UNIVERSIDADE FEDERAL DE ITAJUBÁ

SISTEMAS DE INFORMAÇÃO

# Relatório Banco de Dados II - SPAD02

**Antony Souza Siqueira**
**Diego Reis Fagundes Varella**
**João Lucas Lopes Dias**

Trabalho de Conclusão de Curso
Bacharelado em Sistemas de Informação

INSTITUTO DE MATEMÁTICA E COMPUTAÇÃO
UNIFEI - Itajubá

2025

UNIVERSIDADE FEDERAL DE ITAJUBÁ
INSTITUTO DE MATEMÁTICA E COMPUTAÇÃO
CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

# Relatório Banco de Dados II - SPAD02

Antony Souza Siqueira, Diego Reis Fagundes Varella, João Lucas Lopes Dias

*Orientadora*: Vanessa Cristina Oliveira De Souza

## 1 Introdução

Este trabalho tem como objetivo o desenvolvimento de uma aplicação capaz de gerar relatórios personalizados (*ad hoc*) com base em dados obtidos da API pública do Stack Overflow. A escolha dessa API se deve à sua riqueza de informações relacionadas a perguntas, respostas, usuários, comentários e tags, proporcionando um cenário realista para modelagem de banco de dados e análise de performance.

## 2 Desenvolvimento

### 2.1 Modelo Entidade-Relacionamento

A Figura 1 apresenta o modelo entidade-relacionamento construído a partir da análise dos dados da API. As principais entidades envolvem usuários, perguntas, respostas, comentários e tags, refletindo a estrutura da plataforma Stack Overflow.


Figura 1: Modelo Entidade-Relacionamento da base Stack Overflow

### 2.2 Modelo Relacional

A partir do modelo entidade-relacionamento, foi construído o seguinte modelo relacional:

- **USERS**
  *PK:* **user_id**
  Atributos: display_name, reputation, creation_date, is_employee, location.
- **QUESTIONS**
  *PK:* **question_id**
  *FK:* **user_id** → USERS(user_id)
  Atributos: title, creation_date, score, is_answered, answer_count, view_count.
- **ANSWERS**
  *PK:* **answer_id**
  *FKs:* **user_id** → USERS(user_id), **question_id** → QUESTIONS(question_id)
  Atributos: creation_date, score, is_accepted.
- **COMMENTS**
  *PK:* **comment_id**
  *FK:* **user_id** → USERS(user_id)
  Atributos: creation_date, body.
- **TAGS**
  *PK:* **tag_id**
  Atributos: name, has_synonyms, is_moderator_only, is_required, count.
- **QUESTION_TAGS**
  *PK composta:* (**question_id**, **tag_id**)
  *FKs:* **question_id** → QUESTIONS(question_id), **tag_id** → TAGS(tag_id)

### 2.3 Dicionário de Dados

O dicionário de dados descreve os atributos de cada tabela do banco, com seus tipos e finalidades.

#### Tabela USERS

- **user_id (int)** – Identificador único do usuário *[PK]*
- **display_name (varchar)** – Nome de exibição do usuário
- **reputation (int)** – Pontuação de reputação do usuário
- **creation_date (datetime)** – Data de criação da conta
- **is_employee (boolean)** – Indica se o usuário é funcionário da plataforma (True/False)
- **location (varchar)** – Localização do usuário

## Tabela QUESTIONS

- **question_id (int)** – Identificador único da pergunta *[PK]*
- **title (varchar)** – Título da pergunta
- **creation_date (datetime)** – Data de criação
- **score (int)** – Pontuação da pergunta
- **user_id (int)** – Usuário que fez a pergunta *[FK → USERS(user_id)]*
- **is_answered (boolean)** – Indica se a pergunta foi respondida
- **answer_count (int)** – Número de respostas
- **view_count (int)** – Número de visualizações

## Tabela ANSWERS

- **answer_id (int)** – Identificador único da resposta *[PK]*
- **creation_date (datetime)** – Data de criação da resposta
- **score (int)** – Pontuação da resposta
- **question_id (int)** – Pergunta associada *[FK → QUESTIONS(question_id)]*
- **user_id (int)** – Usuário que respondeu *[FK → USERS(user_id)]*
- **is_accepted (boolean)** – Indica se a resposta foi aceita

## Tabela COMMENTS

- **comment_id (int)** – Identificador único do comentário *[PK]*
- **creation_date (datetime)** – Data de criação
- **user_id (int)** – Usuário que comentou *[FK → USERS(user_id)]*
- **body (text)** – Conteúdo do comentário

## Tabela TAGS

- **tag_id (int)** – Identificador único da tag *[PK]*
- **name (varchar)** – Nome da tag
- **has_synonyms (boolean)** – Indica se a tag possui sinônimos
- **is_moderator_only (boolean)** – Indica se apenas moderadores podem usá-la
- **is_required (boolean)** – Indica se a tag é obrigatória
- **count (int)** – Quantidade de vezes que a tag foi utilizada

## Tabela QUESTION_TAGS

- **question_id (int)** – Pergunta relacionada *[FK → QUESTIONS(question_id)]*
- **tag_id (int)** – Tag relacionada *[FK → TAGS(tag_id)]*
- *Chave Primária Composta:* (**question_id**, **tag_id**)

# 3 Testes de Performance

Com o objetivo de avaliar a escalabilidade e eficiência do banco de dados, foram realizados testes de desempenho utilizando a ferramenta Apache JMeter. Cada cenário foi executado 10 vezes, com reinicialização do JMeter entre os testes para evitar impactos de cache ou conexão persistente. Dois tipos principais de testes foram conduzidos:

- **Latência vs Número de Threads (requisições fixas)**
- **Latência vs Número de Requisições (threads fixas)**

## 3.1 Latência vs Número de Threads

Neste teste, manteve-se o número de requisições fixo por thread, aumentando progressivamente o número de threads (usuários simultâneos). A Tabela 1 apresenta os resultados obtidos.

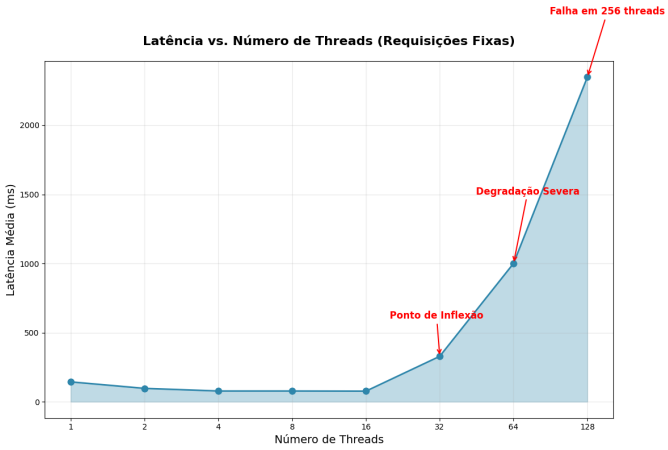


Figura 2: Latência média em função do número de threads, destacando os pontos de inflexão, degradação e falha.

Tabela 1: Latência média em função do número de threads

| Número de Threads | Latência Média (ms) |
|---|---|
| 1 | 144 |
| 2 | 97 |
| 4 | 78 |
| 8 | 78 |
| 16 | 77 |
| 32 | 329 |
| 64 | 1001 |
| 128 | 2348 |
| 256 | ERRO |

**Análise dos Resultados:**

- **Zona Estável (1 a 16 threads):** latência entre 77 ms e 144 ms, com desempenho consistente. O sistema opera de forma ideal, com possível aquecimento de cache e uso eficiente do pool de conexões.
- **Ponto de Inflexão (32 threads):** latência sobe para 329 ms. Indício de saturação do sistema, com aumento de locks e limitação de recursos.

- **Zona de Degradação (64 e 128 threads):** latência cresce exponencialmente, atingindo até 2348 ms. A sobrecarga prejudica o tempo de resposta.
- **Ponto de Falha (256 threads):** o banco de dados falha em responder. O número máximo de conexões pode ter sido excedido, tornando o sistema indisponível.

## 3.2 Latência vs Número de Requisições

Neste segundo cenário, o número de threads foi mantido constante e o número de requisições por thread foi progressivamente aumentado. Os resultados são apresentados na Tabela 2.

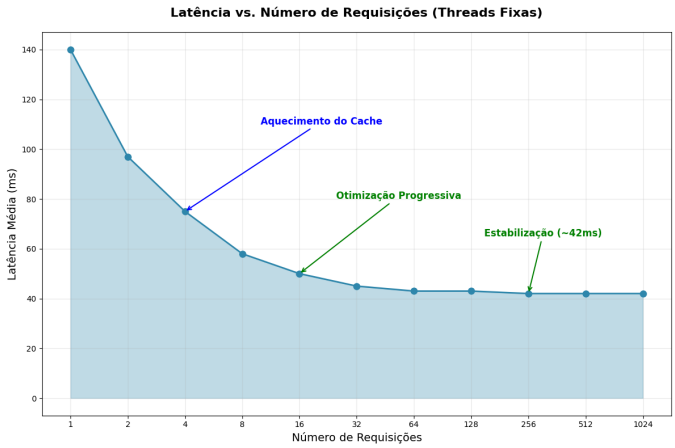


Figura 3: Latência média em função do número de requisições, destacando o aquecimento do cache, otimização progressiva e estabilização.

Tabela 2: Latência média em função do número de requisições

| Número de Requisições | Latência Média (ms) |
|---|---|
| 1 | 140 |
| 2 | 97 |
| 4 | 75 |
| 8 | 58 |
| 16 | 50 |
| 32 | 45 |
| 64 | 43 |
| 128 | 43 |
| 256 | 42 |
| 512 | 42 |
| 1024 | 42 |

**Análise dos Resultados:**

- **Fase de Aquecimento (1 a 8 requisições):** redução de latência de 140 ms para 58 ms, indicando aquecimento do banco e otimização de cache e índices.
- **Fase de Otimização (16 a 32 requisições):** melhora moderada até atingir latência mínima de 45 ms.
- **Zona de Estabilização (64 a 1024 requisições):** estabilidade em torno de 42 ms, sem indícios de sobrecarga. Mostra o bom desempenho do sistema em workloads sequenciais.
- **Ausência de Ponto de Falha:** mesmo com 1024 requisições, o banco manteve a performance estável.

## 3.3 Ambiente de Testes

Todos os testes foram realizados na mesma máquina. A seguir estão as especificações da máquina utilizada:

- **Processador:** AMD Ryzen 5 5600 OC
- **Placa de Vídeo:** NVIDIA RTX 3070
- **Memória RAM:** 24 GB
- **Armazenamento:** SSD 120 GB
- **Sistema Operacional:** Windows 10

## 3.4 Conclusão Comparativa

Os testes demonstraram dois comportamentos distintos do PostgreSQL:

- **Requisições sequenciais:** apresentaram melhoria logarítmica até a estabilização, sem degradação perceptível mesmo com 1024 requisições.
- **Concorrência simultânea:** causou degradação exponencial da latência, com ponto de falha observado a partir de 256 threads.

Esses resultados evidenciam que o sistema é eficiente para grandes volumes de dados quando acessados de forma sequencial, mas exige atenção ao lidar com alta concorrência. Estratégias de controle de concorrência, balanceamento de carga ou enfileiramento de requisições podem ser necessárias para ambientes com múltiplos usuários simultâneos.

# 4 Demonstração da Aplicação

Para complementar a apresentação do projeto, foi produzido um vídeo demonstrando o funcionamento da aplicação desenvolvida.

O vídeo pode ser acessado **clicando aqui**.

Para acessar o repositório do backend do projeto, **clique aqui**.

## Count das tabelas

| | tabela text | total_registros bigint |
|---|---|---|
| 1 | ANSWERS | 158 |
| 2 | COMMENTS | 2704 |
| 3 | QUESTIONS | 2478 |
| 4 | TAGS | 3624 |
| 5 | USERS | 4827 |

Figura 4: Contagem do número de registros em cada tabela.