Aidana Yensetaikyzy

Dinara Zhaki

Ali Shakirzhanuly

Zhanarys Seidakhan

Yernar Tastan

Professor Name: Azamat Serek

INF 305 DBMS 2

24 April 2023

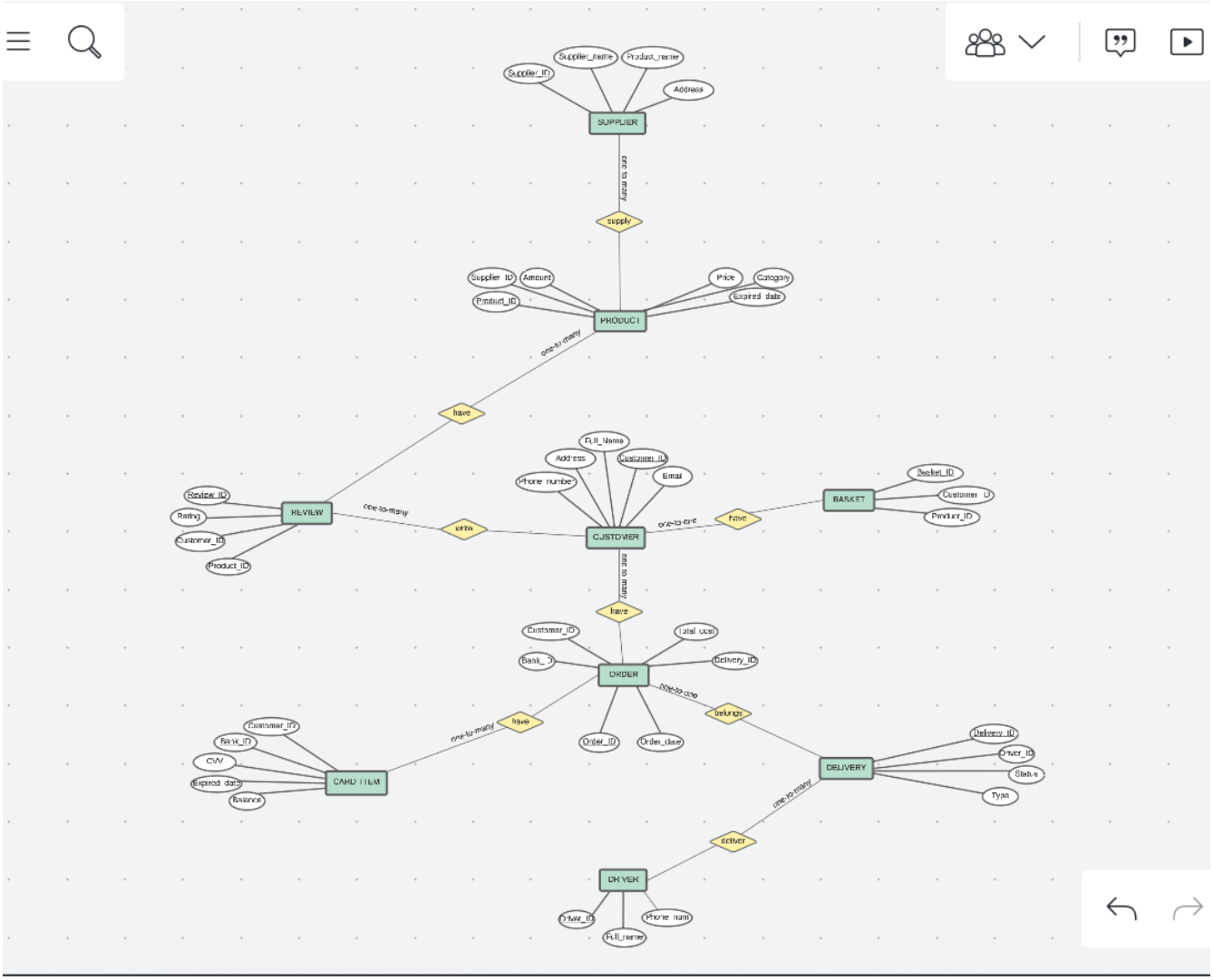
Online Shop MIDTERM PROJECT

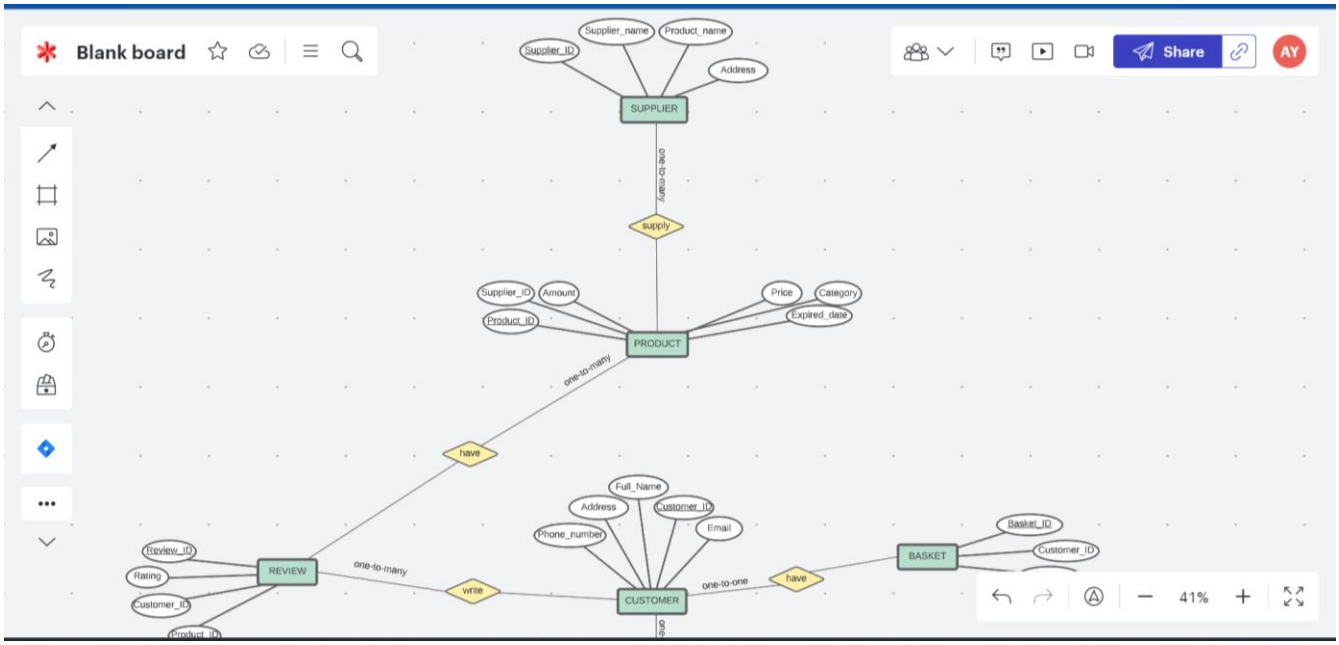
Introduction:

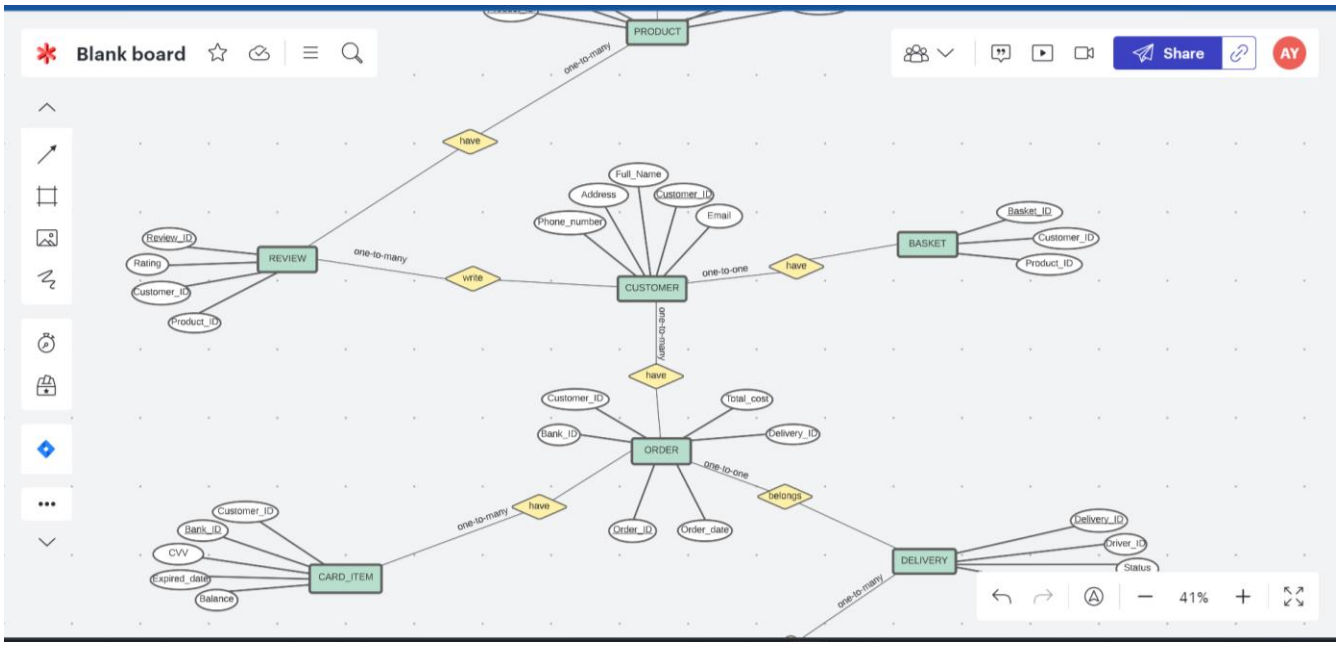
The purpose of this project is to create an online shop system that allows customers to order products from suppliers and get them delivered to their addresses. The system consists of nine tables: Customer, Supplier, Product, Reviews, Order, Delivery, Card_Item, Basket and Driver.

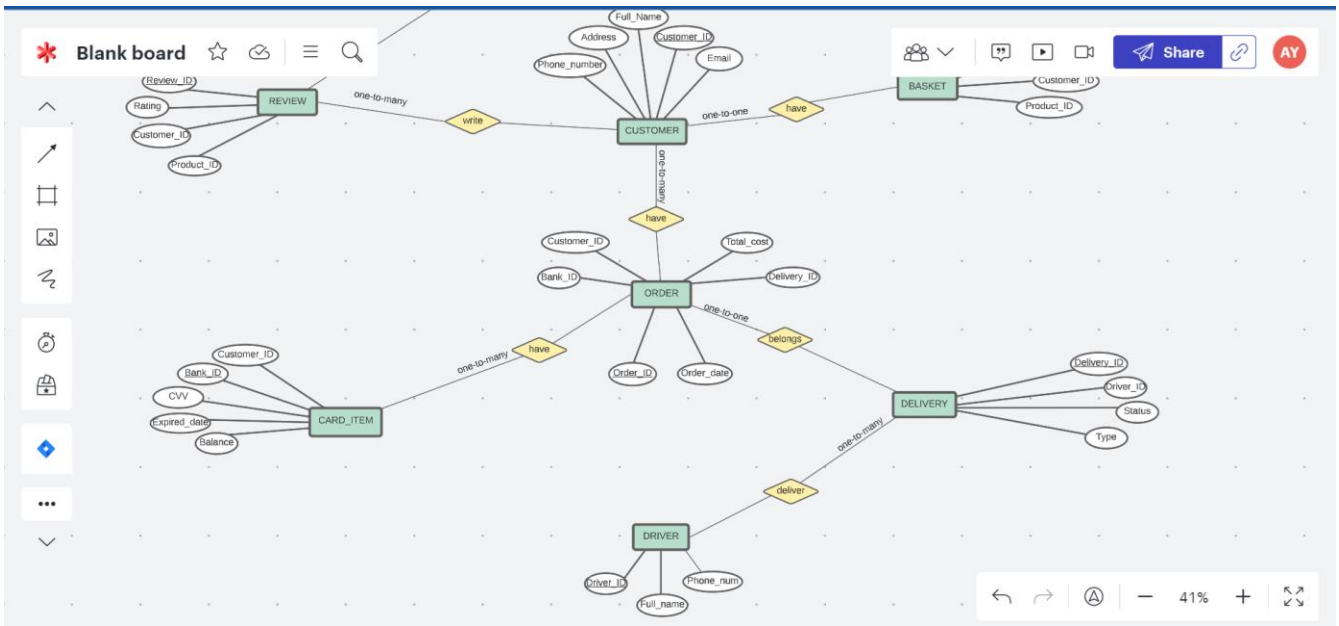
ER diagram:

https://lucid.app/lucidspark/57c066eb-1eed-458a-ad79-e13288c63f99/edit?viewport_loc=-786%2C11%2C3326%2C1562%2C0_0&invitationId=inv_8214c7a9-4dec-42da-957e-edae5d954e15


Supplier_name
Product_name
Supplier_ID
Address
SUPPLIER
one to many
supply
Supplier_ID
Amount
Product_ID
Price
Category
Expired date
PRODUCT
one-to-many
have
Full_Name
Address
Customer_ID
Phone number
Email
Review_ID
Rating
Customer_ID
Product_ID
REVIEW
one-to-many
write
CUSTOMER
one-to-one
have
BASKET
Basket_ID
Customer_ID
Product_ID
one to many
have
Customer_ID
Total cost
Bank_ID
Delivery_ID
ORDER
one-to-one
belongs
Order_ID
Order_date
Customer_ID
Bank_ID
CVV
Expired date
Balance
CARD ITEM
one-to-many
have
Delivery_ID
Driver_ID
Status
Type
DELIVERY
one-to-many
deliver
DRIVER
Driver_ID
Phone num
Full_name


Blank board
Share
AY
Supplier_name
Product_name
Supplier_ID
Address
SUPPLIER
one-to-many
supply
Supplier_ID
Amount
Product_ID
Price
Category
Expired_date
PRODUCT
one-to-many
have
Full_Name
Address
Customer_ID
Phone_number
Email
Review_ID
Rating
Customer_ID
Product_ID
REVIEW
one-to-many
write
CUSTOMER
one-to-one
have
BASKET
Basket_ID
Customer_ID
41%


Blank board
Share
AY
PRODUCT
one-to-many
have
Full_Name
Address
Customer_ID
Phone_number
Email
Review_ID
Rating
Customer_ID
Product_ID
REVIEW
one-to-many
write
CUSTOMER
one-to-one
have
BASKET
Basket_ID
Customer_ID
Product_ID
one-to-many
have
Customer_ID
Total_cost
Bank_ID
Delivery_ID
ORDER
one-to-one
belongs
Order_ID
Order_date
Customer_ID
Bank_ID
CVV
Expired_date
Balance
CARD_ITEM
one-to-many
have
DELIVERY
Delivery_ID
Driver_ID
Status
one-to-many
41%



The diagram shows the following tables:

- Customer: Stores customer information such as their ID, phone number, address, full name, and email.
- Supplier: Stores supplier information such as their ID, supplier name, product name, and address.
- Product: Stores product information such as the ID, supplier ID (foreign key), price, amount, expired date, and category.
- Reviews: Stores customer reviews on a product, including the review ID, rating, customer ID (foreign key), and product ID (foreign key).
- Order: Stores order information, including the order ID, customer ID (foreign key), bank ID (foreign key), total cost, delivery ID (foreign key), and order date.
- Delivery: Stores delivery information, including the delivery ID, driver ID (foreign key), delivery status, and delivery type.
- Card_item: Stores information about customer's bank card, including the bank ID (primary key), customer ID (foreign key), CVV, expiration date, and balance.

- Basket: Stores information about customer's basket, including the basket ID, customer ID (foreign key), and product ID (foreign key).
- Driver: Stores information about drivers, including their ID, full name, and phone number.

Explanation of why the structure follows normal forms:

Customer -> all attributes depend on cust_id (is a primary key). And all attributes can be found using cust__id. Also in this entity there are no transitive dependencies

Supplier -> all attributes are dependent on supp_id (is a primary key). And all attributes can be found using supp_id. Also in this entity there are no transitive dependencies

Product -> all attributes are dependent on prod_id (is a primary key). And so that attributes are not duplicated, we put prod_name in supplier.

Also, all attributes of this entity can be found using prod_id (is a primary key). Also, there are no transitive dependencies in this entity

Reviews -> all attributes depend on review_id (is a primary key). And all attributes can be found using review__id. Also, there are no transitive dependencies in this entity

Order -> all attributes depend on order_id (is a primary key). And all attributes can be found using order_id. Also in this entity there are no transitive dependencies

Delivery -> all attributes are dependent on deliver_id (is a primary key). And all attributes can be found using deliver_id. Also, there are no transitive dependencies in this entity

Card_item -> all attributes depend on bank_id (is a primary key). And all attributes can be found using bank_id. Also, there are no transitive dependencies in this entity

Basket -> all attributes depend on the basket_id (is a primary key). And all attributes can be found using basket_id. Also in this entity there are no transitive dependencies

Driver -> all attributes are dependent on driver_id (is a primary key). And all attributes can be found using driver_id. Also, there are no transitive dependencies in this entity

Explanation and coding part of each item:

Each member of the team had specific tasks and responsibilities for the project, as detailed below.

Aidana:

Aidana was responsible for creating the ERD for the project. She also wrote the procedure that does group by information and the procedure that uses SQL%ROWCOUNT to determine the number of rows affected.

This procedure groups tuples in Product table by Supp_ID, and shows if some products were supplied by the same supplier:

```
CREATE OR REPLACE PROCEDURE proc_group
IS
BEGIN
  FOR i IN (
    SELECT Supp_ID, COUNT(*) AS Sup_count
    FROM Product
    GROUP BY Supp_ID
  )
  LOOP
    DBMS_OUTPUT.PUT_LINE(i.Supp_ID || ': ' ||  i.Sup_count);
  END LOOP;
END;
```

Results | Explain | Describe | Saved SQL | History

```
Procedure created.
```

0.02 seconds

```
begin
proc_group;
end;
```

Results | Explain | Describe | Saved S

```
1001.3: 1
1002.3: 1
1008.3: 2
1010.1: 1
1011.3: 1
1003.1: 1
1004.1: 1
1008.1: 2
1009.2: 1
1001.1: 3
1005.1: 2
1007.1: 1
```

This procedure counts the number of rows that were affected by UPDATE Card_Item table when date of bank card were expired, it adds 2 more years to the current date:

```
CREATE OR REPLACE PROCEDURE row_count(
    c_expired_date IN DATE
)
IS
BEGIN
    UPDATE CARD_ITEM
    SET Expired_date = c_expired_date
    WHERE Expired_date < SYSDATE;

    DBMS_OUTPUT.PUT_LINE('Number of rows updated: ' || SQL%ROWCOUNT);
END;
```

Results | Explain | Describe | Saved SQL | History

```
Procedure created.
```

0.02 seconds

```
begin
DECLARE
    c_expired_date DATE := add_months(SYSDATE, 24);
BEGIN
row_count(c_expired_date);
END;
end;

-- select *
```

Results | Explain | Describe | Saved SQL | History

```
Number of rows updated: 3

Statement processed.
```

0.12 seconds

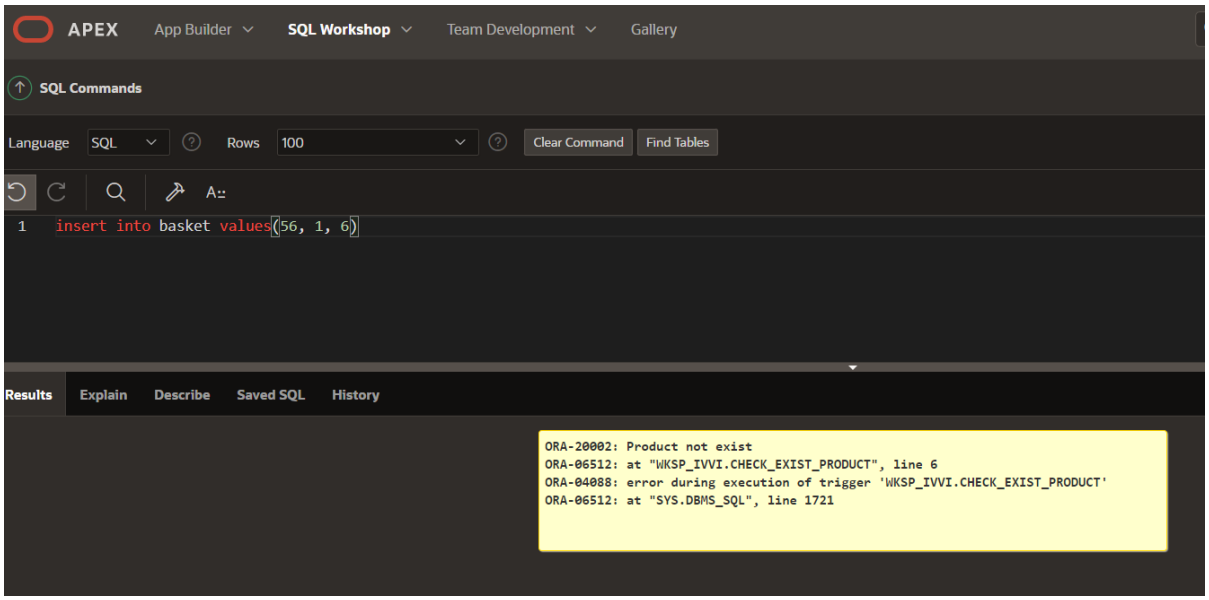
Results | Explain | Describe | Saved SQL | History

3	4400 4300 5500 8902	459	05/23/2023	16871
4	4400 4300 5500 8903	500	07/03/2023	4115
5	4400 4300 5500 8904	157	05/09/2023	3080
6	4400 4300 5501 8905	219	04/24/2025	2768
7	4400 4300 5501 8906	273	08/09/2023	19576
8	4400 4300 5501 8907	449	07/09/2023	4720
9	4400 4300 5501 8908	463	08/08/2023	2959

Ali:

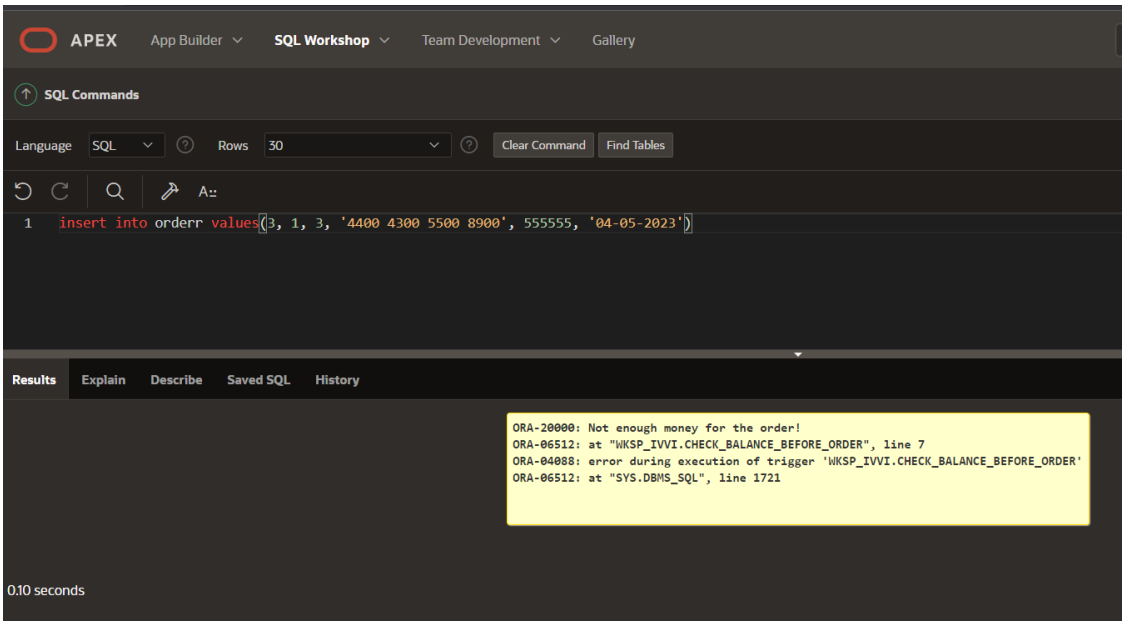
Ali was responsible for creating two triggers. The first trigger checks the existence of the exact product, and the second trigger checks the balance of the customer:

```oracle_plsql
create or replace trigger check_exist_product
before insert on basket
for each row
declare
amount number;
begin
    amount := get_product_amount(:new.prod_id);
    if amount = 0 then
        raise_application_error(-20002, 'Product not exist');
    end if;
end;
```

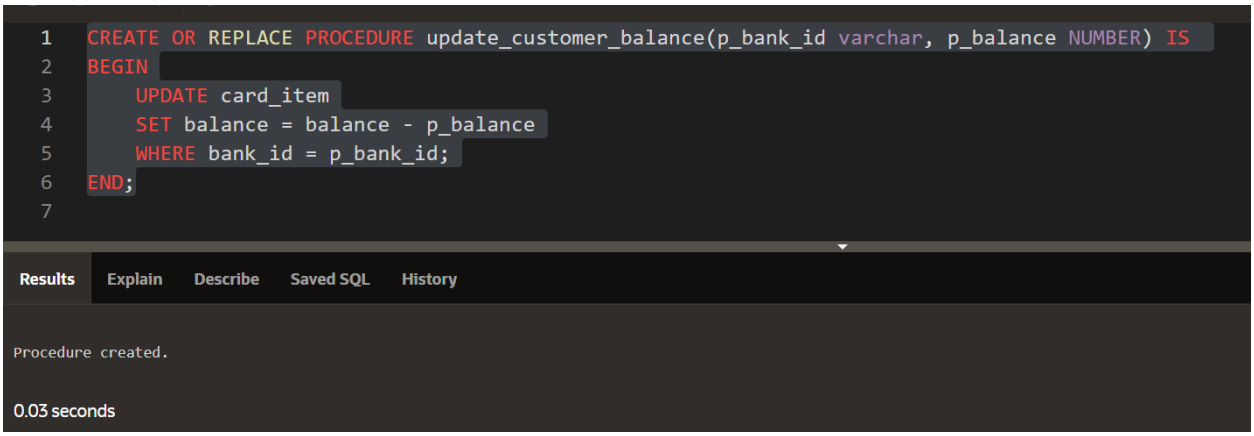



```oracle_plsql
CREATE OR REPLACE TRIGGER check_balance_before_order
BEFORE INSERT ON orderr
FOR EACH ROW
DECLARE
    amount NUMBER;
BEGIN
    amount := get_customer_balance(:NEW.bank_id);

    IF amount < :NEW.total_cost THEN
        RAISE_APPLICATION_ERROR(-20000, 'Not enough money for the order!');
    ELSE
        update_customer_balance(:NEW.bank_id, :NEW.total_cost);
    END IF;
END;
```



He also created a function that returns the number of exact products. Additionally, he wrote the procedure that withdraws money from accounts and the function that returns money to the customer.



```
CREATE OR REPLACE PROCEDURE update_customer_balance(p_bank_id varchar, p_balance NUMBER) IS
BEGIN
    UPDATE card_item
    SET balance = balance - p_balance
    WHERE bank_id = p_bank_id;
END;
```

Results Explain Describe Saved SQL History

Procedure created.

0.03 seconds

```
create or replace function get_product_amount(b_prod_id number)
return number is
    r_amount number;
begin
    select amount into r_amount
    from product
    where prod_id = b_prod_id;
    return r_amount;
end;
```

Results | Explain | Describe | Saved SQL | History

```
Function created.
```

0.01 seconds

```
CREATE OR REPLACE FUNCTION get_customer_balance(p_bank_id varchar)
RETURN NUMBER IS
    amount NUMBER;
BEGIN
    SELECT balance INTO amount
    FROM card_item
    WHERE bank_id = p_bank_id;
    RETURN amount;
END;
```

Results | Explain | Describe | Saved SQL | History

```
Function created.
```

0.01 seconds

Yernar:

Yernar was responsible for creating the function that counts the number of records for each table.

```
create or replace function get_cnt_of_customer
return number
as number_of_table number := 0;
begin
    select count(*) into number_of_table from customer;
    return number_of_table;
end;
```

Results | Explain | Describe | Saved SQL | History

Function created.

0.02 seconds

```
DECLARE
v_num NUMBER;
BEGIN
v_num := get_cnt_of_customer();
dbms_output.put_line('The number of customer: ' || v_num);
END;
```

Results | Explain | Describe | Saved SQL | History

The number of customer: 21

Statement processed.

0.11 seconds

He also created the trigger before insert on any entity, which shows the current number of rows in the table for each table.

```
create or replace trigger write_before_insert_on_customer
before insert on customer
for each row
declare
number_of_table number;
begin
   select count(*) into number_of_table from customer;
   DBMS_OUTPUT.PUT_LINE('current number of rows in the table: ' || number_of_table);
end;



insert into customer values(44, 'Dinara Zhaki', '210103379', '87044213254', 'Las_Vegas');
```

Results | Explain | Describe | Saved SQL | History

Trigger created.

0.05 seconds

```

insert into customer values(44, 'Dinara Zhaki', '210103379', '87044213254', 'Las_Vegas');
```

Results | Explain | Describe | Saved SQL | History

current number of rows in the table: 21

1 row(s) inserted.

0.12 seconds

Dinara:

Dinara was responsible for creating the trigger that checks if the products are expired when ordering. If the product is expired, raises an application with the message, "Sorry, but your ordered product is expired. Please, retry." She also created the function that returns the ID of the product that

was ordered by cust_id using only cust_id. Additionally, she created the function that returns the expiration date of the product that was ordered by cust_id using only cust_id.

```
create or replace trigger check_expired_date
after insert on orderr
for each row
declare
  current_date date;
  expired_date date;
begin
  current_date:=SYSDATE;
  expired_date:=get_expired_date(:new.cust_id);
  if current_date > expired_date then
    raise_application_error(-20001,'Sorry, but your ordered product is expired. Please,retry.');
  end if;
end;

insert into orderr
values (3,3,3,'4400 4300 5500 8902',4875,'04-20-2023');
```

Results | Explain | Describe | Saved SQL | History

```
ORA-20001: Sorry, but your ordered product is expired. Please,retry.
ORA-06512: at "WKSP_IVVI.CHECK_EXPIRED_DATE", line 8
ORA-04088: error during execution of trigger 'WKSP_IVVI.CHECK_EXPIRED_DATE'
ORA-06512: at "SYS.DBMS_SQL", line 1721
```

```
create or replace function get_expired_date(n_cust_id number)
return date is
  p_expired_date date;
begin
  select expired_date into p_expired_date
  from product
  where   get_prod(n_cust_id)=prod_id;
  return p_expired_date;
end;
```

Results | Explain | Describe | Saved SQL | History

```
Function created.
```

0.04 seconds

```oracle
create or replace function get_prod(b_cust_id number)
return number is
    b_prod_id number;
    cursor prodd is
      select distinct prod_id
      from basket
      where cust_id=b_cust_id;
begin
   open prodd;
   loop
    fetch prodd into b_prod_id;
    exit when prodd%NOTFOUND;
   end loop;
   close prodd;
   return b_prod_id;
end;
```

Results | Explain | Describe | Saved SQL | History

```
Function created.
```

0.01 seconds

Zhanarys:

Zhanarys was responsible for creating the trigger with a user-defined exception, which disallows entering the title of bank_id to be less than or greater than 16 digits.

```oracle
create or replace trigger check_bank_id_length
before insert on card_item
for each row
declare
  bank_id_length_by_user integer;
  invalid_data exception;
begin
  bank_id_length_by_user := length(:new.bank_id);
  if bank_id_length_by_user <> 16 then
```

Results | Explain | Describe | Saved SQL | History

```
Trigger created.
```

0.06 seconds

```oracle
create or replace trigger check_bank_id_length
before insert on card_item
for each row
declare
  bank_id_length_by_user integer;
  invalid_data exception;
begin
  bank_id_length_by_user := length(:new.bank_id);
  if bank_id_length_by_user <> 16 then
    raise invalid_data;
  end if;
exception
  when invalid_data then
  raise_application_error(-20001, 'the length of the bank_id must not exceed or be less than 16 characters :)');
end;
```

```
ORA-20001: the length of the bank_id must not exceed or be less than 16 characters
:)
ORA-06512: at "WKSP_IVVI.CHECK_BANK_ID_LENGTH", line 11
ORA-04088: error during execution of trigger 'WKSP_IVVI.CHECK_BANK_ID_LENGTH'
ORA-06512: at "SYS.DBMS_SQL", line 1721


1. INSERT INTO Card_Item (cust_id, bank_id, cvv, expired_date, balance) VALUES (30,
'567890123', 333, '07-07-2023', 8888);
```

CONCLUSION

In conclusion, the team members had specific responsibilities and tasks for the Online Shop project of DBMS 2. Each member contributed to the success of the project by creating different components, including triggers, functions, and procedures. Together, they created a functioning online shop system that allows customers to order products, get them delivered to their addresses, and pay for them using different payment methods.