Real Estate Data Engineering Pipeline – Technical Documentation

1. Overview

This project implements a complete data-engineering workflow for a real-estate company.
The goal of the pipeline is to:

- Extract daily **Leads** and **Sales** data
- Transform the raw data into a Star Schema
- Load the cleaned and modeled data into a data warehouse (MySQL)
- Automate all steps using Apache Airflow

---

2. Understanding the Source Data

Before designing the warehouse, I started by exploring the raw Excel sheets:

- **DE LEADS**
- **DE SALES**

I loaded both sheets into temporary MySQL tables using a simple ingestion script.

```
leads_df = pd.read_excel(file_path, sheet_name="DE LEADS")
sales_df = pd.read_excel(file_path, sheet_name="DE SALES")
leads_df.to_sql("de_leads_raw", engine, if_exists="replace", index=False)
sales_df.to_sql("de_sales_raw", engine, if_exists="replace", index=False)
```

This allowed me to understand each column, its datatype, and its business meaning – as much as I could – which helped determine which attributes should become **dimensions** and which metrics should belong to **fact** tables.

---

3. Environment Setup

Tools & Technologies

- **MySQL** (local)
- **Apache Airflow (Docker-based)**
- **SQLAlchemy**

- **Pandas**
- **OpenPyXL**
- **MySQL Workbench** (for ERD Reverse Engineering)

Challenges

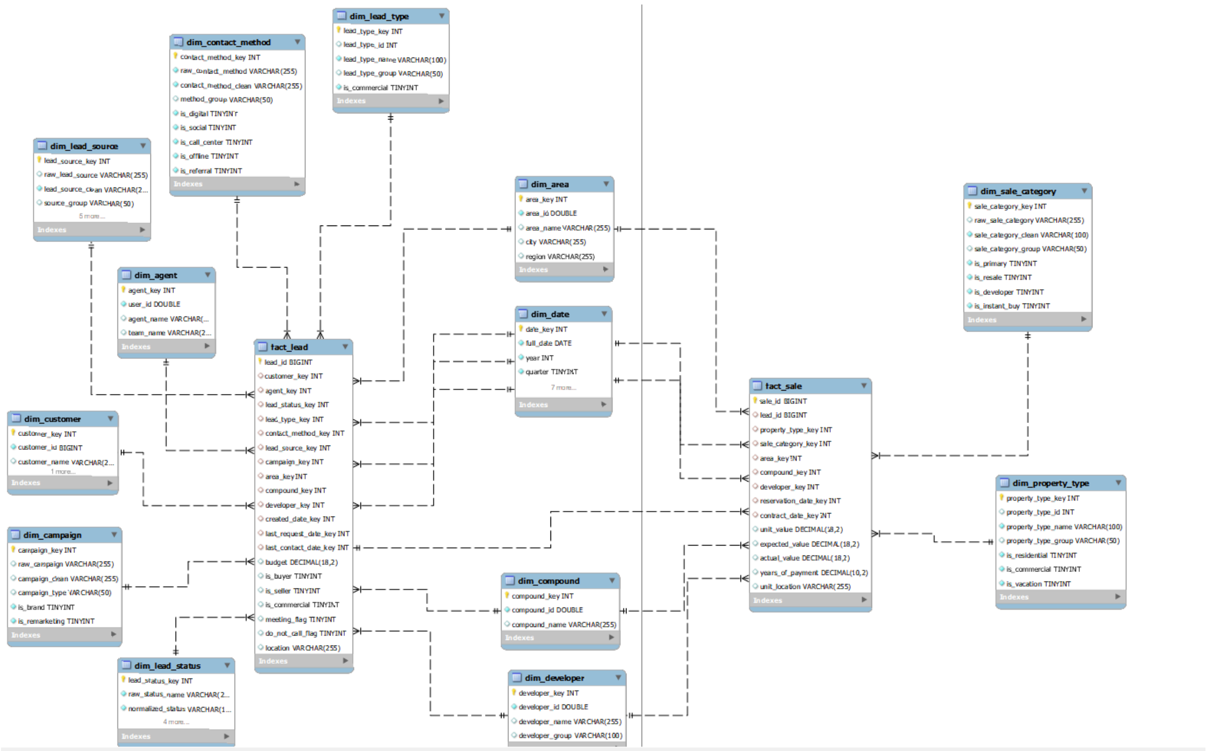
- This was my first time setting up and working with Airflow and I had to configure extra Python packages inside the Airflow container (e.g., pandas, openpyxl, pymysql).
- Time constraints while designing and testing each step of the pipeline.

---

4. High-Level Architecture

4.1 ERD

I generated the ERD automatically using the **Reverse Engineering** feature in MySQL Workbench after loading the DWH tables.



4.2 Logical flow:

1. **Ingestion (Python / Airflow)**
   - Read Excel → pandas

- Write to `realestate_source.de_leads_raw` & `realestate_source.de_sales_raw`

2. **Staging Layer (`realestate_stg`) – `01_staging.sql`**
   - Create:
     - `stg_leads`
     - `stg_sales`
   - These are **cleaned/renamed** versions of raw tables:
     - Rename IDs, basic type/column standardization.
3. **DWH Layer (`realestate_dwh`)**
   - **Lead dims – `02_lead_dims.sql`**
     - `dim_lead_status`: funnel mapping, win/loss/active flags.
     - `dim_lead_type`: primary vs resale vs commercial vs referral broker, with `is_commercial` flag.
     - `dim_contact_method`: cleans `method_of_contact` into categories like:
       - `facebook`, `instagram`, `search/web`, `website form`, `paid ads form`, `referral`, `phone/sms`, `offline`, `broker/partner`, etc.
     - `dim_lead_source`: normalizes `lead_source` to buckets like `facebook`, `instagram`, `google/search`, `website/form`, `broker/partner`, `referral`, `offline`, `internal`, etc.
     - `dim_campaign`: cleans raw campaign names, decodes `%20`, classifies into types:
       - `BRAND`, `REMARKETING`, `PROSPECTING`, `SMS`, `PERFORMANCE_DISPLAY`, `OTHER`.
   - **Core dims – `03_core_dims.sql`**
     - `dim_date` with:
       - `date_key` (YYYYMMDD)
       - `year`, `quarter`, `month`, `month_name`, `day_of_month`, `day_of_week`, `day_name`, `week_of_year`, `is_weekend`.
       - Populated by stored procedure `populate_dim_date('2010-01-01', '2040-12-31')`.
     - `dim_customer` from distinct `customer_id`.
     - `dim_agent` from distinct `user_id`.
     - `dim_area` from `area_id` (leads/sales).
     - `dim_compound` from `compound_id`.
     - `dim_developer` from `developer_id` (for future richer dev analytics).
   - **Sales-specific dims – `04_sales_dims.sql`**
     - `dim_property_type`:
       - Normalizes property type into groups:

                ■ `RESIDENTIAL_APT`, `RESIDENTIAL_VILLA`, `VACATION`, `COMMERCIAL`, `OTHER`.
            ■ Flags: `is_residential`, `is_commercial`, `is_vacation`.
        ■ `dim_sale_category`:
            ■ `Primary`, `Resale Buyer`, `Resale Seller`, `Developer Resale`, `Developer Commercial Sale`, `Nawy Now`, etc.
            ■ Groups: `PRIMARY`, `RESALE`, `DEVELOPER_COMMERCIAL`, `INSTANT_BUY`, `OTHER`.
            ■ Flags: `is_primary`, `is_resale`, `is_developer`, `is_instant_buy`.

4. **Facts**

   `fact_lead` – `05_fact_lead.sql`
    - Grain: **1 row per lead_id (latest snapshot)**.
    - Surrogate FKs:
        - `customer_key`
        - `agent_key`
        - `lead_status_key`
        - `lead_type_key`
        - `contact_method_key`
        - `lead_source_key`
        - `campaign_key`
        - `area_key`
        - `compound_key`
        - `developer_key`
        - `created_date_key`, `last_request_date_key`, `last_contact_date_key`
    - Measures / attributes:
        - `budget`
        - `is_buyer`, `is_seller`, `is_commercial`
        - `meeting_flag`, `do_not_call_flag`
        - `location`
    - Uses `ROW_NUMBER() OVER (PARTITION BY lead_id ORDER BY COALESCE(updated_at, created_at) DESC)` to keep **only the latest version** of each lead.
    - Indexes on natural keys in dim tables (customer_id, user_id, area_id, etc.) to speed up joins.

5. **`fact_sale` – `06_fact_sale.sql`**
    - Grain: **1 row per sale_id**.
    - Links back to:
        - `fact_lead` via `lead_id` (FK).

        - `dim_property_type`, `dim_sale_category`, `dim_area`, `dim_compound`, `dim_date` (reservation/contract).
    - Measures / attributes:
        - `unit_value`, `expected_value`, `actual_value`
        - `years_of_payment`
        - `unit_location`

5. Ingestion & Orchestration (Airflow)

- **ingestion.py**:
    - Uses `MySqlHook(mysql_conn_id="mysql_source")` to get an SQLAlchemy engine.
    - Reads both Excel sheets via pandas and writes into:
        - `realestate_source.de_leads_raw`
        - `realestate_source.de_sales_raw`
    - Performs simple count checks (source vs DB).

- **realestate_dag.py**:
    - Defines DAG `realestate_pipeline`.
    - Uses a helper `run_sql_file(sql_path)` that opens a `.sql` and executes statements one by one with SQLAlchemy.
    - Defines `run_ingestion()` (local version) that also reads Excel and writes into `realestate_source.de_leads_raw` and `de_sales_raw`.
    - Tasks:
        - `ingest_excel` → PythonOperator calling `run_ingestion`
        - `build_staging` → runs `01_staging.sql`
        - `build_lead_dims` → `02_lead_dims.sql`
        - `build_core_dims` → `03_core_dims.sql`
        - `build_sales_dims` → `04_sales_dims.sql`
        - `build_fact_lead` → `05_fact_lead.sql`
        - `build_fact_sale` → `06_fact_sale.sql`
    - Task dependency chain:
      `ingest >> stg >> lead_dims >> core_dims >> sales_dims >> fact_lead >> fact_sale`

6. Analytics & KPI Layer

6.1 Lead Funnel Distribution

Shows how leads are distributed across different funnel stages.

```sql
-- Funnel distribution
SELECT
    dls.funnel_stage,
    COUNT(*) AS leads_count
FROM fact_lead f
JOIN dim_lead_status dls ON dls.lead_status_key = f.lead_status_key
GROUP BY dls.funnel_stage
ORDER BY leads_count DESC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

funnel_stage	leads_count
DISQUALIFIED	15489
CONTACT_ATTEMPT	12764
TO_CONTACT	12484
INTERNAL_REASSIGN	10897
OTHER	1718
MEETING_HELD	1372
WON	604
MEETING_BOOKED	314
RESERVATION	96
CONTRACT	63
MEETING_CANCELED	44

6.2 Funnel Stage Counts

Aggregated counts for each stage of the funnel.

```sql
-- Funnel counts by stage
SELECT
    SUM(dls.funnel_stage = 'TO_CONTACT')      AS to_contact,
    SUM(dls.funnel_stage = 'CONTACT_ATTEMPT') AS contacted,
    SUM(dls.funnel_stage = 'MEETING_BOOKED')  AS meeting_booked,
    SUM(dls.funnel_stage = 'MEETING_HELD')    AS meeting_held,
    SUM(dls.funnel_stage = 'RESERVATION')     AS reservation,
    SUM(dls.funnel_stage = 'CONTRACT')        AS contract_stage,
    SUM(dls.funnel_stage = 'WON')             AS sales
FROM fact_lead f
JOIN dim_lead_status dls ON dls.lead_status_key = f.lead_status_key;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

to_contact	contacted	meeting_booked	meeting_held	reservation	contract_stage	sales
12484	12764	314	1372	96	63	604

6.3 Agent Performance

Summaries leads handled, meetings done, won deals, and win rate.

```
SELECT
    da.user_id as agent_id,
    COUNT(*) AS total_leads,
    SUM(dls.is_won) AS sales,
    SUM(dls.funnel_stage = 'MEETING_HELD') AS meetings,
    ROUND(SUM(dls.is_won) / COUNT(*) * 100, 2) AS win_rate_pct
FROM fact_lead f
JOIN dim_agent da       ON da.agent_key       = f.agent_key
JOIN dim_lead_status dls ON dls.lead_status_key = f.lead_status_key
GROUP BY da.user_id
ORDER BY sales DESC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

agent_id	total_leads	sales	meetings	win_rate_pct
2	388	28	2	7.22
150	809	19	2	2.35
239	67	13	0	19.40
135	38	12	1	31.58
222	38	11	6	28.95
442	92	10	8	10.87
189	287	10	8	3.48
336	46	9	0	19.57
181	13	9	0	69.23
387	44	8	2	18.18
259	33	8	10	24.24
156	20	8	1	40.00

6.4 Lead Source Performance

Measures which marketing channels generate the best leads.

```sql
SELECT
    dlsrc.source_group,
    COUNT(*) AS total_leads,
    SUM(dls.is_won) AS sales,
    ROUND(SUM(dls.is_won) / COUNT(*) * 100, 2) AS win_rate_pct
FROM fact_lead f
JOIN dim_lead_source dlsrc ON dlsrc.lead_source_key = f.lead_source_key
JOIN dim_lead_status dls   ON dls.lead_status_key   = f.lead_status_key
GROUP BY dlsrc.source_group
ORDER BY sales DESC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

source_group	total_leads	sales	win_rate_pct
OTHER	5482	292	5.33
SOCIAL	32669	117	0.36
SEARCH	9590	89	0.93
OWNED_MEDIA	4499	34	0.76
OFFLINE	2815	29	1.03
REFERRAL	164	27	16.46
PARTNER	155	7	4.52
MESSAGING	166	5	3.01
INTERNAL	15	0	0.00
PORTAL	40	0	0.00

6.5 Sales Financial KPIs

```sql
SELECT
    COUNT(*)                AS total_sales,
    SUM(actual_value)       AS total_actual_value,
    AVG(actual_value)       AS avg_actual_value,
    SUM(expected_value)     AS total_expected_value,
    AVG(expected_value)     AS avg_expected_value,
    SUM(unit_value)         AS total_unit_value,
    AVG(unit_value)         AS avg_unit_value,
    AVG(years_of_payment)   AS avg_years_of_payment
FROM realestate_dwh.fact_sale;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

total_sales	total_actual_value	avg_actual_value	total_expected_value	avg_expected_value	total_unit_value	avg_unit_value	avg_years_of_payment
1567	6405783466.00	8037369.468005	4431103624.00	7653028.711572	9722558203.00	6789495.951816	7.188764

7. Next Enhancements

(1) Make Dimensions & Facts Incremental, Not Full Rebuild

Currently, tables are dropped & recreated.
Instead:

- Use `INSERT … ON DUPLICATE KEY UPDATE`
- Keep `is_current` flags
- Use incremental logic on updated_at

(2) Add Data Quality Checks

Great expectations or SQL-based tests:

- Not null checks
- Unique keys
- Referential integrity
- Value distribution checks