FINAL PROJECT

TOPIC: ONLINE MARKET: CAKE SHOP

AUTHORS: NAZERKE BADANOVA, DAMIR AMANKOS, AZAMAT ZHANBOLATOV

Description:

1. **"The System for Ordering" Database project:** **The prime motive of this database is to make the order transaction speedy and smooth for the customers, to maintain and secure the order records.The system is capable enough to book the cake as per the customer need and event.**

- **A customer can register to purchase an item. The customer will provide the account number and bank name.**
- **After registration, each customer will have a unique customer, user id, and password.**
- **A customer can purchase one or more items in different quantities.**

End users:

- **Casual End Users: Customers**
- **Parametric end users:** Reservation clerks basically check availability for a given request,check whether the requested product is in stock
- **Parametric end users:** Clerks who are working at receiving end for company enter the product identifies via barcodes and descriptive information through buttons to update a central database

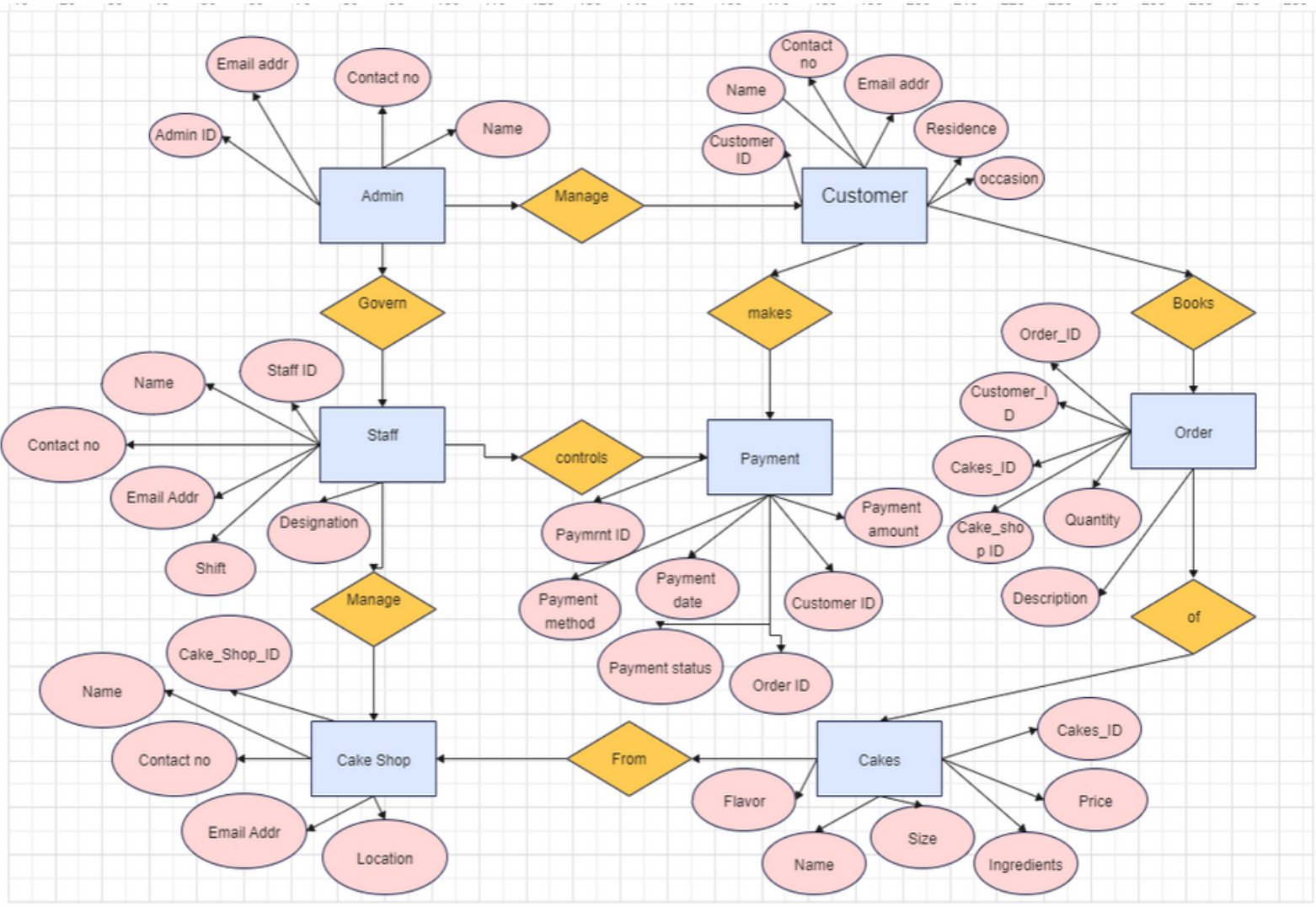
Data obsolescence:

- the best solution is to partition tables by the the **key** which differentiates stale from current data (such as **date, currency_id** or things like that).
- Setting up daily obsolescence for on any rows retrieved by **queries**

Project Idea:

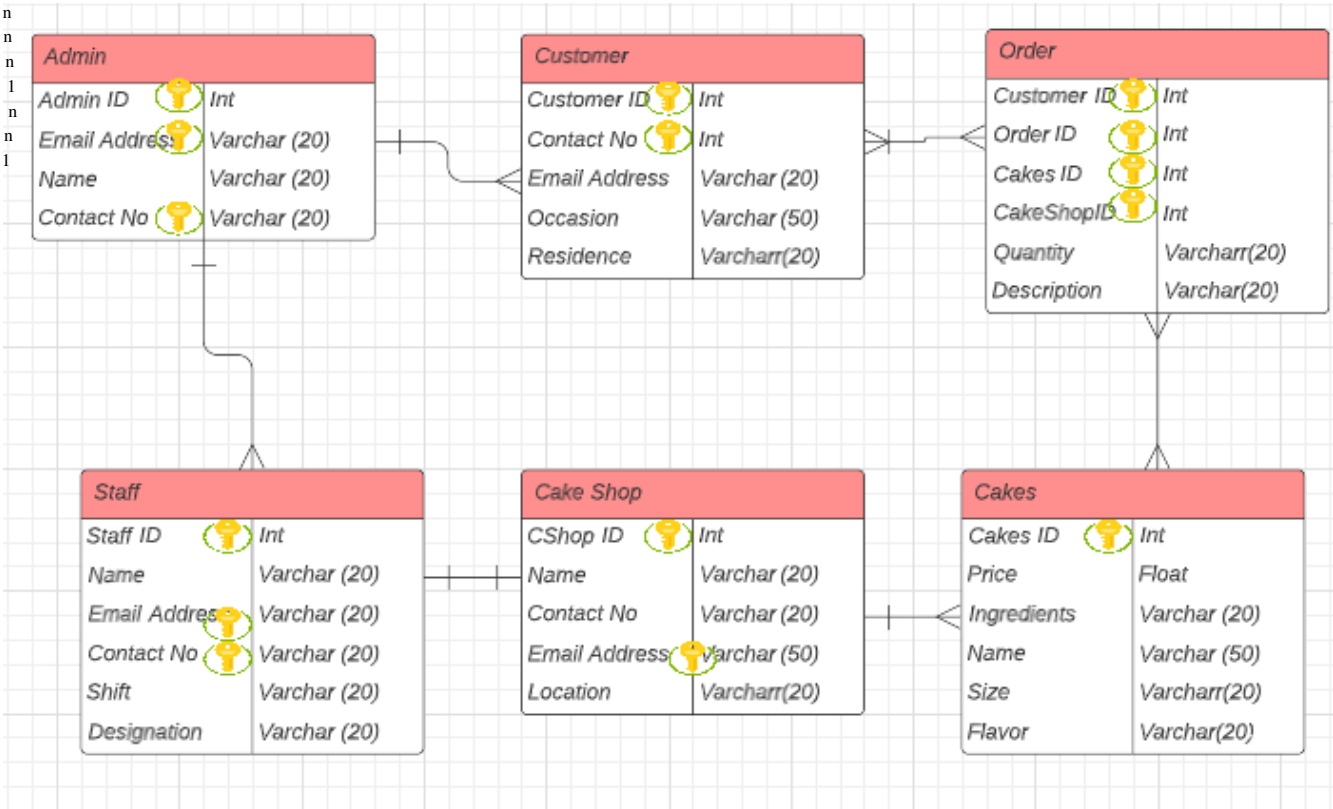
- The Database System increasing Walmart Sales
- Environment

Entity Relationship Diagram



Relationships:

- Admin-Customer: 1 to n
- Customer-Order: n to n
- Order-Cakes: n to n
- Cake shop-Cakes: 1 to n
- Cake shop-staff: 1 to 1
- Admin-staff: 1 to n
- Customer-payment 1 to n
- Staff-Payment n to 1



Normalization

Functional dependencies:

1) CUSTOMER ID--->CONTACT NO, NAME, EMAIL ADDRESS, RESIDENCE, OCCASION
SUPERKEYS:(CUSTOMER_ID); (CUSTOMER_ID, CONTACT_NO); (CUSTOMER_ID,NAME); (CUSTOMER_ID,EMAIL)

2) ORDER ID--->CUSTOMER ID, CAKES ID, CAKE SHOP ID, QUANTITY, DESCRIPTION
SUPERKEYS:(ORDER_ID); (ORDER_ID, CUSTOMER_ID))

3) CAKES ID--->PRICE, INGREDIENTS, SIZE, NAME, FLAVOR
SIZE---> INGREDIENTS, PRICE ⟶ NON-PRIME--->NON-PRIME, SO IT IS TRANSITIVE DEPENDENCY
SUPERKEYS:CAKES_ID

table 1 | C_ID | Name | Flavor |

table 2 | Size | Price | Ingr |

4) CAKE_SHOP_ID--->NAME, CONTACT_NO, LOCATION, EMAIL ADDRESS
NAME---> LOCATION
SUPERKEYS:(CAKE_SHOP ID); (CAKE SHOP ID, NAME))

table 1 | Sh_ID | email | c_no |

table 2 | name | loctn |

5) STUFF_ID--->NAME, CONTACT_NO, SHIFT, DESIGNATION, EMAIL ADDRESS
DESIGNATION---> SHIFT
SUPERKEYS:(STUFF ID); (STUFF ID, NAME); (STUFF_ID,CONTACT_NO)

table 1 | S_ID | Name | Email |

table 2 | Dsgntn | Shift |

6) ADMIN_ID--->NAME, CONTACT_NO, EMAIL ADDRESS
SUPERKEYS:(ADMIN ID); (ADMIN ID, NAME); (ADMIN_ID,CONTACT_NO)

1NF

- There are no repeating groups or arrays of values, assuming that each customer has only one contact number, name, email address, residence, and occasion.
- Each attribute value is atomic. In this case the attribute "Residence" is indivisible since an address is given as one complete data. we have tried to strike a balance between normalization and efficiency when designing a database schema.
- Each row have a unique identifier(Customer ID)
- No rows may be duplicated
- No row/column intersections contain a null value

2 NF

- All non-key attributes depend on the primary key(Customer ID)
- There is no partial dependencies

3 NF

- Every non-prime attribute is dependent on the primary key(Studemt_ID) and nothing else.
- For example, all non-prime attributes such as occasion and residence are dependent on the Student_ID directly, and not indirectly through attribute name.
- When we will change the name, residence or other information about customers, the data change will happen in only one place because there is no transitive dependency

C_ID	Name	Occasion
1	Allison	birthday

→

C_ID	Name	Occasion
1	Allison	new_year

Table 1

```
CREATE TABLE Customer (
    Customer_ID VARCHAR2(50) PRIMARY KEY,
    Customer_name VARCHAR2(50),
    Contact_no VARCHAR2(50),
    email VARCHAR2(50),
    residence VARCHAR2(50),
    occasion VARCHAR2(50)
);
insert into Customer (Customer_ID, Customer_name, Contact_no, email, residence, occasion) values ('223.129.222.106/4',
insert into Customer (Customer_ID, Customer_name, Contact_no, email, residence, occasion) values ('73.55.185.179/30',
insert into Customer (Customer_ID, Customer_name, Contact_no, email, residence, occasion) values ('238.180.75.122/30',
```

All Rows Fetched: 20 in 0,007 seconds

	CUSTOMER_ID	CUSTOMER_NAME	CONTACT_NO	EMAIL	RESIDENCE	OCCASION
1	135.15.2.224/9	Nico	534-985-6060	ngreenwelli@sitemeter.com	0007 Sugar Circle	anniversary
2	130.15.2.224/9	Nico	534-985-6060	ngreenwelli@sitemeter.com	0007 Sugar Circle	anniversary
3	139.218.173.2/28	Darrick	717-164-9919	dsavellh@myspace.com	8559 New Castle Drive	Comedy
4	167.117.10.216/2	Hershel	966-168-5343	hivashkovg@ucoz.com	6 1st Point	New year
5	167.11.10.216/2	Hershel	966-168-5343	hivashkovg@ucoz.com	6 1st Point	New year
6	167.111.10.216/2	Hershel	966-168-5343	hivashkovg@ucoz.com	6 1st Point	New year
7	207.184.96.79/3	Katherine	103-564-0048	kwassellf@exblog.jp	5 Toban Trail	Horror
8	22.225.118.6/1	Isabeau	566-503-9427	idedeie@hugedomains.com	3427 Hollow Ridge Park	empty
9	182.210.154.40/31	Whittaker	577-876-4351	wpeverelld@latimes.com	7 Lien Avenue	empty
10	39.224.179.84/24	Miguelita	607-601-8923	mcruickshankc@multiply.com	26 Eagle Crest Junction	Birtday
11	236.64.230.12/20	Beatrisa	870-926-2812	bnabbso@seesaa.net	73318 Park Meadow Terrace	Adventure|Drama
12	16.196.73.109/7	Noellyn	473-689-3726	nlamondn@github.io	00 Hansons Street	Horror|Mystery|Thriller
13	117.126.69.241/18	Siegfried	887-647-8316	shadcockm@netlog.com	95378 East Center	Drama

Table 2

```
CREATE TABLE Order_ (
  Order_ID VARCHAR2(20),
  Customer_ID VARCHAR2(50),
  Cakes_ID VARCHAR2(50),
  Cake_SHOP_id VARCHAR2(50),
  Quantity INT,
  Description_ VARCHAR2(50),
  PRIMARY KEY (Order_ID)
);
```

Task completed in 0,066 seconds

```
Table ORDER_ created.
```

insert into Order_ (Order_ID, Customer_ID, Cakes_ID, Cake_SHOP_id, Quantity, Description_) values ('64.137.66.0', '178.170.235.184/27', '4548438451', '4412', 1, 'nodesc');.....

	ORDER_ID	CUSTOMER_ID	CAKES_ID	CAKE_SHOP_ID	QUANTITY	DESCRIPTION_
1	64.137.66.0	178.170.235.184/28	4548438452	4413	3	nodesc
2	65.137.66.0	179.170.235.184/28	4548438453	4414	3	no strawberries
3	66.137.66.0	179.170.235.184/29	4548438454	4415	5	(null)
4	67.137.66.0	179.170.235.184/29	4548438454	4415	2	(null)
5	68.137.66.0	179.170.235.184/30	4548438454	4415	1	(null)
6	69.137.66.0	179.170.235.184/31	4548438455	4415	1	(null)
7	70.137.66.0	179.170.235.184/31	4548438456	4415	2	no strawberries
8	71.137.66.0	179.170.235.184/32	4548438457	4415	1	no kiwi
9	72.137.66.0	179.170.235.184/32	4548438460	4415	1	no kiwi
10	73.137.66.0	179.170.235.184/33	4548438460	4415	6	(null)
11	74.137.66.0	179.170.235.184/34	4548438459	4415	2	(null)
12	75.137.66.0	179.170.235.184/34	4548438458	4415	1	(null)
13	76.137.66.0	179.170.235.184/35	4548438457	4415	2	(null)
14	77.137.66.0	179.170.235.184/36	4548438456	4415	1	(null)
15	78.137.66.0	179.170.235.184/37	4548438455	4415	1	(null)
16	79.137.66.0	179.170.235.184/38	4548438454	4415	3	(null)
17	80.137.66.0	179.170.235.184/39	4548438454	4415	1	(null)
18	81.137.66.0	179.170.235.184/40	4548438453	4415	7	(null)
19	82.137.66.0	179.170.235.184/41	4548438452	4415	1	(null)
20	83.137.66.0	179.170.235.184/42	4548438453	4415	1	(null)

Table 3

```
create table Cakes (
    Cakes_id VARCHAR2(50),
    price INT,
    Ingredients VARCHAR2(50),
    Cake_size VARCHAR2(50),
    cake_name VARCHAR2(50),
    flavor VARCHAR2(50)
);
```

```
Table CAKES created.
```

insert into Cakes (Cakes_id, price, Ingredients, cake_size, cake_name, flavor) values (4548438455, 6800, 'callebault', 'M', 'woopie_pie', 'chocolate');.....

	CAKES_ID	PRICE	INGREDIENTS	CAKE_SIZE	CAKE_NAME	FLAVOR
1	4548438452	5499	cheese	M	cheesecake	curd taste
2	4548438453	7000	cheese	L	cheesecake	curd taste
3	4548438454	8500	cream cheese	L	banana cake	yogurt taste
4	4548438455	6800	callebault	M	woopie pie	chocolate
5	4548438456	5700	coconut	S	new york cheesecake	bounty
6	4548438457	7200	walnut	M	carrot gold	walnut
7	4548438458	4500	gluten	S	capcakes	vanilla
8	4548438459	9000	gluten-free	M	sponge	vanilla
9	4548438460	6800	strawberry	M	red velvet	strawberry

Table 4

```
create table Payment (
    Payment_ID INT primary key,
    Customer_ID VARCHAR2(50),
    Order_ID VARCHAR2(50),
    Payment_amount INT,
    Payment_date DATE,
    Payment_method VARCHAR2(50),
    Payment_status VARCHAR2(50)
);
```

```
Table PAYMENT created.
```

insert into Payment (Payment_ID, Customer_ID, Order_ID, Payment_amount, Payment_date, Payment_method, Payment_status) values (10, '89.151.186.61/14', '30550', 17840, '13/04/2022', 'visa', 'paid');.....

	PAYMENT_ID	CUSTOMER_ID	ORDER_ID	PAYMENT_AMOUNT	PAYMENT_DATE	PAYMENT_METHOD	PAYMENT_STATUS
1	1	89.151.186.61/7	30559	13500	10.04.22	mastercard	paid
2	2	89.151.186.61/8	30558	6800	10.04.22	halyk	unpaid
3	3	89.151.186.61/9	30557	9500	11.04.22	kaspi	in progress
4	4	89.151.186.61/10	30556	25000	11.04.22	kaspi	paid
5	5	89.151.186.61/11	30555	10500	10.04.22	cash	paid
6	6	89.151.186.61/12	30554	7800	12.04.22	QR	unpaid
7	7	89.151.186.61/13	30553	7800	12.04.22	QR	paid
8	8	89.151.186.61/14	30552	9200	12.04.22	cash	paid
9	9	89.151.186.61/15	30551	8700	13.04.22	visa	in progress
10	10	89.151.186.61/14	30550	17840	13.04.22	visa	paid

TABLE 5

```
create table Cake_Shop (
    Cake_shop_ID INT PRIMARY KEY,
    Shop_name VARCHAR2(50),
    Contact_no VARCHAR2(50),
    Email_address VARCHAR2(50),
    Location VARCHAR2(50)
);

Table CAKE_SHOP created.
```

INSERT INTO CAKE_SHOP (CAKE_SHOP_ID, SHOP_NAME, CONTACT_NO, EMAIL_ADDRESS, LOCATION) VALUES (6, 'QULPYNAI', '884-295-5545', 'GVITET0@DOT.GOV', 'ROZYBAKIYEVA');

CAKE_SHOP_ID	SHOP_NAME	CONTACT_NO	EMAIL_ADDRESS	LOCATION
1	Qulpynai	884-295-5540	gvitet0@dot.gov	Saina
2	Qulpynai	884-295-5541	gvitet0@dot.gov	Rozybakiyeva
3	Qulpynai	884-295-5542	gvitet0@dot.gov	Seifullina
4	Qulpynai	884-295-5543	gvitet0@dot.gov	Aksai
5	Qulpynai	884-295-5544	gvitet0@dot.gov	Shaimerden
6	Qulpynai	884-295-5545	gvitet0@dot.gov	Foryou

TABLE 6

```
create table Staff (
    Staff_ID INT PRIMARY KEY,
    Staff_name VARCHAR(50),
    Contact_no VARCHAR(50),
    Email_address VARCHAR(50),
    Shift VARCHAR(50),
    Designation VARCHAR(50)
);

Table STAFF created.
```

INSERT INTO STAFF (STAFF_ID, STAFF_NAME, CONTACT_NO, EMAIL_ADDRESS, SHIFT, DESIGNATION) VALUES (11, 'SHELLI', '616-152-6916', 'SSNASSELLA@ANSWERS.COM', 'WEDNESDAY', 'SELLER');

STAFF_ID	STAFF_NAME	CONTACT_NO	EMAIL_ADDRESS	SHIFT	DESIGNATION
1	Amity	323-722-3364	aallsworth0@wordpress.com	Monday	Grocery
2	Giffy	859-731-7314	gdragonette1@imageshack.us	Monday	Delivery
3	Bard	703-924-8855	bcauthra2@exblog.jp	Monday	Manager
4	Hatti	593-261-1558	hfoskett3@dion.ne.jp	Monday	Manager
5	Ashly	141-328-2124	abrumhead4@marriott.com	Tuesday	Seller
6	Tessa	377-977-6142	tdemsey5@springer.com	Tuesday	Cashier
7	Glynda	219-637-9401	gsewall6@yellowpages.com	Wednesday	Cashier
8	Jany	316-483-0561	jheninghem7@mysql.com	Wednesday	Grocery
9	Fanchon	455-993-3720	fbenneyworth8@google.ru	Thursday	Delivery
10	Carlos	411-987-6375	cminichi9@ning.com	Thursday	Delivery
11	Shelli	616-152-6916	ssnassella@answers.com	Wednesday	Seller
12	Ernest	214-981-4588	eblizardb@redcross.org	Friday	Grocery
13	Karlik	705-593-0145	kmolohanc@spotify.com	Friday	Manager
14	Gardiner	338-683-1975	gpennicardd@sitemeter.com	Thursday	Manager
15	Biddy	372-248-2978	bdrachee@illinois.edu	Saturday	Grocery

TABLE 7

```
create table Admin (
    Admin_ID INT PRIMARY KEY,
    Admin_name VARCHAR(50),
    Contact_no VARCHAR(50),
    Email_address VARCHAR(50)
);

Table ADMIN created.
```

INSERT INTO ADMIN (ADMIN_ID, ADMIN_NAME, CONTACT_NO, EMAIL_ADDRESS) VALUES (3, 'KENON', '109-946-1885', 'KSTILLWELL2@SBWIRE.COM');

	ADMIN_ID	ADMIN_NAME	CONTACT_NO	EMAIL_ADDRESS
1	3	Kenon	109-946-1885	kstillwell2@sbwire.com
2	2	Avrit	861-894-9203	agreim1@tripadvisor.com

PL/SQL part

1)

```
select* from customer;

CREATE OR REPLACE PROCEDURE group_by_occasion
IS
BEGIN
  FOR c IN (SELECT occasion, COUNT(*) AS count FROM Customer GROUP BY occasion)
  LOOP
    DBMS_OUTPUT.PUT_LINE(c.occasion || ': ' || c.count);
  END LOOP;
END;
/
```

Script Output | Query Result

Task completed in 0,135 seconds

```
Procedure GROUP_BY_OCCASION compiled
```

2)

```
EXECUTE group_by_occasion;
```

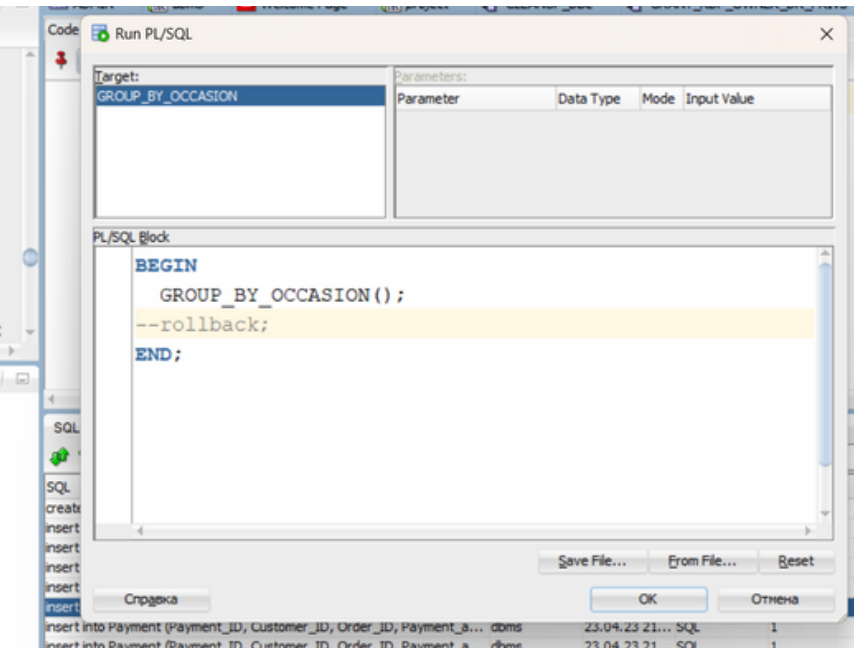
Script Output | Query Result

Task completed in 0,044 seconds

```
PL/SQL procedure successfully completed.
```

- The procedure retrieves data from "Customer" and groups it by the "occasion" column
- The SELECT statement inside the FOR loop counts the number of occurrences of each unique value of the "occasion" column and assigns it "count". The results of the query are stored in a cursor variable named "c".
- Useful for reporting or analysis purposes.

3)




```
Connecting to the database project.
New_year: 3
Documentary: 1
Action|Adventure|Comedy|Romance: 1
Drama|Romance: 1
Action|Comedy: 1
Comedy|Drama|Romance: 1
anniversary: 1
Adventure|Comedy|Musical: 1
Comedy|Drama: 1
Comedy: 1
Drama: 3
Birthday: 1
Action|Drama|Thriller: 1
Horror|Sci-Fi|Thriller: 1
Birtday: 2
Process exited.
Disconnecting from the database project.
```

PL/SQL part

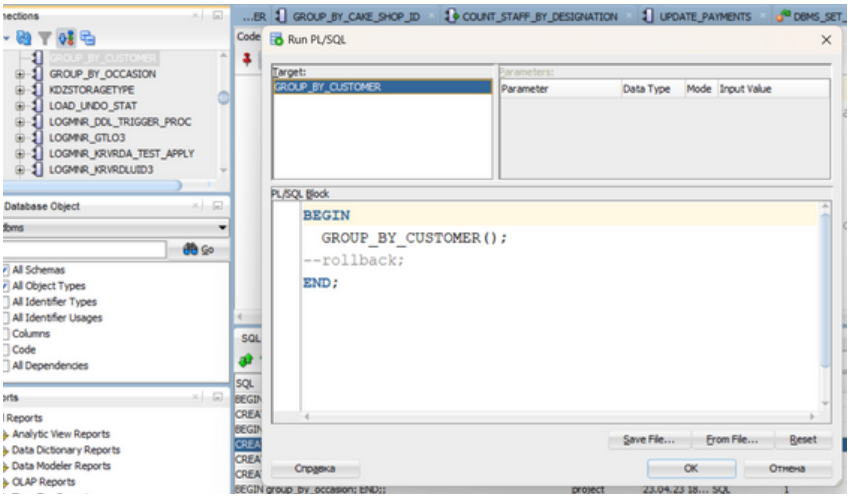
```
CREATE OR REPLACE PROCEDURE group_by_customer IS
  residence_column Customer.residence%TYPE;
  occasion_column Customer.occasion%TYPE;
  customer_count NUMBER;
BEGIN
  -- Select and group the data, and store the results in variables
  SELECT residence, occasion, COUNT(*)
  INTO residence_column, occasion_column, customer_count
  FROM Customer
  GROUP BY residence, occasion;

  -- Display the results
  DBMS_OUTPUT.PUT_LINE('Residence: ' || residence_column);
  DBMS_OUTPUT.PUT_LINE('Occasion: ' || occasion_column);
  DBMS_OUTPUT.PUT_LINE('Customer Count: ' || customer_count);
END;
/
```

Script Output | Query Result | Script Output 1

Task completed in 0,026 seconds

```
Procedure GROUP BY CUSTOMER compiled
```

- **The purpose is to group the customers by their residence and occasion ,then count their number for each group. The result is stored in three variables: "residence_column," "occasion_column," and "customer_count."**
- **The "INTO" clause is used to store the result of the "SELECT" statement into the three variables declared earlier: "residence_column," "occasion_column," and "customer_count."**
- **The procedure uses the "DBMS_OUTPUT.PUT_LINE" function to display the result**
- **this procedure can be used to analyze customer data**
- **by grouping them based on their residence and**
- **occasion, which can help businesses understand**
- **customer behavior and preferences.**



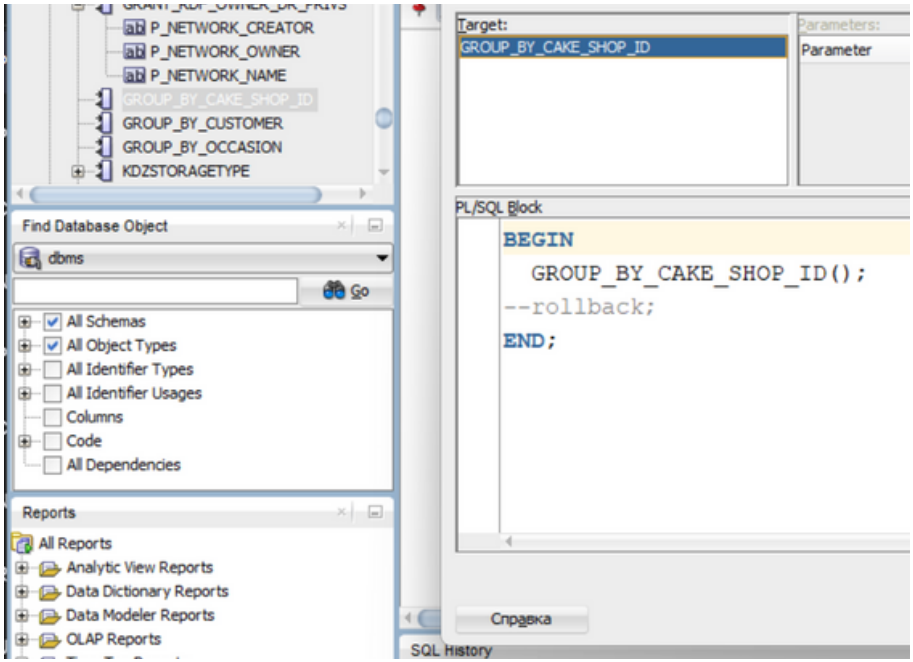

```
Customer Count: 1
Residence: 65310 Village Park
Occasion: New_year
Customer Count: 1
Residence: 38 Tony Court
Occasion: Birtday
Customer Count: 1
Residence: 6 Village Green Point
Occasion: Action|Adventure|Comedy|Romance
Customer Count: 1
Residence: 31857 Dottie Avenue
Occasion: Birtday
Customer Count: 1
Residence: 85200 Lerdahl Alley
Occasion: New_year
Customer Count: 1
Residence: 7340 Manitowish Park
Occasion: Action|Drama|Thriller
Customer Count: 1
Residence: 77 Grover Road
Occasion: Comedy|Drama|Romance
```

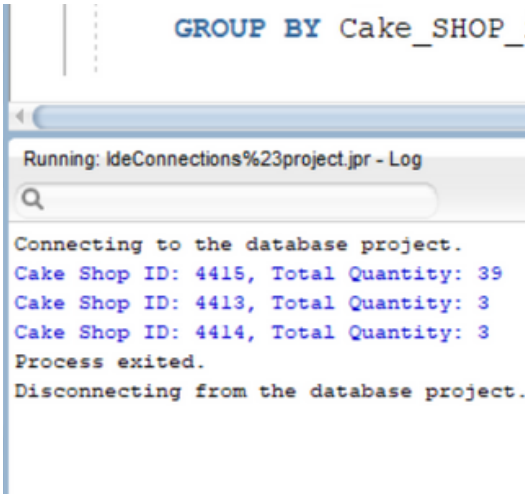
Messages | Statements | Logging Page | Compiler

PL/SQL part

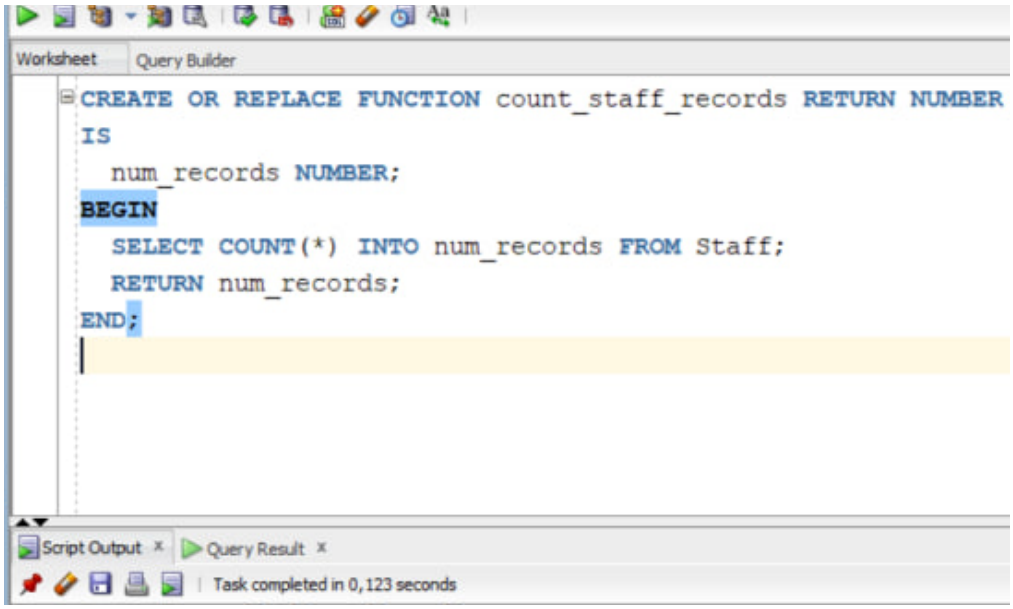
```
create or replace NONEDITIONABLE PROCEDURE group_by_cake_shop_id IS
  CURSOR order_cursor IS
    SELECT Cake_SHOP_id, SUM(Quantity) AS Total_Quantity
    FROM Order_
    GROUP BY Cake_SHOP_id;

  v_cake_shop_id Order_.Cake_SHOP_id%TYPE;
  v_total_quantity Order_.Quantity%TYPE;
BEGIN
  OPEN order_cursor;
  LOOP
    FETCH order_cursor INTO v_cake_shop_id, v_total_quantity;
    EXIT WHEN order_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('Cake Shop ID: ' || v_cake_shop_id || ', Total Quantity: ' || v_total_quantity);
  END LOOP;
  CLOSE order_cursor;
END;
```

- **Defines a cursor named "order_cursor" that retrieves the Cake_SHOP_id and Total_Quantity for each order from the Order_ table, grouped by Cake_SHOP_id.**
- **Declares two variables v_cake_shop_id and v_total_quantity to hold the values fetched by the cursor.**
- **Opens the cursor using the OPEN statement.**
- **Loops through the result set of the cursor using the FETCH statement, and assigns the retrieved values to the variables v_cake_shop_id and v_total_quantity.**
- **The LOOP continues until all the rows from the cursor are fetched, which is determined by the condition EXIT WHEN order_cursor%NOTFOUND.**
- **For each row, it prints the Cake Shop ID and the Total Quantity using the DBMS_OUTPUT.PUT_LINE statement.**
- **Closes the cursor using the CLOSE statement.**

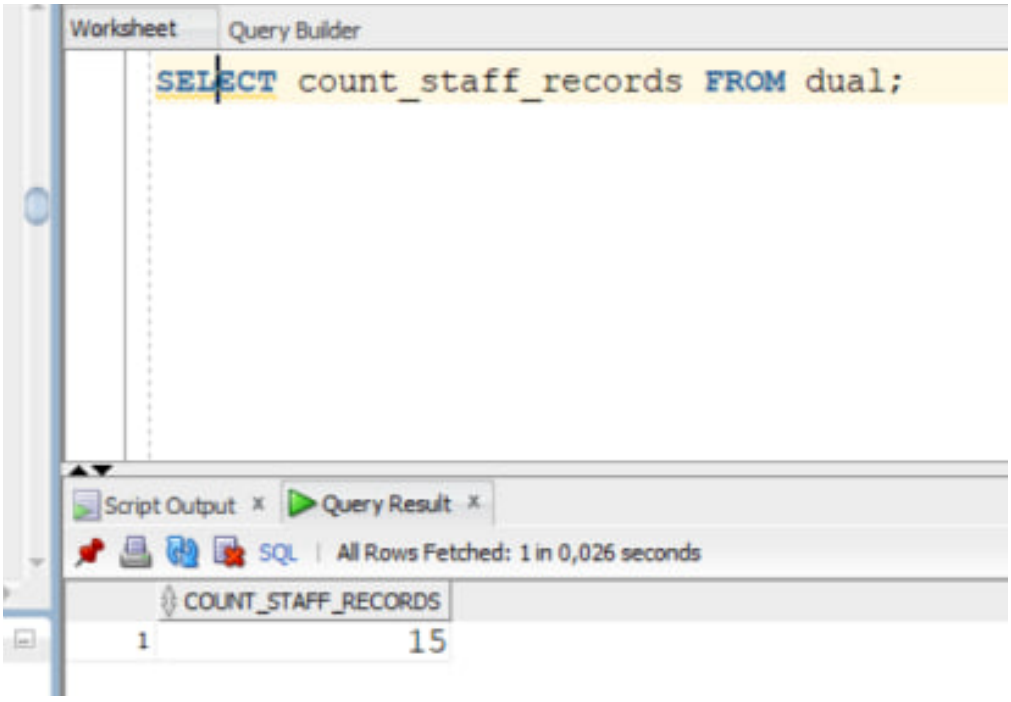




PL/SQL part



- **The "SELECT COUNT(*) INTO num_records FROM Staff" statement counts the number of rows in the "Staff" table and assigns the result to the variable "num_records". The "RETURN" statement then returns this value as the result of the function.**



the following SQL statement would return the number of staff records:. "dual" table is a special table in Oracle that is often used in SQL statements that don't actually query any tables. In this case, we're using it to call the "count_staff_records" function and return its result.

PL/SQL part

Worksheet Query Builder

```
CREATE OR REPLACE FUNCTION count_staff_by_designation
  (p_designation IN VARCHAR2)
RETURN VARCHAR2
IS
  v_count NUMBER := 0;
  v_result VARCHAR2(100);
BEGIN
  SELECT COUNT(*) INTO v_count FROM Staff WHERE Designation = p_designation;

  IF v_count > 0 THEN
    v_result := 'There are ' || v_count || ' staff members with the designation ' || p_designation;
  ELSE
    v_result := 'There are no staff members with the designation ' || p_designation;
  END IF;

  RETURN v_result;
END;
```

Script Output | Query Result

Task completed in 0,043 seconds

```
Function COUNT_STAFF_RECORDS compiled
```

- **The function takes one input parameter "p_designation" of type VARCHAR2 which represents the designation of the staff members to be counted.**
- **The function returns a string message of type VARCHAR2 that displays the number of staff members with the input designation. If there are no staff members with the input designation, the message indicates that there are none.**
- **The function is not editionable, meaning that it cannot be modified or replaced by another version in the same edition.**

PL/SQL Block

```
DECLARE
  v_result VARCHAR2(100);
BEGIN
  v_result := count_staff_by_designation('Manager');
  DBMS_OUTPUT.PUT_LINE(v_result);
END;
```

Save File... From File... Reset

Справка OK Отмена

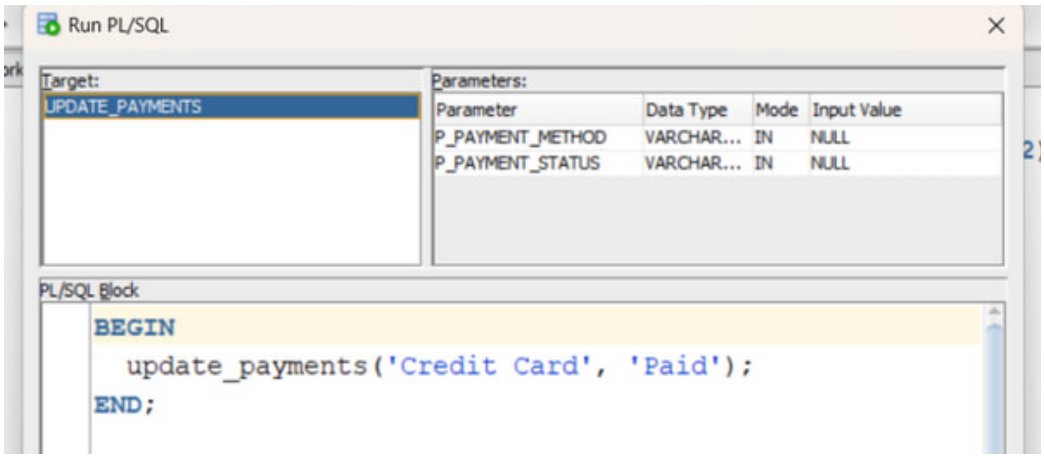
allows you to create a schedule for employees according to their designation

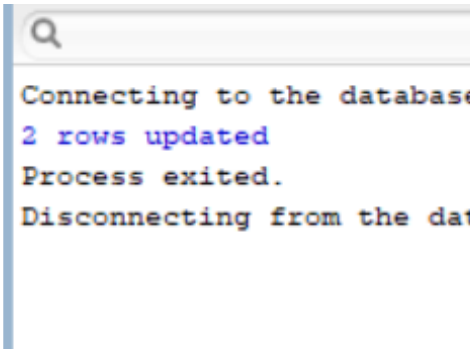
```
Connecting to the database project.
There are 4 staff members with the designation Manager
Process exited.
Disconnecting from the database project.
```

PL/SQL part

```
Worksheet  Query Builder
create or replace NONEDITIONABLE PROCEDURE update_payments
  (p_payment_method IN VARCHAR2, p_payment_status IN VARCHAR2)
IS
  v_count NUMBER;
BEGIN
  UPDATE Payment SET Payment_status = p_payment_status
  WHERE Payment_method = 'kaspi';

  v_count := SQL%ROWCOUNT;

  DBMS_OUTPUT.PUT_LINE(v_count || ' rows updated');
END;
```

```
Procedure UPDATE_PAYMENTS compiled
```





- **This procedure takes two input parameters "p_payment_method" and "p_payment_status" of type VARCHAR2. The procedure updates the "Payment_status" column of the "Payment" table based on the input parameters.**

- **The procedure begins by declaring a local variable "v_count" of type NUMBER. It then executes an UPDATE statement on the "Payment" table, setting the "Payment_status" column to the value of "p_payment_status" for all rows where the "Payment_method" column is equal to "p_payment_method".**

- **The variable "v_count" is then assigned the number of rows that were updated using the SQL%ROWCOUNT system variable. Finally, the procedure uses the DBMS_OUTPUT.PUT_LINE procedure to print the number of rows updated to the console.**

```
select* from payment;
```

Script Output | Query Result

SQL | All Rows Fetched: 10 in 0,005 seconds

	PAYMENT_ID	CUSTOMER_ID	ORDER_ID	PAYMENT_AMOUNT	PAYMENT_DATE	PAYMENT_METHOD	PAYMENT_STATUS
1	1	89.151.186.61/7	30559	13500	10.04.22	mastercard	paid
2	2	89.151.186.61/8	30558	6800	10.04.22	halyk	unpaid
3	3	89.151.186.61/9	30557	9500	11.04.22	kaspi	Paid
4	4	89.151.186.61/10	30556	25000	11.04.22	kaspi	Paid
5	5	89.151.186.61/11	30555	10500	10.04.22	cash	paid
6	6	89.151.186.61/12	30554	7800	12.04.22	QR	unpaid
7	7	89.151.186.61/13	30553	7800	12.04.22	QR	paid
8	8	89.151.186.61/14	30552	9200	12.04.22	cash	paid
9	9	89.151.186.61/15	30551	8700	13.04.22	visa	in progress
10	10	89.151.186.61/14	30550	17840	13.04.22	visa	paid

PL/SQL part

- We have used trigger and added user-defined exception. trigger was designed to check the length of the Description_ field before an insert or update operation is performed on the Order_ table. If the length of the Description_ field is less than 5 characters, the trigger raises an error with a message indicating that the Description_ field must be at least 5 characters long.
- To create trigger in Oracle, we have created new user:

```
Worksheet  Query Builder
CREATE USER c##myuser IDENTIFIED BY mypassword;

GRANT CREATE SESSION TO c##myuser;
```

```
User C##MYUSER created.


Grant succeeded.
```

```
ols  Window  Help
Welcome Page  new  ORDER_
rksheet  Query Builder
CREATE OR REPLACE TRIGGER order_description_check
BEFORE INSERT OR UPDATE ON Order_
FOR EACH ROW
BEGIN
  IF LENGTH(:NEW.Description_) < 5 THEN
    RAISE_APPLICATION_ERROR(-20001, 'Description must be at least 5 characters long.');
  END IF;
END;
```

```
Task completed in 0,181 seconds

Trigger ORDER_DESCRIPTION_CHECK compiled
```

- The IF statement is used to test whether the length of the Description_ field is less than 5 characters. If it is, the RAISE_APPLICATION_ERROR procedure is called with an error code of -20001 and an error message that specifies that the Description_ field must be at least 5 characters long.
- Then we have updated description attribute with following code:

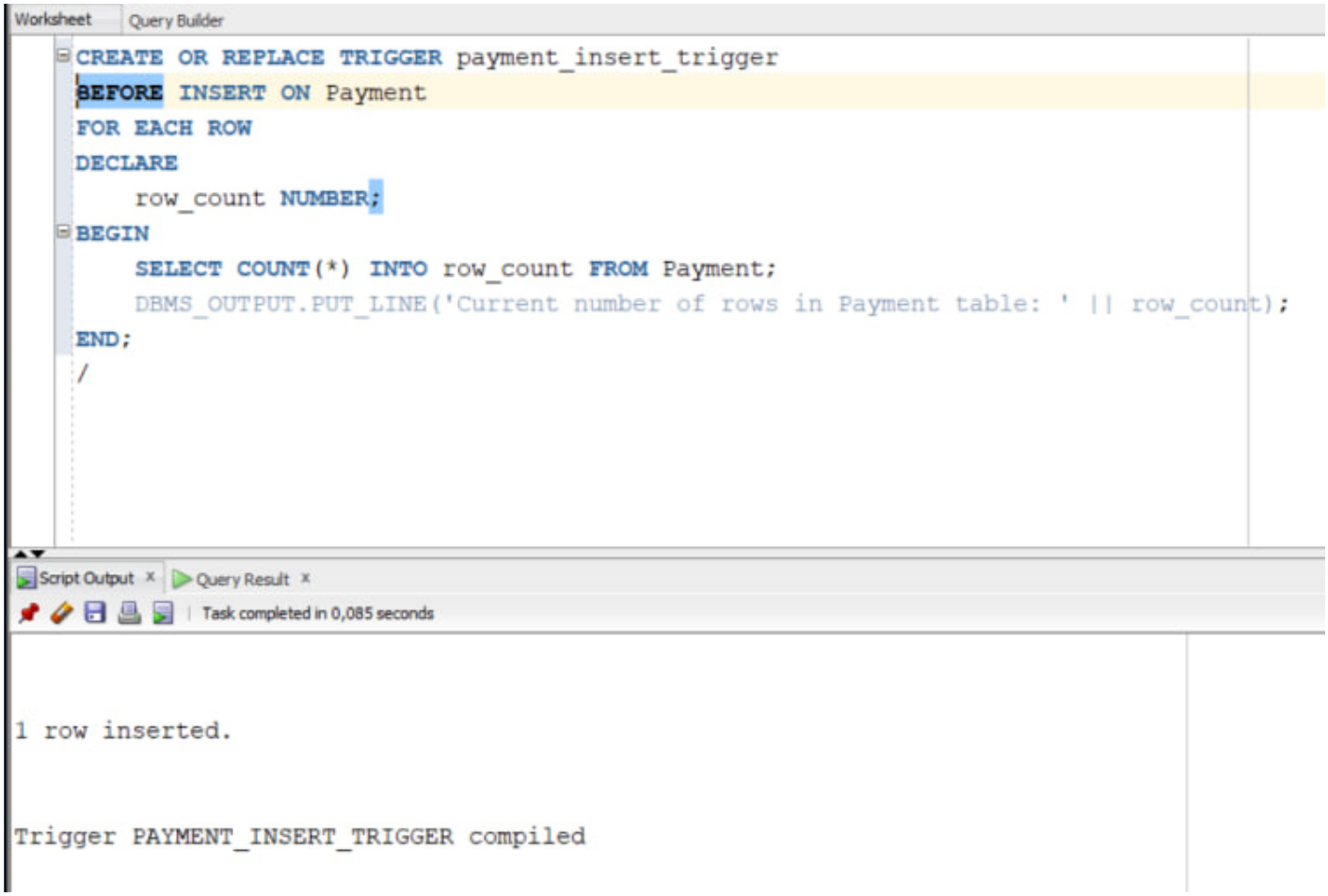
```
Worksheet  Query Builder
update order_
set description_='abc'
where cakes_id=4548438452;
```

Here is the result:

```
Task completed in 0,061 seconds
update order_
set description_='abc'
where cakes_id=4548438452
Error at Command Line : 1 Column : 8
Error report -
SQL Error: ORA-20001: Description must be at least 5 characters long.
ORA-06512: на  "C##MYUSER.ORDER_DESCRIPTION_CHECK", line 3
ORA-04088: ошибка во время выполнения триггера 'C##MYUSER.ORDER_DESCRIPTION_CHECK'
```

PL/SQL part

This trigger is designed to provide information about the number of rows in the Payment table when a new row is inserted into the table.



- **The trigger is set to fire BEFORE INSERT events on the Payment table, which means that the trigger code will be executed before any new rows are inserted into the Payment table.**
- **Therefore, we have insertd some new dara to Payment table**