DEDUCTIVE DATABASES

Diego Garcia-Olano Spring 2014

1. Deductive Databases?
2. Datalog Educational System (DES)
3. Datomic
4. Demo
5. DES vs Datomic
6. Conclusion
7. References and Links

index

Definition:

A **Deductive Database** is a database system that can make deductions based on:

1) rules and
2) facts stored in the (deductive) database

Datalog is the language typically used to specify facts, rules and queries in deductive databases.

Why?

Combine **logic programming** with relational databases to construct systems that are expressive, fast, and able to deal with very large datasets.

What is logic programming?

- Data Centric and Rule Based Paradigm
- Simplifies writing recursive queries (SQL nightmare)

Where?

Deductive Databases have found application in:

data integration, information extraction, networking (graphs) program analysis, security, and cloud computing.

Datalog Primer

Declarations:

```
family:person:is_male(p)->family:person(p).
```

Constraints:

```
!(family:person:is_male(p),family:person:is_female(p)).
```

Relational predicates:

```
family:brother(x,b) -> family:person(x), family:person(b).
```

Functional predicates (->): X -> Y

"in order for X to hold, Y must be true"

family:father[x] = f -> family:person(x),family:person(f).

Derivation rules: (<-) X <- Y

"if X then Y "

family:brother(x,b) **<-**
  family:person:is_male(b),
  family:parent(x,p), family:parent(b,p),
  x!=b.

family:person:is_parent(p), family:person:is_father(p)
  <- family:parent(_,p),family:person:is_male(p).

Recursion:

family:ancestor(x,a) -> family:person(x), family:person(a).

family:ancestor(x,a) <- family:parent(x,a).

family:ancestor(x,a) <- family:parent(x,y),family:ancestor(y,a).

Question:

How would you model ancestor in RDBMS?

THINGS TO WATCH OUT FOR:

Given a relation **owes(X,Y)** indicates that X owes money to Y.

We define a predicate **avoids(X,Y)**, meaning that X tries to avoid running into Y.

In our system, people avoid someone they owe money, and
they avoid anyone that the person they owe money to avoids also:

```
avoids(X,Y) :- owes(X,Y).
avoids(X,Y) :- owes(X,Z), avoids(Z,Y).
```

Given the following:

```
owes(andy,bill).
owes(bill,carl).
owes(carl,bill).
```

What happens if we ask who Andy avoids?

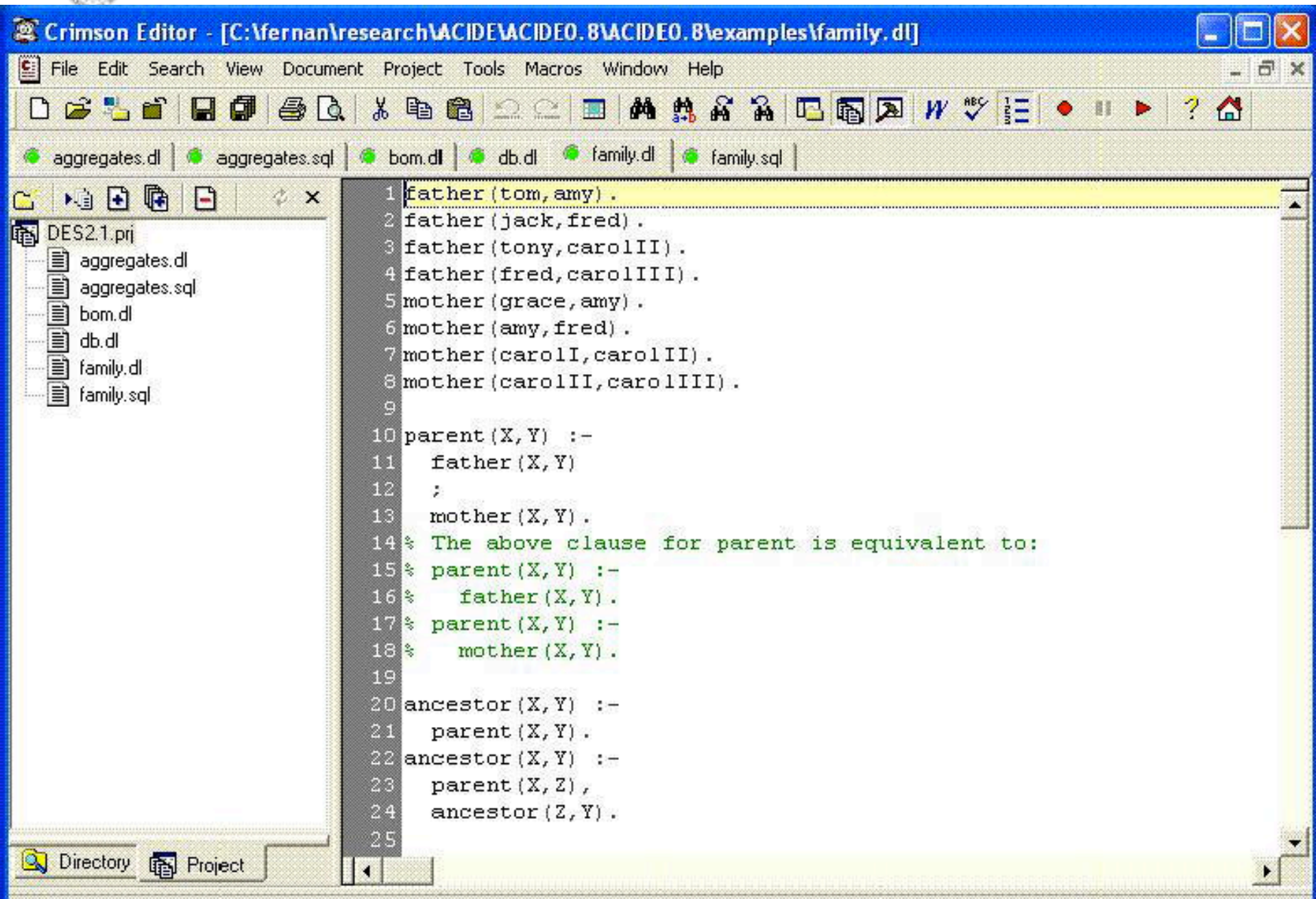
```
| ?- avoids(andy,X).
```

Datalog Educational System (**DES**)

http://www.fdi.ucm.es/profesor/fernan/des/index.html

a deductive database system
with Datalog, SQL, and Relational Algebra
as query languages.

* Free, open-source, multiplatform
* ACIDE GUI IDE or command line
* DES 1.0 released in 2003,
  DES 3.7 released in 2014


Crimson Editor - [C:\fernan\research\ACIDE\ACIDE0.8\ACIDE0.8\examples\family.dl]
File Edit Search View Document Project Tools Macros Window Help
aggregates.dl aggregates.sql bom.dl db.dl family.dl family.sql
DES2.1.prj
aggregates.dl
aggregates.sql
bom.dl
db.dl
family.dl
family.sql
Directory Project
1 father(tom,amy).
2 father(jack,fred).
3 father(tony,carolII).
4 father(fred,carolIII).
5 mother(grace,amy).
6 mother(amy,fred).
7 mother(carolI,carolII).
8 mother(carolII,carolIII).
9
10 parent(X,Y) :-
11 father(X,Y)
12 ;
13 mother(X,Y).
14 % The above clause for parent is equivalent to:
15 % parent(X,Y) :-
16 % father(X,Y).
17 % parent(X,Y) :-
18 % mother(X,Y).
19
20 ancestor(X,Y) :-
21 parent(X,Y).
22 ancestor(X,Y) :-
23 parent(X,Z),
24 ancestor(Z,Y).
25

```
DES> /consult relop.dl
Info: 18 rules consulted.

DES> /listing

a(a1).
a(a2).
a(a3).
b(a1).
b(b1).
b(b2).
c(a1,a1).
c(a1,b2).
c(a2,b2).
cartesian(X,Y) :-
  a(X),
  b(Y).
difference(X) :-
  a(X),
  not b(X).
```

Submitting a query is pretty easy:

```
DES> a(X)
{
  a(a1),
  a(a2),
  a(a3)
}
Info: 3 tuples computed.
```

You can interactively add new rules

```
DES> /assert a(a4)
DES> a(X)
{
  a(a1),
  a(a2),
  a(a3),
  a(a4)
}
Info: 4 tuples computed.
```

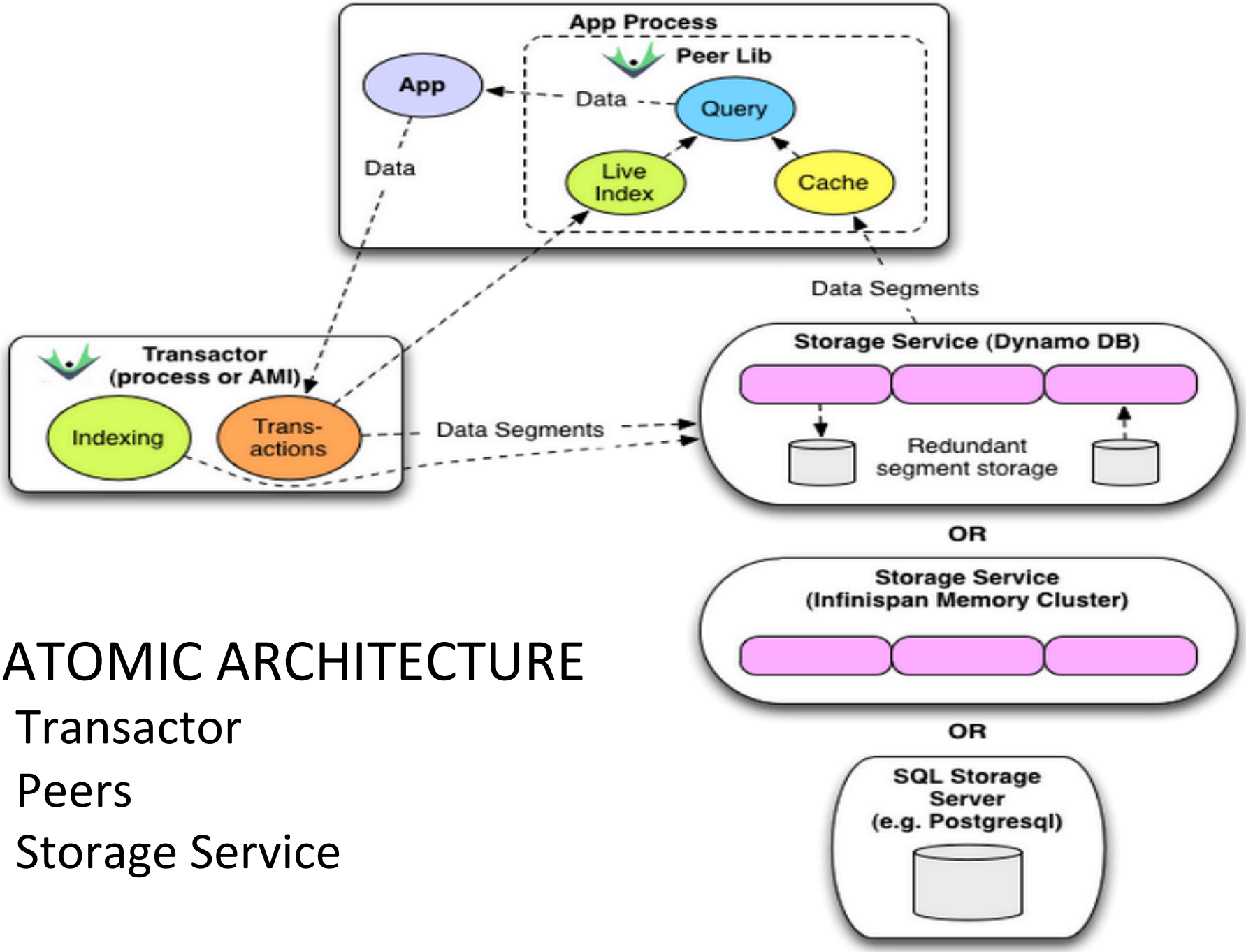
Datomic | The fully transactional, cloud-ready, immutable database.

Deductive database with timestamps

- simplifies the database by separating transactions process, storage and queries, and adopting a simple fact data model
- each transaction is RDF + time stamp
- all inserts, no updates, scalable, caching
- created by author of Clojure (2012)
- Active development

* Free, Pro Starter or Pro editions

* Java, Clojure, Python, Ruby, Groovy



DATOMIC ARCHITECTURE

- Transactor
- Peers
- Storage Service

Datomic In Action:

1) **Making a database and connecting to it**

```
uri = “datomic:mem://seattle”
Peer.createDatabase(uri);
conn = Peer.connect(uri);
```

2) **Adding a schema**

```
schema_rdr = new FileReader(“seattle/schema.edn");
schema_tx = Util.readAll(schema_rdr).get(0);
txResult = conn.transact(schema_tx).get();
```

3) **Adding data**

```
data_rdr = new FileReader(“seattle/data0.edn");
data_tx = Util.readAll(data_rdr).get(0);
txResult = conn.transact(data_tx).get();
```

https://data.seattle.gov/Community/Seattle-Communities-Online-Inventory/5ytf-wban

Name	url	Neighborhood	district	category	community, comm	type	region
15th Ave Community	http://groups.yahoo.com/	Capitol Hill	East	15th avenue residents	community	email list	E
Admiral Neighborhood Associ	http://groups.yahoo.com/	Admiral (West Seattle)	Southwest	neighborhood associati	community	email list	SW
Alki Beach Neighbors Blockwa	http://groups.yahoo.com/	West Seattle	Southwest, Al	crime, neighborhood-iss	community	website	SW
Alki News/Alki Community Cou	http://alkinews.wordpress	Alki	Southwest	news, council meetings	community	blog	SW
All About Belltown	http://www.belltown.org/	Belltown	Downtown	community council	community	website	W

```
;; community

{:db/id #db/id[:db.part/db]
 :db/ident :community/name
 :db/valueType :db.type/string
 :db/cardinality :db.cardinality/one
 :db/fulltext true
 :db/doc "A community's name"
 :db.install/_attribute :db.part/db}

{:db/id #db/id[:db.part/db]
 :db/ident :community/neighborhood
 :db/valueType :db.type/ref
 :db/cardinality :db.cardinality/one
 :db/doc "A community's neighborhood"
 :db.install/_attribute :db.part/db}

{:db/id #db/id[:db.part/db]
 :db/ident :community/category
 :db/valueType :db.type/string
 :db/cardinality :db.cardinality/many
 :db/fulltext true
 :db/doc "All community categories"
 :db.install/_attribute :db.part/db}
```

Schema File (.edn) (partial view)

Community Name
:type String, Cardinality: 1

Community Neighborhood
Type: Reference, Cardinality: 1

Community Category
Type: String, Cardinality: Many

```
[
    {   :district/region :region/e,
        :db/id #db/id[:db.part/user -1000001],
        :district/name "East"     }

    {   :db/id #db/id[:db.part/user -1000002],
        :neighborhood/name "Capitol Hill",
        :neighborhood/district #db/id[:db.part/user -1000001]     }

    {   :community/category ["15th avenue residents"],
        :community/orgtype :community.orgtype/community,
        :community/type :community.type/email-list,
        :db/id #db/id[:db.part/user -1000003],
        :community/name "15th Ave Community",
        :community/url "http://groups.yahoo.com/group/15thAve_Community/",
        :community/neighborhood #db/id[:db.part/user -1000002]     }
...
]
```

looking at the **Data File**

community category is a “many” relation.

```
    {   :community/category ["news" "council meetings"],  …. }
```

4) **Querying the database**

example 1: query to find all communities.

```
rs = Peer.q("[:find ?c :where [?c :community/name]]” , conn.db());
```

example 2: get id and category for all communities with the name "belltown”

```
rs = Peer.q("[:find ?e ?c :where [?e :community/name "belltown"]
                                 [?e :community/category ?c]]” , conn.db());
```

example 3: find all communities in a given region

```
rs = Peer.q("[:find ?cname :where [?c :community/name ?cname]
                                  [?c :community/neighborhood ?n]
                                  [?n :neighborhood/district ?d]
                                  [?d :district/region :region/ne]]", conn.db());
```

5) Advanced Queries

1. Parameterize queries

```
rs = Peer.q('''[:find ?n :in $ ?t :where [?c :community/name ?n]
                                        [?c :community/type ?t]]''', conn.db(),
     "[:community.type/facebook-page :community.type/twitter]");
```

2. Invoke code from within a query

```
rs = Peer.q('''[:find ?n :where [?c :community/name ?n]
                              [(.compareTo ?n "C") ?res] [(< ?res 0)]] ''', conn.db())
```

3. Do a fulltext search

```
query = [:find ?name ?cat :in $ ?type ?search :where [?c :community/name ?name]
[?c :community/type ?type] [(fulltext $ :community/category ?search) [[?c ?cat]]]]

    rs = Peer.q( query, conn.db(), ":community.type/website", "food");
    [["InBallard" "food"] ["Community Harvest of Southwest Seattle" "sustainable food"]]
```

4. Define and use query rules.

```
rules = '''[[[twitter ?c] [?c :community/type :community.type/twitter]]]''';
s = Peer.q('''[:find ?n :in $ % :where [?c :community/name ?n](twitter ?c)]''',
                                        conn.db(), rules);
```

6) **Working with Time** (querying past transactions)

```
rs = Peer.q("[:find ?when :where [?tx :db/txInstant ?when]]", conn.db());
db_asOf_data = conn.db().asOf(rs.get(4));

db_since_data = conn.db().since(rs.get(4));

query = "[:find ?c :where [?c :community/name]]";
Peer.q(query, db_asOf_data)
```

7) **Manipulating Data** (adding, updating, & retracting)

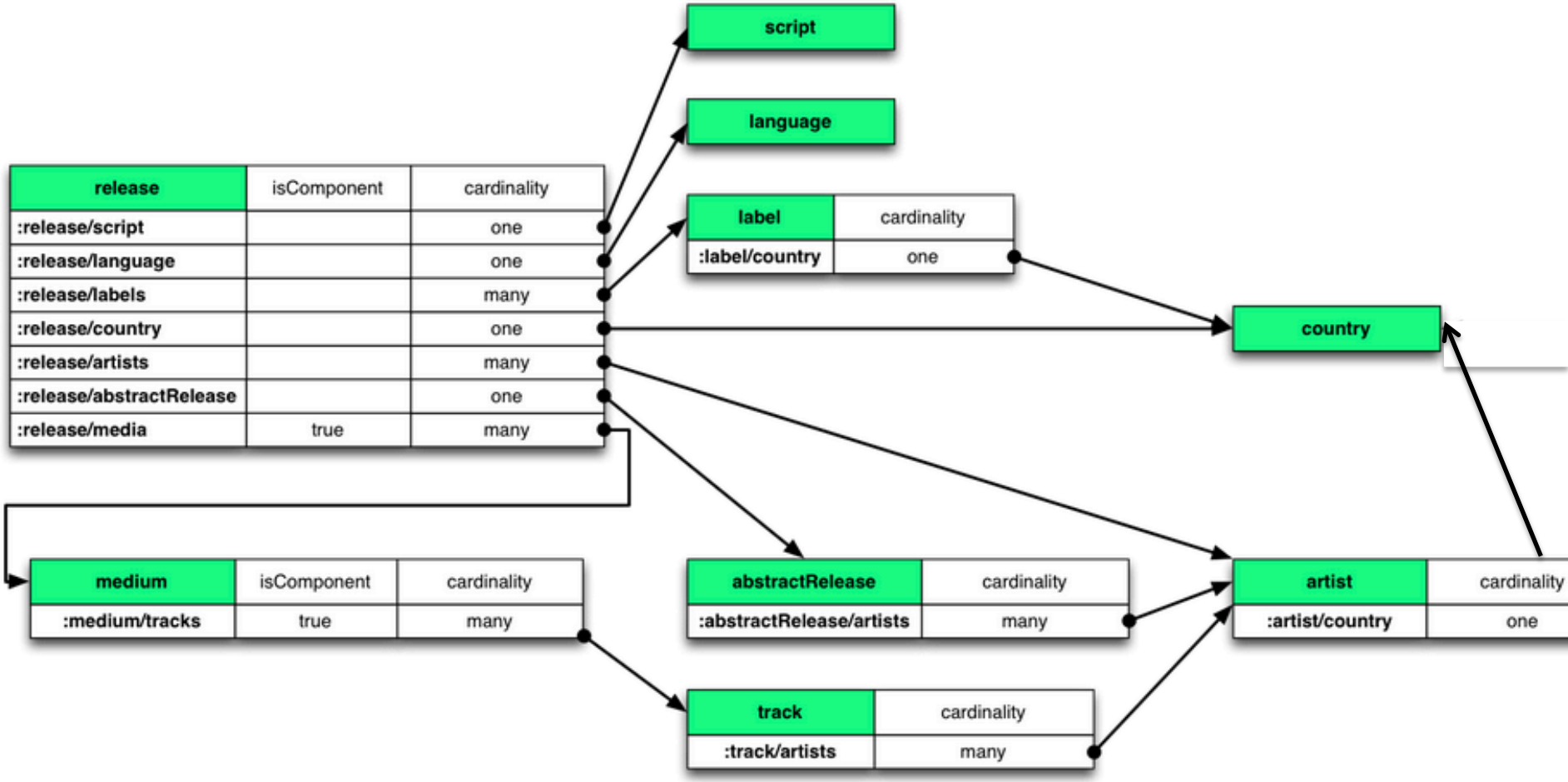
```
add_tx = [[":db/id": Peer.tempid(":communities"), ":community/name": "Easton"]];
txResult = conn.transact(add_tx).get()

rs = Peer.q("[:find ?id :where [?id :community/name \"belltown\"]]", conn.db());
belltown_id = results.iterator().next().get(0)
update_category_tx = [[":db/id": belltown_id,":community/category": "free stuff"]];
txResult = conn.transact(update_category_tx).get()

retract_category_tx = [[":db/retract", belltown_id, ":community/category", "free stuff"]];
txResult = conn.transact(retract_category_tx).get()
```

Demo: **Datomic** and **Music Brainz db**

MusicBrainz.org is an open music encyclopedia that collects music metadata and makes it available to the public.



Demo setup:

1. start transactor

   bin/transactor config/samples/free-transactor-template.properties

2. download mbrainz db backup and extract

   wget http://s3.amazonaws.com/mbrainz/datomic-mbrainz-backup-20130611.tar

   tar -xvf datomic-mbrainz-backup-20130611.tar

3. restore (ie, import) this backup into your environment

   bin/datomic restore-db file:/path/to/datomic-mbrainz-backup-20130611
   datomic:free://localhost:4334/mbrainz

4. launch interactive console and run queries.

   bin/console -p 8080 dev datomic:free://localhost:4334/

5. get the code *(*optional*)

   git clone git@github.com:Datomic/mbrainz-sample.git

DES vs Datomic

DES:

pros: Simple, Easy, Free, Educational Resource
Query hierarchy: Datalog, SQL, RA, Prolog

cons: Educational Resource,

Datomic:

pros: really fast, easy to install, works with java/python/clojure
data storage flexibility (sql / cassandra / infinispan)
community enthusiasm and buzz (rich hickey),
lots of tutorials, resources
use as a graph database *

cons: still quite young/new,
steep learning curve,
pay for performance, scalability enhancements, support

CONCLUSION

Deductive Databases allow you to

put the logic of your app largely within the DB itself
by modeling relations and rules there and
then writing programs which are largely just queries.

DES for education.

Datomic for real use.

RDF + timestamps is a powerful paradigm.

Datomic is not a good fit if you need unlimited write scalability, or have data with a high update churn rate (e.g. counters)

DEDUCTIVE DATABASES references:

Datalog

http://www.cs.sunysb.edu/~warren/xsbbook/node12.html
http://www.lifl.fr/~kuttler/elfe/
http://reexpress.wordpress.com/2011/04/28/

Datalog Educational System

http://www.fdi.ucm.es/profesor/fernan/des/index.html

Datomic (Site, Docs, Examples, Tools)

http://www.datomic.com/faq.html
http://docs.datomic.com/ (getting started, tutorial, console, architecture, examples)
https://github.com/Datomic/mbrainz-sample
https://github.com/Datomic/mbrainz-sample/wiki/Queries
https://github.com/gns24/pydatomic) - Python library for accessing datomic DBMS
https://github.com/Datomic/day-of-datomic - Clojure samples for using Datomic
http://hashrocket.com/blog/posts/using-datomic-as-a-graph-database

Deconstructing the Database (Rich Hickey, creator of Datomic)

https://www.youtube.com/watch?v=Cym4TZwTCNU