# The SQL STORED PROCEDURE

**What is a SQL Stored Procedure?**

The **stored procedure** is a prepared SQL query that you can save so that the query can be **reused** over and over again. So, if the user has an SQL query that you write over and over again, keep it as a stored procedure and execute it. Users can also pass parameters to a stored procedure so that the stored procedure can act based on the parameter value that is given.

**Stored Procedure Syntax**

CREATE PROCEDURE procedure\_name

AS

sql\_statement

GO;

**Execute a Stored Procedure**

EXEC procedure\_name;

# SQL VIEWS STATEMENT

In SQL, the view is a virtual table based on the result-set of an SQL statement. A view holds rows and columns, similar to a real table. The fields in a view are fields from one or more real tables in the database. You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

# CREATE VIEW Syntax

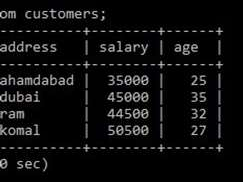
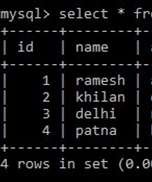
* CREATEVIEW view\_name AS SELECT column1,column2,...

FROM table\_name

WHERE condition;

**Note**:A view always shows up-to-date data! The database engine recreates the data, using the view's SQL statement, every time a user queries a view

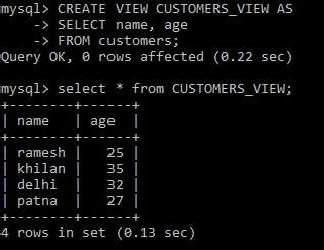
# Create a table customer



Create a view on the table **customers**.Here, the view would be used to have a customer name and age from the **customers** table.

* CREATE VIEW CUSTOMERS\_VIEW AS

SELECT name, age FROM customers;



# The WITH CHECK OPTION

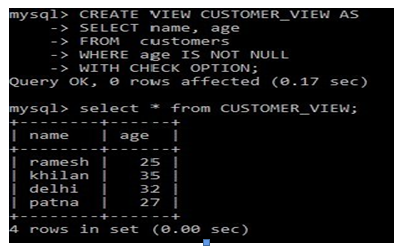
* The **WITH CHECK OPTION** in SQL is a CREATE VIEW statement option. The objective of the WITH CHECK OPTION is to make sure that all UPDATE and INSERTs satisfy the condition(s) in the view definition.
* The following code block has an example of creating the same view

CUSTOMERS\_VIEW with the WITH CHECK OPTION.

CREATE VIEW CUSTOMER\_VIEW AS

SELECT name, age

FROM customers WHERE age IS NOT NULL WITH CHECK OPTION;



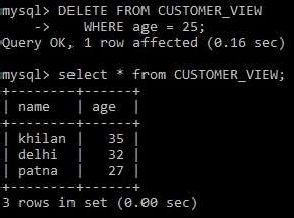
Here we have created a view(CUSTOMER\_VIEW) **with the check** option.

# DELETING ROWS INTO A VIEW

Rows of data can be deleted from a view. The same rules that apply to the UPDATE and INSERT commands apply to the DELETE command.

**Example** Delete a record having AGE = 25.

* DELETE FROM CUSTOMER\_VIEW WHERE age = 25;



* Here we have deleted the row, which contains the age = 25.

# DROPPING VIEWS

# Where the user has a view, you need a method to drop the view if it is no longer needed. The query is straightforward and is given below:

# DROP VIEW view\_name;

# 

It’s similar to the other dropping option, as we have done yet for tables. As

we can see, the view is not available in the database after dropping the

# STORED PROCEDURE AND FUNCTIONS:

Advance MySQL provides better understanding for Stored Procedure,

View, Triggers, Events

and Indexes. In this chapter, we

are going to

understand all of the above terminology one by one in details with the help of MySQL workbench.

**What is a SQL Stored Procedure?**

The **stored procedure** is a prepared SQL query that you can save so that the query can be **reused** over and over again. So, if the user has an SQL query that you write over and over again, keep it as a stored procedure and execute it. Users can also pass parameters to a stored procedure so that the stored procedure can act based on the parameter value that is given.

Syntax for creating a Stored Procedure

DELIMITER $$

CREATE PROCEDURE PROCEDURE\_NAME() BEGIN

SELECT Column\_name1, Column\_name2,………..

FROM Table\_name

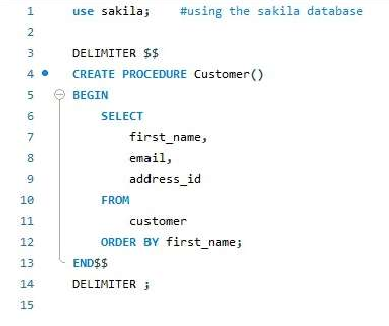
END$$ DELIMITER

Here, the DELIMITER is not the part of Query, the first Delimiter is change the default delimiter to **//** and the second delimiter is change the delimiter to

the default. The Stored while creation.

procedure is saved automatically in the database

To execute the query in MySQL, use the MySQL workbench for better user-interface, and use inbuilt databases to perform the advance MySQL queries.

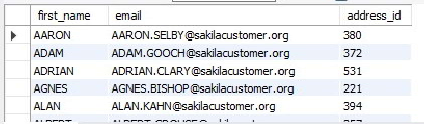


Here we have create a procedure called Customer, and we have mentioned few column names in it. And in last we have closed the procdure. If we want to know the output of the above query, then need to run the procedure by clicking on the execution button on workbench display.

# EXECUTION OF STORE PROCEDURE

Execution of the Stored Procedure is very simple by using the **CALL procedure\_name,** Execute the below query to get the result of the defined stored procedure.

After calling the procedure, we are able to see the selected clumns which are mentioned in the procedure. The output is as follow:

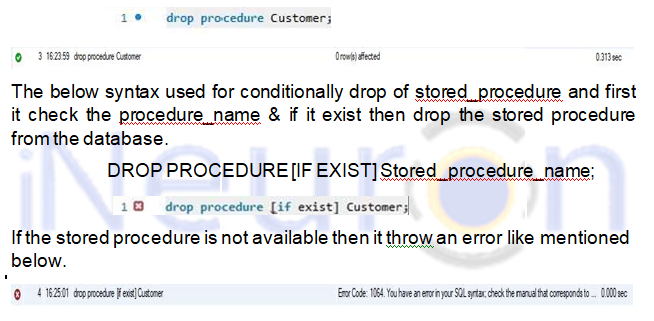


Stored Procedue can have parameters,so while execution we can pass the argument and get the result. We can use control flow (like: IF, LOOP,CASE, etc.)in the stored procedure to make dynamic queries and also we can pass one stored procedure inside the other which will help to modulize the queries.

# DROP THE STORED PROCEDURE:

Drop procedure use to delete the stored procedure from the databases. The following query used to delete the stored procedure for the Database:

* DROP Stored\_procedure\_name



# STORED PROCEDURE PARAMETERS

We can create a stored procedure with the parameters In Stored the procedure are like IN, OUT and INPUT. The parameters make the

Stored Procedure more flexible and useful.

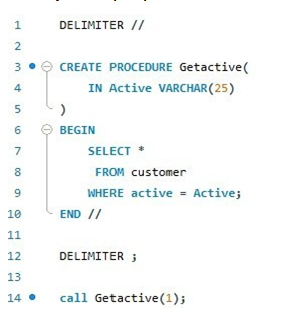
# DEFINING A PARAMETERS

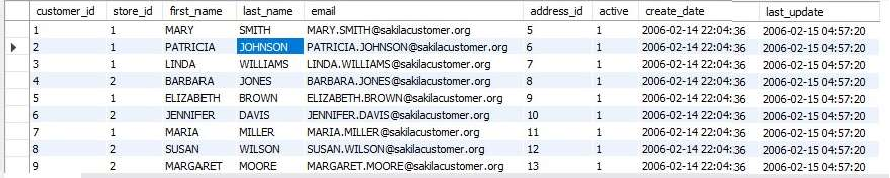
To define the parameter inside the stored procedure, run the below query: [ IN | OUT | INPUT ] PARAMETER\_NAME datatype[(length)]

# IN Parameter

It is the default parameter in Stored Procedure and the calling program should pass an argument to stored Procedure. The value of IN is protected that means even the IN value is changed inside the stored procedure the original value will retained after end of the Stored Procedure.

Example for IN: Create a Stored Procedure that find all the active customers by the input parameter as Active



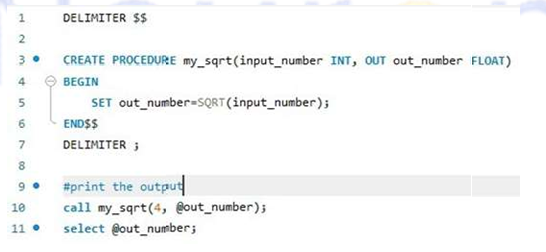
Output:

Out Parameter

The value of the Output Parameter can be changed inside the Store Procedure and pass the new value while calling the Stored Procedure.

Example for OUT:

write a stored procedure to print the square root of a number.



**Output:**

The my\_sqrt stored procedure has two parameters.

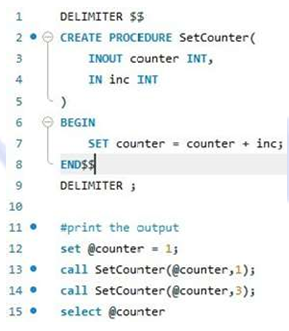
**input\_number** : it takes input from user in interger format. **out\_number** : it store the output of the function.

To print the output, we have call the my\_sqrt() and pass the arguments, the first argument is user input and second is output, and to show the output use select @out\_number.

**INOUT**

It is the combination of the IN and OUT Parameters.

Example for INOUT : Here we are just counting the numbers between a region using INOUT stored procedure.



**Output:**

It will print the query in sequence so if we call the stored

procedure for many time,so it will count all the sum and in last it will print the counter values.

# STORED PROCEDURE VARIABLES

In this unit we will learn about the variables, and also how to declare variables? How to use the variables?. Basically a variable is a called as data object whose value can be change while execution of Stored Procedure.

**DECLARING THE VARIABLE**

To declare the variable inside a stored procedure, use the below query:

DECLARE Variable\_name datatype(size) [DEFAULT Default\_value];

Here,

DECLARE – It is a keyword and it is use to declare the variable. First write DECLARE keyword and then variable\_name.

Datatype(size) – it us use to define the variable datatype (like: IN,

Varchar, or char)and size use to define the length of the variable.

Default – it assign variable with default value option. If we declare the variable without specifying any default values, then it’s values will be NULL.

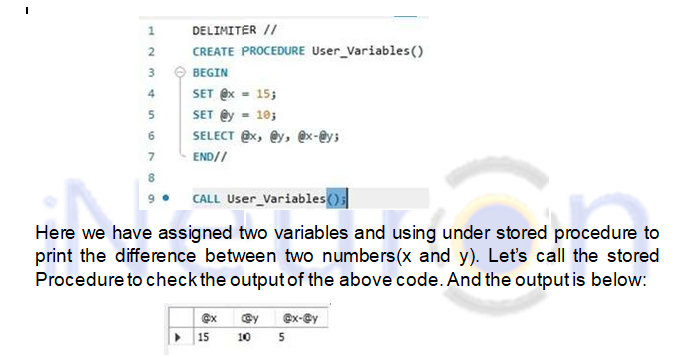
Using MySQL Stored Procedure, we can declare more than one variable.



**ASSIGNING VARIABLE**

Once we declare the variable, now it is ready to use. To assign a value to the variable use **SET** statement:

The value of the variable Total is assigned as 10. We can declare and create the variable in stored procedure as follow:



# VARIABLE SCOPE

# Variable scope is for limited time period. It is defined inside the stored procedure within BEGIN and it will be out of scope once the END statement reaches.

# When we declare any variable inside the BEGIN END Statement, itwill be out of scope once the END statement reached, there after we cannot use it.

# CONDITIONAL STATEMENT

In this unit we will learn about the IF statement in MySQL and we will also learn about how to write the Conditional-Statement in MySQL.

In MySQL, Conditional- Statement has three forms: **IF-THEN**, **IF-THEN-**

**ELSE** and **IF-THEN-ELSEIF-ELSE.** Here we are going to learn about conditional statement in details one by one.

# IF-THEN STATEMENT

**IF-THEN** statement allow user to execute the block of SQL Query based on the specific condition. Here is the syntax for IF-THEN:

IF condition THEN

Statement; END IF;

Here,

First, it will check the condition to execute the statement between the

**IF-THEN** and **END IF** and if the condition is TRUE, otherwise it will go to the next END IF.



In the above query, we are checking the condition as if the subject is as ‘computer’ then set the course as ‘B.Tech’. Let’s check the output of the stored procedure ‘**student\_ifthen’.**



As we can see here, the course is coming as B.Tech, because we set the condition as like that.



**9.2.2. IF-THEN-ELSE STATEMENT**

**IF-THEN-ELSE** is similar to the **IF-THEN.** Where we will execute a block of code on a particular condition and if the condition will not satisfy then it will go for **else** block. The syntax as follow:

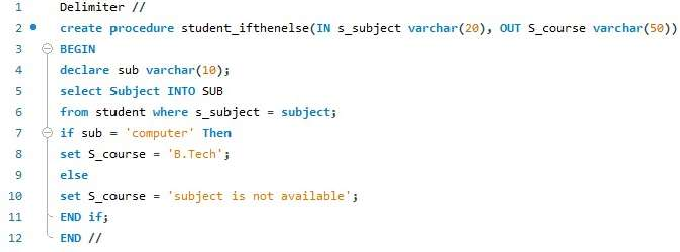
IF Condition THEN;

Statement;

ELSE

Else-Statement;

END IF;



So ler’s call the procedure to check the output of the above query.





As we can see here, we have given the subject name as ‘Hitory’, so that

the else part is executed and given output as the ‘subject is not available’.

# 9.2.3. IF THEN ELSEIF ELSE STATEMENT

**IF-THEN-ELSE** is similar to the **IF-THEN.** Where we will execute a block of code on a particular condition in **IF block** and if the condition will not satisfy

then it will go for **ELSEIF** block and again if the condition is then it will execute the **else** block. The syntax as follow:

IF Condition THEN; IF-Statement;

ELSEIF

ElSEIF-Statement;

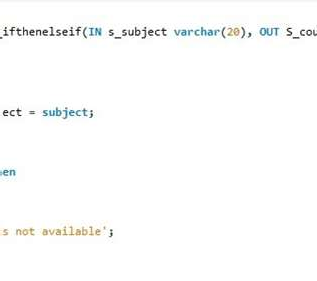
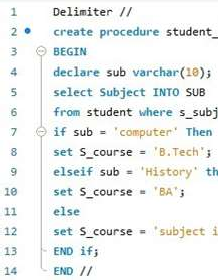
not satisfied

ELSE

ElSE-statement

END IF;

Now we are going to perform the same operation as above, where we are going to check the subject name in **if** block as ‘computer’ and if the condition is not satisfied then the **else-if** part will execute and if the condition is not satisfied then the **else** block will get executed.



So ler’s call the procedure to check the output of the above query.

The else-if block is get executed and print the course name as **BA.**



let’s call the else part,



The conditional statement explained here one-by-one. We saw the three

conditional statements ELSE.

as IF-THEN, IF-THEN-ELSE, IF-THEN-ELSEIF-

# CASE STATEMENT

In this unit, we are going to learn about the **CASE statement**, it is an alternative conditional statement for the IF-Statement and CASE statement makes the code more efficient and readable. CASE statement has two forms: **Simple CASE** and **Searched CASE**.

So, let’s learn about the Procedure.

CASE statement and their use using the Stored

# Simple CASE Statement



The simple CASE statement sequentially compare the case\_values in with the when\_values until it finds as equal. The basic syntax for the Simple CASE statement:

CASE case\_value

When when\_values THEN statement

…..

[ELSE else-statements] END CASE;

If the case\_values will not be equal to the when\_values then it will execute the else statement. And if the else is also not satisfied then it will throw an error as CASE not found for CASE Statement.

To avoid the error when the case\_value will not equal to when\_values then we can use an empty **BEGIN END** block in the else block as follows:

CASE case\_value

When when\_values THEN statement

…..

[ELSE else-statements]

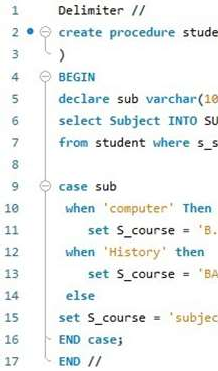
BEGIN END;

END CASE;

Here we are going to do same operation as IF conditional statement. We

are going to check the CASE statement.

subjects are lying under which course using the



Here instead of using IF-ELSE we have used the CASE Statement, under that we are checking the condition as when “condition” is true then set

values. If the condition is not satisfied under the CASE statement then the

ELSE block will get executed.

Let’s run the query and

call the Stored Procedure. Now we

will print the

Case statement as follow:



And also,



Now, Let’s see the else block, if the case\_values are not satis ied then else block will executed.



As we can see, the else part is executed ans showing “subject is not available”.

# Searched CASE Statement



Searched CASE Statement is similar to Simple Case Statement but it only

allows you to compare a value with a set of distinct values. It is equivalent

to the IF Statement but it is more readable than IF Statement. To perform the more complex matches (like ranges), we use the Searched CASE Statement.

The syntax is as follow:

CASE case\_value

When when\_values THEN statement

……..

[ELSE else-statements] END CASE;

The searched **CASE** statement check each **Seach\_condition** inside the WHEN clause until it finds as TRUE then it will execute the corresponding

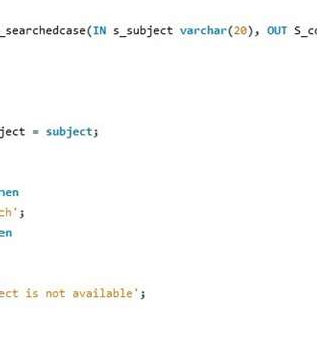
THEN statement. If the **search\_condition** is not satisfied then the **CASE**

evaluates the **ELSE** Statement.

We are going to search the subject names and checking them that under

which course they existing. If the condition is true then it will execute and

return the course name execute the else block.



Here we are searching, course as ‘B.Tech’. And

but if the condition is not satisfied then it will

when the subject name as ‘computer’ then return if the condition is not satisfied then it will execute

the else block and return as ‘subject is not available’.

Let’s call the Stored Procedure & check the satisfied condition



Here we have passed the subject name as ‘computer’ and it B.Tech. Let’s have a look on Output.



Similarly, let’s execute the **else** part:

is belongs in



As we can see here, it executed the else part of the Stored Procedure. And showing output as the ‘subject is not available’.



# CASE Vs IF STATEMENT

IF and CASE are allow you to execute the block of code on specific

condition. We can use both IF and CASE statement inside sto procedure, it’s completely depends on our choice.



* Simple CASE statement is more efficient and readable t Statement while comparing a single expression.

ed

an the IF

* IF statement if better when we are executing complex expressions.
* If we are using CASE statement, then make sure that at least one condition should be satisfied otherwise we need to add error handler inside the stored procedure.

# LOOP STATEMENT

In MySQL, the **LOOP** statement is used to execute one or more than one statement repeatedly.

The syntax for LOOP statement:

[begin loop: ] loop

Statement\_list END LOOP [end\_label]



The loop executes the **statement\_list** repeatedly one by one and the statement\_list can be one or more and separated by the **semicolon (;)**. To terminate the loop we use the **LEAVE** statement after the condition is successfully satisfied.

The syntax for LOOP Statement with LEAVE Statement: [label]: LOOP

IF Condition THEN

LEAVE[label]; END IF;

END LOOP;

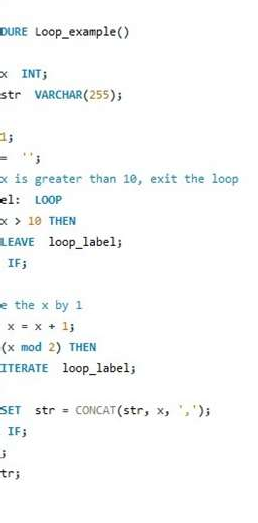
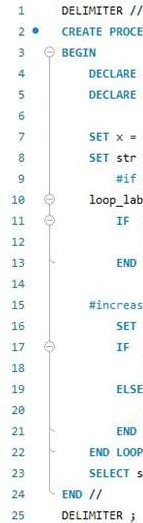
The **LEAVE** statement is exactly work as the **break** in other programming language. It will immediately exit from the loop.

We are going to use the LOOP inside the Stored Procedure to print the even numbers from 1 to 10. To print the even number we need to implement two conditions:

* The number should be less than 10.
* The number should be divisible by 2.

So let’s implement it using SQL query inside the Stored Procedure using the LOOP and IF-THEN Statement. The query is written below to print the even numbers from the 1 to 10.





Let’s call the stored procedure to check the result.

As we can see, the output is the collection of the even numbers between 1 and 10.

# WHILE LOOP STATEMENT

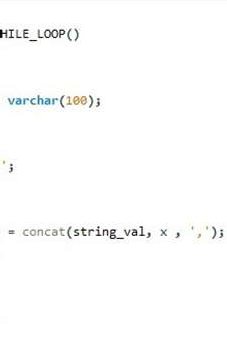
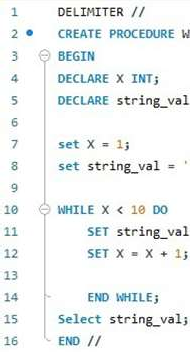
WHILE loop is execute the block of codes until the condition TRUE. The syntax is written below:

[begin loop:] WHILE condition DO Statement

END WHILE [end\_label]

The WHILE statement will check the condition at the beginning of the each iteration and if the condition satisfied, then it execute the Statement until the condition is true. We can have one or more than one statements inside the DO and END WHILE.

We are going to print the numbers from 1 to 10 using while loop inside the stored procedure.



Let’s call the stored procedure and check the output of the above query.



We can see here, the output of the query is the collection of the numbers from 1 to 10 (excluding 10). Because we have given the condition in a while loop as <10, so till that, it will print all the numbers from 1 to till 9.

# REPEAT LOOP STATEMENT

REPEAT statement is used to execute one or more statements until the condition satisfied. It is similar to the DO WHILE LOOP in C. The syntax for the REPEAT loop statement as follows:

REPEAT



Statement

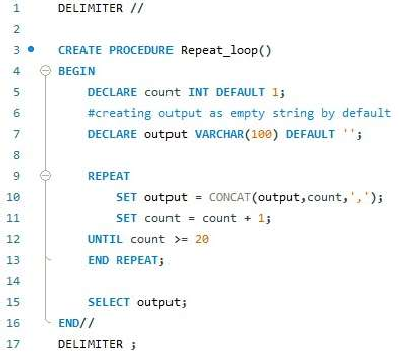
UNTIL

condition

END REPEAT

Here, the REPEAT LOOP executes the statement before checking the condition. Therefore the statement will always execute at least once. It is also known as the post-test loop.

Let’s print the numbers from 1 to 20(excluding 20) by using a REPEAT loop inside the stored procedure.



Let's check the output of the stored procedure by calling it.



We have successfully printed the numbers from 1 to 20 Repeat loop inside the stored procedure.

# CURSOR

by using the

MySQL cursor in Stored Procedure is used to iterate through a result set

returned by a select statement. We use the cursor to handle a result set

inside a stored procedure. A cursor allows you to iterate a set of rows

returned by a select query and process through each row separately. MySQL cursor is read-only, Non-scrollable, and Asensitive.

Syntax to write the cursor in the stored procedure:

1. Declare Cursor\_name CURSOR from SELECT\_Statement.

The cursor is declared after the variable declaration; if you sa it will throw an error.

1. OPEN Cursor\_name.

before, then

We are open the cursor by using the OPEN statement an also OPEN initialize the result set for the cursor.

1. Declare CONTINUE HANDLER FOR NOT FOUND (Termination Statement)

To declare not found a handler, we use the above query. The finished is a variable that indicates the cursor has reached the end of the result list.

1. FETCH cursor name INTO variable\_list.

By using the FETCH statement, retrieve the next row pointed by the cursor and move to the next row in the result list.

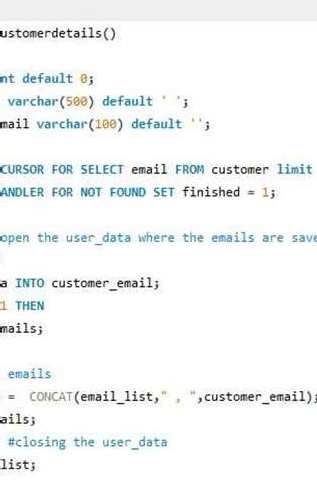
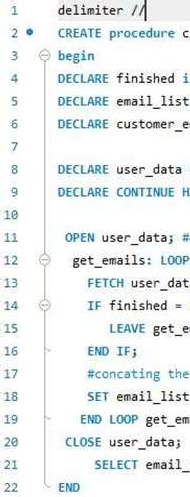
1. Close cursor\_name.

By using the CLOSE, we deactivate the cursor and release associated with it.



Let’s understand cursor by implementing it on the tables. So, we are going to get emails from customers from the customer table. To implement it, we are going to use the above five steps.

the memory



Let’s check the output by calling the stored procedure

In the output set, we have finally extracted the top 5 emails and stored them in the email\_list by using the cursor inside the stored procedure.



# ERROR HANDLING

MySQL ERROR HANDLING uses to encounter Errors in the stored procedure. Whenever any error occurs inside a stored procedure, it is very important to handle it. To handle that, MySQL is providing an easy way to

define handlers that handle the errors (such as warnings or specific conditions).

To declare the handler, we use the following syntax: DECLARE action HANDLER FOR condition statement;

exceptions to

If the condition matches, then the MySQL will execute the statement and continue or exit from the code block based on the action.

Action accepts one of the following values:

**CONTINUE**: the execution of the code block is continuing.



**EXIT**: the execution of declared or terminated.

the enclosing code block, where the handler is

# Declaring the Error Handling for CONTINUE

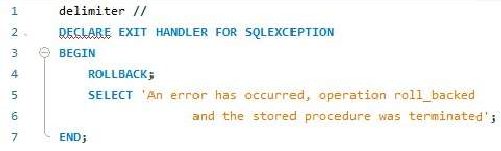
The following handler set the value of error variable equal to 1 and continues the execution if any error occurs;

DECLARE CONTINUE HANDLER FOR SQLEXCEEPTION

SET error\_variable = 1

# Declaring the Error Handling for EXIT

The following handler rolls back the previous operation, issues an error message and exit the current code block in case of error occurs. If we declare it inside the BEGIN END block of a stored procedure, it will terminate the stored procedure immediately.



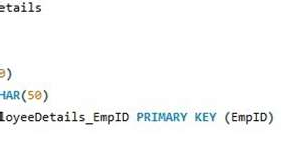
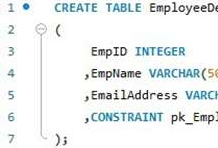
Let’s understand with an

example of the error handler; so we

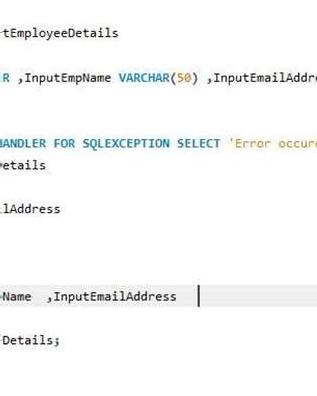
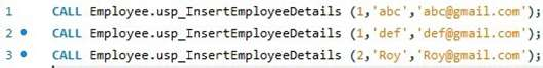
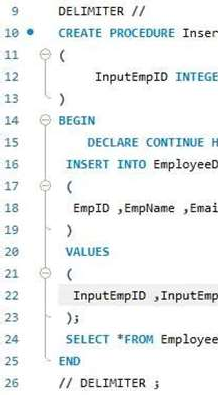
are creating

an employeedetails table and we are trying to pass the (continue and exit).

error handler



We have created a table with EmployeeDetails and given a constraint as pk\_EmployeeDetails with primary key (EmpID). Let’s create a procedure to handle to continue error handler.

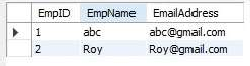


Here we have created a procedure, that will detect the duplicasy in data and throw an error message as error occurred. We are going to pass the values inside the table.



It is giving this message because one error is occurred while inserting the

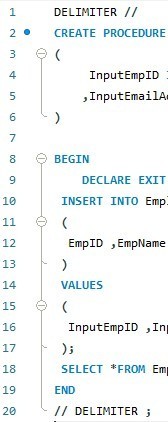
data into the table, and the handler is found the duplicate data. But along with this, it will print the remaining values which will not have any duplicate data.



We have inserted two rows successfully and third is not inserted because

of duplicacy of data. As we have seen the CONTINUE error handler in the

above example. Now let’s do an example with EXIT error handler using stored procedure. We are going to use the same table and same procedure except the error handler. In the below example, we are going to perform the error handler using the exit error handler, which is use to do the execution of the enclosing code block, where the handler is declared or terminated.



Here we are inserting the two rows as follow:

But the result of the error handler using the EXIT error handler will print the error message alone. It will not show the values which are inserted inside the table.



When we call the stored procedur in EXIT handler, it will just give the error message as above.

# ERROR CONDITIONS WITH MySQL SIGNAL/ RESIGNAL STATEMENT

**SIGNAL STATEMENT**

Signal statement is used to return an error or warning conditions to the caller using the stored procedure. It is provide and easy way to get the message and the values based on our need.

The syntax for SIGNAL statement:

SIGNAL SQLSTATE | condition\_name;

SET condition\_information\_item\_name = value1 SET condition\_information\_item\_name = value2

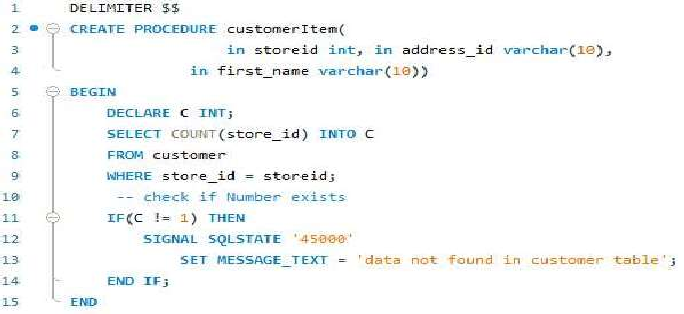
SIGNAL keyword is a SQLSTATE values or condition name declare by using DECLARE CONDITION. The condition\_information\_item\_name can beMESSAGE\_TEXT, MYSQL\_ERROR, CURSOR\_NAME etc.

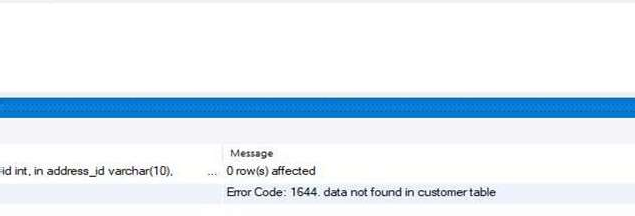
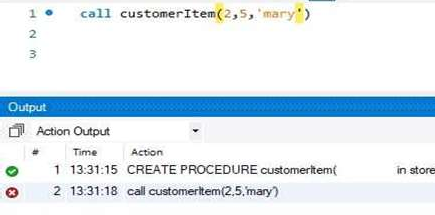


Let’s create a stored procedure with customer table, where we are going to check the store\_id and customer details. If the record is not found then it will give an error message as the “data not found in custmer table”.

Note: **45000** is a generic **SQLSTATE** value that illustrates n unhandled user-defined exception.

a





Output:

As we can see here, the the customer table”.

error message is giving as the “data not found in

# RESIGNAL STATEMENT

RESIGNAL STATEMENT

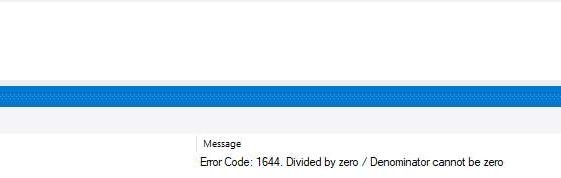
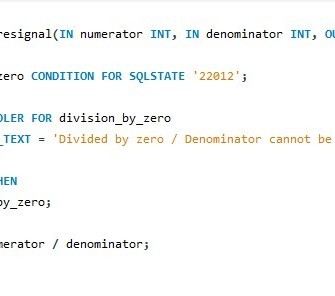
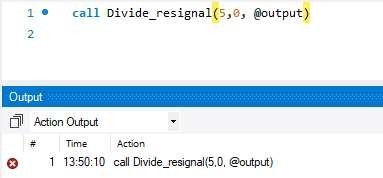
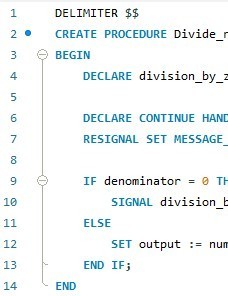
is used to raise a warning or error

messages. It

is similar to the SIGNAL statement in terms of functionality and the syntax,except:

* We must have to use RESIGNAL statement inside the error handler, otherwise it will throw an error as “RESIGNAL when handler is not active”.
* It can omit all the attributes of RESIGNAL statement, even the SQLSTATE values.

We are creating a stored procedure to division of a number with any number, but if any number is divided by zero then it will throw an error message as the “divided by zero”.



Let’s call the stored procedure to check the output, when we divide any number with zero.

Here, we are dividing a number with zero, so it’s throwing an error message as “divided by zero/ denominator can not be zero”.

# STORED FUNCTION

STORED FUNCTION is a special type of the store program where we store

the functions to encapsulate the common formulas and rule, reusable.

# DECLARING STORED FUNCTION

and that are

DELIMITER $$



CREATE FUNCTION function\_name( param1,

param2,…)

RETURNS datatype [NOT] DETERMINISTIC

BEGIN

statements END $$ DELIMITER ;

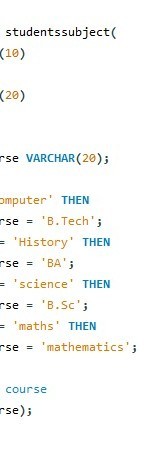
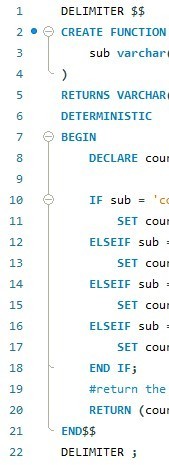
The syntax is easy to understand. It is similar to other programming languages. First we use to create the function with the CREATE FUNCTION function\_name (pass the parameters in it). RETURN type is used to return the values/ statement. Then specify the function as DETERNMINISTIC OR NON-DETERMINISTIC. MySQL uses NON-

DETERMINISTIC by default.

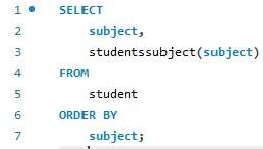
Now we are going to create a stored function on the student table. We are

going to define if the subject\_name is coming then it should respective course\_name.

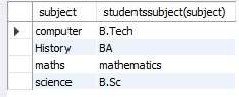
print as their



Let’s check the function output by calling it under the SELECT statement.



We have selected the column as subject and applying the function at the subject. Let’s check the output of the function.



As we can see, the function is giving the subject with their respective courses.

# DROP FUNCTION

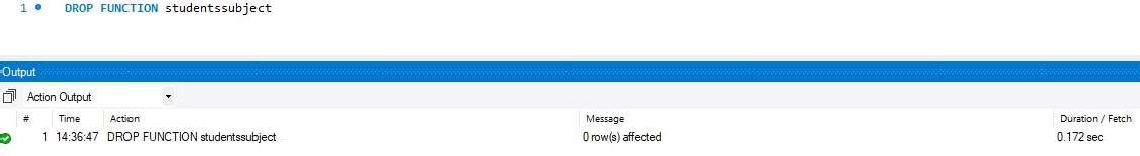


The DROP FUNCTION is use to drop the created stored function from the database. By using the Syntax as:

DROP FUNCTION function\_name;

If the FUNCTION is not exit, In that case check the condition as: DROP FUNCTION IF EXISTS function\_name;

Let’s do an example to show case the above function. For that, we are going to drop the studentssubject() function.



Now, let’s check the second command to drop the table.



It throwing a warning message as function does not exists. That means the function is already deleted.

# TRIGGERS



Trigger is a stored program that invoked automatically in response to an event such as insert, delete or update that occurs in the table. Suppose,

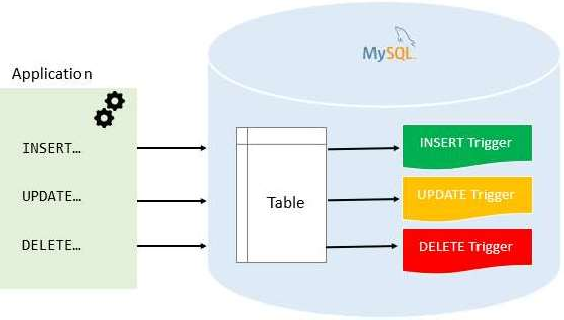
you defined a trigger and you insert a row inside the tabl automatically invoked before or after the insertion of row.

There are two types of the TRIGGERS:

, then it will

1. **Row-level-Triggers**: it is activated for each row that is inserted, deleted or updated.
2. **Statement-level-Triggers**: it is executed for each transaction.

**Note**: It supports only **Row-level-Triggers**.



# Advantage of Triggers



* It provides a way to check the integrity in data.
* It can be useful for auditing the data changes in tables
* It handles the errors from the database layer.
  1. **CREATING TRIGGERS**

CREATE TRIGGER statement is used to create the triggers. The syntax is following:

CREATE TRIGGER trigger\_name

{BEFORE | AFTER} {INSERT | DELETE | UPDATE }

ON table\_name FOR EACH ROW trigger\_body;

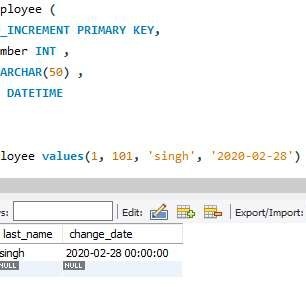
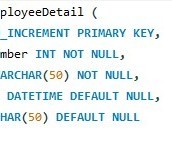
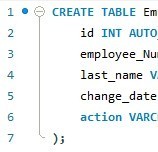
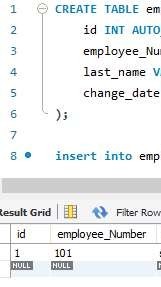
Here,

* The first line is for creating the trigger with trigger\_name.
* It will make the condition that the trigger invokes b any modification in row.

fore or after

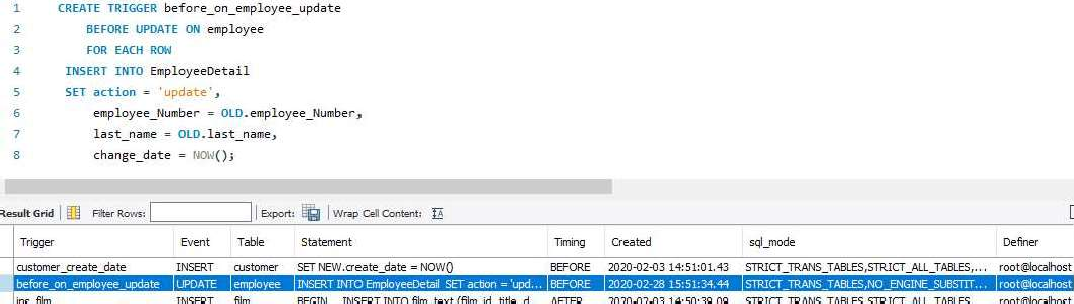
* The operation we can choose as INSERT, DELETE OR UPDATE on the table\_name at any row.

Let’s understand the triggers by using an example. Now, we are going to create a table names as EmployeeDetail. And defined a primary key as id;



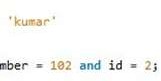
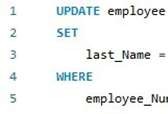
Create another table and insert some rows into that table, now we are going to create one more as “employee”. On the “employee” table, we are going to perform all the trigger operation on it and the operations log will be stored in “EmployeeDetail”.

Let’s create a trigger using the before update operation on employee table. As we can see, the trigger is created as name “before\_on\_employee\_update”.

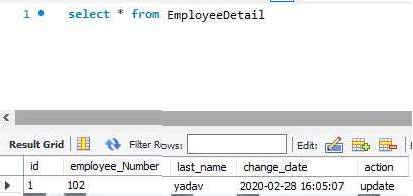


Now, let’s use the trigger and update the row values of employee,

Let’s check the table EmployeeDetail and check the action on it.



As we can see, the trigger is automatically invoked and inserted a new row inside the EmployeeDetail table and the row is updated.



# DROP TRIGGER

To delete the TRIGGER we use the DROP TRIGGER statement, and it will delete the trigger from the database. The syntax is as follow:

DROP TRIGGER [IF EXISTS] trigger\_name;

Here,

OR

* Firstly, it will check the triggers\_name and if it exist then delete that particular trigger.
* To delete any trigger, the trigger\_name should be written after the DROP TRIGGER.

DROP TRIGGER trigger\_name;

It will delete the trigger without checking their existence in the

database.



The trigger **before\_on\_employee\_update** has been deleted from the database.

# BEFORE INSERT TRIGGER

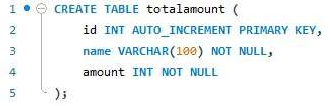
The before insert trigger are automatically fired before an insert occurs on the table. The syntax for before insert trigger as follow:

CREATE TRIGGER trigger\_name BEFORE INSERT

ON table\_name FOR EACH ROW trigger\_body;

let’s understand through an example. We are creating a table as

# totalamount;

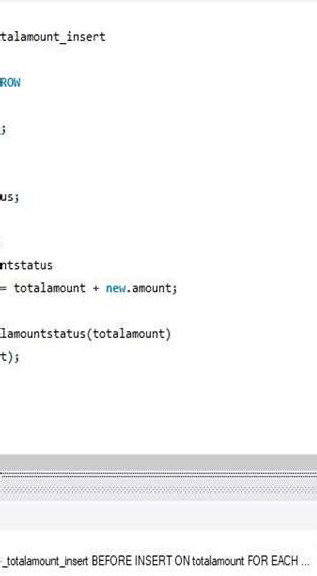
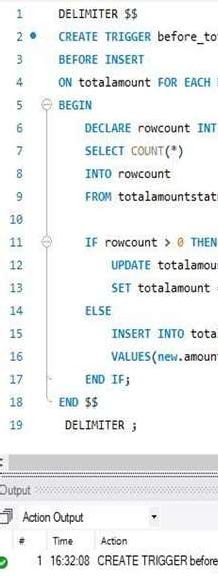


Now, let’s create another table as **totalamountstatus** to store the summary of the triggers.





Lets create a before insert trigger to get the totalamount in the **totalamountstatus** table before a new work center is inserted into the **totalamount** table.



The trigger is created successfully for updating before insert into the

**totalamount** table. Let’s test the trigger by inserting the value in it.



We have successfully insderted the value in the totalamount table. But the value is invoked in the totalamountstatus table. Let’s call the **totalamountstatus** table to check the total amount.



The trigger is invoked and inserted a new row into the totalamountstatus. If we insert another value that will automatically added into the present amount and return the totalamount.

# AFTER INSERT TRIGGER

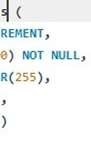
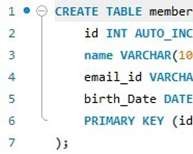
The after insert trigger are automatically fired after an insert occurs on the table. The syntax for after insert trigger as follow:

CREATE TRIGGER trigger\_name AFTER INSERT

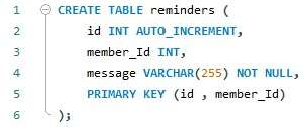
ON table\_name FOR EACH ROW

trigger\_body

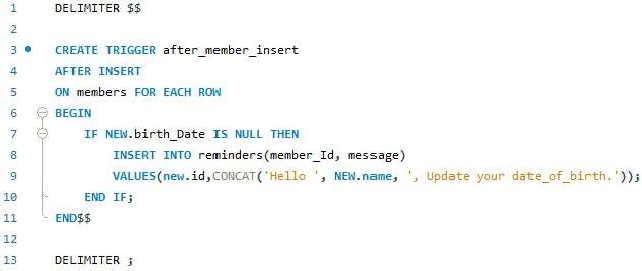
Let’s understand the after insert trigger using an example; Create a table named as **members**.



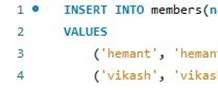
Create another table as **remembers.**



Now, create a after insert trigger as **after\_members\_insert** and that trigger insert into reminders table if the birth\_date of any person is null.



Let’s test the alter insert trigger.



We have inserted the two rows inside the members table and the members table is shown below;

As we can see here, the two rows are inserted but the birthdate of Hemant is null and as we mentioned the condition in trigger, it will invoke a message if birth date is as null. Let’s check the **reminders** table.



As we have made the condition inside the trigger, it has invoked automatically when the birth day found as null. And the message showing as Hello Hemant, update your date\_of\_birth.

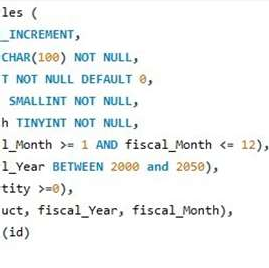
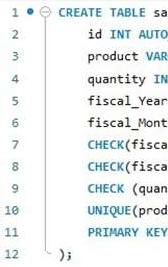
# BEFORE UPDATE TRIGGER

The BEFORE UPDATE TRIGGER is invoked automatically before an update event occurs on the table which associated with the trigger.

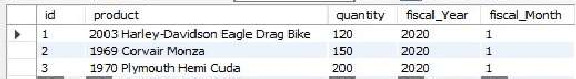
CREATE TRIGGER trigger\_name BEFORE UPDATE

ON table\_name FOR EACH ROW trigger\_body

Let’s understand through an example; Create a table as sales;

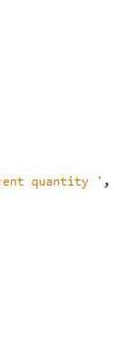
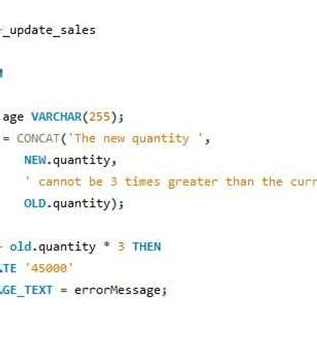
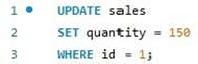
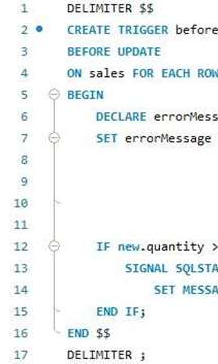


Insert few rows into the sales table;



Creating the BEFORE UPDATE TRIGGER, and assigning the error

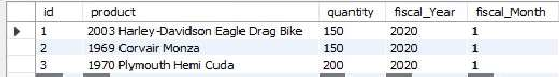
message as the new quantity cannot be greater than 3-times of previous.



The trigger will automatically invoke and fire before updating any values in any row.

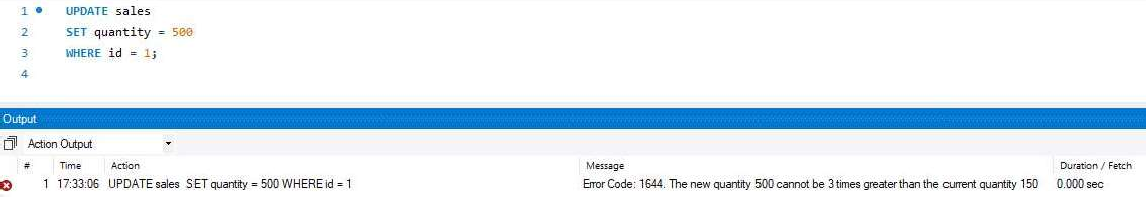
Let’s update the values in row of sales table;

We have updated a value of quantity where the id = 1 but it will not satisfied the condition so it will not give the error message, see the table;



Let’s update the quantity than the quantity150.

as some other value which are 3-times greater



As we have increased the quantity as 3-times higher than previous, it’s

showing message as “the new quantity cannot be 3times greater than the current quantity”.



**10.6.**

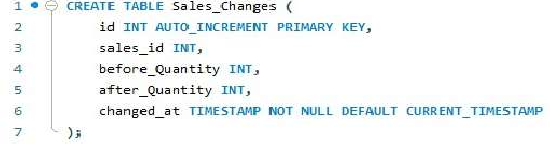
**AFTER UPDATE TRIGGER**

The ALTER UPDATE TRIGGER invoke automatically after updating the events in the associated table. The syntax for AFTER update triggers as follow:

CREATE TRIGGER trigger\_name AFTER UPDATE

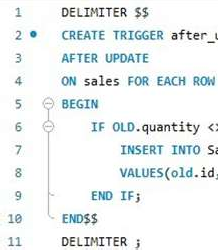
ON table\_name FOR EACH ROW trigger\_body

Let’s understand the after update trigger with an example; we are going to use the first table as **sales** table and the second table as **sales\_changes.** So let’s create the second table sales\_changes.

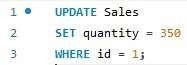


Now, let’s create the AFTER UPDATE TRIGGER;

The after\_update\_sales trigger automatically invoked after updating any row of the **sales** table.



Updating the quantity column in sales table, where id = 1



Let’s check the **sales\_changes** table;



As we can see the value is updated automatically in the sales\_changes.

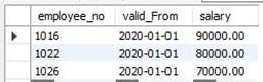
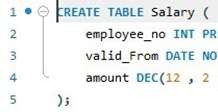
# BEFORE DELETE TRIGGER

The BEFORE DELETE TRIGGER are fired automatically before a delete event occurs in table. The syntax for before delete trigger as follow:

TRIGGER trigger\_name BEFORE DELETE

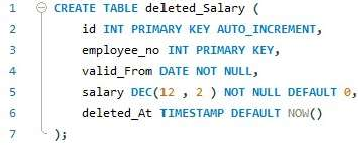
ON table\_name FOR EACH ROW trigger\_body

Let’s create a table as **salary;**

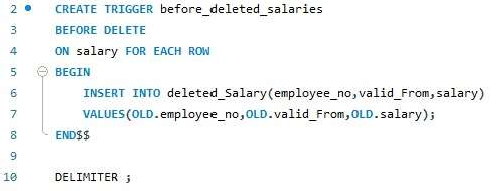


Insert few rows into salary table;

Create another table as deleted\_salary to store the deleted salaries;



Now let’s create a stored procedure, which contains the before delete triggers. Before delete trigger store the deleted value into the deleted\_salary table.



Let’s delete a row from the **salary** table;



Now, check the deleted\_Salary table to check whether the data is stored or



not.

As we can see here, the BEFORE DELETE TRIGGER is automatically invoked the row before event occurs on the **salary** table.

# AFTER DELETE TRIGGER

AFTER DELETE TRIGGERS are invoke automatically after deleting the

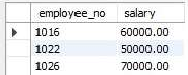
event occurs on the table. The syntax for AFTER DELETE TRIGGERS as follow:

CREATE TRIGGER trigger\_name AFTER DELETE

ON table\_name FOR EACH ROW trigger\_body;

Create a table **salary** and insert few rows into the table;



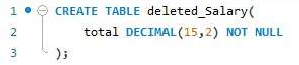


Create another table to

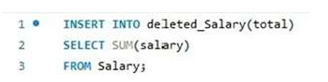
store the deleted row into that,we

are creating

another table as deleted\_salary;



Now,let’s store the value of total into the deleted\_salary table by using the below command. Here,we are using the SUM() function to add the salaries from the salary table and store it into the deleted\_salary as total.

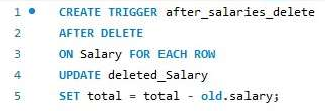


So the total amount is 180000.

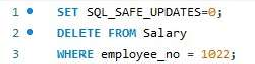


Now, Let’s create AFTER DELETE TRIGGER;

We are creating a trigger which update the total salary into the deleted\_Salary table after deleting from the salary table.



Let’s delete a row where the employee\_no = 1022 inside the salary table;



Check the deleted\_Salary;



As we can the value of total is decresed by 50000, because it is

substracted from the total amount.