CSCI 5408

DATA MANAGEMENT AND WAREHOUSING

ASSIGNMENT - 1

Banner ID: B00955686

GitLab Assignment Link:
https://git.cs.dal.ca/vvinod/b00955686_vishaka_vinod_a1

csci5408_f23_b00955686_vishaka_vinod

Table of Contents

Problem Statement 1:

Perform a systematic literature review and document your findings with critical analysis.

Literature Review:

A federated database system is a system that permits users to access and modify data from across multiple databases and it is managed by a federated server. According to the IBM documentation, this database contains catalog entries that act as a unique key to identify different database sources. The federated server examines the federated database system catalog and the data source and decides on the best way to process SQL statements [2]. This paper focuses on the improvement of transaction recovery in a federated database system by enhancing its architecture to sync global and local database partitions once the user commits. Issues with concurrency control leads to failure prone database and recovery tends to become difficult. The federated database systems are usually heterogenous, which is a distributed database management system (DBMS) that uses different hardware and software for each database which needs to be integrated together. Recovery becomes especially tricky in such systems as the different tech stacks need to be taken into considerations [1].

The authors explore the different types and characteristics of distributed databases such as Synchronous and Asynchronous databases. Synchronous databases provide access to users within the network and processes transactions in real time. However, this can cause latency issues and synchronous processing can be slow. Asynchronous databases provide access to the replicated data that is stored locally. This ensures that the response times are much faster. However, the implementation can become complex. The federated database systems can support either of these types based on how the system is configured. This also increases the complexity of transaction recovery as the structure and syntax of data models, queries, and constraints would be different and needs to be considered [1].

The main and most important goal of a successful transaction is to maintain ACID properties. This stands for Atomicity, Consistency, Isolation, and Durability [3]. This means whether the transaction gets completed or aborted, it needs to ensure that the data in the database is consistent and takes an all or nothing approach. The authors state that "Transactions are, therefore, indivisible and recoverable unit of execution at the atomic level usually and most times isolated, consistent and durable." Another important functionality of a distributed database is serializability. This implies that regardless of the order in which multiple transactions occur, the result should consistently mirror what it would be if the transactions were executed one after the other in sequence. This can be done with concurrency control [1].

The authors then dive into the different types of recoveries such as write-ahead logging and checkpoints. A transaction may fail for a variety of reasons, such as node failures, divide by zero, value dependency failures, deadlocks, system, or server connectivity failures. The DBMS should be designed to recover from such failures. Only the completed transactions should commit, or if necessary, be completely rolled back, or rolled back to the specified savepoints [4].

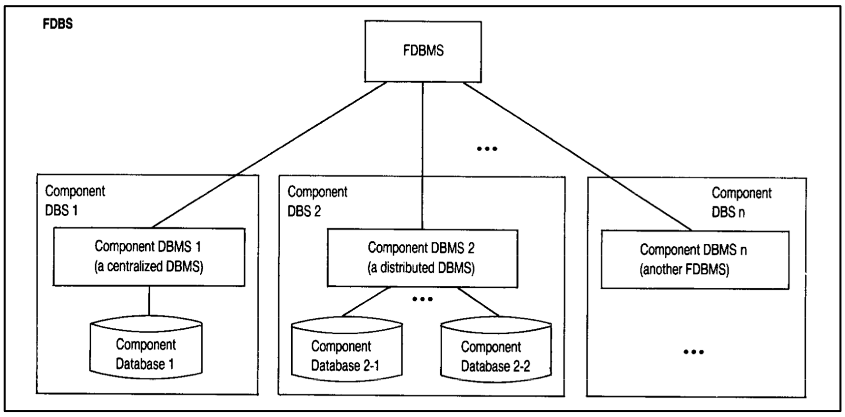


Figure 1: A Federated DBMS and its Components [5]

Every implementation of concurrency control for a federated database management system has its limitations. This paper focuses on addressing them by developing algorithms for the federated DBMS architecture. It starts with a description of a Local Transaction Manager (LTA) which is responsible for managing sub-transactions with the help of a concurrency control agent and a 2-phase commit protocol agent. A Global Transaction Manager (GTM) is used to manage transactions between multiple data sources. It is the primary point of transaction recovery in a federated DBMS. The author's goal is to adapt and improve upon the already existing GTM algorithm and Hwang algorithm to better the sync controller. The sync controller is the interface between the global and local database. Any changes made and committed needs to be consistent across all the databases in the system [1].

The authors implemented their algorithm using Java to make use of its large library collection that help to implement nodes and transactions. It was then tested on a banking transaction system. The implemented algorithm works in the following flow:

1. Once transaction starts, 2 sub-transactions are started by the root node. One for credit and one for debit transaction operation.
2. The 2-phase commit (2PC) protocol is used to maintain serializability between both the sub-transactions.
3. The sync coordinator uses the transaction information and checks to see if the transactions are ready for the commit.
4. Once checks are done, the sync coordinator commits the transactions across the database system.

This improved algorithm can be used to improve the safety and security of transactions on sensitive data while maintaining database integrity. This algorithm can be further researched to improve the current functionality of banking systems and can also be seamlessly integrated onto other disciplines such as cryptocurrency, insurance companies, and medical industry [1].

This research paper provides a foundational analysis of the sync coordinator, paving the way for potential future research. However, rather than limiting the scope to a two-sub-transaction banking application, a more robust approach could have been taken by adopting a larger testing environment.

Similar Research Analysis

In this paper, the authors focus on improving the transaction recovery of a federated database management system using a multilevel transaction management approach. The key challenge that is being addressed is the management of transactions between the local and global databases. The data is being modified across multiple local databases and syncing those changes across the global database becomes quite complex and more prone to transaction failures and data inconsistency between databases. The paper presents an approach for managing federated transactions that is different from the traditional read/write model. It is based on utilizing the semantics of high-level operations and translating them into open sub-transactions. This ensures that only the global database can access the local databases. These limitations aim to ensure autonomy and security. The authors state that in a banking or airline system, the organization does not permit the customers to have access to their databases to maintain data and information integrity and security. With this perspective, they proposed an employ a requestor-server architecture using the multilevel nested transaction model and a synchronization coordinator to enhance concurrent executions. The paper discusses the importance of maintaining consistency across multiple databases, especially for critical applications with sensitive data. The authors proposed a method of work planning based on a type of transaction known as "open nested transactions." This helps to resolve task conflicts and allows multiple tasks to take place at the same time. This can speed up the transaction recovery process [6].

Databases are crucial for any application or organization. And as the world advances and everyone is going remote, it is crucial for the distributed databases to function as perfectly as possible. The research paper presents a thoughtful and relevant approach to transaction management in federated database systems. The authors aim to resolve a genuine and practical concern regarding transactions in a federated DBMS. The focus on transaction management in failure-prone environments is highly relevant in real-world scenarios. The paper explains in detail the problem statement, objective, goals, and proposed solution. The authors focus on maintaining security which is a huge practical concern that needs to always be addressed. While the proposed design is theoretically strong, it may be difficult to execute it in a real-world environment. The author's approach is shown to increase concurrency and enhance the execution autonomy of local databases. It also provides examples of environments where the proposed approach is relevant, like banking and airplane environments, where high levels of autonomy and performance are crucial. However, the proposed system is limited to such scenarios where the local database needs to remain private. The study makes no mention of potential implementation challenges or issues. It is important in any research to identify the shortcoming of the proposed solution as well as its future scope. The abstract does not mention any experimental results or case studies. The authors mention that scope of this paper to achieve the long-term goals of combining traditional approaches to FDBMS transaction management while aiming to maintain the transaction ACID properties [6].

Problem Statement 2:

Prototype of a light-weight DBMS using Java programming language (no 3rd party libraries allowed).

Design Principles Used

The foundational design principles are the SOLID design principles. It helps with code maintainability, readability, and reusability. The SOLID principles are:

1. Single-Responsibility Principle (SRP):
   This states that "a class should have one and only one reason to change, meaning that a class should have only one job [7]."

2. Open-Closed Principle (OCP):
   This states that "objects or entities should be open for extension but closed for modification. [7]".

3. Liskov Substitution Principle (LSP):
   This states that "every subclass or derived class should be substitutable for their base or parent class [7]."

4. Interface Segregation Principle (ISP):
   This states that "a client should never be forced to implement an interface that it doesn't use, or clients shouldn't be forced to depend on methods they do not use [7]."

5. Dependency Inversion Principle (DIP):
   This states that "entities must depend on abstractions, not on concretions. It states that the high-level module must not depend on the low-level module, but they should depend on abstractions [7]."

In this program, I have used the single responsibility principle in All my classes as they are function specific. For example, the Create class is only responsible for table creation, the Update class is only responsible for table data modifications and so on. I have also used the open-closed principle by ensuring that only the entities are accessed only through their getters and setters so that they are open for extension but cannot be modified. Finally, I used the interface segregation concept to ensure that nothing is dependent on something it does not require. The PerformTransactions class, for example, is only dependant on insert, remove, and update and nothing else.

Test Cases 1: Create

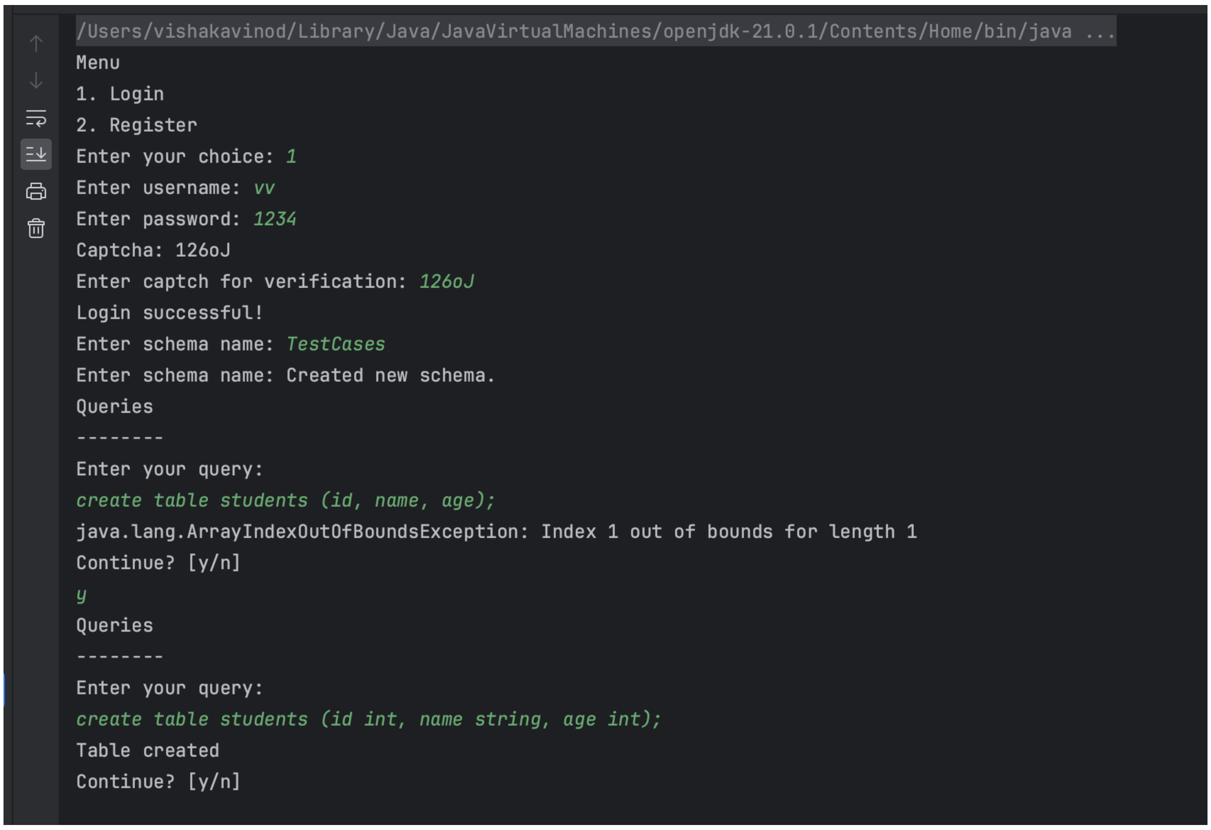
Consider a schema has a table called “students” which contains student id (id), student name (name), and their age (age). The primary key is “id”. If the user wants to create the table but enters the wrong format, then an ArrayIndexOutOfBoundsException is thrown.

Expected format:

    create table students (id int, name string, age int);

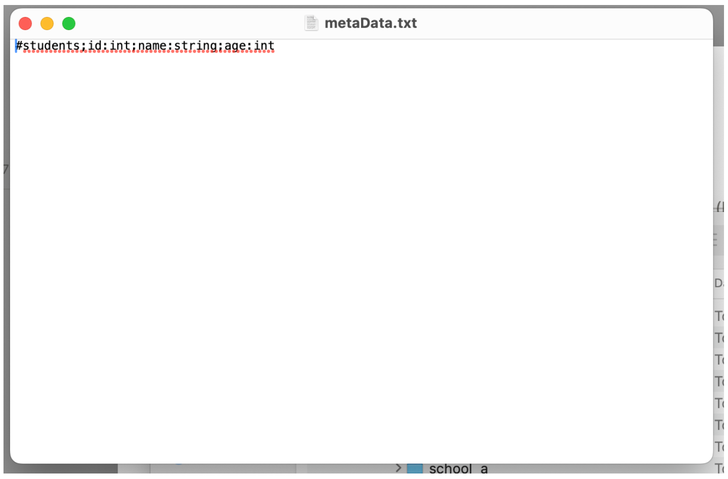
Wrong format:

    create table students (id, name, age);



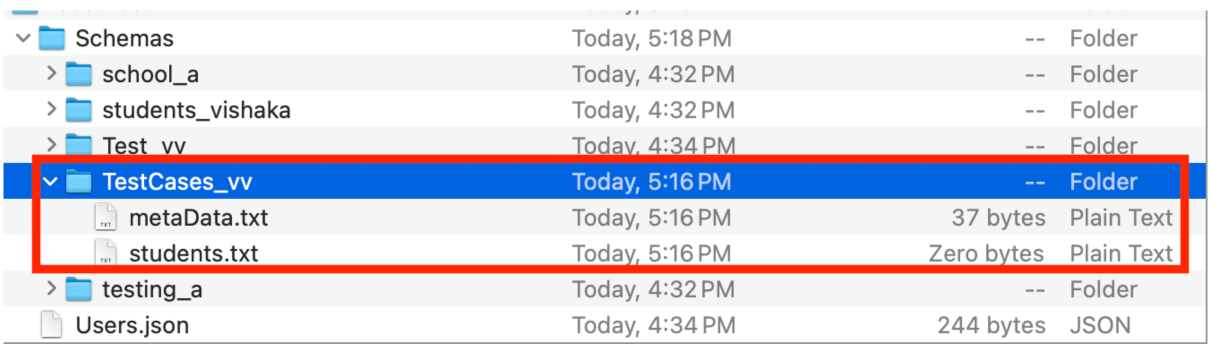
```
/Users/vishakavinod/Library/Java/JavaVirtualMachines/openjdk-21.0.1/Contents/Home/bin/java ...
Menu
1. Login
2. Register
Enter your choice: 1
Enter username: vv
Enter password: 1234
Captcha: 126oJ
Enter captch for verification: 126oJ
Login successful!
Enter schema name: TestCases
Enter schema name: Created new schema.
Queries
--------
Enter your query:
create table students (id, name, age);
java.lang.ArrayIndexOutOfBoundsException: Index 1 out of bounds for length 1
Continue? [y/n]
y
Queries
--------
Enter your query:
create table students (id int, name string, age int);
Table created
Continue? [y/n]
```

Figure 2: When user inputs wrong CREATE statement it throws ArrayIndexOutOfBoundsException exception.



```
#students;id:int;name:string;age:int
```

Figure 3: The metaData.txt stores the table description



Schemas	Today, 5:18 PM	--	Folder
school_a	Today, 4:32 PM	--	Folder
students_vishaka	Today, 4:32 PM	--	Folder
Test_vv	Today, 4:34 PM	--	Folder
TestCases_vv	Today, 5:16 PM	--	Folder
metaData.txt	Today, 5:16 PM	37 bytes	Plain Text
students.txt	Today, 5:16 PM	Zero bytes	Plain Text
testing_a	Today, 4:32 PM	--	Folder
Users.json	Today, 4:34 PM	244 bytes	JSON

Figure 4: The students.txt table file is created under the Schema directory

Test Case 2: Insert

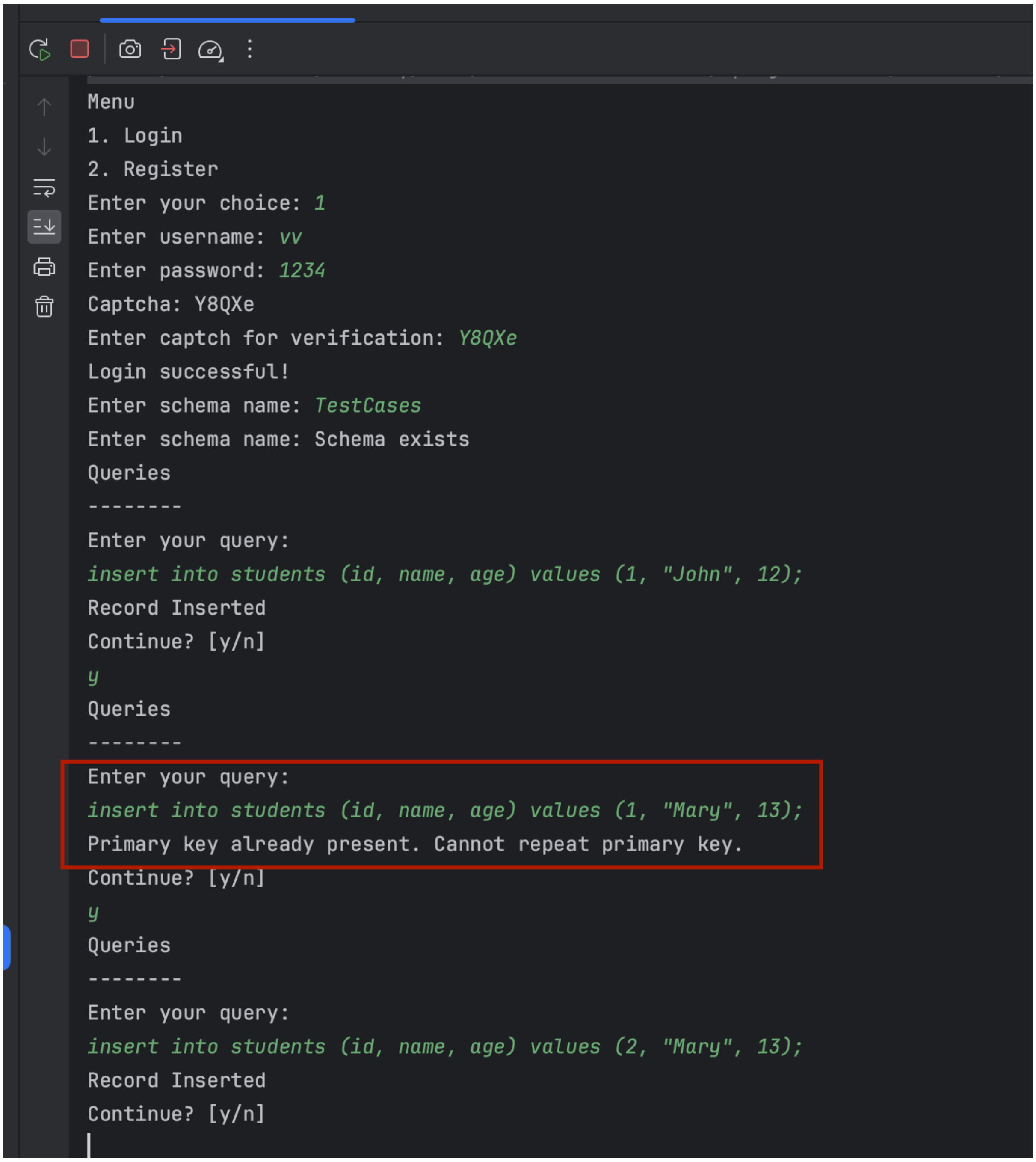
Let us insert into the table 2 records. If the primary key for both the records are the same the system will throw an error.

Expected format:

> insert into students (id, name, age) values (1, “John”, 12);
> insert into students (id, name, age) values (2, “Mary”, 13);

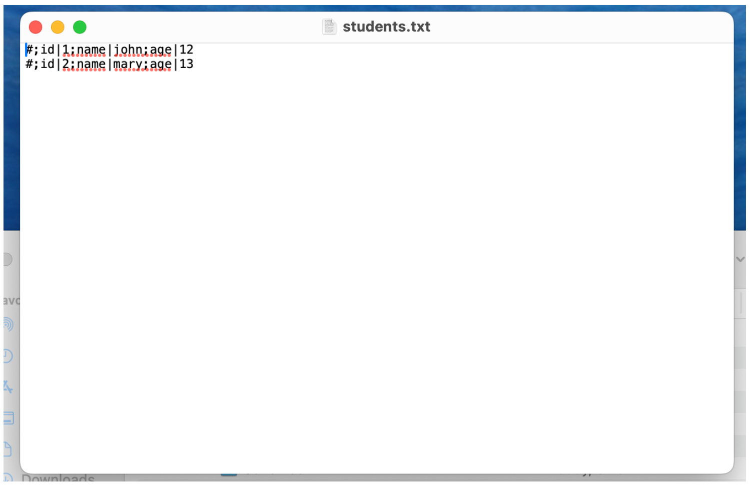
Wrong format:

> insert into students (id, name, age) values (1, “John”, 12);
> insert into students (id, name, age) values (1, “Mary”, 13);



```
Menu
1. Login
2. Register
Enter your choice: 1
Enter username: vv
Enter password: 1234
Captcha: Y8QXe
Enter captch for verification: Y8QXe
Login successful!
Enter schema name: TestCases
Enter schema name: Schema exists
Queries
--------
Enter your query:
insert into students (id, name, age) values (1, "John", 12);
Record Inserted
Continue? [y/n]
y
Queries
--------
Enter your query:
insert into students (id, name, age) values (1, "Mary", 13);
Primary key already present. Cannot repeat primary key.
Continue? [y/n]
y
Queries
--------
Enter your query:
insert into students (id, name, age) values (2, "Mary", 13);
Record Inserted
Continue? [y/n]
```

Figure 5: The insert statement with same primary key failed



```
#;id|1;name|john;age|12
#;id|2;name|mary;age|13
```

Figure 6: The students.txt file is populated with data

Test Case 3: Update

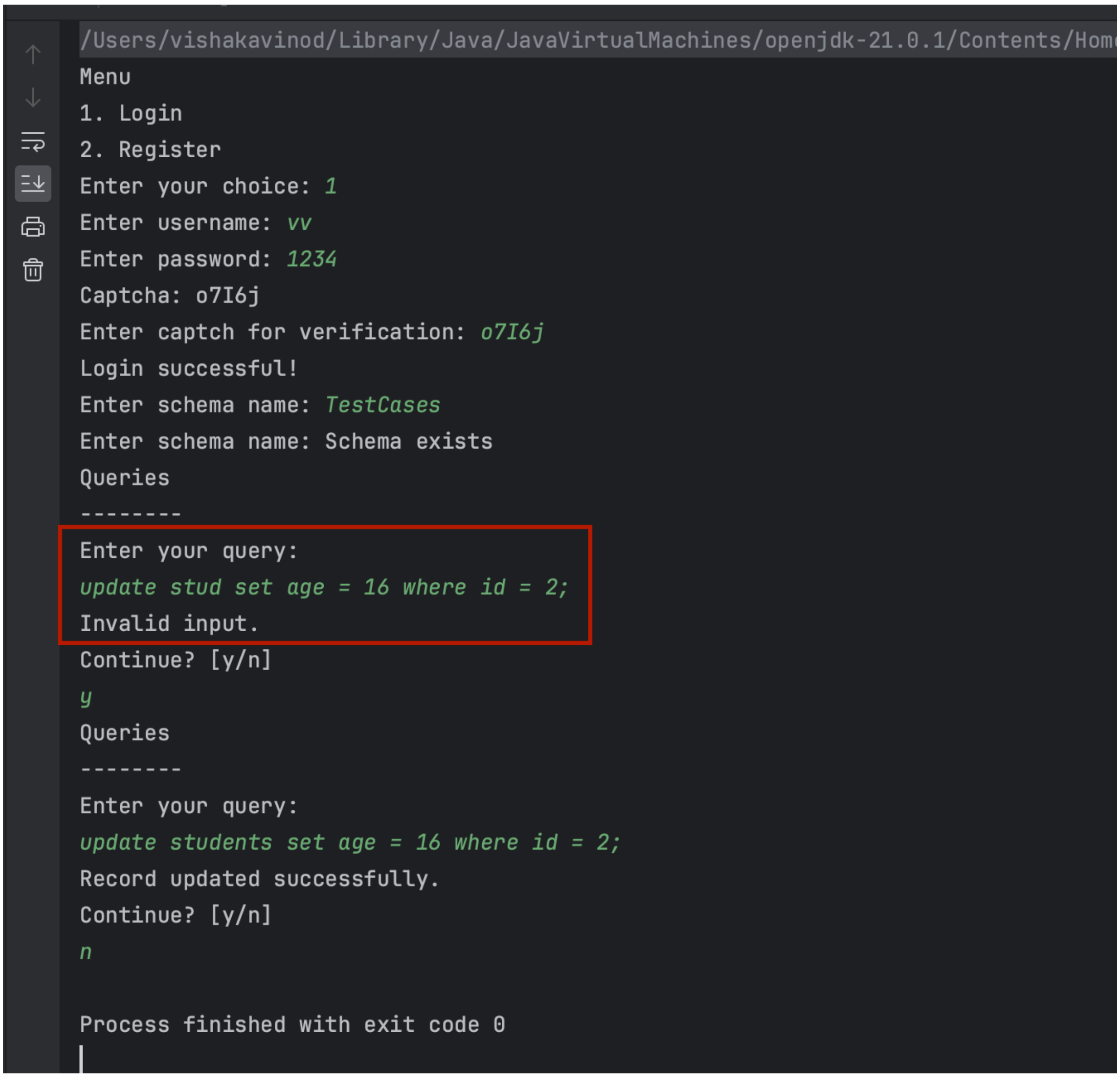
If we want to update a student’s age, but we enter the wrong table name. The system throws an error stating that the table does not exist.

Expected format:

update students set age = 16 where id = 2;

Wrong format:

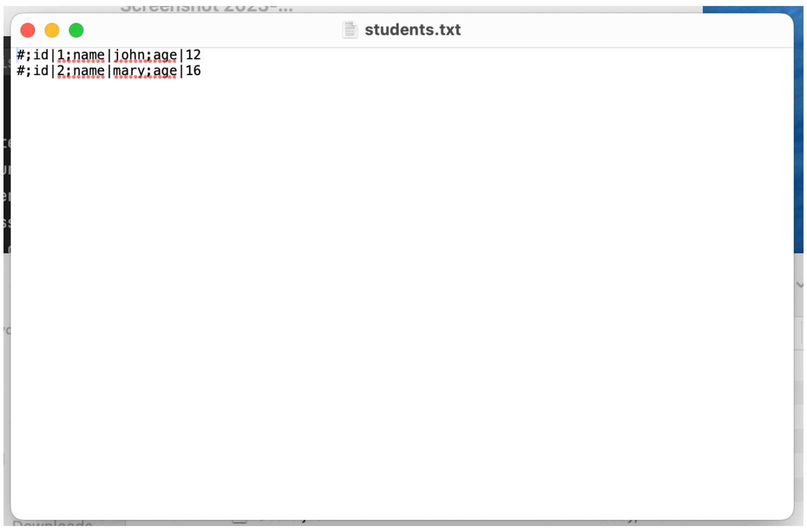
update stud set age = 16 where id = 2;



```
/Users/vishakavinod/Library/Java/JavaVirtualMachines/openjdk-21.0.1/Contents/Hom
Menu
1. Login
2. Register
Enter your choice: 1
Enter username: vv
Enter password: 1234
Captcha: o7I6j
Enter captch for verification: o7I6j
Login successful!
Enter schema name: TestCases
Enter schema name: Schema exists
Queries
--------
Enter your query:
update stud set age = 16 where id = 2;
Invalid input.
Continue? [y/n]
y
Queries
--------
Enter your query:
update students set age = 16 where id = 2;
Record updated successfully.
Continue? [y/n]
n

Process finished with exit code 0
```

Figure 7: When wrong table is queries an error is thrown



```
#;id|1;name|john;age|12
#;id|2;name|mary;age|16
```

Figure 8: The students.txt file has been updated

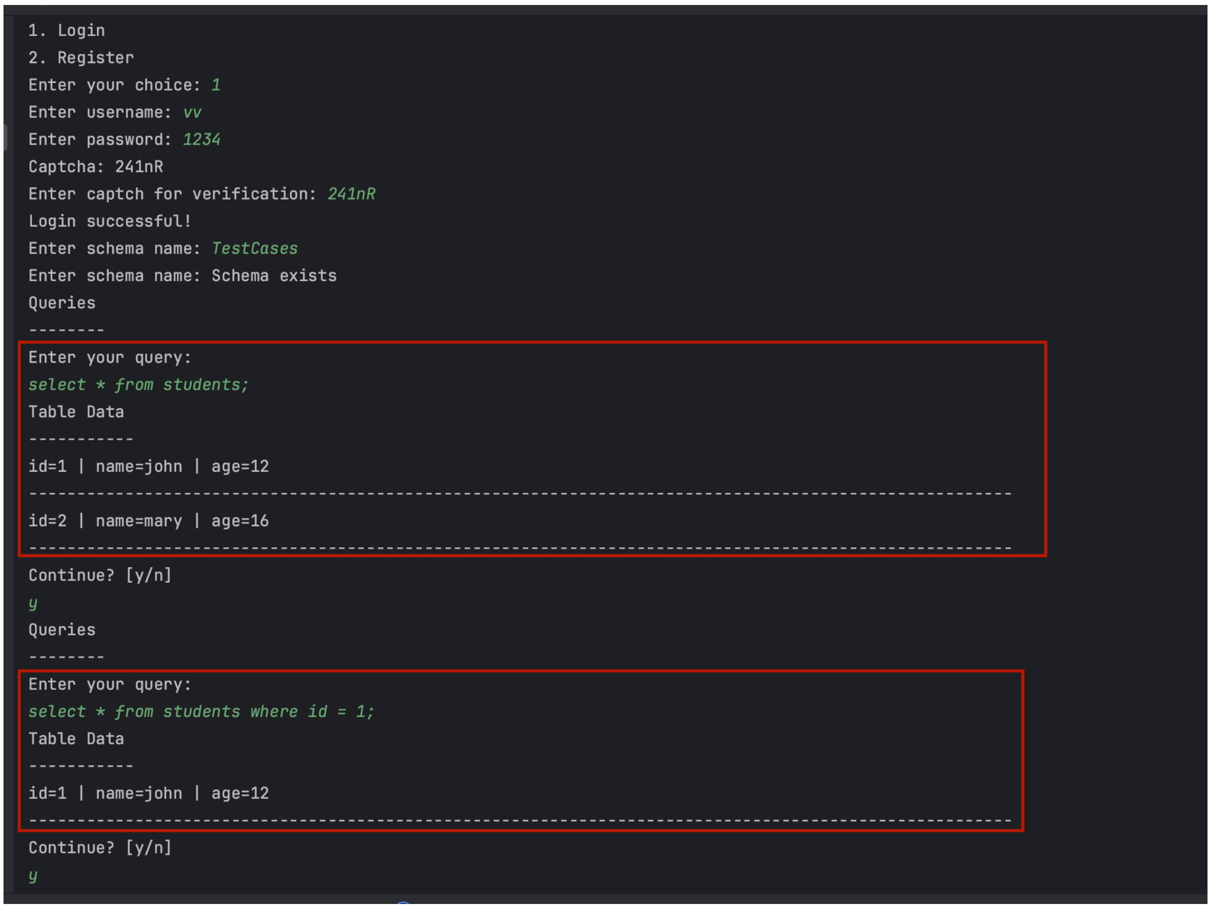
Test Case 4: Select

The select statement supports three types of formats:

1. select * from table_name;
2. select * from table_name where columName = columnValue;
3. select columnName1, columnName2, ... ,columnNameN from table_name;

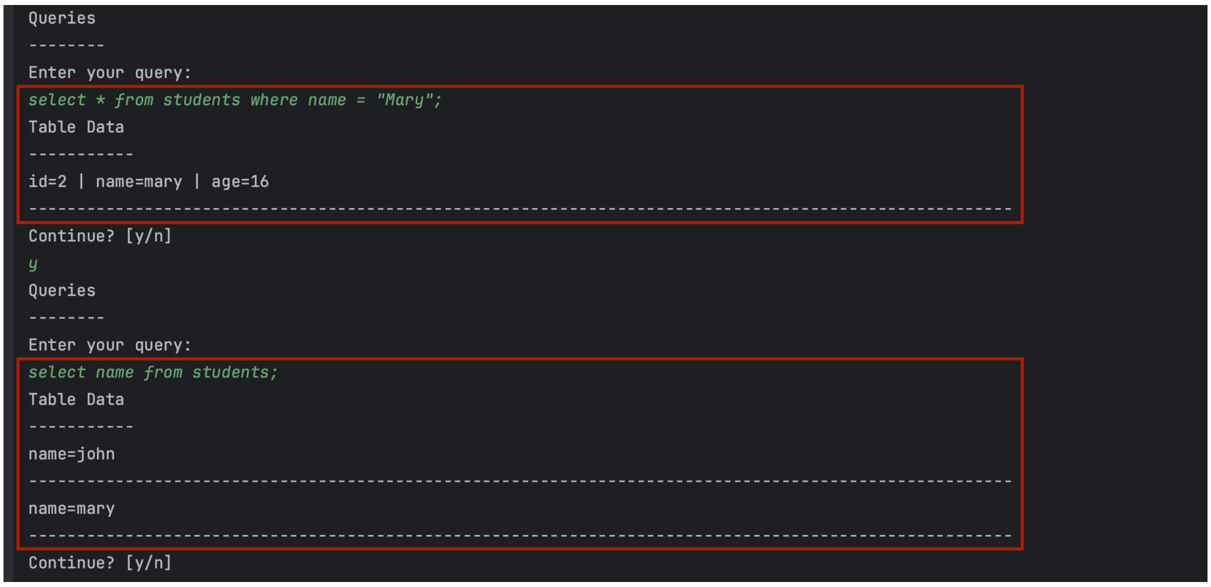
Sample input statement:

- select * from students;
- select * from students where id = 1;
- select * from students where first_name = "John";
- select first_name, last_name from students;



```
1. Login
2. Register
Enter your choice: 1
Enter username: vv
Enter password: 1234
Captcha: 241nR
Enter captch for verification: 241nR
Login successful!
Enter schema name: TestCases
Enter schema name: Schema exists
Queries
--------
Enter your query:
select * from students;
Table Data
-----------
id=1 | name=john | age=12
---------------------------------------------------------------------------------------------------------
id=2 | name=mary | age=16
---------------------------------------------------------------------------------------------------------
Continue? [y/n]
y
Queries
--------
Enter your query:
select * from students where id = 1;
Table Data
-----------
id=1 | name=john | age=12
---------------------------------------------------------------------------------------------------------
Continue? [y/n]
y
```

Figure 9: Output for various SELECT statement's part 1



```
Queries
--------
Enter your query:
select * from students where name = "Mary";
Table Data
-----------
id=2 | name=mary | age=16
---------------------------------------------------------------------------------------------------------
Continue? [y/n]
y
Queries
--------
Enter your query:
select name from students;
Table Data
-----------
name=john
---------------------------------------------------------------------------------------------------------
name=mary
---------------------------------------------------------------------------------------------------------
Continue? [y/n]
```

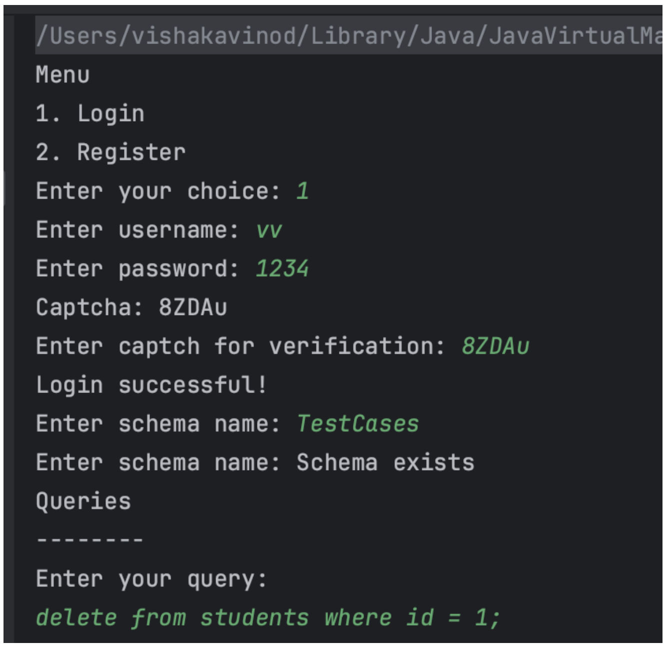
Figure 10: Output for various SELECT statement's part 2

Test Case 5: Delete

The delete functionality is quite simple. It removes all the rows that satisfy the delete condition.

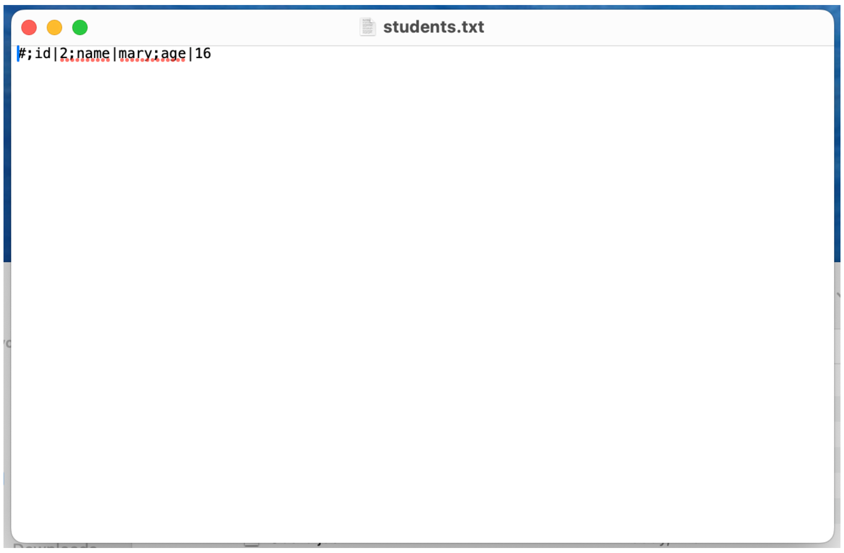
Sample input statement:

delete from students where id = 1;



```
/Users/vishakavinod/Library/Java/JavaVirtualMa
Menu
1. Login
2. Register
Enter your choice: 1
Enter username: vv
Enter password: 1234
Captcha: 8ZDAu
Enter captch for verification: 8ZDAu
Login successful!
Enter schema name: TestCases
Enter schema name: Schema exists
Queries
--------
Enter your query:
delete from students where id = 1;
```

Figure 11: Deleting a record



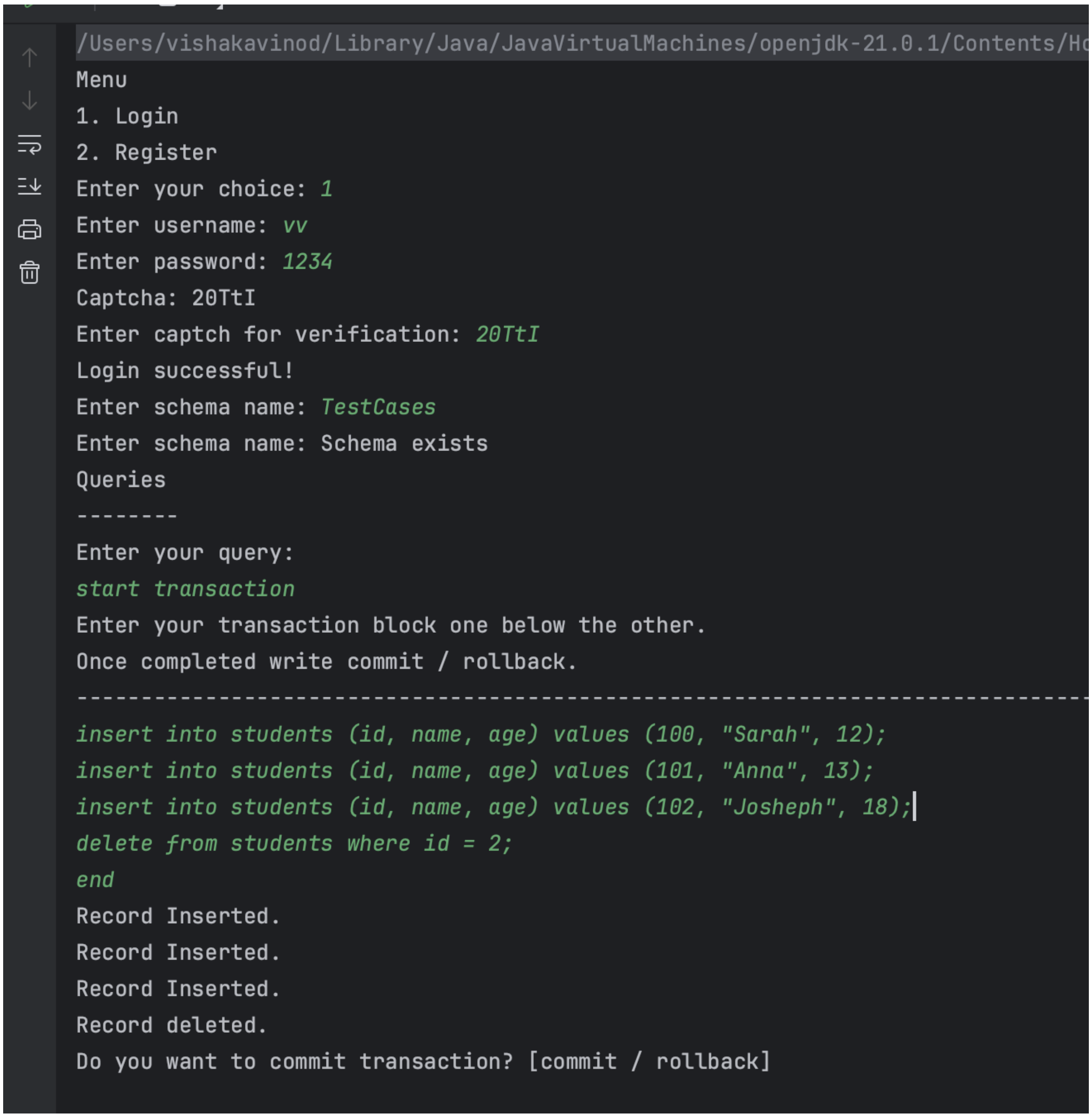
```
#;id|2;name|mary;age|16
```

Figure 12: The students.txt has been updated after the DELETE

Test Case 6: Transactions

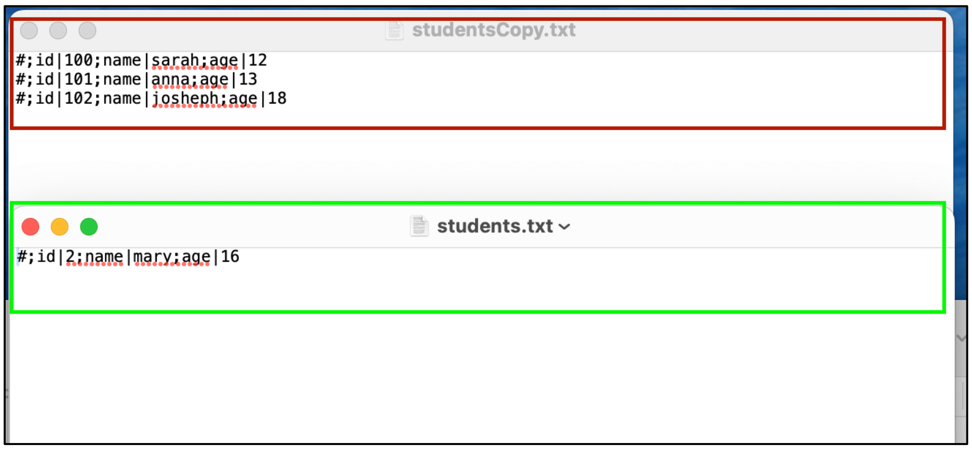
This test case is to demonstrate the transactions functionality. When the user inputs "START TRANSACTION", the system allows the user to input multiple queries until they type "end".

The transactions then start executing one by one and getting saved to the buffer. Once the transactions are completed, the user has the option to commit or rollback the changes.



```
/Users/vishakavinod/Library/Java/JavaVirtualMachines/openjdk-21.0.1/Contents/Ho
Menu
1. Login
2. Register
Enter your choice: 1
Enter username: vv
Enter password: 1234
Captcha: 20TtI
Enter captch for verification: 20TtI
Login successful!
Enter schema name: TestCases
Enter schema name: Schema exists
Queries
--------
Enter your query:
start transaction
Enter your transaction block one below the other.
Once completed write commit / rollback.
------------------------------------------------------------------------------
insert into students (id, name, age) values (100, "Sarah", 12);
insert into students (id, name, age) values (101, "Anna", 13);
insert into students (id, name, age) values (102, "Josheph", 18);
delete from students where id = 2;
end
Record Inserted.
Record Inserted.
Record Inserted.
Record deleted.
Do you want to commit transaction? [commit / rollback]
```

Figure 13: Performing transactions

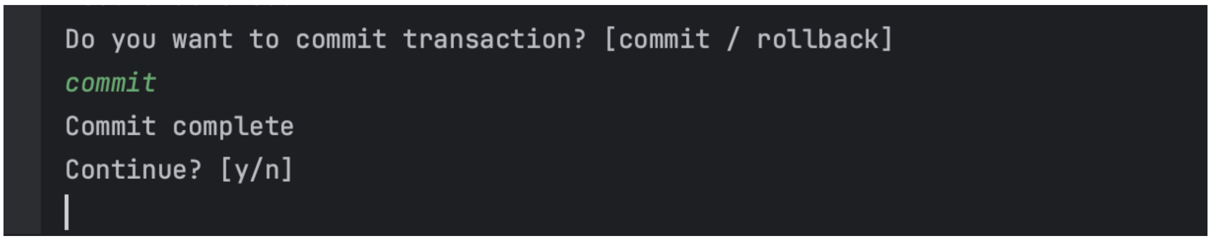


```
#;id|100;name|sarah;age|12
#;id|101;name|anna;age|13
#;id|102;name|josheph;age|18
```

```
#;id|2;name|mary;age|16
```

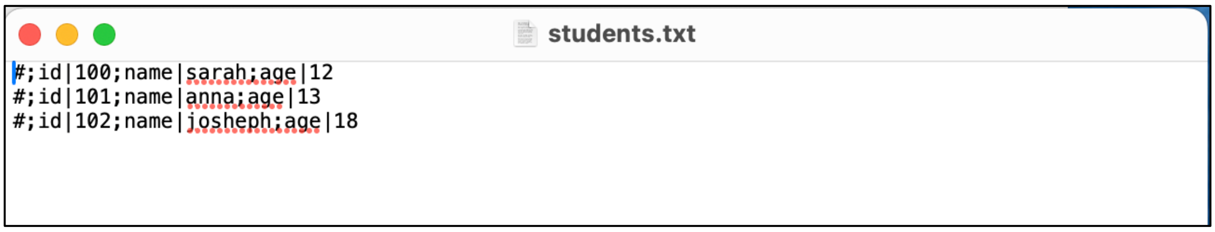
Figure 14: Data not inserted in actual file but is stored in buffer file before commit.

The red highlight is the buffer file “studentsCopy.txt” and the green highlight is the original table file “students.txt”.



```
Do you want to commit transaction? [commit / rollback]
commit
Commit complete
Continue? [y/n]
```

Figure 15: Transaction has been committed.



```
#;id|100;name|sarah;age|12
#;id|101;name|anna;age|13
#;id|102;name|josheph;age|18
```

Figure 16: Transaction commit has updated the students.txt to reflect the changes

References:

[1] D. Damoah, J. B. Hayfron-Acquah, S. Sebastian, E. Ansong, B. Agyemang and R. Villafane, “Transaction recovery in federated distributed database systems,” *Proceedings of IEEE International Conference on Computer Communication and Systems ICCCS14*, Chennai, India, 2014, pp. 116-123, doi: 10.1109/ICCCS.2014.7068178 [Online]. Availble: https ://ieeexplore.ieee.org/document/7068178. [Accessed: October 20, 2023].

[2] “IBM documentation,” *Ibm.com*, January 20, 2023. [Online]. Available: https://www.ibm.com/docs/en/db2/11.1?topic=systems-federated-database. [Accessed: October 20, 2023].

[3] “IBM documentation,” *Ibm.com*, January 10, 2023. [Online]. Available: https://www.ibm.com/docs/en/cics-ts/5.4?topic=processing-acid-properties-transactions. [Accessed: October 20, 2023].

[4] K. Barker and M. T. Ozsu, "Reliable transaction execution in multidatabase systems," *[1991] Proceedings. First International Workshop on Interoperability in Multidatabase Systems*, Kyoto, Japan, 1991, pp. 344-347, doi: 10.1109/IMS.1991.153733 [Online]. Available: https://ieeexplore.ieee.org/document/153733. [Accessed: October 20, 2023].

[5] A. P. Sheth and J. A. Larson, “Federated database systems for managing distributed, heterogeneous, and autonomous databases,” *ACM Comput. Surv*., vol. 22, no. 3, pp. 183–236, 1990 [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/96602.96604. [Accessed: October 21, 2023]

[6] A. Deacon, H. . -J. Schek and G. Weikum, "Semantics-based multilevel transaction management in federated systems," Proceedings of 1994 IEEE 10th International Conference on Data Engineering, Houston, TX, USA, 1994, pp. 452-461, doi: 10.1109/ICDE.1994.283066 [Online]. Available: https://ieeexplore.ieee.org/abstract/document/283066. [Accessed: October 22, 2023]

[7] S. Oloruntoba, “SOLID: The first 5 principles of object oriented design,” *Digitalocean.com*, September 21, 2020. [Online]. Available: https://www.digitalocean.com/community/conceptual-articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design. [Accessed: October 22, 2023].

[8] “How to write doc comments for the javadoc tool,” *Oracle.com*. [Online]. Available: https://www.oracle.com/ca-en/technical-resources/articles/java/javadoc-tool.html. [Accessed: October 26, 2023].

[9] TechMeDevoted, “Captcha Component: How to create and validate captcha using Java,” *YouTube* [Online]. Available: https://www.youtube.com/watch?v=v1edYp2zPGo. [Accessed: October 26, 2023].

[10] “JsonParser (java(TM) EE 7 specification APIs),” *Oracle.com*, June 01, 2015. [Online]. Available: https://docs.oracle.com/javaee/7/api/javax/json/stream/JsonParser.html. [Accessed: October 26, 2023].

[11] “MessageDigest (java platform SE 8),” *Oracle.com,* October 04, 2023. [Online]. Available: https://docs.oracle.com/javase/8/docs/api/java/security/MessageDigest.html. [Accessed: October 26, 2023].

[12] Baeldung, “Guide to BufferedReader,” *Baeldung.com*. [Online]. Available: https://www.baeldung.com/java-buffered-reader#:~:text=BufferedReader%20is%20a%20class%20which,efficient%20reading%20of%20text%20data. [Accessed: October 26, 2023].

[13] Andrew, “File mkdirs() method in Java with examples,” *GeeksforGeeks*, January 28, 2019. [Online]. Available: https://www.geeksforgeeks.org/file-mkdirs-method-in-java-with-examples/. [Accessed: October 27, 2023].

[14] “Path operations,” *Oracle.com*. [Online]. Available: https://docs.oracle.com/javase/tutorial/essential/io/pathOps.html. [Accessed: October 27, 2023].