



ETL Project - First Delivery: Credit Card Transactions

[Repository](#)

[Looker Dashboard](#)

[Notion Page](#)

[Kaggle Dataset](#)

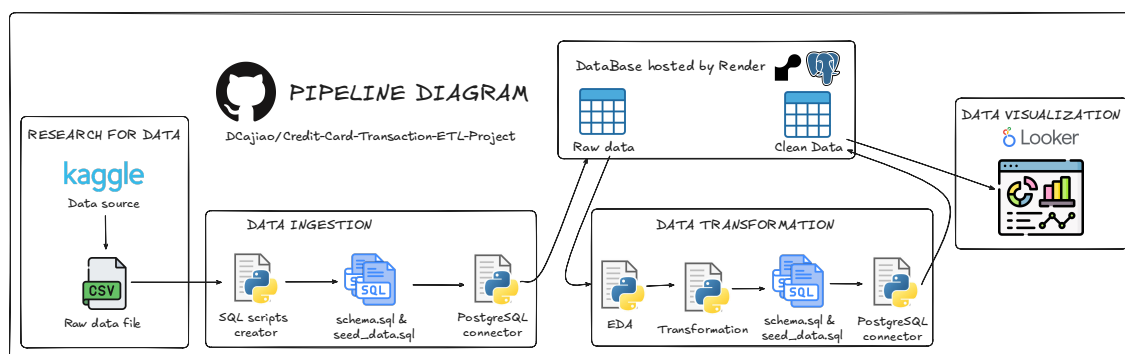
Team dev

- [David Alejandro Cajiao Lazt](#)
- [Sebastián Ortiz](#)
- [Juan Andrés López Alvarez](#)

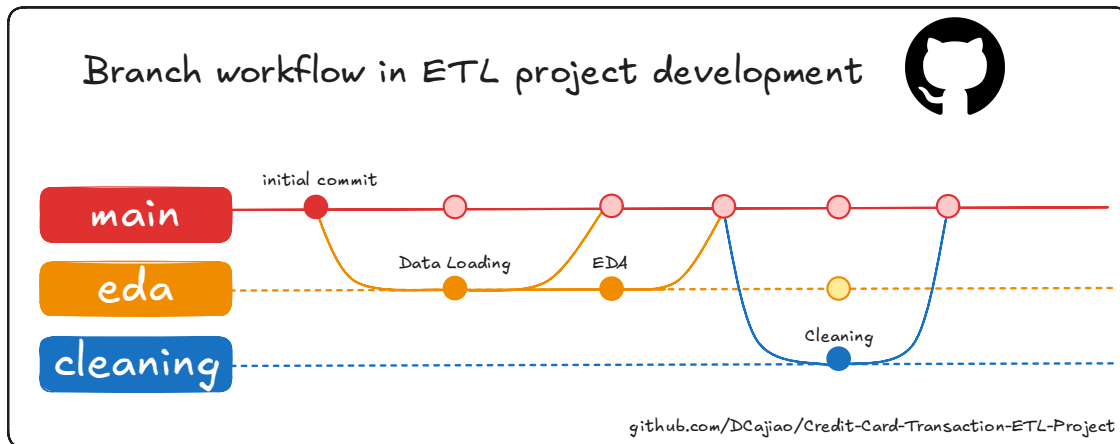
Introduction

This report provides a detailed overview of the ETL (Extract, Transform, Load) process performed in this project, focusing on the specific tasks performed at each step, the key results and the relevance of certain tools and scripts, such as `pysqlschema.py`. The goal was to build a robust pipeline for analyzing credit card transaction data, from data ingestion to final visualization.

The following diagram was designed for the pipeline to be created in the development of the project



In addition, being a team project, the following work plan was developed by branches



1. Data Loading ([00 data load.ipynb](#)).

What was done?

The first step in the ETL pipeline involved loading the raw candidate data into a PostgreSQL database. The raw data was provided in CSV format and needed to be uploaded to a relational database to facilitate structured queries and further analysis.

- **Database Setup:** A connection was established to a free PostgreSQL instance deployed on **Render**. This cloud instance enabled efficient and scalable remote access.
- **Implementation Documentation:** Detailed public documentation was written by [DCajiao](#) to guide other users in creating and deploying databases on Render. This document is an essential resource for those looking to replicate the development environment.
- **Data Ingestion:** The raw data was read using Pandas and inserted into the corresponding tables in the database. The schema for these tables was previously defined in the `schema.sql` file, which was applied before the data load using `seed_data.sql`. For this it was necessary to perform a batch upload due to the size of the df

Render

DashboardBlueprintsEnv Groups

+ New

DAVID ALEJANDRO CAJIAO LAZT

Configure and deploy your new Database

Getting startedService typeConfigureDeploy

Need help? Docs

Name
A unique name for your PostgreSQL instance.

credit-card-transactions

Database Optional
The PostgreSQL `dbname`.

creditcardtransactions

User Optional

dcajiao

Region
Your services in the same [region](#) can communicate over a [private network](#).

Oregon (US West)

PostgreSQL Version

16

Connections

Hostname
An internal hostname used by your Render services.

dpg-cqu2n8jqf0us73a3uimg-a

Port

5432

Database

creditcardtransactions

Username

dcajiao

Password

.....

Internal Database URL

.....

Render

DashboardBlueprintsEnv Groups


+ New

DAVID ALEJANDRO CAJIAO LAZT

Overview

Q Search services

Active (1)Suspended (0)All (1)

<input type="checkbox"/>	Service name	Status	Type	Runtime	Region	Last deployed ↑	
<input type="checkbox"/>	 credit-card-transactions	✓ Available	PostgreSQL	PostgreSQL 16	Oregon	12 hours ago	...

```
# Seed data by executing the seed data script in batches
db.execute_in_batches("../sql/seed_data.sql", batch_size=20000)
```

✓ 16m 59.4s

```
2024-08-24 15:47:42,250 - ✓ Connected to database
2024-08-24 15:47:57,498 - ✓ Executed a batch of 20000 records
2024-08-24 15:48:11,525 - ✓ Executed a batch of 20000 records
2024-08-24 15:48:25,106 - ✓ Executed a batch of 20000 records
2024-08-24 15:48:39,384 - ✓ Executed a batch of 20000 records
2024-08-24 15:49:02,796 - ✓ Executed a batch of 20000 records
2024-08-24 15:49:16,940 - ✓ Executed a batch of 20000 records
2024-08-24 15:49:29,923 - ✓ Executed a batch of 20000 records
2024-08-24 15:49:40,716 - ✓ Executed a batch of 20000 records
2024-08-24 15:50:01,950 - ✓ Executed a batch of 20000 records
2024-08-24 15:50:16,156 - ✓ Executed a batch of 20000 records
2024-08-24 15:50:29,853 - ✓ Executed a batch of 20000 records
2024-08-24 15:50:45,723 - ✓ Executed a batch of 20000 records
2024-08-24 15:51:07,141 - ✓ Executed a batch of 20000 records
2024-08-24 15:51:21,095 - ✓ Executed a batch of 20000 records
2024-08-24 15:51:31,892 - ✓ Executed a batch of 20000 records
2024-08-24 15:51:43,005 - ✓ Executed a batch of 20000 records
2024-08-24 15:52:05,710 - ✓ Executed a batch of 20000 records
2024-08-24 15:52:19,108 - ✓ Executed a batch of 20000 records
2024-08-24 15:52:32,298 - ✓ Executed a batch of 20000 records
2024-08-24 15:52:50,531 - ✓ Executed a batch of 20000 records
2024-08-24 15:53:11,501 - ✓ Executed a batch of 20000 records
2024-08-24 15:53:25,222 - ✓ Executed a batch of 20000 records
2024-08-24 15:53:38,885 - ✓ Executed a batch of 20000 records
2024-08-24 15:54:01,605 - ✓ Executed a batch of 20000 records
2024-08-24 15:54:19,219 - ✓ Executed a batch of 20000 records
2024-08-24 15:54:32,855 - ✓ Executed a batch of 20000 records
2024-08-24 15:54:46,171 - ✓ Executed a batch of 20000 records
2024-08-24 15:55:00,209 - ✓ Executed a batch of 20000 records
2024-08-24 15:55:13,775 - ✓ Executed a batch of 20000 records
```

The screenshot shows the pgAdmin 4 interface. On the left, the Object Explorer shows the database structure. The main pane displays a SQL query: `SELECT * FROM credit_card_transactions LIMIT 10;`. Below the query, the Data Output tab shows the results of the query. The results are displayed in a table with 10 rows and 10 columns. The columns are: `Unnamed: 0` (integer), `trans_date_trans_time` (text), `cc_num` (bigint), `merchant` (text), `category` (text), `amt` (double precision), `first` (text), `last` (text), `gender` (text), and `street` (text). The status bar at the bottom indicates: "Total rows: 10 of 10 Query complete 00:00:00.681 Ln 3, Col 10". A green message box at the bottom right says: "Successfully run. Total query runtime: 681 msec. 10 rows affected."

Unnamed: 0	trans_date_trans_time	cc_num	merchant	category	amt	first	last	gender	street
0	2019-01-01 00:00:18	2703186189652095	fraud_Ripplin, Kub and Mann	misc_net	4.97	Jennifer	Banks	F	561 Perry C
1	2019-01-01 00:00:44	630423337322	fraud_Heller, Gutmann and Zieme	grocery_pos	107.23	Stephanie	Gill	F	43039 Riley
2	2019-01-01 00:00:51	38859492057661	fraud_Lind-Buckridge	entertainment	220.11	Edward	Sanchez	M	594 White I
3	2019-01-01 00:01:16	3534093764340240	fraud_Kutch, Hermiston and Farrell	gas_transport	45	Jeremy	White	M	9443 Cynth
4	2019-01-01 00:03:06	375534208663984	fraud_Keeling-Crist	misc_pos	41.96	Tyler	Garcia	M	408 Bradle
5	2019-01-01 00:04:08	4767265376804500	fraud_Stroman, Hudson and Erdm...	gas_transport	94.63	Jennifer	Conner	F	4655 David
6	2019-01-01 00:04:42	30074693890476	fraud_Rowe-Vandervort	grocery_net	44.54	Kelsey	Richards	F	889 Sarah
7	2019-01-01 00:05:08	6011360759745864	fraud_Corwin-Collins	gas_transport	71.65	Steven	Williams	M	231 Flores
8	2019-01-01 00:05:18	4922710831011201	fraud_Herzog Ltd	misc_pos	4.27	Heather	Chase	F	6888 Hicks
9	2019-01-01 00:06:01	2720830304681674	fraud_Schoen, Kuphal and Nitzsc...	grocery_pos	108.30	Melissa	Reuter	F	21326 Tayl

2. Data Exploration (01_EDA.ipynb)

What was done?

The second step involved exploring the data to understand its structure, content, and potential issues. This step was crucial to identifying the necessary transformations to clean and prepare the data for analysis.

- **Initial Query:** SQL queries were executed directly on the database to obtain the dataframe and process it further from the local environment.
- **Exploratory Data Analysis (EDA):** An EDA was performed using Pandas to generate descriptive statistics, identify outliers, and visualize distributions. This helped to understand the overall shape and characteristics of the dataset.

Key Findings

Results:

1. The dataset has 1.2M records and 24 columns.
2. We have 1 candidate column to be the ID (`Unnamed: 0`).
3. We have 4 relevant categorical columns (`category`, `gender`, `state`, `job`, `is_fraud`).
4. We have 2 relevant numerical columns (`cc_num`, `amt`).
5. We have 2 relevant date columns (`trans_date_trans_time`, `dob`)

3. Data Cleaning (02_cleaning.ipynb)

What was done?

Data cleaning was an essential step to ensure the quality and usability of the dataset for analysis. The process involved the following steps:

- Column `Unnamed: 0` should be renamed to `id`.

From the EDA we identified that this column was the best candidate to become the records ID.

- The `trans_date_trans_time` column should be loaded as `datetime`.
- The `dob` column should be loaded as `datetime`.
- Column `cc_num` should be loaded as string.

From the EDA we identified that this column was the best candidate to become the customer ID.

- Remove the `unix_time`, `city_pop`, `merch_lat`, `merch_long`, columns.

We used only certain columns to perform the analysis and these were discarded.

- Remove records with null values.

From the EDA we knew that the `zipmerchan` column was the one with nulls, and that was because they were virtual merchants, so we decided to focus our analysis on physical merchants.

- Convert `is_fraud` to boolean.
- Calculate the `age` of the customers and convert it to a new column.
- Remove records whose `age` is less than 21 years old.

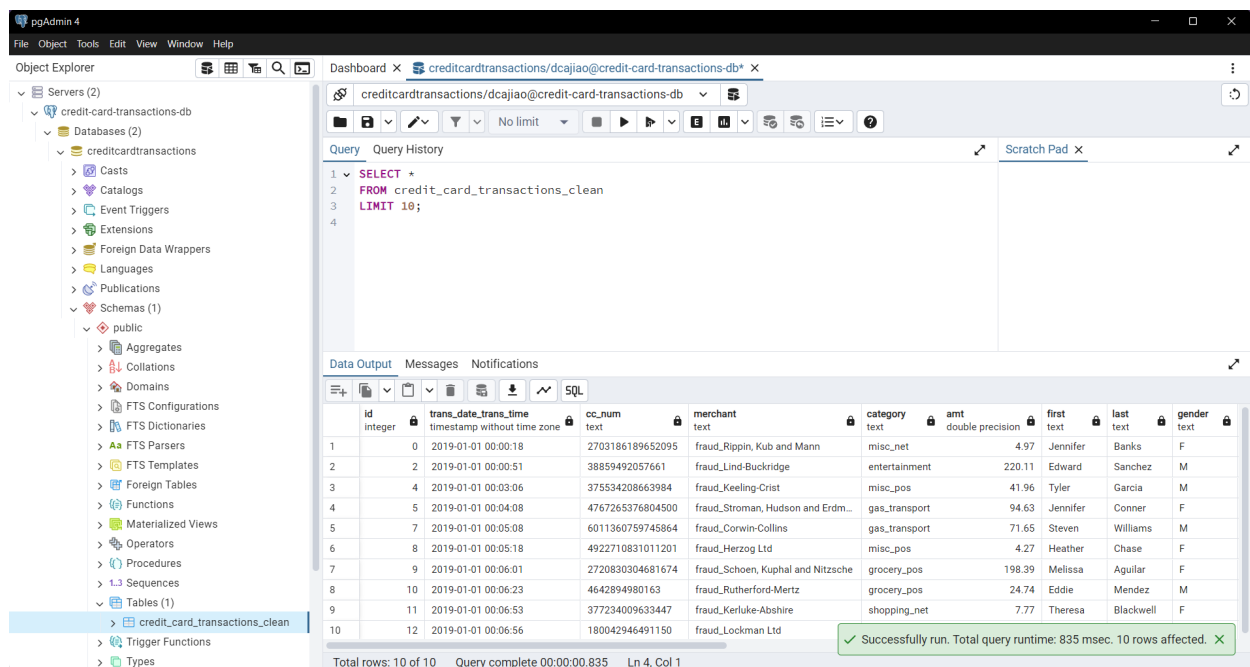
This is because, at least in the USA, people can use credit cards from the age of 21.

Afterward, we proceeded with the Database Update: After cleaning, the data was updated in the database in the `candidates_cleaned` table, ensuring that the database reflected the cleanest and most prepared version of the data. In addition, it was necessary to immediately delete the previous table because the free render instance we are using did not allow us that much space.

```
# Seed data by executing the seed data script in batches
db.execute_in_batches("../sql/seed_data_clean.sql", batch_size=20000)

✓ 11m 2.1s

2024-08-26 23:01:59,927 - ✓ Connected to database
2024-08-26 23:02:12,116 - ✓ Executed a batch of 20000 records
2024-08-26 23:02:23,299 - ✓ Executed a batch of 20000 records
2024-08-26 23:02:34,703 - ✓ Executed a batch of 20000 records
2024-08-26 23:02:45,818 - ✓ Executed a batch of 20000 records
2024-08-26 23:02:57,664 - ✓ Executed a batch of 20000 records
2024-08-26 23:03:17,063 - ✓ Executed a batch of 20000 records
2024-08-26 23:03:28,393 - ✓ Executed a batch of 20000 records
2024-08-26 23:03:39,520 - ✓ Executed a batch of 20000 records
2024-08-26 23:03:50,597 - ✓ Executed a batch of 20000 records
2024-08-26 23:04:04,328 - ✓ Executed a batch of 20000 records
2024-08-26 23:04:20,537 - ✓ Executed a batch of 20000 records
```

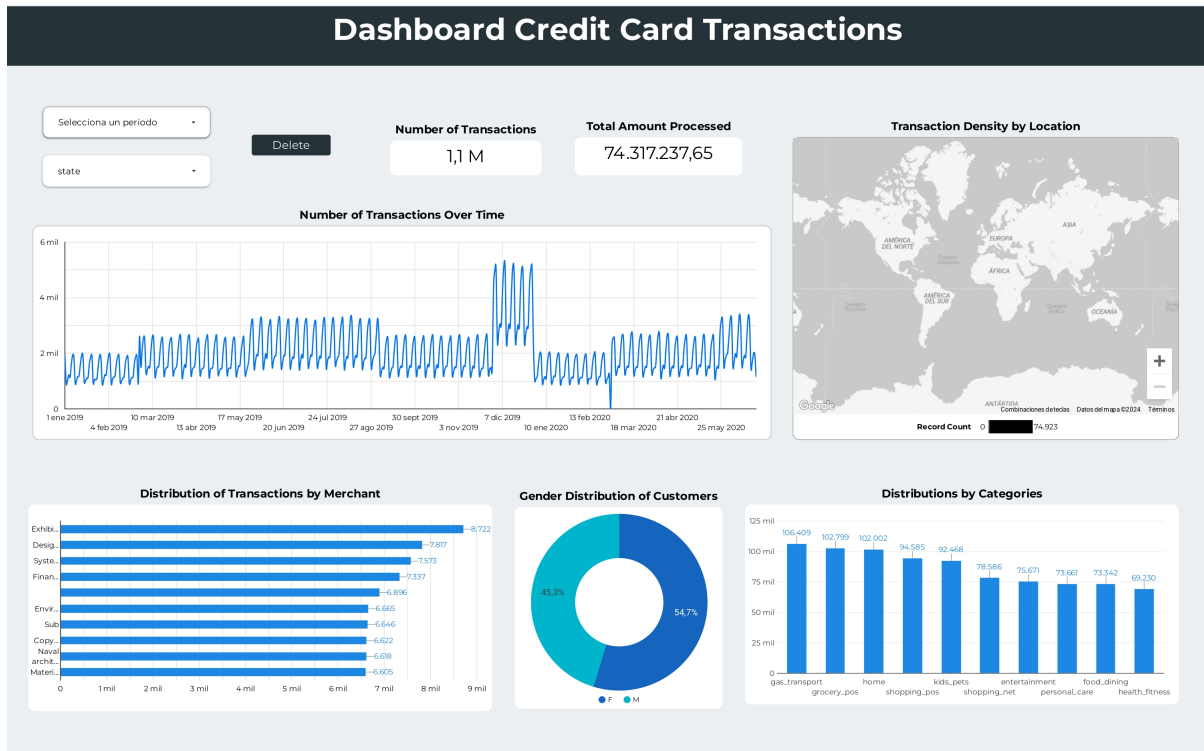


4. Data Visualization (ETL-Credit-Card-Transaction-Dashboard)

What was done?

The final step involved creating visualizations to meaningfully present the cleaned and processed data. These visualizations were designed to uncover trends, patterns, and insights that could guide decision-making.

- **Enhancement of Charts in Looker:** These visualizations were later enhanced in **Looker Studio**, where a more interactive and visually appealing dashboard was built.



- **Automatic Connection with the Cloud Database:** The advantage of using a cloud database, such as the instance on Render, allowed Looker to connect automatically, facilitating real-time updates to the dashboards as data was processed.

PostgreSQL - creditcardtransactions 🕒 Share ? 🧑

[VOLVER A CONECTAR](#)

CAMPOS →

PostgreSQL
De Google

El conector de PostgreSQL te permite acceder a la información de las bases de datos de PostgreSQL desde Looker Studio. Este conector usa el controlador JDBC de PostgreSQL para conectar una fuente de datos de Looker Studio a una sola tabla de base de datos de PostgreSQL.

[MÁS INFORMACIÓN](#) [NOTIFICAR UN PROBLEMA](#)

BÁSICA	Autenticación de la base de datos	TABLAS	Tabla
URL DE JDBC	<p>Nombre de host o IP dpg-cqu2n8jqf0us73a3uimg-a.oregor</p> <p>Puerto (opcional) 5432</p> <p>Base de datos creditcardtransactions</p> <p>Nombre de usuario dcajiao</p> <p>Contraseña</p>	CONSULTA PERSONALIZADA	credit_card_transactions_clean

5. Additional Relevant Aspects

PySQLSchema (pysqlschema.py)

One of the most notable components of this project was the use of pysqlschema.py, a custom script developed by [DCajiao](#) to programmatically manage SQL schemas using Python. This script was essential in ensuring that the database schema remained consistent throughout the project.

- **Schema Management:** `pysqlschema.py` enabled the dynamic creation, validation, and migration of database schemas. This was particularly useful during the data loading and cleaning phases, where the schema needed to be adjusted based on the characteristics of the data.
- **Automation:** The script automated various aspects of schema management, reducing the potential for human error and ensuring that schema changes were systematically applied across the entire database.

Relevance

- **Agile Development:** Thanks to the use of `pysqlschema.py`, the project benefited from an agile development process, where schema changes could be quickly implemented and tested without manual intervention.
 - **Consistency and Integrity:** The script played a crucial role in maintaining the consistency and integrity of the database, ensuring that all data transformations adhered to the defined schema.
-