Information Management (MySQL) [1]

A Study Guide for Students of Sorsogon State University - Bulan Campus[2]

Jarrian Vince G. Gojar[3]

March 6, 2025

[1]A course in the Bachelor of Science in Computer Science.

[2]This book is a study guide for students of Sorsogon State University - Bulan Campus taking up the course Information Management.

[3]https://github.com/godkingjay

Sorsogon State University - Bulan Campus

Contents

List of Figures

List of Tables

List of Codes

Preface

"You can have data without information, but you cannot have information without data."

– Daniel Keys Moran

Jarrian Vince G. Gojar
https://github.com/godkingjay

1

Introduction to MySQL

1.1 What is MySQL?

MySQL is an open-source relational database management system (RDBMS). An RDBMS is a type of database management system (DBMS) that stores data in a structured format, using rows and columns. It is a software that is used to manage the creation, modification, and maintenance of a database. MySQL is a popular choice for database management in web applications and is used by many high-profile websites, including Facebook, Twitter, and YouTube.

1.2 MySQL Elements

1.2.1 Tables

A **table** is a collection of related data stored in rows and columns. Each row in a table represents a record, and each column represents a field. Tables are the basic building blocks of a database and are used to store and organize data in a structured format.

id	**name**	**age**
1	Alice	25
2	Bob	30
3	Charlie	35

Table 1.1: Example of a table

Table 1.1 shows an example of a table with three columns: **id**, **name**, and **age**. Each row in the table represents a record with a unique **id** value, and each column represents a field in the record.

1.2.2 Columns

A **column** is a vertical arrangement of data in a table. Each column represents a field in the table and contains data of a specific type. Columns are used to store different types of data, such as numbers, text, dates, and more.

In the example table above, the columns are **id**, **name**, and **age**. The **id** column stores unique identifiers for each record, the **name** column stores the names of the individuals, and the **age** column stores the ages of the individuals.

1.2.3 Rows

A **row** is a horizontal arrangement of data in a table. Each row represents a record in the table and contains data for each column in the table. Rows are used to store individual records in the table.

In the example table above, each row represents a record with data for the **id**, **name**, and **age** columns. The first row contains data for Alice, the second row contains data for Bob, and the third row contains data for Charlie.

1.2.4 Data Types

A **data type** is a classification of data based on the type of values it can hold. MySQL supports a wide range of data types, including numeric, string, date, and time data types. Data types are used to specify the type of data that can be stored in a column in a table.

1.2.4.1 Numeric Data Types

Numeric data types are used to store numeric values, such as integers, decimals, and floating-point numbers. MySQL supports a variety of numeric data types, including **INT**, **DECIMAL**, **FLOAT**, **DOUBLE**, **TINYINT**, **SMALLINT**, **MEDIUMINT**, and **BIGINT**.

1.2.4.2 Character and String Data Types

Character and string data types are used to store text values, such as names, addresses, and descriptions. MySQL supports a variety of character and string data types, including **CHAR**, **VARCHAR**, and **TEXT**.

1.2.4.3 Date and Time Data Types

Date and time data types are used to store date and time values, such as birthdates, appointment times, and event dates. MySQL supports a variety of date and time data types, including **DATE**, **TIME**, **DATETIME**, and **TIMESTAMP**.

1.2.4.4 Binary Data Types

Binary data types are used to store binary data, such as images, audio files, and video files. MySQL supports a variety of binary data types, including **BLOB**, **MEDIUMBLOB**, and **LONGBLOB**.

1.2.4.5 Boolean Data Type

The **BOOLEAN** data type is used to store boolean values, such as true or false. In MySQL, boolean values are represented as 1 for true and 0 for false.

1.2.5 NULL Values

A **NULL** value is a special value that represents the absence of a value. NULL values are used to indicate that a column does not contain any data. In MySQL, columns can be defined to allow NULL values, which means that the column can contain NULL values in addition to other values.

1.2.6 Comments

Comments are used to add explanatory notes to the SQL code. Comments are ignored by the MySQL server and are used to document the code for reference. Comments can be added to SQL code using the – or `/* */` syntax.

```sql
-- This is a single-line comment

/*
  This is a multi-line comment
  that spans multiple lines
*/

SELECT * FROM users; -- This is a comment at the end of a line
```

Code 1.1: Example of a comment in SQL code

Code 1.1 shows an example of comments in SQL code. Comments can be added to SQL code using the – syntax for single-line comments and the `/* */` syntax for multi-line comments.

2

Managing Databases

2.1 Creating a Database

To create a new database in MySQL, you can use the **CREATE DATABASE** statement followed by the name of the database. The following example creates a new database named **database_name**:

```sql
CREATE DATABASE database_name;
```

Code 2.1: Creating a new database in MySQL

Code 2.1 shows an example of creating a new database in MySQL using the **CREATE DATABASE** statement. The statement creates a new database named **database_name**. Once the database is created, you can use it to store tables and data.

2.2 Showing Databases

To display a list of all databases in MySQL, you can use the **SHOW DATABASES** statement. The following example shows a list of all databases in MySQL:

```sql
SHOW DATABASES;
```

Code 2.2: Showing all databases in MySQL

Code 2.2 shows an example of displaying a list of all databases in MySQL using the **SHOW DATABASES** statement. The statement lists all databases that are currently available in the MySQL server.

2.3 Selecting a Database

To select a database in MySQL, you can use the **USE** statement followed by the name of the database. The following example selects the **database_name** database:

```sql
USE database_name;
```

Code 2.3: Selecting a database in MySQL

Code 2.3 shows an example of selecting a database in MySQL using the **USE** statement. The statement selects the **database_name** database, which allows you to perform operations on the tables and data in that database.

2.4 Removing a Database

To remove a database in MySQL, you can use the **DROP DATABASE** statement followed by the name of the database. The following example removes the **database_name** database:

```sql
DROP DATABASE database_name;
```

Code 2.4: Removing a database in MySQL

Code 2.4 shows an example of removing a database in MySQL using the **DROP DATABASE** statement. The statement deletes the **database_name** database and all tables and data stored in that database.

Exercises

```
1. Create a new database named "employees".
2. Display a list of all databases in MySQL.
3. Select the "employees" database.
4. Remove the "employees" database.
```

3

Managing Tables

3.1 Creating a Table

To create a new table in MySQL, you can use the **CREATE TABLE** statement followed by the name of the table and a list of columns. The following example creates a new table named **table_name** with three columns: **id**, **name**, and **age**:

```sql
CREATE TABLE table_name (
  id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(50),
  age INT
);
```

Code 3.1: Creating a new table in MySQL

Code 3.1 shows an example of creating a new table in MySQL using the **CREATE TABLE** statement. The statement creates a new table named **table_name** with three columns: **id**, **name**, and **age**. Each column is defined with a data type (INT or VARCHAR) and a maximum length (50 for VARCHAR).

3.2 Showing Tables

To display a list of all tables in a database in MySQL, you can use the **SHOW TABLES** statement. The following example shows a list of all tables in the current database:

```sql
SHOW TABLES;
```

Code 3.2: Showing all tables in MySQL

Code 3.2 shows an example of displaying a list of all tables in the current database in MySQL using the **SHOW TABLES** statement. The statement lists all tables that are currently available in the database.

Exercises

```
1. Display a list of all tables in the current database.
```

3.3 Describe Table

To display the structure of a table in MySQL, you can use the **DESCRIBE** statement followed by the name of the table. The following example shows the structure of the **table_name** table:

```sql
DESCRIBE table_name;
```

Code 3.3: Describing a table in MySQL

Code 3.3 shows an example of displaying the structure of a table in MySQL using the **DESCRIBE** statement. The statement shows the columns, data types, and other properties of the **table_name** table.

3.4 Renaming a Table

To rename a table in MySQL, you can use the **RENAME TABLE** statement followed by the current name of the table and the new name of the table. The following example renames the **old_table** table to **new_table**:

```sql
RENAME TABLE old_table TO new_table;
```

Code 3.4: Renaming a table in MySQL

Code 3.4 shows an example of renaming a table in MySQL using the **RENAME TABLE** statement. The statement renames the **old_table** table to **new_table**.

3.5 Dropping a Table

To drop a table in MySQL, you can use the **DROP TABLE** statement followed by the name of the table. The following example drops the **table_name** table:

```sql
DROP TABLE table_name;
```

Code 3.5: Dropping a table in MySQL

Code 3.5 shows an example of dropping a table in MySQL using the **DROP TABLE** statement. The statement deletes the **table_name** table and all data stored in that table.

Exercises

```
1. Create a database named "library".
2. Create a table named "books" with columns:
   - id (INT, PRIMARY KEY, AUTO_INCREMENT)
```

```
    - title (VARCHAR, 100)
    - author (VARCHAR, 50)
    - year (INT)
3. Create a table named "users" with columns:
    - id (INT, PRIMARY KEY, AUTO_INCREMENT)
    - name (VARCHAR, 50)
    - email (VARCHAR, 100)
    - is_staff (BOOLEAN)
4. Display the structure of the "books" table.
5. Rename the "users" table to "members".
6. Drop the "members" table.
```

3.6 Temporary Tables

Temporary tables are a special type of table that is created and destroyed automatically by the MySQL server. Temporary tables are useful for storing temporary data that is only needed for the duration of a session or a query. Temporary tables are created using the **CREATE TEMPORARY TABLE** statement and are automatically dropped when the session ends.

```sql
CREATE TEMPORARY TABLE temp_table (
  id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(50)
);
```

Code 3.6: Creating a temporary table in MySQL

Code 3.6 shows an example of creating a temporary table in MySQL using the **CREATE TEMPORARY TABLE** statement. The statement creates a temporary table named **temp_table** with two columns: **id** and **name**. The temporary table is automatically dropped when the session ends.

Exercises

```
1. Create a temporary table named "temp_data" with columns:
    - id (INT, PRIMARY KEY, AUTO_INCREMENT)
    - value (VARCHAR, 50)
2. Display a list of all tables in the current database.
```

3.7 Altering Tables (Adding Columns and Modifying Columns)

To alter a table in MySQL, you can use the **ALTER TABLE** statement followed by the name of the table and the type of alteration to perform. The following example adds a new column named **email** to the **table_name** table:

```sql
ALTER TABLE table_name ADD COLUMN email VARCHAR(100);
```

Code 3.7: Altering a table in MySQL

Code 3.7 shows an example of altering a table in MySQL using the **ALTER TABLE** statement. The statement adds a new column named **email** with a data type of VARCHAR(100) to the **table_name** table.

If multiple alterations need to be performed on a table, you can use the **ALTER TABLE** statement multiple times to apply each alteration separately.

```
ALTER TABLE table_name
  ADD COLUMN email VARCHAR(100),
  ADD COLUMN phone VARCHAR(20),
  MODIFY COLUMN age INT;
```

Code 3.8: Altering a table in MySQL (multiple alterations)

Code 3.8 shows an example of altering a table in MySQL with multiple alterations using the **ALTER TABLE** statement. The statement adds two new columns named **email** and **phone** and modifies the data type of the **age** column in the **table_name** table.

Exercises

```
  1. In the books table, add new columns named "genre" (VARCHAR, 50), "isbn"
     (VARCHAR, 20), and "price" (DECIMAL).
  2. In the books table, modify the data type of the "year" column to
     SMALLINT.
```

3.7.1 Dropping Columns

To drop a column from a table in MySQL, you can use the **ALTER TABLE** statement followed by the name of the table and the column to drop. The following example drops the **email** column from the **table_name** table:

```
ALTER TABLE table_name DROP COLUMN email;
```

Code 3.9: Dropping a column from a table in MySQL

Code 3.9 shows an example of dropping a column from a table in MySQL using the **ALTER TABLE** statement. The statement drops the **email** column from the **table_name** table.

Exercises

```
1. In the books table, drop the "price" column.
```

3.7.2 Renaming Columns

To rename a column in a table in MySQL, you can use the **ALTER TABLE** statement followed by the name of the table and the new name for the column. The following example renames the **old_name** column to **new_name** in the **table_name** table:

```sql
ALTER TABLE table_name CHANGE COLUMN old_name new_name VARCHAR(50);
```

Code 3.10: Renaming a column in a table in MySQL

Code 3.10 shows an example of renaming a column in a table in MySQL using the **ALTER TABLE** statement. The statement renames the **old_name** column to **new_name** with a data type of VARCHAR(50) in the **table_name** table.

Exercises

```
1. In the books table, rename the "isbn" column to "isbn_number" (VARCHAR,
   20).
```

3.8 Adding Constraints

Constraints are rules that are applied to columns in a table to enforce data integrity and consistency. Constraints can be used to specify conditions that must be met for data to be inserted or updated in a table. MySQL supports a variety of constraints, including **PRIMARY KEY**, **UNIQUE**, **NOT NULL**, and **FOREIGN KEY**.

3.8.1 Primary Key Constraint

The **PRIMARY KEY** constraint is used to uniquely identify each record in a table. A primary key column cannot contain NULL values and must have a unique value for each record. To add a primary key constraint to a column in MySQL, you can use the **PRIMARY KEY** keyword in the column definition.

```sql
CREATE TABLE table_name (
  id INT PRIMARY KEY,
  name VARCHAR(50)
);
```

Code 3.11: Adding a primary key constraint to a column in MySQL

Code 3.11 shows an example of adding a primary key constraint to a column in MySQL using the **PRIMARY KEY** keyword. The statement creates a new table named **table_name** with a primary key column named **id**.

3.8.2 Unique Constraint

The **UNIQUE** constraint is used to ensure that all values in a column are unique. A unique constraint allows NULL values, but only one NULL value is allowed. To add a unique constraint to a column in MySQL, you can use the **UNIQUE** keyword in the column definition.

```sql
CREATE TABLE table_name (
  id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(50),
  email VARCHAR(100) UNIQUE
);
```

Code 3.12: Adding a unique constraint to a column in MySQL

Code 3.12 shows an example of adding a unique constraint to a column in MySQL using the **UNIQUE** keyword. The statement creates a new table named **table_name** with a unique column named **email**.

3.8.3 Not Null Constraint

The **NOT NULL** constraint is used to ensure that a column cannot contain NULL values. A not null constraint requires that a column must have a value for each record. To add a not null constraint to a column in MySQL, you can use the **NOT NULL** keyword in the column definition.

```sql
CREATE TABLE table_name (
  id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(50) NOT NULL,
  email VARCHAR(100)
);
```

Code 3.13: Adding a not null constraint to a column in MySQL

Code 3.13 shows an example of adding a not null constraint to a column in MySQL using the **NOT NULL** keyword. The statement creates a new table named **table_name** with a not null column named **name**.

3.8.4 Foreign Key Constraint

The **FOREIGN KEY** constraint is used to establish a relationship between two tables in a database. A foreign key constraint specifies that the values in a column must match the values in another column in a different table. To add a foreign key constraint to a column in MySQL, you can use the **FOREIGN KEY** keyword in the column definition.

```sql
CREATE TABLE table_name (
  id INT PRIMARY KEY AUTO_INCREMENT,
  user_id INT,
  FOREIGN KEY (user_id) REFERENCES users(id)
);
```

Code 3.14: Adding a foreign key constraint to a column in MySQL

Code 3.14 shows an example of adding a foreign key constraint to a column in MySQL using the **FOREIGN KEY** keyword. The statement creates a new table named **table_name** with a foreign key column named **user_id** that references the **id** column in the **users** table.

3.9 Altering Tables with Constraints

To alter a table in MySQL with constraints, you can use the **ALTER TABLE** statement followed by the name of the table and the type of alteration to perform. The following example adds a primary key constraint to the **id** column in the **table_name** table:

```sql
ALTER TABLE table_name ADD PRIMARY KEY (id);
```

Code 3.15: Adding a primary key constraint to a column in MySQL

Code 3.15 shows an example of adding a primary key constraint to a column in a table in MySQL using the **ALTER TABLE** statement. The statement adds a primary key constraint to the **id** column in the **table_name** table.

```sql
ALTER TABLE table_name MODIFY COLUMN name VARCHAR(50) NOT NULL;
```

Code 3.16: Adding a not null constraint to a column in MySQL

Code 3.16 shows an example of adding a not null constraint to a column in a table in MySQL using the **ALTER TABLE** statement. The statement modifies the **name** column in the **table_name** table to add a not null constraint.

Exercises

```
1. In the books table, add a unique constraint to the "isbn_number" column.
2. In the books table, add a not null constraint to the "title" column.
```

3.10 Truncating Tables

To remove all records from a table in MySQL, you can use the **TRUNCATE TABLE** statement followed by the name of the table. The following example removes all records from the **table_name** table:

```sql
TRUNCATE TABLE table_name;
```

Code 3.17: Truncating a table in MySQL

Code 3.17 shows an example of truncating a table in MySQL using the **TRUNCATE TABLE** statement. The statement removes all records from the **table_name** table, but the table structure remains intact.

4

Managing Data

4.1 Inserting Data

To insert data into a table in MySQL, you can use the **INSERT INTO** statement followed by the name of the table and a list of values. The following example inserts a new record into the **table_name** table:

```sql
INSERT INTO table_name (name, age) VALUES ('Alice', 25);
```

Code 4.1: Inserting data into a table in MySQL

Code 4.1 shows an example of inserting data into a table in MySQL using the **INSERT INTO** statement. The statement inserts a new record with values for the **name** and **age** columns in the **table_name** table.

It is also possible to insert multiple records into a table using a single **INSERT INTO** statement. To insert multiple records, you can specify multiple sets of values separated by commas.

```sql
INSERT INTO table_name (name, age) VALUES
  ('Alice', 25),
  ('Bob', 30),
  ('Charlie', 35);
```

Code 4.2: Inserting multiple records into a table in MySQL

Exercises

```
1. Create a database named "library".
2. Create a table named "books" with columns:
    - id (INT, PRIMARY KEY, AUTO_INCREMENT)
    - title (VARCHAR, 100)
    - author (VARCHAR, 50)
    - year (INT)
    - genre (VARCHAR, 50)
    - isbn (VARCHAR, 20)
3. Insert the following records in the "books" table:
```

id	title	author	year	genre	isbn
1	The Great Gatsby	F. Scott Fitzgerald	1925	Fiction	9780743273565
2	The Shining	Stephen King	1977	Horror	9780307743657
3	The Da Vinci Code	Dan Brown	2003	Mystery	9780307474278
4	Dune	Frank Herbert	1965	Science Fiction	9780441172719
5	The Hobbit	J.R.R. Tolkien	1937	Fantasy	9780618260300
6	Pride and Prejudice	Jane Austen	1813	Romance	9780141439518
7	To Kill a Mockingbird	Harper Lee	1960	Fiction	9780061120084

Table 4.1: Records to insert in the "books" table

4.2 Selecting Data

To select data from a table in MySQL, you can use the **SELECT** statement followed by a list of columns or the wildcard character (*). The following example selects all records from the **table_name** table:

```sql
SELECT * FROM table_name;
```

Code 4.3: Selecting data from a table in MySQL

Code 4.3 shows an example of selecting data from a table in MySQL using the **SELECT** statement. The statement selects all records from the **table_name** table and displays the data in the result set.

4.3 Updating Data

To update data in a table in MySQL, you can use the **UPDATE** statement followed by the name of the table and a list of columns and values to update. The following example updates the **name** column in the **table_name** table:

```sql
UPDATE table_name SET name = 'Bob' WHERE id = 1;
```

Code 4.4: Updating data in a table in MySQL

Code 4.4 shows an example of updating data in a table in MySQL using the **UPDATE** statement. The statement updates the **name** column to **'Bob'** for the record with an **id** value of **1** in the **table_name** table.

Exercises

```
1. Update the "title" column in the "books" table where the "id" is 1 to "The
   Great Gatsby (Revised Edition)".
2. Update the "year" column in the "books" table where the "id" is 2 to 1977.
```

4.4 Deleting Data

To delete data from a table in MySQL, you can use the **DELETE FROM** statement followed by the name of the table and a condition to filter the records to delete. The following example deletes records from the **table_name** table:

```sql
DELETE FROM table_name WHERE id = 1;
```

Code 4.5: Deleting data from a table in MySQL

Code 4.5 shows an example of deleting data from a table in MySQL using the **DELETE FROM** statement. The statement deletes records from the **table_name** table where the **id** value is **1**.

Exercises

```
1. Delete the record from the "books" table where the genre is "Romance".
```

5

References

A. Books

-

B. Other Sources

-