COMMIT and ROLLBACK to Manage Changes

by Sophia

WHAT'S COVERED

This lesson explores using COMMIT and ROLLBACK, in two parts. Specifically, this lesson will cover:

1. Introduction
2. Example Transaction

1. Introduction

Up to this point, we have looked at transactions as single units that start with a BEGIN and end with the **COMMIT** statement, with multiple SQL commands in between that are executed at once. Remember that without the BEGIN command, each SQL statement is viewed as a transaction with an implicit BEGIN and COMMIT statement executed.

ROLLBACK is an operation that cancels all the uncommitted changes prior to it in the transaction. It does not undo any already committed changes. It would typically be used in complex transactions involving multiple steps or operations.

REFLECT

Perhaps you are wondering, “Why would I need ROLLBACK, when a transaction is canceled automatically when an error occurs?” This is a good question.

ROLLBACK is useful because it helps you take more explicit control over a transaction's behavior:

- More precise control: Sometimes you might want to control logical flow through a transaction based on conditions other than errors. For example, you might want to roll back a transaction based on certain business logic criteria, which might not be strictly classified as errors from the database's perspective.

- Clarity: ROLLBACK enhances the code's readability and maintainability because it clearly communicates the developer's intentions.

- Nested transactions: Some database systems allow you to nest transactions. In such cases, an error in an inner transaction might not necessarily cause an automatic rollback of the outer transaction. Explicitly issuing a ROLLBACK statement allows you to manage the rollback behavior at different levels of nesting.

- Partial rollbacks: In complex transactions involving many steps or operations, you might want to roll back only part of the transaction while preserving the changes made by previous steps. Automatic rollback on error typically rolls back the entire transaction, but using ROLLBACK, you can control which parts of the transaction are undone.

- Debugging and testing: Explicitly issuing a ROLLBACK statement during development and testing phases enables developers to manually control the transaction's state and test various scenarios, ensuring that the application behaves as expected under different conditions.

To start a transaction in the PostgreSQL command line, you can start with either:

```
BEGIN;
```

or

```
BEGIN TRANSACTION;
```

The transaction then proceeds through its statements until it encounters either a COMMIT or ROLLBACK statement, or until it encounters an error (which triggers an automatic rollback).

The COMMIT statement is used to save changes from a transaction to the database. This COMMIT statement will save all the SQL statements to the database following the BEGIN command. The syntax for the COMMIT statement looks like this:

```
COMMIT;
```

or

```
END TRANSACTION;
```

The ROLLBACK statement is used to undo or revert SQL statements that have not already been saved to the database. The ROLLBACK statement can only be used to undo SQL statements that follow the BEGIN statement. The syntax looks like this:

```
ROLLBACK;
```

TERMS TO KNOW

COMMIT

A statement that makes tentative changes permanent, marking the end of a transaction and providing durability to ACID transactions.

ROLLBACK

A statement that returns the database to some previous state.

2. Example Transaction

Suppose you are writing SQL code for a banking database. Users can transfer funds between accounts, and you need to make sure that the funds are deducted from one account and added to another account as an atomic operation. In other words, if either of those operations fails, the whole transaction fails.

Let's assume there is an Accounts table with columns that include AccountID and Balance. Here's a transaction that can safely transfer $100 from account 123 to account 456:

```
BEGIN;
UPDATE Accounts
SET Balance = Balance - 100
WHERE AccountID = 123;
UPDATE Accounts
SET Balance = Balance + 100
WHERE AccountID = 456;
```

So far, so good, right? But to ensure that there are no errors, we could add this additional logical test, nested within the main transaction:

```
IF @ROWCOUNT = 2
BEGIN
     COMMIT;
     PRINT 'Transfer Successful';
END
ELSE
BEGIN
     ROLLBACK;
     PRINT 'Transfer failed. Transaction rolled back.';
END
```

In this case, a logical IF clause, and not the presence or absence of an error condition, determined whether the transaction would be committed or rolled back.

SUMMARY

In this lesson's **introduction**, you learned that the ROLLBACK statement can be run within a transaction to undo the transaction when certain conditions exist. The syntax of ROLLBACK, like that of COMMIT, is very simple, consisting only of the single word and a semi-colon (ROLLBACK;).

Then, in the **example transaction**, you saw a practical example of a situation where ROLLBACK would be useful. In this example, an IF clause is used to test for a certain condition, and then either ROLLBACK or COMMIT is implemented depending on the outcome of that test.

TERMS TO KNOW

COMMIT
A statement that makes makes tentative changes permanent, marking the end of a transaction and providing durability to ACID transactions.

ROLLBACK
An operation that returns the database to some previous state.