Assignment 1: SQL in PostgreSQL vs Databricks

Individual Questions

Question 1

Determine the top 3 airlines with the most aircrafts

Results

Small Flight Dataset

Rank	Name	Count of aircrafts
1	Delta Air lines Inc.	12612
2	Southwest Airlines Co.	12039
3	United Airlines	8638

Databricks Query Time: 6.9 seconds
PostgreSQL Query Time: 0.206 seconds

Medium Flight Dataset

Rank	Name	Count of aircrafts
1	Delta Air lines Inc.	124028
2	Southwest Airlines Co.	118738
3	United Airlines	86596

Databricks Query time: 10.3 seconds
PostgreSQL Query Time: 0.731 seconds

Comparison

- For both Flights_small and Flights_medium dataset, the query time of PostgreSQL is much faster than Databricks.
- For the small flight data, PostgreSQL took 97% less time than Databricks
- For the medium flight data, PostgreSQL took 93% less time than Databricks

Query Description

Select airline names and the count the aircraft names as two columns. Join Aircrafts dataset with Flights datasets on same tail numbers first, then join with Airlines dataset on same carrier code. The last step is group by the airline name and sort the count of aircrafts in descending order.

Question 3

Determine which model of aircraft has visited which state the most

Results

Small Dataset

Aircraft Model	State	Count of flights
MD-88	GA	1764

Databricks Query time: 2.55 seconds
PostgreSQL Query time: 0.576 seconds

Medium Dataset

Aircraft Model	State	Count of flights
MD-88	GA	17050

Databricks Query time: 10.92 seconds
PostgreSQL Query time: 1.693 seconds

Comparison

- PostgreSQL runs the same query significantly faster than on Databricks.
- For the small flight dataset, PostgreSQL took 77.4% less time than Databricks
- For the medium flight dataset, PostgreSQL took 84.4 less time than Databricks

Query Description

By joining `flights_small`, `aircrafts`, and `airports`, we can find the number of occurrence for each model arriving to every state and we store it in a temporary table `count_table`. Based on the `count_table`, we find the maximum out of all states for each model. Joining the two tables, filtered by where count equals to max, and ordered by counts will give which model of aircrafts have visited which state the most.

Question 4

Determine the top two airlines with the longest total distance flown

Results

Small Dataset

Airline Name	Total Distance
American Airlines Inc.	20596564
Delta Air Lines Inc.	18477490

Databricks Query Time: 2.65 seconds
PostgreSQL Query Time: 0.407 seconds

Medium Dataset

Airline Name	Total Distance
American Airlines Inc.	204889921
Delta Air Lines Inc.	181494531

Databricks Query Time: 8.95 seconds
PostgreSQL Query Time: 1.698 seconds

Comparison

- PostgreSQL completed the query significantly faster than Databricks for both the small and medium flight dataset
- For the small flight dataset, PostgreSQL took 85% less time than Databricks
- For the medium flight dataset, PostgreSQL took 81% less time than Databricks

Query Description

The query left joined the flight's table with the airline table based on the carrier code attribute. Then it is grouped by Airline name and ordered from biggest sum to smallest where the top 2 airline names are selected based on the sum of the total distance.

Team Questions

Question 1

Results

Small Dataset

Airline Name	Total airline delay	Manufacturer	Model	Cumulative lateness of model	Percentage of total lateness for airline
American Airlines Inc.	533968	MCDONNELL DOUGLAS	DC-9-82 (MD-82)	311035	58.2%

Medium Dataset

Airline Name	Total airline delay	Manufacturer	Model	Cumulative lateness of model	Percentage of total lateness for airline
American Airlines Inc.	4979368	MCDONNELL DOUGLAS	DC-9-82 (MD-82)	2894716	58.1%

Query Description

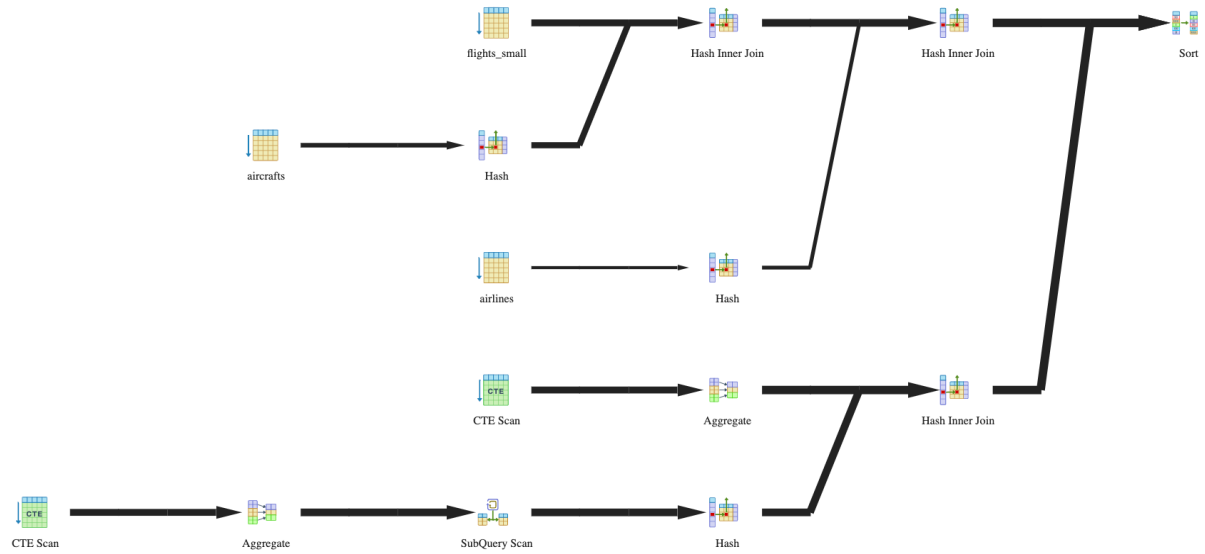
We first utilized the SUBSTRING() method to calculate arrival/departure delay time by minutes for all small flights and stored in a temporary table `delays`. With the `delay` table, we could aggregate the sum of delay time grouped by `airline` and `airline, model` as two additional subqueries, `airline_model_delay` and `airline_delay`. After that, a join operation between these two new subqueries

can project information such as airline total accumulated delay, airline delay by models, and the percentage of delays by each model.

Execution plan on Databricks:

```
== Physical Plan ==
AdaptiveSparkPlan (75)
+- == Final Plan ==
   TakeOrderedAndProject (42)
   +- * Project (41)
      +- * BroadcastHashJoin Inner BuildRight (40)
         :- * HashAggregate (19)
         :  +- AQEShuffleRead (18)
         :     +- ShuffleQueryStage (17), Statistics(sizeInBytes=46.3 KiB, rowCount=519, isRuntime=true)
         :        +- Exchange (16)
         :           +- * HashAggregate (15)
         :              +- * Project (14)
         :                 +- * BroadcastHashJoin Inner BuildRight (13)
         :                    :- * Project (8)
         :                    :  +- * BroadcastHashJoin Inner BuildRight (7)
         :                    :     :- * Filter (2)
         :                    :     :  +- Scan csv spark_catalog.default.flights_small (1)
         :                    :     +- ShuffleQueryStage (6), Statistics(sizeInBytes=28.5 KiB, rowCount=542, isRuntime=true)
         :                    :        +- Exchange (5)
         :                    :           +- * Filter (4)
         :                    :              +- Scan csv spark_catalog.default.airlines (3)
         :                    +- ShuffleQueryStage (12), Statistics(sizeInBytes=272.0 KiB, rowCount=4.48E+3, isRuntime=true)
         :                       +- Exchange (11)
         :                          +- * Filter (10)
         :                             +- Scan csv spark_catalog.default.aircrafts (9)
         +- ShuffleQueryStage (39), Statistics(sizeInBytes=1000.0 B, rowCount=22, isRuntime=true)
            +- Exchange (38)
               +- * HashAggregate (37)
                  +- AQEShuffleRead (36)
                     +- ShuffleQueryStage (35), Statistics(sizeInBytes=2.4 KiB, rowCount=45, isRuntime=true)
                        +- Exchange (34)
                           +- * HashAggregate (33)
                              +- * Project (32)
                                 +- * BroadcastHashJoin Inner BuildRight (31)
                                    :- * Project (25)
                                    :  +- * BroadcastHashJoin Inner BuildRight (24)
                                    :     :- * Filter (21)
                                    :     :  +- Scan csv spark_catalog.default.flights_small (20)
                                    :     +- ShuffleQueryStage (23), Statistics(sizeInBytes=28.5 KiB, rowCount=542, isRuntime=true)
                                    :        +- ReusedExchange (22)
                                    +- ShuffleQueryStage (30), Statistics(sizeInBytes=105.0 KiB, rowCount=4.48E+3, isRuntime=true)
                                       +- Exchange (29)
                                          +- * Project (28)
                                             +- * Filter (27)
                                                +- Scan csv spark_catalog.default.aircrafts (26)
```

Execution plan on PostgreSQL:



	#	Node	Rows Plan
	1.	→ Sort (cost=16841.09..16857.83 rows=6694 width=144)	6694
	2.	→ Hash Inner Join (cost=204.39..11420.22 rows=66944 width=44) **Hash Cond**: (f.carrier_code = a.carrier_code)	66944
	3.	→ Hash Inner Join (cost=187.15..5126.98 rows=66944 width=40) **Hash Cond**: ((f.tail_number)::text = (a2.tailnum)::text)	66944
	4.	→ Seq Scan on flights_small as f (cost=0..3605.1 rows=177410 width=29)	177410
	5.	→ Hash (cost=124.29..124.29 rows=5029 width=24)	5029
	6.	→ Seq Scan on aircrafts as a2 (cost=0..124.29 rows=5029 width=24) **Filter**: (model IS NOT NULL)	5029
	7.	→ Hash (cost=10.44..10.44 rows=544 width=22)	544
	8.	→ Seq Scan on airlines as a (cost=0..10.44 rows=544 width=22)	544
	9.	→ Hash Inner Join (cost=4693.08..4995.51 rows=6694 width=144) **Hash Cond**: ((delays.name)::text = (airline_delay.name)::text)	6694
	10.	→ Aggregate (cost=2510.4..2577.34 rows=6694 width=112)	6694
	11.	→ CTE Scan (cost=0..1338.88 rows=66944 width=104)	66944
	12.	→ Hash (cost=2180.18..2180.18 rows=200 width=40)	200
	13.	→ Subquery Scan (cost=2175.68..2180.18 rows=200 width=40)	200
	14.	→ Aggregate (cost=2175.68..2178.18 rows=200 width=40)	200
	15.	→ CTE Scan (cost=0..1338.88 rows=66944 width=40)	66944

Comparison

Both execution plan in PostgreSQL and Databricks scan the whole dataset first. Then the first main difference between PostgreSQL and Databricks is PostgreSQL aggregates the dataset much earlier than Databricks. The second one is PostgreSQL makes use of subquery, the table ‘delay’ does not need to be access repeated, but spark will execute the table ‘delay’ every time when needed, thats why the execution

speed for PostgreSQL is much faster than Databricks. There are so many Shuffle Query Stages in spark, which is used for redistributing data across partitions, but it can be a computationally expensive operation, so it leads to high network and I/O overhead. The last significant difference is PostgreSQL uses hash inner join to combine several datasets, but Databricks use broadcast hash inner join to do that. Broadcast Hash Join is used when one of the DataFrames is small enough to fit in memory and can be broadcasted to all worker nodes, avoiding shuffling the larger DataFrame. Hash Join is used when both DataFrames are too large to fit in memory, and shuffling is required for the join operation. So for this part, broadcast hash inner join in spark is better than normal hash inner join.

Question 2

Assumptions

We assumed all flights happened on the same day, as well as a departure/arrival time of 2400, is equivalent to 0000 for the query.

Results

Small Flight Dataset

Airport Code	Airport name	Number of long flights per airport	Airline with most long flights	Number of long flights per airline	Average long flight duration (Minutes)
ATL	William B Hartsfield-Atlanta Intl	9697	Delta Air Lines Inc.	5935	162.95
DEN	Denver Intl	5399	United Airlines	2198	154.42
DFW	Dallas-Fort Worth International	8044	American Airlines Inc.	4594	174.15
LAX	Los Angeles International	6004	United Airlines	1096	156.96
ORD	Chicago O'Hare International	9226	United Airlines	3201	195.87

Medium Flight Dataset

Airport Code	Airport name	Number of long flights per airport	Airline with most long flights	Number of long flights per airline	Average long flight duration (Minutes)
ATL	William B Hartsfield-Atlanta Intl	95761	Delta Air Lines Inc.	57764	163.21
DEN	Denver Intl	54833	United Airlines	22106	154.30
DFW	Dallas-Fort Worth International	79375	American Airlines Inc.	46000	175.90
LAX	Los Angeles International	58556	United Airlines	10876	155.14
ORD	Chicago O'Hare International	90825	United Airlines	31288	192.88

Query Description

The query had five temporary tables, called common table expressions (CTE) made in order to store results which are then combined at the end of the query. The five CTE contain aggregates of:

1. Values that are mostly duplicates of either the small or medium dataset. The changes occur when the arrival and departure times recorded as 2400 are changed to 0000, and when the numbers which contained 3 integers were added with a 0 at the front (ie. 630 becomes 0630). This is because the format from the time function does not accept 2400 as a value and needs 4 integers.
2. A filter for selecting flights and their corresponding duration in minutes where it is longer than the average flight time.
3. The count of long flights per airport, which is ordered so that only the top 5 counts were selected
4. A count of long flights of every airline name at each of the top 5 airports
5. A filter that selects the airline at each airport that had the most counts of long flights

Finally, at the end of the query, all the information that was required was selected from the CTEs by joining through the airport code.

Query Statistics

The task asked us to compare the query execution plans for different data sizes, i.e. of either PostgreSQL or Spark/Databricks on the small and the medium (or large) datasets. Hence, the comparison here will be made for PostgreSQL query plans.

Due to the graphic query plan being too wide, the query plan analysis for this query has been carried out according to the recommendation in the following post:
https://edstem.org/au/courses/11494/discussion/1350760.

<table>
<tr><th></th><th>Statistics Per Node Type</th><th>Cost Summary</th></tr>
<tr><td>Small Flights Dataset</td><td>
<table>
<tr><th>Node type</th><th>Count</th></tr>
<tr><td>Aggregate</td><td>9</td></tr>
<tr><td>CTE Scan</td><td>9</td></tr>
<tr><td>Hash</td><td>16</td></tr>
<tr><td>Hash Inner Join</td><td>9</td></tr>
<tr><td>Hash Left Join</td><td>6</td></tr>
<tr><td>Hash Right Join</td><td>1</td></tr>
<tr><td>Index Scan</td><td>1</td></tr>
<tr><td>Limit</td><td>2</td></tr>
<tr><td>Materialize</td><td>1</td></tr>
<tr><td>Merge Left Join</td><td>1</td></tr>
<tr><td>Nested Loop Left Join</td><td>1</td></tr>
<tr><td>Seq Scan</td><td>13</td></tr>
<tr><td>Sort</td><td>6</td></tr>
<tr><td>Subquery Scan</td><td>3</td></tr>
<tr><td>Window Aggregate</td><td>1</td></tr>
</table>
</td><td>Planning Time: 1.792 ms
Execution Time: 2417.795 ms</td></tr>
</table>

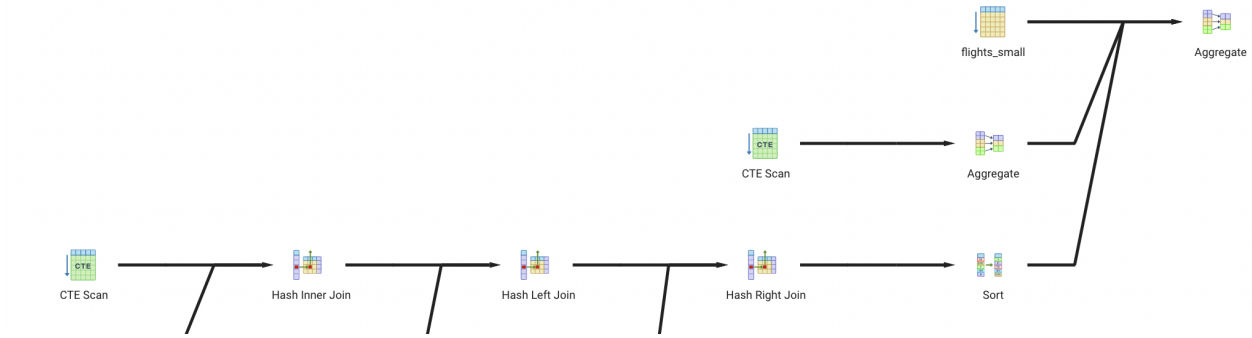
Medium Flights Dataset		Planning Time: 1.821 ms Execution Time: 24529.926 ms

Node type	Count
Aggregate	9
CTE Scan	9
Hash	16
Hash Inner Join	7
Hash Left Join	8
Hash Right Join	1
Index Only Scan	2
Limit	2
Merge Inner Join	2
Seq Scan	12
Sort	7
Subquery Scan	3
Window Aggregate	1

Sub-sample of Query Plan

Small Flight Dataset

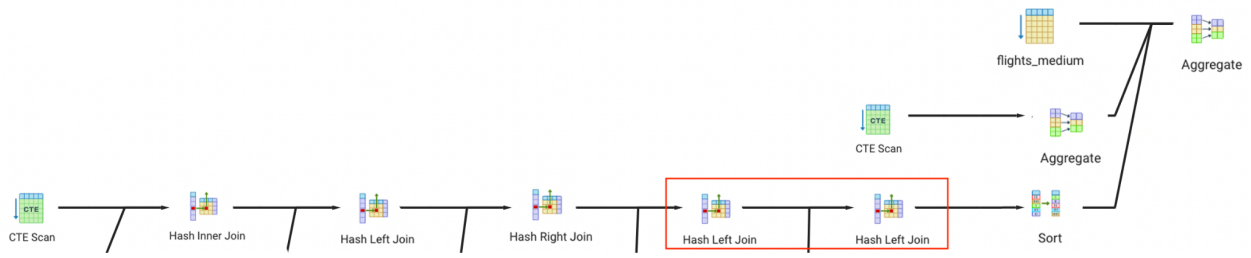
The query plan below shows the first 3 levels of the query plan.



#	Node
1.	→ Aggregate
2.	→ Seq Scan on flights_small as flights_small
3.	→ Aggregate
4.	→ CTE Scan
5.	→ Sort
6.	→ Hash Right Join **Hash Cond**: (p_1.iata = p_2.iata) (cost=64137.85..75537.55 rows=88 width=64) (actual time=2066.585..2409.010 rows=22877 loops=1)
7.	→ Hash Left Join **Hash Cond**: (s.carrier_code = l.carrier_code) (cost=129.16..11306.21 rows=59137 width=12) (actual time=269.622..599.056 rows=97316 loops=1)
8.	→ Hash Inner Join **Hash Cond**: ((s.destination)::text = p_1.iata) (cost=111.92..10475.84 rows=59137 width=44) (actual time=269.490..578.006 rows=97316 loops=1)
9.	→ CTE Scan **Filter**: ((((date_part('epoch'::text, to_timestamp(('2000-01-01 '::text || lpad((actual_arrival_time)::text, 4, '0'::text)), 'yyyy-MM-dd HH24mi'::text)) - date_part('epoch'::text, to_timestamp(('2000-01-01 '::text || lpad ((actual_departure_time)::text, 4, '0'::text)), 'yyyy-MM-dd HH24mi'::text))) / '60'::double precision) > $1)

Medium Flight Dataset

The graphics below shows the difference, which is boxed in a red rectangle, in the query plan for flight_medium from flight_small within the first three levels of the query plan.



#	Node
1.	→ Aggregate (rows=5 loops=1)
2.	→ Seq Scan on flights_medium as flights_medium (rows=1771093 loops=1)
3.	→ Aggregate (rows=1 loops=1)
4.	→ CTE Scan (rows=1771093 loops=1)
5.	→ Sort (rows=225752 loops=1)
6.	→ Hash Left Join (rows=225752 loops=1) Hash Cond: (p_2.iata = p.iata) (cost=629773.67..742912.47 rows=875 width=64) (actual time=23859.511..30545.149 rows=225752 loops=1)
7.	→ Hash Left Join (rows=225752 loops=1) Hash Cond: (p_2.iata = a.iata) (cost=629661.75..742798.25 rows=875 width=47) (actual time=23857.805..30452.898 rows=225752 loops=1)
8.	→ Hash Right Join (rows=225752 loops=1) Hash Cond: (p_1.iata = p_2.iata) (cost=294323.04..407455.69 rows=875 width=20) (actual time=11664.575..18173.879 rows=225752 loops=1)
9.	→ Hash Left Join (rows=970845 loops=1) Hash Cond: (s.carrier_code = l.carrier_code) (cost=129.16..111710.15 rows=590364 width=12) (actual time=2754.597..9060.749 rows=970845 loops=1)
10.	→ Hash Inner Join (rows=970845 loops=1) Hash Cond: ((s.destination)::text = p_1.iata) (cost=111.92..103575.41 rows=590364 width=44) (actual time=2754.440..8700.131 rows=970845 loops=1)
11.	→ CTE Scan (rows=971126 loops=1) Filter: (((date_part('epoch'::text, to_timestamp(('2000-01-01 '::text || lpad((actual_arrival_time)::text, 4, '0'::text)), 'yyyy-MM-dd HH24mi'::text)) - date_part('epoch'::text, to_timestamp(('2000-01-01 '::text || lpad((actual_departure_time)::text, 4, '0'::text)), 'yyyy-MM-dd HH24mi'::text))) / '60'::double precision) > $1) Rows Removed by Filter: 799967

Comparison

Looking at the visual query plan made on PgAdmin4 for different dataset sizes, despite sharing the same SQL code, the medium dataset had two extra hash joins on the third level of the plan. These extra hash joins are added to the front, right after the sort operation. Moreover, the two hash left joins are based on joining the airport code. Otherwise, the first three level of the query plan is the same as the small flight dataset. Unsurprisingly, it can be seen from the node statistic summary that while the amount of total hash joins is 16 for both the medium and the small flight datasets the spread of left and inner hash join varies. Both datasets have one right join, but PgAdmin seems to prefer using more inner hash joins with smaller datasets, as evident by the 9 inner hash joins on the smaller dataset compared to the 7 on the medium dataset. Another difference between the query plans is the number of sort and sub-query scans where there is 1 more number of sorts in the smaller dataset whereas there is 1 less number of subquery scans in the medium dataset. Due to the larger data size, the query execution time took around ten times longer for the medium dataset whereas the planning time stayed roughly the same for both datasets.

Group Contribution

Linh Trinh, 510288769

Completed Individual Question 4

Contributed meaningfully

Xuyang Li, 490153372

Completed Individual Question 3

Contributed meaningfully

Baiyu Chen, 510055611

Completed Individual Question 1

Contributed meaningfully