SPRINT 2

NOCIONS BÀSIQUES SQL

This SPRINT helped me understand when to use JOINS, simple subqueries, and complex subqueries. Additionally, I learned how and when to use the HAVING and WHEN commands to filter queries.

PREPARED BY:

CECIL LUNA

CHECKED AND REVISED BY:

LAURA CUSCURITA

LEVEL 1

Exercise 1:

GOAL 1: include a diagram illustrating the relationship between the different tables and variables.
TABLES:
Company: Contains **id, company_name, phone, email, country,** and **website.**
Transactions: Contains **id, credit_card_id, company_id, user_id, lat, longitude, timestamp, amount,** and **declined.**

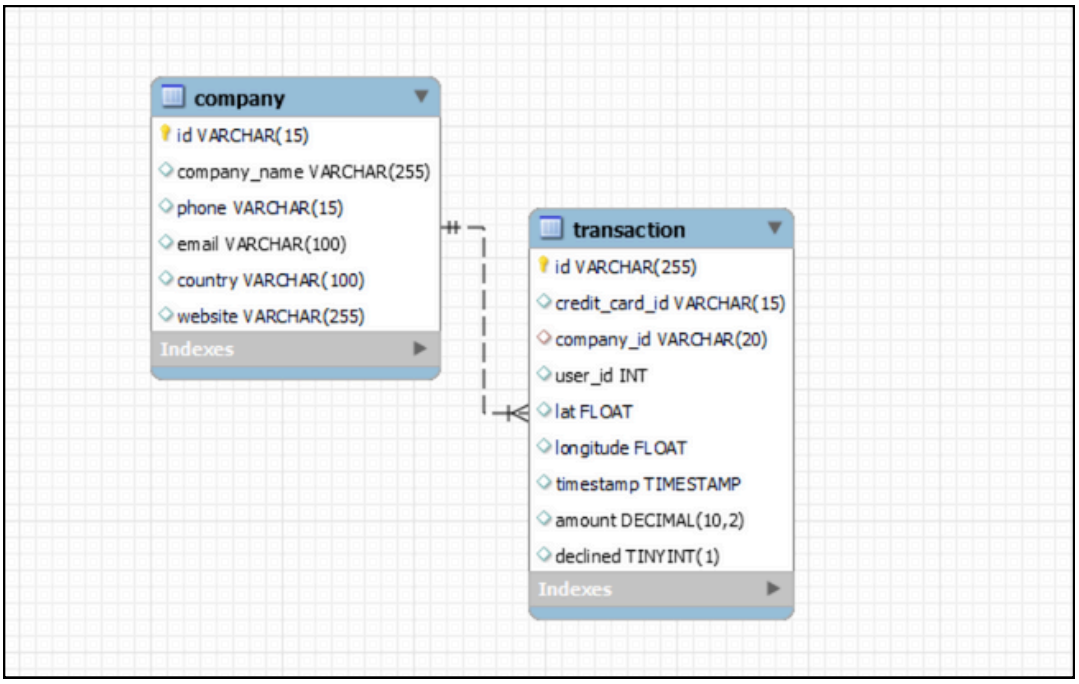


Figure 1.1.1 Relationship

EXPLANATION:
The relationship between the tables Company and Transaction is one-to-many because, a company can have multiple transactions.

COMPANY TABLE
This table has got all the details of the companies.
id- This column has the unique number value for each company. **This number is not repeated.** This serves as the **PRIMARY KEY** of this table.
company_name- This column provides the names of the companies in the table.
phone- Here is where the telephone numbers of the companies are stored.
email- This column has the emails of the companies.
country- This provides the name of the country where the companies are located.
website- Here, the official website of the companies are stored.

TRANSACTION TABLE

In this table, shown the movement of transactions of the companies and the amount spent by the companies each transaction. Besides that, this table provides the details of the declined transactions of the companies.

id- This column, has a unique number value for each transaction.
credit_card_id- This has the details of the credit card used in the transaction.
company_id- This has the number of the company to identify which company is doing the transaction. This is a FOREIGN KEY in this table.
user_id- This column has the user_id of the one who processed the transactions.
lat and longitude- These columns, indicate the location of the transactions.
timestamp- This has the time and date information of each transaction.
amount-This has the amount per transaction.
declined- This has the information if the transaction was processed/successful or not.

LEVEL 1

Exercise 2:

GOAL 1: List all the countries who are buying.
TABLES:
Company: Contains **country.**
Transactions: Contains **company_id** and **declined.**

```
7 •  SELECT DISTINCT company.country
8    FROM company
9    INNER JOIN transaction
10       ON company.id = transaction.company_id
11   WHERE declined = 0;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

country
Germany
Italy
United Kingdom
Sweden
Ireland
United States
Norway
France
New Zealand
Netherlands
Belgium
Australia
China
Spain

Result 1

Output

Action Output

#	Time	Action	Message
1	10:15:55	SELECT DISTINCT company.country FROM company INNER JOIN transaction ON company.id = transaction.c...	15 row(s) returned

Figure 1.2.1 Query + Result

EXPLANATION:
I utilized **INNER JOIN** to incorporate all countries with transactions. To ensure that country names do not repeat, despite multiple companies per country, I applied the **DISTINCT** command. Additionally, I filtered the results using **WHERE declined = 0** under the assumption that only companies without declined transactions are **actively making purchases.**

LEVEL 1

Exercise 2:

GOAL 2: Give the total number of countries that are purchasing.
TABLES:
Company: Contains **country.**
Transactions: Contains **company_id** and **declined.**

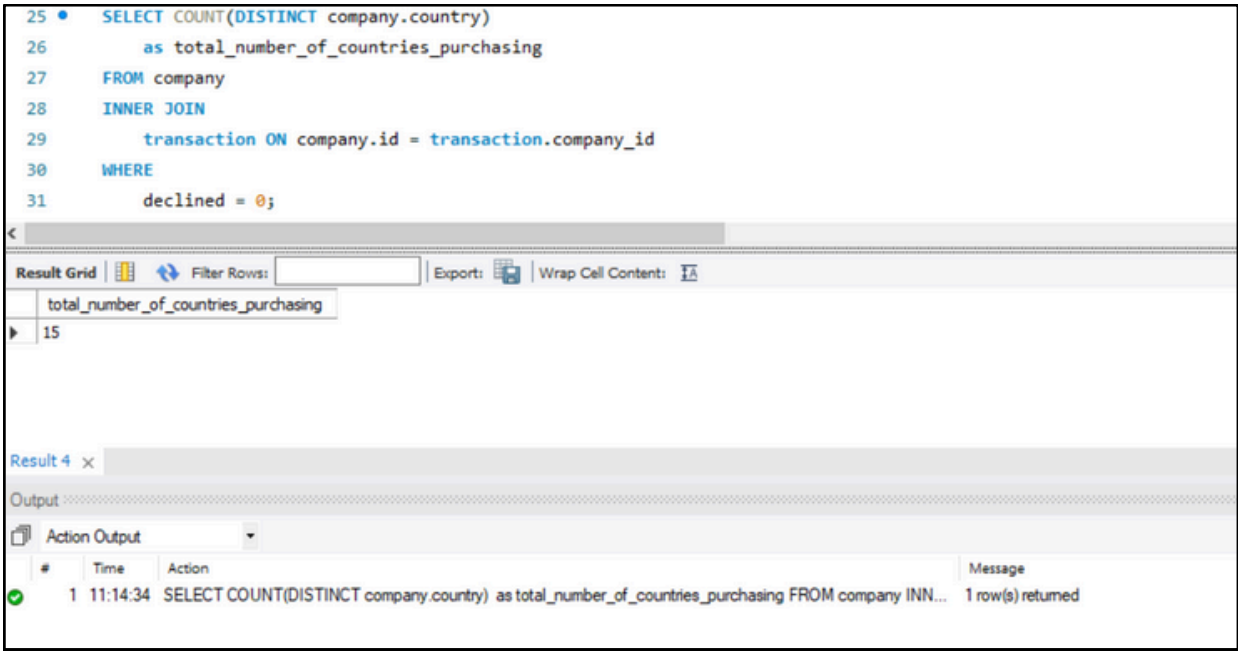


Figure 1.2.2 Query + Result

EXPLANATION:
This query is similar to **Figure 1.1**, but here I included the number of countries involved in transactions. To achieve this, I first used the **DISTINCT command** to identify unique countries actively making purchases. I then counted these distinct countries to ensure accuracy, as counting without DISTINCT would include duplicates and potentially yield **incorrect results**.

LEVEL 1

Exercise 2:

GOAL 3: Identify the company with the highest average sales.
TABLES:
Company: Contains **company_name**
Transactions: Contains **company_id** and **amount.**

```
24 •  SELECT company.company_name,  ROUND (AVG (transaction.amount),2) as average
25    FROM transaction
26    INNER JOIN company on transaction.company_id = company.id
27    WHERE declined = 0
28    GROUP BY company.company_name
29    ORDER BY average DESC
30    LIMIT 1;
31
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

company_name	average
Eget Ipsum Ltd	481.86

Result 13

Output

Action Output

#	Time	Action	Message
1	10:49:20	SELECT company.company_name, ROUND (AVG (transaction.amount),2) as average FROM transaction INNE...	1 row(s) returned

Figure 1.2.3 Query + Result

EXPLANATION:
Here, to include all companies with sales, I used **INNER JOIN**. Afterward, I calculated the **average sales for each company**. Also, I filtered the results using WHERE declined = 0 under the assumption that only companies without declined transactions are actively making purchases should be included. Moreover, to determine which company has the highest average sales, I sorted them in **descending order** and simplified by **limiting the results** to just 1 row.

LEVEL 1

Exercise 3:

GOAL 1: Display all transactions made by companies in Germany.
TABLES:
Company: Contains **company_id** and **country**
Transactions: Contains **id, credit_card_id, company_id, user_id, lat, longitude, timestap, amount** and **declined**

```
39
40 •  SELECT *
41    FROM transaction
42    WHERE company_id IN (SELECT company.id
43                         FROM company
44                         WHERE company.country = 'Germany');
45
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	credit_card_id	company_id	user_id	lat	longitude	timestamp	amount	declined
108B1D1D-5B23-A76C-55EF-C568E49A05DD	CcU-2938	b-2222	275	83.7839	-178.86	2021-07-07 17:43:16	293.57	0
EA2C3281-C9C1-A387-44F8-729FB4B51C76	CcU-2938	b-2222	275	20.2004	-116.84	2021-05-09 10:25:08	119.36	1
0DD2E608-5C9E-D1B3-4999-B99F43AD735A	CcU-2959	b-2234	275	9.68811	130.282	2021-04-17 05:30:17	252.47	1
AB069F53-965E-A2A8-CE06-CA8C4FD92501	CcU-2959	b-2234	275	1.64819	-158.007	2021-04-15 13:37:18	60.99	0
0466A42E-47CF-8D24-FD01-C0B689713128	CcU-4219	b-2302	170	-43.9695	-117.525	2021-07-26 07:29:18	49.53	0
0A476ED9-0C13-1962-F87B-D3563924B539	CcU-4359	b-2302	221	-56.4901	114.801	2022-02-26 20:33:54	430.49	0

transaction 67 ×

Output

Action Output

#	Time	Action	Message
1	22:11:50	SELECT * FROM transaction WHERE company_id IN (SELECT company.id FROM company	W... 118 row(s) returned

Figure 1.3.1 Query + Result

EXPLANATION:
MAIN QUERY:
SELECT * FROM transaction;

- This part of the query **retrieves all** the information from the **transaction** table.

SUBQUERY IN 'SELECT' CLAUSE:

```
SELECT id
    FROM company
            WHERE country = 'Germany'
```

To refine the results of the **main query**, I decided to join the two tables. I matched the **'company_id' column in the 'transaction' table** with the **'id' column in the 'company' table**, setting this as the condition in the main query. Additionally, I included a condition in the subquery to retrieve German companies.

LEVEL 1

Exercise 3:

GOAL 2: List companies that have made transactions for an amount greater than the average of all transactions.
Company: Contains **company_id** and **company_name**
Transactions: Contains **company_id** and **amount**

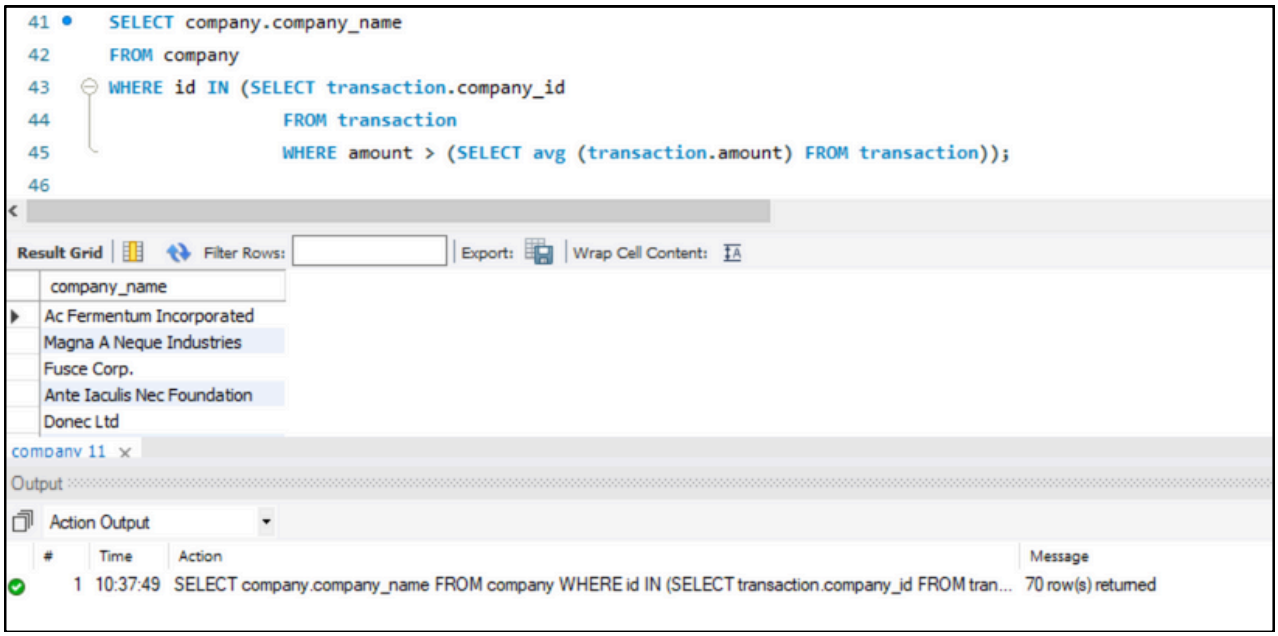


Figure 1.3.2 Query + Result

EXPLANATION:
MAIN QUERY:
The names of the company are fetched that's why the main query used the company table. To connect 2 tables, a condition in where was necessary.
SUBQUERY 1 IN 'WHERE' CLAUSE:
(SELECT transaction.company_id
FROM transaction
WHERE amount >

- This subquery is used to fetch the amount of transactions of the corresponding company_id.

SUBQUERY 2 IN 'WHERE' CLAUSE:

(SELECT avg (transaction.amount) FROM transaction)

- This subquery calculates the average amount of transactions from transaction table.

EXPLANATION OF THE COMPLETE SUBQUERIES

```
WHERE id IN (SELECT transaction.company_id
    FROM transaction
            WHERE amount > (SELECT avg (transaction.amount) FROM transaction))
```

- This fetch the companies only with transactions that are greater than the average amount of transactions.

LEVEL 1

Exercise 3:

GOAL 3: Eliminate from the system the companies that lack registered transactions, and provide a list of these companies.
Company: Contains **company_name**
Transactions: Contains **company_id**

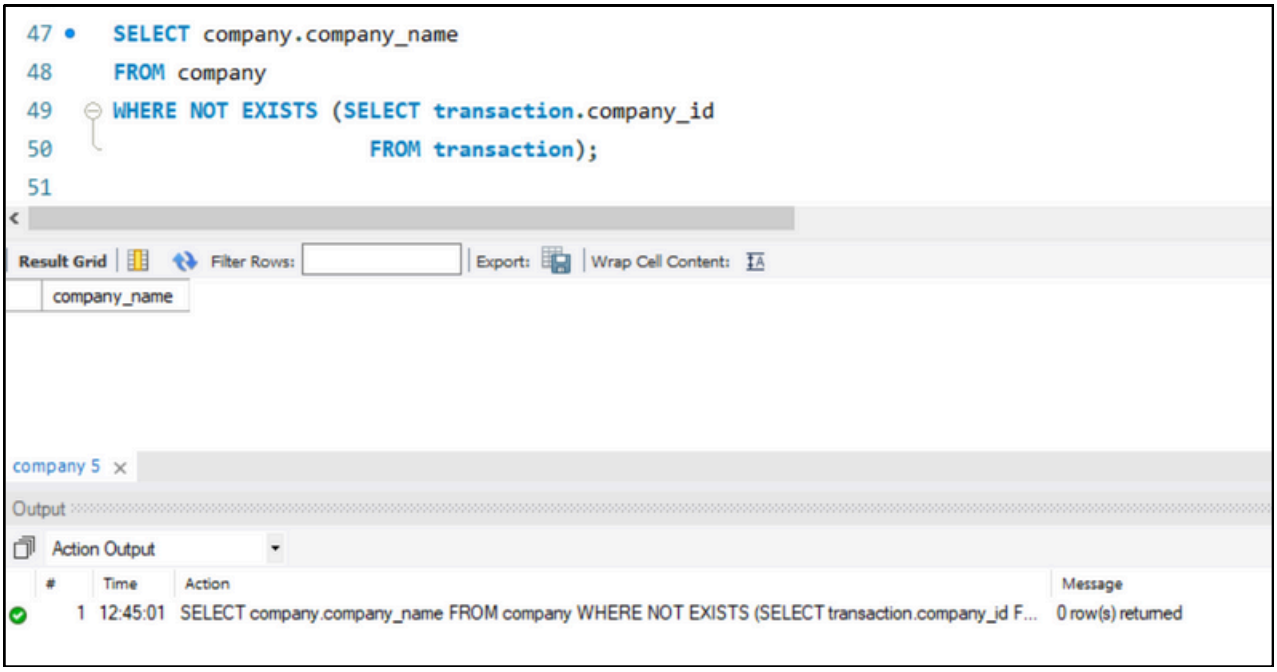


Figure 1.3.3 Query + Result

EXPLANATION:
MAIN QUERY:
SELECT company.company_name
FROM company

- This part of the query retrieves all the companies from the table company.

SUBQUERY IN 'SELECT' CLAUSE:
WHERE NOT EXISTS (SELECT transaction.company_id
FROM transaction);

- This subquery is used to fetch the corresponding company_name of each company_id that doesn't exist in both tables.

LEVEL 2

Exercise 1:

GOAL 1: Identify the five days that generated the most sales revenue for the company. Display the date of each transaction along with the total sales.
Transactions: Contains **timestamp** and **amount.**

```
54 •  SELECT date(timestamp) as date, SUM(transaction.amount) as amount
55    FROM transaction
56    WHERE declined = 0
57    GROUP BY date(timestamp)
58    ORDER BY amount DESC
59    LIMIT 5;
60
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

date	amount
2021-12-20	1532.36
2021-04-22	1397.96
2021-05-09	1344.37
2022-02-26	1337.62
2021-03-29	1325.12

Result 26

Output

Action Output

#	Time	Action	Message
1	13:39:33	SELECT date(timestamp) as date, SUM(transaction.amount) as amount FROM transaction WHERE declined = 0 ...	5 row(s) returned

Figure 2.1.1 Query + Result

EXPLANATION:
All the data is found in the transaction table. To get the desired results, it is necessary to calculate the total sales for the company, excluding those with declined transactions. Then, to retrieve just the date, use DATE (timestamp). Finally, to get the five days with the highest sales, limit the result to 5.

LEVEL 2

Exercise 2:

GOAL 1: Present the results ordered from highest to lowest average.
Company: Contains **country,** and **id.**
Transactions: Contains **amount,** and **company_id.**

```
63 •  SELECT company.country, AVG (transaction.amount) as average_sales
64    FROM transaction
65    INNER JOIN company ON company.id = transaction.company_id
66    WHERE declined = 0
67    GROUP BY company.country
68    ORDER BY average_sales DESC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

country	average_sales
United States	287.531111
Ireland	285.825357
Sweden	276.668382
United Kingdom	271.767527
Canada	261.941930
Belgium	255.217500
Norway	251.114918
Italy	243.342222
Germany	242.239189
Netherlands	240.940000
China	222.240000
Australia	177.331667
France	169.410000
New Zealand	167.061667
Spain	26.220000

Result 3

Output

Action Output

#	Time	Action	Message
1	10:00:11	SELECT company.country, AVG (transaction.amount) as average_sales FROM transaction INNER JOIN compa...	15 row(s) returned

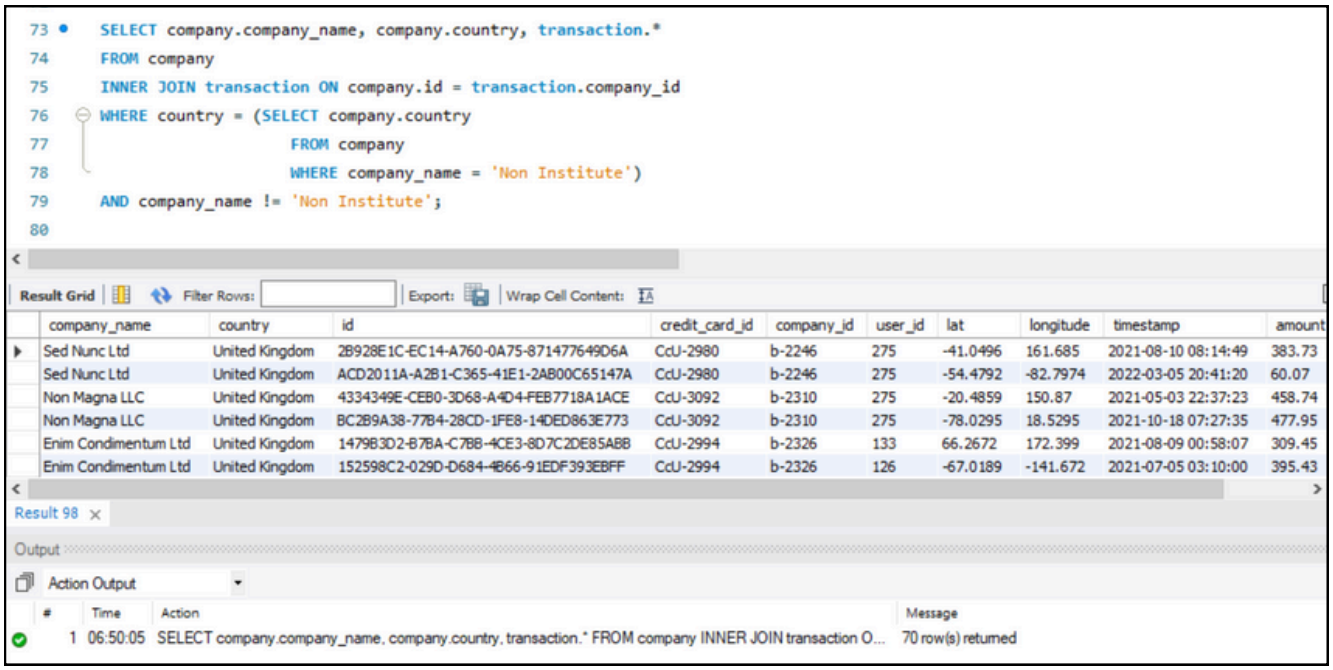
Figure 2.2.1 Query + Result

EXPLANATION:
To obtain the results, data should be collected from the 'company' and 'transactions' tables using the INNER JOIN command. Only those with valid transactions will be included in the query, so transactions that were declined will not be part of the results. The query calculates the average sales amount per country and arranges the results in descending order to show the countries from highest to lowest average sales.

LEVEL 2

Exercise 3:

GOAL 1: In your company, a new project is proposed to launch some advertising campaigns to compete with the company "Non Institute". To do so, they ask you for a list of all the transactions made by companies located in the same country as this company.
Company: Contains **country,** and **company_name.**
Transactions: Contains **id, credit_card_id, company_id, user_id, lat, longitude, timestamp, amount,** and **declined.**



```
73 •  SELECT company.company_name, company.country, transaction.*
74    FROM company
75    INNER JOIN transaction ON company.id = transaction.company_id
76    WHERE country = (SELECT company.country
77                     FROM company
78                     WHERE company_name = 'Non Institute')
79    AND company_name != 'Non Institute';
80
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

company_name	country	id	credit_card_id	company_id	user_id	lat	longitude	timestamp	amount
Sed Nunc Ltd	United Kingdom	2B928E1C-EC14-A760-0A75-871477649D6A	CcU-2980	b-2246	275	-41.0496	161.685	2021-08-10 08:14:49	383.73
Sed Nunc Ltd	United Kingdom	ACD2011A-A2B1-C365-41E1-2AB00C65147A	CcU-2980	b-2246	275	-54.4792	-82.7974	2022-03-05 20:41:20	60.07
Non Magna LLC	United Kingdom	4334349E-CEB0-3D68-A4D4-FEB7718A1ACE	CcU-3092	b-2310	275	-20.4859	150.87	2021-05-03 22:37:23	458.74
Non Magna LLC	United Kingdom	BC2B9A38-77B4-28CD-1FE8-14DED863E773	CcU-3092	b-2310	275	-78.0295	18.5295	2021-10-18 07:27:35	477.95
Enim Condimentum Ltd	United Kingdom	1479B3D2-B7BA-C7BB-4CE3-8D7C2DE85ABB	CcU-2994	b-2326	133	66.2672	172.399	2021-08-09 00:58:07	309.45
Enim Condimentum Ltd	United Kingdom	152598C2-029D-D684-4B66-91EDF393EBFF	CcU-2994	b-2326	126	-67.0189	-141.672	2021-07-05 03:10:00	395.43

Result 98

Output

Action Output

#	Time	Action	Message
1	06:50:05	SELECT company.company_name, company.country, transaction.* FROM company INNER JOIN transaction O...	70 row(s) returned

Figure 2.3.1 Query + Result

EXPLANATION:
Only the competitors of 'Non Institute' are fetched to check their transactions. First, under the SELECT clause, the names of the columns that appear in the results are included. To merge the two tables, the INNER JOIN command is used; with this, only matching data will be retrieved. Lastly, to filter the result where 'Non Institute' is located, a subquery is used in the WHERE clause along with another filter to retrieve only the competitors of 'Non Institute,' excluding itself.

LEVEL 2

Exercise 3:

GOAL 2: In your company, a new project is proposed to launch some advertising campaigns to compete with the company "Non Institute". To do so, they ask you for a list of all the transactions made by companies located in the same country as this company.
Company: Contains **country,** and **company_name.**
Transactions: Contains **id, credit_card_id, company_id, user_id, lat, longitude, timestamp, amount,** and **declined.**

```
SELECT
        (SELECT company.company_name
        FROM company
        WHERE company.id = transaction.company_id
        AND company_name != 'Non Institute'
        AND country IN (SELECT company.country
                        FROM company
                        WHERE company_name = 'Non Institute')) as company_name,
        (SELECT company.country
        FROM company
        WHERE company.id = transaction.company_id
        AND company_name != 'Non Institute'
        AND country IN (SELECT company.country FROM company WHERE company_name = 'Non Institute')) as country,
        transaction.*
FROM transaction
HAVING company_name IS NOT NULL;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

company_name	country	id	credit_card_id	company_id	user_id	lat	longitude	timestamp	amou
Enim Condimentum Ltd	United Kingdom	1479B3D2-B7BA-C7B8-4CE3-8D7C2DE85ABB	CcU-2994	b-2326	133	66.2672	172.399	2021-08-09 00:58:07	309.4
Enim Condimentum Ltd	United Kingdom	152598C2-029D-D684-4B66-91EDF393EBFF	CcU-2994	b-2326	126	-67.0189	-141.672	2021-07-05 03:10:00	395.4
Enim Condimentum Ltd	United Kingdom	1B636B58-A2E8-7C69-D9C9-C54535DAFD3B	CcU-2994	b-2326	131	70.2543	-13.1336	2021-07-06 08:48:46	195.0
Enim Condimentum Ltd	United Kingdom	20418DE5-B804-BE9B-BD7A-A95C1BFDBF5C	CcU-2994	b-2326	126	-79.1145	1.51481	2022-01-03 15:59:29	479.5

Result 100

Output

Action Output

#	Time	Action	Message
1	06:57:04	SELECT (SELECT company.company_name FROM company WHERE company.id = transaction.com...	70 row(s) returned

Figure 2.3.1 Query + Result

EXPLANATION:
MAIN QUERY :
SELECT transaction.*
FROM transaction;
This query retrieves all the transactions from the transaction table.

SUBQUERY QUERY 1 (SELECT CLAUSE)

```
(SELECT company.company_name
FROM company
WHERE company.id = transaction.company_id
AND company_name != 'Non Institute'
AND country IN (SELECT company.country
FROM company
WHERE company_name = 'Non Institute')) as company_name
```

This query adds a company_name column to the main query. And this is the list of the competitors of 'Non Institute' and it follows the same format as in Figure 2.3.1. This subquery excludes 'Non Institute' in the results.

SUBQUERY QUERY 2 (SELECT CLAUSE)

```
(SELECT company.country
      FROM company
 WHERE company.id = transaction.company_id
      AND company_name != 'Non Institute'
    AND country IN (SELECT company.country FROM company WHERE company_name
= 'Non Institute')) as country
```

To identify the country where the 'Non Institute' is located, we added a new column, 'country,' to the main query results. This retrieves only the country where the 'Non Institute' can be found.

HAVING company_name IS NOT NULL

There was a problem in the query because even those without matching data from the set conditions of the main query and subqueries were being retrieved. To filter that out, NULL results were eliminated.

LEVEL 3

Exercise 1:

GOAL 1: Presents the name, telephone number, country, date and amount, of those companies that carried out transactions with a value between 100 and 200 euros and on any of these dates: April 29, 2021, July 20, 2021 and March 13, 2022. Order the results from highest to lowest amount.
Company: Contains **country, company_name,** and **phone.**
Transactions: Contains **amount,** and **timestamp.**

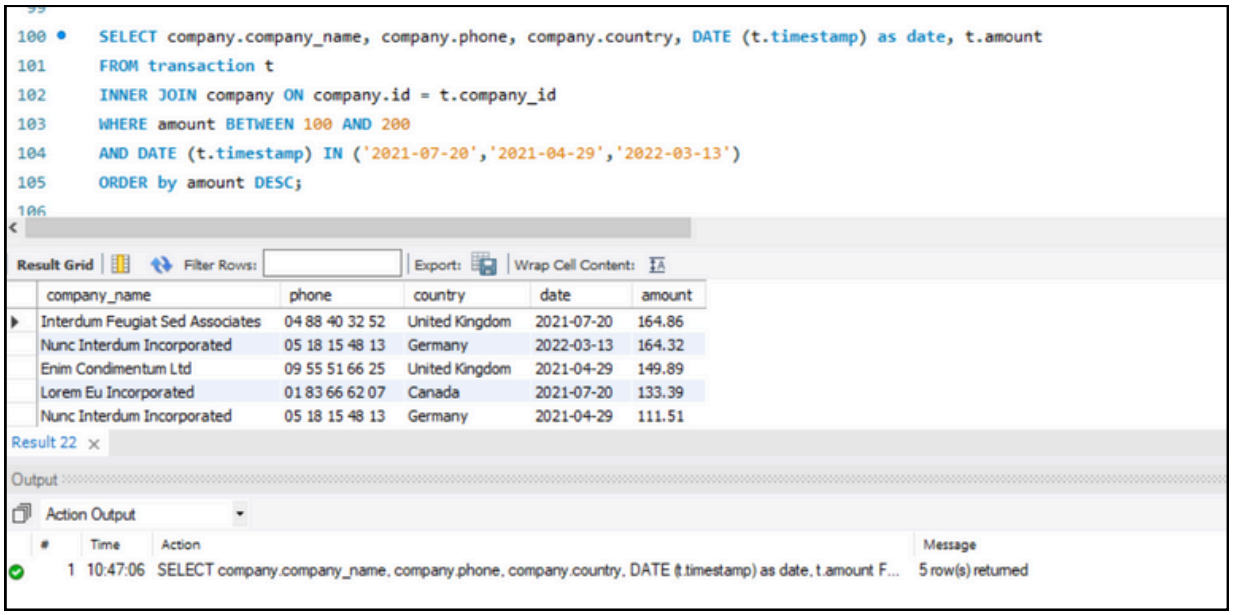


Figure 2.3.1 Query + Result

EXPLANATION:
The query must fetch data of the following: **company_name, phone, country, date, and amount.** The query needs to filter the data of ' the amount' of transactions and the date, that's why the transaction table will be used as the main table in this query for easy filtering. To retrieve data from the company table, the two tables should be merged using an INNER JOIN command.

After merging the tables, the first condition is introduced. Only the companies with transaction amounts ranging between 100 and 200 will be collected. And the second condition is applied on dates and the last filter is ordering the results from highest to lowest.

LEVEL 3

Exercise 2:

GOAL 1: Optimize the allocation of resources and it will depend on the operational capacity required and give the information on the number of transactions performed by the companies, but only the list of the companies where you specify if they have more than 4 or less transactions.
Company: Contains **company_name**
Transactions: Contains **declined.**

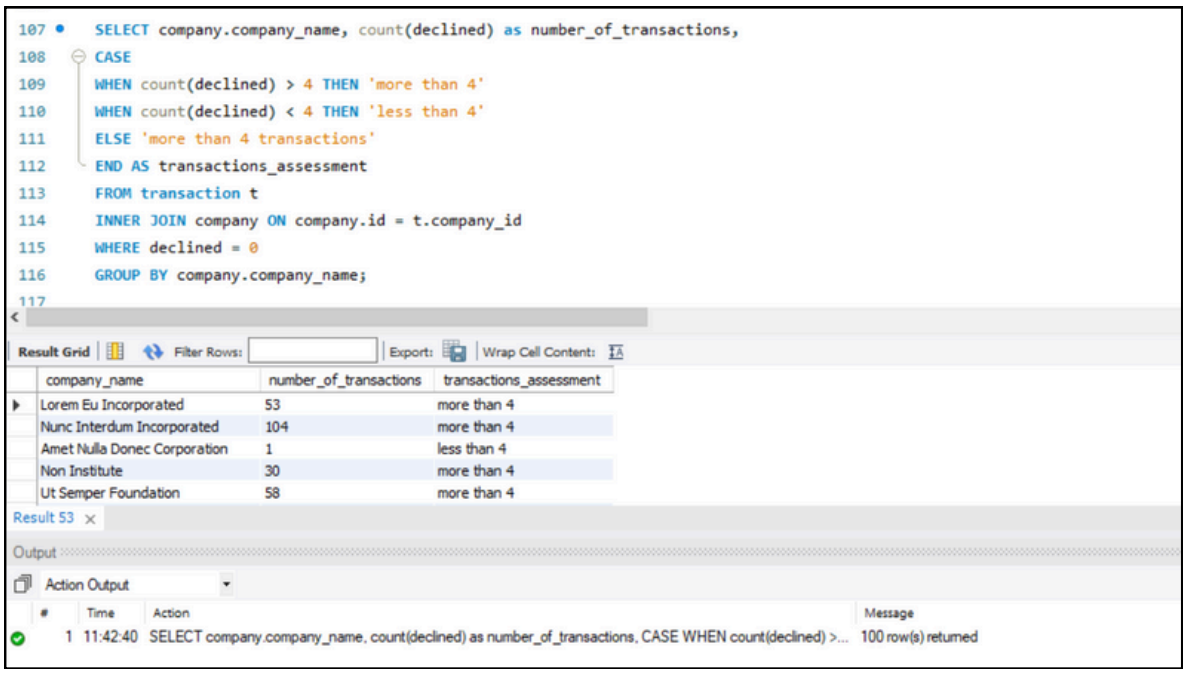


Figure 2.3.1 Query + Result

EXPLANATION:
This query fetches the names of companies and their total number of transactions, assessing whether they have more or fewer than four transactions each. To achieve this, two tables are merged using an **INNER JOIN command**.

Using WHEN with CASE in the SELECT clause:
An additional column is added to calculate whether the transactions are more than four or fewer, based on the retrieved data.

To count only valid transactions, those without declined transactions are included. This is filtered with **declined = 0.**