Transactions

by Sophia Tutorial

WHAT'S COVERED

This tutorial explores the concept of transactions in a database to ensure multiple statements execute in an all-or-nothing scenario, in two parts:

1. Introduction
2. Examples

1. Introduction

Transactions are a core feature of every database system. The purpose of a transaction is to combine multiple SQL statements together into a scenario that would either execute all of the statements or none of them. Each individual SQL statement within a transaction is not visible to other concurrent transactions in the database, as they are not saved to the database unless all of the statements in the transaction have executed successfully. If at any point there is a failure that occurs in any of the statements within a transaction, none of the steps or statements affect the database at all.

2. Examples

Let us take a look at a scenario in which a transaction would be necessary. James has gone to an online computer store and purchased a new computer for $500. In this transaction, there are two things to track. The first is the $500 being transferred from James to the store, and the second is the computer being deducted from the store inventory and transferred to James. The basic SQL statements would look something like the following:

```
UPDATE customer_account
SET balance = balance – 500
WHERE account_id = 1000;

UPDATE store_inventory
SET quantity = quantity – 1
WHERE store_id = 5 AND product_name = ‘Computer’;

INSERT INTO customer_order(account_id, product_name,store_id, quantity,cost)
```

```
VALUES (1000, 'Computer',5, 1, 500);

UPDATE store_account
SET balance = balance + 500
WHERE store_id = 5;
```

As you can see, there are multiple SQL statements that are needed to accomplish this operation. Both James and the store would want to be assured that either all of these statements occur, or none of them do. It would not be acceptable if James's account was deducted by $500, the inventory for the store had the computer removed, and then there was a system failure. This would mean that James would not get the order and the store would not get the $500 that James had paid. We need to ensure that if anything goes wrong at any point within the entire operation, that none of the statements that had been executed so far would take effect. This is where the use of a transaction is valuable.

To set up a transaction, we need to start with the BEGIN command, have our list of commands, and then end with COMMIT. Similar to our prior example:

```
BEGIN;

UPDATE customer_account
SET balance = balance – 500
WHERE account_id = 1000;

UPDATE store_inventory
SET quantity = quantity – 1
WHERE store_id = 5 AND product_name = 'Computer';

INSERT INTO customer_order(account_id, product_name,store_id, quantity,cost)
VALUES (1000, 'Computer',5, 1, 500);

UPDATE store_account
SET balance = balance + 500
WHERE store_id = 5;

COMMIT;
```

A transaction can contain a single statement or a dozen SQL statements. Note that each SQL statement needs to end with a semicolon to separate each out individually.

PostgreSQL and other databases treat each SQL statement that is executed as if it were an individual transaction. If we do not include a BEGIN command, then each of the SQL statements (INSERT, UPDATE, DELETE, etc) has an implicit BEGIN command and a COMMIT command if the statement is successful.

Video Transcription

[MUSIC PLAYING] Transactions allow you to execute multiple statements while ensuring that every single statement must be successful for the entire data set to be completed. By default, if you just run individual statements, like for example, if you just run this particular statement on its own, that piece would actually be implicitly having a begin statement and a commit statement at the end, and it would

just execute that particular statement.

However when you have a transaction block, within this case here, you'll start with a begin or begin transaction, and at the end, there will be a commit. This whole entire statement will only execute if all the different statements are successful. So for example, in this case here, what I'm going to do is I'm going to try to update the first name to Bob for the first name, and then I'm going to update the support rep to 20. However, the support rep it references the employee table, and in the employee table there is no employee ID that's equal to 20. So in this case here, the entire transaction should fail based on this particular update statement.

So if I try to run this statement, you'll notice that it identifies that there's an error. We can go ahead and take a look at the table again, and query it, and we'll see that the first name was not updated being that all those items were rolled back. Now, if we change this to the support rep id equals to 5, and we take a look at the data, we see that this is originally a 3, so it should be updated to 5 if it's possible. So we'll go ahead and execute this entire transaction. And you'll notice that it was completed successfully. Now if we tried to query the data, we should see that the changes for Bob, which is the customer ID equals 1, has the first name Bob, and the support ID is equal to 5 now.

[MUSIC PLAYING]

TRY IT

Your turn! Open the SQL tool by clicking on the LAUNCH DATABASE button below. Then enter in one of the examples above and see how it works. Next, try your own choices for which columns you want the query to provide.

SUMMARY

Transactions in a database are used to ensure multiple statements execute in an all-or-not scenario.

Source: Authored by Vincent Tran