Master's degree in Big Data, Technology and Advanced Analytics

Master's final project

INTEGRATION OF THE 'MASS PAYMENTS' BUSINESS VERTICAL INTO EBURY'S DATA WAREHOUSE

Author
Minerva Bermejo Arcos

Supervised by
Eduardo Lobo Fenouil
Edgli Morales

Madrid
June 2024

Minerva Bermejo Arcos declara, bajo su responsabilidad, que el Proyecto con título **Integration of the 'Mass Payments' business vertical into Ebury's data warehouse** presentado en la ETS de Ingeniería (ICAI) de la Universidad Pontificia Comillas en el curso académico 2023/24 es de su autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Fdo.: ..... Fecha: 21/06/2024

Autorizan la entrega:

Los Directores del Proyecto

Eduardo Lobo Fenouil

Fdo.: ..... Fecha: 03/06/2025

Edgli Morales

Fdo.: Edgli Morales Fecha: 03/06/2025

V. B. del Coordinador de Proyectos

Carlos Morrás Ruiz-Falcó

Fdo.: ..... Fecha: .....

Abstract

Within Ebury, the company in which this project is framed, the 'Ebury Mass Payments' (EMP) business vertical is experiencing a significant revenue growth. The main motivation behind this project is to support its growth and maximize its potential. Specifically, the objective has been to integrate EMP into a robust infrastructure that allows for more centralized and efficient data management.

The solution has been developed within the company's data warehouse using Google Cloud's BigQuery, a database optimized for handling large volumes of data with a serverless, scalable and high-performance architecture. Data transformation was performed using SQL code through dbt tool, which greatly facilitated the creation of data models, dependency control, and the execution of automated tests to ensure the quality and consistency of the transformed data.

As such, the designed models aim to serve as the foundation for enhancing Ebury's ability to generate valuable insights from EMP data. Based on these models, the data team will be able to create more sophisticated and domain-specific models, leading to better decision-making by stakeholders.

Acknowledgments

First and foremost, I would like to thank my parents, without whom none of this would have been possible. Thank you for your unconditional love, encouragement, and belief in me. Especially to my mother, my best guide, and the reason why I decided to pursue this master's degree.

I also want to extend my sincere thanks to my fellow students, whom I deeply admire and from whom I have learned so much. Your companionship and support have been invaluable as we faced and overcame challenges together. It has been an honor to grow alongside you all.

Finally, I am incredibly grateful to my co-workers at Ebury, who have always been willing to lend a helping hand and answer all my questions. Thank you for welcoming me so warmly into the team and teaching me so much, you have made this internship an unforgettable experience.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Business context

1.1.1 Ebury

Ebury, founded in 2009, is a fintech company specializing in international transactions, particulary in foreign exchange, for small and medium-sized enterprises (SMEs). Its value proposition lies in providing a single platform that allows the SMEs to make all their international payments and collections with significantly lower fees than traditional banks, enabling them to operate seamlessly in the global marketplace. [1]

Ebury's comprehensive suite of services also encompasses multi-currency accounts, payment cards, and additional services such as currency insights, payment gateways, and risk management tools. Nevertheless, Ebury's core profit remains in Foreign Exchange (FX), whose mechanism can be simplified as follows:

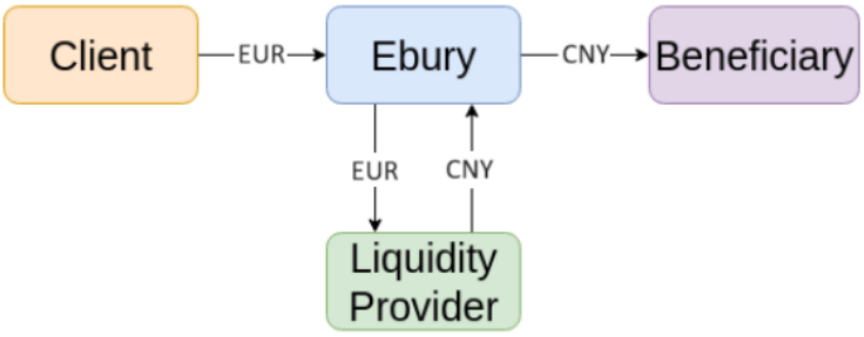

Figure 1.1: Ebury's core profit mechanism

For example, an European solar panel manufacturer (Client) needing to purchase materials from a Chinese supplier (Beneficiary). Client pays Ebury the required amount in EUR. Ebury then acts as an intermediary, converting these euros into CNY (Chinese yuans) through its network of Liquidity Providers. By securing a favorable exchange rate, Ebury obtains a profit margin on the conversion. Finally, Ebury disburses the CNY equivalent to Beneficiary.

1.1.2 Ebury Mass Payments (EMP)

Frontierpay is a company that was bought by Ebury in 2019. It has been renamed to Ebury Mass Payments (EMP), so both terms are used interchangeably along this document and within the company.

EMP offers a payroll service, which simplifies and streamlines the process of paying employees across borders, particularly for companies with staff in multiple countries. Through features like multi-currency payments, automated processing, and robust compliance, EMP empowers businesses to streamline payroll, ensure timely compensation, strengthen compliance, and confidently expand their global reach.

EMP offers FX Trades and Outgoing Payments, products already offered by Ebury Core (i.e. everything that is not Mass Payments within Ebury). However, although the concept is the same, they are treated as different products within the company, in order to avoid reporting problems. Moreover, EMP offers less amount of FX types than Ebury Core.

1.1.3 FX Trades

Foreign Exchange (FX) is a contract between two parties to deliver a set amount of currency on a particular date. An exchange is made as one currency is used to buy another one.

Types within EMP

- **Spot Trades:** These trades involve the immediate exchange of currencies at the prevailing market rate. Clients typically book spot trades when they require immediate currency conversion, often for payroll disbursements within a short timeframe (today, tomorrow, or the day after tomorrow).
- **Forward Trades:** Forward trades allow clients to lock in an exchange rate for a future date (calle *maturity date*), offering protection against currency fluctuations. EMP offers two types of forward trades:

  - **Fixed Forward (FF):** Clients agree upon a fixed exchange rate to be used on a predetermined future date. This option is ideal for situations where a company knows the exact amount of foreign currency needed at a specific future date for payroll purposes.
  - **Window Forward (WF):** Similar to a fixed forward, a window forward allows clients to secure a fixed exchange rate for a specific period. However, unlike fixed forwards, window forwards offer the flexibility to withdraw funds (drawdowns) throughout the defined window. Each drawdown affects the net profit of the trade. This option is well-suited for companies with uncertain disbursement schedules within a specific timeframe.

- **Drawdowns:** In the context of forward trades, a drawdown refers to the withdrawal of funds by the client before the maturity date. A single forward contract can accommodate multiple drawdowns. EMP also provides fields like *fwd_reference* and *deal_set* to facilitate tracking and managing these drawdowns effectively.

Sellbacks and Closeouts

Forward contracts lock in an exchange rate for a future date, offering protection against currency fluctuations. However, there might be situations where a company enters into a forward contract but no longer needs it before the maturity date. There are two main options for exiting a forward contract early:

- **Sellbacks:** If a client requires extending the settlement date of a forward contract, a new contract is created with the adjusted terms and extended maturity date. The original forward is then closed and 'sold back' to the counterparty (the liquidity provider) at the current market price. The difference between the original forward prince and the new price paid for the sellback reflects the cost of extending the contract.

- **Closeouts:** When a forward contract is no longer required, it can be completely closed out by settling at the current market price. This involves both parties exchanging the difference between the agreed-upon forward price and the prevailing market price:

  - If the market price has moved favorably for the client compared to the forward price (client receives more of the foreign currency), the counterparty will compensate the client for the difference.
  - If the market price has moved against the client (client receives less of the foreign currency), the client will compensate the counterparty for the difference.

The flowchart below (see Figure 1.2) provides a visual representation of the decision-making process for identifying sellbacks and closeouts. By following the arrows and considering the questions at each step, it can be determined the appropriate course of action for exiting an unwanted forward contract.

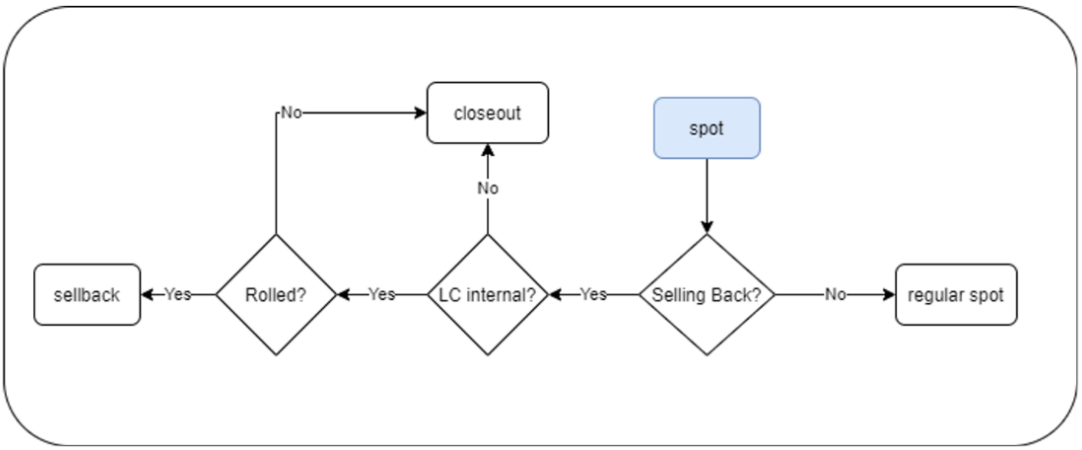

Figure 1.2: How to identify sellbacks and closeouts [2]

1.1.4 Outoing Payments

Through this product, Ebury's customers send payments to beneficiaries and Ebury charges them a fee per payment. This is the source of revenue from the payments made. Payment fees will be charged automatically from the Client's balance, unless it is zero, in that case the fees will be charged once the Client has added funds to their main currency account.

Liquidity Providers, in turn, charge Ebury for facilitating these payments, known as Transfer Costs.

Types

- **Independent:** Payments made without the need of booking a trade.
- **Dependent:** Payments made through the booking of a trade such as Spot or Drawdown (from the original Forward). A single FX trade can have multiple Dependent Outgoing Payments associated.

Handling Dependent Payments in FX Trades

The BOS system, which handles trade booking, currently presents a double-counting issue for FX trades. The Gross Profit and Net Profit values displayed for FX trades include the revenues and costs associated with the corresponding Outgoing Payments. However, these Outgoing Payments themselves already account for these revenues and costs, leading to an inflated representation of the trade's profitability.

To accurately reflect the true profitability of FX trades, it's crucial to adjust the profit calculation methodology. This involves subtracting the sum of all revenues and the sum of all costs from all payments associated with the trade. This ensures that the profits displayed represent the actual net gain from the FX transaction.

1.2 Motivation of the project

Ebury Mass Payments, since its inception in 2019, has been treated somewhat autonomously within the organisation. This separation has resulted in less integrated data management, with processes such as manual reporting in tools like Google Sheets, which is limiting EMP's ability to scale and optimise its operations.

The primary motivation for this project lies in the need to integrate EMP with Ebury's overall structure to maximise efficiency and value. The recent success of this vertical has highlighted the importance of having a robust data infrastructure in place to enable the analytics and decision science team to apply advanced models, generate informative dashboards and extract meaningful value from the data.

In summary, this project is motivated by the need to incorporate EMP transactions and payments into the data warehouse, given the growing importance of the EMP business in the Group's revenues and the requirement to track revenues consistently across all relevant business lines.

1.3 Description of the technologies

Within the technological context of my project, several advanced tools and platforms are used to extract, store, process and analyze all of Ebury's business data. The following diagram shows the data flow architecture assembled in the company and at which point of the pipeline each technology used is framed:

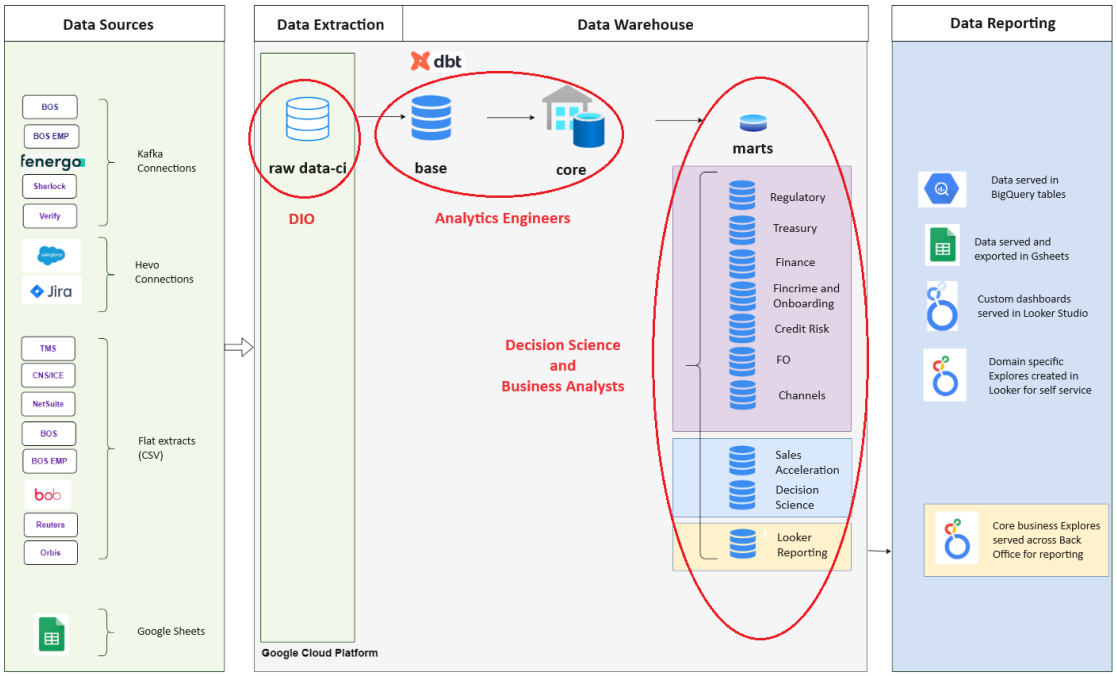

Figure 1.3: Ebury's data flow architecture [2]

Data Extraction

The Data Infrastructure and Operations (DIO) team uses robust tools such as Kafka and Hevo for data extraction from various sources such as Salesforce, Jira, among others. These tools enable efficient ingestion of raw data into the data warehouse. [2]

Data Warehouse and Transformation

The data is stored in BigQuery, a cloud database fully managed by Google Cloud Platform (GCP). BigQuery is known for its speed and scalability, allowing the analysis of large volumes of data quickly and efficiently.[2]

For data transformation and the creation of analytical models, dbt (data build tool) is used. dbt is an open source SQL-based tool that facilitates the creation of dimensional and factual tables, as well as the application of complex business logic. This ensures consistency and quality of data at all stages of the analytical process.

As can be seen in Figure 1.3, the Ebury warehouse is structured in 4 layers:

- **Raw-data layer:** This is the crudest layer, where the raw data extracted by DIO lands.
- **Base layer:** In this layer, the Analytics Engineers are in charge of processing the data to store it in a consistent way in scd2 historical tables (scd2 was chosen over scd4 because it is easier to read). This historical data may include, for example, transaction history or customer data history.
- **Core layer:** This layer also belongs to the Analytics Engineers, and is where they create the tables that contain only the current information, i.e. the latest version of each transaction or client account (following the previous examples), so that each primary key (e.g. *transaction_id, client_id*) only appears once in its respective table. In this layer, the future application of the data to the business is already taken into account, in order to create an efficient table structure based on it.
- **Marts layer:** This last layer is built by Data Analysts together with the Decision Science team, who create specific tables for use cases defined by each business vertical, such as Front Office, Credit Risk or Regulatory.

It is worth mentioning that the data flow is not exactly as described in Figure 1.3, as the marts layer can also draw directly from historical tables of the base layer, if required. Therefore, the relationship between layers would be more correctly represented as follows:

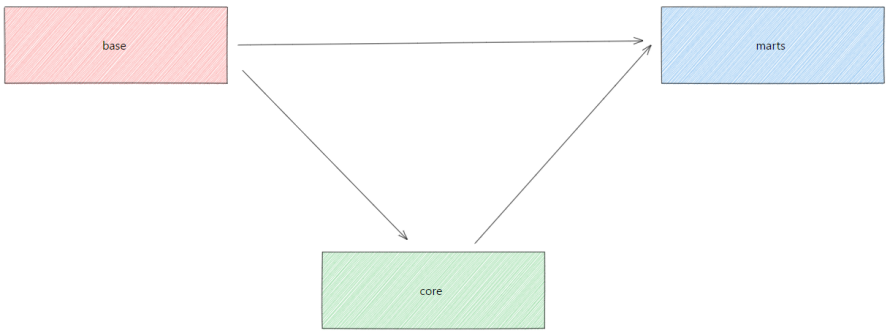

Figure 1.4: Layer communication [2]

Data Reporting

BigQuery integrates seamlessly with Looker Studio, Google's data visualization platform. Looker Studio enables the creation of interactive dashboards and advanced data visualizations, facilitating interpretation and data-driven decision making across the organization.

Chapter 2

State of the art

In recent decades have witnessed an exponential surge in the volume of data generated from a variety of sources: mobile phones, social networks and countless Internet of Things (IoT) devices. Traditional methods have proven inadequate to manage this huge influx of data, which has led to the emergence of the term 'Big Data'. In response, cloud computing has emerged as a fundamental solution, offering models capable of dealing with Big Data storage, particularly through database-as-a-service (DBaaS).

DBaaS is a cloud service model that enables organizations to use database services without the burden of managing underlying hardware, database software, and maintenance operations. DBaaS providers handle all administrative tasks required to run a database, including configuration, monitoring, backup, and recovery.

Key features that make DBaaS attractive for Big Data applications are:

- **Virtualization:** Physical resources are virtualized, allowing multiple users to share access to a common pool of virtualized resources, optimizing infrastructure costs and maintenance.
- **Ubiquity:** Cloud services are always accessible via network, whether through the Internet or a private local network.
- **Pay-as-you-go:** Users pay only for the resources consumed, eliminating the need for initial infrastructure investment. Resources scale automatically to match workload demands. [3]

In this context, BigQuery and dbt (data build tool), used in this project, stand out. This chapter will explore their features and advantages, as well as the best practices and recommended techniques for the optimization of a Data Warehousing (DW) system aimed at Big Data applications.

2.1 BigQuery

BigQuery is the DBaaS solution provided by Google Cloud. As a DBaaS, its main advantage is its serverless architecture, decoupling computation and storage, which allows automatic scaling, adjusting resources based on worload demands. This approach offers immense flexibility and cost controls for customers, as they do not need to keep their expensive computing resources running continuously. [4]

BigQuery leverages the query engine **Dremel**, designed by Google for executing SQL queries at high speed across petabytes of data. To achieve this, Dremel translates SQL queries into execution trees. Leaf nodes, known as 'slots', handle heavy lifting by reading data from storage and performing necessary computations. Branches of the tree, known as 'mixers', perform aggregation. Slots are dynamically allocated to queries as needed, ensuring fairness for concurrent queries from multiple users. So, a single user can be allocated thousands of slots if needed.

Storage, on the other hand, utilizes **Colossus**, Google's global storage system. Data is stored in a columnar format, enhancing query performance by minimizing the need to scan entire tables, as only the columns needed to answer queries are accessed. Colossus also manages replication, recovery in case of disk failures, and distributed administration to avoid single points of failure. [5]

2.2 dbt

dbt (data build tool) is a data transformation tool that facilitates the construction, management, and documentation of data infrastructure using SQL code. Its main advantages are detailed below:

- **SQL-Based transformation:** dbt enables data transformation using SQL, a very simple and easy-to-learn query language. This enables less technical professionals, who may have more business expertise, to create their own data models for specific analyses.
- **Dependency management:** dbt simplifies the management of dependencies between different data models. This ensures transformations are executed in the correct order and that all models are up-to-date and consistent.
- **Automated testing:** dbt includes a framework for running automated tests to verify the integrity and quality of transformed data. These tests are essential for ensuring transformations yield consistent and expected results, preventing errors in subsequent analyses.

- **Documentation and Collaboration:** dbt provides a collaborative development environment where teams can jointly create and improve data warehousing projects. This is achieved through version control and a robust documentation system that updates automatically, allowing collaborators to maintain complete control over the workflow. The documentation includes detailed information about data models, such as their names and locations, descriptions of transformation logic, ownership, employed data sources, and required tests.

- **Integration with BigQuery:** dbt integrates seamlessly with BigQuery, enabling the direct execution of SQL models within the BigQuery environment. This integration simplifies the creation and maintenance of complex data models, leveraging BigQuery's power and scalability. Additionally, by using dbt to transform data, partitioning and clustering can be directly applied in BigQuery, which optimizes data storage and access, leading to more efficient processing and reduced costs. [4]

2.3 Optimization techniques

2.3.1 Partitioning

Partitioning is a fundamental technique to optimize the performance of a data warehouse. It consists of dividing a table into multiple segments, called partitions, which are managed independently. As each partition can be read selectively, the number of data scanned in each query is reduced, resulting in shorter processing times and significant cost savings.

Moreover, partitioning can also benefit from cheaper long-term storage by creating partitions. If a partition is not changed for 90 consecutive days, the price of storage is reduced by 50%. Furthermore, in dbt, partition expiration can be set at the table or dataset level, so that if a partition is not used within a specified time interval, its data is deleted.

BigQuery supports three types of partitioning:

1. **Time-unit column:** By any column whose data type is DATE, DATETIME or TIMESTAMP. This is often used in marts models against which business users can run queries).

2. **Integer range:** By using a range of values on an INTEGER column. This is less common, as it can be tricky to configure the correct number of buckets.

3. **Ingestion time:** By the ingestion time of the source data. BQ automatically assigns rows to partitions based on the time they arrive, and the granularity can be specified: hourly, daily, monthly or yearly.

It is worth warning that, to avoid partition skew and the resulting performance overhead, it is very important not to partition by columns at risk of creating unbalanced partition sizes as the data grows. [5]

2.3.2 Clustering

Clustering is another technique that can be used to enhance query performance. Unlike partitioning, which divides data into physically distinct locations, clustering just sorts the information so that similar values of specified columns are stored together. This is particularly beneficial when:

- The table is larger than 1GB (actually, in smaller tables the benefits of clustering are not really seen).
- Certain columns are frequently filtered or aggregated.
- It is required more granularity than partitioning allows. While partitioning is limited to one column, clustering can be applied to up to four additional columns, providing more detailed data organization [5]

2.3.3 Use of materialized views

A materialized view periodically caches the result of a query, automatically updating itself when the underlying data changes, so that the results are always up-to-date. Using a materialized view eliminates the need to recalculate the results each time the query is run, which is especially convenient for recurring or computationally expensive queries.

Additionally, for small datasets, materialized views can significantly improve performance because the access and retrieval time from cache is much less compared to the cost of executing the entire query, allowing for almost instant responses. [6]

2.3.4 JOIN optimization

JOINS are very used in SQL queries, but they can be resource-intensive if not used in an efficient way. Efficient joins can be achieved by:

- Ensuring join keys are unique and distinct.

- Filtering data early in the query to minimize the amount of data that each join operation has to process.
- Placing the CTE[1] reading the largest table first, and the CTE reading the smallest table next. When BigQuery joins two tables, it employs a broadcast join strategy, where the smaller table is distributed to all slots to be combined with the larger table. By placing its CTE first, the volume of data initially distributed and processed is minimized.
- Utilizing partitioned or clustered tables where possible. Joins on such tables can be significantly faster since BigQuery can leverage these structures to process only relevant partitions or clusters.

2.3.5 SQL query optimization

SQL query optimization is essential to improve DW performance and efficiency. The goal is to write queries that minimize cost and execution time. Some recommended practices include:

- Avoiding using 'SELECT *' and instead select only the necessary columns. As BQ operates as a columnar database, this has a big impact on performance, as only specified columns in the query are processed.
- Utilizing filtering clauses such as 'WHERE' and 'LIMIT' to reduce the number of processed rows.
- Avoiding nested subqueries by replacing them with common table expressions (CTE). This not only improves the readability of the code, but also optimizes the overall performance of the model by allowing the reuse of complex query results, since CTEs are treated as temporary views.
  - Example of optimized query, using CTEs:

```
WITH filtered_data AS (
  SELECT column1, column2
  FROM dataset.source_table
  WHERE condition
)
SELECT column1, SUM(column2)
FROM filtered_data
GROUP BY column1
LIMIT 100;
```

[1] A Common Table Expression (CTE) is a named query that is defined once and can be used as a temporary table within the same main query or in subsequent queries. [7]

– Example of non-optimized query, using subqueries:

```
SELECT column1, SUM(column2)
FROM (
    SELECT *
    FROM dataset.source_table
    WHERE condition
)
GROUP BY column1;
```

2.3.6 Denormalization

Denormalization involves combining data from multiple tables into a single table to reduce the need for costly joins at query time. A common way to denormalize a database is to join a fact table with one or more of its dimensions[2]. However, querying a large denormalized table can be slower than joining two smaller tables that together provide the same data as the large one, so denormalization is not always recommended.

If denormalization is chosen, it is advisable to take advantage of BigQuery's native support for nested and repeated structures by using a combination of ARRAY and STRUCT data types to create the table schema. A series of tables with fictitious data are shown below as an example to understand this:

order_key	cust_key	total_price	order_date
1	73100	26602	2023-01-19
2	92861	17680	2023-02-03

Table 2.1: Dimensional table storing Order information [5]

order_key	line_number	quantity	extended_price	ship_date
1	1	3	4081	2023-01-29
1	2	18	22521	2023-01-23
2	1	41	7010	2023-02-05
2	2	27	3288	2023-02-05
2	3	42	7382	2023-02-09

Table 2.2: Fact table storing Line Item information [5]

[2]Fact tables store events or transactions, which are dynamic data, while Dimension tables contain descriptive and qualitative attributes, which are more static.

order_key	cust_key	total_price	order_date	quantity	price	ship_date
1	73100	26602	2023-01-19	3	4081	2023-01-29
1	73100	26602	2023-01-19	18	22521	2023-01-23
2	92861	17680	2023-02-03	41	7010	2023-02-05
2	92861	17680	2023-02-03	27	3288	2023-02-05
2	92861	17680	2023-02-03	42	7382	2023-02-09

Table 2.3: Flatten Order-Items using Joins [5]

order_key	cust_key	total_price	order_date	quantity	price	ship_date
1	73100	26602	2023-01-19	3	4081	2023-01-29
1				18	22521	2023-01-23
2	92861	17680	2023-02-03	41	7010	2023-02-05
2				27	3288	2023-02-05
2				42	7382	2023-02-09

Table 2.4: Flatten Order-Items using Nested and Repeated Fields [5]

As shown in the example, using nested and repeated fields helps avoid data duplication, preserves the normalized structure of the original data, and enhances performance. However, this structure is only meaningful if the downstream models or applications can use it.

2.4 Conclusion

The combination of BigQuery and dbt provides a robust and efficient solution for developing DW systems. BigQuery delivers scalable, high-performance infrastructure, while dbt facilitates data transformation and management using SQL.

Furthermore, the employment of techniques such as partitioning or the use of materialized views is key to achieve the highest possible performance in DW systems. These tools and best practices in combination, form a solid foundation for creating a modern and efficient data warehouse optimized for large-scale data processing and analysis.

Chapter 3

Definition of the work

3.1 Objectives

As explained in Section 1.2, the main motivation behind this project is the need to incorporate EMP into the data warehouse, as there is a need to have visibility of all the Group's revenues and how they are reflected in our financial statements on a day-to-day basis.

The most suitable long-term solution to this cannot yet be done, as it requires prior work by the DIO team to create new source tables from Kafka[1], as the current available sources are still based on legacy ETL processes. Kafka is the tool used to create the sources in Ebury Core and is preferred for its ability to process data with much higher frequency, scalability and low latency.

However, before performing this ETL to Kafka migration, it is extremely important to save the entire BOS EMP history, and that is what this project aims to do, to establish a comprehensive data structure for BOS EMP data coming via ETL within the warehouse, specifically the Base and Core layers.

For this purpose, the following objectives have been set for each layer:

- **Base layer:** As mentioned above, BOS EMP will eventually be phased out, so it is extremely important to save all the history, which for the moment can only be obtained through ETLs. Thus, the first target is storing the entire history of BOS EMP in two scd2 tables, one for FX Trades and another for Outgoing Payments.

[1] Apache Kafka is an open-source distributed streaming platform that enables fast and durable publishing, consumption and processing of large volumes of data. [8]

- **Core layer:** From the base tables, build two fact tables, one for FX Trades and another for Outgoing Payments, which will hold only the latest version of each transaction based on the unique identifiers: transaction_receipt or payment_id. There Core tables will serve as the bases for creating new domain-specific models within their respective Marts folder, thus facilitating in-depth analysis of EMP data.

3.2 Methodology

1. **Business understanding**

   (a) Thoroughly grasped the concept of EMP, its purpose and the role it plays within Ebury's operations.

   (b) Delved into the specifics of EMP's product offerings: FX Trades and Outgoing Payments, to fully understand their particularities and data requirements.

   (c) Collaborated closely with business stakeholders and analysts to gather their expectations, pain points, and data needs related to EMP.

2. **Tool and method familiarization**

   (a) Gained proficiency in the data modeling and transformation tools utilized in the company, including DBT, Big Query and Looker Studio.

   (b) Conducted ad-hoc data analysis tasks related to Ebury Core to gain hands-on experience with the business and the existing data landscape.

   (c) Learned the validation process that all models must go through before going into production. This mainly consists of the creation of a PR[2] on GitHub, that requires the approval of several analysts in order to merge the changes from the own branch into the *staging* (pre-production) branch.

3. **Source identification and evaluation**

   (a) Identified and documented all potential data sources for EMP information, including internal systems (e.g. BOS EMP, Kafka) and external sources (e.g. Salesforce, BOB).

[2]PR is the short for 'Pull Request', a mechanism used in software development to request the review and integration of changes made in a particular branch of a code repository into the main branch. It functions as a communication channel between developers collaborating in the same project.

(b) Critically assessed each data source in terms of its reliability, always prioritising those with the best data quality.

4. Data modeling and validation

(a) Designed two scd2 tables in the base layer to store the historical data coming from BOS through ETL, and enriched with status information received via Kafka, to enhance their value for analysis. Different transformation techniques were applied in order to ensure data consistency and adherence to business rules.

(b) Designed two fact tables in the core layer, including only the latest version of each transaction from the base layer tables, and excluding the information that can be found in a more reliable way in another tables that are already in production, such as *dimaccount* or *dimemployee*.

(c) Created YAML files to specify source definitions (only for newly used data sources in the warehouse), dbt test to be applied to each data model, as well as detailed field definitions for the final tables.

(d) Validated the final tables against the business records to identify and rectify any discrepancies or inconsistencies, specially in critical values, such as the number of distinct Transaction Receipts or Payment Ids, or the total sum of GBP Volume and GBP Gross Profit.

5. Deployment to production

(a) Developed the PR where all the validation process is exposed in order to be able to merge the own branch into *staging*.

(b) Deployed the new models or tables to the production environment, making the EMP data accessible to analysts for reporting and analysis.

6. Post-deployment support and fixes

(a) Provided ongoing support to analysts in utilizing the new models and addressing any issues or concerns that arose.

(b) Implemented post-deployment fixes, such as the addition of a field that had not been included in the first instance, but is actually really useful for analysts; or the change of logic when calculating a specific revenue type for FX trades.

3.3 Economic impact estimation

3.3.1 Breakdown of estimated annual benefits

Time saving and cost reduction

Currently, about 20 staff members in the operations team spend about 20 minutes a day manually monitoring customer accounts. Monitoring involves downloading reports with transactional information from BOS, analyzing them in Excel, verifying the status of the account associated with each transaction and, in case of anomalies, investigating the SWIFT MT950[3]. This manual process consumes time and resources, limiting efficiency and scalability.

The developed modeling integrates transactions and customers together in the warehouse, so that all information is available and reconciled in a single source, thus avoiding the need for manual downloads from different platforms. This will enable much faster real-time analysis. It has been estimated a saving of 140 hours per month per team, which, considering a working day of 40 hours per week, would be equivalent to approximately 1 FTE (Full Time Equivalent):

$$\frac{140\frac{h}{month}}{40\frac{h}{week} \times 4\frac{week}{month}} = 0.875\ FTE \approx 1\ FTE\ released$$

Considering an average annual salary of 25.000 - 30.000 EUR per FTE, it is estimated that the project will generate annual savings of approximately 27.000 EUR in salary costs.

$$27.000\ EUR \times 1\ \ FTE = 27.000\ EUR\ saved$$

Performance monitoring and better decision making

On the other hand, manual data reconciliation generates a lag in EMP performance information. The lack of access to real-time data limits the ability to identify deviations from sest targets and, therefore, to take preventive or corrective measures.

The integration of data in the warehouse will allow constant monitoring of different KPIs (Key Performance Indicators) in real time, which will enable EMP's

[3] A SWIFT MT950 is a financial statement message sent by an account servicing institution to an account owner. It is used to transmit detailed information about all entries booked to the account, whether caused by a SWIFT message or not. [9]

commercial team to take action faster and thus have better control over their numbers. The information they used to know at the end of the month (lag indicator) will now be available to them on a daily basis, allowing them to respond to market fluctuations as quickly as possible: launching marketing campaigns in specific geographies, promotions to particular types of customers, etc.

For the current fiscal year 2024/25, this business vertical has a target of 40 million euros in revenue. Considering a potential impact of 2%, the revenue increase due to the optimization of commercial strategies would be around 800.000 EUR.

$$40.000.000\ EUR \times 0.02 = 800.000\ EUR\ extra$$

Identification of cross-selling opportunities

Furthermore, the project is expected to have a strong positive impact on the identification of cross-selling opportunities. Cross-selling is a strategy of marketing and sales that consists of offering customers additional products or services that complement their original purchase. The fact of having all the information related to client accounts and their operations in the warehouse facilitates the subsequent detection of behavioral patterns and the segmentation of customers for personalized campaigns.

Last year, a specific model was designed for creating a dashboard in Looker Studio that served to identify cross-selling opportunities in Ebury Core. With this dashboard, it was achieved to increase incomes by 2.7% in that vertical, which resulted in 4.3 million extra profit for the company. If the same success rate is attained for EMP, currently valued at 40 million euros, it would then generate an additional 1 million euros in revenue.

$$40.000.000\ EUR \times 0.025 = 1.000.000\ EUR\ extra$$

3.3.2 Total estimated annual benefit

Based on the above, it is concluded that the integration and modeling of EMP data in Ebury's warehouse will generate significant impact for the company, including:

- Reduced costs due to the elimination of FTEs: 27.000 EUR/year
- Increased revenues due to better control of KPIs: 800.000 EUR/year
- Increased revenues due to the identification of cross-selling opportunities: 1.000.000 EUR/year

→ Total estimated profit: **1.827.000 EUR/year**

As a whole, it is estimated that the project will generate an annual economic benefit of 1.827.000 EUR, which represents a highly profitable investment for the company. Especially, taking into account that the project has been developed by an employee working part-time (20h/week) for 3 months. Knowing that her gross salary was 600 EUR/month, the total investment amounts to only 1.800 EUR.

$$600\frac{EUR}{month} \times 3\ months = \textbf{1.800 EUR invested}$$

Chapter 4

Developed solution

4.1 Description of the final solution

Figure 4.1 represents the pipeline that makes up the final solution, in which, in order to reach the final tables (represented in blue), several intermediate models have been created. These models serve as stages to progressively integrate the different sources and apply different transformations. This modular approach is recommended by the official dbt documentation [7], as it offers several significant advantages:

- First, it improves the readability of the SQL code, allowing for better data management and debugging, as well as the execution of more specific tests on particular parts of the model. This results in greater data integrity and reliability throughout the entire process.
- Another reason is performance optimization. BigQuery, being a data warehousing platform based on MPP (Massively Parallel Processing), benefits from the pre-computation of intermediate steps, as it avoids the repetition of costly calculations by reusing intermediate results. This reduces the complexity of the final queries, thereby optimizing both execution time and resource usage.

Besides, it is important to note that the models that build the final tables are actually intermediate models, but named with the prefix 'build_' instead of 'int_'. As such, the final models are simply a SELECT * from their corresponding 'build_' model, along with any relevant dbt settings, as may be the partitioning configuration (see Section 4.3). This way, if any dbt test fails, it will fail on the 'build_' models and not on the final tables, allowing issues to be identified and resolved before reaching the final tables.

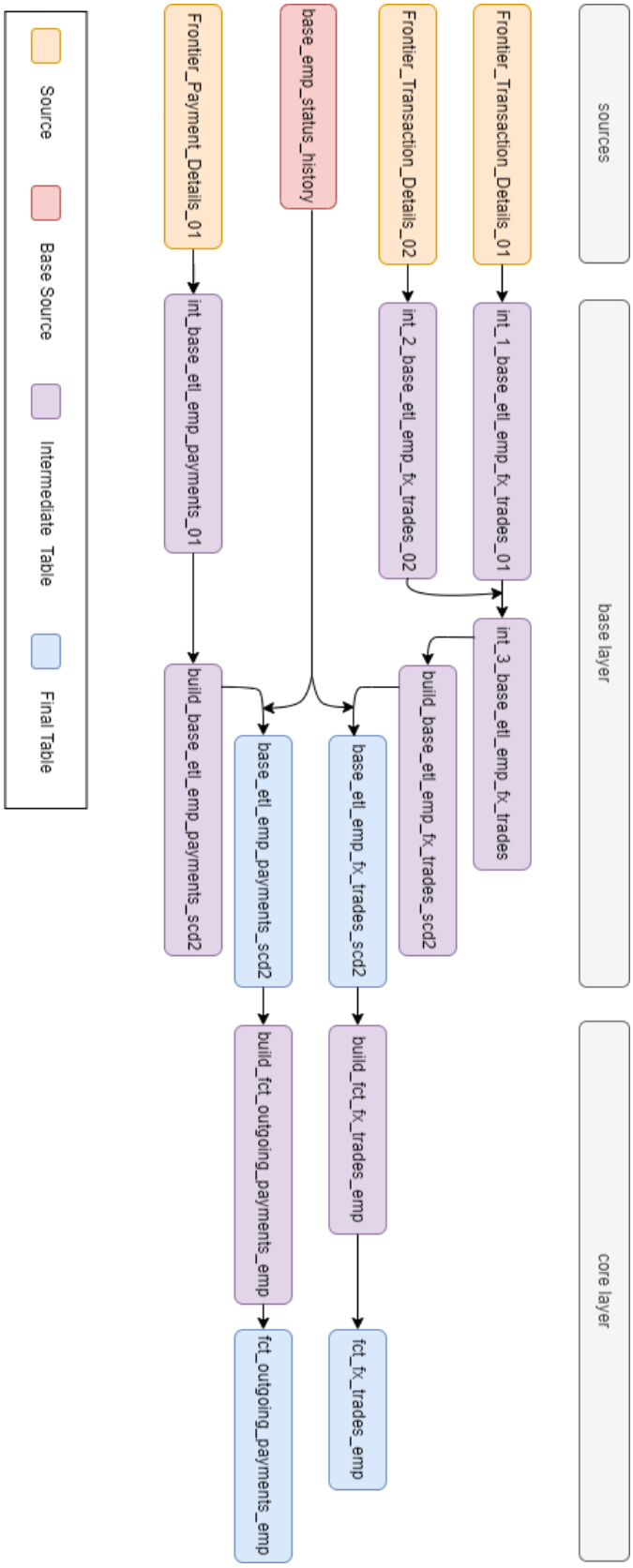

Figure 4.1: Diagram representing the final solution

4.1.1 Sources

As outlined in Section 1.3, this project falls under the purview of the AE team, utilizing data sources provided by the DIO team. This team is responsible for landing the data into the warehouse in a structured format, with each row representing a new record and each column representing a field of interest.

Despite this reliance on DIO, the AE team maintains the responsibility for assessing data quality and provenance in the sources employed. It is also very important to know how often they are received, i.e. their data refresh frequency, in order to determine the appropriate modeling approach (refer to Section **??**).

The data sources used for constructing the new EMP models are summarized in the following table:

Source name	**Extracted from**	**Received at**
Frontierpay_Transaction_Details_01	BOS MP ETL	Every day at 18 UK time.
Frontierpay_Transaction_Details_02	BOS MP ETL	Every day at 18 UK time.
base_emp_status_history	Kafka	Every day at every hour.
Frontierpay_Payment_Details_01	BOS MP ETL	Every day at 18 UK time.
Frontierpay_Payment_Details_02	BOS MP ETL	Every day at 18 UK time.

Table 4.1: Sources employed

4.1.2 Assets in Base layer

In the Base layer, two historical SCD2 tables have been created, one for each product. In this way, when the time comes to migrate to Kafka, the entire history of EMP transactions will be available in a clean and easily debuggable form.

SCD2, or Slowly Changing Dimension Type 2, is a method of tracking historical data changes by creating a new record for each change. This approach was chosen over others because it provides a complete and clear view of the evolution of the data over time.

Another method that was strongly considered was SCD4, which involves maintaining a separate historical table, thereby achieving high storage efficiency as a result of reducing the load on the main table. However, this option was discarded because it was considered that the management of two tables for a single entity could complicate the queries and the maintenance of the history.

Model	Schema	Description
int_1_base_etl_emp_fx_trades_01	intermediate_base	Transformation and renaming of fields in Frontierpay_Transaction_Details_01.
int_2_base_etl_emp_fx_trades_02	intermediate_base	Transformation and renaming of fields in Frontierpay_Transaction_Details_02.
int_3_base_etl_emp_fx_trades	intermediate_base	Join of int_1_base_etl_emp_fx_trades_01 and int_2_base_etl_emp_fx_trades_02.
build_base_etl_emp_fx_trades_scd2	intermediate_base	int_3_base_etl_emp_fx_trades enriched with the status of transactions coming from Kafka.
base_etl_emp_fx_trades_scd2	base_bos_etl	Final SCD2 table storing BOS EMP FX trades history coming from ETL.
int_base_etl_emp_payments_01	intermediate_base	Transformation and renaming of fields in Frontierpay_Payment_Details_01.
build_base_etl_emp_payments_scd2	intermediate_base	int_base_etl_emp_payments_01 enriched with the status of payments coming from Frontierpay_Payment_Status_01.
base_etl_emp_payments_scd2	base_bos_etl	Final SCD2 table storing BOS EMP Outgoing Payments history coming from ETL.

Table 4.2: Assets built in Base layer

The table above shows all the models created up to the two final tables of the Base layer, highlighted in grey. For each model, it also specifies the schema or dataset in which they are grouped, and a brief description of what they do.

Regarding the modelling developed, it is particularly noteworthy is the creation of the fields *date_from* and *date_to* in order to manage the control of the different versions of the same transaction. The *date_from* field indicates the date on which the version is read from the BOS EMP, and is calculated by simply equating it to the *extraction_date* field. The *date_to* field marks the end of the version, which is the date of extraction of the next version or, in its absence, a distant date never to be reached: 2222-12-31.

The following table illustrates a fictitious payment transaction that is intended to serve as a simple example to understand these fields:

payment_id	extraction_date	date_from	date_to	payment_status	is_last_version
PI12345	2023-10-04	2023-10-04	2023-10-07	Pending	False
PI12345	2023-10-07	2023-10-07	2023-10-18	Approved	False
PI12345	2023-10-18	2023-10-18	2222-12-31	Completed	True

Table 4.3: Example to illustrate the version change in a SCD2 table

In order to be able to quickly identify the most current information, the Boolean field *is_last_version* has been created, which indicates whether the record corresponds to the latest version of the transaction. It has been obtained by grouping by transaction ID and identifying the record with the most recent date in the *record_created_date* field, among the records with the highest value of *date_from*, among the records with the highest value of *last_modified_date*.

4.1.3 Assets in Core layer

For EMP, due to the much smaller amount of data compared to Ebury Core, it has been thought to simply build two fact tables in the Core layer, one for each product. Each fact table reads from its corresponding SCD2, collecting only the records whose *is_last_version* is True, so making the transaction IDs fields unique.

The fields that provide specific information about the account which booked each transaction have also been excluded. The reason is that their values are more static, as they change more eventually, so it is not appropriate to store them in a fact table but in a dimensional instead. Actually, this table already exists and is called 'dimaccount'. Fact tables, on the other hand, are intended to store dynamic information, in this particular case related to transactional movements.

It is likely that in the future it will be decided to create an obt (One Big Table) for EMP, which brings together transactional and customer, in order to avoid the need for recurring joins in downstream models. In that case, the fact tables created would be enriched with customer data from 'dimaccount'.

Model	**Schema**	**Description**
build_fct_fx_trades_emp	intermediate_core_dimensional	Last version of each transaction, excluding the specific data of the account that made the transaction (leaving the basics: account_number, etc.).
fct_fx_trades_emp	core_dimensional	Final fact table storing the current EMP FX trades.
build_fct_payments_emp	intermediate_core_dimensional	Last version of each payment, excluding the specific data of the account that made the payment (leaving the most basic).
fct_outgoing_payments_emp	core_dimensional	Final fact table storing the current EMP Outgoing Payments.

Table 4.4: Assets built in Core layer

In terms of the transformations made, it is worth mentioning the adjustment of revenues in the FX fact table. As explained in Subsection 1.1.4, the revenue that appears in BOS EMP for FX trades includes the revenues of all their associated Dependent Payments. Since these revenues are already accounted for in the Payments fact table, it is necessary to discount them.

To do this, the Payments fact table (created beforehand) is read, payments are grouped by *fwd_reference*, and the group's total revenue is calculated. Then, by mapping each FX trade's *transaction_receipt* to the groups' *fwd_reference*, the group's revenue is subtracted from the corresponding FX trade's revenue.

In addition, new fields have been included in the FX fact table. Named with the format 'original_[field]' (e.g. *original_transaction_type*, *original_order_date*), they just take the field value from the FX trade whose *transaction_receipt* matches *selling_back*. If this field is null, another new field named *sub_transaction_type* will take the same value as *transaction_type*; otherwise, it will be 'Sellback' if the liquidity provider is internal, or 'Closeout' if it is not (see Subsection 1.1.3).

4.1.4 YAML files

In this project, four YAML files have been created, one for each set of models and sources used directly in the construction of the four final tables:

YAML File	**Models and Sources**
base_etl_emp_fx_trades_scd2.yml	Frontierpay_Transaction_Details_01, Frontierpay_Transaction_Details_02, base_emp_status_history, int_1_base_etl_emp_fx_trades_01, int_2_base_etl_emp_fx_trades_02, int_3_base_etl_emp_fx_trade, build_base_etl_emp_fx_trades_scd2, base_etl_emp_fx_trades_scd2
base_etl_emp_payments_scd2.yml	Frontierpay_Payment_Details_01, Frontierpay_Payment_Details_02, int_base_etl_emp_payments_01, build_base_etl_emp_payments_scd2, base_etl_emp_payments_scd2
fct_fx_trades_emp.yml	build_fct_fx_trades_emp, fct_fx_trades_emp
fct_outgoing_payments_emp.yml	build_fct_payments_emp, fct_outgoing_payments_emp

Table 4.5: List of models and sources specified in each YAML file

Figure 4.5 shows the sources and models specified in each YAML. Note that, despite the fact that the source base_emp_status_history is also used to build both SCD2s, it only appears in base_etl_emp_fx_trades_scd2.yml. This is because in dbt, both sources and models can only be defined once, to avoid redundancies and possible conflicts in documentation and version control. Yet, they can be referenced many times in multiple models.

For each source used, the name, owner and criticality are specified. For each model developed, the name, a brief description of its purpose and the tests it has to pass to ensure data integrity (see Subsection 5.1.3 for more information). In the case of the final models, in addition, the name and description of all the resulting fields are also detailed.

Besides this, in order to integrate the newly created models into the dbt workflow, some entries had to be added to the 'models' section of the **dbt_project.yml** file. This is the central configuration file of any dbt project, as it is crucial to provide dbt with information about the project structure, the location of the data models and how to interact with the data warehouse (BigQuery).

The added entries specify, for each directory created or used to locate the developed models, the schemas on which the intermediate models, on the one hand, and the final models, on the other hand, must be materialized (these are those specified in Figures 4.2 and 4.4). Also additional metadata, such as the criticality of each model (see Section 5.2).

4.2 Problems encountered

Low quality of status coming from ETL

The field indicating the status of transactions provided by the ETL process was of low quality, as it often contained missing values and inconsistencies in formatting. This made it difficult to reliably determine the actual status of transactions, which could lead to downstream processing errors and incorrect final analyses.

To address this problem, it was decided to enrich the ETL data with the status field coming from Kafka. Although this tool is already the main way of creating sources in Ebury Core, it is still in the process of being implemented for EMP. However, there did exist *base_emp_status_history*, a source built using Kafka that fetched the continuous stream of status updates directly from BOS EMP, which ensured the information from this field was more accurate and up to date.

Thus, it was decided to replace ETL status with Kafka status in the final tables. Although it is generally not good practice to mix data sources, data quality should always be prioritized, and in this particular case, the timeliness and accuracy of the Kafka status data outweighed the potential drawbacks.

It is important to mention that, in order to join both source types, the table 'base_status_history_scd2', had to be filtered so that the last accepted update is before 20 o'clock of the previous day. This is because ETL sources are updated every day at 18 o'clock UK time, while Kafka sources are updated every hour.

Account number mismatch with Salesforce

The account numbers stored in BOS EMP did not directly match the unique identifiers used within the table 'dimaccount'. These IDs in 'dimaccount' are the ones considered to be correct, as they are the ones coming from Salesforce[1], the platform where all customer-related information is stored.

This mismatch has been resolved using the 'clients_client' table, which was built specifically as a quick fix for this problem. This table acts as a bridge between the two systems, mapping the BOS EMP account numbers to the corresponding Salesforce account IDs. Then, finding these IDs in 'dimaccount', the correct account numbers and names are extracted. Therefore, two left join are required:

```
SELECT
    da.account_number,
    da.account_name,
    dt.account_number as bos_account_number,
    dt.account_name as bos_account_name
FROM 'ebi-dev-260310.Frontierpay.Frontierpay_Transaction_Details_01' AS dt
LEFT JOIN 'ebi-dev-260310.EMP_Kafka_Hermes.clients_client' AS clients
    ON dt.Account_Number = SAFE_CAST(clients.ebury_account_id AS STRING)
LEFT JOIN 'ebi-dev-260310.core_dimensional.dimaccount' AS da
    ON clients.client_crm_id = da.Account_ID
```

Also note that the original *account_number* and *account_name* from BOS EMP have been preserved in the final tables under the names *bos_account_number* and *bos_account_name* for potential future reference or reconciliation processes.

[1]Salesforce is a cloud-base customer relationship management (CRM) platform that provides tools to help companies manage and analyze customer interactions and data.[10]

4.3 Partitioning the SCD2 tables

As discussed in the subsection 2.3.1, partitioning is a crucial technique for optimising query performance in BigQuery and most large-scale databases. But for it to be effective, it is important to choose wisely which fields a table should be partitioned by, based on what queries will be run most frequently against it.

Given the fact that the final fact tables in the Core layer will have a very general use and it is still soon to predict trends, it has been decided not to partition them. On the contrary, it is known that the main use of the final historical tables in the Base layer will be to feed the fact tables. This means that every day, a query will be automatically launched to read the scd2 tables, filtering only the records relating to the latest version of the transaction and applying a series of transformations to them.

So, it has been decided to partition the two final scd2 tables of FX and Payments by the *is_last_version* column, aiming to store separately the records in which this field is True. In this way, these records can be retrieved much more quickly and, since they have to be retrieved on a daily basis, this will result in a significant improvement in the overall performance of the system.

However, since it is not possible to partition by Boolean columns in BigQuery (see Section 2.1), there is no way to partition directly by *is_last_version*. In order to do so, an auxiliary column called *is_last_version_int* has been created, which is 1 when *is_last_version* is True and 0 otherwise. In this way, the dbt model responsible for creating 'base_etl_emp_fx_trades_scd2' is configured as follows:

```
{{ config(
    materialized='table',
    partition_by={
        "field": "is_last_version_int",
        "data_type": "int64",
        "range": {
            "start": 0,
            "end": 1,
            "interval": 1
            }
        }
    ) }}
select * from {{ ref('build_base_etl_emp_fx_trades_scd2') }}
```

For 'base_etl_empayments_scd2', the code would be exactly the same, but calling 'build_base_etl_empayments_scd2' instead. This dbt configuration instructs to:

- Materialize the table as a physical table in the data warehouse.
- Partition the table by *is_last_version_int* column.
- Define the partition range as 0 to 1 with an interval of 1. This essentially creates two partitions: one for records where *is_last_version_int* equals 1, and another for those where it equals 0.

Chapter 5

Data Governance

5.1 Data Quality

5.1.1 Data Cleaning and Auditing

All fields coming from the sources have been audited one by one, to check whether they added value or not in the new models. A thorough data quality assessment led to the removal of certain fields due to inconsistencies, redundancies, or irrelevance to EMP context. These reasons can be categorized as follows:

1. **Erroneous Data:** Fields with inaccurate or illogical values were removed to ensure data integrity. For instance, *years_first_trade* was removed due to inconsistencies with *first_trade* values.

2. **Redundant Information:** Fields that provided redundant or overlapping information with other fields were eliminated to streamline the data structure. For example, *opening_window* was removed as it contained the same information as *fwd_open*.

3. **Irrelevance to EMP Context:** Fields that were not applicable to MP transactions and were consistently null were removed to reduce data clutter and focus on relevant information. This included fields like *base_ccy*, *fixing_source*, and *spot_liquidity_provider*.

4. **Unnecessary Fields:** Fields that did not provide meaningful or actionable information were removed to improve data efficiency. For instance, the *client_type* field was deemed unnecessary for the analysis.

For a more detailed justification of the reasons for the removal of the above-mentioned fields and some others, please refer to the Appendix A.

Once the data cleaning wass done, the remaining fields and the new fields calculated in the final models have been documented in a Google Sheet called 'Data Dictionary'. This consists of 4 sheets, one for each final table, in which the names of the fields are listed, together with their type (String, Integer, Boolean, etc.) and their business description. This sheet has been reviewed by the treasury and finance stakeholders, who have ensured its alignment with business needs.

5.1.2 Data Consistency

To ensure consistency and facilitate analysis, all 'flag' fields have been converted to the Boolean data type. This eliminates potential ambiguities arising from BOS, where it can be found a disparity of options: Yes/No, Y/N, 1/0, True/False.

On the other hand, all field names have been standardized to lowercase and separated by underscores for multi-word fields. In this way, fields coming from BOS as 'Account_Number' and 'PI' have been renamed 'account_number' and 'payment_id'. This approach aims to facilitate the use of the tables in future models that draw from it, and to improve the overall readability of the data structure.

5.1.3 Data Integrity and Reliability

To ensure data integrity and reliability, a series of dbt tests have been defined throughout the different stages of the transformation process. These tests leverage SQL queries to validate data against specific criteria. When a test fails, it issues an alert, triggering immediate investigation and corrective action.

Test	**Description**	**Layer**	**Field**	**Action to take**
Freshnesh	All sources are updated.	Sources	date_from	Raise an indicent to DIO.
Mutually exclusive ranges	No overlap in date ranges within a partition.	Base	date_from, date_to	Verify and potentially adjust the logic in dbt to ensure date ranges are mutually exclusive for each transaction ID. Raise an indicent to DIO if necessary.
Unique combination of columns	The combination of specified pairs of columns is unique	Base	transaction_receipt with date_from, payment_id with date_from)	Verify that the combinations of these fields is unique. Raise an incident to AE if necessary.
Unique field	Unique transactions and payments	Base, Core	transaction_receipt, payment_id	Raise severity.

Table 5.1: Tests to ensure Data Quality

The tests applied are described in the table above, which also specifies in which layer the test is applied, to which specific fields, and the action to take if the test triggers and alert.

5.1.4 Data Validation

The last step to ensure data quality is validation, which is done once the models have been created, and before they go into production. In fact, the validation process is so important that, at Ebury, all PRs (see Section 3.2) must have it documented in detail in order to be approved.

The validation process for new models usually consists of comparing certain values with those of existing models. These values can be, for example, the number of unique Transaction Receipt or Payment Id; or the sum total of measures such as Volume, Gross Profit, Payment Amount or Fee Amount.

Given that for EMP there is hardly anything set up in the warehouse, the validation process could not be very comprehensive, as there was not much to compare. There were, though, two tables that had been built as a temporary solution to have EMP information available from BOS ETL: trades_mass_payments for FX trades, and stg__mp_payments_last for Outgoing Payments.

It is worth mentioning that these tables only collected the latest version of each transaction, so it has not been possible to compare all the history collected in the new tables in the base layer. However, it is assumed that if all the latest versions match, the history is correct.

For a more detailed description of the validation process that was carried out for the tables covered by this project, please refer to Appendix B.

5.2 Data Classification

According to their risk governance, the final models are classified as follows:

Model	Owner	Criticality	Sensibility
base_etl_emp_fx_trades_scd2	Analytics Engineers	Medium	Low
base_etl_emp_payments_scd2	Analytics Engineers	Medium	Low
fct_fx_trades_emp	Analytics Engineers	Medium	Low
fct_outgoing_payments_emp	Analytics Engineers	Medium	Low

Table 5.2: Classification of the final models

- **Owner:** The team responsible for owning any issues related to the model.
- **Criticality**: High, medium, or low, measuring the impact on business operations in the event of a model failure.
- **Sensibility:** High, medium, or low, indicating the level of risk associated with potential exposure or loss of the data used in the models.

5.3 Dependencies

In any data governance initiative, understanding dependencies—both upstream and downstream—is crucial for ensuring correct data integration and reliability. It helps in anticipating issues, ensuring data quality and the overall success of data-driven projects. The current dependencies for this project are the following:

- **Upstream Dependencies:** The only upstream dependency lies with the DIO team, as they are responsible for providing the raw data through their ETL and Kafka tables. This ensures that our data pipelines are fed with accurate and timely data from reliable sources.
- **Downstream Dependencies:** Currently, there are no downstream dependencies, as these are all new tables and no other tables draw from them. However, this is a temporary state. As these tables start to be used to create new models in core and marts layers, dependencies will inevitably arise.

As the project progresses, continuous management of dependencies will be required, in order to ensure smooth integration of new models and to mitigate any unforeseen issues that may arise.

Chapter 6

Conclusions and next steps

This project has been successful in exploiting the benefits of Ebury's existing data infrastructure to support the company's growth and adaptability. The integration of the Ebury Mass Payments (EMP) business into the company's data warehouse will enable improved data-driven decision-making in this increasingly relevant vertical. Moreover, the adoption of advanced technologies such as BigQuery and dbt will represent a significant advance in terms of both operational efficiency and the capacity to manage and analyse large volumes of data.

Looking forward, it will be crucial to mantain the new models, by making necessary fixes to ensure their accuracy and continuous operation. Additionally, detailed documentation will be created to assist colleagues who are using the models for the first time, ensuring effective knowledge transfer and proper usage.

Another essential step will be the incorporation of new data sources coming from Kafka, once the DIO team has built them. It will also be high priority to explore and apply new techniques for optimising queries and data warehousing. For example, a pending task is to test different clustering ways, by different fields, and compare the results obtained in terms of processing and storage costs in the long term.

Finally, more specific models will be developed within the marts layer. The most urgent model is the one that serves to build a replica of the 'Revenue Performance Dashboard' of Ebury Core. It is very important because it shows the commissions of each dealer and salesperson based on the revenue they generate for the company. Originally, building this dashboard was also a goal of this project, but it was postponed due to the need for preliminary work by the Salesforce team, which has to integrate EMP-specific logic into the data warehouse to assign revenue percentages based on the employee's role within the sales team.

Appendix A

Deleted fields

- It has been decided to eliminate the field **length** because it does not meet its definition: Length in days of tenor (Maturity Date - Order Date). This is shown in the following query:

```
SELECT
    order_date,
    maturity_date,
    DATE_DIFF(maturity_date, order_date, DAY) AS length_calc,
    length
FROM 'ebi-dev-260310.Frontierpay.Frontierpay_Transaction_Details_01'
LIMIT 100
```

Row	order_date	maturity_date	length_calc	length
1	2021-09-15T00:00:00	2021-09-15T00:00:00	0	6
2	2021-09-15T00:00:00	2021-09-16T00:00:00	1	6
3	2021-09-16T00:00:00	2021-09-17T00:00:00	1	7
4	2021-09-16T00:00:00	2021-09-17T00:00:00	1	6
5	2021-09-16T00:00:00	2021-09-17T00:00:00	1	6
6	2021-09-14T00:00:00	2021-09-14T00:00:00	0	10
7	2023-04-21T00:00:00	2023-04-28T00:00:00	7	7
8	2023-04-21T00:00:00	2023-04-28T00:00:00	7	7
9	2021-09-09T00:00:00	2021-09-09T00:00:00	0	7
10	2023-04-19T00:00:00	2023-04-27T00:00:00	8	8

Figure A.1: Justification for the removal of the field *length*.

- The field **years_first_trade** has been deleted because it does not make sense for it to be greater than 0 in transactions where first_trade = 'Yes':

```
SELECT
    transaction_receipt,
    first_trade,
    years_first_trade
FROM ‘ebi-dev-260310.Frontierpay.Frontierpay_Transaction_Details_01’
WHERE first_trade = ‘Yes’
LIMIT 100
```

Row	transaction_receipt	first_trade	years_first_trade
1	FXFOTR274877	Yes	13
2	FXFOTR314779	Yes	14
3	FXFOTR314385	Yes	14
4	FXFOTR313727	Yes	14
5	FXFOTR314908	Yes	14
6	FXFOTR313602	Yes	14
7	FXFOTR299977	Yes	14
8	FXFOTR300399	Yes	14
9	FXFOTR299077	Yes	14
10	FXFOTR298150	Yes	14

Figure A.2: Justification for the removal of the field *years_first_trade*.

- The field **first_month** is supposed to be a binary variable indicating whether it is the client's first month of activity, but it does not match. Thus, it has been deleted.

```
SELECT
    da.Account_Number,
    dt.transaction_receipt,
    dt.order_date,
    dt.first_trade,
    da.FX_Became_Client,
    dt.first_month
FROM ‘ebi-dev-260310.Frontierpay.Frontierpay_Transaction_Details_01’ AS dt
LEFT JOIN ‘ebi-dev-260310.EMP_Kafka_Hermes.clients_client’ AS clients
    ON dt.Account_Number = SAFE_CAST(clients.ebury_account_id AS STRING)
LEFT JOIN ‘ebi-dev-260310.core_dimensional.dimaccount’ AS da
    ON clients.client_crm_id = da.Account_ID
WHERE First_Month = ‘No’ AND First_Trade = ‘Yes’
    AND order_date > ‘2020-01-01T00:00:00’
    AND FORMAT_DATE(‘%Y-%m’ dt.order_date)
        = FORMAT_DATE(‘%Y-%m’, da.FX_Became_Client)
```

Row	Account_Number	transaction_receipt	order_date	first_trade	FX_Became_Client	first_month
1	517674	FXFBTR108282	2020-01-23T00:00:00	Yes	2020-01-07T00:00:00	No
2	517674	FXFBTR108282	2020-01-23T00:00:00	Yes	2020-01-07T00:00:00	No
3	518255	FXFOTR108454	2020-01-24T00:00:00	Yes	2020-01-13T00:00:00	No
4	518255	FXFOTR108454	2020-01-24T00:00:00	Yes	2020-01-13T00:00:00	No
5	518406	SFPOTR111209	2020-02-24T00:00:00	Yes	2020-02-05T00:00:00	No
6	518406	SFPOTR111209	2020-02-24T00:00:00	Yes	2020-02-05T00:00:00	No
7	518124	FXFOTR111537	2020-02-25T00:00:00	Yes	2020-02-05T00:00:00	No
8	518124	FXFOTR111537	2020-02-25T00:00:00	Yes	2020-02-05T00:00:00	No
9	517853	FXFBTR112744	2020-03-11T00:00:00	Yes	2020-03-10T00:00:00	No
10	517853	FXFBTR112744	2020-03-11T00:00:00	Yes	2020-03-10T00:00:00	No

Figure A.3: Justification for the removal of the field *first_month*.

- The field **opening_window** is a double check for fwd_open, both mean the same. Thus, it has been used to fill the fwd_open values in case they are null, and has been deleted as a separate field.
- Others:
  - The fields **base_ccy**, **fixing_source**, **fixing_date**, **fixing_rate**, **settlement_amount**, **settlement_cost**, have been deleted because they only apply to NDF contracts. Thus, in EMP they are always null.
  - The fields **spot_liquidity_provider** and **deliverable_rate** have been deleted because they are always null.
  - The field **client_type** has been deleted because is not needed.

Appendix B

Validations

FX trades

- The values of Volume and Gross Profit are exactly the same in the new table and in trades_mass_payments, the table that is already in production.

```
WITH new_scd2 AS (
    SELECT transaction_receipt, gbp_volume, gbp_gross_profit
    FROM 'ebi-dev-260310.dbt_marcos_base_bos_etl.base_etl_emp_fx_trades_scd2'
    WHERE is_last_version
)
prod AS (
    SELECT
        transaction_receipt,
        gbp_volume AS gbp_volume_prod,
        gbp_gross_profit AS gbp_gross_profit_prod
    FROM 'ebi-dev-260310.intermediate_core_dimensional.trades_mass_payments'
)
SELECT *
FROM new_scd2
LEFT JOIN prod USING (transaction_receipt)
WHERE gbp_volume != gbp_volume_prod
    OR gbp_gross_profit != gbp_gross_profit_prod
```

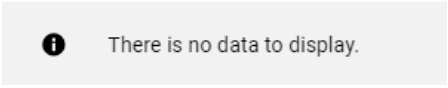

Figure B.1: Result of the first query used to validate FX trades.

- The number of unique transactions in the new table matches the number of transactions in table trades_mass_payments.

```
SELECT COUNT(transaction_receipt)
FROM ‘ebi-dev-260310.dbt_marcos_base_bos_etl.base_etl_emp_fx_trades_scd2’
WHERE is_last_version
UNION ALL
SELECT COUNT(transaction_receipt)
FROM ‘ebi-dev-260310.intermediate_core_dimensional.trades_mass_payments’
```

Row	f0_
1	369789
2	369789

Figure B.2: Result of the first query used to validate FX trades.

Outgoing Payments

- The values of Payment Amount and Fee Amount are exactly the same in the new table and in stg__map_payments_last, the table that is already in production.

```
WITH new_scd2 AS (
    SELECT payment_id, gbp_payment_amount, gbp_fee_amount
    FROM ‘ebi-dev-260310.dbt_marcos_base_bos_etl.base_etl_emp_payments_scd2’
    WHERE is_last_version
)
prod AS (
    SELECT
        PI AS payment_id
        GBP_PAYMENT_AMOUNT AS gbp_payment_amount_prod,
        GBP_FEE_EQUIVALENT AS gbp_fee_amount_prod
    FROM ‘ebi-dev-260310.base_mp_payments.stg__mp_payments_last’
)
SELECT *
FROM new_scd2
LEFT JOIN prod USING (payment_id)
WHERE gbp_payment_amount != gbp_payment_amount_prod
    OR gbp_fee_amount != gbp_fee_amount_prod
```

There is no data to display.

Figure B.3: Result of the first query used to validate Outgoing Payments.

- The number of unique payments in the new table matches the number of payments in table stg__map_payments_last.

```
SELECT COUNT(payment_id)
FROM ‘ebi-dev-260310.dbt_marcos_base_bos_etl.base_etl_emp_payments_scd2’
WHERE is_last_version
UNION ALL
SELECT COUNT(PI)
FROM ‘ebi-dev-260310.base_mp_payments.stg__mp_payments_last’
```

Row	f0_
1	1788302
2	1788302

Figure B.4: Result of the second query used to validate Outgoing Payments.

Bibliography

[1] Ebury. *Ebury's official website*. 2024. URL: https://www.ebury.es.

[2] Ebury. *Internal documentation owned by the company*. Private information created in Notion for Ebury employees usage. 2024.

[3] Manar Abourezq and Abdellah Idrissi. "Database-as-a-Service for Big Data: An Overview". In: *International Journal of Advanced Computer Science and Applications (IJACSA)* 7.1 (2016). URL: https://pdfs.semanticscholar.org/a0e7/e17762d0297f9a4cb7c714aa1ee86962c512.pdf.

[4] Md. Husen Ali, Md. Sarwar Hosain, and Md. Anwar Hossain. "Big Data Analysis using BigQuery on Cloud Computing Platform". In: *Australian Journal of Engineering and Innovation Technology (AJEIT)* 3.1 (2021), p. 9. URL: https://universepg.com/public/storage/journal-pdf/Big%20data%20analysis%20using%20bigquery%20on%20cloud%20computing%20platform.pdf.

[5] Maruti C. Rabi Abbasi. *Build scalable and trustworthy data pipelines with dbt and BigQuery*. 2023. URL: https://services.google.com/fh/files/misc/dbt_bigquery_whitepaper.pdf.

[6] Mohammed J. Zaki. "Data Warehouse Performance: Selected Techniques and Data Structures". In: *Business Intelligence*. Springer, 2012. Chap. 2, pp. 27–62. URL: https://www.cs.put.poznan.pl/rwrembel/DW-Perf-LNBIP.pdf.

[7] dbt. *Official dbt documentation*. 2024. URL: https://docs.getdbt.com/.

[8] Apache Kafka. *Kafka's official website*. 2024. URL: https://kafka.apache.org/.

[9] Hatton National Bank. *SWIFT MT940/950*. URL: https://ebanking.hnb.lk/corp/L001/helpfiles/Help_Files/Retail%20Balance%20and%20Transaction%20Information/Del/SWIFTMT940950.htm#:~:text=Business%20Condition%3A%20MT950,Finacle%20Condition.

[10] Salesforce. *Salesforce's official website*. 2024. URL: https://www.salesforce.com/eu/.