# Lab Report 3 – Part 2

## Databases - CSC 3320 121

### Hudson Arney

### 16 February 2024

# Part 2.1 — finish transferring data to final tables.

1. ERD:

A diagram of a company

Description automatically generated

Relational Schema:

A diagram of a person's identification

Description automatically generated

1. (You may have done this as part of Lab3, part1.) In your table creation script (previous step 7), **add the SQL to load your final schema tables from your imported tables**. This can be a very involved and time-consuming step. We will talk about some strategies in class. See [this example](https://msoe.instructure.com/courses/18290/files/2350478?wrap=1)[Download this example](https://msoe.instructure.com/courses/18290/files/2350478/download?download_frd=1) for how this works in the context of the world\_db.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

3. (You may have done this as part of Lab3, part1.) Verify you have primary key and foreign key constraints (when needed) for each entity and relationship. If not present, create them.

* ***Question for Observation:***Why would I create the primary key index after the table has been created and the data imported versus defining the primary key in the table definition?

Allows for more flexibility in data loading and permits the use of composite primary keys or auto-generated primary key values as needed.

1. (You may have done this as part of Lab3, part1.) Run select commands on each table to verify the data has been successfully imported. Note that you will have 3 "load data" tables (based on the 3 text files you load data from), and then an additional 4 to 8 final design tables based on your ER design. Once we copy data over from the load tables into the final design tables, we no longer use the load tables (but don't delete them in case you mess up and need to reload your final tables).

Query used (For parts 1+2+3):

CREATE TABLE Actor

(

Actor\_ID INT NOT NULL,

Name VARCHAR(50) NOT NULL,

Recording\_ID INT NOT NULL,

PRIMARY KEY (Actor\_ID)

)

ENGINE=INNODB;

CREATE TABLE Category

(

Name VARCHAR(50) NOT NULL,

Category\_ID INT NOT NULL,

PRIMARY KEY (Category\_ID)

)

ENGINE=INNODB;

CREATE TABLE Director

(

Name VARCHAR(50) NOT NULL,

Director\_ID INT AUTO\_INCREMENT,

PRIMARY KEY (Director\_ID)

)

ENGINE=INNODB;

CREATE TABLE Rating

(

Name VARCHAR(50) NOT NULL,

Rating\_ID INT AUTO\_INCREMENT,

PRIMARY KEY (Rating\_ID)

)

ENGINE=INNODB;

CREATE TABLE Recording

(

Recording\_ID INT NOT NULL,

Director VARCHAR(50) NOT NULL,

Title VARCHAR(50) NOT NULL,

Category VARCHAR(50) NOT NULL,

Image\_Name VARCHAR(50) NOT NULL,

Duration INT NOT NULL,

Rating VARCHAR(10) NOT NULL,

Year\_Released INT NOT NULL,

Price DOUBLE NOT NULL,

Stock\_Count INT NOT NULL,

PRIMARY KEY (Recording\_ID)

)

ENGINE=INNODB;

-- ------------------------------------------------------------------------------------------------

CREATE TABLE Associated\_With

(

Recording\_ID INT NOT NULL,

Actor\_ID INT NOT NULL,

PRIMARY KEY (Recording\_ID, Actor\_ID),

FOREIGN KEY (Recording\_ID) REFERENCES Recording(Recording\_ID),

FOREIGN KEY (Actor\_ID) REFERENCES Actor(Actor\_ID)

)

ENGINE=INNODB;

-- ------------------------------------------------------------------------------------------------

-- Populate Actor table

INSERT INTO Actor (Actor\_ID, Name, Recording\_ID)

SELECT DISTINCT Actor\_ID, Name, Recording\_ID

FROM video\_actors;

-- Populate Recording table

INSERT INTO Recording (Recording\_ID, Director, Title, Category, Image\_Name, Duration, Rating, Year\_Released, Price, Stock\_Count)

SELECT DISTINCT Recording\_ID, Director, Title, Category, Image\_Name, Duration, Rating, Year\_Released, Price, Stock\_Count

FROM video\_recordings;

INSERT INTO Category (Category\_ID, Name)

SELECT DISTINCT Category\_ID, Name

FROM video\_categories;

INSERT INTO Director (Name)

SELECT DISTINCT TRIM(Director)

FROM video\_recordings;

INSERT INTO Rating (Name)

SELECT DISTINCT TRIM(Rating)

FROM video\_recordings;

INSERT INTO Associated\_With (Recording\_ID, Actor\_ID)

SELECT Recording\_ID, Actor\_ID

FROM Actor;

# Part 2.2 — SQL

1. Run this query:

select \* from Video\_Recordings, Video\_Categories;

Note the cross-product effect of joining two tables. Record the number of rows generated. Do all  
permutations of Video Recordings × Video Categories make sense? Explain.

330 rows were generated (55 recordings \* 6 categories). The permutations do not make sense as it is adding every category to every recording. This makes the category lose its value as a label for a recording.

A screenshot of a computer screen

Description automatically generated

2. Run this query:

select \* from Video\_Recordings vr, Video\_Categories vc where vr.category=vc.name;

Note the cross-product effect of joining two tables when restricted on the appropriate keys. Record the number of rows generated. Explain the purpose of the join.

There were 55 rows generated (the number of recordings). This query makes much more sense as it groups recordings based on category, and keeps it assigning only one category per recording.

A screenshot of a computer

Description automatically generated

#### **For the remaining questions, use your final design tables.**

1. List the number of videos for each video category.

Query Used: select category, COUNT(\*) from recording group by category;

A screenshot of a computer

Description automatically generated

4. List the number of videos for each video category where the inventory is non-zero.

Query Used:

select category, COUNT(\*) from recording where Stock\_Count > 0 group by category;

A screenshot of a computer

Description automatically generated

5. For each actor, list the video categories that actor has appeared in.

Query Used:

SELECT Actor.Name AS Actor\_Name, Category.Name AS Category\_Name FROM Actor

JOIN Associated\_With ON Actor.Actor\_ID = Associated\_With.Actor\_ID

JOIN Recording ON Associated\_With.Recording\_ID = Recording.Recording\_ID

JOIN Category ON Recording.Category = Category.Name

ORDER BY Actor\_Name, Category\_Name;

A screenshot of a computer

Description automatically generated

6. Which actors have appeared in movies in different video categories?

Query Used:

SELECT Actor.Name AS Actor\_Name, COUNT(DISTINCT Category.Name) AS Num\_Different\_Categories FROM Actor

JOIN Associated\_With ON Actor.Actor\_ID = Associated\_With.Actor\_ID

JOIN Recording ON Associated\_With.Recording\_ID = Recording.Recording\_ID

JOIN Category ON Recording.Category = Category.Name

GROUP BY Actor.Name

HAVING Num\_Different\_Categories > 1

ORDER BY Num\_Different\_Categories DESC;

A screenshot of a computer

Description automatically generated

7. Which actors have not appeared in a comedy?

Returned 272 Rows, so only 100 Actors have appeared in comedies.

Query Used:

SELECT DISTINCT Name AS Actor\_Name FROM Actor

WHERE Actor\_ID NOT IN (

SELECT DISTINCT Actor\_ID FROM Associated\_With

JOIN Recording ON Associated\_With.Recording\_ID = Recording.Recording\_ID

WHERE Recording.Category = 'Comedy');

A screenshot of a computer

Description automatically generated

8. Which actors have appeared in both a comedy and an action adventure movie?

Query Used:

SELECT DISTINCT a.Name AS Actor\_Name

FROM Actor a

JOIN Associated\_With aw ON a.Actor\_ID = aw.Actor\_ID

JOIN Recording r ON aw.Recording\_ID = r.Recording\_ID

JOIN Category c ON r.Category = c.Name

WHERE c.Name IN ('Comedy', 'Action & Adventure')

GROUP BY a.Name

HAVING COUNT(DISTINCT CASE WHEN c.Name = 'Comedy' THEN r.Recording\_ID END) > 0

AND COUNT(DISTINCT CASE WHEN c.Name = 'Action & Adventure' THEN r.Recording\_ID END) > 0;

A screen shot of a computer program

Description automatically generated

There were only these 4 actors in both.

9. Come up with your own question that requires you join at least three tables. List the question, the SQL you used to answer it, and the answer itself.

Which actors appear in both multiple ratings and categories?

Query Used:

SELECT DISTINCT

a.Name AS Actor\_Name,

Num\_Different\_Ratings,

Rating\_Names,

Num\_Different\_Categories,

Category\_Names

FROM Actor a

JOIN (

SELECT

a.Name AS Actor\_Name,

COUNT(DISTINCT r.Rating) AS Num\_Different\_Ratings,

GROUP\_CONCAT(DISTINCT r.Rating ORDER BY r.Rating) AS Rating\_Names,

COUNT(DISTINCT r.Category) AS Num\_Different\_Categories,

GROUP\_CONCAT(DISTINCT r.Category ORDER BY r.Category) AS Category\_Names

FROM Associated\_With aw

JOIN Recording r ON aw.Recording\_ID = r.Recording\_ID

JOIN Actor a ON aw.Actor\_ID = a.Actor\_ID

GROUP BY a.Name

) AS Counts ON a.Name = Counts.Actor\_Name

WHERE Num\_Different\_Ratings > 1 AND Num\_Different\_Categories > 1;

A screenshot of a computer

Description automatically generated

There were 15 actors in this group. I guess it makes sense because the Ratings are somewhat correlating with Categories. (Action is more likely to be PG and Suspense is more likely to be R for example).