Liquor Management System

1st Mitali Deshpande
SEAS
University at Buffalo
Buffalo, New York
mitalila@buffalo.edu

2nd Shraddha Waphare
SEAS
University at Buffalo
Buffalo, New York
swaphare@buffalo.edu

3rd Yash Abhyankar
SEAS
University at Buffalo
Buffalo, New York
yabhyank@buffalo.edu

***Abstract*—This paper focuses on the act of using data and insights to identify how much of a specific product or service customers will want to purchase during a defined time period i.e identifying the demand of sales stock. Improper management or estimation skills could lead to problems like stock out over stock. So, to help overcome these loss causing problems, our system will help retailers understand how much stock to have at hand at a given time based on the previous data. This proposed work has implemented various clauses like partition by and row by, aggregate functions, type casts, joins, etc. We have also performed Normalization for data redundancy.**

***Index Terms*—BCNF, Normalization, Query optimization, Triggers**

I. Introduction

Every business is looking for ways to cut costs. In a business, demand forecasting uses facts and understandings to identify the need of a specific item or service customers tend to buy during a specific time interval. This gives businessmen an idea of the stock to be maintained at a particular period. When we implement a proper methodology to your business, we are cutting costs in a few ways. Firstly, we are reducing the amount of capital we have tied up in unneeded inventory. And the less stock on hand we have, the lower our holding costs. Secondly, we are making sure you capitalize on every sale opportunity by not disappointing customers with out of stocks. Additionally, businesses are expanding on a large scale. With this increasing competition, every business should cope up with the increasing demand by considering various factors like time and seasonality of goods.

Sales analysis helps determine production volume by taking into account capital, manpower, available equipment etc. It is also helpful in guiding production and other business activities for achieving targets.

II. Dataset

A. Description

We have used Iowa State's liquor dataset. The Iowa Department of Commerce requires that every store that sells alcohol in bottled form for off-the-premises consumption must hold a class "E" liquor license. Liquor sales made by stores must be registered with the Iowa Department of Commerce which is then published by the state as an open data. The dataset has data from the year 2012 to 2015. Using this dataset, we can analyse and get insights of liquor sales in Iowa state and can be representative of sale activity in the United States as well. It can help answer questions like how much alcohol is sold in a particular time period? Most popular alcohols? Volume of liquor sold? Total vendors in a particular county? etc.

B. Data Pre-processing

The dataset was around 3.5GB which consisted of over 10 million rows. So, we split the dataset into a smaller .csv form with 20k rows for ease while using PostgreSQL. Certain, rows in .csv had store number, zip code values missing hence, we deleted those rows to weed out the missing values in the dataset. After this, we created tables to establish a schema and generated an ER diagram (Fig.1) to visualize it.

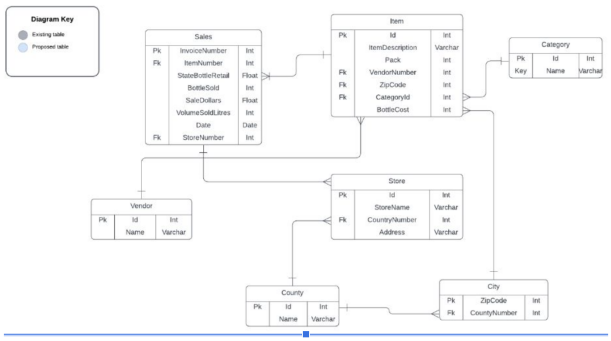


Fig. 1. Initial ER diagram.

III. Normalization

To remove redundancies from the table, we performed Normalization.

A. 1NF

A table is in 1NF if all the columns contain only atomic values. In our dataset, multi-values were not present in any column. So, the dataset was already in 1NF.

B. 2NF

Condition to be in 2NF is that the table should have no partial dependency present in it. We considered the Items Table which contained Item Number, Description, Category ID, Vendor Number, Zip code and Bottle cost. Here, Vendor number and Zip code alone could not decide the value of item. Hence, partial dependency existed and we removed them.

C. 3NF

Table is in 3NF if no transitive dependency exists for non prime attributes. The Sales table consisted of Item Number, Invoice Number, State bottle retail, Bottle sold, Sale dollars, Date, Volume sold and Store Number. Sale dollars was the product of State bottle retail and volume sold. Thus, there was Transitive dependency present which we removed and the table was now in 3NF.

D. BCNF

BCNF must satisfy two conditions:

- Table in 3NF
- For any dependency

$$X -> Y$$

X must be a Candidate or Super key.

Schema already in 3NF. All the candidate keys present has a relation with all the determinants.

- For Sales table:

$$invoicenumber -> date$$
$$invoicenumber -> storenumber$$
$$invoicenumber -> vendornumber$$
$$invoicenumber -> zipcode$$
$$invoicenumber -> itemnumber$$
$$invoicenumber -> bottlesold$$
$$invoicenumber -> statebottleretail$$
$$invoicenumber -> volumesold$$

- For Vendor table:

$$vendorid -> vendorname$$

- For Items table:

$$itemid -> itemdescription$$
$$itemid -> categoryid$$

- For City table:

$$zipcode -> countynumber$$

- For Store table:

$$id -> storename$$
$$id -> countyname$$

- For Address table:

$$storenumber -> addressword$$

- For County table:

$$countyid -> countyname$$

- For Category table:

$$id -> categoryname$$

IV. Relations between Tables

Initially, we only had one table of IOWA liquor. We converted this single table into 8 different tables to remove the redundancies present in the data. The main table of our project is the Sales table. And, after normalization, all the tables are connected using different constraints.

V. Updated ER Diagram

Fig. 2. Final ER Diagram

On comparing Fig. 1. and Fig. 2. we can clearly see that after performing Normalization, the ER diagram and the tables are been updated.

After Normalization we got the following tables:

- Store Table
- County Table
- Category Table
- City Table
- Items Table
- Vendor Table
- Address Table
- Sales Table

A. Table Descriptions

- Store Table: Store table has three columns with primary key as "ID" which also serves as a foreign for for Sales table. It has columns like "countynumber" and "storename". Where "countynumber" is a foreign key from County table.
- County Table: County table has two columns "countyID" and "countyname". Where "countyID" is serving as a foreign key to tables Store and Sales.
- Category Table: Category table has two columns "ID" and "categoryname". Here, "ID" is a primary key and is a foreign key to table Items.
- City Table: This has two columns "zipcode" and "countynumber". "zipcode" is a primary key here.

- Items Table: Table Items has ”itemID” as a primary key and ”categoryID” as a foreign key from table Category and has one more column called ”itemdescription”.
- Vendor Table: Vendor table has two columns ”vendorID” and ”vendorname”. ”vendorID” is a primary key and is serving as a foreign key for table Sales.
- Address Table: Has two columns ”storenumber” and ”address”. ”storenumber” is a foreign key of table Address and also a primary key for table Store.
- Sales Table: Table Sales is our primary table with 9 columns. ”invoicenumber” is the primary key here which uniquely identifies as the data entries of the table. It is of integer type. The ”date” column represents the purchase date of a particular product. Sales table has four foreign keys namely ”storenumber”, ”vendornumber”, ”zipcode” and ”itemnumber”. It also has columns like ”bottlesold”, ”retailprice” and ”volumesold” which represents number of bottles sold of that particular type, retail cost of the liquor and volume in litres of liquor sold respectively.

VI. Query Execution

1. Total sales of liquor of a single category for all the stores in descending order (Top 20)

We, used aggregate function SUM and joined two tables to find top 20 total sales of liquor of a single category for all the stores in descending order. Refer Fig. 3.

select sum(s.volumesold::decimal),ca.categoryname from public.sales s join public.items i on i.itemid=s.itemnumber join category ca on ca.id=i.categoryid group by ca.categoryname order by sum(s.volumesold::decimal) desc limit 20;

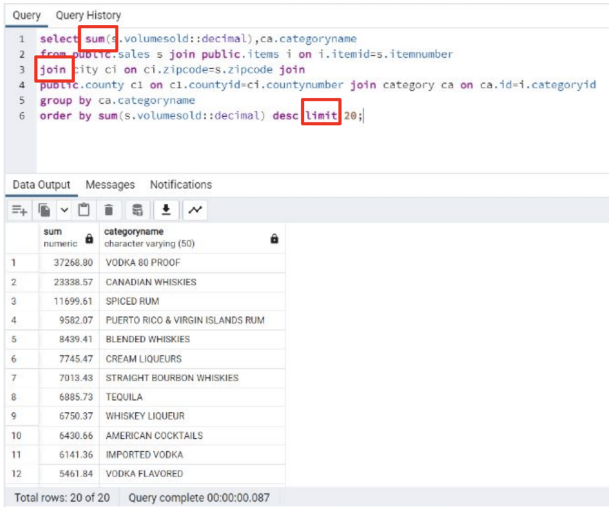


Fig. 3. Total sales of liquor of a single category for all the stores in descending order

2. Highest selling category of liquor for a particular county in a particular year

Using Typecast, we converted volume sold in varchar to integer and found the highest selling liquor in Polk county in 2012. Refer Fig. 4.

select sum(s.volumesold::decimal),ca.categoryname from public.sales s join public.items i on i.itemid=s.itemnumber join city ci on ci.zipcode=s.zipcode join public.county c1 on c1.countyid=ci.countynumber join category ca on ca.id=i.categoryid where c1.countyname='Polk' and s.date::text like '%2012%' group by ca.categoryname, c1.countyname order by sum(s.volumesold::decimal) desc limit 20;

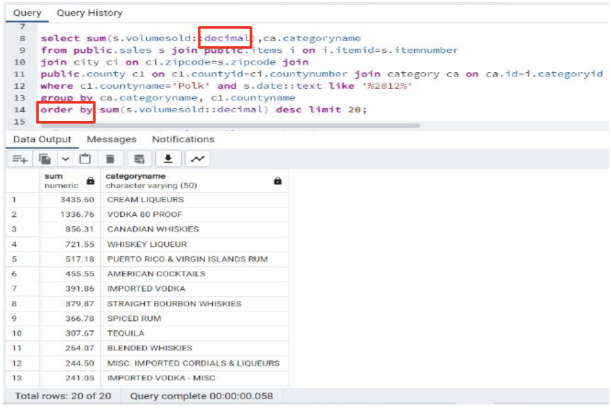


Fig. 4. Highest selling category of liquor for a particular county in a particular year

3. Black Friday sales for individual year

Using GROUP BY and HAVING clause together, we trimmed year using substring function to find Black friday sales for each year.Refer Fig. 5.

select sum(sel.bottlesold) , sel.blackfriday from (select sum(s.bottlesold) as bottlesold,substring(s.date::varchar, 1,4) as blackfriday from public.sales s where s.date::text like ’group by s.date order by sum(s.bottlesold) desc) as sel group by sel.blackfriday having sel.blackfriday::text like ’%201%’

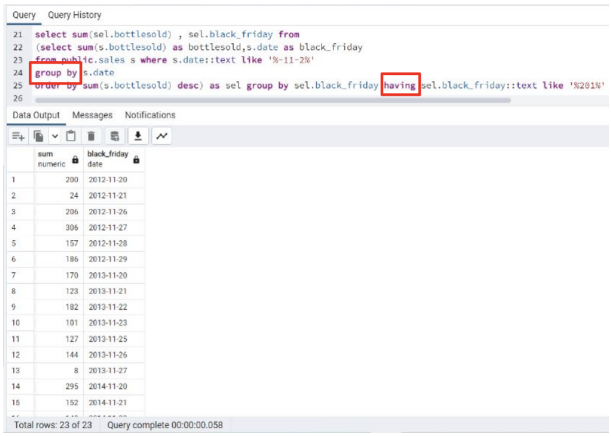


Fig. 5. Black Friday sales for individual year

4. Christmas sales for individual year

Using GROUP BY and HAVING clause together, we trimmed year using substring function to find Christmas sales for each year. Refer Fig. 6.

select sum(s.bottlesold) as bottlesold,s.date as christmas from public.sales s where s.date::text like '%-12-2%' group by s.date order by sum(s.bottlesold) desc;

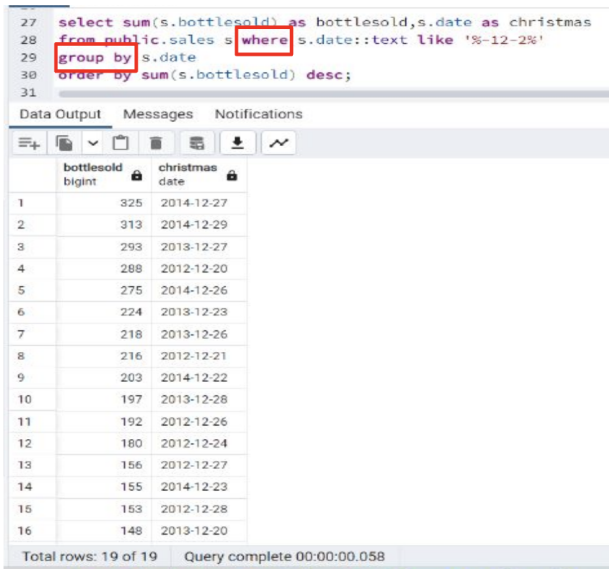


Fig. 6. Christmas sales for individual year

5. List of all the liquor categories sold by individual vendors

We used DISTINCT function to calculate the list of all liquor categories sold by individual vendors. Refer Fig. 7.

select distinct(v.vendorname), c.categoryname from items i join sales si on i.itemid=si.itemnumber join vendor v on v.vendorid=si.vendornumber join category c on c.id=i.categoryid group by v.vendorname,c.categoryname order by v.vendorname;

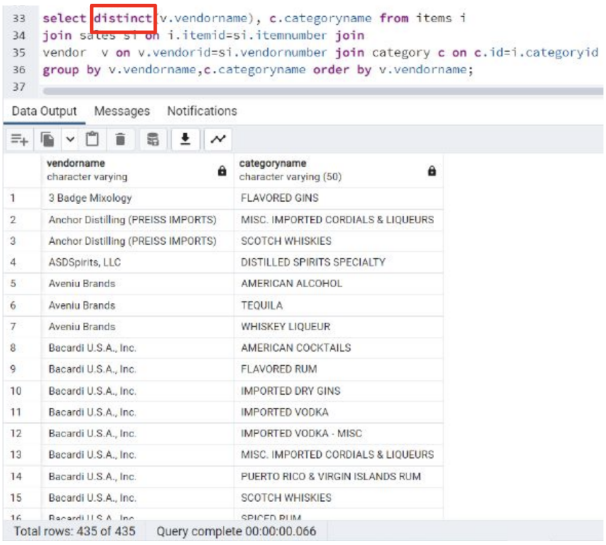


Fig. 7. List of all the liquor categories sold by individual vendors

6. List of all Store names and their addresses

Without using JOIN, we fetched the list of all Store names and their addresses. Refer Fig. 8.

select s.storename, a.addressword from store s , address a where a.storenumber=s.id;

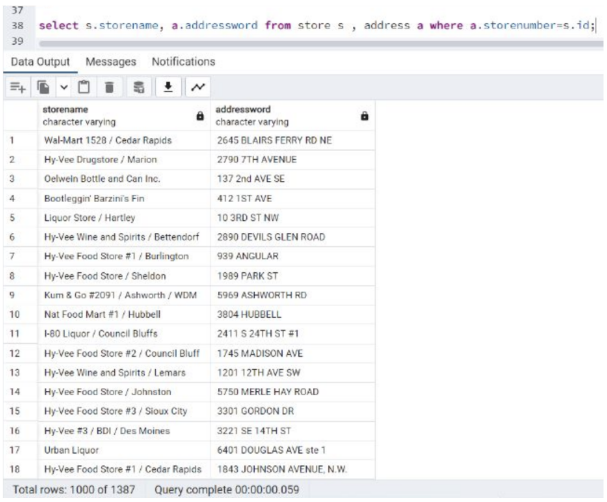


Fig. 8. List of all Store names and their addresses

7. Highest total sales store wise

Using ORDER BY we fetched highest total sales store wise. Refer Fig. 9.

SELECT si.storename, sum(s.volumesold::decimal) FROM SALES s ,store si WHERE si.id=s.storenumber group by storename order by sum(s.volumesold::decimal) desc ;

```
SELECT si.storename,  sum(s.volumesold::decimal)
FROM SALES s ,store si
WHERE  si.id=s.storenumber group by storename order by sum(s.volumesold::decimal) desc ;
```

Data Output | Messages | Notifications

	storename character varying	sum numeric
1	Costco Wholesale #788	5004.0
2	Central City 2	4081.75
3	Sam's Club 6344 / Windsor Heights	3678.0
4	Hy-Vee #3 / BDI / Des Moines	3672.00
5	Hy-Vee Wine and Spirits / Iowa City	2295.94
6	CVS Pharmacy #8633 / Bettendorf	2157.75
7	Sam's Club 8162 / Cedar Rapids	1927.5
8	Hy-Vee Food Store / Muscatine	1634.95
9	Central City Liquor, Inc.	1632.50
10	Sam's Club 8238 / Davenport	1597.5
11	Lot-A-Spirits	1519.75
12	Benz Distributing	1498.50
13	Hy-Vee Wine and Spirits #2	1395.59
14	Sam's Club 6568 / Ames	1389.0
15	Hy-Vee Wine and Spirits / Bettendorf	1361.45
16	Sam's Club 6514 / Waterloo	1351.5
17	Hy-Vee Food Store / Fort Dodge	1332.0

Total rows: 1000 of 1359 Query complete 00:00:00.096

Fig. 9. Highest total sales store wise

8. Items present in a particular category

Using this query we found the ttems present in a particular category. Refer Fig. 10.

select i.itemdescription , c.categoryname from items i join category c on i.categoryid=c.id order by c.categoryname

```
select i.itemdescription , c.categoryname from items i join
category c on i.categoryid=c.id order by c.categoryname
```

Data Output | Messages | Notifications

	itemdescription character varying	categoryname character varying (50)
1	McCormick Vodka	100 PROOF VODKA
2	Mccormick Vodka	100 PROOF VODKA
3	Dueces Wild Vodka	100 PROOF VODKA
4	Hawkeye Vodka 80 Prf Mini	100 PROOF VODKA
5	Mccormick Vodka Pet	100 PROOF VODKA
6	Smirnoff Vodka 100 Prf	100 PROOF VODKA
7	Dehner Distillery 151	AMERICAN ALCOHOL
8	Prichard's Lincoln County Lightning	AMERICAN ALCOHOL
9	Everclear Alcohol	AMERICAN ALCOHOL
10	Paramount Amaretto	AMERICAN AMARETTO
11	Di Amore Amaretto	AMERICAN AMARETTO

Total rows: 1000 of 1863 Query complete 00:00:00.061

Fig. 10. Items present in a particular category

9. Vendors present in a particular Zip code

Using this query we the vendors present in a particular zipcode. Refer Fig. 11.

select v.vendorname,s.zipcode from vendor v , sales s;

```
select v.vendorname,s.zipcode from vendor v , sales s;
```

Data Output | Messages | Notifications

	vendorname character varying	zipcode character varying
1	Diageo Americas	52402
2	Phillips Beverage Company	52402
3	Sazerac Co., Inc.	52402
4	Jim Beam Brands	52402
5	MHW Ltd	52402
6	WILLIAM GRANT AND SONS, INC.	52402
7	Laird And Company	52402
8	Pernod Ricard USA/Austin Nichols	52402
9	Brown-Forman Corporation	52402
10	E AND J GALLO WINERY	52402
11	Heaven Hill Brands	52402
12	REMY COINTREAU USA .	52402
13	Moet Hennessy USA, Inc.	52402
14	Constellation Wine Company, Inc.	52402
15	Mccormick Distilling Company	52402
16	Luxco-St Louis	52402
17	Sazerac North America	52402
18	Louisville Distilling Co	52402
19	Bacardi U.S.A. Inc.	52402

Total rows: 1000 of 1599760 Query complete 00:00:00.562

Fig. 11. Vendors present in a particular Zip code

10. Total number of stores in individual county

Here, we aggregate function COUNT to calculate total number of stores present in each county. Refer Fig. 12.

select c.countyname, count(s.storename) from store s , county c where c.countyid=s.countynumber group by c.countyname order by count(s.storename) desc;

```
select c.countyname, count(s.storename) from store s , county c where c.countyid=s.countynumber gr
order by count(s.storename) desc;
```

Data Output | Messages | Notifications

	countyname character varying (50)	count bigint
1	Polk	207
2	Linn	104
3	Black Hawk	82
4	Scott	69
5	Johnson	49
6	Pottawattamie	49
7	Story	41
8	Dubuque	37
9	Woodbury	37
10	Des Moines	22
11	Muscatine	22
12	Cerro Gordo	21
13	Webster	19
14	Clinton	17
15	Dallas	17

Total rows: 99 of 99 Query complete 00:00:00.080

Fig. 12. Total number of stores in individual county

11. Highest selling category of liquor in every county

We used PARTITION BY to categorize county by their highest selling liquor using ROW NUMBER function. Refer Fig. 13.

select * from (select sum(s.volumesold::decimal) as volume,ca.categoryname as category, c1.countyname as countyname, ROWNUMBER() OVER (PARTITION BY c1.countyname Order by sum(s.volumesold::decimal) DESC) AS Sno from public.sales s join public.items i on i.itemid=s.itemnumber join city ci on ci.zipcode=s.zipcode join public.county c1 on c1.countyid=ci.countynumber join category ca on ca.id=i.categoryid group by ca.categoryname,c1.countyname order by countyname asc ,sum(s.volumesold::decimal) desc) a1 WHERE Sno ¡=1

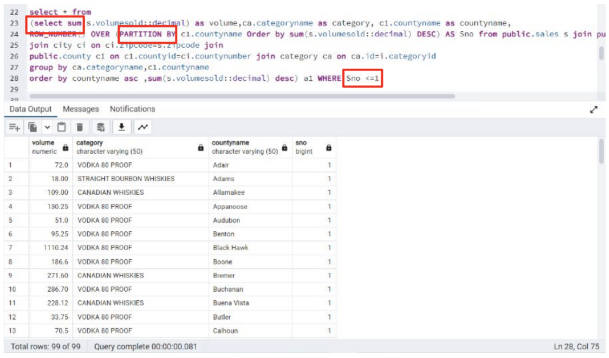


Fig. 13. Highest selling category of liquor in every county

- Query Optimization

1. Yearly liquor revenue for Iowa state

Instead of fetching daily data of each week on Black friday from each year, we, used TRIM and SUBSTRING function and subquery to calculate the retail price on Black friday from each year. Refer Fig.14.

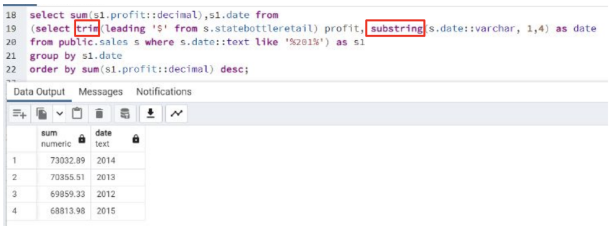
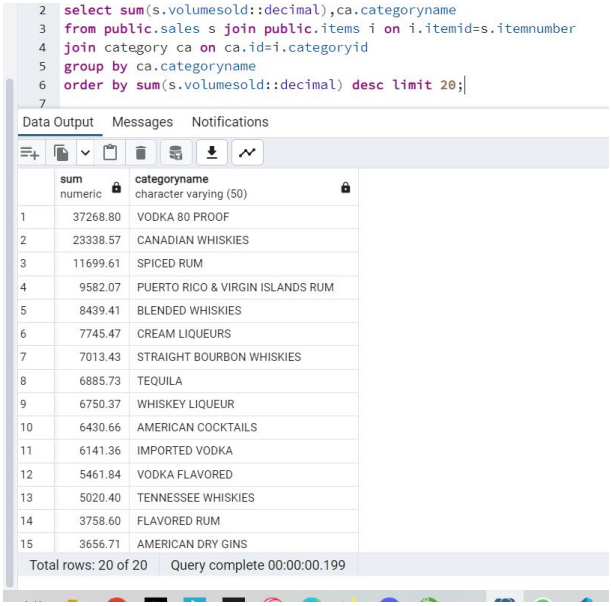


Fig. 14. Query Optimization: Yearly liquor revenue for Iowa state

2. Total sales of liquor of a single category for all the stores in descending order (Top 20)

We removed unwanted JOIN (Fig. 3.) from the query to optimize and fetch more efficient data from the table. Refer Fig. 15.



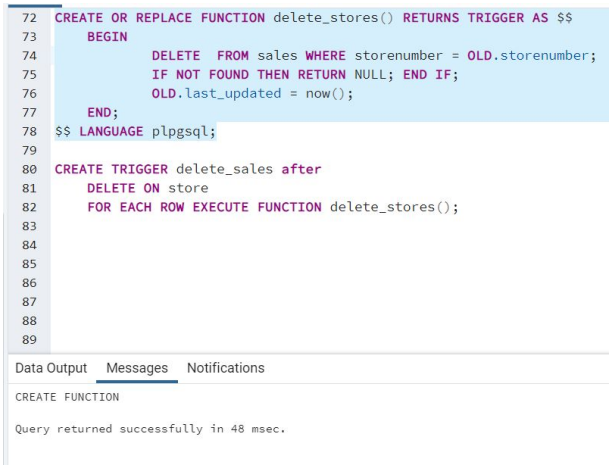
	sum numeric	categoryname character varying (50)
1	37268.80	VODKA 80 PROOF
2	23338.57	CANADIAN WHISKIES
3	11699.61	SPICED RUM
4	9582.07	PUERTO RICO & VIRGIN ISLANDS RUM
5	8439.41	BLENDED WHISKIES
6	7745.47	CREAM LIQUEURS
7	7013.43	STRAIGHT BOURBON WHISKIES
8	6885.73	TEQUILA
9	6750.37	WHISKEY LIQUEUR
10	6430.66	AMERICAN COCKTAILS
11	6141.36	IMPORTED VODKA
12	5461.84	VODKA FLAVORED
13	5020.40	TENNESSEE WHISKIES
14	3758.60	FLAVORED RUM
15	3656.71	AMERICAN DRY GINS

Total rows: 20 of 20 Query complete 00:00:00.199

Fig. 15. Query Optimization: Total sales of liquor of a single category for all the stores in descending order

- Trigger:

Using TRIGGER, we delete the respective entry from Sales table when the entry from Store is deleted. We uses TRIGGER function in TRIGGER to invoke deletesales function after deleting entry from store. Refer Fig.16.



Data Output Messages Notifications

CREATE FUNCTION

Query returned successfully in 48 msec.

Fig. 16. Trigger

VII. Cost Calculation of Query

We used, EXPLAIN function to analyse the cost of each query. Refer Fig.17.

```
7.explain analyse
SELECT si.storename,  sum(s.volumesold::decimal)
FROM SALES s ,store si
WHERE  si.id=s.storenumber group by storename order by sum(s.volumesold::decimal) desc
```

Data Output | Messages | Notifications

	QUERY PLAN text
1	Sort (cost=811.51..814.90 rows=1359 width=58) (actual time=30.371..30.426 rows=1359 loops=1)
2	Sort Key: (sum((s.volumesold)::numeric)) DESC
3	Sort Method: quicksort Memory: 154kB
4	-> HashAggregate (cost=723.80..740.78 rows=1359 width=58) (actual time=28.879..29.615 rows=1359 loops=1)
5	Group Key: si.storename
6	-> Hash Join (cost=43.21..523.83 rows=19997 width=29) (actual time=0.948..11.648 rows=19997 loops=1)
7	Hash Cond: ((s.storenumber)::text = (si.id)::text)
8	-> Seq Scan on sales s (cost=0.00..427.97 rows=19997 width=8) (actual time=0.019..2.007 rows=19997 loops=1)
9	-> Hash (cost=25.87..25.87 rows=1387 width=31) (actual time=0.913..0.913 rows=1387 loops=1)
10	Buckets: 2048 Batches: 1 Memory Usage: 102kB
11	-> Seq Scan on store si (cost=0.00..25.87 rows=1387 width=31) (actual time=0.043..0.580 rows=1387 loops=1)
12	Planning Time: 13.186 ms
13	Execution Time: 30.674 ms

Fig. 17. Trigger

REFERENCES

[1] https://www.postgresqltutorial.com/postgresql-triggers/
[2] https://www.w3resource.com/PostgreSQL/trim-function.php
[3] https://www.guru99.com/postgresql-substring.html
[4] https://www.sqlservertutorial.net/sql-server-window-functions/sql-server-rownumber-function/
[5] https://www.kaggle.com/datasets/residentmario/iowa-liquor-sales