**What I learned from SQL Essential Training**

# **How many customers purchased two songs at $0.99 each?**

**/\***

**Created by: Sayed Muqim Noorani**

**Dated Created: 26/04/2023**

**Description: This query displays the number of customers who purchased two songs at $0.99 each, totalling $1.98. This includes Invoice date,**

**billing address, Billing City and total paid. This is ordered by the invoice date.**

**\*/**

**SELECT**

**InvoiceDate,**

**BillingAddress,**

**BillingCity,**

**Total**

**FROM**

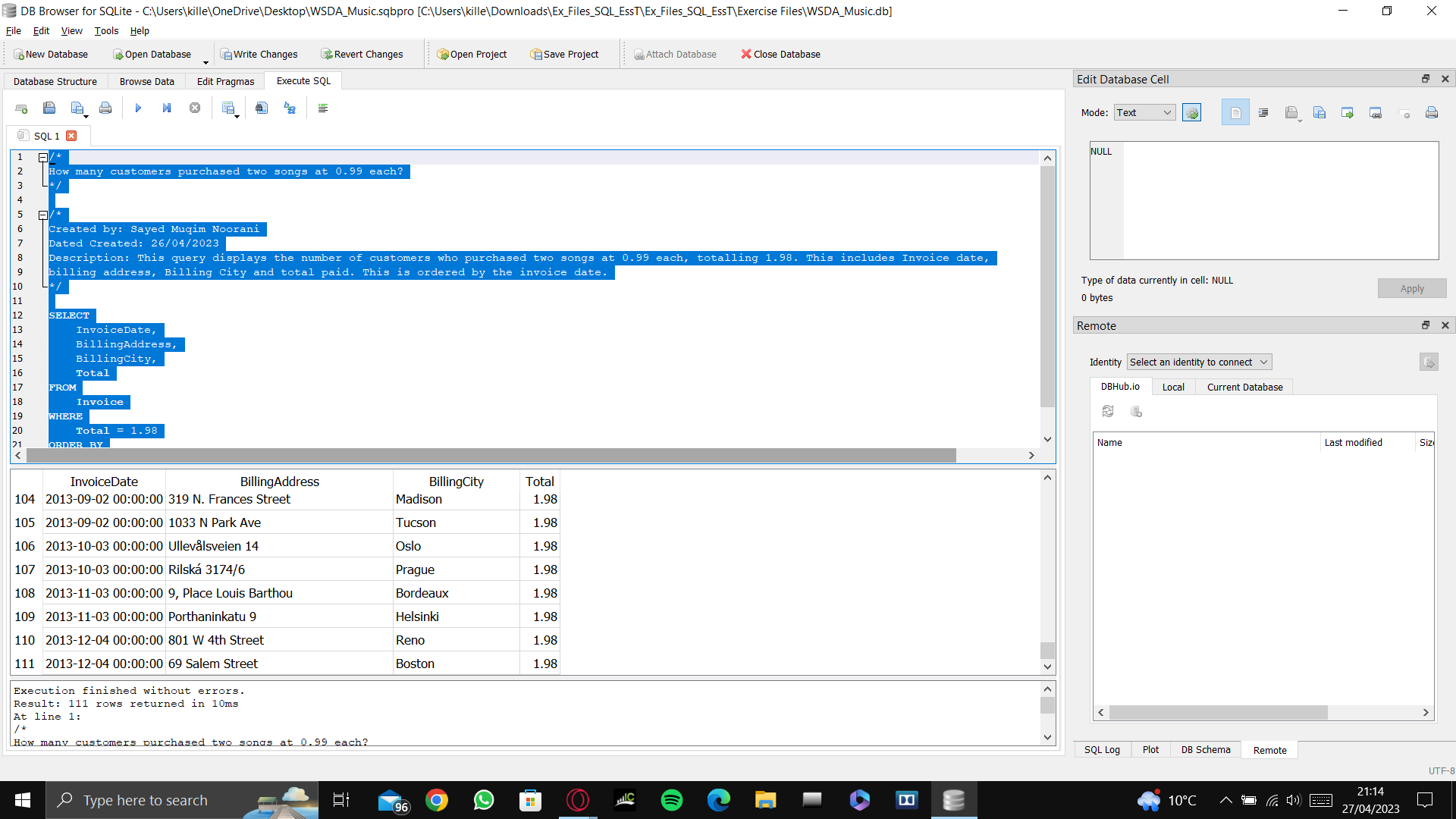
**Invoice**

**WHERE**

**Total = 1.98**

**ORDER BY**

**InvoiceDate**



So 111 customers purchased two songs at $0.99 each.

# **How many invoices exist between $1.98 and $5?**

**/\***

**Created by: Sayed Muqim Noorani**

**Dated Created: 26/04/2023**

**Description: How many invoices exist between $1.98 and $5?**

**\*/**

**SELECT**

**InvoiceDate,**

**BillingAddress,**

**BillingCity,**

**Total**

**FROM**

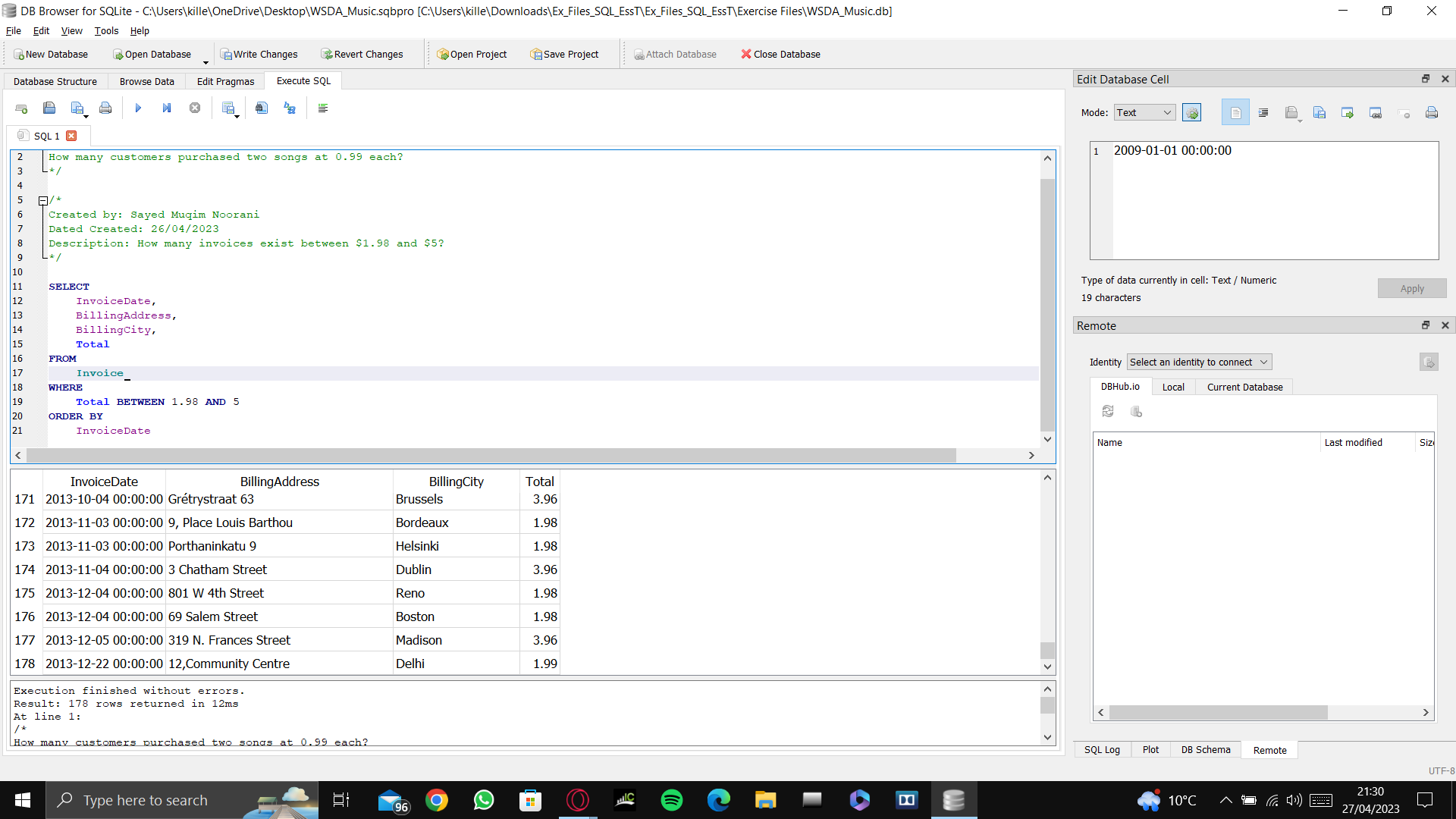
**Invoice**

**WHERE**

**Total BETWEEN 1.98 AND 5**

**ORDER BY**

**InvoiceDate**



So the result shows that there are 178 invoices that total between $1.98 and $3.96.

# **How many invoices do we have that are exactly $1.98 or $3.96?**

**SELECT**

**InvoiceDate,**

**BillingAddress,**

**BillingCity,**

**Total**

**FROM**

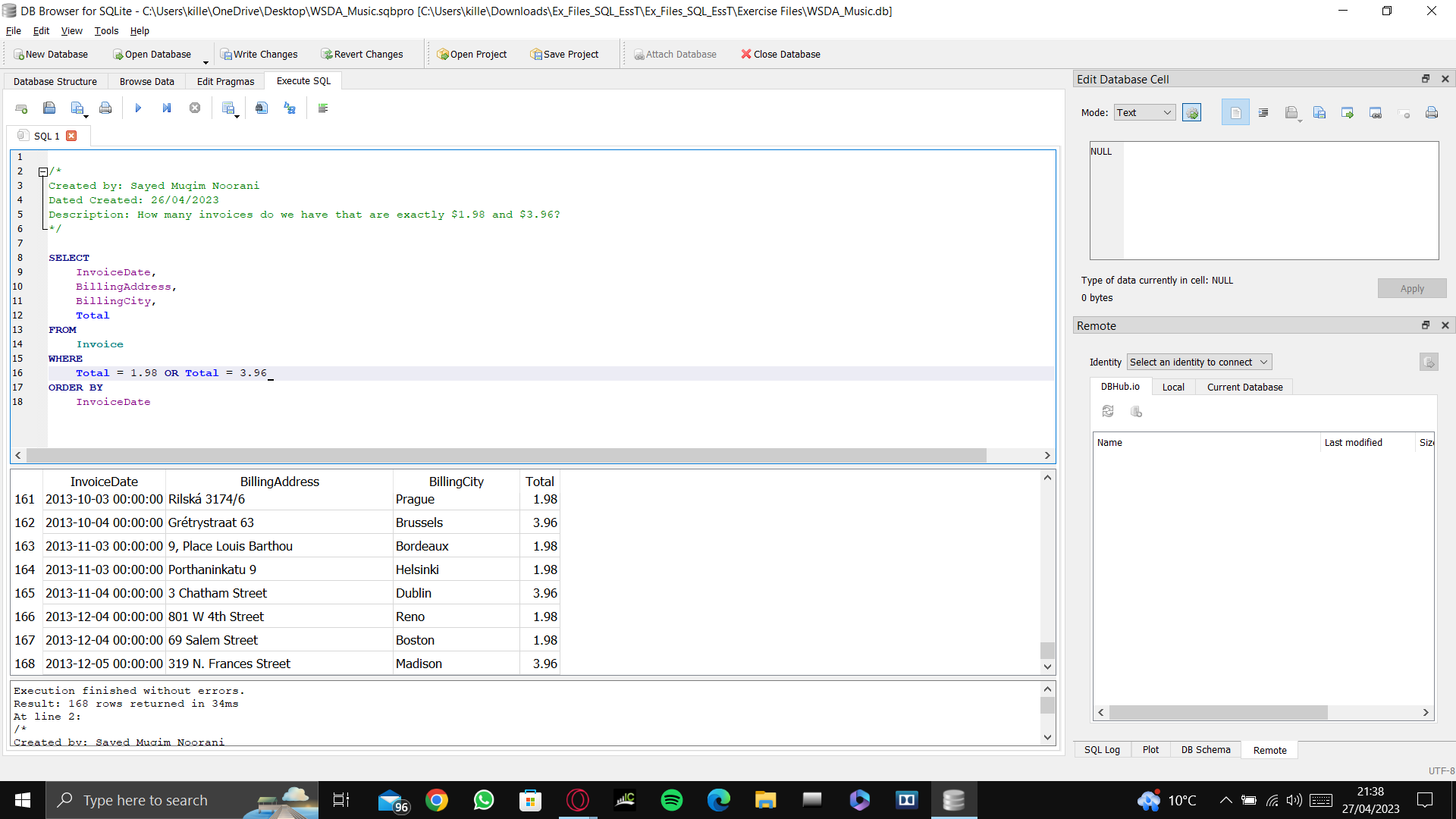
**Invoice**

**WHERE**

**Total = 1.98 OR Total = 3.96**

**ORDER BY**

**InvoiceDate**



**SELECT**

**InvoiceDate,**

**BillingAddress,**

**BillingCity,**

**Total**

**FROM**

**Invoice**

**WHERE**

**Total IN(1.98,3.96)**

**ORDER BY**

**InvoiceDate**

This is a much simpler way of writing the SQL query. Inserting the Logical Function 'IN' and brackets (parenthesis), and inside the brackets add the two totals that you are looking for, which in this example are either 1.98 OR 3.96.

# **How many invoices were billed to Brussels?**

**SELECT**

**InvoiceDate,**

**BillingAddress,**

**BillingCity,**

**Total**

**FROM**

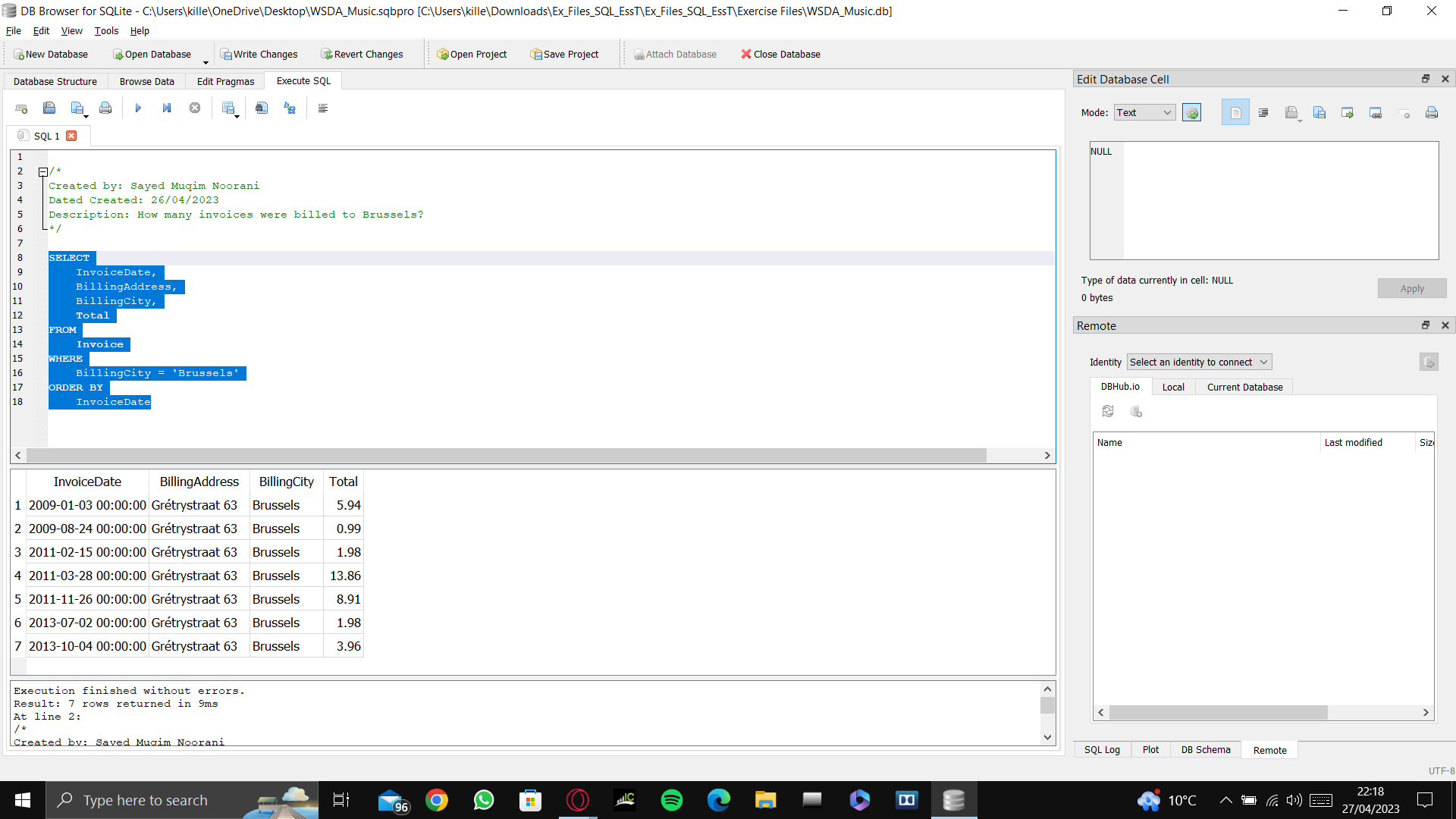
**Invoice**

**WHERE**

**BillingCity = 'Brussels'**

**ORDER BY**

**InvoiceDate**



As we can see from the query results above, the number of invoices that were billed to Brussels, is 7. And in contrast to numerical data, text data must be written in between single quotation marks (‘...’), for example, ‘Brussels’.

# **How many invoices were billed to Brussels, Orlando or Paris?**

**SELECT**

**InvoiceDate,**

**BillingAddress,**

**BillingCity,**

**Total**

**FROM**

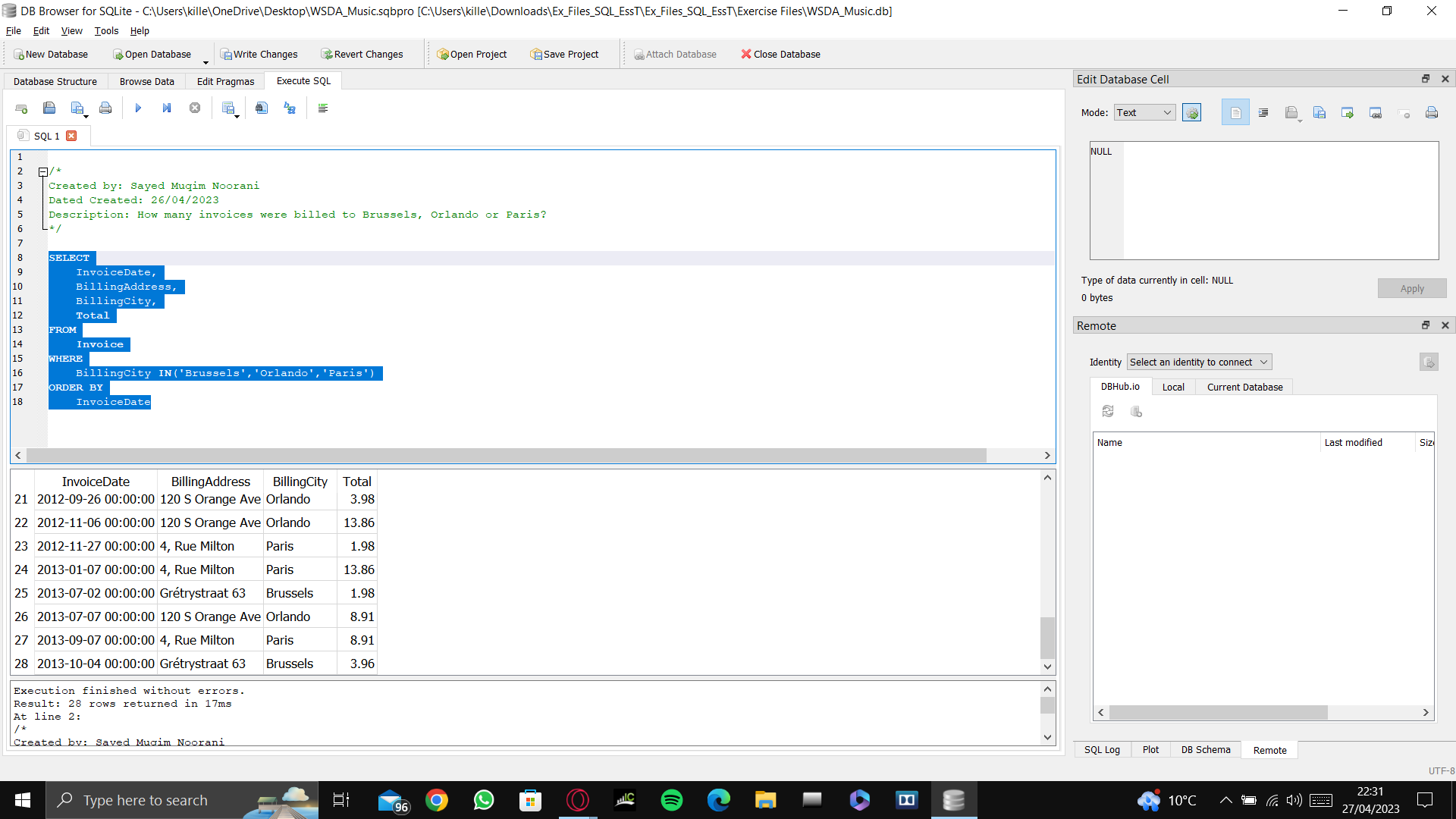
**Invoice**

**WHERE**

**BillingCity IN('Brussels','Orlando','Paris')**

**ORDER BY**

**InvoiceDate**



As can be seen in the results above, there are 28 invoices that were billed to Brussels, Orlando, or Paris. As before, the **IN** operator was used to find the invoices of the three cities, and inside the brackets, the names of the three cities were inserted and each of them were in single quotation marks, separated by commas **IN** (... , … ,...).

# **How many invoices were billed in cities that begin with the letter B?**

**SELECT**

**InvoiceDate,**

**BillingAddress,**

**BillingCity,**

**Total**

**FROM**

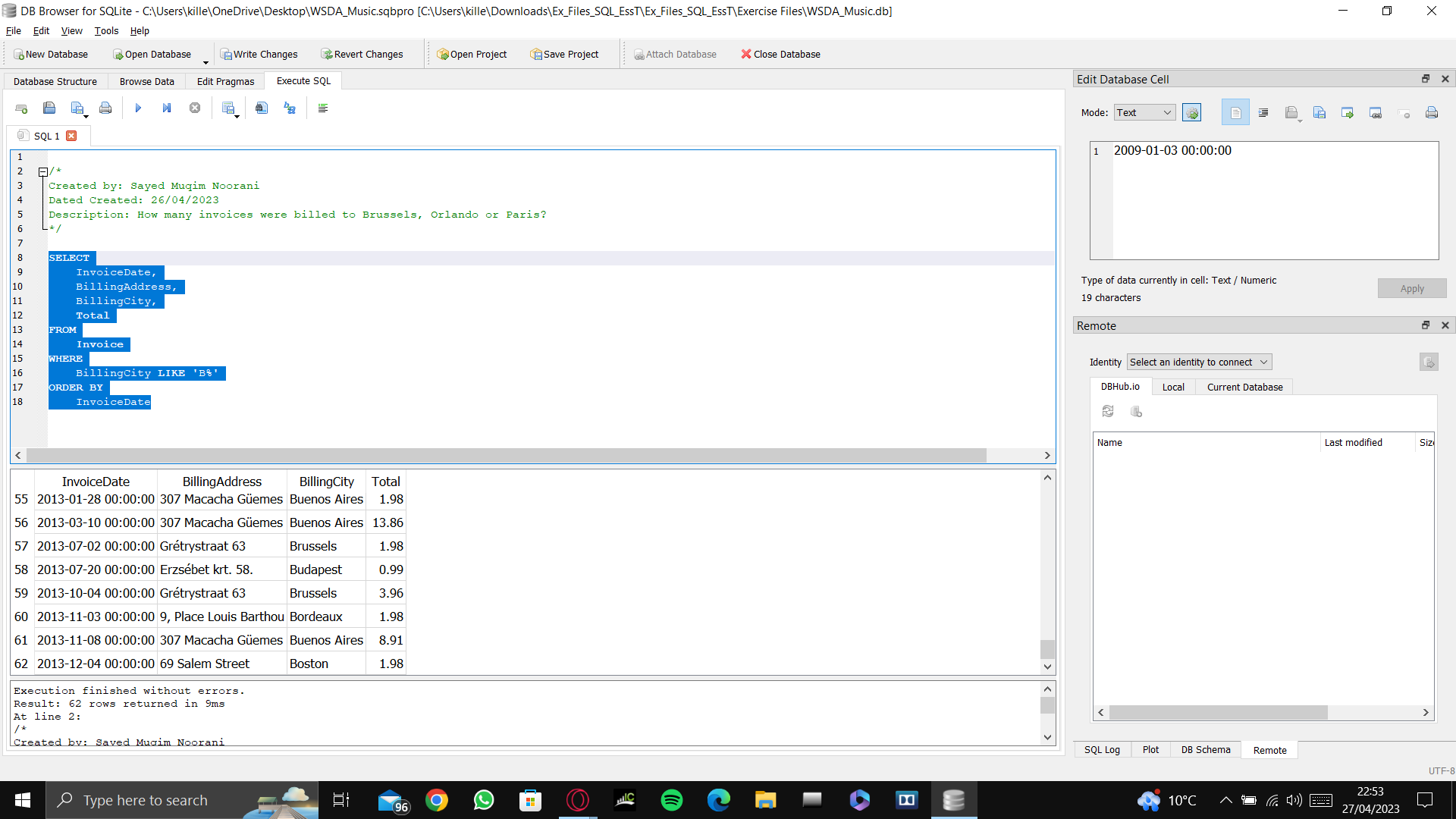
**Invoice**

**WHERE**

**BillingCity LIKE 'B%'**

**ORDER BY**

**InvoiceDate**



As can be seen above, the database shows that there were 62 invoices that were billed in cities that begin with the letter B. The % symbol after the letter B in the query is to specify that we want all cities that begin with the letter B, otherwise the query will search only for the letter B, and there will be no results.

# **How many invoices were billed in cities that have the letter B anywhere in its name?**

**SELECT**

**InvoiceDate,**

**BillingAddress,**

**BillingCity,**

**Total**

**FROM**

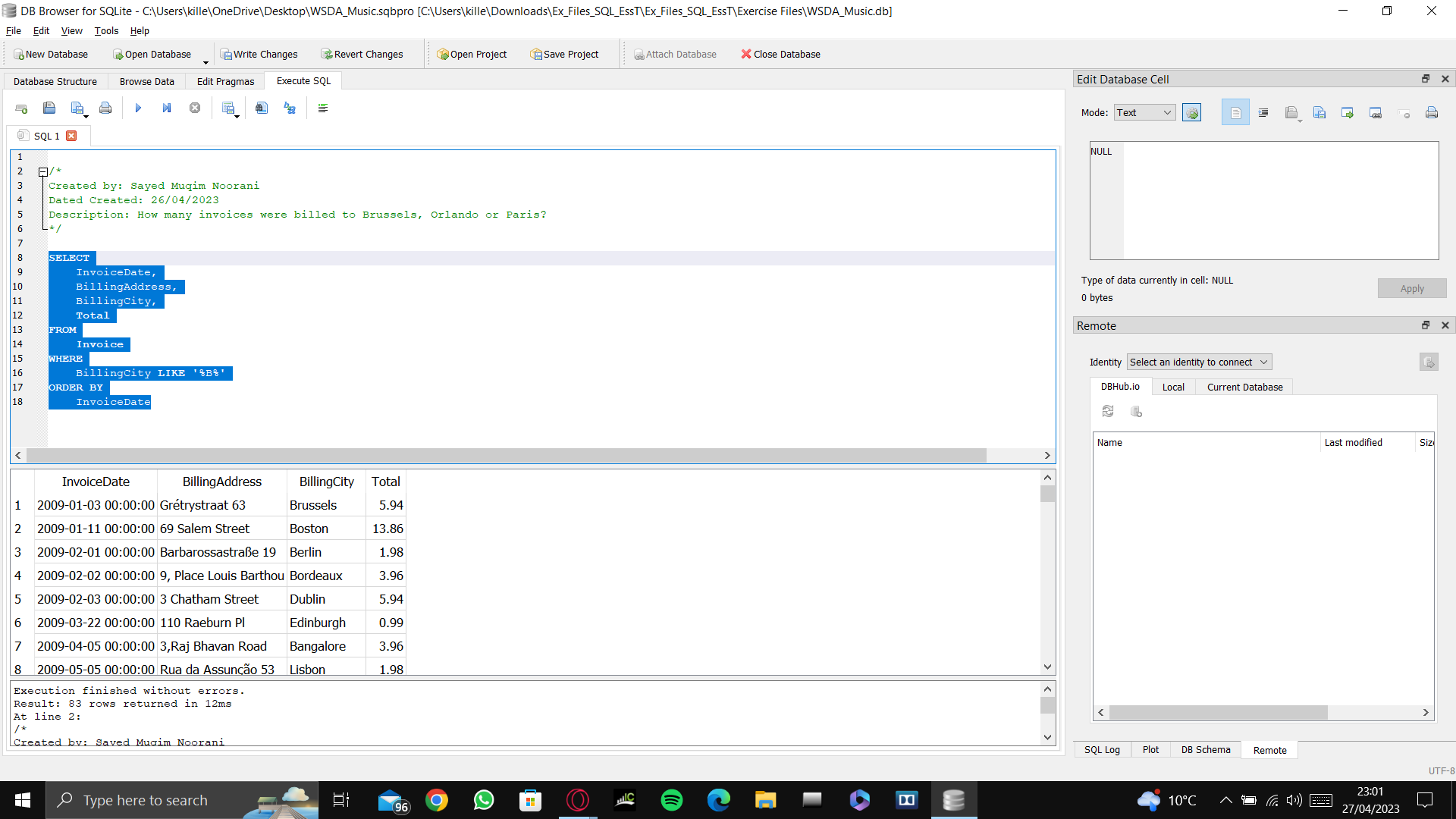
**Invoice**

**WHERE**

**BillingCity LIKE '%B%'**

**ORDER BY**

**InvoiceDate**



As can be seen above, there were 83 invoices that were billed in cities that have B anywhere in its name. As before, the **LIKE** operator was used to search for text data. This time, the % symbol was place both before and after the letter B, to specify that we don’t care what comes before or after the letter B, as long as the letter B is in the name.

# **How many invoices were billed on 2010-05-22 00:00:00?**

**SELECT**

**InvoiceDate,**

**BillingAddress,**

**BillingCity,**

**Total**

**FROM**

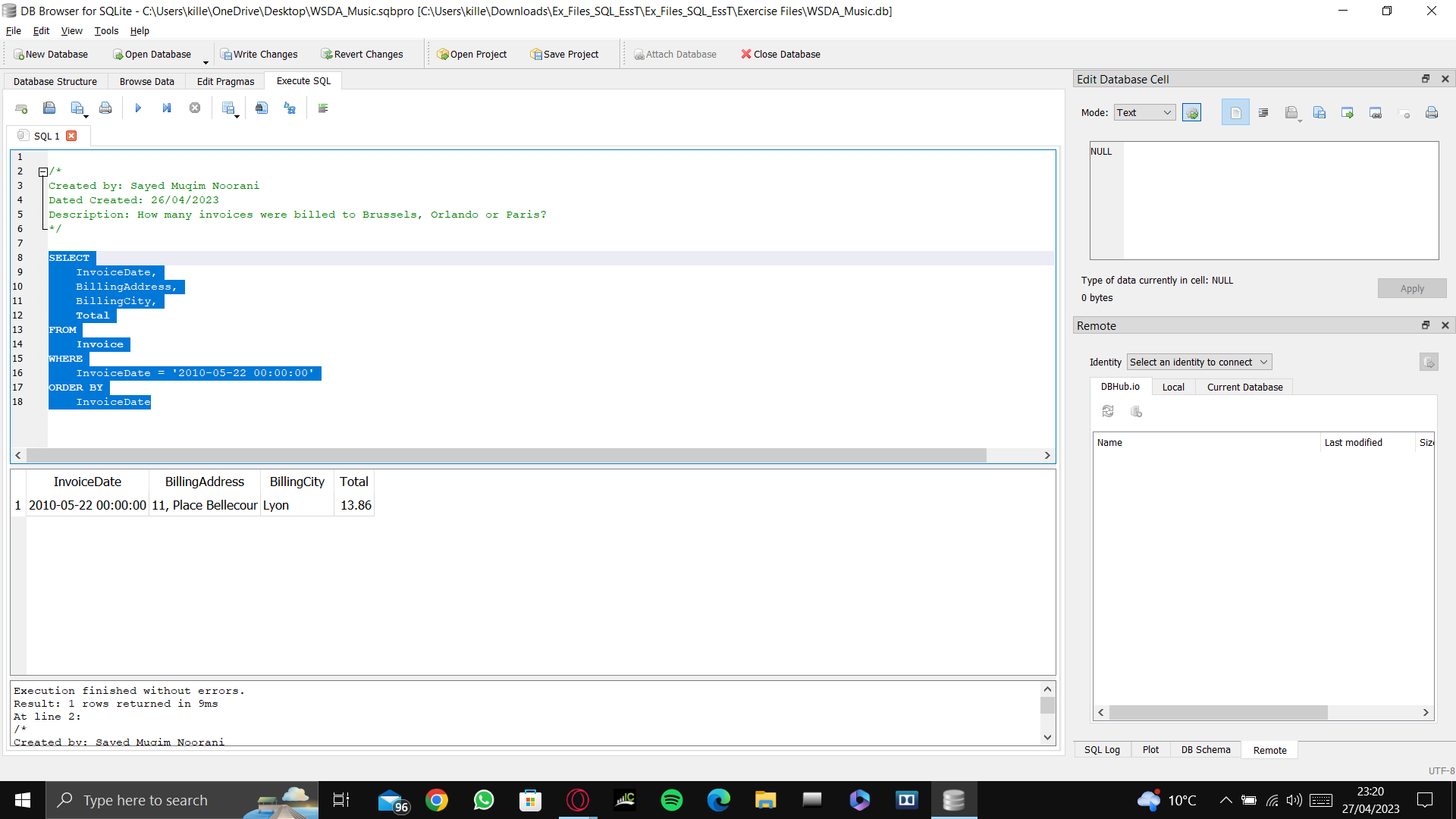
**Invoice**

**WHERE**

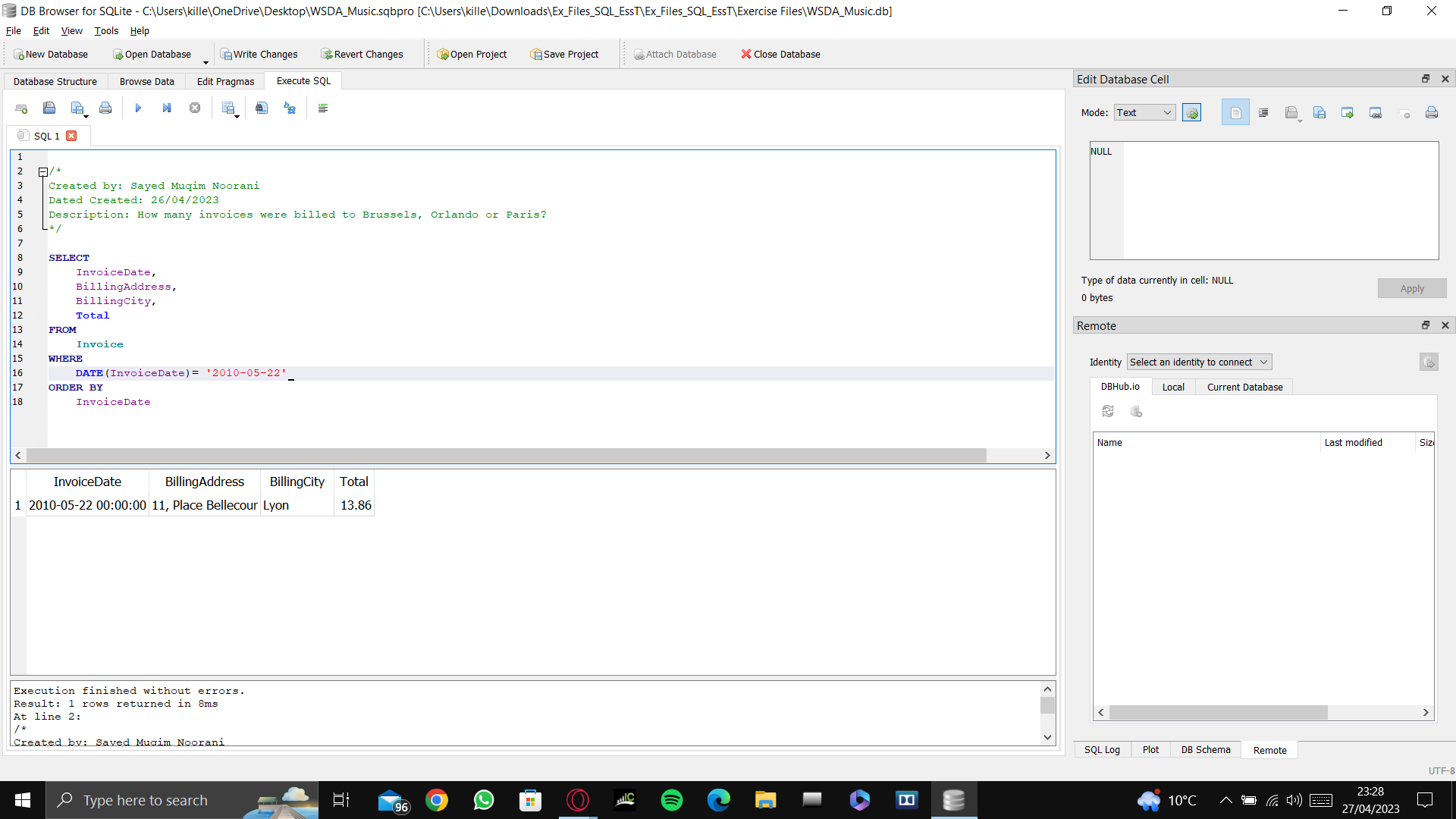
**InvoiceDate = '2010-05-22 00:00:00'**

**ORDER BY**

**InvoiceDate**

****

As can be seen above, the number of invoices that were billed on 2010-05-22 at 00:00:00 is just one invoice. Again, as with text data, dates must also be written within single quotation marks. However, we can use the **DATE** function in order to get rid of the need to write out the time in this case. An example is below:

 Written as DATE(InvoiceDate)=

# **Get all invoices after 2010-05-22 which totalled less than $3.00**

**SELECT**

**InvoiceDate,**

**BillingAddress,**

**BillingCity,**

**Total**

**FROM**

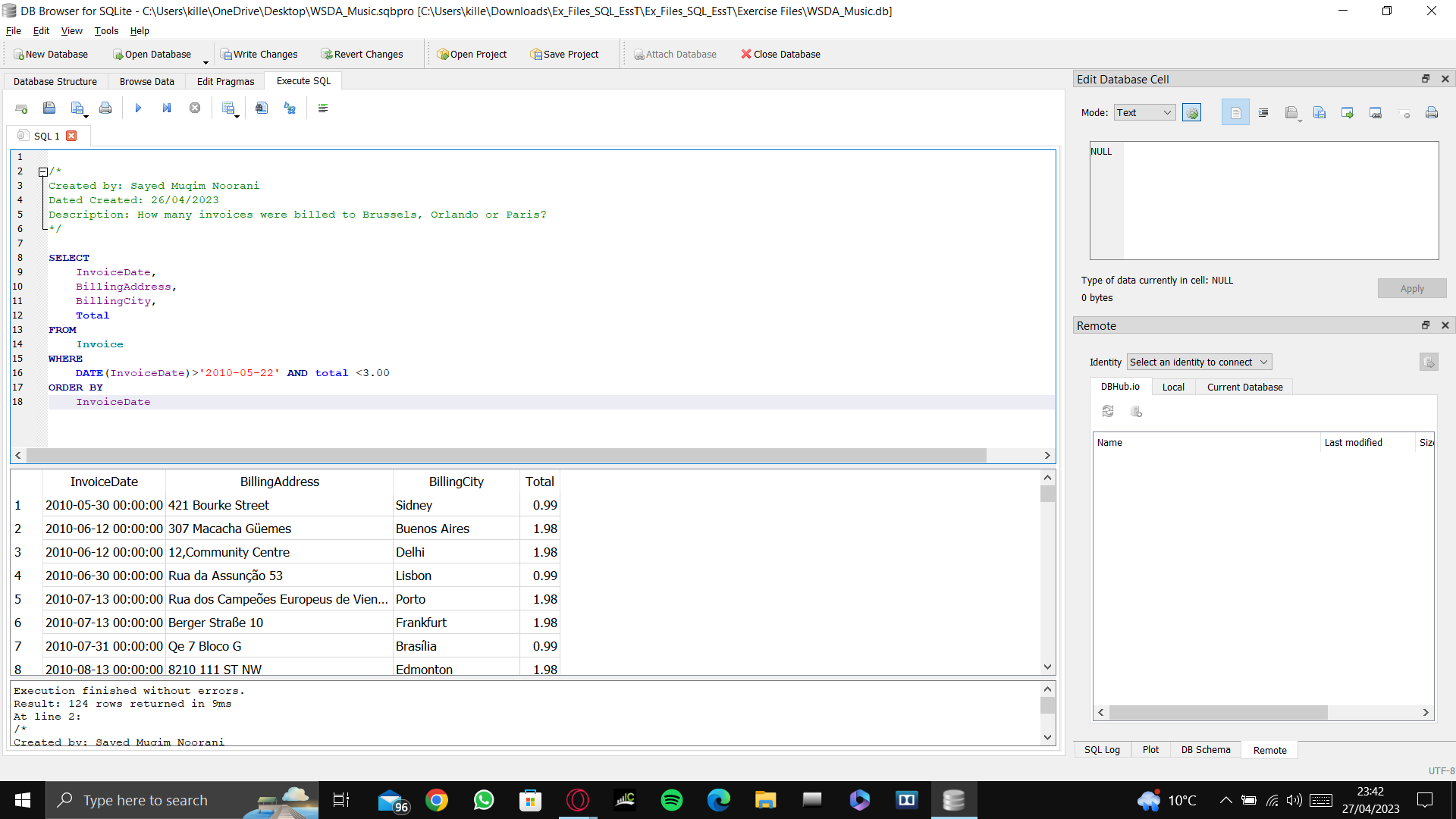
**Invoice**

**WHERE**

**DATE(InvoiceDate)>'2010-05-22' AND total <3.00**

**ORDER BY**

**InvoiceDate**



As can be seen above, the first date that shows up after we run the query is 2010-05-30, which is definitely after 2010-05-22, and the dates are in ascending order, whereby the earliest dates are at the top of the results and the latest dates are at the bottom of the list of results. Also, all the totals of the invoices are lower than $3.00, which is what we needed.

# **Get all invoices where billing cities begin with the letter P or the letter D**

**SELECT**

**InvoiceDate,**

**BillingAddress,**

**BillingCity,**

**Total**

**FROM**

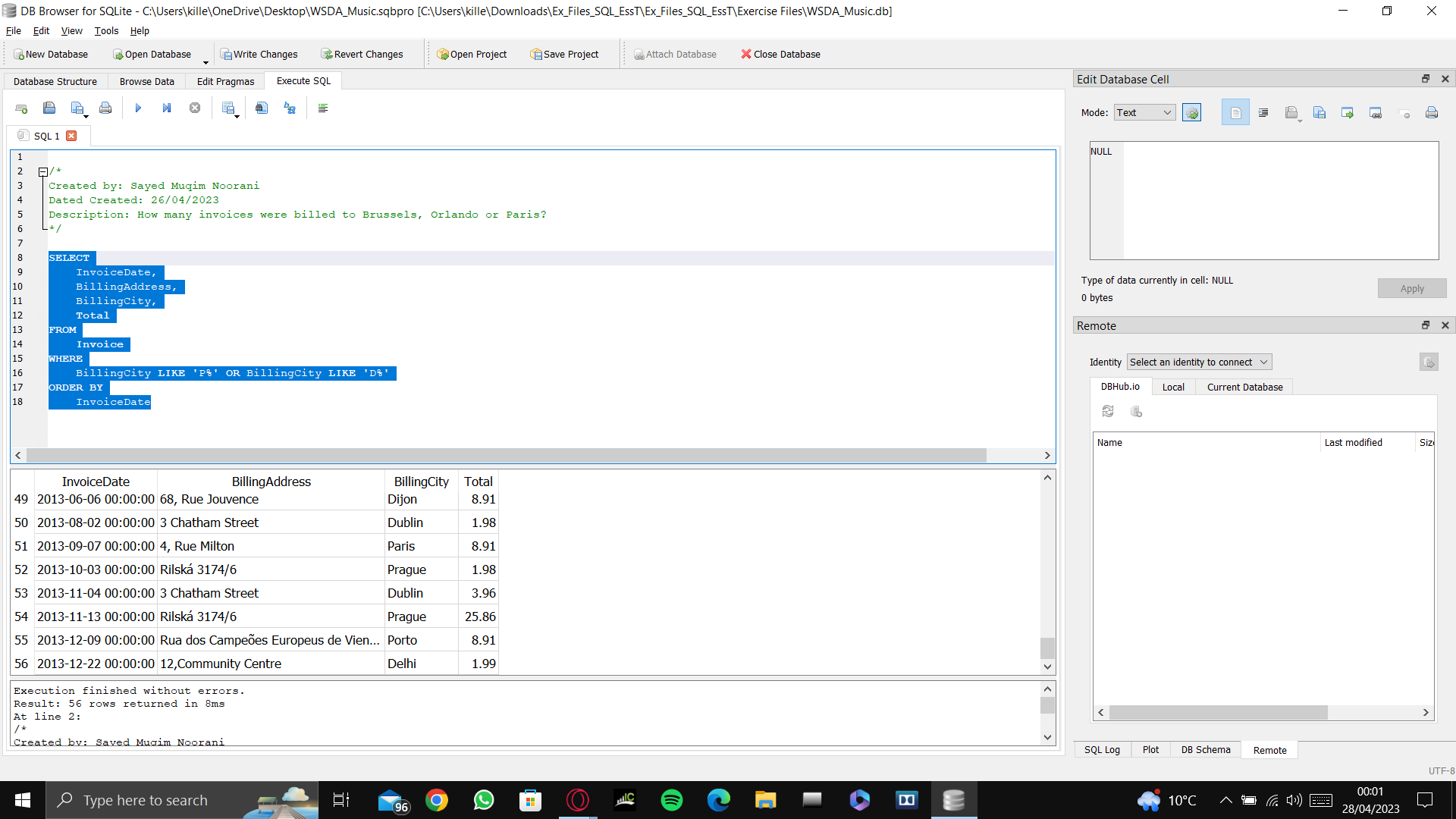
**Invoice**

**WHERE**

**BillingCity LIKE 'P%' OR BillingCity LIKE 'D%'**

**ORDER BY**

**InvoiceDate**



As can be seen above, the number of cities that begin with the letter P or the letter D is 56, and the results above also show that they begin with either P or D.

# **Get all invoices that are greater than $1.98 from any cities whose names begin with the letter P or the letter D**

**SELECT**

**InvoiceDate,**

**BillingAddress,**

**BillingCity,**

**Total**

**FROM**

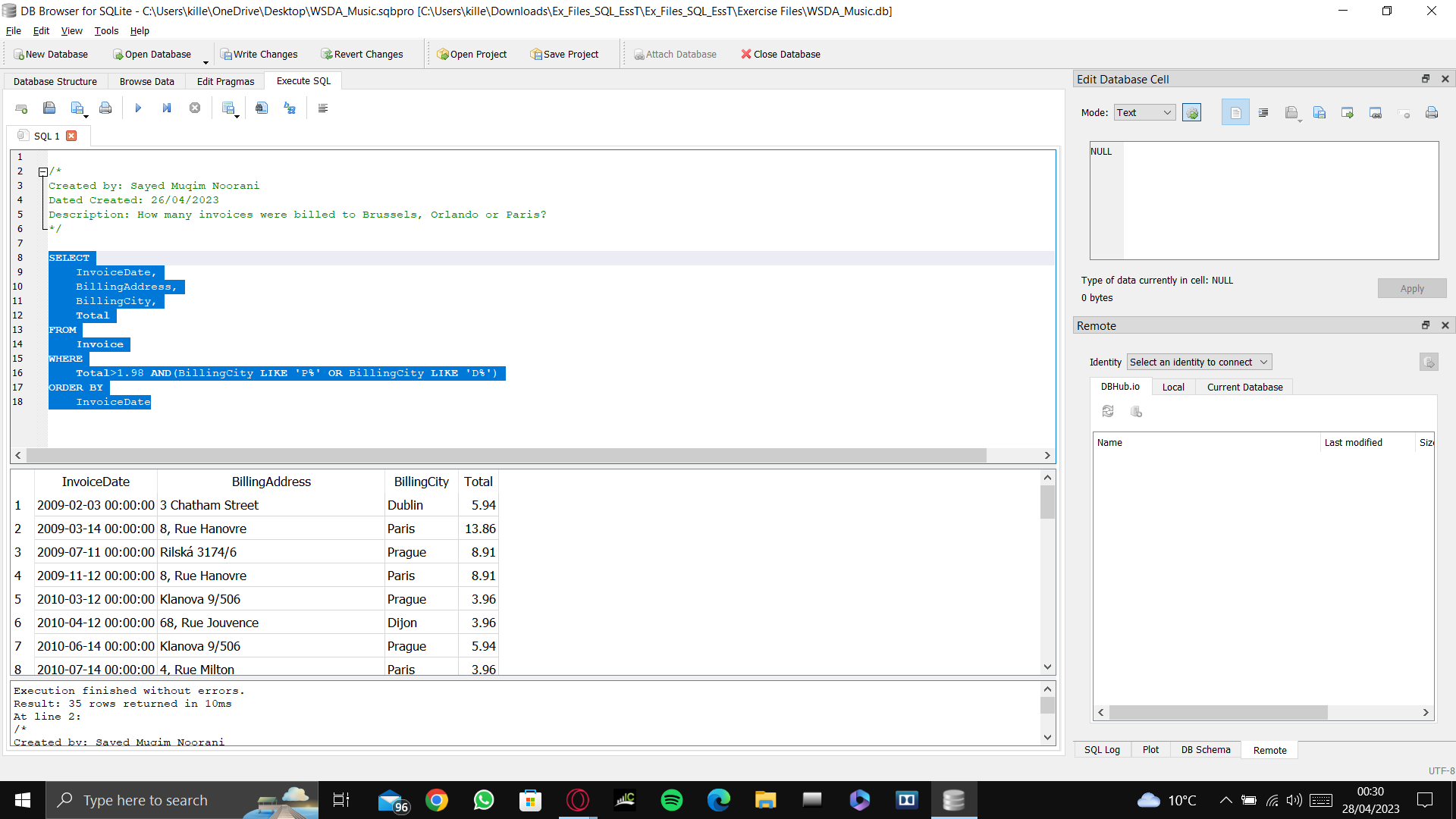
**Invoice**

**WHERE**

**Total>1.98 AND (BillingCity LIKE 'P%' OR BillingCity LIKE 'D%')**

**ORDER BY**

**InvoiceDate**



Using the acronym BEMDAS (Brackets, Exponent, Multiplication, Division, Addition, Subtraction) the query was asking the database to draw info where the total of the invoices were above 1.98 and the billing cities began with either P or D. As can be seen above, the results show that there are 35 invoices that total over 1.98 in cities where the names of the cities begin with either P or D. Utilising BEMDAS, the Total>1.98 was written first, followed by the **AND** clause. Brackets had to then be utilised to make all that is written inside it like one function. Thus we wrote **(BillingCity LIKE ‘P%’ OR Billing City LIKE ‘D%’)**. If this had not been done, we would have received results that included invoices where the total was greater than 1.98 and the billing cities beginning with the letter P. However, the error would have been the fact that the results would also show cities whose names begin with D and it would also show invoice totals that were lower than 1.98, which would be incorrect results. Thus, we had to use brackets in the query to tell the query to process what is inside the brackets first and then what is outside the brackets, which would yield us the results that we wanted.

# **WSDA Music Sales Goal**

WSDA now has a sales goal of having as many customers as possible spending between $7 and $15.

**Sales Categories:**

Baseline Purchase - Between $0.99 and $.199

Low Purchase - Between $2.00 and $6.99

Target Purchase - Between $7 and $15

Top Performer - Above $15

The query will be as follows:

**SELECT**

**InvoiceDate,**

**BillingAddress,**

**BillingCity,**

**Total,**

**CASE**

**WHEN total <2.00 THEN 'Baseline Purchase'**

**WHEN total BETWEEN 2.00 AND 6.99 THEN 'Low Purchase'**

**WHEN total BETWEEN 7 AND 15 THEN 'Target Purchase'**

**ELSE 'Top Performer'**

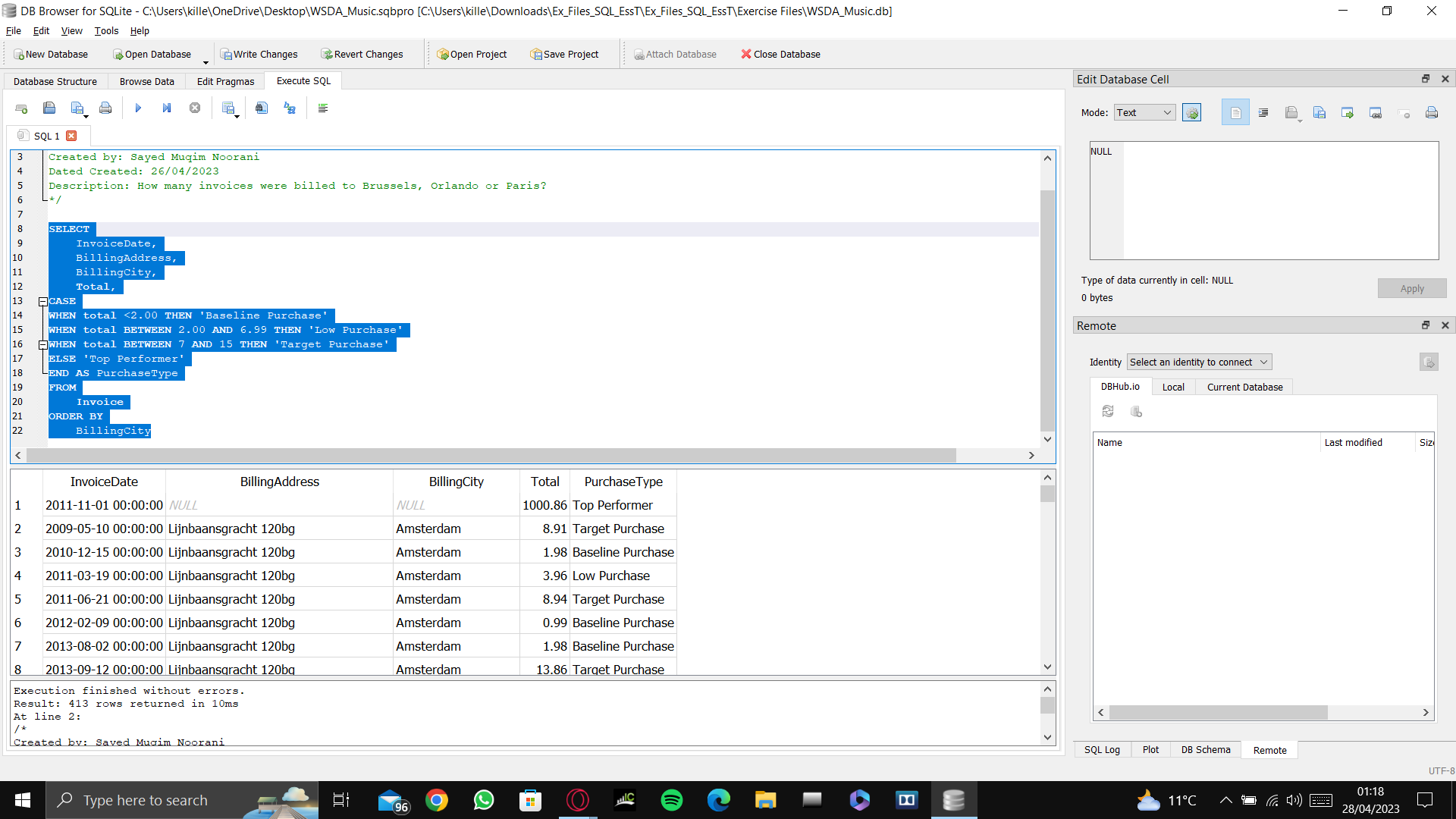
**END AS PurchaseType**

**FROM**

**Invoice**

**ORDER BY**

**BillingCity**



It is very important to add the comma before the **CASE** clause, otherwise it will give us an error. The **CASE** clause is used to add a case to the results, and essentially, another column of data. In this case it was different purchase types, and depending on the total of the invoices, the purchase would be land in one of four categories.

**SELECT**

**InvoiceDate,**

**BillingAddress,**

**BillingCity,**

**Total,**

**CASE**

**WHEN total <2.00 THEN 'Baseline Purchase'**

**WHEN total BETWEEN 2.00 AND 6.99 THEN 'Low Purchase'**

**WHEN total BETWEEN 7 AND 15 THEN 'Target Purchase'**

**ELSE 'Top Performer'**

**END AS PurchaseType**

**FROM**

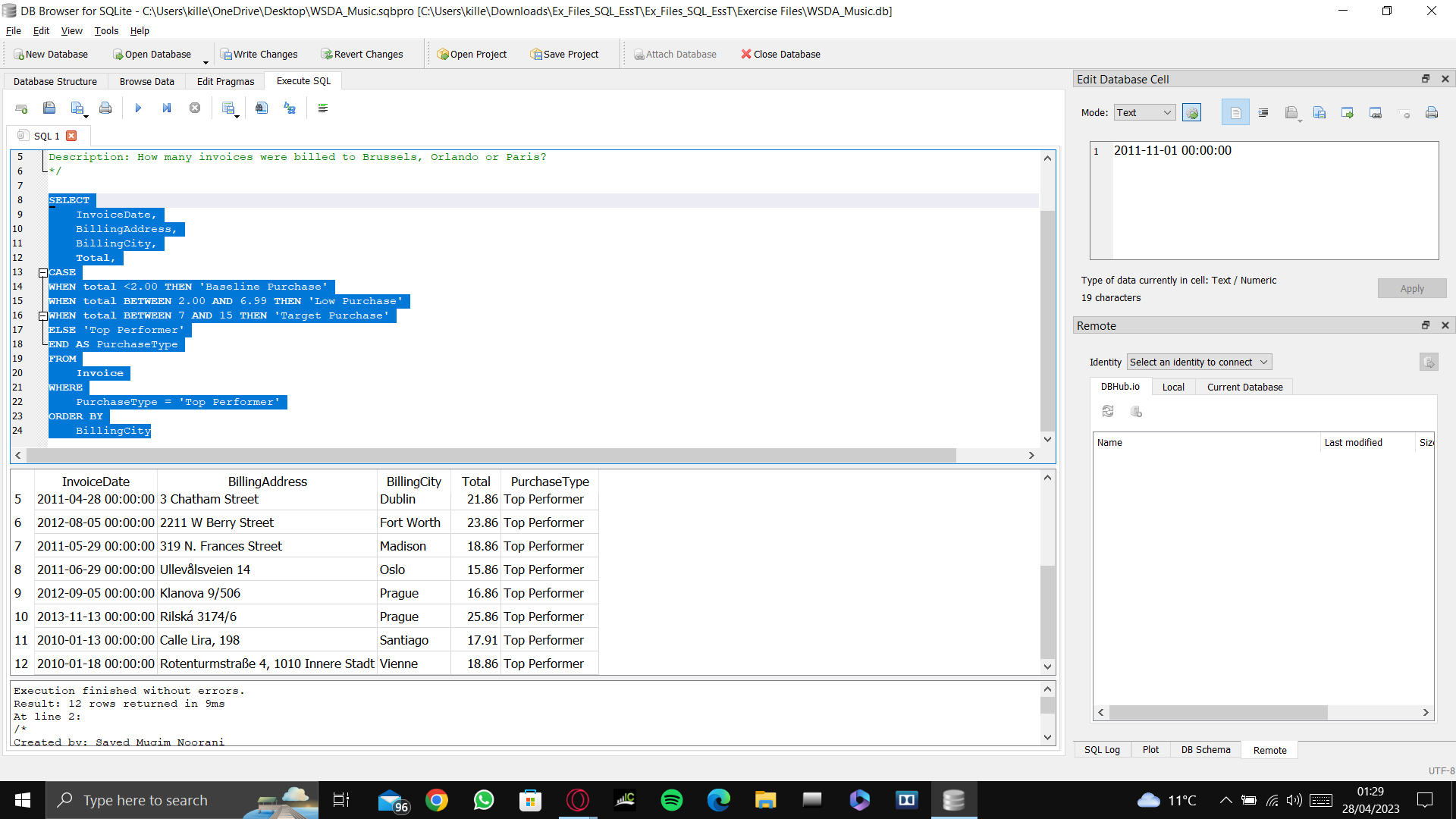
**Invoice**

**WHERE**

**PurchaseType = 'Top Performer'**

**ORDER BY**

**BillingCity**



The **WHERE** clause is then added after the **FROM** clause, and it makes it much easier now to draw information about the invoices and for management to see which cities account for most of the top performers. In the above case we can see that most of the cities that account for most of the top performers are outside of the United States.

# **How to JOIN two tables together**

**SELECT**

**\***

**FROM**

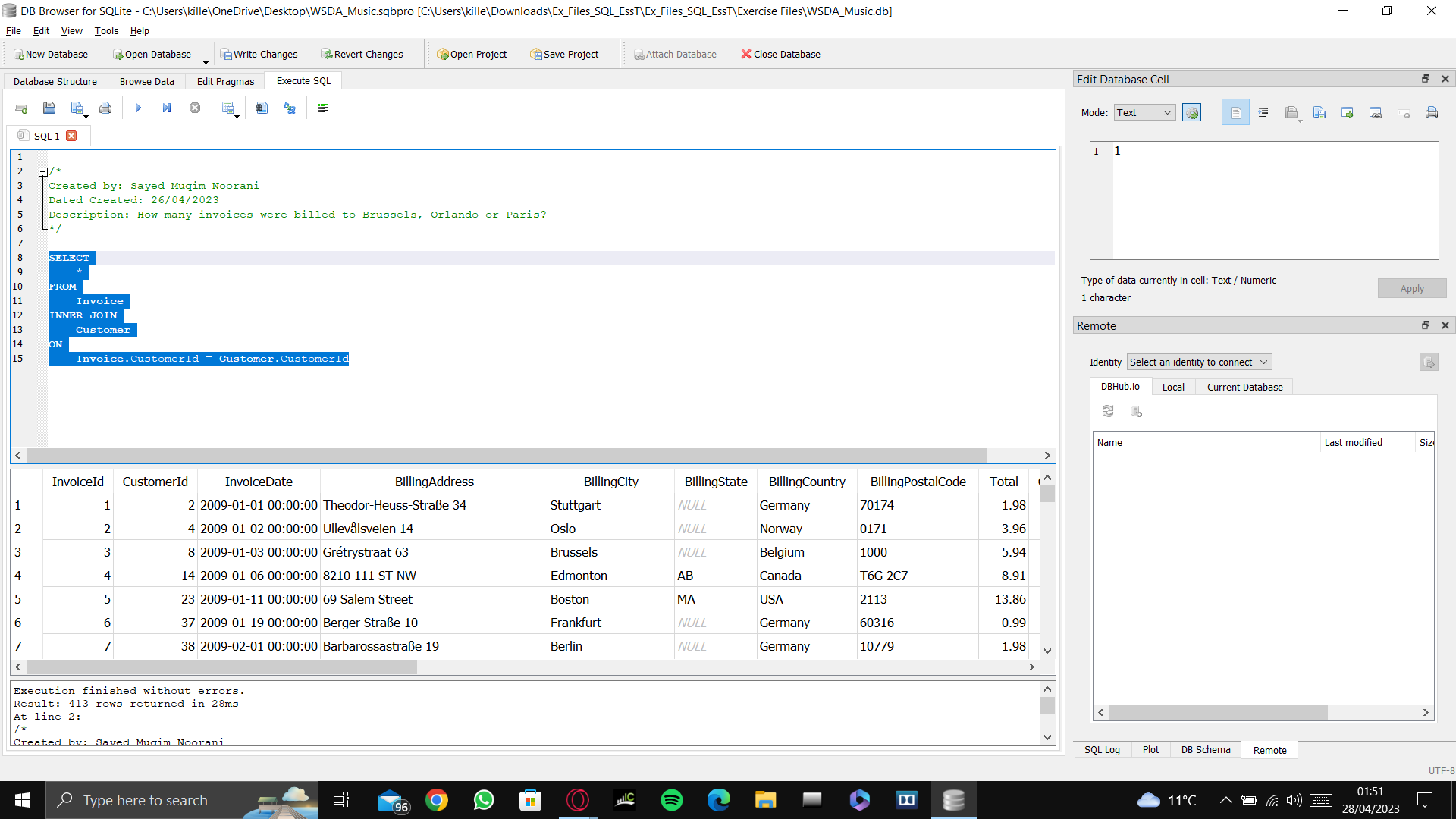
**Invoice**

**INNER JOIN**

**Customer**

**ON**

**Invoice.CustomerId = Customer.CustomerId**



In the above query we are asking to join two tables together which have one similar field with each other, in this case it is the **CustomerId** field. We have asked to select ALL fields from the **Invoice** table and join them using an **INNER JOIN** with the **Customer** table. The **ON** clause is then used to specify which fields are being used to join the two tables together, which in this case is **Invoice.CustomerId = Customer.CustomerId**. This means that we are using the **CustomerId** field from the **Invoice** table to join with the **CustomerId** field from the **Customer** table. Remember the format **TableName.FieldName = TableName.FieldName.**