COMP1044 Coursework Assignment 2 Report A

Group 15 / Chew Language

Leong Chang Yung, 20307078
Lucas Dylan Purnell, 20197316 Tan Zhun Xian, 20313854
Chong Hao Wei, 20194465 Morhaf Allababidi, 20195867
Thomas Tan Kean Yew, 20316601

April 2, 2021

1 Database Design Diagram

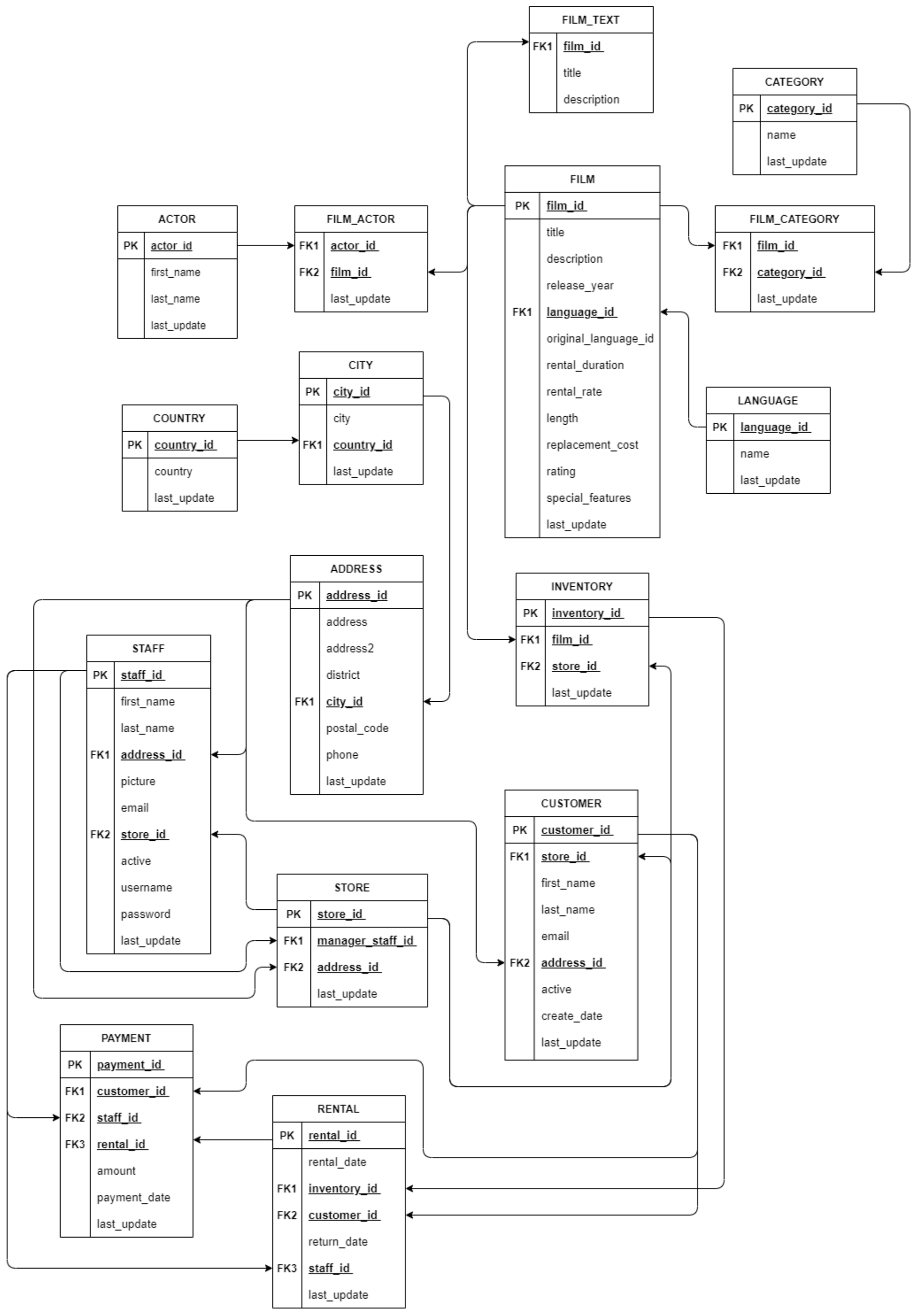

Figure 1: Proposed model for database.

2 Datatype Selection

For every field we chose the datatype depending on what would both match the data we had available and take up the lowest amount of memory possible while still ensuring that the type and length would be suitable for all new entries.

For most of the fields that were purely filled with numbers we set the type to either **int** or **decimal** (depending on whether the number was an integer or a decimal). A few examples of this are the *id* fields and the *length* and *rating* field in the *film* table. A notable exception to this was the *phone number* field. We set this datatype to **varchar** to allow people to enter their phone numbers in a different format despite all phone numbers in the dataset being integers. This was because many phone numbers need either a plus or a dash. Additionally, this field could be parsed and converted into an integer later if needed.

For all **varchar** we used the same principle but for the majority of fields we tried to keep the maximum lengths in neat values (normally multiples of 5). For instance, common values for the maximum length of **varchar** are 10, 25 and 50. We decided to set each field's maximum length to the longest realistic input value. For instance, we set *film_title* to **varchar(100)** as the longest name for a popular movie was 76 characters. We also set *first_name* and *last_name* to 50 as we thought that would be a good estimate based on actual estimates of the longest person names.

All date values were set to the type **datetime** according to their date and time content. For instance the *last_update* and *create_date* fields.

3 Reasoning for Primary and Foreign Key Choice

When we were deciding on what to pick for the primary key for each time we understood that it had to be both unique and non-null. This is because the primary key serves as the identifier for each record in the table and if it is either repeated or NULL it can no longer serve as an identifier. The obvious candidate here was the ID field in every table (for instance the *inventory_id* in the inventory table) as it fulfils both of these criteria.

The foreign key has the same requirements except that it need not be unique – only the primary key it references has to be unique. With the in mind we can use the same fields that we used as primary keys. Therefore for each of our arrays the *id* column is used to link the tables together. For instance, *customer_id* was chosen as the foreign key to link the *payment* table with the *customer* table and *address_id* was chosen as the foreign key to link the *staff* table with the *address* table.

---

4 Completeness of the Dataset

The database was relatively complete and we did not struggle with the quality of the data. We found that the data values were consistent across the different tables. This meant that we didn't run into any issues regarding the foriegn keys and we didn't find conflicting information in the data set.

We found a few different ways that the dataset was incomplete. The first was missing numbers from the auto-incremental data. For instance, there were missing *address_id*'s in *Address.csv*. This problem was easily remedied by enabling the property to **auto increment** during the creation of said file.

The next problem was that many NULL fields were invalid and are supposed to contain a value. For instance, some of the fields of *Rental_id* in *Payment.csv* were NULL when this should not have been the case. This is a problem as it may compromise the integrity of the database, as these NULL fields will lead to orphaned records due to the lack of a foriegn key.

There was also a NULL value in the *password* column of the *staff.csv* table. This would indicate data corruption, data loss, or staff negligence in inputting a proper password. In the interest of system security, the best remedy would be to get the relevant staff member to provide a value for this NULL field. However, while the problem stands, we decided that it would be better to leave these fields as NULL to avoid risking data integrity.

---

5 Errors in the Database

We also noticed that in *country.csv* and *customer.csv* that there wasn't a consistent delimiter. The file alternated between using commas and semicolons to separate data. The lack of support for semicolon delimiters made it impossible to import the data into the database while keeping it in its intended format. To remedy this we replaced the semicolons with commas wherever applicable using text editors.

While studying the dataset, we discovered that some of the entries have been duplicated, such as the occurrence of two film descriptions in *film_text.csv* and *film.csv*. We can resolve this redundancy by removing a duplicate description field on one of the files.

Another error was the existence of a corrupted picture file within *staff.csv*. This corruption made it impossible to view or display the aforementioned picture. We decided that leaving a corrupted picture within the staff database was not of high importance and wouldn't decrease the completeness or understandability of the dataset, so we just removed the picture. A partial solution was found in making the field type storing the picture as **longblob**.

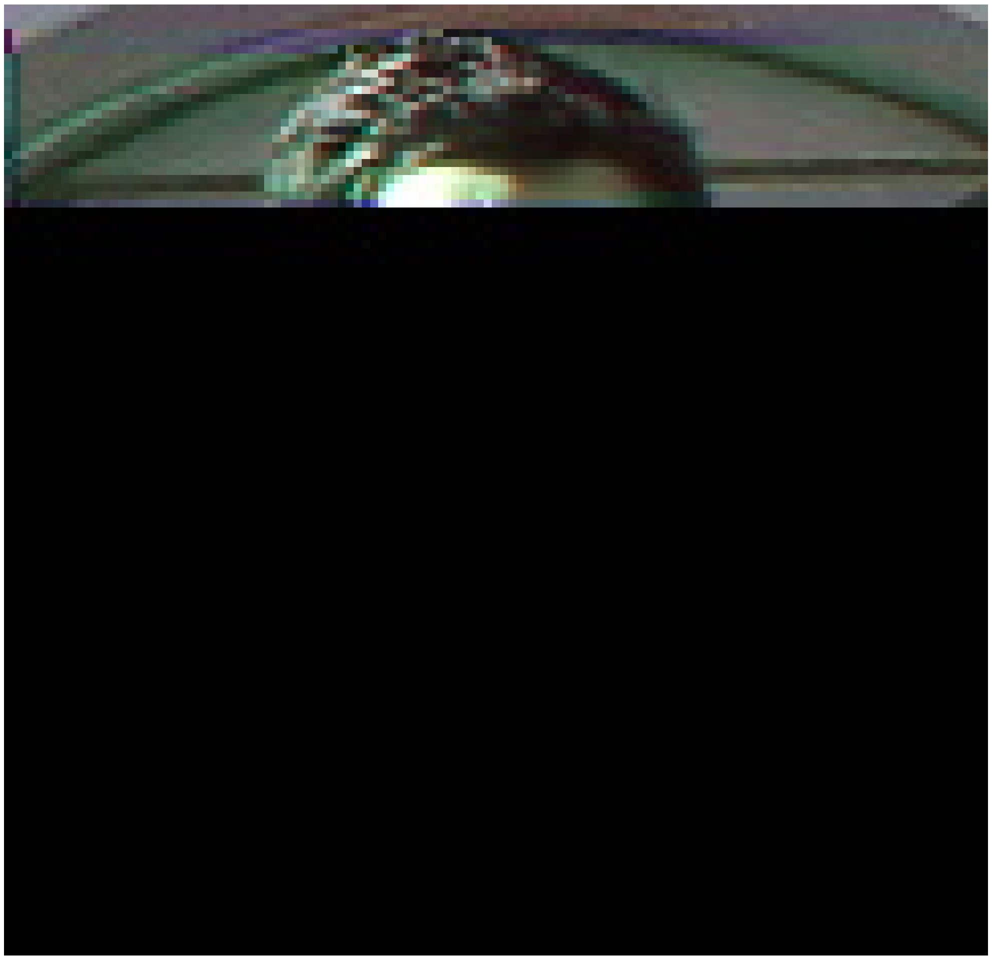

Figure 2: The partially recovered staff photo.

This allowed the picture to be imported, albeit still in an incomplete state.

6 Conclusion

At the point of writing, the team determined the database to be sufficient for further usage, due to SELECT queries functioning properly when used to select specific rows in the tables.