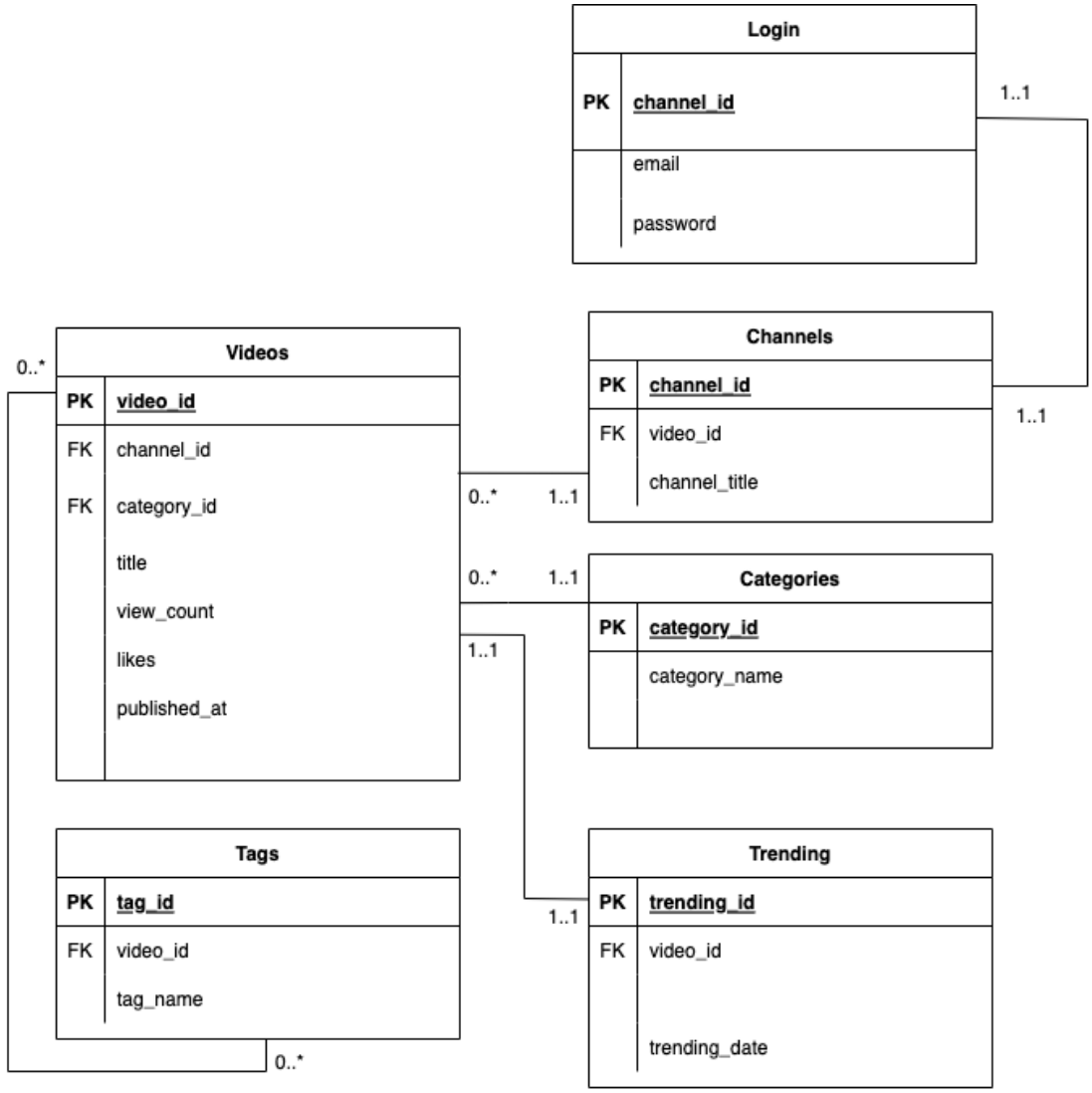
Stage 2: Database Design

MyTube UML Diagram



Normalization and Process:

Looking at the UML Diagram shown above, we think that 3NF is more suitable. Because there can be many videos in categories, videos in channels, videos that are trending, etc, we think that redundancy will be a big theme for our project. 3NF is comparatively a lot more redundant than BCNF. Let's first analyze the UML Diagram and write the functional dependencies of this set of relations. As we denote the functional dependencies of each relation, we will also find the minimal basis and perform 3NF decomposition on each relation.

Login(channel_id, email, password):
FD = {channel_id → email, channel_id → password}

1. **Convert the right hand side to singletons:**
   Already completed. FD = {channel_id → email, channel_id → password}
2. **Remove the redundant/unnecessary left hand side attributes:**
   Since the left hand side of both FDs are singletons, we do not need to remove any redundant left hand side attributes. FD = {channel_id → email, channel_id → password}
3. **Remove inferable relations:**
   We cannot get email from channel_id → password and cannot get password from channel_id → email, so there are no inferable relations to remove.

Thus, the minimal basis: {channel_id → email, channel_id → password}
The relation set is R1(channel_id, email), R2(channel_id, password)

Videos(video_id, channel_id, category_id, title, view_count, likes, published_at):
FD = {video_id → channel_id, video_id → category_id, video_id → title, video_id → view_count, video_id → likes, video_id → published_at}

1. **Convert the right hand side to singletons:**
   Already completed. FD = {video_id → channel_id, video_id → category_id, video_id → title, video_id → view_count, video_id → likes, video_id → published_at}

2. **Remove the redundant/unnecessary left hand side attributes:**
   Since the left hand side of all FDs are singletons, we do not need to remove any redundant left hand side attributes. FD = {video_id → channel_id, video_id → category_id, video_id → title, video_id → view_count, video_id → likes, video_id → published_at}

3. **Remove inferable relations:**

We cannot get any of the RHS from the LHS since the RHS of every FD is not a LHS of another FD.

Thus, the minimal basis: {video_id → channel_id, video_id → category_id, video_id → title, video_id → view_count, video_id → likes, video_id → published_at}
The relation set is R1(video_id, channel_id), R2(video_id, category_id), R3(video_id, title), R4(video_id, view_count), R5(video_id, likes), R6(video_id, published_at)

Channels(channel_id, video_id, channel_title):
FD = {channel_id → video_id, channel_id → channel_title}

1. **Convert the right hand side to singletons:**
   Already completed. FD = {channel_id → video_id, channel_id → channel_title}
2. **Remove the redundant/unnecessary left hand side attributes:**
   Since the left hand side of all FDs are singletons, we do not need to remove any redundant left hand side attributes. FD = {channel_id → video_id, channel_id → channel_title}
3. **Remove inferable relations:**
   We cannot get any of the RHS from the LHS since the RHS of every FD is not a LHS of another FD.

Thus, the minimal basis: {channel_id → video_id, channel_id → channel_title}The relation set is R1(channel_id, video_id), R2(channel_id, channel_title)

Categories(category_id, category_name):
FD = {category_id → category_name}

1. **Convert the right hand side to singletons:**
   Already completed. FD = {category_id → category_name}
2. **Remove the redundant/unnecessary left hand side attributes:**
   Since the left hand side of all FDs are singletons, we do not need to remove any redundant left hand side attributes. {category_id → category_name}
3. **Remove inferable relations:**
   We cannot get any of the RHS from the LHS since the RHS of every FD is not a LHS of another FD

Thus, the minimal basis is: {category_id → category_name}
The relation set is: R1(category_id, category_name)

Tags(tag_id, video_name, tag_name):
FD = {tag_id → video_name, tag_id → tag_name}

1. **Convert the right hand side to singletons:**
   Already completed. FD = {tag_id → video_name, tag_id → tag_name}
2. **Remove the redundant/unnecessary left hand side attributes:**
   Since the left hand side of all FDs are singletons, we do not need to remove any redundant left hand side attributes. FD = {tag_id → video_name, tag_id → tag_name}
3. **Remove inferable relations:**
   We cannot get any of the RHS from the LHS since the RHS of every FD is not a LHS of another FD.

Thus, the minimal basis is: {tag_id → video_name, tag_id → tag_name}
The relation set is: R1(tag_id, video_name), R2(tag_id, tag_name)

Trending(trending_id, video_id, trending_date)
FD = {trending_id → video_id, trending_id → trending_date}

1. **Convert the right hand side to singletons:**
   Already completed. FD = {trending_id → video_id, trending_id → trending_date}
2. **Remove the redundant/unnecessary left hand side attributes:**
   Since the left hand side of all FDs are singletons, we do not need to remove any redundant left hand side attributes. FD = {trending_id → video_id, trending_id → trending_date}
3. **Remove inferable relations:**
   We cannot get any of the RHS from the LHS since the RHS of every FD is not a LHS of another FD

Thus, the minimal basis is: FD = {trending_id → video_id, trending_id → trending_date}
The relation set is: R1(trending_id, video_id), R2(trending_id, trending_date)

As we can see, the relations are all normalized through 3NF decomposition. The “Tags” table, with its many-to-many relationship to the “Videos” table, requires an associative able to manage this relationship effectively. By doing so, we can ensure that tags remain universal and can be used across multiple videos. This eliminates any redundancy that might arise from storing the same tag multiple tags for different videos. We think that 3NF is a good balance between reducing redundancy and maintaining performance, given the requirements of our schema.

Description of Relationship and Cardinality:

Videos can be considered as our main table. It has a many to many relationship with Tags. It has a one to one relationship with trending. It has a one to one relationship with channels. Channels also has a many to many relationship with categories and a one to one relationship with login. Below is an explanation of the assumptions we are making when defining relationships:

Videos - A video has exactly 1 channel since only a single channel can upload a video, A video has exactly 1 category associated with it since only 1 category can be assigned to a video, A video has exactly 1 trending since a video can only appear on the trending tab once, A video can have 0 to many tags since tags describe the video and there are many ways to describe the video.

Login - A login has exactly 1 channel since 1 login has only a single channel associated with it

Channels - A channel has exactly 1 login since 1 channel has exactly a single channel associated with it. A channel has 0 to many videos associated with it since a single channel can upload multiple videos.

Categories - A single category can have 0 to many videos since for example the category "Educational" can contain videos about Python to Volcanoes.

Trending - A trending id has exactly 1 video on it since it would be redundant to put the same video on trending multiple times.

Tags - A tag can have multiple videos associated with it. Consider the tag "podcast", which can have videos from NPR to JJ Reddick for example.

Relational Schema

Videos(video-id:VARCHAR(255) [PK], channel_id: VARCHAR(255) [FK to Channels.channel_id], category_id:VARCHAR(255)[FK to Categories.category_id], title:VARCHAR(255), view_count: INT, likes: INT, published_at: DATE/TIME)

Login(channel_id:VARCHAR(255) [PK], email:VARCHAR(255) , password:VARCHAR(255))

Channels(channel_id:VARCHAR(255) [PK], video_id:VARCHAR(255) [FK to Videos.video_id], channel_title: VARCHAR(255))

Categories(category_id:VARCHAR(255) [PK], category_name:VARCHAR(255))

Trending(trending_id:VARCHAR(255) [PK], video_id:VARCHAR(255) [FK to Videos.video_id], trending_date:DATE)

Tags(tag_id:VARCHAR(255)[PK], video_id:VARCHAR(255) [FK to Videos.video_id], tag_name:VARCHAR(255))

Relational Schema

Tags(tag_id:VARCHAR(255)[PK], video_id:VARCHAR(255) [FK to Videos.video_id], tag_name:VARCHAR(255))

```
CREATE TABLE Tags (
        tags VARCHAR(255),
        video_id VARCHAR(255),
        likes  INT,
        PRIMARY KEY(tags),
        FOREIGN KEY (video_id) REFERENCES Videos(video_id)
);
```

Channels(channel_id:VARCHAR(255) [PK], video_id:VARCHAR(255) [FK to Videos.video_id], channel_title: VARCHAR(255))

```
CREATE TABLE Channels (
        channelId VARCHAR(255),
        video_id VARCHAR(255),
        channelTitle VARCHAR(255),
        PRIMARY KEY(channelId),
        FOREIGN KEY (video_id) REFERENCES Videos(video_id)
);
```

Categories(category_id:VARCHAR(255) [PK], category_name:VARCHAR(255))

```
CREATE TABLE Categories (
        categoryId VARCHAR(255),
        video_id VARCHAR(255),
        categoryName VARCHAR(255),
        PRIMARY KEY(categoryId),
        FOREIGN KEY (video_id) REFERENCES Videos(video_id)
);
```

Videos(video-id:VARCHAR(255) [PK], channel_id: VARCHAR(255) [FK to Channels.channel_id], category_id:VARCHAR(255)[FK to Categories.category_id], title:VARCHAR(255), view_count: INT, likes: INT, published_at: DATE/TIME)

```
CREATE TABLE Videos(
        video_id VARCHAR(255),
        channel_id VARCHAR(255),
        category_id VARCHAR(255),
        title VARCHAR(255),
        view_count INT,
        likes  INT,
        published_at  DATE,
        PRIMARY KEY(video_id)
);
```