PRQL

a simple, powerful, pipelined SQL replacement

Aljaž Mur Eržen

Compiler developer @EdgeDB

August 30, 2023

```
from albums
filter album_id > 100
sort albums.title
take 10
join artists (==artist_id)
select {
    albums.album_id,
    albums.title,
    f"Artist name: {artist.name}",
}
```

Why?

There are transtion costs!

Overview

1. Flaws of SQL
2. Language for relations
3. Compiling queries
4. PRQL, the project

A deef dive into

Flaws of SQL

Origins of the relational model

1970, Edgar F. Codd: abstraction over data storage

$\rightarrow$ Tuple relational calculus

1974, Donald D. Chamberlin & Raymond F. Boyce: SEQUEL

$\rightarrow$ Not a “proper” programming language

Not really composable

```
SELECT album_id, COUNT(*) AS track_count
FROM tracks GROUP BY album_id
```

Not really composable

```
SELECT i.album_id, i.track_count, a.artist_id
FROM (
  SELECT album_id, COUNT(*) AS track_count
  FROM tracks GROUP BY album_id
) AS i
JOIN albums a USING (album_id)
```

Patched syntax

```
SELECT
  SUM(total)
FROM
  invoices
```

Patched syntax

```
SELECT
  total / SUM(total) OVER () AS normalized_total
FROM
  invoices
```

Patched syntax

```
SELECT DISTNICT name
FROM invoices
```

Patched syntax

```
SELECT EVALUATE TYPE-EMPLOYEE
  WHEN "F"
    MOVE "FULL TIME" TO EMP-TYPE-PR
  WHEN "P"
    MOVE "PART TIME" TO EMP-TYPE-PR
  WHEN "C"
    MOVE "CONSULTANT" TO EMP-TYPE-PR
  WHEN OTHER
    MOVE "INVALID" TO EMP-TYPE-PR
```

Patched syntax

Too much syntax

... but also ...

Not enough syntax

Name resolution

```
SELECT title AS title_alias
FROM albums
```

Name resolution

```
SELECT title AS title_alias
FROM albums
WHERE title_alias LIKE 'Do I Wanna %'
GROUP BY title_alias
ORDER BY title_alias
```

Name resolution

More rules:

– ORDER BY positionals

– Correlated subqueries

– LATERAL

Relations vs scalars

```
SELECT * FROM table

SELECT count(*) FROM table
```

Relations vs scalars

```
SELECT emp_id FROM emp WHERE role = 'manager'
```

Relations vs scalars

```
SELECT *
FROM emp
WHERE emp_id = (
    SELECT emp_id FROM emp WHERE role = 'manager'
)
```

Relations cannot be ordered

```
SELECT * FROM albums ORDER BY title
```

Relations cannot be ordered

```
SELECT
  *,
  ... AS my_col
FROM (
    SELECT * FROM albums ORDER BY title
) inner
```

Relations cannot be ordered

```
SELECT
  *,
  ROW_NUMBER()
    OVER (ORDER BY artist_id) AS my_col
FROM (
    SELECT * FROM albums ORDER BY title
) inner
```

Relations cannot be ordered

SELECT returns an **ordered set**

FROM pulls-in a **set**

Relations cannot be ordered

```
SELECT
  *,
  ... AS my_col
FROM (
    SELECT *
    FROM albums
) inner
ORDER BY title
```

Relations cannot be ordered

```
SELECT
  *,
  ... AS my_col
FROM (
    SELECT * FROM albums ORDER BY title LIMIT 10
) inner
ORDER BY title
```

Identity of aggregation

```
SELECT SUM(cost) FROM expenses WHERE FALSE
```

Two possible behaviors: NULL or 0

Both valid

Identity of aggregation

“Every marble in this bag is black”
… but the bag is empty.

Ancient greeks say FALSE

Modern logic says TRUE

SQL says NULL

Identity of aggregation

Homomorphism of addition

```
SUM([1]) + SUM([4, 5]) = SUM([1, 4, 5])

                  1 + 9 = 10
```

Identity of aggregation

```
SUM([1]) + SUM([]) = SUM([1])

       1 +    ?    = 1

        -> SUM([]) = 0
```

identity of addition

Identity of aggregation

```
COUNT([])       = 0
ARRAY_AGG([])   = []
SUM([])         = 0
ANY([])         = false
EVERY([])       = true
STRING_AGG([]) = ''
```

Dialects

Differences in:

- syntax (TOP vs LIMIT)

- available functions

- available data types

Dialects

A class of languages

There is a standard

Slight diviations

Dialects

Different:

- priorities

- backward compatibility guarantees

- implementation limitations

Dialects

No clear & robust specification

Compilers could:

- adapt query to target database

- produce error early

Design of a new

Language for relations

Tuple relational calculus

Relation $\sim$ a set of tuples

```
SELECT track_id, name, title
FROM tracks, albums
WHERE tracks.album_id = albums.album_id
```

Tuple relational calculus

Relation $\sim$ a set of tuples

$T := \text{tracks}$

$A := \text{albums}$

$\pi_{track\_id,name,title}(T * A)$

Data model

Functions

Transforms

Grouping

Nulls

Modern micro-features

The task of a query lanuage

Leaky abstractions

A better database interface

prqlc and its IRs,

general architecture

how to use it

how to contribute