Software Design Document (SDD)

for

<Enhancement on pgAgent Features: Job Dependency & Audit Logging>

Version 2.0

-In collaboration with IITM Pravartak Technologies

Date: 16-04-2025

Prepared by: Brian Christopher, Joel Daniel Pradeep

Institution: TKM College of Engineering

Table of Contents

1. Introduction

1.1. Purpose of the document

This Software Design Document (SDD) provides a detailed technical blueprint for enhancing pgAgent with Job Dependency Management and Audit Logging features. It outlines the architectural design, component specifications, and implementation strategies necessary for developers to implement these enhancements effectively.

1.2. Scope of the project

The project involves enhancing pgAgent with:

- Job Dependency Management: Implementation of sequential, conditional, and parallel execution of PostgreSQL jobs
- Audit Logging: Comprehensive logging of job-related activities for security, accountability, and debugging purposes

The enhancements aim to improve workflow automation capabilities within PostgreSQL environments and provide better monitoring of job execution processes.

1.3. Definitions & Acronyms

- pgAgent: Job scheduling agent for PostgreSQL databases
- Job Dependency: Configuration allowing jobs to be executed based on the status of other jobs
- Audit Log: Chronological record of activities related job creation, modification , deletion and execution
- PostgreSQL: Open-source object-relational database system
- pgAdmin: Administration and management platform for PostgreSQL
- API: Application Programming Interface

1.4. References

- PostgreSQL Official Documentation (https://www.postgresql.org/docs/)
- pgAgent Documentation (https://www.pgadmin.org/docs/pgadmin4/8.13/pgagent.html)
- PostgreSQL Logging & Auditing Best Practices (https://www.postgresql.org/docs/current/runtime-config-logging.html)
- SQL Server Agent Documentation (https://learn.microsoft.com/en-us/sql/ssms/agent/sql-server-agent)
- ISO/IEC/IEEE 26514:2022 Software Documentation Standards

2. System Overview

2.1. Existing System (pgAgent) Overview

pgAgent is a job scheduling agent for PostgreSQL databases that allows database administrators to schedule maintenance and other recurring tasks. Currently, it supports:

- Basic job scheduling with time-based triggers
- Job step execution (SQL commands or operating system scripts)
- Simple job status reporting
- Integration with pgAdmin interface for management

The agent runs as a background process and periodically checks for jobs that need to be executed based on their schedule.

2.2. Limitations of the Existing System

The current pgAgent implementation has several limitations:

- No Job Dependencies: Jobs cannot be configured to execute based on the completion status of other jobs
- Limited Auditing: Insufficient tracking of who created, modified, or executed jobs
- Basic Reporting: Limited capabilities for analyzing job history and execution patterns
- Minimal Error Handling: Limited mechanisms for handling and reporting job failures
- No Complex Workflow Support: Cannot create sophisticated job execution workflows with conditional logic

2.3. Proposed Enhancements

The proposed enhancements address these limitations through:

Job Dependency Management:

- Support for sequential job execution (Job B starts after Job A completes)
- Conditional execution based on parent job status (success/failure)
- Parallel job execution capabilities
- Prevents circular dependency

Audit Logging:

- Comprehensive logging of all job-related actions
- Tracking of job creation, modification, deletion, and execution
- User identification and timestamp recording
- Secure storage of audit logs
- Filtering capabilities for log analysis

3. Design Considerations

3.1. Assumptions and Dependencies

- The system will be built on PostgreSQL version 14 or higher
- pgAdmin 4 will be used as the primary interface
- Users have basic understanding of PostgreSQL and job scheduling concepts
- The implementation will leverage existing pgAgent tables and structures where possible
- Log storage requires additional database space allocation

3.2. Constraints

- Must maintain backward compatibility with existing pgAgent implementations
- Performance impact of dependency checking should be minimal
- Audit logging must not significantly degrade system performance
- Must adhere to PostgreSQL security standards
- Implementation must work within the existing pgAgent architecture
- Limited modifications to the pgAdmin user interface are allowed

3.3. Design Goals and Guidelines

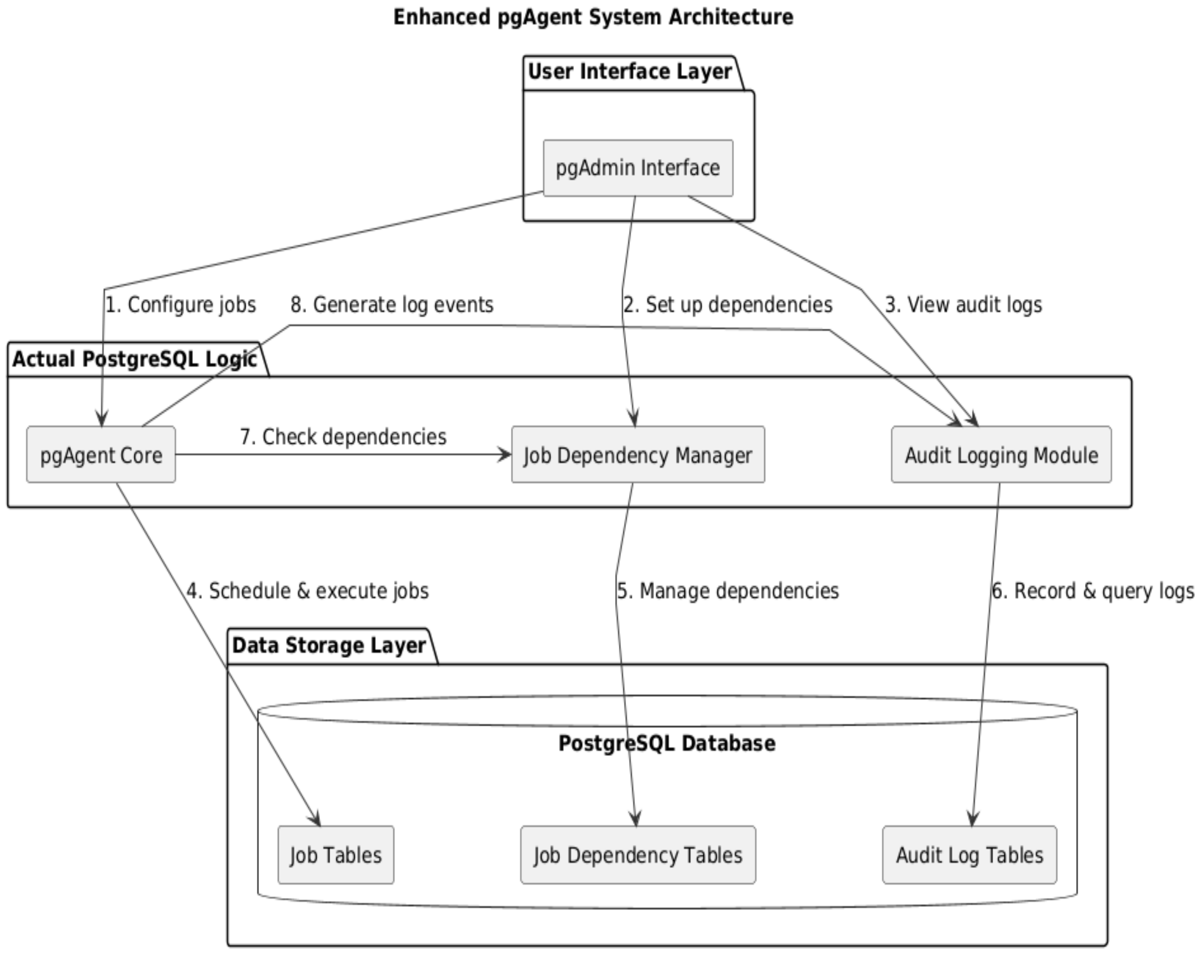
❖ Modularity: Create loosely coupled components for easier maintenance
❖ Extensibility: Design for future enhancements and additional features
❖ Usability: Provide intuitive interfaces for job dependency configuration
❖ Efficiency: Minimize performance overhead from dependency checking and logging
❖ Reliability: Ensure robust error handling and recovery mechanisms
❖ Compatibility: Maintain compatibility with existing pgAgent installations

3.4. Security Considerations

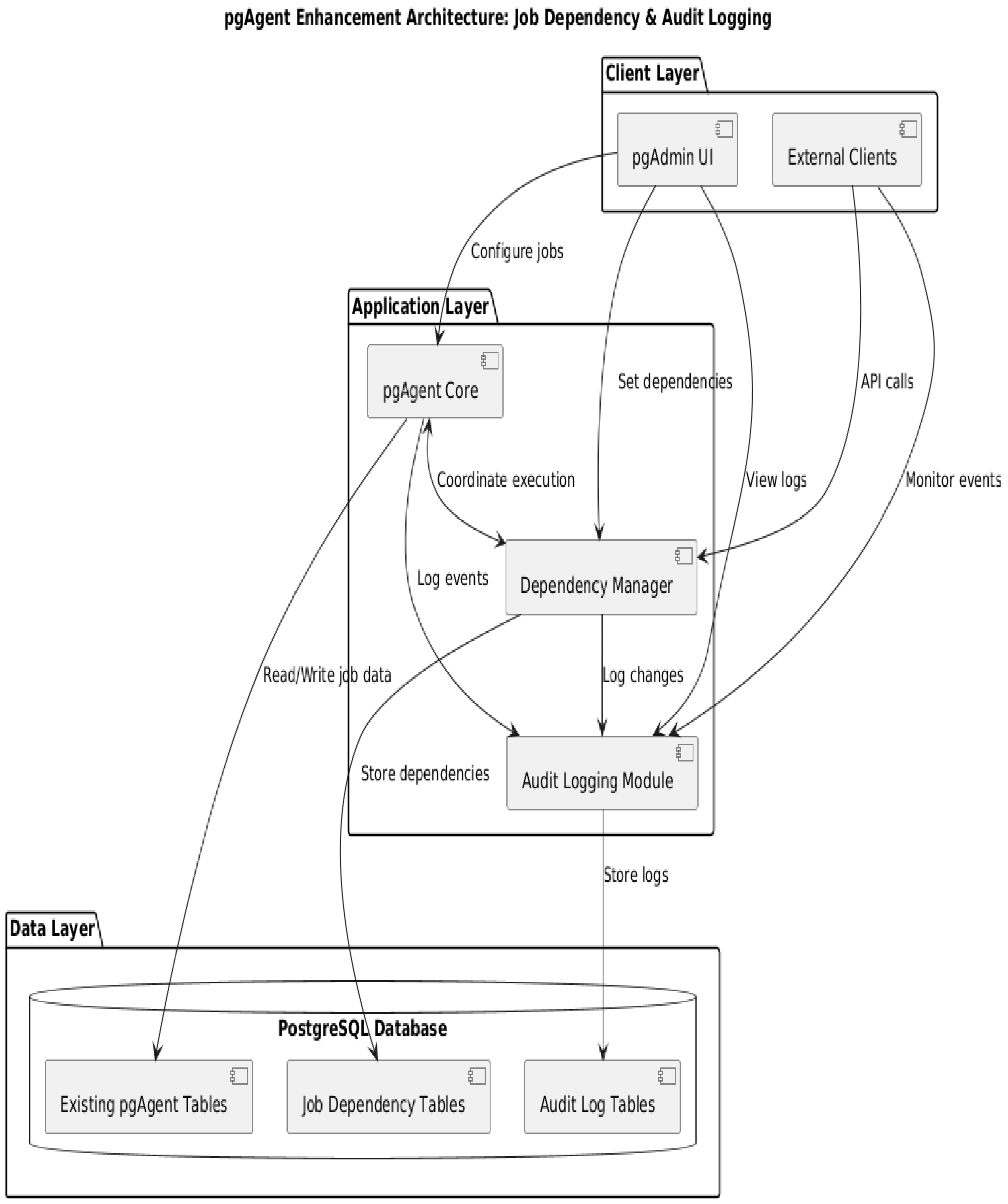
- Log Integrity: Protect audit logs from unauthorized modification
- Access Control: Restrict log access to authorized users only
- Encryption: Use TLS 1.2+ for secure communication
- Input Validation: Validate all input parameters to prevent injection attacks
- Secure Storage: Use AES-256 encryption for sensitive job details
- Auditing: Implement comprehensive logging of all security-relevant events

4. Architectural Design

4.1. High-Level Architecture Diagram



4.2. Application Architecture Design



4.3. Major Components

pgAgent Core:

- Job Scheduling Engine
- Job Execution Engine
- Status Monitoring

Job Dependency Manager:

- Dependency Rule Processor
- Job Execution Order Manager
- Dependency Visualization Component

Audit Logging Module:

- Log Capture Component
- Log Storage Manager
- Log Query Interface

Database Layer:

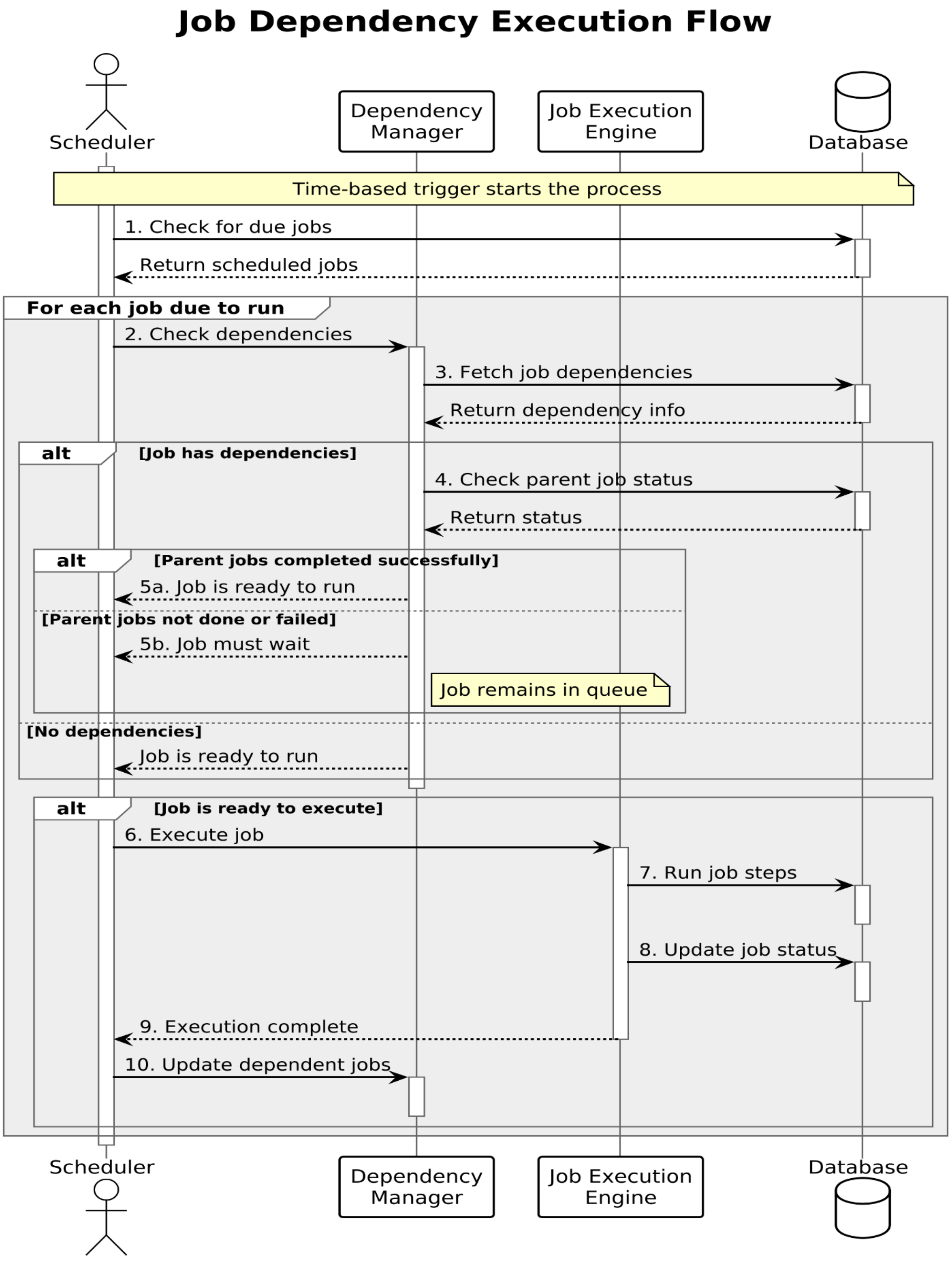
- Job Dependency Tables
- Audit Log Tables
- Job Status Tables

Interface Components:

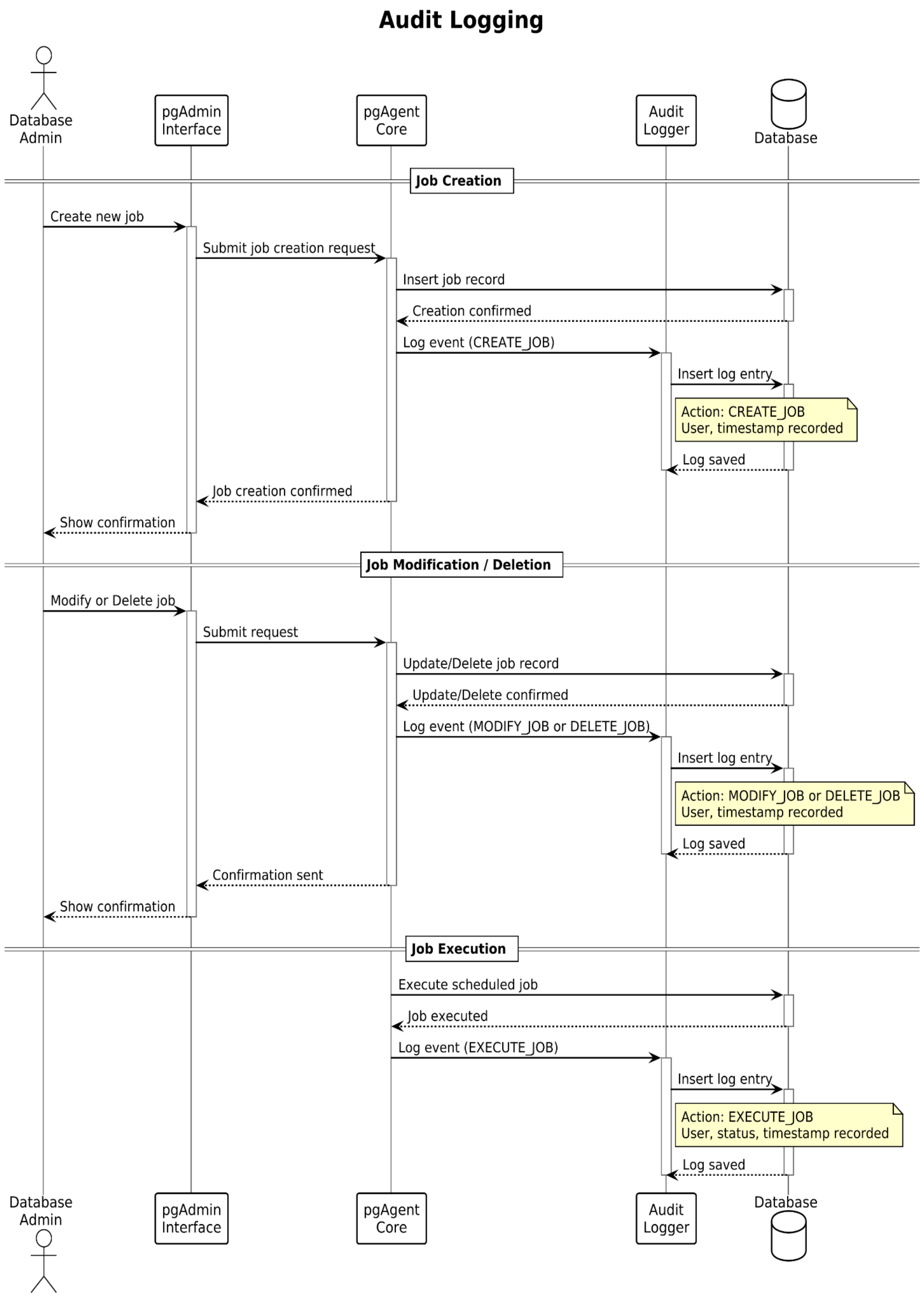
- pgAdmin Extensions for Job Dependency
- Audit Log Viewer and Filter

4.4. Sequence Diagrams

4.4.1. Job Dependency



4.4.2. Audit Logging



5. Detailed Design

5.1. Module Descriptions

5.1.1. Job Dependency Manager

Purpose: Manages the execution order of jobs based on dependency rules.

Functions:

- Identify dependent jobs for a given job
- Verify completion status of parent jobs
- Determine execution readiness based on dependency types
- Manage parallel execution of jobs where applicable
- Handle failure scenarios in dependency chains

Interfaces:

- Job status checking API
- Dependency rule configuration API
- Job execution trigger interface

5.1.2. Audit Logging Module

Purpose: Captures and manages logs of all job-related activities.

Functions:

- Intercept job creation, modification, deletion, and execution events
- Record user information, timestamps, and operation details
- Store logs securely in the database
- Provide query mechanisms for log retrieval and analysis
- Support filtering based on job name, user, operation type, and time period

Interfaces:

- Log capture API
- Log query and filter API
- Log viewer interface

5.2. Database Design

5.2.1. Job Dependency Schema

```
CREATE TABLE pgagent.pga_job_dependency (
    jobid INTEGER NOT NULL,          -- The job that has a dependency
    dependent_jobid INTEGER NOT NULL, -- The job that must complete
first
    PRIMARY KEY (jobid, dependent_jobid),  -- Composite primary key
    CONSTRAINT fk_job FOREIGN KEY (jobid) REFERENCES
pgagent.pga_job(jobid) ON DELETE CASCADE,
    CONSTRAINT fk_dependent_job FOREIGN KEY (dependent_jobid)
REFERENCES pgagent.pga_job(jobid) ON DELETE CASCADE
);
```

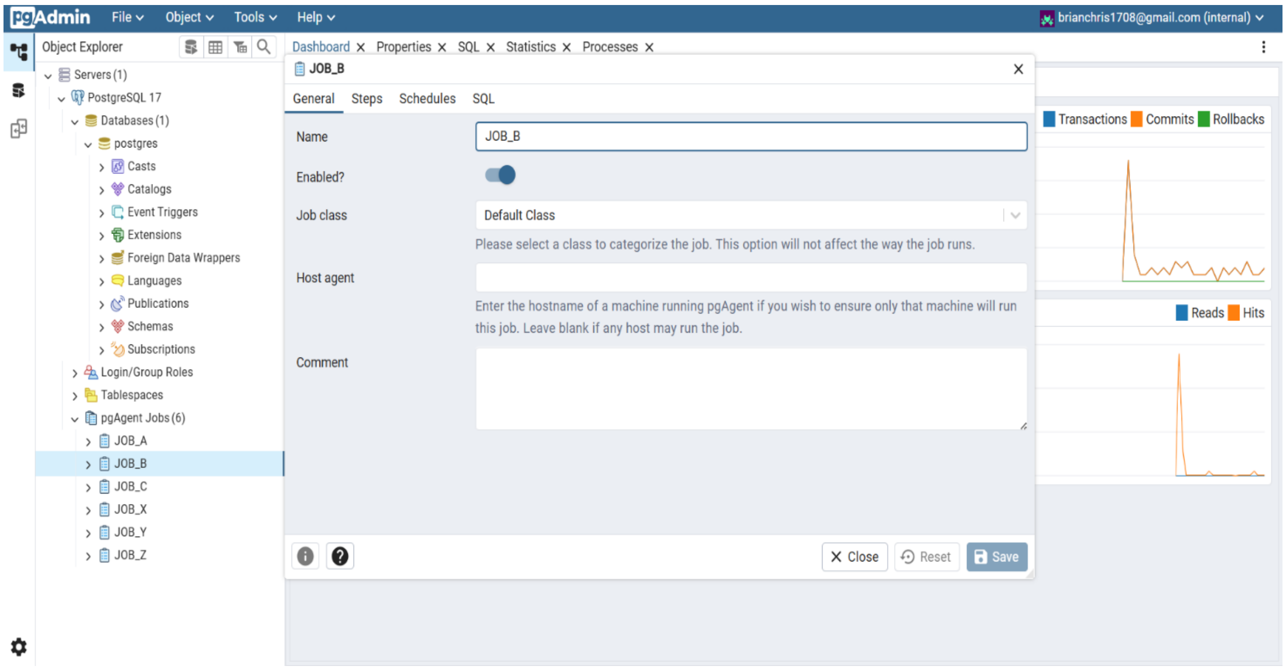
5.2.2. Audit Log Schema

```
CREATE TABLE pgagent.pga_job_audit_log (

    audit_id          serial                NOT NULL PRIMARY KEY,
    job_id            int4                  NOT NULL ,
    operation_type    text                  NOT NULL CHECK
(operation_type IN ('CREATE', 'MODIFY', 'DELETE', 'EXECUTE')),
    operation_time    timestamptz           NOT NULL DEFAULT
current_timestamp,
    operation_user    text                  NOT NULL,
    old_values        jsonb                 NULL,
    new_values        jsonb                 NULL,
    additional_info   text                  NULL
) WITHOUT OIDS;

CREATE INDEX pga_job_audit_log_jobid ON
pgagent.pga_job_audit_log(job_id);
CREATE INDEX pga_job_audit_log_operation_time ON
pgagent.pga_job_audit_log(operation_time);
```

6. User Interface Design

6.1. UI Mock-ups

- pgAdmin before(No job dependency tab)



JOB_B

Name: JOB_B

Enabled?

Job class: Default Class

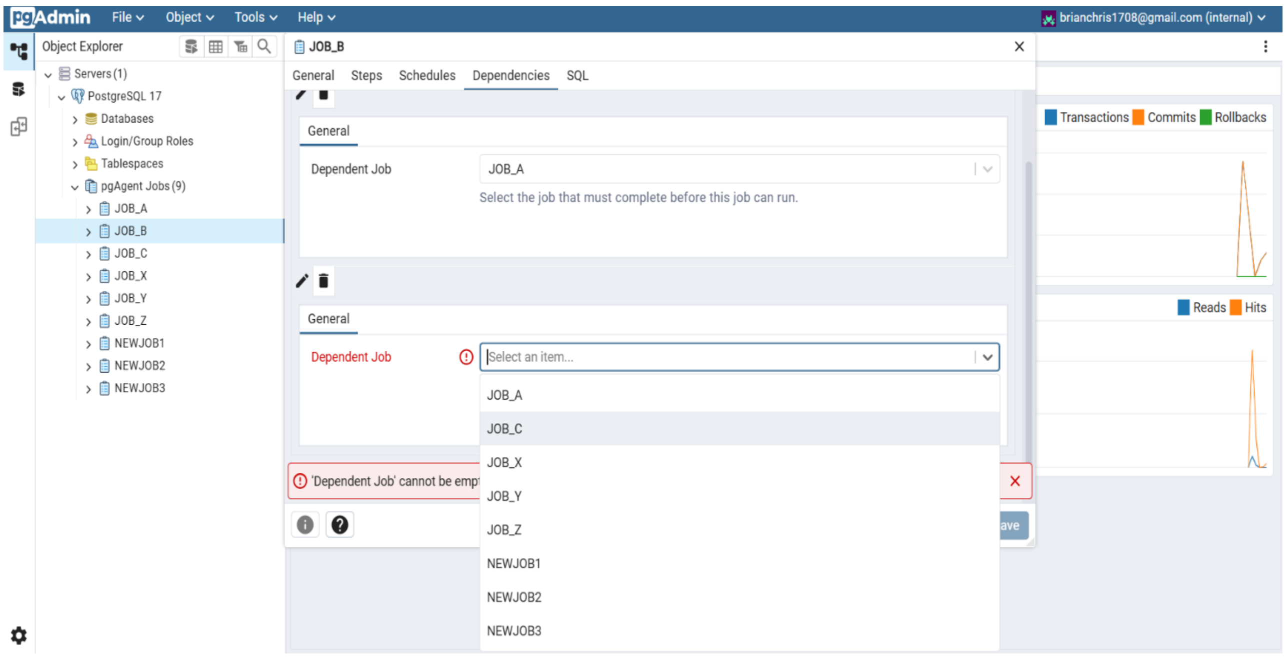
Please select a class to categorize the job. This option will not affect the way the job runs.

Host agent

Enter the hostname of a machine running pgAgent if you wish to ensure only that machine will run this job. Leave blank if any host may run the job.

Comment

- pgAdmin after implementing Job dependency and integrating with existing UI



JOB_B

General

Dependent Job: JOB_A

Select the job that must complete before this job can run.

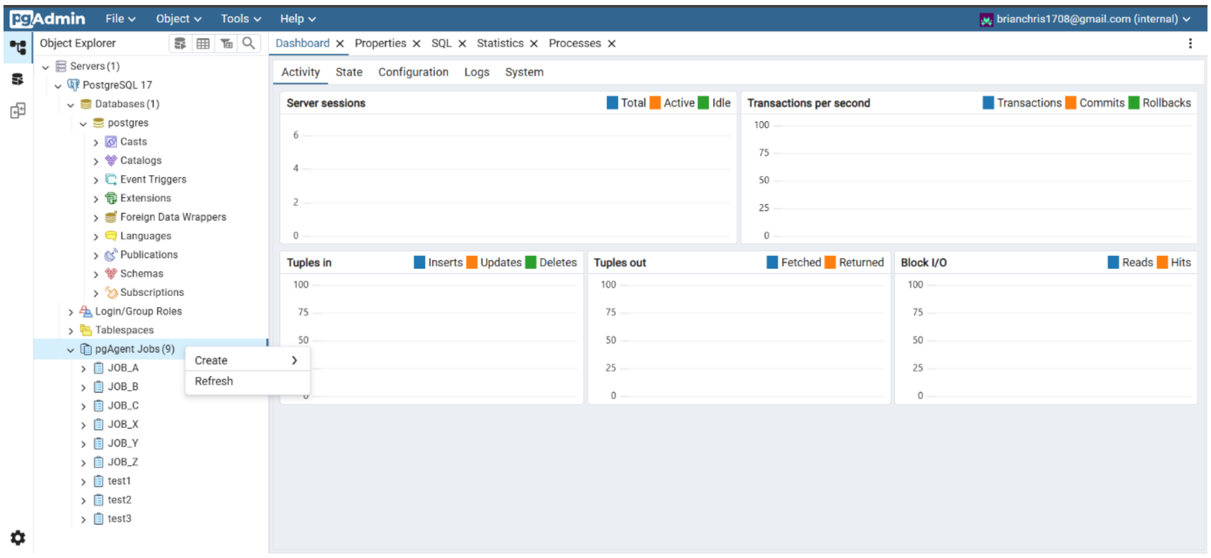
General

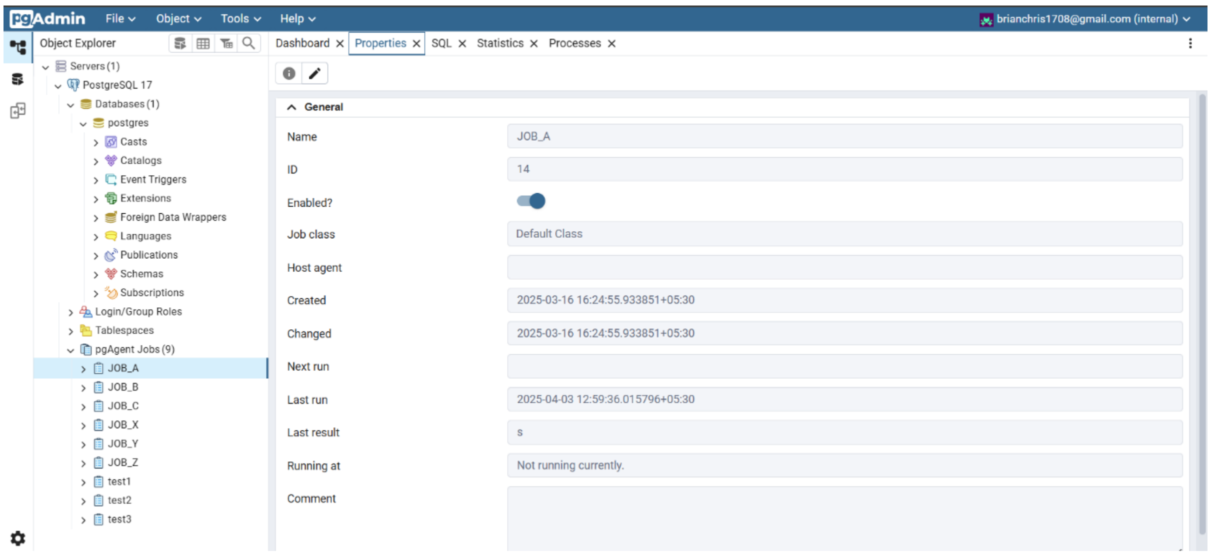
Dependent Job: Select an item...

JOB_A
JOB_C
JOB_X
JOB_Y
JOB_Z
NEWJOB1
NEWJOB2
NEWJOB3

'Dependent Job' cannot be empt

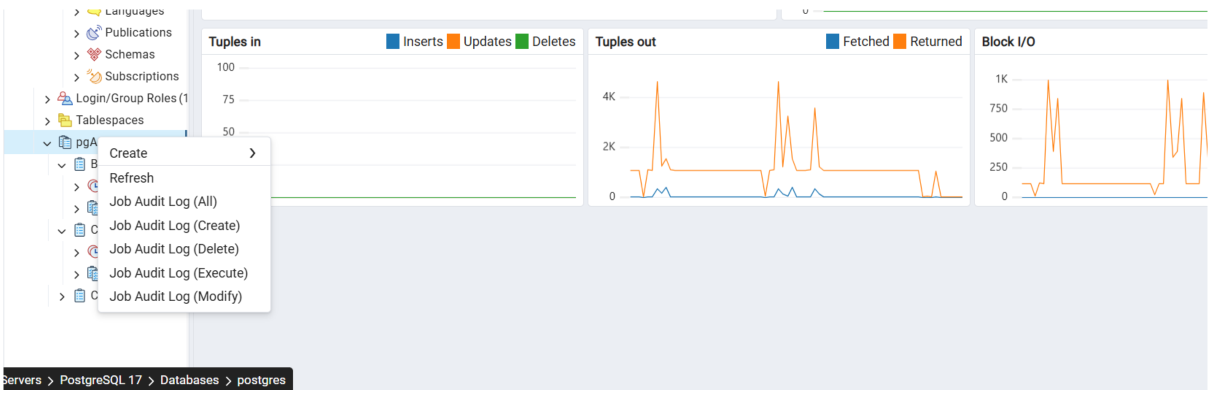
- pgAdmin before(No audit logging)

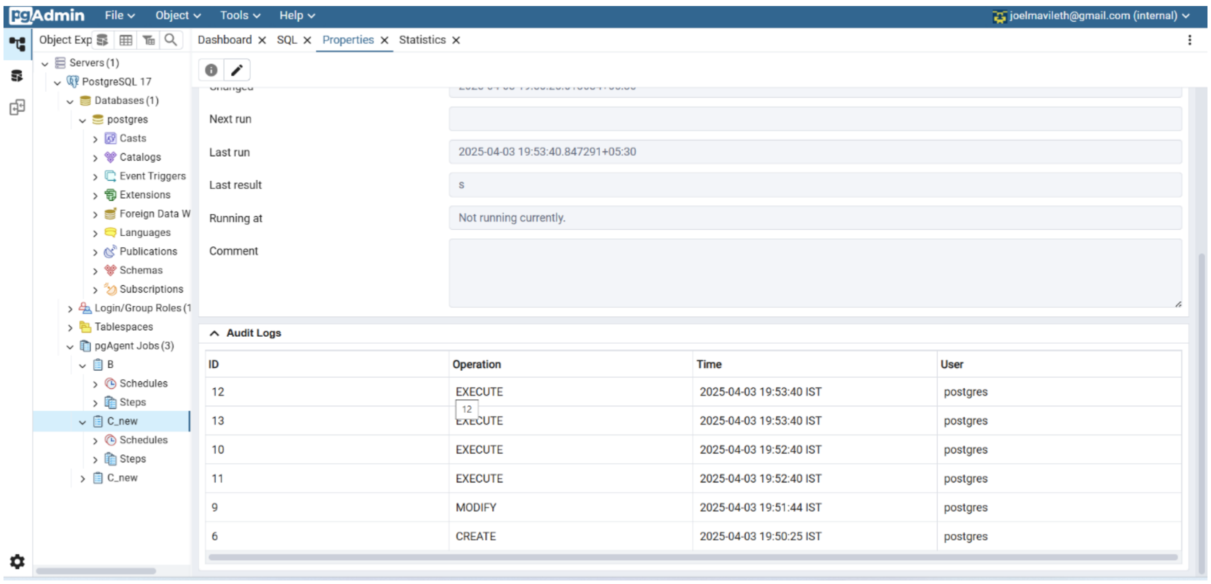




Name	JOB_A
ID	14
Enabled?	●
Job class	Default Class
Host agent	
Created	2025-03-16 16:24:55.933851+05:30
Changed	2025-03-16 16:24:55.933851+05:30
Next run	
Last run	2025-04-03 12:59:36.015796+05:30
Last result	s
Running at	Not running currently.
Comment	

- pgAdmin after implementing audit logging





Next run

Last run 2025-04-03 19:53:40.847291+05:30

Last result s

Running at Not running currently.

Comment

Audit Logs

ID	Operation	Time	User
12	EXECUTE	2025-04-03 19:53:40 IST	postgres
13	EXECUTE	2025-04-03 19:53:40 IST	postgres
10	EXECUTE	2025-04-03 19:52:40 IST	postgres
11	EXECUTE	2025-04-03 19:52:40 IST	postgres
9	MODIFY	2025-04-03 19:51:44 IST	postgres
6	CREATE	2025-04-03 19:50:25 IST	postgres

6.2. User Interaction Flow

6.2.1. Job Dependency Configuration Flow

✓ User selects a job in pgAdmin
✓ User accesses the "Dependencies" tab
✓ User clicks "Add Dependency" and selects a parent job
✓ System validates the dependency
✓ User saves the configuration
✓ System records the action in the audit log

6.2.2. Audit Log Viewing Flow

✓ User selects a job in pgAdmin
✓ User acceses the “Audit Logs” Tab.
✓ System displays matching log entries for the job in a tabular format
✓ User right clicks on PgAgent Jobs in PgAdmin UI.
✓ Shows menu on all audit logs, and audit logs filtered by operation type.
✓ When any one is clicked it opens up the Query Tool with the required query copied to user’s clipboard
✓ User can paste the query and execute in order to view the logs.

6.3. Technologies Used

Frontend: JavaScript, HTML5, CSS3, pgAdmin UI Framework

Backend: PL/pgSQL, C (for pgAgent core extensions)

Database: PostgreSQL 14+ native features

6.4. Customer Use Case

Use Case I: Database Maintenance Workflow

Scenario: A database administrator needs to set up a maintenance workflow with multiple dependent jobs.

Steps:

1. Admin creates the following jobs:
   - Job A: Backup database
   - Job B: Analyze tables
   - Job C: Vacuum database
   - Job D: Update statistics
2. Admin configures dependencies:
   - Job B depends on Job A (sequential, condition: success)
   - Job C depends on Job B (sequential, condition: success)
   - Job D depends on Job C (sequential, condition: success)
3. Admin schedules Job A to run nightly
4. System executes the jobs in sequence based on dependencies
5. Admin reviews audit logs the next morning to verify successful execution
6. If any job fails, admin can see the exact point of failure and subsequent jobs that didn't run

Use Case II: Daily Backup and Cleanup

Scenario: A small business needs to automate their daily database backup and cleanup process.

Steps:

1. The admin creates two simple jobs:
   - Job A: Database backup
   - Job B: Delete temporary files and old log files
2. Dependencies:
   - Job B depends on Job A (sequential, condition: success)
   - This ensures cleanup only happens after a successful backup

3. The admin checks the audit logs each morning to verify the backup completed successfully without having to manually check the backup files.

Use Case III: Student Grade Processing

Scenario: A school needs to process grades at the end of each semester.

Steps:

1. The administrator sets up:
    - Job A: Calculate final grades
    - Job B: Generate grade reports
    - Job C: Archive semester data
2. Dependencies:
    - Job B depends on Job A (sequential, condition: success)
    - Job C depends on Job B (sequential, condition: success)
3. The audit logging provides a record of when grades were calculated and by whom, helpful for addressing student questions.

7. Testing Strategy

7.1. Unit Testing

- Test job dependency rule validation
- Test dependency cycle detection
- Test condition evaluation logic
- Test audit log capture functions
- Test log retrieval with various filters
- Test boundary conditions for job dependencies

7.2. Integration Testing

- Test end-to-end job dependency workflows
- Test interaction between pgAgent and the dependency manager
- Test audit logging during various operations
- Test querying the logs based on operation type, user ,etc.
- Test concurrent job execution with dependencies
- Test system behaviour under heavy load

7.3. Manual Testing / Test Cases

Test ID	Description	Expected Result
TC – 1	Create a job with a dependency on another job	Dependency is created and visible in the UI
TC – 2	Execute a job with a successful parent dependency	Job executes after parent completes successfully
TC – 3	Execute a job with a failed parent dependency	Job doesn't execute when parent fails
TC – 4	Modify a job and verify audit log	Log entry shows the modification with correct user and details
TC – 5	Delete a job with dependencies	System handles orphaned dependencies appropriately
TC – 6	Filter audit logs by date range and operation type	Only matching logs are displayed

8. Future Scope and Recommendations

Advanced Dependency Types:

- Time-based dependencies (e.g., execute Job B at least 1 hour after Job A)
- Resource availability dependencies (e.g., execute only when database load is below threshold)

Enhanced Visualization:

- Interactive dependency graphs with real-time status updates
- Timeline-based visualization of job execution history

Predictive Analytics:

- Job execution time prediction based on historical data
- Failure prediction and preventive measures

Advanced Logging Features:

- Configurable log retention policies
- Automated log analysis for pattern detection
- Real-time alerts based on log patterns

Workflow Templates:

- Reusable workflow templates for common job patterns
- Import/export functionality for job dependency configurations

9. Appendix

9.1. Glossary

Job: A scheduled task in pgAgent

Job Step: A specific action within a job (SQL or batch/shell script)

Dependency: A relationship between jobs that determines execution order

Sequential Execution: Running jobs one after another

Conditional Execution: Running a job only if certain conditions are met

Parallel Execution: Running multiple jobs simultaneously

Audit Trail: A chronological record of activities

9.2. Code Snippets or Logs

Job Dependency Core Logic

```
bool Job::CheckDependencies()
{
    LogMessage("Checking dependencies for job: " + m_jobid, LOG_DEBUG);
    // Fetch all job dependencies
    DBresultPtr res = m_threadConn->Execute(
        "SELECT dependent_jobid FROM pgagent.pga_job_dependency WHERE jobid =
" + m_jobid);

    if (!res || res->RowsAffected() == 0)
    {
        LogMessage("No dependencies found for job: " + m_jobid, LOG_DEBUG);
        return true; // No dependencies, so the job can run
    }
    while (res->HasData())
    {
        std::string depJobId = res->GetString("dependent_jobid");
        // Check if the dependent job completed successfully
        DBresultPtr depRes = m_threadConn->Execute(
            "SELECT jlgstatus FROM pgagent.pga_joblog "
            "WHERE jlgjobid = " +
            depJobId +
            " ORDER BY jlgstart DESC LIMIT 1");

        if (!depRes || depRes->RowsAffected() == 0)
        {
            LogMessage("Dependency job " + depJobId + " has no execution logs.
Cannot proceed.", LOG_WARNING);
```

```
            return false;
        }
        std::string depStatus = depRes->GetString("jlgstatus");
        if (depStatus != "s") // 's' means success
        {
            LogMessage("Dependency job " + depJobId + " did not complete
successfully. Status: " + depStatus, LOG_WARNING);
            return false;
        }
        res->MoveNext();
    }
    LogMessage("All dependencies satisfied for job: " + m_jobid, LOG_DEBUG);
    return true;
}
```

Audit Logging Core Logic

```
if (rc == 1)
    {
        // Get current user
        std::string currentUser = m_threadConn->ExecuteScalar("SELECT
current_user");
        // Get job state before update
        DBresultPtr oldState = m_threadConn->Execute(
            "SELECT row_to_json(j)::jsonb as job_state FROM pgagent.pga_job j
WHERE j.jobid=" + m_jobid
        );
        std::string jobState = oldState ? m_threadConn->qtDbString(oldState-
>GetString("job_state")) : "NULL";
        // Log job execution start
        std::string auditQuery = "SELECT pgagent.pga_log_job_operation(" +
m_jobid +
                                 "', 'EXECUTE', " + m_threadConn-
>qtDbString(currentUser) +
                                 "', " + jobState + ", NULL, 'Job execution
started')";
        m_threadConn->ExecuteVoid(auditQuery);
        // Retrieve job log ID
        DBresultPtr id = m_threadConn->Execute(
            "INSERT INTO pgagent.pga_joblog(jlgjobid, jlgstatus) VALUES (" +
m_jobid + ", 'r') RETURNING jlgid"
        );
        if (id && id->RowsAffected() > 0)
        {
            m_logid = id->GetString("jlgid");
            m_status = "r";
        }
    }
```

```
-- Function to log job operations
CREATE OR REPLACE FUNCTION pgagent.pga_log_job_operation(
    p_job_id integer,
    p_operation_type text,
    p_operation_user text,
    p_old_values jsonb,
    p_new_values jsonb,
    p_additional_info text
) RETURNS void AS $$
BEGIN
    INSERT INTO pgagent.pga_job_audit_log (
        job_id,
        operation_type,
        operation_user,
        old_values,
        new_values,
        additional_info
    ) VALUES (
        p_job_id,
        p_operation_type,
        p_operation_user,
        p_old_values,
        p_new_values,
        p_additional_info
    );
END;
$$ LANGUAGE plpgsql;

-- Trigger function for job modifications
CREATE OR REPLACE FUNCTION pgagent.pga_job_audit_trigger()
RETURNS trigger AS $$
DECLARE
    significant_change boolean;
BEGIN
    significant_change := false;

    IF TG_OP = 'INSERT' THEN
        -- Always log job creation
        PERFORM pgagent.pga_log_job_operation(
            NEW.jobid,
            'CREATE',
            current_user,
            NULL,
            row_to_json(NEW)::jsonb,
            NULL
        );
        RETURN NULL;
    ELSIF TG_OP = 'UPDATE' THEN
```

```
        -- Only log significant changes
        IF (OLD.jobname != NEW.jobname) OR
           (OLD.jobdesc != NEW.jobdesc) OR
           (OLD.jobhostagent != NEW.jobhostagent) OR
           (OLD.jobenabled != NEW.jobenabled) OR
           (OLD.jobjclid != NEW.jobjclid) THEN
            significant_change := true;
        END IF;

        -- Don't log changes to jobagentid, joblastrun, jobnextrun as they are
internal state changes
        IF significant_change THEN
            PERFORM pgagent.pga_log_job_operation(
                NEW.jobid,
                'MODIFY',
                current_user,
                row_to_json(OLD)::jsonb,
                row_to_json(NEW)::jsonb,
                NULL
            );
        END IF;
        RETURN NULL;
    ELSIF TG_OP = 'DELETE' THEN
        -- Log the deletion after the job is actually deleted
        PERFORM pgagent.pga_log_job_operation(
            OLD.jobid,
            'DELETE',
            current_user,
            row_to_json(OLD)::jsonb,
            NULL,
            'Job deleted by user ' || current_user
        );
        RETURN NULL;
    END IF;
    RETURN NULL;
END;
$$ LANGUAGE plpgsql;

-- Create trigger for job table
DROP TRIGGER IF EXISTS pga_job_audit_trigger ON pgagent.pga_job;
CREATE TRIGGER pga_job_audit_trigger
    AFTER INSERT OR UPDATE OR DELETE ON pgagent.pga_job
    FOR EACH ROW
    EXECUTE FUNCTION pgagent.pga_job_audit_trigger();
```