Milestone 4: Schema Diagram and SQL

CPS 353: Internet Programming
Web Development Project
Due Wednesday, October 9, 2013 by the start of class

Introduction

In this milestone, you will create a schema diagram for the take out order system's database. You will also write files containing the SQL statements necessary to create and drop the database tables and constraints used by your web application.

Specifications

Each entity in the system is described below. You will use the requirements and details that follow to fashion your schema diagram and SQL files.

User

The user entity holds data about each user registered with the system. A user can have multiple friendships and sessions. Furthermore, a user can participate in numerous orders, both as an organizer and regular participant. The user entity has the following fields, all of which are required.

- An integer id assigned automatically and incrementally by the system to uniquely identify the user.
- The dates and times at which the user was created and last modified.
- The user's name, email, and account password – all text values of up to 50 characters in length.
- The user's role – a text field of up to 20 characters.

Friendship

The friendship entity connects two different users together, enabling them to participate in a single order. It makes no sense for a friendship to exist apart from either of its corresponding users. The friendship entity has the following fields, all of which must be set.

- An integer id assigned automatically and incrementally by the system to uniquely identify the friendship.
- The date and time at which the friendship relationship was created.
- The integer ids of the two users (i.e. user_id and friend_id) connected by the friendship.

Sessions

The session entity contains data about user visits to your web application. Each user may make several visits to the app, and thus may have multiple sessions. A session cannot exist apart from its associated user. The session entity has the following fields, all of which are required.

- An integer id assigned automatically and incrementally by the system to uniquely identify the session.
- The dates and times at which the session was created and last modified.
- The integer id of the user that the session belongs to.
- A unique string token of up to 100 characters. (This token will be used allow the client-side browser to keep track of the user's current session between requests to the application.)

Restaurant

The restaurant entity stores data about a given restaurant. A restaurant can have many orders associated with it, and includes the following details.

- An integer id assigned automatically and incrementally by the system to uniquely identify the restaurant.
- The dates and times at which the restaurant was created and last modified.
- The restaurant's name, which can be up to 50 characters long.
- The type of cuisine served by the restaurant, which is a string of text of up to 20 characters.
- A description of the restaurant that can be several paragraphs long. The description has a variable length, but generally will be *longer* than other text fields for the restaurant.
- Fields to contain the components of the restaurant's address, including the following:
    - Two text fields of up to 100 characters apiece for the restaurant's street address. Only the first of these street address fields needs to be specified.
    - The restaurant's city, which may be up to 50 characters long.
    - The restaurant's state abbreviation, which is a character field *with a fixed with of exactly* 2 characters.
    - The restaurant's zip code, which is a text field of up to 10 characters. (Zip codes can sometimes contain dashes.)
- The restaurant's (required) phone and (optional) fax numbers, each of which are stored in text fields up to 20 characters long (to account for different phone formats, extensions, etc.).
- The URL of the restaurant's website, which can be up to 100 characters long.
- A Boolean flag indicating whether or not the restaurant delivers.
- The delivery fee that the restaurant charges for its delivery service. This value is stored as a floating-point number.
- The path to a file containing the restaurant's menu. This file path is stored as a text string of up to 100 characters long. (Users will eventually be able to upload menu files for the restaurants they create.)

The restaurant's id, created and last modified timestamps, name, cuisine type, first line of street address, city, state, zip code, phone number, and URL must be set.

Order

The order entity contains information about a take-out order being placed by a group of users (a.k.a. participants) at a given restaurant. An order can have multiple participants, one of which is the organizer (that is, the user who initiated the order). An order is also associated with a single restaurant. Note that it is perfectly reasonable for users and restaurants to exist apart from any particular order. An order has the following fields, all of which must be set.

- An integer id assigned automatically and incrementally by the system to uniquely identify the order.
- The dates and times at which the order was created and last modified.
- The integer id of the restaurant at which the order is being placed.
- The integer id of the user who is organizing the order (e.g. organizer id).
- A string of up to 20 characters indicating the type of the order (i.e. pick-up, delivery).
- The total cost of the order as a floating-point number.
- The date and time at which the order will be placed at the restaurant. (Participants will have until this point in time to specify any line items they wish to include on the order.)
- The current status of the order (such as pending or placed) stored as a text field of up to 20 characters.

Participant

A participant is an entity representing a user who is participating in a group take-out order. Each user taking part in an order is considered a participant in that order, regardless of whether he or she is organizing the order or is just taking part in it. An order can have numerous participants, and a user can participate in multiple orders. The idea of a participant doesn't make sense without corresponding user and order entities. The participant entity has the following fields, all of which are required.

- An integer id assigned automatically and incrementally by the system to uniquely identify the participant.
- The dates and times at which the participant was created and last modified.
- The integer id of the user entity that the participant is associated with.
- The integer id of the order of which the participant is part.
- The role the participant is playing in the order (i.e. organizer, participant) stored as a text value of up to 20 characters.
- The total cost of the line items the participant is purchasing on the order stored as a floating-point number.

Line Item

The line item entity represents a single item a participant is purchasing as part of a take-out order. Each participant can specify several line items for a given order, and each order can have multiple line items. A line item cannot exist apart from its associated participant and order. The participant entity has the following fields.

- An integer id assigned automatically and incrementally by the system to uniquely identify the line item.
- The dates and times at which the line item was created and last modified.
- The integer id of the participant entity that the line item is associated with.
- The integer id of the order of which the line item is part.
- The name of the line item as a text field of up to 100 characters.
- The price of the line item stored as a floating-point number.
- A field to contain notes about the line item (i.e. specific preparation instructions, toppings to include/exclude, etc.). This text can be variable in length and will generally be *longer* than a normal text field.

All of a line item's fields except the notes must be set.

Technical Requirements

Schema Diagram

Create your schema diagram according to the conventions discussed in class.

- Each table (entity) should have its own box.
- The table's name should appear at the top center of the box and be separated from column (field) names by a horizontal line beneath it.
- Column names should be printed in a vertical list within the table's box.
- The names of primary key columns should be underlined.

- A relationship between two tables should be represented with a line between the tables. Arrowheads should be drawn at the end of this line as appropriate to the cardinality of the relationship. Specifically:
    - A one-to-one relationship should have arrowheads at both ends of its line.
    - A one-to-many/many-to-one relationship should have an arrowhead at the end of the line pointing to the "one" entity.
    - A many-to-many relationship line should have no arrowheads.
- Columns with foreign keys should be recognizable as such, either by including the text "(fk)" after their names. Alternatively or additionally, the line representing the relationship that a foreign key column participates in can start or end at the column name.
- As a convention, fields containing timestamp values should end with the word "at" (i.e. created_at, modified_at). This helps connote that the data in the field is an actual date and time. (For instance, "User A was 'created at' 3pm on 9/25/13.")
- You may draw your schema diagram by hand and scan it into electronic form, or you may use a word processor or drawing program to create it.

SQL

Create two files – *create_tables.*sql and *drop_tables.sql* – containing the SQL statements necessary to create and drop your tables, respectively. Observe the following guidelines when constructing your SQL statements.

- Use the plural form of an entity's name for its table (i.e. a *users* table holds user entities).
- Each table must have a primary key defined on it.
- Columns whose values are required should have not null constraints declared on them.
- A column containing unique values (other than a primary key) should have a unique constraint declared on it.
- Two tables connected by a relationship should have an appropriate foreign key declared between their connecting columns. Generally, the table containing entities that "belong to" entities in the other table has the foreign key defined on it.
    - If entities in one table cannot exist apart from entities in the other table, the foreign key should include a "delete cascade" clause that indicates that when an entity that owns the relationship is deleted, its corresponding weak entity (in the table with the foreign key constraint) should also be removed.
- Note that the order of your create table and drop table statements will be important. For instance, a table on which a foreign key constraint is to be defined needs to be created *before* the table that declares the constraint. Similarly, a table defining a foreign key constraint must be dropped *before* the table against which the constraint is declared.

Your *create_tables.sql* and *drop_tables.sql* files must execute correctly and without errors or warnings when run against your project database. Use the following commands on ips.cs.gordon.edu to test the execution your SQL files.

```
mysql -u <username> -p <database name> < create_tables.sql
mysql -u <username> -p <database name> < drop_tables.sql
```

In these commands, *<username>* is your user name on the ips.cs.gordon.edu machine, and *<database name>* is the name of your project database (named "project_*<username>*").

Note that since this milestone does not involve your application's user interface, you do not need to include files from your previous milestones as part of your submission.

Turn In

Submit the following items to the professor via email:

- An electronic copy of your schema diagram as a PDF document.
- Copies of your working *create_tables.sql* and *drop_tables.sql* files.