



TfC Tools Plugin

User Guide

TfC Tools Plugin User Guide v2.4

Date 7/15/2026

File Name	V	Date	Changes	Authors	Contributors
TfC Tools Plugin - User Guide.docx	2.4	15/07/2026 16:55:00		SME	AME

Table of Contents

Table of Contents	2
List of Figures	4
List of Tables	4
1. Introduction	5
1.1 Who this guide is for	5
1.2 Tool requirements	5
2. TfC Tools	6
2.1 Installing the plugin	6
2.1.1 Option 1: Manual installation from ZIP	6
2.1.2 Option 2: Install from QGIS Plugin Repository	7
2.2 Where to find the plugin.....	7
3. RL2SDI Migration Plugin	9
3.1 Purpose of the Tool.....	9
3.2 How to use the Plugin	9
3.2.1 Plugin User Interface	9
3.2.2 Plugin output	10
3.2.3 OD travel-time estimation	11
4. Export RouteLab to GeoPackage	13
4.1 Purpose of the Tool.....	13
4.2 How to use the Plugin	13
4.2.1 Plugin User Interface	13
4.2.2 Plugin output	14
5. Export SDI (PostGIS) to GeoPackage	15
5.1 Purpose of the Tool.....	15
5.2 How to use the Plugin	15
5.2.1 Plugin User Interface	15
5.2.2 Plugin output	16
6. GIS2GTFS Plugin	17
6.1 Purpose of the Plugin	17
6.2 What this tool does.....	17
6.3 How to use the plugin.....	17
6.3.1 Plugin User Interface	17
6.3.2 Plugin output	18
6.3.3 GTFS validation	19
7. Refresh SDI Derived Layers.....	20
7.1 Purpose of the Tool.....	20

7.2	How to use the Plugin	20
7.2.1	Plugin User Interface	20
7.2.2	Plugin output	21
7.3	The GTFS editing workflow	23
7.3.1	What you can edit, and what is protected	23
7.3.2	Attribute validation	23
7.3.3	Safe key repairs	23
8.	<i>Trip and Route Revenue Estimator</i>	24
8.1	Purpose of the Tool.....	24
8.2	What this tool does.....	24
8.3	How to use the Plugin	24
8.3.1	Required input data.....	24
8.3.2	Plugin User Interface	25
8.3.3	How the fare bands work	27
8.3.4	Plugin outputs	28
8.3.5	Interpretation notes.....	28
8.3.6	Recommended starting setup:.....	28
9.	<i>Vehicle and Passenger Flow Plugin</i>	29
9.1	Purpose of the Tool.....	29
9.2	What this tool does.....	29
9.3	How to use the Plugin	29
9.3.1	Plugin User Interface	29
9.3.2	Plugin output	30
10.	<i>Appendix</i>	32
10.1	Setting up a PostgreSQL Database Connection on QGIS.....	32
10.1.1	Create a new connection	32
10.1.2	Verify that credentials are saved	32
10.2	Database Layers.....	34
10.3	RL2SDI Plugin – Generated PostgreSQL Database Schema	36
10.3.1	Schema: raw	36
10.3.2	Schema: transit	40
10.4	GIS2GTFS Plugin – Required Database Schema.....	45
10.5	Vehicle and Passenger Flow Plugin	47
10.5.1	Required PostgreSQL Database Schema.....	47
10.5.2	Output GeoPackage Layers	48
10.6	Revenue Estimator Plugin – Boarding and alighting models.....	50
10.6.1	FIFO — first in, first out.....	50
10.6.2	Proportional — split by cohort size	51
10.6.3	Distance-weighted — longer riders alight more.....	52

10.6.4	Summary	52
--------	---------------	----

List of Figures

Figure 1	Installing the plugin – select the Plugin menu	6
Figure 2	Installing the plugin - Insert from ZIP.....	6
Figure 3	TfC Tools Plugin in the Plugins menu	7
Figure 4	TfC Tools in the Processing Toolbox panel.....	8
Figure 5	RL2SDI Plugin User Interface.....	10
Figure 6	the plugin output PostgreSQL database	11
Figure 7	RL2GPKG Plugin User Interface	14
Figure 8	SDI2GPKG Plugin User Interface.....	16
Figure 9	GIS2GTFS Plugin User Interface	18
Figure 10	Raw output in Folder 1	19
Figure 11	Main GTFS output in Folder 2.....	19
Figure 12	Refresh SDI Derived Layers Plugin User Interface.....	21
Figure 13	Revenue Estimator Plugin User Interface	27
Figure 14	Vehicle and Passenger Flow Plugin User Interface	30
Figure 15	adding a new PostgreSQL connection: step 1	32
Figure 16	adding a new PostgreSQL connection: step 2	32
Figure 17	editing an existing PostgreSQL connection: step 1	33
Figure 18	editing an existing PostgreSQL connection: step 2	33
Figure 19	example for boarding and alighting model - FIFO.....	50
Figure 20	example for boarding and alighting model - proportional	51
Figure 21	example for boarding and alighting model - distance-weighted.....	52

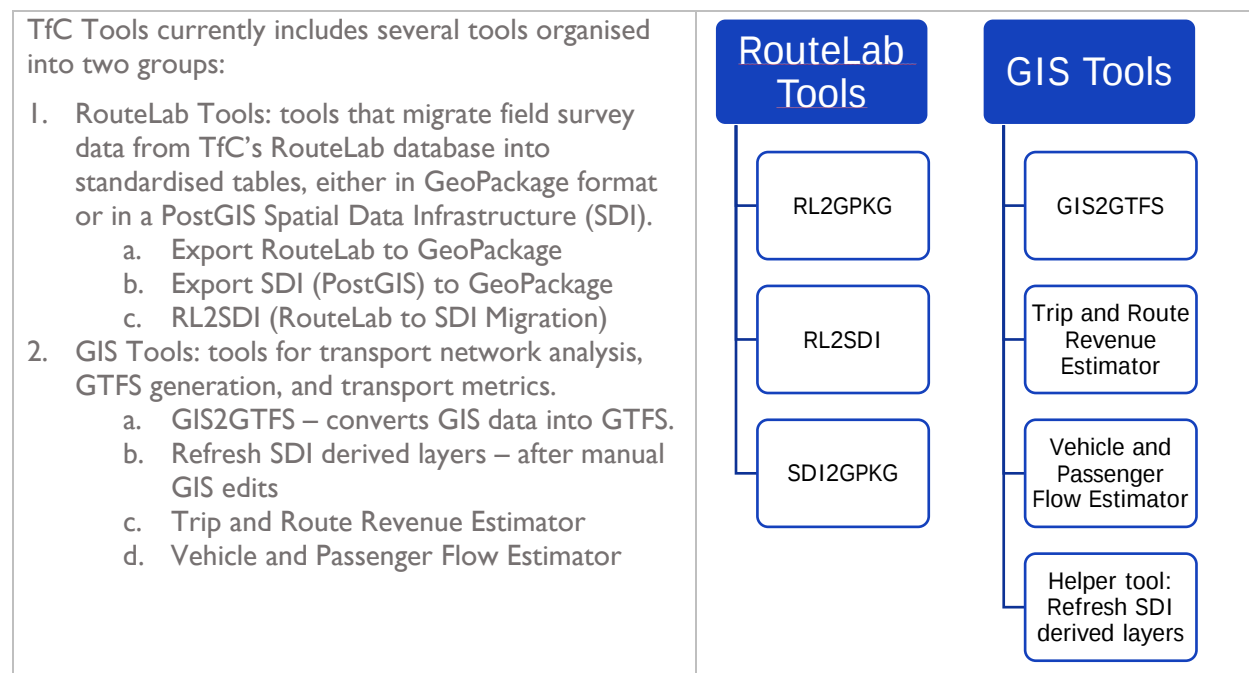
List of Tables

Table 1	GeoPackage contents from the Vehicle and Passenger Flow plugin output	31
Table 2	Raw schema outputs generated by the RL2SDI plugin.....	36
Table 3	Transit schema outputs generated by the RL2SDI plugin.....	40
Table 4	GIS2GTFS Plugin's PostgreSQL database schema requirements	45
Table 5	Vehicle and Passenger Flow Plugin's PostgreSQL database schema requirements.....	47
Table 6	analysis.flow_stop_pair_segments	48
Table 7	analysis.flow_disagg_trip_interval	48
Table 8	analysis.flow_stop_pair_interval_tall.....	49
Table 9	analysis.flow_stop_pair_interval_wide	49

I. Introduction

TfC Tools is a QGIS plugin suite developed by **Transport for Cairo (TfC)** to streamline and standardise transport data processing workflows.

The suite provides user-friendly interfaces for common analytical tasks used in TfC's research and transport data management projects.

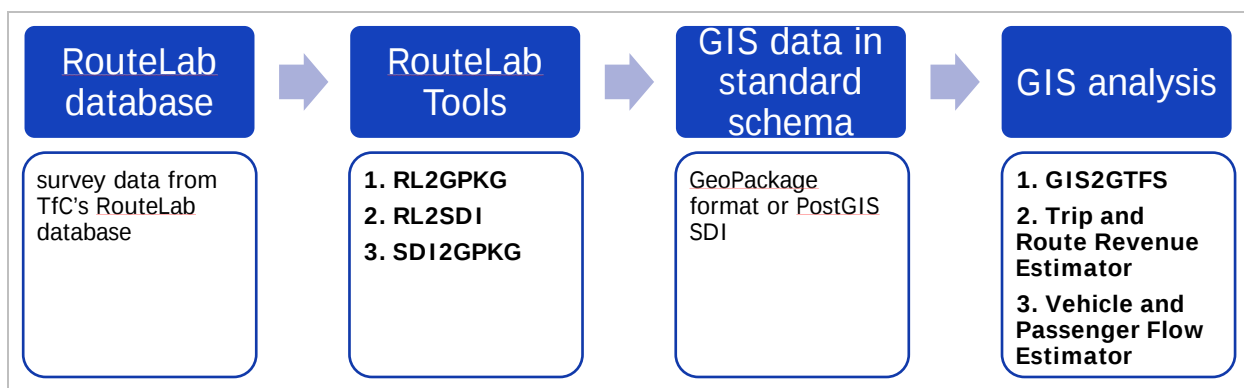


I.1 Who this guide is for

This guide is intended for:

- Transport analysts and GIS specialists working with RouteLab, GTFS, or PostGIS data.
- Researchers and planners aiming to integrate transport data workflows into QGIS.
- Developers or contributors who wish to understand or extend the TfC Tools plugin.

You can use the plugins independently or as part of a workflow:



I.2 Tool requirements

The plugins are compatible with **QGIS 3.40 and later**.

2. TfC Tools

2.1 Installing the plugin

To install **TfC Tools**, you'll first need to download the plugin package and add it to QGIS.

2.1.1 Option 1: Manual installation from ZIP

1. Download the latest plugin ZIP from [the GitHub repository](#)
2. Download and open [QGJS](#) (version 3.40 or higher)
3. From the menu bar, go to **Plugins → Manage and Install Plugins** (Figure 1)

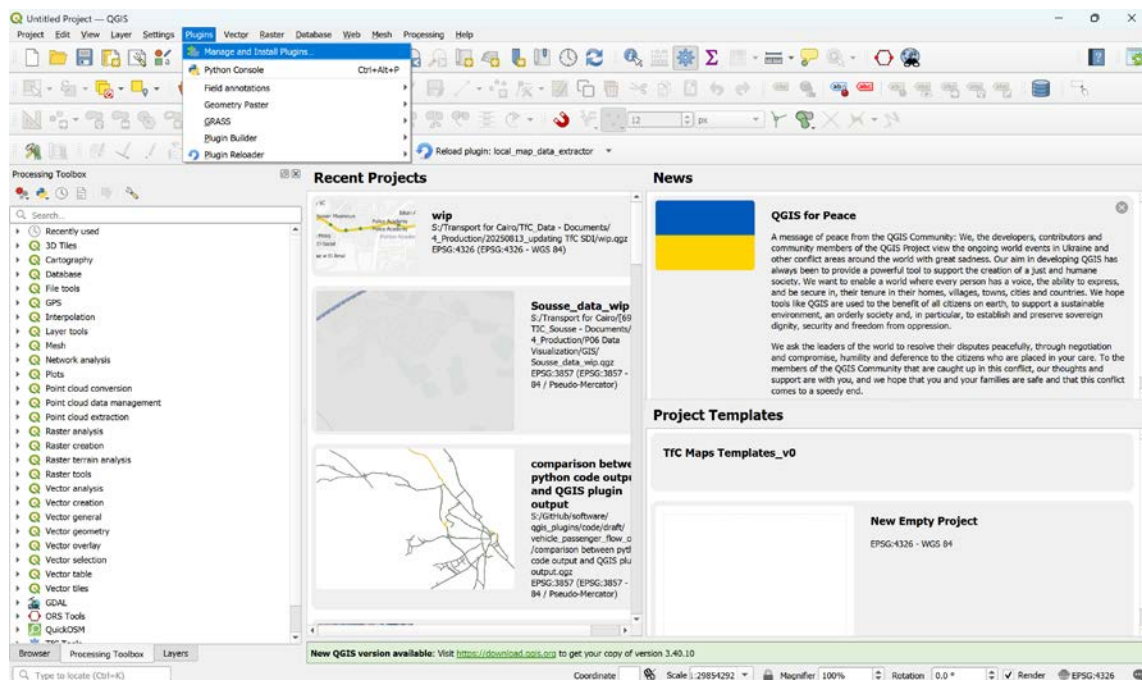


Figure 1 Installing the plugin – select the Plugin menu

4. In the dialog that appears, click the **Install from ZIP** → click **Browse** and select the downloaded ZIP file on your PC → click **Install Plugin** → click **Close** (Figure 2)

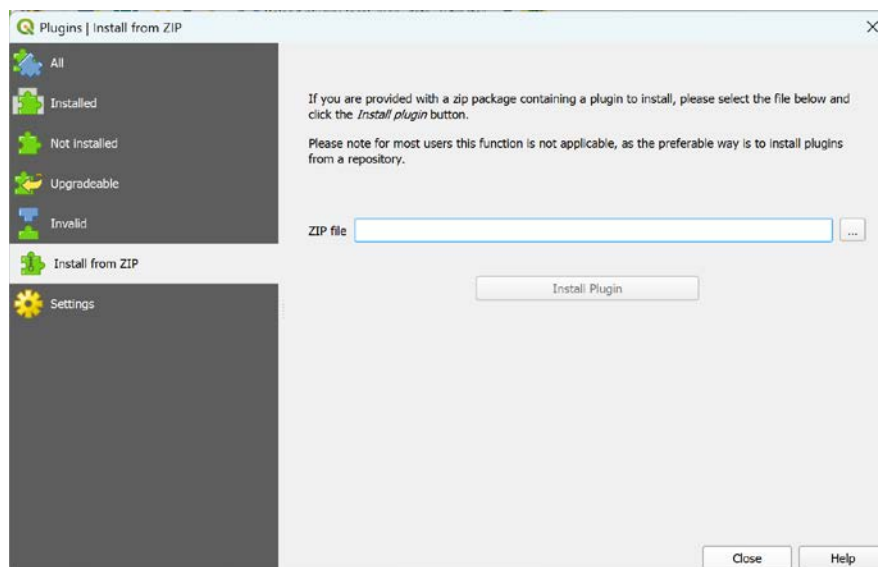


Figure 2 Installing the plugin - Insert from ZIP

2.1.2 Option 2: Install from QGIS Plugin Repository

TfC Tools are available directly through QGIS' Plugin Repository:

1. Open **Plugins** → **Manage and Install Plugins...**
2. Search for **TfC Tools**.
3. Click **Install Plugin**.

2.2 Where to find the plugin

After installation:

Open **Plugins** → **TfC Tools** and make sure it's checked to enable it as in Figure 3.

You can find TfC Tools under the **Processing Toolbox** panel as in Figure 4.

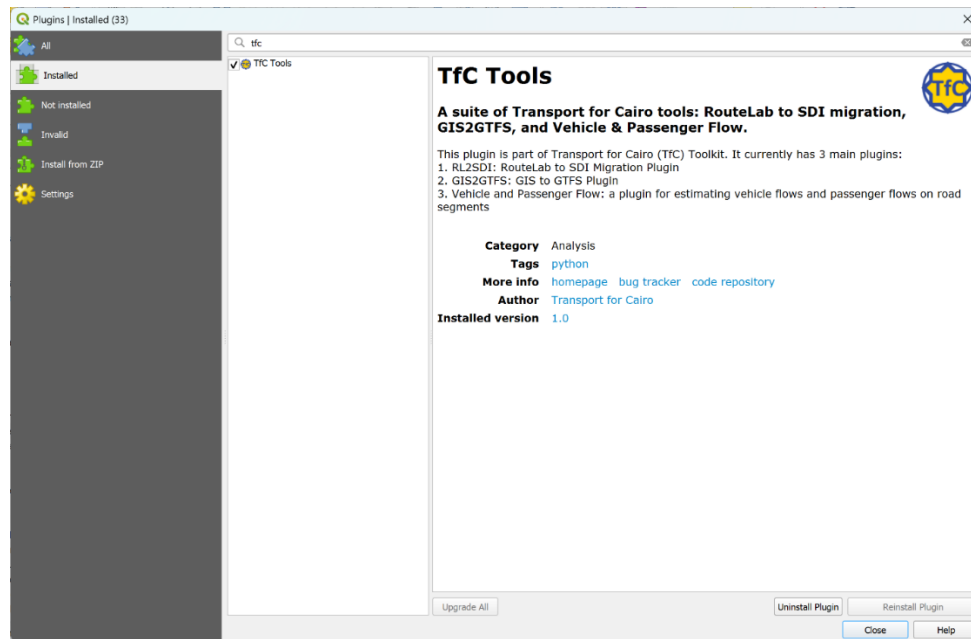


Figure 3 TfC Tools Plugin in the Plugins menu

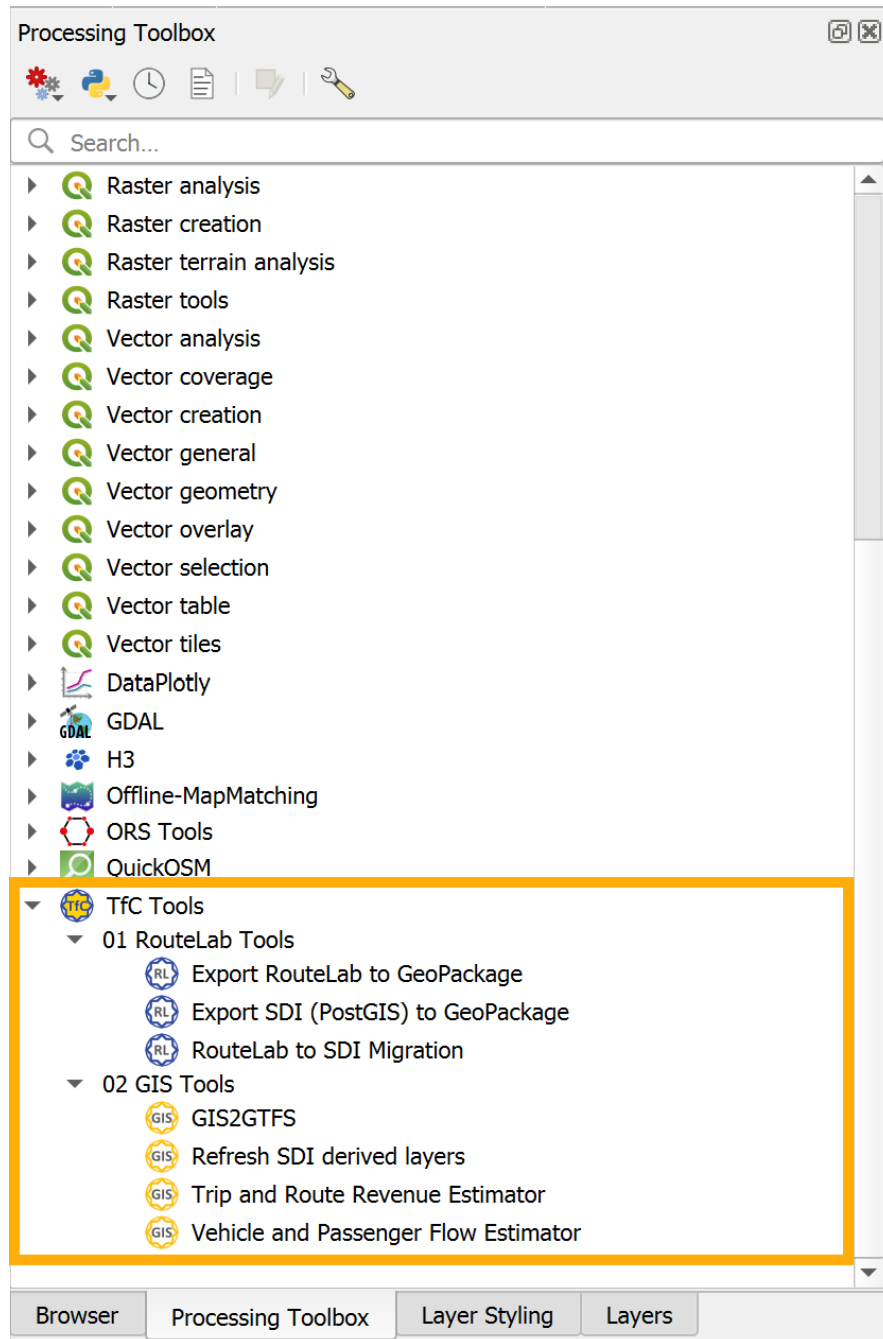


Figure 4 TfC Tools in the Processing Toolbox panel

3. RL2SDI Migration Plugin

3.1 Purpose of the Tool

The plugin migrates field surveys data from the Transport for Cairo TfC's suite RouteLab database to a PostgreSQL database in a standard format that can later be used for analysis.

The plugin automates the migration of field survey data from **Transport for Cairo's (TfC) RouteLab** database into a **Postgres Database**, following a standardised schema that is suitable for analysis and interoperability, particularly with the other **TfC Tools** plugins: **GIS2GTFS**, **Revenue Estimator** and **Vehicle and Passenger Flow**.

3.2 How to use the Plugin

3.2.1 Plugin User Interface

The plugin requires the following inputs:

1. **RouteLab database connection** – The source database containing field survey data (Database credentials are provided by TfC)
2. **PostgreSQL database connection** – The target database where the data will be migrated and structured.
3. **Project ID** – A unique identifier assigned to each project within TfC's RouteLab.
4. **Headway (seconds)** – optional, for trips with missing headway data.
5. **Stop-extraction parameters (advanced, optional)** – fine controls for stop clustering and OD travel-time estimation. Defaults reproduce the historical TfC pipeline, and most users should leave them as is. Adjust them only when the survey data has unusual stop spacing or when per-vehicle trackpoint samples are too small. Available parameters:
 - DBSCAN eps and minpoints – stop clustering neighbourhood radius (m) and minimum neighbours to form a cluster.
 - Grid cell size for cluster deduplication (m).
 - Max cluster-to-trip snap distance (m).
 - Terminal classification radius from trip endpoints (m).
 - Stop-to-trip buffer for sequencing (m).
 - Minimum spacing between consecutive stops on a trip (m).
 - Trackpoint-to-stop buffer for OD travel-time (m).
 - Distinguish speeds by vehicle type – uncheck to pool durations across vehicle types when per-vehicle samples are sparse.

The Plugin interface is shown in Figure 5 below.

01 RouteLab Tools - RouteLab to SDI Migration

Parameters
Log

RouteLab's Observer DB connection

Target SDI DB connection

RouteLab Project ID

Advanced Parameters

Headway (seconds) for empty values (optional) [optional]

DBSCAN eps — stop clustering neighborhood radius (m)

DBSCAN minpoints — min neighbors to form a cluster core

Grid cell size for cluster deduplication (m)

Max cluster-to-trip snap distance (m)

Terminal classification radius from trip endpoints (m)

Stop-to-trip buffer for sequencing (m)

Min spacing between consecutive stops on a trip (m)

Trackpoint-to-stop buffer for OD travel-time (m)

☒ Distinguish speeds by vehicle type (uncheck to pool durations across vehicle types when per-vehicle

RouteLab to SDI Migration

Purpose of the Plugin

The RL2SDI Migration Plugin automates the migration of field survey data from Transport for Cairo's (TfC) RouteLab database into a Spatial Data Infrastructure (SDI).

It creates standardized *raw* and *transit* schemas in a PostGIS database, enabling further analysis or use with the other TfC Tools plugins (GIS2GTFS and Vehicle and Passenger Flow).

How to Use the Plugin

The plugin requires the following inputs:

1. RouteLab database connection (credentials provided by TfC)
2. PostGIS SDI connection where the data will be migrated
3. Enter the Project ID — the unique identifier for your RouteLab project

Advanced Parameters

• Headway fallback value (optional, for trips with missing headway data).

• Stop-extraction tuning — DBSCAN clustering, snap/terminal/spacing thresholds (meters), and per-vehicle vs. pooled OD travel-time speeds. Defaults preserve the historical pipeline.

Outputs

• *raw* schema: cleaned RouteLab data.
• *transit* schema: processed, analysis-ready datasets.

0%

Advanced
Run as Batch Process...
Run
Close

Figure 5 RL2SDI Plugin User Interface

3.2.2 Plugin output

Once the plugin completes its process, two new PostgreSQL schemas will be created within your PostgreSQL database: **raw** and **transit**.

- **raw schema** – contains the cleaned data imported from the RouteLab field surveys.
- **transit schema** – contains the processed and analysis-ready datasets generated from the raw data.

Purpose and overview of each table is provided in Section 10.2 in the Appendix, and a full description of the structure, including all tables, fields, and data types within these two PostgreSQL schemas, is provided in Section 10.3 in the Appendix.

This export also includes the transit_trips base table (in addition to transit_trips_view), so its output matches the Export RouteLab to GeoPackage tool and the Refresh SDI Derived Layers tool can fully rebuild the trip view after edits. The exported GeoPackage also carries the read-only edit form described under Refresh SDI Derived Layers.

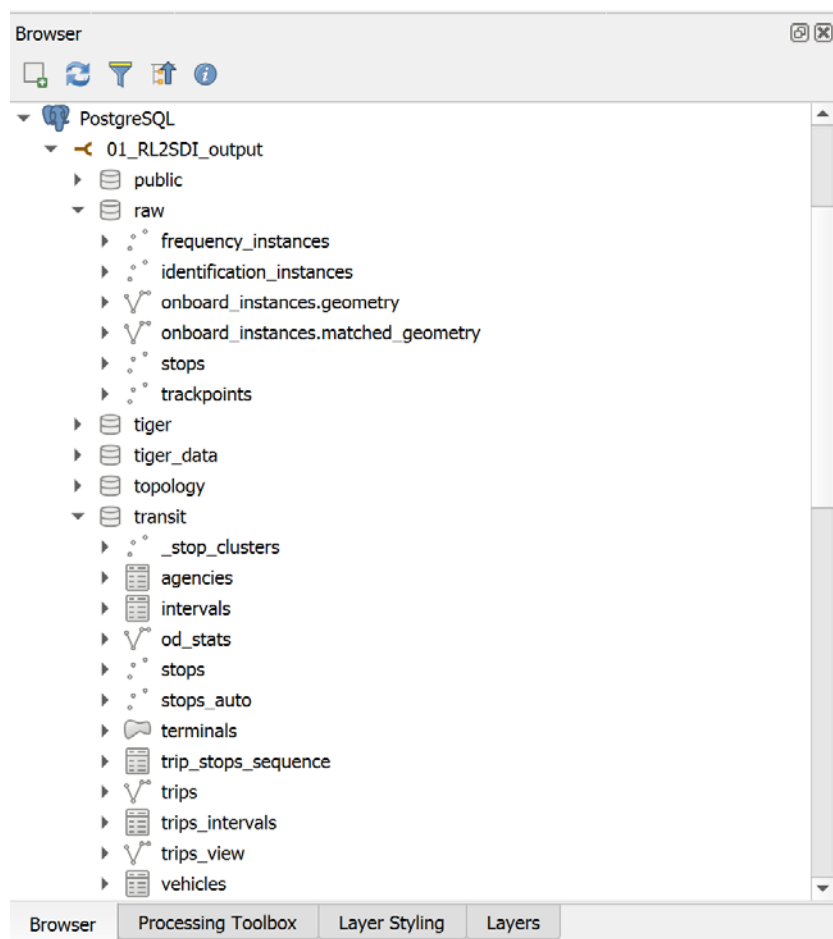


Figure 6 RL2SDI plugin output PostgreSQL database

3.2.3 OD travel-time estimation

When generating transit.od_stats, the plugin estimates segment travel times using a six-tier cascade. It starts from directly observed trackpoint data and falls back to progressively broader averages when observations are sparse. Each row is tagged with the method used so you can filter by data quality:

- **calculated** — direct measurement from trackpoints on the same segment, interval, and vehicle type.
- **interpolated_segment_neighbors** — interpolated from neighbouring segments on the same trip.

- **interpolated_same_segment_other_interval** — taken from the same segment in a different interval.
- **estimated_route_avg** — average speed across the same route.
- **estimated_vehicle_avg** — average speed across the same vehicle type.
- **estimated_citywide** — citywide average speed.

The `n_samples` field reports how many trackpoint observations contributed to the calculation. Rows tagged `calculated` are the most reliable; the deeper the fallback, the more cautiously the value should be interpreted.

4. Export RouteLab to GeoPackage

4.1 Purpose of the Tool

This tool exports RouteLab survey data directly into a GeoPackage file.

The GeoPackage contains standardised tables similar to those produced by the RL2SDI plugin but does not require a PostgreSQL database.

This allows analysts to work with RouteLab data using only local files.

4.2 How to use the Plugin

4.2.1 Plugin User Interface

The plugin requires the following inputs:

1. **RouteLab database connection** – the source database containing field survey data.
2. **Project ID** – unique identifier for the RouteLab project.
3. **Headway (seconds)** – optional, for trips with missing headway data.
4. **Stop-extraction parameters (advanced, optional)** – fine controls for stop clustering and OD travel-time estimation. Defaults reproduce the historical TfC pipeline, and most users should leave them as is. Adjust them only when the survey data has unusual stop spacing or when per-vehicle trackpoint samples are too small. Available parameters:
 - DBSCAN eps and minpoints – stop clustering neighbourhood radius (m) and minimum neighbours to form a cluster.
 - Grid cell size for cluster deduplication (m).
 - Max cluster-to-trip snap distance (m).
 - Terminal classification radius from trip endpoints (m).
 - Stop-to-trip buffer for sequencing (m).
 - Minimum spacing between consecutive stops on a trip (m).
 - Trackpoint-to-stop buffer for OD travel-time (m).
 - Distinguish speeds by vehicle type – uncheck to pool durations across vehicle types when per-vehicle samples are sparse.
5. **Output folder** – location where the GeoPackage file will be generated.

The Plugin interface is shown in Figure 7 below.

01 RouteLab Tools - Export RouteLab to GeoPackage

Parameters
Log

RouteLab PostGIS connection
RouteLab Project ID
☒ Overwrite output if exists
☒ Include raw_* layers
☒ Include QA layers (clusters, stops_auto)
Advanced Parameters
Headway (seconds) for empty values (optional) [optional]
Not set
DBSCAN eps — stop clustering neighborhood radius (m)
150.000000
DBSCAN minpoints — min neighbors to form a cluster core
3
Grid cell size for cluster deduplication (m)
120.000000
Max cluster-to-trip snap distance (m)
30.000000
Terminal classification radius from trip endpoints (m)
75.000000
Stop-to-trip buffer for sequencing (m)
1.000000
Min spacing between consecutive stops on a trip (m)
100.000000
Trackpoint-to-stop buffer for OD travel-time (m)
30.000000
☒ Distinguish speeds by vehicle type (uncheck to pool durations across vehicle types when per-vehicle t

Export RouteLab to GeoPackage
Purpose of the Plugin
This tool exports RouteLab field survey data into a GeoPackage format.
The output follows TfC's standardized schema and can be used without requiring a database connection.
How to Use the Plugin
The plugin requires the following inputs:
1. RouteLab database connection (credentials provided by TfC)
2. Project ID — the unique identifier for your RouteLab project
3. Output folder
Advanced Parameters
• Headway fallback value (optional, for trips with missing headway data).
• Stop-extraction tuning — DBSCAN clustering, snap/terminal/spacing thresholds (meters), and per-vehicle vs. pooled OD travel-time speeds. Defaults preserve the historical pipeline.
Outputs
• GeoPackage containing standardized tables (raw and transit equivalents)
Use Cases
• Offline analysis without PostgreSQL
• Sharing data across teams
• Input to GIS2GTFS, Flow, and Revenue Estimator tools

0%

Advanced
Run as Batch Process...
Run
Close

Figure 7 RL2GPKG Plugin User Interface

4.2.2 Plugin output

The plugin generates a GeoPackage containing standardised tables used for transport analysis and compatible with other TfC Tools workflows.

Purpose and overview of each table is provided in Section 10.2 in the Appendix, and a full description of the structure, including all tables, fields, and data types within the GeoPackage – similar to the output of RL2SDI Plugin – is provided in Section 10.3 in the Appendix.

5. Export SDI (PostGIS) to GeoPackage

5.1 Purpose of the Tool

This tool exports an existing PostgreSQL/PostGIS SDI database into a GeoPackage file.

The exported GeoPackage replicates the SDI schema and allows users to share or analyse data without requiring a database connection.

5.2 How to use the Plugin

5.2.1 Plugin User Interface

The plugin requires the following inputs:

1. **PostgreSQL database connection** – must follow the TfC standard schema, either by: (a) using the PostgreSQL database output from the **RL2SDI Plugin**, or (b) Structuring your data according to the schema described in **Appendix section I0.3**.
2. **Headway (seconds)** – optional, for trips with missing headway data.
3. **Output folder** – location where the GeoPackage will be saved.

The Plugin interface is shown in Figure 8 below.

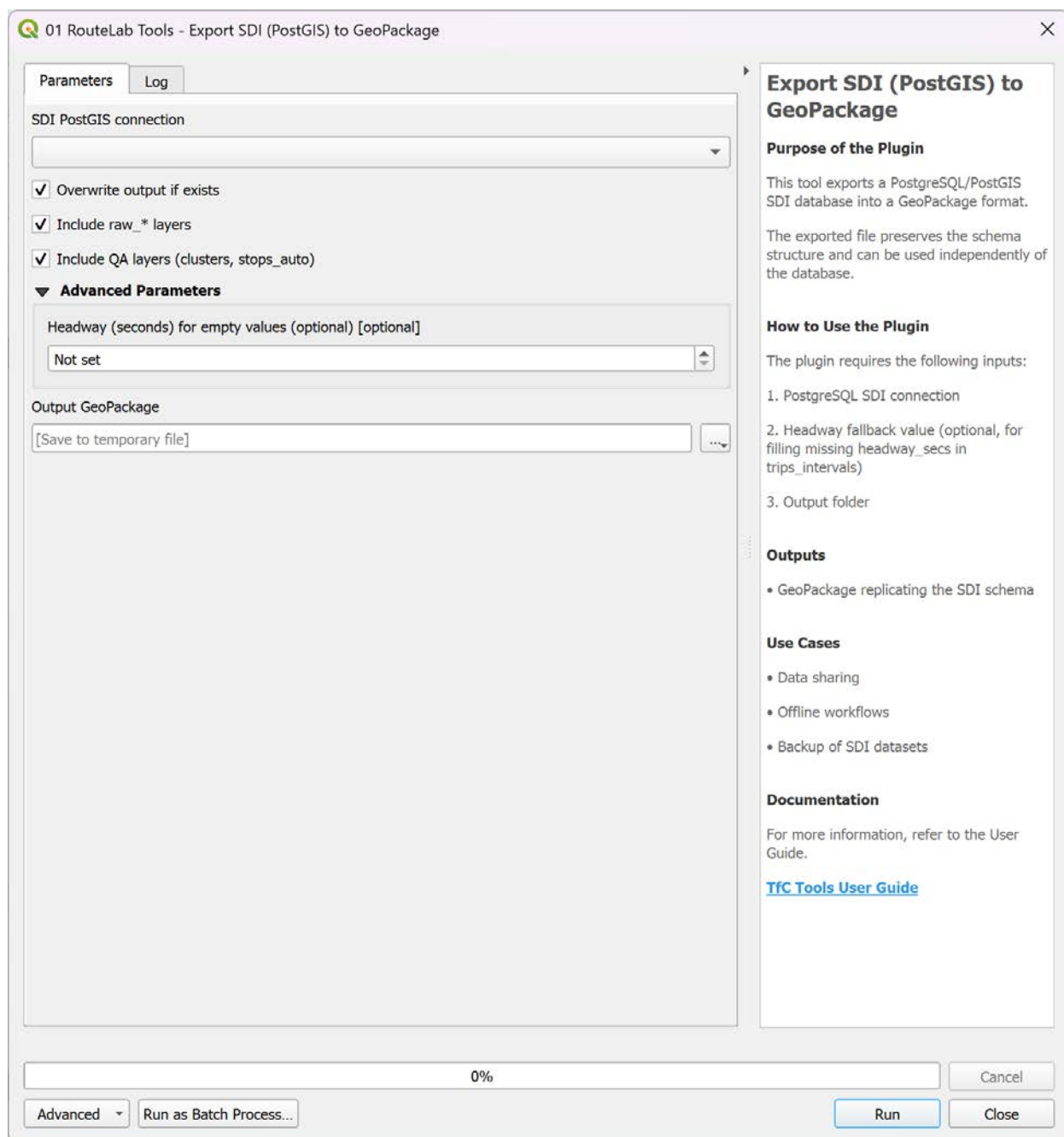


Figure 8 SDI2GPKG Plugin User Interface

5.2.2 Plugin output

The plugin generates a GeoPackage, similar to the output of RL2SDI and RL2GPKG plugins, containing standardised tables used for transport analysis and compatible with other TfC Tools workflows.

Purpose and overview of each table is provided in Section 10.2 in the Appendix, and a full description of the structure, including all tables, fields, and data types within the GeoPackage, is provided in Section 10.3 in the Appendix.

6. GIS2GTFS Plugin

6.1 Purpose of the Plugin

The plugin automates the creation of a **General Transit Feed Specification (GTFS)** dataset from transport data stored in a **PostgreSQL database or GeoPackage** that follows TfC's standardised schema. This includes outputs generated by the RL2SDI and RL2GPKG plugins, as well as other datasets structured according to the same schema. This ensures full compatibility with other TfC tools and workflows.

6.2 What this tool does

1. Reads input data from the selected PostgreSQL database or GeoPackage.
2. Derives service_ids for calendar.txt from the day-of-week flags (mon–sun) on each interval in transit.intervals. Intervals sharing the same day pattern share a service_id:
 - Common patterns get readable names (e.g. svc_daily, svc_weekday, svc_weekend, svc_saturday).
 - Less common patterns get a deterministic svc_<pattern> name (e.g. svc_1010100).
3. Expands each base trip into one GTFS trip per active interval. Trip IDs combine the base ID with the interval suffix, which avoids duplicate (trip_id, start_time) rows in frequencies.txt and correctly splits a base trip running on multiple day-patterns across multiple services.
4. Builds stop_times.txt by applying OD travel-time tables to the stop sequences of each expanded trip.
5. Generates trip shapes from the trip geometries.
6. Writes the GTFS feed as 9 .txt files.

For older SDI exports without day-of-week columns, the plugin falls back to a single service derived from the legacy advanced parameter — see Section 6.3.1.

6.3 How to use the plugin

6.3.1 Plugin User Interface

The plugin requires the following inputs:

1. **Data source** – Choose the source type for the input data, which must follow TfC's standard schema, either by: (a) using the PostgreSQL database output from the **RL2SDI or RL2GPKG Plugin**, or (b) Structuring your data according to the schema described in **Appendix section 10.3**.
 - **PostgreSQL database connection** – Use an existing QGIS PostgreSQL connection pointing to the SDI database.
 - **GeoPackage** – Use an exported SDI / RouteLab GeoPackage.
2. **Feed version** – e.g. 1.0
3. **Start date** – e.g. 20250101
4. **End date** – e.g. 20251231
5. **Use continuous drop-off/pick-up** – an optional GTFS parameter enabled by default; you may disable it if not applicable.
6. **Output folder 1** – temporary folder for intermediate raw files used by the plugin. *(These files are not needed after processing; an empty folder is recommended.)*
7. **Output folder 2** – destination folder for the final GTFS .txt files.

Advanced parameters:

- **Legacy service_id prefix** – used only when the input GeoPackage was produced by an older SDI exporter that did not carry day-of-week flags. In that case, the plugin emits a single fallback service named <prefix>_Daily that runs every day. For modern exports the field is ignored. Default: Ground.

The Plugin interface is shown in Figure 9 below.

Figure 9 GIS2GTFS Plugin User Interface

6.3.2 Plugin output

Upon execution, the plugin generates two sets of outputs corresponding to the two folders selected in the UI:

- **Raw output (Folder 1)** – intermediate processing files not required after the GTFS is created (see Figure 10).
- **GTFS feed (Folder 2)** – the main output, consisting of **9 text files** that make up the GTFS dataset (see Figure 11).

 agency.csv  frequencies.csv  intervals.csv  stops.geojson  terminals.geojson  travel_times_trackpoints.csv  travel_times_trackpoints_filled_na.csv  trip_stop_sequence.csv  trips.geojson  trips_with_intervals.csv Figure 10 Raw output in Folder 1	 agency.txt  calendar.txt  feed_info.txt  frequencies.txt  routes.txt  shapes.txt  stop_times.txt  stops.txt  trips.txt Figure 11 Main GTFS output in Folder 2
--	---

6.3.3 GTFS validation

To ensure the GTFS feed meets specification standards, it is recommended to validate it using **MobilityData's GTFS Validator** by following these steps:

1. Compress the generated .txt files into a single .zip file
2. Download the open [MobilityData GTFS Validator](#)
3. Open the validator and select the location of the GTFS .zip file
4. Click **Validate** To run the checks. A webpage will open displaying the validation report.
5. Review the results:
 - **Errors** must be corrected before the GTFS feed can be considered valid.
 - **Warnings** are non-critical, and the GTFS will still function, but it's recommended to review and resolve them when possible.

7. Refresh SDI Derived Layers

7.1 Purpose of the Tool

This tool rebuilds derived tables and materialized views used in the TfC SDI schema. These layers are generated from primary data and may need to be refreshed when new data is added or modified.

Purpose and overview of each table is provided in Section 10.2 in the Appendix.

Refresh SDI Derived Layers is also the centre of the “edit in QGIS, then re-export GTFS” workflow: you edit the source layers in QGIS, run this tool to validate your edits and rebuild everything that depends on them, then run GIS2GTFS to produce the updated feed. The subsections below describe the validation and key-repair steps and the two options that control them.

7.2 How to use the Plugin

7.2.1 Plugin User Interface

The plugin requires the following inputs:

1. **Data source** – Choose the source type for the input data, which must follow TfC’s standard schema, either by: (a) using the PostgreSQL database output from the **RL2SDI or RL2GPKG Plugin**, or (b) Structuring your data according to the schema described in **Appendix section 10.3**.
 - **PostgreSQL database connection** – Use an existing QGIS PostgreSQL connection pointing to the SDI database.
 - **GeoPackage** – Use an exported SDI / RouteLab GeoPackage.
2. **Include QA layers** – if checked, the tool also attempts to refresh QA outputs
3. **Strict validation** (advanced) – if checked, the refresh stops when any GTFS-attribute error is found, so you fix the data before rebuilding. Unchecked (the default): issues are reported as warnings and the refresh continues.
4. **Apply safe key repairs** (advanced) – if checked, a small set of unambiguous key fixes is written back to the GeoPackage (an agency id stored as a number becomes its text code; integer keys stored as “1.0” become whole numbers). Unchecked (the default): these fixes are only reported, never applied automatically.
5. **Stop-extraction parameters (advanced, optional)** – fine controls for stop clustering and OD travel-time estimation. Defaults reproduce the historical TfC pipeline, and most users should leave them as is. Adjust them only when the survey data has unusual stop spacing or when per-vehicle trackpoint samples are too small. Available parameters:
 - DBSCAN eps and minpoints – stop clustering neighbourhood radius (m) and minimum neighbours to form a cluster.
 - Grid cell size for cluster deduplication (m).
 - Max cluster-to-trip snap distance (m).
 - Terminal classification radius from trip endpoints (m).
 - Stop-to-trip buffer for sequencing (m).
 - Minimum spacing between consecutive stops on a trip (m).
 - Trackpoint-to-stop buffer for OD travel-time (m).

- Distinguish speeds by vehicle type – uncheck to pool durations across vehicle types when per-vehicle samples are sparse.

The Plugin interface is shown in Figure 12 below.

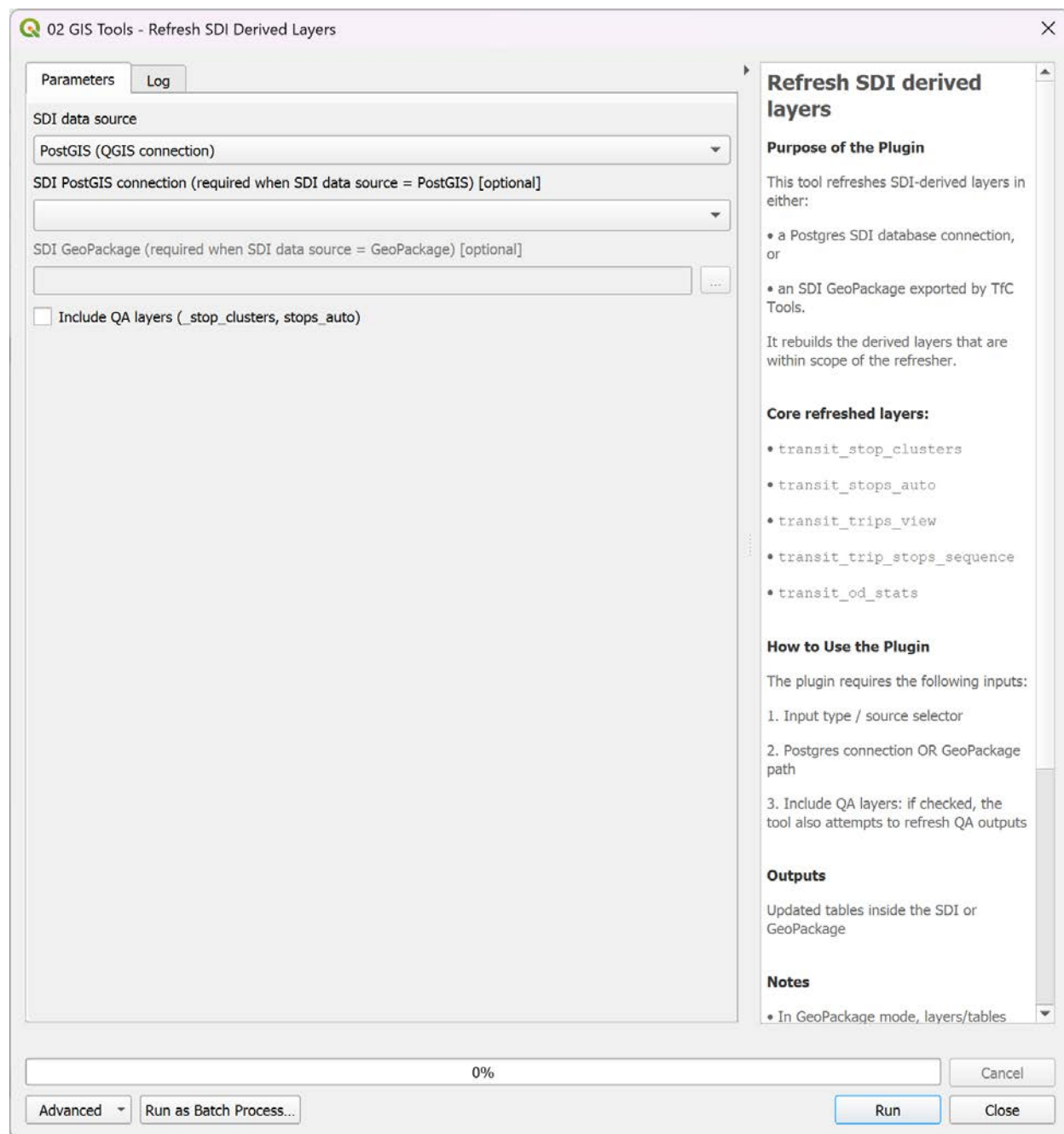


Figure 12 Refresh SDI Derived Layers Plugin User Interface

7.2.2 Plugin output

The tool recalculates derived datasets and overwrites existing layers. The core refreshed layers are:

- transit_stop_clusters
- transit_stops_auto



- transit_trips_view
- transit_trip_stops_sequence
- transit_od_stats

7.3 The GTFS editing workflow

You do not edit the GTFS .txt files by hand. Instead you edit the source layers inside the SDI GeoPackage in QGIS — move a stop, rename a terminal, change a headway, reroute a trip — then run Refresh SDI Derived Layers, and finally run GIS2GTFS to re-export the feed. The Refresh tool recomputes every layer that depends on your edits and checks that your values will produce a valid feed.

7.3.1 What you can edit, and what is protected

Some layers are authored — you edit them directly: transit_stops, transit_terminals, transit_trips_view (trip geometry plus route_id, route_type, direction_id, fare), transit_agencies, transit_vehicles, transit_intervals and transit_trips_intervals. Other layers and columns are derived — the tool recomputes them, so any edit you make to a derived value is overwritten on the next refresh.

To make this clear in QGIS, exported GeoPackages carry a default edit form (from both Export RouteLab to GeoPackage and Export SDI (PostGIS) to GeoPackage). It marks the recomputed transit_trips_view columns — route_short, route_long, trip_short, len_km, origin, destination, vehicle_name and passenger_capacity — as read-only, and adds not-null / unique rules on stop_id and agency_id. To change a read-only value, edit the input it comes from instead:

- **Route long name / headsign** — edit the origin or destination terminal name.
- **Route short name** — edit the agency common_name / serial.
- **Vehicle name or capacity** — edit the row in transit_vehicles.

7.3.2 Attribute validation

Every time you run the tool it validates the GTFS-bound attributes you may have edited and lists any problems in the processing log. Each item is an error (would produce an invalid or misleading feed) or a warning (exportable but worth checking).

Examples of errors: a duplicate or blank stop_id or agency_id; an invalid route_type or a direction_id that is not 0 or 1; an invalid agency timezone; an interval time that does not parse, or two active intervals with the same start and end time; and a foreign key pointing at a row that no longer exists (for example a trip whose agency was deleted). Warnings cover softer issues such as blank names, malformed URLs, or unusual fares.

By default the refresh reports these and continues. Enable the Strict validation option to make it stop until you have fixed every error.

7.3.3 Safe key repairs

Two key problems can be fixed mechanically and unambiguously, and the tool can repair them for you: an agency id stored as an internal number instead of its text code (remapped to the code), and integer keys stored as decimals such as “1.0” (normalised to whole numbers). By default these are only reported; enable Apply safe key repairs to write them into the GeoPackage. Anything that is not one of

these two clear-cut cases — for example a genuinely renamed or deleted key — is left for you to fix, so the tool never guesses.

8. Trip and Route Revenue Estimator

8.1 Purpose of the Tool

This tool estimates revenue for transport services using onboard trip instances, raw stop counts, trip geometry, full route fare, and interval/headway tables.

8.2 What this tool does

1. Reads onboard trip instances, raw stops, trip reference data, and interval/headway tables.
2. Sequences stops using raw stop timestamps.
3. Projects each stop onto the trip geometry to measure cumulative distance traveled along the route.
4. Reconstructs onboard passenger load along the trip using boarding and alighting counts.
5. Infers OD flow blocks from aggregate stop counts using the selected behaviour model.
6. Estimates fare paid by each inferred passenger flow using a distance-based, banded fare model derived from the full route fare.
7. Calculates:
 - observed surveyed-trip revenue
 - interval-scaled service revenue
 - route + direction summary outputs
8. Writes traceable output tables for QA, stop profiles, OD matrix, trip trace, and summary tables.

8.3 How to use the Plugin

8.3.1 Required input data

The selected source must contain, at minimum, the following SDI layers/tables:

- raw.onboard_instances / raw_onboard_instances
- raw.stops / raw_stops
- transit.trips_view / transit_trips_view
- transit.trips_intervals / transit_trips_intervals
- transit.intervals / transit_intervals

Important assumptions:

- transit_trips_view.fare is treated as the full terminal-to-terminal fare.
- Stop sequence is derived from raw_stops.created_at.
- Distance is measured along trip geometry, not straight-line distance.
- Revenue is estimated by inferred OD flow blocks, not directly observed passenger-level OD records.

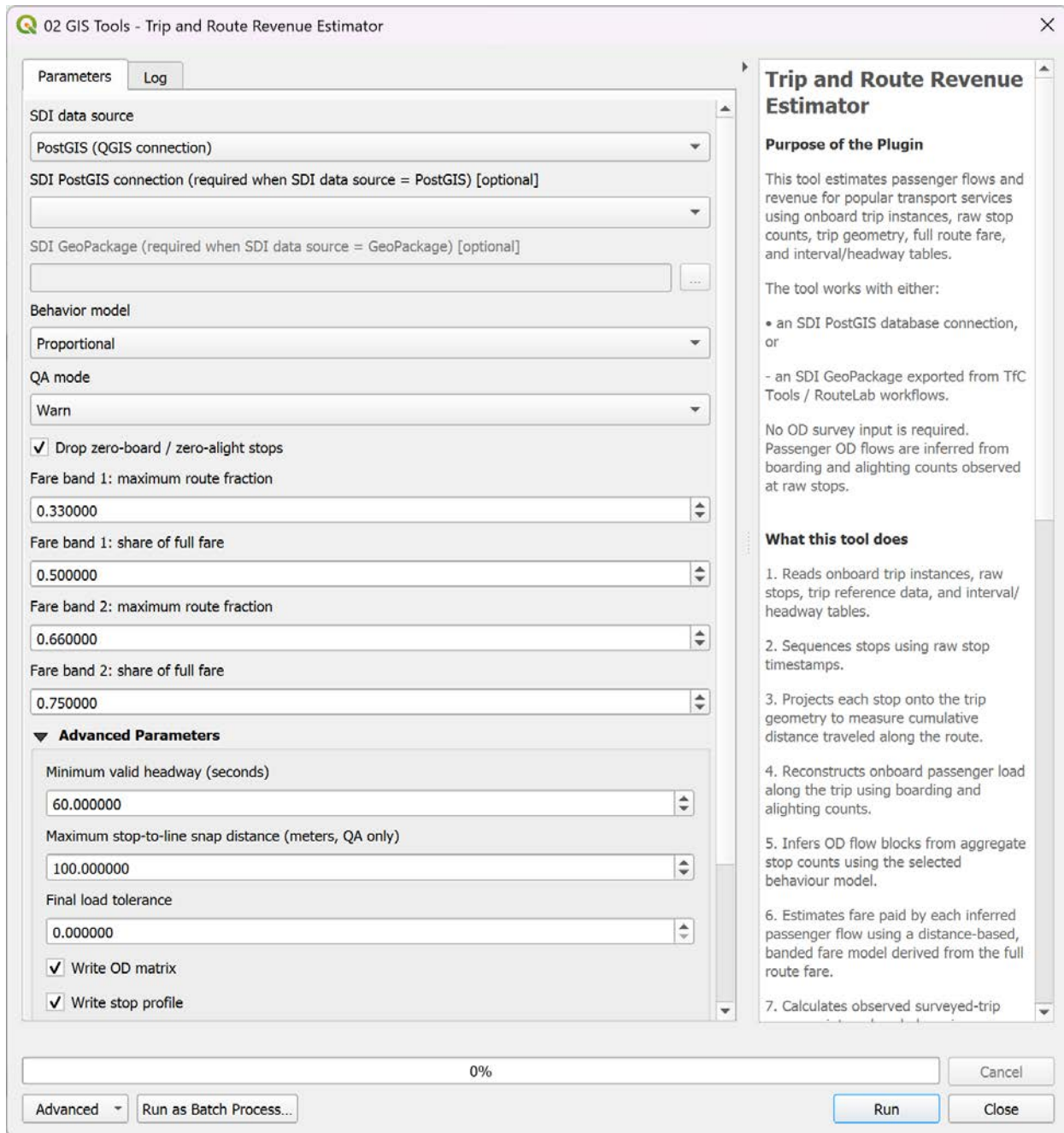
8.3.2 Plugin User Interface

The plugin requires the following inputs:

1. **Data source** – Choose the source type for the input data, which must follow TfC's standard schema, either by: (a) using the PostgreSQL database output from the **RL2SDI or RL2GPKG Plugin**, or (b) Structuring your data according to the schema described in **Appendix section 10.3**.
 - **PostgreSQL database connection** – Use an existing QGIS PostgreSQL connection pointing to the SDI database.
 - **GeoPackage** – Use an exported SDI / RouteLab GeoPackage.
2. **Behaviour model** – Defines how alighting passengers are allocated across the currently onboard passenger cohorts (further details and examples in Appendix section 10.6) Options:
 - **FIFO**: Alighting passengers are removed first from the earliest remaining boarding cohorts. This is the most deterministic and simplest assumption.
 - **Proportional**: Alighting passengers are allocated proportionally across all currently onboard boarding cohorts. This is the recommended default because it is balanced and easy to interpret.
 - **Distance-weighted**: Alighting passengers are allocated across onboard cohorts with greater likelihood for longer traveled distance, while still matching the observed alighting totals exactly. This is more behavioural, but also more assumption-driven.
3. **QA mode** – Controls how the tool reacts to data inconsistencies.
 - **Strict**: Trips with important data or consistency issues are rejected from processing. Use this when you want the cleanest possible outputs and are comfortable dropping problematic trips.
 - **Warn**: The tool keeps processing where possible, but logs QA issues to the output QA tables. This is the recommended default.
 - **Repair minor issues**: The tool attempts limited safe repairs for minor issues, such as filling null board/alight values with zero or dropping certain exact duplicates if supported by the implementation. Core boarding/alighting counts are not silently altered.
4. **Drop zero-board / zero-alight stops** – If enabled, stops where both board = 0 and alight = 0 are excluded before modelling. This can reduce noise from non-events or accidental captures.
5. **Fare band 1 max route fraction** – Upper bound of the first fare band as a fraction of total route length. Example: 0.33 means trips up to 33% of the route length fall into fare band 1.
6. **Fare band 1 share of full fare** – Fare paid for band 1 as a share of the full terminal-to-terminal fare. Example: 0.50 means passengers inferred to travel within band 1 pay 50% of the full fare.
7. **Fare band 2 max route fraction** – Upper bound of the second fare band as a fraction of total route length. Trips above fare band 1 and up to this fraction fall into fare band 2.
8. **Fare band 2 share of full fare** – Fare paid for band 2 as a share of the full terminal-to-terminal fare. Example: 0.75 means passengers inferred to travel within band 2 pay 75% of the full fare.

9. **Minimum valid headway seconds** – Minimum acceptable headway value used during interval scaling. Very small or invalid headways can create unrealistic service scaling results.
10. **Maximum stop-to-line snap distance (meters, QA only)** – Maximum allowed distance between a raw stop point and the trip geometry when projecting stops onto the route. If a stop cannot be projected acceptably, the trip may be flagged in QA.
11. **Nonzero final load tolerance** – Acceptable tolerance for a nonzero passenger load at the end of the trip. A value of 0 means the final load must balance exactly. Larger values allow small residual imbalance before QA flags are raised.
12. **Write OD matrix** – If enabled, writes the inferred OD matrix output table.
13. **Write stop profile** – If enabled, writes the stop-level load reconstruction output table.
14. **Write QA tables** – If enabled, writes QA issue tables summarising problematic trips and issue counts.
15. **Output folder** – Choose the folder where the tool will write its outputs. The tool writes a GeoPackage with multiple result tables and a CSV version of the wide summary table.

The Plugin interface is shown in Figure 13 below.



Parameters Log

SDI data source
PostGIS (QGIS connection)

SDI PostGIS connection (required when SDI data source = PostGIS) [optional]

SDI GeoPackage (required when SDI data source = GeoPackage) [optional]

Behavior model
Proportional

QA mode
Warn

☒ Drop zero-board / zero-alight stops

Fare band 1: maximum route fraction
0.330000

Fare band 1: share of full fare
0.500000

Fare band 2: maximum route fraction
0.660000

Fare band 2: share of full fare
0.750000

Advanced Parameters

Minimum valid headway (seconds)
60.000000

Maximum stop-to-line snap distance (meters, QA only)
100.000000

Final load tolerance
0.000000

☒ Write OD matrix

☒ Write stop profile

0%

Advanced Run as Batch Process... Run Close Cancel

Trip and Route Revenue Estimator

Purpose of the Plugin

This tool estimates passenger flows and revenue for popular transport services using onboard trip instances, raw stop counts, trip geometry, full route fare, and interval/headway tables.

The tool works with either:

- an SDI PostGIS database connection, or
- an SDI GeoPackage exported from TfC Tools / RouteLab workflows.

No OD survey input is required. Passenger OD flows are inferred from boarding and alighting counts observed at raw stops.

What this tool does

1. Reads onboard trip instances, raw stops, trip reference data, and interval/headway tables.
2. Sequences stops using raw stop timestamps.
3. Projects each stop onto the trip geometry to measure cumulative distance traveled along the route.
4. Reconstructs onboard passenger load along the trip using boarding and alighting counts.
5. Infers OD flow blocks from aggregate stop counts using the selected behaviour model.
6. Estimates fare paid by each inferred passenger flow using a distance-based, banded fare model derived from the full route fare.
7. Calculates observed surveyed-trip

Figure 13 Revenue Estimator Plugin User Interface

8.3.3 How the fare bands work

The final band is always the remainder of the route up to 100% of route length, and it always pays 100% of the full fare.

So with the default values:

- 0.00 to 0.33 of route -> 50% of full fare
- >0.33 to 0.66 of route -> 75% of full fare
- >0.66 to 1.00 of route -> 100% of full fare

8.3.4 Plugin outputs

The tool writes the following outputs to the selected output folder.

1. **Main GeoPackage:**

- revenue_stop_profile: Stop-level boarding, alighting, load, and cumulative distance along trip.
- revenue_od_matrix: Inferred passenger OD flow blocks, traveled distance/fraction, fare band, and flow revenue.
- revenue_trip_trace: One row per onboard instance showing the main revenue calculation trace.
- revenue_route_direction_summary: One row per route + direction, with interval-specific columns in wide format.
- revenue_qa_trips: QA issues by onboard trip instance.
- revenue_qa_summary: Summary counts of QA issue types.

2. **Additional CSV:**

- revenue_route_direction_summary.csv

8.3.5 Interpretation notes

- This tool estimates revenue from inferred passenger movements, not directly observed passenger-level OD records.
- Results depend on the selected behaviour model and fare-band assumptions.
- If the route fare field does not represent the full end-to-end fare, revenue estimates will be distorted.
- Interval-scaled revenue depends on the availability and quality of headway values in transit_trips_intervals.

8.3.6 Recommended starting setup:

- Behaviour model = Proportional
- QA mode = Warn
- Fare bands:
 - o band 1 max fraction = 0.33
 - o band 1 fare share = 0.50
 - o band 2 max fraction = 0.66
 - o band 2 fare share = 0.75

9. Vehicle and Passenger Flow Plugin

9.1 Purpose of the Tool

The plugin estimates **vehicle** and **passenger flows** across transport segments using **standardised datasets derived from TfC workflows**. It operates on data structured under the TfC SDI schema, either from a PostgreSQL database or a GeoPackage export. The tool uses onboard survey data, stop sequences, and time intervals to estimate vehicle supply and passenger movements across the network.

The resulting outputs are spatial layers suitable for visualisation and analysis in QGIS.

9.2 What this tool does

Here's a quick overview of what the plugin does once you run it:

1. Reads input data from the selected SDI database or GeoPackage
2. Builds trip segments from stop sequences
3. Uses onboard observations and OD relationships between stops
4. Applies headway information to estimate vehicle supply across intervals
5. Computes vehicle and passenger flows per segment and time interval
6. Writes the results to a GeoPackage file
7. Performs basic validation checks on geometries and attributes

9.3 How to use the Plugin

9.3.1 Plugin User Interface

The plugin requires the following inputs:

1. **Data source** – Choose the source type for the input data, which must follow TfC's standard schema, either by: (a) using the PostgreSQL database output from the **RL2SDI or RL2GPKG Plugin**, or (b) Structuring your data according to the schema described in **Appendix section 10.3**.
 - **PostgreSQL database connection** – Use an existing QGIS PostgreSQL connection pointing to the SDI database.
 - **GeoPackage** – Use an exported SDI / RouteLab GeoPackage.
2. **Trip segments: segmentization threshold (meters)** (*default: 300*)
Controls how trip geometries are divided into segments prior to analysis.
 - Lower values create more (shorter) segments
 - Higher values create fewer (longer) segments
3. **Output folder path** – The location on your computer where the generated files will be saved.

The plugin interface is shown in Figure 14 below.

Figure I4 Vehicle and Passenger Flow Plugin User Interface

9.3.2 Plugin output

The plugin produces a **GeoPackage** (vehicle_passenger_flow.gpkg) containing line-based vehicle and passenger flow outputs for the analysed road network. The GeoPackage contains four layers representing different levels of aggregation and table structure. These layers support both detailed flow tracing and summary mapping in QGIS.

Table I summarises the layers included in the GeoPackage produced by the plugin. Full field definitions and attribute descriptions are provided in Appendix section 10.5.

Table 1 GeoPackage contents from the Vehicle and Passenger Flow plugin output

Layer name	Geometry	Purpose	Typical use
analysis.flow_disagg_trip_interval	LineString	Most detailed output; one record per trip segment and time interval	Detailed QA/QC, tracing individual service/path contributions, vehicle-type analysis
analysis.flow_stop_pair_segments	LineString	Base network flow segments identified by origin stop, destination stop, and segment ID	Reference layer for segment structure and joins
analysis.flow_stop_pair_interval_tall	LineString	Aggregated segment flow by time interval, with one row per segment per time interval	Mapping and filtering by interval using a single field
analysis.flow_stop_pair_interval_wide	LineString	Aggregated segment flow with time interval values stored in separate columns	Simple styling, labelling, and direct comparison of different time intervals

For most mapping and reporting purposes, the *analysis.flow_stop_pair_interval_wide* layer is the most convenient output, as it stores vehicle and passenger flows for different time intervals in separate fields for each road segment. Users requiring interval-based filtering may prefer the *analysis.flow_stop_pair_interval_tall* layer, while the *analysis.flow_disagg_trip_interval* layer provides the most detailed trip-level breakdown for validation and detailed analysis.

10. Appendix

10.1 Setting up a PostgreSQL Database Connection on QGIS

For the TfC Tools plugin to function properly, a **PostgreSQL database connection** with **saved credentials** is required.

This section explains how to create the connection and ensure that the credentials are correctly stored.

Note: This setup only needs to be done once on each computer.

10.1.1 Create a new connection

1. Open the **Browser** panel. If you can't find it go to **View → Panels → check Browser**
2. Right click on **PostgreSQL** and choose **New Connection**
3. In the dialog, fill in the required fields — **Name**, **Host**, **Port**, and **Database**.
4. Under the **Basic** tab, enter your **User name** and **Password**, check **Store**, then click **OK**.
5. Check **Also list tables with no geometry** to be able to see them in the Browser menu

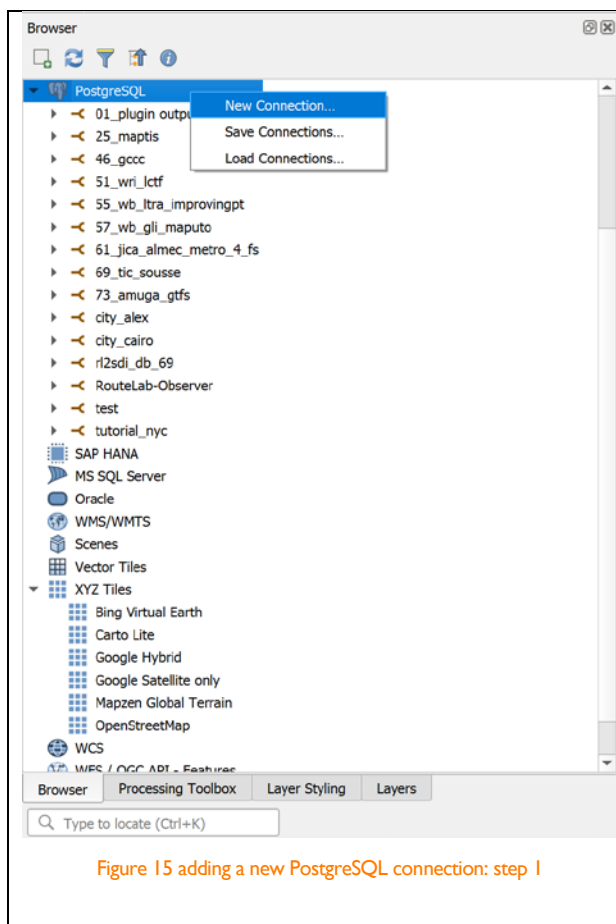


Figure 15 adding a new PostgreSQL connection: step 1

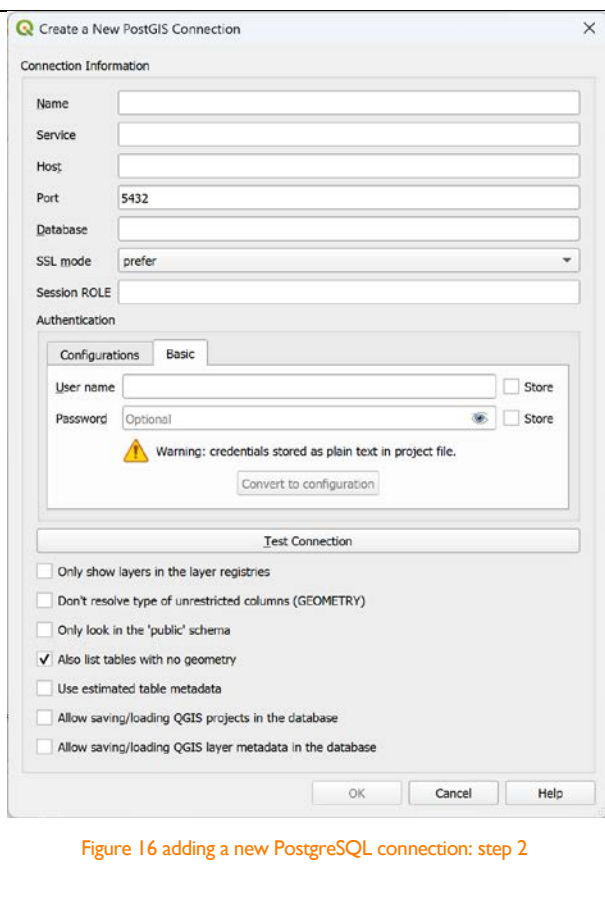


Figure 16 adding a new PostgreSQL connection: step 2

10.1.2 Verify that credentials are saved

Sometimes QGIS allows a PostgreSQL connection with unsaved credentials.

Follow these steps to confirm that your credentials are stored correctly:

1. Open the **Browser** panel
2. Right-click the PostgreSQL database connection you're going to use, and select **Edit Connection**

3. Open the **Basic** tab and verify that your **User name** and **Password** are entered and that **Store** is checked.
4. If the fields are empty, re-enter your credentials, check **Store**, and click **OK**.
5. To confirm that the credentials are saved, close and reopen the **Edit Connection** dialog. If the fields are blank again, repeat the process until the credentials appear after reopening.

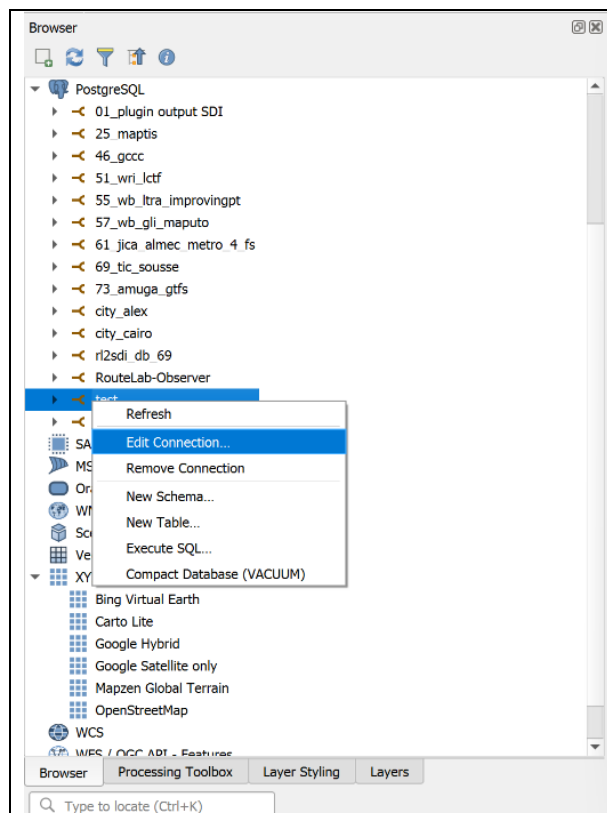


Figure 17 editing an existing PostgreSQL connection: step 1

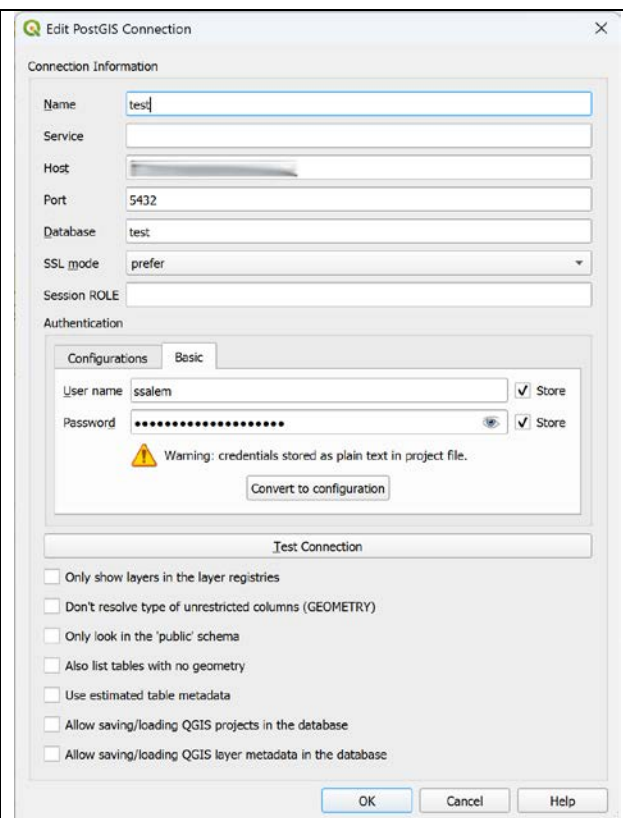


Figure 18 editing an existing PostgreSQL connection: step 2

10.2 Database Layers

Table name	Purpose	What it contains
raw.stops / raw_stops	Stores the raw stop points collected from onboard surveys.	Individual stop observations as recorded in the field, including their geometry and any captured stop name or attributes. Multiple raw points may represent the same real stop.
raw.trackpoints / raw_trackpoints	Stores the raw GPS track points collected during onboard surveys.	Timestamped point locations along surveyed trips.
raw.onboard_instances / raw_onboard_instances	Stores the raw onboard survey instances.	One record per onboard survey run, including survey status or validity and route or trip references.
raw.identification_instances / raw_identification_instances	Stores the raw identification survey records.	Route and terminal identification survey outputs extracted from RouteLab.
raw.frequency_instances / raw_frequency_instances	Stores the raw frequency survey records.	Frequency survey observations captured in RouteLab.
transit.vehicles / transit_vehicles	Stores the vehicle categories used in the processed dataset.	Distinct vehicle or mode names derived from the RouteLab agencies or mode setup, with optional passenger capacity fields.
transit.agencies / transit_agencies	Stores the route operator or service grouping information.	Agency identifiers, names, common names, serial logic, and a link to transit.vehicles.
transit.terminals / transit_terminals	Stores the accepted terminal polygons from the source project.	Terminal names, observer IDs, and terminal geometries.
transit.intervals / transit_intervals	Stores the time intervals used for service and analysis calculations.	Interval names, start and end times, observer IDs, whether the interval is active, and seven day-of-week flags (mon to sun) used by GIS2GTFS to build calendar.txt.
transit.trips / transit_trips	Stores the processed trip geometries derived from RouteLab trip records.	One line geometry per surveyed trip pattern, with links to origin terminal, destination terminal, agency, direction, fare, and RouteLab identifiers.
transit.trips_view / transit_trips_view	Stores an enriched trip layer for analysis and downstream use.	The trip geometry plus route information such as route ID, route short name, route long name, origin, destination, direction, agency, vehicle name, fare, and terminal references.



transit.stops_auto / transit_stops_auto	Stores automatically generated stops before final loading into transit.stops.	Stop points created by clustering raw stop observations, spacing them, snapping them to trip geometries, and assigning a direction bin and stop type.
transit.stops / transit_stops	Stores the final processed stop layer used in the SDI or GeoPackage.	One cleaned stop per processed stop location, including stop ID, name, description, coordinates, geometry, location type, and direction indicator.
transit._stop_clusters / transit__stop_clusters	Stores the stop clusters generated from raw onboard stop observations.	Cluster IDs, number of raw points in each cluster, cluster centroid, and a representative name.
transit.trip_stops_sequence / transit_trip_stops_sequence	Stores the ordered sequence of stops along each trip.	For each trip-stop match, the stop ID, stop name, trip ID, stop sequence, distance along the trip, fractional location along the line, and distance from the previous stop.
transit.trips_intervals / transit_trips_intervals	Stores the estimated service headway for each trip by time interval.	A link between trip ID and interval ID, the final estimated headway in seconds, and the method used to estimate it.
transit.od_stats / transit_od_stats	Stores segment-level operational statistics between consecutive processed stops.	Origin stop ID, destination stop ID, interval ID, interval start, vehicle name, segment distance, average duration, average speed, and segment geometry.

I0.3 RL2SDI Plugin – Generated PostgreSQL Database Schema

The RL2SDI plugin automatically creates two database schemas — **raw** and **transit** — as part of the data migration and standardisation process.

Each schema contains a set of structured tables (and, in some cases, materialized views) that store survey, stop, trip, and route data in a format compatible with TfC's PostgreSQL database structure.

The following tables summarise the output schema of the RL2SDI plugin, including the **database name**, **schema name**, **table/view name**, **field name**, **data type**, and **object type**.

I0.3.1 Schema: raw

Table 2 Raw schema outputs generated by the RL2SDI plugin

Table name	Field name	Data type	Object type
frequency_instances	agency	text	TABLE
	assignment_id	text	TABLE
	avg_headway_sec	double precision	TABLE
	canceled_at	timestamp with time zone	TABLE
	created_at	timestamp with time zone	TABLE
	deleted_at	text	TABLE
	destination	text	TABLE
	finished_at	timestamp with time zone	TABLE
	fr_code	text	TABLE
	fr_id	text	TABLE
	fr_name	text	TABLE
	geometry	geometry(Point,4326)	TABLE
	geometry_properties	text	TABLE
	gid	integer	TABLE
	id	text	TABLE
	interval	text	TABLE
	interval_end	text	TABLE
	interval_id	text	TABLE
	interval_start	text	TABLE
	notes	text	TABLE
	observation_duration	bigint	TABLE
	observations_count	bigint	TABLE
	origin	text	TABLE
	project_id	text	TABLE



	setting_id	text	TABLE
	status	text	TABLE
	survey	text	TABLE
	trip_id	text	TABLE
	updated_at	timestamp with time zone	TABLE
	accepted_at	timestamp with time zone	TABLE
identification_instances	accepted_by	text	TABLE
	agency	text	TABLE
	agency_id	text	TABLE
	bus_number	text	TABLE
	created_at	timestamp with time zone	TABLE
	deleted_at	text	TABLE
	destination	text	TABLE
	destination_name	text	TABLE
	destination_name_local	text	TABLE
	fr_code	text	TABLE
	fr_id	text	TABLE
	fr_name	text	TABLE
	geometry	geometry(Point,4326)	TABLE
	geometry_properties	text	TABLE
	gid	integer	TABLE
	id	text	TABLE
	logs	text	TABLE
	m_destination_name	text	TABLE
	m_destination_name_local	text	TABLE
	m_origin_name	text	TABLE
	m_origin_name_local	text	TABLE
	metadata	text	TABLE
	notes	text	TABLE
	op_from	text	TABLE
	op_to	text	TABLE
	origin	text	TABLE
	origin_name	text	TABLE



	origin_name_local	text	TABLE
	project_id	text	TABLE
	route_id	text	TABLE
	setting_id	text	TABLE
	status	text	TABLE
	trip_id	text	TABLE
	updated_at	timestamp with time zone	TABLE
	user_id	text	TABLE
onboard_instances	agency	text	TABLE
	arrived_at	timestamp with time zone	TABLE
	assignment_id	text	TABLE
	canceled_at	timestamp with time zone	TABLE
	created_at	timestamp with time zone	TABLE
	deleted_at	text	TABLE
	departed_at	timestamp with time zone	TABLE
	destination	text	TABLE
	finished_at	timestamp with time zone	TABLE
	fr_code	text	TABLE
	fr_id	text	TABLE
	fr_name	text	TABLE
	geometry	geometry(LineString,4326)	TABLE
	geometry_properties	text	TABLE
	gid	integer	TABLE
	id	text	TABLE
	interval	text	TABLE
	interval_end	text	TABLE
	interval_id	text	TABLE
	interval_start	text	TABLE
	matched_geometry	geometry(LineString,4326)	TABLE
	notes	text	TABLE
	onboarding_at	timestamp with time zone	TABLE
	origin	text	TABLE
	project_id	text	TABLE



	status	text	TABLE
	status_updated_at	timestamp with time zone	TABLE
	survey	text	TABLE
	trip_id	text	TABLE
	updated_at	timestamp with time zone	TABLE
	valid	boolean	TABLE
	validated_at	timestamp with time zone	TABLE
	validated_by	text	TABLE
	vehicle_id	bigint	TABLE
stops	alight	integer	TABLE
	alight_female	integer	TABLE
	alight_male	integer	TABLE
	board	integer	TABLE
	board_female	integer	TABLE
	board_male	integer	TABLE
	geom	geometry(Geometry,4326)	TABLE
	gid	serial/int	TABLE
	h3_index	text	TABLE
	name	text	TABLE
	observer_id	text	TABLE
	onboard_instance_observer_id	text	TABLE
	parent_onboard_instance_status	text	TABLE
	parent_onboard_instance_valid	boolean	TABLE
stops_created_at	created_at	timestamp with time zone	TABLE
	observer_id	text	TABLE
trackpoints	geom	geometry(Geometry,4326)	TABLE
	gid	serial/int	TABLE
	onboard_instance_id	text	TABLE
	onboard_instance_status	text	TABLE
	onboard_instance_valid	boolean	TABLE

	timestamp	timestamp(0) with time zone	TABLE
--	-----------	-----------------------------	-------

10.3.2 Schema: transit

Table 3 Transit schema outputs generated by the RL2SDI plugin

Table name	Field name	Data type	Object type
_stop_clusters	centroid	geometry	MATERIALIZED VIEW
	cluster_id	integer	MATERIALIZED VIEW
	mode_name	text	MATERIALIZED VIEW
	n_points	bigint	MATERIALIZED VIEW
agencies	agency_id	text	TABLE
	agency_name	text	TABLE
	agency_timezone	text	TABLE
	agency_url	text	TABLE
	common_name	text	TABLE
	gid	serial/int	TABLE
	has_serial	boolean	TABLE
	vehicle_id	integer	TABLE
intervals	gid	serial/int	TABLE
	start_time	time without time zone	TABLE
	end_time	time without time zone	TABLE
	observer_id	text	TABLE
	name	character varying	TABLE
	active	boolean	TABLE
	mon	boolean	TABLE
	tue	boolean	TABLE
	wed	boolean	TABLE
	thu	boolean	TABLE
	fri	boolean	TABLE
	sat	boolean	TABLE
	sun	boolean	TABLE



od_stats	d_id	text	MATERIALIZED VIEW
	dist	double precision	MATERIALIZED VIEW
	duration	integer	MATERIALIZED VIEW
	geom	geometry(LineString,4326)	MATERIALIZED VIEW
	gid	bigint	MATERIALIZED VIEW
	interval_id	integer	MATERIALIZED VIEW
	interval_name	text	MATERIALIZED VIEW
	interval_start	time without time zone	MATERIALIZED VIEW
	o_id	text	MATERIALIZED VIEW
	speed	double precision	MATERIALIZED VIEW
	vehicle_name	text	MATERIALIZED VIEW
	calc_method	text	MATERIALIZED VIEW
	n_samples	bigint	MATERIALIZED VIEW
stops	double	integer	TABLE
	geom	geometry(Point,4326)	TABLE
	gid	serial/int	TABLE
	location_type	integer	TABLE
	stop_desc	text	TABLE
	stop_id	text	TABLE
	stop_lat	double precision	TABLE
	stop_lon	double precision	TABLE
	stop_name	text	TABLE
stops_auto	cluster_id	integer	MATERIALIZED VIEW



	double	integer	MATERIALIZED VIEW
	geom	geometry	MATERIALIZED VIEW
	location_type	integer	MATERIALIZED VIEW
	stop_desc	text	MATERIALIZED VIEW
	stop_lat	double precision	MATERIALIZED VIEW
	stop_lon	double precision	MATERIALIZED VIEW
	stop_name	text	MATERIALIZED VIEW
	stop_type	text	MATERIALIZED VIEW
terminals	geom	geometry(Geometry,4326)	TABLE
	gid	serial/int	TABLE
	name	text	TABLE
	name_ar	text	TABLE
	observer_id	text	TABLE
trip_stops_sequence	distance	double precision	MATERIALIZED VIEW
	distance_frac	double precision	MATERIALIZED VIEW
	distance_from_prev	double precision	MATERIALIZED VIEW
	gid	bigint	MATERIALIZED VIEW
	observer_trip_id	text	MATERIALIZED VIEW
	stop_id	text	MATERIALIZED VIEW
	stop_name	text	MATERIALIZED VIEW
	stop_sequence	bigint	MATERIALIZED VIEW



	t_id	integer	MATERIALIZED VIEW
	vehicle_name	text	MATERIALIZED VIEW
trips	agency_id	integer	TABLE
	agency_serial	text	TABLE
	d_id	integer	TABLE
	direction_id	integer	TABLE
	fare	real	TABLE
	geom	geometry(LineString,4326)	TABLE
	gid	serial/int	TABLE
	o_id	integer	TABLE
	observer_id	text	TABLE
	observer_route_id	text	TABLE
	route_type	integer	TABLE
	service_id	text	TABLE
trips_intervals	gid	serial/int	TABLE
	headway_estimation_method	text	TABLE
	headway_secs	integer	TABLE
	interval_id	integer	TABLE
	trip_id	integer	TABLE
trips_view	agency_id	text	MATERIALIZED VIEW
	d_id	integer	MATERIALIZED VIEW
	destination	text	MATERIALIZED VIEW
	direction_id	integer	MATERIALIZED VIEW
	fare	real	MATERIALIZED VIEW
	geom	geometry(LineString,4326)	MATERIALIZED VIEW
	gid	integer	MATERIALIZED VIEW



	len_km	double precision	MATERIALIZED VIEW
	o_id	integer	MATERIALIZED VIEW
	observer_id	text	MATERIALIZED VIEW
	origin	text	MATERIALIZED VIEW
	passenger_capacity	integer	MATERIALIZED VIEW
	route_id	text	MATERIALIZED VIEW
	route_long	text	MATERIALIZED VIEW
	route_short	text	MATERIALIZED VIEW
	route_type	integer	MATERIALIZED VIEW
	service_id	text	MATERIALIZED VIEW
	trip_short	text	MATERIALIZED VIEW
	vehicle_name	character varying	MATERIALIZED VIEW
vehicles	gid	serial/int	TABLE
	name	character varying	TABLE
	passenger_capacity	integer	TABLE

I0.4 GIS2GTFS Plugin – Required Database Schema

The following table lists everything the GIS2GTFS plugin expects under the schema “transit”: which tables/views must exist, what columns they need, and why. Use it as a checklist to prepare your database before running the plugin.

Table 4 GIS2GTFS Plugin's PostgreSQL database schema requirements

Table Name	Table Description	Column	Type	Notes / Description
transit.agencies	Operator/agency definitions; joined with vehicles and used to set pickup/dropoff behavior.	agency_id	text	Unique agency key; joins from trips_view.agency_id.
		agency_name	text	GTFS agency_name.
		agency_url	text (http/https)	GTFS agency_url.
		agency_timezone	text (IANA TZ)	GTFS agency_timezone (e.g., Africa/Cairo).
		vehicle_id	integer (FK)	FK → transit.vehicles.id.
		has_serial	boolean	When true, plugin sets continuous_pickup & continuous_drop_off = 1.
transit.vehicles	Vehicle types/properties referenced by agencies.	gid	serial / integer (PK)	Primary key; referenced by agencies.vehicle_id.
		name	text	Exported as vehicle_name; used in downstream joins.
transit.stops	Public transport stops with stable IDs and map locations (for stops.txt).	gid	serial / integer (PK)	Stable stop identifier.
		stop_name	text	Human-readable name.
		geom	geometry(Point, 4326)	Source for stop_lat/stop_lon; must be valid points in WGS84.
transit.trips_view	Trips view used for routes, trips, and shapes. Geometry must	gid	serial / integer (unique)	Stable numeric key; used by frequencies.trip_id.



	be non-null for exported rows.	observer_id	text	
		route_id	text	GTFS route reference.
		service_id	text	GTFS service reference.
		direction_id	smallint (0/1)	GTFS direction flag.
		destination	text	Mapped to GTFS trip_headsign.
		route_short	text	Mapped to GTFS route_short_name.
		route_long	text	Mapped to GTFS route_long_name.
		route_type	integer	GTFS route_type.
		agency_id	text (FK)	Joins to agencies.agency_id.
		geom	geometry(Linestring, 4326)	Used to generate shapes.txt (point sequences).
transit.intervals	Time windows used across stop_times and frequencies (e.g., morning peak).	gid	serial / integer (PK)	Primary key; joins from frequencies and od_stats.
		start_time	Time (HH:MM:SS)	Interval start (time of day).
		end_time	Time (HH:MM:SS)	Interval end (time of day).
		mon	boolean (0/1)	Day-of-week flags. The plugin derives a service_id per distinct pattern and writes one calendar.txt row per service. If absent, the plugin falls back to the legacy service prefix.
		tue		
		wed		
		thu		
		fri		
		sat		
		sun		
		gid	serial / integer	Ignored by builder if present.
		Observer_trip_id	text	
		stop_id	text (FK)	FK → stops.stop_id.
transit.trip_stops_sequence	Ordered stop lists per trip; backbone of stop_times.txt.	stop_sequence	integer (1-based)	Visit order of stops along the trip.

transit.od_stats	Observed/estimated travel times between stop pairs per interval (used to compute arrival/departure).	o_id	text (FK)	Origin stop_id; must exist in transit.stops.
		d_id	text (FK)	Destination stop_id; must exist in transit.stops.
		interval_id	integer (FK)	FK → intervals.gid.
		interval_start	time or timestamp	Start time associated with the measurement window.
		duration	integer (seconds) or numeric	Travel time from o_id to d_id.
		vehicle_name	text	Joined from vehicles via agencies.
transit.trips_intervals	Headway values for trips at given time intervals (e.g. trip from x to y departs every 300 seconds during the morning peak interval)	trip_id	integer	Trip reference aligns with trips_view.gid
		interval_id	integer (FK)	FK → intervals.gid.
		headway_secs	integer	Freeform; usage varies.

10.5 Vehicle and Passenger Flow Plugin

10.5.1 Required PostgreSQL Database Schema

Table 5 Vehicle and Passenger Flow Plugin's PostgreSQL database schema requirements

schema	table	field	type	notes
raw	onboard_instances	id	text	Primary identifier of onboard survey instance.
		trip_id	text	Survey 'trip' grouping; used to relate stops to an instance.
		status	text	Expected values like 'finished' used to filter valid runs.
		valid	boolean	Used to exclude invalid runs.
		departed_at	timestampz	Start timestamp for time-binning and headway logic.
		geometry	geometry(LineString,4326)	GPS path of the onboard run (WGS84).
	stops	matched_geometry	geometry(LineString,4326)	Path matched to network if available.
		board	integer	Boardings counted at this observed stop.
		alight	integer	Alightings counted at this observed stop.
		observer_id	text	Join key to stops_created_at for timestamp.
		geom	geometry(Point,4326)	Observed stop location (WGS84).

	created_at	timestampz	Timestamp of stop record creation.
--	------------	------------	------------------------------------

10.5.2 Output GeoPackage Layers

The plugin produces GeoPackage (.gpkg) that contains line-based segment results at multiple levels of detail. The flow_disagg_trip_interval layer stores the most detailed assignment output by trip, segment order, vehicle type, and time interval. The flow_stop_pair_interval_tall and flow_stop_pair_interval_wide layers provide aggregated vehicle and passenger flows by segment for the analysed periods, in tall and wide table structures respectively. The flow_stop_pair_segments layer provides the base segment reference geometry and identifiers.

10.5.2.1 analysis.flow_stop_pair_segments

Purpose:

Base segment index layer. It defines each flow segment using stop pair identifiers and a segment key, with no flow totals included.

Table 6 analysis.flow_stop_pair_segments

Field name	Type	Description
fid	Integer	Unique feature ID
geom	LineString	Segment geometry
from_id	Text	ID of the upstream/origin stop or node for the segment
to_id	Text	ID of the downstream/destination stop or node for the segment
flow_seg_id	Text	Segment identifier, typically combining from_id and to_id (for example I40__I41)

10.5.2.2 analysis.flow_disagg_trip_interval

Purpose:

Detailed disaggregated output. Each row represents one segment belonging to one trip, for one interval, with vehicle and passenger flow values already calculated.

Table 7 analysis.flow_disagg_trip_interval

Field name	Type	Description
fid	Integer	Unique feature ID
geom	LineString	Segment geometry
trip_id	Text	Unique trip identifier
segment_order	MediumInt	Sequence order of the segment within the trip path
from_id	Text	Origin/upstream stop or node ID
to_id	Text	Destination/downstream stop or node ID
vehicle_name	Text	Vehicle/service category associated with the trip
interval_name	Text	Time period for the result, e.g. morning_peak or afternoon
vehicle_trips_in_interval	Integer	Number of trips for that service/trip in the specified interval
occ_median	Real	Median occupancy used in passenger estimation
vehicle_flow	Integer	Vehicle flow assigned to the segment for that trip and interval

passenger_flow	Integer	Passenger flow assigned to the segment for that trip and interval
----------------	---------	---

10.5.2.3 analysis.flow_stop_pair_interval_tall

Purpose:

Segment flow summary in **tall format**. Each segment appears once per interval, with flow values stored in rows rather than separate columns.

Table 8 analysis.flow_stop_pair_interval_tall

Field name	Type	Description
fid	Integer	Unique feature ID
geom	LineString	Segment geometry
from_id	Text	Origin/upstream stop or node ID
to_id	Text	Destination/downstream stop or node ID
flow_seg_id	Text	Segment identifier
interval_name	Text	Time period, e.g. morning_peak or afternoon
vehicle_flow	Integer	Total vehicle flow on the segment during the specified interval
passenger_flow	Integer	Total passenger flow on the segment during the specified interval

10.5.2.4 analysis.flow_stop_pair_interval_wide

Purpose:

Segment flow summary in **wide format**. Each segment appears once, with morning and afternoon values stored in separate fields.

Table 9 analysis.flow_stop_pair_interval_wide

Field name	Type	Description
fid	Integer	Unique feature ID
geom	LineString	Segment geometry
from_id	Text	Origin/upstream stop or node ID
to_id	Text	Destination/downstream stop or node ID
flow_seg_id	Text	Segment identifier
veh_{interval_name}	Integer	Total vehicle flow on the segment during the time interval, where a field is created per time interval carrying the name of said interval.
pax_{interval_name}	Integer	Total passenger flow on the segment during the morning peak, where a field is created per time interval carrying the name of said interval.

10.6 Revenue Estimator Plugin – Boarding and alighting models

When a stop has both boardings and alightings, the engine has to decide which earlier-boarded passengers are the ones now getting off. The revenue estimator supports three behaviour models for this. All three diagrams below use the same shared trip so they can be compared directly:

- Stop 1: 5 passengers board (cohort A)
- Stop 2: 3 passengers board (cohort B)
- Stop 3: 2 passengers board (cohort C)
- Stop 4: 4 passengers alight — but who?

10.6.1 FIFO — first in, first out

FIFO — first in, first out

Oldest cohort onboard alights first. Like a queue.

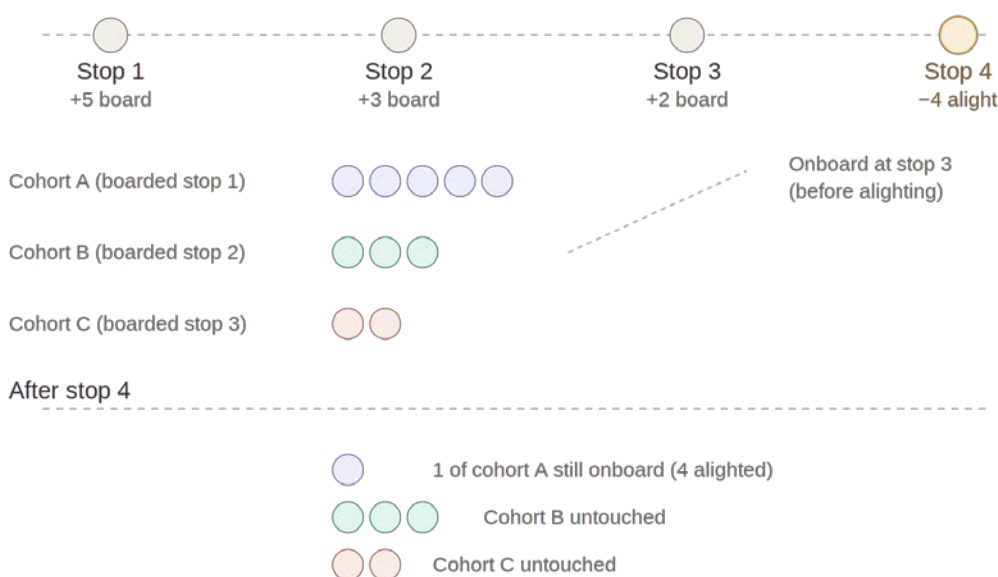


Figure 19 example for boarding and alighting model - FIFO

The oldest cohort onboard alights first. Like a strict queue: the first people to board are the first to get off. In this trip, all 4 alighters come from cohort A.

Properties: FIFO is the only model that produces whole-number alightings, since it treats passengers as a literal queue. It implies riders behave first-in-first-out, which is rarely true in practice but is the simplest to reason about.

10.6.2 Proportional — split by cohort size

Proportional — split by cohort size

Each cohort loses the same fraction (40% here). Larger cohorts give up more.

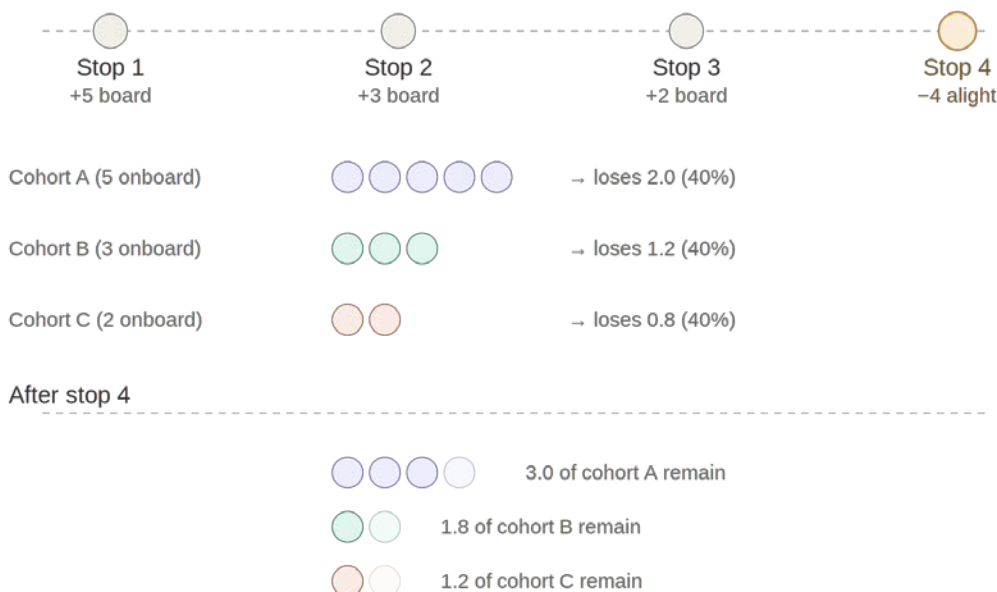


Figure 20 example for boarding and alighting model - proportional

Each cohort loses the same fraction of its members. With 10 onboard and 4 alighting, every cohort gives up 40% of its passengers. Larger cohorts give up more in absolute terms.

Properties: This is the natural “no extra information” assumption — every onboard passenger is equally likely to alight. The fractional counts (2.0, 1.2, 0.8) come from the maths, not from passengers being split in half. The fractions are revenue weights, not real demand counts; they sum correctly to the observed trip revenue.

10.6.3 Distance-weighted — longer riders alight more

Distance-weighted — longer riders alight more

Split is weighted by cohort size $\times$ distance travelled.

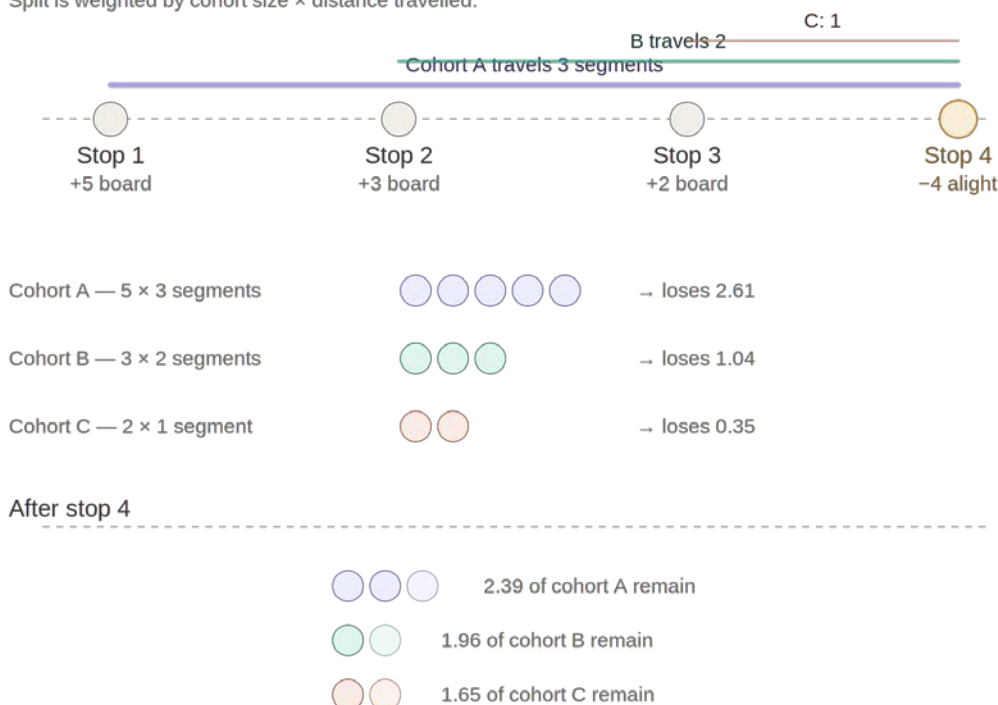


Figure 21 example for boarding and alighting model - distance-weighted

The split is weighted by both cohort size and distance travelled. Each cohort's weight is remaining $\times$ (destination distance — origin distance), so passengers who have been onboard longer are more likely to be the ones alighting.

Calculation: weights are $A = 5 \times 3 = 15$, $B = 3 \times 2 = 6$, $C = 2 \times 1 = 2$, total 23. Of the 4 alighters, A loses $4 \times (15/23) \approx 2.61$, B loses $4 \times (6/23) \approx 1.04$, C loses $4 \times (2/23) \approx 0.35$.

Properties: Adds the intuition that someone who has been on the bus longer is more likely to be near their destination. It tilts alightings toward older cohorts without being as extreme as FIFO.

10.6.4 Summary

Across the three models, the same 4 alightings at stop 4 are attributed differently:

- **FIFO:** A loses 4, B and C untouched.
- **Proportional:** A loses 2.0, B loses 1.2, C loses 0.8.
- **Distance-weighted:** A loses 2.61, B loses 1.04, C loses 0.35.

The choice of model affects how revenue is attributed across origin–destination pairs, since each cohort carries its own boarding stop and therefore its own fare share.