yw3623 - yanlin.wang

yl4754 - yaohong.liang

PostgreSQL: yl4754

Users-Audience-Video_producer(ISA) (Entity) Table

In [ ]:
```
CREATE TABLE Users(
    User_id varchar(20) PRIMARY KEY,
    User_name varchar(30),
    Email varchar(255),
    Address varchar(20),
    Birthdate varchar(10),
    CHECK(Email LIKE '%_@__%.__%')
)
```

In [ ]:
```
CREATE TABLE Audience(
    User_id varchar(20) PRIMARY KEY,
    Num_of_subscriptions int,
    Num_of_given_likes int,
    Num_of_given_reviews int,
    FOREIGN KEY (User_id) references Users
    ON DELETE CASCADE
)
```

In [ ]:
```
CREATE TABLE Video_producer(
    User_id varchar(20) PRIMARY KEY,
    Num_of_launched_video int,
    Total_videoplay_amount int,
    Num_of_subscribers int,
    Num_of_received_likes int,
    Num_of_received_reviews int,
    FOREIGN KEY (User_id) references Users
    ON DELETE CASCADE
)
```

Video quality auditor (Entity) Table

In [ ]:
```
CREATE TABLE Video_quality_auditor(
    Auditor_id varchar(20) PRIMARY KEY,
    First_name varchar(20),
    Last_name varchar(20),
    Gender varchar(10)
)
```

Review_comment (Aggregation) Table

Notes: Only when an user make comments, can a review exist. A user can make many reviews. The relationship is 1-to-many relationship with total participation of review.

In [ ]:
```
#The aggregation showing that a review must been made by a user.
CREATE TABLE Review_comment(
    User_id varchar(20),
    Review_id varchar(25),
    Content text,
    Datetime date,
    PRIMARY KEY (Review_id),
    FOREIGN KEY (User_id) references Users
)
```

Review_associate (Relationship) Table

Notes: A review has to be associated to one video, and one video can have many reviews associated. The relationship is 1-to-many relationship with total participation of review.

In [ ]:
```
CREATE TABLE Review_associate(
    Content_id varchar(30),
    Review_id varchar(20),
    Auditor_id varchar(20),
    PRIMARY KEY (Review_id,Auditor_id,Content_id),
    FOREIGN KEY (Content_id,Auditor_id) references Approved_video,
    FOREIGN KEY (Review_id) references Review_Comment
    ON DELETE NO ACTION
)

#This CHECK constraint has not yet been added into the PostgreSQL, because
#we did not learn trigger yet.
#Each review has to be associated with one uploaded and approved video.
CREATE ASSERTION reviewasso
CHECK(
    NOT EXISTS(
        SELECT Review_id FROM Review_Comment
        WHERE Review_id NOT IN (
              SELECT  Review_id FROM Review_associate
        )
    )
)
```

Ranking (Entity) Table

In [ ]:
```
CREATE TABLE Ranking(
    Title varchar(50),
    Release_year varchar(4),
    Region varchar(20),
        PRIMARY KEY (Title, Release_year)
)
```

Selected (Relationship) Table

In [ ]:
```
CREATE TABLE Selected(
    Title varchar(50),
    Release_year varchar(20),
    Content_id varchar(20),
    Auditor_id varchar(20),
    PRIMARY KEY (Title,Release_year,Auditor_id,Content_id),
    FOREIGN KEY (Content_id,Auditor_id) references Approved_video
    ON DELETE NO ACTION
)
```

Subscribe (Relationship) Table

In [ ]:
```
CREATE TABLE Subscribe(
    Audience_id varchar(20),
    Producer_id varchar(20),
    FOREIGN KEY (Audience_id) references Audience,
    FOREIGN KEY (Producer_id) references Video_producer,
    PRIMARY KEY (Audience_id,Producer_id)
)
```

Watch (Relationship) Table

In [ ]:
```
CREATE TABLE Watch(
    User_id varchar(20),
    Duration varchar(30),
    Datetime varchar(30),
    Content_id varchar(30),
    Auditor_id varchar(20),
    FOREIGN KEY (User_id) references Audience,
    FOREIGN KEY (Content_id,Auditor_id) references Approved_video,
    PRIMARY KEY (User_id,Content_id,Auditor_id)
)
```

Uploaded_approved (Aggregation) Table

Notes: We use this table to describe the relationship between auditors, videos and producers. It shows that all the video shared on this website must be uploaded by producer first and then approved by auditors.

In [ ]:
```
# delect Datetime for Upload
# transfer Release_datetime to Approved_video
# DELECT line 7 data b/c of UTF-8 formatting problem
CREATE TABLE Uploaded_video(
    User_id varchar(20),
    Content_id varchar(30),
    Video_title varchar(255),
    Hyperlink text,
    Genre varchar(20),
    Length varchar(30),
    Num_of_likes int,
    Num_of_reviews int,
    Play_amount int,
    FOREIGN KEY (User_id) references Video_producer,
    PRIMARY KEY (Content_id)
)

# wait to do
CREATE ASSERTION videoprod
CHECK(
    NOT EXISTS(
        SELECT User_id FROM Video_producer
        WHERE User_id NOT IN (
              SELECT  User_id FROM Uploaded_video
        )
    )
)

CREATE TABLE Approved_video(
    Content_id varchar(30),
    Auditor_id varchar(20),
    Release_datetime varchar(30),
    FOREIGN KEY (Content_id) references Uploaded_video,
    FOREIGN KEY (Auditor_id) references Video_quality_auditor,
    PRIMARY KEY (Content_id, Auditor_id)
)


# wait to do
CREATE ASSERTION videoappro
CHECK(
    NOT EXISTS(
        SELECT Content_id FROM Uploaded_video
        WHERE Content_id NOT IN (
              SELECT  Content_id FROM Approved_video
        )
    )
)
```

3 Interesting Queries

Query 1: Get the percentage of videos that have been approved.

We use `with` statement to obtain number of videos uploaded from `uploaded_video` and number of videos approved from `approved_video`, respectively. Then, we calculate the percentage in t.

In [ ]:
```
WITH cte AS (
SELECT
        COUNT(*) as num_approved_video
FROM Approved_video
),
cte2 AS (
SELECT
        CAST(COUNT(*) AS FLOAT) as num_uploaded_video
FROM uploaded_video
)
SELECT
        ROUND(CAST((cte.num_approved_video/cte2.num_uploaded_video)*100 AS NUMERIC), 2) AS approved_per
centage
FROM cte, cte2
```

Query 2: Find the name of users whose uploaded videos were not approved.

We first obtain the name of users whose videos have been approved. Then, we select user names not in those.

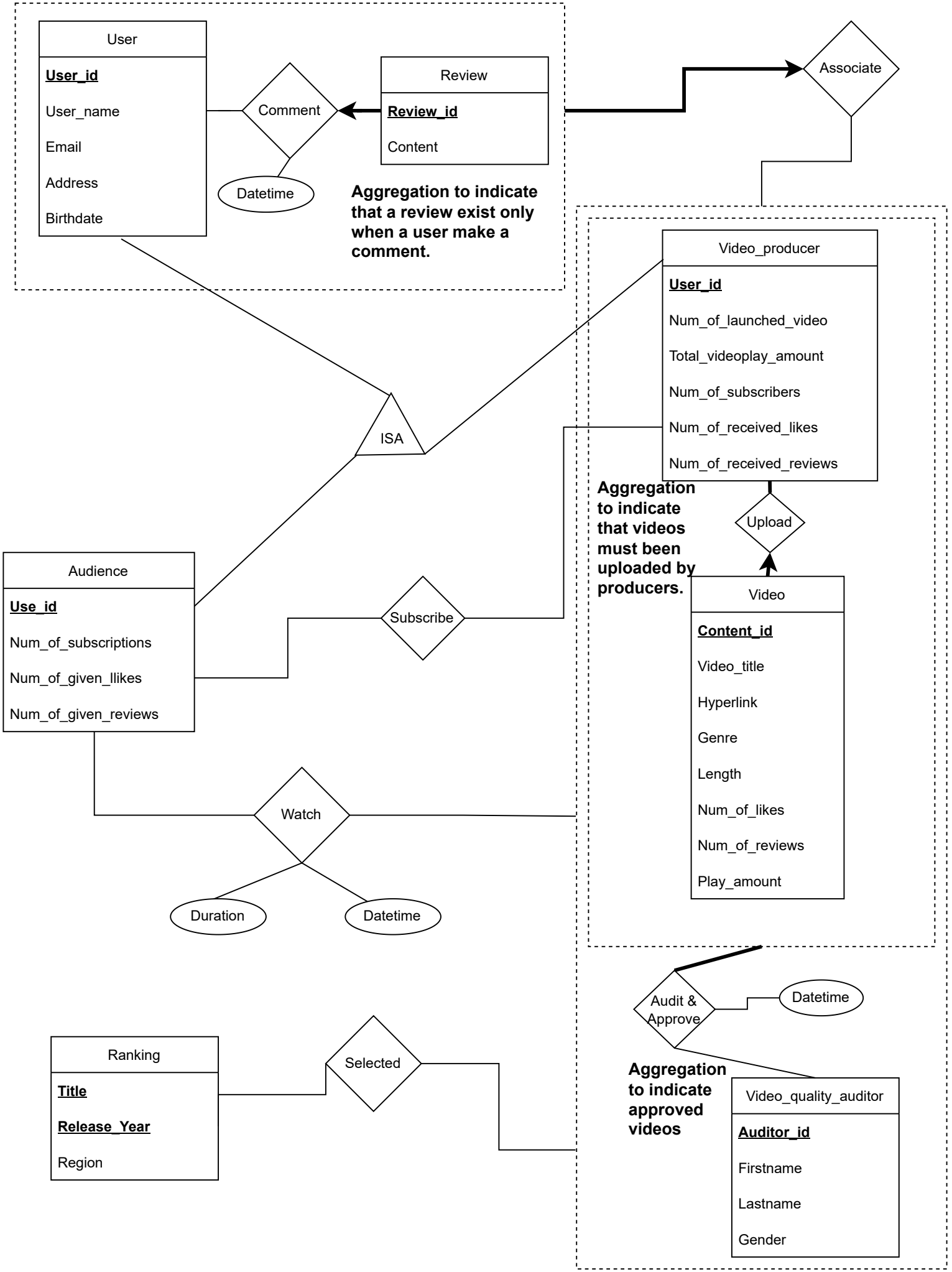
In [ ]:
```
WITH cte AS (
SELECT
        c.user_name
FROM Approved_video a
LEFT JOIN uploaded_video b ON a.content_id = b.content_id
JOIN Users c ON b.User_id = c.User_id
)
SELECT
        user_name
FROM Users
WHERE user_name NOT IN (SELECT * FROM cte)
```

Query 3: What are all the Genre of uploaded videos that has been approved by auditor?

We check whether the content_id of uploaded videos appear in the approved_video table. And if it exists, we output the Genre from uploaded_video and delete duplicates.

In [ ]:
```
SELECT [DISTINCT] Genre
FROM uploaded_video
WHERE Content_id IN(
        SELECT Content_id FROM approved_video)

#We check whether the content_id of uploaded videos appear in the approved_video
#table. And if it exists, we output the Genre from uploaded_video and
#delete duplicates.
```


User
User_id
User_name
Email
Address
Birthdate
Comment
Review
Review_id
Content
Datetime
Aggregation to indicate that a review exist only when a user make a comment.
Associate
Video_producer
User_id
Num_of_launched_video
Total_videoplay_amount
Num_of_subscribers
Num_of_received_likes
Num_of_received_reviews
ISA
Aggregation to indicate that videos must been uploaded by producers.
Upload
Video
Content_id
Video_title
Hyperlink
Genre
Length
Num_of_likes
Num_of_reviews
Play_amount
Audience
Use_id
Num_of_subscriptions
Num_of_given_llikes
Num_of_given_reviews
Subscribe
Watch
Duration
Datetime
Audit & Approve
Datetime
Aggregation to indicate approved videos
Video_quality_auditor
Auditor_id
Firstname
Lastname
Gender
Ranking
Title
Release_Year
Region
Selected