Remove duplicates

Write a query to identify the number of duplicates in "sales_transaction" table. Also, create a separate table containing the unique values and remove the the original table from the databases and replace the name of the new table with the original name.

Hint:

Use the "Sales_transaction" table.
There will be two resulting tables in the output. First, the table where the count of duplicates will be identified and in the second table we can check if the duplicates were removed or not by selecting the whole table.

```
select TransactionID , count(*)
from sales_transaction
group by TransactionID
having count(*) > 1 ;
```

```
create table s as
select distinct transactionID , CustomerID , ProductID , QuantityPurchased ,
TransactionDate , Price
from sales_transaction ;
```

```
drop table sales_transaction ;
```

```
alter table s
rename to sales_transaction ;
```

```
select * from sales_transaction ;
```

```
+---------------+-----------+
| TransactionID | count(*) |
+---------------+-----------+
|          4999 |         2 |
|          5000 |         2 |
+---------------+-----------+
+---------------+------------+-----------+-------------------+-----------------+-------+
| transactionID | CustomerID | ProductID | QuantityPurchased | TransactionDate | Price |
+---------------+------------+-----------+-------------------+-----------------+-------+
|             1 |        103 |       120 |                 3 | 2023-01-01      | 30.43 |
|             2 |        436 |       126 |                 1 | 2023-01-01      | 15.19 |
|             3 |        861 |        55 |                 3 | 2023-01-01      | 67.76 |
|             4 |        271 |        27 |                 2 | 2023-01-01      | 65.77 |
|             5 |        107 |       118 |                 1 | 2023-01-01      | 14.55 |
|             6 |         72 |        53 |                 1 | 2023-01-01      | 26.27 |
|             7 |        701 |        39 |                 2 | 2023-01-01      | 95.92 |
```

Fix incorrect pricing

Problem statement

Send feedback

Write a query to identify the discrepancies in the price of the same product in "sales_transaction" and "product_inventory" tables. Also, update those discrepancies to match the price in both the tables.

Hint:

Use the "sales_transaction" and the "product_inventory" tables.

There will be two resulting tables in the output. First, the table where the discrepancies will be identified and in the second table we can check if the discrepancies were updated or not.

- *finding the values*

```
select distinct  pi.ProductID , pi.Price ,st.Price  from
sales_transaction as st
join Product_inventory as pi
on st.ProductID =pi.ProductID
where pi.Price <> st.Price  ;
```

ProductID	Price	Price
51	93.12	9312

- *fixing the incorrect price problem with a dynamic solution*

```
update sales_transaction as st
set price = ( select pi.price from Product_inventory pi where pi.ProductID = st.ProductID)
where st.ProductID in -- will give us all the product id with prices
 (select pi.ProductID from Product_inventory pi where  st.price<> pi.price)
 ;
```

- *checking the results after the fix*

```
select * from sales_transaction
where ProductID = 51 ;
```

TransactionID	CustomerID	ProductID	QuantityPurchased	TransactionDate	Price
88	562	51	2	2023-01-04	93.12
236	231	51	2	2023-01-10	93.12
591	820	51	2	2023-01-25	93.12
1377	172	51	4	2023-02-27	93.12
1910	482	51	3	2023-03-21	93.12
2608	950	51	1	2023-04-19	93.12
2939	944	51	2	2023-05-03	93.12
3377	422	51	3	2023-05-21	93.12
3635	534	51	4	2023-06-01	93.12
3839	973	51	3	2023-06-09	93.12
3918	619	51	1	2023-06-13	93.12

Fixing null values

Write a SQL query to identify the null values in the dataset and replace those by "Unknown".

Hint:
Use the customer_profiles table.
Identify the column/s which contains null values and count the number of cells containing null values. Update those values with "unknown" and showcase the changes that the query has created.

Cleaning date

Write a SQL query to clean the DATE column in the dataset.

Steps:
Create a separate table and change the data type of the date column as it is in TEXT format and name it as you wish to.
Remove the original table from the database.
Change the name of the new table and replace it with the original name of the table.

Hint:
Use the "Sales_transaction" tables.
The resulting table will display a separate column named TransactionDate_updated.

-- Finding missing values

```
select * from customer_profiles
where CustomerID is null or age is null or gender is null or location is null or
JoinDate is null ;
```

CustomerID	Age	Gender	Location	JoinDate
4	19	Other	NULL	04/01/20
113	21	Male	NULL	02/05/20
115	43	Female	NULL	05/05/20
219	23	Male	NULL	27/08/20
239	40	Other	NULL	18/09/20
322	35	Female	NULL	18/12/20
379	27	Female	NULL	18/02/21
405	26	Female	NULL	19/03/21
448	44	Female	NULL	05/05/21

- *fixing null values*

```
update customer_profiles
set location = 'unknown'
where location is null ;
```

```
select * from customer_profiles ;
```

CustomerID	Age	Gender	Location	JoinDate
1	63	Other	East	01/01/20
2	63	Male	North	02/01/20
3	34	Other	North	03/01/20
4	19	Other	unknown	04/01/20
5	57	Male	North	05/01/20
6	22	Other	South	06/01/20
7	56	Other	East	07/01/20
8	65	Female	East	08/01/20

- Cleaning date

Write a SQL query to clean the DATE column in the dataset.

Steps:

Create a separate table and change the data type of the date column as it is in TEXT format and name it as you wish to.
Remove the original table from the database.
Change the name of the new table and replace it with the original name of the table.

Hint:

Use the “Sales_transaction” tables.
The resulting table will display a separate column named TransactionDate_updated.

```
-- Step 1: Add a new column with DATE type
ALTER TABLE sales_transaction ADD COLUMN TransactionDate_updated DATE;

-- Step 2: Populate the new column with converted date values
UPDATE sales_transaction
SET TransactionDate_updated = STR_TO_DATE(TransactionDate, '%Y-%m-%d');

--Step 3: Drop the original TEXT column if no longer needed
ALTER TABLE sales_transaction DROP COLUMN TransactionDate;

--  Step 4: Rename the new column to the original name
ALTER TABLE sales_transaction RENAME COLUMN TransactionDate_updated TO TransactionDate;
```

— OR

```
Create table Sales_transaction1
 as select TransactionID ,  CustomerID , ProductID ,QuantityPurchased ,
  TransactionDate, Price , str_to_date(TransactionDate , '%Y-%m-%d' )  as TransactionDate_updated
from sales_transaction ;

drop table  Sales_transaction ;

alter table  Sales_transaction1
rename to sales_transaction;

select* from sales_transaction;
```

```
+-------------------------+--------+------+-----+---------+-------+
| Field                   | Type   | Null | Key | Default | Extra |
+-------------------------+--------+------+-----+---------+-------+
| TransactionID           | int    | YES  |     | NULL    |       |
| CustomerID              | int    | YES  |     | NULL    |       |
| ProductID               | int    | YES  |     | NULL    |       |
| QuantityPurchased       | int    | YES  |     | NULL    |       |
| TransactionDate         | date   | YES  |     | NULL    |       |
| Price                   | double | YES  |     | NULL    |       |
| TransactionDate_updated | date   | YES  |     | NULL    |       |
```

Total sales summary

Write a SQL query to summarize the total sales and quantities sold per product by the company.

(Here, the data has been already cleaned in the previous steps and from here we will be understanding the different types of data analysis from the given dataset.)

Hint:

Use the "Sales_transaction" table.
The resulting table will display the total quantity purchased by the customers and the total sales done by the company to evaluate the product performance.

Return the result table in descending order corresponding to Total Sales Column.

```
select ProductID , sum(QuantityPurchased) as TotalUnitsSold ,
sum(QuantityPurchased * Price) as TotalSales
from Sales_transaction
group by 1
order by 3 desc ;
```

ProductID	TotalUnitsSold	TotalSales
17	100	9450
87	92	7817.239999999998
179	86	7388.259999999998
96	72	7132.320000000015
54	86	7052.860000000015
187	82	6915.880000000003
156	76	6827.840000000002
57	78	6622.199999999999
200	69	6479.790000000001
127	68	6415.799999999999

Customer purchase frequency

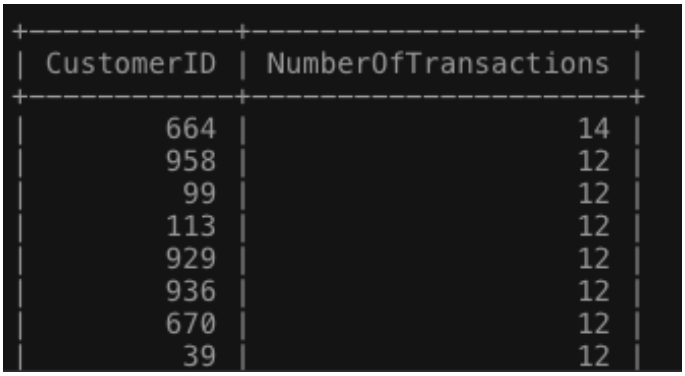
Write a SQL query to count the number of transactions per customer to understand purchase frequency.

Hint:

Use the "Sales_transaction" table.
The resulting table will be counting the number of transactions corresponding to each customerID.

Return the result table ordered by NumberOfTransactions in descending order.

```
select CustomerID , count(TransactionID) as NumberOfTransactions
from Sales_transaction
group by CustomerID
order by 2 desc  ;
```



CustomerID	NumberOfTransactions
664	14
958	12
99	12
113	12
929	12
936	12
670	12
39	12

Product category performance

Write a SQL query to evaluate the performance of the product categories based on the total sales which help us understand the product categories which needs to be promoted in the marketing campaigns.

Hint:

Use the "Sales_transaction" and "product_inventory" table.
The resulting table must display product categories, the aggregated count of units sold for each category, and the total sales value per category.

Return the result table ordering by TotalSales in descending order.

```
select pi.Category , sum(st.QuantityPurchased) as TotalUnitsSold  ,
sum(st.QuantityPurchased*st.Price) as Totalsales
from Sales_transaction as st
join  product_inventory as pi on st.ProductID = pi.ProductID
group by 1
order by 3 desc ;
```

Category	TotalUnitsSold	Totalsales
Home & Kitchen	3477	217755.93999999945
Electronics	3037	177548.4799999996
Clothing	2810	162874.21000000057
Beauty & Health	3001	143824.98999999947

High sales product

Write a SQL query to find the top 10 products with the highest total sales revenue from the sales transactions. This will help the company to identify the High sales products which needs to be focused to increase the revenue of the company.

Hint:

Use the "Sales_transaction" table.
The resulting table should be limited to 10 productIDs whose TotalRevenue (Product of Price and QuantityPurchased) is the highest.

Return the result table ordering by TotalRevenue in descending order.

```
select ProductID,
sum(QuantityPurchased*Price) as TotalRevenue
from Sales_transaction
group by 1
order by 2 desc ;
```

ProductID	TotalRevenue
17	9450
87	7817.239999999998
179	7388.259999999998
96	7132.3200000000015
54	7052.8600000000015
187	6915.880000000003
156	6827.840000000002
57	6622.199999999999

Low sales product

Write a SQL query to find the ten products with the least amount of units sold from the sales transactions, provided that at least one unit was sold for those products.

Hint:

Use the "Sales_transaction" table.
The resulting table should be limited to 10 productIDs whose TotalUnitsSold (sum of QuantityPurchased) is the least. (The limit value can be adjusted accordingly)
Return the result table ordering by TotalUnitsSold in ascending order.

```sql
select ProductID,
sum(QuantityPurchased) as TotalUnitsSold
from Sales_transaction
group by 1
having sum(QuantityPurchased) >= 1
order by 2
limit 10 ;
```

ProductID	TotalUnitsSold
142	27
33	31
174	33
159	35
60	35
41	35
91	35
198	36

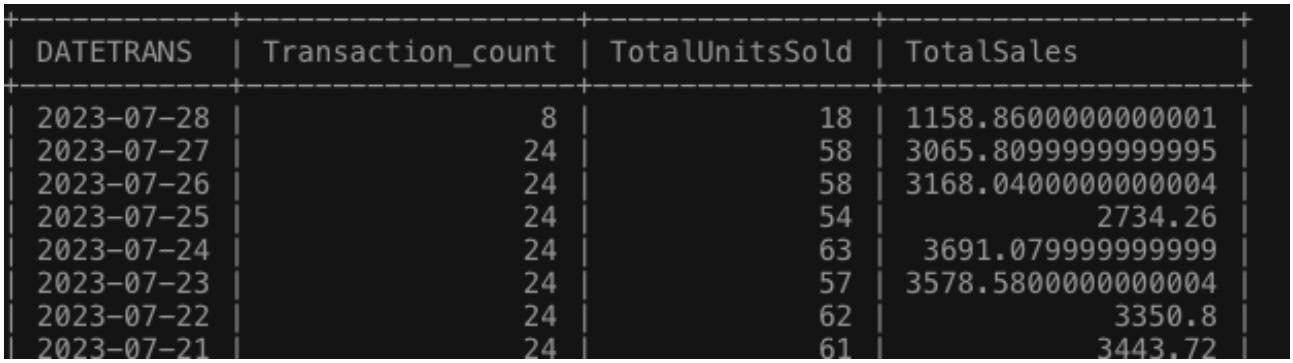
Sales trends

Write a SQL query to identify the sales trend to understand the revenue pattern of the company.

Hint:

- *Use the "sales_transaction" table.*
- *The resulting table must have DATETRANS in date format, count the number of transaction on that particular date, total units sold and the total sales took place.*
- *Return the result table ordered by datetrans in descending order.*

```
select TransactionDate_updated as DATETRANS , count(TransactionID) as
Transaction_count,
sum(QuantityPurchased) as TotalUnitsSold , sum(QuantityPurchased*Price) as
TotalSales
 from sales_transaction
group by 1
order by 1 desc ;
```



DATETRANS	Transaction_count	TotalUnitsSold	TotalSales
2023-07-28	8	18	1158.8600000000001
2023-07-27	24	58	3065.8099999999995
2023-07-26	24	58	3168.0400000000004
2023-07-25	24	54	2734.26
2023-07-24	24	63	3691.079999999999
2023-07-23	24	57	3578.5800000000004
2023-07-22	24	62	3350.8
2023-07-21	24	61	3443.72

Growth rate of sales

Write a SQL query to understand the month on month growth rate of sales of the company which will help understand the growth trend of the company.

Hint:

Use the "sales_transaction" table.
*The resulting table must extract the month from the transactiondate and then the Month on month growth percentange should be calculated. (Total sales present month - total sales previous month/ total sales previous month * 100)*

Return the result table ordering by month.

```
with s as
(select month(TransactionDate_updated) as month ,
sum(QuantityPurchased*Price) as total_sales
 from sales_transaction
 group by 1 )


 select * , lag(total_sales) over ( order by  month ) as previous_month_sales ,
 (total_sales-lag(total_sales) over ( order by  month )) /
 lag(total_sales) over ( order by  month ) * 100 as mom_growth_percentage
 from s
 order by month
```

month	total_sales	previous_month_sales	mom_growth_percentage
1	104289.17999999993	NULL	NULL
2	96690.98999999995	104289.17999999993	-7.285693491884769
3	103271.49	96690.98999999995	6.805701337839299
4	101561.09000000014	103271.49	-1.656217025628141
5	102998.83999999995	101561.09000000014	1.4156504228142972
6	102210.28	102998.83999999995	-0.7656008553105592
7	90981.75000000004	102210.28	-10.985714939827927

High purchase frequency

Problem statement

Send feedback

Write a SQL query that describes the number of transaction along with the total amount spent by each customer which are on the higher side and will help us understand the customers who are the high frequency purchase customers in the company.

Hint:

- *Use the "sales_transaction" table.*
- *The resulting table must have number of transactions more than 10 and TotalSpent more than 1000 on those transactions by the corresponding customers.*
- *Return the result table on the "TotalSpent" in descending order.*

```
select CustomerID , Count(*) as NumberofTransactions ,
sum(Price*QuantityPurchased) as TotalSpent
from sales_transaction
group by CustomerID having  NumberofTransactions > 10 and TotalSpent > 1000
order by TotalSpent desc ;
```

CustomerID	NumberofTransactions	TotalSpent
936	12	2834.4700000000003
664	14	2519.04
670	12	2432.15
39	12	2221.29
958	12	2104.71
75	11	1862.7299999999998
476	11	1821.4399999999998
929	12	1798.42

Occasional customer

Write a SQL query that describes the number of transaction along with the total amount spent by each customer, which will help us understand the customers who are occasional customers or have low purchase frequency in the company.

Hint:

Use the "Sales_transaction" table.
The resulting table must have number of transactions less than or equal to 2 and corresponding total amount spent on those transactions by related customers.
Return the result table of "NumberOfTransactions" in ascending order and "TotalSpent" in descending order.

```
select CustomerID , Count(*) as NumberofTransactions ,
sum(Price*QuantityPurchased) as TotalSpent
from sales_transaction
group by CustomerID having  NumberofTransactions <=2
order by NumberofTransactions , TotalSpent desc ;
```

CustomerID	NumberofTransactions	TotalSpent
94	1	360.64
181	1	298.23
979	1	265.16
317	1	257.73
479	1	254.91
799	1	254.70000000000002
45	1	241.35000000000002
110	1	236.16

Repeate purchases

Write a SQL query that describes the total number of purchases made by each customer against each productID to understand the repeat customers in the company.

Hint:

Use the "Sales_transaction" table.
The resulting table must have "CustomerID", "ProductID" and the number of times that particular customer have purchases the product.
The number of times the customer has purchased should be more than once.
Return the result table in descending order corresponding to the TimesPurchased column.

```
select CustomerID , ProductID , count(*) as TimesPurchased
from Sales_transaction
group by 1,2
having count(*)>1
order by 3 desc ;
```

CustomerID	ProductID	TimesPurchased
685	192	3
758	31	2
75	47	2
233	68	2
133	147	2
602	101	2
584	83	2

Loyalty indicators

Write a SQL query that describes the duration between the first and the last purchase of the customer in that particular company to understand the loyalty of the customer.

Hints:

Use the "Sales_transaction" table.
The DATE column will be majorly in use in the question and the TransactionDate column in Sales_transaction is in text format. Thus, the format of the TransactionDate column should be changed.
The resulting table must have the first date of purchase, the last date of purchase and the difference between the first and the last date of purchase.
The difference between the first and the last date of purchase should be more than 0.
Return the table in descending order corresponding to DaysBetweenPurchases.

```
With transcationDate as (select CustomerID ,
str_to_date(TransactionDate , '%Y-%m-%d') as TransactionDate
From Sales_transaction
)
select CustomerID ,
Min(TransactionDate) as FirstPurchase ,
Max(TransactionDate) as LastPurchase,
datediff( Max(TransactionDate), Min(TransactionDate) ) as DaysBetweenPurchases
from transcationDate
Group by 1
Having (Max(TransactionDate)  - Min(TransactionDate) ) >0
order by 4 desc ;
```

CustomerID	FirstPurchase	LastPurchase	DaysBetweenPurchases
215	2023-01-01	2023-07-28	208
414	2023-01-02	2023-07-26	205
664	2023-01-01	2023-07-24	204
701	2023-01-01	2023-07-23	203
277	2023-01-02	2023-07-24	203
22	2023-01-02	2023-07-24	203
976	2023-01-02	2023-07-24	203
647	2023-01-03	2023-07-25	203
162	2023-01-05	2023-07-27	203
806	2023-01-02	2023-07-23	202
511	2023-01-02	2023-07-23	202
703	2023-01-05	2023-07-26	202

Customer segmentation

Write a SQL query that segments customers based on the total quantity of products they have purchased. Also, count the number of customers in each segment which will help us target a particular segment for marketing.

Hint:

Use the customer_profiles and sales_transaction tables
Create a separate table named customer_segment and create the segments on the total quantity of the purchased products.
To segment customers based on their purchasing behavior for targeted marketing campaigns. Create Customer segments on the following criteria-

Total Quantity of Products Purchased	Customer Segment
1-10	Low
10-30	Mid
>30	High Value

The resulting table should be counting the number of customers in different customer segments.
Return the result table in any order.

```
Create table customer_segment as

select CustomerID ,
case when TotalQuantity > 30 then "High"
when TotalQuantity between 10 and 30 then "Mid"
when TotalQuantity between 1 and 10 then "Low"
else "none" end as CustomerSegment
from
(select cp.CustomerID , sum(st.QuantityPurchased) as TotalQuantity
from customer_profiles as cp
join sales_transaction as st
on cp.CustomerID = st.CustomerID
group by cp.CustomerID ) a ;

select CustomerSegment , count(*) from customer_segment
group by 1;
```

```
+-----------------+----------+
| CustomerSegment | count(*) |
+-----------------+----------+
| Mid             |      627 |
| Low             |      355 |
| High            |        7 |
+-----------------+----------+
```