

□ FAST • LOCAL-FIRST • OPEN SOURCE

What if you could build complete ETL pipelines in Pure SQL...

Without Jinja templates,
without a cloud database tax,
and without JVM / Spark bloat?

□ Pure ANSI SQL

No Jinja, no complex macro spaghetti — clean SQL everywhere.

□ Multi-Source & Sink

Join Excel + Delta + Parquet + CSV; output to multiple destinations.

□ Apache DataFusion

Rust vectorized query engine executing at bare-metal speeds.

□□ Built-in Web IDE

Visual DAG canvas, live SQL validator, and telemetry monitoring.

Why is Modern Data Engineering So Unnecessarily Complex?

1. The Jinja Macro Quagmire

You wanted to write SQL, but ended up debugging nested `{{ ref(...) }}` and custom Jinja macros.

2. The Heavy Cloud Warehouse Tax

Forcing simple transformations into Snowflake or BigQuery burns compute credits and egress fees.

3. The Single-Source Bottleneck

dbt expects data to already live in one warehouse. Joining raw Excel + Parquet + Delta requires extra ETL tools.

4. The Single-Destination Constraint

Models in dbt / SQLMesh write to only one target table per file. Multi-destination publishing requires duplicate runs.

Pure SQL Everywhere. No Jinja. No Database Server.

Powered by Apache DataFusion & Apache Arrow columnar memory.

Capability	Traditional (dbt / SQLMesh)	det (Data Engineering Tool)
Language	SQL + Jinja templates	100% Pure ANSI SQL
Compute Engine	Remote Cloud DB (Snowflake)	Embedded Apache DataFusion (Rust)
Memory Format	Database proprietary	Apache Arrow zero-copy memory
Cloud Dependency	Mandatory active connection	Zero cloud lock-in / 100% Local-first
Intermediate DAG	Temporary DB tables/views	Zero-copy in-engine registered views

❑ How intermediate DAG stages work in det:

```
-- Step 1: Clean raw orders
SELECT order_id, customer_id, net_amount FROM raw_orders WHERE status = 'OK';

-- Step 2: Query Step 1 output directly without disk serialization!
SELECT customer_id, SUM(net_amount) AS ltv FROM clean_orders GROUP BY 1;
```

Extract & Join Across Multiple Formats in One Query

Query Parquet, Delta Lake, Excel, CSV, JSON, and XML in standard ANSI SQL.

▣ Delta Lake (ACID)

▣ Parquet

▣ Excel (.xlsx)

▣ CSV

▣ JSON / NDJSON

▣ XML

▣ Live Query Example: Joining Excel Quota with Delta Lake Transactions

```
SELECT
  s.store_id,
  s.store_name,
  q.q1_target_usd, -- Scanned lazily from Excel (.xlsx)
  SUM(o.net_amount) AS actual_usd, -- Scanned vectorized from Delta Lake
  ROUND(SUM(o.net_amount) / q.q1_target_usd * 100.0, 2) AS attainment_pct
FROM raw_stores s -- CSV customer catalog
JOIN excel_financial_targets q -- Multi-sheet Excel workbook
  ON s.store_id = q.store_id
JOIN delta_orders o -- ACID Delta Lake transaction table
  ON s.store_id = o.store_id
GROUP BY s.store_id, s.store_name, q.q1_target_usd;
```

Publish to Multiple Destinations From a Single Pipeline Step

dbt and SQLMesh tie one model to one table. det routes everywhere.

DAG Pipeline Step: `compute_customer_rfm_tiers`

Destination 1: Delta Lake → `dw_customer_rfm_delta`

ACID time-travel table for downstream analytical queries

Destination 2: Parquet → `dw_customer_rfm.parquet`

Snappy-compressed columnar file for fast cloud BI exports

Destination 3: Delimited CSV → `rfm_executive_report.csv`

Clean CSV report ready for business stakeholders & Excel

□ Declarative YAML Specification:

steps:

- name: `mart_customer_rfm`
sql: "SELECT customer_id, rfm_score, loyalty_tier FROM ranked_customers"
destinations:
 - name: `dw_customer_rfm_delta` # Delta Lake table (mode: overwrite)
 - name: `dw_customer_rfm_parquet` # Parquet file (mode: overwrite)
 - name: `rfm_report_csv` # CSV file (mode: overwrite)

Zero-Install Web IDE

Run Headless or Visually (`det gui`)

Everything you need to author, debug, monitor, and document pipelines.

▣ Visual DAG Canvas

Auto-generated Mermaid DAG topology graphs reflecting exact dependency and step flow in real-time.

▣ Real-Time SQL Runner

Integrated Monaco-grade SQL editor with syntax highlighting, column autocomplete, and pre-flight validation.

▣▣ Data Catalog Previews

Browse all 14 data sources with live schema types, column contracts, and instant table sample views.

▣▣ Telemetry & Monitoring

Inspect execution run histories, step duration breakdown (ms), processed row counts, and health rates.

▣ Living Documentation

Export single-file HTML/Markdown documentation portals with embedded SVG lineages with one click.

Start Building Today In 30 Seconds

● ● ● bash • quickstart

```
$ pip install det-core
# Unpack the production NovaMart sample warehouse
$ det sample-data
# Launch the visual Web IDE in your browser
$ det gui
```

📄 Interactive Web Showcase

Try it right now without installing anything!

det-labs.github.io/det/demo.html

📄 GitHub Repository

Star the repo, check docs, and submit custom connectors.

github.com/det-Labs/det

📄 PyPI Release

Official package release v0.2.0 is live!

pypi.org/project/det-core/

📄 GitHub Wiki

Architecture deep-dives and tutorials.

github.com/det-Labs/det/wiki