IITM Pravartak Technologies

Enhancement to pg_check

Prepared by:

Alan Mathew Varghese (TKM22CS019)

Anna Tomson (TKM22CS033)

Aromal A J (TKM22CS040)

Arya Rajeev (TKM22CS046)

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 implement **Index Integrity Check, Password Policy Validation, History Tracking, and Automation & Alerting** mechanisms within a postgreSQL 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.

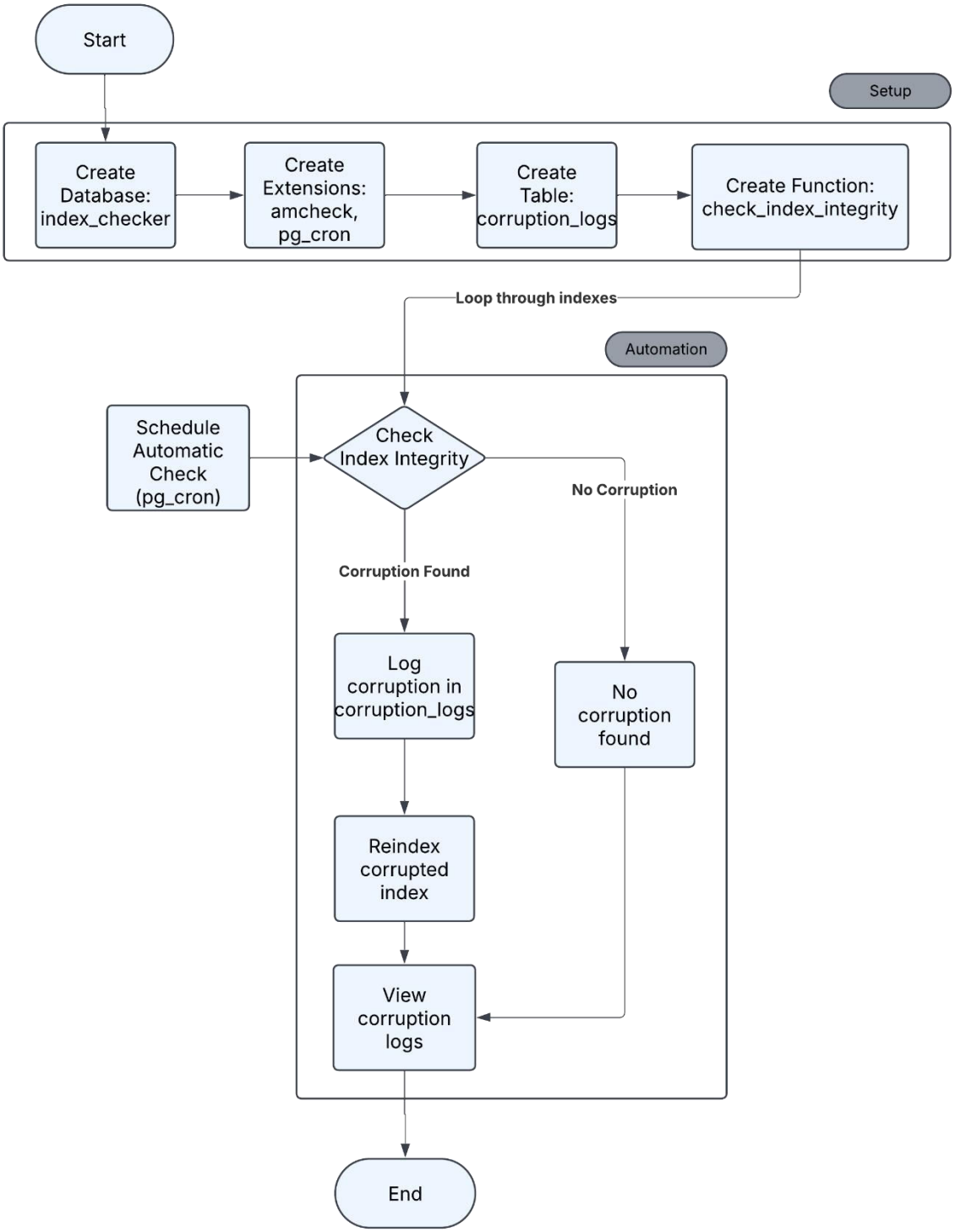
Priority: High

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;
```


Start
Setup
Create Database: index_checker
Create Extensions: amcheck, pg_cron
Create Table: corruption_logs
Create Function: check_index_integrity
Loop through indexes
Automation
Schedule Automatic Check (pg_cron)
Check Index Integrity
No Corruption
Corruption Found
Log corruption in corruption_logs
No corruption found
Reindex corrupted index
View corruption logs
End

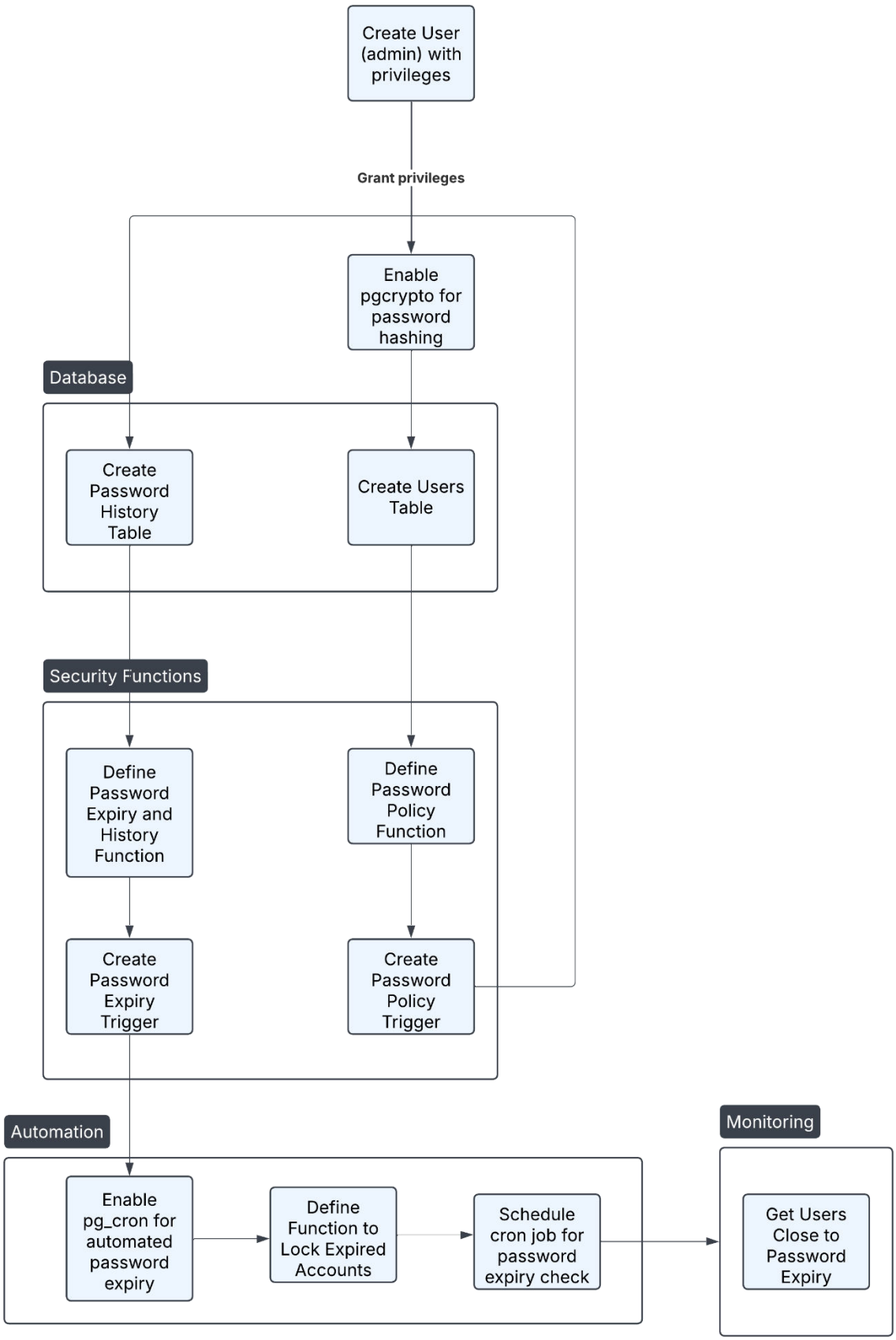
3.2 Password Policy Validation

- 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 **3 passwords**.
- **FR-9**: Automated password expiry check and account locking.


Create User (admin) with privileges
Grant privileges
Enable pgcrypto for password hashing
Database
Create Password History Table
Create Users Table
Security Functions
Define Password Expiry and History Function
Define Password Policy Function
Create Password Expiry Trigger
Create Password Policy Trigger
Automation
Enable pg_cron for automated password expiry
Define Function to Lock Expired Accounts
Schedule cron job for password expiry check
Monitoring
Get Users Close to Password Expiry

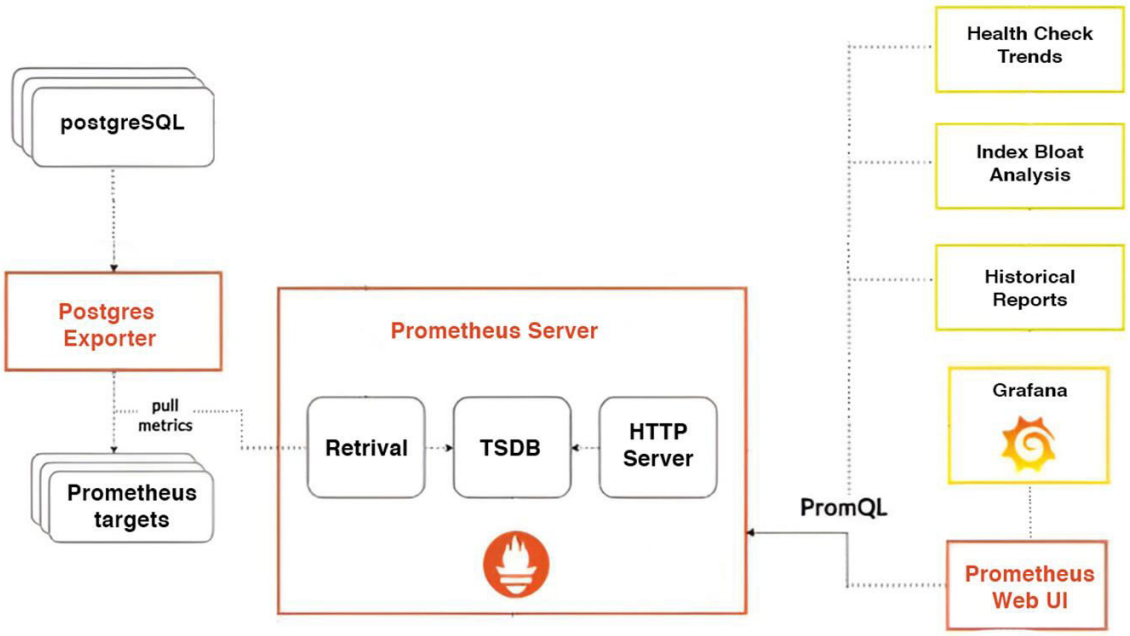
3.3 History Tracking

- 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 using the collected data.
- **FR-12**: The system must provide visual analytics for monitoring trends.
- **FR-13:** The system must track index bloat and provide detailed insights.



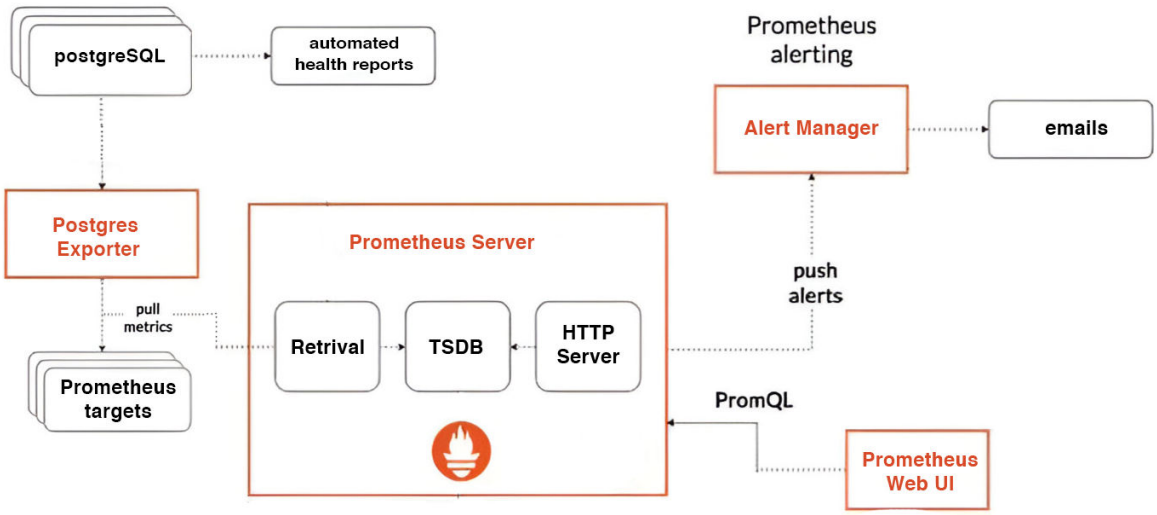
3.4 Automation and Alerting

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

Priority: High

Functional Requirements

- **FR-14**: The system must schedule automated health checks.
- **FR-15**: The system must enable user-defined thresholds for triggering alerts.
- **FR-16**: The system must integrate with monitoring tools like Prometheus.



4. Test Cases

4.1 Index Integrity Check

Test Case ID	Description	Input	Expected Output
TC-01	Validate periodic index scan	Schedule an index scan using pg_cron	System logs a successful scan in pg_cron.job
TC-02	Detect corrupted index	Introduce index corruption	System logs an alert in corruption_logs
TC-03	Check redundant indexes	Create redundant indexes	System flags redundant indexes
TC-04	Perform manual integrity check	Execute CALL check_index_integrity()	System successfully confirms index integrity
TC-05	Auto-fix corrupted index	Corrupt an index, then run CALL check_index_integrity()	System logs corruption and automatically reindexes
TC-06	View corruption history	Query SELECT * FROM corruption_logs	System displays a list of detected corrupt indexes with timestamps
TC-07	Validate scheduled job execution	Query SELECT * FROM cron.job	System confirms job scheduling for daily checks

4.2 Password Policy Validation

Test Case ID	Description	Input	Expected Output
TC-01	Enforce password length	Set password to 8 characters	System rejects the password
TC-02	Enforce character complexity	Set password without numbers or special chars	System rejects the password
TC-03	Allow valid password	Set password to StrongPass123!	System accepts the password
TC-04	Prevent password reuse	Change password to a previously used one	System rejects the password
TC-05	Password expiration check	Try to login with expired password	System prompts for password change
TC-06	Automatic account lock for expiry	Set password expiry to past date, run expiry function	System locks the account
TC-07	Notify users near expiration	Query users with password expiring in 5 days	System returns users close to expiry

4.3 History Tracking

Test Case ID	Description	Input	Expected Output
TC-01	Log health-check results	Run health check	System stores logs in history
TC-02	Generate trend reports	Request report for past month	System generates a report
TC-03	Visual analytics generation	View trend graphs	System displays historical data visually
TC-04	Track index bloat analysis	Execute index bloat query	System shows index bloat insights

4.4 Automation and Alerting

Test Case ID	Description	Input	Expected Output
TC-01	Integration with monitoring tools	Connect to Prometheus	System successfully logs events
TC-02	Schedule automated checks	Set automated health checks	System runs checks automatically
TC-03	Alert based on threshold	Enable user-defined thresholds for triggering alerts	System sends alert when condition is met

5. Timeline

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

6. 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.