dbt, explained with our OverDrive checkouts

how a pile of raw vendor data becomes something you can answer questions with

dbt — "data build tool": it runs your SQL transformations in the right order, tests them, and records where every number comes from.

concept zine · #1 · jun '26

WORDS, UNPACKED

- **dbt** — "data build tool"; runs your SQL transforms in order, tested + documented.
- **DuckDB** — a fast analytics database that runs inside a program (no server).
- **dbt-duckdb** — the connector that lets dbt run on DuckDB.
- **model** — one SELECT statement = one output table/view.
- **ref()** — how one model points to another; sets the build order.

scan it · rayvoelker.github.io/2026-06/issue_1_dbt-explained

dbt-duckdb: all on one machine

DuckDB is "SQLite for analytics" — a fast database that runs *inside* the program, no server. **dbt-duckdb** plugs dbt into it.

The whole enrichment pipeline runs in seconds, on one machine, against our own data — no warehouse, no cloud bill.

The big-company toolchain, library-sized — and the same recipe enriches Hoopla checkouts too.

Lineage: see the whole pipeline

One command — dbt docs generate — draws the dependency graph and the documentation.

You can trace *checkouts by audience rating* all the way back to the raw vendor file.

That traceability matters a lot when the data is about customers.

What dbt is

A data pipeline has three jobs: **Extract** (pull the data), **Load** (store it), and **Transform** (make it usable) — *ETL*. **dbt does the Transform** — turning raw, loaded data into something you can answer questions with.

dbt is a *build tool for SQL*. You write each transformation as one SELECT — called a **model** — and dbt turns it into a table, in the right order, tested and documented.

It has no database of its own. It's the **chef**, not the stove.

The vendor hands us a messy pile of JSON; one model turns it into a clean table:

```sql
-- model: checkouts  (one SELECT = one table)
select checkoutid as checkout_id,
       title, branch, format,
       try_strptime(checkoutDateUtc,
                    '%m/%d/%Y %H:%M:%S') as
           checked_out
from read_json('overdrive_*/page_*.json')
```

ref(): it figures out the order

Enrichment = combining. To get *checkouts by audience rating*, we join checkouts to a **title dimension**.

You point one model at another with **ref()**, and dbt works out what to build first — and draws the map.

```sql
from {{ ref('checkouts') }} c
join {{ ref('dim_title') }} d using (title_id)
```

raw → checkouts → dim_title ↘
daily_by_audience

Arranging data in layers like this — raw → refined → enriched — has a name: **medallion**. dbt moves data *between* the layers. *(More in its own issue.)*

Incremental: only touch what's new

OverDrive is pulled twice a day on an overlapping 2-day window, so the same checkout lands in ~4 files.

An **incremental** model keeps **59,770 unique** checkouts — and a re-run adds **0 rows**.

Meaningful counts *require* this dedup; otherwise every number is inflated.

Tests: the data checks itself

Before you trust an enriched number, dbt checks it. A **test** is a rule that runs on every build:

```yaml
columns:
  - name: checkout_id
    tests: [not_null, unique]
```

No two checkouts share an ID. If that's ever false, the build **stops** — a bad number never ships.