YOUTUBE DATABASE

Information Management II Project

OVERVIEW

The database I have created is designed to store information regarding videos on the video sharing site youtube.com. The database stores information on channels (users), videos, comments and playlists. I attempted to follow the structure of YouTube as close as possible. As such, the database is modelled relationally with many of the other tables relating to the channel table. The attributes and relations can be viewed in the diagrams below.

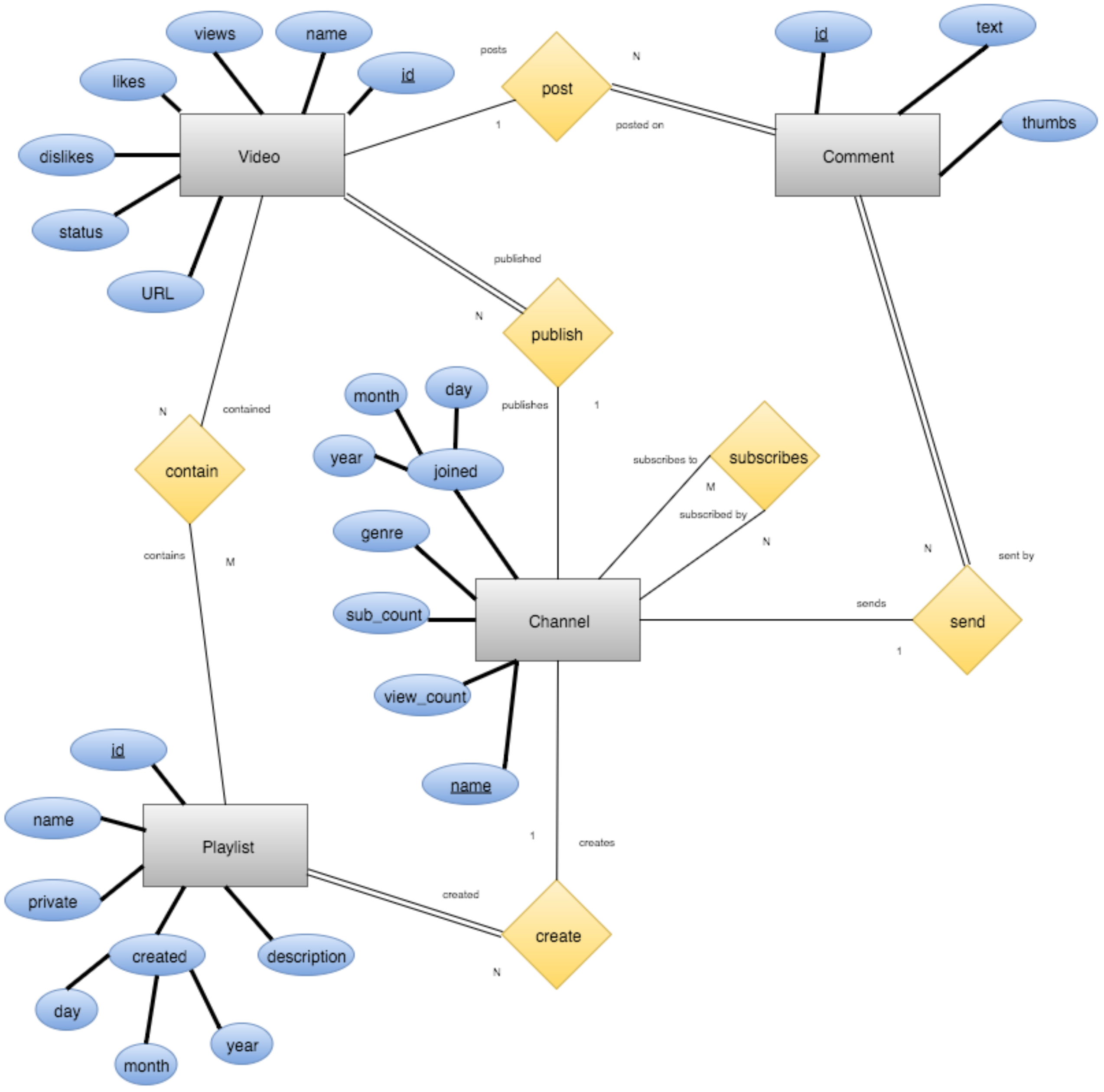
The channel table contains the channel name, the date on which the channel joined YouTube, the number of subscriptions, the number of views on all videos, the channel's genre, the channels that it is subscribed to and the channels that are subscribed to it. A channel can publish many videos, create many playlists, send many comments, subscribe to many other channels and have many channels subscribed to it.

The video table contains a video ID and the video name. It uses an ID because two videos can have the same name. It also contains the number of views, the number of likes and dislikes, the channel that published the video, the status of the video, the playlists that it belongs to and the URL. A video is published by only one channel, can post many comments and can be contained in many playlists.

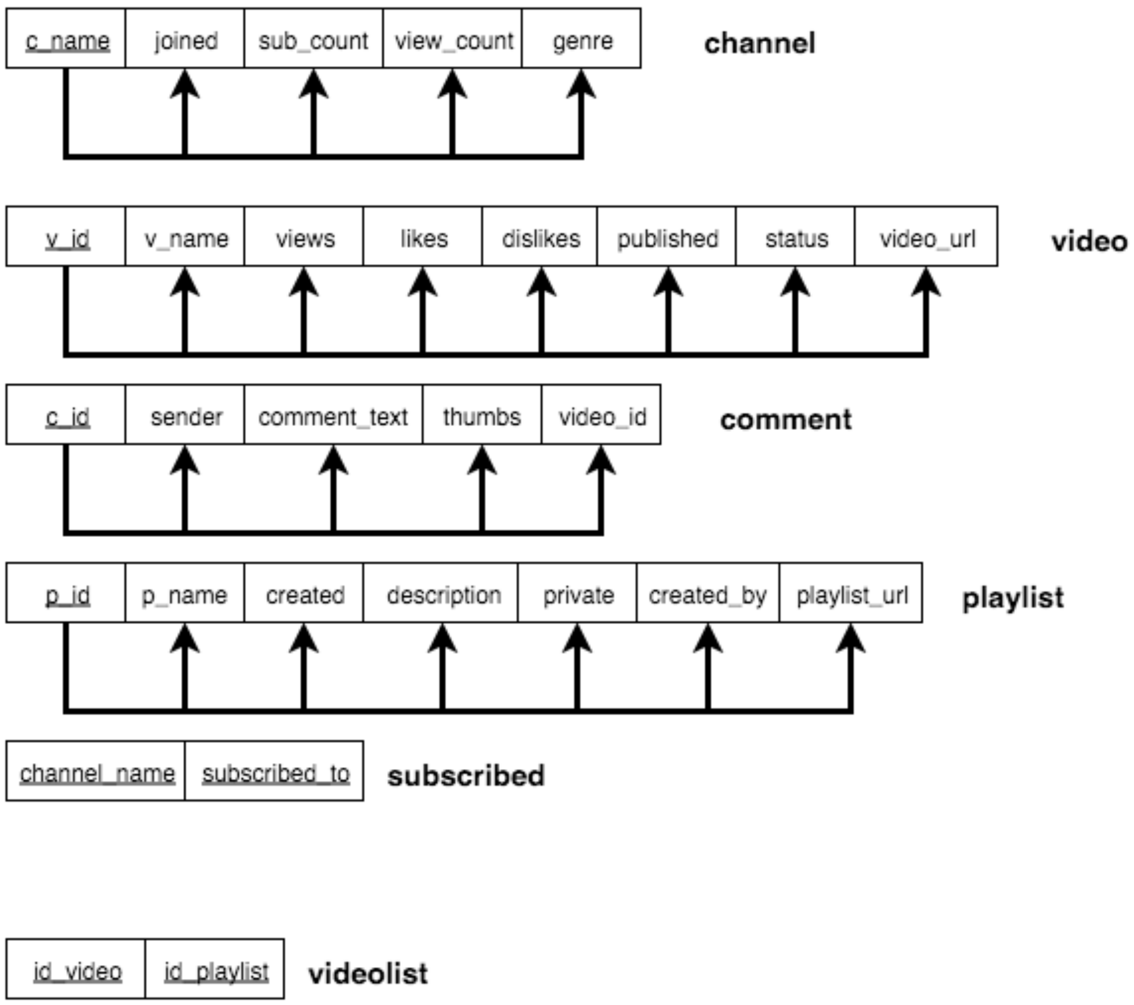
The comments table contains an ID, the sender of the comment, the text, its rating (represented by 'thumbs') and the ID of the video it is posted on. Multiple comments can be posted on a video and is sent by only one channel.

The playlist table contains an ID, the name, the date on which it was created, the description of the playlist, the privacy setting, the channel that created it, the videos in the playlist and the URL of the playlist. A playlist is created by only one channel and a playlist contains many videos.

Entity Relationship Diagram



Normalised Functional Dependencies



*underlined attributes are primary keys

To fully normalise the database, the subscribed to and subscribed by attributes were removed from the channel table and placed in a separate table (subscribed). This accounts for the fact that channels can be subscribed to many channels and channels have many subscribers that are channels. A similar method was used for the playlist attribute of the video table and the videos attribute of the playlist table. This is because any number of videos can be in a playlist and a video can be in any number of playlists.

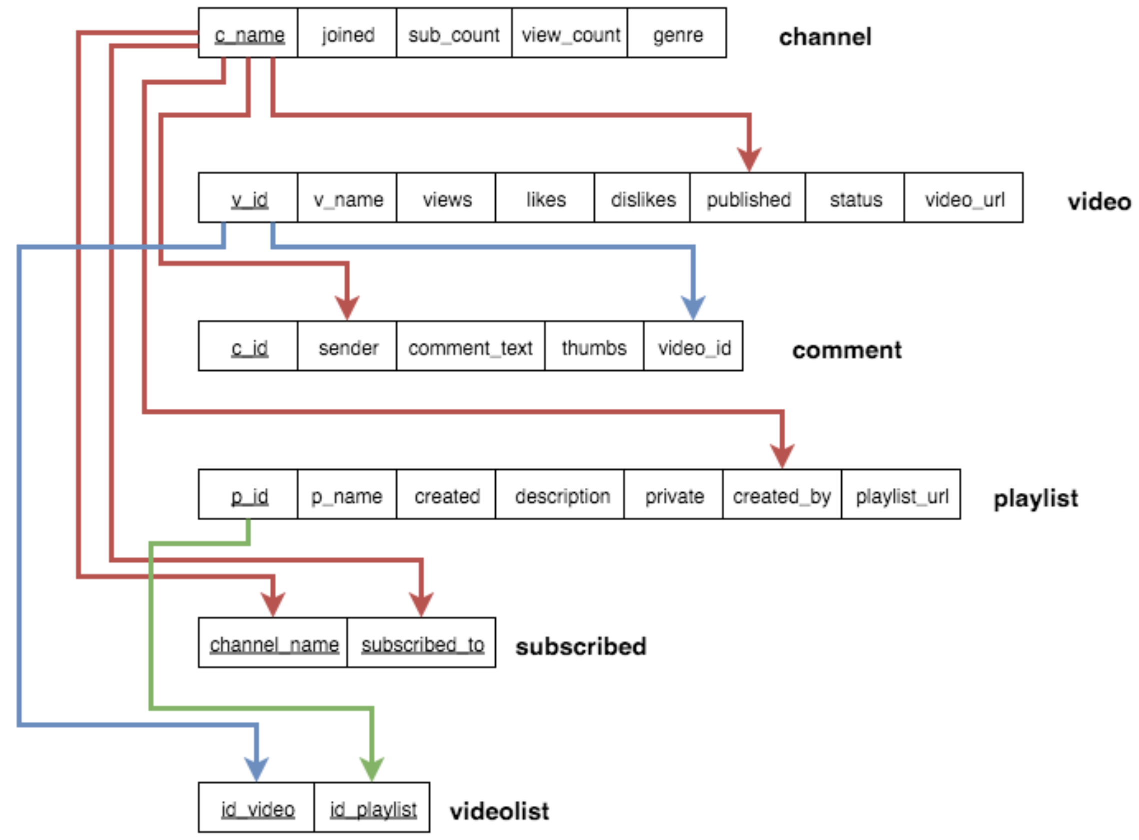
This design of the database is in first normal form because all the attributes are atomic, i.e. they cannot be cut down.

This design of the database is second normal form compliant because it is first normal form compliant and every non-column is fully functionally dependent on the entire primary key.

This design of the database is third normal form compliant because it is second normal form compliant and no non-key attributes are transitively dependent upon the primary key.

This design of the database is Boyce-Codd Normal Form compliant as there is no duplication of rows in any table.

Relational Schema



*underlined attributes are primary keys and the arrows point to foreign keys

Semantic Constraints

One example of a semantic is the status of videos. The status of a video on YouTube is one of 'PRIVATE', 'UNLISTED' or 'PUBLIC'. These indicate who can access the video. PRIVATE means that only the publisher can view the video. UNLISTED means that the video doesn't appear on the channel's feed but it can be accessed by people who have the link. PUBLIC means that anyone can access the video. The semantic constraint ensures that status attribute is one of PRIVATE, UNLISTED and PUBLIC. Other examples include making the URLs for videos and playlists have to be unique, checking privacy in playlist is either 'TRUE' or 'FALSE' and that many attributes are NOT NULL.

Security

For security I would establish access controls to different users. This can help some private information remain private. For example, I would grant access to YouTube administrators and moderators more access to and control of information than regular viewers of videos. For example:

```
GRANT WRITE, UPDATE, READ ON video TO admin_1 WITH GRANT OPTION;
```

However, viewers should only be able to view information about the videos, without editing or changing the data in the tables:

```
GRANT READ ON video TO viewer_1;
```

If any administrator abuses their privilege it can easily be revoked:

```
REVOKE WRITE, UPDATE ON video FROM admin_1;
```

Using the above method can easily be adjusted for every member making it very flexible. Another method of adding security is views. A view is a virtual table based on the result of an SQL statement. Views can limit the information that people can see. For example, if you only want to display the public playlists (the playlists whose private attribute is 'FALSE') then we can set up the following view:

```
CREATE VIEW only_public_playlist AS
     SELECT c_name, p_name
     FROM channel, playlist
     WHERE private = 'FALSE' AND c_name = playlist.created_by;
```

Relational Select and Table Joins

An example of a select statement can be seen in the view above. Example is selecting all the videos that are in a playlist and output their name along with the name of the playlist that they belong to. To do this we have to access three tables – playlist, video and videolist:

```
SELECT p_name, v_name
     FROM playlist, video, videolist
     WHERE p_id = id_playlist AND v_id = id_video
     ORDER BY p_name ASC, v_name ASC;
```

An example of joining tables is selecting as much information about a video as possible by combining two tables. We can select the video creator, genre, name, views and likes from the channel table and the video table:

```
SELECT channel.c_name, channel.genre, video.v_name, video.views,
video.likes
     FROM channel
     INNER JOIN video
     ON channel.c_name = video.published;
```

Update Operations

The YouTube database will have to be updated regularly due to the nature of the website. For example, view and like counts will have to be updated whenever a viewer clicks the like button:

```
UPDATE video
     SET views= 13464567, likes= 290012
     WHERE published= 'PewDiePie';
```

Channels can also change the name of their videos if they wish:

```
UPDATE video
     SET v_name=‘Roller Coaster... of Death!!!!’
     WHERE v_name= ' Roller Coaster... of Death!!1!';
```

Triggers

A good example for a trigger is when a video is deleted. When a video is deleted from a database, it follows that all the comments for that video should also be deleted. If not, then the foreign key in the comment table may point to a value that does not exist. So we must select the ID of the video that is to be deleted and if a comment has that value in the video_id attribute then it should be deleted:

```
CREATE TRIGGER DeleteVideoComments
     BEFORE DELETE ON video
     FOR EACH ROW
DECLARE
     video_name_id NUMBER;
BEGIN
     video_name_id := :OLD.v_id;
     DELETE FROM comments
     WHERE video_name_id = video_id;
END DeleteVideoComments;
.
RUN;
```

APPENDIX

```
CREATE TABLE channel (
            c_name VARCHAR(50) NOT NULL,
            joined DATE NOT NULL,
            sub_count NUMBER NOT NULL,
            view_count NUMBER NOT NULL,
            genre VARCHAR(225) NOT NULL,
            PRIMARY KEY(c_name)
);

INSERT INTO channel VALUES
('PewDiePie',TO_DATE('29/04/2010','dd/mm/yyyy'), 3, 19014393,'Games');
INSERT INTO channel VALUES ('smosh', TO_DATE('19/11/2005',
'dd/mm/yyyy'), 4, 7720313, 'Shows'  );

INSERT INTO channel VALUES ('Fine Brothers
Entertainment',TO_DATE('04/06/2007', 'dd/mm/yyyy'), 2,
2310015,'Film');

INSERT INTO channel VALUES ('Spinnin Records',TO_DATE('12/07/2007',
'dd/mm/yyyy'), 2,590448,'Music');

INSERT INTO channel VALUES ('CollegeHumor',TO_DATE('09/10/2006',
'dd/mm/yyyy'), 3, 2134746,'Shows');

CREATE TABLE video (
            v_id NUMBER NOT NULL,
            v_name VARCHAR(50) NOT NULL,
            views NUMBER NOT NULL,
            likes NUMBER NOT NULL,
            dislikes NUMBER NOT NULL,
            published VARCHAR(50) NOT NULL,
            status VARCHAR(8) NOT NULL,
            video_url VARCHAR(22) NOT NULL,
            PRIMARY KEY(v_id),
            FOREIGN KEY(published) REFERENCES channel(c_name),
            CONSTRAINT status_name CHECK (status IN ('PUBLIC',
'PRIVATE', 'UNLISTED')),
            CONSTRAINT unique_v_url UNIQUE (video_url)
);

INSERT INTO video VALUES (0,'10 BILLION MONTAGE!',13463373, 289978,
4119, 'PewDiePie','PUBLIC', 'youtube.com/v/aushfbye');

INSERT INTO video VALUES (1,'Roller Coaster... of
Death!!1!',2416643,127395,1405,'PewDiePie','PUBLIC','youtube.com/v/apd
kwicn');

INSERT INTO video VALUES (2,'SWEDEN SIMULATOR',3134377,158991,1957,
'PewDiePie','PUBLIC','youtube.com/v/mzidhue');

INSERT INTO video VALUES (3,'EVERY JEDI
EVER',1969255,70539,2124,'smosh','PUBLIC','youtube.com/v/plandhyt');

INSERT INTO video VALUES (4,'THE BAD PARTS OF
HEAVEN',3142794,71023,5398,'smosh','PUBLIC','youtube.com/v/hfudisua');

INSERT INTO video VALUES (5,'MOVIE TRANSLATION
FAILS',2608264,76706,4408,'smosh','PUBLIC','youtube.com/v/ppalddsa');

INSERT INTO video VALUES (6,'STAR WARS IN 1 TAKE IN 6
MINUTES',523437,13746,866,'Fine Brothers
Entertainment','PUBLIC','youtube.com/v/ascsfsuh');

INSERT INTO video VALUES (7,'PARENTS REACT TO SELFIE
STICKS',1081202,29312,421,'Fine Brothers
Entertainment','PUBLIC','youtube.com/v/lapduhss');

INSERT INTO video VALUES (8,'50 YOUTUBE SPOILERS IN 4 MINUTES (OCTOBER
2015)',715376,13668,339,'Fine Brothers
Entertainment','PUBLIC','youtube.com/v/gahsgtef');

INSERT INTO video VALUES (9,'HI-LO - Ooh La La (Official Music
Video)',118968,5554,841,'Spinnin
Records','PUBLIC','youtube.com/v/fhdjdd');

INSERT INTO video VALUES (10,'LVNDSCAPE ft. Joel Baker - Speeches
(Lyric Video)',269840,7962,261,'Spinnin
Records','PUBLIC','youuube.com/v/ploanvcx');

INSERT INTO video VALUES (11,'Blasterjaxx - Heartbreak
[Trailer]',201640,7391,205,'Spinnin
Records','PUBLIC','youtube.com/v/kifhdgyr');
```

```
INSERT INTO video VALUES (12,'Everyone Is Waiting To Talk About
Themselves',440085,18581,471,'CollegeHumor','PUBLIC','youtube.com/v/at
sgdvye');

INSERT INTO video VALUES (13,'If People Acted Like They Do In
Cars',747639,19164,3411,'CollegeHumor','PUBLIC','youtube.com/v/tdufnvh
r');

INSERT INTO video VALUES (14,'Why The Electoral College Ruins
Democracy',947022,32471,644,'CollegeHumor','PUBLIC','youtube.com/v/ufh
dydgd');

INSERT INTO video VALUES
(15,'Unpublished',0,0,0,'PewDiePie','PRIVATE','youtube.com/v/yshdbufi'
);

INSERT INTO video VALUES
(16,'Unpublished',0,0,0,'PewDiePie','PRIVATE','youtube.com/v/jkshdbfi'
);

INSERT INTO video VALUES
(17,'Unpublished',0,0,0,'PewDiePie','PRIVATE','youtube.com/v/quwyetsf'
);


CREATE TABLE comments(
            c_id NUMBER NOT NULL,
            sender VARCHAR(50) NOT NULL,
            comment_text VARCHAR(255) NOT NULL,
            thumbs NUMBER NOT NULL,
            video_id NUMBER NOT NULL,
            PRIMARY KEY(c_id),
            FOREIGN KEY(sender) REFERENCES channel(c_name),
            FOREIGN KEY(video_id) REFERENCES video(v_id)
);

INSERT INTO comments VALUES (0,'smosh','Because thats how we got
George W. Bush.',382,14);
INSERT INTO comments VALUES (1,'smosh','This is my fave Pewds
vid',113,0);
INSERT INTO comments VALUES (2,'smosh','In total on every video',0,0);
INSERT INTO comments VALUES (3,'PewDiePie','Wait there are only 7.8
billion people on earth',5,0);
INSERT INTO comments VALUES (4,'PewDiePie','Isnt that Corys mom...
xD',6,12);

CREATE TABLE playlist(
            p_id NUMBER NOT NULL,
            p_name VARCHAR(255) NOT NULL,
            created DATE NOT NULL,
            description VARCHAR(50) NOT NULL,
            private VARCHAR(5) NOT NULL,
            created_by VARCHAR(50) NOT NULL,
            playlist_url VARCHAR(22) NOT NULL,
            PRIMARY KEY(p_id),
            FOREIGN KEY(created_by) REFERENCES channel(c_name),
            CONSTRAINT boolean CHECK (private IN ('TRUE', 'FALSE')),
            CONSTRAINT unique_p_url UNIQUE(playlist_url)
);

INSERT INTO playlist VALUES (0,'November 2015
Playlist',TO_DATE('01/11/2015', 'dd/mm/yyyy'), 'Videos uploaded
November 2015','FALSE','PewDiePie','youtube.com/p/hdbystfg');

INSERT INTO playlist VALUES (1,'Prank Videos',TO_DATE('13/02/2006',
'dd/mm/yyyy'),'Videos where we prank
people','FALSE','smosh','youtube.com/p/ahysgydu');

INSERT INTO playlist VALUES (2,'Liked Videos',TO_DATE('19/11/2005',
'dd/mm/yyyy'),'Personal
favourites','TRUE','smosh','youtube.com/p/jdufhgyb');

INSERT INTO playlist VALUES (3,'Dubstep',TO_DATE('22/07/2010',
'dd/mm/yyyy'),'All songs are dubstep','FALSE','Spinnin
Records','youtube.com/p/qsvhdnig');

INSERT INTO playlist VALUES (4,'Unpublished',TO_DATE('04/06/2010',
'dd/mm/yyyy'),'Videos to be
released','TRUE','PewDiePie','youtube.com/p/llkfmspo');

CREATE TABLE subscribed (
            channel_name VARCHAR(50) NOT NULL,
            subscribed_to VARCHAR(50) NOT NULL,
            FOREIGN KEY (channel_name) REFERENCES channel(c_name),
            FOREIGN KEY (subscribed_to) REFERENCES channel(c_name),
            CONSTRAINT subscribed_key PRIMARY KEY(channel_name,
subscribed_to),
            CONSTRAINT notEquality CHECK (channel_name !=
subscribed_to)
);

INSERT INTO subscribed VALUES ('PewDiePie','smosh');
INSERT INTO subscribed VALUES ('PewDiePie','CollegeHumor');
INSERT INTO subscribed VALUES ('smosh','Fine Brothers Entertainment');
INSERT INTO subscribed VALUES ('smosh','CollegeHumor');
INSERT INTO subscribed VALUES ('smosh','Spinnin Records');
INSERT INTO subscribed VALUES ('Fine Brothers
Entertainment','PewDiePie');
INSERT INTO subscribed VALUES ('Fine Brothers Entertainment','smosh');
INSERT INTO subscribed VALUES ('Spinnin Records','PewDiePie');
INSERT INTO subscribed VALUES ('Spinnin Records','smosh');
INSERT INTO subscribed VALUES ('Spinnin Records','CollegeHumor');
INSERT INTO subscribed VALUES ('CollegeHumor','PewDiePie');
INSERT INTO subscribed VALUES ('CollegeHumor','smosh');
INSERT INTO subscribed VALUES ('CollegeHumor','Fine Brothers
Entertainment');
INSERT INTO subscribed VALUES ('CollegeHumor','Spinnin Records');

CREATE TABLE videolist  (
            id_video NUMBER NOT NULL,
            id_playlist NUMBER NOT NULL,
            FOREIGN KEY (id_video) REFERENCES video(v_id),
            FOREIGN KEY (id_playlist) REFERENCES playlist(p_id),
            CONSTRAINT videoList_key PRIMARY KEY(id_video, id_playlist)
);

INSERT INTO videolist VALUES (0,0);
INSERT INTO videolist VALUES (1,0);
INSERT INTO videolist VALUES (2,0);
INSERT INTO videolist VALUES (3,1);
INSERT INTO videolist VALUES (4,1);
INSERT INTO videolist VALUES (0,2);
INSERT INTO videolist VALUES (2,2);
INSERT INTO videolist VALUES (12,2);
INSERT INTO videolist VALUES (13,2);
INSERT INTO videolist VALUES (14,2);
INSERT INTO videolist VALUES (9,3);
INSERT INTO videolist VALUES (10,3);
INSERT INTO videolist VALUES (11,3);
INSERT INTO videolist VALUES (15,4);
INSERT INTO videolist VALUES (16,4);
INSERT INTO videolist VALUES (17,4);
```