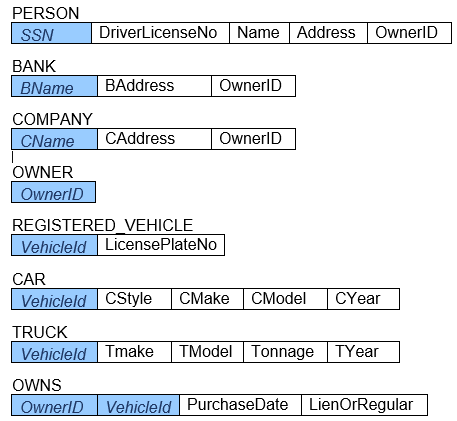
# Please answer all questions

1. Answer the following database-related questions: (50 points)
2. Explain the major challenges you may encounter when you implement an EER with a superclass/subclass relationship. (3 points)

While very feasable to implement EER with a superclass/subclass relationships, some challenges are present in the implementation. One challenge is to determine where the keys for each of the relations should go (in terms of to the parent or child) and determining primary keys for each of the relations. Other challenges include deciding which of the constraints to put on the parent/child relationship, whether the constraint is disjoint of overlapping. For implementing an EER with shared subclass keys, a decision must be made between creating two different child classes or implementing a specialization lattice. Other things to consider are the multiple options for specialization and generalization, as well as taking into consideration complications and performance of a database implementation of the EER. If all of these items are followed well, a good EER with superclass/subclass relationships can be created.

Mapping EER categories to relations shown below: (7 points)



Based on above design, answer the following questions:

Can one PERSON, BANK, or COMPANY have more than one OwnerID? (Yes or No) (1 point)

NO – Each of person, bank and company have only one field for owner id. If someone wanted another owner id they could potentially create another of person, bank or company in the database, but the two would have no relation within the database

Can one PERSON (or BANK, or COMPANY) OWN two REGISTERED\_VEHICLEs? (Yes or No) (1 point)

YES – Owns manages a many to many relationship between an owner (person, bank, company) and registered vehicle, thus one person can own multiple registered vehicles.

Can one PERSON and one BANK join to OWN a REGISTERED\_VEHICLE? (Yes or No) (1 point)

NO – Owns is a relation which manages the many to many relationship between person/bank and registered vehicle, and this specifies only one owner id. While a bank may own the car on lien and transfer ownership after completion of the lien, this would still have just one owner at a time.

Can two PERSONs join to OWN a REGISTERED\_VEHICLE? (Yes or No) (1 point)

NO – For the same reasons as above

Can you delete a PERSON record that has a value for OwnerId? (Yes or No or Depends) (1 point) How do you implement this? (2 points)

DEPENDS – If a person has a record for owner id which is not referenced in Owns you can delete the person. If the person’s owner id is present in owns, you can make a decision for how to handle this. SQL provides several ways. ON DELETE action in SQL allows for SET NULL, which would set the owner id to null, SET DEFAULT, which would set to a default value, perhaps the federal government if an owner disappears, or CASCADE, which would delete the rows in the child table. You can also add a trigger to do a custom action AFTER DELETE to do more complicated business logic. Depending on the exact functionality expected from the system, which would be based on business rules, one of these options may be considered.

1. Describe the analysis concept used during the normalization process. (3 points)

The basics of the analysis concept used during the normalization process is to follow a series of steps which help to analyze relations based on their primary keys and functional dependencies. The basic steps are as follows:

* Remove attributes with multiple values
* Ensure each attribute depends on the whole key for the relation
* Determine which attributes depend on other non-key attributes

This helps to review and validate relations to establish a proper set of relations that meet data requirements by removing all relational structures that provide multiple keys to know the same fact to help control and eliminate redundancy in data storage. A bonus is eliminating update anomalies and lossless joins to ensure no spurious tuple problems. The result of normalization is one fact in one place with one copy, leading to good database design principles, with data describing one and only one thing.

If a data model is normalized to 3NF, can we always say the model is a good design for the business? Explain your answer. (2 points)

No, different businesses have different objectives, and a blanket statement that 3NF is a good design for all businesses would not be accurate. While 3NF aims to eliminate data redundancy and provide a structured way to work with data, this may not always be the best for a given business. One scenario where it may not make sense to convert to 3NF for a business case is if the company wants access to data that would require time-consuming JOIN processes to retrieve with 3NF. Sometimes decomposing the database down to earlier forms 1NF, 2NF, etc. will result in faster data retrieval, even if the data structure of 3NF is not maintained.

What design process can help avoid having to resolve higher forms of normalization? (2 points)

The design process that can help with higher forms of normalization is the conceptual database design phase. This is the phase where the basic structure of the database is determined, thus having a poor conceptual design phase for a database system can be disastrous for the project. While the normalized forms are generally useful in database design, even these do not always perfectly match with the business case for the database. Having a good design will allow less worry about normalizing to 4NF, 5NF, and even makes lower order NF’s more optional so long as it is for a business purpose.

Relation R = {**A**, **B**, C, D, E, F, G, H, I, J} where **A** and **B** are jointed PK.

R has the following Functional dependencies (FDs):

{A, B} **→** {J};

{A} **→** {H, I};

{B} **→** {G};

{G} **→** {E, F};

{H} **→** {C, D}

Use the functional dependency rules to identify and correct two incomplete FDs and explain your reason(s). You do NOT need to perform normalization. (3 points)

Note: Both corrections are done independently of eachother, and do not fully move R into 3NF.

The first thing I notice is that not every non-key attribute is fully dependant on the primary key. To move R into 2NF, we can split into three tables, one that uses A as a primary key, one that uses B as a primary key, and one which uses the composite AB key. This is shown below:

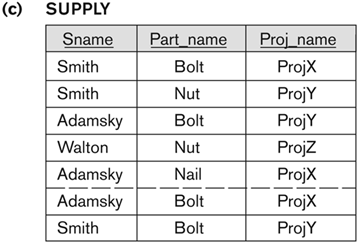
{**A**, H, I, C, D}  
{**B**, G, E, F}  
{**A**, **B**, J}

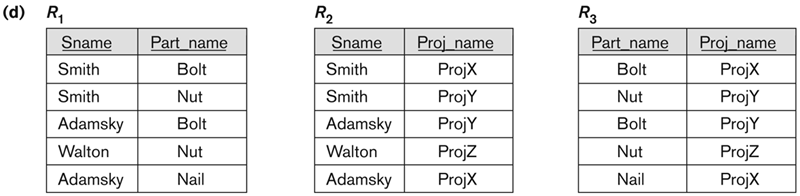
Another thing that I note is that E and F depend only on G. This means that we can split the relation into:  
{**A**, **B**, C, D, G, H, I, J}  
{**G**, E, F}  
Which ensures that no non-key attribute of R is functionally dependant on another non-key attribute, a part of the requirements for 3NF.

1. Explain multivalued dependency (MVD) and why this may be undesirable. (5 points)

MVD, which stands for multivalued dependency represents a dependency between attributes in a relation such that for each value of one entity there exist an independent set of values for two other entities. MVD’s are a consequence of 1NF which disallows attributes with multiple values, and sometimes occur when two independent 1:M relationships are mixed in the same relation. Having MVD’s present in a database can cause data redundancy, and break the principle of a single source of truth in a database.

One reason to perform normalization is to remove redundancy. Please explain why the relation SUPPLY below is normalized to 5NF, but the total size of relations R1, R2, and R3 below increases, not decreases. (5 points)





Supply (c) is normalized from 4NF to 5NF in (d) by removing the join dependencies in the relation. A relation informally may be in 4NF but not 5NF if a relation contains three M:N relationships, which is the case in (c). Splitting the relationships into three different relations allows for the three relations to have no repetition of facts and reconstruction of all three relations using join. The reason the total size of the relations R1, R2, R3 increase in comparison to the relation under (c) is that each of the items from (c) need to be stated twice in (d) to show both of the item’s M:N relationships. Since there are only 2 tuples of repeated data, moving to 5NF creates 3 relations with total size greater than the 4NF model, however it eliminates some of the repeated data and forces one source of truth for the data, which may not be the case in 4NF.

1. Do you create an ordered file whose records are based on a primary key such as customer name and use a primary index on the attribute? Explain your reasons. (5 points)

The reason you would create an ordered file with a primary key is if you want to speed up data retrieval, and have no concerns with data update. So for an application that needed faster data retrieval for a record, using an index would be useful.

One item of note is that customer name may not be the best choice for a primary key for a system. Customer name may not be unique, thus may not be a feasible primary key. Some other options for primary keys are items like username, or customer id, both of which would have to be unique.

One method of indexing data would be to determine a unique item and attach an index to that item. Then by checking the index, the system can easily find the memory location of the index using the index file. This would greatly speed up data retrieval for an application with many records. Ordering the file would allow the system to use a search method like BST to easily find the data record.

Compare and contrast B-Tree and B+-tree. (5 points)

Both the B-tree and B+tree are dynamic, multilevel indexes and ensure balanced tree structure using additional constraints. B-trees have tree pointers, keys and data pointers stored in either leaf or internal nodes, whereas in the B+tree data pointers are stored only in leaf nodes. There is no common storage structure that is optimal for all database applications, the B-tree and B+tree index structure gives good all-around performance for most common database structures. B+trees have high fan out numbers which reduce I/O operations to find a value in the tree and uses the data in the leaf node to find a memory location for a desired record. B+trees also allow for in-order traversal for good range search which has low memory overhead.

1. Explain the following primary file organizations: (1) a heap file (2) a sorted file (3) a hashed file. (3 Points)

Heap files are a simple file structure, records are placed anywhere in the file where there is space, which results in unordered files. Records are stored in random order and data retrieval takes O(n) if there is no access path for the file.

Sorted files are also very simple, and simply sorts the data based on a user-specified key. Searching a sorted file system by primary key will result in O(lgn) runtime. Sorted files are expensive to maintain.

Hashed files are slightly more complicated, using a hashing function from the value of the hash field to generate an address to store a record. Searches using the hash field can return in O(1) time, however some decisions need to be made in terms of collisions and managing unused space.

What type of file organization is commonly used by RDBMS? (1 Point)

RDBMS typically uses heap files to store files in a database. Sorted and heap implementation can be created as well, but the default is heap/unordered.

Why is choosing a file organization important for a database? (1 Point)

Each of the different file organization techniques shared above have different pros and cons. Heap is the fastest for data entry and is the simplest. Sorted is simple, has fast data retrieval, however is relatively slow for data upload. Hashing is fast for entry and retrieval (if the hashing function results in uniform distribution of addresses), however is more complicated and decisions need to be made for collisions. Since each method has distinct advantages and disadvantages in terms of efficiency, it is important to consider each method for optimization of a database dependant on use case.

If the primary index file contains ***bi*** blocks, how many of block accesses to get the block containing a particular record? (2 points)

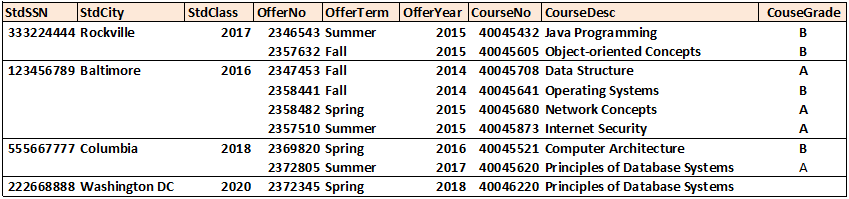
The number of block accesses needed to search and retrieve a record from a file can be given as bi + 1.

In relational database, how do you enforce data quality, integrity rules? (3 points)

In relational databases, you can enforce integrity rules with a series of integrity constraints or rules. For example, each column in a database can have constraints for what type of data may be contained, whether it is a Boolean value, numeric, float, char, varchar, etc. Data length can be specified to limit the size of strings or numeric values. You can also control referential integrity in relational databases by forcing column values to match existing primary key values in a relation (or be null), and rules can be added to determine what should happen if a referenced primary key is updated. Entity integrity concerns the concept of a primary key, and requires that each table have a primary key and each primary key must have a non-null, unique value.

1. Normalization:

**2.a** Describe Functional Dependencies (FDs) of the unnormalized University Database table (see below). Based on your FDs, what is the First Normal Form (1NF) and what are the steps for converting to the 1NF? What is the Second Normal Form (2NF) and what are the steps for converting to the 2NF? What is the Third Normal Form (3NF) and what are the steps for converting to 3NF? Use the example below to demonstrate the process and the functional dependencies between attributes. (15 points)



**Note:**

StdClass is the year the student graduates; each student has an assigned year (e.g. 2020).

OfferNo is a unique number associated with a course offering.

Each student will take many courses and get a Course grade of each taken course offering.

Explain the steps you will take through each NF.

Show functional dependencies on each relation through each NF.

You do NOT need to populate data in your normalization process, just show FDs and relations for 1NF, 2NF, and 3NF.

The data shown in the above table does not meet 1NF, thus three normalization steps must be taken to ensure the data is in 3NF. Each of the steps is taken independently and in order to ensure that the data meets 3NF correctly.

1NF: Remove attributes with multiple values:

We note that students take multiple classes. Thus, the given relation can be split into two relations as shown below:

Student: {StdSSN} → {StdCity, StdClass}  
Student Course: {StdSSN, OfferNo} → {OfferTerm, OfferYear, CourseNo, CourseDesc, CourseGrade}  
This will ensure that the relations are in 1NF

2NF: Ensure each attribute depends on the whole key for the relation.

We note that some of the attributes can be referened with just OfferNo and StdSSN has no bearing on those values. Thus we can further split our relation as shown:

Student: {StdSSN} → {StdCity, StdClass}  
Student Course: {StdSSN, OfferNo} → {CourseGrade}

Offering: {OfferNo} → {OfferTerm, OfferYear, CourseNo, CourseDesc}

This will ensure that each relation is in 2NF

3NF: Determine which attributes depend on other non-key attributes

We note that some attributes like CourseDesc would depend on CourseNo rather than OfferNo. We can split these relations as shown:

Student: {StdSSN} → {StdCity, StdClass}  
Student Course: {StdSSN, OfferNo} → {CourseGrade}

Offering: {OfferNo} → {OfferTerm, OfferYear, CourseNo}

Course: {CourseNo} → {CourseDesc}

These are the functional dependencies for the described table include StdCity and StdClass being dependant on StdSSN, CourseGrade being dependant on both StdSSN and OfferNo, OfferTerm, OfferYear, and CourseNo being dependant on OfferNo, and CourseDesc being dependant on CourseNo. The following are the created functional dependencies:

{StdSSN} → {StdCity, StdClass}  
{StdSSN, OfferNo} → {CourseGrade}

{OfferNo} → {OfferTerm, OfferYear, CourseNo}

{CourseNo} → {CourseDesc}

**2.b** Consider the following relation

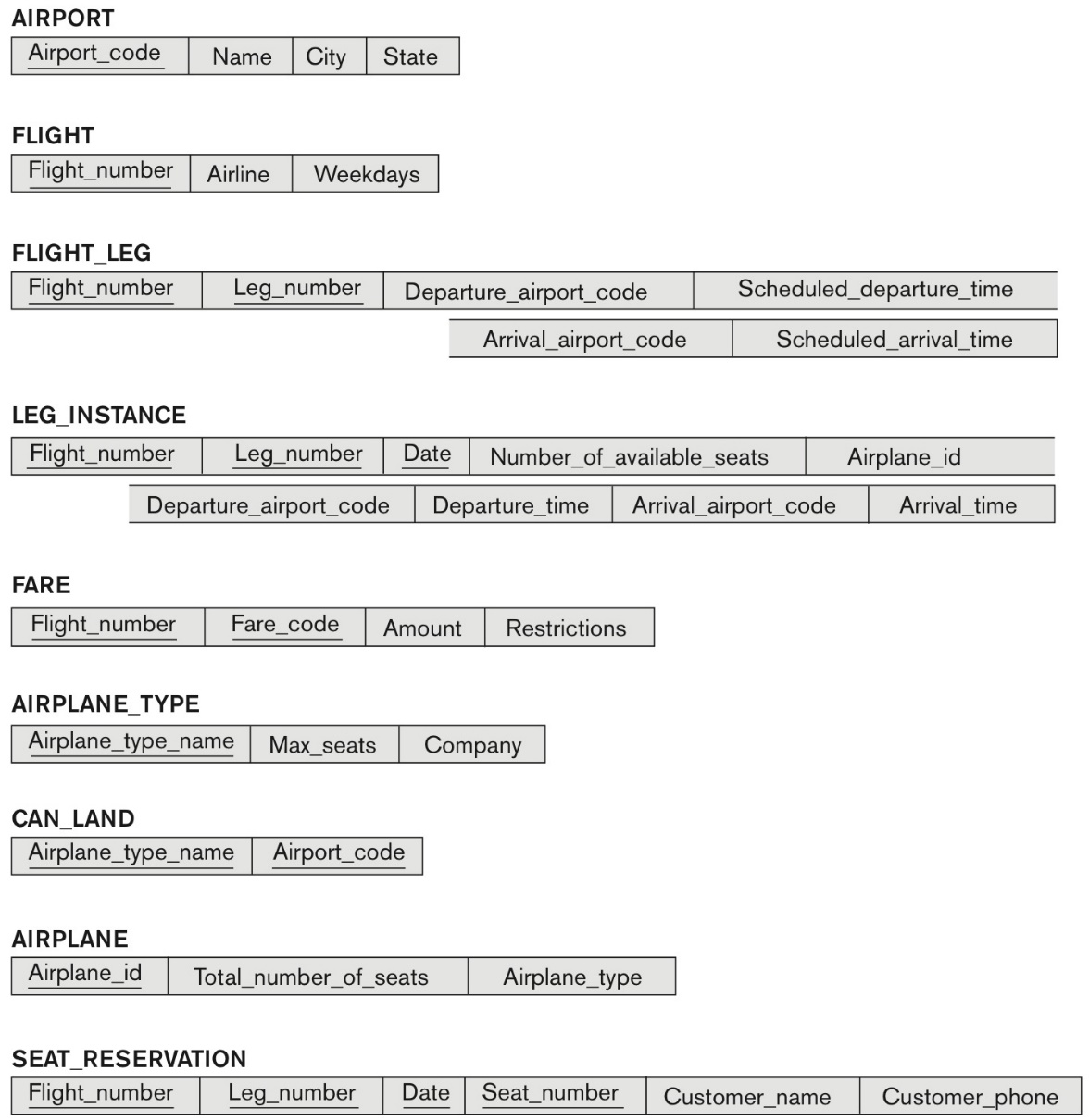
SUPPLIER\_PART\_PROJECT

|  |  |  |  |
| --- | --- | --- | --- |
| Supplier# | Part# | Project# | Quantity |

The business requirement is to track the quantity of a particular part from one particular supplier for one project. Based on the given primary key (Supplier#, Part#, Project#), please specify the relation is in (1NF, 2NF, 3NF, BCNF, 4NF, 5NF) compliant (multiple choices). Please explain your rationale (5 points).

Assuming there are no attributes with multiple values and the quantity of a part is dependent on the supplier#, part# and project#, then the relation is at least in 3NF. The relation seems to be in BCNF as well since every determinant in the relation is a candidate key. Since quantity cannot appear in a relation without all three of supplier#, part# and project#, there is no lossless decomposition that can remove redundancies between supplier#, part# and project#. While there are potentially 3 M:N relationships in the above relation, because quantity depends on all three, the relation is in 5NF. This is further shown by the lack of non-trivial multi-valued dependencies, a prerequisite for 4NF, and lack of join dependencies, a prerequisite for 5NF. Thus, the relation is in 5NF, despite some data redundancy.

1. Create an Airline database schema based on the ERD in the textbook (also see below). You need to implement your DDL statements. Your DDL statements should clearly specify the Primary Key, Foreign Key, Unique, NOT NULL and Check constraints.



Part 1: You need to implement the following business rules that are declaratively specified in your schema or be implemented through triggers. If your RDBMS support triggers, you may implement your trigger solutions. If your RDBMS does not support triggers, you need to show your trigger scripts (no need to implement them) (15 points):

Note: for each part (a-e) an excerpt is taken from the full SQL statement transcript shown below to show how each of the business constraints was handled.

* + - * 1. The Number in the FLIGHT must be unique.

CONSTRAINT flight\_pk

PRIMARY KEY (flight\_number)

* + - * 1. The fare amount (Amount) is in a range of ($0 – $10,000).

CONSTRAINT max\_fare

CHECK (amount <= 10000),

CHECK (amount >= 0)

* + - * 1. The maximum seats (Max\_seats) for any airplane type cannot exceed 600.

CONSTRAINT max\_600\_seats

CHECK (600 >= max\_seats)

* + - * 1. The maximum number of flight legs (leg\_no in the FLIGHT\_LEG) cannot exceed 4.

leg\_number INT NOT NULL,  
CONSTRAINT four\_legs

CHECK (0 < leg\_number),

CHECK (leg\_number <= 4),

* + - * 1. For any instance of a flight leg, the date of (Date in the LEG\_INSTANCE) must be either current date or a future date.

date DATE NOT NULL,  
CONSTRAINT check\_date

CHECK (DATE >= NOW())

See all business rules in constraints in below SQL statements. Note that since the flight number is a primary key in flight, it must be unique.

Part 2: You need to implement your DML statements with sample test data to retrieve the following information (15 points):

1. Create a list of aircraft types that can land in the airport at Washington Dulles International Airport (Airport\_code is '**IAD**').

Query:

select \* from can\_land

where airport\_code = 'IAD'

Result:



1. List all fare information for flight '**United 189**'.

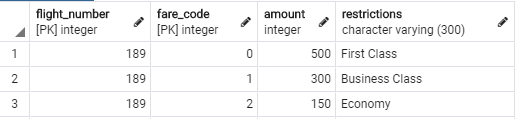
Query:

select fare.flight\_number, fare\_code, amount, restrictions from fare

join flight on (flight.flight\_number = fare.flight\_number)

where flight.flight\_number = 189 and flight.airline = 'United'

Result:



1. Create a list of direct flights (including scheduled departure time and arrival time) starting from Baltimore Washington International Airport (Airport\_code is '**BWI**') and terminating at San Francisco International Airport (Airport\_code is '**SFO**') which have more than two seats available on 8/15/2020. In addition, create a list of direct returning flights from '**SFO**' to '**BWI**' on 8/22/2020.

Query:

select date, number\_available\_seats, leg\_instance.departure\_airport\_code, leg\_instance.arrival\_airport\_code,

departure\_time, scheduled\_departure\_time, arrival\_time, scheduled\_arrival\_time

from leg\_instance

join flight\_leg

on leg\_instance.flight\_number = flight\_leg.flight\_number and

leg\_instance.leg\_number = flight\_leg.leg\_number

where

(leg\_instance.departure\_airport\_code = 'BWI' and

leg\_instance.arrival\_airport\_code = 'SFO' and

leg\_instance.number\_available\_seats >= 2 and

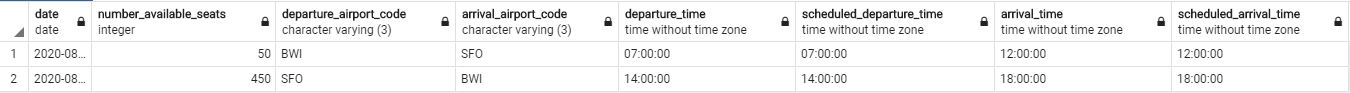
leg\_instance.date = '8/15/2020') or

(leg\_instance.departure\_airport\_code = 'SFO' and

leg\_instance.arrival\_airport\_code = 'BWI' and

leg\_instance.date = '8/22/2020')

Result:



You need to make up sample data for the tables. Then, you will write queries against your database and sample data to demonstrate your queries working as they should. You need to turn in your DDLs (create tables) and your SQLs. You are required to enter a few sample test data and perform queries. You may turn in your DBMS, the log file or screenshots that include your DDLs in a document (e.g., Word or text file) to prove your work is done properly.

Note: I used PostgreSQL for my RDBMS for problem 3. You can view my SQL statements to create the database below:

CREATE TABLE airport (

airport\_code VARCHAR(3) NOT NULL,

name VARCHAR(50) NOT NULL,

city VARCHAR(30) NOT NULL,

state VARCHAR(15) NOT NULL,

CONSTRAINT airport\_code\_pk

PRIMARY KEY (airport\_code)

);

insert into airport(airport\_code, name, city, state) values

('IAD', 'Washington Dulles International Airport', 'Dulles', 'Virginia');

insert into airport(airport\_code, name, city, state) values

('BWI', 'Baltimore Washington International Airport', 'Baltimore', 'Maryland');

insert into airport(airport\_code, name, city, state) values

('SFO', 'San Francisco International Airport', 'San Francisco', 'California');

CREATE TABLE flight (

flight\_number INT NOT NULL,

airline VARCHAR(30) NOT NULL,

weekdays VARCHAR(7) NOT NULL,

CONSTRAINT flight\_pk

PRIMARY KEY (flight\_number)

);

insert into flight(flight\_number, airline, weekdays) values

(0, 'American Airlines', 'MWF');

insert into flight(flight\_number, airline, weekdays) values

(1, 'Delta', 'MTWRFSU');

insert into flight(flight\_number, airline, weekdays) values

(2, 'Spirit', 'T');

insert into flight(flight\_number, airline, weekdays) values

(3, 'United', 'MTWRFSU');

insert into flight(flight\_number, airline, weekdays) values

(189, 'United', 'R');

CREATE TABLE flight\_leg (

flight\_number INT NOT NULL,

leg\_number INT NOT NULL,

departure\_airport\_code VARCHAR(3) NOT NULL,

arrival\_airport\_code VARCHAR(3) NOT NULL,

scheduled\_departure\_time TIME NOT NULL,

scheduled\_arrival\_time TIME NOT NULL,

CONSTRAINT flight\_leg\_pk

PRIMARY KEY (flight\_number, leg\_number),

CONSTRAINT flight\_num\_fk

FOREIGN KEY (flight\_number) REFERENCES flight(flight\_number)

ON DELETE SET NULL ON UPDATE CASCADE,

CONSTRAINT four\_legs

CHECK (0 < leg\_number),

CHECK (leg\_number <= 4),

CONSTRAINT arrival\_airport\_fk

FOREIGN KEY (arrival\_airport\_code) REFERENCES airport(airport\_code)

ON DELETE SET NULL ON UPDATE CASCADE,

CONSTRAINT departure\_airport\_fk

FOREIGN KEY (departure\_airport\_code) REFERENCES airport(airport\_code)

ON DELETE SET NULL ON UPDATE CASCADE

);

insert into flight\_leg(flight\_number, leg\_number, departure\_airport\_code, arrival\_airport\_code, scheduled\_departure\_time, scheduled\_arrival\_time) values

(0, 1, 'BWI', 'IAD', '01:00', '05:00');

insert into flight\_leg(flight\_number, leg\_number, departure\_airport\_code, arrival\_airport\_code, scheduled\_departure\_time, scheduled\_arrival\_time) values

(0, 2, 'IAD', 'SFO', '07:00', '14:30');

insert into flight\_leg(flight\_number, leg\_number, departure\_airport\_code, arrival\_airport\_code, scheduled\_departure\_time, scheduled\_arrival\_time) values

(1, 1, 'BWI', 'SFO', '07:00', '12:00');

insert into flight\_leg(flight\_number, leg\_number, departure\_airport\_code, arrival\_airport\_code, scheduled\_departure\_time, scheduled\_arrival\_time) values

(2, 1, 'SFO', 'BWI', '14:00', '18:00');

CREATE TABLE airplane\_type (

airplane\_type\_name VARCHAR(30) NOT NULL,

max\_seats INT NOT NULL,

company VARCHAR(30) NOT NULL,

CONSTRAINT airplane\_type\_pk

PRIMARY KEY (airplane\_type\_name),

CONSTRAINT max\_600\_seats

CHECK (600 >= max\_seats)

);

insert into airplane\_type(airplane\_type\_name, max\_seats, company) values

('737 NG', 400, 'Boeing');

insert into airplane\_type(airplane\_type\_name, max\_seats, company) values

('A330neo', 450, 'Airbus');

CREATE TABLE airplane (

airplane\_id INT NOT NULL,

total\_number\_seats INT NOT NULL,

airplane\_type VARCHAR(30) NOT NULL,

CONSTRAINT airplane\_pk

PRIMARY KEY (airplane\_id),

CONSTRAINT airplane\_type\_fk

FOREIGN KEY (airplane\_type) REFERENCES airplane\_type(airplane\_type\_name)

ON DELETE SET NULL ON UPDATE CASCADE

);

insert into airplane(airplane\_id, total\_number\_seats, airplane\_type) values

(0, 300, '737 NG');

insert into airplane(airplane\_id, total\_number\_seats, airplane\_type) values

(1, 450, 'A330neo');

CREATE TABLE leg\_instance (

flight\_number INT NOT NULL,

leg\_number INT NOT NULL,

date DATE NOT NULL,

number\_available\_seats INT NOT NULL,

airplane\_id INT NOT NULL,

departure\_airport\_code VARCHAR (3) NOT NULL,

departure\_time TIME NOT NULL,

arrival\_airport\_code VARCHAR(3) NOT NULL,

arrival\_time TIME NOT NULL,

CONSTRAINT leg\_instance\_pk

PRIMARY KEY (flight\_number, leg\_number, date),

CONSTRAINT flight\_leg\_fk

FOREIGN KEY (flight\_number, leg\_number) REFERENCES flight\_leg(flight\_number, leg\_number)

ON DELETE SET NULL ON UPDATE CASCADE,

CONSTRAINT airplane\_id\_fk

FOREIGN KEY (airplane\_id) REFERENCES airplane(airplane\_id)

ON DELETE SET NULL ON UPDATE CASCADE,

CONSTRAINT arrival\_airport\_fk

FOREIGN KEY (arrival\_airport\_code) REFERENCES airport(airport\_code)

ON DELETE SET NULL ON UPDATE CASCADE,

CONSTRAINT departure\_airport\_fk

FOREIGN KEY (departure\_airport\_code) REFERENCES airport(airport\_code)

ON DELETE SET NULL ON UPDATE CASCADE,

CONSTRAINT check\_date

CHECK (DATE >= NOW())

);

insert into leg\_instance(flight\_number, leg\_number, date, number\_available\_seats, airplane\_id, departure\_airport\_code, departure\_time, arrival\_airport\_code, arrival\_time) values

(0, 1, '8/17/2020', 450, 1, 'BWI', '01:00', 'IAD', '05:00');

insert into leg\_instance(flight\_number, leg\_number, date, number\_available\_seats, airplane\_id, departure\_airport\_code, departure\_time, arrival\_airport\_code, arrival\_time) values

(0, 2, '8/17/2020', 450, 1, 'BWI', '07:00', 'IAD', '14:30');

insert into leg\_instance(flight\_number, leg\_number, date, number\_available\_seats, airplane\_id, departure\_airport\_code, departure\_time, arrival\_airport\_code, arrival\_time) values

(1, 1, '8/15/2020', 50, 1, 'BWI', '07:00', 'SFO', '12:00');

insert into leg\_instance(flight\_number, leg\_number, date, number\_available\_seats, airplane\_id, departure\_airport\_code, departure\_time, arrival\_airport\_code, arrival\_time) values

(1, 1, '8/17/2020', 3, 1, 'BWI', '07:00', 'SFO', '12:00');

insert into leg\_instance(flight\_number, leg\_number, date, number\_available\_seats, airplane\_id, departure\_airport\_code, departure\_time, arrival\_airport\_code, arrival\_time) values

(1, 1, '8/19/2020', 0, 1, 'BWI', '07:00', 'SFO', '12:00');

insert into leg\_instance(flight\_number, leg\_number, date, number\_available\_seats, airplane\_id, departure\_airport\_code, departure\_time, arrival\_airport\_code, arrival\_time) values

(2, 1, '8/22/2020', 450, 0, 'SFO', '14:00', 'BWI', '18:00');

insert into leg\_instance(flight\_number, leg\_number, date, number\_available\_seats, airplane\_id, departure\_airport\_code, departure\_time, arrival\_airport\_code, arrival\_time) values

(2, 1, '8/24/2020', 450, 0, 'SFO', '14:00', 'BWI', '18:00');

CREATE TABLE fare (

flight\_number INT NOT NULL,

fare\_code INT NOT NULL,

amount INT NOT NULL,

restrictions VARCHAR(300),

CONSTRAINT fare\_pk

PRIMARY KEY (flight\_number, fare\_code),

CONSTRAINT flight\_number\_fk

FOREIGN KEY (flight\_number) REFERENCES flight(flight\_number)

ON DELETE SET NULL ON UPDATE CASCADE,

CONSTRAINT max\_fare

CHECK (amount <= 10000),

CHECK (amount >= 0)

);

insert into fare(flight\_number, fare\_code, amount, restrictions) values

(0, 0, 350, null);

insert into fare(flight\_number, fare\_code, amount, restrictions) values

(1, 0, 100, 'Must purchase one week before flight');

insert into fare(flight\_number, fare\_code, amount, restrictions) values

(1, 1, 300, null);

insert into fare(flight\_number, fare\_code, amount, restrictions) values

(2, 0, 500, null);

insert into fare(flight\_number, fare\_code, amount, restrictions) values

(3, 0, 2000, null);

insert into fare(flight\_number, fare\_code, amount, restrictions) values

(189, 0, 500, 'First Class');

insert into fare(flight\_number, fare\_code, amount, restrictions) values

(189, 1, 300, 'Business Class');

insert into fare(flight\_number, fare\_code, amount, restrictions) values

(189, 2, 150, 'Economy');

CREATE TABLE can\_land (

airplane\_type\_name VARCHAR(30) NOT NULL,

airport\_code VARCHAR(3) NOT NULL,

CONSTRAINT can\_land\_pk

PRIMARY KEY (airplane\_type\_name, airport\_code),

CONSTRAINT airplane\_type\_name\_fk

FOREIGN KEY (airplane\_type\_name) REFERENCES airplane\_type(airplane\_type\_name)

ON DELETE SET NULL ON UPDATE CASCADE,

CONSTRAINT airport\_code\_fk

FOREIGN KEY (airport\_code) REFERENCES airport(airport\_code)

ON DELETE SET NULL ON UPDATE CASCADE

);

insert into can\_land(airplane\_type\_name, airport\_code) values

('737 NG', 'IAD');

insert into can\_land(airplane\_type\_name, airport\_code) values

('737 NG', 'BWI');

insert into can\_land(airplane\_type\_name, airport\_code) values

('737 NG', 'SFO');

insert into can\_land(airplane\_type\_name, airport\_code) values

('A330neo', 'IAD');

insert into can\_land(airplane\_type\_name, airport\_code) values

('A330neo', 'BWI');

insert into can\_land(airplane\_type\_name, airport\_code) values

('A330neo', 'SFO');

CREATE TABLE seat\_reservation (

flight\_number INT NOT NULL,

leg\_number INT NOT NULL,

date DATE NOT NULL,

seat\_number VARCHAR(3) NOT NULL,

customer\_name VARCHAR(50) NOT NULL,

customer\_phone VARCHAR(20),

CONSTRAINT res\_pk

PRIMARY KEY (flight\_number, leg\_number, date, seat\_number),

CONSTRAINT leg\_instance\_fk

FOREIGN KEY (flight\_number, leg\_number, date) REFERENCES leg\_instance(flight\_number, leg\_number, date)

ON DELETE SET NULL ON UPDATE CASCADE

);

insert into seat\_reservation(flight\_number, leg\_number, date, seat\_number, customer\_name, customer\_phone) values

(0, 1, '8/17/2020', '34A', 'Steven Stamkos', null);

insert into seat\_reservation(flight\_number, leg\_number, date, seat\_number, customer\_name, customer\_phone) values

(0, 2, '8/17/2020', '26F', 'Steven Stamkos', null);