Datomic Cloud Documentation

Search

Schema

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

Transactions

- Processing Transactions
- Transaction Data Reference
  Transaction GrammarTx DataList FormMap FormOpEntity IdentifiersTempidsIdentifiersEntity IdsLookup RefsIdentsValuesTransaction Functions
- 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

- Configuration
- Connecting
- Metaschema
- SQL CLI
- Troubleshooting

Analytics Tools

- Metabase
- R
- Python
- Jupyter
- Superset

Hide All Examples

Transaction Data Reference

Datomic represents transaction requests as data structures. This is a significant difference from SQL databases, where requests are submitted as strings. Using data instead of strings makes it easier to build requests programmatically.

Transaction Grammar

Syntax

```
'' literal
"" string
[] = list or vector
{} = map {k1 v1 ...}
() grouping
| choice
? zero or one
+ one or more
```

The symbols `keyword`, `string`, `boolean`, `instant`, `uuid`, `long`, `bigint`, `float`, `double`, and `bigdec` are the primitive value types supported by datomic.

Grammar

```
tx-data                    = [list-form | map-form]+
list-form                  = ([op n-identifier identifier value] |
                              [tx-fn tx-fn-arg*])
map-form                   = {keyword (value | map-form | [map-form])}
op                         = (':db/add' | ':db/retract')
n-identifier               = (identifier | tempid)
tempid                     = string
identifier                 = eid | lookup-ref | ident
eid                        = nat-int
lookup-ref                 = [identifier value]
ident                      = keyword
value                      = (string | keyword | boolean | ref | instant | uuid | numbe
ref                        = n-identifier
number                     = (long | bigint | float | double | bigdec)
db-fn                      = identifier
classpath-fn               = qualified symbol
tx-fn-arg                  = (value | [value] | {value value})
tx-fn                      = db-fn | classpath-fn
```

Tx Data

```
tx-data                    = [list-form | map-form]+
```

Transaction data is a list of list forms or map forms.

List Form

```
list-form                  = ([op n-identifier identifier value] |
                              [tx-fn tx-fn-arg*])
```

Each list in a transaction represents one of

- addition
- retraction
- invocation of a transaction function

Example List Forms

```
[:db/add "John" :user/last-name "Doe"]
[:db/retract "John" :user/last-name "Doe"]
[:db/cas "my-account" :account/balance + 10]
```

Map Form

```
map-form                    = {keyword (value | map-form | [map-form])}
```

The map form provides a compact representation for multiple assertions about a single entity. Each map form is equivalent to a set of one or more `:db/add` operations. Map forms may include the special `:db/id` key to specify the entity identifier that the assertions are about.

Example Map Form

For example, the following map form:

```
{:order/id "X-001"
 :order/line-items [{:item/sku "SKU-7" :item/count 2}]
```

is equivalent to the following list forms:

```
[:db/add "-42" :order/id "X-001"]
[:db/add "-42" :order/line-items "-43"]
[:db/add "-43" :item/sku "SKU-7"]
[:db/add "-43" :item/count 2]
```

where "-42" and "-43" are tempids created by Datomic, and are guaranteed not to collide with any application tempids.

Op

```
op                          = (':db/add' | ':db/retract')
```

The op is the first item of a list form, and it specifies whether the datoms is an assertion (`:db/add`), or a retraction (`:db/retract`).

Example Ops

The following two transactions first assert that entity 42 likes pizza, then retracts this fact.

```
;; tx 1
[[:db/add 42 :likes "pizza"]]
```

```
;; tx 2
[[:db/retract 42 :likes "pizza"]]
```

Entity Identifiers

```
n-identifier                = (identifier | tempid)
tempid                      = string
identifier                  = eid | lookup-ref | ident
eid                         = nat-int
lookup-ref                  = [identifier value]
ident                       = keyword
```

Every datom is about an entity. There are three possible ways to identify an existing entity:

- an eid for an entity that is already in the database
- an ident for an entity that has a `:db/ident`
- a lookup ref for entity that has a unique attribute

In transactions, there is a fourth option for identifying an entity that might not exist yet:

- a tempid for a new entity being added to the database

Tempids

```
tempid                      = string
```

When you are adding data to a new entity, you identify it using a tempid. Datomic will convert tempids to entity ids when applying a transaction.

A temporary id is simply a string. The content of the string does not matter to Datomic – it is used only as an opaque identifier. Datomic guarantees that multiple uses of the same tempid within a transaction will resolve to the same entity id.

Tempid example

The following list forms use the tempid "foo" twice to indicate that two datoms are about the same entity.

```
[:db/add "foo" :item/sku "sku-42"]
[:db/add "foo" :item/description "Chandrian Repellant"]
```

Datomic does not care what name you use for a tempid, but it is often convenient to use a string that uniquely identifies the entity within the domain. So the previous example might also be stated as:

```
[:db/add "sku-42" :item/sku "sku-42"]
[:db/add "sku-42" :item/description "Chandrian Repellant"]
```

Implicit Tempid

If no *:db/id* value is specified in a transaction map form, Datomic will assign a unique tempid.

For new entities, you only need to explicitly specify a temporary id if:

- you want to recover the assigned entity id from the transaction result
- you need to refer to that entity elsewhere in the same transaction

- Implicit Tempid Example

  In the following example, "sku-42" is explicit so that the item and its occurrence in inventory can be linked. The "sku-43" entity occurs only once, so no explicit tempid is needed.

```
{:db/id "sku-42"
 :item/sku "sku-42"
 :item/description "Chandrian Repellant"}
{:item/sku "sku-43"
 :item/description "Iron Wheel"}
{:inventory/item "sku-42"
 :inventory/count 10}
```

⇧

Identifiers

```
identifier                = eid | lookup-ref | ident
```

An identifier uniquely identifies an entity within a single Datomic database. An identifier is either an entity id, a lookup-ref, or an ident.

Entity Ids

```
eid                       = nat-int
```

Datomic assigns an entity id (eid) to every entity, and this eid is stored in the e position of every datom. Entity ids are non-negative integers, but this is an implementation detail. Applications should treat entity ids as opaque identifiers.

Lookup Refs

```
lookup-ref                = [identifier value]
```

A *lookup-ref* allows you to specify entities by their domain-unique identities. Inside a lookup-ref, the *identifier* names a unique attribute in the database, and the value is a valid value for that attribute. For example, this lookup ref:

```
[:person/email "joe@example.com"]
```

identifies the entity with the *:person/email* value "joe@example.com".

Idents

Idents are programmatic names that associate a keyword with an entity id. All idents for a database are interned and cached in memory on every Datomic compute node.

To create an ident, add a `:db/ident` value for an entity. The transaction data below upserts four idents: `:red`, `:green`, `:blue`, and `:yellow`

```
[{:db/ident :red}
 {:db/ident :green}
 {:db/ident :blue}
 {:db/ident :yellow}]
```

In a subsequent transaction, you can the use an ident instead of referring to an entity by name. The example below uses the `:yellow` instead of an entity id.

```
{:db/doc "The submarine where we live"
 :color :yellow}
```

Values

```
value                    = (string | keyword | boolean | ref | instant | uuid | numbe
```

The value type of a datom is dictated by its attribute's schema. The value types are defined and demonstrated in the Schema Data Reference.

Transaction Functions

```
list-form                = ([op n-identifier identifier value] |
                            [tx-fn tx-fn-arg*])
db-fn                    = identifier
db-fn-arg                = (value | [value] | {value value})
```

Transaction functions are code that is executed inside of a transaction. Transaction functions provide semantics beyond basic asserts or retracts; in particular they allow the data added in a transaction to be derived from the current value of the database.

There are two kinds of transaction functions: *database functions* are loaded from the database, and *classpath functions* are loaded from the Java classpath.

The set of database functions in Datomic Cloud is fixed, and is documented under Built-In Transaction Functions.

You can write your own classpath functions and deploy them using Ions.

Either insert or update an entity, depending on whether the unique entity already exists.