

**Team # 7:**

Bo Liu

Shruti Padmanabhan

Afreen Patel

Rushikesh Pawar

Umang Saxena

1. **Miniworld & Purpose of the application**

SpantanFlix is a video streaming platform that allows viewers to browse, comment, favorite, and watch videos based on their subscription plans. Besides viewers, we have site admin, who manages content being uploaded to the system and approves advertiser’s (another user of our system) content based on ad subscription plans.

1. **Architecture & Flow of data**

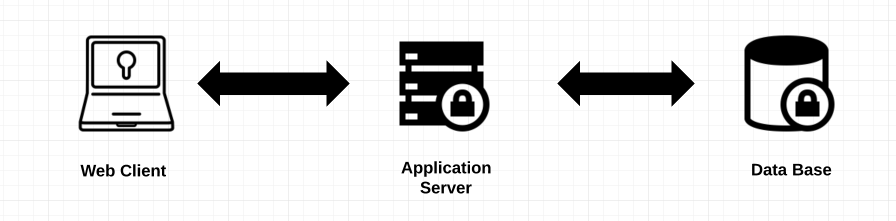


Figure above describes the data flow within our system.Primarily, our application has three components:

1. **Web Client**

This constitutes the application front-end or the UI component or the client facing component of our project. It renders all widgets, forms, content and the visual components of the project. For all user requests it communicated to the application server using AJAX requests for fetching and updating data. The web client is compatible with all modern browsers like Chrome, Firefox, Safari, Microsoft Internet Explorer 11, Microsoft Edge, etc.  
**Technologies used :** HTML5, CSS3, Bootstrap, Javascript, JQuery, etc.

1. **Application Server**

This constitutes the application back-end of our project. It receives and processes all requests from the web client and returns responses after processing them. It partly holds the business logic of the application and has the responsibility of interacting with database in order to process data stored therein. It hosts REST APIs for web clients to reach. It connects to the database using an adapter. It also supports the capability of handling multiple user requests at the same time. It supports user authentication.  
**Technologies used :** Node JS

1. **Database**

This is the component responsible for storing and managing data. It consists of all the data stored in the form of tables and has many stored procedures holding the business logic. It implements encryption in order to protect sensitive data and supports transactions.  
**Technologies used :** MySQL

1. **Functionalities**
2. If different from planned list, status of planned one?
3. Any missing functionalities?
4. Describe each functionality
5. **Major design decisions**
6. trade-offs
7. committed vs changed - explain why (maybe feasibility)
8. **Task list**

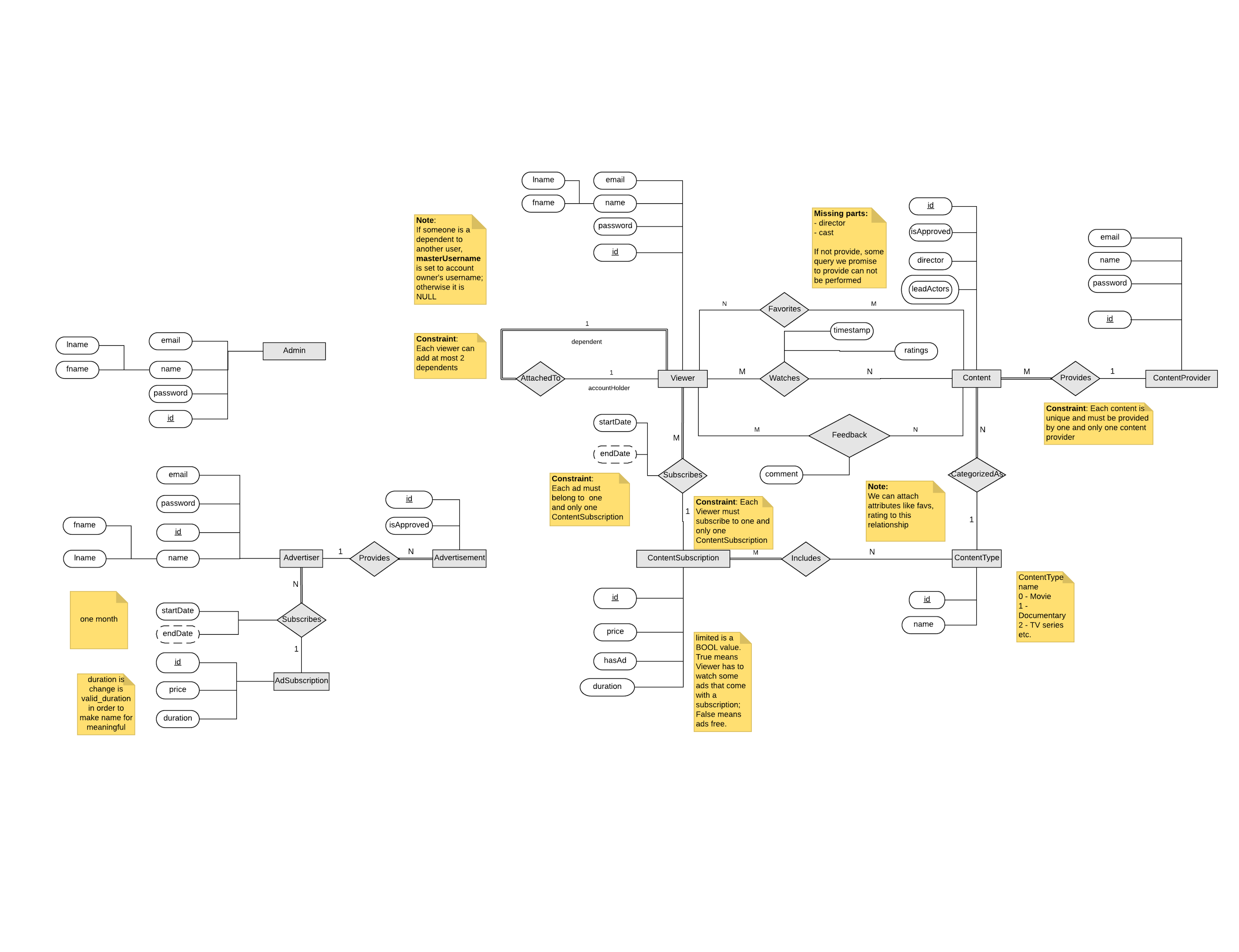
**A** = Afreen, **B** = Bo, **R** = Rushikesh, **S** = Shruti, **U** = Umang

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Person Responsible | Due |
| WEEK #1 | P2: Finalize ERD + Functionalities + comparing with proposal | A, B, S | 11/02 |
| P2: Github private link and share + project title | A, B, S | 11/02 |
| P2: ER to Relational Mapping | A, B | 11/05 |
| P2: Application (what's for front-end, server-side) - discussing how data would flow and users would interact with our system | A, B, R, S, U | 11/05 |
| P2: List out functionalities (for report + prepping for query building), by whom, by when | A, B, R, S, U | 11/05 |
|  |  |  |  |
| WEEK #2 | P2: Normalization + Functional Dependencies P2: Double checked integration of ERD & Relational Mapping | S | 11/09 |
| P2: Create schema | B | 11/09 |
| P2: Write 30% of the report (changes to the submited ER vs. new, architecture design, components, sequence, flow of data) - refer to proposal | S, U | 11/12 |
| P2: Fill sample data (minimum of 10 and max of 15 rows needed for each table) | R, S | 11/12 |
| P2: Script DB queries for ALL THE FUNCTIONALITIES mentioned last week (tables, views, queries, stored procedures) | R, U | 11/12 |
|  |  |  |  |
|  | P2: Frontend ready with placeholders | A, S | 11/16 |
| WEEK #3 | P2: Fully integrated Frontend-Backend BASE CASE (meaning front-end is talking to server and connected to DB and pulling data & showing results as expected) |  | 11/20 |
| P2: COMPLETE report (making sure we hit ALL of his requirements in the doc) + Discuss trade offs and why + Project postmortem (issues uncovered, implementat something differently, potential improvements) |  | 11/20 |
|  |  |  |  |
| WEEK #4 | Develop a Test plan + record results and tweak final output |  | Thanksgiving break |
| Put together a ppt |  | Thanksgiving break |

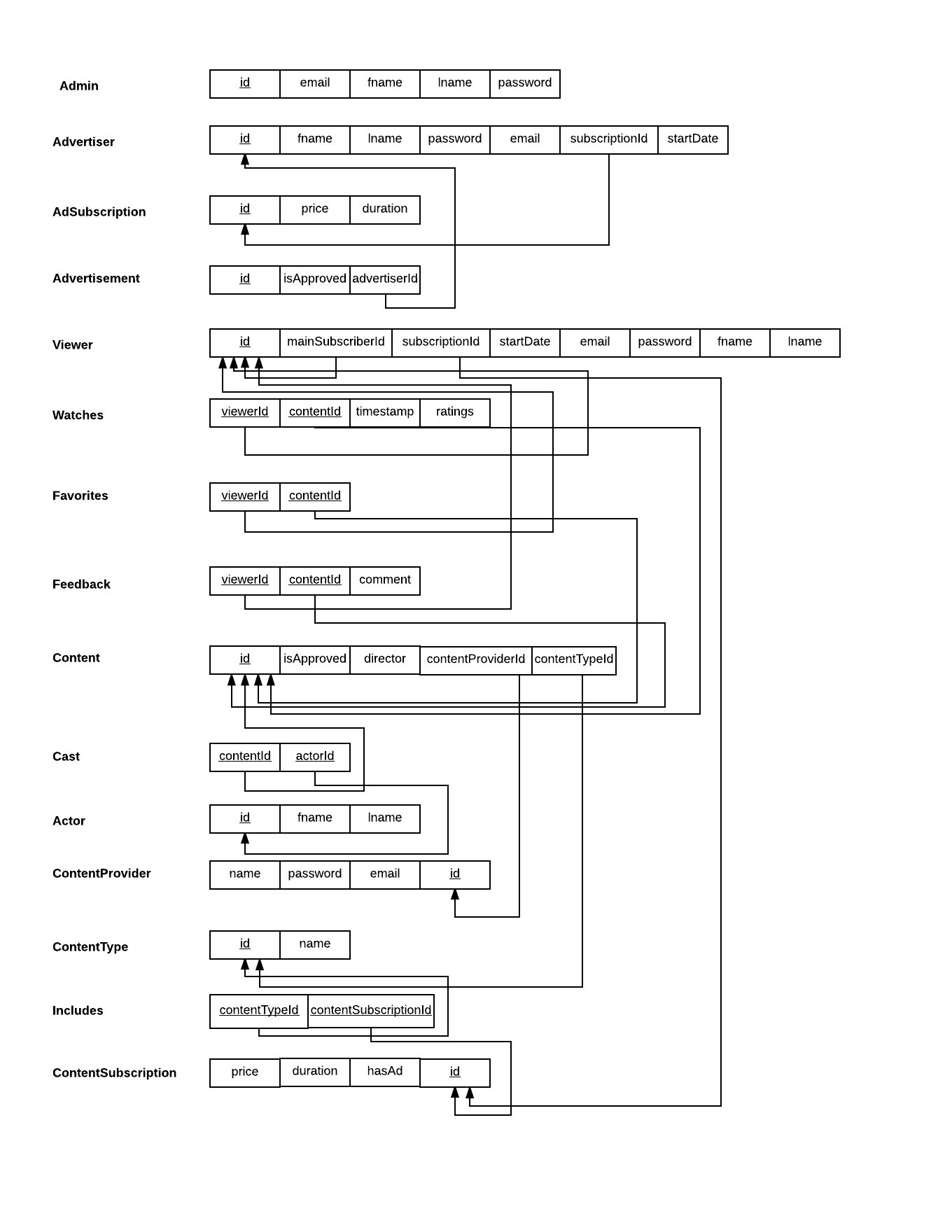
1. **Final Design**
2. ER Diagram (attached separately as well)

There have been some changes in our implementation logic of the system leading to a newer version of ERD. The changes are listed as follows -

* No duration attribute in subscribes.
* Content Subscription includes ContentType
* Provides Feedback
* AttachedTo
* Approve\_Content removed
* No Sees relation
* Content has cast and director attributes
* Content categorized as ContentType
* BelongsTo?
* viewerID in contentProvider



1. Relational schema (attached separately as well)



1. Normalization & Functional dependencies

After normalizing the data, we have 15 relations. Each of the relationships explictly define Primary key (PK), Functional Dependencies (FDs) and reason why it’s 1NF, 2NF, and 3NF.

1. Admin = (id, email, fname, lname, password)
   1. PK: id
   2. FDs: id = (email, fname, lname, password
   3. Since all non-PKs are dependent on 1 PK and have no transitive dependencies, table is perfectly normalized (1NF, 2NF, 3NF)
2. Advertiser = (id, fname, lname, password, email, subscriptionId, startDate)
   1. PK: id
   2. FDs: id -> fname, lname, password, email, subscriptionId, startDate
   3. Since all non-PKs are dependent on 1 PK and have no transitive dependencies, table is perfectly normalized (1NF, 2NF, 3NF)
3. AdSubscription = (id, price, duration)
   1. PK: id
   2. FDs: id -> price, duration
   3. Since all non-PKs are dependent on 1 PK and have no transitive dependencies, table is perfectly normalized (1NF, 2NF, 3NF)
4. Advertisement = (id, isApproved, advertiserId)
   1. PK: id
   2. FDs: id -> isApproved, advertiserId
   3. Since all non-PKs are dependent on 1 PK and have no transitive dependencies, table is perfectly normalized (1NF, 2NF, 3NF)
5. Viewer = (id, mainSubscribedId, subscriptionId, startDate, email, password, fname, lname)
   1. PK: id
   2. FDs: id -> mainSubscribedId, subscriptionId, startDate, email, password, fname, lname
   3. Since all non-PKs are dependent on 1 PK and have no transitive dependencies, table is perfectly normalized (1NF, 2NF, 3NF)
6. Watches = (viewerId, contentId, timestamp, ratings)
   1. PK: viewerId, contentId
   2. FDs:
   3. <Reason>
7. Favorites = (viewerId, contentId)
   1. PK:
   2. FDs:
   3. <Reason>
8. Feedback = (viewerId, contentId, comment)
   1. PK:
   2. FDs:
   3. <Reason>
9. Content = (id, isApproved, director, contentProviderId, contentTypeId)
   1. PK: id
   2. FDs: id -> isApproved, director, contentProviderId, contentTypeId
   3. Since all non-PKs are dependent on 1 PK and have no transitive dependencies, table is perfectly normalized (1NF, 2NF, 3NF)
10. Cast = (contentId, actorId)
    1. PK:
    2. FDs:
    3. <Reason>
11. Actor = (id, fname, lname)
    1. PK: id
    2. FDs: id -> fname, lname
    3. Since all non-PKs are dependent on 1 PK and have no transitive dependencies, table is perfectly normalized (1NF, 2NF, 3NF)
12. ContentProvider = (id, name, password, email)
    1. PK: id
    2. FDs: id -> name, password, email
    3. Since all non-PKs are dependent on 1 PK and have no transitive dependencies, table is perfectly normalized (1NF, 2NF, 3NF)
13. ContentType = (id, name)
    1. PK: id
    2. FDs: id -> name
    3. Since all non-PKs are dependent on 1 PK and have no transitive dependencies, table is perfectly normalized (1NF, 2NF, 3NF)
14. Includes = (contentTypeId, contentSubscriptionId)
    1. PK:
    2. FDs:
    3. <Reason>
15. ContentSubscription = (id, price, duration, hasAd)
    1. PK: id
    2. FDs: id -> price, duration, hasAd
    3. Since all non-PKs are dependent on 1 PK and have no transitive dependencies, table is perfectly normalized (1NF, 2NF, 3NF)
16. Table or view design
17. Specification of each DB object
    1. tables, column, view, stored procedure, trigger etc (meaning/ purpose)
18. Any specific functionality involving modification of more than one table??? (show code snippet of that transaction)
19. ANY Explicit DB transaction initiated from DB server side??? (ie stored procedure) - show code snippet
20. ANY additional DB object/ concepts used? (trigger, index, isolation, concurrency control...etc)
21. **Sample Execution & Test plan (screenshots)**
22. **Issues uncovered**
23. **Unique designs (optional)**
24. **Implemented something differently?**
25. **Additional documents attached**
26. Script to create DB objects (only give filename)
27. Script to load data (only give filename)
28. ERD.png
29. RelationalMapping.png
30. Source code (only give zip filename)
31. Database transaction log (only give filename)
32. **Potential Improvements**