1) What is the relationship between the *Actor* and *Movie* entities shown?

A. 1:1  
B. 1:m  
C. m:n
2) How many joins are needed to find all cast members who acted in 'A Star is Born' and return the name and age for each member?

A. 1  
B. 2  
C. 3
3) What if we wanted to model an actor who directed themselves in a movie? How would the schema change in order to represent the various roles that a single person can perform in movie? (e.g. actor, director, producer, screenwriter)

A. Add a *role* field to the *Cast* table
B. Create a *Director* table with the same schema as the *Actor table*
C. A and B
D. None of the above
4) What is the relationship between the Person and Movie entities in this updated schema?

A. 1:1  
B. 1:m  
C. m:n
5) What can we do to ensure that the role field in Cast_Crew contains consistent data? For example, suppose we want every screenwriter record to be stored consistently as 'Screenwriter' (as opposed to 'Writer', 'Screen-Writer', 'Script Person', etc).

A. Create a reference table for every distinct role
B. Add a foreign key on Cast_Crew.role that points to the new Role table
C. A and B
D. None of the above
Terminology

- Entity: An object or a thing
- Usually a noun
- Common Examples: Person, Team, Product, Sales Order

Analogies with OOP:
- Entity: analogous to Object
- Entity Type: analogous to Class

Questions:
- What are the boundaries?
- How to handle hierarchies?
Design Principles

- A table models a single entity type and an entity type is modeled by a single table.
- Each field in a table is assigned a primitive data type.
- Each field in a table is assigned a precise data type.
- Each table contains a single Primary Key (PK).
- Each child table contains a Foreign Key (FK) that points to its parent(s).
- Each $m:n$ relationship is modeled with a junction table.
What can go wrong

- Insert Anomaly
- Update Anomaly
- Delete Anomaly
Normalized College Tables

**Student**
- PK
- sid: string
- fname: string
- lname: string
- dob: date

**Teacher**
- PK
- tid: string
- fname: string
- lname: string
- dept: string

**Teaches**
- PK, FK
- cno: string

**Takes**
- PK, FK
- sid: string
- cno: string
- grade: string

**Class**
- PK
- cno: string
- cname: string
- credits: integer
Normal Forms

1NF: A database schema is in 1NF if all attributes have scalar values.

2NF: 1NF + all non-key attributes must be functionally determined by the entire primary key.

3NF: 2NF + all non-key attributes must be functionally determined by only the primary key.

Functional Dependencies:
If two records agree on the attributes $A_1, A_2, \ldots, A_n$ then they must also agree on the attributes $B_1, B_2, \ldots, B_n$

Formally:
$A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_n$
Normal Form Violations

```
<table>
<thead>
<tr>
<th>Table</th>
<th>PK</th>
<th>FK</th>
<th>Attributes</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Teacher</td>
<td>tid</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>fname</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>lname</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>dept</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>salary</td>
<td></td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>cno</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>cname</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>Student</td>
<td>sid</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>fname</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>lname</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>dob</td>
<td></td>
<td>date</td>
<td></td>
</tr>
<tr>
<td></td>
<td>email</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>cno</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>grade</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>gpa</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td>Takes</td>
<td>sid</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>cno</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>grade</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>gpa</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td>Class</td>
<td>cno</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>cno</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>grade</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>credits</td>
<td></td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>semester</td>
<td>string</td>
<td></td>
</tr>
</tbody>
</table>
```
Modeling Demo
Practice Problem

Construct a SQL query that finds all Takes records which violate referential integrity with its parent table Class.
iClicker Question

Construct a SQL query that finds all Takes records which violate referential integrity with its parent table Class.

What type of join is needed by this query?

A. Inner join  
B. Outer join  
C. Self join
Milestone 4

1) Requirements: assignment sheet

2) Design questions and/or concerns: sign-up sheet