COMP 2350

Lab Report:

More Database Lab Report

By Yeseol (Sol) Kim

Submitted to: Patrick Guichon

Introduction:

In today's lab engaged in a comprehensive exploration consisting of four distinct exercises. Initially, our focus was on reviewing and actively practicing the creation of many-to-many relationship tables, with particular emphasis on the essential role played by a middle table when establishing connections between two tables. This exercise also facilitated a deeper understanding of the concept of unary relationships, contributing to a more comprehension of database design.

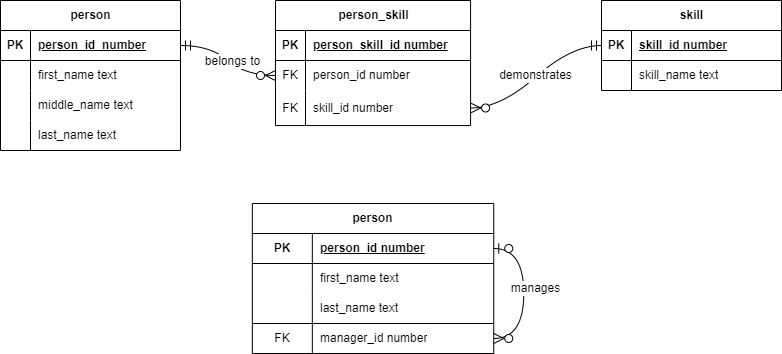
Moving forward, the second and third exercises delved into advanced topics within the realm of MySQL. Participants actively practiced working with many-to-many relationship tables, primary keys, and foreign keys, gaining practical insights into structuring complex database relationships. Furthermore, our exploration of MySQL extended to the fine-tuning of database structures by setting ranges for column values, a skill instrumental in maintaining data integrity. The implementation of unique keys was also covered, showcasing how these constraints contribute to preventing repetitive data entries and ensuring the uniqueness of key values.

The culmination of our lab activities involved a practical application of coding skills to establish a seamless connection between a Render app and a local host. Leveraging the FreeDB service, participants configured crucial parameters such as the host, database name, and password. The integration with GitHub streamlined the deployment process, enabling an efficient and reliable connection between the Render app and the local host. This multifaceted exploration not only reinforced foundational database concepts but also provided practical exposure to advanced MySQL functionalities and the integration of databases with web applications.

Screenshot #1:

First drawing show many to many relationships. One person may or may not get multiple skills also One skill may or may not is gotten multiple people. Many to many cases need middle table that connect two different tables. Second drawing is self-referential (unary) relationship follows all of the same rules as the relationships created between two tables. They can be one-to-one, one-to-many or many-to-many. The same unique constraints still apply and will dictate the type of relationship made.

Filename: 01\_ERD\_with\_Many\_to\_Many\_Relationship.png



Screenshot #2:

Filename: 02\_ERD\_with\_Many\_to\_Many\_Relationship .vsdx



Screenshot #3:

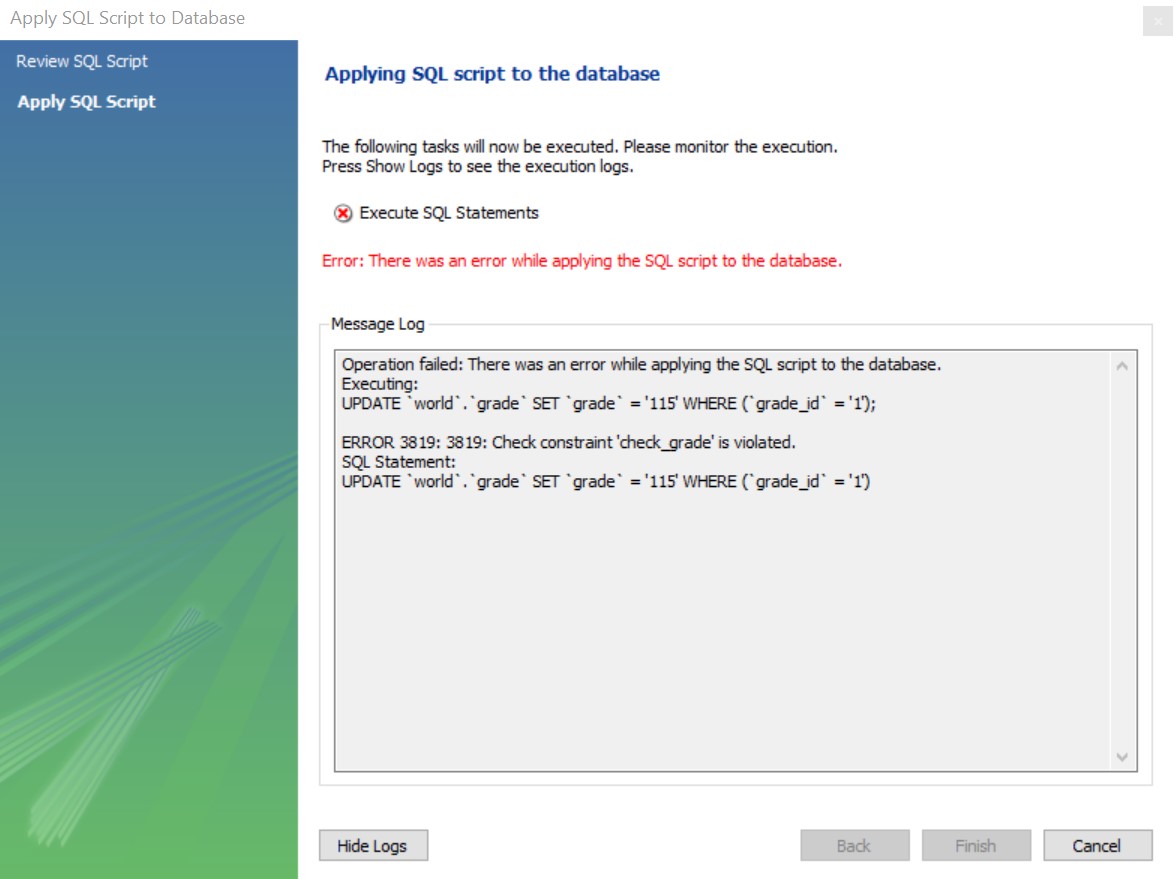
In MySQL, we can make table and set up range of each column. For example, if you want to make in 0 to 100 grades.

ALTER TABLE grade //set up with this table

ADD CONSTRAINT check grade CHECK (grade >= 0 AND grade <= 100); //grade should between 0 to 100.

Then you can’t add data over 100.

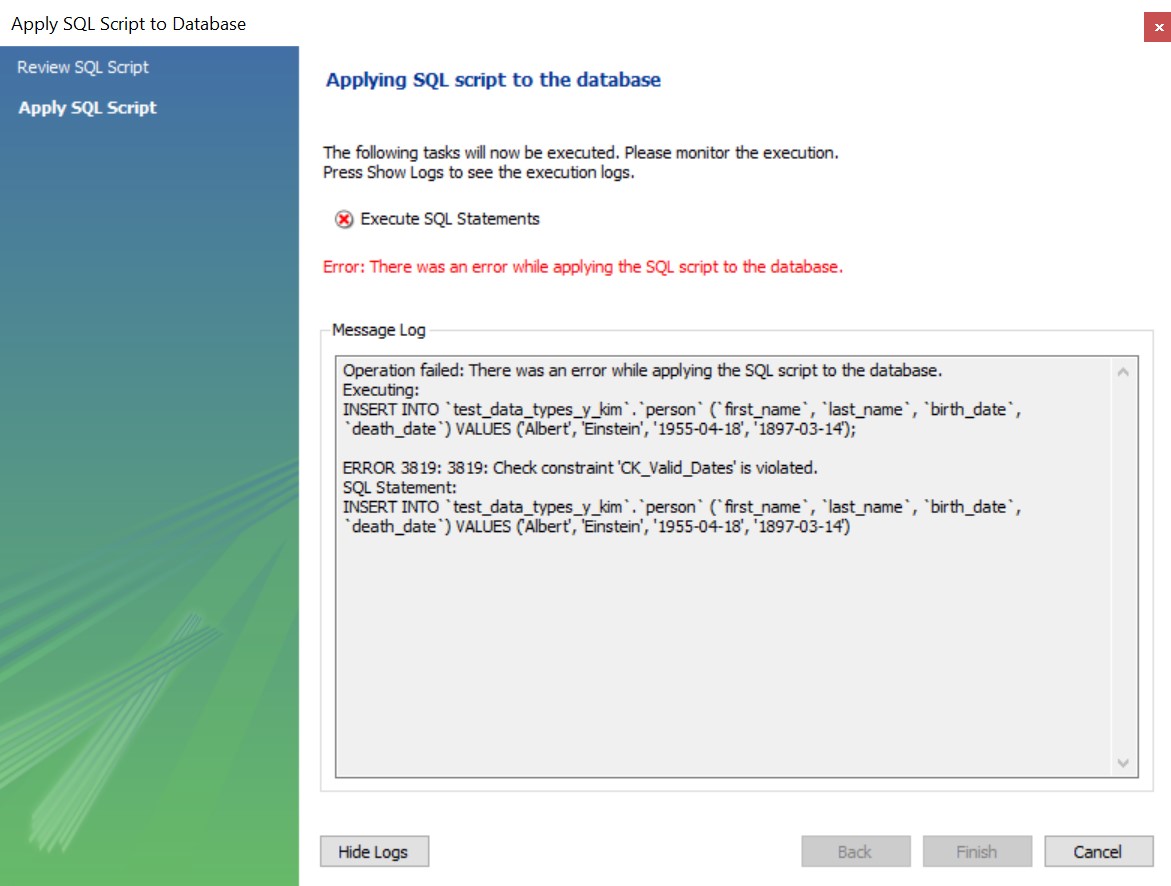
Filename: 03\_grade\_constraint\_failed.jpg



Screenshot #4:

Also connected with screenshot #3, It can set up time line too. Death date is supposed to after birth date. The command is below: ADD CONSTRAINT CK\_Valid\_Dates CHECK (birth\_date <= death\_date);

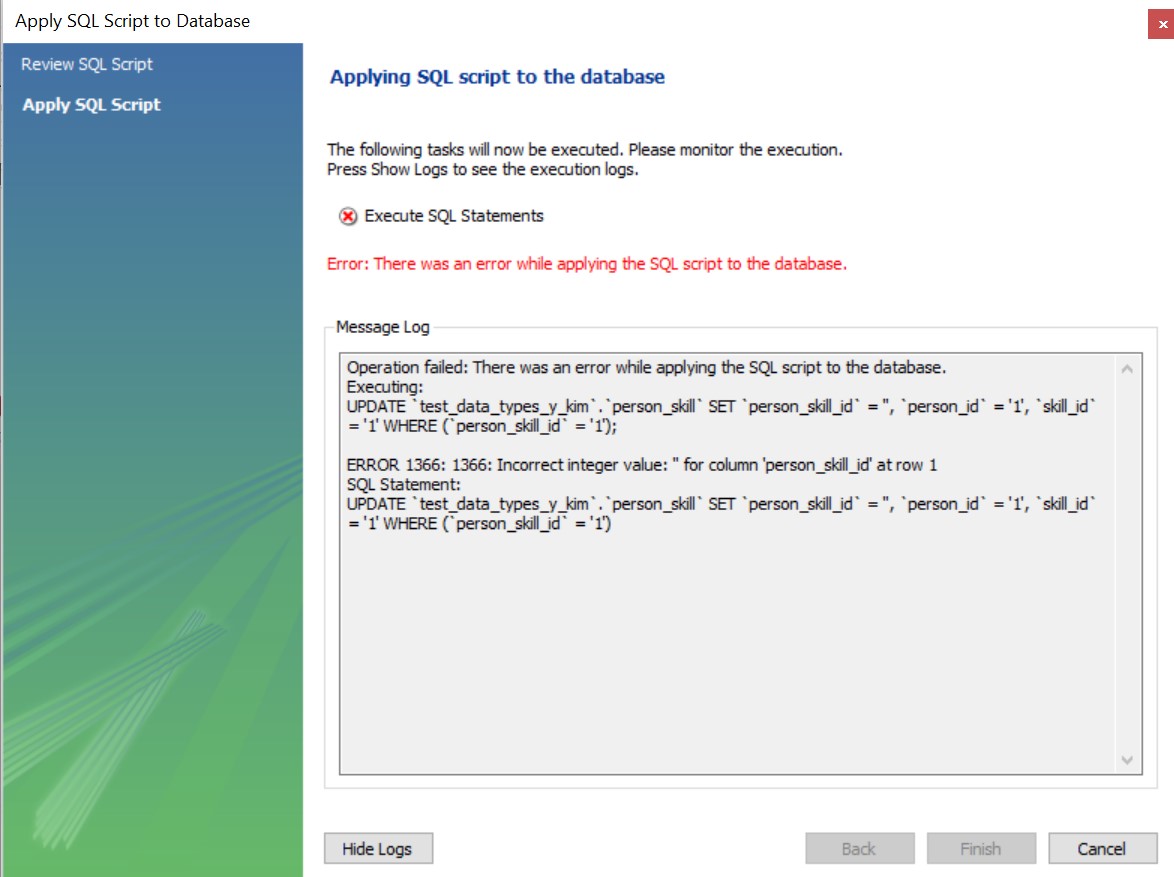
Filename: 04\_invalid\_dates.jpg



Screenshot #5:

I made another work branch at MySQL and made three tables “**person**, **skill** and **person\_skill**”. Person and skill table are many to many relationships so they need middle table to connect together. When there is no data at person and skill tables, you can’t put any data in **person\_skill** as well.

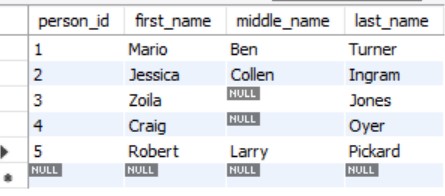
Filename: 05\_data\_in\_wrong\_order.jpg



Screenshot #6:

This table represents individual persons with their full names stored in the columns first\_name, middle\_name, and last\_name, all of which are of text data type. The person\_id column, serving as the primary key, is of numerical data type, uniquely identifying each person in the table.

Filename: 06\_person\_data.jpg



Screenshot #7:

The skill table captures various skills, ensuring uniqueness in each skill's name, denoted by the skill\_name column. Each skill is uniquely identified by its corresponding skill\_id, providing a distinct numerical identifier for each skill in the table.

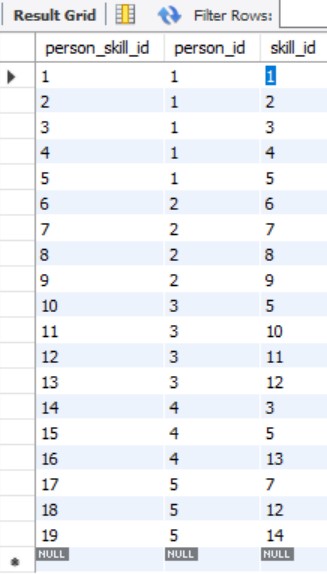
Filename: 07\_skill\_data.jpg



Screenshot #8:

Once the person and skill tables are populated, the **person\_skill** table is established to capture relationships between individuals and their associated skills. The **person\_id** and **skill\_id** columns serve as both unique keys and foreign keys, ensuring referential integrity and facilitating the mapping of skills to specific individuals within the database.

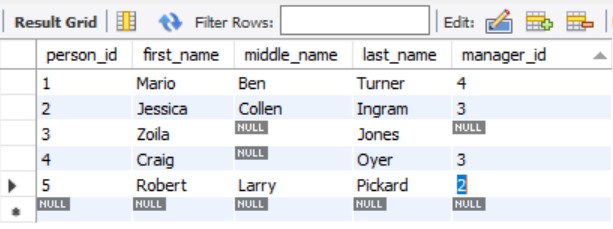
Filename: 08\_person\_skill\_data.jpg



Screenshot #9:

The manager\_id column has been added to the person table, utilizing the person\_id as a foreign key. This enhancement enables the representation of managerial relationships, allowing individuals to be associated with their respective managers within the same person table. The manager\_id column acts as a reference to another person\_id within the same table, establishing hierarchical connections among individuals.

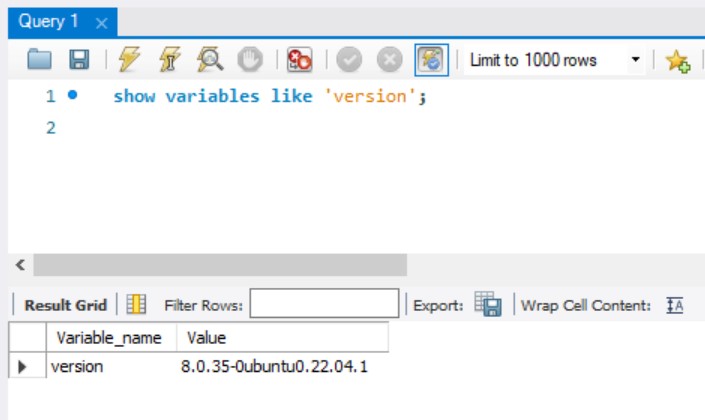
Filename: 09\_person\_manager\_data.jpg



Screenshot #10:

To configure the freeDB server and establish a connection with MySQL, provide the host and username details, and enter the password. After connecting, use the command SHOW VARIABLES LIKE 'version' to retrieve important information for compatibility and understanding the server environment. This command will display the MySQL version details, aiding in further analysis and compatibility checks.

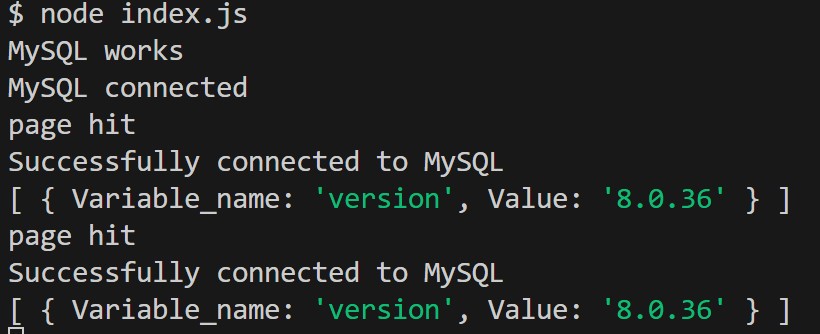
Filename: 10\_DB\_MySQL\_Version.jpg



Screenshot #11:

Creating a Render app with both local and remote database connections was a challenging aspect of the lab. In the JavaScript file, I introduced constants for local and remote hosts, along with user, password, and database name parameters. This configuration was then pushed to a GitHub repository, establishing a connection with the Render app. To ensure successful local host connection, I implemented console statements, including feedback on version and version number display in the terminal.

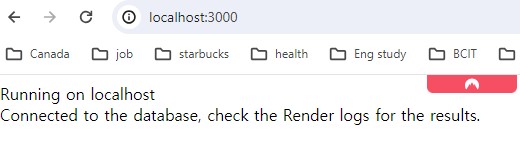
Filename: 11\_Local\_Version.jpg



Screenshot #12:

Additionally, upon accessing localhost:3000, a connection message is displayed, providing further confirmation of the successful establishment of the database connection.

Filename: 12\_Local\_Site.jpg



Screenshot #13:

Following the final git push, the Render app undergoes automatic deployment. This seamless process enables verification of the same successful database connection by checking the results on localhost.

Filename: 13\_Render\_Site.jpg

