MySQL: Subqueries, TCL, DCL, Views

Tushar B. Kute,
http://tusharkute.com

Subquery

- A subquery in MySQL is a query, which is nested into another SQL query and embedded with SELECT, INSERT, UPDATE or DELETE statement along with the various operators.
- We can also nest the subquery with another subquery.
- A subquery is known as the inner query, and the query that contains subquery is known as the outer query.

Subquery

- The inner query executed first gives the result to the outer query, and then the main/outer query will be performed.
- MySQL allows us to use subquery anywhere, but it must be closed within parenthesis.
- All subquery forms and operations supported by the SQL standard will be supported in MySQL also.

Subquery: Rules

- Subqueries should always use in parentheses.
- If the main query does not have multiple columns for subquery, then a subquery can have only one column in the SELECT command.
- We can use various comparison operators with the subquery, such as >, <, =, IN, ANY, SOME, and ALL. A multiple-row operator is very useful when the subquery returns more than one row.
- We cannot use the ORDER BY clause in a subquery, although it can be used inside the main query.
- If we use a subquery in a set function, it cannot be immediately enclosed in a set function.

Subquery: Advantages

- The subqueries make the queries in a structured form that allows us to isolate each part of a statement.
- The subqueries provide alternative ways to query the data from the table; otherwise, we need to use complex joins and unions.
- The subqueries are more readable than complex join or union statements.

Subquery: Syntax

```
SELECT column_list (s) FROM  table_name
WHERE  column_name OPERATOR (SELECT
column_list (s)  FROM table_name
[WHERE])
```

Subquery: Syntax

- Let us understand it with the help of an example.
- Suppose we have a table named "employees" that contains the following data:

```
mysql> SELECT * FROM employees;
+--------+-----------+---------+------------+---------+
| emp_id | emp_name  | emp_age | city       | income  |
+--------+-----------+---------+------------+---------+
|    101 | Peter     |      32 | Newyork    |  200000 |
|    102 | Mark      |      32 | California |  300000 |
|    103 | Donald    |      40 | Arizona    | 1000000 |
|    104 | Obama     |      35 | Florida    | 5000000 |
|    105 | Linklon   |      32 | Georgia    |  250000 |
|    106 | Kane      |      45 | Alaska     |  450000 |
|    107 | Adam      |      35 | California | 5000000 |
|    108 | Macculam  |      40 | Florida    |  350000 |
|    109 | Brayan    |      32 | Alaska     |  400000 |
|    110 | Stephen   |      40 | Arizona    |  600000 |
|    111 | Alexander |      45 | California |   70000 |
+--------+-----------+---------+------------+---------+
```

Subquery: Syntax

- Following is a simple SQL statement that returns the employee detail whose id matches in a subquery:
  - SELECT emp_name, city, income FROM employees WHERE emp_id IN (SELECT emp_id FROM employees);

```
mysql> SELECT emp_name, city, income FROM employees
    ->    WHERE emp_id IN (SELECT emp_id FROM employees);
+-----------+------------+---------+
| emp_name  | city       | income  |
+-----------+------------+---------+
| Peter     | Newyork    |  200000 |
| Mark      | California |  300000 |
| Donald    | Arizona    | 1000000 |
| Obama     | Florida    | 5000000 |
| Linklon   | Georgia    |  250000 |
| Kane      | Alaska     |  450000 |
| Adam      | California | 5000000 |
| Macculam  | Florida    |  350000 |
| Brayan    | Alaska     |  400000 |
| Stephen   | Arizona    |  600000 |
| Alexander | California |   70000 |
+-----------+------------+---------+
```

Subquery: With comparison

- A comparison operator is an operator used to compare values and returns the result, either true or false.
- The following comparison operators are used in MySQL <, >, =, <>, <=>, etc. We can use the subquery before or after the comparison operators that return a single value.
- The returned value can be the arithmetic expression or a column function.
- After that, SQL compares the subquery results with the value on the other side of the comparison operator.

Subquery: Example

- Following is a simple SQL statement that returns the employee detail whose income is more than 350000 with the help of subquery:
  - SELECT * FROM employees WHERE emp_id IN (SELECT emp_id FROM employees WHERE income > 350000);
- Let us see an example of another comparison operator, such as equality (=) to find employee details with maximum income using a subquery.
  - SELECT emp_name, city, income FROM employees WHERE income = (SELECT MAX(income) FROM employees);

Correlated Subquery

- A correlated subquery in MySQL is a subquery that depends on the outer query.
- It uses the data from the outer query or contains a reference to a parent query that also appears in the outer query.
- MySQL evaluates it once from each row in the outer query.

Correlated Subquery

```
SELECT emp_name, city, income  FROM
employees emp WHERE income > (SELECT
AVG(income) FROM employees WHERE city =
emp.city);
```

- In the above query, we select an employee name and city whose income is higher than the average income of all employees in each city.

Correlated Subquery

```
mysql> SELECT emp_name, city, income
    -> FROM employees emp WHERE income > (
    -> SELECT AVG(income) FROM employees WHERE city = emp.city);
+----------+------------+---------+
| emp_name | city       | income  |
+----------+------------+---------+
| Donald   | Arizona    | 1000000 |
| Obama    | Florida    | 5000000 |
| Kane     | Alaska     |  450000 |
| Adam     | California | 5000000 |
+----------+------------+---------+
```

Subquery with EXISTS or NOT EXISTS

- The EXISTS operator is a Boolean operator that returns either true or false result.
- It is used with a subquery and checks the existence of data in a subquery.
- If a subquery returns any record at all, this operator returns true. Otherwise, it will return false. The NOT EXISTS operator used for negation that gives true value when the subquery does not return any row.
- Otherwise, it returns false. Both EXISTS and NOT EXISTS used with correlated subqueries. The following example illustrates it more clearly.

Subquery with EXISTS or NOT EXISTS

```
mysql> SELECT * FROM customer;
+---------+---------+------------+------+
| cust_id | name    | occupation | age  |
+---------+---------+------------+------+
|     101 | Peter   | Engineer   |   32 |
|     102 | Joseph  | Developer  |   30 |
|     103 | John    | Leader     |   28 |
|     104 | Stephen | Scientist  |   45 |
|     105 | Suzi    | Carpenter  |   26 |
|     106 | Bob     | Actor      |   25 |
|     107 | NULL    | NULL       | NULL |
+---------+---------+------------+------+
7 rows in set (0.00 sec)

mysql> SELECT * FROM Orders;
+----------+---------+-----------+------------+
| order_id | cust_id | prod_name | order_date |
+----------+---------+-----------+------------+
|        1 |     101 | Laptop    | 2020-01-10 |
|        2 |     103 | Desktop   | 2020-02-12 |
|        3 |     106 | Iphone    | 2020-02-15 |
|        4 |     104 | Mobile    | 2020-03-05 |
|        5 |     102 | TV        | 2020-03-20 |
+----------+---------+-----------+------------+
```

Subquery with EXISTS or NOT EXISTS

- The below SQL statements uses EXISTS operator to find the name, occupation, and age of the customer who has placed at least one order.

```
SELECT name, occupation, age FROM
customer C  WHERE EXISTS (SELECT * FROM
Orders O WHERE C.cust_id = O.cust_id);
```

Subquery with EXISTS or NOT EXISTS

- This statement uses NOT EXISTS operator that returns the customer details who have not placed an order.

```
SELECT name, occupation, age FROM customer
C  WHERE NOT EXISTS (SELECT * FROM Orders
O  WHERE C.cust_id = O.cust_id);
```

Subquery with EXISTS or NOT EXISTS

```
mysql> SELECT name, occupation, age FROM customer C
    -> WHERE EXISTS (SELECT * FROM Orders O
    -> WHERE C.cust_id = O.cust_id);
+---------+------------+------+
| name    | occupation | age  |
+---------+------------+------+
| Peter   | Engineer   |   32 |
| Joseph  | Developer  |   30 |
| John    | Leader     |   28 |
| Stephen | Scientist  |   45 |
| Bob     | Actor      |   25 |
+---------+------------+------+
5 rows in set (0.10 sec)

mysql> SELECT name, occupation, age FROM customer C
    -> WHERE NOT EXISTS (SELECT * FROM Orders O
    -> WHERE C.cust_id = O.cust_id);
+------+------------+------+
| name | occupation | age  |
+------+------------+------+
| Suzi | Carpenter  |   26 |
| NULL | NULL       | NULL |
+------+------------+------+
```

Transaction Control Language

- A single unit of work in a database is formed after the consecutive execution of commands is known as a transaction.
- There are certain commands present in SQL known as TCL commands that help the user manage the transactions that take place in a database.
- COMMIT. ROLLBACK and SAVEPOINT are the most commonly used TCL commands in SQL.

Commit

- COMMIT command in SQL is used to save all the transaction-related changes permanently to the disk.
- Whenever DDL commands such as INSERT, UPDATE and DELETE are used, the changes made by these commands are permanent only after closing the current session.
- So before closing the session, one can easily roll back the changes made by the DDL commands. Hence, if we want the changes to be saved permanently to the disk without closing the session, we will use the commit command.
- Syntax:
  - COMMIT;

Commit

- mysql> USE school;
- mysql> CREATE TABLE t_school….
- mysql> START TRANSACTION;
- mysql> INSERT INTO t_school…….
- mysql> SELECT *FROM t_school;
- mysql> COMMIT;

Commit

- Autocommit is by default enabled in MySQL. To turn it off, we will set the value of autocommit as 0.

  mysql> SET autocommit = 0;

```
mysql> SET autocommit = 0;
Query OK, 0 rows affected (0.08 sec)
```

- MySQL, by default, commits every query the user executes. But if the user wishes to commit only the specific queries instead of committing every query, then turning off the autocommit is useful.

Savepoint

- We can divide the database operations into parts.
- For example, we can consider all the insert related queries that we will execute consecutively as one part of the transaction and the delete command as the other part of the transaction.
- Using the SAVEPOINT command in SQL, we can save these different parts of the same transaction using different names.

Savepoint

- For example, we can save all the insert related queries with the savepoint named INS.
- To save all the insert related queries in one savepoint, we have to execute the SAVEPOINT query followed by the savepoint name after finishing the insert command execution.
- Syntax:
  - SAVEPOINT savepoint_name;

Rollback

- While carrying a transaction, we must create savepoints to save different parts of the transaction.
- According to the user's changing requirements, he/she can roll back the transaction to different savepoints.
- Consider a scenario: We have initiated a transaction followed by the table creation and record insertion into the table.
- After inserting records, we have created a savepoint INS. Then we executed a delete query, but later we thought that mistakenly we had removed the useful record.

Rollback

- Therefore in such situations, we have an option of rolling back our transaction.
- In this case, we have to roll back our transaction using the ROLLBACK command to the savepoint INS, which we have created before executing the DELETE query.
- Syntax:

```
ROLLBACK TO savepoint_name;
```

Rollback

- mysql> USE school;
- mysql> CREATE TABLE t_school….
- mysql> INSERT INTO t_school
- mysql> SELECT *FROM t_school;
- mysql> START TRANSACTION;
- mysql> SAVEPOINT Insertion;
- mysql> UPDATE t_school SET….
- mysql> SELECT *FROM t_school;
- mysql> SAVEPOINT Updation;
- mysql> ROLLBACK TO Insertion;
- mysql> SELECT *FROM t_school;

Rollback

- mysql> USE bank;
- mysql> CREATE TABLE customer….
- mysql> INSERT INTO customer….
- mysql> SELECT *FROM customer;
- mysql> START TRANSACTION;
- mysql> SAVEPOINT Insertion;
- mysql> DELETE FROM customer WHERE….
- mysql> SELECT *FROM customer;
- mysql> SAVEPOINT Deletion;
- mysql> ROLLBACK TO Insertion;
- mysql> SELECT *FROM customer;

Data Control Language

- Data control language (DCL) is used to access the stored data. It is mainly used for revoke and to grant the user the required access to a database.
- It helps in controlling access to information stored in a database. It complements the data manipulation language (DML) and the data definition language (DDL).
- It provides the administrators, to remove and set database permissions to desired users as needed.
- These commands are employed to grant, remove and deny permissions to users for retrieving and manipulating a database.

Grant

- 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.
- Syntax

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

Grant

- privilege_name(s)
  - It specifies the access rights or grant privilege to user accounts. If we want to give multiple privileges, then use a comma operator to separate them.
- object
  - It determines the privilege level on which the access rights are being granted. It means granting privilege to the table; then the object should be the name of the table.
- user_account_name
  - It determines the account name of the user to whom the access rights would be granted.

Grant : Privilege Levels

- Global

```
GRANT ALL
ON *.*
TO john@localhost;
```

  - It applies to all databases on MySQL server. We need to use *.* syntax for applying global privileges. Here, the user can query data from all databases and tables of the current server.

Grant : Privilege Levels

- Database

```
GRANT ALL
ON mydb.*
TO john@localhost;
```

  - It applies to all objects in the current database. We need to use the db_name.* syntax for applying this privilege.
  - Here, a user can query data from all tables in the given database.

Grant : Privilege Levels

- Table

```
GRANT DELETE
ON mydb.employees
TO john@localhost;
```

  - It applies on all columns in a specified table. We need to use db_name.table_name syntax for assigning this privilege.
  - Here, a user can query data from the given table of the specified database.

Grant : Privilege Levels

- Column

```
GRANT SELECT (col1), INSERT (col1,
col2), UPDATE (col2)
ON mydb.mytable
TO john@localhost;
```

  - It applies on a single column of a table. Here, we must have to specify the column(s) name enclosed with parenthesis for each privilege.
  - The user can select one column, insert values in two columns, and update only one column in the given table.

Grant : Privilege Levels

- Stored Routine

```
GRANT EXECUTE
ON PROCEDURE mydb.myprocedure
TO john@localhost;
```

  - It applies to stored routines (procedure and functions).
  - It contains CREATE ROUTINE, ALTER ROUTINE, EXECUTE, and GRANT OPTION privileges.
  - Here, a user can execute the stored procedure in the current database.

Grant : Privilege Levels

- Proxy

```
GRANT PROXY
ON root
TO tushar@localhost;
```

  - It enables one user to be a proxy for other users.

Grant : Example

- Let us understand the GRANT privileges through the example. First, we need to create a new user named "john@localhost" using the following statement:

```
mysql> CREATE USER john@localhost
IDENTIFIED BY 'john12345';
```

- Next, execute the SHOW GRANT statement to check the privileges assigned to john@localhost using the following query:

```
mysql> SHOW GRANTS FOR john@localhost;
```

```
mysql> GRANT ALL ON mystudentdb.* TO
john@localhost;
```

Revoke Statement

- 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.
- Syntax:

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

Revoke: Privileges

- Global

```
REVOKE ALL, GRANT OPTION FROM
john@localhost;
```

  - It applies to remove all access rights from the user on MySQL server.

Revoke: Privileges

- Database

```
REVOKE ALL ON mydb.*
FROM john@localhost;
```

  - It applies to revoke all privileges from objects in the current database.

Revoke: Privileges

- Table

```
REVOKE DELETE
ON mydb.employees
FROM john@localhost;
```

  - It applies to revoke privileges from all columns in a specified table.

Revoke: Privileges

- Column

```
REVOKE SELECT (col1), INSERT (col1,
col2), UPDATE (col2) ON mydb.mytable
FROM john@localhost;
```

  - It applies to revoke privileges from a single column of a table.

Revoke: Privileges

- Stored Routine

```
REVOKE EXECUTE ON PROCEDURE/FUNCTION
mydb.myprocedure
FROM john@localhost;
```

  - It applies to revoke all privileges from stored routines (procedure and functions).

Revoke: Privileges

- Proxy

```
REVOKE PROXY ON root
FROM peter@localhost;
```

  - It enables us to revoke the proxy user.

Revoke: Example

- `mysql> REVOKE ALL, GRANT OPTION FROM john@localhost;`
- `mysql> GRANT SELECT, UPDATE, INSERT ON mystudentdb.* TO john@localhost;`
- `mysql> SHOW GRANTS FOR john@localhost;`
- `mysql> REVOKE UPDATE, INSERT ON mystudentdb.* FROM john@localhost;`

View

- A view is a database object that has no values. Its contents are based on the base table. It contains rows and columns similar to the real table.
- In MySQL, the View is a virtual table created by a query by joining one or more tables.
- It is operated similarly to the base table but does not contain any data of its own.
- The View and table have one main difference that the views are definitions built on top of other tables (or views).
- If any changes occur in the underlying table, the same changes reflected in the View also.

View

- We can create a new view by using the CREATE VIEW and SELECT statement. SELECT statements are used to take data from the source table to make a VIEW.
- Syntax:

```
CREATE [OR REPLACE] VIEW view_name AS
SELECT columns
FROM tables
[WHERE conditions];
```

View

- ```
  CREATE VIEW trainer AS
  SELECT course_name, trainer
  FROM courses;
  ```

- ```
  SELECT * FROM view_name;
  ```

Update View

- In MYSQL, the ALTER VIEW statement is used to modify or update the already created VIEW without dropping it.
- Syntax:

```
ALTER VIEW view_name AS
SELECT columns
FROM table
WHERE conditions;
```

Drop View

- We can drop the existing VIEW by using the DROP VIEW statement.
- Syntax:

```
DROP VIEW [IF EXISTS] view_name;
```

View using join statement

```
CREATE VIEW Trainer
AS SELECT c.course_name, c.trainer, t.email
FROM courses c, contact t
WHERE c.id = t.id;
```

Summary

- Views are virtual tables; they do not contain the data that is returned. The data is stored in the tables referenced in the SELECT statement.
- Views improve security of the database by showing only intended data to authorized users. They hide sensitive data.
- Views make life easy as you do not have write complex queries time and again.
- It’s possible to use INSERT, UPDATE and DELETE on a VIEW. These operations will change the underlying tables of the VIEW.
- The only consideration is that VIEW should contain all NOT NULL columns of the tables it references. Ideally, you should not use VIEWS for updating.

Thank you

@mitu_skillologies

@mITuSkillologies

@mitu_group

@mitu-skillologies

@MITUSkillologies

kaggle

@mituskillologies

Web Resources
https://mitu.co.in
http://tusharkute.com

@mituskillologies

contact@mitu.co.in

tushar@tusharkute.com