Database Design for GreenChain Insights

1. Database Implementation

1.1 Connection Details

MySQL Version: 8.0.41 **Connection Type:** Done on GCP

```
mysql> status
--------------
mysql  Ver 8.0.41-0ubuntu0.24.04.1 for Linux on x86_64 ((Ubuntu))

Connection id:          19288
Current database:       SupplyChain
Current user:           root@34.170.124.8
SSL:                    Cipher in use is ECDHE-RSA-AES128-GCM-SHA256
Current pager:          stdout
Using outfile:          ''
Using delimiter:        ;
Server version:         8.0.37-google (Google)
Protocol version:       10
Connection:             34.58.150.13 via TCP/IP
Server characterset:    utf8mb4
Db     characterset:    utf8mb4
Client characterset:    utf8mb4
Conn.  characterset:    utf8mb4
TCP port:               3306
Binary data as:         Hexadecimal
Uptime:                 23 hours 23 min 13 sec

Threads: 9  Questions: 300949  Slow queries: 6  Opens: 1062  Flush tables: 3  Open tables: 807  Queries per second avg: 3.574
--------------
```

1.2 DDL Commands for Table Creation

```
-- Industries Table
CREATE TABLE Industries (
    NAICS_Code INT PRIMARY KEY,
    Title VARCHAR(255),
    Description VARCHAR(255),
    Emissions INT
);

-- Category Table
CREATE TABLE Category (
    Category_ID INT PRIMARY KEY,
    Category_Name VARCHAR(255),
    NAICS_Code INT,
    FOREIGN KEY (NAICS_Code) REFERENCES Industries(NAICS_Code)
);
```

```
-- Users Table
CREATE TABLE Users (
    User_ID INT PRIMARY KEY,
    Username VARCHAR(255),
    Email VARCHAR(255),
    Password VARCHAR(255)
);

-- Customers Table
CREATE TABLE Customers (
    Customer_ID INT PRIMARY KEY,
    Total_Sales INT,
    City VARCHAR(255),
    State VARCHAR(255),
    Country VARCHAR(255),
    Fname VARCHAR(255),
    Lname VARCHAR(255)
);

-- Orders Table
CREATE TABLE Orders (
    Order_ID INT PRIMARY KEY,
    Order_Date VARCHAR(20),
    Customer_ID INT,
    Quantity INT,
    Category_ID INT,
    Total INT,
    FOREIGN KEY (Customer_ID) REFERENCES Customers(Customer_ID),
    FOREIGN KEY (Category_ID) REFERENCES Category(Category_ID)
);


-- Location Table
CREATE TABLE Location (
    Location_ID INT PRIMARY KEY,
    City VARCHAR(255),
    State VARCHAR(255),
    Zipcode REAL,
    Country VARCHAR(255)
);

-- Shipping Details Table
CREATE TABLE Shipping_Details (
    Order_ID INT PRIMARY KEY,
```

```
    Location_ID INT,
    Shipping_Date INT,
    Shipping_Delay INT,
    Delivery_Status VARCHAR(255),
    Late_Delivery_Risk BOOLEAN,
    Days_real INT,
    Days_scheduled INT,
    FOREIGN KEY (Order_ID) REFERENCES Orders(Order_ID),
    FOREIGN KEY (Location_ID) REFERENCES Location(Location_ID)
);
```

1.3 Data Population Verification

The following queries confirm that each table has been populated with at least 1000 rows:

SELECT 'Industries' AS Table_Name, COUNT(*) AS Row_Count FROM Industries

```
mysql> select count(*) from Industries;
+----------+
| count(*) |
+----------+
|     1017 |
+----------+
1 row in set (0.07 sec)
```

SELECT 'Category' AS Table_Name, COUNT(*) AS Row_Count FROM Category

```
mysql> select count(*) from Category;
+----------+
| count(*) |
+----------+
|       48 |
+----------+
1 row in set (0.00 sec)
```

The reason that is not 1000 entrees is that the products from the orders we have are boiled down a lot smaller number of categories within our data

SELECT 'Users' AS Table_Name, COUNT(*) AS Row_Count FROM Users

```
mysql> select count(*) from Users;
+----------+
| count(*) |
+----------+
|       30 |
+----------+
1 row in set (0.09 sec)
```

The reason that this not above 1000 entries is that this is auto-generated data we used to simulate Users using our website

SELECT 'Customers' AS Table_Name, COUNT(*) AS Row_Count FROM Customers

```
mysql> select count(*) from Customers;
+----------+
| count(*) |
+----------+
|    20652 |
+----------+
1 row in set (0.51 sec)
```

SELECT 'Orders' AS Table_Name, COUNT(*) AS Row_Count FROM Orders

```
mysql> select count(*) from Orders;
+----------+
| count(*) |
+----------+
|    65752 |
+----------+
1 row in set (0.71 sec)
```

SELECT 'Location' AS Table_Name, COUNT(*) AS Row_Count FROM Location

```
mysql> select count(*) from Location;
+----------+
| count(*) |
+----------+
|     3801 |
+----------+
1 row in set (0.06 sec)
```

SELECT 'Shipping_Details' AS Table_Name, COUNT(*) AS Row_Count FROM Shipping_Details;

```
mysql> select count(*) from Shipping_Details;
+----------+
| count(*) |
+----------+
|    65752 |
+----------+
1 row in set (0.02 sec)
```

2. Advanced Queries

Query 1: List the top 15 customers who ordered from categories linked to industries with emissions exceeding the average industry emissions.

```
SELECT
   c.Customer_ID,
   c.Fname,
   c.Lname,
   c.City,
   i.Emissions
FROM
   Customers c
JOIN
   Orders o ON c.Customer_ID = o.Customer_ID
```

```
JOIN
   Category cat ON o.Category_ID = cat.Category_ID
JOIN
   Industries i ON cat.NAICS_Code = i.NAICS_Code
WHERE
   i.Emissions > (SELECT AVG(Emissions) FROM Industries)
ORDER BY
   i.Emissions DESC
LIMIT 15;
```

Customer_ID	Fname	Lname	City	Emissions
918	Nathan	Smith	Caguas	282
7276	Pamela	Smith	Caguas	282
9198	David	Kerr	Bowling Green	282
5863	Amy	Smith	Cordova	282
2636	Christian	Smith	Caguas	282
4791	Mary	Patton	Hialeah	282
5579	Mary	Hurley	Elgin	282
2821	Mary	Williams	Santa Ana	282
662	Michelle	Moss	Caguas	282
12256	Victoria	Alvarez	Summerville	282
5375	Jeremy	Holmes	Los Angeles	282
2091	Doris	Smith	Woonsocket	282
7509	Mary	Case	Brooklyn	282
12081	Margaret	Smith	Opa Locka	282
7738	Mary	Smith	Pharr	282

15 rows in set (0.28 sec)

Query 2: Combine the top 7 industries with the highest emissions and the top 8 categories with the highest order totals where emissions exceed 100.

```
(SELECT
   i.NAICS_Code,
   i.Title,
   i.Emissions,
   NULL AS Category_Name,
   NULL AS Total_Order_Value
FROM
   Industries i
WHERE
   i.Emissions > 1000
ORDER BY
   i.Emissions DESC
LIMIT 7)
UNION
(SELECT
```

```
    i.NAICS_Code,
    i.Title,
    i.Emissions,
    c.Category_Name,
    SUM(o.Total) AS Total_Order_Value
FROM
    Industries i
JOIN
    Category c ON i.NAICS_Code = c.NAICS_Code
JOIN
    Orders o ON c.Category_ID = o.Category_ID
WHERE
    i.Emissions > 100
GROUP BY
    i.NAICS_Code, i.Title, c.Category_Name
ORDER BY
    Total_Order_Value DESC
LIMIT 8)
LIMIT 15;
```

NAICS_Code	Title	Emissions	Category_Name	Total_Order_Value
327310	Cement Manufacturing	3924	NULL	NULL
112111	Beef Cattle Ranching and Farming	2893	NULL	NULL
112112	Cattle Feedlots	2893	NULL	NULL
112130	Dual-Purpose Cattle Ranching and Farming	2893	NULL	NULL
112120	Dairy Cattle and Milk Production	1724	NULL	NULL
327410	Lime Manufacturing	1623	NULL	NULL
327420	Gypsum Product Manufacturing	1623	NULL	NULL
448210	Shoe Stores	138	Men's Footwear	1554020
451110	Sporting Goods Stores	111	Fishing	1485600
451110	Sporting Goods Stores	111	Water Sports	1444100
451110	Sporting Goods Stores	111	Camping & Hiking	1290900
316210	Footwear Manufacturing	282	Cleats	1133820
339920	Sporting and Athletic Goods Manufacturing	167	Cardio Equipment	1028430
448120	Women's Clothing Stores	138	Women's Apparel	709500
451110	Sporting Goods Stores	111	Indoor/Outdoor Games	697650

15 rows in set (0.26 sec)

Query 3: Combine the top 5 and bottom 5 categories by total order quantity, showing their associated industries.

```
(SELECT
    c.Category_ID,
    c.Category_Name,
    i.Title AS Industry_Title,
    SUM(o.Quantity) AS Total_Quantity
FROM
    Category c
JOIN
    Orders o ON c.Category_ID = o.Category_ID
```

```
JOIN
   Industries i ON c.NAICS_Code = i.NAICS_Code
GROUP BY
   c.Category_ID, c.Category_Name, i.Title
ORDER BY
   Total_Quantity DESC
LIMIT 5)
UNION
(SELECT
   c.Category_ID,
   c.Category_Name,
   i.Title AS Industry_Title,
   SUM(o.Quantity) AS Total_Quantity
FROM
   Category c
JOIN
   Orders o ON c.Category_ID = o.Category_ID
JOIN
   Industries i ON c.NAICS_Code = i.NAICS_Code
GROUP BY
   c.Category_ID, c.Category_Name, i.Title
ORDER BY
   Total_Quantity ASC
LIMIT 5)
LIMIT 15;
```

```
+-------------+----------------------+-----------------------------------------+----------------+
| Category_ID | Category_Name        | Industry_Title                          | Total_Quantity |
+-------------+----------------------+-----------------------------------------+----------------+
|          17 | Cleats               | Footwear Manufacturing                  |          18825 |
|          24 | Women's Apparel      | Women's Clothing Stores                 |          14190 |
|          46 | Indoor/Outdoor Games | Sporting Goods Stores                   |          13953 |
|          18 | Men's Footwear       | Shoe Stores                             |          11954 |
|           9 | Cardio Equipment     | Sporting and Athletic Goods Manufacturing |          10446 |
|           4 | Basketball           | Sporting Goods Stores                   |             36 |
|          34 | Golf Bags & Carts    | Sporting Goods Stores                   |             45 |
|          16 | As Seen on  TV!      | Other Direct Selling Establishments     |             63 |
|          10 | Strength Training    | Sporting and Athletic Goods Manufacturing |             73 |
|           2 | Soccer               | Sporting Goods Stores                   |            125 |
+-------------+----------------------+-----------------------------------------+----------------+
10 rows in set (0.38 sec)
```

Query 4: calculates the total quantity of items ordered and the average delivery delay, grouped by industry and country, and filters for significant order volumes

```
SELECT
   i.Title AS Industry_Title,
   l.Country,
   SUM(o.Quantity) AS Total_Quantity,
```

```
    AVG(sd.Days_real - sd.Days_scheduled) AS Avg_Delivery_Delay
FROM Industries i
JOIN Category cat ON i.NAICS_Code = cat.NAICS_Code
JOIN Orders o ON cat.Category_ID = o.Category_ID
JOIN Shipping_Details sd ON o.Order_ID = sd.Order_ID
JOIN Location l ON sd.Location_ID = l.Location_ID
WHERE o.Customer_ID IN (
    SELECT Customer_ID
    FROM Customers
    WHERE Total_Sales > (SELECT AVG(Total_Sales) FROM Customers)
)
GROUP BY i.Title, l.Country
HAVING SUM(o.Quantity) > 10
ORDER BY Avg_Delivery_Delay DESC
LIMIT 15;
```

```
+---------------------------+------------------+----------------+--------------------+
| Industry_Title            | Country          | Total_Quantity | Avg_Delivery_Delay |
+---------------------------+------------------+----------------+--------------------+
| Women's Clothing Stores   | Uzbekistán       |             11 |             2.0000 |
| Women's Clothing Stores   | Singapur         |             12 |             1.8000 |
| Sporting Goods Stores     | Barbados         |             16 |             1.7500 |
| Footwear Manufacturing    | Uruguay          |             14 |             1.6667 |
| Electronics Stores        | SudAfrica        |             11 |             1.6667 |
| Clothing Accessories Stores | México         |             12 |             1.5000 |
| Used Merchandise Stores   | Francia          |             11 |             1.5000 |
| Clothing Accessories Stores | Australia      |             14 |             1.5000 |
| Women's Clothing Stores   | Noruega          |             19 |             1.5000 |
| Sporting Goods Stores     | República Checa  |             15 |             1.4286 |
| Women's Clothing Stores   | Suiza            |             17 |             1.4286 |
| Footwear Manufacturing    | Suiza            |             16 |             1.4000 |
| Clothing Accessories Stores | Turquía        |             13 |             1.4000 |
| Footwear Manufacturing    | Haití            |             18 |             1.4000 |
| Footwear Manufacturing    | Venezuela        |             20 |             1.3750 |
+---------------------------+------------------+----------------+--------------------+
15 rows in set (1.25 sec)
```

3. Indexing

Query 1: Top 15 Customers in High-Emission Industries

Query Breakdown

- **JOINs**: Customers → Orders → Category → Industries
- **WHERE**: i.Emissions > (SELECT AVG(Emissions) FROM Industries)
- **ORDER BY**: i.Emissions DESC
- **LIMIT**: 15

Query Performance before Indexing:

```
| -> Limit: 15 row(s)  (actual time=236..236 rows=15 loops=1)
    -> Sort: i.Emissions DESC, limit input to 15 row(s) per chunk  (actual time=236..236 rows=15 loops=1)
        -> Stream results  (cost=29683 rows=23265) (actual time=58.8..234 rows=7652 loops=1)
            -> Nested loop inner join  (cost=29683 rows=23265) (actual time=58.8..230 rows=7652 loops=1)
                -> Nested loop inner join  (cost=7328 rows=23265) (actual time=37.3..54.4 rows=7652 loops=1)
                    -> Nested loop inner join  (cost=21.9 rows=16) (actual time=37..37.1 rows=2 loops=1)
                        -> Filter: (cat.NAICS_Code is not null)  (cost=5.05 rows=48) (actual time=36.4..36.4 rows=48 loops=1)
                            -> Covering index scan on cat using NAICS_Code  (cost=5.05 rows=48) (actual time=36.4..36.4 rows=48 loops=1)
                        -> Filter: (i.Emissions > (select #2))  (cost=0.251 rows=0.333) (actual time=0.0138..0.0138 rows=0.0417 loops=48)
                            -> Single-row index lookup on i using PRIMARY (NAICS_Code=cat.NAICS_Code)  (cost=0.251 rows=1) (actual time=0.00196..0.00199 rows=1 loops=48)
                            -> Select #2 (subquery in condition; run only once)
                                -> Aggregate: avg(Industries.Emissions)  (cost=205 rows=1) (actual time=0.385..0.385 rows=1 loops=1)
                                    -> Table scan on Industries  (cost=104 rows=1016) (actual time=0.0517..0.284 rows=1016 loops=1)
                    -> Filter: (o.Customer_ID is not null)  (cost=320 rows=1454) (actual time=0.31..8.39 rows=3826 loops=2)
                        -> Index lookup on o using idx_orders_category_id (Category_ID=cat.Category_ID)  (cost=320 rows=1454) (actual time=0.309..7.98 rows=3826 loops=2)
                -> Single-row index lookup on c using PRIMARY (Customer_ID=o.Customer_ID)  (cost=0.861 rows=1) (actual time=0.0228..0.0228 rows=1 loops=7652)
|
```

Cost: 29683 - High due to sequential scans and sorting a potentially large result set

Indexing Options Explored

1. **Index on Orders.Customer_ID (JOIN attribute)**

CREATE INDEX idx_orders_customer_id ON Orders(Customer_ID);

```
| -> Limit: 15 row(s)  (actual time=149..149 rows=15 loops=1)
    -> Sort: i.Emissions DESC, limit input to 15 row(s) per chunk  (actual time=149..149 rows=15 loops=1)
        -> Stream results  (cost=23491 rows=23265) (actual time=0.968..147 rows=7652 loops=1)
            -> Nested loop inner join  (cost=23491 rows=23265) (actual time=0.959..143 rows=7652 loops=1)
                -> Nested loop inner join  (cost=7328 rows=23265) (actual time=0.885..18.9 rows=7652 loops=1)
                    -> Nested loop inner join  (cost=21.9 rows=16) (actual time=0.575..0.699 rows=2 loops=1)
                        -> Filter: (cat.NAICS_Code is not null)  (cost=5.05 rows=48) (actual time=0.0585..0.0874 rows=48 loops=1)
                            -> Covering index scan on cat using idx_category_naics_code  (cost=5.05 rows=48) (actual time=0.0561..0.0787 rows=48 loops=1)
                        -> Filter: (i.Emissions > (select #2))  (cost=0.251 rows=0.333) (actual time=0.0125..0.0126 rows=0.0417 loops=48)
                            -> Single-row index lookup on i using PRIMARY (NAICS_Code=cat.NAICS_Code)  (cost=0.251 rows=1) (actual time=0.00161..0.00164 rows=1 loops=48)
                            -> Select #2 (subquery in condition; run only once)
                                -> Aggregate: avg(Industries.Emissions)  (cost=205 rows=1) (actual time=0.456..0.456 rows=1 loops=1)
                                    -> Table scan on Industries  (cost=104 rows=1016) (actual time=0.138..0.364 rows=1016 loops=1)
                    -> Filter: (o.Customer_ID is not null)  (cost=320 rows=1454) (actual time=0.281..8.82 rows=3826 loops=2)
                        -> Index lookup on o using idx_orders_category_id (Category_ID=cat.Category_ID)  (cost=320 rows=1454) (actual time=0.279..8.46 rows=3826 loops=2)
                -> Single-row index lookup on c using PRIMARY (Customer_ID=o.Customer_ID)  (cost=0.595 rows=1) (actual time=0.0159..0.016 rows=1 loops=7652)
|
```

Cost: 23491 - Less but not as efficient as Emissions due to speedup of JOIN with Customers

2. **Index on Orders.Category_ID (JOIN attribute)**

CREATE INDEX idx_orders_category_id ON Orders(Category_ID);

```
| -> Limit: 15 row(s)  (actual time=236..236 rows=15 loops=1)
    -> Sort: i.Emissions DESC, limit input to 15 row(s) per chunk  (actual time=236..236 rows=15 loops=1)
        -> Stream results  (cost=29683 rows=23265) (actual time=58.8..234 rows=7652 loops=1)
            -> Nested loop inner join  (cost=29683 rows=23265) (actual time=58.8..230 rows=7652 loops=1)
                -> Nested loop inner join  (cost=7328 rows=23265) (actual time=37.3..54.4 rows=7652 loops=1)
                    -> Nested loop inner join  (cost=21.9 rows=16) (actual time=37..37.1 rows=2 loops=1)
                        -> Filter: (cat.NAICS_Code is not null)  (cost=5.05 rows=48) (actual time=36.4..36.4 rows=48 loops=1)
                            -> Covering index scan on cat using NAICS_Code  (cost=5.05 rows=48) (actual time=36.4..36.4 rows=48 loops=1)
                        -> Filter: (i.Emissions > (select #2))  (cost=0.251 rows=0.333) (actual time=0.0138..0.0138 rows=0.0417 loops=48)
                            -> Single-row index lookup on i using PRIMARY (NAICS_Code=cat.NAICS_Code)  (cost=0.251 rows=1) (actual time=0.00196..0.00199 rows=1 loops=48)
                            -> Select #2 (subquery in condition; run only once)
                                -> Aggregate: avg(Industries.Emissions)  (cost=205 rows=1) (actual time=0.385..0.385 rows=1 loops=1)
                                    -> Table scan on Industries  (cost=104 rows=1016) (actual time=0.0517..0.284 rows=1016 loops=1)
                    -> Filter: (o.Customer_ID is not null)  (cost=320 rows=1454) (actual time=0.31..8.39 rows=3826 loops=2)
                        -> Index lookup on o using idx_orders_category_id (Category_ID=cat.Category_ID)  (cost=320 rows=1454) (actual time=0.309..7.98 rows=3826 loops=2)
                -> Single-row index lookup on c using PRIMARY (Customer_ID=o.Customer_ID)  (cost=0.861 rows=1) (actual time=0.0228..0.0228 rows=1 loops=7652)
|
```

Cost: 29683 - Stays the same

3. **Index on Category.NAICS_Code (JOIN attribute)**

CREATE INDEX idx_category_naics_code ON Category(NAICS_Code);

```
| -> Limit: 15 row(s)  (actual time=38.4..38.4 rows=15 loops=1)
    -> Sort: i.Emissions DESC, limit input to 15 row(s) per chunk  (actual time=38.4..38.4 rows=15 loops=1)
        -> Stream results  (cost=22788 rows=23265) (actual time=4.91..36.9 rows=7652 loops=1)
            -> Nested loop inner join  (cost=22788 rows=23265) (actual time=4.9..33.3 rows=7652 loops=1)
                -> Nested loop inner join  (cost=7328 rows=23265) (actual time=4.88..19.5 rows=7652 loops=1)
                    -> Nested loop inner join  (cost=21.9 rows=16) (actual time=4.61..4.7 rows=2 loops=1)
                        -> Filter: (cat.NAICS_Code is not null)  (cost=5.05 rows=48) (actual time=0.064..0.0862 rows=48 loops=1)
                            -> Covering index scan on cat using idx_category_naics_code  (cost=5.05 rows=48) (actual time=0.0563..0.0732 rows=48 loops=1)
                        -> Filter: (i.Emissions > (select #2))  (cost=0.251 rows=0.333) (actual time=0.0958..0.0958 rows=0.0417 loops=48)
                            -> Single-row index lookup on i using PRIMARY (NAICS_Code=cat.NAICS_Code)  (cost=0.251 rows=1) (actual time=0.00126..0.0013 rows=1 loops=48)
                            -> Select #2 (subquery in condition; run only once)
                                -> Aggregate: avg(Industries.Emissions)  (cost=205 rows=1) (actual time=2.56..2.57 rows=1 loops=1)
                                    -> Table scan on Industries  (cost=104 rows=1016) (actual time=0.0902..0.493 rows=1016 loops=1)
                    -> Filter: (o.Customer_ID is not null)  (cost=320 rows=1454) (actual time=0.236..7.16 rows=3826 loops=2)
                        -> Index lookup on o using idx_orders_category_id (Category_ID=cat.Category_ID)  (cost=320 rows=1454) (actual time=0.234..6.84 rows=3826 loops=2)
                -> Single-row index lookup on c using PRIMARY (Customer_ID=o.Customer_ID)  (cost=0.565 rows=1) (actual time=0.00161..0.00164 rows=1 loops=7652)
|
```

Cost: 22788 - Less but not as efficient as Emissions due to faster JOIN with Industries

4. **Index on Industries.Emissions (WHERE and ORDER BY attribute)**

CREATE INDEX idx_industries_emissions ON Industries(Emissions);

```
| -> Limit: 15 row(s)  (actual time=445..445 rows=15 loops=1)
    -> Sort: i.Emissions DESC, limit input to 15 row(s) per chunk  (actual time=445..445 rows=15 loops=1)
        -> Stream results  (cost=13810 rows=19717) (actual time=36.1..443 rows=7652 loops=1)
            -> Nested loop inner join  (cost=13810 rows=19717) (actual time=36.1..439 rows=7652 loops=1)
                -> Nested loop inner join  (cost=6789 rows=19717) (actual time=36.1..368 rows=7652 loops=1)
                    -> Nested loop inner join  (cost=21.9 rows=13.6) (actual time=19.1..19.2 rows=2 loops=1)
                        -> Filter: (cat.NAICS_Code is not null)  (cost=5.05 rows=48) (actual time=19..19.1 rows=48 loops=1)
                            -> Covering index scan on cat using idx_category_naics_code  (cost=5.05 rows=48) (actual time=19..19.1 rows=48 loops=1)
                        -> Filter: (i.Emissions > (select #2))  (cost=0.251 rows=0.282) (actual time=0.00204..0.00206 rows=0.0417 loops=48)
                            -> Single-row index lookup on i using PRIMARY (NAICS_Code=cat.NAICS_Code)  (cost=0.251 rows=1) (actual time=0.00143..0.00146 rows=1 loops=48)
                            -> Select #2 (subquery in condition; run only once)
                                -> Aggregate: avg(Industries.Emissions)  (cost=205 rows=1) (actual time=0.923..0.923 rows=1 loops=1)
                                    -> Covering index scan on Industries using idx_industries_emissions  (cost=104 rows=1016) (actual time=0.177..0.645 rows=1016 loops=1)
                    -> Filter: (o.Customer_ID is not null)  (cost=364 rows=1454) (actual time=8.61..174 rows=3826 loops=2)
                        -> Index lookup on o using idx_orders_category_id (Category_ID=cat.Category_ID)  (cost=364 rows=1454) (actual time=8.61..174 rows=3826 loops=2)
                -> Single-row index lookup on c using PRIMARY (Customer_ID=o.Customer_ID)  (cost=0.256 rows=1) (actual time=0.00906..0.00909 rows=1 loops=7652)
|
```

Cost: 13810 - Significantly less due to speedup of both subqueries' WHERE and first subquery's ORDER BY

Query 2: Top 7 Industries and Top 8 Categories by Order Value

Query Breakdown

- **Two UNIONed subqueries:**
    1. **Top 7 industries by emissions (WHERE i.Emissions > 1000).**
    2. **Top 8 categories by total order value (WHERE i.Emissions > 100, GROUP BY, ORDER BY SUM(o.Total)).**
- **JOINs: Second subquery uses Industries → Category → Orders.**
- **LIMIT: 15 overall.**

Query Performance before Indexing:

```
| -> Limit: 15 row(s)  (cost=105..107 rows=7) (actual time=951..951 rows=15 loops=1)
    -> Table scan on <union temporary>  (cost=105..107 rows=7) (actual time=951..951 rows=15 loops=1)
        -> Union materialize with deduplication  (cost=105..105 rows=7) (actual time=951..951 rows=15 loops=1)
            -> Limit table size: 15 unique row(s)
                -> Limit: 7 row(s)  (cost=104 rows=7) (actual time=0.522..0.523 rows=7 loops=1)
                    -> Sort: i.Emissions DESC, limit input to 7 row(s) per chunk  (cost=104 rows=1016) (actual time=0.521..0.522 rows=7 loops=1)
                        -> Filter: (i.Emissions > 1000)  (cost=104 rows=1016) (actual time=0.123..0.469 rows=36 loops=1)
                            -> Table scan on i  (cost=104 rows=1016) (actual time=0.115..0.412 rows=1016 loops=1)
            -> Limit table size: 15 unique row(s)
                -> Limit: 8 row(s)  (actual time=950..950 rows=8 loops=1)
                    -> Sort: Total_Order_Value DESC, limit input to 8 row(s) per chunk  (actual time=950..950 rows=8 loops=1)
                        -> Table scan on <temporary>  (actual time=950..950 rows=45 loops=1)
                            -> Aggregate using temporary table  (actual time=950..950 rows=45 loops=1)
                                -> Nested loop inner join  (cost=28.4 rows=16) (actual time=67.9..839 rows=64055 loops=1)
                                    -> Nested loop inner join  (cost=22.6 rows=16) (actual time=30.2..30.9 rows=46 loops=1)
                                        -> Filter: (c.NAICS_Code is not null)  (cost=5.8 rows=48) (actual time=30.1..30.3 rows=48 loops=1)
                                            -> Table scan on c  (cost=5.8 rows=48) (actual time=30.1..30.2 rows=48 loops=1)
                                        -> Filter: (i.Emissions > 100)  (cost=0.251 rows=0.333) (actual time=0.00954..0.0103 rows=0.958 loops=48)
                                            -> Single-row index lookup on i using PRIMARY (NAICS_Code=c.NAICS_Code)  (cost=0.251 rows=1) (actual time=0.00769..0.00795 rows=1 loops=48)
                                    -> Index lookup on o using Category_ID (Category_ID=c.Category_ID)  (cost=0.271 rows=1) (actual time=1.09..17.4 rows=1392 loops=46)
|
```

Cost: 105 - High due to aggregation and sorting in the second subquery

Indexing Options Explored

1. **Index on Industries.Emissions (WHERE and ORDER BY)**

CREATE INDEX idx_industries_emissions ON Industries(Emissions);

```
| -> Limit: 15 row(s)  (cost=24.5..26.7 rows=7) (actual time=201..201 rows=15 loops=1)
    -> Table scan on <union temporary>  (cost=24.5..26.7 rows=7) (actual time=201..201 rows=15 loops=1)
        -> Union materialize with deduplication  (cost=24.1..24.1 rows=7) (actual time=201..201 rows=15 loops=1)
            -> Limit table size: 15 unique row(s)
                -> Limit: 7 row(s)  (cost=23.4 rows=7) (actual time=0.208..0.211 rows=7 loops=1)
                    -> Index range scan on i using idx_industries_emissions over (1000 < Emissions) (reverse), with index condition: (i.Emissions > 1000)  (cost=23.4 rows=36) (actual
time=0.191..0.193 rows=7 loops=1)
            -> Limit table size: 15 unique row(s)
                -> Limit: 8 row(s)  (actual time=199..199 rows=8 loops=1)
                    -> Sort: Total_Order_Value DESC, limit input to 8 row(s) per chunk  (actual time=199..199 rows=8 loops=1)
                        -> Table scan on <temporary>  (actual time=199..199 rows=45 loops=1)
                            -> Aggregate using temporary table  (actual time=199..199 rows=45 loops=1)
                                -> Nested loop inner join  (cost=17679 rows=56198) (actual time=0.244..112 rows=64055 loops=1)
                                    -> Nested loop inner join  (cost=30.9 rows=38.6) (actual time=0.0525..0.506 rows=46 loops=1)
                                        -> Filter: (c.NAICS_Code is not null)  (cost=5.05 rows=48) (actual time=0.0405..0.173 rows=48 loops=1)
                                            -> Table scan on c  (cost=5.05 rows=48) (actual time=0.0382..0.116 rows=48 loops=1)
                                        -> Filter: (i.Emissions > 100)  (cost=0.439 rows=0.805) (actual time=0.00604..0.00642 rows=0.958 loops=48)
                                            -> Single-row index lookup on i using PRIMARY (NAICS_Code=c.NAICS_Code)  (cost=0.439 rows=1) (actual time=0.00477..0.0049 rows=1 loops=48)
                                    -> Index lookup on o using idx_orders_category_id (Category_ID=c.Category_ID)  (cost=315 rows=1454) (actual time=0.175..2.32 rows=1392 loops=46)
|
```

Cost: 24.5 - Significantly less due to speedup of both subqueries' WHERE and first subquery's ORDER BY

2. **Index on Category.NAICS_Code (JOIN)**

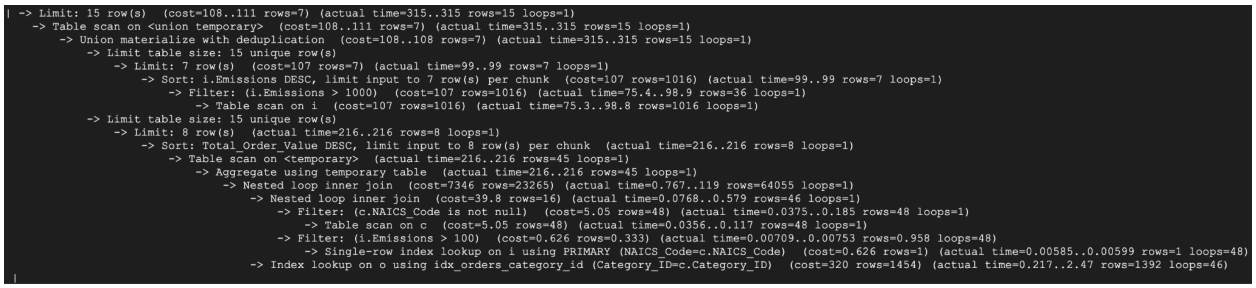
CREATE INDEX idx_category_naics_code ON Category(NAICS_Code);

```
| -> Limit: 15 row(s)  (cost=105..107 rows=7) (actual time=244..244 rows=15 loops=1)
    -> Table scan on <union temporary>  (cost=105..107 rows=7) (actual time=244..244 rows=15 loops=1)
        -> Union materialize with deduplication  (cost=105..105 rows=7) (actual time=244..244 rows=15 loops=1)
            -> Limit table size: 15 unique row(s)
                -> Limit: 7 row(s)  (cost=104 rows=7) (actual time=0.939..0.941 rows=7 loops=1)
                    -> Sort: i.Emissions DESC, limit input to 7 row(s) per chunk  (cost=104 rows=1016) (actual time=0.939..0.939 rows=7 loops=1)
                        -> Filter: (i.Emissions > 1000)  (cost=104 rows=1016) (actual time=0.101..0.898 rows=36 loops=1)
                            -> Table scan on i  (cost=104 rows=1016) (actual time=0.0909..0.651 rows=1016 loops=1)
            -> Limit table size: 15 unique row(s)
                -> Limit: 8 row(s)  (actual time=243..243 rows=8 loops=1)
                    -> Sort: Total_Order_Value DESC, limit input to 8 row(s) per chunk  (actual time=243..243 rows=8 loops=1)
                        -> Table scan on <temporary>  (actual time=243..243 rows=45 loops=1)
                            -> Aggregate using temporary table  (actual time=243..243 rows=45 loops=1)
                                -> Nested loop inner join  (cost=7328 rows=23265) (actual time=0.409..144 rows=64055 loops=1)
                                    -> Nested loop inner join  (cost=21.9 rows=16) (actual time=0.116..0.754 rows=46 loops=1)
                                        -> Filter: (c.NAICS_Code is not null)  (cost=5.05 rows=48) (actual time=0.0673..0.251 rows=48 loops=1)
                                            -> Table scan on c  (cost=5.05 rows=48) (actual time=0.0642..0.177 rows=48 loops=1)
                                        -> Filter: (i.Emissions > 100)  (cost=0.251 rows=0.333) (actual time=0.00917..0.0097 rows=0.958 loops=48)
                                            -> Single-row index lookup on i using PRIMARY (NAICS_Code=c.NAICS_Code)  (cost=0.251 rows=1) (actual time=0.0076..0.00777 rows=1 loops=48)
                                    -> Index lookup on o using idx_orders_category_id (Category_ID=c.Category_ID)  (cost=320 rows=1454) (actual time=0.22..3.01 rows=1392 loops=46)
|
```

Cost: 105 - Same

3. **Index on Orders.Category_ID (JOIN)**

CREATE INDEX idx_orders_category_id ON Orders(Category_ID);



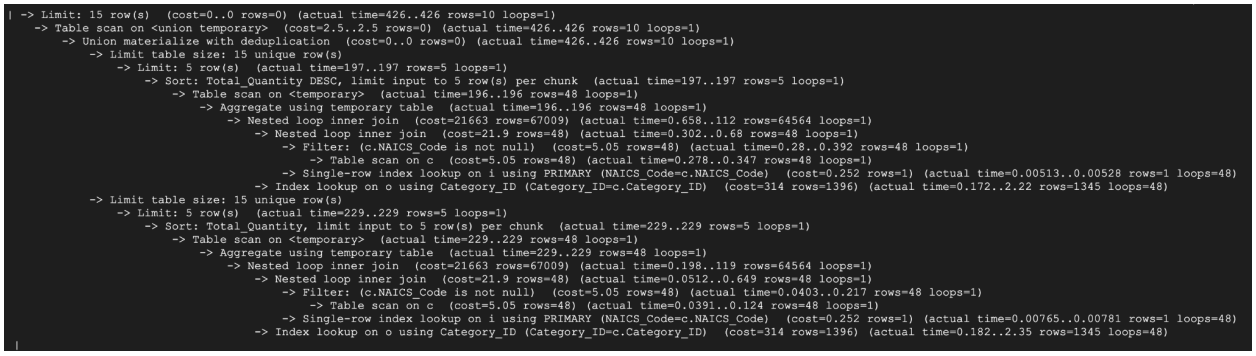
Cost: 108 - Slightly higher because Category_ID has low cardinality (few distinct values), so the index isn't selective enough.

Query 3: Top 5 and Bottom 5 Categories by Order Quantity

Query Breakdown

- **Two UNIONed subqueries: Top 5 and bottom 5 by SUM(o.Quantity).**
- **JOINs: Category → Orders → Industries.**
- **GROUP BY: Category_ID, Category_Name, Industry_Title.**
- **ORDER BY: Total_Quantity DESC and ASC.**

Query Performance Before Indexing:



Cost: 21663 - Really high due to aggregation and dual sorts increasing complexity

Indexing Options Explored

1. **Index on Orders.Category_ID (JOIN and GROUP BY)**

CREATE INDEX idx_orders_category_id ON Orders(Category_ID);

```
| -> Limit: 15 row(s)  (cost=0..0 rows=0) (actual time=1288..1288 rows=10 loops=1)
    -> Table scan on <union temporary>  (cost=2.5..2.5 rows=0) (actual time=1288..1288 rows=10 loops=1)
        -> Union materialize with deduplication  (cost=0..0 rows=0) (actual time=1288..1288 rows=10 loops=1)
            -> Limit table size: 15 unique row(s)
                -> Limit: 5 row(s)  (actual time=1058..1058 rows=5 loops=1)
                    -> Sort: Total_Quantity DESC, limit input to 5 row(s) per chunk  (actual time=1058..1058 rows=5 loops=1)
                        -> Table scan on <temporary>  (actual time=1057..1057 rows=48 loops=1)
                            -> Aggregate using temporary table  (actual time=1057..1057 rows=48 loops=1)
                                -> Nested loop inner join  (cost=24889 rows=69801) (actual time=260..951 rows=64564 loops=1)
                                    -> Nested loop inner join  (cost=40.6 rows=48) (actual time=42.9..43.7 rows=48 loops=1)
                                        -> Filter: (c.NAICS_Code is not null)  (cost=5.8 rows=48) (actual time=42.8..43.1 rows=48 loops=1)
                                            -> Table scan on c  (cost=5.8 rows=48) (actual time=42.8..43 rows=48 loops=1)
                                        -> Single-row index lookup on i using PRIMARY (NAICS_Code=c.NAICS_Code)  (cost=0.627 rows=1) (actual time=0.0103..0.0106 rows=1 loops=48)
                                    -> Index lookup on o using idx_orders_category_id (Category_ID=c.Category_ID)  (cost=375 rows=1454) (actual time=7.28..18.8 rows=1345 loops=48)
            -> Limit table size: 15 unique row(s)
                -> Limit: 5 row(s)  (actual time=229..229 rows=5 loops=1)
                    -> Sort: Total_Quantity, limit input to 5 row(s) per chunk  (actual time=229..229 rows=5 loops=1)
                        -> Table scan on <temporary>  (actual time=228..229 rows=48 loops=1)
                            -> Aggregate using temporary table  (actual time=228..228 rows=48 loops=1)
                                -> Nested loop inner join  (cost=24889 rows=69801) (actual time=0.523..129 rows=64564 loops=1)
                                    -> Nested loop inner join  (cost=40.6 rows=48) (actual time=0.196..0.647 rows=48 loops=1)
                                        -> Filter: (c.NAICS_Code is not null)  (cost=5.8 rows=48) (actual time=0.143..0.283 rows=48 loops=1)
                                            -> Table scan on c  (cost=5.8 rows=48) (actual time=0.141..0.218 rows=48 loops=1)
                                        -> Single-row index lookup on i using PRIMARY (NAICS_Code=c.NAICS_Code)  (cost=0.627 rows=1) (actual time=0.0065..0.00663 rows=1 loops=48)
                                    -> Index lookup on o using idx_orders_category_id (Category_ID=c.Category_ID)  (cost=375 rows=1454) (actual time=0.209..2.58 rows=1345 loops=48)
|
```

Cost: 24889 - Higher because Category_ID has low cardinality (few distinct values), so the index isn't selective enough.

2. **Index on Category.NAICS_Code (JOIN)**

CREATE INDEX idx_category_naics_code ON Category(NAICS_Code);

```
| -> Limit: 15 row(s)  (cost=0..0 rows=0) (actual time=435..435 rows=10 loops=1)
    -> Table scan on <union temporary>  (cost=2.5..2.5 rows=0) (actual time=435..435 rows=10 loops=1)
        -> Union materialize with deduplication  (cost=0..0 rows=0) (actual time=435..435 rows=10 loops=1)
            -> Limit table size: 15 unique row(s)
                -> Limit: 5 row(s)  (actual time=233..233 rows=5 loops=1)
                    -> Sort: Total_Quantity DESC, limit input to 5 row(s) per chunk  (actual time=233..233 rows=5 loops=1)
                        -> Table scan on <temporary>  (actual time=232..232 rows=48 loops=1)
                            -> Aggregate using temporary table  (actual time=232..232 rows=48 loops=1)
                                -> Nested loop inner join  (cost=21942 rows=69801) (actual time=0.37..154 rows=64564 loops=1)
                                    -> Nested loop inner join  (cost=21.9 rows=48) (actual time=0.0762..0.481 rows=48 loops=1)
                                        -> Filter: (c.NAICS_Code is not null)  (cost=5.05 rows=48) (actual time=0.0537..0.185 rows=48 loops=1)
                                            -> Table scan on c  (cost=5.05 rows=48) (actual time=0.0526..0.143 rows=48 loops=1)
                                        -> Single-row index lookup on i using PRIMARY (NAICS_Code=c.NAICS_Code)  (cost=0.252 rows=1) (actual time=0.00454..0.00466 rows=1 loops=48)
                                    -> Index lookup on o using idx_orders_category_id (Category_ID=c.Category_ID)  (cost=314 rows=1454) (actual time=0.174..3.1 rows=1345 loops=48)
            -> Limit table size: 15 unique row(s)
                -> Limit: 5 row(s)  (actual time=203..203 rows=5 loops=1)
                    -> Sort: Total_Quantity, limit input to 5 row(s) per chunk  (actual time=203..203 rows=5 loops=1)
                        -> Table scan on <temporary>  (actual time=202..202 rows=48 loops=1)
                            -> Aggregate using temporary table  (actual time=202..202 rows=48 loops=1)
                                -> Nested loop inner join  (cost=21942 rows=69801) (actual time=0.208..110 rows=64564 loops=1)
                                    -> Nested loop inner join  (cost=21.9 rows=48) (actual time=0.0578..0.427 rows=48 loops=1)
                                        -> Filter: (c.NAICS_Code is not null)  (cost=5.05 rows=48) (actual time=0.045..0.167 rows=48 loops=1)
                                            -> Table scan on c  (cost=5.05 rows=48) (actual time=0.0435..0.118 rows=48 loops=1)
                                        -> Single-row index lookup on i using PRIMARY (NAICS_Code=c.NAICS_Code)  (cost=0.252 rows=1) (actual time=0.0045..0.00461 rows=1 loops=48)
                                    -> Index lookup on o using idx_orders_category_id (Category_ID=c.Category_ID)  (cost=314 rows=1454) (actual time=0.161..2.18 rows=1345 loops=48)
|
```

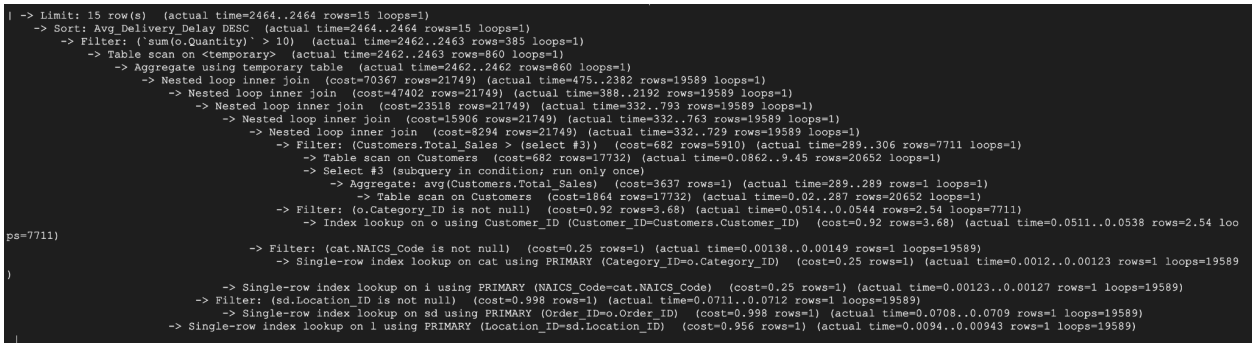
Cost: 21942 - Significantly less due to faster JOIN with Industries

Query 4: Total Quantity and Avg Delivery Delay by Industry and Country

Query Breakdown

- **JOINs: Industries → Category → Orders → Shipping_Details → Location.**
- **WHERE: Subquery on Customers.Total_Sales.**
- **GROUP BY: Industry_Title, Country.**
- **HAVING: SUM(o.Quantity) > 10.**
- **ORDER BY: Avg_Delivery_Delay DESC**

Query Performance before Indexing:

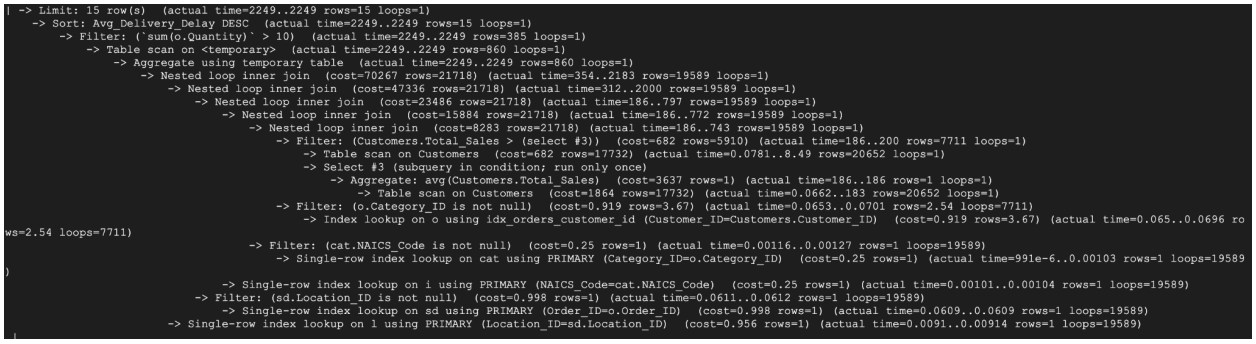


Cost: 70367 - Really high due to multiple JOINs and aggregation

Indexing Options Explored

1. **Index on Orders.Customer_ID (WHERE subquery)**

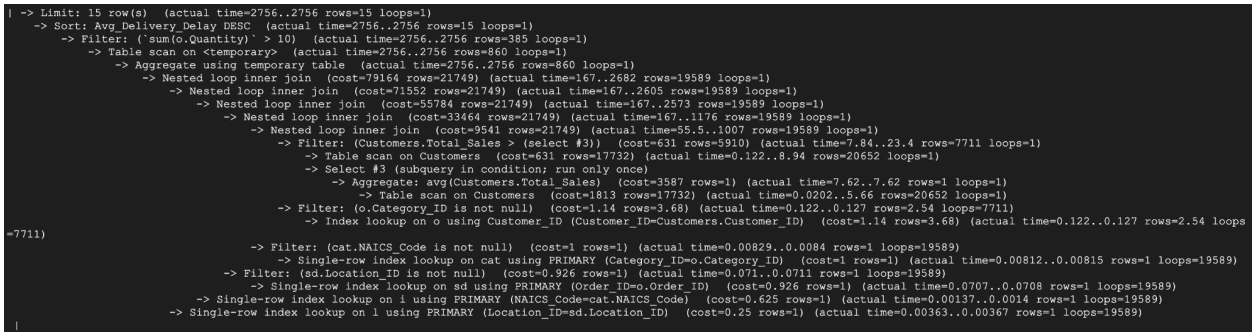
CREATE INDEX idx_orders_customer_id ON Orders(Customer_ID);



Cost: 70267 - Slightly better as there is a speedup of subquery filtering.

2. **Index on Orders.Category_ID (JOIN)**

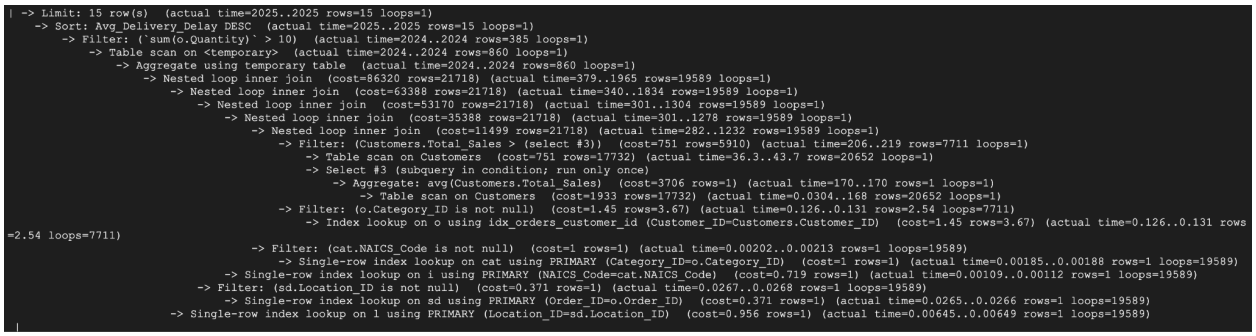
CREATE INDEX idx_orders_category_id ON Orders(Category_ID);



Cost: 79164 - Worse due to the Orders table being small or the selectivity of the JOIN is low (many rows match), so the index scan can be more expensive.

3. **Index on Shipping_Details.Order_ID (JOIN)**

CREATE INDEX idx_shipping_details_order_id ON Shipping_Details(Order_ID);



Cost: 86320 - Worse because with the Shipping_Details.Order_ID index, the optimizer might switch to an index scan to join with Orders. If many rows from Shipping_Details are accessed (e.g., due to a non-selective WHERE clause), an index scan can be more expensive than a sequential scan

4. Final Choices:

For query 1, we chose to index by Emissions because there is a significant speedup of both subqueries' WHERE and first subquery's ORDER BY causing the cost to decrease by 50%

For query 2, we also chose to index by Emissions because there is a significant speedup of both subqueries' WHERE and first subquery's ORDER BY causing the cost to decrease by 80%

For query 3, we chose to index by NAICS_Code because the JOIN with Industries causes a minor speedup and is more cost efficient

For query 4, we chose to index by Customer_Id because of the minor cost decrease due to the speedup of subquery filtering.