University of Windsor

**School of Computer Science**
**Faculty of Science**
**COMP-3150: Database Management System (Fall 2022)**

| Lab# | Date | Title | Due Date | Grade Release Date |
|---|---|---|---|---|
| Lab3 | Week 06 | **Database Schema Design (DDL)** | ~~One~~Two-Week Lab<br>~~October~~ Nov. ~~26~~02, 2022, Wednesday Midnight EDT | ~~October~~ Nov. ~~31~~07, 2022 |

The objective of this lab is to assist you in the process of converting your ERD from a conceptual design into an actual database design using *relations* (tables)[1] in *relational* databases using Data Definition Language (DDL).

## Step 1. Database Design

To implement the database design in a relational DBMS, you will need to create a database. In SQLite, this is done when you run SQLite as follows:

```
$ sqlite3 Moviesion.db
```

I picked `Moviesion.db` as the name of my database. You should replace it with a meaningful name with respect to your own project. Then, for each entity in ER diagram, a table should be created in the database. A table to represent an entity should have

1. A column for each attribute, with an appropriate datatype and with an additional decision as to whether the column can have null values or not.
2. A primary key and potentially candidate keys
3. An Id column (an integer usually) to act as a convenient primary key.

Each column should be set as being `NULL`, meaning that it is not required, or `NOT NULL`, indicating a value must be given. The primary key should be `NOT NULL`. For instance, in my project, for the `Director` entity, I will create a corresponding table, which would have a column called `Name`. Given all directors have a name; therefore, I can set this column to `NOT NULL`.

Candidate keys are alternative primary keys. For instance, in my `Director` relation, I can create a primary key of type integer that assigns every director entry a unique identifier called `Id`. I will designate this to be my primary key. However, the `Name` is also a unique attribute for each director therefore this makes `Name` a candidate key. In short, for the `Director` entity, I will perform the following steps:

1. I will create a table called `Director`
2. The director will have columns called `Id` and `Name`
3. The `Id` column will be of type `INT` and `NOT NULL`
4. The `Name` column will be of type `VARCHAR(255)` and `NOT NULL`
5. `Id` is the primary key, so there will be a `PRIMARY KEY` constraint on it
6. `Name` is a candidate key, so there will be a `UNIQUE` constraint on it

The `UNIQUE` constraint ensures that we cannot have to distinct directors with the same name. This leads to the following table definition:

```
CREATE TABLE Director(
    Id INTEGER PRIMARY KEY AUTOINCREMENT,
    Name VARCHAR(255) NOT NULL,
    CONSTRAINT UK_Name UNIQUE(Name)
);
```

[1] Table is the colloquially equal to relation in relational database management systems. Herein, we use them interchangeably.

There are ways to generate Id for each director in `Director` table. One way to do this is to just make them up, which is fine for a very small database maintained by hand, but for even moderately sized databases, or any sort of automation this becomes a problem. Most DBMSs offer some way to generate Ids. In fact, there is a functionality that would create incremental Ids automatically for you. This is called `AUTOINCREMENT`.

In SQLite, a column with type `INTEGER PRIMARY KEY` is an alias for the `ROWID`, which is always a 64-bit signed integer. On an `INSERT`, if the `ROWID` or `INTEGER PRIMARY KEY` column is not explicitly given a value, then it will be filled automatically with an unused integer, usually one more than the largest `ROWID` currently in use. This is true regardless of whether the `AUTOINCREMENT` keyword is used.

As seen, we use `PascalCase` when naming a table, column, or constraint. However, for keyword in relational query language, i.e., SQL, we use `ALLUPPERCASE`. Also, to name a `UNIQUE` constraint, `UK_` is followed by the name of candidate key, e.g., `UK_Name`.

To design a relation for the `Movie` entity, it follows much the same procedure as for `Director`, but a foreign key is needed since `Movie` is on the many side of a one-to-many relationship. Each movie is related to a director, so this information needs to be stored. To achieve this, we design a `Movie` table with a foreign key to the `Director` table.

Part of a table definition for the `Movie` entity is shown below. The foreign key represents the many-to-one relationship from movies to directors. Given a movie entry we can use `DirectorId` to look up the corresponding director.

```
CREATE TABLE Movie (
    Id INTEGER PRIMARY KEY AUTOINCREMENT,
    Title VARCHAR(255) NOT NULL,
    Genre VARCHAR(255) NOT NULL,
    DirectorId INT NOT NULL,
    ...
    {other attributes go in here}
    ...
    CONSTRAINT FK_Movie_DirectorId_2_Director_Id FOREIGN KEY(DirectorId)
    REFERENCES Director(Id)
);
```

As seen, we use `AUTOINCREMENT` to generate Ids for movies as well. Note that the foreign key naming convention follows the following pattern:

`FK_`*{SourceTableName_ColumnName}*`_2_`*{TargetTableName_ColumnName}*

For SQLite to enforce foreign keys, you must ensure it is enabled on your installation of SQLite. You can always check, enable, or disable foreign key enforcement as in the following series of commands:

```
sqlite> PRAGMA foreign_keys;
0
sqlite> PRAGMA foreign_keys = ON;
sqlite> PRAGMA foreign_keys;
1
sqlite> PRAGMA foreign_keys = OFF;
sqlite> PRAGMA foreign_keys;
0
```

If no Boolean value has been returned instead of a single row containing "0" or "1", then the version of SQLite you are using does not support foreign keys (maybe because it was compiled with `SQLITE_OMIT_FOREIGN_KEY` or `SQLITE_OMIT_TRIGGER` defined).

You will need to create all the tables and the appropriate relationships through tables for your ERD.

**Final Step. Deliverables**

You should complete the steps described above and submit the following items in one single `zip` file `Lab3_uwinid.zip`:

- (70%) `Lab3_uwinid.zip`
  - (20%) `DatabaseName.db` → The database file with tables. Instead of `DatabaseName`, use your own database name, e.g., mine is Movision, so, `Movision.db`.
  - (30%) `DatabaseName.sql` → the series of SQL commands (script) to create the database. Instead of `DatabaseName`, use your own database name.
  - (20%) `Report.pdf` → lab report including name, student id, and explanation on how tables relate to ERD.
  - (Optional) `ReadMe.txt` → optional additional information that helps lab instructor or grader to evaluate
- (30%) Naming (PascalCase) and Formats

Please follow the naming convention as you lose marks otherwise. Instead of `uwinid`, use your own account name, e.g., mine is hfani@uwindsor.ca, so, `Lab3_hfani.zip`