Databases (Key: 1644)
Isolation levels in relational databases

---------- Arellanes Conde Esteban ----------

Contents

Homework: Investigate the Isolation levels in relational databases.

Recovered from: `https://www.geeksforgeeks.org/transaction-isolation-levels-dbms/`

The levels of transaction isolation in DBMS determine how the concurrently running transactions behave and, therefore, ensure data consistency with performance being even. There are four basic levels: Read Uncommitted, Read Committed, Repeatable Read, and Serializable that provide different degrees of data protection from providing fast access with possible incoherence and strict accuracy at the cost of performance. It depends upon choosing the right one based on whether the need is speed

or data integrity.

1 Isolation Levels

Recovered from: `https://www.cockroachlabs.com/blog/sql-isolation-levels-explained/`

As we said earlier there are four basic levels, and according to PostgreSQL 16 and lower shows these are the SQL standard isolation levels:

The SQL standard and PostgreSQL-implemented transaction isolation levels are described in Table 13.1.

Table 13.1. Transaction Isolation Levels

Isolation Level	Dirty Read	Nonrepeatable Read	Phantom Read	Serialization Anomaly
Read uncommitted	Allowed, but not in PG	Possible	Possible	Possible
Read committed	Not possible	Possible	Possible	Possible
Repeatable read	Not possible	Not possible	Allowed, but not in PG	Possible
Serializable	Not possible	Not possible	Not possible	Not possible

Figure 1: Postgres Isolation Levels. Credits: *www.cockroachlabs.com*

1.1 Read Uncommitted

`READ UNCOMMITTED` is the lowest isolation level, where no locks are applied, allowing dirty reads (i.e., reading uncommitted data from other transactions). This means data can change, and rows may appear/disappear before the transaction finishes.

Recommendation: Avoid using `READ UNCOMMITTED` except for non-modifying tasks like large-scale data scanning where accuracy isn't critical (e.g., generating analytics or summaries).

1.2 Read Committed

`READ COMMITTED` is the default isolation level for many databases. It ensures that transactions only read committed data, preventing dirty reads. However, it doesn't guarantee repeatability—data can change between reads within the same transaction.

Best Use: Ideal for applications that need to avoid dirty reads but can tolerate slight inconsistencies between consecutive reads (e.g., e-commerce inventory displays). It balances consistency and

performance without excessive retry errors.

1.3 Repeatable Read

`REPEATABLE READ` ensures that if a row is read twice in a transaction, it returns the same data, preventing non-repeatable reads. However, it may still require locks to prevent anomalies in more complex scenarios.

Best Use: Best for read-only transactions, such as calculating a user's balance, where consistency across multiple reads is important but without the overhead of stricter isolation levels.

1.4 Serializable

`SERIALIZABLE` provides the highest isolation, preventing dirty reads, non-repeatable reads, and phantom reads by ensuring complete isolation between transactions. However, it may result in more transaction retries due to its strict checks.

Best Use: Ideal for applications needing strict data consistency (e.g., financial systems), where transaction integrity is critical, and anomalies could cause significant issues.

2 ACID Properties

Recovered from: `https://www.geeksforgeeks.org/acid-properties-in-dbms/`

Isolation levels define the degree to which a transaction must be isolated from the data modifications made by any other transaction in the database system. A transaction isolation level is defined by the following phenomena:

- **Dirty Read** – A Dirty read is a situation when a transaction reads data that has not yet been committed. For example, Let's say transaction 1 updates a row and leaves it uncommitted, meanwhile, Transaction 2 reads the updated row. If transaction 1 rolls back the change, transaction 2 will have read data that is considered never to have existed.

- **Non Repeatable read** – Non-repeatable read occurs when a transaction reads the same row twice and gets a different value each time. For example, suppose transaction T1 reads data. Due to concurrency, another transaction T2 updates the same data and commit, Now if transaction T1 rereads the same data, it will retrieve a different value.

- **Phantom Read** – Phantom Read occurs when two same queries are executed, but the rows retrieved by the two, are different. For example, suppose transaction T1 retrieves a set of rows that satisfy some search criteria. Now, Transaction T2 generates some new rows that match the search criteria for Transaction T1. If transaction T1 re-executes the statement that reads the rows, it gets a different set of rows this time.

2.1 Atomicity: "All or Nothing"

Atomicity ensures a transaction is fully completed or not executed at all—there's no middle ground. If a transaction involves multiple operations and one fails, the entire transaction is rolled back, leaving the database unchanged. This prevents partial updates and ensures consistency.

Example: In a money transfer from Account X to Account Y, if the transfer fails halfway, the database remains unchanged. If one part fails, the entire transaction is rolled back to its original state.

2.2 Consistency: Valid Data States

Consistency ensures that a transaction only transitions the database from one valid state to another, maintaining all data rules and constraints. If a transaction violates any rule, it's rejected, ensuring the database always contains valid data.

Example: If a bank's total balance is $700 before a transfer, it must still be $700 after. If the transfer fails in the middle, the transaction is rolled back to maintain the balance.

2.3 Isolation: Preventing Interference Between Transactions

Isolation ensures that concurrent transactions don't interfere with each other, preventing inconsistencies like dirty reads, non-repeatable reads, or phantom reads. Each transaction operates as if it's the only one happening, guaranteeing the database's state is consistent even when multiple transactions are running.

Example: Transaction T transfers $50 between accounts X and Y. Transaction T *"reads X and Y at the same time, but if T is still running, T"* must not see the intermediate state—only the final values after T commits.

2.4 Durability: Persistence of Changes

Durability guarantees that once a transaction is committed, its changes are permanently saved to disk, even if a system failure occurs. The database can recover to the last committed state, ensuring no data is lost.

Example: After a successful money transfer, the changes are saved to disk. If the system crashes immediately after the transaction, the transfer will still be intact when the system recovers.

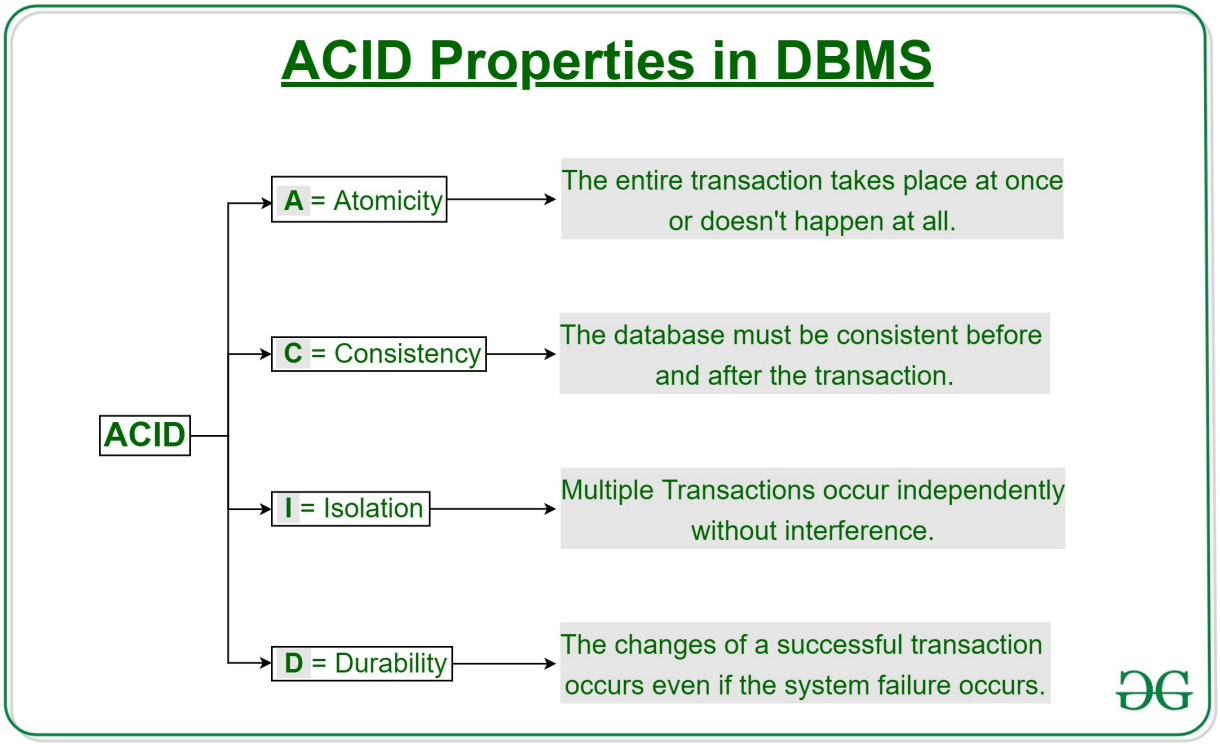

Figure 2: ACID Properties. Credits: *www.geeksforgeeks.org*

3 References

[1] Ramez Elmasri, Shamkant B. Navathe. Fundamentals of Database Systems (Global Edition). Pearson 2017

[2] De Miguel Martínez, Adoración, Piattini, Mario, Esperanza, Marcos Diseño de bases de datos relacionales. México, Alfaomega, 2000.

[3] Kriegel, Alex; Trukhnov. SQL Bible, second edition, Willey 2008.

[4] Alan Beaulieu, Learning SQL: Generate, Manipulate, and Retrieve Data, O'Reilly, 3rd. Edition - 2020.

[5] Joan Casteel, Oracle 12c: SQL. Cengage Learning, Sep 11, 2015.

[6] PostgreSQL Global Development Group, "PostgreSQL Documentation," `https://www.postgresql.org/docs/`.

[7] IEEE, "Guide to Software Requirements Specifications," IEEE Std 830-1984. `https://chumpolm.files.wordpress.com/2018/09/ieee-std-830-1984.pdf`