Why we defined a metalanguage for SQL

Lewis Hemens

We need a **scalable** solution for managing data transformation processes that works for data engineers, analysts and scientists

Why we **love** SQL

SQL is growing in popularity thanks to modern data warehouses

- ➔ A common language for data definitions across roles
- ➔ Modern warehouse SQL engines scale extremely well
- ➔ Easy to iterate, thanks execution usually being one-click
- ➔ Relatively easy to debug

But it has some problems...

Why doesn't SQL scale?

It's hard to adopt **software engineering best practices**

- ➔ Release processes
- ➔ Version control
- ➔ Unit tests
- ➔ Code reuse

Why are these hard, and how can we fix them?

Understanding SQL

dataform

SQL is a **declarative** query language

Declarative programming

When you say what you want

Imperative programming

When you say how to get what you want

Advantages of being declarative

The fact that SQL is declarative means it has many benefits

- ➔ SQL queries can be **parallelized**
- ➔ SQL queries can be automatically **optimized**
- ➔ For most SQL statements there are **no side effects**
- ➔ SQL queries are guaranteed to **eventually terminate**

SQL is **not** a programming language

SQL is few features short of being a programming language

- ➔ SQL has little if any control flow
- ➔ There is no recursion or iteration*
- ➔ SQL is **declarative and static**

*Some flavors of SQL (e.g. T-SQL) add these and are turing complete

Example: writing **reusable** code

```
select
  floor(age / 5) * 5 as age_group,
  count(1) as user_count
from users
group by age_group
```

Example: writing **reusable** code

```
select
  floor(age / 5) * 5 as age_group,
  count(1) as user_count
from users
group by age_group
```

We can't **reuse** this query: the input is fixed 😭

Example: writing **testable** code

```
select
  floor(age / 5) * 5 as age_group,
  count(1) as user_count
from users
group by age_group
```

We can't **test** this query for the same reason 😭

Example: writing **iterative** code

```
user_tables = ["users", "user_stats", "user_events"]

for table in user_tables:
    delete from table
    where user_id in (
        select user_id from gdpr_deletion_requests
    )
```

Example: writing **iterative** code

```
user_tables = ["users", "user_stats", "user_events"]

for table in user_tables:
    delete from table
    where user_id in (
        select user_id from gdpr_deletion_requests
    )
```

Metaprogramming to the rescue

What is metaprogramming?

Metaprogramming is a programming technique in which computer programs have the ability to treat other programs as their data

Metaprogramming can be used to **move computations from run-time to compile-time**

Metaprogramming example

```
select
  floor(age / 5) * 5 as
age_group,
  count(1) as user_count
from users
group by age_group
```

```
function ageDist(input, bucket = 5) {
  return `
      select
          floor(age / ${bucket}) *
${bucket} as age_group,
          count(1) as user_count
      from ${input}
      group by age_group`;
}
```

Fixing SQL with meta programming

- ➔ Enable code reuse through parameterizable functions
- ➔ Allow *some* imperative programming
- ➔ Introduce *some* control flow
- ➔ **Keep our code declarative at run-time**

Dataform

framework

An open-source framework and metalanguage for SQL

Dataform framework overview

- ➔ Makes it easy to write **parameterized SQL**
- ➔ Enables **code reuse**
- ➔ APIs to help build **directed acyclic graphs**
- ➔ Support for writing **data assertions**
- ➔ Support for writing **SQL unit tests**
- ➔ APIs for **documenting** datasets
- ➔ Support for managing multiple **environments**