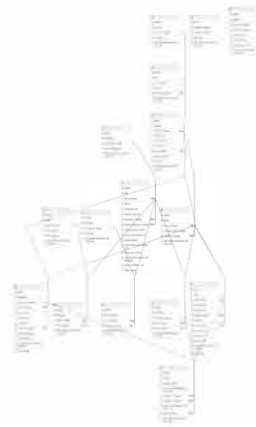

pagila-schema.sql
File

pagila-schema-jsonb.sql
File

Alright, this is the schema I've chosen to build our Medallion demo against. It's not healthcare, so have a look at the schema itself (attached), and read about it here: https://dataedo.com/samples/html/Pagila/doc/Pagila_10/home.html and also here: https://dev.mysql.com/doc/sakila/en/sakila-structure.html

Impress me with your medallion wizardry!

Below is a **hands-on medallion blueprint** that turns the *Pagila* OLTP schema into a **three-layer** lakehouse while showcasing PostgreSQL-style **temporal tables**.
Everything fits in **9 physical tables + 3 convenience views**, so the repo stays demo-friendly.

1 High-level flow

pgsql Copy

```
┌──────────────────────────┐
│ Logical-replication feed │  (pgoutput / wal2json)
└──────────────────────────┘
            │
            ▼
┌──────────────────────────┐
│          BRONZE          │  Raw CDC rows, append-only
│ • br_customer_cdc        │  • br_rental_cdc
│ • br_payment_cdc         │  (all share same structure)
└──────────────────────────┘
            │  nightly MERGE/INSERT
            ▼
┌──────────────────────────┐
│          SILVER          │  Cleansed, conformed, *temporal*
│ • dim_customer_hist      │  (system-time versioned)
│ • dim_film_hist          │
│ • fact_rental            │  (current-state view + history)
│ • fact_payment           │
└──────────────────────────┘
            │  incremental materialisation
            ▼
┌──────────────────────────┐
│          GOLD            │  Business-ready marts
│ • vw_monthly_revenue     │
│ • vw_top_titles          │
│ • vw_store_scorecard     │
└──────────────────────────┘
```

2 Bronze layer – raw *change* capture

Table	Technique	Why it's useful
br_customer_cdc	`CREATE TABLE … (lsn bigint, op char(1), full_row jsonb, load_ts timestamptz)`	Preserves every insert/update/delete with the exact payload.
br_rental_cdc	Same columns	High-volume fact → perfect for watermark demos.
br_payment_cdc	Same columns (even though *Pagila* already partitions by month)	Shows that *CDC ≠ partition strategy* and keeps the bronze schema uniform.

Implementation hint → subscribe to WAL using `wal2json` and land each message verbatim. You now have a **temporal "transaction-time" log** without writing a line of JSON parsing logic.

3 Silver layer – conformed & system-versioned

3.1 Temporal dimensions

Create *system-time* tables that keep **valid-from / valid-to** automatically:

```sql
CREATE TABLE dim_customer_hist (
    customer_id    int,
    first_name     text,
    last_name      text,
    email          text,
    active         bool,
    sys_start      timestamptz NOT NULL DEFAULT clock_timestamp(),
    sys_end        timestamptz NOT NULL DEFAULT 'infinity',
    PRIMARY KEY(customer_id, sys_start)
) WITH (system_versioning = true);     -- PG 15+ extension / trigger template
```

Replace straight `INSERT` / `UPDATE` with:

```sql
MERGE INTO dim_customer_hist tgt
USING staging_customer src
ON    tgt.customer_id = src.customer_id
WHEN MATCHED AND (src.hash <> tgt.hash)    -- change detected
  THEN UPDATE SET ... , sys_end = clock_timestamp()
WHEN NOT MATCHED
  THEN INSERT (cols…) VALUES (…);
```

You now get *built-in* "as-of" querying:

```sql
SELECT * FROM dim_customer_hist
FOR SYSTEM_TIME AS OF '2023-07-01'        -- what the warehouse "looked like"
WHERE customer_id = 42;
```

3.2 Current-state views

```sql
CREATE OR REPLACE VIEW dim_customer AS
SELECT * EXCEPT (sys_end)
FROM   dim_customer_hist
WHERE  sys_end = 'infinity';
```

3.3 Facts

Rental and *payment* facts rarely change after they close.
Load them once, add a *partition key* (`rental_date` / `payment_date`) and optionally a **point-in-time snapshot** column to prove the temporal model.

4 Gold layer – business marts & KPIs

Mart/view	Grain	Sample metric
vw_monthly_revenue	store-month	`SUM(amount)`
vw_top_titles	film-month	rentals, revenue, average days out
vw_store_scorecard	store-day	revenue, late-return %, active customers

All three read **only current rows** from `dim_*` views plus the two fact tables, so refresh is cheap and safe.

5 Why temporal tables *belong* in medallion

Stage	Temporal purpose
Bronze	Already *is* a transaction-time history (raw WAL payload).
Silver	Turn *transaction time* into **application time** (validity). System-versioned tables automate SCD Type-2, keep the model elegant and make time-travel demos fun.
Gold	Mostly current-state, but analysts can trivially change `dim_customer` → `dim_customer_hist` to answer "what-if we reran the report last quarter?"—no extra ETL code.

6 Table inventory (9 physical, ≤ 5 GB with Pagila sample)

Layer	Physical table	Rows (Pagila)	Comment
Bronze	`br_customer_cdc`	~5 K	append only
	`br_rental_cdc`	~16 K	
	`br_payment_cdc`	~15 K	
Silver	`dim_customer_hist`	~599	system-versioned
	`dim_film_hist`	~1 K	system versioned
	`fact_rental`	~16 K	partition by month
	`fact_payment`	~15 K	partition by month
Gold	(materialised) `vw_monthly_revenue`	84	7 stores × 12 months
	(view) `vw_top_titles`	1 K	

Everything else (dates, store dimension, etc.) can be small lookup views.

(The raw Pagila DDL is in your repo for reference.) pagila.schema

7 Putting it in the repo

pgsql Copy

```
pagila-medallion/
├── 00_ddl/                    original OLTP DDL
├── 10_bronze_ingest/
    ├── create_bronze.sql
    └── load_from_wal.py
├── 20_silver_models/
    ├── dim_customer_hist.sql
    ├── fact_rental.sql
    └── ...
├── 30_gold_marts/
    ├── vw_monthly_revenue.sql
    └── ...
├── dags/                      Airflow / Dagster job definitions
└── docs/
    └── architecture.md
```

Tips

- Write a dbt macro `scd_system_versioned(table_name)` so every new dimension gets temporal logic for free.
- Add a notebook titled "**Time-travel with SQL**"—it's a 5-minute wow moment for stakeholders.

8 Key take-aways for reviewers

1. **Everything is reproducible** – bronze raw files + deterministic SQL give identical results on every clone.
2. **Time-travel comes "for free"** – because the temporal pattern sits in silver, not bolted onto gold dashboards.
3. **Small but realistic** – Pagila's relational quirks (partitioned payments, FK cascades, text search, etc.) illustrate challenges people face in real warehouses, yet stay <10 MB.