I. Overview and Schema

This paper goes over the effects of different isolation levels (READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ, SERIALIZABLE) in maintaining the ACID properties of a transaction.

Different isolation levels were observed on how it handles different types of concurrency problems (DIRTY READ, NON-REPEATABLE READ, PHANTOM READS). In order to do this, transactions corresponding to each concurrency problem were run over a sample database while observing the possible side-effects.

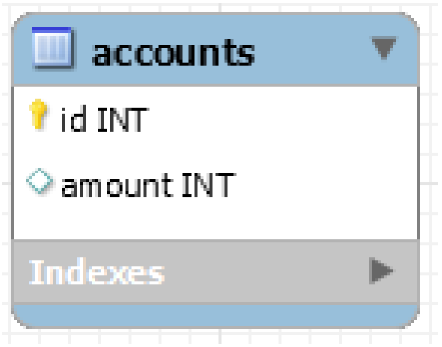

Figure 1. Database Schema

The provided database schema in Figure 1 shows the database schema used for the experiment. It contains the ‘accounts’ table which has only 2 attributes:

- ‘Id’: The identifier of each account
- ‘Amount’: A monetary value associated with each account

By repeating the experiment with different isolation levels, we can observe how each level mitigates or worsen concurrency problems, providing insights into the trade-offs between consistency and performance in database systems.

II. Concurrency Problems

```
INSERT INTO Accounts (amount) VALUES (1000);
```

Query 1. Data Initialization

This query adds an initial record to the database that will be utilized in the transactions used to demonstrate concurrency problems.

DIRTY READ

```
START TRANSACTION;
     UPDATE Accounts SET amount = amount + 1 WHERE id = 1;
     DO SLEEP(10);
COMMIT;
```

Transaction 1. Record Insertion

```
START TRANSACTION;
    SELECT * FROM Accounts;
COMMIT;
```

Transaction 2. Record read

To simulate a dirty read concurrency problem, two transactions were executed. Transaction 1 updates a value however it will remain uncommitted for 10 seconds due to the intentional delay used which introduces a possible dirty read in Transaction 2 during this time.

NON-REPEATABLE READ

```
START TRANSACTION;
    UPDATE Accounts
    SET amount = 2000
    WHERE id = 1;
COMMIT;
```

Transaction 1. Record Update

```
START TRANSACTION;
    SELECT * FROM Accounts WHERE id = 1;
    DO SLEEP(10);
    SELECT * FROM Accounts WHERE id = 1;
COMMIT;
```

Transaction 2. Record reread after 10 seconds

In simulating a non-repeatable concurrency problem, two transactions were also used. Transaction 2 will execute the same `SELECT` query twice with an intentional 10 second delay in between which provides an opportunity for Transaction 1 to update record values which may present a possible non-repeatable read.

PHANTOM READS

```
START TRANSACTION;
     INSERT INTO Accounts (amount) VALUES
    (rand() * 100),
    (rand() * 100),
    (rand() * 100),
    (rand() * 100),
    (rand() * 100);
COMMIT;
```

Transaction 1. Random Record Insertion

```
START TRANSACTION;
    SELECT AVG(amount) FROM Accounts;
    DO SLEEP(10);
    SELECT AVG(amount) FROM Accounts;
COMMIT;
```

Transaction 2. Record reread after 10 seconds

Following the two previous concurrency problems, two transactions were also used in simulating phantom reads. Similar to the non-repeatable read problem previously, Transaction 2 will re-execute the same query as the initial query after a 10 seconds delay where the addition of records by Transaction 1 is possible, introducing possible phantom reads with the aggregate function by Transaction 2.

To simulate each concurrency problem, two instances of MySQL Workbench were used to execute each transaction present. In the dirty read concurrency problem, transaction 1 was started first and transaction 2 was only executed when transaction 1 is in its `DO SLEEP()`

query. On the other hand the non-repeatable and phantom read concurrency problems were executed in a sequence where transaction 1 was executed while transaction 2 is in its `DO SLEEP()` query after the first `SELECT` query.

III. Isolation Levels Results

<u>READ UNCOMMITTED</u>

Transactions executed with a "READ UNCOMMITTED" isolation level were found to be vulnerable to every tested concurrency problem. Dirty reads were observed as a given transaction was able to read data that was yet to be committed at the time of the first call. In the non-repeatable read trial, there did turn out to be an inconsistency in each time that the data was read. Finally, in the phantom reads trial, 2 different values for the average are given as additional rows are inserted into the database in the middle of the first transaction.

<u>READ COMMITTED</u>

Based on the results of the experiments done on Read Committed isolation level, while this isolation level prevents dirty reads, it is still susceptible to both non-repeatable and phantom reads. In the dirty read experiment, transaction 2 returned the initial data despite transaction 1 updating a row because this change is still yet to be committed which shows the prevention of the isolation level to dirty reads. The non-repeatable and phantom read experiments however, presented the issue of inconsistent reads of `SELECT` queries which should return the same results. In particular, the `SELECT` queries in transaction 2 in the non-repeatable read returned the same row with a different `amount` value while it returned a different set of rows in the phantom test experiment.

<u>REPEATABLE READ</u>

The isolation level provides a higher level of data consistency by preventing dirty reads and non-repeatable reads while allowing some possibility of phantom reads. In the conducted tests, despite intentionally delaying the commit of updates in the first transaction to simulate a dirty read scenario, the second transaction successfully avoided reading uncommitted data, indicating the efficacy of the repeatable read isolation level in preventing dirty reads. Similarly, repeated reads within the same transaction yielded consistent values, demonstrating the effectiveness of the isolation level in preventing non-repeatable reads. Surprisingly, reproducing phantom reads in MySQL is difficult considering InnoDB uses a multiversion concurrency control. The results from consistent queries were expected to show different results. Yet, in multiple attempts, the select statements produced the same output. It seems that for repeatable reads, consistent reads on the same transaction use the snapshot of the db established by the first query in that transaction (MySQL, n.d.).

The experiment shows its efficacy in preventing dirty and non-repeatable reads, although phantom reads are still possible, albeit mitigated. While phantom reads were difficult to reproduce in MySQL due to its amazing concurrency control mechanisms, the inherent strengths of the repeatable read isolation level were evident.

SERIALIZABLE

Running the same experiments from above with a SERIALIZABLE isolation level shows that this isolation level prevents all the concurrency problems discussed. Mocking a dirty read scenario by imposing a delay before the commit of the first transaction while simultaneously having a second transaction read the same data returned an output where the second transaction did not read the uncommitted changes of the first transaction. This shows that this isolation level is secure from having dirty reads occur. This isolation level passed the non-repeatable read test where a problem is found when a transaction has a repeated read on the same row with the results appearing to be inconsistent for each read. The experiment showed the data output from the SERIALIZABLE level was consistent. Furthermore, the same consistency is shown with the test for phantom read where a transaction has two read queries while the other translation inserts new records that affect the query of the first transaction. The findings from the tests show that the isolation level of SERIALIZABLE is secure and safe from the three concurrency problems that were tested.

IV. Insights

The choice of isolation levels in a Database Management System (DBMS) is crucial for application developers because it directly impacts data consistency and concurrency controls. By allowing developers to select the appropriate isolation level, DBMS empowers them to balance between data consistency and performance based on the specific needs of their applications.

For instance, the READ UNCOMMITTED isolation level provides the least restrictive environment, allowing transactions to read uncommitted data from other transactions. While this can offer high concurrency, it also introduces risks such as dirty reads, non-repeatable reads, and phantom reads, as observed in the experiments. This level might be suitable for certain applications in analytics or reporting where real-time data access is prioritized over data consistency.

On the other hand, READ COMMITTED addresses the issue of dirty reads by ensuring that transactions only read committed data. However, it still allows for non-repeatable reads and phantom reads, which can lead to inconsistent query results.Developers might choose this level when they require a balance between performance and data consistency, such as in e-commerce applications where real-time inventory updates are necessary but can tolerate occasional inconsistencies in read operations.

REPEATABLE READ offers a higher level of data consistency by preventing dirty reads and non-repeatable reads. While it allows for some possibility of phantom reads, modern DBMSs with advanced concurrency control mechanisms like multiversion concurrency control (MVCC) mitigate this risk effectively. This level might be suitable for financial systems or inventory management applications where maintaining strict consistency is a priority.

Finally, SERIALIZABLE provides the highest level of isolation by ensuring that transactions are executed in a serializable manner, preventing all types of concurrency issues. However, this level might introduce higher overhead and reduced performance due to increased locking. This level is suitable for applications where data integrity is paramount and where strict adherence to transactional boundaries is necessary, such as in banking or healthcare systems.

Insights from these experiments emphasize the importance of carefully selecting the appropriate isolation level based on the specific requirements of the application. The ability to choose isolation levels allows application developers to tailor their applications' concurrency control mechanisms to balance between data consistency and performance based on specific requirements, ensuring optimal application behavior in different scenarios.

V. Declarations

References

MySQL. (n.d.). *MySQL 8.0 Reference Manual: 17.7.2.3 Consistent nonlocking reads*. Retrieved from https://dev.mysql.com/doc/refman/8.0/en/innodb-consistent-read.html

Contribution

Name	Contribution
Corpuz, Joshua	Insights Section, *Repeatable Read* Section
Dimagiba, Rafael	Overview and Schema Section, *Read Uncommitted* Section
Escopete, Steven Errol	Transactions Section, *Read Committed* Section
Pineda, Ralph	*Serializable* Section