Final Destination Travel Agency

Final Destination Travel Agency

Final Project

Professor Alan Labouseur

CMPT 308 Database Design

Marist College

Dan Blossom

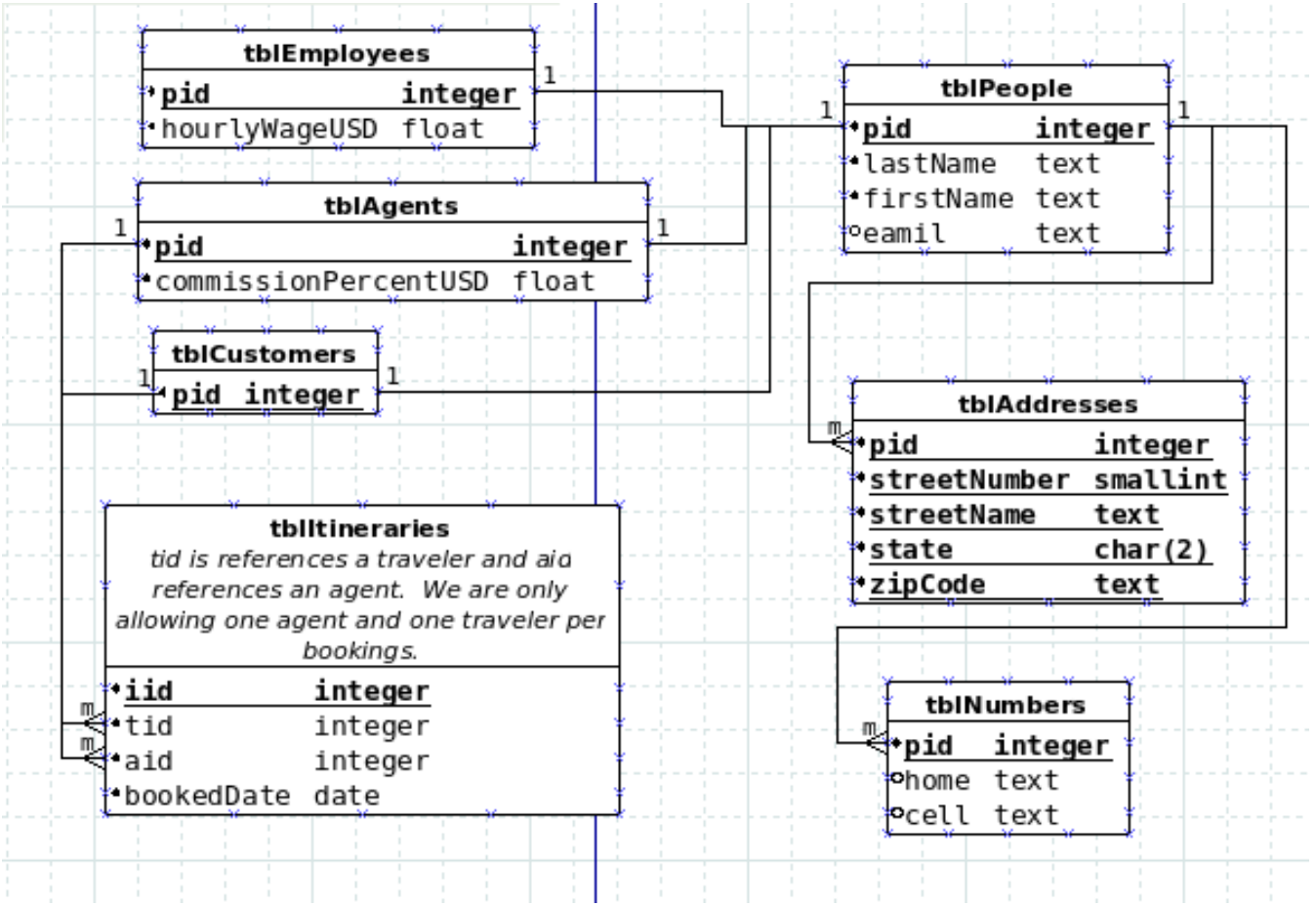
Table of Contents

Executive Summary

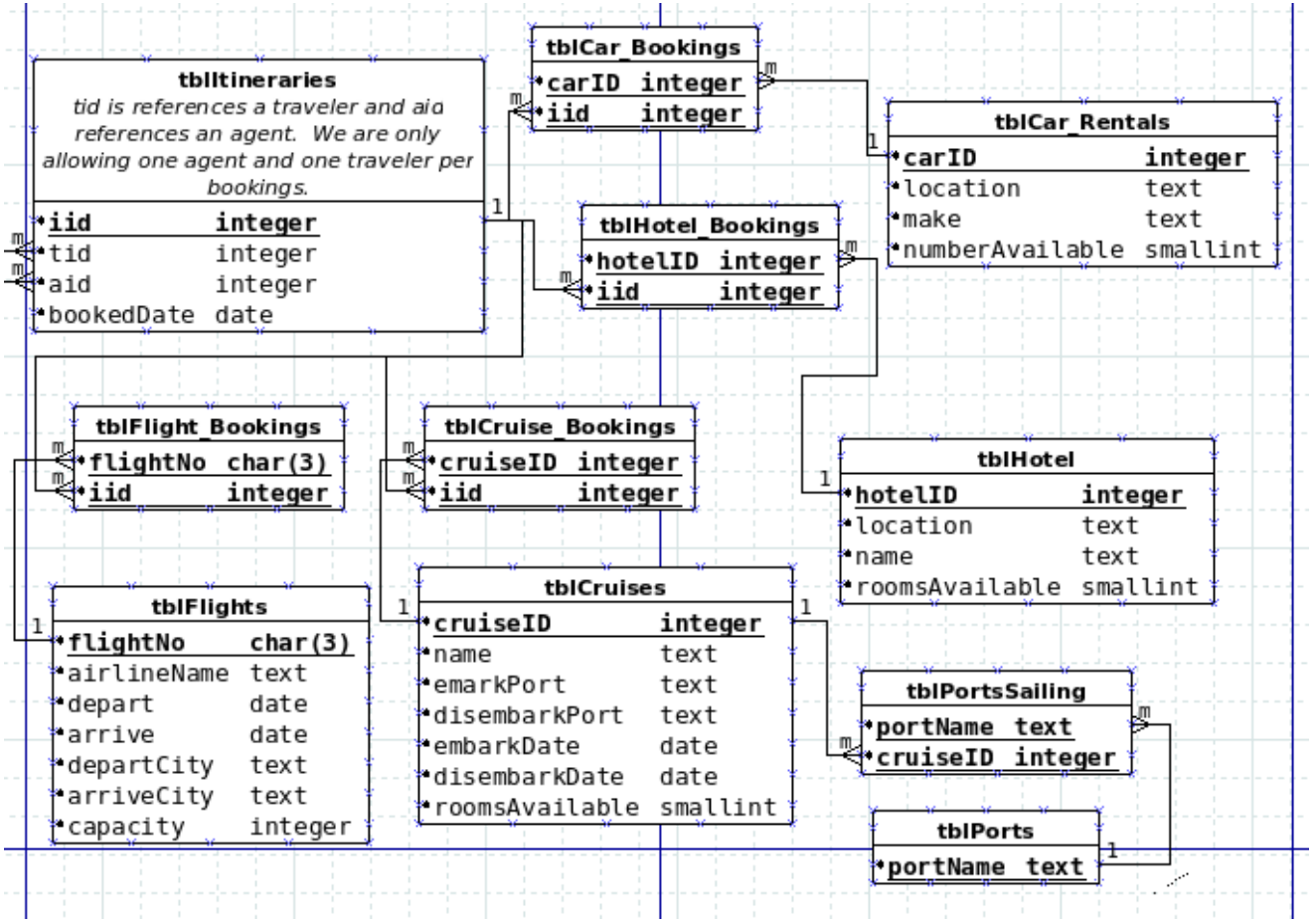
With millions of travelers a year traveling all over the world the Final Destination Travel Agency needs a robust database that will serve as a centralized location to store all data. This data combined with various queries will provide information to various persons of the travel agency. Not only does this database need to meet the needs to today's traveler but needs to be expandable for tomorrow's.

The following slides of this document will outline the basic requirements needed by the Travel Agency. It will outline the E-R diagram of how the database is structured. It will further go into detail of all the tables, their functional dependencies, the statements used to create them and some sample data. Further the following slides will go into detail about about various views, stored procedures, and queries (reports) supported. A brief slide on Security explaining some user types. Finally known issues with solutions to alleviate them and some enhancements to consider for the next release.

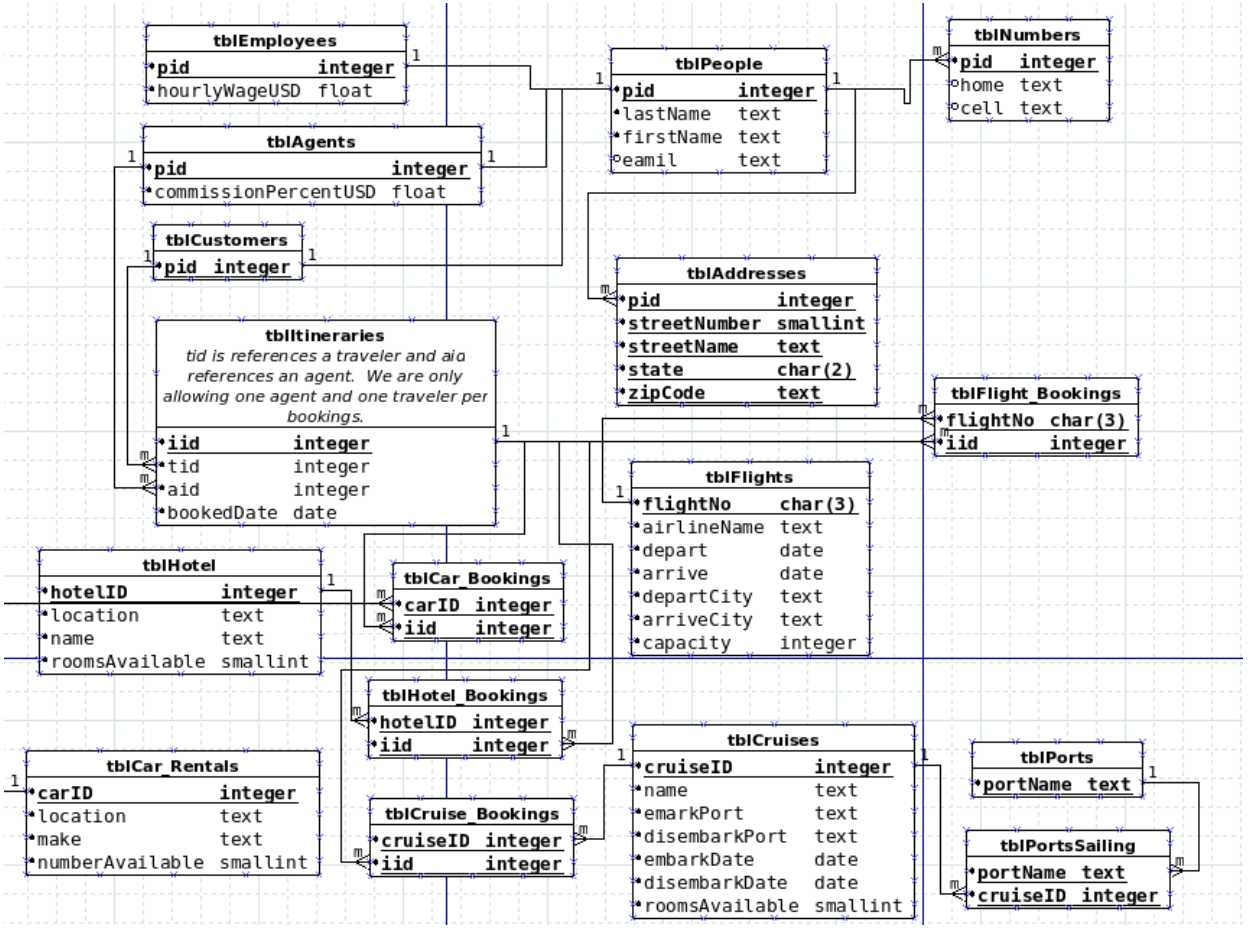
E-R Diagram


tblEmployees
pid integer
hourlyWageUSD float
tblAgents
pid integer
commissionPercentUSD float
tblCustomers
pid integer
tblPeople
pid integer
lastName text
firstName text
eamil text
tblAddresses
pid integer
streetNumber smallint
streetName text
state char(2)
zipCode text
tblNumbers
pid integer
home text
cell text
tblItineraries
tid is references a traveler and aid references an agent. We are only allowing one agent and one traveler per bookings.
iid integer
tid integer
aid integer
bookedDate date

E-R Diagram


tblItineraries
tid is references a traveler and aid references an agent. We are only allowing one agent and one traveler per bookings.
iid integer
tid integer
aid integer
bookedDate date
tblCar_Bookings
carID integer
iid integer
tblCar_Rentals
carID integer
location text
make text
numberAvailable smallint
tblHotel_Bookings
hotelID integer
iid integer
tblFlight_Bookings
flightNo char(3)
iid integer
tblCruise_Bookings
cruiseID integer
iid integer
tblHotel
hotelID integer
location text
name text
roomsAvailable smallint
tblFlights
flightNo char(3)
airlineName text
depart date
arrive date
departCity text
arriveCity text
capacity integer
tblCruises
cruiseID integer
name text
emarkPort text
disembarkPort text
embarkDate date
disembarkDate date
roomsAvailable smallint
tblPortsSailing
portName text
cruiseID integer
tblPorts
portName text

E-R Diagram


tblEmployees
pid integer
hourlyWageUSD float
tblPeople
pid integer
lastName text
firstName text
eamil text
tblNumbers
pid integer
home text
cell text
tblAgents
pid integer
commissionPercentUSD float
tblCustomers
pid integer
tblAddresses
pid integer
streetNumber smallint
streetName text
state char(2)
zipCode text
tblItineraries
tid is references a traveler and aid references an agent. We are only allowing one agent and one traveler per bookings.
iid integer
tid integer
aid integer
bookedDate date
tblFlight_Bookings
flightNo char(3)
iid integer
tblFlights
flightNo char(3)
airlineName text
depart date
arrive date
departCity text
arriveCity text
capacity integer
tblHotel
hotelID integer
location text
name text
roomsAvailable smallint
tblCar_Bookings
carID integer
iid integer
tblHotel_Bookings
hotelID integer
iid integer
tblCruises
cruiseID integer
name text
emarkPort text
disembarkPort text
embarkDate date
disembarkDate date
roomsAvailable smallint
tblPorts
portName text
tblCar_Rentals
carID integer
location text
make text
numberAvailable smallint
tblCruise_Bookings
cruiseID integer
iid integer
tblPortsSailing
portName text
cruiseID integer

People Table

The people table stores information about all people associated with the agency from travelers, agents and salary employees.

Functional Dependency: pid → lastName, firstName, email

```
-- people table
drop table if exists people
create table if not exists people(
  pid       integer not null,
  lastName  text    not null,
  firstName text    not null,
  email     text,
  primary key(pid)
);
```

	pid integer	lastname text	firstname text	email text
1	1	blossom	daniel	yeahright@nohoo.com
2	2	bond	james	jbond@gmail.com
3	3	rogan	seth	seth@mail.com
4	4	doe	jane	doe@yahoomail.com
5	5	griswald	clark	sparky@xmasvaca.com
6	6	rielly	tim	rielly@hotmail.com
7	7	savage	fred	fred@gmail.com
8	8	glass	david	galss@hotmail.com
9	9	west	susan	suewest@hotmail.com
10	10	east	jill	east@direction.com
11	11	pilly	jess	pilly@popper.com

Customers Table

The customers table contains all the people in the database that are travelers of the agency.

Functional Dependency: pid →

```
-- customers table
drop table if exists customers
create table if not exists customers(
  pid  integer not null references people(pid),
  primary key(pid)
);
```

	pid integer
1	1
2	3
3	4
4	5
5	6
6	7

Agents Table

The agents table stores all data about the agents who sell travel, at the moment just their discount.

Functional Dependency: pid → commissionPercentUSD

```
-- agents table
drop table if exists agents
create table if not exists agents(
  pid integer not null references people(pid),
  commissionPercentUSD float not null,
  primary key(pid)
);
```

	pid integer	commissionpercentusd double precision
1	2	0.25
2	8	0.15
3	9	0.1
4	10	0.25

Employees Table

Employees who do not sell travel and do not work by commission are stored here. An example employee type could be Admin for the office or janitor.

Functional Dependency: pid → hourlyWageUSD

```
-- employees table
drop table if exists employees
create table if not exists employees(
  pid integer not null references people(pid),
  hourlyWageUSD float not null,
  primary key(pid)
);
```

	pid integer	hourlywageusd double precision
1	11	23.55

Car Rental & Car BookingTables

The car rental table keeps track of all cars at various locations. A trigger combined with a function will decrement the amount available column when a car is added to the car_booked table. This will ensure no over bookings because we all know all good companies never over book ;).

Functional Dependencies Car_Rental: carID → location, make, numberAvailable

Functional Dependencies Car_Bookings carID, iid →

```
-- car rental table
drop table if exists car_rentals
create table if not exists car_rentals(
  carID integer not null,
  location text not null,
  make     text not null,
  numberAvailable smallint not null,
  primary key(carID)
);
```

```
-- car reservation details table
drop table if exists car_bookings
create table if not exists car_bookings(
  carID integer not null references car_rentals(carID),
  iid integer not null references itineraries(iid),
  primary key(carID, iid)
);
```

	carid integer	location text	make text	numberavailable smallint
1	1	new york	honda	9
2	2	new jersey	nissan	23
3	3	florida	toyota	22
4	4	washington	ford	5
5	5	utah	gmc	0
6	6	utah	honda	50

	carid integer	iid integer
1	1	1
2	2	2
3	3	3
4	4	4

Hotel & Hotel Booking Tables

The hotel table stores information about a hotel, the hotel bookings table links a hotel stay to an itinerary. As with the car rental table, the rooms available will be automatically decremented when a booking is added.

Functional Dependencies Hotel: hotelID → location, name, roomsAvailable

Functional Dependencies Hotel_Bookings: hotelID, iid →

```
-- hotel table
drop table if exists hotels
create table if not exists hotels(
  hotelID integer not null,
  location text not null,
  name text not null,
  roomsAvailable smallint not null,
  primary key(hotelID)
);
```

	hotelid integer	location text	name text	roomsavailable smallint
1	5	new york	hilton	32
2	3	new york	bedbug	4
3	2	new jersey	super 8	18
4	4	florida	roach	0
5	1	new jersey	hilton	97

```
-- hotel reservation details table
drop table if exists hotel_bookings
create table if not exists hotel_bookings(
  hotelID integer not null references hotels(hotelID),
  iid integer not null references itineraries(iid),
  primary key(hotelID, iid)
);
```

	hotelid integer	iid integer
1	1	1
2	5	2
3	3	3
4	2	4
5	2	5
6	4	6
7	1	8
8	1	11

Cruise & Cruise Bookings Tables

The cruise and cruise bookings table is the same as hotel and car rental. Data about the cruise is stored in the cruise table and as a booking is added a room is taken out of the system. There is also a separate tables about ports since a cruise stops at many ports. That will be shown in future slides.

Functional Dependencies (Cruise Table): cruiseID → name, embark, disembark, embarkDate, disembarkDate, roomsAvailable

Functional Dependencies (Cruise Bookings): cruiseID, iid →

```
-- cruise table
drop table if exists cruises
create table if not exists cruises(
  cruiseID integer not null,
  name text not null,
  embark text not null references ports(portName),
  disembark text not null references ports(portName),
  embarkDate date not null,
  disembarkDate date not null,
  roomsAvailable smallint not null,
  primary key(cruiseID)
);
```

```
-- cruise reservation details table
drop table if exists cruise_bookings
create table if not exists cruise_bookings(
  cruiseID integer not null references cruises(cruiseID),
  iid integer not null references itineraries(iid),
  primary key(cruiseID, iid)
);
```

Cruise and Cruse Bookings Tables

	cruiseid integer	name text	embark text	disembark text	embarkdate date	disembarkdate date	roomsavailable smallint
1	1	titanic	belfast	new york	1912-04-01	2013-01-01	50
2	2	britanica	bayonne	new york	2012-12-25	2013-01-01	30
3	4	triumph	mexico	florida	2014-08-04	2014-08-10	100
4	3	concordia	bayonne	bayonne	2014-05-31	2014-06-05	48
5	5	oasis	florida	florida	2015-10-01	2015-10-30	232

	cruiseid integer	iid integer
1	3	10
2	3	11
3	5	4
4	5	2

Cruise Ports Of Call

Since a cruise ship can stop at multiple ports during one sailing, a separate table is created to handle that.

Functional Dependencies (Ports): portName →

Functional Dependencies (Ports Of Call): portName, cruiseID →

```
-- ports of call table
drop table if exists ports
create table if not exists ports(
  portName text not null,
  primary key(portName)
);

-- ports sailing
drop table if exists ports_sailing
create table if not exists ports_sailing(
  portName text not null references ports(portName),
  cruiseID integer not null references cruises(cruiseID),
  primary key(cruiseID, portName)
);
```

	portname text	cruiseid integer
1	middle of ocean	1
2	bermuda	2
3	florida	2
4	new york	3
5	belfast	4
6	bermuda	5
7	mexico	5

	portname text
1	belfast
2	middle of ocean
3	new york
4	mexico
5	bermuda
6	florida
7	bayonne

Flights Table

Flights table works similar to cars, hotels, and cruises however it does check to ensure the departure city and arrival city are different. It also ensures the departure date is less than or equal to the arrival date.

Functional Dependencies: flightNo → airlineName, depart, arrive, departCity, arriveCity, capacity
Check Constraint: departCity != arriveCity and depart <= arrive

```
-- flights table
drop table if exists flights
create table if not exists flights(
  flightNo char(3) not null,
  airlineName text not null,
  depart date not null,
  arrive date not null,
  departCity text not null,
  arriveCity text not null,
  -- ensure we are going someplace
  -- unless the plane crashes during take-off
  check(departCity != arriveCity),
  -- unless we have Doc Browns Delorean
  check(depart <= arrive),
  capacity integer not null,
  primary key(flightNo)
);
```

	flightno character(3)	airlinename text	depart date	arrive date	departcity text	arrivecity text	capacity integer
1	02	contsuckal	2014-01-01	2015-01-02	new jersey	vegas	2
2	04	nowhere	1999-05-22	1999-05-22	new york	la	10
3	01	smashem	2012-01-01	2012-01-02	new york	la	23
4	03	jetblue	2010-03-01	2010-03-02	virginia	atlanta	99
5	06	smashem	2012-01-01	2012-01-02	vegas	new jersey	7
6	05	smashem	2013-10-12	2013-10-12	texas	vegas	50

Flight Bookings

The flight bookings table works similar to the other bookings table, linking flights to itineraries.

Functional Dependencies: flightNo, iid →

```
-- flight reservation details table
drop table if exists flight_bookings
create table if not exists flight_bookings(
  flightNo char(3) not null references flights(flightNo),
  iid integer not null references itineraries(iid),
  primary key(flightNo, iid)
);
```

	flightno character(3)	iid integer
1	01	1
2	01	2
3	03	3
4	06	4
5	06	9
6	05	10

Itineraries Table

The itineraries table brings everything together. It links a customer to various booking types defined in previous slides as flight, hotel, cruise and car rentals. It also links an agent to an itinerary but not to a traveler. We do not want to lock one agent to one traveler, as while an agent wants repeat travelers that might not be the case. It also locks on traveler to an itinerary.

Functional Dependencies: iid → tid, aid, bookedDate

```
-- itinerary table
drop table if exists itineraries
create table if not exists itineraries(
  iid integer not null,
  -- traveler (what if there is more than one traveler)?
  -- we would want to know about "groups" or "families" traveling together
  tid integer not null references customers(pid),
  -- agent... one agent per travel intinerary -- no sharing money
  aid integer not null references agents(pid),
  bookedDate date not null,
  primary key(iid)
);
```

	integer	integer	integer	date
1	1	1	2	2013-04-01
2	2	4	2	2009-04-01
3	3	4	8	2014-07-04
4	4	5	8	2010-08-11
5	5	5	8	2012-08-11
6	6	6	10	2012-09-23
7	7	6	10	2013-10-01
8	8	6	10	2013-12-25
9	9	6	2	2014-07-04
10	10	7	2	2014-01-22
11	11	7	2	2014-01-12

Views:
Future, previous and today Flights

The below views show all previous, future and today views

```
create view Future_Flights As
select f1.*, lastName, firstName
  from flight_bookings f,
       itineraries i,
       customers c,
       people p,
       flights f1
where  i.iid = f.iid
  and  i.tid = c.pid
  and  c.pid = p.pid
  and  f1.flightNo = f.flightNo
  and  f1.depart > current_date
```

```
create view Previous_Flights As
select f1.*, lastName, firstName
  from flight_bookings f,
       itineraries i,
       customers c,
       people p,
       flights f1
where  i.iid = f.iid
  and  i.tid = c.pid
  and  c.pid = p.pid
  and  f1.flightNo = f.flightNo
  and  f1.depart < current_date
```

```
create view Today_Flights As
select f1.*, lastName, firstName
  from flight_bookings f,
       itineraries i,
       customers c,
       people p,
       flights f1
where  i.iid = f.iid
  and  i.tid = c.pid
  and  c.pid = p.pid
  and  f1.flightNo = f.flightNo
  and  f1.depart = current_date
```

Views: Previous Flights Sample

- Below is a sample of previous flights returned from:

  SELECT * FROM Previous_Flights

flightno character(3)	airlinename text	depart date	arrive date	departcity text	arrivecity text	capacity integer	lastname text	firstname text
01	smashem	2012-	2012	new yor	la	23	blosso	daniel
01	smashem	2012-	2012	new yor	la	23	doe	jane
03	jetblue	2010-	2010	virgini	atlant	99	doe	jane
06	smashem	2012-	2012	vegas	new je	7	griswa	clark
06	smashem	2012-	2012	vegas	new je	7	rielly	tim

View - Agents To Customers

Agents to customers view shows who an agents customer might have been. Useful for an agent to send mailers to get repeat customers also shows their travel date.

```
create view agents_customers as
select pc.lastname as "customer_lastname",
       pc.firstname as "customer_firstname",
       pa.lastname as "agent_lastname",
       pa.firstname As "agent_firstname",
       (i.bookeddate) As "Departure"
from itineraries i, agents a, customers c, people pc, people pa
where i.tid = c.pid
  and i.aid = a.pid
  and c.pid = pc.pid
  and a.pid = pa.pid
```

```
select * from agents_customers
where agent_lastname = 'bond'
```

customer_lastname text	customer_firstname text	agent_lastname text	agent_firstname text	Departure date
blossom	daniel	bond	james	2013-04
doe	jane	bond	james	2009-04
rielly	tim	bond	james	2014-07
savage	fred	bond	james	2014-01
savage	fred	bond	james	2014-01

View – Customer Travel Plans

Useful to anyone who needs to see information about the customers travel plans, this is an overview and not trip details.

```
create view customer_ininerary as
select i.iid As "Trip Number",
       p.lastname,
       p.firstname,
       coalesce(cruise.cruiseID, null) As "Cruise #" ,
       coalesce(h.hotelID, null) As "Hotel #",
       coalesce(car.carID, null) As "Car #",
       coalesce(f.flightNo, null) As "Flight #"
from itineraries i left outer join customers c on (i.tid = c.pid)
                   left outer join people p on (c.pid = p.pid)
                   left outer join flight_bookings f on (i.iid = f.iid)
                   left outer join cruise_bookings cruise on (i.iid = cruise.iid)
                   left outer join hotel_bookings h on (i.iid = h.iid)
                   left outer join car_bookings car on (i.iid = car.iid)
```

```
select * from customer_ininerary
where lastname = 'savage'
```

Trip Number integer	lastname text	firstname text	Cruise # integer	Hotel # integer	Car # integer	Flight # character(3)
11	savage	fred	3	1		
10	savage	fred	3			05

View Guest Hotel Details

This view shows the details of a guest staying at a hotel both present and past.

```
create view hotel_guest_details as
select h1.name as "hotel name",
       h1.location as "hotel location",
       p.lastname as "guest last name",
       p.firstname as "guest first name",
       pa.lastname as "booking agent last name",
       pa.firstname as "booking agent first name"
from hotel_bookings h, itineraries i, customers c, people p, hotels h1, agents a, people pa
where h.iid = i.iid
  and i.tid = c.pid
  and c.pid = p.pid
  and h.hotelID = h1.hotelID
  and i.aid = a.pid
  and a.pid = pa.pid
```

hotel name text	hotel location text	guest last name text	guest first name text	booking agent last name text	booking agent first name text
hilton	new jerse	blossom	daniel	bond	james
hilton	new york	doe	jane	bond	james
bedbug	new york	doe	jane	glass	david
super 8	new jerse	griswald	clark	glass	david
super 8	new jerse	griswald	clark	glass	david
roach	florida	rielly	tim	east	jill
hilton	new jerse	rielly	tim	east	jill
hilton	new jerse	savage	fred	bond	james

Stored Procedures

There are 4 stored procedures that work with a trigger upon inserting a new row into one of four tables: flight_bookings, cruise_bookings, hotel_bookings or car_bookings. These procedures will ensure no over bookings by decrementing the space available by one. If there is no space available the booking will not be entered.

```
-- Car Count
drop function updateCars()
create function updateCars() returns trigger as '
declare
      count_down integer;
begin
    select numberAvailable into count_down from car_rentals where carID = new.carID;
    if count_down <= 0 then
    return null;
    end if;
    if count_down > 0 then
       update car_rentals set numberAvailable = count_down - 1 where carID =
new.carID;
    end if;
    return new;
end;
' language plpgsql;

drop trigger update_car_count on car_bookings
create trigger update_car_count
before insert on car_bookings
for each row
execute procedure updateCars();
```

Stored Procedures

```
-- hotels update trigger / function
drop function updateHotels
create function updateHotels() returns trigger as '
declare
    count integer;
begin
    select roomsAvailable into count from hotels where hotelID = new.hotelID;
    if count <= 0 then
    return null;
    end if;
    if count > 0 then
    update hotels set roomsAvailable = count -1 where hotelID = new.hotelID;
    end if;
    return new;
end;
' language plpgsql;

drop trigger update_hotel_rooms on hotel_bookings
create trigger update_hotel_rooms
before insert on hotel_bookings
for each row
execute procedure updateHotels();
```

Stored Procedures

```
-- cruise update / trigger function
drop function updateCruise
create function updateCruise() returns trigger as '
declare
    count integer;
begin
    select roomsAvailable into count from cruises where cruiseID = new.cruiseID;
    if count <= 0 then
    return null;
    end if;
    if count > 0 then
    update cruises set roomsAvailable = count -1 where cruiseID = new.cruiseID;
    end if;
    return new;
end;
' language plpgsql;

drop trigger update_cruise_rooms on cruise_bookings
create trigger update_cruise_rooms
before insert on cruise_bookings
for each row
execute procedure updateCruise();
```

Stored Procedures

```
-- flight availablity
drop function updateFlight
create function updateFlight() returns trigger as '
declare
    count integer;
begin
    select capacity into count from flights where flightNo = new.flightNo;
    if count <= 0 then
    return null;
    end if;
    if count > 0 then
    update flights set capacity = count -1 where flightNo = new.flightNo;
    end if;
    return new;
end;
' language plpgsql;

drop trigger update_flight_avail on flight_bookings
create trigger update_flight_avail
before insert on flight_bookings
for each row
execute procedure updateFlight();
```

Queries

This query is to determine which agent has booked the most reservations.

```
select p.lastname, p.firstname
from people p
where pid in(
            select distinct pid
            from itineraries i, agents a
            where i.aid = a.pid
            and aid in(
                        select aid
                        from(select distinct i.aid, count(i.aid) as "agent_count"
                             from itineraries i
                             group by i.aid
                             order by count(i.aid) desc) sub1
                        limit 1)
             )
```

lastname text	firstname text
bond	james

Queries

Length of a cruise and its embarking and disembarking ports. Calculated from the departure and arrival dates.

```
select name as "ship name",
       embark as "Embark Port",
       disembark as "Disembark Port",
       age(disembarkdate, embarkdate) as "Length"
from cruises
```

ship name text	Embark Port text	Disembark Port text	Length interval
titanic	belfast	new york	100 years 9 mons
britanica	bayonne	new york	7 days
triumph	mexico	florida	6 days
concordia	bayonne	bayonne	5 days
oasis	florida	florida	29 days

Queries

Flights longer than “same day” which would indicate an input error or an overnight flight. As we can see from the sample result set, 3 are overnight, 2 are same day, 1 seems to be very long so an error is found.

```
select airlinename,
       departcity as "departing from",
       arrivecity as "arriving to",
       age(arrive, depart) as "Time"
from flights
```

airlinename text	departing from text	arriving to text	Time interval
contsuckal	new jersey	vegas	1 year 1 day
nowhere	new york	la	00:00:00
smashem	new york	la	1 day
jetblue	virginia	atlanta	1 day
smashem	vegas	new jersey	1 day
smashem	texas	vegas	00:00:00

Queries

Customer contact information. Yearly the agency sends out contact update forms, this information can be useful in determining what critical information could be missing.

```
select p.lastname,
       p.firstname,
       a.streetnumber,
       a.streetname,
       a.state,
       a.zipcode,
       coalesce(n.home, '<N/A>') As "Home Number",
       coalesce(n.cell, '<N/A>') As "Cell Number",
       p.email
from customers c, people p, addresses a, numbers n
where c.pid = p.pid
  and p.pid = a.pid
  and p.pid = n.pid
```

lastname text	firstname text	streetnumber smallint	streetname text	state character(2)	zipcode text	Home Number text	Cell Number text	email text
blossom	daniel	25	fake street	NY	12345	845-555-1212	<N/A>	yeahright@nohoo.com
rogan	seth	222	long road	CT	55555	914-555-3333	<N/A>	seth@mail.com
doe	jane	99	main street	GA	11111	111-555-8888	999-111-1111	doe@yahoomail.com
griswald	clark	112	elm court	VA	22334	845-300-0000	800-800-8000	sparky@xmasvaca.com
rielly	tim	876	simple lane	NJ	07654	555-555-5555	<N/A>	rielly@hotmail.com
savage	fred	44	johns st	NY	22222	<N/A>	000-000-0000	fred@gmail.com

Security

We have identified three kinds of users with a large expandability for many more. Those three are customers to view information about their trip. Agents to view information about a customers trip and book a trip for a customer. Finally a database administrator who has all privileges; it will be the sole purpose of the database administrator to keep the database up to date including various grant and revoke privileges.

```
create role database_admin

grant select, insert, update
on all tables in schema public
to database_admin
```

```
create role customer

grant select
on itineraries
to customer
```

```
create role agent

grant select, insert, update
on cruise_bookings, flight_bookings, hotel_bookings, car_bookings
to agent

grant select
on itineraries, customers, people, addresses, numbers
to agent
```

Known issues

- At present to implement another trip type like resort, two additional tables, a function and trigger will need to be created to support that. A possible solution would be to create a reservation object that would assign ID's to various objects. Creating a generic function that takes in one of these reservation types would also fix the multiple function issue.

- At present a person can only have a home number and cell phone per row. So, if a person has two cell phones, two rows would be created, what if they had no home number – wasted space twice. Another issue with this design is if the person still lived in 1980 and had a fax machine or pager we might for some odd reason want those numbers, this database does not support that. A simple fix would be to create a table for each number type the Final Destination Travel agency wishes to store.

- Once a reservation availability for any type hits zero a null is returned. This is great that it does not allow overbooking but since it does not throw an error the agent will not know the reservation type was not booked. Error handling will need to be added to the present code where “return null” is located.

- Allowing the arrive date for a flight to be greater than the departure date but limiting it to the longest possible flight, it allows for errors; such errors were found with the 1 year flight. A check constraint can be added or possibly make the type more defined, such as adding time to the date.

Future Enhancements

- Create more queries that support details of all reservation types.
- Add purchase amounts so we can keep track of how much a trip cost. Also we will be able to determine how much an agent made per booking.
- Support for groups in an itinerary. As it stands now each member in a trip needs a separate itinerary.
- Create a more detailed reservation system so tables are not needed for future additions.
- Add more details to the various people types such as emergency contact for customer and job type to employees.
- Create a dynamic view built off a function that builds a detailed itinerary for the customer or agent. This will only allow the reservation type and those details that the customer booked. So, if hotel and car reservations were the only booked types then we would not see null for flight and cruise. We would still want to support knowing what a customer did not book, but the customer does not need to see what they did not book … they hopefully know this.