PES UNIVERSITY

100 feet Ring Road, BSK 3rd Stage, Bengaluru 560085

Department of Computer Science and Engineering
Jan – May 2020

UE18CS252
Database Management Systems

Project Report

Cruise Travel Management

PES1201801716-Jagadish Rathod
4th Sem Sec-I Roll No-44

PROJECT SUMMARY

Cruise Travel management system deals with management of Cruise port, Cruise lines and passengers. The system provides broad overview of underlying operational factors that influence the Cruise port management. I explained the Data model with the help of ER Diagram and Relational Schema.

Trigger is A special type of Stored Procedure here I used two triggers Those are 1. When ticket price is updated, old prices are stored in ticket price history table 2. When a Ship is delayed it is inserted in a separate table for easy analysis. And also used some simple and complex Queries to find specific data by filtering specific criteria. This project helped me in strengthening my concepts in Unit 3 and Unit 4 of DBMS course. Also, it gave me confidence in creating any mini world example as a database and relations.

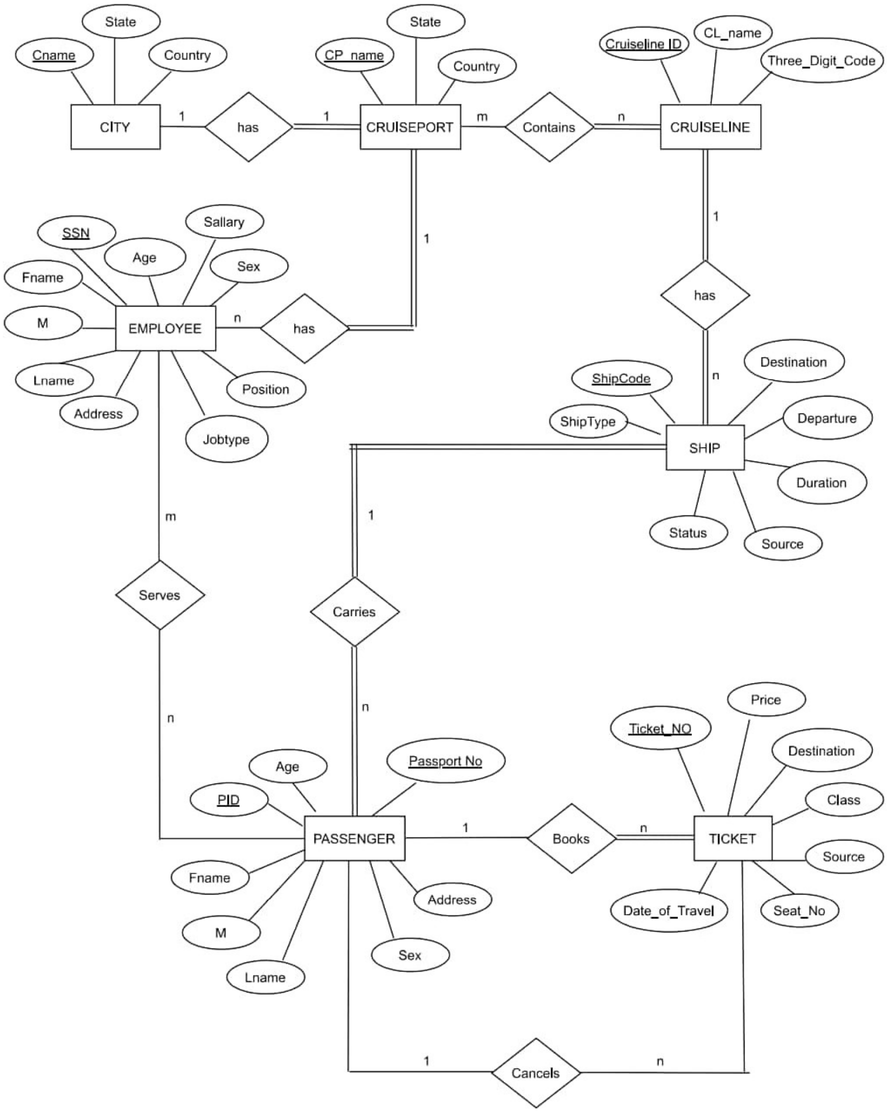
Introduction

The system is based on Cruise Travel management. Cruise Travel management system primarily deals with management of Cruise port, Cruise lines and passengers. The system provides broad overview of underlying operational factors that influence the Cruise port management. Every Cruise line is identified uniquely by Cruise line code Cruise line also has three-digit code which is printed on ticket. Cruise line companies serve Ships. Every Ship is uniquely identified by a Ship code

Ship serves passengers and carries passengers from source to destination. A passenger is uniquely identified by a passenger id and a passport number. For a passenger to travel by a Ship, he needs a ticket. A ticket is used to confirm that an individual has reserved a seat on a Ship. A passenger can book one or multiple tickets. The day on which he books ticket is a booking date Every Cruise port has employees working for it.

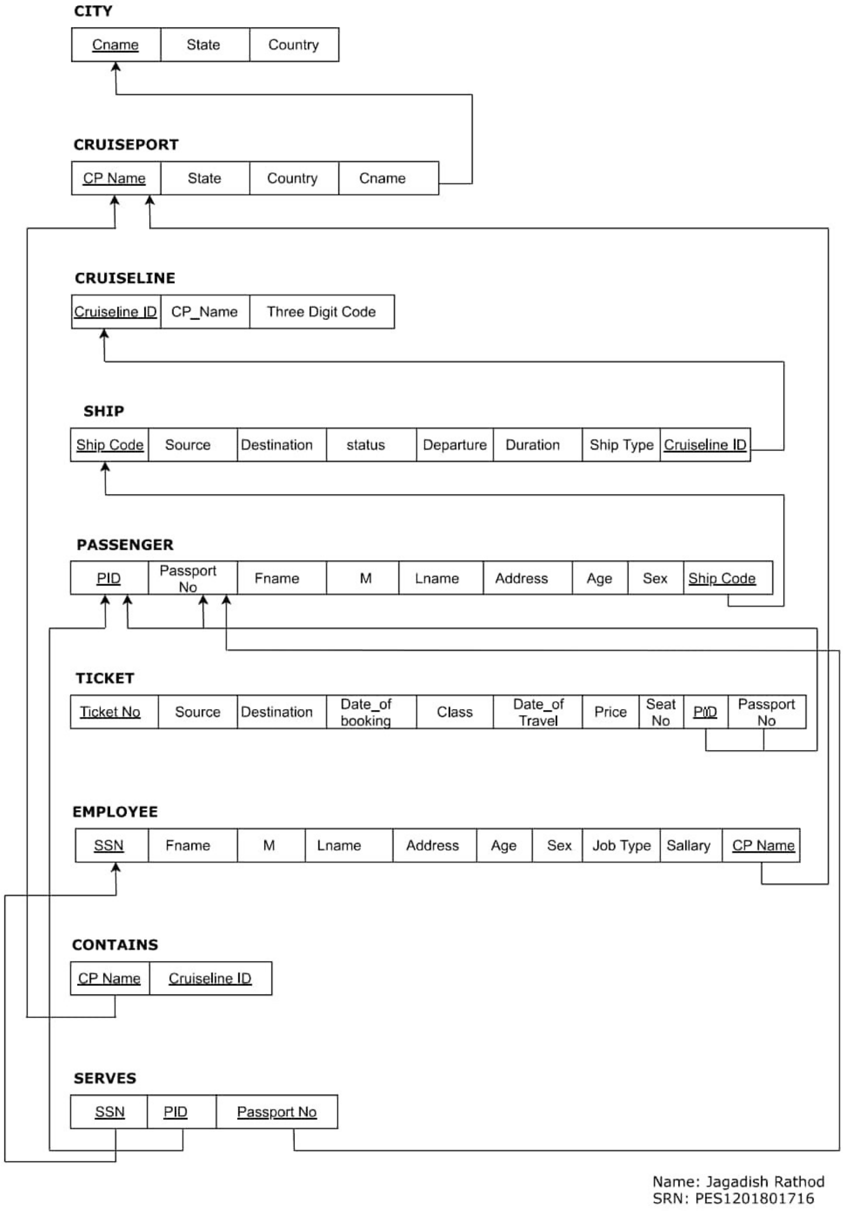
Data Model

ER Diagram For Cruise Travel Management



Name: Jagadish Rathod
SRN: PES1201801716

Relational Schema For Cruise Travel Management



CITY

Cname	State	Country

CRUISEPORT

CP Name	State	Country	Cname

CRUISELINE

Cruiseline ID	CP_Name	Three Digit Code

SHIP

Ship Code	Source	Destination	status	Departure	Duration	Ship Type	Cruiseline ID

PASSENGER

PID	Passport No	Fname	M	Lname	Address	Age	Sex	Ship Code

TICKET

Ticket No	Source	Destination	Date_of booking	Class	Date_of Travel	Price	Seat No	PID	Passport No

EMPLOYEE

SSN	Fname	M	Lname	Address	Age	Sex	Job Type	Sallary	CP Name

CONTAINS

CP Name	Cruiseline ID

SERVES

SSN	PID	Passport No

Name: Jagadish Rathod
SRN: PES1201801716

The binary Relationships in above system are:

1) **one-to-one**

(1) A city has only one Cruise port.

2) one-to-many

(1) Cruise line has multiple Ships, that is many Ships belong to the same Cruise line company.

(2) A Ship carries many passengers.

(3) A passenger can book one or more tickets.

(4) A passenger can cancel one or more tickets.

3) many-to-many

(1) A Cruise port may have many Cruise line offices.

FD and Normalization

FUNCTIONAL DEPENDENCIES:

IN RELATION CITY:

CNAME -> {STATE,COUNTRY}
PRIMARY KEY- CNAME

IN RELATION CRUISEPORT:

CP_NAME -> {STATE,COUNTRY,CNAME}
PRIMARY KEY- CP_NAME
FOREIGN KEY- CNAME

IN RELATION CRUISELINE:

CRUISELINEID -> {CL_NAME,THREE_DIGIT_CODE}
PRIMARY KEY- CRUISELINEID

IN RELATION SHIP:

SHIP_CODE ->
{SOURCE,DESTINATION ,DEPARTURE,STATUS,DURATION,SHIPTYPE,CRUISELINEID}
PRIMARY KEY – SHIP_CODE
FOREIGN KEY- CRUISELINEID

IN RELATION PASSENGER:

PASSPORTNO -> {PID, FNAME,M,LNAME,ADDRESS,AGE,SEX,SHIP_CODE}
PID -> {PASSPORTNO,FNAME,M,LNAME,ADDRESS, AGE,SEX,SHIP_CODE}
Here, both PID and PASSPORTNO uniquely identifies the tuple.
PRIMARY KEY- PASSPORTNO

SECONDARY KEY- PID
FOREIGN KEY- SHIP_CODE

IN RELATION TICKET:

TICKET_NUMBER -> {SOURCE,DESTINATION,DATE_OF_BOOKING,DATE_OF_TRAVEL,SEATNO,CLASS ,PID,PASSPORTNO}
PRIMARY KEY- TICKET_NUMBER
FOREIGN KEYS- PID, PASSPORTNO

IN RELATION EMPLOYEE:

SSN-> {FNAME,M,LNAME,ADDRESS ,AGE,SEX,JOBTYPE, SALARY,CP_NAME}
PRIMARY KEY – SSN
FOREIGN KEY – CP_NAME

IN RELATION CONTAINS:

all attributes are foreign keys which combine to form primary key.
PRIMARY KEY-{CP_NAME, CRUISELINEID}

IN RELATION SERVES:

all attributes are foreign keys which combine to form primary key.
PRIMARY KEY- {SSN, PID, PASSPORTNO}

here, keys are determined by the fact that all attributes of the relation can be determined from that key.

NORMALIZATION:

1) All relations are in 1st normal form as domain of each attributes is simple and indivisible.

2) Following functional dependencies violate 2nd normal form:

PASSPORTNO->FNAME,M,LNAME,ADDRESS, AGE,SEX
PID-> SHIP_CODE
Here, non- prime attributes are partially dependent on candidate keys and hence violate 2nd normal form.

3) Following functional dependencies violate 3rd normal form:

a) TICKET_NUMBER -> DATE_OF_BOOKING,SOURCE,DESTINATION,CLASS
DATE_OF_BOOKING,SOURCE,DESTINATION,CLASS -> PRICE

B) SSN -> JOBTYPE
JOBTYPE -> SALARY
Here, non prime attributes are transitively dependent on the primary key and hence violates 3rd normal form.

TABLES AFTER NORMALIZATION:

CITY (CNAME, STATE, COUNTRY)

CRUISE PORT (CP_NAME, STATE, COUNTRY, CNAME)

CRUISE LINE (CRUISELINEID, CL_NAME, THREE_DIGIT_CODE)

CONTAINS (CRUISELINEID, CP_NAME)

SHIP (SHIP_CODE, SOURCE, DESTINATION, DEPARTURE, STATUS, DURATION, SHIPTYPE, CRUISELINEID)

PASSENGER1 (PID, PASSPORTNO)

PASSENGER2 (PASSPORTNO, FNAME, M, LNAME, ADDRESS, PHONE, AGE, SEX)

PASSENGER3 (PID, SHIP_CODE)

TICKET1 (TICKET_NUMBER, SOURCE, DESTINATION, DATE_OF_BOOKING, DATE_OF_TRAVEL, SEATNO, CLASS, PID, PASSPORTNO)

TICKET2 (DATE_OF_BOOKING, SOURCE, DESTINATION, CLASS, PRICE)

EMPLOYEE1 (SSN, FNAME, M, LNAME, ADDRESS, AGE, SEX, JOBTYPE, POSITION, CP_NAME)

EMPLOYEE2(JOBTYPE, SALARY)

SERVES (SSN, PID, PASSPORTNO)

LOSSLESS JOIN PROPERTY:

a) FOR RELATION
PASSENGER(PID,PASSPORTNO,FNAME,M,LNAME,ADDRESS ,AGE,SEX, SHIP_CODE)

PASSENGER1(PID, PASSPORTNO)
PASSENGER2(PASSPORTNO,FNAME,M,LNAME,ADDRESS ,AGE,SEX)
PASSENGER3(PID,SHIP_CODE)

1) Union of attributes of all sub relations equals the attributes of main relation.
2) PASSENGER1 Ω PASSENGER2 = PASSPORTNO
PASSENGER1 Ω PASSENGER3 = PID WHICH is not equal to null
3) PASSPORTNO is a candidate key of PASSENGER2 as PASSPORTNO-> FNAME,M,LNAME,ADDRESS, AGE,SEX.
PID is a candidate key of PASSENGER3 as PID->SHIP_CODE
hence, the following decomposition satisfies lossless join property.

b) FOR RELATION
EMPLOYEE (SSN,FNAME,M,LNAME,ADDRESS, AGE,SEX,JOBTYPE,SALARY,C P_NAME)
EMPLOYEE1 (SSN,FNAME,M,LNAME,ADDRESS ,AGE,SEX,JOBTYPE,POSITION,CP_NAME)
EMPLOYEE2 (JOBTYPE,SALARY)
1) Union of attributes of all sub relations equals the attributes of main relation.
2)EMPLOYEE1 Ω EMPLOYEE2 = JOBTYPE which is not null
3)JOBTYPE is a candidate key of sub relation EMPLOYEE2 as JOBTYPE-> SALARY
hence, this decomposition of relation is also lossless.

DDL

TABLE SCRIPTS WITH THEIR INTEGRITY CONSTRAINTS:

```
CREATE TABLE CITY
(CNAME VARCHAR(15) NOT NULL,
STATE VARCHAR(15),
COUNTRY VARCHAR(30),
PRIMARY KEY(CNAME));


CREATE TABLE CRUISEPORT
(CP_NAME VARCHAR(100) NOT NULL,
STATE VARCHAR(15),
COUNTRY VARCHAR(30),
CNAME VARCHAR(15),
PRIMARY KEY(CP_NAME),
FOREIGN KEY(CNAME) REFERENCES CITY(CNAME) ON DELETE CASCADE);
```

```
CREATE TABLE CRUISELINE
(CRUISELINEID VARCHAR(3) NOT NULL,
CL_NAME VARCHAR(50),
THREE_DIGIT_CODE VARCHAR(3),
PRIMARY KEY(CRUISELINEID));


CREATE TABLE CONTAINS
(CRUISELINEID VARCHAR(3) NOT NULL,
CP_NAME VARCHAR(100) NOT NULL,
PRIMARY KEY(CRUISELINEID, CP_NAME),
FOREIGN KEY(CRUISELINEID) REFERENCES CRUISELINE(CRUISELINEID) ON DELETE CASCADE,
FOREIGN KEY(CP_NAME) REFERENCES cruiseport(CP_NAME) ON DELETE CASCADE);


CREATE TABLE SHIP
(SHIP_CODE VARCHAR(10) NOT NULL,
SOURCE VARCHAR(3),
DESTINATION VARCHAR(3),
DEPARTURE VARCHAR(10),
STATUS VARCHAR(10),
DURATION VARCHAR(30),
SHIPTYPE VARCHAR(10),
CRUISELINEID VARCHAR(3),
PRIMARY KEY(SHIP_CODE),
FOREIGN KEY(CRUISELINEID) REFERENCES CRUISELINE(CRUISELINEID) ON DELETE CASCADE);


CREATE TABLE PASSENGER1
(PID INT NOT NULL,
PASSPORTNO VARCHAR(10) NOT NULL,
PRIMARY KEY(PID, PASSPORTNO));


CREATE TABLE PASSENGER2
(PASSPORTNO VARCHAR(10) NOT NULL,
FNAME VARCHAR(20),
M VARCHAR(1),
LNAME VARCHAR(20),
ADDRESS VARCHAR(100),
AGE INT,
SEX VARCHAR(1),
PRIMARY KEY(PASSPORTNO));


CREATE TABLE PASSENGER3
(PID INT NOT NULL,
SHIP_CODE VARCHAR(10),
PRIMARY KEY(PID),
FOREIGN KEY(SHIP_CODE) REFERENCES SHIP(SHIP_CODE) ON DELETE CASCADE);



CREATE TABLE EMPLOYEE1
(SSN INT NOT NULL,
FNAME VARCHAR(20),
M VARCHAR(1),
LNAME VARCHAR(20),
ADDRESS VARCHAR(100),
AGE INT,
SEX VARCHAR(1),
JOBTYPE VARCHAR(30),
POSITION VARCHAR(30),
CP_NAME VARCHAR(100),
PRIMARY KEY(SSN),
FOREIGN KEY(CP_NAME) REFERENCES cruiseport(CP_NAME) ON DELETE CASCADE);

CREATE TABLE EMPLOYEE2
(JOBTYPE VARCHAR(30) NOT NULL,
POSITION VARCHAR(30),
SALARY INT,
PRIMARY KEY(JOBTYPE));


CREATE TABLE SERVES
(SSN INT NOT NULL,
PID INT NOT NULL,
PASSPORTNO VARCHAR(10) NOT NULL,
PRIMARY KEY(SSN, PID, PASSPORTNO),
FOREIGN KEY(SSN) REFERENCES EMPLOYEE1(SSN) ON DELETE CASCADE,
FOREIGN KEY(PID, PASSPORTNO) REFERENCES PASSENGER1(PID, PASSPORTNO) ON DE
LETE CASCADE);


CREATE TABLE TICKET1
(TICKET_NUMBER INT NOT NULL,
SOURCE VARCHAR(3),
DESTINATION VARCHAR(3),
DATE_OF_BOOKING DATE,
DATE_OF_TRAVEL DATE,
SEATNO VARCHAR(5),
CLASS VARCHAR(15),
PID INT,
PASSPORTNO VARCHAR(10),
FOREIGN KEY(PID, PASSPORTNO) REFERENCES PASSENGER1(PID, PASSPORTNO) ON DE
LETE CASCADE);
```

```
CREATE TABLE TICKET2
(DATE_OF_BOOKING DATE NOT NULL,
SOURCE VARCHAR(3) NOT NULL,
DESTINATION VARCHAR(3) NOT NULL,
CLASS VARCHAR(15) NOT NULL,
PRICE INT,
PRIMARY KEY(DATE_OF_BOOKING, SOURCE, DESTINATION, CLASS));
```

CHECK CONSTRAINTS:

a) AGE of each employee should be less than 67 years

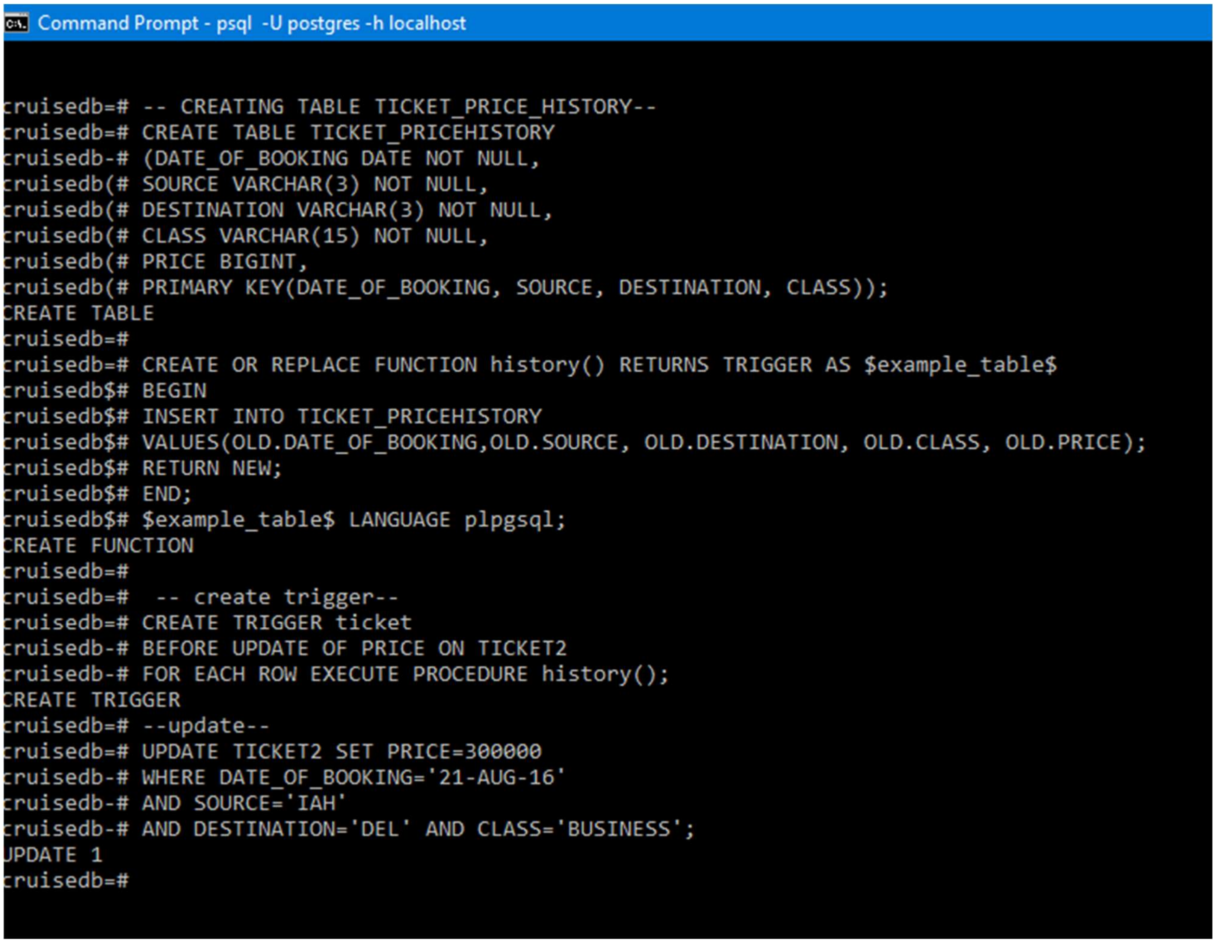
   ```
   ALTER TABLE EMPLOYEE1
   ADD CONSTRAINT AGE_LIMIT CHECK(AGE < 67);
   ```

b) length of the ticket number should be less than or equal to 13 digits.

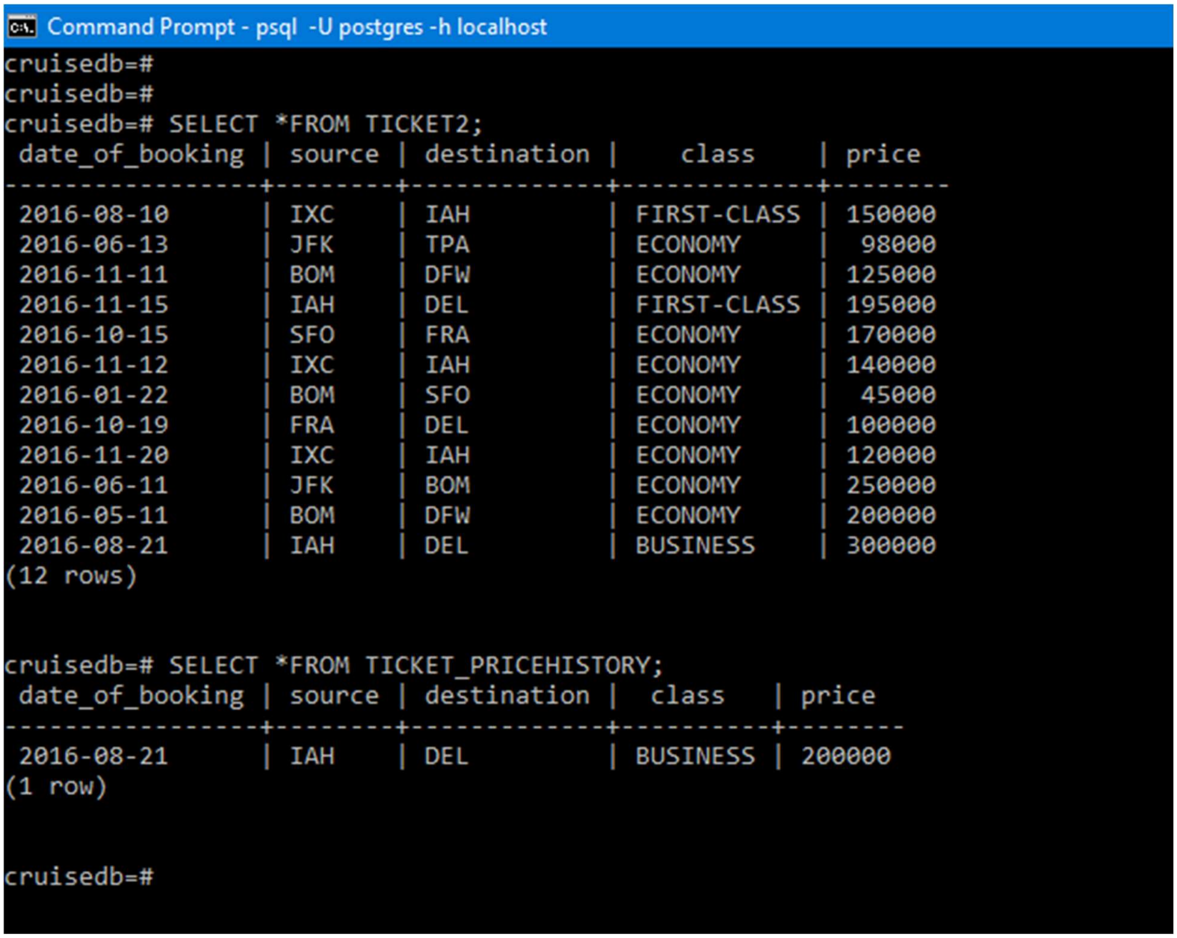
   ```
   ALTER TABLE TICKET1
   ADD CONSTRAINT TICKET_NO_LENGTH
   CHECK(LENGTH(CAST(TICKET_NUMBER AS TEXT))<=13);
   ```

Triggers

a) Whenever ticket price is updated, old prices are stored in ticket price history table



```
cruisedb=# -- CREATING TABLE TICKET_PRICE_HISTORY--
cruisedb=# CREATE TABLE TICKET_PRICEHISTORY
cruisedb-# (DATE_OF_BOOKING DATE NOT NULL,
cruisedb(# SOURCE VARCHAR(3) NOT NULL,
cruisedb(# DESTINATION VARCHAR(3) NOT NULL,
cruisedb(# CLASS VARCHAR(15) NOT NULL,
cruisedb(# PRICE BIGINT,
cruisedb(# PRIMARY KEY(DATE_OF_BOOKING, SOURCE, DESTINATION, CLASS));
CREATE TABLE
cruisedb=#
cruisedb=# CREATE OR REPLACE FUNCTION history() RETURNS TRIGGER AS $example_table$
cruisedb$# BEGIN
cruisedb$# INSERT INTO TICKET_PRICEHISTORY
cruisedb$# VALUES(OLD.DATE_OF_BOOKING,OLD.SOURCE, OLD.DESTINATION, OLD.CLASS, OLD.PRICE);
cruisedb$# RETURN NEW;
cruisedb$# END;
cruisedb$# $example_table$ LANGUAGE plpgsql;
CREATE FUNCTION
cruisedb=#
cruisedb=#  -- create trigger--
cruisedb=# CREATE TRIGGER ticket
cruisedb-# BEFORE UPDATE OF PRICE ON TICKET2
cruisedb-# FOR EACH ROW EXECUTE PROCEDURE history();
CREATE TRIGGER
cruisedb=# --update--
cruisedb=# UPDATE TICKET2 SET PRICE=300000
cruisedb-# WHERE DATE_OF_BOOKING='21-AUG-16'
cruisedb-# AND SOURCE='IAH'
cruisedb-# AND DESTINATION='DEL' AND CLASS='BUSINESS';
UPDATE 1
cruisedb=#
```

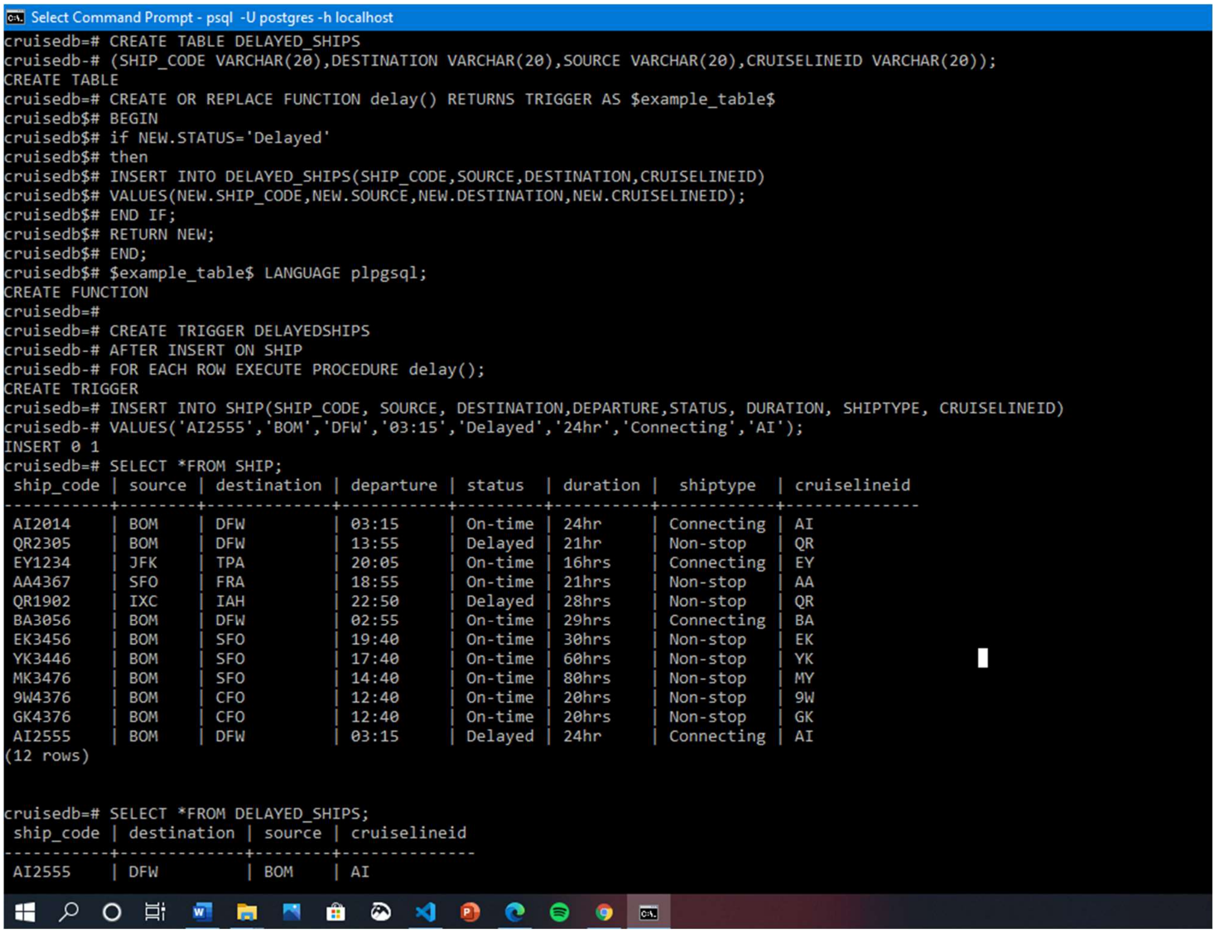


```
cruisedb=#
cruisedb=#
cruisedb=# SELECT *FROM TICKET2;
 date_of_booking | source | destination |    class     | price
-----------------+--------+-------------+--------------+--------
 2016-08-10      | IXC    | IAH         | FIRST-CLASS  | 150000
 2016-06-13      | JFK    | TPA         | ECONOMY      |  98000
 2016-11-11      | BOM    | DFW         | ECONOMY      | 125000
 2016-11-15      | IAH    | DEL         | FIRST-CLASS  | 195000
 2016-10-15      | SFO    | FRA         | ECONOMY      | 170000
 2016-11-12      | IXC    | IAH         | ECONOMY      | 140000
 2016-01-22      | BOM    | SFO         | ECONOMY      |  45000
 2016-10-19      | FRA    | DEL         | ECONOMY      | 100000
 2016-11-20      | IXC    | IAH         | ECONOMY      | 120000
 2016-06-11      | JFK    | BOM         | ECONOMY      | 250000
 2016-05-11      | BOM    | DFW         | ECONOMY      | 200000
 2016-08-21      | IAH    | DEL         | BUSINESS     | 300000
(12 rows)


cruisedb=# SELECT *FROM TICKET_PRICEHISTORY;
 date_of_booking | source | destination |  class    | price
-----------------+--------+-------------+----------+--------
 2016-08-21      | IAH    | DEL         | BUSINESS | 200000
(1 row)


cruisedb=#
```

b) When a Ship is delayed it is inserted in a separate table for easy analysis



```
cruisedb=# CREATE TABLE DELAYED_SHIPS
cruisedb-# (SHIP_CODE VARCHAR(20),DESTINATION VARCHAR(20),SOURCE VARCHAR(20),CRUISELINEID VARCHAR(20));
CREATE TABLE
cruisedb=# CREATE OR REPLACE FUNCTION delay() RETURNS TRIGGER AS $example_table$
cruisedb$# BEGIN
cruisedb$# if NEW.STATUS='Delayed'
cruisedb$# then
cruisedb$# INSERT INTO DELAYED_SHIPS(SHIP_CODE,SOURCE,DESTINATION,CRUISELINEID)
cruisedb$# VALUES(NEW.SHIP_CODE,NEW.SOURCE,NEW.DESTINATION,NEW.CRUISELINEID);
cruisedb$# END IF;
cruisedb$# RETURN NEW;
cruisedb$# END;
cruisedb$# $example_table$ LANGUAGE plpgsql;
CREATE FUNCTION
cruisedb=#
cruisedb=# CREATE TRIGGER DELAYEDSHIPS
cruisedb-# AFTER INSERT ON SHIP
cruisedb-# FOR EACH ROW EXECUTE PROCEDURE delay();
CREATE TRIGGER
cruisedb=# INSERT INTO SHIP(SHIP_CODE, SOURCE, DESTINATION,DEPARTURE,STATUS, DURATION, SHIPTYPE, CRUISELINEID)
cruisedb-# VALUES('AI2555','BOM','DFW','03:15','Delayed','24hr','Connecting','AI');
INSERT 0 1
cruisedb=# SELECT *FROM SHIP;
 ship_code | source | destination | departure | status  | duration |  shiptype  | cruiselineid
-----------+--------+-------------+-----------+---------+----------+------------+--------------
 AI2014    | BOM    | DFW         | 03:15     | On-time | 24hr     | Connecting | AI
 QR2305    | BOM    | DFW         | 13:55     | Delayed | 21hr     | Non-stop   | QR
 EY1234    | JFK    | TPA         | 20:05     | On-time | 16hrs    | Connecting | EY
 AA4367    | SFO    | FRA         | 18:55     | On-time | 21hrs    | Non-stop   | AA
 QR1902    | IXC    | IAH         | 22:50     | Delayed | 28hrs    | Non-stop   | QR
 BA3056    | BOM    | DFW         | 02:55     | On-time | 29hrs    | Connecting | BA
 EK3456    | BOM    | SFO         | 19:40     | On-time | 30hrs    | Non-stop   | EK
 YK3446    | BOM    | SFO         | 17:40     | On-time | 60hrs    | Non-stop   | YK
 MK3476    | BOM    | SFO         | 14:40     | On-time | 80hrs    | Non-stop   | MY
 9W4376    | BOM    | CFO         | 12:40     | On-time | 20hrs    | Non-stop   | 9W
 GK4376    | BOM    | CFO         | 12:40     | On-time | 20hrs    | Non-stop   | GK
 AI2555    | BOM    | DFW         | 03:15     | Delayed | 24hr     | Connecting | AI
(12 rows)


cruisedb=# SELECT *FROM DELAYED_SHIPS;
 ship_code | destination | source | cruiselineid
-----------+-------------+--------+--------------
 AI2555    | DFW         | BOM    | AI
```

SQL Queries

a) SIMPLE QUERIES:

1) Selecting all Cruise ports of India

```
SELECT CP_NAME FROM CRUISEPORT WHERE COUNTRY='India'
```

2) Selecting SHIPCODE from table SHIP where source is ´BOM´

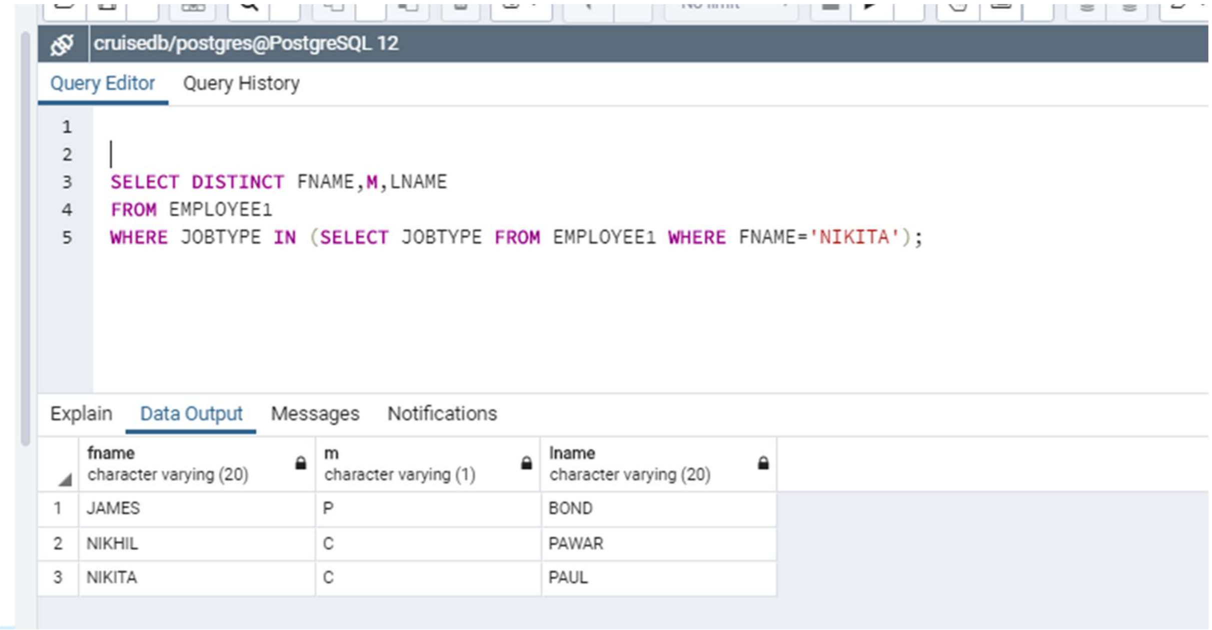
```
SELECT SHIP_CODE FROM SHIP WHERE SOURCE='BOM';
```

3) List of the Passengers their AGE is >=30

```
SELECT FNAME FROM PASSENGER2 WHERE AGE>=30;
```

b) COMPLEX QUERIES:

1) Retrieve the names of all employees who have the same jobtype as Employee ´NIKITA´.

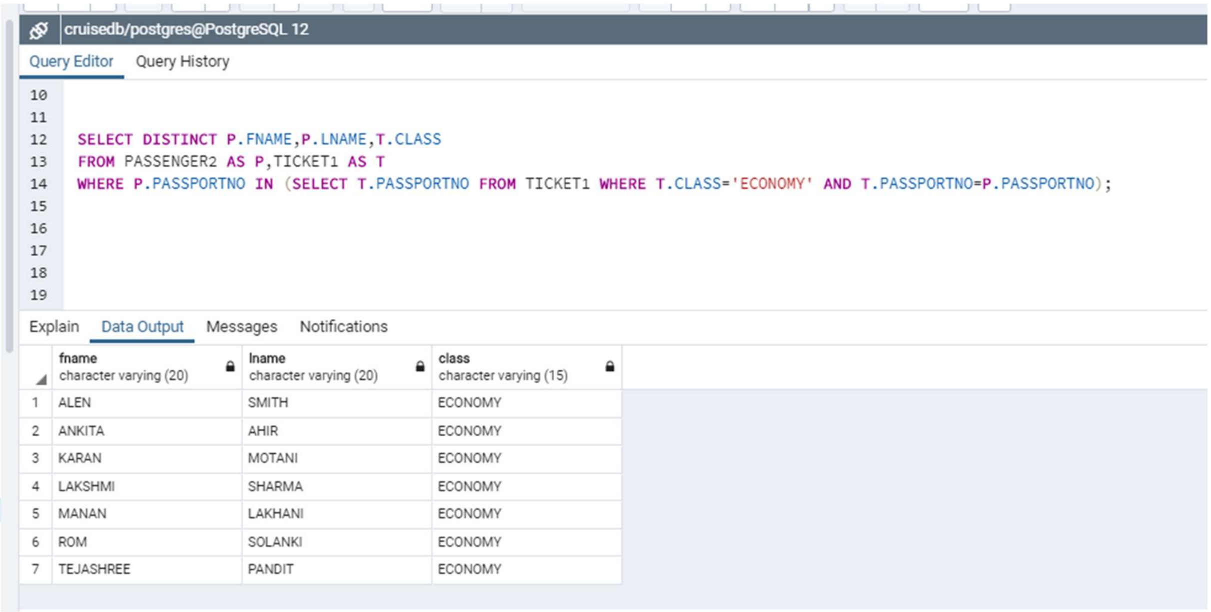


```sql

SELECT DISTINCT FNAME,M,LNAME
FROM EMPLOYEE1
WHERE JOBTYPE IN (SELECT JOBTYPE FROM EMPLOYEE1 WHERE FNAME='NIKITA');
```

	fname character varying (20)	m character varying (1)	lname character varying (20)
1	JAMES	P	BOND
2	NIKHIL	C	PAWAR
3	NIKITA	C	PAUL

2) Retrieve the names of all passengers who have booked their tickets in ECONOMY CLASS.

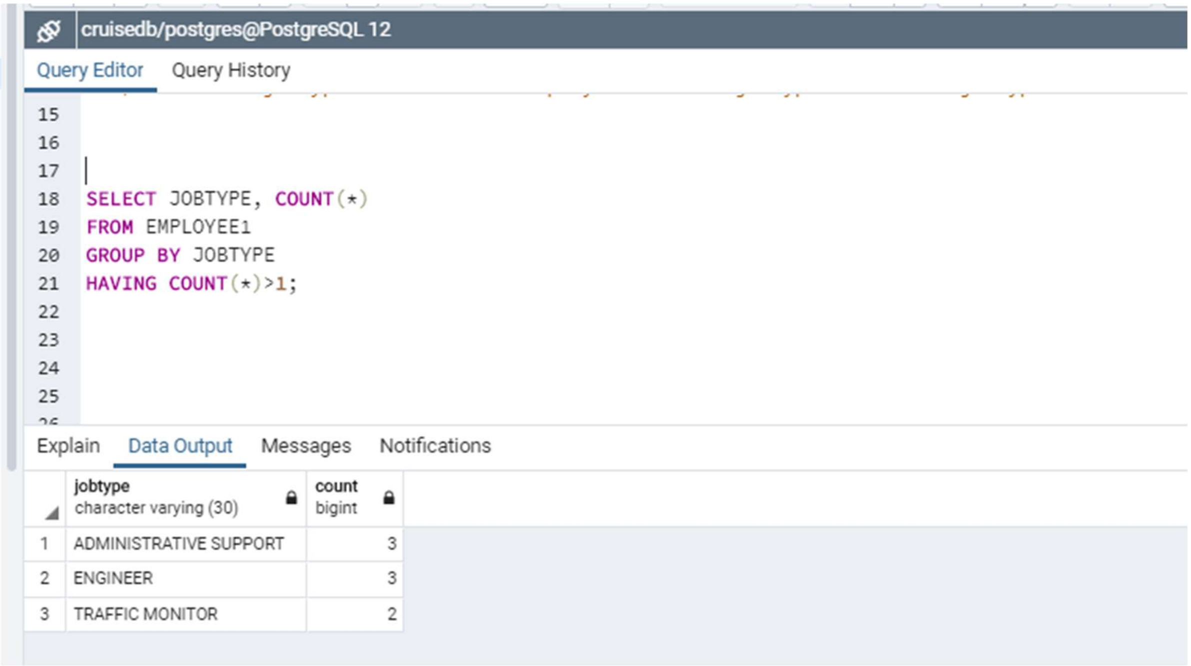


```sql
SELECT DISTINCT P.FNAME,P.LNAME,T.CLASS
FROM PASSENGER2 AS P,TICKET1 AS T
WHERE P.PASSPORTNO IN (SELECT T.PASSPORTNO FROM TICKET1 WHERE T.CLASS='ECONOMY' AND T.PASSPORTNO=P.PASSPORTNO);
```

	fname character varying (20)	lname character varying (20)	class character varying (15)
1	ALEN	SMITH	ECONOMY
2	ANKITA	AHIR	ECONOMY
3	KARAN	MOTANI	ECONOMY
4	LAKSHMI	SHARMA	ECONOMY
5	MANAN	LAKHANI	ECONOMY
6	ROM	SOLANKI	ECONOMY
7	TEJASHREE	PANDIT	ECONOMY

c) AGGREGATE FUNCTIONS:

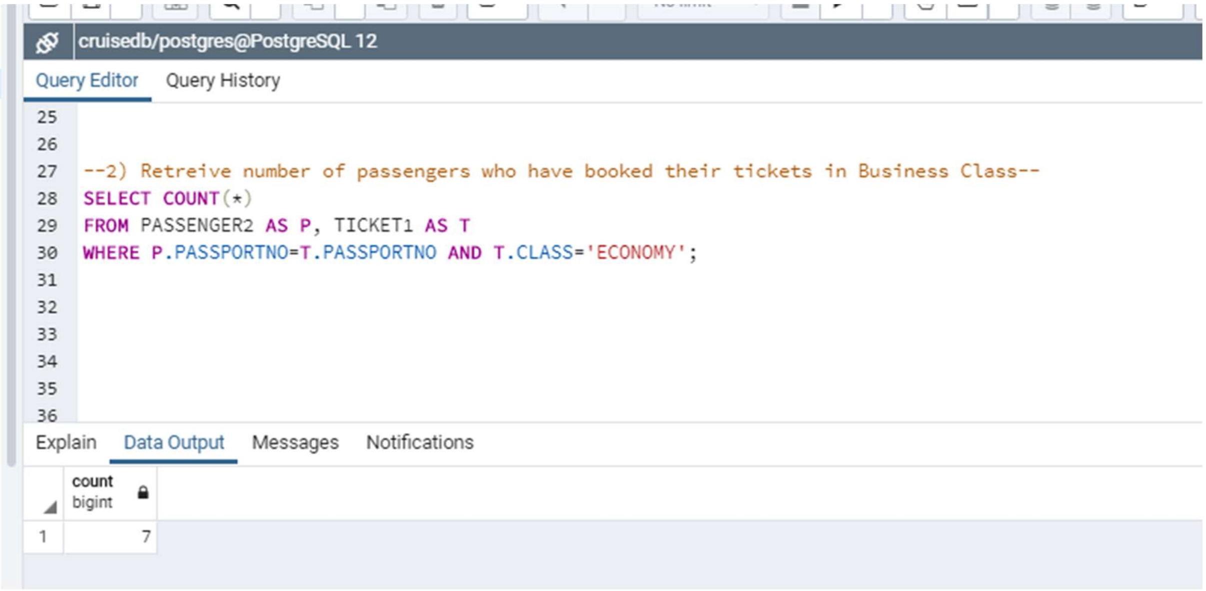
1) Retrieve job type and number of employees in each job type where each job type has more than 1 employee



```
SELECT JOBTYPE, COUNT(*)
FROM EMPLOYEE1
GROUP BY JOBTYPE
HAVING COUNT(*)>1;
```

	jobtype character varying (30)	count bigint
1	ADMINISTRATIVE SUPPORT	3
2	ENGINEER	3
3	TRAFFIC MONITOR	2

2)Retrieve number of passengers who have booked their tickets in Business class



```
--2) Retreive number of passengers who have booked their tickets in Business Class--
SELECT COUNT(*)
FROM PASSENGER2 AS P, TICKET1 AS T
WHERE P.PASSPORTNO=T.PASSPORTNO AND T.CLASS='ECONOMY';
```

	count bigint
1	7

Conclusion

Doing this project was a wonderful experience for me. It helped me in strengthening my concepts in Unit 3 and Unit 4 of DBMS course. It gave me confidence in creating any mini world example as a database and relations. It also taught me various concepts like normalisation, testing for lossless join property and creating triggers and complex queries.