

# Lab – 14

# Transactions and Triggers

October 28, 2021

# Transactions

# Transactions (1)

- A sequential group of database manipulation operations, performed as a one single work unit.
- A transaction will never be complete unless each individual operation within the group is successful.
- If any operation within the transaction fails, the entire transaction will fail.
- Example: Banking transaction, a transfer of 100.
  - Check that the balance of 1<sup>st</sup> account is  $> 100$ .
  - Deduct 100 from the 1<sup>st</sup> account.
  - Add 100 to the 2<sup>nd</sup> account.

# Transactions (2)

- In SQL, we have a session variable AUTOCOMMIT.
- If AUTOCOMMIT is set to 1 (default in MySQL):
  - Each SQL statement is seen as a complete transaction.
  - Each transaction is automatically made permanent.
- If AUTOCOMMIT is set to 0:
  - Subsequent series of statements acts like a transaction.
  - Activities are committed by an explicit COMMIT statement.
- In MySQL, transactions begin with the statement BEGIN and end with a COMMIT or ROLLBACK statement.

# Transactions (3)

AUTO COMMIT = 1;

BEGIN; —————→ AUTO COMMIT = 0;

semicolon separated bulk of SQL commands

COMMIT OR ROLLBACK; —→ AUTO COMMIT = 1;

# Properties of Transactions

- **ACID** - 4 standard properties are:
  - **Atomicity** - All operations are completed successfully.
  - **Consistency** - Database properly changes the states.
  - **Isolation** - Transactions should operate independently and transparently to each other.
  - **Durability** - The result of a committed transaction persists.
- For successful transaction issue COMMIT command.
- If a failure occurs, a ROLLBACK command should be issued.

**CONNECTION 1 : Green**

**CONNECTION 2 : Blue**

BEGIN;

INSERT INTO t (f) VALUES (1);

SELECT \* FROM t;

An Empty Set

COMMIT;

SELECT \* FROM t;

Updated Set

**CONNECTION 1 : Green**

**CONNECTION 2 : Blue**

BEGIN;

INSERT INTO t (f) VALUES (1);

SELECT \* FROM t;

An Empty Set

ROLLBACK;

SELECT \* FROM t;

Empty Set



# Another Transaction Example

- Check the account balance for customer and terminate the account if its balance is less than 500. If the account is older than year 2010 the transaction will “COMMIT” else transaction will “ROLLBACK”

# Revisit Your Database For Answer

## Depositor Table

Customer_name	Account_number
Priya	102
Yash	101
Yash	201
Vinay	217
Anjali	222
Divya	217
Rohit	305

## Account Table

Account_Number	Branch_Name	Balance	Date
101	Wright Town	500	5-2-11
215	Mehgawan	700	3-7-12
102	S street	400	6-8-10
305	Napier town	350	4-6-09
201	Stadium	900	9-4-10
222	Cross square	700	8-11-11
217	Stadium	750	2-10-12

# Another Transaction Example – Contn..

- The code will check the account balance for customer and terminate the account if its balance is less than 500. If the account is older than year 2010 the transaction will “COMMIT” else transaction will “ROLLBACK”
- Run **front.php**
- See Code for Transaction in **transaction.php**
- Run code for customers :
  - Priya - **ROLLBACK**
  - Rohit - **COMMIT**

# Accessing Database with Various Privilege Levels

# Create a Dummy Database

```
CREATE DATABASE trial;
```

```
CREATE TABLE dept(  
    did INT NOT NULL ,  
    dname VARCHAR(15),  
    PRIMARY KEY (did));
```

Populate these tables  
with some data

```
CREATE TABLE emp(  
    eid INT NOT NULL ,  
    ename VARCHAR(15) ,  
    PRIMARY KEY (eid));
```

# Assigning Privileges

**Granting** access to users on your database:

- You should have appropriate privileges to assign rights on your own database.
- A trial database has been created with super-user's privilege.
- Consider following 5 users with widening scope of their rights on 'trial' database:

USER	Privilege
user1	NO Privilege
user2	SELECT
user3	SELECT, INSERT
user4	SELECT, INSERT, UPDATE
user5	SELECT, INSERT, UPDATE, DELETE

# Assigning Privileges by Queries (1)

- The syntax to create any user in MySQL –
  - CREATE USER *user\_specification* [, *user\_specification*] ...

*user\_specification*: user [IDENTIFIED BY [PASSWORD] 'password']

- Ex – Suppose we want to create 'user1' with password 'password' -

```
CREATE USER 'user1'@'localhost' IDENTIFIED BY 'password';
```

**Note** - To create user who can connect remotely to MySQL use '%' instead.

# Assigning Privileges by Queries (2)

- The syntax to GRANT access to user in MySQL –
  - GRANT privilege\_type ON  
{ table\_Name | Database\_Name.\* } TO USER
- Ex – Suppose we want 'user3' to give only SELECT,INSERT access to all the tables in database 'trial'

```
GRANT SELECT,INSERT ON trial.* TO  
'user3'@'localhost';
```



# Assigning Privileges through Interface



The screenshot displays a database management interface with a top navigation bar containing the following tabs: Databases, SQL, Status, Variables, Charsets, Engines, Privileges (selected), Processes, and Export. Below the navigation bar is an 'Import' button. The main content area is titled 'User overview' and features a horizontal list of letters from A to Z, with a '[Show all]' link at the end. Below this list is a table with the following columns: User, Host, Password, Global privileges, Grant, and an edit icon. The table contains five rows of user data. The second row, for 'user2', is highlighted in green. At the bottom left of the table, there is a checkbox and the text 'Check All / Uncheck All'.

	User	Host	Password	Global privileges	Grant	
<input type="checkbox"/>	user1	%	Yes	USAGE	No	
<input type="checkbox"/>	user2	%	Yes	USAGE	No	
<input type="checkbox"/>	user3	%	Yes	USAGE	No	
<input type="checkbox"/>	user4	%	Yes	USAGE	No	
<input type="checkbox"/>	user5	%	Yes	USAGE	No	

☐ Check All / Uncheck All

# Assigning Privileges through Interface

Database-specific privileges

Database	Privileges	Grant	Table-specific privileges	Action
None				
Add privileges on the following database:				
Use text field:		<input type="text"/>		
2010246				
shusain				
students				
test				
trial				
trial1				
wikidb				

Go

# Assigning Privileges through Interface

 User '**user2**'@%' - Database **trial** : Edit Privileges

Database-specific privileges ([Check All](#) / [Uncheck All](#))

*Note: MySQL privilege names are expressed in English*

## Data

- ☒ SELECT
- ☐ INSERT
- ☐ UPDATE
- ☐ DELETE

## Structure

- ☐ CREATE
- ☐ ALTER
- ☐ INDEX
- ☐ DROP
- ☐ CREATE TEMPORARY TABLES
- ☐ CREATE VIEW
- ☐ SHOW VIEW
- ☐ CREATE ROUTINE
- ☐ ALTER ROUTINE
- ☐ EXECUTE

## Administration

- ☐ GRANT
- ☐ LOCK TABLES
- ☐ REFERENCES

Go

# Assigning Privileges through Interface





You have updated the privileges for 'user2'@'%'.

SQL query:

```
GRANT SELECT ON `trial` . * TO 'user2'@'%';
```

## Database-specific privileges

Database	Privileges	Grant	Table-specific privileges	Action
----------	------------	-------	---------------------------	--------

trial	SELECT	No	No	 
-------	--------	----	----	---

Add privileges on the following database:

Use text field:



Go

# Triggers

# TRIGGERS (1)

- It is a named database object that is associated with a table.
- It gets activated when a particular event occurs for the table.
- It can be associated with a permanent table only and **not with a TEMPORARY table or a view.**
- It was added in MySQL 5.0.2.
- In MySQL 5.0 it requires the SUPER privilege.
- Triggers for a table are also dropped if the table is dropped.

# TRIGGERS (2)

- Syntax

DELIMITER \$\$

A number of SQL commands separated by a semi-colon (;) are required to create the full trigger code, therefore delimiter must be changed to something else - such as \$\$.

CREATE TRIGGER *trigger\_name*

*trigger\_time*

*trigger\_event* ON *tbl\_name* FOR EACH ROW

*trigger\_body*

Set the delimiter back to a semi-colon

DELIMITER;

# TRIGGERS (3)

***trigger\_name*** Name of the trigger.

***e***

***trigger\_time*** Trigger action time. It can be BEFORE or AFTER to indicate that the trigger activates before or after each row to be modified.

***trigger\_event*** Indicates the kind of statement that activates the trigger. It can be INSERT, UPDATE or DELETE to indicate that the trigger activates on inserting, updating or deleting a row.

***tbl\_name*** Table to which a trigger is associated.

***trigger\_body*** Statements to be executed when the trigger activates. To execute multiple statements, use the BEGIN ... END compound statement construct.

Note: There cannot be two triggers for a given table that have the same trigger action time and event



# TRIGGERS (4)

Example 1: Let we have a table 'customer\_time' to record which customer is inserted at what moment.

```
CREATE TABLE customer_time(customer_name  
VARCHAR(15) NOT NULL    PRIMARY KEY, TIME  
TIMESTAMP NOT NULL);
```

# TRIGGERS (5)

Example 1 ... contd ...

```
DELIMITER $$
CREATE TRIGGER insert_customer1
AFTER INSERT ON customer
FOR EACH ROW
BEGIN
    INSERT INTO customer_time
        (customer_name , TIME)
    VALUES (NEW.customer_name, CURRENT_TIMESTAMP());
END$$
DELIMITER ;
```

# TRIGGERS (6)

- Example 2: Let we have a table 'Account\_interest' to record when the account balance (in account table) is updated:
- ```
CREATE TABLE account_interest  
( account_number INT(8),  
  balance_before INT(8),  
  balance_after INT (8));
```

# TRIGGERS (7)

Example 2 ... contd ...

```
DELIMITER $$  
  CREATE TRIGGER update_interest  
  AFTER UPDATE ON account  
  FOR EACH ROW  
  BEGIN  
    INSERT INTO account_interest  
  (account_number, balance_before, balance_after) VALUES  
  (NEW.account_number, OLD.balance, NEW.balance);  
  END $$  
DELIMITER ;
```

# TRIGGERS (8)

- Example 2 ... contd ...

- Now run update query

**update** *account*

**set** *balance* = *balance* \* 1.1

**where** *balance* > 500

- This will update balance in account table and make an entry in Account\_interest table

# TRIGGERS (9)

- Example 3: consider the following schema for a university:  
  
    Student(rollno, name, address, CPI)  
  
    Campus(location, rank)  
  
    Apply(rollno, location, date, programme, decision)
- Write trigger for the rule: if a student with  $CPI > 8$  applies for any programme in Jabalpur campus, then he/she is accepted for registration (i.e., decision is set to 'Y').

# TRIGGERS (10)

- If a student with  $CPI > 8$  applies for any program in Jabalpur campus, then he/she is accepted for registration (i.e., decision is set to 'Y').

```
CREATE TRIGGER acceptjbpcampus
```

```
AFTER INSERT ON Apply
```

```
FOR EACH ROW
```

```
BEGIN
```

```
    WHEN (NEW.location = 'Jabalpur' AND
```

```
        (SELECT CPI FROM Student WHERE rollno = NEW.rollno) > 8)
```

```
        UPDATE Apply
```

```
        SET decision = 'Y'
```

```
        WHERE rollno = NEW. Rollno
```

```
            AND location = NEW.location
```

```
            AND date = NEW. date
```

```
END
```

# TRIGGERS (11)

- Trigger for the rule: if a campus registration increases from below 10,000 to 10,000 or more, then delete all applications to that campus dated after 25/04/2014 and set all 'Y' decisions for applications between 20/04/2014 to 24/04/2014 to 'U'.

```
CREATE TRIGGER registrationcontrol  
AFTER UPDATE OF registrations ON Campus  
FOR EACH ROW  
BEGIN  
WHEN (OLD.registrations < 10,000 AND NEW.registrations >= 10,000)  
    DELETE FROM Apply  
    WHERE location = NEW.location AND date > '25/4/2014'  
    UPDATE Apply  
    SET decision = 'U'  
    WHERE location = NEW.location  
        AND decision = 'Y'  
        AND date between '20/04/2014' AND '24/04/2014'  
END
```



# TRIGGERS (8)

- TASK: Create a TRIGGER which insert customer phone number to a new table called Customer\_Phone with customer name on insertion of new customer.

# TRIGGERS (9)

- Example 3: Let we have a table 'Customer\_phone,' to store customer name and the phone number of each customer. If a new customer is added to customer table, he or she is also added to the Customer\_phone table.

```
CREATE TABLE Customer_phone
    (customer_name VARCHAR(15) NOT NULL
    PRIMARY KEY,
    phone_number INTEGER NOT NULL);
```

# TRIGGERS (10)

- Example 2 ... contd ...

```
DELIMITER $$
```

```
CREATE TRIGGER insert_customer2
```

```
AFTER INSERT ON customer
```

```
FOR EACH ROW
```

```
BEGIN
```

```
INSERT INTO Customer_phone
```

```
VALUES(NEW.customer_name, 0);
```

```
END$$
```

```
DELIMITER ;
```

# DROPPING TRIGGERS

- Syntax to drop a trigger:
  - DROP TRIGGER [IF EXISTS]  
    *[schema\_name.]trigger\_name*
- It was added in MySQL 5.0.2.
- Its use requires the SUPER privilege.
- Make use of IF EXISTS to prevent an error from occurring for a trigger that does not exist.
- The IF EXISTS clause was added in MySQL 5.0.32.
- Example:
  - DROP TRIGGER **insert\_customer1**