Query language for relational databases

Author: David Koňařík **Supervisor**: Mgr. Tomáš Petříček, Ph.D.

Currently all popular relational database management systems (RDBMSs) use SQL as their query language, and most SQL alternatives are implemented by translation into SQL.

In our thesis, we looked at SQL's flaws and existing alternatives, designed and described PPPQL, a new query language, and implemented it directly into the Postgres RDBMS as an extension.

Why not SQL?

- SQL queries are composed of fixed parts in a fixed order.
- SQL query syntax doesn't match the semantic order of operations.
- SQL has a number of different kinds of expressions, which can only be used in specific contexts.
- SQL doesn't offer convenient abstraction mechanisms.
- SQL's features are generally defined as new syntax, making the language incohesive.

Why PPPQL

- PPPQL is based on an extensible pipeline concept.
- PPPQL's syntax is simple and directly reflects the semantic order of operations.
- PPPQL unifies all disparate multi-value concepts into one set-of-values concept.
- PPPQL's integration into Postgres allows it to report user-friendly errors, so users don't have to debug any intermediate SQL.

Postgres extension

PPPQL is implemented as a Postgres extension, using a lightweight patch to replace the stock parser and analyser.

Code can be sent via the existing Postgres protocol, simply by starting a query with "`PQL`". Results are returned as if the user had sent an SQL query.

Links

Code repository: `https://gitlab.mff.cuni.cz/konarid/pppql`

Homepage: `https://dvdkon.ggu.cz/pppql`

The thesis is available from the above link and contains comparisons with other relational query languages, more example queries, a full specification of PPPQL and an overview of its implementation.

Basic expressions

PPPQL reuses Postgres' expression semantics, and freestanding expressions are allowed:

```
# 2 + 2
4
# upper("Hello") || " world!"
HELLO world!
```

Operations on scalars can be automatically lifted to work elementwise on sets:

```
# [1,2,3] + 2
{3,4,5}
```

Tables are also expressions, represented as Postgres arrays of records:

```
# buildings
{"(Karlín,MFF,Praha)","(Trója,MFF,Praha)",...}
```

Simple queries

Queries in PPPQL are composed from filters, starting with a data source like `from`, and then further adding modifying filters, like `where`, `order`, `group`, `let`, `limit` and `select`.

A query can consist of only one filter, or multiple filters combined in basically any order:

```
# | from buildings
  | where faculty = "MFF"
  | let un := upper(name)
  | select un, buildings.city

MALÁ STRANA | Praha
KARLÍN      | Praha
TRÓJA       | Praha
```

Grouping queries

In SQL, grouping rows by a key creates a special kind of value: Grouped columns can only be used as arguments to aggregate functions. In PPPQL, grouping simply gives the user the grouped rows as an array:

```
# | from measurements
  | group day := date(timestamp)
  | select day, measurements.temperature

2020-01-30 | {21,22}
2020-01-31 | {21,20}
2020-01-03 | {18,18}
```

Postgres aggregate functions are then represented as functions taking a set argument:

```
# sum([1,2,3,4])
10
```