databricks

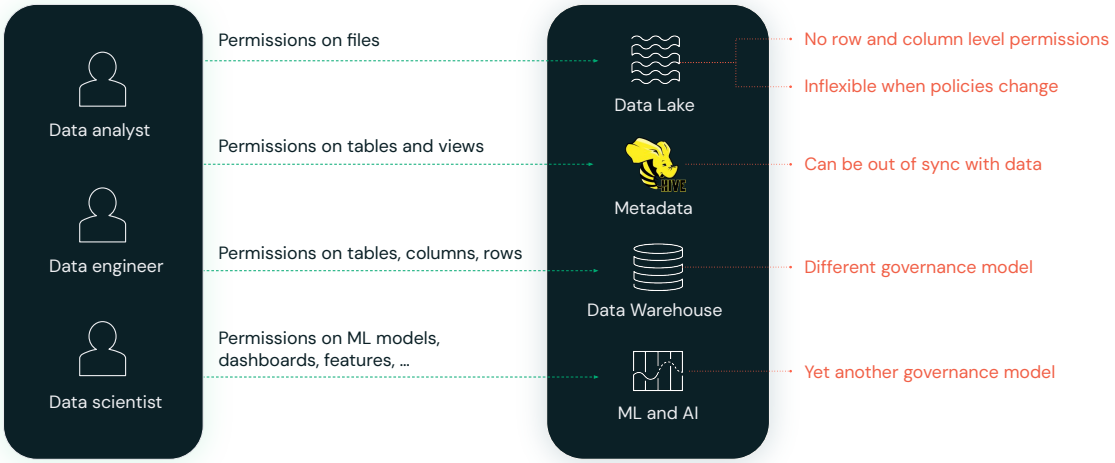
Unity Catalog

LECTURE

# Key Concepts and Components

# Key Concepts and Components

## Governance for Data, Analytics and AI is complex






Talk Track: This is a glimpse into the world before Unity Catalog. Data and AI governance was complex. So the reason that I say the such a complex topic is that we have all of these different personas on the left that we're looking to serve and all of these different types of technology that we're trying to use to serve them and this creates a really complex environment for governance.

In a typical enterprise today, you have lot of data stored in data lakes for example AWS S3.

[Click] To control permissions on data in the data lake, you set permissions on files and directories. [Click] it means you cannot set fine-grained permissions on rows and columns. Because governance controls are at the file level, data teams must carefully structure their data layout to support the desired policies. For example, a team might partition data into different directories by country and give access to each directory to different groups. But what should the team do when governance rules change? If different states inside one country adopt different data regulations, the organization may need to restructure all its directories and files, so it becomes complex to rewrite all the policy changes

[Click] With most data lakes, you not only have files but you also have Metadata for example you might have a hive metastore that keeps track of table definitions and views. So you have to give permission on tables and views. [Click] This can actually go out of sync with underlying data. There is no guarantee that if users have access permissions on files, then they will have permission on the corresponding table or vice

versa. it becomes very confusing to manage permissions.

[Click] then you might have your data warehouse, where the permissions are more fined-grained on tables, columns and views.

[Click] But again it is a different governance model. In typical scenario, you have data movement from your data lakes and data warehouses  And now you've created data silos with data movement across two systems, each with a different governance model.  As a result, your governance method is inconsistent and prone to errors, making it difficult to manage permissions, conduct audits, or discover and share data.

[Click] But data isn't limited to files or tables. You also have assets like dashboards, machine learning models, and notebooks, each with its own permission models and tech stack, making it difficult to manage access permissions for all these assets consistently.

The problem gets bigger when your data assets exist across multiple clouds with different access management solutions.

[Set the stage for the next slide]  What you need is a  unified approach to simplifying governance for data and AI.

---

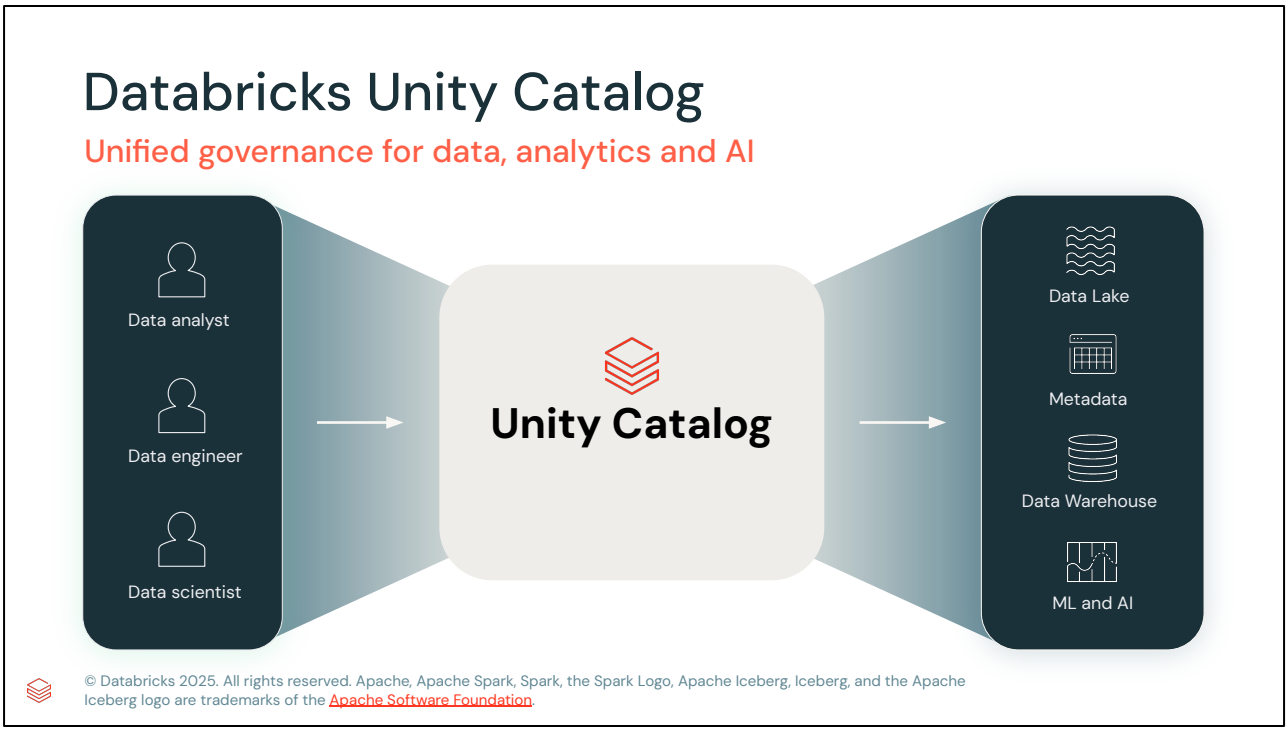
(Additional Notes)

Challenges in the Data Lake

- No fine-grained access controls
    - Cloud storage services only provide access control at the file level through cloud-specific interfaces. This level of access control is coarse and doesn't allow access control at sub-file granularity. For example, if your data governance program requires certain roles to have access to only a subset of the rows or columns within a table, file-level access control does not make this possible. The only way to securely achieve this is to duplicate the subset of data you want to provide access to, which can lead to massive data maintenance issues down the road.
- No common metadata layer
    - Another challenge relates to how governance rules in this environment become tied to the physical layout of files in cloud storage. Suppose you need to reorganize the layout; for instance, maybe you discovered a new way to organize files to improve the system performance. Your entire security model will need to be updated to reflect the updated layout.
- Non-standard cloud-specific governance model
    - On the flip side, suppose your governance rules change. These changes may have an impact on your data that requires restructuring and possibly even rewriting some or all of the data.

- Hard to audit
    - Yet another challenge arises from the different ways enterprises need to access their data.
- No common governance model for different data asset types
    - Suppose you are running analytics jobs that access data in the lake as well as the warehouse, alongside machine learning applications. These systems employ different permission control models that further complicates matters.

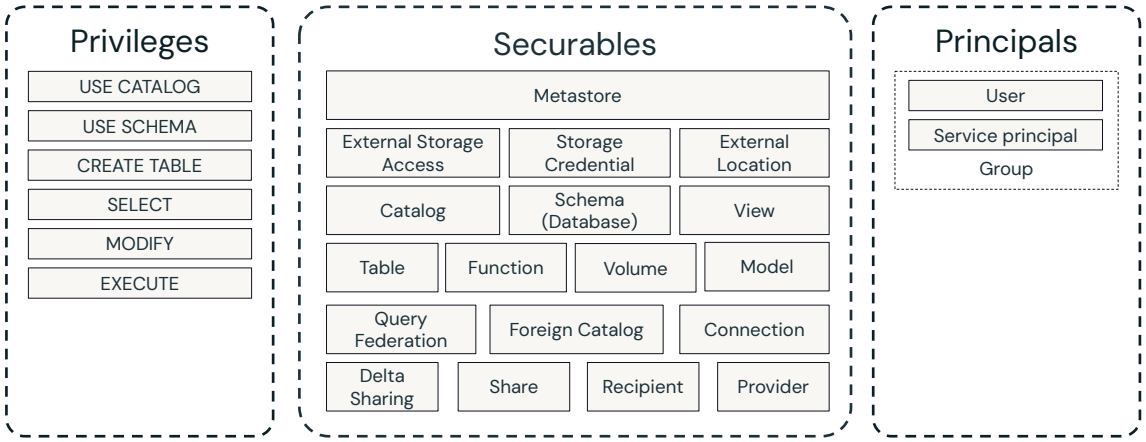
To summarize, the approach to access control provided in a typical data lake environment leads to complicated and inflexible data governance implementations.




To address these challenges, we created unity Catalog, which gives a unified governance layer for all data and AI assets in your lakehouse. Unity Catalog provides a single interface to manage permission or auditing for your data and AI assets.

# Data Security Model

## Access Control Lists (ACLs)






So to recap, Databricks uses access control lists to control who can access what object, and how.

The “how” part of this is define by one of the numerous privilege types available in Databricks, a few of which are exemplified here. Not all privileges apply to all objects though. For example, the MODIFY privilege cannot be applied on a view. That would be meaningless, since views are read-only.

The “what” part refers to a securable object like a table, view or schema. This is the object to which the grant applies.

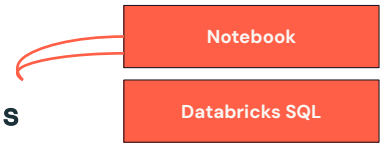
The “who” part of each grant corresponds to a Databricks **principal**. This can be an individual user or service principal, or it can be a group. To reiterate, Databricks recommends using groups when granting permissions as this practice generally lends itself to a more elegant and maintainable governance implementation.

# Data Security Model

## Managing ACLs

1. SQL

```
GRANT SELECT ON TABLE t TO analysts
REVOKE SELECT ON TABLE t FROM analysts
```




2. Catalog explorer
   Data Science & Engineering workspace or DBSQL

3. Programatically
   Databricks CLI, Terraform, and REST APIs



Access can be granted or revoked using GRANT and REVOKE statements in SQL either through a notebook or in DBSQL. Shown is a simple example illustrating the granting of the SELECT privilege on a table named “t” to the analysts group. We’re also showing a statement to revoke such a grant

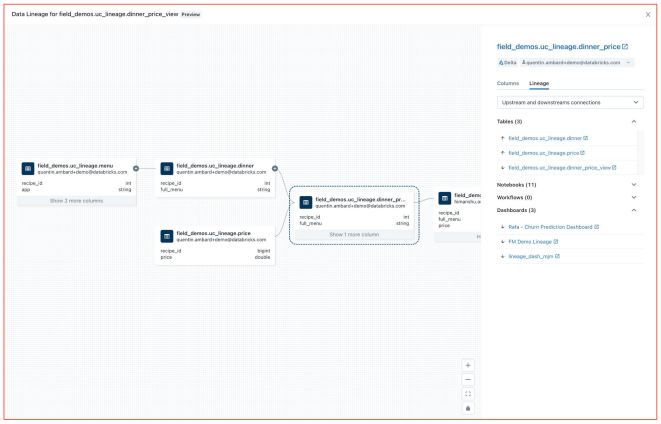
The Data Explorer user interface, available in both Data Science and Engineering Workspace and DBSQL, provides the ability to create, destroy, and control access to data objects interactively.

Finally, like most elements in the Databricks platform, we can manage data access programmatically. At the lowest level, data objects can be managed using REST APIs, though the Databricks CLI provides a more convenient way to manage data objects programmatically. Terraform, an openly available Infrastructure-as-Code tool, uses declarative configuration files to manage data objects and many other aspects of the platform.

# Automated Lineage for all Workloads

## End-to-end visibility into how data flows and consumed in your organization

- Auto-capture runtime data lineage on a Databricks cluster or SQL warehouse
- Leverage common permission model from Unity Catalog
- Lineage across tables, columns, dashboards, Lakeflow Jobs, notebooks, files, external sources, and models



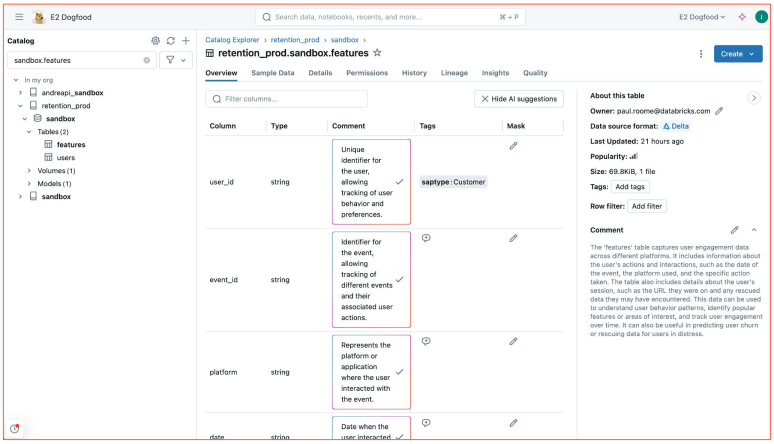



So lineage is pretty exciting! We now have the ability to automatically capture runtime lineage across tables, columns, dashboards, Lakeflow Jobs, notebooks, files, external sources, models, you name it. This is huge when you try to answer questions like, “where did this data come from”, “what downstream data assets will I affect if I make a change in this pipeline”?

There’s not much around best practices here, except, upgrade to UC to take advantage of lineage!

# Tagging + AI generated Documentation

- Auto-generate concise and informative **table and column comments** for Unity Catalog
- Document your backlog of data assets with **missing documentation** in minutes
- **Tag** your data for easier discovery and to facilitate tag-based policies

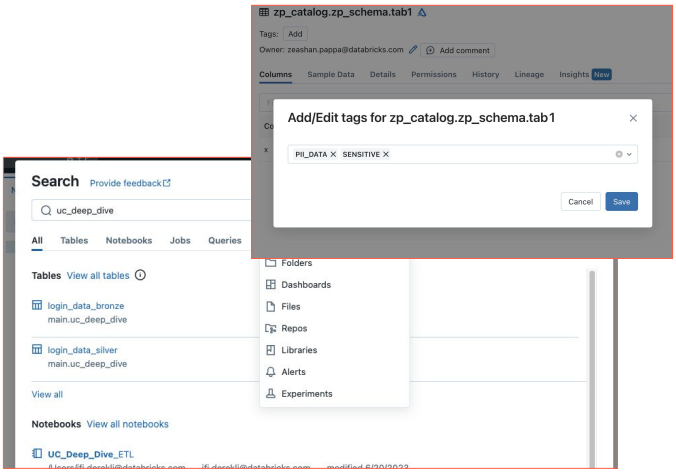


# Built in search and discovery

## Accelerate time to value with low latency data discovery

- Unified UI to search for data assets stored in Unity Catalog
- Leverage common permission model from Unity Catalog
- Tag Column, Table, Schema, Catalog objects in UC
- Search for objects on tags

- *Recommendation: Use comments and Tag your Data Assets on Ingest*






We've also introduced Data Discovery, which is an integrated User Interface to discover assets in Unity Catalog. This is a unified UI across personas and it leverages the common permission model from UC, so if you don't have access to see or read a table, it will not show up in your search results.

The recommendation when it comes to discoverability is to build certain best practices in your organization on the type of tags or comments you use, and make them part of your pipelines and day-to-day data operations.

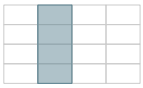
So takeaway here - use UC to take advantage of lineage and discovery and tag your data on ingest, so your lakehouse becomes significantly more user-friendly and productive.

# Fine-grained Access Control

## Common Use Cases

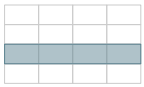
### Limit Access to Columns

Omit column values from output



### Limit Access to Rows

Omit rows from output



### Data Masking

Obscure data

●●●●●●@databricks.com

Can be conditional on a specific user/service principal or group membership through Databricks-provided functions





Fine-grained access control enables access control to rows and columns within a table.

It targets specific users or groups, allowing you to achieve three important use cases:

**Use Case 1:** The first use case is hiding the columns for specific users or groups. When querying the view, targeted users will not see values for the protected columns.

This involves restricting access to column values. Everyone will see that the column is there, but for everyone except specific users or groups, column values will be redacted.

**Use Case 2:** The next use case is similar to the one we just talked about, except now we're flipping it sideways and applying it to rows. In this use case we omit records for specific users or groups. When querying the view, targeted users will only see records that didn't get caught by the filter criteria.

Row filtering omits records entirely from the query output. Which records pass and which ones get dropped depends on the principal formulating the query and the filter criteria.

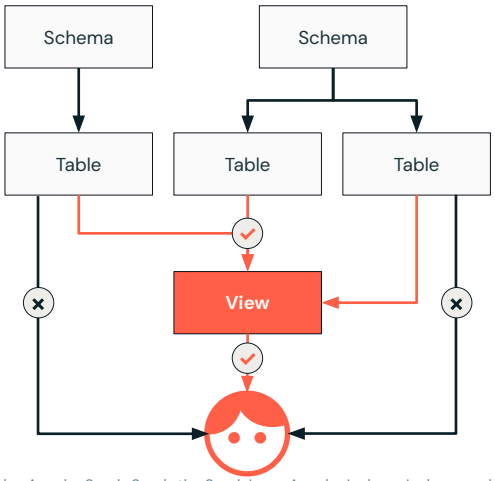
**Use Case 3:** One last use case we'll look at is actually a special case of column restriction, which involves transforming or partially obscuring column values rather than redacting them entirely for specific users or groups. When querying the view, targeted users will only see a transformed version of the data for the protected columns. For example, the domain name of an email address or the last 2 digits of an account number.

This can be a useful technique since it provides additional data that can be used to disambiguate results but does not provide enough information to decide the entire value that we're protecting.

Of course, one could achieve this by creating a secondary table based on the table we want to protect; however, this leads to duplication, increased complexity, and maintenance challenges.

# Fine-grained Access Control

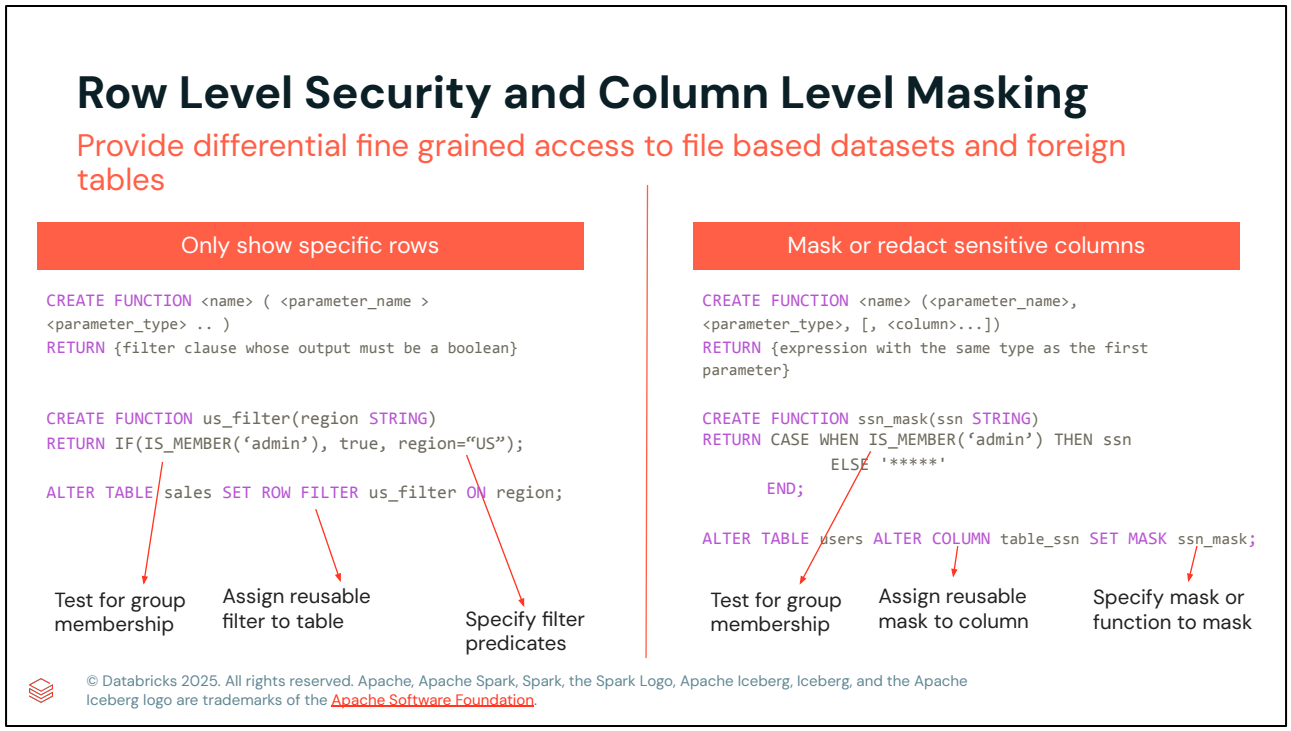
## Dynamic Views





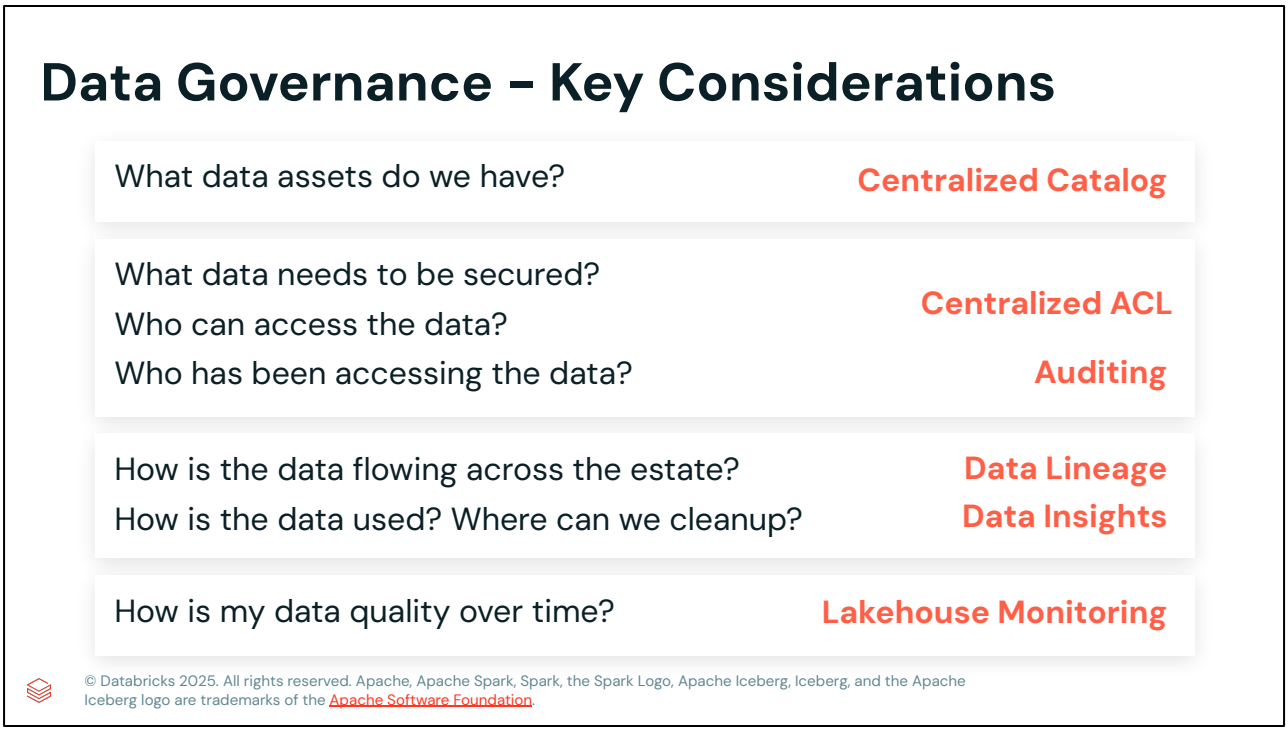

Provided the owner of the view has access to the underlying tables it queries, anyone querying the view does not need direct access to those underlying tables. This means that a view can effectively protect a table, by filtering or transforming rows or columns. Anyone querying the view will only see the data the view is providing - they won't be able to see the data it's protecting.

This, in combination with specialized SQL functions that Databricks provides, enables patterns that make it easy to implement three common use cases.




The second approach is via "Row Filtering and Column Masking."

- Row filtering enables you to apply a filter directly to a table, ensuring that queries return only rows meeting specific criteria. This is applied via a user-defined function attached to your table using the keyword "SET ROW FILTER." Here, on the left, we have a basic example of how this UDF called "us_filter" leverages the "Is_member" function to drive such behavior.
- Column masking allows you to apply a masking function to a table column, replacing each reference of the target column with the results of the masking function. Similarly, this is used via the keyword "SET MASK" to your table. Here on the right, we have another example of a function called "ssn_mask" applied to the user's table, and notice it uses a "case" to return the replaced values.
- Now, be aware that both approaches: Dynamic views, Row filtering, and column masking, can provide the same behavior. However, there are some considerations; for instance, you must create and maintain new views for each case you need to handle in Dynamic Views. Meanwhile, you must create user-defined functions for Row Filtering and Column masks and attach them to your table.




Data Governance is a People-Process-Tools challenge.

Unity Catalog can help simplify the process and tooling aspects of the solution.

Consider a real-world example of Data Governance in action:

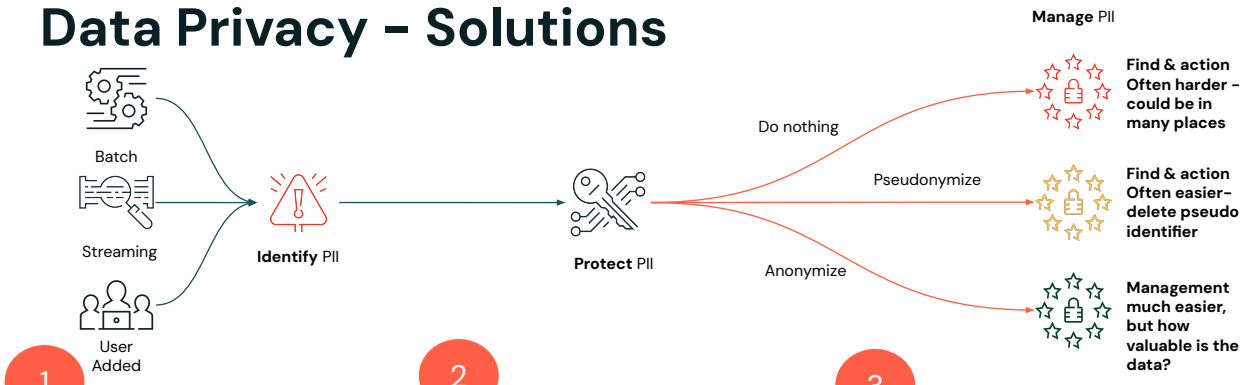
Imagine a large healthcare provider with thousands of patients. They store medical records, billing information, and research data in a complex ecosystem of systems. Without proper data governance:

- **Privacy Breach Risk:** A misconfigured database could expose sensitive patient information, leading to legal action and loss of trust.
- **Inaccurate Treatment Decisions:** Outdated or inconsistent medical records could result in incorrect diagnoses or medication errors.
- **Missed Research Opportunities:** Poor data quality or lack of access could hinder valuable research into new treatments.

With effective data governance, using tools like Databricks and Unity Catalog:

- **Security is Enhanced:** Robust access controls ensure that only authorized personnel can view patient data, minimizing the risk of breaches.
- **Data Accuracy is Improved:** Automated data quality checks and validation processes keep records consistent and reliable, supporting better care.
- **Collaboration is Streamlined:** Researchers can easily discover and access relevant datasets, accelerating discoveries that benefit patients.

# Data Privacy – Solutions




## 1 Identify

- **Data Discovery**
- **Data Classification**
- **Data Mapping**

## 2 Protect

- **Data Security Measures**
- **Data Handling**
- Consent Management
- Privacy by Design

## 3 Manage

- **Data Governance**
- **Compliance Management**
- **Monitoring and Auditing**
- Data Subject Rights Management
- Incident Response
- Privacy Impact Assessment