# Privacy and Governance for Lakehouse Analytics

databricks

# Module Objectives

1. Store sensitive data appropriately to simplify granting access and processing deletes
2. Provide Lakehouse users with appropriate privileges to sensitive data
3. Process deletes to ensure compliance with the right to be forgotten
4. Apply design decisions that minimize costs and latency while making data available for analytics

# Agenda

- Lakehouse and the Query Layer
- Stored Views
- Materialized Gold Tables
- PII & Regulatory Compliance
- Creating a Pseudonymized PII Lookup Tables
- Storing PII Securely
- Managing ACLS for the Enterprise Lakehouse
- Deidentified PII Access
- Propagating Deletes with Delta Change Data Feed
- Deleting at Partition Boundaries

# Lakehouse and the Query Layer

# What is the Query Layer?

- Stores refined datasets for use by data scientists
- Serves results for pre-computed ML models
- Contains enriched, aggregated views for use by analysts
- Star-schemas and data marts for BI queries
- Powers data-driven applications, dashboards, and reports

Also called the serving layer; gold tables exist at this level.

# Tables and Views in the Query Layer

- Gold tables
- Saved views
- Databricks SQL saved queries
- Tables in RDS/NoSQL database

# Gold Tables

- Refined, typically aggregated views of data saved using Delta Lake
- Can be updated with batch or stream processing
- Configured and scheduled as part of ETL workloads
- Results computed on write
- Read is simple deserialization; additional filters can be applied with pushdowns

# Saved Views

- Views can be registered to databases and made available to users using ACLs
- Views are logical queries against source tables
- Logic is executed each time a view is queried
- Views registered against Delta tables will always query the most current valid version of the table

# Databricks SQL Saved Queries

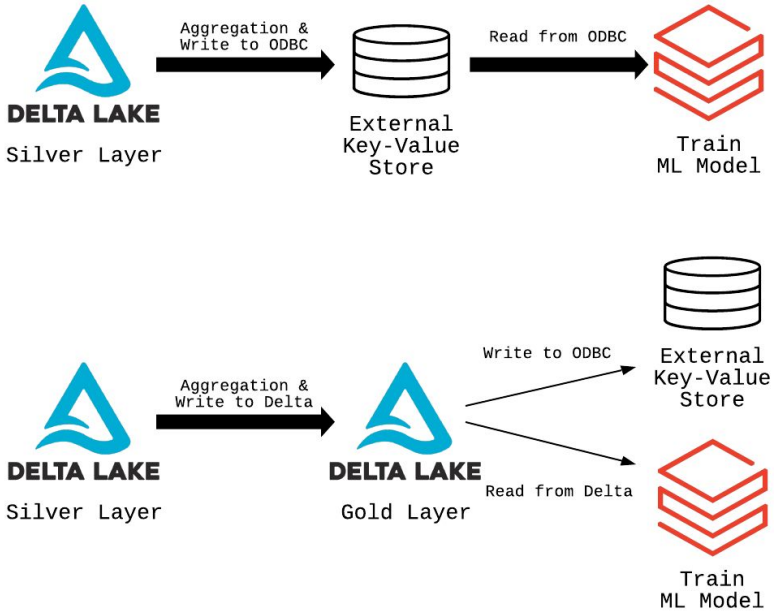
- Similar to saved views in when logic is executed
- Auto-detect changes in upstream Delta tables
- Uses new feature Query Result Cache
- Caching allows reviewing dashboards and downloading CSVs without an active SQL endpoint
- Easy to identify data sources (SQL present in saved query)
- Can be scheduled using Databricks SQL functionality
- Can automatically refresh Databricks SQL dashboards

# Databricks SQL Endpoints

- Clusters optimized for SQL queries
- Serverless option for quick cluster startup and autoscaling
- Photon-enabled for vectorized execution
- Enhanced throughput for exchanging data with external SQL systems
- Optimized connectors for popular BI tools

# Tables in External Systems

- Many downstream applications may require refined data in a different system
  - NoSQL databases
  - RDS
  - Pub/sub messaging
- Must decide where single source of truth lives



# Recommendations

- Use saved views when filtering silver tables

```
CREATE VIEW sales_florida_2020 AS
  SELECT *
  FROM sales
  WHERE state = 'FL' and year = 2020;
```

# Recommendations

- Use Delta tables for common partial aggregates

```
CREATE TABLE store_item_sales AS
  SELECT store_id, item_id, department, date,
    city, state, region, country,
    SUM(quantity) AS items_sold,
    SUM(price) AS revenue
  FROM sales
    INNER JOIN stores ON sales.store_id = stores.store_id
    INNER JOIN items ON sales.item_id = items.item_id
  GROUP BY store_id, item_id, department, date,
    city, state, region, country
```

# Recommendations

- Share Databricks SQL queries and dashboards within teams

```
SELECT date, hour, SUM(quantity * price) hourly_sales
  FROM sales
  WHERE store_id = 42
  AND date > date_sub(current_date(), 14)
  GROUP BY date, hour;
```

# Recommendations

- Create views with column aliases

```
CREATE VIEW sales_marketing AS
  SELECT
    id cust_id,
    date deal_date,
    total sale_amount
  FROM sales;
```

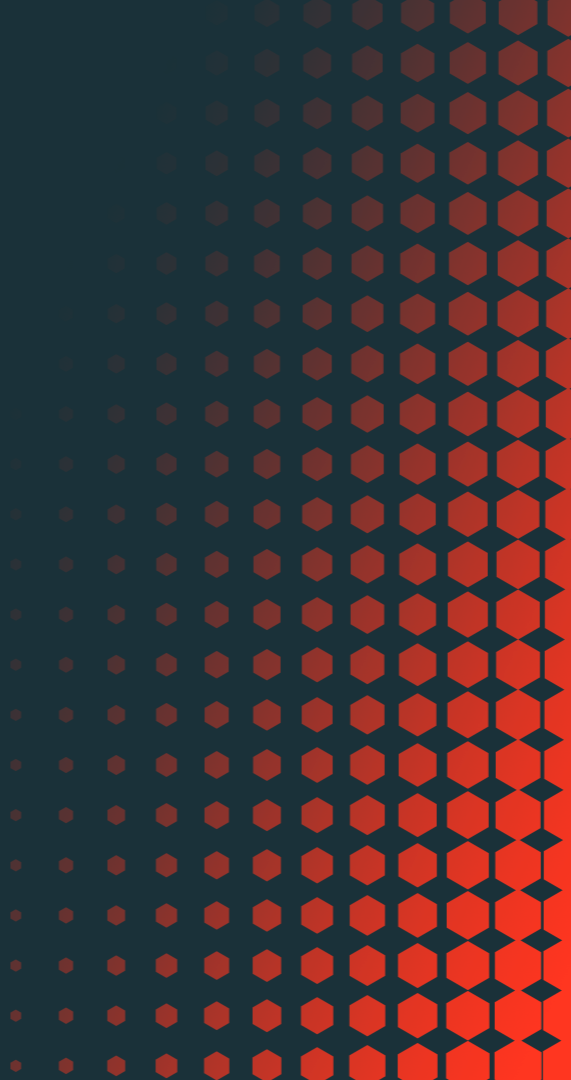
# Recommendations

- Analyze query history to identify new candidate gold tables
  - Admins can access Databricks SQL query history
  - Running analytics against query history can help identify
    - Queries that are long-running
    - Queries that are run regularly
    - Queries that use common datasets
- Transitioning these queries to gold tables and scheduling as engineering jobs may reduce total operating costs
- Query history can also be useful for identifying predicates used most frequently; useful for ZORDER indexing during optimization

# Notebook: Stored Views

# Notebook: Materialized Views

# PII & Regulatory Compliance



databricks

# Regulatory Compliance

- EU = GDPR (General Data Protection Regulation)
- US = CCPA (California Consumer Privacy Act)
- Simplified Compliance Requirements
  - Inform customers what personal information is collected
  - Delete, update, or export personal information as requested
  - Process request in a timely fashion (30 days)

# How Lakehouse Simplifies Compliance

- Reduce copies of your PII
- Find personal information quickly
- Reliably change, delete, or export data
- Use transaction logs for auditing

# Manage Access to PII

- Control access to storage locations with cloud permissions
- Limit human access to raw data
- Pseudonymize records on ingestion
- Use table ACLs to manage user permissions
- Configure dynamic views for data redaction
- Remove identifying details from demographic views

# Pseudonymization

# Pseudonymization

- Switches original data point with pseudonym for later re-identification
- Only authorized users will have access to keys/hash/table for re-identification
- Protects datasets on record level for machine learning
- A pseudonym is still considered to be personal data according to the GDPR

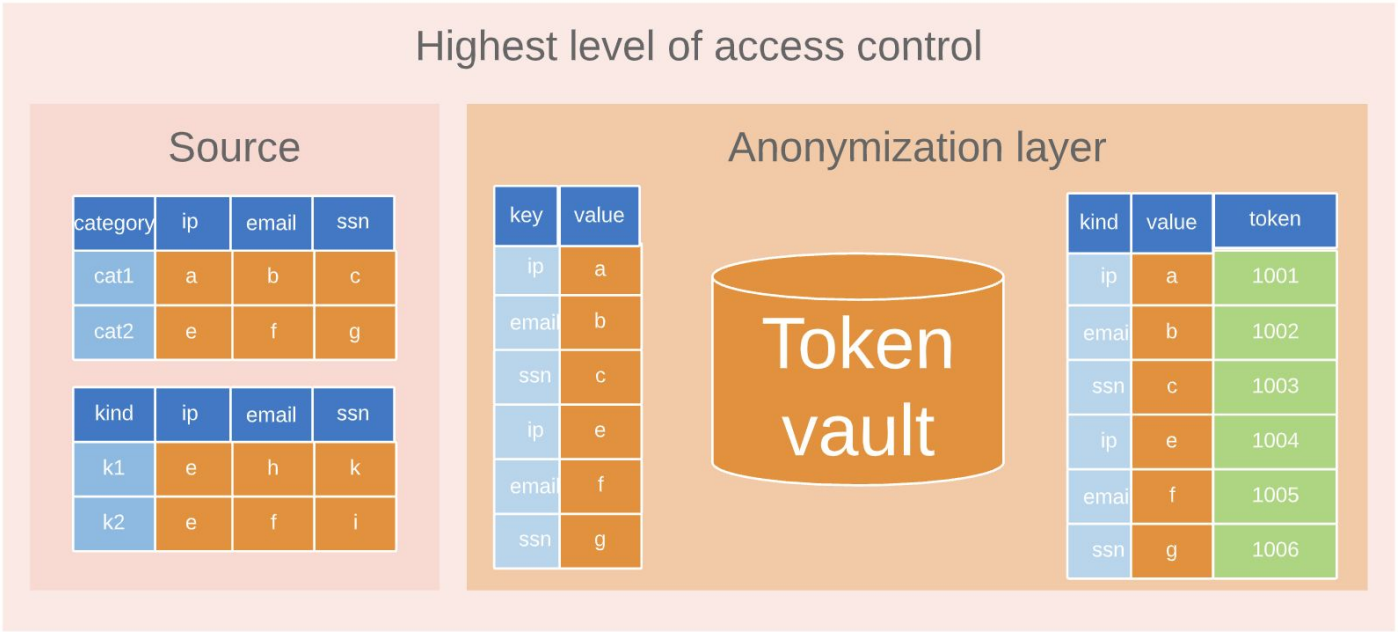
# Hashing

- Apply SHA or other hash to all PII
- Add random string "salt" to values before hashing
- Databricks secrets can be leveraged for obfuscating salt value
- Leads to some increase in data size
- Some operations will be less efficient

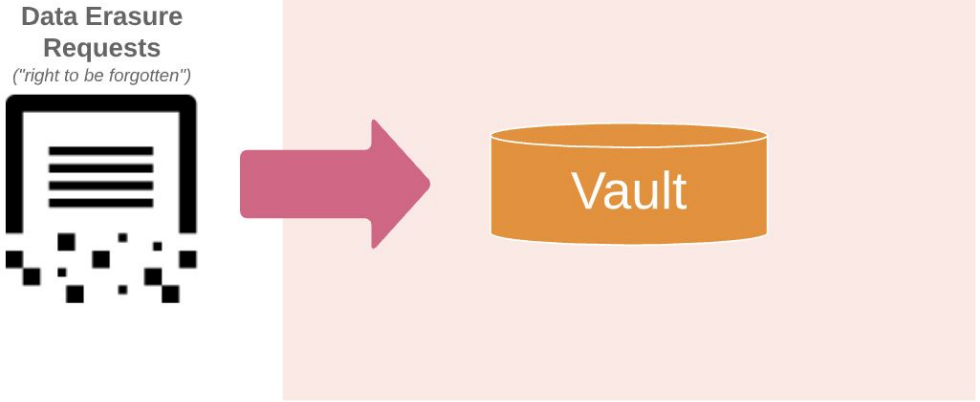
# Tokenization

- Converts all PII to keys
- Values are stored in a secure lookup table
- Slow to write, but fast to read
- De-identified data stored in fewer bytes

# Tokenization



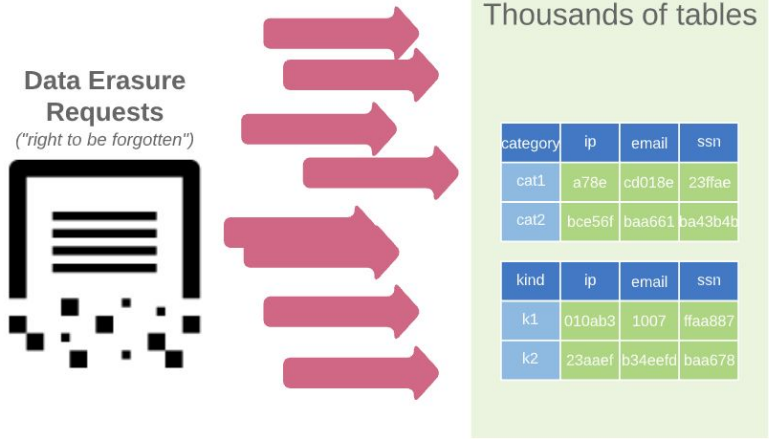
# GDPR with vault




versus

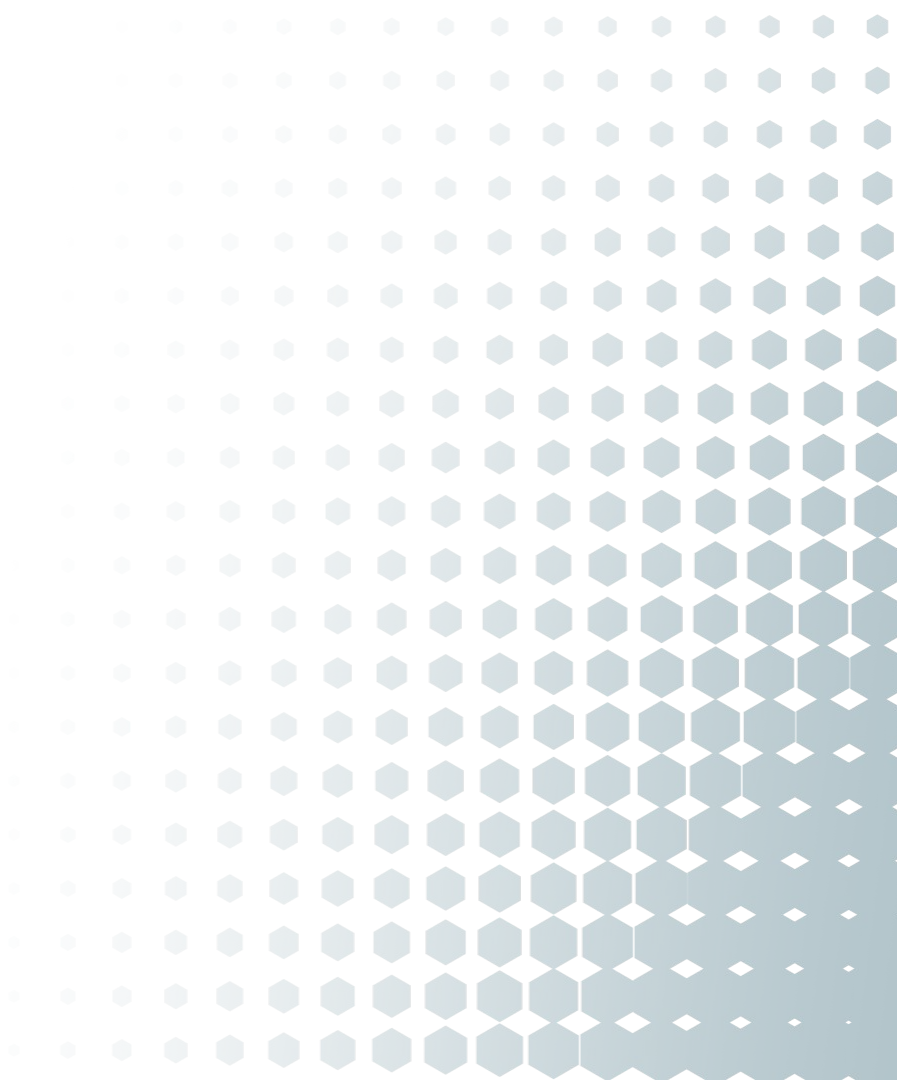
# GDPR without vault

Vault-based systems might be difficult to build, but without it coping with GDPR Data Erasure Requests might be, in fact, more difficult




databricks

# Anonymization



# Anonymization

- Protects entire tables, databases or entire data catalogues mostly for Business Intelligence
- Personal data is irreversibly altered in such a way that a data subject can no longer be identified directly or indirectly
- Usually a combination of more than one technique used in real-world scenarios

# Data Suppression

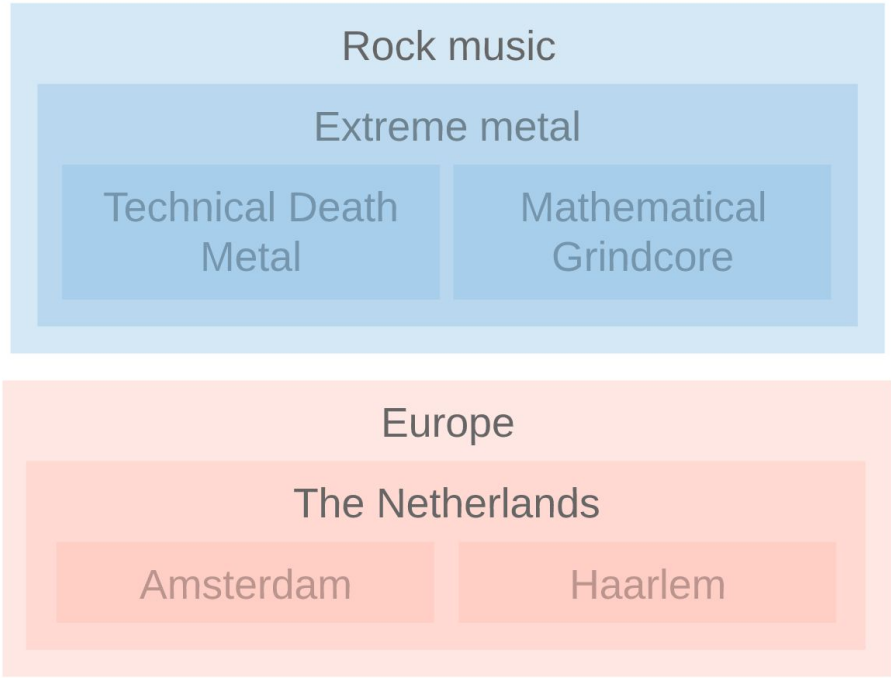
- Exclude columns with PII from views
- Remove rows where demographic groups are too small
- Use dynamic access controls to provide conditional access to full data

# Generalization

- Categorical generalization
- Binning
- Truncating IP addresses
- Rounding

# Categorical Generalization

- Removes precision from data
- Move from specific categories to more general
- Retain level of specificity that still provides insight without revealing identity



# Binning

- Identify meaningful divisions in data and group on boundaries
- Allows access to demographic groups without being able to identify individual PII
- Can use domain expertise to identify groups of interest

# Truncating IP addresses

- Rounding IP address to /24 CIDR
- Replace last byte with 0
- Generalizes IP geolocation to city or neighbourhood level

```
1  import spark._
2  val df = Seq("10.130.176.215", "10.5.56.45",
3      "10.208.126.183", "10.10.14.84",
4      "10.106.62.87", "10.190.48.173",
5      "10.141.100.19", "10.153.248.211",
6      "10.121.140.220", "10.37.240.84").toDF
7    .withColumn("ip_truncated",
8      concat(substring_index('value, ".", 3), lit(".0/24")))
9  display(df)
```

| | value | ip_truncated |
|---|---|---|
| 1 | 10.130.176.215 | 10.130.176.0/24 |
| 2 | 10.5.56.45 | 10.5.56.0/24 |
| 3 | 10.208.126.183 | 10.208.126.0/24 |
| 4 | 10.10.14.84 | 10.10.14.0/24 |
| 5 | 10.106.62.87 | 10.106.62.0/24 |
| 6 | 10.190.48.173 | 10.190.48.0/24 |
| 7 | 10.141.100.19 | 10.141.100.0/24 |

Showing all 10 rows.

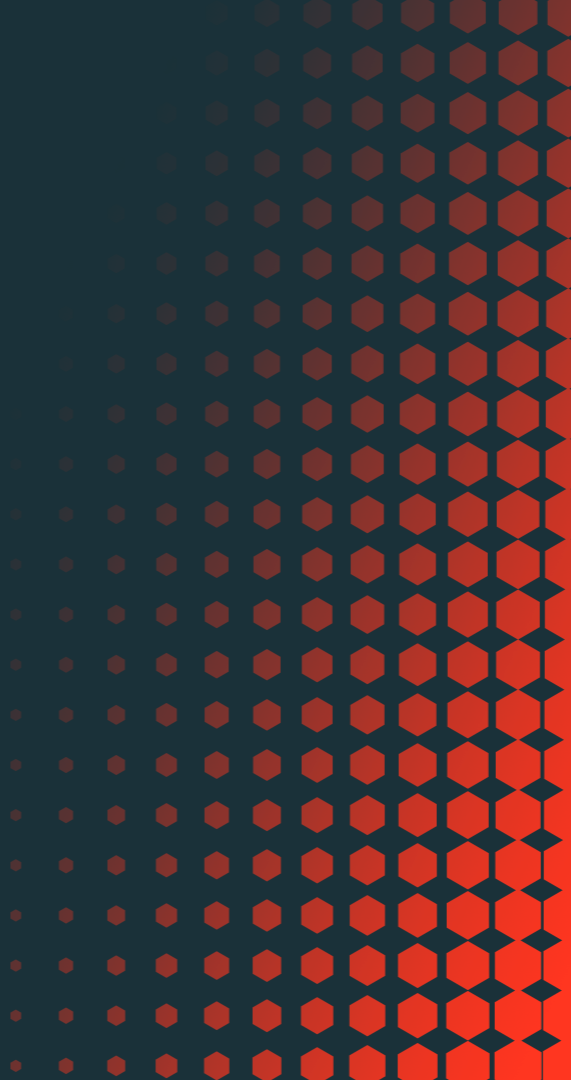
# Rounding

- Apply generalized rounding rules to all number data, based on required precision for analytics
- Provides gross estimates without specific values
- Example:
  - Integers are rounded to multiples of 5
  - Values less than 2.5 are rounded to 0 or omitted from reports
  - Consider suppressing outliers

# Notebook: Creating a Pseudonymized PII Lookup Table

# Notebook: Storing PII Securely

# Managing ACLs for the Enterprise Lakehouse



databricks

# Challenges with Analytics in the Data Lake

- Difficult to provide access on need-to-know basis
- Unclear governance & ownership of data assets
- Data proliferation resulting in possible compliance issues
- Difficult to ensure integrity of data and reporting

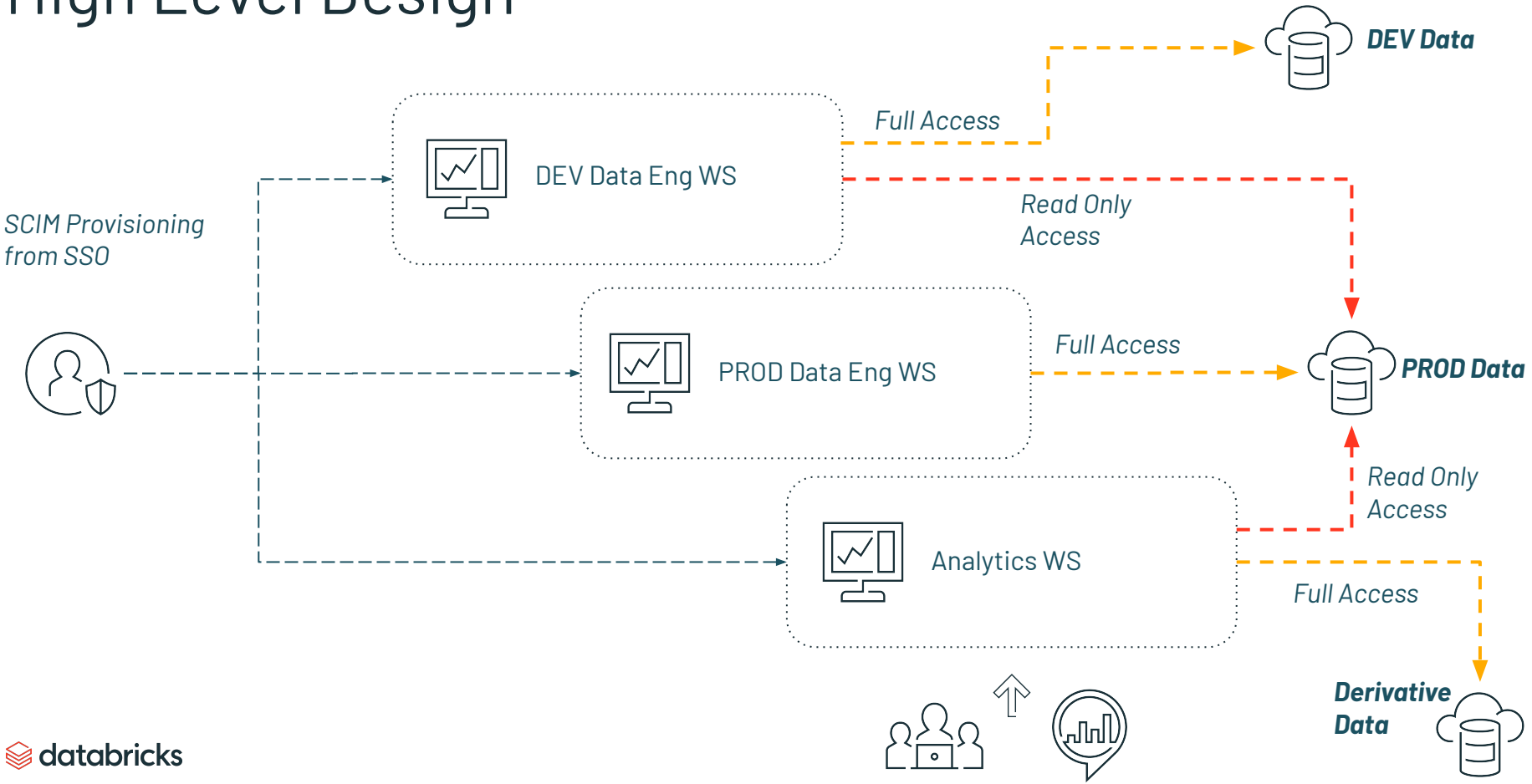
# The Goal

Provide access to valuable data to users across the company in a secure manner.

Ensure users are only able to access the data they're entitled to.

Detect whether any data has been altered or manipulated.

# High Level Design




databricks

# Grant Access to Production Datasets

Assumptions

- End-users need read-only access
- Datasets organized by database

```
GRANT USAGE, SELECT, READ_METADATA ON DATABASE hr TO `HR`;
```

Alternative, grant access on specific tables:

```
GRANT USAGE ON DATABASE hr TO `HR`;
GRANT SELECT, READ_METADATA ON TABLE employees TO `HR`;
GRANT SELECT, READ_METADATA ON TABLE addresses TO `HR`;
```

# Enable Secure Data Sharing

Assumptions

- Teams/depts need a private area to collaborate in
- Datasets must not be shared outside team/dept
- New tables are not automatically shared to other members

```
GRANT USAGE, CREATE ON DATABASE project_data TO `Data Analysts`;
```

Alternatively, members automatically see all new tables:

```
GRANT USAGE, SELECT, READ_METADATA, CREATE ON DATABASE project_data TO `Data Analysts`;
```

# Enable Private User Sandboxes

Assumptions

- Users need a private area to store derivative datasets
- Datasets must not be shared with other users

```
GRANT USAGE, CREATE ON DATABASE user1 TO
`user1@databricks.com`;
```

# Delegate Administration of Access Policies

Assumptions

- Need to delegate management of ACLs to data stewards/owners
- Monitor using SQL Analytics query logs and/or workspace audit logs

```
ALTER DATABASE hr OWNER TO `HR Admins`;
```

# Delegate Database Creation to Trusted Users

Assumptions

- Delegate database creation to trusted users
- Database creators manage ACLs
- Monitor grants using SQL Analytics query logs and/or workspace audit logs

```
GRANT CREATE ON CATALOG TO `Data Admins`;
```

# Give All Users Read-Only Access To All Datasets

Assumptions

- Simple security model where all users have same level of access
- All new datasets are automatically accessible

```
GRANT USAGE, SELECT, READ_METADATA ON CATALOG TO `users`;
```

# Dynamic Views on Databricks

- Need to redact fields based on user's identity
- Do not give access to underlying table, only view
- Uses existing group membership to filter rows or columns

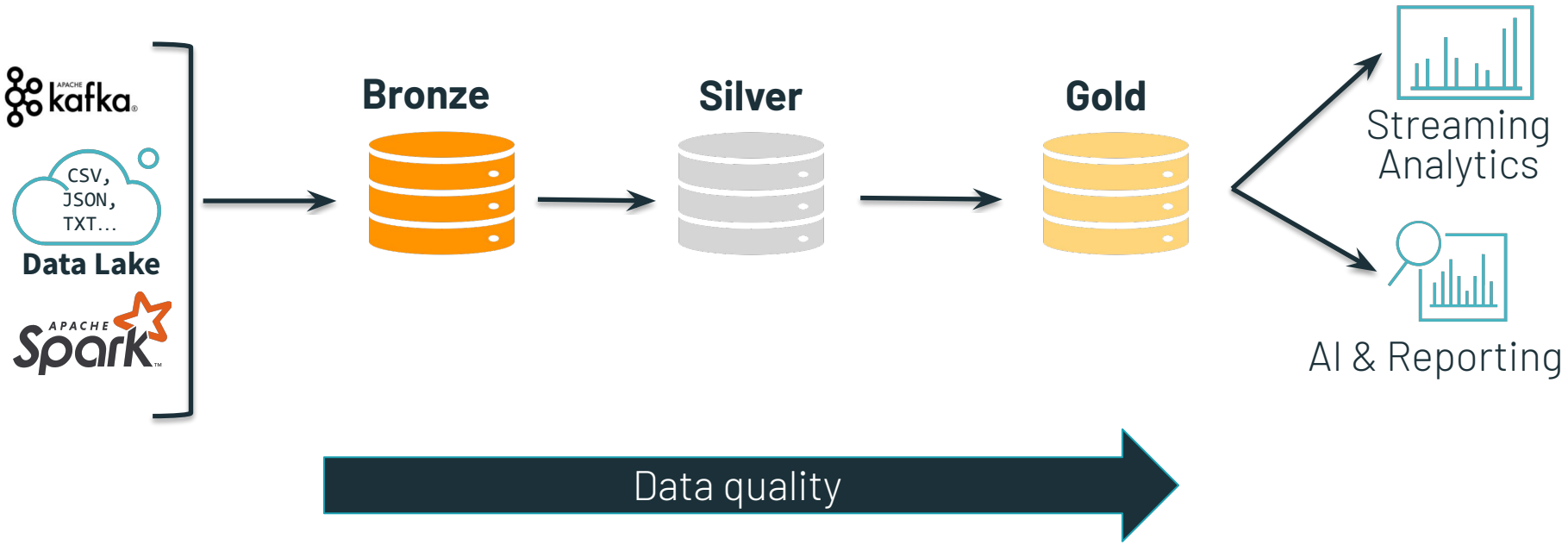
# Rollout Plan

- SCIM provisioning with groups
- Identify & classify initial set of datasets
- Register in new workspace and apply ACLs
- For Fine-Grained
  - Define access policies (e.g. “Sales only see their own accounts…”, “Support technicians can only see accounts within their region”, etc.)
  - Identify missing attributes needed for dynamic views
- Lifecycle policy for derivative datasets
- Define process to “promote” new or derivative datasets
  - Need to classify data & define access policies
  - Legal & compliance review

# Notebook: Deidentified PII Access

# Propagating Changes with Delta Change Data Feed

# Multi-Hop in the Lakehouse



# What is Stream Composability?

- Structured Streaming expects append-only sources
- Delta tables are composable if new streams can be initiated from them

# Operations that break stream composability

- Complete aggregations
- Delete
- UPDATE/MERGE

Data is changed in place, breaking append-only expectations

# Workarounds for Deleting Data

## ignoreDeletes

- Allows deletion of full partitions
- No new data files are written with full partition removal

## ignoreChanges

- Allows stream to be executed against Delta table with upstream changes
- Must implement logic to avoid processing duplicate records
- Subsumes ignoreDeletes

# What Delta Change Data Feed Does for You

## Improve ETL pipelines

Process less data during ETL to increase efficiency of your pipelines

## Unify batch and streaming

Common change format for batch and streaming updates, appends, and deletes

## BI on your data lake

Incrementally update the data supporting your BI tool of choice

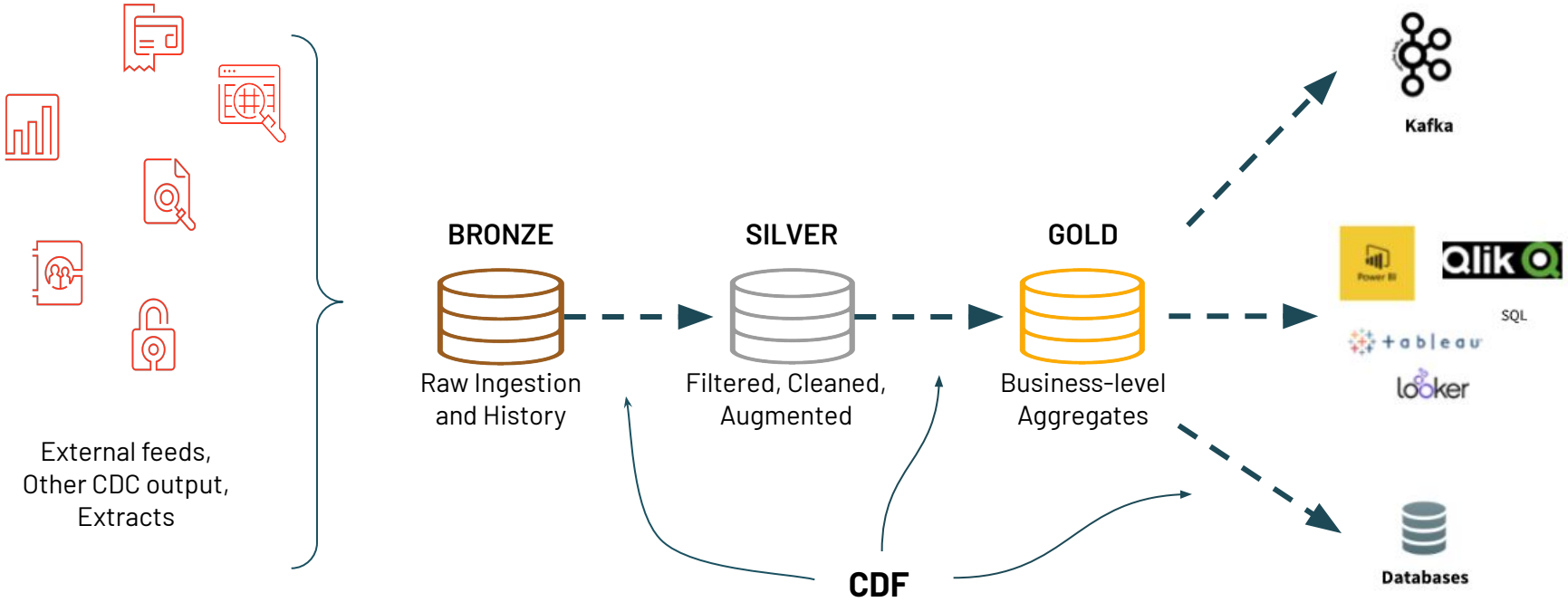
## Meet regulatory needs

Full history available of changes made to the data, including deleted information

**Delta Change Data Feed**

databricks

# Where Delta Change Data Feed Applies



# How Does Delta Change Data Feed Work?

**Original Table (v1)**

| PK | B |
|---|---|
| A1 | B1 |
| A2 | B2 |
| A3 | B3 |

**Change data (Merged as v2)**

| PK | B |
|---|---|
| A2 | Z2 |
| A3 | B3 |
| A4 | B4 |

**Change Data Feed Output**

| PK | B | Change Type | Time | Version |
|---|---|---|---|---|
| A2 | B2 | Preimage | 12:00:00 | 2 |
| A2 | Z2 | Postimage | 12:00:00 | 2 |
| A3 | B3 | Delete | 12:00:00 | 2 |
| A4 | B4 | Insert | 12:00:00 | 2 |

A1 record did not receive an update or delete.
So it will not be output by CDF.

# Notebook: Processing Records from Change Data Feed

# Notebook: Propagating Deletes with Change Data Feed

# Notebook: Deleting at Partition Boundaries

# Module Recap

1. Store sensitive data appropriately to simplify granting access and processing deletes
2. Provide Lakehouse users with appropriate privileges to sensitive data
3. Process deletes to ensure compliance with the right to be forgotten
4. Apply design decisions that minimize costs and latency while making data available for analytics


databricks