# MySQL: Flow-Control Statements

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

# 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.

# Index

- An index is a data structure that allows us to add indexes in the existing table. It enables you to improve the faster retrieval of records on a database table.
- It creates an entry for each value of the indexed columns. We use it to quickly find the record without searching each row in a database table whenever the table is accessed.
- We can create an index by using one or more columns of the table for efficient access to the records.

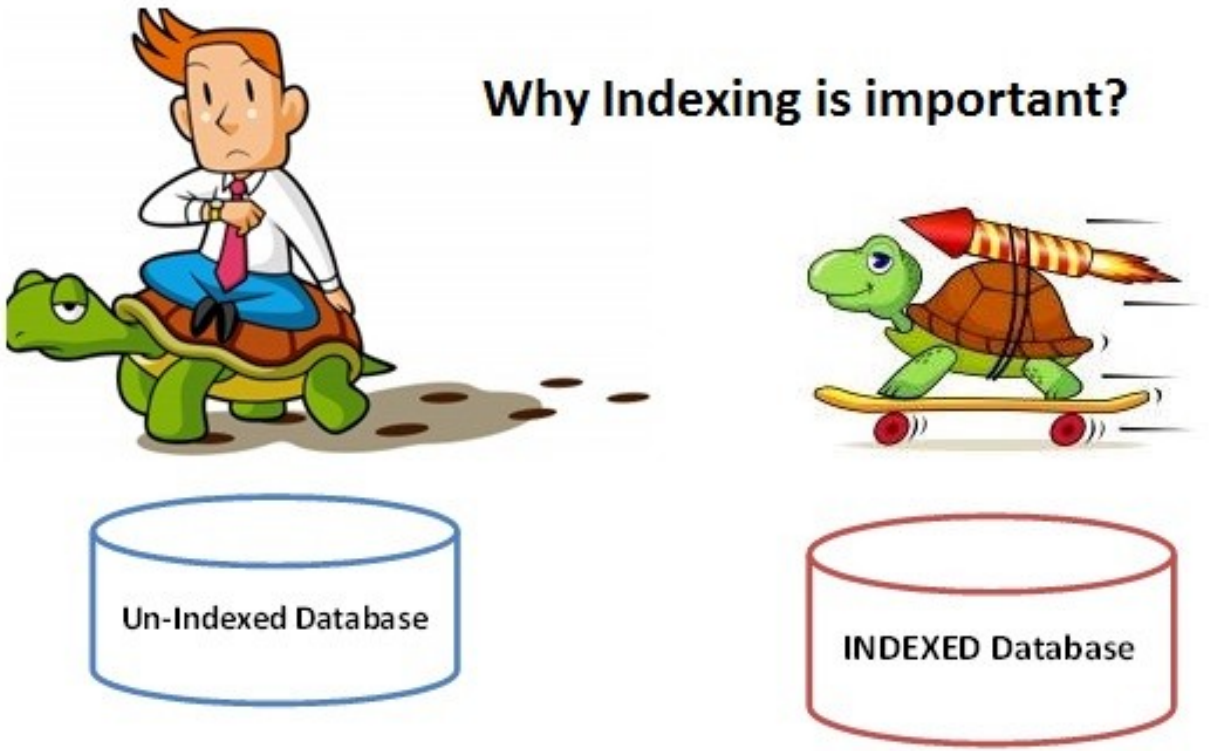
# Index

- When a table is created with a primary key or unique key, it automatically creates a special index named PRIMARY.
- We called this index as a clustered index. All indexes other than PRIMARY indexes are known as a non-clustered index or secondary index.

# Indexing need

- Suppose we have a contact book that contains names and mobile numbers of the user. In this contact book, we want to find the mobile number of Martin Williamson.
- If the contact book is an unordered format means the name of the contact book is not sorted alphabetically, we need to go over all pages and read every name until we will not find the desired name that we are looking for.
- This type of searching name is known as sequential searching.

# Why indexing ?



# Syntax:

- Generally, we create an index at the time of table creation in the database. The following statement creates a table with an index that contains two columns col2 and col3.

```
mysql> CREATE TABLE t_index(
    col1 INT PRIMARY KEY,
    col2 INT NOT NULL,
    col3 INT NOT NULL,
    col4 VARCHAR(20),
    INDEX (col2,col3)
);
```

# Syntax:

- If we want to add index in table, we will use the CREATE INDEX statement as follows:

```
mysql> CREATE INDEX [index_name] ON
[table_name] (column names)
```

- In this statement, index_name is the name of the index, table_name is the name of the table to which the index belongs, and the column_names is the list of columns.
- Let us add the new index for the column col4, we use the following statement:

```
mysql> CREATE INDEX ind_1 ON t_index(col4);
```

# Syntax:

- If you want to show the indexes of a table, execute the following statement:

```
mysql> SHOW INDEXES FROM student;
```

- If we want to get the index information of a table in a different database or database to which you are not connected, MySQL allows us to specify the database name with the Show Indexes statement. The following statement explains it more clearly:

```
mysql> SHOW INDEXES FROM table_name IN
database_name;
```

# Syntax:

- Table: It contains the name of the table.
- Non_unique: It returns 1 if the index contains duplicates. Otherwise, it returns 0.
- Key_name: It is the name of an index. If the table contains a primary key, the index name is always PRIMARY.
- Seq_in_index: It is the sequence number of the column in the index that starts from 1.
- Column_name: It contains the name of a column.

# Syntax:

- Collation: It gives information about how the column is sorted in the index. It contains values where A represents ascending, D represents descending, and Null represents not sorted.
- Cardinality: It gives an estimated number of unique values in the index table where the higher cardinality represents a greater chance of using indexes by MySQL.
- Sub_part: It is a prefix of the index. It has a NULL value if all the column of the table is indexed. When the column is partially indexed, it will return the number of indexed characters.
- Packed: It tells how the key is packed. Otherwise, it returns NULL.

# Syntax:

- NULL: It contains blank if the column does not have NULL value; otherwise, it returns YES.
- Index_type: It contains the name of the index method like BTREE, HASH, RTREE, FULLTEXT, etc.
- Comment: It contains the index information when they are not described in its column. For example, when the index is disabled, it returns disabled.
- Index_column: When you create an index with comment attributes, it contains the comment for the specified index.
- Visible: It contains YES if the index is visible to the query optimizer, and if not, it contains NO.

# Drop index

- MySQL allows a DROP INDEX statement to remove the existing index from the table. To delete an index from a table, we can use the following query:

```
mysql>DROP INDEX index_name ON table_name
[algorithm_option | lock_option];
```

- If we want to delete an index, it requires two things:
  - First, we have to specify the name of the index that we want to remove.
  - Second, name of the table from which your index belongs.

# Summary of index

- Indexes are very powerful when it comes to greatly improving the performance of MySQL search queries.
- Indexes can be defined when creating a table or added later on after the table has already been created.
- You can define indexes on more than one column on a table.
- The SHOW INDEX FROM table_name is used to display the defined indexes on a table.
- The DROP command is used to remove a defined index on a given table.

# Temporary Tables

- MySQL has a feature to create a special table called a Temporary Table that allows us to keep temporary data. We can reuse this table several times in a particular session.
- It is available in MySQL for the user from version 3.23, and above so if we use an older version, this table cannot be used.
- This table is visible and accessible only for the current session. MySQL deletes this table automatically as long as the current session is closed or the user terminates the connection.
- We can also use the DROP TABLE command for removing this table explicitly when the user is not going to use it.

# Temporary Tables

- If we use a PHP script to run the code, this table removes automatically as long as the script has finished its execution.
- If the user is connected with the server through the MySQL client, then this table will exist until the user closes the MySQL client program or terminates the connection or removed the table manually.
- A temporary table provides a very useful and flexible feature that allows us to achieve complex tasks quickly, such as when we query data that requires a single SELECT statement with JOIN clauses.
- Here, the user can use this table to keep the output and performs another query to process it.

# Temporary Tables

- MySQL uses the CREATE TEMPORARY TABLE statement to create a temporary table.
- This statement can only be used when the MySQL server has the CREATE TEMPORARY TABLES privilege.
- It can be visible and accessible to the client who creates it, which means two different clients can use the temporary tables with the same name without conflicting with each other.

# Temporary Tables

- A temporary table in MySQL will be dropped automatically when the user closes the session or terminates the connection manually.
- A temporary table can be created by the user with the same name as a normal table in a database.

# Temporary Tables

- In MySQL, the syntax of creating a temporary table is the same as the syntax of creating a normal table statement except the TEMPORARY keyword.
- Let us see the following statement which creates the temporary table:
  - `mysql> CREATE TEMPORARY TABLE table_name ( column_1, column_2, ..., table_constraints);`

# Temporary Tables

- If the user wants to create a temporary table whose structure is the same as an existing table in the database, then the above statement cannot be used. Instead, we use the syntax as given below:
  - `mysql> CREATE TEMPORARY TABLE temporary_table_name SELECT * FROM original_table_name LIMIT 0;`

# Temporary Tables

- Let us understand how we can create a temporary table in MySQL.
- Execute the following statement that creates a temporary table in the selected database:
  - `mysql> CREATE TEMPORARY TABLE Students( student_name VARCHAR(40) NOT NULL, total_marks DECIMAL(12,2) NOT NULL DEFAULT 0.00, total_subjects INT UNSIGNED NOT NULL DEFAULT 0);`

# Temporary Tables

- Next, we need to insert values in the temporary table:

```
mysql>INSERT INTO Students(student_name,
total_marks, total_subjects) VALUES
('Joseph', 150.75, 2), ('Peter', 180.75,
2);
```

# Temporary Tables

- Here, the structure of a temporary table is created by using the SELECT statement and merge two tables using the INNER JOIN clause and sorts them based on the price.
- Write the following statement in the MySQL prompt:

```
CREATE TEMPORARY TABLE temp_customers
SELECT c.cust_name, c.city, o.prod_name, o.price
FROM orders o
INNER JOIN customer c ON c.cust_id = o.order_id
ORDER BY o.price DESC;
```

# Drop Temporary Tables

```
mysql> DROP TEMPORARY TABLE
table_name;
```

- This query will not remove a permanent table of the database that means it only deletes a temporary table.

# IF function

- The IF function is one of the parts of the MySQL control flow function, which returns a value based on the given conditions.
- In other words, the IF function is used for validating a function in MySQL.
- The IF function returns a value YES when the given condition evaluates to true and returns a NO value when the condition evaluates to false.
- It returns values either in a string or numeric form depending upon the context in which this function is used.
- Sometimes, this function is known as IF-ELSE and IF THAN ELSE function.

# IF function

- The IF function takes three expressions, where the first expression will be evaluated.
- If the first expression evaluates to true, not null, and not zero, it returns the second expression. If the result is false, it returns the third expression.
- Syntax:

    IF ( expression 1, expression 2, expression 3)

# IF function

- Parameters:
- Expression 1
  - Required It is a value, which is used for validation.
- Expression 2
  - Optional It returns a value when the condition evaluates to true.
- Expression 3
  - Optional It returns a value when the condition evaluates to false.

# IF function

- The return type of IF function can be calculated as follows:
- If expression 2 or expression 3 are both strings or produce a string, the result is always a string.
- If expression 2 or expression 3 gives a floating-point value, the result is always a floating-point value.
- If expression 2 or expression 3 is an integer, the result is always an integer.

# Example:

- Example 1

  SELECT IF(200>350,'YES','NO') as result;
- In the above function, the (200>350) is a condition, which is evaluated.
- If the condition is true, it returns a value, YES, and if the condition is false, it returns NO.

# Example:

- Example 2

  SELECT IF(251 = 251,' Correct','Wrong') as result;
- In the above function, the (251 = 251) is a condition, which is evaluated. If the condition is true, it returns value Correct, and if the condition is false, it returns Wrong output.

# Example:

- Example 3

  SELECT IF(STRCMP('Sachin Tendulkar','Rahul Dravid')=0, 'Correct', 'Wrong') as value;

- The above example compares the two strings. If both the string is the same, it returns Correct. Otherwise, the IF function returns Wrong output.

# Example with select

```
SELECT name, class,
   IF(marks>=60,"First Class","NO")
   As Result
FROM student;
```

# Example with select

```
SELECT *,
IF(marks>=60 and class='TE',"YES","NO")
As Result
FROM student;
```

# If statement

- The IF statement is used in stored programs that implement the basic conditional construct in MySQL. Based on a certain condition, it allows us to execute a set of SQL statements.
- It returns one of the three values True, False, or NULL.
- We can use this statement in three ways IF-THEN, IF-THEN-ELSE, IF-THEN-ELSEIF-ELSE clauses, and can terminate with END-IF.

# If-then statement

- This statement executes a set of SQL queries based on certain conditions or expressions. The syntax of the IF-THEN statement is as follows:

  IF condition THEN

    statements;

  END IF;

- In the above syntax, we have to specify a condition for executing the code.
- If the statement evaluates to true, it will execute the statement between IF-THEN and END-IF. Otherwise, it will execute the statement following the END-IF.

# Example:

```
DELIMITER $$
CREATE PROCEDURE myResult(original_rate float, OUT discount_rate
float)
    NO SQL
     BEGIN
       IF (original_rate>200) THEN
         SET discount_rate=original_rate*.5;
       ELSE
         SET discount_rate=original_rate;
       END IF;
       select discount_rate;
    END$$
DELIMITER ;
```

# Example:

- Next, create two variables and set the value for both as below:

```
mysql> set @p = 150;
mysql> set @dp = 180;
```

- Now, call the stored procedure function to get the output.

```
mysql> call myResult(@p, @dp)
```

# IF-THEN-ELSEIF-ELSE Statement

- If we want to execute a statement based on multiple conditions, this statement can be used. The syntax of the IF-THEN-ELSE statement is given below:

```
IF condition THEN
  statements;
ELSEIF elseif-condition THEN
  elseif-statements;
...
ELSE
  else-statements;
END IF;
```

# Example:

```
DELIMITER $$
CREATE PROCEDURE myResult(original_rate float,OUT discount_rate float)
    NO SQL
     BEGIN
       IF (original_rate>500) THEN
          SET discount_rate=original_rate*.5;
       ELSEIF (original_rate<=500 AND original_rate>250) THEN
          SET discount_rate=original_rate*.8;
       ELSE
          SET discount_rate=original_rate;
        END IF;
       select discount_rate;
    END$$
DELIMITER ;
```

# Example:

- Next, create two variables and set the value for both as below:

```
mysql> set @p = 150;
mysql> set @dp = 150;
```

- Now, call the stored procedure function to get the output.

```
mysql> call myResult(@p, @dp)
```

# Case statements

- The CASE expression validates various conditions and returns the result when the first condition is true.
- Once the condition is met, it stops traversing and gives the output. If it will not find any condition true, it executes the else block.
- When the else block is not found, it returns a NULL value.
- The main goal of MySQL CASE statement is to deal with multiple IF statements in the SELECT clause.

# Case statements

- Syntax:

```
CASE value
   WHEN [compare_value] THEN result
   [WHEN [compare_value] THEN result ...]
   [ELSE result]
END
```

# Example:

- Example

```
mysql> SELECT CASE 1 WHEN 1 THEN 'one'
WHEN 2 THEN 'two' ELSE 'more' END;
```

# Searched CASE statement

```
SELECT id, name, marks, class,
CASE class
   WHEN 'TE' THEN '3rd'
   WHEN 'BE' THEN 'Final'
   WHEN 'SE' THEN '2nd'
   ELSE 'NONE'
END AS year from student;
```

# MySQL Function

- A stored function in MySQL is a set of SQL statements that perform some task/operation and return a single value.
- It is one of the types of stored programs in MySQL. When you will create a stored function, make sure that you have a CREATE ROUTINE database privilege.
- Generally, we used this function to encapsulate the common business rules or formulas reusable in stored programs or SQL statements.

# MySQL Function

- The stored function is almost similar to the procedure in MySQL, but it has some differences that are as follows:
  - The function parameter may contain only the IN parameter but can't allow specifying this parameter, while the procedure can allow IN, OUT, INOUT parameters.
  - The stored function can return only a single value defined in the function header.
  - The stored function may also be called within SQL statements.
  - It may not produce a result set.

# MySQL Function

- DELIMITER $$
  CREATE FUNCTION fun_name(fun_parameter(s))
  RETURNS datatype
  [NOT] {Characteristics}
  fun_body;

# MySQL Function

- fun_name
  - It is the name of the stored function that we want to create in a database. It should not be the same as the built-in function name of MySQL.
- fun_parameter
  - It contains the list of parameters used by the function body. It does not allow to specify IN, OUT, INOUT parameters.
- datatype
  - It is a data type of return value of the function. It should any valid MySQL data type.

# MySQL Function

- characteristics
  - The CREATE FUNCTION statement only accepted when the characteristics (DETERMINISTIC, NO SQL, or READS SQL DATA) are defined in the declaration.
- fun_body
  - This parameter has a set of SQL statements to perform the operations. It requires at least one RETURN statement.
  - When the return statement is executed, the function will be terminated automatically. The function body is given below: BEGIN -- SQL statements END $$ DELIMITER

# Example:

- Let us understand how stored function works in MySQL through the example.
- Suppose our database has a table named "customer" that contains the following data:

| 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 |

# Example:

```
DELIMITER $$
CREATE FUNCTION Customer_Occupation(
    age int
)
RETURNS VARCHAR(20)
DETERMINISTIC
BEGIN
    DECLARE customer_occupation VARCHAR(20);
    IF age > 35 THEN
        SET customer_occupation = 'Scientist';
    ELSEIF (age <= 35 AND
            age >= 30) THEN
        SET customer_occupation = 'Engineer';
    ELSEIF age < 30 THEN
        SET customer_occupation = 'Actor';
    END IF;
    -- return the customer occupation
    RETURN (customer_occupation);
END$$
DELIMITER;
```

# Example:

- Now, we are going to see how stored function is called with the SQL statement.
- The following statement uses customer_occupation stored function to get the result:

```
SELECT name, age, Customer_Occupation(age)
FROM customer ORDER BY age;
```

- It will give the output as below.

# Example:

```
mysql> SELECT name, age, Customer_Occupation(age)
    -> FROM customer ORDER BY age;
+---------+------+-----------------------------+
| name    | age  | Customer_Occupation(age)    |
+---------+------+-----------------------------+
| Bob     |   25 | Actor                       |
| Suzi    |   26 | Actor                       |
| John    |   28 | Actor                       |
| Joseph  |   30 | Engineer                    |
| Peter   |   32 | Engineer                    |
| Stephen |   45 | Scientist                   |
+---------+------+-----------------------------+
```

# Loop

- Similar to other programming languages MySQL provides support for the flow control statements such as IF, CASE, ITERATE, LEAVE LOOP, WHILE, and REPEAT.
- You can use these statements in the stored programs (procedures), and RETURN in stored functions. You can use one Flow Control Statement with in another.
- The LOOP is a compound MySQL statement which is used to execute a single or set of statements repeatedly.

# Cursor

- A cursor in database is a construct which allows you to iterate/traversal the records of a table. In MySQL you can use cursors with in a stored program such as procedures, functions etc.
- In other words, you can iterate though the records of a table from a MySQL stored program using the cursors.

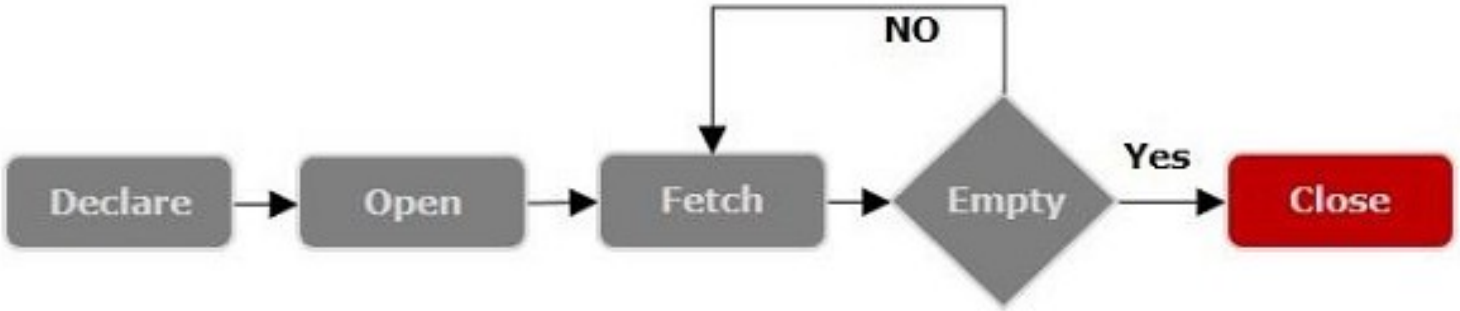
# Cursor

- The cursors provided by MySQL are embedded cursors.
  - READ ONLY – Using these cursors you cannot update any table.
  - Non-Scrollable – Using these cursors you can retrieve records from a table in one direction i.e., from top to bottom.
  - Asensitive – These cursors are insensitive to the changes that are made in the table i.e. the modifications done in the table are not reflected in the cursor. Which means if we have created a cursor holding all the records in a table and, meanwhile if we add some more records to the table, these recent changes will not be reflected in the cursor we previously obtained.

# Cursor

- While Declaring cursors in a stored program you need to make sure these (cursor declarations) always follow the variable and condition declarations.
- To use a cursor, you need to follow the steps given below (in the same order)
  - Declare the cursor using the DECLARE Statement.
  - Declare variables and conditions.
  - Open the declared cursor using the OPEN Statement.
  - Retrieve the desired records from a table using the FETCH Statement.
  - Finally close the cursor using the CLOSE statement.

# Cursor



# Declare Cursor

- The stored function is almost similar to the procedure in MySQL, but it has some differences that are as follows:
  - The function parameter may contain only the IN parameter but can't allow specifying this parameter, while the procedure can allow IN, OUT, INOUT parameters.
  - The stored function can return only a single value defined in the function header.
  - The stored function may also be called within SQL statements.
  - It may not produce a result set.

# Declare Cursor

- Using the DECLARE statement you can declare a cursor and associate It with the SELECT statement which fetches the desired records from a table.
- This SELECT statement associated with a cursor does not allow INTO clause.
- Once you declare a cursor you can retrieve records from it using the FETCH statement.
- You need to make sure the cursor declaration precedes handler declarations. You can create use cursors in a single stored program.
- Syntax:
  - DECLARE cursor_name CURSOR FOR select_statement;

# Open Cursor

- After declaring the cursor the next step is to open the cursor using open statement.
- Syntax:
  - open cursor_name;
- Parameter:
  - cursor_name: name of the cursor which is already declared.

# Fetch Cursor

- After declaring and opening the cursor, the next step is to fetch the cursor. It is used to fetch the row or the column.
- Syntax:
  - FETCH [ NEXT [ FROM ] ] cursor_name INTO variable_list;
- Parameter:
  - cursor_name: name of the cursor
  - variable_list: variables, comma separated, etc. is stored in a cursor for the result set

# Close Cursor

- The final step is to close the cursor.
- Syntax:
  - close cursor_name;
- Parameter:
  - cursor_name: name of the cursor

# Example: Create a table

- Assume we have created a table with name tutorials in MySQL database using CREATE statement as shown below –

```
CREATE TABLE lecture (
  ID INT PRIMARY KEY,
  TITLE VARCHAR(100),
  AUTHOR VARCHAR(40),
  DATE VARCHAR(40)
);
```

# Example: Insert some values

- insert into lecture values(1, 'Java', 'Krishna', '2019-09-01');
- insert into lecture values(2, 'JFreeCharts', 'Satish', '2019-05-01');
- insert into lecture values(3, 'JavaSprings', 'Amit', '2019-05-01');
- insert into lecture values(4, 'Android', 'Ram', '2019-03-01');
- insert into lecture values(5, 'Cassandra', 'Pruthvi', '2019-04-06');

# Example: Create table for backup

- Let us create another table to back up the data –

```
CREATE TABLE backup (
  ID INT,
  TITLE VARCHAR(100),
  AUTHOR VARCHAR(40),
  DATE VARCHAR(40)
);
```

# Create cursor function

```
DELIMITER &&
CREATE PROCEDURE ExampleProc()
  BEGIN
    DECLARE done INT DEFAULT 0;
    DECLARE lectureID INTEGER;
    DECLARE lectureTitle, lectureAuthor, lectureDate VARCHAR(20);
    DECLARE cur CURSOR FOR SELECT * FROM lecture;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
    OPEN cur;
    label: LOOP
    FETCH cur INTO lectureID, lectureTitle, lectureAuthor, lectureDate;
    INSERT INTO backup VALUES(lectureID, lectureTitle, lectureAuthor, lectureDate);
    IF done = 1 THEN LEAVE label;
    END IF;
    END LOOP;
    CLOSE cur;
  END&&
DELIMITER ;
```

# Check the output

- You can call the above procedure as shown below –

```
mysql> CALL ExampleProc;
```

- If you verify the contents of the backup table you can see the inserted records as shown below –

```
mysql> select * from backup;
```

# Trigger

- A trigger in MySQL is a set of SQL statements that reside in a system catalog.
- 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.

# Trigger

- A trigger is called a special procedure because it cannot be called directly like a stored procedure.
- The main difference between the trigger and procedure is that a trigger is called automatically when a data modification event is made against a table.
- In contrast, a stored procedure must be called explicitly.

# Trigger : Types

- Generally, 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.

# Trigger : Why?

- 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.

# Trigger : Limitations

- 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.

# Trigger : Where?

- We can define the maximum six types of actions or events in the form of triggers:
- 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.

# Naming Conventions

- Naming conventions are the set of rules that we follow to give appropriate unique names. It saves our time to keep the work organize and understandable.
- Therefore, we must use a unique name for each trigger associated with a table. However, it is a good practice to have the same trigger name defined for different tables.
- The following naming convention should be used to name the trigger in MySQL:
  - (BEFORE | AFTER) table_name (INSERT | UPDATE | DELETE)

    Thus,
  - Trigger Activation Time: BEFORE | AFTER
  - Trigger Event: INSERT | UPDATE | DELETE

# How to create trigger?

- We can use the CREATE TRIGGER statement for creating a new trigger in MySQL. Below is the syntax of creating a trigger in MySQL:

```
CREATE TRIGGER trigger_name
    (AFTER | BEFORE) (INSERT | UPDATE | DELETE)
        ON table_name FOR EACH ROW
        BEGIN
       --variable declarations
       --trigger code
       END;
```

# How to create trigger?

- trigger_name:
  - It is the name of the trigger that we want to create. It must be written after the CREATE TRIGGER statement. It is to make sure that the trigger name should be unique within the schema.
- trigger_time:
  - It is the trigger action time, which should be either BEFORE or AFTER. It is the required parameter while defining a trigger.
  - It indicates that the trigger will be invoked before or after each row modification occurs on the table.

# How to create trigger?

- trigger_event:
  - It is the type of operation name that activates the trigger. It can be either INSERT, UPDATE, or DELETE operation.
  - The trigger can invoke only one event at one time. If we want to define a trigger which is invoked by multiple events, it is required to define multiple triggers, and one for each event.
- table_name:
  - It is the name of the table to which the trigger is associated. It must be written after the ON keyword. If we did not specify the table name, a trigger would not exist.

# How to create trigger?

- The trigger body can access the column's values, which are affected by the DML statement. The NEW and OLD modifiers are used to distinguish the column values BEFORE and AFTER the execution of the DML statement.
- We can use the column name with NEW and OLD modifiers as OLD.col_name and NEW.col_name.
- The OLD.column_name indicates the column of an existing row before the updation or deletion occurs.
- NEW.col_name indicates the column of a new row that will be inserted or an existing row after it is updated.

# How to create trigger?

- For example, suppose we want to update the column name message_info using the trigger.
- In the trigger body, we can access the column value before the update as OLD.message_info and the new value NEW.message_info.
- We can understand the availability of OLD and NEW modifiers with the below table:

| Trigger Event | OLD | NEW |
|---|---|---|
| INSERT | No | Yes |
| UPDATE | Yes | Yes |
| DELETE | Yes | No |

# Example:

- Let us start creating a trigger in MySQL that makes modifications in the employee table.
- First, we will create a new table named employee by executing the below statement:

```
CREATE TABLE employee(
   name varchar(45) NOT NULL,
   occupation varchar(35) NOT NULL,
   working_date date,
   working_hours varchar(10)
);
```

# Example:

- Next, execute the below statement to fill the records into the employee table:
- ```
  INSERT INTO employee VALUES
  ('Robin', 'Scientist', '2020-10-04', 12),
  ('Warner', 'Engineer', '2020-10-04', 10),
  ('Peter', 'Actor', '2020-10-04', 13),
  ('Marco', 'Doctor', '2020-10-04', 14),
  ('Brayden', 'Teacher', '2020-10-04', 12),
  ('Antonio', 'Business', '2020-10-04', 11);
  ```
  ```

# Example:

- Next, we will create a BEFORE INSERT trigger. This trigger is invoked automatically insert the working_hours = 0 if someone tries to insert working_hours < 0.

```
mysql> DELIMITER //
mysql> Create Trigger before_insert_empworkinghours

BEFORE INSERT ON employee FOR EACH ROW
BEGIN
IF NEW.working_hours < 0 THEN SET NEW.working_hours
= 0;
END IF;
END //
```

# Example:

- Now, we can use the following statements to invoke this trigger:

```
mysql> INSERT INTO employee VALUES
('Markus', 'Former', '2020-10-08', 14);

mysql> INSERT INTO employee VALUES
('Alexander', 'Actor', '2020-10-012', -
13);
```

# After Insert

- After Insert Trigger in MySQL is invoked automatically whenever an insert event occurs on the table.
- Syntax:

```
CREATE TRIGGER trigger_name
AFTER INSERT
ON table_name FOR EACH ROW
trigger_body ;
```

# After Insert

- If we want to execute multiple statements, we will use the BEGIN END block that contains a set of SQL queries to define the logic for the trigger. See the below syntax:

```
DELIMITER $$
CREATE TRIGGER trigger_name AFTER INSERT
ON table_name FOR EACH ROW
BEGIN
  variable declarations
  trigger code
END$$
DELIMITER ;
```

# After Insert

- We can access the NEW values but cannot change them in an AFTER INSERT trigger.
- We cannot access the OLD If we try to access the OLD values, we will get an error because there is no OLD on the INSERT trigger.
- We cannot create the AFTER INSERT trigger on a VIEW.

# After Insert: Example

- Suppose we have created a table named "student_info" as follows:

```
CREATE TABLE student_info (
  stud_id int NOT NULL,
  stud_code varchar(15) DEFAULT NULL,
  stud_name varchar(35) DEFAULT NULL,
  subject varchar(25) DEFAULT NULL,
  marks int DEFAULT NULL,
  phone varchar(15) DEFAULT NULL,
  PRIMARY KEY (stud_id)
)
```

# After Insert: Example

- Again, we will create a new table named "student_detail" as follows:

```
CREATE TABLE student_detail (
  stud_id int NOT NULL,
  stud_code varchar(15) DEFAULT NULL,
  stud_name varchar(35) DEFAULT NULL,
  subject varchar(25) DEFAULT NULL,
  marks int DEFAULT NULL,
  phone varchar(15) DEFAULT NULL,
  Lasinserted Time,
  PRIMARY KEY (stud_id)
);
```

# After Insert: Example

- Next, we will use a CREATE TRIGGER statement to create an after_insert_details trigger on the student_info table. This trigger will be fired after an insert operation is performed on the table.

```
mysql> DELIMITER //
mysql> Create Trigger after_insert_details
AFTER INSERT ON student_info FOR EACH ROW
BEGIN
INSERT INTO student_detail VALUES (new.stud_id,
new.stud_code,
new.stud_name, new.subject, new.marks, new.phone,
CURTIME());
END //
```

# After Insert: Example

- We can use the following statements to invoke the above-created trigger:

- ```
  mysql> INSERT INTO student_info VALUES
  (10, 110, 'Alexandar', 'Biology', 67,
  '2347346438');
  ```
- The table that has been modified after the update query executes is student_detail.
  ```

# Show triggers

- The show or list trigger is much needed when we have many databases that contain various tables.
- Sometimes we have the same trigger names in many databases; this query plays an important role in that case.
- We can get the trigger information in the database server using the below statement.
- This statement returns all triggers in all databases:

```
mysql> SHOW TRIGGERS;
```

# Drop triggers

- We can drop an existing trigger from the database by using the DROP TRIGGER statement with the below syntax:

- ```
  DROP TRIGGER [IF EXISTS]
  [schema_name.]trigger_name;
  ```

- ```
  mysql> DROP TRIGGER
  employeedb.before_update_salaries;
  ```

# 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**