Lab 3: Assignment_3, Functions, Stored Procedures, Triggers

Lab – 3 20-Aug-2024	Assignment_3, Functions, Stored Procedures, Triggers
IT615 Database Management System, Autumn'2024; Instructor: minal_bhise@daiict	

Objectives: I) Understand & Run functions.
II) Use triggers to execute functions.
III) Understand Stored Procedures.
IV) Solve Problems on the given EC_DB database.

Submission: Each student needs to upload a single .pdf file which will contain the following things for all the queries.

1. Write English query, SQL function, &/or Trigger SQL statement in the given sequence.
2. Screenshot of results.
3. Count of tuples in the results.

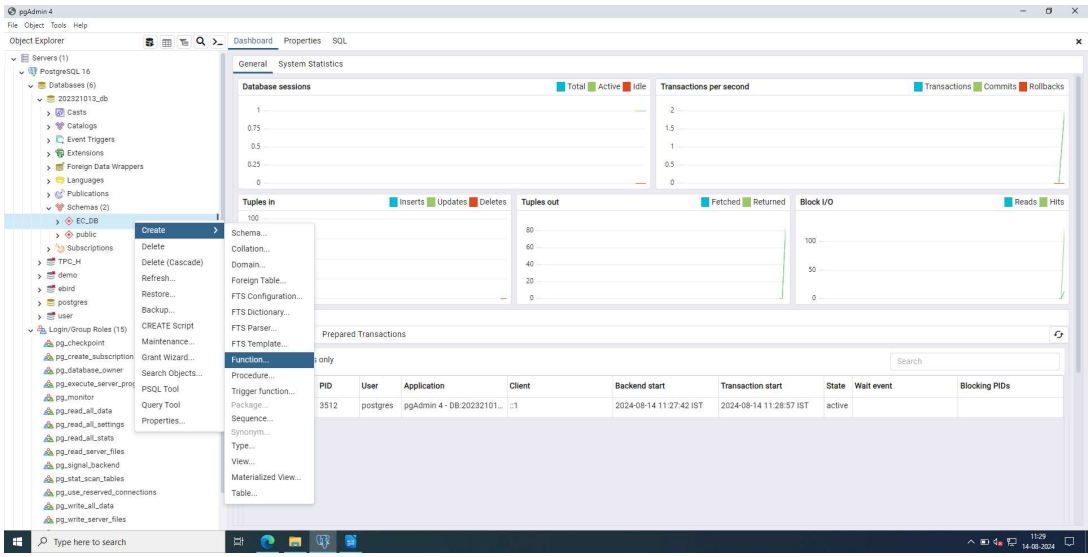
I. Understand & Run functions.

1. Create a simple function to print "Hello World" using GUI.

Step1:

Using GUI – Right click on Functions => Create => Functions

Write Name – Fun_Hello

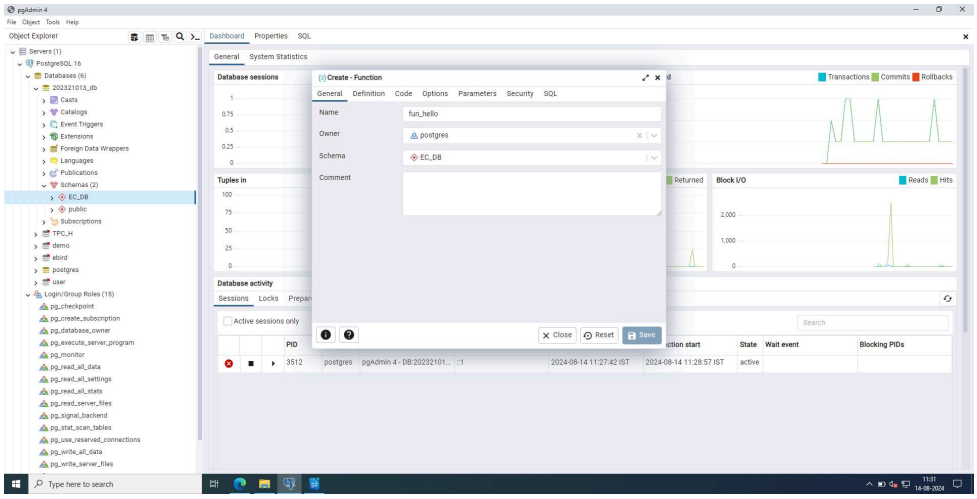


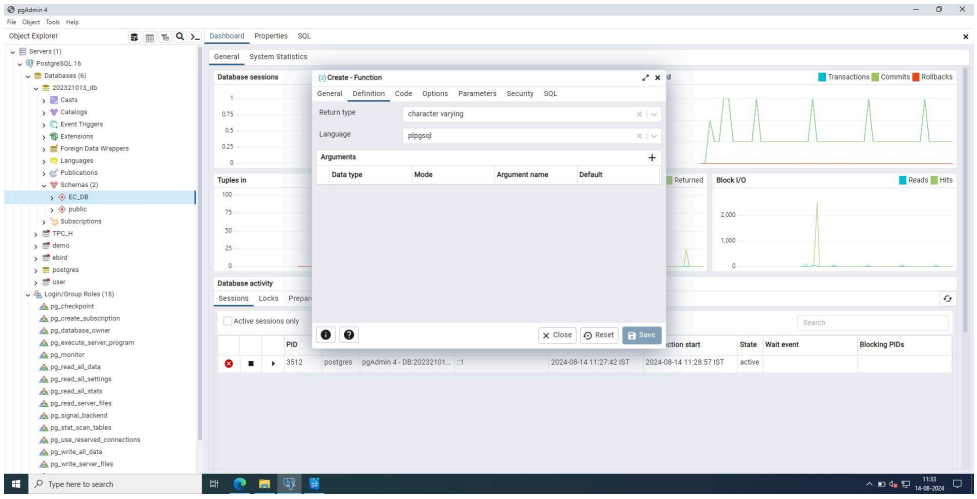
Step2:

Using GUI – Go to next tab => Definition

Choose Return Type => character varying

& Language => **plpgsql**



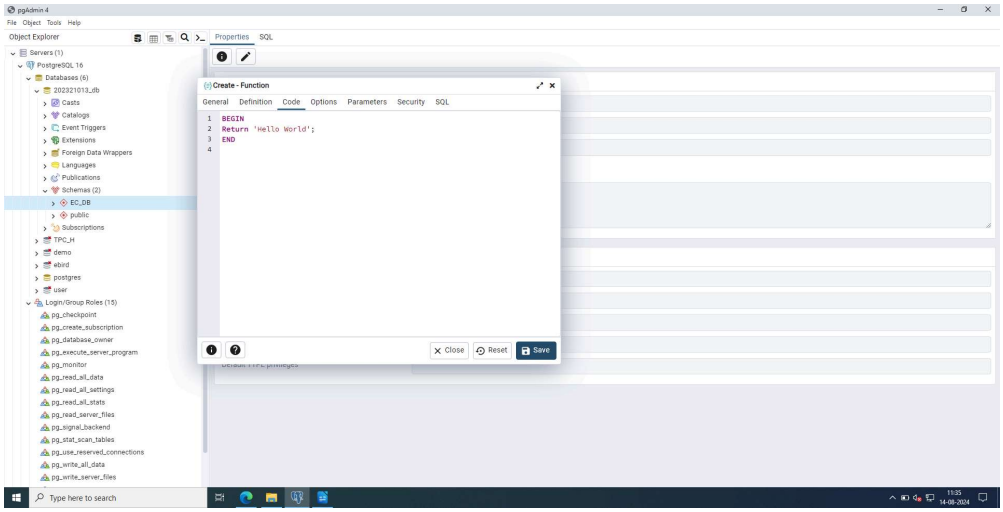


Step3:

Using GUI – Go to next tab => Code

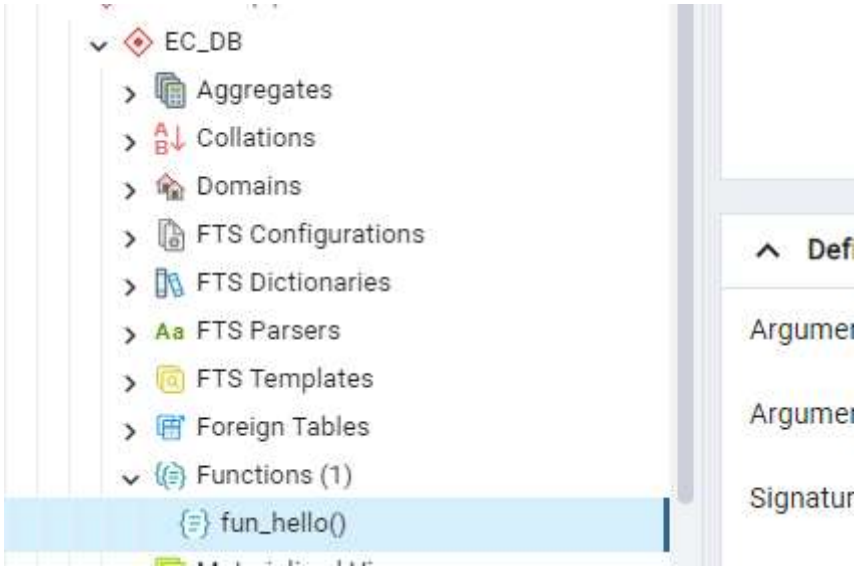
```
BEGIN

Return 'Hello World';

END
```

Using GUI => Save.



Step4:

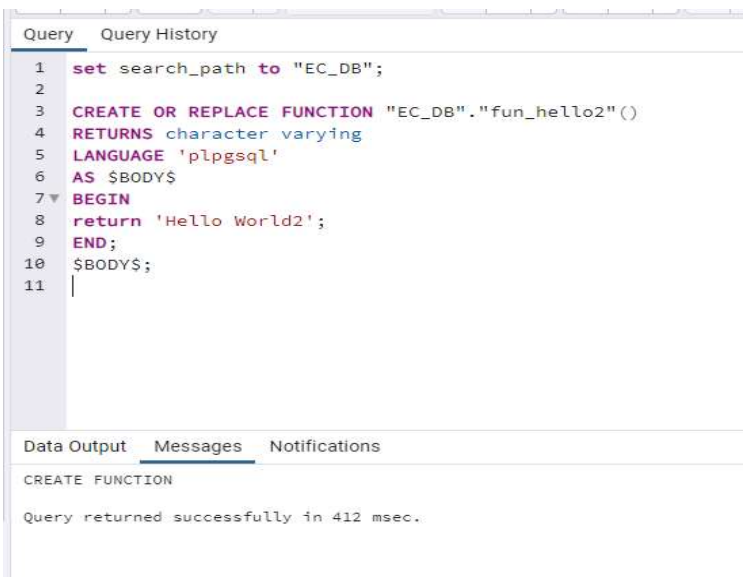
Using GUI – Refresh to Check if a function is created under "Functions".



2. Create a simple function to print "Hello World" using SQL.

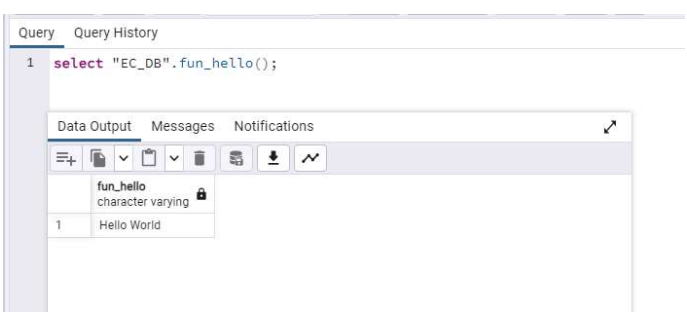
Step1:

```
set search_path to "EC_DB";
CREATE OR REPLACE FUNCTION EC_DB".fun_hello2()
RETURNS character varying
LANGUAGE 'plpgsql'
AS $BODY$
BEGIN
return 'Hello World2';
END;
$BODY$;
```



3. Using the function. "Function calls".

```
SELECT "EC_DB".fun_hello();
```



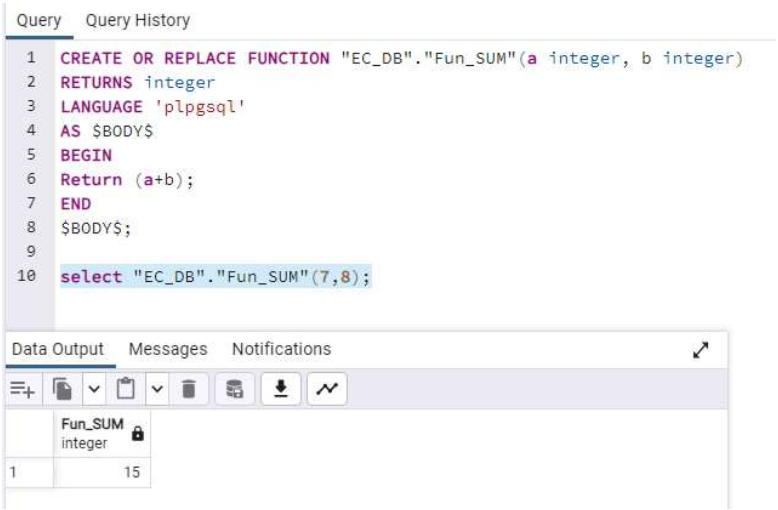
4. **Create functions with parameters that return the SUM of two values.**

Step1: CREATE function.

```
CREATE OR REPLACE FUNCTION "EC_DB".Fun_SUM(a integer, b integer)
RETURNS integer
LANGUAGE 'plpgsql'
AS $BODY$
BEGIN
Return (a+b);
END
$BODY$;
```

Step2: CALL :

```
SELECT "EC_DB".Fun_SUM(7,8);
```

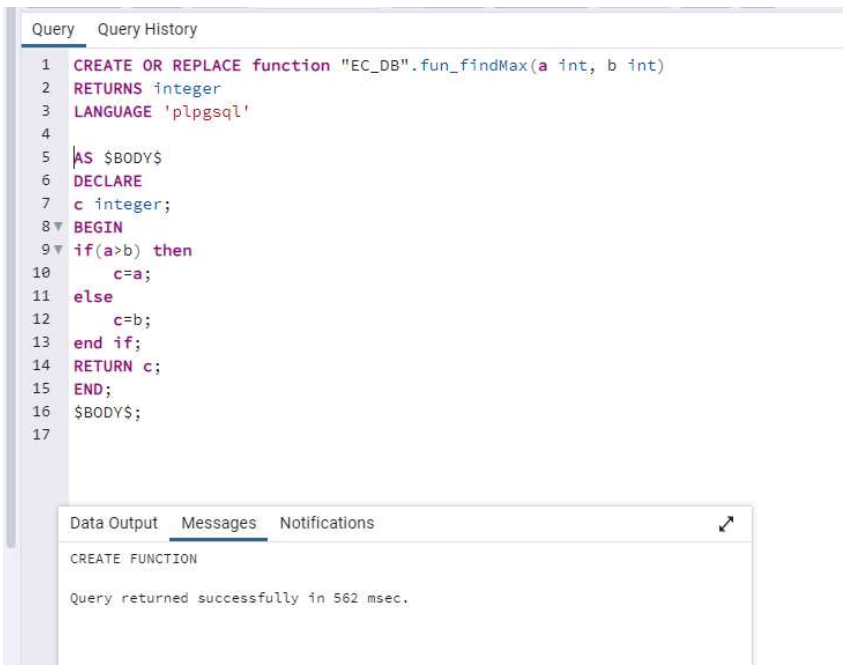


5. **Create a function with If condition to find the largest number of two.**

Step1:

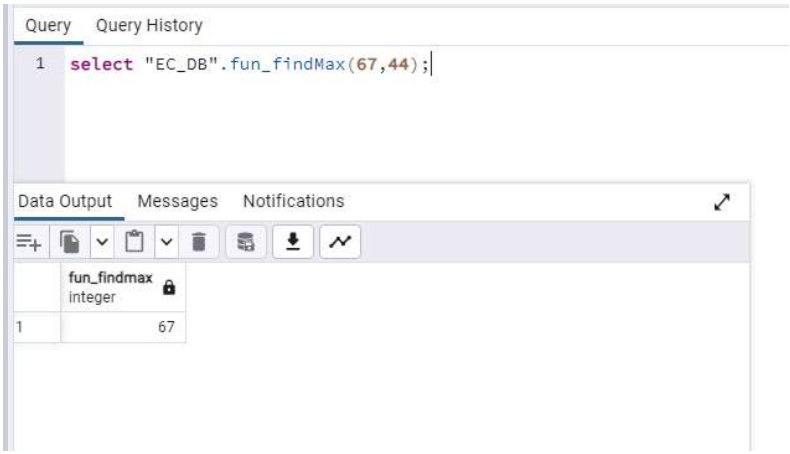
```
CREATE OR REPLACE function "EC_DB".fun_findMax(a int, b int)
RETURNS integer
LANGUAGE 'plpgsql'
AS $BODY$
```

```
DECLARE
c integer;
BEGIN
if(a>b) then
        c=a;
else
        c=b;
end if;
RETURN c;
END;
$BODY$;
```



Step2. CALL :

select "EC_DB".fun_findMax(67,44);



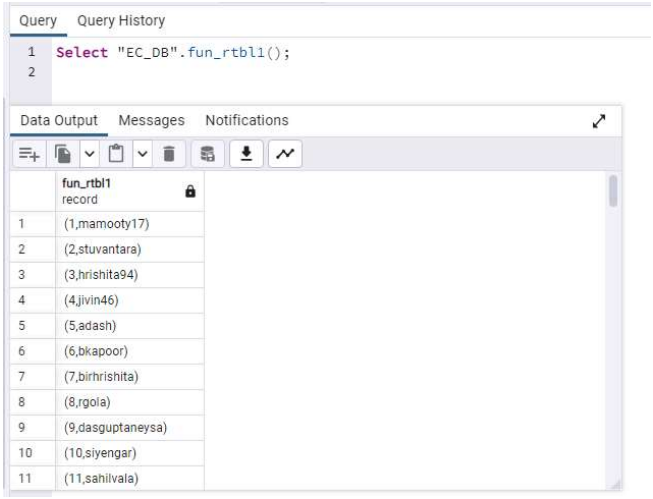
6. Create functions with a table as a return value.

Step1:

```
CREATE OR REPLACE function "EC_DB".fun_rtbl1()
RETURNS TABLE (a int, b character varying(30))
LANGUAGE 'plpgsql'
AS $BODY$
BEGIN
RETURN QUERY EXECUTE format ('SELECT user_id, username
        FROM "EC_DB".users');
END;
$BODY$;
```

Step2. CALL :

Select "EC_DB".fun_rtbl1();



7. Create functions with a Temporary table and use of FOR Loop.

Step1:

```
CREATE OR REPLACE FUNCTION "EC_DB".fun_loop()
RETURNS TABLE(a integer, b character varying)
LANGUAGE 'plpgsql'
AS $BODY$
DECLARE
R_LIST2 record;
BEGIN
CREATE TEMP TABLE test1 (a1 int, b1 character varying(30)) ON COMMIT DROP;
FOR R_LIST2 in (select user_id, username from "EC_DB".users)
loop
Insert into test1 (a1, b1) values (R_LIST2.user_id,R_LIST2.username);
end loop;
RETURN QUERY TABLE test1;
END;
$BODY$;
```

```
CREATE OR REPLACE FUNCTION "EC_DB".fun_loop()
RETURNS TABLE(a integer, b character varying)
LANGUAGE 'plpgsql'
AS $BODY$
DECLARE
R_LIST2 record;
BEGIN
CREATE TEMP TABLE test1 (a1 int, b1 character varying(30)) ON COMMIT DROP;
FOR R_LIST2 in (select user_id, username from "EC_DB".users)
loop
Insert into test1 (a1, b1) values (R_LIST2.user_id,R_LIST2.username);
end loop;
RETURN QUERY TABLE test1;
END;
$BODY$;
```

Data Output Messages Notifications

```
CREATE FUNCTION

Query returned successfully in 292 msec.
```

```
select "EC_DB".fun_loop();
```

Data Output Messages Notifications

	fun_loop record
1	(1,mamooty17)
2	(2,stuvantara)
3	(3,hrishita94)
4	(4,jivin46)
5	(5,adash)
6	(6,bkapoor)
7	(7,birhrishita)
8	(8,rgola)
9	(9,dasguptaneysa)
10	(10,siyengar)
11	(11,sahilvala)

II. Use triggers to execute functions.

The main difference between a normal function & a trigger function is its return type. The trigger function gets automatically called based on some events set earlier, like Insert, Update, &/or Delete on some table.

1. Create a trigger function.

Step1: Create.

create table "EC_DB".message (m char varying);

CREATE OR REPLACE FUNCTION "EC_DB".Trigger_fun()

RETURNS trigger

LANGUAGE 'plpgsql'

AS $BODY$

BEGIN

Insert into "EC_DB".message(m) values (concat('New id added=',NEW.user_id));

RETURN NEW;

END

$BODY$;

Query Query History

```
1
2  CREATE OR REPLACE FUNCTION "EC_DB".Trigger_fun()
3  RETURNS trigger
4  LANGUAGE 'plpgsql'
5  AS $BODY$
6  BEGIN
7  Insert into "EC_DB".message(m) values (concat('New id added=',NEW.user_id));
8  RETURN NEW;
9  END
10 $BODY$;
11
```

Data Output Messages Notifications

```
CREATE FUNCTION

Query returned successfully in 221 msec.
```

Step2. Add the trigger function call using below given code.

CREATE TRIGGER "Trigger_insert"

AFTER INSERT

ON "EC_DB".users

FOR EACH ROW

EXECUTE PROCEDURE "EC_DB".Trigger_fun();

Query Query History

```
1 CREATE TRIGGER "Trigger_insert"
2 AFTER INSERT
3 ON "EC_DB".users
4 FOR EACH ROW
5 EXECUTE PROCEDURE "EC_DB".Trigger_fun();
6
```

Data Output Messages Notifications

```
CREATE TRIGGER

Query returned successfully in 229 msec.
```

Step3: Auto call to trigger a function when inserts happen on the table where we had created the trigger call.

Insert into "EC_DB".users values (2000, 'Vinay123', 'abc@gmail.com', 'vabcd123', 'Vinay', 'Gupta', 'Delhi', '1234567890');

Step4: Check that the trigger function added a new record automatically in the message table.

III. Understand Stored Procedures.

Stored procedures are similar to normal functions. Except minor differences in the create statement and calling.

1. **Create a simple stored procedure to insert records in a table.**

Step1: CREATE stored procedure.

```
CREATE OR REPLACE PROCEDURE "EC_DB".insert_users(
    u_id integer,
    u_name character varying,
    e_m character varying,
    p_w character varying,
    f_name character varying,
    l_name character varying,
    add character varying,
    ph_num character varying
)
```

LANGUAGE plpgsql

AS $BODY$

BEGIN

INSERT INTO "EC_DB".users

(user_id, username, email, password, first_name, last_name, address, phone_number)

VALUES (u_id, u_name, e_m, p_w, f_name, l_name, add, ph_num);

END;

$BODY$;

Step2: CALL the stored procedure.

CALL "EC_DB".insert_users(9000, 'abhay123', 'abhay@gmail.com', '123456', 'abhay', 'sharma', 'Gujarat', '1234567890');

It will insert the new records in the users table with ID 9000 and other details.

2. **One more sample stored procedure to list all records as a message.**

Step1: CREATE stored procedure.

```
CREATE OR REPLACE PROCEDURE "EC_DB"."SP_loop"()

LANGUAGE 'plpgsql'

AS $BODY$

DECLARE

R_LIST record;

BEGIN

FOR R_LIST in (select user_id, username from EC_DB.users)

Loop

RAISE NOTICE 'X = %, Y = %', R_LIST.user_id, R_LIST.username;

end loop;

END;

$BODY$;
```

```
CREATE OR REPLACE PROCEDURE "EC_DB"."SP_loop"()
LANGUAGE 'plpgsql'
AS $BODY$
DECLARE
R_LIST record;
BEGIN
FOR R_LIST in (select user_id, username from "EC_DB".users)
Loop
RAISE NOTICE 'X = %, Y = %', R_LIST.user_id, R_LIST.username;
end loop;
END;
$BODY$;
```

Data Output | Messages | Notifications

```
CREATE PROCEDURE

Query returned successfully in 113 msec.
```

Step2. CALL the stored procedure.

CALL "EC_DB".SP_loop();

Query Query History

```
1  CALL "EC_DB"."SP_loop"();
```

Data Output Messages Notifications

```
NOTICE:  X = 1, Y = mamooty17
NOTICE:  X = 2, Y = stuvantara
NOTICE:  X = 3, Y = hrishita94
NOTICE:  X = 4, Y = jivin46
NOTICE:  X = 5, Y = adash
NOTICE:  X = 6, Y = bkapoor
NOTICE:  X = 7, Y = birhrishita
NOTICE:  X = 8, Y = rgola
NOTICE:  X = 9, Y = dasguptaneysa
NOTICE:  X = 10, Y = siyengar
NOTICE:  X = 11, Y = sahilvala
NOTICE:  X = 12, Y = armaan19
NOTICE:  X = 13, Y = usangha
NOTICE:  X = 14, Y = charamira
NOTICE:  X = 15, Y = ravalyuvraj
NOTICE:  X = 16, Y = divithayre
NOTICE:  X = 17, Y = charstuvan
NOTICE:  X = 18, Y = mohanlalbarad
NOTICE:  X = 19, Y = vsagar
NOTICE:  X = 20, Y = surtarini
```

IV) Solve Problems on the given EC_DB database

Q1. Create a trigger on Table of your choice to check if the Primary key ID already exists or not before inserting a new record. & Send a custom reply instead of an error message.

Q2. Create a trigger on the Table of your choice to check if the Foreign key ID already exists or not before inserting a new record. & Send a custom reply instead of an error message.

Q3. Write a SQL function that takes a product's product_id and a discount percentage as inputs and returns the discounted price of the product.

Hint: Use the formula discounted_price = price - (price * discount_percentage / 100).

Q4. Create a stored procedure named add_order that takes the user_id, shipping_address, and a list of product_id and quantity pairs as inputs, and inserts a new order into the orders and order_details tables. The procedure should also update the stock quantity of the products.

Hint: Use loops and transaction control within the procedure to handle the insertion of multiple order details.

Q5. Write a trigger that automatically decreases the stock quantity of a product in the products table when a new order is inserted into the order_details table.

Hint: Use an AFTER INSERT trigger on the order_details table to update the corresponding stock_quantity in the products table.