ER Model of Online Pizza Ordering Database

GROUP MEMBERS

K.Nandhini U19CS078
Ann Mary Eldo U19CS086
Badavathu Manasa U19CS098

PROJECT DESCRIPTION

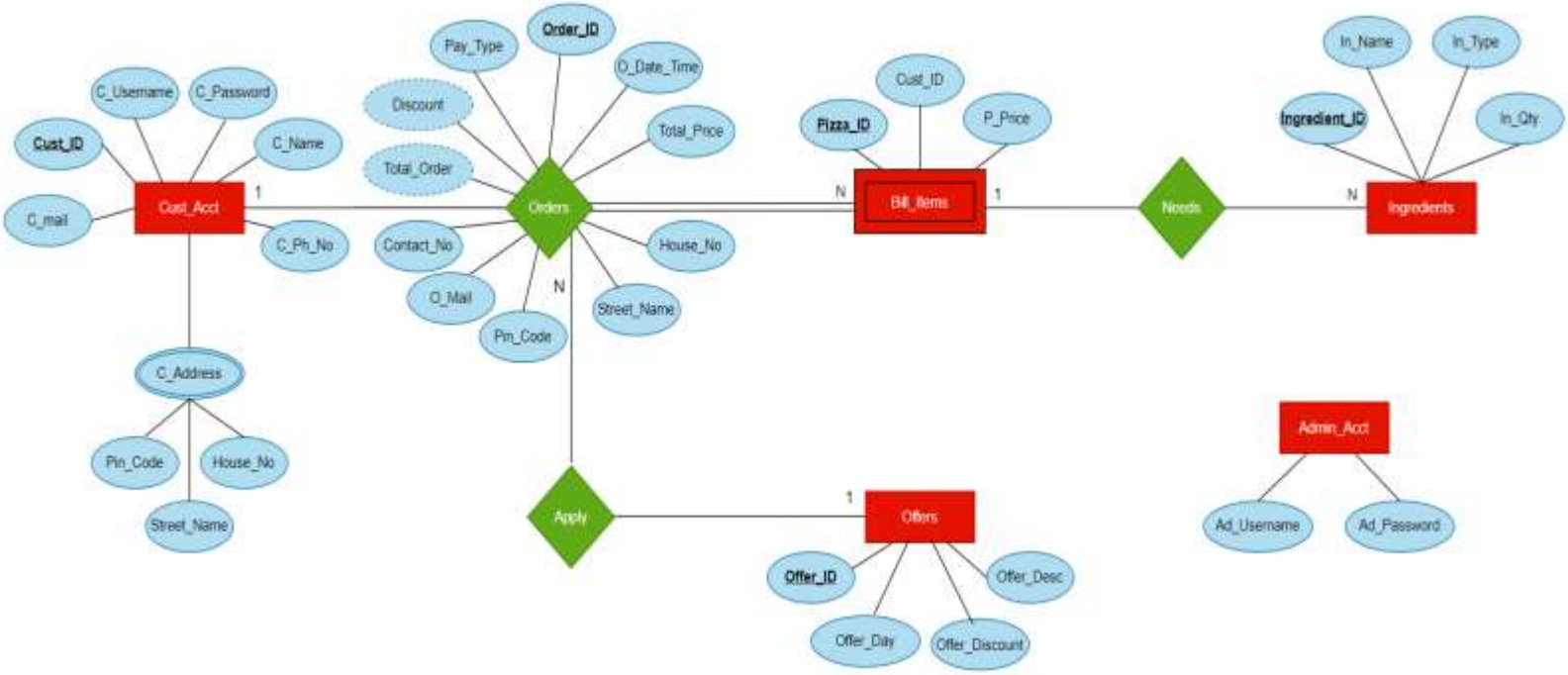
Online Pizza Ordering System is a website for ordering pizza online. Another website was created for management of prices and offers by admins (employees).
NOTE: This particular pizza shop focuses on the build-your-own-pizza business model.

The customer can manage details of their account and orders, and they can access details of pizza ingredients and offers (One offer applied per order).
The admin has access to summary reports, and can view details of ingredients, customers, orders and offers.
The payment type will be cash on delivery or gpay.

ENTITIES

Cust_Acct (PK = Cust_ID)
Admin_Acct (PK = A_Username)
Orders (PK = Order_ID) (FK Cust_ID from Cust_Acct.Cust_ID)
Bill_Items (PK = Pizza_ID, Order_ID) (FK Cust_ID from Cust_Acct.Cust_ID
(FK Order_ID from Orders.Order_ID)
Ingredients (PK = Ingredient_ID
Offers (PK = Offer_ID)

E R DIAGRAM



E R MODEL DESCRIPTION

Cust_Acct - Account details of registered customers

Orders - Orders of each customer for one checkout. It contains contact and address details, total price, final price after discount and so on.

Bill_Items - Each Order is broken up into its constituent items (pizzas ordered in one checkout) and their details are recorded, which include price (calculated from ingredients used) in addition to customer id and order id.

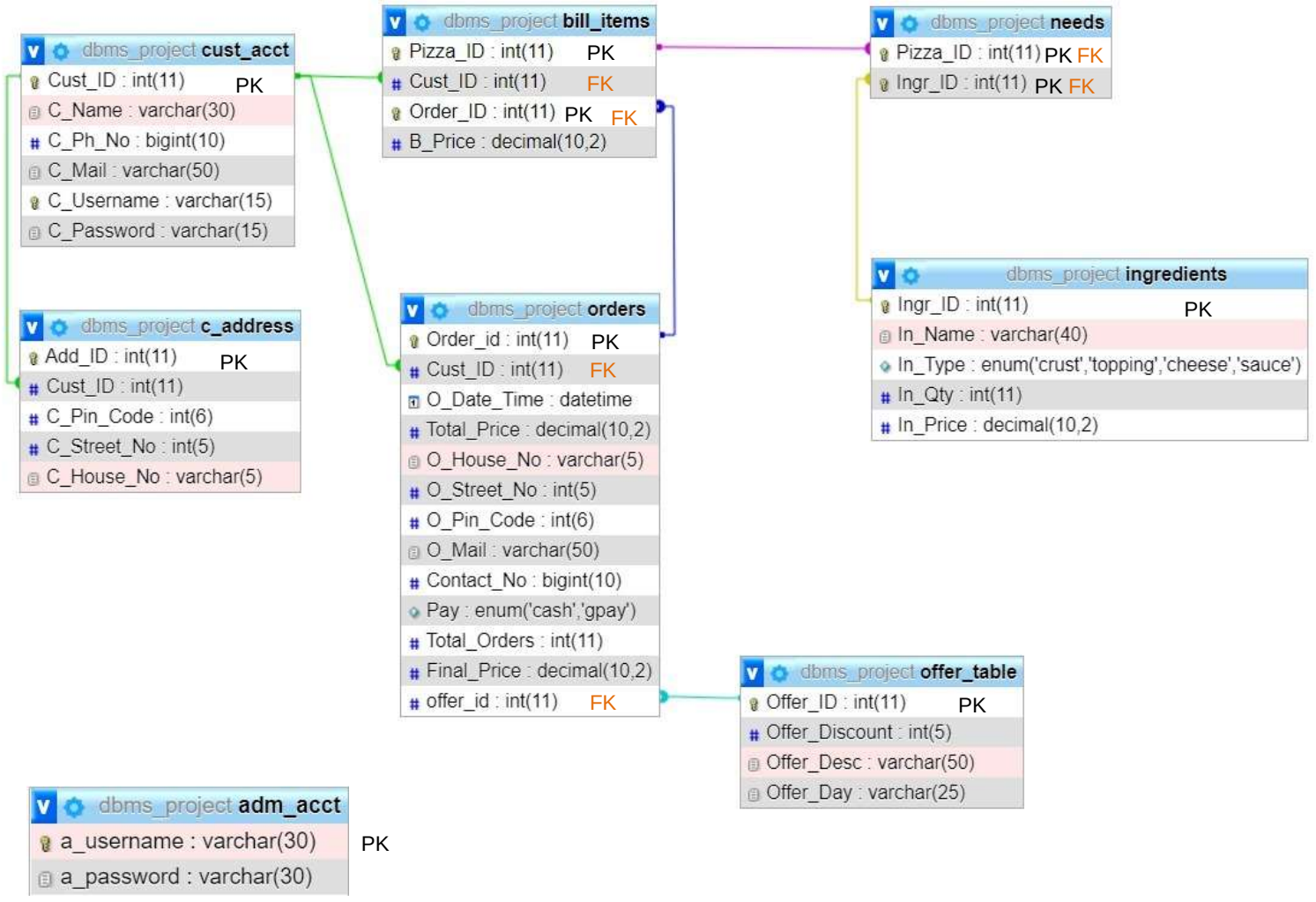
Ingredients - Since this is a build-your-own-pizza establishment, pricing is based on ingredients picked by the customer. Price of ingredients is recorded here.
Specialised into **Crusts, sauce, cheese and toppings**

Offers - Discounts and deals are recorded with their descriptions, requirements and information
Specialised into **Weekly (day of the week).**

NORMALIZATION

- C_Address in Cust table is multivalued, so, applying 1NF, we get the C_Address table.
- The tables are in 2NF as all non-key attributes are dependent on the key(no partial dependencies)
- The tables are in 3NF as there are no transitive functional dependencies.(Customer can give phone numbers that are not linked to their mail id)

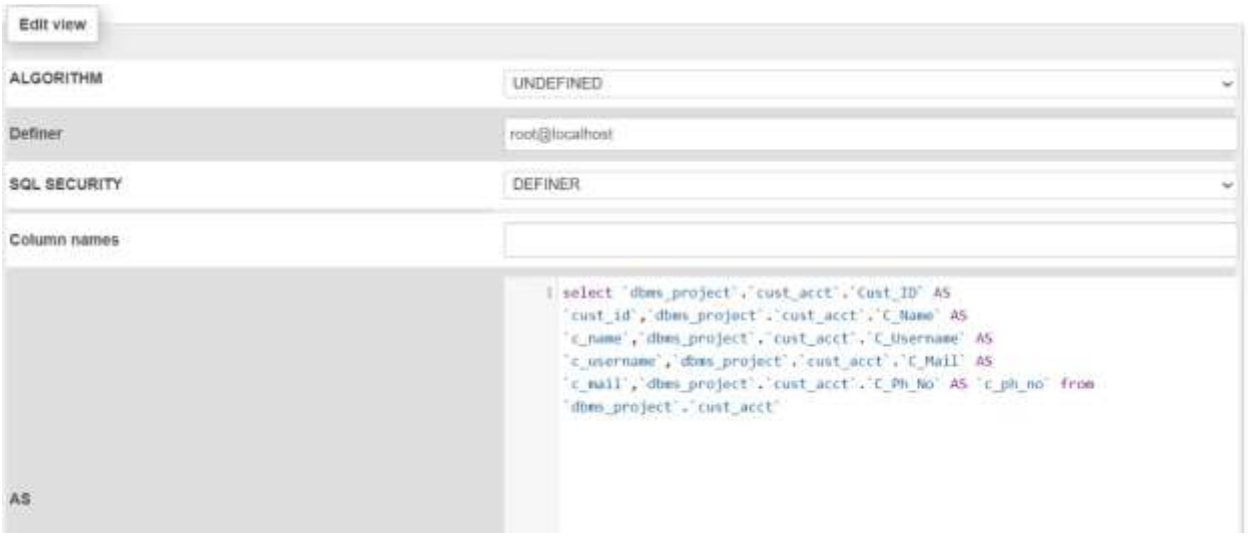
RELATIONAL SCHEM



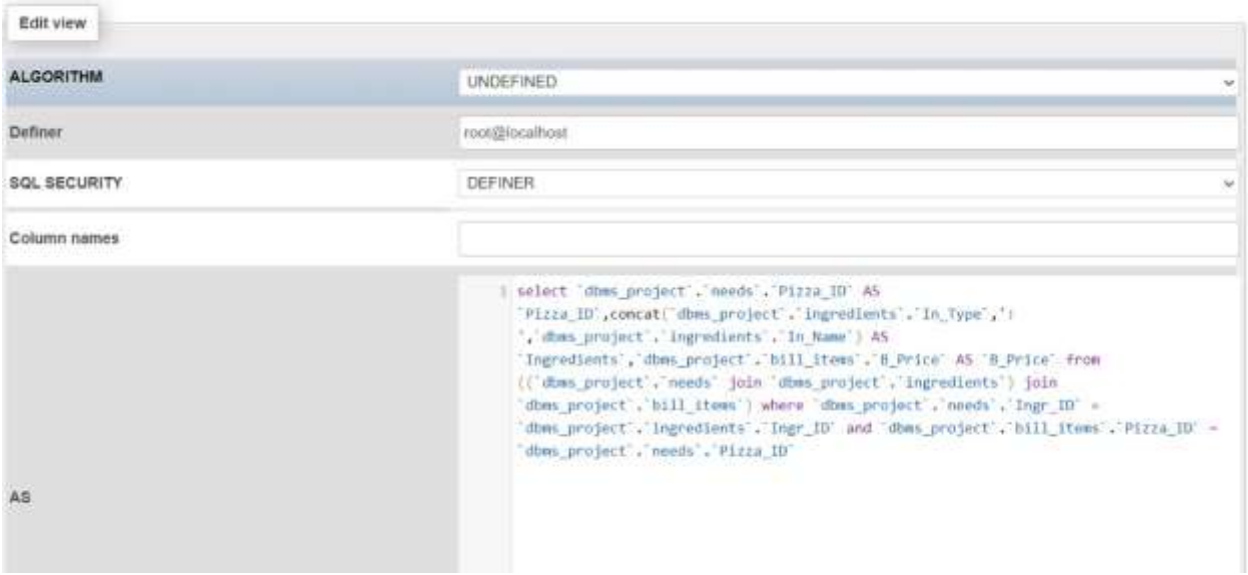
DATABASE

VIEWS –

DISP_CUST



PIZZAINGR



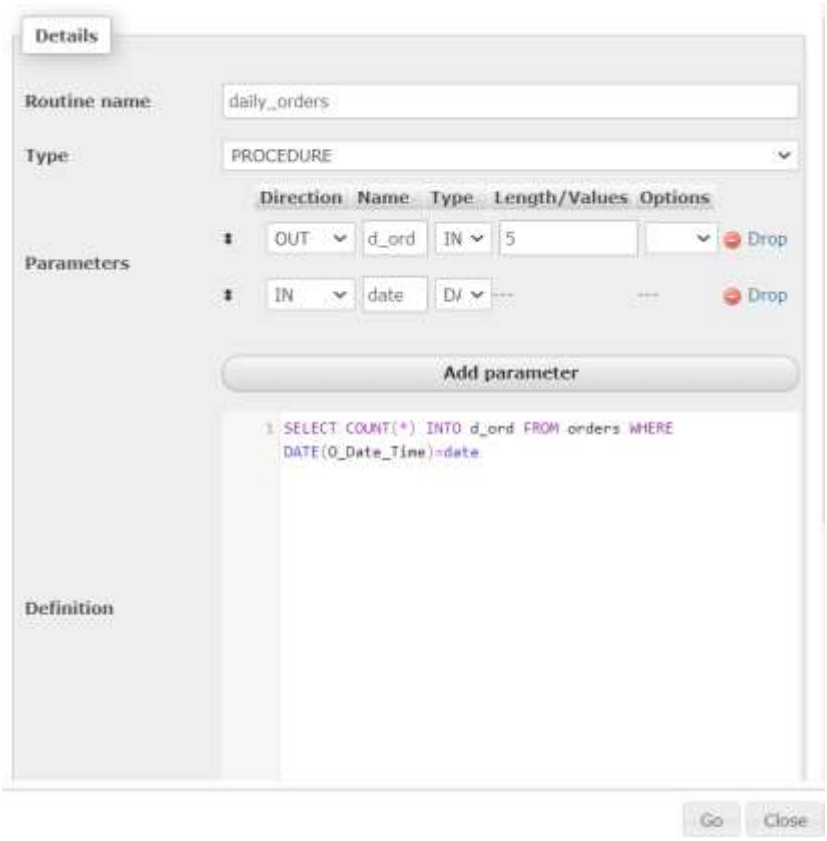
PIZZAINGR_DISP

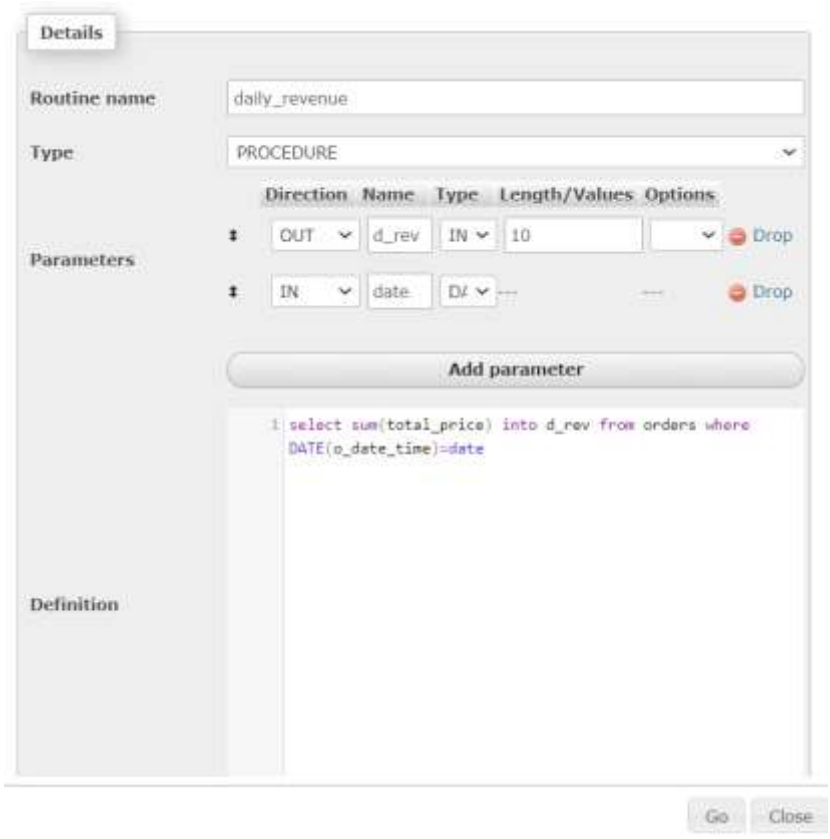
Edit view

ALGORITHM	UNDEFINED
Definer	root@localhost
SQL SECURITY	DEFINER
Column names	
AS	

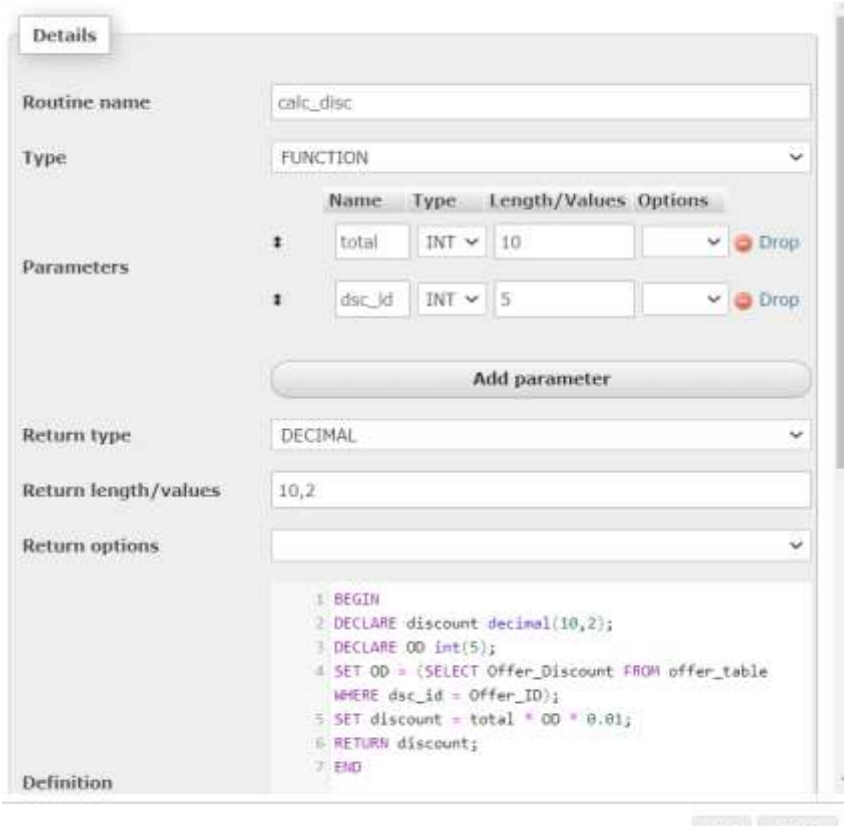
```
SELECT
    `pizzaingr`.`Pizza_ID` AS `Pizza_ID`,
    GROUP_CONCAT(
        `pizzaingr`.`Ingredients` SEPARATOR ','
    ) AS `Ingrs`,
    `pizzaingr`.`B_Price` AS `Price`
FROM
    `dbms_project`.`pizzaingr`
GROUP BY
    `pizzaingr`.`Pizza_ID`,
    `pizzaingr`.`B_Price`
```

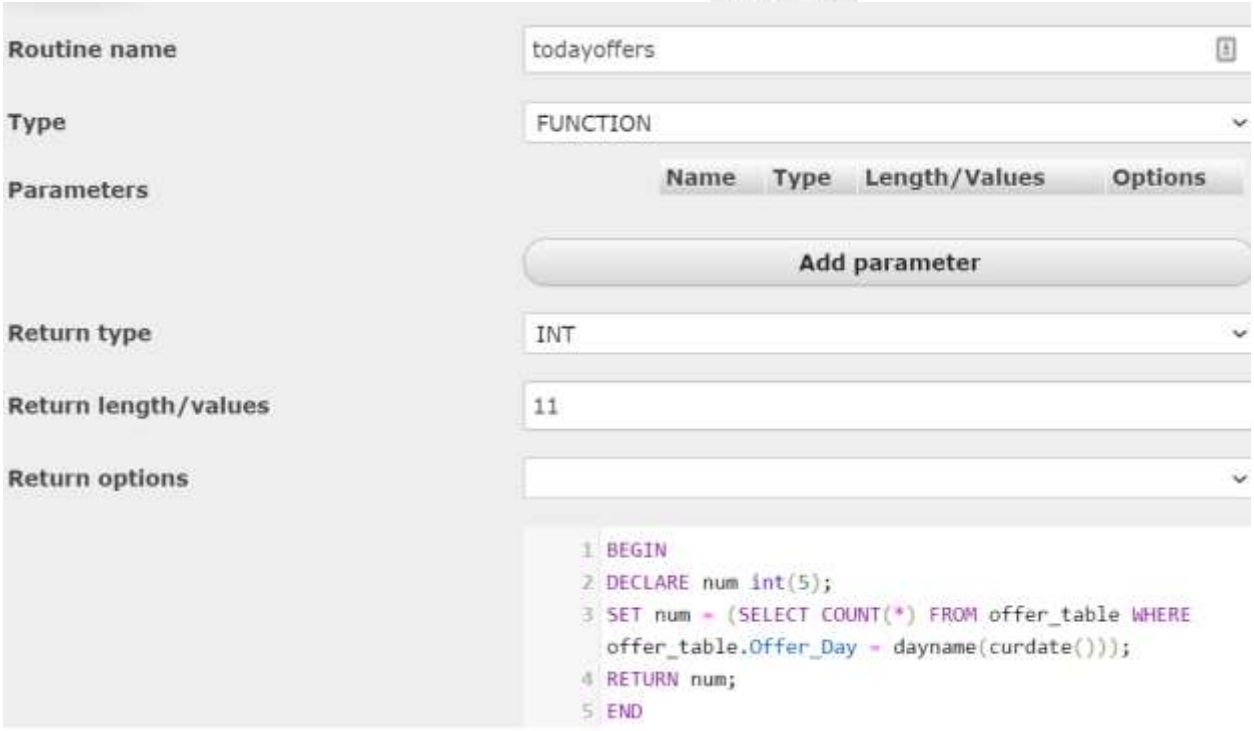
PROCEDURES –


Details
Routine name
daily_orders
Type
PROCEDURE
Direction Name Type Length/Values Options
Parameters
OUT d_ord IN 5 Drop
IN date DA --- --- Drop
Add parameter
1 SELECT COUNT(*) INTO d_ord FROM orders WHERE
DATE(O_Date_Time)=date
Definition
Go
Close


Details
Routine name
daily_revenue
Type
PROCEDURE
Direction Name Type Length/Values Options
Parameters
OUT d_rev IN 10 Drop
IN date DA --- --- Drop
Add parameter
1 select sum(total_price) into d_rev from orders where
DATE(o_date_time)=date
Definition
Go
Close

FUNCTION –


Details
Routine name
calc_disc
Type
FUNCTION
Name
Type
Length/Values
Options
Parameters
total
INT
10
Drop
dsc_id
INT
5
Drop
Add parameter
Return type
DECIMAL
Return length/values
10,2
Return options
1 BEGIN
2 DECLARE discount decimal(10,2);
3 DECLARE OD int(5);
4 SET OD = (SELECT Offer_Discount FROM offer_table
WHERE dsc_id = Offer_ID);
5 SET discount = total * OD * 0.01;
6 RETURN discount;
7 END
Definition


Routine name
todayoffers
Type
FUNCTION
Parameters
Name
Type
Length/Values
Options
Add parameter
Return type
INT
Return length/values
11
Return options
1 BEGIN
2 DECLARE num int(5);
3 SET num = (SELECT COUNT(*) FROM offer_table WHERE
offer_table.Offer_Day = dayname(curdate()));
4 RETURN num;
5 END

TRIGGERS –


Details
Trigger name
Del_Logs
Table
bill_items
Time
AFTER
Event
DELETE
1 UPDATE orders SET Total_Orders =
Total_Orders - 1 where Order_ID =
old.Order_ID
Definition
Definer
root@localhost
Details
Trigger name
Insert_Logs
Table
bill_items
Time
AFTER
Event
INSERT
1 UPDATE orders SET Total_Orders =
Total_Orders + 1 where Order_ID =
new.Order_ID
Definition
Definer
root@localhost
Go
Close

Details

Trigger name: deletefromcart

Table: bill_items

Time: BEFORE

Event: DELETE

Definition:

```
BEGIN
    UPDATE orders SET Total_Price = Total_Price - old.B_Price WHERE orders.Order_id = old.Order_ID;
    UPDATE orders SET Final_Price = Total_Price - (SELECT calc_disc((SELECT Total_Price FROM orders WHERE orders.Order_id = old.Order_ID), (SELECT Offer_ID from orders WHERE orders.Order_id = old.Order_ID))) WHERE orders.Order_id = old.Order_id;
END
```

Definer: root@localhost

Details

Trigger name: insertintocart

Table: bill_items

Time: AFTER

Event: INSERT

Definition:

```
BEGIN
    UPDATE orders SET Total_Price = Total_Price + new.B_Price WHERE orders.Order_id = new.Order_ID;
    UPDATE orders SET Final_Price = Total_Price - (SELECT calc_disc((SELECT Total_Price FROM orders WHERE orders.Order_id = new.Order_ID), (SELECT Offer_ID from orders WHERE orders.Order_id = new.Order_ID))) WHERE orders.Order_id = new.Order_id;
END
```

Definer: root@localhost

Go Close