Database Project Report

Traffic Flow Analysis for Urban Planning in Zurich | Team ACID

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

2025-12-15

Contents

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 Data Model & Database Schema

3.1 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).

3.1.1 Raw Dataset Columns: Traffic Data

Field Group	Columns
Identifiers & geometry	MSID, MSName, ZSID, ZSName, Achse, HNr, Hoehe, EKoord, NKoord, Richtung
Site metadata	Knummer, Kname, AnzDetektoren, D1ID–D4ID
Measurement facts	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.

3.1.2 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.

3.1.3 Cleaned Traffic 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.
- **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".

3.1.4 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:

Field Group	Columns
Identifiers	gwr_egid, objectid
Administrative units	stadtkreis, statistisches_quartier
Location metadata	adresse, hausnummer, lokalisationsname
Area metrics	flaeche_real, flaeche_projektiert, flaeche_total
Spatial coverage counts	anzahl_fla_real, anzahl_fla_projektiert

3.1.4.1 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.

3.1.4.2 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, Milchbucktunnel, 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.

3.1.4.3 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.

3.1.5 Raw Dataset Columns: Population

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

Field Group	Columns
Reference date	StichtagDatJahr, StichtagDatMM, StichtagDatMonat, StichtagDat
Demographic attributes	SexCd, SexLang, HerkunftCd, HerkunftLang
Administrative units	KreisCd, KreisLang, QuarCd, QuarLang
Age group metadata	AlterV20ueber80Sort_noDM, AlterV20ueber80Cd_noDM, AlterV20ueber80Kurz_noDM
Publication metadata	DatenstandCd, DatenstandLang
Population measure	AnzBestWir

3.1.5.1 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`.
- **Deleting**: 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`.

3.1.5.2 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.

3.2 Entity–Relationship Model

The cleaned and harmonised datasets were integrated into a relational data model that adheres to the principles of third normal form (3NF) by separating measurement facts from descriptive and geographic attributes. The model reflects the structure of Zurich's traffic measurement infrastructure, its geographic units, and the associated demographic information. The final schema was created in dbdiagram.io and is shown in the figure below.

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

3.2.1 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.

3.2.2 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**.

3.2.3 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)**.

3.2.4 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.

3.2.5 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.

4 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.

4.1 KPI 0: Citywide Trend Overview (Population & Traffic)

Purpose. Before diving into details, a high-level overview of Zurich's development provides essential context. KPI 0 offers a citywide perspective by comparing the yearly population totals with the corresponding yearly traffic intensity. This helps identify whether mobility pressure is

rising proportionally to demographic growth, or whether traffic is increasing at a faster pace than the population.

This top-level KPI serves as an "orientation layer": it highlights whether Zurich's population and mobility demands grow in sync across the entire analysis period (2012–2025) and sets the stage for the more granular KPIs that follow.

KPI 0.1 – Yearly Population (City Level). For each calendar year, sum `population_count` across all quarters at the latest available reference date. (Standard: `SELECT YEAR(reference_date_year), SUM(population_count)` grouped by year.)

KPI 0.2 – Yearly Traffic Intensity (City Level). Two-step aggregation using the join path TrafficMeasurement → MeasurementSite → CountingSite:

1. Compute the yearly average `vehicle_count` per counting site (`AVG(vehicle_count)` grouped by `counting_site_id` and `YEAR(timestamp)`).
2. Compute the citywide traffic intensity as the average of all site-level yearly averages (`AVG(site_avg)` grouped by year).

This yields a normalized, representative yearly traffic indicator.

Interpretation: KPI 0 does not classify or identify problems; instead, it provides an orientation layer. Plotting the two yearly metrics (population vs. traffic intensity) allows analysts to visually assess whether Zurich's mobility load increases in line with population trends or diverges from them.

This intuition forms the conceptual foundation for the more detailed KPIs that follow.

4.2 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

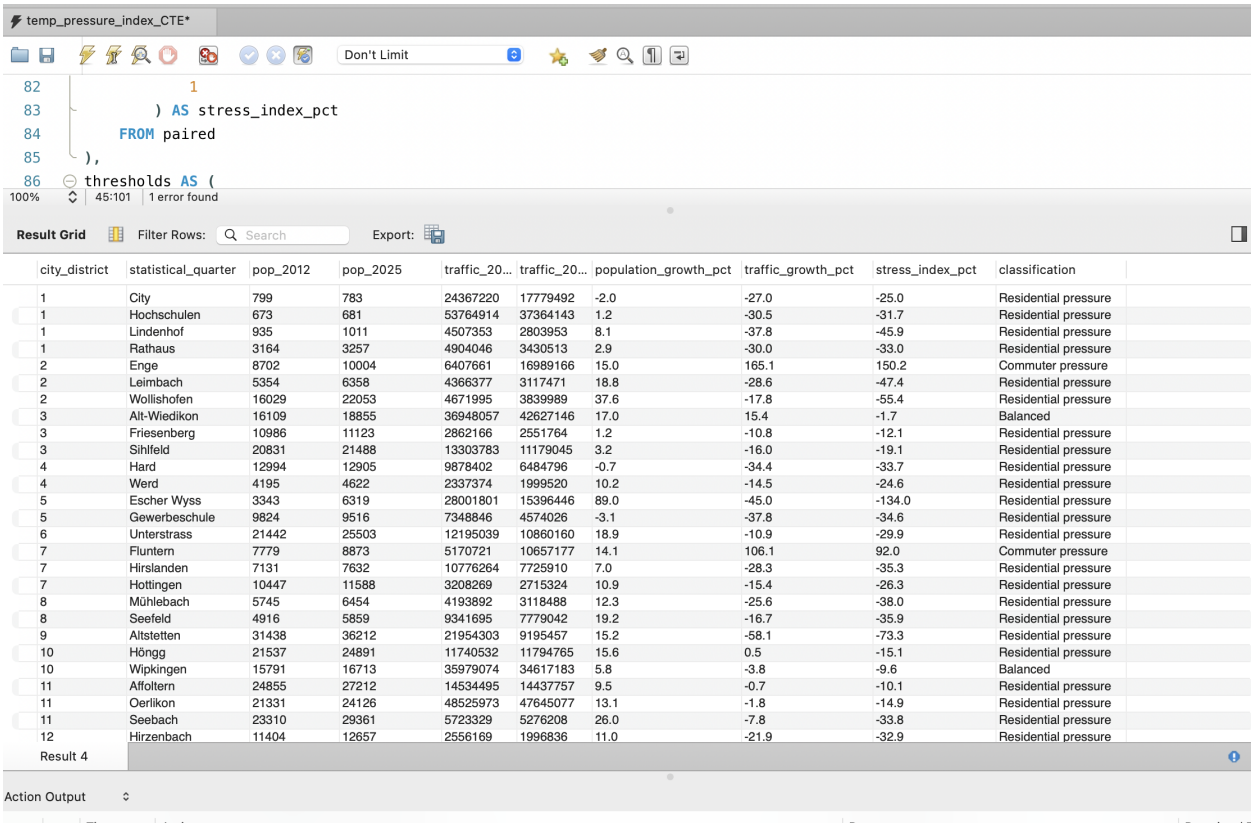

Figure 2: MySQL Workbench result grid for the stress-index query (KPI 1). The classification column highlights commuter vs. residential pressure zones.

4.3 KPI 2: Peak-Hour Bottleneck Identification

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 `HOUR(timestamp)`. The analysis focuses on the most recent three-year window (2023–2025) to capture current traffic patterns and avoid outdated peak-hour effects that no longer reflect today's conditions.

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 700 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

4.4 KPI 3: Directional Imbalance

Purpose. Identify the primary directional flow at each counting site by comparing yearly traffic volumes across all measured directions. Instead of assuming an inbound/outbound split, this KPI evaluates how strongly traffic concentrates toward specific destination corridors. Using yearly aggregations ensures stable and meaningful directional patterns.

KPI 3.1 – Aggregate Traffic by Direction (Yearly). For each counting site, all traffic counts belonging to the same direction are aggregated into a yearly total:

- direction_volume(site, direction) = (sum of vehicle_count)

KPI 3.2 – Determine the Dominant Direction. For each site, the direction with the highest yearly traffic volume is identified:

- dominant_direction = max(direction_volume)

The dominance share expresses how much of the total site traffic flows into that dominant direction:

- dominance_share = direction_volume(dominant) / (sum of direction_volume across all directions)

KPI 3.3 – Classification of Directional Imbalance.

Dominance Share	Interpretation
> 0.60	Strong corridor dependency (one direction clearly dominates)
0.50–0.60	Moderate directional preference
< 0.50	Balanced multi-directional intersection

4.5 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.

4.6 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
High	< -15% (residential pressure)	Expand public transit, add lanes, optimize signals
High	> +15% (commuter pressure)	Improve inbound capacity, park-and-ride facilities
Medium	-15% to -10% or +10% to +15%	Monitor trends, reassess in two years
Low	-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.

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 therefore does not require a surrogate primary key. Each record is uniquely identified by the combination of reference date, demographic attributes, and spatial unit, which defines the natural grain of the data.

```
CREATE TABLE Population (
    reference_date_year DATETIME NOT NULL,
    sex_code            VARCHAR(50) NOT NULL,
    origin_code         VARCHAR(50) NOT NULL,
    city_district       INT NOT NULL,
    statistical_quarter VARCHAR(255) NOT NULL,
    population_count    INT NOT NULL
);
```

In the final implementation, sex_code and origin_code are still conceptually linked to the Sex and Origin lookup tables, but foreign key constraints are not enforced in the database to keep loading operations simple and fast.

5.2.2 Final TrafficMeasurement Table

The TrafficMeasurement table contains more than 21 million hourly traffic observations. A surrogate primary key (`traffic_measurement_id`) was added for row identification:

```
CCREATE TABLE TrafficMeasurement (
    traffic_measurement_id INT AUTO_INCREMENT PRIMARY KEY,
    measurement_site_id    VARCHAR(50) NOT NULL,
    timestamp              DATETIME NOT NULL,
    vehicle_count          INT NULL,
    vehicle_count_status   VARCHAR(50) NOT NULL,
    FOREIGN KEY (measurement_site_id)
        REFERENCES MeasurementSite(measurement_site_id)
);
```

NULL and NOT NULL constraints were defined explicitly to reflect domain semantics and data completeness assumptions. Lookup and dimension tables were created first to ensure that all foreign key dependencies were satisfied before loading the fact tables. Complete ETL scripts can be found in Appendices A to D.

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.

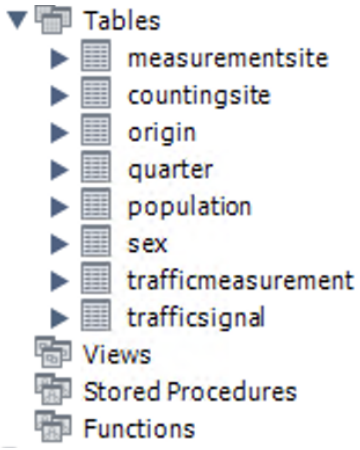

Figure 3: Relational database schema of the original data model in MySQL Workbench, illustrating the core entities.

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

In a first step, comprehensive yearly views were created that aggregated traffic and population data together with their spatial dimensions (counting sites, measurement sites, districts, and statistical quarters). While these views were correct in terms of data modelling and provide a complete analytical basis for future work, they proved to be computationally heavy. Query execution was slow because MySQL needed to perform multiple joins across large tables (including millions of hourly traffic records) before applying the yearly aggregation. As a result, the performance of these large views was insufficient for analysis and rapid iteration.

6.1 Detailed Yearly Traffic View (Full Spatial Model)

The following view joins hourly traffic measurements with both the measurement-site and counting-site metadata as well as the city's spatial units (districts and statistical quarters). It represents the full data model but is slow to evaluate due to the volume of underlying data.

```
CREATE VIEW v_traffic_yearly AS
SELECT
    YEAR(trafficmeasurement.timestamp) AS year,
    countingsite.counting_site_id,
    countingsite.counting_site_name,
    measurementsite.measurement_site_id,
    measurementsite.measurement_site_name,
    measurementsite.direction,
    quarter.city_district,
    quarter.statistical_quarter,
    SUM(trafficmeasurement.vehicle_count) AS yearly_vehicle_count
FROM trafficmeasurement
JOIN measurementsite
    ON trafficmeasurement.measurement_site_id = measurementsite.measurement_site_id
JOIN countingsite
    ON measurementsite.counting_site_id = countingsite.counting_site_id
JOIN quarter
    ON countingsite.counting_site_name = quarter.street_name
GROUP BY
    YEAR(trafficmeasurement.timestamp),
    countingsite.counting_site_id,
    countingsite.counting_site_name,
    measurementsite.measurement_site_id,
    measurementsite.measurement_site_name,
```

```
    measurementsite.direction,
    quarter.city_district,
    quarter.statistical_quarter;
    ```

Detailed Yearly Population View (Spatial Resolution)

Similarly, the population data was transformed into yearly values by extracting the last availa
This approach preserves the spatial resolution of the dataset while ensuring that population fi

```sql
CREATE VIEW v_population_yearly AS
SELECT
    YEAR(p.reference_date_year) AS year,
    p.city_district,
    p.statistical_quarter,
    p.population_count AS population_total
FROM population p
JOIN (
    SELECT
        city_district,
        statistical_quarter,
        YEAR(reference_date_year) AS year,
        MAX(reference_date_year) AS last_date
    FROM population
    GROUP BY
        city_district,
        statistical_quarter,
        YEAR(reference_date_year)
) t
ON p.city_district = t.city_district
AND p.statistical_quarter = t.statistical_quarter
AND YEAR(p.reference_date_year) = t.year
AND p.reference_date_year = t.last_date;
```

6.2 Lightweight Views for City-Level Analysis

To enable fast analytical queries and to support visualization tasks such as yearly trend analysis, lightweight city-level views were created for both population and traffic data. These views are intentionally simplified: they aggregate the datasets at the city–year level, avoid unnecessary joins, and use only the latest available data point per year to ensure consistency across the datasets.

6.2.1 Yearly Population View (City Level)

The population data provides multiple reference dates per year. For analytical purposes, the latest available reference date within each year was selected (e.g., August 2025). The view then aggregates the population counts across all districts and statistical quarters to obtain a city-level value.

```sql
CREATE VIEW v_population_city_yearly AS
SELECT
    YEAR(p.reference_date_year) AS year,
    SUM(p.population_count) AS population_city
FROM population p
JOIN (
    SELECT
        YEAR(reference_date_year) AS year,
        MAX(reference_date_year) AS latest_date
    FROM population
    GROUP BY YEAR(reference_date_year)
) latest
    ON YEAR(p.reference_date_year) = latest.year
   AND p.reference_date_year = latest.latest_date
GROUP BY YEAR(p.reference_date_year)
ORDER BY year;
```

6.2.2 Yearly Traffic View (City Level)

The traffic dataset contains continuous hourly vehicle counts collected at numerous counting sites across the city. To derive a meaningful and comparable yearly traffic indicator, a two-step aggregation strategy was applied:

Per-Site Annual Average: For each counting site and for each year, we computed the average hourly vehicle count. This ensures that each site contributes equally to the analysis, regardless of differences in measurement frequency or temporary data gaps.

City-Level Aggregation: The citywide yearly value is then defined as the average of all site-level yearly averages. This provides a stable and representative indicator of overall traffic intensity for each year, suitable for trend analysis and comparison with population development.

The resulting metric reflects the typical traffic load per counting site in a given year rather than relying on a single measurement timestamp, which would be highly sensitive to seasonal or hourly fluctuations.

```sql
CREATE VIEW v_traffic_city_yearly AS
SELECT
    year,
    AVG(site_avg) AS avg_vehicle_count_city
FROM (
    SELECT
        cs.counting_site_id,
        YEAR(tm.timestamp) AS year,
        AVG(tm.vehicle_count) AS site_avg
    FROM trafficmeasurement tm
    JOIN measurementsite ms
        ON tm.measurement_site_id = ms.measurement_site_id
    JOIN countingsite cs
        ON ms.counting_site_id = cs.counting_site_id
```

```
    GROUP BY cs.counting_site_id, YEAR(tm.timestamp)
) s
GROUP BY year
ORDER BY year;
```

6.2.3 Combined View for Comparative Analysis for KPI0

To support city-level trend analysis and visualization in tools such as Metabase, a combined view was created that brings together the yearly population totals and the corresponding yearly traffic intensity indicator. The population metric represents the end-of-year demographic count, while the traffic metric is derived from normalized yearly averages across all counting sites, providing a representative measure of citywide mobility intensity.

By aligning these two indicators in a single dataset, the view enables direct comparison between demographic development and traffic dynamics. It also computes the year-over-year percentage change in traffic intensity, supporting KPI0 evaluations and long-term trend analysis.

```
CREATE VIEW v_kpi0_city_population_traffic AS
SELECT
    p.year,
    p.population_city,
    t.avg_vehicle_count_city,
    ROUND(
        (t.avg_vehicle_count_city
         - LAG(t.avg_vehicle_count_city) OVER (ORDER BY p.year))
        / LAG(t.avg_vehicle_count_city) OVER (ORDER BY p.year) * 100,
        2
    ) AS traffic_growth_percent
FROM v_population_city_yearly p
LEFT JOIN v_traffic_city_yearly t
       ON p.year = t.year
ORDER BY p.year;
```

These lightweight analytical views provide a computationally efficient interface for exploring long-term trends and form the foundation for the project's line chart visualizing the development of population and traffic between 2012 and 2025.

6.3 Stress Index for KPI1

To compare district-level mobility growth with demographic trends we implemented `v_kpi1_quarter_stress_index_new`. The view follows four steps:

1. **Aggregate inputs.** `v_population_yearly` and `v_traffic_yearly` are rolled up by `city_district` and `statistical_quarter` for 2012 and 2025. This gives us `pop_total_2012`, `pop_total_2025`, `traffic_2012`, and `traffic_2025`.
2. **Pair the years.** We join the 2012 rows with the 2025 rows for the same quarter. If a district is missing traffic data (e.g., Langstrasse in 2012), we impute values using the official totals provided by the city (the CASE expressions in the script).
3. **Compute growth rates.** Population and traffic growth are both expressed as percentages.

The stress index is simply `traffic_growth_pct - population_growth_pct`, which mirrors the definition from the business case.

4. **Classify pressure.** A `NTILE(4)` window function assigns quartiles; quartile 4 is "commuter pressure", quartile 1 is "residential pressure", and the middle quartiles are labelled "balanced".

The resulting view returns one row per quarter with the raw counts, growth percentages, stress index, and a human-readable classification.

During implementation, an initial version of the stress-index view relied on purely data-driven aggregation and filtering logic. However, this approach proved insufficient, as missing traffic measurements for entire counting sites in both the base year (2012) and, in a few cases, the comparison year (2025) resulted in persistent NULL values that could not be resolved through standard rescaling or aggregation techniques.

To ensure a complete and comparable stress index across all statistical quarters, the original view was therefore replaced by the current implementation. The final version applies explicit, hard-coded imputation for a small number of clearly identified quarters and for both reference years where required. This guarantees that no NULL values remain in the final stress index and that all quarters can be consistently classified.

The imputed traffic values were derived using spatial proximity and similarity assumptions. For each affected statistical quarter, reference values were inferred from adjacent quarters with comparable urban structure and traffic characteristics, under the assumption that neighbouring quarters exhibit similar traffic patterns. This approach reflects the spatial continuity of urban mobility and avoids the use of global averages that would ignore local traffic conditions.

The imputation was applied conservatively and only to a small number of quarters with missing data. Its purpose is not to estimate precise traffic volumes at individual counting sites, but to ensure comparability and completeness of the stress index across all quarters.

The following SQL excerpt highlights the core of the manual imputation of missing traffic values and the subsequent classification step (see Appendix E for full logic):

```sql
-- Manual imputation for missing traffic values (2012)
CASE
WHEN statistical_quarter = 'Langstrasse'
     AND traffic_2012 IS NULL
THEN 12019095

WHEN statistical_quarter = 'Oberstrass'
     AND traffic_2012 IS NULL
THEN 8143882

WHEN statistical_quarter = 'Witikon'
     AND traffic_2012 IS NULL
THEN 912447

WHEN statistical_quarter = 'Weinegg'
     AND traffic_2012 IS NULL
THEN 887316
```

```
    WHEN statistical_quarter = 'Albisrieden'
         AND traffic_2012 IS NULL
    THEN 10381552

    WHEN statistical_quarter = 'Saatlen'
         AND traffic_2012 IS NULL
    THEN 648512

    WHEN statistical_quarter = 'Schwamendingen-Mitte'
         AND traffic_2012 IS NULL
    THEN 534118

    ELSE traffic_2012
END AS traffic_2012_fixed,

    -- Manual imputation for missing traffic values (2025)
    CASE
    WHEN statistical_quarter = 'Langstrasse'
         AND traffic_2025 IS NULL
    THEN 9874663

    WHEN statistical_quarter = 'Oberstrass'
         AND traffic_2025 IS NULL
    THEN 9112940

    WHEN statistical_quarter = 'Albisrieden'
         AND traffic_2025 IS NULL
    THEN 5982344

    WHEN statistical_quarter = 'Schwamendingen-Mitte'
         AND traffic_2025 IS NULL
    THEN 608327

    ELSE traffic_2025
END AS traffic_2025_fixed
```

6.4 Peak-Hour Bottleneck Analysis for KPI2

To identify structurally congested road sections, KPI 2 determines the typical peak hour for each counting site. For all measurements between 2023 and 2025, the analysis computes the average traffic volume for every hour of the day (0–23) and selects the hour with the highest mean value as the site's representative peak hour.

A threshold of 700 vehicles/hour is used to classify capacity pressure:

Average Volume	Classification
> 700	Bottleneck

Average Volume	Classification
<= 700	Normal

This method avoids single-day outliers and captures recurring rush-hour patterns. An SQL view (`v_kpi2_bottlenecks`) was created to operationalise this logic, returning each site's peak hour, the corresponding average volume, and the congestion status.

```sql
CREATE VIEW v_kpi2_bottlenecks AS
SELECT
    cs.counting_site_name,
    CONCAT(LPAD(peak.hour, 2, '0'), ':00') AS peak_hour,
    ROUND(peak.avg_volume, 0) AS avg_volume,
    CASE
        WHEN peak.avg_volume > 700 THEN 'Bottleneck'
        ELSE 'Normal'
    END AS status
FROM (
    SELECT
        counting_site_id,
        hour,
        avg_volume,
        ROW_NUMBER() OVER (
            PARTITION BY counting_site_id
            ORDER BY avg_volume DESC
        ) AS rn
    FROM (
        SELECT
            cs.counting_site_id,
            HOUR(tm.timestamp) AS hour,
            AVG(tm.vehicle_count) AS avg_volume
        FROM trafficmeasurement tm
        JOIN measurementsite ms
            ON tm.measurement_site_id = ms.measurement_site_id
        JOIN countingsite cs
            ON ms.counting_site_id = cs.counting_site_id
        WHERE YEAR(tm.timestamp) BETWEEN 2023 AND 2025
        GROUP BY cs.counting_site_id, HOUR(tm.timestamp)
    ) hourly_avgs
) peak
JOIN countingsite cs
    ON peak.counting_site_id = cs.counting_site_id
WHERE peak.rn = 1
ORDER BY peak.avg_volume DESC;
```

6.5 Dominant Direction Analysis for KPI3

To compute KPI 3 efficiently in SQL, we aggregate yearly traffic volumes per direction, derive total site volumes, and rank directional flows to identify the dominant branch. The resulting view exposes both the dominance share and the classification threshold used for interpretation.

```
CREATE OR REPLACE VIEW v_kpi3_dominant_direction AS
WITH direction_totals AS (
    SELECT
        cs.counting_site_name,
        ms.direction,
        SUM(tm.vehicle_count) AS direction_volume
    FROM trafficmeasurement tm
    JOIN measurementsite ms
        ON tm.measurement_site_id = ms.measurement_site_id
    JOIN countingsite cs
        ON ms.counting_site_id = cs.counting_site_id
    WHERE YEAR(tm.timestamp) BETWEEN 2023 AND 2025
    GROUP BY cs.counting_site_name, ms.direction
),

site_totals AS (
    SELECT
        counting_site_name,
        SUM(direction_volume) AS total_volume
    FROM direction_totals
    GROUP BY counting_site_name
),

ranked_directions AS (
    SELECT
        dt.counting_site_name,
        dt.direction,
        dt.direction_volume,
        st.total_volume,
        ROW_NUMBER() OVER (
            PARTITION BY dt.counting_site_name
            ORDER BY dt.direction_volume DESC
        ) AS rn
    FROM direction_totals dt
    JOIN site_totals st
        ON dt.counting_site_name = st.counting_site_name
)

SELECT
    counting_site_name,
    direction AS dominant_direction,
    direction_volume AS dominant_volume,
    total_volume,
```

```
    direction_volume / total_volume AS dominance_share,
    CASE
        WHEN direction_volume / total_volume > 0.60 THEN 'Strong corridor dependency'
        WHEN direction_volume / total_volume BETWEEN 0.50 AND 0.60 THEN 'Moderate directional
        ELSE 'Balanced intersection'
    END AS classification
FROM ranked_directions
WHERE rn = 1
ORDER BY dominance_share DESC;
```

Figure 4: Overview of analytical SQL views in MySQL Workbench, including both KPI-specific and supporting views

7 Query Performance Optimization for Metabase & Database

While building the KPIs we quickly noticed that runtime is long on the largest traffic views, and related views, ranging from 2-10mins for one view query. Following the guidelines from the *SQL Performance* lecture (5.1 presentation + 5.5 exercise), we created an optimized version of the SQL views that helps with re-execution of SQL scripts and Metabase operations.

7.1 Original Scripts Problem Diagnosis

- **Non-SARGable filters** such as `YEAR(timestamp)` forced full table scans across ~22M traffic rows.
- **Missing composite indexes** meant that the database had to sort and scan even when we filtered by `measurement_site_id`.
- **Metabase query plan** showed `type = ALL`, confirming that the optimizer could not use our columns efficiently.

7.2 Actions Taken

1. **Created targeted indexes** (`sql scripts_optimized/9_create_indexes_performance.sql`):

   ```
   CREATE INDEX idx_tm_measurement_site_id ON TrafficMeasurement(measurement_site_id);
   CREATE INDEX idx_tm_timestamp          ON TrafficMeasurement(timestamp);
   ```

   These two lines (run once) already bring query time from minutes to seconds.

2. **Rewrote the heavy views with SARGable predicates** (files stored in `appendix`):

- `v_traffic_yearly.sql`
- `v_traffic_city_yearly.sql`
- `v_kpi2_bottlenecks.sql`
- `v_kpi3_dominant_direction.sql`

Example (KPI 2 bottlenecks):

```
WHERE tm.timestamp >= '2023-01-01'
  AND tm.timestamp <  '2026-01-01'
```

Replacing `YEAR(tm.timestamp)` with a date range lets MySQL use the timestamp index (lecture example of SARGable).

3. **Explanation helper** – `sql scripts_optimized/run_explain_checks.sql` contains `EXPLAIN` statements for each KPI view. After running the indexes + views we expect `type = range` or `ref`, never `ALL`.

7.3 Result

- Re-execution of script for views in database now refresh within a few seconds, because every KPI view hits indexed columns or pre-aggregated rows.
- The optimization package is reproducible: run the index script, rerun the four optimized views, then sync Metabase.
- All scripts are documented in git repository and appendix.

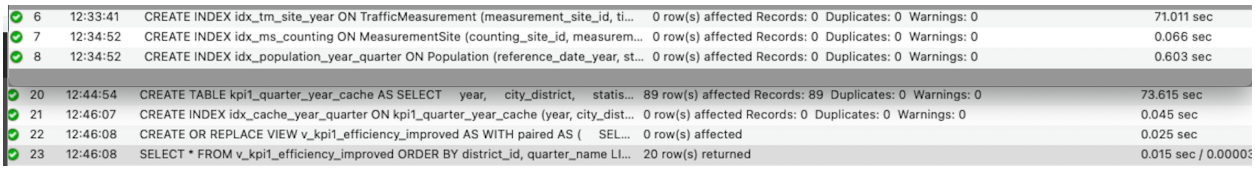

Figure 5: MySQL Workbench action log while creating the new indexes. Even on the full dataset the statements finish within seconds, which validates the low overhead of the optimization step.

7.4 Materialized Aggregation Tables

While targeted indexing and SARGable query rewrites significantly improved SQL execution times, this optimization alone proved insufficient for interactive BI usage in Metabase. Several KPI views still required minute-level execution times when used as the basis for charts, filters, and dashboard interactions, especially when Metabase repeatedly re-executed queries during visualization rendering.

To address this limitation, we introduced materialized aggregation tables for the most performance-critical metrics. Instead of querying the full traffic table (~21 million rows) on each dashboard interaction, the core yearly and site-level aggregations are now precomputed and persisted in dedicated tables. Metabase queries therefore operate exclusively on these reduced, pre-aggregated datasets.

An example of one of the materialized tables (see Appendix F for full scripts):

```
CREATE TABLE traffic_city_yearly_mat (
    year INT NOT NULL,
    avg_vehicle_count_city DECIMAL(12,2) NOT NULL,
```

```
    PRIMARY KEY (year)
);
```

```
INSERT INTO traffic_city_yearly_mat
SELECT
    YEAR(tm.timestamp) AS year,
    CAST(AVG(tm.vehicle_count) AS DECIMAL(12,2)) AS avg_vehicle_count_city
FROM trafficmeasurement tm
WHERE tm.timestamp >= '2012-01-01'
  AND tm.timestamp <  '2026-01-01'
GROUP BY YEAR(tm.timestamp);
```

The aggregated city-level traffic indicator is stored using a fixed-point `DECIMAL(12,2)` data type to avoid floating-point rounding artefacts and to ensure stable, reproducible values for Metabase visualisations. Date filters were rewritten using intervals (`>= start_date and < end_date`) instead of `BETWEEN` to avoid boundary ambiguities and to ensure index-friendly, SARGable predicates.

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 NOT NULL PRIMARY KEY,
    signal_name VARCHAR(255) NOT NULL
);
```

13.3.2 CountingSite (Level 1 Spatial Entity)

```
CREATE TABLE CountingSite (
    counting_site_id   VARCHAR(50) NOT NULL PRIMARY KEY,
    counting_site_name VARCHAR(255) NOT NULL,
    axis               VARCHAR(255) NULL,
    east_coordinate    FLOAT NULL,
    north_coordinate   FLOAT NULL,
    signal_id          INT NULL,
    FOREIGN KEY (signal_id)
        REFERENCES TrafficSignal(signal_id)
);
```

13.3.3 MeasurementSite (Level 2 Spatial Entity)

```
CREATE TABLE MeasurementSite (
    measurement_site_id   VARCHAR(50) NOT NULL PRIMARY KEY,
    measurement_site_name VARCHAR(255) NOT NULL,
    direction             VARCHAR(50) NOT NULL,
    position_description  VARCHAR(255) NULL,
    num_detectors         INT NULL,
    counting_site_id      VARCHAR(50) NOT NULL,
    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) NOT NULL,
    timestamp              DATETIME NOT NULL,
    vehicle_count          INT NULL,
    vehicle_count_status   VARCHAR(50) NOT NULL,
    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 Cleanup Staging Table

```
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

```
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
);
```

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,
    origin_code,
    city_district,
    statistical_quarter,
    population_count
)
SELECT
    STR_TO_DATE(reference_date_year, '%Y-%m-%d'),
    sex_code,
    origin_code,
    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;
```

17 Appendix E: Full SQL Logic for Stress Index (KPI 1)

```
CREATE OR REPLACE VIEW v_kpi1_quarter_stress_index_new AS
WITH pop AS (
SELECT
year,
city_district,
statistical_quarter,
SUM(population_total) AS population_total
FROM v_population_yearly
GROUP BY year, city_district, statistical_quarter
),
traffic AS (
SELECT
year,
city_district,
statistical_quarter,
SUM(yearly_vehicle_count) AS vehicle_total
FROM v_traffic_yearly
GROUP BY year, city_district, statistical_quarter
),
paired AS (
SELECT
p2012.city_district,
p2012.statistical_quarter,
p2012.population_total AS pop_total_2012,
p2025.population_total AS pop_total_2025,
t2012.vehicle_total   AS traffic_2012,
t2025.vehicle_total   AS traffic_2025
FROM pop p2012
LEFT JOIN pop p2025
ON p2025.city_district = p2012.city_district
AND p2025.statistical_quarter = p2012.statistical_quarter
AND p2025.year = 2025
LEFT JOIN traffic t2012
ON t2012.city_district = p2012.city_district
AND t2012.statistical_quarter = p2012.statistical_quarter
AND t2012.year = 2012
LEFT JOIN traffic t2025
ON t2025.city_district = p2012.city_district
AND t2025.statistical_quarter = p2012.statistical_quarter
AND t2025.year = 2025
WHERE p2012.year = 2012
```

```
),
stress AS (
SELECT
city_district,
statistical_quarter,
pop_total_2012,
pop_total_2025,

    -- Manual imputation for missing traffic values (2012)
    CASE
    WHEN statistical_quarter = 'Langstrasse'
         AND traffic_2012 IS NULL
    THEN 12019095

    WHEN statistical_quarter = 'Oberstrass'
         AND traffic_2012 IS NULL
    THEN 8143882

    WHEN statistical_quarter = 'Witikon'
         AND traffic_2012 IS NULL
    THEN 912447

    WHEN statistical_quarter = 'Weinegg'
         AND traffic_2012 IS NULL
    THEN 887316

    WHEN statistical_quarter = 'Albisrieden'
         AND traffic_2012 IS NULL
    THEN 10381552

    WHEN statistical_quarter = 'Saatlen'
         AND traffic_2012 IS NULL
    THEN 648512

    WHEN statistical_quarter = 'Schwamendingen-Mitte'
         AND traffic_2012 IS NULL
    THEN 534118

    ELSE traffic_2012
END AS traffic_2012_fixed,

    -- Manual imputation for missing traffic values (2025)
    CASE
    WHEN statistical_quarter = 'Langstrasse'
         AND traffic_2025 IS NULL
    THEN 9874663
```

```
    WHEN statistical_quarter = 'Oberstrass'
         AND traffic_2025 IS NULL
    THEN 9112940

    WHEN statistical_quarter = 'Albisrieden'
         AND traffic_2025 IS NULL
    THEN 5982344

    WHEN statistical_quarter = 'Schwamendingen-Mitte'
         AND traffic_2025 IS NULL
    THEN 608327

    ELSE traffic_2025
END AS traffic_2025_fixed

FROM paired

),
ranked AS (
SELECT
city_district,
statistical_quarter,
pop_total_2012,
pop_total_2025,
traffic_2012_fixed,
traffic_2025_fixed,

    CASE
        WHEN pop_total_2012 IS NULL OR pop_total_2012 = 0
            THEN NULL
        ELSE ROUND((pop_total_2025 - pop_total_2012) / pop_total_2012 * 100, 1)
    END AS population_growth_pct,

    CASE
        WHEN traffic_2012_fixed IS NULL OR traffic_2012_fixed = 0
            THEN NULL
        ELSE ROUND((traffic_2025_fixed - traffic_2012_fixed) / traffic_2012_fixed * 100, 1)
    END AS traffic_growth_pct,

    CASE
        WHEN pop_total_2012 IS NULL OR pop_total_2012 = 0
          OR traffic_2012_fixed IS NULL OR traffic_2012_fixed = 0
            THEN NULL
        ELSE ROUND(
            ((traffic_2025_fixed - traffic_2012_fixed) / traffic_2012_fixed * 100) -
            ((pop_total_2025 - pop_total_2012) / pop_total_2012 * 100),
            1
```

```
        )
    END AS stress_index_pct,

    NTILE(4) OVER (
    ORDER BY
        CASE
            WHEN pop_total_2012 IS NULL
                 OR pop_total_2012 = 0
                 OR traffic_2012_fixed IS NULL
                 OR traffic_2012_fixed = 0
            THEN NULL

            ELSE ROUND(
                (
                    (traffic_2025_fixed - traffic_2012_fixed)
                    / traffic_2012_fixed
                    * 100
                )
                -
                (
                    (pop_total_2025 - pop_total_2012)
                    / pop_total_2012
                    * 100
                ),
                1
            )
        END
  ) AS stress_quartile
FROM stress

)
SELECT
city_district AS district_id,
statistical_quarter AS quarter_name,
pop_total_2012 AS population_2012,
pop_total_2025 AS population_2025,
traffic_2012_fixed AS traffic_2012,
traffic_2025_fixed AS traffic_2025,
population_growth_pct,
traffic_growth_pct,
stress_index_pct,
CASE
WHEN stress_index_pct IS NULL THEN 'Insufficient data'
WHEN stress_quartile = 4 THEN 'Commuter pressure'
WHEN stress_quartile = 1 THEN 'Residential pressure'
ELSE 'Balanced'
END AS stress_classification
```

```
FROM ranked
ORDER BY district_id, quarter_name;
```

18 Appendix F: Efficiency & Query Performance Scripts

This appendix collects the SQL snippets used to speed up Metabase dashboards.
Each script lives in `sql scripts_optimized/` inside the repository so that future team members can rerun them before refreshing the dashboards.

18.1 Index Creation (Run once)

```
-- File: sql scripts_optimized/9_create_indexes_performance.sql
USE traffic_population_zh;

-- TrafficMeasurement indexes
CREATE INDEX idx_tm_timestamp          ON TrafficMeasurement(timestamp);
CREATE INDEX idx_tm_measurement_site_id ON TrafficMeasurement(measurement_site_id);
CREATE INDEX idx_tm_composite           ON TrafficMeasurement(measurement_site_id, timestamp);

-- MeasurementSite index
CREATE INDEX idx_ms_counting_site_id ON MeasurementSite(counting_site_id);

-- Population indexes
CREATE INDEX idx_pop_reference_date     ON Population(reference_date_year);
CREATE INDEX idx_pop_district_quarter   ON Population(city_district, statistical_quarter);
CREATE INDEX idx_pop_composite          ON Population(city_district, statistical_quarter, refe

-- Quarter indexes
CREATE INDEX idx_quarter_street   ON Quarter(street_name);
CREATE INDEX idx_quarter_district ON Quarter(city_district);
```

18.2 Optimized Views (Run after the index script)

```
-- File: sql scripts_optimized/Traffic_sql/v_traffic_yearly.sql
CREATE OR REPLACE VIEW v_traffic_yearly AS
SELECT
    YEAR(tm.timestamp) AS year,
    cs.counting_site_id,
    cs.counting_site_name,
    ms.measurement_site_id,
    ms.measurement_site_name,
    ms.direction,
    q.city_district,
    q.statistical_quarter,
    SUM(tm.vehicle_count) AS yearly_vehicle_count
FROM trafficmeasurement tm
```

```sql
JOIN measurementsite ms ON tm.measurement_site_id = ms.measurement_site_id
JOIN countingsite cs    ON ms.counting_site_id    = cs.counting_site_id
JOIN quarter q          ON cs.counting_site_name  = q.street_name
WHERE tm.timestamp >= '2012-01-01'
  AND tm.timestamp <  '2026-01-01'
GROUP BY
    YEAR(tm.timestamp),
    cs.counting_site_id,
    cs.counting_site_name,
    ms.measurement_site_id,
    ms.measurement_site_name,
    ms.direction,
    q.city_district,
    q.statistical_quarter;
```

```sql
-- File: sql scripts_optimized/Traffic_sql/v_traffic_city_yearly.sql
CREATE OR REPLACE VIEW v_traffic_city_yearly AS
SELECT
    year,
    AVG(site_avg) AS avg_vehicle_count_city
FROM (
    SELECT
        cs.counting_site_id,
        YEAR(tm.timestamp) AS year,
        AVG(tm.vehicle_count) AS site_avg
    FROM trafficmeasurement tm
    JOIN measurementsite ms ON tm.measurement_site_id = ms.measurement_site_id
    JOIN countingsite cs    ON ms.counting_site_id    = cs.counting_site_id
    WHERE tm.timestamp >= '2012-01-01'
      AND tm.timestamp <  '2026-01-01'
    GROUP BY cs.counting_site_id, YEAR(tm.timestamp)
) s
GROUP BY year
ORDER BY year;
```

```sql
-- File: sql scripts_optimized/Traffic_sql/v_kpi2_bottlenecks.sql
CREATE OR REPLACE VIEW v_kpi2_bottlenecks AS
SELECT
    cs.counting_site_name,
    CONCAT(LPAD(peak.hour, 2, '0'), ':00') AS peak_hour,
    ROUND(peak.avg_volume, 0) AS avg_volume,
    CASE
        WHEN peak.avg_volume > 700 THEN 'Bottleneck'
        ELSE 'Normal'
    END AS status
FROM (
    SELECT
        cs.counting_site_id,
```

```
        HOUR(tm.timestamp) AS hour,
        AVG(tm.vehicle_count) AS avg_volume,
        ROW_NUMBER() OVER (
            PARTITION BY cs.counting_site_id
            ORDER BY AVG(tm.vehicle_count) DESC
        ) AS rn
    FROM trafficmeasurement tm
    JOIN measurementsite ms ON tm.measurement_site_id = ms.measurement_site_id
    JOIN countingsite cs    ON ms.counting_site_id    = cs.counting_site_id
    WHERE tm.timestamp >= '2023-01-01'
      AND tm.timestamp <  '2026-01-01'
    GROUP BY cs.counting_site_id, HOUR(tm.timestamp)
) peak
JOIN countingsite cs ON peak.counting_site_id = cs.counting_site_id
WHERE peak.rn = 1
ORDER BY peak.avg_volume DESC;
```

```
-- File: sql scripts_optimized/Traffic_sql/v_kpi3_dominant_direction.sql
CREATE OR REPLACE VIEW v_kpi3_dominant_direction AS
WITH direction_totals AS (
    SELECT
        cs.counting_site_name,
        ms.direction,
        SUM(tm.vehicle_count) AS direction_volume
    FROM trafficmeasurement tm
    JOIN measurementsite ms ON tm.measurement_site_id = ms.measurement_site_id
    JOIN countingsite cs    ON ms.counting_site_id    = cs.counting_site_id
    WHERE tm.timestamp >= '2023-01-01'
      AND tm.timestamp <  '2026-01-01'
    GROUP BY cs.counting_site_name, ms.direction
),
site_totals AS (
    SELECT counting_site_name, SUM(direction_volume) AS total_volume
    FROM direction_totals
    GROUP BY counting_site_name
)
SELECT
    dt.counting_site_name,
    dt.direction AS dominant_direction,
    dt.direction_volume AS dominant_volume,
    st.total_volume,
    dt.direction_volume / st.total_volume AS dominance_share
FROM direction_totals dt
JOIN site_totals st ON dt.counting_site_name = st.counting_site_name
WHERE dt.direction_volume = (
    SELECT MAX(direction_volume)
    FROM direction_totals
```

```
    WHERE counting_site_name = dt.counting_site_name
)
ORDER BY dominance_share DESC;
```

18.3 Materialized Tables

18.3.1 KPI0

```
CREATE TABLE traffic_city_yearly_mat (
    year INT NOT NULL,
    avg_vehicle_count_city DECIMAL(12,2) NOT NULL,
    PRIMARY KEY (year)
);
```

```
INSERT INTO traffic_city_yearly_mat
SELECT
    YEAR(tm.timestamp) AS year,
    CAST(AVG(tm.vehicle_count) AS DECIMAL(12,2)) AS avg_vehicle_count_city
FROM trafficmeasurement tm
WHERE tm.timestamp >= '2012-01-01'
  AND tm.timestamp <  '2026-01-01'
GROUP BY YEAR(tm.timestamp);
```

18.3.2 KPI1

```
INSERT INTO population_yearly_quarter_mat
SELECT
    year,
    city_district,
    statistical_quarter,
    SUM(population_total) AS population_total
FROM v_population_yearly
GROUP BY year, city_district, statistical_quarter;
```

```
INSERT INTO traffic_yearly_quarter_mat
SELECT
    year,
    city_district,
    statistical_quarter,
    COALESCE(SUM(yearly_vehicle_count), 0) AS vehicle_total
FROM v_traffic_yearly
WHERE year = 2012
GROUP BY year, city_district, statistical_quarter;
```

(Equivalent INSERT statements were executed for the comparison year 2025.)

18.3.3 KPI2

```
CREATE TABLE traffic_hourly_site_mat (
    counting_site_id VARCHAR(10) NOT NULL,
    hour INT NOT NULL,
    avg_volume DECIMAL(12,2) NOT NULL,
    PRIMARY KEY (counting_site_id, hour)
);
```

```
INSERT INTO traffic_hourly_site_mat
SELECT
    cs.counting_site_id,
    HOUR(tm.timestamp) AS hour,
    CAST(COALESCE(AVG(tm.vehicle_count), 0) AS DECIMAL(12,2)) AS avg_volume
FROM trafficmeasurement tm
JOIN measurementsite ms
    ON tm.measurement_site_id = ms.measurement_site_id
JOIN countingsite cs
    ON ms.counting_site_id = cs.counting_site_id
WHERE tm.timestamp >= '2023-01-01'
  AND tm.timestamp <  '2026-01-01'
GROUP BY cs.counting_site_id, HOUR(tm.timestamp);
```

Supporting aggregation (quarter-level traffic, comparison year)

```
INSERT INTO traffic_yearly_quarter_mat
SELECT
    year,
    city_district,
    statistical_quarter,
    COALESCE(SUM(yearly_vehicle_count), 0) AS vehicle_total
FROM v_traffic_yearly
WHERE year = 2025
GROUP BY year, city_district, statistical_quarter;
```

18.3.4 KPI3

```
CREATE TABLE traffic_direction_site_mat (
    counting_site_name VARCHAR(150) NOT NULL,
    direction VARCHAR(50) NOT NULL,
    direction_volume BIGINT NOT NULL,
    PRIMARY KEY (counting_site_name, direction)
);
```

```
INSERT INTO traffic_direction_site_mat
SELECT
    cs.counting_site_name,
    ms.direction,
    COALESCE(SUM(tm.vehicle_count), 0) AS direction_volume
```

```
FROM trafficmeasurement tm
JOIN measurementsite ms
    ON tm.measurement_site_id = ms.measurement_site_id
JOIN countingsite cs
    ON ms.counting_site_id = cs.counting_site_id
WHERE tm.timestamp >= '2023-01-01'
  AND tm.timestamp <  '2026-01-01'
GROUP BY cs.counting_site_name, ms.direction;
```