Universidad Europea

Welford Bank

Data Engineer Report
DATA GENERATION

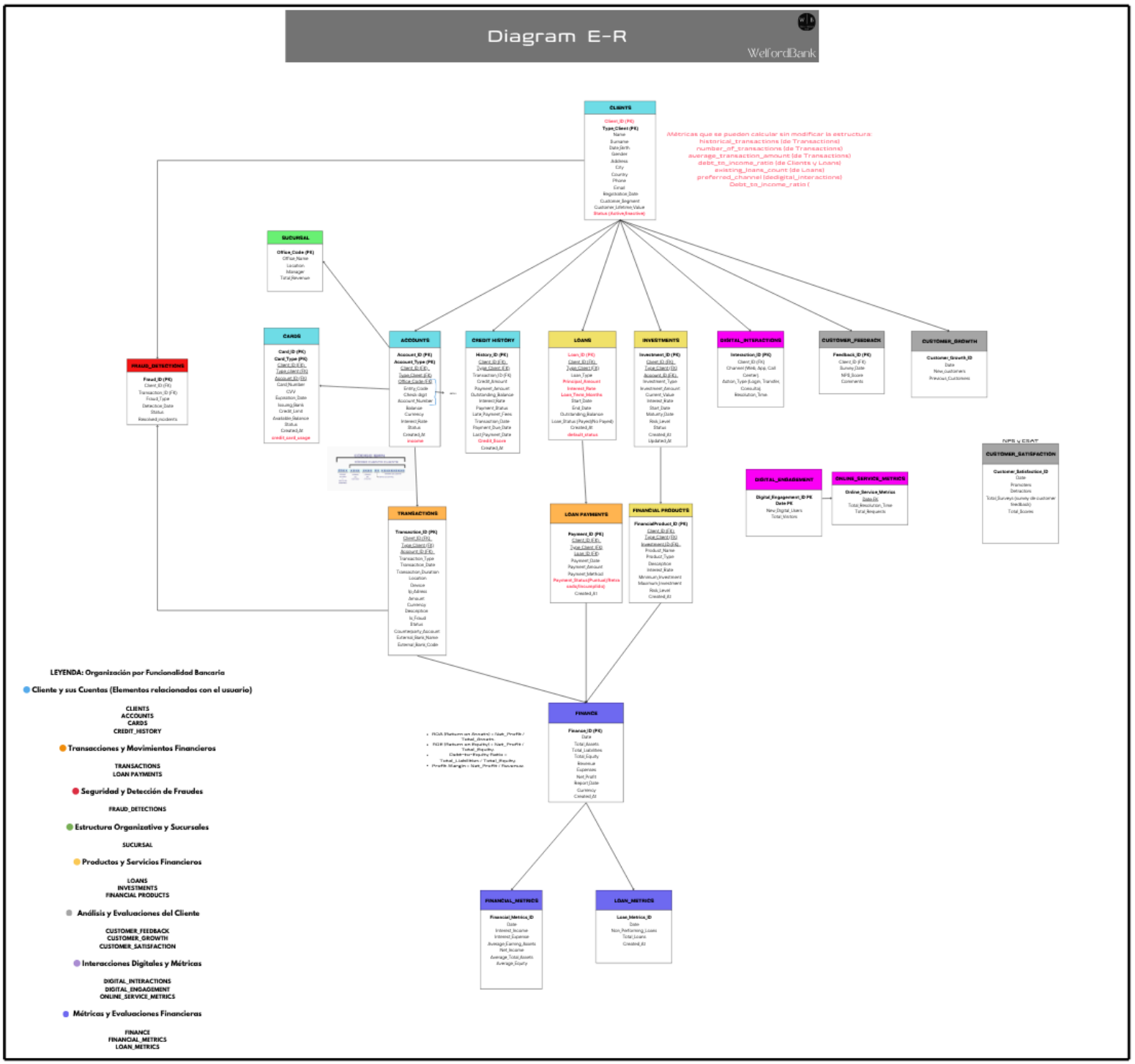
DATA SCIENCE
Alejandro Perdiguero

Data engineer team:
Javier Osuna
Francisco Galiano
Wail Achalhi
Daniela Correa

Universidad Europea

1. Database schema

Definitive Structure



https://www.canva.com/design/DAGeu4mYXBE/Tjy5gn_3_JO7V2X0IkPgFQ/view?utm_content=DAGeu4mYXBE&utm_campaign=designshare&utm_medium=link2&utm_source=uniquelinks&utlId=h8f21e687d4

This is our WelfordBank data schema. We've included a link for a better view. This schema (ER Diagram) was created at the beginning of the project. It explains all of WelfordBank's tables, including their fields, relationships, and primary and foreign keys.

This data schema has a color legend, with each table referring to a section of the bank. We also explain how to obtain certain metrics or data that we believe are important.

In total we have 20 tables, the main one being the CLIENTS table.

2. Library Documentation

pandas
Core of the pipeline. Every synthetic table is assembled as a `DataFrame`, merged where foreign keys apply, summarised with `groupby`, and finally exported to CSV. The library's vectorised operations make it possible to generate 20 + tables with millions of rows in a few seconds.

numpy
Supplies fast, vectorised random-number generation. Gaussian draws create Customer Lifetime Value; Poisson distributions decide "how many loans per client"; a cumulative random walk produces daily assets and liabilities. Using NumPy keeps runtime under three minutes on a laptop.

faker
Generates plausible Spanish data—names, companies, addresses, IBANs, feedback sentences—under the locale `es_ES`. This avoids obviously fake placeholders and gives each record internal coherence (e.g., a city that matches its province).

random (std-lib)
Used for lightweight probabilistic choices when vectorisation is unnecessary: weighted selection of customer segments, device types, fraud flags, or payment status.

datetime / timedelta
Handles all date arithmetic: account-opening dates between client registration and today, loan maturity calculations, 24-hour windows for digital interactions, and weekly/monthly time-series indices.

pathlib
Provides OS-agnostic path handling so the script runs unchanged on Windows, macOS, or Linux. Each CSV is written with `Path.cwd() / filename.csv`, eliminating hard-coded separators.

unicodedata
Normalises strings to ASCII when building unique e-mail addresses. By stripping accents and spaces, `faustino.haro@…` can be compared case-insensitively, preventing collisions during generation.

ipaddress
Creates valid IPv4 addresses for digital transactions and web interactions, adding a benign layer of realism for later network-analytics use cases.

you
Minimal use for environment checks: verifying whether a CSV already exists to avoid accidental overwrites and reading any optional configuration variables.

Together these open-source libraries let us generate a fully relational, reproducible dataset that satisfies the stated volume (> 25 k clients; > 1.8 M transactions) and variety (25 cross-linked tables) requirements within roughly three minutes of runtime.

3. Data Generation Code

The final version of **Tables.ipynb** (GenerateDataEnglish) produces the 20 CSVs with SEED = 15, ensuring repeatability.

It runs from start to finish in**≈ 2 min 37s** on an i7 / 16 GB RAM laptop and generates all files directly in/DataEngineer /GenerateDataEnglish.

- **clients_welfordbank_en.csv** – 25,001 files · ≈ 5 MB
- **accounts_welfordbank_en.csv** – 33,776 files · ≈ 9 MB
- **transactions_welfordbank_en.csv** – 1,048,576 files · ≈ 362 MB
- **loan_payments_welfordbank_en.csv** – 1,048,576 files · ≈ 593 MB
- **loans_welfordbank_en.csv** – 87 136 files · ≈ 28 MB
- **fraud_detections_welfordbank_en.csv** – 35,447 files · ≈ 8 MB
- **Rest of tables (14)**– 1 k – 30 k rows · ≤ 10 MB each

The limit of 1,048,576 rows corresponds to the Excel limit; a cutoff was imposed to prevent files larger than 600 MB.

The notebookTables.ipynb contains**all code commented out**, organized by table sections.

Structure that we have in the notebook:

#a. General bookstores

#b. Notebook/Environment Configuration

#c. Shared enumerations/Reference lists

#d. Helpers:

#and. MAIN TABLE CUSTOMERS

#f. Rest of tables, by sections

Within each table we have this order:

 # 1. Parameters and reference lists

 # 2. Helpers and generator functions

 # 3. Generation loop

 # 4. DataFrame = pd.DataFrame(rows)

 # 5. Export DataFrame.to_csv("<table>_welfordbank_en.csv")

4. Data generation rules

Below, all the hypotheses and distributions used to create the synthetic set of theWelford Bank Project.

Requirements for data generation:

- We have said that a minimum of 25,000 rows are created, because in clients we have generated 25,000 rows.
- The bank was created on 1-1-2020, therefore there can be nothing before that date.
- The CSV files are saved with a specific name.
- Use lists to be reused.
- Branch: There are 1-4 per city more or less, proportional to the number of accounts

- Maximum amounts : `Amount` ≤ 15 000 €, `Balance` ≤ 150 000 €.
- We have separated the clients into segments: Platinum, Gold, Silver, Bronze, along with a weight.
- Use weights so that when the data is generated, it is not totally random.

5. Final deliverable

For the work to be deliverable:

- All datasets are exported to CSV with the pattern:

  <tablename>*welfordbank*_en.csv

- Comment carefully on the notebook where the data is generated. Ensure it has a presentable structure.

The files we generated have more than ≥ 25000 rows each.

TheFixed seed guarantees identical dates between runs; this has allowed us to divide the workload among our teams. We used SEED=15.

We have also set the enclosure “es_Es“.

At the hour run the notebook, the csv is automatically saved in the folder where the notebook is.

6. Limitations and Conclusions

In carrying out this work we encountered several limitations that influenced the planning and technical decisions:

- **CSV size in Git**
  The files`transactions.csv` and `loan_payments.csv`exceeded the**100 MB**, which is why GitHub rejected the *pushes* To solve this we add`*.csv` to the file.gitignore and we kept only the scripts and schematics under version control.

- **Inconsistencies in file names**
  Although the data was generated in the folder`GenerateDataEnglish`, some tables were still being exported with the suffix`_es.csv` (For example, `digital_engagement_es.csv`). We created a renaming script, manually reviewed each file, and now the generator exports only with the

correct suffix._en.csv.

- **Excessive memory usage in categorical coding**
  When applying`OneHotEncoder`about**25,000 customers and** more than 1.8 **million transactions**, the process was consuming more than 12 GB of RAM. We activated the option `handle_unknown='ignore` **and** we simplified the categories, reducing consumption to less than 8 GB.

- **Unrealistic branch allocation**
  We found that some small cities ended up with four branches, which is unlikely. We adjusted the algorithm so that the number of branches depends on the account volume and does not exceed four unless the account quota justifies it.

- **Deadline pressure**
  As the team responsible for data generation, we had to deliver first so that others could move forward. The tight schedule forced us to prioritize and limit the scope of certain improvements; we still managed to complete the work with a satisfactory result given the circumstances.

Together, these steps allowed us to overcome the most significant obstacles and deliver a consistent, reproducible dataset ready for integration by other teams.