codeq: Making Git Repos Smarter

@stuarthalloway

Where Are We?

Motivation

Schema

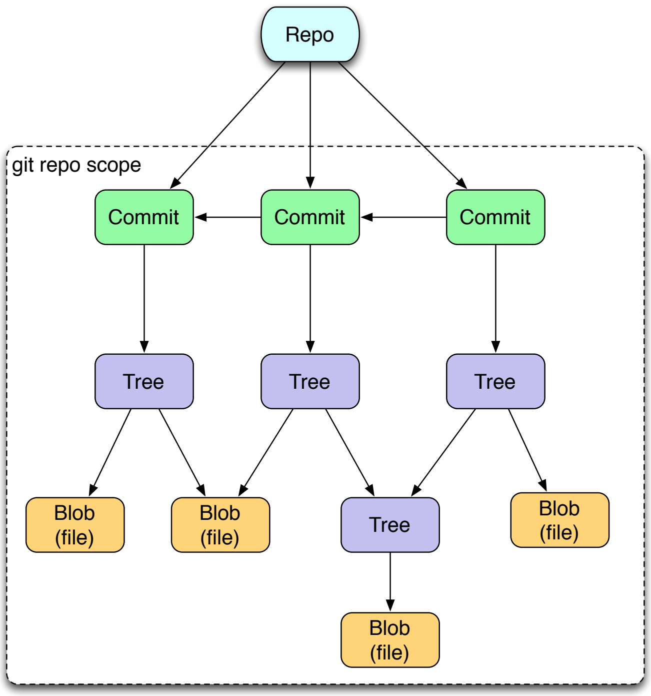
Import

Datalog

Rules

Queries

Git is a Database



Flexible

Immutable Values

Time Aware

Trees

Content Addressing

Opportunity 1: API

How many commits?

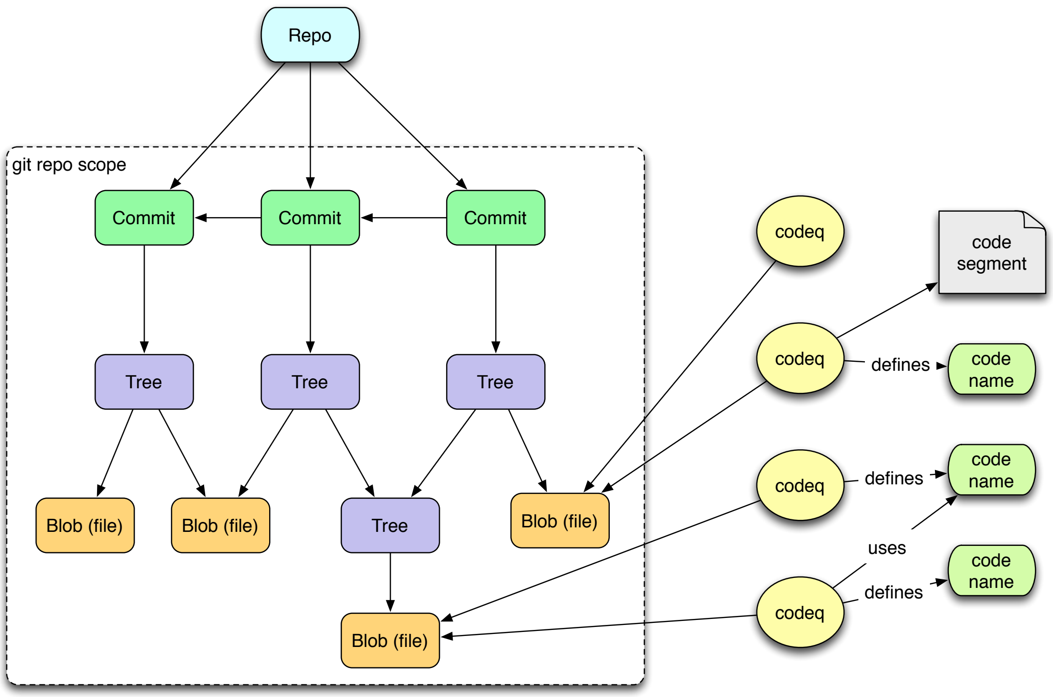
```
>git log -format=oneline | wc -l
1590
```

Opportunity 1: API

How many commits?

```
>SELECT COUNT(*) FROM COMMITS;
1590
```

Opportunity 2: Model



codeq

Foundation Schema

Import Phase (Indexes What Git Knows)

Analysis Phase (Indexes What Tools Know)

Pluggable Analyzers

Datalog Query

Why Datomic?

Power: Datalog Queries

Flexibility: Universal Relation

Immutability: Matches Git Semantics

Extensibility: Add Capabilities Using Java*

Free

Where Are We?

Motivation

Schema

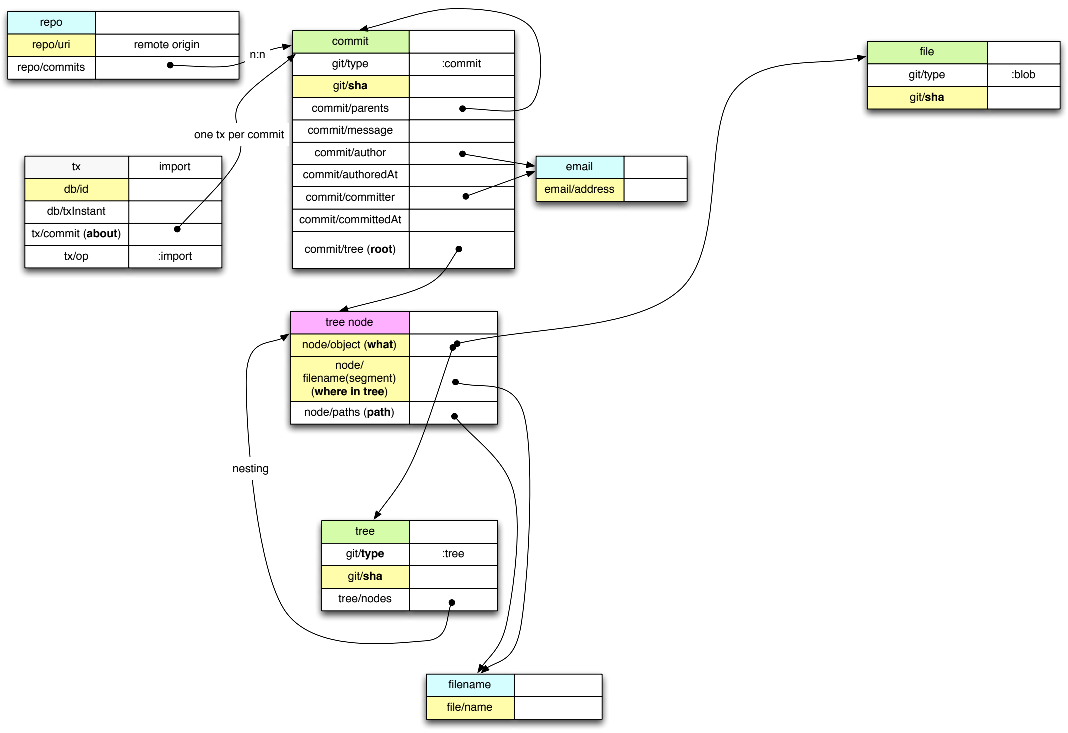
Import

Datalog

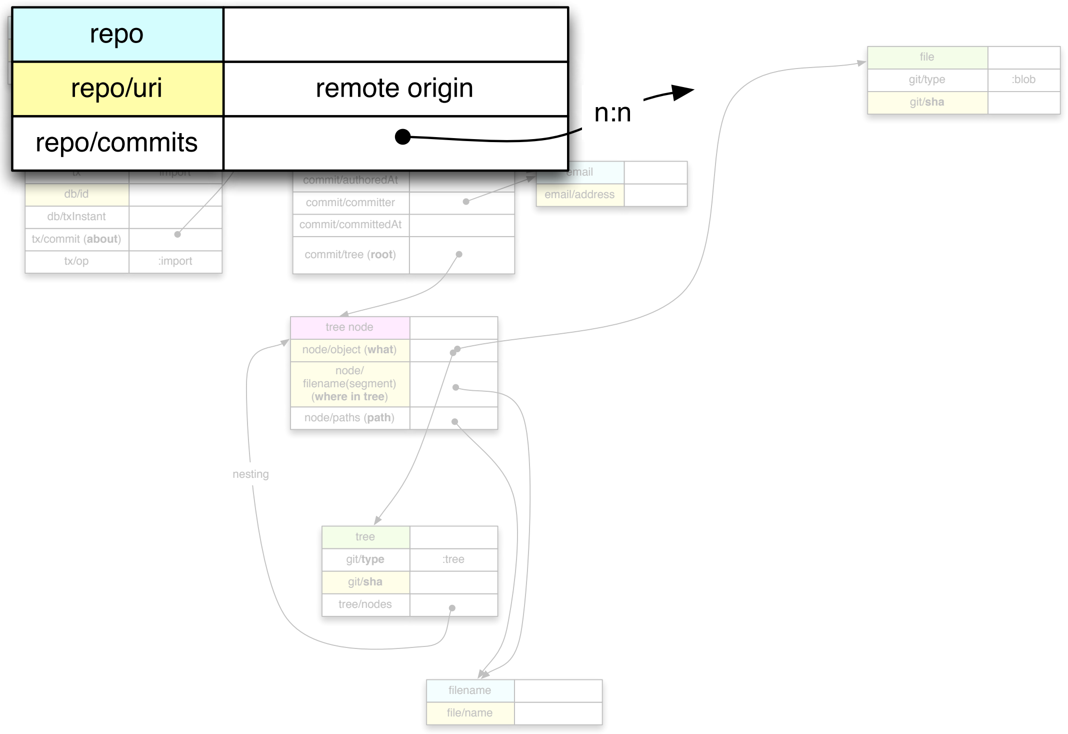
Rules

Queries

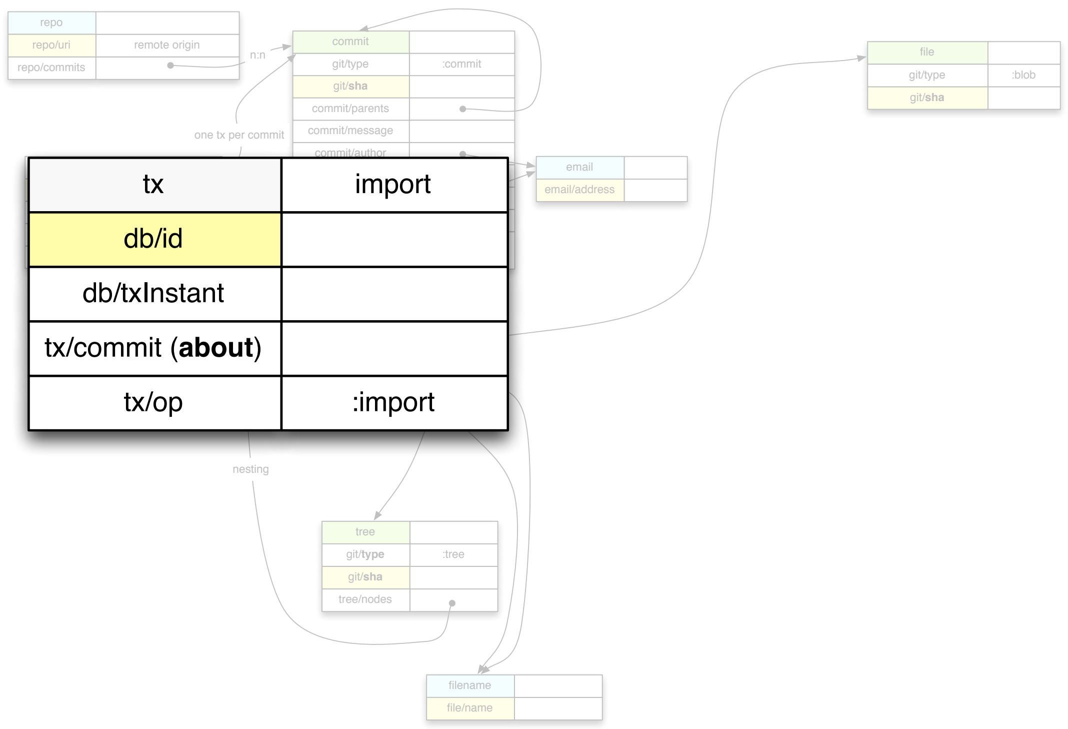
Schema (Import Subset)



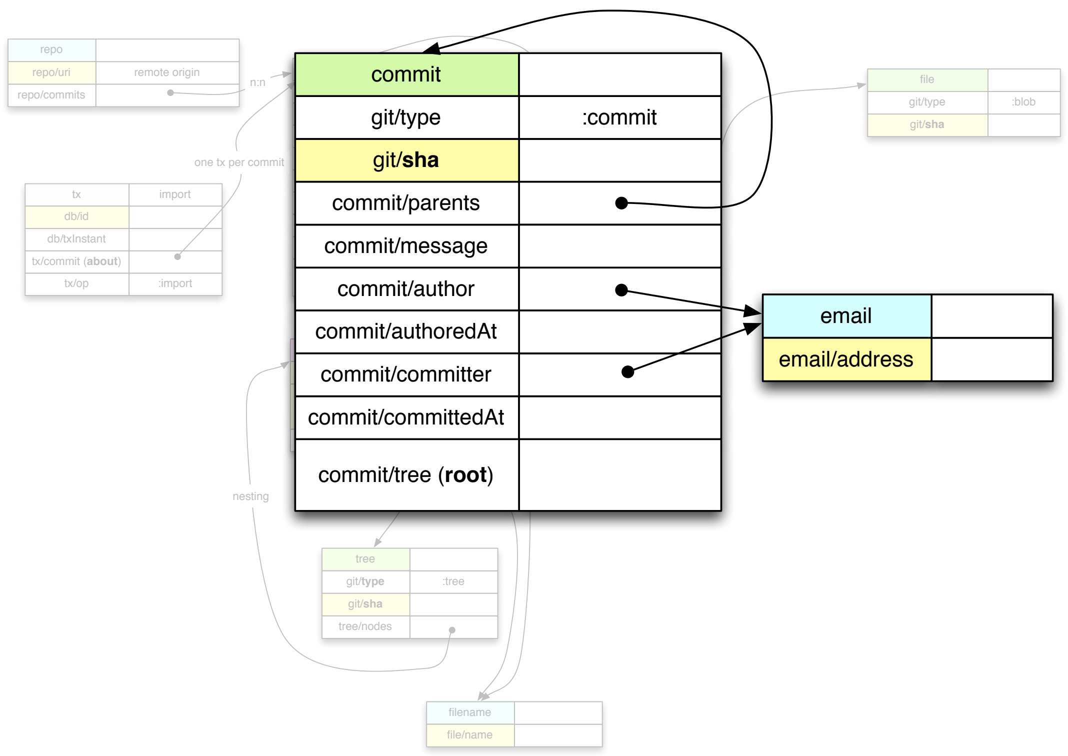
Schema (Import Subset)



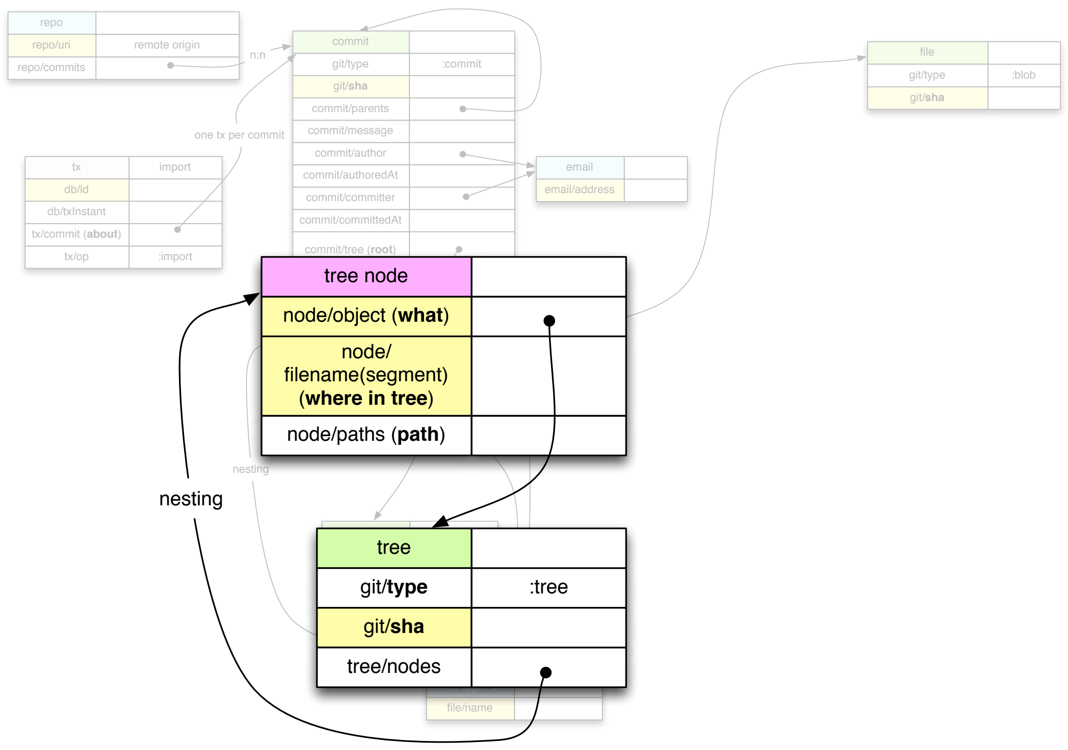
Schema (Import Subset)



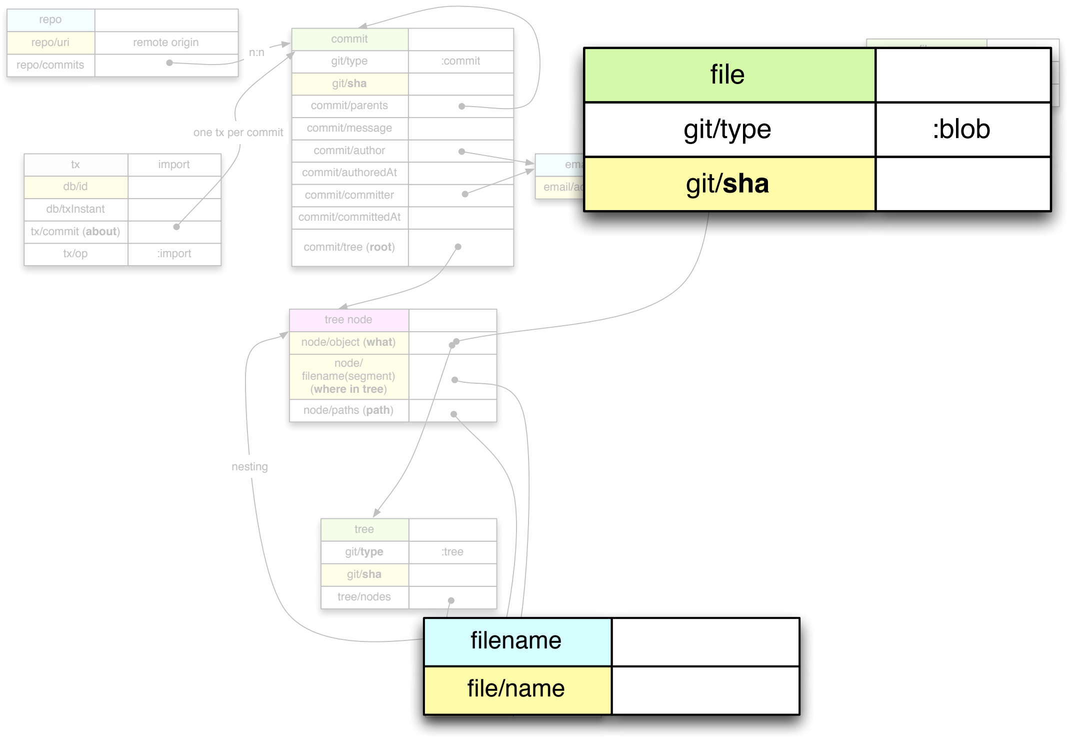
Schema (Import Subset)



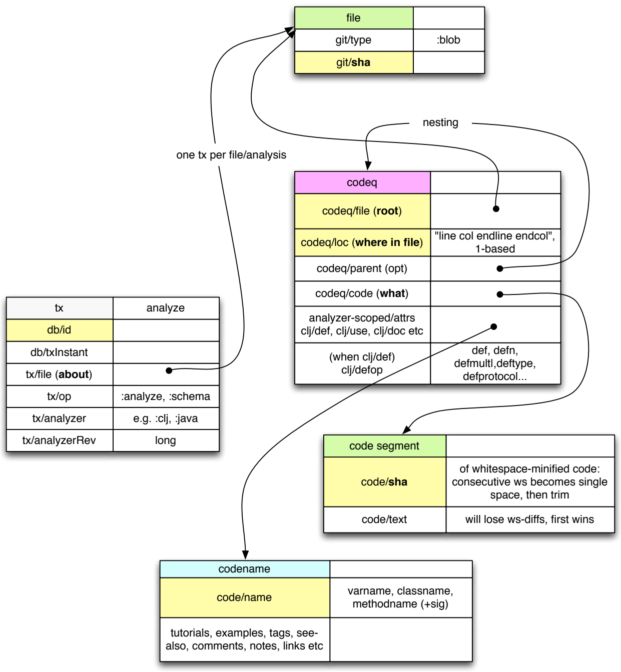
Schema (Import Subset)



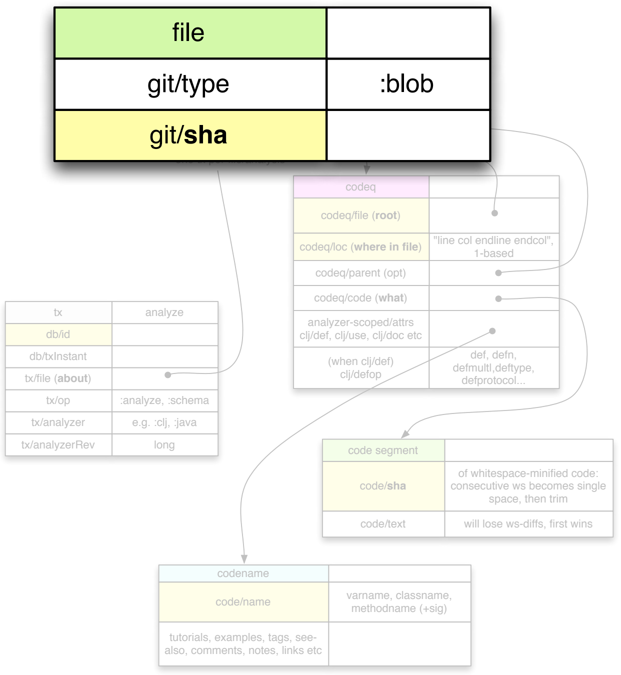
Schema (Import Subset)



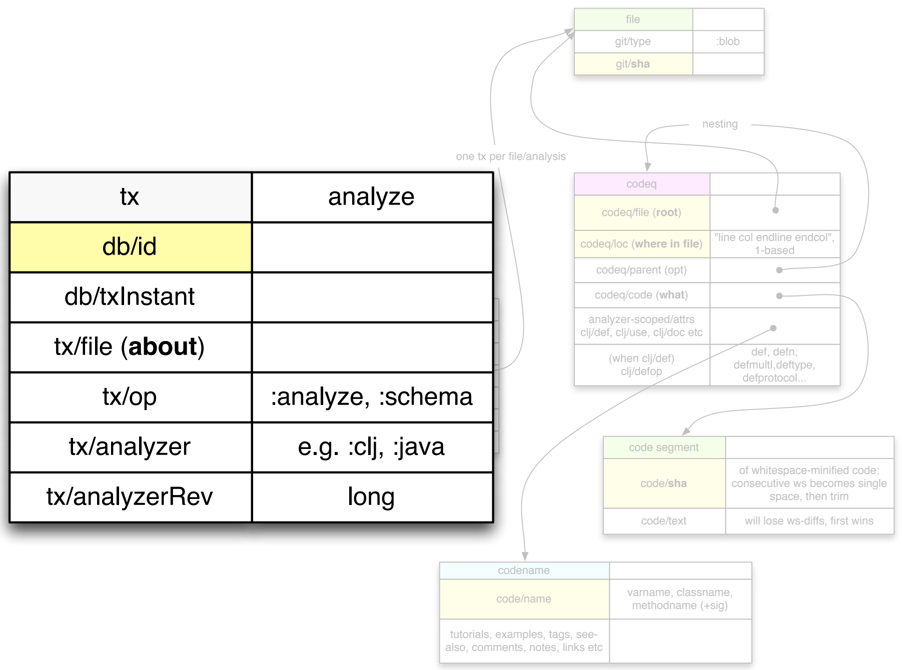
Schema (Analysis Subset)


file
git/type
:blob
git/**sha**
nesting
one tx per file/analysis
codeq
codeq/file (**root**)
codeq/loc (**where in file**)
"line col endline endcol", 1-based
codeq/parent (opt)
codeq/code (**what**)
analyzer-scoped/attrs clj/def, clj/use, clj/doc etc
(when clj/def) clj/defop
def, defn, defmultl,deftype, defprotocol...
tx
analyze
db/id
db/txInstant
tx/file (**about**)
tx/op
:analyze, :schema
tx/analyzer
e.g. :clj, :java
tx/analyzerRev
long
code segment
code/**sha**
of whitespace-minified code: consecutive ws becomes single space, then trim
code/text
will lose ws-diffs, first wins
codename
code/name
varname, classname, methodname (+sig)
tutorials, examples, tags, see-also, comments, notes, links etc

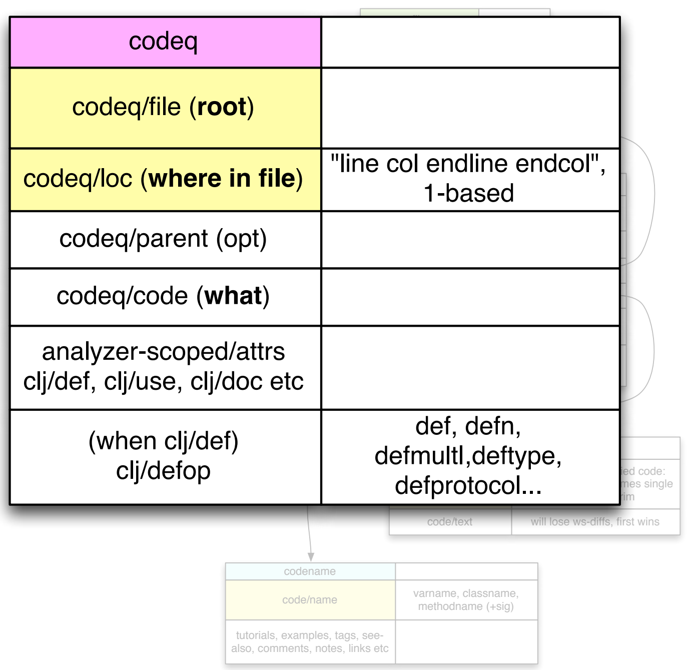
Schema (Analysis Subset)


file
git/type
:blob
git/sha
codeq
codeq/file (root)
codeq/loc (where in file)
"line col endline endcol", 1-based
codeq/parent (opt)
codeq/code (what)
analyzer-scoped/attrs clj/def, clj/use, clj/doc etc
(when clj/def) clj/defop
def, defn, defmulti,deftype, defprotocol...
tx
analyze
db/id
db/txInstant
tx/file (about)
tx/op
:analyze, :schema
tx/analyzer
e.g. :clj, :java
tx/analyzerRev
long
code segment
code/sha
of whitespace-minified code: consecutive ws becomes single space, then trim
code/text
will lose ws-diffs, first wins
codename
code/name
varname, classname, methodname (+sig)
tutorials, examples, tags, see-also, comments, notes, links etc

Schema (Analysis Subset)

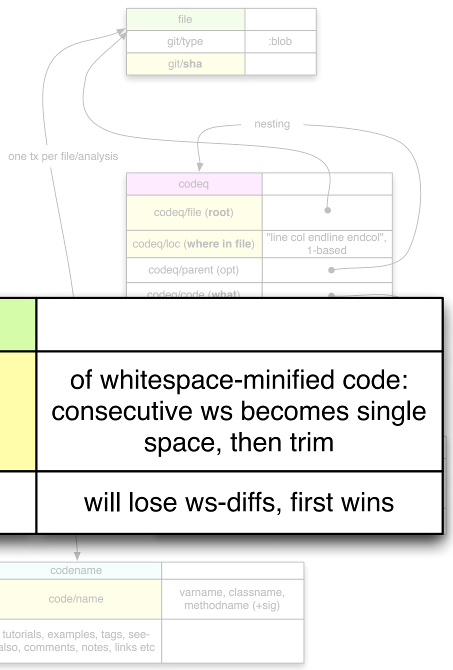

file
git/type
:blob
git/sha
nesting
one tx per file/analysis
tx
analyze
db/id
db/txInstant
tx/file (about)
tx/op
:analyze, :schema
tx/analyzer
e.g. :clj, :java
tx/analyzerRev
long
codeq
codeq/file (root)
codeq/loc (where in file)
"line col endline endcol", 1-based
codeq/parent (opt)
codeq/code (what)
analyzer-scoped/attrs clj/def, clj/use, clj/doc etc
(when clj/def) clj/defop
def, defn, defmulti,deftype, defprotocol...
code segment
code/sha
of whitespace-minified code: consecutive ws becomes single space, then trim
code/text
will lose ws-diffs, first wins
codename
code/name
varname, classname, methodname (+sig)
tutorials, examples, tags, see-also, comments, notes, links etc

Schema (Analysis Subset)


codename
code/name
varname, classname, methodname (+sig)
tutorials, examples, tags, see-also, comments, notes, links etc
code/text
will lose ws-diffs, first wins

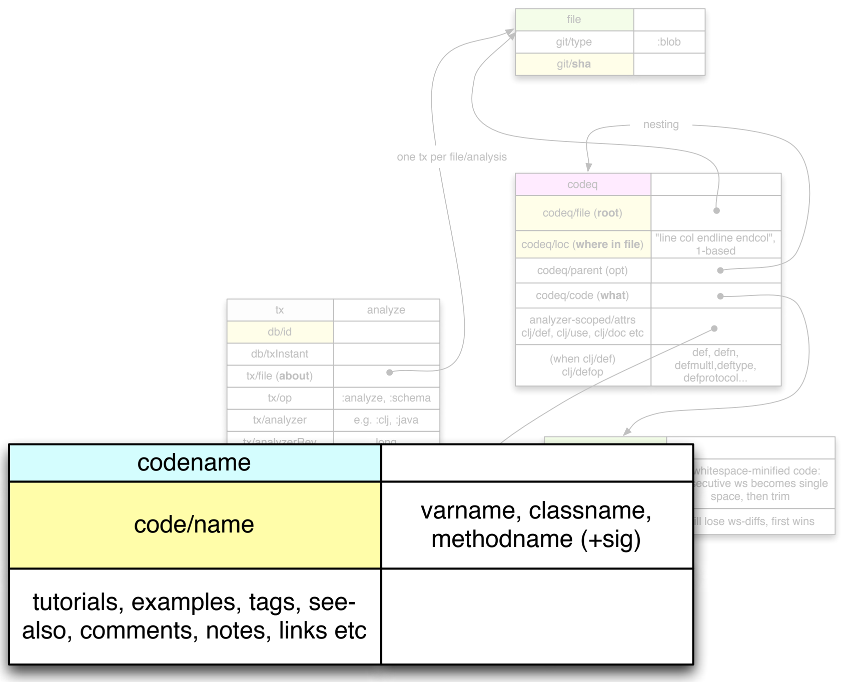
codeq	
codeq/file (**root**)	
codeq/loc (**where in file**)	"line col endline endcol", 1-based
codeq/parent (opt)	
codeq/code (**what**)	
analyzer-scoped/attrs clj/def, clj/use, clj/doc etc	
(when clj/def) clj/defop	def, defn, defmultl,deftype, defprotocol...

Schema (Analysis Subset)

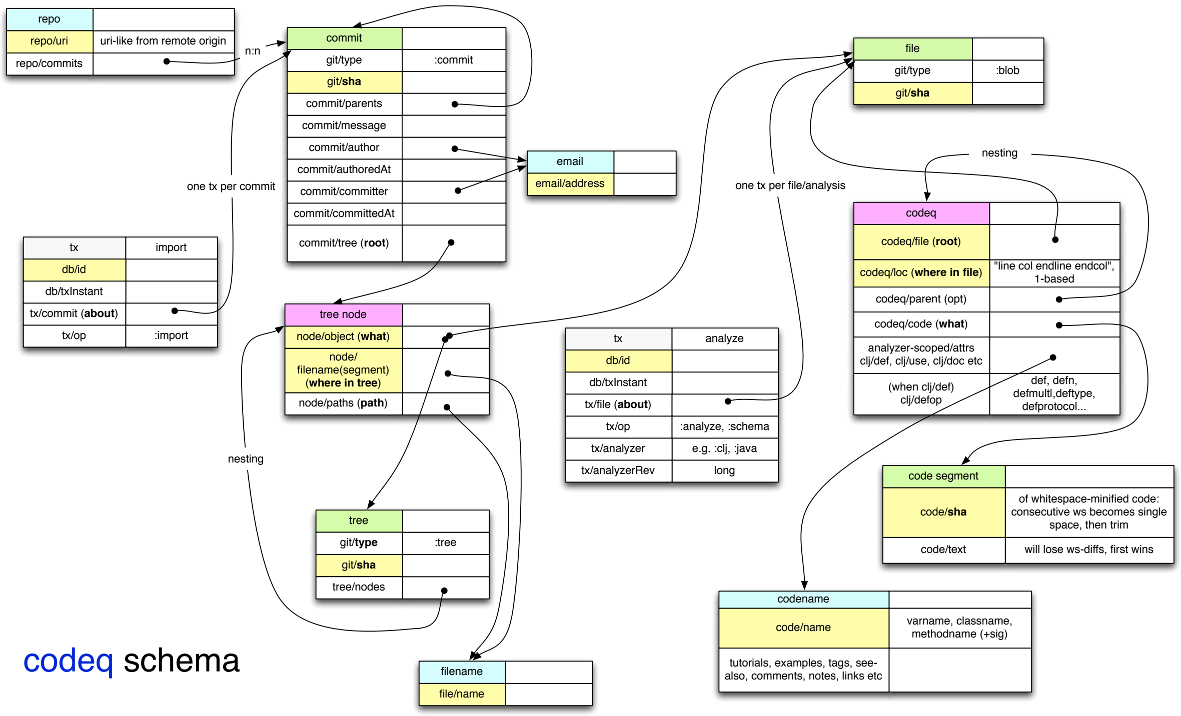

file
git/type
:blob
git/sha
nesting
one tx per file/analysis
codeq
codeq/file (root)
codeq/loc (where in file)
"line col endline endcol", 1-based
codeq/parent (opt)
codeq/code (what)
codename
code/name
varname, classname, methodname (+sig)
tutorials, examples, tags, see-also, comments, notes, links etc

code segment	
code/**sha**	of whitespace-minified code: consecutive ws becomes single space, then trim
code/text	will lose ws-diffs, first wins

Schema (Analysis Subset)



file
git/type
:blob
git/sha
nesting
one tx per file/analysis
codeq
codeq/file (root)
codeq/loc (where in file)
"line col endline endcol", 1-based
codeq/parent (opt)
codeq/code (what)
analyzer-scoped/attrs clj/def, clj/use, clj/doc etc
(when clj/def) clj/defop
def, defn, defmulti, deftype, defprotocol...
tx
analyze
db/id
db/txInstant
tx/file (about)
tx/op
:analyze, :schema
tx/analyzer
e.g. :clj, :java
tx/analyzerRev
long
whitespace-minified code: consecutive ws becomes single space, then trim
will lose ws-diffs, first wins
codename
code/name
varname, classname, methodname (+sig)
tutorials, examples, tags, see-also, comments, notes, links etc



codeq schema

Where Are We?

Motivation

Schema

Import

Datalog

Rules

Queries

java -jar codeq.jar datomic:free://localhost:4334/junit-1

```
Importing repo: git@github.com:junit-team/junit.git as: junit
Adding repo git@github.com:junit-team/junit.git
Importing commit: b6a0693454ac8ded32b3a1ea7c859c5a840169dc
Importing commit: aac9e722a36626e98794748c894cf3db3b24f4eb
 ...
Importing commit: 2a010a89464d9879a740fc611a004a6c15ae6ed1
Import complete!

Analyzing...
Running analyzer: :clj on [.clj]
Running analyzer: :java on [.java]
analyzing file: 17592186045530  - sha:  10b5045c7d23d20775eb20523d615e94277eaa19
analyzing file: 17592186045534  - sha:  dcd205011ab311610cbddf76d6f30bb4b78a23a5
 ...
analyzing file: 17592186085759  - sha:  ea793ff8db6451dfd02d0d89ca73f615bf6ca386
Analysis complete!
```

Where Are We?

Motivation

Schema

Import

Datalog

Rules

Queries

query anatomy

query anatomy

```
q([:find ...
   :in ...
   :where ...],
   input1,
   ...
   inputN);
```

query anatomy

constraints

```
q([:find ...
   :in ...
   :where ...],
   input1,
   ...
   inputN);
```

query anatomy

```
q([:find ...
   :in ...
   :where ...],
   input1,
   ...          ←—— inputs
   inputN);
```

query anatomy

```
q([:find ...
   :in ...       ←—— names for
                     inputs
   :where ...],
   input1,
   ...
   inputN);
```

query anatomy

```
q([:find ...
   :in ...
   :where ...],
   input1,
   ...
   inputN);
```

variables to
return

variables

```
?customer
                ?product

       ?orderId

                  ?email
```

constants

```
42                        :email

             “john”

:order/id

              #inst "2012-02-29"
```

keywords

```
42                        :email

             “john”

:order/id

              #inst "2012-02-29"
```

namespaces

```
                                            :email
42

                      “john”

:order/id

                        #inst "2012-02-29"
```

extensible reader

```
                                            :email
42

                      “john”

:order/id

                        #inst "2012-02-29"
```

data patterns

example database

entity	attribute	value
42	:email	jdoe@example.com
43	:email	jane@example.com
42	:orders	107
42	:orders	141

data pattern

Constrains the results returned,
binds variables

```
[?customer :email ?email]
```

data pattern

Constrains the results returned,
binds variables

```
[?customer :email ?email]

     ↑           ↑          ↑

   entity    attribute    value
```

data pattern

Constrains the results returned,
binds variables

constant

```
[?customer :email ?email]
```

data pattern

Constrains the results returned,
binds variables

variable variable

```
[?customer :email ?email]
```

entity	attribute	value
42	:email	jdoe@example.com
43	:email	jane@example.com
42	:orders	107
42	:orders	141

```
[?customer :email ?email]
```

constants anywhere

"Find a particular customer's email"

```
[42 :email ?email]
```

entity	attribute	value
42	:email	jdoe@example.com
43	:email	jane@example.com
42	:orders	107
42	:orders	141

```
[42 :email ?email]
```

variables anywhere

“What attributes does customer 42 have?

```
[42 ?attribute]
```

entity	attribute	value
42	:email	jdoe@example.com
43	:email	jane@example.com
42	:orders	107
42	:orders	141

```
[42 ?attribute]
```

variables anywhere

“What attributes and values does customer 42 have?

```
[42 ?attribute ?value]
```

entity	attribute	value
42	:email	jdoe@example.com
43	:email	jane@example.com
42	:orders	107
42	:orders	141

```
[42 ?attribute ?value]
```

basic usage

:where clause

data pattern

```
[:find ?customer
 :where [?customer :email]]
```

:find clause

variable to return

```
[:find ?customer
 :where [?customer :email]]
```

implicit join

"Find all the customers who have placed orders."

```
[:find ?customer
 :where [?customer :email]
        [?customer :orders]]
```

API

```
import static datomic.Peer.q;


q("[:find ?customer
    :where [?customer :id]
           [?customer :orders]]",
  db);
```

q

```
import static datomic.Peer.q;

q("[:find ?customer
    :where [?customer :id]
           [?customer :orders]]",
   db);
```

query

```
import static datomic.Peer.q;

q("[:find ?customer
    :where [?customer :id]
           [?customer :orders]]",
   db);
```

input(s)

```
import static datomic.Peer.q;


q("[:find ?customer
    :where [?customer :id]
           [?customer :orders]]",
   db);
```

:in clause

Names inputs so you can refer to them elsewhere in the query

```
:in $database ?email
```

parameterized query

"Find a customer by email."

```
q([:find ?customer
   :in $database ?email
   :where [$database ?customer :email ?email]],
   db,
   "jdoe@example.com");
```

first input

"Find a customer by email."

```
q([:find ?customer
   :in $database ?email
   :where [$database ?customer :email ?email]],
   db,
   "jdoe@example.com");
```

second input

“Find a customer by email.”

```
q([:find ?customer
   :in $database ?email
   :where [$database ?customer :email ?email]],
   db,
   "jdoe@example.com");
```

verbose?

“Find a customer by email.”

```
q([:find ?customer
   :in $database ?email
   :where [$database ?customer :email ?email]],
   db,
   "jdoe@example.com");
```

shortest name possible

“Find a customer by email.”

```
q([:find ?customer
   :in $ ?email
   :where [$ ?customer :email ?email]],
   db,
   "jdoe@example.com");
```

elide $ in :where

“Find a customer by email.”

```
q([:find ?customer
   :in $ ?email
   :where [ ?customer :email ?email]],
   db,
   "jdoe@example.com");
```

no need to
specify $

extending query

predicates

Functional constraints that can appear in a :where clause

```
[(< 50 ?price)]
```

adding a predicate

"Find the expensive items"

```
[:find ?item
 :where [?item :item/price ?price]
        [(< 50 ?price)]]
```

functions

Take bound variables as inputs
and bind variables with output

```
[(shipping ?zip ?weight) ?cost]
```

function args

```
[(shipping ?zip ?weight) ?cost]
```

$\uparrow$

bound inputs

function returns

```
[(shipping ?zip ?weight) ?cost]
```

$\uparrow$

bind return values

putting it all together

"Find me the customer/product combinations where the shipping cost dominates the product cost."

```
[:find ?customer ?product
 :where [?customer :shipAddress ?addr]
        [?addr :zip ?zip]
        [?product :product/weight ?weight]
        [?product :product/price ?price]
        [(Shipping/estimate ?zip ?weight) ?shipCost]
        [(<= ?price ?shipCost)]]
```

putting it all together

"Find me the customer/product combinations where the shipping cost dominates the product cost."

navigate from customer to zip

```
[:find ?customer ?product
 :where [?customer :shipAddress ?addr]
        [?addr :zip ?zip]
        [?product :product/weight ?weight]
        [?product :product/price ?price]
        [(Shipping/estimate ?zip ?weight) ?shipCost]
        [(<= ?price ?shipCost)]]
```

putting it all together

"Find me the customer/product combinations where the shipping cost dominates the product cost."

get product facts needed *during query*

```
[:find ?customer ?product
 :where [?customer :shipAddress ?addr]
        [?addr :zip ?zip]
        [?product :product/weight ?weight]
        [?product :product/price ?price]
        [(Shipping/estimate ?zip ?weight) ?shipCost]
        [(<= ?price ?shipCost)]]
```

putting it all together

"Find me the customer/product combinations where the shipping cost dominates the product cost."

call web service to bind shipCost

```
[:find ?customer ?product
 :where [?customer :shipAddress ?addr]
        [?addr :zip ?zip]
        [?product :product/weight ?weight]
        [?product :product/price ?price]
        [(Shipping/estimate ?zip ?weight) ?shipCost]
        [(<= ?price ?shipCost)]]
```

byo functions

Functions can be plain JVM code.

```
public class Shipping {
  public static BigDecimal
  estimate(String zip1, int pounds);
}
```

putting it all together

“Find me the customer/product combinations where the shipping cost dominates the product cost.”

```
[:find ?customer ?product
 :where [?customer :shipAddress ?addr]
        [?addr :zip ?zip]
        [?product :product/weight ?weight]
        [?product :product/price ?price]
        [(Shipping/estimate ?zip ?weight) ?shipCost]
        [(<= ?price ?shipCost)]]
```

constrain price

putting it all together

"Find me the customer/product combinations where the shipping cost dominates the product cost."

return customer, product pairs

```
[:find ?customer ?product
 :where [?customer :shipAddress ?addr]
        [?addr :zip ?zip]
        [?product :product/weight ?weight]
        [?product :product/price ?price]
        [(Shipping/estimate ?zip ?weight) ?shipCost]
        [(<= ?price ?shipCost)]]
```

Where Are We?

Motivation

Schema

Import

Datalog

Rules

Queries

rule clause

"Products are related if they have a common category."

```
[(relatedProduct ?p1 ?p2)
 [?p1 :category ?c]
 [?p2 :category ?c]
 [(!= ?p1 ?p2)]]
```

rule head

"Products are related if they have a common category."

this is true...

```
[(relatedProduct ?p1 ?p2)
 [?p1 :category ?c]
 [?p2 :category ?c]
 [(!= ?p1 ?p2)]]
```

rule body

“Products are related if they have a common category.”

```
[(relatedProduct ?p1 ?p2)
 [?p1 :category ?c]
 [?p2 :category ?c]
 [(!= ?p1 ?p2)]]
```

...if all these are true

rule inputs

“Find all products related to expensive chocolate.”

```
q("[:find ?p2
    :in $ %
    :where (expensiveChocolate p1)
           (relatedProduct p1 p2)]",
  db,
  rules)
```

rules are a kind of input

naming rule inputs

“Find all products related to expensive chocolate.”

```
q("[:find ?p2
    :in $ %
    :where (expensiveChocolate p1)
           (relatedProduct p1 p2)]",
  db,
  rules)
```

rule names begin with %

using rule patterns

“Find all products related to expensive chocolate.”

```
q("[:find ?p2
    :in $ %
    :where (expensiveChocolate p1)
           (relatedProduct p1 p2)]",
  db,
  rules)
```

rule patterns can appear in :where clause

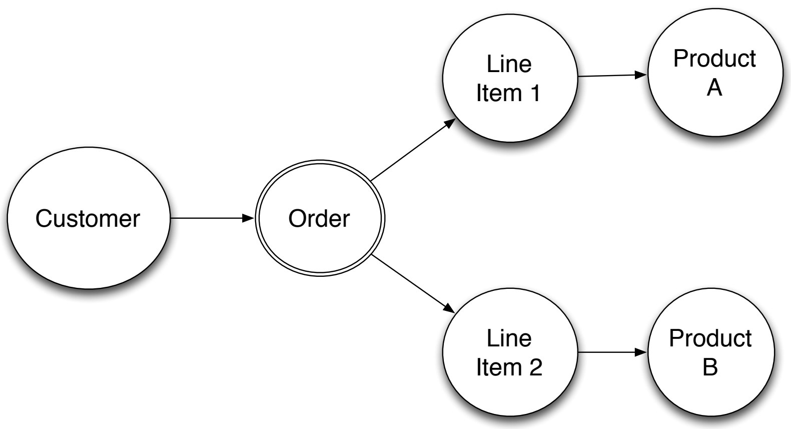
implicit or

"Products are related if they have the same category, or they have appeared in the same order."

```
[[(relatedProduct ?p1 ?p2)
  [?p1 :category ?c]
  [?p2 :category ?c]
  [(!= ?p1 ?p2)]]
 [(relatedProduct ?p1 ?p2)
  [?o :order/item ?item1]
  [?item1 :order/product ?p1]
  [?o :order/item ?item2]
  [?item2 :order/product ?p2]
  [(!= ?p1 ?p2)]]]
```

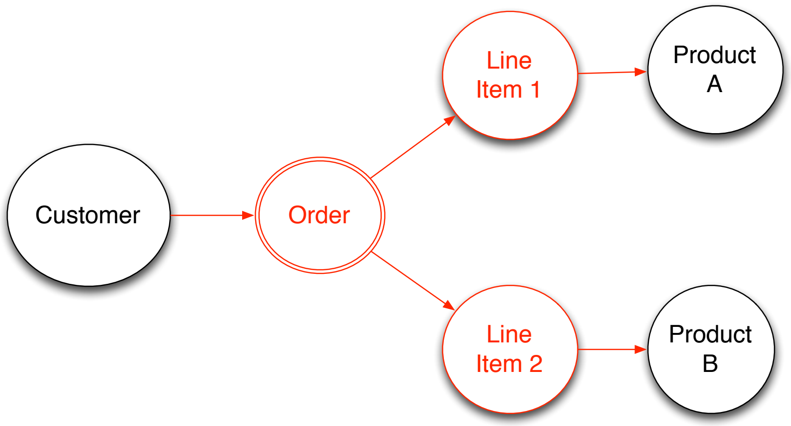
extent

Get "the whole order".



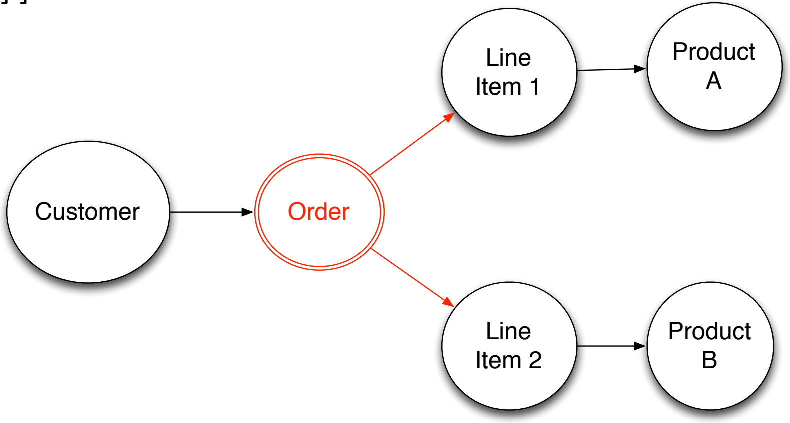
extent

Get "the whole order".



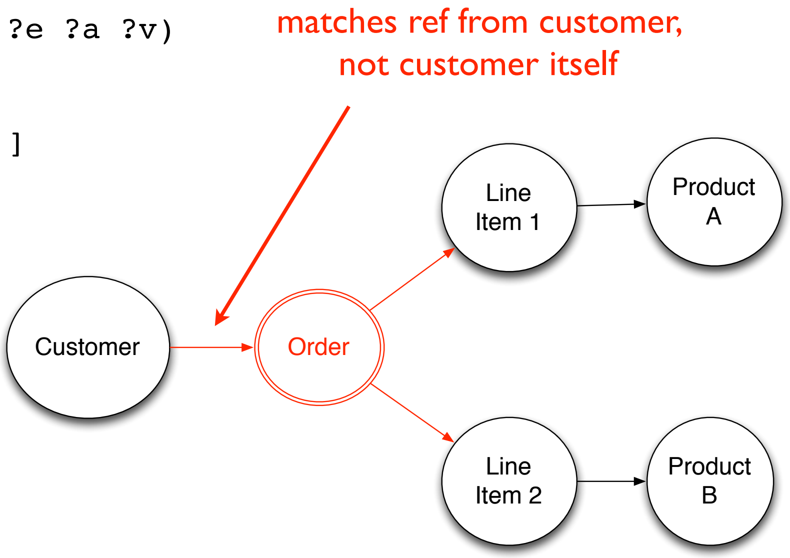
my direct references

```
[(extent ?x ?e ?a ?v)
 (?e ?a ?v)
 (?x ?a ?v)
 [(= ?e ?x)]]
```



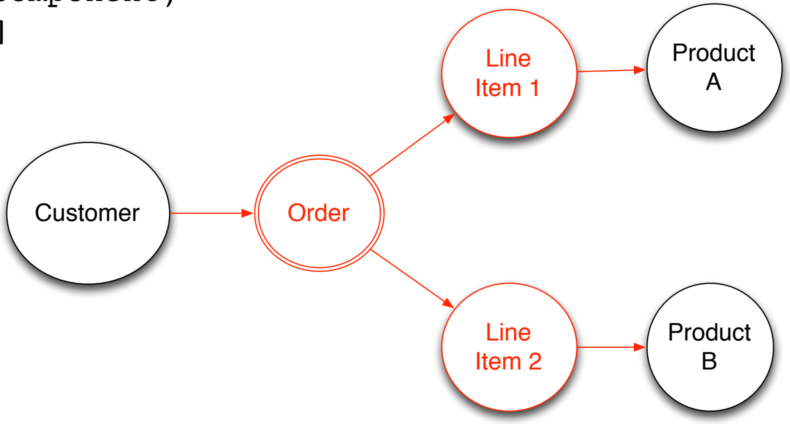
direct references to me

```
[(extent ?x ?e ?a ?v)
 (?e ?a ?v)
 (?e ?a ?x)
 [(= ?v ?x)]]
```



recurse *component* attributes

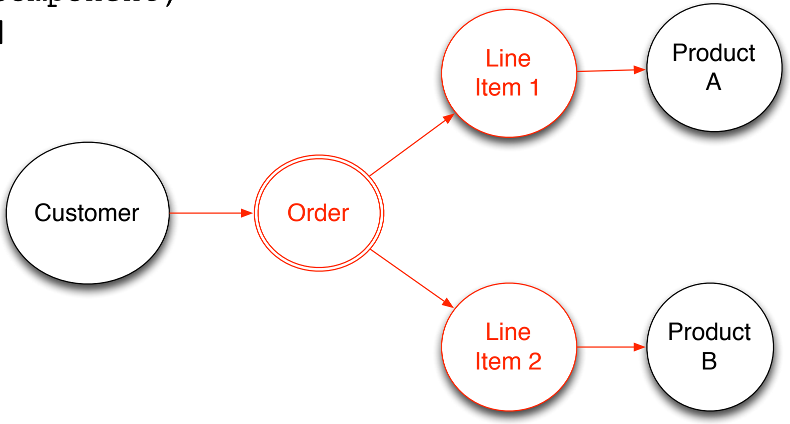
```
[(extent ?x ?e ?a ?v)
 (components ?x ?y)
 (extent ?y ?e ?a ?v)]
[(components ?p ?c)
 (?a :db/isComponent)
 (?p ?a ?c)]
```



recurse *component* attributes

```
[(extent ?x ?e ?a ?v)
 (components ?x ?y)
 (extent ?y ?e ?a ?v)]
[(components ?p ?c)
 (?a :db/isComponent)
 (?p ?a ?c)]
```

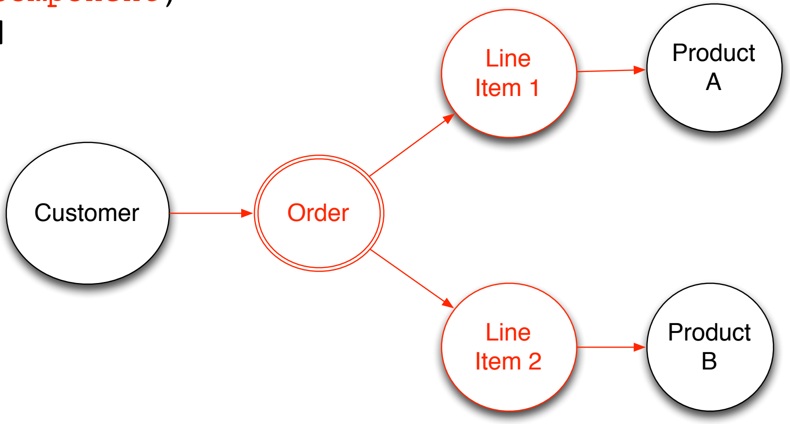
recursive definition



recurse *component* attributes

```
[(extent ?x ?e ?a ?v)
 (components ?x ?y)
 (extent ?y ?e ?a ?v)]
[(components ?p ?c)
 (?a :db/isComponent)
 (?p ?a ?c)]
```

only recurse attributes marked :db/isComponent



Where Are We?

Motivation

Schema

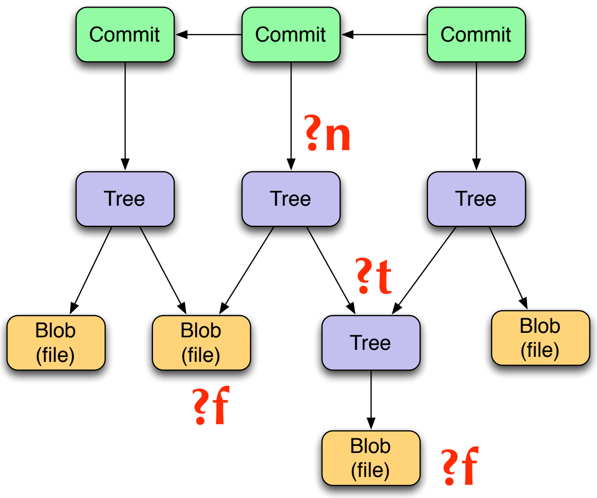
Import

Datalog

Rules

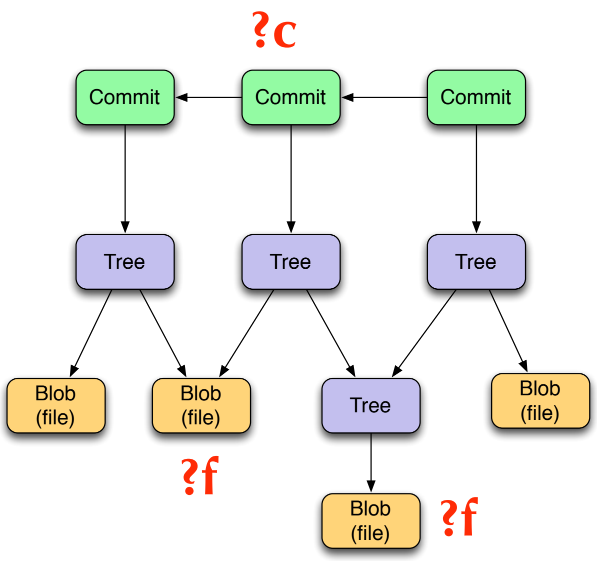
Queries

node-files



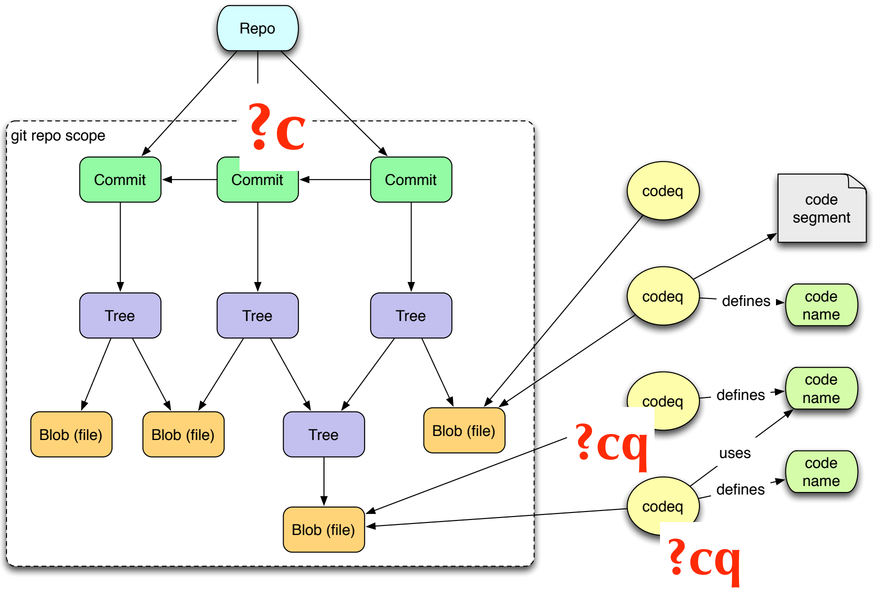
```
[(node-files ?n ?f)
 [?n :node/object ?f] [?f :git/type :blob]]

[(node-files ?n ?f)
[?n :node/object ?t] [?t :git/type :tree] [?t :tree/nodes ?n2] (node-files ?n2 ?f)]
```

commit-files



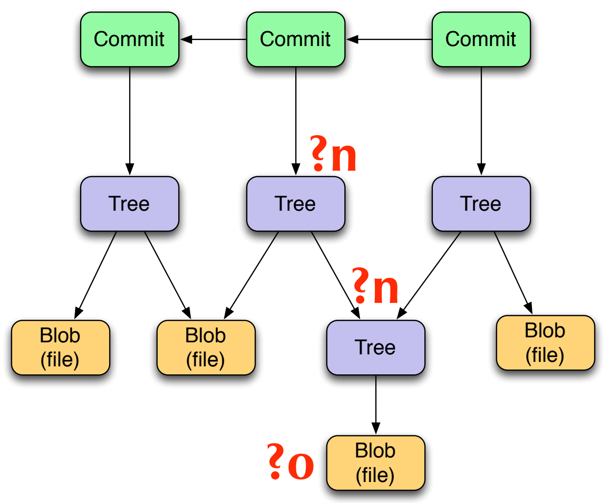
```
[(commit-files ?c ?f)
 [?c :commit/tree ?root] (node-files ?root ?f)]
```

commit-codeqs



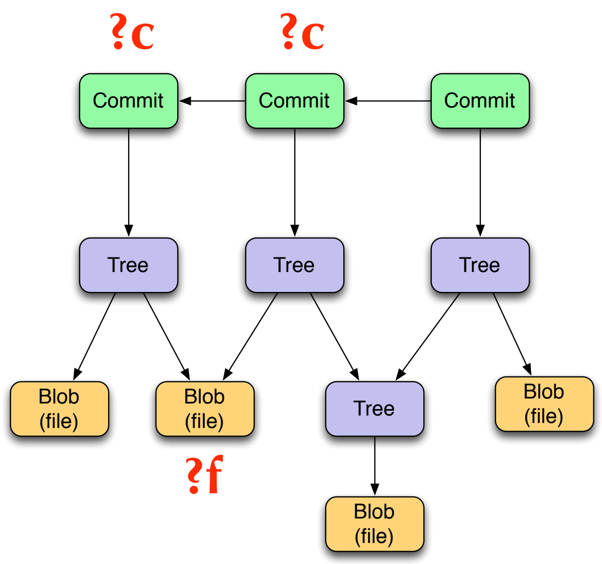
```
[(commit-codeqs ?c ?cq)
 (commit-files ?c ?f) [?cq :codeq/file ?f]]
```

object-nodes



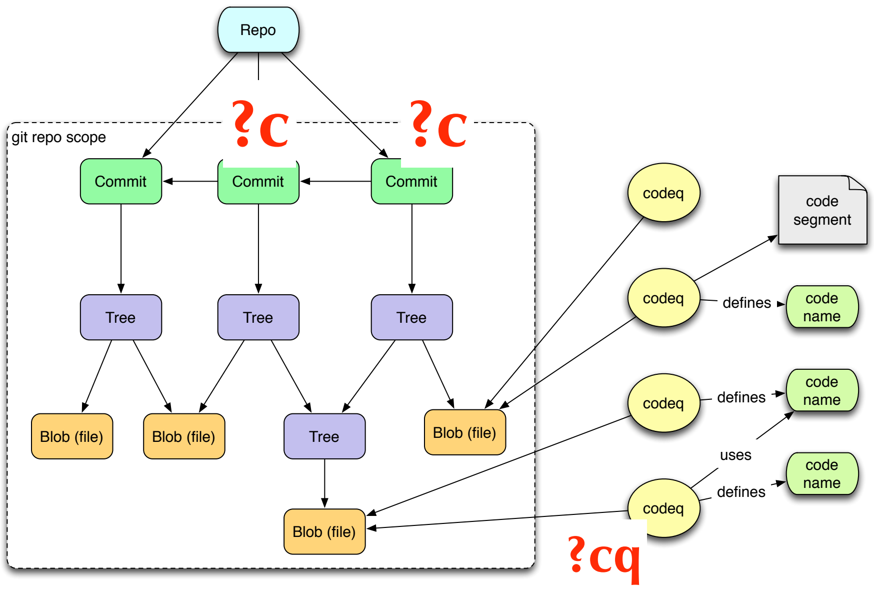
```
[(object-nodes ?o ?n)
 [?n :node/object ?o]]
[(object-nodes ?o ?n)
 [?n2 :node/object ?o] [?t :tree/nodes ?n2] (object-nodes ?t ?n)]
```

file-commits



```
[(file-commits ?f ?c)
 (object-nodes ?f ?n) [?c :commit/tree ?n]]
```

codeq-commits



```
[(codeq-commits ?cq ?c)
 [?cq :codeq/file ?f] (file-commits ?f ?c)]
```

Resources

Codeq
http://blog.datomic.com/2012/10/codeq.html. Introduction to codeq.
https://github.com/Datomic/codeq. codeq repository.
http://www.google-melange.com/gsoc/project/google/gsoc2013/navgeet/7001.
GSOC project.

Datomic
http://edn-format.org. The edn specification.
http://www.datomic.com/. Datomic.
https://github.com/datomic/datomic-groovy-examples. Datomic queries in Groovy.

Stuart Halloway
https://github.com/stuarthalloway/presentations/wiki. Presentations
http://thinkrelevance.com/blog/tags/podcast. The Relevance Podcast.
http://www.linkedin.com/pub/stu-halloway/0/110/543/
https://twitter.com/stuarthalloway