SERVIÇO NACIONAL DE APRENDIZAGEM INDUSTRIAL

Julio Coronetti Regino

Murilo Antunes da Silva Galhardo de Carvalho

Paola de Oliveira

BANCO DE DADOS - SISTEMA DE SAÚDE

Professor André Souza

Banco de dados

Sorocaba

2025

SUMÁRIO

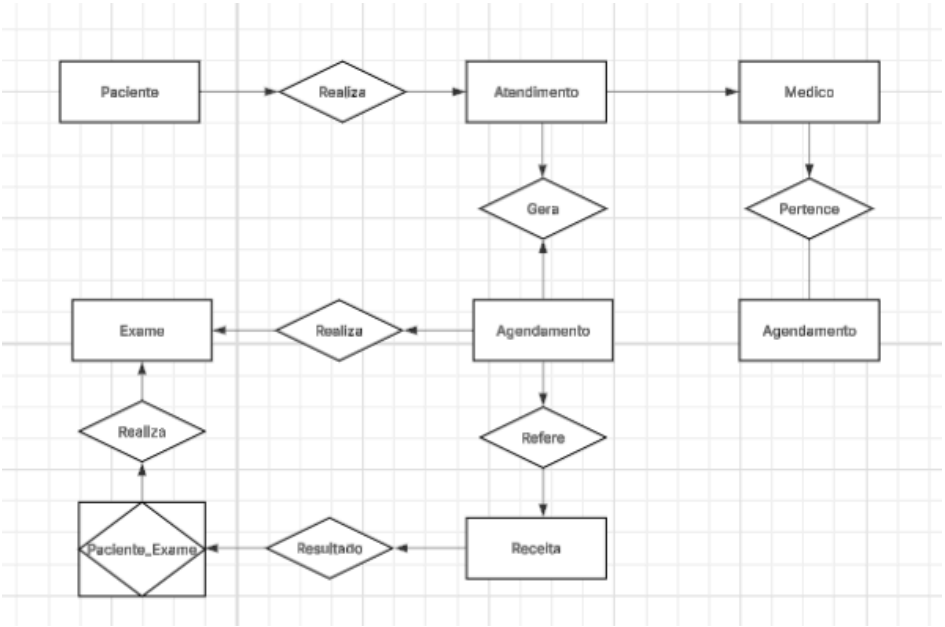
1. INTRODUÇÃO

Este projeto tem como objetivo a criação de um banco de dados relacional para gerenciar informações relacionadas a atendimentos médicos, pacientes, exames e receitas. Foi desenvolvido com base nos conhecimentos adquiridos durante a disciplina de Banco de Dados Relacional, aplicando modelagem de dados, normalização e manipulação via SQL (DDL, DML, DQL), com implementação em PostgreSQL via pgAdmin.

2. MODELAGEM CONCEITUAL

O modelo foi desenvolvido com base no seguinte conjunto de entidades e relacionamentos:

- Paciente: Armazena dados pessoais dos pacientes.
- Médico: Dados de profissionais de saúde.
- Atendimento: Registra as consultas realizadas.
- Agendamento: Guarda os agendamentos de consultas futuras.
- Exame: Lista tipos de exames existentes.
- Paciente_Exame: Relaciona pacientes e exames com resultados.
- Receita: Registra medicamentos prescritos em atendimentos.

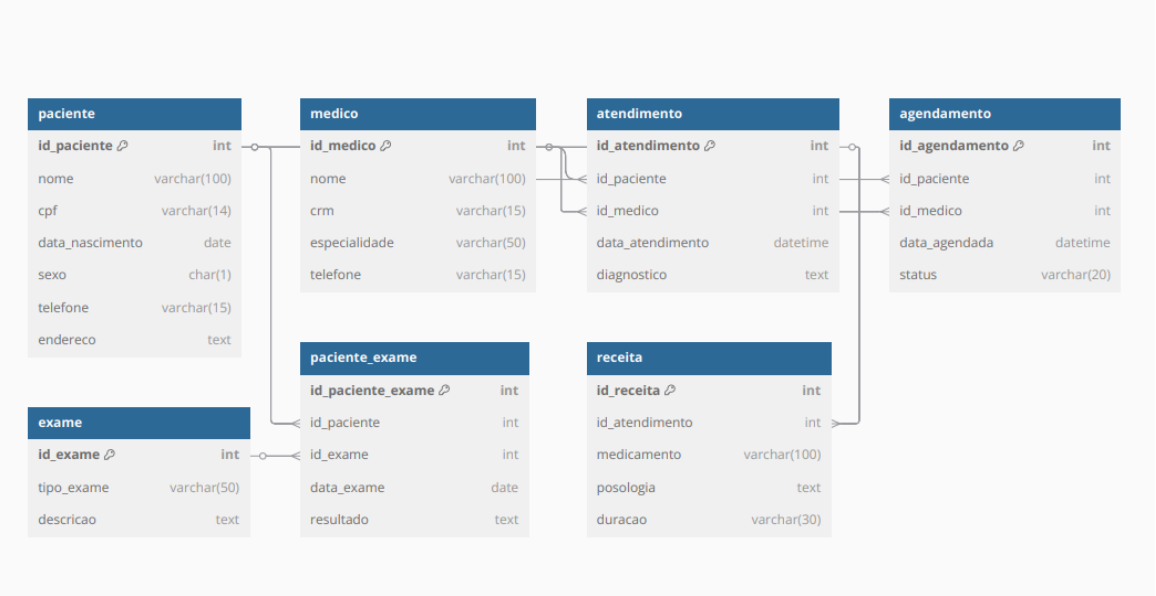


3. MODELAGEM LÓGICA E NORMALIZAÇÃO

As entidades foram convertidas em tabelas relacionais com chaves primárias e estrangeiras. A normalização foi aplicada até a 3ª Forma Normal (3NF), garantindo:

- Ausência de grupos repetitivos (1NF)
- Eliminação de dependências parciais (2NF)
- Remoção de dependências transitivas (3NF)

Modelo Lógico:



4. ESTRUTURA DO BANCO DE DADOS (DDL)

O script apresentado utiliza comandos DDL para **definir a estrutura** do banco de dados de um sistema de saúde. A DDL é responsável por criar, alterar e remover estruturas como tabelas e relacionamentos. Abaixo está a descrição das tabelas criadas:

- **Tabela `paciente`**: Armazena dados dos pacientes, como nome, CPF, data de nascimento, sexo, telefone e endereço. O campo `cpf` é único e `id_paciente` é a chave primária.

- **Tabela `medico`**: Registra informações dos médicos, incluindo nome, CRM (único), especialidade e telefone.

- **Tabela `atendimento`**: Representa os atendimentos realizados, relacionando um paciente e um médico, com a data e diagnóstico. Possui chaves estrangeiras (`FOREIGN KEY`) para garantir integridade referencial com as tabelas `paciente` e `medico`.

- **Tabela `agendamento`**: Controla os agendamentos futuros, ligando pacientes e médicos a uma data e status de agendamento.

- **Tabela `exame`**: Contém os tipos de exames disponíveis e suas descrições.

- **Tabela `paciente_exame`**: Relaciona os pacientes aos exames que realizaram, incluindo data e resultado. Utiliza chaves estrangeiras para vincular à tabela de pacientes e exames.

- **Tabela `receita`**: Armazena as receitas geradas em um atendimento, com informações do medicamento, posologia e duração do tratamento. Está ligada à tabela `atendimento`.

Essas definições garantem a organização e integridade dos dados, estruturando o sistema de forma lógica e relacional para suportar as operações do sistema de saúde.

```sql
CREATE TABLE paciente (
  id_paciente SERIAL PRIMARY KEY,
  nome VARCHAR(100) NOT NULL,
  cpf VARCHAR(14) UNIQUE NOT NULL,
  data_nascimento DATE NOT NULL,
  sexo CHAR(1) NOT NULL,
  telefone VARCHAR(15),
  endereco TEXT
);

CREATE TABLE medico (
  id_medico SERIAL PRIMARY KEY,
  nome VARCHAR(100) NOT NULL,
  crm VARCHAR(15) UNIQUE NOT NULL,
  especialidade VARCHAR(50),
  telefone VARCHAR(15)
);

CREATE TABLE atendimento (
  id_atendimento SERIAL PRIMARY KEY,
  id_paciente INT NOT NULL,
  id_medico INT NOT NULL,
  data_atendimento TIMESTAMP NOT NULL,
  diagnostico TEXT,
  FOREIGN KEY (id_paciente) REFERENCES paciente(id_paciente),
  FOREIGN KEY (id_medico) REFERENCES medico(id_medico)
);
```

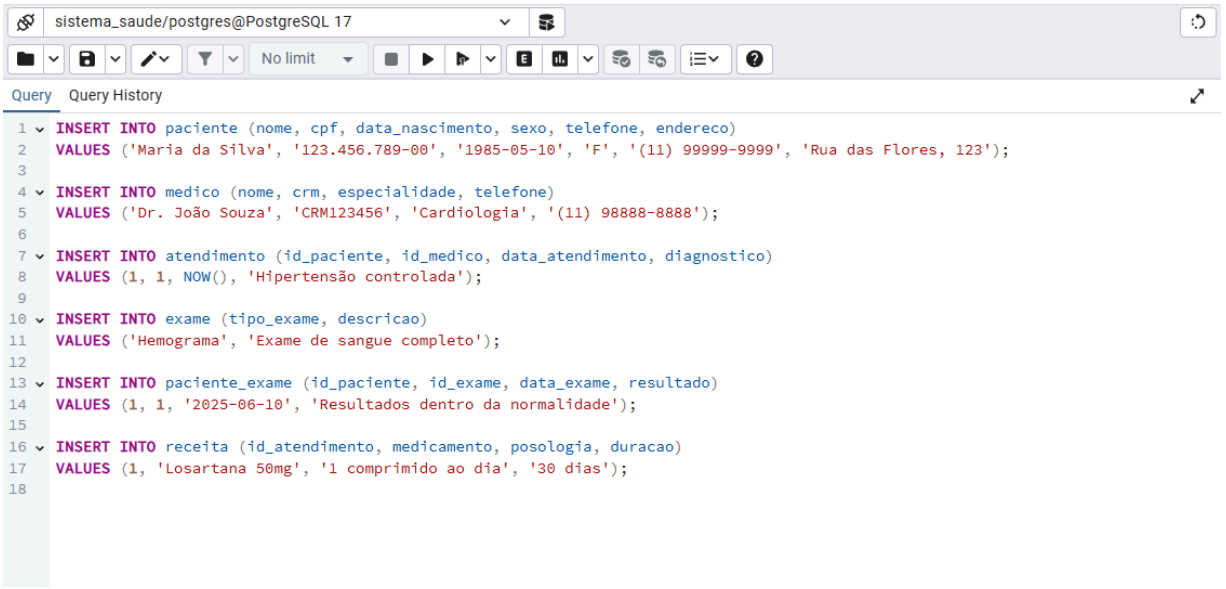
```sql
CREATE TABLE exame (
  id_exame SERIAL PRIMARY KEY,
  tipo_exame VARCHAR(50) NOT NULL,
  descricao TEXT
);

CREATE TABLE paciente_exame (
  id_paciente_exame SERIAL PRIMARY KEY,
  id_paciente INT NOT NULL,
  id_exame INT NOT NULL,
  data_exame DATE NOT NULL,
  resultado TEXT,
  FOREIGN KEY (id_paciente) REFERENCES paciente(id_paciente),
  FOREIGN KEY (id_exame) REFERENCES exame(id_exame)
);

CREATE TABLE receita (
  id_receita SERIAL PRIMARY KEY,
  id_atendimento INT NOT NULL,
  medicamento VARCHAR(100) NOT NULL,
  posologia TEXT,
  duracao VARCHAR(30),
  FOREIGN KEY (id_atendimento) REFERENCES atendimento(id_atendimento)
);
```

5. MANIPULAÇÃO DE DADOS (DML)

Neste exemplo, utilizamos comandos DML para inserir dados em diferentes tabelas de um sistema de saúde. O DML é responsável pela manipulação dos dados armazenados no banco, como inserção (`INSERT`), atualização (`UPDATE`), exclusão (`DELETE`) e consulta (`SELECT`). Abaixo, descrevemos cada inserção realizada:

- Tabela `paciente`: Um novo paciente foi adicionado, contendo informações como nome, CPF, data de nascimento, sexo, telefone e endereço.
- Tabela `medico`: Inserção de um médico com nome, CRM, especialidade e telefone para registro no sistema.
- Tabela `atendimento`: Um atendimento foi registrado, vinculando o paciente ao médico, com a data atual e o diagnóstico de “Hipertensão controlada”.
- Tabela `exame`: Um exame do tipo "Hemograma" foi cadastrado com sua descrição.
- Tabela `paciente_exame`: Associou-se o paciente ao exame realizado, com a data e o resultado “Resultados dentro da normalidade”.
- Tabela `receita`: Foi registrada uma receita médica com os dados do atendimento, nome do medicamento, posologia e duração do tratamento.

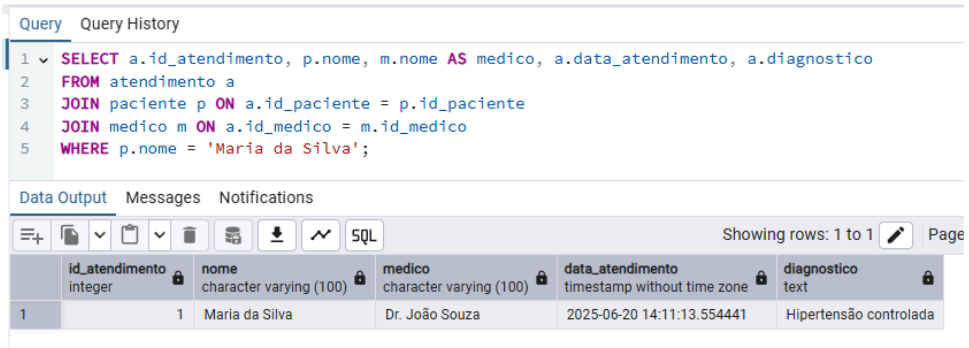
Esses comandos demonstram como a DML é usada para alimentar o banco de dados com informações essenciais ao funcionamento de um sistema de prontuário eletrônico.



6. MANIPULAÇÃO DE DADOS (DQL)

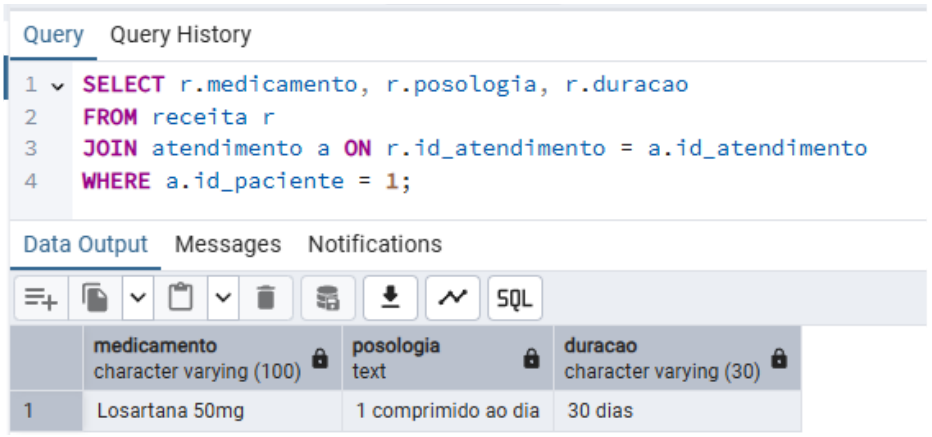
A DQL é responsável pela consulta de dados no banco, sendo representada principalmente pelo comando `SELECT`. Nos exemplos apresentados, foram feitas consultas relacionando múltiplas tabelas por meio de `JOINs`, para obter informações completas de um paciente:

- Consulta de atendimentos:



  Essa consulta retorna os atendimentos da paciente "Maria da Silva", incluindo o nome do médico, a data e o diagnóstico.

- Consulta de receitas:



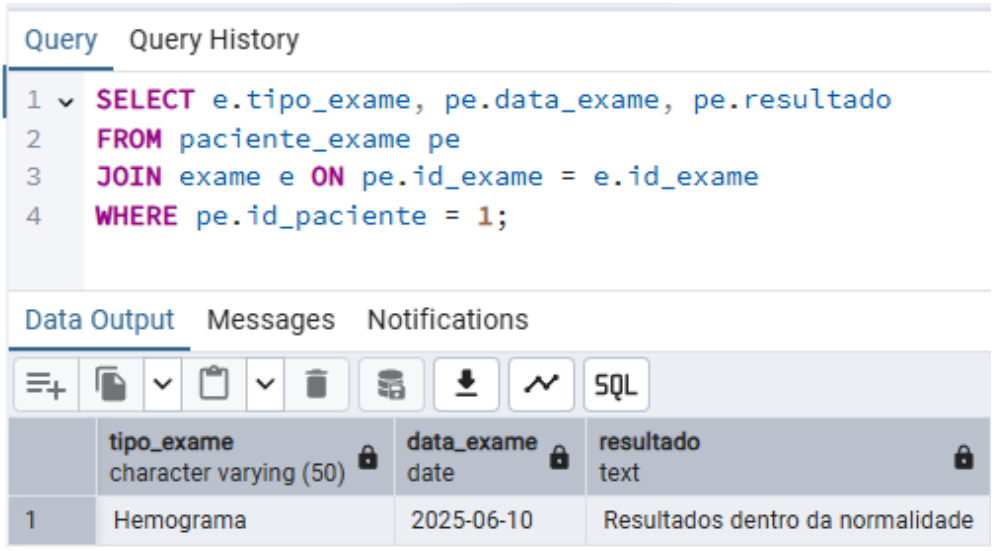
```
Query  Query History
1 SELECT r.medicamento, r.posologia, r.duracao
2 FROM receita r
3 JOIN atendimento a ON r.id_atendimento = a.id_atendimento
4 WHERE a.id_paciente = 1;
```

Data Output | Messages | Notifications

	medicamento character varying (100)	posologia text	duracao character varying (30)
1	Losartana 50mg	1 comprimido ao dia	30 dias

Aqui, buscamos as receitas vinculadas ao paciente de ID 1, mostrando o medicamento prescrito, posologia e duração do tratamento.

- Consulta de exames:



```
Query  Query History
1 SELECT e.tipo_exame, pe.data_exame, pe.resultado
2 FROM paciente_exame pe
3 JOIN exame e ON pe.id_exame = e.id_exame
4 WHERE pe.id_paciente = 1;
```

Data Output | Messages | Notifications

	tipo_exame character varying (50)	data_exame date	resultado text
1	Hemograma	2025-06-10	Resultados dentro da normalidade

Essa consulta exibe os exames realizados pelo paciente de ID 1, com o tipo do exame, a data e o resultado.

Esses exemplos demonstram como a DQL é fundamental para extrair informações específicas e combinadas entre tabelas, permitindo relatórios e análises no sistema de saúde.

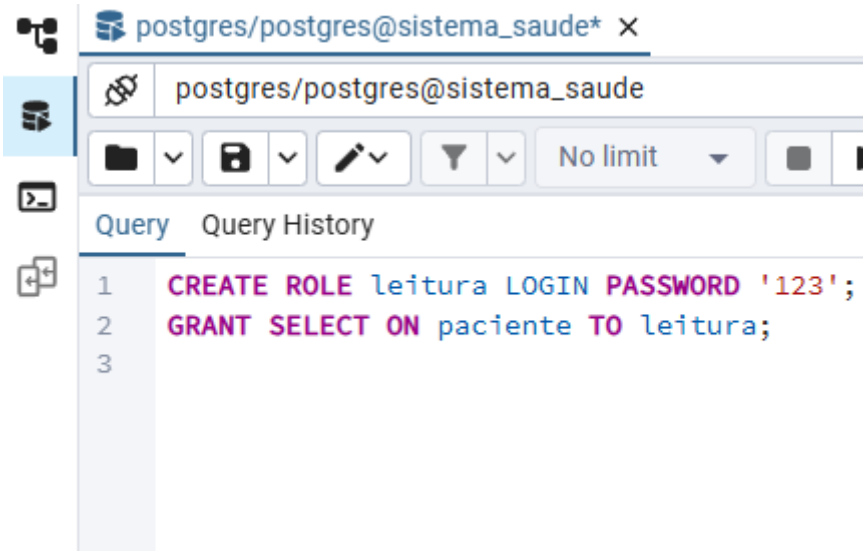
7. CONTROLE DE ACESSO (DCL)

A DCL é usada para **controlar os privilégios de acesso** aos objetos do banco de dados, como tabelas. No exemplo, são utilizados comandos para criar um papel (usuário) com permissão apenas de leitura:

- `CREATE ROLE leitura LOGIN PASSWORD '123';`
  Cria um novo papel (usuário) chamado **leitura**, com permissão de login e senha definida como `'123'`.

- `GRANT SELECT ON paciente TO leitura;`
  Concede permissão de **leitura (SELECT)** na tabela `paciente` para o usuário `leitura`. Com isso, ele pode consultar os dados dessa tabela, mas **não pode inserir, alterar ou excluir** informações.

Esse tipo de controle é essencial para garantir a **segurança e integridade dos dados**, permitindo que diferentes usuários tenham **acessos específicos** de acordo com suas funções no sistema.

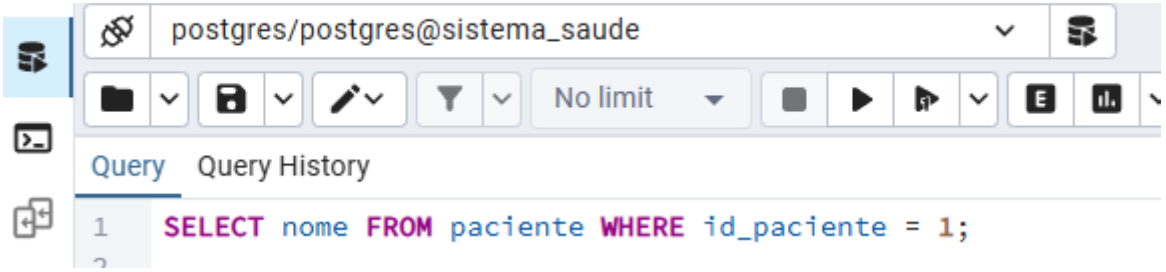


8. CONTROLE DE TRANSAÇÕES (DTL)

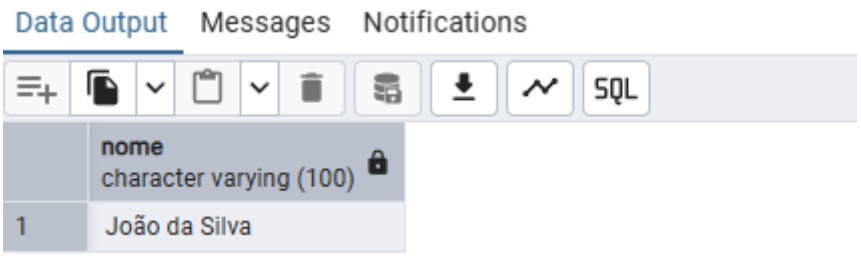
Nesta seção, vamos demonstrar, passo a passo, como realizar operações de leitura, atualização e reversão de dados utilizando comandos SQL no contexto do DTL, conforme ilustrado nas imagens fornecidas.

Consulta Inicial de Dados

O primeiro passo é consultar o nome do paciente com `id_paciente = 1`. Utilizamos o comando abaixo:

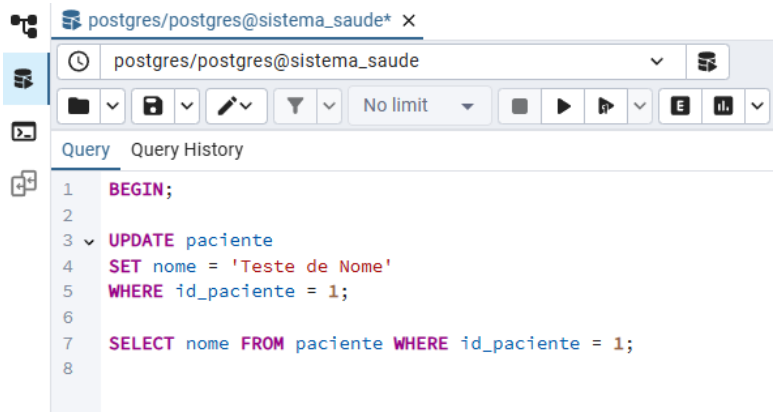


O resultado apresentado mostra que o nome atual do paciente é "João da Silva".

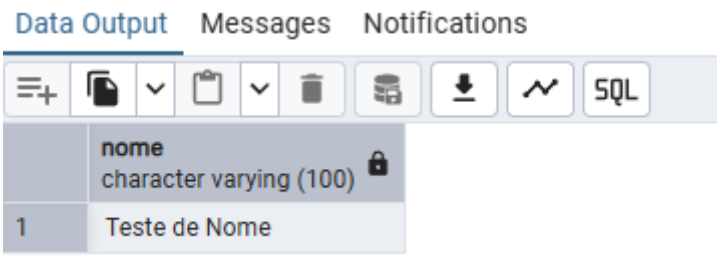


Atualizando Dados com Transação

Para modificar o nome do paciente, iniciamos uma transação com o comando `BEGIN;`. Em seguida, executamos o comando de atualização:



Após a atualização, realizamos uma nova consulta para verificar o resultado:

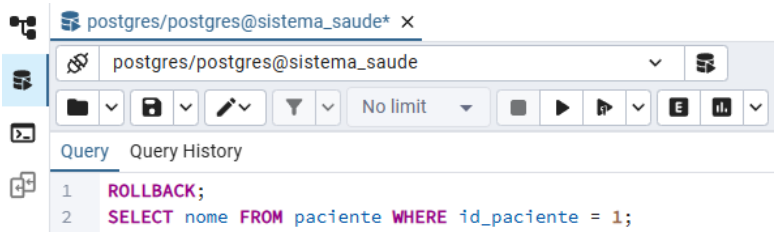


O resultado indica que o nome foi alterado para "Teste de Nome".

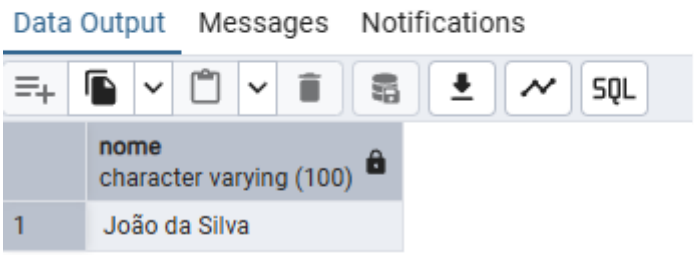
Revertendo a Operação com ROLLBACK

Caso seja necessário desfazer a alteração, podemos utilizar o comando `ROLLBACK;`. Isso faz com que todas as operações realizadas desde o `BEGIN;` sejam desfeitas.

Após o rollback, consultamos novamente o nome do paciente:



O resultado mostra que o nome voltou a ser "João da Silva", confirmando que a reversão foi bem-sucedida.



9. CONCLUSÃO

O projeto proporcionou uma aplicação prática dos principais conceitos de modelagem e implementação de bancos de dados relacionais, consolidando o uso adequado de chaves primárias e estrangeiras, normalização das tabelas e integridade referencial. Por meio de comandos DDL, DML, DCL e DQL, foi possível criar uma estrutura robusta, inserir dados relevantes, controlar permissões de acesso e realizar consultas SQL eficientes e relacionais.

Além disso, o uso de *JOINs* entre tabelas permitiu obter informações completas e contextualizadas, como atendimentos médicos, resultados de exames e prescrições associadas a cada paciente, reforçando a importância da modelagem correta e da manipulação estruturada de dados.

Como possíveis melhorias, destaca-se a implementação de *roles* com diferentes níveis de acesso ao banco, criação de *views* para facilitar a geração de relatórios automatizados e populacionais, além da integração com ferramentas de visualização para apresentar os dados de forma mais dinâmica e interativa, como dashboards com gráficos em tempo real.

10. REFERÊNCIAS

BRASIL. PostgreSQL Global Development Group. **PostgreSQL: Documentation**. 2025. Disponível em: https://www.postgresql.org/docs/. Acesso em: junho 2025.

BRASIL. PostgreSQL Global Development Group. **PostgreSQL: Data Types**. 2025. Disponível em: https://www.postgresql.org/docs/current/datatype.html. Acesso em: junho 2025.

GEEKSFORGEEKS. **Types of Keys in Relational Model – Candidate, Super, Primary, Alternate and Foreign**. Atualizado em 2025. Disponível em: https://www.geeksforgeeks.org/dbms/types-of-keys-in-relational-model-candidate-super-primary-alternate-and-foreign/. Acesso em: junho 2025.

MANIFESTLY Blog. **Normalize Tables with Foreign Keys | Database Normalization**. 25 jan. 2025. Disponível em:

https://www.manifest.ly/blog/normalize-tables-with-foreign-keys-database-normalization/. Acesso em: junho 2025.