
YouTube
DaSaTBaSe ProjecT

Egypt Russian University

Faculty of Management, Economic and Business Technology

Database System2 project

IST207

Project Title

(YouTube database System)

Team Nickname

(كله under control)

❖ Team Members

NO	ID	Name	Notes (By doctor)
1	224015	Mariam Tamer Abdullatif	
2	224013	Ahmed Mohmed Refaat	
3	224044	Mahmoud Mohamed Abdullatif	
4	224052	Toka Khaled Mohamed	
5	224177	Amr Mohamed Ahmed	
6	224201	Menna Allah Atef	

❖ Notes by Doctor

Supervised by:

Dr. Reham Abdallah

(Spring 2024)

✓ Business Rules

User Create many channels, Channel must be created by one user.
Many Channels must have analytics, analytics must be had channels.
Many users can watch many advertisements, advertisements have many contents.
Many users view content, many content viewed by user.
Users can make a play list, play list must be made by one user.
Users can make a report, a report must be made by one user.
Many users can watch many trends, trending watched by many users.
Trending must have many videos, video can has trending.
Many users can add to many watch_later, watch_later added to from many user.
Many watch_later added video, many video added to watch_later.
Many user can add to downloads, downloads must be added to by user.
Downloads can added to many video, many video added to downloads.
User write many comments , comment must be writed by one user.
Many comment has content , one content had by many comment.
Content is the superclass with Shorts, Live, and Video as its subclasses.one live contain live_chat, one live_chat contained by live.
Many content can has many channel , Many channels has many content.

✓ Functional Requirements:

Functional requirements for a YouTube-like application specify the behaviors, features, and functions the system must perform to meet user needs. Here are the essential functional requirements for such an application:

1_User Registration:

User Registration: Allow users to create accounts using email, phone number. Require email verification for account activation.

2_User Login: Provide login functionality with username/email and password.

3_Content Upload and Management

Video Upload: Enable users to upload videos with support for various file formats and sizes. Provide options for adding titles, descriptions, tags, and categories to videos.

Allow users to choose video privacy settings (public, unlisted, private).

4_Video Processing: Automatically process and encode uploaded videos for different resolutions and formats.

5_Search Functionality: Implement a search feature that allows users to search videos by title, description, tags, and categories.

Include advanced filters (upload date, view count, duration, etc.).

5_Recommendations:

Use algorithms to provide personalized video recommendations based on user preferences and viewing history

6_Player Features: Support playback controls (play, pause, rewind, fast forward).

Allow users to adjust playback quality based on available resolutions.

7_Include closed captions and subtitles:

Provide options for adjusting playback speed and enabling full-screen mode.

8_Comments and Replies:

Enable users to comment on videos and reply to other comments.

Implement moderation tools for users to report, delete, or disable comments.

9_Like and Dislike:

Allow users to like or dislike videos and comments. Display like/dislike counts on videos and comments.

10_Subscriptions: Enable users to subscribe to channels and receive notifications for new uploads. Allow users to manage their subscriptions and notification preferences.

11_Playlists: Provide functionality for users to create, manage, and share playlists.
12_Live Streaming: Support live streaming capabilities with real-time chat interaction. Allow streamers to manage chat, moderate comments, and highlight messages.
14_Advertising: Display ads in videos (pre-roll, mid-roll, post-roll, overlay). Provide options for users to skip ads after a certain duration.
15_Analytics and Reporting
Video Analytics: Provide detailed analytics for content creators, including views, watch time, audience demographics, and engagement metrics.
16_User Reports: Allow users to report inappropriate content, comments, and users.
17_Notification System:
User Notifications: Send notifications for new uploads, comments, likes, and replies. these functional requirements, a YouTube-like application can offer a comprehensive, user-friendly platform for video sharing, viewing, and interaction.

✓ *non-functional system requirements*

- Performance
- Security
- Scalability
- Ease of Use
- Accessibility
- User privacy

✓ *Relations*

- **User:**

  Has one-to-many relationship with Content (User creates content)
  Has one-to-many relationship with Playlist (User creates playlists)
  Has one-to-many relationship with Advertisement (User watches advertisements)
  Has one-to-many relationship with Watch Later (User adds videos to watch later)
  Has one-to-many relationship with Report (User makes reports)
  Has one-to-one relationship with Analytics (User has analytics data)
  Attributes: username, password, age, email, theme, ID, verified

- **Content:**

  Belongs to one-to-many relationship with Channel (Content is owned by a channel)
  Belongs to one-to-many relationship with User (Content is created by a user)
  Has one-to-many relationship with Playlist (Content is added to playlists)
  Has one-to-many relationship with Report (Content can be reported)
  Attributes: ID, content_type, creation_date, title, description, like_count, subtitles, caption

- **Channel**:

  Has one-to-many relationship with Content (Channel has content)
  Has one-to-one relationship with Analytics (Channel has analytics data)
  Attributes: channel_name, profile_pic, number_of_subscribers, number_of_viewers, number_of_videos, number_of_shorts, description, create_date, type

- **Playlist:**

  Belongs to one-to-many relationship with User (Playlist is created by a user)
  Belongs to one-to-many relationship with Content (Playlist contains content)
  Attributes: ID, name, apdate_date

- **Advertisement**:

  Belongs to one-to-many relationship with User (Advertisement is watched by a user)
  Attributes: ID, start_date, end_date, type, duration, clicks, cost.

- **Watch Later:**

  Belongs to one-to-many relationship with User (Video is added to watch later list by a user)
  Attributes: ID, date

- **Report**:

  Belongs to one-to-many relationship with User (Report is made by a user)
  Belongs to one-to-many relationship with Content (Report is about a content)
  Attributes: ID, description, date

- **Analytics**:

  Belongs to one-to-one relationship with User (Analytics data belongs to a user)
  Belongs to one-to-one relationship with Channel (Analytics data belongs to a channel)
  Attributes: ID

- **Live Chat:**

  Belongs to one-to-many relationship with Content (Live chat belongs to a live content)
  Attributes: message_content

- **Downloads:**

  Belongs to one-to-many relationship with User (Download is added by a user)
  Attributes: ID, size, quality, date

- **Trending:**

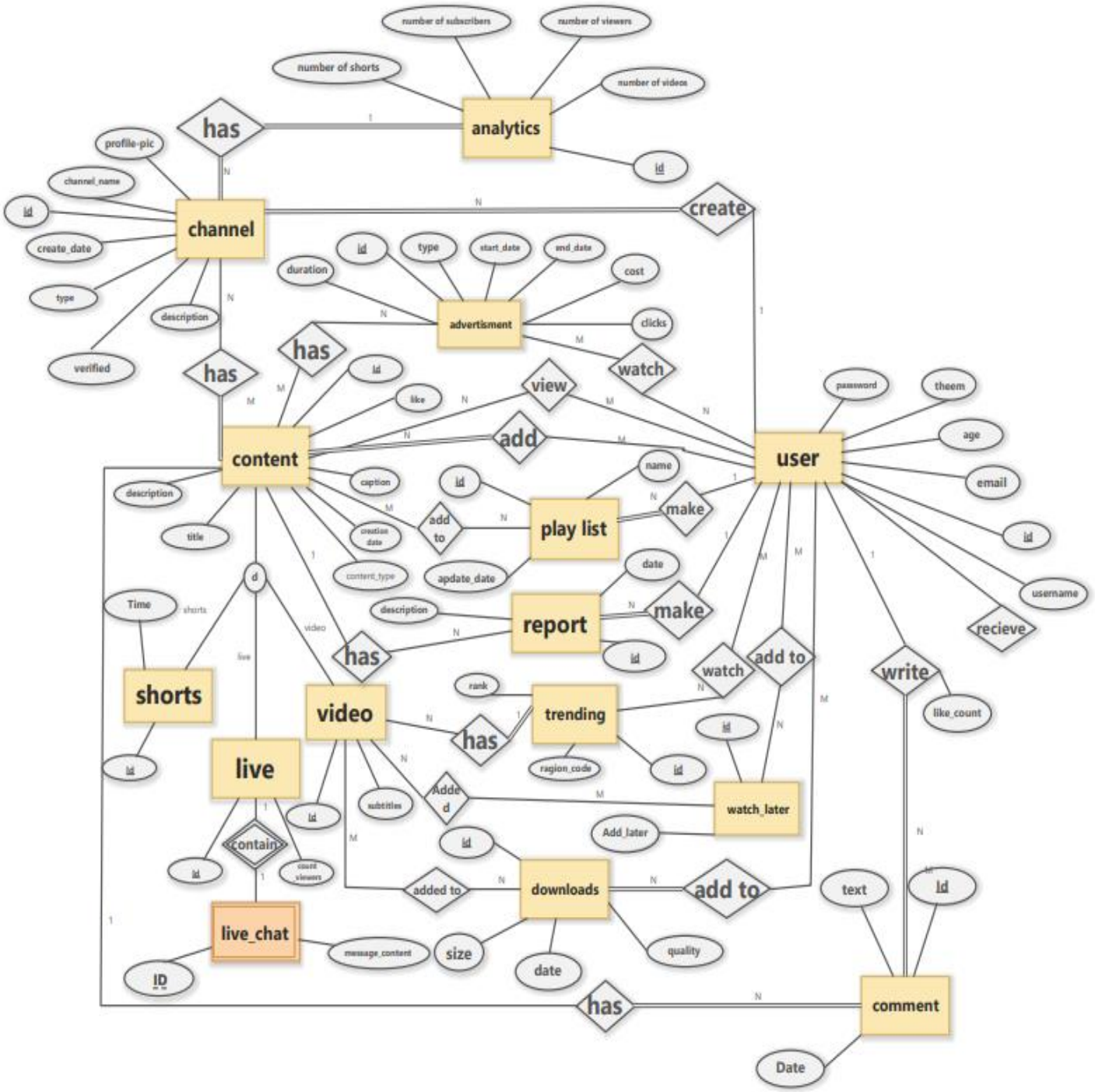
  Has one-to-many relationship with Content (Content is trending)
  Attributes: ID, rank, region_code

Egypt Russian University

Faculty of Management, Economic and Business Technology

✓ *Database Design*

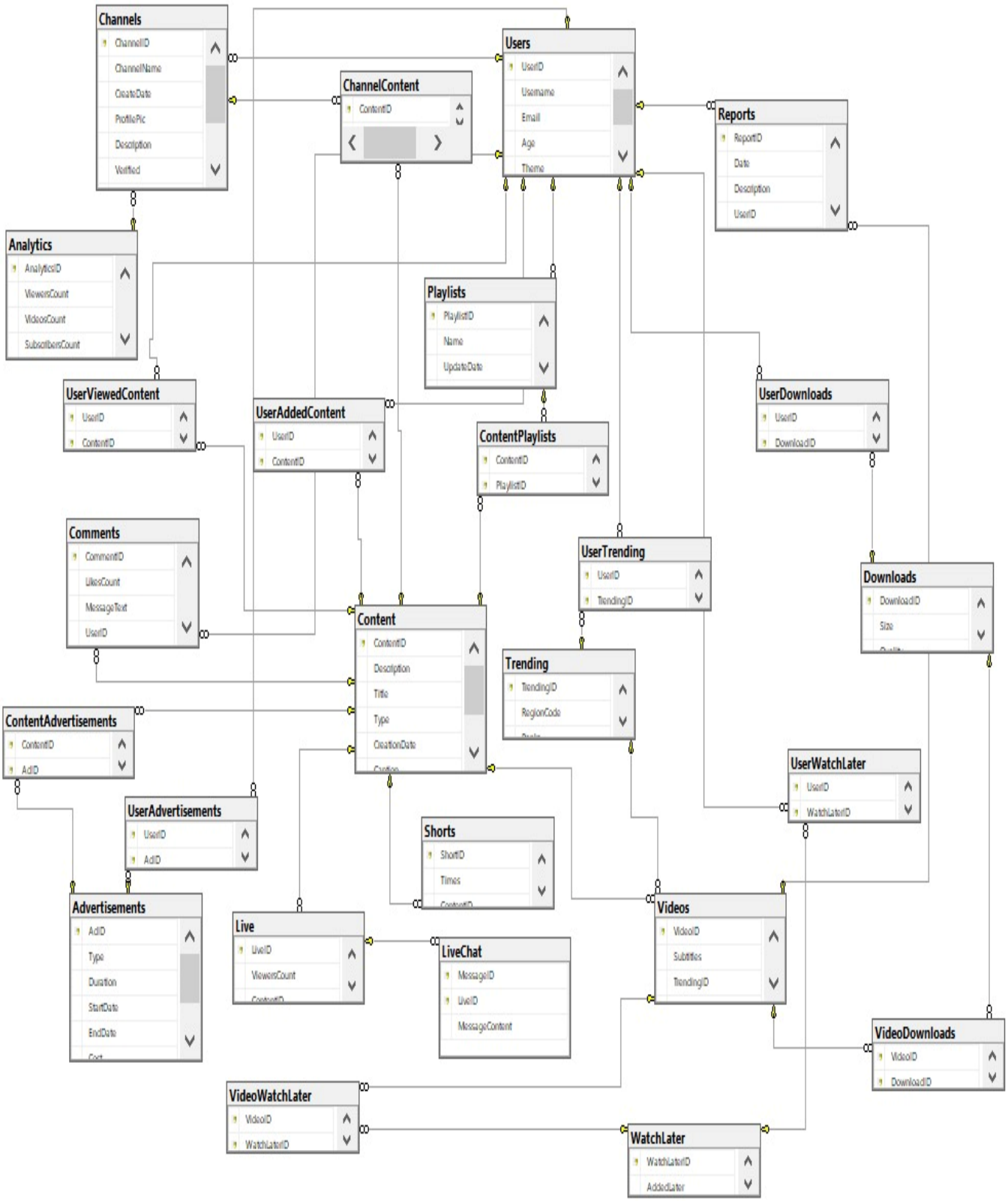
✓ ERD



Egypt Russian University

Faculty of Management, Economic and Business Technology

✓ Database Diagram



Egypt Russian University

Faculty of Management, Economic and Business Technology

✓ DB schema (Mapping)

User (1user_id, username, email, password, age, theme)

Comment (3comment_id, text, like_count, user_id1)

Watch_later (4watch_later_id, add_later)

Download (5download_id, size, quality)

Trending (6trend_id, region_code, rank)

Report (7report_id, date, description, user_id1, content_id12)

Play_list (8play_list_id, update_date, name, user_id1)

Advertisement (9adds_id, type, duration, start_date, end_date, cost, clicks)

Channel (10channel_id, channel name, create_date, profile_pic, type, description, verified, user_id1, analytics_id11)

Analytics (11analytics_id, no_of_viewer, no_of_videos, no_of_subs, no_of_shorts)

Content (12content_id, description, title, conten_type, creation_date, caption, like,)

Video_download (video_id13, download_id5)

Video (13video_id, subtitles, content_id12, trending_id6)

Shorts (14short_id, time, content_id12)

Live (15live_id, content_viewer, content_id12)

Live_chat (16livechat_id, message_content, live_id15)

User_download (user_id1, download_id5)

user_watch_later (user_id1, watch_later_id4)

user_trending (user_id1, trending_id6)

user_view_content (user_id1, content_id12)

user_add_content (user_id1, content_id12)

content_playlist (content_id12, playlist_id8)

video_watch_later (video_id13, watch_later_id4)

content_adds (content_id12, adds_id9)

channel_content (channel_id10, content_id12)

user_adds (user_id1, adds_id9)

✓ Database Implementation

✓ Join (more than two tables)

- This query selects the usernames of users who made comments, the message text of those comments, and the title of the content on which the comments were made by joining the Comments, Users, and Content tables.

```
SELECT Users.Username, Comments.MessageText, Content.Title
FROM Comments
JOIN Users ON Comments.UserID = Users.UserID
JOIN  ON  Comments.ContentID  =
Content.ContentID;
```

Results | Messages

	Username	MessageText	Title
1	user1	Great video!	SQL Tutorial
2	user2	Interesting content	Nature Documentary
3	user3	I enjoyed watching this	Machine Learning 101
4	user4	Very informative	SQL Tutorial
5	user5	Nice work!	Nature Documentary
6	user6	Keep it up!	Machine Learning 101
7	user7	This is awesome!	SQL Tutorial
8	user8	I learned a lot	Nature Documentary

- This query joins the Playlists table with the Users table based on the UserID column to link each playlist with its creator. It selects the playlist name, the username of the creator, and the update date of each playlist.

```
SELECT Playlists.Name, Users.Username, Playlists.UpdateDate
FROM Playlists
JOIN Users ON Playlists.UserID = Users.UserID;
```

Results | Messages

	Name	Username	UpdateDate
1	Favorites	user1	2024-01-01
2	Workout Mix	user2	2024-01-02
3	Chill Vibes	user3	2024-01-03
4	Study Focus	user4	2024-01-04
5	Party Hits	user5	2024-01-05
6	Road Trip	user6	2024-01-06
7	90s Classics	user7	2024-01-07
8	Indie Gems	user8	2024-01-08

- This query joins the Users table with the UserViewedContent table based on the UserID column to link each user with the content they've viewed. Then it joins the Content table to retrieve information about the viewed content, including its title and type. Finally, it selects the username of the user, the title of the content, and the type of content.

```
SELECT Users.Username, Content.Title, Content.Type
FROM Users
JOIN UserViewedContent ON Users.UserID = UserViewedContent.UserID
JOIN Content ON UserViewedContent.ContentID = Content.ContentID;
```

Results | Messages

	Username	Title	Type
1	user1	SQL Tutorial	Tutorial
2	user1	Nature Documentary	Documentary
3	user2	Machine Learning 101	Tutorial
4	user2	Italian Cooking Masterclass	Cooking
5	user3	Fashion Trends 2024	Fashion
6	user3	Ancient Civilizations	Documentary
7	user4	Photography Masterclass	Tutorial
8	user4	World Travel Diaries	Travel

- This query joins the Users table with the UserAdvertisements table based on the UserID column to link each user with the advertisements they've interacted with. Then it joins the Advertisements table to retrieve information about the advertisements, including their type and the number of clicks. Finally, it selects the username of the user, the type of advertisement, and the number of clicks.

```
SELECT Users.Username, Advertisements.Type, Advertisements.Clicks
FROM Users
JOIN UserAdvertisements ON Users.UserID = UserAdvertisements.UserID
JOIN Advertisements ON UserAdvertisements.AdID = Advertisements.AdID;
```

Results | Messages

	Username	Type	Clicks
1	user1	Banner	100
2	user2	Video	150
3	user3	Popup	80
4	user4	Text	60
5	user5	Banner	120
6	user6	Video	140
7	user7	Popup	90
8	user8	Text	70

- This query joins the Users table with the Reports table based on the UserID column to link each user with the reports they've made. Then it joins the Videos table to retrieve information about the reported videos, and finally, it joins the Content table to fetch the title of the reported video. The query selects the username of the user, the title of the reported video, and the description of the report.

```
SELECT Users.Username, Content.Title, Reports.Description
FROM Users
JOIN Reports ON Users.UserID = Reports.UserID
JOIN Videos ON Reports.VideoID = Videos.VideoID
JOIN Content ON Videos.ContentID = Content.ContentID;
```

Results | Messages

	Username	MessageText	Title
1	user1	Great video!	SQL Tutorial
2	user2	Interesting content	Nature Documentary
3	user3	I enjoyed watching this	Machine Learning 101
4	user4	Very informative	SQL Tutorial
5	user5	Nice work!	Nature Documentary
6	user6	Keep it up!	Machine Learning 101
7	user7	This is awesome!	SQL Tutorial
8	user8	I learned a lot	Nature Documentary

- This query joins the Comments table with the Users table based on the UserID column to associate each comment with its commenter. Then it joins the Content table to link the comment with the content it belongs to. Finally, it selects the username of the commenter, the message text of the comment, and the title of the content.

```
SELECT Users.Username, Comments.MessageText, Content.Title
FROM Comments
JOIN Users ON Comments.UserID = Users.UserID
JOIN Content ON Comments.ContentID = Content.ContentID;
```

Results | Messages

	Username	Title	Description
1	user1	SQL Tutorial	Inappropriate content
2	user2	Nature Documentary	Copyright violation
3	user3	Machine Learning 101	Hate speech
4	user4	Italian Cooking Masterclass	Spam
5	user5	Fashion Trends 2024	Violence
6	user6	Ancient Civilizations	Misleading information
7	user7	Photography Masterclass	Harassment
8	user8	World Travel Diaries	Impersonation

- This query joins the Channels table with the ChannelContent table based on the ChannelID column to link each channel with the content they have created. Then it joins the Content table to retrieve information about the content, including its title and type. Finally, it selects the channel name, the title of the content, and the type of content.

```
SELECT Channels.ChannelName, Content.Title, Content.Type
FROM Channels
JOIN ChannelContent ON Channels.ChannelID = ChannelContent.ChannelID
JOIN Content ON ChannelContent.ContentID = Content.ContentID;
```

Results | Messages

	ChannelName	Title	Type
1	Channel 1	SQL Tutorial	Tutorial
2	Channel 2	Nature Documentary	Documentary
3	Channel 3	Machine Learning 101	Tutorial
4	Channel 4	Italian Cooking Masterclass	Cooking
5	Channel 5	Fashion Trends 2024	Fashion
6	Channel 6	Ancient Civilizations	Documentary
7	Channel 7	Photography Masterclass	Tutorial
8	Channel 8	World Travel Diaries	Travel

- This query links the Users table with the UserAddedContent table to associate each user with the videos they've added to their playlists. Then it joins the Videos table to retrieve information about the added videos. It also joins the ContentPlaylists table to link the added videos with their playlists, and finally, it joins the Playlists table to fetch the names of the playlists. The query selects the username of the user, the title of the video, and the name of the playlist.

```
SELECT Users.Username, Videos.Subtitles, Playlists.Name
FROM Users
JOIN UserAddedContent ON Users.UserID = UserAddedContent.UserID
JOIN Videos ON UserAddedContent.ContentID = Videos.VideoID
JOIN ContentPlaylists ON UserAddedContent.ContentID = ContentPlaylists.ContentID
JOIN Playlists ON ContentPlaylists.PlaylistID = Playlists.PlaylistID;
```

Results | Messages

	Username	Subtitles	Name
1	user1	English	Blues Classics
2	user1	English	Folk Tunes
3	user2	English	Party Hits
4	user2	English	Road Trip
5	user2	English	90s Classics
6	user2	English	Indie Gems
7	user2	English	Soulful Sounds
8	user2	English	Alternative Vibes

Egypt Russian University

Faculty of Management, Economic and Business Technology

✓ Sub Query

--This query combines data from multiple tables using nested SELECT and JOIN statements to provide a comprehensive view of users and their associated channels, playlists, and videos has age more than 18.

```
SELECT u.UserID, u.Username, u.Email, u.Age, ch.ChannelID, ch.ChannelName,ch.CreateDate AS ChannelCreationDate,
pl.PlaylistID,pl.Name AS PlaylistName,  pl.UpdateDate AS PlaylistUpdateDate, v.VideoID,  v.ContentID,   c.Title AS
VideoTitle,  c.Description AS VideoDescription,  c.CreationDate AS VideoCreationDate
FROM Users u JOIN Channels ch ON u.UserID = ch.UserID
JOIN Playlists pl ON u.UserID = pl.UserID
JOIN ContentPlaylists cp ON pl.PlaylistID = cp.PlaylistID
JOIN Content c ON cp.ContentID = c.ContentID
JOIN  Videos v ON c.ContentID = v.ContentID
WHERE  u.UserID IN (
      SELECT UserID
      FROM  Users
      WHERE
        Age > 18
  );
```

Results | Messages

	UserID	Username	Email	Age	ChannelID	ChannelName	ChannelCreationDate	PlaylistID	PlaylistName	PlaylistUpdateDate	VideoID	ContentID	VideoTitle	VideoDescription
1	1	user1	user1@example.com	25	1	Channel 1	2024-01-01	1	Favorites	2024-01-01	1	1	SQL Tutorial	A tutorial on SQL queries
2	1	user1	user1@example.com	25	1	Channel 1	2024-01-01	11	Jazz Standards	2024-01-11	6	6	Ancient Civilizations	Exploring ancient civilizations
3	1	user1	user1@example.com	25	1	Channel 1	2024-01-01	21	Gospel Inspirations	2024-01-21	11	11	Python Basics	Introduction to Python programming
4	2	user2	user2@example.com	30	2	Channel 2	2024-01-02	22	Hip Hop Hits	2024-01-22	11	11	Python Basics	Introduction to Python programming
5	2	user2	user2@example.com	30	2	Channel 2	2024-01-02	12	R&B Grooves	2024-01-12	6	6	Ancient Civilizations	Exploring ancient civilizations
6	2	user2	user2@example.com	30	2	Channel 2	2024-01-02	2	Workout Mix	2024-01-02	1	1	SQL Tutorial	A tutorial on SQL queries
7	3	user3	user3@example.com	28	3	Channel 3	2024-01-03	3	Chill Vibes	2024-01-03	2	2	Nature Documentary	Exploring the wonders of nature

✓ Retrieve all videos uploaded by users who are aged 25 or younger:

```
SELECT *
FROM Videos
WHERE ContentID IN (
  SELECT ContentID
  FROM Content
  WHERE ContentID IN (
    SELECT ContentID
    FROM Users
    WHERE Age <= 25
  )
);
```

Results | Messages

	VideoID	Subtitles	TrendingID	ContentID
1	1	English	1	1
2	2	English	2	2
3	3	English	3	3
4	4	English	4	4
5	5	English	5	5
6	6	English	6	6
7	7	English	7	7
8	8	English	8	8

✓ Get all playlists created by users with the theme "dark":

```
SELECT *
FROM Playlists
WHERE UserID IN (
   SELECT UserID
   FROM Users
   WHERE Theme = 'dark'
);
```

Results Messages

	PlaylistID	Name	UpdateDate	UserID
1	2	Workout Mix	2024-01-02	2
2	4	Study Focus	2024-01-04	4
3	6	Road Trip	2024-01-06	6
4	8	Indie Gems	2024-01-08	8
5	10	Rock Anthems	2024-01-10	10
6	12	R&B Grooves	2024-01-12	2
7	14	Country Road	2024-01-14	4
8	16	Reggae Vibes	2024-01-16	6

✓ Retrieve the usernames of users who have reported videos:

```
SELECT Username
FROM Users
WHERE UserID IN (
   SELECT UserID
   FROM Reports
);
```

Results Messages

	Username
1	user1
2	user2
3	user3
4	user4
5	user5
6	user6
7	user7
8	user8

✓ Retrieve all comments on videos with more than 100 likes:

```
SELECT *
FROM Comments
WHERE ContentID IN (
  SELECT ContentID
  FROM Videos
  WHERE VideoID IN (
    SELECT VideoID
    FROM Content
    WHERE Likes > 100
  )
);
```

Results | Messages

	CommentID	LikesCount	MessageText	UserID	ContentID
1	1	10	Great video!	1	1
2	2	5	Interesting content	2	2
3	3	8	I enjoyed watching this	3	3
4	4	3	Very informative	4	1
5	5	12	Nice work!	5	2
6	6	7	Keep it up!	6	3
7	7	9	This is awesome!	7	1
8	8	6	I learned a lot	8	2

✓ Find all users who have added content to their watch later list:

```
SELECT *
FROM Users
WHERE UserID IN (
  SELECT UserID
  FROM UserWatchLater
);
```

Results | Messages

	UserID	Username	Email	Age	Theme	Password
1	1	user1	user1@example.com	25	light	password1
2	2	user2	user2@example.com	30	dark	password2
3	3	user3	user3@example.com	28	light	password3
4	4	user4	user4@example.com	22	dark	password4
5	5	user5	user5@example.com	35	light	password5
6	6	user6	user6@example.com	27	dark	password6
7	7	user7	user7@example.com	29	light	password7
8	8	user8	user8@example.com	31	dark	password8

✓ View

- This view will display the comments number of likes along with the associated message text.

```
CREATE VIEW PopularComments AS
SELECT MessageText, LikesCount
FROM Comments
```

Results | Messages

	MessageText	LikesCount
1	Great video!	10
2	Interesting content	5
3	I enjoyed watching this	8
4	Very informative	3
5	Nice work!	12
6	Keep it up!	7
7	This is awesome!	9
8	I learned a lot	6

- This view will list users who have interacted with the platform frequently, showing their usernames and the total number of comments they have posted.

```
CREATE VIEW ActiveUsers AS
SELECT u.Username, COUNT(c.UserID) AS TotalComments
FROM Users u
LEFT JOIN Comments c ON u.UserID = c.UserID
GROUP BY u.UserID, u.Username
```

Results | Messages

	Username	TotalComments
1	user1	3
2	user2	1
3	user3	1
4	user4	1
5	user5	1
6	user6	1
7	user7	1
8	user8	1

- This View Will list Top 10 Trending

```
CREATE VIEW TopTenTrending AS
SELECT top 10 *
FROM [dbo].[Trending]
```

Results | Messages

	TrendingID	RegionCode	Ranks
1	1	US	1
2	2	US	2
3	3	US	3
4	4	US	4
5	5	US	5
6	6	US	6
7	7	US	7
8	8	US	8
9	9	US	9
10	10	US	10

- This view will display channels that are currently trending based on the number of subscribers.

```
CREATE VIEW TrendingChannels AS
SELECT c.ChannelName, a.SubscribersCount
FROM Channels c
JOIN Analytics a ON c.AnalyticsID = a.AnalyticsID
```

Results | Messages

	ChannelName	SubscribersCount
1	Channel 1	500
2	Channel 2	600
3	Channel 3	700
4	Channel 4	800
5	Channel 5	900
6	Channel 6	1000
7	Channel 7	1100
8	Channel 8	1200

- This view will provide a breakdown of the types of content available on the platform along with the count of each type.

```
CREATE VIEW ContentTypes AS
SELECT Type, COUNT(ContentID) AS Count
FROM Content
GROUP BY Type;
```

Results | Messages

	Type	Count
1	Art	2
2	Cooking	2
3	Documentary	4
4	Fashion	1
5	Finance	3
6	Fitness	1
7	Health	2
8	History	1

✓ Procedure

- This script will alter the stored procedure sp_AddVideo and comment out the execution of the stored procedure so that it does not run immediately.

```
ALTER PROCEDURE sp_AddVideo
   @VideoID INT,
   @Subtitles VARCHAR(55)
WITH ENCRYPTION
AS INSERT INTO Videos (VideoID, Subtitles)
   VALUES (@VideoID, @Subtitles);
EXEC dbo.sp_AddVideo 54, 'English';
```

31	50	English	NULL	NULL
32	51	English	NULL	NULL
33	54	English	NULL	NULL
34	55	English	NULL	NULL
35	57	English	NULL	NULL

- Insert User Procedure: This procedure will insert a new user into the Users table.

```
CREATE PROCEDURE InsertUser
   @UserID INT,
   @Username CHAR(255),
   @Email VARCHAR(255),
   @Age INT,
   @Theme VARCHAR(6),
   @Password VARCHAR(55)
AS
   INSERT INTO Users (UserID, Username, Email, Age, Theme, Password)
   VALUES (@UserID, @Username, @Email, @Age, @Theme, @Password)

EXEC InsertUser @UserID = 101, @Username = 'example_user', @Email = 'user@example.com', @Age = 25, @Theme =
'light', @Password = 'password123'
```

35	39	example...	example@example.c...	NULL	light	password...
36	40	example...	example@example.c...	NULL	light	password...
37	58	user1	user1@example.com	25	Light	password1
38	90	user2	user2@example.com	30	Dark	password2
39	101	example...	user@example.com	25	light	password...
40	111	example...	user@example.com	25	light	password...
41	370	example...	example@example.c...	NULL	light	password...
42	380	example...	example@example.c...	NULL	light	password...
43	390	example...	example@example.c...	NULL	light	password...
44	400	example...	example@example.c...	NULL	light	password...

- Insert Comment Procedure: This procedure will insert a new comment into the Comments table.

```
CREATE PROCEDURE InsertComment
   @CommentID INT,
   @LikesCount VARCHAR(55),
   @MessageText VARCHAR(55),
   @UserID INT,
   @ContentID IN AS
   INSERT INTO Comments (CommentID, LikesCount, MessageText, UserID, ContentID)
   VALUES (@CommentID, @LikesCount, @MessageText, @UserID, @ContentID)
EXEC InsertComment @CommentID = 32, @LikesCount = '10', @MessageText = 'Great video!', @UserID = 1, @ContentID
= 1
```

30	30	14	This is addictive	30	3
31	31	10	Great video!	1	1
32	32	10	Great video!	1	1

- Insert Advertisement Procedure: This procedure will insert a new advertisement into the Advertisements table.

```
CREATE PROCEDURE InsertAdvertisement
   @AdID INT,
   @Type VARCHAR(55),
   @Duration VARCHAR(55),
   @StartDate VARCHAR(55),
   @EndDate VARCHAR(55),
   @Cost DECIMAL(10,5),
   @Clicks VARCHAR(55)
AS
   INSERT INTO Advertisements (AdID, Type, Duration, StartDate, EndDate, Cost, Clicks)
   VALUES (@AdID, @Type, @Duration, @StartDate, @EndDate, @Cost, @Clicks)

EXEC InsertAdvertisement @AdID = 31, @Type = 'banner', @Duration = '30 seconds', @StartDate = '2024-05-29',
@EndDate = '2024-06-29', @Cost = 100.00, @Clicks = '50'
```

28	28	Text	12 seconds	2024-01-28	2024-02-11	500.00000	100
29	29	Banner	18 seconds	2024-01-29	2024-02-12	600.00000	120
30	30	Video	35 seconds	2024-01-30	2024-02-13	850.00000	170
31	31	banner	30 seconds	2024-05-29	2024-06-29	100.00000	50
32	41	banner	30 seconds	2024-05-29	2024-06-29	100.00000	50

✓ Trigger

- This code creates a table named LogUsers to log user-related events, adds a trigger to insert a record into this table after a new user is inserted into the Users table, inserts user data for verification, and finally selects all records from the LogUsers table.

```
CREATE TABLE  LogUsers (
   log_id INT PRIMARY KEY IDENTITY,
   user_id INT,
   user_name VARCHAR(225),
   user_status VARCHAR(225),
   user_phonenumber VARCHAR(20),
   Status VARCHAR(255),
   log_time DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
   FOREIGN KEY (user_id) REFERENCES users([UserID])
);

CREATE TRIGGER AfterUserLog
ON Users
AFTER INSERT
```

```
AS
BEGIN
   INSERT INTO LogUsers (user_id, user_name, user_status, user_phonenumber, Status, log_time)
   SELECT UserID, Username, 'Active', '', 'Insert', GETDATE()
   FROM inserted;
END;

INSERT INTO users ([UserID], [Username], [Email], [Theme], [Password])
VALUES (370, 'example_user', 'example@example.com', 'light', 'password123');

INSERT INTO users ([UserID], [Username], [Email], [Theme], [Password])
VALUES (380, 'example_user', 'example@example.com', 'light', 'password123');

INSERT INTO users ([UserID], [Username], [Email], [Theme], [Password])
VALUES (390, 'example_user', 'example@example.com', 'light', 'password123');

INSERT INTO users ([UserID], [Username], [Email], [Theme], [Password])
VALUES (400, 'example_user', 'example@example.com', 'light', 'password123');

SELECT  * FROM  LogUsers
```

Results | Messages

	log_id	user_id	user_name	user_status	user_phonenumber	Status	log_time
1	1	370	example_user	Active		Insert	2024-05-29 21:28:16.017
2	2	380	example_user	Active		Insert	2024-05-29 21:28:16.033
3	3	390	example_user	Active		Insert	2024-05-29 21:28:16.043
4	4	400	example_user	Active		Insert	2024-05-29 21:28:16.043
5	5	50800	user1	Active		Insert	2024-05-29 21:28:47.923

✓ This code creates a table named UserAudits to record changes, then sets up a trigger named InsertOrDelete to record insertions and deletions in the Users table. Some data is inserted into the Users table for testing purposes, and then data for a specific user (with ID 9000) is deleted. Finally, the data from the UserAudits table is selected to verify that the changes were recorded correctly.

```
CREATE TABLE UserAudits (
   change_id INT IDENTITY PRIMARY KEY,
   UserID INT NOT NULL,
   Username CHAR(255),
   Email VARCHAR(255),
   Age INT,
   Theme VARCHAR(6),
```

```
    Password VARCHAR(55),
    updated_at DATETIME NOT NULL,
    operation CHAR(3) NOT NULL,
    CHECK(operation = 'INS' OR operation = 'DEL')
);
CREATE OR ALTER TRIGGER InsertOrDelete
ON Users
AFTER INSERT, DELETE
AS
BEGIN
    INSERT INTO UserAudits (
        UserID,
        Username,
        Email,
        Age,
        Theme,
        Password,
        updated_at,
        operation
    )
    SELECT i.UserID, i.Username, i.Email, i.Age, i.Theme, i.Password, GETDATE(), 'INS'
    FROM inserted AS i
    UNION ALL
    SELECT d.UserID, d.Username, d.Email, d.Age, d.Theme, d.Password, GETDATE(), 'DEL'
    FROM deleted AS d;
END;

INSERT INTO Users (UserID, Username, Email, Age, Theme, Password)
VALUES
(5080, 'user1', 'user1@example.com', 25, 'Light', 'password1'),
(9000, 'user2', 'user2@example.com', 30, 'Dark', 'password2'),
(10000, 'user2', 'user2@example.com', 30, 'Dark', 'password2');

INSERT INTO Users (UserID, Username, Email, Age, Theme, Password)
VALUES
(50800, 'user1', 'user1@example.com', 25, 'Light', 'password1')
DELETE FROM Users WHERE UserID = 9000;
```

```
SELECT * FROM UserAudits;
```

Results | Messages

	change_id	UserID	Username	Email	Age	Theme	Password	updated_at	operation
1	1	10000	user2	user2@example.com	30	Dark	password2	2024-05-29 21:23:17.150	INS
2	2	9000	user2	user2@example.com	30	Dark	password2	2024-05-29 21:23:17.150	INS
3	3	5080	user1	user1@example.com	25	Light	password1	2024-05-29 21:23:17.150	INS
4	4	9000	user2	user2@example.com	30	Dark	password2	2024-05-29 21:23:23.923	DEL
5	5	370	example_user	example@example.com	NULL	light	password123	2024-05-29 21:28:16.010	INS
6	6	380	example_user	example@example.com	NULL	light	password123	2024-05-29 21:28:16.033	INS
7	7	390	example_user	example@example.com	NULL	light	password123	2024-05-29 21:28:16.040	INS
8	8	400	example_user	example@example.com	NULL	light	password123	2024-05-29 21:28:16.043	INS
9	9	50800	user1	user1@example.com	25	Light	password1	2024-05-29 21:28:47.920	INS

✓ *code*

```
CREATE DATABASE Youtube;
/*create table users */
CREATE TABLE Users(
   UserID INT PRIMARY KEY,
   Username CHAR(255),
   Email VARCHAR(255),
   Age INT,
   Theme VARCHAR(6),
   Password VARCHAR(55)
);
/*create table comments*/

CREATE TABLE Comments (
   CommentID INT PRIMARY KEY,
   LikesCount VARCHAR(55),
   MessageText VARCHAR(55),
   UserID INT,
   ContentID INT,
   FOREIGN KEY (UserID) REFERENCES Users(UserID),
   FOREIGN KEY (ContentID) REFERENCES Content(ContentID)
);
/*create table watchlater */
CREATE TABLE WatchLater (
   WatchLaterID INT PRIMARY KEY,
   AddedLater VARCHAR(55)
);
/*create table downloads */
CREATE TABLE Downloads (
   DownloadID INT PRIMARY KEY,
   Size VARCHAR(55),
   Quality VARCHAR(55)
);
```

```
/*create table trending */
CREATE TABLE Trending (
   TrendingID INT PRIMARY KEY,
   RegionCode VARCHAR(55),
   Ranks VARCHAR(55)
);
/*create table playlists*/
CREATE TABLE Playlists (
   PlaylistID INT PRIMARY KEY,
   Name VARCHAR(55),
   UpdateDate VARCHAR(55),
   UserID INT,
   FOREIGN KEY (UserID) REFERENCES Users(UserID)
);
/*create table advertisements*/
CREATE TABLE Advertisements (
   AdID INT PRIMARY KEY,
   Type VARCHAR(55),
   Duration VARCHAR(55),
   StartDate VARCHAR(55),
   EndDate VARCHAR(55),
   Cost DECIMAL(10,5),
   Clicks VARCHAR(55)
);
/*create table analytics  */
CREATE TABLE Analytics (
   AnalyticsID INT PRIMARY KEY,
   ViewersCount VARCHAR(55),
   VideosCount VARCHAR(55),
   SubscribersCount VARCHAR(55),
   ShortsCount VARCHAR(55)
);
/*create table channels*/
CREATE TABLE Channels (
   ChannelID INT PRIMARY KEY,
   ChannelName VARCHAR(55),
   CreateDate VARCHAR(55),
   ProfilePic VARCHAR(55),
   Description VARCHAR(55),
   Verified VARCHAR(55),
   UserID INT,
   AnalyticsID INT,
   FOREIGN KEY (UserID) REFERENCES Users(UserID),
   FOREIGN KEY (AnalyticsID) REFERENCES Analytics(AnalyticsID)
);
/*create table content*/
CREATE TABLE Content (
   ContentID INT PRIMARY KEY,
   Description VARCHAR(55),
   Title VARCHAR(55),
   Type VARCHAR(55),
   CreationDate VARCHAR(55),
   Caption VARCHAR(55),
```

```
    Likes VARCHAR(55)
);
/*create table videos*/
CREATE TABLE Videos (
    VideoID INT PRIMARY KEY,
    Subtitles VARCHAR(55),
    TrendingID INT,
    ContentID INT,
    FOREIGN KEY (TrendingID) REFERENCES Trending(TrendingID),
    FOREIGN KEY (ContentID) REFERENCES Content(ContentID)
);
/*create table reports*/
CREATE TABLE Reports (
    ReportID INT PRIMARY KEY,
    Date VARCHAR(55),
    Description VARCHAR(55),
    UserID INT,
    VideoID INT,
    FOREIGN KEY (UserID) REFERENCES Users(UserID),
    FOREIGN KEY (VideoID) REFERENCES Videos(VideoID)
);
/*create table shorts*/
CREATE TABLE Shorts (
    ShortID INT PRIMARY KEY,
    Times VARCHAR(55)
);
/*create table live */
CREATE TABLE Live (
    LiveID INT PRIMARY KEY,
    ViewersCount VARCHAR(55),
    ContentID INT,
    FOREIGN KEY (ContentID) REFERENCES Content(ContentID)
);
/*create table livechat*/
CREATE TABLE LiveChat (
    MessageID INT,
    LiveID INT,
    MessageContent VARCHAR(55),
    PRIMARY KEY (MessageID, LiveID),
    FOREIGN KEY (LiveID) REFERENCES Live(LiveID)
);
/*create table userdownload*/
CREATE TABLE UserDownloads (
    UserID INT,
    DownloadID INT,
    PRIMARY KEY (UserID, DownloadID),
    FOREIGN KEY (UserID) REFERENCES Users(UserID),
    FOREIGN KEY (DownloadID) REFERENCES Downloads(DownloadID)
);
/*create table userwatchlater*/
CREATE TABLE UserWatchLater (
    UserID INT,
    WatchLaterID INT,
```

```
    PRIMARY KEY (UserID, WatchLaterID),
    FOREIGN KEY (UserID) REFERENCES Users(UserID),
    FOREIGN KEY (WatchLaterID) REFERENCES WatchLater(WatchLaterID)
);
/*create table usertrending*/
CREATE TABLE UserTrending (
    UserID INT,
    TrendingID INT,
    PRIMARY KEY (UserID, TrendingID),
    FOREIGN KEY (UserID) REFERENCES Users(UserID),
    FOREIGN KEY (TrendingID) REFERENCES Trending(TrendingID)
);
/*create table userviewedcontent*/
CREATE TABLE UserViewedContent (
    UserID INT,
    ContentID INT,
    PRIMARY KEY (UserID, ContentID),
    FOREIGN KEY (UserID) REFERENCES Users(UserID),
    FOREIGN KEY (ContentID) REFERENCES Content(ContentID)
);
/*create table  useraddedcontent*/
CREATE TABLE UserAddedContent (
    UserID INT,
    ContentID INT,
    PRIMARY KEY (UserID, ContentID),
    FOREIGN KEY (UserID) REFERENCES Users(UserID),
    FOREIGN KEY (ContentID) REFERENCES Content(ContentID)
);
/*create table contentplaylists */
CREATE TABLE ContentPlaylists (
    ContentID INT,
    PlaylistID INT,
    PRIMARY KEY (ContentID, PlaylistID),
    FOREIGN KEY (ContentID) REFERENCES Content(ContentID),
    FOREIGN KEY (PlaylistID) REFERENCES Playlists(PlaylistID)
);
/*create table videowatchlater */
CREATE TABLE VideoWatchLater (
    VideoID INT,
    WatchLaterID INT,
    PRIMARY KEY (VideoID, WatchLaterID),
    FOREIGN KEY (VideoID) REFERENCES Videos(VideoID),
    FOREIGN KEY (WatchLaterID) REFERENCES WatchLater(WatchLaterID)
);
/*create table videodownloads*/
CREATE TABLE VideoDownloads (
    VideoID INT,
    DownloadID INT,
    PRIMARY KEY (VideoID, DownloadID),
    FOREIGN KEY (VideoID) REFERENCES Videos(VideoID),
    FOREIGN KEY (DownloadID) REFERENCES Downloads(DownloadID)
);
```

```
/*create tablecontent advertisments  */
CREATE TABLE ContentAdvertisements (
   ContentID INT,
   AdID INT,
   PRIMARY KEY (ContentID, AdID),
   FOREIGN KEY (ContentID) REFERENCES Content(ContentID),
   FOREIGN KEY (AdID) REFERENCES Advertisements(AdID)
);
/*create table channelcontent  */
CREATE TABLE ChannelContent (
   ContentID INT,
   ChannelID INT,
   PRIMARY KEY (ContentID, ChannelID),
   FOREIGN KEY (ContentID) REFERENCES Content(ContentID),
   FOREIGN KEY (ChannelID) REFERENCES Channels(ChannelID)
);
/*create table user advertisments */
CREATE TABLE UserAdvertisements (
   UserID INT,
   AdID INT,
   PRIMARY KEY (UserID, AdID),
   FOREIGN KEY (UserID) REFERENCES Users(UserID),
   FOREIGN KEY (AdID) REFERENCES Advertisements(AdID)
);
/*Alter table shorts */
ALTER TABLE Shorts
ADD ContentID INT;

ALTER TABLE Shorts
ADD CONSTRAINT fk_content_id FOREIGN KEY (ContentID) REFERENCES Content(ContentID);

/*alter table channels */
ALTER TABLE [dbo].[Channels]
ADD SubscribersCount varchar(255);
```