Hey guys!
Welcome to my SQL Notes :)

Context :

To begin, SQL stands for *Structured Query Language*.

With it, you can do things like :

create a database
add or alter data in it
retrieve and update the data
delete the data

Popular DBMS (Database Management Systems) include :

MySQL (we'll use this one!)
SQLite
…

Basic Concepts :

- **Database**: A structured collection of data.

- **Table**: A collection of related data, organized in rows and columns. Think of it like a spreadsheet.
  - Example:

Column

ID	Name	Age	
1	Alice	25	*Row*
2	Bob	30	

- **Row (Record)**: A single data entry.

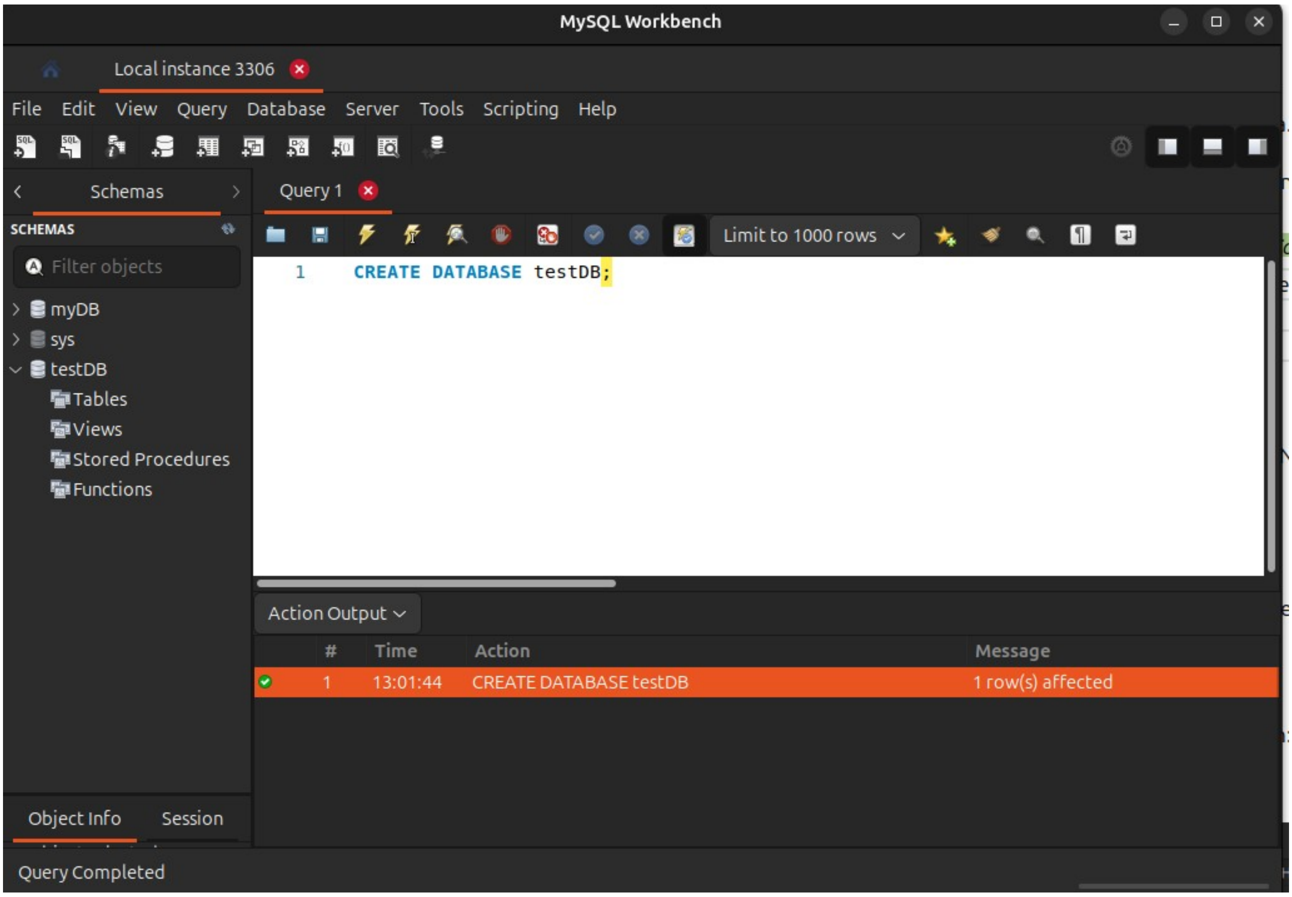
- **Column (Field)**: A category of data (e.g., Name, Age).

Essential SQL Commands :

SELECT	**:**	Retrieve data from a table.
INSERT	**:**	Add new data.
UPDATE	**:**	Modify existing data.
DELETE	**:**	Remove data.

We'll look at each command in depth …

1. We will create a database in MySQL Workbench using the **CREATE DATABASE** statement:



```
CREATE DATABASE testDB;
```

Action Output

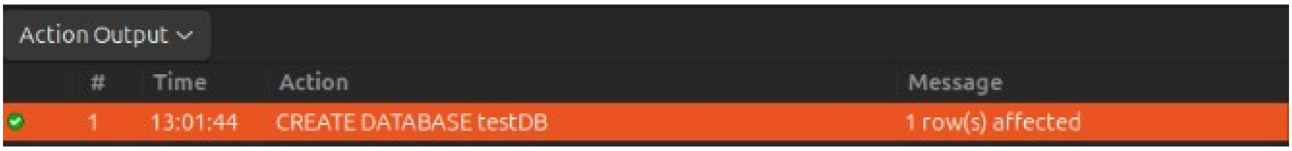
#	Time	Action	Message
1	13:01:44	CREATE DATABASE testDB	1 row(s) affected

Query Completed

After writing down the statement, don't forget to include a **";"**

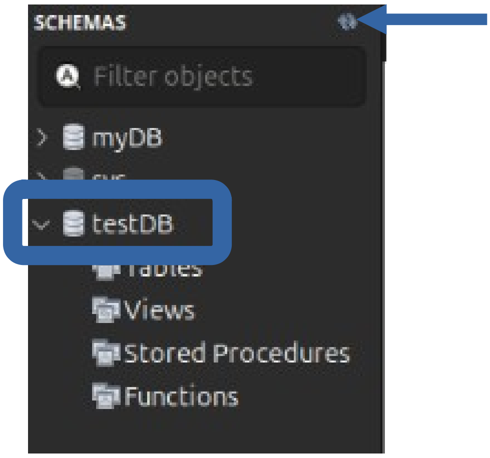
Then hit the **"bolt" icon** to RUN it : ⚡

After successful execution, the **"Action Output"** section at the bottom of the app will show you a **green** circle:



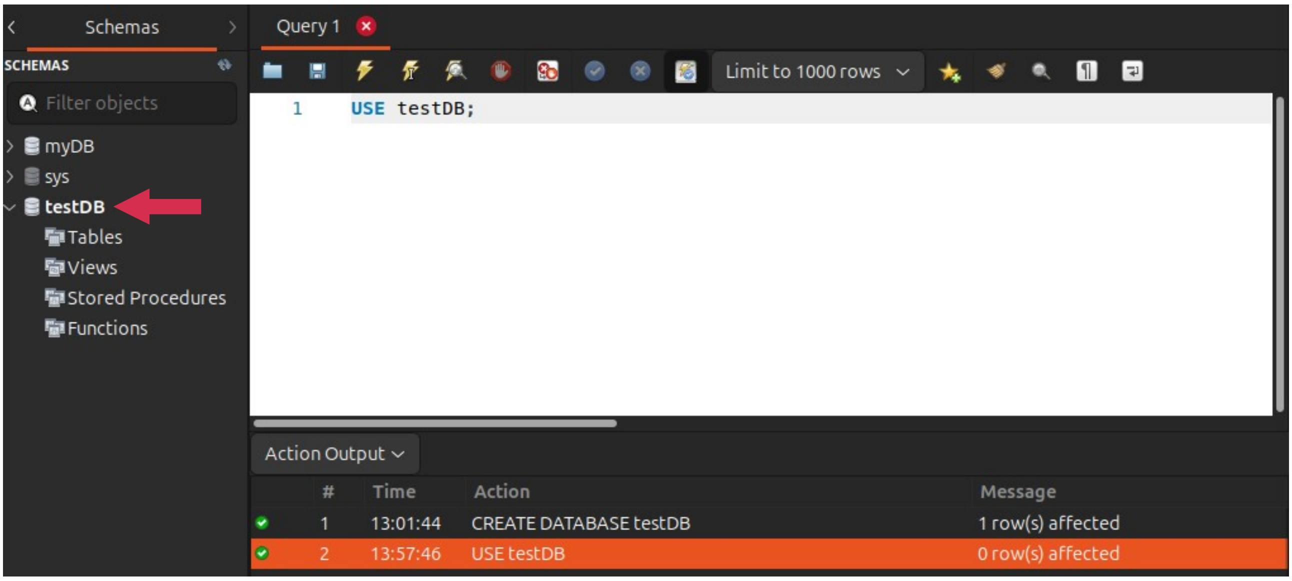
Action Output

#	Time	Action	Message
1	13:01:44	CREATE DATABASE testDB	1 row(s) affected



And hitting the refresh button on the **"SCHEMAS"** tab to the left will show you a new database has been created called **"testDB":**

2. Set the current database to “testDB” with the **USE** statement:



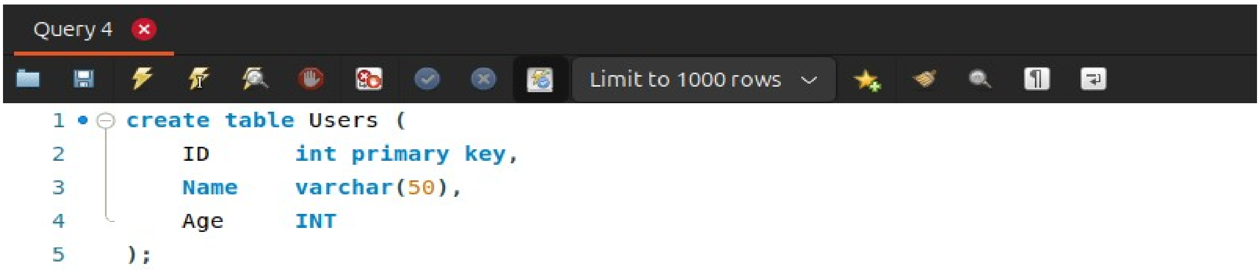
```
USE testDB;
```

#	Time	Action	Message
1	13:01:44	CREATE DATABASE testDB	1 row(s) affected
2	13:57:46	USE testDB	0 row(s) affected

You’ll notice upon execution, that the ***testDB*** schema (in the “SCHEMAS” tab to the left) is now in **bold**.

---

3. Let’s create a table for our database with the **CREATE TABLE** statement !



```
create table Users (
        ID       int primary key,
        Name     varchar(50),
        Age      INT
    );
```

**** you can use upper or lower-case lettering for the terms like “create table”, “varchar” … in MySQL.***

We’ll examine the columns or fields we entered one by one:

- **The “ID” column** : **INT** (integer)
  PRIMARY KEY (ensures no duplicate ID’s are allowed thus, acts as a unique identifier)
- **The “Name” column** : **VARCHAR(50)** (variable character with max number of characters = 50)
- **The “Age” column** : **INT** (integer)

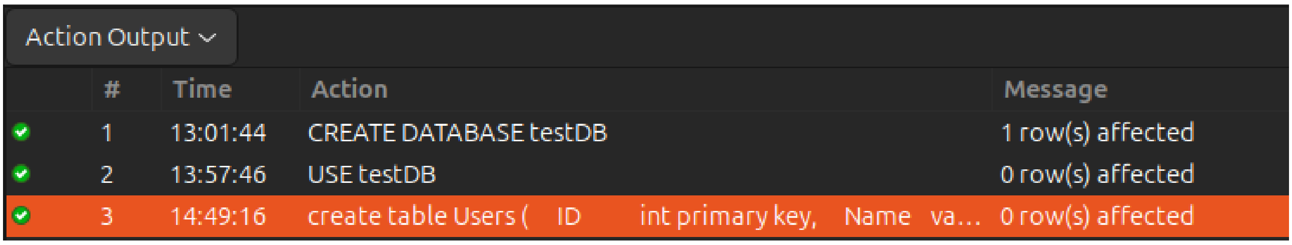
Example:

ID	Name	Age
1	Alice	25
2	Bob	30

Here:

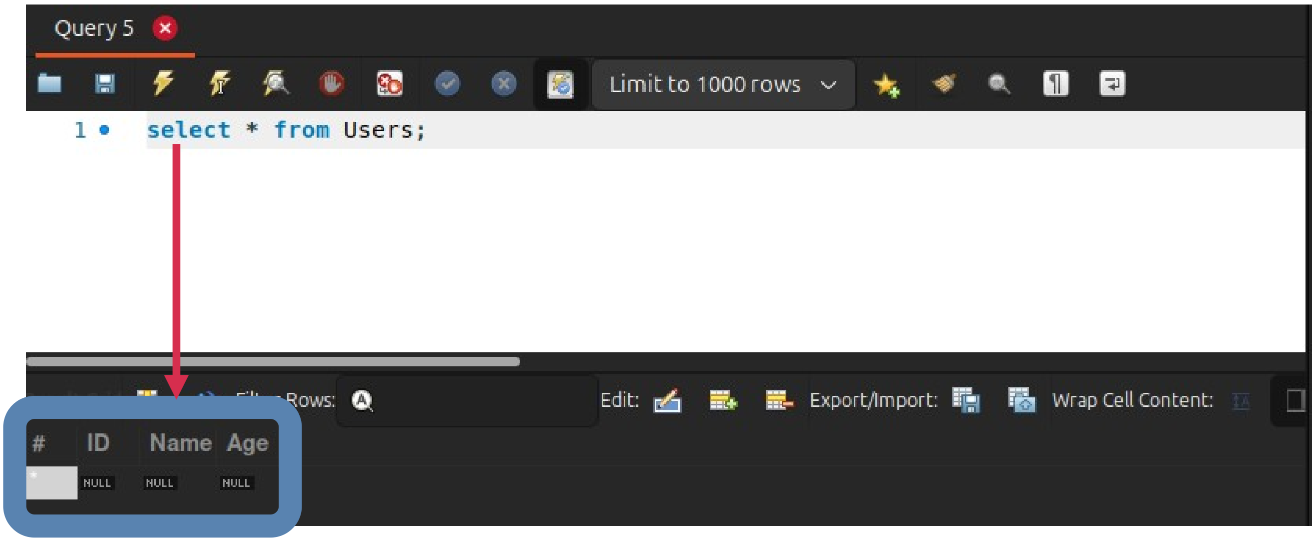
- 1 and 2 are unique primary key values.
- You cannot insert a row with ID = 1 again because it would violate ***the uniqueness constraint.***

Now, once again, we'll hit the *bolt* icon to execute the SQL Query we just wrote and check for the green circle in *Action Outputs:*



	#	Time	Action	Message
✓	1	13:01:44	CREATE DATABASE testDB	1 row(s) affected
✓	2	13:57:46	USE testDB	0 row(s) affected
✓	3	14:49:16	create table Users (ID int primary key, Name va...	0 row(s) affected

You can also see the table we just created using the SELECT statement:



```sql
select * from Users;
```

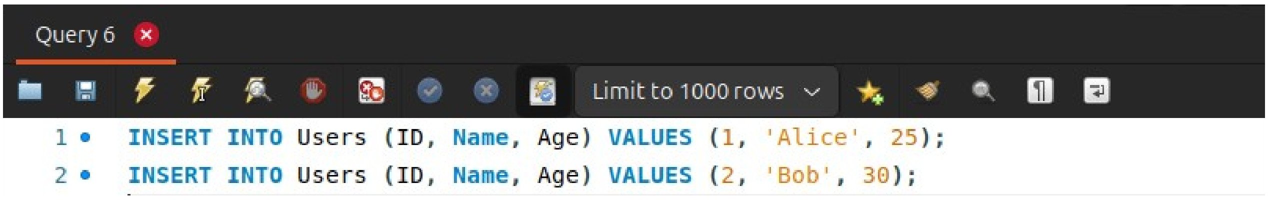
#	ID	Name	Age
*	NULL	NULL	NULL

The " * " means *ALL*. So, the statement reads : *Select ALL columns from the Table Users and display them*.

---

4. Our table looks sad without any data !

So let's insert some data using the **INSERT** statement :)



```sql
INSERT INTO Users (ID, Name, Age) VALUES (1, 'Alice', 25);
INSERT INTO Users (ID, Name, Age) VALUES (2, 'Bob', 30);
```

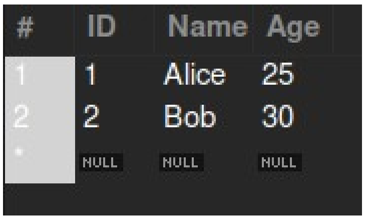
So in the first set of brackets, we include the column names and in the second set, we include the values.

Notice for the ID and Age columns, we entered *integers* and for the Name column, we entered a *character string* 'Alice'.

Now, we run the statements like before and view our table with the SELECT statement from step 3:

Voila !

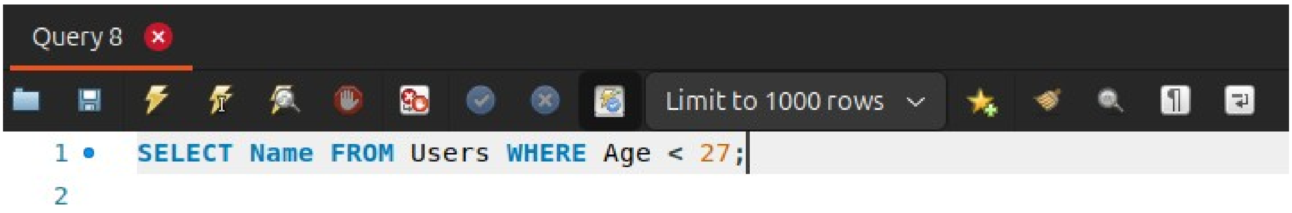
(Take a moment to congratulate yourself ^^ - you deserve it !)



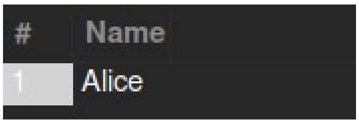
#	ID	Name	Age
1	1	Alice	25
2	2	Bob	30
*	NULL	NULL	NULL

5. Now what if we want to only retrieve data based on some specific criteria or filters. For example, we want to see all our users under the age of 27.

In this case, we'll use a SELECT statement but with some conditions :



```sql
SELECT Name FROM Users WHERE Age < 27;
```



#	Name
1	Alice

Now we'll run this as before and see what happens …

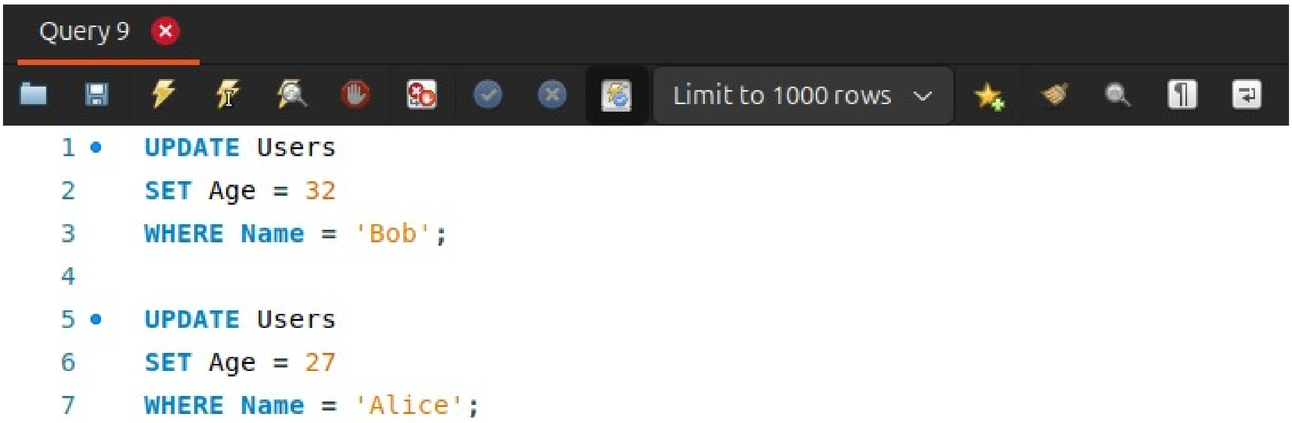
Since we have only one user under the age of 27, and we specified we only want to retrieve the *Name* field, we get a result as such.

*In case you're wondering, the column with a " # " as its label is simply a **row number indicator** that Workbench gives us for easier navigation along the table.*

---

6. Let's say it's been 2 years and we want to update the ages of Alice and Bob accordingly.

We would use the UPDATE statement for this :

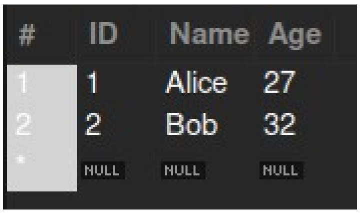


```sql
UPDATE Users
SET Age = 32
WHERE Name = 'Bob';

UPDATE Users
SET Age = 27
WHERE Name = 'Alice';
```

If you encounter an error when running it that says you are in "safe update mode", simply go to Edit > Preferences > SQL Editor > uncheck the last checkbox at the bottom (that says sth about safe update and delete)

Once the run is successful, do step 3 to view the updated table :

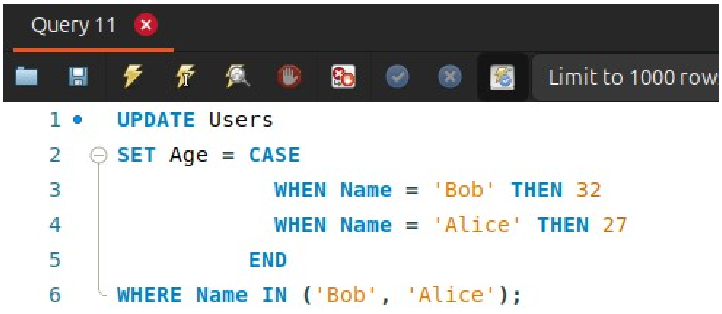


#	ID	Name	Age
1	1	Alice	27
2	2	Bob	32
*	NULL	NULL	NULL

Is there a more efficient way to do this? Imagine we had to update 10 ages, would we repeat the three lines 10 times?

Well, we could do something like this:

We'll understand how this works next .



```sql
UPDATE Users
SET Age = CASE
            WHEN Name = 'Bob' THEN 32
            WHEN Name = 'Alice' THEN 27
          END
WHERE Name IN ('Bob', 'Alice');
```

The **CASE** statement specifies different values for the *Age* column depending on the value of *Name* column.

The **WHERE** clause is really important here, because it ensures that the updates only apply to the specific rows where Alice and Bob exist. No other rows will be modified.

If we didn't have the **WHERE** clause, the **UPDATE** statement is applied to every row and could cause unwanted changes to the values in Age columns, such as turning every *Age* value other than "Bob" and "Alice"'s to ***NULL.***

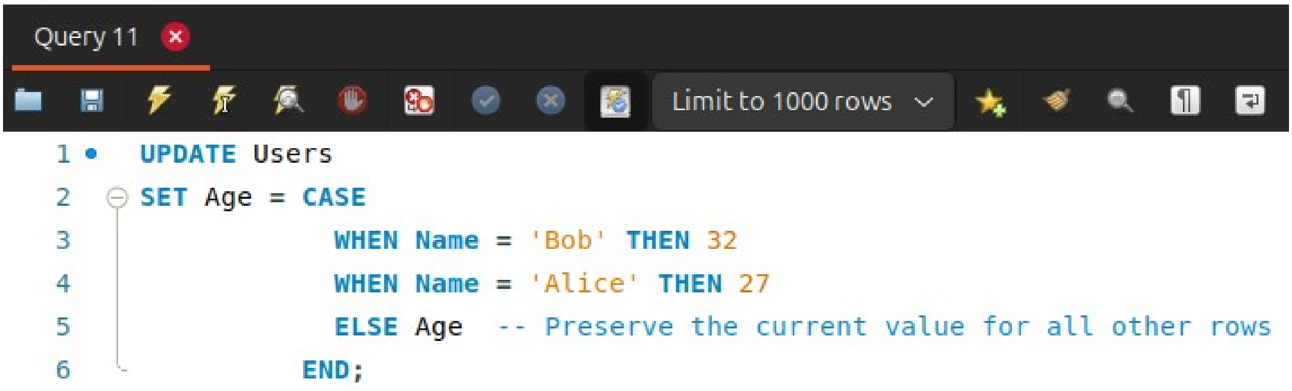
For example, imagine we had another row for "Carol":

ID	Name	Age
1	Alice	25
2	Bob	30
3	Carol	27

After running the query without a **WHERE** or **ELSE** clause:

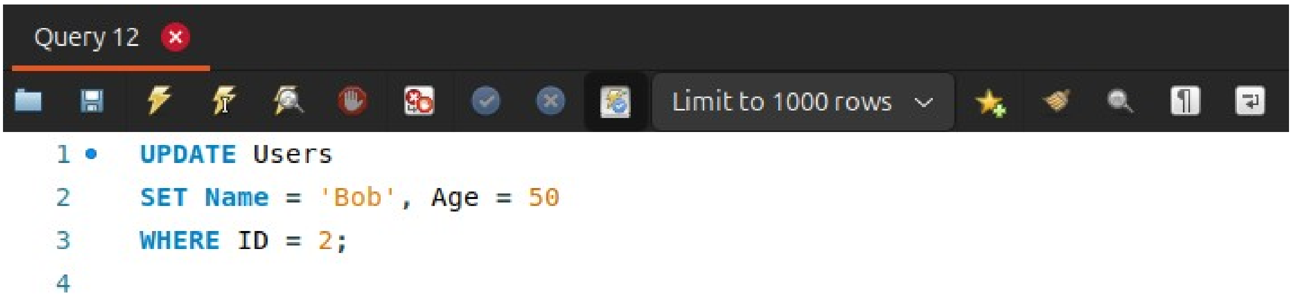
ID	Name	Age
1	Alice	27
2	Bob	32
3	Carol	**NULL**

So, if we omit the **WHERE** clause, we must include an **ELSE** clause instead as such:



```sql
UPDATE Users
SET Age = CASE
            WHEN Name = 'Bob' THEN 32
            WHEN Name = 'Alice' THEN 27
            ELSE Age  -- Preserve the current value for all other rows
          END;
```

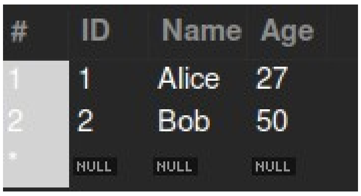
Now imagine Bob changed his name to Alex and was lying about his actual age. Turns out he's 50. How can we change both the *Name* and *Age* of "Bob" using a single **UPDATE** statement?



```sql
UPDATE Users
SET Name = 'Bob', Age = 50
WHERE ID = 2;
```

We simply separate the column updates with a " , ".

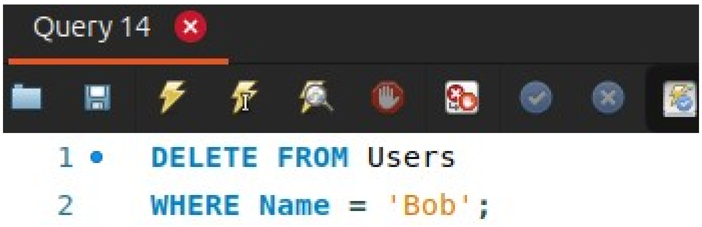
Here's the updated table retrieved with the **SELECT** statement from step 3 :



#	ID	Name	Age
1	1	Alice	27
2	2	Bob	50
*	NULL	NULL	NULL

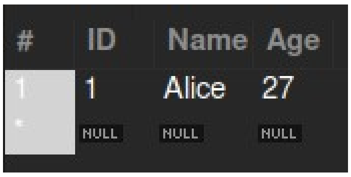
7. Finally, we want to remove Bob … no Alex, from the list completely, because guess what? He's lied about his age again! Seems like a fraudster :(

For this we'll use the **DELETE** statement :



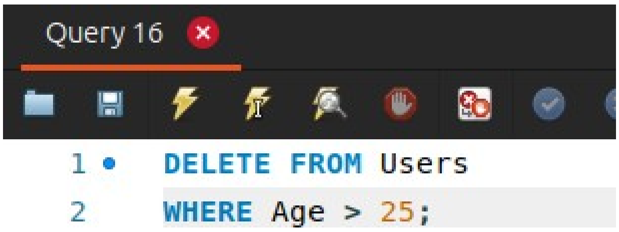
```
DELETE FROM Users
WHERE Name = 'Bob';
```

The resulting table has Bob removed from it :



#	ID	Name	Age
1	1	Alice	27
*	NULL	NULL	NULL

Now, imagine all the users we recorded as age > 25 were actually fraudsters. We want to delete all of them from our Users table :

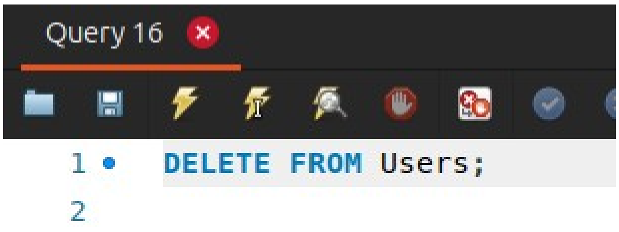


```
DELETE FROM Users
WHERE Age > 25;
```

And that takes care of all of them in one strike :)

What if we want to delete the Users table entirely, and start it from scratch?

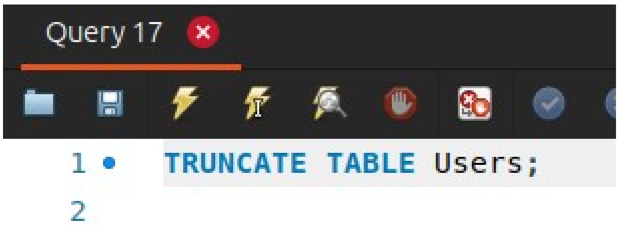
- **DELETE** :



```
DELETE FROM Users;
```

A simple DELETE statement without a WHERE clause.

- **TRUNCATE** :



```
TRUNCATE TABLE Users;
```

TRUNCATE doesn't allow a WHERE clause, so it's faster

And that concludes the basics !