COMP 2350

Lab Report:

Foreign Keys and Inheritance

By Yeseol (Sol) Kim

Submitted to: Patrick Guichon

Introduction:

Today, I conducted a study on 1NF, 2NF, 3NF, as well as inheritance and advanced foreign key setups. I was able to simplify complex table structures by connecting the classes of each table, dividing them into separate tables, and inheriting properties from one another. This involved establishing connections using foreign keys and primary keys when inheriting characteristics. The goal was to organize vast amounts of data in a database efficiently.

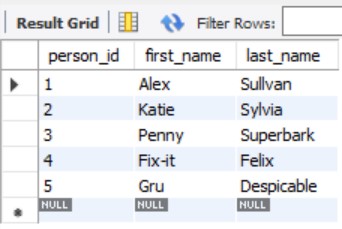
I streamlined a hockey game information-filled Excel sheet by breaking it down into individual tables, aligning each with its appropriate attributes. Throughout this process, careful consideration of the relationships between attributes was crucial in designing the data. Although the data for the lab was a somewhat simplified version, I utilized MySQL Workbench to designate foreign keys. The system prevented data entry when foreign keys were missing, ensuring data integrity.

After completing all the steps, I connected the local SQL database with Render. This allowed for seamless updates; any modifications or updates made in MySQL Workbench were immediately reflected in Render. This practice further reinforced the connection between local SQL and Render, facilitating a smooth workflow for database management and updates.

Screenshot #1:

I created tables using the first name and last name of individuals. Each person is assigned a primary key value called **person\_id**.

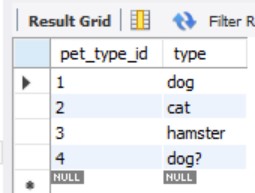
Filename: 01\_Person\_Table.jpg



Screenshot #2:

I created tables for each pet based on the types of individuals. Each pet is assigned a primary key value called **pet\_type\_id**.

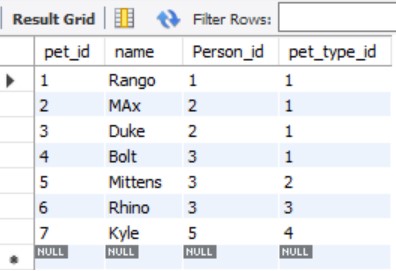
Filename: 02\_Pet\_Type\_Table.jpg



Screenshot #3:

I created tables for each pet based on the name of individuals. Each pet is assigned a primary key value called **pet\_id**. The "pet" table references the "person" table by using the foreign key column "person\_id," and it also references the "pet\_type" table using the foreign key column "pet\_type\_id."

Filename: 03\_Pet\_Table.jpg

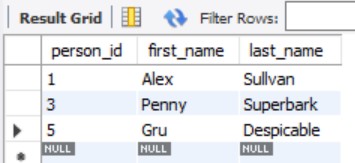


Screenshot #4, #5, #6:

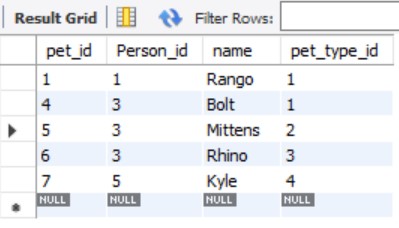
If you delete data linked through foreign keys in the "person" table, the related data in the "pet" table may also be deleted. Since the two tables are connected through foreign keys, the data in the child table (in this case, "pet") that references the primary key of the parent table ("person") is handled together to maintain referential integrity.

For instance, if you delete a record for a specific person in the "person" table, the related records for that person in the "pet" table might be automatically deleted as well. This ensures that the database maintains relationships while preserving data consistency. File04 to 06 show the result.

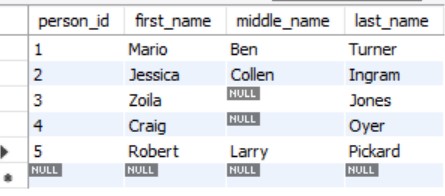
Filename: 04\_Person\_Table\_no\_Katie.jpg



Filename: 05\_Pet\_Table\_no\_Katie.jpg



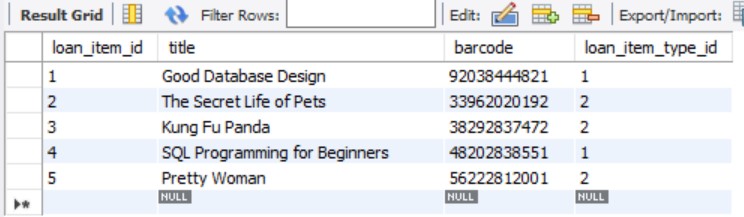
Filename: 06\_Pet\_Table\_no\_Penny.jpg



Screenshot #7:

The "loan\_item" table displays available items for loan. The "item\_type\_id" column serves as a foreign key referencing the "item\_type" table.

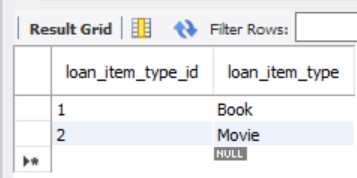
Filename: 07\_Loan\_Item\_Table.jpg



Screenshot #8:

The "item\_type" table consists of two categories: "book" and "movie," each identified by a unique ID key. These IDs can be referenced in other tables to retrieve information about items categorized as books or movies.

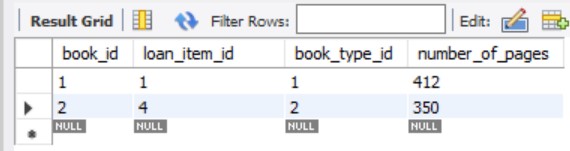
Filename: 08\_Loan\_Item\_Type\_Table.jpg



Screenshot #9:

This is the "book" table where I connected the available loan\_item\_id and book\_type\_id as foreign keys.

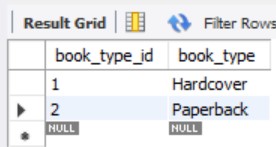
Filename: 09\_Book\_Table.jpg



Screenshot #10:

This table distinguishes between different types of books, categorizing them into hardcover and paperback. I have assigned unique ID values for each of these types.

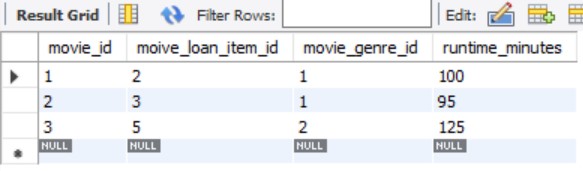
Filename: 10\_Book\_Type\_Table.jpg



Screenshot #11:

This is the "movie" table. If you want to use one foreign key in multiple tables and prefer to change the key name for clarity, it is acceptable. For example, in this table, I used "book\_loan\_item\_id" instead of "loan\_item\_id" as the foreign key, which can make the data more readable and understandable, especially in the context of a movie table referring to loan items associated with books.

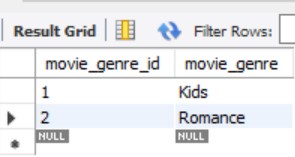
Filename: 11\_Movie\_Table.jpg



Screenshot #12:

The movie genre table follows the same structure as the book type table mentioned earlier.

Filename: 12\_Movie\_Genre\_Table.jpg



Screenshot #13:

First, push your code to GitHub and establish a connection with Render. Obtain the host and database values from freeDB and update your code accordingly. After making changes, push to GitHub, and Render will automatically deploy the updated code.

Create a "todo" table in the MySQL database connected to Render. Update values in this table, and the Render server will automatically read and update the values. This seamless integration allows for smooth development and deployment of your application.

Filename: 13\_SELECT\_from\_a\_Database.jpg

