

# Database Project Report

Traffic Flow Analysis for Urban Planning in Zurich | Team ACID

Ramiro Díez-Liébaná, Valeska Blank, Dongyuan Gao, Cyriel Van Helleputte

2025-11-28

## Contents

<b>1</b>	<b>Introduction &amp; Context (Proposal)</b>	<b>3</b>
<b>2</b>	<b>Project Idea &amp; Use Case (Proposal)</b>	<b>3</b>
<b>3</b>	<b>Key Performance Indicators (KPIs)</b>	<b>3</b>
3.1	KPI 1: District-Level Stress Index . . . . .	3
3.2	KPI 2: Peak-Hour Bottleneck Identification ( <i>only if deeper analysis is required</i> ) . . .	4
3.3	KPI 3: Directional Imbalance ( <i>optional, counting-site level</i> ) . . . . .	4
3.4	KPI 4: Dashboard & Visualization Plan . . . . .	4
3.5	KPI 5: Provide Actionable Insights for Planning Decisions . . . . .	5
<b>4</b>	<b>Data Model &amp; Database Schema</b>	<b>5</b>
4.1	Data Preprocessing & Preparation . . . . .	5
4.2	Entity-Relationship Model . . . . .	9
<b>5</b>	<b>Loading &amp; Transforming the Data</b>	<b>11</b>
5.1	Two-Stage Loading Strategy . . . . .	11
5.2	Schema and Table Creation . . . . .	11
<b>6</b>	<b>Analyzing &amp; Evaluating Data</b>	<b>14</b>
<b>7</b>	<b>Efficiency &amp; Query Performance</b>	<b>14</b>
<b>8</b>	<b>Visualization &amp; Decision Support</b>	<b>14</b>
<b>9</b>	<b>Conclusions &amp; Lessons Learned</b>	<b>14</b>
<b>10</b>	<b>Individual Team Member Reflections (required)</b>	<b>14</b>
10.1	Individual Member Structure . . . . .	14
<b>11</b>	<b>Generative AI Declaration &amp; Guidelines</b>	<b>15</b>
<b>12</b>	<b>How to render this report</b>	<b>15</b>
<b>13</b>	<b>Appendix A: Traffic Data – Complete ETL Script</b>	<b>16</b>

13.1	Staging Table Creation . . . . .	16
13.2	Load Raw Traffic CSV into Staging . . . . .	16
13.3	Production Schema Creation (Normalized Tables) . . . . .	17
13.4	Transformation into Production Schema . . . . .	18
13.5	Final Cleanup . . . . .	19
<b>14</b>	<b>Appendix B: Quarter Data – Complete ETL Script</b>	<b>20</b>
14.1	Staging Table Creation . . . . .	20
14.2	Load Raw Quarter CSV into Staging . . . . .	20
14.3	Create Normalized Quarter Table . . . . .	20
14.4	Transformation: Deduplication & Type Conversion . . . . .	21
14.5	Cleanup Staging Table . . . . .	21
<b>15</b>	<b>Appendix C: Population Data – Complete ETL Script</b>	<b>21</b>
15.1	Staging Table Creation . . . . .	22
15.2	Load Raw Population CSV into Staging . . . . .	22
15.3	Create Normalized Population Table . . . . .	22
15.4	Transformation: Cleaning, Casting, Filtering . . . . .	23
15.5	Cleanup Staging Table . . . . .	23
<b>16</b>	<b>Appendix D: Lookup Tables – Complete ETL Script</b>	<b>23</b>
16.1	Staging Table Creation . . . . .	24
16.2	Load Lookup Fields from Raw CSV . . . . .	24
16.3	Create Normalized Lookup Tables . . . . .	24
16.4	Populate Lookup Tables (Deduplicated) . . . . .	25
16.5	Normalize Population Table by Removing Redundant Columns . . . . .	25
16.6	Add Foreign Key Constraints to Population . . . . .	25
16.7	Cleanup Staging Table . . . . .	26

# 1 Introduction & Context (Proposal)

The City of Zurich faces growing challenges related to traffic congestion, mobility planning, and urban development. As the population increases and commuting patterns evolve, many intersections and arterial roads experience significant pressure, particularly during peak hours. Congestion affects not only travel times but also road safety, air quality, and the overall effectiveness of the city's transport network.

Urban planning authorities must therefore make complex decisions about where to prioritize infrastructure investments, how to optimize traffic signals, and which areas require redesign or alternative mobility solutions. These decisions rely heavily on an accurate understanding of traffic flows, temporal patterns, directional imbalances, and long-term trends. High-quality traffic data and transparent analytical methods are essential to support evidence-based planning, enable more targeted interventions, and ensure that resources are allocated efficiently.

To contribute to this goal, this project focuses on deriving meaningful insights and key performance indicators (KPIs) that can inform data-driven urban planning decisions in line with the OECD Data Value Cycle, which emphasizes the transformation of raw data into actionable value for public administration.

## 2 Project Idea & Use Case (Proposal)

To support Zurich's urban planners in addressing congestion and mobility challenges, this project aims to transform traffic count data into meaningful indicators that can guide evidence-based decision-making. The central use case focuses on identifying traffic pressure points at intersections and understanding how traffic patterns vary by time, location, and direction.

## 3 Key Performance Indicators (KPIs)

The KPI suite was first developed collaboratively by the team to ensure that every indicator directly supports the Zurich urban-planning use case. The procedure followed three steps: (1) define the analytical questions for districts and counting sites, (2) design SQL-ready calculations and classification thresholds, and (3) allow minor adjustments later in the project as additional data nuances emerged. This predefined framework now guides all downstream analytics and remains stable unless new evidence requires a targeted refinement.

### 3.1 KPI 1: District-Level Stress Index

**Purpose.** Compare traffic volume growth with population growth for each district to detect commuter hubs and residential pressure zones.

**KPI 1.1 – Aggregate Population by District & Year.** Join `Population` with `Quarter` and sum `population_count` per `city_district` for 2012 vs. 2025 (standard `SELECT city_district, SUM(population_count)` grouped by district and year).

**KPI 1.2 – Aggregate Traffic Volume by District & Year.** Use the `TrafficMeasurement` → `MeasurementSite` → `CountingSite` → `Quarter` join path to sum measured hourly `vehicle_count` per district for 2012 and 2025 (group by `city_district` and `YEAR(timestamp)`).

**KPI 1.3 – Calculate Growth Rates.**

- Population Growth (%) = ((Pop\_2025 - Pop\_2012) / Pop\_2012) × 100
- Traffic Growth (%) = ((Traffic\_2025 - Traffic\_2012) / Traffic\_2012) × 100

**KPI 1.4 – Stress Index.** Stress Index = Traffic Growth % - Population Growth %

Interpretation: positive values highlight commuter hubs; negative values show residential pressure; near zero indicates balanced growth.

**KPI 1.5 – Classification.**

- Stress Index > +10% : High commuter pressure
- Stress Index < -10% : High residential pressure
- -10% ≤ Stress Index ≤ +10% : Balanced

Sample output:

District	Pop Growth %	Traffic Growth %	Stress Index	Classification
1	15.2%	8.3%	-6.9%	Balanced
2	12.5%	22.1%	+9.6%	Commuter Hub

### 3.2 KPI 2: Peak-Hour Bottleneck Identification (*only if deeper analysis is required*)

**Purpose.** Detect counting sites where peak-hour traffic consistently exceeds capacity.

**KPI 2.1 – Average Hourly Traffic per Counting Site.** Calculate `AVG(vehicle_count)` grouped by `counting_site_id` and `HOOR(timestamp)` for measured rows from 2023–2025.

**KPI 2.2 – Peak Hour per Site.** For each counting site (e.g., Seestrasse, Wollishofen), pick the hour with the maximum average volume.

**KPI 2.3 – Bottleneck Threshold.** Flag sites where peak-hour volume exceeds 800 vehicles/hour. Example:

Counting Site Name	Peak Hour	Avg Volume	Status
Hardbrücke	08:00	1,245	Bottleneck
Bellevue	17:00	1,102	Bottleneck
Seestrasse (Wollishofen)	07:00	650	Normal

### 3.3 KPI 3: Directional Imbalance (*optional, counting-site level*)

**Purpose.** Reveal inbound vs. outbound imbalance using `MeasurementSite.direction`. Aggregate inbound and outbound totals per counting site (e.g., `SUM(CASE WHEN direction='inbound' ...)`), compute the ratio, and highlight sites where the inbound/outbound ratio is above 1.5 or below 0.67.

### 3.4 KPI 4: Dashboard & Visualization Plan

- **Visual 1: District Stress Index Map** – Heatmap based on KPI 1 results.
- **Visual 2: Peak-Hour Heatmap** – Time-of-day heatmap per counting site using KPI 2 outputs.

- **Visual 3: Growth Trend Comparison** – Grouped bar chart/scatterplot (population vs. traffic growth per district).
- **Additional Visuals** – e.g., Any additional visualizations that the team discovers while doing the analysis.

### 3.5 KPI 5: Provide Actionable Insights for Planning Decisions

**Purpose.** Convert KPI outputs into budget and intervention guidance for the Urban Planning Office.

#### KPI 5.1 – District Priorities.

Priority	Stress Index Range	Recommended Action
<b>High</b>	< -15% (residential pressure)	Expand public transit, add lanes, optimize signals
<b>High</b>	> +15% (commuter pressure)	Improve inbound capacity, park-and-ride facilities
<b>Medium</b>	-15% to -10% or +10% to +15%	Monitor trends, reassess in two years
<b>Low</b>	-10% to +10% (balanced)	Maintain current infrastructure

#### KPI 5.2 – Site(street)-Level Interventions.

- Top five bottleneck streets : immediate signal-timing optimization or capacity studies.
- Streets with directional imbalance > 1.5 : directional lane adjustments or reversible lanes.

#### KPI 5.3 – Budget Allocation Guide.

1. Short-term (1–2 years): immediate action on the top ten bottleneck intersections flagged in KPI 2.
2. Long-term (3–5 years): invest in infrastructure with persistent stress index extremes.

Together, these five KPIs provide a structured, predefined guideline for our project and analysis.

## 4 Data Model & Database Schema

### 4.1 Data Preprocessing & Preparation

- Hourly **traffic count** data, **address** data (including city district and quarter) and **population** data from the City of Zurich’s open data portal was collected.
- The raw datasets consisted of multiple CSV files (2012–2025).

#### 4.1.1 Raw Dataset Columns: Traffic Data

- MSID
- MSName
- ZSID
- ZSName
- Achse
- HNr
- Hoehe

- EKoord
- NKoord
- Richtung
- Knummer
- Kname
- AnzDetektoren
- D1ID–D4ID
- MessungDatZeit
- LieferDat
- AnzFahrzeuge
- AnzFahrzeugeStatus

Because the raw dataset included a mixture of analytical attributes and highly technical metadata, each field was examined to determine its relevance for traffic-flow analysis. The review was guided by the perspective of the City of Zurich’s urban planning department, which is primarily interested in temporal and spatial traffic patterns at specific locations and intersections.

#### 4.1.2 Removed Fields

- **HNr**, due to inconsistent content.
- **D1ID–D4ID**, as these detector identifiers are technical metadata without analytical value.
- **LieferDat**, which is a delivery timestamp not required for traffic analysis.

All fields describing the measurement location, the measurement configuration, and the traffic counts themselves were retained. To improve clarity and support further processing and database integration, the remaining column names were translated into English equivalents. In a further preprocessing step, several categorical values contained in German were translated to English. After all preprocessing steps were completed, the resulting cleaned dataset contained 21,721,493 rows and 14 columns.

#### 4.1.3 Cleaned Dataset Columns

- **measurement\_site\_id**: Unique technical identifier of the measurement site. A measurement site represents a specific traffic-flow direction or lane at a counting location.
- **measurement\_site\_name**: Technical name of the measurement site. In this dataset, this field contains “Unknown” for all entries.
- **counting\_site\_id**: Identifier of the counting site, representing the physical location where traffic measurements are collected.
- **counting\_site\_name**: Human-readable name of the counting site, describing the location (street).
- **axis**: Categorization of the counting site into a traffic axis (street).
- **position\_description**: A textual descriptor indicating where along the street segment the measurement site is located. Contains “Unknown” for many entries.
- **east\_coordinate**: The east coordinate of the measurement site in the Swiss CH1903+ / LV95 reference system.
- **north\_coordinate**: The north coordinate of the measurement site in the Swiss CH1903+ / LV95 reference system.
- **direction**: The direction of traffic flow being measured (e.g., “inbound”, “outbound”).

- **signal\_id**: Identifier of the associated traffic signal or intersection controller regulating traffic at the measurement site.
- **signal\_name**: Name of the associated traffic signal or intersection.
- **num\_detectors**: Number of detectors installed at the measurement site.
- **timestamp**: The timestamp indicating the start of the hourly measurement interval (ISO-8601 format).
- **vehicle\_count**: The number of vehicles recorded during the hourly measurement interval.
- **vehicle\_count\_status**: Indicates how the vehicle count was produced: “Measured”, “Missing”, or “Imputed”.

#### 4.1.4 Raw Dataset Columns: Quarters

The raw quarter dataset contained address-level information used by the City of Zurich for administrative and statistical purposes. The fields included:

- `adresse`
- `anzahl_fla_projektiert`
- `anzahl_fla_real`
- `flaeche_projektiert`
- `flaeche_real`
- `flaeche_total`
- `gwr_egid`
- `hausnummer`
- `lokalisationsname`
- `objectid`
- `stadtkreis`
- `statistisches_quartier`

**4.1.4.1 Cleaning and Transformation** To link traffic data with geographic units, the raw fields required several preprocessing steps:

- **Address splitting and cleaning**  
Street names and house numbers were embedded in a single free-text field (e.g., "Heinrich-Federer-Strasse 12A", "Widmerstrasse 88", "Seeblickstrasse 17d"). Using regular expressions, house numbers (including suffixes such as 12b, 17d) were removed, leaving a clean street-level identifier.
- **Data-type normalisation**  
`statistical_quarter` was normalised as a trimmed string. Inconsistent labels such as "Schwamend.-Mitte" were harmonised to "Schwamendingen-Mitte" to ensure joinability with the population dataset.

**4.1.4.2 Additional Standardisation for Street Name Matching** To ensure that traffic measurement locations could be linked reliably to the quarter dataset, several street names from the traffic dataset required manual standardisation. While most addresses were harmonised through regex-based cleaning, a small number of street names contained compound forms that could not be resolved automatically (e.g., combined street names, hyphenated patterns, or slash-separated names). These were corrected manually to create a consistent set of street identifiers.

Examples of manual mappings include:

- Sood-/Leimbachstrasse → Soodstrasse
- Tobelhof-/Dreiwiesenstrasse → Tobelhofstrasse
- Manessestrasse - Schimmelstrasse → Manessestrasse
- Angererstrasse Tunnelstrasse → Angererstrasse

Additionally, several counting site names referred to non-street features such as bridges, tunnels or motorway segments (e.g., *Quaibrücke*, *Milchbuckttunnel*, *A1L*). For these cases, the closest corresponding street-level name from the quarter dataset was identified manually.

These manual adjustments were essential to achieve a fully joinable set of street names between the traffic and quarter datasets and ensured that all counting sites could be assigned to a statistical quarter.

#### 4.1.4.3 Cleaned Quarter Dataset Columns

- **address**: Original full address as provided in the raw data.
- **house\_number**: Extracted house number component (kept for completeness).
- **street\_name**: Cleaned and standardised street name used as the spatial key.
- **city\_district**: Official Zurich district number (1–12).
- **statistical\_quarter**: Statistical quarter (“Statistisches Quartier”) providing fine-grained spatial units.

The cleaned quarter dataset provides the geographic reference layer used to integrate traffic measurements with demographic information.

#### 4.1.5 Raw Dataset Columns: Population

The raw population dataset consisted of quarterly demographic counts published by the City of Zurich. The original fields included:

- **StichtagDatJahr**: Year of the reference date.
- **StichtagDatMM**: Month of the reference date (numeric).
- **StichtagDatMonat**: Month of the reference date (German label).
- **StichtagDat**: Full reference date (e.g., “1998-03-31”), indicating quarterly population snapshots.
- **SexCd** / **SexLang**: Numeric code and German label representing the sex category.
- **AlterV20ueber80Sort\_noDM**, **AlterV20ueber80Cd\_noDM**, **AlterV20ueber80Kurz\_noDM**: Age-group fields (not relevant for this project).
- **HerkunftCd** / **HerkunftLang**: Numeric code and German label representing population origin.
- **KreisCd** / **KreisLang**: District number and district name.
- **QuarCd** / **QuarLang**: Statistical quarter code and label.
- **DatenstandCd**, **DatenstandLang**: Metadata describing the publication state.
- **AnzBestWir**: Population count.

**4.1.5.1 Cleaning and Transformation** To prepare the dataset for integration and analysis, several cleaning and normalisation steps were performed:

- **Column reduction**  
Multiple date-related fields were redundant. **StichtagDatJahr** was retained and renamed to **reference\_date\_year**.

- Age-group columns and metadata fields were removed.
- **Category translation and code preservation**  
German labels for sex and origin (e.g., "männlich", "weiblich", "Schweizer\*in", "Ausländer\*in") were translated to English.  
The numeric codes (`SexCd`, `HerkunftCd`) were preserved and renamed to `sex_code` and `origin_code`, as these form stable keys for later relational integration.
- **Data-type normalisation**
  - `QuarLang` was cleaned and normalised (whitespace trimming, harmonisation of abbreviations), then renamed to `statistical_quarter`.

#### 4.1.5.2 Cleaned Population Dataset Columns

- **reference\_date\_year**: Year of the population count, extracted from the quarterly reference date.
- **sex\_code**: Numeric code representing the sex category.
- **sex**: Human-readable sex label derived from `sex_code`, such as “male”, “female”, or “unknown”.
- **origin\_code**: Numeric code representing the origin category.
- **origin**: Human-readable origin label derived from `origin_code`, such as “Swiss” or “Foreign”.
- **city\_district**: Zurich district number (1–12).
- **statistical\_quarter**: Statistical quarter (“Statistisches Quartier”), harmonised to match the Quarter dataset.
- **population\_count**: Number of residents in the demographic segment.

## 4.2 Entity–Relationship Model

The cleaned and harmonised datasets were integrated into a relational data model that reflects the structure of Zurich’s traffic measurement infrastructure, its geographic units, and the associated demographic information. The final model was created in *dbdiagram.io* and is shown in the figure below.

### 4.2.1 Overview

The model is centred around the **traffic measurement process**, which is represented by three core entities:

- **CountingSite**  
Represents a physical traffic counting location (e.g., a road segment or intersection). It stores geographic coordinates, the human-readable location name, and the traffic axis.
- **MeasurementSite**  
Represents a directional or lane-specific measurement point located within a counting site. Each measurement site records traffic in a single direction and may contain multiple detectors.
- **TrafficMeasurement**  
Stores the hourly measured vehicle counts, including measurement status and timestamp.

These three entities form a 1:n:n hierarchy:

**CountingSite** → **MeasurementSite** → **TrafficMeasurement**.



Figure 1: Entity-Relationship Diagram of the integrated Traffic-Quarter-Population model.

#### 4.2.2 Traffic Signal Dimension

Some counting sites are associated with a traffic signal or intersection controller. This metadata is normalised into the **TrafficSignal** table. Since not every counting site is linked to a signal, the relationship is **optional**.

#### 4.2.3 Geographic Dimension: Quarter

The **Quarter** table provides spatial context by assigning each cleaned street and address to:

- a **city district** (1–12), and
- a **statistical quarter** (fine-grained geographic unit).

Counting sites may refer to road infrastructure that is not part of the standard quarter dataset (e.g., motorway segments, tunnels, bridges). Therefore, the relationship between **CountingSite** and **Quarter** is modelled as **optional (0–1)**.

#### 4.2.4 Demographic Dimension: Population

The **Population** table contains demographic counts by sex, origin, and year, referenced at both district and statistical-quarter level. To normalise categorical attributes, the model includes two lookup tables:

- **Sex** (sex\_code → sex)
- **Origin** (origin\_code → origin)

Every population record references a sex and origin category (mandatory), while the link to a statistical quarter is **optional**. This reflects the presence of entries such as “*Unbekannt (Stadt Zürich)*”, which cannot be assigned to a specific spatial unit.

#### 4.2.5 Optional Relationships (--0--<)

Several foreign-key relationships are intentionally optional to reflect real characteristics of the source data:

- Some counting sites lie outside the quarter system.
- Some population entries cannot be mapped to a statistical quarter.
- Not all counting sites are linked to a traffic signal.

By modelling these relationships as optional, the ERD mirrors the true structure and limitations of the underlying open data and avoids enforcing artificial or incorrect mappings.

## 5 Loading & Transforming the Data

### 5.1 Two-Stage Loading Strategy

To ensure data quality and maintain a clear separation between raw and transformed data, a two-stage workflow was applied. All CSV files were initially loaded into **staging tables** within the schema `traffic_population_zh`. These staging tables stored the raw content without constraints and served as a safe environment for validation and transformation.

This approach offered several advantages:

- **Data validation:** Raw data could be inspected before applying constraints.
- **Flexible transformations:** Cleaning and restructuring could be performed iteratively in SQL.
- **Error isolation:** Problems in staging did not affect the normalized production tables.

### 5.2 Schema and Table Creation

The production schema was created based on the Entity-Relationship Diagram (ERD) described earlier. The design follows **Third Normal Form (3NF)** and includes:

- primary keys on all dimension tables,
- foreign keys from fact tables to dimensions,
- lookup tables (`Sex`, `Origin`, `TrafficSignal`) for categorical attributes,
- a geographical reference table (`Quarter`),
- no redundant attributes stored in fact tables.

#### 5.2.1 Final Population Table

The `Population` table functions as a fact table and does not require a surrogate primary key. It stores only essential attributes and references to dimension tables:

```
CREATE TABLE Population (  
  reference_date_year DATETIME,  
  sex_code            VARCHAR(50),  
  origin_code         VARCHAR(50),  
  city_district       INT,  
  statistical_quarter  VARCHAR(255),  
  population_count    INT,  
  FOREIGN KEY (sex_code) REFERENCES Sex(sex_code),
```

```
FOREIGN KEY (origin_code) REFERENCES Origin(origin_code)
);
```

### 5.2.2 Final TrafficMeasurement Table

The TrafficMeasurement table contains more than 21 million hourly traffic observations and was therefore implemented without a surrogate primary key:

```
CREATE TABLE TrafficMeasurement (
    measurement_site_id    VARCHAR(255),
    timestamp              DATETIME,
    vehicle_count          INT,
    vehicle_count_status   VARCHAR(50),
    FOREIGN KEY (measurement_site_id) REFERENCES MeasurementSite(measurement_site_id)
);
```

Lookup and dimension tables were created first to ensure that all foreign key dependencies were satisfied before loading the fact tables.

### 5.2.3 Loading Raw Data into Staging Tables

#### CSV Accessibility Verification

Before loading data, the ability of MySQL to read CSV files inside the virtual machine environment was confirmed:

```
SELECT LOAD_FILE('C:\\Users\\labadmin\\Downloads\\population_data_cleaned_final.csv')
IS NOT NULL AS can_read;
```

#### *Bulk Loading with LOAD DATA LOCAL INFILE*

CSV data was imported into staging tables using MySQL's LOAD DATA LOCAL INFILE command. Example for the population dataset:

```
LOAD DATA LOCAL INFILE 'C:\\Users\\labadmin\\Downloads\\
population_data_cleaned_final.csv'
INTO TABLE stg_population_data
FIELDS TERMINATED BY ','
OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\\n'
IGNORE 1 LINES;
```

Similar staging tables were used for loading (see Appendices):

- stg\_traffic\_data
- stg\_quarter\_data
- stg\_population\_lookup (for lookup dimensions)

### 5.2.4 Transformation into the Production Schema

After loading the staging tables, all transformations were executed in SQL. The workflow included deduplication, type conversion, creation of lookup tables, and population of normalized production

tables.

### 1. Deduplication Using GROUP BY

Several entities—such as Quarter, CountingSite, and MeasurementSite—appeared multiple times due to repeated values in the raw datasets. Deduplication was achieved using GROUP BY combined with safe selection of consistent values:

```
SELECT street_name,  
       CAST(CAST(MIN(city_district) AS DECIMAL(10,2)) AS SIGNED),  
       MIN(statistical_quarter)  
FROM stg_quarter_data  
GROUP BY street_name;
```

### 2. Type Conversion and Cleaning

Many numeric fields in the raw CSV files appeared as strings (“1.0”, “224.0”) or contained empty values (“”). To ensure safe conversion to integers, nested casting was used:

```
CAST(CAST(NULLIF(TRIM(city_district), '') AS DECIMAL(10,2)) AS SIGNED)
```

This approach ensures correct handling of decimals and empty strings.

### 3. Construction of Lookup Tables (Sex and Origin)

A lightweight lookup staging table was reloaded to extract dimension values for Sex and Origin:

```
LOAD DATA LOCAL INFILE ...  
INTO TABLE stg_population_lookup  
(@date, sex_code, sex, origin_code, origin, @d1, @d2, @d3);
```

Deduplicated lookup tables were then created as follows:

```
INSERT INTO Sex      SELECT DISTINCT sex_code, sex      FROM stg_population_lookup;  
INSERT INTO Origin   SELECT DISTINCT origin_code, origin FROM stg_population_lookup;
```

### 4. Normalization of the Population Table

During the initial loading, descriptive columns such as sex and origin were included in the staging and production tables. After constructing the lookup tables, these redundant attributes were removed:

```
ALTER TABLE Population  
DROP COLUMN sex,  
DROP COLUMN origin;
```

Foreign keys were then added to ensure referential integrity.

## 5.2.5 Challenges and Solutions

### Challenge 1: Decimal-formatted numeric values

In the Quarter dataset, the column `city_district` contained numeric values stored as strings in decimal format (e.g., “1.0”). Direct casting of these values into integer columns caused MySQL conversion errors during the ETL process (Error Code: 1366. Incorrect DECIMAL value: ‘1.0’).

*Solution: A two-stage conversion using DECIMAL(10,2) and NULLIF cleaned and normalized the input values.*

### Challenge 2: Empty strings in numeric fields

Several numeric columns in the Population dataset (e.g., `population_count`, `city_district`) contained empty strings ('') or whitespace (' '). These produced errors when MySQL attempted to convert them into integers (Error Code: 1292. Truncated incorrect INTEGER value:'). *Solution:*

```
NULLIF(TRIM(value), '')
```

*was used to convert empty strings to NULL before casting.*

### Challenge 3: Large-scale inserts with foreign key dependencies

Issue: Bulk loading millions of records into a normalized schema requires temporary suspension of referential integrity checks to achieve acceptable performance. *Solution: Foreign key checks were intentionally disabled during insertion and reactivated after the operation, following standard best practices for ETL pipelines.*

### Challenge 4: Reconstruction of lookup data

Issue: The original staging table containing lookup information for Sex and Origin had accidentally been deleted. *Solution: A minimal stg\_population\_lookup table was recreated and reloaded to regenerate the necessary dimension tables.*

## 6 Analyzing & Evaluating Data

- place holder:

## 7 Efficiency & Query Performance

- place holder:

## 8 Visualization & Decision Support

- place holder:

## 9 Conclusions & Lessons Learned

- place holder:

## 10 Individual Team Member Reflections (required)

### 10.1 Individual Member Structure

- Member Overview:
- Technical Takeaways:
- Collaboration Insights:
- Next Steps:

## 11 Generative AI Declaration & Guidelines

Generative AI was used as a supplementary tool on top of the assisting material provided in the DBM lecture.

### 11.0.1 Guidelines for Responsible and Effective Usage of Gen-AI

1. **Human-in-the-loop verification:** All AI-suggested content was treated as a draft. Text, code, or sources delivered by AI had to be critically reviewed and tested before inclusion.
2. **Emphasis on learning:** Gen-AI functioned as a learning companion rather than an automated code generator. After receiving suggestions, team members engaged in follow-up questions to grasp the underlying logic and method.
3. **Transparency:** The team openly acknowledged where and how AI was used in the workflow so the benefits and quality safeguards remained clear.

### 11.0.2 Use Cases of AI Tools in the DBM Course Context

AI-based assistants were applied in the following specific cases: 1. **Brainstorming:** Elaborating initial ideas, structuring thoughts, and outlining coding approaches. 2. **Debugging & optimization:** Explaining error messages and helping improve self-developed scripts to raise efficiency. 3. **Proofreading:** Accelerating grammar and typo checks during documentation. 4. **Information acquisition:** Searching for methodological references or code documentation for SQL, R Markdown, or data-modeling concepts (always followed by human verification of credibility).

### 11.0.3 Benefits and Challenges in Using Generative AI Tools

1. **Benefits:** Faster debugging (e.g., resolving R Markdown knitting errors or MySQL query issues), on-demand tutoring for advanced questions, and more time spent on analytical reasoning instead of routine fixes.
2. **Challenges:** The ease of getting answers can create a temptation to trust outputs blindly. The “human-in-the-loop” guideline ensured the team retained ownership, validating every AI-assisted contribution.

## 12 How to render this report

A single R command will render the file. In an R console run: `rmarkdown::render("report.Rmd")`. Make sure your R working directory is the report folder (or use the full path). For PDF output you need a LaTeX engine (e.g. TinyTeX).

## 13 Appendix A: Traffic Data – Complete ETL Script

This appendix contains the full SQL workflow used to load, clean, transform, and normalize the traffic dataset.

The process consists of three stages:

1. **Staging** — Load raw CSV data without constraints
  2. **Schema Creation** — Create normalized production tables
  3. **Transformation** — Insert cleaned, deduplicated, type-safe data into the production schema
- 

### 13.1 Staging Table Creation

```
USE traffic_population_zh;

CREATE TABLE stg_traffic_data (
    measurement_site_id    VARCHAR(255),
    measurement_site_name  VARCHAR(255),
    counting_site_id       VARCHAR(255),
    counting_site_name     VARCHAR(255),
    axis                   VARCHAR(255),
    position_description    VARCHAR(255),
    east_coordinate        VARCHAR(255),
    north_coordinate       VARCHAR(255),
    direction              VARCHAR(255),
    signal_id              VARCHAR(255),
    signal_name            VARCHAR(255),
    num_detectors          VARCHAR(255),
    timestamp              VARCHAR(255),
    vehicle_count          VARCHAR(255),
    vehicle_count_status   VARCHAR(255)
);
```

---

### 13.2 Load Raw Traffic CSV into Staging

```
LOAD DATA LOCAL INFILE 'C:\\Users\\labadmin\\Downloads\\traffic_data_cleaned_final.csv'
INTO TABLE stg_traffic_data
FIELDS TERMINATED BY ','
OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 LINES;
```

---

## 13.3 Production Schema Creation (Normalized Tables)

### 13.3.1 TrafficSignal (Lookup Dimension)

```
CREATE TABLE TrafficSignal (  
    signal_id      INT PRIMARY KEY,  
    signal_name    VARCHAR(255)  
);
```

### 13.3.2 CountingSite (Level 1 Spatial Entity)

```
CREATE TABLE CountingSite (  
    counting_site_id    VARCHAR(50) PRIMARY KEY,  
    counting_site_name  VARCHAR(255),  
    axis                VARCHAR(255),  
    east_coordinate     FLOAT,  
    north_coordinate    FLOAT,  
    signal_id           INT,  
    FOREIGN KEY (signal_id) REFERENCES TrafficSignal(signal_id)  
);
```

### 13.3.3 MeasurementSite (Level 2 Spatial Entity)

```
CREATE TABLE MeasurementSite (  
    measurement_site_id    VARCHAR(50) PRIMARY KEY,  
    measurement_site_name  VARCHAR(255),  
    direction              VARCHAR(50),  
    position_description   VARCHAR(255),  
    num_detectors          INT,  
    counting_site_id       VARCHAR(50),  
    FOREIGN KEY (counting_site_id) REFERENCES CountingSite(counting_site_id)  
);
```

### 13.3.4 TrafficMeasurement (Fact Table with 21M+ Rows)

```
CREATE TABLE TrafficMeasurement (  
    traffic_measurement_id INT AUTO_INCREMENT PRIMARY KEY,  
    measurement_site_id    VARCHAR(50),  
    timestamp              DATETIME,  
    vehicle_count          INT,  
    vehicle_count_status   VARCHAR(50),  
    FOREIGN KEY (measurement_site_id) REFERENCES MeasurementSite(measurement_site_id)  
);
```

## 13.4 Transformation into Production Schema

Foreign key checks are disabled temporarily to allow inserting dimension rows before the referencing fact table is loaded.

```
SET FOREIGN_KEY_CHECKS = 0;
```

---

### 13.4.1 Insert Traffic Signals (Deduplicated)

```
INSERT INTO TrafficSignal (signal_id, signal_name)
SELECT
    signal_id_int,
    MIN(signal_name) AS signal_name
FROM (
    SELECT
        CAST(signal_id AS UNSIGNED) AS signal_id_int,
        signal_name
    FROM stg_traffic_data
    WHERE signal_id REGEXP '[0-9]+$'
) AS sub
GROUP BY signal_id_int;
```

---

### 13.4.2 Clean CountingSite (Deduplicate & Normalize)

Drop axis column due to inconsistent values:

```
ALTER TABLE CountingSite
DROP COLUMN axis;
```

Insert cleaned counting sites:

```
INSERT INTO CountingSite (
    counting_site_id,
    counting_site_name,
    east_coordinate,
    north_coordinate,
    signal_id
)
SELECT
    counting_site_id,
    MIN(counting_site_name) AS counting_site_name,
    MIN(CAST(east_coordinate AS FLOAT)) AS east_coordinate,
    MIN(CAST(north_coordinate AS FLOAT)) AS north_coordinate,
    MIN(CAST(signal_id AS UNSIGNED)) AS signal_id
FROM stg_traffic_data
GROUP BY counting_site_id;
```

---

### 13.4.3 Clean MeasurementSite (Deduplicate & Normalize)

```
INSERT INTO MeasurementSite (  
    measurement_site_id,  
    measurement_site_name,  
    direction,  
    position_description,  
    num_detectors,  
    counting_site_id  
)  
SELECT  
    measurement_site_id,  
    MIN(measurement_site_name) AS measurement_site_name,  
    MIN(direction) AS direction,  
    MIN(position_description) AS position_description,  
    MIN(CAST(num_detectors AS UNSIGNED)) AS num_detectors,  
    MIN(counting_site_id) AS counting_site_id  
FROM stg_traffic_data  
GROUP BY measurement_site_id;
```

---

### 13.4.4 Insert TrafficMeasurement (21M+ Records)

```
INSERT INTO TrafficMeasurement (  
    measurement_site_id,  
    timestamp,  
    vehicle_count,  
    vehicle_count_status  
)  
SELECT  
    measurement_site_id,  
    STR_TO_DATE(timestamp, '%Y-%m-%dT%H:%i:%s'),  
    ROUND(vehicle_count),  
    vehicle_count_status  
FROM stg_traffic_data;
```

---

## 13.5 Final Cleanup

```
SET FOREIGN_KEY_CHECKS = 1;  
  
DROP TABLE stg_traffic_data;
```

## 14 Appendix B: Quarter Data – Complete ETL Script

This appendix contains the complete SQL workflow used to load, clean, deduplicate and transform the quarter-level geographic dataset.

The process consists of:

1. **Staging** — Load the raw CSV without constraints
  2. **Schema Creation** — Create the normalized `Quarter` table
  3. **Transformation** — Deduplicate and convert types
  4. **Cleanup** — Remove staging data
- 

### 14.1 Staging Table Creation

```
USE traffic_population_zh;

CREATE TABLE stg_quarter_data (
  address          VARCHAR(255),
  house_number     VARCHAR(255),
  street_name      VARCHAR(255),
  city_district    VARCHAR(255),
  statistical_quarter VARCHAR(255)
);
```

---

### 14.2 Load Raw Quarter CSV into Staging

```
LOAD DATA LOCAL INFILE 'C:\\Users\\labadmin\\Downloads\\quarter_data_cleaned_final.csv'
INTO TABLE stg_quarter_data
FIELDS TERMINATED BY ','
OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\\n'
IGNORE 1 LINES;
```

---

### 14.3 Create Normalized Quarter Table

```
USE traffic_population_zh;

CREATE TABLE Quarter (
  street_name      VARCHAR(255) PRIMARY KEY,
  city_district    INT,
```

```
    statistical_quarter VARCHAR(255)
);
```

---

## 14.4 Transformation: Deduplication & Type Conversion

A single consistent row is required per `street_name`.

Several values in the raw CSV appear as decimals (e.g., "1.0"), strings, or blanks ("").

Nested `CAST()` and `NULLIF()` ensure reliable type handling.

```
INSERT INTO Quarter (
    street_name,
    city_district,
    statistical_quarter
)
SELECT
    street_name,

    -- Convert empty strings to NULL, then cast numeric strings like "1.0" to INT
    CAST(
        CAST(
            NULLIF(TRIM(MIN(city_district)), '') AS DECIMAL(10,2)
        ) AS SIGNED
    ) AS city_district,

    -- statistical_quarter is a string → take MIN() as deterministic representative
    MIN(statistical_quarter)
FROM stg_quarter_data
GROUP BY street_name;
```

---

## 14.5 Cleanup Staging Table

```
DROP TABLE stg_quarter_data;
```

# 15 Appendix C: Population Data – Complete ETL Script

This appendix documents the full SQL workflow for loading and transforming the population dataset. The ETL process follows the standard structure used throughout the project:

1. **Staging** — Load raw CSV values as untyped strings
2. **Schema Creation** — Define the normalized production table
3. **Transformation** — Clean, cast and filter rows (2012 onward)

#### 4. Cleanup — Remove staging data

---

### 15.1 Staging Table Creation

```
USE traffic_population_zh;

CREATE TABLE stg_population_data (
    reference_date_year    VARCHAR(50),
    sex_code               VARCHAR(50),
    sex                   VARCHAR(50),
    origin_code            VARCHAR(50),
    origin                 VARCHAR(255),
    city_district          VARCHAR(50),
    statistical_quarter    VARCHAR(255),
    population_count       VARCHAR(50)
);
```

---

### 15.2 Load Raw Population CSV into Staging

```
LOAD DATA LOCAL INFILE 'C:\\Users\\labadmin\\Downloads\\population_data_cleaned_final.csv'
INTO TABLE stg_population_data
FIELDS TERMINATED BY ','
OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 LINES;
```

---

### 15.3 Create Normalized Population Table

```
USE traffic_population_zh;

CREATE TABLE Population (
    reference_date_year    DATETIME,
    sex_code               VARCHAR(50),
    sex                   VARCHAR(50),
    origin_code            VARCHAR(50),
    origin                 VARCHAR(255),
    city_district          INT,
    statistical_quarter    VARCHAR(255),
    population_count       INT
);
```

---

## 15.4 Transformation: Cleaning, Casting, Filtering

The dataset contains mixed formats such as "1.0", "0", " " or "224.0". Nested casts ensure consistent numeric values:

- `NULLIF(TRIM(x), '')` converts empty strings to `NULL`
- `DECIMAL(10,2)` handles values like "1.0"
- `SIGNED` converts final value to an integer

Additionally, only rows **from 2012 onward** are included.

```
INSERT INTO Population (
    reference_date_year,
    sex_code,
    sex,
    origin_code,
    origin,
    city_district,
    statistical_quarter,
    population_count
)
SELECT
    STR_TO_DATE(reference_date_year, '%Y-%m-%d'),
    sex_code,
    sex,
    origin_code,
    origin,
    CAST(CAST(NULLIF(TRIM(city_district), '') AS DECIMAL(10,2)) AS SIGNED),
    statistical_quarter,
    CAST(CAST(population_count AS DECIMAL(10,2)) AS SIGNED)
FROM stg_population_data
WHERE STR_TO_DATE(reference_date_year, '%Y-%m-%d') >= '2012-01-01';
```

---

## 15.5 Cleanup Staging Table

```
DROP TABLE stg_population_data;
```

## 16 Appendix D: Lookup Tables – Complete ETL Script

This appendix documents the workflow for extracting clean lookup values for `Sex` and `Origin` from the population dataset.

The ETL process consists of:

1. **Staging** — Load untyped lookup fields

2. **Schema Creation** — Create normalized lookup tables
  3. **Transformation** — Deduplicate and insert unique values
  4. **Integration** — Attach foreign keys to the Population table
  5. **Cleanup** — Drop staging table
- 

## 16.1 Staging Table Creation

```
CREATE TABLE stg_population_lookup (  
    sex_code      VARCHAR(50),  
    sex           VARCHAR(50),  
    origin_code   VARCHAR(50),  
    origin        VARCHAR(255)  
);
```

---

## 16.2 Load Lookup Fields from Raw CSV

Only the lookup-related columns are imported.  
All other columns are ignored via dummy variables.

```
LOAD DATA LOCAL INFILE 'C:\\Users\\labadmin\\Downloads\\population_data_cleaned_final.csv'  
INTO TABLE stg_population_lookup  
FIELDS TERMINATED BY ','  
OPTIONALLY ENCLOSED BY '"'  
LINES TERMINATED BY '\\n'  
IGNORE 1 LINES  
(  
    @dummy_reference_date_year,  
    sex_code,  
    sex,  
    origin_code,  
    origin,  
    @dummy_city_district,  
    @dummy_statistical_quarter,  
    @dummy_population_count  
);
```

---

## 16.3 Create Normalized Lookup Tables

```
CREATE TABLE Sex (  
    sex_code      VARCHAR(50) PRIMARY KEY,
```

```

        sex          VARCHAR(50)
    );

CREATE TABLE Origin (
    origin_code    VARCHAR(50) PRIMARY KEY,
    origin         VARCHAR(255)
);

```

---

## 16.4 Populate Lookup Tables (Deduplicated)

### 16.4.1 Insert Sex Codes

```

INSERT INTO Sex (sex_code, sex)
SELECT DISTINCT
    sex_code,
    sex
FROM stg_population_lookup
WHERE sex_code IS NOT NULL AND sex_code != '';

```

### 16.4.2 Insert Origin Codes

```

INSERT INTO Origin (origin_code, origin)
SELECT DISTINCT
    origin_code,
    origin
FROM stg_population_lookup
WHERE origin_code IS NOT NULL AND origin_code != '';

```

---

## 16.5 Normalize Population Table by Removing Redundant Columns

The descriptive values `sex` and `origin` are removed from the `Population` fact table. Only the codes remain, backed by lookup tables.

```

ALTER TABLE Population
DROP COLUMN sex,
DROP COLUMN origin;

```

---

## 16.6 Add Foreign Key Constraints to Population

```

ALTER TABLE Population
ADD CONSTRAINT fk_population_sex
    FOREIGN KEY (sex_code)
    REFERENCES Sex(sex_code);

```

```
ALTER TABLE Population
ADD CONSTRAINT fk_population_origin
    FOREIGN KEY (origin_code)
    REFERENCES Origin(origin_code);
```

---

## 16.7 Cleanup Staging Table

```
DROP TABLE stg_population_lookup;
```