среда, 5 октября 2022 г. 22:33

Relational

relations ~ table
relation ~ row
(tuple)

1 - *

* - *

on write

declarative QL

users

(ui64	str	str)	
user_id	first_name	second_name	born
251	Bill	Gates	USA, ...

positions

comp_id	user_id	job	company
400	251	co-chair	Bill Gates Foundation
401	251	co-founder	Microsoft

education

id	user_id	name	start	end

user_id	comp_id
251	400
251	401
138	401

object-relation mismatch

ORM

Object-relation mapping

schema on write

schema on read

JSON

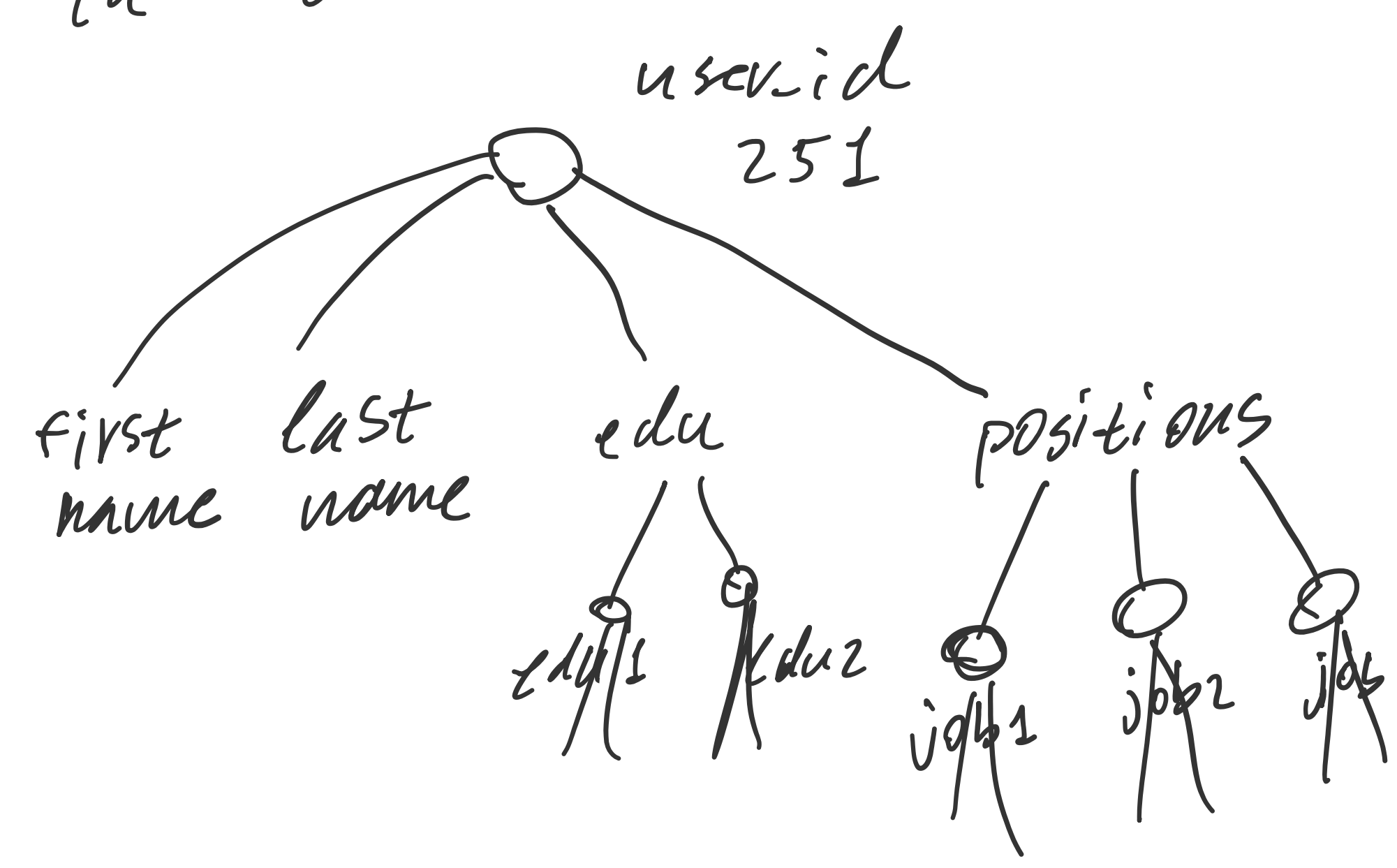
```
{
  "user_id": 251,
  "first_name": Bill,

}
```

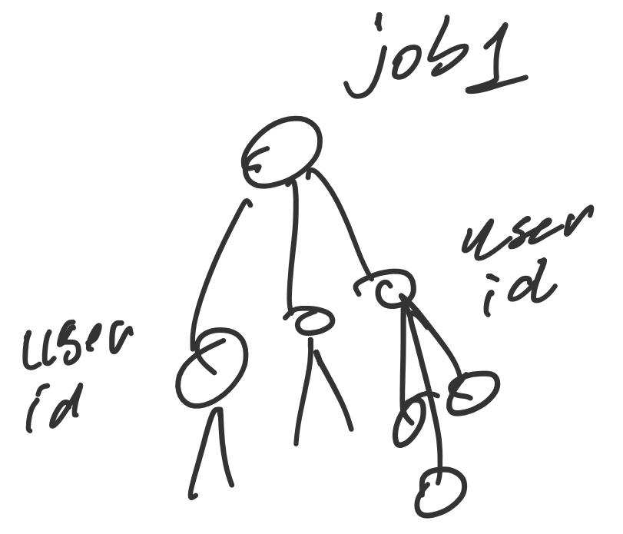
Hierarchical

1 - *

on write

imperative QL





Network

1 - *

* - *

On write

CODASYL model

foreign key ~ links

access path
imperative QL) user code

job {

}

Document

1 - *

on read

imperative QL

```
profile {
  "user_id": 251,
  "first_name": Bill,
  "positions": [ {"job": "co-founder", "company": "Microsoft"},
                 { . - - ... }
               ],
  "education": [...]
}
```

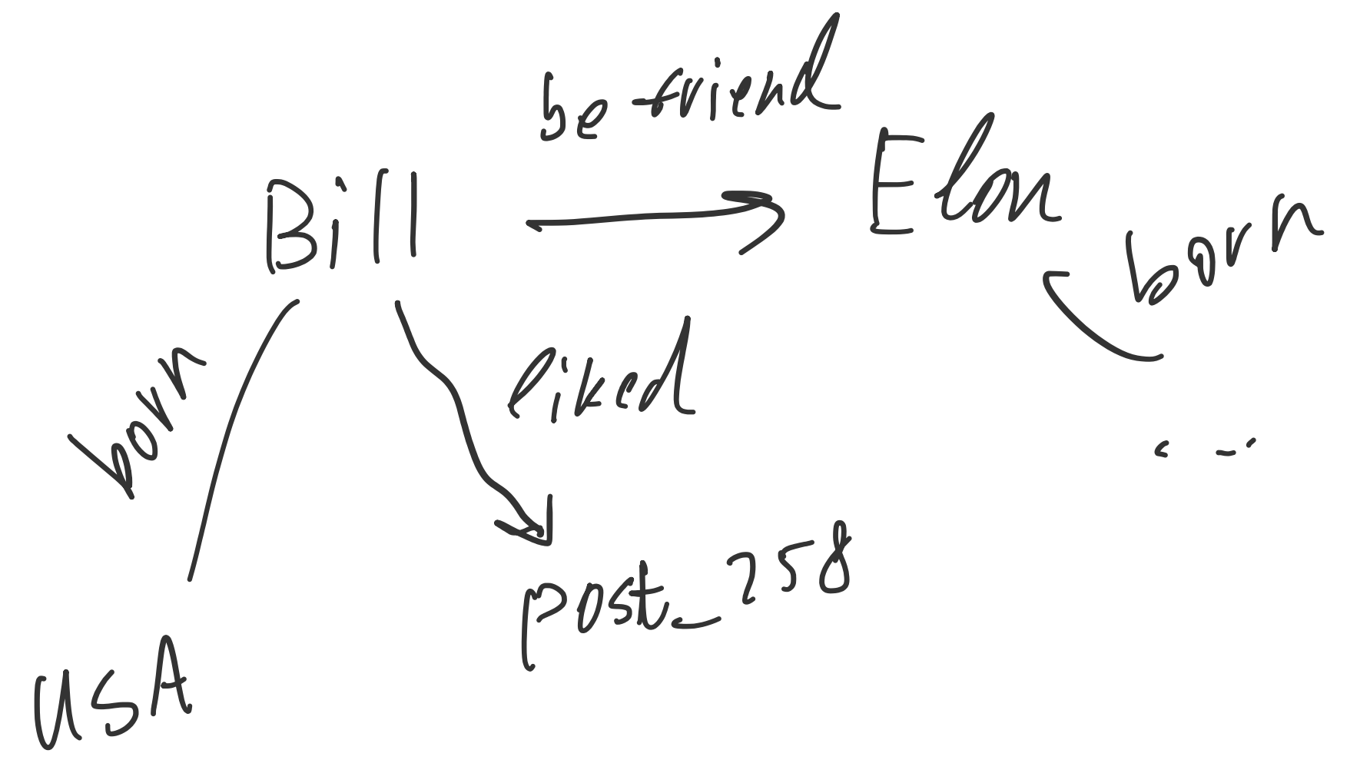
Graph

1 - *

* - *

non-homogeneous data

declarative QL



property graph — CYPHER QL

triple-store — SparQL

(subject, predicate, object)
(lucy, age, 33)
(lucy, married_to, alan)

Relational

- + general-purpose optimizer
- + joins
- - ORM
  - normalization
    - - joins + writes
  - denormalization
    - - update several places + read

Hierarchical

- + easy
- - no joins
- - copies of fields

Network

- + fast query
- - complex querying
- - expensive mutations

Document

- + flexibility
- + no mismatch
- + locality
- - no joins

Graph

- + flexibility
- + relational representation
- - work overheads

среда, 5 октября 2022 г.   19:50

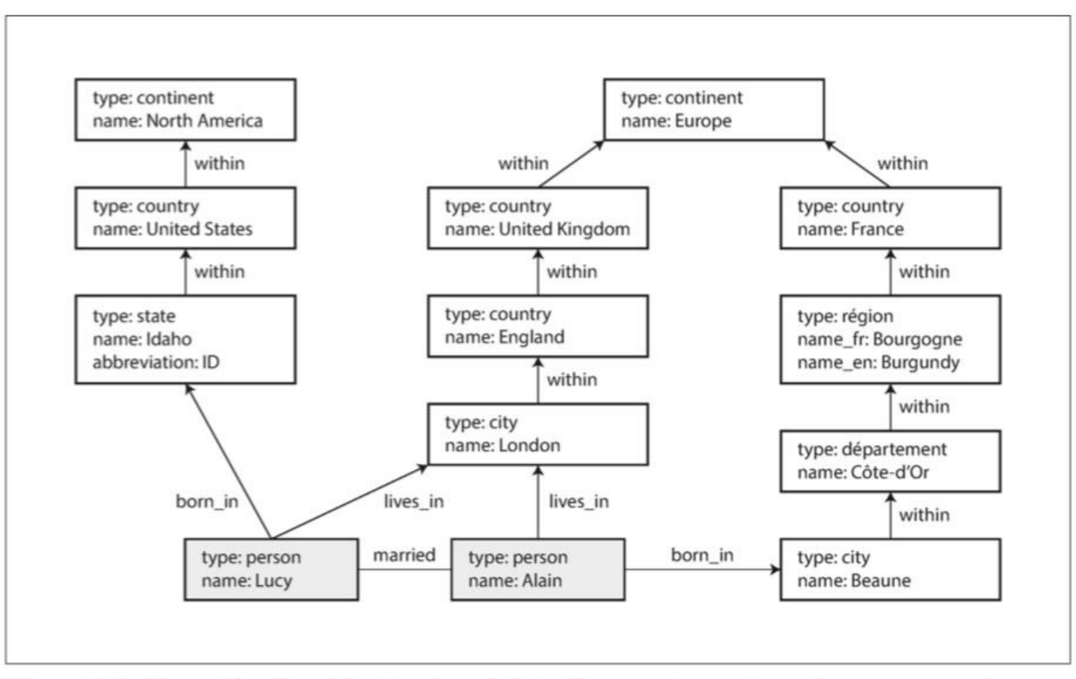


Figure 2-5. Example of graph-structured data (boxes represent vertices, arrows represent edges).

```
CREATE TABLE vertices (
    vertex_id   integer PRIMARY KEY,
    properties  json
);

CREATE TABLE edges (
    edge_id     integer PRIMARY KEY,
    tail_vertex integer REFERENCES vertices (vertex_id),
    head_vertex integer REFERENCES vertices (vertex_id),
    label       text,
    properties  json
);

CREATE INDEX edges_tails ON edges (tail_vertex);
CREATE INDEX edges_heads ON edges (head_vertex);
```

```
CREATE
  (NAmerica:Location {name:'North America', type:'continent'}),
  (USA:Location      {name:'United States', type:'country'  }),
  (Idaho:Location    {name:'Idaho',         type:'state'    }),
  (Lucy:Person       {name:'Lucy' }),
  (Idaho) -[:WITHIN]->  (USA)  -[:WITHIN]-> (NAmerica),
  (Lucy)  -[:BORN_IN]-> (Idaho)
```

```
MATCH
  (person) -[:BORN_IN]->  () -[:WITHIN*0..]-> (us:Location {name:'United States'}),
  (person) -[:LIVES_IN]-> () -[:WITHIN*0..]-> (eu:Location {name:'Europe'})
RETURN person.name
```

```
WITH RECURSIVE

  -- in_usa is the set of vertex IDs of all locations within the United States
  in_usa(vertex_id) AS (
      SELECT vertex_id FROM vertices WHERE properties->>'name' = 'United States' ❶
    UNION
      SELECT edges.tail_vertex FROM edges ❷
        JOIN in_usa ON edges.head_vertex = in_usa.vertex_id
        WHERE edges.label = 'within'
  ),

  -- in_europe is the set of vertex IDs of all locations within Europe
  in_europe(vertex_id) AS (
      SELECT vertex_id FROM vertices WHERE properties->>'name' = 'Europe' ❸
    UNION
      SELECT edges.tail_vertex FROM edges
        JOIN in_europe ON edges.head_vertex = in_europe.vertex_id
        WHERE edges.label = 'within'
  ),

  -- born_in_usa is the set of vertex IDs of all people born in the US
  born_in_usa(vertex_id) AS ( ❹
    SELECT edges.tail_vertex FROM edges
      JOIN in_usa ON edges.head_vertex = in_usa.vertex_id
      WHERE edges.label = 'born_in'
  ),
```

```
  -- lives_in_europe is the set of vertex IDs of all people living in Europe
  lives_in_europe(vertex_id) AS ( ❺
    SELECT edges.tail_vertex FROM edges
      JOIN in_europe ON edges.head_vertex = in_europe.vertex_id
      WHERE edges.label = 'lives_in'
  )

SELECT vertices.properties->>'name'
FROM vertices
-- join to find those people who were both born in the US *and* live in Europe
JOIN born_in_usa     ON vertices.vertex_id = born_in_usa.vertex_id ❻
JOIN lives_in_europe ON vertices.vertex_id = lives_in_europe.vertex_id;
```

```
@prefix : <urn:example:>.
_:lucy     a       :Person.
_:lucy     :name   "Lucy".
_:lucy     :bornIn _:idaho.
_:idaho    a       :Location.
_:idaho    :name   "Idaho".
_:idaho    :type   "state".
_:idaho    :within _:usa.
_:usa      a       :Location.
_:usa      :name   "United States".
_:usa      :type   "country".
_:usa      :within _:namerica.
_:namerica a       :Location.
_:namerica :name   "North America".
_:namerica :type   "continent".
```

Example 2-8. The data of Example 2-7, expressed using RDF/XML syntax

```
<rdf:RDF xmlns="urn:example:"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

  <Location rdf:nodeID="idaho">
    <name>Idaho</name>
    <type>state</type>
    <within>
      <Location rdf:nodeID="usa">
        <name>United States</name>
        <type>country</type>
        <within>
          <Location rdf:nodeID="namerica">
            <name>North America</name>
            <type>continent</type>
          </Location>
        </within>
      </Location>
    </within>
  </Location>

  <Person rdf:nodeID="lucy">
    <name>Lucy</name>
    <bornIn rdf:nodeID="idaho"/>
  </Person>
</rdf:RDF>
```

```
PREFIX : <urn:example:>

SELECT ?personName WHERE {
  ?person :name ?personName.
  ?person :bornIn  / :within* / :name "United States".
  ?person :livesIn / :within* / :name "Europe".
}
```