Conditions Databases at FNAL

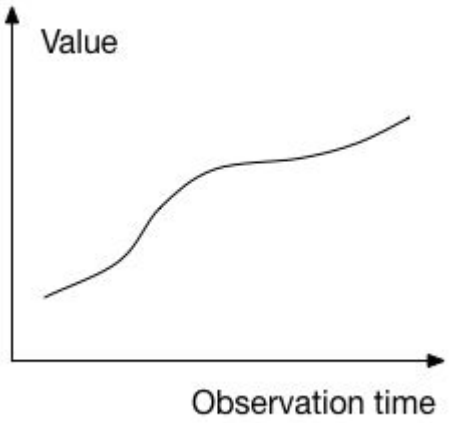
Igor Mandrichenko

DUNE databases meeting

October 2, 2019

What is a Conditions Database ?

Record of a “quantity” as a function of “time”



More generally, “quantity” can be:

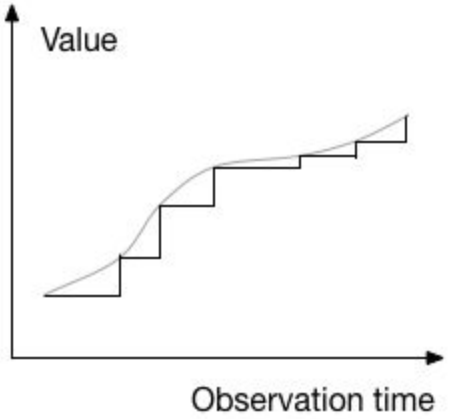
- Scalar
- Tuple
- Array of tuples indexed by “channel”
- BLOB

“Time” is “observation” or “validity” time

- Can be date/time or an integer or a floating point number

Interpolation

It is impractical to actually record complete history of the quantity as a function of time



Instead: record a series of measurements of the quantity at discrete time points and interpolate between them

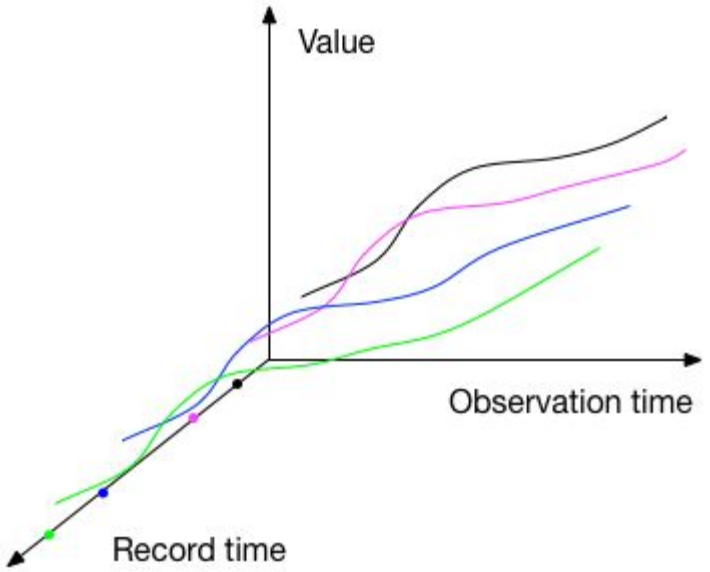
- Simplest interpolation: assume the quantity is constant until next measurement

Rollback or Version Control

In fact, there is another time dimension - *record* time

Every time the data is updated in the database, the “history is re-written” and sometimes we need to recall a previous state of the database:

- rollback to certain *record* time
- named (tagged) state of the database - database version



Never delete any data.

Instead: override but keep old state accessible.

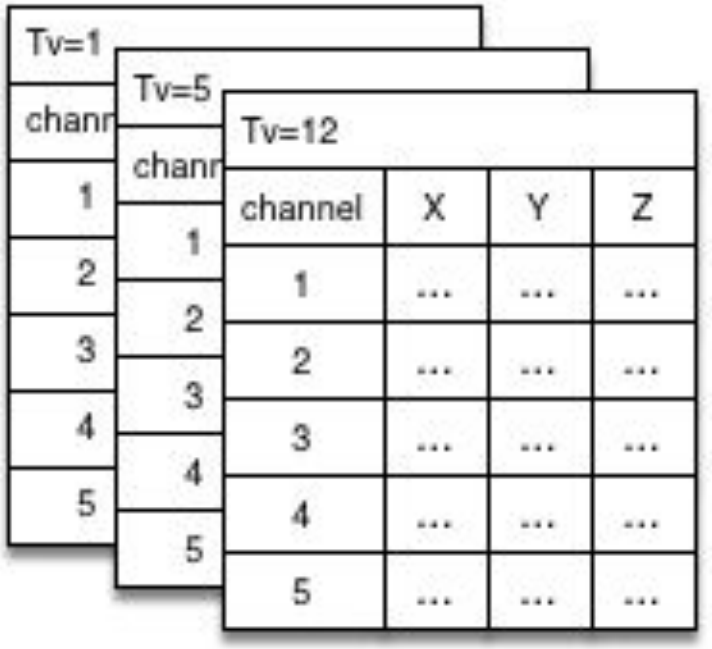
Products: Minerva Conditions DB

The oldest: introduced in 2011 to replace COOL

Records state of an array of tuples indexed by integer channel number

All channels are measured simultaneously

Currently used my Minerva, MicroBooNe, G-2



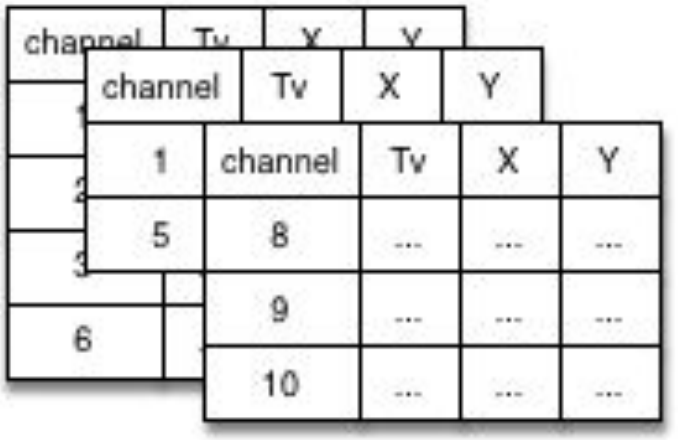
Products: NOvA Conditions DB

Since 2013

Records state of an array of tuples indexed by integer channel number

Each channel measured independently

Data compression by not recording new value if it is too close to the previous

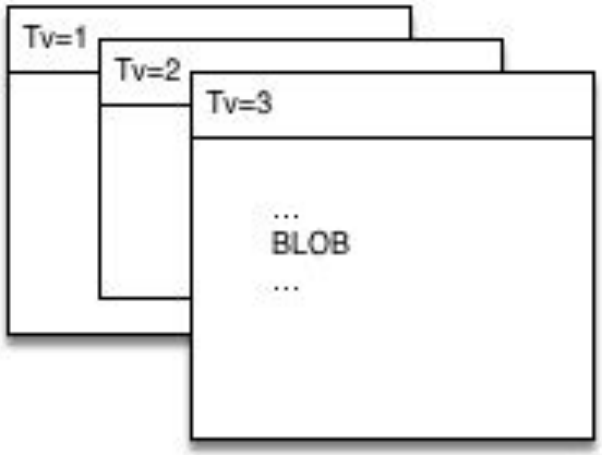


Products: UConDB

UConDB = Unstructured Conditions DB

Introduced in 2015

Original use case: store large number of FHICL files, providing interpolation and version control features



Stores state of a BLOB as a function of time

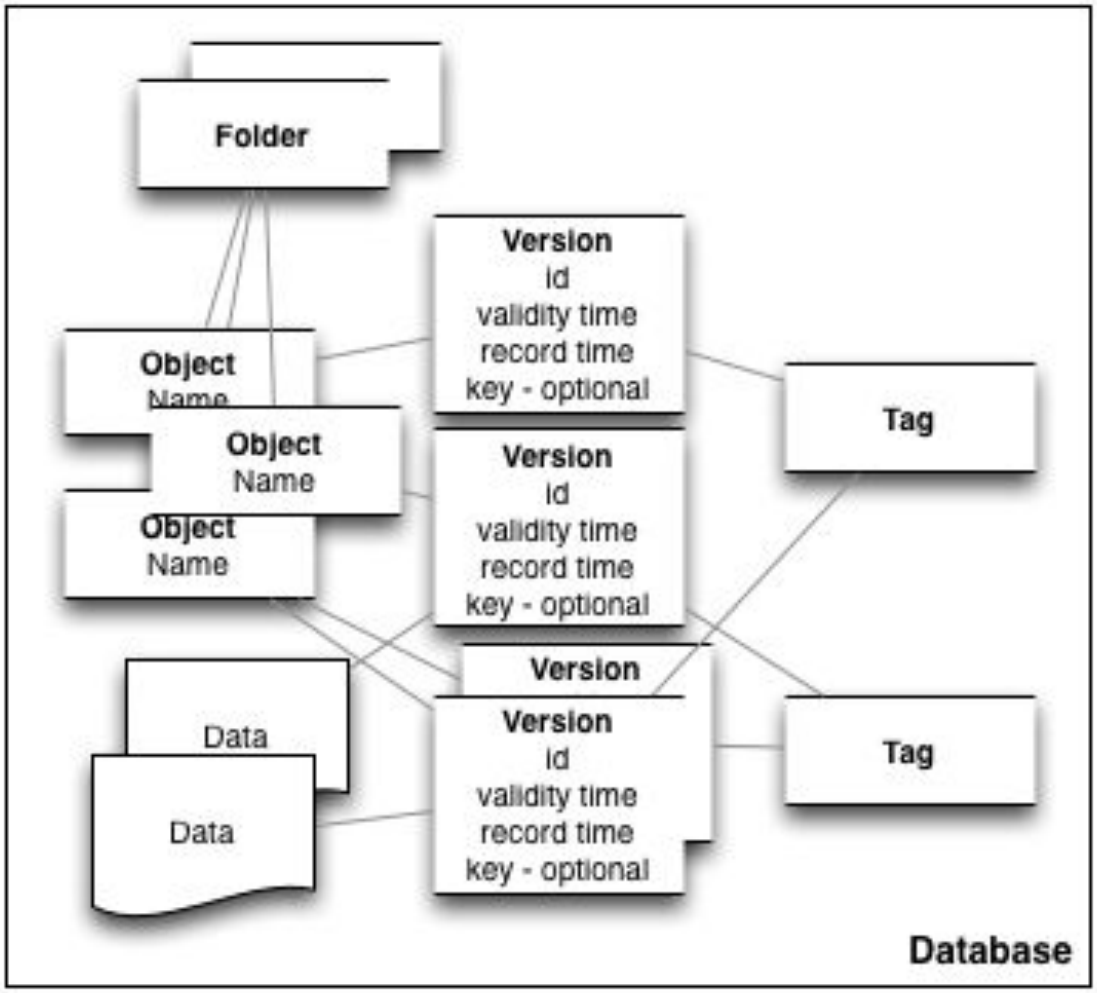
- Documents
- Binary data
- Anything else interpretable by the client

Currently used by ProtoDUNE to record run configurations

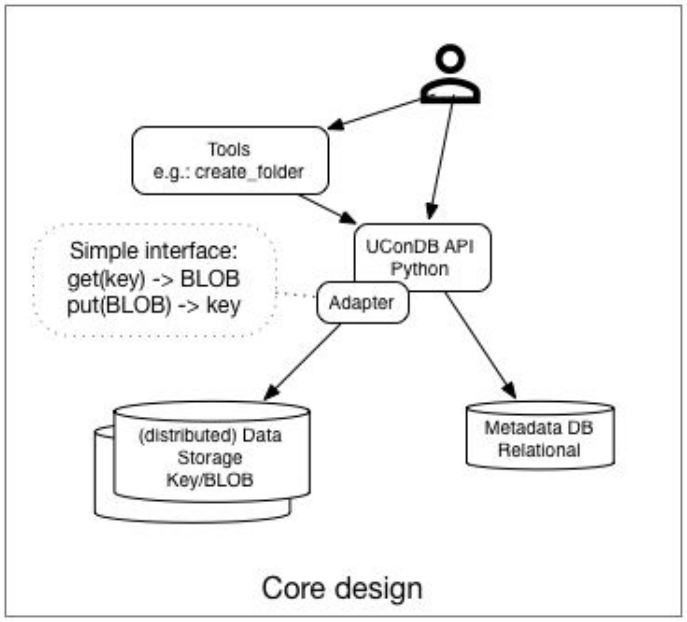
UConDB Concepts

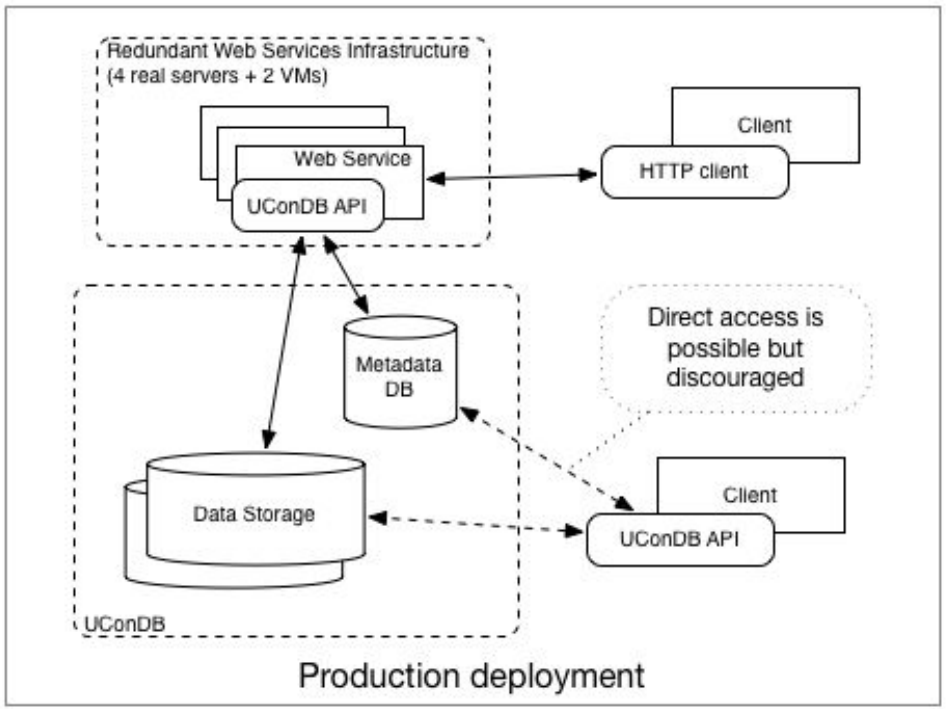
- Database is organized as a set of named folders
- Objects (BLOBs) live inside folders
- Object has a name unique for the folder
  - Object identification: <folder name, object name>
- Object has versions (i.e. measurements, observations)
- Version properties:
  - Tv - numeric validity time (observation time)
  - Tr - version creation time (record time)
  - Value - BLOB
  - Id - an integer, automatically assigned by the DB when the version is created and returned to the user
  - Key - text - unique user defined version identifier (optional)
  - Tag(s) - text - optional - can be shared by many objects/versions

UConDB Data Model



UConDB Design and Deployment





API, tools - Python

Metadata - Postgres

Data storage - pluggable implementations

- key/BLOB interface adaptor
- Postgres
  - smaller databases
- CouchBase - distributed NoSQL product
  - larger databases, horizontal scalability, memory cache, data replication, HA

Recording Data

To record an object version, specify:

- Folder name (required)
- Object name (required)
- Tag (optional)
- Key (optional)
- Value - BLOB

Database returns

- Generated numeric version ID (in case you care to remember it)

Reading Data

Object version can be retrieved by:

- Folder name (required)
- Object name (required)
- Tv (optional, default: current time)
- Tr - record time specification (optional)
- Tag (optional)
- Key (optional)
- ID (optional)

Metadata

- List of objects in the folder
- Versions for object
- Version metadata
- Tags for folder

REST Interface

Uploading:

```
curl -T /data/file.txt \
     -X POST \
     http://server.fnal.gov/ucondb/data/folder_name/object_name?...
```

Authentication:

- RFC2617, digest authentication
- shared password used to calculate digital signature but not sent over the wire

```
curl -T /data/file.txt -X POST --digest -u user:password …
```

Retrieving:

```
curl -o /data/file.txt \
     http://server.fnal.gov/ucondb/data/folder_name/object_name?...
```

ProtoDUNE Run Configurations DB

UConDB with Postgres data backend

~5000 run configurations

- FHICL file
- Average size ~3MB

~16 GB total data size

Example: