Birla Institute of Technology & Science Pilani, K.K Birla Goa Campus
CS-F212 Database Systems
Sem2 2020-2021
Lab 5 Reading Material - Constraints, Authorization, Triggers and Assertions
- Vivek Arora, Arshika Lalan, Arjun Bajpai

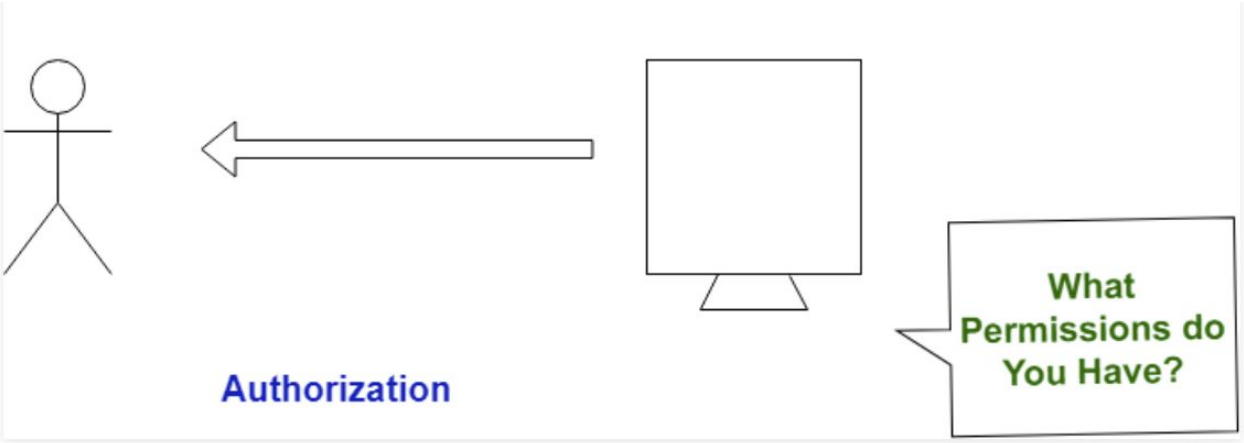
Integrity Constraints

- **Domain integrity constraints:**
  These constraints set a range, and any violations that take place will prevent the user from performing the manipulations that caused the breach.
  Two types are:
  1. `NOT NULL:` By default, the table can contain null values. This constraint ensures that the table has a value.
  2. `CHECK`: This can be defined to allow only a particular range of values.

- **Entity integrity constraints:**
  There are two types of entity integrity constraints.
  1. `UNIQUE`: The unique constraint designates a column or a group of columns as a unique key. This constraint allows only unique values to be stored in a column and thus it rejects duplication of records.
  2. `PRIMARY KEY`: This constraint is the same as a unique constraint,, but in addition to preventing duplication it also does not allow null values. This constraint cannot be put on the column having ‘long’ data type.

- **Referential integrity constraints:**
  It enforces the relationship between tables. It designates a column or combination of columns as a foreign key. The foreign key establishes a relationship with a specified primary or unique key in another table called the referenced key. In this relationship, the table containing the foreign key is called the child table, and the table containing the referenced key is called the parent table.

Authorization



Authorization is the allowance to the user or process to access the set of objects.

This control of access is done mostly using - Privileges.

Privileges can allow permitting a particular user to connect to the database. In other words, privileges are the allowance to the database by the database object.

- **Database privileges —**

  A privilege is permission to execute one particular type of SQL statement or access a second persons' object. Database privilege controls the use of computing resources. Database privilege does not apply to the Database administrator of the database.

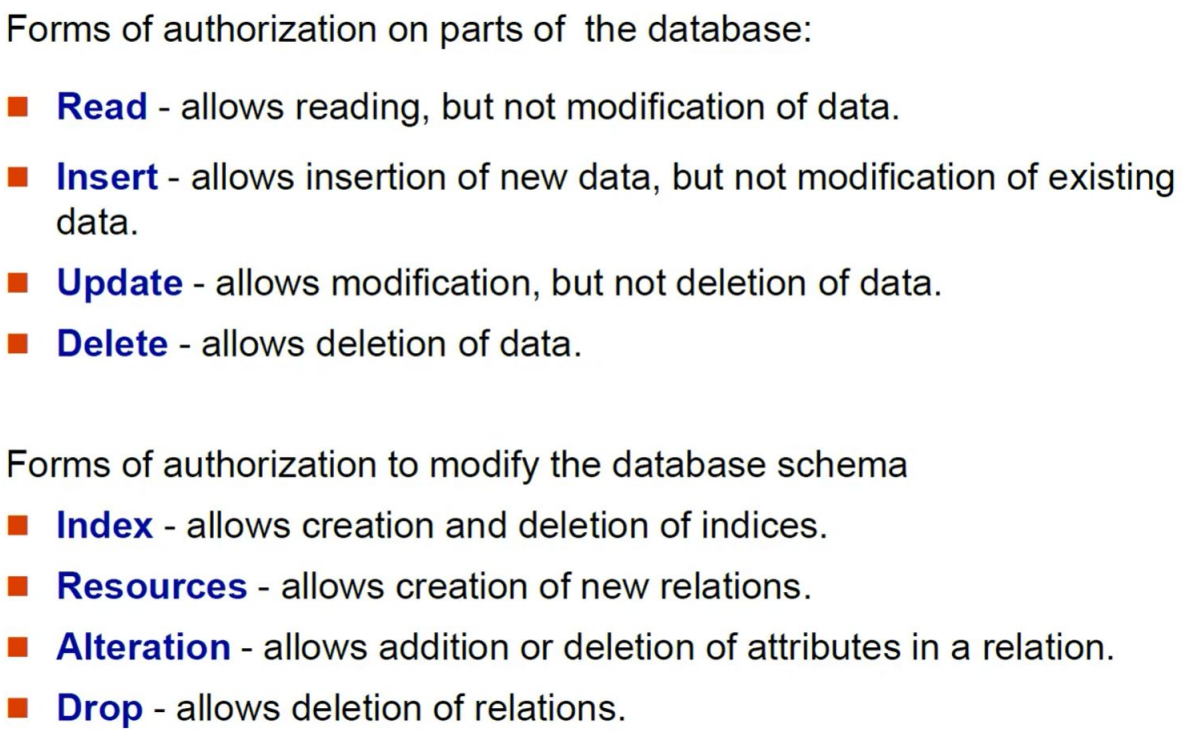
- **System privileges —**

  A system privilege is the right to perform an activity on a specific type of object. for example, the privilege to delete rows of any table in a database is system privilege. There are a total of 60 different system privileges. System privileges allow users to CREATE, ALTER, or DROP the database objects.

- **Object privilege —**

  An object privilege is a privilege to perform a specific action on a particular table, function, or package. For example, the right to delete rows from a table is an object privilege.. Object privilege allows the user to INSERT, DELETE, UPDATE, or SELECT the data in the database object.

This image should show what kind of access can be granted by the administrator.



Till this point we were using only one user for out commands, let us now see how we can implement this authorization in MySQL.

Before we start ensure you are logged into MySQL as administrator

```
sudo mysql -u root (-p if password protected)
```

To list out all the users on the database:

```
SELECT user FROM mysql.user
```

To identify the current user:

```
SELECT user();
```

Lets see how to create a new user now:

```
CREATE USER 'yourUserName'@'localhost' IDENTIFIED BY
'newUserPassword';
```

To Delete a user, we use this command

```
DROP USER 'username'@'host';
```

Sometimes you could face issues with removing a user because of inconsistency on the server and system, in those scenarios the administrator can manually remove the user from the list of users.

```
DELETE from mysql.user where user='yourUserName' and
host='localhost';
```

Now try logging in with this user into MySQL

```
sudo mysql -u yourUserName (-p if password protected)
```

Depending on how you set up MySQL as administrator you will see if you are allowed to login or not. Since I did not setup administrator to have any security the same applied for users and they could login.

To grant privileges we use the command “grant”

```
grant <privilege> on <relation> to <user> ;
```

So for the above example let's grant all privileges on everything we would type:

```
GRANT ALL PRIVILEGES ON * . * TO 'yourUserName'@'localhost';
```

These are the privileges that can be granted:

- insert
- update
- delete
- all

The privileges have to be listed out in the command mentioned above.

The <user> could also actually be a list of users and also these could be mentioned in place of it.

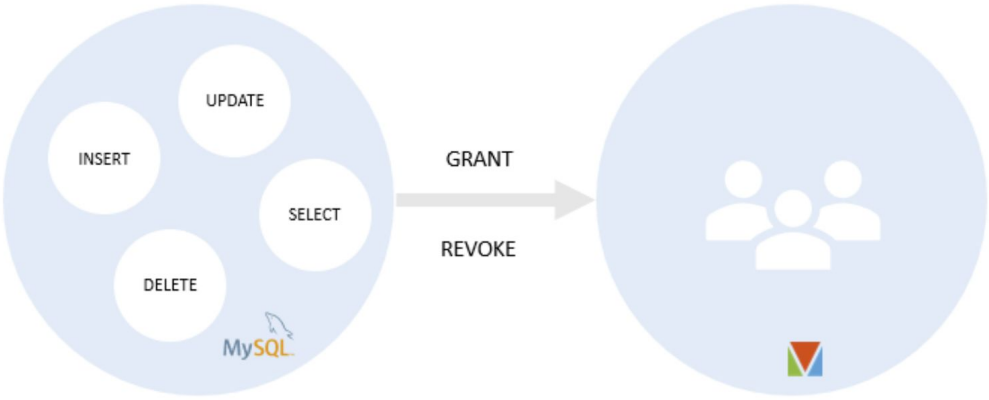
- User-id
- public
- Roles

Similarly we can revoke privileges the same way using “revoke”:

```
revoke <privilege> on <relation> to <user> ;
```

We will discuss Privilege Tree in the lab to understand how revoking has to be done carefully.

Roles



Roles are used to grant the same set of privileges to multiple users. This is different from listing out user-id because this creates groups of users with different privileges.

The process of granting privileges using roles is:

1. create a new role
2. grant privileges to the role
3. grant the role to the users

To create a role we use the command:

```
CREATE ROLE <rolename>;
```

To grant a role to users we use the command:

```
grant <rolename> to <yourUserName>;
```

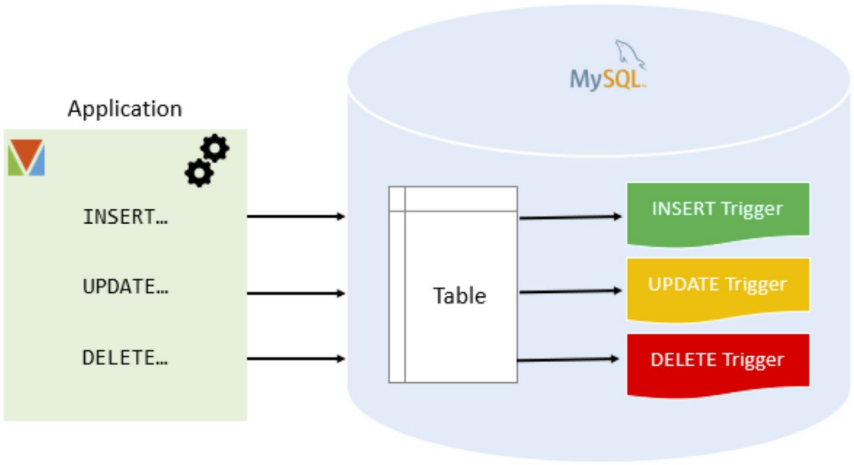
To verify the role assignments, you use the `SHOW GRANTS` statement:

```
SHOW GRANTS FOR <yourUserName>;
```

To specify which roles should be active each time a user account connects to the database server, you use the `SET DEFAULT ROLE` statement.

```
SET DEFAULT ROLE ALL TO <yourUserName>;
```

Triggers



A MySQL trigger is a stored program (with queries) which is executed automatically to respond to a specific event such as insertion, updation or deletion occurring in a table.

MySQL supports triggers that are invoked in response to the `INSERT`, `UPDATE` or `DELETE` event.

MySQL supports only row-level triggers. It doesn't support statement-level triggers.

In MySQL, a trigger is a set of SQL statements that is invoked automatically when a change is made to the data on the associated table. You can to define maximum six triggers for each table.

- BEFORE INSERT – activated before data is inserted into the table.
- AFTER INSERT – activated after data is inserted into the table.
- BEFORE UPDATE – activated before data in the table is updated.
- AFTER UPDATE – activated after data in the table is updated.
- BEFORE DELETE – activated before data is removed from the table.
- AFTER DELETE – activated after data is removed from the table

When you use a statement that does not use `INSERT`, `DELETE` or `UPDATE` statement to change data in a table, the triggers associated with the table are not invoked. For example, the `TRUNCATE` statement removes all data of a table but does not invoke the trigger associated with that table.

There are some statements that use the `INSERT` statement behind the scenes such as the `REPLACE` statement or `LOAD DATA` statement. If you use these statements, the corresponding triggers associated with the table are invoked.

You must use a unique name for each trigger associated with a table.
However, you can have the same trigger name defined for different tables though it is a good practice.

You should name the triggers using the following naming convention:

```
(BEFORE/ AFTER)_TABLENAME_(INSERT/UPDATE/DELETE)
```

For example, `before_order_update` is a trigger invoked before a row in the `ordertable` is updated.

The following naming convention is as good as the one above.

```
TABLENAME_(BEFORE/ AFTER)_ (INSERT/UPDATE/DELETE)
```

For example, `order_before_update` is the same as `before_update_update` trigger above.

MySQL Syntax
The following illustrates the syntax of the CREATE TRIGGER statement:

```
CREATE TRIGGER trigger_name trigger_time trigger_event
 ON table_name
 FOR EACH ROW
 BEGIN
 ...
 END;
```

Let’s examine the syntax above in more detail.

- You put the trigger name after the CREATE TRIGGER statement. The trigger name should follow the naming convention [trigger time]_[table name]_[trigger event], for example before_employees_update.
- Trigger activation time can be BEFORE or AFTER. You must specify the activation time when you define a trigger. You use the BEFORE keyword if you want to process action prior to the change is made on the table and AFTER if you need to process action after the change is made.
- The trigger event can be INSERT, UPDATE or DELETE. This event causes the trigger to be invoked. A trigger only can be invoked by one event. To define a trigger that is invoked by multiple events, you have to define multiple triggers, one for each event.
- A trigger must be associated with a specific table. Without a table trigger would not exist therefore you have to specify the table name after the ON keyword.
- You place the SQL statements between BEGIN and END block. This is where you define the logic for the trigger.

Example:

First, create a new table named `employees_audit` to keep the changes of the `employee` table. The following statement creates the `employee_audit` table.

```
CREATE TABLE employees_audit (
    id INT AUTO_INCREMENT PRIMARY KEY,
    employeeNumber INT NOT NULL,
    lastname VARCHAR(50) NOT NULL,
    changedat DATETIME DEFAULT NULL,
    action VARCHAR(50) DEFAULT NULL
);
```

Next, create a `BEFORE UPDATE` trigger that is invoked before a change is made to the `employees` table

```
DELIMITER $$
CREATE TRIGGER before_employee_update
    BEFORE UPDATE ON employees
    FOR EACH ROW
BEGIN
    INSERT INTO employees_audit
    SET action = 'update',
     employeeNumber = OLD.employeeNumber,
        lastname = OLD.lastname,
        changedat = NOW();
END$$
DELIMITER ;
```

Inside the body of the trigger, we used the `OLD` keyword to access `employeeNumber` and `lastname`column of the row affected by the trigger. Notice that in a trigger defined for INSERT, you can use `NEW` keyword only. You cannot use the `OLD`keyword. However, in the trigger defined for DELETE, there is no new row so you can use the `OLD`keyword only. In the UPDATE trigger, `OLD` refers to the row before it is updated and `NEW` refers to the row after it is updated.

Then, to view all triggers in the current database, you use `SHOW TRIGGERS` statement as follows:

```
SHOW TRIGGERS;
```

	Trigger	Event	Table	Statement	Timing	Created	sql_mode	Definer
▶	before_employee_update	UPDATE	employees	BEGIN INSERT INTO employ...	BEFORE	2015-11-14 21:39:09.08	STRICT_TRANS_TABLES,NO_AUTO_CREATE_U...	root@localhost

After that, update the `employees` table to check whether the trigger is invoked.

```
UPDATE employees
SET
    lastName = 'Phan'
WHERE
    employeeNumber = 1056;
```

Finally, to check if the trigger was invoked by the `UPDATE` statement, you can query the `employees_audit` table using the following query:

```
SELECT
    *
FROM
    employees_audit;
```

The following is the output of the query:

	id	employeeNumber	lastname	changedat	action
▶	1	1056	Phan	2015-11-14 21:39:12	update

As you see, the trigger was really invoked and it inserted a new row into the `employees_audit` table.

Assertions

When a constraint involves 2 (or) more tables, the table constraint mechanism is sometimes hard and results may not come as expected.

To cover such situation SQL supports the creation of assertions which are constraints not associated with only one table.

And an assertion statement should ensure a certain condition will always exist in the database.

DBMS always checks the assertion whenever modifications are done in the corresponding table.

Assertions are used when we know that the given particular condition is always true.

When the SQL condition is not met then there are chances to an entire table or even Database to get locked up.

Assertions are not linked to specific table or event. It performs task specified or defined by the user.

Syntax

```
CREATE ASSERTION  [ assertion_name ]
CHECK ( [ condition ] );
```

Example

```
CREATE TABLE sailors (sid int,sname varchar(20), rating int,primary
key(sid),
CHECK(rating >= 1 AND rating <=10)
CHECK((select count(s.sid) from sailors s) + (select count(b.bid)from
boats b)<100) );
```