Datomic Cloud Documentation

Search

What is Datomic?

- Data Model
- Architecture
- Supported Operations
- Programming with Data and EDN

Local Dev and CI

Cloud Setup

- Start a System
- Configure Access
- Get Connected

Tutorial

- Client API
- Assertion
- Read
- Accumulate
- Read Revisited
- Retract
- History

Client API

Videos

- AWS Setup
- Edn
- Datalog
- Datoms
- HTTP Direct
- CLI tools

Schema

- Defining Attributes
- Schema Reference
- Changing Schema
- Data Modeling
- Schema Limits

Transactions

- Processing Transactions
- Transaction Data Reference
- Transaction Functions
- ACID
- Client Synchronization

Query and Pull

- Executing Queries
- Query Data Reference
- Pull
- Index-pull
- Raw Index Access

Time in Datomic

- Log API
- Time Filters

Ions

- Ions Tutorial
- Ions Reference
- Monitoring Ions

Analytics Support

ConceptHow it worksSQL mappingMetaschemasPreview Status

- Configuration
- Connecting
- Metaschema
- SQL CLI
- Troubleshooting

Analytics Tools

- Metabase
- R
- Python
- Jupyter
- Superset

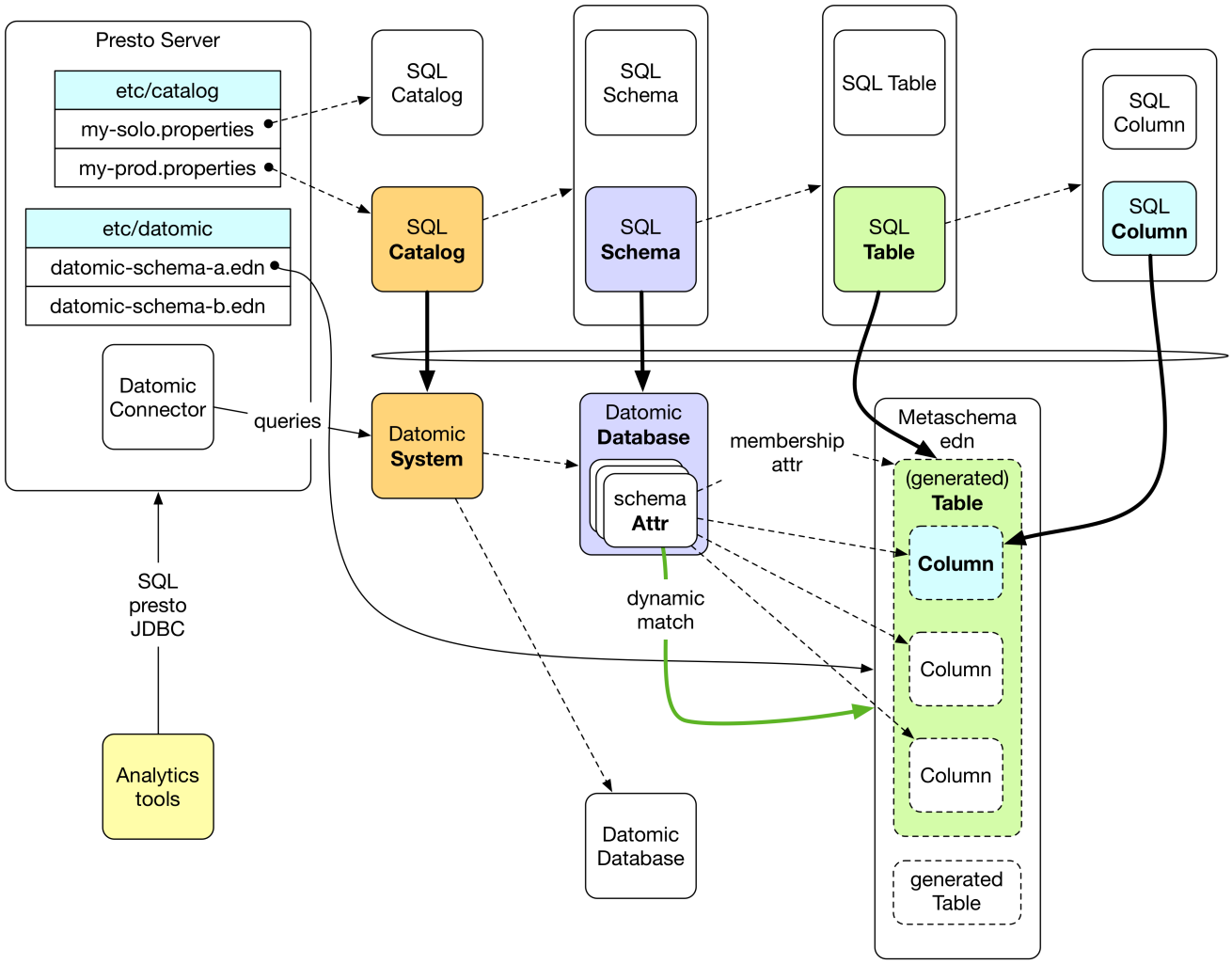
Analytics Support (Preview)

Concept

Datomic is a system of facts - values of attributes of entities. This is a rich model well suited for information programming, and Datomic's query language fits seamlessly into the information programming model of Clojure. However, you may have other members of your team - data scientists, analysts, operations people et al who need to leverage the information stored in Datomic but are not versed in Clojure. Instead they might have skills in Python, R, SQL or high-level analytics tools like Metabase, Tableau or Superset. The analytics support preview provides access to Datomic for these users, via those tools and more.

How it works

The Datomic information model is a graph, but analytics tools invariably expect data to be in rectangles. While there are many ways to present data as rectangles (CSV files etc), all analytics tools support data from relational, SQL sources. We've developed a Datomic driver for Presto SQL, a SQL query engine, and incorporated a Presto server into the access gateway for Datomic Cloud (see Configuration). Presto in turn supplies a SQL command-line interface, and there is native Presto protocol support for Python, R, Metabase, Superset, Tableau, and more. Presto also supports JDBC for integration with Java-based analytics tooling. In short, if you are looking to connect an analytics tool to Datomic, you will look for the "presto" connector, or use JDBC.



SQL mapping

The SQL information model is one of catalogs (data places and collections of schemas), schemas (collections of tables), tables (relations) and their columns. You will create a catalog properties file (Java style) to dynamically expose each Datomic system as a SQL 'catalog', and its databases as SQL 'schemas' (see configuration).

For each Datomic schema you use (perhaps shared across one or more databases) you will create a metaschema edn file, giving the file a name ending in .edn, which determines which SQL tables are available and their columns.

Metaschemas

We need a way to generate dynamic SQL tables and columns from Datomic entities and attributes. We need to determine:

- What are the tables?
- Which entities are in each table?
- Which attributes will provide its columns?

Metaschemas are edn files ending with a .edn extension that, with minimal entries, answer these questions.

Membership attributes

Membership attributes (attrs) determine both which tables are generated and which entities are in each table. For each membership attr at least one table is generated (see cardinality-many discussion below). If an entity has the membership attr, it is in the table, and if it does not have that attr it isn't in the table. The top level `:tables` entry of a metaschema is simply a map with membership attrs as keys, and options maps as values.

Namespace conventions determine (default) columns

The metaschema is designed to be minimal - you can supply more directives in the options maps, but nothing else is required. The default generation leverages the Datomic convention of putting (most of) the attributes used for a 'kind' of entity in the same namespace.

For example, here's what you'll get if you choose :product/code as a membership attr with an empty options map:

- a table named with the namespace part of the membership attr - "product"
- a row in that table for every entity that has a value for :product/code, even if they don't have values for other attrs/columns
- a column in that table called "db__id", containing the entity id
- a column in that table for every cardinality-one attr in the namespace :product, named with the name part of the attr – e.g. :product/name becomes a column called "name" with a type of varchar

Note that nothing about this convention limits you - you can exclude some default columns, include attrs from other namespaces, control the generated table and column names, etc. See the metaschema reference for details.

An additional table is generated per cardinality-many column

Since SQL columns are single-valued, cardinality-many (card-many) attrs need special handling. For each card-many column included either via the namespace defaults or explicitly, another table is generated containing all of the card-one columns of the base table plus a column for the card-many attr. Thus there will be repeating groups as there would be in a join.

For example, if there was a card-many attr called :product/tags, you would get:

- another table called "product_x_tags"
- with the "db__id", "name" etc columns from above, which may repeat
- plus a column called "tags"

Column joins - more denormalizing power

While analytics tools work with SQL, many are happiest to be supplied with wide, denormalized data sets rather than constructing and manipulating SQL joins in the tool. Normally one would accomplish this by encapsulating

the joins in SQL views. But such views would have to be duplicated in all dbs that share the same schema, and performing those joins in the presto engine is less efficient than doing so in Datomic. To facilitate the construction of these 'wide' views, the metaschema system provides column joins.

The `:joins` map maps ref attrs to tables already defined by the metaschema (think 'foreign keys'). When that attr contributes to a table the columns of the joined table are included as well.

For example, if there was a "country" table based on `:country/code` containing columns "code" and "name" for `:country/code` and `:country/name`, and a ref attr called `:product/country`, there would be an integer column in the "product" table called "country", containing the eid of a country. Adding an entry in the `:joins` map of `{:product/country "country"}` would add "country__code" and "country__name" columns to the "product" table. See the metaschema reference for more details.

One metaschema per Datomic *schema*

Quite often you might have multiple databases in a system that use the same schema. You need only create a metaschema for each Datomic *schema*, not for each database. The association of a metaschema with a database is done dynamically - if a database has all of the membership attrs of a metaschema, that metaschema will be used to generate tables for that database.

It's all virtual

It's important to note that the various wide and denormalized tables implied above are not pre-realized. They take no space. They are generated on the fly and only to the extent needed, i.e. you will only incur the cost to retrieve those columns you select. The card-many and column joins are more efficient than the equivalent SQL joins and views would be.

Preview Status

Analytics support is in preview. Note that:

- Details are subject to change.
- Some performance optimizations are not included in the preview release.
- We are interested in your feedback.

Copyright © Cognitect, Inc
Datomic® and the Datomic logo are registered trademarks of Cognitect, Inc