

SPRINT 4

CREACIÓ DE BASE DE DADES



Designing and creating database starting from some CSV files.

PREPARED BY:

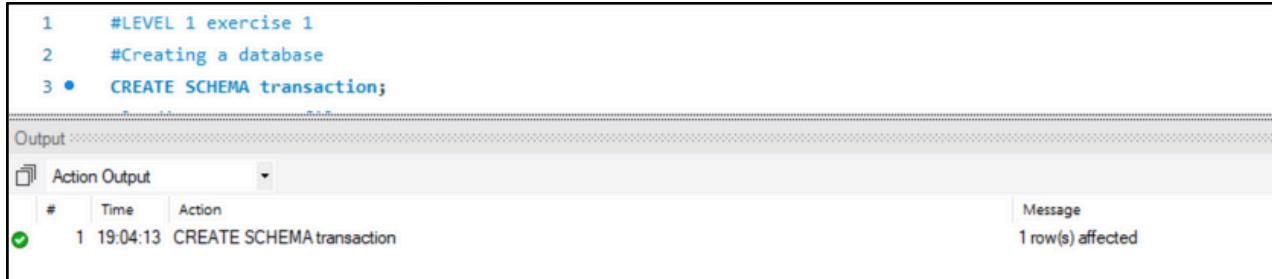
CECIL LUNA

CHECKED AND REVISED BY:

Sara Gutierrez

LEVEL 1

Download the CSV files, study them, and design a database with a star schema that has at least 4 tables.



The screenshot shows a command-line interface for creating a database. The input area contains the following SQL commands:

```
1 #LEVEL 1 exercise 1
2 #Creating a database
3 • CREATE SCHEMA transaction;
```

The output area is titled "Output" and shows the results of the command:

Action Output
Time Action
1 19:04:13 CREATE SCHEMA transaction

On the right side of the output area, there is a "Message" section that says "1 row(s) affected".

Figure 1.0.1 Creating a Database

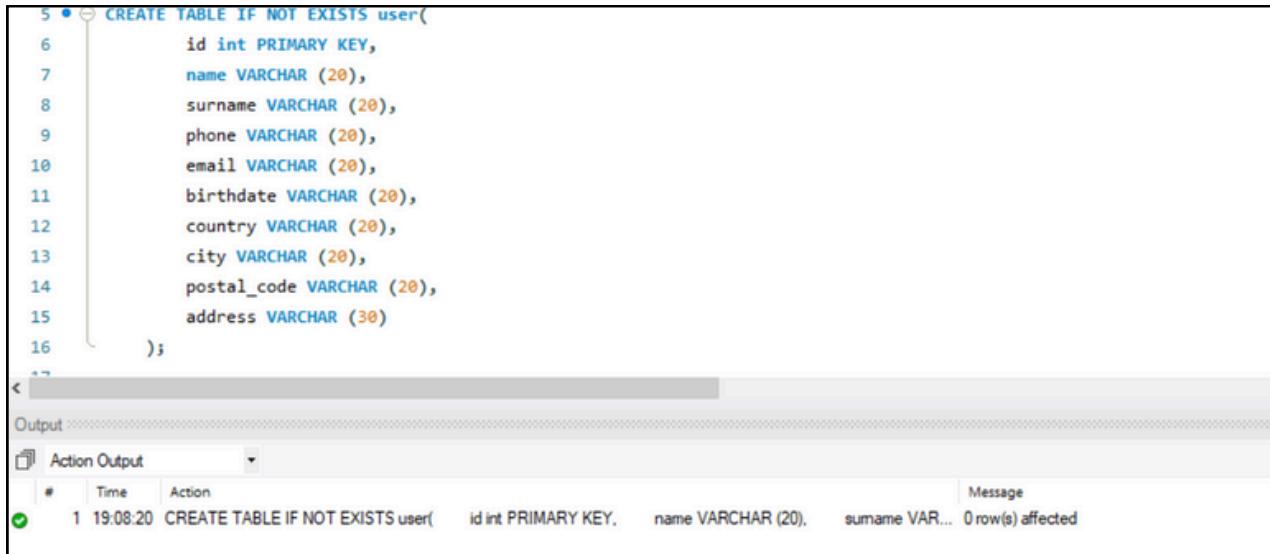
EXPLANATION:

After analyzing the CSV files, I created a database named **transaction**, as all the tables are similar to those from Sprint 3. I ensured that I was aware of this change to avoid confusion when retrieving data from the tables.

To create the database, I used the **CREATE SCHEMA** command to create a database named **transaction**.

LEVEL 1

Download the CSV files, study them, and design a database with a star schema that has at least 4 tables.



The screenshot shows the MySQL Workbench interface. In the SQL editor pane, a CREATE TABLE statement is being typed:

```
5 * CREATE TABLE IF NOT EXISTS user(
6     id int PRIMARY KEY,
7     name VARCHAR (20),
8     surname VARCHAR (20),
9     phone VARCHAR (20),
10    email VARCHAR (20),
11    birthdate VARCHAR (20),
12    country VARCHAR (20),
13    city VARCHAR (20),
14    postal_code VARCHAR (20),
15    address VARCHAR (30)
16 );
17
```

In the Output pane, the results of the query are shown:

#	Time	Action	Message
1	19:08:20	CREATE TABLE IF NOT EXISTS user(id int PRIMARY KEY, name VARCHAR (20), surname VAR... 0 row(s) affected

Figure 1.0.2 Creating user table

EXPLANATION:

After analyzing the CSV files by viewing them like an Excel file, I created tables based on the data presented using the CREATE TABLE command. Add all the column names and their corresponding data types. For instance, the user table has the following columns and corresponding data types:

- **id INT PRIMARY KEY:** I use the INT data type because id consists of numbers.
- **name VARCHAR(20):** name contains letters, and I limit it to 20 characters because it's unlikely to have a name longer than that.
- **surname VARCHAR(20):** surname contains letters, and I limit it to 20 characters because it's unlikely to have a surname longer than that.
- **phone VARCHAR(50):** I use VARCHAR here because the phone number can include special characters like -, (, and).
- **email VARCHAR(50):** email contains alphanumeric and special characters, so I use VARCHAR.
- **birthdate VARCHAR(20):** To accommodate various date formats safely, I use VARCHAR.
- **country VARCHAR(20):** country contains letters, so I use VARCHAR.
- **city VARCHAR(50):** Similar to country, city contains letters, so I use VARCHAR.
- **postal_code VARCHAR(20):** Due to the presence of alphanumeric characters, I use VARCHAR.
- **address VARCHAR(100):** Because address includes alphanumeric characters, I use VARCHAR.

```

14 • LOAD DATA INFILE 'C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\SPRINT 4\\users_usa.csv'
15 INTO TABLE user
16 FIELDS TERMINATED BY ','
17 ENCLOSED BY ""
18 LINES TERMINATED BY '\\r\\n'
19 IGNORE 1 ROWS;
< -----
Output: Action Output
# Time Action Message
1 11:54:16 LOAD DATA INFILE 'C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\SPRINT 4\\users_usa.csv' INT... 150 row(s) affected Records: 150 Deleted: 0 Skipped: 0 Warnings: 0

21 • LOAD DATA INFILE 'C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\SPRINT 4\\users_uk.csv'
22 INTO TABLE user
23 FIELDS TERMINATED BY ','
24 ENCLOSED BY ""
25 LINES TERMINATED BY '\\r\\n'
26 IGNORE 1 ROWS;
27
< -----
Output: Action Output
# Time Action Message
1 12:01:42 LOAD DATA INFILE 'C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\SPRINT 4\\users_uk.csv' INT... 50 row(s) affected Records: 50 Deleted: 0 Skipped: 0 Warnings: 0

28 • LOAD DATA INFILE 'C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\SPRINT 4\\users_ca.csv'
29 INTO TABLE user
30 FIELDS TERMINATED BY ','
31 ENCLOSED BY ""
32 LINES TERMINATED BY '\\r\\n'
33 IGNORE 1 ROWS;
34
< -----
Output: Action Output
# Time Action Message
1 12:04:26 LOAD DATA INFILE 'C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\SPRINT 4\\users_ca.csv' INT... 75 row(s) affected Records: 75 Deleted: 0 Skipped: 0 Warnings: 0
  
```

Figure 1.0.3 Loading users csv files

EXPLANATION:

After analyzing the CSV files **users_ca**, **users_uk**, and **users_usa**, I found that they have the same columns and data types. Merging them into one table will make the database more organized and cleaner.

Using the **LOAD DATA INFILE** command, I loaded the data from the locations where the CSV files are saved and inserted it into the user table in MySQL Workbench.

File Path Issue

I encountered a problem importing the CSV files due to their file location. The files should be stored where the MySQL Workbench files are stored. To find this location, I used the following command:

SHOW VARIABLES LIKE "secure_file_priv";

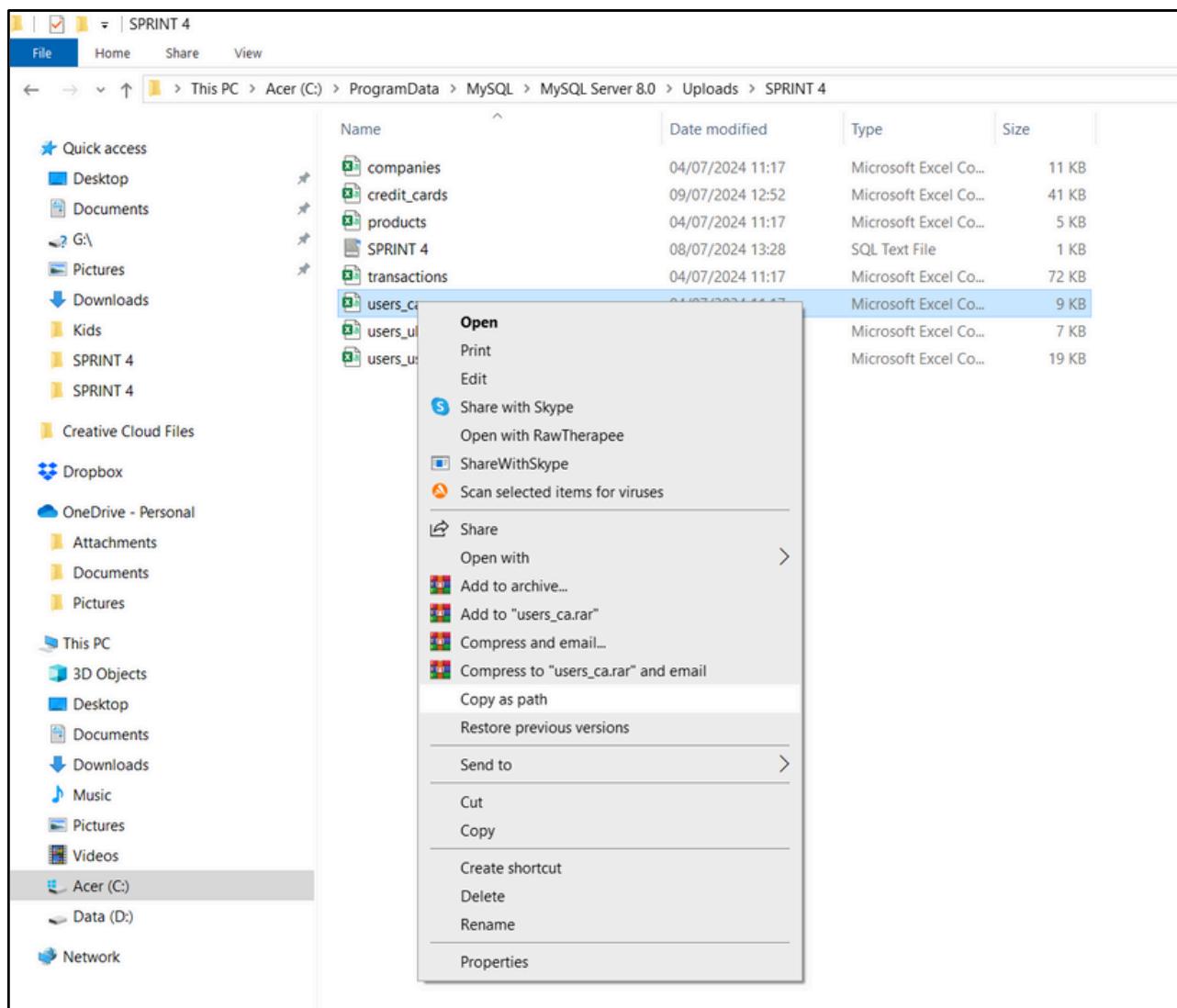
C:\ProgramData\MySQL\MySQL Server 8.0\Uploads\

HOW TO GET THE FILE PATH LINK

To get the file path after saving the CSV files where the MySQL Workbench files are located, follow these steps:

1. STEP 1: Click SHIFT and right-click on the file you want to get the file path for.
2. STEP 2: Select "Copy as path" from the context menu.

This will copy the file path to your clipboard, making it easy to use in your MySQL commands.



Handling Line Termination

The line LINES TERMINATED BY '\r\n' ensures that the query proceeds to the next line, even if there is a special character at the end of the line of values.

```

39      #loading companies csv file
40 • CREATE TABLE IF NOT EXISTS companies(
41          company_id VARCHAR (20) PRIMARY KEY,
42          company_name VARCHAR (100),
43          phone VARCHAR (50),
44          email VARCHAR (50),
45          country VARCHAR (50),
46          website VARCHAR (50)
47      );
48
<
Output:
Action Output
# Time Action Message
✓ 1 19:18:53 CREATE TABLE IF NOT EXISTS companies( company_id VARCHAR (20) PRIMARY KEY, company_n... 0 row(s) affected

49 • LOAD DATA INFILE 'C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\SPRINT 4\\companies.csv'
50     INTO TABLE companies
51     FIELDS TERMINATED BY ','
52     ENCLOSED BY ''
53     IGNORE 1 ROWS;
54
<
Output:
Action Output
# Time Action Message
✓ 1 19:19:56 LOAD DATA INFILE 'C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\SPRINT 4\\companies.csv' IN... 100 row(s) affected Records: 100 Deleted: 0 Skipped: 0 Warnings: 0
  
```

Figure 1.0.4 Loading companies csv file

EXPLANATION:

After analyzing the companies csv file by viewing it like an excel file, create a table after based from the data presented using CREATE TABLE command. Add all the column names and its data types. In this case, companies table has this following columns and corresponding data types:

- **company_id VARCHAR (20) PRIMARY KEY:** This column has special characters and numbers that's why VARCHAR is the best data type choice.
- **company_name VARCHAR (100):** This contains letters of the alphabet.
- **phone VARCHAR (50):** I use VARCHAR here because the phone number can include special characters like -, (, and).
- **email VARCHAR (50):** email contains alphanumeric and special characters, so I use VARCHAR.
- **country VARCHAR (50):** country contains letters, so I use VARCHAR.
- **website VARCHAR (50):** websites contain alphanumeric and special characters, so I use VARCHAR.

Using the **LOAD DATA INFILE** command recalls the data from where the companies csv file is saved. Then inserted to the companies table in MySQL workbench.

```

55      #loading credit_cards csv file
56 • CREATE TABLE IF NOT EXISTS credit_cards(
57          id VARCHAR (20) PRIMARY KEY,
58          user_id int,
59          iban VARCHAR (50),
60          pan VARCHAR (50),
61          pin VARCHAR (4),
62          cvv INT,
63          track1 VARCHAR (100),
64          track2 VARCHAR (100),
65          expiring_date VARCHAR (10)
66      );

```

Output :

#	Time	Action	Message
1	19:21:20	CREATE TABLE IF NOT EXISTS credit_cards(id VARCHAR (20) PRIMARY KEY, user_id int, iba...	0 row(s) affected


```

66 • LOAD DATA INFILE 'C:\\\\ProgramData\\\\MySQL\\\\MySQL Server 8.0\\\\Uploads\\\\SPRINT 4\\\\credit_cards.csv'
67   INTO TABLE credit_cards
68   FIELDS TERMINATED BY ','
69   ENCLOSED BY '\"'
70   IGNORE 1 ROWS;
71

```

Output :

#	Time	Action	Message
1	10:58:04	LOAD DATA INFILE 'C:\\\\ProgramData\\\\MySQL\\\\MySQL Server 8.0\\\\Uploads\\\\SPRINT 4\\\\credit_cards.csv' I... 275 row(s) affected Records: 275 Deleted: 0 Skipped: 0 Warnings: 0	

Figure 1.0.5 Loading credit_cards csv file

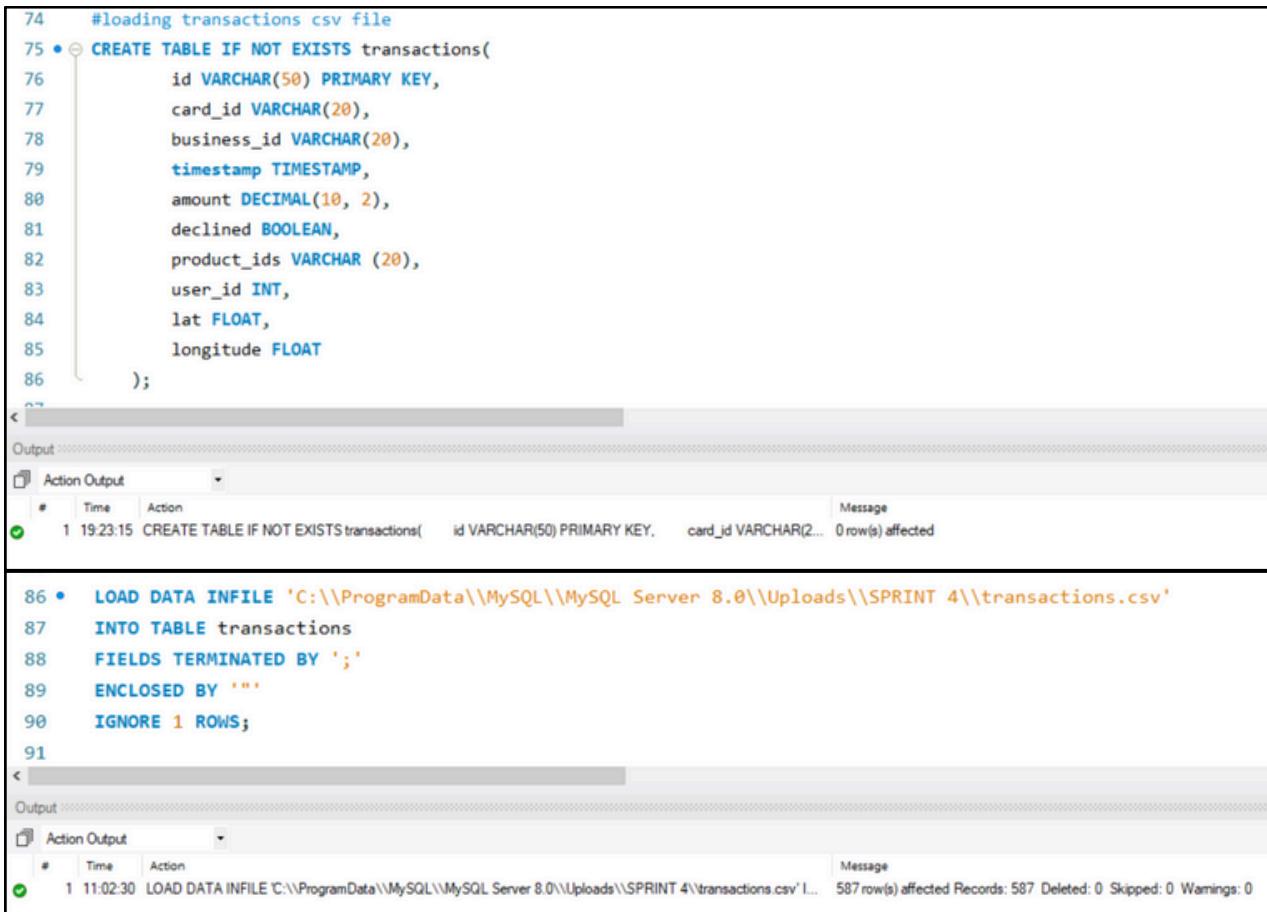
EXPLANATION:

After analyzing the credit_cards csv file by viewing it like an excel file, create a table after based from the data presented using CREATE TABLE command. Add all the column names and its data types. In this case, credit_cards table has this following columns and corresponding data types:

- **id VARCHAR (20) PRIMARY KEY:** This column has special characters and numbers that's why VARCHAR is the best data type choice.
- **user_id int:** I use the INT data type because id consists of numbers and it should be the same as the mother table.
- **iban VARCHAR (50):** It contains alphanumeric and special characters, so I use VARCHAR.
- **pan VARCHAR (50):** It contains alphanumeric and special characters, so I use VARCHAR.
- **pin VARCHAR (4):** I use the INT data type because id consists of numbers.
- **cvv INT:** I use the INT data type because id consists of numbers.

- **track1 VARCHAR (100):** It contains alphanumeric and special characters, so I use VARCHAR.
- **track2 VARCHAR (100):** It contains alphanumeric and special characters, so I use VARCHAR.
- **expiring_date VARCHAR (10):** To accommodate various date formats safely, I use VARCHAR.

Using the **LOAD DATA INFILE** command recalls the data from where the credit_cards csv file is saved. Then inserted to the credit_cards table in MySQL workbench.



The screenshot shows the MySQL Workbench interface. At the top, there is a code editor window containing SQL commands. Below it is an 'Output' window showing the results of the executed commands.

Create Table:

```
74 #loading transactions csv file
75 • CREATE TABLE IF NOT EXISTS transactions(
76     id VARCHAR(50) PRIMARY KEY,
77     card_id VARCHAR(20),
78     business_id VARCHAR(20),
79     timestamp TIMESTAMP,
80     amount DECIMAL(10, 2),
81     declined BOOLEAN,
82     product_ids VARCHAR (20),
83     user_id INT,
84     lat FLOAT,
85     longitude FLOAT
86 );
```

Output:

#	Time	Action	Message
1	19:23:15	CREATE TABLE IF NOT EXISTS transactions(id VARCHAR(50) PRIMARY KEY, card_id VARCHAR(2...)	0 row(s) affected

Load Data Infile:

```
86 • LOAD DATA INFILE 'C:\\\\ProgramData\\\\MySQL\\\\MySQL Server 8.0\\\\Uploads\\\\SPRINT 4\\\\transactions.csv'
87 INTO TABLE transactions
88 FIELDS TERMINATED BY ';'
89 ENCLOSED BY '\"'
90 IGNORE 1 ROWS;
91
```

Output:

#	Time	Action	Message
1	11:02:30	LOAD DATA INFILE 'C:\\\\ProgramData\\\\MySQL\\\\MySQL Server 8.0\\\\Uploads\\\\SPRINT 4\\\\transactions.csv' ...	587 row(s) affected Records: 587 Deleted: 0 Skipped: 0 Warnings: 0

Figure 1.0.6 Loading transactions csv file

EXPLANATION:

After analyzing the transactions csv file by viewing it like an excel file, create a table after based from the data presented using **CREATE TABLE command**. Add all the column names and its data types. In this case, transactions table has this following columns and corresponding data types:

- **id VARCHAR(50) PRIMARY KEY:** This column includes special characters and numbers, making VARCHAR the ideal data type choice.
- **card_id VARCHAR(20):** This column also contains special characters and numbers, making VARCHAR the best data type choice.
- **business_id VARCHAR(20):** Similar to card_id, this column includes special characters and numbers, making VARCHAR suitable.
- **timestamp TIMESTAMP:** This data type is used because the column stores date and time information.
- **amount DECIMAL(10, 2):** This is used for monetary values, allowing storage of 10-digit numbers with 2 decimal places.
- **declined BOOLEAN:** This column accepts inputs of 0 and 1, representing true or false values.
- **product_ids VARCHAR(20):** Contains alphanumeric characters and special characters.

- **user_id INT:** This data type is used for numerical user IDs, ensuring consistency with the primary key of the parent table.
- **lat FLOAT:** This data type is chosen due to the wide range of numerical values expected for latitude.
- **longitude FLOAT:** Similarly, FLOAT is used for longitude due to its ability to handle a wide range of numerical values.

Using the **LOAD DATA INFILE** command recalls the data from where the transactions csv file is saved. Then inserted to the transactions table in MySQL workbench.

NOTE: The transactions csv file, each value is terminated by a semi-colon (;) this should be noted in the query to retrieve the data without an error.

```
96     #loading products csv file
97 • CREATE TABLE IF NOT EXISTS products(
98         id INT PRIMARY KEY,
99         product_name VARCHAR(50),
100        price VARCHAR (50),
101        colour VARCHAR (20),
102        weight VARCHAR (10),
103        warehouse_id VARCHAR (10)
104    );
105
< -----
Output:
Action Output
# Time Action Message
1 21:30:02 CREATE TABLE IF NOT EXISTS products( id INT PRIMARY KEY, product_name VARCHAR(50), pric... 0 row(s) affected
102 • LOAD DATA INFILE 'C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\SPRINT 4\\products.csv'
103 INTO TABLE products
104 FIELDS TERMINATED BY ','
105 ENCLOSED BY '\"'
106 IGNORE 1 ROWS;
107
< -----
Output:
Action Output
# Time Action Message
1 11:04:54 LOAD DATA INFILE 'C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\SPRINT 4\\products.csv' INTO ... 100 row(s) affected Records: 100 Deleted: 0 Skipped: 0 Warnings: 0
```

Figure 1.0.7 Loading products csv file

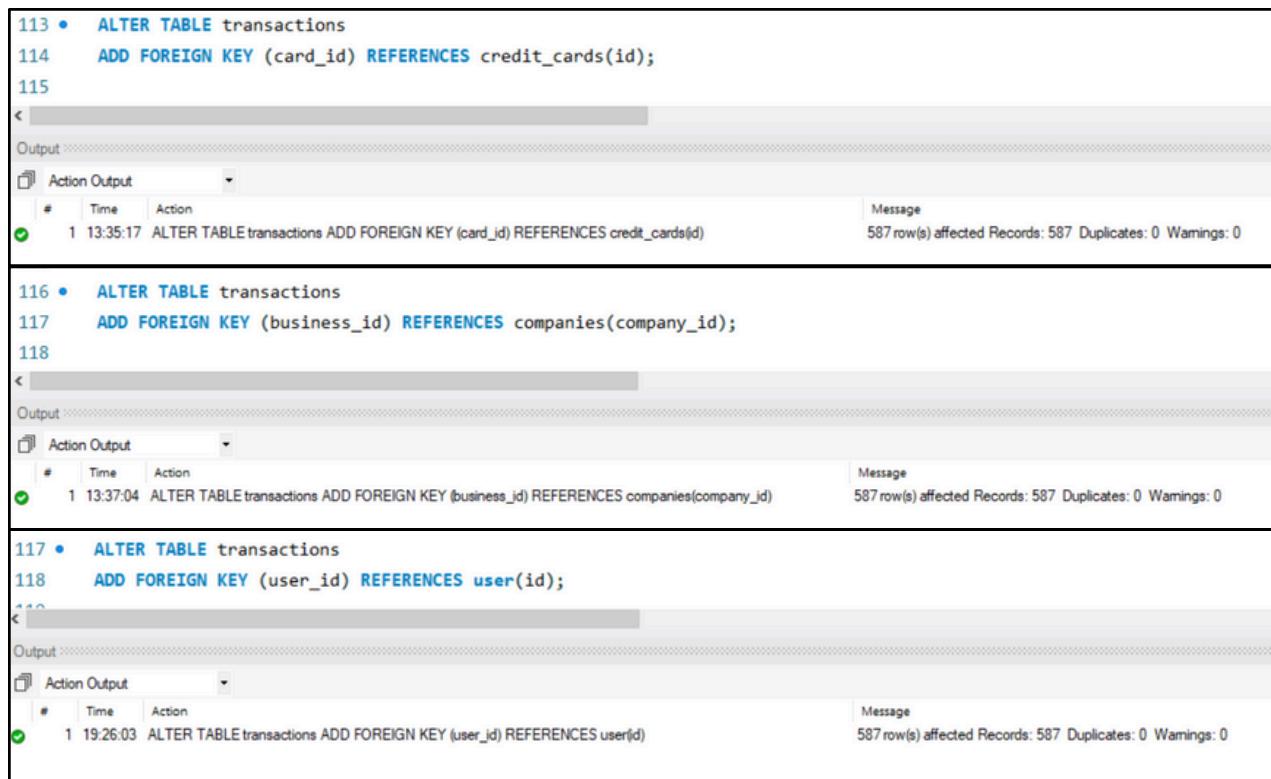
EXPLANATION:

After analyzing the products csv file by viewing it like an excel file, create a table after based from the data presented using CREATE TABLE command. Add all the column names and its data types. In this case, products table has this following columns and corresponding data types:

- **id INT PRIMARY KEY**
 - **product_name VARCHAR(50)**
 - **price VARCHAR (50)**
 - **colour VARCHAR (20)**
 - **weight VARCHAR (10)**
 - **warehouse_id VARCHAR (10)**

Using the LOAD DATA INFILE command recalls the data from where the products csv file is saved. Then inserted to the products table in MySQL workbench.

ADDING FOREIGN KEYS:



The screenshot shows three separate ALTER TABLE queries being run in MySQL Workbench:

- Query 113: `ALTER TABLE transactions ADD FOREIGN KEY (card_id) REFERENCES credit_cards(id);`. The output shows 587 rows affected.
- Query 116: `ALTER TABLE transactions ADD FOREIGN KEY (business_id) REFERENCES companies(company_id);`. The output shows 587 rows affected.
- Query 117: `ALTER TABLE transactions ADD FOREIGN KEY (user_id) REFERENCES user(id);`. The output shows 587 rows affected.

Figure 1.0.7 Foreign Keys

EXPLANATION:

After creating all the tables, analyze the data input in each table and decide the connections of the tables. In this case, the transaction table has a `card_id` column which is the `id` column in `credit_cards`. Moreover, it consists the `business_id` column, which is the `company_id` in `companies` table. In addition, `user_id` column in `transactions` is the `id` column in the `user` table.

To assign foreign keys, it's necessary to identify the table that needs to be altered in this case, the `transactions` table. Then using the `ADD FOREIGN KEY` command, identify the column which has the same values and use the table where the same values are found as a reference, and identify the name of the column in the referenced table.

NOTE: The `product_ids` column in `transactions` has more than one `product_id`. In this case, establishing relationship won't be possible without creating a new table.

TABLE RELATIONSHIPS:

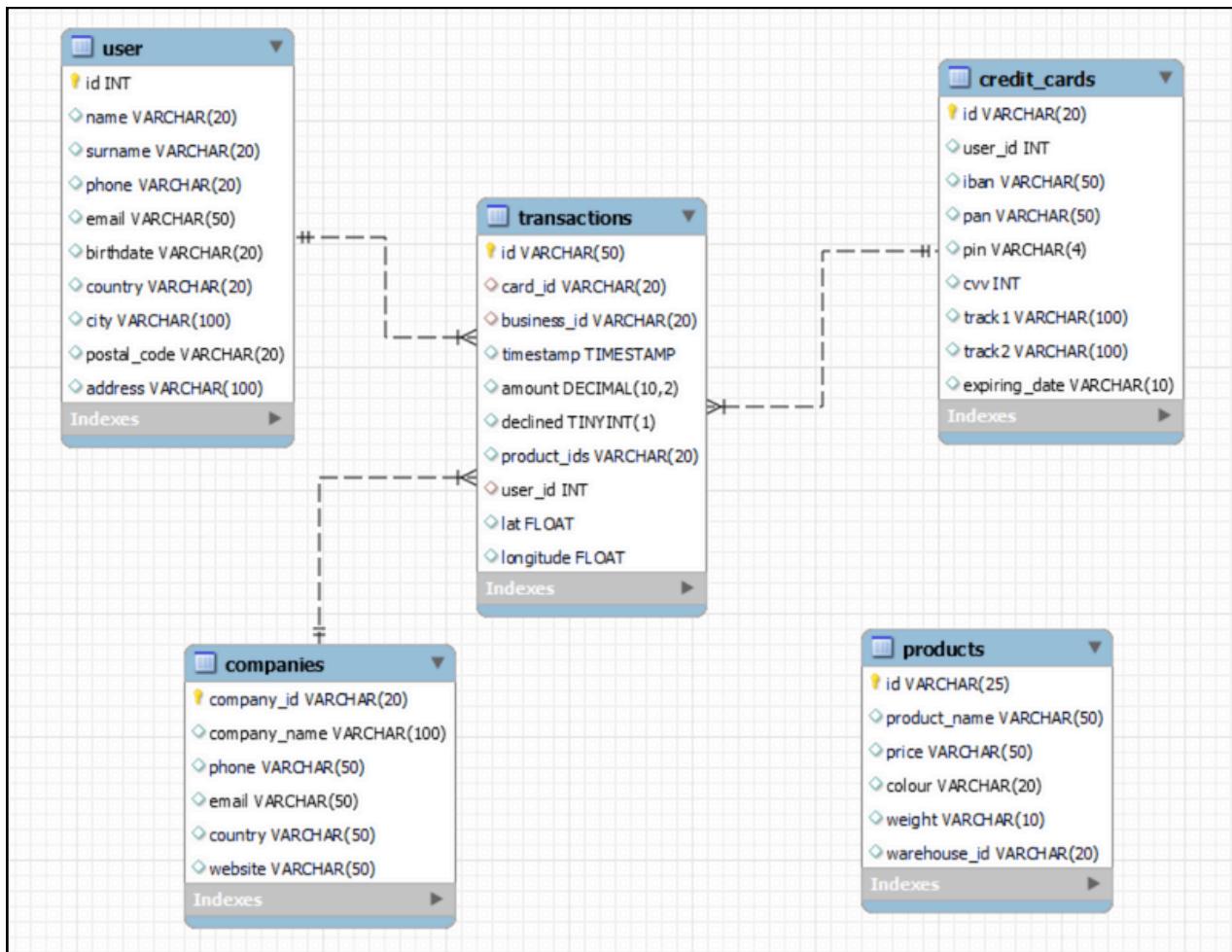


Figure 1.0.8 Relationships

EXPLANATION:

OVERALL RELATIONSHIPS: The database structure resembles a star schema, where the transactions table serves as the fact table because it contains more records and multiple foreign keys. Each of the companies, user, and credit_cards tables has a natural key.

ONE-TO-MANY RELATIONSHIPS:

- **Company to Transaction:** A company can have one or more transactions.
- **User to Transaction:** A user can have one or more transactions.
- **Credit Card to Transaction:** A credit card can be used in one or more transactions.

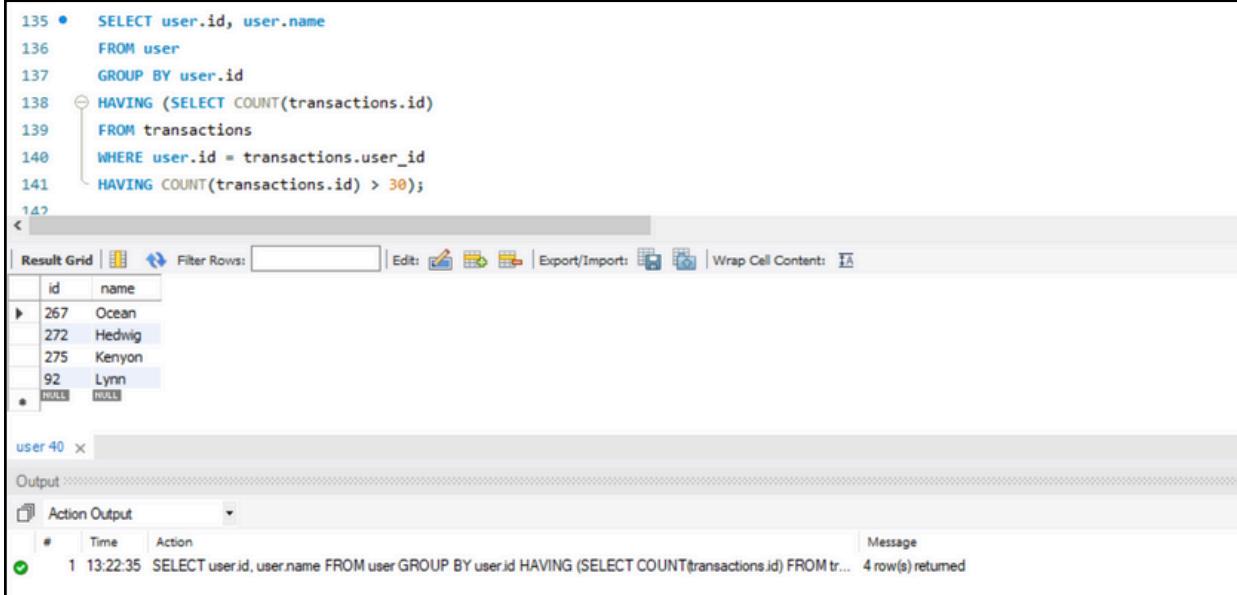
NOT ESTABLISHED RELATIONSHIP

The products table and transactions table don't have a well-established relationship due to the product_ids column. This column contains more than one ID from the products table. To establish a relationship, creating a new table that will serve as a bridge to connect these tables is necessary.

LEVEL 1

Exercise 1:

Do a subquery that shows all users with more than 30 transactions using at least 2 tables.



The screenshot shows a MySQL Workbench interface. The SQL editor contains the following query:

```
135 •  SELECT user.id, user.name
136   FROM user
137   GROUP BY user.id
138   HAVING (SELECT COUNT(transactions.id)
139   FROM transactions
140   WHERE user.id = transactions.user_id
141   HAVING COUNT(transactions.id) > 30);
142
```

The Result Grid displays the following data:

	id	name
▶	267	Ocean
▶	272	Hedwig
▶	275	Kenyon
▶	92	Lynn
*	NULL	NULL

The Output pane shows the command run and the message "4 row(s) returned".

Figure 1.1.1 Query + Result

EXPLANATION:

I fetched the results by selecting data from the user table and grouping the results by the user.id. Then I filtered these groups to only include users who have more than 30 transactions.

HAVING COMMAND

I used this to filter the groups based on a condition.

(SELECT COUNT(transactions.id) FROM transactions WHERE user.id = transactions.user_id):

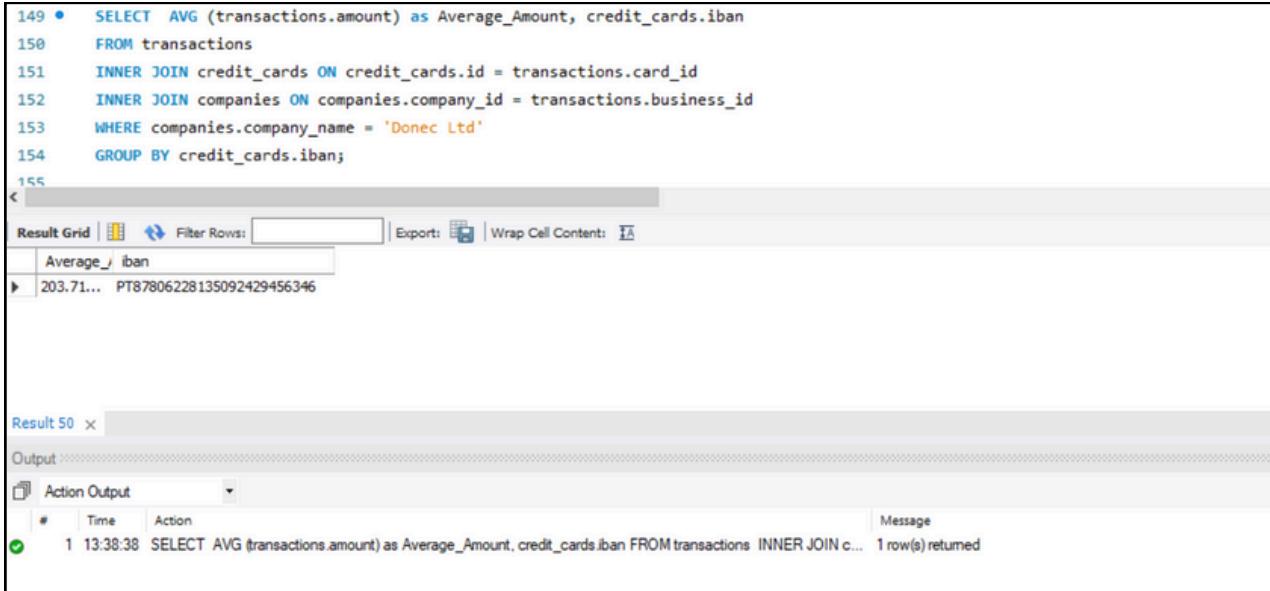
This subquery counts the number of transactions for each user.

HAVING COUNT(transactions.id) > 30: I only included users who have more than 30 transactions.

LEVEL 1

Exercise 2:

Show the average amount per IBAN of credit cards at the company Donec Ltd, using at least 2 tables.



The screenshot shows a MySQL query execution interface. The SQL code is as follows:

```
149 •  SELECT AVG (transactions.amount) as Average_Amount, credit_cards.iban
150   FROM transactions
151   INNER JOIN credit_cards ON credit_cards.id = transactions.card_id
152   INNER JOIN companies ON companies.company_id = transactions.business_id
153   WHERE companies.company_name = 'Donec Ltd'
154   GROUP BY credit_cards.iban;
155
```

The results grid displays one row:

Average_Amount	iban
203.71...	PT87806228135092429456346

The output log shows the query execution details:

#	Time	Action	Message
1	13:38:38	SELECT AVG (transactions.amount) as Average_Amount, credit_cards.iban FROM transactions INNER JOIN c...	1 row(s) returned

Figure 1.2.1 Query + Result

EXPLANATION:

I got the results by creating a query that calculates the average transaction amount for each credit card used in transactions with a specific company, 'Donec Ltd'. I started by selecting the average transaction amount from the transactions table, rounding this value to two decimal places and labeling it as Average_Amount. Additionally, the query retrieved the IBAN (International Bank Account Number) from the credit_cards table.

INNER JOIN credit_cards ON credit_cards.id = transactions.card_id:

I joined the transactions table with the credit_cards table using the card_id to match transactions to their respective credit cards.

INNER JOIN companies ON companies.company_id = transactions.business_id:

I joined the transactions table with the companies table using the business_id to identify the company involved in the transactions.

I filtered the results to include only those transactions where the company name is 'Donec Ltd'. Finally, the results are grouped by the IBAN, so the average transaction amount is calculated for each unique credit card.