

A Semantic Layer for OMOP with Malloy: Reusable, Portable, and Correct-by-Construction Analytics

Ofer Mendelevitch, Kyle Nesbit, James Swirhun

Credible Data Inc

BACKGROUND

Analyses on the OMOP Common Data Model (CDM) share a recurring shape and a recurring set of difficulties. Clinical events live in several one-to-many fact tables (conditions, drug exposures, visits, procedures) that all hang off a single person. Meaning is carried by CONCEPT vocabulary codes rather than readable names. The same cohort, demographic, and temporal logic is rewritten in SQL for every study. The resulting analyses are verbose, hard to reuse, and easy to get subtly wrong, which matters because OHDSI network studies depend on results that are correct and portable across sites [1,2,3].

Malloy is an open-source language for modeling and querying relational data, from the team behind Looker and LookML [5]. An analyst declares once, in a semantic model, each table's join cardinality, vocabulary lookups, and reusable metrics and views; Malloy then compiles every query to SQL. For OMOP this offers four advantages: (i) symmetric aggregates evaluate each metric over its own join path, so several fact tables can be summarized in one query without fan-out errors; (ii) named views and measures capture cohort, baseline, and outcome logic once; (iii) the CONCEPT table is reused under aliases (gender, race, condition, drug), so queries read in clinical terms; and (iv) the model is written against the CDM's table definitions, so it can be pointed at another OMOP v5.3/v5.4 database by changing table paths. The fan-out hazard most often produces silently wrong results, so we use it as the worked example.

METHODS

We built a generic Malloy semantic model for OMOP CDM v5.3/v5.4. It declares each fact table's relationship to PERSON once, reuses the CONCEPT vocabulary under multiple aliases, joins every fact table to CONCEPT_ANCESTOR for concept-set expansion, and defines reusable measures and named views (e.g., `total_conditions`, `total_drugs`, a `baseline-characteristics` view). We exercised it on the Synthea COVID-19 dataset distributed with the `darwin-eu CDMConnector` R package (OMOP CDM v5.3; 10,754 synthetic persons; populated death table) [4,6]. The cohort is persons with a COVID-19 diagnosis (SNOMED concept 37311061; $n=964$; index = first diagnosis). For each analysis we wrote the Malloy query against the model and the equivalent hand-written SQL, and compared results. As a controlled correctness probe we counted each patient's conditions and drugs in one query, both through the model and through a naive join. All numbers were computed on DuckDB over the CDM parquet files; the model and notebooks are open source [6], served with Malloy Publisher [7].

RESULTS

Consider a routine request: for the cohort, the number of conditions and drugs per outcome group (survivors vs decedents). Figure 1 shows it three ways. The Malloy query (a) is compact and returns the correct totals: survivors carried 5,361 conditions and 34,460 drug exposures; decedents 689 and 3,269. The naive SQL (b), the version most analysts write first, joins both fact tables to the patient and counts. Each patient then contributes one row per (condition, drug) pair, so in each group both COUNTs return the same inflated number: 218,913 for survivors and 49,552 for decedents. Conditions equal to drugs is the tell-tale sign of the error.

Query (a) compiles to the SQL in (c). Malloy keeps the same join as (b) but counts each fact table's distinct row IDs, so no row is counted twice and both totals are correct. A careful analyst can get the same totals by hand, either this way or by totalling each fact table per patient before joining. The analyst who writes (a) needs to know neither workaround.

(a) Malloy: one query against the shared model

```
run: condition_occurrence -> {
  where: condition_concept_id
    = 37311061 // COVID-19
  group_by: person_id
} -> {
  join_one: patient with person_id
  group_by:
    vital_status is
    patient.vital_status
  aggregate:
    patients is count()
    total_conditions is
    patient.conditions.count()
    total_drugs is
    patient.drugs.count()
}
```

-- Survived: 916 | 5,361 | 34,460
-- Died: 48 | 689 | 3,269

(b) Naive SQL: the typical error (fan-out)

```
WITH cohort AS ( -- COVID-19
  SELECT DISTINCT person_id
  FROM condition_occurrence
  WHERE condition_concept_id
    = 37311061)
SELECT
  CASE WHEN dt.person_id IS NOT NULL
    THEN 'Died' ELSE 'Survived'
  END AS vital_status,
  -- both COUNTs see the same
  -- fanned-out rows
  COUNT(co.condition_occurrence_id)
    AS total_conditions,
  COUNT(de.drug_exposure_id)
    AS total_drugs
FROM cohort
LEFT JOIN condition_occurrence co
  ON cohort.person_id = co.person_id
LEFT JOIN drug_exposure de
  ON cohort.person_id = de.person_id
LEFT JOIN death dt
  ON cohort.person_id = dt.person_id
GROUP BY vital_status;
```

-- Survived: 218,913 | 218,913
-- Died: 49,552 | 49,552

(c) The SQL Malloy generates from (a)

```
WITH cohort AS (
  SELECT person_id
  FROM condition_occurrence
  WHERE condition_concept_id
    = 37311061
  GROUP BY 1)
SELECT
  CASE WHEN death.person_id IS NOT NULL
    THEN 'Died' ELSE 'Survived'
  END AS vital_status,
  COUNT(DISTINCT cohort.person_id)
    AS patients,
  -- same join as (b); each row ID
  -- is counted once
  COUNT(DISTINCT
    co.condition_occurrence_id)
    AS total_conditions,
  COUNT(DISTINCT de.drug_exposure_id)
    AS total_drugs
FROM cohort
LEFT JOIN person p
  ON p.person_id = cohort.person_id
LEFT JOIN condition_occurrence co
  ON p.person_id = co.person_id
LEFT JOIN drug_exposure de
  ON p.person_id = de.person_id
LEFT JOIN death
  ON p.person_id = death.person_id
GROUP BY 1;
```

Figure 1. The same question (conditions and drugs per outcome group) three ways. (a) The Malloy query against the shared model. (b) The naive SQL analysts reach for first: its two LEFT JOINS fan out, so in each group both COUNTs return the same inflated number. (c) The SQL that Malloy generates from (a), with table aliases shortened and file paths replaced by table names. It keeps the fan-out join but counts each table's distinct row IDs, which recovers the true totals (6,050 conditions and 37,729 drugs).

The size of this error is large and varies by group (Table 1). Across the cohort the naive join inflates the true condition count 44-fold and the drug count 7-fold; among decedents the condition count is inflated 72-fold. Because the inflation depends on how many condition and drug rows each patient happens to have, it cannot be eyeballed or corrected by a constant factor.

Table 1. True counts vs the naive join of Figure 1(b), COVID-19 cohort.

	True count (Malloy)	Naive join	Inflation
Conditions, survived (n=916)	5,361	218,913	41×
Conditions, died (n=48)	689	49,552	72×
Conditions, all	6,050	268,465	44×
Drug exposures, survived	34,460	218,913	6.4×
Drug exposures, died	3,269	49,552	15×
Drug exposures, all	37,729	268,465	7.1×

The same model drove the downstream clinical analyses with no new join logic. The cohort (916 survived, 48 died; index dates 2020-03-15 to 2021-09-14) showed the expected mortality gradient by age at diagnosis, rising from 0.0% (ages 0–17, n=216) to 17.0% (ages 75+, n=223), and reproduced the expected baseline contrast (Table 2). Switching the stratifier (age to race) or the index condition was a one-line change.

Table 2. Baseline characteristics of the COVID-19 cohort by vital status.

Characteristic	Survived (n=916)	Died (n=48)
Age at index, mean (SD), yr	44.6 (28.8)	81.5 (14.9)
Male, %	47.4	68.8
Age 75+, %	20.2	79.2
Age <18, %	23.6	0.0

Concept sets and CONCEPT_ANCESTOR

OHDSI phenotypes are built from concept sets, so the model joins every fact table to CONCEPT_ANCESTOR. Filtering on an ancestor ID selects every record filed under that concept. This matters in practice: no record in this dataset uses *Heart disease* (321588) directly, so an exact-ID filter finds 0 patients, while including descendants finds 2,207. An ATLAS concept set maps onto the model item by item. Items marked “include descendants” filter on the ancestor ID, other items filter on the concept ID, and “exclude” items are removed in a second stage. Drug ingredients work the same way through the RxNorm hierarchy: filtering on the ingredient enoxaparin collects all of its products, and shows that 43 of the 48 decedents received it, against 121 of the 916 survivors.

The software demonstration, *Cohort Studio*, is an HTML data app served by Malloy Publisher on top of this model. A user builds the index event by searching the vocabulary or pasting a concept set exported from ATLAS as JSON. They then set inclusion criteria and baseline and follow-up windows, and see attrition, mortality curves by age, baseline covariate balance (standardized mean differences), and treatments started in follow-up. Every control is a typed query parameter that Publisher binds on the server, so the page never writes query text from user input. A side panel shows, for each chart, the one-line Malloy query, the model view it names, and the SQL Publisher generated with the current settings.

Relation to OHDSI tools, and limitations

The semantic layer complements ATLAS and HADES; it does not replace them. ATLAS and its Circe compiler remain the reference for cohort generation: they write the COHORT table that FeatureExtraction, CohortMethod, and PatientLevelPrediction read, and network studies share cohort definitions as ATLAS JSON. The Malloy model sits beside that pipeline for characterization and exploratory analysis. Concept sets come in from ATLAS unchanged, and a site publishes the model once so analysts can query it from notebooks, dashboards, or data apps. This work has three limitations. We have run the model only on DuckDB with one synthetic CDM; running it on other databases and SQL dialects is designed for but not yet validated. Malloy’s SQL for Figure 1 keeps the fan-out join and removes duplicates with COUNT(DISTINCT), which can cost more on a large database than totalling each table first; we have not measured performance. Finally, Cohort Studio builds a simplified cohort (first diagnosis plus inclusion criteria) and does not reproduce Circe’s full cohort-era logic.

CONCLUSION

A semantic-modeling layer gives OMOP analysts one place to declare join cardinality, vocabulary, concept-set expansion, and reusable logic. The resulting analyses are shorter, read in clinical terms, and reuse cohort and baseline definitions. The fan-out example makes the correctness benefit concrete: a naive join inflated condition counts 44-fold across the cohort and 72-fold among decedents, while the same question asked through the model returned the true totals. Concept sets from ATLAS carry over directly through CONCEPT_ANCESTOR. For the OHDSI community this is a practical route to analyses that are easier to read, reuse, and check. Malloy, Malloy Publisher, and the OMOP sample are open source [5,6,7].

REFERENCES

1. Hripcsak G, Duke JD, Shah NH, et al. Observational Health Data Sciences and Informatics (OHDSI): opportunities for observational researchers. *Stud Health Technol Inform*. 2015;216:574-578.
2. Hripcsak G, Ryan PB, Duke JD, et al. Characterizing treatment pathways at scale using the OHDSI network. *Proc Natl Acad Sci U S A*. 2016;113(27):7329-7336.
3. Schuemie MJ, Ryan PB, Hripcsak G, Madigan D, Suchard MA. Improving reproducibility by using high-throughput observational studies with empirical calibration. *Philos Trans A Math Phys Eng Sci*. 2018;376(2128):20170356.
4. Walonoski J, Kramer M, Nichols J, et al. Synthea: an approach, method, and software mechanism for generating synthetic patients and the synthetic electronic health care record. *J Am Med Inform Assoc*. 2018;25(3):230-238.
5. Malloy: an open-source language for analyzing, transforming, and modeling data. <https://github.com/malloydata/malloy>
6. OMOP CDM Malloy sample, model, and notebooks. <https://github.com/malloydata/malloy-samples/tree/main/omop>. Example data: CDMConnector (darwin-eu), <https://darwin-eu.github.io/CDMConnector/>
7. Malloy Publisher: a server for Malloy semantic models and data apps. <https://github.com/malloydata/publisher>