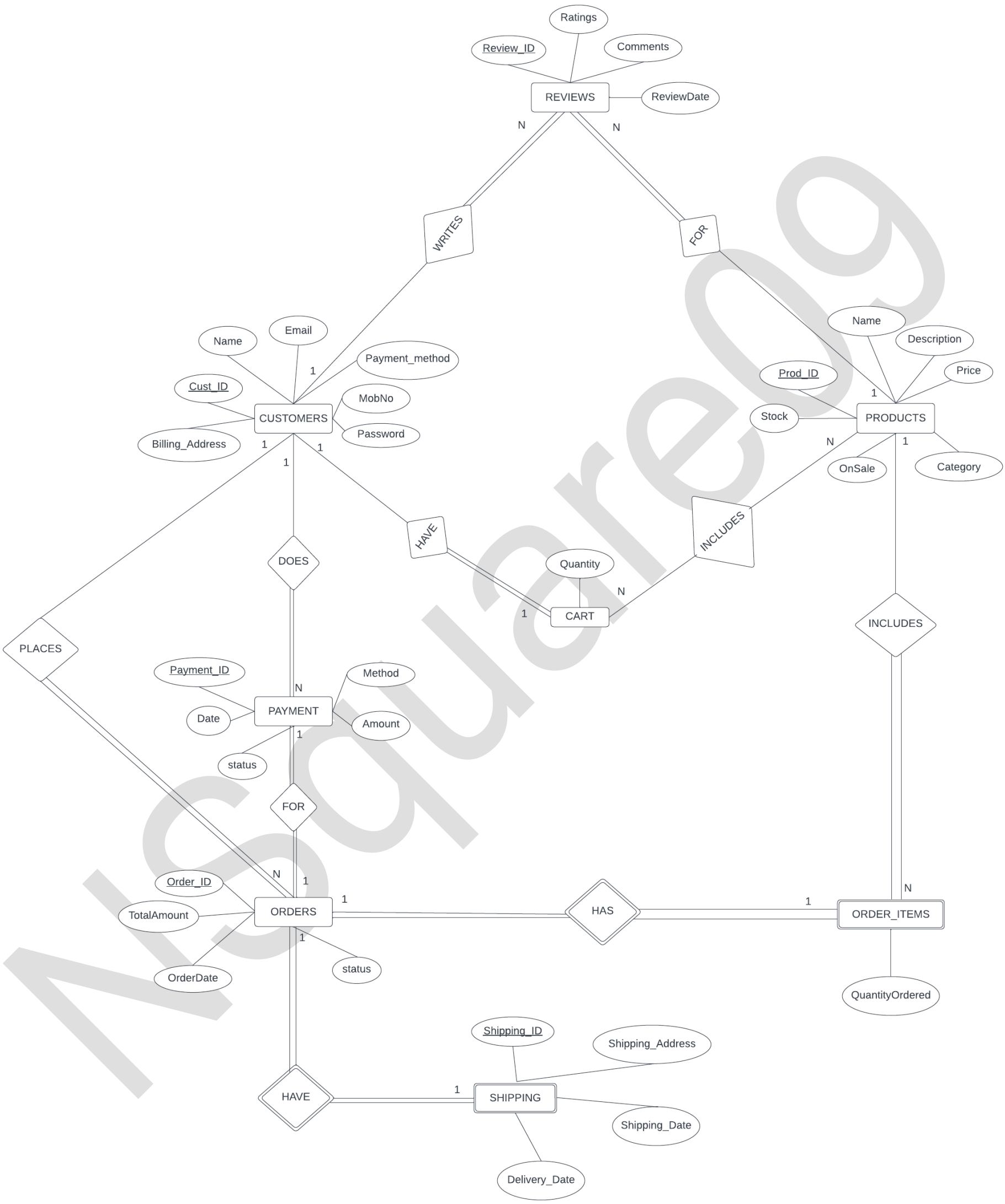
E-commerce Project

Below illustrated is the Entity-Relationship Diagram for an Ecommerce application where a user can browse for products, select products, add them to cart, make payment and several other functionalities that will be illustrated through the ER Diagram and the attached SQL Queries.

The diagram is drawn using LucidChart and follows the basic convention of E-R Diagrams followed in the course content.

ER DIAGRAM



Here we describe the various entities, their corresponding attributes and the the entity type.

ENTITY	ATTRIBUTE
CUSTOMERS	Name Email Payment_method phone Password Billing_Address Cust_ID (PK)
REVIEWS	ReviewID (PK) Ratings Comments ReviewDate
PRODUCTS	Product_ID (PK) ProdName Description Price Category OnSale Stock
CART	Quantity
PAYMENT	Payment_ID (PK) Date Status Method Amount
ORDERS	Order_ID (PK) TotalAmt OrderDate status
ORDER_ITEMS	Quantiy_Ordered
SHIPPING	Shipping_ID (PK) Shipping_Address Shipping_Date Delivery_Date

The E-R Diagram also illustrates the following relations:

1. WRITES: **Customers** can write **Reviews**. One customer can write multiple reviews but one review cannot be written by multiple customers (CARDINALITY: 1:N). All reviews must be written by some customers but not all customers will write reviews (CUSTOMERS: PARTIAL PARTICIPATION AND REVIEWS: TOTAL PARTICIPATION)
2. FOR: **Reviews** are made for the **Products**. One product can have multiple reviews but one review cannot be associated with multiple products (CARDINALITY 1:N). All reviews must be written for some products but not all products will have reviews (PRODUCTS: PARTIAL PARTICIPATION AND REVIEWS: TOTAL PARTICIPATION)
3. INCLUDES: **Cart** includes **Products**. One product can be part of multiple carts and one cart can have multiple products (CARDINALITY N:N). All products need not be part of a cart and all carts need not contain products (It is possible to have an empty cart). (CART: PARTIAL PARTICIPATION AND PRODUCTS: PARTIAL PARTICIPATION)
4. HAVE: **Customers** have a **Cart.** One customer will have only one cart and one cart will be uniquely associated with only one customer (CARDINALITY 1:1). All customers need not have a cart but all carts must be associated with some customer. (CUSTOMER: PARTIAL PARTICIPATION AND CART: TOTAL PARTICIPATION)
5. DOES: **Customer** does **Payment**. One customer can make multiple payments but one payment cannot be made by multiple customers. (CARDINALITY 1:N). Further every payment will be made by a customer but every customer need not make a payment (CUSTOMER: PARTIAL PARTICIPATION AND PAYMENT: TOTAL PARTICIPATION)
6. PLACES: **Customer** places **Orders**. One customer can make multiple orders but one order cannot be associated with multiple customers (CARDINALITY 1:N). All customers need not place an order but all orders must be associated with a customer. (CUSTOMER: PARTIAL PARTICIPATION AND ORDERS: TOTAL PARTICIPATION)
7. HAS: **Orders** has **Order_items** is a weak relationship. Not every order_item will be associated with an order and vice versa.
8. INCLUDES: **Order_Items** includes **Products**. One product can be part of multiple order items but one order item cannot be associated with multiple products. Further, every product need not be part of an order_item but every order_item must be associated with some products. (ORDER_ITEMS: TOTAL PARTICIPATION AND PRODUCTS: PARTIAL PARTICIPATION)
9. HAVE: **Orders** have **Shipping**. One order is associated with one shipping and one shipping is associated with a single order. All shipping must be associated with an order and all orders must be associated with a shipping. This is a weak relationship.
10. FOR: **Payment** is made for **Orders**. 1 payment is made for 1 order (1:1) and all orders are associated with a payment and all payments are associated with an order. Both show total participation.

RELATIONAL SCHEMA:

Here are the constraints mentioned below:

NOT NULL: This constraint is used to make sure that no NULL values are present in a column. To ensure that no customer is created without this mandatory information, the name, email, phone, and password columns in the Customers table, are all set to NOT NULL.

UNIQUE: This constraint is intended to guarantee that each value in a column is distinct. For instance, a primary key on the cust id column in the Customers table instantly generates a unique index. To further verify that each customer has a distinct email address, it is advised to add a unique index to the email column.

PRIMARY KEY: We utilize this constraint to locate a distinct record in a table. For instance, the Customers database uses the cust id column as its primary key to ensure that every customer has a distinct identification.

Relational Model:

- Customers(cust_id , name, email, phone, password, payment_method, billing address)
  PK= cust_id
- Products(prod_id, name, description, onsale, price, category, stock)
  PK= prod_id
- Cart(cust_id, prod_id, quantity)
  PK= (cust_id, prod_id)
  FK= cust_id refernces Customers(cust_id)
  FK= prod_id refrences Products(prod_id)
- Orders(order_id, cust_id, order_date, total_amount, status)
  PK=order_id
  FK= cust_id refernces Customers(cust_id)
- Shipping(shipping_id, Order_id, shipping_date, delivery_date, shipping_address)
  PK= shipping_id
  FK=(order_id) refernces Orders(order_id)
- Order_Items(order_id, prod_id, quantity_ordered)
  PK=(order_id, prod_id)
  FK=(prod_id) refrences Products(prod_id))
  FK=(order_id) refernces Orders(order_id)
- Payments(payment_id, order_id, payment_date, amount, method, status)
  PK=payment_id
  FK=order_id refernces Orders(order_id)
- Reviews(review_id, cust_id, prod_id, rating, comment, review_date)
  PK= review_id
  FK=cust_id references Customers(cust_id)
  FK=prod_id refernces Products(prod_id)

FUNCTIONAL DEPENDENCIES (BEFORE NORMALIZATION)

Customers : cust_id → name, email, phone, password, payment_method, billing address

Products : prod_id → name, description, onsale, price, category, stock

Cart : cust_id, prod_id → quantity

Orders: order_id → cust_id, order_date, total_amount, status

Shipping : shipping_id → Order_id, shipping_date, delivery_date, shipping_address

Order_Items : order_id, prod_id → quantity_ordered

Payments : payment_id → order_id, payment_date, amount, method, status

Reviews : review_id → cust_id, prod_id, rating, comment, review_date

NORMALIZATION (1NF, 2NF, 3NF)

First Normal Form (1NF):

1NF is already achieved as each column has atomic values and no repeating groups.

Second Normal Form (2NF):

To achieve 2NF, we need to ensure that each non-key column in the table is fully dependent on the primary key. In the given schema, we can see that the stock in the Product table has a partial dependency on the primary key, which could cause update anomalies. For instance, the stock attribute of the associated Product record needs to be updated whenever an order is placed. If one of the changes fails and the stock attribute is repeated across numerous entries, there could be some inconsistencies. So, we divided the Products table into two tables, Products and Stock, to solve this problem. Just the product attributes that are directly relevant to the product, such as prod id, prod name, price, and category id, are now present in the Products table. Prod id and stock characteristics are contained in the Stock table. By doing this, redundancy and update anomalies are eliminated.

To remove this partial dependency, we can split the Product table into two tables:

1. Products(prod_id, name, description, onsale, price, category)
   PK= prod_id
2. Stock(prod_id, stock)
   PK= prod_id
   FK= prod_id refernces Products(prod_id)

Also, we separated the Category to a new table and deleted the category field from the Products table. By doing this, the duplication that would result from having several products in the same category is avoided. This eliminates the redundancy that could occur if multiple products belonged to the same category.

Category(cat_id, cat_name)
PK: cat_id

Products (prod_id, name, description, onsale, price, cat_id)
PK: prod_id

FK: cat_id references Category(cat_id)

Third Normal Form (3 NF):

To normalize to 3NF, we need to identify the functional dependencies and eliminate transitive dependencies.

1. Customers (cust_id, name, email, phone, password, billing_address)
   - The primary key is cust_id. No further normalization is needed as all attributes are functionally dependent on the primary key.
2. Products (prod_id, name, description, onsale, price, cat_id)
   - The primary key is prod_id. No further normalization is needed as all attributes are functionally dependent on the primary key.
3. Stock (prod_id, stock)
   - The primary key is prod_id. No further normalization is needed as all attributes are functionally dependent on the primary key.

4. Cart (cust_id, prod_id, quantity)
    - The primary key is (cust_id, prod_id). No further normalization is needed as all attributes are functionally dependent on the primary key.
5. Orders (order_id, cust_id, order_date, total_amount, status)
    - The primary key is order_id.
    - The following functional dependencies exist:
        - order_id → cust_id, order_date, total_amount, status
        - cust_id → billing_address
    - To eliminate the transitive dependency of cust_id → billing_address, we create the following new tables

    Orders (order_id, cust_id, order_date, total_amount, status)

    Customer_Address (billing_address, cust_id(FK))
6. Shipping (shipping_id, order_id, shipping_date, shipping_address, delivery_date)
    - The primary key is shipping_id. No further normalization is needed as all attributes are functionally dependent on the primary key.
7. Order_Items (order_id, prod_id, quantity)
    - The primary key is (order_id, prod_id). No further normalization is needed as all attributes are functionally dependent on the primary key.
8. Payments (payment_id, order_id, payment_date, amount, status)
    - The primary key is payment_id. No further normalization is needed as all attributes are functionally dependent on the primary key.
9. Reviews (review_id, cust_id, prod_id, rating, comment, review_date)
    - The primary key is review_id. No further normalization is needed as all attributes are functionally dependent on the primary key.
10. Category(cat_id, cat_name)
    - The primary key is cat_id. No further normalization is needed as all attributes are functionally dependent on the primary key.

The 3NF normalized E-commerce database is as follows:

- Customers(cust_id, name, email, phone, password, payment_method)
  PK: cust_id
- Customer_Address(billing_address, cust_id)
  FK:cust_id references Customers(cust_id)
- Category(cat_id, cat_name)
  PK: cat_id
- Products(prod_id, name, description, onsale, price, cat_id)
  PK: prod_id
  FK: cat_id references Category(cat_id)
- Stock(prod_id, stock)
  PK: prod_id
  FK: prod_id refernces Products
- Orders(order_id, cust_id, order_date, total_amount, status)
  PK: order_id
  FK: cust_id references Customers(cust_id)

- Order_Items(order_id, prod_id, quantity_ordered)
  PK: order_id, prod_id
  FK: order_id references Orders(order_id)
  prod_id references Products(prod_id)
- Payments(payment_id, order_id, payment_date, amount, method, status)
  PK: payment_id
  FK: order_id references Orders(order_id)
- Reviews(review_id, cust_id, prod_id, rating, comment, review_date)
  PK: review_id
  FK: cust_id references Customers(cust_id)
  prod_id references Products(prod_id)
- Cart(cust_id, prod_id, quantity)
  PK: cust_id, prod_id
  FK: cust_id references Customers(cust_id)
  prod_id references Products(prod_id)
- Shipping(shipping_id, order_id, shipping_date, delivery_date, shipping_address)
  PK: shipping_id
  FK: order_id references Orders(order_id)

FUNCTIONAL DEPENDENCIES (AFTER NORMALIZATION)

Customers:

cust_id → name, email, phone, password, payment_method

Customer_Address:

cust_id → billing_address

Category:

cat_id → cat_name

Products:

prod_id → name, description, onsale, price, cat_id
cat_id → cat_name

Stock:

prod_id → stock

Cart:

(cust_id, prod_id) → quantity

Orders:

order_id → cust_id, order_date, total_amount, status

Shipping:

shipping_id → order_id, shipping_date, delivery_date, shipping_address

Order_Items:

(order_id, prod_id) → quantity_ordered

Payments:

payment_id → order_id, payment_date, amount, method, status

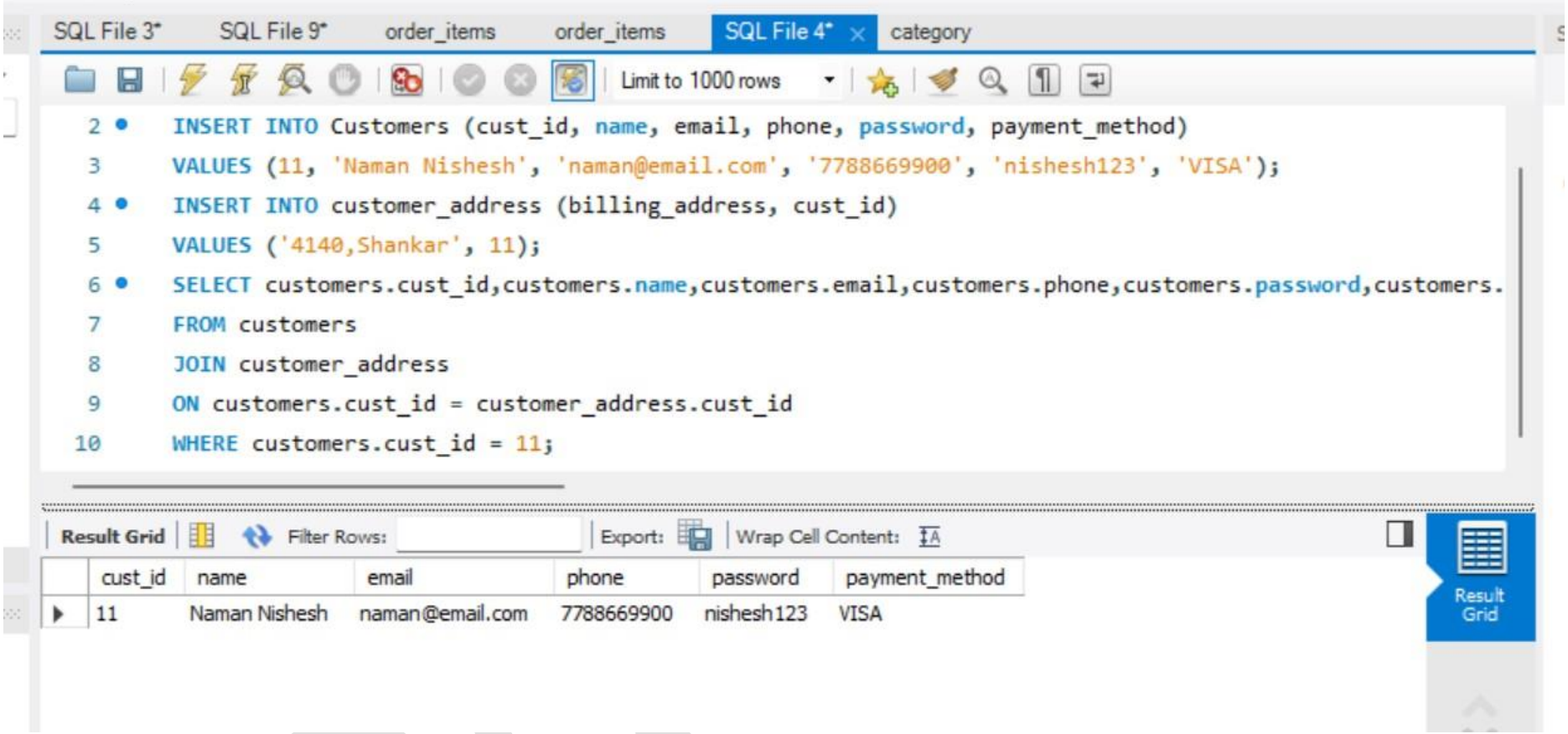
Reviews:

review_id → cust_id, prod_id, rating, comment, review_date

SQL QUERIES FOR USER REQUESTS:

Below we have mentioned 15 example queries which illustrates the depth and extent of the various functionalities that are supported in our e-commerce application:

1. **-to register a new customer on the platform.**



2. **- Write a query to show all products in a specific category: clothing the cat_id of clothing is 2**

SQL File 3* SQL File 9* order_items order_items SQL File 4* category SQL File 5* ×

Limit to 1000 rows

```
1  -- Write a query to show all products in a specific category: clothing
2  -- the cat_id of clothing is 2
3  SELECT *
4  FROM products
5  WHERE cat_id = 2;
6
```

Execute the selected portion of the script or everything, if there is no selection

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

prod_id	name	description	onsale	price	cat_id
1	Puma Jacket	Black Puma Jacket SIZE:M	no	2000.00	2
6	Levis Shirt	Stylish Blue and White Shirt SIZE: M	yes	2000.00	2
10	US Polo T Shirt	Comfortable and stylish red T shirt SIZE:M	yes	1500.00	2
NULL	NULL	NULL	NULL	NULL	NULL

Result Grid

products 2 × Apply Revert

3. **- Write a query to recommend similar products to the one a customer just purchased:**
   -- Assuming that the product the customer just purchased has a prod_id of 1

SQL File 3* SQL File 9* order_items order_items SQL File 4* category SQL File 5* SQL File 6* ×

Limit to 1000 rows

```
1  -- Write a query to recommend similar products to the one a customer just purchased:
2  -- Assuming that the product the customer just purchased has a prod_id of 1
3  SELECT *
4  FROM Products
5  WHERE cat_id = (SELECT cat_id FROM Products WHERE prod_id = 1) AND prod_id <> 1;
6
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

prod_id	name	description	onsale	price	cat_id
6	Levis Shirt	Stylish Blue and White Shirt SIZE: M	yes	2000.00	2
10	US Polo T Shirt	Comfortable and stylish red T shirt SIZE:M	yes	1500.00	2
NULL	NULL	NULL	NULL	NULL	NULL

Result Grid

4. **- Write a query to show status of any order:**
   -- Assuming that the order_id of the order in question is 1.

SQL File 3* | SQL File 9* | order_items | order_items | SQL File 4* | category | SQL File 5* | SQL File 6* | SQL File 7*

Limit to 1000 rows

```
1   -- Write a query to show status of any order:
2   SELECT status
3   FROM Orders
4   WHERE order_id = 1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

status
Processing

Result Grid

5. **- Write a query to update the billing address of a customer:**
   -- Assuming that the customer's cust_id is 1 and their new billing address is '121, BITS,Pilani'.

QL File 9* | order_items | order_items | SQL File 4* | category | SQL File 5* | SQL File 6* | SQL File 7* | SQL File 8*

Limit to 1000 rows

```
1   -- Write a query to update the billing address of a customer:
2   -- Assuming that the customer's cust_id is 1 and their new billing address is '121, BITS,Pilani'.
3   UPDATE Customer_Address
4   SET billing_address = '121, BITS,Pilani'
5   WHERE cust_id = 1;
6
7   SELECT billing_address
8   FROM customer_address
9   WHERE cust_id = 1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

billing_address
121, BITS,Pilani

Result Grid

mer_address 1 | Read Only

6. **Write a query to show details of a product:**
   Assuming that the product_id of the product in question is 1

AND

Write a query to see the order history of customers.

_items | order_items | SQL File 4* | category | SQL File 5* | SQL File 6* | SQL File 7* | SQL File 8* | SQL File 9*

Limit to 1000 rows

```sql
-- Write a query to show details of a product:
-- Assuming that the product_id of the product in question is 1
SELECT Products.prod_id, Products.name, Products.description, Products.price, Products.onsale, Category
FROM Products
INNER JOIN Category ON Products.cat_id = Category.cat_id
WHERE Products.prod_id = 1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

prod_id	name	description	price	onsale	cat_name
1	Puma Jacket	Black Puma Jacket SIZE:M	2000.00	no	Clothing

Result Grid

Result 3 | Read Only

SQL File 8* | SQL File 9* | SQL File 10* | SQL File 11* | SQL File 12* | SQL File 13 | SQL File 14* | SQL File 15*

Limit to 1000 rows

```sql
-- Write a query to see the order history of customer:
SELECT Orders.order_id, Orders.order_date, Orders.total_amount, Orders.status
FROM Orders
WHERE Orders.cust_id = 1;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

order_id	order_date	total_amount	status
1	2023-04-10	100.00	completed
NULL	NULL	NULL	NULL

Result Grid

Orders 1 | Apply | Revert

7. **- To suggest products that are currently on sale:**

der_items | SQL File 4* | category | SQL File 5* | SQL File 6* | SQL File 7* | SQL File 8* | SQL File 9* | SQL File 10*

Limit to 1000 rows

```
1   -- To suggest products that are currently on sale:
2   SELECT Products.prod_id, Products.name, Products.description, Products.price
3   FROM Products
4   WHERE onsale = 'yes';
5
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

prod_id	name	description	price
2	Tropicana Juice	Healthy orange juice 500ML	200.00
4	Garnier Men Face Wash	Face wash for acne prone skin	150.00
6	Levis Shirt	Stylish Blue and White Shirt SIZE: M	2000.00
8	BTWIN Bottle	500ML Capacity Transparent bottle	150.00
10	US Polo T Shirt	Comfortable and stylish red T shirt SIZE:M	1500.00
NULL	NULL	NULL	NULL

Result Grid | Form Editor | Field Types

Products 1 | Apply | Revert

8. **- To add a product to my cart:**

IL File 4* | category | SQL File 5* | SQL File 6* | SQL File 7* | SQL File 8* | SQL File 9* | SQL File 10* | SQL File 11*

Limit to 1000 rows

```
1   -- To add a product to my cart:
2   INSERT INTO Cart (cust_id, prod_id, quantity)
3   VALUES (1, 6, 2);
4
5   SELECT *
6   FROM cart
7   WHERE cust_id = 1;
8
```

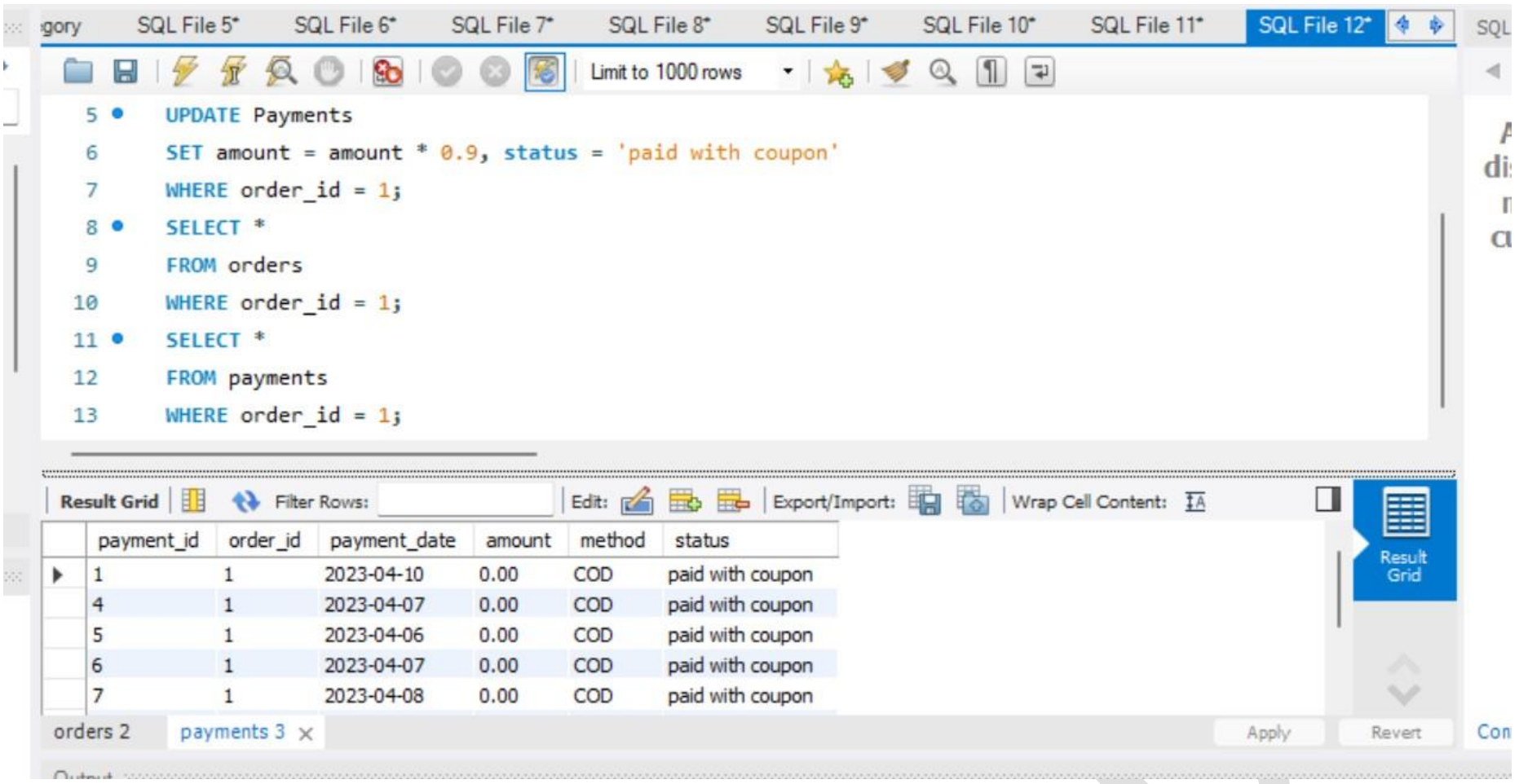
Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

cust_id	prod_id	quantity
1	3	3
1	4	2
1	5	1
1	6	2
NULL	NULL	NULL

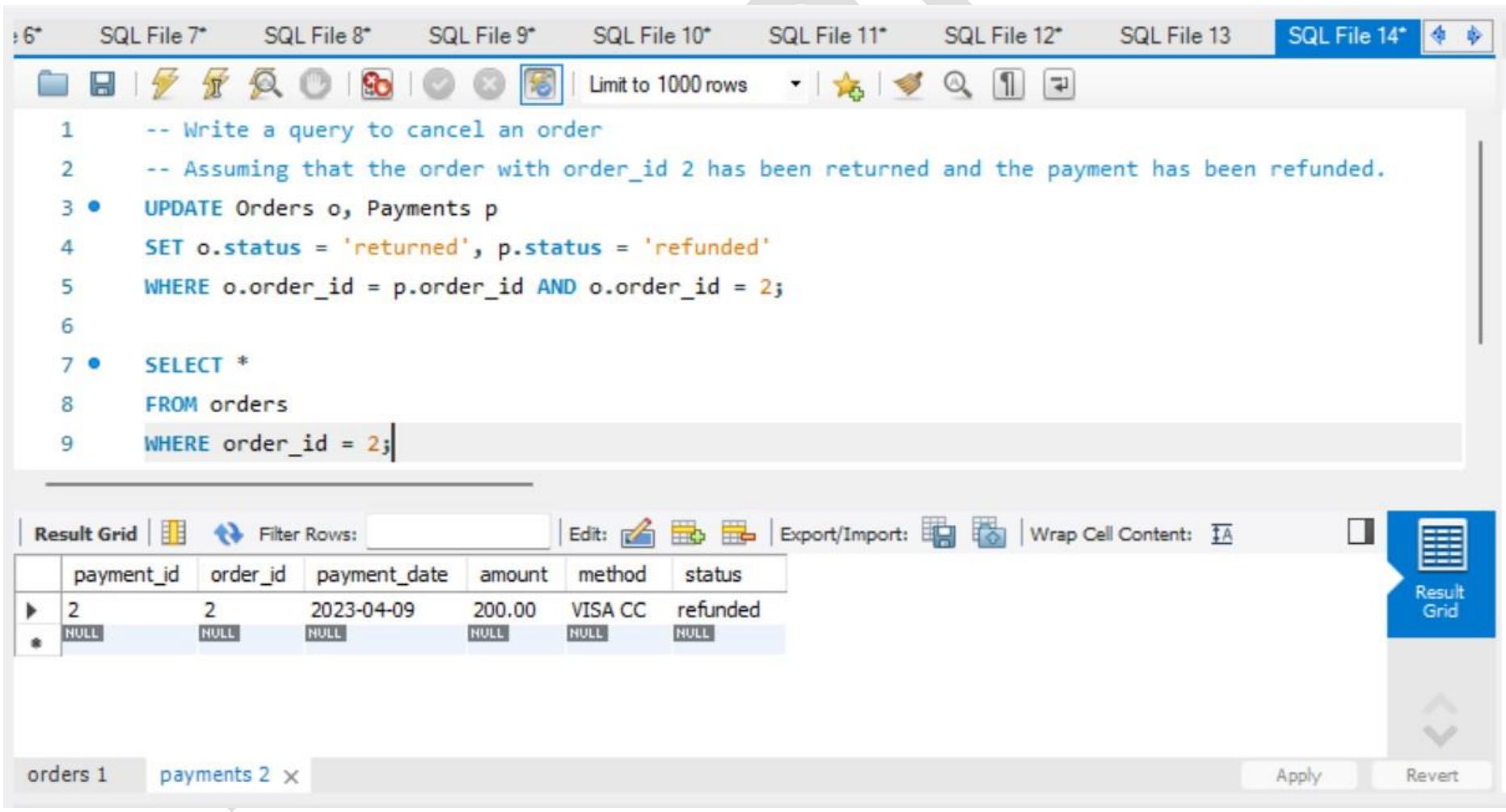
Result Grid

cart 1 | Apply | Revert

9. **- Write a query to place an order using a coupon code:**
   -- Assuming that the coupon code provides a 10% discount on the order with order_id 1



10. **Write a query to cancel an order. Assuming that the order with order_id 2 has been returned and the payment has been refunded.**



11. **- Write a query to see the order history of the customer:**

SQL File 8* SQL File 9* SQL File 10* SQL File 11* SQL File 12* SQL File 13 SQL File 14* SQL File 15*

Limit to 1000 rows

```sql
-- Write a query to see the order history of customer:
SELECT Orders.order_id, Orders.order_date, Orders.total_amount, Orders.status
FROM Orders
WHERE Orders.cust_id = 1;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

order_id	order_date	total_amount	status
1	2023-04-10	100.00	completed
NULL	NULL	NULL	NULL

Result Grid

Orders 1 × Apply Revert

12. - **Write a query to show products that are currently trending on the platform.**

SQL File 9* SQL File 10* SQL File 11* SQL File 12* SQL File 13 SQL File 14* SQL File 15* SQL File 16*

Limit to 1000 rows

```sql
-- Write a query to show products that are currently trending on the platform.
SELECT p.prod_id, p.name, COUNT(*) as num_orders
FROM Products p
JOIN Order_Items oi ON p.prod_id = oi.prod_id
JOIN Orders o ON oi.order_id = o.order_id
WHERE o.order_date >= DATE_SUB(NOW(), INTERVAL 1 MONTH)
GROUP BY p.prod_id
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

prod_id	name	num_orders
1	Puma Jacket	3
3	I-phone case	2
2	Tropicana Juice	2
4	Garnier Men Face Wash	2
5	Biotique Green Apple Shampoo	1

Result Grid | Form Editor

Result 1 × Read Only

Output

13. - **Write a query to update the quantity of a product in the cart.**

SQL File 10* SQL File 11* SQL File 12* SQL File 13 SQL File 14* SQL File 15* SQL File 16* SQL File 17*

Limit to 1000 rows

```sql
-- Write a query to update the quantity of a product in the cart.
UPDATE Cart
SET quantity = 5
WHERE cust_id = 1 AND prod_id = 5;

SELECT *
FROM cart
WHERE cust_id = 1;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

cust_id	prod_id	quantity
1	3	3
1	4	2
1	5	5
1	6	2
NULL	NULL	NULL

Result Grid

cart 1 × Apply Revert

14. **- Retrieve the average rating for a specific product:**

SQL File 11* SQL File 12* SQL File 13 SQL File 14* SQL File 15* SQL File 16* SQL File 17* SQL File 18*

Limit to 1000 rows

```sql
-- Retrieve the average rating for a specific product:
SELECT AVG(rating) FROM Reviews WHERE prod_id = 2;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

AVG(rating)
2.6667

Result Grid

15. **- Retrieve all reviews for a specific product:**

SQL File 12* SQL File 13 SQL File 14* SQL File 15* SQL File 16* SQL File 17* SQL File 18* SQL File 19*

Limit to 1000 rows

```
1   -- Retrieve all reviews for a specific product:
2 • SELECT * FROM Reviews WHERE prod_id = 2;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

review_id	cust_id	prod_id	rating	comment	review_date
2	2	2	3	Not a good product	2023-04-09
4	1	2	2	Loved it!	2023-04-07
9	3	2	3	Loved it!	2023-04-02
NULL	NULL	NULL	NULL	NULL	NULL

Result Grid

Reviews 1 ×

Apply Revert