ONESTOP MEDIA LIBRARY

Logical database design – phase 2

Addis Ababa institute of technology

SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

DATABASE SYSTEMS PROJECT

Group 3

AUTHORS

Names	ID	E-mail	GitHub
Adonay Dereje	ATR/7743/10	donniedereje1@gmail.com	adonaydereje
Eyob Eshetu	ATR/0164/10	eyobgeletu@tutanota.com	eyob-geletu
Fekadesilassie Tsegaye	ATR/5519/10	beshoo.f1990@gmail.com	beshoo-07
Fraol Bereket	ATR/8289/10	fraolbereket@gmail.com	fraolb
Henok Hailu	ATR/4973/10	henokhailu96@gmail.com	henokhailu96

Phase-2: Logical database design

In this phase we tried to design a relational database and map our ER-diagram to the relational database. We prepared tables for entities, classes and subclasses and their attributes by following the algorithms that we learnt during lecture classes. After that we pointed out the functional dependencies of each table and tried to minimize them. Finally we made sure the tables are in normalized forms. The normalization extends from First Normal Form (1NF) to Boyce-Codd Normal Form (BCNF).

1. Mapping

- Primary Key and Foreign Key identification
- Tables for Normal Entities
- Tables for Super classes and Subclasses by adding an attribute

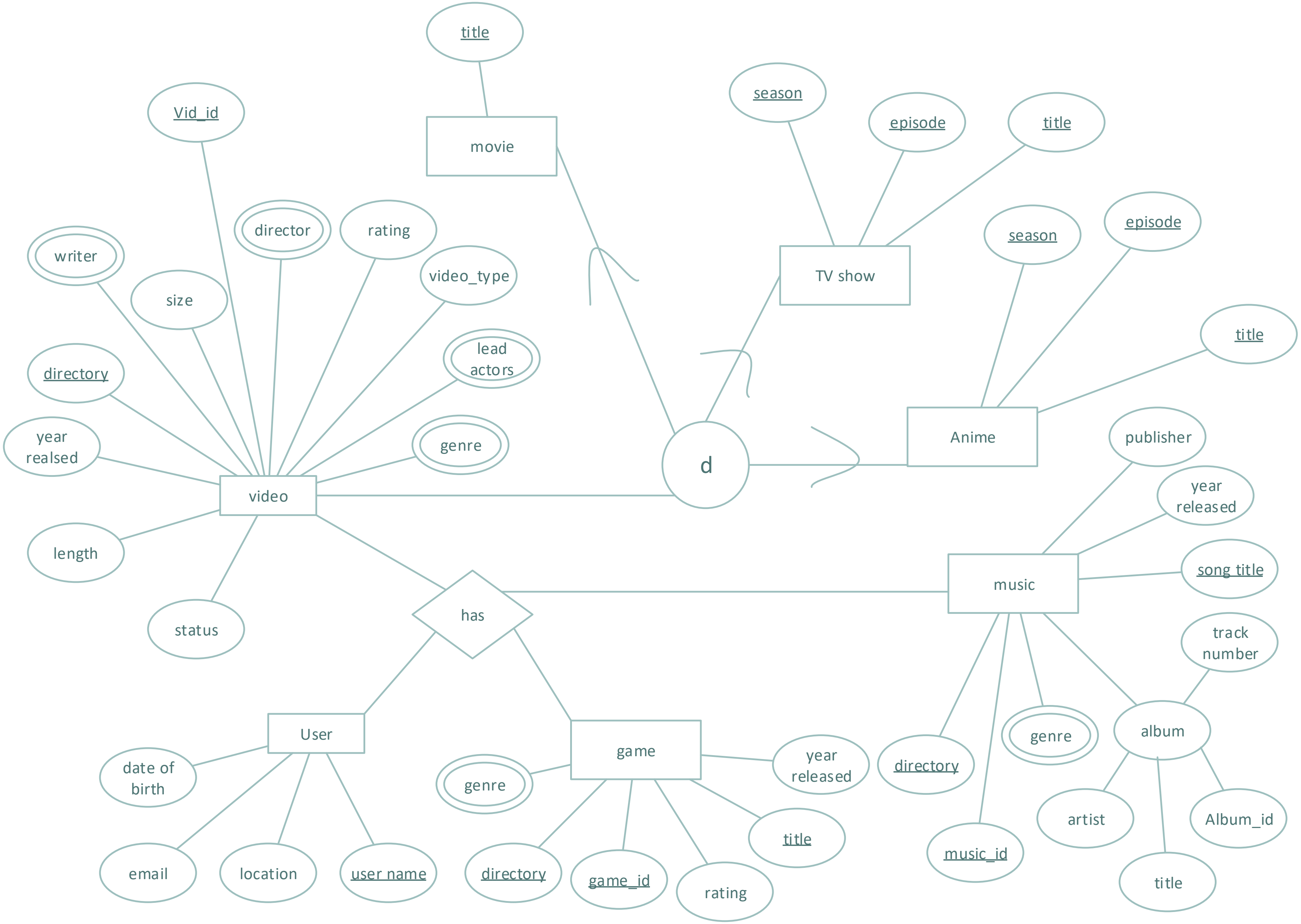
2. Functional dependencies

- Extraction of functional dependency of each table
- Minimize Functional Dependencies
- Finding the closed dependency

3. Normalization

- Finding out the normalization states
- Normalize to 1NF, 2NF, 3NF and BCNF

Additionally, we also tried to modify our previous ER-diagram by adding missing and necessary attributes, removing unnecessary relations and entities, creating superclass and subclasses for video, movie, TV show and Anime entities. This is shown in the diagram shown on the next page



title
movie
Vid_id
season
episode
title
TV show
season
episode
writer
director
rating
video_type
size
title
directory
lead actors
year realsed
genre
Anime
d
publisher
video
year released
length
music
song title
has
status
track number
User
genre
album
game
year released
date of birth
genre
directory
artist
Album_id
title
email
location
user name
directory
game_id
rating
music_id
title

1. Mapping

Step 1: Mapping regular entities

We have three regular entities (music, game and user) and a super class called video. The primary keys are music_id, game_id, username and Vid_id respectively. We first create a relation for the simple attributes only.

Music

Music_Id	Song_title	publisher	Rating	Year_released	directory

Game

Game_id	title	Rating	Year_released	directory

User

Username	email	location	DOB

Video

Vid_id	Size	Length	Rating	Year_released	directory	status	Video_Type

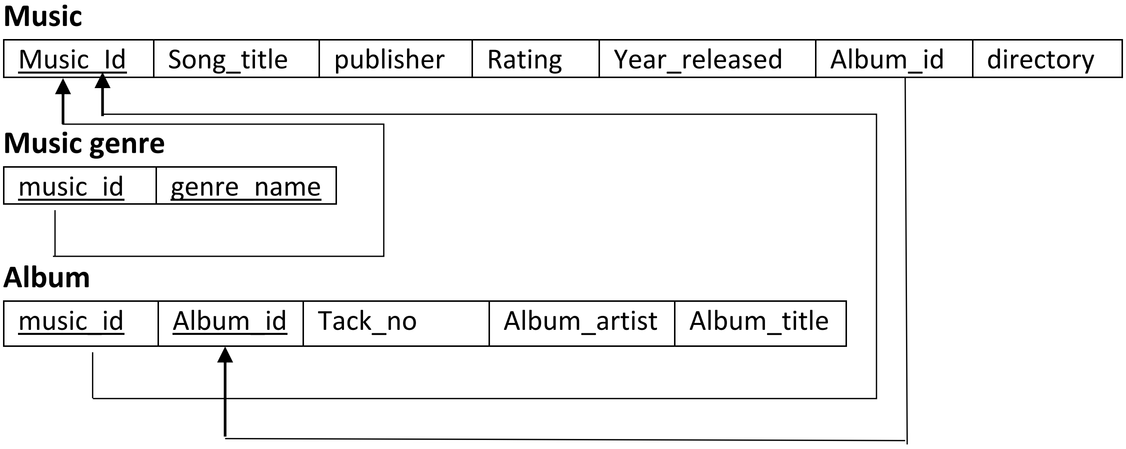
Step 2-5:

- These steps are not applicable in our case.
- We don't have weak entities and binary relations.

Step 6: Mapping multivalued and composite attributes

We have a multivalued attribute called **genre** in game, music and video entities and also attributes called **writer, director** and **lead actors** for video entity in particular. We create a relation for each of these attributes. The primary key of the relations will be the attributes and a foreign key from the entity they were a part of.

We also created a relation called Album for the composite attribute album in music entity. The primary key of the relation Album is the combination of a foreign key from music entity and album_id attribute.



Music

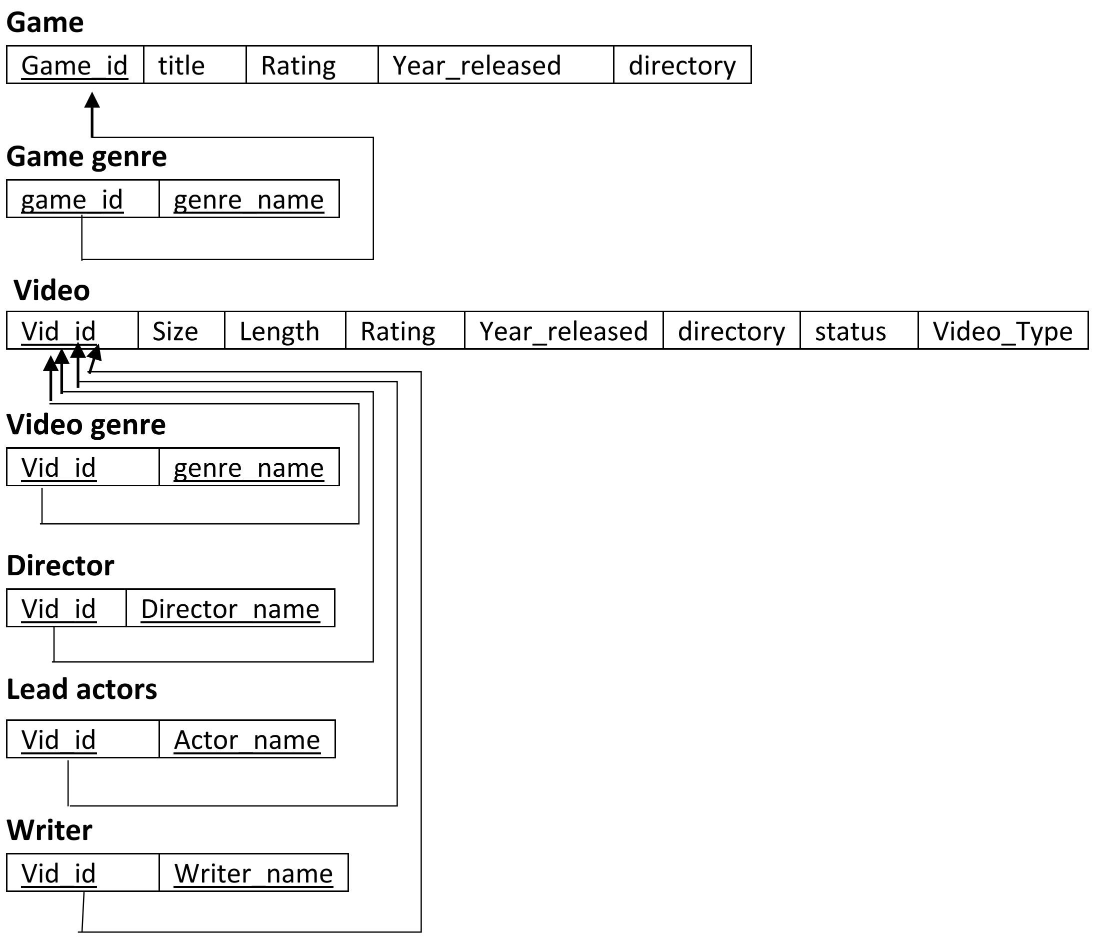
Music_Id	Song_title	publisher	Rating	Year_released	Album_id	directory

Music genre

music_id	genre_name

Album

music_id	Album_id	Tack_no	Album_artist	Album_title



Game

Game_id	title	Rating	Year_released	directory

Game genre

game_id	genre_name

Video

Vid_id	Size	Length	Rating	Year_released	directory	status	Video_Type

Video genre

Vid_id	genre_name

Director

Vid_id	Director_name

Lead actors

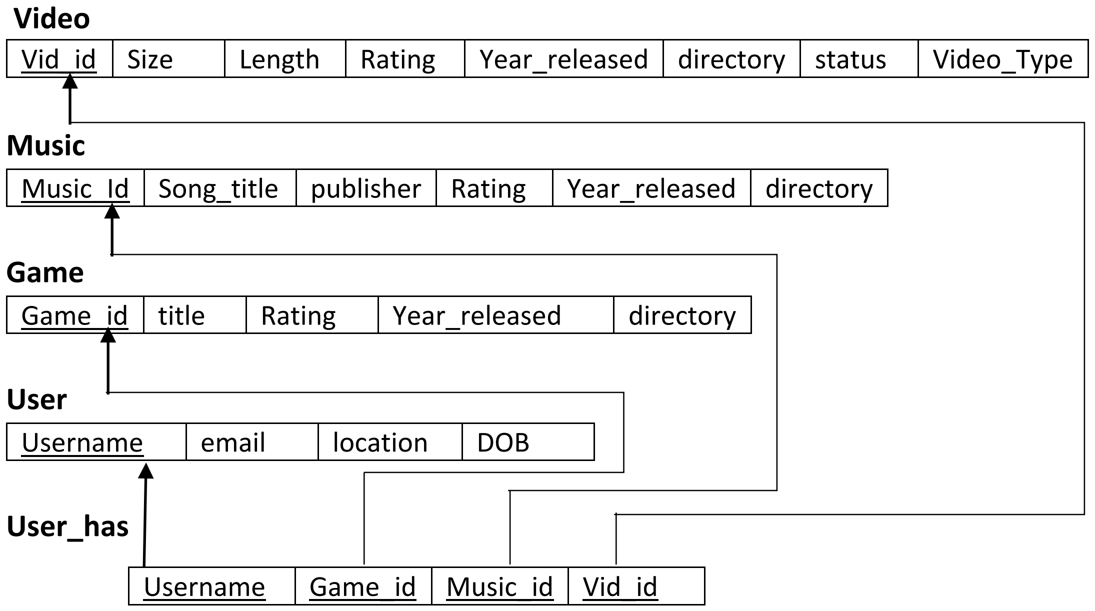
Vid_id	Actor_name

Writer

Vid_id	Writer_name

Step 7: Mapping N-ary relations

We have a relation called "has" between user and video, music & game entities. This is a tertiary relationship. It defines to whom the information belongs to if there is a second user. For this we created a relation called **user has** and it has a composite primary key by taking foreign keys form the four participating entities.



Video

Vid_id	Size	Length	Rating	Year_released	directory	status	Video_Type

Music

Music_Id	Song_title	publisher	Rating	Year_released	directory

Game

Game_id	title	Rating	Year_released	directory

User

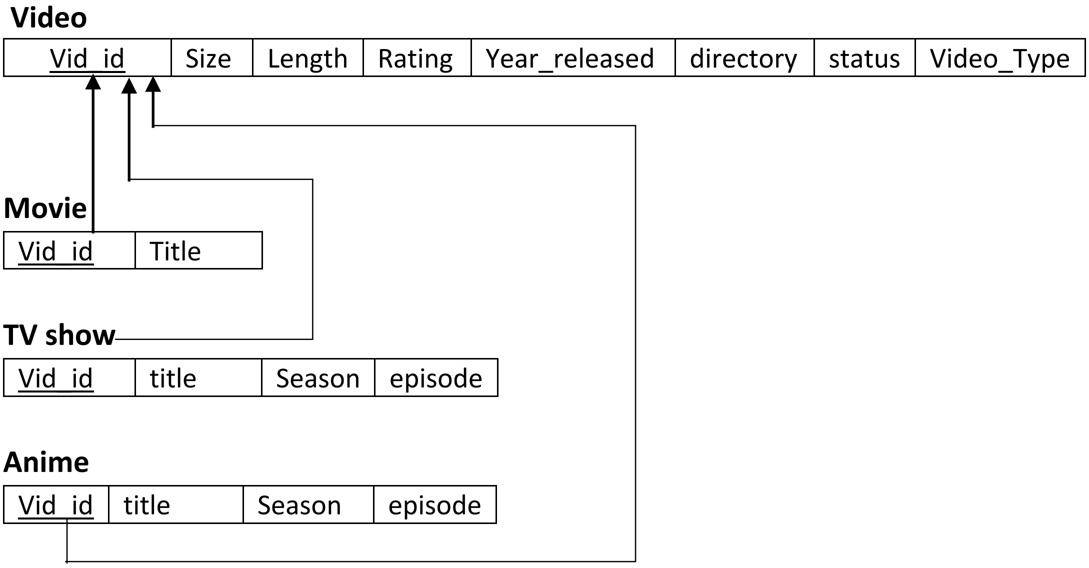
Username	email	location	DOB

User_has

Username	Game_id	Music_id	Vid_id

Step 8: Mapping Specializations

In our case we have a super class called video and subclasses of video that define the type of the videos called **movie, TV show** and **anime**. We mapped this using the **multiple relations Superclass and subclasses** option.



Video

Vid_id	Size	Length	Rating	Year_released	directory	status	Video_Type

Movie

Vid_id	Title

TV show

Vid_id	title	Season	episode

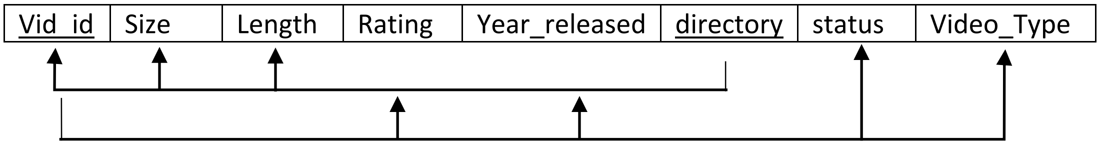
Anime

Vid_id	title	Season	episode

2. Functional dependency

In this section we determined the candidate keys, the functional dependencies of each relation to the keys and the closure of the keys. Transitive dependencies that may occur can be fixed when normalizing the relations.

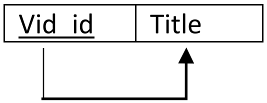
Video



Vid_id	Size	Length	Rating	Year_released	directory	status	Video_Type

- **FD1:** Vid_id → { rating, year_released, status, video_type}
- **Closure of Vid_id** $\{Vid\_id\}^+$ = { Vid_id, rating, year_released, status, video_type}
- **FD2:** Directory → { Vid_id, size, length}
- **Closure of directory** $\{Directory\}^+$ = { Directory, Vid_id, size, length, rating, year_released, status, video_type } = Video

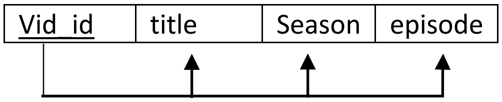
Movie



Vid_id	Title

- **FD1:** Vid_id → { title}
- **Closure of Vid_id** $\{Vid\_id\}^+$ = { Vid_id, title} = Movie

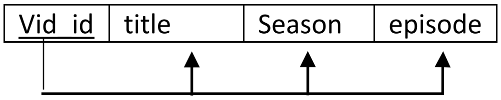
TV show



Vid_id	title	Season	episode

- **FD1:** Vid_id → { title, season, episode}
- **Closure of Vid_id** $\{Vid\_id\}^+$ = { Vid_id, title, season, episode } = TV show

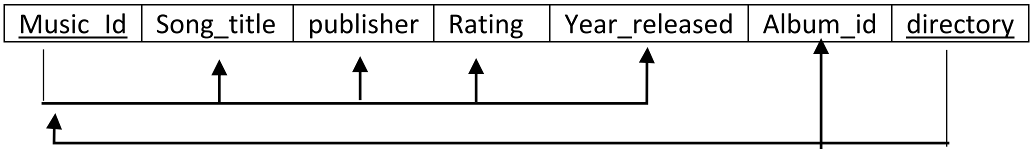
Anime



Vid_id	title	Season	episode

- **FD1:** Vid_id → { title, season, episode}
- **Closure of Vid_id** $\{Vid\_id\}^+$ = { Vid_id, title, season, episode } = Anime

Music



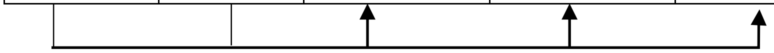
Music_Id	Song_title	publisher	Rating	Year_released	Album_id	directory

- **FD1:** Music_id → { Song_title, publisher, Rating, Year_released }

- **Closure of music_id** {music_id}$^+$ = { music_id, Song_title, publisher, Rating, Year_released }
- **FD2:** Directory → { music_id, Album_id}
- **Closure of Directory** { Directory }$^+$ = { Directory, music_id, Album_id, Song_title, publisher, Rating, Year_released } = Music

Album

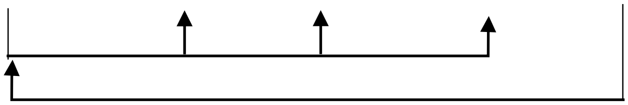
music_id	Album_id	Tack_no	Album_artist	Album_title



- **FD1:** {Music_id, Album_id} → { Tack_no, Album_artist, Album_title }
- **Closure of** {Music_id, Album_id}$^+$ = {Music_id, Album_id, Tack_no, Album_artist, Album_title } = Album

Game

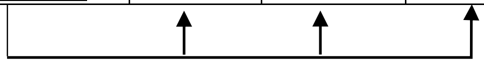
Game_id	title	Rating	Year_released	directory



- **FD1:** game_id → { title, Rating, Year_released }
- **Closure of game_id** { game_id }$^+$ = { game_id, title, Rating, Year_released }
- **FD2:** Directory → { game_id}
- **Closure of Directory** {Directory}$^+$ = {Directory,game_id,title,Rating,Year_released}=Game

User

Username	email	location	DOB



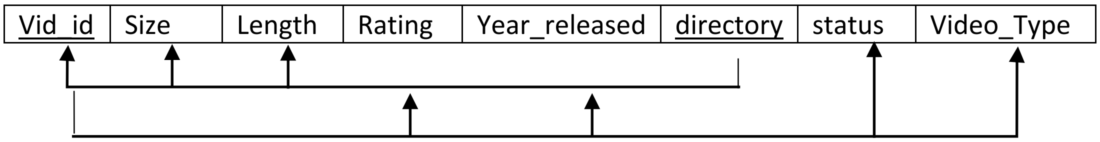
- **FD1:** username → { email, location, DOB }
- **Closure of username** { username }$^+$ = { username, email, location, DOB } = User

❖ The rest of the relations have no dependencies that can be extracted from them. They only contain key attributes.

3. Normalization

Here we tried to determine the normalization form of the relations are in at this point and we further normalized those are not in BCN form already. Most of the relations are In BCNF form already after mapping is done. The relations below are not in BCNF form.

Video



Vid_id	Size	Length	Rating	Year_released	directory	status	Video_Type

- There is a transitive dependency between directory and vid_id
- We have to split this relation into two

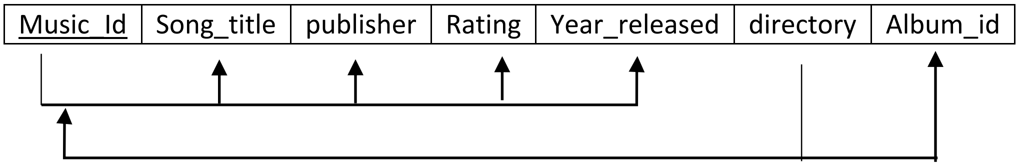
Video File

directory	Size	Length	Vid_id

Video

Vid_id	Rating	Year_released	Video_Type

Music



Music_Id	Song_title	publisher	Rating	Year_released	directory	Album_id

- There is a transitive dependency between directory and music_id
- We have to split this relation into two

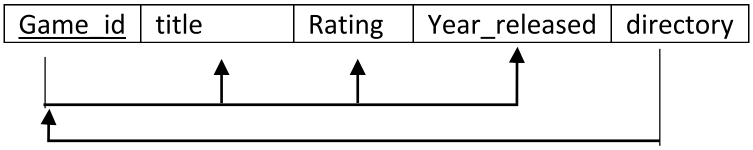
Music file

directory	Music_Id	Album_id

Music

Music_Id	Song_title	publisher	Rating	Year_released

Game



Game_id	title	Rating	Year_released	directory

- There is a transitive dependency between directory and game_id
- We have to split this relation into two

Game

Game_id	title	Rating	Year_released

Game file

directory	Game_id

- ❖ Now all relations are in BCNF form.
- ❖ Now the relation table of the database will be as follows

Video File

directory	Size	Length	Vid_id

Video

Vid_id	Rating	Year_released	Video_Type

Video genre

Vid_id	genre_name

Director

Vid_id	Director_name

Lead actors

Vid_id	Actor_name

Writer

Vid_id	Writer_name

Movie

Vid_id	Title

TV show

Vid_id	title	Season	episode

Anime

Vid_id	title	Season	episode

Music file

directory	Music_Id	Album_id

Music

Music_Id	Song_title	publisher	Rating	Year_released

Music genre

music_id	genre_name

Album

music_id	Album_id	Tack_no	Album_artist	Album_title

Game file

directory	Game_id

Game

Game_id	title	Rating	Year_released

Game genre

game_id	genre_name

User

Username	email	location	DOB