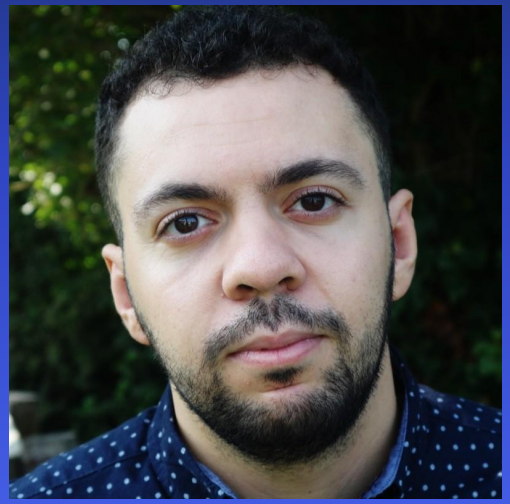
RivetSQL

DEFINE ONCE RUN ANYWHERE

Portable SQL pipelines for modern data platforms

DEVSUMMIT
<MOVE_
FORWARD/>



Massil CHABANE

Staff Data Engineer @Decathlon

DEVSUMMIT <MOVE_FORWARD/>

What is Rivet?

An open-source **Python framework** that separates **what** to compute from **how** to compute it.

Write your pipeline logic once as declarative joints, then run it on any engine without changing a single line.

```
$ pip install 'rivetsql[all]'
```

DuckDB Polars PySpark Postgres

Features

Multi-Engine

Swap between DuckDB, Spark, and Postgres without rewriting a single query.

Declarative

Define joints in SQL or Python. Rivet compiles an immutable execution plan.

Quality Checks

Assertions and Audits validate data before and after writes. Catch bad data early.

Built-in Testing

Validate logic using offline fixtures without a database. Fast and CI-friendly.

Plugin System

Extend with new engines (AWS, Databricks) and custom catalog adapters.

Interactive REPL

Full-screen TUI for exploring data, ad-hoc queries, and debugging.

The Paradox of Choice

Radical Simplicity

One engine for everything. Easy setup, but high vendor lock-in and performance "averaging".

Snowflake BigQuery

Best of Breed

Highly specialized tools. Superior performance, but absolute fragmentation of logic.

DuckDB Spark ClickHouse

RivetSQL removes the choice: Define logic once, execute where it's optimized.

Trap 1: Locked in Dialects

SQL is a "standard" in name only. Your logic gets trapped inside vendor-specific syntactic sugar, making migration nearly impossible.

- **JSON Parsing** differs wildly.
- **Date Math** is never consistent.
- **Type Casting** has proprietary safety functions.

1. JSON EXTRACTION

```
-- Snowflake
data:user:email::string
-- Postgres
data->'user'->>'email'
```

2. DATE TRUNCATION

```
-- Spark SQL
SELECT TRUNC(date, 'MM')
-- BigQuery
SELECT DATE_TRUNC(date, MONTH)
```

3. SAFE CASTING

```
-- Postgres
SELECT TRY_CAST(val AS INT)
-- BigQuery
SELECT SAFE_CAST(val AS INT64)
```

Trap 2: Orchestration Coupling

The *way* logic is executed becomes completely coupled to the scheduler. You are no longer writing SQL, you are writing framework config.

- **dbt:** Logic trapped in proprietary Jinja macros.
- **Airflow:** Hidden inside Python Operator classes.
- **Dagster:** Wrapped entirely in Python Asset decorators.

1. DBT (JINJA MACROS)

```
{% if is_incremental() %}
  WHERE updated_at > (SELECT MAX(...) FROM {{ this }})
{% endif %}
```

2. AIRFLOW (OPERATORS)

```
BigQueryExecuteQueryOperator(
  task_id='run_sales',
  sql="SELECT * FROM raw..."
)
```

3. DAGSTER (PYTHON WRAPPER)

```
@asset
def my_sql_asset(context):
  context.resources.snowflake.execute("SELECT...")
```

Trap 3: Infra Logic Leakage

Physical hardware constraints and engine tuning leak directly into your logical transformation code.

- **Spark:** Cluster shuffling and explicit partition tuning.
- **DuckDB:** Hardware pragmas to prevent local crashes.
- **Pandas:** Manual chunking loops for out-of-core data.

1. SPARK (CLUSTER SHUFFLING)

```
spark.conf.set("spark.sql.shuffle.partitions", 200)
df.repartition(50).write.save()
```

2. DUCKDB (HARDWARE LIMITS)

```
PRAGMA memory_limit='8GB';
PRAGMA threads=4;
```

3. PANDAS (OUT-OF-CORE BATCHING)

```
# Manual chunking to avoid OOM
for chunk in pd.read_csv("data.csv", chunksize=100000):
  process_logic(chunk)
```

Trap 4: Testing Friction

Local testing is a nightmare because logic is tied to infrastructure.

- **Mocking Pain:** Need a full local Spark/S3 mock just to test a filter.
- **Feedback Loop:** 5-minute wait for Cloud CI just to find a typo.

Production is the only "real" test.

The Philosophy

The Lore behind Rivet

Mechanical Fastening for Data Structures.

1. The Material: Catalogs

In mechanical fastening, you start with the **Material** (Steel Plates). In Rivet, these are your **Catalogs**.

- **Storage:** S3, Postgres, or Local Files are the sheets to be joined.
- **Agnostic:** Data is separate from the tool that shapes it.

2. The Joints: Logic

The **Joint** is the specific place where data meets. Rivet has 4 distinct types:

Source

Extraction

Transform

Pure SQL

Sink

Writing

Python

API/ML

3. The Fastener: Engines

The **Rivet Gun**. This is the **Engine** that physically performs the work.

- **Interchangeable:** Use a DuckDB gun for local, Spark for prod.
- **Portability:** The logic stays fixed while the engine changes.

Secret Sauce: Apache Arrow

The universal language for in-memory data.

- **Columnar Format:** Standardized memory layout shared by all.
- **Zero-Copy:** Move data from SQL to Python instantly without overhead.

Secret Sauce: SQLGlot

The universal translator for SQL dialects.

- **AST Parsing:** Breaks SQL into an Abstract Syntax Tree.
- **Transpilation:** Rewrites standard SQL to the native dialect at runtime.

ANSI SQL IN

```
SELECT date_trunc('month', d)
```

SPARK SQL OUT

```
SELECT trunc(d, 'MM')
```

Advanced: Query Fusion

Fusing adjacent nodes on the same engine into multi-CTE queries.

Advanced: Pushdowns

Moving filters upstream to where the data lives.

Quality Schema

Tests

DEVELOPMENT PHASE

Context: Offline / Local
Data: Mock Fixtures
Goal: Validate Logic Logic

Assertions

EXECUTION PHASE

Context: Inline / Pre-Write
Data: Computed Memory
Goal: Prevent Bad Writes

Audits

PERSISTENCE PHASE

Context: Post-Write / Scheduled
Data: Stored Material
Goal: Verify State & Freshness

Data Assertions

Run validation checks **inline** before the final write. If an assertion fails, the fastener is never driven home.

- Checks the in-memory result set.
- Supports not_null, unique, and row_count.

YAML JOINT CONFIG

```yaml
name: revenue_sink
type: sink
upstream: daily_revenue
quality:
  assertions:
    - type: not_null
      columns: [revenue]
    - type: row_count
      min: 1
```

Data Audits

Verify data integrity **after** persistence. Audits read from storage to ensure consistency and freshness over time.

- Checks the actual state of the Catalog.
- Monitors data drift and pipeline health.

YAML JOINT CONFIG

```yaml
name: revenue_sink
type: sink
quality:
  audits:
    - type: row_count
      min: 1
    - type: freshness
      column: order_date
      max_age_hours: 25
```

Thanks.