DATABRICKS USER GROUP · DNIPRO

From a six-line ticket to embedded analytics on Databricks

Field notes: what we built, what we measured, and what we threw away

Andrii Taran

Software Engineer, TechFabric

techfabric × databricks

THE JOURNEY

Eight steps, most of them rebuilt at least once

1
Lakehouse Federation
live data, zero copy

2
One dataset per dashboard
query count is the lever

3
Pages, not dashboards
one embed, many angles

4
Tenant isolation
one JOIN, fail-closed

5
Materialize all or nothing
Delta fact and Delta dims

6
Business hours
freshness as a schedule

7
Refresh button
the one Databricks lacks

8
Filters, cost, ownership
three traps from prod

THE ASK

Six lines in the ticket, a seventh in conversation

1. Customizable without deployments
2. Dynamic query and filter options
3. No impact on the application's performance
4. No sync issues
5. Good charts, trends, graphs
6. AI-enabled features

7

Real-time numbers

The row the application wrote a minute ago, on a chart, inside the client's portal. A nightly refresh would not do.

THE TENSION

"Real time" and "no impact" cannot both be true

REAL TIME

Every chart reads the live row. Every viewer is a query on the transactional database.

VS

NO IMPACT

Analytics reads a copy. The copy is as fresh as the last refresh, never fresher.

Our constraints: no data team, no budget for an ETL project, reports living in Power BI over SQL views. So: build the cheapest thing that shows real data, then measure where the line falls.

PLATFORM CHOICE

Why Databricks, not a BI tool on a read replica

AI / ML direction

The sixth line was not decoration. The client wants to grow there, and so do we. A BI stack now means a second stack later.

One governed place

Unity Catalog: the same tables and the same grants serve AI/BI dashboards today, Genie next, models after that.

Lakebase

A managed Postgres registered in the same Unity Catalog. Lakehouse tables sync into Postgres; Postgres changes land as Delta (preview).

End state: the application's own data lives next to its analytics, one set of permissions, no copy jobs. We are not there. Every step tonight stays on the platform for that reason.

1 LAKEHOUSE FEDERATION

Step one: read the live database in place

- No pipeline, no storage, no schedule: a demo in front of the client within days
- Serverless workspace; SQL reached through the Azure-services firewall rule. Private connectivity for serverless is an account-level NCC, a later step
- Credentials live in Key Vault; a setup script loads them into the UC connection
- Supported SQL Server connection options: host, port, user, password, trustServerCertificate, applicationIntent. "database" belongs to the foreign catalog

Pushdown is per table, not per join

−90%

rows scanned on Azure SQL
130,438 → 12,330

−22%

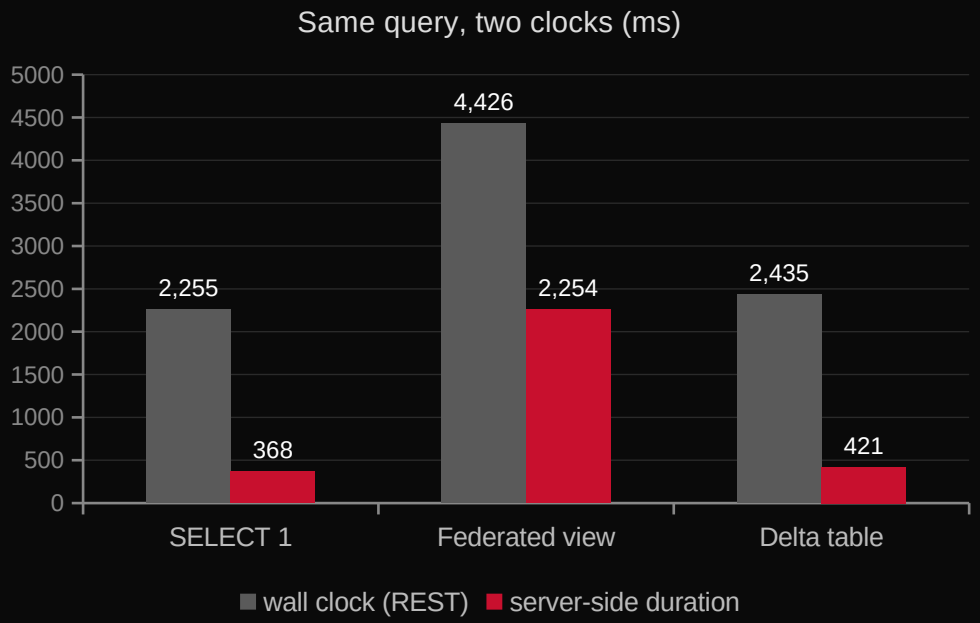
end-to-end query time
4,745 ms → 3,719 ms

```
-- EXPLAIN FORMATTED, after the fix
SELECT "Id" FROM "dbo"."Assignments"
WHERE ("HasReportA" = 1 OR "HasReportB" = 1)
  AND "Id" IS NOT NULL
```

- The view filtered one side of the join. The other side had no predicate of its own, so SQL returned every row (6.46M in prod) to produce about 800
- Federation pulls join keys first. Roughly a second of planning, connection and Spark overhead sits under every federated query, and no predicate moves it
- Benchmarking with COUNT(*) hid the whole effect: 4,140 → 3,853 ms, which is noise
- NULL trap: the flags are bit NULL, so NOT (a = 1 OR b = 1) returns nothing for the rows you hunt. Use COALESCE

1 MEASUREMENT

Measure server-side: wall clock lies by 1.5–2 s




```
GET /api/2.0/sql/history/queries
    ?max_results=200

/* TAG variant-C */ SELECT ...

query_source.dashboard_id
```

- Read "duration" from query history, not your stopwatch
- GET, not POST: POST answers ENDPOINT_NOT_FOUND
- Tag benchmark statements, match on the tag
- Attribute board traffic by dashboard_id, never by SQL text

Step two: one dataset per widget hurt the database

BEFORE

Widget → its own pre-aggregated dataset → its own query. A filter change re-runs all of them.

AFTER

One row-level dataset per board. Counters, bars, lines and pies aggregate in their encodings. Filters bind to the shared dataset.

285

queries from one dashboard in the window, 980 s total

114

queries per combined load reading 38 rows in total

~17 s

of that load is per-query compile overhead

47 s

slowest single query before (range 13–47 s)

Four Lakeview traps that cost us a day each

Counter value

Accepts only a simple aggregate of a column. Bake ×100 into a derived dataset column, not a formula.

`"--" comments in dataset SQL`

A combo widget inlines the SQL to one line. The comment eats the rest of the query.

Matrix pivots

A pivot forces a pre-aggregated dataset, which is one more query per load. Prefer viz that aggregates client-side.

Field filters

They hijack the filter widget's option list. We backed them out and bound filters to the shared dataset.

Step three: pages, not dashboards

5 → 1

reports folded into one dashboard

17

pages, shown as tabs

~20

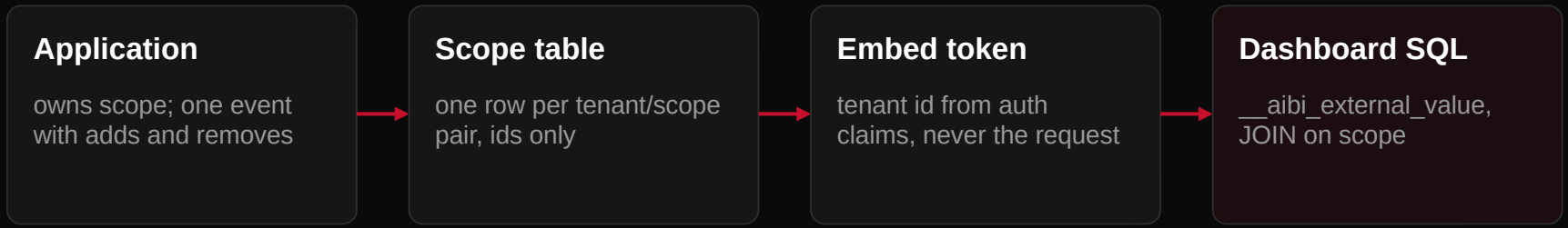
datasets, defined once, shared by every page

A "report" is a page.
A "dashboard" is a dataset family.

- Dashboard switch = new iframe + new embed token + shell reload + dataset re-run
- Page switch = client-side, no token, no re-run of a loaded dataset
- The benchmark board stayed separate: different fact table, it would drag its datasets into every load
- Filters are declared per page; keep the base population identical across pages or totals stop matching the export

4 TENANT ISOLATION

Step four: the requirement nobody wrote down



- Why not Unity Catalog row filters: they bind to a Databricks identity, and embedded viewers have none. One service principal renders every board
- Why not a portal-side parameter: parameters are visible and editable in the URL. Scope must not be settable by the viewer
- Outside an embed the value is empty, the JOIN returns 0 rows. Fail-closed by construction
- A colon (":__aibi_external_value") makes it a named parameter and the board fails to render

4 TENANT ISOLATION

One JOIN, three ways to get zero rows silently

```
FROM fact_view f
JOIN tenant_scope s
  ON s.TenantId =
     lower(split_part(__aibi_external_value, '~', 1))
 AND ( (s.ScopeType = 'Client'
        AND upper(s.ScopeId) = f.ClientId)
    OR (s.ScopeType = 'Partner'
        AND upper(s.ScopeId) = f.PartnerId) )
```

Scope-adaptive client picker: one UNION, only the tenant's branch returns rows. A single-select passes the shown value verbatim, so the picker value is the name and the base resolves name → id through the master JOIN.

Case

Scope ids are lowercase GUIDs, fact ids are upper. Spark "=" is case-sensitive. Normalize the small side only, keep the fact column bare and sargable.

EXISTS

A correlated EXISTS runs per row over federation. It was our first version and the latency killer. Use joins.

Bare read

The key is tenantId~scopeVersion. Compare it whole and every board renders empty.

4 INCIDENT

One character took every board down

```
urn:aibi:external_data:<external_value>:<external_viewer_id>:<dashboard_id>
```

`tenantId|scopeVersion`

400 "Dashboard ID is missing in token claim." on every load. The portal clipped the banner: users saw a blank white box.

`tenantId~scopeVersion`

200, queries run. "-" and "_" are out as well: both occur inside a GUID, so split_part would cut the tenant id.

- We checked the size limit at design time. We did not check the character set
- Build guard: fails on a reserved separator or a bare read of the value
- Deploy guard: a smoke test opens all 7 boards with a real token. Probed with the old separator it fails 7 of 7 with the exact 400

4 TENANT ISOLATION

Scope version: bust one tenant's cache only

Deterministic

`8d71515002df twice for the same tenant`

Distinct per tenant

`8d71515002df vs 990353059816`

Changes with the set

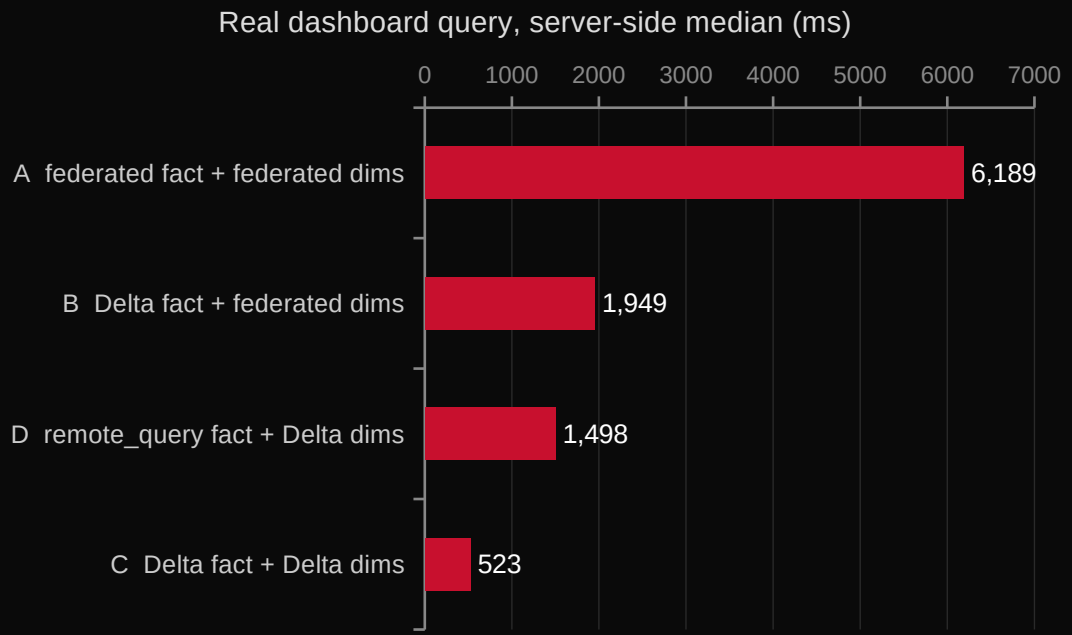
`full 8d71515002df → narrowed 9ff560cd66bf`

Order-independent

`sort_array: natural == reversed`

The scope key doubles as the cache key. A scope change costs one cold load for that tenant only. The 24-hour dashboard result cache keeps serving everybody else.

Step five: materialize all or nothing



11.8×

C against A: −92%

- The dimensions were the untapped half: the fact was Delta for weeks while every query still joined scope and lookups over federation
- Mixing a Delta fact with federated lookups breaks all-in-SQL pushdown
- remote_query is slower than Delta for serving. It belongs in the refresh job

5 MATERIALIZATION

What it bought: a full dashboard load, replayed

110–138 s

before: sum per load, 19 datasets, federated dims, live fact

20.5 s

after: Delta fact + Delta dims median query 1.0 s, max 2.6 s

5.4–6.7×

faster, and the 13–47 s outliers are gone

```
tables hit, last 1000 history entries
  scope table        196   federated
  lookup A           132   federated
  lookup B           132   federated
  serving snapshot   122   Delta
```

- The query history told us where the time went, not the query plan
- Result and disk cache are unsupported over foreign catalogs. Caching starts when data is in Delta
- Join pushdown preview measured net-negative for us (8.8 s → 24.8 s) and was turned off

5 DEAD ENDS

Things we tried so you do not have to

`ALTER CONNECTION ... SET OPTIONS`	Invalid. The form is OPTIONS (...), and it replaces the whole list: omit user/password and you strip the credentials
`"database" on the SQL Server connection`	Not a connection option. It belongs to the foreign catalog
`Three-part names via remote_query`	Azure SQL rejects them; USE is unavailable too
`TRIGGER ON UPDATE on foreign tables`	"unsupported table type". There is no dashboard-to-job hook
`REST schema listing of a foreign catalog`	Does not live-fetch metadata: returns empty with no error. Use SHOW TABLES on a warehouse
`Timing through the REST API`	A 2,255 ms "floor" that is really 368 ms

6 FRESHNESS

Step six: business hours as architecture

~92%
of human activity falls inside 07:00–19:00 ET, Mon–Fri

30 min
refresh cadence against 4–15 changed rows per hour

$43
per month for refresh, down from $57 with the buffer

$970
per month to keep the warehouse warm instead

- We bought query speed, not warmth. The dashboard warehouse still cold-starts on purpose
- A job carries exactly one schedule, and no single Quartz expression says "every 30 min on weekdays 7–18, hourly otherwise". So: separate night and weekend jobs, same task files
- Refresh jobs run max_concurrent_runs: 1 with the queue off. A late run is skipped, never stacked
- A partitioned refresh_status table keeps the MERGEs of different jobs disjoint

7 REFRESH

Step seven: the Refresh button Databricks lacks

```
POST /api/2.1/jobs/run-now   (second click)

state = TERMINATED
termination_details.code =
   MAX_CONCURRENT_RUNS_EXCEEDED
```

A naive button polls that run, sees TERMINATED, and reports success having refreshed nothing.

- The built-in dashboard Refresh re-runs dataset queries and cannot write. A CTAS dataset is accepted at author time and never executed
- Always look for an in-flight run first and join it. Two users clicking at once share one run and get fresher data at no extra compute
- active_only=true returns the doomed run too. Take the run with the earliest start_time
- Do not turn the queue on to hide this: that trades a visible skip for an invisible backlog
- "Last updated" is shown per board, relative, in the viewer's time zone

8 EMBEDDING

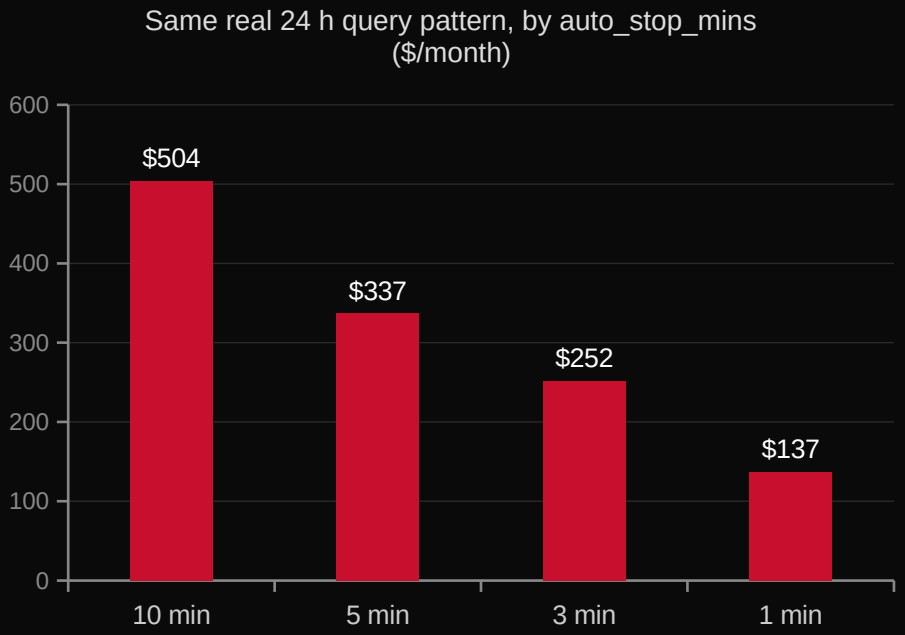
Step eight: filtering past NOT_SUPPORTED

```
<embed url> ? f_<page>~<widget>=<value>
```

`getFilters via postMessage`	NOT_SUPPORTED: embedded filter controls are not enabled for this dashboard
`URL + f_main~filter_3=<real client>`	Filter chip shown and KPIs computed: 137 / 677.2K / 9.36M
`URL + f_main~filter_3=<nonexistent>`	KPIs empty. The board re-ran its queries: a real filter, not a cosmetic echo

The hint was in the SDK typings all along: onFiltersChanged fires on "a URL restore", and FilterState.id is called the URL identifier. The host owns the iframe, so the host owns the URL.

Serverless cost: the idle tail is the whole story



89%

of billed time was idle tail 0.66 h executed, 6.00 h billed

1.6%

of prod board queries waited for a cold start

- Prod, first days, 3,509 queries: cold median 12.6 s, warm 1.0 s
- 83% of bytes read came from cache
- Nothing polls on a timer; warm-up is demand-driven and rate-limited
- Filtering costs nothing and buys little: the work is scan plus cache, not result size

The ownership transfer that caused an outage

```
Insufficient privileges: Table '...' does not
have sufficient privilege to execute because
the owner of one of the underlying resources
failed an authorization check.
```

```
-- before you transfer ownership
SELECT table_schema, table_name, table_owner
FROM   <catalog>.information_schema.tables
WHERE  table_owner = '<outgoing owner>';
```

- A UC view resolves against its owner's privileges. "The dashboard works, so I have access" is false
- An owner holds privileges implicitly and appears on no grant list. Transfer the object and that access disappears
- USE CATALOG alone is not enough: the next error names the schema and reads as if the catalog grant failed
- USE CONNECTION is independent again
- Prod is owned by a group, never by one person

DELIVERY

Shipping dashboards like code

One artifact, every environment

The same .lvdash.json deploys to each target through a dataset_catalog variable

Guards in the pipeline

A build assertion on the scope key, a smoke test that opens every board with a real token

Jobs run as the deploy principal

Not as whoever deployed last. Ownership cannot drift to a person

Fresh environments lie

Event-driven projections create their tables on the first event, not on deploy. Check SHOW TABLES before the first job run

SCORECARD

The ticket, answered honestly

Customizable without deployments	Boards edited in the workspace, semantic layer is UC views	*No app release; boards still ship through bundles*
Dynamic filters	Filters on one shared dataset, URL channel from the host	*A filter must not spawn a dataset*
No impact on the application	Delta snapshots on a separate warehouse; OLTP sees one bounded job	*Federation did hit OLTP at first*
No sync issues	One semantic layer, refreshed_at shown, race-safe Refresh	*Known freshness, not real time everywhere*
Charts and trends	In-widget aggregation, 17 pages on one board	*Some viz choices follow query count*
AI features	Genie space on the same governed views	*Tenant isolation per surface still to do*

WHAT IS NEXT

Still scaffolding: the steps ahead of us

1	**Dataset consolidation**	Per-tile datasets remain on some boards: 1.4× median slowdown when filtering the full range, up to 3.0×
2	**Private connectivity**	NCC for serverless instead of the Azure-services firewall rule
3	**Genie for clients**	Natural-language Q&A needs its own tenant isolation story
4	**Lakebase**	Move application data next to its analytics, one set of permissions
5	**ML on the same tables**	The reason we picked the platform in the first place

TAKEAWAYS

Six rules we would give ourselves on day one

1 Start naive, on the end-state platform

Federation bought a demo in days. Swap layers under a stable semantic layer later.

2 Measure server-side

Query history "duration", tagged statements, dashboard_id. Never the stopwatch.

3 Count queries before tuning them

One dataset per board, pages instead of dashboards.

4 Materialize all or nothing

A Delta fact with federated lookups is worse than either.

5 Fail closed, then guard it

Isolation in the query; a build assertion and a smoke test, because empty boards raise no error.

6 Pay for queries, not for warmth

The idle tail is the bill. Freshness is a schedule you can measure.

QUESTIONS

Thank you

The written series, part 1:

techfabric.com/blog/embedded-analytics-on-databricks-part-1

Andrii Taran

linkedin.com/in/i-m-andrii-taran

techfabric × databricks