Chocolate Shop

Made by: Maxim Kim
ID: 210103163
Subject: DBMS 2
Lecture Instructor: Azamat Serek

Introduction:

Midterm theme: Chocolate Shop.
In the project I have:

- ER Diagram
- 9 Entities (Shop, Categories, Products, Clients, Orders, Employees, Yandex, Couriers, Reviews)
- All tables follows normal forms (1NF, 2NF, 3NF)
- Procedure, that shows products by particular category
- Function, that shows number of records (Shows how many products belong to particular category)
- Procedure, which deletes orders of particular client, and shows how many rows were deleted
- Trigger, that disallows enter review_rating in the Reviews table more than 5 or less than 1
- Trigger, which shows count of rows before inserting new one
- Examples of how all procedures, functions and triggers work

---

ER Diagram:

ER Diagram was made in the Miro
Here is the link to the board:
https://miro.com/app/board/uXjVP8ut8bQ=/?share_link_id=392133902318

You can see the picture of ERD bellow as well, but for better consideration it is better to follow the link

Relationships:
Products - Clients (M-M): One client can buy many products, one product can be bought by many clients
Clients - Review (O-M): One client can leave many reviews, but one exactly review can be left only by one client

Clients - Orders (O-M): Clients can order many times, but one exactly order belongs to one exactly client
Orders - Couriers (O-O): One order can be delivering by one courier
Shop - Employees (O-M): Shop can have many employees, but one employee works in the one shop
Products - Categories (O-M): One category can contain many products, but one product can be belong to only one category

---

Whole process:

Remarks:

Entities ***Shop*** and ***Yandex Delivery*** are just for better understanding and for better connection in ERD, without these entities I still have 7 entities

PK - Primary Key

FK - Foreign Key

1. There are several Internet chocolate shops in Almaty, which has only *Shop ID* as a Primary Key and *Shop Name* attributes

2. Clients can sign up, they input *Name, Surname, Email, Street Address, House Number and Phone Number* after registration each client get *Client ID (PK)*

3. Products in the shop have attributes: *Product ID (PK), Product Name (Milka, Alpen Gold etc), Product Description, Product Price and Category ID (FK)*

4. Each product assigned to some category, category has attributes: *Category ID (PK) and Category Name (White chocolate, Bitter Chocolate etc)*

5. There are many employees, each employee has *Employee ID (PK), Employee Name, Employee Surname, Employee Street Address, Employee House Number, Employee Phone Number and Position (Manager, SMM Manager,Shop Assistant)*

6. Clients can order products. Order has been created and has attributes: *Order ID (PK), Client ID (FK), Order Sum Price, Order Status (Processing, Shipping, Delivered)*

Status changes manually by manager. (For example, status is shipping, but when client get the order, manager get message about that and change the status to 'Delivered')

7. Chocolate shop cooperates with Yandex Delivery, so after the client pays, the courier from Yandex will deliver the order. Couriers have attributes: *Courier ID (PK), Order ID (FK)*

Via Order ID we can get Destination Address (Client Address), but Client Address is in another table, so this structure does not violates Normal Forms

8. After receiving chocolate, clients can leave the review. Review consists of: *Review ID (PK), Client ID (FK), Review Text, Review Rating (1 - 5)*

All entities
Explanations why the structure follows normal forms:

9 entities:

1. **Shop (<u>shop_id (PK)</u>, shop_name)**

1NF: Each cell in the table is atomic value
2NF: Non-key column depends on the whole primary key
(<u>shop_id (PK)</u> -> shop_name)
3NF: There isn't any non-key element that depends on another non-key element

2. Products (<u>product_id (PK)</u>, category_id (FK), product_name, product_description, product_price)

1NF: Each cell in the table is atomic value

2NF: Non-key columns depend on the whole primary key
(<u>product_id (PK)</u> -> category_id (FK), product_name, product_description, product_price)

3NF: There isn't any non-key element that depends on another non-key element. Here category_id is FK, and by category_id we can find category_name, but category_name is in another table, where category_id is PK. So the structure does not violate normal forms

3. Categories (<u>category_id (PK)</u>, category_name)

1NF: Each cell in the table is atomic value

2NF: Non-key column depend on the whole primary key
(<u>category_id (PK)</u> -> category_name)

3NF: There isn't any non-key element that depends on another non-key element.

4. Clients (<u>client_id (PK)</u>, client_name, client_surname, client_mail, client_street, client_house, client_phonenumber)

1NF: Each cell in the table is atomic value
(Client Address splitted to Street Name and House Number)

2NF: Non-key columns depend on the whole primary key
(<u>client_id (PK)</u> -> client_name, client_surname, client_mail, client_street, client_house, client_phonenumber)

3NF: There isn't any non-key element that depends on another non-key element.

5. Orders (order_id (PK), client_id (FK), order_price, order_status)

1NF: Each cell in the table is atomic value

2NF: Non-key columns depend on the whole primary key
(order_id (PK) -> client_id (FK), order_price, order_status)

3NF: There isn't any non-key element that depends on another non-key element. Like in the Products table, all information that we can find by *client_id* is located in another table.

6. Employees (employee_id (PK), employee_name, employee_surname, employee_position, employee_mail, employee_street, employee_house, employee_phonenumber)

1NF: Each cell in the table is atomic value
(Employee Address splitted to Street Name and House Number)

2NF: Non-key columns depend on the whole primary key
(employee_id (PK) -> employee_name, employee_surname, employee_position, employee_mail, employee_street, employee_house, employee_phonenumber)

3NF: There isn't any non-key element that depends on another non-key element.

7. Yandex Delivery (name (PK))

1NF: Cell in the table is atomic value

2NF: In this table there is only one column

3NF: In this table there is only one column

8. Couriers (courier_id (PK), order_id (FK))

1NF: Each cell in the table is atomic value

2NF: Non-key column depends on the whole primary key
(courier_id (PK) -> order_id (FK))

3NF: There isn’t any non-key element that depends on another non-key element. Like in the Products table, all information that we can find by *order_id*, for example information about a client's address, is located in another table.

9. Reviews (review_id (PK), client_id (FK), review_text, review_rating)

1NF: Each cell in the table is atomic value

2NF: Non-key columns depend on the whole primary key (review_id (PK) -> client_id (FK), review_text, review_rating)

3NF: There isn’t any non-key element that depends on another non-key element. Like in the Products table, all information that we can find by *client_id*, for example information about a client's name, is located in another table.

Explanations of PL SQL coding part:

Procedure, that shows products by particular category

```
CREATE OR REPLACE PROCEDURE getProductsByCategory (
  c_category_id IN Categories.category_id%TYPE,
  c_cur OUT SYS_REFCURSOR
) AS
BEGIN
  OPEN c_cur FOR
    SELECT C.category_name, P.product_name, P.product_price
    FROM Categories C
    JOIN Products P ON P.category_id = C.category_id
    WHERE P.category_id = c_category_id
    GROUP BY C.category_name, P.product_name, P.product_price;
END;
```

Here we create or replace procedure and named it getProductsByCategory

It accepts category id from table Categories
We also declare cursor with type SYS_REFCURSOR, this cursor will keep information from procedure and then we can use this information when we called procedure from another code
We open cursor and put there category_name from table Categories and product_name, product_price from table Products
We use JOIN to connect two tables by category_id
We choose only tuples where category_id are same
At the end we group information by C.category_name, P.product_name, P.product_price

---
Example:

```
DECLARE
  v_cur SYS_REFCURSOR;
  v_category_name Categories.category_name%TYPE;
  v_product_name Products.product_name%TYPE;
  v_product_price Products.product_price%TYPE;
BEGIN
  getProductsByCategory(1, v_cur);

  LOOP
    FETCH v_cur INTO v_category_name, v_product_name, v_product_price;
    EXIT WHEN v_cur%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(v_category_name || ', ' || v_product_name || ', ' ||
v_product_price);
  END LOOP;

  CLOSE v_cur;
END;
```

Here we declare all variables: cursor, category_name, product_name and product_price
In the BEGIN block we call procedure getProductsByCategory. Send category_id 1, and send cursor which will keep information from the procedure
In the loop we pull out data from cursor and put it in the variables category_name, product_name and product_price
Then we output data on the screen
Loop will stop, when cursor became empty

Function, that shows number of records

```
CREATE OR REPLACE FUNCTION productCountOfCategory (
   c_category_id IN Categories.category_id%TYPE
) RETURN NUMBER AS
   v_count NUMBER;
BEGIN
   SELECT COUNT(*)
   INTO v_count
   FROM Products P
   JOIN Categories C ON P.category_id = C.category_id
   WHERE C.category_id = c_category_id;

   RETURN v_count;
END;
```

Here we create or replace function productCountOfCategory
It accepts category-id from table Categories and returns variable v_count of type Number
Then we got count of products and put it into variable v_count
We connect two tables: Categories and Products and then we choose only those tuples, where category id is equal to the id, that user sent when called the function

---

Example:

```
DECLARE
   v_count NUMBER;
BEGIN
   v_count := productCountOfCategory(3);
   dbms_output.put_line(v_count);
END;
```

Here we declare variable v_count
We call the function and send category_id 3, function returns count of products and we save this number in the v_count
Then we output variable v_count

Procedure, which deletes orders of particular client, and shows how many rows were deleted

```
CREATE OR REPLACE PROCEDURE deleteOrdersOfClient (
   c_client_id IN Clients.client_id%TYPE
) AS
```

```
BEGIN
   DELETE FROM Orders
   WHERE client_id = c_client_id;

   IF SQL%ROWCOUNT > 0 THEN
      DBMS_OUTPUT.PUT_LINE('Deleted ' || SQL%ROWCOUNT || '
orders, that make client with id: ' || c_client_id);
   ELSE
      DBMS_OUTPUT.PUT_LINE('Client with id ' || c_client_id || ' did not
order any chocolate');
   END IF;
END;
```

Here we create or replace procedure deleteOrdersOfClient
It accepts client_id from table Clients
Then we just delete rows from table Orders where client_id equals to the parameter that user sent when called the procedure
If count of deleted rows more than 0, procedure output how many rows were deleted
If there is no tuples with exact ‘client_id’, procedure output, that this client did not order chocolate

---
Example:

```
DECLARE
BEGIN
   deleteOrdersOfClient(1);
END;
```

Here we just call procedure and sent some parameter

Trigger, that disallows enter review_rating in the Reviews table more than 5 or less than 1

```
CREATE OR REPLACE TRIGGER appropriateRatingMark
BEFORE INSERT ON Reviews
FOR EACH ROW
DECLARE
BEGIN
   IF :NEW.review_rating < 1 OR :NEW.review_rating > 5 THEN
```

```
        RAISE_APPLICATION_ERROR(-20001, 'Rating mark cannot be less
than 1 or more than 5');
    END IF;
END;
```

Here we create or replace trigger appropriateRetingMark that works before inserting tuples in the table Reviews
Trigger checks the review_rating, and if it is less than 1 or more than 5 trigger raises exception

You can see that here I used another syntax, different from the one in the lecture. Firstly I wrote Exception Handling like in the lecture slides, everything worked, exception message appeared on the screen, however data was added to the table, so I read forums from the Internet and find solution via RAISE_APPLICATION_ERROR

Example:

```
INSERT INTO Reviews (client_id, review_text, review_rating)
VALUES (3, 'Tasty chocolate, but it takes too long for answering me in
instagram', 9)
```

Output:
ORA-20001: Rating mark cannot be less than 1 or more than 5

Trigger, which shows count of rows before inserting new one

```
CREATE OR REPLACE TRIGGER rowCount
BEFORE INSERT ON Clients
DECLARE
    row_count NUMBER;
BEGIN
    SELECT COUNT(*) INTO row_count FROM Clients;
    DBMS_OUTPUT.PUT_LINE('Current row count in the Clients table: ' ||
row_count);
END;
```

Here we create or replace trigger rowCount, which works before inserting tuples in the table clients

We declare variable row_count
Get count of tuples and put it into row_count
Then we just output how many rows in the table

Example:
```
INSERT INTO Clients (client_name, client_surname, client_mail,
client_street, client_house, client_phone)
VALUES ('Zhanibeck', 'Kupesbaev', 'zhanibek.kupesbaev@gmail.com',
'Shevchenko', 165, 87771110000)
```

Output:
Current row count in the Clients table: 4