GILBERT DATABRICKS USER GROUP · FRIDAY 18 SEPTEMBER 2026 · 6 PM

Building applications on Lakebase.

Postgres, Databricks Apps and AI in one place. Live build, then questions. Food provided.

PREETHAM REDDY · FOUNDER AND CEO, TECHFABRIC

ORGANISED BY techfabric databricks

EZ SPACES COWORKING, GILBERT

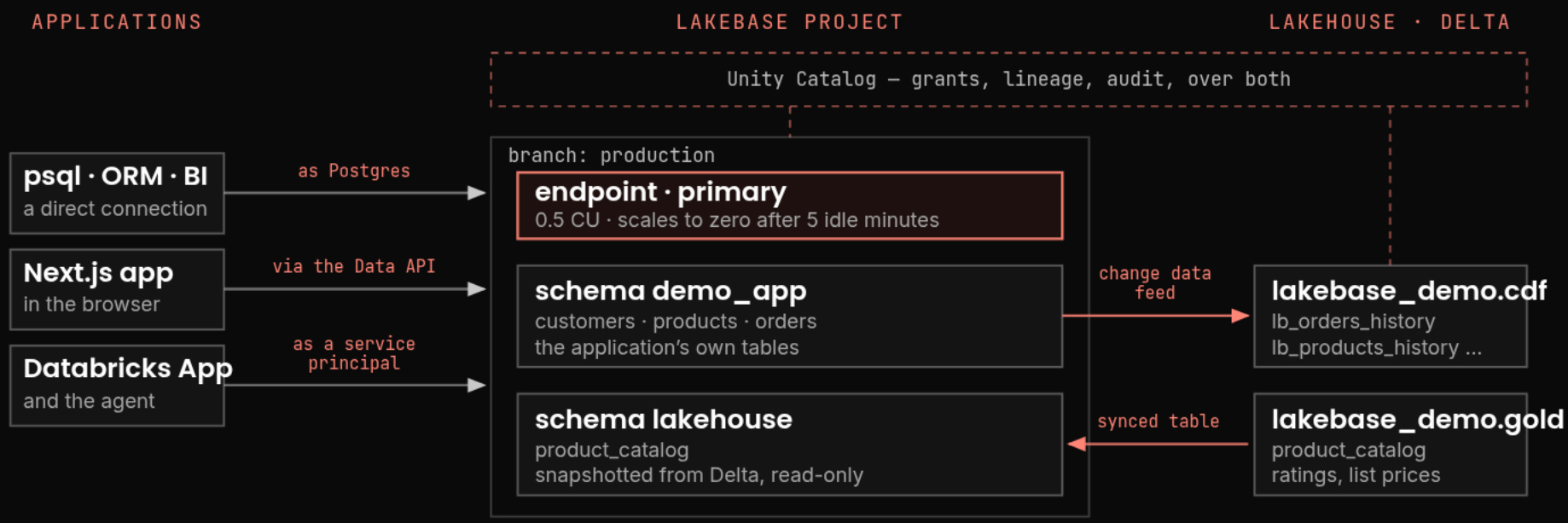
WHAT WE'LL BUILD

Four things, live.

01	PostgreSQL with Databricks identity	01_postgres_and_identity.sh
02	Safe branching and recovery	02_branching_and_pitr.sh
03	A secure Data API	06_data_api.sh
04	Lakehouse and application data together	03_lakehouse_integration.sh

THE SHAPE OF IT

One database, two neighbourhoods.



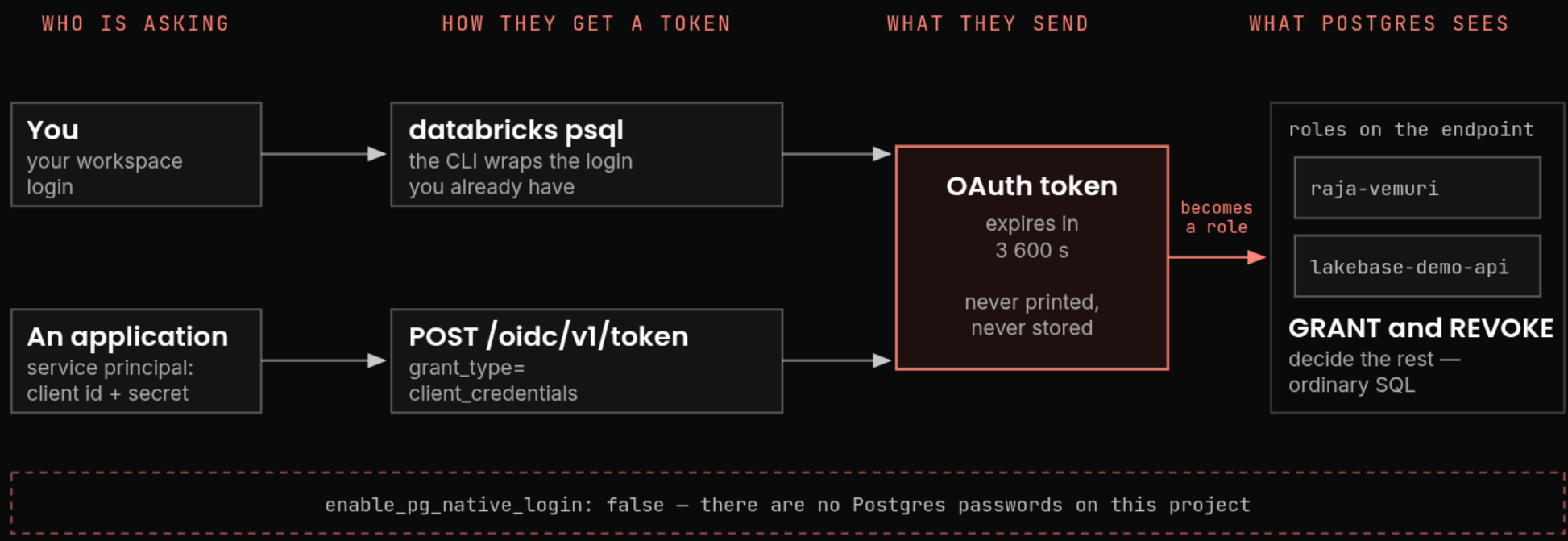
Lakebase is a managed Postgres inside a Databricks project. The application connects to it as Postgres, Unity Catalog governs it, curated Delta data snapshots in, and every application write streams back out.

01 / 04

PostgreSQL with Databricks identity.

Existing tools connect unchanged. Tokens replace passwords.

Where the password would have been.



Both paths mint the same thing: a one-hour OAuth token. Postgres sees a role, and ordinary `GRANT` decides the rest.

Nothing to rewrite.

PROTOCOL

It is ordinary PostgreSQL.

A `demo_app` schema of customers, products and orders, with foreign keys and check constraints. A join, an aggregate and `select version()` answer. Any client or ORM connects unmodified.

IDENTITY

No database passwords at all.

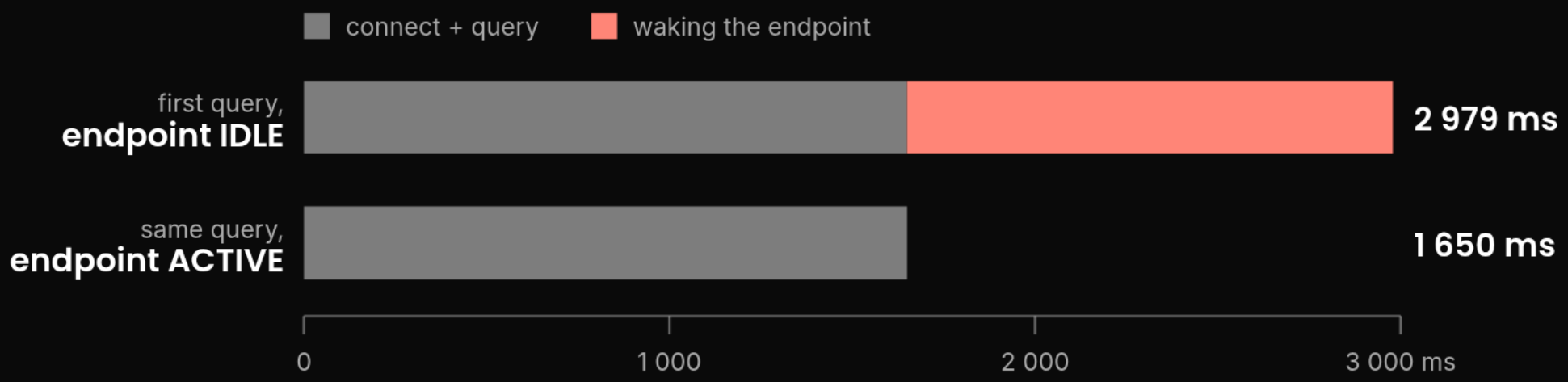
The project sets `enable_pg_native_login: false`. `databricks psql` wraps your workspace login; an application mints its own one-hour token. Nothing to rotate, nothing to leak.

GRANTS

Authorization is the SQL you know.

`postgres create-role` gives a service principal a Postgres role, then plain `GRANT`

What scale-to-zero costs you.



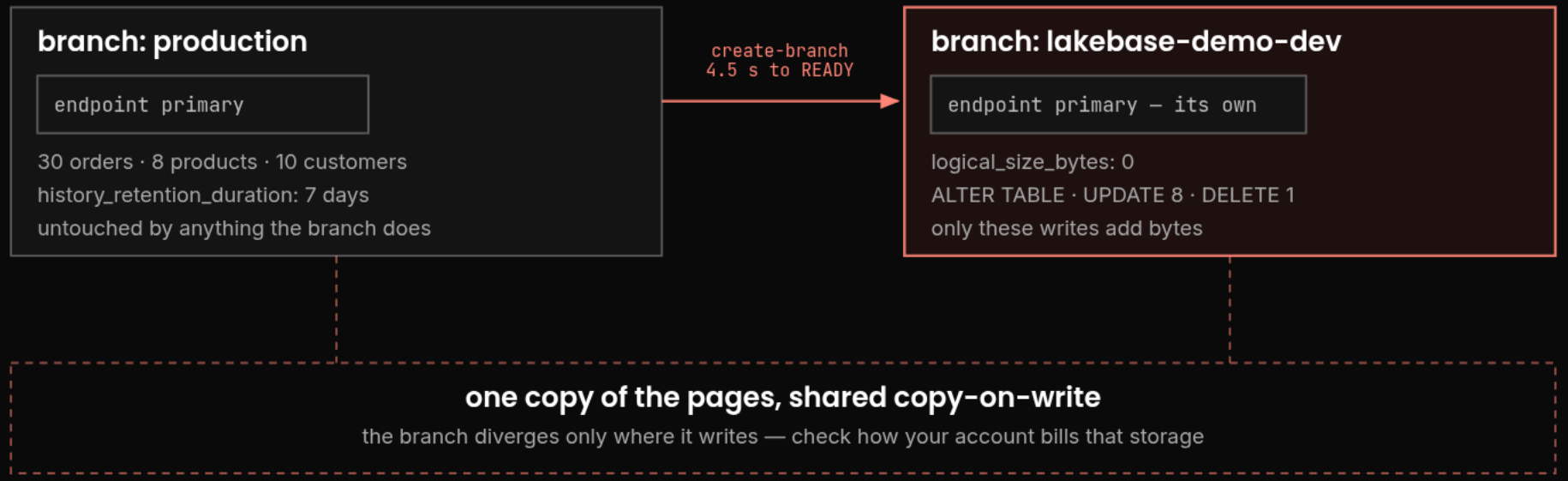
Measured on this project, not a datasheet: about 1.3 s of the cold query is the endpoint waking. `min_cu 0.5`, `max_cu 0.5`, `suspend_timeout 300s` — five idle minutes and it scales back to zero on its own.

02 / 04

Safe branching and recovery.

An isolated copy in seconds. A way back after a bad update.

A database in four seconds.



`create-branch` returns `READY` with its own read-write endpoint. `logical_size_bytes` starts at 0 because the pages are shared with the parent — check how your account bills branch storage before calling it free.

02 · ISOLATION

The same question, two answers.

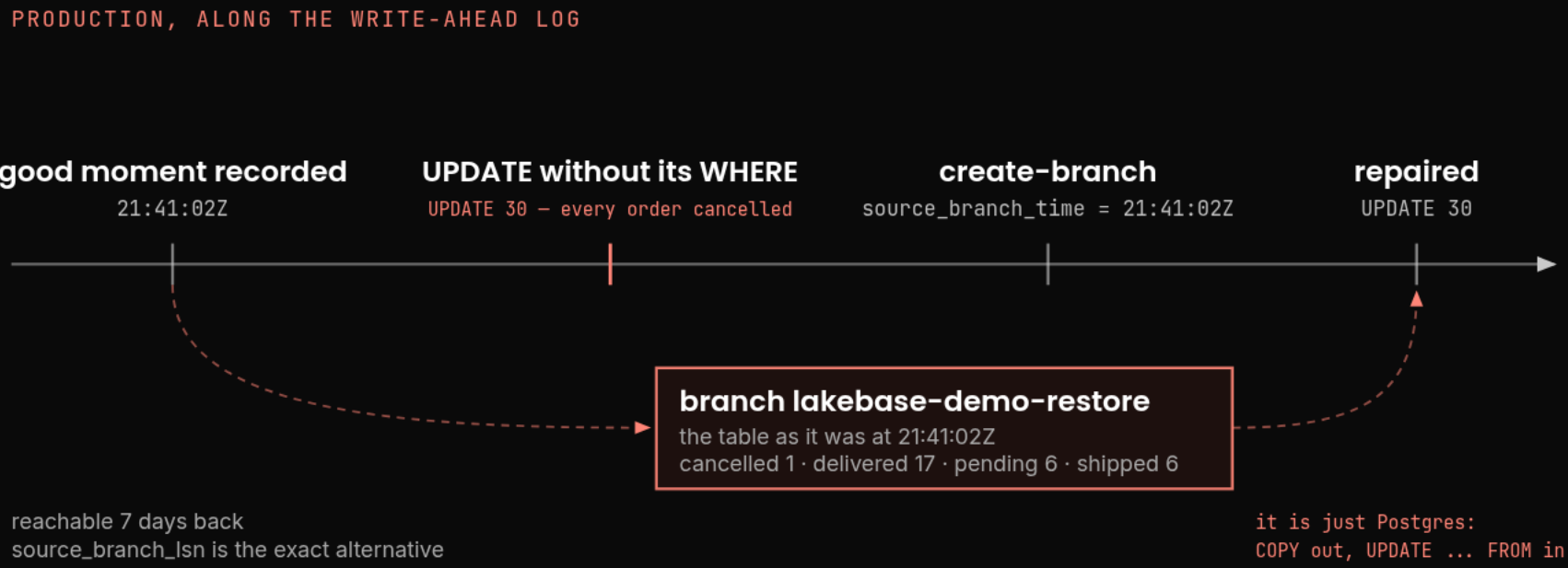
One fingerprint query, run on both branches after `ALTER TABLE`, `UPDATE 8` and `DELETE 1` on the branch only.

branch	orders	cancelled	order_columns	open_revenue_usd
lakebase-demo-dev	29	0	8	1573.00
production	30	1	7	1430.00

This is the “test the migration against real data” case: a schema change, a price change and a destructive `DELETE`, none of which production ever saw.

02 · POINT-IN-TIME RESTORE

The UPDATE that lost its WHERE.



Recovery is a branch at a timestamp. `source_branch_time` snaps to the last committed write at or before the moment you ask for; `source_branch_lsn` is the exact alternative.

02 · THE EDGES

What the script is honest about.

RETENTION

Seven days, on this project.

`history_retention_duration` is a per-project setting. Past it there is no moment left to branch from, so it is a number worth agreeing on before you need it.

SCOPE

The repair assumes nothing else is writing.

It copies every order's status back to the restore point. True for a demo project. Say it out loud before you run the same shape against a live one.

ALTERNATIVE

For a total loss, don't copy at all.

Point the application at the restore branch's endpoint, or promote the branch. That is a talking point here, not something the script does.

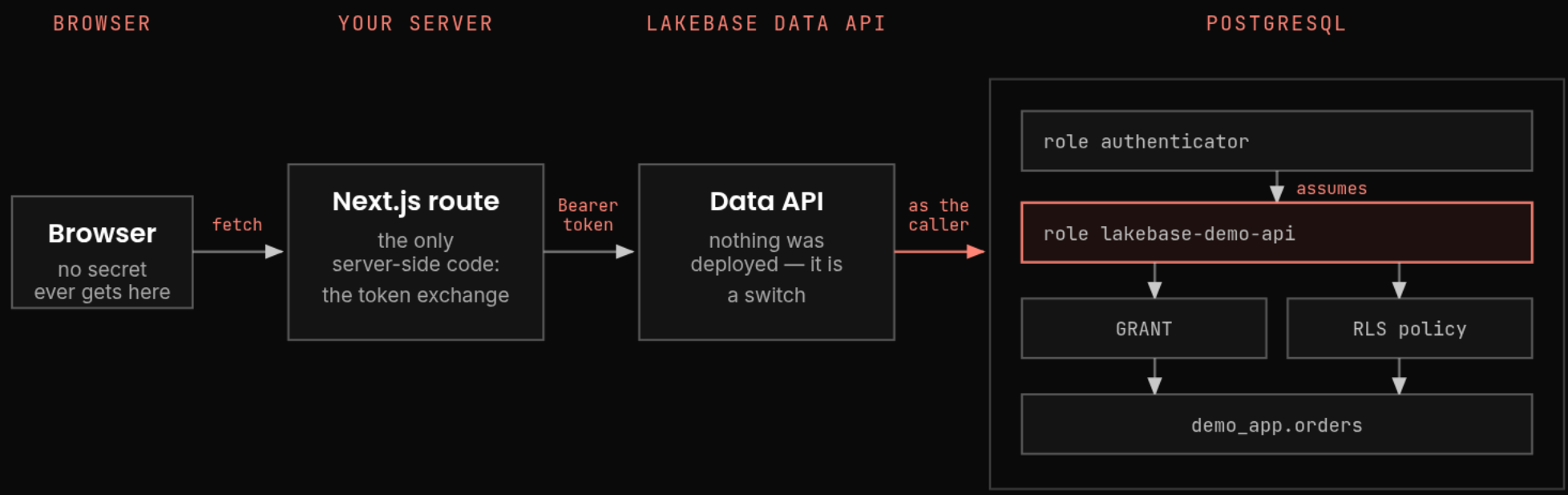
03 / 04

A secure Data API.

REST over Postgres, with no API service to build and host.

03 · THE REQUEST PATH

From the browser to the policy.



The only server-side code is the token exchange; the browser never sees the secret. Because the API runs each request as the caller's identity, `GRANT` and row-level security are the whole authorization model.

03 · READS

The URL is the query.

Filter and order are query parameters — products under $30, cheapest first

```
GET $API/demo_app/products?select=product_id,name,price_cents&price_cents=lt.3000&order=price_cents
200 · 536 ms · {"product_id":3,"name":"Notebook A5","price_cents":900} …
```

Foreign keys come back as embedded objects — no join written anywhere

```
GET $API/demo_app/orders?select=order_id,status,customers(name),products(name,price_cents)
200 · 542 ms · {"order_id":30,"status":"pending","customers":{"name":"Omar Haddad"},…}
```

Counts and pagination are headers

```
GET -H 'Prefer: count=exact' -H 'Range: 0-4' $API/demo_app/orders
206 · 424 ms · content-range: 0-4/30 · preference-applied: count=exact
```

Writes are the same URL — Postgres assigns the id, no `order_id` in the body

```
POST -d '{"customer_id":1,"product_id":2,"quantity":3}' $API/demo_app/orders
201 · 384 ms · {"order_id":31,"status":"pending",…}  →  PATCH status=shipped  →  DELETE · 204
```

03 · ROW-LEVEL SECURITY

One policy, three answers.

`CURRENT_USER <> 'lakebase-demo-api' OR status <> 'cancelled'` — one policy on `demo_app.orders`, tested against the role the token became.

who asks	how	what comes back
the owner, over psql	`select status, count(*)`	cancelled 1, delivered 17, pending 6, shipped 6 — owners bypass RLS
the service principal	GET orders? status=eq.cancelled	200 · [] — those rows do not exist for this identity
the service principal	POST a cancelled order	403 · 42501 new row violates row-level security policy

Identity flows from the OAuth token to the Postgres role to the policy with no application code in between — and the policy refuses to create what it would hide.

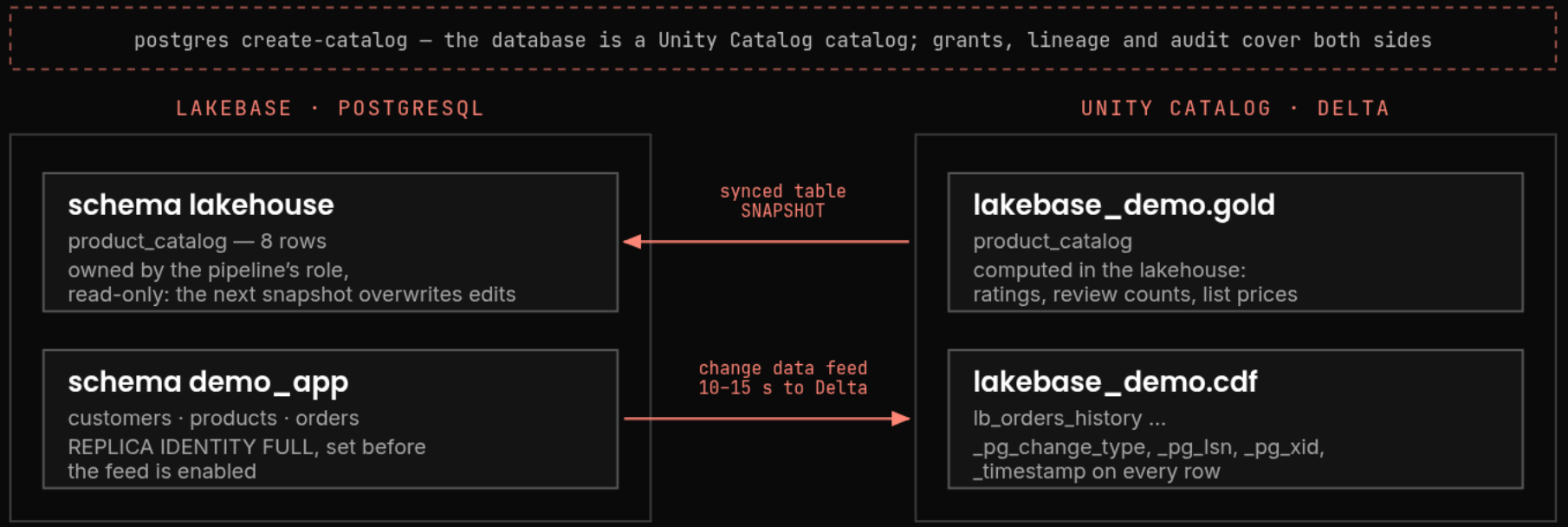
04 / 04

Lakehouse and application data together.

One governance model over Delta and Postgres, and traffic both ways.

04 · BOTH DIRECTIONS

In as a snapshot, out as history.



Two managed pipelines and no reverse-ETL code. The synced copy lives in its own `lakehouse` schema so the change feed on `demo_app` does not replicate it straight back out.

Registered, then synced.

CATALOG

The database becomes a catalog.

`postgres create-catalog`, 5.7 s. Its schemas and tables are securables like any other, and a SQL warehouse reads live Postgres rows in place — no copy.

SYNC

Delta lands in its own schema.

`create-synced-table` in `SNAPSHOT` mode: online 51 s after the call, through a managed serverless pipeline. The app then joins it in plain SQL.

REFRESH

Re-snapshot on demand.

Change the gold table, `pipelines start-update`, about 33 s. The table is owned by the pipeline's role — treat it as read-only, because the next snapshot overwrites local edits.

04 · THE WRITE PATH

Every write is a Delta record.

One order, three transactions, as they land in `lakebase_demo.cdf.lb_orders_history`.

_pg_change_type	order_id	status	quantity	_pg_xid
insert	9001	pending	1	9241
update_preimage	9001	pending	1	9242
update_postimage	9001	shipped	1	9242
delete	9001	shipped	1	9243

`REPLICA IDENTITY FULL` has to be set **before** the feed is enabled — a table without it is `SKIPPED`, and flipping it afterwards is not reliably picked up. Writes are queryable in Delta 10–15 s later.

OVER TO YOU

Questions.

Then war stories, and what you'd like the next session to be.

The Gilbert Databricks User Group meets at the TechFabric office, EZ Spaces Coworking, 1530 E Williams Field Rd, Gilbert.
meetup.com/phoenix-data-ai

ORGANISED BY techfabric databricks

THANKS FOR COMING