Atomicity

by Sophia Tutorial

WHAT'S COVERED

This tutorial explores the atomicity property in a transaction and how it affects the database in two parts:

1. Atomicity in Transactions
2. Transaction Example

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 fails, 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.

2. Transaction 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. Jennifer's (10) account would be deducted by $100.
2. The banking location where Jennifer has her account would have their location's account deducted by $100.
3. The banking location 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 was 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 on the fourth UPDATE statement, an error would be returned. Regardless of what the error was, the first three SQL statements would revert back to what they were before the transaction started. Otherwise, Jennifer's account would be deducted by $100, the bank branch that holds Jennifer's account would have their balance deducted by $100, 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

The atomicity property ensures that either all SQL statements are executed or none of them are executed in a transaction.

Source: Authored by Vincent Tran