

# Development of Multi-Language Connectors Using GaiaDB

Ritesh Bhirud

University of Massachusetts Amherst

Jacob S. Zelko

University of Saskatchewan

## 1 Background

The OHDSI GIS toolchain extends the HADES ecosystem to integrate geospatial, social determinant of health (SDoH), and other environmental data with OMOP CDM patient data [1]. Central to the GIS toolchain is GaiaDB, a PostGIS-enabled geospatial data store, and gaiaCore, a processing engine for spatial-temporal data ingestion and exposure generation [4]. Together, they work to provide two OMOP CDM-compatible extension tables: `LOCATION_HISTORY`, which records temporal sequences of patient residences, and `EXTERNAL_EXPOSURE`, which links persons to place-based measurements such as neighborhood income, social vulnerability scores, and transit access indices. With the growing maturity of GIS tools, the GIS work group has been exploring how GaiaDB can be used as a multipurpose resource that spans and bridges communities.

The gaiaCore repository currently provides GaiaDB interfaces (referred to as “connectors”) written in Python, R, Julia, Bash, and Java that are each separately maintained manually without a shared specification [4]. At the moment, this maintenance approach does not scale: as the PostgREST API surface of GaiaDB grows, maintaining feature parity across connectors by hand becomes error-prone and burdensome. To address this gap, we have been developing a common API specification based on OpenAPI that automatically yields multi-language connectors [3]. In this work, we discuss initial OpenAPI specification draft development, its early capabilities in generating multiple connectors automatically, demonstrate a connector-powered example analysis, and a summary of future plans.

## 2 Methods

To address the connector maintenance challenge systematically, we are developing an OpenAPI specification for the gaiaCore PostgREST API [3]. The specification defines endpoint signatures, request parameters, and response schemas in a language-agnostic format, with the goal of enabling automatic generation of connector code across all supported languages from a single source of truth. Rather than manually maintaining feature parity between language-specific implementations, contributors update the OpenAPI spec and regenerate the foundational capabilities of connectors, ensuring consistency as the GaiaDB API surface evolves.

As an initial effort, we developed `GaiaDB.jl`, a connector based off the original gaiaCore Julia connector. This package exists alongside the other manually maintained language connectors . A paginated internal helper, `_get_all()`, transparently handles PostgREST’s 1,000-row response cap across all endpoints. The package ships with 24 unit tests, all passing against a live local PostgREST instance. Listing 1 demonstrates the minimal usage pattern for querying location and data source records.

```

1 using GaiaDB
2
3 const BASE_URL = "http://localhost:3000"
4
5 locs      = list_locations(BASE_URL; county = "Cook", limit = 5)
6 sources = get_data_sources(BASE_URL)
7
8 println(first(locs, 3))
9 # 3x7 DataFrame
10 # Row / location_id address_1          city    state  county  latitude  longitude
11 # -----
12 #   1 /          224 Census Tract 8051.02 Chicago IL    Cook    42.0555  -87.6719
13 #   2 /          886 Census Tract 4910  Chicago IL    Cook    41.7681  -87.6980
14 #   3 /          887 Census Tract 5101  Chicago IL    Cook    41.7894  -87.7034

```

Listing 1: Querying Cook County locations and data sources using `GaiaDB.jl`. Results are returned as `DataFrames.jl` objects composable with the broader Julia ecosystem, matching the response schemas defined in the OpenAPI specification.

## Example Analysis: SDoH-Informed Agent-Based Modeling

To illustrate the analytics that connector-powered GaiaDB access could enable across programming ecosystems, we demonstrate an end-to-end SDoH-informed epidemic modeling workflow. We populated GaiaDB with real SDoH data using a Python ingestion script that fetches ACS 5-Year 2022 estimates for all 1,332 census tracts in Cook County, Illinois from the US Census Bureau REST API without requiring an API key. Four SDoH measures were extracted per tract: median household income (table B19013), percent zero-vehicle households as a proxy for transit dependency (table B08141), percent uninsured (profile DP03), and population density derived from TIGERweb land area estimates.

As the connectors themselves can provide the foundational functionality to work with GaiaDB, we easily extended `GaiaDB.jl` with an `initialize_agents_from_sdoh` function that queries GaiaDB via the connector and returns a typed `EpiAgent` population suitable for use in agent-based epidemic simulations. Home locations are sampled proportional to tract population density rather than uniformly at random; transit dependency is drawn from a distribution centered on each tract's percent zero-vehicle households; income quintile is determined from the full Cook County income distribution. Listing 2 shows the core initialization logic.

```

1 function initialize_agents_from_sdoh(base_url::AbstractString, n_agents::Int;
2                                   county::AbstractString = "Cook")
3     tracts = _load_tract_sdoh(base_url; county)
4     tracts = dropmissing(tracts, :pop_density)
5     tracts = filter(row -> row.pop_density > 0, tracts)
6
7     weights = Float64.(tracts.pop_density); weights ./= sum(weights)
8     quintile_fn = _income_to_quintile(collect(skipmissing(tracts.median_income)))
9     pct_novehicle = coalesce.(tracts.pct_no_vehicle, 0.0)
10
11     home_indices = wsample(1:nrow(tracts), weights, n_agents)
12     work_indices = wsample(1:nrow(tracts), weights, n_agents)
13
14     agents = Vector{EpiAgent}(undef, n_agents)
15     for i in 1:n_agents
16         hi = home_indices[i]
17         td = clamp(pct_novehicle[hi] / 100.0 + randn() * 0.05, 0.0, 1.0)
18         agents[i] = EpiAgent(
19             id = i,
20             home_location_id = tracts.location_id[hi],
21             work_location_id = tracts.location_id[work_indices[i]],
22             transit_dependency = td,
23             income_quantile = quintile_fn(tracts.median_income[hi]),
24         )
25     end
26     return agents
27 end

```

Listing 2: `initialize_agents_from_sdoh`: agents are initialized from GaiaDB census tract data queried via `GaiaDB.jl`. Home locations are weighted by population density; transit dependency and income quintile are drawn from real ACS tract-level distributions rather than uniform priors.

This `GaiaDB.jl`-initialized agent population was used to replicate and extend the MacRury et al. analytical epidemic toy model [2] with SDoH stratification, using 2,000 agents swept across transit headways from 1 to 25 minutes with 50 stochastic runs per frequency (1,250 total runs).

### 3 Results

The following results illustrate an example of the kind of analysis that becomes possible when connector packages provide programmatic SDoH data access to downstream modeling workflows.

Agent population distributions produced via `GaiaDB.jl` matched census benchmarks closely: mean transit dependency was 17.2% against a population-density-weighted census mean of 17.29%, and the income quintile Q1 share was 20.3% in both the agent population and census data, confirming that real SDoH distributions are faithfully reproduced rather than approximated by uniform priors.

The connector-powered simulation reproduced all eight core MacRury et al. findings, including the inverted-U relationship between transit headway and time to 99% infected, peaking at 15–16 minutes (276 iterations).

Stratifying agents by transit dependency revealed two findings not present in MacRury et al. First, high-transit-dependency agents (top quartile) reached 50% infection approximately 9 iterations earlier than low-dependency agents at 5–7 minute headways, the regime where transit is attractive but insufficiently frequent to prevent crowding. Second, this disparity reverses above 16-minute headways, where low-dependency agents, who walk more at infrequent headways, face higher pedestrian exposure and are infected faster. Income quintile tracked transit dependency across all headway conditions but was not independently predictive once transit dependency was controlled for, suggesting that transit dependency is the proximate mechanism through which income-related disparities manifest in this model. Table 1 summarizes the stratified results across representative headways.

| Headway (min) | $\alpha_{transit}$ | High-TD $t_{50}$ (iters) | Low-TD $t_{50}$ (iters) | Gap |
|---------------|--------------------|--------------------------|-------------------------|-----|
| 1             | 0.998              | 50                       | 50                      | 0   |
| 5             | 0.622              | 84                       | 93                      | +9  |
| 7             | 0.518              | 104                      | 113                     | +9  |
| 15            | 0.269              | 138                      | 139                     | +1  |
| 16            | 0.244              | 139                      | 138                     | -1  |
| 25            | 0.088              | 125                      | 119                     | -6  |

Table 1: Mean iterations to 50% infected for high- and low-transit-dependency agent quartiles across representative transit headways (50 stochastic runs per frequency). Positive gap indicates high-TD agents infected faster; negative gap indicates the disparity crossover where low-TD agents are infected faster due to pedestrian crowding. These results are an example of connector-powered SDoH analytics.

## 4 Conclusions

We presented an OpenAPI-based approach to multi-language connector development for `gaiaCore` and demonstrated through `GaiaDB.jl` the kind of SDoH-informed analytics such connectors enable. The draft OpenAPI 3.1.0 specification covers eight `gaiaCore` endpoints with full parameter and response schema definitions, providing a foundation from which connectors across Python, R, Julia, Bash, and Java can eventually be generated and maintained automatically rather than by hand. The example agent-based modeling analysis shows that making `GaiaDB` programmatically accessible to downstream simulation frameworks reveals health equity dimensions of public health interventions invisible under uniform agent initialization assumptions, and demonstrates the kind of workflow that will generalize to any language once the CI-based connector generation pipeline is complete.

Future plans include finalizing the OpenAPI specification through a four-stage workflow [3]:

1. Review available connectors to determine common features
2. Refining the specification
3. Automating per connector test suites
4. Develop CI pipeline for automatic connector regeneration across all supported languages

## References

- [1] George Hripcsak et al. “Observational Health Data Sciences and Informatics (OHDSI): Opportunities for Observational Researchers”. In: *Studies in Health Technology and Informatics* 216 (2015), pp. 574–578.
- [2] Calum MacRury et al. “Optimizing transport frequency in multi-layered urban transportation networks for pandemic prevention”. In: *Public Transport* 16 (2024), pp. 381–418. DOI: 10.1007/s12469-024-00351-0.
- [3] OHDSI GIS Working Group. *Discuss OpenAPI-Based Connector Auto-Generation: Issue #429*. <https://github.com/OHDSI/GIS/issues/429>. Accessed June 2025. 2025.
- [4] TuftsCTSI. *gaiaCore: Central Component of the OHDSI GIS Gaia Toolchain*. <https://github.com/TuftsCTSI/gaiaCore>. Accessed June 2025. 2025.