MySQL Transaction

**Summary**: Here, you will learn about **MySQL transaction**and how to use the COMMIT and ROLLBACK statements to manage transactions in MySQL.

## **Introducing to MySQL transactions**

To understand what a transaction in MySQL is, let’s take a look at an example of adding a new sales order in our sample database. The steps of adding a sales order are as described as follows:

* First, query the latest sales order number from the orders table and use the next sales order number as the new sales order number.
* Next, insert a new sales order into the orders table.
* Then, get the newly inserted sales order number
* After that, insert the new sales order items into the orderdetails table with the sales order number
* Finally, select data from both orders and orderdetails tables to confirm the changes

Now, imagine what would happen to the sales order data if one or more steps above fail due to some reasons such as table locking? For example, if the step of adding order’s items into orderdetails table fails, you will have an empty sales order.

That is why the transaction processing comes to the rescue. MySQL transaction allows you to execute a set of MySQL operations to ensure that the database never contains the result of partial operations. In a set of operations, if one of them fails, the rollback occurs to restore the database to its original state. If no error occurs, the entire set of statements is committed to the database.

## **MySQL transaction statements**

MySQL provides us with the following important statement to control transactions:

* To start a transaction, you use the START TRANSACTION  statement. The BEGIN or  BEGIN WORK are the aliases of the START TRANSACTION.
* To commit the current transaction and make its changes permanent,  you use the COMMIT statement.
* To roll back the current transaction and cancel its changes, you use the ROLLBACK statement.
* To disable or enable the auto-commit mode for the current transaction, you use the SET autocommit statement.

By default, MySQL automatically commits the changes permanently to the database. To force MySQL not to commit changes automatically, you use the following statement:

SET autocommit = 0;

Or

SET autocommit = OFF

You use the following statement to enable the autocommit mode explicitly:

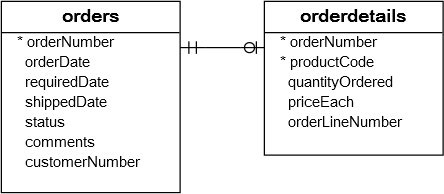
SET autocommit = 1;

Or

SET autocommit = ON;

## **MySQL transaction example**

We will use the  orders and orderDetails table from the sample database for the demonstration.



### **COMMIT example**

In order to use a transaction, you first have to break the SQL statements into logical portions and determine when data should be committed or rolled back.

The following illustrates the step of creating a new sales order:

* First, start a transaction by using the START TRANSACTION  statement.
* Next, select the latest sales order number from the orders table and use the next sales order number as the new sales order number.
* Then, insert a new sales order into the orders table.
* After that, insert sales order items into the orderdetails table.
* Finally, commit the transaction using the COMMIT statement.

Optionally, you can select data from both orders and orderdetails tables to check the new sales order.

The following is the script that performs the above steps:

-- 1. start a new transaction

START TRANSACTION;

-- 2. Get the latest order number

SELECT

@orderNumber:=MAX(orderNUmber)+1

FROM

orders;

-- 3. insert a new order for customer 145

INSERT INTO orders(orderNumber,

orderDate,

requiredDate,

shippedDate,

status,

customerNumber)

VALUES(@orderNumber,

'2005-05-31',

'2005-06-10',

'2005-06-11',

'In Process',

145);

-- 4. Insert order line items

INSERT INTO orderdetails(orderNumber,

productCode,

quantityOrdered,

priceEach,

orderLineNumber)

VALUES(@orderNumber,'S18\_1749', 30, '136', 1),

(@orderNumber,'S18\_2248', 50, '55.09', 2);

-- 5. commit changes

COMMIT;

MySQL Transaction Example

To get the newly created sales order, you use the following query:

SELECT

a.orderNumber,

orderDate,

requiredDate,

shippedDate,

status,

comments,

customerNumber,

orderLineNumber,

productCode,

quantityOrdered,

priceEach

FROM

orders a

INNER JOIN

orderdetails b USING (orderNumber)

WHERE

a.ordernumber = 10426;

Here is the output:

MySQL Transaction Example Order

### **ROLLBACK example**

First, log in to the MySQL database server and delete data from the orders table:

mysql> START TRANSACTION;

Query OK, 0 rows affected (0.00 sec)

mysql> DELETE FROM orders;

Query OK, 327 rows affected (0.03 sec)

As you can see from the output, MySQL confirmed that all the rows from the orders table were deleted.

Second, log in to the MySQL database server in a separate session and query data from the orders table:

mysql> SELECT COUNT(\*) FROM orders;

+----------+

| COUNT(\*) |

+----------+

| 327 |

+----------+

1 row in set (0.00 sec)

In this second session, we still can see the data from the orders table.

We have made the changes in the first session. However, the changes are not permanent. In the first session, we can either commit or roll back the changes.

For the demonstration purpose, we will roll back the changes in the first session.

mysql> ROLLBACK;

Query OK, 0 rows affected (0.04 sec)

in the first session, we will also verify the contents of the orders table:

SELECT COUNT(\*) FROM orders;

+----------+

| COUNT(\*) |

+----------+

| 327 |

+----------+

1 row in set (0.00 sec)

As you can see clearly from the output, the changes have been rolled back.

In this tutorial, you have learned how to use the MySQL transaction statements that include START TRANSACTION, COMMIT, and ROLLBACK to manage transactions.