IITM Pravartak Technologies

GROUP 5 - Enhancement to **pg_check**

Prepared by:

Alan Mathew Varghese (R6B 08), Anna Tomson (R6B 17)

Aromal A J (R6B 21), Arya Rajeev (R6B 23)

Date Created: 28-01-2025

Table of Contents

1. Introduction

1.1 Purpose

The purpose of this document is to enhance the key features of pgcheck. This aims to improve data integrity and security by implementing **Index Integrity Check, Password Policy Validation, History Tracking, and Automation & Alerting** mechanisms within a database environment.

1.2 Document Conventions

- **Must**: Indicates mandatory requirements.
- **Should**: Represents desirable but optional features.
- **May**: Denotes optional or future enhancements.

1.3 Product Scope

The system provides database administrators with tools to validate index integrity, enforce strict password policies, track historical health data, and automate alerting. This enhances database performance, prevents data corruption, and secures user authentication mechanisms.

1.4 References

- PostgreSQL Official Documentation
- NIST Password Security Guidelines
- OWASP Database Security Best Practices

2. Overall Description

2.1 Problem Statement

2.1.1 Existing Issues in pgCheck

- Limited customization and flexibility across features (e.g., password policies, health check parameters).

- Reactive functionality rather than proactive detection (e.g., index corruption, historical trends).
- Lack of advanced reporting and visualization tools for actionable insights.
- Minimal integration with third-party tools and limited support for modern DevOps workflows.

2.1.2 Possible Enhancements

- Enable real-time monitoring and predictive analytics to prevent potential issues.
- Introduce customizable configurations for thresholds, policies, and reports.
- Add advanced visualizations and detailed, actionable recommendations for all features.
- Integrate with external platforms (e.g., Prometheus, Grafana, Slack) for seamless workflow compatibility.

2.1.3 Product Functions

The system provides the following functionalities:

- **Index Integrity Check**: Detects index corruption and redundant indexes.
- **Password Policy Validation**: Enforces password complexity, expiration, and uniqueness requirements.
- **History Tracking**: Tracks historical health-check data and presents trends over time (e.g., table growth, index bloat).
- **Automation and Alerting**: Automates health checks and integrates with monitoring tools.

2.2 Product Perspective

This system integrates with existing database management systems (e.g., PostgreSQL) to:

- Detect and rectify index corruption.
- Enforce robust password policies.
- Track and analyze historical database health trends.
- Automate health checks and alerting.

3. System Features

3.1 Index Integrity Check

- Validates the integrity and usability of indexes.
- Detects **corrupt** or **redundant** indexes.

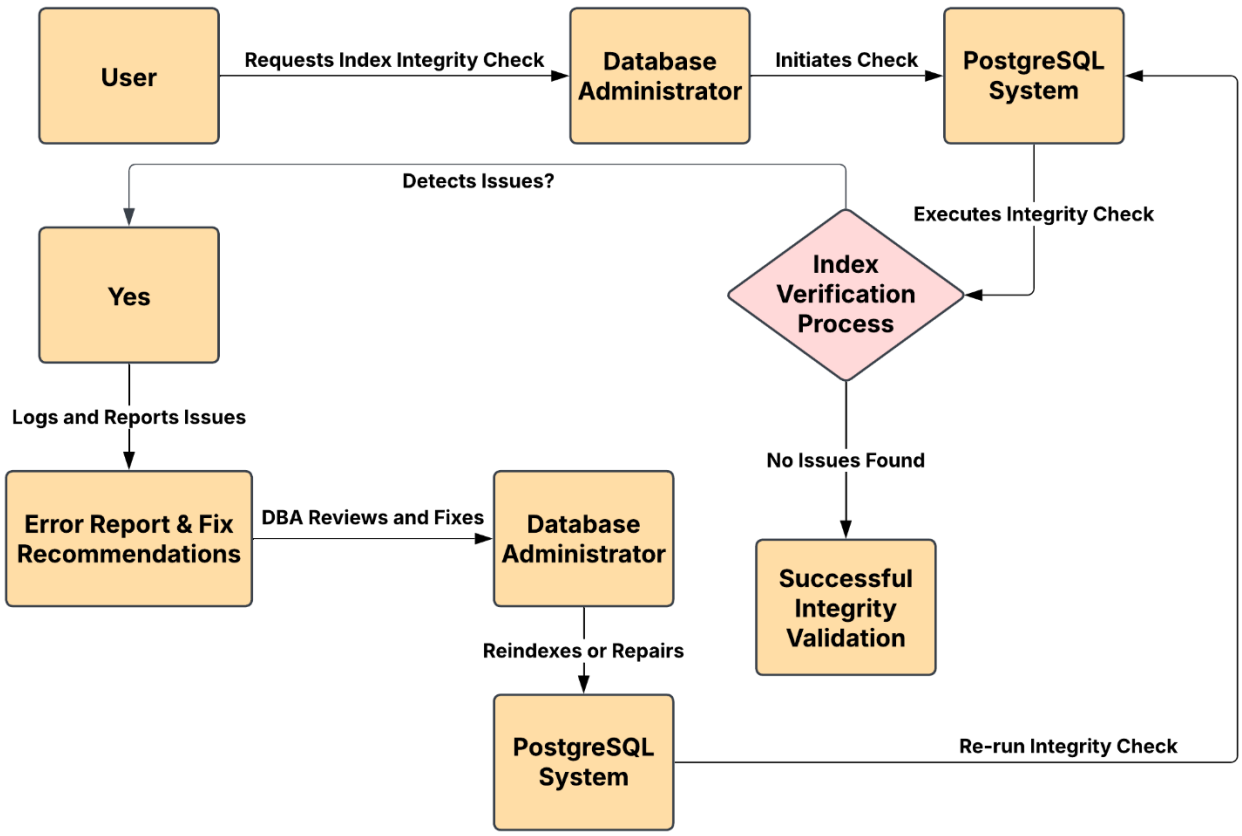
Priority: High

Functional Requirements

- **FR-1**: The system must scan indexes periodically.
- **FR-2**: If an index is corrupted, the system must log an alert.
- **FR-3**: The system should provide recommendations for index optimization.
- **FR-4**: Support manual and automated integrity checks.

Example Query for Checking Index Integrity:

```
REINDEX INDEX index_name;
```



3.2 Password Policy Validation

- Enforces password security policies.
- Prevents the use of weak, compromised, or expired passwords.

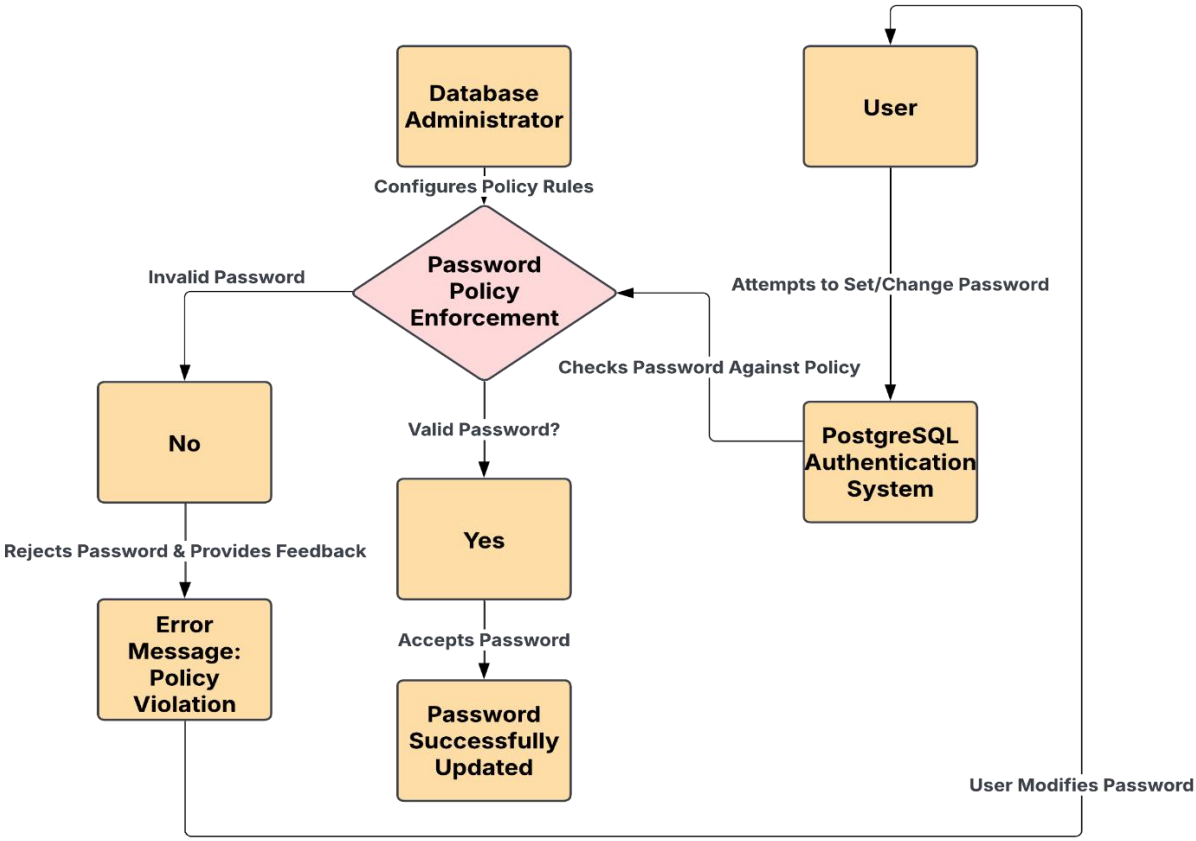
Priority: Critical

Functional Requirements

- **FR-5**: Passwords must be at least **12 characters** long.
- **FR-6**: Passwords must include **uppercase, lowercase, numbers, and special characters**.
- **FR-7**: Passwords must **expire every 90 days**.
- **FR-8**: Users must not reuse any of their last **5 passwords**.
- **FR-9**: Failed login attempts should trigger **account lockout after 5 tries**.

Example Query for Enforcing Password Policy in PostgreSQL:

```
ALTER SYSTEM SET passwordcheck.min_length TO 12;
```



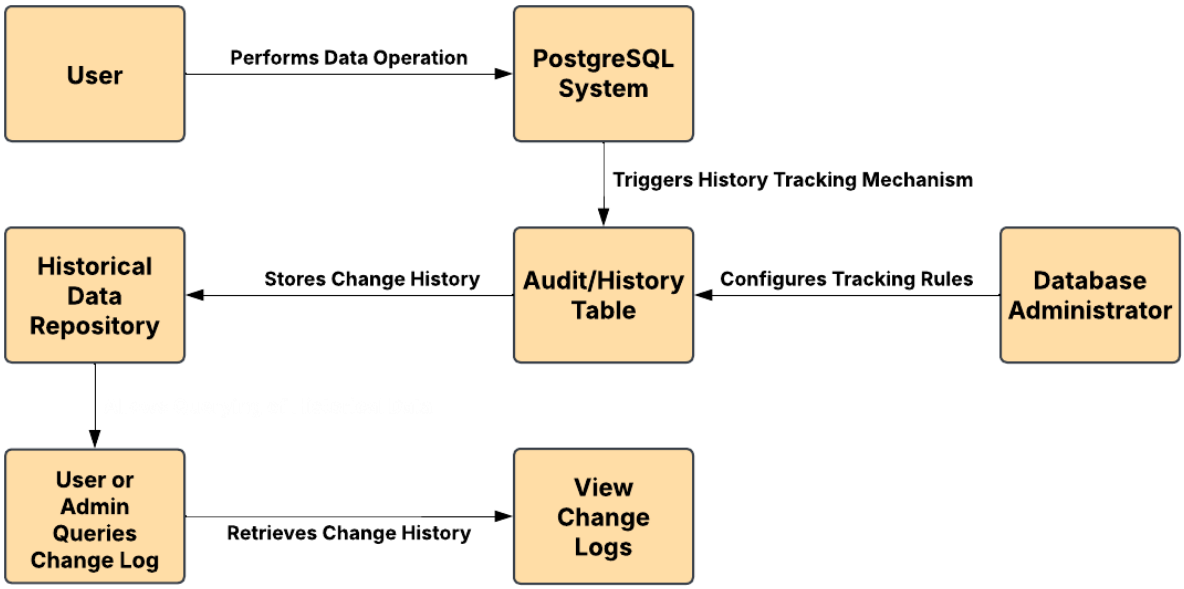
3.3 History Tracking

- Tracks historical health-check data and presents trends over time.
- Monitors database table growth, index bloat, and other performance metrics.

Priority: Medium

Functional Requirements

- **FR-10**: The system must log health-check results over time.
- **FR-11**: The system should generate reports on historical trends.
- **FR-12**: The system must provide visual analytics for trend monitoring.



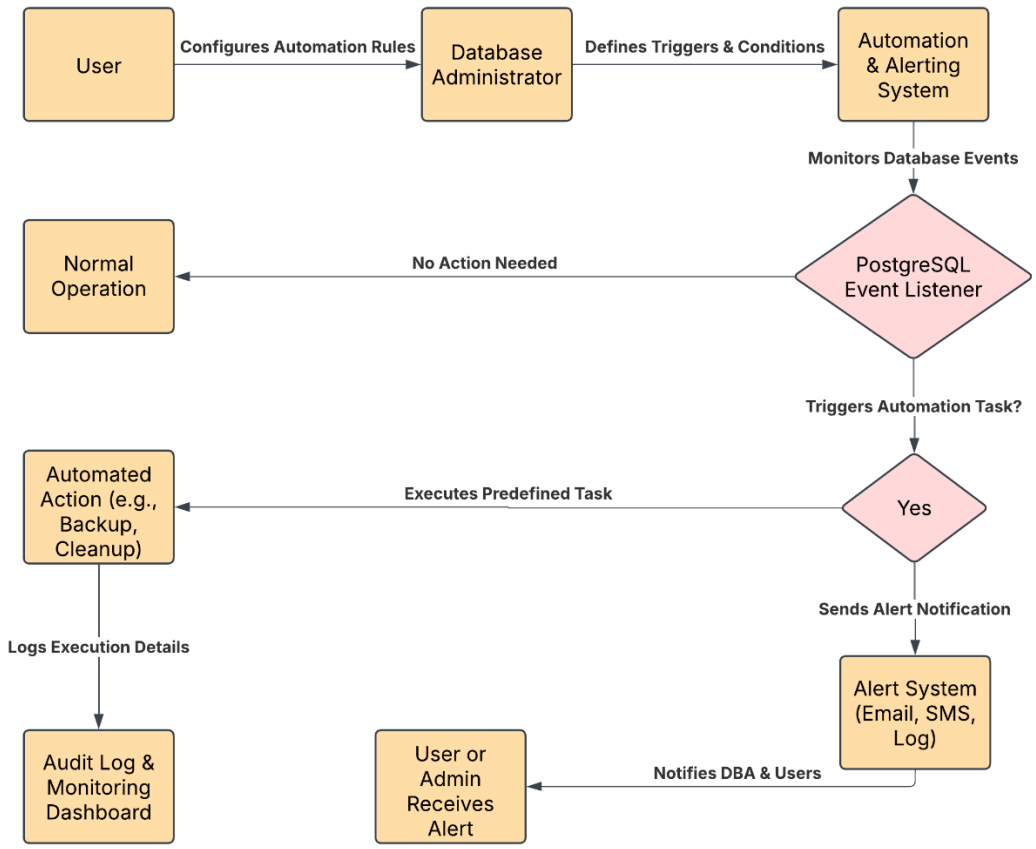
3.4 Automation and Alerting

- Automates health checks and integrates with monitoring tools.
- Sends alerts based on predefined thresholds.

Priority: High

Functional Requirements

- **FR-13**: The system must schedule automated health checks.
- **FR-14**: The system should generate daily/weekly health reports via email or webhooks.
- **FR-15**: The system must enable user-defined thresholds for triggering alerts.
- **FR-16**: The system must integrate with monitoring tools like Prometheus, Grafana, or Zabbix.



4. Nonfunctional Requirements

- Performance: Minimal impact on database performance.
- Security: **SCRAM-SHA-256** hashing for passwords.

Working:

1. **Salting:** The user's password is combined with a random value (salt) before hashing.
2. **Hashing:** The salted password is hashed multiple times using SHA-256 to make brute-force attacks difficult.
3. **Challenge-Response:** The server and client exchange authentication messages, ensuring that the password is not sent in plaintext over the network.
4. **Replay Attack Protection:** Each authentication attempt includes unique, time-sensitive data, preventing reuse of old authentication messages.

5. Test Cases

5.1 Index Integrity Check

Test Case ID	Test Case ID	Test Case ID	Test Case ID
TC-01	Validate periodic index scan	Schedule an index scan	System logs successful scan
TC-02	Detect corrupted index	Corrupt an index	System logs an alert
TC-03	Check redundant indexes	Create redundant indexes	System flags redundant indexes
TC-04	Perform manual integrity check	Execute REINDEX INDEX index_name;	System successfully reindexes

5.2 Password Policy Validation

Test Case ID	Description	Input	Expected Output
TC-05	Enforce password length	Set password to 8 characters	System rejects the password
TC-06	Enforce character complexity	Set password to only lowercase letters	System rejects the password
TC-07	Validate expiration policy	Login with expired password (after 90 days)	System prompts for password change
TC-08	Prevent password reuse	Set password to a previous one	System rejects the password
TC-09	Lock account after failed attempts	Enter incorrect password 5 times	System locks the account

5.3 History Tracking

Test Case ID	Description	Input	Expected Output
TC-10	Log health-check results	Run health check	System stores logs in history
TC-11	Generate trend reports	Request report for past month	System generates a report
TC-12	Visual analytics generation	View trend graphs	System displays historical data

5.4 Automation and Alerting

Test Case ID	Description	Input	Expected Output
TC-13	Schedule automated checks	Set daily health checks	System runs checks automatically
TC-14	Generate email reports	Configure weekly email reports	System sends health summary via email
TC-15	Alert based on threshold	Set alert for index corruption	System sends alert when condition is met
TC-16	Integration with monitoring tools	Connect to Prometheus/Grafana	System successfully logs events

6. Timeline

- **Phase 1**: Requirements gathering and design.
- **Phase 2**: Development and initial testing.
- **Phase 3**: Deployment and user training.

7. Conclusion

By enhancing critical features like index integrity checks, password policy enforcement, historical health tracking, and automated alerting, the system ensures compliance with security standards and promotes efficient database management. With its focus on scalability, minimal performance impact, and secure communication, pgcheck provides a reliable framework for database administrators, developers, and security auditors to prevent data corruption and maintain secure user authentication mechanisms.