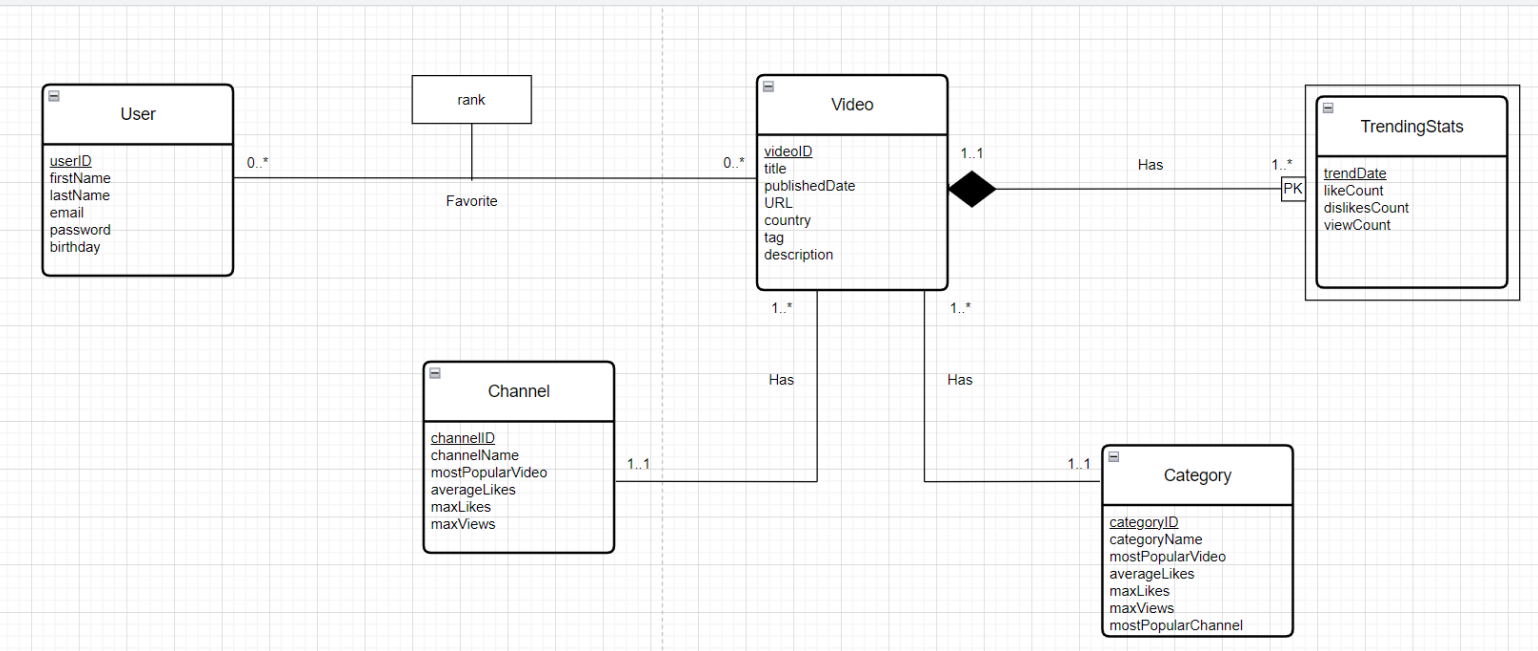
https://console.cloud.google.com/sql/instances/cs-411-tubetrendz/overview?project=cs-411-tubetrendz

TubeTrendz Database UML Diagram:



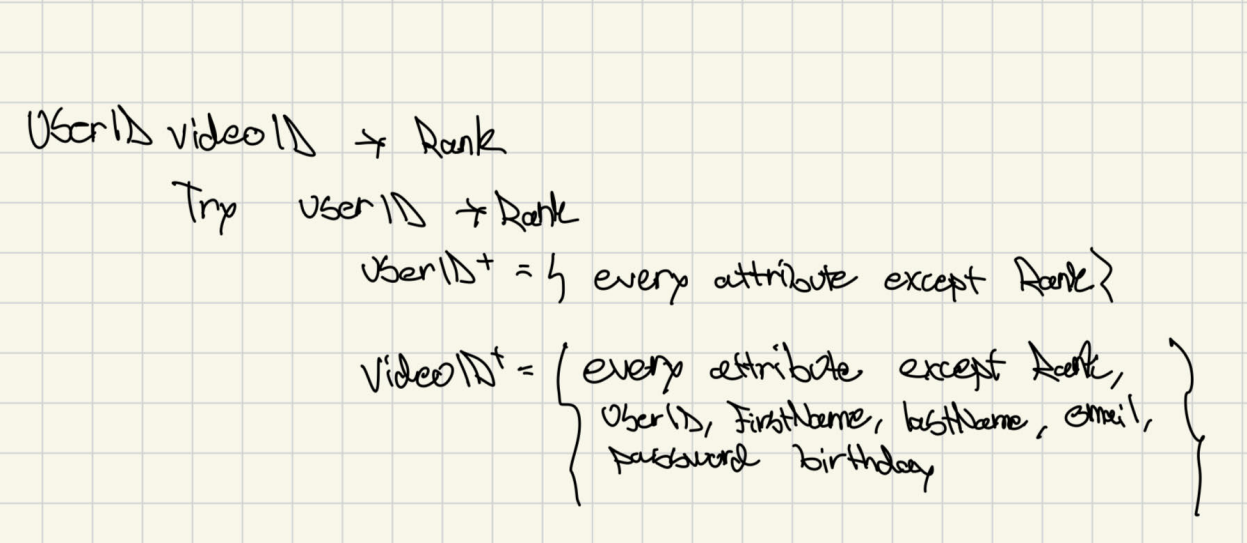
Database Normalization:

Let relation R define our Database:

Functional Dependencies = {
userID -> firstName, lastName, email, password, birthday
userID, videoID -> rank
userID -> videoID,
videoID -> title, publishedDate, URL, country, tag, description
videoID -> trendDate,
trendDate -> likeCount, dislikesCount, viewCount
videoID -> categoryID,
categoryID -> categoryName, mostPopularVideo1, averageLikes1, maxLikes1, maxViews1, mostPopularChannel
videoID -> channelID
ChannelID -> channelName, mostPopularVideo2, averageLikes2 maxLikes2, maxViews2
mostPopularVideo1 -> videoID
mostPopularVideo2 ->videoID
mostPopualurChannel -> ChannelID
}

Decomposing R into 3NF:

1. Find Minimal basis of R:
    a. Remove unnecessary attributes from LHS
        i. userID,videoID -> Rank



   ii. Since userID and videoID cannot reach attribute Rank by closure of userID or closure of videoID functional, we cannot simplify the LHS of this FD any further.
- b. Remove FDs that can be inferred from the rest
   i. None of the FDs listed above can be removed from the relation as they cannot be inferred from the rest.

2. For each FD in the minimal basis, use the attributes in the FD to define the schema for the new relation.
   a. Users (userID, firstName, lastName, email, password, birthday)
   b. Favorites(userID, videoID, rank)
   c. Videos(videoID, title, publishedDate, URL, country, tag, description)
   d. Channel(channelID, channelName, mostPopularVideo2, averageLikes2, maxLikes2, maxViews2)
   e. Category(categoryID, categoryName, mostPopularVideo1, averageLikes1, maxLikes1, maxViews1, mostPopularChannel)
   f. Pop(mostPopularVideo1, videoID)
   g. Pop2(PopularVideo2, videoID)
   h. Pop3(mostPopularChannel, ChannelID)
3. The schema labeled Favorites(userID, videoID, rank) includes a superkey as you can reach all attributes in the database from attributes userID and videoID. Since a superkey is included in the database already, the definition of our database satisfies the conditions for 3NF.

Our schema as illustrated above demonstrates our database follows 3NF. We chose 3NF over BCNF, because 3NF conserves functional dependency while BCNF does not during decomposition. 3NF is also less strict compared to BCNF which makes it easier to implement.

Database Relational Schema:

Relational Schema

User(userID: INT [Primary Key], FirstName: VARCHAR(MAX), LastName: VARCHAR(MAX), Email: VARCHAR(MAX), Password: VARCHAR(MAX), Birthday: DATE)

```
CREATE_TABLE USER (
        userID INT NOT NULL,
        FirstName VARCHAR(MAX),
        LastName VARCHAR(MAX),
        Email VARCHAR(MAX),
         Password VARCHAR(MAX) NOT NULL,
        Birthday DATE,
        PRIMARY KEY(userID)
);
```

Video(videoID: CHAR(11) [PK], channelID: CHAR(24) [FK to Channel.channelID], categoryID: INT [FK to Category.categoryID], title: VARCHAR(100), publishedDate: DATE, URL: VARCHAR(MAX), country: VARCHAR(MAX), tag: VARCHAR(MAX), description: VARCHAR(MAX))

```
CREATE_TABLE Video(
        videoID CHAR(11) NOT NULL,
        channelID CHAR(11),
        categoryID INT,
        title VARCHAR(100),
```

```
    publishedDate DATE,
    URL VARCHAR(MAX),
    country VARCHAR(MAX),
    tag VARCHAR(MAX),
    description VARCHAR(MAX),
    PRIMARY KEY(videoID),
    FOREIGN KEY(channelID) REFERENCES Channel ON DELETE SET NULL,
    FOREIGN KEY(categoryID) REFERENCES Category ON DELETE SET NULL
);
```

TrendingStats(trendDate: DATE [PK], videoID: CHAR(11) [FK to Video.videoID], likeCount: INT, dislikeCount: INT, viewCount: INT)

```
CREATE_TABLE TrendingStats(
    TrendDate DATE,
    videoID CHAR(11).
    likeCount INT,
    dislikeCount INT,
    viewCount INT,
    PRIMARY KEY (TrendDate, videoID)
    FOREIGN KEY (videoID) REFERENCES Video(videoID)  ON DELETE
    CASCADE
);
```

Favorite(userID: INT [FK to User.userID], videoID: CHAR(11) [FK to Video.videoID], rank: INT)

// Many to Many Relationship between Users and Videos:

```
CREATE_TABLE Favorite (
        userID INT NOT NULL,
        videoID  CHAR(11) NOT NULL,
        rank INT,
        PRIMARY KEY(userID, videoID)
        FOREIGN KEY (userID) REFERENCES Users(userID) ON DELETE CASCADE
        FOREIGN KEY (videoID) REFERENCES Videos(videoID) ON DELETE
CASCADE
);
```

Channel(channelID: CHAR(24) [Primary Key], channelName: VARCHAR(MAX), mostPopularVideo2: VARCHAR(MAX) [FK to Video.videoID], averageLikes2: INT, maxLikes2: INT, maxViews2: INT)

```
CREATE_TABLE Channel (
        channelID CHAR(24) NOT NULL,
        channelName VARCHAR(MAX),
        mostPopularVideo2 INT,
```

averageLikes2 INT,

maxLikes2 INT,

maxViews2 INT

);

Category(categoryID: INT [PK], categoryName: VARCHAR(MAX), mostPopularVideo1: VARCHAR(MAX) [FK to Video.videoID], averageLikes1: INT, maxLikes1: INT, maxViews1: INT, mostPopularChannel: VARCHAR(MAX) [FK to Channel.channelID])

CREATE_TABLE Category (

categoryID INT NOT NULL,

categoryName VARCHAR(MAX),

FOREIGN KEY (mostPopularVideo1) REFERENCES Videos(videoID) ON DELETE

averageLikes1 INT,

maxLikes1 INT,

maxViews1 INT,

FOREIGN KEY (mostPopularChannel) REFERENCES Channel(channelID) ON DELETE

);

Relationship Descriptions

1. User ⇔ Video
    a. Many-to-Many relationship
    b. A user can have multiple favorite videos, and each video can be a favorite of multiple users
2. Video ⇔ Category
    a. 1-to-Many relationship
    b. Each video has one category, each category can have multiple videos
3. Video ⇔ Channel
    a. 1-to-Many relationship
    b. Each video has one channel, each channel can have multiple videos
4. Video ⇔ TrendingStats
    a. Weak entity (TrendingStats)
    b. 1-to-Many relationship
    c. Each video can have multiple trending stats (stats are by date)