

# CSCI 5408 DATA MANAGEMENT AND WAREHOUSING

Group:

DBMS\_Builder- 6

Sprint 2 Report

Group Members:

Name	Enrollment Number
Vedant Patel	B00984592
Jems Patel	B00984406
Rushil Borad	B00977837

Group Project Git Lab Link for code:

[https://git.cs.dal.ca/jems/csci\\_5408\\_s24\\_group06](https://git.cs.dal.ca/jems/csci_5408_s24_group06)

## Contents

Pseudocodes .....	3
Pseudocode for Export Structure and Value .....	3
Pseudocode for Transaction Management Implementation .....	5
TransactionManagerImpl Implementation: .....	5
Pseudocode for InsertIntoTable Class for Transaction .....	7
InsertIntoTable Class: .....	7
Pseudocode for UpdateTableClass for Transaction .....	9
UpdateTable Class.....	9
Pseudocode for SelectFromTableClass for Transaction .....	10
SelectFromTable Class .....	10
Pseudocode for DeleteFromTableClass for Transaction .....	11
DeleteFromTable Class .....	11
Pseudocode for EventLog Class .....	12
Pseudocode for GeneralLog Class .....	13
Pseudocode for QueryLog Class .....	13
Functional Testing:.....	14
Transaction processing (Module 3): .....	14
Log management (Module 4): .....	26
Export Structure and Value (Module 6): .....	28
Sprint 2 Meeting Logs: .....	32
Sprint 1 meeting Recordings:.....	33
References .....	34

# Pseudocodes

## Pseudocode for Export Structure and Value

### 1. Prompting User for Database Name:

- Initialise scanner for user input.
- Declare a variable for the database name.
- Repeat until a valid database name is entered:
- Prompt the user to enter the database name.
- Call the function to check if the database exists.
  - If it exists, print a message indicating the start of the export process.
  - Call the function to export the database.
  - Exit the loop.
  - If it does not exist, print an error message.

### 2. Checking Database Existence:

- Hash the entered password.
- Define the path where databases are stored.
- Create a path to the specific database using the provided database name.
- Verify if the path corresponds to a directory
  - Return true if it is a directory.
  - Return false if it is not a directory.

### 3. Exporting Database Data and Structure:

- Define the path where databases are stored.
- Define the path where the exported files will be saved.
- Create the path to the specific database using the provided database name.
- Define the output file path for the exported SQL file.
- Ensure the export directory exists.
- Initialize a list to store the SQL dump.
- Traverse all files in the database directory:
  - Return true if it is a directory.
  - For each regular file (table):
  - Extract the table name from the file name.
  - Call the function to export the structure of the table and add the result to the SQL dump.

- Call the function to export the data of the table and add the result to the SQL dump.
  - Call the function to save the SQL dump to the output file.
  - Print a success message.
4. **Extracting Table Structure:**
- Read all lines from the table file.
  - Initialize a StringBuilder for the SQL structure.
  - Add a SQL command to drop the table if it exists.
  - Extract the columns from the first line of the file.
  - Add a SQL command to create the table with the extracted columns:
    - Replace primary key indicators as needed.
  - Return the SQL structure as a string.
5. **Extracting Table Data:**
- Read all lines from the table file.
  - Initialize a StringBuilder for the SQL data.
  - Extract the columns from the first line of the file.
  - For each subsequent line (representing a row):
    - Split the line into values.
    - Add a SQL command to insert the values into the table.
    - Handle null values appropriately.
  - Return the SQL data as a string.
6. **Saving SQL Dump to the file:**
- Open a buffered writer for the specified file.
  - Write each line.

## Pseudocode for Transaction Management Implementation

### TransactionManagerImpl Implementation:

1. Declare a buffer to store data during a transaction.
  - `public static Map<String, ArrayList<ArrayList<String>>> buffer = new HashMap<>();`
2. Declare a flag to indicate if a transaction is active.
  - `public static boolean transactionActive = false;`
3. Implement `startTransaction()`:
  - Set `transactionActive` to true.
  - Print a message indicating the transaction has started.
  - Log the transaction event.
4. Implement `commitTransaction()`:
  - If `transactionActive` is true:
    - Iterate through the buffer.
    - For each table in the buffer:
      - For each row in the table:
        - Depending on the operation type (INSERT, UPDATE, DELETE):
          - Execute the respective operation.
    - Clear the buffer.
    - Set `transactionActive` to false.
    - Print a success message.
    - Log the transaction event.
  - If `transactionActive` is false:
    - Print an error message.
    - Log the transaction event.
5. Implement `rollbackTransaction()`:
  - If `transactionActive` is true:
    - Clear the buffer.

- Set transactionActive to false.
    - Print a success message.
    - Log the transaction event.
  - If transactionActive is false:
    - Print an error message.
    - Log the transaction event.
6. Implement isTransactionActive():
- Return the value of transactionActive.
7. Implement addQueryToTransaction(String tableName, String values):
- Get the buffer for the specified table.
  - Add the insert operation to the buffer.
  - Print a message indicating the operation has been added to the buffer.
  - Log the transaction event.
8. Implement addUpdateToTransaction(String tableName, String updateColumn, String updateValue, String conditionColumn, String conditionValue):
- Get the buffer for the specified table.
  - Add the update operation to the buffer.
  - Print a message indicating the operation has been added to the buffer.
  - Log the transaction event.
9. Implement addDeleteToTransaction(String tableName, String column, String value):
- Get the buffer for the specified table.
  - Add the delete operation to the buffer.
  - Print a message indicating the operation has been added to the buffer.
  - Log the transaction event.

## Pseudocode for InsertIntoTable Class for Transaction

### InsertIntoTable Class:

1. Declare a TransactionManager instance.
  - `private static final TransactionManager transactionManager = new TransactionManagerImpl();`
2. Implement insert (String query):
  - Check if a database is selected.
  - Match the query against the insert pattern.
  - If the query matches:
    - Extract the table name and values.
    - If a transaction is active:
      - Add the query to the transaction buffer.
      - Print a message indicating the operation has been buffered.
      - Log the database change.
    - If no transaction is active:
      - Execute the insert operation directly on the table.
  - If the query does not match:
    - Print an error message.
    - Log the database change.
3. Implement executeInsert(String tableName, String values):
  - Get the table file.
  - Read the table file.
  - Validate the primary key.
  - Format the values.
  - Write the formatted values to the table file.
  - Print a success message.
  - Log the database change.

4. Implement `validatePrimaryKey(File tableFile, String[] columnDefinitions, String[] valuesArray)`:

- Find the primary key index.
- Check if the primary key value is null.
- Read the table file.
- Check for duplicate primary key values.
- Return true if the primary key is valid.
- Return false if the primary key is invalid.



# Pseudocode for UpdateTableClass for Transaction

## UpdateTable Class

### 1. Class UpdateTable:

- transactionManager = new TransactionManagerImpl()

### 2. Method update(query):

- Check if a database is selected.
- Match the query against the update pattern.
- If the query matches:
  - Extract the table name, update column, update value, condition column, and condition value.
  - If a transaction is active:
    - Add the update operation to the transaction buffer.
    - Print "Update operation buffered."
    - Log the database change.
  - If no transaction is active:
    - Execute the update operation directly on the table.
- If the query does not match:
  - Print "Invalid UPDATE query format."
  - Log the database change.

### 3. Method executeUpdate(tableName, updateColumn, updateValue, conditionColumn, conditionValue):

- Get the table file.
- Read the table file.
- Validate the update and condition columns.
- Perform the update operation on the table file.
- If successful, print "Record(s) updated successfully."
- Log the database change.

## Pseudocode for SelectFromClass for Transaction

### SelectFromClass Class

#### 1. Class SelectFromClass:

- transactionManager = new TransactionManagerImpl()

#### 2. Method select(query):

- Check if a database is selected.
- Match the query against the select pattern.
- If the query matches:
  - Extract the columns part, table name, condition column, and condition value.
  - If a transaction is active:
    - Read buffered data from the transaction for the table.
    - Log transaction events.
  - Read and process data from the table file.
  - Print matching records.
- If the query does not match:
  - Print "Invalid SELECT query format."
  - Log the database change.

#### 3. Method getColumnIndices(columnsPart, headers, tableName):

- Determine column indices based on requested columns.
- Return column indices or handle errors.

## Pseudocode for DeleteFromTableClass for Transaction

### DeleteFromTable Class

#### 1. Class DeleteFromTable:

- transactionManager = new TransactionManagerImpl()

#### 2. Method deletes(query):

- Check if a database is selected.
- Match the query against the delete pattern.
- If the query matches:
  - Extract the table name, column, and value for deletion.
  - If a transaction is active:
    - Add the delete operation to the transaction buffer.
    - Print "Delete operation buffered."
    - Log the database change.
  - If no transaction is active:
    - Execute the delete operation directly on the table.
- If the query does not match:
  - Print "Invalid DELETE query format."
  - Log the database change.

#### 3. Method executeDelete(tableName, column, value):

- Get the table file.
- Read the table file.
- Validate the column and value for deletion.
- Perform the delete operation on the table file.
- If successful, print "Record(s) deleted successfully."
- Log the database change.

## Pseudocode for EventLog Class

### 1. Initialization:

- a. Initialize the logFilePath variable.

### 2. initialize(databaseName):

- a. Set logFilePath to "./databases/" + databaseName + "/event\_log.json".

### 3. logDatabaseChange(changeDescription):

- a. Create a log entry with the current timestamp and details including currentUserID and changeDescription.
- b. Add "databaseChange" as the type of log entry.
- c. Write the log entry to logFilePath.

### 4. logTransactionEvent(transactionEvent):

- a. Create a log entry with the current timestamp and details including currentUserID and transactionEvent.
- b. Add "transactionEvent" as the type of log entry.
- c. Write the log entry to logFilePath.

### 5. logCrashReport(crashDescription):

- a. Create a log entry with the current timestamp and details including currentUserID and crashDescription.
- b. Add "crashReport" as the type of log entry.
- c. Write the log entry to logFilePath.

### 6. writeLogEntry(logEntry):

- a. Open a FileWriter for logFilePath.
- b. Convert logEntry to JSON format using mapToJson() and write it to the file.
- c. Handle IOException if writing fails.

### 7. mapToJson(map):

- a. Initialize a StringBuilder json.
- b. Construct JSON format from the provided map, handling nested maps recursively.
- c. Return the constructed JSON string.

## Pseudocode for GeneralLog Class

- **Initialization:**
  - Initialize the logFilePath variable.
- **initialize(databaseName):**
  - Set logFilePath to "../databases/" + databaseName + "/general\_log.json".
- **log(query, executionTime, numberOfTables, totalRecords):**
  - Create a log entry with the current timestamp and details including query, executionTime, numberOfTables, and totalRecords.
  - Write the log entry to logFilePath.
- **writeLogEntry(logEntry):**
  - Open a FileWriter for logFilePath.
  - Convert logEntry to JSON format using mapToJson() and write it to the file.
  - Handle IOException if writing fails.

## Pseudocode for QueryLog Class

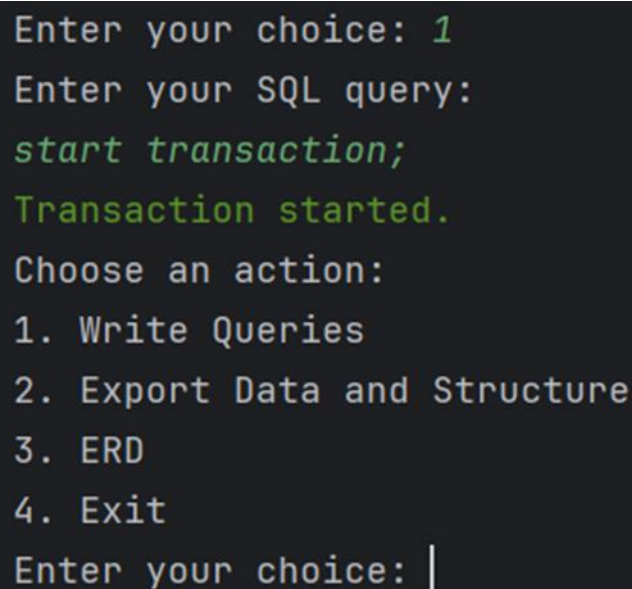
- **Initialization:**
  - Initialize the logFilePath variable.
- **initialize(databaseName):**
  - Set logFilePath to "../databases/" + databaseName + "/query\_log.json".
- **logUserQuery(query, timestamp):**
  - Create a log entry with the provided query, currentUserID, and timestamp.
  - Write the log entry to logFilePath.
- **writeLogEntry(logEntry):**
  - Open a FileWriter for logFilePath.
  - Convert logEntry to JSON format using mapToJson() and write it to the file.
  - Handle IOException if writing fails.

# Functional Testing:

## Transaction processing (Module 3):

- The operations performed by this module emphasizes the ACID property of the database maintaining the atomicity, consistency, isolation and durability
- The operation performed by this module either it is executed fully or it wont work fully i.e giving some error hence adhering to the atomicity.
- The operation performed by this module is fully consistent starting from transaction to committing the transaction database state is changed from one state to another and accessible that change to the other user hence maintaining the consistency.
- For isolation, any operation performed by one user within the transaction is stored in that buffer and at the same time if another transaction is on to the same table then second transaction is not able to access the first transaction's buffer data this is included in our code hence maintaining the isolation below images shows that, when doing insert from one user and select from another user so second user wont be able to access the first's once data adhering to isolation property.
- Once the data is written after committing the transaction to the txt file it will be saved to that file, and wont be deleted in case of failure of the different transactions, thus maintaining to the durability.

All these details given above are right, one can check the code provided in the gitlab repository for the confirmation.



```
Enter your choice: 1
Enter your SQL query:
start transaction;
Transaction started.
Choose an action:
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
Enter your choice: |
```

*Figure: Transaction started*

```
Enter your choice: 1
Enter your SQL query:
start transaction;
Transaction started.
Choose an action:
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
Enter your choice: 1
Enter your SQL query:
use student;
Using database student.
Choose an action:
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
Enter your choice: |
```

*Figure: executing query after starting the transaction*

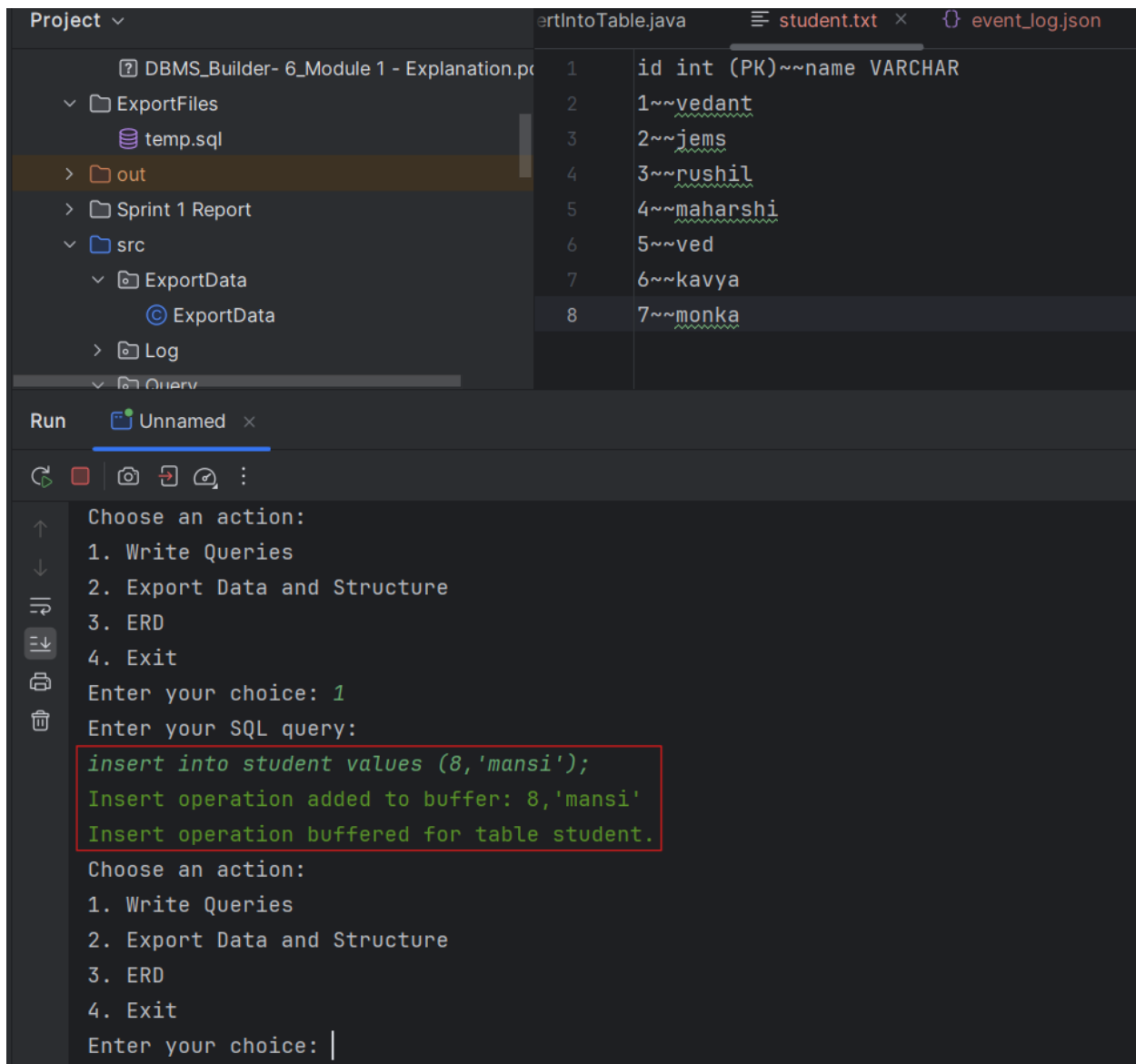


Figure: executing insert query after starting the transaction current table state not updated



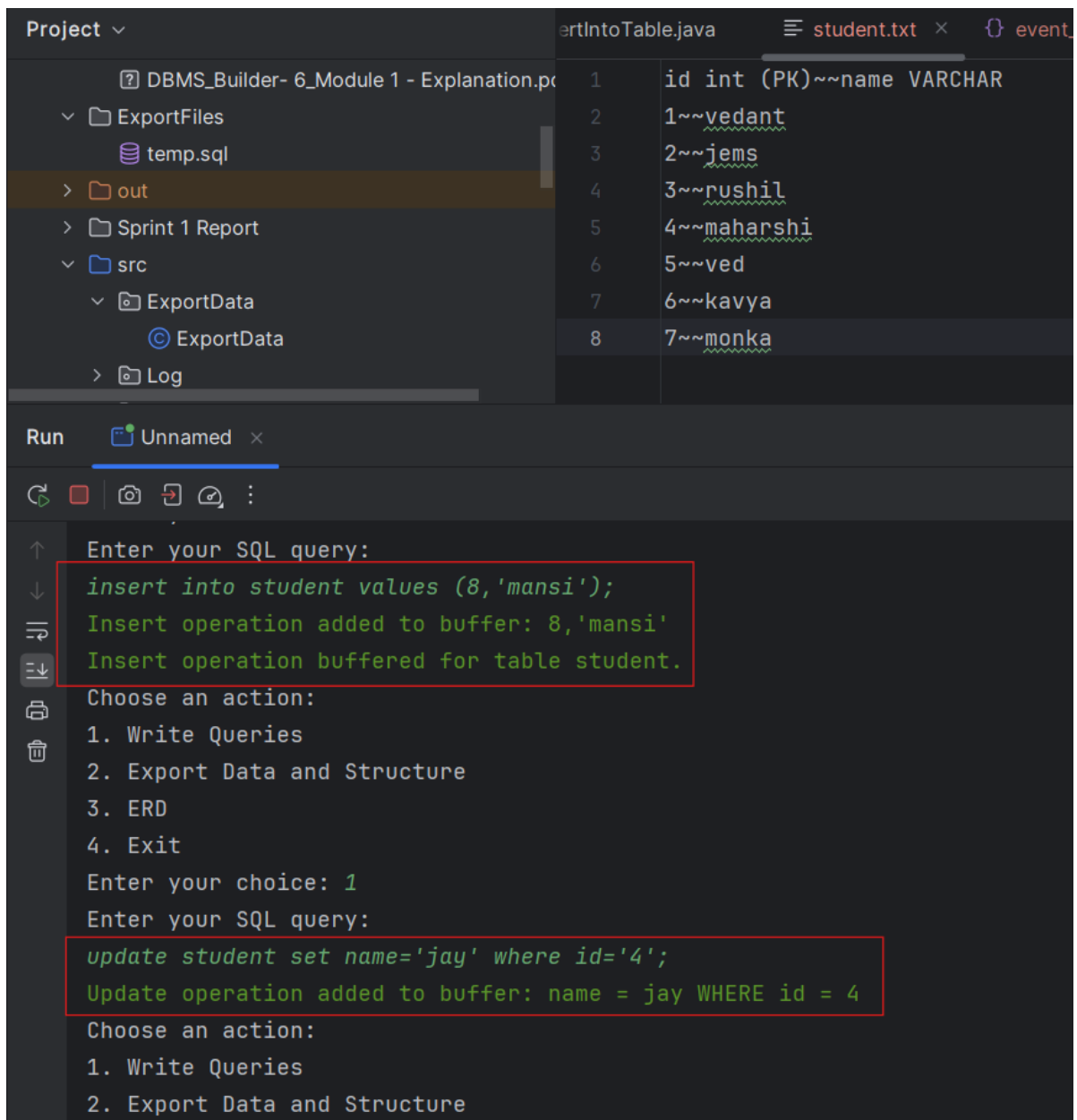


Figure: executing both queries simultaneously after the transaction started, within same transaction and table not updated

Data is changed from one state to another in the table while performing the transaction, hence maintaining consistency.

```
ertIntoTable.java
1 id int (PK)~~name VARCHAR
2 1~~vedant
3 2~~jems
4 3~~rushil
5 4~~jay
6 5~~ved
7 6~~kavya
8 7~~monka
9 8~~manshi
10

Run Unnamed x
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
Enter your choice: 1
Enter your SQL query:
commit transaction;
Record inserted successfully into table student.
Record(s) updated successfully.
Transaction committed.
Choose an action:
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
Enter your choice:
```

Figure: committing the transaction after executing some queries into buffer and table got updated, data transferred from buffer to table

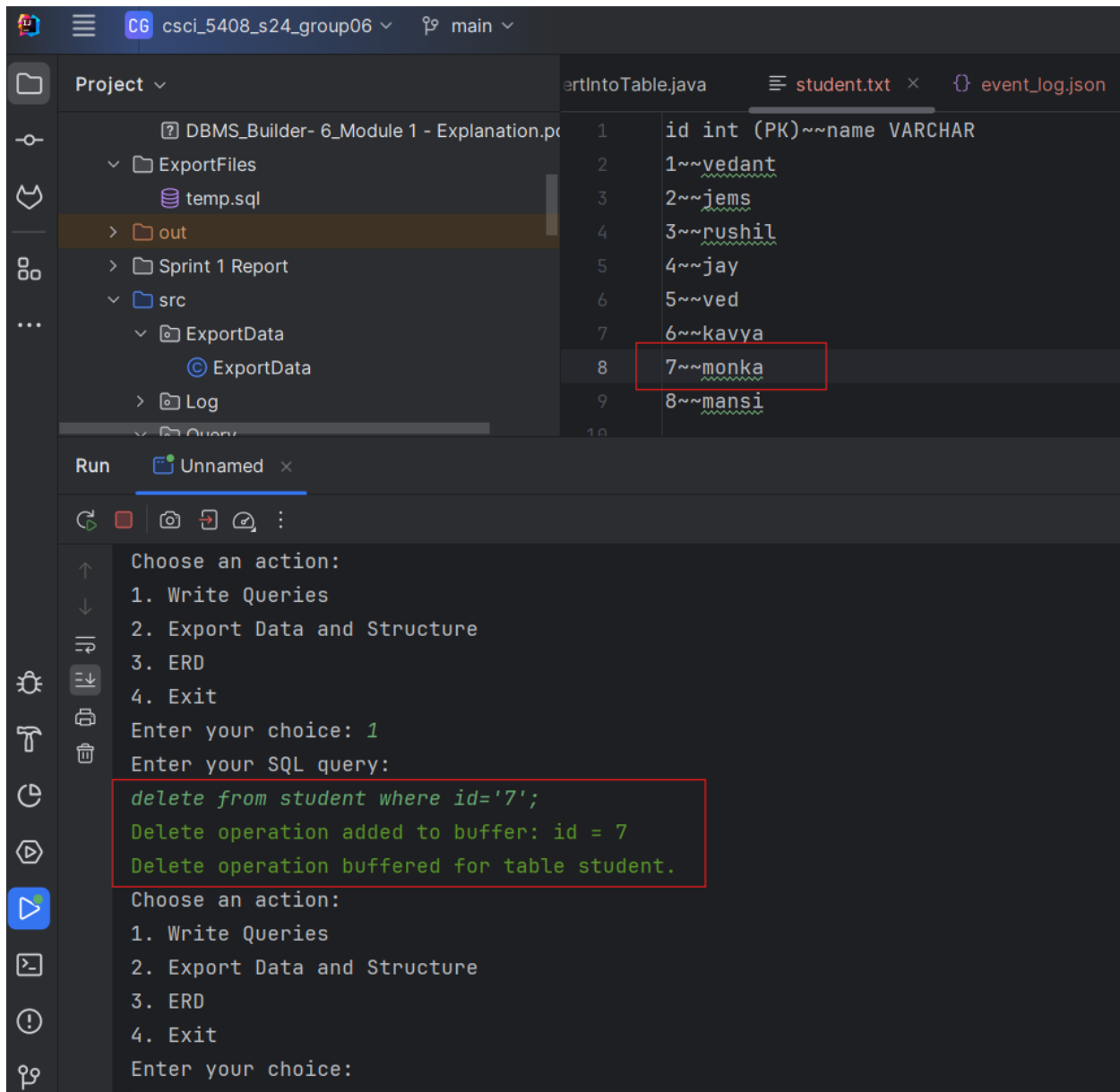


Figure: performing delete after the transaction query got executed still entry is present in the table

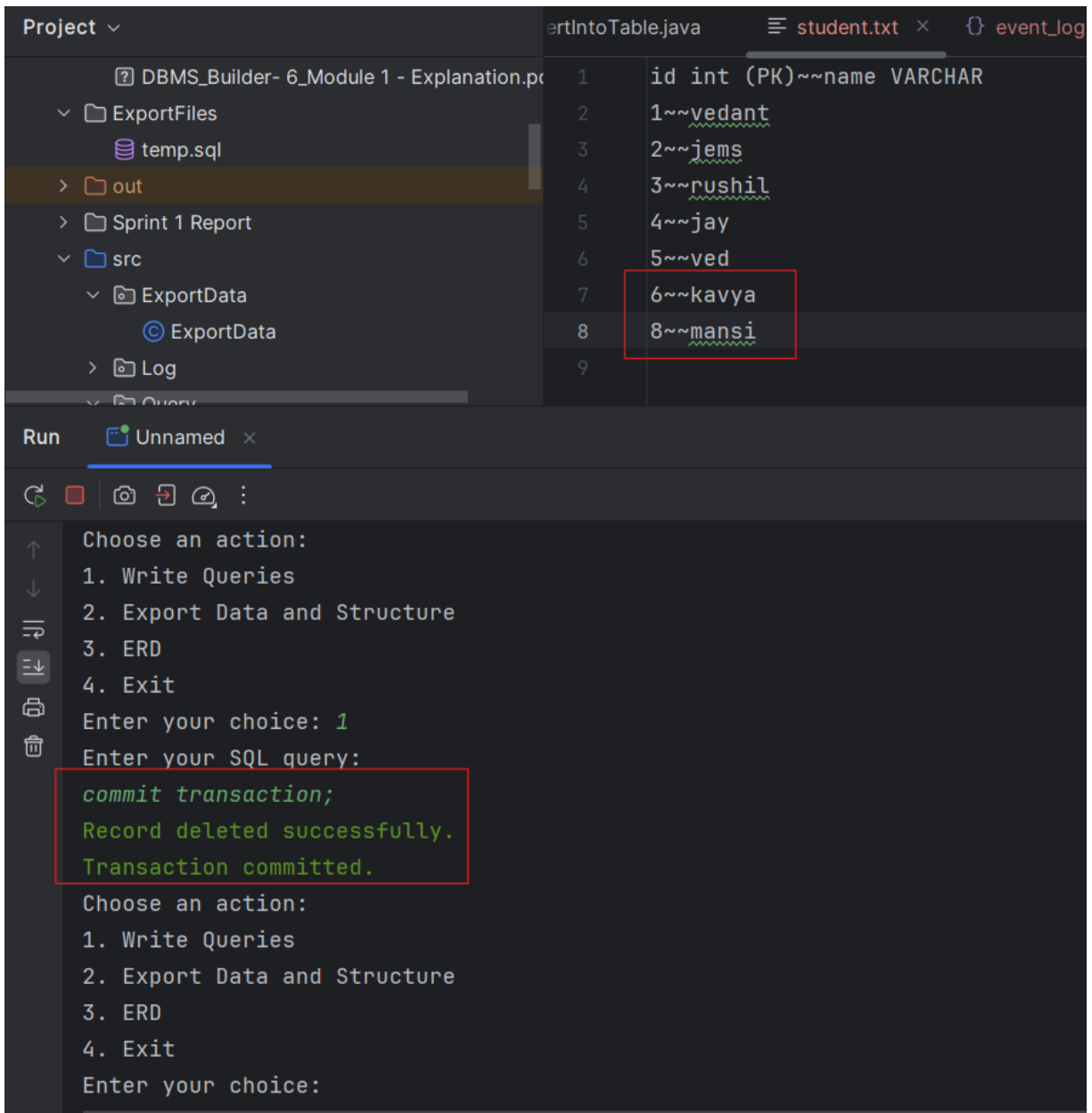


Figure: after committing entry with id 7 got deleted

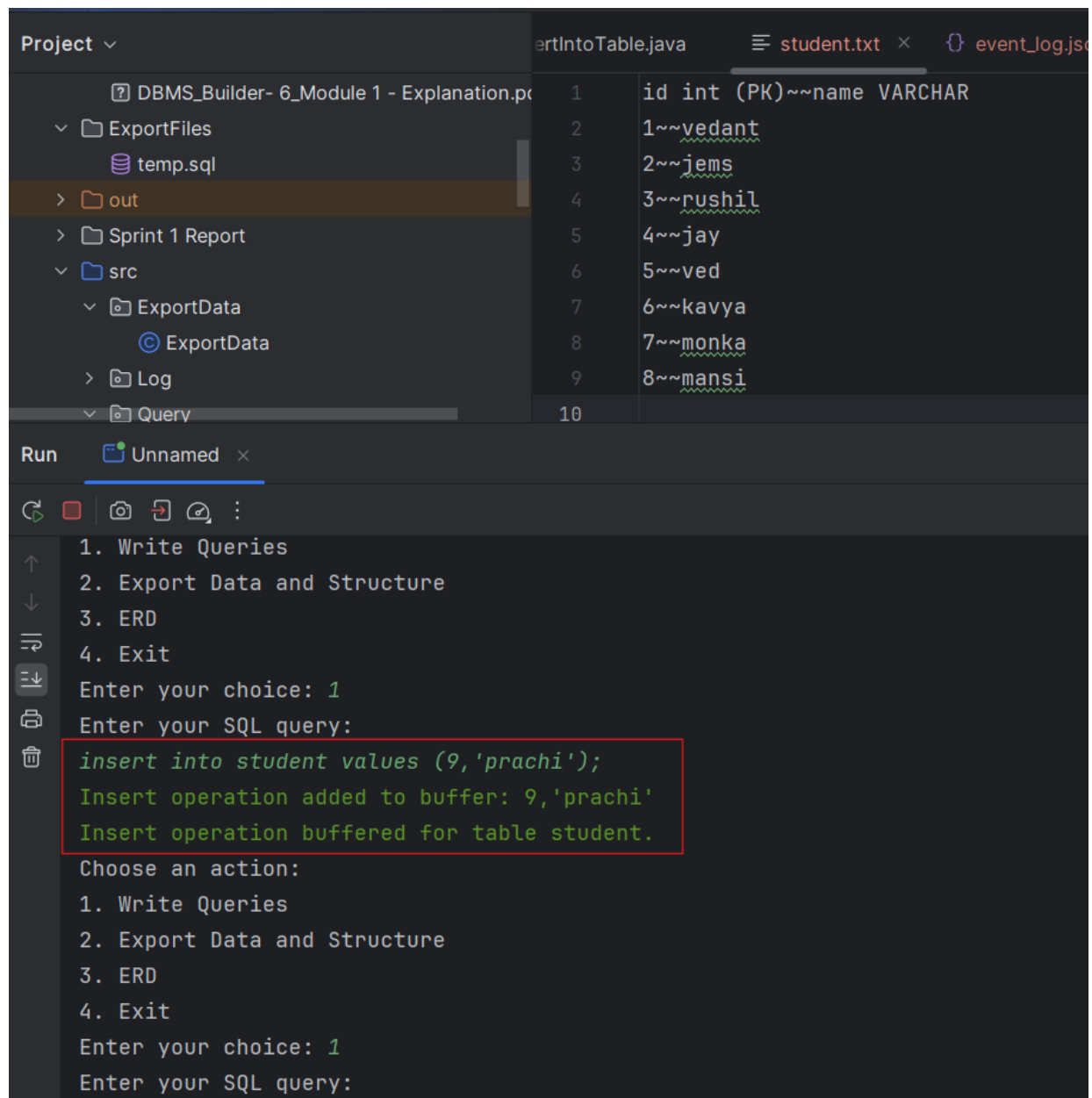


Figure: performed insert query before select query within the transaction

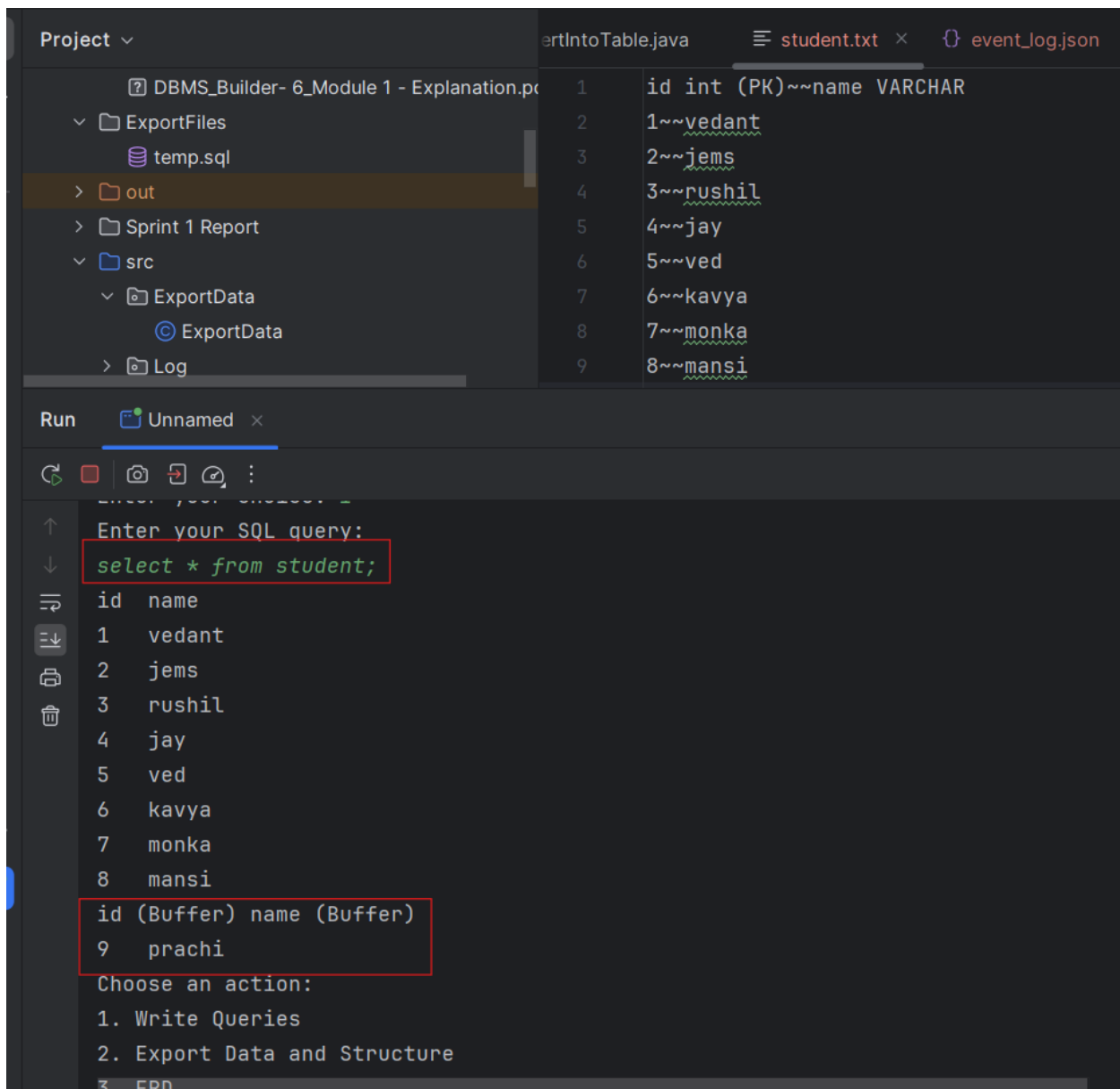


Figure: inserted data before select query this is current user buffer data it will not be shown other user's buffer data maintaining isolation

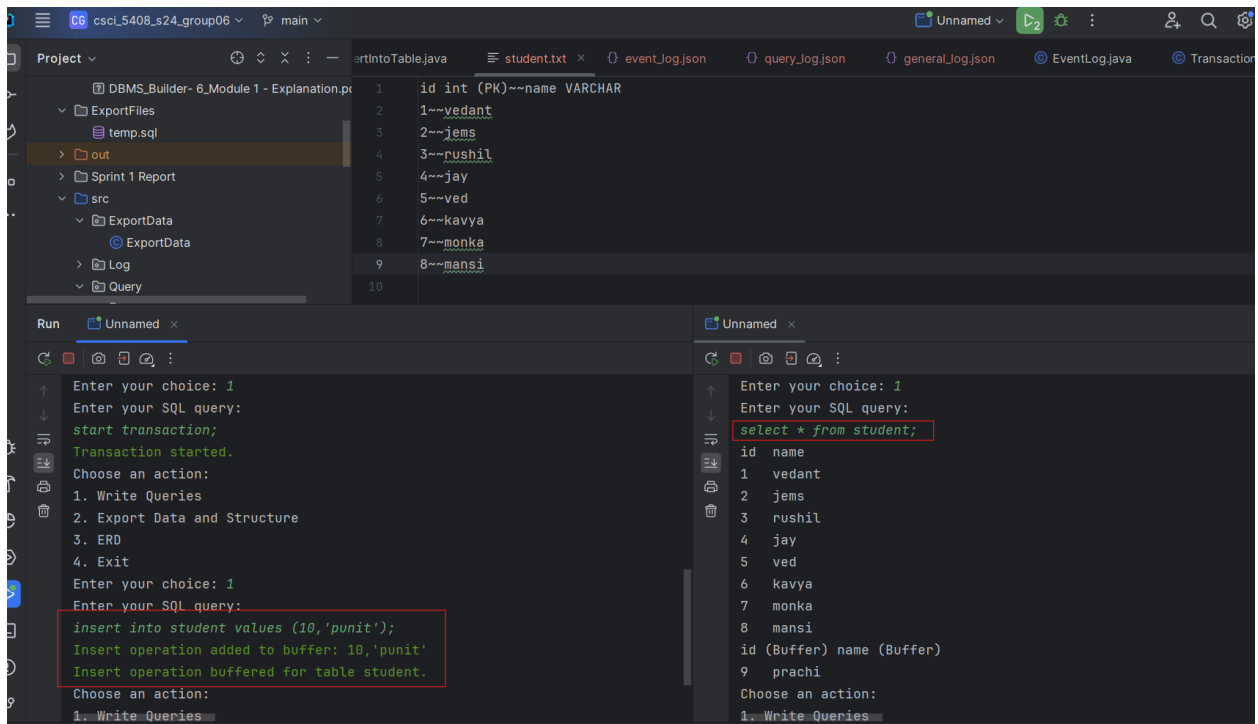


Figure: logged in with 2 different users and performed insert in one user side and select from another user side still another user is not able to get other user's buffer data before committing maintaining isolation property.

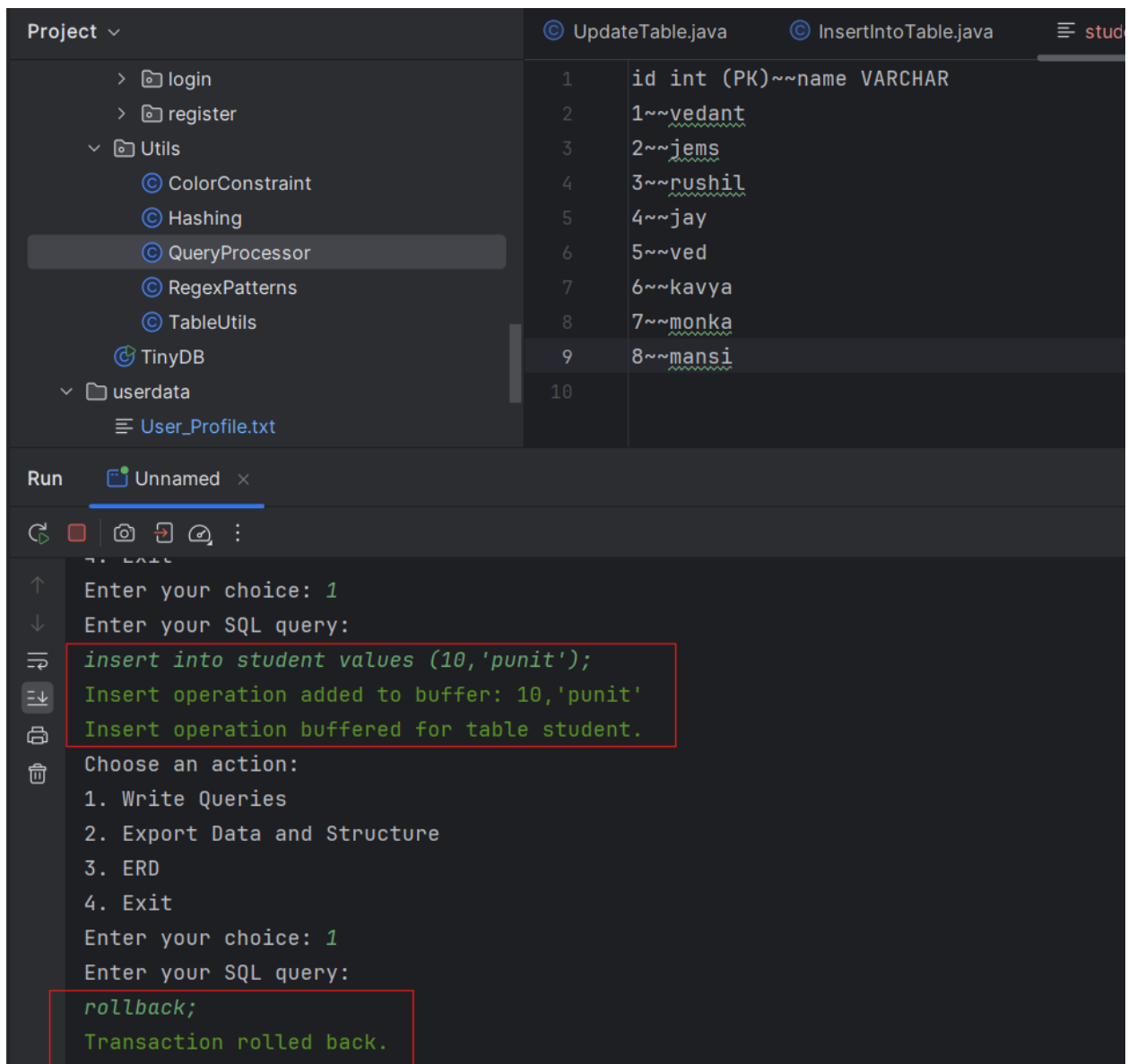


Figure: transaction rolled back after performing some operation within transaction and table not updated.



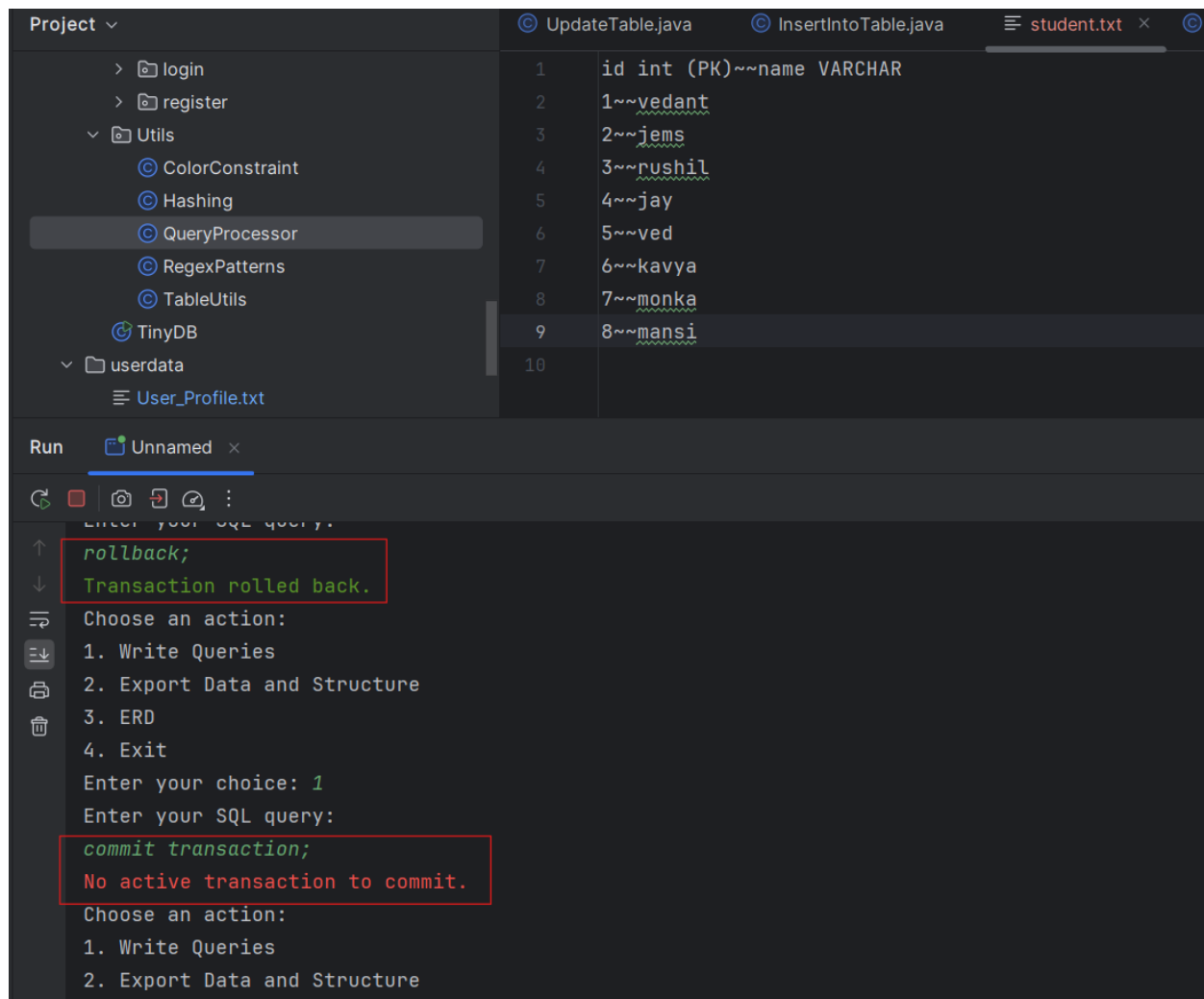
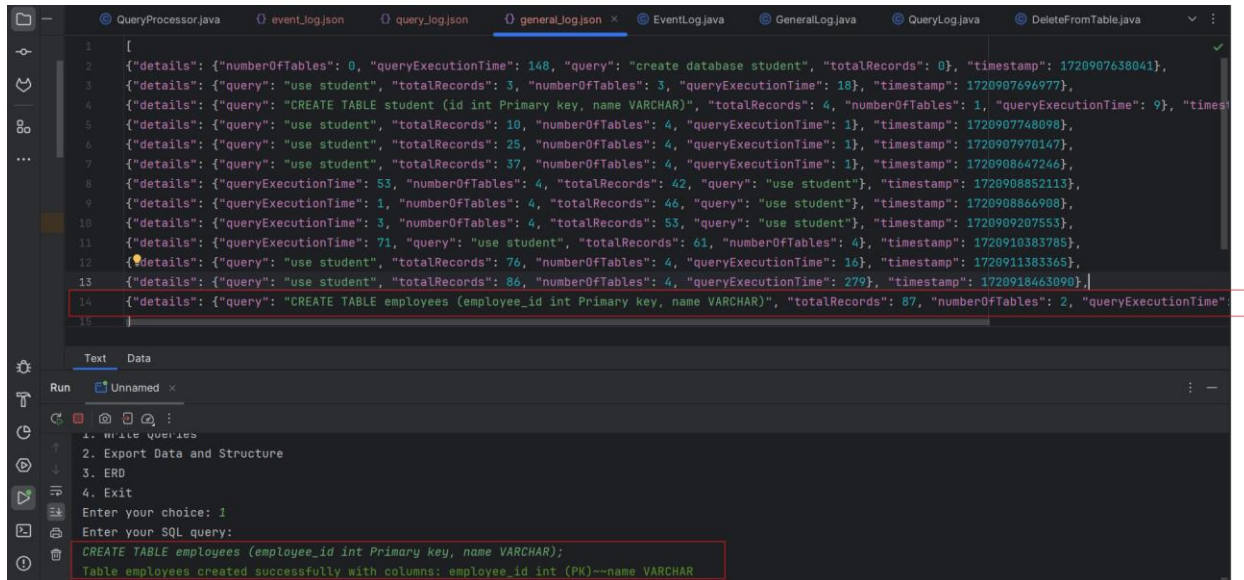


Figure: committing transaction after rolling it back.

## Log management (Module 4):



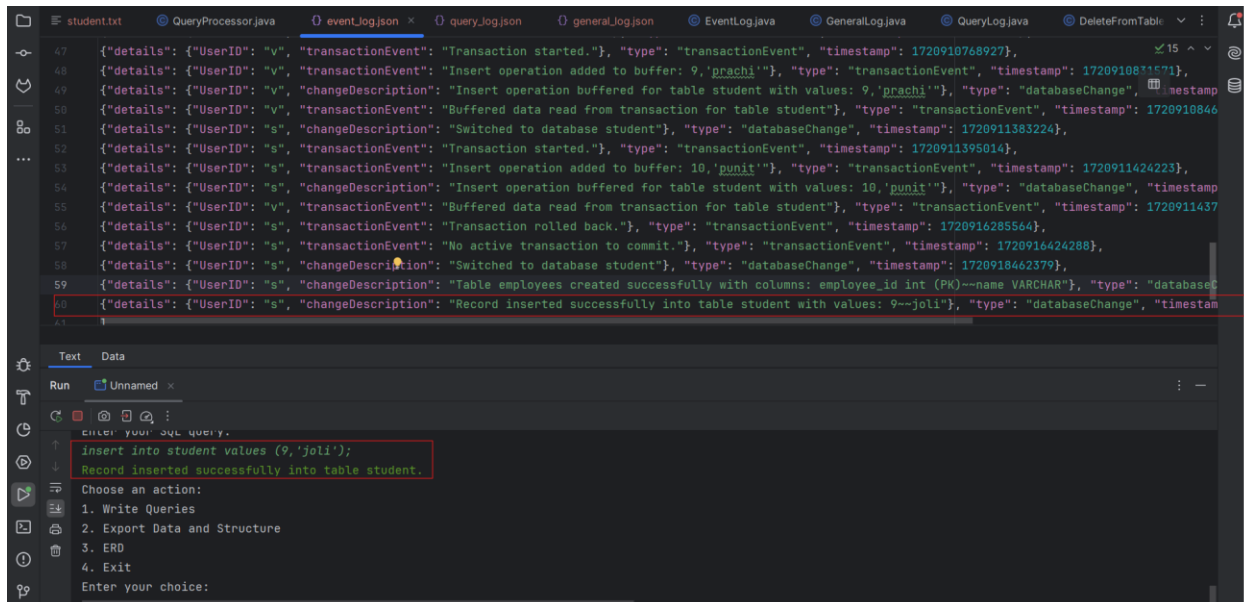
The screenshot shows a code editor with several files open. The 'general\_log.json' file is selected and displays a list of database events. The events are JSON objects containing details like 'query', 'totalRecords', 'numberOfTables', 'queryExecutionTime', and 'timestamp'. The last event, highlighted with a red box, is a 'CREATE TABLE' statement for 'employees'.

```
[{"details": {"numberOfTables": 0, "queryExecutionTime": 148, "query": "create database student", "totalRecords": 0, "timestamp": "1720907638041"}, {"details": {"query": "use student", "totalRecords": 3, "numberOfTables": 3, "queryExecutionTime": 18, "timestamp": "1720907696977"}, {"details": {"query": "CREATE TABLE student (id int Primary key, name VARCHAR)", "totalRecords": 4, "numberOfTables": 1, "queryExecutionTime": 9, "timestamp": "1720907748098"}, {"details": {"query": "use student", "totalRecords": 10, "numberOfTables": 4, "queryExecutionTime": 1, "timestamp": "1720907748098"}, {"details": {"query": "use student", "totalRecords": 25, "numberOfTables": 4, "queryExecutionTime": 1, "timestamp": "1720907748098"}, {"details": {"query": "use student", "totalRecords": 37, "numberOfTables": 4, "queryExecutionTime": 1, "timestamp": "1720908647246"}, {"details": {"queryExecutionTime": 53, "numberOfTables": 4, "totalRecords": 42, "query": "use student", "timestamp": "1720908852113"}, {"details": {"queryExecutionTime": 1, "numberOfTables": 4, "totalRecords": 46, "query": "use student", "timestamp": "1720908866908"}, {"details": {"queryExecutionTime": 3, "numberOfTables": 4, "totalRecords": 53, "query": "use student", "timestamp": "1720909207553"}, {"details": {"queryExecutionTime": 71, "query": "use student", "totalRecords": 61, "numberOfTables": 4, "timestamp": "1720910383785"}, {"details": {"query": "use student", "totalRecords": 76, "numberOfTables": 4, "queryExecutionTime": 16, "timestamp": "1720911383365"}, {"details": {"query": "use student", "totalRecords": 86, "numberOfTables": 4, "queryExecutionTime": 279, "timestamp": "1720918463090"}, {"details": {"query": "CREATE TABLE employees (employee_id int Primary key, name VARCHAR)", "totalRecords": 87, "numberOfTables": 2, "queryExecutionTime": 1, "timestamp": "1720918463090"}]
```

The bottom panel shows a terminal window with the following text:

```
Enter your choice: 1
Enter your SQL query:
CREATE TABLE employees (employee_id int Primary key, name VARCHAR);
Table employees created successfully with columns: employee_id int (PK)--name VARCHAR
```

Figure: general log after writing the create table query.



The screenshot shows a code editor with several files open. The 'event\_log.json' file is selected and displays a list of database events. The events are JSON objects containing details like 'User', 'transactionEvent', 'changeDescription', and 'timestamp'. The last event, highlighted with a red box, is a 'Record inserted successfully into table student'.

```
[{"details": {"User": "v", "transactionEvent": "Transaction started.", "type": "transactionEvent", "timestamp": "1720910768927"}, {"details": {"User": "v", "transactionEvent": "Insert operation added to buffer: 9,'prachi'", "type": "transactionEvent", "timestamp": "1720910811511"}, {"details": {"User": "v", "changeDescription": "Insert operation buffered for table student with values: 9,'prachi'", "type": "databaseChange", "timestamp": "1720910846"}, {"details": {"User": "v", "transactionEvent": "Buffered data read from transaction for table student", "type": "transactionEvent", "timestamp": "1720910846"}, {"details": {"User": "s", "changeDescription": "Switched to database student", "type": "databaseChange", "timestamp": "1720911383224"}, {"details": {"User": "s", "transactionEvent": "Transaction started.", "type": "transactionEvent", "timestamp": "1720911395014"}, {"details": {"User": "s", "transactionEvent": "Insert operation added to buffer: 10,'punith'", "type": "transactionEvent", "timestamp": "1720911424223"}, {"details": {"User": "s", "changeDescription": "Insert operation buffered for table student with values: 10,'punith'", "type": "databaseChange", "timestamp": "1720911437"}, {"details": {"User": "v", "transactionEvent": "Buffered data read from transaction for table student", "type": "transactionEvent", "timestamp": "1720911437"}, {"details": {"User": "s", "transactionEvent": "Transaction rolled back.", "type": "transactionEvent", "timestamp": "1720916285564"}, {"details": {"User": "s", "transactionEvent": "No active transaction to commit.", "type": "transactionEvent", "timestamp": "1720916424288"}, {"details": {"User": "s", "changeDescription": "Switched to database student", "type": "databaseChange", "timestamp": "1720918462379"}, {"details": {"User": "s", "changeDescription": "Table employees created successfully with columns: employee_id int (PK)--name VARCHAR", "type": "databaseChange", "timestamp": "1720918463090"}, {"details": {"User": "s", "changeDescription": "Record inserted successfully into table student with values: 9--joli", "type": "databaseChange", "timestamp": "1720918463090"}]
```

The bottom panel shows a terminal window with the following text:

```
Enter your SQL query:
Insert into student values (9,'joli');
Record inserted successfully into table student.
```

Figure: event log after inserting or performing any event related operation

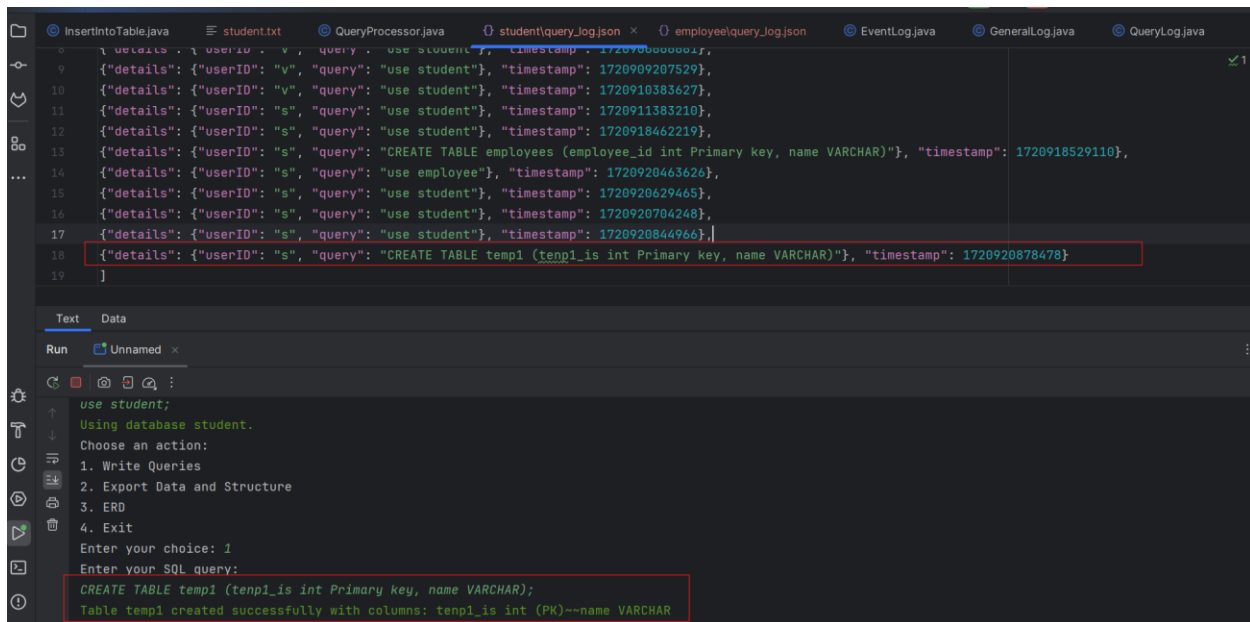
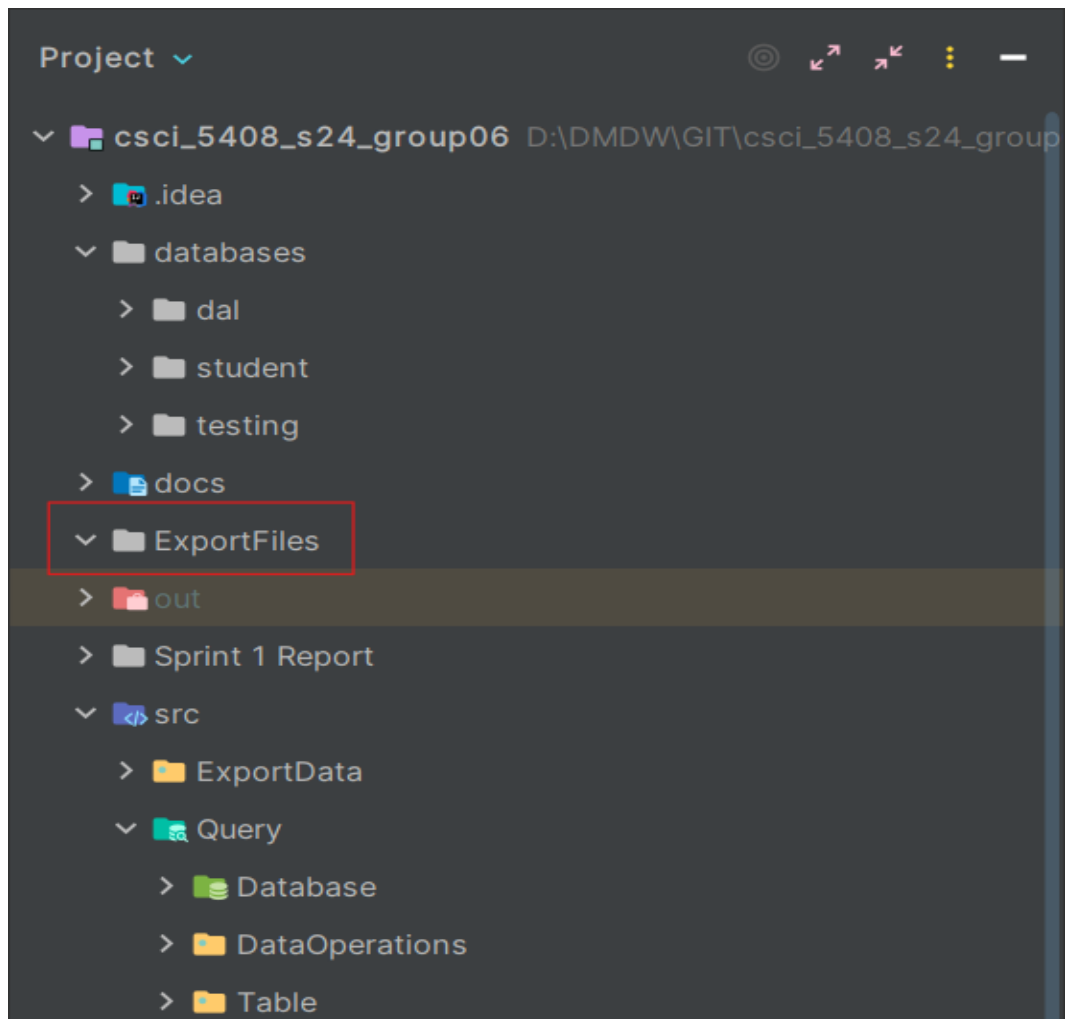
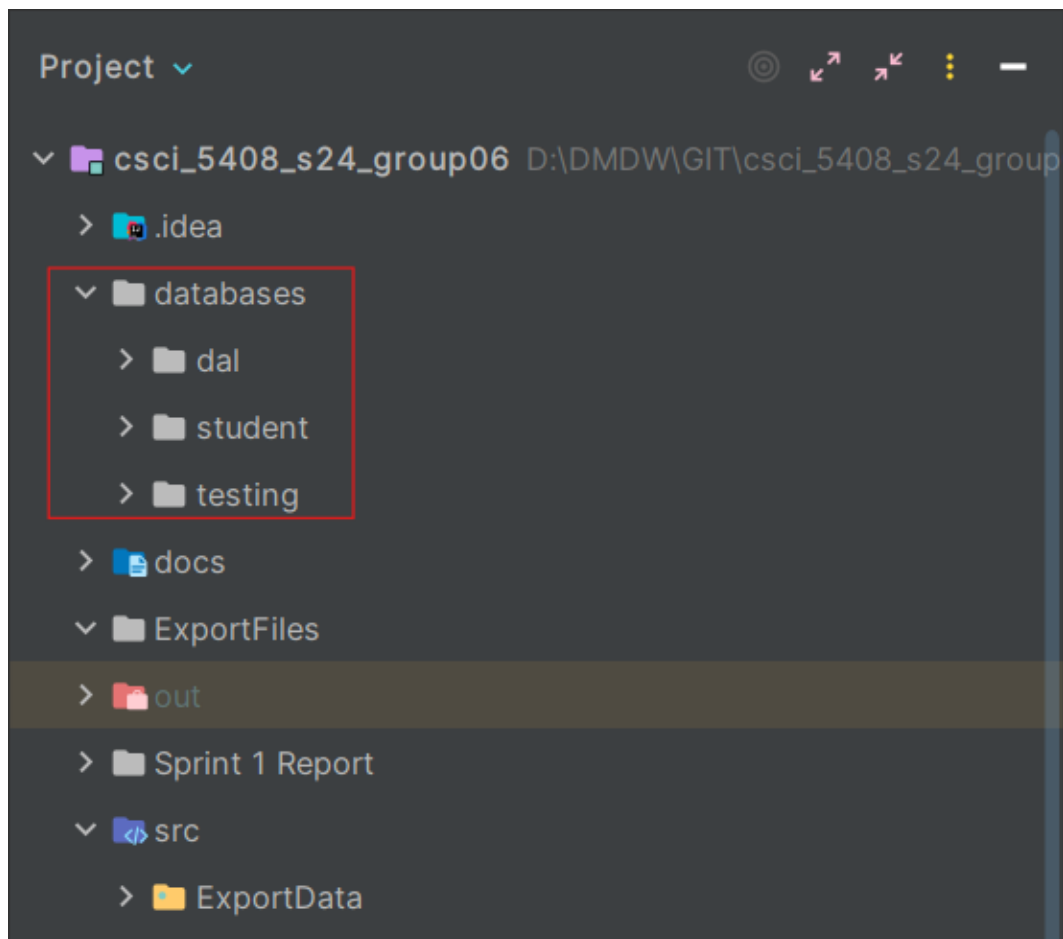


Figure: query logs after executing the create table commend

## Export Structure and Value (Module 6):



*Figure: Export Files folder to store the exported SQL files which user want.*



*Figure: Current created databases in the folder.*

```
Choose an action:
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
Enter your choice: 2
Enter the database name to export: dalhousie;
Database dalhousie; does not exist. Please try again.
Enter the database name to export: |
```

*Figure: Export database which does not exist.*

```
Choose an action:
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
Enter your choice: 2
Enter the database name to export: dal
Exporting data and structure...
Database export for dal completed successfully!
Choose an action:
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
Enter your choice: |
```

Figure: Export the database 'dal' which exists.

student.txt	1	faculty_id int (PK)~~name varchar~~course varchar
	2	1~~MIKE~~MACS
faculty.txt	3	2~~Saurabh Dey~~MCS
	4	3~~Shuntiang~~MACS

Figure: Current state of the faculty table in the dal database

student.txt	1	student_id int (PK)~~name varchar~~course varchar
	2	1~~jems~~MACS
faculty.txt	3	2~~vedant~~MCS
	4	3~~rushil~~MACS

Figure: Current state of the student table in the dal database

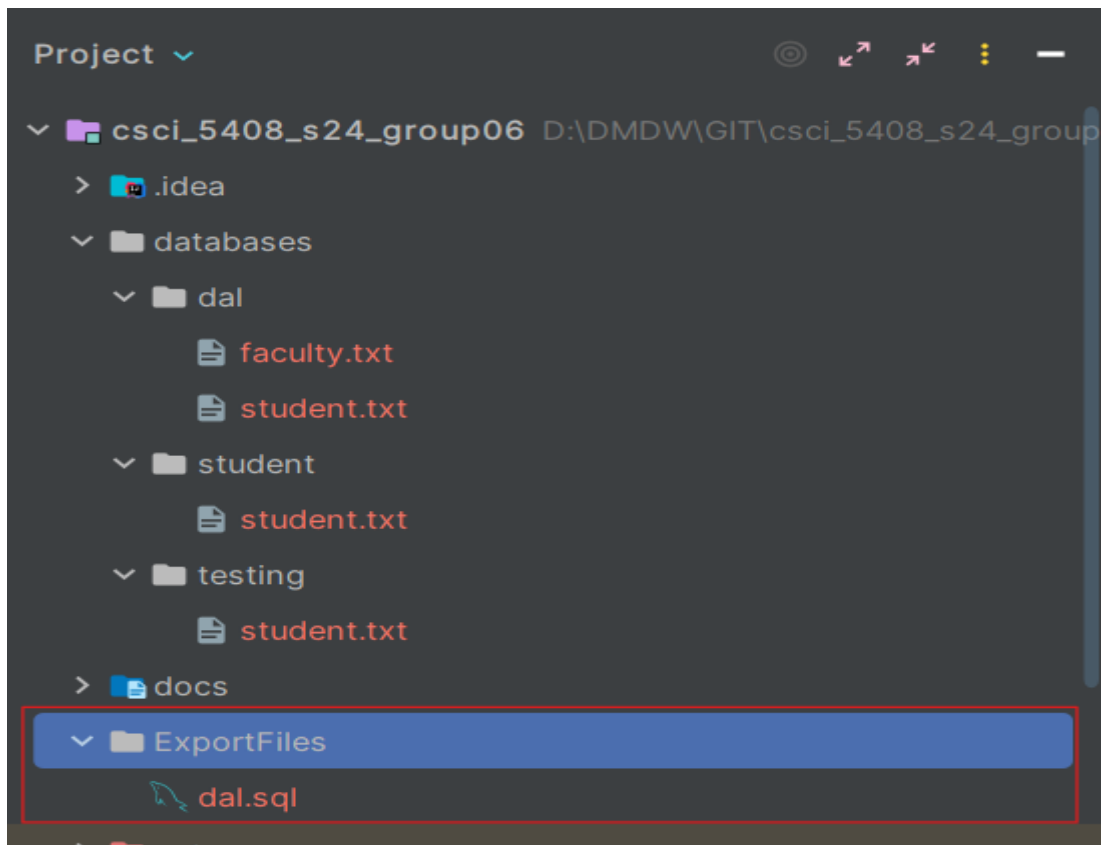


Figure: After exporting the database.

```

1  DROP TABLE IF EXISTS faculty;
2  CREATE TABLE faculty (
3      faculty_id int PRIMARY KEY,
4      name varchar,
5      course varchar
6  );
7
8  INSERT INTO faculty (faculty_id, name, course) VALUES ('1', 'MIKE', 'MACS');
9  INSERT INTO faculty (faculty_id, name, course) VALUES ('2', 'Saurabh Dey', 'MCS');
10 INSERT INTO faculty (faculty_id, name, course) VALUES ('3', 'Shuntiang', 'MACS');
11
12 DROP TABLE IF EXISTS student;
13 CREATE TABLE student (
14     student_id int PRIMARY KEY,
15     name varchar,
16     course varchar
17 );
18
19 INSERT INTO student (student_id, name, course) VALUES ('1', 'jems', 'MACS');
20 INSERT INTO student (student_id, name, course) VALUES ('2', 'vedant', 'MCS');
21 INSERT INTO student (student_id, name, course) VALUES ('3', 'pushil', 'MACS');
22

```

Figure: Exported file for the dal databasee

## Sprint 2 Meeting Logs:

Meeting Date	Topic of Discussion	Meeting Mode	Members	Outcome
1/072024 - Monday	Sprint 2 Planning and task division	In Person (at Goldberg CS building)	Rushil, Vedant, Jems	Vedant, Jems Discussed the tasks for Sprint 2. Rushil will handle Module 4: Log Management, Vedant will handle Module 3: Transaction Processing, and Jems will handle Module 6: Export Structure & Value.
4/07/2023 - Thursday	Mid-Progress Meeting	In Person (at Goldberg CS building)	Rushil, Vedant, Jems	Reviewed progress on the assigned modules. Rushil has started on the JSON logs for Log Management, Vedant is working on identifying transactions, and Jems is setting up export structure.
6/072023	Final changes and intergation	In Person (at Goldberg CS building)	Rushil, Vedant, Jems	Finalized and integrated the modules. Rushil completed the logging system, Vedant ensured transaction processing follows ACID properties, and Jems completed the export functionality.
12/07/2023	Final Report Discussion	In Person (at Goldberg CS building)	Rushil, Vedant, Jems	Discussed and compiled the final report. Reviewed each module's functionality and ensured all requirements were met. Prepared the documentation and finalized the submission.



### Sprint 1 meeting Recordings:

- Meeting 1 - [link](#)
- Meeting 2 - [link](#)

## References

- [1] Databricks. (n.d.). ACID Transactions. Retrieved July 13, 2024, from <https://www.databricks.com/glossary/acid-transactions#:~:text=properties%3A%20Atomicity%2C%20Consistency%2C%20Isolation,Consistency%2C%20Isolation%2C%20and%20Durability>.
- [2] Stack Overflow. (2022). HashMap using lists as a buffer. Retrieved July 13, 2024, from <https://stackoverflow.com/questions/71308108/hashmap-using-lists-as-a-buffer>
- [3] Stack Overflow. (2021). Write to a file in transactional manner. Retrieved July 13, 2024, from <https://stackoverflow.com/questions/67351093/write-to-a-file-in-transactional-manner>
- [4] S. Khalid, "How to write logs in text file when using java.util.logging.Logger," Stack Overflow, Apr. 2013. [Online]. Available: <https://stackoverflow.com/questions/15758685/how-to-write-logs-in-text-file-when-using-java-util-logging-logger>. [Accessed: 13-Jul-2024].
- [5] M. Smith, "Logging to JSON," Stack Overflow, Jan. 2012. [Online]. Available: <https://stackoverflow.com/questions/8904907/logging-to-json>. [Accessed: 13-Jul-2024].
- [6] Exabeam, "Event Logging," Exabeam Explainers. [Online]. Available: <https://www.exabeam.com/explainers/event-logging/event-log/>. [Accessed: 13-Jul-2024].
- [6] Atlassian, "What is the Query Log?" Atlassian Support. [Online]. Available: <https://support.atlassian.com/analytics/docs/what-is-the-query-log/>. [Accessed: 13-Jul-2024].
- [7] Oracle Corporation, "The general query log," MySQL 8.4 Reference Manual. [Online]. Available: <https://dev.mysql.com/doc/refman/8.4/en/query-log.html#:~:text=The%20general%20query%20log%20is,SQL%20statement%20received%20from%20clients>. [Accessed: 13-Jul-2024].