will be permanent until another transaction is going to be performed on that data.

Types of Transaction

- **Local Transaction**
  - A local transaction means, all operation in a transaction are executed against one database.
    For example; If transfer money from first account to second account belongs to same bank then transaction is local transaction.
- **Global Transaction**
  - A global transaction means, all operations in a transaction are executed against multiple database.
  - For Example; If transfer money from first account to second account belongs to different banks then the transaction is a global transaction.

- **Jdbc technology perform only local transactions.**
- **For global transaction in java we need either EJB or spring framework.**

Things required for transaction in Jdbc

Step 1: Disable auto commit mode of Jdbc

Step 2: Put all operation of a transaction in try block.

Step 3: If all operation are done successfully then commit in try block, otherwise rollback in catch block.

- By default in Jdbc autocommit mode is enabled but we need to disable it.
- To disable call setAutoCommit() method of connection Interface.

con.setAutoCommit(false);

- To commit a transaction, call commit() and to rollback a transaction, call rollback() method of connection Interface respectively.

con.commit();
con.rollback();

Transaction Example

```
Statement stmt=con.createStatement();
con.setAutoCommit(false);
Try {
int i1=stmt.executeUpdate("insert into student values(110,‘Ravi',685)");
int i2=stmt.executeUpdate("update customer set custadd=‘Bangalore‘ where
custid=111");
int i3=stmt.executeUpdate("delete from student where sid=101");
con.commit();
System.out.println("Transaction is success"); }//end of try
catch (Exception e) {
Try {
con.rollback();
System.out.println("Trasaction is failed"); }
```

Resultsetmetadata Interface in JDBC

- The metadata means data about data, in case of database we get metadata of a table like total number of column, column name, column type etc.
- While executing a select operation on a database if the table structure is already known for the programmer, then a programmer of JDBC can read the data from ResutlSet object directly. If the table structure is unknown then a JDBC programmer has to take the help of ResultSetMetadata.
- A ResultSetMetaData reference stores the metadata of the data selected into a ResutlSet object.

Object & Methods of ResultSetMetaData

- **ResultSetMetaData rsmd=rs.getMetaData();**

method	Discription
getColumnCount()	To find the number of columns in a ResultSet
getColumnName()	To find the column name of a column index.
getColumnTypeName()	To find data type of a column.
getColumnDisplaySize()	To find size of a column.

Example of ResultSetMetaData

```
Statement stmt=con.createStatement();
ResultSet rs=stmt.executeQuery("select * from student");
ResultSetMetaData rsmd=rs.getMetaData();
//find the no of columns
int count=rsmd.getColumnCount();
for(int i=1;i <=count;i++)
{
System.out.println("column no :"+i);
System.out.println("column name :"+rsmd.getColumnName(i));
System.out.println("column type :"+rsmd.getColumnTypeName(i));
System.out.println("column size :"+rsmd.getColumnDisplaySize(i));
System.out.println("----------");
}
```

DatabaseMetaData in JDBC

- This metadata interface is used for reading metadata about a database.
- Metadata about database is nothing but reading database server, version, driver name and its version, maximum number of columns allowed in a table etc.
- To obtain a object of DatabaseMetaData, we need to call **getMetaData()** method of Connection object.

DatabaseMetaData dbms=con.getMetaData();

Methods of DatabaseMetaData

Method	Description
getDatabaseProductName()	To read database server name.
getDatabaseProductVersion()	To read database server version.
getDriverName()	To read driver software name.
getColumnInTable()	To find maximum number of columns allowed in the table..

Example of DatabaseMetaData

```
Statement stmt=con.createStatement();
DatabaseMetaData dbmd=con.getMetaData();
System.out.println("database server
name:"+dbmd.getDatabaseProductName());
System.out.println("database server
version:"+dbmd.getDatabaseProductVersion());
System.out.println("driver server version:"+dbmd.getDriverVersion());
System.out.println("driver server name:"+dbmd.getDriverName());
System.out.println("max columns:"+dbmd.getMaxColumnsInTable());
stmt.close();
con.close();
```

END