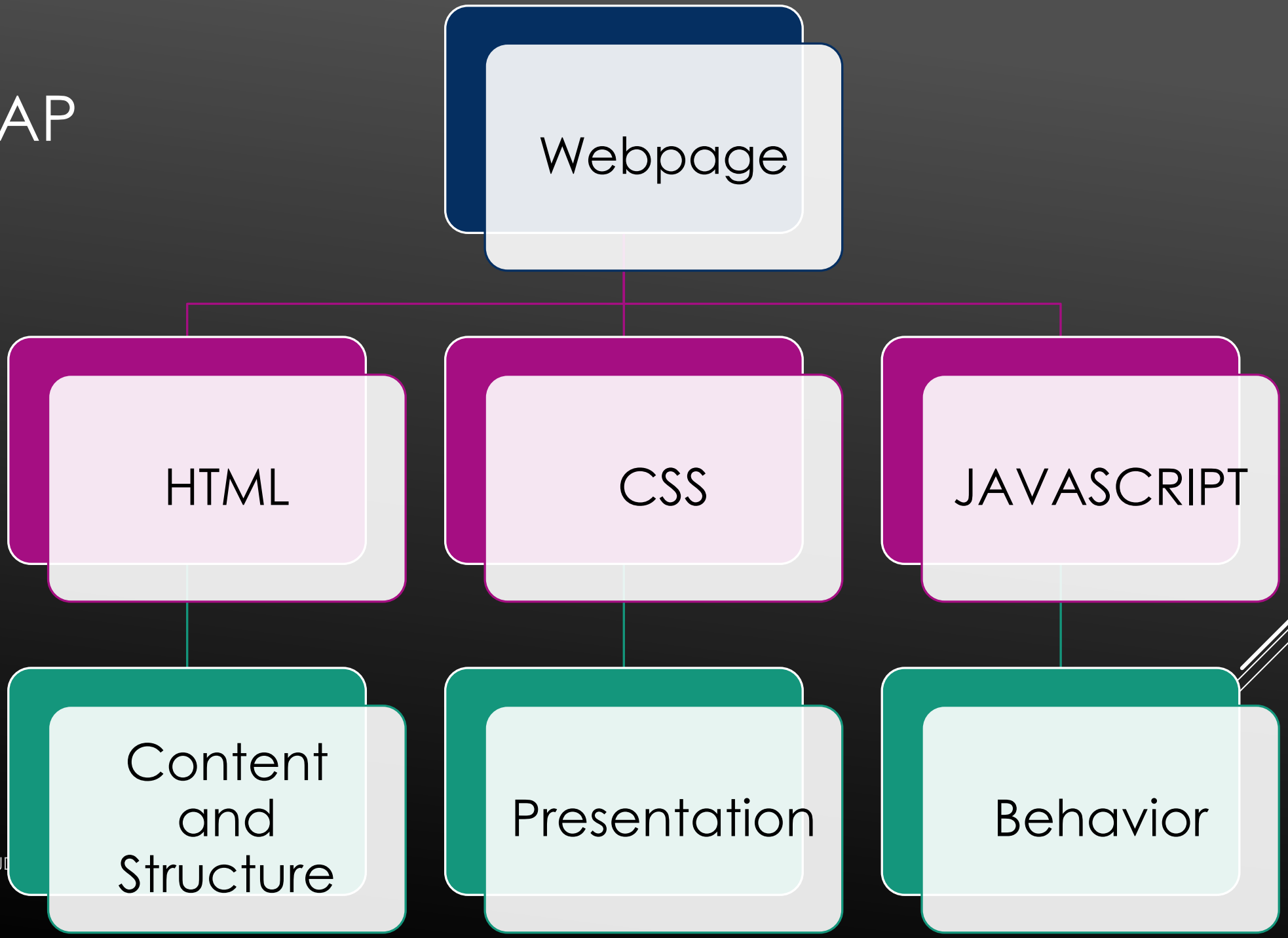
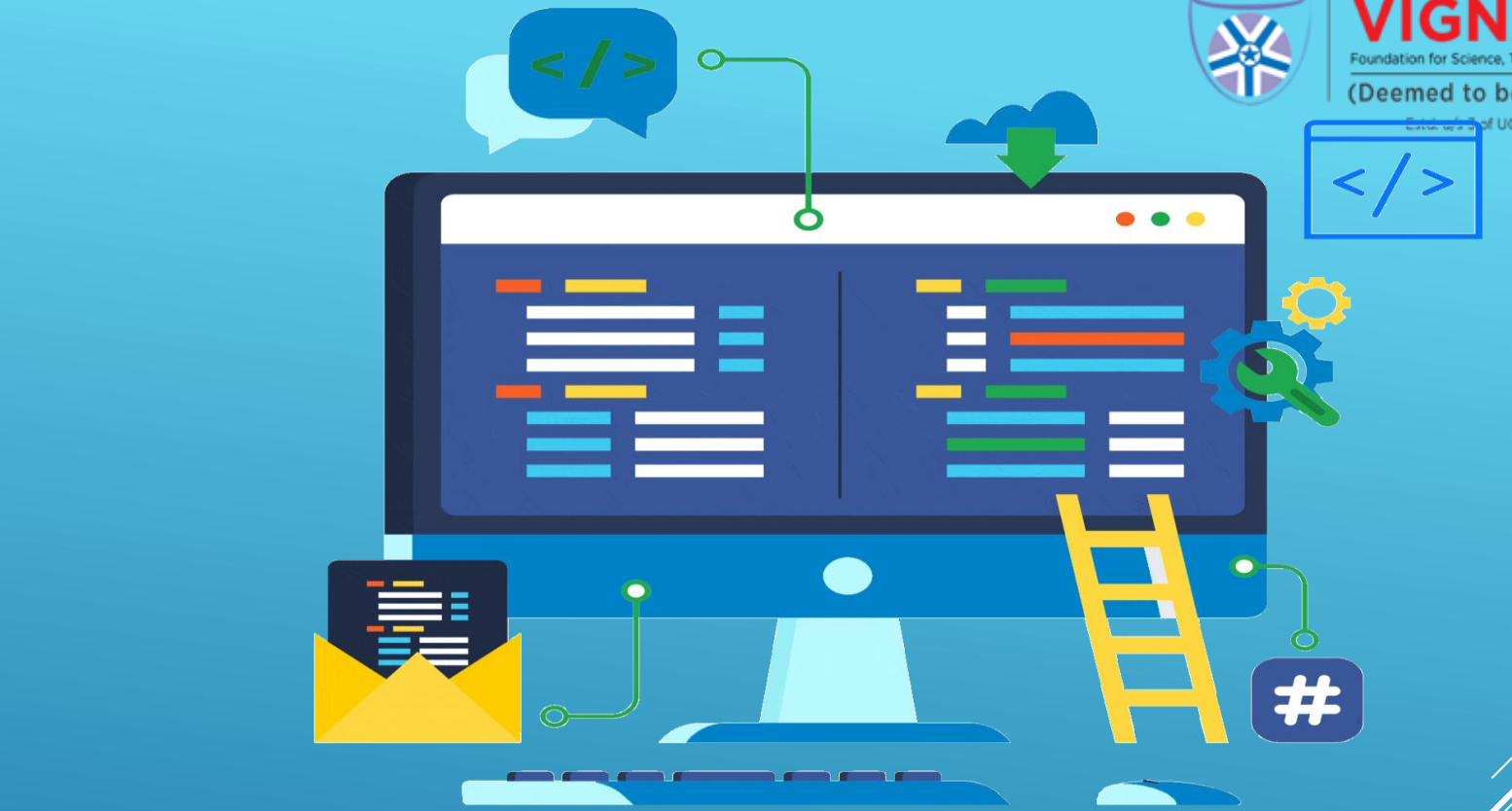
RECAP


Webpage
HTML
CSS
JAVASCRIPT
Content and Structure
Presentation
Behavior

NEXT



JDBC

MR. AMAR JUKUNTLA

ASSISTANT PROFESSOR, CSE

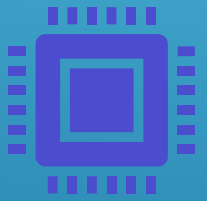
VFSTR DEEMED TO BE UNIVERSITY

INDEX

OBJECTIVES

Understand the reasons for using database.

Understand different types of JDBC connections.



Can design and develop database driven application.

DATABASE

A database is an organized collection of **data**, generally **stored** and **accessed electronically** from a computer system.

Example

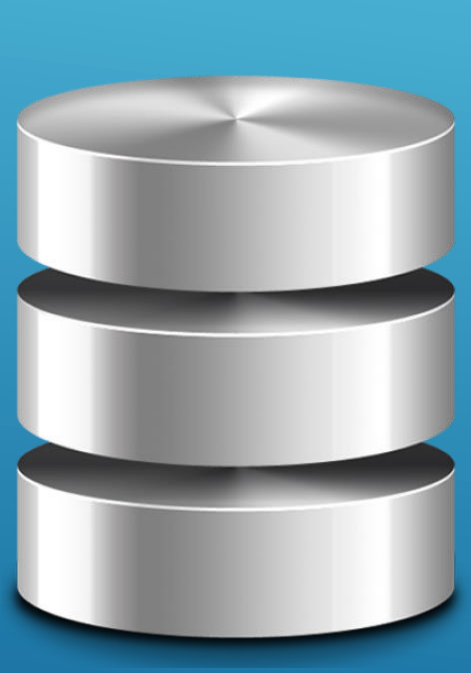
StudentID	Name	Branch	Year	Section
181AY4001	Aman	CSE	3	A
181AY4002	Ahan	CSE	3	A
181AY4003	Amit	CSE	3	A
181AY4004	Yash	CSE	3	B
181AY4005	Buddy	CSE	3	B
181AY4006	Ben	CSE	3	C
191AY4009	John	CSE	2	A

Entities

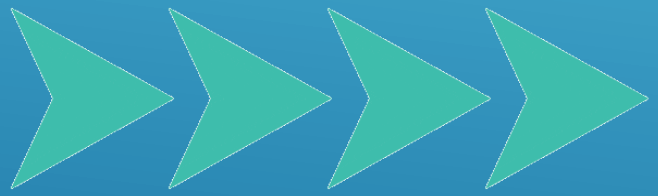
Key Field

Attributes

DATABASE MANAGEMENT SYSTEM



Program

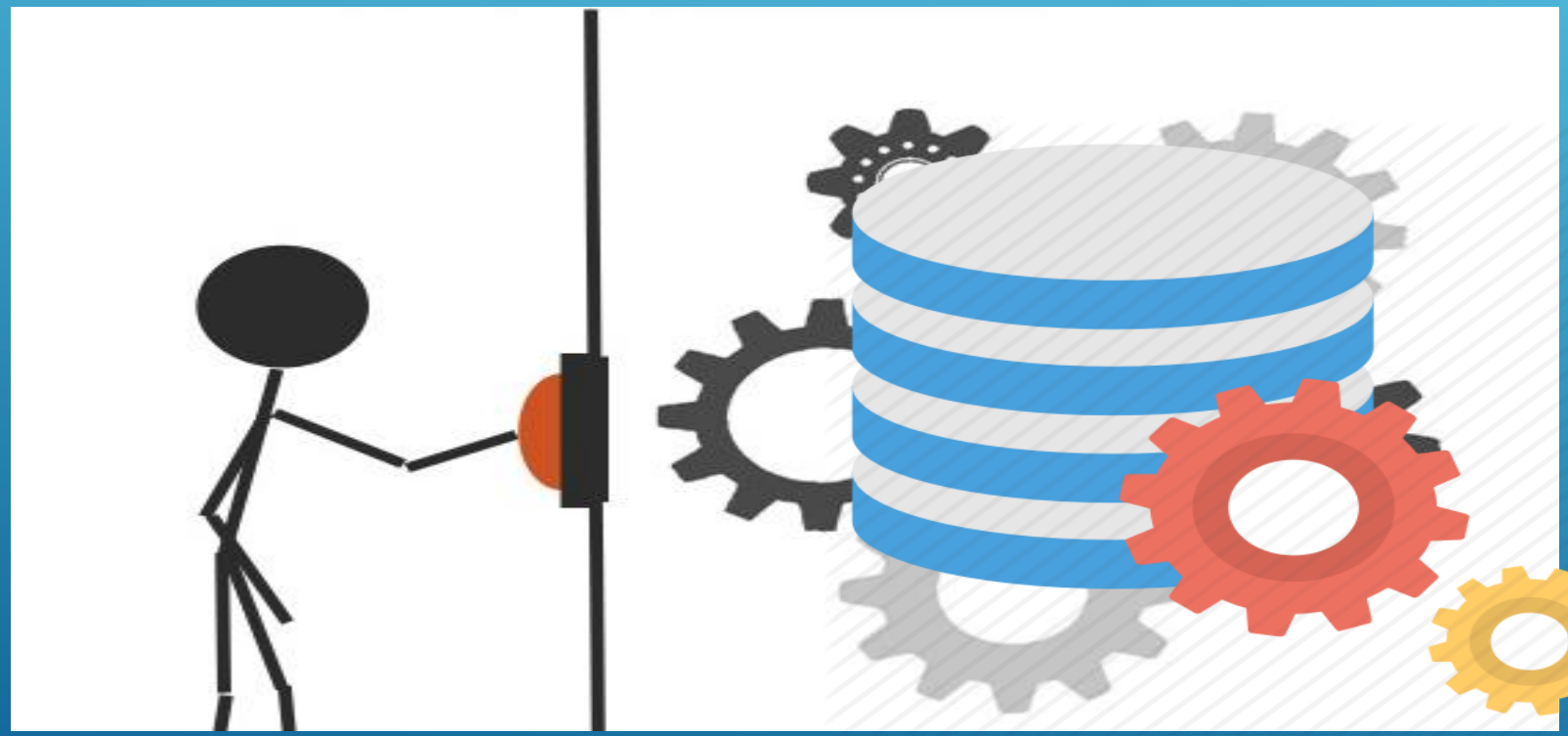


Database

Manipulation

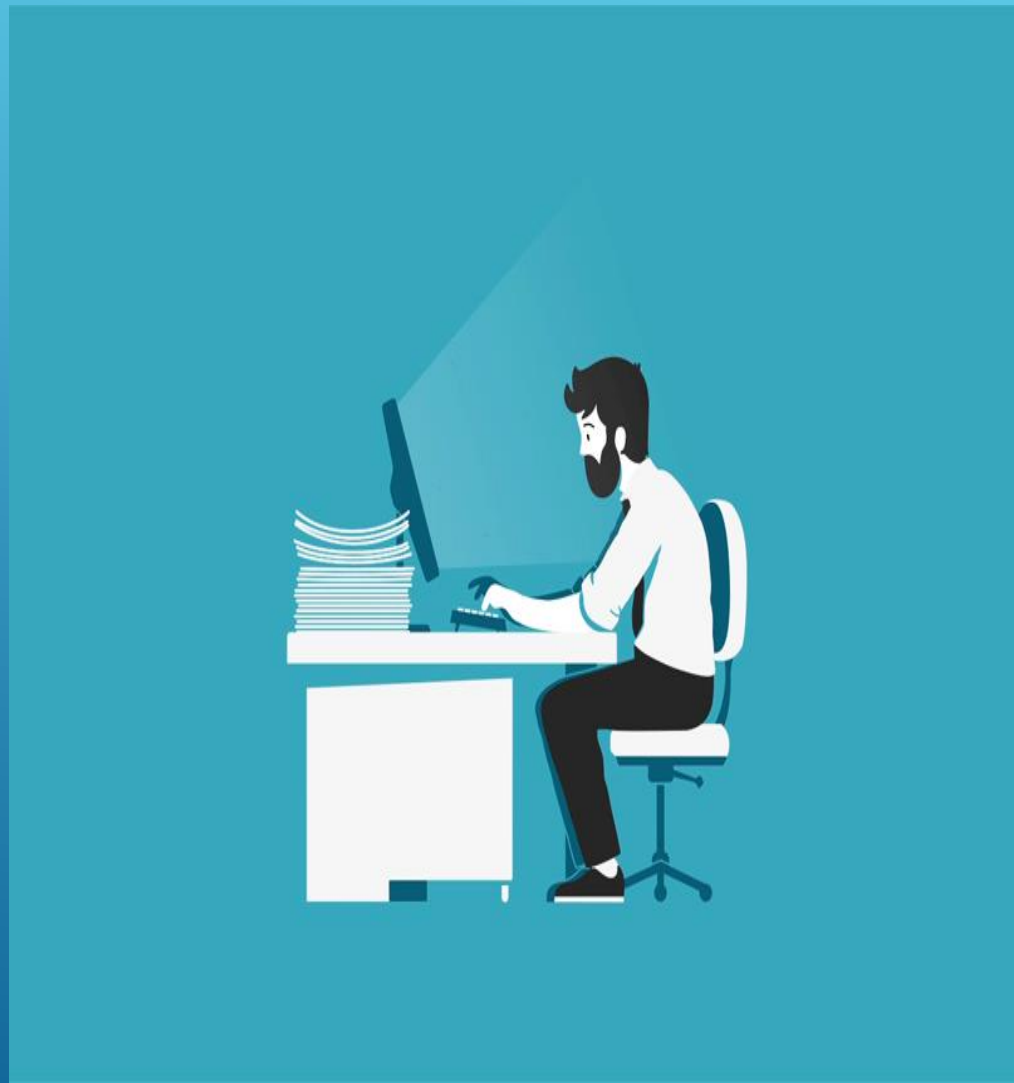
DATABASE MANAGEMENT SYSTEM

DBMS provides an **interface** between user and database



DATABASE MANAGEMENT SYSTEM

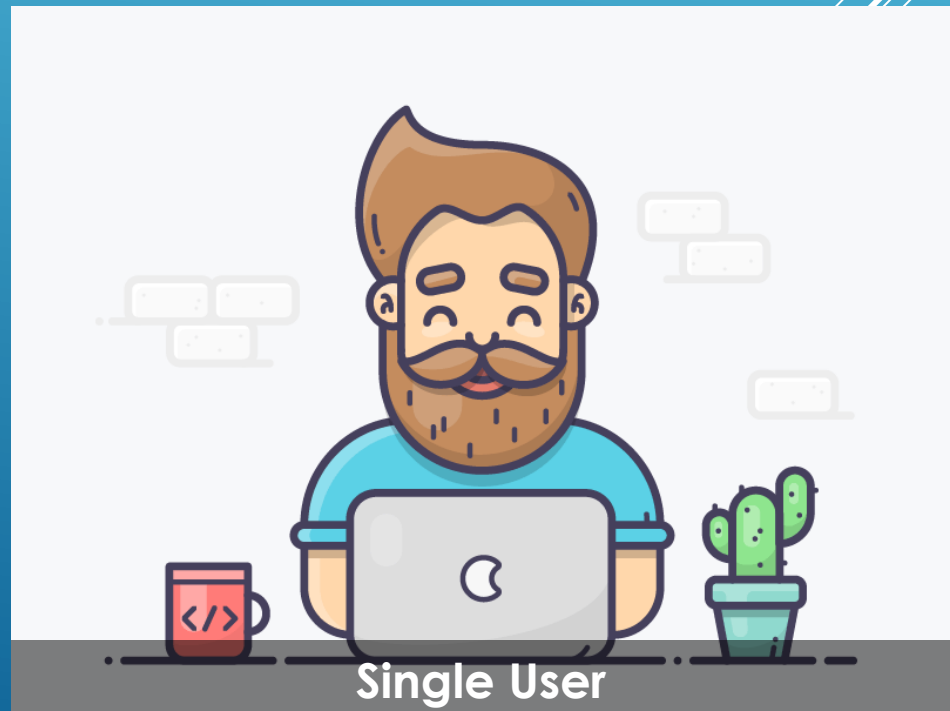
To ensure that data is consistently organized and remain easy to assess



VIGNAN'S
Foundation for Science, Technology & Research
(Deemed to be University)
-Estd. u/s 3 of UGC Act 1956

TYPES OF DATABASE SOFTWARE

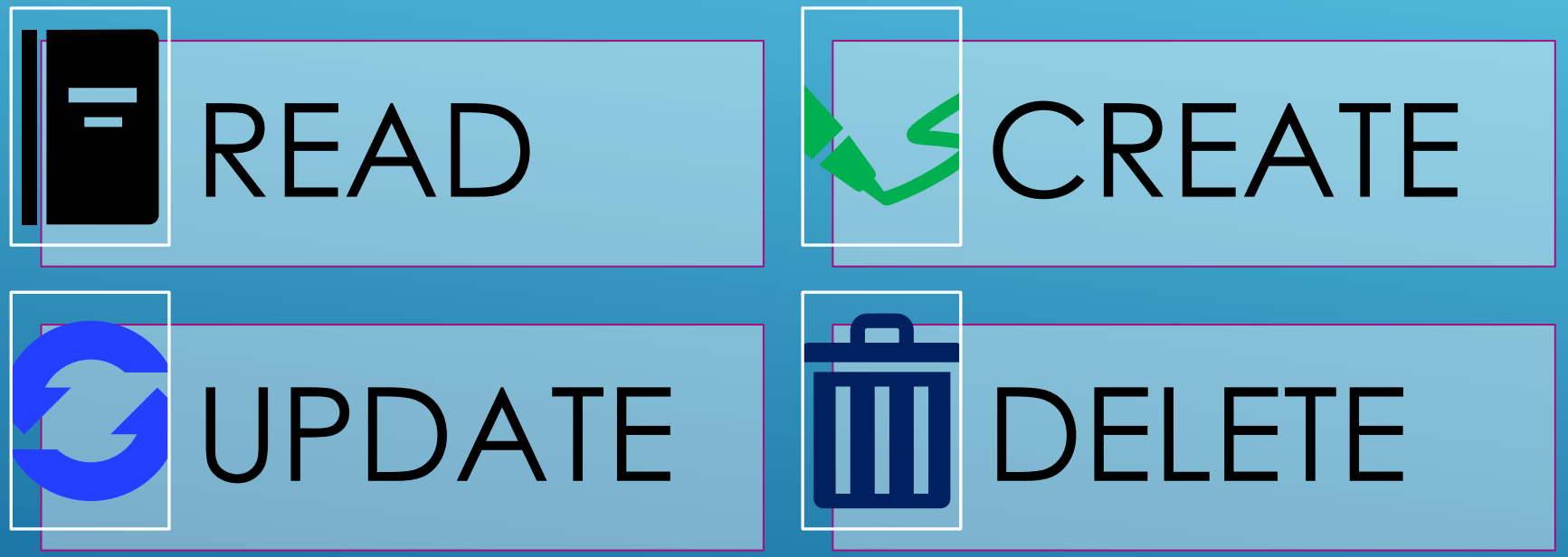
Multiple Users



Single User

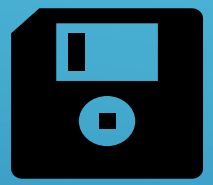
EXAMPLES

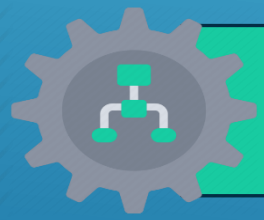
PURPOSE OF DATABASE



DATABASE ADMINISTRATOR

Professionals





VIGNAN'S
Foundation for Science, Technology & Research
(Deemed to be University)
-Estd. u/s 3 of UGC Act 1956

SCOPE OF DBA

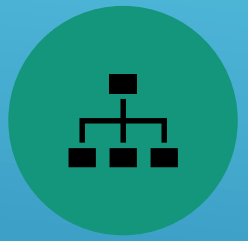
- Capacity Planning
- Database Design
- Migration
- Backup and Data Recovery
- Installation
- Configuration
- Performance monitoring
- Security

DBA directs all activities related to an organization's database.

Without DBA an organization cannot successfully complete most business activity.

APPLICATIONS

Banking and Finance



Business Organization

Education

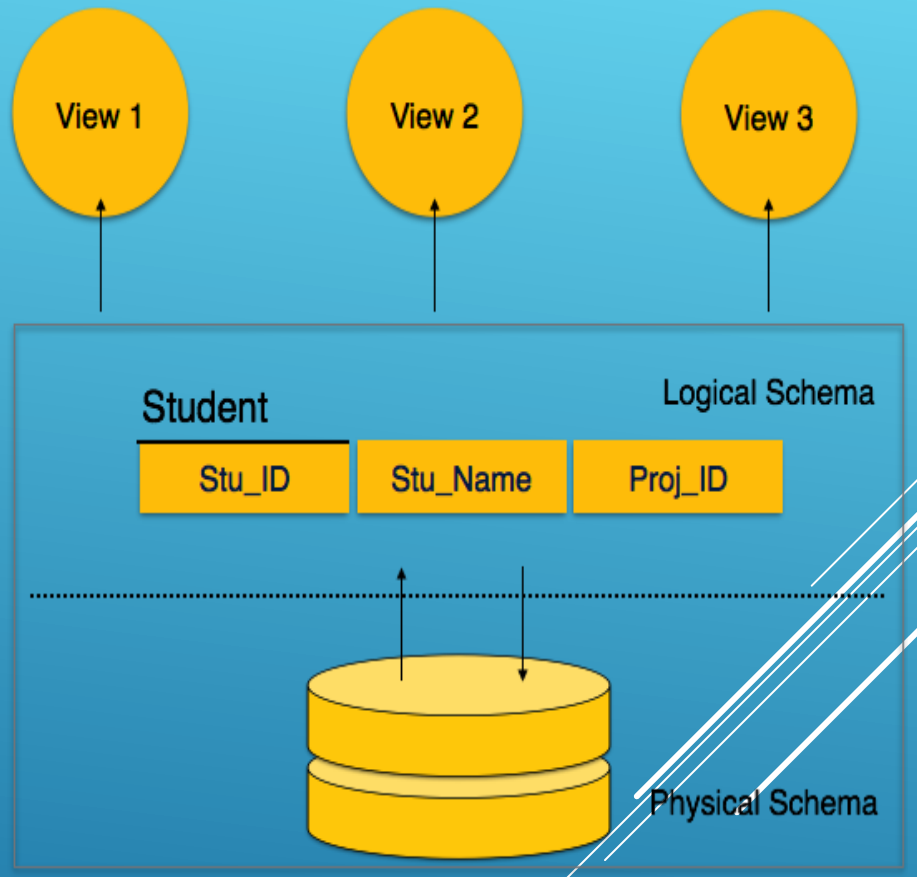
Manufacturing

DATABASE SCHEMA

- Database Schema refers to the **overall structure** of a database.
- A database schema is an **abstraction** used to represent the storage of data in a database.
- It not only describes the organization of data but also represents the **relationship between various tables in a database**.

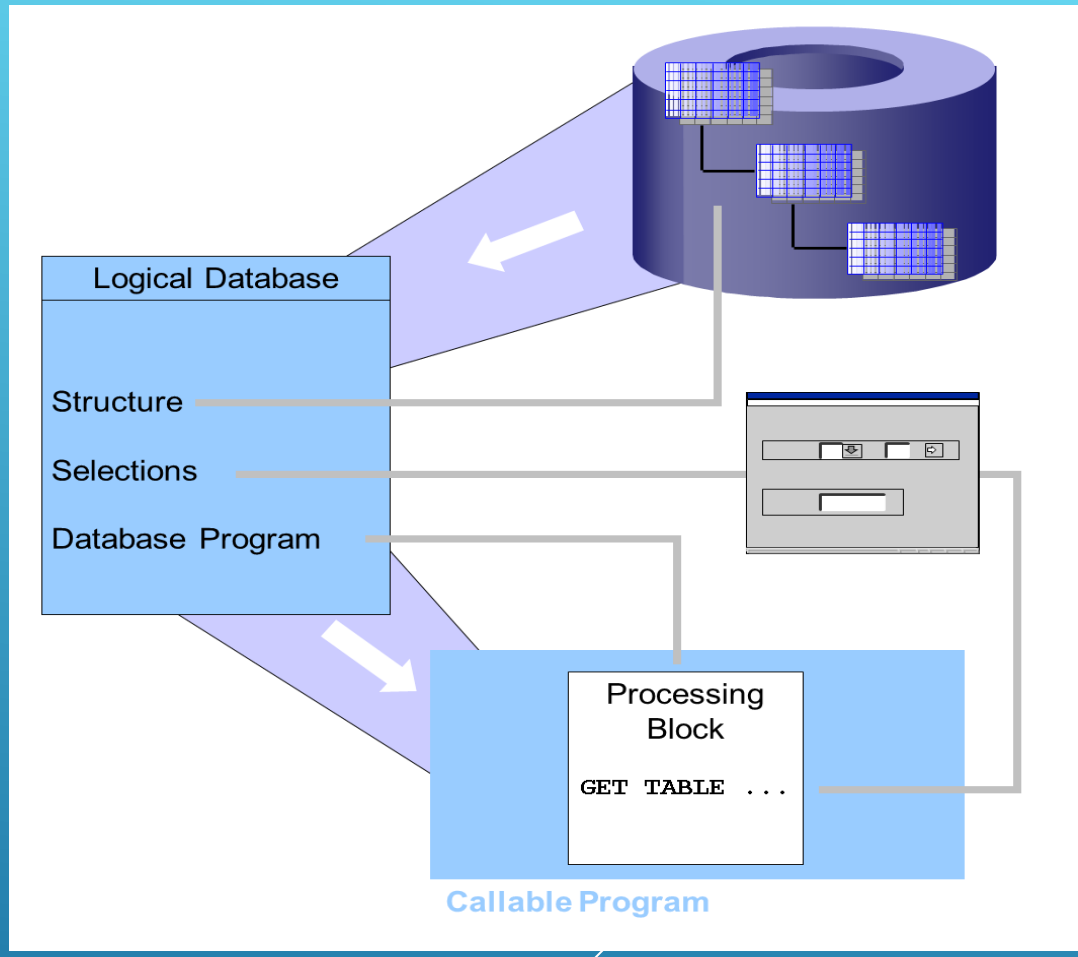
TYPES OF SCHEMAS

- There are two types of database schemas.
  - Logical Database Schema
  - Physical Database Schema



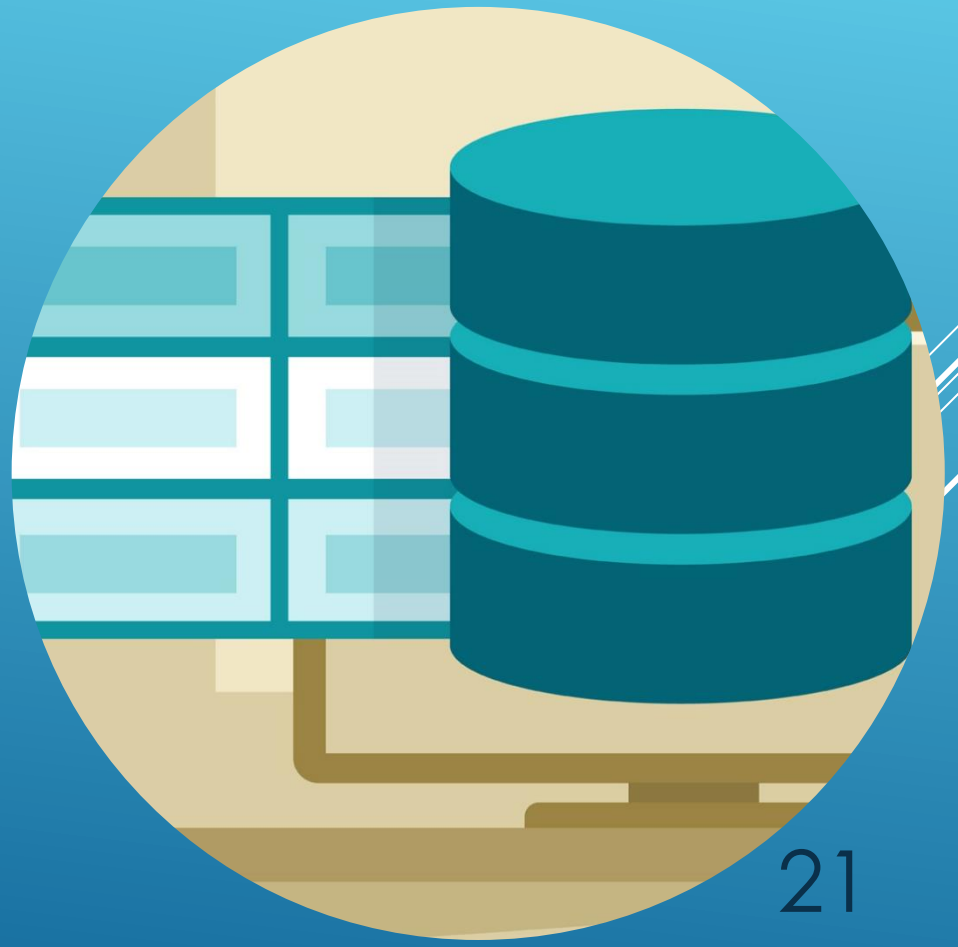
LOGICAL DATABASE SCHEMA

- **This schema defines all the logical constraints that need to be applied on the data stored. It defines tables, views, and integrity constraints.**



PHYSICAL DATABASE SCHEMA

- This schema pertains to the actual storage of data and its form of storage like files, indices, etc. It defines how the data will be stored in a secondary storage.



A BRIEF OVERVIEW OF THE JDBC PROCESS

Java Database Connectivity (**JDBC**) is an API specification for connecting applications written in Java to data in popular databases.

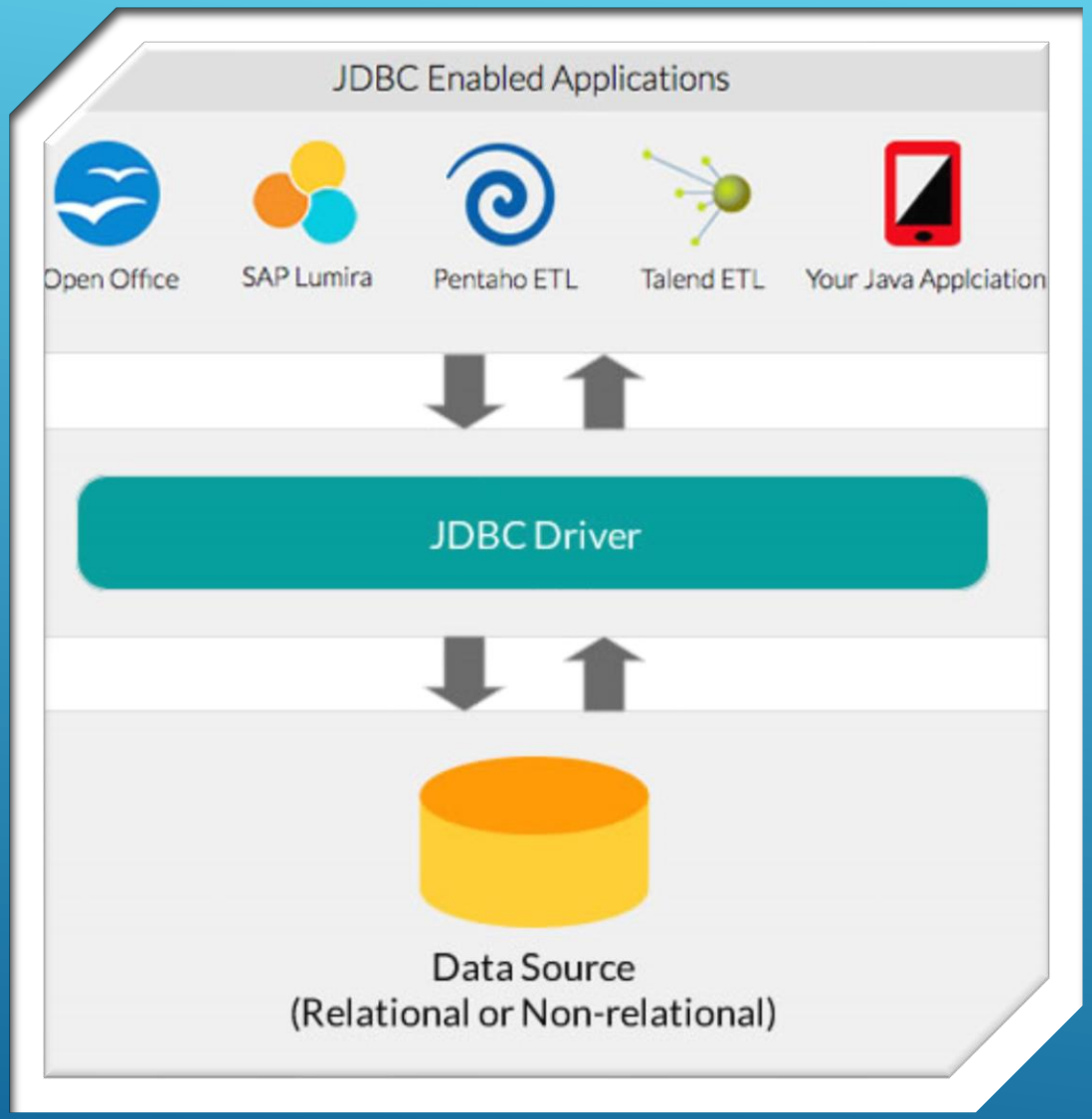
The JDBC API lets you encode access request statements in Structured Query Language (SQL) that are then passed to the application that manages the database.

It returns the results through a similar interface.

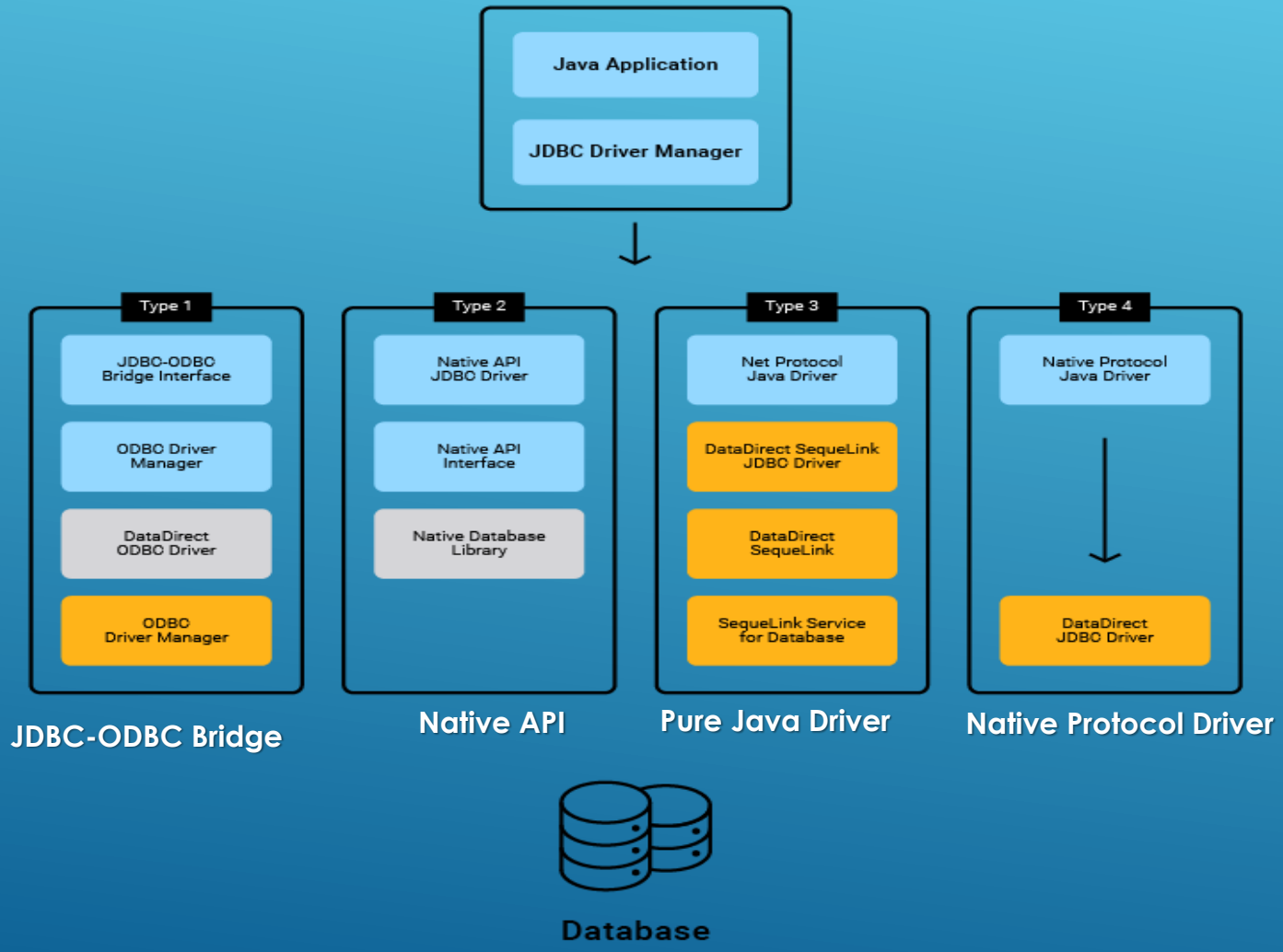
CONTINUE…

- JDBC is an API developed by Sun Microsystems to enable a standards-based method to access data using the Java language.
- As is the case with ODBC, JDBC enables a single JDBC application to access several data sources and can run on any machine with a Java Virtual Machine (JVM).
- The JDBC data standard defines a set of Java interfaces to enable application developer's abstract data access functionality including:
  - Establish Connection with a data source
  - Execute SQL queries
  - Process result sets

JDBC ARCHITECTURE

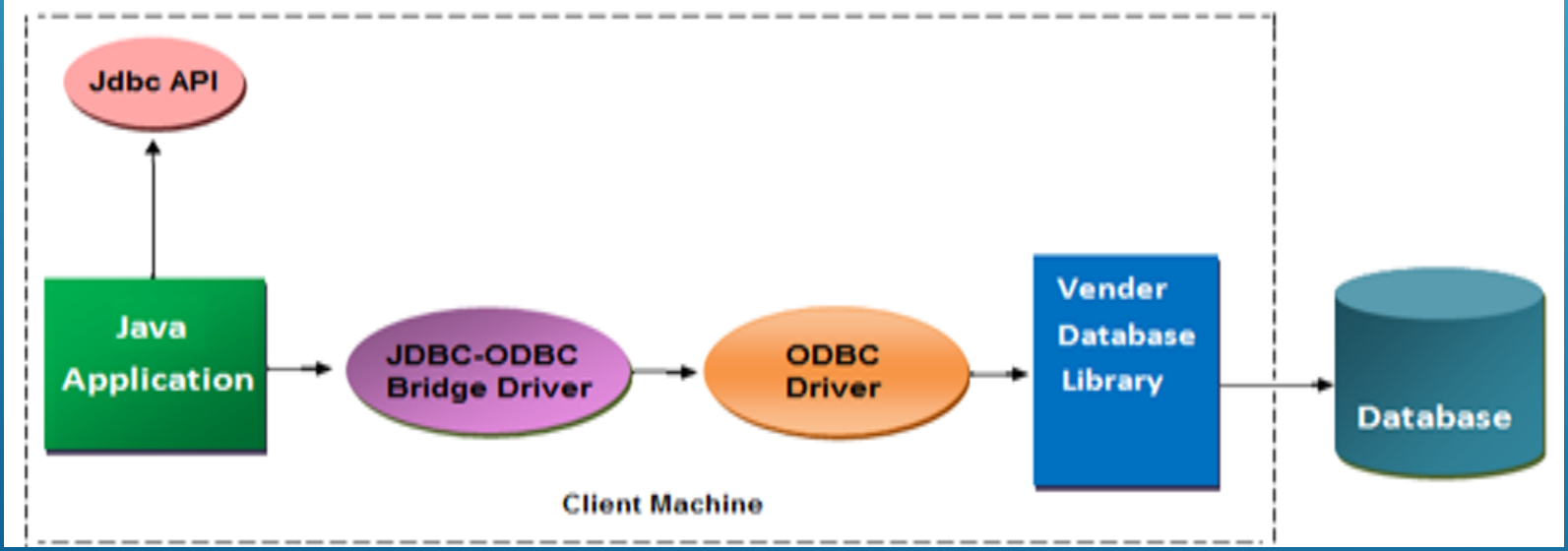


TYPES OF DRIVERS

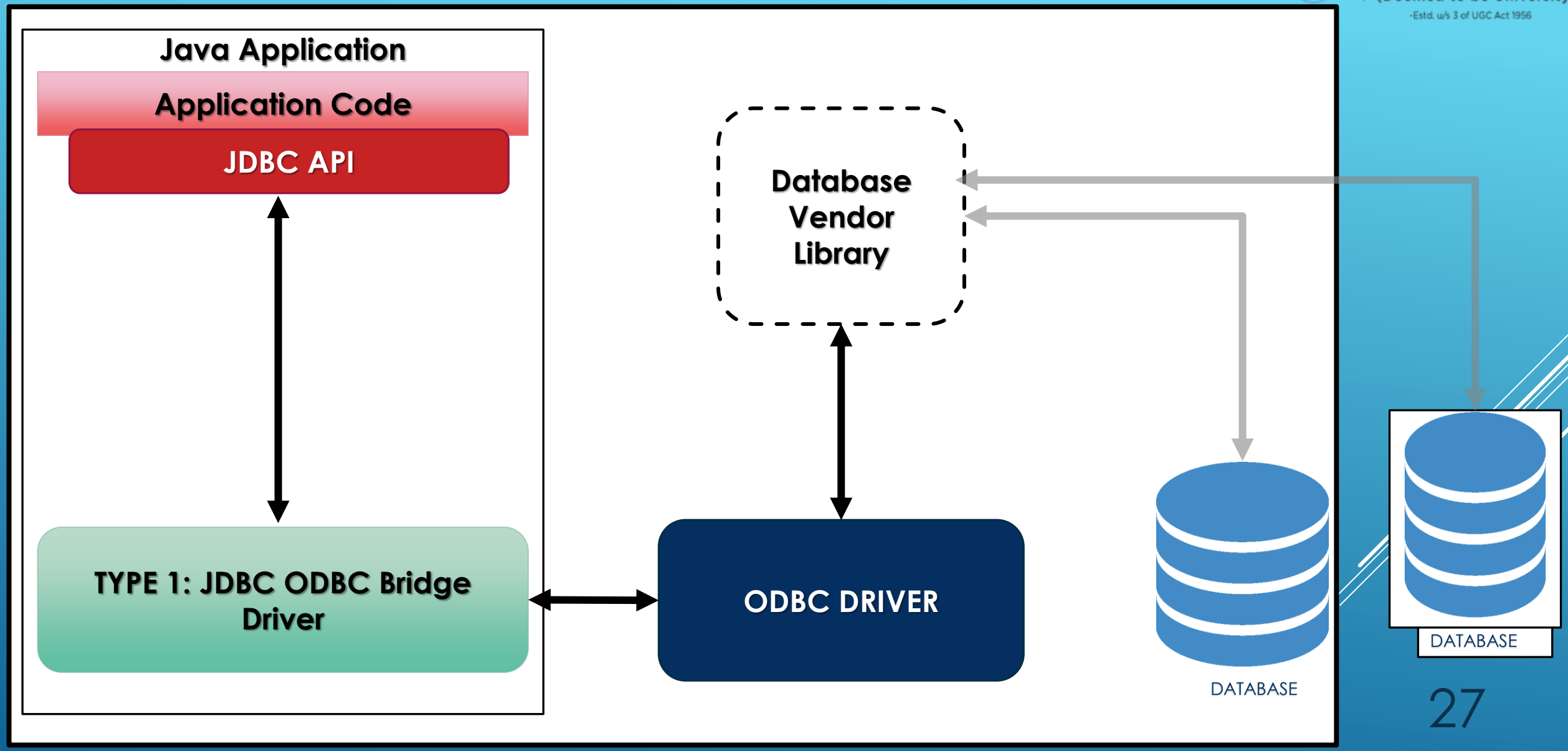


TYPE 1:JDBC-ODBC BRIDGE

- A JDBC-ODBC Bridge provides application developers with a way to access JDBC drivers via the JDBC API.
- Type 1 JDBC drivers translate the JDBC calls into ODBC calls and then sends the calls to the ODBC driver.
- Type 1 JDBC drivers are generally used when the database client libraries need to be loaded on every client machine.



Client Machine



CONTINUE…

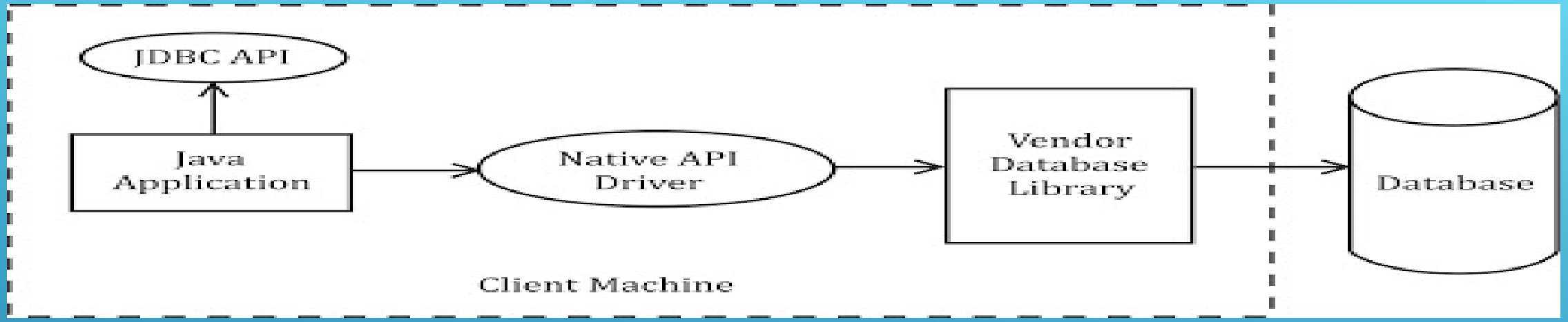
- ODBC driver converts ODBC calls to database specific calls.
- Sun provides a JDBC-ODBC Bridge driver by "sun.jdbc.odbc.JdbcOdbcDriver".
- The driver is a **platform dependent** because it uses ODBC which is depends on native libraries of the operating system.
- For example, ODBC must be **installed** on the computer and the **database** must **support ODBC driver**.
- Type 1 is the **simplest** compare to all other driver but it's a platform specific i.e. only on Microsoft platform.
- The JDBC-ODBC Bridge is used only when there is **no PURE-JAVA** driver available for a database.

Advantages

- Easy to use
- Can be easily connected to any database

Disadvantages

- Performance is **degraded** due to JDBC method is **converted** into the ODBC function calls.
- ODBC driver need to be installed in the client's machine



- Native API driver uses **native API** to connect a java program **directly** to the database.
- Native API id s C, C++ library, which contains a set of function used to connect with database directly.
- Native API will be different from one database to another database. So this Native API driver is a database dependent driver.

TYPE 2:NATIVE API DRIVER

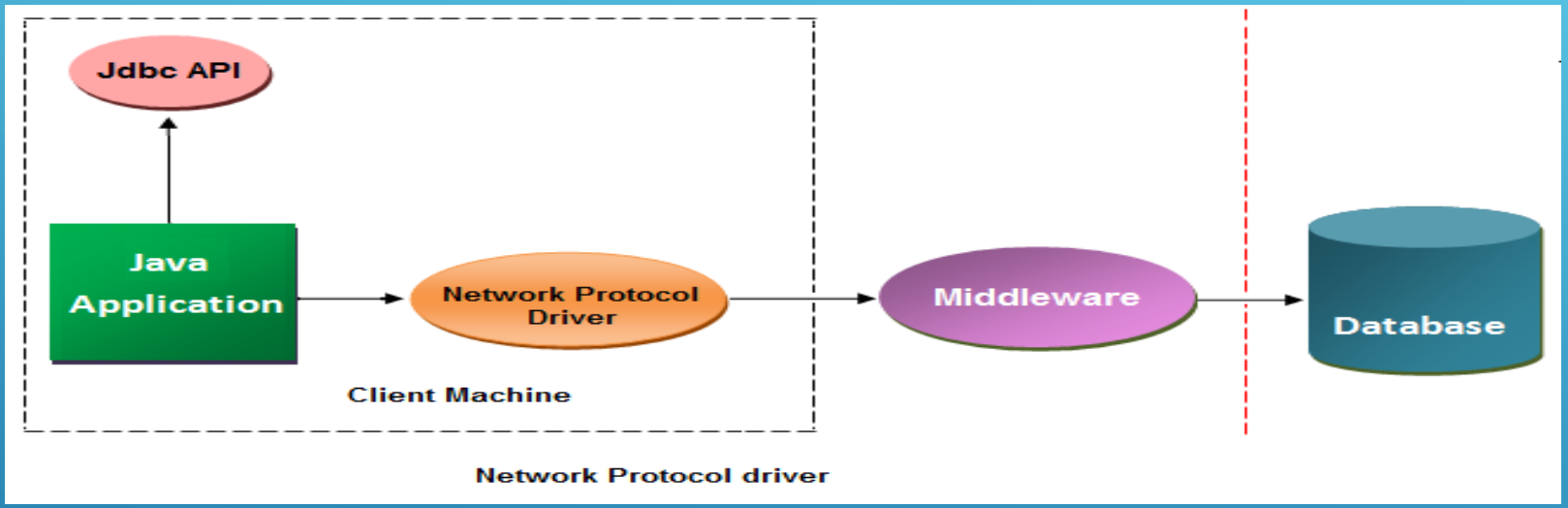
Advantages

- Native API driver comparatively faster than JDBC-ODBC bridge driver.

Disadvantages

- Native API driver is database dependent and also platform dependent because of Native API

TYPE 3: PURE JAVA DRIVER



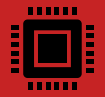
- The **Network Protocol** driver uses middle-ware (application server) that **converts** JDBC calls directly or indirectly into the **vendor-specific database protocol**.
- It is fully written in java, so it is a pure java driver.

It uses three tier architecture for communication.

It can interface multiple databases –Not a vendor specific.

The JDBC Client driver written in java, communicates with a middleware-net-server using a database independent protocol, and then this net server translates this request into *database commands* for that database.

Thus the client driver to middleware communication is **database independent.**



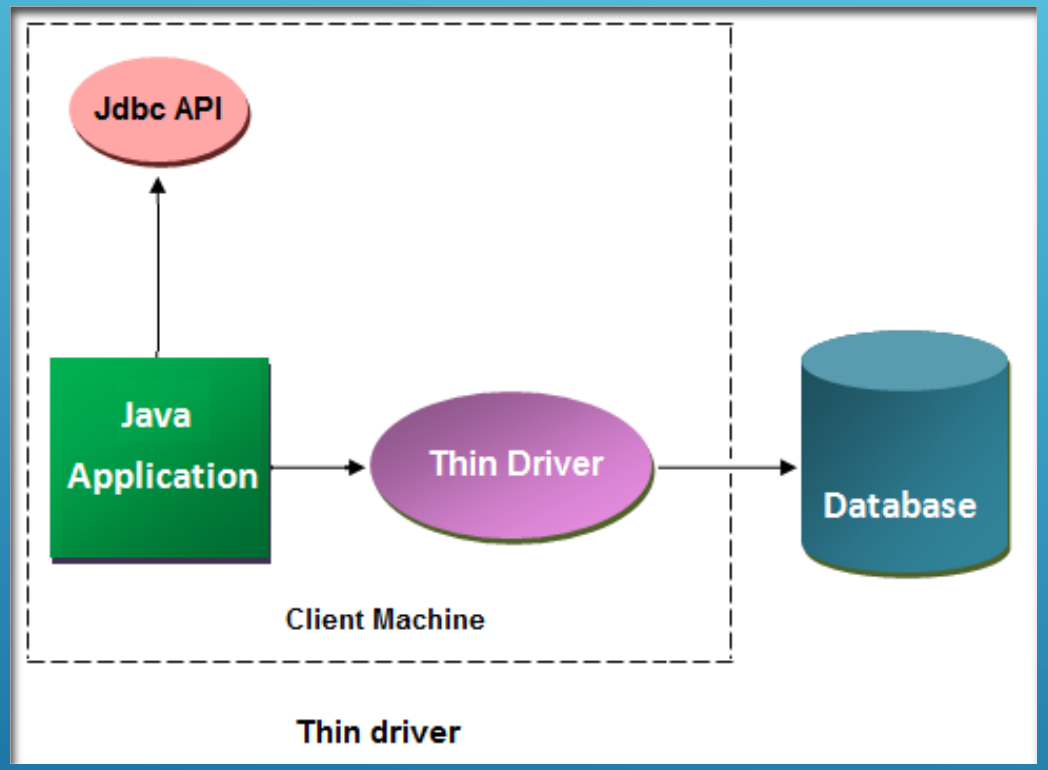
Client -> JDBC Driver -> Network-protocol driver -> Middleware-Net Server -> Any Database.

Advantages

- Platform independent
- We can switch over from one database to another without changing the client-side driver classes, by just changing the configuration of the middleware.
- Easy deployment.

Disadvantages

- Requires database-specific coding to be done in the middle tier



- The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver.
- It is fully written in Java language.

TYPE 4:NETWORK PROTOCOL/THIN DRIVER

Fully written in java, communicates directly with a vendor's database, through socket connections.

No translation or middleware layers are required, improving performance.

The driver converts JDBC calls into the vendor specific database specific protocol so that client applications can communicate directly with the database server.

It is completely implemented in java to achieve platform independence.

Client →Native Protocol JDBC Driver → Database Server

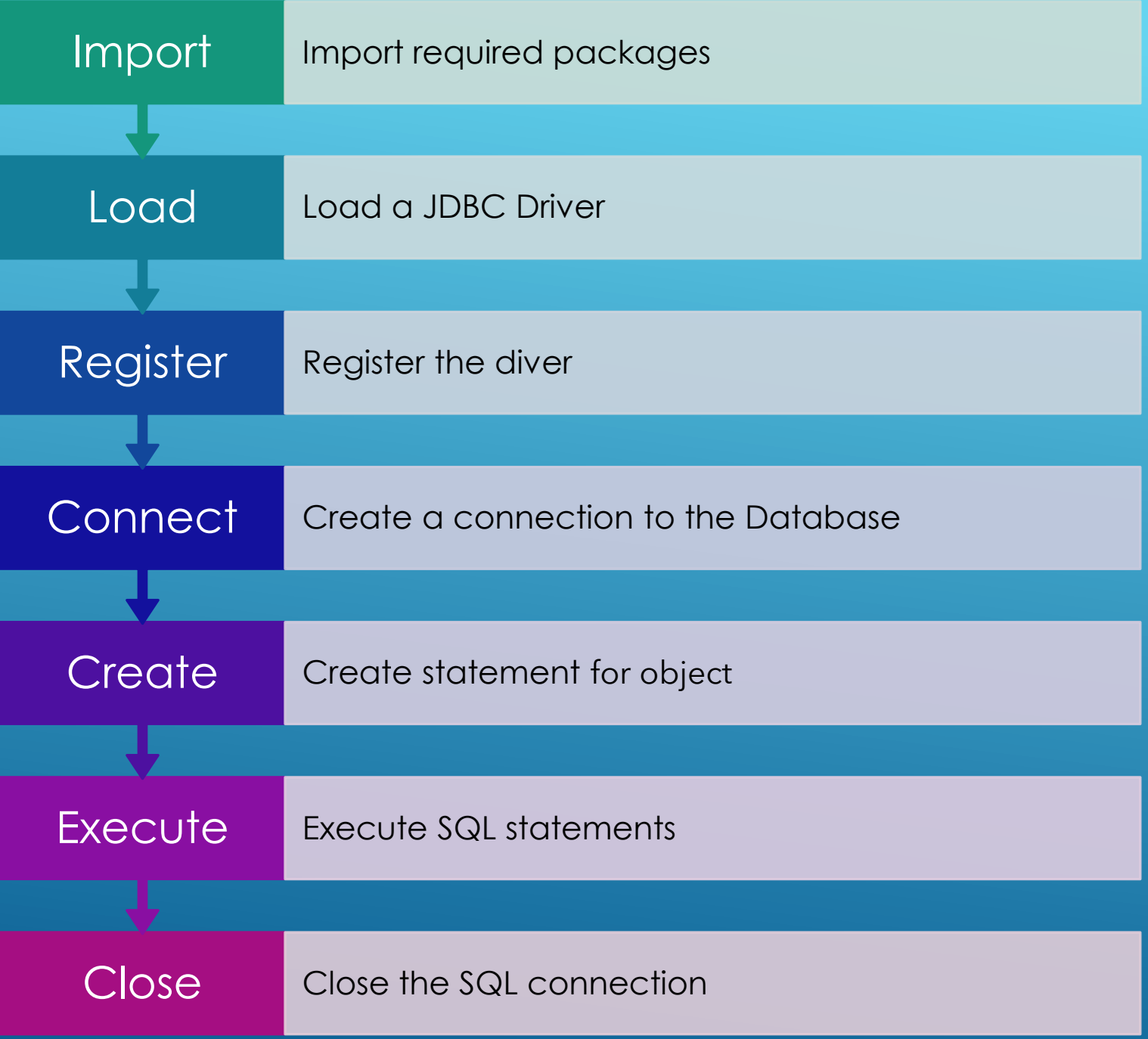
Advantages

- These drivers don't translate the requests into an intermediary format (such as ODBC), nor do they need a middleware layer to service requests.
- This can enhance performance considerably.
- The JVM can manage all aspects of the application-to-database connection; this can facilitate debugging.

Disadvantages

- Drivers are database dependent.

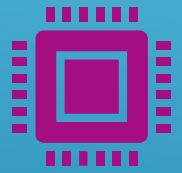
STEPS FOR CONNECTING JAVA APPLICATION TO DATABASE



JDBC PACKAGES

- JDBC contains in two packages
  - java.sql
  - javax.sql

JAVA.SQL PACKAGE



The first package is called **java.sql** and contains core JDBC interfaces of the JDBC API.

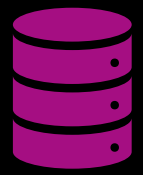
These include the JDBC interfaces that provide the basics for connecting to the DBMS and **interacting with data stored** in the DBMS.

This package include classes and interface to perform almost all JDBC operation such as creating and **executing SQL Queries**.

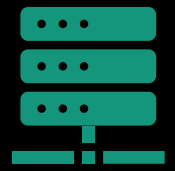
CLASSES AND INTERFACES OF JAVA.SQL PACKAGE

Classes/Interface	Description
java.sql.Connection	creates a connection with specific database
java.sql.CallableStatement	Execute stored procedures
java.sql.Driver	create an instance of a driver with the DriverManager.
java.sql.DriverManager	This class loads the database drivers.
java.sql.PreparedStatement	Used to create and execute parameterized query.
java.sql.ResultSet	It is an interface that provide methods to access the result row-by-row.
java.sql.SQLException	Encapsulate all JDBC related exception.
java.sql.Statement	This interface is used to execute SQL statements

JAVAX.SQL PACKAGE

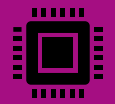


This package is also known as JDBC extension API.



It provides classes and interface to access server-side data

LOAD AND REGISTER A JDBC DRIVER



The JDBC driver must be **loaded before** the Java application can connect to the DBMS.

The **Class.forName()** method is used to load the JDBC driver

Here forName() is a method of class Class.

The driver is loaded by calling the Class.forName() method and passing it the name of the driver

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

CREATE A CONNECTION TO CONNECT DATABASE

- Once the driver is loaded, getConnection() method of DriverManager Class is used to create a connection with the database
- The Java application must connect to the DBMS using the DriverManager.getConnection() method

  Syntax: getConnection(String url, String username, String password)
- URL is the location of database.
- The DriverManager.getConnection() returns a Connection interface that is used throughout the process to reference the database.
- Example:

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

- The next step after the JDBC driver is loaded and a connection is successfully made with a particular database is to send an SQL query to the DBMS for processing.
- The statement object is responsible to execute queries with the database.
- **createStatement()** method of Connection object is used create a Statement object.
- Example:

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

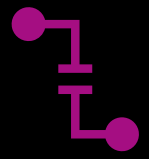
CREATE A STATEMENT FOR OBJECT

EXECUTING A SQL STATEMENT

- The methods of Statement object is used to execute a query and return a ResultSet object that contains the response from the DBMS.
- Methods statement object:
  - **executeQuery()** is used for SELECT statement.
  - **executeUpdate()** is used for create, alter or drop table.
  - **execute()** is used when multiple results may be returned
- **Example**:

```
String query="create table employee1(empid
varchar(20),empname varchar(30),empsal varchar(20))";

stmt.executeUpdate(query);
```

CLOSE THE CONNECTION



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

con.close()

EXAMPLE

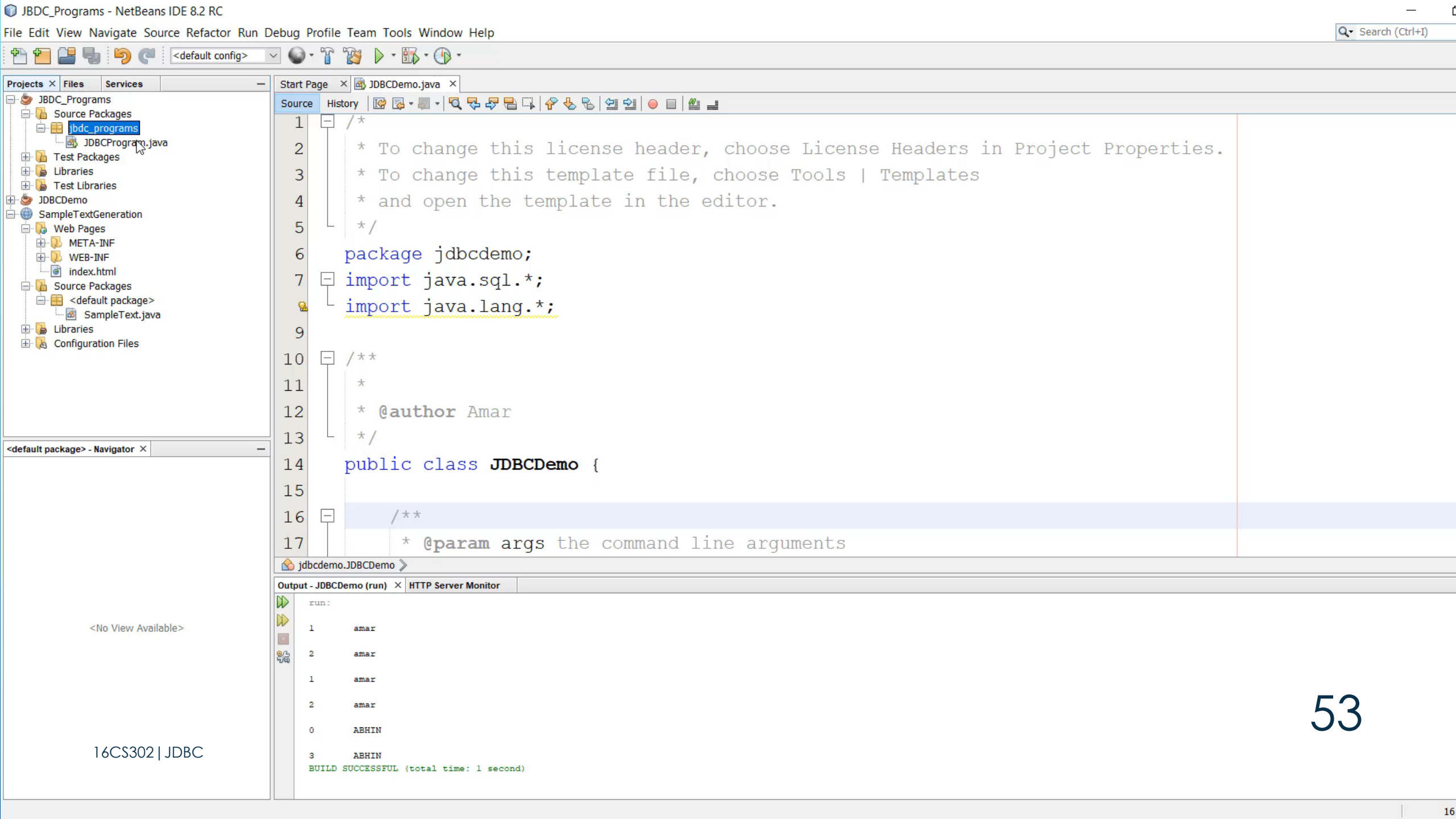
```
import java.sql.*;
public class JDBCProgram
{
    public static void main(String[] args)throws Exception
    {
        try
        {
            Class.forName("com.mysql.jdbc.Driver");
            Connection con=DriverManager.getConnection("jdbc:mysql://localhost/test","root","");
            Statement stmt=con.createStatement();
            ResultSet rs=stmt.executeQuery("select * from Login");
            while(rs.next()){
                    System.out.println("\n"+rs.getInt(1)+"\t"+rs.getString(2)+"\t");
                }
            con.close();
        }catch(Exception e)
        {
            System.out.println("Error:"+e.getMessage());
        }

    }
}
```

- A Java application does not directly connect to a DBMS.
- Instead, the Java application connects with the JDBC driver that is associated with the DBMS.
- However, before this connection is made, the JDBC driver must be loaded and registered with the DriverManager.
- The purpose of loading and registering the JDBC driver is to bring the JDBC driver into the Java Virtual Machine (JVM).
- The Class.forName() is used to load the JDBC driver.
- The Class.forName() throws a ClassNotFoundException if an error occurs when loading the JDBC driver.

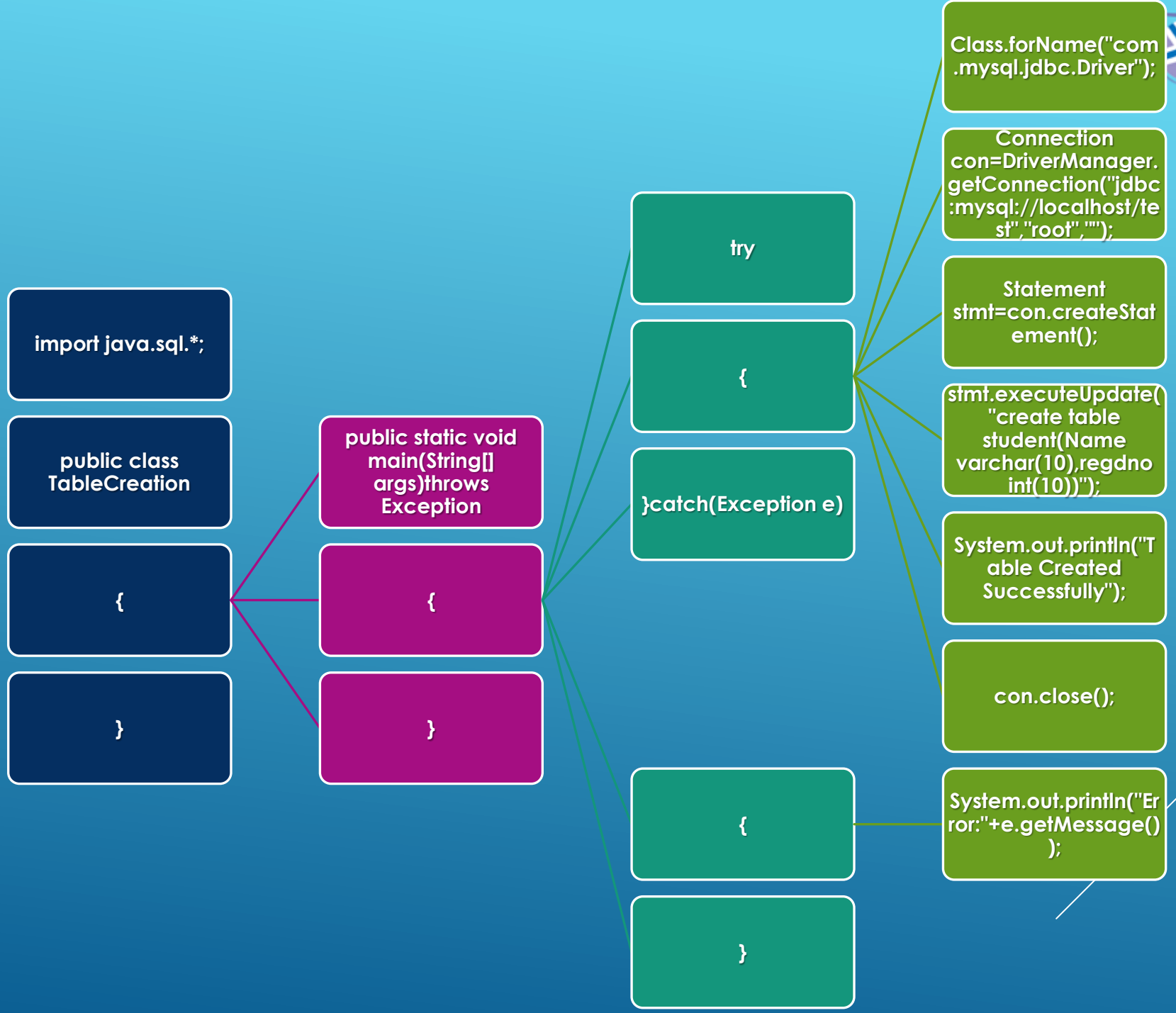
DATABASE CONNECTION

- The connection to the database is established by using one of three getConnection() methods of the DriverManager object.
- The getConnection() method requests access to the database from the DBMS.
- It is up to the DBMS to grant or reject access.
- A Connection object is returned by the getConnection() method if access is granted, otherwise the getConnection() method throws an SQLException.

CONTINUE…


JBDC_Programs - NetBeans IDE 8.2 RC
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
<default config>
Projects
Files
Services
JBDC_Programs
Source Packages
jbdc_programs
JDBCProgram.java
Test Packages
Libraries
Test Libraries
JDBCDemo
SampleTextGeneration
Web Pages
META-INF
WEB-INF
index.html
Source Packages
<default package>
SampleText.java
Libraries
Configuration Files
<default package> - Navigator
<No View Available>
Start Page
JDBCDemo.java
Source
History
1 /*
2 * To change this license header, choose License Headers in Project Properties.
3 * To change this template file, choose Tools | Templates
4 * and open the template in the editor.
5 */
6 package jdbcdemo;
7 import java.sql.*;
8 import java.lang.*;
9
10 /**
11 *
12 * @author Amar
13 */
14 public class JDBCDemo {
15
16 /**
17 * @param args the command line arguments
jdbcdemo.JDBCDemo
Output - JDBCDemo (run)
HTTP Server Monitor
run:
1 amar
2 amar
1 amar
2 amar
0 ABHIN
3 ABHIN
BUILD SUCCESSFUL (total time: 1 second)

CREATING A TABLE


import java.sql.*;
public class TableCreation
{
}
public static void main(String[] args)throws Exception
{
}
try
{
}catch(Exception e)
{
}
Class.forName("com.mysql.jdbc.Driver");
Connection con=DriverManager.getConnection("jdbc:mysql://localhost/test","root","");
Statement stmt=con.createStatement();
stmt.executeUpdate("create table student(Name varchar(10),regdno int(10))");
System.out.println("Table Created Successfully");
con.close();
System.out.println("Error:"+e.getMessage());

INSERTION

```
import java.sql.*;
public class TableInsertion
{
	public static void main(String[] args)throws Exception
	{
		try
		{
			Class.forName("com.mysql.jdbc.Driver");
			Connection con=DriverManager.getConnection("jdbc:mysql://localhost/test","root","");
			Statement stmt=con.createStatement();
			stmt.executeUpdate("insert into student values('ABHIN',1)");
			System.out.println("Table Insertion Successfully");
			con.close();
		}catch(Exception e)
		{
			System.out.println("Error:"+e.getMessage());
		}

	}
}
```

UPDATION

```
import java.sql.*;
public class TableUpdation
{
    public static void main(String[] args)throws Exception
    {
        try
        {
            Class.forName("com.mysql.jdbc.Driver");
            Connection con=DriverManager.getConnection("jdbc:mysql://localhost/test","root","");
            Statement stmt=con.createStatement();
            stmt.executeUpdate("update student set Name='Abhi' where Name='ABHIN'");
            System.out.println("Table Updation Successfully");
            con.close();
        }catch(Exception e)
        {
            System.out.println("Error:"+e.getMessage());
        }

    }
}
```

```
C:\Users\amarj\Documents>java TableUpdation
Table Updation Successfully
```

DELETION

```
import java.sql.*;
public class TableDeletion
{
	public static void main(String[] args)throws Exception
	{
		try
		{
			Class.forName("com.mysql.jdbc.Driver");
			Connection con=DriverManager.getConnection("jdbc:mysql://localhost/test","root","");
			Statement stmt=con.createStatement();
			stmt.executeUpdate("delete from student where Name='Ahan'");
			System.out.println("Table Deletion Successfull");
			con.close();
		}catch(Exception e)
		{
			System.out.println("Error:"+e.getMessage());
		}

	}
}
```

```
C:\Users\amarj\Documents>java TableDeletion
Table Deletion Successfull
```

RESULTSET

RESULTSET

The ResultSet uses a virtual cursor to point to a row of the virtual table.

The virtual cursor is positioned above the first row of data when the ResultSet is returned by the executeQuery() method.

The virtual cursor moved to the next row using the next() method.

next() method returns a boolean true if the row contains data, otherwise a boolean false is returned, indicating that no more rows exist in the ResultSet.

- Once the virtual cursor points to a row, the getXXX() method is used to copy data from the row to a variable.
- For example, the getString() method is used to copy String data from a column of the ResultSet.
- The getXXX() method requires one parameter, which is an integer that represents the number of the column that contains the data.
- For example, getString(1) copies the data from the first column of the ResultSet.

CONTINUE…

SCROLLABLE RESULT SET

- Until the release of JDBC 2.1 API, the virtual cursor could only be moved down the ResultSet object.
- But today the virtual cursor can be moved backwards or even positioned at a specific row.
- The first() method moves the virtual cursor to the first row in the ResultSet.
- The last() method positions the virtual cursor at the last row in the ResultSet.
- The previous() method moves the virtual cursor to the previous row.
- The absolute() method positions the virtual cursor at the row number specified by the integer passed as a parameter to the absolute() method.
- The getRow() method returns an integer that represents the number of the current row in the ResultSet.

- The Statement object that is created using the createStatement() of the Connection object must be set up to handle a scrollable ResultSet by passing the createStatement() method one of three constants.
- These constants are TYPE_FORWARD_ONLY,TYPE_SCROLL_INSENSITIVE, and TYPE_SCROLL_SENSITIVE.
- The TYPE_FORWARD_ONLY constant restricts the virtual cursor to downward movement.
- TYPE_SCROLL_INSENSITIVE and TYPE_SCROLL_SENSITIVE constants permit the virtual cursor to move in both directions.

CONTINUE…

STATEMENT OBJECTS

- The statement object is responsible to execute queries with the database.
- There are 3 types of Statement Objects
  - Statement
  - Prepared Statement
  - Callable Statement
- Any One of the three types of Statement objects is used to execute the query.
- The Statement object is used whenever a Java application needs to execute a query immediately without first having the query compiled.

CONTINUE…

- The statement object is responsible to execute queries with the database.
- createStatement() method of Connection object is used create a Statement object.
- **Example;**

Statement stmt=con.createStatement();

- The Statement interface provides three different methods for executing SQL statements, executeQuery(), executeUpdate(), and execute().
- The method executeQuery(String sql) is designed for SELECT statements that returns result set object

- The method executeUpdate(String sql) is used to execute INSERT, UPDATE, DELETE statements and also SQL DDL statements like CREATE and DROP .
- The return value of executeUpdate() is an integer indicating the number of rows that were affected.
- The method execute(String sql) is used to execute statements that return more than one result set, more than one update count, or a combination of the two

CONTINUE…

PREPARED STATEMENT OBJECT

PREPARED STATEMENT OBJECT

- A prepared statement is a feature used to execute the same (or similar) SQL statements repeatedly with high efficiency.
- Use the when you plan to use the SQL statements many times. The PreparedStatement interface accepts input parameters at runtime.
- It is used to execute parameterized query.
- The query is constructed similar to queries but a question mark is used as a placeholder for a value that is inserted into the query after the query is compiled.

insert into emp values(?,?,?)

- It is the value that changes each time the query is executed.

The preparedStatement() method of the Connection object is called to return the PreparedStatement object.

The preparedStatement() method is passed the query that is then precompiled.

```
PreparedStatement ps=con.prepareStatement("DELETE FROM employee
WHERE esal = ?");
```

The setXXX() method of the PreparedStatement object is used to replace the question mark with the value passed to the setXXX() method.

CONTINUE…

CALLABLE STATEMENT

- CallableStatement interface is used to **call** the **stored procedures and functions**.
- A stored procedure is a block of code and is identified by a unique name.
- The type and style of code depend on the DBMS vendor and can be written in PL/SQL, TransactSQL, C, or another programming language.
- The stored procedure is executed by invoking the name of the stored procedure.
- The CallableStatement object uses three types of parameters when calling a stored procedure.

CALLABLE STATEMENT

- These parameters are IN, OUT, and INOUT.
- The IN parameter contains any data that needs to be passed to the stored procedure and whose value is assigned using the setXXX() method
- The OUT parameter contains the value returned by the stored procedures
- The INOUT parameter is a single parameter used for both passing information to the stored procedure and retrieving information from a stored procedure

CONTINUE…

REFERENCES

- Google
- TutorialPoints
- JavaTpoint
- W3Schools
- Slideshare