Midterm Defense Presentation

STUDENTS

Duman Yessenbay
Omargali Tlepbergenov
Bekbolat Sabir

LECTURER

Azamat Serek

Database topic - Online store

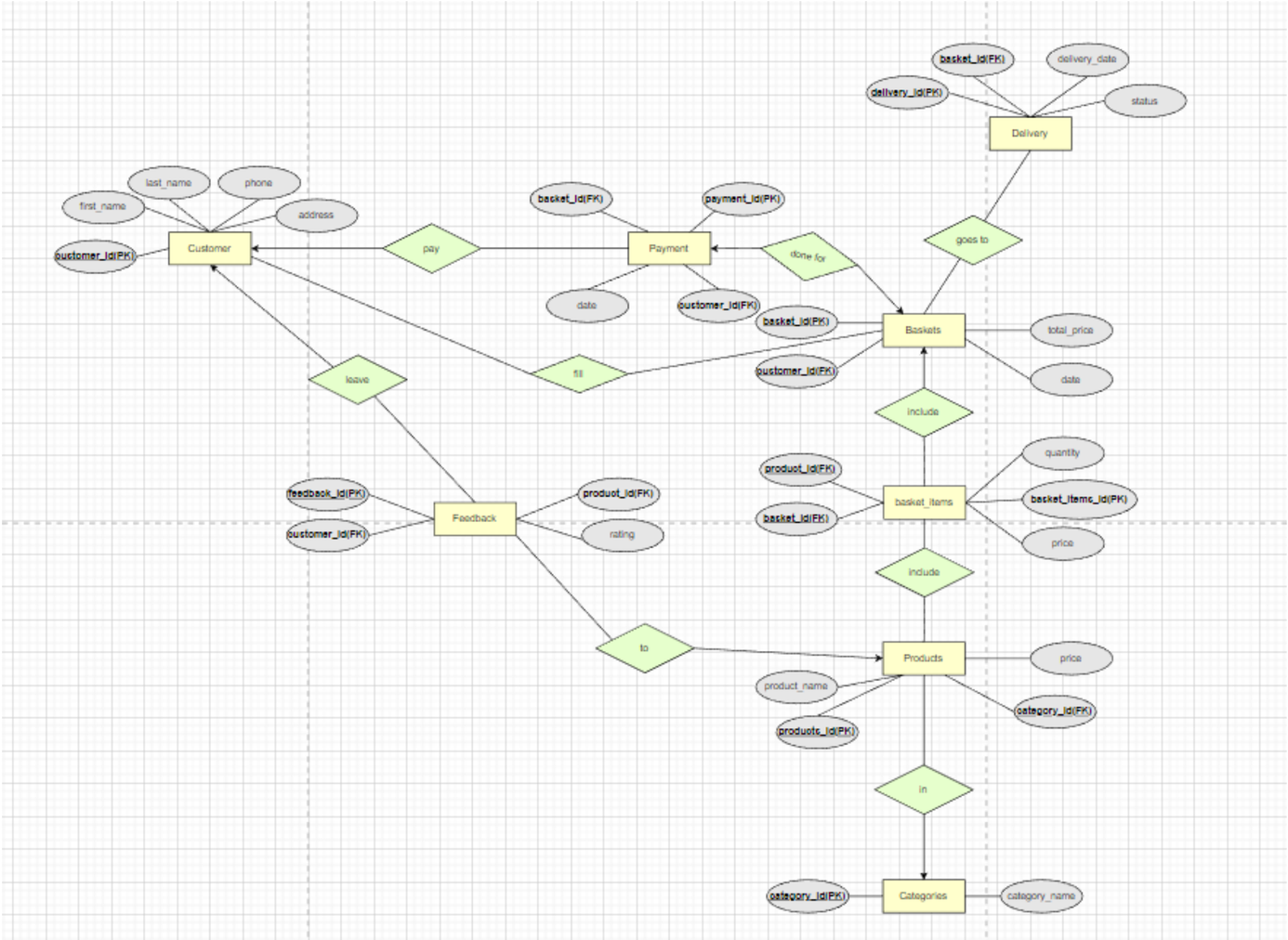
Our database is for all ages

The base is intended for online ordering products

The end users of the database are all people

We took the idea from real life(like sulpak)

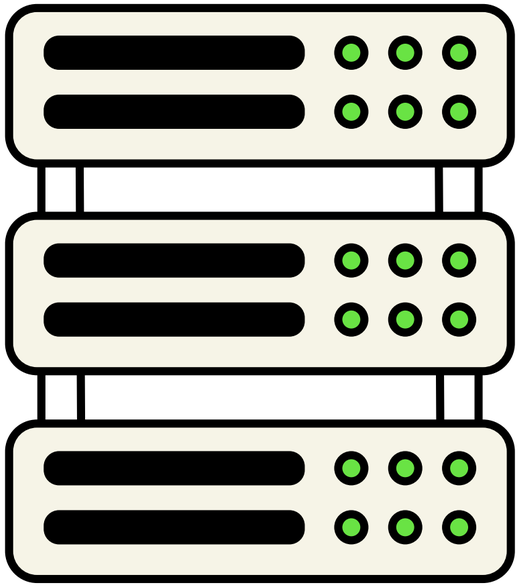
ER diagram



https://drive.google.com/file/d/1iOxZOQm88byWSkWwK_Dj9nwf8Su4pCZD/view?usp=sharing

1NF

- There should be no duplicate rows in the table
- There should be no arrays and lists
- The column must store data of the same type



We have all tables that satisfy the 1NF condition

2NF

- The table should be in the second normal form (1NF)
- All non-key columns of the table must depend on the full key (if it is composite)

We have also all tables that satisfy the 2NF condition

Products

2-NF

Product_id	Product_name	price	Category_id

Payment

2-NF

Payment_id	Basket_id	Payment_date	Customer_id

3NF

- The table should be in the second normal form (2NF)
- There should be no transitive dependency in the table. Non-key columns should not depend on the values of other non-key columns.

Customer

3-NF

Customer_id	first_name	Last_name	Phone	Address

Products

2-NF

Product_id	Product_name	price	Category_id

3-NF

Product_id	price	Category_id

Product_id	Product_name

Categories

3-NF

Category_id	Category_name

Baskets

3-NF

Basket_id	Customer_id	Total_price

Basket_Items

3-NF

Basket_items_id	Product_id	quantity	price	Basket_

Payment

2-NF

Payment_id	Basket_id	Payment_date	Customer_id

3-NF

Payment_id	Payment_date

Basket_id	Customer

Delivery

3-NF

Delivery_id	Basket_id	Delivery_date	Status

Feedback

3-NF

Feedback_id	Customer_id	Product_id	Rating

Before insert trigger

```oracle_plsql
create or replace TRIGGER trigger_show_current_num_of_rows
BEFORE INSERT ON basket_items
FOR EACH ROW
DECLARE
    cnt NUMBER;
BEGIN
    SELECT COUNT(*) INTO cnt FROM basket_items;
    DBMS_OUTPUT.PUT_LINE('Current number of rows: ' || cnt);
END;
```

this trigger is created to display the current number of rows in the "basket_items" table whenever a new row is inserted into the table. It does this by executing a SELECT COUNT(*) statement and printing the result using the DBMS_OUTPUT.PUT_LINE function.

New basket for new customer

```sql
create or replace TRIGGER trigger_basket_for_new_customer
AFTER INSERT ON Customers
FOR EACH ROW
DECLARE
    cnt INTEGER;
BEGIN
    SELECT COUNT(*) + 1 INTO cnt FROM Baskets;
    INSERT INTO Baskets(basket_id, customer_id, total_price)
    VALUES (cnt, :NEW.customer_id, 0);
END;
```

this trigger creates a new basket record for each new customer added to the Customers table. It does this by taking the current number of baskets in the Baskets table and adding 1 to it to get the new basket ID, then inserting a new entry in the Baskets table with the corresponding values.

Calculate total price in basket_items table

```
create or replace TRIGGER trigger_insert_total_price_basket_item
BEFORE INSERT or UPDATE ON basket_items
FOR EACH ROW
DECLARE
    price DECIMAL(10, 2);
BEGIN
    SELECT price INTO price FROM Products WHERE product_id = :NEW.product_id;
    :NEW.total_price := price * :NEW.quantity;
END;
```

this trigger calculates and sets the total price for each new or updated row in the "basket_items" table based on the product price and quantity. It does this by retrieving the product price from the Products table using the product ID stored in the ":NEW" keyword, then calculating the total price and setting the value of the "total_price" column in the ":NEW" keyword. to the calculated value.

for example here we insert a new row

```
INSERT INTO BASKET_ITEMS(BASKET_ITEMS_ID, PRODUCT_ID, QUANTITY, BASKET_ID) VALUES (21, 1, 4, 1)
```

as we can see, we do not need to calculate the total price, our trigger will do it for us

BASKET_ITEMS_ID	PRODUCT_ID	QUANTITY	TOTAL_PRICE	BASKET_ID
21	1	4	2400000	1

Calculate total price in baskets table

```oracle_plsql
create or replace TRIGGER trigger_update_baskets_total_price
FOR INSERT OR UPDATE OR DELETE ON basket_items
COMPOUND TRIGGER
    TYPE coll IS TABLE OF Baskets.basket_id%TYPE;
    basket_ids coll := coll();

    BEFORE STATEMENT IS BEGIN
        basket_ids.DELETE;
    END BEFORE STATEMENT;

    AFTER EACH ROW IS BEGIN
        IF :NEW.basket_id NOT MEMBER OF basket_ids THEN
            basket_ids.EXTEND;
            basket_ids(basket_ids.LAST) := :NEW.basket_id;
        END IF;
    END AFTER EACH ROW;

    AFTER STATEMENT IS BEGIN
        FOR i IN 1..basket_ids.COUNT LOOP
            UPDATE Baskets SET total_price = (
                SELECT SUM(total_price) FROM basket_items WHERE basket_id = basket_ids(i)
            ) WHERE basket_id = basket_ids(i);
        END LOOP;
    END AFTER STATEMENT;
END;
```

this trigger collects total prices with similar basket_id from the basket_items table, sums them up, and updates the total_price in the baskets table

here we insert new basket items

BASKET_ITEMS_ID	PRODUCT_ID	QUANTITY	TOTAL_PRICE	BASKET_ID
21	2	2	1000000	1
1	1	3	1800000	1

as we can see, total_price is inserted into baskets table

BASKET_ID	CUSTOMER_ID	TOTAL_PRICE
1	1	2800000

A check

```
create or replace TRIGGER trigger_show_total_price_payment
after INSERT ON payment
FOR EACH ROW
DECLARE
    total_p NUMBER;
    cust_id number;
BEGIN
    SELECT total_price INTO total_p FROM baskets where :new.basket_id = basket_id;
    SELECT customer_id INTO cust_id from customers where :new.customer_id = customer_id;
    DBMS_OUTPUT.PUT_LINE(cust_id || ' total price: ' || total_p);
END;
```

this trigger works like a check

```
INSERT INTO PAYMENT(PAYMENT_ID, BASKET_ID, PAYMENT_DATE, CUSTOMER_ID) VALUES (1, 1, '01/01/2022', 1);
```

Results | Explain | Describe | Saved SQL | History

```
1 total price: 2800000

1 row(s) inserted.
```

when we insert the row, trigger shows total price for the basket

Function which counts the number of records

```oracle
create or replace FUNCTION count_records (
    table_name IN VARCHAR2
) RETURN NUMBER
IS
    record_count NUMBER;
BEGIN
    EXECUTE IMMEDIATE 'SELECT COUNT(*) FROM ' || table_name INTO record_count;
    RETURN record_count;
END;
```

this is a function called "count_records" that takes in a table name as a parameter and returns the number of records in that table. The function first declares a local variable called "record_count" of type NUMBER to store the count of records. It then uses dynamic SQL to execute a SELECT COUNT(*) statement on the table passed as a parameter. The result of the query is stored in the "record_count" variable using the INTO clause. And the function returns the value of the "record_count" variable.

to call the "count_records" function for the "products" table, we can use the following code:

```oracle
BEGIN
  DBMS_OUTPUT.PUT_LINE('Number of records: ' || count_records('products'));
END;
```

as we can see we get the number of records in the products table

```
Number of records: 20
```

User-defined exception which disallows to enter title of item to be less than 5 characters

```oracle
create or replace TRIGGER trigger_exception_title
BEFORE INSERT ON products
FOR EACH ROW
DECLARE
  name_length EXCEPTION;
BEGIN
  IF LENGTH(:new.product_name) < 5 THEN
    RAISE name_length;
  ELSE
    DBMS_OUTPUT.PUT_LINE('Item added successfully.');
  END IF;
EXCEPTION
  WHEN name_length THEN
    RAISE_APPLICATION_ERROR(-20001, 'Error: Product name must be at least 5 characters');
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END;
```

this trigger is designed to fire before each insertion into the "products" table. It checks whether the length of the new product's name is less than 5 characters. If it is, then it raises a user exception called "name_length". If the length is equal to or greater than 5, it will display the message "Item added successfully" in the output.

```oracle
INSERT INTO PRODUCTS(PRODUCT_ID, PRODUCT_NAME, PRICE, CATEGORY_ID) VALUES (21, 'SONY', 450000, 3);
```

if we insert a product name that has a length less than 5 then we get a user-defined exception.

```
ORA-20001: Error: Product name must be at least 5 characters
```

III Procedure which does group by information

```sql
CREATE OR REPLACE PROCEDURE group_basket_items_info IS
  CURSOR basket_items_cur IS
    SELECT basket_id, SUM(quantity) AS total_quantity, SUM(total_price) AS total_price
    FROM basket_items
    GROUP BY basket_id;
BEGIN
  FOR basket_item_rec IN basket_items_cur LOOP
    DBMS_OUTPUT.PUT_LINE('Basket ID: '  basket_item_rec.basket_id
                         ' | Total Quantity: '  basket_item_rec.total_quantity
                         ' | Total Price: ' || basket_item_rec.total_price);
  END LOOP;
END;
/
BEGIN
  group_basket_items_info;
END;
```

Results | Explain | Describe | Saved SQL | History

```
Basket ID: 6 | Total Quantity: 5 | Total Price: 3588000
Basket ID: 14 | Total Quantity: 1 | Total Price: 199900
Basket ID: 1 | Total Quantity: 6 | Total Price: 3600000
Basket ID: 15 | Total Quantity: 1 | Total Price: 400000
Basket ID: 7 | Total Quantity: 1 | Total Price: 290000
Basket ID: 11 | Total Quantity: 1 | Total Price: 260000
Basket ID: 8 | Total Quantity: 2 | Total Price: 900000
Basket ID: 2 | Total Quantity: 3 | Total Price: 1500000
Basket ID: 12 | Total Quantity: 2 | Total Price: 399800
Basket ID: 4 | Total Quantity: 2 | Total Price: 860000
Basket ID: 10 | Total Quantity: 1 | Total Price: 200000
Basket ID: 5 | Total Quantity: 2 | Total Price: 939990
```

210103128@stu.sdu.edu.kz bekbolat2004 en

This PL/SQL procedure retrieves information about basket items, groups the results by basket ID, and outputs the total quantity and price for each basket.

The procedure uses a cursor to select data from the "basket_items" table, sums up the quantity and total price columns for each basket, and then loops through the results and displays the basket ID, total quantity, and total price for each basket item using the DBMS_OUTPUT.PUT_LINE function. The final block of code calls the procedure to execute it and display the results.

IV Procedure which uses SQL%ROWCOUNT to determine the number of rows affected

Language SQL Rows 10 Clear Command Find Tables

```sql
create or replace PROCEDURE update_customers
(
  p_id IN NUMBER,
  p_value IN VARCHAR2
)
IS
  row_count INTEGER;
BEGIN
  UPDATE customers SET first_name = p_value WHERE customer_id = p_id;
  row_count := SQL%ROWCOUNT;

  IF row_count = 0 THEN
    dbms_output.put_line('No rows updated.');
  ELSIF row_count = 1 THEN
    dbms_output.put_line('1 row updated.');
  ELSE
    dbms_output.put_line(row_count || ' rows updated.');
  END IF;
END;
BEGIN
  update_customers(25, 'Henry');
END;
```

Results Explain Describe Saved SQL History

```
1 row updated.

Statement processed.

0.02 seconds
```

That PL/SQL procedure named "update_customers" takes two input parameters, an ID number and a string value. It updates the "first_name" column of the "customers" table with the provided string value for the customer with the specified ID number. It then retrieves the number of rows affected by the update and uses an IF-ELSE statement to print out a message indicating the number of rows updated.

The final block of code calls the procedure with an example ID number and string value to update the corresponding customer's first name.

SDU

THANK YOU