dbql kicks in Project Report

Sungjoon Park
School of Engineering and Applied Sciences
State University of New York at Buffalo
Buffalo, New York, USA
spark55@buffalo.edu

Dennis M. Kyalo
School of Engineering and Applied Sciences
State University of New York at Buffalo
Buffalo, New York, USA
dkyalo@buffalo.edu

Sol Jang
School of Engineering and Applied Sciences
State University of New York at Buffalo
Buffalo, New York, USA
soljang@buffalo.edu

Abstract*— *Recording and maintaining a database is one of the significant and demanding factors for travel agency companies. Recently, with the pent-up demand for a trip, the importance of flight information database is increasing. We are going to design the flight information DB containing flight origin and destination, flight delay or not, etc.. so that the travlel agency can effectively give customers requested information whenever customers want to query flight schedules.

I. INTRODUCTION

According to Bloomberg Report released in 2021: " Pent-Up U.S. Travel Demand Poised to Trigger 30%..". The article said that All that travel demand has been piling up for more than a year, and now it's about to burst out into one hot summer for U.S. domestic flights -- and the jet fuel producers hammered by 2020's lockdowns.

Now, in 2022, the lockdown has been lifted, and travel demand is more explosive than last year. The travel agency company using this database can administer the database, and utilize it with the response to customer's flight information queries.

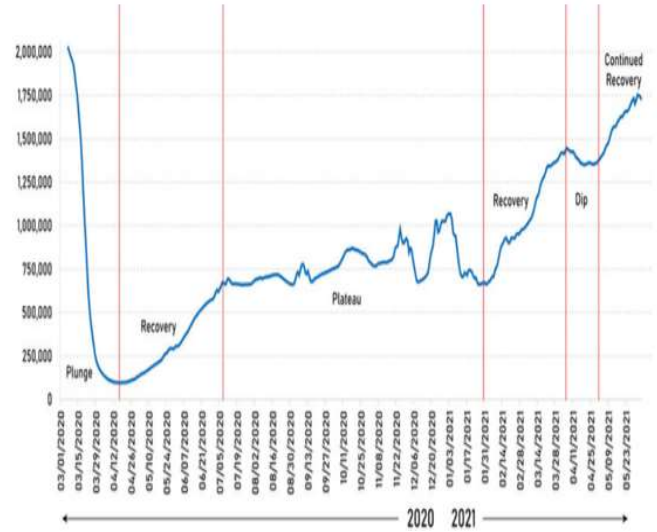

Fig 1. 2020 – 2021 flight demand trend

II. MOTIVATION

It will be imperative for travelers to know how accurate their flight plans are. To meet the needs of travelers, it is necessary to use various information which have impact on flight schedule. The flight schedule is basically affected by various factors such as weather, aircraft safety check, administration, excess demand, etc.

In this regard, our main purpose is to build a database for our target users so that they can take flight delay information into account before they make decisions. The database must be suffice BCNF normal form so that it efficiently maintains flight information and abide by integrity constraints for some events such as update, insert and delete.

III. IMPLEMENTATION

Our previous database has some drawbacks, such as having a duplicate primary key, or one primary key having two values at the same time. So we made new 4 tables to divide them into different primary key.

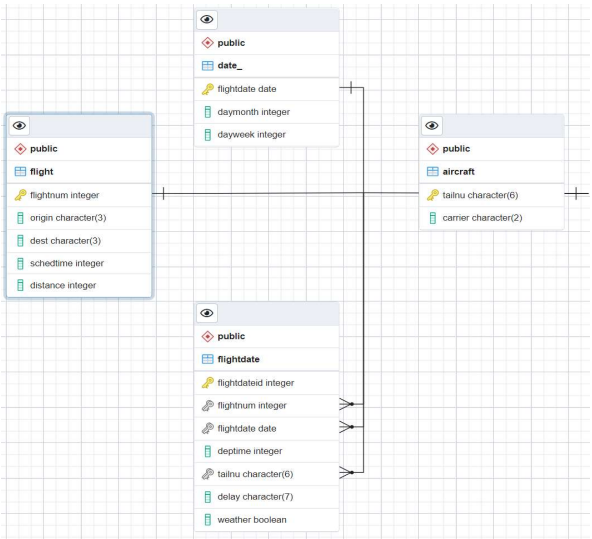

Fig 2. E/R diagram

First of all, we designed necessary relations along with NULL constraint and the foreign key constraint. Regarding flight schedule, all the given attributes are significantly meaningful for both the travle agency and customer. That's why we set all attributes as NOT NULL. In terms of foreign key constraint, we set all the referenced attributes as ON DELETE CASCADE, as partial record of data may cause the error while searching for the flight schedule.

[FLIGHTDATE Table]
FLIGHTDATE(FlightdateID, FlightNum, FlightDate, Deptime, Tailnu, Delay, Weather)
(1)Indicate the primary key and foreign keys:
Primary Key: FlightdateID. Each flight information is a unique record so that each flight set as primary key. The table has 3 foreign keys, and all of them has 'ON DELETE CASCADE' constraints. 'FlightNum' attribute references FLIGHT table on the same attribute. 'FlightDate' attribute references FlightDate table on the same attribute.'Tailnu' attribute attribute references AIRCRAFT table on the same attribute. I selected these 3 foreign keys to prevent any duplicated data so that database can be managed efficiently.
(2)The detailed description of each attribute:
FlightdateID indicates the row number of each flight information. The purpose is to identify unique flight contents. It is INT datatype as it only consists of an integer number.
Delay indicates if the schedule is delayed or not. The purpose is to give both customers and travel agencies existence of flight delay. It is CHAR(7) datatype which can has maximum 7 characters.
Weather indicates if the weather is good(False) or not(True). The purpose is to check the airport's weather when landing. It is BOOLEAN datatype which can has either FALSE or TRUE.
(3)NOT NULL for all attiributes because all the contents are important ones as I forementioned.
(4)Explain the actions taken on any foreign key when the primary key: All foreign keys are set as ON DELETE CACADE so that when a row is deleted from the parent table, the row of the referenced child table is also deleted.

```
CREATE TABLE FLIGHTDATE(
        FlightdateID    INT,
        FlightNum       INT NOT NULL,
        FlightDate      DATE NOT NULL,
        Deptime         INT NOT NULL,
        Tailnu          CHAR(6) NOT NULL,
        Delay           CHAR(7) NOT NULL,
        Weather         BOOLEAN NOT NULL,
        PRIMARY KEY(FlightdateID),
        FOREIGN KEY(FlightNum) REFERENCES FLIGHT(FlightNum) ON DELETE CASCADE,
        FOREIGN KEY(FlightDate) REFERENCES DATE_(FlightDate) ON DELETE CASCADE,
        FOREIGN KEY(Tailnu) REFERENCES AIRCRAFT(Tailnu) ON DELETE CASCADE);
```

Fig 3. FLIGHTDATE table schema

[FLIGHT Table]
FLIGHT(FlightNum, Origin, Dest, Schedtime, Distance)
(1)Primary Key: FlightNum. Each flight number must be unique in the relation considering the definition of flight number itself.
(2)The detailed description of each attribute:
FlightNum indicates the allocated unique flight number. The purpose is to check the flight number if needed from the customer. It is INT datatype as it only consists of integer number.Origin and Dest indicates the departure and arrival airport respectively. The purpose is to match the flight schedule. It is CHAR(3) datatype which can have maximum 3 characters. Schedtime indicates the scheduled departued time. The purpose is to compare it with the real departure time. It is INT datatype which will indicate time with 24 hour system
Distance indicates literally distance between the origin and arrival airport. The purpose is to give customer multiple searching options. It is INT datatype and we used mile unit.
(3)NOT NULL for all attiributes because all the contents are important ones as I forementioned.

```
CREATE TABLE FLIGHT(
        FlightNum       INT,
        Origin          CHAR(3) NOT NULL,
        Dest            CHAR(3) NOT NULL,
        Schedtime       INT NOT NULL,
        Distance        INT NOT NULL,
        PRIMARY KEY(FlightNum)
        );
```

Fig 4. FLIGHT table schema

[DATE_ Table]
DATE_(FlightDate, DayMonth, DayWeek)

(1)Primary Key: FlightDate. Each row contains the date of possible flight schedule, so it must be unique in the given relation.
(2)The detailed description of each attribute:
FlightDate indicates the possible flight date given dataset. The purpose is to give both agency and customer to choose available trvale date and search for the flight information. It is DATE datatype such as 1/01/2004.
DayMonth and DayWeek indicates actual date and week in a month respectively, such as 1(1st), 15(15th), 1(first week), or 2(second week). The purpose is to give a range while searching for the schedule. They are int datatype.

```
CREATE TABLE DATE_(
        FlightDate      DATE,
        DayMonth        INT NOT NULL,
        DayWeek         INT NOT NULL,
        PRIMARY KEY(FlightDate));
```

Fig 5. DATE_ table schema

[AIRCRAFT Table]
AIRCRAFT(Tailnu, Carrier)
(1)Primary Key: Tailnu. Each row can be identified uniquely since every tailnu has their own number.
(2)The detailed description of each attribute:
Tailnu indicates the code of a specific airplane. The purpose is for customer to identify the type of airplane with the tail number. It is CHAR(6) datatype such as N695BR.
Carrier indicates which company the airplane belong to. The purpose is for customer to give airline information. It is CHAR(2) datatype such as UA(United Airline).

```
CREATE TABLE AIRCRAFT(
        Tailnu  CHAR(6),
        Carrier CHAR(2) NOT NULL,
        PRIMARY KEY (Tailnu));
```

Fig 6. AIRCRAFT table schema

Theses tables respect the BCNF. Since we designed each relation schema to have primary key with atomic domain, the database schema suffices the first normal form. In addition, it does not violate the second normal form as well because it does not have any primary key which is a composite key. Now, we need to check if our relations respect the third normal form and the BCNF. We can say that a Relation satisfies the BCNF if its all attributes other than the primary key do not functionally determine the other attributes. There is only one FD in each relation. Only does the primary key in each relation functionally determine the remaining attributes. For example, in Date Relation, even though two tuples agree on DayWeek, there is no guarantee that those tuples have the same DayMonth or the same Flight values. That is, all the tables respect the third normal form and the BCNF.

We used Insertion Commands to insert our dataset. The examples of insertion commands are as follows.

[FLIGHTDATE Table Insertion]

```
INSERT INTO FLIGHTDATE(FlightdateID, FlightNum, FlightDate, Deptime, Tailnu, Delay, Weather)
VALUES
(
1, 5935, '1/1/2004', 1455, 'N940CA', 'ontime', 'FALSE'
);

INSERT INTO FLIGHTDATE(FlightdateID, FlightNum, FlightDate, Deptime, Tailnu, Delay, Weather)
VALUES
(
2, 6155, '1/1/2004', 1640, 'N405FJ', 'ontime', 'FALSE'
);

INSERT INTO FLIGHTDATE(FlightdateID, FlightNum, FlightDate, Deptime, Tailnu, Delay, Weather)
VALUES
(
3, 7208, '1/1/2004', 1245, 'N695BR', 'ontime', 'FALSE'
);

INSERT INTO FLIGHTDATE(FlightdateID, FlightNum, FlightDate, Deptime, Tailnu, Delay, Weather)
VALUES
(
4, 7215, '1/1/2004', 1709, 'N662BR', 'ontime', 'FALSE'
);

INSERT INTO FLIGHTDATE(FlightdateID, FlightNum, FlightDate, Deptime, Tailnu, Delay, Weather)
VALUES
(
5, 7792, '1/1/2004', 1035, 'N698BR', 'ontime', 'FALSE'
);
```

Fig 7. FLIGHTDATE table Insertion commands

[FLIGHT Table Insertion]

```
INSERT INTO FLIGHT(FlightNum, Origin, Dest, Schedtime, Distance)
VALUES
(
5935, 'BWI', 'JFK', 1455, 184
);

INSERT INTO FLIGHT(FlightNum, Origin, Dest, Schedtime, Distance)
VALUES
(
6155, 'DCA', 'JFK', 1640, 213
);

INSERT INTO FLIGHT(FlightNum, Origin, Dest, Schedtime, Distance)
VALUES
(
7208, 'IAD', 'LGA', 1245, 229
);

INSERT INTO FLIGHT(FlightNum, Origin, Dest, Schedtime, Distance)
VALUES
(
7215, 'IAD', 'LGA', 1715, 229
);

INSERT INTO FLIGHT(FlightNum, Origin, Dest, Schedtime, Distance)
VALUES
(
7792, 'IAD', 'LGA', 1039, 229
);
```

Fig 8. FLIGHT table Insertion commands

[DATE_ Table Insertion]

```
INSERT INTO DATE_(FlightDate, DayMonth, DayWeek)
VALUES
(
'1/1/2004', 1, 4
);

INSERT INTO DATE_(FlightDate, DayMonth, DayWeek)
VALUES
(
'1/2/2004', 2, 5
);

INSERT INTO DATE_(FlightDate, DayMonth, DayWeek)
VALUES
(
'1/3/2004', 3, 6
);

INSERT INTO DATE_(FlightDate, DayMonth, DayWeek)
VALUES
(
'1/4/2004', 4, 7
);

INSERT INTO DATE_(FlightDate, DayMonth, DayWeek)
VALUES
(
'1/5/2004', 5, 1
);
```

Fig 9. DATE_ table Insertion commands

[AIRCRAFT Table Insertion]

```
INSERT INTO AIRCRAFT(Tailnu, Carrier)
VALUES
(
'N940CA', 'OH'
);

INSERT INTO AIRCRAFT(Tailnu, Carrier)
VALUES
(
'N405FJ', 'DH'
);

INSERT INTO AIRCRAFT(Tailnu, Carrier)
VALUES
(
'N695BR', 'DH'
);

INSERT INTO AIRCRAFT(Tailnu, Carrier)
VALUES
(
'N662BR', 'DH'
);

INSERT INTO AIRCRAFT(Tailnu, Carrier)
VALUES
(
'N698BR', 'DH'
);
```

Fig 10. AIRCRAFT table Insertion commands

We tried to use INDEX structure to scan faster while querying. When searching by indexing the columns of the TABLE, the speed of searching si accelerated by searching the index file rather than performing a full scan of the records of the corresponding table.

```
CREATE INDEX PG_IDX
ON FLIGHTDATE(flightdateid);

EXPLAIN ANALYZE
SELECT *
FROM FLIGHTDATE;

DROP INDEX PG_IDX;

EXPLAIN ANALYZE
SELECT flightdateid
FROM FLIGHTDATE;
```

Fig 11. INDEX Usage

But, in order to see a remarkable improvement of query performance, we need to get a lot of rows(such as 100,000 rows). Even though we have only around 2200 rows, after executing same query with and without INDEX 30 times, we can see that there are slight improvement of query execution time with index. We can see averagely 0.3ms improvement with 30 times execution.

We wrote total 9 SQL queries in which we identified 3 problematic queries and improved the performance with replacing them with more efficient queries.

(1)

```
EXPLAIN ANALYZE
SELECT DISTINCT
t1.tailnu
FROM FLIGHTDATE as t1
INNER JOIN FLIGHT as t2
ON t1.flightnum = t2.fl
OR t1.deptime = t2.schedtime;
```

```
EXPLAIN ANALYZE
SELECT
t1.tailnu
FROM FLIGHTDATE as t1
INNER JOIN FLIGHT as t2
ON t1.flightnum = t2.flightnum
UNION
SELECT
        t3.tailnu
FROM FLIGHTDATE as t3
INNER JOIN FLIGHT as t4
ON t3.deptime = t4.schedtime;
```

	tailnu character (6)
1	N995CA
2	N16147
3	N635BR
4	N315UE
5	N730UW

Fig 12. Query comparison between OR & UNION , and the execution result of the query(first 5 rows only)

The queries above return the same result as logically they yield the same table. Seemingly, the right query looks more complex, but in terms of the execution time, it showed 10 times faster compared to the left one. It is basically derived from the difference in the executin plan. When using OR query, SQL Server cannot process it faster if the table has lots of columns. It is because OR condition takes too much computing resource when executing. So, if possible, we should avoid using OR query. Instead, I introduced UNION condition so that the total read of the rows has been reduced significantly. The QUERY PLAN figure below showed that there is remarkable execution improvement in the case of using UNION condition.

	QUERY PLAN text
1	HashAggregate (cost=176.63..205.89 rows=2926 width=28) (actual time=4.389..4.521 rows=548 loops=1)
2	[...] Group Key: t1.tailnu
3	[...] Batches: 1 Memory Usage: 145kB
4	[...] -> Append (cost=3.32..169.32 rows=2926 width=28) (actual time=0.102..3.000 rows=2975 loops=1)
5	[...] -> Hash Join (cost=3.32..48.34 rows=2201 width=7) (actual time=0.101..1.433 rows=2200 loops=1)
6	[...] Hash Cond: (t1.flightnum = t2.flightnum)
7	[...] -> Seq Scan on flightdate t1 (cost=0.00..39.01 rows=2201 width=11) (actual time=0.024..0.350 rows=2200 loops=1)
8	[...] -> Hash (cost=2.03..2.03 rows=103 width=4) (actual time=0.058..0.059 rows=103 loops=1)
9	[...] Buckets: 1024 Batches: 1 Memory Usage: 12kB
10	[...] -> Seq Scan on flight t2 (cost=0.00..2.03 rows=103 width=4) (actual time=0.014..0.028 rows=103 loops=1)
11	[...] -> Hash Join (cost=3.32..77.09 rows=725 width=7) (actual time=0.188..1.216 rows=775 loops=1)
12	[...] Hash Cond: (t3.deptime = t4.schedtime)
13	[...] -> Seq Scan on flightdate t3 (cost=0.00..39.01 rows=2201 width=11) (actual time=0.037..0.370 rows=2200 loops=1)
14	[...] -> Hash (cost=2.03..2.03 rows=103 width=4) (actual time=0.108..0.108 rows=103 loops=1)
15	[...] Buckets: 1024 Batches: 1 Memory Usage: 12kB
16	[...] -> Seq Scan on flight t4 (cost=0.00..2.03 rows=103 width=4) (actual time=0.020..0.052 rows=103 loops=1)
17	Planning Time: 0.464 ms
18	Execution Time: 4.760 ms

	QUERY PLAN text
1	HashAggregate (cost=4015.90..4021.39 rows=549 width=7) (actual time=44.777..44.947 rows=548 loops=1)
2	[...] Group Key: t1.tailnu
3	[...] Batches: 1 Memory Usage: 73kB
4	[...] -> Nested Loop (cost=0.00..4008.60 rows=2919 width=7) (actual time=0.042..41.888 rows=2863 loops=1)
5	[...] Join Filter: ((t1.flightnum = t2.flightnum) OR (t1.deptime = t2.schedtime))
6	[...] Rows Removed by Join Filter: 223737
7	[...] -> Seq Scan on flightdate t1 (cost=0.00..39.01 rows=2201 width=15) (actual time=0.013..0.962 rows=2200 loops=1)
8	[...] -> Materialize (cost=0.00..2.55 rows=103 width=8) (actual time=0.000..0.006 rows=103 loops=2200)
9	[...] -> Seq Scan on flight t2 (cost=0.00..2.03 rows=103 width=8) (actual time=0.007..0.018 rows=103 loops=1)
10	Planning Time: 0.133 ms
11	Execution Time: 45.084 ms

Fig 13. Execution time comparison between UNION & OR

(2)

```
EXPLAIN ANALYZE
SELECT FD.FlightNum, FL.Schedtime,
MAX(FD.Deptime)
FROM FLIGHTDATE FD
LEFT JOIN FLIGHT FL
ON FD.FlightNum=FL.FlightNum
GROUP BY FL.Schedtime,FD.FlightNum
HAVING FL.Schedtime>0900;
```

```
EXPLAIN ANALYZE
SELECT  FD.FlightNum,FL.Schedtime,
MAX(FD.Deptime)
FROM FLIGHTDATE FD
LEFT JOIN FLIGHT FL
ON FD.FlightNum=FL.FlightNum
WHERE FL.Schedtime>0900
GROUP BY FL.Schedtime,FD.FlightNum;
```

	flightnum integer	schedtime integer	max integer
1	7304	2120	2330
2	1744	930	842
3	2180	1700	1733
4	2170	1100	1202
5	7808	1245	1539

Fig 14. Query comparison between GROUP BY→HAVING & WHERE→GROUP BY , and the execution result of the query(first 5 rows only)

The queries above also return the same result, but the right query run faster than the left one. If we write GROUP BY first and then HAVING clause, GROUP BY clause will sort the set of rows first, and apply to HAVING clause. On the other hand, if we write WHERE clause first, the server can get rid of unnecessary rows, then execute GROUP BY clause so that it can be executed faster.

	QUERY PLAN text
1	HashAggregate (cost=61.47..78.99 rows=1752 width=12) (actual time=1.236..1.259 rows=82 loops=1)
2	[...] Group Key: fl.schedtime, fd.flightnum
3	[...] Batches: 1 Memory Usage: 73kB
4	[...] -> Hash Join (cost=3.31..48.33 rows=1752 width=12) (actual time=0.051..0.788 rows=1841 loops=1)
5	[...] Hash Cond: (fd.flightnum = fl.flightnum)
6	[...] -> Seq Scan on flightdate fd (cost=0.00..39.01 rows=2201 width=8) (actual time=0.014..0.187 rows=2200 loops=1)
7	[...] -> Hash (cost=2.29..2.29 rows=82 width=8) (actual time=0.032..0.032 rows=82 loops=1)
8	[...] Buckets: 1024 Batches: 1 Memory Usage: 12kB
9	[...] -> Seq Scan on flight fl (cost=0.00..2.29 rows=82 width=8) (actual time=0.007..0.019 rows=82 loops=1)
10	[...] Filter: (schedtime > 900)
11	[...] Rows Removed by Filter: 21
12	Planning Time: 0.217 ms
13	Execution Time: 1.313 ms

1	HashAggregate (cost=61.47..78.99 rows=1752 width=12) (actual time=2.786..2.822 rows=82 loops=1)
2	[...] Group Key: fl.schedtime, fd.flightnum
3	[...] Batches: 1 Memory Usage: 73kB
4	[...] -> Hash Join (cost=3.31..48.33 rows=1752 width=12) (actual time=0.090..1.777 rows=1841 loops=1)
5	[...] Hash Cond: (fd.flightnum = fl.flightnum)
6	[...] -> Seq Scan on flightdate fd (cost=0.00..39.01 rows=2201 width=8) (actual time=0.023..0.439 rows=2200 loops=1)
7	[...] -> Hash (cost=2.29..2.29 rows=82 width=8) (actual time=0.059..0.061 rows=82 loops=1)
8	[...] Buckets: 1024 Batches: 1 Memory Usage: 12kB
9	[...] -> Seq Scan on flight fl (cost=0.00..2.29 rows=82 width=8) (actual time=0.014..0.037 rows=82 loops=1)
10	[...] Filter: (schedtime > 900)
11	[...] Rows Removed by Filter: 21
12	Planning Time: 0.390 ms
13	Execution Time: 2.902 ms

Fig 15. Execution time comparison between GROUP BY→HAVING & WHERE→GROUP BY

(3)

```
EXPLAIN ANALYZE SELECT *
FROM FLIGHTDATE
WHERE FlightDate in
(SELECT FlightDate
FROM FLIGHTDATE
WHERE FlightDate<='2004-01-20');
```

```
EXPLAIN ANALYZE SELECT *
FROM FLIGHTDATE
WHERE FlightDate BETWEEN
'2004-01-01' AND '2004-01-20';
```

	flightdateid [PK] integer	flightnum integer	flightdate date	deptime integer	tailnu character (6)	delay character (7)	weather boolean
1	2	6155	2004-01-01	1640	N405FJ	ontime	false
2	3	7208	2004-01-01	1245	N695BR	ontime	false
3	4	7215	2004-01-01	1709	N662BR	ontime	false
4	5	7792	2004-01-01	1035	N698BR	ontime	false
5	6	7800	2004-01-01	839	N687BR	ontime	false

Fig 16. Query comparison between subquery & without subquery, and the execution result of the query(first 5 rows only)

They had the same table as the result, but the modified query has an execution 5 times faster than the original query as it does not contain a subquery. If we use a subquery, it would demand a higher computing resource as the query plan shows below. Using a subquery means making a virtual table so that whenever they have to access to a subquery, they need to add more select query to make virtual data, and this result in longer execution time. So, we should avoid using a subquery as possible as we can.

1	Seq Scan on flightdate (cost=0.00..50.02 rows=1431 width=32) (actual time=0.018..0.584 rows=1430 loops=1)
2	[...] Filter: ((flightdate >= '2004-01-01'::date) AND (flightdate <= '2004-01-20'::date))
3	[...] Rows Removed by Filter: 770
4	Planning Time: 0.143 ms
5	Execution Time: 0.671 ms

1	Nested Loop (cost=41.19..75.91 rows=2201 width=32) (actual time=1.020..2.929 rows=1430 loops=1)
2	[...] -> HashAggregate (cost=40.90..41.21 rows=31 width=4) (actual time=0.993..1.010 rows=20 loops=1)
3	[...] Group Key: flightdate_1.flightdate
4	[...] Batches: 1 Memory Usage: 24kB
5	[...] -> Index Only Scan using pg_index on flightdate flightdate_1 (cost=0.28..37.32 rows=1431 width=4) (actual time=0.052..0.421 rows=1430 loops=1)
6	[...] Index Cond: (flightdate <= '2004-01-20'::date)
7	[...] Heap Fetches: 136
8	[...] -> Memoize (cost=0.29..4.24 rows=71 width=32) (actual time=0.009..0.066 rows=72 loops=20)
9	[...] Cache Key: flightdate_1.flightdate
10	[...] Cache Mode: logical
11	[...] Hits: 0 Misses: 20 Evictions: 0 Overflows: 0 Memory Usage: 91kB
12	[...] -> Index Scan using pg_index on flightdate (cost=0.28..4.23 rows=71 width=32) (actual time=0.007..0.035 rows=72 loops=20)
13	[...] Index Cond: (flightdate = flightdate_1.flightdate)
14	Planning Time: 0.503 ms
15	Execution Time: 3.274 ms

Fig 17. Execution time comparison between subquery & without subquery

(4)

To search for origination & destination airport, departure time, and tail number of the airplane

```
SELECT FL.origin,FL.dest, FL.schedtime, FD.deptime,FD.tailnu
FROM FLIGHTDATE FD
LEFT JOIN FLIGHT FL
ON FD.flightnum=FL.flightnum
WHERE FD.delay='delayed' AND FD.deptime-FL.schedtime>=600
```

	origin character (3)	dest character (3)	schedtime integer	deptime integer	tailnu character (6)
1	BWI	EWR	700	1416	N13949
2	DCA	LGA	700	2030	N14933
3	DCA	LGA	900	1833	N16649
4	DCA	EWR	759	1856	N59630
5	DCA	JFK	1455	2222	N14573

(5)

To search for tailnumber, flight number, and the number of flight dates given in the information before.

```
SELECT tailnu, flightnum, COUNT(FLIGHTDATE) as flight_count_month
FROM FLIGHTDATE
group by tailnu,flightnum
ORDER BY COUNT(FLIGHTDATE) DESC
```

	tailnu character (6)	flightnum integer	flight_count_month bigint
1	N223DZ	1748	9
2	N223DZ	1756	8
3	N223DZ	1764	8
4	N242DL	1752	7
5	N333UE	7812	7

(6)

To update departure time with 900 corresponding to the value with tail number=N14949 and departure time=14:16

```
UPDATE
FLIGHTDATE
SET
deptime =900
WHERE tailnu='N13949' and deptime=1416;
```

```
UPDATE 1

Query returned successfully in 88 msec.
```

```
SELECT * FROM FLIGHTDATE
WHERE tailnu='N13949' and deptime=900;
```

	flightdateid [PK] integer	flightnum integer	flightdate date	deptime integer	tailnu character (6)	delay character (7)	weather boolean
1	210	2703	2004-01-04	900	N13949	delayed	false

(7)

To update scheduled time with 9:00 corresponding to the value with flight number=5935 and scheduled time=14:55

```
UPDATE
FLIGHT
SET
schedtime = 900
WHERE flightnum = 5935 and schedtime=1455;

UPDATE 1

Query returned successfully in 99 msec.

SELECT * FROM FLIGHT
WHERE flightnum = 5935 and schedtime=900;
```

	flightnum [PK] integer	origin character (3)	dest character (3)	schedtime integer	distance integer
1	5935	BWI	JFK	900	184

(8)

To delete the row which has tail number =N940CA

```
DELETE FROM aircraft
WHERE tailnu='N940CA'

DELETE 0

Query returned successfully in 83 msec.

SELECT * FROM aircraft
WHERE tailnu='N940CA'
```

	tailnu [PK] character (6)	carrier character (2)

```
SELECT * FROM flightdate
WHERE tailnu='N940CA'
```

	flightdateid [PK] integer	flightnum integer	flightdate date	deptime integer	tailnu character (6)	delay character (7)	weather boolean

(9)

To insert a new value into aircraft relation with tail number=N599BR and carrier=DH

```
INSERT INTO aircraft( tailnu, carrier)
VALUES('N599BR','DH')

INSERT 0 1

Query returned successfully in 75 msec.

SELECT * FROM aircraft
WHERE tailnu='N599BR' and carrier='DH';
```

	tailnu [PK] character (6)	carrier character (2)
1	N599BR	DH

IV. CONCLUSION

We have designed a database schema of flight information for both customers and travel agencies, and checked if the database is developed well with SQL queries such as inner & outer join, order by, group by, subquery, and having clauses. We also identified three problematic queries by using ‘explain analyze’ which can analyze a query execution. We have learned that both accurate database schema and efficient queries are significant factors in terms of maintenance and costs.