David Schulz
CS 3860 041
Lab 3
10/6/2020

Lab 3 Report

Part I – Create a Video Database

1. Review videodb-readme.txt
2. Create a Video database
    a. CREATE DATABASE Video;
3. Create an SQL script to create import tables for importing the data as tab-delimited text files

```
CREATE TABLE `Video`.`Video_Recordings` (
  `recording_id` INT NOT NULL,
  `director` VARCHAR(45) NULL,
  `title` VARCHAR(45) NULL,
  `category` VARCHAR(45) NULL,
  `image_name` VARCHAR(45) NULL,
  `duration` INT NULL,
  `rating` VARCHAR(5) NULL,
  `year_released` INT NULL,
  `price` FLOAT NULL,
  `stock_count` INT NULL,
  PRIMARY KEY (`recording_id`),
  UNIQUE INDEX `recording_id_UNIQUE` (`recording_id` ASC) VISIBLE);

CREATE TABLE `Video`.`Video_Categories` (
  `id` INT NOT NULL,
  `name` VARCHAR(45) NULL,
  PRIMARY KEY (`id`),
  UNIQUE INDEX `id_UNIQUE` (`id` ASC) VISIBLE);

CREATE TABLE `Video`.`Video_Actors` (
  `id` INT NOT NULL,
  `name` VARCHAR(45) NULL,
  `recording_id` INT NULL,
  PRIMARY KEY (`id`),
  UNIQUE INDEX `id_UNIQUE` (`id` ASC) VISIBLE);
```

    a.
    b. Why would you select one format over another?
        i. If you expect one of the delimiter characters to be in the data itself, you would want to use the other character as the delimiter. For example, if commas are expected to be in the data, such as “Milwaukee,WI”, then you would want to use tabs as the delimiter so that the commas in the data are not confused for delimiters instead.
4. Load the tables from the data files. Use select commands on each table to verify the data has been successfully imported

```
LOAD DATA LOCAL Infile 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/Video_Recordings.txt' into table Video.Video_Recordings
fields terminated by '\t'
optionally enclosed by '"'
lines terminated by '\r\n';

LOAD DATA LOCAL Infile 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/Video_Categories.txt' into table Video.Video_Categories
fields terminated by '\t'
optionally enclosed by '"'
lines terminated by '\r\n';

LOAD DATA LOCAL Infile 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/Video_Actors.txt' into table Video.Video_Actors
fields terminated by '\t'
optionally enclosed by '"'
lines terminated by '\r\n';
```

a. SELECT * FROM Video.Video_Recordings;

b. Returned 55 rows

recording_id	director	title	category	image_name	duration	rating	year_released	price	stock_count
3000	Francis Ford Coppola	Apocalypse Now	Action & Adventure	apocalypse_now.gif	9180	R	1979	22.99	0
3001	Michael Bay	Bad Boys	Action & Adventure	badboys.gif	7140	R	1995	15.99	782
3002	Mel Gibson	Braveheart	Action & Adventure	braveheart.gif	10620	R	1995	14.99	582
3003	Mimi Leder	Deep Impact	Action & Adventure	deep_impact.gif	5700	PG-13	1998	11.99	501
3004	J. Robert Wagoner	Disco Godfather	Action & Adventure	disco_godfather.gif	5580	R	1993	10.99	872
3005	Stanley Kubrick	Full Metal Jacket	Action & Adventure	full_metal_jacket.gif	7020	R	1987	16.99	872
3006	Roland Emmerich	Independence Day	Action & Adventure	independence_day.gif	8700	PG-13	1996	16.99	0
3007	John McTiernan	The Hunt for Red October	Action & Adventure	hunt_for_red_october.gif	8100	PG	1990	10.99	618
3008	Michael Bay	The Rock	Action & Adventure	the_rock.gif	8160	R	1996	20.99	514
3009	Tony Scott	Top Gun	Action & Adventure	top_gun.gif	6600	PG	1986	11.99	499
3010	Jay Roach	Austin Powers	Comedy	austin_powers.gif	5220	PG-13	1997	17.99	688
3011	Harold Ramis	Caddyshack	Comedy	caddyshack.gif	5880	R	1980	18.99	938

c. SELECT * FROM Video.Video_Categories;

d. Returned 6 rows

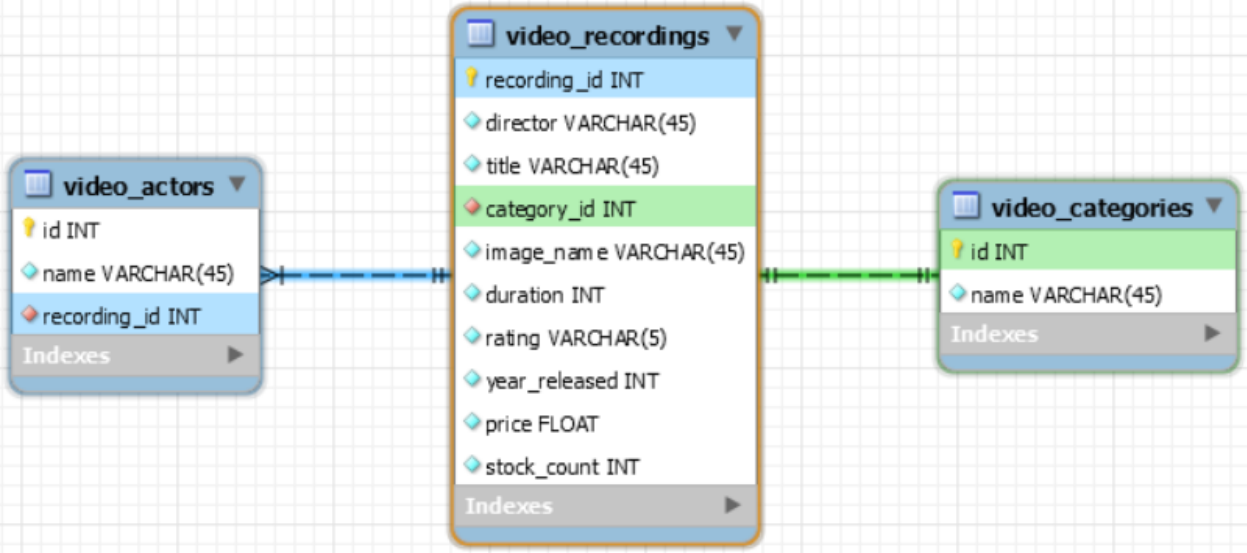
id	name
0	Action & Adventure
1	Comedy
2	Drama
3	Horror
4	Science Fiction
5	Suspense

e. SELECT * FROM Video.Video_Actors;

f. Returned 372 rows

id	name	recording_id
0	Marlon Brando	3000
1	Martin Sheen	3000
2	Robert Duvall	3000
3	Frederic Forrest	3000
4	Dennis Hopper	3000
5	Scott Glenn	3000
6	Harrison Ford	3000
7	Laurence Fishburne	3000
8	Sam Bottoms	3000
9	Will Smith	3001
10	Martin Lawrence	3001
11	Tea Leoni	3001

5. Create both an ERD and a relational schema for your final data model



6. From MySQL Workbench, generate an SQL script to define and create the database tables for your DB schema. Use ENGINE=INNODB at the end of each create table statement, but before the semi-colon

```
CREATE SCHEMA IF NOT EXISTS `rel_Video` DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci ;
USE `rel_Video` ;

-- -----------------------------------------------------
-- Table `rel_Video`.`video_categories`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `rel_Video`.`video_categories` (
  `id` INT NOT NULL,
  `name` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE INDEX `id_UNIQUE` (`id` ASC) VISIBLE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
```

```sql
-- -----------------------------------------------------
-- Table `rel_Video`.`video_recordings`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `rel_Video`.`video_recordings` (
  `recording_id` INT NOT NULL,
  `director` VARCHAR(45) NOT NULL,
  `title` VARCHAR(45) NOT NULL,
  `category_id` INT NOT NULL,
  `image_name` VARCHAR(45) NOT NULL,
  `duration` INT NOT NULL,
  `rating` VARCHAR(5) NOT NULL,
  `year_released` INT NOT NULL,
  `price` FLOAT NOT NULL,
  `stock_count` INT NOT NULL,
  PRIMARY KEY (`recording_id`),
  UNIQUE INDEX `recording_id_UNIQUE` (`recording_id` ASC) VISIBLE,
  INDEX `fk_video_recordings_video_categories1_idx` (`category_id` ASC) VISIBLE,
  CONSTRAINT `fk_video_recordings_video_categories1`
    FOREIGN KEY (`category_id`)
    REFERENCES `video`.`video_categories` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
```

```
-- -----------------------------------------------------
-- Table `rel_Video`.`video_actors`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `rel_Video`.`video_actors` (
  `id` INT NOT NULL,
  `name` VARCHAR(45) NOT NULL,
  `recording_id` INT NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE INDEX `id_UNIQUE` (`id` ASC) VISIBLE,
  INDEX `fk_video_actors_video_recordings_idx` (`recording_id` ASC) VISIBLE,
  CONSTRAINT `fk_video_actors_video_recordings`
    FOREIGN KEY (`recording_id`)
    REFERENCES `video`.`video_recordings` (`recording_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
```

7. Draft SQL to load your final schema tables from your import tables
    a. SQL: INSERT INTO video_actors SELECT * FROM video.video_actors;
    INSERT INTO video_categories SELECT * FROM video.video_categories;
    CREATE TEMPORARY TABLE temp AS SELECT * FROM video.video_recordings;
    UPDATE temp SET category=0 WHERE category='Action & Adventure';
    UPDATE temp SET category=1 WHERE category='Comedy';
    UPDATE temp SET category=2 WHERE category='Drama';
    UPDATE temp SET category=3 WHERE category='Horror';
    UPDATE temp SET category=4 WHERE category='Science Fiction';
    UPDATE temp SET category=5 WHERE category='Suspense';
    ALTER TABLE temp CHANGE COLUMN category category_id INT;
    INSERT INTO video_recordings SELECT * FROM temp;
8. Verify primary key and foreign key constraints for each relation. If not present, create them
    a. They were already present.
    b. 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?
        i. If the primary key is defined during table creation, and if there's a chance the column being used as the primary key may have duplicates, it will skip any rows with duplicate primary keys when importing data. It's safer to import the data first to make sure you have it all and then make sure the column you want can be used as a primary key.
9. Run select commands on each table to verify the data has been successfully imported
    a. SQL: SELECT * FROM video_recordings;
    SELECT * FROM video_actors;
    SELECT * FROM video_categories;

Part II – Document and Answer Questions Using SQL

1. Execute *select * from Video_Recordings, Video_Categories*. Notice the cross-product effect of joining two tables. Record the number of rows generated. Do all permutations of Video_Recordings X Video_Categories make sense? Explain.
    a. SQL: SELECT * FROM Video_Recordings, Video_Categories
    b. Returned 330 rows

recording_id	director	title	category	image_name	duration	rating	year_released	price	stock_count	id	name
3000	Francis Ford Coppola	Apocalypse Now	Action & Adventure	apocalypse_now.gif	9180	R	1979	22.99	0	0	Action & Adventure
3000	Francis Ford Coppola	Apocalypse Now	Action & Adventure	apocalypse_now.gif	9180	R	1979	22.99	0	1	Comedy
3000	Francis Ford Coppola	Apocalypse Now	Action & Adventure	apocalypse_now.gif	9180	R	1979	22.99	0	2	Drama
3000	Francis Ford Coppola	Apocalypse Now	Action & Adventure	apocalypse_now.gif	9180	R	1979	22.99	0	3	Horror
3000	Francis Ford Coppola	Apocalypse Now	Action & Adventure	apocalypse_now.gif	9180	R	1979	22.99	0	4	Science Fiction
3000	Francis Ford Coppola	Apocalypse Now	Action & Adventure	apocalypse_now.gif	9180	R	1979	22.99	0	5	Suspense
3001	Michael Bay	Bad Boys	Action & Adventure	badboys.gif	7140	R	1995	15.99	782	0	Action & Adventure
3001	Michael Bay	Bad Boys	Action & Adventure	badboys.gif	7140	R	1995	15.99	782	1	Comedy
3001	Michael Bay	Bad Boys	Action & Adventure	badboys.gif	7140	R	1995	15.99	782	2	Drama
3001	Michael Bay	Bad Boys	Action & Adventure	badboys.gif	7140	R	1995	15.99	782	3	Horror

    c. No they don't because all 6 different categories are joined with each recording, multiplying the number of rows of the original Video_Recordings table (55) by 6.
2. Execute *select * from Video_Recordings vr, Video_Categories vc where vr.category=vc.name*. Notice the cross-product of joining two tables when restricted on the appropriate keys. Record the number of rows generated. Explain the purpose of the join.
    a. SQL: SELECT * FROM Video_Recordings vr, Video_Categories vc WHERE vr.category=vc.name
    b. Returned 55 rows

recording_id	director	title	category	image_name	duration	rating	year_released	price	stock_count	id	name
3000	Francis Ford Coppola	Apocalypse Now	Action & Adventure	apocalypse_now.gif	9180	R	1979	22.99	0	0	Action & Adventure
3001	Michael Bay	Bad Boys	Action & Adventure	badboys.gif	7140	R	1995	15.99	782	0	Action & Adventure
3002	Mel Gibson	Braveheart	Action & Adventure	braveheart.gif	10620	R	1995	14.99	582	0	Action & Adventure
3003	Mimi Leder	Deep Impact	Action & Adventure	deep_impact.gif	5700	PG-13	1998	11.99	501	0	Action & Adventure
3004	J. Robert Wagoner	Disco Godfather	Action & Adventure	disco_godfather.gif	5580	R	1993	10.99	872	0	Action & Adventure
3005	Stanley Kubrick	Full Metal Jacket	Action & Adventure	full_metal_jacket.gif	7020	R	1987	16.99	872	0	Action & Adventure
3006	Roland Emmerich	Independence Day	Action & Adventure	independence_day.gif	8700	PG-13	1996	16.99	0	0	Action & Adventure
3007	John McTiernan	The Hunt for Red October	Action & Adventure	hunt_for_red_october.gif	8100	PG	1990	10.99	618	0	Action & Adventure
3008	Michael Bay	The Rock	Action & Adventure	the_rock.gif	8160	R	1996	20.99	514	0	Action & Adventure
3009	Tony Scott	Top Gun	Action & Adventure	top_gun.gif	6600	PG	1986	11.99	499	0	Action & Adventure

    c. The purpose of the join is to link each recording to its corresponding category. It appears the end goal is to remove the category from the recordings table and instead replace it with the category id to be linked to the category name from the categories table.
3. List the number of videos for each video category.
    a. SQL: SELECT vr.category, Count(*) FROM Video_Recordings vr GROUP BY vr.category
    b. Returned 6 rows

category	Count(*)
Action & Adventure	10
Comedy	10
Drama	10
Horror	8
Science Fiction	9
Suspense	8

4. List the number of videos for each video category where the inventory is non-zero.
    a. SQL: SELECT vr.category, Count(*) FROM Video_Recordings vr WHERE NOT vr.stock_count = 0 GROUP BY vr.category
    b. Returned 6 rows

category	Count(*)
Action & Adventure	8
Comedy	8
Drama	9
Horror	6
Science Fiction	8
Suspense	6

5. For each actor, list the video categories.
    a. SQL: SELECT va.name, GROUP_CONCAT(vr.category) AS "categories"
    FROM Video_Actors va, Video_Recordings vr WHERE va.recording_id=vr.recording_id
    GROUP BY va.name
    b. Returned 335 rows

name	categories
Adam Baldwin	Action & Adventure
Adrian Pasdar	Action & Adventure
Adrienne Corri	Science Fiction
Adrienne King	Horror
Alec Baldwin	Suspense,Action & Adventure
Alec Guinness	Science Fiction
Alfre Woodard	Comedy
Alice Krige	Science Fiction
Amy Irving	Horror
Angela Bassett	Science Fiction

6. Which actors have appeared in movies in different video categories?
    a. SQL: SELECT actor_cats.name, Count(*) FROM
    (SELECT va.name, vr.category, Count(*)
    FROM Video_Actors va, Video_Recordings vr
    WHERE va.recording_id=vr.recording_id
    GROUP BY va.name, vr.category)
    actor_cats GROUP BY actor_cats.name HAVING Count(*) > 1
    b. Returned 25 rows

name	Count(*)
Scott Glenn	2
Harrison Ford	2
Morgan Freeman	3
Rudy Ray Moore	2
Jerry Jones	2
Lady Reed	2
Jeff Goldblum	2
Bill Pullman	2
James Rebhorn	2
Harvey Fierstein	2

7. Which actors have not appeared in a comedy?
    a. SQL: SELECT va.name FROM Video_Actors va WHERE va.name NOT IN
    (SELECT va.name FROM Video_Recordings vr, Video_Actors va
    WHERE vr.recording_id=va.recording_id AND vr.category='Comedy')

GROUP BY va.name

b. Returned 265 rows

name
Marlon Brando
Martin Sheen
Robert Duvall
Frederic Forrest
Dennis Hopper
Scott Glenn
Harrison Ford
Laurence Fishburne
Sam Bottoms
Will Smith

8. Which actors have appeared in comedy and action adventure movies?
    a. SQL: SELECT va.name FROM Video_Actors va, Video_Recordings vr WHERE vr.recording_id=va.recording_id AND vr.category='Action & Adventure' AND va.name IN (SELECT va.name FROM Video_Recordings vr, Video_Actors va WHERE vr.recording_id=va.recording_id AND vr.category='Comedy')
    b. Returned 4 rows

name
Rudy Ray Moore
Jerry Jones
Lady Reed
Harvey Fierstein