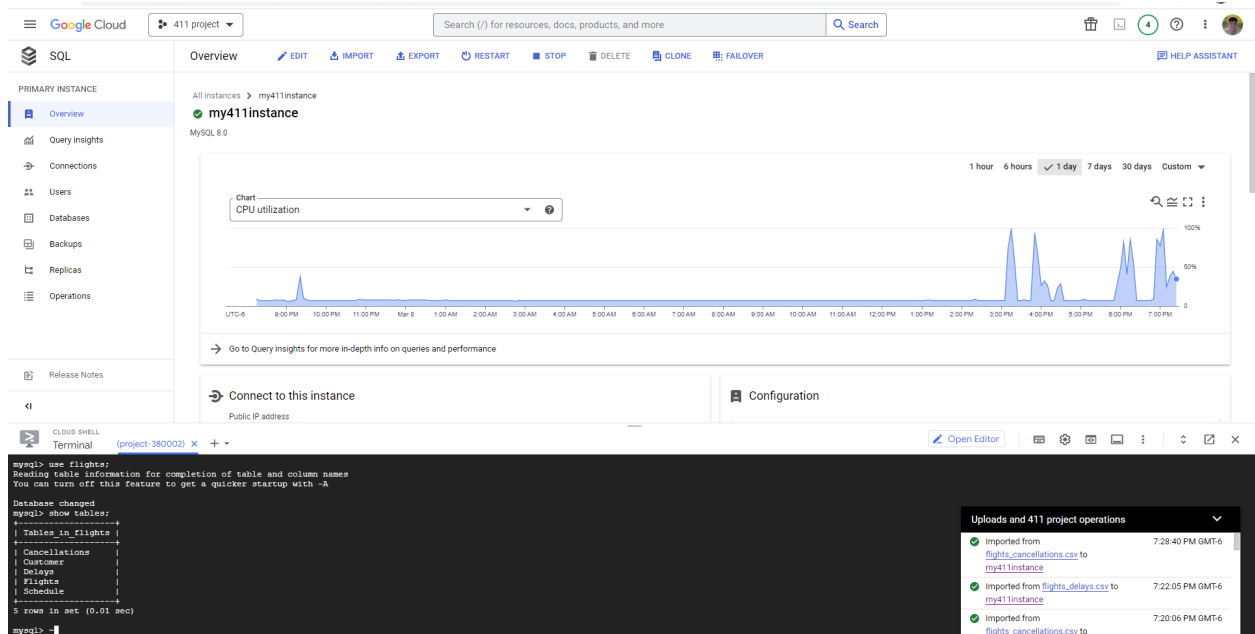


## Database implementation:



## DDL commands:

```
CREATE TABLE Flights (  
    FlightId INT AUTO_INCREMENT,  
    Airline VARCHAR(50),  
    Origin VARCHAR(5),  
    Destination VARCHAR(5),  
    Date INT,  
    Month INT,  
    PRIMARY KEY(FlightId)  
);
```

```
CREATE TABLE Customer (  
    CustomerId INT,  
    Username VARCHAR(50),  
    Password_ VARCHAR(25),  
    PRIMARY KEY(CustomerId)  
);
```

```
CREATE TABLE Schedule (  
    FlightId INT AUTO_INCREMENT,  
    CustomerId INT,  
    PRIMARY KEY(FlightId, CustomerId),  
    FOREIGN KEY (FlightID) REFERENCES Flights(FlightID) ON DELETE CASCADE,
```

```
    FOREIGN KEY (CustomerId) REFERENCES Customer(CustomerId) ON DELETE  
    CASCADE  
);
```

```
CREATE TABLE Cancellations (  
    FlightId INT AUTO_INCREMENT,  
    Diverted BIT,  
    Cancelled BIT,  
    PRIMARY KEY(FlightId),  
    FOREIGN KEY (FlightID) REFERENCES Flights(FlightID)  
        ON DELETE CASCADE  
);
```

```
CREATE TABLE Delays (  
    FlightId INT AUTO_INCREMENT,  
    DepartureDelay INT,  
    ArrivalDelay INT,  
    PRIMARY KEY(FlightId),  
    FOREIGN KEY (FlightID) REFERENCES Flights(FlightID)  
        ON DELETE CASCADE  
);
```







## Indexing:

query 1:

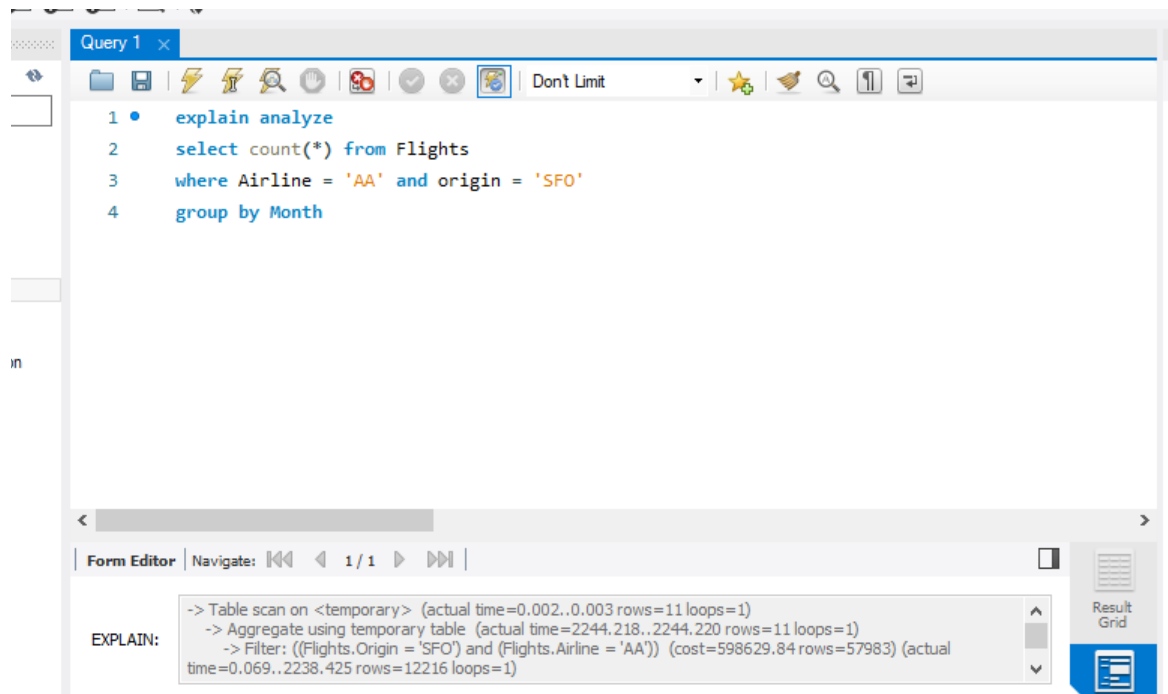
The screenshot shows a database query editor window titled "Query 1". The query is as follows:

```
1 • select count(*) from Flights
2   where airline = 'AA' and origin = 'SFO'
3   group by Month
4   limit 15;
```

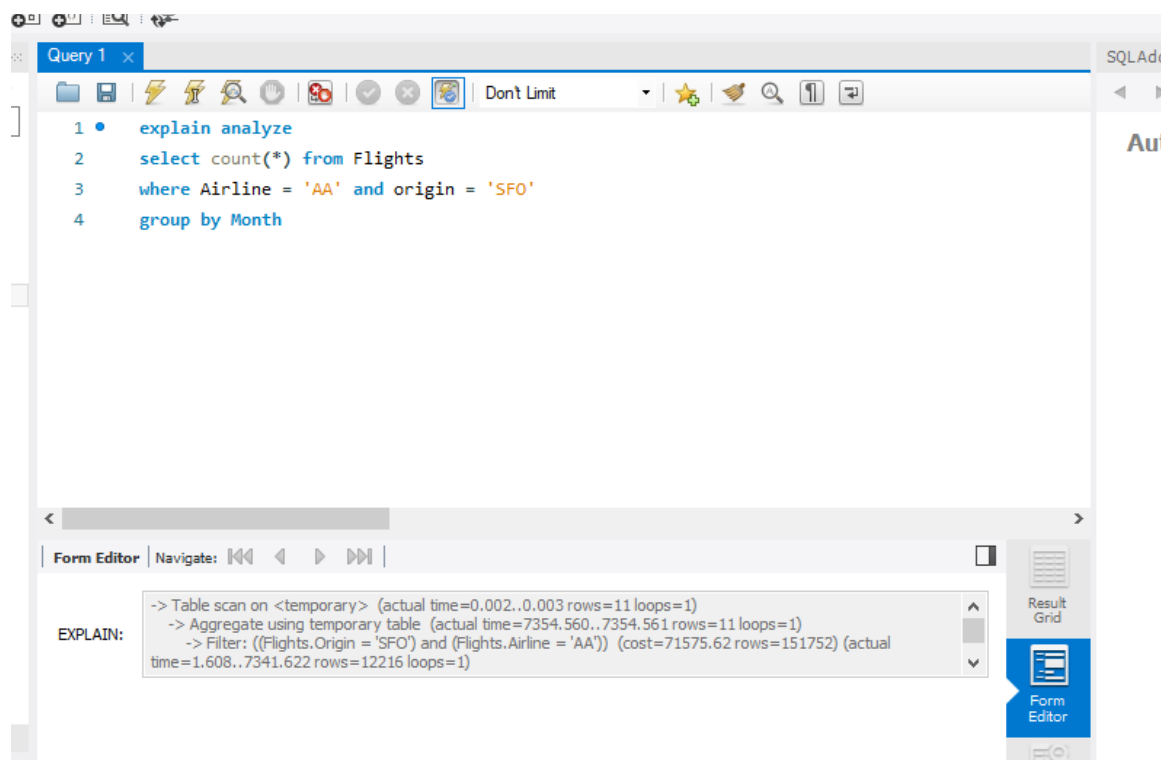
The results are displayed in a table below the query editor. The table has one column, "count(\*)", and 15 rows of data. The results are as follows:

count(*)
877
791
900
871
895
903
1449
1438
1365
1353
1374

The interface includes a toolbar with various icons for file operations, a "Don't Limit" dropdown, and a "Result Grid" button. The "Result Grid" button is highlighted in blue. The "Field Types" button is also visible.



After indexing on Airline name: cost decreased from around 600k to 71k. Chose this index because the query is filtering by airline, so indexing would make it faster for query to filter.



After indexing on Airline and Origin name: cost decreased from around 600k to 64k. Chose this index because the query is filtering by airline and origin, so indexing would make it faster for query to filter.

The screenshot shows a SQL query editor with a toolbar at the top. The query is as follows:

```
1 • create index origin_idx on Flights(Origin(5));
2
3 • explain analyze
4 select count(*) from Flights
5 where Airline = 'AA' and origin = 'SFO'
6 group by Month;
```

Below the query editor, the 'EXPLAIN:' section displays the execution plan:

```
-> Table scan on <temporary> (actual time=0.002..0.003 rows=11 loops=1)
-> Aggregate using temporary table (actual time=871.846..871.848 rows=11 loops=1)
-> Filter: (Flights.Airline = 'AA') (cost=64194.75 rows=77943) (actual time=0.343..867.500 rows=12216 loops=1)
```

On the right side of the editor, there are buttons for 'Result Grid' and 'Form Editor'.



After indexing on month: cost did not decrease. Chose this index to test and see if indexing by flight month would speed up this query.

The screenshot shows a SQL query editor window titled "Query 1". The query is as follows:

```
1  -- create index month_idx on Flights(Month);
2
3  • explain analyze
4  select count(*) from Flights
5  where Airline = 'AA' and origin = 'SFO'
6  group by Month;
```

Below the query editor, the "EXPLAIN:" section displays the execution plan:

```
-> Group aggregate: count(0) (cost=604428.14 rows=57983) (actual time=1102.565..17492.657 rows=11 loops=1)
    -> Filter: ((Flights.Origin = 'SFO') and (Flights.Airline = 'AA')) (cost=598629.84 rows=57983) (actual time=0.190..17483.241 rows=12216 loops=1)
```

The interface includes a toolbar with various icons, a "Don't Limit" dropdown, and a "Form Editor" button in the bottom right corner.

query 2:

The screenshot shows a SQL query editor with a toolbar at the top. The query is as follows:

```
1 • select Airline, count(*) from Flights
2   where destination = 'DFW' and origin = 'SFO'
3   group by Airline
4   limit 15;
```

Below the query editor, the 'Result Grid' tab is active, displaying the following data:

Airline	count(*)
AA	3320
UA	482
OO	753

On the right side of the interface, there are buttons for 'Result Grid' and 'Form Editor'.

The screenshot shows the same SQL query editor with the following query:

```
1 • explain analyze
2   select Airline, count(*) from Flights
3   where destination = 'DFW' and origin = 'SFO'
4   group by Airline;
```

Below the query editor, the 'Form Editor' tab is active, displaying the execution plan (EXPLAIN) for the query:

```
EXPLAIN:
-> Table scan on <temporary> (actual time=0.002..0.002 rows=3 loops=1)
-> Aggregate using temporary table (actual time=2297.602..2297.602 rows=3 loops=1)
-> Filter: ((Flights.Origin = 'SFO') and (Flights.Destination = 'DFW')) (cost=598629.84 rows=57983) (actual
time=0.061..2289.283 rows=4555 loops=1)
```

On the right side of the interface, there are buttons for 'Result Grid' and 'Form Editor'.

After indexing on Origin name: cost decreased from around 600k to 60k. Chose this index because the query is filtering by origin, so indexing would make it faster for query to filter.

The screenshot shows a SQL query editor with a query window and a results window. The query window contains the following SQL code:

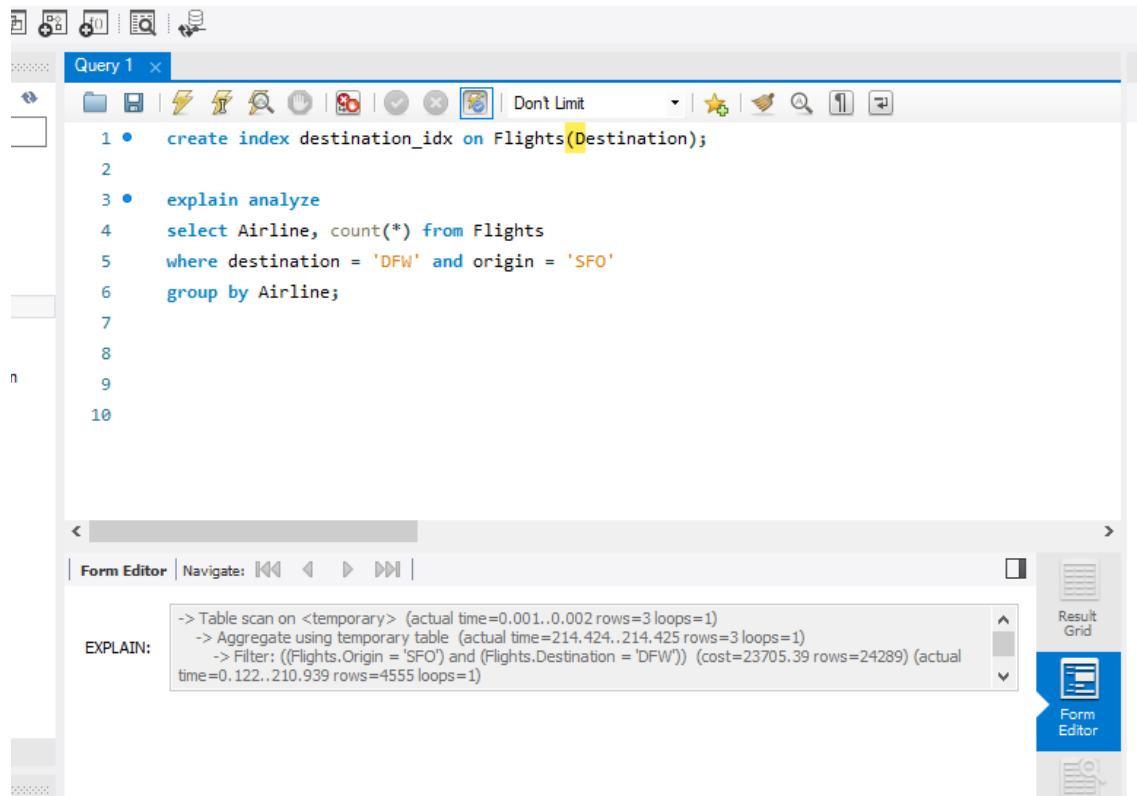
```
1 • create index origin_idx on Flights(Origin);
2
3 • explain analyze
4 select Airline, count(*) from Flights
5 where destination = 'DFW' and origin = 'SFO'
6 group by Airline;
7
8
9
10
```

The results window shows the execution plan for the query. The plan is as follows:

```
EXPLAIN:
-> Table scan on <temporary> (actual time=0.002..0.003 rows=3 loops=1)
-> Aggregate using temporary table (actual time=547.716..547.717 rows=3 loops=1)
-> Filter: (Flights.Destination = 'DFW') (cost=59378.56 rows=29781) (actual time=0.688..543.273
rows=4555 loops=1)
```

The results window also includes a "Result Grid" button and a "Form Editor" button.

After indexing on Origin and Destination name: cost decreased from around 600k to 24k. Chose this index because the query is filtering by origin and destination, so indexing would make it faster for query to filter.



The screenshot shows a SQL query editor with a query window titled "Query 1". The query is as follows:

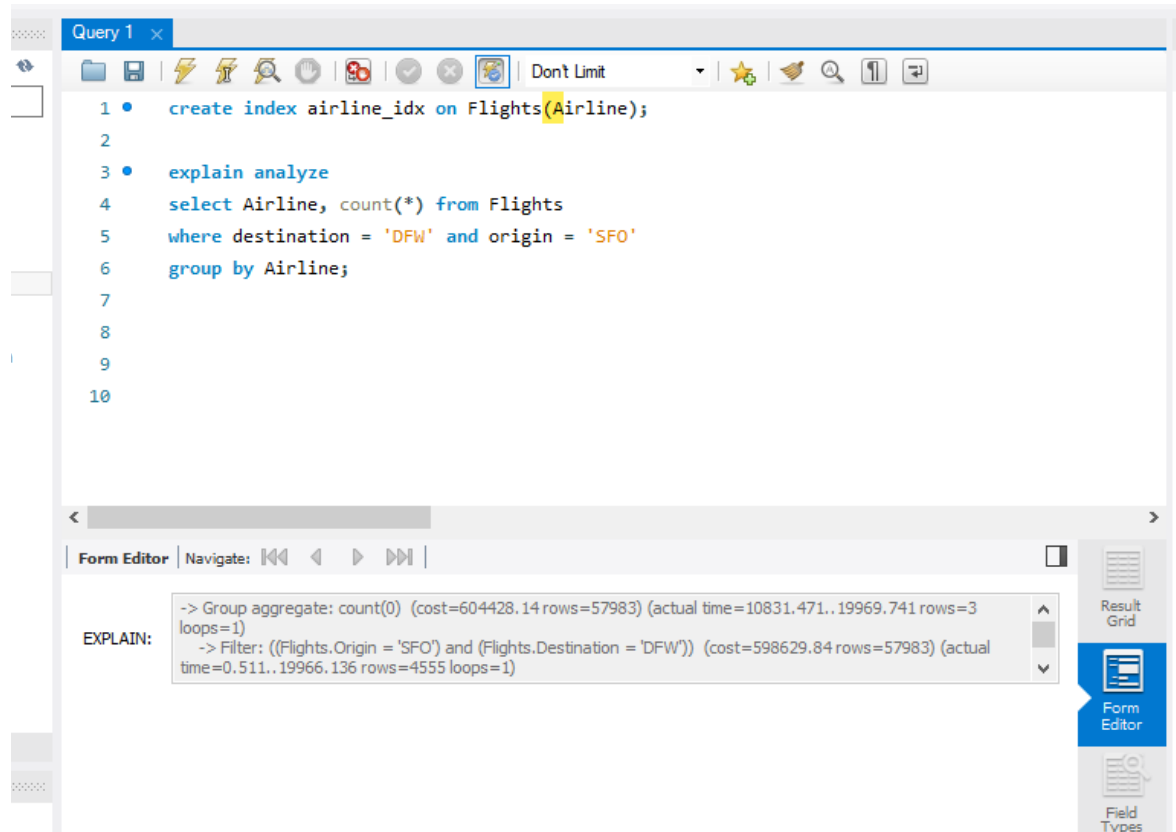
```
1 • create index destination_idx on Flights(Destination);
2
3 • explain analyze
4 select Airline, count(*) from Flights
5 where destination = 'DFW' and origin = 'SFO'
6 group by Airline;
```

The "EXPLAIN:" section shows the execution plan for the query:

```
-> Table scan on <temporary> (actual time=0.001..0.002 rows=3 loops=1)
  -> Aggregate using temporary table (actual time=214.424..214.425 rows=3 loops=1)
    -> Filter: ((Flights.Origin = 'SFO') and (Flights.Destination = 'DFW')) (cost=23705.39 rows=24289) (actual
      time=0.122..210.939 rows=4555 loops=1)
```

The interface includes a toolbar with various icons, a "Don't Limit" dropdown, and a "Form Editor" button on the right side.

After indexing on Airline name: cost did not decrease. Chose this index because query was grouping by airline, so we thought this index might reduce query cost.



The screenshot shows a SQL query editor with a query window titled "Query 1". The query contains the following SQL statements:

```
1 • create index airline_idx on Flights(Airline);
2
3 • explain analyze
4 select Airline, count(*) from Flights
5 where destination = 'DFW' and origin = 'SFO'
6 group by Airline;
```

The bottom panel of the editor shows the "EXPLAIN:" output for the query:

```
-> Group aggregate: count(0) (cost=604428.14 rows=57983) (actual time=10831.471..19969.741 rows=3 loops=1)
-> Filter: ((Flights.Origin = 'SFO') and (Flights.Destination = 'DFW')) (cost=598629.84 rows=57983) (actual time=0.511..19966.136 rows=4555 loops=1)
```

The right sidebar contains icons for "Result Grid", "Form Editor", and "Field Types".