Akvo Lumen

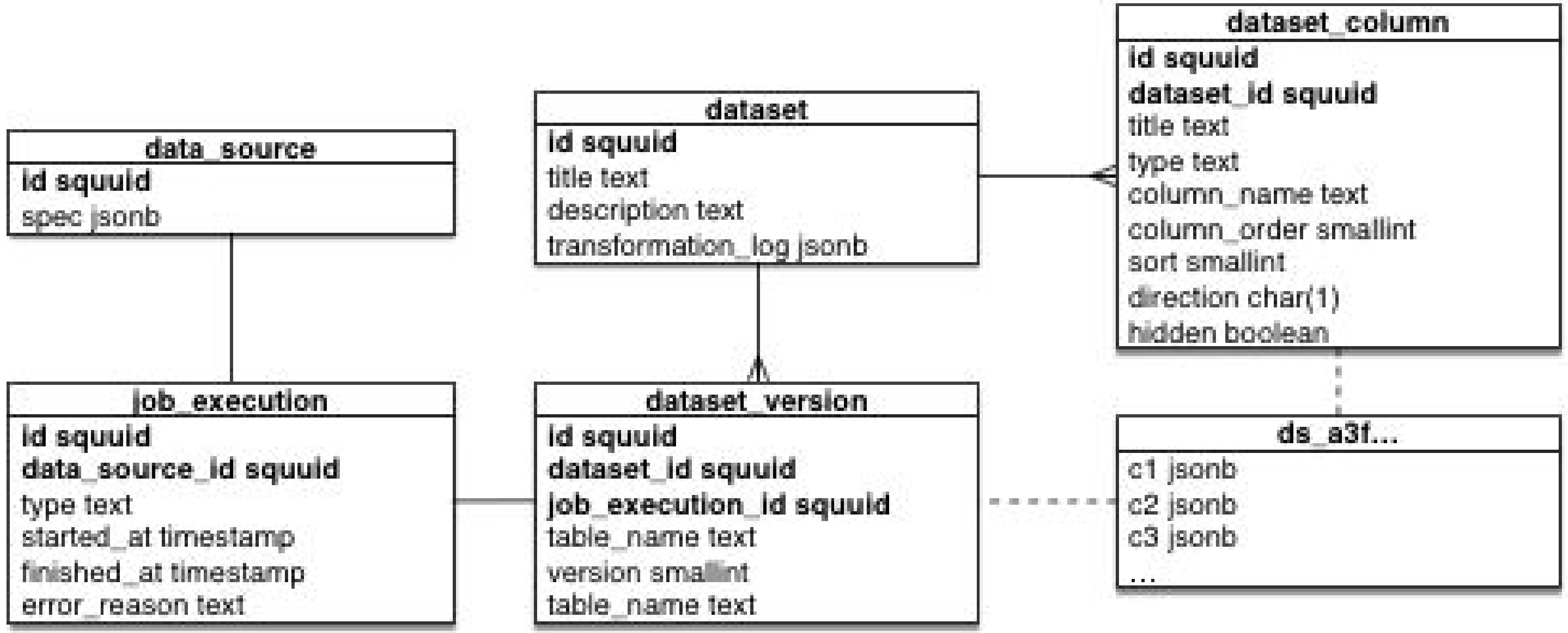
Data model and internals

Overview

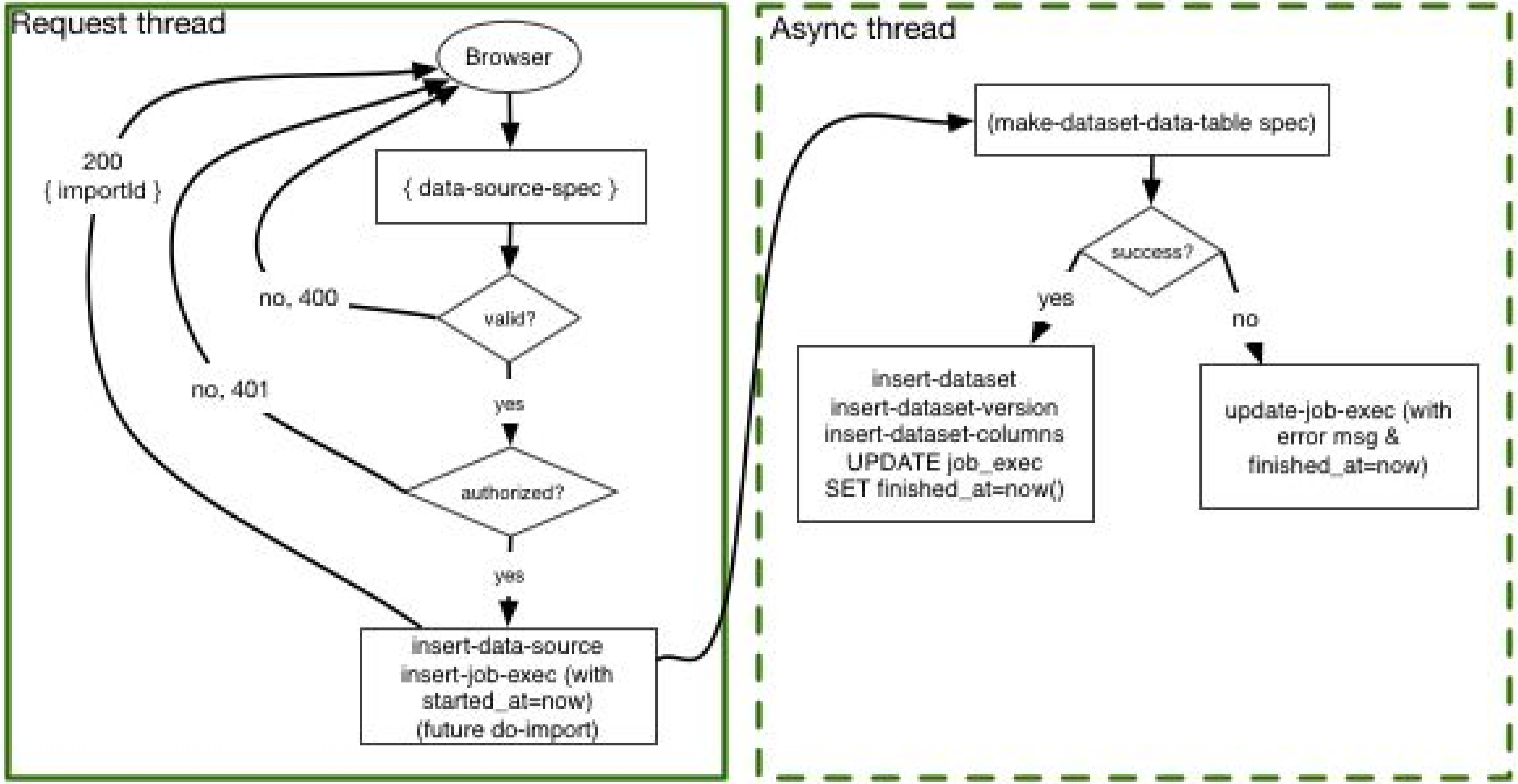
- Capture
  - Import datasets from differents sources
  - Transform, join and aggregate datasets
- Understand
  - Visualise datasets with charts and maps
- Share
  - Publish visualisations and dashboards

[frontend demo]

Datasets and Imports



IMPORT


Request thread
Browser
{ data-source-spec }
valid?
no, 400
yes
authorized?
no, 401
yes
200
{ importId }
insert-data-source
insert-job-exec (with
started_at=now)
(future do-import)
Async thread
(make-dataset-data-table spec)
success?
yes
no
insert-dataset
insert-dataset-version
insert-dataset-columns
UPDATE job_exec
SET finished_at=now()
update-job-exec (with
error msg &
finished_at=now)

[curl demo]

Transformations

$$f(dv^1) = dv^2$$

Transformations

transformation log application

$$f(dv^1) = dv^2$$

dataset v1

dataset v2

Concepts

- A dataset has `transformations` as property
- The `transformations` is a JSON encoded “payload”
- The application of each entry in `transformations` must succeed to continue with the process (fail early)
- The application of the transformation log is *“atomic”*
- Create a new *“dataset version”* only if the log got fully applied
- User-defined functions are written in JavaScript

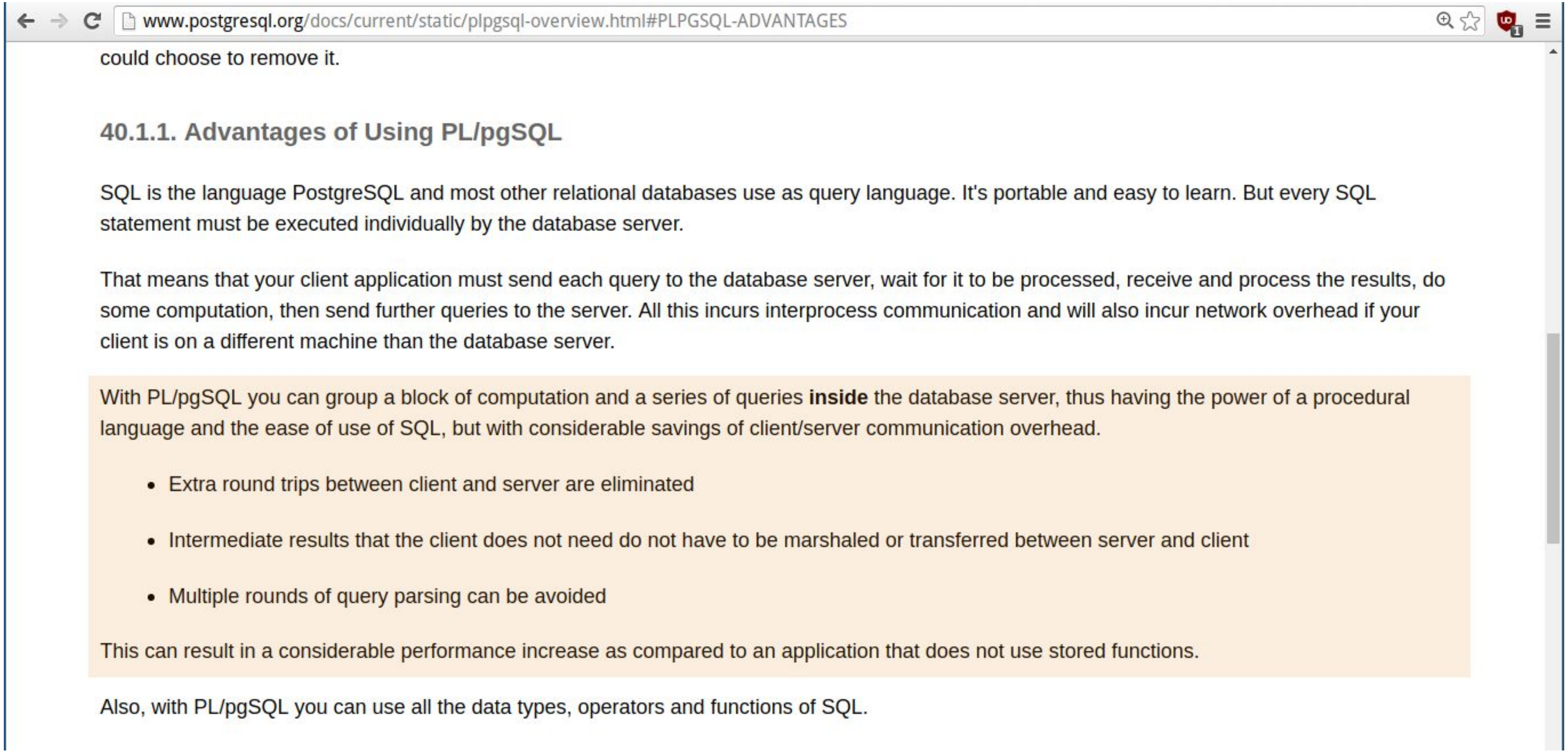
Transformation types

- Column based: change data type, title-case, lower-case, trim white spaces, split column, derived column, etc
- Row based: filter, drop/keep rows
- Metadata (implementation detail) - sort, sort direction, filter, hide, column order

Failed implementation (Round 1)

- All transformations are executed in the same runtime/engine on the application server (JDK 8 Nashorn)
- Core transformation functions are JavaScript functions

- Performance over a large dataset is unacceptable due to too much client-server round-trip
- Update `to_number()` in JS over 1M rows took 50min (using different strategies, single transaction, multiple transactions, fsync disabled in PG)


www.postgresql.org/docs/current/static/plpgsql-overview.html#PLPGSQL-ADVANTAGES

could choose to remove it.

40.1.1. Advantages of Using PL/pgSQL

SQL is the language PostgreSQL and most other relational databases use as query language. It's portable and easy to learn. But every SQL statement must be executed individually by the database server.

That means that your client application must send each query to the database server, wait for it to be processed, receive and process the results, do some computation, then send further queries to the server. All this incurs interprocess communication and will also incur network overhead if your client is on a different machine than the database server.

With PL/pgSQL you can group a block of computation and a series of queries **inside** the database server, thus having the power of a procedural language and the ease of use of SQL, but with considerable savings of client/server communication overhead.

- Extra round trips between client and server are eliminated
- Intermediate results that the client does not need do not have to be marshaled or transferred between server and client
- Multiple rounds of query parsing can be avoided

This can result in a considerable performance increase as compared to an application that does not use stored functions.

Also, with PL/pgSQL you can use all the data types, operators and functions of SQL.

Current implementation (Round 2)

- PL/pgSQL when possible
  - `to_number, lower, upper, initcap`
- Transformations application is now a set of *"SQL calls"*

```json
{
    "op": "core/to-titlecase",
    "args": {
     "columnName": "c1",
     "defaultValue": "identity"
    },
    "onError": "default-value"
}
```

```sql
UPDATE ds_uuid_1
   SET c1 = to_titlecase(c1);
```

Trivia

“The **identity transform** is a data transformation that copies the source data into the destination data without change”

https://en.wikipedia.org/wiki/Identity_transform

- `core/to-titlecase`
- `core/to-lowercase`
- `core/to-uppercase`
- `core/change-data-type`
- `core/sort-column`
- `core/remove-sort`
- `core/filter`
- `core/trim`