Transactions

SQL

Transactions

Servers, database software, and the users that make use of them are not perfect. We cannot always expect that our application or database interactions will be carried out in full without interruption or failure. To ensure that important or complex tasks are carried out properly, we can introduce transactions.

Consider an online store where transfer of funds from a user to the store occurs. It would be catastrophic for customers' payments to be interrupted halfway through, run multiple times, or any other potential issue that will have harsh financial consequences.

Transactions are a way of grouping SQL statements into collection of instructions. When the transaction is run, it does not actually execute in full the instructions at first. Essentially, it runs the code in a sandbox to see if it is successful or not—only if it is successful and run to completion will the database commit the changes and have them implemented. Should the transaction fail, any change will be rolled back and dismissed, leaving the database unchanged by the situation.

How MySQL Handles Transactions

By default, each individual SQL statement run in MariaDB is treated as its own transaction and is committed once it has been run to completion. For this reason, it is of the utmost importance that you treat commands like UPDATE and DELETE with the respect they deserve, as once these sorts of statements are run, there is no rollback unless you have created a backup of your own.

We do have more control than this basic statement-by-statement transaction feature, as MariaDB affords us the "START TRANSACTION" command. After this statement, you would enter a number of SQL statements that you would like to have run as a transaction. If any of these commands fail they would, as we've said, be rolled back. After your transaction statements, you can end the transaction block with the "COMMIT" command.

Example Transaction

Let's look at a transaction written in SQL to see what the format looks like in action:

```
START TRANSACTION;

-- Create new order..
INSERT INTO Orders (`customer_id`, `order_date`)
  VALUES
    (2,  '2021-01-15' );

-- Ship the order.
INSERT INTO Shipments (`order_id`, `shipment_date`)
  VALUES
    (LAST_INSERT_ID(), '2021-02-20');

COMMIT;
```

If there are any issues within this transaction, the changes will be undone and discarded.

Effectively, the database will experience no change and the script is not executed and applied to the data.

If the inside of the transaction occurs successfully, the COMMIT command will run it against the database and save those changes.

Transactions Organized into Pieces

It is possible to break a transaction up into various smaller pieces if, or when, necessary. This allows smaller, more focused rollbacks depending on where you place a SAVEPOINT. This way, if something after a save point fails and there is a ROLLBACK command in place, the progress made *before* the SAVEPOINT will be committed and executed in full.

SAVEPOINTs are named, so that multiple can be used or rolled back to in your SQL script.

Example Save Point

Let's look at a transaction with a save point written in SQL to see what the format looks like in action:

```
START TRANSACTION;

-- Create new order..
INSERT INTO Orders (`customer_id`, `order_date`)
  VALUES
    (2,  '2021-01-15' );

SAVEPOINT order_created;

-- Ship the order.
INSERT INTO Shipments (`order_id`, `shipment_date`)
  VALUES
    (LAST_INSERT_ID(), '2021-02-20');

COMMIT;
```

The SAVEPOINT acts as a place the transaction will rollback to if there is an issue following it and within the transaction.

Looking at Rollback

How about use of ROLLBACK?

```
START TRANSACTION;

-- Create new order..
INSERT INTO Orders (`customer_id`, `order_date`)
  VALUES
    (2,  '2021-01-15' );

SAVEPOINT order_created;

-- Ship the order.
INSERT INTO Shipments (`order_id`, `shipment_date`)
  VALUES
    (LAST_INSERT_ID(), '2021-02-20');

ROLLBACK TO SAVEPOINT order_created;
```

In this example, note that once the code hits the ROLLBACK line, the shipment statement will be undone.

The new order will remain committed, as it was successful and did not experience a rollback.

Recommended Reading

It’s a good idea to read up more on what we’ve covered today. Have a look at the following:

- Beaulieu, A. (March 2020). Learning SQL, 3rd Edition. O’Reilly Media, Inc.
  - Chapter 12. Transactions