Assignment 5: The Entity-Relationship Model

Since this assignment is new, it is especially important that we get feedback about the assignment. You will get +3 points for filling out the feedback survey. Thanks!

Overview

This week's assignment involves visual representations of database schemas using the Entity-Relationship model. Similar to the midterm, you will need to turn in a hard copy of your diagrams, but you can submit the relational model and SQL portions of the assignment on Moodle using the typical format.

For questions that involve drawing diagrams, you can use a program such as Visio, PowerPoint, xfig, dia, etc. Or, you can turn in hand-drawn diagrams. **This is the most important guideline: Regardless of how you create your diagrams, they must be legible!** If your work is difficult to read, you will lose points. If you can't draw reasonably straight lines then use a ruler, or do your drawing on the computer.

For questions involving SQL, please submit them in the usual format. Template files are provided for these questions. Submit these files in a tarball or zip-file called **`cs121hw5-`*`username`***.

Please follow the unique-role assumption[1] as much as possible. Note that in some ways this is "built into" the E-R model, since relationship-sets implicitly include the participating entity-sets' primary keys, and so forth.

Airline Database *(100 points)*

For this problem you will be creating a database schema to record some basic information for an airline, such as what flights are available, what airplanes are used on each flight, and what tickets have been purchased. We will approach this problem in stages to keep it from becoming unwieldy; at the end, you will assemble the various parts into a complete database system.

This problem will give you some design guidance, but there are many places where it will not. You must figure out what mapping cardinalities and participation constraints should be enforced by the database, based on the descriptions below, as well as your intuitions of what would "make the most sense" given the problem domain.

Problem 1: Flights and Airplanes *(15 points)*

For this problem you must create an E-R diagram that can record what flights are available, what kind of airplane is used for each flight, and what seats are available on each kind of

[1] If you have forgotten what this is, it means that every attribute name in the schema has a unique meaning. If we use the same attribute name in two tables, it's because the two attributes mean the same thing across both tables.

plane. Your diagram should represent flights, kinds of aircraft used for flights, and the seats available on those flights. It is not necessary to create other kinds of entities besides these. Additional details are given below.

Flight Information

Each flight has a flight number (a short string, e.g. "QF11" or "QF108"), a date (e.g. "2007-05-21"), and a time (e.g. "14:10:00") associated with it. A given flight number will be reused on different days, but the combination of flight number and date will be unique. You can see how it would definitely be best to keep the date and time values separate in this schema.

Flights also have a source and destination airport, which should be represented by their 3-letter International Air Transport Association (IATA) airport code[2] (e.g. "LAX" for Los Angeles International Airport, or "SYD" for Sydney Airport).

> *(One could imagine creating another entity-set to represent airports, but you definitely do not need to do this for the assignment.)*

There should also be a way to mark a flight as domestic (within the country) or international (between two countries). The reason for this is that travelers must provide additional information when on an international flight.

Airplane Information

Each flight also has a certain kind of aircraft associated with it. This allows customers to reserve specific seats for the flight. Each kind of aircraft is represented by several pieces of information:

- The manufacturer's company (e.g. "Airbus" or "Boeing")
- The aircraft's model (e.g. "A380" or "747-400")
- The IATA aircraft type code,[3] a 3-character value that the International Air Transport Association uses to specify the kind of airplane that a flight uses. The value is unique for every kind of aircraft. For example, the Airbus A380 is designated by the code "380", and the Boeing 747-400 is designated by the code "744". This value can include letters and numbers.

Along with each type of aircraft the company has, information about the seats available on that aircraft must be available. Individual seats have these details associated with them:

- A seat number such as "34A" or "15E" – the numeric component specifying the row of the seat, and the letter specifying the position within the row. (You can represent seat numbers as a single field; there's no reason to break the values apart.)
- A "seat class" such as "first class", "business class", or "coach". (Different classes can be represented as 1-character codes, or as strings, or whatever you prefer.)
- A "seat type" specifying whether the seat is an aisle, middle, or window seat
- A flag specifying whether the seat is in an exit row

---

[2] http://en.wikipedia.org/wiki/International_Air_Transport_Association_airport_code
[3] http://www.airlinecodes.co.uk/arctypes.asp

Obviously, each seat number will be unique on a specific kind of aircraft (i.e. on the 747-400, "17B" will specify exactly one seat), but different kinds of aircraft will definitely have seats with overlapping seat numbers. This suggests that it would make sense to represent seats as a weak entity-set.

Problem 2: Airline Customers *(15 points)*

Next you must create an E-R diagram to represent customers of the airline. Customers fall into two categories – "purchasers," who are responsible for buying tickets, and "travelers," who actually do the traveling. These sets of customers can certainly overlap, but they are not required to: people may buy tickets for their own use, and/or they may also buy tickets for other people. You can see that some kind of generalization hierarchy would likely be a useful approach for this schema.

Note that for this problem, you are not specifying how to represent tickets. Tickets turn out to be a bit tricky, so you will do this in Problem 3, when you assemble the various parts of the database schema together.

Purchasers and Purchases

As stated before, "purchasers" are the people who purchase tickets for specific flights. Note that purchasers might buy tickets for people other than themselves, such as a department administrator buying flights for other people in the department. Purchasers have the following information:

- A first and last name (should be represented as separate attributes in the database)
- A contact email address
- One or more contact phone numbers
- Optional payment information, including a 16-digit credit card number, expiration date (MM/YY), and 3-digit card verification code. This data may all be *null* if the purchaser doesn't trust the airline to properly secure this data… ☺

A "purchase" is a collection of one or more tickets bought by a particular purchaser in a single transaction. (As stated earlier, don't worry about incorporating tickets at this point; we will get there.) A purchaser can make multiple purchases. A purchase includes the following information:

- An integer ID that uniquely identifies the purchase
- A timestamp specifying when the purchase occurred
- A six-character "confirmation number" that the purchaser can use to access the purchase. In the real world, confirmation numbers are not guaranteed to be unique, but we will simplify our database design by enforcing that they are unique across all purchases in the database. (If we take our confirmation numbers from the capital letters A-Z and the digits 0-9, we can represent about 2.1 billion purchases before we start having trouble.)

Travelers

Travelers are the people who are actually going on a particular flight. Travelers have some similar attributes as purchasers:

- A first and last name (should be represented as separate attributes in the database)
- A contact email address
- One or more contact phone numbers

Travelers do not have payment information.

As mentioned before, some flights are domestic, and other flights are international. For international flights, travelers must provide these additional details:

- A passport number. The length of the passport number varies from country to country, but it is usually pretty long, so provide up to 40 characters for this value.
- The country of citizenship for the passport.
- The name of an emergency contact (the first and last name can be together in a single field, if you wish).
- A single phone number for the emergency contact.

Travelers are not required to provide these details immediately; they only need to be entered at least 72 hours before the flight. Therefore, it is perfectly acceptable to allow *null* values in the above fields.

Finally, this airline has a frequent-flyer program for travelers to participate in. Therefore, you should also include an attribute to hold the traveler's frequent flyer number, if they have one. Although we call it a "frequent flyer number," the value is actually comprised of both numbers and letters, and is always 7 characters long.

Problem 3: Bringing Everything Together – Tickets! *(15 points)*

At this point you should have an E-R diagram representing kinds of aircraft, flights and seats, and another E-R diagram representing travelers, purchasers and purchases. The thing that ties these two parts together is the ticket. For this problem you will create a complete E-R diagram representing the entire database schema, with tickets tying together the concepts of purchases, travelers, flights and seats. You will need to figure out the appropriate relationships, mapping cardinalities and participation constraints to satisfy the description below, as well as whatever other constraints would simply "make sense" to include in the system.[4]

Tickets themselves are very simple:

- Each ticket should have a unique ID (unique across all tickets ever issued by the airline, not just unique to a specific flight).
- Each ticket should also store the sale price of the ticket. Airlines frequently vary the prices of seats on a particular flight, both by where the seat is located (e.g. first class or coach), as well as how close the purchase-date is to the flight-date. The schema must be able to represent this variation in pricing. You should assume that tickets will always cost less than $10,000 each on this airline.

[4] An example of this that we have discussed all term, is that every loan must have at least one customer associated with the loan; otherwise, our bank won't be in business for very long. You must think of similar considerations for our airline.

What is much more complicated is the relationship between tickets and the other entities in our database. Here are the relevant details:

- A purchase may include one or more tickets.
- A purchase may include multiple tickets for the same traveler, e.g. for a multi-leg flight. Of course, a purchase may include tickets for multiple travelers as well. In other words, we really don't want to constrain what tickets appear in the purchase, just that there must be at least one ticket in each purchase.

- Each traveler will have one or more tickets associated with them.
- As with purchasers, we really don't want to impose any other constraints between tickets and travelers. For example, a traveler can even hold multiple tickets on the same flight (sometimes necessary for very large/fat people, or people traveling with pets or small children). So again, we don't want to impose too many constraints between travelers and tickets, just that each traveler will have at least one ticket.

- Each ticket represents a particular *seat* on a particular *flight*. It should not be possible to associate a specific ticket with multiple seats, or with multiple flights, in the database.
- Similarly, it should only be possible to give each ticket to one traveler.
- These two constraints together should guarantee that when we give a ticket to a traveler, they won't have to share that seat on the flight with anybody else… an uncomfortable prospect at best!

Problem 4: Translation to Relational Model *(15 points)*

Once you have completed your entire E-R diagram in Problem 3, translate it into a relational model schema. Write up your schema in the file **`airline-schema.txt`**. For each relation schema, make sure to identify any candidate keys besides the primary key, and also any foreign keys and what they reference.

If you perform any schema combinations then you must explain why it is a good idea to combine the schemas.

Problem 5: Translation to SQL DDL *(25 points)*

Next, translate your relational model schema into SQL DDL. Your DDL file needs to load successfully into MySQL. Write your DDL in a file called **`make-airline.sql`**. At the top of your file you should also include **`DROP TABLE`** commands so your schema can be reloaded easily. (Your **`DROP TABLE`** commands must respect referential integrity.)

Here are additional guidelines:

- Specify all relevant not-*null* constraints, default values where appropriate, and make sure to include all primary / foreign / candidate key constraints.
- Specify cascade operations where there are existence-dependencies between records in the database schema.
- Your DDL must be clearly documented. Every table should have at least *some* documentation explaining its contents, particularly tables that might have critical meaning within the database schema. Any fields whose meaning isn't completely

obvious should also include some explanatory comments. If you have cascade operations or other unusual constraints (e.g. multi-column primary or foreign keys, **`CHECK`** constraints, etc.) then briefly comment these as well.

- Make sure to choose reasonable column types and size/precision limits.

Problem 6: Use-Cases and SQL Queries *(15 points; 5 points per query)*

Here are some common use-cases that might need to be performed against the database schema. Write the SQL queries for these operations to see how easy (or hard) they are to do against your schema. Put your answers in the file **`airline-queries.sql`**.

a) We need to provide a way to display all the purchase history for a single customer via the company website. To demonstrate how this will work, write a SQL query that will retrieve all purchases and associated ticket information for the customer (aka "purchaser") with ID 54321. The results should be ordered by these columns: purchase date (descending order), flight date, traveler last name, traveler first name (all other columns ascending order).

b) Write a query that reports that total revenue from ticket sales for each kind of airplane in our flight booking database, generated from flights with a departure time within the last two weeks. Include *all* kinds of airplanes in the database, whether they were used for flights in the last 2 weeks or not.

c) Write a query that reports all travelers on international flights that have not yet specified all of their international flight information.