Group 7 - Online Social Dating Database Application

Requirement Analysis

Introduction

Purpose

This application stores the information and constraints required to host an online social dating platform that focuses on breaking the conversational “ice” between users that are matched by our application. The application is inspired by the concept of speed-dating, with the intention that users are matched with many different people according to their selected interests and dating preferences and share short conversations centered around pre-selected icebreaker topics that matched users are both interested in. Users involved in matches they deemed successful can request to connect with their match to deepen their relationship and continue to chat.

Scope

We limit the format of messages that are exchanged by users to plain text and emoticons. We assume that security (e.g. passwords) is not a concern and that users input their correct personal information, interests, and dating preferences. We further limit the matchmaking process to the interests and dating preferences provided to users by the application.

Resources

https://thenextweb.com/tech/2019/11/20/this-elitist-dating-app-matches-you-up-for-2-minute-livestream-dates/

Database Description

Entities and Attributes

The following entities and their respective attributes will be stored in the tables of a relational database:

Users: A user is a person who is seeking a relationship. A user has several attributes, including a *name*, *gender*, *birth date*, *email*, *password*, *date and time* of the creation of their account, *profile photo*, and *geographic location*. Further, the *isActive* and *isMatchmaking* attributes denote whether a user is online on the application and is currently seeking to be matched with another user, respectively. A user is identified by their artificial primary key, *userID*.

Interests: A user has interests, which are chosen from a pre-selected list when creating their account. A user's interests are used to help the application find suitable matches and fitting icebreaker topics of conversation. An interest has a primary key attribute *type*, which is the unique name of the interest, and is classified according to its *category*.
Topics: A topic is the icebreaker conversation-starter subject. It has a primary key attribute *subject*, which is the unique name of the topic.
Chats: A chat is the medium wherein matched users engage in short conversations and can request to further chat beyond the icebreaker topics. A chat has a couple of attributes, including an artificial primary key *chatID*, *date and time* of the creation of the chat, and *isActive* flag, which denotes whether a chat is ongoing.
Conversations: A conversation is used to distinguish consecutive icebreaker conversations between the same matched users within a chat. It is also used to distinguish icebreaker conversations from free-chat conversations, in the case of matched users that want to continue to chat beyond the icebreaker topics. A conversation is a weak entity of *Chats* and has a primary key *conversationNumber*.
Messages: A message is a written communication sent by a user. A message's attributes include an artificial primary key *msgID*, string of *content*, *timestamp* denoting when the message was sent, and delivery *status* (e.g. sent, delivered, failed).
Dating Preferences: A user has dating preferences, which are chosen from a pre-selected list when creating their account. A user's dating preferences are used to help the application find suitable matches. A dating preference has a primary key attribute *type*, which is the unique name of the dating preference.
Connections: A connection is issued by a user to request to further chat with their matched user beyond the icebreaker topics, instead of being matched with new users. It has an artificial primary key *ctnID*. A connection also has attributes that include the *status* of the connection (e.g. accepted, declined, pending), *userID1* and *userID2* (the user's involved), and *date and time* of the creation of their account.
Icebreakers: An icebreaker is a subclass of *Conversations* and has an additional attribute *duration* that defines how long an icebreaker conversation is to last. An icebreaker uses a *Topic* selected by the application.

Relationships
Likes: A user *likes* an interest (e.g. sports, music) from a pre-selected list. This is a many-to-many relationship between *Users* and *Interests* that has a participation constraint on *Users*, as a user is required to have at least one interest in order to be properly matched by the application.
Possesses: A user *possesses* a dating preference from a pre-selected list. This is a many-to-many relationship between *Users* and *Dating Preferences* that has a

participation constraint on *Users*, as, similar to *Likes*, a user is required to have at least one dating preference in order to be properly matched by the application.
Relates: An interest *relates* to other interests in order to provide better matchmaking. The relation has a *strength* attribute that ranks how strongly related interests are likely to produce matches that lead to successful connections. It is a many-to-many relationship.
Generates: Interests *generate* topics through the application's matchmaking algorithm that considers users' interests and the *strength* between the interests. It is a many-to-many relationship and has a participation constraint on both of the involved entities as all topics must be generated by interests, and all interests must contribute to generating a topic.
Guides: A topic *guides* an icebreaker conversation. It is a many-to-one relationship with a key and participation constraint on *Icebreakers*, as each icebreaker conversation must have a topic of discussion.
Belongs: An individual conversation *belongs* to a chat. It is a one-to-many relationship and has a participation and key constraint on *Conversations* as it is the weak entity of *Chats*. There is also a participation constraint on *Chats* as a chat cannot exist without a conversation.
Participates: A user *participates* in a chat with other users once a match has been made by the application. It is a many-to-many relationship and has a participation constraint on *Chats* as a chat cannot exist without participating users.
Posts: A user *posts* a message in a conversation of a chat. It is a one-to-one relationship and has a participation constraint on *Messages* as a message cannot be posted without a user to post it.
Has: A user *has* a connection with other users after a request was issued to connect with a matched user to further chat beyond the icebreaker topics. It is a many-to-many relationship and has a participation constraint on *Connections* as a connection cannot exist without the two users participating in the connection.
Requests: A user (requestor) in a chat *requests* to connect with their matched user (requestee) in the same chat. It is a quaternary relationship and has key constraints on all involved entities as a user can only request one connection with another user for a given chat. There is also a participation constraint on *Connections* as a connection cannot exist without the requestor and requestee users and the chat used to facilitate communication.
Contains: A conversation *contains* a posted message. It is a many-to-one relationship and has a participation and key constraint on *Messages* as a given message must belong to exactly one conversation, but a conversation can have many messages.

Operations

Many of the operations required by our application involve matching and connecting users. For example, a critical operation will be querying a user's interests to generate an appropriate conversation topic. Another functional requirement is a method for a user to request a connection with their matched user (see operation signature below). The system should allow a user to accept (see below) or reject a connection request. Further, account creation and login (see below) operations will be needed by the application. The application will also require loading a specific chat (see below) and its messages (see below), and modifying a user's dating preferences and interests (see below).

Sample operations:

match(userid): chatid
requestConnection(userid, chatid) : ctnid
acceptRequest(userid, ctnid) : void
login(email, password): userid
loadChat(userid, chatid): conversations[]
loadMessage(userid, chatid, msgid): content
addInterest(userid, type): void

Relations

Entities

Users(userid, gender, birth_date, name, email, password, date_time_created, profile_photo, location, is_active, is_matchmaking)
Interests(type, category)
Topics(subject)
Chats(chatid, date_time_created, is_active)
Messages(msgid, content, timestamp, status)
Dating Preferences(type)
Connections(ctnid, date_time_created, userid1, userid2, status)
Icebreakers(chatid, conversation_number, duration) (chatid references *Chats*)

Weak Entities

Conversations(chatid, conversation_number) (chatid references *Chats*)

Relationships

Likes(userid, type) (userid references *Users* and type references *Interests*)
Possesses(userid, type) (userid references *Users* and type references *Dating Preferences*)
Relates(type1, type2, strength) (type1 and type2 reference *Interests*)
Generates(type, subject) (type references *Interests* and subject references *Topics*)

Guides(subject, chatid, conversation_number) (subject references *Topics*, chatid references *Chats* and conversation_number references *Conversations*)
Belongs(chatid, conversation_number) (chatid references *Chats*, conversation_number references *Conversation*s)
Participates(userid, chatid) (userid references *Users* and chatid references *Chats*)
Posts(userid, msgid) (userid references *Users* and msgid references *Messages*)
Has(userid, ctnid) (userid references *Users* and ctnid references *Connections*)
Requests(userid1, userid2, chatid, ctnid) (userid1 and userid2 reference *Users*, chatid references *Chats*, and ctnid references *Connections*)
Contains(msgid, chatid, conversation_number) (msgid references *Messages*, chatid references *Chats* and conversation_number references *Conversations*)

Are there opportunities to combine relations?

I think we could have combined the *Likes* and *Possesses* relations without introducing redundancy as *Interests* and *Dating Preferences* will not be redundant. However, this will introduce unwanted complexity for generating *Topics*, as we want *Topics* to be selected according to non-dating interests (e.g. sports, music), hence the separation of the relations.

How does our application domain stand out?

This application extends the online dating scene by focusing on the ease of quickly meeting new people and, most importantly, having a suitable conversation starter automatically generated so that matched users can spend more time getting to know each other and less time worrying about what to talk about. Breaking the “ice” has never been more relaxing!