AIRLINE MANAGEMENT SYSTEM

TEAM NAME: MIRACLE CODERS

EQUALLY CONTRIBUTED BY

Pothumudi Sony Priya

Potti Sruthik

Potturi Sarvani

Pemmadi Maha Lakshmi

Pranathi Boggarapu

Prajapati Kiran Kumar

Prajith Kumar Shanmugam

Pranay Kasibhatta

TABLE OF CONTENTS

SPRINT-1

1.1 INTRODUCTION:

This document aims to provide the data requirements for the development of a reservation system for an airline. The system will include tables to store passengers, flights, airlines, travel classes, and reservation status information.

1.2 DATA REQUIREMENTS:

1. **Passenger details** :
   This includes the information of the passenger using the following attributes and their respective datatype & size
   - Passenger ID (NUMBER(10))
   - First (VARCHAR(20))
   - Middle (VARCHAR(20))
   - Last (VARCHAR(20))
   - Gender (CHAR(1))
   - Phone (NUMBER(10))
   - Email (VARCHAR(50))
   - PassengerType (VARCHAR(20))

   PK – PassengerID

2. **Flight information :**
   This includes the information of the flight using the following attributes and their respective datatype & size
   - FlightNumber (CHAR(10))
   - Destination (VARCHAR(50))
   - FlightDate (DATE)
   - DepartureTime (TIMESTAMP(0))
   - ArrivalTime (TIMESTAMP(0))
   - FlightTime (TIMESTAMP(0))
   - AirlineCode (CHAR(10))

   PK-FlightNumber
   FK-AirlineCode (References AIRLINE Table)

3. **Airline details** :
   This includes the information of the Airline using the following attributes and their respective datatype & size
   - AirlineCode (CHAR(10))
   - AirlineName (VARCHAR(50))
   - HeadquartersAddress (VARCHAR(50))

   PK-AirlineCode

4**. Travel agency details:**

This includes the information of the Travel Agency using the following attributes and their respective datatype & size

- TPassengerID (NUMBER(10))
- TravelAgencyID (NUMBER(10))
- TravelAgencyName (VARCHAR(50))
- AgencyAddress (VARCHAR(50))
- AgencyPhone (NUMBER(10))
- BookingAgentName (VARCHAR(50))
- AgencyDiscountCode (NUMBER(10))

PK-TPassengerID

FK-AgencyDiscountCode (References AGENCY_DISCOUNT Table)

5. Agency discount details:

This includes the information of the Agency Discount using the following attributes and their respective datatype & size

a. AgencyDiscountCode (NUMBER(10))
b. AgencyDiscountAmount (DEC(8,2))

PK-AgencyDiscountCode

6. Reservation details:

This includes the information of the Reservation using the following attributes and their respective datatype & size

a. ReservationID (NUMBER(10))
b. PassengerID (NUMBER(10))
c. FlightNumber (CHAR(10))
d. ReservationDate (DATE)
e. SeatNumber (CHAR(4))
f. Cost (DEC(8,2))
g. TravelClassID (NUMBER(10))
h. ReservationStatusID (NUMBER(10))

PK-ReservationID

FK-FlightNumber (References FLIGHT Table)

FK-PassengerID (References PASSENGER Table)

FK-TravelClassID (References TRAVEL_CLASS Table)

FK-ReservationStatusID (References RESERVATION_STATUS Table)

7. **Reservation status details:**
   This includes the information of the Reservation Status using the following attributes and their respective datatype & size
   a. ReservationStatusID (NUMBER(10))
   b. ReservationStatus (NUMBER(10))
   PK-ReservationStatusID

8. **Travel class details:**
   This includes the information of the Reservation using the following attributes and their respective datatype & size
   a. TravelClassID (NUMBER(10))
   b. TravelClass (CHAR(20))
   PK-TravelClassID

1.3 Functional Requirements

- The system should Store passenger information including their ID, name, gender, contact details, and passenger type.
- Allow adding, updating, and deleting passenger records.
- Support querying and searching for passengers based on various criteria.
- Maintain information about airlines including their code, name, and headquarters address.
- Enable adding, updating, and deleting airline records.
- Support querying and searching for airlines based on different parameters.
- Store flight details such as flight number, destination, flight date, departure time, arrival time, flight time, and associated airline code.
- Allow adding, updating, and deleting flight records.
- Support querying and searching for flights based on various criteria.
- Maintain different travel classes and their corresponding IDs.
- Enable adding, updating, and deleting travel class records.
- Support querying and searching for travel classes based on their IDs or names.
- Store reservation status information including status IDs and names.
- Allow adding, updating, and deleting reservation status records.
- Support querying and searching for reservation statuses based on their IDs or names.
- Store reservation details such as reservation ID, passenger ID, flight number, reservation date, seat number, cost, travel class ID, and reservation status ID.

- Enable adding, updating, and deleting reservation records.
- Support querying and searching for reservations based on various criteria.
- Store information about agency discounts including discount codes and discount amounts.
- Allow adding, updating, and deleting agency discount records.
- Support querying and searching for agency discounts based on their codes or amounts.
- Maintain information about travel agencies including their IDs, names, addresses, phone numbers, booking agent names, and associated agency discount codes.
- Enable adding, updating, and deleting travel agency records.
- Support querying and searching for travel agencies based on different parameters.

1.3.1 Non-functional requirements for the airline database:

Performance:

The database should handle a large number of concurrent users efficiently. Queries and transactions should be processed quickly to provide a responsive user experience. Indexing and optimization techniques should be employed to enhance performance.

Security:

The database should enforce appropriate access controls to ensure data privacy and security. User authentication and authorization mechanisms should be implemented. Sensitive information such as passwords and payment details should be stored securely using encryption.

Reliability:

The database should have backup and recovery mechanisms to prevent data loss in case of failures. It should provide data integrity and consistency mechanisms, such as constraints and validation rules. Error handling and logging should be implemented to detect and handle exceptions effectively.

Scalability:

The database should be designed to handle a growing volume of data and an increasing number of users. It should support horizontal and vertical scaling as per the organization's needs.

Data Integrity:

The database should enforce referential integrity constraints to maintain the consistency and accuracy of data. Proper data validation should be performed to ensure that only valid data is stored.

Maintainability:

The database design should follow best practices and be modular and well-documented. Changes to the database structure should be easily manageable without disrupting the system's functionality. The code and database schema should be maintainable and easily understandable by developers and administrators.

Compatibility:

The database should be compatible with the chosen database management system (DBMS) and other related technologies. It should support the required SQL standards and features. If required, integration with other systems or APIs should be considered for data exchange.

Data Backup and Recovery:

Regular backups of the database should be performed to ensure data can be restored in case of any issues or failures. Backup and recovery procedures should be well-documented and tested periodically.

Data Privacy and Compliance:

The database should comply with relevant data privacy regulations and standards (e.g., GDPR, HIPAA). Personally identifiable information (PII) and sensitive data should be handled securely and in accordance with privacy policies.

1.4 Project Goals:

Efficiently store and manage passenger information:

- Store passenger details such as first name, middle name, last name, gender, phone number, email, and passenger type.
- Ensure the PassengerID serves as the primary key for quick and unique identification of each passenger.

Manage airline information:

- Store airline details including the airline code, airline name, and headquarters address.
- Ensure the AirlineCode serves as the primary key for efficient retrieval of airline information.

Track flight details:

- Store flight information such as flight number, destination, flight date, departure time, arrival time, flight time, and the associated airline.
- Use the Flight Number as the primary key for easy identification of each flight.
- Establish a foreign key relationship with the AirlineCode from the AIRLINE table to associate each flight with its respective airline.

Define travel classes:

- Maintain a table to store various travel class types offered by the airline.
- Use TravelClassID as the primary key to uniquely identify each travel class.

Manage reservation statuses:

- Maintain a table to store different reservation statuses, such as "booked," "confirmed," "canceled," etc.
- Assign a ReservationStatusID as the primary key for efficient management of reservation statuses.

Track reservations:

- Store reservation information, including the reservation ID, associated passenger, flight number, reservation date, seat number, cost, travel class, and reservation status.

- Utilize ReservationID as the primary key to uniquely identify each reservation.
- Establish foreign key relationships with PassengerID, FlightNumber, TravelClassID, and ReservationStatusID to link reservations with corresponding passenger, flight, travel class, and reservation status.

Manage agency discounts:

- Store agency discount codes along with their corresponding discount amounts.
- Use AgencyDiscountCode as the primary key for efficient tracking of agency discounts.
- Establish a foreign key relationship with AgencyDiscountCode in the TRAVEL_AGENCY table to associate discounts with travel agencies.

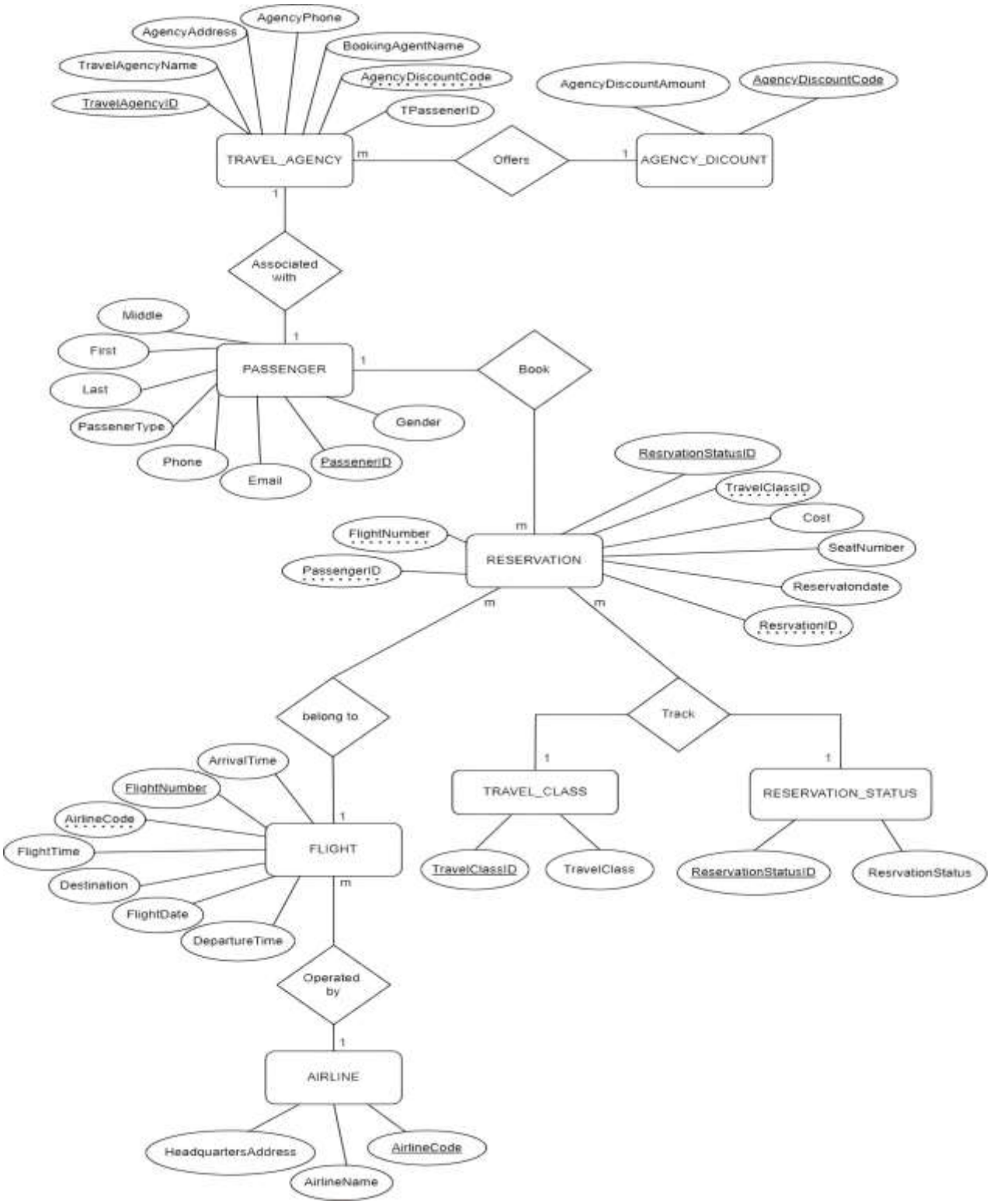
Track travel agencies:

- Store travel agency details such as the agency ID, agency name, agency address, agency phone number, booking agent name, and the associated agency discount code.
- Utilize TPassengerID as the primary key to uniquely identify each travel agency.
- Establish a foreign key relationship with the AgencyDiscountCode from the AGENCY_DISCOUNT table to associate each travel agency with their respective discount code.

RELATIONSHIP:

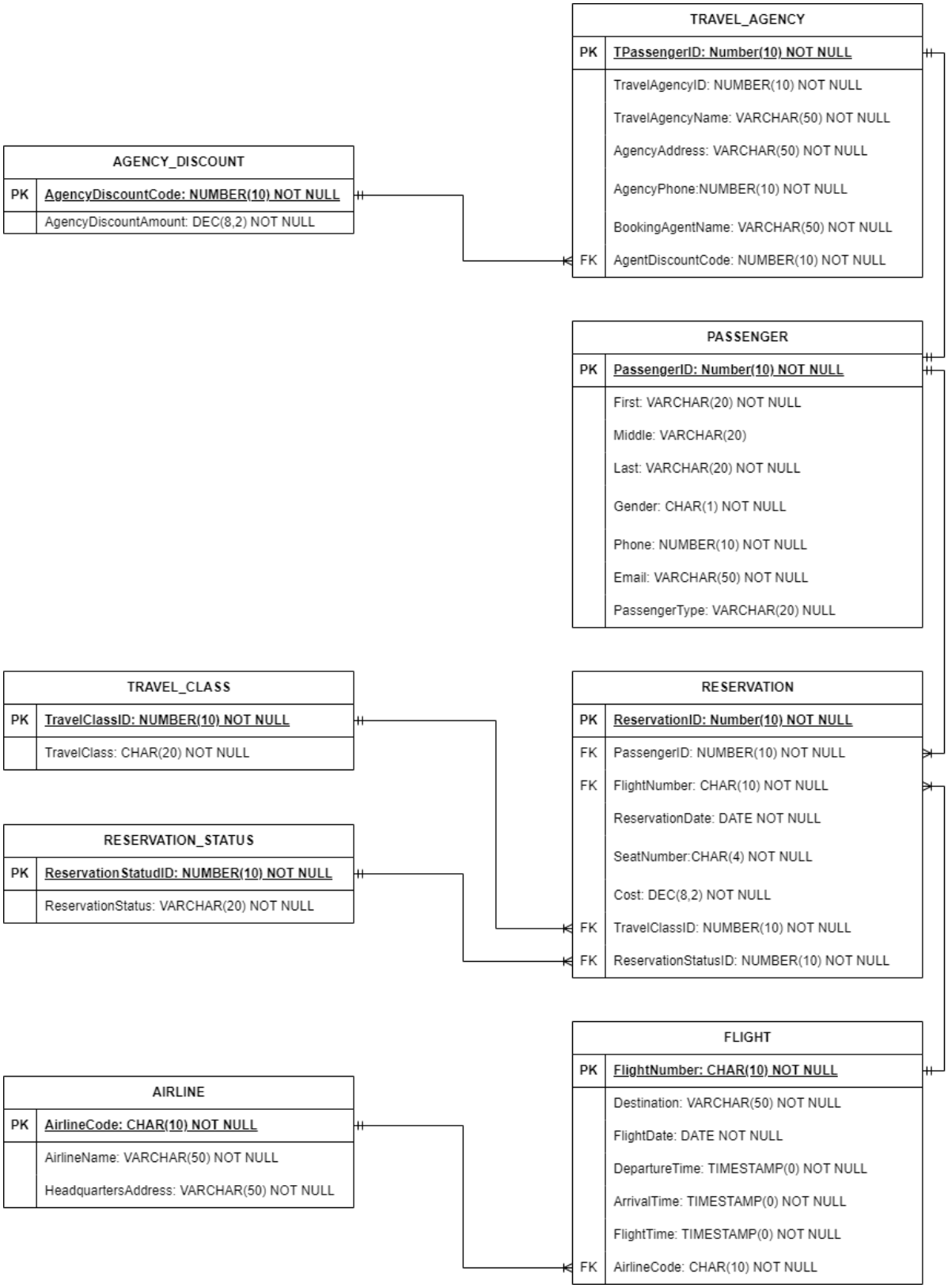
- ✓ One Passenger can have multiple reservations
- ✓ One Airline can have multiple flights
- ✓ Multiple flights can belong to one Airline
- ✓ One travel class can be used in multiple reservations
- ✓ One reservation status can be used in multiple reservations
- ✓ One agency discount can be used by multiple travel agencies
- ✓ One passenger can be associated with only one travel agency
- ✓ Multiple reservations can belong to one passenger.
- ✓ Multiple reservations can belong to one flight.
- ✓ Multiple reservations can have the same travel class
- ✓ Multiple reservations can have the same reservation status
- ✓ Multiple travel agencies can use the same agency's discount

1.5 ER Diagram



SPRINT-2

2.1 Relational Schema



2.2 Normalization

- Normalization is the process of organizing the data in the database.
- Normalization is used to minimize the redundancy from a relation or set of relations. It also eliminates undesirable characteristics like Insertion, Update, and Deletion Anomalies.
- Normalization divides the larger table into smaller ones and links them using relationships.
- The normal form is used to reduce redundancy from the database table.

Rules for 1st NF:

- A single cell must have one value
- No rows may be duplicated
- Each table must be identified with a unique column

Rules for 2NF:

A relation is said to be in 2NF when:

- It satisfies the rules of 1NF
- All non-key attributes are fully functional and dependent on the primary key.
- And it shouldn't have any partial dependencies

Partial Dependency

In a relation, when a proper subset of a candidate key or a partial primary keycan define a non-key attribute it is said to have a partial dependency

  - Here we are referencing the tables which are already in 1NF so we need to check for the partial dependency

Rules for 3NF:

- A relation will be in 3NF if it is in 2NF and
- It should not contain any transitive partial dependency.

Transitive Partial Dependency:

A transitive dependency occurs when one non-prime attribute is dependenton another non-prime attribute.

Functional dependency pattern:

A→B and B→C;

Therefore A→C

Normalizing tables:

Passenger Table:

PK	PassengerID
	First
	Middle
	Last
	Gender
	Phone
	Email
	PassengerType

1CNF: This table violates 1st NF because the Phone number may contain a multivalued attribute. A single cell must not hold more than one value. So, it is split into a new table name as **"p_phonenumber"**.

After Normalization:

PK	PassengerID
	First
	Middle
	Last
	Gender
	Email
	PassengerType

p_phonenumber Table:

FK	PassengerID
	Phone

2CNF: This table does not violate the 2 NF. There are no non-prime attributes depending on the partial primary key hence there are no partial dependencies.

3CNF: In p_phonenumber Table, all non-key attributes are directly dependent on the foreign key and there are no non-key attributes dependent on other non-key attributes i.e. there is no transitive dependency. Hence, it doesn't violate 3NF.

Airline Table:

PK	AirlineCode
	AirlineName
	HeadquatersAddress

1CNF: This table does not violate the 1st NF. There are no repeating groups or composite values observed.

2CNF: This table does not violate the 2 NF. There are no non-prime attributes depending on the partial primary key hence there are no partial dependencies.

3CNF: In the Airline Table, all non-key attributes are directly dependent on the primary key and there are no non-key attributes dependent on other non-key attributes i.e. there is no transitive dependency. Hence, it doesn't violate 3NF.

Reservation Status:

PK	ReservationStatusID
	ReservationStatus

1CNF: This table does not violate the 1st NF. There are no repeating groups or composite values observed.

2CNF: This table does not violate the 2 NF. There are no non-prime attributes depending on the partial primary key hence there are no partial dependencies.

3CNF: In Reservation Status, all non-key attributes are directly dependent on the primary key and there are no non-key attributes dependent on other non-key attributes i.e. there is no transitive dependency. Hence, it doesn't violate 3NF.

Flight Table:

PK	FlightNumber
	Destination
	FlightDate
	DepartureTime
	ArrivalTime
	FlightTime
FK	AirlineCode

1CNF: This table does not violate the 1st NF. There are no repeating groups or composite values observed.

2CNF: This table does not violate the 2 NF. There are no non-prime attributes depending on the partial primary key hence there are no partial dependencies.

3CNF: In Flight Table, all non-key attributes are directly dependent on the primary key, foreign key and there are no non-key attributes dependent on other non-key attributes i.e. there is no transitive dependency. Hence, it doesn't violate 3NF.

Reservation Table:

PK	**ReservationID**
FK	**PassengerID**
FK	**FlightNumber**
FK	**TravelAgencyID**
	ReservationDate
	SeatNumber
	Cost
FK	**TravelClassID**
FK	**ReservationStatusID**

1CNF: This table does not violate the 1st NF. There are no repeating groups or composite values observed.

2CNF: This table does not violate the 2 NF. There are no non-prime attributes depending on the partial primary key hence there are no partial dependencies.

3CNF: In the Reservation Table, all non-key attributes are directly dependent on the primary key, foreign key and there are no non-key attributes dependent on other non-key attributes i.e. there is no transitive dependency. Hence, it doesn't violate 3NF.

Agency_Discount:

PK	**AgencyDiscountCode**
	AgencyDiscountAmount

1CNF: This table does not violate the 1st NF. There are no repeating groups or composite values observed.

2CNF: This table does not violate the 2 NF. There are no non-prime attributes depending on the partial primary key hence there are no partial dependencies.

3CNF: In Agency_Discount Table, all non-key attributes are directly dependent on the primary key and there are no non-key attributes dependent on other non-key attributes i.e. there is no transitive dependency. Hence, it doesn't violate 3NF.

Travel_Agency:

PK	**TravelAgencyID**
FK	**TPassengerID**
	TravelAgencyName
	AgencyAddress
	AgencyPhone
	BookingAgentName
FK	**AgentDiscountCode**

1CNF: This table violates 1st NF as the AgencyPhone may be a multivalued attribute. So, the AgencyPhone column is added to a new table named **"Travel_Ag_phone"**.

After Normalization:

PK	**TravelAgencyID**
FK	**PassengerID**
	TravelAgencyName
	AgencyAddress
	BookingAgentName
FK	**AgentDiscountCode**

Travel_Ag_phone Table:

FK	**TravelAgencyID**
	AgencyPhone

2CNF: This table does not violate the 2 NF. There are no non-prime attributes depending on the partial primary key hence there are no partial dependencies.

3CNF: In Agency_Discount Table, all non-key attributes are directly dependent on the primary key and there are no non-key attributes dependent on other non-key attributes i.e. there is no transitive dependency. Hence, it doesn't violate 3NF.

Travel_Class:

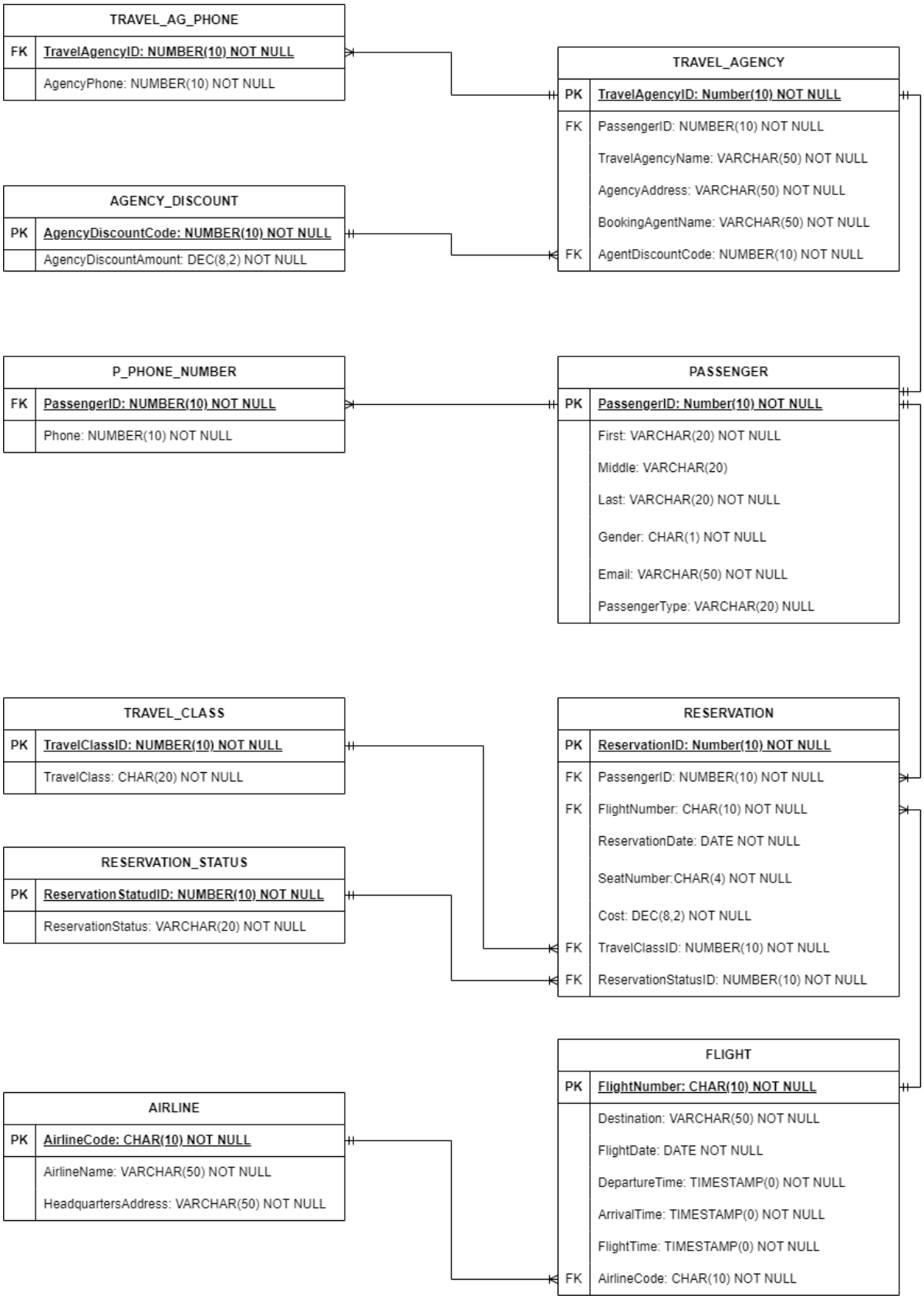
PK	TravelCalssID
	TravelClass

1CNF: This table does not violate the 1st NF. There are no repeating groups or composite values observed.

2CNF: This table does not violate the 2 NF. There are no non-prime attributes depending on the partial primary key hence there are no partial dependencies.

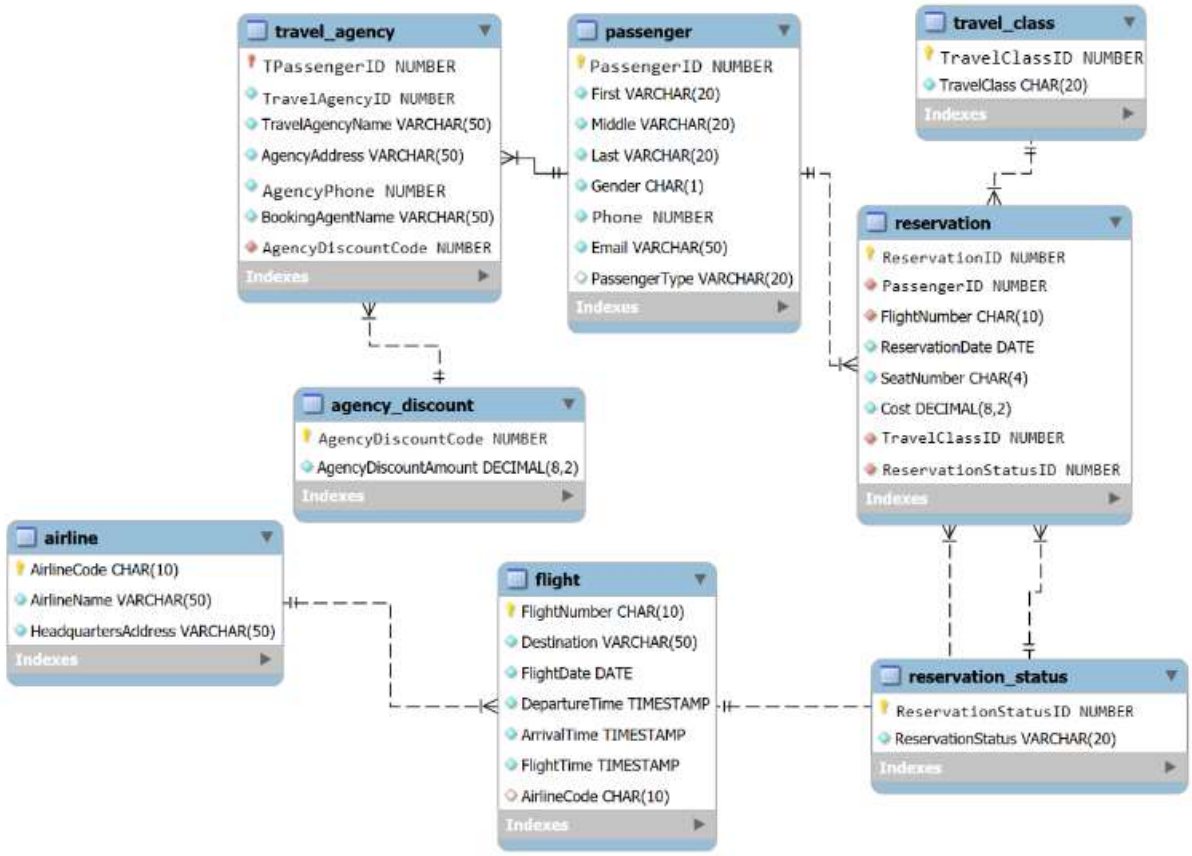
3CNF: In Travel_Class Table, all non-key attributes are directly dependent on the primary key and there are no non-key attributes dependent on other non-key attributes i.e. there is no transitive dependency. Hence, it doesn't violate 3NF.

2.3 Relational Schema after Normalization



SPRINT-3

Relational Schema



3.1 Table Creation

1. Write a SQL Query to create a table Passenger.

```
CREATE TABLE PASSENGER (PASSENGERID NUMBER PRIMARY KEY,
 FIRST VARCHAR(20) NOT NULL,
 MIDDLE VARCHAR(20) NOT NULL,
 LAST VARCHAR(20) NOT NULL,
 GENDER CHAR(1) NOT NULL,
 PHONE NUMBER NOT NULL,
 EMAIL VARCHAR(50) NOT NULL,
 PASSENGERTYPE VARCHAR(20));
```

2. Write a SQL Query to Create a table Agency_Discount.

```
CREATE  TABLE AGENCY DISCOUNT (AGENCYDISCOUNTCODE NUMBER
PRIMARY KEY,    AGENCYDISCOUNTAMOUNT NUMBER (8,2) NOT NULL);
```

3. Write a SQL Query to create a table Travel_agency.

```
CREATE TABLE TRAVEL_ AGENCY (TPASSENGERID NUMBER PRIMARY
KEY,
TRAVELAGENCYID NUMBER NOT NULL,
TRAVELAGENCYNAME VARCHAR(50) NOT NULL,
AGENCYADDRESS VARCHAR(50) NOT NULL,
AGENCYPHONE NUMBER NOT NULL,
BOOKINGAGENTNAME VARCHAR(50) NOT NULL,
AGENCYDISCOUNTCODE NUMBER NOT NULL, FOREIGN KEY
(AGENCYDISCOUNTCODE) REFERENCES AGENCY_DISCOUNT
(AGENCYDISCOUNTCODE));
```

4. Write a SQL Query to create a table Travel_class.

```
CREATE TABLE TRAVEL_CLASS(TRAVELCLASSID NUMBER PRIMARY
KEY,
TRAVELCLASS CHAR(20) NOT NULL);
```

5. Write a SQL Query to create a table Reservation_status.

```
CREATE TABLE RESERVATION_STATUS
(RESERVATIONSTATUSID NUMBER PRIMARY KEY,
 RESERVATIONSTATUS VARCHAR(20) NOT NULL);
```

6. Write a SQL Query to create a table Airline.

```
CREATE TABLE AIRLINE(AIRLINECODE CHAR(10) PRIMARY KEY,
AIRLINENAME VARCHAR(50) NOT NULL,
HEADQUARTERSADDRESS VARCHAR(50) NOT NULL)
```

7. Write a SQL Query to Create a table Flight.

```
CREATE TABLE FLIGHT (FLIGHTNUMBER CHAR(10) PRIMARY KEY,
DESTINATION VARCHAR(50) NOT NULL,
FLIGHTDATE DATE NOT NULL,
DEPARTURETIME TIMESTAMP(0) NOT NULL,
ARRIVALTIME TIMESTAMP(0) NOT NULL,
FLIGHTTIME TIMESTAMP(0) NOT NULL,
AIRLINECODE CHAR(10),
FOREIGN KEY (AIRLINECODE) REFERENCES  AIRLINE
(AIRLINECODE) );
```

8. Write a SQL Query to Create a table Reservation.

```
CREATE TABLE RESERVATION (RESERVATIONID NUMBER PRIMARY KEY,
PASSENGERID NUMBER NOT NULL,
FLIGHTNUMBER CHAR(1) NOT NULL,
RESERVATIONDATE DATE NOT NULL,
SEATNUMBER CHAR(4) NOT NULL,
COST DECIMAL (8,2) NOT NULL,
TRAVELCLASSID NUMBER NOT NULL,
RESERVATIONSTATUSID NUMBER NOT NULL,
FOREIGN KEY (PASSENGERID)
REFERENCES PASSENGER (PASSENGERID),
FOREIGN KEY (FLIGHTNUMBER)
REFERENCES FLIGHT (FLIGHTNUMBER),
FOREIGN KEY (TRAVELCLASSID)
REFERENCES TRAVEL_CLASS(TRAVELCLASSID),
FOREIGN KEY (RESERVATIONSTATUSID)
REFERENCES RESERVATION_STATUS (RESERVATIONSTATUSID) );
```

3.2 Fetching Records from Single Table

1)Write a SQL Query to List the passenger details.

Query >> select * from passenger ;

2)Write a SQL query to list the airline names and their headquarter addresses from the airline table.

Query >> select AirlineName, HeadquartersAddress from airline;

3)Write a SQL query to retrieve the first name of all passengers whose second letter is 'y' from the "passenger" table.

Note: use Columnname First to retrieve first name

Query >> select First from passenger where first like '_y%';

4)Write a SQL Query to Display the details of the airline with the code 'MO787878'.

Note: use Columnname Airlinecode to retrieve the code of an airline.

Query >> select * from airline where AirlineCode='M0787878';

5)Write a SQL Query to Display the details of the passenger first name that ends with the 'y'.

Note: use Columnname First to retrieve the first name

Query >> select * from passenger where first like '%y' ;

6)Write a SQL Query to count the flight which are having a destination "berlin"

Alias Name: COUNT

Query >> select Count(*) as COUNT from flight
where Destination='Berlin';

7)Write a SQL query to list the date that flights 'BBFGGGGGG5' and 'FT099999' are scheduled to travel.

Query >> select FLIGHTDATE from flight
where FlightNumber IN ('BBFGGGGGG5','FT099999');

8)Write a SQL Query to display all the travel agency details which should be ordered by agency address.

Query >> select * from travel_agency
order by AgencyAddress;

9)Write a SQL Query to Display the arrivaltime of flight number "HT135262".

Query >> select ArrivalTime from flight
where FlightNumber='HT135262';

10)Write a SQL query to show the details of the airlines.

Query >> select * from airline;

3.3 Fetching records from multiple tables

1)Write a query to show the flight number, flight destination, airport code, airport name, airline name, and airline headquarters details of a flight that has a destination of "London".

Query>>select f1.flightnumber, f1.destination, f1.flighttime, a1.airportcode,
a1.airportname, a2.airlinename, a2.headquartersaddress
From flight f1
Inner join airline a2
On f1.destination = "London" and f1.airlinecode = a2.airlinecode
Inner join flight_arrival_airport f2
On f1.flightnumber = f2.flightnumber
Inner join airport a1
On a1.airportcode = f2.airportcode;

2)Write a query to show the first name of passengers whose reservation has been successful along with their reservation details.

Note: reservationstatusid = 1 is considered as success

Query>>select p.first, r.*

From passenger p

Inner join reservation r

On r.passengerid = p.passengerid and r.reservationstatusid = 1;

3)Write a query to find the no of services offered by each flight and the total price of the services. The Query should display flight_id, number of services as "No_of_Services" (alias name), and the cost as "Total_price" (alias name). Order the results by Highest Total Price

Hint: The number of services can be calculated from the number of scheduled dates of the flight

Query>>select flightnumber, count(reservationDate) as No_of_services,

sum(cost) as Total_Price

From reservation

Group by flightnumber

Order by cost desc;

4)Write a query to display the reservation Id, cost, flightnumber, and flighttime for the flight reservation

Query>>select r.reservationid, r.cost, r.flightnumber, f.flighttime

From reservation r

Inner join flight f

On f.flightnumber = r.flightnumber;

5)Write a Join query to show the first name of the passengers, flight number, cost and travel class for their flight reservations.

Query>> select p.first, r.flightnumber, r.cost, t.travelclass

From passenger p

Inner join reservation r

On p.passengerid = r.passengerid

Inner join travel_class t

On t.travelclassid = r.travelclassid;

6)Write a query to display the name of the passenger “airana”, along with the flight number, seat number, and cost for her reservation

Query>>select p.first, r.flightnumber, r.cost

From passenger p

Inner join reservation r

On p.first = ‘ariana’ and p.passengerid = r.passengerid;

7)Write a query to show the first name of the passenger “lilly” and the details of her travel agency.

Note: use Columnname First to retrive the first name
AliasName:PASSENGERNAME

Query>>select p.first as PASSENGERNAME, t.travelagencyname,

t.bookingagentname, t.agencyaddress

from passenger p

inner join travel_agency t

On p.first = ‘lilly’ and p.passengerid = t.passengerid;

8)Write a query to show the information of the airline along with the flight details.

Query>>select f.flightnumber, f.flightdate, f.departuretime, a.airlinename, a.headquartersaddress
From flight f
Inner join airline a
On f.airlinecode = a.airlinecode;

9)Write a query to show the flight details for the reservation made on September 18, 2021.

Query>>select r.reservationDate, f.flightnumber, f.Destination, f.flightdate
From reservation r
Inner join flight f
On r.reservationdate = '18-SEP-21' and r.flightnumber = f.flightnumber;

3.4 Subqueries

1) Write a SQL Query to present the flight details of the passenger named '**Kylie'**.

Query >> select * from flight where FlightNumber in
(select FlightNumber from reservation where PassengerID in
(select PassengerID from passenger where First='kylie'));

2) Create a query to show the first names of passengers who have booked a flight ticket in the "**first**" class.

Note: use Column name **First** to retrieve the first name

Query >> select First from passenger where PassengerID in
(select PassengerID from reservation where TravelClassID in
(select TravelClassID from travel_class where TravelClass='first'));

3) Write a query to show the list of passengers' first names who have booked a flight with the "**British**" airline.

Note: use Column name **First** to retrieve the first name

Query >> select First from passenger where PassengerID in
(select PassengerID from reservation where FlightNumber in
(select FlightNumber from flight where AirlineCode in
(select AirlineCode from airline where AirlineName ='British')));

4) Write a query to show the reservation status of a passenger with the first name "**Scott**"

Note: use Column name **First** to retrieve the first name

Query>>select Reservationstatus from reservation_status
where ReservationStatusID in
(select ReservationStatusID from reservation where PassengerID in
(select PassengerID from passenger where First = 'Scott'));

5) Write a query to show the count of passengers who has booked a flight with "**United Airways**".

Alias Name: **COUNT**

Query >> select count (*) as COUNT from reservation
where FlightNumber in (select FlightNumber from flight
where AirlineCode in (select AirlineCode from airline
where AirlineName='United Airways'));

6) Write a query to show the first name, gender, and phone number of passengers who have received a travel agency discount greater than 500.

Query >> select First, Gender, Phone from passenger
where PassengerID in (select TPassengerID from travel_agency
where AgencyDiscountCode in (select AgencyDiscountCode from
agency_discount where AgencyDiscountAmount>500));

7) Write a query to calculate the number of passengers whose booking status is "**success**".

Alias Name: **reservation_success_count**

Query >> select count (*) as reservation_success_count from passenger

where PassengerID in (select PassengerID from reservation

where ReservationStatusID in (select ReservationStatusID from

reservation_status where ReservationStatus ='success'));

8) Write a query to calculate the number of passengers who have booked an economy travel class.

Note: use Column name **travelclass** to retrieve the type of Travelclass

Query >> select count (*) as count from passenger where PassengerID in

(select PassengerID from reservation where TravelClassID in

(select TravelClassID from travel_class

where TravelClass='economy'));

9) Write a query to show the name of the airline and its headquarters address for the booking made by the passenger named '**lilly**'

Note: use Column name **First** to retrieve the first name

Query >> select AirlineName, HeadquartersAddress from airline

where AirlineCode in (select AirlineCode from flight

where FlightNumber in (select FlightNumber from reservation

where PassengerID in (select PassengerID from passenger

where First='lilly')));

10) Write a query to show the first name, gender, and phone number of female passengers who have made a reservation for a ticket costing more than 12000

Note: use Column name **First** to retrieve the first name

Query >> select First, Gender, Phone from passenger

where Gender='F' and PassengerID in

(select PassengerID from reservation where cost>12000);

SPRINT-4

4.1 Stored Procedures

Procedures − A procedure is a reusable unit that encapsulates the specific business logic of the application. It is a named block stored as a schema object in the Oracle Database and these are mainly used to perform an action.

Syntax

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
{IS | AS}
BEGIN
  < procedure_body >
END procedure_name;
```

Functions − A function is a named block that returns a value. It is also known as a subroutine or a subprogram. A function is a subprogram that returns a value. The data type of the value is the data type of the function. A function invocation (or call) is an expression, whose data type is that of the function.

Syntax

```
CREATE [OR REPLACE] FUNCTION function_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
RETURN return_datatype
{IS | AS}
BEGIN
  < function_body >
END [function_name];
```

Queries

1)Write a user-defined function to return a count of the total records inserted in the 'passenger' table.

Query >>
```
Create or replace function totalpassenger return number
as
tot int;
begin
select count(*) into tot from passenger;
return tot;
end;
/
```

2)Write a stored procedure in PLSQL to delete a record from a 'passenger' table.

Query >>
```
Create or replace procedure deletepass (delid char)
is
begin
delete from passenger where passengerid = to_number(delid);
end;
/
```

3)Write a User-defined function that retrieves the total number of records from the "travel_class" table where the "TravelClassID" matches 'economy'.

Query >>
```
Create or replace function totalpassenger
return number
as
temp int;
begin
select count(*) into tmp
from travel_class t, reservation r
where r.travelclassid = t.travelclassid and
```

```
travelclass= 'economy';
return tmp;
end;
/
```

4)Write a stored procedure in PLSQL to insert a new record to the 'passenger' table.

Query >>

```
Create or replace procedure insertPASSENGER(pid in number,
f_n in varchar, m_n in varchar, l_n in varchar, gender in char, ph
in number, email in varchar, p_type in varchar)
as
begin
insert
into passenger values(pid,f_n,m_n,l_n,gender,ph,email,p_type);
end;
/
```

5)Write a stored procedure in PLSQL to update the 'reservation' table.

Query >>

```
Create or replace procedure updatecustomer(id int, f_num
varchar)
as
begin
update reservation set flightnumber = f_num
where passengerid=id;
end;
/
```

4 .2 Triggers

Triggers − A trigger is a named PL/SQL unit that is stored in the database and can be invoked repeatedly. Unlike a stored procedure, you can enable and disable a trigger, but you cannot explicitly invoke it.

A trigger is a named database object that encapsulates and defines a set of actions that are to be performed in response to an insert, update, or delete operation against a table.

Triggers are stored programs, which are automatically executed or fired when some event occurs.

Syntax

```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
  Declaration-statements
BEGIN
  Executable-statements
EXCEPTION
  Exception-handling-statements
END;
```

Queries

1)Write a query to insert the name into the pass_info table before inserting the records in the 'passenger' table using triggers and display the inserted name.

Query>> Create or replace trigger PASSENGERNAME

```
Before insert on passenger for each row
Begin
  Insert into pass_info(name) values (:NEW.first);
End;
/
```

2)Write a query to insert the name of the female passenger into the 'pass_info' table before inserting the records in the passenger table using triggers and display the inserted names.

Query>> Create or replace trigger FEMALEPASSENGERLIST

```
Before insert on passenger for each row
Begin
 If :NEW.gender='F' then
    Insert into pass_info(name) values(:NEW.first);
 End if;
End;
/
```

3)Write a query to insert the first 3 passengers first names into the pass_info table before inserting the records in the 'passenger' table using trigger.

Query>>Create or replace trigger FIRSTNAME

```
Before insert on passenger for each row
Declare
  V number;
Begin
   Select count(*) into V from pass_info;
   If V<3 then insert into pass_info(name) values(:NEW.first);
```

```
    End if;
End;
/
```

4) Write a query to insert the first 3 passengers' first names converted to uppercase into the pass_info table before inserting the records in the 'passenger' table using trigger.

Query>>
```
Create or replace trigger FIRSTNAME
Before insert on passenger for each row
Declare
  V number;
Begin
  Select count(*) into V from pass_info;
   If V<3 then insert into pass_info(name) values(upper(:NEW.first));
   End if;
End
/
```

5)Write a query to insert the updated first names and convert them to uppercase into the pass_info table using triggers and display names.

Query>>
```
Create or replace trigger PASSENGERNAME
Before update on passenger for each row
Begin
  Insert into pass_info(name) values(:NEW.first);
End;
/
```

4.3 CURSORS

Cursors − A cursor is a pointer to a result set or the data that results from a query. Cursors let you fetch one or more rows from the database into memory, process them, and then either commit or roll back those changes. A cursor is a PL/SQL construct that allows the user to name the work area and access the stored information in it. PL/SQL has two types of cursors: implicit cursors and explicit cursors.

- Implicit cursors are used for all SQL statements that are not named.
- Explicit cursors are used for queries that return multiple rows.

Syntax

```
DECLARE
CURSOR <cursor_name> IS <SELECT statement^>
<cursor_variable declaration>
BEGIN
OPEN <cursor_name>;
FETCH <cursor_name> INTO <cursor_variable>;
.
.
CLOSE <cursor_name>;
END;
```

Cursor Attributes

Both the Implicit cursor and the explicit cursor have certain attributes that can be accessed. Below are the different cursor attributes and their usage.

%FOUND	It returns the Boolean result 'TRUE' if the most recent fetch operation fetched a record successfully, else it will return FALSE.
%NOTFOUND	This works oppositely to %FOUND it will return 'TRUE' if the most recent fetch operation could not able to fetch any record.
%ISOPEN	It returns the Boolean result 'TRUE' if the given cursor is already opened, else it returns 'FALSE'
%ROWCOUNT	It returns the numerical value. It gives the actual count of records that got affected by the DML activity.

Queries

1) Write a cursor in PLSQL to display all the Passengers list in ascending order.

Query >>
```
declare
   cursor passengerlist is
   select first from passenger
   order by first;
begin
   for i in passengerlist loop
   dbms_output. put_line (i. first);
   end loop;
end;
/
```

2) Write a cursor in PLSQL to display all the Passengers list and reservationdate, Seatnumber, cost information in ascending order.

Query >>
```
declare
   cursor passengerlist is
   select p.first, r.reservationdate, r.seatnumber, r.cost
   from reservation r, passenger p
   where r.passengerid = p.passengerid
   order by first;
 begin
   for i in passengerlist loop
   dbms_output. put_line(i. first ||' '||i.reservationdate ||' '||
   i.seatnumber||i.cost);
   end loop;
 end;
 /
```

3) Write a cursor in PLSQL to display all the Passengers list and travelclass.

Query >>
```
declare
  cursor passengerlist is
  select p.first, t.travelclass from passenger p,travel_class t,
  reservation r where p.passengerid = r.passengerid
  and r.travelclassid = t.travelclassid;
begin
  for i in passengerlist loop
  dbms_output. put_line(i. first ||' '|i.travelclass);
  end loop;
end;
/
```

4) Write a cursor in PLSQL to display all the Passengers list and count how many first names are fetched as well.

Query >>
```
declare
  cursor passengerlist is
  select first from passenger;
  fetch_count number: = 0;
begin
  select count (*) into fetch_count from passenger;
  for i in passengerlist loop
  dbms_output. Put_line('Passenger name||i.first) ;
  end loop;
  dbms_output. put_line('Total rows fetched is ||' '|| fetch_count);
end;
/
```

5) Write a cursor in PLSQL that retrieves and display the names of passengers who have booked a ‘first’ class ticket from a passenger table.

Query >>

```
declare
   cursor passengerlist is
   select p.first from passenger p, reservation r, travel_class t
   where p.passengerid = r.travelclassid and t.travelclassid =
   r.travelclassid and t.travelclass = 'first';
begin
   for i in passengerlist loop
   dbms_output. put_line (i. first);
   end loop;
end;
 /
```

PERFORMANCE TUNING

Query Optimization:

- Analyze the execution plans of your queries using tools like EXPLAIN PLAN or SQL Developer's Query Analyzer. Identify any full table scans or inefficient access paths.
- Optimize the query logic by rewriting complex queries, eliminating unnecessary joins, and using appropriate join types (e.g., INNER JOIN instead of OUTER JOIN).
- Consider using hints or query optimization techniques such as materialized views, partitioning, or parallel execution if applicable.
- Make use of bind variables and bind variable peeking to improve SQL statement reuse.

Indexing Strategies:

- Review the existing indexes and ensure they are being used effectively by the queries. Unused or redundant indexes can impact performance.
- Identify frequently accessed columns in WHERE or JOIN conditions and create indexes on those columns.
- Consider indexing foreign key columns to speed up join operations.
- Regularly monitor and maintain index fragmentation and rebuild or reorganize indexes when necessary.

Log Files:

- Regularly monitor and manage the size of the redo logs and archive logs to prevent excessive growth.
- Configure an appropriate log file size and number of log groups to balance performance and recovery needs.
- Ensure the log files are stored on separate disks or disk groups for better I/O performance.
- Regularly backup and archive log files to a separate location to free up disk space.

Review and Optimize SQL Queries:

- Analyze the execution plans of your queries using tools like EXPLAIN PLAN or query analyzers to identify inefficient access paths or full table scans.

- Ensure that your queries are using appropriate indexes. Create indexes on frequently accessed columns in WHERE or JOIN conditions.
- Rewrite complex queries to simplify them and eliminate unnecessary joins or subqueries.
- Use appropriate join types (e.g., INNER JOIN instead of OUTER JOIN) based on the data requirements.
- Consider using query optimization techniques such as materialized views, partitioning, or parallel execution if applicable.
- Regularly monitor and tune the SQL statements using tools like Oracle SQL Tuning Advisor or other performance monitoring tools.

Optimize Database Schema and Indexing:

- Review and optimize your database schema design. Normalize tables where necessary to reduce redundancy and improve data retrieval efficiency.
- Identify and eliminate unused or redundant indexes, as they can impact performance during data modification operations.
- Regularly analyze and maintain index fragmentation. Rebuild or reorganize indexes to improve performance.
- Consider using index compression to reduce storage requirements and improve query performance.

Optimize Server and Database Configuration:

- Ensure that your server hardware meets the requirements for your database workload. Pay attention to CPU, memory, and disk I/O capabilities.
- Configure appropriate memory settings, such as the buffer cache and shared pool size, to optimize data caching and query performance.
- Adjust database parameters, such as optimizer settings, to fine-tune the query execution plans.
- Monitor and adjust the database block size and redo log configuration to optimize I/O performance.

Regular Database Maintenance:

- Perform regular database maintenance tasks, such as database statistics gathering, index rebuilding, and table reorganization.
- Schedule regular backups and monitor disk space usage to prevent any issues related to log files or datafiles.

PROJECT WORKSPACE

```
SET TIMING ON;

--Creating index
CREATE INDEX QUERY ON AIRLINE(S_ID,GENDER,AGE,CLASS);

--1. Write a SQL query to find the number of female passengers who have made bookings for air tickets.
SELECT COUNT(*) as Female_Passenger_count FROM AIRLINE WHERE GENDER='Female';

--2. Write a sql query to find the total number of passengers who have booked tickets in the business class.
SELECT COUNT(*) AS Passenger_count FROM AIRLINE WHERE CLASS='Business';

--3. Write a sql query to find the number of passengers who have booked tickets in 'eco' class.
SELECT COUNT(*) AS Eco_Passenger FROM AIRLINE WHERE CLASS='Eco';

--4. Write a sql query to find the number of childrens who are all book the tickets childresn are below age 13.
SELECT COUNT(*) AS Kid_count FROM AIRLINE WHERE AGE<13;

--5. Write a sql query to count of passengers who are above the age 16 and have booked tickets in business class.
SELECT COUNT(*) AS Adult_count FROM AIRLINE WHERE AGE>16 AND CLASS='Business';
```

Contribution

Sprints	Contribution
SPRINT-1	**Data Requirements –** Every individual has contributed by preparing their own data requirements document. **ER Diagram –** Diagram was drawn by **Sony, Kiran, Pranay, Prajith, Maha Lakshmi, Sarvani**
SPRINT-2	**Task-1** **Relational Schema with PK and FK –** The diagram was drawn by **Sony, Kiran, Prajith, Pranay** **Task-2** **Identifying 1NF Violations –** These are identified by **Pranay, Sony, Prajith**, and the 1NF document was also prepared by them. **Identifying 2NF Violations –** These are identified by **Maha Lakshmi, Sarvani**, and the 2NF document was also prepared by them. **Identifying 3NF Violations –** These are identified by **Kiran, Pranathi, Sruthik**, and the 3NF document was also prepared by them. **Task-3** **Relational Schema after Normalization –** The diagram was drawn by **Kiran, Sony**
SPRINT-3	**Table Creation –** Sony, Prajith, Pranay, Kiran, Maha Lakshmi, Sarvani, Pranathi, Sruthik **Fetching Records from a single table –** Sony, Prajith, Pranay, Kiran, Maha Lakshmi, Sarvani, Pranathi, Sruthik **Fetching Records from multiple tables using joins-** Sony, Prajith, Pranay, Kiran, Maha Lakshmi, Sarvani, Pranathi, Sruthik **Sub-queries-** Sony, Prajith, Pranay, Kiran, Maha Lakshmi, Sarvani, Pranathi, Sruthik
SPRINT-4	**Procedures and functions-** Sony, Prajith, Pranay, Kiran, Maha Lakshmi, Sarvani, Pranathi, Sruthik **Triggers-** Sony, Prajith, Pranay, Kiran, Maha Lakshmi, Sarvani, Pranathi, Sruthik **Cursors-** Sony, Prajith, Pranay, Kiran, Maha Lakshmi, Sarvani, Pranathi, Sruthik
SPRINT-5	**Project Workspace-** Every individual has contributed by sharing their own optimized queries.