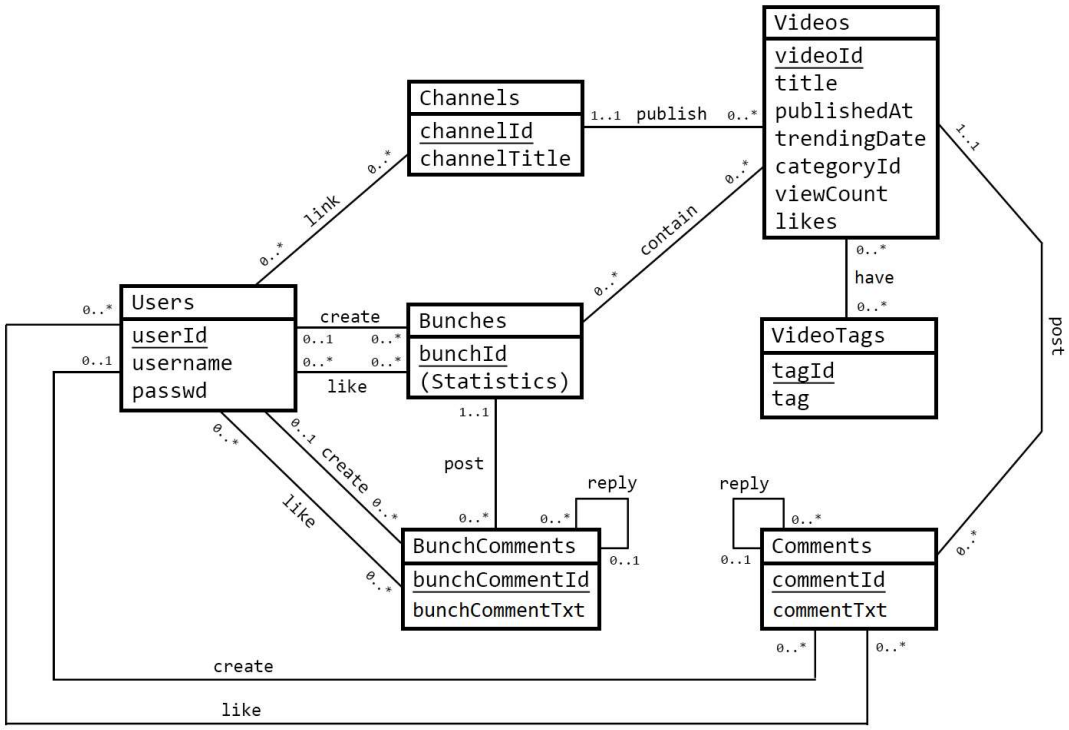
① UML



Videos:

Store the video data from YouTube trending dataset.

VideoTags:

Store the video tags from dataset separately.

- 'have' relation between Videos and VideoTags is many-to-many, since a video can have multiple tags and a tag can be referred by multiple videos.

Channels:

Store the channels from dataset separately.

- In order to reduce redundancy of attribute 'channelTitle'. (see BCNF analysis)
- 'publish' relation between Channels and Videos is one-to-many, since a video must be published on exactly one channel.

Users:

Store the user info of those who sign up for our website.

- 'link' relation between Users and Channels is many-to-many. It is used when a user wants to specify that which channels he/she owns, uses or concerns about.

Bunches:

Store the bunches of videos a user collected, along with some statistics. We use **TRIGGER**s to synchronize the statistics (some sums, maxima, and minima, which can be easily updated).

- 'create' relation between Users and Bunches is one-to-many. If a user deletes the account, the bunch should be kept the same except for its owner set to null, so we use 0..1 to indicate its possibility to be null.
- 'like' relation between Users and Bunches is many-to-many, because everyone can like any of the bunches, and a bunch can be liked by multiple users.
- 'contain' relation between Bunches and Videos is many-to-many, since a bunch can contain multiple videos and a video can be added into multiple bunches.

Comments:

Store the comments of our users on certain videos.

- 'create' and 'like' relations are similar to those related to Bunches.
- 'post' relation between Comments and Videos is many-to-one, since a comment must rely on the specific video and a video can have multiple comments on it.
- 'reply' relation inside Comments is one(parent)-to-many(childs) as a tree-like structure, and the root comments can have no parent.

BunchComments:

Store the comments of our users on certain bunches.

- 'create', 'like', 'post', and 'reply' relations are similar to those related to Comments.

② BCNF

1. Channels (channelId, channelTitle)
   - All attributes are atomic,
   - for every non-trivial function X→Y: channelId→channelTitle, X is the superkey, so it follows BCNF
2. Videos (videoId, title, publishedAt, trendingDate, categoryId, viewCount, likes, channelId)
   - All attributes are atomic
   - for every non-trivial function X→Y: only videoId is a primary key and it is the only attribute that can determine the video, so X is the superkey, so it follows BCNF
3. Bunches (bunchId, userId)
   - All attributes are atomic
   - for every non-trivial function X→Y: bunchId→userId, X is the superkey, so it follows BCNF
4. Users (userId, userName, passwd)
   - All attributes are atomic
   - for every non-trivial function X→Y: userId→userName, userId→passwd, X is the superkey, so it follows BCNF
5. Comment (commentId, commentTxt, userId, videoId, parentCommentId)
   - All attributes are atomic
   - for every non-trivial function X→Y: commentId→{commentTxt, userId, videoId, parentCommentId}, X is the superkey, so it follows BCNF
6. BunchComment (bunchCommentId, bunchCommentTxt, userId, bunchId, parentBunchCommentId)
   - All attributes are atomic
   - for every non-trivial function X→Y: BunchCommentId→{bunchCommentTxt, userId, bunchId, parentBunchCommentId}. userId and bunchId are the foreign keys to Comment and User. All of those X in the LHS are the superkey, so it follows BCNF.

All of the other tables representing the relations contain foreign keys to the entity tables above.

Actually, we designed the tables with the consideration about BCNF criteria. We choose to make Channels an individual table, because if the channelId and channelTitle are in the Videos, we have

videoId→{videoId, title, publishedAt, trendingDate, categoryId, viewCount, likes, channelId, channelTitle}

and

channelId→ channelTitle,

which can be decomposed into two, so we create Channels as a new table including channelId and channelTitle and use channelId as a foreign key. Then, all the tables can’t be decomposed, thus follow BCNF.