Project 1 Requirements Document

We will build a task tracker that lets the users organize and manage tasks. We need to design a database to store the data, with the following requirements:

Basic:

- We'll have 6 classes in the UML:
  - Task
  - List
  - Tag
  - Subtask
  - User
  - Comment
- There will be many to many relationships (task-tag) and one to many (user- task, list-task, task-comment, task-subtask) relationships between the classes.

Part 1: task-user-tags: Tim Li

- Each task has a title, due date, create date, and status. Users can add the URL and a priority to the task.
- Each user has a userID, first name, last name, and email. Users can create tasks, finish tasks, delete tasks.
- Users can assign tasks to another user. The default assignee of the task is the creator.
- The task has two kinds of status: todo and done.
- Each task can have 0 or more tags. Each tag has a name. Each tag can also be added to 0 or more tasks

Part 2: task-list-subtask-comment shuyichicken@gmail.com

- The assignee can make multiple comments on the task.
- Each Comment has an update time and a content. The task can have 0 or more comments.
- Each List has a name. The tasks can be moved to a list. The list can have 0 or more tasks.
- Each task can have 0 or more subtasks. The subtask has the name and status of either todo or done.

Others:

- The app will support filters such as the "Unscheduled work tasks", "Important tasks in the next 2 weeks", "Older than a month", "assigned to me" and so on, which demands queries contain a join of at least three tables, subquery, and complex search criterion.
- The app will also have a reporting feature to analyze the number of finished tasks grouped by list, or month, which will demand queries using the clause of "GROUP BY", "HAVING" and "PARTITION BY ".

Nouns as Class:

- Task
- List
- Tag
- Subtask
- User
- Comment

Nouns as Attributes:

- Task
    - taskID
    - title
    - due date
    - create date
    - URL
    - priority
    - status
- Tag
    - tagID
    - name
- User
    - userID
    - first name
    - last name
- List
    - listID
    - name

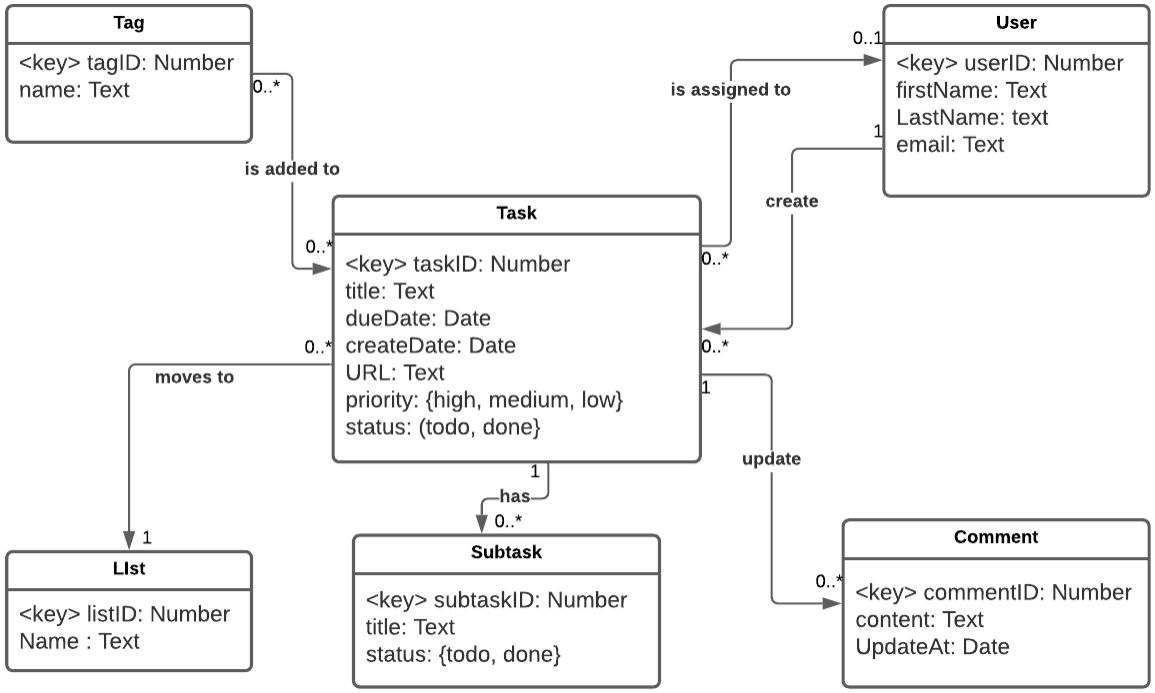
- Subtask
    - subtaskID
    - title
    - status
- Comment
    - commentID
    - content
    - updateAt

Verbs:

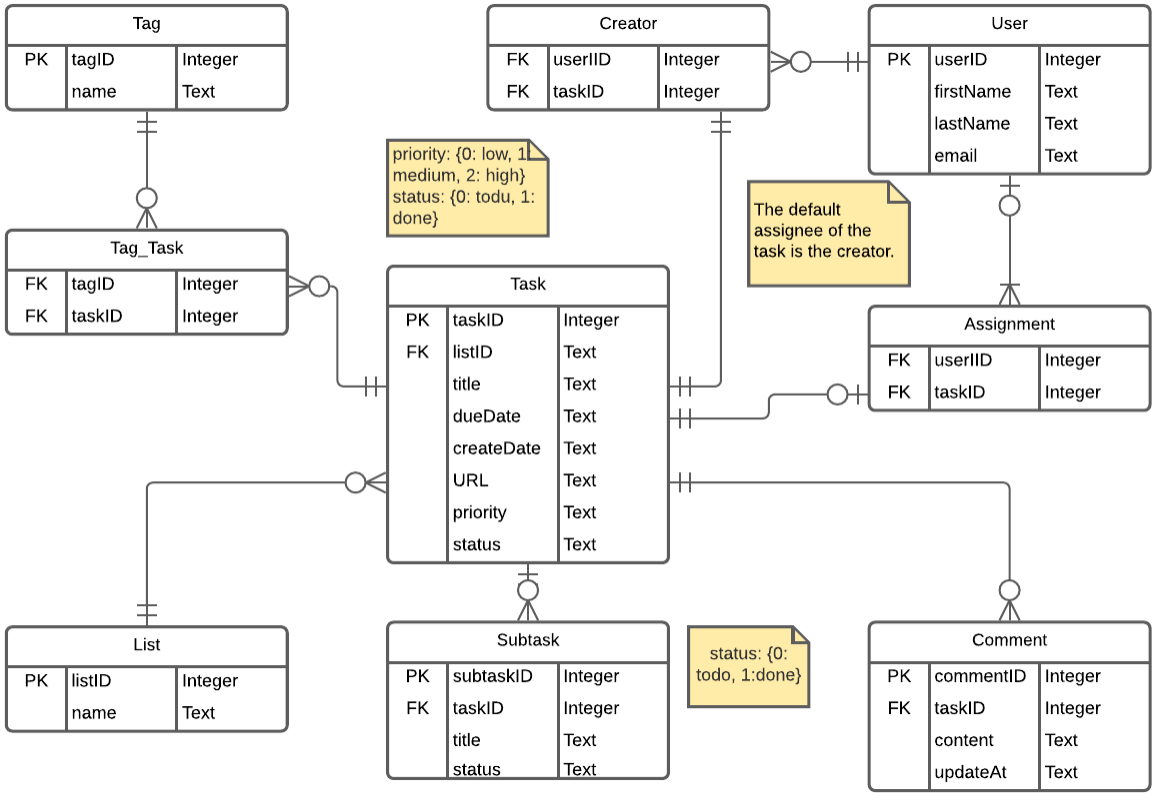
- User- task
    - Move
    - Create
    - Finish
    - Assign

    - delete
- Task-tag
    - add
- task-list
    - move
- Task-subtask
    - has
- Task-comment
    - update

UML



ERD



Definition of relational schema with proof that it is in BCNF.

- Tag(tagID, name)
  - tagID -> name, tagID is a superkey
- Tag_Task(tagID, taskID)
  - {tagID, taskID} -> {tagID, taskID} is a trivial FD
- Task(taskID, listID, title, dueDate, createDate, URL, priority, status)
  - taskID -> {listID, title, dueDate, createDate, URL, priority, status}, taskID is a superkey
- User(userID, firstName, lastName, email)
  - userID -> {firstName, lastName}, userID is a superkey
  - email -> {firstName, lastName}, email is a superkey
- Creator(userID, taskID)
  - {userID, taskID} -> {userID, taskID} is a trivial FD
- Assignment(userID, taskID)
  - {userID, taskID} -> {userID, taskID} is a trivial FD
- SubtaskID(subtaskID, taskID, title, status)
  - subtaskID -> {taskID, title, status}, subtaskID is a superkey

- ListID(<u>listID</u>, name)
    - listID -> name, listID is a superkey
- Comment(<u>commentID</u>, taskID, content, updateAt)
    - commentID -> {taskID, content, updateAt}, commentID is a superkey

SQL file (text file) with the table definitions and creation SQL statements that can be executed with SQLite3.(10 pts)

Create a set of SQL data definition statements for the above model and realize that schema in SQLite3 by executing the script from the SQLite3, the console or Node. You can use DB Browser to generate these statements. Show that the tables were created and conform to the constraints through screen shots or other means.