What is NoWDB?

Tobias Schoofs

April 20, 2023

Overview

- What?
- Why?
- How?
- Vision!
- Market!
- Where are we now?
- What means the name?

What?

$$\text{NoWDB} = (graph + timeseries + \text{SQL}) \times (Python + Lua)$$

Why Timeseries? (1/2)

- Growing amount of data are timeseries *to some extent*
- Important driver: IoT
- But also: "traditional" industries:
  - IT infrastructure providers,
  - Energy,
  - Manufacturing ("Industry 4.0"),
  - Retail,
  - Fintech,
  - ...
- Timeseries data have special requirements:
  - Fast ingestion ($> 1M/s$)
  - Scalability of queries ($> 1Bmetrics$)
  - Special handling of time dimension
  - Timeseries are less rigid compared to relational data (time points may be lost or duplicated)

Why Timeseries? (2/2)

- Avalailable timeseries databases are too narrow, *i.e. pure* timeseries.
- Examples:
  - InfluxDB
  - OpenTSDB
- There are projects trying to solve this issue, *e.g.*:

$$timescale = postgres + timeseries$$

. . . but it's still Postgres: somewhat slow

Why Graph?

- Make timeseries more widely applicable!
- Natural candidates: Relational and Graph
- Graph is more flexible:
  we can add new edges
  without changing the structure of entities
- Graph is less rigid:
  edges pointing to nowhere or
  pointing to duplicated data are no issue
- Graph may be easier to grasp:
  and this way reduce common errors in data modelling.
  (This, however, is pure speculation on my side . . .)

The Founder of Graph Theory:
Leonhard Euler

Why SQL?

- ~~Make SQL great again!~~
- SQL is standard.
- SQL is actively developed with new ideas flowing in.
- SQL is known by virtually everybody in the industry.
- SQL is easy to learn (even though tricky to master . . .)
- SQL is designed for integration with other languages and tools.
- SQL is supported by thousands of tools.
- SQL is not bad anyway.
- Well . . .
- Next slide!

Why Python?

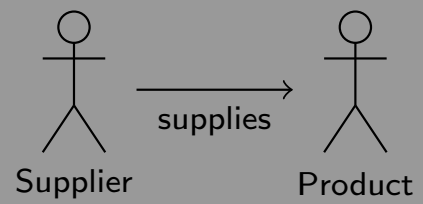
- An important application domain is data science.
- There will be demand for tools, libraries, languages supporting
  - Statistics
  - Calculus
  - Numerical analysis
  - Function analysis and function plotting
  - Linear algebra (vectors, matrices, *etc.*)
  - Graph(!)-oriented data structures
  - ...
- Millions of lines of code available in (or through) Python
- Python is **the** data science language.
  (There also is R, but R is very limited as general purpose language)
- It is in particular interesting to have Python available on server side.
  This will reduce client/server round trips and network traffic.

Why Lua?

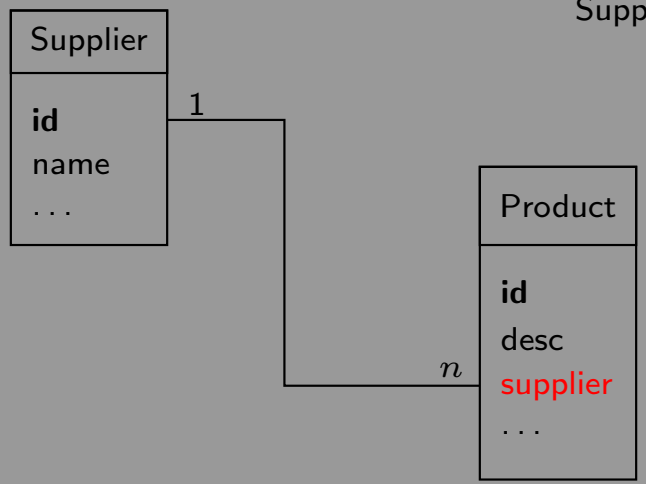
- Python has an issue with its interpreter:
  - difficult to run many interpreters (= sessions)
  - in parallel in one process (= NoWDB server).
- In consequence: Python is not good for tasks that frequently run in parallel.
- We therefore need a second language for more down-to-earth, day-to-day DBA jobs.
- Lua appears to be a good candidate because:
  - Lua is well known in the data science community
  - Lua already had its appearance on the database stage (namely in the redis *in-memory* database)
  - Lua is designed as configuration language (and DBA tasks are often configuration tasks)
  - Lua is designed for integration with the C language.

How? Data Modelling

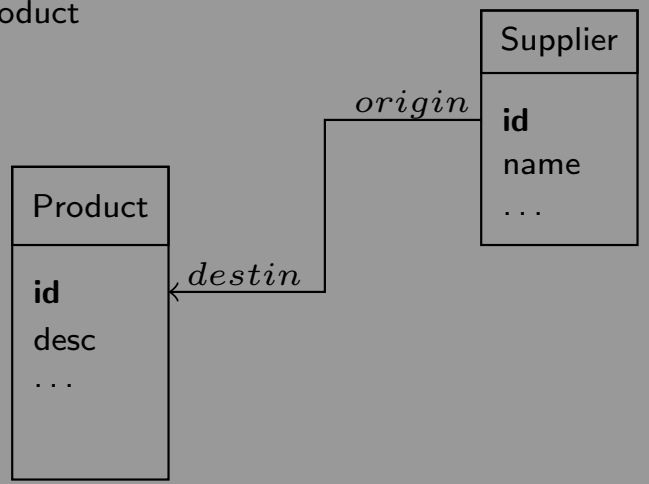
The Logical Fact



The Relational Way



The Graph Way



How? Graph

Create a vertex type

```
create type supplier (
  id uint primary key,
  name text,
  ...
);
```

Create another vertex type

```
create type product (
  id uint primary key,
  description text,
  ...
);
```

Add an edge
without changing the types

```
create edge supplies (
  origin supplier,
  destin product
);
```

How? Timeseries

Product type

Client type

Timeseries edge

```
create type product (
  id uint primary key,
  description text,
  ...
);
create type client (
  id uint primary key,
  name text,
  ...
);
create edge buys (
  origin client,
  destin product,
  timestamp time,
  quantity float,
  price    float
);
```

How? Queries (1/2)

Who bought product 12345?

(This query runs in ms!)

```
select origin
  from buys
 where destin = 12345
   and year(stamp) = 2018
   and month(stamp) >= 10;
```

Traditional Join

```
select p.description
  from buys (join product as p on destin)
            (join client  as c on origin)
 where c.name = 'Smith'
   and year(stamp) = 2018;
```

Implicit join

```
select destin.description
  from buys
 where origin.name = 'Smith'
   and year(stamp) = 2018;
```

How? Queries (2/2)

Who are the suppliers of products bought by client 'Smith'?

Can we use CTE for recursive search on paths?
Well ...
Next Slide!

```
select destin.supplies.origin.name
  from buys
 where origin.name = 'Smith'
   and year(stamp) = 2018;

with path (n,e) as (
  select 1, origin.friend
    from friend
   where origin.name = 'Smith'
     and year(stamp) = 2018
  union
  select n+1, destin.friend
    from friend
   where year(stamp) = 2018
     and n < 10)
select count(*)
  from path
 where origin.name = 'Muller';
```

How? Storage Engine (1/2)

Two storage types:

1. Edge Store:
   - optimised for fast ingestion
   - data packed to favour typical queries
   - special handling of timestamps
   - small memory footprint
   - optional compression (recommended for timeseries edges)
2. Vertex Store:
   - consistent data (no duplicates)
   - column-oriented (*i.e.* optimised for large data types)
   - enforce primary key
   - in-memory caches to speed up inserts and queries
   - optional compression (recommended only for large tables)

How? Storage Engine (2/2)

Four index types:

1. Primary Keys (Vertex)
   - B^+Tree
   - storing set of *pageids*
2. Secondary Vertex Index
   - B^+Tree
   - storing set of primary keys
3. Page Index (Edge):
   - B^+Tree
   - storing "embedded" B^+Trees storing pageids
   - used for queries and joins on edges
4. Label Index (Edge):
   - B^+Tree
   - storing fixed number of pageids
   - used to find related edges

Vision!

- **Scalability!**
  Reduce hardware and software license cost by allowing more data to be stored on one node, *i.e.* billions of metrics per server.
- **Data Analysis**
  Provide server- and client-side Python modules to ease data science.
- **Integration with big-data tools**
  Ease big-data by integration with existing tools *e.g.* Kafka for reliable ingestion of data.
- **Integration with distribution engines**
  Ease large-scale computation by integration with existing tools *e.g.* SPARK for distributed computing.
- **Integration with visualisation frameworks**
  Ease application development by integration with existing GUI frameworks, *e.g.* Grafana.
- **Integration with traditional tools**
  Use ODBC/JDBC to make the platform available to traditional tools.
- **Publish/Subscribe Capability**
  Provide pub/sub capability to ease the development of monitoring and real-time data analysis.

Market!

- **Complement** existing OLTP infrastructures
- **Compete** with complex big-data and data science platforms providing a low-cost alternative to frameworks with high cost of ownership (*e.g.*: Hadoop, large OLAP systems, *etc.*)
- Target industries with "timeserish" data and need of data science (*e.g.* IoT, retail, fintech, "Industry 4.0")
- Provide pre-packaged out-of-the-box solutions (*e.g.* integration with Kafka, Zookeeper, SPARK & Grafana)
- Operate with a mix of open-source and proprietary licenses (*e.g.* generic modules for free, specific modules for pay)
- Foster a community among customers to grow repository of available solutions and modules
- Foster community of *certified* service providers

Where are we now? (1/2)

- Prototype is available:
  - Client/Server (with clients for Python, C and Go)
  - SQL with many operators and functions:
    - arithmetic and boolean operators and functions,
    - CASE and other conditionals,
    - time and geospatial functions,
    - trigonometry,
    - string operators and functions,
    - ...
  - Python client- and server-side
  - Simple command line client
  - Foundations for grouping, ordering and joins
  - Storage engine needs revision
  - User manual on the way
  - Query planner still rudimentary
  - Todo: security and user management

Where are we now? (2/2)

- We are currently working with the IoT startup **Loka Systems** (`https://loka.systems`) to identify strengths and weaknesses.
  - The Loka database has about 30M metrics
  - It was imported into NoWDB in $< 30s$
  - Queries were implemented by means of server-side Python modules
  - The feedback of the Loka team already led to conceptional and technical improvements
- We created a database with data by the World Meteorological Organisation (WMO)
  - with weather data covering the years 1970 – 2018
  - with more than 1B metrics
  - Import takes about 5 minutes
  - Index-based queries run in milliseconds
  - Benchmarks reveal some weaknesses (*e.g.* grouping)

www.wmo.int

The funny Name

- a reference to time
- a pun on *no*SQL and *new*SQL
- . . . and a homage to Norbert Wiener, the father of timeseries analysis for Control & Communication