Building a Scalable 3-Tier Data Lakehouse for Mobility Analysis in Spain

Big Data Engineering Project

Joan Sánchez Verdú **María López Hernández**
Fernando Blanco Membrives

January 5, 2026

Abstract

Abstract. This paper presents the design and implementation of a scalable 3-tier Data Lakehouse architecture tailored for the analysis of large-scale mobility data in Spain. Leveraging open data from the Spanish Ministry of Transport (MITMA) and the National Statistics Institute (INE), the system integrates high-volume Origin-Destination (OD) matrices with demographic and economic indicators. The infrastructure utilizes DuckDB for efficient analytical processing and DuckLake for ACID-compliant storage, orchestrated via Apache Airflow. We demonstrate the system's utility through two key use cases: characterizing typical daily mobility patterns and identifying transport infrastructure gaps using gravity models. The result is a robust, cost-effective platform enabling transport experts to derive data-driven insights for urban planning.

1 Introduction

Mobility analysis is critical for modern urban planning...

1.1 Objectives

The primary objective of this work was to construct a robust data infrastructure...

- Build a scalable Lakehouse for mobility and economic data.
- Implement ELT processes across Bronze, Silver, and Gold tiers.
- Demonstrate analytical value through specific mobility use cases.

2 Data Sources

The data lakehouse architecture is designed to ingest and integrate public domain data exclusively, ensuring reproducibility and open access. The following subsections detail the specific datasets and sources selected to build the information system.

2.1 MITMA Open Data

The core mobility dataset is sourced from the Spanish Ministry of Transport, Mobility and Urban Agenda (MITMA [1]), specifically the *Estudios Básicos de Movilidad*. This dataset provides high-resolution insights into the daily movements of residents derived from mobile network big data. From the MITMA the following files have been used:

- ***(202301-202312)_ Viajes_ municipios.tar*:** These files, located in the MITMA web at *estudios_ basicos/por-municipios/viajes/meses-completos/*, contain the information of the mobility from an origin to a destination municipality, per day, per hour, for the year 2023.
- ***nombres_ municipios.csv*:** This file, located in the MITMA web at *zonificacion/zonificacion_ municipios/*, contains the name of each municipality.
- ***relacion_ ine_ zonificacionMitma.csv*:** This file, located in the MITMA web at *zonificacion/*, cointains the mapping between the MITMA codes and the INE codes for the municipalities.
- ***poblacion.csv*:** This file, located in the MITMA web at *zonificacion/*, cointains the population of each municipality per year.
- ***zonificacion_ municipios.shp*:** This ESRI Shapefile (along with its *.dbf*, *.shx*, and *.prj* components), located in the MITMA web at *zonificacion/zonificacion_ municipios/*, contains the official geospatial geometries (polygons) for the municipal study zones.

2.2 INE Demographics and Economics

To contextualize mobility flows and enable the gravity model analysis, mobility data is enriched with official statistics from the Spanish National Statistics Institute (INE [2]). The statistics used from INE are the following:

- ***ine_ rent_ municipalities.csv*:** This file, located in the INE web at *https://www.ine.es/jaxiT3/files/t/es/csv_bd/*, contains the rent information per year and municipality.

2.3 Open Data

To categorize mobility patterns by day type (e.g., distinguishing standard workdays from holidays), specific calendar data is integrated from Open Data portals.

- ***calendario_ laboral.csv*:** This file, sourced from the Madrid City Council Open Data Portal (*Datos Abiertos Madrid* [4]) at *https://datos.madrid.es/*, contains the official working calendar. It provides the classification of dates (working days, weekends, and holidays) from 2013 to 2026 (included).

3 Lakehouse Architecture

The system follows the Medallion Architecture pattern, decoupled into storage and compute layers to ensure scalability.

3.1 Technology Stack

The platform is built upon a modular, open-source stack designed to decouple compute from storage, allowing for independent scaling and cost efficiency. The architecture leverages the following core components:

- **Compute Engine (DuckDB [5]):** Selected for its high-performance, vectorized execution engine optimized for OLAP workloads. DuckDB serves as the primary processing unit, running in-process for orchestration tasks while being deployed via containerized environments for heavy transformations. This approach significantly reduces infrastructure overhead while maintaining the ability to process datasets larger than RAM through out-of-core execution.

- **Elastic Compute (AWS Batch [6]):** To handle high-volume data and computationally intensive joins, the platform offloads heavy SQL workloads to *AWS Batch*. This provides an ephemeral, cloud-native compute layer that automatically scales memory-optimized instances (e.g., EC2 R4 and R5 family) on demand. By executing DuckDB in the same cloud region as the data storage, the system minimizes network latency and maximizes throughput, while utilizing *Spot Instances* to achieve up to a 90% reduction in compute costs.
- **Lakehouse Management (DuckLake [7]):** A specialized extension that brings Data Lakehouse capabilities to the ecosystem. It provides ACID transaction support (Atomicity, Consistency, Isolation, Durability) and snapshot isolation. This ensures that long-running ingestion tasks do not block analytical queries and prevents data corruption during partial write failures.
- **Storage - Cloud Infrastructure (Neon + S3):** The system adopts a cloud-native architecture that physically separates data from metadata:
  - **Data Plane (AWS S3):** Stores the actual physical files (Parquet and CSV). S3 provides durability and scalable object storage for the Bronze, Silver, and Gold layers.
  - **Control Plane (Neon Postgres):** A serverless PostgreSQL instance acts as the centralized catalog and metadata store. It manages table definitions, schemas, and transaction locking, enabling safe concurrent writes to S3 from multiple Airflow workers.
- **Orchestration (Apache Airflow [8]):** Manages the end-to-end lifecycle of data pipelines. Airflow allows for the definition of Directed Acyclic Graphs (DAGs) to handle complex task dependencies, backfilling strategies for historical data, and granular retries in case of network or source failures.

3.2 Data Layering Strategy

The data pipeline is structured following the standard "Medallion" architecture pattern (Multi-hop), which organizes data quality into progressive stages. This design ensures that raw data is preserved for auditability while downstream layers provide cleaned and enriched datasets optimized for analytical consumption. The architecture consists of three distinct zones, each with a specific purpose and schema design.

3.2.1 Bronze Layer (Raw)

The Bronze layer acts as the landing zone (Staging Area) for all external data. Its primary function is to capture data from source systems in its native format with minimal modification, ensuring an immutable record of the original input.

In this implementation (Figure 1), a distinct table is instantiated for each source entity to maintain data isolation. To ensure pipeline robustness and prevent ingestion failures due to type mismatches, tabular data columns (such as CSV files) are initially cast as `VARCHAR`. A notable exception is the geographic data: by utilizing DuckDB's `spatial` extension, the ESRI Shapefile components are ingested via the `ST_Read` function. This allows the system to capture municipal boundaries directly as native `GEOMETRY` objects, preserving spatial precision from the source.

Additionally, the ingestion process is configured to ignore malformed records, ensuring that isolated errors do not halt the entire pipeline. To enhance data governance and auditability, metadata columns are appended to every record, including `ingestion_timestamp` and `source_url`. Finally, to optimize storage and performance, the high-volume mobility data is physically partitioned by date (`fecha`), enabling efficient batch processing.

Figure 1: Entity Diagram of the Bronze Layer. One table for each file.

3.2.2 Silver Layer (Cleaned & Integrated)

The Silver layer represents the refined, enterprise-wide view of the data. In this stage, data undergoes validation, cleaning, standardization, and enrichment processes to transform raw inputs into structured, trustworthy tables. The schema design (Figure 2) follows a Star Schema approach, establishing a central fact table for mobility flows linked to surrounding dimension tables for spatial and socio-economic context. All tables include a `TIMESTAMP WITH TIME ZONE` column `processed_at` for auditability.

Dimension Modeling and Spatial Integration (dim_zones)

The foundation of the Silver layer is the creation of a unified spatial dimension, `dim_zones`. This table integrates the descriptive zoning metadata (MITMA codes, INE codes, and municipality names) with the geospatial vector data ingested in the Bronze layer.

- **Spatial Integration:** The `dim_zones` table leverages the native `GEOMETRY` objects already parsed in the Bronze layer. By joining the municipal geometries with the MITMA-INE crosswalk (`mapping_ine_mitma`), the Silver layer establishes a clean, spatially-enabled dimension.
- **Surrogate Keys:** A unique integer identifier (`zone_id`) is generated for each municipality. This decouples downstream analytics from changes in external string codes (MITMA/INE IDs) and improves join performance.

Spatial Connectivity (Distance Matrix)

To support spatial network analysis and distance-based filtering, a derivative dimension table, `dim_zone_distances`, is established. This table functions as a pre-computed distance matrix representing the relationships between municipality pairs. By calculating the distance between the centroids of the geometries stored in `dim_zones`, the system avoids computationally expensive geospatial operations during query time.

- **Zone Linkage:** The table maps relationships using `origin_zone_id` and `destination_zone_id`, which act as foreign keys referencing the central `dim_zones` table.
- **Metric Storage:** The `dist_km` column stores the distance in kilometers as a `DOUBLE` precision value, allowing for precise downstream analysis of trip lengths and gravity models.

Socio-Economic Metrics and Filtering

Complementary metric tables are generated by cleaning raw CSVs. For the `metric_population`, data is cast to `BIGINT` while removing metadata headers.

For the `metric_ine_rent` table, specific business logic is applied to the INE source file to resolve granularity mismatches. The pipeline filters the dataset to isolate the "Average Net Income per Person" indicator and explicitly excludes district-level or census-section entries (filtering out rows where `Distritos` or `Secciones` are populated). This ensures that the resulting economic metrics align perfectly with the municipal granularity of the `dim_zones` table.

Temporal Context (Holidays)

To support temporal analysis, a `dim_zone_holidays` table is constructed. This process involves filtering the raw working calendar for events classified as "National Holidays" and cross-referencing them with the zones. This dimension allows the Gold layer to accurately distinguish mobility patterns between standard working days and holidays.

Mobility Fact Table Construction

The core mobility data is transformed into the `fact_mobility` table. This process converts

the raw, text-based daily CSVs into a strongly typed, time-series optimized format. Key transformations include:

1. **Temporal Standardization:** The raw separate date and hour fields are combined into a single `TIMESTAMP WITH TIME ZONE` column, explicitly localized to 'Europe/Madrid'.
2. **Spatial Lookup:** Origin and Destination codes are replaced by their corresponding `zone_id` through double joins against the `dim_zones` table, ensuring referential integrity.
3. **Type Casting:** The "trips" column is cleaned (handling European decimal formats) and cast to `DOUBLE` precision to support fractional trip expansion factors.

Data Observability and Quality Logging

To ensure the reliability of the pipeline, a dedicated table named `data_quality_logs` has been implemented within the Silver layer. Unlike the business data tables, this table serves as a metadata registry to persist the results of automated audit checks performed during ingestion (e.g., null rate calculations, row count validations, or statistical anomalies). This allows for longitudinal tracking of data health.

The schema is designed to be generic enough to store heterogeneous metrics:

- `check_timestamp`: The exact time when the audit was executed.
- `table_name`: The target table being audited (e.g., *fact_mobility*).
- `metric_name`: A descriptive identifier for the KPI (e.g., *avg_income_per_capita*, *null_zone_rate*).
- `metric_value`: A numeric field (`DOUBLE`) storing the result of the check.
- `notes`: Textual field for context, such as the specific batch date range or error warnings associated with the metric.

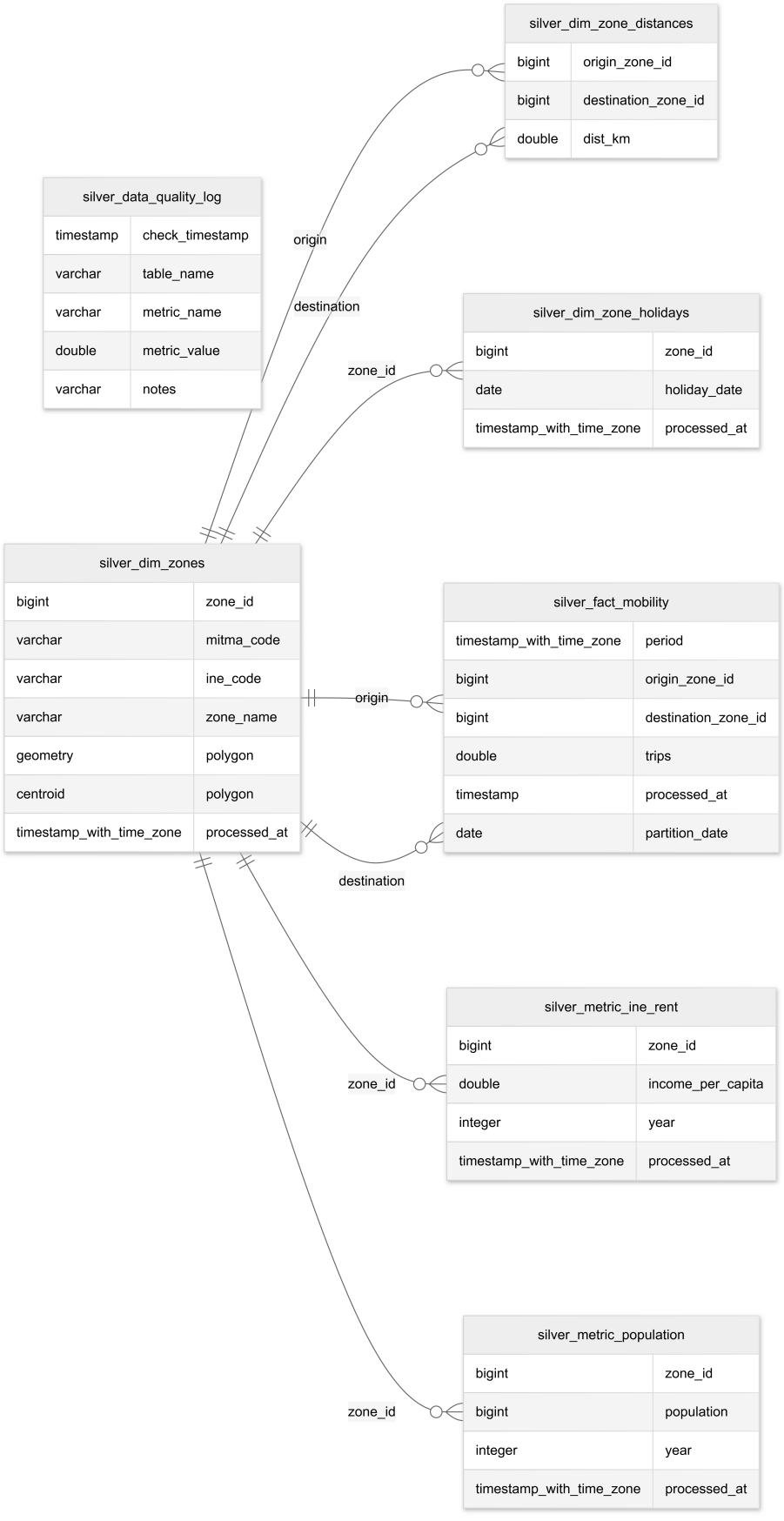

Figure 2: Entity-Relationship Diagram (ERD) of the Silver Layer. The schema follows a Star Schema design, centering on the `fact_mobility` table linked to spatial (`dim_zones`) and temporal dimensions.

3.2.3 Gold Layer (Mart)

The Gold layer constitutes the "Data Mart" of the lakehouse architecture. While the Silver layer focuses on data integrity and normalization, the Gold layer is organized around specific business problems. Tables in this layer are denormalized, pre-aggregated, and enriched with derived metrics to ensure low-latency performance for Business Intelligence (BI) tools and final reporting.

As illustrated in Figure 3, the schema consists of three distinct analytical tables, each corresponding to a specific use case:

- `gold_typical_day_patterns`: This table supports temporal analysis (Use Case 1). It reduces billions of raw trip records into a lightweight set of hourly profiles.
  - **Aggregation:** Data is grouped by `cluster_id` (representing patterns like "Weekday" or "Holiday") and `hour`.
  - **Metrics:** It stores the `avg_trips` to allow for immediate plotting of demand curves without recalculating the underlying raw data.
- `gold_infrastructure_gaps`: Designed for the Gravity Model analysis (Use Case 2), this table stores Origin-Destination (OD) pairs. Unlike standard OD matrices, this table is enriched with socio-economic context.
  - **Contextual Data:** Includes `total_population` (Origin) and `rent` (Destination proxy for economic attraction) alongside the spatial impedance `dist_km`.
  - **Key KPI:** The `mismatch_ratio` is the critical derivative, quantifying the gap between theoretical potential and actual observed trips.
- `gold_zone_functional_classification`: This table provides a semantic layer for the spatial zones (Use Case 3). Instead of raw ID numbers, it assigns functional roles to municipalities.
  - **Flow Ratios:** It persists calculated indices such as `net_flow_ratio` (balance of trade in trips) and `retention_rate` (local self-sufficiency).
  - **Categorization:** The `functional_label` column stores the final human-readable classification (e.g., "Bedroom Community"), simplifying downstream map visualizations.

Figure 3: Schema design of the Gold Layer. These tables represent the final analytical products, denormalized and enriched with calculated metrics (e.g., mismatch ratios, functional labels) ready for visualization.

4 Implementation Methodology

The operational logic of the Data Lakehouse is implemented through a modular, automated workflow orchestrated by Apache Airflow [8]. The system adopts a decoupled architecture where distinct Directed Acyclic Graphs (DAGs) handle specific stages of the data lifecycle based on their update frequency and functional scope.

This section details the engineering strategies employed to ensure scalable data ingestion, atomic processing of daily batches, and the automated generation of analytical models. The implementation prioritizes robustness through strict transaction management, task isolation, and the separation of heavy computational ELT from on-demand reporting queries.

4.1 Orchestration Strategy

To manage the dependencies between the Bronze, Silver, and Gold layers efficiently, the system is architected into a multi-DAG ecosystem. This design allows for independent scaling and maintenance of static dimensions versus high-volume mobility facts.

1. **Infrastructure & Dimensions DAG:** Handles the ingestion of low-frequency static data (INE demographics, MITMA zoning, Calendars). It establishes the schema foundations and Bronze/Silver dimension tables.
2. **Mobility Ingestion DAG:** A parameterized worker pipeline designed for high-volume processing. It accepts date ranges to ingest, clean, and transform the daily mobility files (MITMA OD Matrices) from Bronze to Silver using atomic tasks.
3. **Gold Generations and consultings DAGs:** These pipelines (DAGs 31–33) consolidate the logic into unified flows. Triggered on-demand with custom parameters (e.g., specific spatial polygons or date ranges), they first materialize the aggregated Gold tables using DuckDB and immediately generate the final visual assets (Kepler.gl maps, Matplotlib charts), uploading them to S3 in a single execution context.

4.2 Infrastructure & Dimensions (DAG 1)

The foundational pipeline, `infrastructure_and_dimensions`, is responsible for establishing the Lakehouse schema and ingesting low-velocity reference data. Unlike the daily mobility processing, this DAG is designed to be executed on-demand or annually, as zoning definitions and census statistics change infrequently.

The workflow is architected into three distinct logical phases to maximize parallelism while ensuring referential integrity.

4.2.1 Phase 1: Infrastructure Initialization

Before data movement begins, the pipeline ensures the environment is consistent. The task called `create_schemas` connects to the Neon catalog to verify the existence of the logical namespaces (*bronze, silver, gold*). Simultaneously, the `create_stats_table` task initializes the `data_quality_log` table in the Silver layer, which is a prerequisite for audit logging in all subsequent pipelines.

4.2.2 Phase 2: Parallel Bronze Ingestion

To optimize total execution time, all external data extraction tasks are configured to run in parallel. These tasks use the Airflow *@task* notation arguments to specify 3 retries with a gap of 1 minute in case they fail. This phase handles heterogenous data formats through specialized ingestion logic:

- **Spatial Ingestion:** The `br_ingest_geo_data` task utilizes DuckDB's `spatial` extension. It performs an HTTP HEAD request to validate the existence of the remote Shapefiles (.shp, .shx, .dbf) before executing `ST_Read` to ingest the vector geometry directly into the Bronze layer.
- **Statistical Ingestion:** Tasks targeting INE data (e.g., `br_ingest_ine_rent`) implement specific handling for legacy formats, forcing `ISO-8859-1` encoding and tab separators to prevent parsing errors common in Spanish government datasets.
- **Dictionaries:** Tasks for zoning and mapping ingest standard CSVs via DuckDB's `read_csv_auto`, adding audit columns (`source_url`, `ingestion_timestamp`) on the fly.

4.2.3 Phase 3: Silver Transformation and Convergence

The dependency structure in the Silver layer follows a "Funnel" pattern (Figure 4). The system enforces a hard dependency on the creation of the master dimension table before populating satellite metric tables.

1. **Convergence Point (`dim_zones`):** This task acts as the synchronization barrier. It waits for the completion of three specific Bronze tasks—Geometry, Zoning names, and INE Mapping. Once available, it performs a 3-way join to construct the master `dim_zones` table, generating the surrogate `zone_id`.
2. **Satellite Expansion:** Once the master dimension exists, the downstream tasks called `metric_population`, `metric_ine_rent`, `dim_zone_distance_matrix` and `dim_zone_holidays` trigger in parallel. Each task performs a join between its respective raw Bronze source and the newly created `dim_zones` to assign the correct surrogate keys to the statistical data.

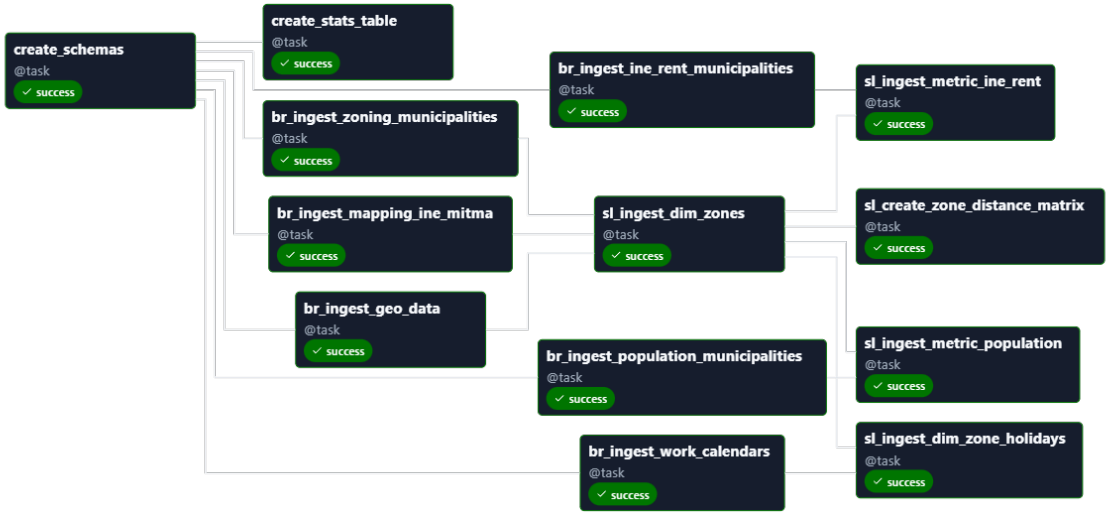

Figure 4: Dependency graph of the Infrastructure DAG. Note the convergence of spatial and dictionary data into `dim_zones` before metric processing.

4.3 Mobility Ingestion (DAG 2)

The second pipeline, `mobility_ingestion`, constitutes the core factory of the Lakehouse. It is responsible for the daily ingestion and transformation of the high-volume Origin-Destination matrices. Given that the source data is partitioned by day (one compressed CSV per day), this DAG utilizes Airflow's Dynamic Task Mapping to scale horizontally, processing date ranges supplied via run-time parameters.

4.3.1 Phase 1: Dynamic Scheduling and Initialization

Unlike the static infrastructure DAG, this workflow begins by generating a execution plan based on user input. The `generate_date_list` task parses the `start_date` and `end_date` parameters to produce a list of strings (e.g., `['20230101', '20230102', ...]`).

Simultaneously, the pipeline executes idempotent initialization tasks (`ensure_br_mobility...` and `ensure_sl_mobility...`). These tasks utilize `CREATE TABLE IF NOT EXISTS` logic to prepare the partitioning structures in the Bronze and Silver layers, ensuring that parallel workers do not encounter race conditions when attempting to write to non-existent tables.

4.3.2 Phase 2: Resilient Bronze Ingestion

The `br_process_single_day` task is mapped over the generated date list. Each instance handles a specific day, implementing robust checks to handle the known inconsistencies in the MITMA source server:

- **Pre-flight Validation:** Before attempting extraction, the task performs an HTTP `HEAD` request to the calculated URL. If the source file is missing (a common occurrence for specific dates in late 2023), the task logs a warning and skips execution gracefully rather than failing the pipeline.
- **Throttling and Retries:** To prevent overwhelming the source server or saturating local memory, concurrency is strictly limited via `max_active_tis_per_dag=2`. Network resilience is enforced with a retry policy of 5 attempts with a 30-second delay.
- **Streaming Ingestion:** Valid files are streamed directly from the remote server into the `mobility_data` table using DuckDB's `read_csv_auto`, bypassing local disk storage.

4.3.3 Phase 3: Silver Transformation

The transformation phase (`sl_process_single_day`) mirrors the mapping of the ingestion phase but introduces a strict cross-layer dependency: Silver processing for the batch only begins once Bronze ingestion is confirmed.

Inside this task, raw text data is transformed into the analytical schema. This involves joining the raw mobility records with the `dim_zones` table (created in DAG 1) to resolve MITMA/INE codes into internal surrogate keys. This effectively acts as a foreign key validation constraint; records with invalid zone codes are implicitly filtered during the inner join, ensuring referential integrity in the Silver layer.

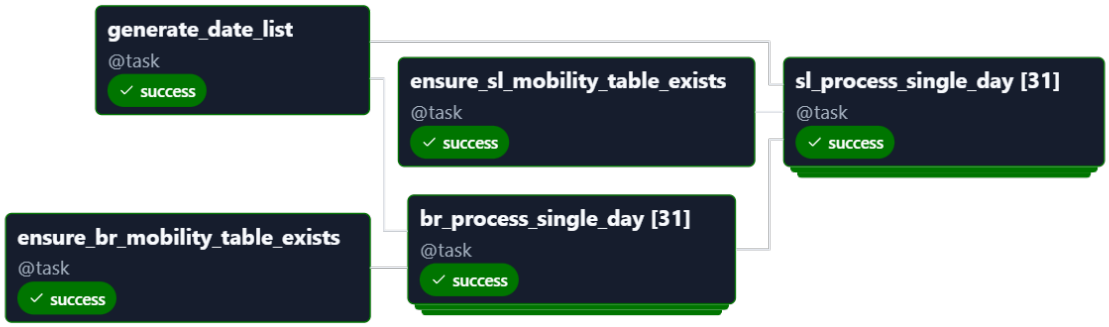

Figure 5: The Mobility Ingestion DAG utilizing Dynamic Task Mapping. The central block expands into N parallel tasks corresponding to the number of days in the requested date range.

4.4 Gold Generations and consultings (DAGs 3)

The analytical layer is materialized through three dedicated Airflow pipelines. Each DAG is designed to answer a specific business question by transforming Silver data into aggregated Gold tables and subsequently generating visual reports stored in S3.

- **DAG 31: Typical Day Clustering** (`31_bq1_clustering`)
  This pipeline addresses the characterization of mobility patterns.
  - **Transformation:** It aggregates hourly trip profiles from `silver.fact_mobility` and extracts them into a Python task. Using Scikit-Learn, it applies a K-Means algorithm ($k = 3$) to classify days into distinct patterns (e.g., Weekdays vs. Weekends). The results are persisted in the `gold.typical_day_patterns` table.
  - **Output:** The DAG queries the generated Gold table to plot the temporal profiles using Matplotlib. The resulting image (`.png`) is uploaded to S3, visualizing the average hourly intensity for each identified cluster.
- **DAG 32: Infrastructure Gap Analysis** (`32_bq2_gaps`)
  This pipeline implements the Gravity Model to identify infrastructure deficits.
  - **Transformation:** It executes a batch SQL job to create the `gold.infrastructure_gaps` table. This process joins mobility flows with population and economic metrics, calculating the "Gravity Score" and comparing it against actual trips to derive a `mismatch_ratio` for every origin-destination pair.
  - **Output:** The DAG includes two visualization tasks using Kepler.gl. One generates a bubble map ranking zones by service level, and the other generates an arc map

highlighting specific connections where potential demand exceeds actual supply. Both HTML reports are saved to S3.

- **DAG 33: Functional Classification (`33_bq3_functional_classification`)**
  This pipeline categorizes municipalities based on their role in the metropolitan network.
  - **Transformation:** It filters mobility data via spatial polygons and calculates inflow/outflow ratios. A logic-based SQL query applies a decision tree to assign labels (e.g., "Bedroom Community", "Activity Hub") to each zone, storing the result in `gold.zone_functional_classification`.
  - **Output:** The final task joins these labels with the zone geometries to generate a Kepler.gl Choropleth map. This interactive dashboard allows urban planners to visualize the functional structure of the region and is automatically published to S3.

5 Results and Use Cases

5.1 Use Case 1: Typical Mobility Patterns

To characterize the "typical day", we aggregated hourly flows and applied K-Means clustering...

Figure 6: Hourly mobility profiles identified (Weekdays vs. Weekends).

5.2 Use Case 2: Infrastructure Gap Analysis

The second business objective addresses the identification of zones where transport infrastructure fails to meet potential demand. To solve this, we implemented a **Gravity Model** in the Gold layer, comparing the theoretical interaction potential between municipalities against the actual mobility flows recorded by MITMA.

Implementation Approach

The solution is orchestrated via an Airflow DAG (`32_bq2_gaps`). The logic follows a three-step process:

1. **Theoretical Modeling:** We calculate a "Gravity Score" for every origin-destination pair using the classic impedance formula:

$$T_{ij} = k \cdot \frac{P_i \cdot E_j}{d_{ij}^2} \tag{1}$$

   where P_i is the population at the origin (from `metric_population`), E_j is the economic attractiveness (using `income_per_capita` from `metric_ine_rent` as a proxy), and d_{ij} is the distance from our pre-computed `dim_zone_distances`.

2. **Dynamic Normalization (k-factor):** To make the theoretical score comparable to physical trips, the code dynamically calculates a normalization factor (k) by computing the ratio between the total sum of actual MITMA trips and the total sum of gravity scores for the period.

3. **Gap Detection:** The final metric, `mismatch_ratio`, is derived by dividing actual trips by the normalized potential. A ratio significantly below 1.0 indicates a connection where mobility is lower than demographic and economic factors suggest, highlighting a potential infrastructure deficit.

Spatial Analytics and Visualization

The pipeline includes a spatial filtering mechanism allowing users to run this analysis on specific regions (e.g., passing a WKT Polygon for the "Comunidad de Madrid"). Using DuckDB's spatial extension (`ST_Intersects`), the system aggregates the results to generate two key visualizations using Kepler.gl:

- **Service Level Ranking:** A bubble map where zones are sized by importance ($P \times E$) and colored by their average service level, instantly highlighting underserved economic hubs.
- **Mobility Arcs:** A flow map visualizing specific origin-destination connections that show the highest disparity between potential and actual traffic.

5.3 Use Case 3: Functional Zoning Classification

The third analytical product moves beyond simple traffic counting to characterize the *functional role* of each municipality within the metropolitan network. By analyzing the directionality and retention of mobility flows, this use case automatically classifies zones into semantic categories (e.g., "Bedroom Communities" vs. "Activity Hubs"), aiding urban planners in understanding regional dependencies.

Methodology and Classification Logic

The implementation is encapsulated in the DAG `33_bq3_functional_classification`. The logic aggregates the granular records from `fact_mobility` into three distinct flow types per zone: **Inflow** (trips entering), **Outflow** (trips leaving), and **Internal** (trips starting and ending in the same zone).

Using these aggregates, two key indices are calculated:

1. **Net Flow Ratio:** Measures the balance of attraction.

$$R_{\text{net}} = \frac{\text{Inflow} - \text{Outflow}}{\text{Inflow} + \text{Outflow}}$$

2. **Retention Rate:** Measures the zone's self-sufficiency.

$$R_{\text{retention}} = \frac{\text{Internal}}{\text{Outflow} + \text{Internal}}$$

Based on these metrics, the SQL pipeline applies a hierarchical decision tree to assign a `functional_label` to each municipality:

- **Self-Sustaining Cell:** Zones with high retention ($R_{\text{retention}} > 0.20$), indicating a municipality that satisfies most of its residents' needs locally.
- **Activity Hub (Importer):** Zones with a positive net flow ($R_{\text{net}} > 0$), typically business districts or commercial centers attracting commuters.
- **Bedroom Community (Exporter):** Zones with a negative net flow ($R_{\text{net}} < 0$), characterizing residential areas where the population commutes out for work.
- **Balanced / Transit Zone:** Areas with neutral flow ratios acting as connectors.

Visualization Strategy

The results are materialized in a Kepler.gl dashboard combining two visual layers to provide context:

- **Polygon Layer (Choropleth):** Displays the administrative boundaries colored by their `functional_label`. This reveals regional clusters (e.g., a ring of "Bedroom Communities" surrounding a central "Activity Hub").
- **Centroid Layer:** Displays bubbles sized by `total_activity` (Sum of In + Out + Internal), allowing users to distinguish between major metropolitan centers and smaller rural nodes regardless of their functional classification.

6 Discussion

The implemented lakehouse allows experts to query billions of rows using standard SQL without managing infrastructure...

7 Conclusions and Limitations

The project successfully established a scalable 3-tier architecture. However, limitations exist regarding data anomalies in late 2023...

References

[1] Spanish Ministry of Transport (MITMA). *Open Data Mobility*. Available at: https://www.transportes.gob.es/ministerio/proyectos-singulares/estudios-de-movilidad-con-big-data/opendata-movilidad (Accessed on 4 January 2026).

[2] Spanish National Statistics Institute (INE). Available at: https://www.ine.es/ (Accessed on 4 January 2026).

[3] Centro Nacional de Información Geográfica (CNIG). Available at: https://www.ign.es/ (Accessed on 4 January 2026).

[4] Ayuntamiento de Madrid. *Portal de Datos Abiertos*. Available at: https://datos.madrid.es/ (Accessed on 4 January 2026).

[5] DuckDB Documentation. Available at: https://duckdb.org/ (Accessed on 4 January 2026).

[6] Amazon Web Service Batch. *AWS Batch Documentation*. Available at: https://aws.amazon.com/es/batch/ (Accessed on 4 January 2026).

[7] DuckLake. *DuckLake Documentation*. Available at: https://ducklake.select/docs/stable/ (Accessed on 4 January 2026).

[8] Apache Software Foundation. *Apache Airflow Documentation*. Available at: https://airflow.apache.org/ (Accessed on 4 January 2026).