Atomicity

by Sophia

WHAT'S COVERED

This lesson explores the atomicity property in a transaction and how it affects the database, in two parts. Specifically, this lesson will cover:

1. Atomicity in Transactions

Transactions in a database consist of multiple SQL statements that are executed together. **Atomicity** is important as it ensures that each transaction is treated as a single statement. Atomicity ensures that if any of the SQL statements in a transaction fail, the entire transaction fails, and the attempted changes within the transaction are reverted. If all of the statements in a transaction are executed successfully, then the transaction is successful and committed.

This approach prevents the database from making updates that may only be partially completed. The database will do one of two operations to ensure atomicity. It will either:

1. Commit—If the transaction is successful, the changes are applied and saved to the database.
2. Abort—If a transaction has any issues, the transaction is aborted, and the changes are rolled back so that they are not reflected in the database.

This includes all INSERT, UPDATE and DELETE statements in a transaction.

TERM TO KNOW

Atomicity
The quality of indivisibility. In SQL database terms, atomicity means the transaction cannot be partially executed: Either all statements are executed or none of them are.

2. Transaction Example

EXAMPLE Jennifer would like to make a payment to Randall for $100 through an account transfer. This transaction is a balance transfer between two accounts at two different branches of the same bank. Let us take a look at what the transaction would look like:

1. $100 would be deducted from Jennifer's (10) account.
2. The banking location where Jennifer has her account would have $100 deducted from their location's account.
3. The banking location's account where Randall (50) has his account would be increased by $100.
4. Randall's account would be increased by $100.

The transaction would look something like this in PostgreSQL:

```
BEGIN;
UPDATE customer_account
SET balance = balance - 100
WHERE account_id = 10;
UPDATE branch_account
SET balance = balance - 100
WHERE branch_id = (SELECT branch_id FROM customer_account where account_id = 10);
UPDATE branch_account
SET balance = balance + 100
WHERE branch_id = (SELECT branch_id FROM customer_account where account_id = 50);<br>
UPDATE customer_account
SET balance = balance +100
WHERE account_id = 50;
COMMIT
```

With the atomicity property, if there were an error at any point in the four statements, then the entire transaction would be rolled back. For example, imagine that Randall's account had a freeze on it that prevented any changes. The first three statements would execute, but an error would be returned on the fourth UPDATE statement. Regardless of the error, the first three SQL statements would revert back to what they were before the transaction started. Otherwise, $100 would be deducted from Jennifer's account, the bank branch that holds Jennifer's account would have $100 deducted from their balance, Randall's bank branch would have $100 added, but Randall's account would have its original balance. That certainly would not be acceptable to Randall.

SUMMARY

Atomicity in SQL transactions ensures that a sequence of database operations is treated as a single indivisible unit of work. SQL atomicity ensures that all operations within a transaction are executed successfully or none of them are executed if an error occurs. The purpose of this property is to prevent partial or incomplete changes to the database. This ensures the integrity and consistency of the data. Committing a transaction signifies that all its constituent operations have been successfully executed and that the changes have been permanently saved. If an error occurs during the execution of a transaction's operation, the entire transaction will be rolled back, restoring the database to its state prior to its initiation. This prevents scenarios where some changes are applied, but others are left incomplete regardless of whether the transaction succeeds or fails.

In the **transaction example**, you learned that the atomicity of SQL operations plays a crucial role in ensuring their reliability. This ensures that even if a transaction fails, the database won't be left in an uncertain or inconsistent state. It also facilitates error handling since developers can wrap multiple operations within a single transaction and be assured that all the changes will be made successfully, or none will be made at all. Data integrity is enhanced and prevented from being corrupted by unexpected interruptions by atomicity, which ensures the accuracy and reliability of SQL transactions.

TERMS TO KNOW

Atomicity
The quality of indivisibility. In SQL database terms, atomicity means the transaction cannot be partially executed: Either all statements are executed or none of them are.