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 Reservation System

PES1201802127 Siddharth K Rao
4th Sem Sec. A Roll No. 62

PROJECT SUMMARY

A Cruise Reservation System has been implemented here. The functions of this database system is to maintain records of the various transactions that happen, from the booking of ticket to the departure of the passenger. The various entities used are PORTS, LOCATION, CRUISE, CRUISE_COMPANY, CRUISE_PORT_ACCESS, STATEROOM, FARE, TRAVELLER, TRAVELLER_ITINERARY, CRUISE_TRIP, END_USER.

The design of the database was done using an ER model. A schema and an ER diagram have been depicted for a better understanding. Various relations and tables have been shown and how each of them are related. The cardinality of these relationships is also given. The tables were made using the DDL commands.

Triggers were used to counter certain constraints like updating of availability of rooms once it has been allocated to a passenger. Certain queries were also created to find out the people travelling in a particular month, most common type of room type booked, to find the duration of the trip of passengers travelling from source to the destination, people who have paid more than a certain amount to travel in cruise, etc.

This database is capable of automating all the basic processes of a reservation system. The Cruise Reservation System created is capable of managing large transactions. It has its limitations but is a good approach to automating the reservation system.

Introduction

The mini world being depicted here is of a Cruise Reservation System.

Cruise Reservation System is one of the most used database systems in the world. It is an example of Transaction processing systems. **Transaction processing systems** are systems with large databases and hundreds of concurrent users executing database transactions. These systems require high availability and fast response time for hundreds of concurrent users. In this project, the database part of the whole system is dealt with insertions, deletions, and updates as the primary task of our system along with maintenance of the integrity of the system at all stages of a transaction. The system that is being described in this report handles everything that the most practical systems would do.

The basic transactions of this database is that there is a User who can be an agent also books the ticket, the cruise company receives the booking, collects the fare and then allocates the room after allowing the user to choose the room type, taking the source and destination in mind. A traveller itinerary is then issued for every passenger, which has the important details of their trip such as the trip ID, room number and the ship ID they are travelling in.

Some of the important entities are as follows:

<u>LOCATION:</u> This entity is uniquely identified by the ZIP and the other non-prime attributes are the Port state, city, and the country it is located in.

<u>PORTS:</u> In this entity, PORT_ID is the Primary Key and gives us the details about the ports such as the port name, their size which varies from medium to very large and the zip code of that port.

<u>CRUISE_COMPANY:</u> This gives us the details about the different cruise companies. Each company has a Company ID which uniquely determines the name of the company.

<u>CRUISE:</u> This entity gives us details about the ships/cruise. Each cruise has a unique cruise number which determines all the other attributes. This gives us information about the cruise capacity, cruise name and the company they belong to.

<u>CRUISE_PORT_ACCESS:</u> This gives us information about the ships that are present in certain ports. Both ports and cruise_ship together form the primary key.

<u>STATEROOM:</u> This entity gives details about the rooms. It specifies the ship code along with the room number and its availability. It also tells about the type of rooms available in particular decks of those ships. Cruise_ship which is the cruise code along with the room number form the primary key.

END_USER: This is used to store the details about the user who books the tickets. Now the user could be a normal passenger or an agent who ca book tickets for passengers or himself. Here the primary key is the user’s email ID.

CRUISE_TRIP: One of the important tables which stores details such as cruise_trip_id which is basically the ID given to the user along with the number of guests travelling with the user. It gives insights about the source and destination along with the arrival and departure dates. Cruise_trip_id is the primary key here which determines the other non-prime attribute keys.

FARE: This gives details about total cost that a passenger/user would have to pay with the tax and discount details. This is a weak entity and depends on cruise_trip_id of the CRUISE_TRIP table.

TRAVELLER: This is used to store traveller details such as their first and last names, their phone numbers, and the country they belong to.

TRAVELLER_ITINERARY: This is a weak entity which gives the passenger certain important details such as their cruise_trip_id, their room number and the cruise_ship ID, which tells them about the ship.

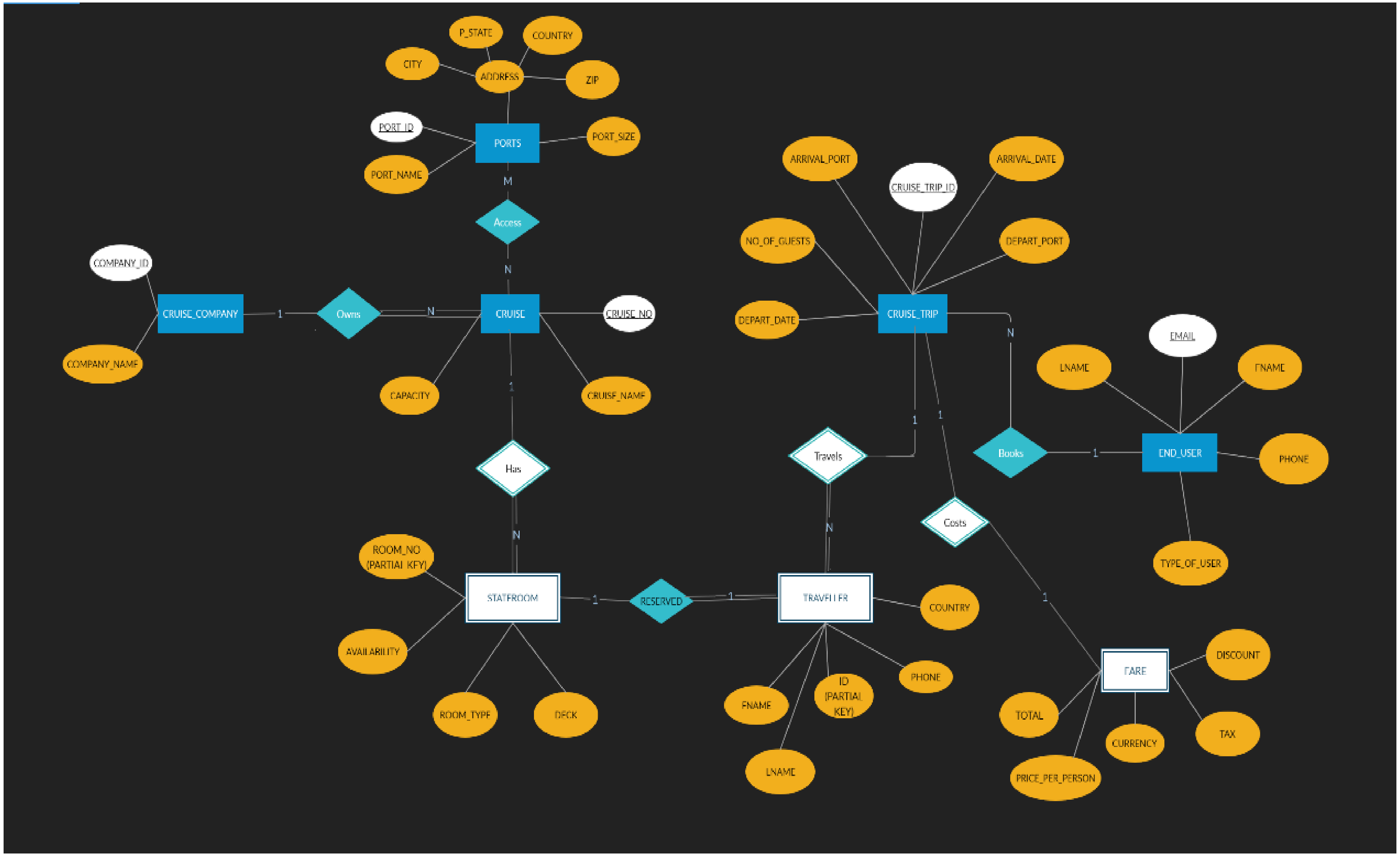
ASSUMPTIONS

1) Every ship/cruise must belong to a Cruise Company. So, a full participation of cruise is justified. A company can have multiple cruises, but a cruise should belong to a cruise company.
2) Each stateroom belongs to a cruise and cruise has multiple staterooms. Without a cruise, stateroom cannot exist. Stateroom is a dependent attribute (Cruise).
3) Each passenger who has booked the ticket must have a stateroom. However, staterooms can be empty, so it is not necessary that every stateroom belongs to a passenger, but the opposite is possible. So, a full participation of passenger is justified.
4) End User can be of two types:
   - Agent: Usually books for multiple travellers regularly.
   - Traveller: Usually books for himself or at most family/friends.

   A user can book tickets for multiple passengers and is an independent entity unlike passenger.

Data Model

<u>ER Diagram:</u>



One-to-One Binary Relationships (1:1)

- STATEROOM-reserved-TRAVELLER
- CRUISE_TRIP-costs-FARE

One-to-Many Binary Relationships (1:N)

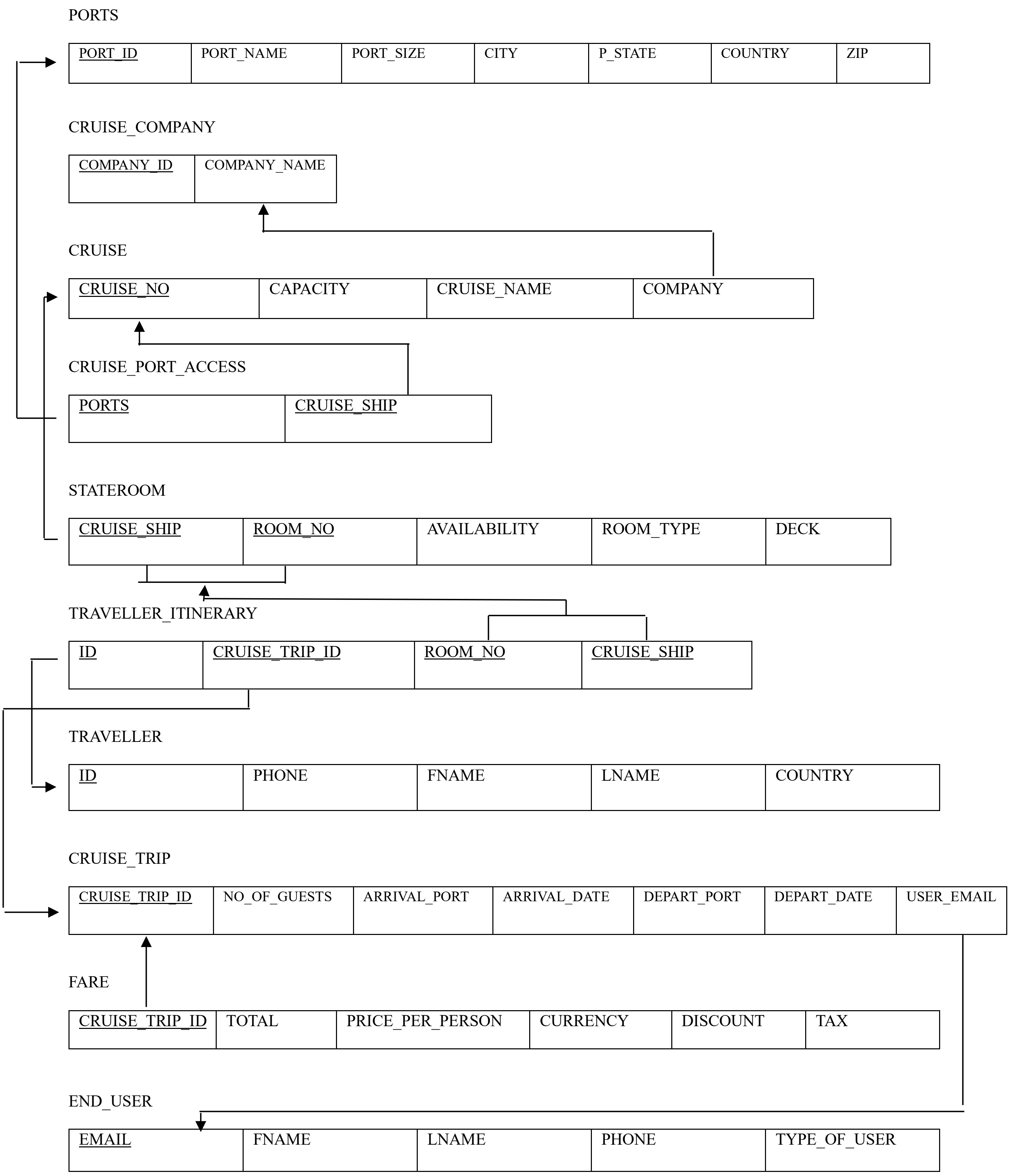
- CRUISE_COMPANY-owns-CRUISE
- CRUISE-has-STATEROOM
- CRUISE_TRIP-travels-TRAVELLER
- END_USER-books-CRUISE_TRIP

Many-to-Many Binary Relationships (M:N)

- PORTS-access-CRUISE

Relational Schema:

The schema below shows the relations between the entities.


PORTS
PORT_ID | PORT_NAME | PORT_SIZE | CITY | P_STATE | COUNTRY | ZIP
CRUISE_COMPANY
COMPANY_ID | COMPANY_NAME
CRUISE
CRUISE_NO | CAPACITY | CRUISE_NAME | COMPANY
CRUISE_PORT_ACCESS
PORTS | CRUISE_SHIP
STATEROOM
CRUISE_SHIP | ROOM_NO | AVAILABILITY | ROOM_TYPE | DECK
TRAVELLER_ITINERARY
ID | CRUISE_TRIP_ID | ROOM_NO | CRUISE_SHIP
TRAVELLER
ID | PHONE | FNAME | LNAME | COUNTRY
CRUISE_TRIP
CRUISE_TRIP_ID | NO_OF_GUESTS | ARRIVAL_PORT | ARRIVAL_DATE | DEPART_PORT | DEPART_DATE | USER_EMAIL
FARE
CRUISE_TRIP_ID | TOTAL | PRICE_PER_PERSON | CURRENCY | DISCOUNT | TAX
END_USER
EMAIL | FNAME | LNAME | PHONE | TYPE_OF_USER

FD and Normalization

<u>Functional Dependency (FD):</u>
The original ‘PORTS’ table had to be decomposed to form two relations called ‘PORTS’ and the other one as ‘LOCATION’ for normalization purposes.

<u>PORTS TABLE:</u>

PORT_ID -> {PORT_NAME, PORT_SIZE, ZIP}

So, in the ‘PORTS’ relation all the non-prime attributes are dependent on the Primary Key ‘PORT_ID’. ‘ZIP’ here is a Foreign Key and references ‘ZIP’ of ‘LOCATION’ relation.

<u>LOCATION:</u>

ZIP -> {P_STATE, CITY, COUNTRY}

In this relation, all the attributes are functionally dependent on the Primary Key ‘ZIP’.

<u>CRUISE_COMPANY:</u>

COMPANY_ID -> {COMPANY_NAME}

In the CRUISE_COMPANY relation, the non-prime attribute ‘COMPANY_NAME’ is functionally dependent on ‘COMPANY_ID’ which is the Primary Key.

<u>CRUISE:</u>

CRUISE_NO -> {CAPACITY, CRUISE_NAME, COMPANY}

The CRUISE relation has all the non-prime attributes being functionally dependent to the Primary Key ‘CRUISE_NO’. The ‘COMPANY’ attribute is a foreign key which references to the ‘COMPANY_NAME’ of the ‘CRUISE_COMPANY’ relation.

<u>CRUISE_PORT_ACCESS:</u>

In the CRUISE_PORT_ACCESS relation, both the attributes, that is ‘PORTS’ and ‘CRUISE_SHIP’ combine to form a single Primary Key, because for each port, there can be many ships and therefore to uniquely identify each row/tuple in the relation, both the attributes need to combine to become the Primary Key.

STATEROOM:

{CRUISE_SHIP, ROOM_NO} -> {AVAILABILITY, ROOM_TYPE, DECK}

In STATEROOM relation, the attributes ‘CRUISE_SHIP’ and ‘ROOM_NO’ together form the primary key as each ship can have multiple rooms, and all the other non-prime attributes are functionally dependent to this primary key.

END_USER:

EMAIL -> {FNAME, LNAME, PHONE, TYPE_OF_USER}

In the END_USER relation, all the non-prime attributes are functionally dependent on the primary key ‘EMAIL’.

CRUISE_TRIP:

CRUISE_TRIP_ID -> {NO_OF_GUESTS, ARRIVAL_PORT, ARRIVAL_DATE, DEPART_PORT, DEPART_DATE, USER_EMAIL}

In the CRUISE_TRIP relation, all the non-prime attributes are functionally dependent on the primary key ‘CRUISE_TRIP_ID’ and ‘USER_EMAIL’ references ‘EMAIL’ (primary) key of the ‘END_USER’ relation.

FARE:

CRUISE_TRIP_ID -> {TOTAL, PRICE_PER_PERSON, CURRENCY, DISCOUNT, TAX}

In the FARE relation, all the non-prime attributes are functionally dependent on the key CRUISE_TRIP_ID. FARE is a weak entity and it cannot exist if CRUISE_TRIP does not exist. So, CRUISE_TRIP_ID of ‘FARE’ references ‘CRUISE_TRIP_ID’ of ‘CRUISE_TRIP’
relation.

TRAVELLER:

ID - > {PHONE, FNAME, LNAME, COUNTRY}

The TRAVELLER relation is a weak entity, and ‘ID’ here is the partial key. Without CRUISE_TRIP, a TRAVELLER cannot exist. All the other non-prime attributes are functionally dependent on ‘ID’ which is the partial key.

TRAVELLER_ITINERARY:

In the TRAVELLER_ITINERARY relation, the attributes 'ROOM_NO' and 'CRUISE_SHIP' depend on the 'ID' and 'CRUISE_TRIP_ID'. Both 'ID' and 'CRUISE_TRIP_ID' references 'ID' from 'TRAVELLER' relation and 'CRUISE_TRIP_ID'
from 'CRUISE_TRIP' relation, respectively. The attributes 'ROOM_NO' and 'CRUISE_SHIP' references 'CRUISE_SHIP' and 'ROOM_NO' from the 'STATEROOM' relation.

Violations of Normalization:

1NF -> The normalization violation occurs when a tuple holds more than one value for an attribute and therefore the atomicity is lost. Atomicity here refers to values in the table not being further divided.
From the ER Diagram, we know that the address attribute is a composite attribute. So, 1NF will be violated. So, tackle this by breaking the composite attribute into 'ZIP', 'P_STATE', 'COUNTRY', 'CITY' attributes in the 'PORTS' relation. By doing this, we do not violate the atomicity and hence the schema will not violate 1NF because all values are atomic.

2NF -> For 2NF to not be violated, it must satisfy 1NF. And all non-key attributes must be fully functionally dependent on the Primary Key.
Here, 2NF is not violated as all the non-key attributes are fully dependent on the primary keys of their relation. But to understand how it gets violated, if we add the 'CAPACITY' attribute to the 'STATEROOM' relation, it violates the 2NF, because it depends only on the 'CRUISE_SHIP' attribute while the primary key is 'CRUISE_SHIP-ROOM_NO'.

3NF -> For 3NF to not be violated, it must satisfy 2NF. The other condition is there should be no transitive dependency for non-prime attributes. That means non-prime attributes should not be dependent on other non-prime attributes in a given table. So, a transitive dependency is a functional dependency in which $X \rightarrow Z$ (X determines Z) indirectly, by virtue of $X \rightarrow Y$ and $Y \rightarrow Z$ (where it is not the case that $Y \rightarrow X$).
In this schema, the 'PORTS' relation violates 3NF. The attributes 'P_STATE', 'CITY', 'COUNTRY' depend on the non-prime attribute 'ZIP'. Now, the 'ZIP' attribute depends on the primary key 'PORT_ID'. So, 'PORT_ID' is transitively dependent on the attributes that are dependent on the 'ZIP' attribute.
To make this table complies with 3NF we must break the table into two tables to remove the transitive dependency. So, we create two tables called 'PORTS' and the other one called 'LOCATION'.

After normalization, ‘PORTS’ relation has been decomposed to other relations called ‘LOCATION’ where ‘ZIP’ is the primary key and the other attributes are ‘P_STATE’, ‘CITY’, ‘COUNTRY’. The other relation is ‘PORTS’ where ‘PORT_ID’ is the primary key and the other attributes are ‘PORT_NAME’, ‘PORT_SIZE’ and ‘ZIP’ which references to ‘ZIP’ in the ‘LOCATION’ relation.
Test for lossless join has also been done.

```
SELECT *
FROM
    PORTS
NATURAL JOIN LOCATION;
```

No data was lost and all the rows were present.

DDL

Tables are created below:

1) **LOCATION**

```
CREATE TABLE LOCATION(
ZIP INT NOT NULL,
P_STATE VARCHAR(50) NOT NULL,
CITY VARCHAR(50) NOT NULL,
COUNTRY VARCHAR(50) NOT NULL,
PRIMARY KEY(ZIP)
);
```

2) **PORTS**

```
CREATE TABLE PORTS(
PORT_ID VARCHAR(10),
PORT_NAME VARCHAR(100) NOT NULL,
PORT_SIZE VARCHAR(50) NOT NULL,
ZIP INT NOT NULL,
PRIMARY KEY (PORT_ID),
FOREIGN KEY (ZIP) REFERENCES LOCATION(ZIP)
);
```

3) **CRUISE_COMPANY**

```
CREATE TABLE CRUISE_COMPANY(
COMPANY_ID VARCHAR(50) ,
COMPANY_NAME VARCHAR(100) NOT NULL,
PRIMARY KEY (COMPANY_ID)
);
```

4) **CRUISE**

```
CREATE TABLE CRUISE(
CRUISE_NO VARCHAR(25) ,
CAPACITY INT NOT NULL,
CRUISE_NAME VARCHAR(100) NOT NULL,
COMPANY VARCHAR(50) NOT NULL,
PRIMARY KEY (CRUISE_NO),
FOREIGN KEY (COMPANY) REFERENCES
CRUISE_COMPANY(COMPANY_ID)
);
```

5) **CRUISE_PORT_ACCESS**

```
CREATE TABLE CRUISE_PORT_ACCESS(
PORTS VARCHAR(25) NOT NULL,
CRUISE_SHIP VARCHAR(25) NOT NULL,
PRIMARY KEY (PORTS,CRUISE_SHIP),
FOREIGN KEY (PORTS) REFERENCES PORTS(PORT_ID) ON DELETE
CASCADE,
FOREIGN KEY (CRUISE_SHIP) REFERENCES CRUISE(CRUISE_NO) ON
DELETE CASCADE
);
```

6) **STATEROOM**

```
CREATE TABLE STATEROOM(
CRUISE_SHIP VARCHAR(25) NOT NULL,
ROOM_NO VARCHAR(5) NOT NULL,
AVAILABILITY VARCHAR(15) DEFAULT 'AVAILABLE',
ROOM_TYPE VARCHAR(50) NOT NULL,
DECK VARCHAR(25) NOT NULL,
PRIMARY KEY (CRUISE_SHIP,ROOM_NO),
FOREIGN KEY (CRUISE_SHIP) REFERENCES CRUISE(CRUISE_NO) ON
DELETE CASCADE,
CHECK (ROOM_TYPE = 'Interior Upper/Lower' OR ROOM_TYPE = 'Ocean View
Upper/Lower' OR ROOM_TYPE = 'Suite Upper/Lower')
);
```

7) END_USER

```
CREATE TABLE END_USER (
EMAIL VARCHAR(50) NOT NULL,
FNAME VARCHAR(25) NOT NULL,
LNAME VARCHAR(25) NOT NULL,
PHONE varchar(10) ,
TYPE_OF_USER VARCHAR(25) NOT NULL,
PRIMARY KEY (EMAIL),
CONSTRAINT CHK_END_USER CHECK (LENGTH(PHONE) = 10)
);
```

8) CRUISE_TRIP

```
CREATE TABLE CRUISE_TRIP(
CRUISE_TRIP_ID VARCHAR(15) ,
NO_OF_GUESTS INT NOT NULL,
ARRIVAL_PORT VARCHAR(25) NOT NULL,
ARRIVAL_DATE DATE NOT NULL,
DEPART_PORT VARCHAR(25) NOT NULL,
DEPART_DATE DATE NOT NULL,
USER_EMAIL VARCHAR(50),
PRIMARY KEY (CRUISE_TRIP_ID),
FOREIGN KEY (USER_EMAIL) REFERENCES END_USER(EMAIL) ON
DELETE SET NULL
);
```

9) FARE

```
CREATE TABLE FARE(
CRUISE_TRIP_ID VARCHAR(15) ,
TOTAL DECIMAL(9,2) NOT NULL,
PRICE_PER_PERSON DECIMAL(9,2) NOT NULL,
CURRENCY VARCHAR(5) NOT NULL,
DISCOUNT DECIMAL(4,2) NOT NULL,
TAX DECIMAL(5,2) NOT NULL,
PRIMARY KEY (CRUISE_TRIP_ID),
FOREIGN KEY (CRUISE_TRIP_ID) REFERENCES
CRUISE_TRIP(CRUISE_TRIP_ID)ON DELETE CASCADE
);
```

10) TRAVELLER

```
CREATE TABLE TRAVELLER(
ID INT auto_increment PRIMARY KEY,
PHONE VARCHAR(10),
FNAME VARCHAR(25) NOT NULL,
LNAME VARCHAR(25) NOT NULL,
COUNTRY VARCHAR(25) NOT NULL,
CONSTRAINT CHK_TRAVELLER CHECK (LENGTH(PHONE) = 10)
);
```

11) TRAVELLER_ITINERARY

```
CREATE TABLE TRAVELLER_ITINERARY(
ID INT auto_increment,
CRUISE_TRIP_ID VARCHAR(15) NOT NULL,
ROOM_NO VARCHAR(5),
CRUISE_SHIP VARCHAR(25),
PRIMARY KEY (ID,CRUISE_TRIP_ID,ROOM_NO,CRUISE_SHIP),
FOREIGN KEY (ID) REFERENCES TRAVELLER(ID),
FOREIGN KEY (CRUISE_TRIP_ID) REFERENCES
CRUISE_TRIP(CRUISE_TRIP_ID) ON DELETE CASCADE,
FOREIGN KEY (CRUISE_SHIP,ROOM_NO) REFERENCES
STATEROOM(CRUISE_SHIP,ROOM_NO)
);
```

Cascades have been used wherever applicable, to remove irregularity in the data when it is updated.

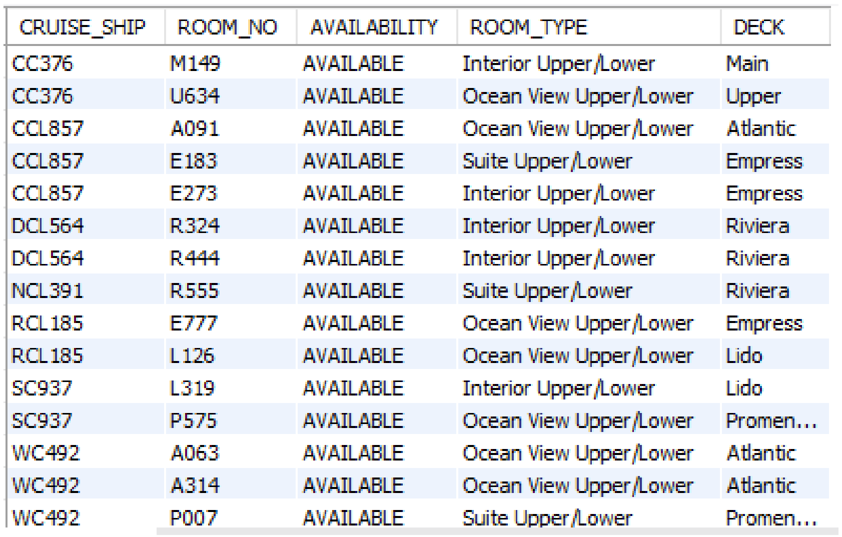
Triggers

1) Create a Trigger which sets the room availability to 'UNAVAILABLE' for a particular room, automatically when a traveller itinerary has been issued i.e. when new values are inserted to TRAVELLER_ITINERARY.

```
CREATE TRIGGER ROOM_UPDATE
```

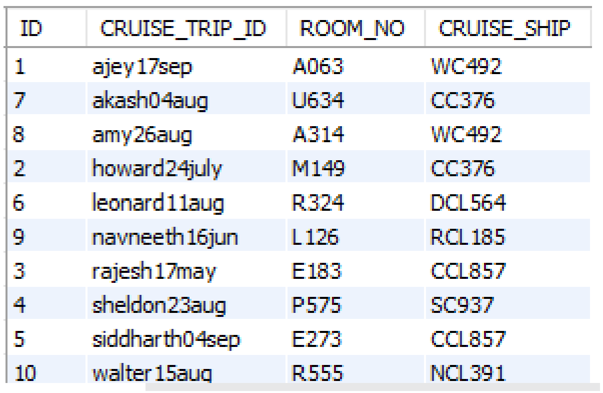
```
AFTER INSERT
ON TRAVELLER_ITINERARY
FOR EACH ROW
UPDATE STATEROOM AS ST
SET AVAILABILITY = 'UNAVAILABLE' WHERE
NEW.ROOM_NO = ST.ROOM_NO AND NEW.CRUISE_SHIP = ST.CRUISE_SHIP;
```

<u>SCREENSHOT:</u>



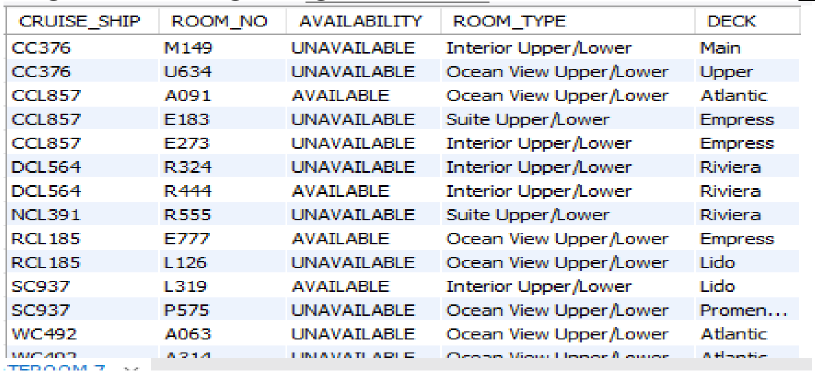
CRUISE_SHIP	ROOM_NO	AVAILABILITY	ROOM_TYPE	DECK
CC376	M149	AVAILABLE	Interior Upper/Lower	Main
CC376	U634	AVAILABLE	Ocean View Upper/Lower	Upper
CCL857	A091	AVAILABLE	Ocean View Upper/Lower	Atlantic
CCL857	E183	AVAILABLE	Suite Upper/Lower	Empress
CCL857	E273	AVAILABLE	Interior Upper/Lower	Empress
DCL564	R324	AVAILABLE	Interior Upper/Lower	Riviera
DCL564	R444	AVAILABLE	Interior Upper/Lower	Riviera
NCL391	R555	AVAILABLE	Suite Upper/Lower	Riviera
RCL185	E777	AVAILABLE	Ocean View Upper/Lower	Empress
RCL185	L126	AVAILABLE	Ocean View Upper/Lower	Lido
SC937	L319	AVAILABLE	Interior Upper/Lower	Lido
SC937	P575	AVAILABLE	Ocean View Upper/Lower	Promen...
WC492	A063	AVAILABLE	Ocean View Upper/Lower	Atlantic
WC492	A314	AVAILABLE	Ocean View Upper/Lower	Atlantic
WC492	P007	AVAILABLE	Suite Upper/Lower	Promen...

These are some of the rooms available as of now. Once the traveller itinerary is issued to the passenger, based on the room number allotted to the passengers, those rooms will become UNAVAILABLE in the STATEROOM relation.



ID	CRUISE_TRIP_ID	ROOM_NO	CRUISE_SHIP
1	ajey17sep	A063	WC492
7	akash04aug	U634	CC376
8	amy26aug	A314	WC492
2	howard24july	M149	CC376
6	leonard11aug	R324	DCL564
9	navneeth16jun	L126	RCL185
3	rajesh17may	E183	CCL857
4	sheldon23aug	P575	SC937
5	siddharth04sep	E273	CCL857
10	walter15aug	R555	NCL391

This is a screenshot of the TRAVELLER_ITINERARY table, which gives us information about the passengers travelling along with room number and the cruise_ship_id.



CRUISE_SHIP	ROOM_NO	AVAILABILITY	ROOM_TYPE	DECK
CC376	M149	UNAVAILABLE	Interior Upper/Lower	Main
CC376	U634	UNAVAILABLE	Ocean View Upper/Lower	Upper
CCL857	A091	AVAILABLE	Ocean View Upper/Lower	Atlantic
CCL857	E183	UNAVAILABLE	Suite Upper/Lower	Empress
CCL857	E273	UNAVAILABLE	Interior Upper/Lower	Empress
DCL564	R324	UNAVAILABLE	Interior Upper/Lower	Riviera
DCL564	R444	AVAILABLE	Interior Upper/Lower	Riviera
NCL391	R555	UNAVAILABLE	Suite Upper/Lower	Riviera
RCL185	E777	AVAILABLE	Ocean View Upper/Lower	Empress
RCL185	L126	UNAVAILABLE	Ocean View Upper/Lower	Lido
SC937	L319	AVAILABLE	Interior Upper/Lower	Lido
SC937	P575	UNAVAILABLE	Ocean View Upper/Lower	Promen...
WC492	A063	UNAVAILABLE	Ocean View Upper/Lower	Atlantic
WC492	A314	UNAVAILABLE	Ocean View Upper/Lower	Atlantic

This is the updated STATEROOM table after new values are added to TRAVELLER_ITINERARY table.

2) Create a trigger which the sets the discount to 10 if the updated value in the table is less than 10.

```
DELIMITER $$
CREATE TRIGGER TAX_CHECK
BEFORE UPDATE
ON FARE
FOR EACH ROW
BEGIN
IF (NEW.TAX>15) THEN
 SET NEW.TAX = NEW.TAX;
ELSE
 SET NEW.TAX = 15;
END IF;
END $$
```

SQL Queries

1) Write a query to find the number of guests travelling along with the person who has booked the tickets and is not an agent and show the departure and the arrival date.

```
SELECT
 USER_EMAIL,
   NO_OF_GUESTS,
   DEPART_DATE,
   ARRIVAL_DATE
FROM
 CRUISE_TRIP
WHERE
 USER_EMAIL IN
   (
      SELECT
          EMAIL
      FROM
          END_USER
      WHERE
          TYPE_OF_USER ='NORMAL'
 )
```

```
GROUP BY
 USER_EMAIL;
```

2) Write a query to find the total price of the person who is travelling to Singapore by arranging them from highest to lowest price.

```
SELECT
 CRUISE_TRIP_ID,
   CURRENCY,
   TOTAL AS TOTAL_PRICE
FROM
 FARE
WHERE
 CRUISE_TRIP_ID IN
   (
      SELECT
           CRUISE_TRIP_ID
      FROM
           CRUISE_TRIP
      WHERE
           ARRIVAL_PORT = 'SGSIN'
 )
ORDER BY
 TOTAL_PRICE DESC;
```

3) Finding all the passengers travelling in the month of October.

```
SELECT
 EU.FNAME,
   EU.LNAME,
   CT.DEPART_DATE AS DEPARTURE
FROM
 END_USER AS EU
INNER JOIN CRUISE_TRIP AS CT
 ON EU.EMAIL = CT.USER_EMAIL
WHERE
 EU.EMAIL IN
   (
      SELECT
           CT.USER_EMAIL
      FROM
```

```
            CRUISE_TRIP AS CT
        WHERE
             CT.DEPART_DATE >= '20201001' AND CT.DEPART_DATE < '20201101'
 )
ORDER BY DEPARTURE ASC;
```

4) To find the list of all the passengers who have spent more than $1000 for their voyage along with the source and destination ports.

```
SELECT
 CRUISE_TRIP_ID,
   DEPART_PORT AS DEPARTING_PORT,
   ARRIVAL_PORT AS DESTINATION,
   ARRIVAL_DATE AS ARRIVAL
FROM
 CRUISE_TRIP
WHERE
 CRUISE_TRIP_ID IN
   (
      SELECT
           CRUISE_TRIP_ID
      FROM
           FARE
      WHERE
           TOTAL >= 1000
 );
```

5) Finding all the people whose duration to reach the destination takes 5 or more days.

```
SELECT
 FNAME,
   LNAME,
   EMAIL
FROM
 END_USER
WHERE
 EMAIL IN
   (
      SELECT
           USER_EMAIL
      FROM
```

```
            CRUISE_TRIP
        WHERE
            DATEDIFF(ARRIVAL_DATE,DEPART_DATE) >= 5
    )
```

6) Query to find the number of days it takes to take to reach the destination from the source for every passenger.

```
SELECT
 EU.FNAME,
   EU.LNAME,
   CT.DEPART_PORT,
   CT.ARRIVAL_PORT,
   DATEDIFF(ARRIVAL_DATE,DEPART_DATE) AS DURATION
FROM
 CRUISE_TRIP AS CT
INNER JOIN END_USER AS EU
 ON CT.USER_EMAIL = EU.EMAIL
```

7) Finding the most commonly booked type of room.

```
SELECT
 ROOM_TYPE,
   COUNT(*) AS TOTAL
FROM
 STATEROOM
WHERE
 AVAILABILITY = 'UNAVAILABLE'
GROUP BY ROOM_TYPE
ORDER BY TOTAL DESC;
```

8) Finding all the ships which can carry more than 2500 people.

```
SELECT
 CRUISE_NO,
   CRUISE_NAME,
   CAPACITY
FROM
 CRUISE
WHERE
 CAPACITY >= 2500
```

```
ORDER BY CAPACITY DESC;
```

9) Query to find a list of all the ships travelling between given two locations.

```
SELECT
 TI.CRUISE_SHIP,
   C.CRUISE_NAME,
   CT.DEPART_PORT,
   CT.ARRIVAL_PORT,
   COUNT(*) AS TOTAL
FROM
 TRAVELLER_ITINERARY AS TI
INNER JOIN CRUISE_TRIP AS CT
 ON TI.CRUISE_TRIP_ID = CT.CRUISE_TRIP_ID
INNER JOIN CRUISE AS C
 ON TI.CRUISE_SHIP = C.CRUISE_NO
GROUP BY TI.CRUISE_SHIP;
```

Conclusion

An efficient database management is important for every industry, especially the tourism industry where n numbers of transactions are happening every minute. This Cruise Reservation System database that has been created, tries to automate the whole process. There are a few limitations and constraints that are involved where future enhancement can be done.

LIMITATIONS:

- The more complex operations like refund on a cancellation are not done in this project.
- In an ideal world the details of the drivers also should be provided to the user which has not been done in this project.
- Many more entities such as HOP, which tells us about a passenger travelling to some destination midway between the destination port and the arrival port.

These are some of the limitations.

FUTURE ENHANCEMENTS:

- Application of GUI and front-end software.
- Web check in system could be emulated.
- Live Cruise Tracking details could be stored in the database and shared with the passengers.