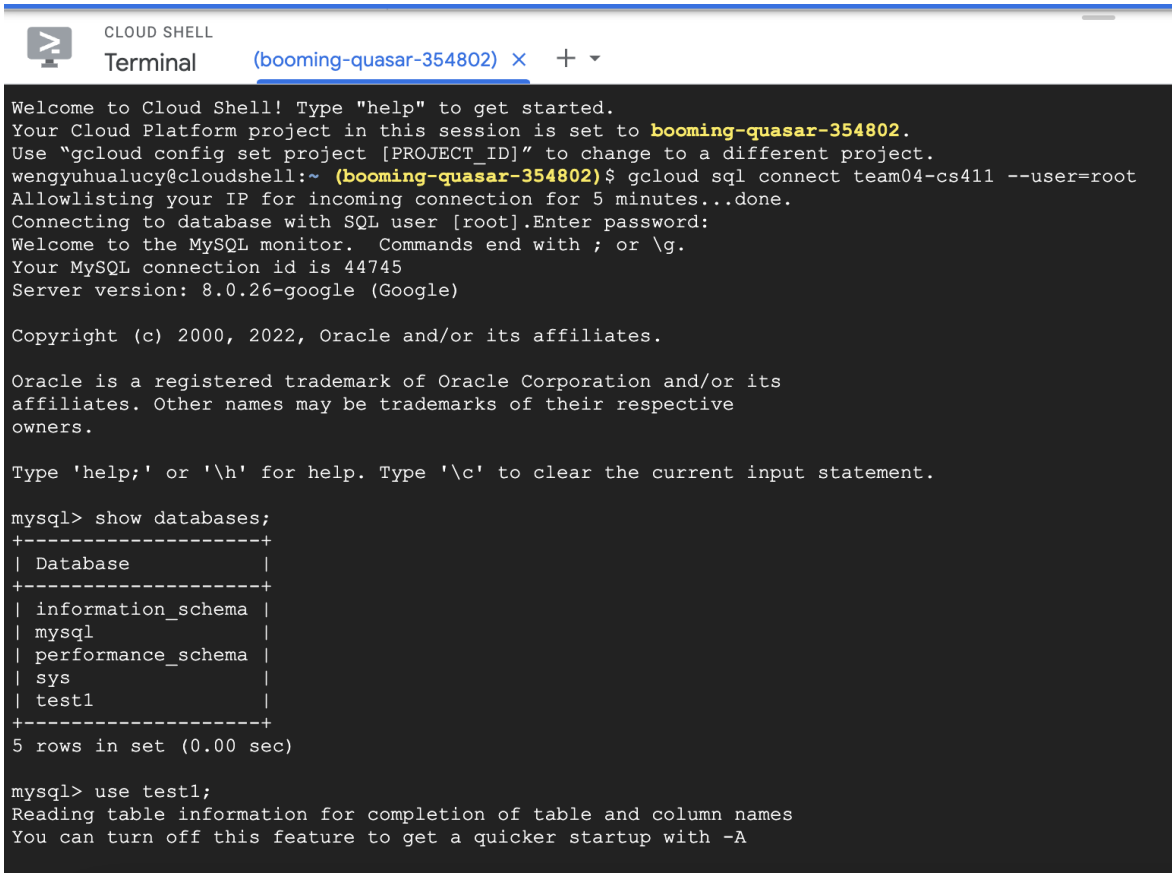
Screenshot of connection



```
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| test1              |
+--------------------+
5 rows in set (0.00 sec)

mysql> use test1;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+-----------------+
| Tables_in_test1 |
+-----------------+
| AirlineCompany  |
| Airport         |
| Flight          |
| Membership      |
| Passenger       |
| Ticket          |
+-----------------+
6 rows in set (0.00 sec)

mysql>
```

Create table DDL commands

```
CREATE TABLE `AirlineCompany`(`AirlineID` INT NOT NULL PRIMARY KEY,
`Name` VARCHAR(50) NOT NULL,
`Rating` DOUBLE NOT NULL)ENGINE=InnoDB DEFAULT CHARSET=latin1;


CREATE TABLE `Airport`(`AirportID` INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
`Name` VARCHAR(50) NOT NULL,
`City` VARCHAR(50) NOT NULL)ENGINE=InnoDB DEFAULT CHARSET=latin1;


CREATE TABLE `Flight`(`FlightCode` INT NOT NULL PRIMARY KEY,
`AirlineID` INT NOT NULL,
```

```
`DepartingAirportID` INT NOT NULL,
`ArrivingAirportID` INT NOT NULL,
`DepartureTime` Date NOT NULL,
`NumberOfStops` INT NOT NULL,
`ArrivalTime` Date NOT NULL,
FOREIGN KEY(AirlineID) REFERENCES AirlineCompany(AirlineID),
FOREIGN KEY(DepartingAirportID) REFERENCES Airport(AirportID),
FOREIGN KEY(ArrivingAirportID) REFERENCES Airport(AirportID))ENGINE=InnoDB
DEFAULT CHARSET=latin1;


CREATE TABLE `Passenger`(`UserID` INT NOT NULL PRIMARY KEY,
`UserName` VARCHAR(50) NOT NULL,
`Password` VARCHAR(50) NOT NULL)ENGINE=InnoDB DEFAULT CHARSET=latin1;


CREATE TABLE `Ticket`(`TicketID` INT NOT NULL PRIMARY KEY,
`UserID` INT NOT NULL,
`FlightCode` INT NOT NULL,
`TicketClass` INT NOT NULL,
`TicketPrice` REAL NOT NULL,
FOREIGN KEY(UserID) REFERENCES Passenger(UserID),
FOREIGN KEY(FlightCode) REFERENCES Flight(FlightCode))ENGINE=InnoDB DEFAULT
CHARSET=latin1;


CREATE TABLE `Membership`(`UserID` INT NOT NULL,
`AirlineID` INT NOT NULL,
`Level` INT NOT NULL,
`Miles` REAL NOT NULL,
FOREIGN KEY(UserID) REFERENCES Passenger(UserID),
FOREIGN KEY(AirlineID) REFERENCES AirlineCompany(AirlineID),
PRIMARY KEY(UserID, AirlineID))ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

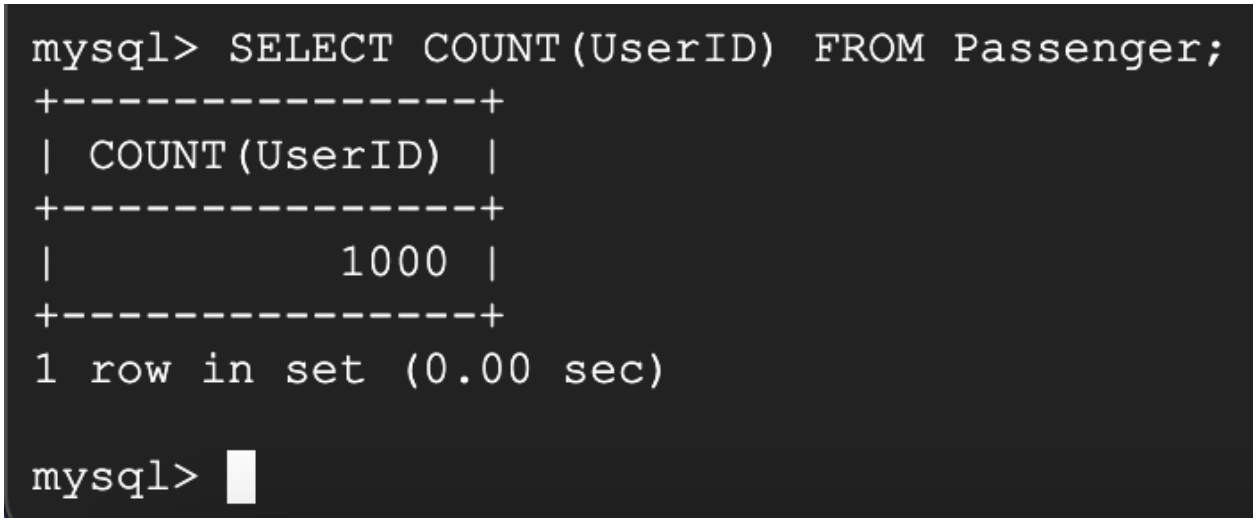
inserting at least 1000 rows in the tables

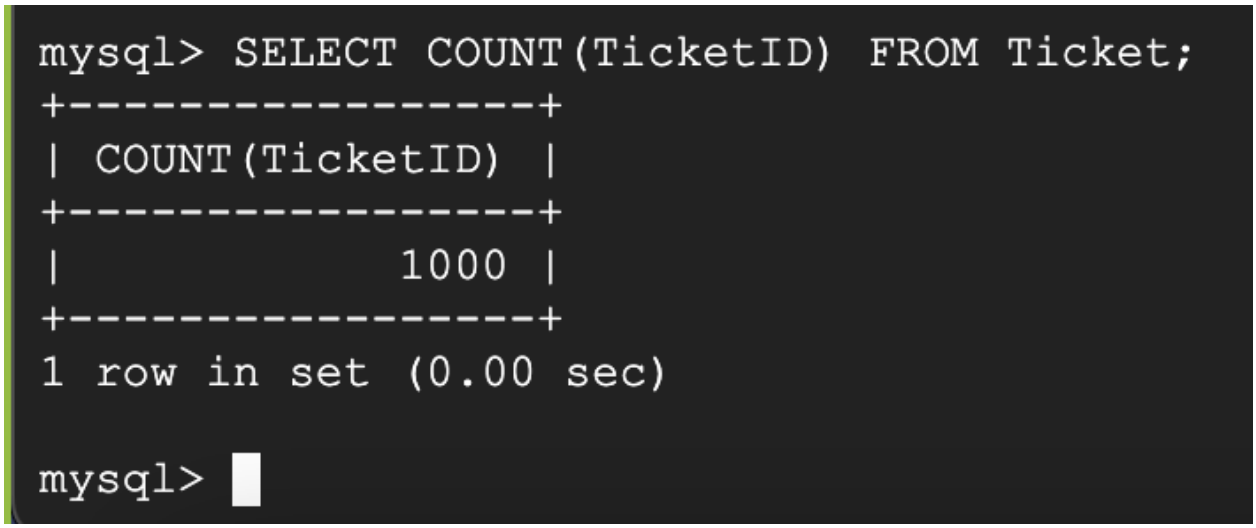
```
mysql> SELECT COUNT(AirlineID) FROM AirlineCompany;
+------------------+
| COUNT(AirlineID) |
+------------------+
|             1000 |
+------------------+
1 row in set (0.00 sec)

mysql>
```

```
mysql> SELECT COUNT(FlightCode) FROM Flight;
+--------------------+
| COUNT(FlightCode)  |
+--------------------+
|               1000 |
+--------------------+
1 row in set (0.00 sec)

mysql>
```





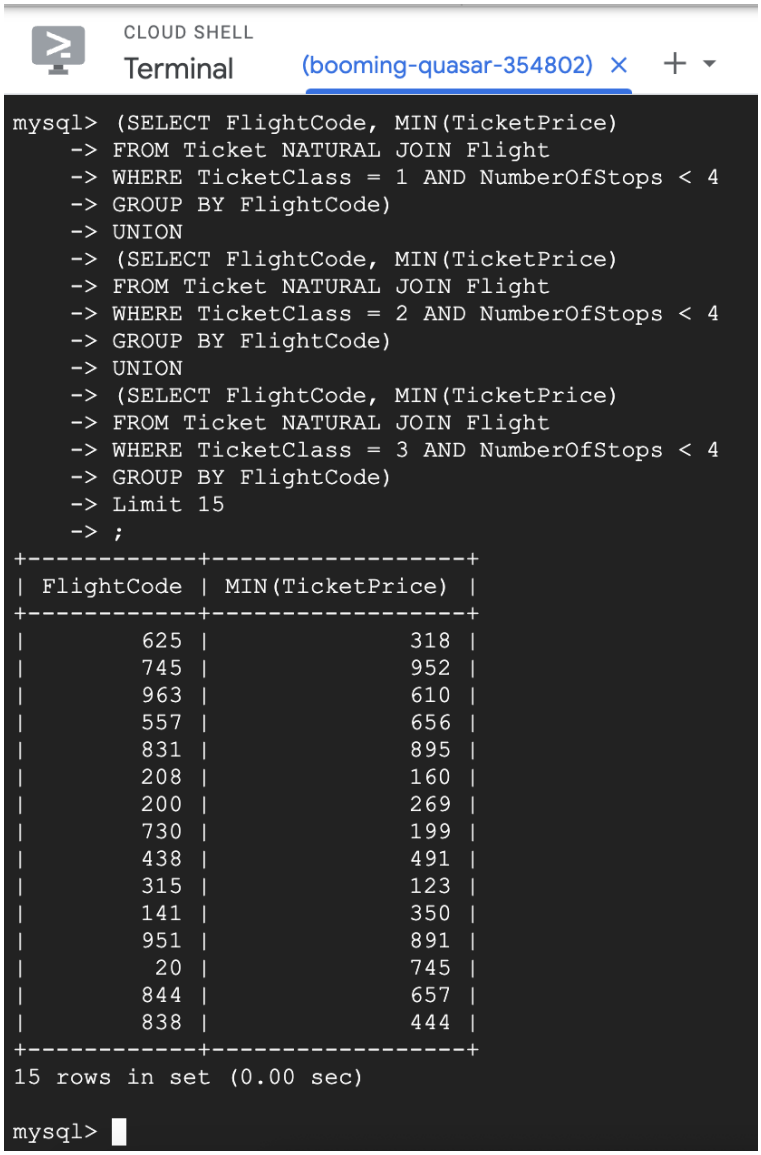
Advanced Queries

1.

Calculate min ticket price for first class, business class, economy class in specific flight which has less than 4 stops

```
(SELECT FlightCode, MIN(TicketPrice)
FROM Ticket NATURAL JOIN Flight
WHERE TicketClass = 1 AND NumberOfStops < 4
GROUP BY FlightCode)
UNION
```

```
(SELECT FlightCode, MIN(TicketPrice)
FROM Ticket NATURAL JOIN Flight
WHERE TicketClass = 2 AND NumberOfStops < 4
GROUP BY FlightCode)
UNION
(SELECT FlightCode, MIN(TicketPrice)
FROM Ticket NATURAL JOIN Flight
WHERE TicketClass = 3 AND NumberOfStops < 4
GROUP BY FlightCode)
Limit 15
```



2.

To find the flight code of the flights and their corresponding airline companies, where this flight departs at __ city and arrives at __ city with less than 2 stops.

```
SELECT f.FlightCode, al.Name, ArrivalTime
FROM Flight f NATURAL JOIN AirlineCompany al JOIN Airport ap1 ON
f.DepartingAirportID = ap1.AirportID JOIN Airport ap2 ON f.ArrivingAirportID =
ap2.AirportID
WHERE DepartingAirportID IN
(SELECT AirportID FROM Airport WHERE City = 'Shanghai')
AND ArrivingAirportID IN
(SELECT AirportID FROM Airport WHERE City = 'Beijing')
AND NumberOfStops < 2
ORDER BY ArrivalTime ASC
Limit 15;
```

```
mysql> SELECT f.FlightCode, al.Name, ArrivalTime
    -> FROM Flight f NATURAL JOIN AirlineCompany al JOIN Airport ap1 ON f.DepartingAirportID = ap1.AirportID JOIN Airport ap2 ON f.ArrivingAirportID = ap2.AirportID
    -> WHERE DepartingAirportID IN
    -> (SELECT AirportID FROM Airport WHERE City = 'Shanghai')
    -> AND ArrivingAirportID IN
    -> (SELECT AirportID FROM Airport WHERE City = 'Beijing')
    -> AND NumberOfStops < 2
    -> ORDER BY ArrivalTime ASC
    -> Limit 15;
+------------+--------------------+-------------+
| FlightCode | Name               | ArrivalTime |
+------------+--------------------+-------------+
|         11 | Delta Air Lines    | 2022-07-11  |
|         22 | Delta Air Lines    | 2022-07-12  |
|         59 | JetBlue            | 2022-07-12  |
|        248 | American Airlines  | 2022-07-12  |
|         13 | Delta Air Lines    | 2022-07-12  |
|         57 | Hawaiian Airlines  | 2022-07-13  |
|        116 | American Airlines  | 2022-07-13  |
|        881 | Southwest Airlines | 2022-07-13  |
+------------+--------------------+-------------+
8 rows in set (0.00 sec)

mysql>
```

Indexing analysis

1.

NO index:

```
mysql> explain analyze (SELECT FlightCode, MIN(TicketPrice)
    -> FROM Ticket NATURAL JOIN Flight
    -> WHERE TicketClass = 1 AND NumberOfStops < 4
    -> GROUP BY FlightCode)
    -> UNION
    -> (SELECT FlightCode, MIN(TicketPrice)
    -> FROM Ticket NATURAL JOIN Flight
    -> WHERE TicketClass = 2 AND NumberOfStops < 4
    -> GROUP BY FlightCode)
    -> UNION
    -> (SELECT FlightCode, MIN(TicketPrice)
    -> FROM Ticket NATURAL JOIN Flight
    -> WHERE TicketClass = 3 AND NumberOfStops < 4
    -> GROUP BY FlightCode)
    -> Limit 15
    -> ;
```

Original analysis

```
| -> Limit: 15 row(s)  (cost=2.50 rows=0) (actual time=0.001..0.003 rows=15 loops=1)
    -> Table scan on <union temporary>  (cost=2.50 rows=0) (actual time=0.001..0.001 rows=15 loops=1)
        -> Union materialize with deduplication  (cost=2.50..2.50 rows=0) (actual time=2.306..2.309 rows=15 loops=1)
            -> Limit table size: 15 unique row(s)
                -> Table scan on <temporary>  (actual time=0.001..0.002 rows=15 loops=1)
                    -> Aggregate using temporary table  (actual time=2.283..2.285 rows=15 loops=1)
                        -> Nested loop inner join  (cost=136.25 rows=100) (actual time=0.071..2.143 rows=201 loops=1)
                            -> Filter: (Ticket.TicketClass = 1)  (cost=101.25 rows=100) (actual time=0.041..0.416 rows=201 loops=1)
                                -> Table scan on Ticket  (cost=101.25 rows=1000) (actual time=0.039..0.329 rows=1000 loops=1)
                            -> Filter: (Flight.NumberOfStops < 4)  (cost=0.25 rows=1) (actual time=0.008..0.008 rows=1 loops=201)
                                -> Single-row index lookup on Flight using PRIMARY (FlightCode=Ticket.FlightCode)  (cost=0.25 rows=1) (actual time=0.008..0.008 rows=1 loops=201)
            -> Limit table size: 15 unique row(s)
                -> Table scan on <temporary>  (never executed)
                    -> Aggregate using temporary table  (never executed)
                        -> Nested loop inner join  (cost=136.25 rows=100) (never executed)
                            -> Filter: (Ticket.TicketClass = 2)  (cost=101.25 rows=100) (never executed)
                                -> Table scan on Ticket  (cost=101.25 rows=1000) (never executed)
                            -> Filter: (Flight.NumberOfStops < 4)  (cost=0.25 rows=1) (never executed)
                                -> Single-row index lookup on Flight using PRIMARY (FlightCode=Ticket.FlightCode)  (cost=0.25 rows=1) (never executed)
            -> Limit table size: 15 unique row(s)
                -> Table scan on <temporary>  (never executed)
                    -> Aggregate using temporary table  (never executed)
                        -> Nested loop inner join  (cost=136.25 rows=100) (never executed)
                            -> Filter: (Ticket.TicketClass = 3)  (cost=101.25 rows=100) (never executed)
                                -> Table scan on Ticket  (cost=101.25 rows=1000) (never executed)
                            -> Filter: (Flight.NumberOfStops < 4)  (cost=0.25 rows=1) (never executed)
                                -> Single-row index lookup on Flight using PRIMARY (FlightCode=Ticket.FlightCode)  (cost=0.25 rows=1) (never executed)
|
```

CREATE INDEX idx_tic_price ON Ticket(TicketPrice)

As we are finding the min price of each class, TicketPrice might be a good attribute to try.

```
| -> Limit: 15 row(s)  (cost=2.50 rows=0) (actual time=0.001..0.003 rows=15 loops=1)
    -> Table scan on <union temporary>  (cost=2.50 rows=0) (actual time=0.000..0.001 rows=15 loops=1)
        -> Union materialize with deduplication  (cost=2.50..2.50 rows=0) (actual time=0.935..0.938 rows=15 loops=1)
            -> Limit table size: 15 unique row(s)
                -> Table scan on <temporary>  (actual time=0.000..0.001 rows=15 loops=1)
                    -> Aggregate using temporary table  (actual time=0.924..0.926 rows=15 loops=1)
                        -> Nested loop inner join  (cost=136.25 rows=33) (actual time=0.100..0.841 rows=201 loops=1)
                            -> Filter: (Ticket.TicketClass = 1)  (cost=101.25 rows=100) (actual time=0.086..0.472 rows=201 loops=1)
                                -> Table scan on Ticket  (cost=101.25 rows=1000) (actual time=0.083..0.386 rows=1000 loops=1)
                            -> Filter: (Flight.NumberOfStops < 4)  (cost=0.25 rows=0) (actual time=0.001..0.002 rows=1 loops=201)
                                -> Single-row index lookup on Flight using PRIMARY (FlightCode=Ticket.FlightCode)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=201)
            -> Limit table size: 15 unique row(s)
                -> Table scan on <temporary>  (never executed)
                    -> Aggregate using temporary table  (never executed)
                        -> Nested loop inner join  (cost=136.25 rows=33) (never executed)
                            -> Filter: (Ticket.TicketClass = 2)  (cost=101.25 rows=100) (never executed)
                                -> Table scan on Ticket  (cost=101.25 rows=1000) (never executed)
                            -> Filter: (Flight.NumberOfStops < 4)  (cost=0.25 rows=0) (never executed)
                                -> Single-row index lookup on Flight using PRIMARY (FlightCode=Ticket.FlightCode)  (cost=0.25 rows=1) (never executed)
            -> Limit table size: 15 unique row(s)
                -> Table scan on <temporary>  (never executed)
                    -> Aggregate using temporary table  (never executed)
                        -> Nested loop inner join  (cost=136.25 rows=33) (never executed)
                            -> Filter: (Ticket.TicketClass = 3)  (cost=101.25 rows=100) (never executed)
                                -> Table scan on Ticket  (cost=101.25 rows=1000) (never executed)
                            -> Filter: (Flight.NumberOfStops < 4)  (cost=0.25 rows=0) (never executed)
                                -> Single-row index lookup on Flight using PRIMARY (FlightCode=Ticket.FlightCode)  (cost=0.25 rows=1) (never executed)
 |
```

It has significant improvements over no indexing.

CREATE INDEX idx_tic_class ON Ticket(`TicketClass`)
As we are grouping by TicketClass, TicketClass might be a good attribute to try.

```
| -> Limit: 15 row(s)  (cost=2.50 rows=0) (actual time=0.001..0.003 rows=15 loops=1)
    -> Table scan on <union temporary>  (cost=2.50 rows=0) (actual time=0.001..0.002 rows=15 loops=1)
        -> Union materialize with deduplication  (cost=2.50..2.50 rows=0) (actual time=1.859..1.862 rows=15 loops=1)
            -> Limit table size: 15 unique row(s)
                -> Table scan on <temporary>  (actual time=0.001..0.002 rows=15 loops=1)
                    -> Aggregate using temporary table  (actual time=1.846..1.848 rows=15 loops=1)
                        -> Nested loop inner join  (cost=94.20 rows=67) (actual time=1.246..1.749 rows=201 loops=1)
                            -> Index lookup on Ticket using idx_tic_class (TicketClass=1)  (cost=23.85 rows=201) (actual time=1.225..1.381 rows=201 loops=1)
                            -> Filter: (Flight.NumberOfStops < 4)  (cost=0.25 rows=0) (actual time=0.001..0.002 rows=1 loops=201)
                                -> Single-row index lookup on Flight using PRIMARY (FlightCode=Ticket.FlightCode)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=201)
            -> Limit table size: 15 unique row(s)
                -> Table scan on <temporary>  (never executed)
                    -> Aggregate using temporary table  (never executed)
                        -> Nested loop inner join  (cost=96.00 rows=68) (never executed)
                            -> Index lookup on Ticket using idx_tic_class (TicketClass=2)  (cost=24.25 rows=205) (never executed)
                            -> Filter: (Flight.NumberOfStops < 4)  (cost=0.25 rows=0) (never executed)
                                -> Single-row index lookup on Flight using PRIMARY (FlightCode=Ticket.FlightCode)  (cost=0.25 rows=1) (never executed)
            -> Limit table size: 15 unique row(s)
                -> Table scan on <temporary>  (never executed)
                    -> Aggregate using temporary table  (never executed)
                        -> Nested loop inner join  (cost=90.60 rows=64) (never executed)
                            -> Index lookup on Ticket using idx_tic_class (TicketClass=3)  (cost=23.05 rows=193) (never executed)
                            -> Filter: (Flight.NumberOfStops < 4)  (cost=0.25 rows=0) (never executed)
                                -> Single-row index lookup on Flight using PRIMARY (FlightCode=Ticket.FlightCode)  (cost=0.25 rows=1) (never executed)
 |
```

It has improvements over no indexing.

CREATE INDEX idx_tic_stops ON Flight(`NumberOfStops`)
As we finding the information of flights with less than 2 stops, `NumberOfStops` might be a good attribute to try.

```
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (cost=2.50 rows=0) (actual time=0.001..0.003 rows=15 loops=1)
    -> Table scan on <union temporary>  (cost=2.50 rows=0) (actual time=0.000..0.001 rows=15 loops=1)
        -> Union materialize with deduplication  (cost=2.50..2.50 rows=0) (actual time=0.874..0.878 rows=15 loops=1)
            -> Limit table size: 15 unique row(s)
                -> Table scan on <temporary>  (actual time=0.001..0.002 rows=15 loops=1)
                    -> Aggregate using temporary table  (actual time=0.861..0.863 rows=15 loops=1)
                        -> Nested loop inner join  (cost=136.25 rows=100) (actual time=0.043..0.779 rows=201 loops=1)
                            -> Filter: (Ticket.TicketClass = 1)  (cost=101.25 rows=100) (actual time=0.034..0.414 rows=201 loops=1)
                                -> Table scan on Ticket  (cost=101.25 rows=1000) (actual time=0.031..0.318 rows=1000 loops=1)
                            -> Filter: (Flight.NumberOfStops < 4)  (cost=0.25 rows=1) (actual time=0.001..0.002 rows=1 loops=201)
                                -> Single-row index lookup on Flight using PRIMARY (FlightCode=Ticket.FlightCode)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=201)
            -> Limit table size: 15 unique row(s)
                -> Table scan on <temporary>  (never executed)
                    -> Aggregate using temporary table  (never executed)
                        -> Nested loop inner join  (cost=136.25 rows=100) (never executed)
                            -> Filter: (Ticket.TicketClass = 2)  (cost=101.25 rows=100) (never executed)
                                -> Table scan on Ticket  (cost=101.25 rows=1000) (never executed)
                            -> Filter: (Flight.NumberOfStops < 4)  (cost=0.25 rows=1) (never executed)
                                -> Single-row index lookup on Flight using PRIMARY (FlightCode=Ticket.FlightCode)  (cost=0.25 rows=1) (never executed)
            -> Limit table size: 15 unique row(s)
                -> Table scan on <temporary>  (never executed)
                    -> Aggregate using temporary table  (never executed)
                        -> Nested loop inner join  (cost=136.25 rows=100) (never executed)
                            -> Filter: (Ticket.TicketClass = 3)  (cost=101.25 rows=100) (never executed)
                                -> Table scan on Ticket  (cost=101.25 rows=1000) (never executed)
                            -> Filter: (Flight.NumberOfStops < 4)  (cost=0.25 rows=1) (never executed)
                                -> Single-row index lookup on Flight using PRIMARY (FlightCode=Ticket.FlightCode)  (cost=0.25 rows=1) (never executed)
 |
+-------------------------------------------------------------------------------
```

It has improvements over no indexing.

2.

Original

```
---------------+
| -> Limit: 15 row(s)  (actual time=0.798..0.800 rows=8 loops=1)
    -> Sort: f.ArrivalTime, limit input to 15 row(s) per chunk  (actual time=0.798..0.799 rows=8 loops=1)
        -> Stream results  (cost=230.30 rows=5) (actual time=0.144..0.774 rows=8 loops=1)
            -> Nested loop inner join  (cost=230.30 rows=5) (actual time=0.139..0.764 rows=8 loops=1)
                -> Nested loop inner join  (cost=211.90 rows=53) (actual time=0.136..0.746 rows=11 loops=1)
                    -> Nested loop inner join  (cost=193.50 rows=53) (actual time=0.130..0.726 rows=11 loops=1)
                        -> Nested loop inner join  (cost=175.10 rows=53) (actual time=0.126..0.667 rows=11 loops=1)
                            -> Nested loop inner join  (cost=156.71 rows=53) (actual time=0.098..0.607 rows=11 loops=1)
                                -> Filter: (Airport.City = 'Shanghai')  (cost=101.50 rows=100) (actual time=0.051..0.440 rows=30 loops=1)
                                    -> Table scan on Airport  (cost=101.50 rows=1000) (actual time=0.044..0.324 rows=1000 loops=1)
                                -> Filter: (f.NumberOfStops < 2)  (cost=0.39 rows=1) (actual time=0.005..0.005 rows=0 loops=30)
                                    -> Index lookup on f using DepartingAirportID (DepartingAirportID=Airport.AirportID)  (cost=0.39 rows=2) (actual time=0.004..0.005 rows=1 loops=
30)
                            -> Single-row index lookup on al using PRIMARY (AirlineID=f.AirlineID)  (cost=0.25 rows=1) (actual time=0.005..0.005 rows=1 loops=11)
                        -> Single-row index lookup on ap1 using PRIMARY (AirportID=Airport.AirportID)  (cost=0.25 rows=1) (actual time=0.005..0.005 rows=1 loops=11)
                    -> Single-row index lookup on ap2 using PRIMARY (AirportID=f.ArrivingAirportID)  (cost=0.25 rows=1) (actual time=0.001..0.002 rows=1 loops=11)
                -> Filter: (Airport.City = 'Beijing')  (cost=0.25 rows=0) (actual time=0.001..0.001 rows=1 loops=11)
                    -> Single-row index lookup on Airport using PRIMARY (AirportID=f.ArrivingAirportID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=11)
 |
+-------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
---------------+
1 row in set (0.16 sec)
```

CREATE INDEX idx_ap_city ON `Airport`(`City`)

As we are finding the flight that matching the arriving city and departing city, `City` might be a good attribute to try.

```
-------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=0.360..0.362 rows=8 loops=1)
    -> Sort: f.ArrivalTime, limit input to 15 row(s) per chunk  (actual time=0.360..0.361 rows=8 loops=1)
        -> Stream results  (cost=26.66 rows=1) (actual time=0.097..0.330 rows=8 loops=1)
            -> Nested loop inner join  (cost=26.66 rows=1) (actual time=0.093..0.321 rows=8 loops=1)
                -> Nested loop inner join  (cost=26.38 rows=1) (actual time=0.091..0.312 rows=8 loops=1)
                    -> Nested loop inner join  (cost=26.11 rows=1) (actual time=0.088..0.278 rows=8 loops=1)
                        -> Nested loop inner join  (cost=25.83 rows=1) (actual time=0.079..0.240 rows=8 loops=1)
                            -> Nested loop inner join  (cost=20.31 rows=16) (actual time=0.070..0.197 rows=11 loops=1)
                                -> Index lookup on Airport using idx_ap_city (City='Shanghai')  (cost=3.75 rows=30) (actual time=0.017..0.022 rows=30 loops=1)
                                -> Filter: (f.NumberOfStops < 2)  (cost=0.40 rows=1) (actual time=0.005..0.006 rows=0 loops=30)
                                    -> Index lookup on f using DepartingAirportID (DepartingAirportID=Airport.AirportID)  (cost=0.40 rows=2) (actual time=0.005..0.005 rows=1 loops=
30)
                            -> Filter: (Airport.City = 'Beijing')  (cost=0.25 rows=0) (actual time=0.004..0.004 rows=1 loops=11)
                                -> Single-row index lookup on Airport using PRIMARY (AirportID=f.ArrivingAirportID)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=11)
                        -> Single-row index lookup on al using PRIMARY (AirlineID=f.AirlineID)  (cost=0.38 rows=1) (actual time=0.004..0.004 rows=1 loops=8)
                    -> Single-row index lookup on ap1 using PRIMARY (AirportID=Airport.AirportID)  (cost=0.38 rows=1) (actual time=0.004..0.004 rows=1 loops=8)
                -> Single-row index lookup on ap2 using PRIMARY (AirportID=f.ArrivingAirportID)  (cost=0.38 rows=1) (actual time=0.001..0.001 rows=1 loops=8)
 |
+-------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-------------------------------------------------------------------------------+
1 row in set (0.16 sec)
```

It is improved from no indexing.

CREATE INDEX idx_tic_stops ON Flight(NumberOfStops)

As we finding the flights with less than 2 stops, NumberOfStops might be a good attribute to try.

```
---------------+
| -> Limit: 15 row(s)  (actual time=2.149..2.150 rows=8 loops=1)
    -> Sort: f.ArrivalTime, limit input to 15 row(s) per chunk  (actual time=2.148..2.149 rows=8 loops=1)
        -> Stream results  (cost=228.69 rows=5) (actual time=0.106..2.128 rows=8 loops=1)
            -> Nested loop inner join  (cost=228.69 rows=5) (actual time=0.100..2.113 rows=8 loops=1)
                -> Nested loop inner join  (cost=210.70 rows=51) (actual time=0.097..2.091 rows=11 loops=1)
                    -> Nested loop inner join  (cost=192.70 rows=51) (actual time=0.095..2.074 rows=11 loops=1)
                        -> Nested loop inner join  (cost=174.70 rows=51) (actual time=0.091..2.055 rows=11 loops=1)
                            -> Nested loop inner join  (cost=156.71 rows=51) (actual time=0.084..2.004 rows=11 loops=1)
                                -> Filter: (Airport.City = 'Shanghai')  (cost=101.50 rows=100) (actual time=0.045..0.452 rows=30 loops=1)
                                    -> Table scan on Airport  (cost=101.50 rows=1000) (actual time=0.039..0.322 rows=1000 loops=1)
                                -> Filter: (f.NumberOfStops < 2)  (cost=0.39 rows=1) (actual time=0.051..0.051 rows=0 loops=30)
                                    -> Index lookup on f using DepartingAirportID (DepartingAirportID=Airport.AirportID)  (cost=0.39 rows=2) (actual time=0.050..0.051 rows=1 loops=
30)
                            -> Single-row index lookup on al using PRIMARY (AirlineID=f.AirlineID)  (cost=0.25 rows=1) (actual time=0.004..0.004 rows=1 loops=11)
                        -> Single-row index lookup on ap1 using PRIMARY (AirportID=Airport.AirportID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=11)
                    -> Single-row index lookup on ap2 using PRIMARY (AirportID=f.ArrivingAirportID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=11)
                -> Filter: (Airport.City = 'Beijing')  (cost=0.25 rows=0) (actual time=0.002..0.002 rows=1 loops=11)
                    -> Single-row index lookup on Airport using PRIMARY (AirportID=f.ArrivingAirportID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=11)
 |
+----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------+
1 row in set (0.16 sec)
```

No difference, perhaps the index of NumberOfStops only increase the complexity.

CREATE INDEX idx_tic_arr_time ON Flight(ArrivalTime)

As we finding the flights arrive before some time, ArrivalTime might be a good attribute to try.

```
---------------+
| -> Limit: 15 row(s)  (actual time=0.722..0.723 rows=8 loops=1)
    -> Sort: f.ArrivalTime, limit input to 15 row(s) per chunk  (actual time=0.721..0.722 rows=8 loops=1)
        -> Stream results  (cost=230.30 rows=5) (actual time=0.123..0.701 rows=8 loops=1)
            -> Nested loop inner join  (cost=230.30 rows=5) (actual time=0.117..0.690 rows=8 loops=1)
                -> Nested loop inner join  (cost=211.90 rows=53) (actual time=0.113..0.645 rows=11 loops=1)
                    -> Nested loop inner join  (cost=193.50 rows=53) (actual time=0.111..0.630 rows=11 loops=1)
                        -> Nested loop inner join  (cost=175.10 rows=53) (actual time=0.107..0.614 rows=11 loops=1)
                            -> Nested loop inner join  (cost=156.71 rows=53) (actual time=0.101..0.586 rows=11 loops=1)
                                -> Filter: (Airport.City = 'Shanghai')  (cost=101.50 rows=100) (actual time=0.053..0.426 rows=30 loops=1)
                                    -> Table scan on Airport  (cost=101.50 rows=1000) (actual time=0.046..0.315 rows=1000 loops=1)
                                -> Filter: (f.NumberOfStops < 2)  (cost=0.39 rows=1) (actual time=0.005..0.005 rows=0 loops=30)
                                    -> Index lookup on f using DepartingAirportID (DepartingAirportID=Airport.AirportID)  (cost=0.39 rows=2) (actual time=0.004..0.005 rows=1 loops=
30)
                            -> Single-row index lookup on al using PRIMARY (AirlineID=f.AirlineID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=11)
                        -> Single-row index lookup on ap1 using PRIMARY (AirportID=Airport.AirportID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=11)
                    -> Single-row index lookup on ap2 using PRIMARY (AirportID=f.ArrivingAirportID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=11)
                -> Filter: (Airport.City = 'Beijing')  (cost=0.25 rows=0) (actual time=0.004..0.004 rows=1 loops=11)
                    -> Single-row index lookup on Airport using PRIMARY (AirportID=f.ArrivingAirportID)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=11)
 |
+----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------+
1 row in set (0.16 sec)
```

No difference, perhaps the index of ArrivalTime only increase the complexity.