**Schema-Design-Questions**

**Question 1**

**Movies Schema Design Project**

This project is a sample implementation of a database schema for a movie database. The database contains information about movies such as the title, year of release, director, and actors.

**Tables and Attributes**

The schema consists of the following tables and attributes:

**Movies**

* **movie\_id** (integer): Unique identifier for each movie
* **title** (string): Title of the movie
* **year** (integer): Year of the movie release
* **director\_id** (integer): Foreign key to the Directors table

**Directors**

* **director\_id** (integer): Unique identifier for each director
* **name** (string): Name of the director

**Actors**

* **actor\_id** (integer): Unique identifier for each actor
* **name** (string): Name of the actor

**Movie\_Actors**

* **movie\_id** (integer): Foreign key to the Movies table
* **actor\_id** (integer): Foreign key to the Actors table

**Relationships**

The relationships between the tables are as follows:

* A movie is directed by one and only one director, and a director can direct multiple movies. Therefore, the Movies table has a foreign key to the Directors table.
* An actor can star in multiple movies, and a movie can have multiple actors. Therefore, we have a many-to-many relationship between the Movies and Actors tables, which is implemented using a junction table called Movie\_Actors.

**Database Design**

The following diagram illustrates the database schema design:

**Queries**

The schema is normalized enough to avoid duplicating strings too much, and it can efficiently answer questions like the following:

* **Who acted in Fight Club (1999)?**

To answer this question, we can join the Movies table with the Movie\_Actors and Actors tables, and filter the results by the movie title and year.

* **What are the 10 most recent movies that George Clooney starred in?**

To answer this question, we can join the Movies table with the Movie\_Actors and Actors tables, and filter the results by the actor name. We can then sort the results by the movie year in descending order and limit the results to 10.

**Conclusion**

This project provides a sample implementation of a database schema for a movie database. The schema is designed to efficiently store and query information about movies, directors, and actors. It can be extended and customized to suit specific needs and requirements.

**Question 2**

**Scaler Academy Schema Design Project**

This project is a sample implementation of a database schema for an online learning platform called Scaler Academy. The database contains information about students, batches, lectures, assignments, homework, instructors, and more.

**System Requirements**

The schema satisfies the following system requirements:

1. Students can register for different batches (e.g., Feb'20).
2. Students attend daily lectures on various topics (like Trees, Linked Lists, Stacks, etc.).
3. Every lecture has an assignment and homework.
4. Every assignment and homework can have 0-10 problems.
5. Every class has an instructor, some pre-lecture content, and lecture notes.
6. Students can see the class timeline, problem-solving percentage, attendance percentage, and status of all the problems in all the assignments/homework in the dashboard.
7. Instructors can see a list of classes taken by them and their upcoming classes.
8. Instructors can create lessons for all the batches and add/modify the assignments and homework problems.

**Future Scope**

The schema can be extended to support the following features:

1. Store ratings and feedback for the class.
2. Collaborative lectures taken by multiple instructors.
3. Daily streak calculation.

**Tables and Attributes**

The schema consists of the following tables and attributes:

**Students**

* **student\_id** (integer): Primary key for each student
* **name** (string): Name of the student
* **email** (string): Email address of the student

**Batches**

* **batch\_id** (integer): Primary key for each batch
* **name** (string): Name of the batch (e.g., Feb'20)

**Lectures**

* **lecture\_id** (integer): Primary key for each lecture
* **batch\_id** (integer): Foreign key to the Batches table
* **topic** (string): Topic of the lecture (e.g., Trees, Linked Lists, Stacks)
* **instructor\_id** (integer): Foreign key to the Instructors table
* **prelecture\_content** (text): Pre-lecture content for the lecture
* **lecture\_notes** (text): Notes for the lecture

**Assignments**

* **assignment\_id** (integer): Primary key for each assignment
* **lecture\_id** (integer): Foreign key to the Lectures table
* **name** (string): Name of the assignment
* **num\_problems** (integer): Number of problems in the assignment

**Homework**

* **homework\_id** (integer): Primary key for each homework
* **lecture\_id** (integer): Foreign key to the Lectures table
* **name** (string): Name of the homework
* **num\_problems** (integer): Number of problems in the homework

**Problems**

* **problem\_id** (integer): Primary key for each problem
* **assignment\_id** (integer): Foreign key to the Assignments table
* **homework\_id** (integer): Foreign key to the Homework table
* **name** (string): Name of the problem

**Students\_Lectures**

* **student\_id** (integer): Foreign key to the Students table
* **lecture\_id** (integer): Foreign key to the Lectures table
* **attendance** (boolean): Whether the student attended the lecture
* **problem\_solved** (integer): Number of problems solved by the student for the lecture

**Instructors**

* **instructor\_id** (integer): Primary key for each instructor
* **name** (string): Name of the instructor
* **email** (string): Email address of the instructor

**Question 3**

**Netflix Schema Design**

We are going to design a database schema for a system similar to Netflix. The system should have the following use cases:

1. Users can create profiles that have separate independent environments. Each profile has a name and a type. Type can be KID or ADULT.
2. There are multiple videos on Netflix. For each video, there will be a title, description, and a cast.
3. A cast is a list of actors who were part of the video. For each actor, we need to know their name and the list of videos they were part of.
4. For every video, for any profile who watched that video, we need to know the status (COMPLETED/ IN PROGRESS).
5. For every profile for whom a video is in progress, we want to know their last watch timestamp.

**Tables:**

1. User:

* user\_id (PK)
* email
* password

1. Profile:

* profile\_id (PK)
* user\_id (FK)
* name
* type (KID or ADULT)

1. Video:

* video\_id (PK)
* title
* description

1. Actor:

* actor\_id (PK)
* name

1. Video\_Actor:

* video\_id (PK) (FK)
* actor\_id (PK) (FK)

1. Watch:

* profile\_id (PK) (FK)
* video\_id (PK) (FK)
* status (COMPLETED/IN PROGRESS)
* last\_watched\_timestamp

**Relationships:**

1. User to Profile: One user can have multiple profiles, but each profile belongs to only one user.
2. Profile to Watch: One profile can have multiple watches, but each watch belongs to only one profile.
3. Video to Video\_Actor: One video can have multiple actors, and one actor can be a part of multiple videos.
4. Video to Watch: One video can have multiple watches, and one watch belongs to only one video.
5. Profile and Video to Watch: One profile can have multiple watches of different videos, and one video can have multiple watches by different profiles.

**Indexes:**

1. User: email
2. Profile: user\_id
3. Video: title
4. Actor: name
5. Video\_Actor: video\_id, actor\_id
6. Watch: profile\_id, video\_id