

# 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

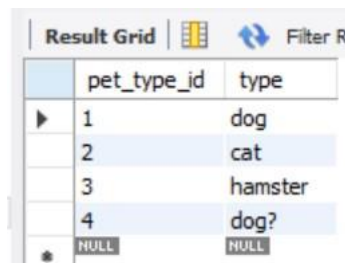


	person_id	first_name	last_name
▶	1	Alex	Sullivan
	2	Katie	Sylvia
	3	Penny	Superbark
	4	Fix-it	Felix
	5	Gru	Despicable
*	NULL	NULL	NULL

### 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



	pet_type_id	type
▶	1	dog
	2	cat
	3	hamster
	4	dog?
*	NULL	NULL

### 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



Result Grid				
Filter Rows: <input type="text"/>				
	pet_id	name	Person_id	pet_type_id
▶	1	Rango	1	1
	2	MAx	2	1
	3	Duke	2	1
	4	Bolt	3	1
	5	Mittens	3	2
	6	Rhino	3	3
	7	Kyle	5	4
✱	NULL	NULL	NULL	NULL

#### 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



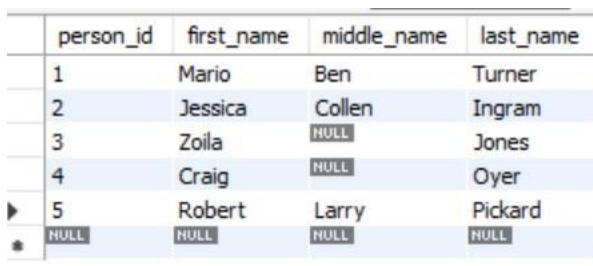
	person_id	first_name	last_name
	1	Alex	Sullivan
	3	Penny	Superbark
▶	5	Gru	Despicable
*	NULL	NULL	NULL

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



	pet_id	Person_id	name	pet_type_id
	1	1	Rango	1
	4	3	Bolt	1
▶	5	3	Mittens	2
	6	3	Rhino	3
	7	5	Kyle	4
*	NULL	NULL	NULL	NULL

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



	person_id	first_name	middle_name	last_name
	1	Mario	Ben	Turner
	2	Jessica	Collen	Ingram
	3	Zoila	NULL	Jones
	4	Craig	NULL	Oyer
▶	5	Robert	Larry	Pickard
*	NULL	NULL	NULL	NULL

### 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

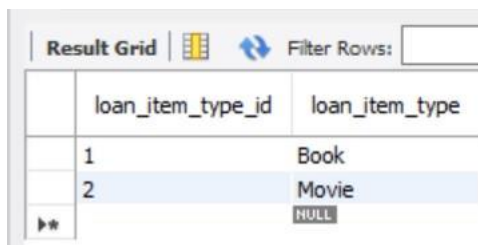


	loan_item_id	title	barcode	loan_item_type_id
	1	Good Database Design	92038444821	1
	2	The Secret Life of Pets	33962020192	2
	3	Kung Fu Panda	38292837472	2
	4	SQL Programming for Beginners	48202838551	1
	5	Pretty Woman	56222812001	2
▶*		NULL	NULL	NULL

### 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

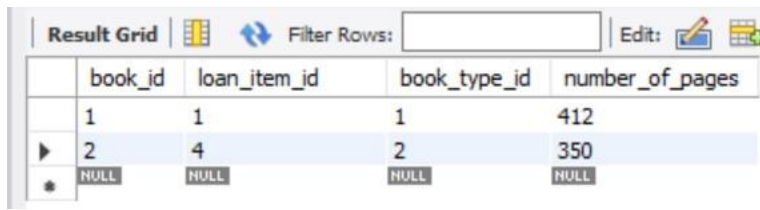


	loan_item_type_id	loan_item_type
	1	Book
	2	Movie
▶*		NULL

### 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



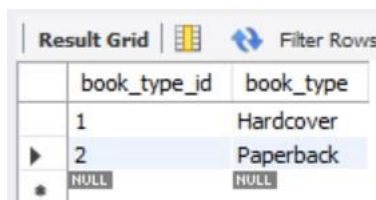
The screenshot shows a database interface with a 'Result Grid' tab. The grid displays data for the 'book' table. The columns are 'book\_id', 'loan\_item\_id', 'book\_type\_id', and 'number\_of\_pages'. There are three rows: the first row has values 1, 1, 1, and 412; the second row has values 2, 4, 2, and 350; the third row has NULL values for all four columns. The interface includes a 'Filter Rows' field and an 'Edit' button.

	book_id	loan_item_id	book_type_id	number_of_pages
	1	1	1	412
▶	2	4	2	350
*	NULL	NULL	NULL	NULL

### 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



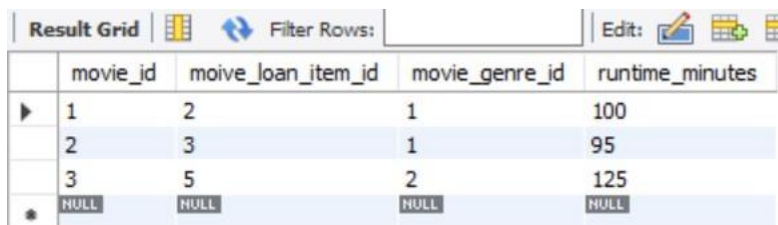
The screenshot shows a database interface with a 'Result Grid' tab. The grid displays data for the 'book\_type' table. The columns are 'book\_type\_id' and 'book\_type'. There are three rows: the first row has values 1 and Hardcover; the second row has values 2 and Paperback; the third row has NULL values for both columns. The interface includes a 'Filter Rows' field.

	book_type_id	book_type
	1	Hardcover
▶	2	Paperback
*	NULL	NULL

### 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



	movie_id	moive_loan_item_id	movie_genre_id	runtime_minutes
▶	1	2	1	100
	2	3	1	95
	3	5	2	125
*	NULL	NULL	NULL	NULL

### Screenshot #12:

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

Filename: 12\_Movie\_Genre\_Table.jpg



	movie_genre_id	movie_genre
	1	Kids
▶	2	Romance
*	NULL	NULL



### 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

