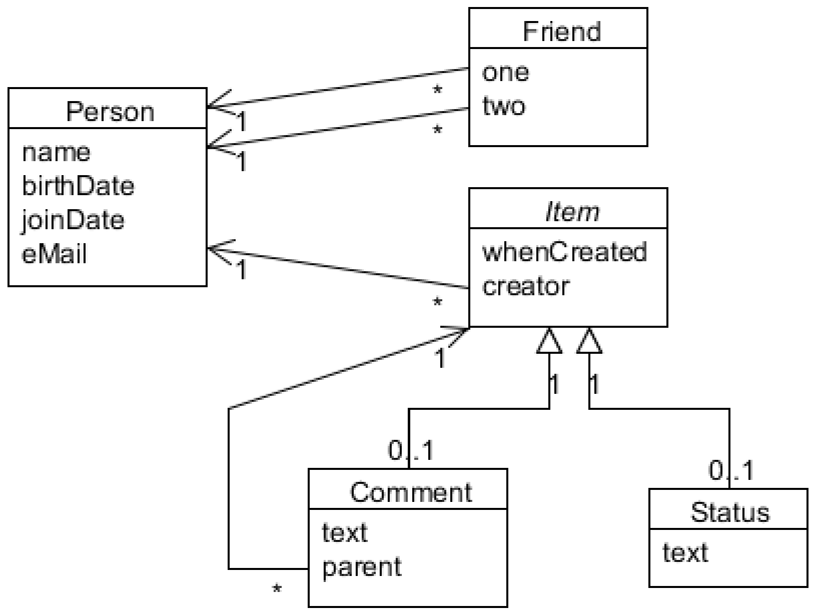
Model Database Design Project: Facebook-Lite

Brian Howard, CSC 480, Spring 2012

This is an adaptation of the Facebook replica that we designed in class, to give an idea of what I am looking for in a design project. This subset of Facebook keeps track of a number of people, who may be mutual friends of each other. Over time, people may post a series of items which are either status updates or comments on other items. The items they may comment on show up in a list of recent posts created by themselves and their friends.

Here is the class diagram (created with UMLet: http://www.umlet.com/):



I have used some non-standard notation in this diagram, borrowed from object-oriented design—the open arrowheads linking the Comment and Status classes to Item emphasize that they behave as subclasses of Item, and the italics on the name Item emphasize that it is an abstract class (that is, all of the rows in Item must correspond to rows in either Comment or Status). The 0..1-to-1 relations support this: each Comment has a corresponding Item, but not every Item is a Comment; the same holds for each Status post. The cardinalities do not show the extra condition (which we may add as an integrity constraint) that every Item is either a Comment or a Status, but not both. I am also using the regular arrowheads to highlight the foreign key relationships; the tail of each such arrow is aligned on the foreign key field (while the head refers to a synthetic primary key field, not shown).

The given relationships are adequate to represent all of the information needed for the application: given a Status or Comment post, it is possible to get its creation time and, through the creator field, the person who posted it. The Friend table allows all of that person's friends to be located. Comments also link to the Item on which they are commenting (which may be a Status post or another Comment).

There are no redundant relationships in the diagram. The only candidates are the two ways to get from Comment to Item (but one gives the time and creator of the Comment, while the other gives that information for the Item on which it comments), and the two ways to get from Friend to Person (which correspond to the two distinct Persons participating in the friendship).

Here is the SQL schema produced from the above diagram:

```
create table Person (
    personID int,
    name varchar(50) not null,
    birthDate date not null,
    joinDate date not null,
    eMail varchar(50) not null,

    primary key (personID),
    -- check that person is at least 13 and email
    -- looks valid (these are both hacks...)
    check (year(joinDate) >= year(birthDate) + 13),
    check (eMail like '_%@_%._%')
);

create table Friend (
    one int not null,
    two int not null,

    foreign key (one) references Person
        on delete cascade
        on update no action,
    foreign key (two) references Person
        on delete cascade
        on update no action,
    -- check that a person is not their own friend
    check (one <> two)
);

create table Item (
    itemID int,
    whenCreated timestamp not null,
    creator int not null,

    primary key (itemID),
    foreign key (creator) references Person
        on delete cascade
        on update no action
);

create table Status (
    itemID int,
    text varchar(500) not null,
    primary key (itemID),
    foreign key (itemID) references Item
        on delete cascade
        on update no action
);

create table Comment (
    itemID int,
    text varchar(500) not null,
    parent int not null,

    primary key (itemID),
    foreign key (itemID) references Item
        on delete cascade
        on update no action,
    foreign key (parent) references Item
        on delete cascade
        on update no action
);
```

Beyond the design considerations discussed above, I have added a few additional constraints. The name and email fields are each set at 50 characters, while the text of a status or comment item is allowed 500 characters; these sizes are arbitrary, and might not be large enough for real use (I envy the boldness of Twitter in just setting a limit of 140 characters and turning it into part of their brand). An attempt is made to ensure that users are at least 13 (the proper way to do this would be to use the SQL interval type, but Derby does not support it…), and that they have entered a valid email address (the "like" operator here says that the eMail field has to match the pattern "_%@_%._%", which is three groups of one or more characters each, separated first by an at sign and then by a dot; the pattern "_%" means "an arbitrary character followed by zero or more additional characters"); the proper way to check for a valid email address is probably just to send a verification email…. The friendship relation is checked to make sure that a person doesn't befriend themselves. Finally, none of the fields are allowed to be null, deletes are cascaded through the foreign keys (so deleting a user will delete their items and any friendship entries in which they participated), and updates are not allowed on fields used by foreign keys.

Here are four example queries:

- Return a table of users and the number of friends they have:

  ```
  select p.personID, p.name, count(*) as friendCount
  from Person p, Friend f
  where p.personID = f.one or p.personID = f.two
  group by p.personID, p.name;
  ```

- List all the friends of the person with email address "bhoward@depauw.edu":

  ```
  select p2.name
  from Person p1, Person p2, Friend f
  ```

```
where p1.eMail = 'bhoward@depauw.edu'
  and ((p1.personID = f.one and p2.personID = f.two)
    or (p1.personID = f.two and p2.personID = f.one));
```

- Retrieve the most recent status post for "bhoward@depauw.edu":

```
select s.text, i.whenCreated
from Status s, Item i, Person p
where s.itemID = i.itemID
  and i.creator = p.personID
  and p.eMail = 'bhoward@depauw.edu'
  and i.whenCreated =
    (select max(i2.whenCreated)
     from Status s2, Item i2
     where s2.itemID = i2.itemID
       and i2.creator = p.personID);
```

- Get all comments for item 42:

```
select c.text, i.whenCreated
from Comment c, Item i
where c.parent = 42
  and c.itemID = i.itemID;
```

One possible view is to show a user a single table consisting of only the status posts of their friends, together with a count of the number of comments on each post. This might be the initial screen displayed upon logging in to the site (the table should be ordered in reverse chronological order, and some technique should be used to only retrieve perhaps 20 items at a time—more on this later).