FINAL PROJECT REPORT

EXECUTIVE SUMMARY

Topic Area	Description	Points
Database Design	This part should include a logical database design (for the relational model), using normalization to control redundancy and integrity constraints for data quality.	30
Query Writing	This part is another chance to write SQL queries, explore transactions, and even do some database programming for stored procedures.	25
Performance Tuning	In this section, you can capitalize and extend your prior experiments with indexing, optimizer modes, partitioning, parallel execution and any other techniques you want to further explore.	20
Other Topics	Here you are free to explore any other topics of interest. Suggestions include DBA scripts, database security, interface design, data visualization, data mining, and NoSQL databases.	25

TEAM

1. Alphonse Aloia
2. Krishna Sai Surya Teja Basetty
3. Malavika Parakkat Byju
4. Sahil Shah
5. Shruthi Priya Athikam

Here is a list of different things we explored as part of above stated sections:

1. We made a completely new design from scratch so that we could balance the realistic simulation and also keep up with deliverable constraints.

   We tried to normalize the design to a state that we thought was apt.

   We generated some realistic data using a python utility and also generated more sample data that we used as place holders for other sections.

   We looked into Data integrity and came up with some constraints.

2. We explored some queries on our tables.

   We explored some stored procedures that we put in place which we later used as part of a C# windows app form that could be used as a user interface for our booking system.

   We explored triggers and sequences and tested them on our database.

   Screenshots for all these are attached part of the report.

3. In this section we aimed to look into indexing and parallelism and its effects on the database. We made use of the huge placeholder data that we put in in the data generation part.

   We tested different parallel execution commands to see its effects on the performance of the database queries.

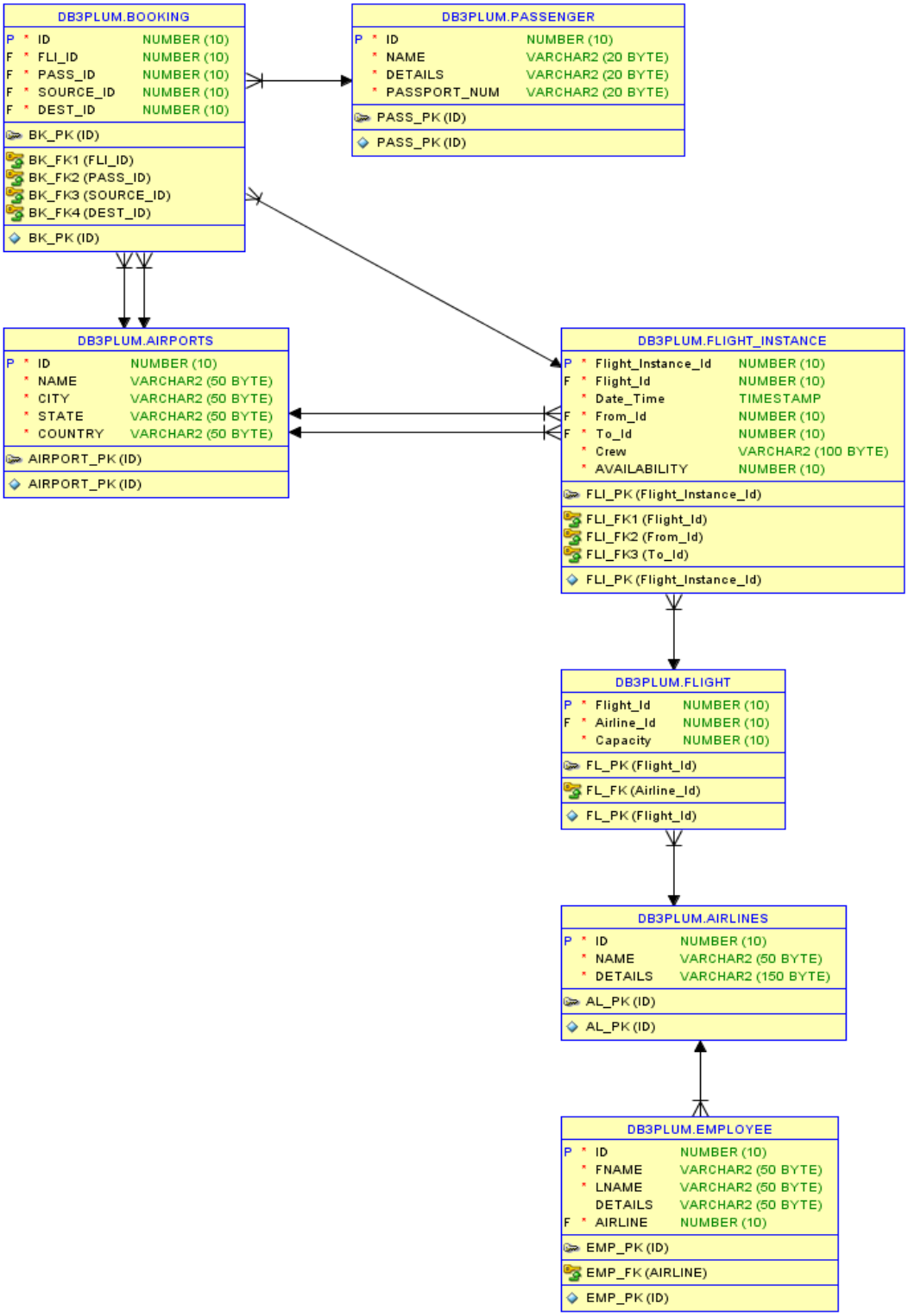
   We tried indexing on our biggest table and observed the time difference indexing makes.

4. We created a windows forms application for admin and booking portal to explore the user interface section

   We made some reports to explore Data Visualization techniques, we made use of all the data.

   We explored some beginner level DBA scripts and corresponding screenshots.

SECTION-1 DATABASE DESIGN



ERD Diagram of The Airline Management System

Our project is based on a real-time Airline Management System. We have explored two areas pertaining to the Airlines Industry – one from the Administration perspective where an admin can schedule flights for participating airlines. The other is from a passenger perspective where a customer can book flights based on their requirements from the available flights. As part of this system we have designed and implemented the database in our group schema account DB3PLUM containing seven tables which interact with each other with one-to-one or many-to-one relations, thus creating a Relational Database Management System. To accurately mimic our project to that of a real-time portal we have used C# MS .NET WINDOWS FORMS to build a desktop application and have integrated it with our implemented database. We have created an application where admins and passengers can perform the appropriate functions and interact with our database. Details of the application are included in the subsequent sections. The following section explains the structure and purpose of each table in our schema.

1. **AIRPORTS** – This table holds the data pertaining to all airports in the world which might be the source or destination for the flight instances scheduled by the admin. It's structure is as shown below:

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	ID	NUMBER(10,0)	No	(null)	1	(null)
2	NAME	VARCHAR2(50 BYTE)	No	(null)	2	(null)
3	CITY	VARCHAR2(50 BYTE)	No	(null)	3	(null)
4	STATE	VARCHAR2(50 BYTE)	No	(null)	4	(null)
5	COUNTRY	VARCHAR2(50 BYTE)	No	(null)	5	(null)

It has an ID column of NUMBER data type which acts as a Primary Key column to the table, uniquely defining each airport. The NAME, CITY, STATE and COUNTRY columns are of VARCHAR type and have a NOT NULL constraint enabled on them. Integrity and other constraints are detailed further in subsequent sections. The airport table has a one-to-many relationship with the Flight_Instance table where each flight_instance has a source airport id and a destination airport id as foreign keys referencing the ID column of airports table. An airport can be a source or destination for multiple flights and thus the many side of the relation. It also has a one-to-many relation

with the Booking table where each booking made by a passenger will have source and destination airport printed on their boarding pass.

A. PASSENGERS – This table holds the passenger information who have booked flights through our airline system. We have loaded sample data into this table using C# code about which we'll learn more in the Data Generation section. Alternatively we can also insert data into this table using the C# application we have developed. The structure is as below:

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	ID	NUMBER(10,0)	No	(null)	1	(null)
2	NAME	VARCHAR2(20 BYTE)	No	(null)	2	(null)
3	DETAILS	VARCHAR2(20 BYTE)	No	(null)	3	(null)
4	PASSPORT_NUM	VARCHAR2(20 BYTE)	No	(null)	4	(null)

It has an ID column of Number data type which being a Primary Key uniquely identifies each passenger and the NAME, DETAILS, PASSPORT_NUM columns are Varchar type of data with Not Null constraints enabled on them. This table has a one-to-many relationship with the Booking table where each booking record has a PASS_ID foreign key referencing to the ID column of this table detailing about the passenger who has made that booking. This functionality allows the same passenger to make multiple bookings.

B. AIRLINES – This table holds the details about the multiple airline giants present in the Aviation Industry like Etihad, Spirit, Delta and so on. Its structure is as below:

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	ID	NUMBER(10,0)	No	(null)	1	(null)
2	NAME	VARCHAR2(50 BYTE)	No	(null)	2	(null)
3	DETAILS	VARCHAR2(150 BYTE)	No	(null)	3	(null)

It has a PK ID column to uniquely define each airline and the NAME and DETAILS are of Varchar type with NOT NULL constraints enabled on them. This table has a one-to-many relation with both Employee table and Flight table. The employee table holds information about the employees available for each airline. The flight table holds records about the multiple physical flights available in each airline company.

C. EMPLOYEE – This table holds employee information of each airline company like pilots, co-pilots and cabin-crew. Its structure is as shown below:

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	ID	NUMBER(10,0)	No	(null)	1	(null)
2	FNAME	VARCHAR2(50 BYTE)	No	(null)	2	(null)
3	LNAME	VARCHAR2(50 BYTE)	No	(null)	3	(null)
4	DETAILS	VARCHAR2(50 BYTE)	Yes	(null)	4	(null)
5	AIRLINE	NUMBER(10,0)	No	(null)	5	(null)

The ID column is a PK column of Number type. The FNAME, LNAME show the first and last names of each employee which cannot be null. The DETAILS column tells about the designation of the employee which can be pilot, co-pilot or cabin-crew. The AIRLINE column is a FK referencing the ID column of Airlines table thus having a many-to-one relation between Employee and Airlines tables. An employee in this table has to be from only one airline company and an airline company can have multiple employees working for them.

D. FLIGHT – This table holds details about the physical flights present in each airline company say Boeing-787, Jet-89 and so on. These are just physical flights and do not have a date-time , source or destination attached to it. It has the following structure :

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	Flight_Id	NUMBER(10,0)	No	(null)	1	(null)
2	Airline_Id	NUMBER(10,0)	No	(null)	2	(null)
3	Capacity	NUMBER(10,0)	No	(null)	3	(null)

The Flight_Id column is a PK and the Airline_Id is a FK referring the ID column of the Airlines table showing as to which airline company that flight belongs to and thus a many-to-one relation with the Airlines table. The Capacity column is of number type and holds the number of passengers that flight can accommodate. This table also has a one-to-many relation with the Flight_Instance table where one particular flight can have multiple instances like it can fly from NY to LA and NY to TA and at multiple times during the day.

E. FLIGHT_INSTANCE – This table holds the records the passengers actually see while booking flights. The table stores date-time , source , destination and availability attributes which a passenger can use to book a flight_instance. The structure is shown below:

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	Flight_Instance_Id	NUMBER(10,0)	No	(null)	1	(null)
2	Flight_Id	NUMBER(10,0)	No	(null)	2	(null)
3	Date_Time	TIMESTAMP(6)	No	(null)	3	(null)
4	From_Id	NUMBER(10,0)	No	(null)	4	(null)
5	To_Id	NUMBER(10,0)	No	(null)	5	(null)
6	Crew	VARCHAR2(100 BYTE)	No	(null)	6	(null)
7	AVAILABILITY	NUMBER(10,0)	No	(null)	7	(null)

The Flight_Instance_Id is a PK column, Flight_Id is a FK referencing the Flight_Id column of the Flight table which indicates what physical flight is being used for the trip such as Boeing, Airbus etc. Thus there's a many-to-one relation with the Flight table. Using this Flight_Id and an inner join with Flight and Airlines tables we can see which airline company this flight_instance belongs to. The Date_Time shows the schedule of the flight_instance. From_Id and To_Id are Foreign keys referencing the ID column of the airports table showing the departure and arrival airports of the flight_instance and thus have a many-to-one relation with airports table. Crew is a Varchar field which holds the names of employees who function as pilot, co-pilot and cabin-crew for that particular flight_instance. Availability tells about the remaining seats present in that flight_instance.

F. BOOKING – The Booking table holds all boarding information of the booked passengers till now. Its similar to boarding pass of all customers and has all columns present on a real time boarding pass. The structure is as below:

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	ID	NUMBER(10,0)	No	(null)	1	(null)
2	FLI_ID	NUMBER(10,0)	No	(null)	2	(null)
3	PASS_ID	NUMBER(10,0)	No	(null)	3	(null)
4	SOURCE_ID	NUMBER(10,0)	No	(null)	4	(null)
5	DEST_ID	NUMBER(10,0)	No	(null)	5	(null)

The ID column is PK uniquely identifying each boarding pass. The FLI_ID is a FK referencing the Flight_Instance_Id column of the Flight_Instance table and thus has a

many-to-one relation with flight_instance table. PASS_ID is also a FK detailing about the passenger who made that booking and references the ID column of the Passenger table. The SOURCE_ID and DEST_ID are Foreign keys and refers the ID column of Airports table and tell the source airport and destination airport for that boarding pass and thus has a many-to-one relation with Airports table.

1.1 NORMALIZATION

We have ensured that our database design is normalized unto the 3rd Normalization Form. Normalization is the process of minimizing data redundancy and have proper joins to be used to retrieve required data set rather than storing bulk data in a single table. Below we have provided a detail breakdown of what each normalization is and how we ensured it.

a) First Normalization Form: A database is said to be first normalized if no relation contains composite or multi-valued attribute. Here while loading data into our tables, we have made sure that each relation has only single value for any attribute. Moreover, a look at the attributes we have for our relations show that they are well-defined and do not offer any ambiguity in terms of the number of values it can hold.

b) Second Normalization Form: A database is said to be second normalized if all non-key attributes depend on the entire composite candidate key rather than subset of the candidate key. In our database there is no question of violating second normalization as we haven't incorporated any composite candidate keys. If we look at our ERD above all tables have a single attribute PK which uniquely identifies each relation in the table and thus all non-PK attributes depend completely on the PK attribute.

c) Third Normalization Form: A database has to have no inter-dependencies between non-prime attributes to be third normalized. Here we have ensured that no non-prime attribute depends on any other non-prime attribute. For example looking at Flight_Instance table – each attribute of a relation such as flight_id, source, destination, date-time the flight is scheduled, current availability, crew for that schedule all depend on that particular flight_instance_id and only that. There are no inter-dependencies.

1.2 DATA INTEGRITY

Integrity constraints help us to preserve data integrity as well as incorporate any restrictions or business requirements on the data our database relations can hold. As part of our database design we have explored Primary Keys, Foreign Keys and Check constraints. Here we explain these constraints using our Booking table. The constraints enforced and the DDL statements used are given below.

	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	R_OWNER	R_TABLE_NAME	R_CONSTRAINT_NAME	DELETE_RULE	STATUS
1	BK_FK1	Foreign_Key	(null)	DB3PLUM	FLIGHT_INSTANCE	FLI_PK	NO ACTION	ENABLED
2	BK_FK2	Foreign_Key	(null)	DB3PLUM	PASSENGER	PASS_PK	NO ACTION	ENABLED
3	BK_FK3	Foreign_Key	(null)	DB3PLUM	AIRPORTS	AIRPORT_PK	NO ACTION	ENABLED
4	BK_FK4	Foreign_Key	(null)	DB3PLUM	AIRPORTS	AIRPORT_PK	NO ACTION	ENABLED
5	BK_PK	Primary_Key	(null)	(null)	(null)	(null)	(null)	ENABLED
6	SYS_C00128213	Check	"ID" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED
7	SYS_C00128214	Check	"FLI_ID" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED
8	SYS_C00128215	Check	"PASS_ID" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED
9	SYS_C00128216	Check	"SOURCE_ID" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED
10	SYS_C00128217	Check	"DEST_ID" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED

```
CREATE TABLE "DB3PLUM"."BOOKING"
   (    "ID" NUMBER(10,0) NOT NULL ENABLE,
        "FLI_ID" NUMBER(10,0) NOT NULL ENABLE,
        "PASS_ID" NUMBER(10,0) NOT NULL ENABLE,
        "SOURCE_ID" NUMBER(10,0) NOT NULL ENABLE,
        "DEST_ID" NUMBER(10,0) NOT NULL ENABLE,
         CONSTRAINT "BK_PK" PRIMARY KEY ("ID"),
         CONSTRAINT "BK_FK1" FOREIGN KEY ("FLI_ID") REFERENCES "DB3PLUM"."FLIGHT_INSTANCE" ("Flight_Instance_Id") ENABLE,
         CONSTRAINT "BK_FK3" FOREIGN KEY ("SOURCE_ID") REFERENCES "DB3PLUM"."AIRPORTS" ("ID") ENABLE,
         CONSTRAINT "BK_FK4" FOREIGN KEY ("DEST_ID") REFERENCES "DB3PLUM"."AIRPORTS" ("ID") ENABLE,
         CONSTRAINT "BK_FK2" FOREIGN KEY ("PASS_ID") REFERENCES "DB3PLUM"."PASSENGER" ("ID") ENABLE
   );
```

<u>Primary Key</u>: Primary key constraints have been enabled on the ID columns of all tables in our database and is our source for Foreign Keys used in other tables. As seen above the “BK_PK” constraint name denotes a Primary key constraint on the “ID” column of our Booking table.

<u>Foreign Key</u>: Foreign keys are the way to enforce Referential Integrity and here we can see that the “BK_FK1”, “BK_FK2”, “BK_FK3” and “BK_FK4” constraints denoted the Foreign keys enabled on “FLI_ID” column which actually refers the PK Flight_Instance_Id column of the Flight_Instance table, “SOURCE_ID” and “DEST_ID” refers to the ID columns of the Airports table, “PASS_ID” refers to the “ID” column of the “Passenger” table. These Foreign Keys ensure that a boarding pass cannot contain a passenger id, departure or arrival airports or Flight_Instance not registered with us in our database.

Check Constraints: Here we have enabled the NOT NULL constraints on almost all attributes in our database so as to avoid any plausible inconsistencies when the data is queried. Here we can see that our Booking table cannot take a NULL value for any attribute.

1.3 DATA GENERATION AND LOADING

For the data generator I wrote a few quick python scripts. See below

SCRIPT #1

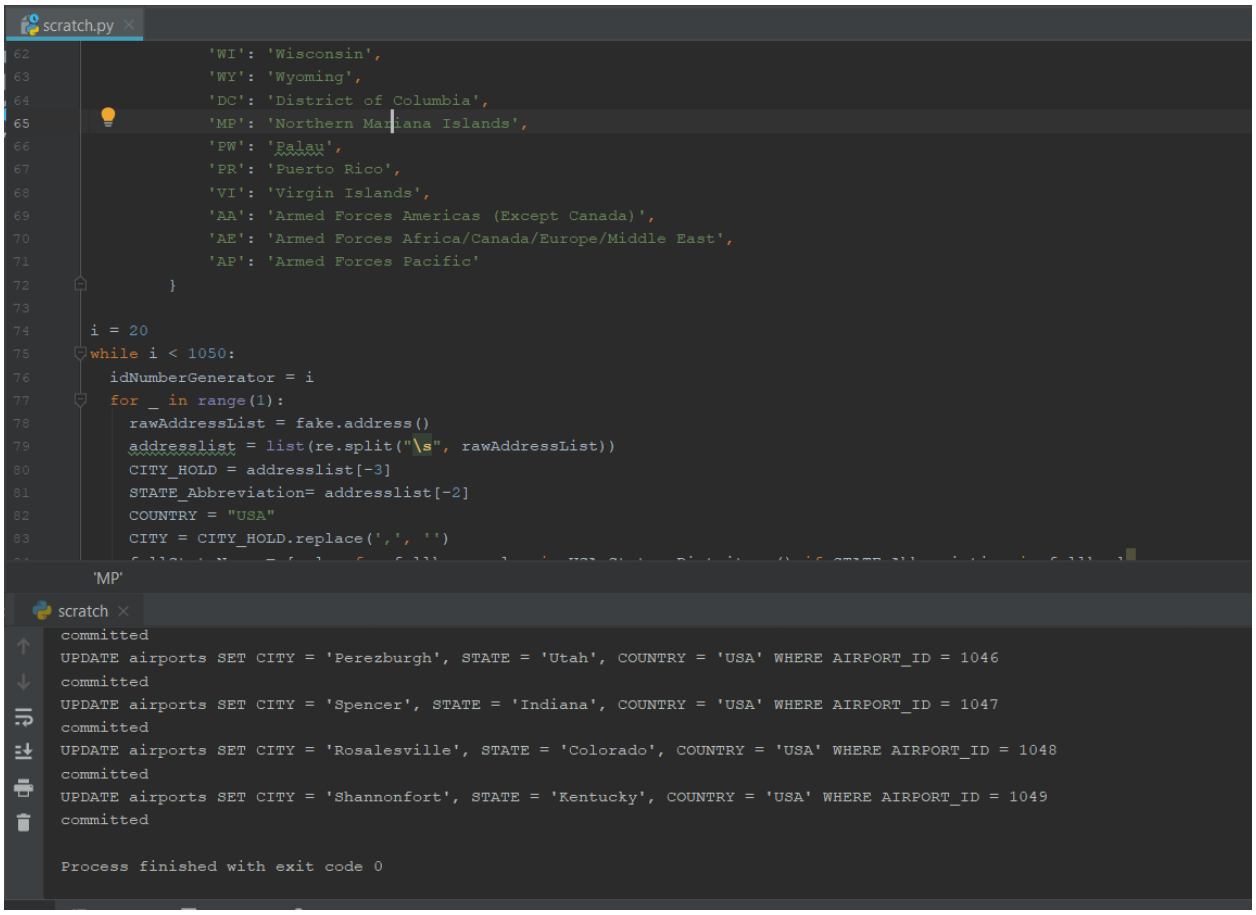
```
from faker import Faker

fake = Faker()

import re

import cx_Oracle

connect = cx_Oracle.connect('DB3PLUM/db3group@reade.forest.usf.edu:1521/cdb9')

print(connect.version)

cursor = connect.cursor()

USA_States_Dict = {

          'AL': 'Alabama',

           'AK': 'Alaska',

           'AZ': 'Arizona',

           'AR': 'Arkansas',

           'CA': 'California',

           'CO': 'Colorado',

           'CT': 'Connecticut',

           'DE': 'Delaware',

           'FL': 'Florida',

           'GA': 'Georgia',

           'HI': 'Hawaii',

           'ID': 'Idaho',
```

```
'IL': 'Illinois',
'IN': 'Indiana',
'IA': 'Iowa',
'KS': 'Kansas',
'KY': 'Kentucky',
'LA': 'Louisiana',
'ME': 'Maine',
'MD': 'Maryland',
'MA': 'Massachusetts',
'MI': 'Michigan',
'MN': 'Minnesota',
'MS': 'Mississippi',
'MO': 'Missouri',
'MT': 'Montana',
'NE': 'Nebraska',
'NV': 'Nevada',
'NH': 'New Hampshire',
'NJ': 'New Jersey',
'NM': 'New Mexico',
'NY': 'New York',
'NC': 'North Carolina',
'ND': 'North Dakota',
'OH': 'Ohio',
'OK': 'Oklahoma',
'OR': 'Oregon',
'PA': 'Pennsylvania',
'RI': 'Rhode Island',
```

```
            'SC': 'South Carolina',

            'SD': 'South Dakota',

            'TN': 'Tennessee',

            'TX': 'Texas',

            'UT': 'Utah',

            'VT': 'Vermont',

            'VA': 'Virginia',

            'WA': 'Washington',

            'WV': 'West Virginia',

            'WI': 'Wisconsin',

            'WY': 'Wyoming',

            'DC': 'District of Columbia',

            'MP': 'Northern Mariana Islands',

            'PW': 'Palau',

            'PR': 'Puerto Rico',

            'VI': 'Virgin Islands',

            'AA': 'Armed Forces Americas (Except Canada)',

            'AE': 'Armed Forces Africa/Canada/Europe/Middle East',

            'AP': 'Armed Forces Pacific'

        }

i = 20

while i < 1050:

 idNumberGenerator = i

 for _ in range(1):

   rawAddressList = fake.address()

   addresslist = list(re.split("\s", rawAddressList))

   CITY_HOLD = addresslist[-3]
```

```
   STATE_Abbreviation= addresslist[-2]

   COUNTRY = "USA"

   CITY = CITY_HOLD.replace(',', '')

       fullStateName = [value for fullkey, value in USA_States_Dict.items() if
STATE_Abbreviation in fullkey];

   regexStateName = re.sub("[^a-zA-Z]","", str(fullStateName))

    sql_statement = "UPDATE airports SET CITY = '"+CITY+"', STATE = '"+regexStateName+
"',"" COUNTRY = '"+COUNTRY+"' WHERE AIRPORT_ID = "+str(idNumberGenerator);

   print(sql_statement)

   cursor.execute(sql_statement)

   connect.commit()

   print("committed")

 i += 1

cursor.close()
```

```
connect.close()
```



SCRIPT #2

```
import random

from faker import Faker

fake = Faker()

import re

import cx_Oracle

connect = cx_Oracle.connect('DB3PLUM/db3group@reade.forest.usf.edu:1521/cdb9')

print(connect.version)

cursor = connect.cursor()
```

```
i = 1

while i < 10:

 idNumberGenerator = i

 detailGenerator = random.randint(1, 3)

 if detailGenerator == 1:

   detailType = "pilot"

 if detailGenerator == 2:

   detailType = "co-pilot"

 if detailGenerator == 3:

   detailType = "cabin-crew"

 airlineNumberGenerator = random.randint(1, 53)

 for _ in range(1):

   rawNameList = fake.name()

   nameList = list(re.split("\s", rawNameList))

   sql_statement = "UPDATE employee SET "+ "FNAME = '" + nameList[0] + "', LNAME = '"
+ nameList[1] + "'," + "DETAILS = '" + str(detailType) + "'," + " AIRLINE = '" +
str(airlineNumberGenerator) + "'" +" WHERE employee_id = "+ str(idNumberGenerator)

   print(sql_statement)

   cursor.execute(sql_statement)

   connect.commit()

   print("committed")

 i += 1

cursor.close()

connect.close()
```

```python
import re
import cx_Oracle
connect = cx_Oracle.connect('DB3PLUM/db3group@reade.forest.usf.edu:1521/cdb9')
print(connect.version)
cursor = connect.cursor()

i = 1
while i < 10:
  idNumberGenerator = i
  detailGenerator = random.randint(1, 3)
  if detailGenerator == 1:
    detailType = "pilot"
  if detailGenerator == 2:
    detailType = "co-pilot"
  if detailGenerator == 3:
    detailType = "cabin-crew"
  airlineNumberGenerator = random.randint(1, 53)
  for _ in range(1):
    rawNameList = fake.name()
    nameList = list(re.split("\s", rawNameList))
    sql_statement = "UPDATE employee SET "+ "FNAME = '" + nameList[0] + "', LNAME = '" + nameList[1] + "'," + "DETAILS = '" + str(detailType) + "',"
    print(sql_statement)
    cursor.execute(sql_statement)
```

```
committed
UPDATE employee SET FNAME = 'Helen', LNAME = 'Hill',DETAILS = 'co-pilot', AIRLINE = '21' WHERE employee_id = 6
committed
UPDATE employee SET FNAME = 'Charles', LNAME = 'Pittman',DETAILS = 'pilot', AIRLINE = '1' WHERE employee_id = 7
committed
UPDATE employee SET FNAME = 'Frank', LNAME = 'Reyes',DETAILS = 'cabin-crew', AIRLINE = '12' WHERE employee_id = 8
committed
UPDATE employee SET FNAME = 'Paul', LNAME = 'Walton',DETAILS = 'co-pilot', AIRLINE = '37' WHERE employee_id = 9
committed

Process finished with exit code 0
```

1.3.2 Data Generation Using Coding:

For realistic data we used above python scripts, but in order to check some performance tuning cases, we need a huge amount of data. For that purpose we wrote code to generate insert statements and print them into an SQL file. A sample SQL file is also attached as part of the project ZIP. These insert statements are loaded with arbitrary namesake values that act as placeholders.

We used c# language to generate these statements and code appropriate realistic data. As part of this we have inserted over 100000 records in passaneer, over 10000 records in each of employee and fight tables.

We aim for these tables in the 3rd section when we plan to explore more into performance tuning scenarios and also the Visualization part of 4th section where the different charts that are part of this report are generated using this data.

Some sample screenshots from each table are attached below for reference.

User Interface - WINDOWS FORMS APPLICATION

We built Windows forms in C# and connected them to an Oracle database. The form focuses on two major roles:

- Passenger
- Admin

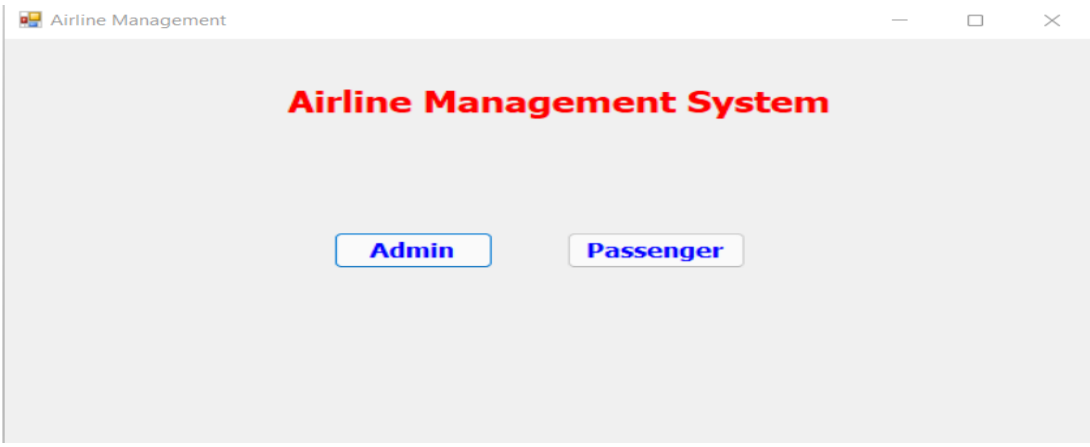
Admin can do two operations through the form:

- View the capacity and availability of the flights
- Add Flight Instances

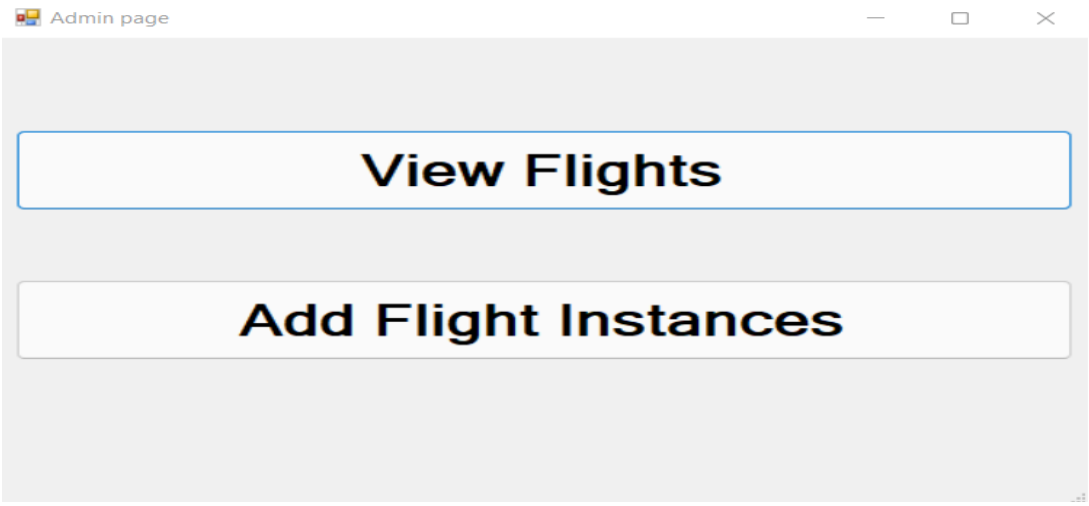
Passenger can perform two operations through the form:

- Book Flights
- View Booking Details

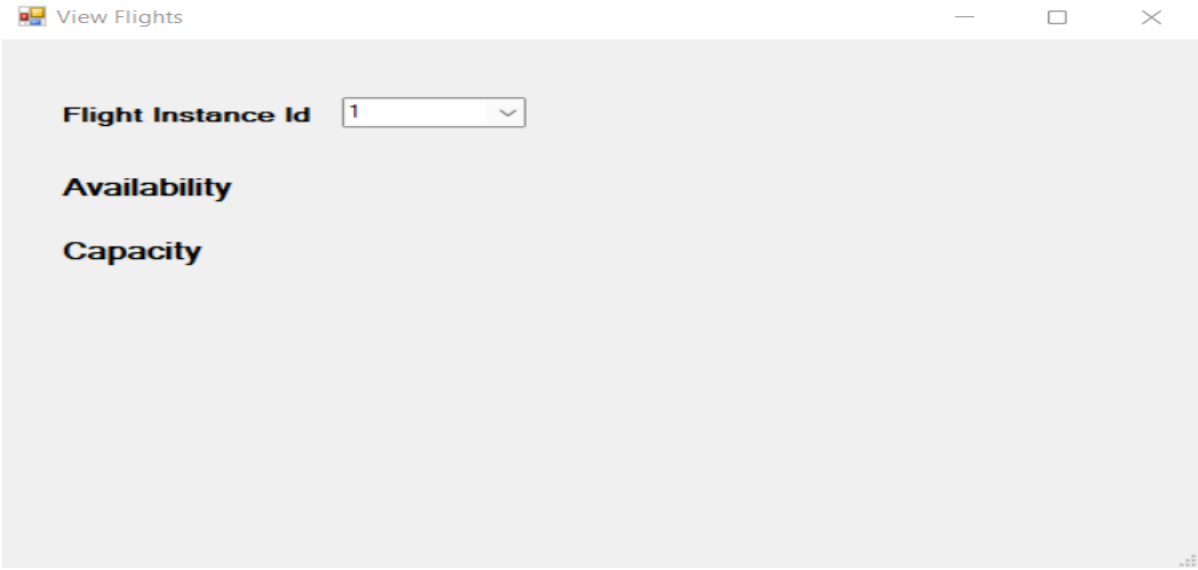
Admin



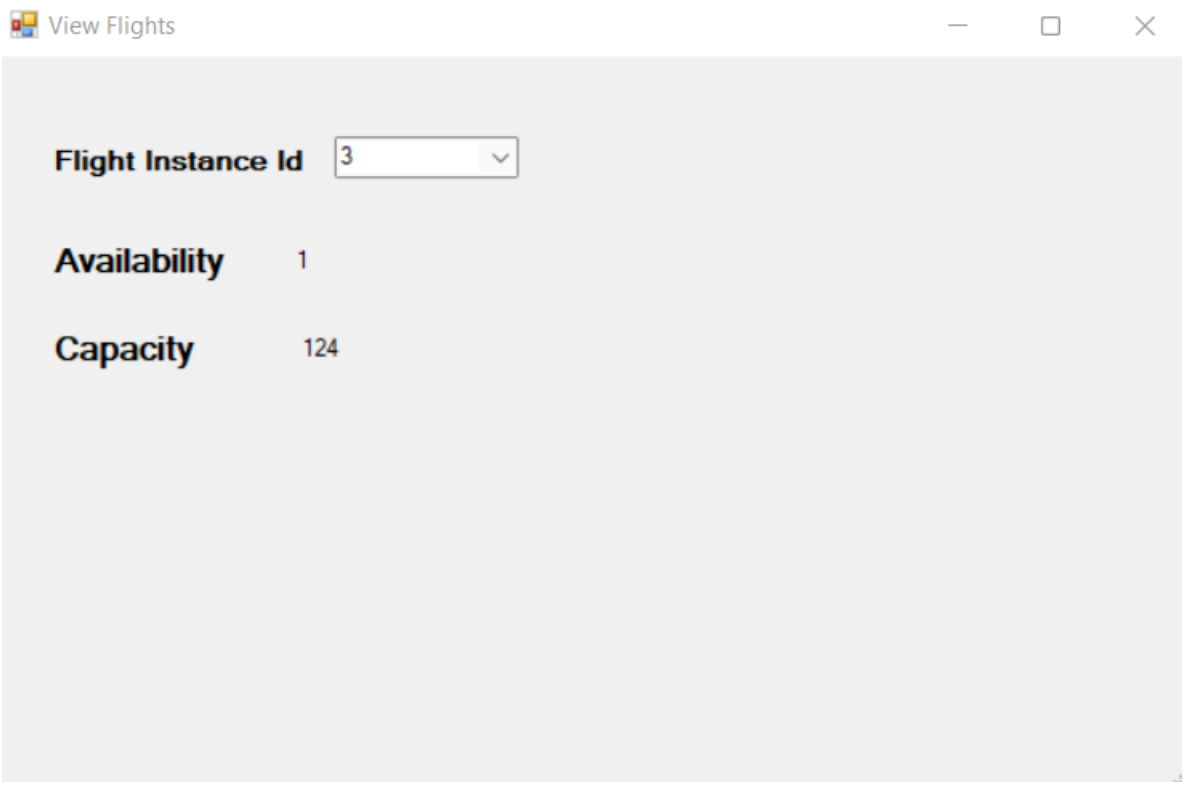
Click on Admin button in the above form. The admin page is displayed as shown below.



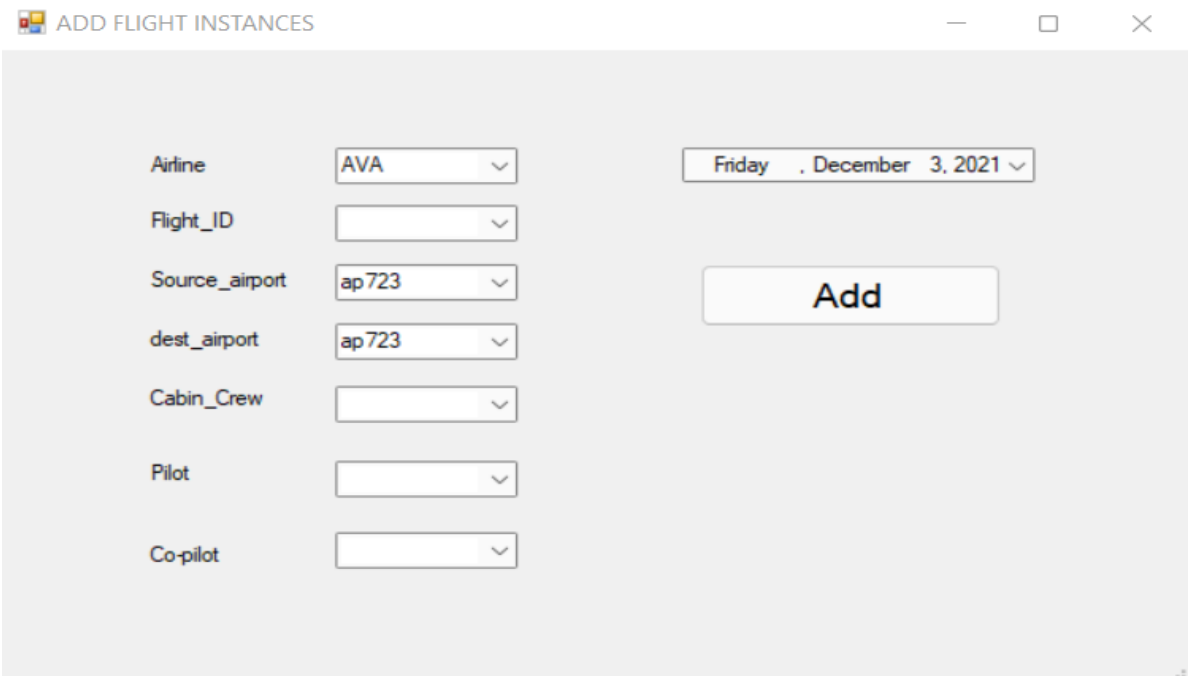
Click on View Flights Button in the above form and the “View Flight” page is displayed, as shown below.



Select the flight instance Id, and the corresponding Flight's availability and capacity will be displayed.



From the admin page, click the Add Flight Instances button. The page "Add flight instances" appears.

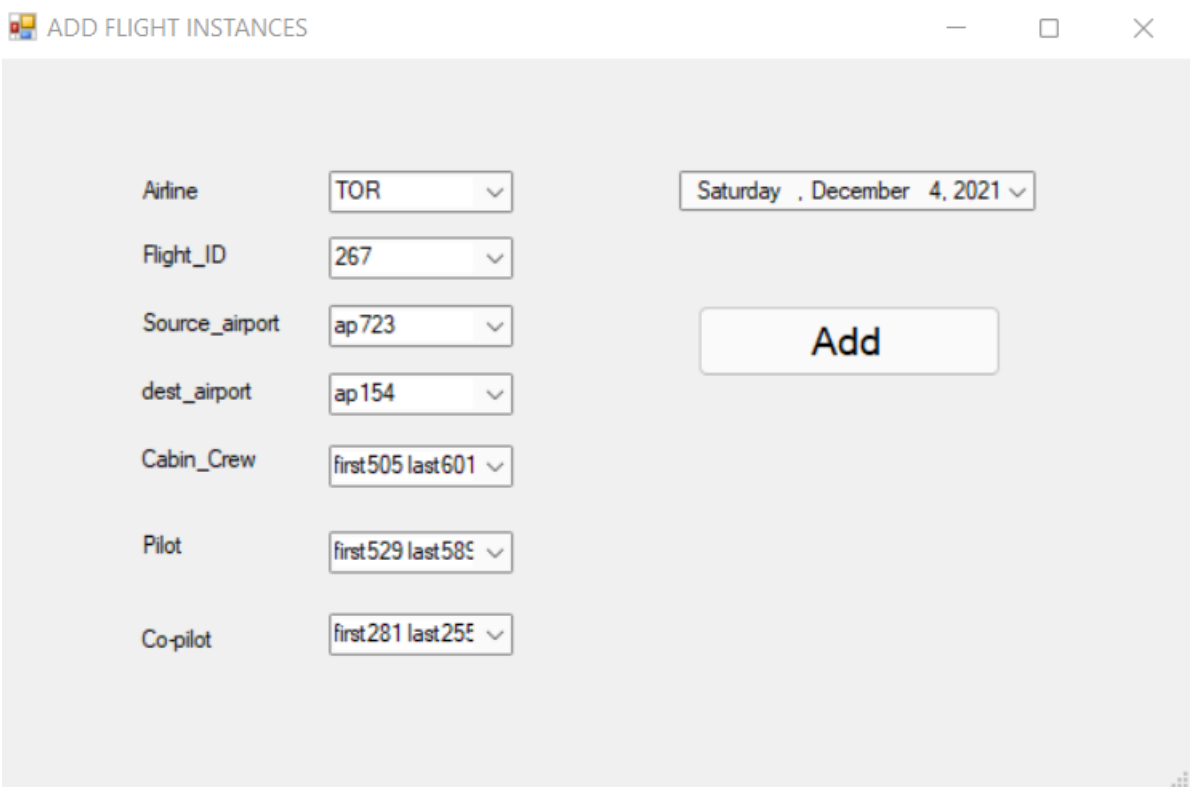


Flight Instance table before adding Flight instance is shown below:

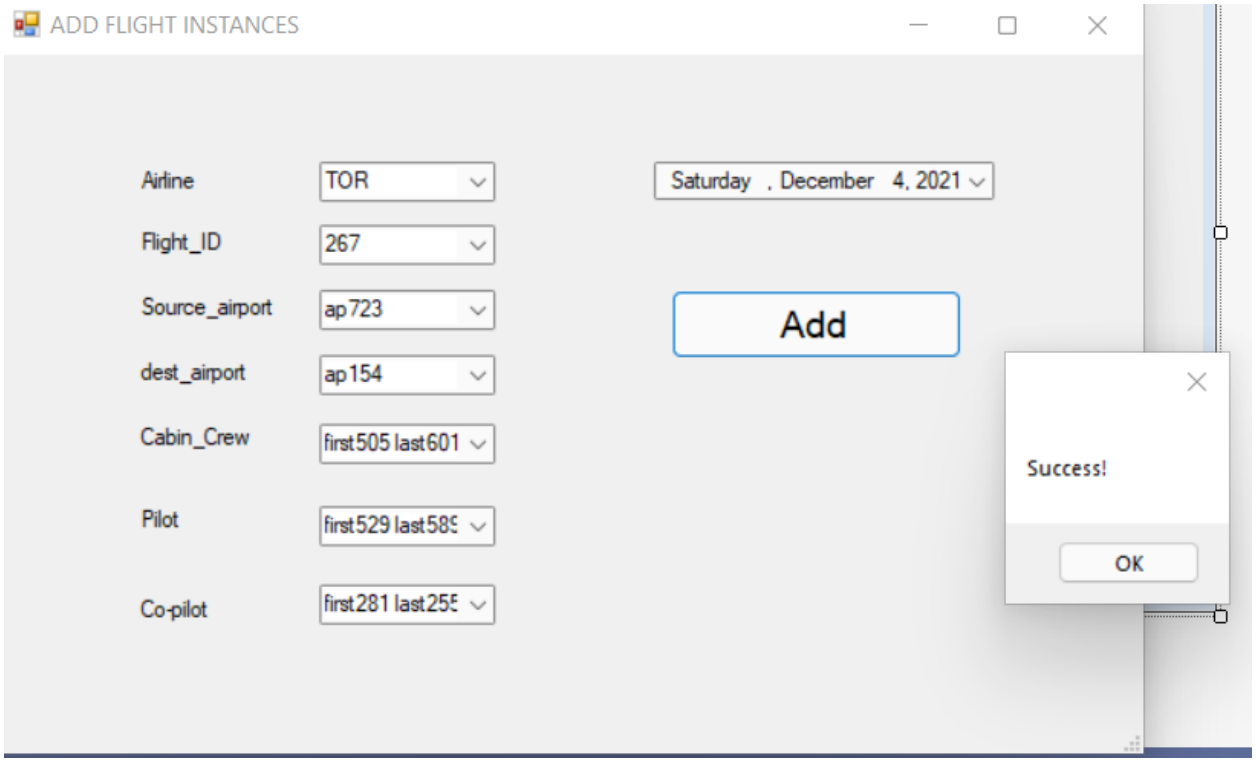
	Flight_Instance_Id	Flight_Id	Date_Time	From_Id	To_Id	Crew	AVAILABILITY
1	5	5	02-DEC-21 12.00.00.000000000 AM	1	31	SHRUTHI+PRIYA	1000
2	6	389	23-DEC-21 04.46.16.000000000 PM	341	344	first894 last82--first230 last455--first879 last600	700
3	2	5	30-NOV-21 10.00.00.174000000 PM	1	31	6	1
4	3	6	01-DEC-21 10.01.25.601000000 PM	1	31	7	1
5	4	7	01-DEC-21 10.02.29.756000000 PM	1	31	8	0
6	7	60	02-DEC-21 08.28.46.000000000 PM	341	344	first966 last12--first339 last210--first204 last169	800
7	1	1	30-NOV-21 08.53.39.712000000 PM	1	31	5	100

Adding Flight instance through the windows Form:

Using the dropdown, enter the information for Airline,Flight Id,Source airport,Dest airport,Cabin crew,Pilot,Coo-Pilot, and Date. After you've input all of your information, click the "Add" button.



After clicking add button we get a Message box on successful insertion as shown below.

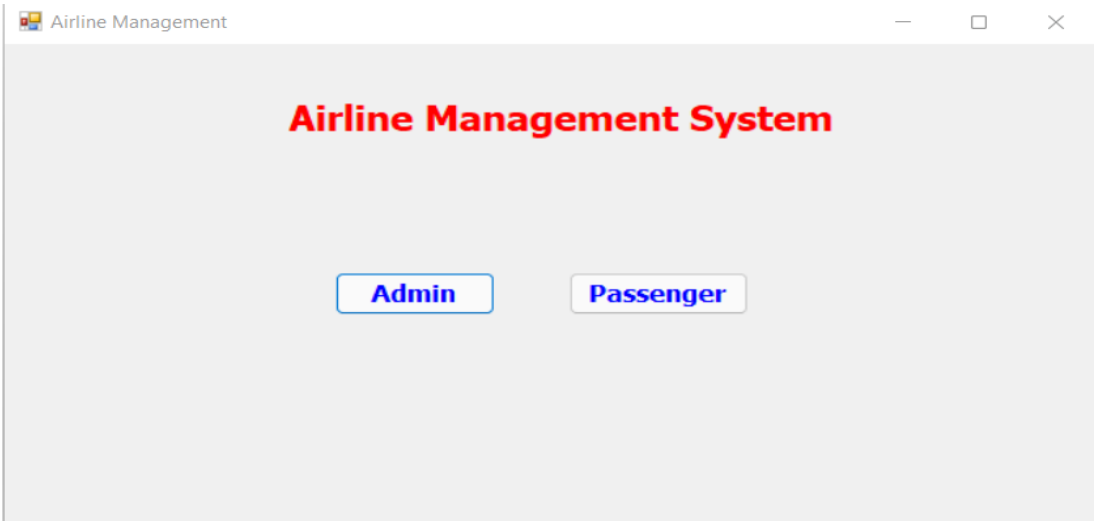


After inserting the Flight instance through the form:

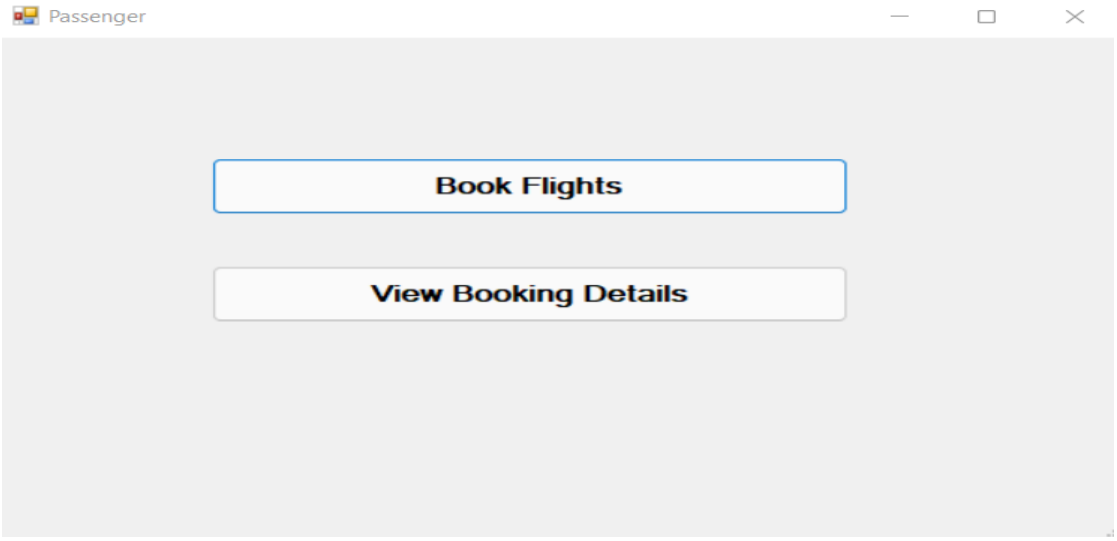
	Flight_Instance_Id	Flight_Id	Date_Time	From_Id	To_Id	Crew	AVAILABILITY
1	5	5	02-DEC-21 12.00.00.000000000 AM	1	31	SHRUTHI+PRIYA	1000
2	6	389	23-DEC-21 04.46.16.000000000 PM	341	344	first894 last82--first230 last455--first879 last600	700
3	2	5	30-NOV-21 10.00.00.174000000 PM	1	31	6	1
4	3	6	01-DEC-21 10.01.25.601000000 PM	1	31	7	1
5	4	7	01-DEC-21 10.02.29.756000000 PM	1	31	8	0
6	7	60	02-DEC-21 08.28.46.000000000 PM	341	344	first966 last12--first339 last210--first204 last169	800
7	8	267	04-DEC-21 04.01.33.000000000 PM	334	342	first529 last589--first281 last255--first505 last601	300
8	1	1	30-NOV-21 08.53.39.712000000 PM	1	31	5	100

The record with Instance ID 8 is successfully inserted.

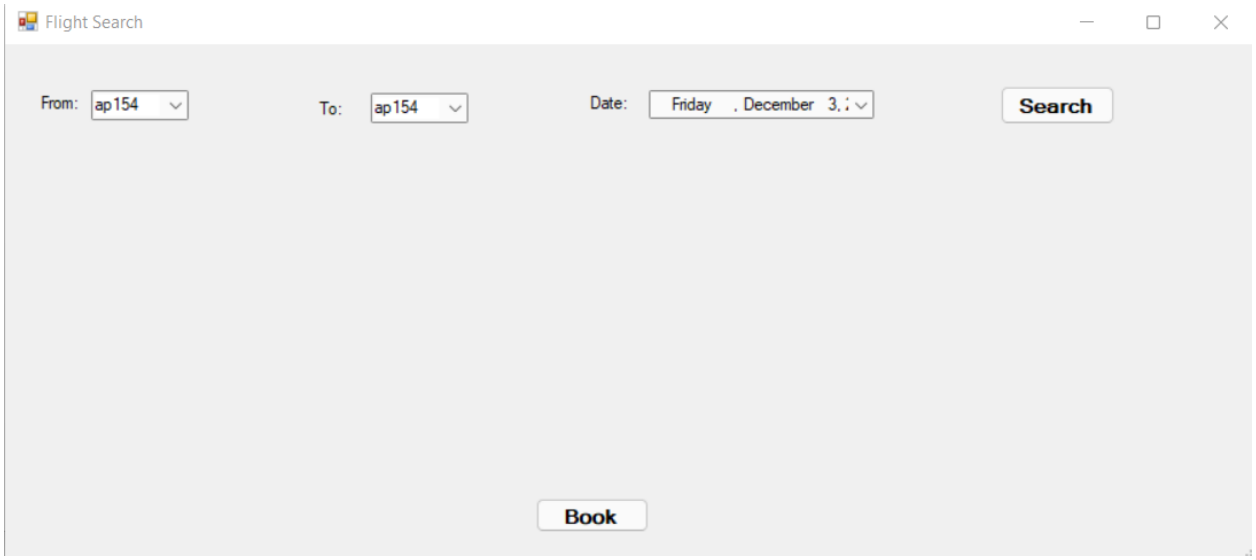
Passenger



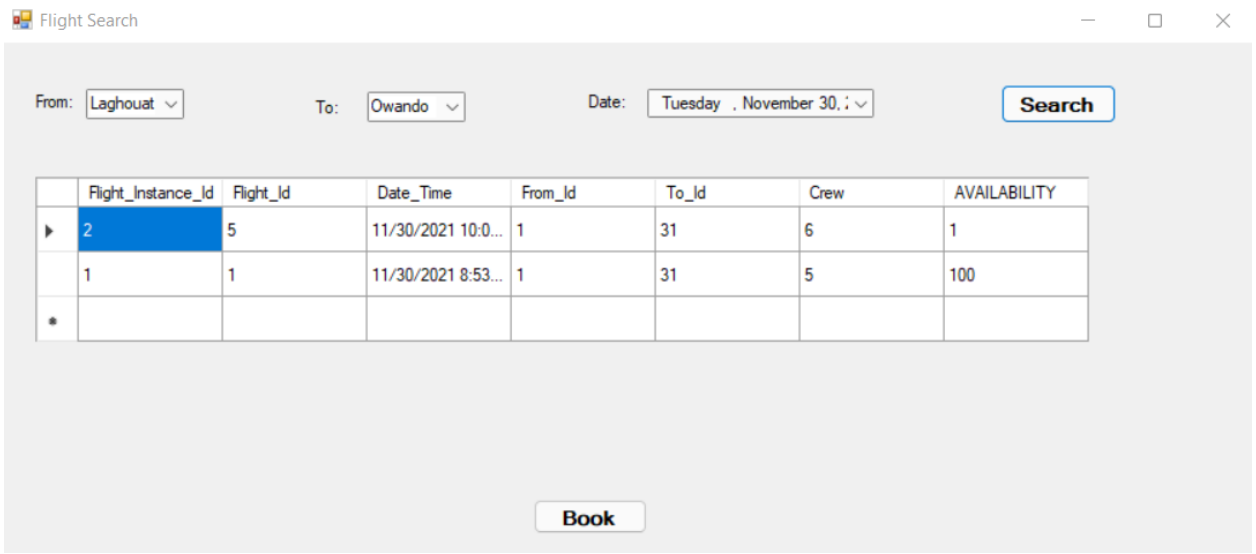
Click on the Passenger Button in the above page.



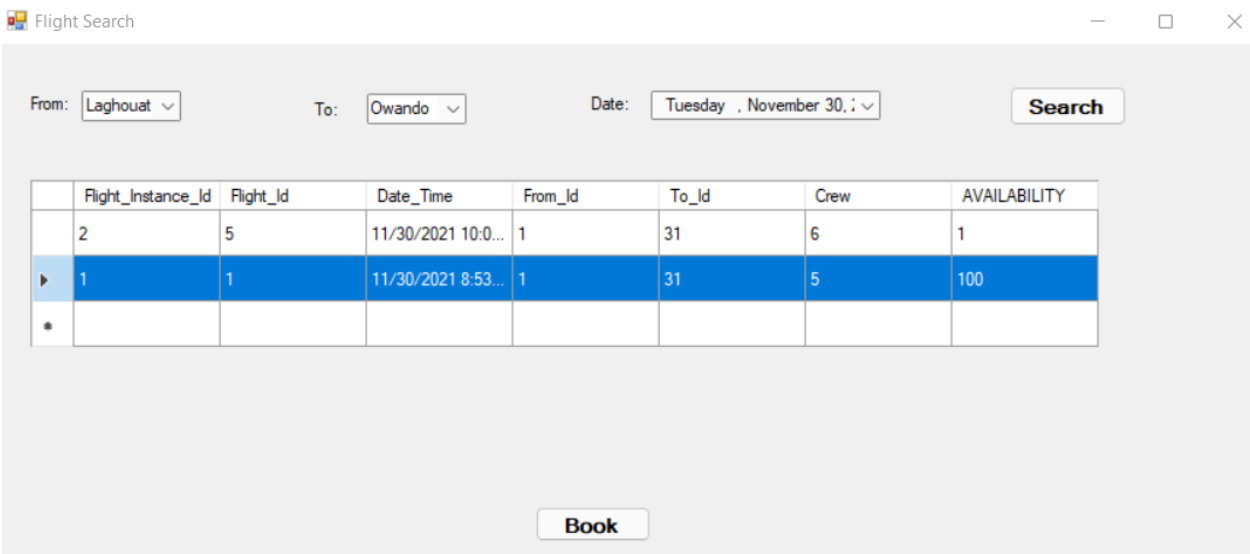
Click on “Book Flights” in the above page. View Flights page is displayed as shown below:



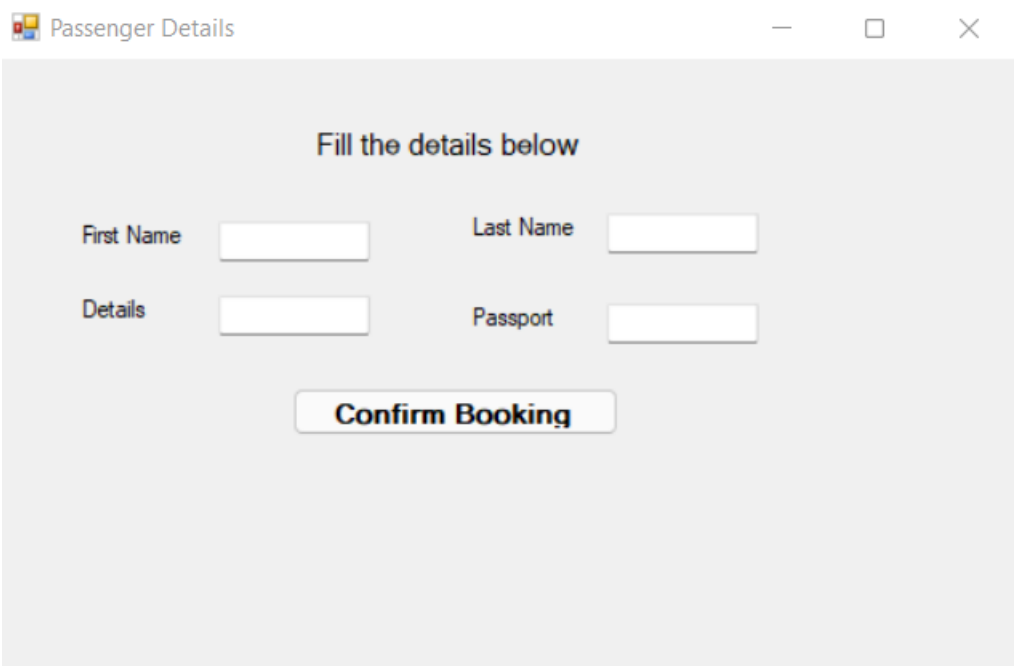
Enter the From Airport, To Airport, and Date information, then click Search. All flights departing from the given source and destination airports on the selected day will be presented as follows:



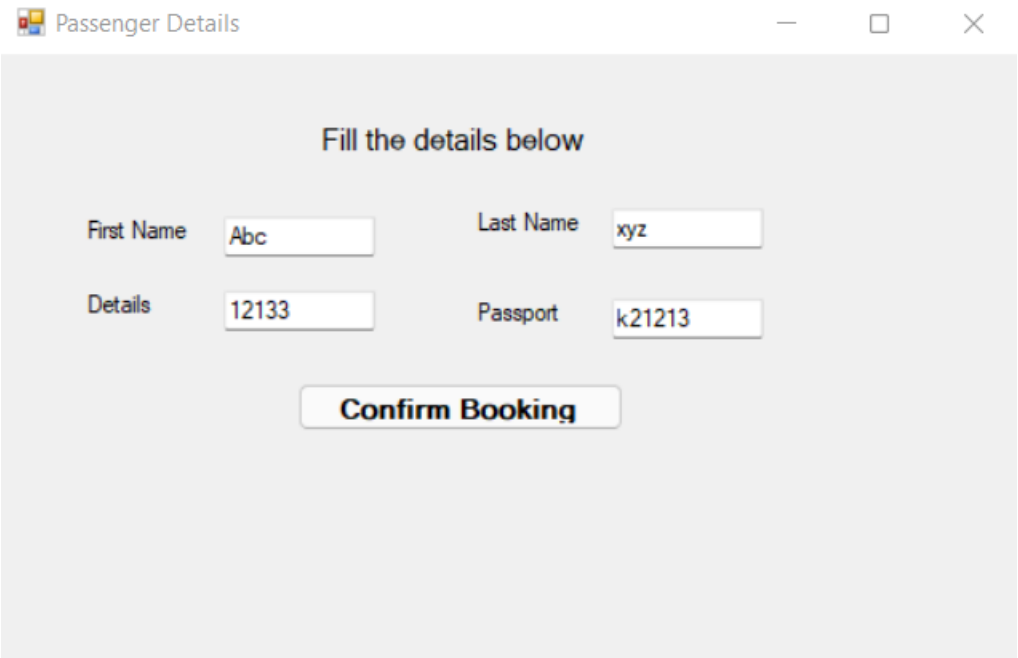
Select any flight you want to book and press the "Book" button.



The passenger details page appears after pressing the "Book" button.



Enter all the details and click on confirm booking.



Before clicking “confirm booking “:

Passenger table:

	ID	NAME	DETAILS	PASSPORT_NUM
1	100055	stevejohnson	test@gm...	k223444
2	100054	captainamerica	test@gm...	ak11232
3	100053	steveroger	test2@g...	k223444
4	100052	SamRoger	test@gm...	k223445
5	100051	NULLSFSDFS	NULLSFD...	NULLSDFSFSDF
6	100050	name132	Details270	876-19-6496
7	100049	name630	Details947	381-45-4309
8	100048	name726	Details133	802-24-3273
9	100047	name297	Details121	614-16-7064
10	100046	name617	Details168	270-88-9421
11	100045	name816	Details556	905-95-3034
12	100044	name269	Details564	468-12-9442
13	100043	name996	Details46	881-55-2565
14	100042	name955	Details117	340-33-3612
15	100041	name965	Details452	667-79-2906
16	100040	name5	Details735	158-11-8311
17	100039	name177	Details139	431-34-8915

Booking table:

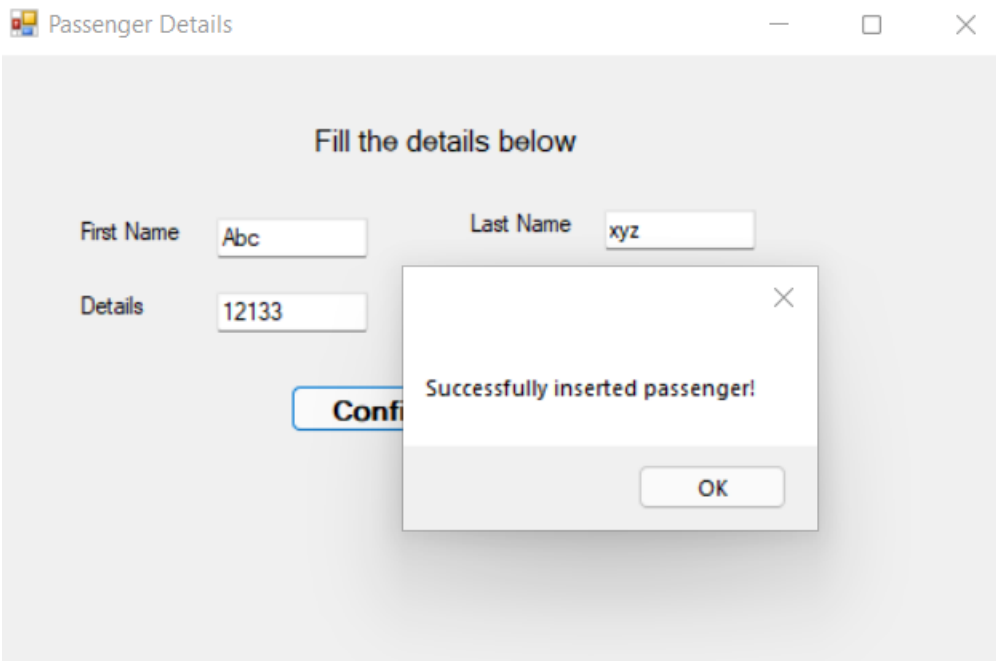
	ID	FLI_ID	PASS_ID	SOURCE_ID	DEST_ID
1	5	1	6	1	31
2	7	1	100053	1	31
3	3	1	6	1	31
4	2	1	6	1	31
5	1	1	1	1	2
6	8	1	100054	1	31
7	9	4	100055	1	31
8	6	1	100052	1	31

Availability for the flight:

	Flight_Instance_Id	Flight_Id	Date_Time	From_Id	To_Id	Crew	AVAILABILITY
1	5	5	02-DEC-21 12.00.00.000000000 AM	1	31	SHRUTHI+PRIYA	1000
2	6	389	23-DEC-21 04.46.16.000000000 PM	341	344	first894 last82--first230 last455--first879 last600	700
3	2	5	30-NOV-21 10.00.00.174000000 PM	1	31	6	1
4	3	6	01-DEC-21 10.01.25.601000000 PM	1	31	7	1
5	4	7	01-DEC-21 10.02.29.756000000 PM	1	31	8	0
6	7	60	02-DEC-21 08.28.46.000000000 PM	341	344	first966 last12--first339 last210--first204 last169	800
7	8	267	04-DEC-21 04.01.33.000000000 PM	334	342	first529 last589--first281 last255--first505 last601	300
8	1	1	30-NOV-21 08.53.39.712000000 PM	1	31	5	100

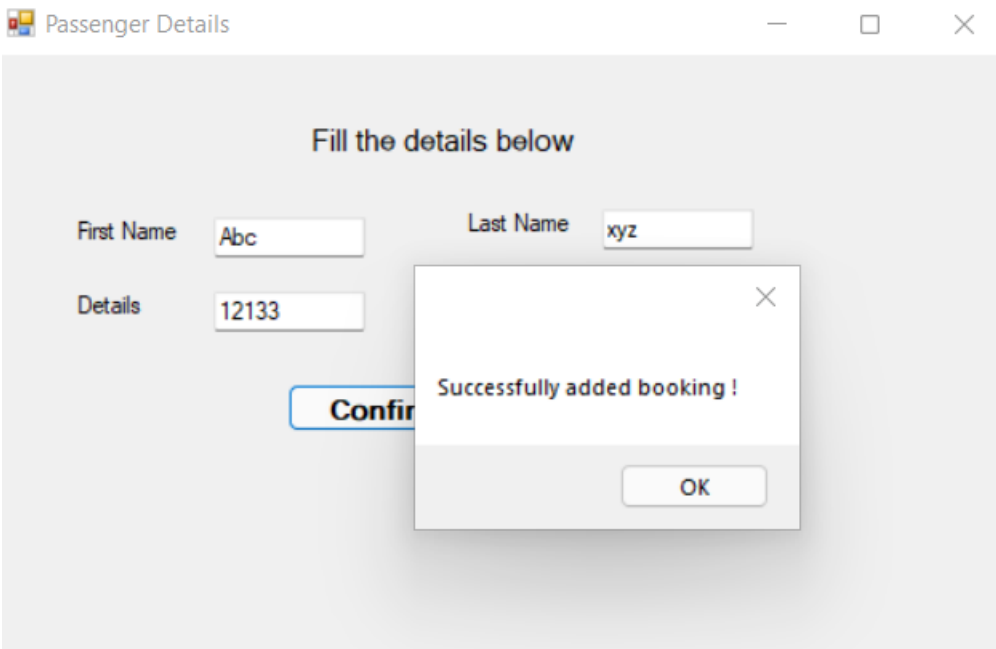
After clicking confirm booking:

In the Windows Form:



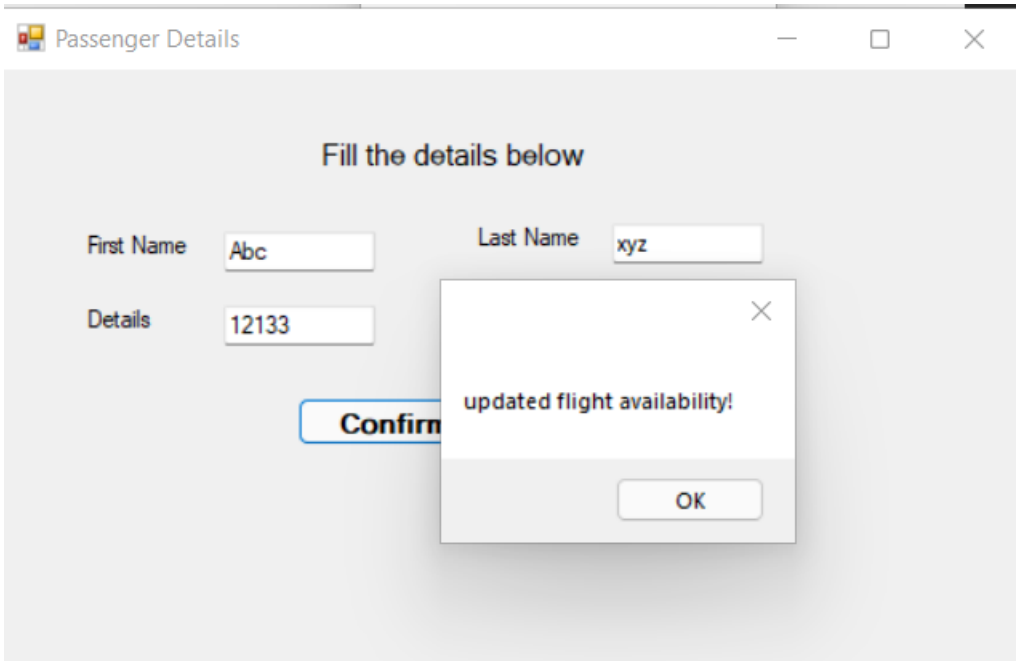
In the DB:

	ID	NAME	DETAILS	PASSPORT_NUM
1	100056	Abcxyz	12133	k21213
2	100055	stevejohnson	test@gm...	k223444
3	100054	captainamerica	test@gm...	ak11232
4	100053	steveroger	test2@g...	k223444
5	100052	SamRoger	test@gm...	k223445
6	100051	NULLSFSDFS	NULLSFD...	NULLSDFSFSDF
7	100050	name132	Details270	876-19-6496
8	100049	name630	Details947	381-45-4309
9	100048	name726	Details133	802-24-3273
10	100047	name297	Details121	614-16-7064
11	100046	name617	Details168	270-88-9421
12	100045	name816	Details556	905-95-3034
13	100044	name269	Details564	468-12-9442
14	100043	name996	Details46	881-55-2565
15	100042	name955	Details117	340-33-3612
16	100041	name965	Details452	667-79-2906
17	100040	name5	Details735	158-11-8311



In the DB:

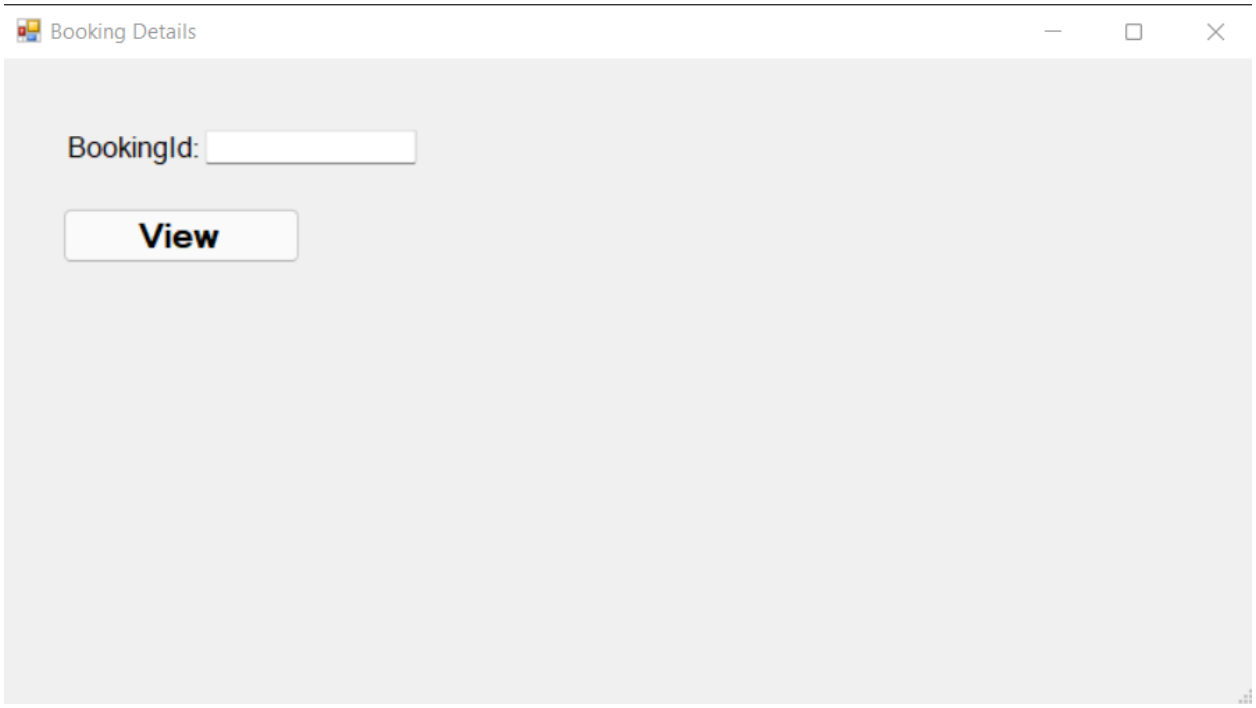
	ID	FLI_ID	PASS_ID	SOURCE_ID	DEST_ID
1	5	1	6	1	31
2	7	1	100053	1	31
3	3	1	6	1	31
4	10	1	100056	1	31
5	2	1	6	1	31
6	1	1	1	1	2
7	8	1	100054	1	31
8	9	4	100055	1	31
9	6	1	100052	1	31



In the DB:Availabilty is updated.

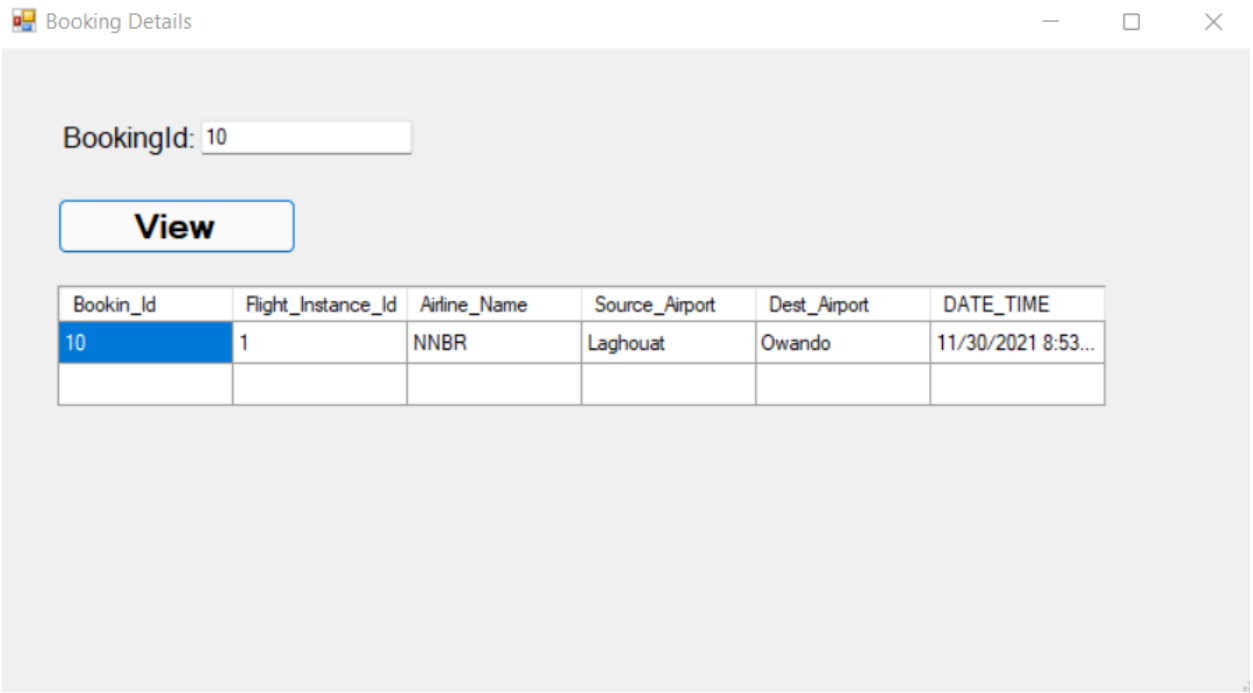
	Flight_Instance_Id	Flight_Id	Date_Time	From_Id	To_Id	Crew	AVAILABILITY
1	5	5	02-DEC-21 12.00.00.000000000 AM	1	31	SHRUTHI+PRIYA	1000
2	6	389	23-DEC-21 04.46.16.000000000 PM	341	344	first894 last82--first230 last455--first879 last600	700
3	2	5	30-NOV-21 10.00.00.174000000 PM	1	31	6	1
4	3	6	01-DEC-21 10.01.25.601000000 PM	1	31	7	1
5	4	7	01-DEC-21 10.02.29.756000000 PM	1	31	8	0
6	7	60	02-DEC-21 08.28.46.000000000 PM	341	344	first966 last12--first339 last210--first204 last169	800
7	8	267	04-DEC-21 04.01.33.000000000 PM	334	342	first529 last589--first281 last255--first505 last601	300
8	1	1	30-NOV-21 08.53.39.712000000 PM	1	31	5	99

Click on “View Booking details” from the passenger page and the “Booking Details” Page opens as shown below:



Enter the booking id and Click on View Button.

Booking_id,Flight_Instance_id,Airline_name,Source and destination airports and the date for the booked flight is displayed.



Query used to populate the table above:

SELECT a.id as Bookin_Id , a.fli_id as Flight_Instance_Id,f."NAME" as Airline_Name, b.name as Source_Airport, c.name as Dest_Airport , d."Date_Time" as DATE_TIME FROM booking a INNER JOIN airports b ON a.source_id = b.id INNER JOIN airports c ON a.dest_id = c.id INNER JOIN flight_instance d ON a.fli_id = d."Flight_Instance_Id" inner join flight e ON d."Flight_Id" = e."Flight_Id" INNER JOIN airlines f ON e."Airline_Id" = f."ID" WHERE a.id = @entered booking_id above;

SECTION 2 QUERY WRITING

2.1 INTERESTING QUERIES

Query #1

Count the number of employees by a specific airlines:

```
select count(AIRLINE)
from employee
LEFT JOIN airlines ON airlines.airline_id = employee.airline
WHERE AIRLINE = 21
```

Query #2

Select all the Pilots from a specific airline

```
select *
from employee
LEFT JOIN airlines ON airlines.airline_id = employee.airline
WHERE AIRLINE = 21 AND employee.details = 'pilot'
```

Query #3

Select all flight to a specific country

```
select *
from airports
LEFT JOIN flight_instance on flight_instance."To_Id" = airports.airport_id
Where "To_Id" IS NOT NULL AND airports.country = 'USA'
```

Query #4

```
select airport_id,airports.name,airports.city,
airports.state,airports.country,flight_instance."To_Id",
```

flight_instance."Flight_Id",flight_instance."Flight_Instance_Id",

flight_instance."Crew",flight_instance."Date_Time"

from airports

LEFT JOIN flight_instance on flight_instance."To_Id" = airports.airport_id

Where "To_Id" IS NOT NULL

ORDER BY airports.country

Query #5

Find a specific employee and the airlines he/she works for

select * from employee

LEFT JOIN airlines on airlines.airline_id = employee.airline

WHERE LNAME LIKE 'Flores%' AND FNAME LIKE 'Heather%'

Queries used in the application:

<u>Add flight instances operation by admin-</u>

```
List<ComboItem> comboItemLlist1 = GetComboData("SELECT a.\"Flight_Id\" FROM FLIGHT a INNER JOIN AIRLINES b ON a.\"Airline_Id\" = b.\"AIRLINE_ID\" WHERE b.\"NAME\" = \'" + airline_name + "\'");
//Console.WriteLine(comboItemLlist1[1].text);
comboBox4.DataSource = comboItemLlist1;
comboBox4.DisplayMember = "text";
comboBox4.ValueMember = "text";
List<ComboItem> comboItemLlist2 = GetComboData("SELECT CONCAT(CONCAT(FNAME,' '),LNAME) AS FULL_NAME FROM EMPLOYEE a INNER JOIN AIRLINES b ON a.AIRLINE = b.AIRLINE_ID WHERE a.DETAILS = 'pilot' AND b.NAME = \'" + airline_name + "\'");
//Console.WriteLine(comboItemLlist1[1].text);
comboBox5.DataSource = comboItemLlist2;
comboBox5.DisplayMember = "text";
comboBox5.ValueMember = "text";
List<ComboItem> comboItemLlist3 = GetComboData("SELECT CONCAT(CONCAT(FNAME,' '),LNAME) AS FULL_NAME FROM EMPLOYEE a INNER JOIN AIRLINES b ON a.AIRLINE = b.AIRLINE_ID WHERE a.DETAILS = 'co-pilot' AND b.NAME = \'" + airline_name + "\'");
//Console.WriteLine(comboItemLlist1[1].text);
comboBox6.DataSource = comboItemLlist3;
comboBox6.DisplayMember = "text";
comboBox6.ValueMember = "text";
List<ComboItem> comboItemLlist4 = GetComboData("SELECT CONCAT(CONCAT(FNAME,' '),LNAME) AS FULL_NAME FROM EMPLOYEE a INNER JOIN AIRLINES b ON a.AIRLINE = b.AIRLINE_ID WHERE a.DETAILS = 'cabin-crew' AND b.NAME = \'" + airline_name + "\'");
//Console.WriteLine(comboItemLlist1[1].text);
comboBox7.DataSource = comboItemLlist4;
comboBox7.DisplayMember = "text";
comboBox7.ValueMember = "text";
```

In the above screenshot,Query1 is used to populate Flight_Id based on the airline ID.Query2,3,4 are used to display the Cabin_crew,Pilot, and Co-pilot full names in the selected airline.

View Flights capacity and availability by the Admin-

Admin can view the capacity and availability by selecting the Flight instance from the DropDown.

```
OracleCommand command = new OracleCommand();
command.CommandText = "SELECT a.\"AVAILABILITY\", b.\"Capacity\" FROM FLIGHT_INSTANCE a INNER JOIN FLIGHT b ON a.\"Flight_Id\" = b.\"Flight_Id\" WHERE a.\"Flight_Instance_Id\" = " + fli_id_selected;
command.Connection = dbCon;
```

Search Flights By passenger-

```
command.CommandText = "select * from FLIGHT_INSTANCE where \"From_Id\" in (select AIRPORT_ID from airports where NAME like '" + fromCity+"') " +
    "and \"To_Id\" in (select AIRPORT_ID from airports where NAME like '" + toCity + "') "
    + "and to_char(\"Date_Time\",  'mm/dd/yyy') = to_char(TO_DATE('" + dateValue + "', 'YYYY-MM-DD'), 'mm/dd/yyy')";
command.Connection = dbCon;
dbCon.Open();
```

This query displays the Flights based on the selected Date and the user entered Source and Destination Airports.

2.2 STORED PROCEDURES

We used four stored procedures, all of which were called from the Windows Form.

They are as follows:

- BK_SP- When a booking is made by the passenger, this stored procedure is called. It inserts a booking record into the booking table.

```
create or replace PROCEDURE BK_SP
(
  BK_ID IN NUMBER
, FLI_ID IN NUMBER
, PASS_ID IN NUMBER
, SOURCE_AIRPORT IN NUMBER
, DEST_AIRPORT IN NUMBER
) AS
BEGIN
  INSERT INTO BOOKING VALUES(BK_ID,FLI_ID,PASS_ID,SOURCE_AIRPORT,DEST_AIRPORT);
  COMMIT;
END BK_SP;
```

- FLI_SP-It inserts a Flight instance record into the Flight_instance table. This stored procedure is invoked by the admin when he tries to add flight instances from the Windows Form.

```
  create or replace PROCEDURE FLI_SP
(
  PARAM_FLI_ID IN NUMBER
, PARAM_FL_ID IN NUMBER
, PARAM_DATE IN DATE
, PARAM_SOURCE_ID IN NUMBER
, PARAM_DEST_ID IN NUMBER
, PARAM_CREW IN VARCHAR2
, PARAM_AVAILABILITY IN NUMBER
) AS
BEGIN
  INSERT INTO flight_instance VALUES (PARAM_FLI_ID,PARAM_FL_ID,PARAM_DATE,PARAM_SOURCE_ID,PARAM_DEST_ID,PARAM_CREW,PARAM_AVAILABILITY);
  COMMIT;
END FLI_SP;
```

- PASS_SP-It inserts a passenger record into the passenger table. When a passenger confirms a booking, this procedure is invoked, and the information from the passenger details page is sent into it.

```
create or replace PROCEDURE PASS_SP
(
  PARAM_ID IN NUMBER
, PARAM_NAME IN VARCHAR2
, PARAM_DETAILS IN VARCHAR2
, PARAM_PASSPORT IN VARCHAR2
) AS
BEGIN
  INSERT INTO passenger VALUES(PARAM_ID,PARAM_NAME,PARAM_DETAILS,PARAM_PASSPORT);
  COMMIT;
END PASS_SP;
```

- UPDATE_FLI_SP- This stored procedure is used to update the availability of a flight that is selected for booking by the passenger. Once the passenger confirms booking this stored procedure is invoked.

```
create or replace PROCEDURE UPDATE_FLI_SP
(
  PARAM_FLI_ID IN NUMBER
, PARAM_AVAILABILITY IN NUMBER
) AS
BEGIN
  UPDATE flight_instance SET availability = PARAM_AVAILABILITY WHERE "Flight_Instance_Id" = PARAM_FLI_ID;
  COMMIT;
END UPDATE_FLI_SP;
```

2.3 TRIGGERS & SEQUENCES

Sequences: Sequence are database objects which generate a sequence of numbers and can be useful for creating unique Primary Key Ids for tables. In our database we have created sequences for the ID columns of Airports, Airlines and Employee tables. They all have the same sort of SQL commands and do the same function. The following image depicts the declarative command to create a sequence for the Airport table which has a minimum value of 1 and no specified maximum. It defaults to the number as shown below. The sequence generates numbers starting from 1 and increments each output by 1 and can cache up to 20 values.

```
CREATE SEQUENCE  "DB3PLUM"."AIRPORT_SEQ"
        MINVALUE 1
        MAXVALUE 9999999999999999999999999999
        INCREMENT BY 1
        START WITH 1
        CACHE 20;
```

Triggers: A trigger is an event-action combination that we use to tell the database engine to perform a particular task when a certain event occurs. The two type of triggers are DDL triggers, firing when any DDL query takes place and DML triggers, firing when certain DML events trasnpire such as Insert, Update and Delete. Here we have created three DML triggers in our database on the PK ID columns of Airports, Airlines and Employee table to mimic the

AUTO_INCREMENT functionality of SQL Server. As an indicative example, we can examine the “AI_BIR” trigger we have created. We see that trigger is created so that it fires before any Insert is executed on the Airlines table. In the BEGIN section it selects the next value of the sequence which we coded above and inserts it into the ID column of the current, or latest, row which is being inserted. These triggers can be enabled and disabled at our convenience.

```
create or replace TRIGGER ai_bir
BEFORE INSERT ON  Airlines
FOR EACH ROW

BEGIN
  SELECT al_seq.NEXTVAL
  INTO   :new.ID
  FROM   dual;
END;
```

SECTION 3 - PERFORMANCE TUNING

Indexing Strategies and Testing:

We used our passenger table and tried executing some queries and attached screenshots for times taken, before and after index creations.

select * from passenger where name = 'name265';

Script Output × Query Result ×

SQL | All Rows Fetched: 112 in 0.092 seconds

	PASSENGER_ID	NAME	DETAILS	PASSPORT_NUM
97	17991	name265	Details342	674-25-8287
98	18163	name265	Details290	888-12-2836
99	21422	name265	Details27	500-21-9106
100	27195	name265	Details959	808-83-5671
101	24646	name265	Details33	956-32-6015
102	24764	name265	Details525	366-59-8672

Query... x

SQL | All Rows Fetched: 112 in 0.086 seconds

	PASSENGER_ID	NAME	DETAILS	PASSPORT_NUM
95	49543	name265	Details875	472-72-5126
96	46781	name265	Details181	244-67-3114
97	49720	name265	Details937	943-14-2886
98	50189	name265	Details990	729-88-5442
99	44486	name265	Details738	110-59-7162
100	30177	name265	Details778	931-58-1067

select * from passenger where name like '%name26%';

Script Output x | Query Result x

SQL | All Rows Fetched: 1114 in 0.706 seconds

	PASSENGER_ID	NAME	DETAILS	PASSPORT_NUM
1099	96780	name263	Details70	247-49-3163
1100	88469	name261	Details693	139-59-9349
1101	88611	name261	Details596	360-15-4910
1102	94053	name269	Details680	751-58-7617
1103	94094	name268	Details776	975-49-9754

Query Result x

SQL | All Rows Fetched: 1114 in 0.633 seconds

	PASSENGER_ID	NAME	DETAILS	PASSPORT_NUM
1097	93888	name262	Details590	833-23-8120
1098	96679	name266	Details430	589-64-7293
1099	96780	name263	Details70	247-49-3163
1100	88469	name261	Details693	139-59-9349
1101	88611	name261	Details596	360-15-4910
1102	94053	name269	Details680	751-58-7617

select name, count(*) from passenger

where name like '%name36%'

group by name;

Script Output × Query Result ×

SQL | All Rows Fetched: 11 in 0.036 seconds

	NAME	COUNT(*)
1	name360	95
2	name368	106
3	name36	117
4	name361	115
5	name369	104
6	name363	112
7	name365	118
8	name366	75
9	name367	91
10	name362	96
11	name364	94

Query Result ×

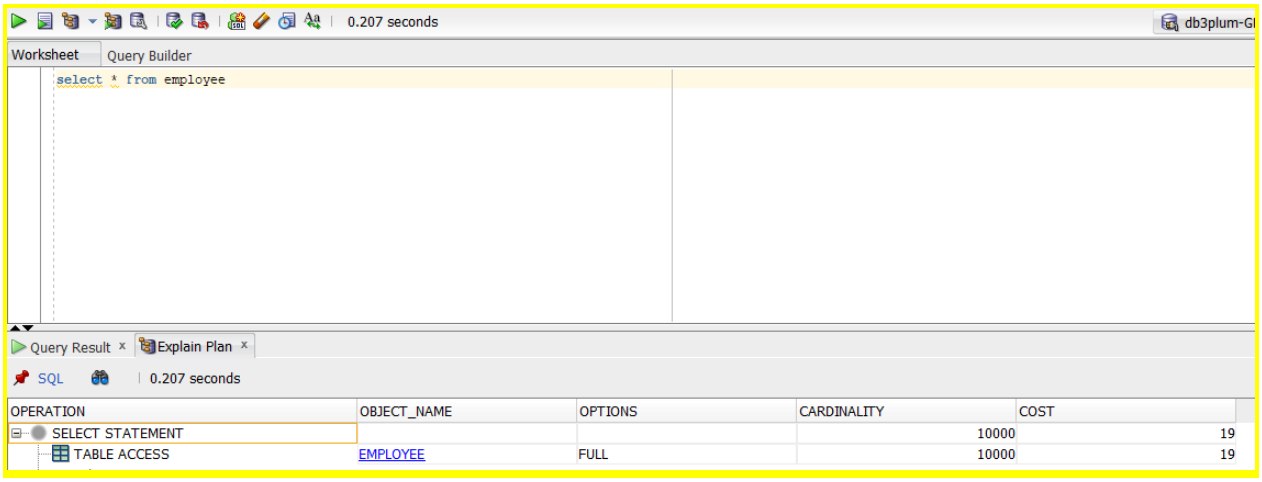
SQL | All Rows Fetched: 11 in 0.027 seconds

	NAME	COUNT(*)
1	name360	95
2	name368	106
3	name36	117
4	name361	115
5	name369	104
6	name363	112
7	name365	118
8	name366	75
9	name362	96
10	name367	91
11	name364	94

We can see slight difference after creating indexes, but may be the time in performance tuner would give more accurate information than this execution time.

Parallel Execution and Testing:

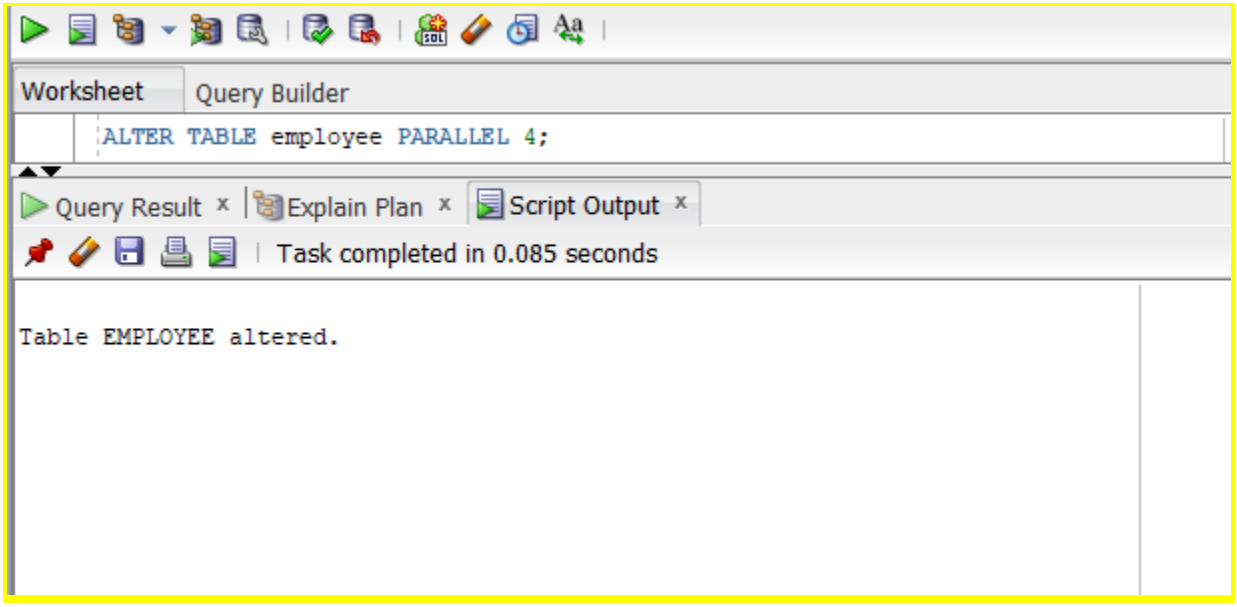
Unaltered parallel execution version had a cost of 19 with a cardinality of 10,000. After creating and testing different parallel execution, the cost of a full table search went from 19 down to 3. In addition to that, during our testing we noticed a time difference. Starting full table scan time was 0.207. The final test had the time at 0.104. Below are the commands tested.

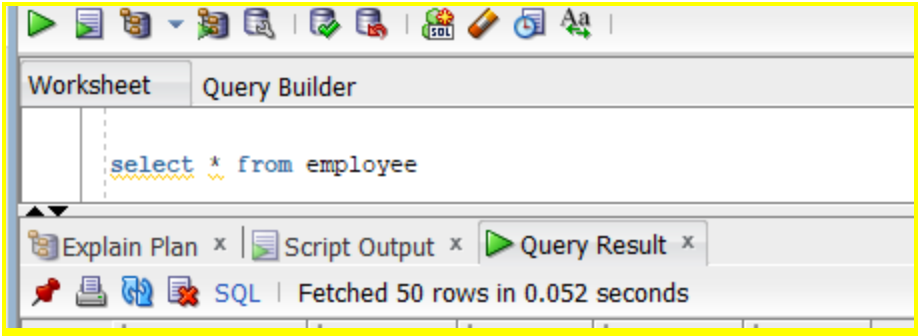


Test #1

ALTER TABLE employee PARALLEL 4;

select * from employee

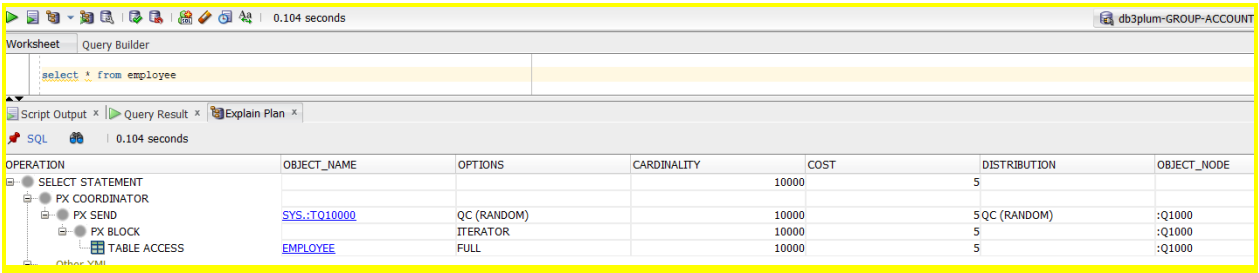




Test #2

ALTER TABLE employee PARALLEL 6;

select * from employee

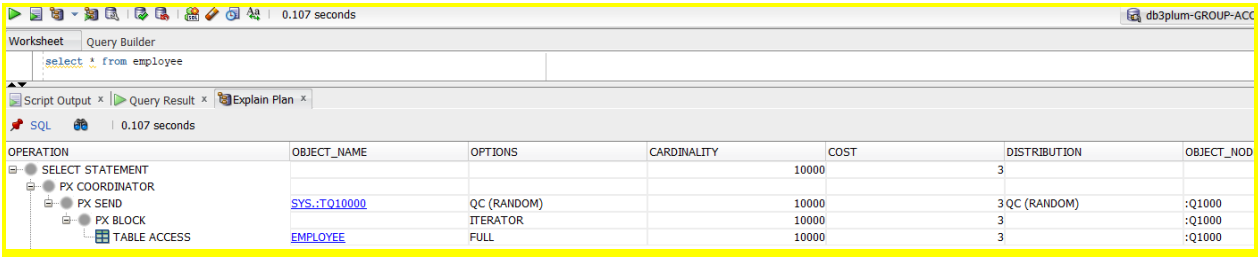


OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	DISTRIBUTION	OBJECT_NODE
SELECT STATEMENT			10000	5		
PX COORDINATOR						
PX SEND	SYS.:TQ10000	QC (RANDOM)	10000	5	QC (RANDOM)	:Q1000
PX BLOCK		ITERATOR	10000	5		:Q1000
TABLE ACCESS	EMPLOYEE	FULL	10000	5		:Q1000

Test #3

ALTER TABLE employee PARALLEL 8;

select * from employee



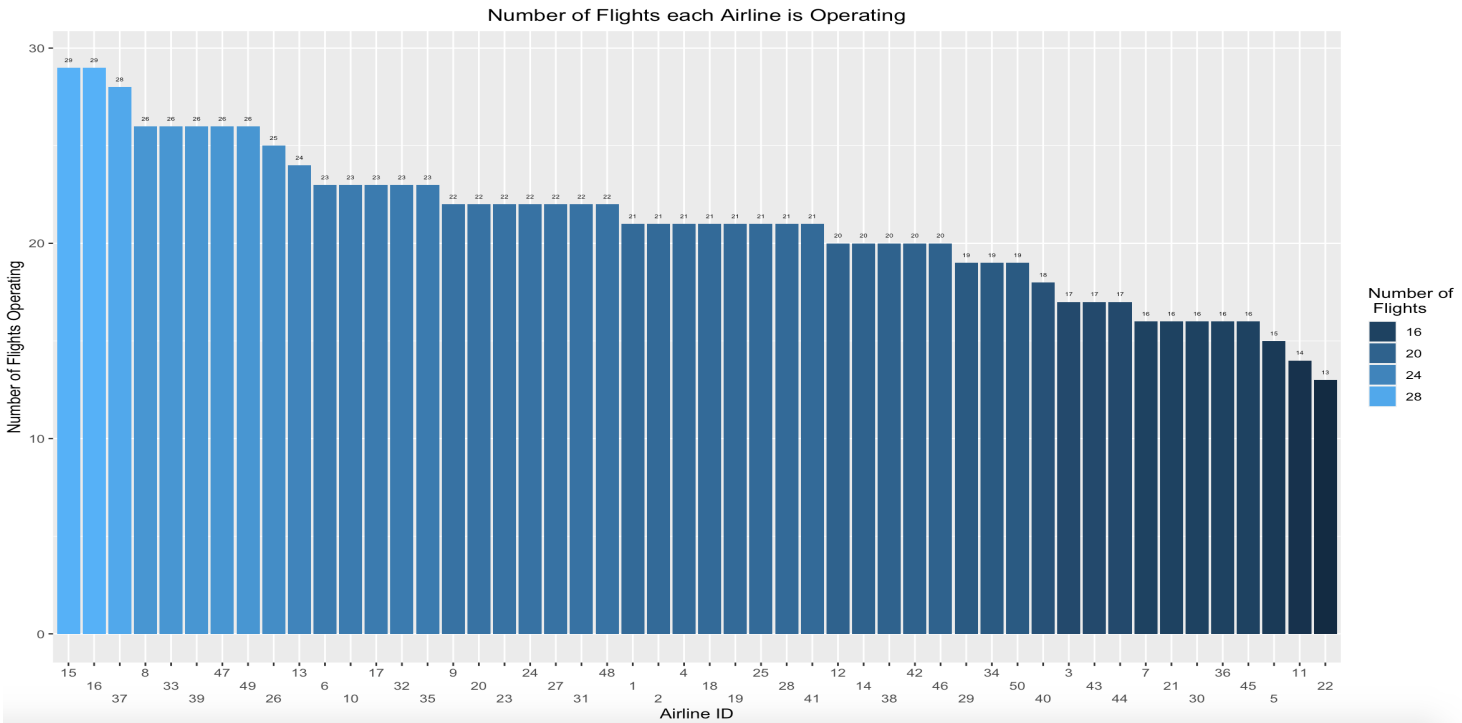
OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	DISTRIBUTION	OBJECT_NOD
SELECT STATEMENT			10000	3		
PX COORDINATOR						
PX SEND	SYS.:TQ10000	QC (RANDOM)	10000	3	QC (RANDOM)	:Q1000
PX BLOCK		ITERATOR	10000	3		:Q1000
TABLE ACCESS	EMPLOYEE	FULL	10000	3		:Q1000

SECTION 4

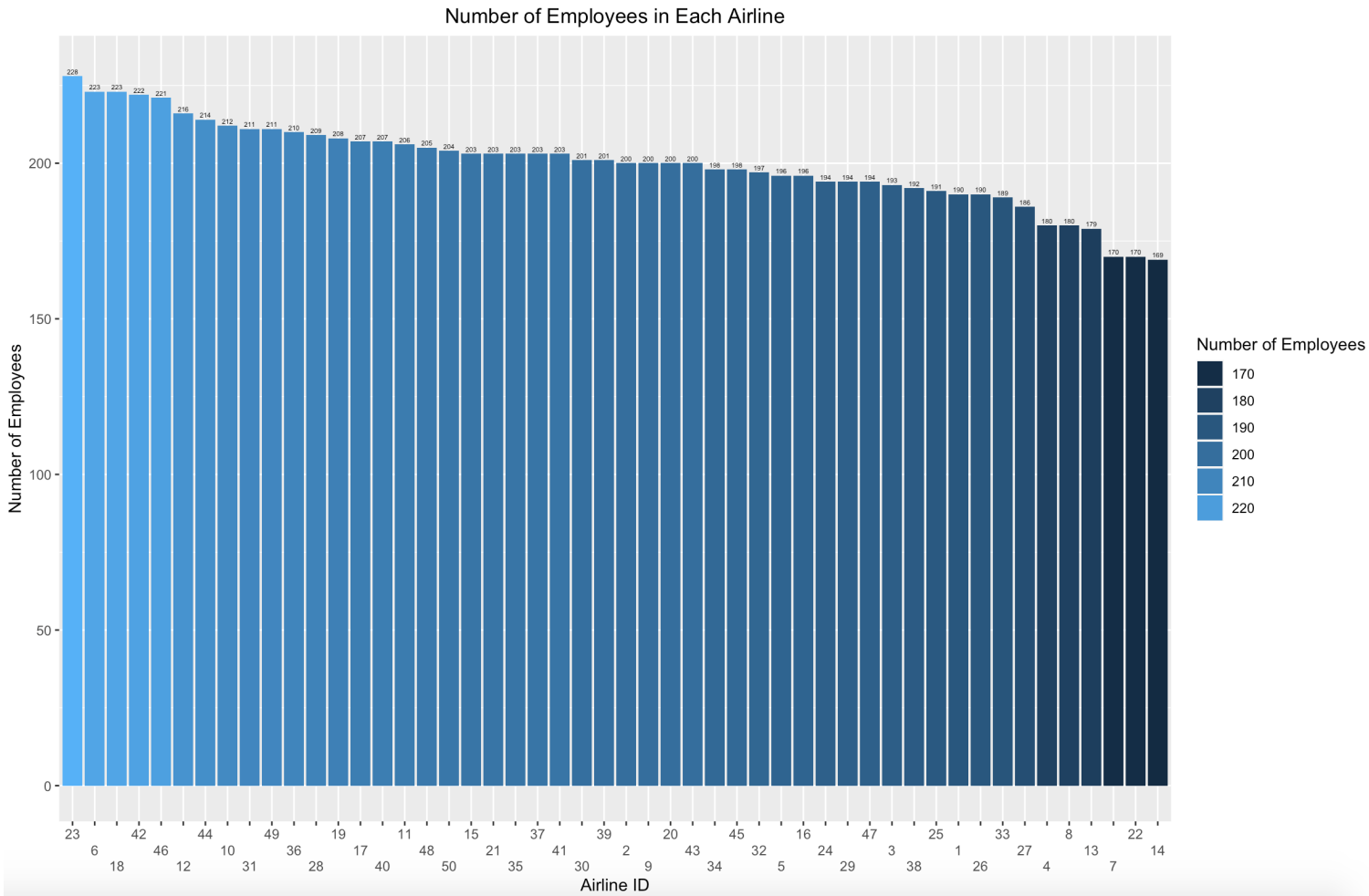
We already showed samples from our app in above sections, this section will cover some visualizations and DBA scripts that we explored. There is also an attached video which shows the usage of the app.

4.1 Data Visualization

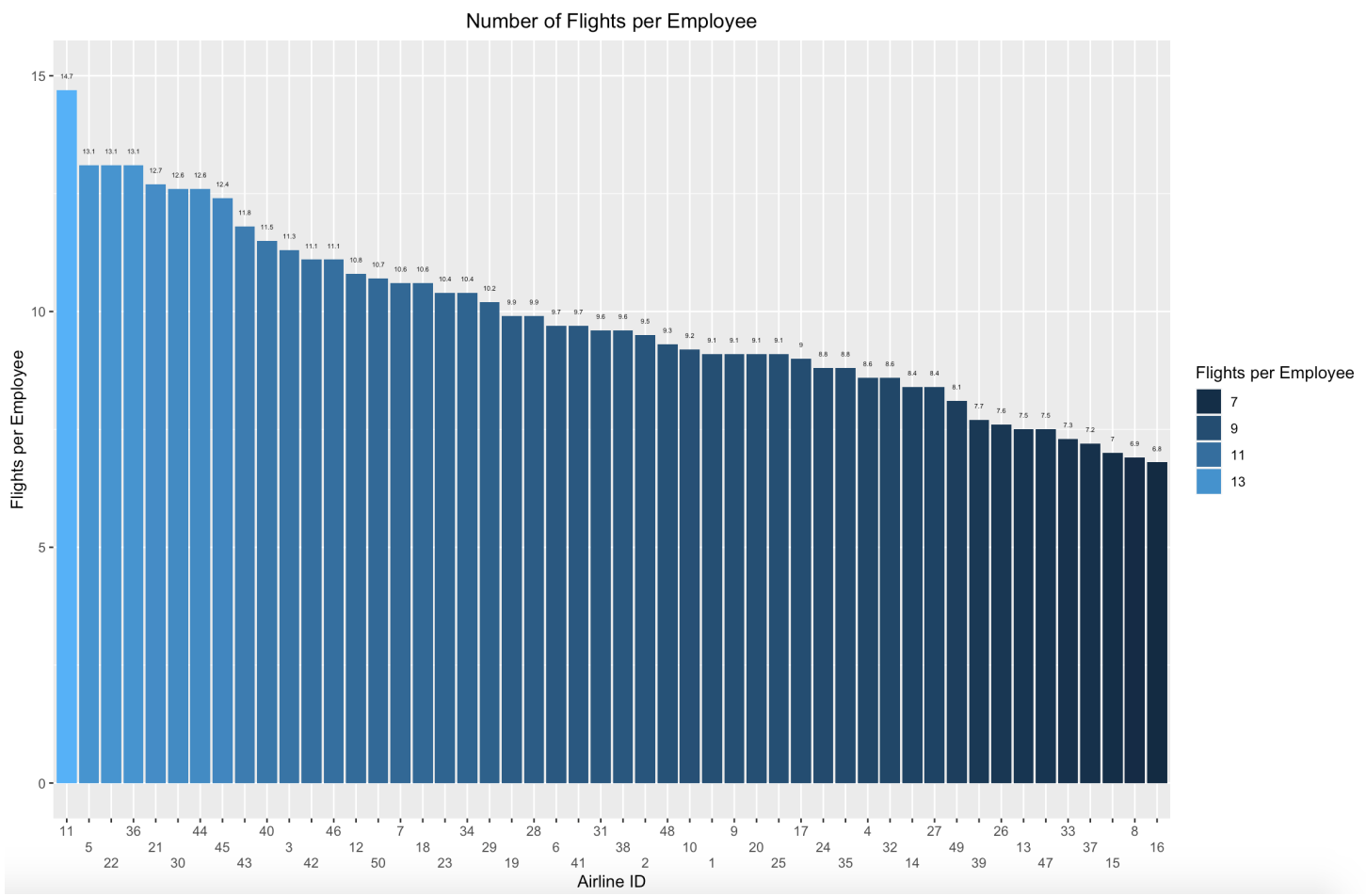
Data visualization is a critical component of analysis, giving the user a quick way to understand the data in the database. In the case of the Airline database it would be useful to get an overarching view of some important metrics the data is capturing.



The first plot shows the number of flights each airline is operating, in order of greatest to least. The most prolific airlines are operating 29 flights, while the least prolific are operating just 13.



This plot shows the total number of employees each Airline has on payroll. We would expect that all else equal, the number of flights each airline operates should correlate to the number of total employees. Interestingly here Airline 15 and 16 operate the most flights, but neither are in the top 10 in total employees.



Here we see a plot of the ratio of number of airline employees to number of flights. We can see that Airline 11 has 11.7 employees for each flight it operates, whereas Airline 16 is managing operations with just 6.5 employees per airline. This may indicate an over abundance of staff which is a major cost factor in airline operation. Visualizations like this can prompt further critical questioning to illuminate inefficiencies/obstacles in airline operation.

All of the plots were created by exporting relevant tables from the Airline DB and then using R Studio's GGPlot2 package to analyze and visualize the data.

DBA Scripts:

For any database management system, we need to monitor the system as a whole, look at different objects that are part of the database, look into user base, constraints etc. The following are some of the scripts we explored and corresponding screenshots are attached.

Get all the users in the system. The same script can be modified to check the validity and other fields.

select * from dba_users;

	USERNAME	USER_ID	PASSWORD	ACCOUNT_STATUS	LOCK_DATE	EXPIRY_DATE	DEFAULT_TABLESPACE	TEMPORARY_TABLESPACE	CREATED	PROFILE
1	SYS	0	(null)	OPEN	(null)	30-OCT-18	SYSTEM	TEMP	11-SEP-14	DEFAULT
2	AUDSYS	7	(null)	EXPIRED & LOCKED	11-SEP-14	11-SEP-14	USERS	TEMP	11-SEP-14	DEFAULT
3	SYSTEM	8	(null)	OPEN	(null)	04-MAY-21	SYSTEM	TEMP	11-SEP-14	DEFAULT
4	SYSBACKUP	2147483617	(null)	EXPIRED & LOCKED	11-SEP-14	11-SEP-14	USERS	TEMP	11-SEP-14	DEFAULT
5	SYSDG	2147483618	(null)	EXPIRED & LOCKED	11-SEP-14	11-SEP-14	USERS	TEMP	11-SEP-14	DEFAULT
6	SYSKM	2147483619	(null)	EXPIRED & LOCKED	11-SEP-14	11-SEP-14	USERS	TEMP	11-SEP-14	DEFAULT
7	OUTLN	13	(null)	EXPIRED & LOCKED	11-SEP-14	11-SEP-14	SYSTEM	TEMP	11-SEP-14	DEFAULT
8	XS$NULL	2147483638	(null)	EXPIRED & LOCKED	11-SEP-14	11-SEP-14	USERS	TEMP	11-SEP-14	DEFAULT
9	GSMADMIN_INTERNAL	21	(null)	EXPIRED & LOCKED	11-SEP-14	11-SEP-14	SYSAUX	TEMP	11-SEP-14	DEFAULT
10	GSMUSER	22	(null)	EXPIRED & LOCKED	11-SEP-14	11-SEP-14	USERS	TEMP	11-SEP-14	DEFAULT
11	DIP	23	(null)	EXPIRED & LOCKED	11-SEP-14	11-SEP-14	USERS	TEMP	11-SEP-14	DEFAULT
12	ORACLE_OCM	36	(null)	EXPIRED & LOCKED	11-SEP-14	11-SEP-14	USERS	TEMP	11-SEP-14	DEFAULT
13	DBSNMP	48	(null)	EXPIRED & LOCKED	11-SEP-14	11-SEP-14	SYSAUX	TEMP	11-SEP-14	DEFAULT
14	APPQOSSYS	49	(null)	EXPIRED & LOCKED	11-SEP-14	11-SEP-14	SYSAUX	TEMP	11-SEP-14	DEFAULT
15	XDB	50	(null)	EXPIRED & LOCKED	11-SEP-14	11-SEP-14	SYSAUX	TEMP	11-SEP-14	DEFAULT
16	ANONYMOUS	51	(null)	EXPIRED & LOCKED	11-SEP-14	11-SEP-14	SYSAUX	TEMP	11-SEP-14	DEFAULT
17	MDSYS	79	(null)	EXPIRED & LOCKED	11-SEP-14	11-SEP-14	SYSAUX	TEMP	11-SEP-14	DEFAULT
18	GSMCATUSER	61	(null)	EXPIRED & LOCKED	11-SEP-14	11-SEP-14	USERS	TEMP	11-SEP-14	DEFAULT
19	WMSYS	62	(null)	EXPIRED & LOCKED	11-SEP-14	11-SEP-14	SYSAUX	TEMP	11-SEP-14	DEFAULT
20	OJVMSYS	70	(null)	EXPIRED & LOCKED	11-SEP-14	11-SEP-14	USERS	TEMP	11-SEP-14	DEFAULT

Get all objects that our user has created.

```
select * from all_objects where Owner like 'DB3PLUM';
```

	OWNER	OBJECT_NAME	SUBOBJECT_NAME	OBJECT_ID	DATA_OBJECT_ID	OBJECT_TYPE	CREATED	LAST_DDL_TIME	TIMESTAMP	STATUS
1	DB3PLUM	UPPER_BEER_NAME	(null)	530491	530491	INDEX	01-DEC-21	01-DEC-21	2021-12-01:13:36:25	VALID
2	DB3PLUM	UPDATE_FLI_SP	(null)	530847	(null)	PROCEDURE	02-DEC-21	02-DEC-21	2021-12-02:20:29:21	VALID
3	DB3PLUM	UNQBBER	(null)	530488	530488	INDEX	01-DEC-21	01-DEC-21	2021-12-01:12:53:00	VALID
4	DB3PLUM	SYS_C00128978	(null)	530485	530485	INDEX	01-DEC-21	01-DEC-21	2021-12-01:12:35:01	VALID
5	DB3PLUM	SYSTPsvWe1lfnSQ64MZkbO2uwlA==	(null)	532448	(null)	TYPE	09-DEC-21	09-DEC-21	2021-12-09:13:55:49	VALID
6	DB3PLUM	STYLE_IDX	(null)	530483	530483	INDEX	01-DEC-21	01-DEC-21	2021-12-01:12:24:09	VALID
7	DB3PLUM	SP_BEERS2	(null)	530661	(null)	PROCEDURE	01-DEC-21	01-DEC-21	2021-12-01:23:04:47	VALID
8	DB3PLUM	SP_BEERS	(null)	530642	(null)	PROCEDURE	01-DEC-21	01-DEC-21	2021-12-01:21:46:15	VALID
9	DB3PLUM	PERSONS	(null)	530484	530484	TABLE	01-DEC-21	01-DEC-21	2021-12-01:12:35:01	VALID
10	DB3PLUM	PASS_SP	(null)	530846	(null)	PROCEDURE	02-DEC-21	09-DEC-21	2021-12-09:11:01:45	VALID
11	DB3PLUM	PASS_PK	(null)	528425	528425	INDEX	19-NOV-21	19-NOV-21	2021-11-19:16:00:09	VALID
12	DB3PLUM	PASSENGER_IDE	(null)	532449	532449	INDEX	09-DEC-21	09-DEC-21	2021-12-09:15:22:45	VALID
13	DB3PLUM	PASSENGER	(null)	528424	528424	TABLE	19-NOV-21	09-DEC-21	2021-12-07:10:38:08	VALID
14	DB3PLUM	FL_PK	(null)	528125	528125	INDEX	17-NOV-21	17-NOV-21	2021-11-17:12:25:12	VALID
15	DB3PLUM	FLI_SP	(null)	530716	(null)	PROCEDURE	02-DEC-21	02-DEC-21	2021-12-02:16:21:44	VALID
16	DB3PLUM	FLI_PK	(null)	528128	528128	INDEX	17-NOV-21	17-NOV-21	2021-11-17:13:03:35	VALID
17	DB3PLUM	FLIGHT_INSTANCE	(null)	528127	528127	TABLE	17-NOV-21	30-NOV-21	2021-11-30:16:23:17	VALID
18	DB3PLUM	FLIGHT	(null)	528124	528124	TABLE	17-NOV-21	17-NOV-21	2021-11-17:12:25:12	VALID
19	DB3PLUM	EMP_SEQ	(null)	528113	(null)	SEQUENCE	17-NOV-21	17-NOV-21	2021-11-17:12:13:03	VALID
20	DB3PLUM	EMP_PK	(null)	528117	528117	INDEX	17-NOV-21	17-NOV-21	2021-11-17:12:19:46	VALID
21	DB3PLUM	EMP_BIR	(null)	528115	(null)	TRIGGER	17-NOV-21	30-NOV-21	2021-11-17:12:20:53	INVALID
22	DB3PLUM	EMPLOYEE	(null)	528116	530330	TABLE	17-NOV-21	09-DEC-21	2021-12-07:10:19:57	VALID
23	DB3PLUM	BREWERY_IDX	(null)	530490	530490	INDEX	01-DEC-21	01-DEC-21	2021-12-01:13:11:25	VALID
24	DB3PLUM	BOOKING	(null)	528421	530329	TABLE	19-NOV-21	07-DEC-21	2021-12-07:10:43:52	VALID
25	DB3PLUM	BK_SP	(null)	528423	(null)	PROCEDURE	19-NOV-21	09-DEC-21	2021-12-09:11:01:47	VALID
26	DB3PLUM	BK_PK	(null)	528422	530328	INDEX	19-NOV-21	30-NOV-21	2021-11-19:13:21:06	VALID
27	DB3PLUM	BEER_NAMES2	(null)	530489	530489	TABLE	01-DEC-21	01-DEC-21	2021-12-01:12:58:54	VALID
28	DB3PLUM	BEER_NAMES1	(null)	530482	530482	TABLE	01-DEC-21	01-DEC-21	2021-12-01:12:22:51	VALID
29	DB3PLUM	AL_SEQ	(null)	527945	(null)	SEQUENCE	16-NOV-21	16-NOV-21	2021-11-16:12:24:32	VALID
30	DB3PLUM	AL_PK	(null)	527944	527944	INDEX	16-NOV-21	16-NOV-21	2021-11-16:12:24:32	VALID
31	DB3PLUM	AI_BIR	(null)	527946	(null)	TRIGGER	16-NOV-21	30-NOV-21	2021-11-16:12:24:32	INVALID
32	DB3PLUM	AIRPORT_SEQ	(null)	528109	(null)	SEQUENCE	17-NOV-21	17-NOV-21	2021-11-17:11:56:14	VALID
33	DB3PLUM	AIRPORT_PK	(null)	528108	528108	INDEX	17-NOV-21	17-NOV-21	2021-11-17:11:56:14	VALID
34	DB3PLUM	AIRPORT_BIR	(null)	528110	(null)	TRIGGER	17-NOV-21	30-NOV-21	2021-11-17:11:56:14	INVALID
35	DB3PLUM	AIRPORTS	(null)	528107	528107	TABLE	17-NOV-21	07-DEC-21	2021-12-07:10:54:14	VALID
36	DB3PLUM	AIRLINES	(null)	527943	527943	TABLE	16-NOV-21	07-DEC-21	2021-12-07:10:45:51	VALID

SELECT * FROM user_constraints where owner = 'DB3PLUM';

	OWNER	CONSTRAINT_NAME	CONSTRAINT_TYPE	TABLE_NAME	SEARCH_CONDITION	SEARCH_CONDITION_VC
1	DB3PLUM	EMP_FK	R	EMPLOYEE	(null)	(null)
2	DB3PLUM	FL_FK	R	FLIGHT	(null)	(null)
3	DB3PLUM	BK_FK3	R	BOOKING	(null)	(null)
4	DB3PLUM	BK_FK4	R	BOOKING	(null)	(null)
5	DB3PLUM	FLI_FK2	R	FLIGHT_INSTANCE	(null)	(null)
6	DB3PLUM	FLI_FK3	R	FLIGHT_INSTANCE	(null)	(null)
7	DB3PLUM	FLI_FK1	R	FLIGHT_INSTANCE	(null)	(null)
8	DB3PLUM	BK_FK1	R	BOOKING	(null)	(null)
9	DB3PLUM	BK_FK2	R	BOOKING	(null)	(null)
10	DB3PLUM	AIRPORT_PK	P	AIRPORTS	(null)	(null)
11	DB3PLUM	AL_PK	P	AIRLINES	(null)	(null)
12	DB3PLUM	BIN$/Pk50wKpT3OtqCt4R6X0hw==$0	P	BIN$3VVrI19TTiqVYSDrzswD1Q==$0	(null)	(null)
13	DB3PLUM	BIN$6Jhtj3iES/G5uQGXSBL3Bw==$0	C	BIN$SpZpGhdESf2f14CpHwMueQ==$0	"name" IS NOT NULL	"name" IS NOT NULL
14	DB3PLUM	BIN$AQTWEhU4RjC5DrYEZgRFfQ==$0	P	BIN$v3Kc2EwMQkKNoN/cwcgEvw==$0	(null)	(null)
15	DB3PLUM	BIN$McqCxvHsQImJbLhNKGB3YA==$0	P	BIN$SpZpGhdESf2f14CpHwMueQ==$0	(null)	(null)
16	DB3PLUM	BIN$OvDADDE0Tz+7OWFB2kceNw==$0	C	BIN$v3Kc2EwMQkKNoN/cwcgEvw==$0	"LNAME" IS NOT NULL	"LNAME" IS NOT NULL
17	DB3PLUM	BIN$Y3t53E7wQc+np7DDd+Vcmw==$0	C	BIN$3VVrI19TTiqVYSDrzswD1Q==$0	"Flight_Id" IS NOT NULL	"Flight_Id" IS NOT NULL
18	DB3PLUM	BIN$dxTBF3VXTpq88pVR1k5G9Q==$0	C	BIN$SpZpGhdESf2f14CpHwMueQ==$0	"passport_number" IS NOT NULL	"passport_number" IS NOT NULL
19	DB3PLUM	BIN$gpYD31b9SfKP7cnpdMNZFQ==$0	C	BIN$3VVrI19TTiqVYSDrzswD1Q==$0	"Airline_Id" IS NOT NULL	"Airline_Id" IS NOT NULL
20	DB3PLUM	BIN$hUSKO5jQSDGAyzbg7v6ULQ==$0	C	BIN$SpZpGhdESf2f14CpHwMueQ==$0	"id" IS NOT NULL	"id" IS NOT NULL
21	DB3PLUM	BIN$iVaevATPRX+ilVoPo0RDrw==$0	C	BIN$3VVrI19TTiqVYSDrzswD1Q==$0	"Capacity" IS NOT NULL	"Capacity" IS NOT NULL
22	DB3PLUM	BIN$mA0LbPcPT/+dyW3VZkI8zQ==$0	P	BIN$og23goVTRDGBIqtWgU0nHQ==$0	(null)	(null)
23	DB3PLUM	BIN$mgmSfJrCTnel4bwn7pi6WQ==$0	C	BIN$v3Kc2EwMQkKNoN/cwcgEvw==$0	"FNAME" IS NOT NULL	"FNAME" IS NOT NULL
24	DB3PLUM	BIN$uODruUQcQsii6zqA06zpLg==$0	C	BIN$v3Kc2EwMQkKNoN/cwcgEvw==$0	"ID" IS NOT NULL	"ID" IS NOT NULL
25	DB3PLUM	BIN$xS/nRH4nSWapoUTRLzT66A==$0	C	BIN$SpZpGhdESf2f14CpHwMueQ==$0	"details" IS NOT NULL	"details" IS NOT NULL

Similarly we can modify the all_obejcts table to look into triggers, sequences and indexes that are part of the database.

We also might have a case where we need to see active sessions or sessions which hold a loc on a particular object and in those scenarios the following query can be used.

select * from v$sessions and corresponding

```
select * from v$session;
```

Query Result

All Rows Fetched: 44 in 0.078 seconds

	SADDR	SID	SERIAL#	AUDSID	PADDR	USER#	USERNAME	COMMAND	OWNERID	TADDR	LOCKWAIT	STATUS
1	000007FF65F011C8	2	56384	0	000007FF65A44D28	0	(null)	0	2147483644	(null)	(null)	ACTIVE
2	000007FF65EFF0F8	3	47694	0	000007FF65A45878	0	(null)	0	2147483644	(null)	(null)	ACTIVE
3	000007FF65EFD028	4	44866	0	000007FF65A463C8	0	(null)	0	2147483644	(null)	(null)	ACTIVE
4	000007FF65EFAF58	5	51972	0	000007FF65A46F18	0	(null)	0	2147483644	(null)	(null)	ACTIVE
5	000007FF65EF8E88	6	56168	0	000007FF65A47A68	0	(null)	0	2147483644	(null)	(null)	ACTIVE
6	000007FF65EF6DB8	7	29021	0	000007FF65A485B8	0	(null)	0	2147483644	(null)	(null)	ACTIVE
7	000007FF65EF4CE8	8	48456	0	000007FF65A49108	0	(null)	0	2147483644	(null)	(null)	ACTIVE
8	000007FF65EF2C18	9	50285	0	000007FF65A49C58	0	(null)	0	2147483644	(null)	(null)	ACTIVE
9	000007FF65EF0B48	10	43741	0	000007FF65A4A7A8	0	(null)	0	2147483644	(null)	(null)	ACTIVE
10	000007FF65EEEA78	11	44119	0	000007FF65A4B2F8	0	(null)	0	2147483644	(null)	(null)	ACTIVE
11	000007FF65EEC9A8	12	58677	0	000007FF65A4BE48	0	(null)	0	2147483644	(null)	(null)	ACTIVE
12	000007FF65EEA8D8	13	18434	0	000007FF65A4C998	0	(null)	0	2147483644	(null)	(null)	ACTIVE
13	000007FF65EE8808	14	31221	0	000007FF65A4D4E8	0	(null)	0	2147483644	(null)	(null)	ACTIVE
14	000007FF65EE6738	15	47688	0	000007FF65A4E038	0	(null)	0	2147483644	(null)	(null)	ACTIVE

THANK YOU.