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**

Dataform compilation process

- ➔ **Imperative**
- ➔ Sandboxed
- ➔ JavaScript
- ➔ SQLX

- ➔ **Declarative**
- ➔ Sandboxed
- ➔ **Pure SQL**
- ➔ JSON

- ➔ Declarative
- ➔ **!Sandboxed**
- ➔ Pure SQL
- ➔ JSON

By introducing a compilation step, we maintain a static, declarative runtime format

Dataform example: Publishing a table

```
// copy_users_table.js

publish("copy_users_table")
  .type("table")
  .query(ctx => `
    select *
    from ${ctx.ref("users")}`
  );
```

```
-- copy_users_table.sqlx

config {
  type: "table"
}

select *
from ${ref("users")}
```

(Our SQL queries are now parameterized!)

Dataform compiled graph


```
{
  "name": "dataform_dc_talk.copy_users_table",
  "dependencies": ["dataform_dc_talk.users"],
  "type": "table",
  "target": {
      "schema": "dataform_dc_talk",
      "name": "copy_users_table"
  },
  "query": "select * from dataform_dc_talk.users",
}
```

Running dataform projects

```
$ dataform compile
Compiling...

Compiled 56 action(s).
35 dataset(s):
  dataform_data.organisations [view]
  dataform_data.project_users [view]
  dataform_data.projects [view]
  dataform_data.sessions [table]
  dataform_data.users [view]
  ...
```

```
$ dataform run
Compiling...
Compiled successfully.
Running...

Dataset created: dataform_data.organisations [view]
Dataset created: dataform_data.project_users [view]
Dataset created: dataform_data.projects [view]
Dataset created: dataform_data.sessions [table]
Dataset created: dataform_data.users [view]
...
```

Dataform framework summary

It’s basically a SQL compiler.

- ➔ We can write any* code we like during the **compilation phase**
- ➔ Dataform’s runtime format is declarative, pure SQL with only non-iterative control structures during the **runtime phase**

A note on reproducibility

- ➔ Executing compile on the same project with the same parameters twice should always yield the same result
- ➔ Sandboxing helps enforce this, no network requests, file reads, or or DB access possible
- ➔ User should avoid non deterministic algorithms (e.g. Math.random())

Dataform examples

dataform

For loops

```
const userTables = ["users", "user_stats", "user_events"];

userTables.forEach(tableName =>
  operate(
    `${tableName}_gdpr_cleanup`,
    ctx => `
        delete from ${ctx.ref(tableName)}
        where user_id in (
            select user_id
            from ${ctx.ref("gdpr_deletion_requests")}
        )`
  )
);
```

Unit testing

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


publish("users_by_age")
  .query(ctx => ageDist(ctx.ref("users"), 5))
  .type("table");
```

Unit testing

```
test("ageDist_test")
  .query(ageDist(`(
    select 15 as age union all
    select 21 as age union all
    select 24 as age)`,
    10
  ))
  .expected(`
    select 10 as age_group, 1 as user_count union all
    select 20 as age_group, 2 as user_count
`);
```

Can be run with:
`dataform test`

Environment sampling

```
publish(
  "sourcetable_view",
  ctx => `
    select *
    from ${ctx.ref("sourcetable")}
    where ${
    ctx.env === "staging"
        ? `rand() < ${constants.stagingSamplingRate}`
        : "true"
  }`
);
```

Loading data from S3

```
s3_load_csv("load_example_csv", {
  path: "s3://.../sample_data.csv",
  schema: {
    country: "varchar(256)",
    revenue: "float8"
  },
  role: "arn:aws:iam::...",
  ignoreheader: true
});
```

```
create table
dataform_dc_talk.load_example_csv (
    country varchar(256),
    revenue FLOAT8
);
copy
dataform_dc_talk.load_example_csv
from 's3://.../sample_data.csv'
iam_role 'arn:aws:iam::...'
ignoreheader 1
delimiter ',';
```

Dataform

Web

An collaborative IDE and deployment platform for dataform projects

Dataform Web

What good is a new language without an IDE?

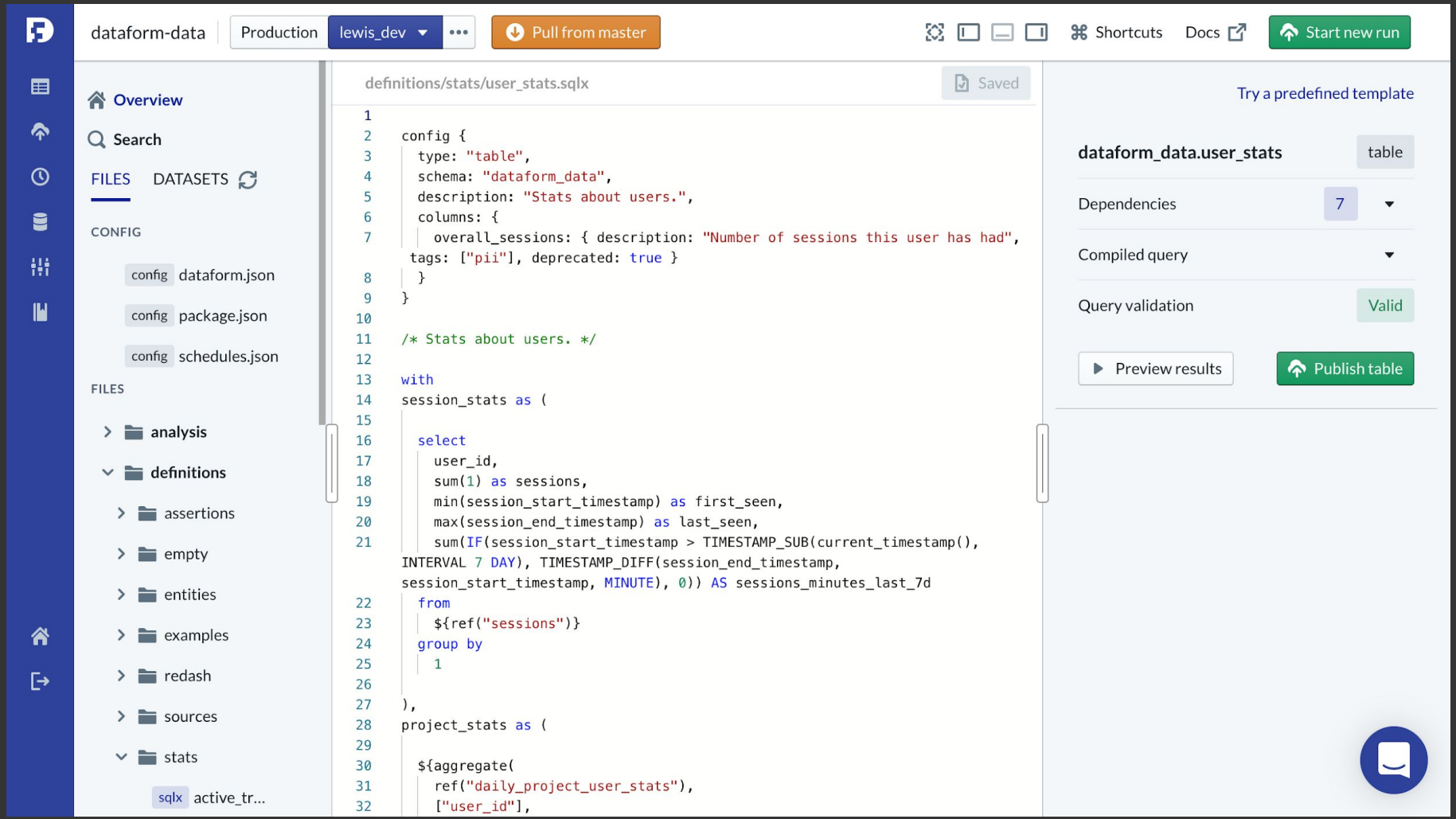
Dataform Web: a complete toolkit to manage your data warehouse

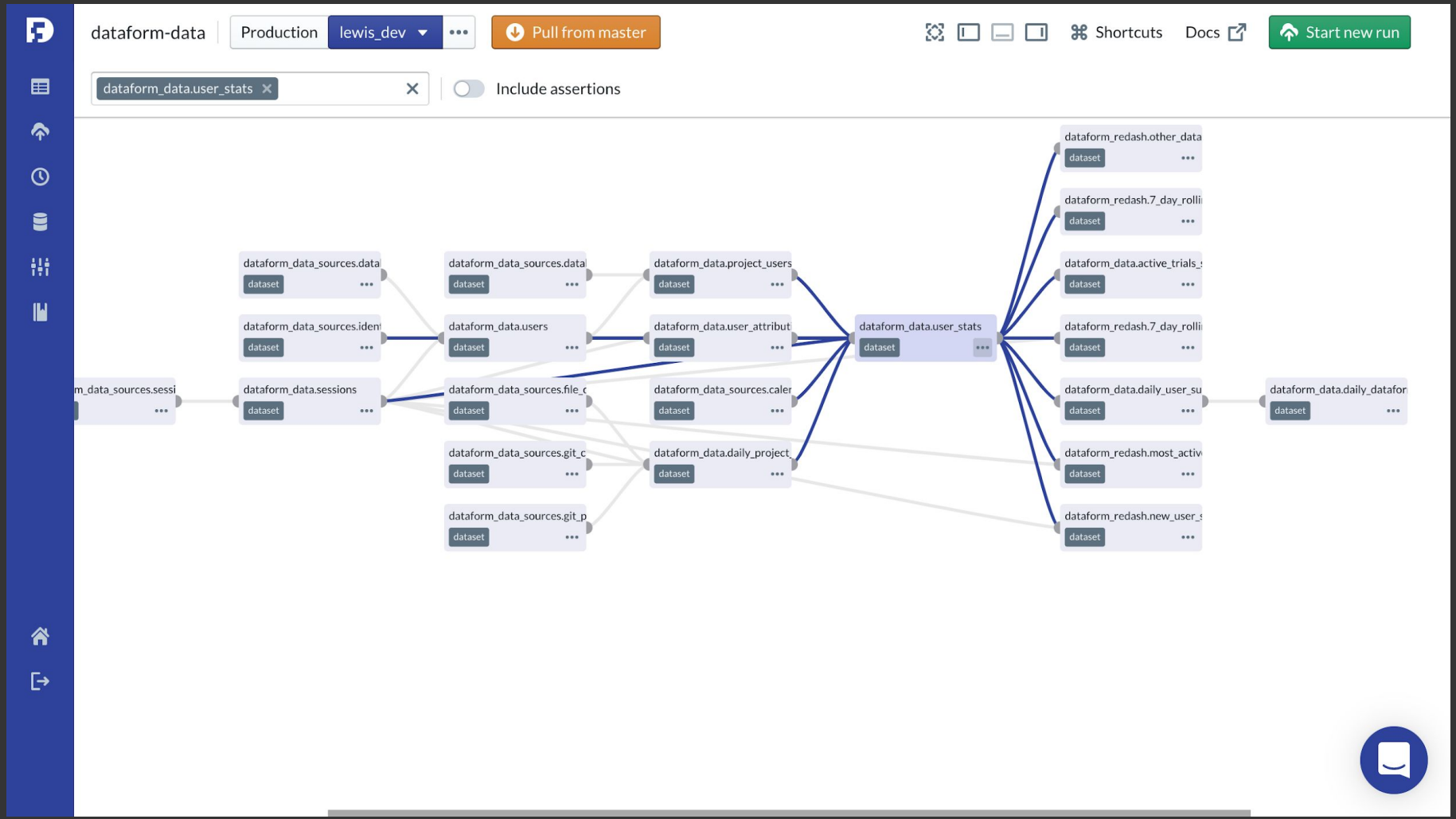
IDE for dataform projects

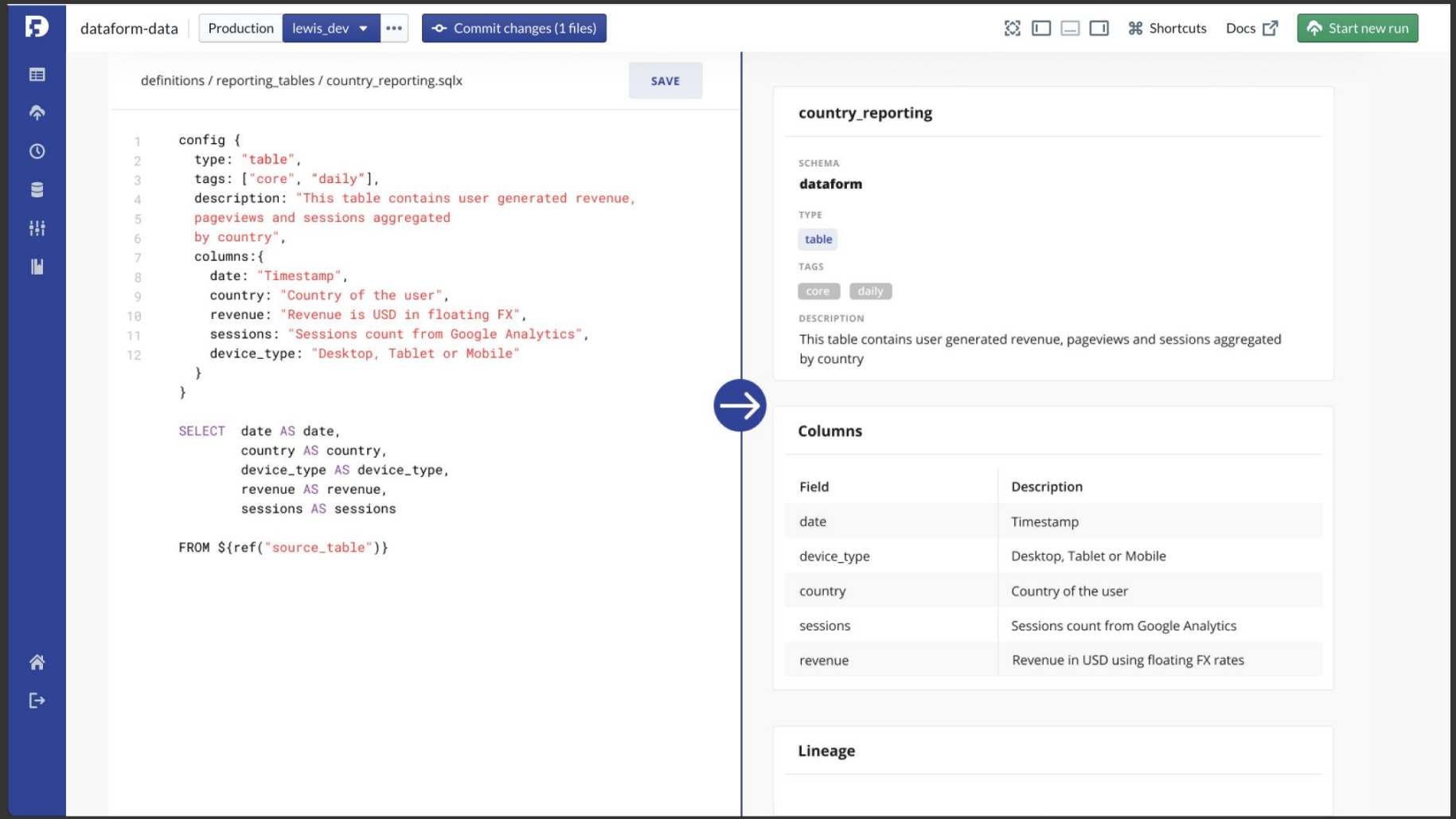
➔ Compiles and validates SQL graph in real time
➔ Full integration with Git, user branches and pull requests
➔ Managed continuous deployment and environments

But also:

- **Pipeline orchestration**, run logs and notifications
- Documentation and **Data catalog**


dataform-data
Production
lewis_dev
Pull from master
Shortcuts
Docs
Start new run
Overview
Search
FILES
DATASETS
CONFIG
config dataform.json
config package.json
config schedules.json
FILES
analysis
definitions
assertions
empty
entities
examples
redash
sources
stats
sqlx active_tr...
definitions/stats/user_stats.sqlx
Saved
config {
type: "table",
schema: "dataform_data",
description: "Stats about users.",
columns: {
overall_sessions: { description: "Number of sessions this user has had", tags: ["pii"], deprecated: true }
}
}
/* Stats about users. */
with
session_stats as (
select
user_id,
sum(1) as sessions,
min(session_start_timestamp) as first_seen,
max(session_end_timestamp) as last_seen,
sum(IF(session_start_timestamp > TIMESTAMP_SUB(current_timestamp(), INTERVAL 7 DAY), TIMESTAMP_DIFF(session_end_timestamp, session_start_timestamp, MINUTE), 0)) AS sessions_minutes_last_7d
from
${ref("sessions")}
group by
1
),
project_stats as (
${aggregate(
ref("daily_project_user_stats"),
["user_id"],
Try a predefined template
dataform_data.user_stats
table
Dependencies
7
Compiled query
Query validation
Valid
Preview results
Publish table


dataform-data
Production
lewis_dev
Pull from master
Shortcuts
Docs
Start new run
dataform_data.user_stats
Include assertions
dataform_data_sources.data
dataform_data_sources.data
dataform_data.project_users
dataform_data_sources.iden
dataform_data.users
dataform_data.user_attribut
dataform_data.user_stats
dataform_redash.other_data
dataform_redash.7_day_rolli
dataform_data.active_trials_
dataform_redash.7_day_rolli
m_data_sources.sessi
dataform_data.sessions
dataform_data_sources.file_
dataform_data_sources.caler
dataform_data.daily_user_su
dataform_data.daily_datafor
dataform_data_sources.git_c
dataform_data.daily_project_
dataform_redash.most_activ
dataform_data_sources.git_p
dataform_redash.new_user_
dataset


dataform-data
Production
lewis_dev
Commit changes (1 files)
Shortcuts
Docs
Start new run
definitions / reporting_tables / country_reporting.sqlx
SAVE
config {
type: "table",
tags: ["core", "daily"],
description: "This table contains user generated revenue, pageviews and sessions aggregated by country",
columns:{
date: "Timestamp",
country: "Country of the user",
revenue: "Revenue is USD in floating FX",
sessions: "Sessions count from Google Analytics",
device_type: "Desktop, Tablet or Mobile"
}
}
SELECT date AS date,
country AS country,
device_type AS device_type,
revenue AS revenue,
sessions AS sessions
FROM ${ref("source_table")}
country_reporting
SCHEMA
dataform
TYPE
table
TAGS
core
daily
DESCRIPTION
This table contains user generated revenue, pageviews and sessions aggregated by country
Columns
Field
Description
date
Timestamp
device_type
Desktop, Tablet or Mobile
country
Country of the user
sessions
Sessions count from Google Analytics
revenue
Revenue in USD using floating FX rates
Lineage

Thanks!

Dataform framework: github.com/dataform-co/dataform

Dataform docs: docs.dataform.co

Examples from this talk: github.com/dataform-co/dataform-dc-talk

Dataform web IDE: dataform.co

Questions?