INFO20003 Tutorial 3

Starting ~2.20 pm

Today's tutorial

- ER Modelling Review
- Cinema case study ER model
  - group work
- Case study: Logical Model
  - together

GitHub Link for Tutorials:

- https://github.com/Prashansa-Singh/INFO20003-Sem1-2022

ER Modelling Review

- Entity
- Weak Entity
- Attribute
- Business Rules
- Key constraints and participation constraints

- Entity

Weak Entity

- **Definition**:
  - **If an entity cannot be uniquely identified without referring to another entity, it is called a weak entity**
    - □ You might've heard "a weak entity can't exist without its owner entity"
    - □ But this isn't always the case because making an entity weak or not is a design choice so it's better to use the above definition

- Can only be identified uniquely by considering PK of owner entity combined with its partial key

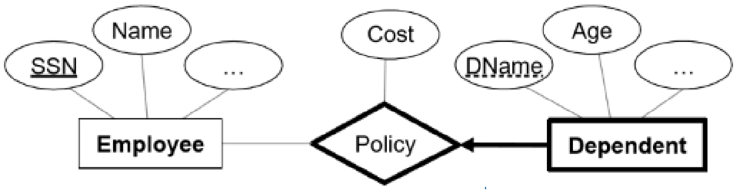
- "identifying" relationship
  - ▪ bold entity rectangle
  - ▪ bold diamond

  - ▪ a weak entity must have total participation in this relationship
    - □ why?

    - □ Dependent cannot exist in the system without the employee

  - Weak entities

- always have a bold arrow coming out of them
- must have a partial key (dashed underline)

- 

- How to identify:
  - Read the text to see if an entity cannot be uniquely identified with its partial key, but also needs another entity's PK
  - Look at your conceptual model: entities with bold arrow coming out might be weak entities

  - But it's also a modelling choice! You could make it a weak entity OR you could make it a normal entity and give it a surrogate key
    - Sometimes there's no one right way of doing things

- Attributes - help describe the entity
- Business Rules
  - used to define the entities, attributes, their relationships and constraints.

<u>4 types of one side of a relationship (Chen's)</u>

lower bound upper bound

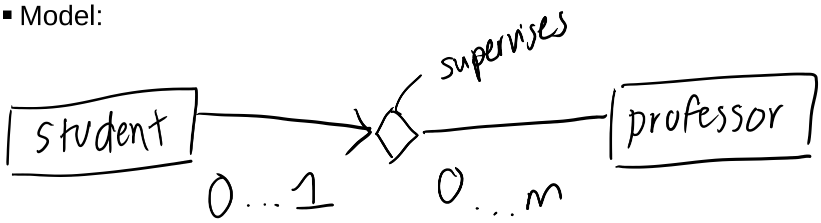
—— 0..m	0 to many	partial participation	optional many
—— 1..m	1 to many	total participation	mandatory many
→ 0..1	0 to 1	partial participation key constraint	optional one
→ 1..1	1 to 1	total participation key constraint	mandatory one "1 and only 1"

- These are just about one side of the relationship not the whole side

Key Constraints

- Constraint on the upper bound
  - □ can be 1 or many
- At most one = arrowhead
- many = no arrow head

- Example:
- **A student is allowed to be supervised by at most one professor, but the professor on the other hand can supervise more than one student**
- Model:



Chen's notation : Read it like english L to R
a student can be supervised by 0 or 1 professors
a professor can supervise 0 to many students

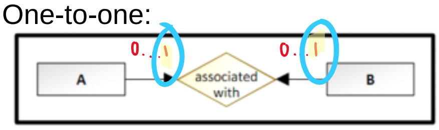
Notice language:

"can" and "allowed" implies optional (partial participation). So choosing a lower bound of 0 is a good idea. If it said "must", this implies total participation.

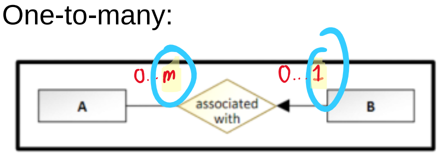
3 types of Relationships (Key Constraints)

Notice: We only care about the upper bounds

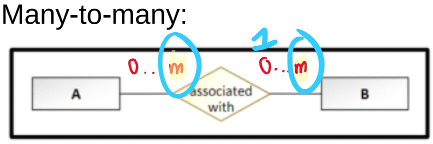
One-to-one:



One-to-many:



Many-to-many:



These are the 3 types of relationships

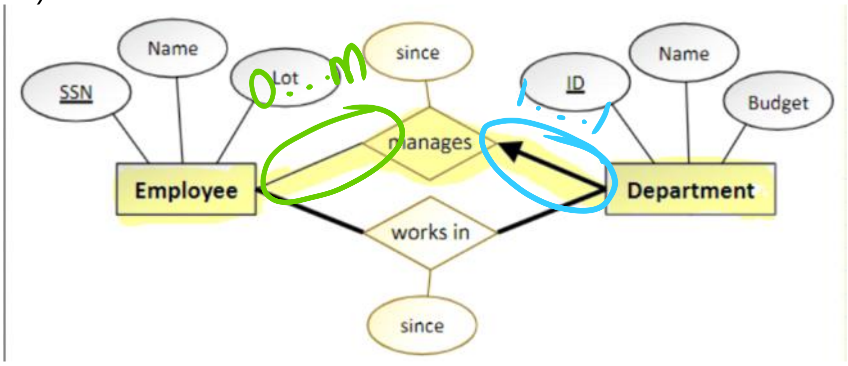
- **BE CAREFUL**: WHOLE relationship vs ONE side of the relationship

Participation Constraints

- participation of an entity in a relationship can be either
  - total (bold line)
  - partial (normal line)

- constraint on the lower bound
  - can be 0 or 1
    - 0 (optional)
    - 1 (total) "at least one"

Ex)

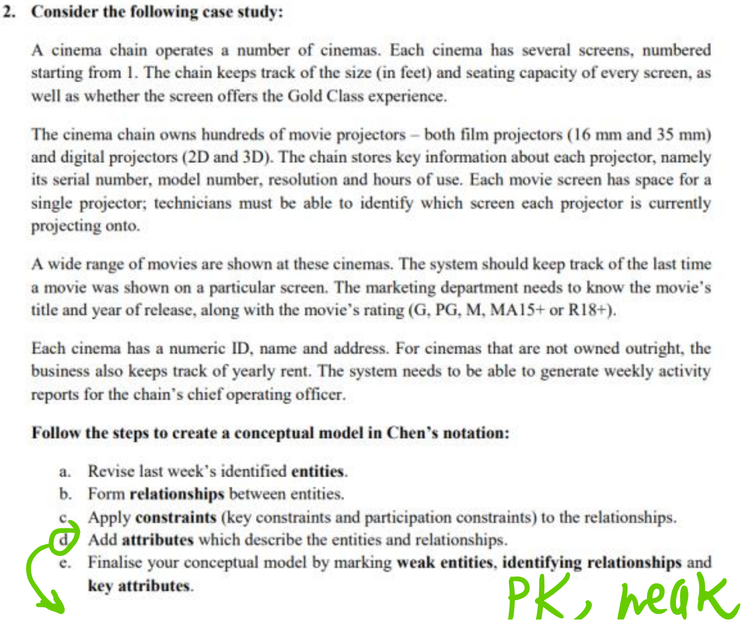


What does this 'manages' relationship mean (Read it out)?

- a department must have a manager (total participation)
  - and can only have at most 1 manager (key constraint)

- not every employee is a manager of a department (partial participation)
  - an employee can manage 0 to many departments

Group work (15-20 mins):



2. **Consider the following case study:**

A cinema chain operates a number of cinemas. Each cinema has several screens, numbered starting from 1. The chain keeps track of the size (in feet) and seating capacity of every screen, as well as whether the screen offers the Gold Class experience.

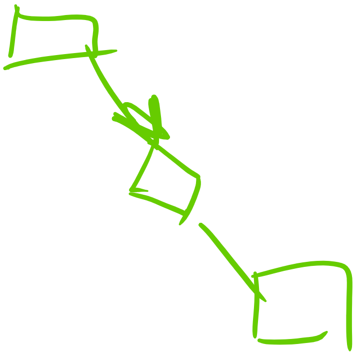
The cinema chain owns hundreds of movie projectors – both film projectors (16 mm and 35 mm) and digital projectors (2D and 3D). The chain stores key information about each projector, namely its serial number, model number, resolution and hours of use. Each movie screen has space for a single projector; technicians must be able to identify which screen each projector is currently projecting onto.

A wide range of movies are shown at these cinemas. The system should keep track of the last time a movie was shown on a particular screen. The marketing department needs to know the movie's title and year of release, along with the movie's rating (G, PG, M, MA15+ or R18+).

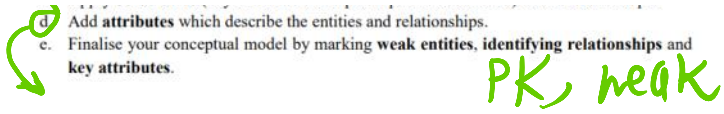
Each cinema has a numeric ID, name and address. For cinemas that are not owned outright, the business also keeps track of yearly rent. The system needs to be able to generate weekly activity reports for the chain's chief operating officer.

Follow the steps to create a conceptual model in Chen's notation:

a. Revise last week's identified **entities**.
b. Form **relationships** between entities.
c. Apply **constraints** (key constraints and participation constraints) to the relationships.
d. Add **attributes** which describe the entities and relationships.
e. Finalise your conceptual model by marking **weak entities**, **identifying relationships** and **key attributes**.



PK, weak



d. Add **attributes** which describe the entities and relationships.

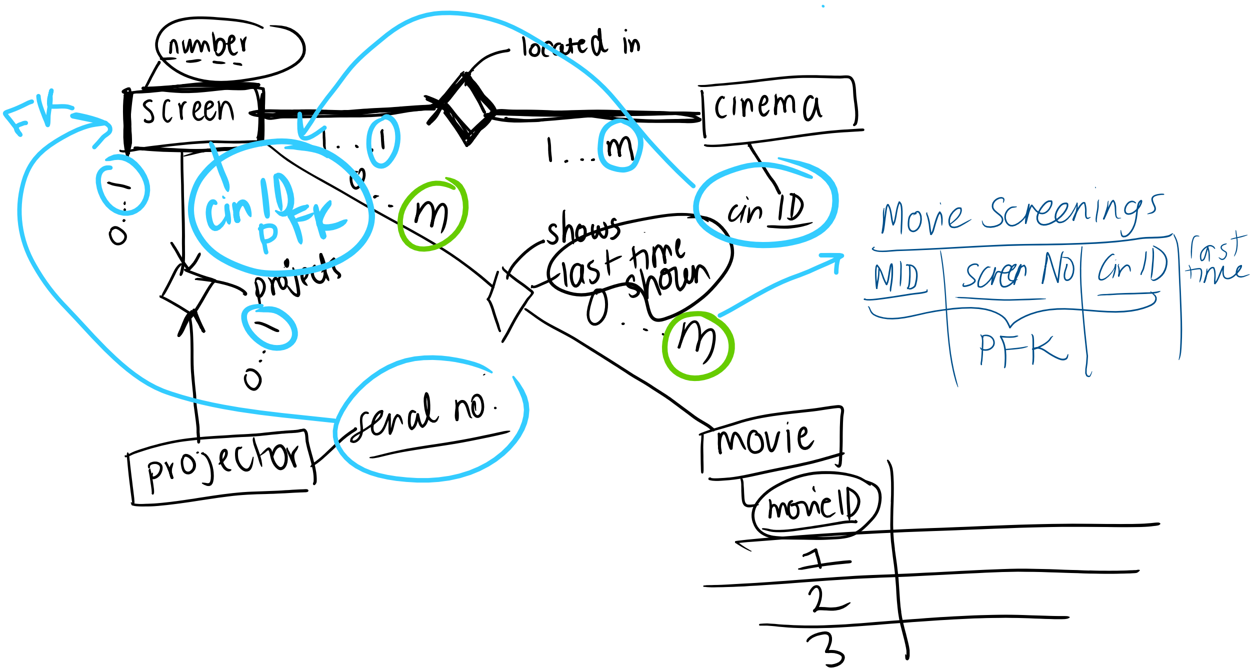
e. Finalise your conceptual model by marking **weak entities**, **identifying relationships** and **key attributes**.

- First do (b) just figure out which entities have a relationship and are connected
- Then do (c) and start thinking about constraints (e.g. bold line/add arrow)
- Do (d) after (e)

Cinema (ID, name, address, yearly rent)
Screen (number, size, seating capacity, has Gold Class?)
Projector (format, serial number, model number, resolution, hours of use)
Movie (title, year of release, rating)

Not too much time in this tutorial unfortunately, so please:

- **focus on doing the relationships and constraints and part e)**
- **don't worry about how it looks too much, putting in ovals etc or writing up all the attributes for this example. But do this for assessments**



Movie Screenings

MID	screen No	cin ID	last time

PFK

- Choose descriptive names for relationships
  - avoid "has" (vague)

- For this example, we modelled address as a simple attribute. But in future models, usually you should model address as a composite attribute or a separate entity

- Notice:
  - for the "projects onto" relationship, we care about which screen each projector is currently projecting onto
    - thus a projector can at most project onto 1 screen
    - and a screen can be projected onto by at most 1 projector
  - But for the "shown on" relationship, we care about which movie was shown on which screen over time (since it asks about the last screening date)
    - thus a movie can be shown on many screens (over time)
    - and a screen may show many movies (over time)

- Constraints

  - sometimes not all are given in text
  - use common sense
  - Pay attention to the language
    - "must" = mandatory (key constraint)
    - "may" = optional (partial participation)

- Does an attribute belong to an entity or a relationship?

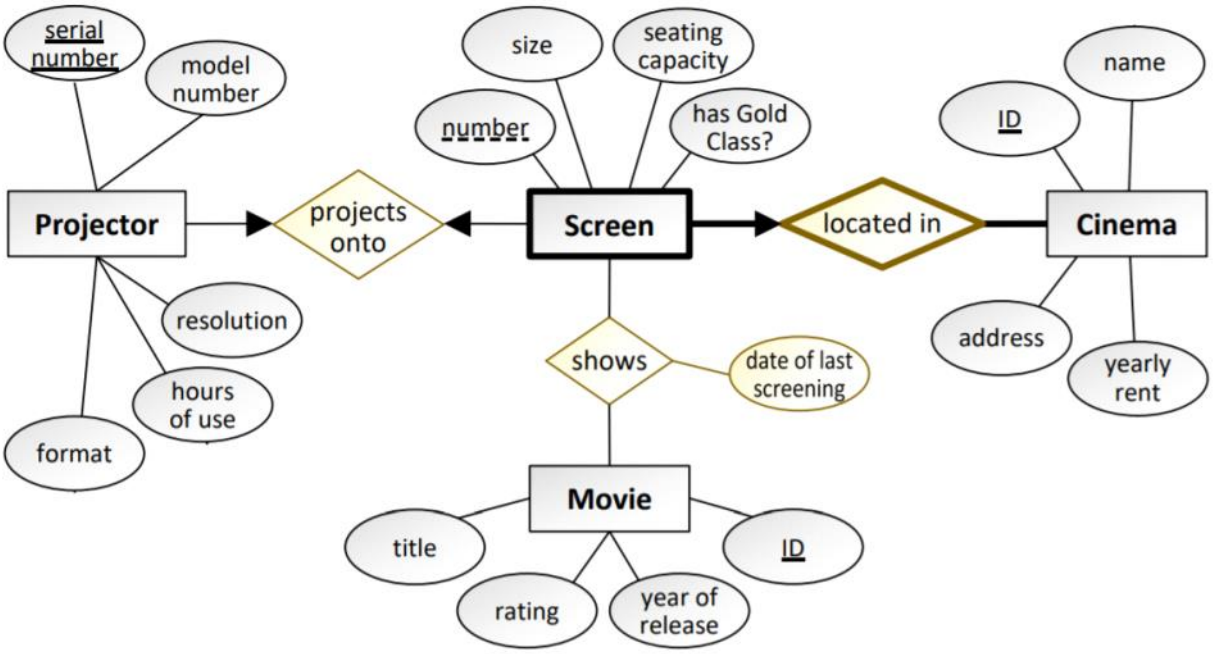
- Each entity must have a PK (or partial key if weak)
- Usually you shouldn't have circular/cyclic models
  - should be able to backtrack to get the info you need

- Are there any weak entities?
- weak entities must have at least 1 identifying relationship
- weak entities can have 'normal' relationships with other entities

Surrogate key:

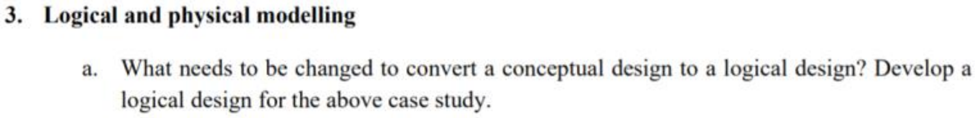
  - Unique, meaningless number used to identify an instance of an entity e.g. Bank account number
    - only invented for use in one DB
    - Movie relation has no unique attribute to use (title, year, rating may not be unique).
      - So 'MovieID' is invented so we can store it in the DB
    - CinemaID can also be considered a surrogate key

<u>Final Conceptual Model:</u>



What type is each relationship?

- One-to-One, One-to-Many, Many-to-Many?


3. Logical and physical modelling

a. What needs to be changed to convert a conceptual design to a logical design? Develop a logical design for the above case study.

<u>RULES - To go from conceptual to logical:</u>

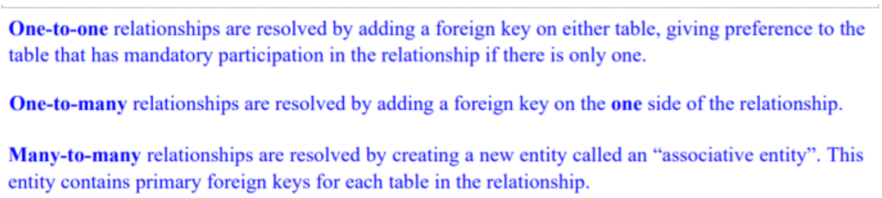
- Choose a model e.g. relational model

- **Flatten multivalued and composite attributes**
- **Remove any derived attributes**

- **Resolve many-to-many relationships**
  - create an associative entity
  - bring the PKs from the LHS and RHS of the relationship to form a composite PK
  - also bring along any relationship descriptive attributes

- **Resolve one-to-many relationships**
  - Add the foreign keys (FKs) at the one side of the relationship (the side of the relationship with a key constraint)

- **Resolve one-to-one relationships**
  - add FK to either side, giving preference to the side that has total participation (if there is one)
  - if symmetric (both sides are total participation or both sides partial participation), can pick arbitrarily where to add FK



One-to-one relationships are resolved by adding a foreign key on either table, giving preference to the table that has mandatory participation in the relationship if there is only one.

One-to-many relationships are resolved by adding a foreign key on the **one** side of the relationship.

Many-to-many relationships are resolved by creating a new entity called an "associative entity". This entity contains primary foreign keys for each table in the relationship.

FK = reference to another relation's PK (primary key)

- What do I do with relationship attributes?
  - they go where the FK is added

- Convention: Change names of attributes to CamelCase

So after doing our conceptual model, we have

Cinema (CinemaID PK, name, address, yearly rent)

Screen (number Partial Key , size, seating capacity, has Gold Class?)

Projector (serial number PK, format, model number, resolution, hours of use)

Movie (MovieID PK, title, year of release, rating)

Now we look at our ER model
- we have no composite or multi-valued attributes
- so let's resolve relationships!

Projector-Screen relationship
- this is a one-to-one relationship
- both of them are partial participation (0 on lower bound)
- so we can add FK to either side
  - What if the one side had partial participation and the other side had total participation?
    - Then we'd move the FK to the side with total participation. Thus, each instance in the relation on that side would always have a value for the FK (NOT NULL).
    - If we moved FK to other side (the side with partial participation), some instances in the relation would have a NULL value in the FK column.
    - So we always prefer moving FK to side with total participation (if there is one) to avoid having lots of NULL values.
- could add FK on the Projector table that references the PK of the Screen table
- or
- could add FK on the Screen table that references the PK of the Projector table

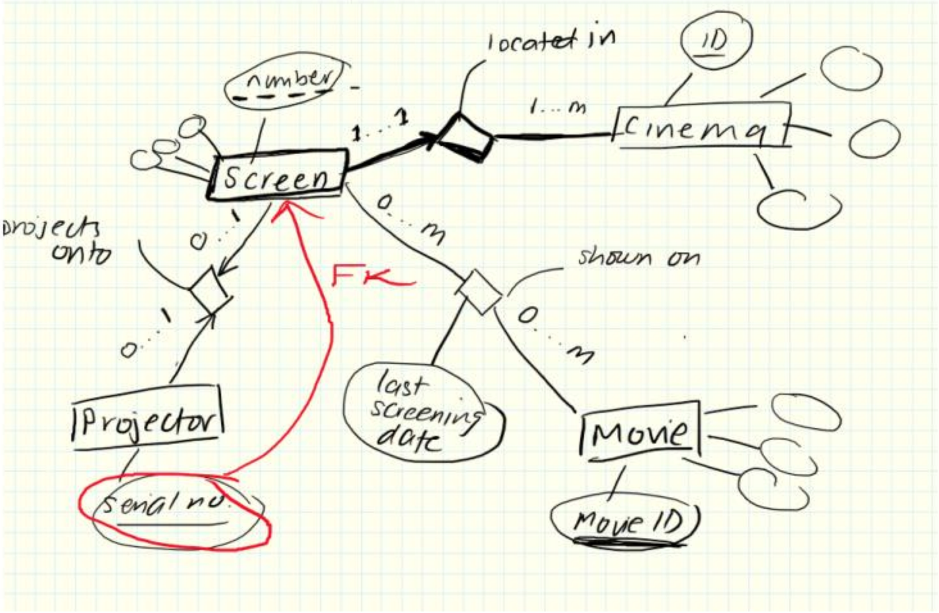
- ○ let's do this:
- ○ so we get

Cinema (CinemaID PK, name, address, yearly rent)

Screen (number Partial Key , size, seating capacity, has Gold Class?, serial number FK)

Projector (serial number PK, format, model number, resolution, hours of use)

Movie (MovieID PK, title, year of release, rating)



Cinema-Screen relationship

- is one-to-many
- Add the foreign keys (FKs) at the one side of the relationship (the side of the relationship with a key constraint)
    - Why?
    - As each instance in the relation with a key constraint is associated with at most 1 instance in the relation on the many side
    - If you tried putting the FK on the many side, it just wouldn't work because one instance on the relation on the many side can be associated with many instances in the relation on the one side. So you'd have to somehow be able to store multiple values in that FK column but that's not a valid relational model.
        - And we can't also just keep on adding more and more columns to store each of those multiple values because we need to have a fixed schema with a known number of columns (but we don't know what the maximum number of possible values is)
    - Example:
    - In the Cinema-Screen relationship
    - One screen is associated with one and only one cinema. So it's easy to just add another column in the Screen relation that stores the CinemaID of the cinema that the screen is located in
    - But one cinema can contain many screens. So we'd have to somehow try to store all the screen numbers that are associated with one CinemaID. But we only have one column for the FK in the Cinema relation to store all the screen numbers. This is not possible in relational models.
- Add FK (**CinemaID)** to Screen
- But because this is an identifying relationship of the weak entity screen, the FK CinemaID also becomes a PK in the Screen table. This is a **PFK**

- We need both the partial key of Screen and the PK of its owner entity to uniquely identify instances in Screen table
- This is a composite PK (2 or more columns form PK)
- So we get :

Cinema (CinemaID PK, name, address, yearly rent)

Screen (number Partial Key , CinemaID PFK, size, seating capacity, has Gold Class?, serial number FK)

Projector (serial number PK, format, model number, resolution, hours of use)

Movie (MovieID PK, title, year of release, rating)

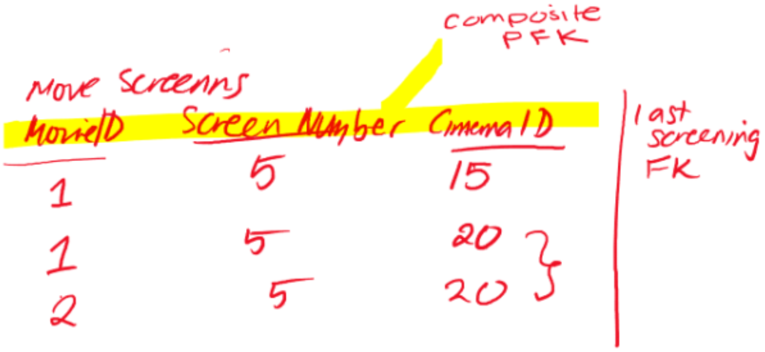
Movie-Screen Relationship

- is many-to-many
- create an associative entity (a new relation!)
- let's call it 'MovieScreening'
- bring the PKs from both Movie and Screen relations, and make them **PFKs** in this associative entity

- What to do with any relationship descriptive attributes like 'last screening date'?
  - relationship attributes always go in the same table that the FKs go to
  - Here, the FKs were placed in the associative entity, so Date of Last Screening gets placed in this table as a non-key column

- so our new relation is:
  - **MovieScreening(CinemaID PFK, ScreenNumber PFK, MovieID PFK, DateOfLastScreening)**
- notice: whenever bringing the PK of Screen, must bring the whole composite PK

- Why do we need to make an associative entity for a many-to-many relationship?
  - Because each movie can be shown on many screens (over time). So you can't just add a column to the Movie relation to store the screens it has been shown on since there are possibly multiple values to store in that field. (And you also can't just keep adding more and more columns to store all the screen numbers because we need to know beforehand how many columns we have)
  - Each screen can also show many movies (over time). Similarly, you can't just add a column to capture the MovieID associated with a screen since there may be multiple MovieID values to store for each screen.
  - So we need to create a new entity with a combined (composite) PK
  - There's no other way to store the **history** of which movies have been shown on which screens.



MovieID	Screen Number	CinemaID
1	5	15
1	5	20
2	5	20

Completed Logical Design:

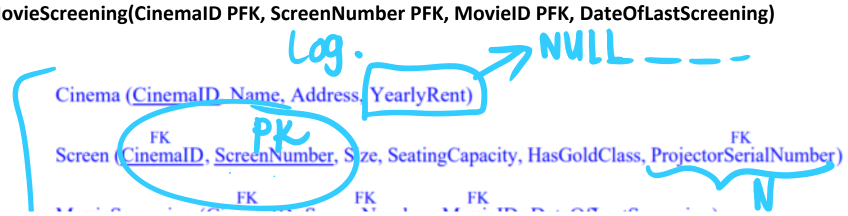
- convert to CamelCase

Cinema (CinemaID PK, Name, Address, YearlyRent)

Screen (Number Partial Key , CinemaID PFK, Size, SeatingCapacity, HasGoldClass, ProjectorSerialNumber FK)

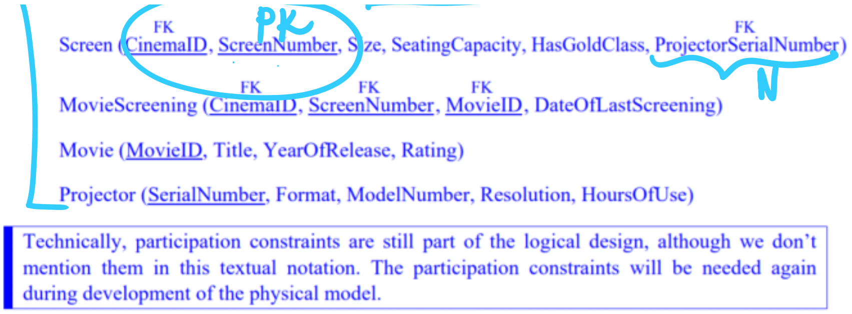
Projector (SerialNumber PK, Format, ModelNumber, Resolution, HoursOfUse)

Movie (MovieID PK, title, year of release, rating)
MovieScreening(CinemaID PFK, ScreenNumber PFK, MovieID PFK, DateOfLastScreening)



Cinema (CinemaID, Name, Address, YearlyRent)

Screen (CinemaID, ScreenNumber, Size, SeatingCapacity, HasGoldClass, ProjectorSerialNumber)



Screen (CinemaID FK, ScreenNumber, Size, SeatingCapacity, HasGoldClass, ProjectorSerialNumber FK)

MovieScreening (CinemaID FK, ScreenNumber FK, MovieID FK, DateOfLastScreening)

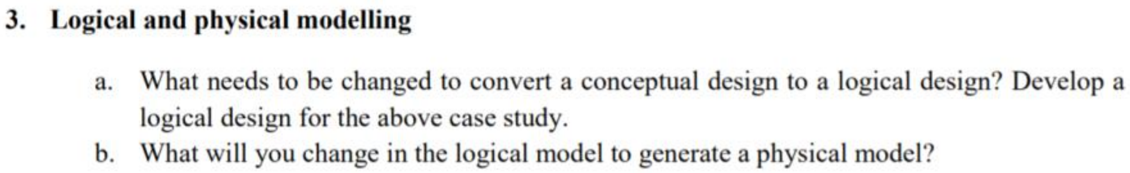
Movie (MovieID, Title, YearOfRelease, Rating)

Projector (SerialNumber, Format, ModelNumber, Resolution, HoursOfUse)

Technically, participation constraints are still part of the logical design, although we don't mention them in this textual notation. The participation constraints will be needed again during development of the physical model.

Note:
- the hardest part is the conceptual model
- going from conceptual to logical is pretty easy (just follow the rules)

3 (b) Physical model



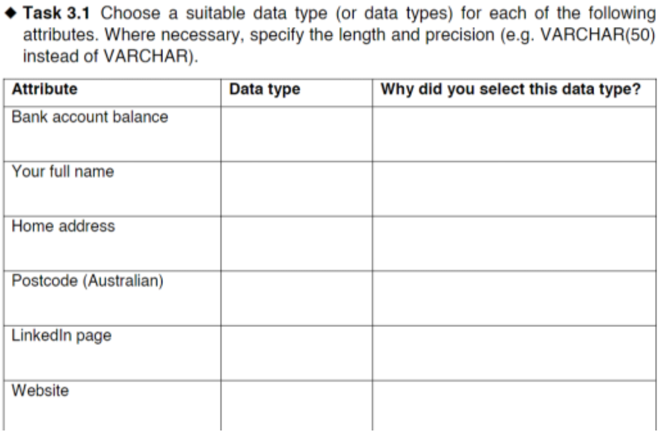
3. Logical and physical modelling

a. What needs to be changed to convert a conceptual design to a logical design? Develop a logical design for the above case study.

b. What will you change in the logical model to generate a physical model?

-
- Choose DBMS e.g. MySQL
- For each column:
  - add data types
  - specify NULL/NOT NULL

- For every column:
  - **Add data types**
    - read through text for ideas to choose appropriate data type
    - FK columns have same data type as PK column they refer to
    - Usually PKs which are an ID have data type = INT
    - If you aren't sure what data type to use, choose something reasonable. This can be subjective
      - E.g. ModelNumber may not just have numbers, but could have letters and numbers
      - so maybe a good choice is VARCHAR
    - For attributes where the data should be a finite number of known options, use ENUM
      - e.g. Format ENUM('16mm', '35mm', '2D', '3D')
      - Look-up tables can also be used instead of ENUM and can be better as you don't need to know the options beforehand
    - *Look at the Labs for more tips on choosing data types*
      - *Week 3 Lab has a large section on it*

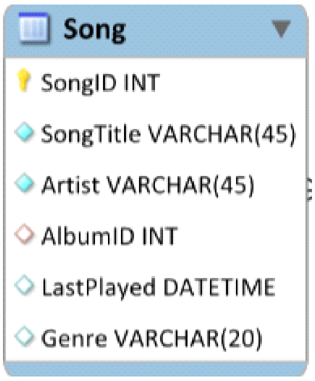


◆ **Task 3.1** Choose a suitable data type (or data types) for each of the following attributes. Where necessary, specify the length and precision (e.g. VARCHAR(50) instead of VARCHAR).

Attribute	**Data type**	**Why did you select this data type?**
Bank account balance		
Your full name		
Home address		
Postcode (Australian)		
LinkedIn page		
Website		

- **Specify NOT NULL/NULL**
  - if a column is specified as NOT NULL, it means the DBMS will not accept NULL values if they are entered in that column. The attribute MUST be filled it, it can't be left empty.

  - if a column is specified as NULL, DBMS will accept NULL values if they are entered



Song

SongID INT

SongTitle VARCHAR(45)

Artist VARCHAR(45)

AlbumID INT

LastPlayed DATETIME

Genre VARCHAR(20)

  - For PK columns
    - always NOT NULL

  - For FK columns
    - look at the ER model
    - is the participation constraint total or partial?
    - if total: FK must be NOT NULL
    - if partial: FK can be NULL
    - e.g.
      - Screen has total participation in Screen-Cinema relationship
      - this means the FK CinemaID must be NOT NULL
    - e.g.
      - Screen has partial participation in Screen-Projector relationship (not every screen is projected onto by a projector)
      - so the ProjectorSerialNumber FK in the Screen relation should be NULL (it is optional)

  - For other columns
    - think - is this info required or optional?
    - e.g. YearlyRent should be specified as NULL since some Cinemas may not have a yearly rent if owned outright
    - e.g. but every movie has a title, so that should be NOT NULL