DataKnots.jl

an extensible and coherent algebra of query combinators

Clark C. Evans ⟨cce@clarkevans.com⟩,
Kyrylo Simonov ⟨xi@resolvent.net⟩

Wed, 24th July, 2019
JuliaCon 2019

Prometheus Research, LLC

Clinical Research Workflow

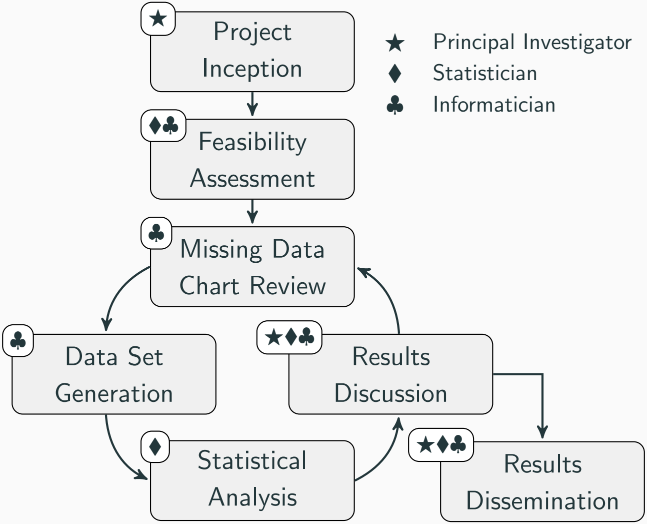


Figure 1: Clinical Research Workflow as inspired from Hruby's observations at Columbia University [**?**]

Current Practice: Multiple Query Approaches

Which anti-hypertensive medications are effective in improving blood pressure?

Principal Investigator

```
SELECT *
FROM patient
JOIN observation
ON (...)
WHERE ...
```

Informatician

```
d=read.csv(...)
...
a.lm=lm(...,data=d)
summary(a.lm)
    $r.squared
```

Statistician

An Observational Inquiry

Consider the inquiry, ''Which anti-hypertensive medications are effective in improving blood pressure?''. This inquiry could be operationalized as:

> *Within 6 months of a hypertension diagnosis, when an anti-hypertensive medication was added or intensified, was there a blood pressure decrease of 5 mmHg or more within 5 days after the medication adjustment?*

An Observational Inquiry

Consider the inquiry, "Which anti-hypertensive medications are effective in improving blood pressure?". This inquiry could be operationalized as:

> *Within 6 months of a* **hypertension diagnosis**, *when an* **anti-hypertensive medication** *was* **added or intensified**, *was there a* **blood pressure decrease** *of 5 mmHg or more within 5 days after the* **medication adjustment***?*

What are the query components?

The first thing to do is convert specialized vocabulary in this inquiry into query component definitions in a *query mediation* session.

Component	Mediation Notes
hypertension_diagnosis	exclude pregnancy & kidney failure
antihypertensive_medication	a product list is provided
added_or_intensified	new therapy or larger dose
blood_pressure_decrease	of both systolic & diastolic
medication_adjustment	change of daily medication
active_ingredient	normalize dosage records across compound products

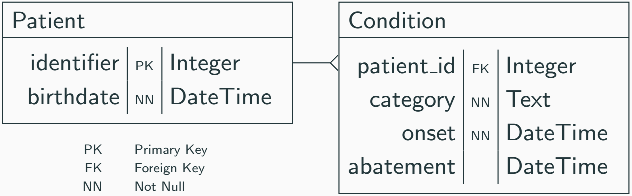
Table 1: Anti-hypertensive Query Components

Anti-Hypertensive Query

```
patient.keep(it)
antihypertensive_medication
active_ingredient
medication_adjustment
filter(added_or_intensified &&
    previous(6months).includes(
      patient.hypertension_diagnosis))
collect(is_effective =>
    subsequent(5days).includes(
      patient.blood_pressure_decrease(5mmHg)))
group(active_ingredient)
{ active_ingredient,
  count(medication_adjustment.filter(is_effective)),
  count(medication_adjustment.filter(not(is_effective))) }
```

Thinking in Query Combinators

Tabular Model of Clinical Research Data Repository



Patient		
identifier	PK	Integer
birthdate	NN	DateTime

Condition		
patient_id	FK	Integer
category	NN	Text
onset	NN	DateTime
abatement		DateTime

PK	Primary Key
FK	Foreign Key
NN	Not Null

Figure 2: Tabular Model for CRDR

Hierarchical Model of Clinical Research Data Repository

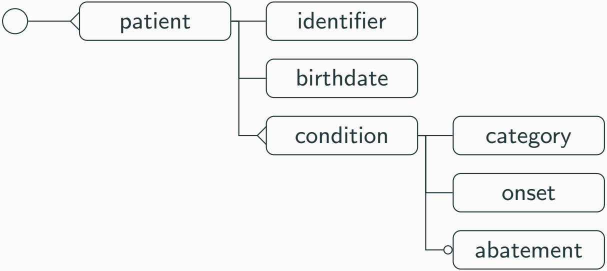
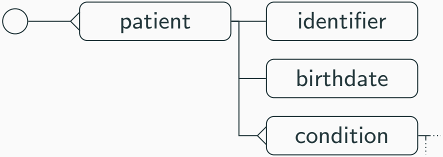


Figure 3: Hierarchical Model for CRDR

Example Queries



- patient
- count(patient)
- patient.condition
- patient.count(condition)
- mean(patient.count(condition))

Query Combinator Algebra

Query Combinators are an algebra of query functions.

- This algebra's elements, or *queries*, represent relationships among class entities and datatypes.
- This algebra's operations, or *combinators*, are applied to construct query expressions.

Query expressions, such as count(condition) are constructed by applying combinators, such as count to queries, such as condition.

Functional Model

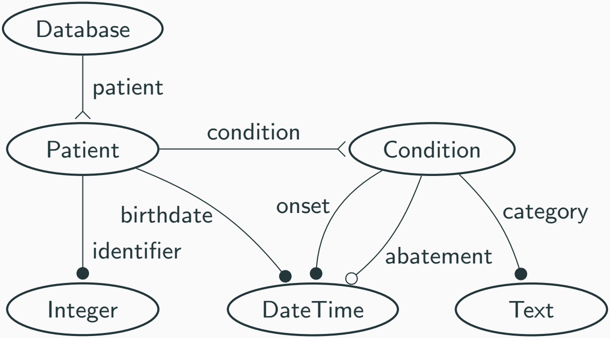


Figure 4: Functional Model for CRDR

Query Primitives

Primitive	Signature
patient	Database → Patient*
identifier	Patient → Integer
birthdate	Patient → DateTime
condition	Patient → Condition*
category	Condition → Text
onset	Condition → DateTime
abatement	Condition → DateTime$^?$

Table 2: Query Primitives for CRDR

The Count Combinator

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

$$\frac{\text{patient} \qquad \text{Database} \to \text{Patient}^*}{\text{count(patient)} \qquad \text{Database} \to \text{Integer}}$$

$$\frac{\text{condition} \qquad \text{Patient} \to \text{Condition}^*}{\text{count(condition)} \qquad \text{Patient} \to \text{Integer}}$$

The Composition Combinator

$$\begin{array}{rl} f & A \to B^* \\ g & B \to C^* \\ \hline f\,.g & A \to C^* \end{array}$$

$$\begin{array}{rl} \text{patient} & \text{Database} \to \text{Patient}^* \\ \text{condition} & \text{Patient} \to \text{Condition}^* \\ \hline \text{patient.condition} & \text{Database} \to \text{Condition}^* \end{array}$$

$$\begin{array}{rl} \text{condition} & \text{Patient} \to \text{Condition}^* \\ \text{category} & \text{Condition} \to \text{Text}^* \\ \hline \text{condition.category} & \text{Patient} \to \text{Text}^* \end{array}$$

Example: Feasibility Assessment

Suppose that an informatician would like to conduct a feasibility assessment to see if the CRDR database has at least some candidate patients relevant to this hypertension effectiveness inquiry.

> *How many patients, ages 18 or older, have an active diagnosis of Essential Hypertension?*

Components of Feasibility Assessment

How many patients, ages 18 or older, have an active diagnosis of *Essential Hypertension*?

Component	Definition
essential_hypertension	`'59621000'`
age	years(now() − birthdate)
has_active_diagnosis(x)	exists(condition.filter(category $= x$ && is_null(abatement)))

Table 3: Component Definitions for Feasibility Assessment

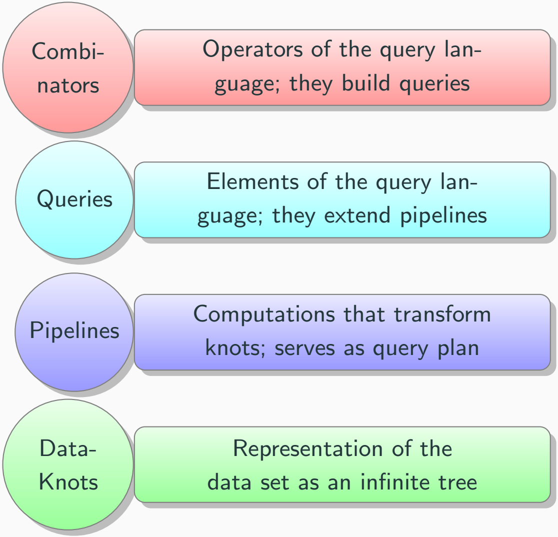
Adults /w Hypertension

How many patients, ages 18 or older, have an active diagnosis of Essential Hypertension?

```
patient
filter    (age >= 18  &&
              has_active_diagnosis(
                    essential_hypertension))
count()
```

Architecture & Julia Interface

DataKnots.jl has Four Conceptual Levels



Combinators: Operators of the query language; they build queries

Queries: Elements of the query language; they extend pipelines

Pipelines: Computations that transform knots; serves as query plan

DataKnots: Representation of the data set as an infinite tree

No Macros Required

The DataKnots macro syntax is completely optional. Primitive queries that navigate a data source can be constructed via `Get`.

- `Patient = Get(:PATIENT)`
- `Condition = Get(:CONDITION)`

In native Julia syntax, combinators like `Count` are functions that return queries. The $>>$ operator is overloaded for query composition.

- `Count(Patient)`
- `Patient >> Condition`
- `Patient >> Count(Condition)`

Automatic Lifting of Functions

Constants can be lifted as primitive queries that produce a constant result. Functions can be lifted to combinators. Vectors are lifted to queries returning plural results.

- `Lift("hello world")`
- `titlecase.(Lift("hello world"))`
- `Lift(1:3)`

Functions taking vector arguments are lifted to aggregate combinators. Functions returning vectors are lifted to queries returning plural results.

- `mean.(Patient >> Count(Condition))`

Novel Queries & Combinators

Novel primitive queries, that access new data sources, such as web resources or specific data sources such as FHIR or HDF5, can be written in Julia using the DataKnots and Pipelines APIs.

Novel data transformations, that cannot be simply lifted from julia functions, can also be written using Julia to extend the query language. There are many functions, such as `Group` and the like which simply cannot be lifted.

The query plan, or data pipeline view, of a query can be shown to see how it would perform at an implementation level.

See DataKnots.jl on GitHub

There is an implementation of Query Combinators for the Julia Language, called `DataKnots.jl`.

- this implementation is MIT/Apache licensed
- it includes an in-memory, column-oriented database
- it has adapters to CSV (and soon XML, JSON)
- essential query operators are implemented
- Julia statistics can be *lifted* to a combinator
- an adapter to SQL datasources is in progress!

https://github.com/rbt-lang/DataKnots.jl