Assignment 1 (Stage 2)
ER to Relational Mapping (worth 15%)

Introduction

This document contains the standard ER design for Stage 2 of Assignment 1. You must convert this design into a PostgreSQL relational schema (i.e., a collection of `create table`statements) and submit it via the `give` command under Assignments link on the course web site. In performing the conversion from the ER design to a relational schema, you should follow the approach given in the lecture notes on `ER to Relational Mapping`. You may need to investigate and/or derive mappings for the constructs that have not been discussed in the lecture.

Submission

Submission: Login to a CSE Linux machine such as `wagner` and use the `give` command below to submit the assignment (note that the `give` command does not work on `grieg`):

```
give cs9311 a1 a1.sql
```

Deadline: Sunday 2 September 2018 @ 23:59 (Week 6)

Late Penalty: 0.075 *marks* of the total mark (i.e., 15 marks) for each hour late (i.e., 1.8 marks per day).

Notes: For fairness to all students in the class, no special considerations will be given to those:

- who claim to have submitted their assignment but the assignment has not been received by the `give` system above (use the "classrun" command to check if your assignment has been submitted, e.g., "9311 classrun -check a1");
- who submit their assignment a few minutes late and request to be considered as non-late submissions (please do submit your assignment early in case of your network connection problem, computer breakdown, etc);
- who claim that their assignments worked perfectly on their home computers but somehow did not work on CSE Linux machines (we will only test and mark your assignments on CSE Linux machines, and will not consider the results on your own machines. Hence, please test your assignments on CSE Linux machines before submission).

Submission Requirements

The schema you submit will be partially marked by a program (auto-marked). In order for the program to recognise what you've done as being correct, your SQL **must adhere to the following requirements**:

- all tables must have an appropriate *primary key* defined; all *foreign keys* must be identified
- use appropriate domains for each attribute (e.g. a date-of-birth would be done as an SQL `date`, a counter would be done as an SQL `integer` constrained to be ≥ 0)
- if an attribute is a string, and no maximum length is specified, use PostgreSQL's (non-standard) `text` type; otherwise, use an appropriate `varchar(N)` type or one of the supplied domain types
- if an attribute is a boolean value, use the SQL `boolean` type
- wherever possible, not-null, unique and domain constraints must be used to *enforce constraints* implied by the ER design
- derived (computed) attributes should not be included in the SQL schema
- wherever possible, *participation constraints* should be implemented using the appropriate SQL constructs
- map all of the entity class hierarchies in the ER design using the **ER-style** mapping (i.e., one table for each entity class).
- all relationships should be mapped using the approaches described in the lecture notes; in particular, you should avoid over-generalising your SQL schema relative to the ER design (e.g., a 1:n relationship should *not* be mapped in such a way that it can actually be used to form an n:m relationship)

Since the assignment is going to be partially auto-marked, it is very helpful if you use the names that the auto-marker expects. Please follow as much as possible the following naming conventions:

- each table that represents an entity should be given a name which is the "pluralised" version of the entity name (e.g., entity `Person` is mapped to a table called `People` and entity `Photo` is mapped to a table called `Photos`)
- each table that represents an n:m relationship `R` between tables `S` and `T` should be called `S_R_T` (e.g., `Photos_in_Collections`)
- each data attribute in the SQL schema should be given the same name as the corresponding attributes in the ER
- if an attribute in the SQL schema is derived from a relationship in the ER diagram, name it after the relationship (suitably modified to make sense, e.g., if the relationship is `owns` and the attribute is in the table for the entity that is being owned, then you would change the name to `ownedBy`)
- when mapping multi-valued attributes, name the new table by concatenating the entity and attribute names
- when mapping composite attributes, use the names of the "leaf" attributes

- if names in the ER diagram contain multiple words, separate the words by underscores in the SQL schema (e.g., `date_registered`)

Note: if the name you want to use clashes with a PostgreSQL keyword (e.g., `user`), you will need to write the name in double-quotes (i.e., `"user"`) and in all lower-case.

Place the schema in a file called `a1.sql` and submit it via the `give` command (see above) or via WebCMS3 before the deadline. To give you a head-start, a template for the schema is available, which has (parts of) some of the required tables already defined. Note that you will need to add more tables, as well as filling out the attributes in the supplied tables. Your submission must follow this format, so save a copy of this and edit it to produce your submittable `a1.sql` file.

The reason for insisting on strict conformance to the above is that your submission will be partially auto-marked as follows:

- we will create an initially empty database (no tables, etc.)
- we will load your schema into this database
- we will use a script to compare your schema with the expected schema

The comparison will make use of the meta-data which has been added to the database by loading your schema. If your schema has load-time errors, then it is not going to be possible to compare it against the correct version, so it will not get any marks. Therefore it is essential that you check that your schema can load into an *initially empty* database before you submit it.

Following the instructions above is considered to be a requirement of this assignment. If you stray from the expected schema, your submission will be marked as incorrect. Our auto-checking scripts have a little flexibility, but not much, so do not rely on it. Manual checking will be used to examine specific implementations that are difficult to be auto-checked.

Please do not try to second-guess or improve the standard ER design below. Even if you think it can be further improved, just translate it as given. If you think that it is incorrect or that the information supplied is not enough to do the mapping unambiguously, either send an email to the Lecturer or Course Admin or post a message on the course website Foruns (section: Assignments/Assignment 1/Assignment 1 (Stage 2)) and we will either explain or fix the issue. Also, if you want to give opinions on the standard ER schema use the Assignments/Assignment 1/Assignment 1 (Stage 2) section on the course website Forums.

Standard ER Design

This ER design gives one possible data model for the `myPhotos.net` application introduced in the first stage of this assignment. The design here is based on the discussions on the Forums and on our interpretation of the more ambiguous aspects of the requirements. This is not necessarily the design that would be used in practice and may not even follow all of the requirements from Stage 1 precisely. It has been designed to make Stage 2 of the assignment more interesting (i.e., to give you experience with a range of modelling constructs and translation mechanisms).

To make the presentation clearer, the design is broken into a number of sections. Note that an entity will have its attributes and class hierarchy defined exactly once. If an entity is used in a later section of the design (e.g., to show relationships), it will simply be shown as an unadorned entity box (and you should assume all of the attributes and sub/super-classes from its original definition).

The development of any significant design requires assumptions. Assumptions specific to particular entities and relationships are presented below each diagram.

A general strategy used in the design is to introduce a numeric primary key called `id` into all major entities. This is despite the fact that we could have made a primary key from existing attributes in many cases (e.g., `email`). The reason for doing this is that primary keys typically end up as foreign keys in other tables, and thus their values need to be copied to many places in the database. "Natural" keys (such as `email`) are strings (typically 40-60 bytes), whereas numeric keys are 4-byte integers, so there is a clear space saving in maintaining copies of smaller keys. Also, natural keys have a habit of changing (e.g., someone gets a new email account) and changing a primary key value can have effects that propagate throughout the database. Using numeric keys also makes indexing and various query processing techniques faster. One disadvantage is that we add an extra attribute into each table.

Other notational conventions in the ER diagrams:

- primary key attributes for entities are underlined
- total participation in a relationship is indicated by a thick line
- an arrow indicates that each entity at the non-arrow end is associated with at most one entity at the arrow end

Note that the data here is sufficient to allow the `myPhotos.net` site to be built. Some notions mentioned in the Stage 1 requirements are related to the working of the application and do not need to be explicitly modelled here. Actions (e.g., adding a person to a contact list) typically do not have a presence in the data model either, although they clearly affect the data in the database.

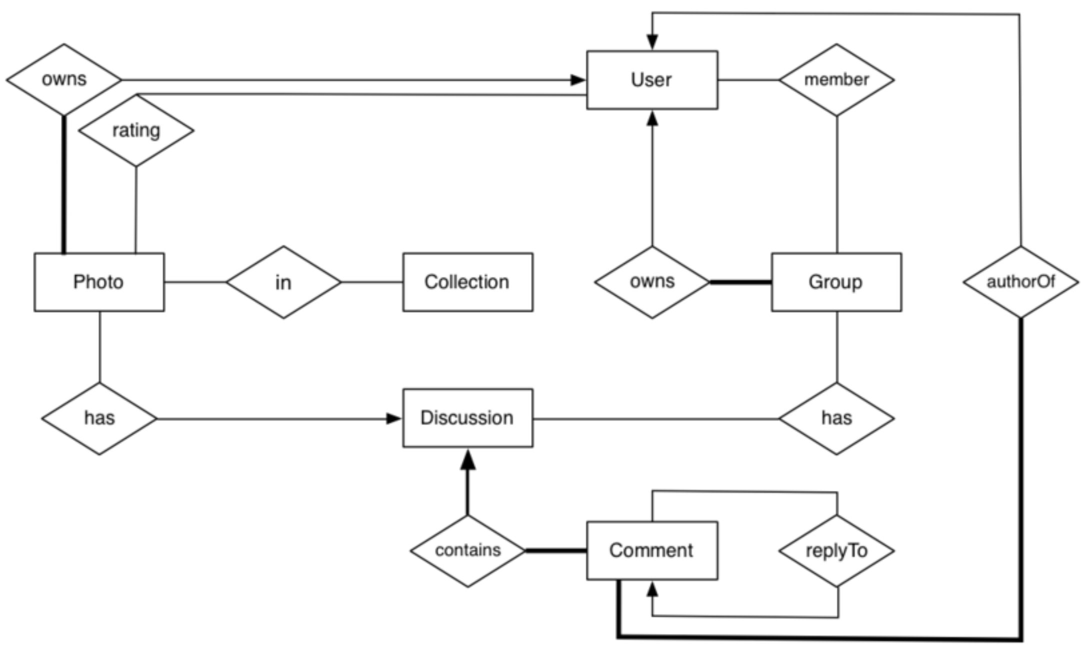
Data Types

To make your life simpler, we have defined some useful data types using the `create domain` statement. Some of the `create domain` statements use standard SQL patterns for specifying constraints, while others use PostgreSQL-specific regular expressions for this purpose. The domain definitions are given at the top of the template file.

You should not need to use many `varchar(N)` types in this assignment. The above types ought to be sufficient for most of the fields in the database. Use them wherever you think it is appropriate.

Overview

The following diagram provides an overview of the major entities in **myPhotos.net** and the major relationships between them. Other entities and relationships and all attributes can be found in the diagrams below.

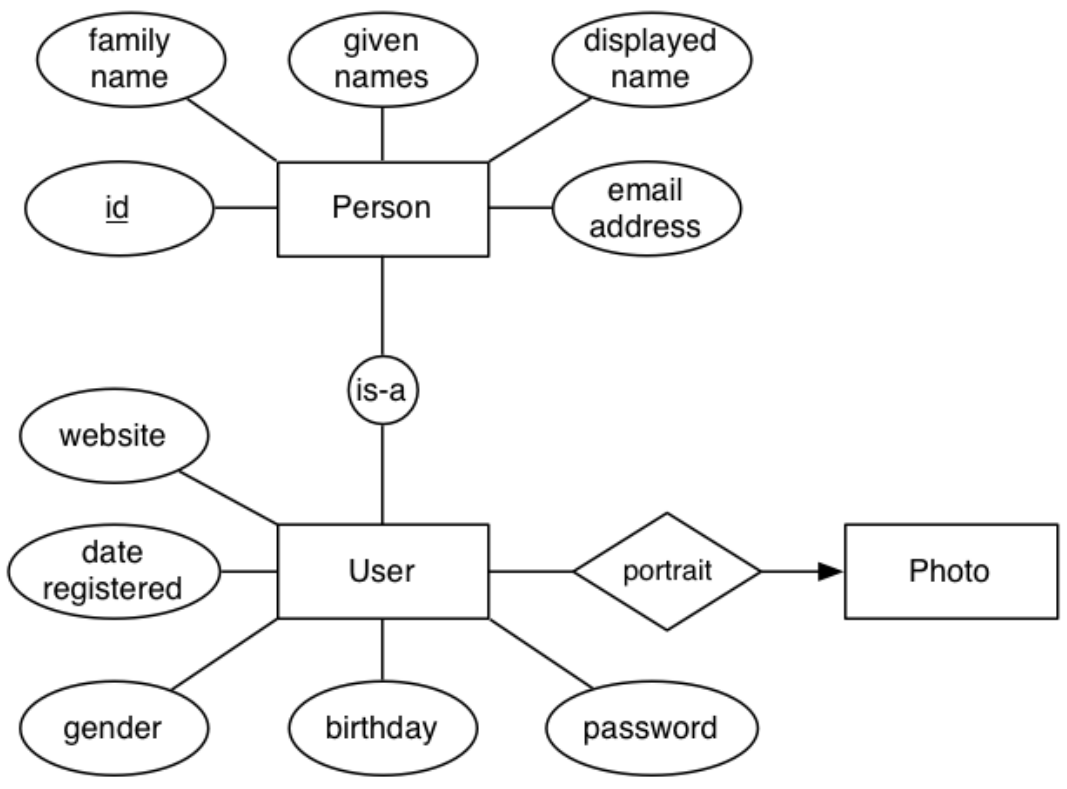


Comments:

- every photo and group has an owner who is a user
- users can rate many photos; each photo can be rated by many users
- all comments occur in the context of a discussion
- each discussion is created when its first message is posted
- every comment has an author (i.e., the user who posted it)
- a comment may be a reply to some other comment

Users and People

The following diagram shows the entities, attributes and relationships that provide the information about people on the **myPhotos.net** site.



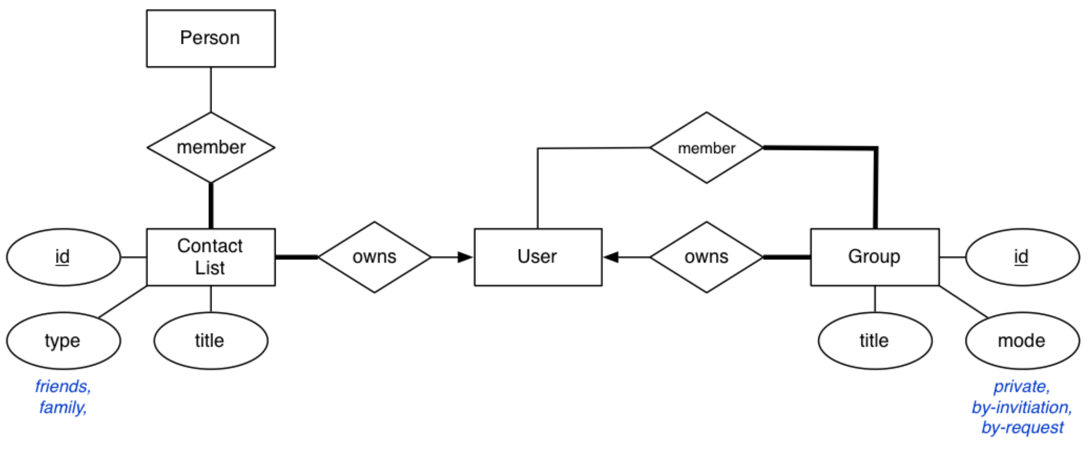
Comments on People:

- we use a numeric ID as a primary key, since People and Users will be extensively referenced in the database
- for every person in the database, we need to know their email and their given-names
- information which every user is required to provide (as well as their person data): password

- users may provide additional data such as their birthday (date) and their gender ('male', 'female')
- all people have a name for the system to display them as; users may provide such a name for themselves; if no name is supplied, myPhotos.net will form a name from the family- and given-names
- note the use of is-a in a circle to indicate that the Person entity has only a single sub-class (User); remember that you must implement this (very small) class hierarchy via the ER-style mapping
- we assume that names are no longer than 50 chars (for the given-names and family-name components) and up to 100 chars for displayed names
- a user's website (if supplied) is simply a URL text string
- users may supply a portrait to be displayed when they are referenced on the website; this portrait is a photo of them in JPEG format

Contact Lists and Groups

Contact Lists and Groups both represent collections of people, but in the case of Groups they are collections of people who are users. Since the membership relations refer to different entities, we also treat Contact Lists and Groups as different entities. The following diagram shows entitites, attributes and relationships that deal with various groups of people on the **myPhotos.net** site. Note that, to keep things simple, we are completely ignoring the process by which users become members of groups; this would typically require additional tables related to invitations and requests.

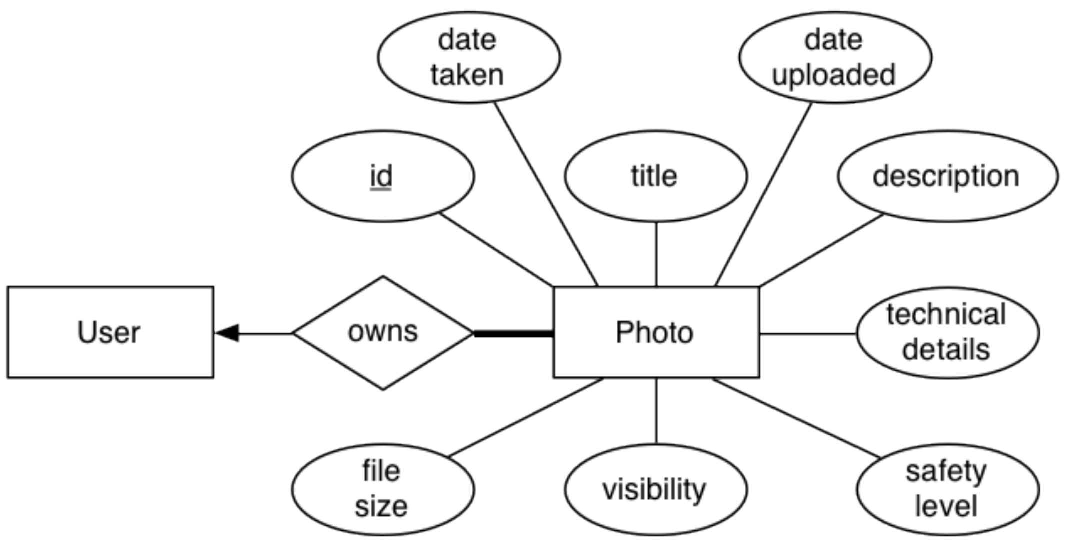


Comments on Contact Lists and Groups:

- users can build contact lists for easy reference to groups of people
- every contact list must have a title; contact lists cannot be empty and are created in conjunction with the first member
- all members of a contact list must be entered into the database as people
- contact lists may have a type specified, either "friends" or "family" or null if no specified type
- groups are also created by users; the creator becomes the owner
- groups also have IDs and titles, and must have a mode to describe how membership of the group is organized
- any user may be a member of multiple groups; every group must have at least one member (the owner is automatically a member)

Photos

The following diagram shows entities, attributes and relationships relevant to photos on the **myPhotos.net** site. We assume that the photo files themselves are stored in the file system with a name based on the photo ID and that the application logic can retrieve a photo given this ID. Thus, no photo image file data is actually stored in the database (except file size).



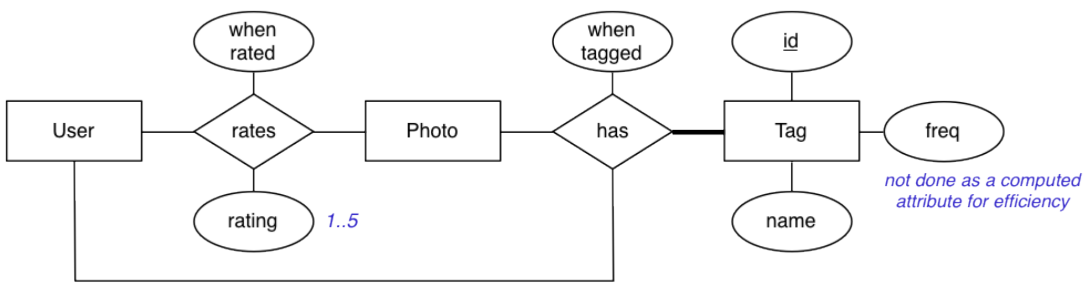
Comments on Photos:

- each photo has an automatically defined unique numeric ID
- a photo must have a title, a string up to 50 characters long
- a photo may have a description, which is arbitrary text

- the user may supply the date on which the photo was taken; the system will automatically provide the date it was uploaded
- the file size of a photo is stored as a whole number of KB
- visibility is given by one of the values 'private','friends','family','friends+family','public'
- safety level is given by one of the values 'safe','moderate','restricted'
- since the description of the technical details of photos can vary widely, such details are supplied, if the owner wants, simply as a text string

Tags and Ratings

Associated with each photo is user-supplied information on its content and quality. Including it on the diagram above would have made things to messy, so we show entities, attributes and relationships for photos and their ratings and tags below:

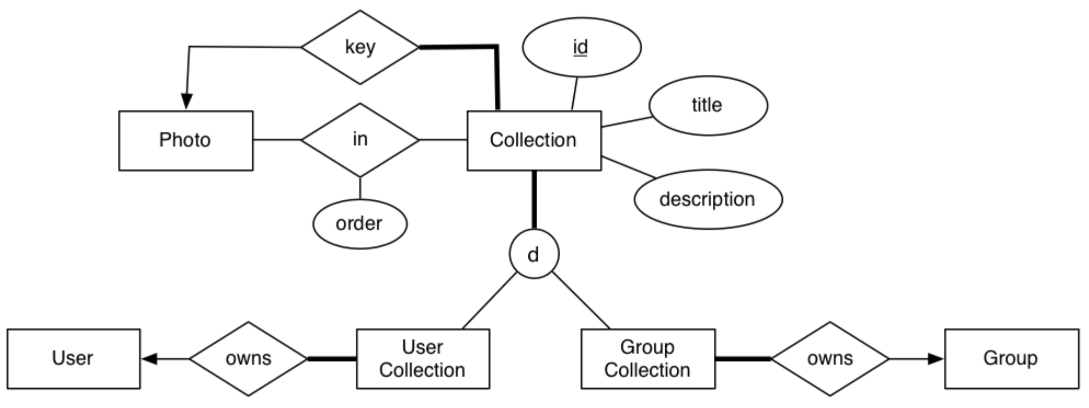


Comments on Tags and Ratings

- a photo may have an arbitrary number of tags (hash tags) associated with it
- each tag is defined primarily by a descriptive name (up to 50 characters in length)
- in order to efficiently accomodate interface notions like "tag clouds", a count of the number of times each tag is used is stored in the tag record and maintained by the application
- a photo can be tagged by many users (and each user might tag it differently); we keep a record of when they tag it (time-stamp)
- each time a photo is tagged, the frequency counter for that tag is incremented
- a photo may also be rated on a 1..5 star scale by users
- each time a rating is made, the system records the user who made the rating, a time-stamp for when the rating was made, and the actual rating value

Collections

The following diagram shows entities, attributes and relationships for the photo collections that are managed in the **myPhotos.net** site.

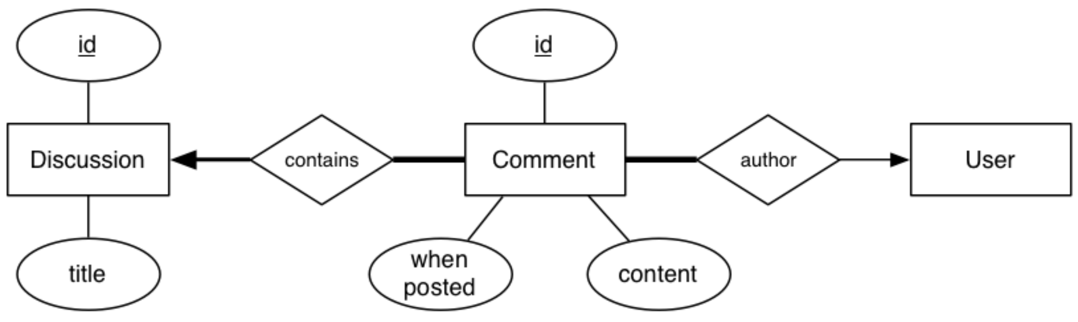


Comments on Collections:

- a collection is simply a group of photos, identified by a numeric ID
- each collection must have a title (up to 50 chars) and may have an arbitrary text description
- each collection has an identified member photo which is used for displaying the collectio at the interface
- photos are marked with the rank order (a small positive integer) in which they appear when the entire collection is displayed; this allows users to choose how the collection is viewed
- some collections are built by an individual user and owned by that user, who is the only person who can modify which photos are in the collection and the order that they appear
- other collections are associated with a group, and any member of the group can modify which photos are in the collection and the order that they appear

Discussions and Comments

The following diagram shows entities, attributes and relationships for the discussions and comments in the **myPhotos.net** site.



Comments on Discussions:

- discussions are essentially holders for collections of comments
- each discussion has an identifying numeric ID and may have a title (up to 50 chars)
- a discussion contains one or more comments and is created when the first comment is posted
- a collection of comments on one photo needs no title; the discussion is associated with the photo being commented on
- discussions under a group will have a title and will be associated the group

Comments on Comments:

- comments are messages posted by users as part of a discussion
- each comment has an identifying numeric ID
- the system time-stamps each comment with the precise posting date and time
- the content of a comment is an arbitrary text string