SPACE EXPLORATION

SEMINAR REPORT

SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS

FOR THE DEGREE OF

MASTER OF COMPUTER APPLICATIONS

DEPARTMENT OF MATHEMATICAL AND COMPUTATIONAL SCIENCES
NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA, SURATHKAL

OCTOBER-2023

SUBMITTED BY:

NAME: ROSHAN PRASAD

ROLL NO.: 224CA051

SUBMITTED TO:

Dr. Vishwanath K P

Mr. Karuppasamy S

Mr. Ayyanar K

DECLARATION

I hereby to declare that the seminar report entitled "**SPACE EXPLORATION**" which is being submitted to the National Institute of Technology Karnataka, Surathkal in partial fulfillment of the requirements for Mandatory Learning Course (MLC) of Master of Computer Applications in the department of Mathematical and Computational Sciences, is a bonafide report of the work prepared by me. This material is collected from various sources with utmost care and is based on facts and truth.

ROSHAN PRASAD
224CA051
MCA III SEM
NITK, SURATHKAL.

CERTIFICATE

This is to certify that the P.G. Seminar Report entitled "**SPACE EXPLORATION**" submitted by ROSHAN PRASAD (Roll NO:-224CA051) as the record of the work carried out by him is accepted as the P.G. Seminar Work Report submission in partial fulfilment of the requirements for mandatory learning of Master Of Computer Applications in the departmentof Mathematical And Computational Sciences.

ABSTRACT

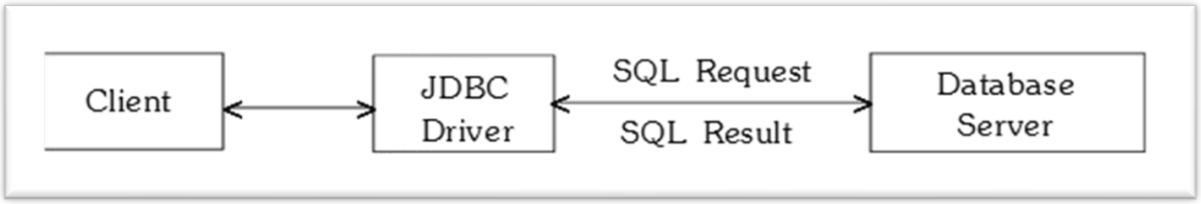
This seminar report on JDBC (Java Database Connectivity) provides a concise overview of its significance and functionality. JDBC acts as a crucial link between Java applications and databases, enabling efficient data interaction. It explains JDBC's architecture, core operations (connecting to databases, executing SQL queries), advanced features (connection pooling, security), and its integration with web technologies. The report also touches on JDBC's relevance in the context of emerging database technologies and cloud solutions. Whether you're a developer or student, this report offers valuable insights into JDBC's role in modern software development.

CONTENTS

1. Introduction

The term JDBC is taken as an acronym for Java Database Connectivity, while Java developers state that it is not so. For a reader, JDBC may be taken to represent Java Database Connectivity. The Java Application Program Interface (API) makes a connection between Java application or applet and a database management system (DBMS or RDBMS). A DBMS has its own structure to organize its data. There are different types of DBMS developed by different vendors. Each DBMS has its own distinct structure. Any application written to access a DBMS of one vendor cannot be used to access the DBMS of another vendor. To solve this problem, Microsoft developed a standard called Open Database Connectivity (ODBC), which is free from any vendor-specific DBMS structure.

A client using ODBC API can send SQL requests to a database server andget the results. A JDBC client does not make a direct link to a DBMS server. A JDBC makes use of ODBC to connect to DBMS ser vers. The bridge between a Java program and a database is a JDBC-ODBC driver



2. JDBC Overview

In the realm of software development, the need for efficient and reliable data management is paramount. Databases serve as the foundation for storing, retrieving, and managing data, making them an integral part of virtually every application. However, to leverage the power of databases, a mechanism is needed to connect programming languages, such as Java, to these data repositories. JDBC, which stands for Java Database Connectivity, emerges as the quintessential solution to bridge this crucial gap.

JDBC is a Java-based API (Application Programming Interface) that provides a standardized way for Java applications to interact with relational databases. It offers a set of Java classes and interfaces that enable developers to perform a wide range of database operations seamlessly. Whether it's retrieving data, updating records, or executing complex queries, JDBC serves as the conduit through which Java applications communicate with databases.

JDBC plays a pivotal role in achieving database connectivity and data manipulation within Java applications. It abstracts the intricacies of different database management systems (DBMS) and provides a uniform interface for developers. This means that, regardless of the underlying DBMS (e.g., MySQL, Oracle, PostgreSQL), developers can write Java code to access and manipulate data without needing to rewrite application logic for each specific database system

JDBC comprises several essential components that facilitate this connectivity:

DriverManager: This class helps in managing a list of database drivers. It is responsible for establishing a connection to the database.

Connection: The Connection interface represents a connection to the database. It is used to establish communication with the database, create Statement objects, and manage transactions.

Statement: The Statement interface is used to execute SQL queries on the database. There are different types of statements, such as Statement, PreparedStatement, and CallableStatement, catering to various SQL execution scenarios.

ResultSet: A ResultSet object represents the result set of a database query. It allows developers to retrieve data from the database and manipulate it within their Java code.

DataSource: JDBC DataSource provides a more efficient and robust way to manage database connections, particularly in enterprise-level applications.

One of the standout features of JDBC is its platform independence. Java applications that use JDBC can run on different operating systems and work with various database systems without modification, as long as the appropriate JDBC driver is available.

3. JDBC Architecture

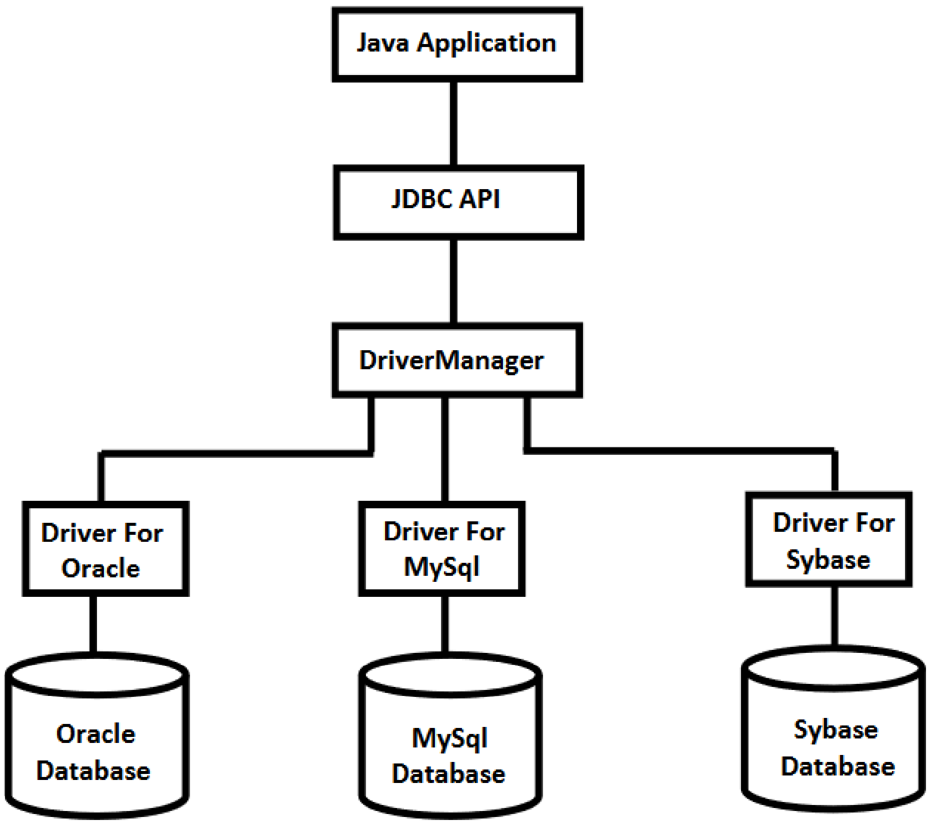
The JDBC API supports both two-tier and three-tier processing models for database access but in general JDBC Architecture consists of two layers:

JDBC API:

This provides the application-to-JDBC Manager connection

JDBC Driver API:

This supports the JDBC Manager-to-Driver Connection.



4.Common JDBC Components

<u>DriverManager:</u>

This class manages a list of database drivers. Matches connection requests from the java application with the proper database driver using communication subprotocol. The first driver that recognizes a certain subprotocol under JDBC will be used to establish a database Connection.

<u>Driver:</u>

This interface handles the communications with the database server. You will interact directly with Driver objects very rarely. Instead, you use DriverManager objects, which manages objects of this type. It also abstracts the details associated with working with Driver objects.

<u>Connection :</u>

This interface with all methods for contacting a database. The connection object represents communication context, i.e., all communication with database is through connection object only

<u>Statement :</u>

You use objects created from this interface to submit the SQL statements to the database. Some derived interfaces accept parameters in addition to executing stored procedures.

<u>ResultSet :</u>

These objects hold data retrieved from a database after you execute an SQL query using Statement objects. It acts as an iterator to allow you to move through its data.

<u>SQLException:</u>

This class handles any errors that occur in a database application.

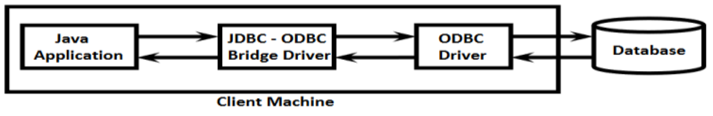
5. Types of Drivers

JDBC drivers implement the defined interfaces in the JDBC API for interacting with your database server.For example, using JDBC drivers enable you to open database connections and to interact with it by sending SQL or database commands then receiving results with Java.

JDBC Drivers Types:

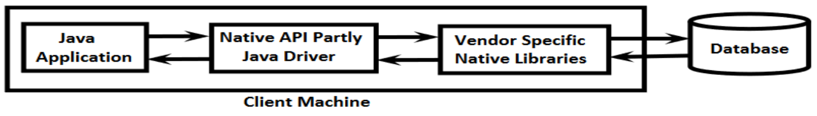
Type 1: JDBC-ODBC Bridge Driver:

The Type-1 driver, also known as the JDBC-ODBC Bridge driver, is a bridge between JDBC and ODBC (Open Database Connectivity). It relies on an ODBC driver to connect to the database. Essentially, it translates JDBC calls into corresponding ODBC calls.



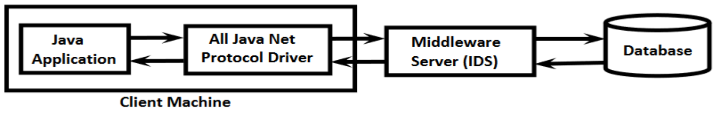
Type 2: JDBC-Native API:

The Type-2 driver uses a database-specific native library to establish a direct connection between the Java application and the database. It communicates with the database through calls to the native library.


Java Application
Native API Partly Java Driver
Vendor Specific Native Libraries
Database
Client Machine

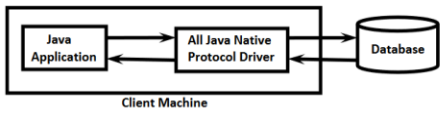
Type 3: JDBC-Network Protocol :

The Type-3 driver, also known as the Network Protocol Driver, employs a middleware or server to mediate communication between the Java application and the database. The Java application communicates with the middleware, which, in turn, connects to the database.



Type 4: Thin Driver:

The Type-4 driver, often referred to as the Thin Driver, establishes a direct connection between the Java application and the database server using a vendor-specific network protocol. It does not rely on native code libraries or middleware.



6. Java Database Connectivity with 5 Steps

Register the driver class :

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

Create the connection object :

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

Create the Statement object :

The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database.

Execute the query :

The executeQuery() method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table.

Close the connection object :

By closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection.

7. Conclusions

In conclusion, JDBC (Java Database Connectivity) emerges as a fundamental technology, bridging the gap between Java applications and relational databases. Its architecture, encompassing various driver types, offers versatility and performance to meet diverse application needs. JDBC's platform independence simplifies cross-system development, while its emphasis on efficient connection management and SQL query execution ensures data integrity. As data-centricity continues to drive modern software development, JDBC remains pivotal, facilitating seamless communication between Java and databases, empowering developers to create robust, data-driven solutions for various domains and industries.

8. References

- Types of drivers and connectivity steps - https://www.javatpoint.com/java-jdbc
- JDBC overview and Architecture - https://www.youtube.com/playlist?list=PLd3UqWTnYXOniKfYRNY___weULTRd9Co0
- Common Compoents - https://www.academia.edu/9529617/JDBC_Architecture