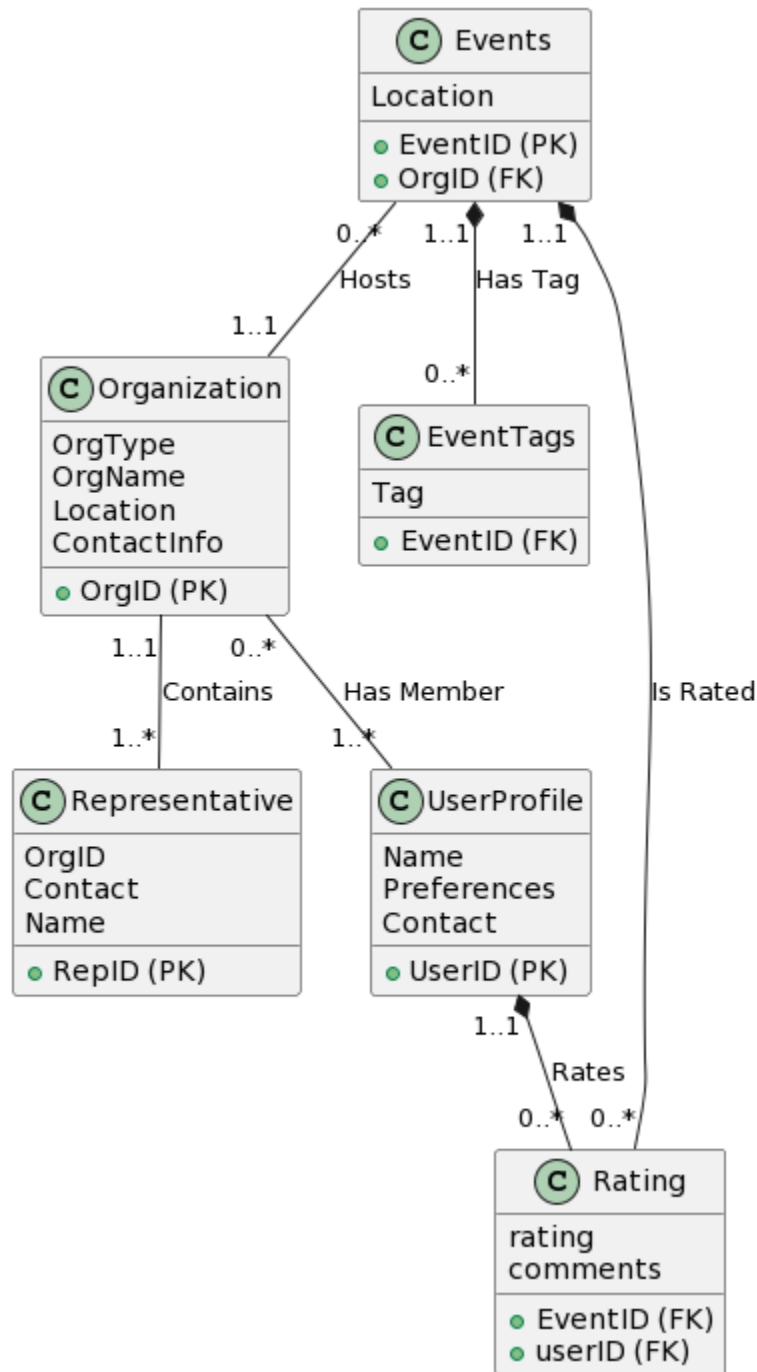


Database Conceptual Design

UML Diagram:



Entities (assumptions):

1. Organization:

- a. OrgID (INT)
 - i. Primary Key
 - ii. Unique Identifier of Organization
- b. OrgName (VARCHAR)
 - i. RSO/Fraternity/Social Body name
- c. OrgType (VARCHAR)
 - i. Organization category, like social, business, engineering, etc.
 - ii. There will be predetermined categories to select from.
- d. Location (VARCHAR)
 - i. Location (address)
 - ii. Can be NULL (no in-person location/office)
- e. ContactInfo (VARCHAR)
 - i. Email address or social media link
 - ii. Can be NULL

2. Representative:

- a. RepID (INT)
 - i. Primary Key
 - ii. Unique identifier of the representative of a particular Organization
 - iii. Assumption is that every organization will have one representative, and this representative exists outside of the 'UserProfile' space. The goal is to separate 'business' side from 'consumer' side.
- b. OrgID (INT)
 - i. Foreign key to Organization. Used to identify which organization being represented.
- c. UserID (INT)
 - i. Foreign key to UserProfile. Used to identify which particular user is representing the organization/
 - ii. Can be NULL (if particular representative doesn't have UserProfile)
- d. First Name (VARCHAR)
- e. Last Name (VARCHAR)
- f. Contact (VARCHAR): can be NULL

3. UserProfile:

- a. UserID (INT)
 - i. Primary Key: uniquely identify user profile
- b. First Name (VARCHAR)
- c. Last Name (VARCHAR)
- d. Preferences (VARCHAR): Event and Org Preference: can be used for recommendations.
 - i. Choose from predefined categories
 - ii. Can be NULL

4. Event:

- a. EventID (INT)
 - i. Primary Key: uniquely identifies event
- b. OrgID (INT)
 - i. Foreign Key to Organization. Assume each event can only belong to single organization. Single organization can have multiple distinct events
- c. Location (VARCHAR)
- d. Date (DATETIME)

5. EventTag: Weak Entity (dependent on EventID for primary key)

- a. Tag (VARCHAR)
 - i. Select from pre-defined options
 - ii. Exists as a csv (tag1, tag2...etc) for a given EventID

6. Rating: Weak Entity (Dependent on EventID and UserID as a composite primary key)

- a. Rating (INT)
 - i. Gives the rating for a given event
- b. Comments(VARCHAR)
 - i. Gives the comments made by a user for a given event as a CSV (comment1,comment2, etc.)
- c.

Relationships (cardinality):

We start at the UserProfile Table. Here, with primary key UserID, we can have each student in multiple organizations, and multiple students in the same organization (though a student can be in no organization and an organization must have at least one member), so a many-to-many relationship from UserProfile to Organization, from UserID to OrgID.

At the Organization table, we must have at least one representative from Representative for an organization, and each representative must have one organization they represent to be a representative (we are assuming multiple would be

an issue of juggling). So this relationship is one-to-many, from RepID and OrgID in Representative to OrgID in Organization. We see a similar relationship between Organization and Events: each organization can hold multiple events, and it would make sense that each event is held by one organization, so this is a one-to-many relationship between OrgID in Organization to EventID and OrgID in Events.

For the Events Table, there are other relations to keep in mind. Events, with its EventID and OrgID will have a one-to-many relation with EventID from EventIDs, since this is a mapping with the topics of the relation, and different topics can be pertained to. Then we look at the relationship between Events and Ratings: Each event can have multiple ratings for it so that relation is one-to-many.

We also note that UserProfile to Ratings has a one-to-many relationship, since each user can make multiple ratings.

Functional Dependencies and Justifications:

1. OrgId -> OrgType, Location, ContactInfo
 - a. Strong entity: uniquely identifies an organization so all the attributes in the Organizations table rely on the OrgId
2. UserId -> Name, Preferences, Contact
 - a. Strong entity: uniquely identifies a user so all the attributes in the Organizations table rely on the OrgId
3. RepId -> OrgId, Contact, Name
 - a. We only need the RepId since that will tell us information about the organization, and information about the representative. There's only one representative for one organization
4. EventId , OrgId -> Location
 - a. For a given event, which is unique, we still need to know what organization that event was associated with, so the OrgId is required.
5. EventId, UserId -> rating, comments
 - a. A single user can leave a rating on one event, and provide multiple comments too. However, we store the comments as a csv, so one record represents a single user's 'rating activity.' Therefore, everytime they have some new 'rating activity', a new record appears in the table. With the user Id, we can identify which user has had 'rating activities.' With the EventId, we can figure out which events the user has had 'rating activities' on

6. EventID -> Tag

- a. This uniquely identifies the different tags for an event that is going on, and is dependent on the Events table. One might ask why OrgId is not on the left hand side. This is because we don't care about the organization. We only care about the event and what tags it has.

7. EventId , UserId -> rating, comments

- a. EventId, UserId creates a primary Key, so it becomes a weak entity, but we don't need to list its primary keys, only show which strong entities it relies on.

- Memberships is a trivial Functional dependency. It highlights the relationship between. Organizations and UserProfile. An organization may have members, but a user does not have to be part of an organization. We could define this relationship as a combination of the OrgId and UserId which uniquely identifies a member of an organization, but has no other attributes apart from that. We prove it is a relationship, and hence a trivial functional dependency, by the fact that membership is a subset of the cross product of UserProfile and Organizations.

Normalization:

We choose to normalize using 3NF due to its simplicity (as BCNF is just a stricter form of 3NF) as well as attempting to avoid further decomposition.

Our normalized UML diagram is already in 3NF form.

We know it is in 3NF form because every relation has a primary key and all other attributes are dependent on the primary key. Additionally there are obviously no transitional dependencies.

Translated Relational Schema:

Organization (OrgID: INT [PK], OrgName: VARCHAR(255), OrgType: VARCHAR(55), Location: VARCHAR(255), ContactInfo: VARCHAR(100))

Representative (RepID: INT [PK], OrgID: INT [FK to Organization.OrgID], Name: VARCHAR(100), Contact: VARCHAR(100))

UserProfile (UserID: INT [PK], Name: VARCHAR(100), Preferences: VARCHAR(255), Contact: VARCHAR(100))

Events (EventID: INT [PK], OrgID: INT [FK to Organization.OrgID], Location: VARCHAR(100))

EventTags (EventID: INT [PK], Tag: VARCHAR(25))

Rating (EventID: INT [FK to Events.EventID], userID: INT [FK to UserProfile.userID], rating: INT, comments VARCHAR(8192))