CREDIT CARD TRANSACTION SIMULATION & CHURN PREDICTION

AGUSTIN DANIEL RIZZO

2025-02-23

INTRODUCTION:

This project involves creating a synthetic database that simulates the data structure of a credit card company. The database is built using MySQL following a relational model. Daily, credit card transactions are simulated and stored in MySQL, orchestrated through Apache Airflow. Additionally, monthly payments are processed for every credit card in the database.

Each month, Airflow executes queries and procedures to analyze customer spending and payment patterns. This extracted information is then used to feed a machine learning model, trained on real-world data, to predict potential customer churn. Finally, a PDF report is automatically generated, summarizing key insights on customer churn patterns.

BUSINESS LOGIC:

The following rules define the credit card transaction model:

- The client portfolio consists of **500 fixed clients**.
- Each client can hold up to **6 credit cards**.
- Credit card limits are assigned based on the card category, ranging from **$100 to $5000**.
- Clients are randomly assigned a **sign-up date (date_on)** within the past **three years**.
- Each card has an **issuance date** (always after the client's sign-up date) and an **expiration date** set **five years later**.
- The initial dataset includes **5,000 transactions over the past two years**.
- Consumption amounts are randomly generated but cannot exceed **30% of the credit limit**.
- There are **two types of clients**:
    - **20% of clients** fully pay off their balances every month (**debt-free cards**).
    - **80% of clients** carry over varying amounts of debt each month.
- Transactions occur between the **1st and 20th of each month**, while payments are processed between the **20th and 28th**.
- A MySQL stored procedure ensures that transactions exceeding the credit limit are automatically **rejected** and corresponding transactions for that month are deleted.

DATABASE STRUCTURE:

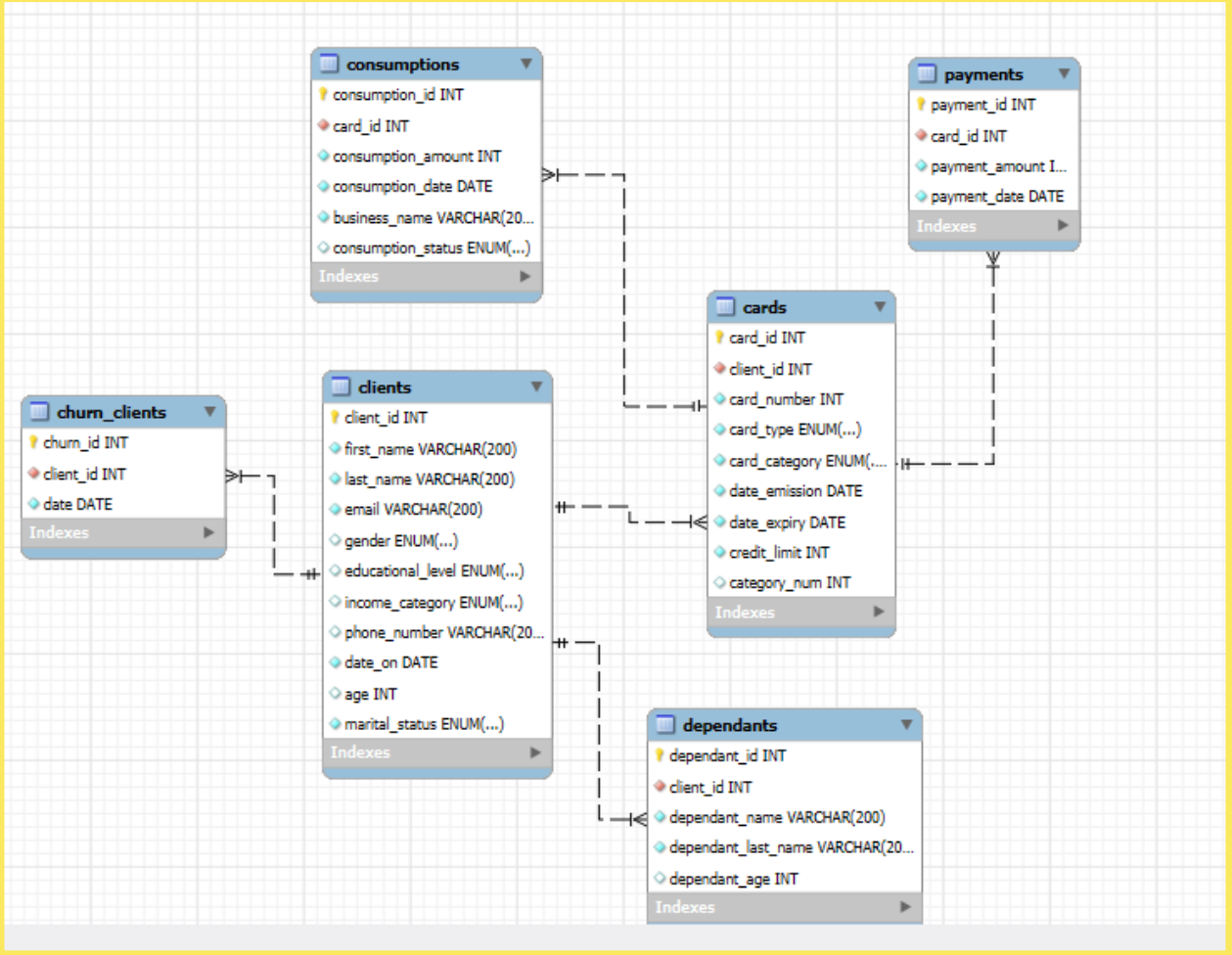

Table description:

1. Clients (Dimensional Table)

Stores general information about clients.

- **client_id** (Primary Key)
- **first_name, last_name, email, phone_number** (VARCHAR)
- **gender, marital_status, educational_level, income_category** (ENUM)
- **date_on** (DATE) – Sign-up date
- **age** (INTEGER)

2. Dependents (Dimensional Table)

Stores information on financially dependent individuals.

- **dependent_id** (Primary Key)
- **client_id** (Foreign Key)
- **name, last_name, age** (VARCHAR)

3. Cards (Dimensional Table)

Stores credit card details.

- **card_id** (Primary Key)
- **client_id** (Foreign Key)
- **card_type** (ENUM: VISA, MASTERCARD)
- **card_category** (ENUM: Blue, Silver, Gold, Platinum)
- **date_emission, date_expiry** (DATE)
- **credit_limit** (INTEGER)

4. Transactions (Transactional Table)

Records all credit card transactions.

- **transaction_id** (Primary Key)
- **card_id** (Foreign Key)
- **transaction_amount** (INTEGER)
- **business_name** (VARCHAR)
- **transaction_date** (DATE)
- **transaction_status** (ENUM: Accepted, Rejected)

5. Payments (Transactional Table)

Stores monthly payment records.

- **payment_id** (Primary Key)
- **card_id** (Foreign Key)
- **payment_amount** (INTEGER)
- **payment_date** (DATE)

6. Churn Clients

Stores clients predicted as potential churners.

- **churn_id** (Primary Key)
- **client_id** (Foreign Key)
- **date** (DATE)

PROJECT STRUCTURE:

```
Create_Data/
├── CSV_Files/
├── dags/
│   ├── daily_tasks/
│   │   ├── daily_CSV/
│   │   ├── clean_daily_data.sql
│   │   ├── create_daily_consumptions.py
│   │   ├── load_daily_data.py
│   │   ├── MOCK_names.py
│   │   └── validate_daily_consumption.sql
│   ├── monthly_tasks/
│   │   ├── monthly_data_csv/
│   │   ├── monthly_payments_csv/
│   │   ├── monthly_reports/
│   │   ├── sql_scripts/
│   │   │   ├── procedure_[1-2].sql
│   │   │   └── view_[1-15].sql
│   │   ├── cards_debtless.csv
│   │   ├── create_monthly_payment.py
│   │   ├── create_monthly_report.py
│   │   ├── download_monthly_data.py
│   │   ├── load_monthly_data.py
│   │   ├── ml_pipeline.pkl
│   │   ├── plot.png
│   │   └── predict_churn.py
│   ├── dag_[1-4].py
├── logs/
├── plugins/
├── schemas/
├── Create_Data.py
└── credentials.py
```

PROJECT WORKFLOW:

Step 1: Database Initialization

1. **Generate synthetic client and transaction data** using Create_Data.py.
2. **Store data in MySQL** using initiate_db.py.
3. **Validate and clean data** using MySQL stored procedures to ensure credit limits are not exceeded.
   ! For further details check table 1

Step 2: Database Maintenance via Airflow DAGs

DAG 1: Daily Transactions (Executed 1st–20th each month)

- Simulates **50 daily transactions**.
- Loads transactions into MySQL.
- Validates credit limits and rejects over-limit transactions.

DAG 2: Monthly Payments (Executed 28th each month)

- Processes **monthly payments** based on transaction history.
- Loads payments into MySQL.

DAG 3: Data Extraction & Churn Prediction (Executed last day each month)

- Generates SQL **views** to calculate spending patterns and credit utilization.
- **Predicts churn** using a pre-trained machine learning model (ml_pipeline.pkl).
- Stores churn predictions in MySQL.
  ! For further details check table 2

DAG 4: PDF Report Generation (Executed last day each month)

- Creates an **automated PDF report** summarizing potential churners.

Machine Learning Model

- **Dataset:** Trained on **real customer churn data** from Kaggle.
- **Training & Preprocessing:** Available on GitHub: https://github.com/AgusDRizzo/Proyecto_final_DS_1
- **Model:** Serialized using Pickle (ml_pipeline.pkl).
- **Prediction:** predict_churn.py applies the model on newly extracted data.

Table 1: Functions used to create the initial database. They are contained in the file Create_Data and called through the file initiate_db.

Name	Description
Generate_credit_cards(total_clients, max_cards, start_date, max_limit, min_limit)	Creates client data, card data, and dependent data, storing each dataset in a separate dataframe.
Generate_consumptions(df, total_consumptions, year1)	Uses the card data from Generate_credit_cards to create consumption records for cards issued before year1, ensuring chronological consistency. Stores the results in a dedicated dataframe.
Generate_payments(df_consumptions, df, total_clients)	Processes consumption data to generate two types of payment patterns: full monthly payments for debtless cards (20%) and partial payments for the rest based on a randomly selected percentage.
generate_consumptions_and_payments(df, total_consumptions, year1, total_clients)	Calls Generate_consumptions and Generate_payments, returning two dataframes containing consumption and payment data.
menu_generate_consumptions_and_payments()	Stores all parameters defining business logic and executes all related functions. Saves data as CSV files in Create_Data\CSV_Files and records debtless cards in Create_Data\dags\monthly_tasks to maintain consistency in database maintenance.

Table 2: MySQL views and queries automatically run every month by airflow

Name	Description
View_1	Calculates the total revolving credit balance per card over the past year.
View_2	Computes the total revolving credit balance per client over the past year.
View_3	Identifies the highest card category assigned to each client.
View_4	Sums the total credit limit of all cards held by each client.
View_5	Determines the total number of dependents for each client.
View_6	Calculates the number of months a client was inactive over the past year.
View_7	Measures the duration (in months) of the client's relationship with the bank.
View_8	Counts the number of cards held by each client.
View_9	Computes the total amount spent on transactions by each client over the past year.
View_10	Calculates the fold change in transaction spending between the last quarter and the first quarter of the year.
View_11	Determines the total number of transactions made by each client over the past year.
View_12	Computes the fold change in transaction count between the last quarter and the first quarter of the year.
View_13, View_14,View_15	Perform joins to merge previously calculated information.
Procedure_1	Creates a table called open_credit_per_month, computing the available credit for a specific card_id based on the revolving balance up to that month and the card's maximum credit limit.
Procedure_2	Iterates Procedure_1 for each card_id in the database and stores the results in open_credit_per_month. This data is utilized in **View_1** and **View_2** to calculate the average monthly open credit per client over the past year.

QUICKSTART GUIDE:

Follow these steps to run the project in your local environment.

1. Install docker desktop. You may also install MySQL workbench. Although it is not mandatory, it may provide you with a better visual interaction with the database.
2. Within a terminal, set the directory inside Create_Data folder and run the docker container using the command: `docker compose up -d`. This way docker engine will start and lift airflow and mysql image together with some accessory services.
3. Create a virtual environment and activate it. Then replicate the virtual environment specified in the requirements.txt file using the command: `pip install -r requirements.txt`. This will guarantee that your environment has all dependencies required to initiate the database.
4. Initiate database by running the command `python run initiate_db.py`. Automatically an initial database will be created in mysql. You may check this by accessing the database through MySQL workbench. After the script running finishes, CSV files will appear in the folder CSV_Files. Moreover, a csv called debtless_cards will be generated inside folder daily_tasks. At this point, step 1 is completed.
5. Access airflow UI by typing in the browser http://localhost:8080/. Access through the username and password (username:airflow, password:airflow). You may change username and password by editing the variables in the docker compose file.
6. Create a mysql connection: go to Admin/connections/add a new record. Complete the form with the following data:
    - Connection Id: mysql_conn
    - Connection Type: MySQL
    - Host: db
    - Schema: credit_card_db
    - Login: root
    - Password: admin (you may change the password editing the variable in the docker compose file)
    - Port: 3306
    - Extra: {"allow_local_infile": true}
7. Now you may activate dags by clicking the grey button next to the dag name in the DAGs section.
8. If you do not want to wait for a complete month in order to generate the report you may use airflow's backfill function in order to generate predictions based on data from the past. First backfill dag_3 using the command: `Docker exec -it create_data-airflow-scheduler-1 airflow dags backfill dag_3 -s <start_date> -e <end_date>`. For example, if you start running the

project on february of 2025, you may backfill dag_3 one month backwards using the command: Docker exec -it create_data-airflow-scheduler-1 airflow dags backfill dag_3 -s 2025-01-01 -e 2025-02-01. It's advaisable to backfill dag_3 one month at a time, because if not syncronization issues may arise. After each backfill, a csv called ML_DATA_<year>_<month> will appear inside the folder monthly_data_csv.

9. After backfilling dag_3 for a couple of months, you may backfill dag_4 using the command: : Docker exec -it create_data-airflow-scheduler-1 airflow dags backfill dag_4 -s <start_date> -e <end_date>. For example, if you start running the project on february of 2025, you may backfill dag_3 one month backwards using the command: Docker exec -it create_data-airflow-scheduler-1 airflow dags backfill dag_4 -s 2025-01-01 -e 2025-02-01. Automatically, a PDF report called report_<year>_<month> will be generated inside the folder monthly_reports.