



**National University of Computer and Emerging Sciences**

# Data Warehousing

---

## Final Project

Haroon Omer

18I-1651

**DEGREE PROGRAM:**

BSCS

**Section:**

A

**SUBJECT NAME:**

Data Warehousing

**DATE OF SUBMISSION:**

November 25, 2021

**SUBMITTED TO:**

Mr. Asif Naeem

## Contents

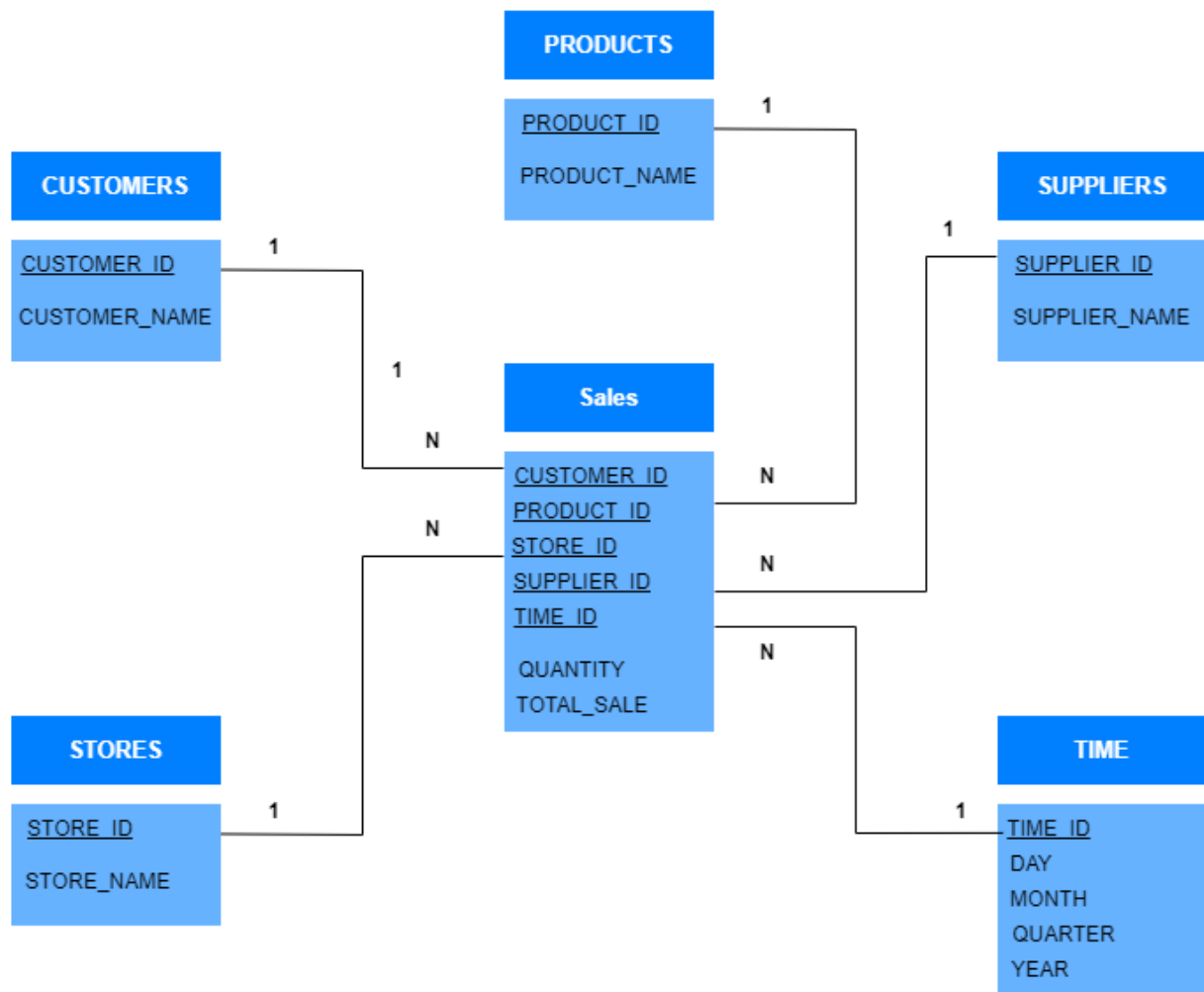
Project Overview: .....	3
Data Warehouse Schema: .....	4
MESHJOIN:.....	5
Shortcomings of MESHJOIN .....	5
MESHJOIN Algorithm .....	6
WHAT I LEARNED: .....	7
OLAP QUERIES .....	8

## Project Overview:

Every day, the world's population grows by around 200,000 people, and this trend does not appear to be slowing down anytime soon. Every day, we produce 2.5 quintillion bytes of data. That being said, we generate so much data every day, and with the population rising, there needs to be a location where this massive amount of data can be kept. Data warehouses are useful in this situation. Big enterprises that create enormous volumes of data on a regular basis require a centralized data storage location where they can store all of their data and assess their operations. As a result, in this project, I created a data warehouse for **METRO** store, then utilized the ETL process to collect operational data, change it into a format that could be stored in the warehouse, and lastly load it in the warehouse.

## Data Warehouse Schema:

There should be a schema in place to store such incoming data in order for the data to be correctly loaded into the data warehouse through the ETL process. I constructed a **star – schema** to store the incoming data stream and store it in the dimension and fact tables, respectively. The following diagram shows all of the dimensions that were developed, as well as the fact table in the center.



## MESHJOIN:

I implemented the **MESHJOIN** algorithm in the following way.

1. As the chunk size which contained incoming tuples was not fixed so I choose the chunk size of **500** tuples. These chunks were loaded one by one from **TRANSACTIONS** table as input data into the hash table with their join attribute values in the **QUEUE**.
2. Loaded next **MASTER DATA** partition into the disk buffer.
3. Looked-up each tuple from the disk buffer to the **HASH TABLE**.
4. If the tuples matched, added the required attributes into the transaction tuple from **MASTER DATA** and also calculated the price using quantity from transaction data and price from master data.
5. The new transaction tuple with new attributes was then loaded into **DW**. If dimension table already contained the information, then only fact table was updated, otherwise both were updated.
6. After completing the look-up of all disk buffer tuples into the hash table, I removed all join attribute values from the last partition of the queue along with their transaction tuples from the hash table.
7. Repeated steps 1 to 6 until I successfully loaded all the data from **TRANSACTIONS** table to DW.

### Shortcomings of MESHJOIN

- There is a dependency on the size of Master Data and Queue partitions. Mesh join assumes both are equal.
- Uses a single hash table which could make the join slightly less efficient.
- Chunk size is flexible and not fixed so Mesh join allows the user to choose any chunk size which also could cause inefficiencies.
- Algorithm slows down due to queue.
- Disk I/O dependencies also make the algorithm slower.
- Stream buffer might not work efficiently as the incoming stream of data increases in size as well as pace.

## MESHJOIN Algorithm

```
-- TD => Transactional Data
-- Q  => Queue
-- Sn => New Chunk Tuples
-- Mi => Current Master Data Partition
-- X  => Join Symbol
-- HT => Hash Table
-- Qn => Last Queue Chunk

while TD is not empty or Q is not empty:
    load Sn in Q
    load Sn in HT
    load Mi
    output = Mi X HT
    remove entries from HT
    load output in dwh
    POP Qn
```

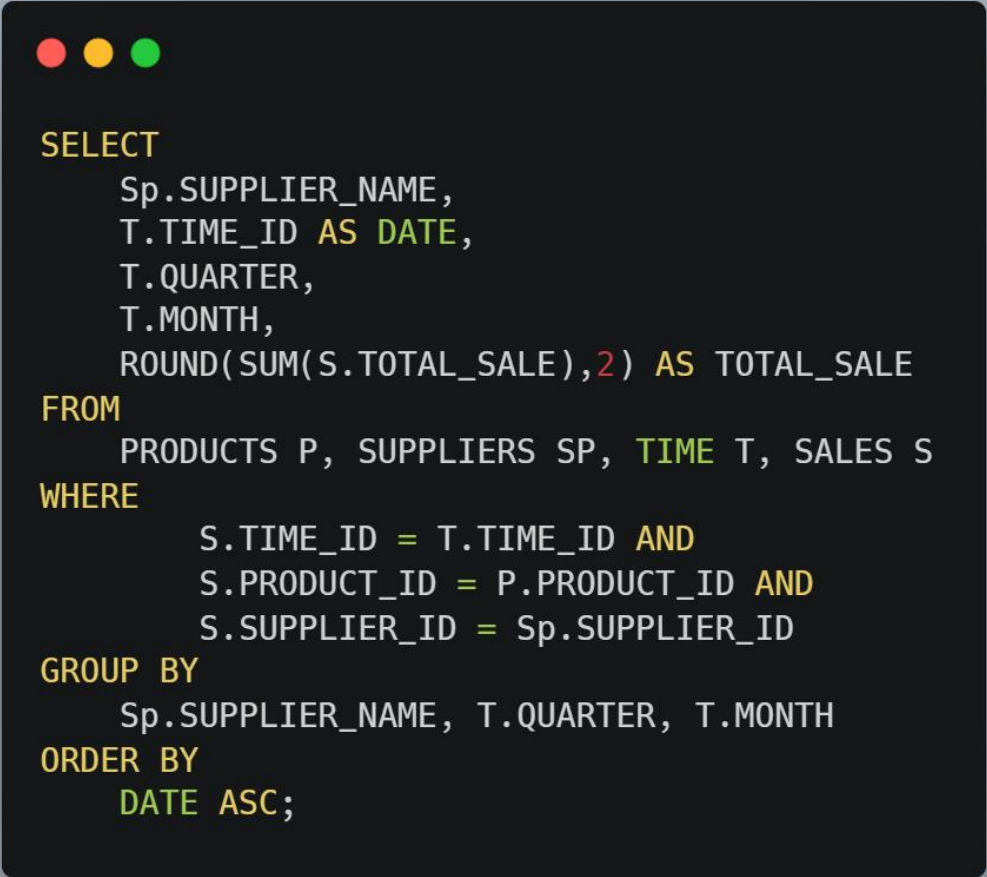
## WHAT I LEARNED:

Before beginning this project, I had no idea how an ETL process might be used to load operational data from many data bases into a warehouse in real time. This project showed me how effective MESHJOIN is in lowering I/O costs while also reducing the number of joins necessary to load data. MESHJOIN can handle any data stream that comes in. It uses a queue to load stream tuples more slowly and conducts a staggered execution of the hash table creation. MESHJOIN accounts for the disparity in access costs between the two join inputs by relying only on quick sequential scans of Relational data R and spreading the I/O cost of accessing R across several tuples of S. When compared to existing join algorithms, it performs much better. I also learnt where and when star – schema is useful, and I used MYSQL to apply it in this project. After successfully loading all of the data into the star schema, I ran a few OLAP queries and discovered a considerable reduction in query time when compared to traditional joins. After completing this assignment, the entire goal of the course became much clearer to me than it had ever been before.

## OLAP QUERIES

### Query 1

Present total sales of all products supplied by each supplier with respect to quarter and month.



```
SELECT
    Sp.SUPPLIER_NAME,
    T.TIME_ID AS DATE,
    T.QUARTER,
    T.MONTH,
    ROUND(SUM(S.TOTAL_SALE),2) AS TOTAL_SALE
FROM
    PRODUCTS P, SUPPLIERS SP, TIME T, SALES S
WHERE
    S.TIME_ID = T.TIME_ID AND
    S.PRODUCT_ID = P.PRODUCT_ID AND
    S.SUPPLIER_ID = Sp.SUPPLIER_ID
GROUP BY
    Sp.SUPPLIER_NAME, T.QUARTER, T.MONTH
ORDER BY
    DATE ASC;
```



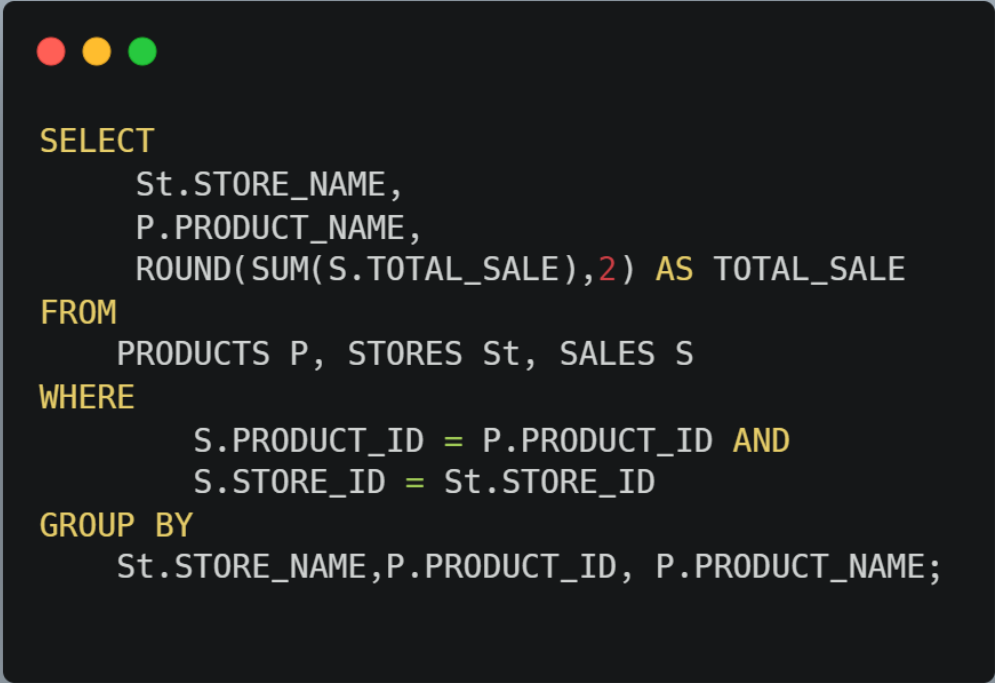
## Query 1 Results

The query returns a total of **240** rows, few of them are as below. For each supplier, the total sale of all products is calculated for each quarter and month.

Result Grid					
		Filter Rows:			
		Export:			
		Wrap Cell Content:			
	SUPPLIER_NAME	DATE	QUARTER	MONTH	TOTAL_SALE
►	Cendant Corp	2016-01-01	1	1	2069.65
	3Com Corp	2016-01-03	1	1	2987.05
	Campbell Soup Co.	2016-01-08	1	1	1046.79
	Advantica Restaurant Group	2016-01-08	1	1	2159.89
	Calpine Corp.	2016-01-11	1	1	2203.67
	The AES Corporation	2016-01-11	1	1	6473.99
	Casey's General Stores Inc.	2016-01-12	1	1	1542.01
	Coca-Cola Co	2016-01-12	1	1	2426.82
	CellStar Corp.	2016-01-13	1	1	3187.42
	CDI Corp.	2016-01-16	1	1	2944.25
	AFLAC Incorporated	2016-01-19	1	1	2051.02
	Abercrombie and Fitch Co.	2016-01-19	1	1	2561.52
	A G Edwards Inc	2016-01-22	1	1	7917.23

## Query 2

Present total sales of each product sold by each store. The output should be organized store wise and then product wise under each store.



```
SELECT
    St.STORE_NAME,
    P.PRODUCT_NAME,
    ROUND(SUM(S.TOTAL_SALE),2) AS TOTAL_SALE
FROM
    PRODUCTS P, STORES St, SALES S
WHERE
    S.PRODUCT_ID = P.PRODUCT_ID AND
    S.STORE_ID = St.STORE_ID
GROUP BY
    St.STORE_NAME,P.PRODUCT_ID, P.PRODUCT_NAME;
```

## Query 2 Results

The query returns a total of **997** rows, few of them are as below. The product **Tomatoes** occur twice with 2 different product ids', prices as well as suppliers hence why the rows are 997. Here the assumption was made that both products are different. For each supplier, the total sale of all products is calculated for each quarter and month.

Result Grid	Filter Rows:	Export:
STORE_NAME	PRODUCT_NAME	TOTAL_SALE
Queen St.	Cereal	842.7
Queen St.	Pancake / Waffle mix	585.76
Queen St.	Vegetable oil	144.84
Queen St.	Salad dressing	371.7
Queen St.	Salsa	280.82
Queen St.	Baked beans	265.29
Queen St.	Tinned meats	1295.05
Queen St.	Potatoes	105.57
Queen St.	Cinnamon	379.8
Queen St.	Ginger	762.6
Queen St.	Fruit juice	531.06
Queen St.	Carrots	164.4
Queen St.	Gravy	524.96

Assuming tomatoes are same even though they occur with 2 different ids', suppliers and prices, we get **987** rows. Few of them are as below.

Result Grid	Filter Rows:	Export:
STORE_NAME	PRODUCT_NAME	TOTAL_SALE
Queen St.	Cereal	842.7
Queen St.	Pancake / Waffle mix	585.76
Queen St.	Vegetable oil	144.84
Queen St.	Salad dressing	371.7
Queen St.	Salsa	280.82
Queen St.	Baked beans	265.29
Queen St.	Tinned meats	1295.05
Queen St.	Potatoes	105.57
Queen St.	Cinnamon	379.8
Queen St.	Ginger	762.6
Queen St.	Fruit juice	531.06
Queen St.	Carrots	164.4
Queen St.	Gravy	524.96

### Query 3

Find the 5 most popular products sold over the weekends.

```
SELECT
    S.PRODUCT_ID,
    P.PRODUCT_NAME,
    SUM(S.Quantity) AS NUMBER_OF_SALES
FROM
    PRODUCTS P, SALES S
WHERE
    S.PRODUCT_ID = P.PRODUCT_ID AND
    (
        dayname(S.TIME_ID) LIKE 'Saturday' OR
        dayname(S.TIME_ID) LIKE 'Sunday'
    )
GROUP BY
    S.PRODUCT_ID, P.PRODUCT_NAME
ORDER BY
    NUMBER_OF_SALES DESC
LIMIT 5;
```

### Query 3 Results

The query returns a total of **5** rows. We can see the top 5 products with the most sales over the weekend in terms of **quantity sold**.

	PRODUCT_ID	PRODUCT_NAME	NUMBER_OF_SALES
▶	P-1086	Tuna / Chicken	228
	P-1091	Black pepper	226
	P-1015	Apples	224
	P-1034	Fruit juice	221
	P-1011	Potatoes	216

## Query 4

Present the quarterly sales of each product for year 2016 using drill down query concept.

```
SELECT
  S.PRODUCT_ID,
  P.PRODUCT_NAME,
  ROUND(SUM(
    CASE
      WHEN (T.QUARTER = 1)
      THEN S.TOTAL_SALE END),2) AS FIRST_QUARTER_SALES,
  ROUND(SUM(
    CASE
      WHEN (T.QUARTER = 2)
      THEN S.TOTAL_SALE END),2) AS SECOND_QUARTER_SALES,
  ROUND(SUM(
    CASE
      WHEN (T.QUARTER = 3)
      THEN S.TOTAL_SALE END),2) AS THIRD_QUARTER_SALES,
  ROUND(SUM(
    CASE
      WHEN (T.QUARTER = 4)
      THEN S.TOTAL_SALE END),2) AS FOURTH_QUARTER_SALES,
  ROUND(SUM(
    CASE
      WHEN (T.QUARTER = 1 OR T.QUARTER = 2 OR T.QUARTER = 3 OR T.QUARTER = 4)
      THEN S.TOTAL_SALE END),2) AS TOTAL_YEARLY_SALES
FROM
  SALES S, TIME T, PRODUCTS P
WHERE
  S.TIME_ID = T.TIME_ID AND
  S.PRODUCT_ID = P.PRODUCT_ID
GROUP BY
  S.PRODUCT_ID
ORDER BY
  S.PRODUCT_ID;
```

## Query 4 Results

The query returns a total of **100** rows, one for each product along with its sales for each quarter and combined yearly sales.

Result Grid		Filter Rows:	Export:	Wrap Cell Content:			
	PRODUCT_ID	PRODUCT_NAME	FIRST_QUARTER_SALES	SECOND_QUARTER_SALES	THIRD_QUARTER_SALES	FOURTH_QUARTER_SALES	TOTAL_YEARLY_SALES
▶	P-1000	Asparagus	612.75	840.75	997.5	869.25	3320.25
	P-1001	Broccoli	2740.56	2866.77	2452.08	3425.7	11485.11
	P-1002	Carrots	1106.96	328.8	586.36	630.2	2652.32
	P-1003	Cauliflower	1726	2554.48	1570.66	2847.9	8699.04
	P-1004	Celery	3327.66	3277.62	4003.2	3427.74	14036.22
	P-1005	Corn	4102.56	3980.46	5543.34	2661.78	16288.14
	P-1006	Cucumbers	1524.02	1288.26	1481.92	1220.9	5515.1
	P-1007	Lettuce / Greens	2451.96	3230.36	2082.22	3308.2	11072.74
	P-1008	Mushrooms	2091.24	2024.64	2131.2	2504.16	8751.24
	P-1009	Onions	2615.1	2592.36	2546.88	4661.7	12416.04
	P-1010	Peppers	1708.16	2110.08	1896.56	2097.52	7812.32
	P-1011	Potatoes	835.38	817.02	500.31	472.77	2625.48
	P-1012	Spinach	1169.77	1671.95	1651.44	1612.12	6055.28

## Query 5

Extract total sales of each product for the first and second half of year 2016 along with its total yearly sales.

```
SELECT
  S.PRODUCT_ID,
  P.PRODUCT_NAME,
  ROUND(SUM(
    CASE
      WHEN (T.QUARTER = 1 OR T.QUARTER = 2)
      THEN S.TOTAL_SALE END),2) AS FIRST_HALF_SALES,
  ROUND(SUM(
    CASE
      WHEN (T.QUARTER = 3 OR T.QUARTER = 4)
      THEN S.TOTAL_SALE END),2) AS SECOND_HALF_SALES,
  ROUND(SUM(
    CASE
      WHEN (T.QUARTER = 1 OR T.QUARTER = 2 OR T.QUARTER = 3 OR T.QUARTER = 4)
      THEN S.TOTAL_SALE END),2) AS TOTAL_YEARLY_SALES
FROM
  SALES S, TIME T, PRODUCTS P
WHERE
  S.TIME_ID = T.TIME_ID AND
  S.PRODUCT_ID = P.PRODUCT_ID
GROUP BY
  S.PRODUCT_ID;
```

## Query 5 Results

The query returns a total of **100** rows, one for each product along with its sales for first and second half of the year and combined yearly sales.

Result Grid					
		Filter Rows:	Export:		Wrap Cell Content:
	PRODUCT_ID	PRODUCT_NAME	FIRST_HALF_SALES	SECOND_HALF_SALES	TOTAL_YEARLY_SALES
▶	P-1000	Asparagus	1453.5	1866.75	3320.25
	P-1001	Broccoli	5607.33	5877.78	11485.11
	P-1002	Carrots	1435.76	1216.56	2652.32
	P-1003	Cauliflower	4280.48	4418.56	8699.04
	P-1004	Celery	6605.28	7430.94	14036.22
	P-1005	Corn	8083.02	8205.12	16288.14
	P-1006	Cucumbers	2812.28	2702.82	5515.1
	P-1007	Lettuce / Greens	5682.32	5390.42	11072.74
	P-1008	Mushrooms	4115.88	4635.36	8751.24
	P-1009	Onions	5207.46	7208.58	12416.04
	P-1010	Peppers	3818.24	3994.08	7812.32
	P-1011	Potatoes	1652.4	973.08	2625.48
	P-1012	Spinach	2791.72	3263.56	6055.28



## Query 6

Find an anomaly in the data warehouse dataset. write a query to show the anomaly and explain the anomaly in your project report.

```
SELECT
    DISTINCT(T.PRODUCT_ID),
    M.PRODUCT_NAME,
    M.PRICE,
    M.SUPPLIER_ID
FROM
    TRANSACTIONS T, MASTERDATA M
WHERE
    M.PRODUCT_ID = T.PRODUCT_ID AND
    M.PRODUCT_NAME LIKE 'TOMATOES';
```

## Query 6 Results

The anomaly was that there was one unique product **Tomatoes** with 2 different ids, suppliers as well as prices. This affected the result set of some of the queries which was pointed out earlier in the report.

Result Grid    Filter Rows: <input type="text"/>   Export: 				
	PRODUCT_ID	PRODUCT_NAME	PRICE	SUPPLIER_ID
▶	P-1014	Tomatoes	1.79	SP-4
	P-1088	Tomatoes	19.40	SP-9



## Query 7

Create a materialized view with name “STOREANALYSIS\_MV” that presents the product-wise sales analysis for each store.

```
CREATE OR REPLACE VIEW STOREANALYSIS_MV AS (  
  SELECT  
    St.STORE_NAME,  
    P.PRODUCT_NAME,  
    ROUND(SUM(S.TOTAL_SALE),2) AS TOTAL_SALE  
  FROM  
    PRODUCTS P, STORES St, SALES S  
  WHERE  
    S.PRODUCT_ID = P.PRODUCT_ID AND  
    S.STORE_ID = St.STORE_ID  
  GROUP BY  
    St.STORE_NAME, P.PRODUCT_ID, P.PRODUCT_NAME  
);
```

## Query 7 Results

The following is the output of the view which has a total of **997** records due to the anomaly **Tomatoes** which occurs twice for 2 different ids, suppliers and prices.

Result Grid	Filter Rows:	Export:
STORE_NAME	PRODUCT_NAME	TOTAL_SALE
Queen St.	Cereal	842.7
Queen St.	Pancake / Waffle mix	585.76
Queen St.	Vegetable oil	144.84
Queen St.	Salad dressing	371.7
Queen St.	Salsa	280.82
Queen St.	Baked beans	265.29
Queen St.	Tinned meats	1295.05
Queen St.	Potatoes	105.57
Queen St.	Cinnamon	379.8
Queen St.	Ginger	762.6
Queen St.	Fruit juice	531.06
Queen St.	Carrots	164.4
Queen St.	Gravy	524.96