ROLLBACK Statement

The ROLLBACK operation undoes all the changes done by the current transaction

If you invoke this statement, all the modifications are reverted until the last commit or the START TRANSACTION statement.

```
SET autocommit=0;
insert values into table;
COMMIT;
SELECT * FROM emp;
ROLLBACK;
SELECT * FROM emp;
```

A save point is a logical rollback point within a transaction

The SAVEPOINT statement is used to set a save point for the transaction with the specified name.

Grant Privilege

The grant statement enables system administrators to assign privileges and roles to the MySQL user accounts so that they can use the assigned permission on the database whenever required.

```
GRANT privilege_name(s)
ON object
TO user_account_name;
```

```
GRANT ALL ON mystudentdb.* TO xyz@localhost;
```

Revoke Privilege

The revoke statement enables system administrators to revoke privileges and roles to the MySQL user accounts so that they cannot use the assigned permission on the database in the past.

```
REVOKE privilege_name(s)
ON object
FROM user_account_name;
```

```
GRANT ALL ON mystudentdb.* TO xyz@localhost;
```

Stored Procedure

A stored procedure is a prepared SQL code that you can save,so the code can be reused over and over again.

procedure always contains a name, parameter lists, and SQL statements. We can invoke the procedures by using triggers, other procedures and applications such as Java, Python, PHP, etc.

Stored Procedure Features

- Stored Procedure increases the performance of the applications. Once stored procedures are created, they are compiled and stored in the database.
- Stored procedure reduces the traffic between application and database server. Because the application has to send only the stored procedure's name and parameters instead of sending multiple SQL statements.
- Stored procedures are reusable and transparent to any applications.
- A procedure is always secure. The database administrator can grant permissions to applications that access stored procedures in the database without giving any permissions on the database tables.

Creating Stored Procedure

```
DELIMITER &&
CREATE PROCEDURE procedure_name [[IN | OUT | INOUT] parameter_name datatype [, parameter datatype]) ]
BEGIN
   Declaration_section
   Executable_section
END &&
DELIMITER ;
```

Parameter Name	Descriptions
procedure_name	It represents the name of the stored procedure.
parameter	It represents the number of parameters. It can be one or more than one.
Declaration_section	It represents the declarations of all variables.
Executable_section	It represents the code for the function execution.

MySQL procedure parameter has one of three modes:

IN parameter

It is the default mode. It takes a parameter as input, such as an attribute. When we define it, the calling program has to pass an argument to the stored procedure. This parameter's value is always protected.

```
DELIMITER ##
CREATE PROCEDURE emp1(IN empno INT)
BEGIN
SELECT *FROM emp where empno=empno;
END##

CALL emp1(102)
```

OUT parameters

It is used to pass a parameter as output. Its value can be changed inside the stored procedure, and the changed (new) value is passed back to the calling program. It is noted that a procedure cannot access the OUT parameter's initial value when it starts.

```
DELIMITER ##
CREATE PROCEDURE emp3(OUT empno INT)
BEGIN
SELECT id into empno FROM emp;
END##

CALL emp3(@o)
SELECT @o
```

INOUT parameters

It is a combination of IN and OUT parameters. It means the calling program can pass the argument, and the procedure can modify the INOUT parameter, and then passes the new value back to the calling program.

```
DELIMITER ##
CREATE PROCEDURE emp4(INOUT empno INT)
BEGIN
SELECT id into empno FROM emp where id=empno;
END##

SET @o=11
CALL emp4(@o)
SELECT @o
```

Triger

It is a special type of stored procedure that is invoked automatically in response to an event.Each trigger is associated with a table, which is activated on any DML statement such as INSERT, UPDATE, or DELETE.

triggers are of two types according to the SQL standard: **row-level triggers** and **statement-level triggers**.

Row-Level Trigger: It is a trigger, which is activated for each row by a triggering statement such as insert, update, or delete. For example, if a table has inserted, updated, or deleted multiple rows, the row trigger is fired automatically for each row affected by the insert, update, or delete statement.

Statement-Level Trigger: It is a trigger, which is fired once for each event that occurs on a table regardless of how many rows are inserted, updated, or deleted.

use triggers in MySQL?

- Triggers help us to enforce business rules.
- Triggers help us to validate data even before they are inserted or updated.
- Triggers help us to keep a log of records like maintaining audit trails in tables.
- SQL triggers provide an alternative way to check the integrity of data.
- Triggers provide an alternative way to run the scheduled task.
- Triggers increases the performance of SQL queries because it does not need to compile each time the query is executed.
- Triggers reduce the client-side code that saves time and effort.
- Triggers help us to scale our application across different platforms.
- Triggers are easy to maintain.
- Limitations of Using Triggers in MySQL
- MySQL triggers do not allow to use of all validations; they only provide extended validations. For example, we can use the NOT NULL, UNIQUE, CHECK and FOREIGN KEY constraints for simple validations.
- Triggers are invoked and executed invisibly from the client application. Therefore, it isn't easy to troubleshoot what happens in the database layer.
- Triggers may increase the overhead of the database server.

Types of Triggers in MySQL?

- Before Insert: It is activated before the insertion of data into the table.
- After Insert: It is activated after the insertion of data into the table.
- Before Update: It is activated before the update of data in the table.
- After Update: It is activated after the update of the data in the table.
- Before Delete: It is activated before the data is removed from the table.
- After Delete: It is activated after the deletion of data from the table.

```
CREATE
    [DEFINER = user]
    TRIGGER trigger_name
    trigger_time trigger_event
    ON tbl_name FOR EACH ROW
    [trigger_order]
    trigger_body

trigger_time: { BEFORE | AFTER }

trigger_event: { INSERT | UPDATE | DELETE }

trigger_order: { FOLLOWS | PRECEDES } other_trigger_name
```

Table

```
DROP TABLE IF EXISTS `characters`;
CREATE TABLE `characters` (
  `character_id` int(10) UNSIGNED NOT NULL,
  `character_name` varchar(50) NOT NULL,
  `race_id` tinyint(3) UNSIGNED NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

 Dumping data for table `characters`

INSERT INTO `characters` (`character_id`, `character_name`, `race_id`) VALUES
(1, 'Aragorn', 12),
(2, 'Bilbo', 3),
(3, 'Gimli', 1),
(4, 'Legolas', 4);
```

creating a triger

```
create triger trname
before insert on characters
for each row
set new.character_name=upper(new.character_name)
```

old

```
create triger trname
before update on characters
for each row
set new.character_name=lower(new.characters)
```