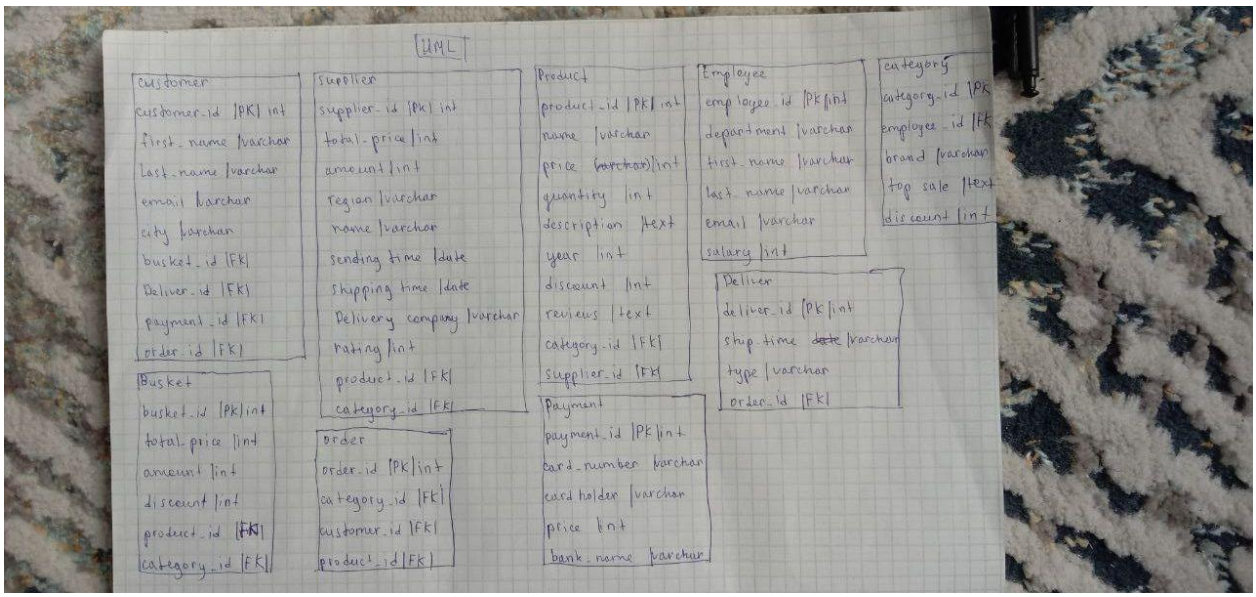
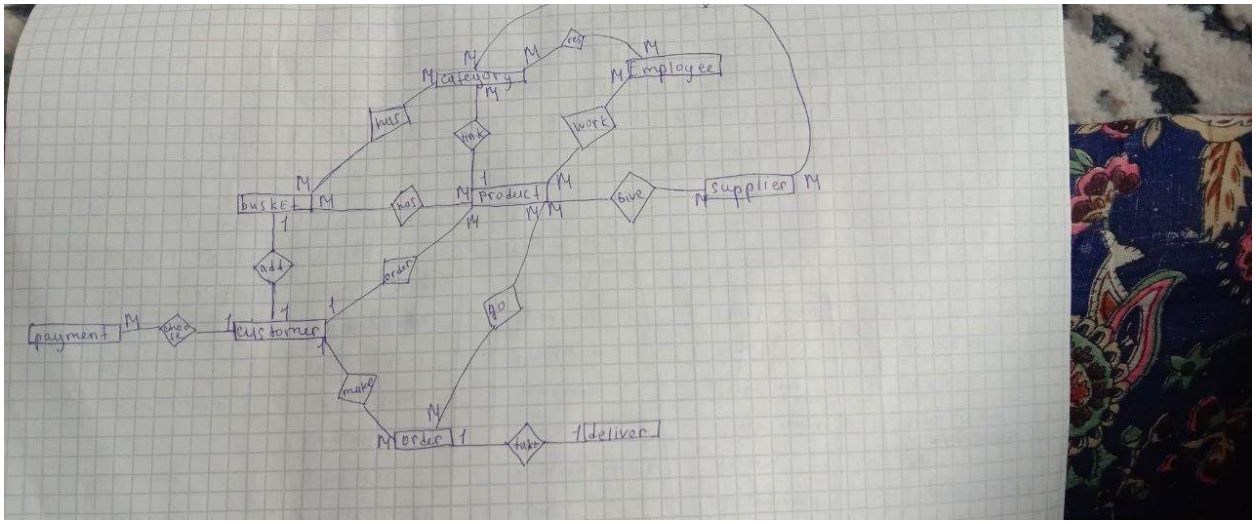
ID : 210103089
Mamushev Arup

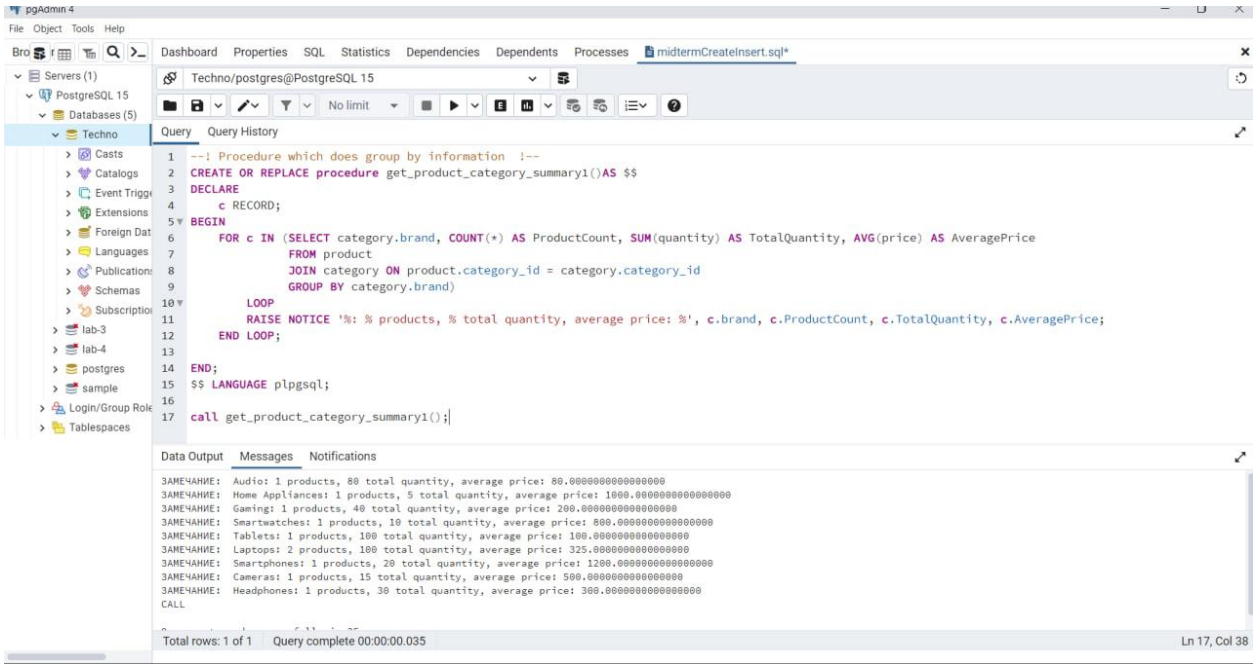
Hello. Now I am going to tell you about TECHNO E-commerce shop. Which consist of 8 entities. Let’s start from the bottom-up. Firstly we have a supplier which has relation with the product which it delivers and category in sales. I am positioning a product in one particular model like ‘Google Pixel 7’ has one product_id. And has different categories for sale. And there is an employee table where employees support and improve products and categories. And on the other side I have a number of tables which are close to the customer. And I connect them with products and customers. Customers can add products to the basket and if there discount exists get it. And they can choose a bank to pay and then it will go to the order table. After that at the end it goes to the delivery table. That is the entire logic for my database.






And the next part explains why it follows normal forms. My entire work follows 3nf and that's why.
1NF requires that in one field there must be one value. It keeps atomicity. We can't store lists or arrays or something else. Only one value. 2NF it completely follows for 1NF and then it must also ensure that all non-key attributes of the table are functionally dependent on the entire primary key of the table. 3NF it completely follows 2NF and any non-key columns that are functionally dependent on other non-key columns are removed from the table and placed in their own table. So my database completely follows the 3NF.

Next coding part is to write some procedures, triggers , exceptions and functions. For advanced syntax of sql I will use PostgreSQL and its syntax extensions plpgsql.




Here I combine two tables and record it to the variable c which has record type. As it is a procedure there is no need to return anything and for calling this procedure I use the CALL keyword.

```
22
23
24
25  --! Function which counts the number of records  !--
26  CREATE OR REPLACE FUNCTION count_records(table_name text)
27  RETURNS integer AS $$
28  DECLARE
29      record_count integer;
30  BEGIN
31      EXECUTE 'SELECT COUNT(*) FROM ' || table_name INTO record_count;
32      RETURN record_count;
33  END;
34  $$ LANGUAGE plpgsql;
35
36  SELECT count_records('supplier');
37
38
39
```

Data Output | Messages | Notifications

| | count_records integer |
|---|---|
| 1 | 10 |

Here I write a function by requirements to count the number of records. It takes one parameter table name and by given table count the number of records. It returns an integer.

```sql

--! Procedure which uses SQL%ROWCOUNT to determine the number of rows affected !--
CREATE OR REPLACE PROCEDURE update_table(p_product_id IN integer, p_new_price IN integer)
LANGUAGE plpgsql
AS $$
DECLARE
    rows_affected integer;
BEGIN
    UPDATE employee SET salary = p_new_price WHERE employee_id = p_product_id;
    GET DIAGNOSTICS rows_affected = ROW_COUNT;
    RAISE NOTICE 'Number of rows affected: %', rows_affected;
END;
$$;

call update_table(3,145000);
```

Data Output | Messages | Notifications

```
ЗАМЕЧАНИЕ:  Number of rows affected: 1
CALL

Query returned successfully in 59 msec.
```

In this procedure we accept 2 parameters and update the salary employee by getting his id. And then find how many rows are changed. Because of using id we change only one row. Get diagnostic statement is used to retrieve the status of the last SQL statement that was executed.

```sql
--! Add user-defined exception which disallows to enter title of item (e.g. book) to be less than 5 characters !--
CREATE OR REPLACE FUNCTION add_product(IN product_title VARCHAR(100), IN product_price NUMERIC) RETURNS INTEGER AS $$
DECLARE
    product_id23 INTEGER;
BEGIN
    IF LENGTH(product_title) < 5 THEN
        RAISE EXCEPTION 'Product title should be at least 5 characters long.';
    END IF;

    INSERT INTO product(product_name, price) VALUES (product_title, product_price) RETURNING product_id INTO product_id23;
    RAISE NOTICE 'Product with ID % has been added.', product_id23;
    return product_id23;
END;
$$ LANGUAGE plpgsql;

SELECT add_product('Mini', 10.99);
```

Data Output | Messages | Notifications

```
ERROR: ОШИБКА:  Product title should be at least 5 characters long.
CONTEXT:  функция PL/pgSQL add_product(character varying,numeric), строка 6, оператор RAISE


SQL state: P0001
```

Here we work with exceptions and take two in parameters and as it is a function we return an integer. If our product title length is less than 5 we raise an exception otherwise insert into the product table.

```sql
--! Create a trigger before insert on any entity which will show the current number of rows in the table !--
CREATE OR REPLACE FUNCTION show_row_count()
RETURNS TRIGGER AS $$
DECLARE
  row_count INTEGER;
BEGIN
  SELECT count(*) INTO row_count FROM customer;
  RAISE NOTICE 'Current row count: %', row_count;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER before_insert_my_table
BEFORE INSERT ON customer
FOR EACH ROW
EXECUTE FUNCTION show_row_count();

insert into customer(first_name, last_name) values ('Jordan', 'Peterson')

```

Data Output | Messages | Notifications

```
ЗАМЕЧАНИЕ:  Current row count: 10
INSERT 0 1

Query returned successfully in 41 msec.
```

Here for triggers I write a separate function which runs the main logic. We have row_count int type and when we insert it to the customer table before that we run the function and save it to row count and message it.