Ferramentas para Geração de APIs REST a partir de Bancos de Dados PostgreSQL: uma avaliação comparativa

Vinicius Perão da Silva[1], **Evandro Miguel Kuszera**[1]

[1]Universidade Tecnológica Federal do Paraná (UTFPR)
Dois Vizinhos – PR – Brasil

vyny2015@hotmail.com, evandrokuszera@utfpr.edu.br

Abstract. *This paper presents a comparative analysis of open source tools for the automatic generation of REST APIs from PostgreSQL. They were evaluated regarding query expressivity and performance under two workload profiles. The results reveal differences among the tools, providing users with support in choosing suitable solutions for data ingestion and data manipulation scenarios.*

Resumo. *Este artigo apresenta uma análise comparativa de ferramentas open source para geração automática de APIs REST a partir do PostgreSQL. Elas foram avaliadas em termos de expressividade de consultas e desempenho sob dois perfis de carga de trabalho. Os resultados mostram diferenças entre as ferramentas, fornecendo subsídios aos usuários na escolha de soluções adequadas para cenários de ingestão e manipulação de dados.*

1. Introdução

Integrar APIs REST a bancos de dados relacionais continua sendo um desafio técnico relevante. Nesse contexto, a geração automática de APIs surge como alternativa para reduzir o tempo de desenvolvimento e facilitar o acesso aos dados. Abordagens baseadas em modelos demonstram que é possível criar *endpoints* CRUD configuráveis e extensíveis a partir de metamodelos e esquemas de dados [Guizzardi et al. 2015]. Diferentes ferramentas foram propostas para gerar APIs REST a partir de bancos de dados relacionais [Hasura Inc. 2024, Directus Foundation 2024, Strapi Solutions 2024, NocoDB 2025, PostgREST 2025, pREST 2025, DreamFactory 2025]. Essas soluções diferem em aspectos como suporte a bancos de dados, autenticação e autorização, desempenho e adaptabilidade a diferentes cenários, tornando a escolha da ferramenta adequada uma tarefa não trivial.

Diante desse cenário, este trabalho investiga soluções de geração automática de APIs REST por meio de uma avaliação comparativa de quatro ferramentas com suporte ao PostgreSQL, utilizando critérios quantitativos. Os resultados indicam diferenças relevantes em termos de desempenho e expressividade de consultas entre as ferramentas.

2. Aspectos Conceituais

Esta seção apresenta os conceitos necessários para a compreensão adequada do trabalho.

2.1. *APIs REST*

REST (*Representational State Transfer*) é um estilo arquitetural para a criação de serviços web que utiliza o protocolo HTTP e métodos como GET, POST, PUT, DELETE e PATCH

para disponibilizar recursos de maneira uniforme. APIs REST têm como principais características a arquitetura cliente–servidor, a comunicação sem estado (*stateless*), o uso de cache e a interface uniforme, expondo recursos por meio de URLs e representações como JSON ou XML [Fielding 2000].

2.2. Ferramentas para Geração Automática de *APIs REST*

Neste trabalho, são consideradas ferramentas que, por meio da introspecção do esquema do banco de dados (tabelas, chaves e relacionamentos), publicam endpoints HTTP padronizados para operações CRUD, dispensando a implementação manual de controladores e rotas. Em geral, essas ferramentas mapeiam entidades relacionais para recursos REST, serializam respostas em JSON e oferecem mecanismos de consulta, como filtros, paginação e ordenação, em conformidade com o estilo REST [Fielding 2000]. Essas soluções podem ser utilizadas tanto para acesso quanto para ingestão de dados em cenários de engenharia e ciência de dados.

3. Metodologia

Esta seção descreve o processo de seleção das ferramentas de geração de APIs, o ambiente experimental, o esquema da base de dados e processo de carga de dados, os procedimentos e as métricas utilizadas para a avaliação comparativa.

3.1. Processo de Seleção das Ferramentas

A busca por ferramentas foi realizada no *GitHub*, a partir das palavras-chave "*API generation*", "*PostgreSQL to REST*", "*automatic API REST generation*" e "*database to API REST*". **Critérios de inclusão** aplicados: *i)* Ferramenta relacionada à geração de *APIs REST* e *ii)* Documentação oficial existente - resultando em 30 ferramentas candidatas. **Critérios de exclusão** aplicados: *i)* Ausência de geração própria de *APIs REST* (suporte apenas indireto), *ii)* Ausência de compatibilidade nativa com o PostgreSQL, *iii)* Inatividade (sem *commit* nos últimos 6 meses), *iv)* Não *open source* e *v)* Popularidade (menos de 1.000 *stars* no *GitHub*) - resultando na exclusão de 26 ferramentas. As quatro ferramentas selecionadas são apresentadas na próxima seção. Ademais, os *scripts* de configuração e execução dos experimentos estão disponíveis em repositório público[1].

3.2. Ferramentas Selecionadas

A Tabela 1 exibe as ferramentas selecionadas após aplicação dos critérios de exclusão, incluíndo a versão utilizada, a data do último *commit* e o número de estrelas no github.

Tabela 1. Características das ferramentas selecionadas

Ferramenta	Versão	Licença	Último Commit	Popularidade
NocoDB	0.265.1	AGPL-3.0	out/2025	58.300 *stars*
PostgREST	14.0	MIT	out/2025	25.900 *stars*
pREST	2.0.0-rc4	MIT	out/2025	4.400 *stars*
DreamFactory	7.3.0	Apache-2.0	out/2025	1.700 *stars*

[1]https://github.com/iLordVini/tcc2

NocoDB é uma plataforma *open source* que transforma bancos relacionais em interfaces visuais e *APIs REST*, detectando tabelas e relacionamentos e gerando *endpoints CRUD* com interface administrativa integrada [NocoDB 2025]. **PostgREST** é um servidor que converte diretamente um banco PostgreSQL em uma *API REST*, mapeando requisições *HTTP* para *SQL* e reutilizando permissões e políticas do próprio banco [PostgREST 2025]. **pREST** é uma ferramenta em Go para gerar *APIs REST* sobre PostgreSQL, com foco em operações de leitura e escrita [pREST 2025]. **DreamFactory** é uma plataforma que provisiona *APIs REST*, incluindo PostgreSQL, com console de administração para chaves, papéis e integrações [DreamFactory 2025].

3.3. Ambiente Experimental e Procedimento de Execução

Os experimentos foram executados em uma máquina com CPU de 8 núcleos, 16 GB de RAM e SSD, executando Windows 10 build 19045. Os serviços foram estruturados via *docker compose* (Docker 28.5.1, Compose v2.40.0-desktop.1), utilizando PostgreSQL 15 e as imagens oficiais das ferramentas. Os testes de desempenho foram realizados com o Grafana k6, uma ferramenta *open source* para testes de carga e desempenho que permite automatizar a execução de requisições *HTTP* sob diferentes perfis de concorrência e coletar métricas como latência e *throughput* [Grafana Labs 2025]. Os cenários experimentais foram implementados em scripts k6 escritos em JavaScript, garantindo reprodutibilidade e controle sobre os padrões de carga aplicados. O procedimento de execução consiste em: *(i)* provisionar o banco de dados PostgreSQL e aplicar a carga inicial; *(ii)* iniciar o contêiner de cada ferramenta de forma isolada, permitindo que se conecte ao banco e gere automaticamente as *APIs REST* a partir do esquema relacional; *(iii)* executar para cada ferramenta os experimentos; e *(iv)* sumarizar as saídas geradas para análise.

3.4. Modelo de Dados

O modelo de dados usado considera um cenário típico de vendas, incluindo tabelas como *clients*, *sellers*, *products*, *orders* e *order_items*. Esse modelo explora relacionamentos 1:N (por exemplo, um cliente para muitos pedidos) e N:N (por exemplo, pedidos e produtos relacionados por *order_items*), representando cenário tipico para avaliar as ferramentas selecionadas. A Figura 1 apresenta o diagrama entidade-relacionamento e o número de registros por tabela, após a carga inicial de dados.

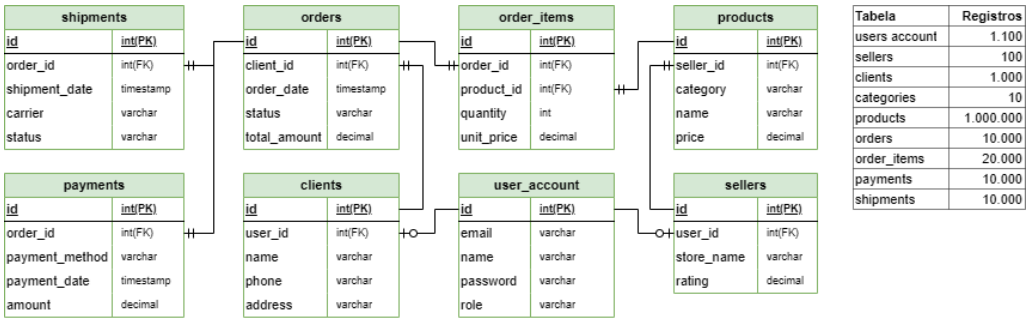

Figura 1. Diagrama entidade-relacionamento e número de registros por tabela.

3.5. Critérios e Métricas de Avaliação

A avaliação das ferramentas selecionadas considera as dimensões expressividade de consultas e desempenho, que são requisitos em cenários de ingestão e integração de dados. Os dois critérios usados são descritos a seguir.

3.5.1. Expressividade de Consultas

Expressividade, neste artigo, representa a capacidade das ferramentas em expor operações sobre o esquema relacional, por meio de consultas e relacionamentos. Cinco consultas com níveis de complexidade diferentes foram definidas (Q1 a Q5). O processo consiste de duas etapas: *i)* as ferramentas recebem o esquema do banco de dados como entrada e geram os endpoints automaticamente; *ii)* as ferramentas são avaliadas quanto à capacidade de reproduzir o comportamento dessas consultas por meio de combinações de filtros (*where*), paginação (*limit*), ordenação (*order by*), agregações e navegação entre relacionamentos. A avaliação verifica se há ou não suporte para implementar as consultas abaixo.

```
-- Q1. Consulta por chave primária
SELECT id, name FROM clients WHERE id = 1;

-- Q2. Consulta paginada e ordenada
SELECT id, name FROM products ORDER BY name ASC LIMIT 10;

-- Q3. Consulta com agregação
SELECT client_id, COUNT(*) AS total FROM orders WHERE client_id = 1 GROUP BY client_id;

-- Q4. Consulta com relacionamento simples
SELECT o.id, o.order_number, c.name AS client_name FROM orders o JOIN
clients c ON o.client_id = c.id WHERE c.id = 1;

-- Q5. Consulta com relacionamento múltiplo
SELECT c.name AS client_name, p.name AS product_name FROM clients c
JOIN orders o ON c.id = o.client_id JOIN order_items oi ON o.id = oi.order_id
JOIN products p ON oi.product_id = p.id WHERE c.id = 1;
```

3.6. Desempenho

Três cenários foram definidos para representar perfis de carga de trabalho distintas, permitindo avaliar o impacto no desempenho das ferramentas em operações de leitura e escrita.

- **Cenário A 100-0 (100% *GET*)**: execução exclusiva de consultas com filtros na tabela *products*; estabelece a linha de base para leitura pura ao isolar os efeitos de escrita.
- **Cenário B 80-20 (80% *GET* / 20% *POST*)**: carga mista com predominância de leitura e fluxo transacional leve de escrita na tabela *products*, refletindo padrões comuns em produção.
- **Cenário C 60-40 (60% *GET* / 40% *POST*)**: carga mista com escrita mais intensiva na tabela *products*, gerando maior contenção e concorrência transacional sobre os recursos do banco.

Cada cenário é avaliado por dois perfis que representam duas cargas de trabalho distintas, permitindo avaliar latência, percentil 95 (P95), *throughput* e uso de memória sob pressões crescentes de concorrência e transações. P95 é definido como o valor de latência (em ms) abaixo do qual se encontram 95% das requisições (por exemplo, um P95 de 100 ms indica que 95% das requisições foram respondidas em até 100 ms). **Perfil 1 (*load*)**: elevação até 25 usuários virtuais em 2 minutos, manutenção em 25 usuários por 7 minutos e redução em 2 minutos (totalizando 11 minutos). **Objetivo**: observar o comportamento sob o uso representativo da operação cotidiana. **Perfil 2 (*stress*)**: elevação até 50 usuários virtuais em 2 minutos, manutenção em 50 usuários por 5 minutos e redução em 1 minuto (totalizando 8 minutos). **Objetivo**: identificar limites de capacidade e taxas de falha sob alta concorrência sustentada.

Métricas de saída: tempo médio de resposta por operação *CRUD* (ms), *throughput* (requisições/segundo), P95 e consumo de *RAM* (MB).

4. Resultados

4.1. Expressividade

Todas as ferramentas implementaram com sucesso as operações básicas (CRUD), filtros, agregação e relacionamento simples, representado pelas consultas Q1 a Q4. No entanto, as consultas que envolvem relacionamentos mais complexos (Q5) expõem limitações específicas em pREST e DreamFactory para cenários multinível.

O PostgREST não apresentou limitações dentro do escopo avaliado, atendendo Q1 a Q5 de forma direta, a partir de recursos e parâmetros de consulta. Em Q3 (agregação), o DreamFactory retorna o *count* em metadados, exigindo pós-processamento pela aplicação, enquanto o NocoDB requer a definição prévia de uma *view* na própria interface da ferramenta para expor agregados como recursos. Em Q4 (relacionamento simples), o NocoDB novamente depende dessa *view* para encapsular os *JOINs* necessários.

No caso de Q5 (relacionamentos multinível), pREST e DreamFactory não executam a expansão encadeada apenas por meio de parâmetros da *API*, pois ambos apresentam uma limitação prática de navegação com até um único *JOIN* por requisição. Para reproduzir a consulta Q5 nessas ferramentas é necessário emitir múltiplas requisições encadeadas na aplicação ou criar previamente uma *view* no banco de dados. Por esse motivo, ambas foram classificadas como "não suportam" Q5 de forma nativa. O NocoDB consegue atender à Q5 ao possibilitar a definição de uma *view* via interface da aplicação, expondo os dados como um recurso da *API*.

4.2. Desempenho

Os experimentos realizados avaliaram o comportamento das ferramentas sob dois perfis de carga de trabalho (*load* e *stress*), com foco na capacidade de processamento (*throughput*), nos tempos de resposta (latência) e na estabilidade ao longo do tempo (consumo de RAM). A Tabela 2 sumariza os resultados.

Considerando em conjunto os três cenários avaliados, o PostgREST apresentou menor latência, alto *throughput* e uso mínimo de memória, enquanto o pREST ficou em segundo lugar, com resultados semelhantes em *throughput* e uma pequena penalidade de latência, o que reforça a adequação de ambas as soluções para cenários que exigem alta eficiência no PostgreSQL. Ambas as ferramentas implementam uma estratégia de mapeamento direto, na qual cada requisição *HTTP* é traduzida em uma operação PostgreSQL, com mínima lógica intermediária de processamento. Essa abordagem tende a reduzir o custo computacional por requisição, pois elimina abstrações desnecessárias e se concentra em validar, autorizar e encaminhar.

O NocoDB apresentou *throughput* moderado. Porém, as latências médias e o consumo de memória foram significativamente superiores, o que pode impactar o dimensionamento da infraestrutura em ambientes de produção. O DreamFactory apresentou as latências mais elevadas, um *throughput* inferior e um aumento no consumo de memória nos três cenários, indicando uma sobrecarga maior sob cargas intensivas e sugerindo a necessidade de capacidade computacional superior para atingir metas de desempenho

Tabela 2. Desempenho das ferramentas nos cenários avaliados

Cenário	Ferramenta	Perfil	Latência (ms)	Latência P95	Throughput	RAM
A 100–0	PostgREST	*load*	**1,81**	**2,83**	**101,54**	**0,035**
A 100–0	pREST	*load*	11,36	15,13	96,95	0,041
A 100–0	NocoDB	*load*	19,45	33,47	93,41	0,617
A 100–0	DreamFactory	*load*	138,17	256,85	60,66	0,416
A 100–0	PostgREST	*stress*	**1,82**	**2,98**	**201,05**	**0,038**
A 100–0	pREST	*stress*	15,08	28,00	188,68	0,043
A 100–0	NocoDB	*stress*	94,78	214,43	137,81	0,632
A 100–0	DreamFactory	*stress*	511,85	948,52	57,14	0,782
B 80–20	PostgREST	*load*	**2,11**	**3,32**	**101,40**	**0,039**
B 80–20	pREST	*load*	11,93	16,85	96,72	0,042
B 80–20	NocoDB	*load*	16,54	32,45	94,66	0,658
B 80–20	DreamFactory	*load*	265,77	492,09	44,06	1,279
B 80–20	PostgREST	*stress*	**1,99**	**3,38**	**200,94**	**0,040**
B 80–20	pREST	*stress*	16,06	31,00	187,84	0,044
B 80–20	NocoDB	*stress*	97,11	224,52	136,68	0,657
B 80–20	DreamFactory	*stress*	679,55	1.405,88	46,24	1,258
C 60–40	PostgREST	*load*	**2,11**	**3,32**	**101,40**	**0,039**
C 60–40	pREST	*load*	12,60	19,72	96,41	0,042
C 60–40	NocoDB	*load*	15,58	30,72	95,09	0,631
C 60–40	DreamFactory	*load*	280,98	551,72	42,68	1,056
C 60–40	PostgREST	*stress*	**1,99**	**3,38**	**200,94**	**0,040**
C 60–40	pREST	*stress*	17,19	33,52	186,81	0,044
C 60–40	NocoDB	*stress*	92,62	206,81	138,83	0,652
C 60–40	DreamFactory	*stress*	694,38	1.585,93	45,47	1,104

equivalentes. NocoDB e o DreamFactory funcionam como plataformas generalistas, orientadas para múltiplas fontes de dados e interfaces de negócio. Essas plataformas podem incorporar camadas de abstração de dados, sistemas de metadados, interpretadores de configurações visuais e funcionalidades administrativas na memória, o que impacta no desempenho final.

5. Conclusão

Este estudo realizou uma análise comparativa entre ferramentas de geração automática de *APIs REST* com suporte ao PostgreSQL: NocoDB, PostgREST, pREST e DreamFactory. Foram realizados experimentos para avaliar a expressividade (implementação de consultas) e desempenho (latência, *throughput*, consumo de memória sob cargas variadas).

Os resultados indicam que o PostgREST tem menor latência, alto *throughput* e uso contido de memória, além de não apresentar limitações relevantes nas consultas avaliadas, o que o torna adequado para cenários em que a eficiência e a expressividade de consultas em PostgreSQL são prioridades. O pREST apresenta desempenho próximo em *throughput*, com penalidade moderada de latência, mantendo suporte a operações sobre o modelo relacional. Já o NocoDB e o DreamFactory demonstraram maior sobrecarga de processamento e consumo de memória, mas oferecem contrapartidas importantes, como suporte nativo a múltiplos SGBDs, interfaces administrativas mais ricas e maior foco em configuração visual. O uso do NocoDB e o DreamFactory tendem a ser mais apropriados em contextos nos quais a centralização da administração de dados, a integração com diferentes fontes e a facilidade de uso são mais determinantes do que o desempenho, cabendo ao profissional ponderar à luz das necessidades específicas de cada projeto.

6. Origem do Trabalho e Uso de IA

Trabalho de Conclusão de Curso (TCC) do curso de Bacharelado em Engenharia de Software da UTFPR - Campus Dois Vizinhos, defendido no segundo semestre de 2025. Foi utilizado IA de forma parcial para correção gramatical e revisão linguística.

Referências

Directus Foundation (2024). Directus documentation: Dynamic api generation. `https://docs.directus.io`. Accessed: 2026-05.

DreamFactory (2025). DreamFactory: Instant APIs for any database. Acesso em: 20 out. 2025.

Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine. Acesso em: 20 out. 2025.

Grafana Labs (2025). Grafana k6 documentation. Acesso em: 20 out. 2025.

Guizzardi, G., Guizzardi, R. S. S., and Almeida, J. P. A. (2015). EMF-REST: Generation of RESTful APIs from models. In *Proceedings of the 2015 International Conference on Model-Driven Engineering Languages and Systems*, Ottawa, Canada. ACM. Acesso em: 20 out. 2025.

Hasura Inc. (2024). Hasura graphql engine documentation. `https://hasura.io`. Accessed: 2026-05.

NocoDB (2025). NocoDB: Open source airtable alternative. Acesso em: 20 out. 2025.

PostgREST (2025). PostgREST: REST API for any postgres database. Acesso em: 20 out. 2025.

pREST (2025). pREST: PostgreSQL RESTful API. Acesso em: 20 out. 2025.

Strapi Solutions (2024). Strapi documentation. `https://strapi.io`. Accessed: 2026-05.