

# CSCI 5408 DATA MANAGEMENT AND WAREHOUSING

## ASSIGNMENT - 1

Banner ID: B00955671

GitLab Assignment Link:

[https://git.cs.dal.ca/ssavaliya/csci5408\\_f23\\_b00955671\\_sumit\\_savaliya/-/tree/main/A1](https://git.cs.dal.ca/ssavaliya/csci5408_f23_b00955671_sumit_savaliya/-/tree/main/A1)

## Table of Contents

<b>Review:</b> Transaction recovery in federated distributed database systems .....	3
<b>Review:</b> Research on the Network Management System of Students' Art Works based on DB2 Database..	5
<b>Task:</b> Prototype of a light-weight multiuser Database System using Java.....	7
<b>Task A:</b> Multiuser Authentication with Captcha.....	8
<b>Task B:</b> Query Processor .....	10
<b>Task C:</b> Single Transaction Management .....	12
References: .....	12

## Review: Transaction recovery in federated distributed database systems

The paper [1] primarily focusses on Transaction recovery in a federated distributed database system. These database systems are usually autonomous and heterogenous. The objective is to create a more dependable and effective system for handling transactions in an environment where errors are common. This kind of system architecture leads to concurrency and recovery issues. This paper deals explains various ways to tackle and recover from such failures.

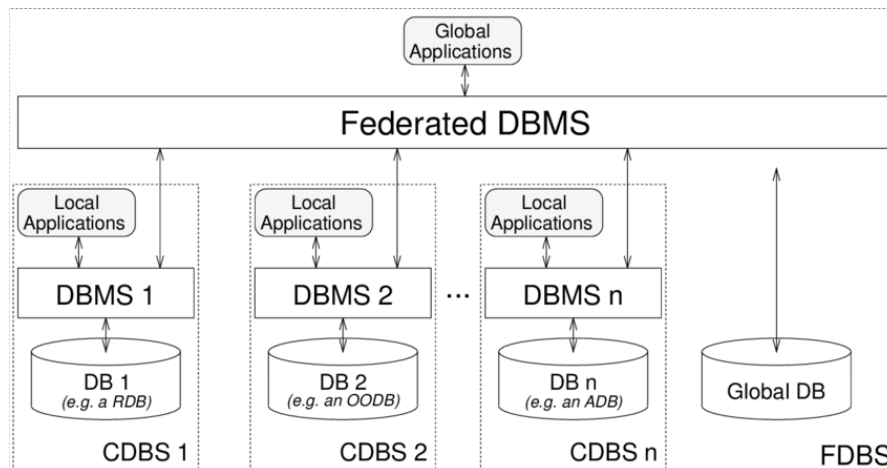


Figure 1: Architecture of FDS [2].

This study's focus is on the challenging the issue of handling transactions in federated database systems. These systems have several diverse, perhaps independent databases connected by a network. As such, they have difficulties with data consistency, failure recovery, and concurrency management.

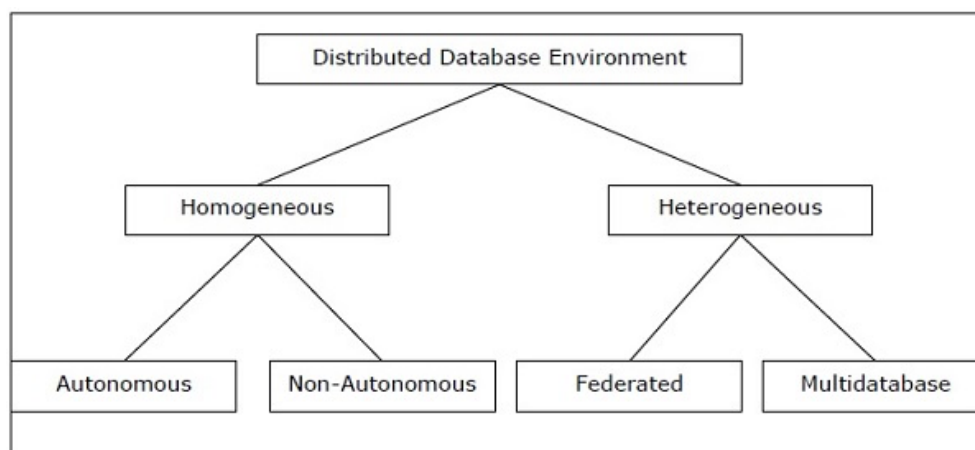


Figure 2: Database Environments [3].

The writers examine topics including replication, data fragmentation, heterogeneous databases, and distributed databases' benefits—such increased scalability, availability, and reliability—while citing relevant research. This creates the context for the study that is being done.

For Federated Distributed Database Systems (FDDS), several transaction management techniques have been developed to ensure error-free operation. Each of these algorithms has certain limitations that impact various aspects of local autonomy:

One way to tackle this issue is to employ two different sets of data in local databases: one for updates by users globally and the other for modifications made by local users. Furthermore, they make it impossible for users to access locally updated data from anywhere in the world. Local DBMSs are not allowed to stop a

transaction once all its activities have been completed under certain circumstances. Sometimes control autonomy is given up, which leaves local transactions unable to carry out certain operations. These approaches often restrict the kind of concurrency control strategies that local Database Management Systems (DBMSs) can employ (*Look at Figure 1*). There are limitations on the concurrency control strategies that can be applied to each FDBS recovery method that has been reported in the literature.

In the context of this paper, the following elements contribute to the research's success:

**Well-Defined Problem:** The paper clearly identifies the problem that federated distributed database systems have with transaction recovery. It highlights the unique challenges these systems face, such as node failures and communication issues.

**Algorithm Development:** The paper presents a unique transaction recovery approach for federated systems to address the mentioned problem. The technique handles issues with concurrency control, data synchronisation, and failure recovery.

**Real-World Application:** The authors test and implement their algorithm within the context of a bank transfer transaction to demonstrate how their technique might operate in a real-world scenario.

**References to Prior Research:** By referencing pertinent papers, the paper builds on earlier research on distributed databases. This suggests that the authors are cognizant of the corpus of prior work.

**Design and Implementation:** The authors clarified the thought process behind the creation and use of the recommended algorithm by providing a helpful guide for academics and industry experts in the field.

**Limitations:** The paper has some benefits, but it also has several serious drawbacks:

**Limited evaluation:** The report mentions evaluating the approach in a bank transfer scenario, but it doesn't go into detail about performance standards and assessment results. A more extensive evaluation may have improved the work.

**Complexity:** The paper's methodology may make it challenging to develop and maintain real-world systems. It's probable that the article discussed potential issues that may have arisen from this complexity.

**Lack of Comparison:** The paper does not compare the proposed algorithm to any existing methods. It would have been beneficial to compare the advantages and special qualities of the proposed algorithm with those of other approaches.

## **Conclusion:**

The paper doesn't provide a whole new algorithm. Instead, it analyses and implements an improved architecture and algorithm for distributed database systems that are federated, with a focus on transaction recovery. The key contribution of the study is the addition of a sync coordinator to synchronise partitioned global and local databases upon transaction commit, hence improving the architecture of federated database systems. The sync coordinator is an essential component that builds upon established transaction management concepts and methods, refining and implementing them to enhance transaction recovery in a federated environment.

## Review: Research on the Network Management System of Students' Art Works based on DB2 Database

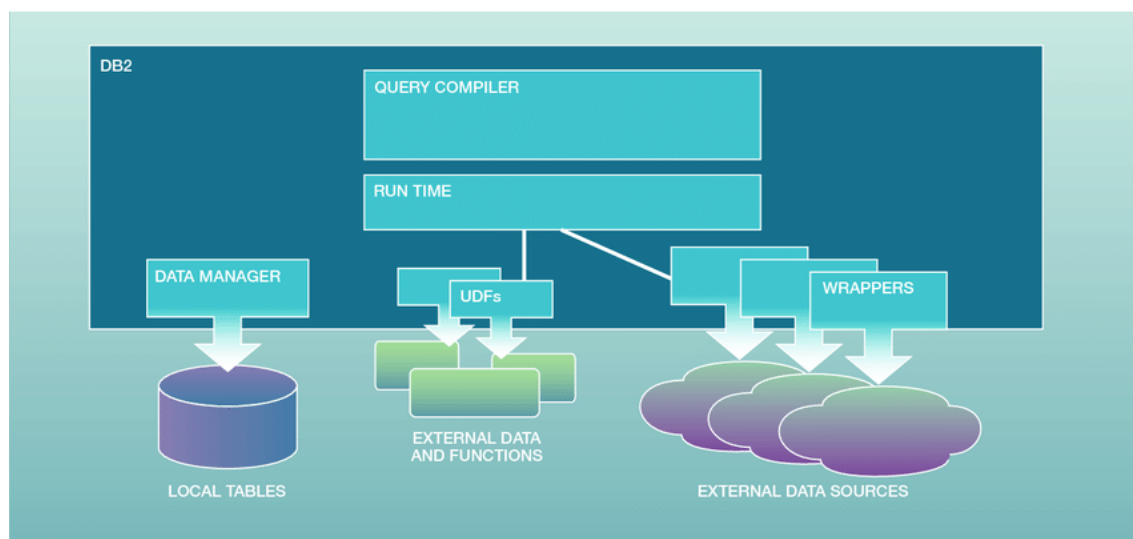


Figure 3: DB2 Database Architecture [4].

This article [5] describes a project that built a network management system for student artwork using the DB2 database technology. The study highlights the importance of Database2, a solid choice for managing enormous volumes of student artwork, which is well-known for its parallelism and multi-platform capabilities. The primary goal is to transform these pieces of art into invaluable teaching resources that support both student learning and productive teacher feedback.

The article discusses the components of the DB2 database, which together provide a variety of capabilities and interactions: the DB2 Optimizer, Engine, Extender, and Connect.

The two steps in the database design process are conceptual structure design and logical structure design. The conceptual design of the system consists of three main parts: student information, instructor information, and artwork. Everybody has a unique bond with every other. During the logical design phase, these entities are converted into database tables with matching attributes.

The study emphasises the unique nature of art education—pupils pick up knowledge from the examples set by their teachers, therefore effective instruction is crucial. It also highlights how important it is to teach assessment in the art education process. The DB2-based online management system has the potential to significantly improve art education in HE, according to the paper's conclusion, by improving teaching and evaluation procedures and streamlining the administration of student artworks.

The major goal of this research paper is to develop a helpful system that uses the DB2 database to manage student artwork in order to raise the bar for art education in colleges and institutions.

## **Critical Analysis**

The creation of a Network Management System for student artwork utilising the DB2 database is the focus of this article, with particular emphasis on the system's possible uses in art education. The main issue raised is the ineffective handling of numerous student artworks at colleges nowadays, which mostly entails the use of resources and paper. The study suggests using the DB2 database to improve the administration of art-related works. It offers a digital solution that makes it easier for teachers to assess the works, for students to learn, and for these works to be transformed into priceless teaching tools.

In the first half of the essay, DB2 is briefly introduced with an emphasis on its salient characteristics, potential applications, and large-scale distributed data management capabilities. The ability of DB2 to handle massive amounts of data and run queries simultaneously demonstrates how relevant it is to the current issue. Nevertheless, the paper is devoid of an extensive literature analysis that would provide background information on the subject, show how this study fits into the existing body of knowledge on managing databases, and provide examples of how art education is taught.

The database's creative and sensible structural design, which offers a structured and orderly framework for categorising student artworks, is a clear indication of the study's effectiveness.

Nevertheless, the work has a number of problems. The lack of a comprehensive literature review, which would have provided a more comprehensive evaluation of the methods and solutions currently employed in this sector, is one major disadvantage. Moreover, the study provides no empirical evidence or case studies from the actual world to bolster the applicability and efficacy of the recommended approach. It is important to confirm the efficacy of this research through empirical data and user feedback.

The study does not delve into the technical aspects of developing the web-based management system itself, leaving the reader with questions regarding the actual software development process. It outlines the intellectual architecture of the database but doesn't explain how it will be integrated into a functional solution.

The essay may be improved by a more in-depth analysis of the user experiences and educational implications of integrating this technology into art schools. Moreover, adding pertinent information and enhancing the paper's value would have been possible by contrasting and comparing similar systems or solutions.

The study concludes with a feasible solution for handling student artwork in an educational setting. However, it may greatly increase the impact of its study by doing a more thorough examination of the literature, providing specific proof of the system's effectiveness, and delving further into the nuances of its real-world application.

# Task: Prototype of a light-weight multiuser Database System using Java

## Design Principles:

Here are the design principles [6] followed in the provided code files:

### QueryProcessor.java:

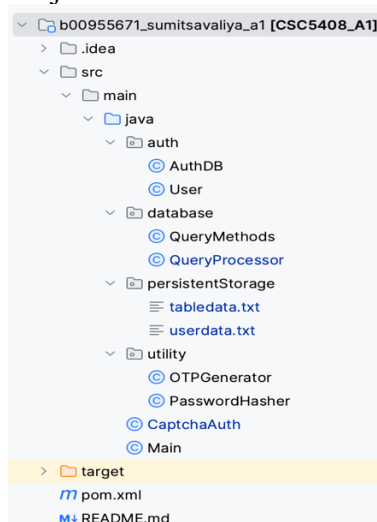
- **Single Responsibility Principle (SRP):** The QueryProcessor class is responsible for processing and executing SQL queries. It delegates the actual database operations to the QueryMethods class, following the SRP.
- **Open/Closed Principle (OCP):** The QueryProcessor class is open for extension (e.g., you can add new query types) but closed for modification, as it doesn't need to be altered to support new query types.
- **Interface Segregation Principle (ISP):** Although the code doesn't explicitly show interfaces, the separation of concerns between QueryProcessor and QueryMethods follows the ISP, ensuring that each class has a specific and limited set of responsibilities.

### QueryMethods.java:

- **Single Responsibility Principle (SRP):** The QueryMethods class handles database operations, including creating tables, inserting data, loading and saving data to a file, selecting all data from a table, deleting tables, and updating data. Each method in this class follows SRP.
- **Dependency Injection:** The QueryMethods class is initialized with a tables map and loads data from a file, demonstrating a form of dependency injection. This makes it more flexible and testable.
- **Encapsulation:** Private methods like saveTableDataToFile and handleCreateTableQuery encapsulate specific functionality, making the code more modular and easier to maintain.
- **Comments and Documentation:** The code includes comments and JavaDocs, which promote readability and understanding of the code.
- **Error Handling:** The code appropriately handles IOExceptions when loading and saving data to a file, ensuring robust error management.

Overall, the code follows important design principles such as SRP, OCP, ISP, dependency injection, and encapsulation, which contribute to code maintainability and extensibility. It also includes error handling and documentation for clarity and robustness.

## Project Structure:



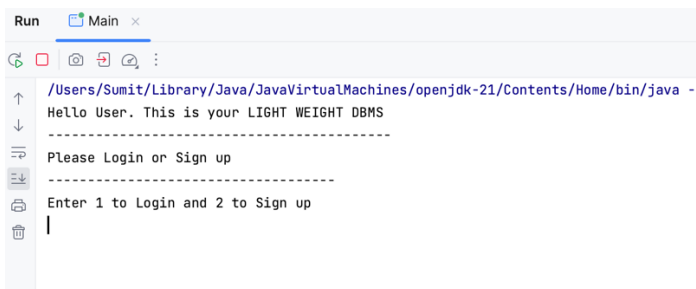
## Task A: Multiuser Authentication with Captcha.

Successfully implemented user authentication and random captcha generation/verification in the code. Below are features implemented:

1. Sign up feature for user.
2. Login feature for user.
3. Hashed password using MD5 to increase security.
4. Stored the data in text file to retain persistent storage.

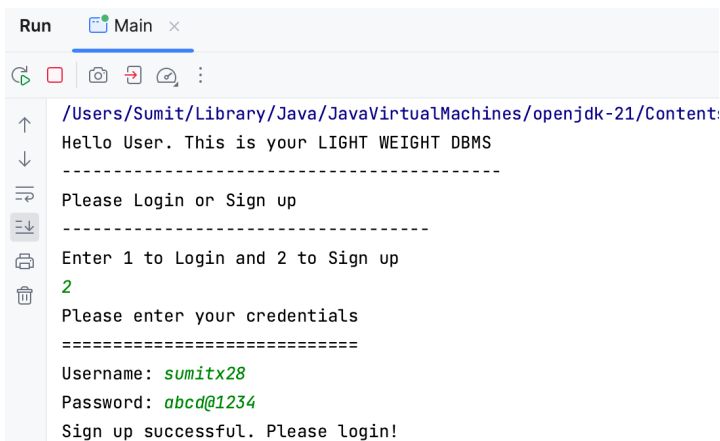
Console:

1. On Starting the app, The user is prompted to either login or sign up.



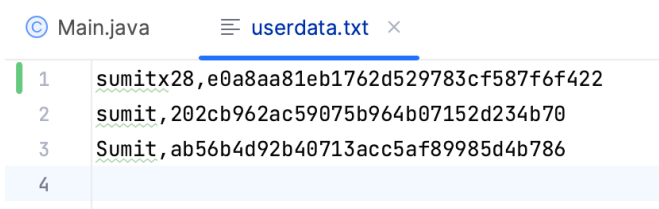
```
Run Main x
/Users/Sumit/Library/Java/JavaVirtualMachines/openjdk-21/Contents/Home/bin/java -
Hello User. This is your LIGHT WEIGHT DBMS
-----
Please Login or Sign up
-----
Enter 1 to Login and 2 to Sign up
|
```

2. Input 2 for sign up



```
Run Main x
/Users/Sumit/Library/Java/JavaVirtualMachines/openjdk-21/Content:
Hello User. This is your LIGHT WEIGHT DBMS
-----
Please Login or Sign up
-----
Enter 1 to Login and 2 to Sign up
2
Please enter your credentials
=====
Username: sumitx28
Password: abcd@1234
Sign up successful. Please login!
```

In the case of a successful sign up, this username and password are storied in the userdata.txt file. Below is the screenshot of how the text file looks after signing up. You can see the latest user sign up on the top as demonstrated above.



```
Main.java userdata.txt x
1 sumitx28,e0a8aa81eb1762d529783cf587f6f422
2 sumit,202cb962ac59075b964b07152d234b70
3 Sumit,ab56b4d92b40713acc5af89985d4b786
4
```



### 3. Input 1 for Login – Case of successful login (Verified Captcha and correct credentials)

In case of a valid credential and captcha, the program converts the entered password to a hash using MD5 and verifies if it is the same as stored in the userdata.txt file.

```
Run Main x
Hello User. This is your LIGHT WEIGHT DBMS
-----
Please Login or Sign up
-----
Enter 1 to Login and 2 to Sign up
1
Please enter your credentials
=====
Username: sumitx28
Password: abcd@1234
CAPTCHA: 49114
Enter the OTP:
49114
Welcome sumitx28
Enter a SQL query (or 'exit' to quit):
```

### 4. Input 1 for Login – Case of Invalid Credentials

As observed below, as they entered password is not the same as stored in our persistent storage, the user is now allowed to login and thrown an incorrect credential error.

```
Run Main x
/Users/Sumit/Library/Java/JavaVirtualMachines/openjdk-21/Contents/
Hello User. This is your LIGHT WEIGHT DBMS
-----
Please Login or Sign up
-----
Enter 1 to Login and 2 to Sign up
1
Please enter your credentials
=====
Username: sumitx28
Password: 18264
CAPTCHA: 21363
Enter the OTP:
21363
Incorrect Credentials. Please try again!
```

### 5. Input 1 for Login – Case of Invalid Captcha

As seen below, when the captcha is invalid, the user is not logged in and prompted to login again.

```
Run Main x
/Users/Sumit/Library/Java/JavaVirtualMachines/openjdk-21/Contents/
Hello User. This is your LIGHT WEIGHT DBMS
-----
Please Login or Sign up
-----
Enter 1 to Login and 2 to Sign up
1
Please enter your credentials
=====
Username: sumitx28
Password: abcd@1234
CAPTCHA: 92171
Enter the OTP:
96552
INVALID CAPTCHA! TRY LOGGING IN AGAIN!!
```

## Task B: Query Processor

### 1. CREATE TABLE QUERY

QUERY: *CREATE TABLE Users (id INT PRIMARY KEY, username VARCHAR(255) NOT NULL, email VARCHAR(255) UNIQUE, created\_at TIMESTAMP);*

Result: Table created successfully.

```
Run Main x
Please enter your credentials
=====
Username: sumitx28
Password: abcd
CAPTCHA: 83238
Enter the OTP:
83238
Welcome sumitx28
Enter a SQL query (or 'exit' to quit): CREATE TABLE Users (id INT PRIMARY KEY, username VARCHAR(255) NOT NULL, email VARCHAR(255) UNIQUE, created_at TIMESTAMP);
Query Result: Table created.
Enter a SQL query (or 'exit' to quit):
```

Table txt File After Creation of Table

```
© Main.java tabledata.txt x
1 Table: Users
2 created_at=TIMESTAMP
3 id=INT
4 email=VARCHAR(255)
5 username=VARCHAR(255)
```

### CASE OF INVALID QUERY

```
Enter a SQL query (or 'exit' to quit): CREAT TABLE User;
Query Result: Unsupported command.
Enter a SQL query (or 'exit' to quit): |
```

### 2. INSERT QUERY

QUERY: *INSERT INTO Users (id, username, email, created\_at) VALUES (1, 'user1', 'user1@example.com', '2023-10-24 12:00:00');*

Result: Data Inserted Successfully.

```
Enter a SQL query (or 'exit' to quit): INSERT INTO Users (id, username, email, created_at) VALUES (1, 'user1', 'user1@example.com', '2023-10-24 12:00:00');
Query Result: Data inserted.
Enter a SQL query (or 'exit' to quit):
```

Txt File for Stored Data, In Case of multiple inserts, data will be stored as comma separated values.

```
1 Table: Users
2 created_at='2023-10-24 12:00:00'
3 id=1
4 email='user1@example.com'
5 username='user1'
```

### 3. SELECT QUERY

QUERY: *SELECT \* FROM Users;*

Result: All data from users table is fetched in the following format.

```
Run Main x
Enter a SQL query (or 'exit' to quit): SELECT * FROM Users
Query Result: SELECT Result for Users:
+-----+-----+
| Key      | Value                                |
+-----+-----+
| created_at | '2023-10-24 12:00:00' |
+-----+-----+
| email      | 'user1@example.com' |
+-----+-----+
| id         | 1 |
+-----+-----+
| username   | 'user1' |
+-----+-----+
Enter a SQL query (or 'exit' to quit): |
```

### 4. UPDATE QUERY

QUERY: *UPDATE Users SET username = sumit WHERE id = 1;*

Result: Successfully updates the table field matching the condition.

```
Enter a SQL query (or 'exit' to quit): UPDATE Users SET username = sumit WHERE id = 1
Query Result: Table Updated.
Enter a SQL query (or 'exit' to quit):
```

Txt file after updating.

```
Main.java QueryMethods.java QueryProcessor.java tabledata.txt x
1 Table: Users
2 created_at='2023-10-24 12:00:00'
3 id=1
4 email='user1@example.com'
5 username='sumit'
```

### 5. DELETE QUERY

QUERY: *DELETE Users;*

Result: All data along with Users table is deleted successfully

```
Enter a SQL query (or 'exit' to quit): DELETE Users
Query Result: Table deleted.
Enter a SQL query (or 'exit' to quit):
```

Txt File is empty/void of Users table and data after deletion.

## Task C: Single Transaction Management

This can be achieved through maintaining an array list that will contain all the queries while they execute and having an intermediate temporary transaction txt file created at the backend.

This txt file will store all the transaction related updates and will only transfer the updates to persistent storage when a commit command is found.

In case of rollback, the changes won't be saved to the persistent storage and be deleted from this temporary txt file.

Once the whole transaction is complete, the initial array list must be emptied to maintain consistency.

## References:

- [1] D. Damoah, J. B. Hayfron-Acquah, S. Sebastian, E. Ansong, B. Agyemang and R. Villafane, "*Transaction recovery in federated distributed database systems*", Available at: <https://ieeexplore-ieee-org.ezproxy.library.dal.ca/document/7068178> (Accessed: 26 October 2023).
- [2] S. Conrad, "*Active Integrity Maintenance in Federated Database Systems*", Available at: [https://www.researchgate.net/figure/Architecture-of-federated-database-systems\\_fig1\\_2420718](https://www.researchgate.net/figure/Architecture-of-federated-database-systems_fig1_2420718) (Accessed: 26 October 2023).
- [3] "*Distributed DBMS - Database Environments*", Available at: [https://www.tutorialspoint.com/distributed\\_dbms/distributed\\_dbms\\_database\\_environments.htm](https://www.tutorialspoint.com/distributed_dbms/distributed_dbms_database_environments.htm) (Accessed: 26 October 2023).
- [4] L. H. Haas, "*DB2 architecture for database federation*", Available at: [https://www.researchgate.net/figure/DB2-architecture-for-database-federation\\_fig1\\_220354241](https://www.researchgate.net/figure/DB2-architecture-for-database-federation_fig1_220354241) (Accessed: 26 October 2023).
- [5] T. Zhao, "*Research on the Network Management System of Students Art Works based on DB2 Database*", Available at: <https://ieeexplore-ieee-org.ezproxy.library.dal.ca/document/9898868> (Accessed: 26 October 2023).
- [6] P. Lee, "*Software Design Principles Every Programmer Should Know*", Available at: <https://medium.com/@peterlee2068/software-design-principles-every-programmer-should-know> (Accessed: 27 October 2023).