Smart Group Financial

Project Design

CS 157A by Team 31

Christian Castro

Cuong “Calvin” Nguyen

Pranika Bedi

Professor Mike Wu

Oct 15th, 2019

Project Overview

Problem Statement:

When you're on a trip with a group of friends, Person A spends $95 on food, Person B spends about $50 on gas, and Person C spends $100 on amenities. Then, it will be significantly time-consuming to figure out how much one should pay back to another. Some groups use Excel, but they have to create their own formula, which might result in mistakes and take even more time to fix and re-calculate again. Additionally, there is a possibility that they may request the wrong amount of money. This can lead to problems with trust issues, misunderstandings, and uneven money distribution. As a result, we want to create an application that will not only solve these issues but also provide everything needed to manage a trip on one platform.

Stakeholders: This app focuses on any travel group

Project Description:

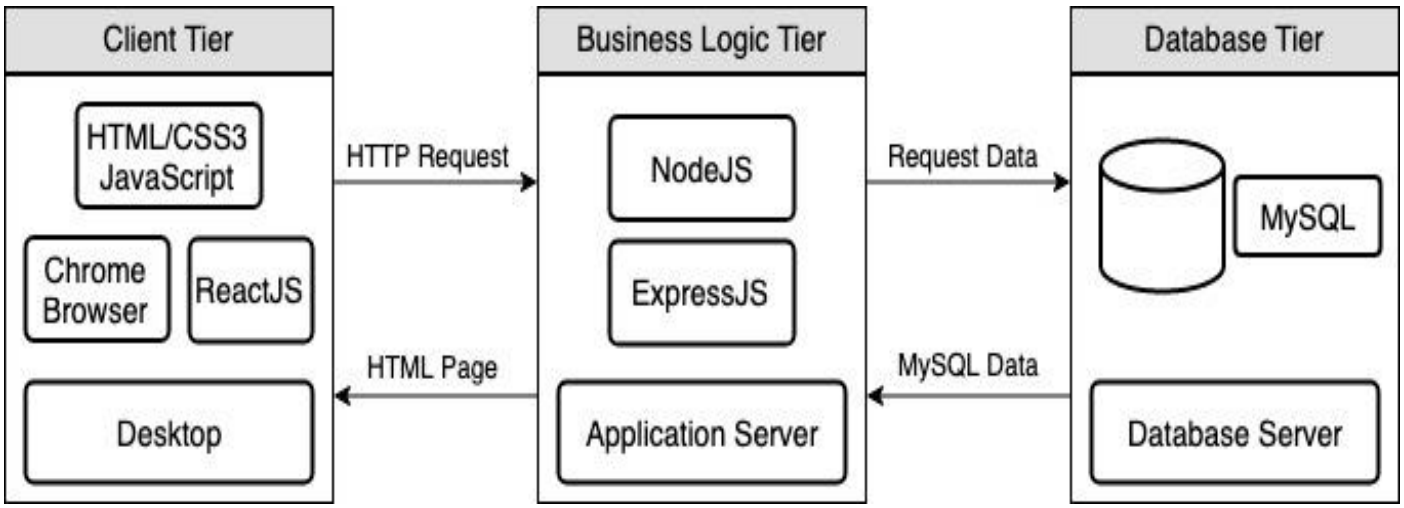
We want to develop a database application that will serve as a platform for various groups to manage their events, event members, and finances involved with the event. The app will allow users to create different parties/trips/planners, invite other users of the app to be apart of the event, and enter the amount of money that each person has contributed to the event. Finally, the app will assist in calculating the total amount spent on the event, and then generate the amount of money that one person in the group may owe another. This will in effect create a way to fairly distribute the expenses used for the event. This application will also have a user-friendly interface that will allow users to easily take advantage of these properties.

Possible Data Model:

Entities

1. User (first name, last name, like a profile, etc., 1 user can create/own many events/trips)
2. Events: id, name, location, time, 1 event will contain a list of users participate in that specific event, multiple categories, a history of who changes the price
3. Categories: (Title, amount of money)
4. Groups: store a list of users, can store many events
5. History of events: List of users, and event's information

System Environment



HW/SW: Desktop Web Browser, Mac, Windows, Linux

Overview:

The architecture of our application is based on a typical MVC model. Our Client tier (View) will be written in Javascript, HTML, and CSS, using ReactJS as the framework. This level of the architecture is what the user will interact with to access the features of our application. The Business Logic Tier (Controller) will be written using NodeJs and ExpressJS, and this tier represents the Application Server that will act as the bridge of communication for the Client Tier and Database Tier. This tier will serve HTML pages to the user's device and accept HTTP requests from the user and follow with the appropriate response. Our Database Tier (Model) will be hosting MySQL as our application's Relational Database Management System. This is where we will store all of the crucial data our application needs to function.

Functional Requirements

- **Any user is able to login/register/logout an account**
    - Goal:
        - A new user can register an account with our app or log in with their Facebook/Google credentials

    - Functional Processes:
        - Input: Email/Password
        - Output: Redirect to the user dashboard/homepage
    - Prerequisite:
        - User should be able to make a new account with active email, Facebook, or Google credentials
        - User should be able to log in with correct email and password combination
    - Post-Condition:
        - The user will be redirected to the dashboard/homepage where they will be able to see options to view their profile, groups, and events.
    - Exceptions:
        - If the user inputs the wrong email/password combination, an error message will be displayed.
- **User is able to create any events**
    - Goal:
        - Registered user clicks a “Create Event” button to create a new event/trip
    - Functional Processes:
        - Input: User will log in to the web app, and click on the “Create Event” button. Fill in the form with the event: name, date, location, and description.
        - Output: The page will be redirected to the new event page where they can start adding users and money they spend on the trip
        - Const express = require(‘express’);
    - Prerequisite:
        - User is logged into their account
        - User completely fills out the form for event information
    - Post-Condition:
        - Event will be created and added to the database
        - User will be redirected to result page for new event
    - Exceptions:
        - If the user does not completely fill out Event information form an error message will be displayed
- **User is able to create groups and add other users to that group**

- Goal:
  - User can click on “Create Group” and add other users within that group for an event/trip
- Functional Processes:
  - Input: User logs into the application, clicks on the button to create a group, and inputs other usernames to add within the group.
  - Output: An invitation to others will be sent which they can accept and join the group.
- Prerequisites:
  - User is logged into their account
  - Users that group created is trying to add exist in the system’s database (Tracked by email address)
- Post-Condition:
  - New group is added to the database and all affiliated users are tied to it
- Exceptions:
  - If the user tries to invite a member that does not exist and error message will be thrown
  - If the user is not logged in to their account they will not have access to the “Create New Group” menu

- **User is able to add a category that he/she pays within those events**
  - Goal:
    - User enters the category of what they pay for, then the app will calculate and display the right amount for each person in the events.
  - Functional Processes:
    - Input: The category that they pay for (food, gas, etc.) and the amount of money they pay. Users can input specifics, such as locations, restaurant names, etc.
    - Output: A real-time update to display the charges for each person.
  - Prerequisite:
    - User is already a member of an event
    - Creator of the event is notified and must verify payment
  - Post-Condition:

        - New payment information will be applied to the event and database will be updated
    - Exceptions:
        - If event creator does not verify payment it is voided
- **User is able to see a history of changes in the events**
    - Goal:
        - User can refer back to details (group members, total, location) of past or favorite events
    - Functional Processes:
        - Input: User presses the “History” button
        - Output: List of all user’s past events and that event’s info
    - Prerequisite:
        - User has past events that can be displayed
    - Post-Condition:
        - User will be redirected to a History page that displays a list of all the past events they were apart of
    - Exceptions:
        - If the user was not previously apart of any events they will be shown an empty result page
- **User is able to sort their list of events by time**
    - Goal:
        - User will be able to sort their list of events in chronological order
    - Functional Processes:
        - Input: On the History page, user will click drop down sort button and choose how they want the list sorted
        - Output: A sorted list will be presented to the user
    - Prerequisite:
        - There are a list of events displaying on the UI
    - Post-Condition:
        - The list will be sorted based on the time it is created
    - Exceptions:
        - If the user has no history of events there will be nothing to sort
        - If the user only has one event listed the sort function will have no effect on the list

Non-functional Requirement

Implementation

- Develop a responsive web application in HTML/CSS/JavaScript, and ReactJS library for real-time update
- Host and language the database/application server on Google Cloud Platform

Usability

- The application shall be displayed in English.

Reliability

- Support on major internet browsers such as Google Chrome.

Performance

- Respond to the user's action and query data within 5 seconds.

Security

- Use JWT to authenticate and check for the user's identity to protect users' privacy

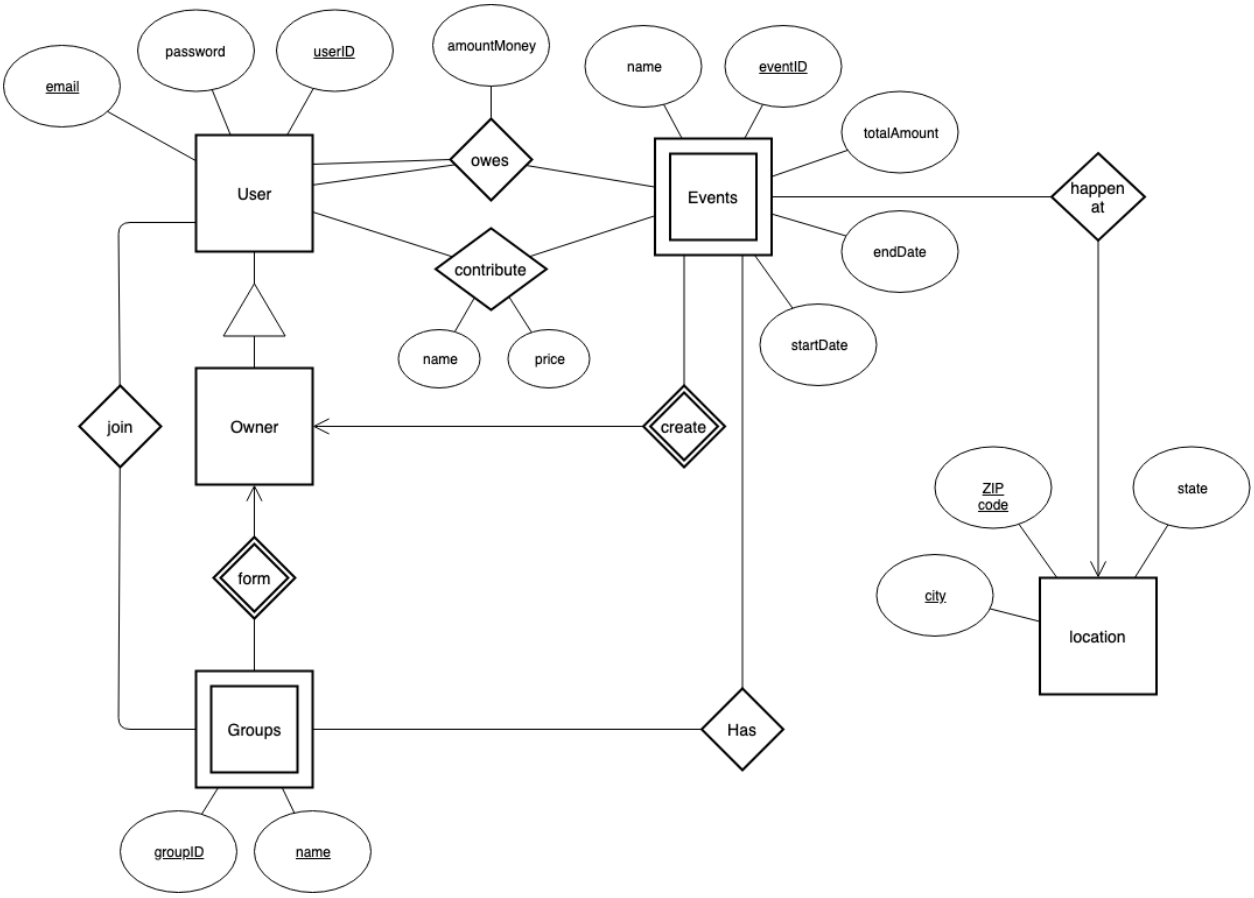
Scalability

- Multiple users will be able to access the application's database without any latency issues
- The application will be able to serve multiple users at once

Packaging

- Application will be available to use on these platforms without any additional setup:
    - Windows and Mac

Project Data Model



Entity Sets

1. **User**

The user has attributes password and email. This stands as the user's login information for the application.

2. **Owner**

Owner and User have a ISA relation, which implies that the owner is a user.

3. **Groups**

Groups is a weak entity set which means a primary key must exist in order for it to be distinguished from another group. Each group has a unique name and ID.

4. **Events**

Events is also a weak entity set and has a primary key, eventID. It also has other attributes, such as name, endDate, and totalAmount.

5. **Location**

Location is an entity set that has attribute, state, and keys, ZIP code and city. A location can be distinguished by its unique zip code and city name combination.

Relations between Entity Sets

1. **User owes the user an amount of money for a specific event**

The user has the option to request/charge for a certain amount of money from an event. The application will save this request in the person's profile. However, the application does not support transactions.

2. **Owner is a user who can form groups**

The user can form groups by adding and deleting other users. A user with "owner" status is the only member of the group that has access to such actions.

3. **Owner is a user who can create events**

The user has the ability to create events that pertain to various locations, date range, name, ID, and email.

4. **User has groups**

A user can be a member of multiple groups. The user can view their groups in their dashboard. Groups can be distinguished from one another by a unique ID and name.

5. **Groups have events**

After a group is formed, the users have the option to create an event within the group. This event is accessed by all members of the group and can proceed to creating categories within the event.

6. **User contributes to events**

Within an event, there can be a variety of categories that the user adds. Each category within the event has a name and price which represents the total of each sub category. As a result, one event will contain attributes, such as name, endDate, and resulting totalAmount, as well as primary key, eventID.

7. **Events happen at location**

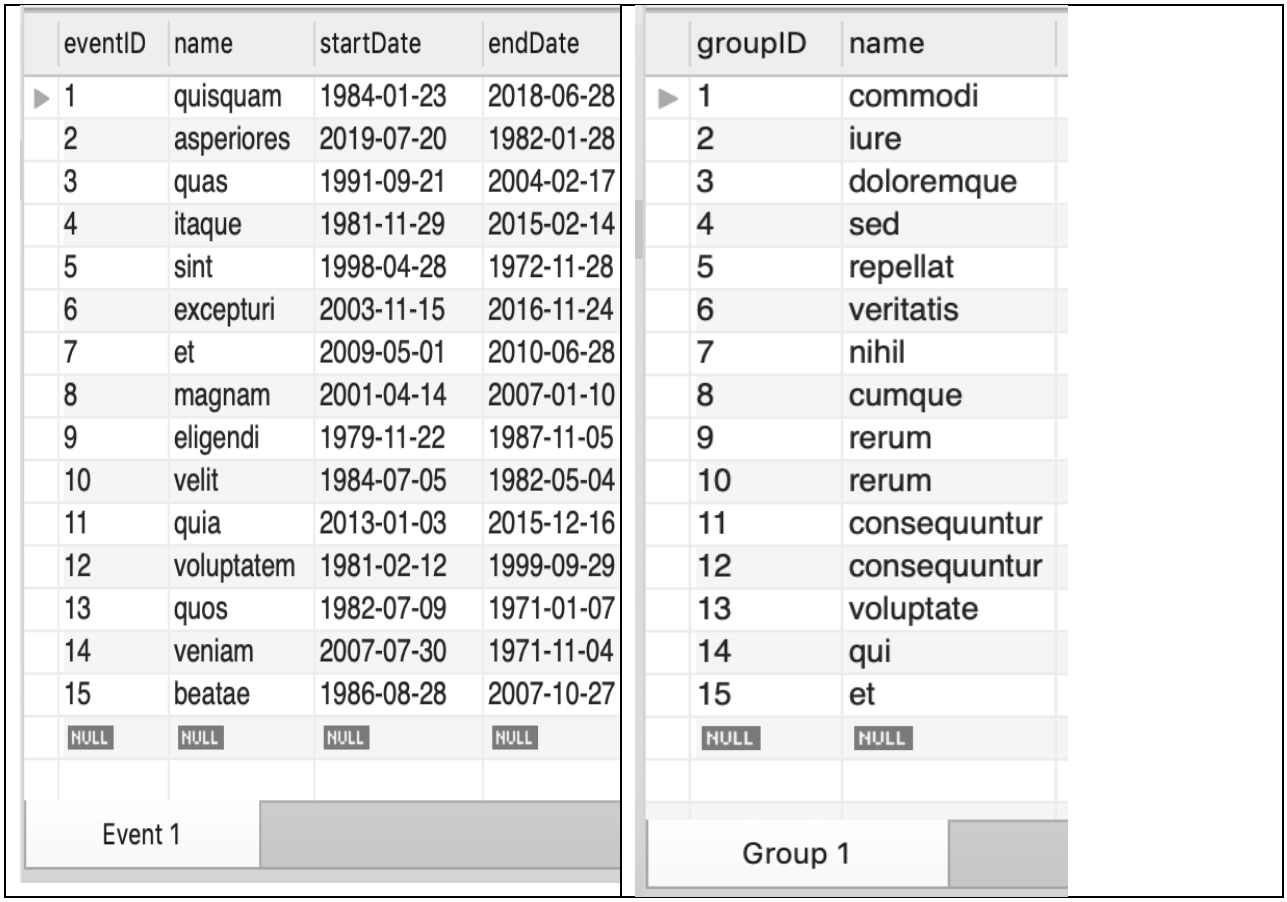
Events is a way for users to track their trip efficiently. Consequently, it is important to distinguish these events by location. Each event corresponds to one particular locationID.

Scheme Entity Sets and Relations

User (UserID, email, password)
Groups(groupID, name)
Location(city, ZIPcode, state)
Events(name, eventID, totalAmount, endDate, startDate, locationID)

Owner-Create-Event(userID, eventID)
Group-Has-Event(groupID, eventID)
User-Contribute-Event(eventID, userID, category, amount)
Owner-Form-Group(userID, groupID)
User-Join-Group(userID, groupID)
User-Join-Event(userID, eventID)
User-Owes-User(eventID, amountMoney, recipientID, payerID)

Sample Database Tables



eventID	name	startDate	endDate
1	quisquam	1984-01-23	2018-06-28
2	asperiores	2019-07-20	1982-01-28
3	quas	1991-09-21	2004-02-17
4	itaque	1981-11-29	2015-02-14
5	sint	1998-04-28	1972-11-28
6	excepturi	2003-11-15	2016-11-24
7	et	2009-05-01	2010-06-28
8	magnam	2001-04-14	2007-01-10
9	eligendi	1979-11-22	1987-11-05
10	velit	1984-07-05	1982-05-04
11	quia	2013-01-03	2015-12-16
12	voluptatem	1981-02-12	1999-09-29
13	quos	1982-07-09	1971-01-07
14	veniam	2007-07-30	1971-11-04
15	beatae	1986-08-28	2007-10-27
NULL	NULL	NULL	NULL

Event 1

groupID	name
1	commodi
2	iure
3	doloremque
4	sed
5	repellat
6	veritatis
7	nihil
8	cumque
9	rerum
10	rerum
11	consequuntur
12	consequuntur
13	voluptate
14	qui
15	et
NULL	NULL

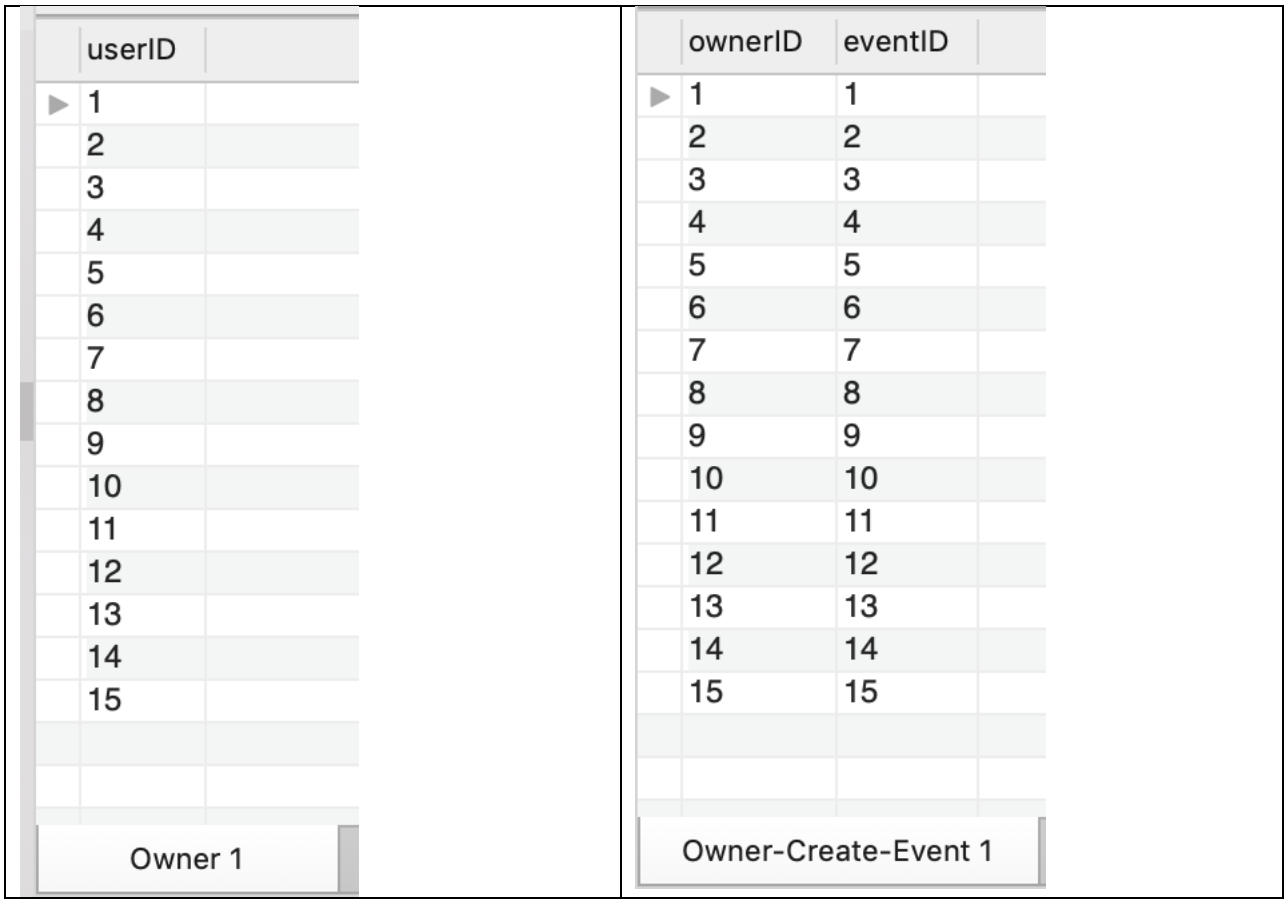
Group 1

eventID	groupID
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11
12	12
13	13
14	14
15	15

Group-Has-Event 1

city	zipCode	st
Champlinberg	45008497	C
Davisville	0	O
Hoegerchester	5573207	W
Lake Leon	24	In
Lebsackport	913160	K
New Christophertown	909	Fl
North Kathleenview	149	N
Ollieborough	85735	U
Reichertton	895912506	R
Schoenland	23998	M
South Braedenview	5696	W
South Chasitymouth	0	H
Stephanton	353405003	K
West Aishaland	50558	Io
West Beth	765374888	A
NULL	NULL	N

Location 1


userID
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
Owner 1
ownerID
eventID
1 1
2 2
3 3
4 4
5 5
6 6
7 7
8 8
9 9
10 10
11 11
12 12
13 13
14 14
15 15
Owner-Create-Event 1

	userID	email	password
▶	1	calvin@imi.ai	123456789
	2	christian.p.castro@sjsu.edu	9VetyUW5pZ4ZD4f
	3	christian@sjsu.edu	9VetyUW5pZ4ZD4f
	4	edward@gmail.com	123456789
	5	email@email.com	password
	6	jason	123456789
	7	mike@wu.com	123456789
	8	mike@wu.com1	123456789
	9	patient@imi.ai	123456789
	10	phu@gmail.com	123456789
	11	pranika@gmail.com	123456789
	12	test@sjsu.edu	$2a$05$nj4gVwlz…
	13	jon@gmail.com	password123
	14	karen@gmail.com	password456
	15	max@gmail.com	password789
	16	cuongvqnguyen@gmail.com	$2a$05$6lYwFAAh…
	17	cuongvqnguyen@gmail.com	$2a$05$lyUjpX674…
	18	test@gmail.com	$2b$10$DWLIZZLg…

user 1

eventID	userID	category	amount
1	1	Gas	10
1	2	Food	10
1	3	Food	10
1	4	Drinks	10
1	5	Equipment	10
1	6	Snacks	70
2	2	Paper	10
2	7	Glue	10
2	8	Staplers	10
2	9	Staples	10
2	10	Scissors	60
3	3	Pizza	10
3	11	Water	10
3	12	Gatorade	10
3	13	Chips	10
3	14	Plates	10
3	15	Cups	70

User-Contribute-Event 1

userID	groupID
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11
12	12
13	13
14	14
15	15

Owner-Form-Group 1

userID	groupID
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11
12	12
13	13
14	14
15	15
2	1
3	1

User-Join-Group 1

receipientID	payerID	eventID	amount
1	2	1	30
2	2	1	20
3	2	1	60
4	3	1	10
5	1	2	32
6	4	2	35
7	5	2	450
8	2	2	32
9	4	4	3
10	1	10	130
11	2	10	309
12	1	11	301
13	3	11	240
14	5	12	387
15	1	10	31

User-Owes-User 1

	userID	groupID
	1	1
	2	2
	3	3
	4	4
	5	5
	6	6
	7	7
	8	8
	9	9
	10	10
▶	11	11
	12	12
	13	13
	14	14
	15	15

Owner-Form-Group 1