# Discussion-4: Preventing accidental changes to the tables

**Discussion Topic:**

Executing Insert, Update and Delete is an instance operation. You might delete a table that you did not intend to. What are some practices that you can deploy to prevent accidental changes to the tables?

**My Post:**

Hello Class,

Structured Query Language (SQL) is a standard language that is used within Database Management System (RDBMS) to create and manipulate databases and the data they store. Within SQL, manipulating records (data values) is handled by Data Manipulation Language (DML), a subset of SQL. The manipulation of the databases themselves, also called schemas, is handled by the Data Definition Language (DDL). This article explores MySQL best practices for handling data manipulation safely when using DML and DDL operations.

**MySQL and the Principle of Least Privilege**

MySQL, an RDBMS based on SQL, provides a privilege system that manages applications' and users' access to database servers, including the ability to drop (delete) databases, modify any data (records), and change user permissions. This system is based on the Principle of Least Privilege (PoLP) where users and applications are granted only the exact permissions required to perform their intended tasks, and no more. These permissions are managed by a *root* account, which is usually used only for that purpose. The permissions are based on the users/applications' scopes and needed functionality; additionally, these permissions are not static or a one-time based configuration, as the permissions should be revoked or modified as soon as they are no longer needed or the users/applications' scopes change.

**Permission Setting and Description**

In MySQL, the permissions are set using the GRANT statement, which is used to assign (grant) privileges, operations allowed, and the scope (such as global, database, table, or column) to the target user. The following tables showcase various permissions for DML and DDL operations.

**Table 1**
*MySQL User Permissions for DML Operations*

| Scope | Description | Example Grant for DML | |
|---|---|---|---|
| **GLOBAL (.)** | Applies to all databases on the server. | `GRANT SELECT, INSERT ON *.* TO 'user'@'host';` | User can read data from and add new data to any table in any database on the server. |
| **DATABASE (db.*)** | Applies to all objects within a specific database. | `GRANT UPDATE, DELETE ON my_app_db.* TO 'user'@'host';` | User can change existing data in and remove data from any table or view within the my_app_db database. |

| | | | |
|---|---|---|---|
| **TABLE (db.tbl)** | Applies to a specific table. | `GRANT INSERT ON my_app_db.orders TO 'user'@'host';` | User can add new data (rows) to the orders table within the my_app_db database. |
| **COLUMN** | Applies to specific columns within a table (SELECT, INSERT, UPDATE only). | `GRANT SELECT (col1, col2), UPDATE (col3) ON db.tbl TO 'user'@'host';` | User can read the data in col1 and col2, and change the existing data in col3 for any row in the table tbl within the database db. |
| **ROUTINE** | Applies to stored procedures/functions (EXECUTE privilege). | `GRANT EXECUTE ON PROCEDURE db.proc_name TO 'user'@'host';` | User can run the stored procedure named proc_name located in the database db. The actions performed depend on the SQL statements within this procedure. |

*Note:* The table lists the MySQL permission scopes for DML operations, describes them, and what actions the user is permitted to do.

**Table 2**

*User Permissions for DDL Operations*

| Privilege Scope | Description | Example DDL Privileges & Grant | |
|---|---|---|---|
| **GLOBAL (.)** | Applies to all databases and server-wide objects (e.g., users, roles, tablespaces). | CREATE, ALTER, DROP (for databases, tables, views, routines, etc.), CREATE USER, CREATE ROLE, DROP ROLE, CREATE TABLESPACE, INDEX, REFERENCES, TRIGGER.<br>Example:<br>`GRANT CREATE, ALTER, DROP ON *.* TO 'user'@'host';`<br>Example:<br>`GRANT CREATE USER ON *.* TO 'admin_user'@'host';` | With GRANT CREATE, ALTER, DROP ON *.*, the 'user' can create new databases and tables, modify the structure of any existing database or table, or delete any database or table on the server. With GRANT CREATE USER ON *.*, the 'admin_user' can create new MySQL users on the server. |
| **DATABASE (db.*)** | Applies to all objects (tables, views, routines, triggers, indexes) within a specific database. | CREATE (tables, views, routines, triggers, indexes within the DB), ALTER (tables, views, routines within the DB), DROP (tables, views, routines, triggers, indexes within the DB), CREATE TEMPORARY TABLES, EVENT.<br>Example:<br>`GRANT CREATE, ALTER, DROP ON my_app_db.* TO 'user'@'host';` | User can create new objects (like tables, views), change the structure of existing objects, or remove any object within the my_app_db database. |
| **TABLE (db.tbl)** | Applies to a specific table. | ALTER (the table structure), CREATE VIEW (on the table), DROP (the table itself), INDEX (create/drop indexes on the table), TRIGGER (create/drop triggers on the table), REFERENCES (create foreign keys referencing columns of the table).<br>Example: | With GRANT ALTER, INDEX, TRIGGER ON my_app_db.orders, the user can change the structure of the orders table, create or remove indexes on it, and define or delete triggers for it. With GRANT DROP ON |

| | | | |
|---|---|---|---|
| | | `GRANT ALTER, INDEX, TRIGGER ON my_app_db.orders TO 'user'@'host';`<br>`Example: GRANT DROP ON my_app_db.temporary_table TO 'user'@'host';` | my_app_db.temporary_table, the user can delete the temporary_table. |
| **ROUTINE (PROCEDURE db.proc_name or FUNCTION db.func_name)** | Applies to specific stored procedures or functions. | ALTER ROUTINE, EXECUTE (while EXECUTE is DML-related for running, ALTER ROUTINE is DDL for modifying).<br><br>Example:<br>`GRANT ALTER ROUTINE ON PROCEDURE my_app_db.process_data TO 'user'@'host';` | User can change the definition of or delete the stored procedure named process_data within the my_app_db database. |

*Note:* The table lists the MySQL permission scopes for DDL operations, describes them, and what actions the user is permitted to do.

**MySQL Best Practices for Manipulating Records**

As shown in Table 1, to insert new records, update or modify existing records, and delete records, DML provides three core commands: INSERT, UPDATE, and DELETE. These commands are powerful tools that need to be used with meticulous care, as they can result in accidental and permanent record/data loss or corruption. In addition to managing permissions, MySQL provides commands such as *SQL_SAFE_UPDATES* and START TRANSACTION that add a layer of protection against accidental data modifications or loss, as they allow for some control and to reverse changes, notably when using the command UPDATE and DELETE.

The *SQL_SAFE_UPDATES* is a server system variable that helps prevent accidental mass updates or deletions. When enabled (SET SQL_SAFE_UPDATES = 1;), MySQL will reject *UPDATE* and *DELETE* statements unless they include a *WHERE* clause (selecting specifies rows using). Note if you are using MySQL Workbench, you can disable Safe Mode by going to Edit -> Preferences, selecting the SQL Editor tab, and unchecking the "Safe Updates" box.

The *START TRANSACTION* command allows multiple SQL statements to be treated as a single, atomic unit of work, a module, providing the option to rollback any modifications (done by the SQL statements that are part of the transaction module) before permanently committing them. The MySQL documentation provides the following syntax for it:

```
START TRANSACTION
    [transaction_characteristic [, transaction_characteristic] ...]

transaction_characteristic: {
    WITH CONSISTENT SNAPSHOT
  | READ WRITE
  | READ ONLY
}

BEGIN [WORK]
COMMIT [WORK] [AND [NO] CHAIN] [[NO] RELEASE]
```

```
ROLLBACK [WORK] [AND [NO] CHAIN] [[NO] RELEASE]
SET autocommit = {0 | 1}
```

Note: from "15.3.1 START TRANSACTION, COMMIT, and ROLLBACK Statements" by MySQL Documentation (n.d.)

- START TRANSACTION or BEGIN start a new transaction.
- COMMIT commits the current transaction, making its changes permanent.
- ROLLBACK rolls back the current transaction, canceling its changes.
- SET autocommit disables or enables the default autocommit mode for the current session.

Note that by default *autocommit* mode is turned ON (SET *autocommit = 1*), and, SQL statement you execute that modifies data (like *INSERT*, *UPDATE*, *DELETE*) is treated as its own independent transaction, meaning that when the statement finishes successfully, MySQL automatically performs a *COMMIT* **->** the changes are immediately saved and you cannot use *ROLLBACK* to undo.
So it is essential to explicitly set *autocommit = 0*. That is, if you are planning to use *ROLLBACK*.

Here is an example:

```
-- Step 0: Verify the row that need to be change
SELECT product_id, product_name, discount_percent
FROM products
WHERE category_id = 1;

-- Step 1: Start the transaction
START TRANSACTION;

-- Step 2: Apply temporary changes
UPDATE products
SET discount_percent = 50.00
WHERE category_id = 1;

-- Step 3: Check the temporary change within the transaction
SELECT product_id, product_name, discount_percent
FROM products
WHERE category_id = 1;

-- Step 4: Decide not to proceed and undo the changes
ROLLBACK;

--  Step 5: Verify that the changes were undone
SELECT product_id, product_name, discount_percent
FROM products
WHERE category_id = 1;
```

**MySQL Best Practices for Manipulating Schema Data**

Similar to DML, DDL provides three core commands to manipulate database schemas and tables: *CREATE*, *ALTER*, and *DROP*. These commands are even more powerful than DML commands, and they need to be used with meticulous care, as they can result in accidental and permanent vast data loss. Some best practices are in addition to managing permissions, follow best practices such as planning the change, creating backups, and decoupling (separating the schema change from the application deployment or release process).

To summarize, manipulating data improperly within RDBNS can result in accidental loss of data that can be significant to catastrophic. To mitigate these risks, MySQL provides a permissions framework and tools (e.g., *GRANT*), based on the Principle of Least Privilege (PoLP), that, when combined with best practice such as using transactions (including START TRANSACTION with ROLLBACK) and safe update modes (like SQL_SAFE_UPDATES) for DML operations, alongside planning, creating backups, and decoupling changes for DDL operations—greatly reduces the risk of accidental data loss or corruption.

-Alex

**References:**

MySQL Documentation (n.d.). 15.3.1 START TRANSACTION, COMMIT, and ROLLBACK statement. MySQL. https://dev.mysql.com/doc/refman/8.4/en/commit.html