Estudo de Caso Sobre o Uso de Agentes de Inteligência Artificial na Otimização de Consultas e Recomendação de Índices em Bancos de Dados PostgreSQL*

Rávilla N. S. Moreira[1], **Danilo Boechat Seufitelli**[1]

[1]Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG)
Departamento de Computação (DECOM)
Belo Horizonte, MG – Brasil

ravillanoely@gmail.com, danilobs@cefetmg.br

Abstract. *This paper proposes and evaluates an Artificial Intelligence multi-agent system to automate performance analysis, query optimization, and index recommendation in databases. The objective is to assess the reliability of the recommendations and the reduction of manual intervention by specialists (DBAs and DBREs). The solution integrates a Python application with the CrewAI framework. The evaluation runs in PostgreSQL environments using synthetic queries and anonymized real data. The results confirm the tool's ability to identify performance bottlenecks and suggest suitable indexing strategies. This research contributes an automated approach to support decision-making. The proposed solution reduces the operational effort required for database maintenance.*

Resumo. *Este trabalho propõe e avalia um sistema multiagente de Inteligência Artificial para automatizar a análise de desempenho, a otimização de consultas e a recomendação de índices em bancos de dados. O objetivo é aferir a confiabilidade das recomendações e a redução da intervenção manual de especialistas (DBAs e DBREs). A solução integra uma aplicação Python ao framework CrewAI. A validação ocorre em ambientes PostgreSQL com consultas sintéticas e dados reais anonimizados. Os resultados atestam a capacidade da ferramenta para identificar gargalos e sugerir indexações adequadas. A pesquisa contribui com uma abordagem automatizada de apoio à tomada de decisão. A proposta diminui o esforço operacional na manutenção de bancos de dados.*

1. Introdução

O desempenho de sistemas de bancos de dados relacionais está diretamente ligado à eficiência das consultas SQL executadas. Em ambientes com alto volume de dados, consultas mal estruturadas geram gargalos significativos que comprometem o tempo de resposta e o consumo de recursos computacionais [Amaral et al. 2025]. Tradicionalmente, a detecção e a correção desses problemas dependem da intervenção manual de profissionais especializados, como *Database Administrators* (DBAs) ou *Database Reliability Engineers* (DBREs). Esses profissionais analisam planos de execução e aplicam estratégias de ajuste (*tuning*) [Sharma et al. 2018]. No entanto, esse processo é frequentemente lento e

*Este trabalho é resultado do Trabalho de Conclusão de Curso que será apresentado ao curso de Engenharia de Computação do CEFET-MG. Os autores declaram que ferramentas de Inteligência Artificial foram utilizadas como auxílio na revisão linguística e na estruturação de trechos do texto.

propenso a falhas em ambientes complexos. Esse cenário impulsiona a transição de abordagens rígidas baseadas em regras para métodos fundamentados em *Machine Learning*, *Reinforcement Learning* e *Large Language Models* (LLMs) [Mozaffari et al. 2024].

Nesse cenário, a Inteligência Artificial (IA) surge como um caminho promissor para automatizar tarefas de administração ao reduzir o esforço operacional e permitir a otimização em tempo real [Chaudhari 2025]. Pesquisas demonstram que técnicas de *Deep Reinforcement Learning* podem substituir heurísticas clássicas em desafios como a enumeração da ordem de junções (*join order*) [Marcus and Papaemmanouil 2018]. Por outro lado, há sistemas baseados em LLMs (e.g. GPTuner) capazes de interpretar documentações técnicas para acelerar o ajuste de parâmetros de configuração [Lao et al. 2024]. Aplicar IA para otimizar consultas visa melhorar o desempenho e reduzir drasticamente os custos computacionais da execução de SQL [Amaral et al. 2025].

Este trabalho investiga a eficácia de um sistema multiagente de IA no apoio à análise e à otimização de consultas em bancos de dados **PostgreSQL**. O estado da arte em sistemas autônomos nessa área ainda demanda validações experimentais práticas e sistematizadas [Mozaffari et al. 2024]. Para tanto, a pesquisa desenvolveu uma aplicação capaz de coletar métricas de desempenho e de analisar planos de execução para gerar recomendações automáticas. O objetivo é aferir o potencial desta ferramenta como apoio à tomada de decisão técnica. Essa proposta alinha-se à tendência atual de criação de sistemas gerenciadores de bancos de dados mais inteligentes e independentes de intervenção humana constante [Marcus and Papaemmanouil 2018].

2. Trabalhos Relacionados

A literatura sobre administração de bancos de dados busca diminuir a dependência de especialistas humanos por meio de automação e Inteligência Artificial. Estudos pioneiros, como o de Morelli et al. (2009), que estabeleceram a viabilidade de mecanismos autônomos como a reindexação baseada no monitoramento da fragmentação.

Pesquisas recentes exploram três frentes principais: sistemas autoajustáveis, IA aplicada à otimização de consultas e análise de desempenho. Na primeira categoria, destacam-se o uso de *Deep Reinforcement Learning* para decisões administrativas [Marcus and Papaemmanouil 2018] e revisões sobre técnicas de autoajuste em sistemas SQL e NoSQL [Mozaffari et al. 2024]. Na otimização de consultas e *tuning* autônomo, há abordagens que utilizam modelos de linguagem, otimização bayesiana e sistemas multiagentes para recomendar melhorias [Lao et al. 2024, Amaral et al. 2025, Oliveira et al. 2022, Petrola et al. 2025]. Especificamente, o uso de agentes inteligentes para a geração e correção de consultas *Text-to-SQL* tem demonstrado ganhos em tempo de execução [Petrola et al. 2025]. Além disso, o sistema NoDBA exemplifica os benefícios da automação prática na administração de bancos de dados [Sharma et al. 2018].

A maioria dessas propostas foca em abordagens de caixa preta ou ajustes globais de servidor. Sistemas como o GPTuner limitam-se ao ajuste de parâmetros de configuração, enquanto o NoDBA opera sem supervisão humana direta. Este trabalho diferencia-se ao propor um sistema multiagente focado especificamente na análise de planos de execução e na recomendação de índices para PostgreSQL. A abordagem utiliza LLMs para fornecer suporte explicável à decisão, mantendo o especialista humano no controle das ações e priorizando a validação da confiabilidade das recomendações.

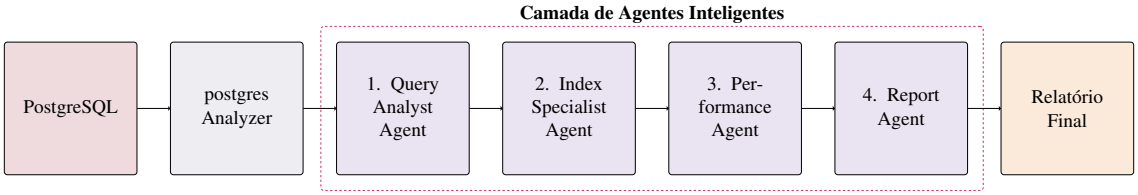

Figura 1. Arquitetura do sistema multi-agente para otimização de consultas

3. Metodologia

Este trabalho caracteriza-se como pesquisa aplicada e experimental. O foco é desenvolver e avaliar um sistema multiagente para apoio à análise de desempenho em bancos de dados PostgreSQL. Para isso, foi desenvolvida uma aplicação em Python para coletar métricas, analisar desempenho e gerar automaticamente recomendações de otimização por meio de agentes especializados. Cada agente desempenha uma função específica, cujas responsabilidades incluem a coleta de consultas de maior impacto, a análise estrutural das *queries*, a verificação da existência e adequação de índices e a interpretação dos planos de execução gerados pelo banco de dados. Essas etapas são realizadas de forma encadeada, permitindo a consolidação dos resultados em um relatório estruturado.

Os experimentos ocorreram em ambientes PostgreSQL com diferentes cenários de carga e de volume de dados. Essa abordagem possibilita a observação do sistema em diferentes contextos. Os testes utilizaram consultas sintéticas e dados reais provenientes de um ambiente produtivo. A pesquisa aplicou técnicas de anonimização nos dados reais para preservar a confidencialidade da origem. O estudo analisou consultas representativas de diferentes níveis de complexidade. A análise utilizou métricas como tempo de execução, número de execuções e plano de execução gerado pelo otimizador. A partir dessas análises, o sistema gerou recomendações de otimização. O estudo utilizou essas sugestões para medir a capacidade da ferramenta de identificar padrões de ineficiência. O objetivo é validar o apoio do sistema ao processo de diagnóstico de desempenho.

3.1. Desenvolvimento dos agentes

A arquitetura do *postgresAnalyzer* baseia-se no paradigma de Sistemas Multiagentes (MAS), no qual múltiplos agentes especializados colaboram para produzir análises mais completas. Essa estratégia supera a capacidade de um único agente genérico [Wooldridge 2009]. O framework CrewAI realiza a orquestração desses agentes e oferece suporte à definição de equipes (*crews*). A ferramenta também inclui o encadeamento de tarefas com dependências explícitas e o compartilhamento de contexto entre agentes [CrewAI Inc. 2026]. Como base cognitiva dos agentes, foi empregado o modelo Claude 3.5 Sonnet [ANTHROPIC 2026]. A Figura 1 ilustra a visão geral da arquitetura. O sistema é composto por quatro agentes, cada um com responsabilidade bem definida no fluxo de análise, conforme detalhado a seguir.

1. Query Analyst Agent. Atua como ponto de entrada do processo, utilizando a extensão `pg_stat_statements` para coletar consultas com maior tempo total de execução. Além disso, realiza a análise estrutural das consultas, identificando tabelas e colunas por meio de um *parser* SQL baseado na biblioteca `sqlparse`.

2. Index Specialist Agent. Recebe as informações estruturais e verifica, para cada coluna relevante, a existência de índices e a cardinalidade dos atributos. Com base nesses dados,

avalia a necessidade de criação de novos índices, gerando recomendações utilizando o comando `CREATE INDEX CONCURRENTLY` quando aplicável.

3. Performance Agent. executa o comando `EXPLAIN ANALYZE` nas consultas identificadas, obtendo o plano de execução real. A partir dessas informações, identifica operações custosas (ex. *Sequential Scans*) e produz uma análise com problemas de desempenho.

4. Report Agent. Consolida os resultados produzidos pelos demais agentes em um relatório estruturado. O relatório contém um sumário executivo, análises detalhadas, recomendações priorizadas e comandos SQL sugeridos.

3.2. Configuração do ambiente experimental

O ambiente experimental é composto por instâncias *PostgreSQL Flexible Server* provisionadas na plataforma Microsoft Azure. As instâncias são configuradas em dois perfis distintos (*Burstable* e *General Purpose*) para avaliar o comportamento do sistema sob diferentes cargas de trabalho. As especificações detalhadas de cada ambiente são apresentadas na Tabela 1.

Métrica	**Ambiente 1 (Burstable)**	**Ambiente 2 (General Purpose)**
Plano Azure	Burstable	General Purpose
Instância	B1ms	D4s_v3
vCPUs	1	4
Memória	2 GiB	16 GiB
Armazenamento	32 GiB	128 GiB
Tipo de workload	Baixa carga	Alta carga
Objetivo	Teste leve	Teste intensivo

Tabela 1. Configuração dos ambientes experimentais utilizados nos testes

A comunicação entre a aplicação e o banco de dados foi realizada por meio das bibliotecas `psycopg2`, `SQLAlchemy` e `azure-identity`. A aplicação *postgresAnalyzer* foi executada em ambiente local com processador Intel Core i7-1355U, 32 GB de memória RAM e sistema operacional Windows 11 Enterprise 64 bits.

3.3. Execução dos experimentos

Para cada cenário, foram executadas consultas SQL representativas de diferentes padrões de acesso e níveis de complexidade. As métricas coletadas incluem o tempo de execução, o número de execuções e os planos de execução gerados pelo otimizador do PostgreSQL. A partir desses dados, os agentes geraram recomendações de otimização. Tais recomendações permitem avaliar sua capacidade de identificar gargalos de desempenho e sugerir melhorias. No entanto, as sugestões não foram aplicadas automaticamente e serviram apenas para análise da efetividade do sistema.

A validação prática das recomendações de criação de índices foi realizada apenas no Ambiente 1. Este é um ambiente de *staging* e permite alterações sem impacto em produção. O Ambiente 2 tem caráter produtivo; por isso foi utilizado somente para coleta de métricas e geração dos relatórios. No Ambiente 1, foram avaliados os dezoito índices sugeridos pelo sistema, e as recomendações foram classificadas como válidas ou espúrias para embasar o cálculo de precisão da ferramenta. Em seguida, o autor implementou doze índices válidos. O estudo realizou uma nova coleta de métricas com o comando

EXPLAIN ANALYZE. O objetivo foi comparar os resultados e mensurar o impacto direto das recomendações na latência das consultas.

Para avaliar o valor adicionado pelos modelos de linguagem, definiu-se uma linha de base (*baseline*). Para isso, compara-se o sistema multiagente com uma heurística determinística simples. A regra heurística recomendou índices para todas as colunas presentes nas cláusulas `WHERE` e `JOIN` das consultas com maior tempo de execução. Essa comparação evidencia a capacidade do sistema inteligente em evitar recomendações espúrias e em avaliar a cardinalidade dos dados antes da criação dos índices.

3.4. Ferramentas e tecnologias utilizadas

A condução dos experimentos e o desenvolvimento da aplicação demandam um conjunto específico de ferramentas. A escolha da linguagem Python considera sua ampla adoção e integração nativa com bibliotecas de Inteligência Artificial. A plataforma `Azure Flexible Server` hospeda o banco de dados PostgreSQL. Essa escolha reproduz um ambiente escalável e fiel a cenários de produção. O *framework* `CrewAI` orquestra os agentes e facilita a definição de papéis e de tarefas específicas. O modelo `Claude 3.5 Sonnet` assume o processamento cognitivo. Este modelo apresenta alto desempenho documentado em tarefas de raciocínio lógico e em interpretação de código SQL. As bibliotecas `psycopg2` e `SQLAlchemy` operam a conexão com o banco de dados e asseguram a coleta eficiente das métricas. A ferramenta `sqlparse` executa a análise estrutural e a formatação dos comandos SQL antes do envio ao modelo de linguagem. Por fim, o `Azure Identity SDK` gerencia a autenticação e garante a segurança do acesso à nuvem. Essa composição tecnológica padroniza o ambiente experimental e viabiliza a reprodutibilidade da pesquisa.

4. Resultados

A Tabela 2 consolida as métricas coletadas nos dois ambientes de teste. Os resultados comprovam a capacidade do sistema multiagente na identificação de gargalos de desempenho em servidores PostgreSQL. A ferramenta gera recomendações de otimização de forma automática. Os testes também revelaram limitações da solução. Essa constatação possibilita a avaliação de pontos positivos e de oportunidades de melhoria. A principal limitação decorre das informações da extensão *pg_stat_statements*. A conexão ocorre em um banco de dados específico. Contudo, essa extensão armazena estatísticas em nível de servidor. Assim, o relatório apresenta consultas de diferentes bases sem a identificação clara da origem, o que torna algumas recomendações menos intuitivas para o usuário.

Apesar dessa limitação, o sistema obteve resultados satisfatórios ao identificar problemas de desempenho. Os principais achados incluem a sugestão de índices, a identificação de índices com ordem inadequada de colunas e recomendações de reescrita de consultas. O relatório também detectou *joins* desnecessários, conversões implícitas em campos JSON com bloqueio de índices e excesso de leituras em disco. A ferramenta sugeriu ajustes em parâmetros do PostgreSQL, como *shared_buffers*, *wal_buffers* e *synchronous_commit*. Um especialista implementou doze dos dezoito índices sugeridos pelo sistema. O objetivo foi aferir a qualidade prática das recomendações. O estudo executou uma nova análise para comparar os resultados antes e após as alterações.

A criação dos índices alterou o foco das recomendações. Problemas com consultas lentas deixaram a posição de principal gargalo. A ferramenta passou a sugerir

Métrica	Ambiente 1	Ambiente 2
Execuções (máx)	1.880.956	19.721.602
Tempo total (s)	2.615,941	4.174.390
Tempo total (h)	0,727	1.159
Tempo médio (s)	0,003	até 0,93
Blocos em cache	2.09 mi	>80 bi
Blocos em disco	592.325	>1 mi
Linhas processadas	∼1.710.740	>395 mil (por query crítica)
Uso de Seq Scan	Alto (problemático)	Alto (problemático)
Funções em filtro	Não	Sim (*lower*)
Tipo de gargalo	Frequência	I/O + CPU + cardinalidade
Complexidade das queries	Moderada	Alta

Tabela 2. Comparação dos dados obtidos nos ambientes analisados

otimizações de aplicação e de configuração do servidor. As novas recomendações incluíram inserções em lote, substituição de comandos com `SELECT *`, remoção de *casts* desnecessários, monitoramento do *VACUUM* e avaliação de índices com baixa cardinalidade. Esse comportamento atesta a eficácia das recomendações originais. A resolução do gargalo primário expôs problemas secundários de desempenho.

A análise dos planos de execução confirmou esse resultado. Operações de *Sequential Scan* evoluíram para *Index Scan* ou *Index Only Scan* em diversas consultas. Consultas com *JOINs* deixaram de executar varreduras completas nas tabelas e adotaram os índices apropriados. A comparação entre os cenários sem índice, heurística simples e PostgresAnalyzer evidenciou ganhos progressivos de desempenho. O tempo de execução caiu de $3596,429$ ms para $14,130$ ms com a heurística simples e para $0,045$ ms com a abordagem proposta. Além disso, o custo estimado pelo otimizador do PostgreSQL, métrica usada para avaliar a eficiência do plano de execução, reduziu de $5339,39$ para $7,53$ com as recomendações geradas. Esses resultados indicam que a solução proposta foi a mais eficiente na redução do custo e do tempo de resposta da consulta. Essas mudanças atestam o melhor aproveitamento dos recursos do SGBD.

O sistema atua de forma eficiente como ferramenta de apoio à análise de desempenho. A solução automatiza a coleta de métricas, a identificação de gargalos e a geração de recomendações. No entanto, a organização das informações no relatório demanda aprimoramentos. Trabalhos futuros podem definir agentes mais especializados e aplicar *prompts* específicos para cada etapa da análise. Conclui-se que a ferramenta proposta não substitui o trabalho do DBA ou do DBRE. O sistema consolida-se como apoio prático à tomada de decisão no processo de otimização de bancos de dados PostgreSQL.

5. Conclusão

Este trabalho apresentou uma estrutura baseada em agentes de Inteligência Artificial para apoiar a análise e a otimização de bancos de dados PostgreSQL. A solução automatiza a coleta e a análise de métricas e gera recomendações para otimização de consultas e criação de índices. Desta forma, contribui-se para reduzir a dependência de DBAs e DBREs em atividades rotineiras. Embora o sistema não realize alterações automaticamente no banco de dados, todas as etapas de análise são executadas de forma automatizada após sua inicialização pelo usuário. Essa abordagem fornece suporte à tomada de decisão, preservando a segurança do ambiente ao evitar modificações diretas.

Trabalhos Futuros. Pretende-se ampliar as funcionalidades do agente ao incorporar novas análises e desenvolver uma interface gráfica que torne a ferramenta mais acessível. Com isso, espera-se ampliar sua utilização por DBAs, DBREs,desenvolvedores e outros profissionais envolvidos na administração e manutenção de bancos de dados.

Referências

Amaral, K., Andrade, L., and Campos, E. (2025). O uso da inteligência artificial na otimização de consultas em bancos de dados. *Revista Camalotes*, 4.

ANTHROPIC (2026). Claude 3.5 sonnet: Technical documentation. `https://www.anthropic.com/claude`. Disponível em: `https://www.anthropic.com/claude`. Acesso em: mar. 2026.

Chaudhari, B. (2025). Automated database tuning using ai: A comprehensive framework for real-time performance optimization. In *myresearchgo Vidya-Varta 2025*, pages 12–31. Vidya-Varta Publication, India.

CrewAI Inc. (2026). CrewAI documentation. `https://docs.crewai.com`. Acesso em: mar. 2026.

Lao, J., Wang, Y., Li, Y., Wang, J., Zhang, Y., Cheng, Z., Chen, W., Tang, M., and Wang, J. (2024). Gptuner: A manual-reading database tuning system via gpt-guided bayesian optimization. *Proc. VLDB Endow.*, 17(8):1939–1952.

Marcus, R. and Papaemmanouil, O. (2018). Deep reinforcement learning for join order enumeration. In *Proceedings of the First International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*, aiDM'18, New York, NY, USA. Association for Computing Machinery.

Morelli, E. T., Monteiro, J. M., Almeida, A. C., and Lifschitz, S. (2009). Reindexação automática em sgbds relacionais. In Brayner, A., editor, *XXIV Simpósio Brasileiro de Banco de Dados, 05-09 de Outubro, Fortaleza, Ceará, Brasil, Anais*, pages 31–45. SBC.

Mozaffari, M., Dignös, A., Gamper, J., and Störl, U. (2024). Self-tuning database systems: A systematic literature review of automatic database schema design and tuning. *ACM Comput. Surv.*, 56(11).

Oliveira, R., Baião, F., Machado, J., Almeida, A. C., and Lifschitz, S. (2022). Autonomic combination and selection of tuning actions. In *Anais do XXXVII Simpósio Brasileiro de Bancos de Dados*, pages 39–51, Porto Alegre, RS, Brasil. SBC.

Petrola, L., Brayner, A., and Franco, W. (2025). Heuristic-guided text-to-sql translation with llms: Optimizing natural language interfaces for relational databases. In *Anais do XL Simpósio Brasileiro de Bancos de Dados*, pages 126–139, Porto Alegre, RS, Brasil. SBC.

Sharma, A., Schuhknecht, F., and Dittrich, J. (2018). The case for automatic database administration using deep reinforcement learning. *https://doi.org/10.48550/arXiv.1801.05643*.

Wooldridge, M. (2009). *An Introduction to Multiagent Systems*. John Wiley & Sons, 2 edition.