ONLINE RETAIL STORE

Group 50

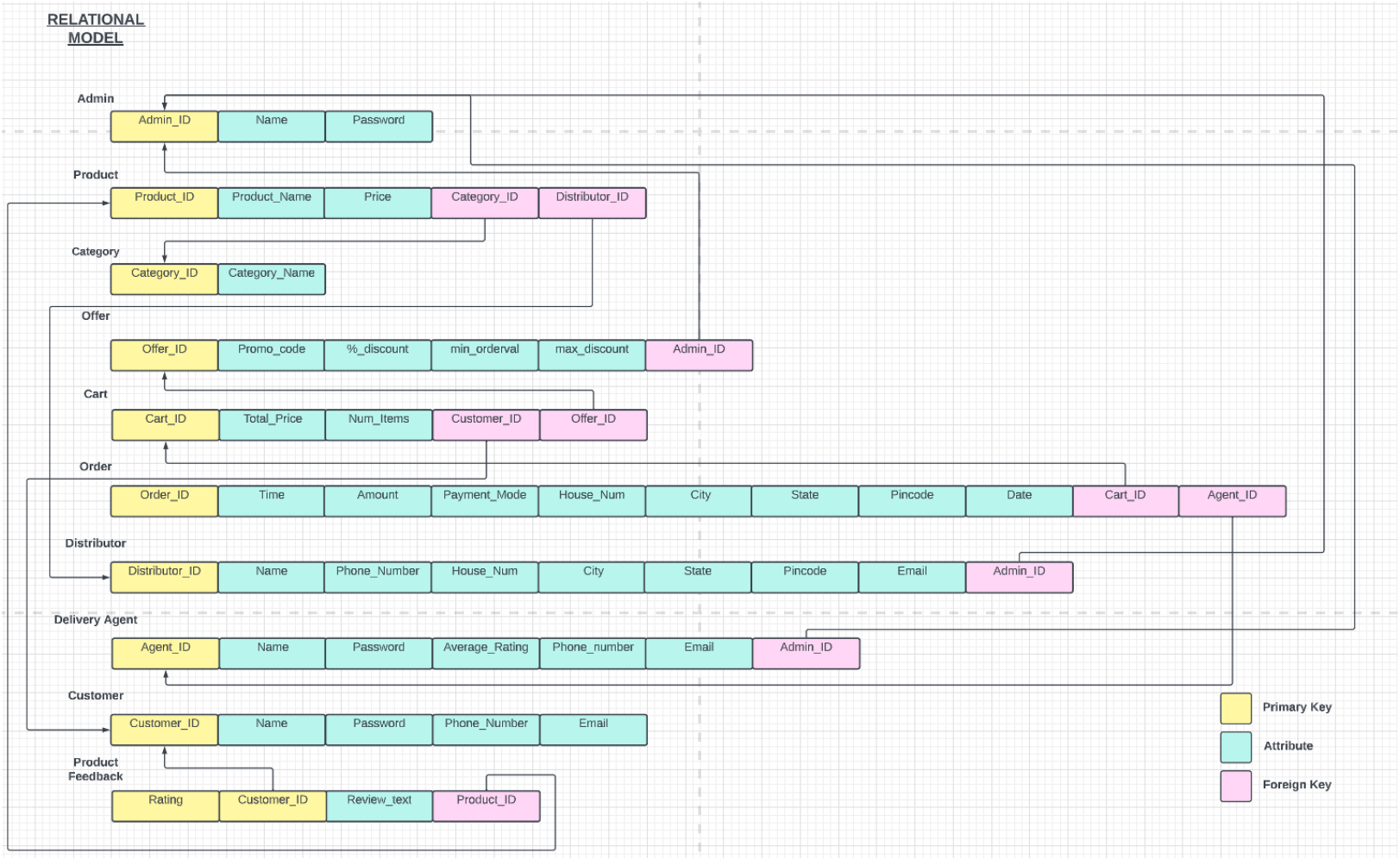
Samanyu Kamra 2021487

Shriya Verma 2021490

Our database has a total of 10 entities;

1. Customer
2. Distributor
3. Product
4. Category
5. Administration
6. Delivery Agent
7. Cart
8. Order
9. Product Feedback
10. Offers

RELATIONAL SCHEMA: (linked here)



QUERIES

A database query is a request for data from a database.

Query 1

Query1.sql

```sql
SELECT customer.customer_id , customer.full_name
FROM customer
WHERE customer.customer_id IN (SELECT cart.customer_id
                               FROM cart
                               WHERE cart.cart_id IN (SELECT _order.cart_id
                                                      FROM _order
                                                      WHERE _order.amount>=500 ));
```

Explanation 1:

Return Customer ID and Names of customers who placed orders worth 500 or more.

This nested query works on multiple tables (namely, customer, cart, and order) and returns the customer id and full name of the customer(s) who have placed an order worth more than or equal to Rs. 500.

Query 2

Query2.sql

```sql
-- Retrieve the number of products in each category:

SELECT category.category_name,(SELECT COUNT(*)
                               FROM product WHERE category_id = category.category_id) as Product_Count
FROM category;

```

Explanation 2:

This nested query calculates and returns the product count for each category.

Query 3

Query3.sql

```sql
-- Retrieve the sellers who have products in multiple categories:

SELECT distributor.full_name , COUNT(DISTINCT product.category_id) as Category_Count
FROM distributor
INNER JOIN product ON distributor.distributor_id = product.distributor_id
GROUP BY distributor.distributor_id
HAVING Category_Count>1
```

Explanation 3:

This query returns distributors who have products in more than one category.

This query works on two tables, namely, distributor and product. We have used an *inner-join* on the above-stated two tables, which combined records from both tables whenever there are matching values in distributor_id, a field common to both tables, distributor, and product.

Query 4

Query4.sql

```sql
-- Top 10 products.

SELECT product.product_name , COUNT(*) AS FEEDBACK_COUNT FROM product
JOIN product_feedback
ON product.product_id = product_feedback.product_id
GROUP BY product.product_id
ORDER BY FEEDBACK_COUNT DESC
LIMIT 10
```

Explanation 4:

This query returns the top 10 products with the most reviews.

This query returns the top 10 popular products by sorting them based on the number of reviews received. The product with the most number of feedbacks/reviews is assumed to be the most popular product. We have utilized *JOIN* and *ORDER BY (DESC) in this query*. We have set the limit to 10 as we want to view only the top 10 products.

Query 5

Query5.sql

```sql
-- Details of all orders at a particular city and payment mode

SELECT order_id, payment_mode
FROM _order
WHERE city = 'Delhi'
UNION
SELECT order_id, 'COD' AS payment_mode
FROM _order
WHERE _order.cart_id IN (
    SELECT cart.cart_id
    FROM cart , _order
    WHERE cart.total_price>0
)
```

Explanation 5:

This query filters orders by city and payment mode and returns the order_id and Payment mode.

In this query, we are selecting and displaying the order_id and payment mode of all orders placed in a particular city (Delhi) and having a specific mode of payment (Cash On Delivery). We have used *UNION* in this query.

Query 6

Query6.sql

```sql
-- Change delivery agents contact info

UPDATE Delivery
SET phone = '+91-XXXXX-XXXXX'
WHERE Delivery.avg_rating < 3.5 AND Delivery.agent_id IN (
    SELECT _order.date_
    FROM _order
    WHERE _order.date_ > '2023-01-01'
)
```

Explanation 6:

This query allows us to update the phone number of a delivery agent.

This is an *UPDATE* query, which allows us to change the delivery agent's phone number, we have applied the conditions of the avg_rating being greater than 3.5 and the order delivered by the said delivery agent being delivered on 2023-01-01 to select the agent whose phone number is to be updated.

Query 7

Query7.sql

```sql
-- Update price of a product in a particular category

UPDATE product
SET price = price * 1.1
WHERE product.category_id = (
    SELECT category.category_id
    FROM category
    WHERE category.category_name = 'Dairy'
)
```

Explanation 7:

This query allows us to update the prices of all products in a category at once.

This is an *UPDATE* query, which allows us to change the price of all products in a particular category. In this case, we have increased the prices of all dairy products by 1.1 times their original cost.

Query 8

Query8.sql

```sql
-- Work done by a particular admin

SELECT full_name,phone
FROM distributor
WHERE distributor.admin_id = 3
UNION
SELECT promocode,min_orderval
FROM offer
WHERE offer.admin_id = 3
UNION
SELECT full_name,avg_rating
FROM Delivery
WHERE Delivery.admin_id = 3

```

Explanation 8:

This Query shows all entities (distributors/offers/agents) added by a particular admin.

This query allows us to check all the additions (distributors, offers, and the delivery agents added) made by a particular admin (in this case, admin no.3)

Query 9

Query9.sql

```sql
-- Create a view named 'CustomerOrders' that shows the order_id, payment_mode, and amount for all
-- orders made by a cart with cart_id 'Cart001':

CREATE VIEW CustomerOrders AS
SELECT order_id, payment_mode, amount
FROM _order
WHERE _order.cart_id = 242;
```

Explanation 9:

Returns details of an order made by a particular cart_id.

This query creates a *VIEW* named Customer0rders which checks what orders are placed by a particular cart (cart_id = 242) and returns the order_id, payment_mode, and amount of the order.

Query 10

Query10.sql

```sql
-- Delete all orders with COD as payment mode
DELETE FROM _order
WHERE payment_mode = 'COD';
```

Explanation 10:

This is a DELETE query. It checks for all orders with payment_mode = 'COD' and deletes them.

Query 11

Query11.sql

```sql
-- Find details of a all carts by a customer.
SELECT *
FROM Customer
WHERE customer.customer_id = 20095
UNION
SELECT *
FROM cart
WHERE cart.customer_id = 20095
```

Explanation 11:

Displays all the details of the customer associated with a particular Cart (Cart_ID = 20095 in this case)

Query 12

Query12.sql

```sql
-- Retrieve the names of all customers who have ordered from the same delivery agent
-- more than once, and the delivery agent's average rating
SELECT customer.full_name, AVG(Delivery.avg_rating) AS avg_rating
FROM Customer
JOIN cart ON customer.customer_id = cart.customer_id
JOIN _order ON cart.cart_id = _order.cart_id
JOIN Delivery  ON Delivery.agent_id = _order.agent_id
WHERE Delivery.agent_id IN (
    SELECT _order.agent_id
    FROM _order
    JOIN Delivery ON _order.agent_id = Delivery.agent_id
    GROUP BY Delivery.agent_id
    HAVING COUNT(DISTINCT _order.cart_id) >= 2
)
GROUP BY customer.customer_id;
```

Explanation 12:

This query returns the names of all customers who have been allotted the same delivery agent twice or more times.

This query uses *JOINS.* We group the data by delivery.agent_id. We have joined four tables: Customer, Cart, _Order, and Delivery.

Query 13

Query13.sql

```sql
-- Customers who have not placed an order

SELECT customer.customer_id, customer.full_name
FROM Customer
WHERE customer_id NOT IN (
    SELECT DISTINCT customer_id
    FROM Cart
    JOIN _order ON cart.cart_id = _order.cart_id
)
```

Explanation 13:

Find all customers who have not placed an order.

This query uses NOT IN to find the carts that have not placed an order and, in turn, find the customer_id associated with the said carts to see all the customers who haven't placed an order yet.

Query 14

Query14.sql

```sql
-- Displays order placed by customer and the delivery address

SELECT
    CONCAT(_order.house_no, ', ', _order.city, ', ', _order.State, ' ', _order.pin_code) AS full_address,
    _order.order_id,
    CONCAT(customer.full_name, ' ') AS customer_name
FROM _order
JOIN cart ON _order.cart_id = cart.cart_id
JOIN customer ON cart.customer_id = customer.customer_id;
```

Explanation 14:

Display all orders, the names of the customers who placed them, and the delivery address.

This query uses CONCATENATE Function to combine all the columns corresponding to the address and displays the customer associated with the said order.

Query 15

Query15.sql

```sql
-- Apply discount to price of cart and update order price

UPDATE _order
JOIN cart ON _order.cart_id = cart.cart_id
JOIN offer ON cart.offer_id = offer.offer_id
SET _order.amount = cart.total_price * (1 - offer.percentagediscount/100)
WHERE _order.order_id = 15091;
```

Explanation 15:

Apply a discount on the cart price and reflect the updated price in the order price.

This query is an update query that uses multiplication and subtraction operations to update the price of an order after a discount.