In [1]:

```python
import numpy as np
import pandas as pd
import psycopg2
import pgspecial
```

In [2]:

```
!docker ps
```

```
CONTAINER ID   IMAGE      COMMAND                  CREATED       STATUS        PORTS
        NAMES
3f9812e56ca3   postgres   "docker-entrypoint.s…"   2 weeks ago   Up 21 hours   0.0.0.
0:5432->5432/tcp   sales
```

In [3]:

```
# examine port connection
!nc -zv localhost 5432
```

```
found 0 associations
found 1 connections:
     1: flags=82<CONNECTED,PREFERRED>
        outif lo0
        src ::1 port 58341
        dst ::1 port 5432
        rank info not available
        TCP aux info available

Connection to localhost port 5432 [tcp/postgresql] succeeded!
```

In [4]:

```python
# connect database located on docker with psycopg2
try:
    conn = psycopg2.connect("dbname='postgres' user='postgres' host='0.0.0.0' passworc
except:
    print("I am unable to connect to the database")
```

In [5]:

```python
# Open a cursor to perform database operations
cur = conn.cursor()
```

In [6]:

```
%load_ext sql
```

In [7]:

```
# # connect database located on docker with ipython-sql
%sql postgresql://postgres:huyuan3@localhost/postgres
```

Out[7]:

```
'Connected: postgres@postgres'
```

In [8]:

```python
def query_to_df(conn, query, column_names):
    """
    Tranform a SELECT query into a pandas dataframe
    """
    cur = conn.cursor()
    try:
        cur.execute(query)
    except (Exception, psycopg2.DatabaseError) as error:
        print("Error: %s" % error)
        cur.close()
        return 1

    # Naturally we get a list of tuples
    tuples = cur.fetchall()
    cur.close()

    # We just need to turn it into a pandas dataframe
    df = pd.DataFrame(tuples, columns=column_names)
    return df
```

Table Statistics

In [41]:

```python
# load csv to pandas Dataframe
states = pd.read_csv('states.csv')
categories = pd.read_csv('categories.csv')
customers = pd.read_csv('customers.csv')
products = pd.read_csv('products.csv')
sales = pd.read_csv('sales.csv')
```

In [46]:

```
states
```

Out[46]:

	id	state_name
0	0	Alabama
1	1	Alaska
2	2	Arizona
3	3	Arkansas
4	4	California
5	5	Colorado
6	6	Connecticut
7	7	Delaware
8	8	Florida
9	9	Georgia
10	10	Hawaii
11	11	Idaho
12	12	Illinois
13	13	Indiana
14	14	Iowa
15	15	Kansas
16	16	Kentucky
17	17	Louisiana
18	18	Maine
19	19	Maryland
20	20	Massachusetts
21	21	Michigan
22	22	Minnesota
23	23	Mississippi
24	24	Missouri
25	25	Montana
26	26	Nebraska
27	27	Nevada
28	28	New Hampshire
29	29	New Jersey
30	30	New Mexico
31	31	New York
32	32	North Carolina
33	33	North Dakota
34	34	Ohio
35	35	Oklahoma
36	36	Oregon
37	37	Pennsylvania
38	38	Rhode Island
39	39	South Carolina
40	40	South Dakota
41	41	Tennessee
42	42	Texas
43	43	Utah
44	44	Vermont
45	45	Virginia
46	46	Washington
47	47	West Virginia
48	48	Wisconsin
49	49	Wyoming

In [45]:

```
categories
```

Out[45]:

	id	name	description
0	0	C0	Products in this category have properties PSET0
1	1	C1	Products in this category have properties PSET1
2	2	C2	Products in this category have properties PSET2
3	3	C3	Products in this category have properties PSET3
4	4	C4	Products in this category have properties PSET4
5	5	C5	Products in this category have properties PSET5
6	6	C6	Products in this category have properties PSET6
7	7	C7	Products in this category have properties PSET7
8	8	C8	Products in this category have properties PSET8
9	9	C9	Products in this category have properties PSET9
10	10	C10	Products in this category have properties PSET10
11	11	C11	Products in this category have properties PSET11
12	12	C12	Products in this category have properties PSET12
13	13	C13	Products in this category have properties PSET13
14	14	C14	Products in this category have properties PSET14
15	15	C15	Products in this category have properties PSET15
16	16	C16	Products in this category have properties PSET16
17	17	C17	Products in this category have properties PSET17
18	18	C18	Products in this category have properties PSET18
19	19	C19	Products in this category have properties PSET19

In [44]:

```
customers
```

Out[44]:

	id	f_name	l_name	state_id
0	0	Jwan	SMITH	45
1	1	Sonnie	SMITH	11
2	2	Thary	SMITH	26
3	3	Kwana	SMITH	41
4	4	Javonte	SMITH	41
...	...	...	...	...
887985	887985	Quenton	AALDERINK	3
887986	887986	Almon	AALDERINK	26
887987	887987	Jalal	AALDERINK	12
887988	887988	Calista	AALDERINK	36
887989	887989	Judea	AALDERINK	27

887990 rows × 4 columns

In [43]:

```
products
```

Out[43]:

	id	name	price	category_id
0	0	P0	509.18	17
1	1	P1	846.74	19
2	2	P2	846.36	12
3	3	P3	66.93	12
4	4	P4	915.68	11
...	...	...	...	...
95	95	P95	578.25	3
96	96	P96	375.21	17
97	97	P97	862.62	15
98	98	P98	540.14	19
99	99	P99	499.00	1

100 rows × 4 columns

In [42]:

```
sales
```

Out[42]:

	id	product_id	customer_id	price	quantity	discount
0	1	93	0	193.28	141	0.80
1	2	90	0	315.25	341	0.42
2	3	14	0	527.38	106	0.32
3	4	32	2	199.87	703	0.97
4	5	7	3	782.10	124	0.46
...	...	...	...	...	...	...
1776411	1776412	96	887988	878.98	224	0.46
1776412	1776413	10	887988	346.44	605	0.98
1776413	1776414	5	887989	970.33	496	0.48
1776414	1776415	27	887989	22.75	936	0.50
1776415	1776416	43	887989	81.31	585	0.66

1776416 rows × 6 columns

QUERY 1 (No indexing)

In [9]:

```python
#query 1
#1.1.Show the total sales (total quantity sold and total dollar value) for each custon
#(If customer C has made no purchases, still output C, with 0 quantity and dollars).

query1 = """
         SELECT c.id, sum(quantity) AS total_quantity, sum(price) AS total_value
         FROM sales.sales s
         FULL JOIN sales.customers c
         ON c.id = s.customer_id
         GROUP BY c.id
         """
result1 = query_to_df(conn, query1, ["customer_id","total_quantity","total_value"])
result1.to_csv("sales_query1_res.csv", index=False)
```

In [10]:

```python
res1 = pd.read_csv("sales_query1_res.csv")
res1.head(17).T
```

Out[10]:

	0	1	2	3	4	5	6	7
customer_id	124906.0	82302.00	43164.00	160019.00	134938.0	123219.00	68341.00	133638.00
total_quantity	1895.0	1172.00	946.00	1869.00	13.0	1292.00	1153.00	2479.00
total_value	1919.3	616.95	2291.68	2356.21	852.0	1462.41	1594.69	2034.18

> ### QUERY PLAN without index

In [11]:

```sql
%%sql
EXPLAIN ANALYZE
SELECT c.id, sum(quantity) AS total_quantity, sum(price) AS total_value
FROM sales.sales s
FULL JOIN sales.customers c
ON c.id = s.customer_id
GROUP BY c.id;
```

```
 * postgresql://postgres:***@localhost/postgres
15 rows affected.
```

Out[11]:

QUERY PLAN
HashAggregate (cost=203900.74..232348.43 rows=887990 width=44) (actual time=59265.537..66018.878 rows=887990 loops=1)
Group Key: c.id
Planned Partitions: 128 Batches: 129 Memory Usage: 4241kB Disk Usage: 122408kB
-> Hash Full Join (cost=29351.78..85658.05 rows=1776416 width=14) (actual time=10697.742..46135.539 rows=1953447 loops=1)
Hash Cond: (s.customer_id = c.id)
-> Seq Scan on sales s (cost=0.00..30826.16 rows=1776416 width=14) (actual time=0.035..11003.621 rows=1776416 loops=1)
-> Hash (cost=14782.90..14782.90 rows=887990 width=4) (actual time=10696.720..10696.738 rows=887990 loops=1)
Buckets: 131072 Batches: 16 Memory Usage: 2981kB
-> Seq Scan on customers c (cost=0.00..14782.90 rows=887990 width=4) (actual time=7.174..5303.537 rows=887990 loops=1)
Planning Time: 0.264 ms
JIT:
Functions: 18
Options: Inlining false, Optimization false, Expressions true, Deforming true
Timing: Generation 1.621 ms, Inlining 0.000 ms, Optimization 0.680 ms, Emission 9.679 ms, Total 11.980 ms
Execution Time: 71523.050 ms

QUERY 2 (No indexing)

In [12]:

```python
#query 2
#2.Show the total sales (total quantity sold and total dollar value) for each state.
#(If a state has 0 sales,  list it explicitly as such in the output).
query2 = """
         SELECT st.name, sum(quantity) AS total_quantity, sum(price) AS total_value
         FROM sales.sales s
         FULL JOIN sales.customers c
         ON s.customer_id = c.id
         FULL JOIN sales.states st
         ON st.id = c.state_id
         GROUP BY st.name
         """
result2 = query_to_df(conn, query2, ["state_name","total_quantity","total_value"])
result2.to_csv("sales_query2_res.csv", index=False)
```

In [13]:

```python
res2 = pd.read_csv("sales_query2_res.csv")
res2.head(17).T
```

Out[13]:

	0	1	2	3	4	5
state_name	Nevada	West Virginia	South Carolina	New Mexico	Arkansas	South Dakota
total_quantity	18004571	17836883	17598327	17760982	17655939	17709025
total_value	1.80456e+07	1.79368e+07	1.77322e+07	1.79111e+07	1.77309e+07	1.77539e+07

> ### QUERY PLAN without index

In [14]:

```sql
%%sql
EXPLAIN ANALYZE
SELECT st.name, sum(quantity) AS total_quantity, sum(price) AS total_value
FROM sales.sales s
FULL JOIN sales.customers c
ON s.customer_id = c.id
FULL JOIN sales.states st
ON st.id = c.state_id
GROUP BY st.name
```

```
 * postgresql://postgres:***@localhost/postgres
20 rows affected.
```

Out[14]:

QUERY PLAN
HashAggregate (cost=103703.01..103705.51 rows=200 width=158) (actual time=84484.176..84484.580 rows=50 loops=1)
Group Key: st.name
Batches: 1 Memory Usage: 48kB
-> Hash Full Join (cost=29373.93..90379.89 rows=1776416 width=128) (actual time=11448.221..71326.633 rows=1953447 loops=1)
Hash Cond: (c.state_id = st.id)
-> Hash Full Join (cost=29351.78..85658.05 rows=1776416 width=14) (actual time=11436.234..47187.854 rows=1953447 loops=1)
Hash Cond: (s.customer_id = c.id)
-> Seq Scan on sales s (cost=0.00..30826.16 rows=1776416 width=14) (actual time=0.119..11057.730 rows=1776416 loops=1)
-> Hash (cost=14782.90..14782.90 rows=887990 width=8) (actual time=11435.287..11435.304 rows=887990 loops=1)
Buckets: 131072 Batches: 16 Memory Usage: 3199kB
-> Seq Scan on customers c (cost=0.00..14782.90 rows=887990 width=8) (actual time=0.068..5632.895 rows=887990 loops=1)
-> Hash (cost=15.40..15.40 rows=540 width=122) (actual time=11.958..11.975 rows=50 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 11kB
-> Seq Scan on states st (cost=0.00..15.40 rows=540 width=122) (actual time=11.086..11.474 rows=50 loops=1)
Planning Time: 0.213 ms
JIT:
Functions: 23
Options: Inlining false, Optimization false, Expressions true, Deforming true
Timing: Generation 2.033 ms, Inlining 0.000 ms, Optimization 0.419 ms, Emission 10.419 ms, Total 12.871 ms
Execution Time: 84487.303 ms

QUERY 3 (Indexing)

In [15]:

```python
#query 3
#3.Show the total sales for each product, for a given customer.
#Only products that were actually bought by the given customer are listed.
#Order by dollar value. It is fine if your query hardcodes a specific customer id (fu
#Better would be to write a parameterized query (postgres function) that takes the cu
#Also great: list for EVERY customer.
#The output schema should be (pid, cust_id, total).

query3 = """
         SELECT product_id, sum(quantity) AS total_quantity, sum(price) AS total_value
         FROM sales.sales s
         WHERE customer_id = 999
         GROUP BY product_id
         ORDER BY total_value
         """
result3 = query_to_df(conn, query3, ["product_id","total_quantity","total_value"])
result3.to_csv("sales_query3_res.csv", index=False)
```

In [16]:

```python
res3 = pd.read_csv("sales_query3_res.csv")
res3
```

Out[16]:

	product_id	total_quantity	total_value
0	35	61	787.43
1	34	558	835.31
2	38	472	977.56

> ### QUERY PLAN without index

In [17]:

```sql
%%sql
EXPLAIN ANALYZE
SELECT product_id, sum(quantity) AS total_quantity, sum(price) AS total_value
FROM sales.sales s
WHERE customer_id = 999
GROUP BY product_id
ORDER BY total_value
```

```
 * postgresql://postgres:***@localhost/postgres
20 rows affected.
```

Out[17]:

QUERY PLAN
Sort (cost=23314.54..23314.55 rows=3 width=44) (actual time=81.734..87.929 rows=3 loops=1)
Sort Key: (sum(price))
Sort Method: quicksort Memory: 25kB
-> Finalize GroupAggregate (cost=23314.20..23314.51 rows=3 width=44) (actual time=81.486..87.822 rows=3 loops=1)
Group Key: product_id
-> Gather Merge (cost=23314.20..23314.46 rows=2 width=44) (actual time=81.294..87.725 rows=3 loops=1)
Workers Planned: 2
Workers Launched: 2
-> Partial GroupAggregate (cost=22314.18..22314.20 rows=1 width=44) (actual time=55.327..55.445 rows=1 loops=3)
Group Key: product_id
-> Sort (cost=22314.18..22314.18 rows=1 width=14) (actual time=55.302..55.356 rows=1 loops=3)
Sort Key: product_id
Sort Method: quicksort Memory: 25kB
Worker 0: Sort Method: quicksort Memory: 25kB
Worker 1: Sort Method: quicksort Memory: 25kB
-> Parallel Seq Scan on sales s (cost=0.00..22314.17 rows=1 width=14) (actual time=28.324..55.178 rows=1 loops=3)
Filter: (customer_id = 999)
Rows Removed by Filter: 592138
Planning Time: 0.197 ms
Execution Time: 88.063 ms

> ### Create index on customer_id

In [18]:

```sql
%%sql
CREATE INDEX customer_id_index ON sales.sales(customer_id)
```

```
 * postgresql://postgres:***@localhost/postgres
Done.
```

Out[18]:

```
[]
```

> ### QUERY PLAN with index

In [19]:

```sql
%%sql
EXPLAIN ANALYZE
SELECT product_id, sum(quantity) AS total_quantity, sum(price) AS total_value
FROM sales.sales s
WHERE customer_id = 999
GROUP BY product_id
ORDER BY total_value
```

```
 * postgresql://postgres:***@localhost/postgres
12 rows affected.
```

Out[19]:

QUERY PLAN
Sort (cost=8.60..8.60 rows=3 width=44) (actual time=0.244..0.321 rows=3 loops=1)
Sort Key: (sum(price))
Sort Method: quicksort Memory: 25kB
-> GroupAggregate (cost=8.50..8.57 rows=3 width=44) (actual time=0.141..0.240 rows=3 loops=1)
Group Key: product_id
-> Sort (cost=8.50..8.51 rows=3 width=14) (actual time=0.103..0.149 rows=3 loops=1)
Sort Key: product_id
Sort Method: quicksort Memory: 25kB
-> Index Scan using customer_id_index on sales s (cost=0.43..8.48 rows=3 width=14) (actual time=0.033..0.067 rows=3 loops=1)
Index Cond: (customer_id = 999)
Planning Time: 0.222 ms
Execution Time: 0.409 ms

> ### Analyze

Index on customer_id in sales table reduced execution time from 88.06ms to 0.41ms.

QUERY 4 (No indexing)

In [20]:

```python
#query 4
#4.Show the total sales for each product and customer. Order by dollar value.
#Compared to 3. you will return all tuples 3. returns, plus also show entries for cus
#products P such that C did not buy P (list C, P with 0 total sales).

query4 = """
         SELECT product_id, customer_id, sum(s.price) AS total_value
         FROM sales.sales s
         FULL JOIN sales.products p
         ON s.product_id = p.id
         GROUP BY product_id, customer_id
         ORDER BY total_value DESC
         """
result4 = query_to_df(conn, query4, ["product_id","customer_id","total_value"])
result4.to_csv("sales_query4_res.csv", index=False)
```

In [21]:

```python
res4 = pd.read_csv("sales_query4_res.csv")
res4.head(13).T
```

Out[21]:

	0	1	2	3	4	5	6	7	8
product_id	NaN	87.0	96.00	49.00	25.0	70.0	75.00	2.00	74.00
customer_id	NaN	2737.0	404383.00	617244.00	295440.0	878086.0	98646.00	666381.00	24896.00
total_value	NaN	2856.6	2442.66	2402.27	2358.7	2340.6	2337.77	2234.86	2234.09

> ### QUERY PLAN without index

In [22]:

```sql
%%sql
EXPLAIN ANALYZE
SELECT product_id, customer_id, sum(s.price) AS total_value
FROM sales.sales s
FULL JOIN sales.products p
ON s.product_id = p.id
GROUP BY product_id, customer_id
ORDER BY total_value DESC
```

```
 * postgresql://postgres:***@localhost/postgres
18 rows affected.
```

Out[22]:

QUERY PLAN
Sort (cost=248811.49..250242.21 rows=572286 width=40) (actual time=69339.317..80705.026 rows=1758574 loops=1)
Sort Key: (sum(s.price)) DESC
Sort Method: external merge Disk: 43056kB
-> HashAggregate (cost=153932.82..178434.20 rows=572286 width=40) (actual time=44612.690..57335.453 rows=1758574 loops=1)
Group Key: s.product_id, s.customer_id
Planned Partitions: 64 Batches: 373 Memory Usage: 4281kB Disk Usage: 61416kB
-> Hash Full Join (cost=3.25..35690.13 rows=1776416 width=14) (actual time=11.306..32706.572 rows=1776417 loops=1)
Hash Cond: (s.product_id = p.id)
-> Seq Scan on sales s (cost=0.00..30826.16 rows=1776416 width=14) (actual time=0.077..10831.063 rows=1776416 loops=1)
-> Hash (cost=2.00..2.00 rows=100 width=4) (actual time=11.189..11.206 rows=100 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 12kB
-> Seq Scan on products p (cost=0.00..2.00 rows=100 width=4) (actual time=9.434..10.215 rows=100 loops=1)
Planning Time: 0.221 ms
JIT:
Functions: 18
Options: Inlining false, Optimization false, Expressions true, Deforming true
Timing: Generation 1.905 ms, Inlining 0.000 ms, Optimization 0.643 ms, Emission 11.339 ms, Total 13.887 ms
Execution Time: 91137.638 ms

QUERY 5 (No indexing)

In [23]:

```python
#query 5
#5.Show the total sales for each product category and state.
#The output schema should be (category id, state).

query5 = """
         SELECT ca.id, st.name, sum(s.price) AS total_value
         FROM sales.sales s
         NATURAL JOIN sales.customers cu
         NATURAL JOIN sales.states st
         JOIN sales.products pr
             ON s.product_id = pr.id
         JOIN sales.categories ca
             ON ca.id = pr.category_id
         GROUP BY ca.id, st.name
         """
result5 = query_to_df(conn, query5, ["category_id","state_name","total_value"])
result5.to_csv("sales_query5_res.csv", index=False)
```

In [24]:

```python
res5 = pd.read_csv("sales_query5_res.csv")
res5.head(15).T
```

Out[24]:

	0	1	2	3	4	5	6	7
category_id	0	1	3	3	3	3	4	4
state_name	Delaware	Oklahoma	Iowa	Nevada	Pennsylvania	Wisconsin	Florida	Maryland
total_value	978.29	375.39	405.71	4.91	367.47	796	980.1	622.53

> ### QUERY PLAN without index

In [25]:

```sql
%%sql
EXPLAIN ANALYZE
SELECT ca.id, st.name, sum(s.price) AS total_value
FROM sales.sales s
NATURAL JOIN sales.customers cu
NATURAL JOIN sales.states st
JOIN sales.products pr
    ON s.product_id = pr.id
JOIN sales.categories ca
    ON ca.id = pr.category_id
GROUP BY ca.id, st.name
```

```
 * postgresql://postgres:***@localhost/postgres
25 rows affected.
```

Out[25]:

QUERY PLAN
GroupAggregate (cost=2521.79..2527.87 rows=270 width=154) (actual time=8.705..10.358 rows=49 loops=1)
Group Key: ca.id, st.name
-> Sort (cost=2521.79..2522.47 rows=270 width=128) (actual time=8.663..9.245 rows=49 loops=1)
Sort Key: ca.id, st.name
Sort Method: quicksort Memory: 28kB
-> Hash Join (cost=24.00..2510.89 rows=270 width=128) (actual time=2.474..8.252 rows=49 loops=1)
Hash Cond: (pr.category_id = ca.id)
-> Hash Join (cost=4.10..2490.27 rows=270 width=128) (actual time=2.019..6.996 rows=49 loops=1)
Hash Cond: (s.product_id = pr.id)
-> Nested Loop (cost=0.85..2486.28 rows=270 width=128) (actual time=0.249..4.275 rows=49 loops=1)
-> Nested Loop (cost=0.42..2202.35 rows=540 width=126) (actual time=0.055..2.251 rows=50 loops=1)
-> Seq Scan on states st (cost=0.00..15.40 rows=540 width=122) (actual time=0.014..0.405 rows=50 loops=1)
-> Index Only Scan using customers_pkey on customers cu (cost=0.42..4.05 rows=1 width=4) (actual time=0.011..0.011 rows=1 loops=50)
Index Cond: (id = st.id)
Heap Fetches: 0
-> Index Scan using sales_pkey on sales s (cost=0.43..0.53 rows=1 width=14) (actual time=0.013..0.013 rows=1 loops=50)
Index Cond: (id = cu.id)
-> Hash (cost=2.00..2.00 rows=100 width=8) (actual time=1.740..1.762 rows=100 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 12kB
-> Seq Scan on products pr (cost=0.00..2.00 rows=100 width=8) (actual time=0.014..0.906 rows=100 loops=1)
-> Hash (cost=14.40..14.40 rows=440 width=4) (actual time=0.425..0.448 rows=20 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 9kB
-> Seq Scan on categories ca (cost=0.00..14.40 rows=440 width=4) (actual time=0.013..0.252 rows=20 loops=1)
Planning Time: 0.722 ms
Execution Time: 10.822 ms

QUERY 6 (Indexing)

In [26]:

```python
#query 6
#6.6.For each one of the top 20 product categories (by total revenue) and top 20 custc
#(by total purchase revenue), return a tuple (top product category ID, top customer II
#It is possible that a top-20 customer spent $0 on a top-20 category. List this fact (
#(id of “comic books”, id of “jane”, 0, 0) is possible.
#Extra credit if you can list the rank of the customer and of the category:
#(cat_id, cat_rank, cust_id, cust_rank, quantity, dollar value).

query6 = """
         SELECT top_ca.id, top_cu.customer_id, sum(s.quantity), sum(s.price)
         FROM
                    (SELECT ca.id AS id, sum(s.price) AS total_value
                     FROM sales.categories ca
                     JOIN sales.products pr
                        ON ca.id = pr.category_id
                     JOIN sales.sales s
                        ON pr.id = s.product_id
                     GROUP BY ca.id
                     ORDER BY total_value DESC limit 20) AS top_ca,

                    (SELECT customer_id, sum(price) AS dollar_value
                     FROM sales.sales
                     GROUP BY customer_id
                     ORDER BY dollar_value DESC limit 20) AS top_cu,

                    sales.sales s,
                    sales.products pr

         WHERE pr.category_id = top_ca.id
                  and s.customer_id = top_cu.customer_id
                  and s.product_id = pr.id
         GROUP BY top_ca.id, top_cu.customer_id
         ORDER BY top_ca.id
    """
result6 = query_to_df(conn, query6, ["category_id","customer_id","total_quantity", "tc
result6.to_csv("sales_query6_res.csv", index=False)
```

In [27]:

```python
res6 = pd.read_csv("sales_query6_res.csv")
res6
```

Out[27]:

	category_id	customer_id	total_quantity	total_value
0	0	369359	923	970.89
1	0	606859	397	980.88
2	1	56939	745	989.69
3	1	293832	871	985.59
4	1	359264	812	988.35
...	...	...	...	...
69	18	856166	449	992.63
70	18	859589	38	884.75
71	19	15293	651	926.91
72	19	389832	602	989.52
73	19	859589	2	962.53

74 rows × 4 columns

> ### Query plan without index

In [28]:

```sql
%%sql
EXPLAIN ANALYZE
SELECT top_ca.id, top_cu.customer_id, sum(s.quantity), sum(s.price)
FROM
        (SELECT ca.id AS id, sum(s.price) AS total_value
         FROM sales.categories ca
         JOIN sales.products pr
            ON ca.id = pr.category_id
         JOIN sales.sales s
            ON pr.id = s.product_id
         GROUP BY ca.id
         ORDER BY total_value DESC limit 20) AS top_ca,

        (SELECT customer_id, sum(price) AS dollar_value
         FROM sales.sales
         GROUP BY customer_id
         ORDER BY dollar_value DESC limit 20) AS top_cu,

        sales.sales s,
        sales.products pr

WHERE pr.category_id = top_ca.id
         and s.customer_id = top_cu.customer_id
         and s.product_id = pr.id
GROUP BY top_ca.id, top_cu.customer_id
ORDER BY top_ca.id
```

```
 * postgresql://postgres:***@localhost/postgres
65 rows affected.
```

Out[28]:

QUERY PLAN
GroupAggregate (cost=114512.97..114514.52 rows=62 width=48) (actual time=55973.979..55975.995 rows=74 loops=1)
Group Key: top_ca.id, sales.customer_id
-> Sort (cost=114512.97..114513.12 rows=62 width=18) (actual time=55973.938..55974.749 rows=80 loops=1)
Sort Key: top_ca.id, sales.customer_id
Sort Method: quicksort Memory: 31kB
-> Merge Join (cost=114510.09..114511.12 rows=62 width=18) (actual time=55971.771..55973.747 rows=80 loops=1)
Merge Cond: (pr.category_id = top_ca.id)
-> Sort (cost=85183.37..85183.52 rows=62 width=18) (actual time=35709.277..35709.879 rows=80 loops=1)
Sort Key: pr.category_id
Sort Method: quicksort Memory: 31kB
-> Hash Join (cost=85011.32..85181.52 rows=62 width=18) (actual time=35705.430..35708.820 rows=80 loops=1)
Hash Cond: (s.product_id = pr.id)

```
QUERY PLAN
-> Nested Loop (cost=85008.07..85178.10 rows=62 width=18) (actual time=35670.842..35673.294 rows=80 loops=1)
-> Limit (cost=85007.65..85007.70 rows=20 width=36) (actual time=35670.789..35671.247 rows=20 loops=1)
-> Sort (cost=85007.65..86438.36 rows=572286 width=36) (actual time=35670.769..35670.904 rows=20 loops=1)
Sort Key: (sum(sales.price)) DESC
Sort Method: top-N heapsort Memory: 26kB
-> GroupAggregate (cost=0.43..69779.32 rows=572286 width=36) (actual time=0.271..30570.890 rows=710959 loops=1)
Group Key: sales.customer_id
-> Index Scan using customer_id_index on sales (cost=0.43..53743.67 rows=1776416 width=10) (actual time=0.041..12571.945 rows=1776416 loops=1)
-> Index Scan using customer_id_index on sales s (cost=0.43..8.48 rows=3 width=18) (actual time=0.011..0.038 rows=4 loops=20)
Index Cond: (customer_id = sales.customer_id)
-> Hash (cost=2.00..2.00 rows=100 width=8) (actual time=34.558..34.575 rows=100 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 12kB
-> Seq Scan on products pr (cost=0.00..2.00 rows=100 width=8) (actual time=32.607..33.522 rows=100 loops=1)
-> Sort (cost=29326.72..29326.77 rows=20 width=4) (actual time=20262.461..20262.796 rows=20 loops=1)
Sort Key: top_ca.id
Sort Method: quicksort Memory: 25kB
-> Subquery Scan on top_ca (cost=29326.04..29326.29 rows=20 width=4) (actual time=20261.677..20262.532 rows=20 loops=1)
-> Limit (cost=29326.04..29326.09 rows=20 width=36) (actual time=20261.663..20262.263 rows=20 loops=1)
-> Sort (cost=29326.04..29327.14 rows=440 width=36) (actual time=20261.649..20261.956 rows=20 loops=1)
Sort Key: (sum(s_1.price)) DESC
Sort Method: quicksort Memory: 25kB
-> Finalize GroupAggregate (cost=29199.56..29314.33 rows=440 width=36) (actual time=20260.337..20261.644 rows=20 loops=1)
Group Key: ca.id
-> Gather Merge (cost=29199.56..29302.23 rows=880 width=36) (actual time=20260.268..20261.048 rows=60 loops=1)
Workers Planned: 2
Workers Launched: 2
-> Sort (cost=28199.54..28200.64 rows=440 width=36) (actual time=20237.697..20237.904 rows=20 loops=3)
Sort Key: ca.id
Sort Method: quicksort Memory: 26kB
Worker 0: Sort Method: quicksort Memory: 26kB
Worker 1: Sort Method: quicksort Memory: 26kB
-> Partial HashAggregate (cost=28174.72..28180.22 rows=440 width=36) (actual time=20237.441..20237.642 rows=20 loops=3)
Group Key: ca.id
Batches: 1 Memory Usage: 37kB
Worker 0: Batches: 1 Memory Usage: 37kB
Worker 1: Batches: 1 Memory Usage: 37kB
-> Hash Join (cost=23.15..24473.85 rows=740173 width=10) (actual time=8.331..16826.589 rows=592139 loops=3)
Hash Cond: (pr_1.category_id = ca.id)
-> Hash Join (cost=3.25..22492.28 rows=740173 width=10) (actual time=8.040..10154.969 rows=592139 loops=3)
Hash Cond: (s_1.product_id = pr_1.id)
-> Parallel Seq Scan on sales s_1 (cost=0.00..20463.73 rows=740173 width=10) (actual time=0.013..3376.901 rows=592139 loops=3)
-> Hash (cost=2.00..2.00 rows=100 width=8) (actual time=8.003..8.019 rows=100 loops=3)
Buckets: 1024 Batches: 1 Memory Usage: 12kB
-> Seq Scan on products pr_1 (cost=0.00..2.00 rows=100 width=8) (actual time=6.777..7.407 rows=100 loops=3)
-> Hash (cost=14.40..14.40 rows=440 width=4) (actual time=0.270..0.287 rows=20 loops=3)
Buckets: 1024 Batches: 1 Memory Usage: 9kB
-> Seq Scan on categories ca (cost=0.00..14.40 rows=440 width=4) (actual time=0.024..0.153 rows=20 loops=3)
Planning Time: 0.466 ms
JIT:
Functions: 110
Options: Inlining false, Optimization false, Expressions true, Deforming true
Timing: Generation 8.315 ms, Inlining 0.000 ms, Optimization 2.057 ms, Emission 49.918 ms, Total 60.290 ms
Execution Time: 55982.377 ms
```

Creat index on porduct_id and category_id

In [29]:

```
%%sql
CREATE INDEX product_index ON sales.sales(product_id);
CREATE INDEX category_index ON sales.products(category_id);
```

```
 * postgresql://postgres:***@localhost/postgres
Done.
Done.
```

Out[29]: `[]`

> ### Query plan with index

In [30]:

```
%%sql
EXPLAIN ANALYZE
SELECT top_ca.id, top_cu.customer_id, sum(s.quantity), sum(s.price)
FROM
        (SELECT ca.id AS id, sum(s.price) AS total_value
         FROM sales.categories ca
         JOIN sales.products pr
            ON ca.id = pr.category_id
         JOIN sales.sales s
            ON pr.id = s.product_id
         GROUP BY ca.id
         ORDER BY total_value DESC limit 20) AS top_ca,

        (SELECT customer_id, sum(price) AS dollar_value
         FROM sales.sales
         GROUP BY customer_id
         ORDER BY dollar_value DESC limit 20) AS top_cu,

        sales.sales s,
        sales.products pr

WHERE pr.category_id = top_ca.id
          and s.customer_id = top_cu.customer_id
          and s.product_id = pr.id
GROUP BY top_ca.id, top_cu.customer_id
ORDER BY top_ca.id
```

```
 * postgresql://postgres:***@localhost/postgres
65 rows affected.
```

Out[30]:

```
QUERY PLAN
GroupAggregate (cost=114606.74..114608.29 rows=62 width=48) (actual time=54249.005..54250.949 rows=74 loops=1)
Group Key: top_ca.id, sales.customer_id
-> Sort (cost=114606.74..114606.90 rows=62 width=18) (actual time=54248.960..54249.803 rows=80 loops=1)
Sort Key: top_ca.id, sales.customer_id
Sort Method: quicksort Memory: 31kB
-> Nested Loop (cost=114340.12..114604.90 rows=62 width=18) (actual time=54148.095..54248.762 rows=80 loops=1)
Join Filter: (pr.id = s.product_id)
Rows Removed by Join Filter: 7920
-> Merge Join (cost=29332.05..29333.65 rows=100 width=8) (actual time=20912.381..20914.612 rows=100 loops=1)
Merge Cond: (top_ca.id = pr.category_id)
-> Sort (cost=29326.72..29326.77 rows=20 width=4) (actual time=20911.006..20911.381 rows=20 loops=1)
Sort Key: top_ca.id
Sort Method: quicksort Memory: 25kB
-> Subquery Scan on top_ca (cost=29326.04..29326.29 rows=20 width=4) (actual time=20910.289..20911.126 rows=20 loops=1)
-> Limit (cost=29326.04..29326.09 rows=20 width=36) (actual time=20910.273..20910.877 rows=20 loops=1)
-> Sort (cost=29326.04..29327.14 rows=440 width=36) (actual time=20877.199..20877.532 rows=20 loops=1)
Sort Key: (sum(s_1.price)) DESC
Sort Method: quicksort Memory: 25kB
-> Finalize GroupAggregate (cost=29199.56..29314.33 rows=440 width=36) (actual time=20875.873..20877.243 rows=20 loops=1)
Group Key: ca.id
-> Gather Merge (cost=29199.56..29302.23 rows=880 width=36) (actual time=20875.775..20876.630 rows=60 loops=1)
Workers Planned: 2
Workers Launched: 2
-> Sort (cost=28199.54..28200.64 rows=440 width=36) (actual time=20843.226..20843.440 rows=20 loops=3)
Sort Key: ca.id
Sort Method: quicksort Memory: 26kB
Worker 0: Sort Method: quicksort Memory: 26kB
Worker 1: Sort Method: quicksort Memory: 26kB
-> Partial HashAggregate (cost=28174.72..28180.22 rows=440 width=36) (actual time=20842.958..20843.162 rows=20 loops=3)
Group Key: ca.id
Batches: 1 Memory Usage: 37kB
Worker 0: Batches: 1 Memory Usage: 37kB
Worker 1: Batches: 1 Memory Usage: 37kB
-> Hash Join (cost=23.15..24473.85 rows=740173 width=10) (actual time=10.915..17328.330 rows=592139 loops=3)
Hash Cond: (pr_1.category_id = ca.id)
-> Hash Join (cost=3.25..22492.28 rows=740173 width=10) (actual time=10.534..10454.931 rows=592139 loops=3)
Hash Cond: (s_1.product_id = pr_1.id)
-> Parallel Seq Scan on sales s_1 (cost=0.00..20463.73 rows=740173 width=10) (actual time=0.019..3473.527 rows=592139 loops=3)
-> Hash (cost=2.00..2.00 rows=100 width=8) (actual time=10.482..10.499 rows=100 loops=3)
Buckets: 1024 Batches: 1 Memory Usage: 12kB
-> Seq Scan on products pr_1 (cost=0.00..2.00 rows=100 width=8) (actual time=8.851..9.634 rows=100 loops=3)
-> Hash (cost=14.40..14.40 rows=440 width=4) (actual time=0.356..0.373 rows=20 loops=3)
Buckets: 1024 Batches: 1 Memory Usage: 9kB
-> Seq Scan on categories ca (cost=0.00..14.40 rows=440 width=4) (actual time=0.031..0.175 rows=20 loops=3)
-> Sort (cost=5.32..5.57 rows=100 width=8) (actual time=1.346..1.925 rows=100 loops=1)
Sort Key: pr.category_id
Sort Method: quicksort Memory: 29kB
-> Seq Scan on products pr (cost=0.00..2.00 rows=100 width=8) (actual time=0.021..0.616 rows=100 loops=1)
-> Materialize (cost=85008.07..85178.41 rows=62 width=18) (actual time=332.339..332.842 rows=80 loops=100)
-> Nested Loop (cost=85008.07..85178.10 rows=62 width=18) (actual time=33233.321..33235.671 rows=80 loops=1)
-> Limit (cost=85007.65..85007.70 rows=20 width=36) (actual time=33233.273..33233.683 rows=20 loops=1)
-> Sort (cost=85007.65..86438.36 rows=572286 width=36) (actual time=33233.258..33233.392 rows=20 loops=1)
Sort Key: (sum(sales.price)) DESC
Sort Method: top-N heapsort Memory: 26kB
-> GroupAggregate (cost=0.43..69779.32 rows=572286 width=36) (actual time=0.077..28484.885 rows=710959 loops=1)
Group Key: sales.customer_id
-> Index Scan using customer_id_index on sales (cost=0.43..53743.67 rows=1776416 width=10) (actual time=0.023..11670.400 rows=1776416 loops=1)
-> Index Scan using customer_id_index on sales s (cost=0.43..8.48 rows=3 width=18) (actual time=0.011..0.035 rows=4 loops=20)
Index Cond: (customer_id = sales.customer_id)
Planning Time: 0.940 ms
JIT:
Functions: 108
Options: Inlining false, Optimization false, Expressions true, Deforming true
Timing: Generation 10.696 ms, Inlining 0.000 ms, Optimization 2.416 ms, Emission 56.181 ms, Total 69.294 ms
Execution Time: 54256.870 ms
```

In [22]:

```
#conn.rollback()
```