R.M.K GROUP OF ENGINEERING INSTITUTIONS

R.M.K
GROUP OF
INSTITUTIONS

R.M.K GROUP OF INSTITUTIONS

R.M.K GROUP OF INSTITUTIONS

22CS202

Java Programming

Department: Computer Science and Business Systems

Batch/Year: 2022-2026 / I Year

Created by:

Er. C.M. Varun Assistant Professor / CSBS

Date: 15.03.2023

Table of Contents

R.M.K
GROUP OF
INSTITUTIONS

Course Objectives

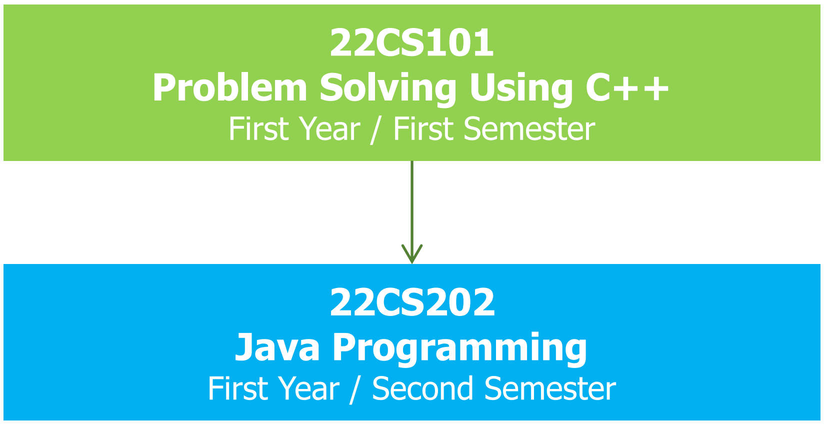
Course Objective

The Course will enable learners to:

- To explain object oriented programming concepts and fundamentals of Java
- To apply the principles of packages, interfaces and exceptions
- To develop a Java application with I/O streams, threads and generic programming
- To build applications using strings and collections.
- To apply the JDBC concepts

Pre Requisites

Prerequisites


22CS101
Problem Solving Using C++
First Year / First Semester
22CS202
Java Programming
First Year / Second Semester

Syllabus

Syllabus

22CS202 Java Programming

L	T	P	C
3	0	0	3

UNIT I JAVA FUNDAMENTALS **9**

An Overview of Java - Data Types, Variables, and Arrays – Operators - Control Statements – Class Fundamentals – Declaring objects – Methods – Constructors – this keyword - Overloading methods - Overloading constructors - Access Control – Static – Final..

UNIT II INHERITANCE, INTERFACES AND EXCEPTION HANDLING **9**

Inheritance: Inheritance basics, Using super, Method Overriding, Using Abstract Classes, Using final with Inheritance - Package and Interfaces: Packages, Packages and member access, Importing Packages, Interfaces, Static Methods in an Interface – Exception Handling: Exception-Handling Fundamentals, Exception Types, Uncaught Exceptions, Using try and catch, Multiple catch Clauses, Nested try Statements, throw, throws, finally, Java's Built-in Exceptions.

UNIT III MULTITHREADING, I/O AND GENERIC PROGRAMMING **9**

Multithreaded Programming: Creating a Thread, Thread Priorities, Synchronization, Interthread Communication – I/O: I/O Basics, Reading Console Input, Writing Console Output, Reading and Writing Files – Generics: Introduction, Generic class, Bounded Types, Generic Methods, Generic Interfaces, Generic Restrictions.

UNIT IV STRING HANDLING AND COLLECTIONS **9**

Lambda Expressions - String Handling – Collections: The Collection Interfaces, The Collection Classes – Iterator – Map - Regular Expression Processing.

UNIT V JDBC CONNECTIVITY **9**

JDBC – DataSource, Configurations, Connection, Connection Pools, Driver Types, ResultSet, Prepared Statement, Named Parameter, Embedded SQL (Insert, Update, Delete, Join, union etc), ResultSet Navigation, Connection Close and Clean up.

Course Outcomes

Course Outcomes

On completion of the course, students will be able to:

CO1: Understand the object oriented programming concepts and fundamentals of Java.

CO2: Develop Java programs with the packages, interfaces and exceptions.

CO3: Build Java applications with I/O streams, threads and generics programming.

CO4: Apply strings and collections in developing applications.

CO5: Implement the concepts of JDBC.

CO – PO/PSO Mapping

CO-PO/PSO Mapping

COs	POs/PSOs														
	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1	3	3	3	-	-	-	-	-	-	-	-	-	3	2	2
CO2	3	2	2	-	-	-	-	-	-	-	-	-	3	3	2
CO3	3	2	2	-	-	-	-	-	-	-	-	-	3	3	2
CO4	3	2	2	-	-	-	-	-	-	-	-	-	3	3	2
CO5	3	2	2	-	-	-	-	-	-	-	-	-	3	3	2

Lecture Plan

Unit V

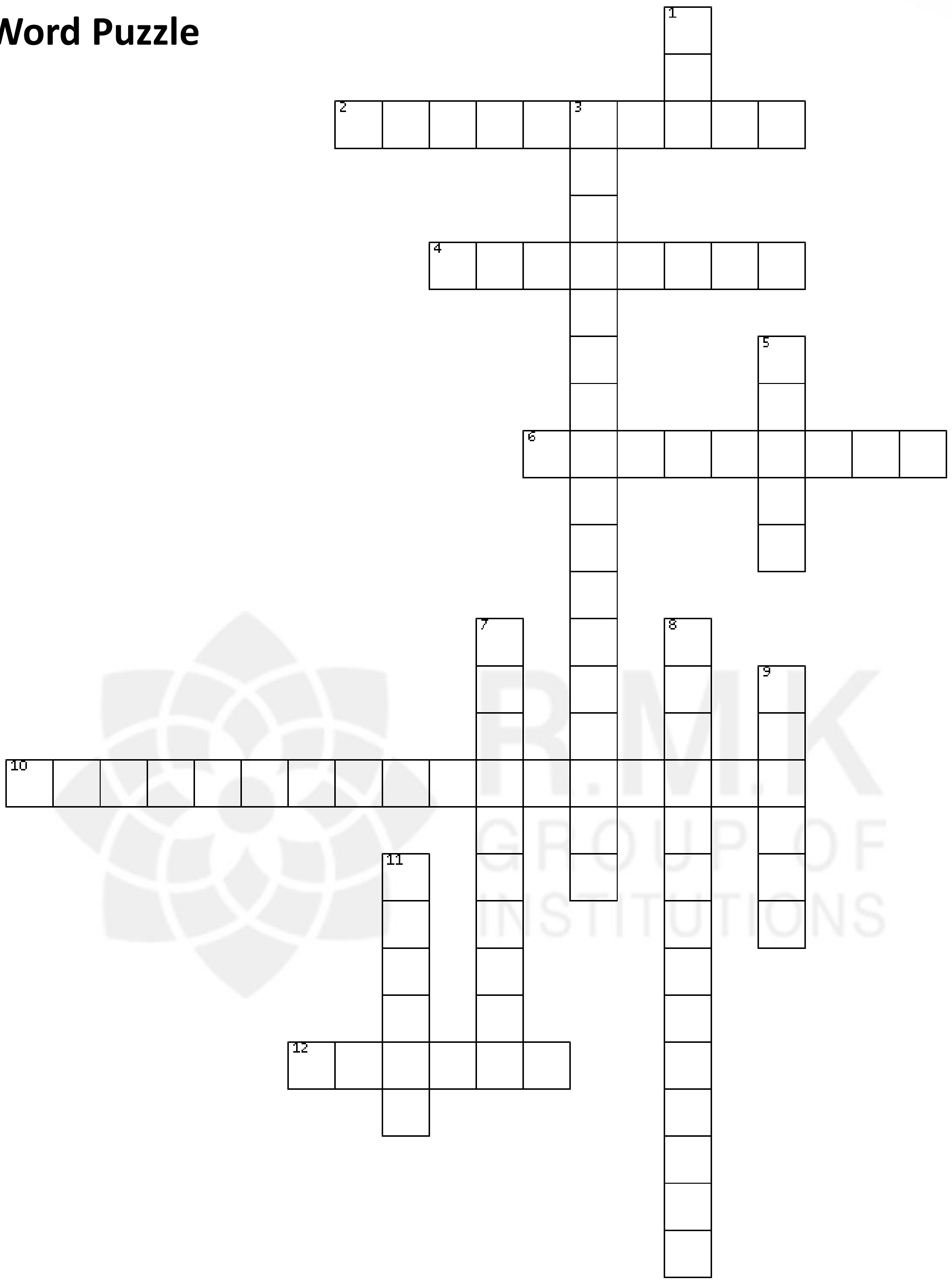
Lecture Plan – Unit V

UNIT - V						
S. No.	Topic	**Scheduled Date**	**Actual Date of Completion**	**CO**	**Mode of Delivery**	**Taxonomy Level**
1	JDBC			CO5	Video/ PPT	K1
2	DataSource			CO5	PPT	K1
3	Configurations			CO5	PPT	K1
4	Connection Pools			CO5	Demonstration	K1
5	Driver Types			CO5	PPT	K1
6	ResultSet			CO5	PPT	K1
7	Prepared Statement			CO5	Demonstration	K1
8	Named Parameter			CO5	Demonstration	K1
9	Embedded SQL (Insert, Update, Delete, Join, union etc)			CO5	Demonstration	K1
10	Connection Close and Cleanup			CO5	Demonstration	K1

Activity Based Learning
Unit V

Activity Based Learning

Cross Word Puzzle



ACROSS

2. _____ pooling is a process where we maintain a cache of database connections
4. This method is called when any one operation in a transaction fails
6. This interface contains the data returned from the databse
10. This interface is a subinterface and it is used to execute parameterized query
12. JDBC _____ supports the JDBC database connection

DOWN

1. JDBC _____ provides the application to JDBC manager connection
3. This interface is used to execute SQL stored procedures
5. _____ processing allows to group similar queries into one unit
7. It is present in javax.sql package and it only declare two overloaded methods getConnection() and getConnection(String str1,Stringstr2)
8. _____ Statement helps in tracking indices for large queries

Lecture Notes
Unit V

Unit V

5.0 Introduction

- A database is a structured collection of data that is organized, managed, and stored in a way that enables efficient retrieval, manipulation, and analysis of information. It is designed to store large amounts of data, ranging from simple text or numerical values to complex multimedia files.
- In a database, data is organized into tables, which consist of rows and columns. Each row represents a record or a specific instance of data, while each column represents a specific attribute or data field. This tabular structure allows for easy organization and retrieval of data based on various criteria.
- Databases provide a structured and standardized way to store and manage data, ensuring data integrity, consistency, and security. They offer functionalities such as data validation, data indexing, and data relationships, which help maintain the accuracy and reliability of the stored information.
- There are different types of databases, including:
    1. Relational Databases: Relational databases are based on the relational model, which organizes data into tables and establishes relationships between them. They use Structured Query Language (SQL) for querying and manipulating data. Examples of popular relational databases include MySQL, Oracle Database, Microsoft SQL Server, and PostgreSQL.
    2. NoSQL Databases: NoSQL (Not Only SQL) databases are designed to handle unstructured or semi-structured data, providing flexible data models and scalability. They are suitable for handling large volumes of data or scenarios where data structures vary. Examples of NoSQL databases include MongoDB, Cassandra, and Redis.
    3. Object-Oriented Databases: Object-oriented databases store data in the form of objects, similar to how objects are used in object-oriented programming. They provide support for encapsulation, inheritance, and polymorphism. Examples of object-oriented databases include db4o and Versant.

4. Hierarchical Databases: Hierarchical databases organize data in a tree-like structure, with parent-child relationships between data elements. They are suitable for storing data with hierarchical relationships, such as file systems. Examples include IBM's Information Management System (IMS) and Windows Registry.
5. Graph Databases: Graph databases represent data using nodes and edges, where nodes represent entities, and edges represent relationships between entities. They are optimized for traversing complex relationships and are commonly used for social network analysis, recommendation systems, and network management. Examples include Neo4j and Amazon Neptune.

- Databases play a crucial role in modern applications and systems, enabling efficient data storage, retrieval, and manipulation. They provide a foundation for data-driven decision-making, business intelligence, and application development across various domains, such as finance, healthcare, e-commerce, and more.

Components of Database

- A typical database system consists of several components that work together to manage and organize data. Here are the main components of a database:

1. Data: Data is the fundamental component of a database. It refers to the actual information that is stored, such as customer details, product information, transaction records, or any other relevant data specific to the application or system.
2. Database Management System (DBMS): The DBMS is the software responsible for managing and controlling the database. It provides an interface between the user or application and the underlying database, facilitating data storage, retrieval, manipulation, and security. Popular DBMSs include MySQL, Oracle Database, Microsoft SQL Server, and PostgreSQL.
3. Database Schema: The database schema defines the structure and organization of the database. It describes the tables, relationships between tables, constraints, and other metadata. The schema defines the logical and physical design of the database, including the data types, primary and foreign keys, indexes, and other structural elements.

4. Tables: Tables are the basic storage units in a database. They consist of rows and columns, with each row representing a record or instance of data, and each column representing a specific attribute or field. Tables hold the actual data and define the structure of the data, including the column names, data types, and constraints.
5. Relationships: Relationships establish connections between tables in a database. They define how data in one table is related to data in another table. The main types of relationships include one-to-one, one-to-many, and many-to-many. Relationships help maintain data integrity, enforce data consistency, and enable efficient data retrieval through joins.
6. Queries: Queries are used to retrieve, manipulate, and analyze data in a database. They allow users or applications to extract specific data based on specified conditions or criteria. Queries can range from simple searches to complex aggregations and calculations, and they are typically expressed using SQL (Structured Query Language) for relational databases.
7. Indexes: Indexes are data structures that improve the speed of data retrieval operations. They provide a quick lookup mechanism by creating a sorted representation of one or more columns in a table. Indexes help optimize query performance by reducing the number of disk reads required to locate data.
8. Views: Views are virtual tables derived from one or more existing tables in a database. They present a customized or filtered view of the data, hiding certain columns or rows based on predefined conditions or security rules. Views provide a way to simplify complex queries, enhance data security, and offer a logical separation between the physical data and its presentation.
9. Transactions: Transactions ensure the atomicity, consistency, isolation, and durability (ACID) properties of database operations. A transaction is a logical unit of work that consists of one or more database operations, such as inserting, updating, or deleting records. Transactions help maintain data integrity by ensuring that all operations within a transaction either succeed or fail together.

10. Security and Access Control: Database systems include security features to protect data from unauthorized access, modification, or disclosure. Access control mechanisms ensure that only authorized users or applications can interact with the database. This includes user authentication, role-based access control, data encryption, and auditing capabilities.

11. Backup and Recovery: Backup and recovery mechanisms are essential for protecting data against loss or corruption. Database systems provide utilities and techniques to create backups of the database, enabling recovery in the event of hardware failures, software errors, or data corruption. Backup strategies may include full backups, incremental backups, or continuous data replication.

- These components work together to provide a robust and efficient data management system, allowing users and applications to store, retrieve, manipulate, and secure data effectively. Robust and efficient management in a database refers to the ability of the database system to handle data effectively, ensuring reliability, performance, and scalability.

Need of a data base connectivity

- Connecting a database with Java offers several advantages and enables powerful data-driven applications. Here are some reasons why you might want to connect a database with Java:

1. Data Storage and Retrieval: Databases provide a structured and organized way to store and retrieve data. By connecting a database with Java, you can efficiently store, retrieve, and manipulate data from within your Java application.

2. Data Persistence: Connecting a database allows you to persist data beyond the lifespan of your application. This means that data can be stored permanently and accessed later, even if the application is restarted or the server is shut down.

3. Data Integrity and Security: Databases offer mechanisms to enforce data integrity constraints, such as unique keys, foreign key relationships, and validation rules. Connecting a database with Java ensures that your application interacts with the data in a secure and controlled manner, preventing data corruption or unauthorized access.

4. Scalability: Databases provide scalable solutions for handling large volumes of data and concurrent access. By connecting a database with Java, you can build applications that can handle increasing data loads and support multiple users accessing and modifying the data simultaneously.

5. Data Analysis and Reporting: Connecting a database with Java enables you to perform complex data analysis, generate reports, and extract valuable insights from the data. Java's rich ecosystem of libraries and frameworks can be leveraged to process, analyze, and visualize data from the connected database.

6. Integration with Other Systems: Many enterprise systems and third-party applications rely on databases as their data source. By connecting a database with Java, you can seamlessly integrate your application with other systems, exchange data, and leverage the functionalities provided by those systems.

7. Transaction Management: Databases support transactional operations, allowing you to group multiple database operations into atomic units of work. Connecting a database with Java enables you to manage transactions, ensuring that changes to the data are consistent and can be rolled back if necessary.

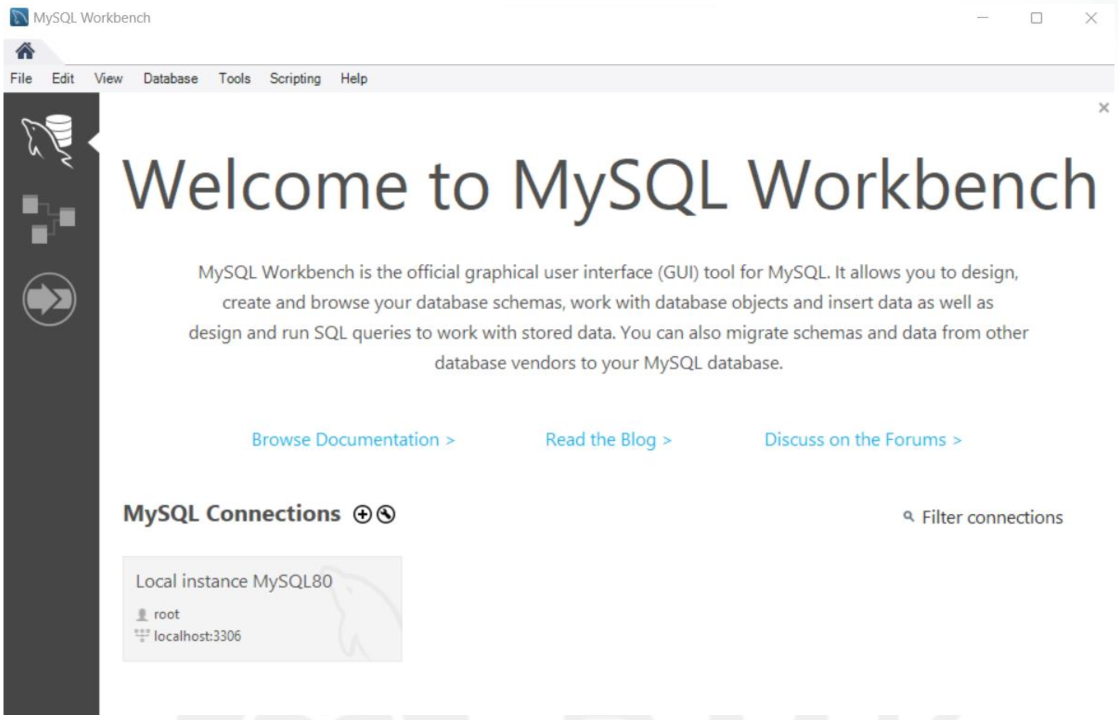
8. Data Modeling and Schema Design: Databases offer features for data modeling and schema design, allowing you to define the structure and relationships of your data. By connecting a database with Java, you can utilize object-relational mapping (ORM) frameworks to map database tables to Java objects and simplify the interaction between your application and the database.

- In summary, connecting a database with Java empowers you to leverage the full potential of databases, enabling efficient data storage, retrieval, persistence, scalability, data analysis, integration, and transaction management within your Java applications.

MySQL connection using Eclipse IDE

- To connect MySQL with Java using Eclipse, you can follow these steps:
- Set up MySQL and the MySQL Connector/J library:
    - Install MySQL on your machine and make sure it is running.
    - Download the MySQL Connector/J library (JDBC driver) from the MySQL website or Maven repository.
- Create a new Java project in Eclipse:
    - Launch Eclipse and create a new Java project by selecting "File" > "New" > "Java Project".
    - Provide a name for your project and click "Finish".
- Add the MySQL Connector/J library to your project:
    - Right-click on your project in the Package Explorer and select "Properties".
    - In the properties window, go to "Java Build Path" > "Libraries" tab.
    - Click on "Add External JARs" and navigate to the location where you saved the MySQL Connector/J JAR file. Select the JAR file and click "Open" to add it to your project's build path.
- Write Java code to establish a connection:
    - Create a new Java class in your project by right-clicking on the project, selecting "New" > "Class", and providing a class name.
    - Inside the class, import the necessary packages.
- Run the Java program:
    - Right-click on your Java class and select "Run As" > "Java Application".
- The program will execute and attempt to establish a connection to the MySQL database. If the connection is successful, you will see the message "Connection to MySQL database established successfully!" printed in the console.

MySQL Environment


MySQL Workbench

File Edit View Database Tools Scripting Help

Welcome to MySQL Workbench

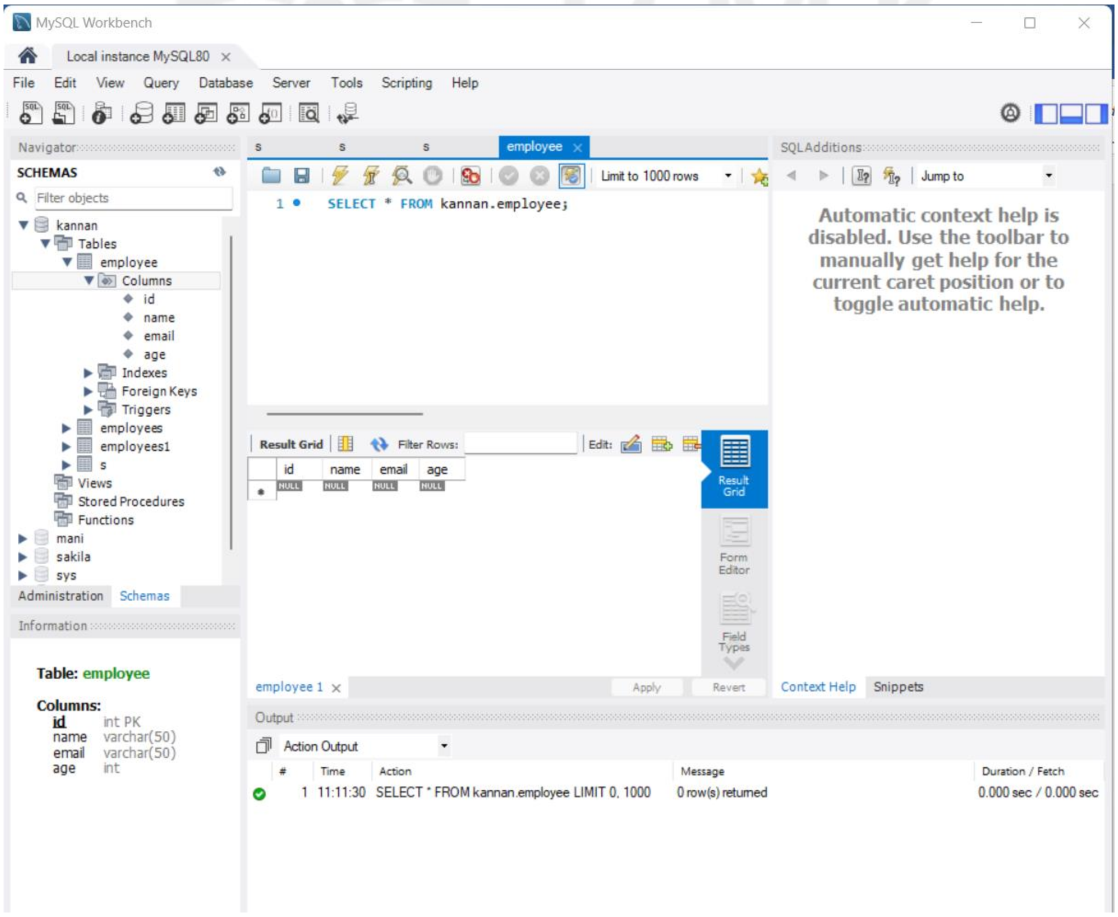
MySQL Workbench is the official graphical user interface (GUI) tool for MySQL. It allows you to design, create and browse your database schemas, work with database objects and insert data as well as design and run SQL queries to work with stored data. You can also migrate schemas and data from other database vendors to your MySQL database.

Browse Documentation > Read the Blog > Discuss on the Forums >

MySQL Connections

Filter connections

Local instance MySQL80
root
localhost:3306


MySQL Workbench — Local instance MySQL80

File Edit View Query Database Server Tools Scripting Help

Navigator — SCHEMAS

- kannan
  - Tables
    - employee
      - Columns
        - id
        - name
        - email
        - age
      - Indexes
      - Foreign Keys
      - Triggers
    - employees
    - employees1
    - s
  - Views
  - Stored Procedures
  - Functions
- mani
- sakila
- sys

```
SELECT * FROM kannan.employee;
```

Limit to 1000 rows

Result Grid

id	name	email	age
NULL	NULL	NULL	NULL

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Information

Table: employee

Columns:

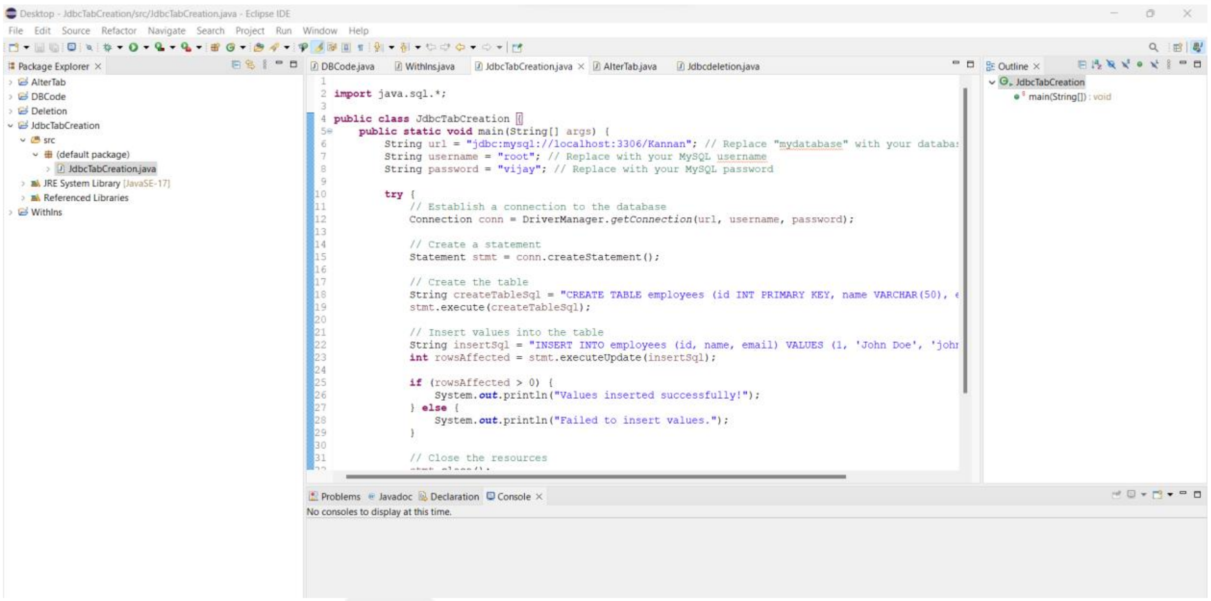
id	int PK
name	varchar(50)
email	varchar(50)
age	int

Output — Action Output

```
#  Time      Action                                       Message             Duration / Fetch
1  11:11:30  SELECT * FROM kannan.employee LIMIT 0, 1000  0 row(s) returned   0.000 sec / 0.000 sec
```

Eclipse IDE Environment

❁ Compilation Environment

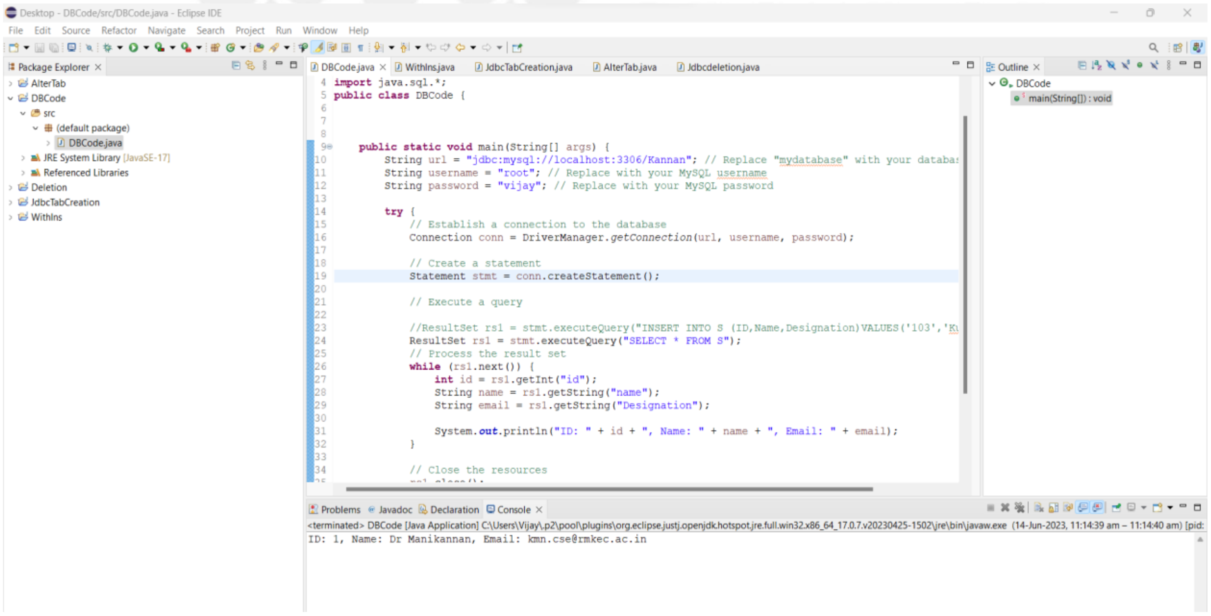


Desktop - JdbcTabCreation/src/JdbcTabCreation.java - Eclipse IDE

```
import java.sql.*;

public class JdbcTabCreation {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/Kannan"; // Replace "mydatabase" with your databa
        String username = "root"; // Replace with your MySQL username
        String password = "vijay"; // Replace with your MySQL password

        try {
            // Establish a connection to the database
            Connection conn = DriverManager.getConnection(url, username, password);

            // Create a statement
            Statement stmt = conn.createStatement();

            // Create the table
            String createTableSql = "CREATE TABLE employees (id INT PRIMARY KEY, name VARCHAR(50), e
            stmt.execute(createTableSql);

            // Insert values into the table
            String insertSql = "INSERT INTO employees (id, name, email) VALUES (1, 'John Doe', 'johr
            int rowsAffected = stmt.executeUpdate(insertSql);

            if (rowsAffected > 0) {
                System.out.println("Values inserted successfully!");
            } else {
                System.out.println("Failed to insert values.");
            }

            // Close the resources
```

```
No consoles to display at this time.
```

❁ Interpretation Environment



Desktop - DBCode/src/DBCode.java - Eclipse IDE

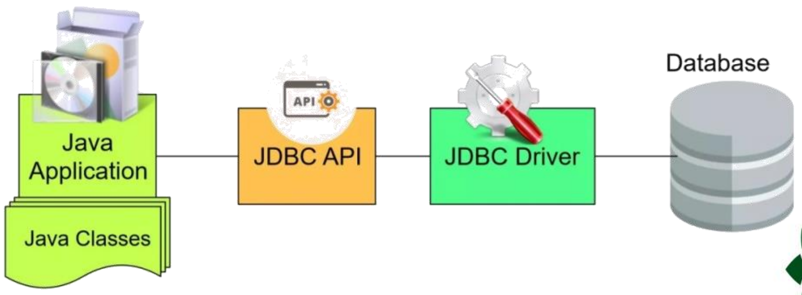
```
import java.sql.*;
public class DBCode {



    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/Kannan"; // Replace "mydatabase" with your databa
        String username = "root"; // Replace with your MySQL username
        String password = "vijay"; // Replace with your MySQL password

        try {
            // Establish a connection to the database
            Connection conn = DriverManager.getConnection(url, username, password);

            // Create a statement
            Statement stmt = conn.createStatement();

            // Execute a query

            //ResultSet rs1 = stmt.executeQuery("INSERT INTO S (ID,Name,Designation)VALUES('103','Ku
            ResultSet rs1 = stmt.executeQuery("SELECT * FROM S");
            // Process the result set
            while (rs1.next()) {
                int id = rs1.getInt("id");
                String name = rs1.getString("name");
                String email = rs1.getString("Designation");

                System.out.println("ID: " + id + ", Name: " + name + ", Email: " + email);
            }

            // Close the resources
```

```
<terminated> DBCode [Java Application] C:\Users\Vijay\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.7.v20230425-1502\jre\bin\javaw.exe (14-Jun-2023, 11:14:39 am – 11:14:40 am) [pid:
ID: 1, Name: Dr Manikannan, Email: kmn.cse@rmkec.ac.in
```

5.1 JDBC

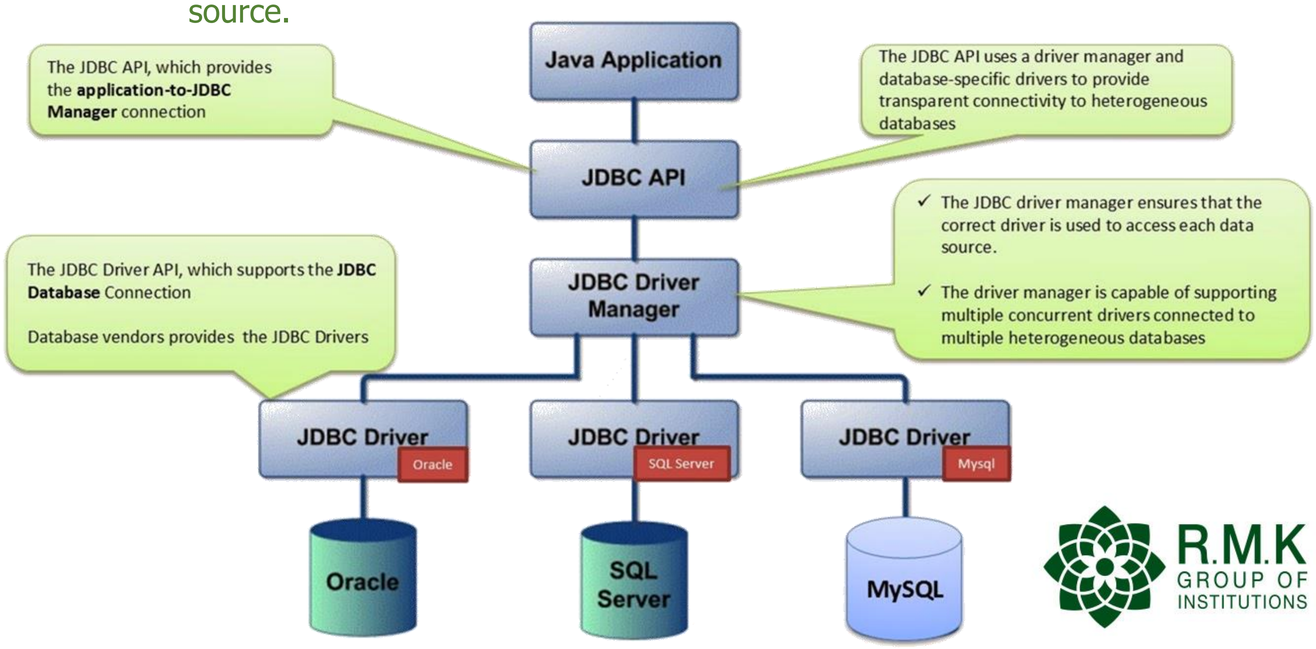
- JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the query with the database.
- It is a part of JavaSE (Java Standard Edition). JDBC API uses JDBC drivers to connect with the database.
- There are four types of JDBC drivers:
    1. JDBC-ODBC Bridge Driver,
    2. Native Driver,
    3. Network Protocol Driver, and
    4. Thin Driver
- Components of JDBC

1. JDBC API: It provides various methods and interfaces for easy communication with the database. It provides two packages as follows, which contain the java SE and Java EE platforms to exhibit WORA(write once run anywhere) capabilities.
2. JDBC Driver manager: It loads a database-specific driver in an application to establish a connection with a database. It is used to make a database-specific call to the database to process the user request.
3. JDBC Test suite: It is used to test the operation(such as insertion, deletion, updation) being performed by JDBC Drivers.
4. JDBC-ODBC Bridge Drivers: It connects database drivers to the database. This bridge translates the JDBC method call to the ODBC function call. It makes use of the sun.jdbc.odbc package which includes a native library to access ODBC characteristics.



- **Architecture of JDBC**
- Two-tier model: A java application communicates directly to the data source. The JDBC driver enables the communication between the application and the data source. When a user sends a query to the data source, the answers for those queries are sent back to the user in the form of results. The data source can be located on a different machine on a network to which a user is connected. This is known as a client/server configuration, where the user's machine acts as a client, and the machine has the data source running acts as the server.
- Three-tier model: In this, the user's queries are sent to middle-tier services, from which the commands are again sent to the data source. The results are sent back to the middle tier, and from there to the user.

1. Application: It is a java applet or a servlet that communicates with a data source.
2. The JDBC API: The JDBC API allows Java programs to execute SQL statements and retrieve results. Some of the important classes and interfaces defined in JDBC API are as follows:
   i. DriverManager: It plays an important role in the JDBC architecture. It uses some database-specific drivers to effectively connect enterprise applications to databases.
   ii. JDBC drivers: To communicate with a data source through JDBC, you need a JDBC driver that intelligently communicates with the respective data source.


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

5.1 JDBC Data Source

```
public interface DataSource
extends CommonDataSource, Wrapper
```

- Java DataSource interface is present in javax.sql package and it only declare two overloaded methods getConnection() and getConnection(String str1,String str2).
- It is the responsibility of different Database vendors to provide different kinds of implementation of DataSource interface. For example MySQL JDBC Driver provides basic implementation of DataSource interface with com.mysql.jdbc.jdbc2.optional.MysqlDataSource class

```
import java.io.FileInputStream;
import java.io.IOException;
import java.sql.SQLException;
import java.util.Properties;
import javax.sql.DataSource;
import oracle.jdbc.pool.OracleDataSource;
import com.mysql.jdbc.jdbc2.optional.MysqlDataSource;
public class MyDataSourceFactory {
    public static DataSource getMySQLDataSource() {
        Properties props = new Properties();
        FileInputStream fis = null;
        MysqlDataSource mysqlDS = null;
        try {
            fis = new FileInputStream("db.properties");
            props.load(fis);
            mysqlDS = new MysqlDataSource();
            mysqlDS.setURL(props.getProperty("MYSQL_DB_URL"));
            mysqlDS.setUser(props.getProperty("MYSQL_DB_USERNAME"));
            mysqlDS.setPassword(props.getProperty("MYSQL_DB_PASSWORD"));
        } catch (IOException e) {
            e.printStackTrace();
        }
        return mysqlDS;
}
```

5.3 JDBC Configuration

- Java configuration properties are added to the agent automatically.
- If the JDBC data source defined in the agent, the following required, common configuration fields are added to the agent automatically,
  - JDBC database type: Type of database to which you are connecting, IBM® DB2®, Microsoft SQL Server, or Oracle Database Server.
  - JDBC user name: User name that is used to authenticate with the database server.
  - JDBC password: Password that is used to authenticate with the database server.
  - Base paths: List of directories that are searched for JAR files that are named in the Class Path field, or directories that are named in the JAR directories field, that are not fully qualified. Directory names are separated by a semi-colon (;) on Windows, and by a semi-colon (;) or colon (:) on UNIX systems.
  - Class path: Explicitly named JAR files to be searched by the agent. Any files that are not fully qualified are appended to each of the Base Paths until the JAR file is found.
  - JAR directories: List of directories that are searched for JAR files. Directory names are separated by a semi-colon (;) on Windows, and by a semi-colon (;) or colon (:) on UNIX systems. The JAR files in these directories do not have to be explicitly identified; they are found because they are in one of these directories. Subdirectories of these directories are not searched. Any directories that are not fully qualified are appended to each of the Base Paths until the directory is found.

5.4 JDBC Connection Pools

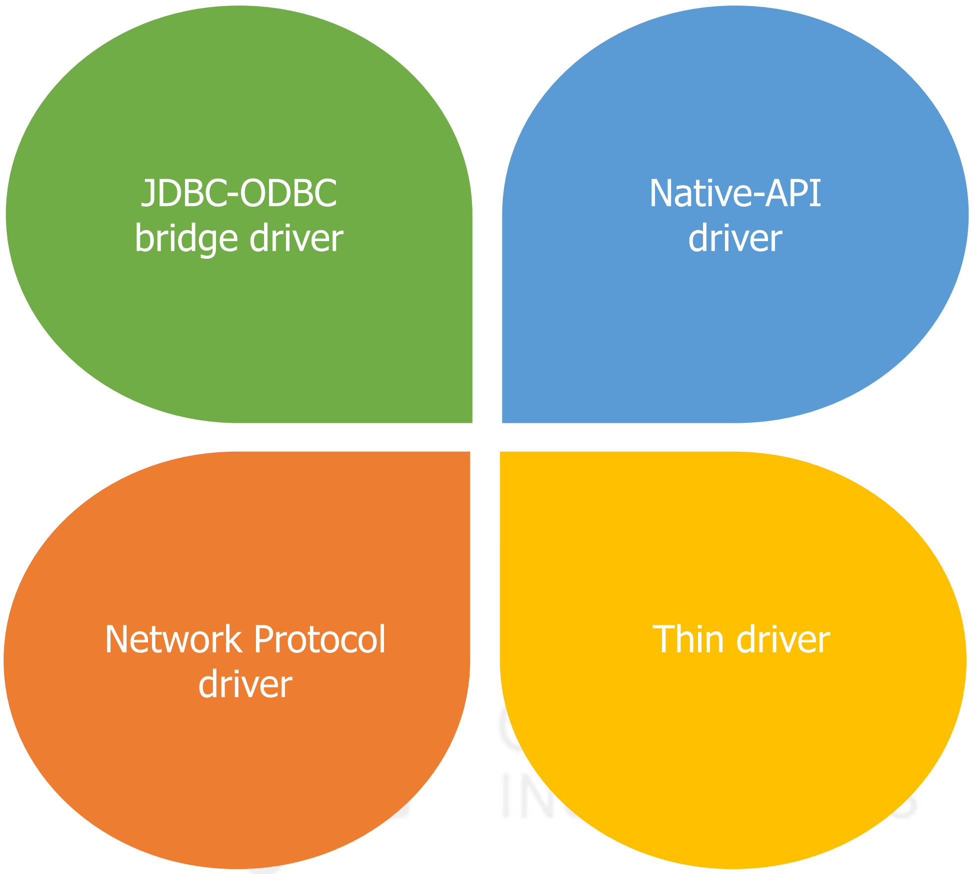
- Connection pooling is a process where we maintain a cache of database connections and has become the standard for middleware database drivers.
- The process of creating a connection, always an expensive, time-consuming operation, is multiplied in these environments where a large number of users are accessing the database in short, unconnected operations.
- Creating connections over and over in these environments is simply too expensive.
- In the case of a JDBC connection pool, a pool of Connection objects are created at the time the application server starts. These objects are then managed by a pool manager that disperses connections as they are requested by clients and returns them to the pool when it determines the client is finished with the Connection object.
- When the connection pool server starts, it creates a predetermined number of Connection objects. A client application would then perform a lookup to retrieve a reference to a DataSource object that implements the ConnectionPoolDataSource interface.
- The client application would not need to make any special provisions to use the pooled data source; the code would be no different from code written for a non-pooled DataSource.
- When the client application requests a connection from the ConnectionPoolDataSource, the data source implementation would retrieve a physical connection to the client application. The ConnectionPoolDataSource would return a Connection object that implemented the PooledConnection interface.
- The PooledConnection interface dictates the use of event listeners. These event listeners allow the connection pool manager to capture important connection events, such as attempts by the client application to close the connection.

- When the driver traps a close-connection event, it intercedes and performs a pseudo-close operation that merely takes the Connection object, returns it to the pool of available connection, and performs any housekeeping that is necessary.

```
public DataSource setUpPool() throws Exception {
        Class.forName(JDBC_DRIVER);

        // Creates an Instance of GenericObjectPool That Holds Our Pool of Connections
Object!
        gPool = new GenericObjectPool();
        gPool.setMaxActive(5);

        // Creates a ConnectionFactory Object Which Will Be Use by the Pool to Create
the Connection Object!
        ConnectionFactory cf = new
DriverManagerConnectionFactory(JDBC_DB_URL, JDBC_USER, JDBC_PASS);

       // Creates a PoolableConnectionFactory That Will Wraps the Connection Object
Created by the ConnectionFactory to Add Object Pooling Functionality!
        PoolableConnectionFactory pcf = new PoolableConnectionFactory(cf,
gPool, null, null, false, true);
        return new PoolingDataSource(gPool);
    }

    public GenericObjectPool getConnectionPool() {
        return gPool;
    }
```

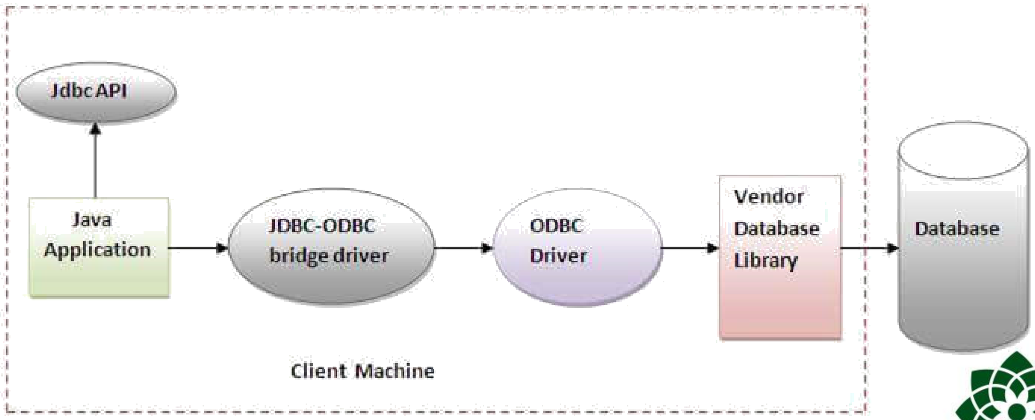
5.5 JDBC Driver Types

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



1. JDBC-ODBC bridge driver

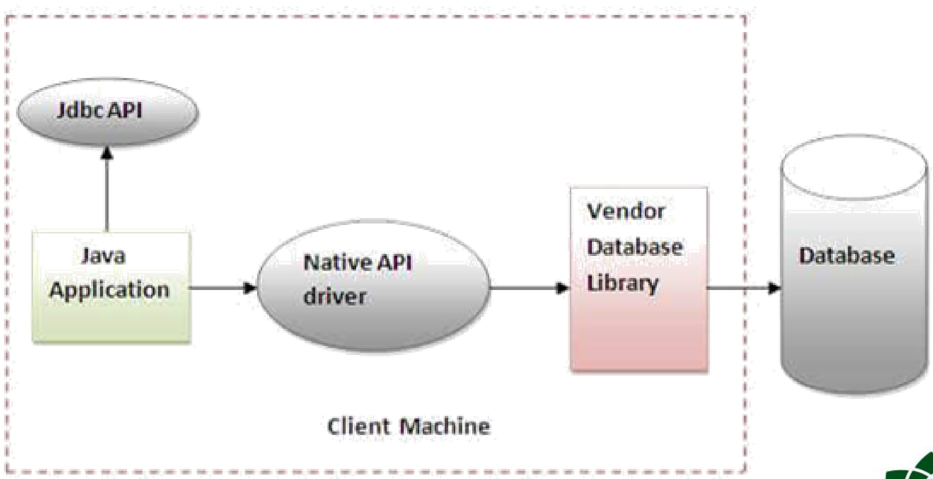
- The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. This is now discouraged because of thin driver.



- Oracle does not support the JDBC-ODBC Bridge from Java 8. Oracle recommends that you use JDBC drivers provided by the vendor of your database instead of the JDBC-ODBC Bridge.
- Advantages
  - easy to use.
  - can be easily connected to any database.
- Disadvantages
  - Performance degraded because JDBC method call is converted into the ODBC function calls.
  - The ODBC driver needs to be installed on the client machine.

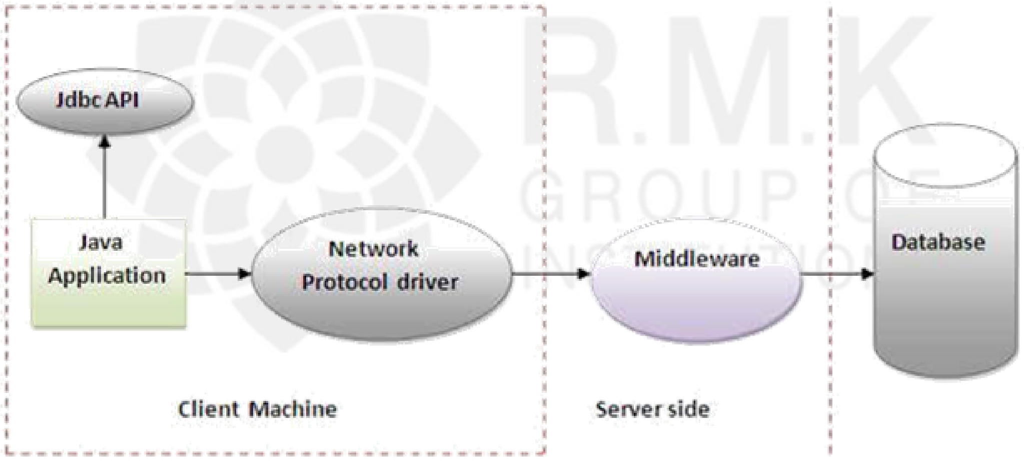
2. Native-API driver

- The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java.
- Advantage
  - performance upgraded than JDBC-ODBC bridge driver.
- Disadvantages
  - The Native driver needs to be installed on the each client machine.
  - The Vendor client library needs to be installed on client machine.



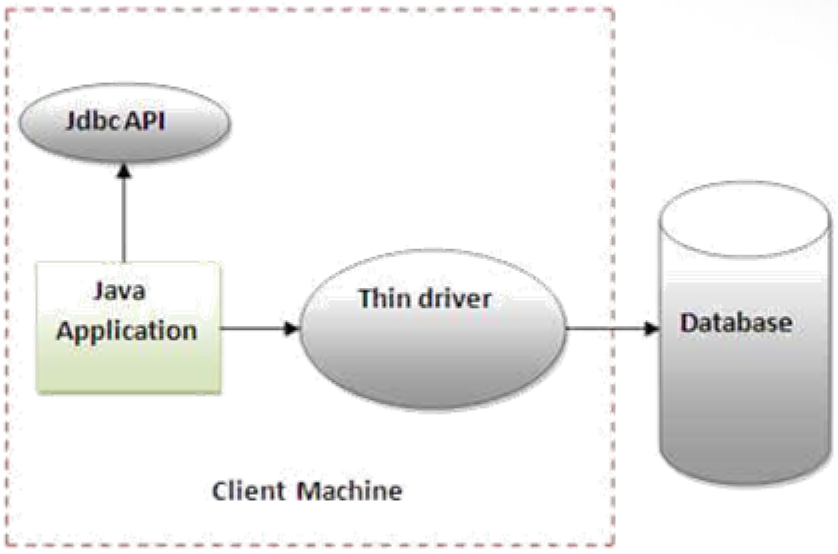
3. Network Protocol driver

- The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.
- Advantage
  - No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.
- Disadvantages
  - Network support is required on client machine.
  - Requires database-specific coding to be done in the middle tier.
  - Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.



4. Thin driver

- 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.
- Advantages
  - Better performance than all other drivers.
  - No software is required at client side or server side.
- Disadvantage
  - Drivers depend on the Database.



5.6 JDBC ResultSets

- The SQL statements that read data from a database query, return the data in a result set.
- The SELECT statement is the standard way to select rows from a database and view them in a result set.
- The java.sql.ResultSet interface represents the result set of a database query.
- A ResultSet object maintains a cursor that points to the current row in the result set.
- The term "result set" refers to the row and column data contained in a ResultSet object.
- The methods of the ResultSet interface can be broken down into three categories
  - Navigational methods – Used to move the cursor around.
  - Get methods – Used to view the data in the columns of the current row being pointed by the cursor.
  - Update methods – Used to update the data in the columns of the current row. The updates can then be updated in the underlying database as well.
- The cursor is movable based on the properties of the ResultSet.

- JDBC provides the following connection methods to create statements with desired ResultSet
  - createStatement(int RSType, int RSConcurrency);
  - prepareStatement(String SQL, int RSType, int RSConcurrency);
  - prepareCall(String sql, int RSType, int RSConcurrency);
- The first argument indicates the type of a ResultSet object and the second argument is one of two ResultSet constants for specifying whether a result set is read-only or updatable.
- **Type of ResultSet**

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.

- **Concurrency of ResultSet**

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

Navigating a Result Set

Methods & Description
public void beforeFirst() throws SQLException Moves the cursor just before the first row.
public void afterLast() throws SQLException Moves the cursor just after the last row.
public boolean first() throws SQLException Moves the cursor to the first row.
public void last() throws SQLException Moves the cursor to the last row.
public boolean absolute(int row) throws SQLException Moves the cursor to the specified row.
public boolean relative(int row) throws SQLException Moves the cursor the given number of rows forward or backward, from where it is currently pointing.
public boolean previous() throws SQLException Moves the cursor to the previous row. This method returns false if the previous row is off the result set.
public boolean next() throws SQLException Moves the cursor to the next row. This method returns false if there are no more rows in the result set.
public int getRow() throws SQLException Returns the row number that the cursor is pointing to.
public void moveToInsertRow() throws SQLException Moves the cursor to a special row in the result set that can be used to insert a new row into the database. The current cursor location is remembered.
public void moveToCurrentRow() throws SQLException Moves the cursor back to the current row if the cursor is currently at the insert row; otherwise, this method does nothing

Viewing a Result Set

Methods & Description
public int getInt(String columnName) throws SQLException Returns the int in the current row in the column named columnName.
public int getInt(int columnIndex) throws SQLException Returns the int in the current row in the specified column index. The column index starts at 1, meaning the first column of a row is 1, the second column of a row is 2, and so on.

Updating a Result Set

Methods & Description
public void updateString(int columnIndex, String s) throws SQLException Changes the String in the specified column to the value of s.
public void updateString(String columnName, String s) throws SQLException Similar to the previous method, except that the column is specified by its name instead of its index.
public void updateRow() Updates the current row by updating the corresponding row in the database.
public void deleteRow() Deletes the current row from the database
public void refreshRow() Refreshes the data in the result set to reflect any recent changes in the database.
public void cancelRowUpdates() Cancels any updates made on the current row.
public void insertRow() Inserts a row into the database. This method can only be invoked when the cursor is pointing to the insert row.

```
import java.sql.*;
class FetchRecord{
public static void main(String args[])throws Exception{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system"
,"oracle");
Statement
stmt=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCU
R_UPDATABLE);
ResultSet rs=stmt.executeQuery("select * from emp765");
//getting the record of 3rd row
rs.absolute(3);
System.out.println(rs.getString(1)+" "+rs.getString(2)+" "+rs.getString(3));
con.close();
}}
```

5.7 PreparedStatement Interface

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

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

- "?" is passed as the parameter, the value will be set by calling the setter methods of PreparedStatement.
- The prepareStatement() method of Connection interface is used to return the object of PreparedStatement.

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

- The important methods of PreparedStatement interface are given below

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.

Inserting the record

```
import java.sql.*;
class InsertPrepared{
public static void main(String args[]){
try{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system"
,"oracle");
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();
}catch(Exception e){ System.out.println(e);}

}
}
```

Updating the record

```
PreparedStatement stmt=con.prepareStatement("update emp set name=? where
id=?");
stmt.setString(1,"Sonoo");//1 specifies the first parameter in the query i.e. name
stmt.setInt(2,101);

int i=stmt.executeUpdate();
System.out.println(i+" records updated");
```

Deleteing the record

```
PreparedStatement stmt=con.prepareStatement("delete from emp where id=?");
stmt.setInt(1,101);

int i=stmt.executeUpdate();
System.out.println(i+" records deleted");
```

Retrieving the records of a table

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

Inserting records until user press n

```
import java.sql.*;
import java.io.*;
class RS{
public static void main(String args[])throws Exception{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system"
,"oracle");
PreparedStatement ps=con.prepareStatement("insert into emp130 values(?,?,?)");
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
do{
System.out.println("enter id:");
int id=Integer.parseInt(br.readLine());
System.out.println("enter name:");
String name=br.readLine();
System.out.println("enter salary:");
float salary=Float.parseFloat(br.readLine());
ps.setInt(1,id);
ps.setString(2,name);
ps.setFloat(3,salary);
int i=ps.executeUpdate();
System.out.println(i+" records affected");
System.out.println("Do you want to continue: y/n");
String s=br.readLine();
if(s.startsWith("n")){
break;
}
}while(true);

con.close();
}}
```

5.8 Named Parameters

- In PreparedStatement parameters are anonymous and accessed by index for small queries with one or two parameters, this is not an issue. for larger queries, keeping track of the indices becomes very difficult.
- NamedParameterStatement helps in overcoming this drawback.
- Named parameters are parameters in a query that are prefixed with a colon (:). Named parameters in a query are bound to an argument by the javax.persistence.Query.setParameter(String name, Object value) method.
- The syntax is the same as PreparedStatement except that, instead of question marks, parameters are represented as a colon followed by an identifier.

```
String query = "select * from people where (first_name = :name or last_name
= :name) and address = :address");
NamedParameterStatement p = new NamedParameterStatement(con, query);
p.setString("name", name);
p.setString("address", address);
```

- Behind the scenes, the class works by replacing the parameter markers with questions marks and creating a PreparedStatement.
- A mapping is kept between parameter names and their indices. This mapping is referred to when the parameters are injected.

```
Connection con=getConnection();
String query="select * from my_table where name=:name or  address=:address";
NamedParameterStatement p=new NamedParameterStatement(con, query);
p.setString("name", "bob");
p.setString("address", "123 terrace ct");
ResultSet rs=p.executeQuery();
```

- Named parameters are very handy when we start having methods with a lot of parameters and we want to allow invoking the method with an arbitrary subset of them, using default values for the rest.

5.9 Embedded SQL (SQLJ)

- SQLJ is an emerging database programming tool that allows embedding of static SQL statements in Java programs.
- The SQLJ translator converts Java programs embedded with static SQL statements into pure Java code, which can then be executed through a JDBC driver against the database. Programmers can also perform dynamic SQL access to the database using JDBC features.

1. Import necessary classes.In addition to the JDBC classes, java.sql.*, every SQLJ program will need to include the SQLJ run-time classes sqlj.runtime.* and sqlj.runtime.ref.*. In addition, to establish the default connection to Oracle, the Oracle class from the oracle.sqlj.runtime.* package is required.

```
import java.sql.*;
import sqlj.runtime.*;
import sqlj.runtime.ref.*;
import java.io.*;
import oracle.sqlj.runtime.*;
```

2. Register the JDBC driver, if needed. If a non-Oracle JDBC driver is being used, a call to the registerDriver method of the DriverManager class is necessary.
3. Connect to the database. Connecting to the Oracle database is done by first obtaining a DefaultContext object using the getConnection method. url is the database URL, and user and password are the Oracle user ID and password, respectively. Setting autoCommit to true would create the connection in autocommit mode, and setting it to false would create a connection in which the transactions must be committed by the programmer.

```
public static DefaultContext getConnection
(String url,String user,String password,boolean autoCommit)
throws SQLException
```

```
DefaultContext cx1 =
Oracle.getConnection("jdbc:oracle:oci8:@","book","book",true);
```

- The DefaultContext object so obtained is then used to set the static default context.

```
DefaultContext.setDefaultContext(cx1);
```

- This DefaultContext object now provides the default connection to the database.

4. Embed SQL statements in the Java program. Once the default connection has been established, SQL statements can be embedded within the Java program using the following syntax

```
#sql {<sql-statement>}
```

- where #sql indicates to the SQLJ translator, called sqlj, that what follows is an SQL statement and <sql-statement> is any valid SQL statement, which may include host variables and host expressions.

```
import sqlj.runtime.*;
import sqlj.runtime.ref.*;
import java.sql.*;
import java.io.*;
import oracle.sqlj.runtime.*;
public class Simple1 {
public static void main (String args[]) throws SQLException {
DefaultContext cx1 =
Oracle.getConnection("jdbc:oracle:oci8:@",
"book","book",true);
DefaultContext.setDefaultContext(cx1);
String cn;
Double ap,bp,cp;
String sym = readEntry("Enter symbol : ").toUpperCase();
try {
#sql {select cname,current_price,ask_price,bid_price
into :cn,:cp,:ap,:bp
from security
where symbol = :sym };
}
```

```
 catch (SQLException e) {
System.out.println("Invalid symbol.");
return;
}
System.out.println("\n Company Name = " + cn);
System.out.println(" Last sale at = " + cp);
if (ap == null)
System.out.println(" Ask price = null");
else
System.out.println(" Ask price = " + ap);
if (bp == null)
System.out.println(" Bid price = null");
else
System.out.println(" Bid price = " + bp);
}
}
```

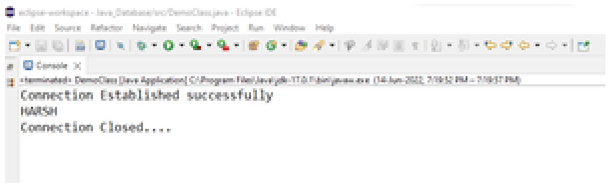
5.10 Connection Close and Cleanup

- In a Java program, we sometimes need to make sure we close a resource after we've finished using it.
- The database-related ones are particular important – if we don't close them, we can be left with unclosed connections to the database.

```
public static void close(ResultSet rs, Statement st) {
    if (rs != null) {
        try {
            rs.close();
        } catch (SQLException e) {
            SilverTrace.error("util", "DBUtil.close", "util.CAN_T_CLOSE_RESULTSET",
e);
        }
    }
    if (st != null) {
        try {
            st.close();
        } catch (SQLException e) {
            SilverTrace.error("util", "DBUtil.close", "util.CAN_T_CLOSE_STATEMENT",
e);
        }
    }
}
```

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

public class Main {
    static public void cleanupJDBCResouce(ResultSet rs, Statement stmt, String
oldCatalog, Connection conn) {
        try {
            if (rs != null)
                rs.close();
        } catch (SQLException e) {
            // TODO log properly JDBCASEPlugin.getDefault().log(e);
            e.printStackTrace();
        }
        try {
            if (stmt != null)
                stmt.close();
        } catch (SQLException e) {
            // TODO log properly JDBCASEPlugin.getDefault().log(e);
            e.printStackTrace();
        }
        try {
            if (oldCatalog != null)
                conn.setCatalog(oldCatalog);
        } catch (SQLException e) {
            // TODO log properly JDBCASEPlugin.getDefault().log(e);
            e.printStackTrace();
        }
    }
}
```

```
// This code is for establishing connection with MySQL
// database and retrieving data
// from db Java Database connectivity
/*
 *1. import --->java.sql
 *2. load and register the driver ---> com.jdbc.
 *3. create connection
 *4. create a statement
 *5. execute the query
 *6. process the results
 *7. close
 */
import java.io.*;
import java.sql.*;
class JDBCExample {
    public static void main(String[] args) throws Exception
    {
        String url
            = "jdbc:mysql://localhost:3306/table_name"; // table details
        String username = "root"; // MySQL credentials
        String password = "";
        String query
            = "select *from students"; // query to be run
        Class.forName(
            "com.mysql.cj.jdbc.Driver"); // Driver name
        Connection con = DriverManager.getConnection(
            url, username, password);
        System.out.println(
            "Connection Established successfully");
        Statement st = con.createStatement();
        ResultSet rs
            = st.executeQuery(query); // Execute query
        rs.next();
        String name
            = rs.getString("name"); // Retrieve name from db

        System.out.println(name); // Print result on console
        st.close(); // close statement
        con.close(); // close connection
        System.out.println("Connection Closed....");
    }
}
```



```
Connection Established successfully
HARSH
Connection Closed....
```

Advantages of database connection

❁ Connecting a database with Java offers several advantages. Here are some of the key benefits:

1. Data Management: Java provides a robust and structured way to interact with databases, allowing you to efficiently store, retrieve, update, and delete data. You can use SQL queries or higher-level abstractions like ORM frameworks to handle database operations, making data management tasks easier and more organized.
2. Data Persistence: By connecting a database with Java, you can persist data beyond the lifespan of your application. This means that data remains stored even after the application is closed or the server is shut down. This ensures data integrity and allows for seamless data retrieval in subsequent sessions.
3. Scalability: Databases are designed to handle large volumes of data and concurrent access. By connecting a database with Java, you can build scalable applications that can handle increasing data loads and support multiple users accessing and modifying data simultaneously.
4. Data Security: Databases offer built-in security features to protect sensitive data. By connecting a database with Java, you can enforce access controls, implement authentication and authorization mechanisms, and encrypt data to ensure its confidentiality and integrity.
5. Data Consistency: Databases support transaction management, ensuring that changes to data follow the principles of ACID (Atomicity, Consistency, Isolation, Durability). This allows you to maintain data consistency and handle concurrent updates or rollbacks in a controlled manner.
6. Data Analysis: Connecting a database with Java enables you to perform complex data analysis and generate insights from your data. You can use SQL queries or data processing frameworks to aggregate, filter, and analyze data, facilitating decision-making processes and business intelligence.

7. Integration with Other Systems: Many enterprise systems and third-party applications rely on databases as their primary data source. By connecting a database with Java, you can seamlessly integrate your application with other systems, exchange data, and leverage the functionalities provided by those systems.

8. Modularity and Maintainability: Separating data storage and retrieval logic from the application code improves modularity and maintainability. By connecting a database with Java, you can design a clear separation of concerns, making it easier to update, modify, or extend your application without affecting the underlying data storage layer.

9. Frameworks and Libraries: Java has a rich ecosystem of frameworks and libraries that facilitate database connections. JDBC (Java Database Connectivity) provides a standard API for connecting Java applications with databases. Additionally, Object-Relational Mapping (ORM) frameworks like Hibernate or JPA simplify the mapping of Java objects to database tables, reducing the need for manual SQL handling.

10. Reliability and Support: Databases are mature and reliable technologies that have been extensively tested and optimized. They are backed by dedicated support teams and active communities, providing resources, documentation, and assistance in case of any issues or questions.

- In summary, connecting a database with Java brings numerous advantages, including efficient data management, data persistence, scalability, security, data consistency, data analysis capabilities, integration possibilities, modularity, and the availability of robust frameworks and support. These advantages contribute to the development of robust, scalable, and data-driven applications.

Disadvantages of database connection

- While connecting a database with Java offers many advantages, there are also a few potential disadvantages to consider. Here are some of the common drawbacks associated with Java database connections:

1. Performance Overhead: Database connections in Java can introduce performance overhead compared to direct access to data in memory. The process of establishing a connection, executing queries, and retrieving data involves additional layers of abstraction and network communication, which can impact the overall performance of the application.

2. Complexity: Working with databases in Java often involves writing SQL queries, handling database connections, and managing transactions. This complexity can be challenging, especially for developers who are new to database programming or lack familiarity with SQL. It may require additional effort to learn and understand the intricacies of database interactions.

3. Dependency on Database Vendor: Connecting a database with Java usually involves using a specific JDBC driver provided by the database vendor. This dependency on a particular vendor's driver may limit portability and make it more challenging to switch to a different database system in the future, as the driver implementation and behavior may vary.

4. Potential for SQL Injection: When constructing SQL queries dynamically by concatenating user input, there is a risk of SQL injection attacks. If not handled properly, malicious users may exploit vulnerabilities in the application to execute unauthorized SQL commands or gain access to sensitive data. Preventing SQL injection requires careful input validation and the use of parameterized queries or prepared statements.

5. Data Access Coupling: Directly accessing the database from Java code can lead to tight coupling between the application and the database structure. Any changes to the database schema may require corresponding modifications in the Java code, potentially increasing maintenance efforts and making the application more rigid to adapt to evolving database requirements.

6. Database Portability: While JDBC provides a standard API for database connectivity, some database-specific features or SQL syntax may not be fully supported across different database systems. If the application needs to be compatible with multiple database vendors, it may require additional effort to ensure that the SQL queries and database-specific features are handled consistently across different database platforms.

7. Configuration and Setup: Setting up and configuring the database connection parameters, including the database URL, credentials, and connection pool settings, can be a complex task. It requires careful configuration management and may involve coordinating with database administrators or considering specific network and security requirements.

8. Deployment and Environment Considerations: Deploying a Java application that connects to a database requires ensuring that the target environment has the necessary database drivers and proper network connectivity. Configuration differences between development, testing, and production environments may introduce deployment complexities and compatibility issues.

- While these disadvantages exist, they can be mitigated or overcome with proper design, best practices, and careful consideration of the application's requirements and the chosen database technology.

Java JDBC Cheat Sheet

Basics

What is JDBC?

JDBC - Java Database Connectivity - is an API which is used by the Java applications to interact with the database management systems.

It consists of several classes and interfaces - written entirely in Java - which can be used to establish connection with the database, send the queries to the database and process the results returned by the database.

What are JDBC Drivers?

JDBC API doesn't directly interact with the database. It uses JDBC driver of that particular database with which it wants to interact.

JDBC drivers are nothing but the implementations of classes and interfaces provided in the JDBC API. These implementations are provided by a particular database vendor and supplied along with the database. These implementations are used by the JDBC API to interact with that database.

Types Of JDBC Drivers

There are four types of JDBC drivers.

1) Type 1 JDBC Drivers / JDBC-ODBC Bridge Drivers

This type of drivers translates all JDBC calls into ODBC calls and sends them to ODBC driver which interact with the database.

These drivers just acts as a bridge between JDBC and ODBC API and hence the name JDBC-ODBC bridge drivers.

They are partly written in Java.

2) Type 2 JDBC Drivers / Native API Drivers

This type of drivers translates all JDBC calls into database specific calls using native API of the database.

They are also not entirely written in Java.

3) Type 3 JDBC Drivers / Network Protocol Drivers

This type of drivers make use of application server or middle-tier server which translates all JDBC calls into database specific network protocol and then sent to the database.

They are purely written in Java.

4) Type 4 JDBC Drivers / Native Protocol Drivers

This type of JDBC drivers directly translate all JDBC calls into database specific network protocols without a middle tier.

They are most popular of all 4 type of drivers. They are also called thin drivers. They are entirely written in Java.

JDBC API

JDBC API is comprised of two packages - java.sql and javax.sql. Below are the some important classes and interfaces of JDBC API.

java.sql.DriverManager (Class) :

It acts as a primary mediator between your Java application and the driver of the database you want to connect with. Driver class of every database you want to connect with first has to get registered with this class before you start interacting with the database.

java.sql.Connection (Interface) :

It represents a session between Java application and a database. All SQL statements are executed and results are returned within the context of a Connection object. It is mainly used to create Statement, PreparedStatement and CallableStatement objects. You can also use it to retrieve the metadata of a database like name of the database product, name of the JDBC driver, major and minor version of the database etc...

java.sql.Statement (Interface) :

It is used to execute static SQL queries.

java.sql.PreparedStatement (Interface) :

It is used to execute parameterized or dynamic SQL queries.

java.sql.CallableStatement (Interface) :

It is used to execute SQL stored procedures.

java.sql.ResultSet (Interface) :

It contains the data returned from the database.

java.sql.ResultSetMetaData (Interface) :

This interface provides quick overview about a ResultSet object like number of columns, column name, data type of a column etc...

java.sql.DatabaseMetaData (Interface) :

It provides comprehensive information about a database.

java.sql.Date (Class) :

It represents a SQL date value.

java.sql.Time (Class) :

It represents a SQL time value.

java.sql.Blob (Interface) :

It represents a SQL BLOB (Binary Large Object) value. It is used to store/retrieve image files.

java.sql.Clob (Interface) :

It represents a SQL CLOB (Character Large Object) value. It is used to store/retrieve character files.

Database Connection Using JDBC API

Step 1 : Updating the class path with JDBC Driver

Add JDBC driver of a database with which you want to interact in the class path. JDBC driver is the jar file provided by the database vendors along with the database. It contains the implementations for all classes and interfaces of JDBC API with specific to that database.

Step 2 : Registering the driver class

Class.forName("Pass_Driver_Class_Here");

Step 3 : Creating the Connection object.

Connection con = DriverManager.getConnection(URL, username, password);

Step 4 : Creating the Statement Object

Statement stmt = con.createStatement();

Step 5 : Execute the queries.

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

Step 6 : Close the resources.

Close ResultSet, Statement and Connection objects.

Transaction Management

A transaction is a group of operations used to perform a particular task.

A transaction is said to be successful only if all the operations in a transaction are successful. If any one operation fails, the whole transaction will be cancelled.

In JDBC, transactions are managed using three methods of a Connection interface.

setAutoCommit() : It sets the auto commit mode of this connection object. By default it is true. It is set to false to manually manage the transactions.

commit() : It is called only when all the operations in a transaction are successful.

rollback() : It is called if any one operation in a transaction fails.

Batch Processing

Batch processing allows us to group similar queries into one unit and submit them all at once for execution. It reduces the communication overhead significantly and increases the performance.

Three methods of Statement interface are used for batch processing.

addBatch() : It is used to add SQL statement to the batch.

executeBatch() : It executes all SQL statements of a batch and returns an array of integers where each integer represents the status of a respective SQL statement.

clearBatch() : It removes all SQL statements added in a batch.

executeQuery() Vs executeUpdate() Vs execute()

executeQuery()	executeUpdate()	execute()
This method is used to execute the SQL statements which retrieve some data from the database.	This method is used to execute the SQL statements which update or modify the database.	This method can be used for any kind of SQL statements.
This method returns a ResultSet object which contains the results returned by the query.	This method returns an int value which represents the number of rows affected by the query. This value will be the 0 for the statements which return nothing.	This method returns a boolean value. TRUE indicates that query returned a ResultSet object and FALSE indicates that query returned an int value or returned nothing.
This method is used to execute only select queries.	This method is used to execute only non-select queries.	This method can be used for both select and non-select queries.
Ex : SELECT	Ex : DML → INSERT, UPDATE and DELETE DDL → CREATE, ALTER	This method can be used for any type of SQL statements.

Statement Vs PreparedStatement Vs CallableStatement

Statement	PreparedStatement	CallableStatement
It is used to execute normal SQL queries.	It is used to execute parameterized or dynamic SQL queries.	It is used to call the stored procedures.
It is preferred when a particular SQL query is to be executed only once.	It is preferred when a particular query is to be executed multiple times.	It is preferred when the stored procedures are to be executed.
You cannot pass the parameters to SQL query using this interface.	You can pass the parameters to SQL query at run time using this interface.	You can pass 3 types of parameters using this interface. They are – IN, OUT and IN OUT.
This interface is mainly used for DDL statements like CREATE, ALTER, DROP etc.	It is used for any kind of SQL queries which are to be executed multiple times.	It is used to execute stored procedures and functions.
The performance of this interface is very low.	The performance of this interface is better than the Statement interface (when used for multiple execution of same query).	The performance of this interface is high.

Assignment
Unit V

Assignment

Write a Java (JDBC) program to connect with the MySQL database, and Find the Month wise sales

Use the below Database Credentials

Table: Orders

```
+---------------+---------
| order_date  | Sales  |
+---------------+---------
| 2022-05-01 | 1000
| 2022-05-12 | 1230
| 2022-05-11 | 2580
| 2022-08-20 | 3200
| 2022-09-21 | 1045
| 2022-09-11 | 2568
+---------------+---------
```

Part A – Q & A
Unit V

Part A Question & Answers

1. Define JDBC? (K1,CO5)

JDBC stands for Java Database Connectivity. JDBC is a Java Application Programming Interface (API) to connect and execute the query with the database. It provides methods to query and update data in a database, and is oriented toward relational databases.

2. List four types of JDBC Drivers? (K1,CO5)

i. JDBC-ODBC Bridge Driver,
ii. Native Driver,
iii. Network Protocol Driver, and
iv. Thin Driver

3. What is JDBC API? (K1,CO5)

It provides various methods and interfaces for easy communication with the database. It provides two packages as follows, which contain the java SE and Java EE platforms to exhibit WORA(write once run anywhere) capabilities.

4. What is JDBC Driver manager? (K1,CO5)

It loads a database-specific driver in an application to establish a connection with a database. It is used to make a database-specific call to the database to process the user request.

5. What is JDBC-ODBC Bridge Drivers? (K1,CO5)

It connects database drivers to the database. This bridge translates the JDBC method call to the ODBC function call. It makes use of the sun.jdbc.odbc package which includes a native library to access ODBC characteristics.

Part A Question & Answers

6. How JDBC connection pool works? (K2,CO5)

JDBC connection pool, a pool of Connection objects are created at the time the application server starts. These objects are then managed by a pool manager that disperses connections as they are requested by clients and returns them to the pool when it determines the client is finished with the Connection object.

7. Mention the use of PooledConnection? (K1,CO5)

The PooledConnection interface dictates the use of event listeners. These event listeners allow the connection pool manager to capture important connection events, such as attempts by the client application to close the connection.

8. What happens when the driver traps a close-connection event? (K2,CO5)

When the driver traps a close-connection event, it intercedes and performs a pseudo-close operation that merely takes the Connection object, returns it to the pool of available connection, and performs any housekeeping that is necessary.

9. List the advantages and disadvantages of JDBC-ODBC bridge driver? (K1,CO5)

Advantages

i. Easy to use.

ii. Can be easily connected to any database.

Disadvantages

i. Performance degraded because JDBC method call is converted into the ODBC function calls.

ii. The ODBC driver needs to be installed on the client machine.

10. What is native API? (K1,CO5)

The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java.

Part A Question & Answers

11. List the advantages and disadvantages on native API? (K1, CO5)

Advantage

i. Performance upgraded than JDBC-ODBC bridge driver.

Disadvantages

i. The Native driver needs to be installed on the each client machine.

ii. The Vendor client library needs to be installed on client machine.

12. What is the use of network protocol driver? (K2, CO5)

The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.

13. List the advantages and disadvantages of network protocol driver. (K1, CO5)

Advantage

i. No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

Disadvantages

i. Network support is required on client machine.

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

iii. Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

14. What is thin driver? (K2, CO5)

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.

13. List the advantages and disadvantages of thin driver. (K1, CO1)

Advantage

i. Better performance than all other drivers.

ii. No software is required at client side or server side.

Disadvantages

i. Drivers depend on the Database.

Part A Question & Answers

15. List the categories of ResultSet interface? (K1, CO5)

Navigational methods – Used to move the cursor around.

Get methods – Used to view the data in the columns of the current row being pointed by the cursor.

Update methods – Used to update the data in the columns of the current row. The updates can then be updated in the underlying database as well.

16. List the different types of ResultSet? (K1, CO5)

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.

17. Mention the use of PreparedStatement interface? (K1, CO5)

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

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

"?" is passed as the parameter, the value will be set by calling the setter methods of PreparedStatement.

18. What is the need for SQLJ transalator? (K1, CO5)

The SQLJ translator converts Java programs embedded with static SQL statements into pure Java code, which can then be executed through a JDBC driver against the database. Programmers can also perform dynamic SQL access to the database using JDBC features.

Part B – Questions
Unit V

Part B Questions

1. Explain the architecture of JDBC with diagram. (K1, CO5)
2. What is JDBC? List and explain the various components of JDBC with suitable diagram. (K1, CO5)
3. Explain JBDC DataSource interface with an example. (K1, CO5)
4. Explain JDBC connection pool. (K1, CO5)
5. With suitable diagram explain the various JDBC driver types. (K1, CO5)
6. Explain JDBC Resultsets. (K1, CO5)
7. With an example demonstrate the use of PreparedStatement interface. (K2, CO5)
8. Explain embedded SQL. (K1, CO5)

Supportive online Certification courses (NPTEL, Swayam, Coursera, Udemy, etc.,)

Online Certifications

1. https://www.hackerrank.com/skills-verification/java_basic
2. https://www.sololearn.com/Course/Java/
3. https://www.coursera.org/specializations/object-oriented-programming
4. https://www.udemy.com/course/java-the-complete-java-developer-course/ **[Paid]**
5. https://nptel.ac.in/courses/106/105/106105191/106105191/ **[Paid]**
6. https://education.oracle.com/java se 8 fundamentals/courP_3348 **[Paid]**

Real time Applications in day to day life and to Industry

REAL TIME APPLICATIONS : UNIT - V

- Railway Reservation System
- Movie Ticket Booking System
- Online Shopping
- Student Management System

Content Beyond Syllabus

CONTENTS BEYOND SYLLABUS : UNIT – V

JDBC - Streaming ASCII and Binary Data

- A PreparedStatement object has the ability to use input and output streams to supply parameter data. This enables you to place entire files into database columns that can hold large values, such as CLOB and BLOB data types.
- There are following methods, which can be used to stream data –
    1. setAsciiStream() – This method is used to supply large ASCII values.
    2. setCharacterStream() – This method is used to supply large UNICODE values.
    3. setBinaryStream() – This method is used to supply large binary values.
- The setXXXStream() method requires an extra parameter, the file size, besides the parameter placeholder. This parameter informs the driver how much data should be sent to the database using the stream.

```
import java.io.*;
import java.sql.*;
public class TestApplication {
   static final String DB_URL = "jdbc:mysql://localhost/javaprogramming";
   static final String USER = "root";
   static final String PASS = "";
   static final String QUERY = "SELECT Data FROM XML_Data WHERE id=100";
   static final String INSERT_QUERY="INSERT INTO XML_Data VALUES (?,?)";
   static final String CREATE_TABLE_QUERY = "CREATE TABLE XML_Data (id
INTEGER, Data LONG)";
   static final String DROP_TABLE_QUERY = "DROP TABLE XML_Data";
   static final String XML_DATA =
"<Employee><id>100</id><first>Zara</first><last>Ali</last><Salary>10000</
Salary><Dob>18-08-1978</Dob></Employee>";
public static void createXMLTable(Statement stmt) throws SQLException{
      System.out.println("Creating XML_Data table..." );
try{
         stmt.executeUpdate(DROP_TABLE_QUERY);
      }catch(SQLException se){ }
      stmt.executeUpdate(CREATE_TABLE_QUERY);
   }
public static void main(String[] args) {
      // Open a connection
      try(Connection conn = DriverManager.getConnection(DB_URL, USER, PASS);
         Statement stmt = conn.createStatement();
         PreparedStatement pstmt = conn.prepareStatement(INSERT_QUERY);
      ) {
         createXMLTable(stmt);

         ByteArrayInputStream bis = new
ByteArrayInputStream(XML_DATA.getBytes());

         pstmt.setInt(1,100);
         pstmt.setAsciiStream(2,bis,XML_DATA.getBytes().length);
         pstmt.execute();

         //Close input stream
         bis.close();

         ResultSet rs = stmt.executeQuery(QUERY);
         // Get the first row
         if (rs.next ()){
            //Retrieve data from input stream
            InputStream xmlInputStream = rs.getAsciiStream (1);
            int c;
            ByteArrayOutputStream bos = new ByteArrayOutputStream();
            while (( c = xmlInputStream.read ()) != -1)
               bos.write(c);
            //Print results
            System.out.println(bos.toString());
         }
         // Clean-up environment
         rs.close();

      } catch (SQLException | IOException e) {
         e.printStackTrace();
      }
   }
}
```

R.M.K
GROUP OF
INSTITUTIONS

Assessment Schedule (Proposed Date & Actual Date)

Assessment Schedule

Assessment Tool	Proposed Date	Actual Date	Course Outcome
Assessment I	06.05.2023		CO1, CO2
Assessment II	10.06.2023		CO3, CO4
Model	05.07.2023		CO1, CO2, CO3, CO4, CO5

Prescribed Text Books & Reference

Text Books & References

TEXT BOOKS

1. Herbert Schildt, Java:The complete reference, 11th Edition, McGraw Hill Education, 2019
2. Mark Richards, Software Architecture patterns, 2015, O'Reilly.

REFERENCES

1. Cay S. Horstmann, Gary cornell,cornell,"Core Java Volume I Fundamentals", 11th Edition, Prentice Hall, 2019.
2. Paul Deitel, Harvey Deitel, Java SE 8 for programmers, 3rd Edition, Pearson, 2015.
3. Steven Holzner, Java 2 Black book, Dreamtech press, 2011.
4. Timothy Budd, Understanding Object oriented programming with Java, Third Edition, Pearson Education, 2008.
5. https://www.tutorialspoint.com/java/index.htm
6. https://www.javatpoint.com/java-tutorial
7. https://www.w3schools.com/java/
8. https://www.geeksforgeeks.org/java tutorial/
9. https://docs.oracle.com/javase/tutorial/

Mini Project Suggestions

Mini Project

Create a Java Project which gets the details of the students as input. The details include the name, register number, mark details and contact details. The project should permit the user to enter the mark details of the students. The details need to be stored in the database. The user should also be able to generate the report which gives the CGPA of the student.

Thank you