# Medical inventory optimization

## Exploratory Data Analysis (SQL) by Vaka Prasanna

### Software: postgreSQL

1. **Displaying the table.**

SELECT \* FROM practice LIMIT 20;

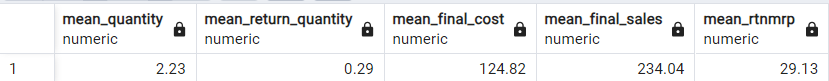
### A screen shot of a computer Description automatically generatedOutput:

1. **Calculating the first moment (measures of central tendency such as mean, median, mode) for the dataset.**

### Mean:

SELECT

ROUND(AVG(Quantity), 2) AS mean\_quantity, ROUND(AVG(ReturnQuantity), 2) AS mean\_return\_quantity, ROUND(AVG(Final\_Cost), 2) AS mean\_final\_cost, ROUND(AVG(Final\_Sales), 2) AS mean\_final\_sales, ROUND(AVG(RtnMRP), 2) AS mean\_rtnmrp FROM practice; **Output:**



### Median:

SELECT

ROUND(AVG(Final\_Cost), 2) AS median\_final\_cost, ROUND(AVG(Final\_Sales), 2) AS median\_final\_sales, ROUND(AVG(Quantity), 2) AS median\_quantity, ROUND(AVG(ReturnQuantity), 2) AS median\_return\_quantity, ROUND(AVG(RtnMRP), 2) AS median\_rtnmrp

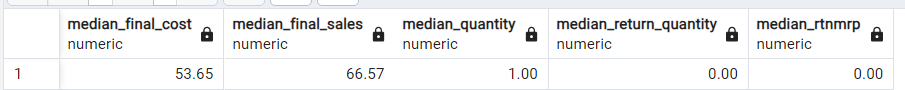
FROM (

SELECT Final\_Cost, Final\_Sales, Quantity, ReturnQuantity, RtnMRP, ROW\_NUMBER() OVER (ORDER BY Final\_Cost) AS row\_num, COUNT(\*) OVER () AS total\_rows FROM practice

) AS subquery

WHERE row\_num IN (FLOOR((total\_rows + 1) / 2), CEILING((total\_rows + 1) / 2));

**Output:**



**Mode:**

SELECT

mode\_quantity.mode\_value AS mode\_quantity, mode\_return\_quantity.mode\_value AS mode\_return\_quantity, mode\_final\_cost.mode\_value AS mode\_final\_cost, mode\_final\_sales.mode\_value AS mode\_final\_sales, mode\_rtnmrp.mode\_value AS mode\_rtnmrp

FROM (

SELECT Quantity AS mode\_value, COUNT(\*) AS mode\_count FROM practice

GROUP BY Quantity ORDER BY COUNT(\*) DESC LIMIT 1

) AS mode\_quantity, (

SELECT ReturnQuantity AS mode\_value, COUNT(\*) AS mode\_count FROM practice

GROUP BY ReturnQuantity ORDER BY COUNT(\*) DESC LIMIT 1

) AS mode\_return\_quantity, (

SELECT Final\_Cost AS mode\_value, COUNT(\*) AS mode\_count FROM practice

GROUP BY Final\_Cost ORDER BY COUNT(\*) DESC LIMIT 1

) AS mode\_final\_cost, (

SELECT Final\_Sales AS mode\_value, COUNT(\*) AS mode\_count FROM practice

GROUP BY Final\_Sales ORDER BY COUNT(\*) DESC LIMIT 1

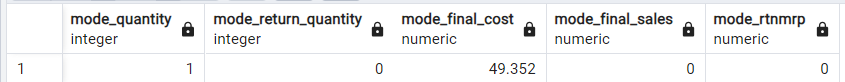
) AS mode\_final\_sales, (

SELECT RtnMRP AS mode\_value, COUNT(\*) AS mode\_count FROM practice

GROUP BY RtnMRP ORDER BY COUNT(\*) DESC LIMIT 1

) AS mode\_rtnmrp;

### Output:



1. **Calculating the second moment (measures of dispersion such as variance, standard deviation, range) for the dataset.**

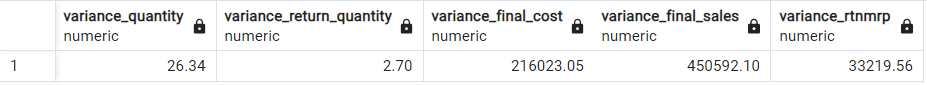
### Variance:

SELECT

ROUND(VARIANCE(Quantity), 2) AS variance\_quantity, ROUND(VARIANCE(ReturnQuantity), 2) AS variance\_return\_quantity, ROUND(VARIANCE(Final\_Cost), 2) AS variance\_final\_cost, ROUND(VARIANCE(Final\_Sales), 2) AS variance\_final\_sales, ROUND(VARIANCE(RtnMRP), 2) AS variance\_rtnmrp

FROM practice;

### Output:



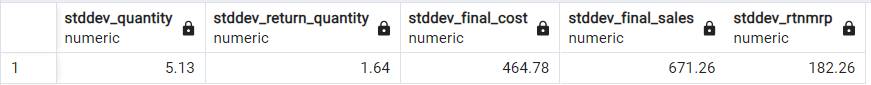
**Standard Deviation:**

SELECT

ROUND(STDDEV(Quantity), 2) AS stddev\_quantity, ROUND(STDDEV(ReturnQuantity), 2) AS stddev\_return\_quantity, ROUND(STDDEV(Final\_Cost), 2) AS stddev\_final\_cost, ROUND(STDDEV(Final\_Sales), 2) AS stddev\_final\_sales, ROUND(STDDEV(RtnMRP), 2) AS stddev\_rtnmrp

FROM practice;

### Output:



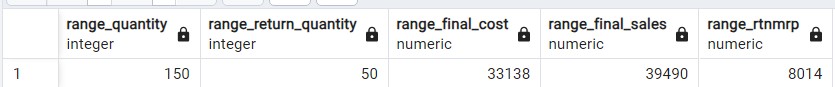
**Range:**

SELECT

MAX(Quantity) - MIN(Quantity) AS range\_quantity, MAX(ReturnQuantity) - MIN(ReturnQuantity) AS range\_return\_quantity, MAX(Final\_Cost) - MIN(Final\_Cost) AS range\_final\_cost, MAX(Final\_Sales) - MIN(Final\_Sales) AS range\_final\_sales, MAX(RtnMRP) - MIN(RtnMRP) AS range\_rtnmrp

FROM practice;

### Output:



1. **Calculating the third moment (skewness) for the dataset. Skewness:**

SELECT 'Quantity' AS column\_name,

ROUND((SUM(POW(Quantity - (SELECT AVG(Quantity) FROM practice), 3)) / (COUNT(\*) \* POW(STDDEV(Quantity), 3))), 2) AS skewness\_value

FROM practice UNION ALL

SELECT 'ReturnQuantity' AS column\_name,

ROUND((SUM(POW(ReturnQuantity - (SELECT AVG(ReturnQuantity) FROM practice), 3)) / (COUNT(\*) \* POW(STDDEV(ReturnQuantity), 3))), 2) AS skewness\_value

FROM practice UNION ALL

SELECT 'Final\_Cost' AS column\_name,

ROUND((SUM(POW(Final\_Cost - (SELECT AVG(Final\_Cost) FROM practice), 3)) / (COUNT(\*) \* POW(STDDEV(Final\_Cost), 3))), 2) AS skewness\_value

FROM practice UNION ALL

SELECT 'Final\_Sales' AS column\_name,

ROUND((SUM(POW(Final\_Sales - (SELECT AVG(Final\_Sales) FROM practice), 3)) / (COUNT(\*) \* POW(STDDEV(Final\_Sales), 3))), 2) AS skewness\_value

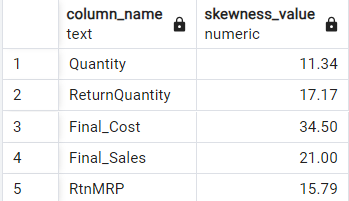
FROM practice UNION ALL

SELECT 'RtnMRP' AS column\_name,

ROUND((SUM(POW(RtnMRP - (SELECT AVG(RtnMRP) FROM practice), 3)) / (COUNT(\*) \*

POW(STDDEV(RtnMRP), 3))), 2) AS skewness\_value FROM practice;

### Output:



1. **Calculating the fourth moment (kurtosis) for the dataset. Kurtosis:**

SELECT

ROUND((SUM(POWER(Quantity - avg\_value, 4)) / (COUNT(Quantity) \* POWER(STDDEV(Quantity), 4))), 2) AS kurtosis\_quantity,

ROUND((SUM(POWER(ReturnQuantity - avg\_value, 4)) / (COUNT(ReturnQuantity) \* POWER(STDDEV(ReturnQuantity), 4))), 2) AS kurtosis\_return\_quantity,

ROUND((SUM(POWER(Final\_Cost - avg\_value, 4)) / (COUNT(Final\_Cost) \* POWER(STDDEV(Final\_Cost), 4))), 2) AS kurtosis\_final\_cost,

ROUND((SUM(POWER(Final\_Sales - avg\_value, 4)) / (COUNT(Final\_Sales) \* POWER(STDDEV(Final\_Sales), 4))), 2) AS kurtosis\_final\_sales,

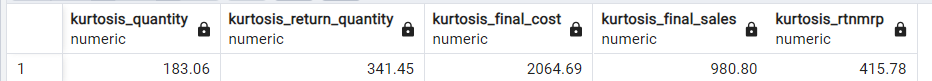
ROUND((SUM(POWER(RtnMRP - avg\_value, 4)) / (COUNT(RtnMRP) \* POWER(STDDEV(RtnMRP), 4))),

2) AS kurtosis\_rtnmrp FROM

(SELECT

AVG(Quantity) AS avg\_value, STDDEV(Quantity) AS stddev\_value, COUNT(Quantity) AS count\_value FROM practice) AS subquery, practice;

**Output:**



# Medical inventory optimization

## Pre-processing Code (SQL) by Vaka Prasanna

**Software: postgreSQL**

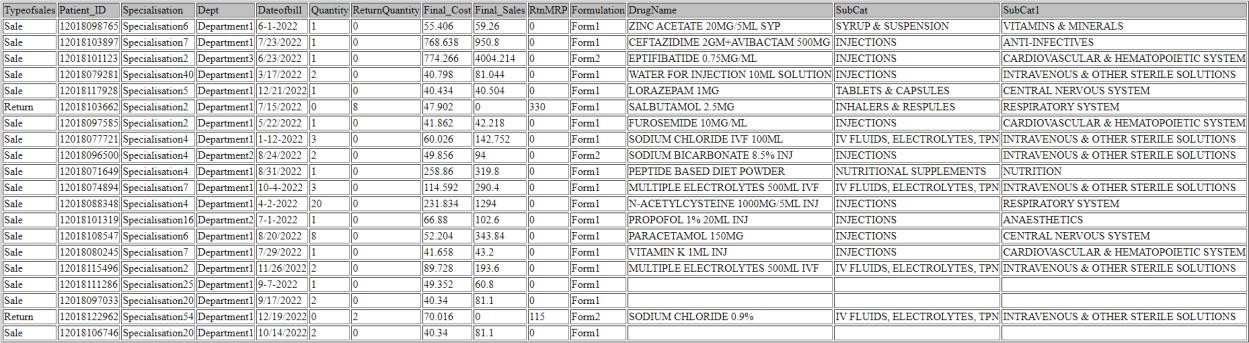
### Importing the table in CSV format into postgreSQL.

Right-click on tables -> table data import wizard -> browse to select the table ‘practice’.

### Displaying the table.

SELECT \* FROM practice LIMIT 20;

### Output:

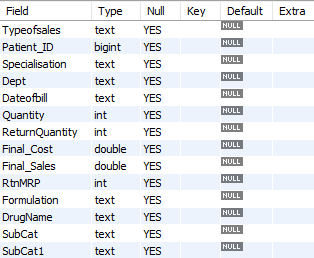


**Observation:** The 'Dateofbill' column in the dataset exhibits inconsistent date formats, with a combination of forward slashes ("/") and dashes ("-"). To ensure uniformity, we can store dates in a consistent format by replacing the forward slashes with dashes.

### Checking the schema of the dataset to ensure that all columns have the correct format.

DESCRIBE practice;

### Output:



**Observation:** The 'Dateofbill' column in the dataset is currently set as "text", but it would be better to store dates in a date-specific data type like "DATE" to ensure proper handling and sorting of dates.

### Fixing inconsistent date formats in the Dateofbill column and creating a new table clean\_projectfinaldata with transformed date values and the other selected columns from the ‘practice’ table.

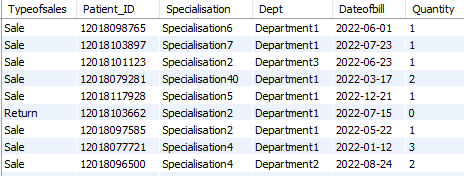
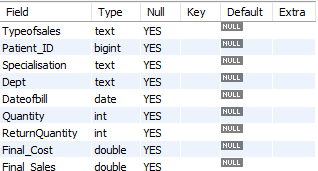
CREATE TABLE clean\_projectfinaldata AS

SELECT Typeofsales, Patient\_ID, Specialisation, Dept,

STR\_TO\_DATE(REPLACE(Dateofbill,'/','-'),'%m-%d-%Y') AS Dateofbill, Quantity, ReturnQuantity, Final\_Cost, Final\_Sales, RtnMRP, Formulation, DrugName, SubCat, SubCat1

FROM practice;

### Datatype and format of 'Dateofbill' column after Formatting:



1. **Counting the missing and non-missing values for each column and the total number of rows in the 'clean\_projectfinaldata' table.**

SELECT

COUNT(CASE WHEN TRIM(Typeofsales) = '' OR Typeofsales IS NULL THEN 1 END) AS typeofsales\_missing,

COUNT(CASE WHEN TRIM(Typeofsales) <> '' AND Typeofsales IS NOT NULL THEN 1 END) AS

typeofsales\_non\_missing,

COUNT(CASE WHEN Patient\_ID IS NULL THEN 1 END) AS patient\_id\_missing, COUNT(CASE WHEN Patient\_ID IS NOT NULL THEN 1 END) AS patient\_id\_non\_missing,

COUNT(CASE WHEN TRIM(Specialisation) = '' OR Specialisation IS NULL THEN 1 END) AS specialisation\_missing,

COUNT(CASE WHEN TRIM(Specialisation) <> '' AND Specialisation IS NOT NULL THEN 1 END) AS specialisation\_non\_missing,

COUNT(CASE WHEN TRIM(Dept) = '' OR Dept IS NULL THEN 1 END) AS dept\_missing,

COUNT(CASE WHEN TRIM(Dept) <> '' AND Dept IS NOT NULL THEN 1 END) AS dept\_non\_missing,

COUNT(CASE WHEN TRIM(Dateofbill) = '' OR Dateofbill IS NULL THEN 1 END) AS dateofbill\_missing, COUNT(CASE WHEN TRIM(Dateofbill) <> '' AND Dateofbill IS NOT NULL THEN 1 END) AS

dateofbill\_non\_missing,

COUNT(CASE WHEN Quantity IS NULL THEN 1 END) AS quantity\_missing, COUNT(CASE WHEN Quantity IS NOT NULL THEN 1 END) AS quantity\_non\_missing, COUNT(CASE WHEN ReturnQuantity IS NULL THEN 1 END) AS returnquantity\_missing,

COUNT(CASE WHEN ReturnQuantity IS NOT NULL THEN 1 END) AS returnquantity\_non\_missing, COUNT(CASE WHEN Final\_Cost IS NULL THEN 1 END) AS final\_cost\_missing,

COUNT(CASE WHEN Final\_Cost IS NOT NULL THEN 1 END) AS final\_cost\_non\_missing, COUNT(CASE WHEN Final\_Sales IS NULL THEN 1 END) AS final\_sales\_missing, COUNT(CASE WHEN Final\_Sales IS NOT NULL THEN 1 END) AS final\_sales\_non\_missing, COUNT(CASE WHEN RtnMRP IS NULL THEN 1 END) AS rtnmrp\_missing,

COUNT(CASE WHEN RtnMRP IS NOT NULL THEN 1 END) AS rtnmrp\_non\_missing,

COUNT(CASE WHEN TRIM(Formulation) = '' OR Formulation IS NULL THEN 1 END) AS formulation\_missing,

COUNT(CASE WHEN TRIM(Formulation) <> '' AND Formulation IS NOT NULL THEN 1 END) AS

formulation\_non\_missing,

COUNT(CASE WHEN TRIM(DrugName) = '' OR DrugName IS NULL THEN 1 END) AS

drugname\_missing,

COUNT(CASE WHEN TRIM(DrugName) <> '' AND DrugName IS NOT NULL THEN 1 END) AS

drugname\_non\_missing,

COUNT(CASE WHEN TRIM(SubCat) = '' OR SubCat IS NULL THEN 1 END) AS subcat\_missing, COUNT(CASE WHEN TRIM(SubCat) <> '' AND SubCat IS NOT NULL THEN 1 END) AS

subcat\_non\_missing,

COUNT(CASE WHEN TRIM(SubCat1) = '' OR SubCat1 IS NULL THEN 1 END) AS subcat1\_missing, COUNT(CASE WHEN TRIM(SubCat1) <> '' AND SubCat1 IS NOT NULL THEN 1 END) AS

subcat1\_non\_missing, COUNT(\*) AS total\_rows FROM clean\_projectfinaldata;

### Output:



**Observation:**

Missing values have been identified in the following columns: Formulation, DrugName, SubCat, and SubCat1.

### Replacing the missing values with ‘unknown’ in the columns Formulation, DrugName, SubCat and SubCat1.

UPDATE clean\_projectfinaldata SET

Formulation = CASE WHEN Formulation = '' THEN 'unknown' ELSE Formulation END;

UPDATE clean\_projectfinaldata SET

DrugName = CASE WHEN DrugName = '' THEN 'unknown' ELSE DrugName END;

UPDATE clean\_projectfinaldata SET

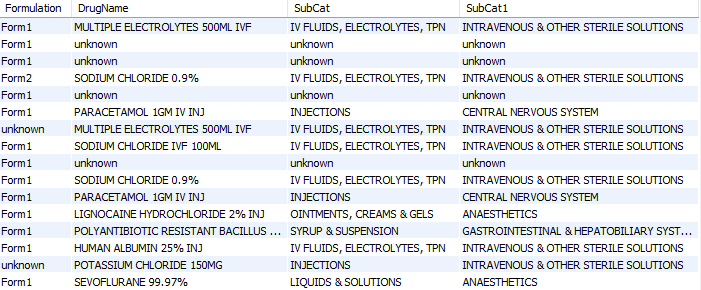
SubCat = CASE WHEN SubCat = '' THEN 'unknown' ELSE SubCat END;

UPDATE clean\_projectfinaldata SET

SubCat1 = CASE WHEN SubCat1 = '' THEN 'unknown' ELSE SubCat1 END;

### Showing the columns after replacing the missing values with ‘unknown’:

SELECT Formulation, DrugName, SubCat, SubCat1 FROM clean\_projectfinaldata;



### Creating a new table called `missing\_values` by selecting rows from `clean\_projectfinaldata` where any of the columns (`Formulation`, `DrugName`, `SubCat`, or `SubCat1`) has the value 'unknown'.

CREATE TABLE missing\_values AS SELECT \*

FROM clean\_projectfinaldata WHERE Formulation = 'unknown'

OR DrugName = 'unknown' OR SubCat = 'unknown'

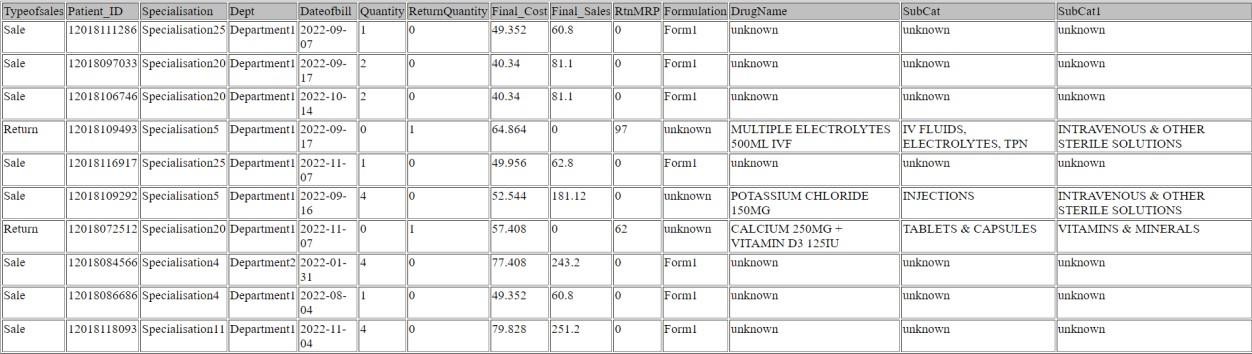
OR SubCat1 = 'unknown';

### Showing missing\_values table and count of records with at least one or more missing values:

SELECT \* FROM missing\_values;

SELECT COUNT(\*) AS missing\_records\_count FROM missing\_values;

### Output:



**Observation:**

A total of 2181 records with missing values have been retrieved. These records need to be rechecked with the client for further discussion. We can refer to the 'missing\_values' table for reviewing the specific data.

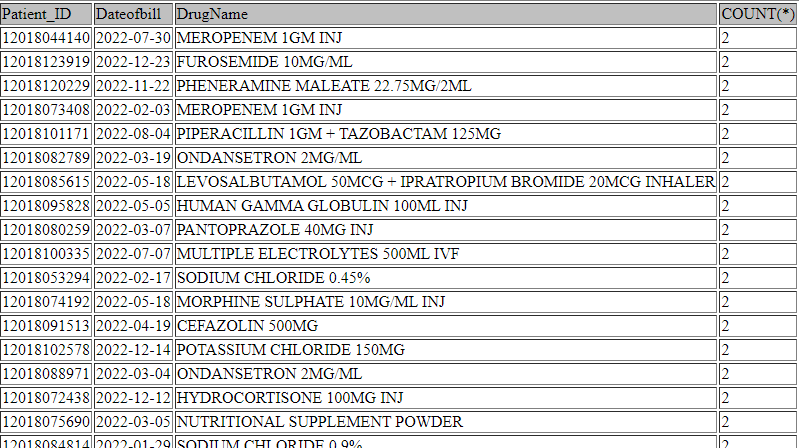
### Identifying duplicate rows based on Patient\_ID, Dateofbill, and DrugName excluding rows where the DrugName is 'unknown'.

SELECT Patient\_ID, Dateofbill, DrugName, COUNT(\*) FROM clean\_projectfinaldata

WHERE DrugName <> 'unknown'

GROUP BY Patient\_ID, Dateofbill, DrugName HAVING COUNT(\*) > 1;

### Output:



1. **Removing the duplicate rows from clean\_projectfinaldata table and counting the remaining rows.**

DELETE FROM clean\_projectfinaldata

WHERE (Patient\_ID, Dateofbill, DrugName) IN ( SELECT t.Patient\_ID, t.Dateofbill, t.DrugName FROM (

SELECT Patient\_ID, Dateofbill, DrugName FROM clean\_projectfinaldata

GROUP BY Patient\_ID, Dateofbill, DrugName HAVING COUNT(\*) > 1

) AS t

);

SELECT COUNT(\*) AS total\_rows FROM clean\_projectfinaldata;

### Output:



**Observation:**

After removing the duplicate rows from the `clean\_projectfinaldata` table, the total number of rows has been reduced from 14,218 to 13,967.

### In a normal distribution, approximately 68%, 95%, and 99.7% of the data falls within one, two, and three standard deviations of the mean respectively. By using three standard deviations as the threshold for removing outliers, we are effectively removing data points that are more than three standard deviations away from the mean.

CREATE TABLE new\_table AS SELECT \*

FROM clean\_projectfinaldata WHERE Quantity BETWEEN

(SELECT AVG(Quantity) - 3 \* STDDEV(Quantity) FROM clean\_projectfinaldata) AND

(SELECT AVG(Quantity) + 3 \* STDDEV(Quantity) FROM clean\_projectfinaldata)

AND ReturnQuantity BETWEEN

(SELECT AVG(ReturnQuantity) - 3 \* STDDEV(ReturnQuantity) FROM clean\_projectfinaldata) AND

(SELECT AVG(ReturnQuantity) + 3 \* STDDEV(ReturnQuantity) FROM clean\_projectfinaldata)

AND Final\_Cost BETWEEN

(SELECT AVG(Final\_Cost) - 3 \* STDDEV(Final\_Cost) FROM clean\_projectfinaldata) AND

(SELECT AVG(Final\_Cost) + 3 \* STDDEV(Final\_Cost) FROM clean\_projectfinaldata)

AND Final\_Sales BETWEEN

(SELECT AVG(Final\_Sales) - 3 \* STDDEV(Final\_Sales) FROM clean\_projectfinaldata) AND

(SELECT AVG(Final\_Sales) + 3 \* STDDEV(Final\_Sales) FROM clean\_projectfinaldata)

AND RtnMRP BETWEEN

(SELECT AVG(RtnMRP) - 3 \* STDDEV(RtnMRP) FROM clean\_projectfinaldata) AND

(SELECT AVG(RtnMRP) + 3 \* STDDEV(RtnMRP) FROM clean\_projectfinaldata);

### Showing the count of rows after removing the outliers:

SELECT COUNT(\*) AS total\_rows FROM new\_table;

### Output:



**Observation:**

Outliers have been excluded from the ‘clean\_projectfinaldata’ table and the filtered records have been stored in the ‘new\_table’. The new\_table’ now contains 13,399 records, excluding the outliers. To save the cleaned dataset with 13,399 rows in CSV format, we can go to "Query" -> "Export Results" -> "Save as CSV". We can specify the file name and location to save the CSV file, allowing us to import it for future exploratory data analysis (EDA) or other purposes.