E COMMERCE ORDER PROCESS USING SQL JOINS

A PROJECT TO REPLICATE WALMART'S ORDER FILLING SYSTEM

JUNE 20, 2024

UNDERSTANDING SCHEMA

Schemas are fundamental to learning Structured Query Language (SQL) as they form the foundation of your database, providing critical information about the structure of tables. A database is essentially a collection of tables, each defined by its columns and the data they hold. Key concepts to understand within schemas include primary keys and foreign keys. A primary key uniquely identifies each record within a table, while a foreign key is a field in one table that refers to the primary key in another table, establishing a relationship between the two.

To illustrate the concept of schemas, I've created a model inspired by Amazon's order fulfillment system, incorporating elements of employee tracking. The project includes four databases: HR, which contains employee information; STORE, which manages customer data and their orders; INVOICING, which handles client details and outstanding payments; and Inventory, which monitors product stock levels. This schema design provides a clear example of how primary and foreign keys function within a database structure.

.

SCHEMA FOR SQL_STORE

TABLE: CUSTOMERS	DATA TYPE
Customer ID	INT (11) – **PK**
First name	VARCHAR (50)
Last name	VARCHAR (50)
Birth date	DATE
Phone	VARCHAR (50)
Address	VARCHAR (50)
City	VARCHAR (50)
State	CHAR (2)
Points	INT (11)

TABLE: ORDERS	DATA TYPE
Order ID	INT (11) – **PK**
Customer ID	INT (11) – **FK FROM CUSTOMERS TABLE**
Order date	DATE
Status	TINYINT (4) – **FK FROM ORDER STATUSES**
Comments	CHAR (2000)
Shipped date	DATE
Shipper ID	SMALL INT (6) – **FK FROM SHIPPERS**

TABLE: PRODUCTS	DATA TYPE
Product ID	INT (11) – **PK**
Name	VARCHAR (50)
Quantity in stock	INT (11)
Unit Price	DECIMAL (4,2)

TABLE: SHIPPERS	DATA TYPE
Shipper ID	VARCHAR (50) – **PK**
Name	VARCHAR (50)

TABLE: ORDER STATUSES	DATA TYPE
Order Status ID	TINYINT (4) – **PK**
Name	VARCHAR (50)

TABLE: ORDER ITEMS	DATA TYPE
Order ID	INT (11) – **CPK – FK FROM ORDERS**
Product ID	INT (11) – **CPK – FK FROM PRODUCTS**
Quantity	INT (11) – **FK FROM PRODUCTS**
Unit Price	DECIMAL (4,2) – **FK FROM PRODUCTS**

DATABASE SQL STORE

The first table created is the 'customers' table, which includes various details about the customers, such as their name, address, and phone number, with a unique identifier called 'customer_id' serving as the primary key. Similar structures are followed for other tables, each with its own unique identifier: for instance, the 'shippers' table uses 'shipper_id,' the 'products' table uses 'product_id,' and the 'order_statuses' table uses 'order_status_id.' These tables store specific details about the shippers, the products purchased, and the status of shipments, respectively. Collectively, these four tables provide comprehensive information about customers, products, shippers, and order statuses.

To connect these tables and streamline the process of retrieving order information, we need to establish relationships between them. This is done by creating an 'orders' table, which has an 'order_id' as its primary key. The 'orders' table also includes the 'customer_id' from the 'customers' table, assigned as a foreign key. Similarly, the 'shipper_id' from the 'shippers' table and the 'order_status_id' from the 'order_statuses' table are also added to the 'orders' table as foreign keys.

To further detail the order information, columns for 'order_date' and 'shipped_date' are included in the 'orders' table. Unlike 'order_id,' these date fields are not primary keys since multiple orders may share the same date. Additionally, a 'comments' column can be added to capture any special notes related to the customer's order.

Finally, to track the specific items ordered, a new table is created, which uses a composite primary key consisting of 'product_id' and 'order_id,' both of which are foreign keys referencing the 'products' and 'orders' tables, respectively. The use of a composite primary key is necessary because the same product can appear in multiple orders, but each 'order_id' is unique. Additional columns for 'quantity' and 'unit_price' are included, referencing data from the 'products' table, to complete the details of the ordered items.

DATABASE SQL INVENTORY

This database has been created to manage the inventory by assigning the name of the product, product ID, the quantity in stock and the price of the product as an individual unit to an individual table named products.

SCHEMA FOR TABLE PRODUCTS

TABLE PRODUCTS	**DATA TYPE**
Product ID	INT (11)
Name	VARCHAR (50)
Quantity in stock	INT (11)
Unit Price	DECIMAL (4,2)

SCHEMA FOR DATABASE_INVOICING

TABLE: CLIENTS	DATA TYPE
CLIENT ID	INT (11) **PK**
NAME	VARCHAR (50)
ADDRESS	VARCHAR (50)
CITY	VARCHAR (50)
STATE	CHAR (2)
PHONE	VARCHAR (50)

TABLE: INVOICES	DATA TYPE
INVOICE ID	INT (11) - **PK**
NUMBER	VARCHAR (50)
CLIENT ID	INT (11) – **FK FROM TABLE CLIENTS**
INVOICE TOTAL	DECIMAL (9,2)
PAYMENT TOTAL	DECIMAL (9,2)
INVOICE DATE	DATE
DUE DATE	DATE
PAYMENT DATE	DATE

TABLE: PAYMENTS	DATA TYPE
PAYMENT ID	INT (11) - **PK**
CLIENT ID	INT (11) – **FK FROM TABLE CLIENTS**
INVOICE ID	INT (11) – **FK FROM TABLE INVOICES**
DATE	DATE
AMOUNT	DECIMAL (9,2)
PAYMENT METHOD	TINYINT (4) – **FK FROM TABLE PAYMENT METHODS**

TABLE: PAYMENT METHODS	DATA TYPE
PAYMENT METHOD ID	TINYINT (4) – **PK**
NAME	VARCHAR (50)

DATABASE INVOICING

The database Invoicing includes four tables named clients, payments, payment methods invoices.

The Table client includes the details about clients starting from the Client ID, the name, address, and their phone number and in this table, we have assigned client ID as the primary key for the table.

The Table payment method includes Payment method ID and the payment method name which the client has used to identify the mode of payment. To give a unique Identity in this table Payment Method ID has been assigned as the primary key.

The Table Invoices has been created to investigate the total Invoices created for the clients and their payment details along with the invoice ID which will be the unique Identifier in the table. Apart from this the date at which the invoice is being generated, the due date and payment date are also included in this table.

The final table is the table Payments through which we get to know which client has made the payment and which payment method is being used. Date of payment and amount will also be reflected in this table with Payment ID being the primary key in the table, whereas Client ID, Invoice ID, and Payment Method have been taken as foreign keys from the tables: clients, payment methods, and invoices respectively.

INSIGHTS

To add value to this project, I have calculated some reward for the most valued customers of the store.

```
SELECT *
FROM sql_store.customers;
```

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

customer_id	first_name	last_name	birth_date	phone	address	city	state	points
1	John	Doe	1990-01-01	555-1234	123 Main St	New York	NY	1000
2	Jane	Smith	1985-05-15	555-5678	456 Elm St	Los Angeles	CA	1500
3	Bob	Johnson	1978-11-30	555-9876	789 Oak St	Chicago	IL	750
4	David	Brown	1980-09-22	555-6789	654 Pine St	Seattle	WA	800
5	Emily	Davis	1995-11-30	555-1357	987 Cedar Ln	Austin	TX	950
6	Michael	Garcia	1975-06-25	555-2468	147 Birch St	Boston	MA	1100
7	Sophia	Martinez	1988-12-12	555-3698	258 Spruce St	Miami	FL	700
8	James	Miller	1991-01-15	555-4820	369 Willow Dr	Denver	CO	1300
9	Olivia	Wilson	1993-04-05	555-5912	741 Oak Dr	Portland	OR	850
10	William	Moore	1986-07-08	555-7389	852 Elm Dr	Atlanta	GA	950

Through this table we get to know that there are customers with some loyalty points that they have accumulated. Now the store wants to thank the customers and reward them with some shopping bonuses. The customers will be rewarded according to their category.

```sql
select *,case when points > 1000 then 'gold'
when points > 500 then 'silver'
else 'bronze'
end as rewards
from sql_store.customers;
```

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

customer_id	first_name	last_name	birth_date	phone	address	city	state	points	rewards
1	John	Doe	1990-01-01	555-1234	123 Main St	New York	NY	1000	silver
2	Jane	Smith	1985-05-15	555-5678	456 Elm St	Los Angeles	CA	1500	gold
3	Bob	Johnson	1978-11-30	555-9876	789 Oak St	Chicago	IL	750	silver
4	David	Brown	1980-09-22	555-6789	654 Pine St	Seattle	WA	800	silver
5	Emily	Davis	1995-11-30	555-1357	987 Cedar Ln	Austin	TX	950	silver
6	Michael	Garcia	1975-06-25	555-2468	147 Birch St	Boston	MA	1100	gold
7	Sophia	Martinez	1988-12-12	555-3698	258 Spruce St	Miami	FL	700	silver
8	James	Miller	1991-01-15	555-4820	369 Willow Dr	Denver	CO	1300	gold
9	Olivia	Wilson	1993-04-05	555-5912	741 Oak Dr	Portland	OR	850	silver
10	William	Moore	1986-07-08	555-7389	852 Elm Dr	Atlanta	GA	950	silver
11	Isabella	Taylor	1990-10-21	555-8495	963 Maple St	Phoenix	AZ	800	silver
12	Daniel	Anderson	1979-02-14	555-9526	654 Oak Ave	Philadelphia	PA	1000	silver

For the store to reward the customers, the customers are divided into three categories, gold, bronze, and silver. The customers in the gold category will get a $50 shopping bonus, the silver category customers will receive a $30 shopping bonus and the customers in the bronze category will get a $10 shopping bonus at the end of every quarter.

This process can be used as a marketing campaign to attract more customers to the store. The customer accumulating highest points at the end of the quarter will get an extra $50 shopping bonus as a reward for being the most loyal customer.

```sql
SELECT *,
       CASE WHEN points = (SELECT MAX(points) FROM sql_store.customers) THEN 'loyal' ELSE 'not loyal' END AS loyalty_status,
       MAX(points) OVER () AS loyal_customer
FROM sql_store.customers
ORDER BY points DESC
LIMIT 1;
```

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

customer_id	first_name	last_name	birth_date	phone	address	city	state	points	loyalty_status	loyal_customer
2	Jane	Smith	1985-05-15	555-5678	456 Elm St	Los Angeles	CA	1500	loyal	1500

I had a customer call me on the store asking about their shipment status of the product, as they were not able to see whether their product has been shipped or not, they called on the store. Using joins, I told the customer about their shipment status.

```
SELECT orders.order_id,orders.order_date,customers.last_name,customers.first_name,orders.shipper_id,order_statuses.name AS status
FROM sql_store.orders
LEFT JOIN sql_store.customers ON sql_store.customers.customer_id = sql_store.orders.customer_id
LEFT JOIN sql_store.order_statuses ON sql_store.order_statuses.order_status_id = sql_store.orders.status
LIMIT 0, 1000;
```

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

order_id	order_date	last_name	first_name	shipper_id	status
1	2023-01-01	Doe	John	1	Shipped
2	2023-01-15	Smith	Jane	NULL	Processed
3	2023-02-01	Johnson	Bob	2	Delivered
4	2023-02-15	Brown	David	NULL	Pending
5	2023-03-01	Davis	Emily	3	Processed
6	2023-03-10	Garcia	Michael	1	Shipped
7	2023-03-20	Martinez	Sophia	NULL	Cancelled
8	2023-04-01	Miller	James	2	Delivered
9	2023-04-10	Wilson	Olivia	NULL	Pending
10	2023-04-15	Moore	William	3	Shipped

The table gives us the names of the customer, the date on which they ordered, and the status of the shipment and the shipper has been assigned a shipper Id for convenience. By using another left join in the queries, we can also get the details of who is shipping the order.

```sql
SELECT
    p.product_id,
    p.name AS product_name,
    p.quantity_in_stock,
    p.unit_price
FROM
    SQL_STORE.products p
WHERE
    p.quantity_in_stock < 500;
```

Result Grid

product_id	product_name	quantity_in_stock	unit_price
4	Apple Watch	80	399.99
5	Google Pixel	120	799.99
6	Dell XPS 13	40	1299.99
7	Sony Headphones	200	149.99
8	Kindle Paperwhite	150	129.99
9	iPad Air	70	599.99
10	Fitbit Charge	90	149.99
11	Microsoft Surface	30	1199.99
12	HP Envy	60	999.99
13	Oculus Quest	50	299.99
NULL	NULL	NULL	NULL

products 10

The query provided is designed to identify products in the store that have a low quantity in stock. Specifically, it lists all products where the stock is less than 500 units. This can help the store's inventory management team to focus on replenishing these items to avoid stockouts.

```
        SUM(oi.quantity * oi.unit_price) AS total_sales
    FROM
        SQL_STORE.orders o
    JOIN
        SQL_STORE.order_items oi ON o.order_id = oi.order_id
    GROUP BY
        month
    ORDER BY
        month;
```

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

month	total_sales
2023-01	2799.97
2023-02	1299.97
2023-03	2999.95
2023-04	3499.96

The query provided calculates the total sales for each month by summing up the revenue generated from the items sold in that month

```sql
SELECT i.invoice_id, i.number AS invoice_number, c.name AS client_name,i.invoice_total,SUM(p.amount) AS total_payments,
    CASE WHEN SUM(p.amount) >= i.invoice_total THEN 'Fully Paid'WHEN SUM(p.amount) > 0 AND SUM(p.amount) < i.invoice_total THEN 'Partially Paid'
        ELSE 'Unpaid'END AS payment_status
FROM invoicing.invoices i
LEFT JOIN invoicing.payments p ON i.invoice_id = p.invoice_id
LEFT JOIN invoicing.clients c ON i.client_id = c.client_id
GROUP BY i.invoice_id, i.number, c.name,i.invoice_total
ORDER BY i.invoice_id;
```

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

invoice_id	invoice_number	client_name	invoice_total	total_payments	payment_status
1	INV001	Acme Corp	1200.00	1200.00	Fully Paid
2	INV002	Acme Corp	800.00	400.00	Partially Paid
3	INV003	Global Industries	1500.00	1500.00	Fully Paid
4	INV004	Tech Solutions	600.00	600.00	Fully Paid
5	INV005	Retail Hub	900.00	450.00	Partially Paid

This query is useful for financial reporting and auditing purposes. It allows the business to track which invoices have been paid in full, partially paid, or have outstanding balances. By knowing the total payments for each invoice, the business can manage its cash flow better and follow up on unpaid invoices promptly.

As the store has five clients, details of every client have been recorded here about the recent payment they have made for the store. We can track the payment method, the name of the client and the amount that the client has paid through this table and the invoice number can be used to generate the details of the invoices.

```sql
SELECT c.name AS client_name, pm.name AS payment_method, p.date AS payment_date, p.amount AS payment_amount, i.number AS invoice_number
FROM invoicing.payments p
JOIN invoicing.clients c ON p.client_id = c.client_id
JOIN invoicing.invoices i ON p.invoice_id = i.invoice_id
JOIN invoicing.payment_methods pm ON p.payment_method = pm.payment_method_id
ORDER BY p.date DESC;
```

Result Grid

client_name	payment_method	payment_date	payment_amount	invoice_number
Retail Hub	Gift Card	2024-03-25	450.00	INV005
Tech Solutions	PayPal	2024-02-28	600.00	INV004
Global Industries	Debit Card	2024-02-15	1500.00	INV003
Acme Corp	Bank Transfer	2024-02-10	400.00	INV002
Acme Corp	Credit Card	2024-01-30	1200.00	INV001

Supposedly we want to track the details of invoice number 2.

```
select * from invoicing.invoices where invoice_id = 2
```

invoice_id	number	client_id	invoice_total	payment_total	invoice_date	due_date	payment_date
2	INV002	2	800.00	400.00	2024-01-20	2024-02-20	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

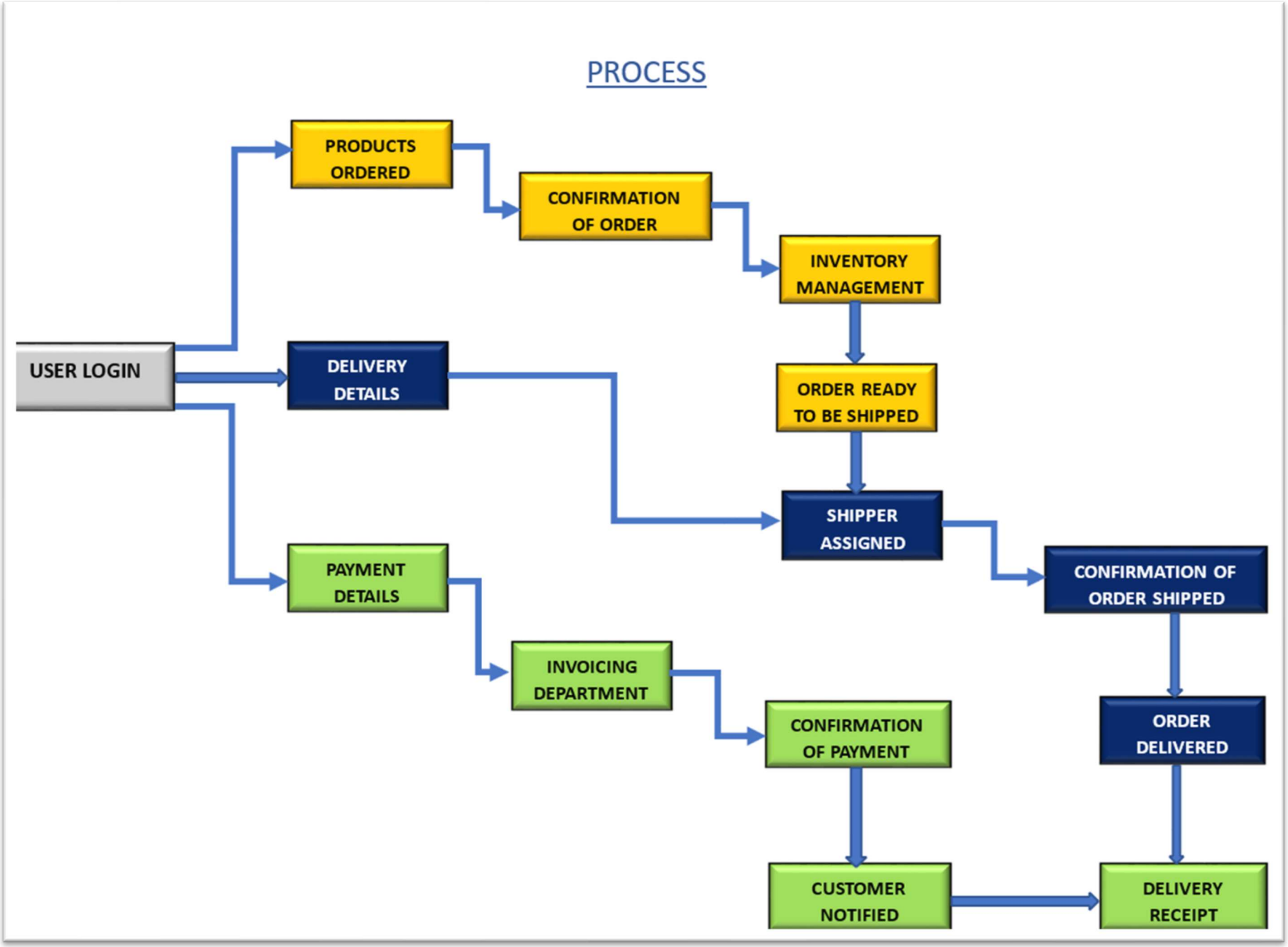
In this project we have seen the details of the customers, the reward the store can provide them, tracked the shipment of the product and the details of the client, the invoices that have been generated under their name, the payment method used by the clients and the amount that is to be paid.

Till now we have seen the processing of the order from the store till it reaches the customer. The last step left which goes along with this is the tracking of inventory of products left after the orders have been processed.

```sql
SELECT p.name AS product_name,p.quantity_in_stock AS quantity_in_stock,COALESCE(SUM(oi.quantity), 0)
 AS sold_items,p.quantity_in_stock - COALESCE(SUM(oi.quantity), 0) AS stock_left
FROM SQL_STORE.products p
LEFT JOIN SQL_STORE.order_items oi ON p.product_id = oi.product_id
GROUP BY p.product_id, p.name, p.quantity_in_stock
ORDER BY p.name;
```

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

product_name	quantity_in_stock	sold_items	stock_left
Apple Watch	80	2	78
Dell XPS 13	40	1	39
Fitbit Charge	90	1	89
Google Pixel	120	1	119
HP Envy	60	0	60
iPad Air	70	2	68
iPhone	100	1	99
Kindle Paperwhite	150	1	149
MacBook Pro	50	1	49
Microsoft Surface	30	0	30
Oculus Quest	50	0	50


PROCESS
PRODUCTS ORDERED
CONFIRMATION OF ORDER
INVENTORY MANAGEMENT
USER LOGIN
DELIVERY DETAILS
ORDER READY TO BE SHIPPED
SHIPPER ASSIGNED
PAYMENT DETAILS
CONFIRMATION OF ORDER SHIPPED
INVOICING DEPARTMENT
CONFIRMATION OF PAYMENT
ORDER DELIVERED
CUSTOMER NOTIFIED
DELIVERY RECEIPT