AI-Driven Autonomous Database Management System (ADBMS)

# Research Document

## Abstract

This document presents the design and implementation of an AI-driven Autonomous Database Management System (ADBMS) that leverages intelligent indexing, predictive query optimization, self-tuning, and security modules. The system aims to automate database management tasks, improve performance, and enhance security using machine learning and data-driven techniques.

## 1. Introduction

Modern databases require continuous tuning and monitoring to maintain optimal performance and security. Manual management is time-consuming and error-prone. The ADBMS project addresses these challenges by integrating AI techniques into core database management functions, enabling autonomous operation.

## 2. System Architecture

The ADBMS is organized into modular components, each responsible for a key aspect of autonomous database management:

- Intelligent Indexing: Dynamically monitors data access patterns and recommends or applies indexes to optimize query performance.

- Predictive Query Optimization: Analyzes incoming queries and predicts optimal execution plans using machine learning.

- Self-Tuning Engine: Continuously monitors system metrics and tunes database parameters for optimal resource utilization.

- Security & Compliance Layer: Detects anomalous activities and enforces compliance policies.

Project Structure:

docs/  
 architecture.md  
requirements.txt  
src/  
 config.py  
 intelligent\_indexing/  
 index\_manager.py  
 main.py  
 query\_optimization/  
 main.py  
 query\_optimizer.py  
 security/  
 anomaly\_detector.py  
 main.py  
 self\_tuning/  
 main.py  
 parameter\_tuner.py  
 workload\_monitor.py  
tests/  
 integration/  
 test\_intelligent\_indexing.py  
 test\_placeholder.py  
 test\_query\_optimization.py  
 test\_security.py  
 test\_self\_tuning.py  
README.md

## 3. Module Descriptions

### 3.1 Intelligent Indexing

Purpose: Automates index management by monitoring access patterns and recommending or applying indexes.

Implementation: Uses the IndexManager class to track column usage and manage index creation.

### 3.2 Predictive Query Optimization

Purpose: Predicts the most efficient query execution plan using features extracted from SQL queries.

Implementation: The QueryOptimizer class analyzes queries and applies machine learning models to select plans.

### 3.3 Self-Tuning Engine

Purpose: Monitors workload and system metrics, adjusting parameters such as buffer size and cache policy.

Implementation: Combines WorkloadMonitor and ParameterTuner to automate tuning cycles.

### 3.4 Security & Compliance Layer

Purpose: Detects and responds to anomalous or potentially malicious activities.

Implementation: The AnomalyDetector class inspects logs and flags suspicious actions.

## 4. Setup and Usage

### 4.1 Prerequisites

- Python 3.x

- Required packages (see requirements.txt): scikit-learn, pandas, numpy, tensorflow, psycopg2-binary, mysql-connector-python, pymongo

### 4.2 Installation

git clone https://github.com/janakachinthana/ai-autonomous-dbms.git

cd ai-autonomous-dbms

python -m venv venv

.\venv\Scripts\activate

pip install -r requirements.txt

### 4.3 Running Modules

Run each module from the project root:

python src/intelligent\_indexing/main.py

python src/query\_optimization/main.py

python src/security/main.py

python src/self\_tuning/main.py

## 5. Testing

Integration tests are provided in the tests/integration/ directory for each module.

## 6. Conclusion

The ADBMS project demonstrates the feasibility and benefits of integrating AI into database management. By automating indexing, query optimization, tuning, and security, the system reduces manual intervention and improves overall database performance and reliability.

## 7. References

Project documentation: docs/architecture.md

Source code: https://github.com/janakachinthana/ai-autonomous-dbms