SDM COLLEGE OF ENGINEERING AND TECHNOLOGY

Dhavalagiri, Dharwad-580002, Karnataka State, India.

Email: cse.sdmcet@gmail.com

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

A Report

on

DBMS CTA Major Work

COURSE CODE: 22UCSC501 COURSE TITLE: Database Management Systems
SEMESTER: 5 DIVISION: A
COURSE TEACHER: Dr. U P Kulkarni

[Academic Year- 2024-25]

Date of Submission: 10-12-2024

Submitted By

Sl. No.	Name	USN
1	Archit Shetty	2SD22CS016
2	K K Sagar	2SD22CS039
3	Abhishek Todurkar	2SD22CS006
4	Aditya Teekinavar	2SD22CS132
5	Prajwal Nekar	2SD22CS057
6	K P Prarthan	2SD22CS040

Table of Contents

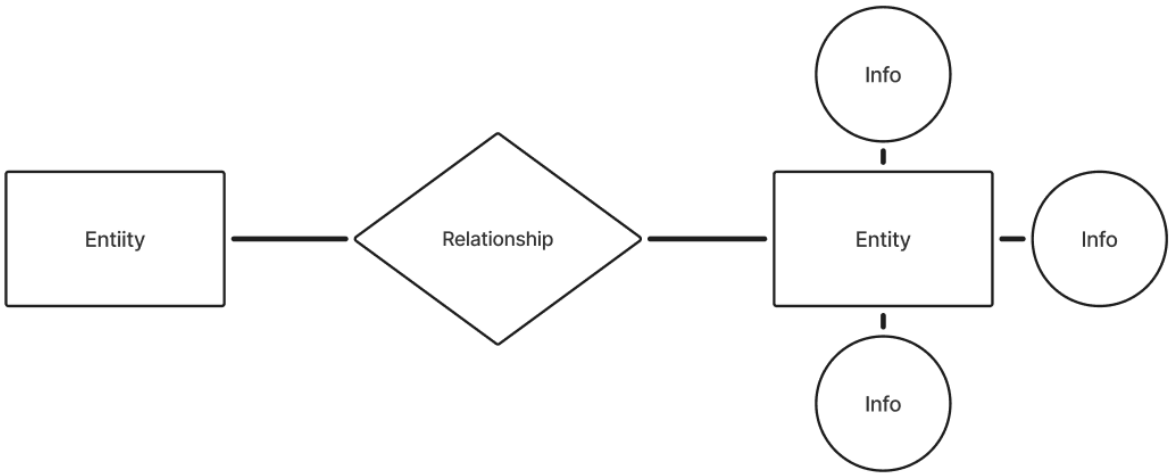
1.Use of software tools to create ER model and create database schema.

Introduction

The objective of this project is to use software tools to design an Entity-Relationship (ER) model and create a corresponding database schema. The ER model helps visualize the data structure, while the database schema implements the structure using SQL commands. This report documents the steps, tools, and processes used to achieve the task.

Entity-Relationship (ER) Diagram

The ER diagram is a graphical representation of the database structure. It defines entities, their attributes, and relationships between entities. This serves as the blueprint for database schema creation.



Tools for Creating ER Diagrams:

1. **Figma:-**

- **Description:** A collaborative interface design tool that can also be used for database modeling through plugins.
- **Key Features:**

- Use plugins like Database Diagram or custom shapes for entities and relationships.
- Collaborative editing for teams working on a single ER diagram.
- Customizable templates and elements for designing complex diagrams.

- **Advantages:**
  - Cloud-based and accessible from anywhere.
  - Great for team-based projects with real-time updates.
- **Limitations:**
  - Requires additional plugins for advanced database modeling.

2. **Lucid chart:-**

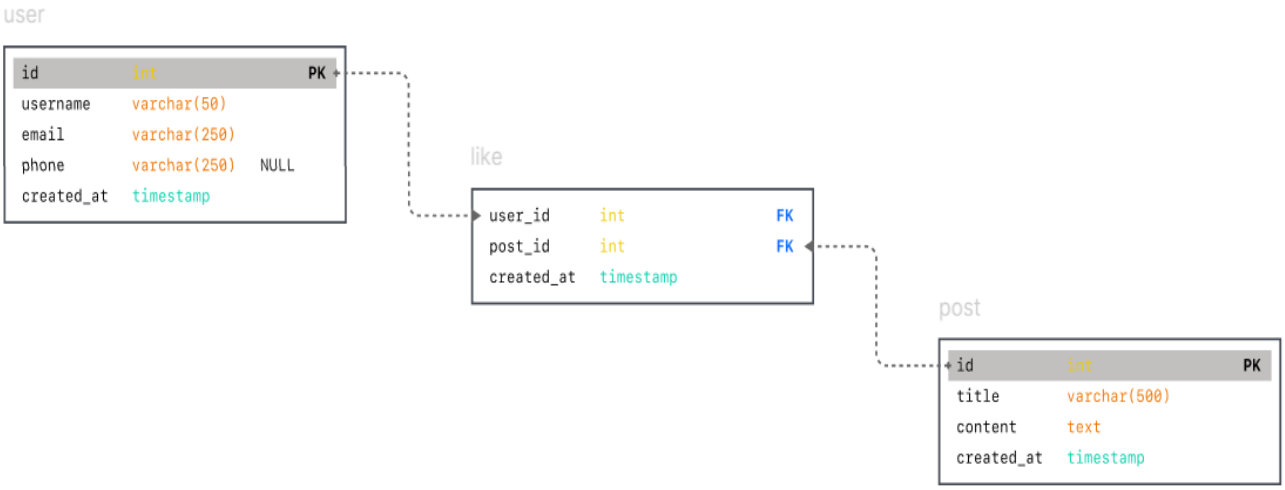
- **Description:** A dedicated online diagramming tool with database modeling capabilities.
- **Key Features:**
  - Drag-and-drop elements for entities, attributes, and relationships.
  - Real-time collaboration and cloud storage.
- **Advantages:**
  - Intuitive interface for rapid design.
  - Includes templates for ER diagrams.
- **Limitations:**
  - Limited features in the free version.

3. **Draw.io / Diagrams.net**

- **Description:** A versatile, open-source diagramming tool for all types of diagrams.
- **Key Features:**
  - Provides basic shapes for entities and relationships.
  - Works both online and offline.
- **Advantages:**
  - Free to use.
  - Simple and lightweight for smaller projects.
- **Limitations:**
  - No direct database integration.

Database Schema

The database schema defines the logical structure of the database, including tables, columns, data types, keys, and relationships. It is derived from the ER diagram and implemented in a relational database management system (RDBMS).



Tools for Creating Database Schema

1. **MySQL Workbench**

- **Description:** A popular IDE for MySQL databases.
- **Key Features:**
  - Schema generation from ER diagrams.
  - SQL editor for manual schema creation.
- **Advantages:**
  - Seamless integration with MySQL.
  - Supports forward and reverse engineering.
- **Limitations:**
  - MySQL-specific.

2. pgAdmin

- **Description:** A feature-rich management tool for PostgreSQL.
- **Key Features:**
  - GUI for creating and modifying schemas.
  - SQL execution capabilities for advanced customization.
- **Advantages:**
  - Free and open-source.
  - Best suited for PostgreSQL databases.
- **Limitations:**
  - PostgreSQL-specific.

3. Oracle SQL Developer

- **Description:** An integrated tool for Oracle database development.
- **Key Features:**
  - Graphical tools for schema creation.
  - Export and import ER diagrams.
- **Advantages:**
  - Comprehensive for enterprise-level projects.
  - Supports data modeling and schema generation.
- **Limitations:**
  - Oracle-specific.

Conclusion

The use of tools like Figma, Lucidchart, and MySQL Workbench simplifies database design and schema creation, enabling faster and more accurate workflows. While Figma excels in collaborative ER diagram creation, MySQL Workbench and pgAdmin provide robust features for schema generation. The selection of tools depends on the project scope, database type, and team requirements.

2.Application development in HLL to demonstrate database connectivity.

To develop an application in a **High-Level Language (HLL)**, such as **Java**, to demonstrate **database connectivity**, you need to establish a connection between your application and the database using **JDBC** (Java Database Connectivity). Here's a simple step-by-step guide to create a **Java application** that connects to a database and performs basic operations like querying and inserting data.

Prerequisites

1. Integrated Development Environment (IDE) :-
   Some of the IDE for java are

   1. Eclipse
   2. IntelliJ IDEA
   3. NetBeans

2. JDBC Driver (for Database Connection) :-

   jdbc driver for oracle: you need the jdbc driver for your database. for oracle, this would be the ojdbc8.jar file.
   download the driver from the oracle jdbc page.

3. Database :-

   Oracle database: for oracle, you need an oracle database installed or access to one like SQLPLUS.

Sample Table For Demo

EMPNO	EMPNAME
39	Raju
35	Prakash
44	Ramesh
36	John
37	Alice
38	David
40	Robert

Steps to connect eclipse ide and JDBC Driver

Step 1:- Right click on the Project folder and select build path and add external archives

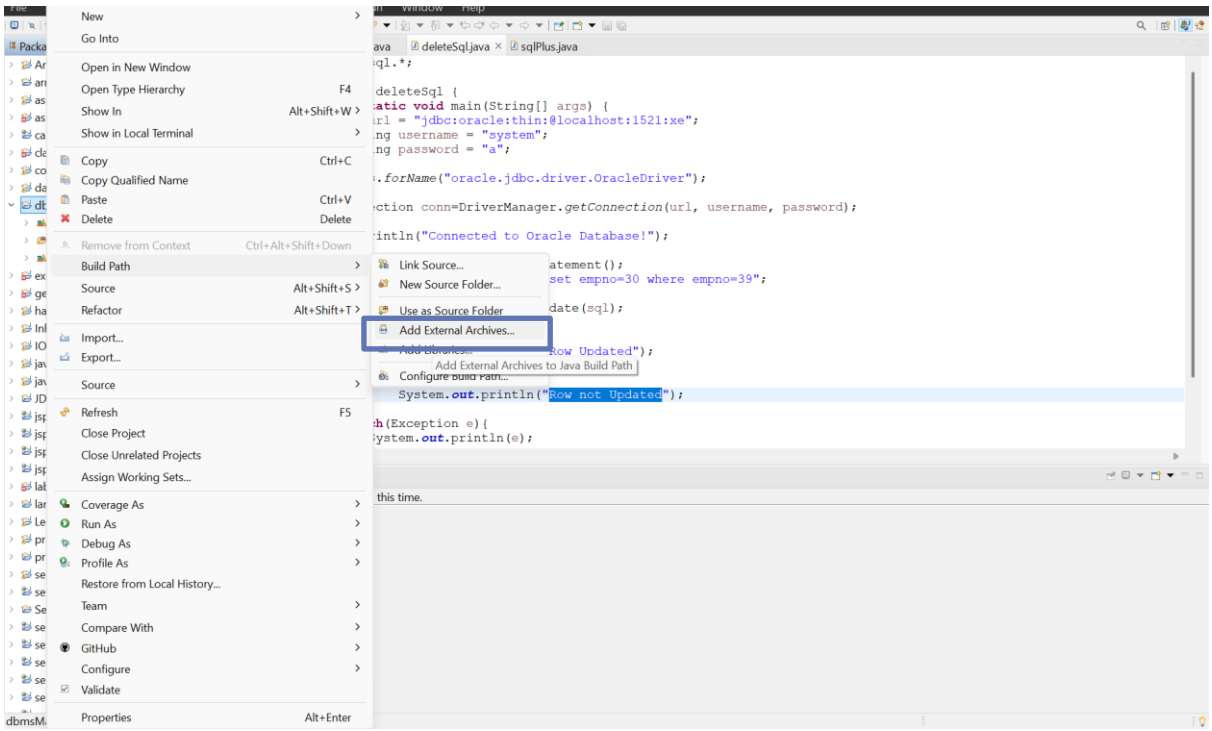


Figure: Connecting eclipse ide

Step 2:- Select ojdbc17.jar

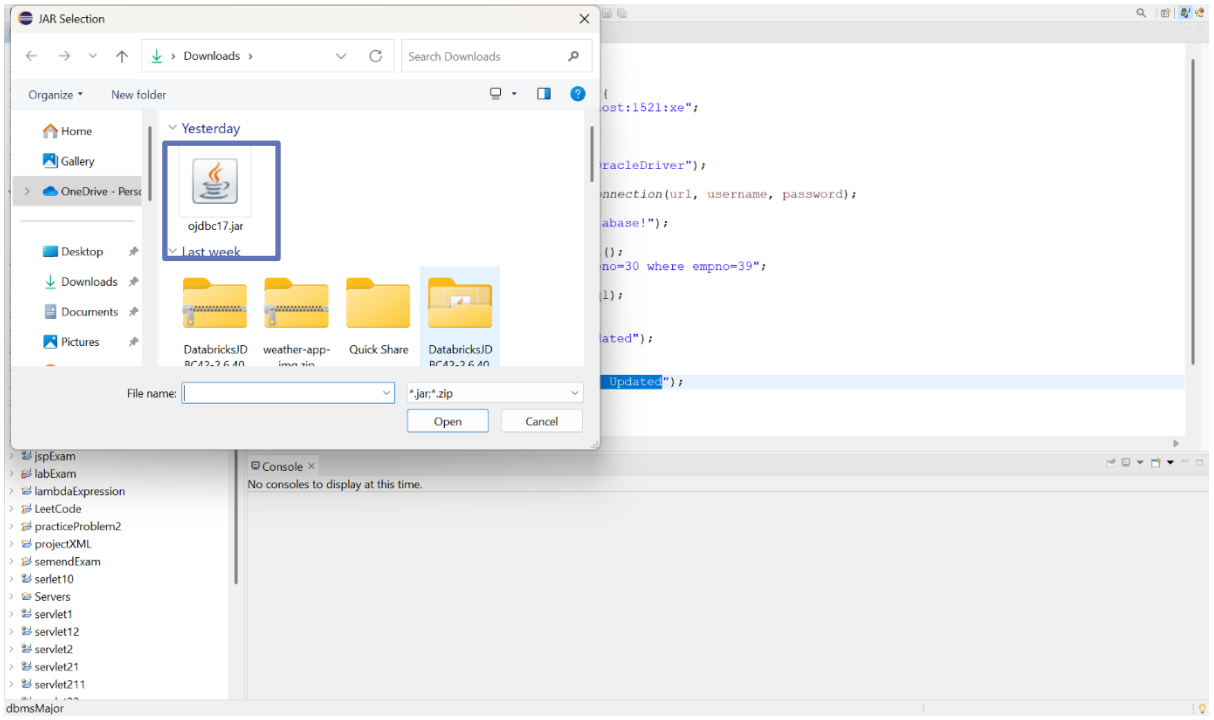
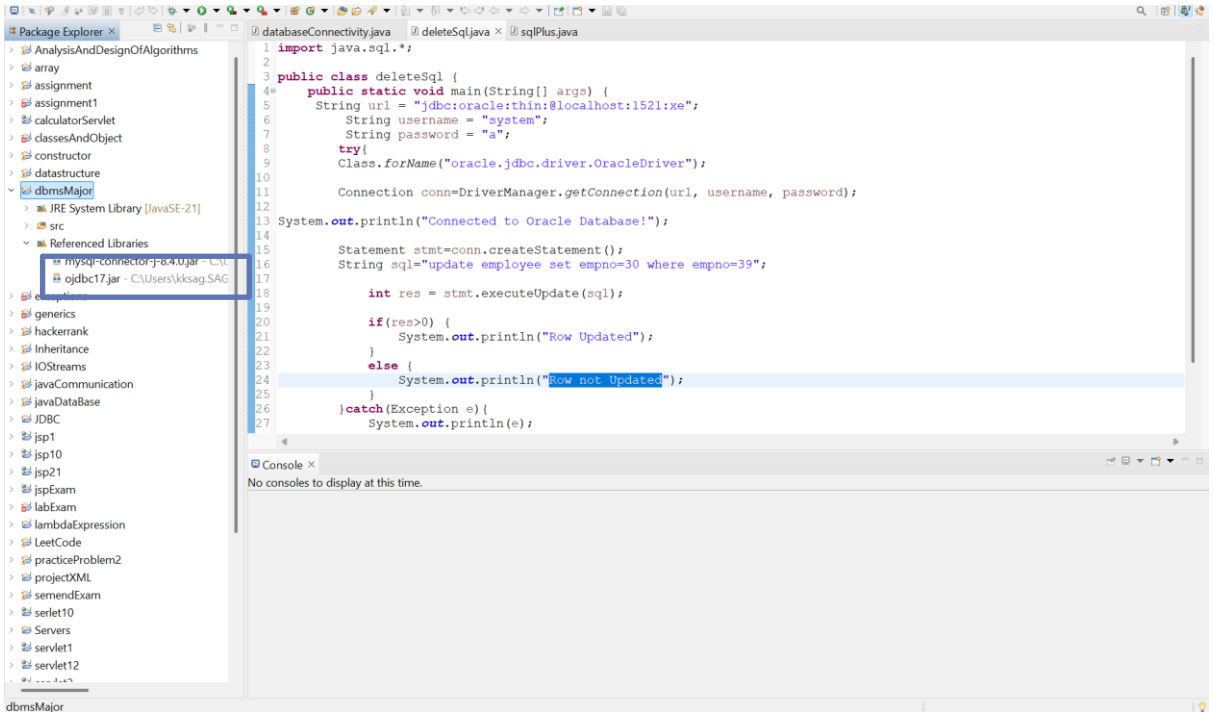


Figure: selecting ojdbc17.jar

Step 3:- In Referenced Libraries we can see the ojdbc17.jar then the connection is established



After adding the Oracle JDBC driver to your project's build path, you can proceed to write the code to **connect** to the database and perform SQL operations

Some of SQL operations are

1. Displaying values in table
2. Inserting row in table
3. Deleting row in table
4. Updating row in table

Displaying employee table values using java

```
import java.sql.*;

public class sqlPlus {
        public static void main(String[] args) {
                String url = "jdbc:oracle:thin:@localhost:1521:xe";
                String username = "system";
                String password = "a";

        try {

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

        Connection connection = DriverManager.getConnection(url, username,
        password);
          System.out.println("Connected to Oracle Database!");

          String query = "SELECT * FROM employee";
          Statement statement = connection.createStatement();
          ResultSet resultSet = statement.executeQuery(query);
```

```
                while (resultSet.next()) {
                   System.out.println(resultSet.getInt(1)+" "+resultSet.getString(2));
                      }

                      connection.close();
                      System.out.println("Database connection closed.");
                   } catch (Exception e) {
          e.printStackTrace();
       }
    }
}
```

Output :-

Connected to Oracle Database!

39 Raju
35 Prakash
44 Ramesh
36 John
37 Alice
38 David
40 Robert
Database connection closed.

Inserting Row into employee table using java

```
import java.sql.*;

public class insertSql {
      public static void main(String[] args) {
    String url = "jdbc:oracle:thin:@localhost:1521:xe";
          String username = "system";
          String password = "a";
              try{
              Class.forName("oracle.jdbc.driver.OracleDriver");

              Connection conn=DriverManager.getConnection(url, username, password);

              System.out.println("Connected to Oracle Database!");

              Statement stmt=conn.createStatement();
              String sql="insert into employee values(46,'Ramesh')";

                    int res = stmt.executeUpdate(sql);

                    if(res>0) {
                          System.out.println("Inserted");
                    }
                    else {
                          System.out.println("Not inserted");
                    }
                    connection.close();
              System.out.println("Database connection closed.");
}

catch(Exception e){
                    System.out.println(e);
              }
      }
```

```
}
```

Output :-

```
Connected to Oracle Database!
Inserted
Database connection closed.
```

```
Updated employee table

Connected to Oracle Database!
39 Raju
35 Prakash
44 Ramesh
36 John
37 Alice
38 David
40 Robert
46 Ramesh
Database connection closed.
```

Deleting Row in employee table using java

```
import java.sql.*;

public class deleteSql {
	public static void main(String[] args) {
	    String url = "jdbc:oracle:thin:@localhost:1521:xe";
	    String username = "system";
	    String password = "a";

	        try{
	        Class.forName("oracle.jdbc.driver.OracleDriver");

	        Connection conn=DriverManager.getConnection(url, username, password);

	        System.out.println("Connected to Oracle Database!");

	        Statement stmt=conn.createStatement();
	        String sql="delete from employee where empno=46";

	            int res = stmt.executeUpdate(sql);
```

```
                if(res>0) {
                        System.out.println("Row deleted ");
                }
                else {
                        System.out.println("Row not deleted ");
                }
                connection.close();
                System.out.println("Database connection closed.");
            }catch(Exception e){
                System.out.println(e);
            }
        }

}
```

Output :-

```
Connected to Oracle Database!
Row deleted
Database connection closed.
```

Updated employee table

```
Connected to Oracle Database!
39 Raju
35 Prakash
44 Ramesh
36 John
37 Alice
38 David
40 Robert
Database connection closed.
```

Updating Row in employee table using java

```
import java.sql.*;

public class updateSql {
    public static void main(String[] args) {
        String url = "jdbc:oracle:thin:@localhost:1521:xe";
        String username = "system";
        String password = "a";
```

```
            try{
            Class.forName("oracle.jdbc.driver.OracleDriver");

            Connection conn=DriverManager.getConnection(url, username, password);

            System.out.println("Connected to Oracle Database!");

            Statement stmt=conn.createStatement();
            String sql="update employee set empno=30 where empno=39";

                  int res = stmt.executeUpdate(sql);

                  if(res>0) {
                        System.out.println("Row Updated");
                  }
                  else {
                        System.out.println("Row not Updated");
                  }
                  connection.close();
                        System.out.println("Database connection closed.");
            }catch(Exception e){
                  System.out.println(e);
            }
      }

}
```

Output :-

```
Connected to Oracle Database!
Row Updated
Database connection closed.
```

Updated employee table

```
Connected to Oracle Database!
30 Raju
35 Prakash
44 Ramesh
36 John
37 Alice
38 David
40 Robert
Database connection closed.
```

3.Application development in HLL to demonstrate BLOB and CLOB

BLOB: A BLOB is used to store large amounts of binary data such as images, audio, video, or other multimedia files in a database. It allows storage of data that cannot be easily represented as simple text.

CLOB: A CLOB is used to store large text data, such as documents, logs, or XML data, in a database. It can store very large strings of text, making it ideal for storing non-binary data in textual form.

Objectives:

- To create a database schema that utilizes BLOB and CLOB columns.
- To develop an application in a high-level language (Python in this case) for handling multimedia and text data.
- To showcase insertion, retrieval, and display of BLOB and CLOB data

Tools and Technologies Used:

- Programming Language: **Python**
- Database Management System: **SQLite**
- Tools/IDE: **Visual Studio Code**
- Libraries:

- sqlite3 for database connectivity.
- Pillow for handling images (optional).

Database Schema:

- **id:** Integer, Primary Key, Auto-Increment
- **file_name:** Text, stores the file name.
- **file_data:** BLOB**,** stores the binary data of the file.
- **description:** CLOB**,** stores a description of the file.

The application performs the following tasks:

1. Insert Data:

- Upload a multimedia file (image/video).

- Add a description for the file.

- Store both in the database.

2**. Retrieve Data:**

- Retrieve the binary data and save it to the disk.

- Display the description associated with the file.

3. **Delete Data (Optional):**

- Remove specific files based on their ID.

CODE IMPLEMENTATION:

SQL Command to Create the Table:

sql

```
CREATE TABLE files (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    file_name TEXT NOT NULL,
    file_data BLOB,
    description CLOB
);
```

Code for Database Setup:

```
python
import sqlite3
# Connect to the SQLite database
conn = sqlite3.connect('example.db')
cursor = conn.cursor()
```

 # Create table

```
cursor.execute('''
CREATE TABLE IF NOT EXISTS files (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    file_name TEXT NOT NULL,
    file_data BLOB,
    description CLOB
)
''')
conn.commit()
print("Table created successfully.")
```

Output:

Table created successfully

Code to Insert Data into BLOB and CLOB Columns:

```
python
def insert_file(file_path, description):
    with open(file_path, 'rb') as file:
        blob_data = file.read()
    cursor.execute('''
    INSERT INTO files (file_name, file_data, description) VALUES (?, ?, ?)
    ''', (file_path, blob_data, description))
    conn.commit()
    print("File inserted successfully.")
```

 # Example usage

insert_file('example_image.jpg', 'This is an example description.')

Input:

- File path: 'example_image.jpg' (ensure the file exists in your project directory).

- Description: "This is an example description."

Output:

-File inserted successfully.

Code to Retrieve and Save BLOB Data:

python

```
def retrieve_file(file_id, output_path):
    cursor.execute('SELECT file_name, file_data FROM files WHERE id = ?', (file_id,))
    row = cursor.fetchone()
    if row:
        file_name, blob_data = row
        with open(output_path, 'wb') as file:
            file.write(blob_data)
        print(f"File '{file_name}' retrieved and saved as '{output_path}'.")
    else:
        print("File not found.")
```

 # Example usage

retrieve_file(1, 'retrieved_image.jpg')

Input:

- File ID: 1 (this corresponds to the first file inserted into the database).

- Output path: 'retrieved_image.jpg'.

Output:

File 'example_image.jpg' retrieved and saved as 'retrieved_image.jpg'.

Code to Display CLOB Data:

python

```
def display_description(file_id):
    cursor.execute('SELECT description FROM files WHERE id = ?', (file_id,))
    row = cursor.fetchone()
    if row:
        print(f"Description: {row[0]}")
    else:
        print("Description not found.")
# Example usage
display_description(1)
```

Input:

- File ID: 1 (the ID of the file to retrieve the description for).

Output:

Description: This is an example description.

Close the Database Connection (Optional):

Once all operations are completed, close the database connection to release resources.

python

```
conn.close()
print("Database connection closed.")
```

Output:

Database connection closed.

Conclusion:

The project successfully demonstrated the use of BLOB and CLOB in managing multimedia and large text data. The application allows seamless insertion and retrieval of files, proving its utility in real-world scenarios.

4. Case study on Architecture and supporting features of Industry relevant RDBMS.

RDBMS: A relational database management system (RDBMS) is a collection of programs and capabilities that enable IT teams and others to create, update, administer and interact with a relational database. A relational database is a type of database that stores related data points.

RDBMS store data in the form of tables, with most commercial relational database management systems using Structured Query Language (SQL) to access the database.

The RDBMS is the most popular database system among organizations. It provides a dependable method of storing and retrieving large amounts of data while offering a combination of system performance and ease of implementation. It's also the basis for modern database systems like MySQL.

Architecture of RDBMS

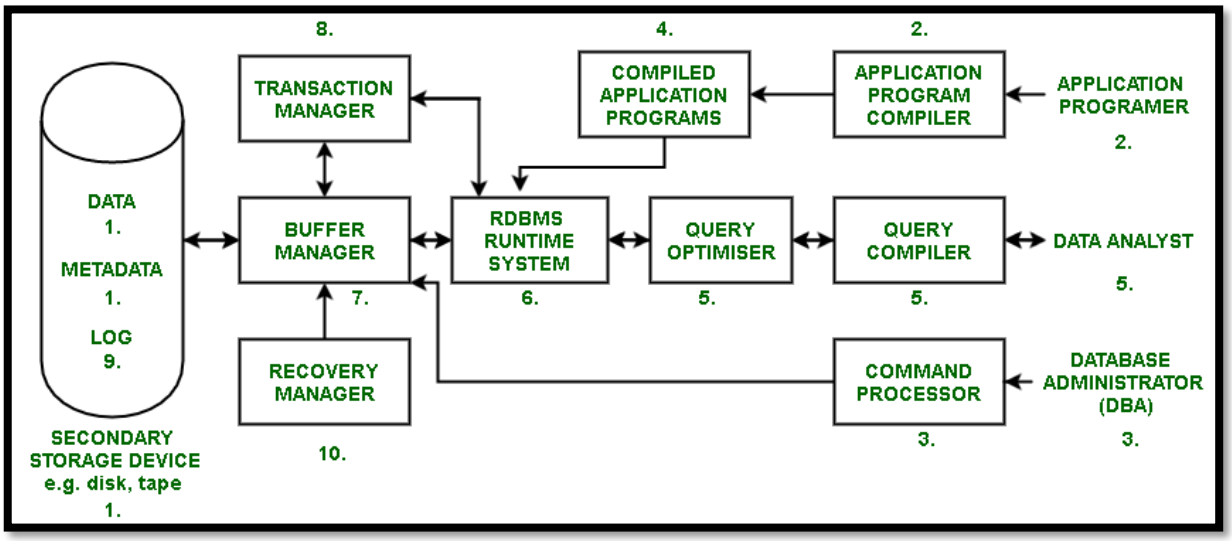


Figure: Architecture of RDBMS

1. All data, data about data (metadata) and logs are stored in the Secondary Storage devices (SSD), such as Disks and Tapes. The programs that are used to do the day-to-day tasks of an enterprise are called Application programs. These programs provide the functionality for the day-to-day operations of the enterprise. They are written in high level languages (HLL) like Java, C etc, which along with the SQL, are used to communicate with the databases.

2. RDBMS has a compiler that converts the SQL commands to lower-level language, processes it and stores it into the secondary storage device.
3. It is the job of Database Administrator (DBA) to set up the structure of the database using command processor. The DDL stands for Data Definition Language and is used by the DBA to create or drop tables, add columns etc. The DBA also uses other commands which are used to set constraints and access controls.
4. Application Programmers compile the applications using a compiler and create executable files (compiled application programs) and then store the data on the secondary storage device.
5. Job of Data Analyst is to use the Query Compiler and Query Optimizer (uses relational properties for executing queries) to manipulate the data in the database.
6. RDBMS Run Time System executes the compiled queries and application programs and also interacts with the transaction manager and buffer manager.
7. Buffer Manager temporarily stores the data of the database in the main memory and uses paging algorithm so that operations can be performed faster and the disk space can be managed.
8. Transaction Manager deals with the principle of either completely doing a task or not doing it at all (Atomicity property). E.g. Suppose a person named X, wants to send money to his sister. He sends the money and system crashes in between. In no case should it happen that he has sent money but his sister has not received it. This is handled by the transaction manager. The transaction manager would either refund the money to X or transfer it to his sister.
9. Log is a system, which records the information about all the transactions, so that whenever a system failure (disk failure, system shut down due to no power etc.) arises, the partial transactions can be undone.
10. Recovery Manager takes control of the system so that it reaches a steady state after failure. The Recovery Manager takes into account the log files and undoes the partial transactions and reflects the complete transaction in the database.

Supporting Features of RDBMS

ACID Properties:

- Atomicity: Ensures that transactions are executed as a single, indivisible unit.
- Consistency: Guarantees that the database remains in a consistent state after a transaction.
- Isolation: Prevents interference between concurrent transactions.
- Durability: Ensures that committed transactions are permanent, even in the event of system failures.

Indexing:

- Creates data structures to speed up data retrieval.
- Indexes are used to efficiently locate specific rows based on specific columns.

Stored Procedures and Functions:

- Precompiled code blocks that can be executed to perform complex database operations.
- Improve performance and code reusability.

Triggers:

- Automatically execute predefined actions in response to specific events (e.g., insert, update, delete).
- Used for data validation, auditing, and cascading updates.

Views:

- Virtual tables based on the result-set of an SQL statement.
- Simplify complex queries and provide a customized view of the data.

Security:

- Role-based access control (RBAC) to manage user permissions.
- Encryption to protect sensitive data.
- Auditing to track database activity.

Industry Relevant RDBMS:

Oracle Database: A comprehensive database solution known for its scalability, performance, and security features.
Microsoft SQL Server: A versatile database platform that integrates well with Microsoft technologies.
MySQL: A popular open-source database that is widely used in web applications.
PostgreSQL: A powerful open-source object-relational database system known for its advanced features.
IBM DB2: A high-performance database platform used in enterprise applications.

Selection of the right RDBMS:

- **Performance:** Scalability, query optimization, and indexing capabilities.
- **Security:** Encryption, authentication, and authorization features.
- **High Availability:** Fault tolerance and disaster recovery mechanisms.
- **Cost:** Licensing fees and maintenance costs.
- **Ease of Use:** User-friendly interface and management tools.

Challenges in using RDBMS:

1. **Scalability**: Horizontal scaling in RDBMS can be challenging compared to NoSQL databases.
2. **Cost**: Enterprise-grade RDBMS like Oracle come with significant licensing and maintenance costs.
3. **Complexity**: Advanced features (e.g., RAC, partitioning) require specialized knowledge for implementation.

5.Study of CODD's rule.

Introduction

Codd's 12 Rules define the essential requirements for a database to qualify as a Relational Database Management System (RDBMS). These rules ensure that data is stored in a structured, accessible, and consistent manner using tables. They also emphasize logical independence, robust integrity constraints, and standardized language support, providing flexibility and reliability in data management.

Rule 0: Foundation Rule

The system must manage data using relational capabilities.
Example: Using MySQL to manage relational data.

Sql

```
CREATE TABLE EMPLOYEES (
    ID INT PRIMARY KEY,
    Name VARCHAR(50),
    Department VARCHAR(50)
);
```

Output: The table EMPLOYEES is created to manage relational data in rows and columns.

Rule 1: The Information Rule

All information should be stored in tables.
Example:

Sql

```
SELECT * FROM EMPLOYEES;
```

Output:

ID	Name	Department

------ ------------------- -----------------

1 PRajwal Nekar HR

2 Archit IT

Rule 2: Guaranteed Access Rule

Every piece of data should be accessible using a combination of table name, primary key, and column name.
Example: Fetch the department of employee with ID 2:

Sql

SELECT Department FROM EMPLOYEES WHERE ID = 2;

Output:

Department

IT

Rule 3: Systematic Treatment of NULL Values

NULL must be used to represent missing or inapplicable data.
Example: Insert a row with a missing phone number:

Sql

INSERT INTO EMPLOYEES (ID, Name, Department, Phone)

VALUES (3, 'Prarthan', 'Finance', NULL);

SELECT * FROM EMPLOYEES;

Output

ID Name Department Phone

---------- -------------------- ----------------- ---------------

1	PRajwal Nekar	HR	NULL
2	Archit	IT	9876546542
3	Prarthan	Finance	NULL

Rule 4: Active Online Catalog

Metadata must also be stored in tables and accessible using SQL.
Example: Query the database catalog for table information:

Sql

```
SELECT TABLE_NAME, COLUMN_NAME
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME = 'EMPLOYEES';
```

Output:

TABLE_NAME	COLUMN_NAME
EMPLOYEES	ID
EMPLOYEES	Name
EMPLOYEES	Department
EMPLOYEES	Phone

Rule 5: Comprehensive Data Sub-Language Rule

The database must use a single language (e.g., SQL) for all operations.
Example: Perform operations using SQL:

- **Data Definition Language(DDL)**:

Sql

CREATE TABLE DEPARTMENTS (DeptID INT PRIMARY KEY, DeptName VARCHAR(50));

- **Data Manipulation Language (DML)**:

  Sql

  INSERT INTO DEPARTMENTS VALUES (1, 'HR');

  SELECT * FROM DEPARTMENTS;

- **Data Control Language(DCL)**:

  Sql

  GRANT SELECT ON DEPARTMENTS TO user1;

  Output:

  DeptID DeptName

  --------- ----------

  1 HR

Rule 6: The View Updating Rule

Views should be updatable when possible.
Example:

Sql

CREATE VIEW HR_EMPLOYEES AS

SELECT * FROM EMPLOYEES WHERE Department = 'HR';

UPDATE HR_EMPLOYEES

SET Name = ‘PRajwal Nekar'

WHERE Name = 'Prajwal Nekar’;

SELECT * FROM EMPLOYEES;

Output:

ID	Name	Department	Phone	Salary
1	Prajwal Nekar	Admin	NULL	NULL
2	Archit	IT	9876546542	NULL
3	Prarthan	Finance	NULL	NULL

Rule 7: High-Level Insert, Update, and Delete

The system must allow set-level operations.

Example:

Sql

UPDATE EMPLOYEES

SET Department = 'Admin'

WHERE Department = 'HR';

SELECT * FROM EMPLOYEES;

Output:

ID	Name	Department
1	Prajwal Nekar	HR
2	Archit Shetty	IT
3	Prarthan	Finance

Rule 8: Physical Data Independence

Data storage changes should not affect queries.
Example: If employee data is moved to a partitioned structure, this query still works:

Sql

```
SELECT * FROM EMPLOYEES;
```

Output:

```
ID Name            Department     Phone         Salary
---- ------------------- ----------------- -------------- ----------
  1 Prajwal Nekar    Admin          NULL        NULL
  2 Archit           IT             9876546542 NULL
  3 Prarthan         Finance        NULL        NULL
```

Rule 9: Logical Data Independence

Schema changes should not affect applications.
Example: Add a column Salary to the EMPLOYEES table:

Sql

```
ALTER TABLE EMPLOYEES ADD Salary INT;

SELECT * FROM EMPLOYEES;
```

Output:

```
ID Name            Department     Salary
---- ------------------ ----------------- -----------
  1 Prajwal Nekar   HR             NULL
  2 Archit          IT             NULL
```

3 Prarthan Finance NULL

Rule 10: Integrity Independence

The database should enforce integrity constraints.
Example: Define a foreign key constraint:

Sql

```
ALTER TABLE EMPLOYEES

ADD CONSTRAINT FK_Dept FOREIGN KEY (Department) REFERENCES
DEPARTMENTS(DeptName);
```

Rule 11: Distribution Independence

The system should work regardless of data distribution.
Example: Whether EMPLOYEES is stored on one server or partitioned across multiple, this query retrieves all records:

Sql

```
SELECT * FROM EMPLOYEES;
```

Output:

ID	Name	Department	Phone	Salary
1	Prajwal Nekar	Admin	NULL	NULL
2	Archit	IT	9876546542	NULL
3	Prarthan	Finance	NULL	NULL

Rule 12: Non-Subversion Rule

Constraints should not be bypassed via low-level operations.
Example: Attempting to insert duplicate primary key:

Sql

INSERT INTO EMPLOYEES (ID, Name, Department)

VALUES (1, 'Duplicate', 'Admin');

Output:

ERROR at line 1:

ORA-00001: unique constraint (SYSTEM.EMPLOYEE_PK_VIOLATION) violated

6.Demonstration of NOSQL and comparison with RDBMS

NoSQL (short for "Not Only SQL") is a category of database management systems designed to handle a wide variety of data types and scalable workloads that traditional relational databases (RDBMS) may struggle with. Unlike relational databases, which use structured schemas and SQL (Structured Query Language) for queries, NoSQL databases often use flexible schema designs and diverse query languages suited to their specific use cases.

NoSQL databases are highly versatile and are used in many modern applications, such as content management, IoT, real-time data processing, and social networking platforms.

Types of NOSQL databases: -

1. Document-Oriented Databases

- Store data as documents (e.g., JSON, BSON). Each document can have different fields, making them flexible.
- **Examples:** MongoDB, CouchDB.

2.Key-Value Stores

- Store data as key-value pairs. Keys are unique identifiers, and values can be simple or complex objects.
- **Examples:** Redis, DynamoDB.

3.Column-Family Stores

- Organize data into columns rather than rows, optimized for large-scale queries.
- **Examples:** Cassandra, HBase.

4.Graph Databases

- Store data in the form of nodes and edges, representing entities and their relationships.
- **Examples:** Neo4j, Amazon Neptune.

Demonstation of NOSQL :-

1.Document -Oriented Databases:- MongoDB stores data in a JSON-like format called BSON.The basic operations are :

Insert Data:

```
db.users.insertOne({
  name: "Alice",
   age: 30,
   skills: ["JavaScript", "Python"],
   address: { city: "New York", zip: "10001" }
});
```

Output:-

```
{
  "acknowledged": true,
  "insertedId": ObjectId("64970c8c79a22c80c5f4b0a2")
}
```

Read Data:

```
db.users.find();
```

Output:

```
[
   {
```

```
    "_id": ObjectId("64970c8c79a22c80c5f4b0a2"),
    "name": "Alice",
    "age": 30,
    "skills": ["JavaScript", "Python"],
    "address": { "city": "New York", "zip": "10001" }
  }
]
```

Update Data:

```
db.users.updateOne(
  { name: "Alice" },
  { $set: { age: 31 } }
);
```

Output :

```
{
  "acknowledged": true,
  "matchedCount": 1,
  "modifiedCount": 1
}
```

Delete Data:-

```
db.users.deleteOne({ name: "Alice" });
```

Output:

```
{
   "acknowledged": true,
   "deletedCount": 1
}
```

2.Redis (Key-value store):

Set and Get a Key-Value Pair:

```
SET user:1 "Alice"
GET user:1
```

Output:

```
OK
"Alice"
```

Store Complex Data (Hash Map):

```
HSET user:1 name "Alice" age 30
HGETALL user:1
```

Output:

```
(integer) 2
1) "name"
2) "Alice"
3) "age"
```

4) "30"

Increment a Counter

INCR page_views

GET page_views

Output:

(integer) 1

"1"

3.Cassandra(Column-Family Store):

Create a Keyspace and Table

```
CREATE KEYSPACE demo WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 1};

USE demo;

CREATE TABLE users (
   id UUID PRIMARY KEY,
   name TEXT,
   age INT
);
```

Output:

Keyspace created.

Using keyspace demo.

Table created.

Insert Data

INSERT INTO users (id, name, age) VALUES (uuid(), 'Alice', 30);

Output:

INSERT statement succeeded.

Query Data

SELECT * FROM users;

Output:

id	name	age
d5b7c0a0-37c5-11ec-8d3d-0242ac130003	Alice	30

4.Neo4j Outputs (Graph Database):

Create Nodes and Relationships

CREATE (a:Person {name: "Alice", age: 30})

CREATE (b:Person {name: "Bob", age: 35})

CREATE (a)-[:FRIEND]->(b);

Output:

Nodes created: 2

Relationships created: 1

Properties set: 4

Query the Graph

MATCH (a:Person)-[:FRIEND]->(b:Person) RETURN a, b;

Output:

a b

--- ---

{name: "Alice", age: 30} {name: "Bob", age: 35}

Comparison OF NOSQL with RDBMS:

Feature	**NoSQL**	**RDBMS (Relational Databases)**
Data Model	Flexible, schema-less (e.g., JSON, key-value).	Fixed schema (tables with rows and columns).
Data Structure	Works well with unstructured or semi-structured data.	Best for structured data.
Schema	No predefined schema; data structure can evolve.	Requires a predefined schema; structure must be defined upfront.
Joins	No or limited support for joins.	Supports complex joins and relationships.
Query Language	Query language varies (e.g., MongoDB's query API, Cypher for graphs).	Uses SQL (Structured Query Language).
Performance	Faster for simple operations on large datasets.	Can be slower for handling unstructured data or high traffic.
Transaction Support	Weak support for ACID transactions (depends on the database).	Strong support for ACID transactions (atomicity, consistency, isolation, durability).

Scenarios	Ideal for big data, real-time analytics, IoT, and unstructured data.	Ideal for financial systems, inventory management, and systems with clear relationships.
Examples	MongoDB, Redis, Cassandra, Neo4j.	MySQL, PostgreSQL, Oracle, SQL Server.
Cost	Often open-source and cheaper to scale.	Licensing and hardware can be expensive.