Project: Cruise Ship Database

Table of contents

Executive Summary

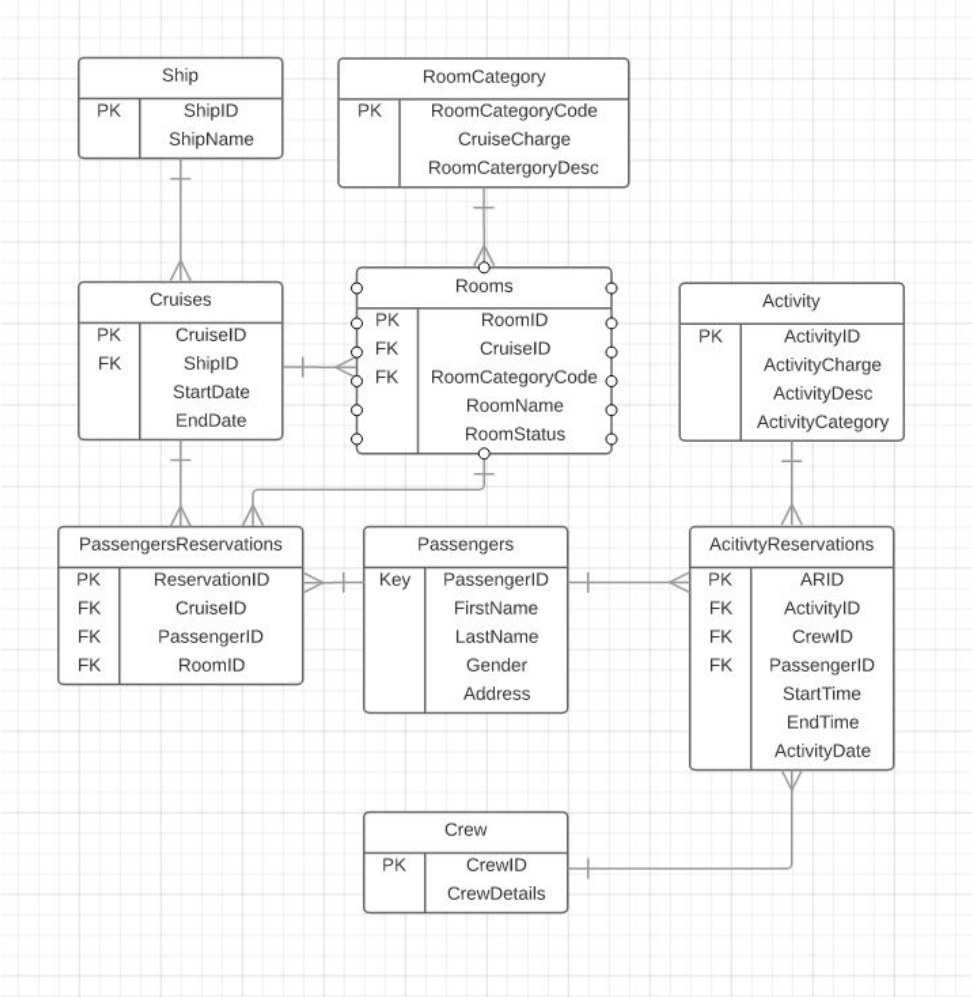
Cruises are often seen as messy, not well-organised and more tiring than anything else made for holidays and visiting. All that because of a lack in our mediums to keep informations that would be required at all time.

This document outlines the structure and entities involved in the design and implementations of a database system of the passengers of a cruise.

The objectives of this database is to test a new gestionale system. The impact of this new system in terms of effectiveness is still difficult to assess due to the lack of use and analysis but the impact should be significant anyway. Gathering the information in one database to make all transactions, clear and simple.

The passengers, their room, their activities and more will be cataloged into the system to benefits responsiveness from the crew to answer the passenger's needs. The objective is to accentuate our competitiveness against our competitors to make the best profit possible.

Entity Relationship Diagram of the Database



Ship information table

```
CREATE TABLE IF NOT EXISTS Ship(
        ShipID SERIAL NOT NULL UNIQUE,
        ShipName VARCHAR(30) NOT NULL,
        PRIMARY KEY (ShipID)
);
INSERT INTO Ship(ShipName) VALUES('The Star Ship');
INSERT INTO Ship(ShipName) VALUES('The Liberty Cruise');
```

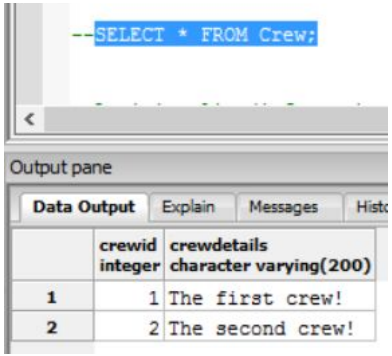
```
SELECT * FROM ship;
```

Data Output | Explain | Messages | History

	shipid integer	shipname character varying(30)
1	1	The Star Ship
2	2	The Liberty Cruise

Crew information table

```
CREATE TABLE IF NOT EXISTS Crew (

        CrewID SERIAL NOT NULL UNIQUE,

        CrewDetails VARCHAR(200),

        PRIMARY KEY(CrewID)

);

INSERT INTO Crew(CrewDetails) VALUES ('The first crew!');

INSERT INTO Crew(CrewDetails) VALUES ('The second crew!');
```



Activity information table

```
CREATE TABLE IF NOT EXISTS Activity (

        ActivityID SERIAL NOT NULL UNIQUE,

        ActivityCategory VARCHAR(100) NOT NULL,

        ActivityCharge NUMERIC(15,2) NOT NULL,

        ActivityDesc VARCHAR(200),

        PRIMARY KEY(ActivityID)

);
```

```
SELECT * FROM Activity;
```

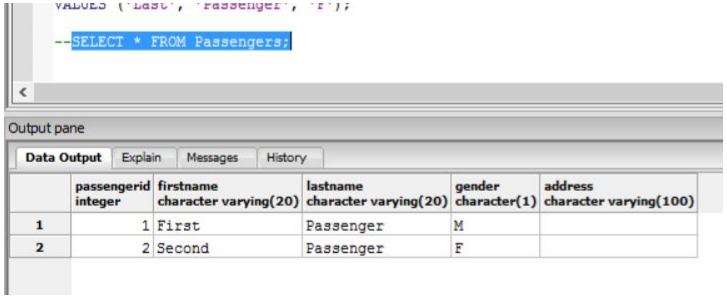
Output pane

Data Output | Explain | Messages | History

	activityid integer	activitycategory character varying(100)	activitycharge numeric(15,2)	activitydesc character varying(200)
1	1	Sports	100.00	Dive into the ocean!
2	2	Sports	499.99	MinGolf on the cruise deck!

Passenger information table

```
CREATE TABLE IF NOT EXISTS Passengers (
        PassengerID SERIAL NOT NULL UNIQUE,
        FirstName VARCHAR(20) NOT NULL,
        LastName VARCHAR(20) NOT NULL,
        Gender CHAR(1) NOT NULL,
        Address VARCHAR(100),
        PRIMARY KEY (PassengerID)
);
```



	passengerid integer	firstname character varying(20)	lastname character varying(20)	gender character(1)	address character varying(100)
1	1	First	Passenger	M	
2	2	Second	Passenger	F	

Cruise information table

```
CREATE TABLE IF NOT EXISTS Cruises (

        CruiseID SERIAL NOT NULL UNIQUE,

        ShipID INTEGER NOT NULL,

        StartDate DATE NOT NULL,

        EndDate DATE NOT NULL,

        PRIMARY KEY (CruiseID),

        FOREIGN KEY (ShipID) REFERENCES Ship(ShipID)

);
```

```
--SELECT * FROM Cruises;
```

Output pane

Data Output | Explain | Messages | History

	cruiseid integer	shipid integer	startdate date	enddate date
1	1	1	2016-10-10	2016-10-20
2	2	1	2016-10-11	2016-11-19

Room category table (category determine the price)

```
CREATE TABLE IF NOT EXISTS RoomCategory (

        RoomCategoryCode CHAR(5) NOT NULL UNIQUE,

        CruiseCharge NUMERIC(15,2) NOT NULL,

        RoomCategoryDesc VARCHAR(200),

        PRIMARY KEY (RoomCategoryCode)

);
```

```
--SELECT * FROM RoomCategory;
```

Output pane

Data Output | Explain | Messages | History

	roomcategorycode character(5)	cruisecharge numeric(15,2)	roomcategorydesc character varying(200)
1	CABIN	2000.00	
2	STARB	2555.99	

Room status information table

Occupied (O) or Vacant (V)

```
CREATE TABLE IF NOT EXISTS Rooms (

        RoomID SERIAL NOT NULL UNIQUE,

        CruiseID INTEGER NOT NULL,

        RoomCategoryCode CHAR(5) NOT NULL,

        RoomName VARCHAR(100) NOT NULL,

        RoomStatus CHAR(1) NOT NULL,

        PRIMARY KEY (RoomID),

        FOREIGN KEY (CruiseID) REFERENCES Cruises(CruiseID),

        FOREIGN KEY (RoomCategoryCode) REFERENCES RoomCategory(RoomCategoryCode)

);
```

```
--SELECT * FROM Rooms;
```

Output pane

Data Output | Explain | Messages | History

	roomid integer	cruiseid integer	roomcategorycode character(5)	roomname character varying(100)	roomstatus character(1)
1	1	1	CABIN	The Cabin Room	V
2	2	1	STARB	The StarBoard Room	O
3	3	2	CABIN	The Cabin Room	V
4	4	2	STARB	The StarBoard Room	O

Activity reservation table

```
CREATE TABLE IF NOT EXISTS ActivityReservations (
        ActivityReservationID SERIAL NOT NULL UNIQUE,
        ActivityID INTEGER NOT NULL,
        CrewID INTEGER NOT NULL,
        PassengerID INTEGER NOT NULL,
        ActivityDate DATE NOT NULL,
        StartTime TIME NOT NULL,
        EndTime TIME NOT NULL,
        OtherDetails VARCHAR(200),
        PRIMARY KEY (ActivityReservationID),
        FOREIGN KEY(ActivityID) REFERENCES Activity(ActivityID),
        FOREIGN KEY(PassengerID) REFERENCES Passengers(PassengerID),
        FOREIGN KEY(CrewID) REFERENCES Crew(CrewID)
);
```

```
--SELECT * FROM ActivityReservations;
```

Output pane

Data Output | Explain | Messages | History

	activityreservationid integer	activityid integer	crewid integer	passengerid integer	activitydate date	starttime time without time zone	endtime time without time zone	otherdetails character varying(200)
1	1	1	1	1	2016-10-10	11:00:00	15:00:00	
2	2	2	2	2	2016-10-10	11:00:00	15:00:00	

Activity reservation table

```
CREATE TABLE IF NOT EXISTS ActivityReservations (

        ActivityReservationID SERIAL NOT NULL UNIQUE,

        ActivityID INTEGER NOT NULL,

        CrewID INTEGER NOT NULL,

        PassengerID INTEGER NOT NULL,

        ActivityDate DATE NOT NULL,

        StartTime TIME NOT NULL,

        EndTime TIME NOT NULL,

        OtherDetails VARCHAR(200),

        PRIMARY KEY (ActivityReservationID),

        FOREIGN KEY(ActivityID) REFERENCES Activity(ActivityID),

        FOREIGN KEY(PassengerID) REFERENCES Passengers(PassengerID),

        FOREIGN KEY(CrewID) REFERENCES Crew(CrewID)

);
```

```
--SELECT * FROM ActivityReservations;
```

Output pane

Data Output | Explain | Messages | History

	activityreservationid integer	activityid integer	crewid integer	passengerid integer	activitydate date	starttime time without time zone	endtime time without time zone	otherdetails character varying(200)
1	1	1	1	1	2016-10-10	11:00:00	15:00:00	
2	2	2	2	2	2016-10-10	11:00:00	15:00:00	

View of the Passenger List

CREATE VIEW CruisePassengers AS

SELECT pr.ReservationID, sh.ShipName, p.FirstName, p.LastName, r.RoomID, r.RoomName

FROM Ship sh, Cruises c, Rooms r, PassengersReservations pr, Passengers p

WHERE sh.ShipID = c.ShipID

AND c.CruiseID = pr.CruiseID

AND r.RoomID = pr.RoomID

AND p.PassengerID = pr.PassengerID;

Data Output | Explain | Messages | History

	reservationid integer	shipname character varying(30)	firstname character varying(20)	lastname character varying(20)	roomid integer	roomname character varying(100)
1	1	The Star Ship	First	Passenger	4	The StarBoard Room
2	2	The Star Ship	Second	Passenger	2	The StarBoard Room

View Passenger ActivityList

CREATE VIEW PassengerActivities AS

SELECT ar.ActivityReservationID, p.PassengerID, p.FirstName, p.LastName, a.ActivityID, a.activityDesc, ar.ActivityDate, ar.StartTime, ar.EndTime, a.activityCharge

FROM ActivityReservations ar, Activity a, Passengers p

WHERE ar.ActivityID = a.ActivityID

AND ar.PassengerID = p.PassengerID

ORDER BY p.PassengerID;

Output pane

Data Output | Explain | Messages | History

	activityreservationid integer	passengerid integer	firstname character varying(20)	lastname character varying(20)	activityid integer	activitydesc character varying(200)	activitydate date	starttime time without time zone	endtime time without time zone	activitycharge numeric(15,2)
1	1	1	First	Passenger	1	Dive into the ocean!	2016-10-10	11:00:00	15:00:00	100.00
2	2	2	Second	Passenger	2	MinGolf on the cruise deck!	2016-10-10	11:00:00	15:00:00	499.99

Report Room charge & Activity charge for a passenger

```
CREATE VIEW PassengerTotal AS

SELECT p.PassengerID, p.FirstName, p.LastName, calculateRoomCharge(pr.reservationID) AS roomTotal, calculateActivityCharge(pr.passengerID) AS activityTotal, (calculateRoomCharge(pr.reservationID) + calculateActivityCharge(pr.passengerID)) AS total

FROM Passengers p, PassengersReservations pr

WHERE p.passengerID = pr.passengerID;
```

To calculate the total room charge for a reservation.

```
CREATE OR REPLACE FUNCTION calculateRoomCharge(RID int) RETURNS NUMERIC(15,2)

AS $total$

DECLARE

	Day_total INTEGER; --Used to store total reservation days

	totalCharge NUMERIC(15,2); --Used to store the total room charge before returning

BEGIN

	Day_total := (SELECT (c.endDate - c.startDate) AS days

		FROM cruises c, PassengersReservations pr

		WHERE c.CruiseID = pr.CruiseID

		AND pr.ReservationID = RID);
```

```
        SELECT INTO totalCharge (Day_Total * rc.CruiseCharge) AS totalCharge

        FROM PassengersReservations pr, Passengers p, Cruises c, Rooms r, RoomCategory rc

        WHERE pr.CruiseID = c.CruiseID

        AND pr.PassengerID = p.PassengerID

        AND pr.RoomID = r.RoomID

        AND r.RoomCategoryCode = rc.RoomCategoryCode

        AND pr.ReservationID = RID;



        RETURN totalCharge;

END;

$total$

LANGUAGE plpgsql;
```

```
CREATE OR REPLACE FUNCTION calculateActivityCharge(PID int) RETURNS NUMERIC(15,2)

AS $total$

DECLARE

        activityTotal NUMERIC(15,2); --Used to store the total room charge before returning

BEGIN

        SELECT INTO activityTotal (SUM(a.ActivityCharge)) AS totalActivityCharge

        FROM Activity a, ActivityReservations ar

        WHERE a.ActivityID = ar.ActivityID

        AND ar.PassengerID = PID

        GROUP BY ar.PassengerID;

        RETURN activityTotal;

END;

$total$

LANGUAGE plpgsql;
```

To calculate total activity charge for a passenger.

Change the status of the rooms

```
CREATE OR REPLACE FUNCTION new_room_status()
RETURNS trigger AS $$
DECLARE
	Status CHAR(1);
BEGIN
	Status := (SELECT RoomStatus FROM Rooms WHERE Rooms.RoomID = New.RoomID);
	IF(status = 'V') THEN
	UPDATE Rooms SET RoomStatus = 'O'
	WHERE Rooms.RoomID = NEW.RoomID;
	RETURN NEW;
	ELSE RETURN NULL;
	END IF;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER new_room_status
AFTER INSERT ON PassengersReservations
FOR EACH ROW EXECUTE PROCEDURE new_room_status();
```

Update room status to occupied or vacant part 1

```
CREATE OR REPLACE FUNCTION update_room_status()
RETURNS trigger AS $$
DECLARE
	Status CHAR(1);
	End_date DATE := (
		SELECT c.endDate
		FROM PassengersReservations pr, Cruises c
		WHERE c.CruiseID = pr.CruiseID
		AND pr.ReservationID = NEW.ReservationID);
BEGIN
	IF (End_date > now()) THEN
		UPDATE Rooms SET Roomstatus = 'V'
		WHERE Rooms.RoomID = OLD.RoomID;

		UPDATE Rooms SET Roomstatus = 'O'
		WHERE Rooms.RoomID = NEW.RoomID;
```

Update room status to occupied or vacant part 2

```
        ELSE
                UPDATE Rooms SET RoomStatus = 'V'
                WHERE Rooms.RoomID = NEW.RoomID
                OR Rooms.RoomID = OLD.RoomID;
        END IF;
        RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER update_room_status
AFTER UPDATE ON PassengersReservations
FOR EACH ROW EXECUTE PROCEDURE update_room_status();
```

Refresh room status stored procedure

```
CREATE OR REPLACE FUNCTION refresh_room_status()
RETURNS void AS $$
BEGIN
	UPDATE Rooms SET Rooms.RoomStatus = 'V';

	UDPATE Rooms SET RoomStatus = 'O'
	WHERE Rooms.RoomID IN (
		SELECT pr.RoomID
		FROM PassengersReservations pr, Cruises c
		WHERE c.CruiseID = pr.CruiseID
		AND c.EndDate > now()
	);
END;
$$ LANGUAGE plpgsql;
```

Security: Admin, cruise desk, room desk, activity desk

```
CREATE ROLE admin;
GRANT ALL ON ALL TABLES
IN SCHEMA PUBLIC
TO admin;
```

```
CREATE ROLE cruisedesk;
GRANT SELECT, INSERT, UPDATE ON PassengerReservations,
Cruises, Rooms, Ship
TO cruisedesk;
```

```
CREATE ROLE roomdesk;
GRANT SELECT, INSERT, UPDATE ON PassengersReservations,
Passengers, Rooms, RoomCategory
TO roomdesk;
```

```
CREATE ROLE activitydesk;
GRANT SELECT, INSERT, UPDATE ON Crew, Activity,
ActivityReservations
TO activitydesk;
```

Issues and future improvements

This present database is great but actually do not appear as complex as it could be and with the number of passengers growing, a lot of data could be misused.

The few problems of this database is that in only classify for the trip but if the customer wants to cruise again with us he will have to fill up forms with the same information over and over again. We also don't know if they came for a particular reason and if we ever forget to wish passengers the best time possible for a birthday or a honeymoon we will be doomed!

We'll take a step back to make different database based on the cruiseship and maybe become a business that no CruiseShip company could avoid to have successful days.

I would think to consider having my database management teacher on this cruise and get him everything all included so this database would prove itself almost worthless but he is awesome!