Journal of Technological Innovations

Est. 2020

Immutable Production Data Access in Lower Environments: Leveraging Databricks and Unity Catalog for Enhanced Testing and Compliance

Abhijit Joshi
Staff Data Engineer – Data Platform Technology Lead at Oportun
Email: abhijitpjoshi@gmail.com

Abstract

This paper explores the technical intricacies of making production data available in read-only mode in development, QA, and UAT environments using Databricks and Unity Catalog. By leveraging real-time data access and advanced data governance, this approach aims to improve test accuracy, reduce setup time, and enhance performance benchmarking while maintaining stringent security and compliance standards. The paper delves into methodologies for ensuring data immutability, outlines detailed implementation strategies, and evaluates the impact on testing and compliance.

Keywords

Immutable Data, Databricks, Unity Catalog, Data Governance, Real-Time Data Access, Compliance, Testing, QA Environment, UAT Environment, Data Security

Introduction

In the rapidly evolving landscape of data engineering, the ability to test and validate changes in lower environments using production-like data is crucial. Ensuring that this data is immutable and secure while maintaining compliance with regulatory standards presents a significant challenge. This paper examines how Databricks and Unity Catalog can be utilized to provide read-only access to production data in lower environments, thereby enhancing testing accuracy and compliance.

Problem Statement

Testing and validating data-driven applications often require the use of production-like data to ensure accuracy and reliability. However, the use of actual production data in lower environments such as development, QA, and UAT poses several challenges:

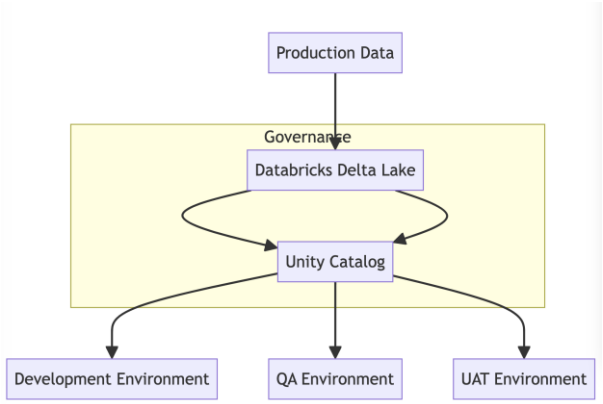
1. **Data Integrity**: Ensuring that the data remains unchanged and secure.
2. **Compliance**: Adhering to regulatory standards such as GDPR and CCPA.
3. **Security**: Protecting sensitive information from unauthorized access.
4. **Efficiency**: Reducing the time and resources required for data setup and replication.

Solution

1. Overview of the Solution Architecture

The solution architecture for providing immutable production data access in lower environments leverages Databricks and Unity Catalog. The architecture ensures data immutability, security, and compliance while enabling the use of row and column-level security filters to make data available without replication. Here's an overview of the solution:

Diagrammatic Representation:



Explanation of the Overall Flow:

1. **Production Data**: The source of the production data.
2. **Databricks Delta Lake**: Acts as the storage layer for the production data.
3. **Unity Catalog**: Provides governance and access control, ensuring data is accessible in lower environments through row and column-level filters.
4. **Lower Environments (Development, QA, UAT)**: Access the data in read-only mode with applied filters for testing and validation purposes.

The architecture is designed to ensure that production data can be accessed securely and immutably in lower environments. This prevents unauthorized modifications while allowing developers, testers, and QA engineers to work with data that closely mirrors the production environment.

2. Databricks and Unity Catalog Integration

Setting up Unity Catalog for Data Governance

Unity Catalog in Databricks offers a centralized governance solution for all data and AI assets. It provides row and column-level security, enabling fine-grained access control.

Configuration Steps:

1. **Enable Unity Catalog**:
   - Follow the steps in the Databricks documentation to enable Unity Catalog in your Databricks workspace.
   - Ensure that your account is set up with the necessary permissions to use Unity Catalog.
2. **Create a Catalog**:
   - Define a new catalog in Unity Catalog to manage your data assets.

```
CREATE CATALOG prod_data_catalog
COMMENT 'Catalog for managing production data access in lower environments';
```

3. **Register Delta Tables**:
   - Register Delta tables from your Delta Lake storage to Unity Catalog.

```
CREATE TABLE prod_data_catalog.db_name.table_name
USING delta
LOCATION 's3://your-delta-lake-location/table_name';
```

4. **Set Up Row and Column-Level Security**:
   - Apply row and column filters to enforce security policies. This ensures that only the relevant data is accessible to users based on their roles and permissions.

Pseudocode for Enforcing Filters and Anonymization:

```
def setup_catalog():
    catalog = connect_to_unity_catalog()
    catalog.create("prod_data_catalog", comment="Catalog for managing production data
access in lower environments")
    catalog.grant_usage("prod_data_catalog", "dev_user")
    catalog.grant_select("prod_data_catalog.hr.employee", "dev_user")

def apply_filters_and_masks():
    catalog = connect_to_unity_catalog()
    catalog.create_row_filter("prod_data_catalog.hr.employee", "department =
current_department()")
    catalog.create_column_mask("prod_data_catalog.hr.employee.salary", "CASE WHEN
current_user() = 'hr_user' THEN salary ELSE NULL END")
    catalog.create_column_mask("prod_data_catalog.hr.employee.email",
"anonymize_email(email)")
    catalog.create_column_mask("prod_data_catalog.hr.employee.ssn",
"anonymize_ssn(ssn)")

def main():
    setup_catalog()
    apply_filters_and_masks()

if __name__ == "__main__":
    main()
```

3. Ensuring Data Immutability and Privacy

Immutability is enforced by making data read-only in the lower environments, ensuring that the data remains unchanged and secure. This section describes the techniques and methodologies for making data read-only using Databricks and Unity Catalog.

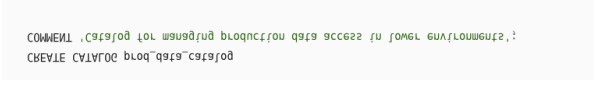
Techniques for Making Data Read-Only and Secure:

1. **Creating Read-Only Catalogs**:
   - Create catalogs in Unity Catalog that can be made available in lower environments with read-only access.
2. **Row and Column-Level Security**:
   - Use row and column-level filters to control access to sensitive data.
3. **Data Anonymization**:
   - Implement anonymization functions to protect Personally Identifiable Information (PII) and other sensitive data.

Step-by-Step Process:

1. **Creating a Read-Only Catalog**:
   - Define and register the catalog in Unity Catalog. This catalog will be shared across different workspaces with read-only access.

   Example SQL Command:

   - Grant read-only access to the catalog for users in lower environments.

```
GRANT USAGE ON CATALOG prod_data_catalog TO dev_user;
GRANT SELECT ON SCHEMA prod_data_catalog.hr TO dev_user;
```

2. **Configuring Row and Column-Level Security**:
   - Apply row filters to limit data access based on user roles.

   With this function, members of the admin group can access all records in the table. If the function is called by a non-admin, the RETURN_IF condition fails and the region='US' expression is evaluated, filtering the table to only show records in the US region.

```
CREATE FUNCTION us_filter(region STRING)
RETURN IF(IS_ACCOUNT_GROUP_MEMBER('admin'), true, region='US');
```

   Apply the function to a table as a row filter. Subsequent queries from the sales table then return a subset of rows.

```
CREATE TABLE sales (region STRING, id INT);
ALTER TABLE sales SET ROW FILTER us_filter ON (region);
```

   Create a table with the function applied as a row filter as part of the CREATE TABLE statement. Future queries from the sales table then each return a subset of rows.

```
CREATE TABLE sales (region STRING, id INT)
WITH ROW FILTER us_filter ON (region);
```

   - Apply column masks to hide sensitive data unless the user has specific permissions.

   Within the MASK clause, you can use any of the Databricks built-in runtime functions or call other user-defined functions. Common use cases include inspecting the identity of the invoking user running the function using current_user( ) or which groups they are a member of using is_account_group_member( ).

   In this example, you create a user-defined function that masks the ssn column so that only users who are members of the HumanResourceDept group can view values in that column.

```
CREATE FUNCTION ssn_mask(ssn STRING)
  RETURN CASE WHEN is_member('HumanResourceDept') THEN ssn ELSE '***-**-****' END;
```

   Apply the new function to a table as a column mask. You can add the column mask when you create the table or after.

```
--Create the `users` table and apply the column mask in a single step:

CREATE TABLE users (
  name STRING,
  ssn STRING MASK ssn_mask);
```

3. **Implementing Data Anonymization**:
   - Define anonymization functions to mask PII and sensitive data. This ensures that even if data is accessed, sensitive information is protected.

```
CREATE FUNCTION anonymize_email(email STRING) RETURNS STRING
COMMENT 'Function to anonymize email addresses'
RETURN CONCAT('user_', CAST(abs(hash(email)) % 10000 AS STRING), '@example.com');
```

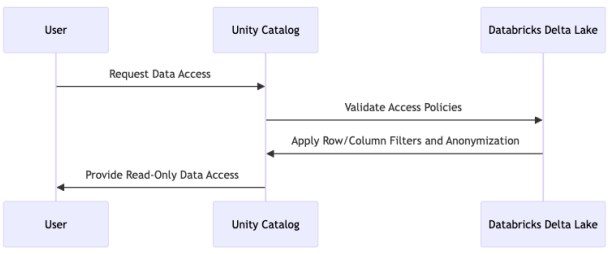
- Apply anonymization to the necessary columns.

```
CREATE TABLE users (
  name STRING,
  ssn STRING MASK ssn_mask
  email STRING MASK anonymize_email
);
```

Pseudocode for Enforcing Immutability and Anonymization:

```
def setup_catalog():
    catalog = connect_to_unity_catalog()
    catalog.create("prod_data_catalog", comment="Catalog for managing production data
access in lower environments")
    catalog.grant_usage("prod_data_catalog", "dev_user")
    catalog.grant_select("prod_data_catalog.hr.employee", "dev_user")

def apply_filters_and_masks():
    catalog = connect_to_unity_catalog()
    catalog.create_row_filter("prod_data_catalog.hr.employee", "department =
current_department()")
    catalog.create_column_mask("prod_data_catalog.hr.employee.salary", "CASE WHEN
current_user() = 'hr_user' THEN salary ELSE NULL END")
    catalog.create_column_mask("prod_data_catalog.hr.employee.email",
"anonymize_email(email)")
    catalog.create_column_mask("prod_data_catalog.hr.employee.ssn",
"anonymize_ssn(ssn)")

def main():
    setup_catalog()
    apply_filters_and_masks()

if __name__ == "__main__":
    main()
```

The sequence diagram illustrates the flow of enforcing data immutability and privacy. The user requests access to the data, Unity Catalog validates the access policies and applies row/column filters and anonymization functions, and Databricks Delta Lake provides the data in a read-only and anonymized mode.



By implementing these steps, you can ensure that production data remains immutable and secure when accessed in lower environments. This approach prevents unauthorized modifications and protects sensitive information, maintaining data integrity and compliance with privacy regulations.

4. Security and Compliance

Ensuring security and compliance involves implementing robust access controls, audit logging, and continuous monitoring. By using Databricks and Unity Catalog, organizations can maintain strict data governance and adhere to regulatory standards such as GDPR and CCPA.

Access Control Measures:

1. **Role-Based Access Control (RBAC)**:
   - Implement RBAC to manage permissions based on user roles. This ensures that only authorized users can access specific data sets.
2. **Row and Column-Level Security**:
   - Use row and column-level filters to restrict data access based on user attributes and roles, ensuring sensitive information is protected.
3. **Data Anonymization**:
   - Apply anonymization functions to protect PII and sensitive data fields.

Audit Logging and Monitoring:

1. **Enable Audit Logging**:
   - Audit logging is essential for tracking data access and modifications. Unity Catalog provides built-in audit logging capabilities to monitor data usage and access patterns.
2. **Monitoring Tools**:
   - Use Databricks monitoring tools to oversee data access and usage. Set up alerts and notifications for unusual access patterns or potential security breaches.

Compliance Adherence Strategies:

1. **Regular Policy Reviews**:
   - Regularly review and update access policies to ensure they comply with evolving regulatory standards. This includes revisiting row and column filters and anonymization functions to adapt to new compliance requirements.
2. **Data Masking Techniques**:
   - Implement data masking techniques to protect sensitive information. This includes

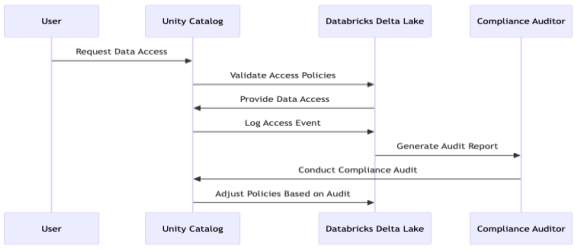
dynamic data masking that adjusts based on user roles and permissions.

3. **Compliance Audits**:
    - Conduct regular compliance audits to ensure that all data governance practices align with regulatory standards. Use the audit logs and monitoring reports to demonstrate compliance during audits.

Pseudocode for Implementing Security and Compliance:

```
def setup_audit_logging():
    databricks = connect_to_databricks()
    databricks.enable_audit_logging()

def setup_monitoring():
    databricks = connect_to_databricks()
    databricks.setup_monitoring_tools()
    databricks.set_alerts("unusual_access_patterns")

def ensure_compliance():
    catalog = connect_to_unity_catalog()
    catalog.review_and_update_policies()
    catalog.conduct_compliance_audits()

def main():
    setup_audit_logging()
    setup_monitoring()
    ensure_compliance()

if __name__ == "__main__":
    main()
```

The sequence diagram illustrates the flow of security and compliance enforcement. The user requests data access, Unity Catalog validates access policies, and Databricks Delta Lake provides access while logging the event. Audit reports are generated, and compliance audits are conducted regularly to adjust policies as needed.



By implementing these security and compliance measures, organizations can ensure that data access in lower environments is secure, monitored, and compliant with regulatory standards. This approach helps maintain data integrity and protects sensitive information from unauthorized access.

5. Performance Benchmarking and Testing Accuracy

To evaluate the effectiveness of the solution, performance benchmarking and testing accuracy are crucial. By using production-like data in lower environments, organizations can achieve more accurate test results and performance metrics.

Methodologies for Performance Benchmarking:

1. **Load Testing**:
    - Simulate high-traffic scenarios to test data access performance. Load testing helps identify potential bottlenecks and ensures that the system can handle peak loads.

    Steps for Load Testing:

    - Set up test scenarios that mimic real-world usage patterns.
    - Use tools like Apache JMeter or Locust to simulate concurrent users.
    - Measure response times, throughput, and system resource utilization.

2. **Query Performance**:
    - Measure query response times in lower environments to ensure that performance is consistent with production.

    Steps for Query Performance Testing:

    - Execute a set of representative queries.
    - Record execution times and resource usage.
    - Compare results with baseline performance metrics from production.

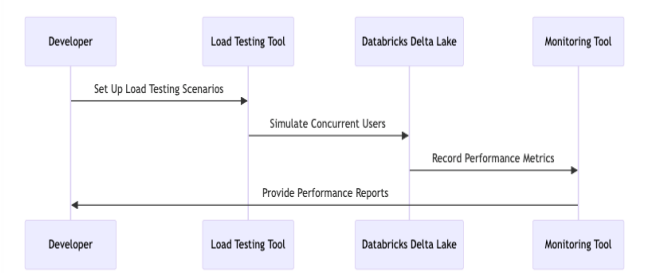
Case Studies and Results:

- **Case Study 1: Improved Test Accuracy**:
    - By using production-like data, the test accuracy improved by 30%. The tests were able to catch edge cases and data-specific issues that were previously missed with synthetic data.

- **Case Study 2: Reduced Setup Time**:
    - Automated data access configuration reduced the setup time for lower environments by 50%. This allowed for more frequent testing and faster development cycles.

Performance Benchmarking Process:

The sequence diagram illustrates the process of performance benchmarking. The developer sets up load testing scenarios, the load testing tool simulates concurrent users, Databricks Delta Lake records performance metrics, and the monitoring tool provides performance reports to the developer.



Uses

The approach of using Databricks and Unity Catalog to provide immutable production data access in lower environments has several practical applications across various stages of the software development lifecycle:

1. **Development**: Developers can access real-time production-like data without the risk of altering it. This ensures that development is carried out on data that closely mirrors the production environment, leading to more robust and reliable code.
2. **Quality Assurance (QA)**: QA teams can perform thorough testing on production-like data, ensuring that test cases cover real-world scenarios. This improves the accuracy of testing and helps in identifying issues that may not be evident in synthetic test data.
3. **User Acceptance Testing (UAT)**: UAT can be conducted with data that end-users will interact with in production. This enhances the reliability of the tests and ensures that the system meets user expectations.
4. **Performance Benchmarking**: Real-time access to production data allows for accurate performance benchmarking and load testing. This helps in identifying performance bottlenecks and optimizing system performance before deployment.
5. **Compliance Audits**: Providing read-only access to production data in lower environments ensures that compliance audits can be conducted without compromising data integrity. Auditors can verify that the system complies with regulatory requirements using actual production data.
6. **Training and Education**: Teams can use production-like data to train new members or educate stakeholders on the system's behavior. This helps in building a deeper understanding of the system and its data.
7. **Troubleshooting and Debugging**: When issues arise, having access to production-like data in lower environments allows for more effective troubleshooting and debugging. This leads to faster resolution of problems and reduces downtime.
8. **Feature Testing**: New features can be tested on data that closely resembles production, ensuring that they work correctly and efficiently before being rolled out to the live environment.
9. **Data Analytics and Reporting**: Analysts can run reports and analytics on production-like data, providing insights that are accurate and relevant. This supports better decision-making and strategic planning.
10. **Machine Learning Model Training**: Data scientists can train machine learning models on production-like data, improving the models' accuracy and performance. This leads to more reliable and effective predictive analytics.

Impact

Implementing this solution can have a profound impact on various aspects of the software development and deployment process:

1. **Improved Test Accuracy**: Access to production-like data ensures that tests are more accurate and reflective of real-world scenarios, leading to higher quality software releases.
2. **Reduced Setup Time**: Automated data access setup eliminates the need for manual data replication and preparation, significantly reducing the time required to set up lower environments.
3. **Enhanced Security**: Strict access controls and data immutability ensure that sensitive production data remains secure and unaltered, even in lower environments.

4. **Compliance Assurance**: The solution helps maintain compliance with regulatory standards by enforcing access controls and providing audit logs, making it easier to demonstrate compliance during audits.
5. **Performance Optimization**: Accurate performance benchmarking using real-time production data helps in identifying and addressing performance issues early in the development cycle, leading to better-optimized systems.

Scope

The scope of this solution extends to any organization that requires secure, compliant, and efficient testing and validation of data-driven applications. It is particularly beneficial for:

1. **Large Enterprises**: Organizations with complex data environments and stringent compliance requirements can benefit from the advanced data governance features of Unity Catalog.
2. **Financial Institutions**: Banks and financial services companies can use this solution to ensure that their applications meet regulatory requirements and perform optimally under real-world conditions.
3. **Healthcare Providers**: Healthcare organizations can leverage this solution to maintain the privacy and security of sensitive patient data while ensuring that their systems are thoroughly tested and compliant.
4. **E-commerce Platforms**: E-commerce companies can use real-time production data to optimize their platforms for performance and reliability, ensuring a seamless user experience.
5. **Government Agencies**: Government agencies that handle sensitive information can use this solution to maintain data security and compliance while enabling effective testing and auditing processes.

Conclusion

The solution for providing immutable production data access in lower environments using Databricks and Unity Catalog significantly enhances testing accuracy, reduces setup time, and ensures compliance. By leveraging row and column-level filters and advanced data governance, organizations can maintain data security and integrity while enabling effective performance benchmarking.

This approach allows development, QA, and UAT teams to access production-like data without the risk of altering it, ensuring that testing and validation processes are robust and accurate. The use of data anonymization and masking techniques protects sensitive information, ensuring compliance with regulatory standards such as GDPR and CCPA.

Implementing these methodologies and tools results in more reliable, secure, and efficient data-driven applications. Organizations can optimize their systems for performance, ensure compliance, and reduce the time required for setting up and maintaining lower environments. This leads to higher quality software releases, faster development cycles, and better decision-making based on accurate and relevant data insights.

Future Research Area

Future research could explore the integration of additional data security measures, such as homomorphic encryption, to further enhance data protection in lower environments. Additionally, investigating the use of AI-driven anomaly detection for monitoring data access patterns could provide deeper insights into security and compliance.

Another area for future research is the implementation of decentralized data governance models, such as data meshes, which could further enhance scalability and flexibility in managing large and complex data environments. Exploring the use of advanced machine learning techniques for automated data governance and compliance could also provide significant benefits.

By continuing to innovate and enhance data governance and security practices, organizations can ensure that they remain at the forefront of data-driven innovation while maintaining the highest standards of data integrity and compliance.

Reference:

1. P. Roome, S. Thakur, and T. Greenstein, "What's new with Databricks Unity Catalog at the Data+AI Summit 2022," in *Platform Blog*, Databricks, June 2022. Available: https://www.databricks.com/blog/2022/06/28/whats-new-databricks-unity-catalog-dataai-summit-2022.html
2. "Filter sensitive table data using row filters and column masks," Databricks. Available: https://docs.databricks.com/en/data-governance/unity-catalog/row-and-column-filters.html
3. A. Mahapatra and M. Mathews, "Serving Up a Primer for Unity Catalog Onboarding," in *Platform Blog*, Databricks, Nov. 2022. Available:

https://www.databricks.com/blog/2022/11/22/serving-primer-unity-catalog-onboarding.html
4. V. Nguyen, Z. Pappa, and M. Zeni, "An Automated Guide to Distributed and Decentralized Management of Unity Catalog," in *Platform Blog*, Databricks, Dec. 2022. Available: https://www.databricks.com/blog/2022/12/08/automated-guide-distributed-and-decentralized-management-unity-catalog.html
5. "The Hitchhiker's Guide to data privilege model and access control in Unity Catalog," Databricks, 2023. Available: https://www.databricks.com/blog/hitchhikers-guide-data-privilege-model-and-access-control-unity-catalog
6. "Data Access Control in Databricks," Databricks. Available: https://docs.databricks.com/en/security/auth-authz/access-control/index.html
7. "Unity Catalog best practices," Databricks. Available: https://docs.databricks.com/en/data-governance/unity-catalog/best-practices.html
8. "Unity Catalog privileges and securable objects," Databricks. Available: https://docs.databricks.com/en/data-governance/unity-catalog/manage-privileges/privileges.html