**COMP353 Project #1**

**Warm-up Project**

Mair Elbaz, 40004558

Daniel Vigny-Pau, 40034769

Francois David, 40046319

Alexandre Therrien, 40057134

Charles-Antoine Guite, 40063098

Professor: Dr. Desai

Fall 2019

Concordia University

**Creating the Tables**

Below are the lines of code that we have put in our program to create the various tables required for this project.

//query the database with the command in (). Creates table 'people'

$mysqli->query("CREATE TABLE people (id smallint unsigned not null auto\_increment, lastname varchar(20) not null, firstname varchar(20) not null, middle\_name varchar(20), userID int not null, password int not null, constraint pk\_people primary key (id) );");

This code queries our database to create a new table with the name ‘people’. That is, assuming you are already connected to your database and that you have selected a database to use. The $mysqli variable is the variable used to store the connection to our database. The characters in blue compose the query.

We are to create a table that has a variable id, which is never to be attributed the value null (from the constraint “not null”), that is always positive (“unsigned”), and that increments automatically (“auto\_increment”) so that there are never two same id values. It is also a small int value, which means it has a smaller range than an int value. The last statement of this query (constraint pk\_people primary key (id) ) indicates that this variable will be used as the primary key.

The rest of the statements between commas are giving the different columns this new table will have. The varchar type is a type of data element that contains a series of characters. We have the ‘lastname’ element, which has the constraint to be not null and a maximum length of 20. We have the ‘firstname’ element, which has the constraint to be not null and a maximum length of 20. We have the ‘middle\_name’ element, which has a maximum length of 20 and can be null.

Then we have two int values, userID and password. These are integer types, both of which have the constraint to be not null.

//query the database with the command in (). Creates table 'event'

$mysqli->query("CREATE TABLE event (id smallint unsigned not null auto\_increment, Event varchar(20), EventID int, start\_date varchar(20), end\_date varchar(20), AdminUserID int, constraint pk\_event primary key (id));");

Similar to the previous table, this statement creates a table called ‘event’ and has the same primary key statements as in the previous example.

It creates the different columns Event (varchar of maximum length 20 that can be null), EventID (integer value that can be null), start\_date (varchar of maximum length 20 that can be null), end\_date (varchar of maximum length 20 that can be null), and AdminUserID (integer value that can be null).

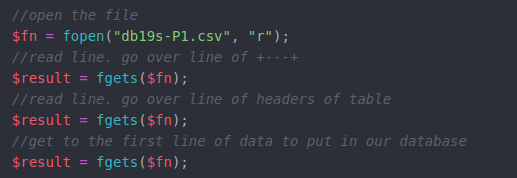
//query the database with the command in (). Creates table 'role\_of\_people\_in\_the\_event'

$mysqli->query("CREATE TABLE role\_of\_people\_in\_the\_event (id smallint unsigned not null auto\_increment, userid int, EventID int, constraint pk\_role primary key (id));");

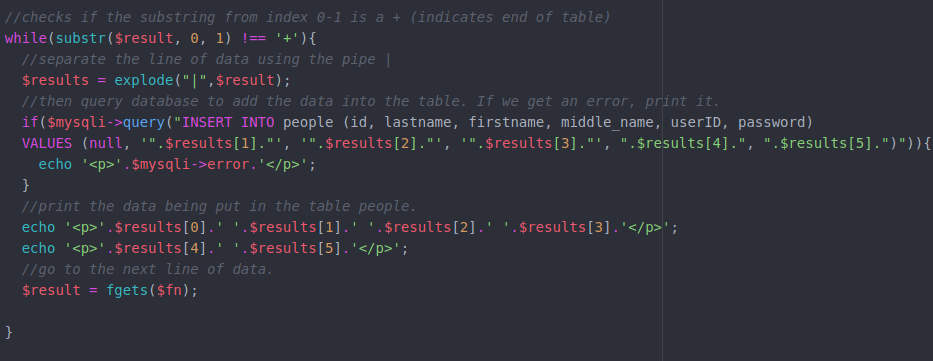
Similar to the previous table, this statement creates a table called ‘role\_of\_people\_in\_the\_event’ and has the same primary key statements as in the previous example.

This table has a userid column (integer, could be null) and an EventID column (integer, not null).

**Prepare and Read Input Data**

To start reading the input data, some operations had to be done before.

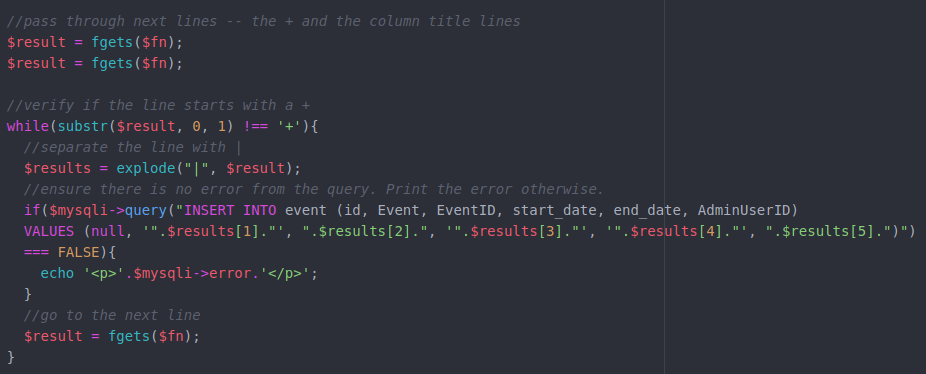
First, I needed to open the file in php with read authorization. Then, I placed my $result variable to the first line of data of the first table.



Then, I am doing a loop until I read a line starting with the + value (which is the line that separates the different tables from one another). To read all the different data to go in the different columns of the table, I separate the line of data received by using the pipe ( | ) character, which is what separates those values. Because the line starts with a pipe character, the first array element we are interested in is the index 1, as index 0 will always be an empty string.

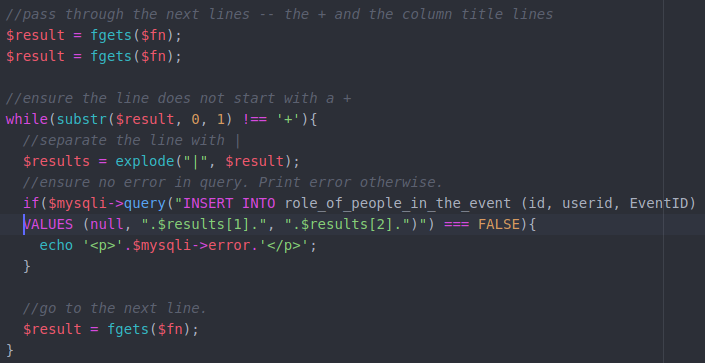
I follow that with an if condition that does the query and verifies that the output is not FALSE, which would indicate that there is an error in the query. If there is an error, I would print the error in the browser. This query indicates that we want to add some data to the table ‘people’ in the following order: (id, lastname, firstname, middle\_name, userID, password). The INSERT INTO query adds that data into the table ‘people’. Then, we use VALUES (…) with the values in the same order as they were mentioned before.

Then the results of my separation using the pipe character are printed to the screen, and I follow that by going into the next line of the file doing $result = fgets($fn);



Next, once we have reached a line starting with a + (which separates the tables), we exit the loop from the previous image. We then go over that + line and the column titles’ line using fgets, and we enter a new loop which is very similar to the previous one. The only difference is that the results are not getting printed and the query has been adapted to add the data into the ‘event’ table. This time I put the first result from the split as Event, then EventID, start\_date, end\_date, and finally AdminUserID.

I then go to the next line.



Same process here. Once we reach the line starting with + separating the tables, we exit the loop in the previous image and enter this one after going over the + and the column titles’ lines. This time I add the content of the split into the ‘role\_of\_people\_in\_the\_event’.

**Accessing the data**

We begin by first creating our table and its header, creating a column for each corresponding field in the tables we have created earlier. Take for example the table `event`.

A picture containing indoor, screenshot

Description automatically generated

We define a title and print it, followed by our opening tags for our table, along with its table header.

A picture containing indoor, screenshot

Description automatically generated

To access the data, we create an SQL query where we select all the data from a table, such as the table `event` we have previously created. `\*` denotes that we are selecting all the data, whilst `ORDER BY id ASC` indicates that we are ordering the entries in the table in ascending order by ID.

A picture containing indoor, screenshot

Description automatically generated

Now we have two cases: if the number of rows returned by our previous call is more than 0, we will display the results in the table. If there are no results from our query, we will print that there are no results. Using `fetch\_assoc()`, we can iterate through each entry of our query results and print a table row for each one.

A picture containing indoor, screenshot

Description automatically generated

Once we have finished printing our table rows, we can close the tags of our table. This gives us the following table when we execute the code above:

A screenshot of a video game

Description automatically generated

However, by modifying our query, we can obtain a variety of different results.

Our original query for obtaining the table of people was:

A close up of a screen

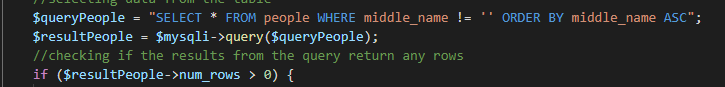
Description automatically generated

This one selects all entries from the table `people` and orders them by `id` in ascending order.

A screenshot of a cell phone

Description automatically generated

However we can change this query to a number of different ones, such as the following:



Which would return only people with a middle name, or rather where their field for `middle\_name` is not empty, sorted by their middle name in alphabetical order.

A screenshot of a social media post

Description automatically generated

Or consequently, if the query were changed to something like:

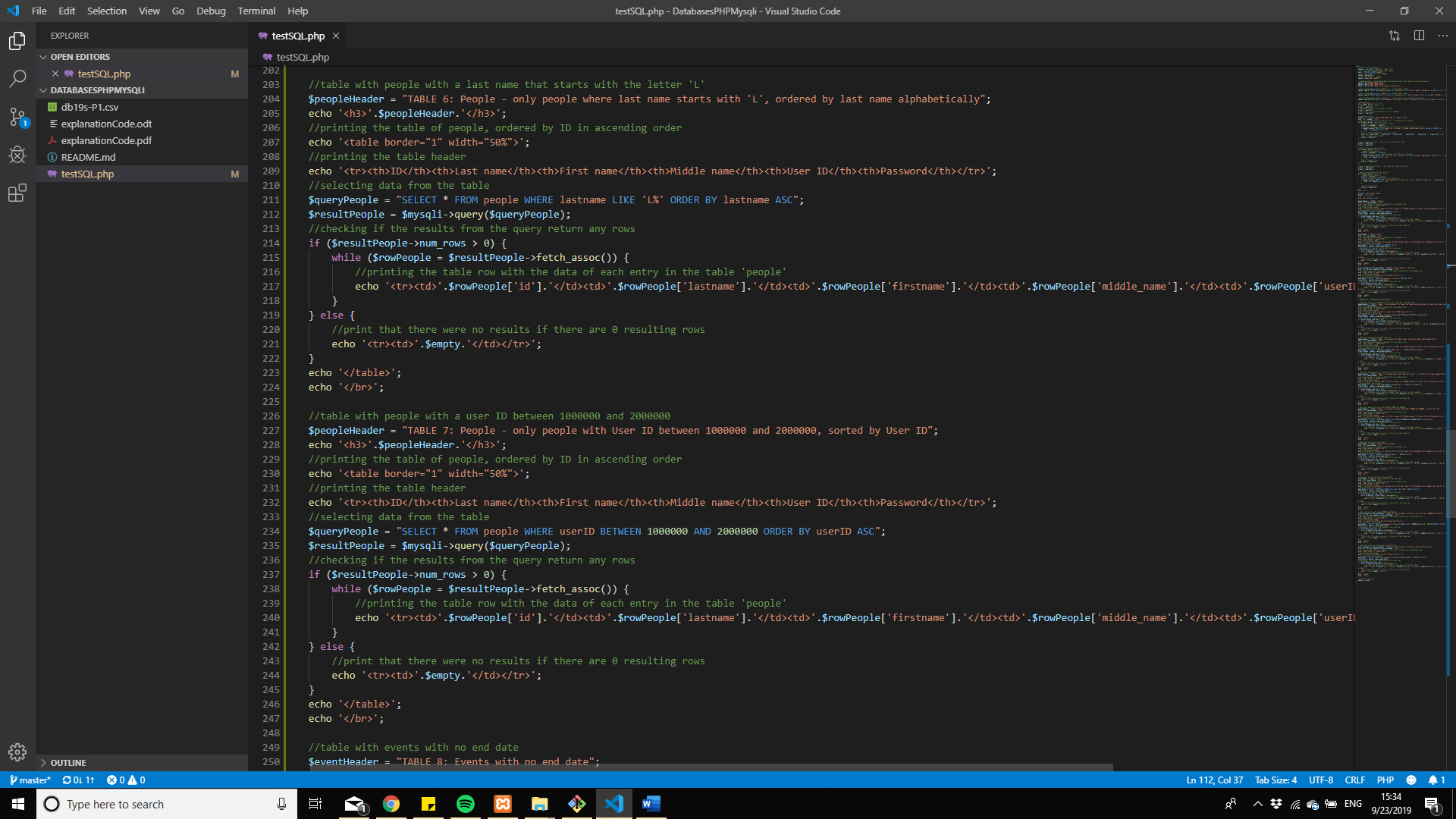


This query selects all entries from the table `people` with a `lastname` field beginning with the letter `L`, essentially returning all people with a last name starting with L, sorted by last name in alphabetical order. The `%` symbol means that the character `L` can be followed by anything.

A screenshot of a cell phone

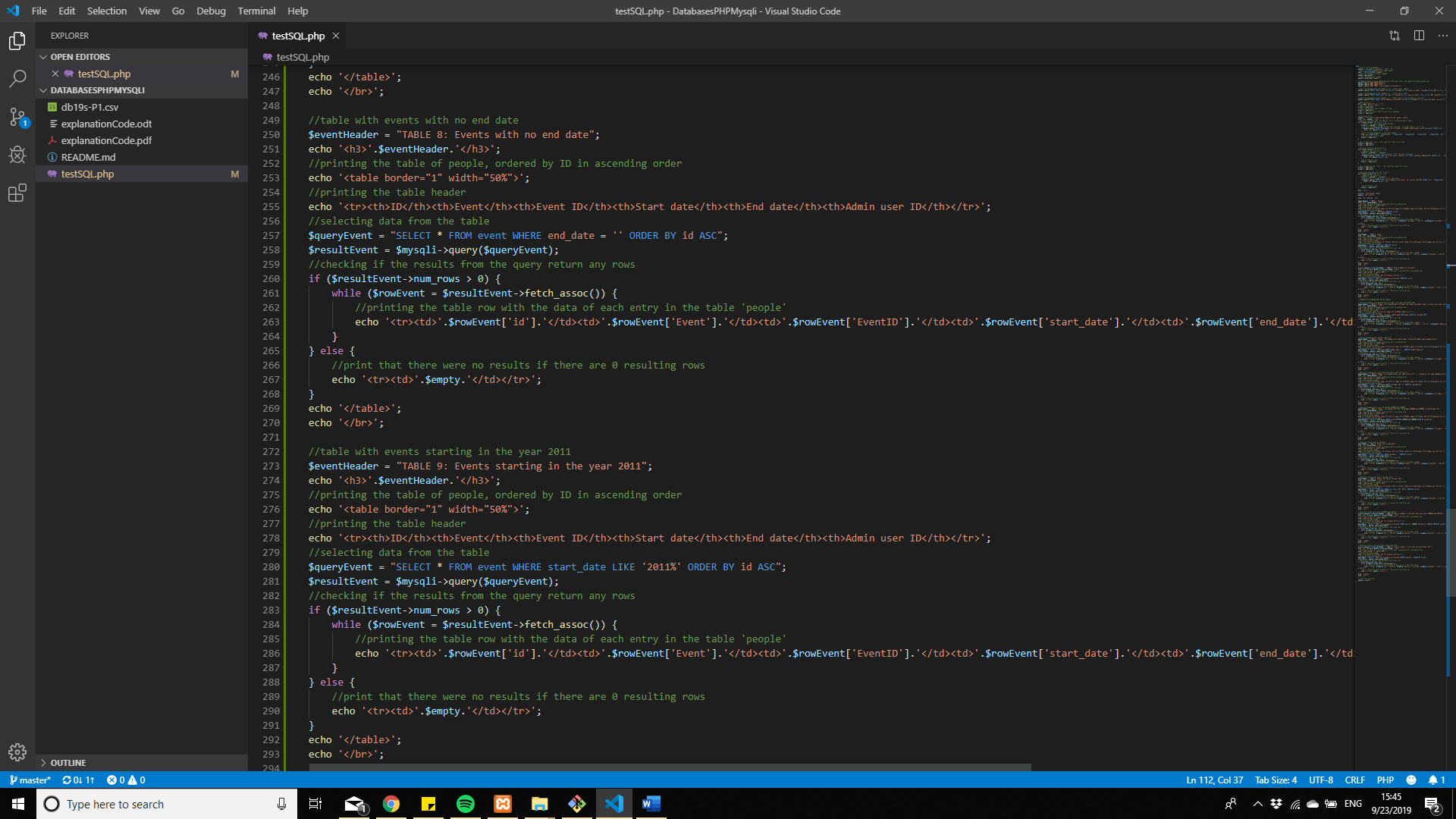
Description automatically generated

This third query below will select all entries from people with a user ID between 1000000 and 2000000, ordered by user ID in ascending order.



A screenshot of a cell phone

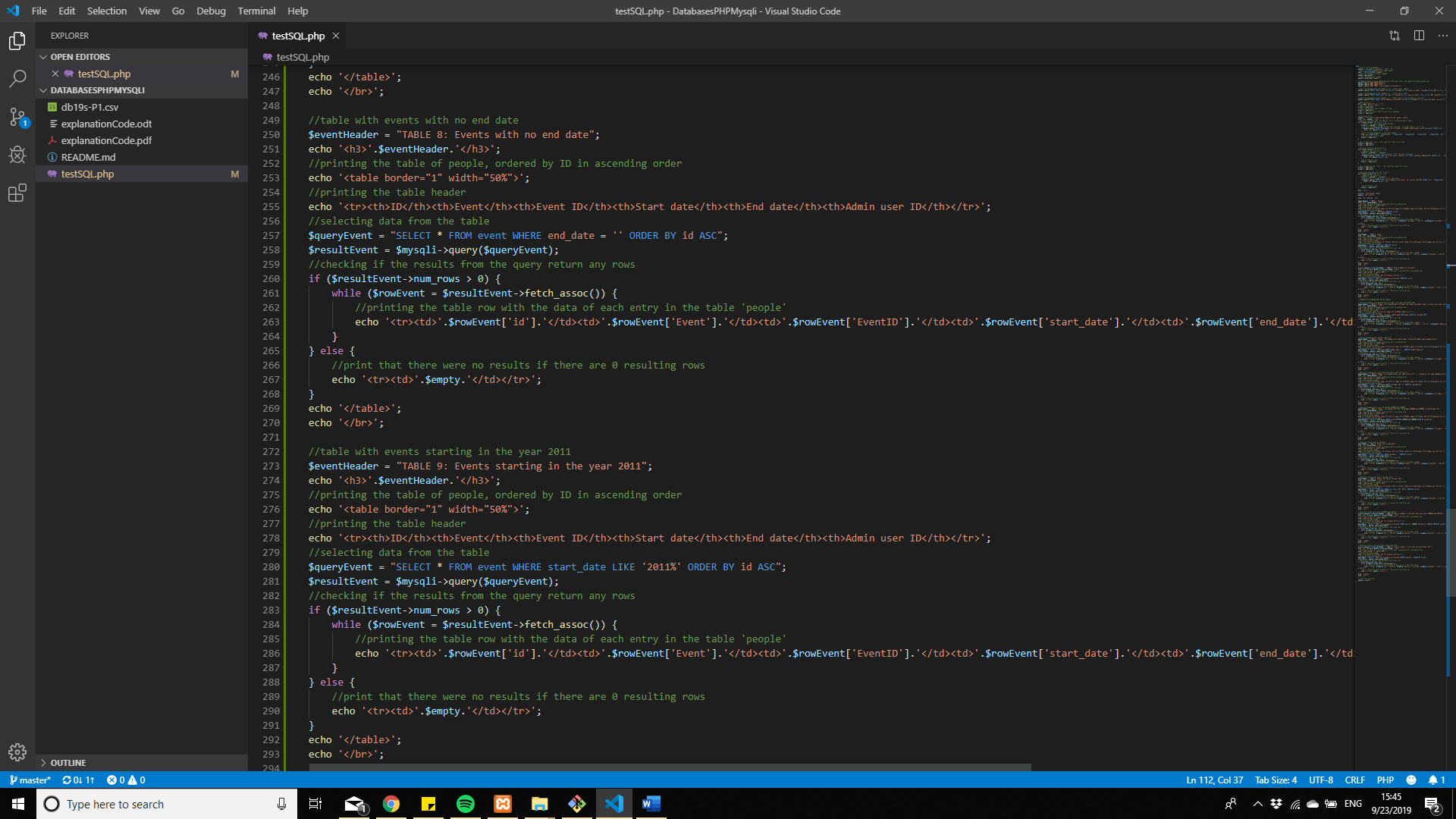
Description automatically generated



This query allows us to view all events with no end date, noted by entries where the `end\_date` field is empty, sorted by ID in ascending order.

A screenshot of a computer

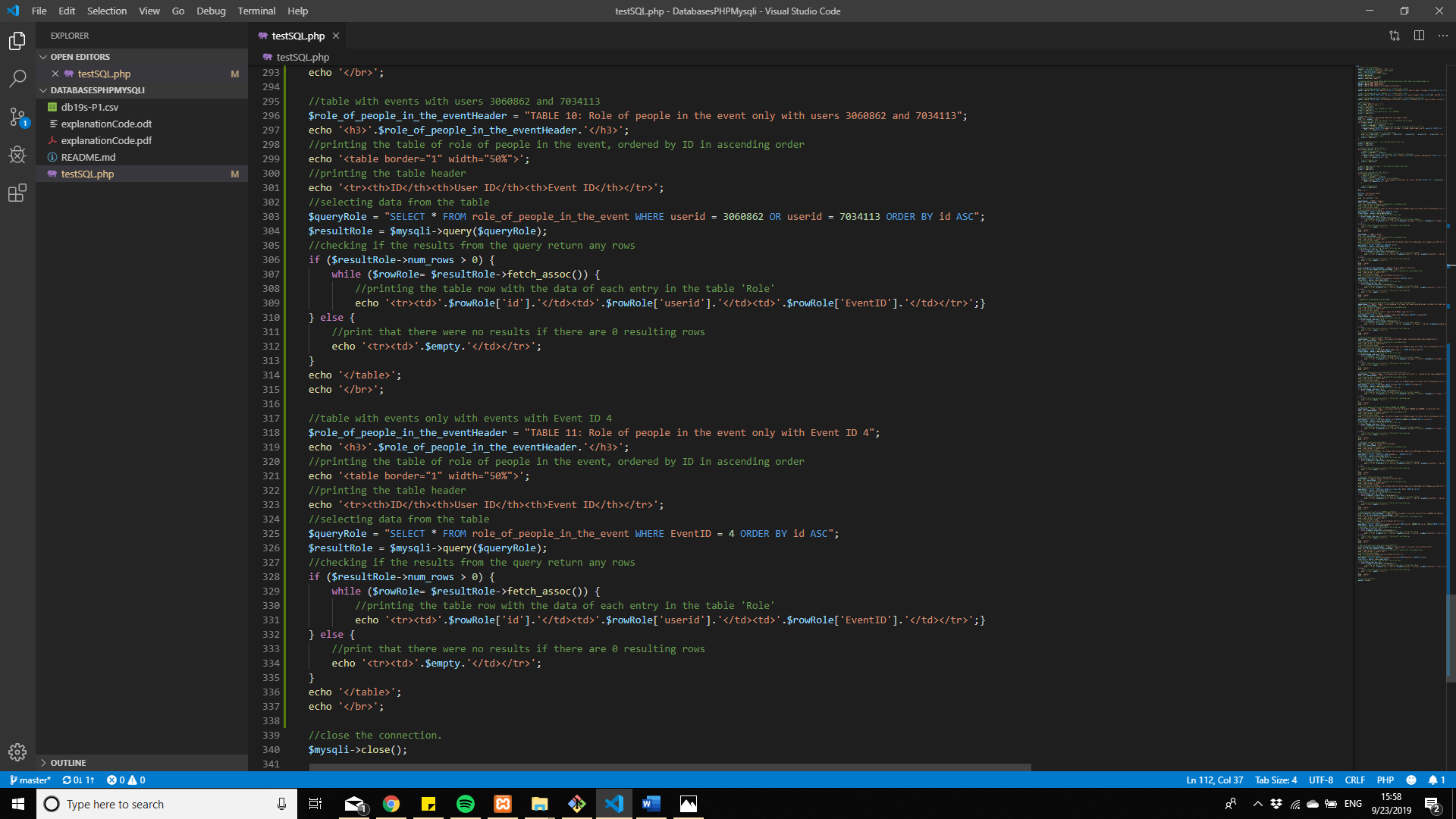
Description automatically generated



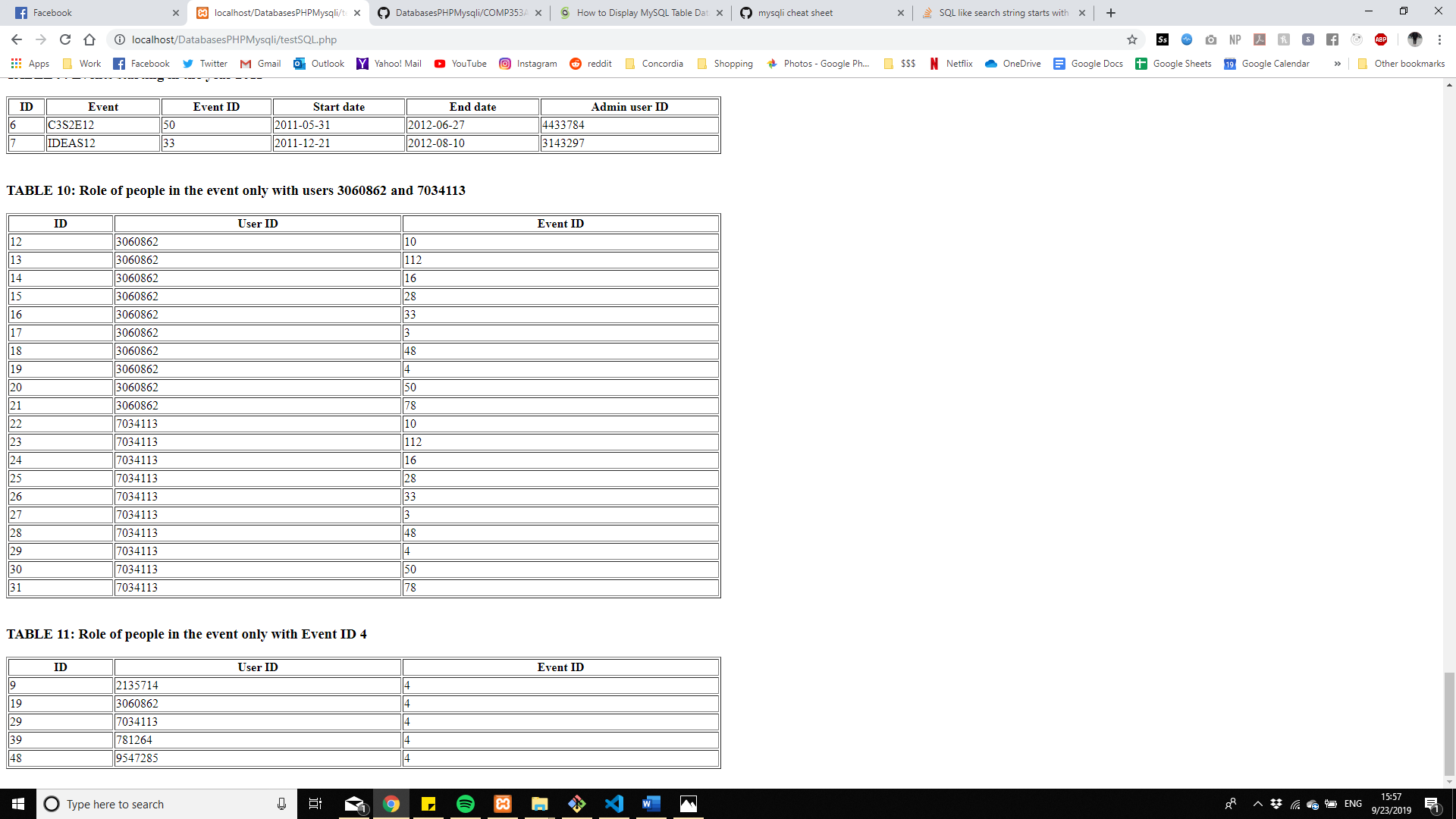
This query allows us to view all events where the start date is in 2011, noted by entries where the `start\_date` field begins with 2011 and is followed by anything else pertaining to the date, thanks to the `%` wildcard character, as mentioned above.

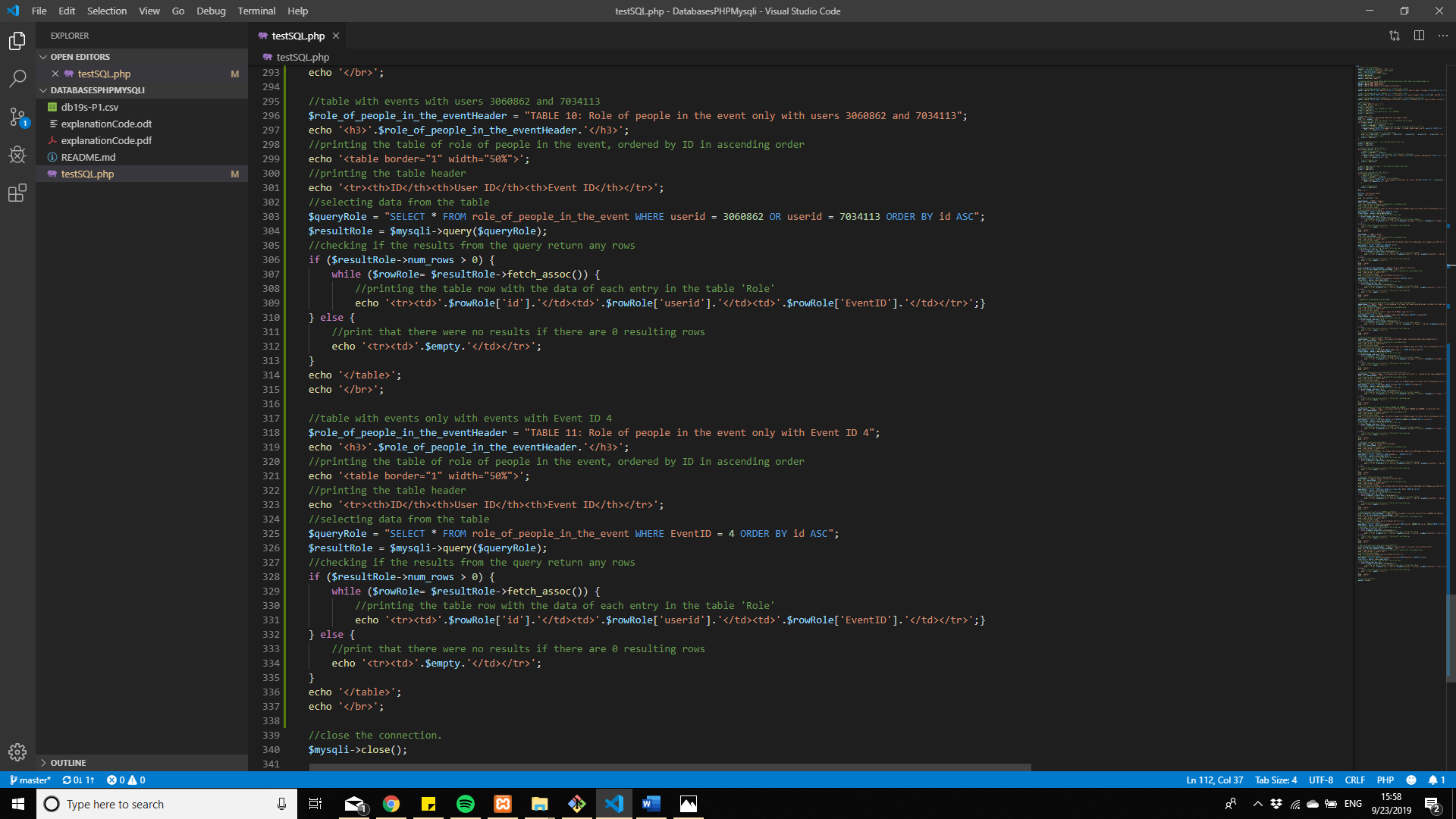
A screenshot of a computer

Description automatically generated

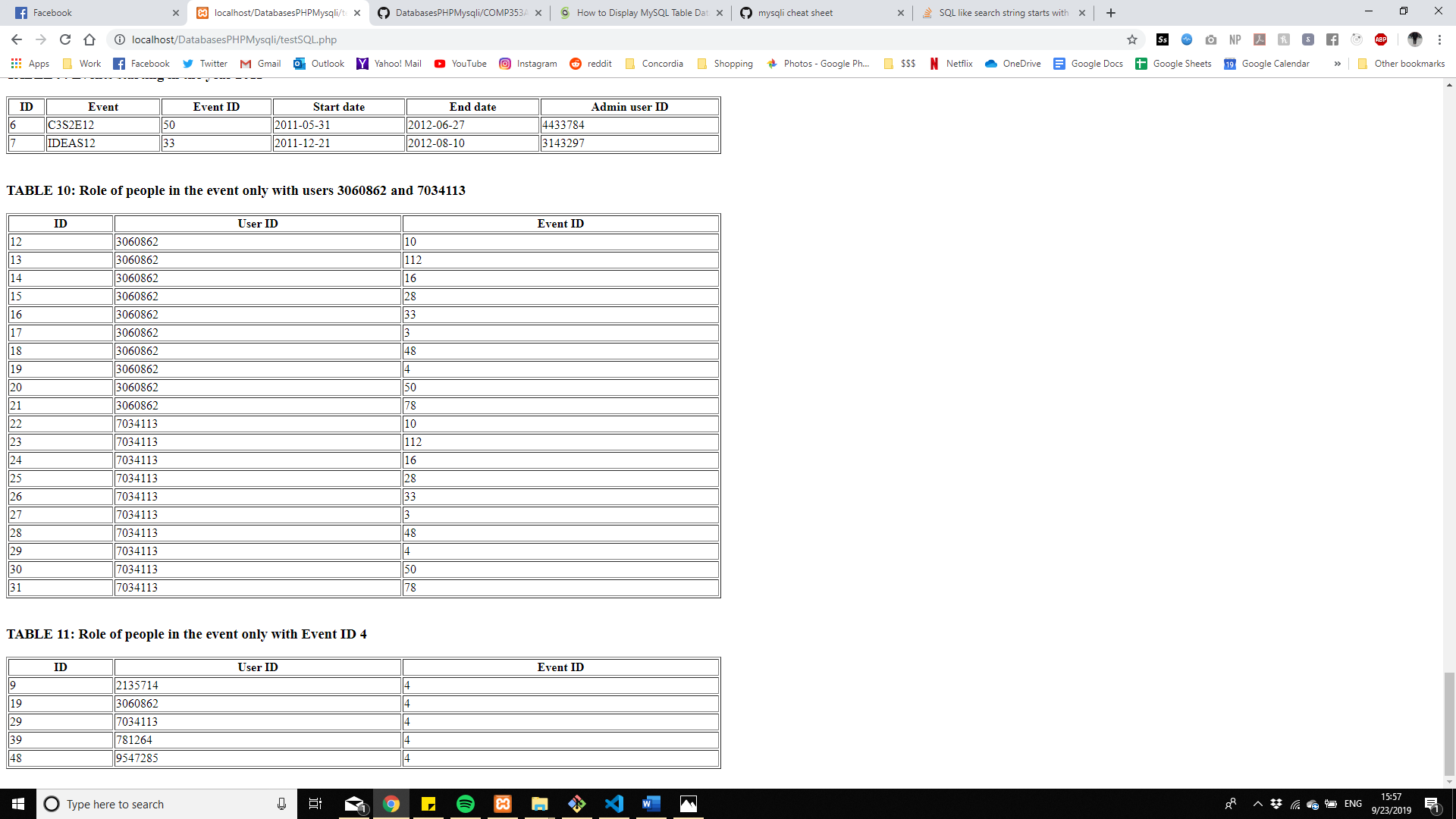


This query allows us to select all entries in the `role\_of\_people\_in\_the\_event` table with user IDs 3060862 or 7034113, thanks to the `OR` key word.

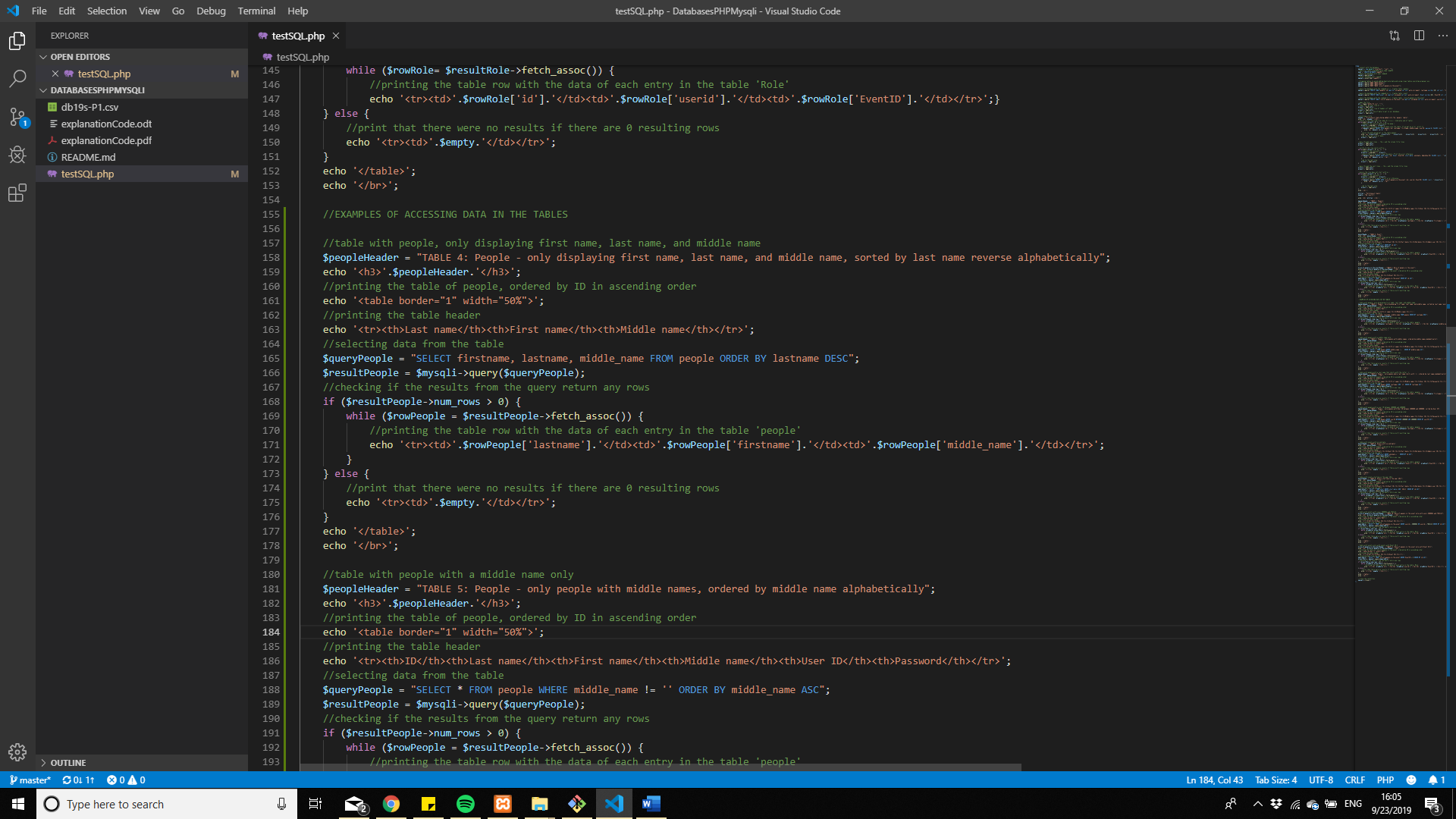




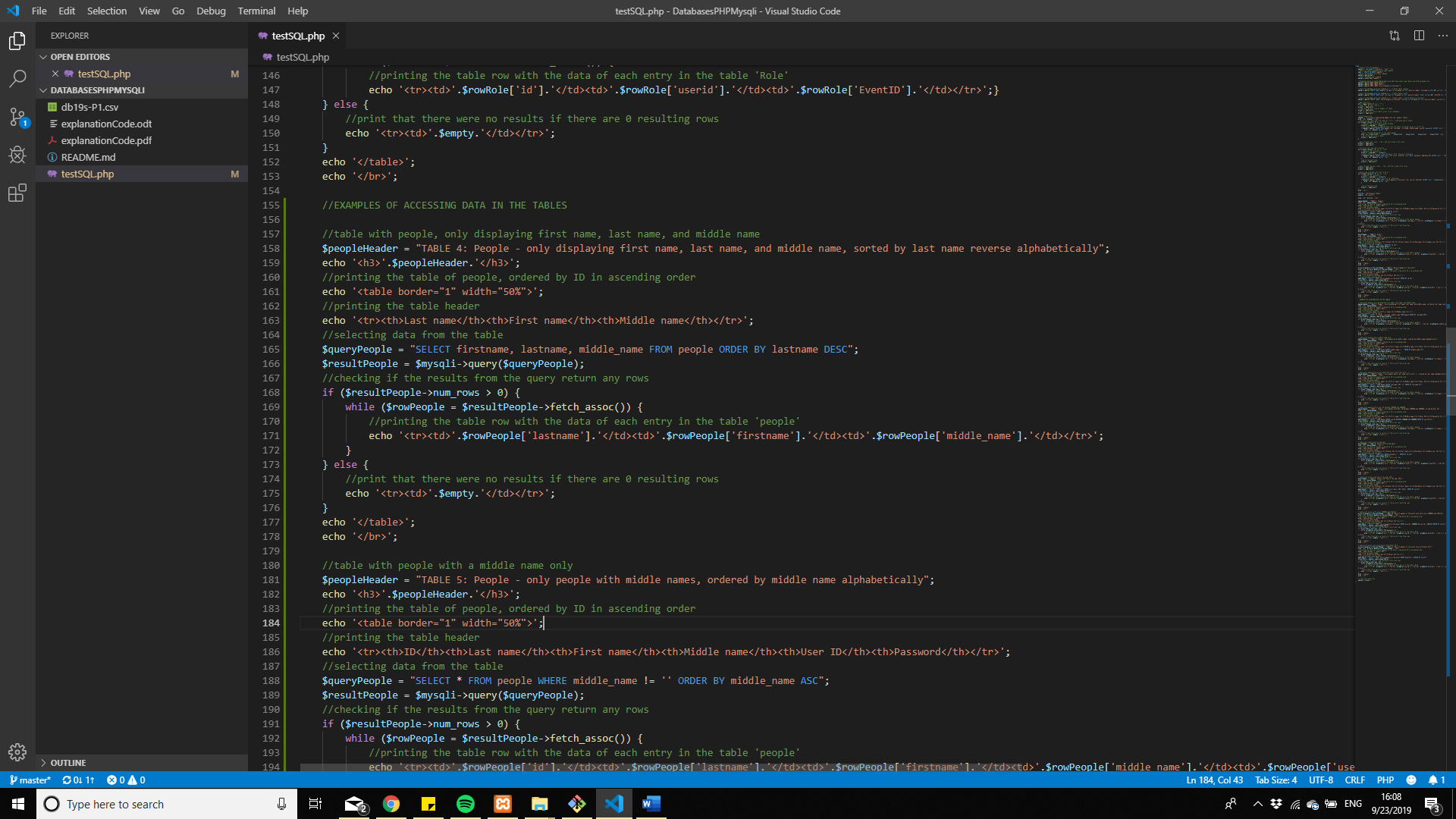
This last query selects all entries in `role\_of\_people\_in\_the\_event` where the `EventID` field is equal to 4.



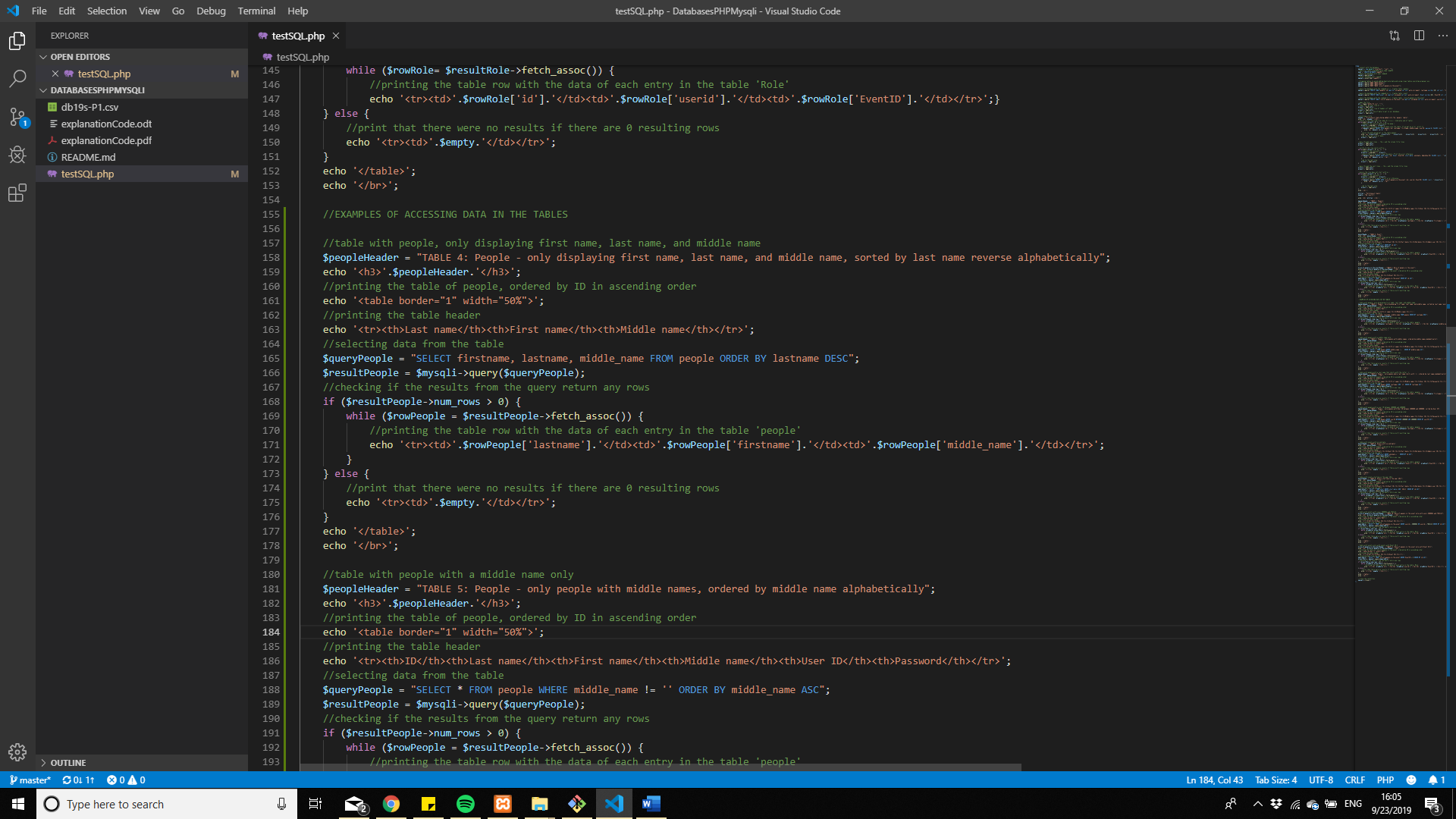
We can even modify the table itself and select only specific field names. If our header is only the following columns:



We can consequently select only the corresponding ones, being `firstname`, `lastname`, and `middle\_name`, sorting them by `lastname` in descending order:



And finally printing the corresponding smaller number of rows:



Giving us a final table of:

