a database design proposal for
UNITED
<table>
<thead>
<tr>
<th>Table/View</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>persons table</td>
<td>5</td>
</tr>
<tr>
<td>employees table</td>
<td>6</td>
</tr>
<tr>
<td>passengers table</td>
<td>7</td>
</tr>
<tr>
<td>aircrafts table</td>
<td>8</td>
</tr>
<tr>
<td>seat_classes table</td>
<td>9</td>
</tr>
<tr>
<td>aircraft_seats table</td>
<td>10</td>
</tr>
<tr>
<td>countries table</td>
<td>11</td>
</tr>
<tr>
<td>airports table</td>
<td>12</td>
</tr>
<tr>
<td>routes table</td>
<td>13</td>
</tr>
<tr>
<td>flight_statuses table</td>
<td>14</td>
</tr>
<tr>
<td>flights table</td>
<td>15</td>
</tr>
<tr>
<td>itineraries table</td>
<td>16</td>
</tr>
<tr>
<td>passengers_on_flights table</td>
<td>17</td>
</tr>
<tr>
<td>triggers</td>
<td>18</td>
</tr>
<tr>
<td>valid_flight_seat_trigger</td>
<td>18-19</td>
</tr>
<tr>
<td>views</td>
<td>22</td>
</tr>
<tr>
<td>flights_arrivals</td>
<td>22</td>
</tr>
<tr>
<td>reports</td>
<td>23</td>
</tr>
<tr>
<td>passenger manifest for flight</td>
<td>23</td>
</tr>
<tr>
<td>reports</td>
<td>24</td>
</tr>
<tr>
<td>passengers with a TSA redress</td>
<td>24</td>
</tr>
<tr>
<td>a single itinerary</td>
<td>25</td>
</tr>
<tr>
<td>security</td>
<td>26</td>
</tr>
<tr>
<td>flight_search role</td>
<td>26</td>
</tr>
<tr>
<td>flight_book role</td>
<td>27</td>
</tr>
<tr>
<td>passenger role</td>
<td>28</td>
</tr>
<tr>
<td>ticket_agent role</td>
<td>29</td>
</tr>
<tr>
<td>implementation notes</td>
<td>30</td>
</tr>
<tr>
<td>known problems</td>
<td>31</td>
</tr>
<tr>
<td>future enhancements</td>
<td>32</td>
</tr>
</tbody>
</table>
With over 144 million passengers a year traveling between 372 worldwide destinations, United Airlines needs a database that with all the data for the passengers. Security is especially critical in this design. The data must be accessible by the appropriate persons in the traveling process, as well as customers. Due to federal regulations, the data must be accurate and consistent. Inconsistent data is unacceptable.

The design focuses on the tables that are necessary to book flights. This data includes: passengers, passenger itineraries, employees, flights, and aircraft seating. The design assumes that United does not have any airline partners, in which they share flights.

An overview of the database will be presented, followed by the details of how each of the database tables will be created. Each table will be followed with a table of sample data. Ideal database user roles will be suggested and their purposes will be explained. To assist in the mission of data integrity, a trigger will be shown and explained. To see how meaningful data can be retrieved, sample reports will be shown. More details about the implementation are provided towards the end of the proposal. Like any design or product, improvements and new features are needed, so they will be explained.

This design was targeted for and tested on PostgreSQL 9.2.4, which was released on April 4th 2013.
**persons table**

Since employees may also be passengers (and not working as a pilot or flight attendant) and passengers may also be employees, their basic information (i.e. name and phone number) is separated into this table.

```sql
CREATE TABLE IF NOT EXISTS persons ( 
    person_id SERIAL NOT NULL UNIQUE, 
    first_name VARCHAR(50) NOT NULL, 
    middle_name VARCHAR(50) NOT NULL, 
    last_name VARCHAR(50) NOT NULL, 
    date_of_birth DATE NOT NULL, 
    gender CHAR(1) NOT NULL, 
    email VARCHAR(256) NOT NULL, 
    phone_primary CHAR(15) NOT NULL, 
    CONSTRAINT valid_gender CHECK (gender = 'M' OR gender = 'F'), 
    PRIMARY KEY (person_id) 
);
```

**functional dependencies**

`person_id` → `first_name`, `middle_name`, `last_name`, `date_of_birth`, `gender`, `email`, `phone_primary`

**sample data**

<table>
<thead>
<tr>
<th>person_id</th>
<th>first_name</th>
<th>middle_name</th>
<th>last_name</th>
<th>date_of_birth</th>
<th>gender</th>
<th>email</th>
<th>phone_primary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Juliet</td>
<td>Maria</td>
<td>Banks</td>
<td>1992-02-10</td>
<td>F</td>
<td><a href="mailto:juliet@gmail.com">juliet@gmail.com</a></td>
<td>808-222-4255</td>
</tr>
<tr>
<td>2</td>
<td>Alexander</td>
<td>William</td>
<td>Arnold</td>
<td>1956-06-09</td>
<td>M</td>
<td><a href="mailto:alexander@yahoo.com">alexander@yahoo.com</a></td>
<td>212-689-9722</td>
</tr>
<tr>
<td>3</td>
<td>Deborah</td>
<td>Josephine</td>
<td>Clark</td>
<td>1990-08-30</td>
<td>F</td>
<td><a href="mailto:deborah@me.com">deborah@me.com</a></td>
<td>845-436-7954</td>
</tr>
<tr>
<td>4</td>
<td>Jeffrey</td>
<td>Domin</td>
<td>Garces</td>
<td>1975-05-18</td>
<td>M</td>
<td><a href="mailto:jeffrey@icloud.com">jeffrey@icloud.com</a></td>
<td>310-514-9791</td>
</tr>
<tr>
<td>5</td>
<td>Eileen</td>
<td>Andin</td>
<td>Anderson</td>
<td>1994-12-15</td>
<td>F</td>
<td><a href="mailto:eileen@hotmail.com">eileen@hotmail.com</a></td>
<td>424-689-7547</td>
</tr>
<tr>
<td>6</td>
<td>Rosalie</td>
<td>Nancy</td>
<td>Morgan</td>
<td>1997-01-23</td>
<td>F</td>
<td><a href="mailto:rosalie@gmail.com">rosalie@gmail.com</a></td>
<td>808-239-4133</td>
</tr>
<tr>
<td>7</td>
<td>Sean</td>
<td>Jacob</td>
<td>Domingo</td>
<td>1955-03-13</td>
<td>M</td>
<td><a href="mailto:sean@icloud.com">sean@icloud.com</a></td>
<td>970-569-1583</td>
</tr>
<tr>
<td>8</td>
<td>Susan</td>
<td>Freeman</td>
<td></td>
<td>1983-10-31</td>
<td>F</td>
<td><a href="mailto:susan@yahoo.com">susan@yahoo.com</a></td>
<td>630-712-6948</td>
</tr>
<tr>
<td>9</td>
<td>Bryan</td>
<td>Colvin</td>
<td>Davis</td>
<td>2005-06-16</td>
<td>M</td>
<td><a href="mailto:bryan@gmail.com">bryan@gmail.com</a></td>
<td>717-378-1987</td>
</tr>
</tbody>
</table>
**employees table**

```sql
CREATE TABLE IF NOT EXISTS employees (  
    person_id INTEGER NOT NULL,  
    hire_date DATE NOT NULL DEFAULT CURRENT_TIMESTAMP,  
    hourly_wage_usd MONEY NOT NULL,  
    PRIMARY KEY (person_id),  
    FOREIGN KEY (person_id) REFERENCES persons(person_id) 
);
```

**functional dependencies**

person_id → hire_date, hourly_wage_usd

**sample data**

<table>
<thead>
<tr>
<th>person_id</th>
<th>hire_date</th>
<th>hourly_wage_usd</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1995-02-21</td>
<td>25.32</td>
</tr>
<tr>
<td>2</td>
<td>1972-06-14</td>
<td>34.25</td>
</tr>
<tr>
<td>8</td>
<td>1991-04-18</td>
<td>30.15</td>
</tr>
<tr>
<td>3</td>
<td>2013-01-31</td>
<td>28.44</td>
</tr>
<tr>
<td>7</td>
<td>1992-11-13</td>
<td>26.75</td>
</tr>
</tbody>
</table>
CREATE TABLE IF NOT EXISTS passengers (  
  person_id    INTEGER  NOT NULL,  
  tsa_redress_num   CHAR(13)   NOT NULL UNIQUE DEFAULT '',  
  known_traveler_num   CHAR(25)   NOT NULL UNIQUE DEFAULT '',  
  needs_special_assistance   BOOLEAN   NOT NULL DEFAULT FALSE,  
  PRIMARY KEY (person_id),  
  FOREIGN KEY (person_id) REFERENCES persons(person_id)  
);
**aircrafts table**
The list of the possible aircraft models used for a particular route.

```
CREATE TABLE IF NOT EXISTS aircrafts (  
  aircraft_id SERIAL NOT NULL,  
  manufacturer VARCHAR(25) NOT NULL,  
  model VARCHAR(15) NOT NULL,  
  PRIMARY KEY (aircraft_id)
);
```

**functional dependencies**

```
aircraft_id → manufacturer, model
```

**sample data**

<table>
<thead>
<tr>
<th>aircraft_id</th>
<th>manufacturer</th>
<th>model</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Boeing</td>
<td>767-300ER</td>
</tr>
<tr>
<td>2</td>
<td>Airbus</td>
<td>A330</td>
</tr>
<tr>
<td>3</td>
<td>Bombardier</td>
<td>CRJ700</td>
</tr>
<tr>
<td>4</td>
<td>Embraer</td>
<td>ERJ145</td>
</tr>
</tbody>
</table>
**seat_classes table**
The list of the possible seat classes for a particular aircraft's seat number.

```sql
CREATE TABLE IF NOT EXISTS seat_classes (
    class_id SERIAL NOT NULL,
    name VARCHAR(25) NOT NULL,
    PRIMARY KEY (class_id)
);
```

**functional dependencies**

```
class_id → name
```

**sample data**

<table>
<thead>
<tr>
<th>class_id</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Economy</td>
</tr>
<tr>
<td>2</td>
<td>Economy Plus</td>
</tr>
<tr>
<td>3</td>
<td>First Class</td>
</tr>
<tr>
<td>4</td>
<td>Business Class</td>
</tr>
</tbody>
</table>

**aircraft_seats table**
The list of seat numbers for a particular model of an aircraft.

```
CREATE TABLE IF NOT EXISTS aircraft_seats (  
    aircraft_id INTEGER NOT NULL,  
    seat_num VARCHAR(3) NOT NULL,  
    class_id INTEGER NOT NULL,  
    PRIMARY KEY (aircraft_id, seat_num),  
    FOREIGN KEY (aircraft_id) REFERENCES aircrafts(aircraft_id),  
    FOREIGN KEY (class_id) REFERENCES seat_classes(class_id)
);
```

**functional dependencies**
`aircraft_id, seat_num` → `class_id`

**sample data**

<table>
<thead>
<tr>
<th>aircraft_id</th>
<th>seat_num</th>
<th>class_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1A</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1B</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1C</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1D</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>14A</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>14B</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>14C</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>14D</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>20A</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>aircraft_id</th>
<th>seat_num</th>
<th>class_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>18A</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>18B</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>18C</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>18D</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>22A</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>22B</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>22C</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>22D</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>31A</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>10A</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>10B</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>10C</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>10D</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>11A</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>11B</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>11C</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>11D</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>28A</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>30A</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>30B</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>30C</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>30D</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>31A</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>31B</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>31C</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>31D</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>32A</td>
<td>1</td>
</tr>
</tbody>
</table>
**countries table**
Contains the list of countries where an airport may be located.

```
CREATE TABLE IF NOT EXISTS countries (
    country_code CHAR(2) NOT NULL UNIQUE,
    name VARCHAR(40) NOT NULL,
    PRIMARY KEY(country_code)
);
```

**functional dependencies**

```
country_code \rightarrow name
```

**sample data**

<table>
<thead>
<tr>
<th>country_code</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>US</td>
<td>United States</td>
</tr>
<tr>
<td>UK</td>
<td>United Kingdom</td>
</tr>
<tr>
<td>CA</td>
<td>Canada</td>
</tr>
<tr>
<td>CN</td>
<td>China</td>
</tr>
<tr>
<td>AT</td>
<td>Austria</td>
</tr>
<tr>
<td>CL</td>
<td>Chile</td>
</tr>
<tr>
<td>CR</td>
<td>Costa Rica</td>
</tr>
<tr>
<td>DE</td>
<td>Germany</td>
</tr>
<tr>
<td>FJ</td>
<td>Fiji</td>
</tr>
<tr>
<td>ES</td>
<td>Spain</td>
</tr>
<tr>
<td>GH</td>
<td>Ghana</td>
</tr>
<tr>
<td>GR</td>
<td>Greece</td>
</tr>
<tr>
<td>MX</td>
<td>Mexico</td>
</tr>
</tbody>
</table>
**airports table**
Contains the list of airports the airline flies to and from. The primary key, *iata_code*, is a unique 3-letter abbreviation for an airport that is assigned by the International Air Transport Association (IATA). Since, the code is unique, it serves as the primary key, rather than creating and using an automatically incremented integer column.

```sql
CREATE TABLE IF NOT EXISTS airports (
    iata_code CHAR(3) NOT NULL UNIQUE,
    airport_name VARCHAR(40) NOT NULL,
    country_code CHAR(2) NOT NULL,
    PRIMARY KEY (iata_code),
    FOREIGN KEY(country_code) REFERENCES countries(country_code)
);
```

**functional dependencies**
*iata_code* → airport_name, country_code

**sample data**

<table>
<thead>
<tr>
<th>iata_code</th>
<th>airport_name</th>
<th>country_code</th>
</tr>
</thead>
<tbody>
<tr>
<td>JFK</td>
<td>John F. Kennedy International Airport</td>
<td>US</td>
</tr>
<tr>
<td>EWR</td>
<td>Newark International Airport</td>
<td>US</td>
</tr>
<tr>
<td>HNL</td>
<td>Honolulu International Airport</td>
<td>US</td>
</tr>
<tr>
<td>DEN</td>
<td>Denver International Airport</td>
<td>US</td>
</tr>
<tr>
<td>LAX</td>
<td>Los Angeles International Airport</td>
<td>US</td>
</tr>
<tr>
<td>ORD</td>
<td>O’Hare International Airport</td>
<td>US</td>
</tr>
<tr>
<td>LHR</td>
<td>London Heathrow Airport</td>
<td>UK</td>
</tr>
</tbody>
</table>
**create table statements**

```sql
CREATE TABLE IF NOT EXISTS routes (
    route_id SERIAL NOT NULL UNIQUE,
    flight_num SMALLINT NOT NULL CHECK (flight_num > 0),
    origin_airport CHAR(3) NOT NULL,
    destination_airport CHAR(3) NOT NULL,
    aircraft_id INTEGER NOT NULL,
    distance_miles SMALLINT NOT NULL CHECK (distance_miles > 0),
    CONSTRAINT diff_orig_dest_airport CHECK(origin_airport != destination_airport),
    PRIMARY KEY (route_id),
    FOREIGN KEY (origin_airport) REFERENCES airports(iata_code),
    FOREIGN KEY (destination_airport) REFERENCES airports(iata_code)
);
```

**functional dependencies**

route_id → flight_num, origin_airport, destination_airport, aircraft_id, distance_miles

**sample data**

<table>
<thead>
<tr>
<th>route_id</th>
<th>flight_num</th>
<th>origin_airport</th>
<th>destination_airport</th>
<th>aircraft_id</th>
<th>distance_miles</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15</td>
<td>EWR</td>
<td>HNL</td>
<td>1</td>
<td>4962</td>
</tr>
<tr>
<td>2</td>
<td>14</td>
<td>HNL</td>
<td>EWR</td>
<td>1</td>
<td>4962</td>
</tr>
<tr>
<td>3</td>
<td>110</td>
<td>EWR</td>
<td>LHR</td>
<td>2</td>
<td>3466</td>
</tr>
<tr>
<td>4</td>
<td>535</td>
<td>JFK</td>
<td>LAX</td>
<td>1</td>
<td>2475</td>
</tr>
<tr>
<td>5</td>
<td>1293</td>
<td>LAX</td>
<td>JFK</td>
<td>1</td>
<td>2475</td>
</tr>
<tr>
<td>6</td>
<td>1025</td>
<td>HNL</td>
<td>LAX</td>
<td>1</td>
<td>2556</td>
</tr>
<tr>
<td>7</td>
<td>1742</td>
<td>LAX</td>
<td>ORD</td>
<td>2</td>
<td>1745</td>
</tr>
<tr>
<td>8</td>
<td>377</td>
<td>ORD</td>
<td>EWR</td>
<td>3</td>
<td>719</td>
</tr>
<tr>
<td>9</td>
<td>1671</td>
<td>DEN</td>
<td>LAX</td>
<td>4</td>
<td>862</td>
</tr>
<tr>
<td>10</td>
<td>383</td>
<td>HNL</td>
<td>DEN</td>
<td>2</td>
<td>3365</td>
</tr>
</tbody>
</table>
**flight_statuses table**
Contains the list of possible statuses for scheduled flights.

```
CREATE TABLE IF NOT EXISTS route_statuses (
    status_id SERIAL NOT NULL UNIQUE,
    name VARCHAR(30) NOT NULL,
    PRIMARY KEY (status_id)
);
```

**functional dependencies**

status_id → name

**sample data**

<table>
<thead>
<tr>
<th>status_id</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>On Time</td>
</tr>
<tr>
<td>2</td>
<td>Delayed</td>
</tr>
<tr>
<td>3</td>
<td>Arrived</td>
</tr>
<tr>
<td>4</td>
<td>Scheduled</td>
</tr>
</tbody>
</table>
flights table
The list of all scheduled flights. A flight is defined as a route with a departure date and time. There cannot be a flight with the same route that departs on the same date and time. The base price (in USD) is the minimum cost for a single passenger traveling on the route. This price can vary depending on the time of year (i.e. February vs. Christmas Eve).

CREATE TABLE IF NOT EXISTS flights (  
  route_id INTEGER NOT NULL,  
  depart_timestamp TIMESTAMP WITH TIME ZONE NOT NULL,  
  arrive_timestamp TIMESTAMP WITH TIME ZONE NOT NULL,  
  base_price_usd MONEY NOT NULL CHECK (base_price_usd > 0.0::text::money),  
  status_id INTEGER NOT NULL,  
  PRIMARY KEY (route_id, depart_timestamp),  
  FOREIGN KEY (route_id) REFERENCES routes(route_id),  
  FOREIGN KEY (status_id) REFERENCES route_statuses(status_id)  
);

functional dependencies
route_id, depart_timestamp → arrive_timestamp, base_price_usd, status_id

sample data

<table>
<thead>
<tr>
<th>route_id</th>
<th>depart_timestamp</th>
<th>arrive_timestamp</th>
<th>base_price_usd</th>
<th>status_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2013-05-18 13:18:00-04</td>
<td>2013-05-18 23:23:00-04</td>
<td>525.36</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>2014-05-18 13:18:00-04</td>
<td>2013-05-18 23:23:00-04</td>
<td>525.36</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>2014-05-28 02:35:00-04</td>
<td>2013-05-28 11:40:00-04</td>
<td>851.49</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>2013-07-17 12:05:00-04</td>
<td>2013-07-17 17:36:00-04</td>
<td>970.67</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>2013-07-30 04:00:00-04</td>
<td>2013-07-30 10:53:00-04</td>
<td>756.94</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>2013-07-17 19:25:00-04</td>
<td>2013-07-18 01:15:00-04</td>
<td>491.29</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>2013-11-10 11:05:00-04</td>
<td>2013-07-17 17:36:00-04</td>
<td>689.24</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>2013-08-01 14:04:00-04</td>
<td>2013-08-01 18:00:00-04</td>
<td>567.71</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>2013-08-26 09:00:00-04</td>
<td>2013-07-30 03:53:00-04</td>
<td>1,124.65</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>2013-08-15 04:00:00-04</td>
<td>2013-08-15 10:53:00-04</td>
<td>925.95</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>2013-07-29 07:00:00-04</td>
<td>2013-07-29 09:05:00-04</td>
<td>289.73</td>
<td>4</td>
</tr>
</tbody>
</table>
**itineraries table**

The table containing the list of passenger itineraries. Many passengers can have the same itinerary. Many passengers can have many itineraries. Once an itinerary, (which may contain more than one flight), has been booked, the per person cost for the itinerary (not the flight) will be stored. This cost would be determined by the interfacing application, after taxes and fees have been included. When the total cost needs to be calculated, the cost can be multiplied by the number of passengers with the same itinerary number.

```sql
CREATE TABLE IF NOT EXISTS itineraries (  itinerary_id CHAR(6) NOT NULL UNIQUE,  per_person_price_usd MONEY NOT NULL CHECK (per_person_price_usd > 0.0::text::money),  PRIMARY KEY (itinerary_id)));
```

**functional dependencies**

```
  itinerary_id → per_person_price_usd
```

**sample data**

<table>
<thead>
<tr>
<th>itinerary_id</th>
<th>per_person_price_usd</th>
</tr>
</thead>
<tbody>
<tr>
<td>BM87C0</td>
<td>1347.57</td>
</tr>
<tr>
<td>DZB665</td>
<td>350.89</td>
</tr>
<tr>
<td>MGEWFT</td>
<td>1187.12</td>
</tr>
<tr>
<td>WXPL21</td>
<td>689.67</td>
</tr>
<tr>
<td>AX9R3E</td>
<td>975.41</td>
</tr>
<tr>
<td>P4XBRR</td>
<td>734.72</td>
</tr>
<tr>
<td>KL2CA5</td>
<td>513.14</td>
</tr>
</tbody>
</table>
passengers_on_flights table
Contains information about which flight a passenger is on, their itinerary number, and the seat number that he or she is assigned. Since `seat_num` is not normalized and has no constraints, a trigger has been defined to address this issue as it is important that one seat not be assigned to multiple passengers on a scheduled flight.

```
CREATE TABLE IF NOT EXISTS passengers_on_flights (  
  itinerary_id INTEGER NOT NULL,  
  route_id INTEGER NOT NULL,  
  depart_timestamp TIMESTAMP WITH TIME ZONE NOT NULL,  
  person_id INTEGER NOT NULL,  
  seat_num CHAR(3) NOT NULL,  
  PRIMARY KEY (itinerary_id, route_id, depart_timestamp, person_id),  
  FOREIGN KEY (person_id) REFERENCES passengers(person_id)
);
```

functional dependencies
- `itinerary_id, route_id, depart_timestamp, person_id` → `seat_num`

sample data

<table>
<thead>
<tr>
<th>itinerary_id</th>
<th>route_id</th>
<th>depart_timestamp</th>
<th>person_id</th>
<th>seat_num</th>
</tr>
</thead>
<tbody>
<tr>
<td>BM87C0</td>
<td>1</td>
<td>2013-05-18 13:18:00-04</td>
<td>1</td>
<td>1A</td>
</tr>
<tr>
<td>BM87C0</td>
<td>1</td>
<td>2013-05-18 13:18:00-04</td>
<td>2</td>
<td>1B</td>
</tr>
<tr>
<td>BM87C0</td>
<td>1</td>
<td>2013-05-18 13:18:00-04</td>
<td>3</td>
<td>1C</td>
</tr>
<tr>
<td>P4XBRR</td>
<td>2</td>
<td>2014-05-28 02:35:00-04</td>
<td>1</td>
<td>2A</td>
</tr>
<tr>
<td>P4XBRR</td>
<td>5</td>
<td>2013-07-17 19:25:00-04</td>
<td>1</td>
<td>2A</td>
</tr>
<tr>
<td>AX9R3E</td>
<td>3</td>
<td>2013-08-26 09:00:00-04</td>
<td>7</td>
<td>31B</td>
</tr>
<tr>
<td>AX9R3E</td>
<td>3</td>
<td>2013-08-26 09:00:00-04</td>
<td>8</td>
<td>11A</td>
</tr>
<tr>
<td>KL2CA5</td>
<td>10</td>
<td>2013-08-15 04:00:00-04</td>
<td>4</td>
<td>11B</td>
</tr>
<tr>
<td>KL2CA5</td>
<td>10</td>
<td>2013-08-15 04:00:00-04</td>
<td>5</td>
<td>11C</td>
</tr>
<tr>
<td>KL2CA5</td>
<td>10</td>
<td>2013-08-15 04:00:00-04</td>
<td>6</td>
<td>11D</td>
</tr>
<tr>
<td>KL2CA5</td>
<td>10</td>
<td>2013-08-15 04:00:00-04</td>
<td>1</td>
<td>10D</td>
</tr>
</tbody>
</table>
CREATE OR REPLACE FUNCTION valid_flight_seat_trigger()
RETURNS trigger AS $$
DECLARE
  seat_count INTEGER := 0;
  seat_avail INTEGER := 0;
BEGIN

  -- Is seat number specified?
  IF NEW.seat_num IS NULL THEN
    RAISE EXCEPTION 'Invalid seat_num given';
  END IF;

  -- Is seat number valid for the aircraft flying this route?
  SELECT COUNT(s.seat_num)
  INTO seat_count
  FROM routes r
  INNER JOIN aircrafts a
    ON r.aircraft_id = a.aircraft_id
  INNER JOIN aircraft_seats s
    ON a.aircraft_id = s.aircraft_id
  WHERE r.route_id = NEW.route_id
    AND s.seat_num = NEW.seat_num;

  IF (seat_count = 1) THEN

    -- Is seat number for the flight available?
    SELECT COUNT(seat_num)
    INTO seat_avail
    FROM passengers_on_flights
    WHERE route_id = NEW.route_id
      AND depart_timestamp = NEW.depart_timestamp
      AND seat_num = NEW.seat_num;

    IF (seat_avail != 0) THEN
      RAISE EXCEPTION 'Seat for this flight is occupied.';
    END IF;

  ELSE
    RAISE EXCEPTION 'Invalid seat number for this aircraft.';
  END IF;
END IF;
$$_;

In PostgreSQL, the main logic for triggers is contained in a stored procedure that is specified by the code: RETURNS trigger. The procedure must then be specified in the CREATE TRIGGER statement. The trigger will be called every time an UPDATE or INSERT command is executed on the passenger_on_flights table. Then, the valid_flight_seat_trigger procedure will be executed. There are two validation steps before the data in the tables can be modified. First, the trigger needs to determine if the seat number exists on the aircraft that is flying this route. Then, it must determine if the seat is occupied by another passenger. If there is a conflict, an error occurs and the changes will not be committed.
valid_flight_seat_trigger (continued)

IF (TG_OP = 'INSERT') THEN
    INSERT INTO passengers_on_flights (itinerary_id, route_id, depart_timestamp, person_id, seat_num)
    VALUES (NEW.itinerary_id, NEW.route_id, NEW.depart_timestamp, NEW.person_id, NEW.seat_num);
    RAISE NOTICE 'Passenger was assigned to flight and seat successfully.';

ELSIF (TG_OP = 'UPDATE') THEN
    UPDATE passengers_on_flights
    SET (itinerary_id, route_id, depart_timestamp, person_id, seat_num)
    = (NEW.itinerary_id, NEW.route_id, NEW.depart_timestamp, NEW.person_id, NEW.seat_num)
    WHERE itinerary_id = OLD.itinerary_id
    AND route_id = OLD.route_id
    AND depart_timestamp = OLD.depart_timestamp
    AND person_id = OLD.person_id;
    RAISE NOTICE 'Passenger seat assignment and/or flight updated successfully.';
END IF;

RETURN NULL;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER valid_flight_seat_trigger
BEFORE INSERT OR UPDATE ON passengers_on_flights
FOR EACH ROW
WHEN (pg_trigger_depth() = 0)
EXECUTE PROCEDURE valid_flight_seat_trigger();

Technical Note:  

pg_trigger_depth() is a PostgreSQL system information function that provides the current nesting level of the trigger. This is needed to prevent the INSERT INTO and UPDATE commands inside the trigger procedure from recursively activating the trigger, thus causing an infinite loop.
The duration of a flight is information that will need to be calculated frequently as it is useful to have on a passenger itinerary. The stored procedure allows the database to calculate the duration without forcing the interfacing application to calculate the value. At the same time, if the interfacing application wants to calculate the duration instead of the database, then `depart_timestamp` and `arrive_timestamp` can be selected from the `flights` table. The duration is calculated by subtracting the departure time stamp from the arrival time stamp.

```sql
CREATE OR REPLACE FUNCTION flight_duration(route_pk INTEGER, depart_ts_pk TIMESTAMP WITH TIME ZONE) RETURNS TIME AS $$
DECLARE
    duration TIME WITHOUT TIME ZONE;
BEGIN
    SELECT (arrive_timestamp::TIMESTAMP WITH TIME ZONE - depart_timestamp::TIMESTAMP WITH TIME ZONE) INTO duration
    FROM flights
    WHERE route_id = route_pk
    AND depart_timestamp = depart_ts_pk;
RETURN duration AS duration;
END;
$$ LANGUAGE plpgsql;
```
CREATE OR REPLACE FUNCTION layover_time(route_1 INTEGER, depart_ts_1 TIMESTAMPTZ, route_2 INTEGER, depart_ts_2 TIMESTAMPTZ)
RETURNS TIME AS $$
DECLARE
    arrive_time_1 TIMESTAMP WITH TIME ZONE;
    depart_time_2 TIMESTAMP WITH TIME ZONE;

BEGIN
    SELECT arrive_timestamp
    INTO arrive_time_1
    FROM flights
    WHERE route_id = route_1
     AND depart_timestamp = depart_ts_1;

    SELECT depart_timestamp
    INTO depart_time_2
    FROM flights
    WHERE route_id = route_2
     AND depart_timestamp = depart_ts_2;

RETURN (depart_time_2 - arrive_time_1);
END;
$$ LANGUAGE plpgsql;

Calculates the layover time between two flights. The layover time is when a passenger is not aboard an active flight and must wait for the next flight in his or her itinerary. The required parameters are the route_id and depart_timestamp for the first and second flight. The TIMESTAMPTZ is a synonym for the TIME STAMP WITH TIME ZONE data type. The arrival time for the first flight is subtracted from the departure time of the second flight.
flights_arrivals

Access to flight status information is absolutely necessary. This view could be used by airport systems that display information about flights. It could also be used by third parties, such as FlightAware—a flight tracking and status website.

CREATE VIEW flights_arrivals AS
    SELECT f.depart_timestamp,
           f.arrive_timestamp,
           r.flight_num,
           r.origin_airport,
           r.destination_airport
    FROM flights f
    INNER JOIN routes r
             ON f.route_id = r.route_id
    INNER JOIN airports a
             ON r.origin_airport = a.iata_code
             AND r.destination_airport = a.iata_code;

use example

SELECT *
FROM flights_arrivals
WHERE destination_airport = 'EWR'
ORDER BY arrive_timestamp DESC
LIMIT 20;

In the above example, arrival information can be narrowed down to show only those flights arriving at the airport the information screen is located. In addition, the list can be sorted in descending order, with the latest arrivals at the top of the result set. Since only so many rows can be displayed on the screen at once, the results can be reduced to the latest 20 flights.
passenger manifest for a scheduled flight

This report is useful for gate agents and flight attendants to be able print a physical copy of the list of passengers on the flight. Airlines are required to have a manifest on board the flight. Should a problem or unfortunate event occur, the crew will be able to identify and account for all on board passengers.

```sql
SELECT pers.first_name,
       pers.middle_name,
       pers.last_name,
       pers.gender,
       pf.seat_num
FROM passengers_on_flights pf
INNER JOIN passengers pass
    ON pf.person_id = pass.person_id
INNER JOIN persons pers
    ON pass.person_id = pers.person_id
WHERE route_id = '----'
    AND depart_timestamp = '----'
ORDER BY pers.last_name ASC;
```

use example

```sql
SELECT pers.first_name,
       pers.middle_name,
       pers.last_name,
       pers.gender,
       pf.seat_num
FROM passengers_on_flights pf
INNER JOIN passengers pass
    ON pf.person_id = pass.person_id
INNER JOIN persons pers
    ON pass.person_id = pers.person_id
WHERE route_id = '6'
    AND depart_timestamp = '2013-07-17 12:05:00-04'
ORDER BY pers.last_name ASC;
```
passengers with a TSA redress number

This report allows the airline to retrieve all individuals who are have a TSA redress number. It may be necessary for the airline to provide a list of these people for TSA security officers. Also, it should able to quickly identify these individuals for security reasons, should the airline be involved in a federal investigation.

```sql
SELECT
    pers.first_name,
    pers.middle_name,
    pers.last_name,
    pers.date_of_birth,
    pers.gender,
    pers.phone_primary,
    pass.tsa_redress_num
FROM
    passengers pass
INNER JOIN persons pers
    ON pass.person_id = pers.person_id
WHERE tsa_redress_num != '';```
a single itinerary

For functions related to searching for a flight, that is an individual who is a potential passenger of a flight, the application should interact with the database with the `flight_search` user. This user has read-only functionality on the appropriate tables. When the user decides to book a flight, the application will then use the `flight_book` user (next page).

```
SELECT pers.first_name,
pers.middle_name,
pers.last_name,
f.depart_timestamp,
arrive_timestamp,
r.origin_airport,
r.destination_airport,
r.distance_miles,
a.manufacturer,
a.model,
flight_duration(f.route_id, f.depart_timestamp)
FROM passengers_on_flights pf
INNER JOIN passengers pass
ON pf.person_id = pass.person_id
INNER JOIN persons pers
ON pass.person_id = pers.person_id
INNER JOIN flights f
ON pf.route_id = f.route_id
AND pf.depart_timestamp = f.depart_timestamp
INNER JOIN routes r
ON f.route_id = r.route_id
INNER JOIN aircrafts a
ON r.aircraft_id = a.aircraft_id
WHERE pf.itinerary_id = 'BM87C0';
```
**flight_search role**

For functions related to searching for a flight, that is an individual who is a potential passenger of a flight, the application should interact with the database with the `flight_search` user. This user has read-only functionality on the appropriate tables. When, the user decides to book a flight, the application will then use the `flight_book` user (next page).

```sql
REVOKE ALL PRIVILEGES ON employees FROM flight_search;
REVOKE ALL PRIVILEGES ON persons FROM flight_search;
REVOKE ALL PRIVILEGES ON passengers FROM flight_search;
REVOKE ALL PRIVILEGES ON itineraries FROM flight_search;
REVOKE ALL PRIVILEGES ON flights FROM flight_search;
REVOKE ALL PRIVILEGES ON route_statuses FROM flight_search;
REVOKE ALL PRIVILEGES ON routes FROM flight_search;
REVOKE ALL PRIVILEGES ON passengers_on_flights FROM flight_search;
REVOKE ALL PRIVILEGES ON countries FROM flight_search;
REVOKE ALL PRIVILEGES ON airports FROM flight_search;
REVOKE ALL PRIVILEGES ON aircrafts FROM flight_search;
REVOKE ALL PRIVILEGES ON aircraft_seats FROM flight_search;
REVOKE ALL PRIVILEGES ON seat_classes FROM flight_search;

GRANT SELECT ON flights FROM flight_search;
GRANT SELECT ON route_statuses FROM flight_search;
GRANT SELECT ON routes FROM flight_search;
GRANT SELECT ON passengers_on_flights FROM flight_search;
GRANT SELECT ON countries FROM flight_search;
GRANT SELECT ON airports FROM flight_search;
GRANT SELECT ON aircrafts FROM flight_search;
GRANT SELECT ON aircraft_seats FROM flight_search;
GRANT SELECT ON seat_classes FROM flight_search;
```
flight_book role

Once an individual decides to complete the purchase of his or her itinerary, the application should use the `flight_book` user that allows the potential passenger to book a flight and be classified as a passenger. When the user decides to book a flight, then he or she should be granted permission to insert new rows of data into the appropriate tables.

```sql
REVOKE ALL PRIVILEGES ON employees FROM flight_book;
REVOKE ALL PRIVILEGES ON persons FROM flight_book;
REVOKE ALL PRIVILEGES ON passengers FROM flight_book;
REVOKE ALL PRIVILEGES ON itineraries FROM flight_book;
REVOKE ALL PRIVILEGES ON flights FROM flight_book;
REVOKE ALL PRIVILEGES ON route_statuses FROM flight_book;
REVOKE ALL PRIVILEGES ON routes FROM flight_book;
REVOKE ALL PRIVILEGES ON passengers_on_flights FROM flight_book;
REVOKE ALL PRIVILEGES ON countries FROM flight_book;
REVOKE ALL PRIVILEGES ON airports FROM flight_book;
REVOKE ALL PRIVILEGES ON aircrafts FROM flight_book;
REVOKE ALL PRIVILEGES ON aircraft_seats FROM flight_book;
REVOKE ALL PRIVILEGES ON seat_classes FROM flight_book;

GRANT INSERT, SELECT ON persons FROM flight_book;
GRANT INSERT, SELECT ON passengers FROM flight_book;
GRANT SELECT ON flights FROM flight_book;
GRANT SELECT ON route_statuses FROM flight_book;
GRANT SELECT ON routes FROM flight_book;
GRANT SELECT ON passengers_on_flights FROM flight_book;
GRANT SELECT ON itineraries FROM flight_book;
GRANT SELECT ON countries FROM flight_book;
GRANT SELECT ON airports FROM flight_book;
GRANT SELECT ON aircrafts FROM flight_book;
GRANT SELECT ON aircraft_seats FROM flight_book;
GRANT SELECT ON seat_classes FROM flight_book;
```
passenger role

A passenger is a person (employee or non-employee) with a valid itinerary. After the individual becomes a passenger, the application should use this user role. This user also permits returning/loyal passengers with user accounts (not part of this database design) to edit information that may change over time. For example, name, phone number and email. For the tables in this design, the passenger should be allowed to only update rows of data, not insert new rows.

REVOKE ALL PRIVILEGES ON employees FROM passenger;
REVOKE ALL PRIVILEGES ON persons FROM passenger;
REVOKE ALL PRIVILEGES ON passengers FROM passenger;
REVOKE ALL PRIVILEGES ON itineraries FROM passenger;
REVOKE ALL PRIVILEGES ON flights FROM passenger;
REVOKE ALL PRIVILEGES ON route_statuses FROM passenger;
REVOKE ALL PRIVILEGES ON routes FROM passenger;
REVOKE ALL PRIVILEGES ON passengers_on_flights FROM passenger;
REVOKE ALL PRIVILEGES ON countries FROM passenger;
REVOKE ALL PRIVILEGES ON airports FROM passenger;
REVOKE ALL PRIVILEGES ON aircrafts FROM passenger;
REVOKE ALL PRIVILEGES ON aircraft_seats FROM passenger;
REVOKE ALL PRIVILEGES ON seat_classes FROM passenger;

GRANT SELECT, UPDATE ON persons FROM passenger;
GRANT SELECT, UPDATE ON passengers FROM passenger;
GRANT SELECT, UPDATE ON itineraries FROM passenger;
GRANT SELECT ON flights FROM passenger;
GRANT SELECT ON route_statuses FROM passenger;
GRANT SELECT ON routes FROM passenger;
GRANT SELECT ON passengers_on_flights FROM passenger;
GRANT SELECT ON countries FROM passenger;
GRANT SELECT ON airports FROM passenger;
GRANT SELECT ON aircrafts FROM passenger;
GRANT SELECT ON aircraft_seats FROM passenger;
GRANT SELECT ON seat_classes FROM passenger;
**ticket_agent role**

This user role is for applications that allow employees to check in customers when they arrive. Ticket agents should be allowed to update customer data, such as name, traveler numbers and seat assignments. In addition, they should be able to book flights for customers in the event of delayed, canceled or missed flights. They have access to all tables except for the `employees` table.

```sql
REVOKE ALL PRIVILEGES ON employees FROM ticket_agent;
REVOKE ALL PRIVILEGES ON persons FROM ticket_agent;
REVOKE ALL PRIVILEGES ON passengers FROM ticket_agent;
REVOKE ALL PRIVILEGES ON itineraries FROM ticket_agent;
REVOKE ALL PRIVILEGES ON flights FROM ticket_agent;
REVOKE ALL PRIVILEGES ON route_statuses FROM ticket_agent;
REVOKE ALL PRIVILEGES ON routes FROM ticket_agent;
REVOKE ALL PRIVILEGES ON passengers_on_flights FROM ticket_agent;
REVOKE ALL PRIVILEGES ON countries FROM ticket_agent;
REVOKE ALL PRIVILEGES ON airports FROM ticket_agent;
REVOKE ALL PRIVILEGES ON aircrafts FROM ticket_agent;
REVOKE ALL PRIVILEGES ON aircraft_seats FROM ticket_agent;
REVOKE ALL PRIVILEGES ON seat_classes FROM ticket_agent;

GRANT UPDATE, INSERT, SELECT ON persons FROM ticket_agent;
GRANT UPDATE, INSERT, SELECT ON passengers FROM ticket_agent;
GRANT UPDATE, INSERT, SELECT ON itineraries FROM ticket_agent;
GRANT SELECT ON flights FROM ticket_agent;
GRANT SELECT ON route_statuses FROM ticket_agent;
GRANT SELECT ON routes FROM ticket_agent;
GRANT UPDATE, INSERT, SELECT ON passengers_on_flights FROM ticket_agent;
GRANT SELECT ON countries FROM ticket_agent;
GRANT SELECT ON airports FROM ticket_agent;
GRANT SELECT ON aircrafts FROM ticket_agent;
GRANT SELECT ON aircraft_seats FROM ticket_agent;
GRANT SELECT ON seat_classes FROM ticket_agent;
```
• The interfacing software is expected to dynamically calculate
  – Appropriate connecting flights (i.e. two flights that do not overlap in time)
• Time zones
  – In the case of an INSERT: The server is expected to convert timestamps to the
    server's time zone.
  – The server stores time zones in UTC (Universal Coordinated Time), also known as
    GMT (Greenwich Mean Time).
• Airlines always change prices of flights according to demand/popularity, availability,
  date, time, etc.
  – The interfacing application is expected to calculate pricing that incorporates the
    mentioned factors. The database simply stores the base (or minimum) price of a
    single flight.
• TSA (Transportation Security Administration)
  – The Secure Flight Passenger Data Definitions document (version 1.0) provides
    guidelines on the different pieces of data to help companies in the airlines industry
    design their systems.
  – They recommend the length of the Redress Number be 13 characters
  – They recommend the length of the Known Traveler Number be 25 characters.
  – These numbers are assigned by the Department of Homeland Security (DHS)
What happens when flights have been completed?
  – The data should be transferred to a historical flight records table
Generation of unique itinerary numbers is not implemented by the database
Create more user roles, as the airline industry has more roles
More views should be created and used to interface with applications, to protect the underlying implementation and data.
• Add support for
  – Groups of flights for a single itinerary. For example, itineraries that have multiple destinations (or multiple sets of flights). Passengers should be allowed to book a multiple destination itinerary. A passenger might want to fly from Los Angeles to Denver on October 1st, then on October 9th, fly from Denver to New York. Then, on October 18th, fly from New York back to Los Angeles. With the current database design, this cannot be done by sorting all the flights according to departure time in ascending order. The system cannot determine which individual flight (leg) belongs to which group of flights.

  – Frequent flyer program (United MileagePlus®)

  – Special baggage (i.e. surfboards, live animals)

  – Codeshare flights (when two or more airlines share the same flight). This would require another table with a list of airlines, their unique carrier code (United's code is UA).