Table of Contents

1. Transaction

To understand the commit and roll back statements, we need to understand a transaction. As a logical term, a transaction is a logical unit of work—a sequence of commands that should either all succeed or all fail. In terms of MySQL syntax, a transaction is a sequence of SQL operations that starts with a Begin Tran statement and that ends with a commit command or a rollback command.

Statements that start a Transaction		Statements that change data while a transaction is in effect		Statements that end a Transaction
Begin Start Transaction		Insert Update Delete Select- some versions		Commit Rollback Statement that change the schema

Changes that you make to a table via the Update, Insert, Delete statements are written to a buffer, not directly to the table in storage. You can make these changes permanent (i.e. written to the table) by issuing a commit or by working in autocommit mode.

2. Autocommit

Suppose you enter a row into a table and then decide that you do not want that row to be stored. Can you undo this. In a dbms system the undo is called a rollback.

MySQL has a system variable that is used to control your ability to simply undo action statements. That system variable is @@autocommit. It has two possible values:

- OFF or 0 means that we do not have an automatic commit of action statements.
- ON or 1 means that we do have an automatic commit of action statements.

If the system is working in an autocommit mode then each change is written back to the disk as soon as the change is made in the buffer. That also means that the system cannot do a rollback of those changes.

The default behavior of MySQL on my installation is to commit each DML statement when it is executed. Each of these statements forms its own transaction. To do multi-statement transactions, we need to change this. This is done by setting the value of the autocommit mode.

You can check the status of this setting

```
select @@autocommit;
+--------------+
| @@autocommit |
+--------------+
|            0 |
+--------------+
```

Change the value with

```
set autocommit =0;
set autocommit =1;
```

When you do this command that mode stays in effect until you explicitly turn it off or close your session.

Let's start with a very simple demo.

Set autocommit to 1. Create a small table. Insert several rows; do some updates and deletes. Then try to roll back the changes.

Set autocommit to 0. Create a small table. Insert several rows; do some updates and deletes. Then try to roll back the changes.

Demo 01: Using autocommit ON

```
use a_testbed;
drop table if exists z_tst_auto;
create table z_tst_auto ( col_1 int, col_2 varchar(15)) engine = Innodb;

set autocommit =1;
select @@autocommit;
+--------------+
| @@autocommit |
+--------------+
|            1 |
+--------------+

insert into z_tst_auto values
   (11, 'red'), (21, 'green'), (31, 'blue'),(41, 'yellow'),
   (51, 'black'), (61, 'ecru');

update z_tst_auto
   set col_2 = 'olive'
   where col_1 in ( 31,51);

delete from z_tst_auto
   where col_2 in ('green');

select * from z_tst_auto;
+-------+--------+
| col_1 | col_2  |
+-------+--------+
|    11 | red    |
|    31 | olive  |
|    41 | yellow |
|    51 | olive  |
|    61 | ecru   |
+-------+--------+
5 rows in set (0.00 sec)

ROLLBACK;
Query OK, 0 rows affected (0.00 sec)

select * from z_tst_auto;
+-------+--------+
| col_1 | col_2  |
+-------+--------+
|    11 | red    |
```

```
|    31 | olive  |
|    41 | yellow |
|    51 | olive  |
|    61 | ecru   |
+-------+--------+
5 rows in set (0.00 sec)
```

Demo 02: Using autocommit OFF

```
set autocommit =0;
select @@autocommit;
+--------------+
| @@autocommit |
+--------------+
|            0 |
+--------------+

insert into z_tst_auto values
  (10, 'tomato'), (20, 'forest'), (30, 'cyan'), (40, 'orange'),
  (50, 'darkgray'), (60, 'cornsilk');

update z_tst_auto
  set col_2 = 'aliceblue'
  where col_1 in ( 30, 31, 50, 51);

delete from z_tst_auto
  where col_2 in ('forest','yellow' );

select * from z_tst_auto;
+-------+-----------+
| col_1 | col_2     |
+-------+-----------+
|    11 | red       |
|    31 | aliceblue |
|    51 | aliceblue |
|    61 | ecru      |
|    10 | tomato    |
|    30 | aliceblue |
|    40 | orange    |
|    50 | aliceblue |
|    60 | cornsilk  |
+-------+-----------+

ROLLBACK;

select * from z_tst_auto;
+-------+--------+
| col_1 | col_2  |
+-------+--------+
|    11 | red    |
|    31 | olive  |
|    41 | yellow |
|    51 | olive  |
|    61 | ecru   |
+-------+--------+
```

When autocommit is set to ON then the changes were committed as each statement was executed and there was nothing in the buffer to roll back. The rollback command does not tell us anything about whether or not it actually rolled back any changes- the response is simply that the rollback command executed.

When autocommit is set to OFF then the changes were NOT committed as each statement was executed and the changes are stored in the buffer to roll back. Again, the rollback command does not tell us anything about whether or not it actually rolled back any changes- the response is simply that the rollback command executed. In this case the rollback undid all the changes since I changed the autocommit setting to off. The rollback applied to the inserts and the updates and the deletes.

Demo 03: Using commit. I am still in autocommit Off mode. This inserts some rows does a commit and then a rollback.

```
select @@autocommit;
+--------------+
| @@autocommit |
+--------------+
|            0 |
+--------------+

insert into z_tst_auto values
   (100, 'indigo'), (200, 'lawngreen'), (300, 'deepskyblue');

select * from z_tst_auto;
+-------+-------------+
| col_1 | col_2       |
+-------+-------------+
|    11 | red         |
|    31 | olive       |
|    41 | yellow      |
|    51 | olive       |
|    61 | ecru        |
|   100 | indigo      |
|   200 | lawngreen   |
|   300 | deepskyblue |
+-------+-------------+

COMMIT;
select * from z_tst_auto;
+-------+-------------+
| col_1 | col_2       |
+-------+-------------+
|    11 | red         |
|    31 | olive       |
|    41 | yellow      |
|    51 | olive       |
|    61 | ecru        |
|   100 | indigo      |
|   200 | lawngreen   |
|   300 | deepskyblue |
+-------+-------------+

ROLLBACK;
select * from z_tst_auto;
+-------+-------------+
| col_1 | col_2       |
+-------+-------------+
|    11 | red         |
```

```
|    31 | olive       |
|    41 | yellow      |
|    51 | olive       |
|    61 | ecru        |
|   100 | indigo      |
|   200 | lawngreen   |
|   300 | deepskyblue |
+-------+-------------+
```

Even though autocommit was set to off, I can do an explicit commit with the commit command. That makes the changes persistent and a Rollback does not undo them.

Demo 04: This time I insert some more rows and then create another table- this new table has nothing to do with my table z_tst_auto but it did cause a commit to be issued and the rollback cannot undo the changes to the table.

```
select @@autocommit;


insert into z_tst_auto values
(400, 'LightCoral'), (500, 'DodgerBlue'), (600, 'Lavender');
select * from z_tst_auto;
+-------+-------------+
| col_1 | col_2       |
+-------+-------------+
|    11 | red         |
|    31 | olive       |
|    41 | yellow      |
|    51 | olive       |
|    61 | ecru        |
|   100 | indigo      |
|   200 | lawngreen   |
|   300 | deepskyblue |
|   400 | LightCoral  |
|   500 | DodgerBlue  |
|   600 | Lavender    |
+-------+-------------+

create table z_tst_1 ( col1 int) engine = Innodb;

ROLLBACK;

select * from z_tst_auto;
+-------+-------------+
| col_1 | col_2       |
+-------+-------------+
|    11 | red         |
|    31 | olive       |
|    41 | yellow      |
|    51 | olive       |
|    61 | ecru        |
|   100 | indigo      |
|   200 | lawngreen   |
|   300 | deepskyblue |
|   400 | LightCoral  |
|   500 | DodgerBlue  |
|   600 | Lavender    |
+-------+-------------+
```

Demo 05: I am setting the autocommit back to ON.

```
set autocommit =1;
```

2.1. Explicitly defining a transaction

Even when the autocommit is set to ON we can define a transactions and - in a sense- protect it from the automatic commit. We do this by starting an explicit transaction with the BEGIN command. Within the transaction we can do action queries and then decide to issue either a ROLLBACK or a COMMIT command to end the transaction. Normally we would have some program logic that decides to rollback or commit. Since that involved procedural code (not the focus of this class) I am simply going to do a few demos where I issue either the commit or rollback.

Demo 06:

```
drop table if exists z_tst_trans;

create table z_tst_trans ( col1 int, col2 varchar(15)) engine = Innodb;
```

Demo 07: Start a transaction, do inserts; rollback

```
select @@autocommit;
+--------------+
| @@autocommit |
+--------------+
|            1 |
+--------------+
BEGIN;
insert into z_tst_trans values ( 1, 'hare'),( 2, 'dragon');

select * from z_tst_trans;
+------+--------+
| col1 | col2   |
+------+--------+
|    1 | hare   |
|    2 | dragon |
+------+--------+
ROLLBACK;

select * from z_tst_trans;
Empty set (0.00 sec)
```

What happened is that we started a transaction and did two inserts while we were in the transaction. These rows can be displayed. Then we issued a rollback command- this did two things (1) it undid all of the actions in the transaction - in this case the two inserts and then (2) ended the transaction. When we try to display the data in the table we do not have any rows.

Demo 08: Start a transaction, do inserts; commit

```
begin ;
insert into z_tst_trans values ( 3, 'snake'), ( 4, 'horse');

select * from z_tst_trans;
```

```
+------+-------+
| col1 | col2  |
+------+-------+
|    3 | snake |
|    4 | horse |
+------+-------+
-- Do a commit - this makes the inserts persistent and ends the transaction
commit;

-- Display the data in the table.
select * from z_tst_trans;
+------+-------+
| col1 | col2  |
+------+-------+
|    3 | snake |
|    4 | horse |
+------+-------+
```

Demo 09: Start a transaction, do inserts; do a schema object change

```
begin ;
insert into z_tst_trans values ( 5, 'sheep'),( 6, 'monkey');

select * from z_tst_trans;
+------+--------+
| col1 | col2   |
+------+--------+
|    3 | snake  |
|    4 | horse  |
|    5 | sheep  |
|    6 | monkey |
+------+--------+
create table z_tst_2( col1 int);

-- Display the data in the table.
select * from z_tst_trans;
+------+--------+
| col1 | col2   |
+------+--------+
|    3 | snake  |
|    4 | horse  |
|    5 | sheep  |
|    6 | monkey |
+------+--------+
ROLLBACK;
select * from z_tst_trans;
+------+--------+
| col1 | col2   |
+------+--------+
|    3 | snake  |
|    4 | horse  |
|    5 | sheep  |
|    6 | monkey |
+------+--------+
```

2.2. Begin

You start a transaction with the BEGIN command.

2.3. Commit

A commit occurs when one of the following occurs:

- The user issues a COMMIT command- this is an explicit commit
- The user changes a schema level object- this does an implicit commit.

2.4. Rollback

The user can issue the ROLLBACK command to undo all changes in the buffer, all changes since the last commit.

Transaction are supported by the InnoDB engine but not by the MyISAM engine. The discussion of transactions get more complex when stored routines are involved and also with multi-user systems. You can experiment with some of the multi-user aspects by opening two mysql sessions. This is discussed in the last section of chapter 37 but is not required for this class.