WOMEN'S CLOTHING STORE

210103397 – Kozhamuratova Aizhan
210103469 – Kuralay Mukhtar
210103159 – Bergen Saule
210103190 – Ayana Bakytzhanova
210103038 – Ayala Narbekova

Description:

i) Introduction and database description.

Our project is designed for an online women's clothing store. The "women's clothing store" database contains information about sellers, buyers and couriers.

At the beginning, when a user registers, he chooses his category, to which one of the above subclasses belongs. Each type of user specifies its own address and card. Users register only by phone number.

In our database, a seller can sell many products. Products have own pid, sizes, category, price and sid which connects with seller.

Each customer has a shopping cart, where you can store different products that you want to buy.

Transactions will take correspond the shopping cart and stores all the information of the product that you are going to buy. After that, the information is delivered to the courier.

ii) What functions should the system perform?

- Allowing users to register an account with their personal information.
- Manage profile information, such as updating their name, address, and card details.
- To place an order for a product, specify your address for delivery and card details for payment.
- Allowing sellers to list their products for sale, providing product details such as category, price and size.
- Buyers can search for products available from sellers in the system.
- Find out the registration date for each user.

iii) Who are the end users?

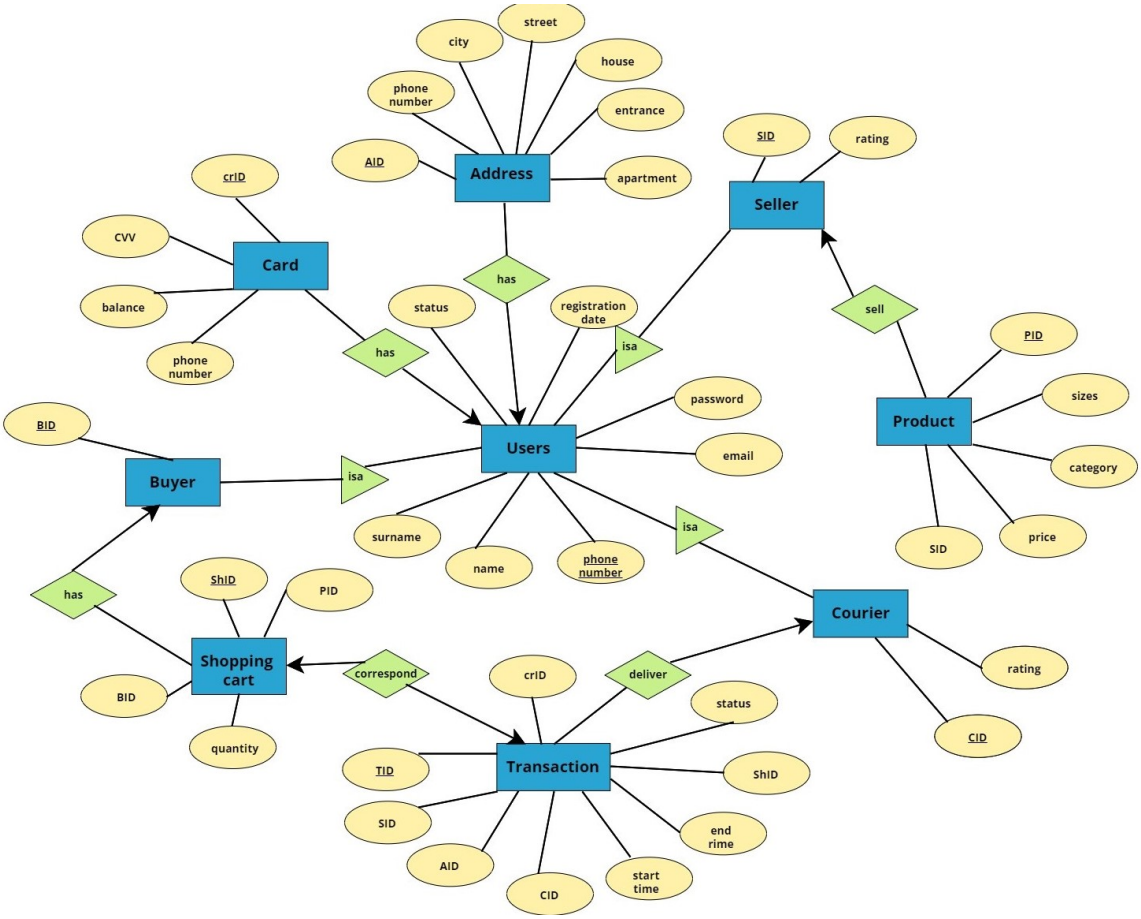
Sellers, Buyers and Couriers.

iv) Where did we get the idea for this project?

v) How will data obsolescence be handled?

ENTITY RELATIONSHIP DESIGN

The



"users" entity has the attributes "phone_number", "name", "surname", "email", "password", "registration_date" and "status". The user registers only by phone number. So, this is the key attribute. The status can be buyer, seller or courier. Users have subclasses: "buyer", "seller" and "courier".

In a one-to-many relationship between "user" and "card", it means that one user can have multiple cards, while each card belongs to only one user. "card" has the attributes crID, CVV, balance and phone_number.

In a one-to-many relationship between "user" and "address", it means that one user can have multiple addresses, while each address ID belongs to only one user. "address" has the attributes AID, phone_number, city, street, house, entrance and apartment.

In this one-to-many relationship between "seller" and "product", a seller (identified by SID) can sell multiple products, while each product (identified by PID) is associated with only one seller. The "seller" entity has a "raiting" attribute, which could represent the seller's rating. The "product" entity has attributes such as "size", "category", and "price" (which could represent the current price of the

product). The "category" attribute defines the type of clothing (for example, skirt, jacket, jeans, etc.)

In this one-to-many relationship between "buyer" and "shopping cart", a buyer (identified by BID) can have multiple shopping carts, and each shopping cart (identified by ShID) is uniquely associated with the buyer's account and can contain product ID and quantity of product.

In this one-to-one relationship between "shopping cart" and "transaction", each shopping cart(identified by ShID) is linked to a single transaction(identified by TID), and vice versa. Once the buyer proceeds to the checkout process and completes the purchase, the shopping cart is converted into a transaction, capturing all the relevant details of the purchase. The "transaction" entity has attributes such as "start time", "end time", "crID"(card ID), ShID, AID(address ID), SID(seller ID), status and "CID" (courier ID). The status of transaction can be declined, approved or pending.

In this many-to-one relationship between "transaction" and "courier", multiple transactions (identified by TID) can be associated with a single courier (identified by CID), who can be rated by users. The "Courier" entity represents individual couriers with attributes such as "CID" (which serves as the primary key for the courier) and "rating" (which represents the rating given by users).

NORMALIZATION

Table User

Key: phone_number

phone_number -> password, email, name, surname, status, registration_date

1NF: Each user will register with a phone number and this will be the primary key. And each user can have only one password, email, name, surname, status, registration_date.

2NF: All nonkey attributes are functionally dependent on the entire phone_number, so that's 2NF.

3NF: In the existing FD, the left hand side is the key, so it's 3NF.

Table Seller

Key: SID

Foreign Key: phone_number

phone_number -> SID, rating

SID -> phone_number, rating

1NF: Each seller can have only one phone_number, SID, rating.

2NF: Nonkey attribute rating is functionally dependent on the entire SID or phone_number.

3NF: In the existing FDs, the left hand side is the key or its right hand side is a key.

Table Buyer

Key: BID

Foreign Key: phone_number

phone_number -> BID

BID -> phone_number

1NF: Each buyer can have only one phone_number, BID

2NF: in 1NF and does not have nonkey attributes.

3NF: In the existing FDs, the left hand side is the key or its right hand side is a key.

Table Courier

Key: CID

Foreign Key: phone_number

phone_number -> CID, rating

CID -> phone_number, rating

1NF: Each courier can have only one phone_number, CID, rating.

2NF: Nonkey attribute rating is functionally dependent on the entire CID or phone_number.

3NF: In the existing FDs, the left hand side is the key or its right hand side is a key.

Table Card

Key: crID

crID -> CVV, balance, phone_number

1NF: Each card can have only one CVV, balance, phone_number

2NF: Nonkey attributes are functionally dependent on the entire crID.

3NF: In the existing FD, the left hand side is the key.

Table Address

Key: AID

AID -> phone_number, city, street, house, entrance, apartment

1NF: Each address ID can have only one phone_number, city, street, house, entrance, apartment.

2NF: Nonkey attributes are functionally dependent on the entire AID.

3NF: In the existing FD, the left hand side is the key.

Table Product

Key: PID

Foreign Key: SID

PID -> size, category, price, SID

1NF: Each address product can have only one category, price, SID.

2NF: Nonkey attributes are functionally dependent on the entire PID.

3NF: In the existing FD, the left hand side is the key.

Table Shopping_cart

Key: ShID

ShID -> PID, BID, quantity

1NF: Each shopping cart can have only one product ID, buyer and quantity.

2NF: Nonkey attributes are functionally dependent on the entire ShID.

3NF: In the existing FD, the left hand side is the key.

Table Transaction

Key: TID

TID -> SID, status, ShID, CID, AID, start_time, end_time, crID

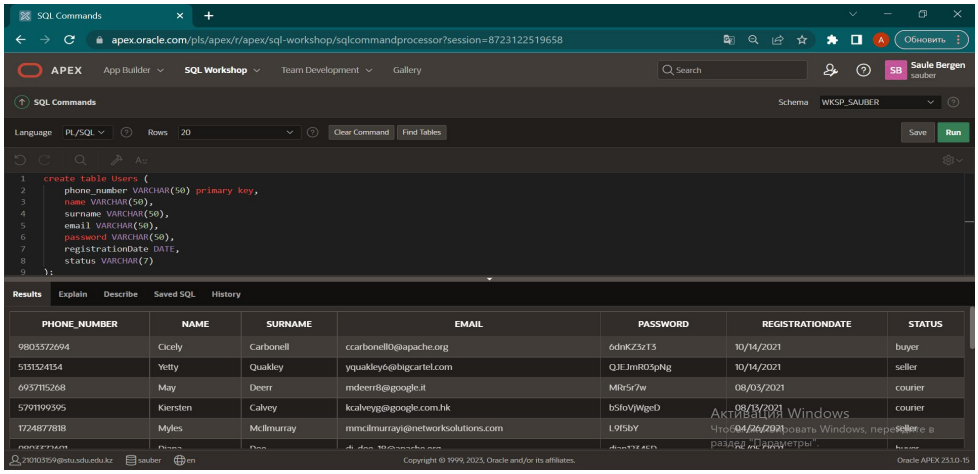
1NF: Each transaction can have only one SID, status, ShID, CID, AID, start_time, end_time, crID.

2NF: Nonkey attributes are functionally dependent on the entire TID.

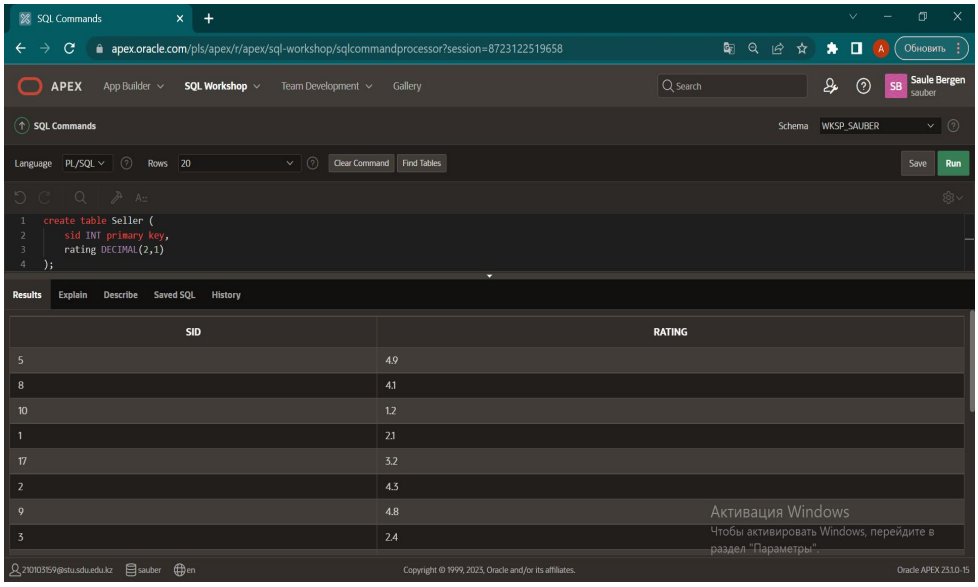
3NF: In the existing FD, the left hand side is the key.

CODING PART

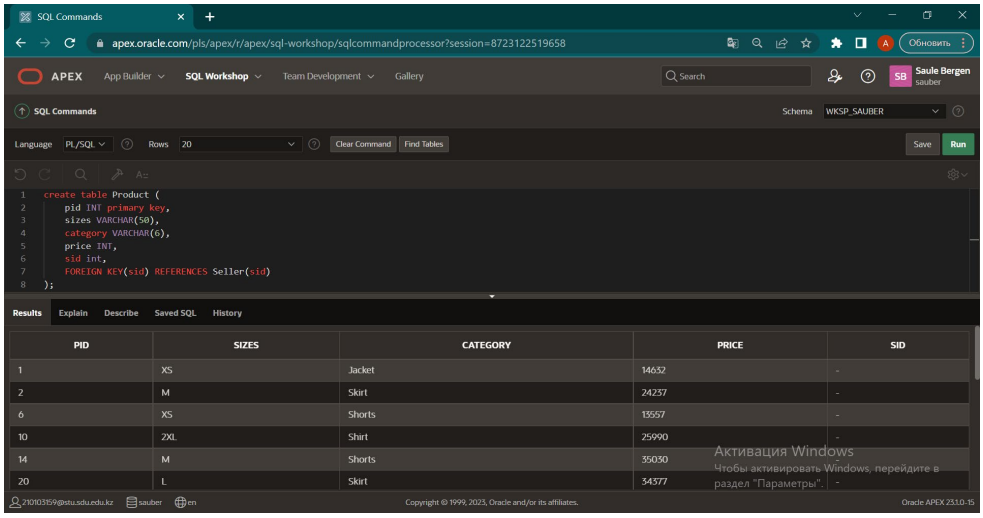
1) Create table Users



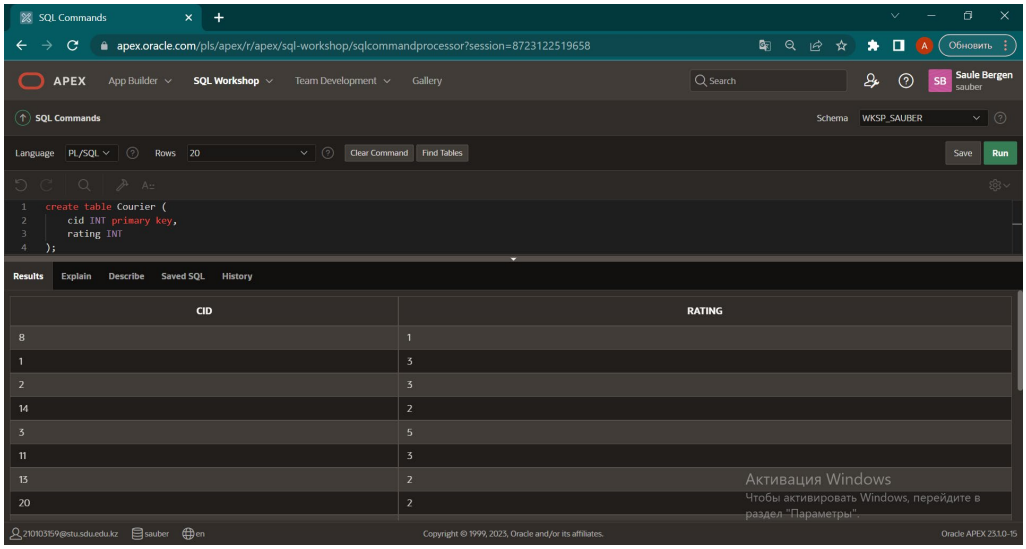
2) Create table Seller



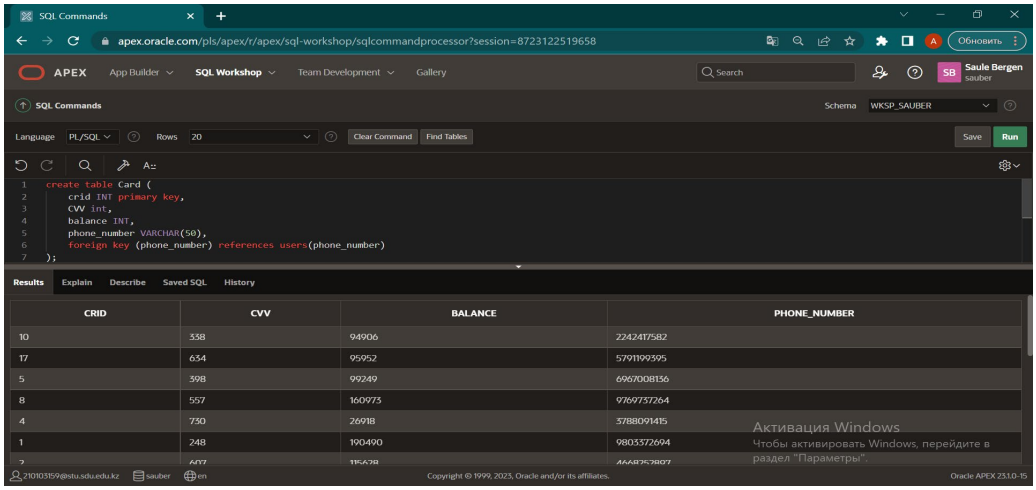
3) Create table Product



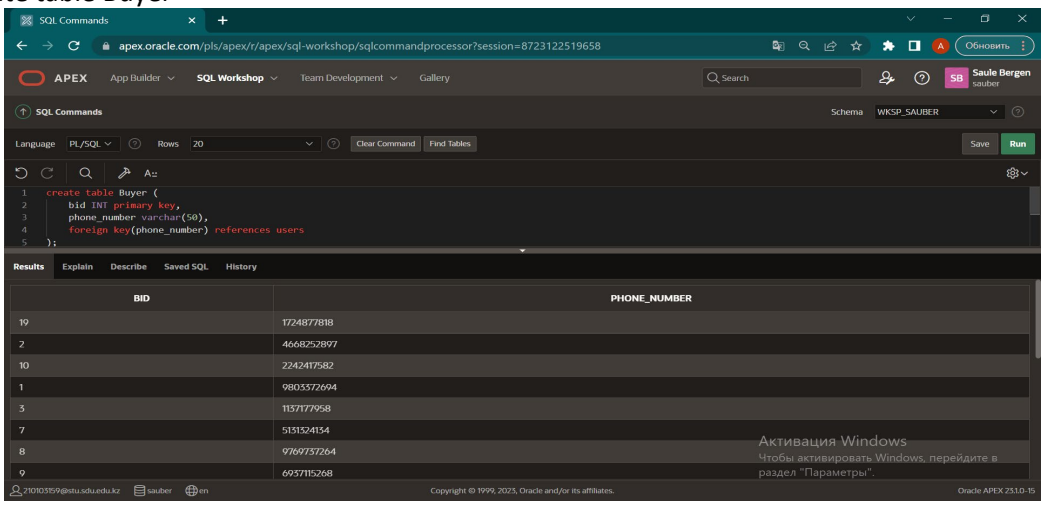
4) Create table Courier



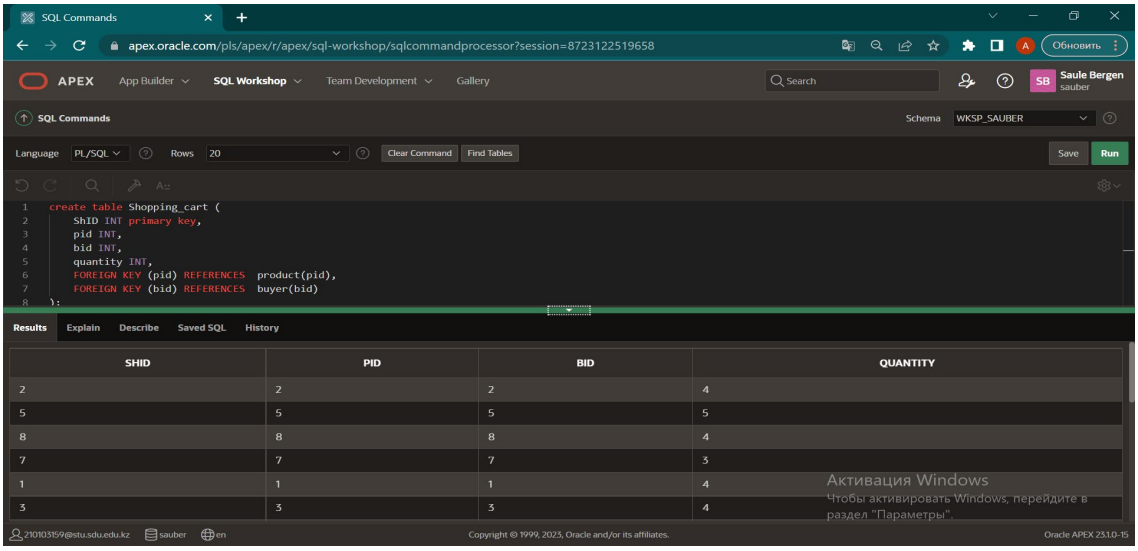
5) Create table Card



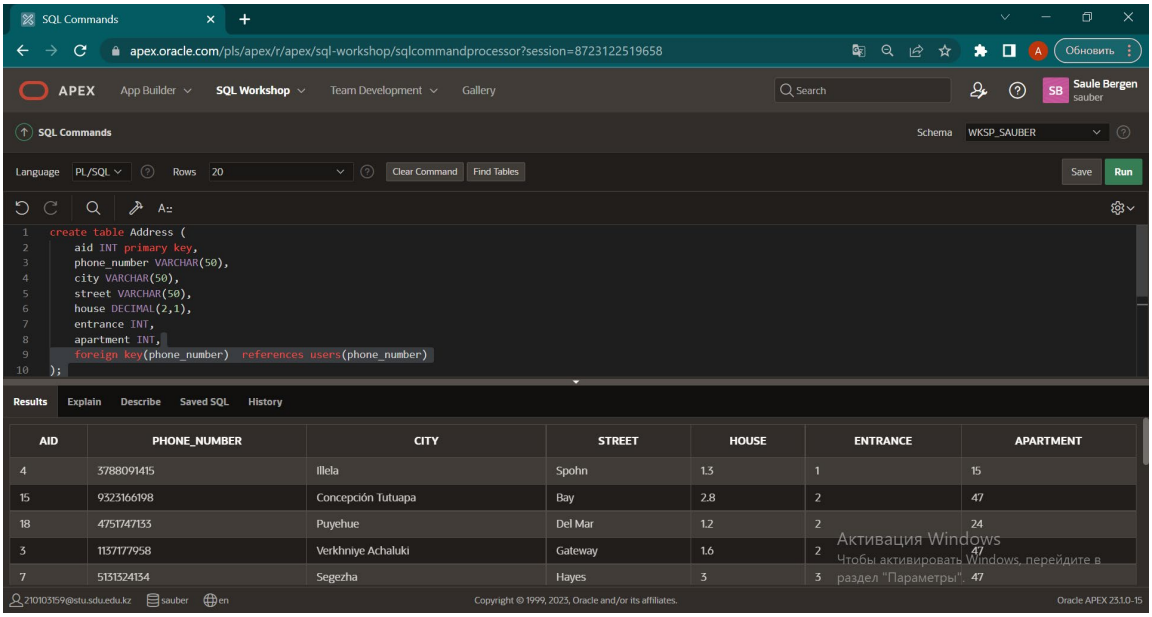
6) Create table Buyer



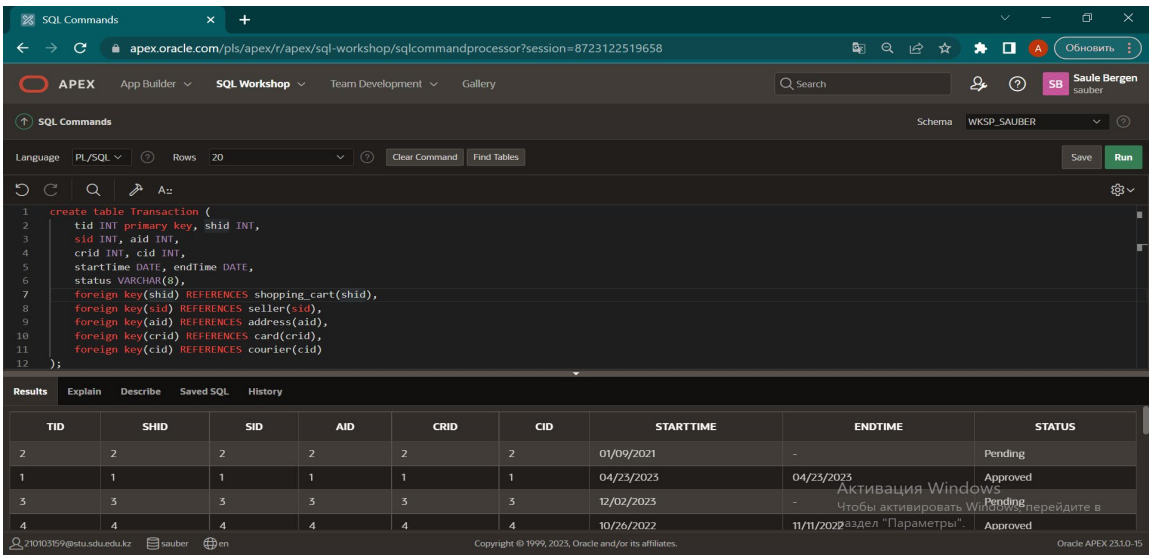
7) Create table Shopping_cart



8) Create table Address



9) Create table Transaction



EXCEPTIONS

FIRST

1.1

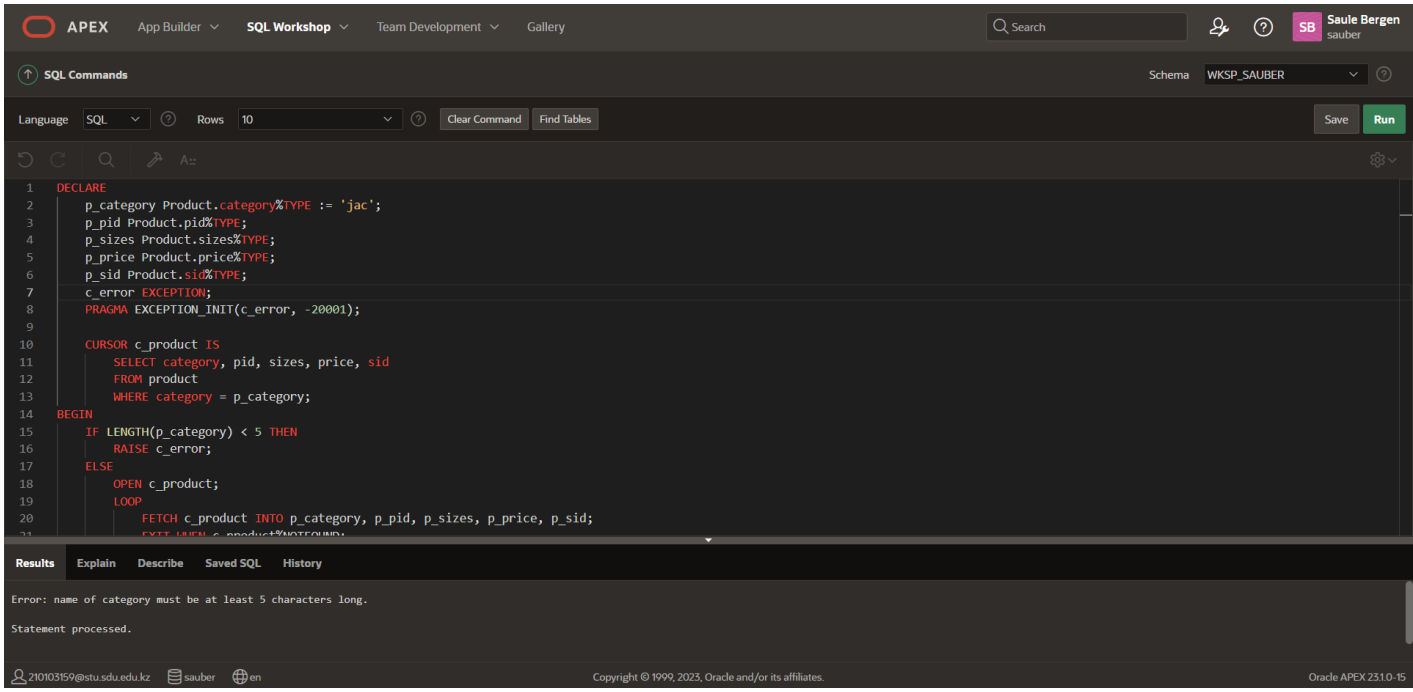
```
CREATE OR REPLACE PROCEDURE insert_product (
  p_pid IN Product.pid%TYPE,
  p_sid IN Product.sid%TYPE,
  p_price IN Product.price%TYPE,
  p_category IN Product.category%TYPE,
  p_sizes IN Product.sizes%TYPE
) As
  category_error EXCEPTION;
BEGIN
  IF LENGTH(p_category) < 5 THEN
    RAISE category_error;
  ELSE
    INSERT INTO Product (pid, sid, price, category, sizes) VALUES (p_pid, p_sid, p_price, p_category, p_sizes);
  END IF;
EXCEPTION
  WHEN category_error THEN
    DBMS_OUTPUT.PUT_LINE('Error: name of category must be at least 5 characters long.');
END;
```

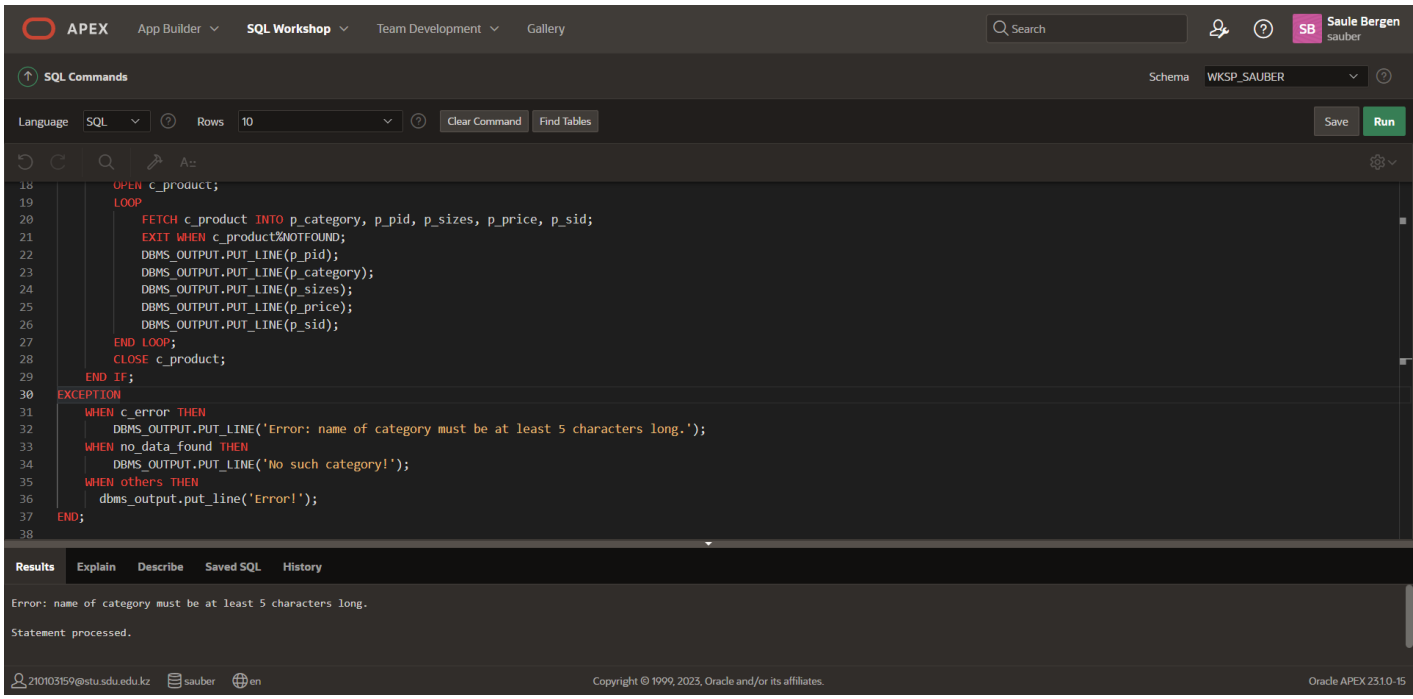
1.2

```
DECLARE
   p_category Product.category%TYPE := 'Jacket';
   p_pid Product.pid%TYPE;
   p_sizes Product.sizes%TYPE;
   p_price Product.price%TYPE;
   p_sid Product.sid%TYPE;
   c_error EXCEPTION;
   PRAGMA EXCEPTION_INIT(c_error, -20001);

   CURSOR c_product IS
```

```
        SELECT category, pid, sizes, price, sid
        FROM product
        WHERE category = p_category;
BEGIN
    IF LENGTH(p_category) < 5 THEN
        RAISE c_error;
    ELSE
        OPEN c_product;
        LOOP
            FETCH c_product INTO p_category, p_pid, p_sizes, p_price, p_sid;
            EXIT WHEN c_product%NOTFOUND;
            DBMS_OUTPUT.PUT_LINE(p_pid);
            DBMS_OUTPUT.PUT_LINE(p_category);
            DBMS_OUTPUT.PUT_LINE(p_sizes);
            DBMS_OUTPUT.PUT_LINE(p_price);
            DBMS_OUTPUT.PUT_LINE(p_sid);
        END LOOP;
        CLOSE c_product;
    END IF;
EXCEPTION
    WHEN c_error THEN
        DBMS_OUTPUT.PUT_LINE('Error: name of category must be at least 5 characters long.');
    WHEN no_data_found THEN
        DBMS_OUTPUT.PUT_LINE('No such category!');
    WHEN others THEN
      dbms_output.put_line('Error!');
END;
```

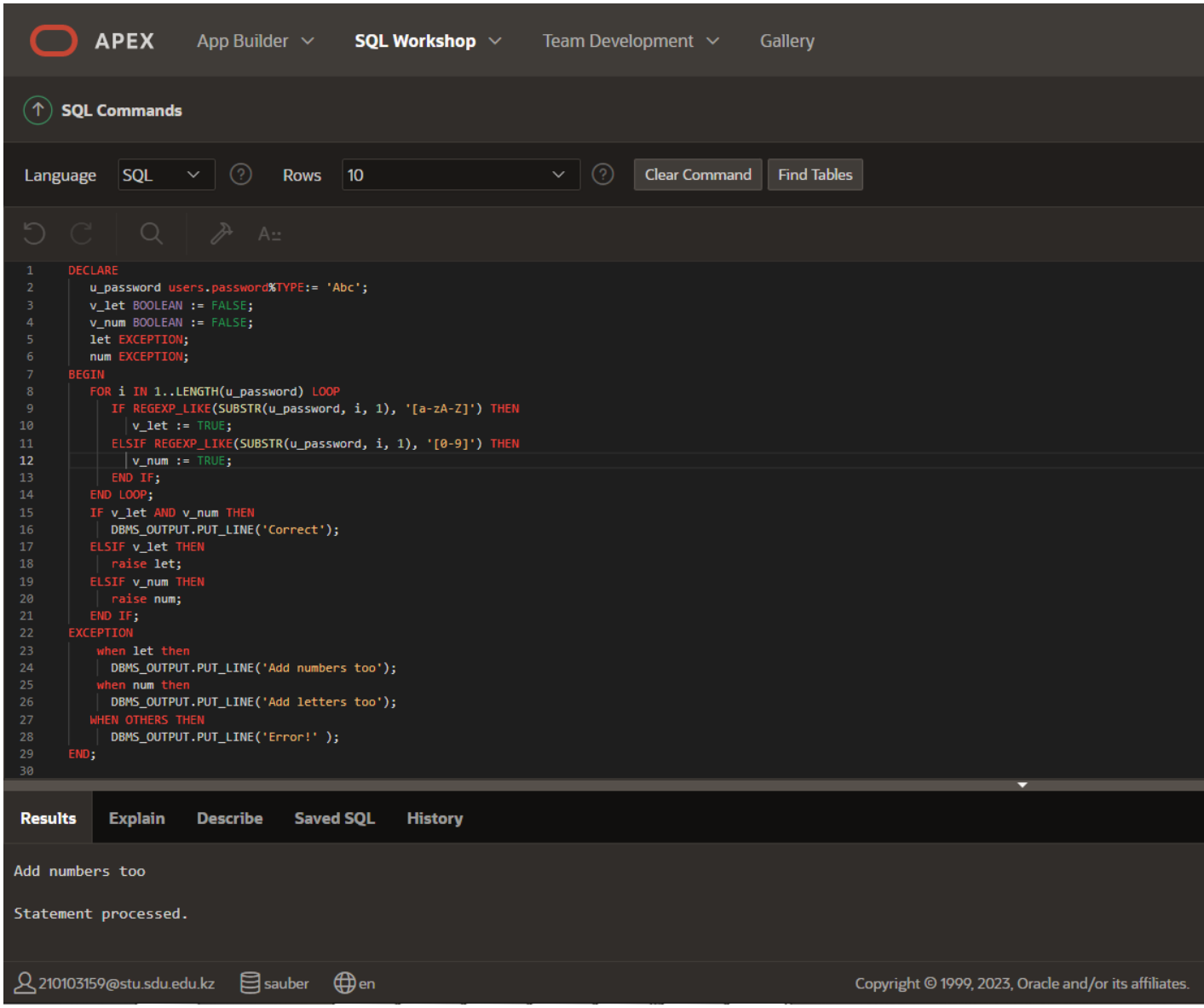




SECOND

```
DECLARE
  u_password users.password%TYPE:= 'Abc';
  v_let BOOLEAN := FALSE;
  v_num BOOLEAN := FALSE;
  let EXCEPTION;
  num EXCEPTION;
BEGIN
  FOR i IN 1..LENGTH(u_password) LOOP
```

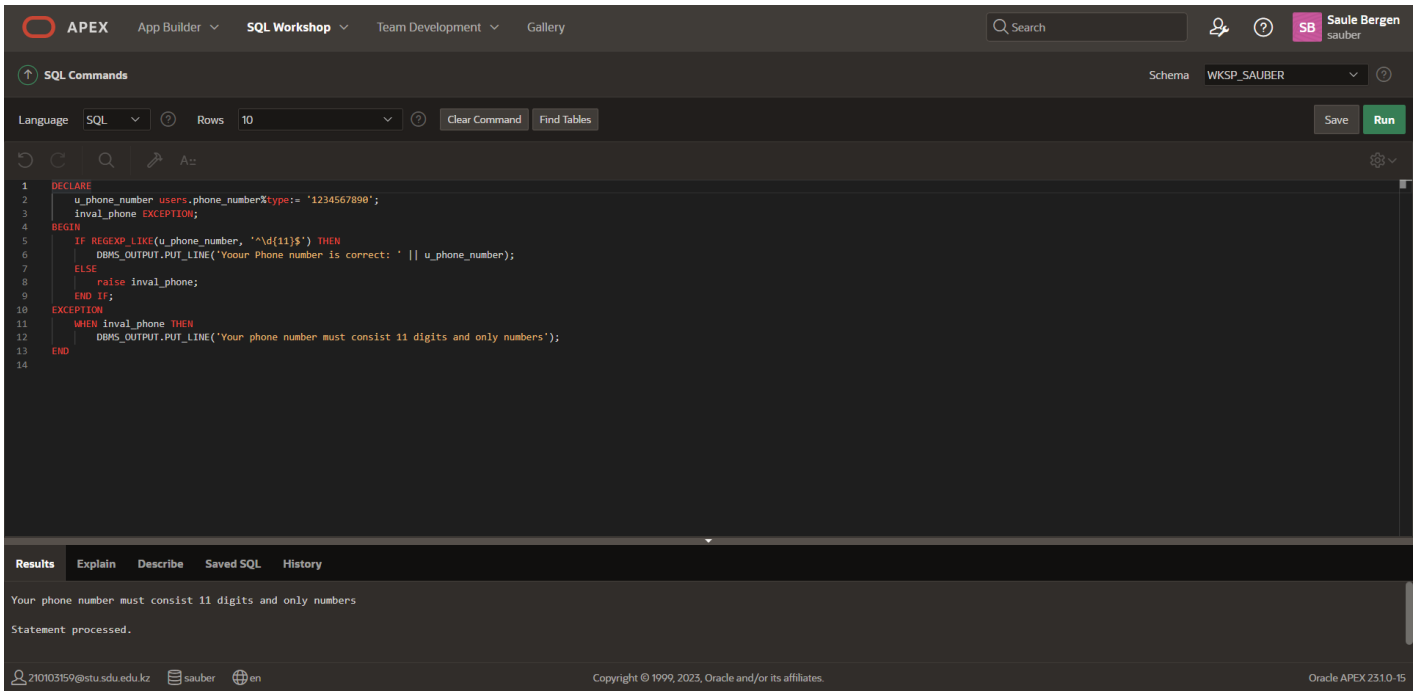
```
      IF REGEXP_LIKE(SUBSTR(u_password, i, 1), '[a-zA-Z]') THEN
        v_let := TRUE;
      ELSIF REGEXP_LIKE(SUBSTR(u_password, i, 1), '[0-9]') THEN
        v_num := TRUE;
      END IF;
    END LOOP;
    IF v_let AND v_num THEN
      DBMS_OUTPUT.PUT_LINE('Correct');
    ELSIF v_let THEN
      raise let;
    ELSIF v_num THEN
      raise num;
    END IF;
  EXCEPTION
     when let then
      DBMS_OUTPUT.PUT_LINE('Add numbers too');
     when num then
      DBMS_OUTPUT.PUT_LINE('Add letters too');
    WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error!' );
  END;
```



THIRD

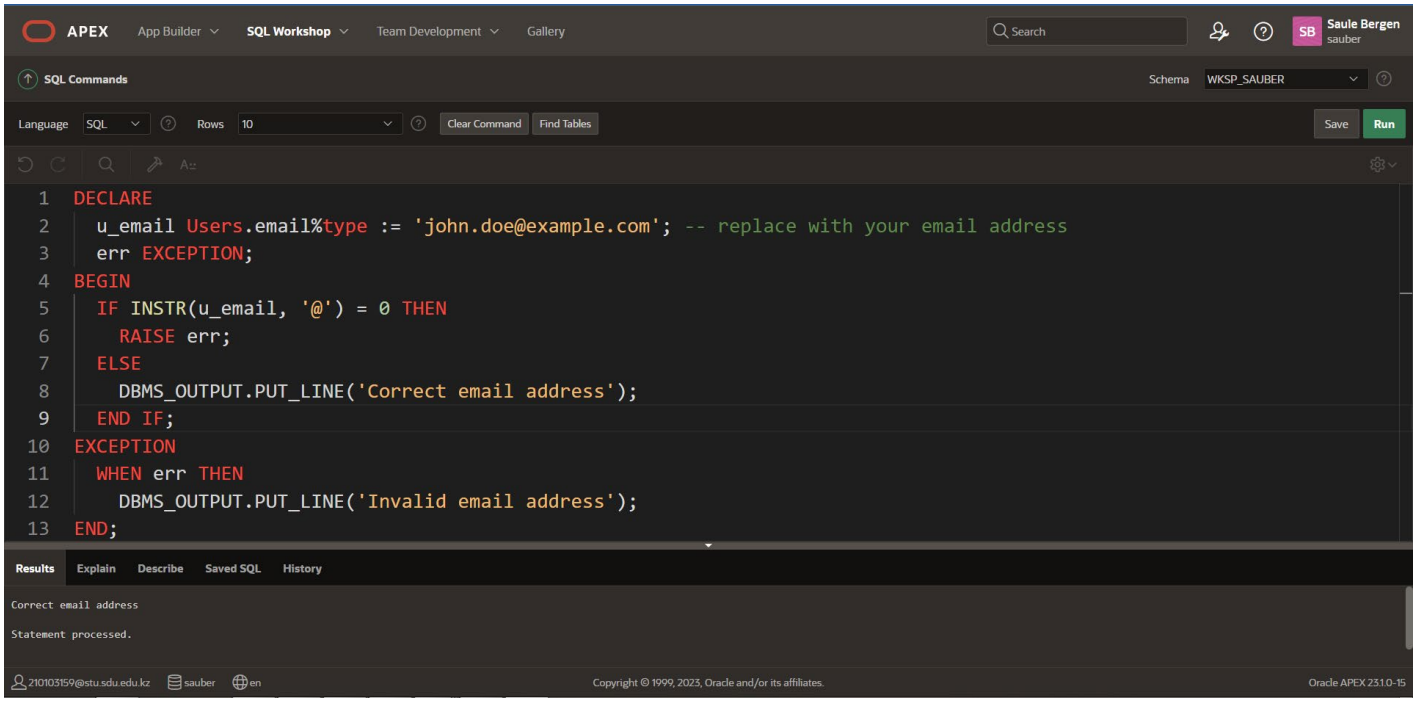
```
DECLARE
   u_phone_number users.phone_number%type:= '1234567890';
   inval_phone EXCEPTION;
BEGIN
   IF REGEXP_LIKE(u_phone_number, '^\d{11}$') THEN
      DBMS_OUTPUT.PUT_LINE('Yoour Phone number is correct: ' || u_phone_number);
   ELSE
      raise inval_phone;
   END IF;
EXCEPTION
   WHEN inval_phone THEN
```

```
    DBMS_OUTPUT.PUT_LINE('Your phone number must consist 11 digits and only numbers');
END;
```



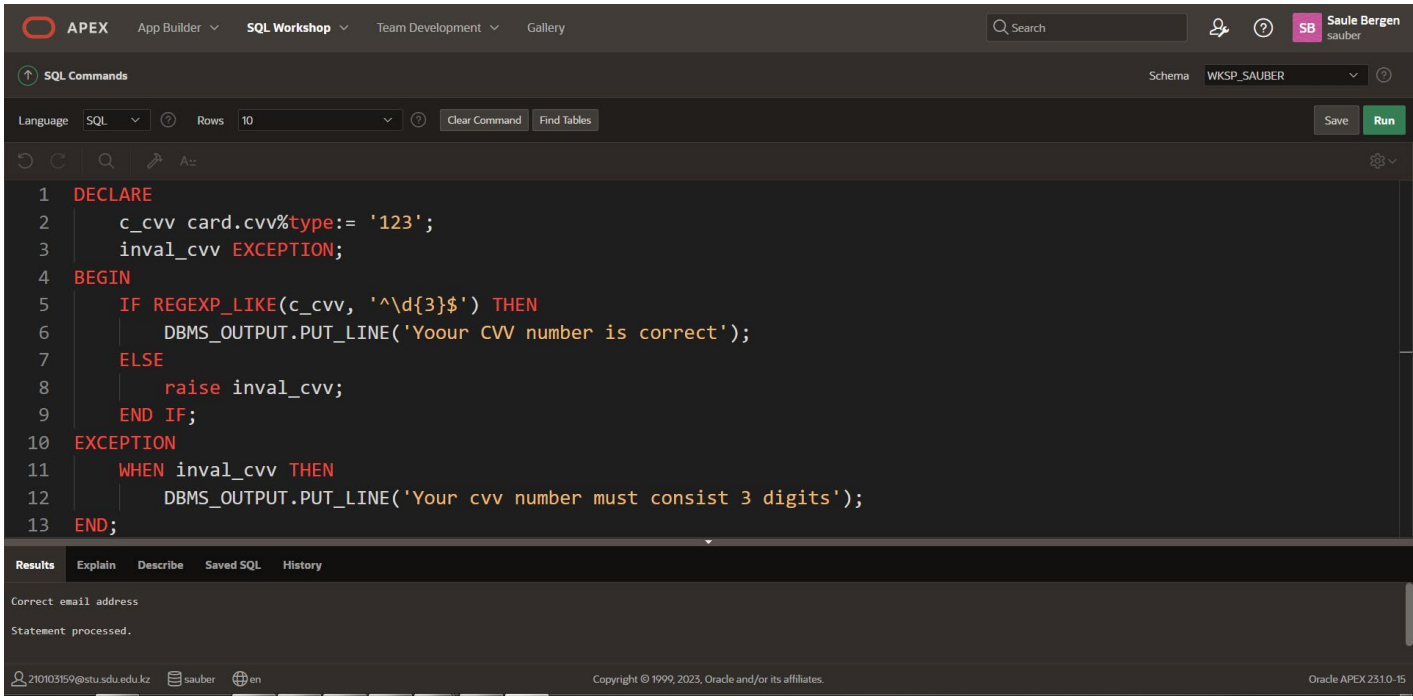
FOURTH

```
DECLARE
  u_email Users.email%type := 'john.doe@example.com'; -- replace with your email address
  err EXCEPTION;
BEGIN
  IF INSTR(u_email, '@') = 0 THEN
    RAISE err;
  ELSE
    DBMS_OUTPUT.PUT_LINE('Correct email address');
  END IF;
EXCEPTION
  WHEN err THEN
    DBMS_OUTPUT.PUT_LINE('Invalid email address');
END;
```



FIFTH

```
DECLARE
   c_cvv card.cvv%type:= '123';
   inval_cvv EXCEPTION;
BEGIN
   IF REGEXP_LIKE(c_cvv, '^\d{3}$') THEN
      DBMS_OUTPUT.PUT_LINE('Yoour CVV number is correct');
   ELSE
      raise inval_cvv;
   END IF;
EXCEPTION
   WHEN inval_cvv THEN
      DBMS_OUTPUT.PUT_LINE('Your cvv number must consist 3 digits');
END;
```



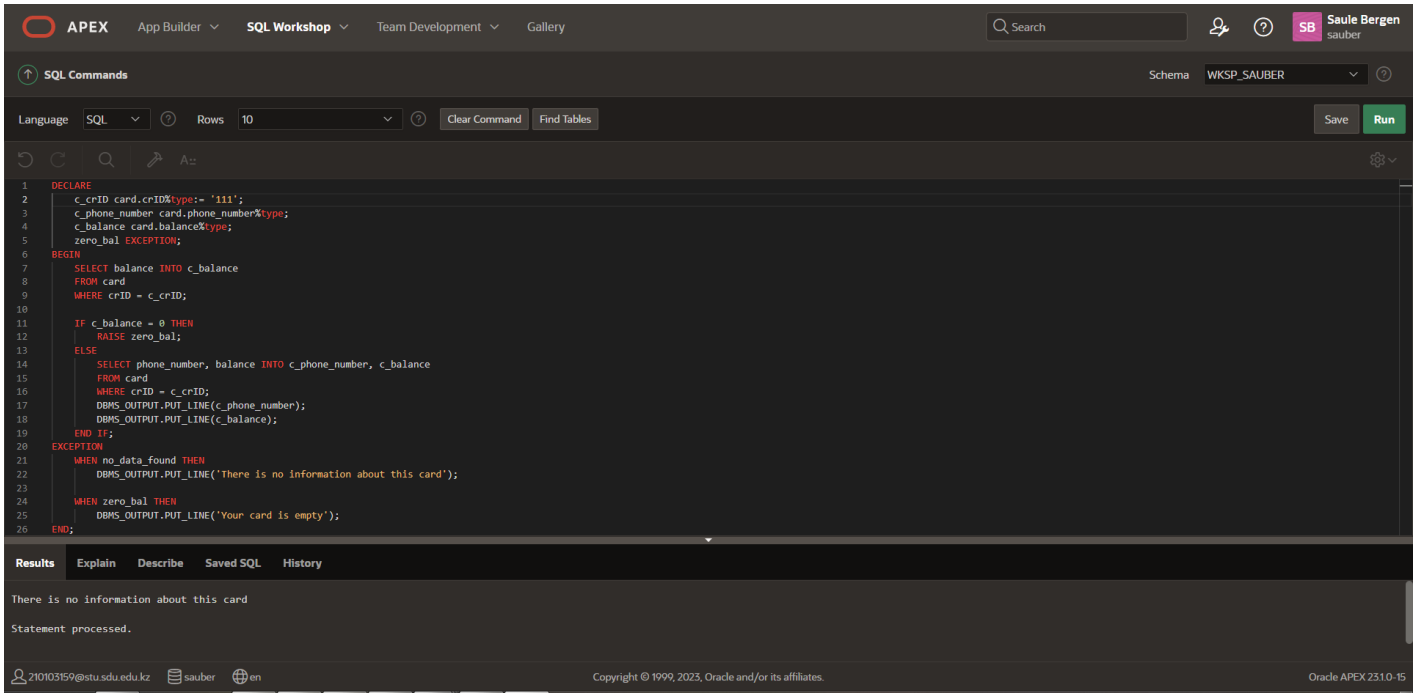
SIXTH

```
DECLARE
   c_crID card.crID%type:= '1';
   c_phone_number card.phone_number%type;
   c_balance card.balance%type;
   zero_bal EXCEPTION;
BEGIN
   SELECT balance INTO c_balance
   FROM card
   WHERE crID = c_crID;

   IF c_balance = 0 THEN
      RAISE zero_bal;
   ELSE
      SELECT phone_number, balance INTO c_phone_number, c_balance
      FROM card
      WHERE crID = c_crID;
```

```
        DBMS_OUTPUT.PUT_LINE(c_phone_number);
        DBMS_OUTPUT.PUT_LINE(c_balance);
    END IF;
EXCEPTION
    WHEN no_data_found THEN
        DBMS_OUTPUT.PUT_LINE('There is no information about this card');

    WHEN zero_bal THEN
        DBMS_OUTPUT.PUT_LINE('Your card is empty');
END;
```



SEVENTH

```
CREATE OR REPLACE TRIGGER duplicate_check
BEFORE INSERT OR UPDATE ON address
FOR EACH ROW
DECLARE
  a_AID Address.AID%TYPE;
  duplicate_count INTEGER;
```

```
  dupl_error EXCEPTION;
BEGIN
  SELECT COUNT(*) INTO duplicate_count
  FROM address
  WHERE AID = a_AID;

  IF duplicate_count > 0 THEN
    RAISE dupl_error;
  END IF;
EXCEPTION
  when dupl_error THEN
    DBMS_OUTPUT.PUT_LINE('The address of such AID already exists');
END;
```

Create a trigger before insert on any entity which will show the current number of rows in the table

```
CREATE OR REPLACE FUNCTION count_row_f(table_name IN VARCHAR2)
RETURN NUMBER
IS
  count_row NUMBER;
BEGIN
  EXECUTE IMMEDIATE 'SELECT COUNT(*) FROM ' || table_name INTO count_row;
  RETURN count_row;
END count_row_f;

CREATE OR REPLACE TRIGGER count_row_tr
BEFORE INSERT ON users
```

```
FOR EACH ROW

BEGIN

DBMS_OUTPUT.PUT_LINE('The current number of rows in the table (not including the row you are adding): ' ||
count_row_f('users'));

END;


CREATE OR REPLACE TRIGGER count_row_tr2

BEFORE INSERT ON address

FOR EACH ROW

BEGIN

DBMS_OUTPUT.PUT_LINE('The current number of rows in the table (not including the row you are adding): ' ||
count_row_f('address'));

END;


CREATE OR REPLACE TRIGGER count_row_tr3

BEFORE INSERT ON card

FOR EACH ROW

BEGIN

DBMS_OUTPUT.PUT_LINE('The current number of rows in the table (not including the row you are adding): ' ||
count_row_f('card'));

END;


CREATE OR REPLACE TRIGGER count_row_tr4

BEFORE INSERT ON seller

FOR EACH ROW

BEGIN

DBMS_OUTPUT.PUT_LINE('The current number of rows in the table (not including the row you are adding): ' ||
count_row_f('seller'));

END;


CREATE OR REPLACE TRIGGER count_row_tr5

BEFORE INSERT ON buyer

FOR EACH ROW

BEGIN
```

```
DBMS_OUTPUT.PUT_LINE('The current number of rows in the table (not including the row you are adding): ' ||
count_row_f('buyer'));

END;


CREATE OR REPLACE TRIGGER count_row_tr6

BEFORE INSERT ON shopping_cart

FOR EACH ROW

BEGIN

DBMS_OUTPUT.PUT_LINE('The current number of rows in the table (not including the row you are adding): ' ||
count_row_f('shopping_cart'));

END;


CREATE OR REPLACE TRIGGER count_row_tr7

BEFORE INSERT ON transaction

FOR EACH ROW

BEGIN

DBMS_OUTPUT.PUT_LINE('The current number of rows in the table (not including the row you are adding): ' ||
count_row_f('transaction'));

END;


CREATE OR REPLACE TRIGGER count_row_tr8

BEFORE INSERT ON courier

FOR EACH ROW

BEGIN

DBMS_OUTPUT.PUT_LINE('The current number of rows in the table (not including the row you are adding): ' ||
count_row_f('courier'));

END;


CREATE OR REPLACE TRIGGER count_row_tr9

BEFORE INSERT ON product

FOR EACH ROW

BEGIN

DBMS_OUTPUT.PUT_LINE('The current number of rows in the table (not including the row you are adding): ' ||
count_row_f('product'));

END;
```

```
1  CREATE OR REPLACE FUNCTION count_row_f(table_name IN VARCHAR2)
2  RETURN NUMBER
3  IS
4     count_row NUMBER;
5  BEGIN
17 CREATE OR REPLACE TRIGGER count_row_tr2
18 BEFORE INSERT ON address
19 FOR EACH ROW
84 Insert into users(PHONE_NUMBER,NAME,    SURNAME,    EMAIL,  PASSWORD,   REGISTRATIONDATE,   STATUS)
85 values ('9803372670',  'Mika', 'Joly'   ,'meru_j@mail.com'  ,'mika_j12345','1/30/2021', 'buyer');
86
87 SELECT count(*) FROM users;
```

Results | Explain | Describe | Saved SQL | History

```
The current number of rows in the table (not including the row you are adding): 24

1 row(s) inserted.
```

Results | Explain | Describe | Saved SQL | History

```
Trigger created.

0.04 seconds
```

```
87 SELECT count(*) FROM users;
```

Results | Explain | Describe | Saved SQL | Histor

```
25
```

Create trigger status upgrade

Trigger change status where we update end or start time in table transaction.

Approved: if start time is before 14 hours from end time Declined: if start time is after 14 hours from end time Pending: if start and end time are within 14 hours

Language PL/SQL Rows 20 Clear Command Find Tables

```
CREATE OR REPLACE TRIGGER status_upgrade
BEFORE UPDATE ON transaction
FOR EACH ROW
DECLARE
    date_dif NUMBER;
BEGIN
    date_dif := 14;
    IF :new.endtime - :new.startTime >= date_dif THEN
        :new.status := 'Declined';
    ELSIF :new.endtime - :new.startTime < date_dif THEN
        :new.status := 'Approved';
    ELSE
        :new.status := 'Pending';
    END IF;
END;
```

Results Explain Describe Saved SQL History

```
Trigger created.
```

0.12 seconds

210105159@stu.sdu.edu.kz sauber en

APEX App Builder SQL Workshop Team Development Gallery Search SB Saule Bergen sauber

SQL Commands Schema WKSP_SAUBER

Language PL/SQL Rows 20 Clear Command Find Tables Save Run

```
select * from transaction where tid = 1;
```

Results Explain Describe Saved SQL History

TID	SHID	SID	AID	CRID	CID	STARTTIME	ENDTIME	STATUS
1	1	1	1	1	1	04/23/2023	04/23/2023	Approved

1 rows returned in 0.00 seconds Download

APEX App Builder SQL Workshop Team Development Gallery Search SB Saule Bergen sauber

SQL Commands Schema WKSP_SAUBER

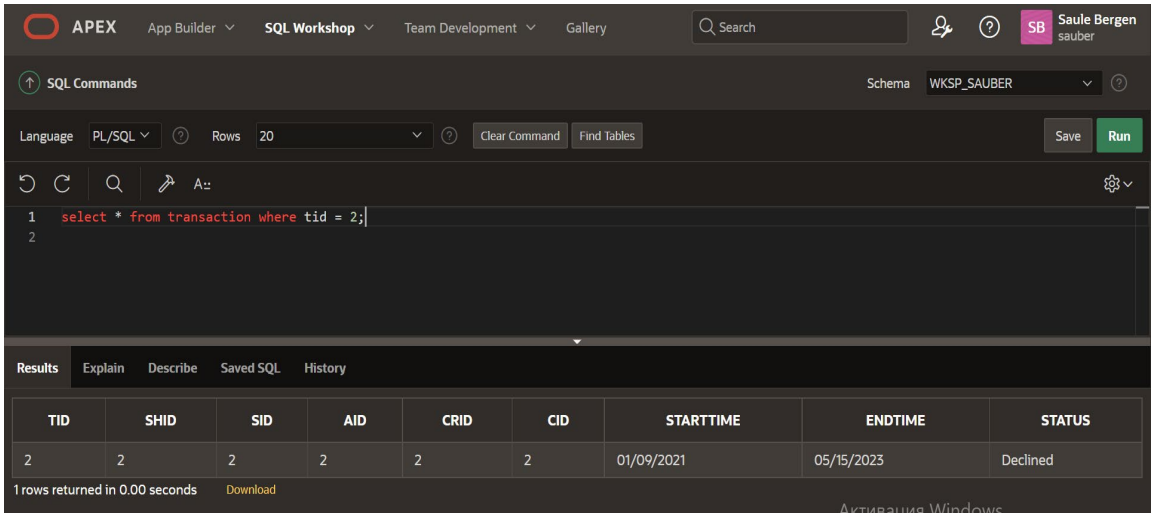
Language PL/SQL Rows 20 Clear Command Find Tables Save Run

```
update transaction set endTime = '05/15/2023' where tid = 2;
```

Results Explain Describe Saved SQL History

```
1 row(s) updated.
```

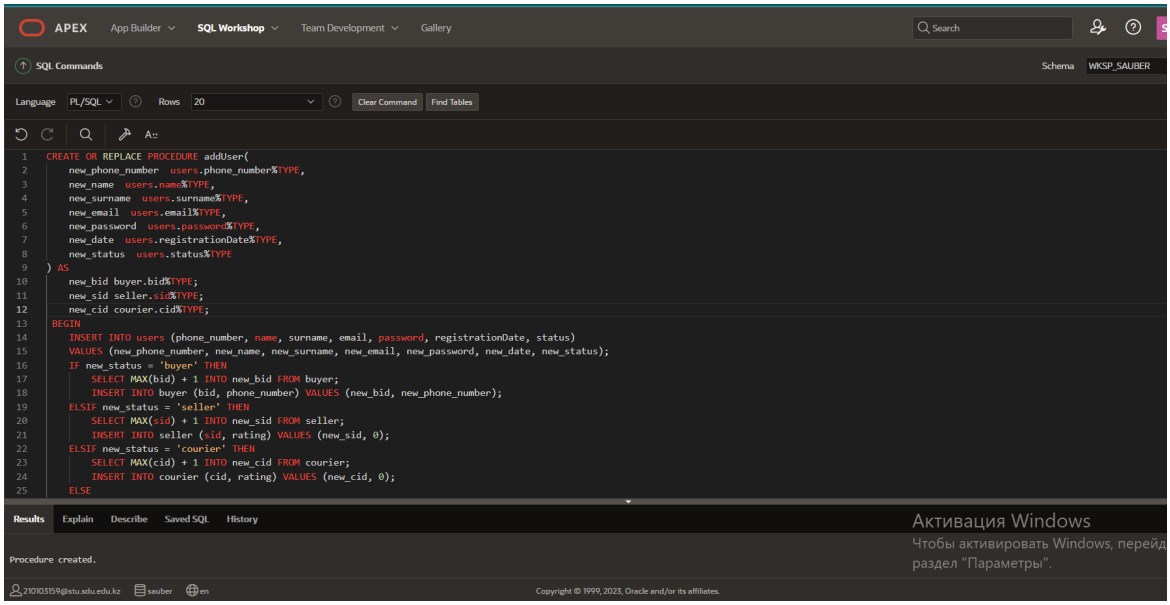
0.02 seconds

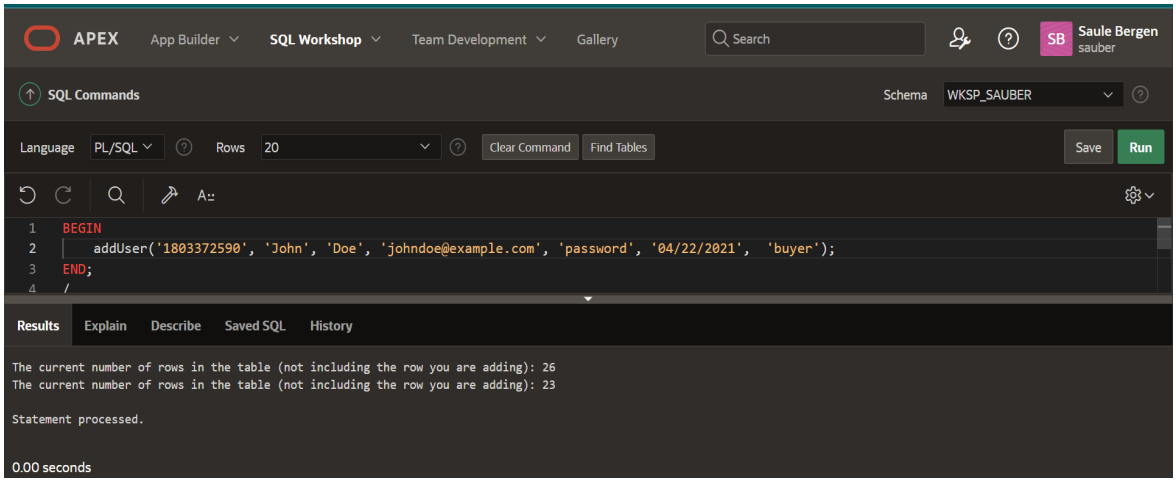


Create procedure add new user

We add new user to our db. And table add this user by their status.

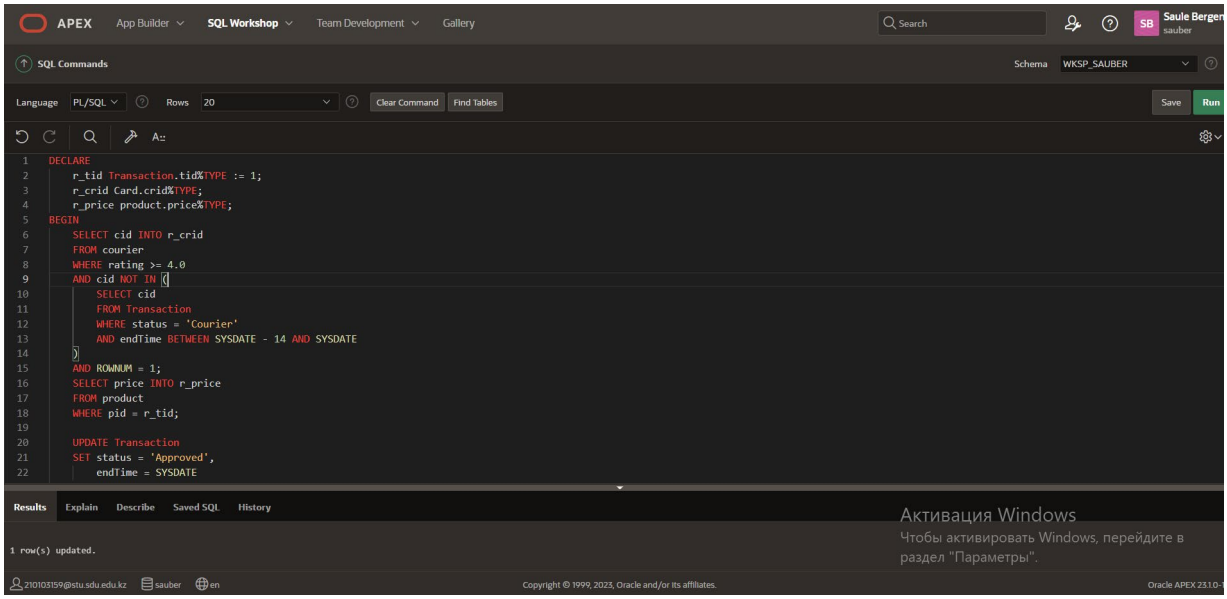
If add courier status, add in table courier too.



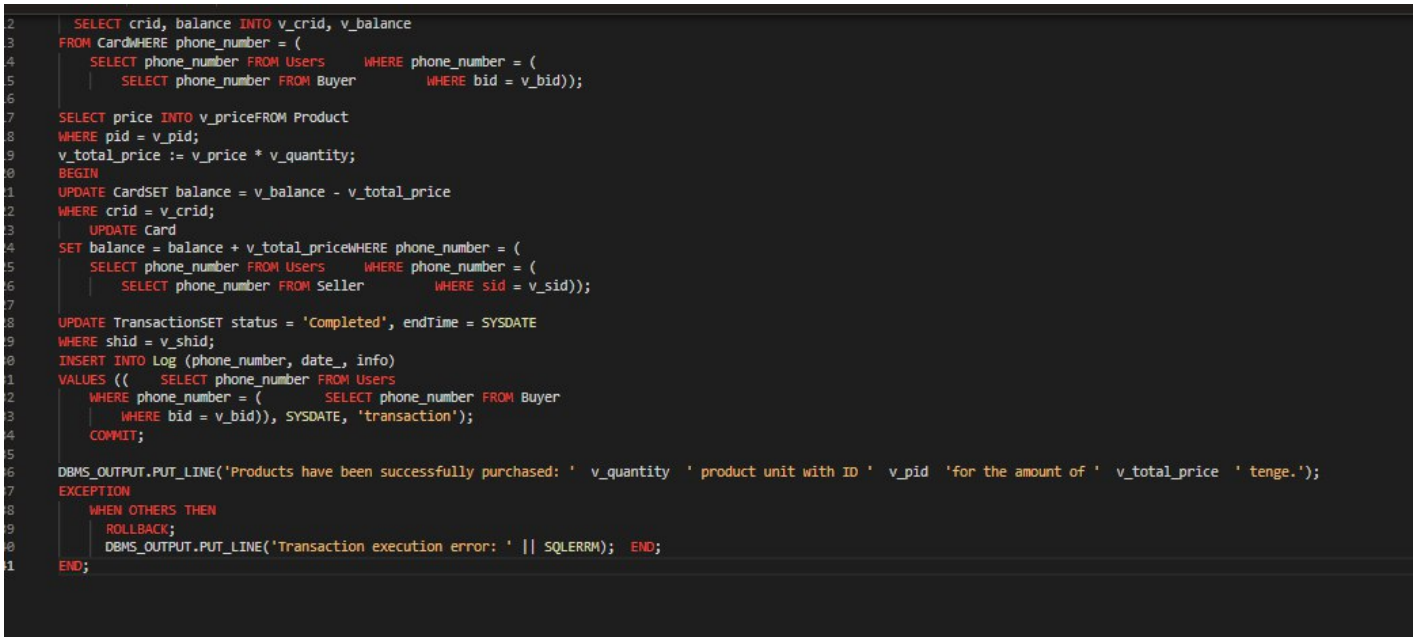


TRANSACTION

In particular, it selects a courier with a rating of 4 or higher that has not delivered any transactions in the last 14 days, selects the price of an item with a specific ID, updates the status and end time of a transaction with a specific ID to "Approved" and the current date, and updates card balance with a specific identifier associated with the selected courier. If the courier is unavailable or an error occurs, then throws an application error or rolls back the changes and throws an error again.



Transaction 2.



```
SELECT crid, balance INTO v_crid, v_balance
FROM CardWHERE phone_number = (
    SELECT phone_number FROM Users     WHERE phone_number = (
        SELECT phone_number FROM Buyer         WHERE bid = v_bid));

SELECT price INTO v_priceFROM Product
WHERE pid = v_pid;
v_total_price := v_price * v_quantity;
BEGIN
UPDATE CardSET balance = v_balance - v_total_price
WHERE crid = v_crid;
    UPDATE Card
SET balance = balance + v_total_priceWHERE phone_number = (
    SELECT phone_number FROM Users     WHERE phone_number = (
        SELECT phone_number FROM Seller         WHERE sid = v_sid));

UPDATE TransactionSET status = 'Completed', endTime = SYSDATE
WHERE shid = v_shid;
INSERT INTO Log (phone_number, date_, info)
VALUES ((    SELECT phone_number FROM Users
    WHERE phone_number = (        SELECT phone_number FROM Buyer
        WHERE bid = v_bid)), SYSDATE, 'transaction');
    COMMIT;

DBMS_OUTPUT.PUT_LINE('Products have been successfully purchased: '  v_quantity  ' product unit with ID '  v_pid  'for the amount of '  v_total_price  ' tenge.');
EXCEPTION
    WHEN OTHERS THEN
      ROLLBACK;
      DBMS_OUTPUT.PUT_LINE('Transaction execution error: ' || SQLERRM);  END;
END;
```

PROCEDURES

1) The first procedure returns us the average price of a product in a particular category. For example, we have a category of skirts, t-shirts, and so on. This procedure returns the average price of each category.

This is the code:

```
--return average price of product category
create or replace procedure my_procedure is
cursor my_cursor is
select category, AVG(price) as avg_price
from product
group by category;
v_category product.category%TYPE;
v_avg_price product.price%TYPE;
begin
open my_cursor;
loop
fetch my_cursor into v_category, v_avg_price;
exit when my_cursor%NOTFOUND;
dbms_output.put_line(v_category || ': ' || v_avg_price);
end loop;
close my_cursor;
end;

begin
my_procedure;
end;
```

And this is the result of our function:

```
198    v_avg_price prod
```

Results | Explain | Desc

```
Shorts: 19818
Shirt: 25050
Dress: 6407
Jacket: 19542
Skirt: 20799

Statement processed.

0.01 seconds
```

2) This procedure changes the price of a product and throws an error if the number of rows changed is 0 (through %ROWCOUNT).

Table look like this before update:

This is the code:

```
--notifies you of a successful procedure
create or replace procedure update_price(
    pr_id in number,
    new_price in number
)
is
begin
update product set price = new_price
where pid = pr_id;
if SQL%ROWCOUNT = 0 then
RAISE_APPLICATION_ERROR(-20002, 'Product not found');
else
dbms_output.put_line('Price updated successfully');
end if;
exception
when others then
dbms_output.put_line('Error: ' || SQLERRM);
end;

begin
update_price(10, 26990);
end;
```

PID	SIZES	CATEGORY	PRICE
1	XS	Jacket	14632
2	M	Skirt	24237
6	XS	Shorts	13557
10	2XL	Shirt	26990

And this is result of our function (You can see that price of product with PID 10 is 26990):

3) The following procedure returns all information about the product to us if we enter its ID:

PID	SIZES	CATEGORY	PRICE
1	XS	Jacket	14632
2	M	Skirt	24237
6	XS	Shorts	13557
10	2XL	Shirt	25990

```
--if we enter product id that procedure return us other information about product
create or replace procedure get_product(v_pid product.pid%TYPE)
as
v_size product.sizes%TYPE;
v_category product.category%TYPE;
v_price product.price%TYPE;
begin
select sizes, category, price into v_size, v_category, v_price
from product
where pid = v_pid;
dbms_output.put_line('Product information');
dbms_output.put_line('size: ' || v_size);
dbms_output.put_line('category: ' || v_category);
dbms_output.put_line('price: ' || v_price);
exception
when NO_DATA_FOUND then
dbms_output.put_line('Record not found');
end;

begin
get_product(546);
end;
```

This is the result of procedure:

```
Results   Explain   Desc

Product information
size: 2XL
category: Shirt
price: 25990

Statement processed.

0.00 seconds
```

Functions

1) This function declare how many users our shop had(Count(*)):

```
--declare how many users our shop have
create or replace function count_users
return number
is
v_users_count number;
begin
select count(*) into v_users_count
from users;
return v_users_count;
end;

declare
res number;
begin
res:=count_users;
dbms_output.put_line('Count of users is ' || res);
end;
```

This is the result of function:

Results | Explain | Describe

```
Count of users is 25

Statement processed.

0.02 seconds
```

2) This function returns average price of current category. For example, we have skirts category. This function returns us average price of skirts:

```oracle_plsql
--return us average price of current category
create or replace function avg_price(category varchar2) return number
is
total number := 0;
product_count number := 0;
begin
for prod in (select price from product where category = category)
loop total := total + prod.price;
product_count := product_count + 1;
end loop;
if product_count = 0 then
return 0;
else
return total / product_count;
end if;
end;

declare
res number;
begin
res := avg_price('Skirt');
dbms_output.put_line('Average price of skirts is: ' || res);
end;
```

Result of this function is:

Results | Explain | Describe | Saved SQL

```
Average price of skirts is: 20708.25

Statement processed.

0.02 seconds
```