**Epic Title**

Automated Daily Email Summary for DQ4QD Rules, Workflows, and Pipeline Execution

**Objective**

To develop an automated system within the DQ4QD (Data Quality for Quality Data) framework that generates and sends a comprehensive daily email report to requested clients, summarizing the execution status and details of all rules, workflows, and pipelines by the end of each day. The system aims to enhance transparency, improve client communication, and ensure stakeholders have timely insights into data quality processes.

**Description**

The DQ4QD framework ensures high-quality data through defined rules, workflows, and pipeline executions. Currently, clients lack a consolidated, automated report summarizing these activities. This epic focuses on creating a feature to automate the generation and delivery of a daily email report. The report will include:

* Summary of executed rules (e.g., rule ID, description, pass/fail status, error details).
* Workflow execution details (e.g., workflow name, start/end time, status).
* Pipeline execution overview (e.g., pipeline ID, data volume processed, success/failure metrics).
* Key metrics and insights (e.g., total rules executed, success rate, critical failures).  
  The system will allow clients to opt-in for the report, support customizable report formats, and ensure secure delivery via email. The solution will integrate with the existing DQ4QD framework, leveraging its logging and monitoring capabilities to extract relevant data.

**Acceptance Criteria**

1. **Report Generation**:
   * The system automatically generates a daily report by end of day (configurable time, e.g., 11:59 PM).
   * The report includes a summary of all rules, workflows, and pipeline executions within the DQ4QD framework for the given day.
   * The report contains:
     + Rule details: Rule ID, name, description, execution status (pass/fail), and any error messages.
     + Workflow details: Workflow ID, name, start/end time, status (completed/failed), and duration.
     + Pipeline details: Pipeline ID, name, data volume processed, status, and critical metrics (e.g., records processed, errors encountered).
     + Summary metrics: Total rules executed, success/failure rates, and highlighted critical issues.
   * The report format is clear, structured, and available in both HTML and PDF formats.
2. **Client Opt-In Mechanism**:
   * Clients can opt-in or opt-out of receiving the daily report via a secure web interface or configuration file.
   * The system maintains a client subscription list, ensuring only opted-in clients receive the report.
3. **Email Delivery**:
   * Reports are sent securely via email using a configured SMTP server.
   * Emails include a subject line with the date and a clear title (e.g., "DQ4QD Daily Summary - YYYY-MM-DD").
   * The system supports encrypted email delivery (e.g., TLS/SSL) to ensure data security.
4. **Customization**:
   * Clients can select specific rules, workflows, or pipelines to include in their report via a configuration interface.
   * The report template supports customization (e.g., branding, specific metrics) based on client preferences.
5. **Error Handling and Reliability**:
   * The system logs any failures in report generation or email delivery and notifies administrators.
   * The system retries failed email deliveries at least twice before marking them as failed.
   * Reports are archived for at least 30 days for auditing or resending purposes.
6. **Performance**:
   * Report generation completes within 10 minutes for a dataset with up to 10,000 rules, 1,000 workflows, and 100 pipelines.
   * The system handles multiple client reports concurrently without performance degradation.
7. **Testing and Validation**:
   * The system passes unit and integration tests covering report generation, data accuracy, and email delivery.
   * A sample report is validated by stakeholders to ensure clarity and completeness.

**Design Details**

**System Architecture**

* **Data Source**: The system integrates with the DQ4QD framework’s existing logging and monitoring database (assumed to be a relational database like PostgreSQL or a similar system).
* **Report Generator**: A Python-based module queries the DQ4QD database to extract rule, workflow, and pipeline execution data for the day.
  + Uses SQL queries or ORM (e.g., SQLAlchemy) to fetch data.
  + Aggregates metrics (e.g., success/failure rates, total records processed).
* **Template Engine**: Uses Jinja2 for rendering report templates in HTML and LaTeX for PDF output.
  + HTML reports are styled with a responsive CSS framework (e.g., Bootstrap).
  + PDF reports are generated using a LaTeX engine (PDFLaTeX) for professional formatting.
* **Email Service**: Integrates with an SMTP server (e.g., Postfix or a cloud service like AWS SES) for secure email delivery.
  + Supports TLS/SSL for secure transmission.
  + Includes retry logic for failed deliveries.
* **Client Configuration**: A web interface (built with Flask or Django) or configuration file allows clients to opt-in/out and customize report preferences.
* **Scheduling**: A cron job or task scheduler (e.g., Apache Airflow) triggers report generation daily at a configurable time.

**Technology Stack**

* **Backend**: Python 3.9+ for report generation and data processing.
* **Database**: Integrates with existing DQ4QD database (e.g., PostgreSQL).
* **Template Rendering**: Jinja2 for HTML, LaTeX for PDF.
* **Email Delivery**: SMTP client library (e.g., smtplib in Python or AWS SES API).
* **Web Interface**: Flask/Django for client configuration (optional).
* **Scheduler**: Cron or Apache Airflow for daily execution.
* **Storage**: Reports archived in a secure file system or cloud storage (e.g., AWS S3) for 30 days.

**Data Flow**

1. **Data Extraction**: At the scheduled time, the system queries the DQ4QD database for the day’s execution data.
2. **Report Generation**:
   * Data is processed to compute metrics (e.g., success rates, error counts).
   * Templates (HTML/LaTeX) are populated with data using Jinja2.
   * PDF reports are compiled using PDFLaTeX.
3. **Client Selection**: The system retrieves the list of opted-in clients and their preferences.
4. **Email Delivery**: Reports are attached (HTML inline, PDF as attachment) and sent via the SMTP server.
5. **Archiving**: Reports are saved to a secure storage location with metadata (e.g., client ID, date).

**Security Considerations**

* **Data Access**: Database queries use least-privilege credentials.
* **Email Security**: Emails are sent over TLS/SSL to protect sensitive data.
* **Client Data**: Client preferences are stored securely (e.g., encrypted database fields).
* **Audit Logging**: All report generation and delivery actions are logged for troubleshooting.

**Scalability**

* The system supports up to 100 concurrent client reports.
* Database queries are optimized with indexes on frequently accessed fields (e.g., execution date, rule ID).
* Report generation is parallelized for large datasets using multiprocessing or async frameworks (e.g., asyncio).

**Monitoring and Maintenance**

* **Monitoring**: The system logs metrics (e.g., generation time, delivery success) to a monitoring tool (e.g., Prometheus).
* **Alerts**: Administrators are notified of failures via email or a monitoring dashboard.
* **Maintenance**: The system includes scripts for cleaning up archived reports older than 30 days.

**LaTeX Preamble for PDF Reports**

* Uses a minimal set of packages from texlive-full and texlive-fonts-extra for compatibility.
* Fonts: Uses Noto Serif for standard text to ensure readability across platforms.
* Ensures all environments (e.g., document, table) are properly closed to avoid compilation errors.

**Epic: Standardization of Trend Rules and Readjustment of Legacy Rules on DQ4QD Platform**

**Objective**

To standardize all custom trend rules into a unified format on the DQ4QD platform, enabling users to map rules during workflow creation without needing to create separate rules for each use case, and to readjust all previously executed trend rules to align with the new standardized format.

**Description**

This epic focuses on enhancing the DQ4QD platform by implementing a standardized format for trend rules, streamlining rule creation and management. Users will be able to configure rule mappings directly within the workflow creation process, reducing redundancy and improving efficiency. Additionally, the epic includes readjusting all existing trend rules to conform to the new standard, ensuring consistency across the platform. This initiative aims to improve user experience, maintain data integrity, and simplify maintenance of trend rules.

**Acceptance Criteria**

1. **Standardized Rule Format**:
   * A single, unified trend rule format is defined and implemented on the DQ4QD platform.
   * Users can map trend rules during workflow creation without creating new rules for each use case.
   * The standardized format supports all existing and foreseeable use cases for trend rules.
2. **Rule Mapping in Workflow Creation**:
   * The workflow creation interface includes a rule mapping feature that allows users to configure trend rules using the standardized format.
   * The mapping process is intuitive, with clear documentation and tooltips for guidance.
   * Mapping supports dynamic inputs based on workflow requirements.
3. **Readjustment of Legacy Rules**:
   * All previously executed trend rules are identified and migrated to the new standardized format.
   * Data integrity is maintained during the readjustment process, with no loss of historical rule execution data.
   * A validation process ensures that readjusted rules produce equivalent outcomes to their original versions.
4. **User Experience and Performance**:
   * The rule mapping and readjustment processes do not degrade platform performance (e.g., response times remain within acceptable thresholds).
   * Users receive clear feedback on the success or failure of rule mapping and readjustment operations.
   * Training materials or help documentation are updated to reflect the new standardized rule format and processes.
5. **Testing and Validation**:
   * Unit tests cover the new standardized rule format and mapping functionality.
   * Integration tests verify that rule mapping integrates seamlessly with workflow creation.
   * End-to-end tests confirm that readjusted legacy rules function correctly and align with the standardized format.
   * A rollback mechanism is in place in case of issues during legacy rule readjustment.

**Design Details**

**Standardized Rule Format**

* **Structure**: The standardized trend rule format will be defined as a JSON schema, including fields for rule ID, conditions, actions, parameters, and metadata (e.g., creation date, author).
* **Flexibility**: The schema will support dynamic conditions and actions to accommodate various use cases, using a key-value pair approach for parameters.
* **Validation**: A server-side validation layer will ensure that all rules conform to the schema before saving or execution.

**Rule Mapping in Workflow Creation**

* **UI Component**: A new rule mapping section will be added to the workflow creation interface, featuring a form-based editor with dropdowns, text inputs, and conditional logic builders.
* **Backend Support**: A REST API endpoint will handle rule mapping requests, validating inputs against the standardized schema and saving mappings to the database.
* **Dynamic Inputs**: The UI will dynamically adjust available fields based on the selected workflow type, using metadata from the standardized rule schema.

**Legacy Rule Readjustment**

* **Migration Script**: A Python-based migration script will be developed to parse existing trend rules, transform them into the standardized format, and update the database.
* **Data Integrity Checks**: Pre- and post-migration validation will compare rule execution outcomes to ensure equivalence.
* **Logging and Auditing**: The migration process will log all changes, including any errors or warnings, for audit purposes.
* **Rollback Mechanism**: A backup of the original rules will be maintained, allowing restoration in case of migration failures.

**Database Changes**

* **Schema Update**: The trend rules table will be updated to support the new JSON schema, with indexes on frequently queried fields (e.g., rule ID, workflow ID).
* **Migration Plan**: A phased migration will be implemented, starting with a pilot subset of rules, to minimize risk.

**Performance Considerations**

* **Caching**: Frequently accessed rules will be cached using Redis to reduce database load.
* **Batch Processing**: Legacy rule readjustment will be performed in batches to avoid performance bottlenecks.
* **Monitoring**: Performance metrics (e.g., API response times, migration throughput) will be monitored using existing platform tools.

**Documentation and Training**

* **User Guides**: Updated documentation will cover the standardized rule format, rule mapping process, and FAQs.
* **Training Sessions**: Optional webinars or tutorials will be offered to help users transition to the new system.

**Epic: Standardization and Readjustment of Reconciliation Rules on DQ4QD Platform**

**Epic Title**

Standardization of Reconciliation Rules and Readjustment of Legacy Rules on DQ4QD Platform

**Objective**

To streamline the reconciliation process by standardizing all custom reconciliation rules into a unified format on the DQ4QD platform, enabling users to map rules during workflow creation without needing to create separate rules for each use case. Additionally, readjust all previously executed reconciliation rules to align with the new standardized format, ensuring consistency and efficiency across the platform.

**Description**

The DQ4QD platform currently supports custom reconciliation rules tailored to specific use cases, requiring users to create new rules for each scenario. This approach leads to inefficiencies, increased maintenance, and potential inconsistencies. This epic aims to introduce a standardized rule format that allows users to perform rule mapping directly within the workflow creation process. By implementing a unified structure, users can configure rules flexibly without redundant rule creation. Furthermore, all existing (legacy) reconciliation rules will be realigned to conform to the new standardized format, ensuring seamless integration and consistency across past and future reconciliations.

The solution will enhance user experience, reduce rule management overhead, and improve the scalability of the reconciliation process on the DQ4QD platform.

**Design Details**

1. **Standardized Rule Format**:
   * Develop a universal rule template that supports all existing and future reconciliation use cases.
   * Include configurable parameters (e.g., data fields, conditions, thresholds) to allow rule mapping during workflow creation.
   * Ensure the format is extensible to accommodate new reconciliation requirements without structural changes.
2. **Rule Mapping in Workflow Creation**:
   * Integrate a rule mapping interface within the DQ4QD workflow creation module.
   * Allow users to select the standardized rule template and configure parameters specific to the workflow.
   * Provide validation checks to ensure rule configurations are complete and error-free.
3. **Readjustment of Legacy Rules**:
   * Identify and catalog all existing custom reconciliation rules executed on the DQ4QD platform.
   * Develop an automated migration script or tool to convert legacy rules into the standardized format.
   * Include a manual review process for complex or ambiguous rules to ensure accuracy during migration.
   * Preserve historical rule execution data by mapping old rule outputs to the new format.
4. **User Interface Enhancements**:
   * Update the DQ4QD platform’s UI to support the new rule mapping functionality.
   * Provide clear documentation and tooltips to guide users through the rule configuration process.
   * Add a rule management dashboard to view, edit, and audit both new and migrated rules.
5. **Testing and Validation**:
   * Validate the standardized rule format against all known use cases to ensure coverage.
   * Test the migration tool with a sample of legacy rules to confirm accurate conversion.
   * Perform end-to-end testing of the rule mapping feature within workflow creation.
   * Conduct user acceptance testing (UAT) with stakeholders to verify usability and functionality.
6. **Performance Considerations**:
   * Optimize the rule mapping and execution process to minimize performance impact on the DQ4QD platform.
   * Ensure the migration of legacy rules does not disrupt ongoing reconciliations or platform availability.

**Acceptance Criteria**

1. **Standardized Rule Format**:
   * A universal rule template is implemented and supports all current reconciliation use cases.
   * Users can configure rules via parameters during workflow creation without creating new rule definitions.
   * The template is validated to handle at least 95% of existing use cases without requiring custom workarounds.
2. **Rule Mapping Functionality**:
   * Users can access a rule mapping interface within the workflow creation module.
   * Rule configurations are saved, validated, and applied correctly during reconciliation execution.
   * Error messages are displayed for invalid or incomplete rule configurations.
3. **Legacy Rule Readjustment**:
   * 100% of legacy reconciliation rules are cataloged and assessed for migration.
   * At least 90% of legacy rules are automatically migrated to the standardized format using the migration tool.
   * A manual review process is available for rules that cannot be automatically migrated.
   * Historical rule execution data remains accessible and correctly mapped to the new format.
4. **User Experience**:
   * The updated UI is intuitive, with clear instructions for rule mapping and configuration.
   * Documentation and tooltips are available to assist users in adopting the new process.
   * A rule management dashboard allows users to view and audit standardized and migrated rules.
5. **Testing and Stability**:
   * The standardized rule format and mapping functionality pass all functional and performance tests.
   * The migration tool successfully converts a representative sample of legacy rules without data loss.
   * UAT confirms that stakeholders can use the new features without significant issues.
   * No critical bugs or performance degradation are observed during testing.
6. **Deployment and Rollout**:
   * The solution is deployed to the DQ4QD platform with minimal downtime.
   * A rollback plan is in place in case of deployment issues.
   * Training materials and release notes are provided to users to facilitate adoption.

**Epic: Intelligent Rule Suggestion Engine for Data Quality Rule Application**

**Objective**

To develop a rule suggestion engine within the DQ4QD platform that leverages Generative AI to automatically recommend data quality rules for data elements based on column business descriptions, data profiling details, past rule execution patterns, and usage across data elements. The engine will suggest both client-specific custom rules and standard rules, streamlining the rule selection process and enhancing data quality management efficiency.

**Acceptance Criteria**

1. **Rule Suggestion Accuracy**:
   * The engine suggests relevant rules with at least 80% accuracy based on test datasets with known business descriptions and profiling details.
   * Suggestions include both client-specific custom rules and standard rules from the rule repository.
2. **Input Processing**:
   * The system processes column business descriptions, data profiling details (e.g., data types, null counts, value distributions), past rule execution history, and rule usage across other data elements.
   * Input data is validated for completeness and consistency before processing.
3. **Generative AI Integration**:
   * A Generative AI model is integrated to analyze input data and generate rule suggestions.
   * The AI model is trained or fine-tuned on a dataset of existing rule applications and metadata.
4. **User Interface**:
   * Users can view suggested rules for each data element in the DQ4QD platform UI.
   * Suggestions include rule definitions, logic, and a confidence score for each recommendation.
   * Users can accept, reject, or modify suggested rules, with feedback stored for model improvement.
5. **Performance**:
   * Rule suggestions are generated within 10 seconds per data element for datasets with up to 100 columns.
   * The system supports batch processing for multiple data elements.
6. **Feedback Loop**:
   * User feedback on suggestions (accept/reject) is captured and used to retrain or fine-tune the AI model.
   * A mechanism exists to track suggestion improvements over time.
7. **POC Validation**:
   * The Proof of Concept (POC) is tested on a sample dataset with at least 10 data elements and 50 rules.
   * The POC demonstrates integration with the existing rule repository and DQ4QD platform.

**Design Details**

**System Architecture**

* **Data Ingestion Module**:
  + Extracts column metadata, including business descriptions, data profiling results (e.g., min/max values, null percentages, data types), and historical rule execution data.
  + Connects to the DQ4QD rule repository to fetch rule definitions and logic.
* **Generative AI Model**:
  + Utilizes a pre-trained large language model (LLM) fine-tuned on rule application data.
  + Inputs: Column metadata, profiling details, and rule usage patterns.
  + Outputs: A ranked list of suggested rules with confidence scores.
  + Training Data: Historical rule applications, client-specific rule metadata, and standard rule definitions.
* **Rule Mapping Engine**:
  + Maps AI-generated suggestions to specific rules in the repository.
  + Filters suggestions based on client-specific constraints (e.g., custom rule priority).
* **Feedback Loop**:
  + Stores user feedback (accept/reject) in a feedback database.
  + Periodically retrains the AI model using feedback to improve suggestion accuracy.
* **User Interface**:
  + Integrates with the DQ4QD platform UI to display suggestions.
  + Provides options to accept, reject, or edit suggested rules.
  + Displays rule details, including definition, logic, and confidence score.

**Workflow**

1. **Data Input**:
   * User selects a data element in the DQ4QD platform.
   * The system retrieves column metadata, profiling details, and rule usage history.
2. **AI Processing**:
   * The Generative AI model processes inputs and generates a list of suggested rules.
   * The rule mapping engine matches suggestions to repository rules.
3. **Suggestion Presentation**:
   * Suggested rules are displayed in the UI with confidence scores and rule details.
   * Users can accept, reject, or modify suggestions.
4. **Feedback Collection**:
   * User actions are logged and stored for model retraining.
5. **Rule Application**:
   * Accepted rules are applied to the data element for data quality checks.

**Technical Considerations**

* **AI Model Selection**:
  + Use a transformer-based LLM (e.g., BERT or a similar model) for natural language understanding of business descriptions.
  + Fine-tune the model on a dataset of rule applications to align with DQ4QD’s domain.
* **Scalability**:
  + Deploy the AI model on a cloud-based infrastructure to handle large datasets.
  + Optimize for low-latency inference using techniques like model quantization.
* **Security**:
  + Ensure column metadata and business descriptions are anonymized if sensitive.
  + Implement role-based access control for rule suggestions.
* **Integration**:
  + Use APIs to connect the rule suggestion engine with the DQ4QD rule repository and UI.
  + Ensure compatibility with existing rule execution pipelines.

**POC Scope**

* **Dataset**:
  + Use a synthetic dataset with 10 data elements, each with business descriptions, profiling details, and associated rules.
  + Include 50 rules (30 standard, 20 client-specific) in the rule repository.
* **AI Model**:
  + Fine-tune a small LLM (e.g., DistilBERT) on synthetic rule application data.
  + Evaluate suggestion accuracy using precision and recall metrics.
* **Deliverables**:
  + Functional rule suggestion engine integrated with the DQ4QD platform.
  + Documentation on model training, integration, and usage.
  + Test report with accuracy and performance metrics.

**Description**

This epic focuses on building an intelligent rule suggestion engine for the DQ4QD platform to automate the selection of data quality rules for data elements. By leveraging Generative AI, the engine analyzes column business descriptions, data profiling details, past rule execution history, and rule usage patterns to recommend relevant rules from both client-specific and standard rule repositories. The system aims to reduce manual effort, improve rule application accuracy, and enhance data quality management efficiency. As a Proof of Concept, the engine will be tested on a small dataset to validate its feasibility and integration with the DQ4QD platform. User feedback will be incorporated to refine the AI model, ensuring continuous improvement in suggestion accuracy.

**Epic: Custom Notification Configuration for DQ4QD Rule Execution**

**Epic Title**

Enable Custom Notification Configuration Based on User-Defined Rule Execution Requirements in DQ4QD

**Objective**

To enhance the DQ4QD system by providing users with the ability to define custom notification preferences based on specific rule execution outcomes, such as rule failure or non-execution, tailored to their unique requirements. This will improve user experience by allowing flexible and personalized notification settings per rule, ensuring users receive relevant alerts for their data quality monitoring needs.

**Description**

The DQ4QD system currently lacks the flexibility to allow users to configure custom notifications based on specific rule execution outcomes. Some users require notifications when a rule fails due to an empty underlying table, while others prefer notifications when a rule is not executed. This epic aims to introduce a feature that enables users to define custom notification rules within the DQ4QD system. Users will be able to specify conditions for notifications (e.g., rule failure, non-execution) and configure the type, content, and delivery method of notifications per rule. The system will support multiple notification channels (e.g., email, in-app alerts) and provide an intuitive interface for managing these configurations. This enhancement will ensure that users receive timely and relevant alerts tailored to their specific data quality requirements.

**Acceptance Criteria**

1. **User Interface for Configuration**:
   * Users can access a new section in the DQ4QD UI to configure custom notifications for each rule.
   * The UI allows users to select conditions for notifications (e.g., rule failure, rule not executed, specific failure reasons like empty table).
   * Users can specify notification details, including message content, priority, and delivery channel (e.g., email, in-app notification).
   * The UI validates user inputs to ensure configurations are complete and consistent.
2. **Notification Rule Processing**:
   * The system evaluates rule execution outcomes and triggers notifications based on user-defined conditions.
   * Notifications are generated accurately for scenarios such as:
     + Rule failure due to an empty underlying table.
     + Rule not executed due to specific conditions.
   * The system supports multiple notification conditions per rule.
3. **Notification Delivery**:
   * Notifications are delivered through user-selected channels (e.g., email, in-app alerts).
   * Notifications include user-defined message content and relevant rule execution details (e.g., rule name, failure reason, timestamp).
   * The system ensures notifications are sent in a timely manner after rule execution.
4. **Configuration Management**:
   * Users can view, edit, or delete existing notification configurations.
   * The system persists notification configurations securely and associates them with specific rules and users.
   * Changes to configurations are logged for audit purposes.
5. **Error Handling and Validation**:
   * The system provides clear error messages if a notification configuration is invalid or cannot be processed.
   * Notifications are not triggered for misconfigured rules to prevent spam or irrelevant alerts.
6. **Scalability and Performance**:
   * The system supports notification configurations for a large number of rules without significant performance degradation.
   * Notification processing does not impact the performance of rule execution in DQ4QD.
7. **Testing and Validation**:
   * Unit tests cover notification rule evaluation and delivery logic.
   * Integration tests verify end-to-end functionality from configuration to notification delivery.
   * User acceptance testing confirms the feature meets diverse user requirements (e.g., empty table failure, non-execution scenarios).

**Design Details**

**System Architecture**

* **Frontend**:
  + A new "Notification Configuration" module will be added to the DQ4QD UI, built using the existing frontend framework (e.g., React).
  + The module includes forms for defining notification conditions, message templates, and delivery preferences.
  + Client-side validation ensures user inputs are correct before submission.
* **Backend**:
  + A new API endpoint will be created to manage notification configurations (e.g., /api/rules/{ruleId}/notifications).
  + The backend will store notification configurations in the existing DQ4QD database, with a new table (notification\_configs) linked to rules and users.
  + A notification service will be implemented to evaluate rule execution outcomes and trigger notifications based on user configurations.
  + The service integrates with existing notification channels (e.g., email server, in-app messaging system).
* **Database Schema**:
  + Table: notification\_configs
    - Columns: id, rule\_id, user\_id, condition\_type (e.g., failure, not\_executed), failure\_reason (e.g., empty\_table), message\_template, delivery\_channel, priority, created\_at, updated\_at.
  + Indexes on rule\_id and user\_id for efficient querying.
* **Notification Processing**:
  + A background job (e.g., using a task queue like Celery) will process rule execution outcomes and match them against notification configurations.
  + The job generates notifications and dispatches them to the appropriate delivery channel.
  + Circuit breakers or rate limiting will be implemented to prevent notification overload.

**Integration Points**

* **Rule Execution Engine**: The notification service subscribes to rule execution events (e.g., success, failure, not executed) to trigger notifications.
* **Notification Channels**: Integration with existing email and in-app notification systems, with extensibility for additional channels (e.g., Slack) in the future.
* **Audit Logging**: Notification configuration changes and notification triggers are logged in the existing audit trail system.

**User Flow**

1. User navigates to the "Notification Configuration" section for a specific rule in the DQ4QD UI.
2. User defines a notification condition (e.g., "Notify if rule fails due to empty table").
3. User specifies the notification details (e.g., message: "Rule {rule\_name} failed due to empty table", channel: email, priority: high).
4. User saves the configuration, which is validated and persisted.
5. When the rule is executed, the system evaluates the outcome and triggers the notification if the condition is met.
6. User receives the notification via the selected channel and can view it in the DQ4QD notification history.

**Technical Considerations**

* **Security**: Notification configurations are scoped to authenticated users, with role-based access control to prevent unauthorized access.
* **Extensibility**: The system is designed to support additional condition types and notification channels in the future.
* **Localization**: Notification message templates support multiple languages using existing DQ4QD localization framework.
* **Monitoring**: Metrics are collected on notification processing (e.g., success rate, latency) and integrated with the existing monitoring system.

**Epic: Python-Based Stand Block Support for DQ4QD System**

**Epic Title**

Enhance the DQ4QD System with Python-Based Stand Block Support

**Objective**

The objective of this epic is to extend the DQ4QD system's staging block functionality to support Python-based (PySpark) code execution for generating staging data, while maintaining existing SQL-based query support. This enhancement will provide users with greater flexibility to create staging data using PySpark scripts alongside custom SQL queries, enabling more complex data processing within the DQ4QD workflow.

**Description**

Currently, the DQ4QD system allows users to stage data from multiple tables using a staging block, where users define custom SQL queries that are executed to prepare staging data for workflow execution. This epic introduces a new Python-based staging block, enabling users to write and execute PySpark code to generate staging data. The system will support both SQL-based and Python-based staging blocks, allowing users to choose the appropriate method based on their requirements. The Python-based staging block will integrate seamlessly with the existing DQ4QD architecture, ensuring compatibility with workflows and maintaining performance and scalability.

**Acceptance Criteria**

1. **Functional Requirements**:
   * Users can create a Python-based staging block in the DQ4QD system to write and execute PySpark code for staging data.
   * The system continues to support existing SQL-based staging blocks without any disruption.
   * Users can switch between SQL and Python-based staging blocks within the same workflow configuration.
   * PySpark code in the Python-based staging block can read data from multiple tables and generate staging data compatible with downstream workflows.
   * The system validates PySpark code syntax and provides meaningful error messages for invalid code.
2. **Integration**:
   * Python-based staging blocks integrate with the existing DQ4QD workflow execution pipeline.
   * Staging data generated by PySpark code is accessible to subsequent workflow steps in the same format as SQL-based staging data.
   * The system supports authentication and authorization for PySpark code execution, consistent with SQL-based staging blocks.
3. **Performance and Scalability**:
   * Python-based staging blocks execute PySpark code efficiently, with performance comparable to or better than SQL-based staging blocks for equivalent operations.
   * The system handles large datasets in PySpark-based staging blocks without significant performance degradation.
   * Resource allocation for PySpark execution (e.g., Spark cluster resources) is configurable to optimize performance.
4. **User Experience**:
   * The DQ4QD UI provides a code editor with syntax highlighting and autocompletion for PySpark code in Python-based staging blocks.
   * Users can test and debug PySpark code within the staging block configuration interface.
   * Documentation and examples for Python-based staging blocks are available in the DQ4QD user guide.
5. **Reliability and Error Handling**:
   * The system logs execution details and errors for Python-based staging blocks, accessible via the DQ4QD monitoring interface.
   * Failures in PySpark code execution do not impact the stability of the DQ4QD system.
   * The system supports retry mechanisms for transient failures in PySpark code execution.

**Design Details**

**System Architecture**

* **Staging Block Module**:
  + Extend the existing staging block module to support two types: SQL-based and Python-based.
  + Introduce a new Python-based staging block handler that interfaces with a PySpark execution engine.
* **PySpark Execution Engine**:
  + Integrate a Spark cluster (managed or serverless) to execute PySpark code.
  + Use a SparkSession to initialize and manage PySpark jobs within the DQ4QD system.
  + Support dynamic resource allocation for PySpark jobs based on data size and complexity.
* **Data Interface**:
  + Provide APIs for PySpark code to read from source tables (e.g., via JDBC or Spark connectors).
  + Ensure staging data output from PySpark code is written to a temporary storage location (e.g., Parquet files) compatible with downstream workflows.
* **Security**:
  + Implement role-based access control (RBAC) for Python-based staging block execution.
  + Sanitize and validate PySpark code to prevent injection attacks or unauthorized operations.
* **UI Enhancements**:
  + Add a PySpark code editor to the staging block configuration interface, using a library like Monaco Editor for syntax highlighting and autocompletion.
  + Include a "Test Run" feature to execute PySpark code in a sandbox environment and display sample output or errors.

**Workflow Integration**

* **Execution Pipeline**:
  + Modify the workflow orchestrator to recognize Python-based staging blocks and route them to the PySpark execution engine.
  + Ensure staging data from Python-based blocks is registered in the DQ4QD metadata catalog for downstream access.
* **Error Handling**:
  + Capture and log PySpark job exceptions, including stack traces, in the DQ4QD monitoring system.
  + Implement configurable retry policies for PySpark job failures (e.g., network timeouts).

**Monitoring and Logging**

* **Metrics**:
  + Track execution time, resource usage (CPU, memory), and data volume for Python-based staging blocks.
  + Expose metrics via the DQ4QD monitoring dashboard.
* **Logs**:
  + Store detailed PySpark job logs in a centralized logging system (e.g., Elasticsearch).
  + Provide a log viewer in the DQ4QD UI for users to troubleshoot Python-based staging block issues.