## Simple datatypes

Numeric

Each maps to a java type

bigint => long

decimal => BigDecimal

varint => BigInteger

Strings (all map to String)

ascii (US ascii string)

text (UTF8 string)

varchar (synonym for text)

Dates

timestamp

timeuuid (provides uniqueness)

Other

boolean

uuid

inet (ip addresses)

blob (binary data)

## Complex datatypes

Collections

TTL works with elements in a collection

update user using TTL 60 set foo = foo + {‘shortlived’:234} where id = ‘test’;

set

useful where you want to model something, like features, which may vary over time, and for which you don’t want to add a new column every time a new feature is created.

feature<varchar>

Insert

insert into foo (id,name,features) values (1, ‘intro’, {‘cc’});

Add an entry to exiting row

update foo set features = features + {‘color’} where id = 1;

Remove entry

update foo set features = features - {‘color’} where id = 1;

Remove all entries

update foo set features = features + {} where id = 1;

list

maintains order

clips list<varchar>

Insert

insert into foo (id,name,clips) values (1, ‘intro’, [‘cc’]);

Add an entry to exiting row

update foo set clips = clips + [‘color’] where id = 1;

update foo set clips = [‘color’] + clips where id = 1;

Remove entry

update foo set features = clips - [‘color’] where id = 1;

Removes all matching entries

Update element by position index

update foo set clips[3] = ‘hello’ where id = 1;

Remove entry by index

delete clips[3] from foo where id = 1;

Remove all entries

update foo set features = features + [] where id = 1;

maps

CREATE TABLE users (

id varchar,

name varchar,

password varchar,

reset\_token varchar,

// map of device key to last login

last\_login map<varchar,timestamp>,

PRIMARY KEY (id)

);

INSERT INTO users (id, name, last\_login) VALUES

('john-doe', 'John Doe', {'383cc02343991f': '2016-02-28 09:30:29'});

Update existing map

update users set last\_login['383cc02343991f'] = '2016-02-28 09:30:29' where id = 'john-doe';

update users set last\_login = last\_login + {'23f23f23fff':'2016-01-05 14:11:45'} where id = 'john-doe';

delete from users set last\_login[‘asdfsdafsafs’] where id = ‘john-doe’;

update users set last\_login = last\_login - {‘adsfasdav2v’} where id = ‘john-doe’;

update users set last\_login = {} where id = ‘john-doe’;

### Tuples / Nested types

Tuple - series of values which may be of different types, yet which maintain a certain order

Useful if you want to store muliple entries of different types as a single entry in a map value.

Nested complex types

Types in which a complex type is nested in another complex type.

The nested complex type must be associated with the frozen keyword

Frozen

Nested types are stored as a blob in the datastore

They must be read and set as an atomic whole

User-defined types and collections quality as nested types.

tuple<timestamp,inet>

map<int, frozen<tuple<timestamp,inet>>>

list<frozen<set<varchar>>>

### User-defined types

Creates a new type

CREATE TYPE CLIP (name varchar, duration int);

Nested types are similar to tuples, except that items of a user type can be referenced by name rather than by index as is done with tuples.

Better documentation of the data model

Often, it’s wise to start modelling as a tuple, then switch to user-defined type once you’ve settled on things.

User-defined types are reusable.

User-defined types must be defined as frozen even if not nested.

### Modeling relationships

Composite primary key with a clustering key

Within a partition (course), rows are grouped by the clustering key (module).

One to many relationship

Modules contain clips.

Within each clustered index, or module, a collection of user-defined clip types models the one to many relationship between modules and clips.

All data for a given course is in a single partition