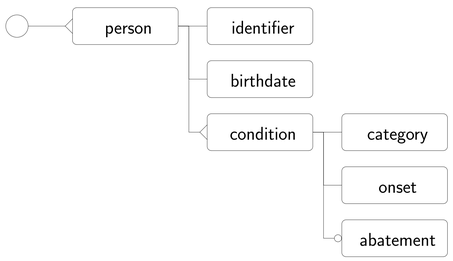
Query Combinators for OHDSI

Clark C. Evans & Kyrylo Simonov
Prometheus Research, LLC

Query Combinators are an algebra of query functions. This algebra's elements, or *queries*, represent relationships among class entities and datatypes. Query primitives are used to retrieve or navigate though a data source's records. These query primitives can be seen as paths in a tree.



We call a query's type its *signature*. Each query's signature specifies the type of input it expects and the type of output it produces. If more than one output value can be produced, we use the asterix; queries that produce optional output are indicated with a question mark.

Primitive	Signature
person	Database ⟶ Person*
identifier	Person ⟶ Integer
birthdate	Person ⟶ Date
condition	Person ⟶ Condition*
category	Condition ⟶ Text
onset	Condition ⟶ Date
abatement	Condition ⟶ Date?

In this algebra, *combinators* are operations on queries. They take queries for their arguments and build queries from them. For example, if person is a query, the count combinator can be used to build a query count(person). Each combinator has a rule which permits us to automatically compute the signature of the query it builds. In this case, the query signature for count(person) is Database ⟶ Integer.

$$\frac{f \quad A \to B^*}{\text{count}(f) \quad A \to \text{Integer}} \qquad \frac{\text{person} \quad \text{Database} \to \text{Person}^*}{\text{count}(\text{person}) \quad \text{Database} \to \text{Integer}}$$

To provide a path-like navigation, queries can be composed so long as their intermediate type is compatible. In this example, person.condition is build from two primitive queries.

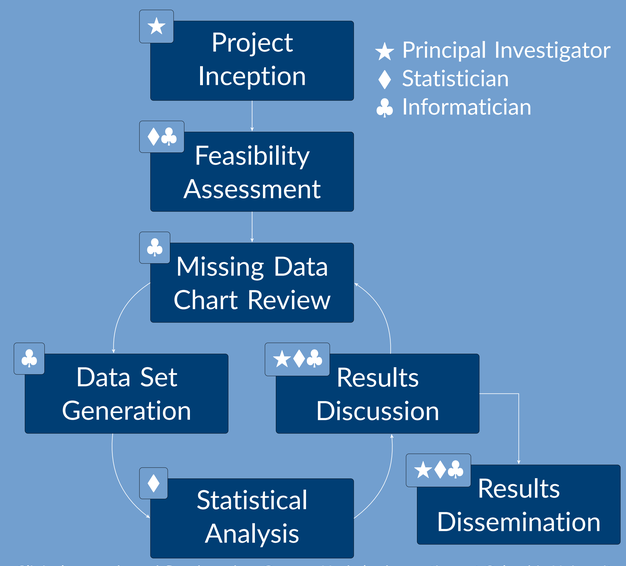
$$\frac{f \quad A \to B^* \qquad g \quad B \to C^*}{f.g \quad A \to C^*} \qquad \frac{\text{person} \quad \text{Database} \to \text{Person}^* \qquad \text{condition} \quad \text{Person} \to \text{Condition}^*}{\text{person.condition} \quad \text{Database} \to \text{Condition}^*}$$

Query expressions are algebraic. Since count and person.condition are defined, count(person.condition) is also a valid query in the algebra: it counts the number of condition records across all people in the database. This query's signature can be automatically computed as Database ⟶ Integer.

For more information about *Query Combinators* visit https://querycombinators.org. Our `DataKnots.jl` implementation is at https://github.com/rbt-lang/DataKnots.jl

a shared query environment that is usable by informaticians, statisticians, and clinicians

Currently, clinicians, statisticans, and informaticians use different tools and work in their own silos, sharing only limited datasets and analysis outputs.



Clinical research workflow based on Gregory Hruby's observations at Columbia University

Combinators can be designed to represent data sources, integrations, statistical measures and clinical concepts, defining a high-level query language meaningful to investigators and relevant to their research questions.

A query language tailored to OHDSI's *schema* and *operators* permits human readable-queries to be incrementally constructed, graphically displayed, and reviewed among the entire research team. An example cohort query using this language is shown to the right. Notice how data access primitives such as `person` and domain specific operations such as `during` make the relevant cohort selection logic flow without unnecessary detail.

For more information, see https://doi.org/10.1101/737619.

An OHDSI Cohort Query

This query is a translation from The Book Of OHDSI, §10.6 Defining a Cohort for Hypertension, *patients who initiate ACE inhibitors monotherapy as first-line treatments for hypertension*.

```
is_hypertensive = iscoded("SNOMED", 38341003)
is_hypertension_drug = iscoded("RxNorm", 149, 325646,
is_ace_inhibitor = iscoded("RxNorm", 18867, 1998, ...

person
keep(person => it)
first(drug_exposure.
      filter(concept.is_ace_inhibitor).
      sort(start_date))
keep(index_date => start_date)
filter(exists(
    person.observation_period.
       filter(includes(index_date.
                       and_previous(365days)))))
filter(exists(
    person.condition.filter(
        concept.is_hypertensive &&
        start_date.during(index_date.
                          and_previous(365days))))
filter(!exists(
    person.drug_exposure.filter(
      concept.is_hypertension_drug &&
      start_date < index_date)))
keep(combination_treatments =>
       person.drug_era.filter(
         concept.is_hypertension_drug &&
         start_date.during(index_date.
                           and_subsequent(7days))))
filter(1 == count(combination_treatments))
keep(custom_era =>
      person.drug_exposure.
      filter((concept.is_ace_inhibitor ||
              source_concept.is_ace_inhibitor) &&
              start_date >= index_date).
      record(start_date,
             end_date => coalesce(end_date,
                         start_date + days_supply,
                         start_date + 1days)).
      collapse_intervals(30days).
      first())
{ subject_id => person.person_id,
  cohort_enter_date => custom_era.start_date,
  cohort_exit_date => custom_era.end_date }
```

This working example is cohort #17707 in the repository https://github.com/rbt-lang/SynPUF-HCFU/.

OHDSI