Tables

Before we get started with SQL, let's learn how to create tables and columns.

As an example, we are going to create a `users` table with the following columns:

- `id` - this is going to be the primary key of the table and would be the unique identifier of each user.
- `username` - this column would hold the username of our users.
- `name` - here, we will store the full name of users.
- `status` - here, we will store the status of a user, which would indicate if a user is active or not.

You need to specify the data type of each column.

In our case it would be like this:

- `id` - Integer
- `username` - Varchar
- `name` - Varchar
- `status` - Number

Data types

The most common data types that you would come across are:

- `CHAR`(size): Fixed-length character string with a maximum length of 255 bytes.
- `VARCHAR`(size): Variable-length character string. Max size is specified in parenthesis.
- `TEXT`(size): A string with a maximum length of 65,535 bytes.
- `INTEGER`(size) or `INT`(size): A medium integer.
- `BOOLEAN` or `BOOL`: Holds a true or false value.
- `DATE`: Holds a date.

Let's have the following users table as an example:

- `id`: We would want to set the ID to `INT`.
- `name`: The name should fit in a `VARCHAR` column.
- `about`: As the about section could be longer, we could set the column data type to `TEXT`.
- `birthday`: For the birthday column of the user, we could use `DATE`.

For more information on all data types available, make sure to check out the official documentation here.

Creating a database

As we briefly covered in the previous chapter, before you could create tables, you would need to create a database by running the following:

- First access MySQL:

```
mysql -u root -p
```

- Then create a database called `demo_db`:

```
CREATE DATABASE demo_db;
```

Note: the database name needs to be unique, if you already have a database named `demo_db` you would receive an error that the database already exists.

You can consider this database as the container where we would create all of the tables in.

Once you've created the database, you need to switch to that database:

```
USE demo_db;
```

You can think of this as accessing a directory in Linux with the `cd` command. With `USE`, we switch to a specific database.

Alternatively, if you do not want to 'switch' to the specific database, you would need to specify the so-called fully qualified table name. For example, if you had a `users` table in the `demo_db`, and you wanted to

select all of the entries from that table, you could use one of the following two approaches:

- Switch to the `demo_db` first and then run a select statement:

```
USE demo_db;
SELECT username FROM users;
```

- Alternatively, rather than using the `USE` command first, specify the database name followed by the table name separated with a dot: `db_name.table_name`:

```
SELECT username FROM demo_db.users;
```

We are going to cover the `SELECT` statement more in-depth in the following chapters.

Creating tables

In order to create a table, you need to use the `CREATE TABLE` statement followed by the columns that you want to have in that table and their data type.

Let's say that we wanted to create a `users` table with the following columns:

- `id`: An integer value
- `username`: A varchar value
- `about`: A text type
- `birthday`: Date
- `active`: True or false

The query that we would need to run to create that table would be:

```
CREATE TABLE users
(
    id INT,
    username VARCHAR(255),
    about TEXT,
    birthday DATE,
    active BOOL
);
```

Note: You need to select a database first with the `USE` command as mentioned above. Otherwise you will get the following error: `ERROR 1046 (3D000): No database selected.

To list the available tables, you could run the following command:

```
SHOW TABLES;
```

Output:

```
+--------------------+
| Tables_in_demo_db |
+--------------------+
| users              |
+--------------------+
```

Creating a new table from an existing table

You can create a new table from an existing table by using the `CREATE TABLE AS` statement.

Let's test that by creating a new table from the table `users` which we created earlier.

```
CREATE TABLE users2 AS
(
    SELECT * FROM users
);
```

The output that you would get would be:

```
Query OK, 0 rows affected (0.07 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Note: When creating a table in this way, the new table will be populated with the records from the existing table (based on the SELECT Statement)

Rename tables

You can rename a table by using `ALTER TABLE` statement.

Let's change name of user2 table to user3

```
ALTER TABLE user2 RENAME TO user3
```

Dropping tables

You can drop or delete tables by using the `DROP TABLE` statement.

Let's test that and drop the table that we've just created:

```
DROP TABLE users;
```

The output that you would get would be:

```
Query OK, 0 rows affected (0.03 sec)
```

And now, if you were to run the `SHOW TABLES;` query again, you would get the following output:

```
Empty set (0.00 sec)
```

Allowing NULL values

By default, each column in your table can hold NULL values. In case that you don't wanted to allow NULL values for some of the columns in a specific table, you need to specify this during the table creation or later on change the table to allow that.

For example, let's say that we want the `username` column to be a required one, we would need to alter the table create statement and include `NOT NULL` right next to the `username` column like this:

```
CREATE TABLE users
(
    id INT,
    username VARCHAR(255) NOT NULL,
    about TEXT,
    birthday DATE,
    active BOOL
);
```

That way, when you try to add a new user, MySQL will let you know that the `username` column is required.

Specifying a primary key

The primary key column, which in our case is the `id` column, is a unique identifier for our users.

We want the `id` column to be unique, and also, whenever we add new users, we want the ID of the user to autoincrement for each new user.

This can be achieved with a primary key and `AUTO_INCREMENT`. The primary key column needs to be `NOT NULL` as well.

If we were to alter the table creation statement, it would look like this:

```
CREATE TABLE users
(
    id INT PRIMARY KEY AUTO_INCREMENT,
    username VARCHAR(255) NOT NULL,
    about TEXT,
    birthday DATE,
    active BOOL
);
```

Index Optimization for Database Queries

In database management, establishing a PRIMARY KEY for tables is fundamental. Using our previous example, the `id` column serves as this primary key. However, as the volume of data grows, searching by attributes other than the primary key, like the `date of birth`, can become increasingly slow. To optimize such queries, you can introduce an INDEX on specific columns.

Consider the `birthday` column. To enhance the speed of queries focused on this column, an INDEX can be pivotal:

```
CREATE INDEX birthday_idx ON users(birthday);
```

Tip: For queries spanning multiple fields, you have the option to create a composite index incorporating all the relevant fields. Let's say, for example, you want to index both the `birthday` and `active` columns.

The order in which you list the fields in a composite index matters. For instance, given that the `active` column might have limited unique values (e.g., true or false) compared to the `birthday` column, the sequence of fields in the index can influence efficiency. Designing the index like this:

```
CREATE INDEX users_multi_idx ON users(active, birthday);
```

May not be as efficient as:

```
CREATE INDEX users_multi_idx ON users(birthday, active);
```

Placing `birthday` first in the index ensures a quicker reduction in potential matches, optimizing the server's data manipulation process.

Updating tables

In the above example, we created a new table and then dropped it as it was empty. However, in a real-life scenario, this would really be the case.

So whenever you need to add or remove a new column from a specific table, you would need to use the `ALTER TABLE` statement.

Let's say that we wanted to add an `email` column with type varchar to our `users` table.

The syntax would be:

```
ALTER TABLE users ADD email VARCHAR(255);
```

After that, if you were to describe the table, you would see the new column:

```
DESCRIBE users;
```

Output:

```
+----------+--------------+------+-----+---------+
| Field    | Type         | Null | Key | Default |
+----------+--------------+------+-----+---------+
| id       | int          | NO   | PRI | NULL    |
| username | varchar(255) | NO   |     | NULL    |
| about    | text         | YES  |     | NULL    |
| birthday | date         | YES  |     | NULL    |
| active   | tinyint(1)   | YES  |     | NULL    |
| email    | varchar(255) | YES  |     | NULL    |
+----------+--------------+------+-----+---------+
```

If you wanted to drop a specific column, the syntax would be:

```
ALTER TABLE table_name DROP COLUMN column_name;
```

Note: Keep in mind that this is a permanent change, and if you have any critical data in the specific column, it would be deleted instantly.

You can use the `ALTER TABLE` statement to also change the data type of a specific column. For example, you could change the `about` column from `TEXT` to `LONGTEXT` type, which could hold longer strings.

Note: Important thing to keep in mind is that if a specific table already holds a particular type of data value like an integer, you can't alter it to varchar, for example. Only if the column does not contain any values, then you could make the change.

Truncate table

The `TRUNCATE TABLE` command is used to **delete all of the data** from an existing table, but not the table itself.

- Syntax of Truncate table:

```
TRUNCATE TABLE table_name;
```

- Example:

Consider a Sellers table having the following records:

```
+----+----------+-----+-----------+----------+
| ID | NAME     |Items| CITY      | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Shivam   |  34 | Ahmedabad |  2000.00 |
|  2 | Ajay     |  22 | Delhi     |  4400.00 |
|  3 | Kaushik  |  28 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6600.00 |
|  5 | Hardik   |  26 | Bhopal    |  8100.00 |
|  6 | Maria    |  23 | MP        |  4200.00 |
|  7 | Muffy    |  29 | Indore    |  9000.00 |
+----+----------+-----+-----------+----------+
```

Following is the example of a Truncate command:

```
TRUNCATE TABLE Sellers;
```

After that if you do a `COUNT(*)` on that table you would see that the table is completely empty.