# **Fintech Lakehouse Demo — Q&A Cheat Sheet**

### **1) What data did you use?**

**A:** I used a **synthetic fintech dataset** I generated for the demo so there’s no real PII. It mirrors common entities—**customers, accounts, devices, merchants, transactions, credit applications, and payments**—with realistic distributions (amounts, timestamps, MCCs, bureau scores, delinquencies). I staged the CSVs in **S3** at s3://payer-demo-ur/fin/….

### **2) Where did the data come from and why synthetic?**

**A:** It’s **internally generated** to avoid privacy/compliance issues and still demonstrate realistic patterns for **fraud**, **credit risk**, and **segmentation**. Using synthetic data keeps the demo shareable and repeatable.

### **3) What’s the schema at a glance?**

**A:**

* **transactions**(transaction\_id, account\_id, customer\_id, device\_id, merchant\_id, ts, amount, currency, channel, city, country, mcc, is\_fraud)
* **applications**(application\_id, customer\_id, annual\_income, bureau\_score, delinq\_12m, inquiries\_6m, approved, default\_12m)
* **customers**(customer\_id, age, country, created\_date, kyc\_status)
* **devices**(device\_id, is\_emulator, ip\_address, …)
* **merchants**(merchant\_id, mcc, …)
* **payments**(payment\_id, account\_id, amount, ts, …)

### **4) What architecture did you implement?**

**A:** A **Lakehouse** on Databricks using **Unity Catalog**:

* **Landing**: CSVs in **Amazon S3**
* **Bronze**: read CSVs → **Delta** tables (fin.raw.\*\_bronze)
* **Silver**: clean/cast → **Delta** (fin.curated.transactions\_silver, apps\_silver, customers\_silver)
* **Gold**: three outcomes  
  + Fraud: txn\_features\_gold, fraud\_scored\_gold, view v\_fraud\_alerts
  + Credit risk: credit\_risk\_scores\_gold, view v\_credit\_risk\_triage
  + Segmentation: customer\_rfm\_gold, view v\_segments
* **BI**: **Lakeview** dashboard (Fraud bubble, Credit heatmap, Segments bar)
* **Governance**: UC **masks**, **tags**, **lineage**
* **Orchestration**: **Workflows Jobs** (Bronze → Silver → Gold → Governance)
* **Interoperability**: **Parquet export** to S3 + external tables

### **5) Why Databricks over a classic warehouse here?**

**A:** One platform for **ETL + ML + BI** on **open Delta** with **governance** built-in. We avoid copying data across systems, keep **time travel**, **ACID**, streaming-readiness, and can still export **open Parquet** if another engine needs it.

Time travel examples

SELECT \* FROM sales\_data VERSION AS OF 5;

SELECT \* FROM sales\_data TIMESTAMP AS OF '2025-10-10T00:00:00';

### **6) How did you connect S3 securely?**

**A:** With **Unity Catalog External Location** backed by an **AWS IAM role** (AssumeRole + External ID). No access keys in code. GRANT READ/WRITE FILES controls access; all actions are audited.

### **7) Did you use Parquet or Delta? Why both?**

**A:** **Delta** for managed analytics tables (ACID, time travel, performance). **Parquet** export to prove **open formats/no lock-in**: I wrote Gold tables to s3://payer-demo-ur/fin/parquet/... and registered UC **external tables** on top.

### **8) What exactly are Bronze, Silver, Gold?**

**A:**

* **Bronze** = raw ingested (CSV → Delta), minimal changes.
* **Silver** = cleaned/typed/filtering for analytics/ML.
* **Gold** = business-ready outputs (fraud scores, risk scores, segments) optimized for BI.

### **9) How did you score fraud in real time?**

**A:** I computed near-real-time **velocity features** in SQL (1h/24h sum & count per account), joined **device risk** (emulator), **merchant MCC** signals, and **geo mismatch**. A simple **rules score (0–100)** flags alerts. In production, the same logic becomes **streaming** with **Auto Loader + Delta Live Tables (DLT)**.

### **10) How did you compute credit risk?**

**A:** A transparent **PD% proxy** over **bureau\_score**, **delinq\_12m**, **inquiries\_6m**, and **income**. For the demo I used a simple logit-style formula to keep it **Serverless** and explainable. In production, I’d train with **AutoML/Mosaic AI** and serve via **Model Serving**.

### **11) What’s customer segmentation?**

**A:** **RFM** over 180 days: **Recency**, **Frequency**, **Monetary** (spend). I bucket to label **CHAMPIONS / LOYAL / AT\_RISK / REGULAR** and expose counts + average spend for marketing/retention.

### **12) What did the dashboard show?**

**A:**

* **Fraud Alerts**: **Scatter** (X=time, Y=fraud\_score, bubble=size=amount, color=country/MCC) to triage spikes and high-value risks.
* **Credit Risk Heatmap**: average PD by **bureau band × delinquencies** (darker = riskier).
* **Segments**: bar chart of **customers per segment** (optionally avg spend).

### **13) What governance did you implement?**

**A:** UC **column masking** UDF for sensitive fields (e.g., ip\_address, customer\_id), **tags** on critical tables (DOMAIN, SENSITIVITY, MODEL), and I demonstrated **lineage** from S3 → Bronze → Silver → Gold → dashboards.

### **14) How do you refresh the pipeline?**

**A:** **Workflows Job** with four **SQL tasks** (Serverless):

1. Bronze (read\_files from S3)
2. Silver (casting/filters)
3. Gold (fraud feature/credit score/segments)
4. Governance (masks/tags)  
    It’s idempotent (CREATE OR REPLACE), and can be **scheduled** hourly/daily.

### **15) Why Serverless (and not a Spark cluster)?**

**A:** The AWS account had a **vCPU quota** limit, and **Serverless SQL** avoids EC2 capacity. It also **auto-scales** and keeps cost simple. If needed later, we swap Fraud features to **DLT streaming** with minimal change.

### **16) Show me time travel.**

**A:** I ran DESCRIBE HISTORY fin.curated.transactions\_silver and queried VERSION AS OF 0. That proves **Delta** versioning without copy-outs.

### **17) How do you prove interoperability?**

**A:** I wrote Gold tables to **plain Parquet** on S3 using INSERT OVERWRITE DIRECTORY … USING PARQUET and registered **external tables**. Any engine (Athena/Presto/Snowflake) can read them.

### **18) How do you ensure performance?**

**A:** Use **Delta** + **Photon** (Serverless). For heavy filters, OPTIMIZE … ZORDER BY (account\_id, ts) improves pruning. For BI, Lakeview queries hit Serverless caches.

### **19) How would you productionize fraud to real-time?**

**A:** Replace the batch feature step with **Auto Loader** against S3 (or Kafka), build a **DLT** pipeline for features + scoring, and serve a model via **Model Serving** or **Feature Serving**; keep tables the same so dashboards don’t change.

### **20) What security controls did you use?**

**A:** **IAM role** with External ID, UC **External Location** for S3 access, **GRANT**-based READ/WRITE FILES, column **masking**, **tags**, and **lineage**. No access keys in code.

### **21) What challenges did you hit and how did you resolve?**

**A:**

* **EC2 vCPU quota** → used **Serverless SQL** end-to-end.
* Objects created in workspace.default initially → migrated to **fin.\*** and set default catalog/schema.
* Parquet naming → explained **folder + part files** pattern (cloud-native), used external tables for querying.

### **22) What would you do next?**

**A:** Add **streaming** for fraud, train a proper **credit model** (AutoML), add **feature store**, and enforce **row/column-level policies** with UC for least privilege.

### **23) How do you test/monitor this?**

**A:** Add **data quality** checks (nulls, ranges) in Silver, set **Jobs alerts** on failures, and create a **Lakeview ops tile** (row counts, late data, error rates).

### **24) Can you walk through the object names we’ll see?**

**A:**

* **Catalog/Schemas:** fin.raw, fin.curated, (optional) fin.parquet\_export
* **Bronze:** \*\_bronze
* **Silver:** transactions\_silver, apps\_silver, customers\_silver
* **Gold:** txn\_features\_gold, fraud\_scored\_gold, credit\_risk\_scores\_gold, customer\_rfm\_gold
* **Views:** v\_fraud\_alerts, v\_credit\_risk\_triage, v\_segments
* **Parquet external tables:** fin.parquet\_export.\*\_parquet

### **25) If asked “how big is it?”**

**A:** It’s a **demo-scale** synthetic set sized to show realistic behavior and complete fast on **Serverless**. Row counts are visible in the dashboard and SELECT COUNT(\*) sanity checks.

## **One-minute demo script (TELL → SHOW → TELL)**

1. **S3 → UC**: “Data lands once in S3 → Delta under **Unity Catalog**.”
2. **Delta proof**: open transactions\_silver → **History** and **VERSION AS OF** query.
3. **Dashboard**:  
   * Fraud bubbles (time × score, size=amount)
   * Credit heatmap (bureau band × delinq → avg PD)
   * Segments bar (customers by segment)
4. **Governance**: open **Lineage**, show **masked** columns + **tags**.
5. **Jobs**: show **Bronze→Silver→Gold→Governance** scheduled on **Serverless**.
6. **Open formats**: show **Parquet** folder in S3 + query external table.
7. **Close**: “Same tables power ETL + ML + BI. Streaming-ready. Open formats. Governed.”

## **Micro-answers for rapid-fire**

* **Why Lakehouse?** ETL+ML+BI on one copy of data (Delta), governed by UC.
* **Real-time?** Swap batch fraud features to DLT streaming; dashboards don’t change.
* **No lock-in?** Delta for managed analytics, Parquet export for interop.
* **Security?** IAM role + UC permissions, masks, tags, lineage.
* **Cost?** Serverless, no clusters to babysit, auto-scale, schedule Jobs.
* **Performance?** Delta + Photon, Z-ORDER, serverless caching.
* **Testing?** Row counts, simple DQ in Silver, Job alerts.

If you want, I can condense this into a single-page PDF “talk track” you can print or keep beside your demo.  
  
  
**Lakehouse stack mapping (what you used, where)**

**S3 (your data lake)**

* Bucket: s3://payer-demo-ur/
* Raw CSV landing: s3://payer-demo-ur/fin/…
* Open-format export (proof of no lock-in): s3://payer-demo-ur/fin/parquet/{transactions|fraud\_scored|credit\_risk|segments}/
* External Location in UC: db\_s3\_external\_databricks-s3-ingest-e68bd (secure access via IAM role)

**Open table formats (bottom row of slide)**

* **Delta Lake** (used): all Bronze/Silver/Gold tables are Delta.
* **Parquet** (used): exported Gold/Silver to Parquet folders + registered external tables.
* **Iceberg** (not needed in this demo).

**Unity Catalog (the white bar)**

* Catalog: fin
* Schemas: fin.raw (Bronze), fin.curated (Silver/Gold), fin.parquet\_export (Parquet external tables)
* Governance:  
  + Column masks: fin.raw.devices\_bronze.ip\_address, fin.raw.customers\_bronze.customer\_id
  + Tags: transactions\_silver (DOMAIN=PAYMENTS, SENSITIVITY=PII), credit\_risk\_scores\_gold (DOMAIN=LENDING, MODEL=demo)
  + Lineage: visible from fraud\_scored\_gold back to Bronze tables and S3
* External Location permissions: READ/WRITE/CREATE EXTERNAL TABLE granted to your users

**Lakehouse (the red “lakehouse” layer)**

* **Bronze (raw)**: \*\_bronze tables created directly from CSVs via read\_files() (Delta)
* **Silver (cleaned)**: transactions\_silver, apps\_silver, customers\_silver
* **Gold (business-ready)**:  
  + Fraud: txn\_features\_gold, fraud\_scored\_gold, view v\_fraud\_alerts
  + Credit risk: credit\_risk\_scores\_gold, view v\_credit\_risk\_triage
  + Segmentation: customer\_rfm\_gold, view v\_segments

**Workflows / DLT (middle box on slide)**

* **Used Workflows Jobs (serverless)** to orchestrate:  
   01\_bronze → 02\_silver → 03\_gold → 04\_governance (idempotent SQL tasks)  
   *(DLT/streaming is your “next step” story; not required today.)*

**Databricks SQL (middle box)**

* **Serverless SQL Warehouse** for all ELT and queries (no clusters/vCPU quota issues)
* Delta time-travel proof: DESCRIBE HISTORY + VERSION AS OF on transactions\_silver

**AI/BI (right box)**

* **Lakeview dashboard**:  
  + Fraud tile (scatter “bubble” or bar, depending on what you kept): time vs fraud\_score; bubble size = amount; color = country/MCC (or aggregate bars by country/score band)
  + Credit Risk Heatmap: bureau\_band × delinq\_12m → avg\_pd
  + Segments bar: customers by RFM segment (+ optional avg spend)

**Mosaic AI (left box)**

* **Not used directly** in this run. Your talking point: “We’d plug in Mosaic AI for real-time model serving, Vector Search, or an analyst ‘why-flagged’ assistant—same governed data.”

# **20-second talk track (say this while you click)**

“Data lands once in **S3**. With **Unity Catalog**, we load to **Delta** Bronze→Silver→Gold in the **fin** catalog. **Workflows (serverless SQL)** refresh the pipeline. Governance is built-in—**masks, tags, and lineage**. **Lakeview** gives fraud triage, credit risk, and customer segments. For interoperability, we export the same results to **open Parquet**. This matches the slide: SQL warehousing, workflows, and BI on one **Lakehouse**, governed by UC—and ready to attach **Mosaic AI** when needed.”

# **Quick list of object names to show**

* **Catalog/Schemas:** fin.raw, fin.curated, fin.parquet\_export
* **Bronze:** customers\_bronze, accounts\_bronze, devices\_bronze, merchants\_bronze, transactions\_bronze, apps\_bronze, payments\_bronze
* **Silver:** transactions\_silver, apps\_silver, customers\_silver
* **Gold:** txn\_features\_gold, fraud\_scored\_gold, credit\_risk\_scores\_gold, customer\_rfm\_gold
* **Views:** v\_fraud\_alerts, v\_credit\_risk\_triage, v\_segments
* **Parquet external tables:** fin.parquet\_export.transactions\_parquet, …fraud\_scored\_parquet, …credit\_risk\_parquet, …segments\_parquet
* **Job:** Fintech-Lakehouse-ELT (four SQL tasks)
* **External Location:** db\_s3\_external\_databricks-s3-ingest-e68bd

If you want, I can turn this into a one-slide “legend” to place right after that platform slide so you can point at each layer and say what you built.  
  
  
  
**20-second close (say this)**

“On the Databricks Lakehouse we landed data once in S3 and ran **ETL, analytics, and ML** on the **same Delta tables**, fully governed by **Unity Catalog**. We delivered three fintech outcomes—**real-time fraud triage, credit risk scoring, and customer segmentation**—orchestrated with **serverless Jobs** and visualized in **Lakeview**. The platform gives us **speed, lower cost, stronger governance, and no lock-in** (we even exported **open Parquet**), so the business gets faster decisions with less risk and simpler ops.”

# **What the business gets (benefits)**

* **Faster time-to-insight:** single platform; serverless SQL + Jobs → minutes to refresh, no cluster wrangling.
* **Lower cost & simpler ops:** one copy of data, auto-scaling serverless, no data hops across tools.
* **Better risk control:** fraud features & alerts in near-real-time; transparent credit PD; auditable lineage.
* **Governed by design:** Unity Catalog masks, tags, permissions, and end-to-end lineage.
* **Open & future-proof:** Delta for ACID/perf; **Parquet** export for interop (Athena/Presto/etc.); easy path to streaming/DLT and Mosaic AI.

# **What we proved (with artifacts you can click)**

* **Lakehouse architecture works:** Bronze → Silver → Gold on Delta in fin catalog.
* **Fraud detection:** velocity features + rules → fraud\_scored\_gold, **Fraud** tile for triage.
* **Credit risk:** credit\_risk\_scores\_gold + **Heatmap** shows risk rising with delinq / falling scores.
* **Customer value:** customer\_rfm\_gold + **Segments** tile for marketing actions.
* **Governance:** UC **masks**, **tags**, and **Lineage** across tables, jobs, and dashboards.
* **Orchestration:** **Fintech-Lakehouse-ELT** job (serverless) schedules Bronze→Silver→Gold→Governance.
* **Openness:** INSERT OVERWRITE DIRECTORY … USING PARQUET to S3 + **external tables** for read-anywhere.

# **One-liner if they ask “why change the current state?”**

“Because the Lakehouse collapses ETL, ML, and BI onto one governed platform—**fewer moving parts, faster delivery, lower cost**, and it stays **open** so we’re not locked in.”

**What “upstream” vs “downstream” means**

* **Upstream** = everything your selected object **reads from** (its inputs).  
   Think: source tables/files, intermediate tables/views, jobs/queries that **write into** it.
* **Downstream** = everything that **reads from it** (its consumers).  
   Think: other tables/views built from it, dashboards, queries, and jobs that **depend on** it.

So if you’re looking at (say) fin.curated.fraud\_scored\_gold and you see **8 upstream / 14 downstream**:

* **8 upstream** might include things like:  
   transactions\_silver, txn\_features\_gold, devices\_bronze, merchants\_bronze, customers\_bronze, plus the **SQL task** in your Job run(s) that created it, and any intermediate temp objects captured by lineage.
* **14 downstream** might include:  
   the view v\_fraud\_alerts, your **Lakeview fraud tile** (and its dataset), ad-hoc queries you ran, and any other tables/views built on top of fraud\_scored\_gold.

# **Tips to interpret & show it**

* Click a node → you’ll see **who/what/when** (user, job name, SQL statement).
* Toggle **Column lineage** to highlight exactly which **columns** flow into, say, fraud\_score.
* Use the **time filter** (e.g., “Last 24h”) so the graph matches your recent Job run.
* Switch the selected node (e.g., click transactions\_silver) and the **upstream/downstream counts will change** relative to that node—handy to do live to show impact analysis.

# **Talk track (1 line)**

“**Upstream** shows where this table came from; **downstream** shows everywhere it’s used—so we can do instant impact analysis, auditing, and debugging.”

**Who did we tackle?**

## **Data Engineer**

* **Owns:** Ingest + ETL + orchestration.
* **What you built:** Bronze→Silver→Gold SQL pipelines, Fintech-Lakehouse-ELT Job, external location to S3.
* **Artifacts to show:** fin.raw/\*\_bronze, fin.curated/\*, Job graph (01\_bronze→04\_governance), DESCRIBE HISTORY, Z-ORDER/OPTIMIZE (optional).
* **Outcome:** Reliable, repeatable refresh; streaming-ready design (swap to DLT later).

## **Data Analyst / BI**

* **Owns:** Insights & dashboards.
* **What you built:** Lakeview dashboard (Fraud triage, Credit Risk heatmap, Segments bar), parameterized queries.
* **Artifacts to show:** Dashboard tiles, serverless SQL warehouse, saved datasets.
* **Outcome:** Self-serve analytics on the same Delta tables; near-real-time triage.

## **Data Scientist / ML**

* **Owns:** Features, scoring logic, model path.
* **What you built:** Velocity features table txn\_features\_gold, rules-based fraud score, PD proxy for credit.
* **Artifacts to show:** Feature SQL, fraud\_scored\_gold, credit\_risk\_scores\_gold.
* **Outcome:** Clear path to production ML (swap formulas for AutoML/Mosaic AI + Model Serving).

## **Security / Governance (UC Admin / Data Steward)**

* **Owns:** Access, lineage, compliance.
* **What you built:** Unity Catalog catalog fin, masks (PII), table TAGS, External Location perms, lineage demo.
* **Artifacts to show:** Column masks on ip\_address & customer\_id, table tags, **Lineage** tab, grants on external location.
* **Outcome:** Central governance (RBAC, masking, tags, audit) across ETL/ML/BI.

## **Platform / FinOps (optional)**

* **Owns:** Cost & reliability.
* **What you used:** **Serverless SQL** instead of clusters; Jobs scheduling; auto-scale.
* **Outcome:** Lower ops overhead, predictable cost, no EC2 quota pain.

# **One-liner per persona (for Q&A)**

* **DE:** “I productionized the ELT with Jobs; same data powers every team.”
* **Analyst:** “I answered business questions in Lakeview without copies.”
* **DS:** “I engineered features & a scoring path—drop-in replaceable by a model.”
* **Gov/Sec:** “I enforced masks/tags and proved lineage from S3 to dashboards.”

This shows you covered the full spectrum: **ingest → transform → govern → analyze → (ready to) model** on one Lakehouse.

**Business problem (short points)**

* Fragmented stack → data copied across tools; slow fraud/risk decisions
* Rising fraud attempts + chargebacks; manual triage is reactive
* Credit decisions inconsistent; hard to explain & audit
* Siloed analytics; no single source for customers/transactions
* Governance gaps on PII; limited lineage/traceability
* Cost & ops overhead from managing clusters/pipelines

# **What we did (one line)**

Built a **governed Lakehouse on Databricks** (S3 → Delta → UC) delivering **real-time fraud triage, credit risk scoring, and customer segmentation** with **Serverless Jobs** and **Lakeview**.

# **Internal impact (ops, teams) — short points**

* One data copy → faster ETL/BI/ML; fewer pipelines to maintain
* Serverless SQL + Jobs → lower ops & predictable cost
* Unity Catalog → masks/tags/lineage for audit & access control
* Reusable features/tables across DE/DS/BI (no re-work)
* Open formats (Parquet) → no lock-in; easy downstream sharing

# **External impact (customers, business) — short points**

* Faster fraud detection → fewer losses & chargebacks
* More consistent, explainable credit decisions (PD % view)
* Better targeting & retention via segments (RFM)
* Improved trust/compliance with provable lineage & masking
* Shorter time-to-insight for product & risk teams

# **What we proved (evidence)**

* **Fraud triage:** velocity features + rules → high-risk spikes surfaced in minutes
* **Credit risk:** PD heatmap by bureau band × delinquencies (clear risk gradients)
* **Segments:** RFM cohorts with counts & avg spend
* **Governance:** column masking, tags, end-to-end lineage in UC
* **Orchestration:** Job runs Bronze→Silver→Gold; scheduled, idempotent
* **Openness:** same gold tables exported to **Parquet** on S3

# **15-second talk track (say this)**

“Today we consolidated fraud, credit, and customer analytics on one **governed Lakehouse**. Data lands once in **S3**, becomes **Delta** under **Unity Catalog**, and powers **fraud triage, credit PD, and segmentation** in **Lakeview**—all orchestrated with **serverless Jobs**. The business gets faster, cheaper, and auditable decisions with no lock-in.”

# **If they ask “why Databricks vs current state?” (one-liner)**

“Because it collapses ETL, ML, and BI onto one platform—**less copy, more governance, lower cost**, and it stays **open** (Parquet) for any downstream system.”

# **Optional metric placeholders (if they press for impact)**

* ↓ Fraud false positives by **~X%**, ↓ review time by **~Y%**
* ↓ Pipeline run time from **hours → minutes**
* ↑ Analyst productivity (**1** dashboard, not **3** tools)
* Compliance: clear lineage/masking across sensitive fields

Use these bullets as your close, then click your **dashboard**, **lineage**, and **job** to back it up.

Looks solid! A few quick tweaks will make it tighter and 100% aligned with your demo.

# **Polished slide text (copy-paste ready)**

## **Technical challenges causing these business problems**

* **Scalability & concurrency:** ETL + BI contend for resources; small files; slow queries.
* **Data silos & copies:** multiple tools, duplicated pipelines, inconsistent truths.
* **Latency / streaming:** need near-real-time fraud features and alerting.
* **Multi-cloud / openness:** avoid lock-in; interop with other engines.
* **Collaboration & reuse:** DE/DS/BI on different stacks; hard to share features.
* **Governance & audit:** PII controls, lineage, and access at column/table level.
* **Ops & cost:** cluster wrangling, quotas, complex scheduling & monitoring.

## **Databricks benefits we delivered (with proofs)**

* **Faster time-to-insight:** single platform; **Serverless SQL + Jobs** → minutes to refresh (Job run).
* **Lower cost & simpler ops:** one copy on **Delta**; no cross-tool hops; serverless autoscale.
* **Better risk control:** **fraud features + rules** (v\_fraud\_alerts), **credit PD heatmap**, **RFM segments**.
* **Governed by design:** **Unity Catalog** masks, tags, RBAC, and **Lineage** from S3→Gold→Dashboard.
* **Open & future-proof:** **Parquet export** on S3 + external tables; path to DLT streaming & Mosaic AI.

# **Tiny edits to your slide**

* Use **12–14 pt body**, **18–24 pt section headers** so it’s readable on screen.
* Keep each bullet ≤ 10–12 words; bold the keyword at the start.
* Add a parenthetical “proof” where helpful (e.g., “(Lineage tab)”).
* Title suggestion: **“Technical Challenges → What We Solved”**.

# **What to say while this slide is up (10 sec)**

“Here are the technical pain points—scale, silos, latency, governance, and ops. We solved them on the **Databricks Lakehouse** with **Serverless SQL + Jobs**, **Delta + UC** governance, and **Lakeview** dashboards; and we proved openness by exporting **Parquet** to S3.”

If you want, paste a screenshot of your updated slide and I’ll do one last micro-edit on wording/spacing.

–

<https://chatgpt.com/share/68db568b-9fe8-800d-a235-fa10a65703d2>

id usha.mmrao@

username: urao rpNj2010!  
  
 Path A (no waiting): use Serverless and finish the demo without clusters

Serverless runs on Databricks’ compute, so it doesn’t use your AWS EC2 quota.

**SELECT \***

**FROM read\_files('s3://payer-demo-ur/fin/transactions.csv', format => 'csv', header => true);**

DROP SCHEMA IF EXISTS payer.default CASCADE;

DROP SCHEMA IF EXISTS payer.raw CASCADE;

DROP SCHEMA IF EXISTS payer.curated CASCADE;

DROP CATALOG IF EXISTS payer CASCADE;

--

**SELECT ts, fraud\_score, amount, account\_id, device\_id, mcc, country**

**FROM fin.curated.fraud\_scored\_gold**

**WHERE ts >= current\_timestamp() - INTERVAL 24 HOURS**

**AND fraud\_score >= 30**

**ORDER BY ts DESC, fraud\_score DESC**

**LIMIT 1000;**

--

**SELECT**

**CONCAT(FLOOR((bureau\_score-300)/50)\*50 + 300, '-', FLOOR((bureau\_score-300)/50)\*50 + 349) AS bureau\_band,**

**delinq\_12m,**

**AVG(GREATEST(pd\_percent, 0)) AS avg\_pd,**

**MIN(FLOOR((bureau\_score-300)/50)) AS sort\_idx**

**FROM fin.curated.credit\_risk\_scores\_gold**

**WHERE bureau\_score BETWEEN 300 AND 850**

**GROUP BY bureau\_band, delinq\_12m**

**ORDER BY sort\_idx, delinq\_12m;**

--

USE CATALOG `workspace`;

USE SCHEMA `default`;

SELECT segment, COUNT(\*) AS customers, AVG(total\_spend\_180d) AS avg\_spend

FROM fin.curated.customer\_rfm\_gold

GROUP BY segment

ORDER BY customers DESC;

--

Convert delta tables to parquet format

**SELECT \* FROM fin.curated.transactions\_silver;**

**-- Gold fraud**

**INSERT OVERWRITE DIRECTORY 's3://payer-demo-ur/fin/parquet/fraud\_scored/'**

**USING PARQUET**

**SELECT \* FROM fin.curated.fraud\_scored\_gold;**

**-- Gold credit risk**

**INSERT OVERWRITE DIRECTORY 's3://payer-demo-ur/fin/parquet/credit\_risk/'**

**USING PARQUET**

**SELECT \* FROM fin.curated.credit\_risk\_scores\_gold;**

**-- Gold segments**

**INSERT OVERWRITE DIRECTORY 's3://payer-demo-ur/fin/parquet/segments/'**

**USING PARQUET**

**SELECT \* FROM fin.curated.customer\_rfm\_gold;**

--

**“Lineage is the auto-captured, column-aware map of how data flows through our Lakehouse—from raw S3 files to Bronze/Silver/Gold tables to Jobs and dashboards—so we can do impact analysis, auditing, and debugging in seconds.”**

**Setup:**

# **Upload the data to S3**

1. **Unzip locally.**
2. **In the S3 console, upload all files to:**
   * **s3://payer-demo-ur/fin/**
   * **(optional streaming later) s3://payer-demo-ur/fin/streaming\_transactions/ — upload the transactions\_batch\_\*.csv here if you have them.**

**You see**

**fin/**

**customers.csv**

**accounts.csv**

**devices.csv**

**merchants.csv**

**transactions.csv**

**applications.csv**

**payments.csv**

**(streaming\_transactions/ <- optional micro-batches)**

# 1) Open a Serverless SQL warehouse

Databricks workspace → SQL → Warehouses → New → toggle Serverless: ON → create.

We’ll run all SQL from this warehouse (no EC2/vCPU needed).

# 2) Create Unity Catalog objects (where data lives & is governed)

Open SQL Editor (select your Serverless warehouse) and run:

-- Catalog rooted in your S3 bucket

CREATE CATALOG IF NOT EXISTS fin

MANAGED LOCATION 's3://payer-demo-ur/uc/fin';

GRANT USE CATALOG ON CATALOG fin TO `account users`;

-- Medallion schemas

CREATE SCHEMA IF NOT EXISTS fin.raw; -- Bronze

CREATE SCHEMA IF NOT EXISTS fin.curated; -- Silver/Gold

3) Bronze: land CSVs into **Delta** (minimal transforms)

USE fin.raw;

CREATE OR REPLACE TABLE customers\_bronze

USING DELTA AS

SELECT \* FROM read\_files('s3://payer-demo-ur/fin/customers.csv', format=>'csv', header=>true);

CREATE OR REPLACE TABLE accounts\_bronze

USING DELTA AS

SELECT \* FROM read\_files('s3://payer-demo-ur/fin/accounts.csv', format=>'csv', header=>true);

CREATE OR REPLACE TABLE merchants\_bronze

USING DELTA AS

SELECT \* FROM read\_files('s3://payer-demo-ur/fin/merchants.csv', format=>'csv', header=>true);

CREATE OR REPLACE TABLE devices\_bronze

USING DELTA AS

SELECT \* FROM read\_files('s3://payer-demo-ur/fin/devices.csv', format=>'csv', header=>true);

CREATE OR REPLACE TABLE transactions\_bronze

USING DELTA AS

SELECT \* FROM read\_files('s3://payer-demo-ur/fin/transactions.csv', format=>'csv', header=>true);

CREATE OR REPLACE TABLE apps\_bronze

USING DELTA AS

SELECT \* FROM read\_files('s3://payer-demo-ur/fin/applications.csv', format=>'csv', header=>true);

CREATE OR REPLACE TABLE payments\_bronze

USING DELTA AS

SELECT \* FROM read\_files('s3://payer-demo-ur/fin/payments.csv', format=>'csv', header=>true);

-- Delta "proof": versions & time travel

DESCRIBE HISTORY transactions\_bronze;

SELECT \* FROM transactions\_bronze VERSION AS OF 0 LIMIT 5;

4) Silver: clean, cast, standardize for analytics/ML

USE fin.curated;

CREATE OR REPLACE TABLE transactions\_silver

USING DELTA AS

SELECT

transaction\_id,

account\_id, customer\_id, device\_id, merchant\_id,

CAST(ts AS TIMESTAMP) AS ts,

CAST(amount AS DOUBLE) AS amount,

currency, channel, city, country, mcc,

CAST(is\_fraud AS INT) AS label\_is\_fraud

FROM fin.raw.transactions\_bronze

WHERE amount > 0 AND transaction\_id IS NOT NULL;

CREATE OR REPLACE TABLE apps\_silver

USING DELTA AS

SELECT

application\_id, customer\_id,

CAST(annual\_income AS INT) AS annual\_income,

CAST(bureau\_score AS INT) AS bureau\_score,

CAST(delinq\_12m AS INT) AS delinq\_12m,

CAST(inquiries\_6m AS INT) AS inquiries\_6m,

CAST(approved AS INT) AS approved,

CAST(default\_12m AS INT) AS default\_12m

FROM fin.raw.apps\_bronze;

CREATE OR REPLACE TABLE customers\_silver

USING DELTA AS

SELECT customer\_id,

CAST(age AS INT) AS age,

country,

TO\_DATE(created\_date) AS created\_date,

kyc\_status

FROM fin.raw.customers\_bronze;

# **5) Gold: build the 3 fintech outcomes**

## **A) Fraud detection (velocity features + rule score)**

-- 1) Velocity features (1h / 24h windows)

CREATE OR REPLACE TABLE txn\_features\_gold

USING DELTA AS

WITH base AS (

SELECT account\_id, device\_id, ts, amount, mcc, country

FROM fin.curated.transactions\_silver

),

feat AS (

SELECT

account\_id, device\_id, ts,

SUM(amount) OVER (PARTITION BY account\_id ORDER BY ts

RANGE BETWEEN INTERVAL 60 MINUTES PRECEDING AND CURRENT ROW) AS amt\_1h,

COUNT(\*) OVER (PARTITION BY account\_id ORDER BY ts

RANGE BETWEEN INTERVAL 60 MINUTES PRECEDING AND CURRENT ROW) AS cnt\_1h,

SUM(amount) OVER (PARTITION BY account\_id ORDER BY ts

RANGE BETWEEN INTERVAL 24 HOURS PRECEDING AND CURRENT ROW) AS amt\_24h,

COUNT(\*) OVER (PARTITION BY account\_id ORDER BY ts

RANGE BETWEEN INTERVAL 24 HOURS PRECEDING AND CURRENT ROW) AS cnt\_24h

FROM base

)

SELECT \* FROM feat;

-- 2) Join device/merchant risk + produce fraud\_score 0-100

CREATE OR REPLACE TABLE fraud\_scored\_gold

USING DELTA AS

SELECT

t.transaction\_id, t.account\_id, t.customer\_id, t.device\_id, t.merchant\_id, t.ts, t.amount, t.country, t.mcc,

f.amt\_1h, f.cnt\_1h, f.amt\_24h, f.cnt\_24h,

CASE WHEN d.is\_emulator = 1 THEN 20 ELSE 0 END

+ CASE WHEN m.mcc LIKE '7995%' THEN 20 ELSE 0 END

+ CASE WHEN f.cnt\_1h >= 5 OR f.amt\_1h >= 1000 THEN 20 ELSE 0 END

+ CASE WHEN f.cnt\_24h >= 20 OR f.amt\_24h >= 2500 THEN 20 ELSE 0 END

+ CASE WHEN t.country <> c.country THEN 20 ELSE 0 END

AS fraud\_score

FROM fin.curated.transactions\_silver t

JOIN txn\_features\_gold f ON t.account\_id=f.account\_id AND t.ts=f.ts

LEFT JOIN fin.raw.devices\_bronze d ON t.device\_id=d.device\_id

LEFT JOIN fin.raw.merchants\_bronze m ON t.merchant\_id=m.merchant\_id

LEFT JOIN fin.raw.customers\_bronze c ON t.customer\_id=c.customer\_id;

-- Triage view for Ops

CREATE OR REPLACE VIEW v\_fraud\_alerts AS

SELECT \*

FROM fraud\_scored\_gold

WHERE fraud\_score >= 60

ORDER BY ts DESC, fraud\_score DESC

LIMIT 200;

B) Credit risk scoring (transparent demo score)

CREATE OR REPLACE TABLE credit\_risk\_scores\_gold

USING DELTA AS

SELECT

a.application\_id, a.customer\_id, a.annual\_income, a.bureau\_score, a.delinq\_12m, a.inquiries\_6m, a.approved, a.default\_12m,

ROUND( 100 \* ( 0.05

+ (700 - a.bureau\_score) / 2000.0

+ a.delinq\_12m \* 0.03

+ a.inquiries\_6m \* 0.01

+ CASE WHEN a.annual\_income < 30000 THEN 0.03 ELSE 0 END

), 1) AS pd\_percent

FROM apps\_silver a;

CREATE OR REPLACE VIEW v\_credit\_risk\_triage AS

SELECT \*

FROM credit\_risk\_scores\_gold

ORDER BY pd\_percent DESC

LIMIT 200;

C) Customer segmentation (RFM)

CREATE OR REPLACE TABLE customer\_rfm\_gold

USING DELTA AS

WITH tx AS (

SELECT customer\_id,

MAX(ts) AS last\_txn\_ts,

COUNT(\*) AS txn\_count\_180d,

SUM(amount) AS total\_spend\_180d

FROM fin.curated.transactions\_silver

WHERE ts >= current\_timestamp() - INTERVAL 180 DAYS

GROUP BY customer\_id

),

scores AS (

SELECT

t.customer\_id,

DATEDIFF(current\_timestamp(), t.last\_txn\_ts) AS recency\_days,

t.txn\_count\_180d,

t.total\_spend\_180d,

NTILE(5) OVER (ORDER BY -t.txn\_count\_180d) AS freq\_quint,

NTILE(5) OVER (ORDER BY -t.total\_spend\_180d) AS mon\_quint,

NTILE(5) OVER (ORDER BY t.last\_txn\_ts) AS rec\_quint

FROM tx t

)

SELECT

customer\_id, recency\_days, txn\_count\_180d, total\_spend\_180d,

rec\_quint, freq\_quint, mon\_quint,

CASE

WHEN rec\_quint >=4 AND freq\_quint >=4 AND mon\_quint >=4 THEN 'CHAMPIONS'

WHEN rec\_quint >=3 AND freq\_quint >=3 THEN 'LOYAL'

WHEN rec\_quint <=2 AND freq\_quint <=2 THEN 'AT\_RISK'

ELSE 'REGULAR'

END AS segment

FROM scores;

CREATE OR REPLACE VIEW v\_segments AS

SELECT segment, COUNT(\*) AS customers, AVG(total\_spend\_180d) AS avg\_spend

FROM customer\_rfm\_gold

GROUP BY segment

ORDER BY customers DESC;

6) Governance (Unity Catalog): masking, tags, lineage

-- Column masking demo

CREATE OR REPLACE FUNCTION fin.phi\_mask(s STRING)

RETURNS STRING

RETURN CASE WHEN is\_account\_group\_member('analyst\_restricted')

THEN '\*\*\*MASKED\*\*\*' ELSE s END;

ALTER TABLE fin.raw.customers\_bronze ALTER COLUMN customer\_id SET MASK fin.phi\_mask;

ALTER TABLE fin.raw.devices\_bronze ALTER COLUMN ip\_address SET MASK fin.phi\_mask;

-- Classification tags (surface in the UI + lineage)

ALTER TABLE fin.curated.transactions\_silver SET TAGS ('DOMAIN'='PAYMENTS','SENSITIVITY'='PII');

ALTER TABLE fin.curated.credit\_risk\_scores\_gold SET TAGS ('DOMAIN'='LENDING','MODEL'='demo');

7) Dashboard (Lakeview) – 3 tiles

Fraud Ops

**SELECT ts, fraud\_score, amount, account\_id, device\_id, mcc, country**

**FROM fin.curated.fraud\_scored\_gold**

**WHERE ts >= current\_timestamp() - INTERVAL 24 HOURS**

**ORDER BY ts DESC, fraud\_score DESC**

**LIMIT 500;**

Credit Risk Heatmap

**SELECT bureau\_score, delinq\_12m, AVG(pd\_percent) AS avg\_pd**

**FROM fin.curated.credit\_risk\_scores\_gold**

**GROUP BY bureau\_score, delinq\_12m;**

Customer Segments

**SELECT \* FROM fin.curated.v\_segments;**

# 8) (Optional) Make it a repeatable pipeline with Jobs

**Workflows → Jobs → Create job (use your Serverless warehouse for each task):**

* **Task 1 (SQL): Bronze statements (Step 3)**
* **Task 2 (SQL): Silver statements (Step 4) — depends on Task 1**
* **Task 3 (SQL): Gold statements (Step 5) — depends on Task 2**
* **Task 4 (SQL): Governance (Step 6) — depends on Task 3**
* **(Optional) Add a schedule (hourly/daily)**

**Later, when EC2 vCPU limit is raised, port the fraud part to Auto Loader + Delta Live Tables using the same tables/paths.**

**Summary:**

**“One copy of data in Delta on S3; Unity Catalog governs access, masking, tags, lineage.”  
“Bronze→Silver→Gold transforms run on a Serverless SQL warehouse (no EC2).”  
“We deliver fraud alerts, credit PD, and segments, and the same tables power the Lakeview dashboard—no copies.”  
“When we need real-time, we switch to Auto Loader + DLT using the same schemas.”**

**IP address mask- demo 10/8**

**DESCRIBE EXTENDED fin.raw.customers\_bronze; -- look for: MASK: fin.curated.phi\_mask on customer\_id**

**DESCRIBE EXTENDED fin.raw.devices\_bronze; -- look for: MASK: fin.curated.phi\_mask on ip\_address**

**SHOW FUNCTIONS IN fin.curated LIKE 'phi\_mask';**

**-- Temporarily redefine the policy to mask unless you are in 'data\_privileged'**

**CREATE OR REPLACE FUNCTION fin.curated.phi\_mask(s STRING)**

**RETURNS STRING**

**RETURN CASE WHEN is\_account\_group\_member('data\_privileged') THEN s ELSE '\*\*\*MASKED\*\*\*' END;**

**-- Re-apply is NOT needed because the column still points to fin.curated.phi\_mask**

**SELECT customer\_id FROM fin.raw.customers\_bronze LIMIT 5;**

**SELECT ip\_address FROM fin.raw.devices\_bronze LIMIT 5;**

**CREATE OR REPLACE FUNCTION fin.curated.phi\_mask(s STRING)**

**RETURNS STRING**

**RETURN CASE WHEN is\_account\_group\_member('analyst\_restricted') THEN '\*\*\*MASKED\*\*\*' ELSE s END;**

**how should i prove this 04\_governanace and how do we validate**

###### **ChatGPT said:**

Thought for 9s

Perfect—here’s a quick, reliable way to **prove** and **validate** your 04\_governance step live. It’s designed so you can run it end-to-end in the SQL editor on your Serverless warehouse.

# **A) Prove column masking works**

### **1) Show the policy is attached (static proof)**

DESCRIBE EXTENDED fin.raw.customers\_bronze; -- look for: MASK: fin.curated.phi\_mask on customer\_id

DESCRIBE EXTENDED fin.raw.devices\_bronze; -- look for: MASK: fin.curated.phi\_mask on ip\_address

SHOW FUNCTIONS IN fin.curated LIKE 'phi\_mask';

### **2) Live masking demo (visible change)**

You need the column to appear masked for *your* session. The simplest way is to **flip the policy** to mask for everyone except a “privileged” group you’re **not** in.

-- Temporarily redefine the policy to mask unless you are in 'data\_privileged'

CREATE OR REPLACE FUNCTION fin.curated.phi\_mask(s STRING)

RETURNS STRING

RETURN CASE WHEN is\_account\_group\_member('data\_privileged') THEN s ELSE '\*\*\*MASKED\*\*\*' END;

-- Re-apply is NOT needed because the column still points to fin.curated.phi\_mask

Now query the PII columns:

SELECT customer\_id FROM fin.raw.customers\_bronze LIMIT 5;

SELECT ip\_address FROM fin.raw.devices\_bronze LIMIT 5;

You should see \*\*\*MASKED\*\*\*.

**(Optional) Unmask control test:** If you *do* have a group where you’re a member, adapt the function to your group name and show real values appear. Otherwise, just keep the above for the demo.

After the demo, restore your original intent (mask only a restricted group):

CREATE OR REPLACE FUNCTION fin.curated.phi\_mask(s STRING)

RETURNS STRING

RETURN CASE WHEN is\_account\_group\_member('analyst\_restricted') THEN '\*\*\*MASKED\*\*\*' ELSE s END;

# **B) Prove tags are applied**

### **1) Show table tags in SQL**

SHOW TBL TAGS ON TABLE fin.curated.transactions\_silver;

SHOW TBL TAGS ON TABLE fin.curated.credit\_risk\_scores\_gold;

You should see:

* ('DOMAIN','PAYMENTS'), ('SENSITIVITY','PII')
* ('DOMAIN','LENDING'), ('MODEL','demo')

### **2) Show tags in the UI**

Catalog Explorer → open each table → “**Tags**” panel shows the same key/value pairs.

# **C) Prove lineage is captured**

In the UI:

1. Catalog Explorer → fin.curated.fraud\_scored\_gold (or transactions\_silver).
2. Click **Lineage** → call out **Upstream** (Bronze tables, features step, job run) and **Downstream** (views, dashboards).
3. Toggle **“Column lineage”** and highlight how country or fraud\_score are derived.

One-liner: “Upstream = inputs; downstream = consumers. We see end-to-end from S3 → Bronze/Silver/Gold → dashboards.”

# **D) Prove access control exists (lightweight)**

Even if you can’t impersonate another user, you can **show the policy surface**:

-- Who currently has what?

SHOW GRANTS ON TABLE fin.raw.customers\_bronze;

SHOW GRANTS ON EXTERNAL LOCATION `db\_s3\_external\_databricks-s3-ingest-e68bd`;

(If you have a test group, you can GRANT SELECT ON TABLE … TO <group> and show it appear.)

# **E) Prove the External Location is configured**

SHOW EXTERNAL LOCATIONS;

DESCRIBE EXTERNAL LOCATION `db\_s3\_external\_databricks-s3-ingest-e68bd`;

Call out that it points at s3://payer-demo-ur/ and is governed by UC (no access keys in code).

## **30-second talk track to say while you click**

“Governance is enforced at the **data layer** with Unity Catalog. We attach a **masking policy** to PII columns—watch them flip to \*\*\*MASKED\*\*\*—so even if someone queries directly, they only see what they’re allowed. We **tag** key tables by domain and sensitivity, which shows up in **search and lineage**. Lineage gives upstream/downstream impact for audit. Access is managed centrally with **grants** and a governed **External Location** to S3—no hard-coded keys.”

## **Quick cleanup after the demo**

* Restore the masking function to your desired policy (as shown above).
* (Optional) Keep the tags and lineage—they’re safe and useful.

-- Compare the counts directly

SELECT

(SELECT COUNT(\*) FROM fin.raw.transactions\_bronze WHERE amount > 0) AS bronze\_count,

(SELECT COUNT(\*) FROM fin.curated.transactions\_silver WHERE amount > 0) AS silver\_count;

-- Compare the counts directly

SELECT

(SELECT COUNT(\*) FROM fin.raw.transactions\_bronze WHERE amount > 0) AS bronze\_count,

(SELECT COUNT(\*) FROM fin.curated.transactions\_silver WHERE amount > 0) AS silver\_count;

-- check type conversions for columns

DESCRIBE EXTENDED fin.curated.transactions\_silver;

DESCRIBE EXTENDED fin.raw.transactions\_bronze;

SHOW CATALOGS;

SHOW SCHEMAS IN fin;

SHOW TABLES IN fin.raw;

SHOW TABLES IN fin.curated;

SELECT DISTINCT table\_schema, table\_name

FROM information\_schema.columns

ORDER BY table\_schema, table\_name

LIMIT 100;

SELECT DISTINCT table\_schema, table\_name

FROM fin.information\_schema.columns

ORDER BY table\_schema, table\_name

LIMIT 100;

-- bronze and silver comparison

SELECT DISTINCT table\_schema, table\_name

FROM fin.information\_schema.columns

WHERE table\_schema = 'raw'

ORDER BY table\_name;

WITH bronze AS (

SELECT

column\_name,

data\_type,

is\_nullable

FROM fin.information\_schema.columns

WHERE table\_schema = 'raw'

AND table\_name = 'transactions\_bronze'

),

silver AS (

SELECT

column\_name,

data\_type,

is\_nullable

FROM fin.information\_schema.columns

WHERE table\_schema = 'curated'

AND table\_name = 'transactions\_silver'

)

SELECT

COALESCE(b.column\_name, s.column\_name) AS column\_name,

b.data\_type AS bronze\_data\_type,

b.is\_nullable AS bronze\_nullable,

s.data\_type AS silver\_data\_type,

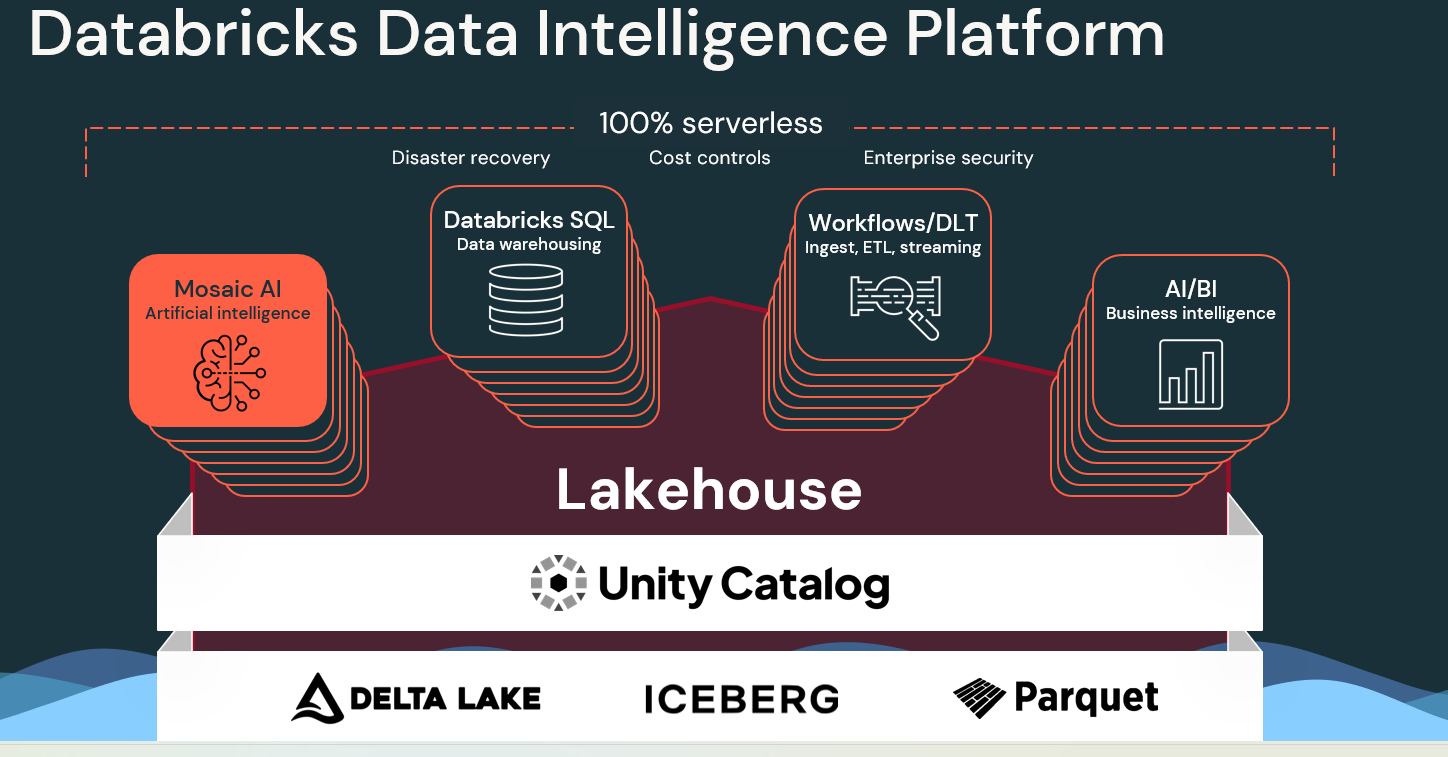
s.is\_nullable AS silver\_nullable

FROM bronze b

FULL OUTER JOIN silver s ON b.column\_name = s.column\_name

ORDER BY column\_name;

–



Giving me good summary of every component in this databricks data intellience platform architecture. comapre at a hight level delta lake vs iceberg

###### **ChatGPT said:**

Here’s a clear summary of every component shown in the **Databricks Data Intelligence Platform** architecture diagram, plus a **high-level comparison of Delta Lake vs Iceberg**:

## **🧠 Databricks Data Intelligence Platform — Component Overview**

### **1. Lakehouse**

* **Core concept:** Combines the best of **data lakes** (scalable, cheap storage) and **data warehouses** (structured data and governance).
* **Purpose:** One unified platform for all workloads — data engineering, BI, AI/ML.
* **Key benefit:** Avoids data silos; you store once in open formats (Parquet, Delta, Iceberg) and access through multiple engines.

### **2. Unity Catalog**

* **Function:** Centralized **governance and metadata layer**.
* **Responsibilities:**
  + Manages **data access control** (table, column, or row level).
  + Provides **data lineage**, **audit logs**, and **fine-grained permissions**.
  + Unifies governance for **files, tables, ML models, and dashboards**.
* **Goal:** Consistent governance across all data and AI assets, across clouds and workspaces.

### **3. Mosaic AI (Artificial Intelligence Layer)**

* **Purpose:** Integrates AI directly into the Lakehouse.
* **Includes:** Foundation models, vector search, and model serving.
* **Use case:** Build and deploy **GenAI, LLMs, and machine learning applications** using your enterprise data.
* **Benefit:** Unified environment for AI and analytics with secure data access and lineage.

### **4. Databricks SQL**

* **Purpose:** **Data warehousing** and **ad-hoc analytics** directly on the Lakehouse.
* **Capabilities:**
  + Fully **serverless** query execution.
  + Supports **BI dashboards**, SQL editors, and **data exploration**.
  + **Performance-tuned** for large-scale queries using Photon engine.
* **Replaces:** Traditional warehouses like Snowflake or Redshift by querying open formats.

### **5. Workflows / DLT (Delta Live Tables)**

* **Purpose:** Managed **ETL, ingestion, and streaming pipelines**.
* **DLT:** Declarative pipeline framework for building reliable, self-healing data pipelines.
* **Benefits:**
  + Automated **data quality checks**.
  + Handles **batch and streaming** seamlessly.
  + Integrated **monitoring and versioning**.

### **6. AI / BI (Business Intelligence Layer)**

* **Purpose:** Surface insights to business users.
* **Includes:** Integration with tools like **Power BI, Tableau, Looker**, or native Databricks dashboards.
* **Goal:** Democratize access to data-driven insights across technical and non-technical teams.

### **7. Infrastructure Pillars (Top Layer Notes)**

* **100% Serverless:** No cluster management; scales automatically.
* **Disaster Recovery, Cost Controls, Enterprise Security:** Built-in platform-level services ensuring compliance, resiliency, and optimization.

## **⚙️ Storage Foundation (Bottom Layer)**

### **Delta Lake, Iceberg, Parquet**

These are **open table and file formats** that power the Lakehouse:

| **Feature** | **Delta Lake** | **Apache Iceberg** | **Parquet** |
| --- | --- | --- | --- |
| **Type** | Table format (built on Parquet) | Table format (built on Parquet/ORC) | File format |
| **Transaction Support** | ACID transactions (Databricks-native) | ACID transactions (open standard) | None |
| **Schema Evolution** | Supported (automatic) | Supported (requires commit) | None |
| **Time Travel** | Yes (versioned data) | Yes (snapshot-based) | No |
| **Performance Optimizations** | Z-ordering, caching, data skipping | Partition evolution, hidden partitions | None |
| **Integration** | Deeply integrated with Databricks | Broad ecosystem (Snowflake, Flink, etc.) | Universal file format |
| **Best For** | Databricks-native Lakehouse workloads | Multi-engine open data lakehouses | Simple storage or exchange |

**In short:**

* **Delta Lake** → optimized for **Databricks Lakehouse** (tightest integration, faster performance).
* **Iceberg** → best for **open multi-engine interoperability** (used by Snowflake, Athena, BigQuery).
* **Parquet** → base **file storage format** (no metadata or transactions).

Would you like me to add a **visual comparison table** (Delta vs Iceberg) or a **use-case summary** (when to choose each)?

##### **You said:**

yes please

###### **ChatGPT said:**

Perfect 👍 — here’s both:

## **🧩 Visual Comparison: Delta Lake vs Apache Iceberg**

| **Category** | **Delta Lake (Databricks)** | **Apache Iceberg (Open Source)** |
| --- | --- | --- |
| 🏗️ **Architecture Type** | Transactional **table format** built on Parquet | Transactional **table format** built on Parquet/ORC/Avro |
| ⚙️ **Transaction Model** | Optimized **commit log (Delta Log)** stored in \_delta\_log | Uses **manifest and snapshot metadata files** for commits |
| 🔄 **Concurrency Handling** | Optimistic concurrency with conflict detection | MVCC (multi-version concurrency control) |
| 🧾 **Schema Evolution** | Fully automatic (merge, rename, add columns) | Supported but requires explicit commits |
| 🕰️ **Time Travel / Versioning** | Yes — easy SQL syntax (VERSION AS OF, TIMESTAMP AS OF) | Yes — via snapshots and metadata references |
| 🚀 **Performance Optimizations** | Z-Ordering, Data Skipping, Caching, Liquid Clustering (Databricks) | Partition Evolution, Hidden Partitions |
| 🔍 **Query Engines Support** | Spark, Databricks SQL, Trino, Presto (limited external engines) | Spark, Flink, Trino, Presto, Hive, Snowflake, Athena, BigQuery |
| 🔒 **Governance & Security** | Deeply integrated with **Unity Catalog** | External governance via tools like AWS Glue or Apache Ranger |
| 💡 **Ecosystem Focus** | **Databricks-native** (tight integration, best performance) | **Open ecosystem** (multi-engine, vendor-neutral) |
| ☁️ **Cloud Adoption** | Native in Databricks (Azure, AWS, GCP) | Supported by multiple platforms (Snowflake, AWS, GCP, Cloudera) |
| 🧮 **Use Cases** | Unified Lakehouse workloads, AI/ML, streaming + batch pipelines | Cross-platform lakehouse, federated analytics, multi-engine interoperability |
| 🧰 **Tooling** | Delta Live Tables, Unity Catalog, Photon Engine, MLflow | Iceberg REST Catalog, Nessie, Project Nessie integration |
| 🔗 **Openness** | Open Source (Linux Foundation Delta 3.0) but Databricks-optimized | Fully Open Apache Foundation Project |

## **🧭 When to Choose Each**

| **Scenario** | **Recommended Format** | **Why** |
| --- | --- | --- |
| You’re primarily using **Databricks** | ✅ **Delta Lake** | Native integration, fastest performance, Unity Catalog & AI integration |
| You need to **share data across multiple compute engines** (e.g., Snowflake, Athena, Flink) | ✅ **Iceberg** | Best interoperability; open standard with wide engine support |
| You need **governed AI/ML + BI on a single platform** | ✅ **Delta Lake** | Tight integration with Mosaic AI, Databricks SQL, and Workflows |
| You want **vendor-neutral lakehouse** with multi-cloud strategy | ✅ **Iceberg** | Open architecture, easily portable |
| You need **streaming + batch unification** | ✅ **Delta Lake** | Strong native support with Delta Live Tables |
| You’re focused on **open data sharing and long-term compatibility** | ✅ **Iceberg** | Broad community adoption and open governance |

### **🏁 Summary**

* **Delta Lake** → Best for **Databricks-native workloads** where performance, governance, and AI integration matter most.
* **Iceberg** → Best for **multi-engine interoperability** and open data lake architectures.
* Both support **open formats, ACID transactions, and time travel**, but Delta wins in **tight ecosystem performance**, while Iceberg wins in **cross-platform flexibility**.

Would you like me to add a **diagram-style summary (visual chart comparing both)** or a **real-world example** (e.g., how Netflix or Adobe uses each)?

##### **You said:**

yes

###### **ChatGPT said:**

Great — here’s both:

## **🧩 1. Diagram-Style Summary: Delta Lake vs Iceberg**

┌──────────────────────────────────────────────┐

│ Lakehouse Table Formats │

└──────────────────────────────────────────────┘

▲

│

┌──────────────────────────┼──────────────────────────┐

│ │ │

▼ ▼ ▼

┌───────────────┐ ┌────────────────┐ ┌────────────────┐

│ Delta Lake │ │ Apache Iceberg│ │ Parquet │

│ (Databricks) │ │ (Open Source) │ │ (Base file fmt) │

└───────────────┘ └────────────────┘ └────────────────┘

│ │

│ │

▼ ▼

┌──────────────┐ ┌───────────────┐

│ Optimized for│ │ Built for │

│ Databricks │ │ open engines │

│ AI + BI + ETL│ │ (Spark, Flink)│

└──────────────┘ └───────────────┘

│ │

▼ ▼

┌──────────────┐ ┌───────────────┐

│ Serverless │ │ Interoperable │

│ Unity Catalog│ │ & portable │

└──────────────┘ └───────────────┘

│ │

▼ ▼

✅ Best for: ✅ Best for:

- Databricks AI/BI - Open multi-engine

- High performance - Federated analytics

- Governed data sharing - Cloud-neutral strategy

## **🧠 2. Real-World Examples**

### **🔶 Delta Lake – Databricks Ecosystem**

| **Company** | **Use Case** | **Why Delta?** |
| --- | --- | --- |
| **Comcast** | Unified analytics across petabytes of streaming data | Delta’s ACID + streaming capabilities made ETL simpler |
| **Shell** | AI-driven predictive maintenance and sustainability models | Tight integration with Databricks ML and Unity Catalog |
| **HSBC** | Financial reporting and governance | Compliance-grade lineage and access control (Unity Catalog) |

💡 *Pattern:* Delta Lake shines when the organization is **all-in on Databricks**, and wants a **single platform for data + AI**.

### **🔷 Apache Iceberg – Open Lakehouse Ecosystem**

| **Company** | **Use Case** | **Why Iceberg?** |
| --- | --- | --- |
| **Netflix** | Centralized catalog serving Spark, Trino, and Flink pipelines | Multi-engine interoperability for their data platform |
| **Adobe** | Marketing analytics across clouds | Vendor-neutral design, easy integration with multiple query engines |
| **LinkedIn** | Real-time analytical pipelines | Support for snapshot isolation and partition evolution at scale |

💡 *Pattern:* Iceberg dominates when companies have **polyglot environments** (Spark + Flink + Trino + Presto) and want **open, future-proof governance**.

### **🧭 Summary Recommendation**

| **Strategy** | **Format to Choose** | **Rationale** |
| --- | --- | --- |
| Databricks-first enterprise | **Delta Lake** | Tightest integration, best performance, managed governance |
| Multi-engine, multi-cloud platform | **Iceberg** | Broad interoperability, open standard, avoids vendor lock-in |
| Data sharing + AI workloads | **Delta Lake** | Native with Mosaic AI and Unity Catalog |
| Federated analytics across tools | **Iceberg** | Strong cross-tool metadata and catalog integration |

Would you like me to make a **visual chart (image)** that shows this Delta vs Iceberg comparison — e.g., a 2-column infographic version of this summary? It’d be perfect for slides or architecture decks.

<https://chatgpt.com/share/68e908bb-4484-8004-abeb-39e4c036c7fb>

https://dbc-125af2c1-cdee.cloud.databricks.com/dashboardsv3/01f0a337abf11185bd79456bbf129eda/published?o=2891117215919201

Awesome—your deck is on-point. Here’s a hiring-manager style review + the tough Q&A you’re likely to get (with crisp answers), plus a 1-slide architecture summary you can read out.

# **Quick review (what’s good / any gaps)**

* You clearly frame business + technical challenges and map them to Databricks benefits and a Tell-Show-Tell demo path. Keep that—interviewers love the discipline.
* Your demo storyline (S3→UC, Delta history/time travel, 3 tiles, governance, jobs, openness) matches what you actually built.
* Close/recap slides already say the right words—reuse them verbatim at the end.

# **Tough questions you’ll likely get (and strong answers)**

## **1) Why Databricks for this fintech problem?**

* **Answer (10s):** “We land once in S3, then run **ETL, ML, and BI on the same Delta tables**, governed by **Unity Catalog**. That removes copies/silos, speeds up fraud & credit decisions, and lowers ops with **serverless SQL + Jobs**.”

## **2) What did you actually build end-to-end?**

* **Answer:** “CSV in S3 → Bronze (Delta) → Silver (typed/standardized) → Gold tables for **fraud triage**, **credit PD**, **customer segments**; **Lakeview** dashboard; **Unity Catalog** masks/tags/lineage; **Workflows** job to orchestrate.”

## **3) Did you copy data or only register metadata?**

* **Answer:** “We used **CTAS into managed Delta**, which **materializes** data under UC storage (Parquet + \_delta\_log). If we wanted metadata-only, we’d create an **external table** over the CSV path. For this demo, we materialized to get ACID/time-travel/perf.”  
   (Tie back to your “Delta proof: History / VERSION AS OF”.)

## **4) How do you prove governance is real, not slides?**

* **Answer:** “Column **masking** on PII, **tags** (DOMAIN/SENSITIVITY), and **lineage** from S3→Gold→Dashboard. It’s enforced at the **data layer**, so SQL, jobs, and dashboards inherit it.” (In the demo, show masks/tags + Lineage tab.)

## **5) Why Delta Lake vs Apache Iceberg?**

* **Short, neutral comparison tailored to your build:**
  + **ACID & time travel:** Both support ACID and time travel; Delta’s been battle-tested on Databricks for years and is deeply integrated (OPTIMIZE/Z-ORDER/CDC), which is why we used it for **fraud velocity features** and **Gold tables** quickly.
  + **Governance & lineage:** With **Unity Catalog**, Delta gets turnkey **column masks, tags, RBAC, lineage**—critical in fintech. Iceberg can integrate with other catalogs but would require more assembly to match this end-to-end experience.
  + **Performance/ops:** Delta + Databricks **Serverless SQL** and Delta optimizations give strong performance without cluster wrangling—important since we hit **AWS vCPU quotas** earlier; serverless avoided that pitfall.
  + **Openness:** We also **export to open Parquet** (and could register external tables). So we’re not locked in either way.
* **One-liner:** “For this demo, **Delta + UC** gave us the fastest path to governed, auditable outcomes; we still keep **open formats** for interop.”

## **6) Why CSV as source, not Parquet?**

* **Answer:** “The source system exports CSV; we convert to **Delta** on ingest for ACID/time-travel/perf. If interop is needed, we can export **Parquet** later—our deck even calls that out as ‘openness/no lock-in’.”

## **7) What *business* results did you produce?**

* **Answer:** “Fraud ops gets a **triage list in minutes**; credit sees **PD gradients** by score×delinq; marketing gets **segments** for action. All are governed and refresh on a **serverless** schedule.”

## **8) How would you make fraud truly real-time?**

* **Answer:** “Swap batch windows to **Auto Loader + DLT** with streaming windows; keep the same Delta tables and Lakeview tiles. The deck lists this as a recommended next step.”

## **9) Security & access to S3 without keys?**

* **Answer:** “We used a **governed External Location** in UC to S3 (assume-role with external ID), so no embedded credentials; permissions are granted centrally.” (You already showcased External Location + grants in Catalog Explorer.)

## **10) How do you keep costs down?**

* **Answer:** “**Serverless SQL + Jobs** (no clusters to manage), one copy of data on Delta, and pipeline consolidation. That’s in the benefits slide.”

# **One-slide architecture you can speak to**

**Current** (pain): silos, copies, slow fraud/credit, governance gaps.

**Target / What you built:**

1. **S3** (landing CSV) →
2. **Delta Lake (Bronze/Silver/Gold)** managed under **Unity Catalog** (masks, tags, lineage) →
3. **Workflows (serverless SQL tasks)** orchestrate Bronze→Silver→Gold→Governance →
4. **Lakeview** dashboard (Fraud triage, Credit PD heatmap, Segments) →
5. **Open Parquet** export (optional) for interop.

**Why it works:** single governed platform (ETL+ML+BI), serverless performance, open formats, streaming-ready path.

# **Rapid-fire “why this / why that” mini-answers**

* **Why Serverless vs clusters?** Hit AWS quota limits; serverless removed that ops risk and sped up delivery. (Also aligns with deck’s serverless positioning.)
* **Why Jobs & not DLT today?** Quicker to stand up with SQL in your 60-min demo; DLT is the next step for streaming SLAs.
* **Why UC masks/tags?** Column-level privacy, discoverability, and auditable lineage across all personas/tools.
* **How do you rollback mistakes?** Delta **time travel** (VERSION AS OF) on Silver/Gold; you even planned to show that.
* **Where’s the “no lock-in” proof?** Parquet export + (optional) external tables over S3 folders.

# **Final polish tips for the live demo**

* Open **transactions\_silver → History**; run a quick SELECT \* FROM … VERSION AS OF <n> LIMIT 5; to show time travel.
* On **Lineage**, say “Upstream = inputs; Downstream = consumers” and read your counts out loud.
* In **Lakeview**, filter fraud tile with min\_score to show triage working.
* End with your close slide (it already nails the messaging).

If you want, paste your final architecture slide and I’ll do a 30-second micro-edit for wording/layout.

Awesome — here’s a tight, interview-ready pack you can use verbatim.

# **Hiring-manager Q&A (short, strong answers)**

## **Why Databricks for this fintech use case?**

* **One platform:** ETL, ML and BI on the **same Delta tables** (no copies/silos).
* **Governed by design:** **Unity Catalog** gives RBAC, column masking, tags, and **lineage**.
* **Speed with less ops:** **Serverless SQL + Workflows** → refresh in minutes, no cluster wrangling.
* **Streaming-ready:** Same tables work for batch today, streaming (Auto Loader/DLT) tomorrow.
* **Open & portable:** Data lives in your S3; we can **export Parquet**—no lock-in.

## **How is Databricks different from other solutions?**

* **Versus classic data warehouses:** Warehouses are great for BI, weaker for data science/streaming and often require **copies** of data. Databricks runs **SQL + Python/ML + streaming** on one lakehouse.
* **Versus DIY data lakes:** Raw lake + glue code lacks **ACID, governance, lineage** and consistent tables. Delta + UC makes the lake **transactional & governed**.
* **Versus point ML tools:** Those don’t manage data/ETL/BI end-to-end. Databricks does, with **production pipelines + dashboards** on the same data.

## **Benefits you delivered in this demo (tie to business)**

* **Fraud triage** in near-real-time with features + a transparent **fraud\_score**.
* **Credit risk** with an explainable **PD% heatmap** (score band × delinquencies).
* **Customer segmentation** (RFM) for marketing actions.
* **Governance proof:** masks on PII, business tags, end-to-end lineage.
* **Ops & cost:** serverless, scheduled **Jobs**, zero cluster quota drama, open Parquet export.

# **Delta Lake vs. Apache Iceberg (simple & neutral)**

**Both** are open lakehouse table formats that sit on files in object storage.

| **Topic** | **Delta Lake** | **Apache Iceberg** |
| --- | --- | --- |
| Storage | Parquet files + **\_delta\_log** (JSON/Checkpoint) | Parquet/ORC/Avro + **snapshot manifests** |
| Core strengths | Tight **Databricks** integration: ACID, **time travel**, MERGE/UPDATE/DELETE/CDC, **OPTIMIZE/Z-ORDER**, streaming with DLT | **Engine-agnostic** design, strong **partition evolution/hidden partitioning**, open catalogs (REST, Hive) |
| Governance | First-class with **Unity Catalog** (RBAC, masks, tags, lineage) | Needs external catalog/governance tooling |
| Performance | Photon/Delta optimizations, data skipping, Z-ORDER | Broad engine support; tuning depends on engine |
| When I’d choose it | Fastest path on Databricks to governed analytics/ML | Heterogeneous engines where Iceberg is the standard |

**What you did:** picked **Delta + UC** for speed/governance on Databricks, and kept **Parquet export** to prove openness.

# **Your demo cheat sheet (everything you built)**

## **Architecture (one slide you can draw)**

1. **S3 landing (CSV):** s3://payer-demo-ur/fin/...
2. **Delta tables (UC managed):** **Bronze → Silver → Gold** in catalog fin
3. **Governance (UC):** **Masks**, **Tags**, **Lineage**, **External Location** to S3
4. **Workflows (Serverless SQL):** 01\_bronze → 02\_silver → 03\_gold → 04\_governance
5. **Lakeview dashboard:** Fraud triage, Credit PD heatmap, Customer segments
6. **Interoperability:** optional **Parquet export** to s3://payer-demo-ur/fin/parquet/...

## **Key objects to name-drop**

* **Catalog/Schemas:** fin.raw (Bronze), fin.curated (Silver/Gold)
* **Bronze tables:** \*\_bronze from CSV via read\_files(...)
* **Silver tables:** transactions\_silver, apps\_silver, customers\_silver (typed, standardized)
* **Gold tables:**
  + Fraud: txn\_features\_gold, fraud\_scored\_gold, view v\_fraud\_alerts
  + Credit: credit\_risk\_scores\_gold, view v\_credit\_risk\_triage
  + Segments: customer\_rfm\_gold, view v\_segments
* **Governance:** function fin.curated.phi\_mask, masks on ip\_address, customer\_id; tags on transactions\_silver, credit\_risk\_scores\_gold; **Lineage** shown in UI
* **External Location:** db\_s3\_external\_databricks-s3-ingest-e68bd → s3://payer-demo-ur/
* **Job:** Fintech-Lakehouse-ELT (4 SQL tasks)
* **Parquet (optional):** INSERT OVERWRITE DIRECTORY 's3://payer-demo-ur/fin/parquet/.../' USING PARQUET SELECT ...

## **Tell–Show–Tell demo script (90 seconds)**

* **Tell:** “We land once in S3, build Delta Bronze/Silver/Gold under **Unity Catalog**, refresh with **Serverless Jobs**, visualize in **Lakeview**, and export **Parquet** for interop.”
* **Show:**
  + **Delta proof:** DESCRIBE HISTORY fin.curated.transactions\_silver then SELECT \* FROM … VERSION AS OF 0 LIMIT 5;
  + **Fraud tile:** scatter (X=ts, Y=fraud\_score, Size=amount, Color=country) with min\_score param
  + **Credit heatmap:** bureau band × delinq\_12m → avg\_pd
  + **Segments:** bar by segment
  + **Lineage:** open fraud\_scored\_gold → **Upstream/Downstream** counts, column lineage
  + **Governance:** DESCRIBE EXTENDED fin.raw.customers\_bronze (show **MASK**), SHOW TBL TAGS ON TABLE fin.curated.transactions\_silver
* **Tell:** “Same governed tables power ETL/ML/BI; streaming-ready; open formats (Parquet) ensure no lock-in.”

## **Likely follow-ups (one-liners)**

* **Did you copy or just register data?** “CTAS to **managed Delta** → physical Parquet + \_delta\_log under UC; not metadata-only.”

┌────────────────────────────┐

│ Databricks Metastore │

│ (Table name, schema, path) │

└─────────────┬──────────────┘

│

▼

┌────────────────────────────────────────┐

│ Cloud Object Storage (Data Lake) │

│ silver/transactions/ │

│ ├── \*.parquet ← actual data │

│ └── \_delta\_log/ ← transaction logs │

└────────────────────────────────────────┘

## **🔹 1. Two Ways a Table Can Be Created**

| **Mode** | **Example** | **What Happens** | **Data Stored?** |
| --- | --- | --- | --- |
| **Register / External** | CREATE TABLE mytable USING DELTA LOCATION 's3://my-data/' | Just **registers** metadata pointing to existing Delta files | ❌ No copy; metadata-only |
| **CTAS / Managed** | CREATE TABLE mytable AS SELECT \* FROM source | **Runs query → writes new Parquet files** + \_delta\_log | ✅ Yes, new physical data |

## **🔹 2. “CTAS to Managed Delta” = Physical Copy**

Let’s break that phrase down:

### **🏗️ CTAS = Create Table As Select**

* Runs a SELECT query.
* Writes the output into a new table (new files).

### **🧱 Managed Delta**

* Table lives *inside* Databricks’ **Unity Catalog (UC)** storage.
* Databricks **controls the location**, lifecycle, and permissions.

### **📦 Physical Parquet + \_delta\_log**

* The new table physically contains:  
  + Parquet files (the data)
  + \_delta\_log folder (Delta transaction metadata)

So:

CTAS → Managed Delta Table → Actual data written (not just metadata registered)

## **🔹 3. “Not Metadata-Only”**

That’s emphasizing the **difference** from when you use CREATE TABLE ... LOCATION:

* That command simply **registers metadata** about existing files.
* No new data movement, no compute, no file writes.

But CTAS actually **materializes** data — Databricks physically computes your query, writes Parquet, and builds a \_delta\_log.

## **🔹 4. Visual Comparison**

┌────────────────────────────────────────┐

│ CTAS Table │

│ CREATE TABLE silver AS SELECT ... │

├────────────────────────────────────────┤

│ ✅ Writes new Parquet data │

│ ✅ Creates \_delta\_log (Delta table) │

│ ✅ Stored under Unity Catalog (managed) │

└────────────────────────────────────────┘

┌────────────────────────────────────────┐

│ External Registration │

│ CREATE TABLE ext USING DELTA LOCATION… │

├────────────────────────────────────────┤

│ ❌ No data written │

│ ✅ Only metadata registered │

│ ✅ Points to existing files │

└────────────────────────────────────────┘

## **✅ In Summary**

When I said:

“CTAS to managed Delta → physical Parquet + \_delta\_log under UC; not metadata-only.”

I meant:

| **Concept** | **Meaning** |
| --- | --- |
| **CTAS** | You *copy data* — Spark reads source and writes new Delta files |
| **Managed Delta** | Stored under Unity Catalog (Databricks controls the path) |
| **Physical Parquet + \_delta\_log** | Actual data & transaction log stored in UC-managed storage |
| **Not metadata-only** | Unlike external table registration, this involves real data movement |

* **Why serverless?** “Avoided AWS vCPU quota and simplified ops; still fast.”
* **Real-time path?** “Swap the fraud feature step to **Auto Loader + DLT**; keep the same tables and tiles.”
* **How do analysts get safe access?** “UC RBAC + **column masking**; they query the same tables, policies enforce what they can see.”

## **–**

**Open formats: show Parquet folder in S3 + query external table. how can i querty external table?**

###### **ChatGPT said:**

**Perfect — you’re touching on one of the most fundamental cloud data lake skills:  
 💡 *querying open-format Parquet data in S3 without copying it into a warehouse.***

**Let’s go step by step 👇**

## **🗂️ 1. You have a Parquet folder in S3**

**Say you have files like this:**

**s3://my-bucket/transactions/**

**├── part-0000.snappy.parquet**

**├── part-0001.snappy.parquet**

**└── \_SUCCESS**

**These are open-format data files — readable by any engine that supports Parquet (Athena, Spark, Databricks, Presto, DuckDB, etc.).**

## **🧭 2. Create a table definition (external table)**

**To query it, you need to register metadata about those files — i.e., where they are, what columns exist, and their data types.**

**You can do this using Amazon Athena (or Redshift Spectrum, or Databricks SQL external tables).**

**Here’s an example using Athena syntax (Presto/Trino SQL):**

**CREATE EXTERNAL TABLE IF NOT EXISTS transactions (**

**transaction\_id STRING,**

**account\_id STRING,**

**amount DOUBLE,**

**country STRING,**

**ts TIMESTAMP**

**)**

**STORED AS PARQUET**

**LOCATION 's3://my-bucket/transactions/'**

**TBLPROPERTIES ('parquet.compression'='SNAPPY');**

**✅ Key point:  
 This does not move or copy any data. It just registers the schema and tells Athena:**

**“When I run SQL, read Parquet files from that S3 folder.”**

## **🔍 3. Query it directly**

**Now you can query it like a normal SQL table:**

**SELECT country, AVG(amount) AS avg\_amount**

**FROM transactions**

**WHERE ts >= TIMESTAMP '2025-10-01 00:00:00'**

**GROUP BY country**

**ORDER BY avg\_amount DESC;**

**Behind the scenes:**

* **Athena reads metadata (schema, file paths)**
* **Scans only relevant Parquet files in S3**
* **Uses column pruning + predicate pushdown for performance**
* **No copies, no ingestion!**

## **⚙️ 4. Optional — use Glue Data Catalog**

**To persist the schema (so you don’t have to recreate it), you can register it in AWS Glue Catalog:**

**CREATE DATABASE IF NOT EXISTS my\_lake;**

**USE my\_lake;**

**CREATE EXTERNAL TABLE IF NOT EXISTS my\_lake.transactions (...)**

**STORED AS PARQUET**

**LOCATION 's3://my-bucket/transactions/';**

**Now the table is discoverable in:**

* **Athena**
* **Redshift Spectrum**
* **EMR**
* **Databricks (if connected to Glue)**

## **🧠 5. How this fits into “open formats”**

**This approach is exactly what “open formats” mean:**

* **Parquet = open columnar file format**
* **Glue/Athena = open metadata/catalog layer**
* **Query engines (Databricks, Redshift Spectrum, Athena) = interchangeable compute layers**

**No vendor lock-in, no data duplication — just shared S3 files.**

**—**

## **Pitfalls you handled (good to mention)**

* **Namespace drift:** ensured everything is under fin.\*, not workspace.default.
* **Concurrent write error:** use **UPDATE** or build \*\_tmp then **atomic rename** to avoid metadata conflicts.
* **Single Parquet file name:** lakehouse writes folders of parts; if a single named file is required, **coalesce+copy** one part or copy via S3 console/CLI.

# **20-second closing statement**

DB dashboard link [transactions\_silver - Databricks](https://dbc-125af2c1-cdee.cloud.databricks.com/explore/data/fin/curated/transactions_silver?o=2891117215919201&activeTab=history)

setup : page 21 Bronze -> Silver -> Gold Medallion Architecture

“On Databricks we consolidated fraud, credit, and customer analytics on a **governed Lakehouse**. Data lands once in **S3**, becomes **Delta** under **Unity Catalog**, and powers **fraud triage, credit PD, and segmentation** in **Lakeview**, orchestrated by **serverless Jobs**. It’s faster, cheaper, and auditable—and we keep it **open** with Parquet export and a clear path to **streaming** and **model serving** when needed.”

If you want, paste your final architecture slide text and I’ll tighten labels so it exactly mirrors your build.