

1) Designed the completed database, schema and tables. Included all the constraints while creating the table.

```
create database bikestore;
```

```
Create schema bikestore.sales;
```

```
Create schema bikestore.production;
```

```
CREATE TABLE production.categories (  
    category_id INT IDENTITY (1, 1) PRIMARY KEY,  
    category_name VARCHAR (255) NOT NULL  
);
```

```
CREATE TABLE production.brands (  
    brand_id INT IDENTITY (1, 1) PRIMARY KEY,  
    brand_name VARCHAR (255) NOT NULL  
);
```

```
CREATE TABLE production.products (  
    product_id INT IDENTITY (1, 1) PRIMARY KEY,  
    product_name VARCHAR (255) NOT NULL,  
    brand_id INT NOT NULL,  
    category_id INT NOT NULL,  
    model_year SMALLINT NOT NULL,  
    list_price DECIMAL (10, 2) NOT NULL  
);
```

```
CREATE TABLE production.stocks (  
    store_id INT,  
    product_id INT,  
    quantity INT,  
    PRIMARY KEY (store_id, product_id)  
);
```

```
CREATE TABLE sales.stores (  
    store_id INT IDENTITY (1, 1) PRIMARY KEY,  
    store_name VARCHAR (255) NOT NULL,  
    phone VARCHAR (25),  
    email VARCHAR (255),  
    street VARCHAR (255),  
    city VARCHAR (255),  
    state VARCHAR (10),  
    zip_code VARCHAR (5)
```

);

```
CREATE TABLE sales.staffs (  
    staff_id INT PRIMARY KEY,  
    first_name VARCHAR (50) NOT NULL,  
    last_name VARCHAR (50) NOT NULL,  
    email VARCHAR (255) NOT NULL UNIQUE,  
    phone VARCHAR (25),  
    active INT NOT NULL,  
    store_id INT NOT NULL,  
    manager_id INT  
);
```

```
CREATE TABLE sales.orders (  
    order_id INT IDENTITY (1, 1) PRIMARY KEY,  
    customer_id INT,  
    order_status tinyint NOT NULL,  
    order_date varchar(20),  
    required_date varchar(20),  
    shipped_date varchar(20),  
    store_id INT NOT NULL,  
    staff_id INT NOT NULL  
);
```

```
CREATE TABLE order_items (  
    order_id INT,  
    item_id INT,  
    product_id INT NOT NULL,  
    quantity INT NOT NULL,  
    list_price DECIMAL (10, 2) NOT NULL,  
    discount DECIMAL (10, 2) NOT NULL,  
    PRIMARY KEY (order_id, item_id)  
);
```

```
CREATE TABLE sales.customers (  
    customer_id INT IDENTITY (1, 1) PRIMARY KEY,  
    first_name VARCHAR (255) NOT NULL,  
    last_name VARCHAR (255) NOT NULL,  
    phone VARCHAR (25),  
    email VARCHAR (255) NOT NULL,  
    street VARCHAR (255),  
    city VARCHAR (50),  
    state VARCHAR (25),  
    zip_code VARCHAR (5)
```

);

Home | Scaler Aca...

discovery+

(6) Home

DOCUME

< Worksheets

2023-08-24 7:03pm

Retail\_Table\_Sno

Databases

Worksheets

Pinned (1)

fct\_order\_analysis\_summary

Q All Objects

...

BIKESTORES

INFORMATION\_SCHEMA

PRODUCTION

Tables

BRANDS

CATEGORIES

PRODUCTS

STOCKS

PUBLIC

SALES

Tables

CUSTOMERS

ORDERS

ORDER\_ITEMS

STAFFS

STORES

DEMODATABASE

GB\_DEMO\_DB

MATALLION

NEW\_DEMO\_DB

BIKESTORES

4713

4714

4715

4716

4717

4718

4719

4720

4721

4722

4723

4724

4725

4726

4727

4728

4729

4730

4731

4732

4733

4734

4735

4736

4737

INS

INS

INS

INS

INS

INS

INS

INS

INS

INS

INS

INS

INS

INS

INS

INS

INS

INS

INS

INS

INS

INS

INS

INS

INS

Results

2) Implemented the foreign key constraint using alter table command for the required tables. Below are the changes

```
alter table staffs
add CONSTRAINT store_id
FOREIGN KEY (store_id) REFERENCES stores(store_id);
```

```
alter table staffs
add CONSTRAINT manager_id
FOREIGN KEY (manager_id) REFERENCES stores(store_id);
```

---

```
alter table order_items
add CONSTRAINT order_id
FOREIGN KEY (order_id) REFERENCES orders(order_id);
```

```
alter table order_items
add CONSTRAINT product_id
FOREIGNKEY(product_id)REFERENCES
BIKESTORE.PRODUCTION.PRODUCTS(product_id);
```

----

```
alter table orders
add CONSTRAINT customer_id
FOREIGN KEY (customer_id) REFERENCES customers(customer_id);
```

```
alter table orders
add CONSTRAINT store_id
FOREIGN KEY (store_id) REFERENCES stores(store_id);
```

```
alter table orders
add CONSTRAINT staff_id
FOREIGN KEY (staff_id) REFERENCES staffs(staff_id);
```

----

```
alter table stocks
add CONSTRAINT store_id
FOREIGN KEY (store_id) REFERENCES BIKESTORE.SALES.STORES(store_id);
```

```
alter table stocks
add CONSTRAINT product_id
FOREIGN KEY (product_id) REFERENCES products(product_id);
```

-----

```
alter table products
add CONSTRAINT category_id
FOREIGN KEY (category_id) REFERENCES categories(category_id);
```

```
alter table products
add CONSTRAINT brand_id
FOREIGN KEY (brand_id) REFERENCES brands(brand_id);
```

3) Does any of the table has missing or NULL value ? If yes which are those and what are their counts ?

```
select *
from table_name
where column_name IS NULL or column_name IS NULL ...
/* This is general query which will give us the record where column is null */
```

```
select COUNT(*) from customers where phone='NULL' --1267
```

The screenshot shows a database management tool interface. On the left, there is a sidebar with a tree view of the database structure. The 'SALES' database is expanded, showing tables: CUSTOMERS, ORDERS, ORDER\_ITEMS, STAFFS, and STORES. The 'CUSTOMERS' table is selected. The main area displays a SQL query: `select COUNT(*) from customers where phone='NULL' --1267`. Below the query, there is a 'Results' tab showing a table with one row and one column, 'COUNT(\*)', with the value 1,267. The table structure for 'CUSTOMERS' is also visible on the left, showing columns: CUSTOMER\_ID, FIRST\_NAME, LAST\_NAME, PHONE, EMAIL, STREET, CITY, STATE, and ZIP\_CODE.

CUSTOMER_ID	FIRST_NAME	LAST_NAME	PHONE	EMAIL	STREET	CITY	STATE	ZIP_CODE
1								

```
select count(*) FROM ORDERS WHERE SHIPPED_DATE ='NULL' --170
```

Worksheets 2023-08-30 12:58pm

Databases Worksheets

Pinned (1)

BRANDS

Q All Objects

SALES

Tables

- CUSTOMERS
- ORDERS
- ORDER\_ITEMS
- STAFFS
- STORES

SNOWFLAKE

SNOWFLAKE\_SAMPLE\_DATA

BIKESTORE.SALES Settings

```
1 select count(*) FROM ORDERS WHERE SHIPPED_DATE = 'NULL' --170
```

Results Chart

ORDERS 1.6K Rows

	COUNT(*)
1	170

ORDERS

	ORDER_ID	CUSTOMER_ID	ORDER_STATUS	ORDER_DATE	REQUIRED_DATE	SHIPPED_DATE	STORE_ID	STAFF_ID
#	NUMBER(38,0)	NUMBER(38,0)	NUMBER(38,0)	DATE	DATE	DATE	NUMBER(38,0)	NUMBER(38,0)

4) Does the datasets has any DUPLICATE(identical rows) ? If yes – can you just keep the first record and remove all rest if its possible without using any JOINS or WINDOW function

No the dataset doesn't contain any duplicate rows. I checked by grouping all the columns and taking count of it.

Example - select order\_id, customer\_id, order\_status, order\_date, required\_date, shipped\_date, store\_id, staff\_id, count(\*) as duplicate\_count

FROM ORDERS

group by order\_id, customer\_id, order\_status, order\_date, required\_date, shipped\_date, store\_id, staff\_id

order by count(\*) desc

ACCOUNTADMIN COMPUTE\_WH Share

Latest Version

```
1 select order_id, customer_id, order_status, order_date, required_date, shipped_date, store_id, staff_id, count(*) as duplicate_count FROM ORDERS
2 group by order_id, customer_id, order_status, order_date, required_date, shipped_date, store_id, staff_id
3 order by count(*) desc
```

Results Chart

	ORDER_STATUS	ORDER_DATE	REQUIRED_DATE	SHIPPED_DATE	STORE_ID	STAFF_ID	DUPLICATE_COUNT
560	4	2018-01-27	2018-01-28	2018-01-28	2	6	1
561	4	2018-02-19	2018-02-21	2018-02-21	2	6	1
562	4	2017-11-13	2017-11-16	2017-11-14	2	7	1
563	4	2017-09-28	2017-10-01	2017-09-30	2	7	1
564	4	2017-09-14	2017-09-16	2017-09-15	3	8	1
565	4	2017-09-02	2017-09-05	2017-09-05	2	7	1
566	4	2017-08-07	2017-08-09	2017-08-08	3	8	1
567	4	2017-06-30	2017-07-03	2017-07-01	2	7	1
568	4	2017-06-26	2017-06-28	2017-06-27	2	6	1

Query Details

Query duration 274ms

Rows 1.6K

Query ID 01aa88d-0404-ba89-

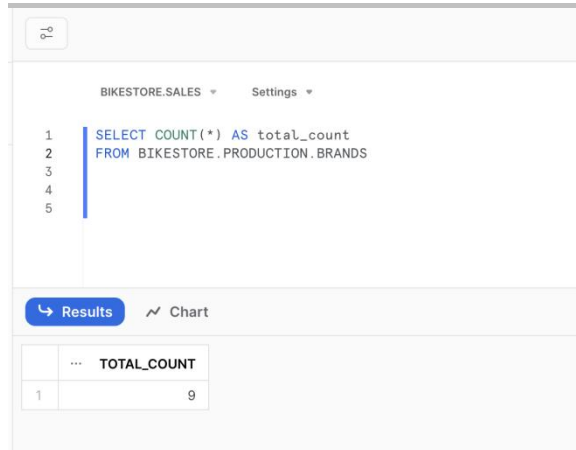
ORDER\_ID

CUSTOMER\_ID

5) How many unique tables are present in each schema and under each table how many records are we having ? (Write SQL Script for the same – I don't need answer like 3/5/4 etc)

There are 4 unique tables present in production schema and 5 tables present in sales schema.

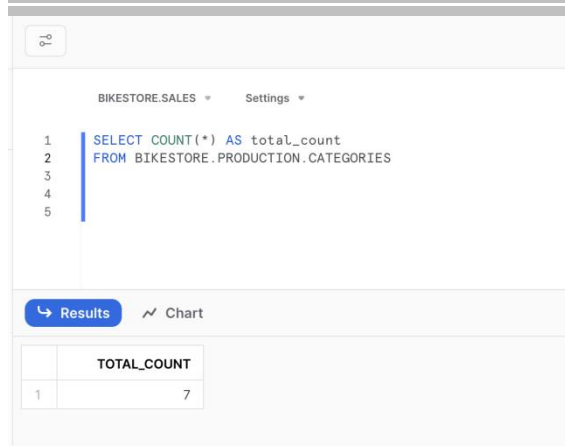
**BIKESTORE.PRODUCTION.BRANDS - 9 records**



```
1 SELECT COUNT(*) AS total_count
2 FROM BIKESTORE.PRODUCTION.BRANDS
3
4
5
```

	TOTAL_COUNT
1	9

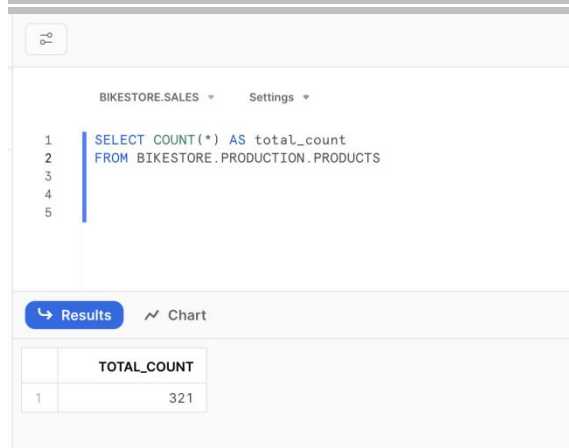
**BIKESTORE.PRODUCTION.CATEGORIES - 7 records**



```
1 SELECT COUNT(*) AS total_count
2 FROM BIKESTORE.PRODUCTION.CATEGORIES
3
4
5
```

	TOTAL_COUNT
1	7

**BIKESTORE.PRODUCTION.PRODUCTS - 321 records**



```
1 SELECT COUNT(*) AS total_count
2 FROM BIKESTORE.PRODUCTION.PRODUCTS
3
4
5
```

	TOTAL_COUNT
1	321

BIKESTORE.PRODUCTION.STOCKS-939records

BIKESTORE.SALES ▾ Settings ▾

1

2

3

4

5

SELECT COUNT(\*) AS total\_count

FROM BIKESTORE.PRODUCTION.STOCKS

↶ Results

↷ Chart

	TOTAL_COUNT
1	939

BIKESTORE.SALES.CUSTOMERS - 1445 records

BIKESTORE.SALES ▾ Settings ▾

1

2

3

4

5

SELECT COUNT(\*) AS total\_count

FROM BIKESTORE.SALES.CUSTOMERS

↶ Results

↷ Chart

	TOTAL_COUNT
1	1,445



BIKESTORE.SALES.STORES - 3 records

18

BIKESTORE.SALES ▾ Settings ▾

1

2

3

4

5

SELECT COUNT(\*) AS total\_count

FROM BIKESTORE.SALES.STORES

↶ Results

~ Chart

	... TOTAL_COUNT
1	3

BIKESTORE.SALES.STAFFS - 10 records

18

BIKESTORE.SALES ▾ Settings ▾

1

2

3

4

5

SELECT COUNT(\*) AS total\_count

FROM BIKESTORE.SALES.STAFFS

↶ Results

~ Chart

	... TOTAL_COUNT
1	10

BIKESTORE.SALES.ORDER\_ITEMS - 4722 records

18

BIKESTORE.SALES ▾ Settings ▾

1

2

3

4

5

SELECT COUNT(\*) AS total\_count

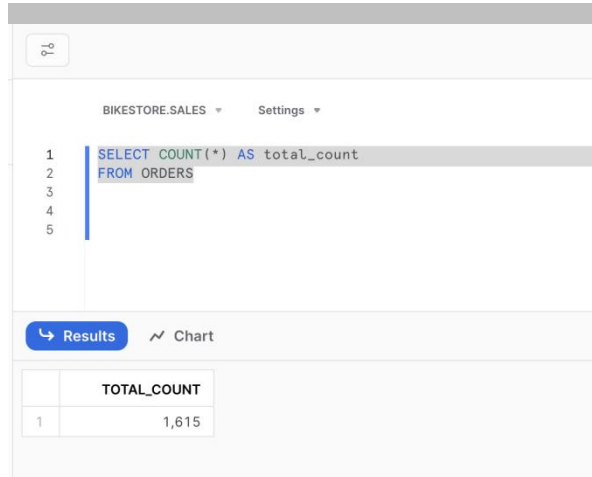
FROM BIKESTORE.SALES.ORDER\_ITEMS

↶ Results

~ Chart

	TOTAL_COUNT
1	4,722

BIKESTORE.SALES.ORDERS - 1615 records

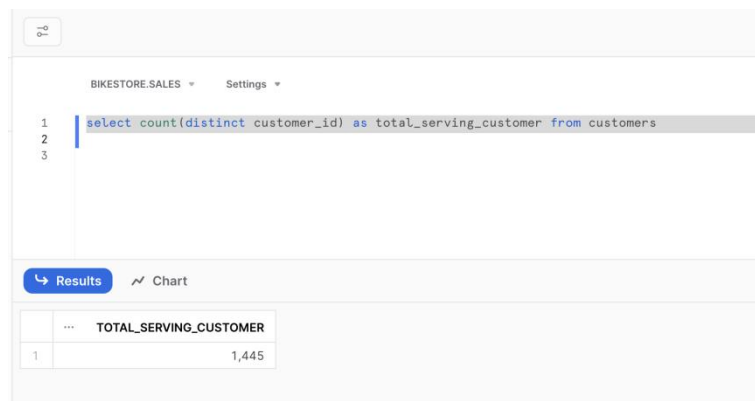


The screenshot shows a SQL query editor with a query bar at the top containing the text "BIKESTORE.SALES" and "Settings". Below the query bar, a SQL query is entered: `SELECT COUNT(*) AS total_count FROM ORDERS`. The query is highlighted in blue. Below the query bar, there are two tabs: "Results" and "Chart". The "Results" tab is selected, and it displays a table with one row and one column, showing the result of the query: `TOTAL_COUNT` with a value of `1,615`.

	TOTAL_COUNT
1	1,615

6) How many total serving customer BikeStore has ?

`select count(distinct customer_id) as total_serving_customer from customers -- 1445`

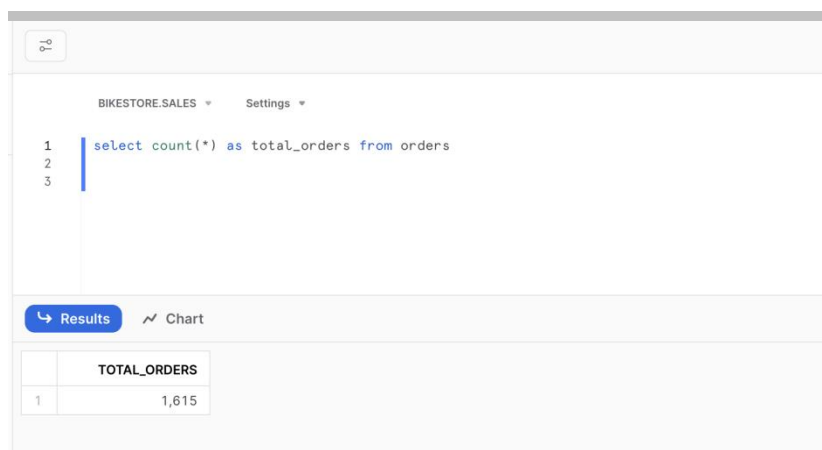


The screenshot shows a SQL query editor with a query bar at the top containing the text "BIKESTORE.SALES" and "Settings". Below the query bar, a SQL query is entered: `select count(distinct customer_id) as total_serving_customer from customers`. The query is highlighted in blue. Below the query bar, there are two tabs: "Results" and "Chart". The "Results" tab is selected, and it displays a table with one row and one column, showing the result of the query: `TOTAL_SERVING_CUSTOMER` with a value of `1,445`.

	TOTAL_SERVING_CUSTOMER
1	1,445

7) How many total orders are there ?

`select count(*) as total_orders from orders - 1615`



The screenshot shows a SQL query editor with a query bar at the top containing the text "BIKESTORE.SALES" and "Settings". Below the query bar, a SQL query is entered: `select count(*) as total_orders from orders`. The query is highlighted in blue. Below the query bar, there are two tabs: "Results" and "Chart". The "Results" tab is selected, and it displays a table with one row and one column, showing the result of the query: `TOTAL_ORDERS` with a value of `1,615`.

	TOTAL_ORDERS
1	1,615

8) Which store has the highest number of sales ?

```
select o.store_id,sum((oi.list_price*oi.quantity)-oi.discount) as total_sales, s.store_name
```

```
From orders as o join order_items as oi on oi.order_id=o.order_id
```

```
join stores as s on o.store_id=s.store_id
```

```
group by o.store_id,s.store_name
```

```
order by total_sales desc
```

```
Limit
```

```
1
```

The screenshot shows a SQL query editor interface. At the top, there's a header with a logo and a user profile icon labeled 'ACCOUNTADMIN'. Below the header, there are tabs for 'BIKESTORE.SALES' and 'Settings'. The main area contains a SQL query: 

```
1 select o.store_id,sum((oi.list_price*oi.quantity)-oi.discount) as total_sales, s.store_name
2 From orders as o join order_items as oi on oi.order_id=o.order_id
3 join stores as s on o.store_id=s.store_id
4 group by o.store_id,s.store_name
5 order by total_sales desc
6 limit 1
7
8
```

 Below the query, there are two buttons: 'Results' and 'Chart'. The 'Results' button is active, and below it, a table displays the results: 

	STORE_ID	TOTAL_SALES	STORE_NAME
1	2	5,825,901.85	Baldwin Bikes

9) Which month the sales was highest and for which store ?

```
select month(cast(orders.order_date as date)) as month,stores.store_name, count(*)
```

```
from orders inner join stores on orders.store_id=stores.store_id
```

```
group by month(cast(orders.order_date as date)),stores.store_name
```

```
order by count(*) desc
```

```
limit 1
```

The screenshot shows a SQL query editor interface. At the top, there's a header with a logo and a user profile icon labeled 'ACCOUNTADMIN'. Below the header, there are tabs for 'BIKESTORE.SALES' and 'Settings'. The main area contains a SQL query: 

```
1 select month(cast(orders.order_date as date)) as month,stores.store_name, count(*)
2 from orders inner join stores on orders.store_id=stores.store_id
3 group by month(cast(orders.order_date as date)),stores.store_name
4 order by count(*) desc
5 limit 1
6
```

 Below the query, there are two buttons: 'Results' and 'Chart'. The 'Results' button is active, and below it, a table displays the results: 

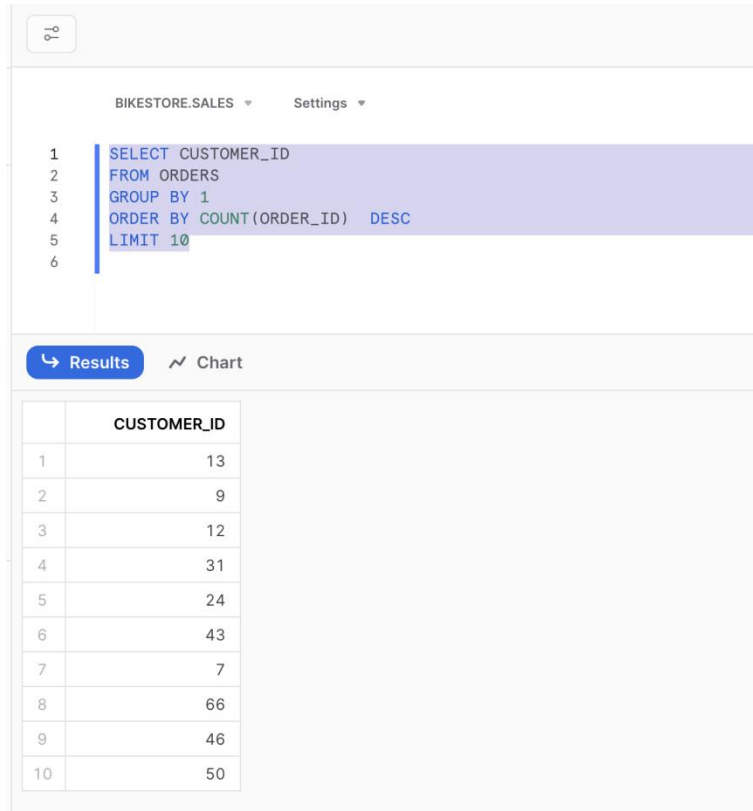
	MONTH	STORE_NAME	COUNT(*)
1	3	Baldwin Bikes	133

10) How many orders each customer has placed (give me top 10 customers)

```
SELECT CUSTOMER_ID FROM ORDERS
```

```
GROUP BY CUSTOMER_ID
```

```
ORDER BY COUNT(ORDER_ID) DESC LIMIT 10
```



The screenshot shows a SQL query editor with the following query:

```
1 SELECT CUSTOMER_ID
2 FROM ORDERS
3 GROUP BY 1
4 ORDER BY COUNT(ORDER_ID) DESC
5 LIMIT 10
6
```

Below the query editor, there are tabs for "Results" and "Chart". The "Results" tab is active, displaying a table with 10 rows and 2 columns: "CUSTOMER\_ID" and an unlabeled column representing the count of orders.

	CUSTOMER_ID
1	13
2	9
3	12
4	31
5	24
6	43
7	7
8	66
9	46
10	50

11) Which are the TOP 3 selling product ?

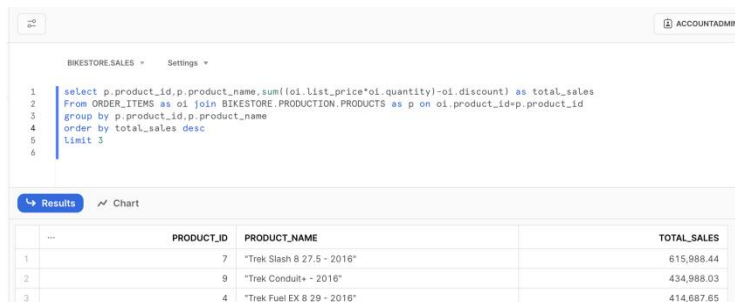
```
select p.product_id,p.product_name,sum((oi.list_price*oi.quantity)-oi.discount) as
total_sales
```

```
From ORDER_ITEMS as oi join BIKESTORE.PRODUCTION.PRODUCTS as p on
oi.product_id=p.product_id
```

```
group by p.product_id,p.product_name
```

```
order by total_sales desc
```

```
limit 3
```



The screenshot shows a SQL query editor with the following query:

```
1 select p.product_id,p.product_name,sum((oi.list_price*oi.quantity)-oi.discount) as total_sales
2 from ORDER_ITEMS as oi join BIKESTORE.PRODUCTION.PRODUCTS as p on oi.product_id=p.product_id
3 group by p.product_id,p.product_name
4 order by total_sales desc
5 limit 3
6
```

Below the query editor, there are tabs for "Results" and "Chart". The "Results" tab is active, displaying a table with 4 columns: "PRODUCT\_ID", "PRODUCT\_NAME", and "TOTAL\_SALES".

	PRODUCT_ID	PRODUCT_NAME	TOTAL_SALES
1	7	"Trek Slash 8 27.5 - 2016"	615,988.44
2	9	"Trek Condukt+ - 2016"	434,988.03
3	4	"Trek Fuel EX 8 29 - 2016"	414,687.65

12) Which was the first and last order placed by the customer who has placed maximum number of orders ?

```
select customer_id, min(order_date) as first_order_date, max(order_date) as last_order_date, count(order_id)
from orders
group by 1
order by count(order_id) desc, min(order_date)
limit 1
```

BIKESTORE.SALES Settings ACCOUNTADMIN

```
1 select customer_id, min(order_date) as first_order_date, max(order_date) as last_order_date, count(order_id)
2 from orders
3 group by 1
4 order by count(order_id) desc, min(order_date)
5 limit 1
6
```

Results Chart

	CUSTOMER_ID	FIRST_ORDER_DATE	LAST_ORDER_DATE	...	COUNT(ORDER_ID)
1	50	2016-02-11	2018-04-12		3

13) For every customer , which is the cheapest product and the costliest product which the customer has bought.

```
Select orders.customer_id, max(order_items.list_price) AS costliest_product,
min(order_items.list_price) AS cheapest_product
from orders inner join order_items on orders.order_id=order_items.order_id
group by orders.customer_id
```

BIKESTORE.SALES Settings ACCOUNTADMIN COMPUTE

```
1 select orders.customer_id, max(order_items.list_price) AS costliest_product, min(order_items.list_price) AS cheapest_product
2 from orders inner join order_items on orders.order_id=order_items.order_id
3 group by orders.customer_id
4
```

Results Chart

	CUSTOMER_ID	COSTLIEST_PRODUCT	CHEAPEST_PRODUCT
1	259	2,899.99	599.99
2	1212	599.99	599.99
3	523	999.99	599.99
4	175	2,999.99	679.99
5	1324	1549.00	429.00
6	94	4,999.99	449.00
7	324	999.99	429.00
8	1204	599.99	269.99
9	60	3,999.99	209.99
10	442	269.99	269.99
11	1326	1,799.99	269.99
12	91	6,499.99	499.99
13	552	1,799.99	269.99
14	1175	1,799.99	299.99

Query De  
Query du  
Rows  
Query ID  
CUSTOME  
1  
COSTLIES  
109.99  
CHEAPEST  
en no

14) Which product has orders more than 200 ?

```
select product_id, count(*) as number_of_orders
from order_items
group by product_id
having count(*)>200
```

There are no product whose orders are more than 200



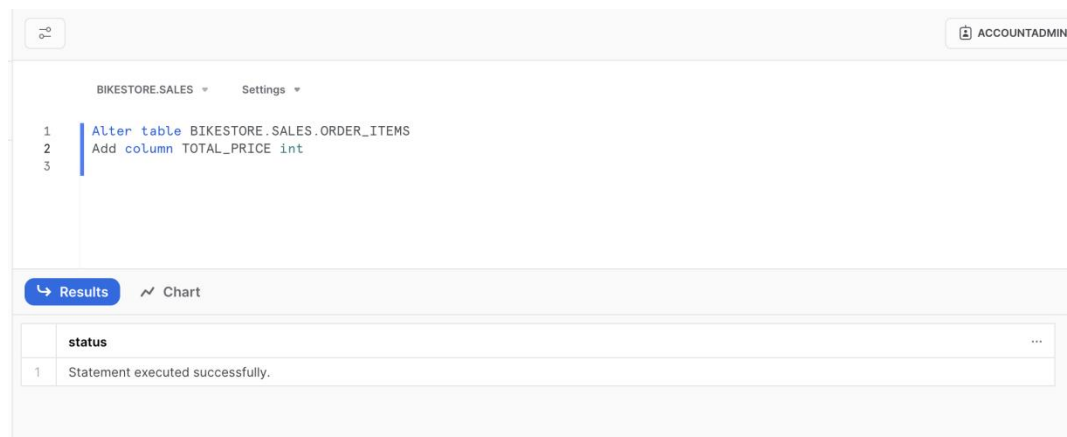
The screenshot shows a SQL query editor interface. The query is: `select product_id, count(*) as number_of_orders from order_items group by product_id having count(*)>200`. Below the query, there are tabs for 'Results' and 'Chart'. The 'Results' tab is active, showing a table with two columns: 'PRODUCT\_ID' and 'NUMBER\_OF\_ORDERS'. The table is empty, and a message below it states 'Query produced no results'.

PRODUCT_ID	NUMBER_OF_ORDERS
------------	------------------

15) Add a column TOTAL\_PRICE with appropriate data type into the sales.order\_items

Alter table sales.order\_items

Add column TOTAL\_PRICE int



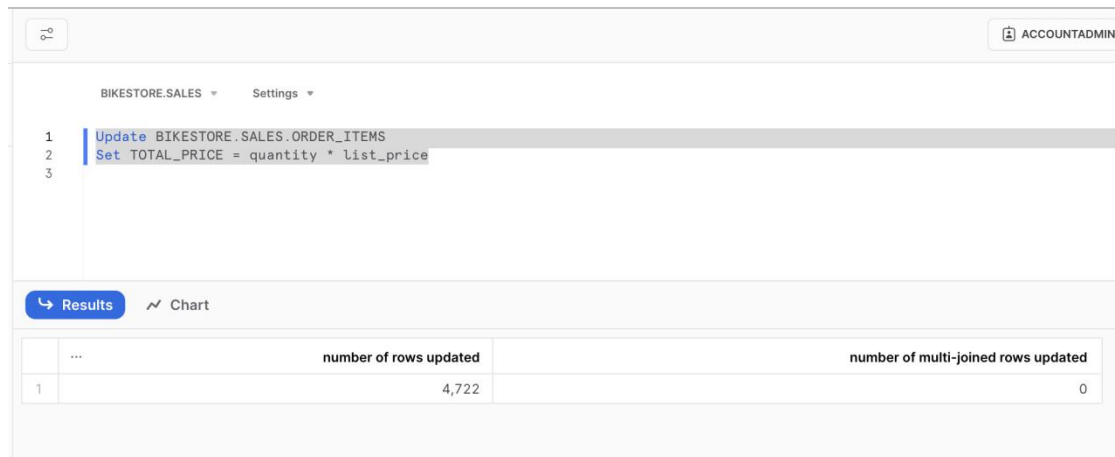
The screenshot shows a SQL query editor interface. The query is: `Alter table BIKESTORE.SALES.ORDER_ITEMS Add column TOTAL_PRICE int`. Below the query, there are tabs for 'Results' and 'Chart'. The 'Results' tab is active, showing a table with one row: 'status' with the value 'Statement executed successfully.'.

status
Statement executed successfully.

16) Calculate  $TOTAL\_PRICE = quantity * list\ price$  and Update the value for all rows in the sales.order\_items table.

Update sales.order\_items

Set  $TOTAL\_PRICE = quantity * list$



The screenshot shows a database management interface with a top bar containing a database icon and the username 'ACCOUNTADMIN'. Below the bar, there are tabs for 'BIKESTORE.SALES' and 'Settings'. The main area displays a SQL statement:

```
1 Update BIKESTORE.SALES.ORDER_ITEMS
2 Set TOTAL_PRICE = quantity * list_price
3
```

Below the SQL editor, there are two buttons: 'Results' (active) and 'Chart'. The results section shows a table with two columns: 'number of rows updated' and 'number of multi-joined rows updated'.

	number of rows updated	number of multi-joined rows updated
1	4,722	0

17) What is the value of the TOTAL\_PRICE paid for all the sales.order\_items ?

Select sum(TOTAL\_PRICE ) from sales.order\_items



The screenshot shows the same database management interface. The SQL editor now contains a SELECT statement:

```
1 Select sum(TOTAL_PRICE) from sales.order_items
2
```

Below the SQL editor, there are two buttons: 'Results' (active) and 'Chart'. The results section shows a table with one column: 'SUM(TOTAL\_PRICE)'.

	SUM(TOTAL_PRICE)
1	8,579,054