Course: DBMS

AY: 2023-24

Table of Contents

Sr no.	Topic	Page no.
1	Storyline	
2	Components of Database Design	
3	Entity Relationship Diagram	
4	Relational Model	
5	Normalization	
6	SQL Queries	
7	Learning from the Project	
8	Project Demonstration	
9	Self-learning beyond classroom	
10	Learning from the project	

8	Challenges faced	
9	Conclusion	

I. Storyline

Title: "From Clicks to Bricks: Unraveling the Eshop Saga in Modern Retail Evolution"

Introduction

In today's digital age, the realm of retail has undergone a significant transformation with the advent of e-commerce platforms. Eshop emerges as a pioneering project, aimed at revolutionizing the traditional retail experience by seamlessly integrating technology into the shopping process. This report delves into the intricacies of Eshop, exploring its architecture, functionalities, and the underlying database management system that powers its operations.

Implementation

In implementing our Eshop project, we focused on creating a smooth and user-friendly experience. We designed a database system to store information about products, customers, orders, and payments. Our aim was to ensure that users can easily browse through products, add items to their cart, and complete transactions securely. By integrating the database with our website, we enabled real-time updates on product availability and order status. Additionally, we implemented features like personalized recommendations and efficient search functionality to enhance the overall shopping experience.

II. Components of Database Design

2.1 Address:
Attributes: user_id, street_address, city, state, pin_code, country

2.2 Categories:
Attributes: category_id, category_name

2.3 Orders:
Attributes: order_id, product_id, user_id, quantity, date_ordered

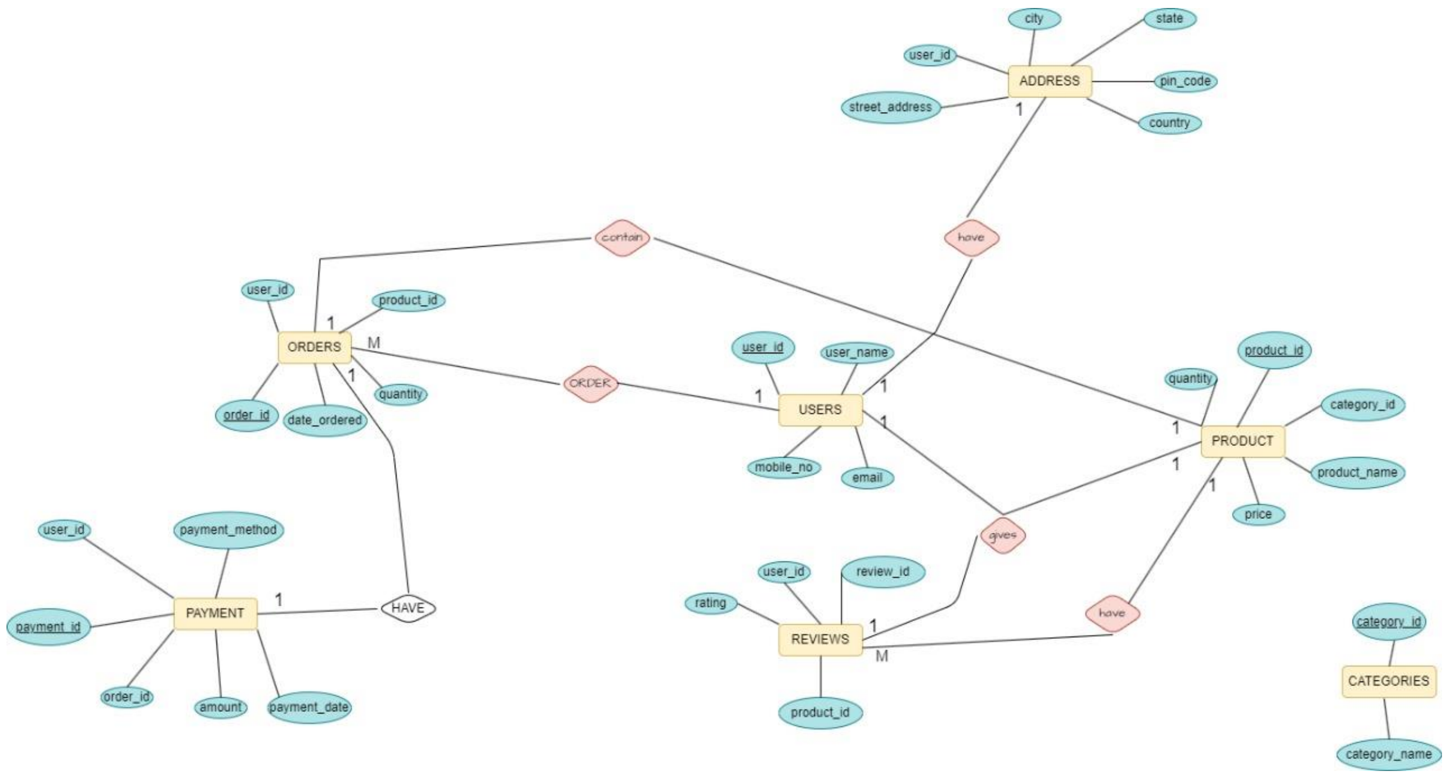
2.4 Payment:
Attributes: payement_id, user_id, order_id, payement_date, amount, payment_method

2.5 Product:
Attributes: product_id, category_id, product_name, price, quantity

2.6 Reviews:
Attributes: review_id, product_id, user_id, rating

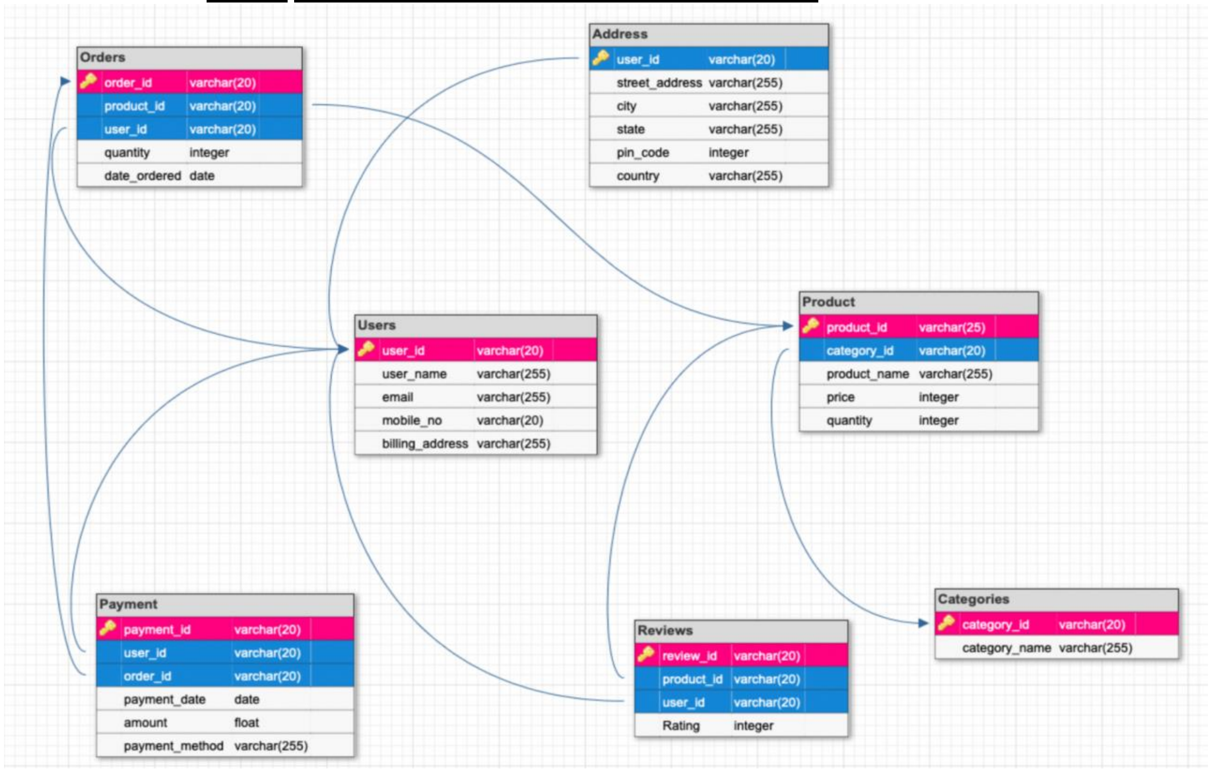
2.7 Users:
Attributes: user_id, user_name, email, mobile_no, billing_address

III. Entity Relationship Diagram



city
state
user_id
ADDRESS
pin_code
street_address
country
1
contain
have
user_id
product_id
1
ORDERS
M
1
order_id
date_ordered
quantity
ORDER
user_id
user_name
1
USERS
1
1
mobile_no
email
quantity
product_id
category_id
1
PRODUCT
1
product_name
1
price
user_id
payment_method
payment_id
PAYMENT
1
HAVE
order_id
amount
payment_date
gives
user_id
review_id
rating
REVIEWS
1
M
have
product_id
category_id
CATEGORIES
category_name

IV. Relational Schema



V. Normalization

To determine the highest normal form (NF) of the functional dependencies of each table, we need to analyze them one by one and check if they satisfy the requirements of each normal form.

Address Table (address):
This table appears to be in the First Normal Form (1NF) as there are no repeating groups and each attribute contains atomic values.
Functional Dependencies:
user_id → {street_address, city, state, pin_code, country}

Categories Table (categories):
This table appears to be in 1NF as well.
Functional Dependencies:
category_id → category_name

Orders Table (orders):
This table seems to be in 1NF.
Functional Dependencies:

order_id → {product_id, user_id, quantity, date_ordered}

Payment Table (payment):
This table also appears to be in 1NF.
Functional Dependencies:
payment_id → {user_id, order_id, payment_date, amount, payment_method}

Product Table (product):
This table seems to be in 1NF.
Functional Dependencies:
product_id → {category_id, product_name, price, quantity}\

Reviews Table (reviews):
Appears to be in 1NF.
Functional Dependencies:
review_id → {product_id, user_id, rating}

Users Table (users):
Appears to be in 1NF.
Functional Dependencies:
user_id → {user_name, email, mobile_no, billing_address}

Functional dependencies

Functional dependencies describe the relationship between attributes within a table. They are typically expressed as X → Y, where X and Y are sets of attributes, and Y is functionally dependent on X. In other words, for every unique combination of values in X, there is exactly one corresponding value in Y.

Let's analyze the functional dependencies for each table:

Address Table (address):
user_id → {street_address, city, state, pin_code, country}

Categories Table (categories):
category_id → category_name

Orders Table (orders):

order_id → {product_id, user_id, quantity, date_ordered}

Payment Table (payment):
payment_id → {user_id, order_id, payment_date, amount, payment_method}

Product Table (product):
product_id → {category_id, product_name, price, quantity}

Reviews Table (reviews):
review_id → {product_id, user_id, rating}

Users Table (users):
user_id → {user_name, email, mobile_no, billing_address}

These functional dependencies help in understanding how the attributes are related to each other within each table and can be useful for database normalization and query optimization.

VI. SQL Queries

1.Create Query

```
CREATE TABLE Address (

   user_id INT PRIMARY KEY,

   street_address VARCHAR (255),

   city VARCHAR (50),

   state VARCHAR (50),

   pin_code VARCHAR(10),

   country VARCHAR(50)

);


CREATE TABLE Categories (
   category_id INT PRIMARY KEY,

   category_ name VARCHAR(100)

);


CREATE TABLE Orders (

   order_id INT PRIMARY KEY,

   product_id INT,

   user_id INT,

   quantity INT,

   date_ordered DATE,

   FOREIGN KEY (product_id) REFERENCES Product(product_id),

   FOREIGN KEY (user_id) REFERENCES Users(user_id)

);


CREATE TABLE Payment (

   payment_id INT PRIMARY KEY,

   user_id INT,

   order_id INT,

   payment_date DATE,

   amount DECIMAL(10,2),

   payment_method VARCHAR(50),

   FOREIGN KEY (user_id) REFERENCES Users(user_id),

   FOREIGN KEY (order_id) REFERENCES Orders(order_id)
```

```
 );
CREATE TABLE Product (
    product_id INT PRIMARY KEY,
    category_id INT,
    product_name VARCHAR(255),
    price DECIMAL(10,2),
    quantity INT,
    FOREIGN KEY (category_id) REFERENCES Categories(category_id)
 );

 CREATE TABLE Reviews (
    review_id INT PRIMARY KEY,
    product_id INT,
    user_id INT,
    rating INT,
    FOREIGN KEY (product_id) REFERENCES Product(product_id),
    FOREIGN KEY (user_id) REFERENCES Users(user_id)
 );
 CREATE TABLE Users (
    user_id INT PRIMARY KEY,
    user_name VARCHAR(50),
    email VARCHAR(100),
    mobile_no VARCHAR(15),
    billing_address INT,

    FOREIGN KEY (billing_address) REFERENCES Address(user_id)

);
```

2. Insertion

```
INSERT INTO Address (user_id, street_address, city, state,
    pin_code, country) VALUES
(1, '123 Main St', 'New York', 'NY', '10001', 'USA'),
(2, '456 Elm St', 'Los Angeles', 'CA', '90001', 'USA'),
(3, '789 Oak St', 'Chicago', 'IL', '60601', 'USA'),
(4, '101 Pine St', 'Houston', 'TX', '77002', 'USA'),
(5, '202 Maple St', 'Miami', 'FL', '33101', 'USA');

INSERT INTO Categories (category_id, category_name)
    VALUES
(1, 'Electronics'),
(2, 'Clothing'),
(3, 'Books'),
(4, 'Home & Kitchen'),
(5, 'Sports & Outdoors');
```

```
INSERT INTO Orders (order_id, product_id, user_id, quantity,
      date_ordered) VALUES
(1, 101, 1, 2, '2024-03-21'),
(2, 102, 2, 1, '2024-03-20'),
(3, 103, 3, 3, '2024-03-19'),
(4, 104, 4, 1, '2024-03-18'),
(5, 105, 5, 2, '2024-03-17');



INSERT INTO Payment (payment_id, user_id, order_id,
      payment_date, amount, payment_method) VALUES
(1, 1, 1, '2024-03-21', 100.00, 'Credit Card'),
(2, 2, 2, '2024-03-20', 50.00, 'PayPal'),
(3, 3, 3, '2024-03-19', 150.00, 'Credit Card'),
(4, 4, 4, '2024-03-18', 75.00, 'Debit Card'),
(5, 5, 5, '2024-03-17', 90.00, 'PayPal');

INSERT INTO Product (product_id, category_id, product_name,
      price, quantity) VALUES
(101, 1, 'Smartphone', 500.00, 10),
(102, 2, 'T-shirt', 20.00, 50),
(103, 3, 'The Great Gatsby', 10.00, 30),
(104, 4, 'Knife Set', 50.00, 20),
(105, 5, 'Soccer Ball', 30.00, 25);


INSERT INTO Reviews (review_id, product_id, user_id, rating)
    VALUES
(1, 101, 1, 5),
(2, 102, 2, 4),
(3, 103, 3, 5),
(4, 104, 4, 4),
(5, 105, 5, 3);

INSERT INTO Users (user_id, user_name, email, mobile_no,
    billing_address) VALUES
(1, 'John Doe', 'john@example.com', '123-456-7890', 1),
(2, 'Jane Smith', 'jane@example.com', '987-654-3210', 2),
(3, 'Michael Johnson', 'michael@example.com', '555-555-5555', 3),
(4, 'Emily Brown', 'emily@example.com', '111-222-3333', 4),
(5, 'David Wilson', 'david@example.com', '444-444-4444', 5);
```

3. Order by:

```
SELECT * FROM Users ORDER BY user_name DESC;
```

4.Group by:
SELECT product_id, AVG(rating) AS average_rating FROM Reviews GROUP BY product_id;

5.Delete:
DELETE FROM Product WHERE product_id = 104;

6.Update:
UPDATE Product SET quantity = 15 WHERE product_id = 101;

Clauses:
1. Between:
   SELECT * FROM Product WHERE price BETWEEN 20 AND 50;

2.Like:
SELECT * FROM Users WHERE user_name LIKE 'J%';

3.IN:
SELECT * FROM Product WHERE category_id IN (1, 2, 3);

4. AS:
SELECT user_id AS ID, user_name AS Name FROM Users;

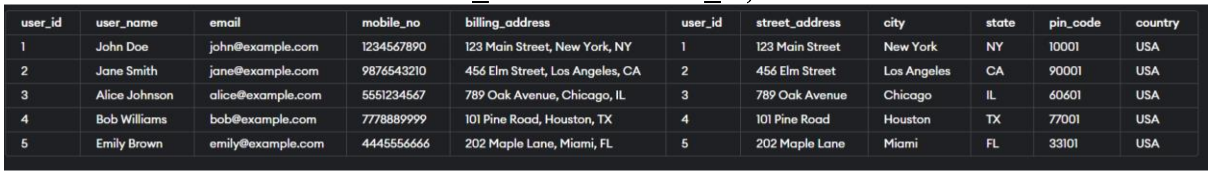
5. Subqueries:

```
SELECT DISTINCT user_id
FROM Orders
WHERE product_id IN (SELECT product_id FROM Product
WHERE category_id = 1);
```

Join

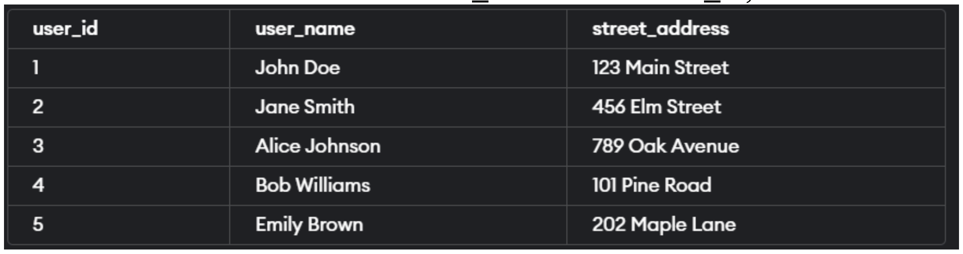
1. Inner join

```
SELECT *
FROM users
INNER JOIN address ON users.user_id = address.user_id;
```



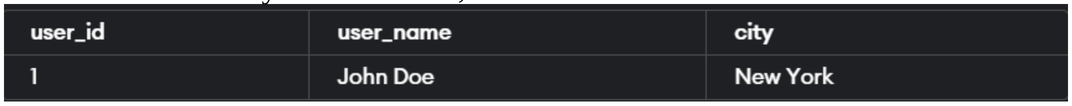
user_id	user_name	email	mobile_no	billing_address	user_id	street_address	city	state	pin_code	country
1	John Doe	john@example.com	1234567890	123 Main Street, New York, NY	1	123 Main Street	New York	NY	10001	USA
2	Jane Smith	jane@example.com	9876543210	456 Elm Street, Los Angeles, CA	2	456 Elm Street	Los Angeles	CA	90001	USA
3	Alice Johnson	alice@example.com	5551234567	789 Oak Avenue, Chicago, IL	3	789 Oak Avenue	Chicago	IL	60601	USA
4	Bob Williams	bob@example.com	7778889999	101 Pine Road, Houston, TX	4	101 Pine Road	Houston	TX	77001	USA
5	Emily Brown	emily@example.com	4445556666	202 Maple Lane, Miami, FL	5	202 Maple Lane	Miami	FL	33101	USA

```
SELECT users.user_id, users.user_name, address.street_address
FROM users
INNER JOIN address ON users.user_id = address.user_id;
```



user_id	user_name	street_address
1	John Doe	123 Main Street
2	Jane Smith	456 Elm Street
3	Alice Johnson	789 Oak Avenue
4	Bob Williams	101 Pine Road
5	Emily Brown	202 Maple Lane

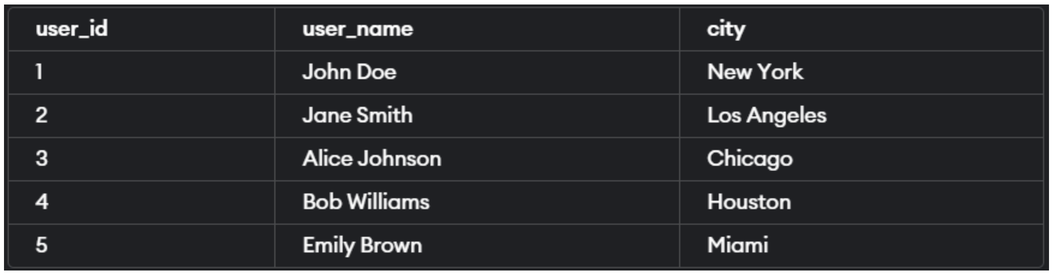
```
SELECT users.user_id, users.user_name, address.city
FROM users
INNER JOIN address ON users.user_id = address.user_id
WHERE address.city = 'New York';
```



user_id	user_name	city
1	John Doe	New York

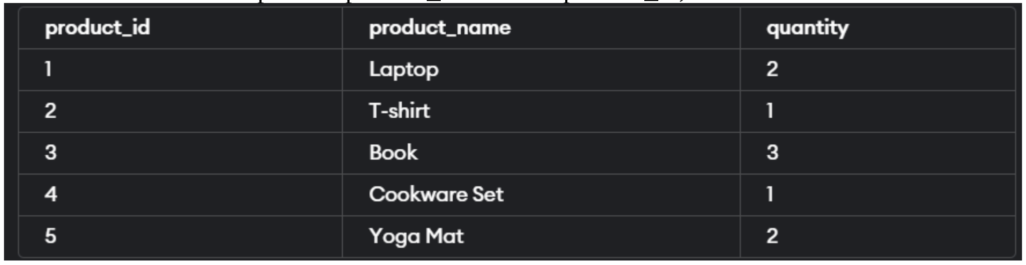
2. Left Join

```
SELECT users.user_id, users.user_name, address.city
FROM users
LEFT JOIN address ON users.user_id = address.user_id;
```



user_id	user_name	city
1	John Doe	New York
2	Jane Smith	Los Angeles
3	Alice Johnson	Chicago
4	Bob Williams	Houston
5	Emily Brown	Miami

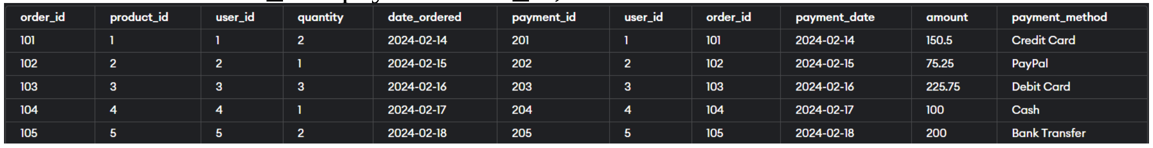
```
SELECT product.product_id, product.product_name, orders.quantity
FROM product
LEFT JOIN orders ON product.product_id = orders.product_id;
```



product_id	product_name	quantity
1	Laptop	2
2	T-shirt	1
3	Book	3
4	Cookware Set	1
5	Yoga Mat	2

3. Cross Join

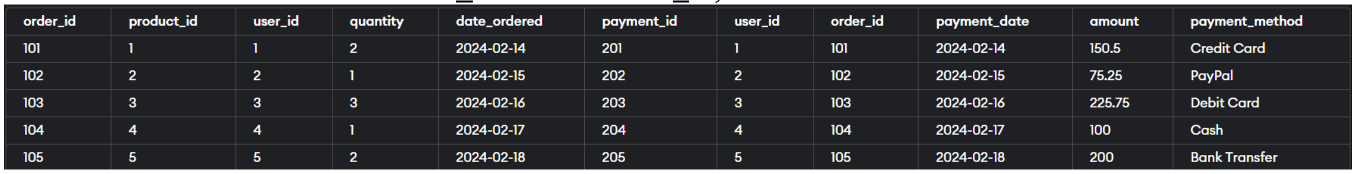
```
SELECT *
FROM orders
CROSS JOIN payment
WHERE orders.user_id = payment.user_id;
```



order_id	product_id	user_id	quantity	date_ordered	payment_id	user_id	order_id	payment_date	amount	payment_method
101	1	1	2	2024-02-14	201	1	101	2024-02-14	150.5	Credit Card
102	2	2	1	2024-02-15	202	2	102	2024-02-15	75.25	PayPal
103	3	3	3	2024-02-16	203	3	103	2024-02-16	225.75	Debit Card
104	4	4	1	2024-02-17	204	4	104	2024-02-17	100	Cash
105	5	5	2	2024-02-18	205	5	105	2024-02-18	200	Bank Transfer

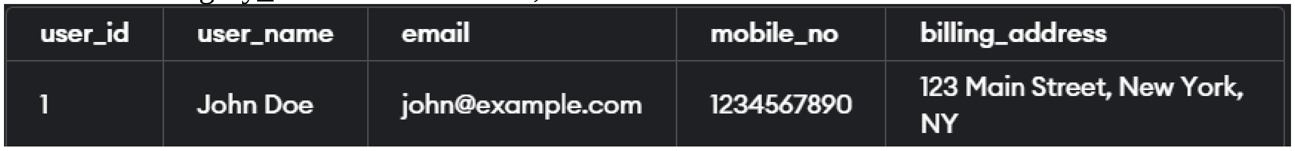
4. Right join

```
SELECT users.user_id, users.user_name, orders.order_id
FROM users
RIGHT JOIN orders ON users.user_id = orders.user_id;
```



order_id	product_id	user_id	quantity	date_ordered	payment_id	user_id	order_id	payment_date	amount	payment_method
101	1	1	2	2024-02-14	201	1	101	2024-02-14	150.5	Credit Card
102	2	2	1	2024-02-15	202	2	102	2024-02-15	75.25	PayPal
103	3	3	3	2024-02-16	203	3	103	2024-02-16	225.75	Debit Card
104	4	4	1	2024-02-17	204	4	104	2024-02-17	100	Cash
105	5	5	2	2024-02-18	205	5	105	2024-02-18	200	Bank Transfer

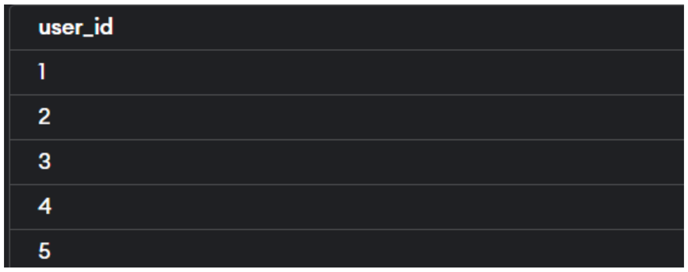
```
SELECT DISTINCT u.* FROM users u
JOIN orders o ON u.user_id = o.user_id
JOIN product p ON o.product_id = p.product_id
JOIN categories c ON p.category_id = c.category_id
WHERE c.category_name = 'Electronics';
```



user_id	user_name	email	mobile_no	billing_address
1	John Doe	john@example.com	1234567890	123 Main Street, New York, NY

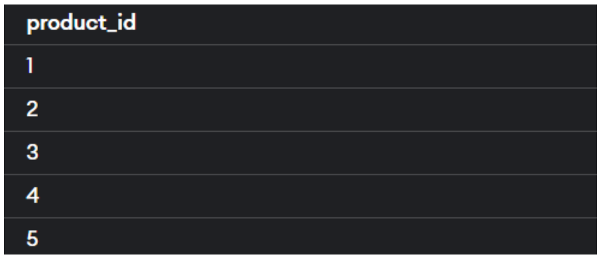
5. Set functions

SELECT user_id FROM orders
UNION
SELECT user_id FROM payment;



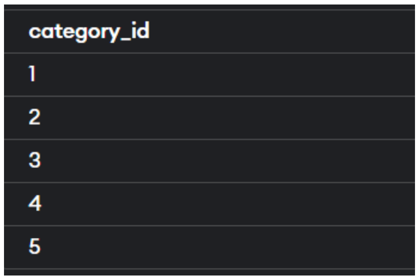
user_id
1
2
3
4
5

SELECT product_id FROM product
UNION
SELECT product_id FROM reviews;



product_id
1
2
3
4
5

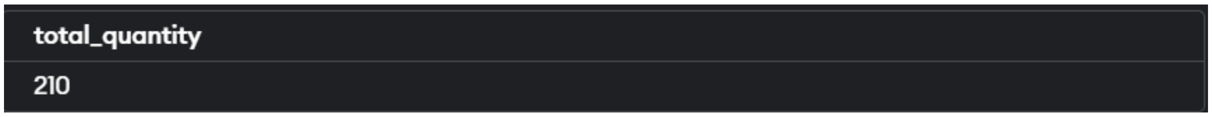
SELECT category_id FROM categories
UNION
SELECT category_id FROM product;



category_id
1
2
3
4
5

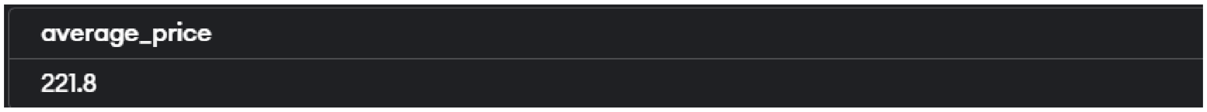
6. Aggregate Functions

SELECT SUM(quantity) AS total_quantity FROM product;



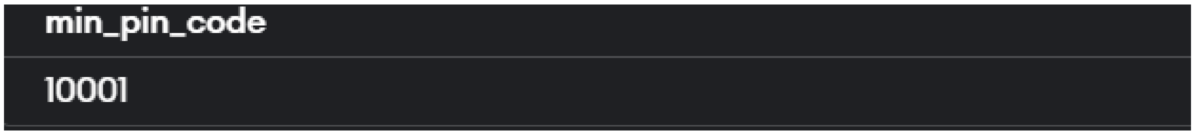
total_quantity
210

SELECT AVG(price) AS average_price FROM product;

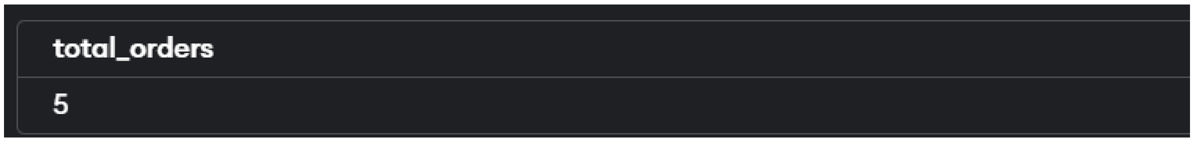


average_price
221.8

SELECT MIN(pin_code) AS min_pin_code FROM address;


```
min_pin_code
10001
```

SELECT COUNT(*) AS total_orders FROM orders;


```
total_orders
5
```

VI. Project demonstration

- Database Management System (DBMS): MySQL, PostgreSQL, SQLite, or MongoDB are popular choices for storing and managing project data.
- In SQL (Structured Query Language), demonstrating a project typically involves showcasing database design, data manipulation, and querying capabilities

VII. Self -Learning beyond classroom

- Self-learning beyond the classroom in database management has been instrumental in our professional development. Here are some new aspects we have learned :
- Advanced Database Design Patterns: While classroom education covers fundamental concepts of database design, self-learning has allowed me to explore advanced design patterns such as star schema, snowflake schema, and hybrid designs. Understanding these patterns has enabled me to design more efficient and scalable databases tailored to specific application requirements.
- Database Optimization Techniques: Beyond the basics of SQL queries and indexing, self-learning has introduced me to various database optimization techniques. This includes query optimization, indexing strategies, denormalization, and partitioning methods. Applying these techniques has significantly improved the performance of database systems I work with.
- By actively pursuing self-learning in database management beyond the classroom, I've been able to deepen my understanding of advanced concepts, emerging technologies, and best practices in database administration and development. This continuous learning journey has been essential in staying competitive and adapting to evolving trends in the field of database management.

VIII. Learning from the Project

Learning from a database management project can be multifaceted, encompassing both technical skills and broader insights into project management and problem-solving. Here are some aspects I might have learned from a database management project:

Database Design Principles:

Through the project, I gained a deeper understanding of database design principles, including normalization, indexing, and data modeling. I learned how to structure databases efficiently to ensure data integrity and optimize performance.

Project Management Skills:

In addition to technical skills, the project helped me develop project management skills such as task prioritization, time management, and communication. I learned how to break down complex tasks into manageable milestones, collaborate with team members effectively, and adapt to changing requirements throughout the project lifecycle.

In summary, the database management project provided valuable hands-on experience and learning opportunities across various aspects of database design, optimization, security, project management, and continuous learning. It helped me develop practical skills and insights that are applicable not only to database management but also to broader areas of information technology and software development.

IX. Challenges Faced

Complexity

Addressing the complexity of managing diverse Eshop data effectively.

Integration

Implementing seamless integration of various data sources within a Eshop.

Scalability

Scaling the database architecture to support the evolving needs of the Eshop.

X. Conclusion

In conclusion, the E-shop presented in this project offers a comprehensive solution for airlines to efficiently manage flight reservations. By leveraging a robust database management system and implementing essential features, the ARS streamlines the reservation process, enhances customer satisfaction, and improves operational efficiency for airlines.

THANK U 😄