**Hackers Database Design Doc:**

* userID (Primary Key)
* company
* email
* latitude
* longitude
* name
* phone
* picture

users

* eventID (Primary Key)
* eventName (Unique)

events

* eventID
* userID

Unique(eventID, userID)

userEvents

I ended up splitting the user data into 3 sql tables.

* The first table is ‘users’, which only stores the userID, company, email, latitude, longitude, name, phone, and picture url of that user. The ‘users’ table does not have any event information
* The second table is ‘events’. This table contains all of the events and their ID’s (there are 10 events in the JSON provided). This table contains no user information
* The third table is the ‘userEvents’ table which contains rows of eventID and userID pairs. This table maps every user in the JSON using their userID to the events they are attending.

**Some Notes:**

For the users table, I made the ‘userID’ a primary key. This is to provide a unique identifier for every user.

For the events table, the ‘eventID’ is a primary key, and the ‘eventName’ is Unique. The eventID is a unique identifier for an event in the database, and the eventName is Unique because there should not be a duplicate event in the database (this is in case an endpoint was created in the future to allow people to add events to the events table).

For the userEvents table, there is a Unique(eventID, userID) constraint. This is to ensure that when someone hits the /events/<id>/attendees endpoint for a POST request, they don’t place in a duplicate userEvent mapping into the table. This constraint was added because I did not have enough time to implement a proper primary-foreign key relationship between the three tables, so in order to reach a compromise, I added the Unique constraint which works for the purposes of the endpoints required. This is an improvement that needs to be made for a future release of the API.