



FORMAÇÃO EM

DATA ANALYTICS

UNIDADE 1:

BANCO DE DADOS

MÓDULO 1:

> **BANCO DE DADOS RELACIONAIS** >

Sumário

Introdução	05
1.1 Objetivos dos sistemas de dados	05
1.2 Abstração de dados	06
1.3 Modelos de dados	07
1.3.1. Modelos Lógicos Baseados em Objetos	08
1.3.2. Modelos Lógicos Baseados em Registro	09
1.3.3. Modelos Físicos de Dados	09
1.4. Instâncias e Esquemas	10
1.5. Independência dos Dados	11
1.6. Linguagem de Definição de Dados (DDL)	12
1.7. Linguagem de Manipulação de Dados (DML)	13
1.8. Gerenciador de Banco de Dados	14
1.9. Administrador do Banco de Dados (DBA)	16
1.10. Usuários de Banco de Dados	17
1.11. Estrutura Geral do Sistema de Gerenciamento de Banco de Dados	20
2. Linguagem SQL	21
2.1. A estrutura básica de uma expressão SQL	23
2.1.1. Cláusulas Distinct e All	26
2.2. Variáveis Tuplas (Renomeação)	28
2.3. Operações de Conjuntos	29
2.4. Exercícios	31
2.5. Ordenando a exibição de Tuplas	32
2.6. Membros de conjuntos	33
2.7. Funções agregadas	36
2.8. Modificando o Banco de Dados	38

Sumário

2.9. Valores Nulos	41
3. Definição de Dados	42
3.1 Visões (views)	44
3.2 Restrições de integridade	46
3.2.1. Restrições de domínios	47
3.2.2. Restrições de integridade referencial	48
3.2.3. Asserções	51
3.3. Exercícios	52
3.4. O esquema banco	58
3.5. O esquema empresa	59
4. Modelagem de Dados	61
4.1. O que é modelagem de dados?	62
4.2. Como modelos de dados são usados na prática?	63
4.3. O que dizer sobre modelos conceituais?	65
4.4. Notações comuns de modelagem de dados	66
4.5. Como modelar dados	67
4.6. Identificar os tipos de entidade	68
4.7. Identificar atributos	69
4.8. Aplicar convenções de nome	70
4.9. Identificar relacionamentos	71
4.10. Associar chaves	72
4.11. Normalizar para reduzir redundância de dados	73
4.12. Modelagem de dados evolucionária/ágil	75
5. Instalando e Configurando o PostgreSQL	76
5.1. Conhecendo o PgAdmin	85

Sumário

6. Introdução ao PostgreSQL DDL	114
6.1. Create	115
6.2. Alter	116
6.3. Truncate	119
6.4. Drop	120
7. Data Modification Language - DML	121

Um Sistema de Gerenciamento de Banco de Dados (SGBD) consiste em uma coleção de dados inter-relacionados e em um conjunto de programas para acessá-los.

1. Introdução

1.1. Objetivos dos Sistemas de Banco de Dados

Um sistema que possui vários programas aplicativos acessando dados armazenados em arquivos convencionais apresenta uma série de problemas:

- **Redundância e inconsistência dos dados** – a redundância ocorre quando a mesma informação é mantida em diferentes arquivos, ou seja, quando há duplicação da informação. Isso pode gerar inconsistência dos dados, uma vez que podemos ter cópias diferentes do mesmo dado.
- **Dificuldade de acesso aos dados** – o ambiente convencional de processamento de arquivos não permite que dados necessários sejam recuperados de maneira conveniente e eficiente.
- **Isolamento dos dados** – uma vez que os dados estão espalhados em diversos arquivos e podem ter formatos diferentes, é difícil escrever programas para recuperar os dados adequados.
- **Anomalias de acesso concorrente** – o acesso de vários usuários simultaneamente pode gerar dados inconsistentes no sistema.
- **Problemas com segurança** – nem todo usuário deve ter acesso a todos os dados do sistema. Se programas aplicativos forem adicionados ao sistema de maneira arbitrária, é difícil assegurar restrições de segurança.

- **Problemas de integridade** – a integridade dos dados e do sistema é mantida pelos programas aplicativos, tornando-se, dessa forma, mais vulnerável a falhas. O problema torna-se mais complicado quando as restrições envolvem itens de dados e arquivos diferentes.

Esse grande número de problemas promoveu o desenvolvimento dos Sistemas Gerenciadores de Banco de Dados, que elimina todas essas complicações de forma eficiente.

1.2. Abstração dos Dados

O grande objetivo de um sistema de banco de dados é prover aos usuários uma visão abstrata dos dados, isto é, os detalhes de como os dados são armazenados e mantidos são escondidos do usuário. Não interessa ao usuário as complexas estruturas de armazenamento usadas para garantir a eficiência de acesso aos dados. Uma vez que muitos dos usuários de bancos de dados são leigos em computação, toda esta complexidade é escondida deles por meio de vários níveis de abstração que simplificam a interação do usuário com o sistema. Assim, os seguintes níveis são usados:

- **Nível físico** – é o nível mais baixo de abstração e descreve **COMO** os dados estão realmente armazenados. Num nível físico, complexas estruturas de dados de baixo nível são descritas em detalhe.
- **Nível conceitual** – este nível descreve **QUAIS** dados estão armazenados de fato no database e as relações que existem entre eles. O nível conceitual é usado por administradores do banco de dados, que podem decidir quais informações devem ser mantidas nesse banco.
- **Nível de visões** – a maioria dos usuários do sistema de banco de dados não está interessada em todas as informações existentes. Cada grupo de usuários deve enxergar somente os dados que lhe dizem respeito. Assim, cada grupo tem uma visão do database. O nível mais alto de abstração é composto de visões que cada grupo de usuários tem.

1.3. Modelos de Dados

Entende-se por **Modelo de Dados** uma coleção de ferramentas conceituais para descrição, relacionamentos e semântica de dados, e restrições de consistência.

Há vários modelos de dados que podem ser divididos em três grupos:

- Modelos lógicos baseados em objetos;
- Modelos lógicos baseados em registros;
- Modelos físicos de dados.

1.3.1. Modelos Lógicos Baseados em Objetos

- São usados na descrição de dados nos níveis conceitual e de visões.
- Fornecem, de modo conveniente, capacidades de estruturação flexíveis.
- Permitem especificar explicitamente restrições de dados.
- Existem vários modelos:
 - O modelo entidade-relacionamento;
 - O modelo orientado a objetos;
 - O modelo binário;
 - O modelo semântico de dados;
 - O modelo infológico;
 - O modelo funcional de dados.

Modelo Entidade-Relacionamento (E-R)

O modelo entidade-relacionamento é baseado numa percepção de um mundo real, que consiste em uma coleção de objetos básicos chamados entidades e em relacionamentos entre esses objetos. Alguns conceitos desse modelo são:

- Entidades;
- Relacionamentos;
- Atributos;
- Cardinalidade.

A estrutura lógica geral de um banco de dados pode ser expressa graficamente por um *diagrama E-R*.

Modelo Orientado a Objetos

Os objetos do mundo real são agrupados em classes, de acordo com suas características comuns, e cada instância é um objeto daquela classe.

- Objetos;
- Classes;
- Variáveis de instância;
- Métodos;
- Troca de mensagens;
- Hierarquia de classes.

1.3.2. Modelos Lógicos Baseados em Registro

- São usados, também, na descrição de dados nos níveis conceitual e de visões.
- O banco de dados é estruturado em registros de formato fixo de diversos tipos. Cada tipo de registro define um número fixo de campos (ou atributos) e cada campo é usualmente de um tamanho fixo.
- Não incluem um mecanismo para representação direta do código dentro do banco de dados.

Modelo Relacional

- Representa dados e relacionamentos entre dados por um conjunto de tabelas, cada uma tendo um número de colunas com nomes únicos.

Modelo de Redes

- Os dados no modelo rede são representados por coleções de registros, e os relacionamentos entre os dados são representados por ligações, que podem ser encaradas como ponteiros.
- Os registros no banco de dados são organizados como coleções de grafos arbitrários.

Modelo Hierárquico

- É similar ao modelo de redes, pois os dados e relacionamentos são representados por registros e ligações, respectivamente. A diferença é que os registros são organizados como coleções de árvores em vez de grafos arbitrários.

1.3.3. Modelos Físicos de Dados

Os modelos físicos são usados para descrever dados no nível mais baixo.

Dois dos mais conhecidos são:

- Modelo unificador (*Unifying Model*)
- Estrutura de memória (*Frame Memory*)

1.4. Instâncias e Esquemas

Uma **instância do banco de dados** é uma coleção de informações armazenadas em um determinado momento.

O **esquema do banco de dados** é o projeto geral do banco de dados, ou seja, a definição de todas as estruturas que compõem o database.

Ex: Instâncias e esquemas em banco de dados são similares aos conceitos de variáveis e seus valores de linguagens de programação.

```
var x : integer;
```

> esquema

```
x := 10;
```

> instância (valor da variável em um dado instante)

1.5. Independência dos Dados

O termo **independência dos dados** diz respeito à habilidade de modificar a definição de um esquema em um nível sem afetar a definição do esquema num nível mais alto.

A independência dos dados pode ser de dois tipos:

- **Independência física dos dados**

Habilidade de modificar o esquema físico sem modificar os programas aplicativos.

- **Independência lógica dos dados**

Habilidade de modificar o esquema lógico sem modificar os programas aplicativos.

Obs.: A independência lógica é mais difícil de ser alcançada, uma vez que os programas são muito dependentes da estrutura lógica.

1.6. Linguagem de Definição de Dados (DDL)

Entende-se por **Linguagem de Definição de Dados (DDL - Data Definition Language)** o conjunto de comandos que permitem definir esquemas do banco de dados, logo, esta linguagem vai atuar na própria estrutura desse banco.

Entre as operações de definição de dados estão:

- criar tabelas, índices e visões;
- alterar a estrutura de uma tabela;
- remover tabelas, índices e visões.

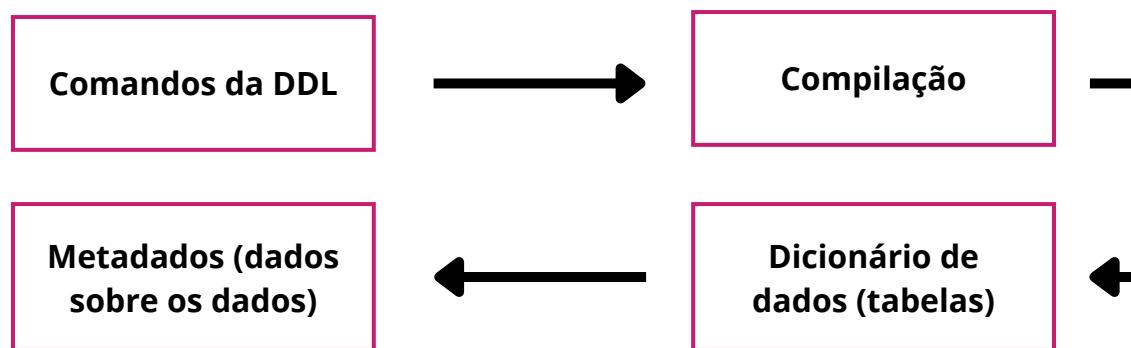


Gráfico 1

Um tipo especial de DDL é a **Linguagem de Armazenagem e Definição de Dados** que permite a definição da estrutura de armazenamento e dos métodos de acesso do banco de dados.

1.7. Linguagem de Manipulação de Dados (DML)

Entende-se por **Linguagem de Manipulação de Dados (DML - Data Manipulation Language)** o conjunto de comandos que permitem acessar e manipular os dados, logo, esta linguagem vai atuar sobre os dados propriamente ditos armazenados no banco de dados.

Manipular significa:

- buscar informações;
- inserir informações;
- eliminar informações;
- modificar informações.

Há dois tipos de linguagens de manipulação de dados:

- DMLs procedurais – o usuário especifica qual dado quer e diz como obtê-lo.
- DMLs não-procedurais – o usuário especifica qual dado quer, **sem** dizer como obtê-lo.
 - Mais fáceis de aprender e usar.

Uma **consulta** é um comando requisitando a busca de uma informação.

A **linguagem de consulta** é a parte da DML que envolve a busca de informações.

1.8. Gerenciador de Banco de Dados

Os SGBDs trabalham com grandes quantidades de informação armazenada, na área dos GB (gigabyte) e TB (terabyte).

$1\text{ GB} \cong 1.000\text{ MB} \cong 1.000.000\text{ KB} \cong 1.000.000.000$ (1 bilhão de bytes)
 $1\text{ TB} \cong 1\text{ trilhão de bytes}$

Esta grande quantidade de informação deve ser acessada de maneira eficiente (desempenho), para isso faz-se uso de complexas estruturas de dados. Esses dados são armazenados nos discos rígidos e, como a transferência de dados entre a memória principal e os discos é bastante lenta em comparação com a velocidade da CPU, o sistema de banco de dados deve estruturar esses dados de forma a minimizar o movimento deles entre os discos e a memória principal.

Deve-se ter em mente que o fator principal na satisfação de um usuário de banco de dados, bem como de qualquer sistema computacional, é seu **desempenho**. De nada adianta uma bela interface se o sistema é lento.

O desempenho de um sistema de banco de dados depende de dois fatores:

- da eficiência das estruturas de dados usadas para representar esses dados;
- de quão eficientemente o sistema é capaz de operar essas estruturas de dados.

O **Gerenciador de Banco de Dados** é um módulo de programa que faz parte do SGDB e que provê a interface entre os dados de baixo nível armazenados num banco de dados e os programas de aplicação e as solicitações submetidas ao sistema.

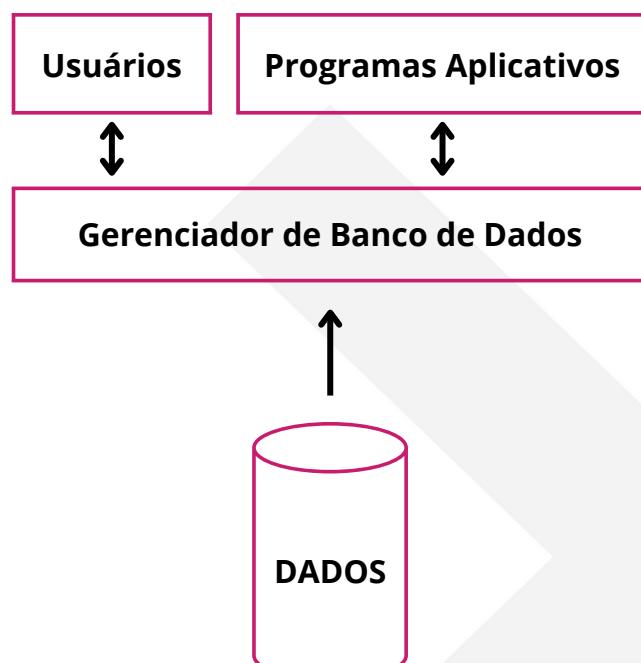


Gráfico 2

O gerenciador de banco de dados é responsável pelas seguintes tarefas:

1. Interação com o gerenciador de arquivos do sistema operacional
 - o Armazenamento dos dados;
 - o Busca dos dados;
 - o Atualização dos dados;
2. Cumprimento da integridade
3. Cumprimento da segurança (nem todo usuário precisa ver todo o banco de dados)
4. Cópias de reserva (backup) e recuperação
5. Controle de concorrência.

É importante notar que o porte do sistema de banco de dados depende do porte da empresa ou da aplicação. Sistemas de bancos de dados projetados para uso em pequenos computadores pessoais podem não ter todos os recursos descritos anteriormente.

1.9. Administrador do Banco de Dados (DBA)

Uma das principais razões para usar um banco de dados é a necessidade de ter controle central desses dados e dos programas de acesso a eles.

O **Administrador de Banco de Dados (DBA - DataBase Administrator)** é a pessoa que centraliza o controle sobre os registros e programas de acesso num sistema de banco de dados.

As funções do administrador do banco de dados incluem:

- Definição de esquemas (usando DDL);
- Definição de estruturas de armazenamento e métodos de acesso;
- Modificação de esquemas e da organização do armazenamento físico;
- Concessão de autorização para acesso aos dados;
- Especificação de restrições de integridade.

1.10. Usuários de Banco de Dados

Há, basicamente, quatro tipos de usuários de um sistema de banco de dados:

1. Programadores de aplicativos (ou de aplicação) – profissionais que escrevem programas em uma linguagem de programação qualquer, contendo, além dos comandos próprios da linguagem de programação, outros comandos de acesso ao banco de dados.

Essas linguagens de programação são chamadas de linguagens hospedeiras ou linguagens host (host languages) por hospedarem comandos de banco de dados estranhos ao seu ambiente.

Os comandos de banco de dados estranhos ao ambiente da linguagem host são denominados chamadas à DML embutidas, por estarem embutidas (inseridas) na linguagem host.

Uma chamada a um comando da DML embutida se faz usando um caractere (ou comando) especial, que varia de sistema para sistema. No Oracle, por exemplo, temos o comando EXEC SQL, que precede qualquer comando de acesso ao banco de dados. Veja o trecho de programa abaixo contendo uma **chamada à DML embutida** numa linguagem de programação tipo Pascal:

```
a := b + c;
```

> comando da linguagem hospedeira

```
EXEC SQL SELECT * FROM TABELA1;
```

> chamada à DML embutida

```
while (a < d) do a := a + 1;
```

> outro comando da linguagem hospedeira

O processo de compilação procede-se da seguinte maneira:

- O programa com chamadas à DML embutidas passa pelo pré-compilador da DML, que transforma as chamadas à DML embutidas em chamadas de procedimentos normais na linguagem hospedeira;
- O programa pré-compilado, resultado da operação anterior, vai para o compilador da linguagem hospedeira, que efetua a compilação de forma normal, usando as bibliotecas do sistema de banco de dados e gerando o código-objeto final.

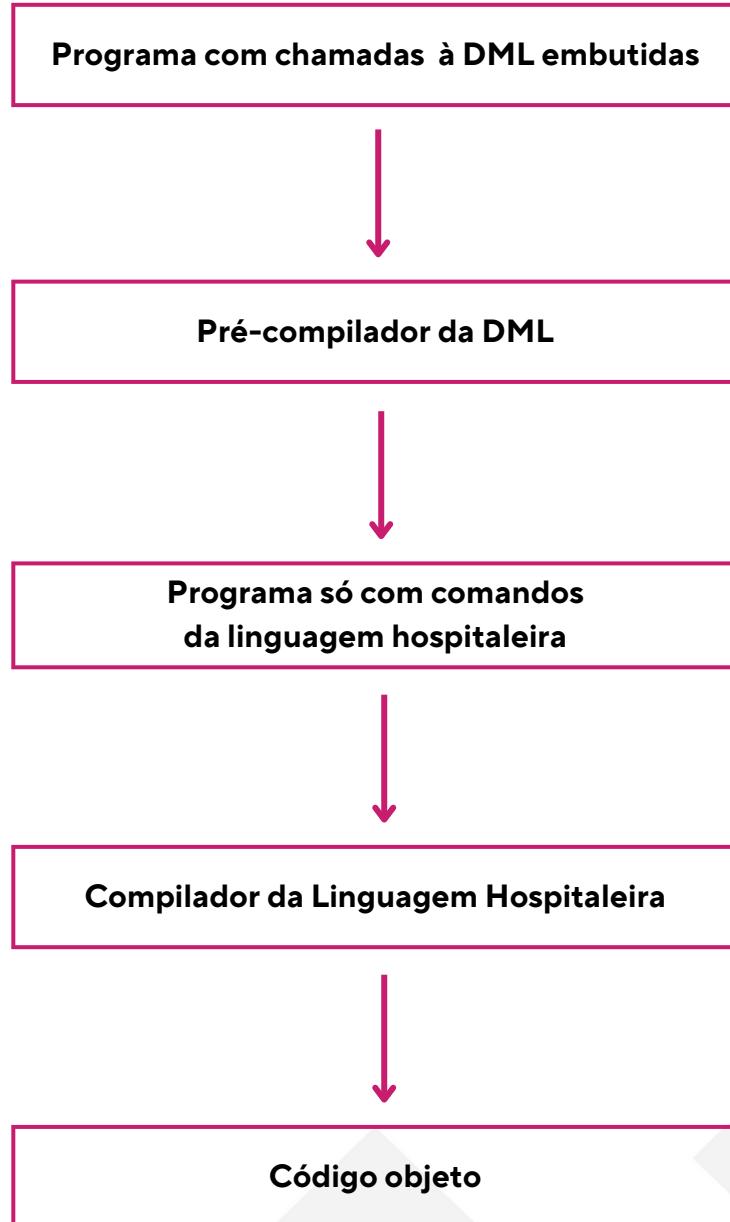


Gráfico 3

Existem, atualmente, as chamadas linguagens de 4^a geração, que combinam as estruturas de controle das linguagens de programação com as estruturas de controle para manipulação de objetos do banco de dados. Tais linguagens são comumente usadas para criação de formulários (telas) e relatórios.

2. Usuários de alto nível – não escrevem programas, mas escrevem consultas em linguagens de consulta ao banco de dados.
3. Usuários especializados – escrevem programas especializados (CAD, CAM, sistemas especialistas, sistemas de dados complexos).
4. Usuários ingênuos ou leigos – utilizam os programas aplicativos.

1.11. Estrutura Geral do Sistema de Gerenciamento de Banco de Dados

Um sistema de gerenciamento de banco de dados é composto das seguintes partes:

- Gerenciador de arquivos
 - Gerencia a alocação do espaço de armazenamento.
 - Gerencia as estruturas de dados usadas para representar os dados armazenados no disco.
- Gerenciador de banco de dados
 - Fornece a interface entre dados de baixo nível (em disco) e programas aplicativos, bem como consultas de usuários.
- Processador de consultas
 - Traduz comandos numa linguagem de consulta para instruções que o gerenciador de banco de dados entende.
 - É responsável pela otimização de consultas.
- Pré-compilador da DML
 - Traduz comandos da DML embutidos para chamadas de procedimento normais na linguagem hospedeira (linguagem "host").
 - Interage com o processador de consultas.
- Compilador da DDL
 - Converte comandos da DDL em tabelas contendo metadados.

Além desses componentes, algumas estruturas de dados são usadas para implementação do sistema físico:

- *Arquivos de dados* – armazenam o banco de dados propriamente dito.
- *Dicionário de dados* – armazena os metadados, isto é, os dados sobre os dados.
- *Índices* – estrutura que proporciona acesso rápido aos dados.

2. Linguagem SQL

Quando os Bancos de Dados Relacionais estavam sendo desenvolvidos, foram criadas linguagens destinadas à sua manipulação. O Departamento de Pesquisas da IBM, desenvolveu a SQL como forma de interface para o sistema de BD relacional denominado SYSTEM R, no início dos anos 70. Em 1986 o American National Standard Institute (ANSI) publicou um padrão SQL. A SQL estabeleceu-se como linguagem padrão de Banco de Dados Relacional.

SQL apresenta uma série de comandos que permitem a definição dos dados, chamada de **DDL (Data Definition Language)**, composta, entre outros, pelos comandos Create, que são destinados à criação do Banco de Dados, das Tabelas que o compõem, e também das relações existentes entre as tabelas. Como exemplo de comandos da classe DDL, temos os comandos Create, Alter e Drop.

Os comandos da série **DML (Data Manipulation Language)** são destinados a consultas, inserções, exclusões e alterações em um ou mais registros de uma ou mais tabelas de maneira simultânea. Como exemplo de comandos da classe DML, temos os comandos Select, Insert, Update e Delete. Uma subclasse de comandos DML, a **DCL (Data Control Language)**, dispõe de comandos de controle como Grant e Revoke.

A Linguagem SQL tem como grandes virtudes sua capacidade de gerenciar índices sem a necessidade de controle individualizado de índice corrente, algo muito comum nas linguagens de manipulação de dados do tipo "registro a registro" (dos modelos Hierárquico e Redes).

Outra característica muito importante disponível em SQL é sua capacidade de construção de visões, que são modos de visualizar os dados na forma de listagens, independentemente das tabelas e da organização lógica dos dados.

Outra característica interessante na linguagem SQL é a capacidade que se tem de cancelar uma série de atualizações ou de gravá-las depois de iniciar uma sequência de atualizações. Os comandos Commit e Rollback são responsáveis por estas facilidades.

Existem inúmeras versões de SQL e sua versão original foi desenvolvida no Laboratório de Pesquisa da IBM. Essa linguagem, originalmente chamada Sequel, foi implementada como parte do projeto System R no início dos anos 70. A linguagem evoluiu desde então, e seu nome foi mudado para SQL (Structured Query Language).

Em 1986, o American National Standard Institute (ANSI) publicou um padrão SQL, enquanto a IBM publicou o seu próprio SQL standard, o *Systems Application Architecture Database Interface* (SAA-SQL) em 1987.

Em 1989, tanto ANSI quanto ISO publicaram padrões substitutos (ANSI X3.135-1989 e ISO/IEC 9075:1989) que aumentaram a linguagem e acrescentaram uma capacidade opcional de integridade referencial, permitindo que projetistas de bancos de dados pudessem criar relacionamentos entre dados em diferentes partes do banco de dados. A versão em uso do padrão ANSI / ISO SQL é o padrão SQL-92 (ANSI X3.135-1992), mas algumas aplicações atuais dão suporte apenas ao padrão SQL-89.

Desde 1993 há trabalhos sendo desenvolvidos para atualizar o padrão de modo para que este atenda às características das últimas versões de bancos de dados relacionais lançadas no mercado. A principal inovação da nova versão (chamada provisoriamente de SQL3) é o suporte à orientação a objetos.

Algumas das características da linguagem SQL são:

- Permite trabalhar com várias tabelas;
- Permite utilizar o resultado de uma instrução SQL em outra instrução SQL (sub-queries);
- Não necessita especificar o método de acesso ao dado;
- É uma linguagem para vários usuários, como:
 - Administradores do sistema;
 - Administradores do banco de dados (DBA);
 - Programadores de aplicações;
 - Pessoal da agência e de tomada de decisão;
- É de fácil aprendizado;
- Permite a utilização dentro de uma linguagem procedural como C, COBOL, FORTRAN, Pascal e PL/I - SQL embutida.

2.1. A estrutura básica de uma expressão SQL

A estrutura básica de uma expressão SQL consiste em três cláusulas: **select**, **from** e **where**.

- A cláusula **select** corresponde à operação projeção da álgebra relacional. É usada para listar os atributos desejados no resultado de uma consulta.
- A cláusula **from** corresponde à operação produto cartesiano da álgebra relacional. Ela lista as relações a serem examinadas na avaliação da expressão.
- A cláusula **where** corresponde ao predicado de seleção da álgebra relacional. Consiste em um predicado envolvendo atributos de relações que aparecem na cláusula **from**.

Uma típica consulta SQL segue a seguinte ordem:

```
select a1, a2, ..., an β 3a
from T1, T2, ..., Tm β 1a
where P β 2a
```

Cada ai representa um atributo, cada Ti é uma relação, e cada P é um predicado. Esta consulta é equivalente à expressão da seguinte álgebra relacional

```
p a1, a2, ..., an (s P (T1 x T2 x ... x Tm))
```

O SQL forma o produto cartesiano das relações chamadas na cláusula from, executa uma seleção da álgebra relacional usando o predicado da cláusula where e, então, projeta o resultado para os atributos da cláusula select. Na prática, o SQL pode converter esta expressão em uma forma equivalente que pode ser processada mais eficientemente.

Exemplo

No esquema BANCO, encontre os nomes de todos os clientes de 'Vitória'.

```
select cliente_nome
from CLIENTE
where cidade = 'Vitória'
```

Encontre os nomes de todos os clientes.

```
select cliente_nome
from CLIENTE
```

Exemplo

No esquema EMPRESA, selecionar o nome e o RG dos funcionários que trabalham no departamento número 2 na tabela EMPREGADO.

```
select nome, rg
from EMPREGADOS
where depto = 2;
```

A partir disso, obtém-se o seguinte resultado:

NOME	RG
Fernando	20202020
Ricardo	30303030
Jorge	40404040

A consulta acima é originária da seguinte função em álgebra relacional:

```
πnome, rg (σdepto = 2 (EMPREGADOS));
```

Em SQL também é permitido o uso de condições múltiplas. Veja o exemplo a seguir:

```
select nome, rg, salario
from EMPREGADOS
where depto = 2 AND salario > 2500.00;
```

O que fornece o seguinte resultado

NOME	RG	SALÁRIO
Jorge	40404040	R\$4.200,00

Originário da seguinte função em álgebra relacional

```
nome, rg, salario (σdepto = 2 .and. salario > 3500.00(EMPREGADOS));
```

O operador * dentro do especificador *select* seleciona todos os atributos de uma tabela, enquanto que a exclusão do especificador *where* faz com que todas as tuplas de uma tabela sejam selecionadas. Desta forma, a expressão

```
select *
from empregado;
```

gera o seguinte resultado:

Nome	RG	CPF	Deptº	RG Supervisor	Salário
João Luiz	10101010	11111111	1	NULO	R\$ 3000,00
Fernando	20202020	22222222	2	10101010	R\$ 2500,00
Ricardo	30303030	33333333	2	10101010	R\$ 2300,00
Jorge	40404040	44444444	2	20202020	R\$ 4.200,00
Renato	50505050	55555555	3	20202020	R\$ 1300,00

2.1.1. Cláusulas Distinct e All

Diferentemente de álgebra relacional, a operação *select* em SQL permite a geração de tuplas duplicadas como resultado de uma expressão. Para evitar isto, devemos utilizar a cláusula **distinct**.

Exemplo:

Sejam as seguintes consultas no esquema EMPRESA

```
select depto
from EMPREGADO;
```

```
select distinct depto
from EMPREGADO;
```

DEPTO	DEPTO
1	1
2	2
2	3
2	
3	

Gera-se o seguinte resultado:

A cláusula All é o default para o select, ou seja, Select All indica para obter todas as tuplas. Logo, esta cláusula não precisa ser colocada (a não ser, talvez, por motivos de documentação).

Predicados e ligações

O SQL não tem uma representação da operação ligação natural. No entanto, uma vez que a ligação natural é definida em termos de um produto cartesiano, uma seleção e uma projeção, é relativamente simples escrever uma expressão SQL para uma ligação natural.

Exemplo:

Encontre os nomes e as cidades de clientes que possuam empréstimos em alguma agência.

```
Select distinct cliente_nome, cidade
From Cliente, Emprestimo
Where Cliente.cliente_cod=Emprestimo.cliente_cod
```

O SQL inclui os conectores **and**, **or** e **not**; caracteres especiais: (,), ., :, _, %<, >, <= , >= , = , <>, +, - , * e /; e operador para comparação: **between**, como mostra o exemplo a seguir.

Exemplo:

Selecionar todas as contas que possuam saldo entre 10000 e 20000.

```
Select conta_numero  
From CONTA  
Where saldo >= 10000 and saldo <= 20000
```

Que equivale à consulta

```
Select conta_numero  
From CONTA  
Where saldo between 10000 and 20000
```

A SQL inclui também um operador para comparações de cadeias de caracteres, o **like**. Ele é usado em conjunto com dois caracteres especiais:

- Por cento (%): substitui qualquer subcadeia de caracteres.
- Sublinhado (_): substitui qualquer caractere.

Exemplo:

Encontre os nomes de todos os clientes cujas ruas incluem a subcadeia 'na'

```
Select distinct cliente_nome  
From CLIENTE  
Where rua like '%na%'
```

Exemplo:

Encontre os nomes de todos os clientes cujas ruas finalizam com a subcadeia 'na', seguido de um caractere.

```
Select distinct cliente_nome  
From CLIENTE  
Where rua like '%na_'
```

Para que o padrão possa incluir os caracteres especiais (isto é, % , _ , etc...), a SQL permite a especificação de um caractere de escape. O caractere de escape é usado imediatamente antes de um caractere especial para indicar que o caractere especial deverá ser tratado como um caractere normal. Define-se o caractere de escape para uma comparação **like** usando a palavra-chave **escape**. Para ilustrar, considere os padrões seguintes que utilizam uma barra invertida como caractere de escape.

- Like ' ab\%cd%' escape '\' substitui todas as cadeias começando com ' ab%cd';
- Like ' ab_cd%' escape '\' substitui todas as cadeias começando com ' ab_cd'.

A procura por não-substituições em vez de substituições dá-se por meio do operador **not like**.

2.2. Variáveis Tuplas (Renomeação)

Exemplo:

No esquema EMPRESA, selecione o número do departamento que controla projetos localizados em Rio Claro;

```
select t1.numero_dept
from departamento_projeto as t1, projeto as t2
where t1.numero_projeto = t2.numero;
```

Na expressão SQL acima, *t1* e *t2* são chamados “*alias*” (apelidos) e representam a mesma tabela a qual estão referenciando. Um *alias* é muito importante quando há redundância nos nomes das colunas de duas ou mais tabelas que estão envolvidas em uma expressão. Em vez de utilizar o “*alias*”, é possível utilizar o nome da tabela, mas isto pode ficar cansativo em consultas muito complexas, além do fato que essa ação impossibilitaria a utilização da mesma tabela mais de uma vez em uma expressão SQL. A palavra chave **as** é opcional.

Exemplo:

No esquema EMPRESA, selecione o nome e o RG de todos os funcionários que são supervisores.

```
select e1.nome as "Nome do Supervisor", e1.rg as "RG do Supervisor"
from empregado e1, empregado e2
where e1.rg = e2.rg_supervisor;
```

O que gera o seguinte resultado:

Nome do supervisor	RG do Supervisor
João Luiz	10101010
Fernando	20202020

Observação: a consulta acima é decorrente da seguinte expressão em álgebra relacional:

```
πnome, rg (EMPREGADOS |x| tg_t1 = rg_supervisor_t2 EMPREGADOS)
```

Exemplo:

Encontre o nome e a cidade de todos os clientes com uma conta em qualquer agência.

```
Select distinct C.cliente_nome, C.cidade
from CLIENTE C, CONTA S
where C.cliente_cod = S.cliente_cod
```

2.3. Operações de Conjuntos

A SQL inclui as operações de conjunto **union**, **intersect** e **minus** que operam em relações e correspondem às operações , e – da álgebra relacional.

Uma vez que as relações são conjuntos, na união destas, as linhas duplicadas são eliminadas. Para que uma operação $R \cup S$, $R \cap S$ ou $R - S$ seja válida, necessita-se que duas condições sejam cumpridas:

- As relações R e S devem ter o mesmo número de atributos;
- Os domínios do i-ésimo atributo de R e do i-ésimo atributo de S devem ser os mesmos.

Observação: nem todos os interpretadores SQL suportam todas as operações de conjunto. Embora a operação union seja relativamente comum, são raros os que suportam intersect ou minus.

Exemplo:

Mostrar o nome dos clientes que possuem conta, empréstimo ou ambos na agência de código '051':
Empréstimo na agência '051':

```
Select distinct cliente_nome
From CLIENTE, EMPRESTIMO
Where CLIENTE.cliente_cod= EMPRESTIMO.cliente_cod and agencia_cod = '051'
```

Conta na agência '051':

```
Select distinct cliente_nome
From CLIENTE, CONTA
Where CLIENTE.cliente_cod= CONTA.cliente_cod and CONTA.agencia_cod = '051'
```

Fazendo a união dos dois:

```
(Select distinct cliente_nome
From CLIENTE, EMPRESTIMO
Where CLIENTE.cliente_cod= EMPRESTIMO.cliente_cod and agencia_cod = '051' )

Union

(Select distinct cliente_nome
From CLIENTE, CONTA
Where CLIENTE.cliente_cod= CONTA.cliente_cod and CONTA.agencia_cod = '051'
)
```

Exemplo

Achar todos os clientes que possuam uma conta e um empréstimo na agência de código '051'.

```
(Select distinct cliente_nome
From CLIENTE, EMPRESTIMO
Where CLIENTE.cliente_cod= EMPRESTIMO.cliente_cod and agencia_cod = '051' )
intersect
(Select distinct cliente_nome
From CLIENTE, CONTA
Where CLIENTE.cliente_cod= CONTA.cliente_cod and CONTA.agencia_cod = '051' )
```

Exemplo

Achar todos os clientes que possuem uma conta mas não possuem um empréstimo na agência de código '051'.

```
(Select distinct cliente_nome
From CLIENTE, EMPRESTIMO
Where CLIENTE.cliente_cod=Emprestimos.cliente_cod and
EMPRESTIMO.agencia_cod = '051' )
minus
(Select distinct cliente_nome
From CLIENTE, CONTA
Where CLIENTE.cliente_cod= CONTA.cliente_cod and CONTA.agencia_cod = '051'
)
```

2.4. Exercícios

Baseado no esquema EMPRESA, faça as seguintes consultas SQL:

1. Selecione todos os empregados com salário maior ou igual a 2000,00.
2. Selecione todos os empregados cujos nomes começam com 'J'.
3. Mostre todos os empregados que têm 'Luiz' ou 'Luis' no nome.
4. Mostre todos os empregados do departamento de 'Engenharia Civil'.
5. Mostre todos os nomes dos departamentos envolvidos com o projeto 'Motor 3'.
6. Liste o nome dos empregados envolvidos com o projeto 'Financeiro 1'.
7. Mostre os funcionários cujo supervisor ganha entre 2000 e 2500.

Baseado no esquema BANCO, faça as seguintes consultas SQL:

1. Liste todas as cidades onde há agências.
2. Liste todas as agências existentes em 'Vitória'.
3. Liste todas as agências que possuem conta com saldo maior que 100.000,00.
4. Mostre os dados de todos os clientes da agência 'Centro' da cidade de 'Vitória' que têm saldo negativo.
5. Liste os dados dos clientes que possuem conta, mas não possuem empréstimo na agência de código '005'.

2.5. Ordenando a exibição de tuplas (ORDER BY)

A cláusula **order by** ocasiona o aparecimento de tuplas no resultado de uma consulta em uma ordem determinada. Para listar em ordem alfabética todos os clientes do banco, faz-se:

```
Select distinct cliente_nome  
From CLIENTE  
Order by cliente_nome
```

Como padrão, SQL lista tuplas na ordem ascendente. Para especificar a ordem de classificação, podemos especificar **asc** para ordem ascendente e **desc** para descendente. Podemos ordenar uma relação por mais de um elemento. Como se segue:

```
Select *  
From EMPRESTIMO  
Order by quantia desc, agencia_cod asc
```

Para colocar as tuplas (linhas) em ordem, é realizada uma operação de *sort*. Esta operação é relativamente custosa e, portanto, só deve ser usada quando realmente for necessário.

2.6. Membros de Conjuntos

O conectivo **in** testa os membros de conjunto, onde o conjunto é uma coleção de valores produzidos por uma cláusula select. Da mesma forma, pode ser usada a expressão **not in**.

Exemplo:

Selecione todas as agências com código 1, 2 ou 3.

```
select *
from agencia
where agencia_cod in (1,2,3)
```

Exemplo

Selecione o nome de todos os funcionários que trabalham em projetos localizados em Rio Claro.

```
select e1.nome, e1.rg, e1.depto
from empregado e1, empregado_projeto e2
where e1.rg = e2.rg_empregado and e2.numero_projeto in (select numero
from projeto
where localizacao = 'Rio Claro');
```

Exemplo

Encontre todos os clientes que possuem uma conta e um empréstimo na agência 'Princesa Isabel'.

```
Select distinct cliente_nome
From CLIENTE
Where CLIENTE.cliente_cod in

(select cliente_cod
from CONTA, AGENCIA
where CONTA.agencia_cod = AGENCIA.agencia_cod and agencia_nome =
'Princesa Isabel')
and CLIENTE.cliente_cod in

(select cliente_cod

from EMPRESTIMO, AGENCIA
where EMPRESTIMO.agencia_cod= AGENCIA.agencia_cod and agencia_nome =
'Princesa Isabel')
```

Exemplo

Encontre todas as agências que possuem ativos maiores que alguma agência de Vitória.

```
select distinct t.agencia_nome
from AGENCIA t, AGENCIA s
where t.ativos > s.ativos and s.cidade = 'Vitória'
```

Observação: Uma vez que isto é uma comparação "maior que", não se pode escrever a expressão usando a construção **in**.

A SQL oferece o operador **some** (equivalente ao operador **any**), usado para construir a consulta anterior. São aceitos pela linguagem: **>some, <some, >=some, <=some, =some**

```
select agencia_nome
from AGENCIA
where ativos > some
  (select ativos
   from AGENCIA
   where agencia_cidade = 'Vitória')
```

Como o operador **some**, o operador **all** pode ser usado como: **>all, <all, >=all, <=all, =all e <>all**. A construção **> all** corresponde à frase “maior que todos”.

Exemplo:

Encontrar as agências que possuem ativos maiores do que todas as agências de Vitória.

```
select agencia_nome
from AGENCIA
where ativos > all
  (select ativos
   from AGENCIA
   where agencia_cidade = 'Vitória')
```

A SQL possui um recurso para testar se uma subconsulta tem alguma tupla em seus resultados. A construção **exists** retorna **true** se o argumento subconsulta está não-vazio. Pode-se usar também a expressão **not exists**.

Exemplo:

No esquema EMPRESA, liste os nomes dos gerentes que têm ao menos um dependente.

```
Select nome
from EMPREGADO
where exists
  (select *
   from DEPENDENTES
   where DEPENDENTES.rg_responsavel = EMPREGADO.rg)
and exists
  (select *
   from DEPARTAMENTO
   where DEPARTAMENTO.rg_gerente = EMPREGADO.rg)
```

Exemplo

Usando a construção **exists**, encontre todos os clientes que possuem uma conta e um empréstimo na agência 'Princesa Isabel'.

```
Select cliente_nome
from CLIENTE
where exists
  (select *
   from CONTA, AGENCIA
   where CONTA.cliente_cod= CLIENTE.cliente_cod and AGENCIA.agencia_cod
         = CONTA.agencia_cod and agencia_nome = 'Princesa Isabel')
and exists
(select *
from EMPRESTIMO, AGENCIA
where EMPRESTIMO.cliente_cod= CLIENTE.cliente_cod and AGENCIA.agencia_cod =
EMPRESTIMO.agencia_cod and agencia_nome = 'Princesa Isabel')
```

2.7. Funções Agregadas

A SQL oferece a habilidade para computar funções em grupos de tuplas usando a cláusula **group by**. Os(s) atributo(s) dados na cláusula group by são usados para formar grupos. Tuplas com o mesmo valor em todos os atributos na cláusula group by são colocadas em um grupo. A SQL inclui funções para computar:

Média: **avg** // Mínimo: **min** // Máximo: **max** // Soma: **sum** // Contar: **count**

Exemplo:

Encontre o saldo médio de conta em cada agência.

```
Select agencia_nome, avg(saldo)
From CONTA, AGENCIA
where CONTA.agencia_cod = AGENCIA.agencia_cod
Group by agencia_nome
```

Exemplo:

Encontre o número de depositantes de cada agência.

```
Select agencia_nome, count(distinct cliente_nome)
From CONTA, AGENCIA
where CONTA.agencia_cod = AGENCIA.agencia_cod
Group by agencia_nome
```

Observação: Note que nesta última consulta é importante a existência da cláusula *distinct*, pois um cliente pode ter mais de uma conta em uma agência, mas esse dado só deverá ser contado uma única vez.

Exemplo:

Encontre o maior saldo de cada agência.

```
Select agencia_nome, max(saldo)
From CONTA, AGENCIA
Where CONTA.agencia_cod= AGENCIA.agencias_cod
Group by agencia_nome
```

Às vezes, é útil definir uma condição que se aplique a grupos em vez de tuplas. Por exemplo, poderia-se ter interesse apenas em agências nas quais a média dos saldos é maior que 1200. Nesse caso, essa condição seria aplicada a cada grupo e não a tuplas simples, e seria definida por meio da cláusula **having**. Expressa-se essa consulta em SQL assim:

```
Select agencia_nome, avg(saldo)
From CONTA, AGENCIA
Where CONTA.agencia_cod= AGENCIA.agencias_cod
Group by agencia_nome Having avg(saldo)>1200
```

Às vezes, deseja-se tratar a relação inteira como um grupo simples. Nesses casos, não se utiliza a cláusula **group by**.

Exemplo:

Encontre a média de saldos de todas as contas.

```
Select avg(saldo)  
From CONTA
```

2.8. Modificando o banco de dados.

• Remoção

Pode-se remover somente tuplas inteiras, não se pode remover valores apenas em atributos particulares.

Sintaxe:

```
Delete R  
Where P
```

em que R representa uma relação, e P representa um predicado.

Note que o comando **delete** opera em apenas uma relação. O predicado da cláusula **where** pode ser tão complexo como o predicado **where** do comando **select**.

Exemplo:

Remover todas as tuplas de empréstimo.

```
delete EMPRESTIMO
```

Exemplo:

Remover todos os registros da conta de 'João'.

```
delete CONTA  
where cliente_cod in  
      (select cliente_cod  
       from CLIENTE  
       where cliente_nome = 'João')
```

Exemplo

Remover todos os empréstimos com números entre 1300 e 1500.

```
delete EMPRESTIMO  
where emprestimo_numero between 1300 and 1500
```

Exemplo

Remover todas as contas de agências localizadas em 'Vitória'.

```
Delete CONTA  
where agencia_cod in (select  
                      agencia_cod  
                      from AGENCIA  
                      where agencia_cidade='Vitoria')
```

- **Inserção**

Para inserir um dado em uma relação ou especificar uma tupla para ser inserida, escreve-se uma consulta cujo resultado seja um conjunto de tuplas a serem inseridas. Os valores dos atributos para tuplas inseridas precisam necessariamente ser membros do mesmo domínio do atributo.

Exemplo

Inserir uma nova conta para João (código = 1), número 9000, na agência de código=2 cujo valor seja 1200.

```
insert into CONTA  
values (2,9000,1,1200)
```

Na inserção acima, é considerada a ordem na qual os atributos correspondentes estão listados no esquema *relação*. Caso o usuário não se lembre da ordem desses atributos, pode fazer o mesmo comando da seguinte forma:

```
insert into CONTA (agencia_cod, conta_numero, cliente_cod, saldo)  
values (2,9000,1,1200)
```

Pode-se querer também inserir tuplas baseadas no resultado de uma consulta.

Exemplo:

Inserir todos os clientes que possuam empréstimos na agência 'Princesa Isabel' na relação CONTA com um saldo de 200. O número da nova conta é o número do empréstimo * 10000.

```
insert into CONTA (agencia_cod, conta_numero, cliente_cod, saldo) select  
AGENCIA.agencia_cod, emprestimo_numero*10000, cliente_cod, 200 from  
EMPRESTIMO, AGENCIA  
where EMPRESTIMO.agencia_cod= AGENCIA.agencia_cod and  
agencia_nome = 'Princesa Isabel'
```

- **Atualizações**

Em certas situações, pode-se desejar mudar um valor em uma tupla sem mudar todos os valores na tupla. Para isso, o comando **update** pode ser usado.

Suponha que esteja sendo feito o pagamento de juros e que em todos os saldos sejam acrescentados 5%. Escreve-se

```
update CONTA  
set saldo = saldo * 1,05
```

Suponha que todas as contas com saldo superiores a 10000 recebam aumento de 6% e as demais de 5%.

```
Update CONTA
set saldo = saldo * 1,06
where saldo >10000
```

```
Update CONTA
set saldo = saldo * 1,05
where saldo<=10000
```

A cláusula **where** pode conter uma série de comandos select aninhados. Considere, por exemplo, que todas as contas de pessoas que possuem empréstimos no banco terão acréscimo de 1%.

```
Update CONTA
set saldo = saldo * 1,01
where cliente_cod in (select
    cliente_cod
    from EMPRESTIMO )
```

Exemplo

No esquema EMPRESA, atualize o salário de todos os empregados que trabalham no departamento 2 para R\$ 3.000,00.

```
update empregado
set salario = 3000
where depto = 2;
```

Exemplo

No esquema BANCO, atualize o valor dos ativos. Os ativos são os valores dos saldos das contas da agência.

```
update agencia
set ativos =
    (select sum(saldo)
     from conta
     where conta.agencia_cod = agencia.agencia_cod)
```

2.9. Valores Nulos

É possível dar valores a apenas alguns atributos do esquema para tuplas inseridas em uma dada relação. Os atributos restantes são designados como nulos. Considere a requisição:

```
insert into CLIENTE (cliente_cod, cliente_nome, rua, cidade)
values (123, 'Andrea', null, null)
```

A palavra chave `null` pode ser usada em um predicado para testar se um valor é nulo. Assim, para achar todos os clientes que possuem valores nulos para rua, escrevemos:

```
select distinct cliente_nome
from CLIENTE
where rua is null
```

O predicado **is not null** testa a ausência de um valor nulo.

3. Definição de dados.

O conjunto de relações de um Banco de Dados precisa ser especificado ao sistema por meio de uma linguagem de definição de dados - DDL. A SQL DDL permite a especificação não apenas de um conjunto de relações, mas também de informações sobre cada relação, incluindo:

- Esquema para cada relação;
- Domínio de valores associados a cada atributo;
- Conjunto de índices a ser mantido para cada relação;
- Restrições de integridade;
- A estrutura física de armazenamento de cada relação no disco.

Uma relação SQL é definida usando o comando **create table**. A forma geral do comando **create table** então é:

```
create table <nome_tabela> (
    <nome_coluna1> <tipo_coluna1> <NOT NULL>,
    <nome_coluna2> <tipo_coluna2> <NOT NULL>,
    ...
    <nome_colunam> <tipo_colunam> <NOT NULL>);
```

A restrição **not null** indica que o atributo deve ser obrigatoriamente preenchido; se não for especificado, então o default é que o atributo possa assumir o valor nulo.

Exemplo:

Ao se criar a tabela EMPREGADO do esquema EMPRESA, teria-se o seguinte comando:

```
create table EMPREGADO
    (nome char (30) NOT NULL, rg integer NOT NULL, cic integer,
    depto integer NOT NULL, rg_supervisor integer, salario, decimal
    (7,2)NOT NULL)
```

O comando **create table** inclui opções para especificar certas restrições de integridade, conforme se verá adiante.

A relação criada acima está inicialmente vazia. O comando *insert* poderá ser usado para carregar os dados para uma relação.

Para remover uma tabela de banco de dados SQL, usamos o comando **drop table**. Esse comando remove todas as informações sobre a relação retirada do banco de dados. A forma geral para o **drop table** é:

```
drop table <nome_tabela>;
```

Exemplo

Para se eliminar a tabela EMPREGADO do esquema EMPRESA, tem-se o seguinte comando:

```
drop table EMPREGADOS;
```

Observe que, neste caso, a chave da tabela EMPREGADOS (rg) é utilizada como chave estrangeira ou como chave primária, composta por diversas tabelas que devem ser devidamente corrigidas. Este processo não é assim tão simples, pois, como vemos neste caso, a exclusão da tabela EMPREGADO implica a alteração do projeto físico de diversas tabelas. Isso acaba implicando a construção de uma nova base de dados.

O comando **alter table** é usado para alterar a estrutura de uma relação existente. Ele também permite que o usuário faça a inclusão de novos atributos em uma tabela. A forma geral para o comando **alter table** é a seguinte:

```
alter table <tabela> <add,drop,modify> <coluna> <tipo_coluna>;
```

Em que *add* adiciona uma coluna, *drop* remove uma coluna, e *modify* modifica algo em uma tabela.

No caso do comando **alter table**, a restrição NOT NULL não é permitida, pois assim que se insere um novo atributo na tabela, o valor para ele em todas as tuplas da tabela receberá o valor NULL.

Não é permitido eliminar algum atributo de uma relação já definida. Assim, caso se deseje eliminar uma chave primária devidamente referenciada em outra tabela como chave estrangeira, em vez de obter a eliminação do campo, obterá apenas um erro.

3.1 Visões (Views)

Uma view em SQL é uma tabela que é derivada de outras tabelas ou de outras views. Uma view não necessariamente existe em forma física; é considerada uma tabela virtual (em contraste com as tabelas cujas tuplas são efetivamente armazenadas no banco de dados).

Uma view é definida usando o comando **create view**. Para definir uma visão, precisa-se dar a ela um nome e definir a consulta que a processa.

A forma geral é:

```
create view <nomevisao> as <expressão de consulta>
```

Em que <expressão de consulta> é qualquer consulta SQL válida. Como exemplo, considere a visão consistindo em nomes de agências e de clientes.

```
Create view TOTOS_CLIENTES as (select
agencia_nome,cliente_nome
from CLIENTE, CONTA, AGENCIA
where CLIENTE.cliente_cod = CONTA.cliente_cod and
CONTA.agencia_cod = AGENCIA.agencia_cod)

union

(select agencia_nome,cliente_nome
from CLIENTE, EMPRESTIMO, AGENCIA
where CLIENTE.cliente_cod = EMPRESTIMO.cliente_cod and
EMPRESTIMO.agencia_cod = AGENCIA.agencia_cod)
```

Nomes de visões podem aparecer em qualquer lugar onde nome de relação (tabela) possa aparecer. Usando a visão TOTOS_CLIENTES, pode-se achar todos os clientes da agência 'Princesa Isabel', escrevendo:

```
select cliente_nome
from TOTOS_CLIENTES
where agencia_nome = 'Princesa Isabel'
```

Uma modificação é permitida por intermédio de uma visão apenas se a visão em questão está definida em termos de uma relação do atual banco de dados relacional.

Exemplo

Selecionando tuplas de empréstimos.

```
Create view emprestimo_info as
(select agencia_cod, emprestimo_numero, cliente_cod
from EMPRESTIMO)
```

Uma vez que a SQL permite a um nome de visão aparecer em qualquer lugar em que um nome de relação aparece, pode-se escrever:

```
insert into emprestimo_info
values (1,40,7)
```

Essa inserção é representada por uma inserção na relação EMPRESTIMO, uma vez que é a relação a partir da qual a visão emprestimo_info foi construída. Deve-se, entretanto, ter algum valor para *quantia*. Esse valor é um valor nulo. Assim, o **insert** acima resulta na inserção da tupla: (1,40,7,null) na relação EMPRESTIMO.

Da mesma forma, poderia-se usar os comandos *update* e *delete*.

Para apagar uma visão, usa-se o comando

```
drop view <nomevisão>
```

Exemplo

Apagar a visão emprestimo_info.

```
drop view emprestimo_info
```

3.2. Restrições de integridade

As restrições de integridade fornecem meios para assegurar que mudanças feitas no banco de dados por usuários autorizados não resultem na inconsistência dos dados.

Há vários tipos de restrições de integridade que seriam cabíveis a bancos de dados. Entretanto, as regras de integridade são limitadas às que podem ser verificadas com um mínimo de tempo de processamento.

3.2.1. Restrições de domínios

Um valor de domínio pode ser associado a qualquer atributo. Restrições de domínio são as mais elementares formas de restrições de integridade. Elas são facilmente verificadas pelo sistema sempre que um novo item de dado é incorporado ao banco de dados.

O princípio que está por trás do domínio de atributos é similar ao dos tipos em linguagem de programação. De fato é possível criar domínios em SQL por meio do comando **create domain**.

A forma geral é:

```
CREATE DOMAIN nome_do_domínio tipo_original
[ [ NOT ] NULL ]
[ DEFAULT valor_default ]
[ CHECK ( condições ) ]
```

Exemplo

Cria o tipo_codigo como um número de 3 algarismos com valores permitidos de 100 a 800.

```
create domain tipo_codigo numeric(3)
not null
check ((tipo_codigo >= 100) and (tipo_codigo <= 800))
```

A cláusula **check** da SQL-92 permite modos poderosos de restrições de domínio. Ela pode ser usada também no momento de criação da tabela.

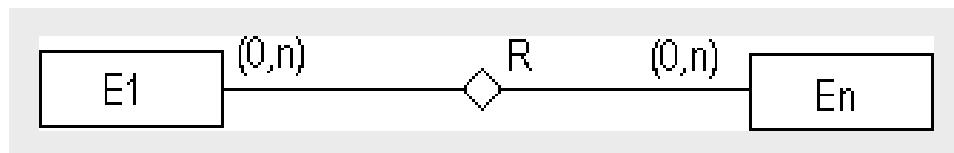
Exemplo

```
create table dept
(deptno integer(2) not null, dname char(12),
loc char(12),
check (dptono >= 100))
```

3.2.2. Restrições de integridade referencial

Muitas vezes deseja-se assegurar que um valor que aparece em uma relação para um dado conjunto de atributos apareça também para um certo conjunto de atributos em outra relação. Isso é chamado de *Integridade Referencial*.

Neste relacionamento, considerando **K1** como sendo *chave primária* de **E1**, e **Kn** como sendo *chave primária* de **En**, temos em **R** tuplas que serão identificadas inclusive por **K1** e **Kn** (na forma de chaves estrangeiras). Desta forma, não poderá existir um elemento **K1** em **R** que não faça parte de **E1**, tampouco um **Kn** em **R** que não faça parte de **En**.



As modificações no banco de dados podem causar violações de integridade referencial. Considerações devem ser feitas ao inserir, remover e atualizar tuplas.

Considerar: $\pi_a(R2) \subseteq \pi_k(R1)$

- **Inserir.** Se uma tupla t_2 é inserida em R_2 , o sistema precisa assegurar que existe uma tupla t_1 em R_1 tal que $t_1[k] = t_2[a]$. Isto é, $t_2[a] \in \pi_k(R1)$
- **Remover.** Se uma tupla t_2 é removida de R_2 , o sistema precisa computar o conjunto de tuplas em R_2 que referencia t_1 :

$$\sigma_a(R1) = t1[k](R2)$$

Se este conjunto não for vazio, o comando remover é rejeitado como um erro, ou as tuplas que se referem a t_1 precisam, elas mesmas, serem removidas. A última solução pode levar a uma remoção em cascata, uma vez que as tuplas precisam se referir a outras tuplas que se referem a t_1 e assim por diante.

- **Atualizar.** Precisa-se considerar dois casos para atualização: a atualização da relação referenciadora (filha - R_2) e a atualização da relação referenciada (pai - R_1).

Se a tupla t_2 é atualizada na relação R_2 e a atualização modifica valores para a chave estrangeira a , então é feito um teste similar para o caso de inserção. Para o caso de t_2 denotar o novo valor da tupla t_2' , o sistema precisa assegurar que:

$$t2' [a] \in \pi_k(R1)$$

Se a tupla t_1 é atualizada em R_1 e a atualização modifica valores para a chave primária (K), então é feito um teste similar ao caso remover. O sistema precisa computar

$$\sigma_a(R1) = t1[k](R2)$$

A SQL permite a especificação de chaves primárias, candidatas e estrangeiras como parte da instrução `create table`.

- A cláusula **primary key** da instrução `create table` inclui uma lista de atributos que compreende a *chave primária*;
- A cláusula **unique key** da instrução `create table` inclui uma lista de atributos que compreende a *chave candidata*;
- A cláusula **foreign key** da instrução `create table` inclui uma lista de atributos que compreende a *chave estrangeira* e o nome da relação referida pela chave estrangeira.

Exemplo:

Criar as relações cliente, agência e conta para o esquema do banco.

```
Create table CLIENTE
(cliente_cod integer not null,
cliente_nome char(30),
rua char(30),
cidade char(30),
primary key (cliente_cod))
```

```
Create table AGENCIA
(agencia_cod integer not null,
agencia_nome char(30),
agencia_cidade char(30),
fundos decimal (7,2),
primary key (agencia_cod),
check (fundos >= 0))
```

```
Create table CONTA
(agencia_cod int,
conta_numero char(10) not null,
cliente_cod int not null, saldo
decimal (7,2),
primary key (cliente_cod,conta_numero), foreign
key (cliente_cod) references clientes, foreign
key (agencia_cod) references agencias, check
(fundos >= 0))
```

Note que os atributos chave precisam ter a especificação **not null**.

Quando uma regra de integridade referencial é violada, o procedimento normal é rejeitar a ação que ocasionou essa violação. Entretanto, é possível criar ações para modificação das tabelas associadas em que tenha havido(ou em que haveria) a quebra de integridade referencial.

Isso é feito por meio das cláusulas **on delete cascade** e **on update cascade** associadas à cláusula **foreign key**.

Exemplo

```
Create table CONTA
(...

foreign key (agencia_cod) references agencia)
on delete cascade on update cascade,
...)
```

Nesse exemplo, se a remoção de uma tupla da tabela agência resultar na violação da regra de integridade, o problema é resolvido removendo as tuplas associadas a esta agência da tabela conta. De forma semelhante, se o código de uma agência for alterado, as contas desta agência serão também alteradas.

3.2.3. Asserções

Uma asserção (afirmação) é um predicado expressando uma condição que desejamos que o banco de dados sempre satisfaça. Quando uma assertiva é criada, o sistema testa sua validade. Se a afirmação é válida, então qualquer modificação posterior no banco de dados será permitida apenas quando a asserção não for violada.

Restrições de domínio e regras de integridade referencial são formas especiais de asserções.

O alto custo de testar e manter afirmações tem levado a maioria de desenvolvedores de sistemas a omitir o suporte para afirmações gerais. A proposta geral para a linguagem SQL inclui uma construção de proposta geral, chamada instrução **create assertion**, para a expressão de restrição de integridade. Uma afirmação pertencente a uma única relação toma a seguinte forma:

```
create assertion <nome da asserção> check <predicado>
```

Exemplo

Definir uma restrição de integridade que não permita saldos negativos.

```
create assert saldo_restricao
check (not exists (select * from CONTA where saldo < 0))
```

Exemplo

Só permitir a inserção de saldos maiores que quantias emprestadas para aquele cliente.

```
create assert saldo_restricao2
check (not exists
(select * from
conta where
saldo <
(select max(quantia)
from EMPRESTIMO
where EMPRESTIMO.cliente_cod = CONTA.cliente_cod)))
```

3.3. Exercícios.

Considerando o esquema:

```
Pessoas = (CodPessoa, NomePessoa, Cidade, Chefe)
Empresas = (CodEmpresa, NomeEmpresa, Cidade)
Trabalha = (CodPessoa, CodEmpresa, Salario)
```

1. Consulte todas as pessoas que trabalham em Vitória.

```
Select Pessoas.NomePessoa
from Pessoas
where Cidade = 'Vitória'
```

2. Consulte todas as pessoas que trabalham na mesma cidade onde moram.

```
Select Pessoas.NomePessoa
from Pessoas, Trabalha, Empresas
where Pessoas.CodPessoa = Trabalha.CodPessoa and
      Trabalha.CodEmpresa = Empresas.CodEmpresa and
      Pessoa.Cidade = Empresas.Cidade
```

3. Consulte todas as pessoas que moram na mesma cidade do chefe.

```
Select Pessoas.NomePessoa
from Pessoas, Pessoas_Chefs
where Pessoas.Chefe = Chefs.CodPessoa and
      Pessoa.Cidade = Chefs.Cidade
```

4. Consulte todas as empresas que funcionam em cidades que não moram pessoas cujo primeiro nome seja Maria (usar operações de conjunto).

```
Select Empresas.NomeEmpresa
from Empresas
Where Empresas.Cidade not in
      (select Cidade
       from Pessoas
       Where Pessoas.NomePessoa like 'Maria%')
```

5. Consulte todas as pessoas que não trabalham em Vitória e que ganham acima de 2000 em ordem decrescente da cidade onde moram.

```
Select Pessoas.NomePessoa
from Pessoas, Trabalha, Empresas
where Pessoas.CodPessoa = Trabalha.CodPessoa and
      Trabalha.CodEmpresa = Empresas.CodEmpresa and
      Empresas.Cidade < > 'Vitória' and Salario > 2000
order by pessoa.cidade desc
```

6. Consulte todas as pessoas que não trabalham na empresa que comece com 'inf_' em ordem alfabética.

```
Select Pessoas.NomePessoa
from Pessoas, Trabalha, Empresas
where Pessoas.CodPessoa = Trabalha.CodPessoa and
      Trabalha.CodEmpresa = Empresas.CodEmpresa and
      Empresas.nomeEmpresa not like 'inf%_%' escape '&'
order by nomeEmpresa
```

Considere o esquema seguinte para as questões que se seguem.

```
Fabricante =(codf, nomef)
Automovel =(coda, nomea, preço, codf)
Pessoa =(codp, nomep)
Venda =(codp, coda, valor, cor, data)
```

7. Quem comprou 'Ford'? (fazer duas consultas).

```
select Pessoa.NomeP
from Pessoa, venda
where Pessoa.codp = Venda.codp and
      venda.coda in
          (select coda
           from Automovel, Fabricante
           where Automovel.codf = Fabricante.codf
           and Fabricante.Nomef = 'Ford')
```

Outra resposta:

```
select Pessoa.NomeP
from Pessoa, venda, Automovel, Fabricante
where Pessoa.codp = Venda.codp and
      venda.coda = Automovel.coda and
      Automovel.codf = Fabricante.codf and
      Fabricante.Nomef = 'Ford'
```

8. Quem não comprou 'Ford'?

```
select Pessoa.NomeP
from Pessoa, venda
where Pessoa.codp = Venda.codp and
      venda.coda not in
          (select coda
           from Automovel, Fabricante
           where Automovel.codf = Fabricante.codf and
           Fabricante.Nomef = 'Ford')
```

9. Quem comprou carro com ágio?

```
select Pessoa.NomeP
from Pessoa, venda, Automovel
where Pessoa.codp = Venda.codp and
      venda.coda = Automovel.coda and
      venda.valor > Automovel.preço
```

10. Quem comprou 'Ford' e não comprou 'Volks'?

```

select Pessoa.NomeP
from Pessoa, venda
where Pessoa.codp = Venda.codp

and venda.coda in

(select coda
from Automovel, Fabricante
where Automovel.codf = Fabricante.codf
and Fabricante.Nomef = 'Ford')

and venda.coda not in
(select coda
from Automovel, Fabricante
where Automovel.codf = Fabricante.codf
and Fabricante.Nomef = 'Volks')
    
```

11. Quem comprou carro entre 01/01/97 e 01/01/98?

```

select Pessoa.NomeP
from Pessoa, venda
where Pessoa.codp = Venda.codp and
venda.data between '01/01/97' and '01/01/98'
    
```

Considere o esquema BANCO reproduzido abaixo.

```

Agencias = (agencia_cod, agencia_nome, agencia_cidade, ativos)
Clientes = (cliente_cod, cliente_nome, rua, cidade)
Depositos = (agencia_cod, conta_numero, cliente_cod, saldo)
Emprestimos = (agencia_cod, cliente_cod, emprestimo_numero, quantia)
    
```

12. Selecione todos os clientes que possuem contas em agência(s) que possui(em) o maior ativo.

```

Select cliente_nome
from clientes, depositos
where clientes.cliente_cod = depositos.cliente_cod and
depositos.agencia_cod in
(select agencia_cod
from agencias
where ativos >= all
(select ativos
from agencias))
    
```

13. Selecione o total de agências por cidade, classificado por cidade.

```

Select cidade, count(agencia_cod) as total_agencias
from agencias
group by cidade
order by cidade
    
```

14. Selecione, por agências, o(s) cliente(s) com o maior saldo.

```
Select agencia_nome, cliente_nome, saldo
from clientes, depositos, agencias
where clientes.cliente_cod = depositos.cliente_cod and
      depositos.agencia_cod = agencia.agencia_cod and
      <agencia_cod, saldo> in
          (Select agencia_cod, max(saldo) as maior_saldo
           from depositos
           group by agencia_cod)
```

15. Selecione o valor médio de empréstimos efetuados por cada agência em ordem crescente das cidades onde estas agências se situam.

```
Select agencia_nome, avg(quantia) as valor_medio
from emprestimos, agencias
where emprestimos.agencia_cod = agencias.agencia_cod
group by agencia_nome
order by cidade
```

16. Selecione a(s) agência(s) que possui(em) a maior média de quantia emprestada.

```
Select agencia_nome
from agencias, emprestimos
where emprestimos.agencia_cod = agencias.agencia_cod
group by agencia_nome
having avg(quantia) >= all
       (select avg(quantia)
        from emprestimos
        group by agencia_cod)
```

17. Selecione todas as agências situadas fora de Vitória que possuem a média de depósitos maior do que alguma agência localizada em Vitória.

```
Select agencia_nome
from agencias, depositos
where depositos.agencia_cod = agencias.agencia_cod and
      agencia.cidade <> 'Vitória'
group by agencia_nome
having avg(saldo) > some
       (select avg(saldo)
        from depositos, empresas
        where depositos.codEmpresa = empresas.codEmpresa and
              empresas.cidade='Vitória'
        group by agencia_cod)
```

18. Selecione o menor saldo de clientes, por agências.

```
Select agencia_nome, min(saldo) as menor_saldo
from depositos, agencias
where depositos.agencia_cod = agencias.agencia_cod
group by agencia_cod
```

19. Selecione o saldo de cada cliente, caso ele possua mais de uma conta no banco.

```
Select cliente_nome, sum(saldo)
from clientes, depositos
where clientes.cliente_cod = depositos.cliente_cod
group by cliente_nome
having count(agencia_cod) >1
```

Considere o esquema abaixo para as questões que se seguem

```
Pessoas = (CodPessoa, NomePessoa, Cidade, Chefe)
Empresas = (codEmpresa, NomeEmpresa, Cidade)
Trabalha = (CodPessoa, CodEmpresa, Salario)
```

20. Excluir todas as pessoas que possuem salário = 1000.

```
Delete pessoas where CodPessoa in
(select cod_pessoa from trabalha where salario = 1000)
```

21. Excluir todas as pessoas que trabalham em empresas situadas em Vitória.

```
Delete pessoas where CodPessoa in
(select cod_pessoa from trabalha, Empresas
where trabalha.CodEmpresa=Empresas.CodEmpresa and cidade='Vitoria')
```

22. Incluir na empresa de código '01' todos os moradores de Vitória com um salário = 100.

```
Insert into trabalha (codPessoa, codEmpresa, Salario)
Select codpessoa, '01', 100 from pessoas where cidade = 'Vitória'
```

23. Uma determinada empresa de código 'x' vai contratar todos os funcionários da empresa de código 'y' que ganham acima de 1000, dando um aumento de salário de 10%. Faça comando(s) SQL para que tal transação seja efetuada.

```
Insert into trabalha (codPessoa, codEmpresa, Salario)
Select codpessoa, 'x', salario*1.1
from trabalha
where codempresa='y' and salario >1000
```

```
Delete trabalha where codempresa = 'y' and salario >1000
```

24. Uma determinada empresa de código 'xyz' quer contratar todos que moram em Vitória e estão desempregados. Serão contratados com salário = 200. Faça comando(s) SQL para fazer tal transação.

```
Insert into trabalha (codPessoa, codEmpresa, Salario)
Select codp, 'xyx', 200
from pessoas
where cidade = 'Vitória' and codpessoa
not in (select codpessoa from trabalha)
```

25. Fazer um comando SQL para ajustar o salário de todos os funcionários da empresa 'Campana' em 5%.

```
Update trabalha
Set salario = salario * 1.05
Where codEmpresa in
  (select codEmpresa
   from empresas
   where nomeEmpresa = 'Campana')
```

26. Todas as pessoas que moram em Colatina e trabalham na empresa 'Campana' deverão mudar para Vitória, devido aos requisitos do diretor da empresa. Faça comando(s) SQL para fazer esta atualização da cidade.

```
Update Pessoas
Set cidade = 'Vitória'
Where codPessoa in
  (select codPessoa
   from Pessoas, Trabalha, empresas
   where Pessoas.codPessoa = Trabalha.CodPessoa and
         Pessoas.cidade = 'Colatina' and
         Trabalha.CodEmpresa = Empresas.codEmpresa and
         nomeEmpresa = 'Campana')
```

3.4. O Esquema Banco

AGÊNCIA

```
= (agencia_cod, agencia_nome, agencia_cidade, ativos)
```

CLIENTE

```
= (cliente_cod, cliente_nome, rua, cidade)
```

CONTA

```
= (agencia_cod, conta_numero, cliente_cod, saldo)
```

EMPRÉSTIMO

```
= (agencia_cod, cliente_cod, emprestimo_numero, quantia)
```

De uma forma visual:

AGÊNCIA	CONTA	EMPRÉSTIMO	CLIENTE
agencia_cidade	cliente_cod	cliente_cod	cliente_cod
agencia_nome	agencia_cod	emprestimo_numero	cliente_nome
agencia_cod	conta_numero	quantia	rua
ativos	saldo	agencia_cod	cidade

3.5. O Esquema Empresa

Tabela EMPREGADO

Nome	RG	CPF	Deptº	RG Supervisor	Salário
João Luiz	10101010	11111111	1	NULO	R\$3000,00
Fernando	20202020	22222222	2	10101010	R\$2500,00
Ricardo	30303030	33333333	2	10101010	R\$2300,00
Jorge	40404040	44444444	2	20202020	R\$4200,00
Renato	50505050	55555555	3	20202020	R\$1300,00

Tabela DEPARTAMENTO

Nome	Número	RG Gerente
Contabilidade	1	10101010
Engenharia Civil	2	30303030
Engenharia Mecânica	3	20202020

Tabela PROJETO

Nome	Número	Localização
Financeiro 1	5	São Paulo
Motor 3	10	Rio Claro
Prédio Central	20	Campinas

Tabela DEPENDENTES

RG Responsável	Nome Dependente	Nascimento	Relação	Sexo
10101010	Jorge	27/12/86	Filho	Masculino
10101010	Luiz	18/11/79	Filho	Masculino
20202020	Fernanda	14/02/69	Cônjuge	Feminino
20202020	Ângelo	10/02/95	Filho	Masculino
30303030	Andreia	01/05/90	Filho	Feminino

Tabela DEPARTAMENTO_PROJETO

Número Depto	Número Projeto
2	5
3	10
2	20

Tabela EMPREGADO_PROJETO

RG Empregado	Número Projeto	Horas
20202020	5	10
20202020	10	25
30303030	5	35
40404040	20	50
50505050	20	35

4. Modelagem de Dados

A **modelagem de dados** é uma etapa importante e essencial em qualquer projeto de desenvolvimento ou de manutenção de software. Assim, estar atualizado neste assunto é importante para qualquer profissional da área.

4.1. O que é modelagem de dados?

Modelagem de dados é o ato de explorar estruturas orientadas a dados. Como outros artefatos de modelagem, modelos de dados podem ser usados para uma variedade de propósitos, desde modelos conceituais de alto nível até modelos físicos de dados. Do ponto de vista de um desenvolvedor atuando no paradigma orientado a objetos, modelagem de dados é conceitualmente similar à modelagem de classes. Com a modelagem de dados, identifica-se tipos de entidades (da mesma forma que na modelagem de classes é possível identificar classes). Atributos de dados são associados a tipos de entidades, exatamente como associados atributos e operações às classes. Existem associações entre entidades similares às associações entre classes – relacionamento, herança, composição e agregação são todos os conceitos aplicáveis em modelagem de dados.

Modelagem de dados tradicional é diferente da modelagem de classes porque o foco daquela é totalmente nos dados – os modelos de classes permitem explorar os aspectos comportamentais e de dados em um domínio de aplicação, já com o modelo de dados, pode-se apenas explorar o aspecto dado. Por causa desse foco, projetistas de dados tendem a ser melhores em identificar os dados “corretos” em uma aplicação do que modeladores de objetos. No entanto, algumas pessoas modelam métodos de banco de dados (*stored procedures, stored functions e triggers*) quando estão realizando a modelagem física dos dados.

4.2. Como modelos de dados são usados na prática?

Embora as questões de metodologias sejam abordadas depois, precisa-se discutir como os modelos de dados podem ser usados na prática para melhor entender-lhes. Provavelmente, vai ser possível se deparar com três estilos básicos de modelos de dados:

- **Modelos de dados conceituais:** esses modelos, algumas vezes chamados modelos de domínio, são tipicamente usados para explorar conceitos do domínio com os envolvidos no projeto. Em equipes ágeis, modelos conceituais de alto nível são normalmente criados como parte do esforço inicial do entendimento dos requisitos do sistema, pois eles são usados para explorar as estruturas e conceitos de negócio estáticos de alto nível. Em equipes tradicionais (não ágeis), modelos de dados conceituais são normalmente criados como precursores aos modelos lógicos de dados (MLD) ou suas alternativas.
- **Modelos Lógico de Dados (MLDs):** MLDs são usados para explorar os conceitos do domínio e seus relacionados. Isso pode ser feito para o escopo de um simples projeto ou para uma empresa inteira. MLDs descrevem os tipos de entidades lógicas (tipicamente referenciadas simplesmente como tipos de entidades), os atributos de dados que descrevem essas entidades, bem como os relacionamentos entre elas. MLDs são raramente usados em projetos ágeis, apesar de normalmente estarem presentes em projetos tradicionais (onde eles raramente adicionam muito valor à prática).
- **Modelos Físicos de Dados (MFDs):** MFDs são usados para projetar o esquema interno de um banco de dados, descrevendo as tabelas de dados, as colunas de dados das tabelas e o relacionamento entre as tabelas. MFDs normalmente são bastante úteis em projetos ágeis e tradicionais.

Embora MLDs e MFDs pareçam similares, e eles de fato são, o nível de detalhes que eles modelam pode ser significativamente diferente. Isso porque o objetivo de cada diagrama é diferente – podemos usar um MLD para explorar conceitos do domínio com os envolvidos no projeto e MFD para definir o projeto do banco de dados.

A seguir, a **Figura 1** apresenta um simples MLD e a **Figura 2** um simples MFD, ambos modelando o conceito de clientes e endereços, assim como o relacionamento entre eles. Ambos os diagramas seguem a notação de Barker, que será descrita mais adiante. Note como o MFD mostra mais detalhes, incluindo uma tabela associativa necessária para implementar a associação, assim como as chaves necessárias para manter os relacionamentos. Mais detalhes sobre esses conceitos serão descritos à frente..

MFDs devem também refletir os padrões de nomenclatura de banco de dados da organização. Nesse caso, uma abreviação do nome da entidade é colocada para cada nome de coluna e uma abreviação para “número” foi consistentemente introduzida. Um MFD deve também indicar os tipos de dados das colunas, tais como integer e char(5). Apesar de a **Figura 2** não mostrá-las, tabelas de referência como para o endereço é usado, assim como para estados e países estão implícitos pelos atributos END_USADO_CODIGO, END_ESTADO_CODIGO, END_PAIS_CODIGO.

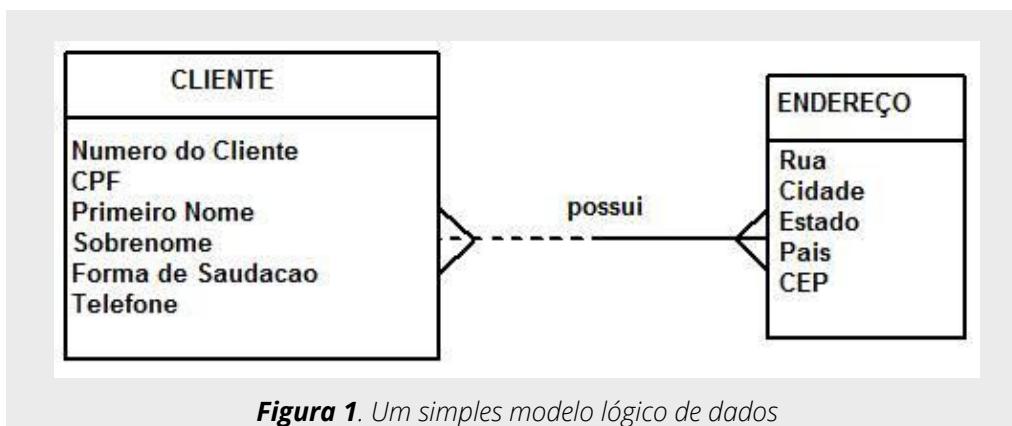


Figura 1. Um simples modelo lógico de dados

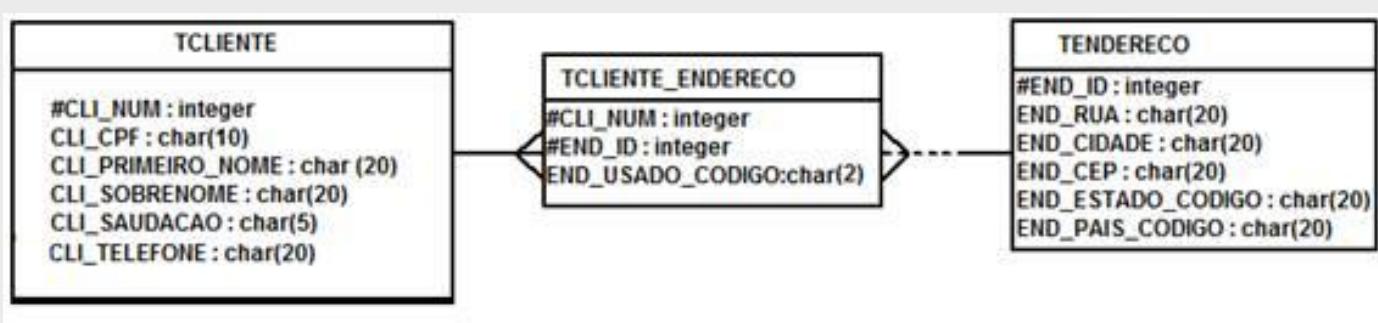


Figura 2. Um simples modelo físico de dados

Modelos de dados podem ser usados efetivamente tanto no nível da empresa como de projetos. Os arquitetos da empresa normalmente criariam um ou mais MLDs de alto nível que poderiam descrever as estruturas de dados que apoiam toda a empresa – normalmente chamados de modelos de dados da empresa ou modelos de informação da empresa. Um modelo de dados da empresa é uma das várias visões que os arquitetos da empresa podem escolher para manter e apoiar – outras visões podem explorar a infraestrutura de rede/hardware, a estrutura da organização, infraestrutura de softwares, o processo de negócios, dentre outros. Esses modelos provêm informações que uma equipe de projeto pode usar como conjunto de restrições e como descrição da estrutura do sistema.

Equipes de projeto tipicamente criariam MLDs como um dos principais artefatos de análise quando seu ambiente de implementação for predominantemente procedural por natureza (um exemplo seria quando usam COBOL estruturado como linguagem de implementação). MLDs são também boas escolhas quando um projeto é orientado a dados, como um *data warehouse* ou sistema de relatório. No entanto, MLDs são normalmente escolhas ruins quando uma equipe de projeto está usando tecnologias orientadas a objeto ou baseadas em componentes, porque é fato que os desenvolvedores trabalhariam melhor com diagramas UML ou quando o projeto não é orientado a dados. Como uma dica de modelagem, aplique os artefatos corretos para aquele trabalho a ser desenvolvido.

Quando um banco de dados relacional é usado para armazenar dados, equipes de projeto são aconselhadas a criar um MFD para modelar um esquema interno. MFD normalmente é apenas um dos artefatos de projeto críticos para projetos de desenvolvimento de aplicações de negócios.

4.3. O que dizer sobre modelos conceituais?

Muitos profissionais de dados preferem criar um ORM (Object-Role Model), como o apresentado no exemplo da **Figura 3**, em vez de um MLD para um modelo conceitual. A vantagem é que a notação é muito simples, algo que os envolvidos no projeto podem rapidamente interpretar, apesar da desvantagem que seria o fato de os modelos se tornarem grandes rapidamente. ORMs permitem primeiramente explorar os exemplos de dados reais em vez de simplesmente saltar para uma abstração potencialmente incorreta – por exemplo, a **Figura 3** examina o relacionamento entre clientes e um endereço em detalhe.

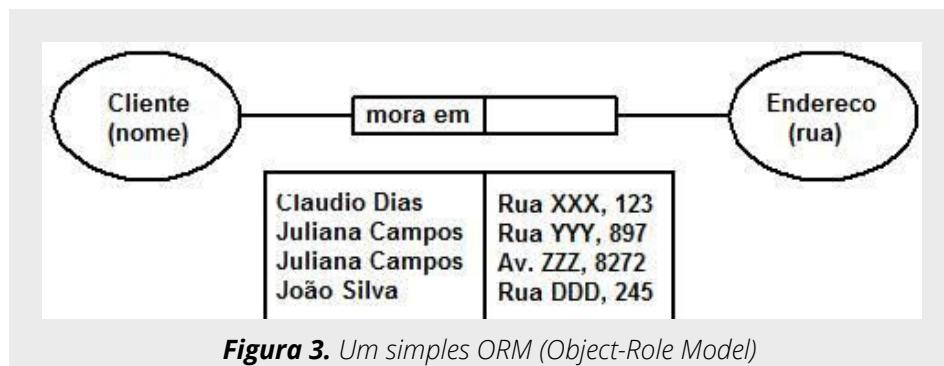


Figura 3. Um simples ORM (Object-Role Model)

Normalmente os ORMs são usados para explorar o domínio da aplicação com os envolvidos no projeto, mas depois ele é substituído por um artefato mais tradicional, como um MLD, um [diagrama de classes](#) ou até um MFD.

4.4. Notações comuns de modelagem de dados

A **Figura 4** apresenta um resumo da sintaxe das quatro notações mais comuns para modelagem de dados: Engenharia da Informação (EI), Notação de Barker, IDEF1X e UML (*Unified Modeling Language*). Este diagrama não tem a pretensão de ser altamente comprehensivo, mas sim de prover uma visão geral básica sobre as notações. Além disso, para não se estender muito no texto, escolheu-se não descrever a abordagem altamente detalhada para nomenclatura de relacionamento como sugerido por Barker.

Notação	Engenharia da Informação	Notação de Barker	IDEF1X	UML
<u>Multiplicidade:</u>				
Zero ou Um				
Somente um				
Zero ou mais				
Um ou mais				
Intervalo específico	NA	NA	NA	
<u>Atributos:</u>				
Nomes	NA	Attribute Name: Type	attribute-name: Type	attributeName: Type
Chave Primária	NA	# Attribute Name	attribute-name	attributeName <>PK><order=>
Chave Estrangeira	NA	NA	attribute-name (FK)	attributeName <>FK><to=tablename>
<u>Associações:</u>				
Rótulos				
Papéis de Entidade	NA	NA	NA	
Subtipo				
Agregação				
Composição			NA	
ou Restrição		NA		

Figura 4. Comparando a sintaxe das notações comuns para modelagem de dados

4.5. Como modelar dados

É crucial para um desenvolvedor de aplicação ter uma noção dos fundamentos de modelagem de dados não apenas para ler os modelos de dados, mas também para trabalhar efetivamente com os DBAs responsáveis pelos aspectos relacionados aos dados do projeto. O objetivo ao ler esta seção não é aprender como se tornar um modelador de dados, mas sim obter uma apreciação a respeito do que é envolvido nesta tarefa.

As seguintes tarefas são realizadas de forma iterativa:

- Identificar os tipos de entidade;
- Identificar atributos;
- Aplicar convenção de nomes;
- Identificar relacionamentos;
- Associar chaves;
- Normalizar para reduzir a redundância dos dados;
- Diversificar para melhorar o desempenho.

4.6. Identificar os tipos de entidade

Um tipo de entidade, ou simplesmente “entidade”, é conceitualmente similar ao conceito de orientação a objeto de uma classe – que representa uma coleção de objetos similares, podendo representar uma coleção de pessoas, lugares, coisas, eventos ou conceitos. Exemplos de entidades em um sistema de vendas incluiria: Cliente, Endereço, Venda, Item e Taxa. Se a necessidade fosse modelar classes, esperaria-se descobrir classes exatamente com esses nomes. No entanto, a diferença entre uma classe e um tipo de entidade é que classes possuem dados e comportamentos, enquanto tipos de entidade possuem apenas dados.

Idealmente, uma entidade deveria ser normal, descrevendo de forma coesa uma informação do mundo real. Uma entidade normalmente descreve um conceito, tal como uma classe coesa modela um conceito. Por exemplo, cliente e venda são claramente dois conceitos diferentes, portanto, faz sentido modelá-los como entidades diferentes.

4.7. Identificar atributos

Cada tipo de entidade terá um ou mais atributos de dados. Por exemplo, na **Figura 1** podemos ver que a entidade Cliente possui atributos como Primeiro Nome e Sobrenome e na **Figura 2** que a tabela CLIENTE possui colunas de dados correspondentes CLI_PRIMEIRO_NOME e CLI_SOBRENOME (uma coluna é a implementação de um atributo de dados em um banco de dados relacional).

Atributos devem ser coesos do ponto de vista do domínio da aplicação. Na **Figura 1**, decidiu-se que se iria modelar o fato de pessoas possuírem primeiro nome e sobrenome em vez de apenas um nome (ex: "Cláudio" e "Dias" VS. "Cláudio Dias"). Usar o nível de detalhe correto pode ter um impacto significativo no esforço de desenvolvimento e manutenção. Refatorar uma simples coluna de dados em várias colunas pode ser difícil, o que pode resultar em construir o sistema com elementos desnecessários e, portanto, provocar um custo de desenvolvimento e de manutenção maior do que o realmente necessário.

4.8. Aplicar convenções de nome

Sua organização deve dispor de normas e diretrizes aplicáveis à modelagem de dados, algo que você deve ser capaz de obter dos administradores da empresa (se não existir, você deve fazer algum *lobby* para incluí-la). Essas diretrizes devem incluir as convenções de nomenclatura para a modelagem lógica e física. As convenções de nomenclatura lógica devem ser focadas na capacidade de leitura de humanos, enquanto as convenções de nomenclatura física refletirão considerações técnicas. Você pode ver claramente que diferentes convenções de nomenclatura foram aplicadas nas **Figuras 1 e 2**.

A ideia básica é que analistas sigam um conjunto comum de padrões de modelagem em um projeto de *software*. Tal como é importante seguir convenções comuns de codificação, um código limpo que segue as diretrizes escolhidas é mais fácil de ser compreendido. Isso funciona da mesma forma para as convenções de modelagem de dados.

4.9. Identificar relacionamentos

No mundo real, entidades possuem relacionamentos entre elas. Por exemplo, clientes FAZEM compras, clientes MORAM EM endereços e itens de venda SÃO PARTE DAS vendas. Todos esses termos destacados definem relacionamentos entre entidades. Os relacionamentos entre entidades são conceitualmente idênticos aos relacionamentos (associações) entre objetos.

A **Figura 5**, a seguir, descreve um MLD parcial para um sistema de compra on-line. O primeiro elemento a se notar são os vários estilos aplicados aos nomes dos relacionamentos e papéis – diferentes relacionamentos requerem diferentes abordagens. Por exemplo, o relacionamento entre Cliente e Venda possui dois nomes (“compra” e “comprado por”, mesmo o relacionamento entre essas entidades sendo apenas um). Neste exemplo, tendo um segundo nome no relacionamento, a ideia seria especificar como ler o relacionamento em cada direção. O ideal seria colocar apenas um nome por relacionamento.

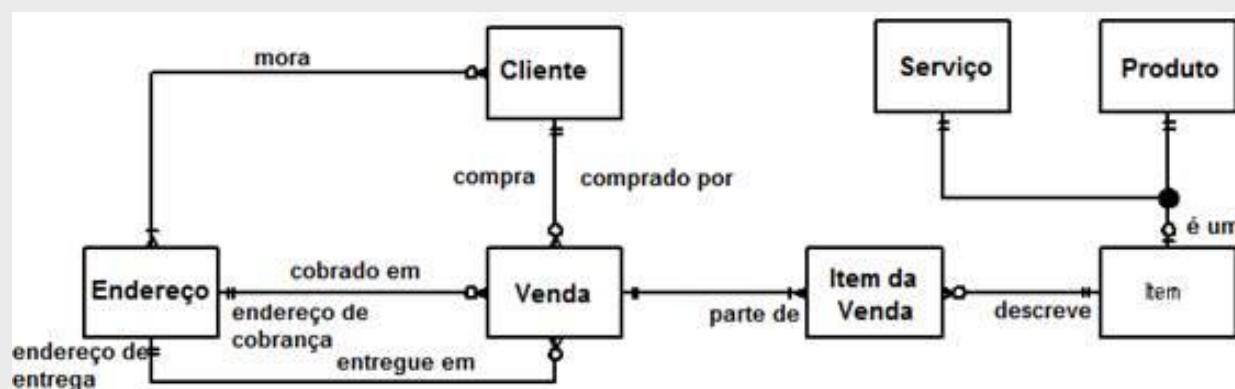


Figura 5. Um modelo lógico de dados (notações Engenharia da Informação)

Precisa-se também identificar cardinalidade e opcionalmente de um relacionamento (a UML combina os conceitos de “cardinalidade” e “opcionalmente” de um relacionamento em um conceito único de multiplicidade). Cardinalidade representa o conceito de “quantos”, enquanto Opcionalmente representa o conceito de “se é obrigatória a existência da entidade”. Por exemplo, não é suficiente saber que clientes fazem vendas. Quantas vendas um cliente pode realizar? Nenhuma, uma ou várias? Além disso, os relacionamentos existem nos dois sentidos: não apenas clientes fazem vendas, mas vendas são realizadas por clientes. Isso nos leva a questões como: quantos clientes podem ser envolvidos em uma dada venda? É possível ter uma venda com nenhum cliente envolvido?

A **Figura 5** mostra que clientes fazem nenhuma ou mais vendas e que qualquer venda é realizada por um e somente um cliente. Ela também mostra que um cliente possui um ou mais endereços e que qualquer endereço possui zero ou mais clientes associados a ele.

Apesar de a UML distinguir entre diferentes tipos de relacionamentos – associações, hierarquia, agregação, composição e dependência – modeladores de dados normalmente não estão por dentro dessa questão. Subtipo, uma aplicação de hierarquia, é normalmente encontrada em modelos de dados. Agregação e composição são muito menos comuns, assim como dependências, que são tipicamente uma construção de software e, portanto, não aparecem no modelo de dados, a menos que tenhamos um modelo físico de dados bastante detalhado que mostre como *views*, *triggers* ou *stored procedures* dependem de outros aspectos do esquema do banco de dados.

4.10. Associar chaves

Existem duas estratégias fundamentais para associar chaves às tabelas. Primeiro, podemos associar uma chave natural, que é um ou mais atributos de dados existentes, únicos para o conceito do negócio. Imagine uma tabela Cliente, por exemplo. Ela possui de imediato duas chaves candidatas, as colunas NumeroCliente e CPF. A segunda forma é introduzir uma nova coluna, chamada chave substituta, que é uma chave que não possui qualquer significado para o negócio. Um exemplo disso seria uma coluna idEndereco de uma tabela Endereco. Endereços não possuem uma chave natural "trivial" porque seria necessário usar todas as colunas da tabela Endereco para formar uma chave. Assim, introduzir uma chave substituta é uma opção muito melhor neste caso.

O debate entre "natural vs. substituta" é uma das grandes questões "religiosas" na comunidade de banco de dados. O fato é que não existe estratégia perfeita, e com o tempo percebemos que na prática, algumas vezes, faz sentido usar chaves naturais, e em outras situações é mais adequado o uso de chaves substitutas.

4.11. Normalizar para reduzir redundância de dados

Normalização de dados é um processo no qual atributos de dados são organizados para aumentar a coesão dos tipos de entidade. Em outras palavras, o objetivo da normalização é reduzir e até eliminar redundância de dados, uma questão importante para desenvolvedores, pois é incrivelmente difícil armazenar objetos em um banco de dados relacional que mantém a informação em vários lugares. A **Tabela 8**, a seguir, resume as três principais regras de normalização, descrevendo como aumentar os níveis de normalização em tipos de entidade.

Com respeito à terminologia, um esquema de dados é considerado estar em um nível de normalização do seu tipo de entidade menos normalizado. Por exemplo, se todos os tipos de entidade estão na segunda forma normal (2NF) ou maior, então dizemos que o esquema de dados está na 2NF.

Nível	Regra
Primeira Forma Normal (1NF)	Uma entidade está na 1NF quando ela não contém grupos de dados repetidos.
Segunda Forma Normal (2NF)	Uma entidade está na 2NF quando ela está na 1NF e quando todos seus atributos que não são chaves primárias são completamente dependentes de sua chave primária.
Terceira Forma Normal (3NF)	Uma entidade está na 3NF quando ela está na 2NF e quando todos seus atributos são diretamente dependentes da chave primária.

Tabela 08. Regras de normalização de dados

A **Figura 6**, exposta mais a frente, descreve um esquema de banco de dados na ONF, enquanto a **Figura 7** descreve um esquema normalizado na 3NF.

Por que normalização de dados? A vantagem de ter um esquema de dados altamente normalizado é que a informação é armazenada em um lugar apenas, reduzindo a possibilidade de dados inconsistentes. Além disso, esquemas de dados altamente normalizados em geral são conceitualmente mais próximos dos esquemas orientados a objeto, pois os objetivos da orientação a objetos de promover alta coesão e pouco acoplamento entre as classes resulta em soluções similares (ao menos do ponto de vista de dados). Isso geralmente torna mais simples mapear os objetos para o esquema de dados.

Infelizmente, a normalização normalmente traz um custo para o desempenho. Com o esquema de dados da **Figura 6**, todos os dados para uma venda estão armazenados em uma linha (assumindo que vendas poderão ter até dois itens), simplificando o acesso. Com o esquema de dados da **Figura 6**, pode-se rapidamente determinar a quantidade total de uma venda lendo uma única linha da tabela. Para fazer o mesmo com o esquema de dados da **Figura 7**, precisa-se ler dados a partir de uma linha na tabela Venda, dados a partir de linhas na tabela ItemVenda para aquela venda e dados a partir das linhas correspondentes na tabela Item. Para esta consulta, o esquema de dados da **Figura 6** provavelmente obteve o melhor resultado.

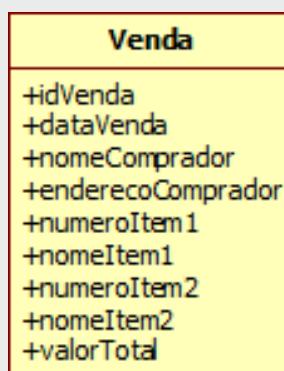


Figura 6. Um esquema de dados inicial para Venda (notação UML)

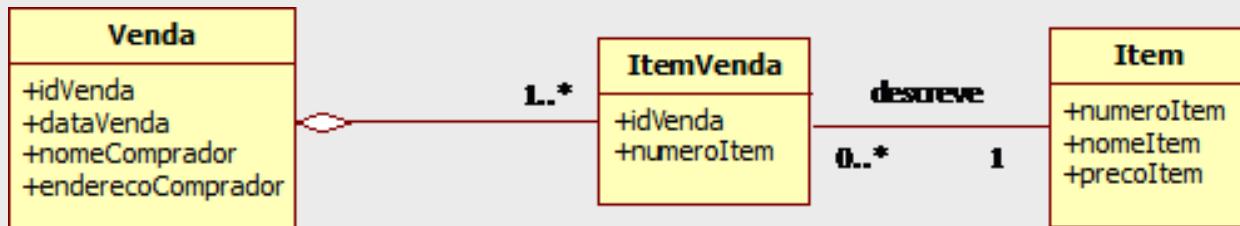


Figura 7. Um esquema normalizado em 3NF (notação UML)

Esquemas de dados normalizados, quando colocados em produção, normalmente sofrem problemas de desempenho. Isso faz sentido, já que as regras de normalização focam em reduzir redundância de dados, não em melhorar o desempenho do acesso aos dados. Uma parte importante da modelagem de dados é diversificar porções do esquema de dados para melhorar o tempo de acesso a esses dados.

Observe que se o projeto inicial e normalizado dos dados atinge o desempenho necessário para a aplicação, nada precisa ser feito. A Diversificação deve ser aplicada apenas quando os testes de desempenho mostram que temos um problema com os objetos, revelando que precisamos melhorar o tempo de acesso aos dados.

4.12. Modelagem de dados evolucionária/ágil

Modelagem de dados evolucionária é a modelagem de dados realizada de forma incremental. Modelagem de dados ágil é a modelagem de dados evolucionária feita de forma colaborativa.

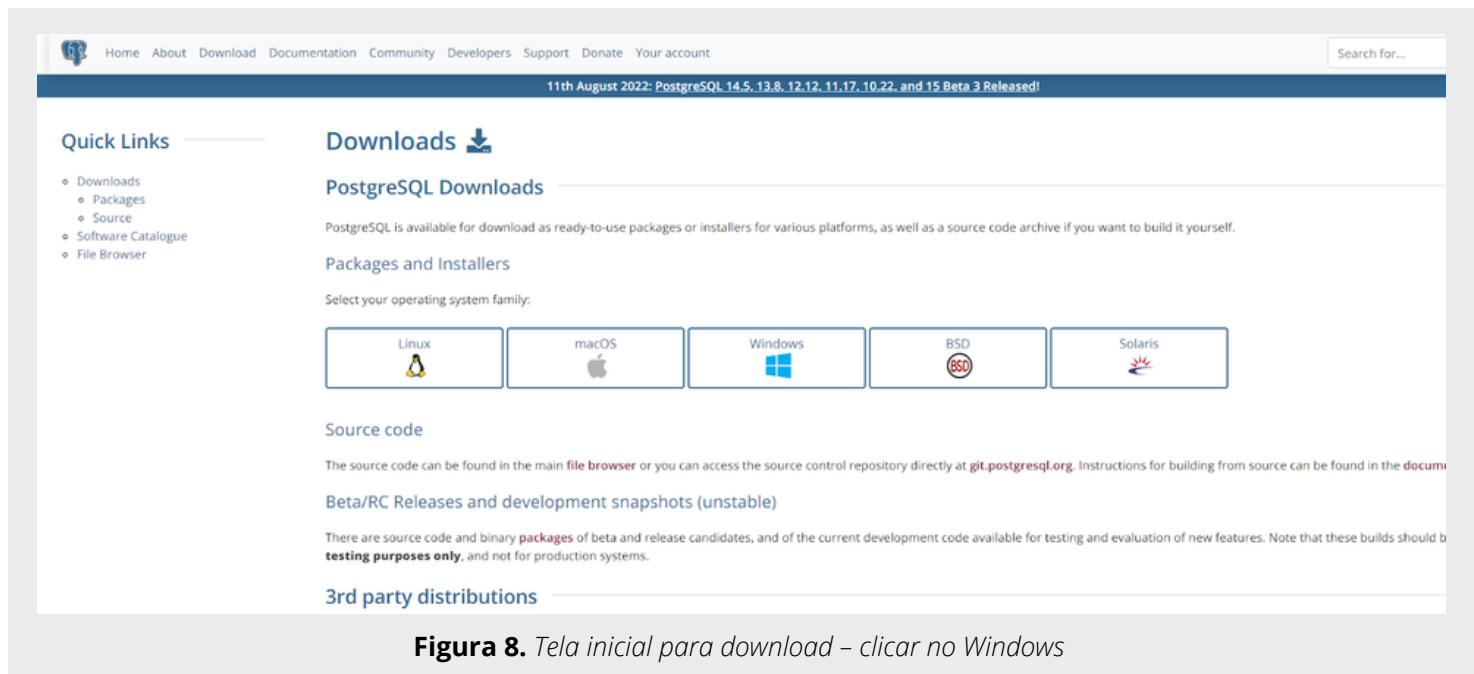
Apesar de nem todos pensarem assim, a modelagem de dados pode ser uma das mais desafiantes tarefas em que um DBA pode estar envolvido tendo em vista um projeto de desenvolvimento ágil.

Como você melhora suas habilidades de modelagem de dados? Prática, prática, prática.

Se você tiver a chance, você deve trabalhar próximo a DBAs, se oferecer para trabalhar na modelagem de dados e perguntar sobre o andamento do trabalho. Isso fará com que o seu conhecimento prático na área possibilite a sua evolução profissional, tanto na modelagem lógica como na modelagem física. Além disso, você pode ter a oportunidade de trabalhar com o arquiteto da empresa. Estes profissionais também possuem bastante conhecimento sobre o domínio das aplicações e como abstraí-los para a modelagem dos dados. Além de tudo isso, é preciso estar sempre atualizado lendo informações sobre a área.

5. Instalando e Configurando o PostgreSQL

<https://www.postgresql.org/download/>



Quick Links

- Downloads
- Packages
- Source
- Software Catalogue
- File Browser

Downloads

PostgreSQL Downloads

PostgreSQL is available for download as ready-to-use packages or installers for various platforms, as well as a source code archive if you want to build it yourself.

Packages and Installers

Select your operating system family:

Source code

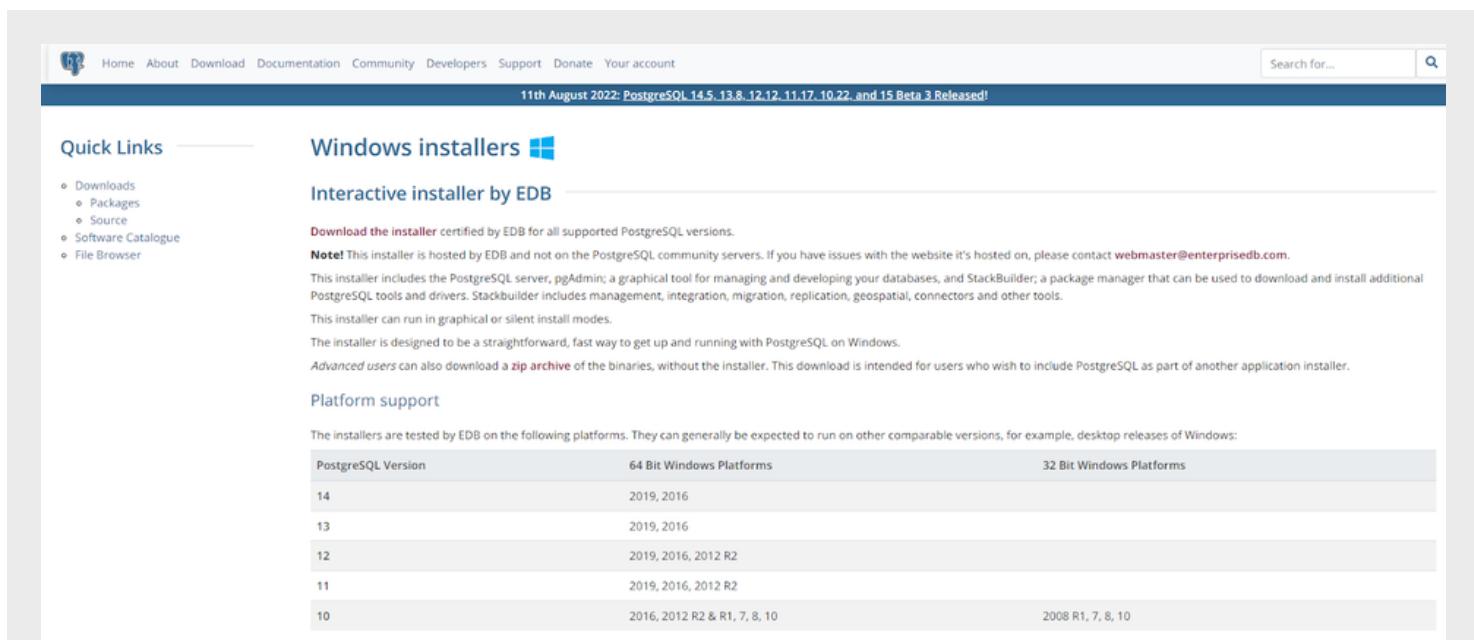
The source code can be found in the main [file browser](#) or you can access the source control repository directly at git.postgresql.org. Instructions for building from source can be found in the [documentation](#).

Beta/RC Releases and development snapshots (unstable)

There are source code and binary [packages](#) of beta and release candidates, and of the current development code available for testing and evaluation of new features. Note that these builds should be used for [testing purposes only](#), and not for production systems.

3rd party distributions

Figura 8. Tela inicial para download – clicar no Windows



Quick Links

- Downloads
- Packages
- Source
- Software Catalogue
- File Browser

Windows installers

Interactive installer by EDB

Download the [installer](#) certified by EDB for all supported PostgreSQL versions.

Note! This installer is hosted by EDB and not on the PostgreSQL community servers. If you have issues with the website it's hosted on, please contact webmaster@enterprisedb.com.

This installer includes the PostgreSQL server, pgAdmin; a graphical tool for managing and developing your databases, and StackBuilder; a package manager that can be used to download and install additional PostgreSQL tools and drivers. StackBuilder includes management, integration, migration, replication, geospatial, connectors and other tools.

This installer can run in graphical or silent install modes.

The installer is designed to be a straightforward, fast way to get up and running with PostgreSQL on Windows.

Advanced users can also download a [zip archive](#) of the binaries, without the installer. This download is intended for users who wish to include PostgreSQL as part of another application installer.

Platform support

The installers are tested by EDB on the following platforms. They can generally be expected to run on other comparable versions, for example, desktop releases of Windows:

PostgreSQL Version	64 Bit Windows Platforms	32 Bit Windows Platforms
14	2019, 2016	
13	2019, 2016	
12	2019, 2016, 2012 R2	
11	2019, 2016, 2012 R2	
10	2016, 2012 R2 & R1, 7, 8, 10	2008 R1, 7, 8, 10

Figura 9. Clicar no link "Download the installer"

Download PostgreSQL

Open source PostgreSQL packages and installers from EDB

PostgreSQL Version	Linux x86-64	Linux x86-32	Mac OS X	Windows x86-64	Windows x86-32
14.5	postgresql.org	postgresql.org			Not supported
13.8	postgresql.org	postgresql.org			Not supported
12.12	postgresql.org	postgresql.org			Not supported
11.17	postgresql.org	postgresql.org			Not supported
10.22					
9.6.24*					
9.5.25*					

Figura 10. Clicar na opção 14.5 Windows x86-64

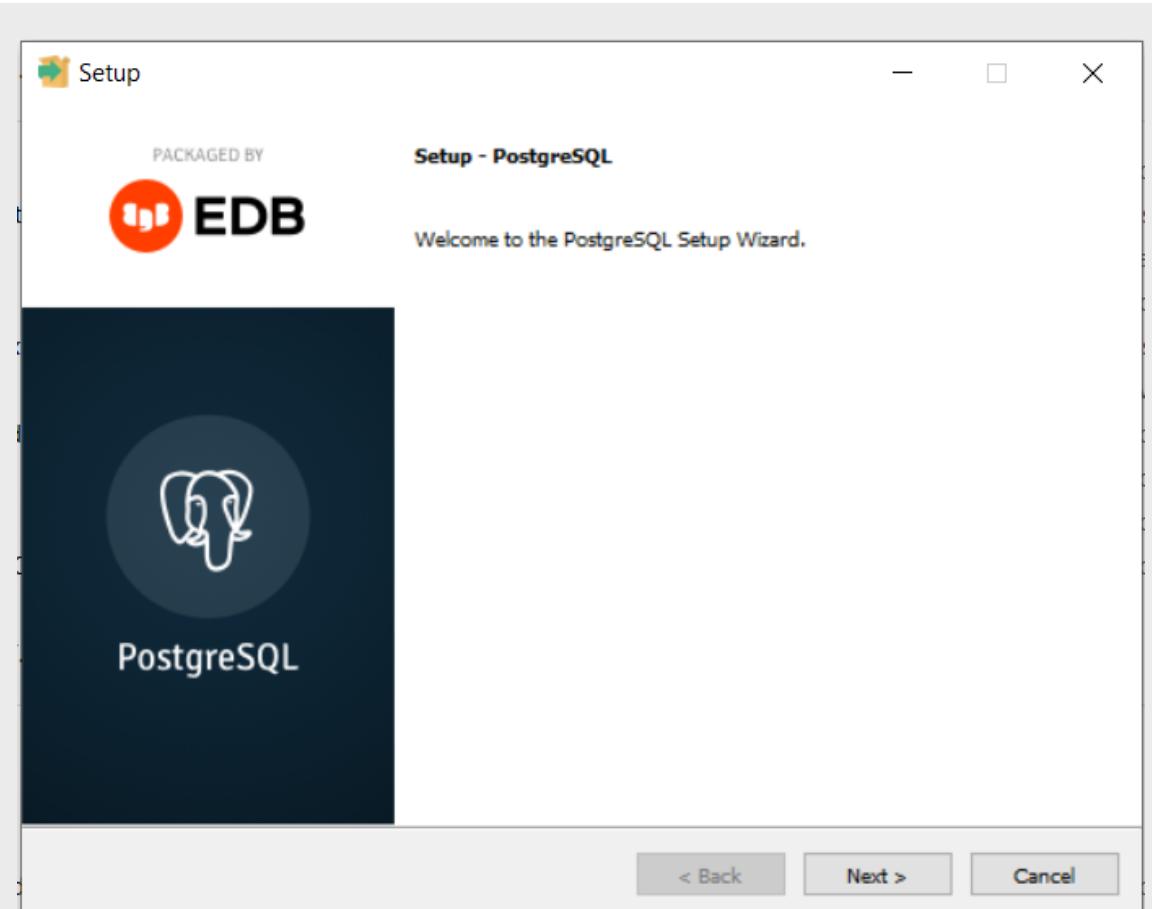


Figura 11. Iniciando a instalação

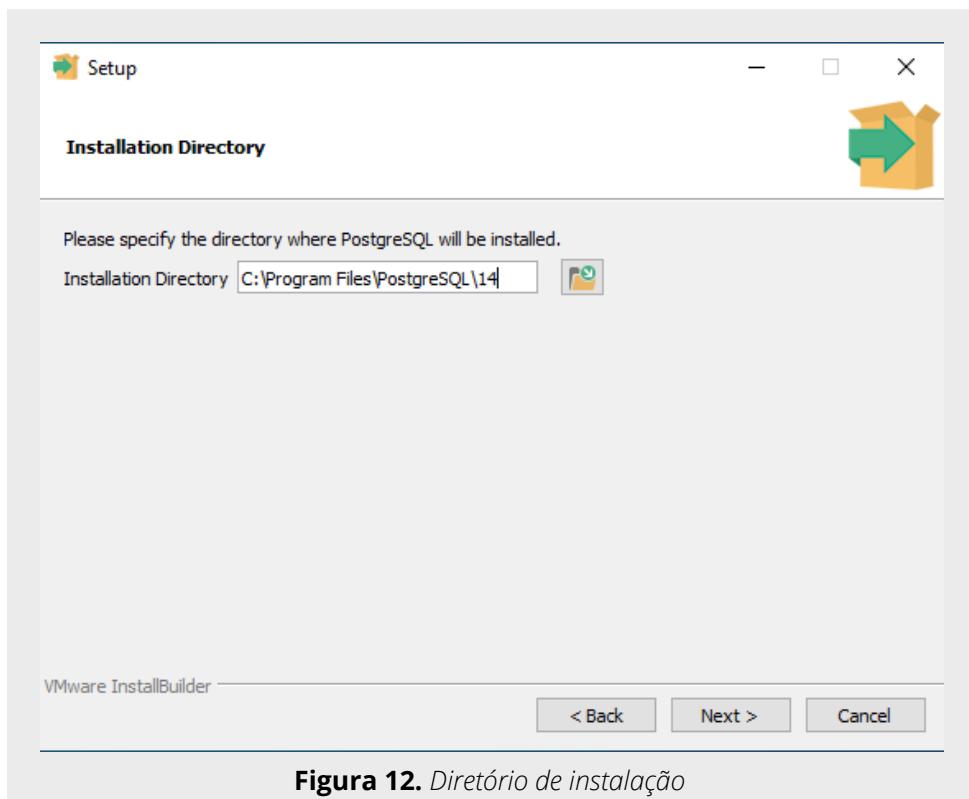


Figura 12. Diretório de instalação

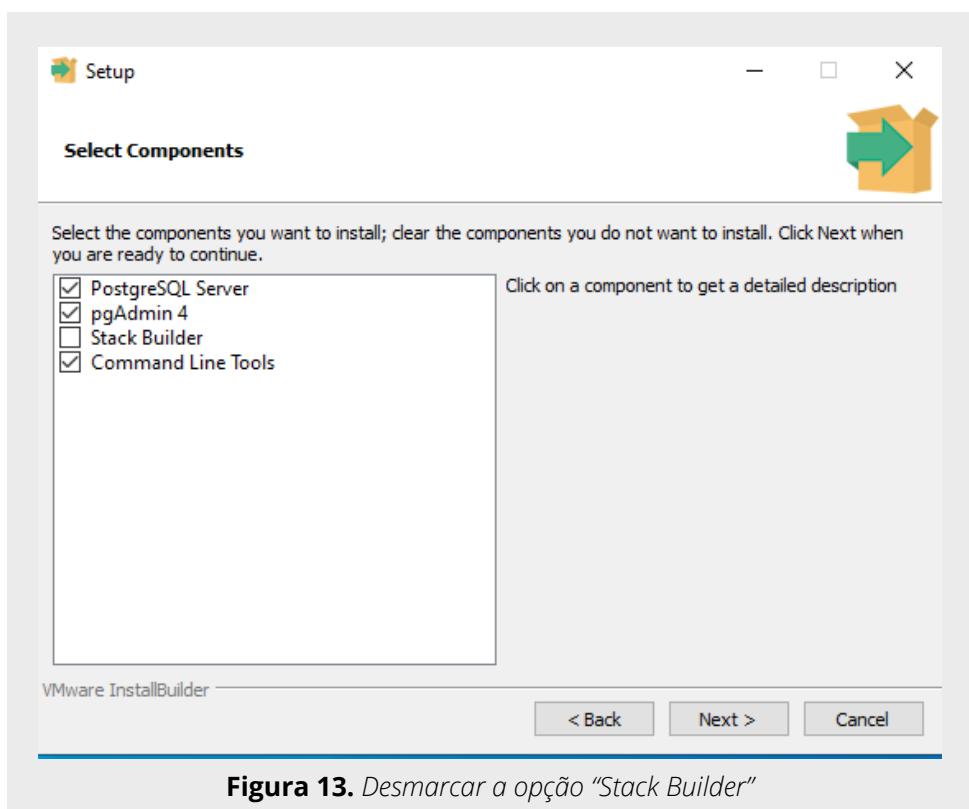


Figura 13. Desmarcar a opção "Stack Builder"

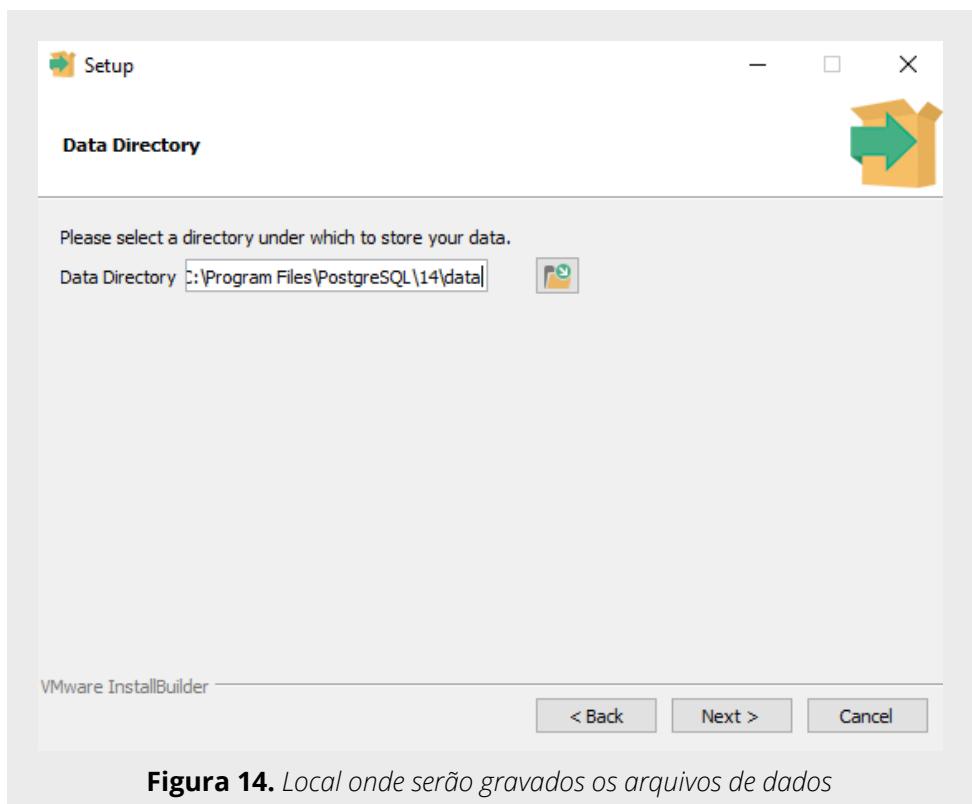


Figura 14. Local onde serão gravados os arquivos de dados

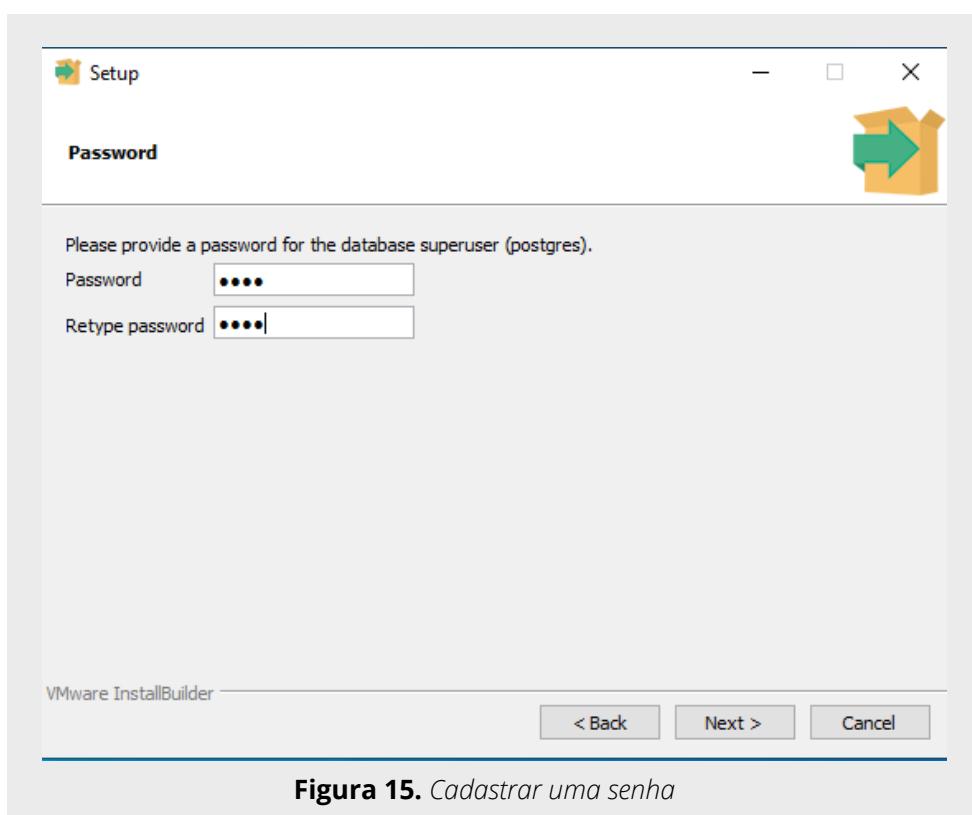


Figura 15. Cadastrar uma senha

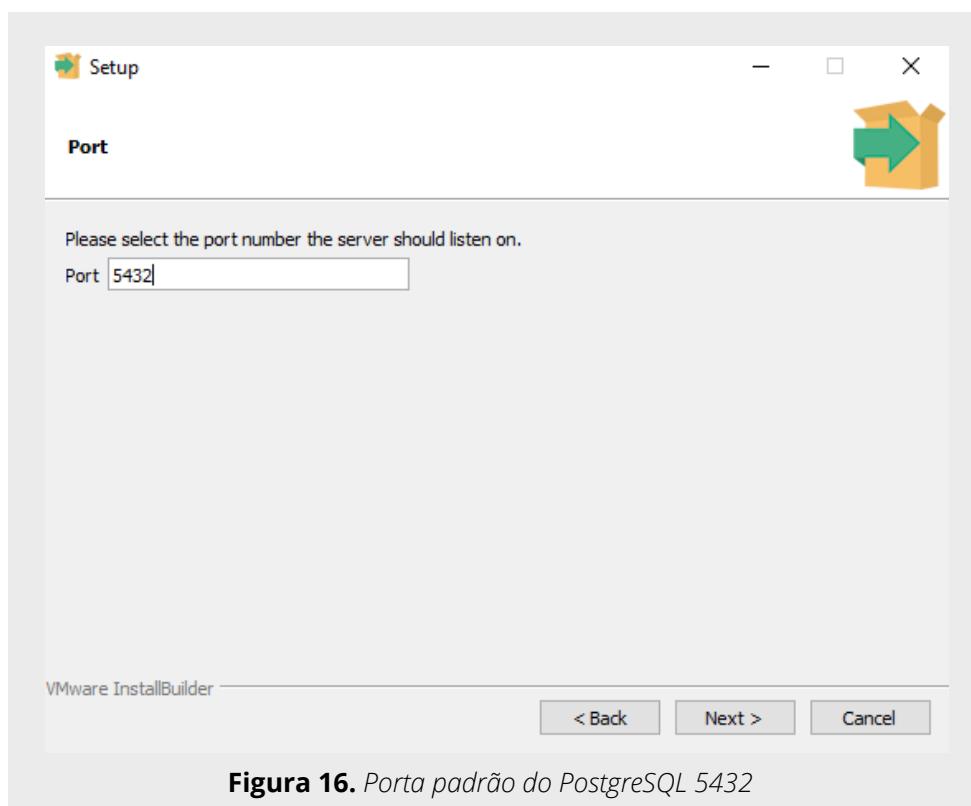


Figura 16. Porta padrão do PostgreSQL 5432

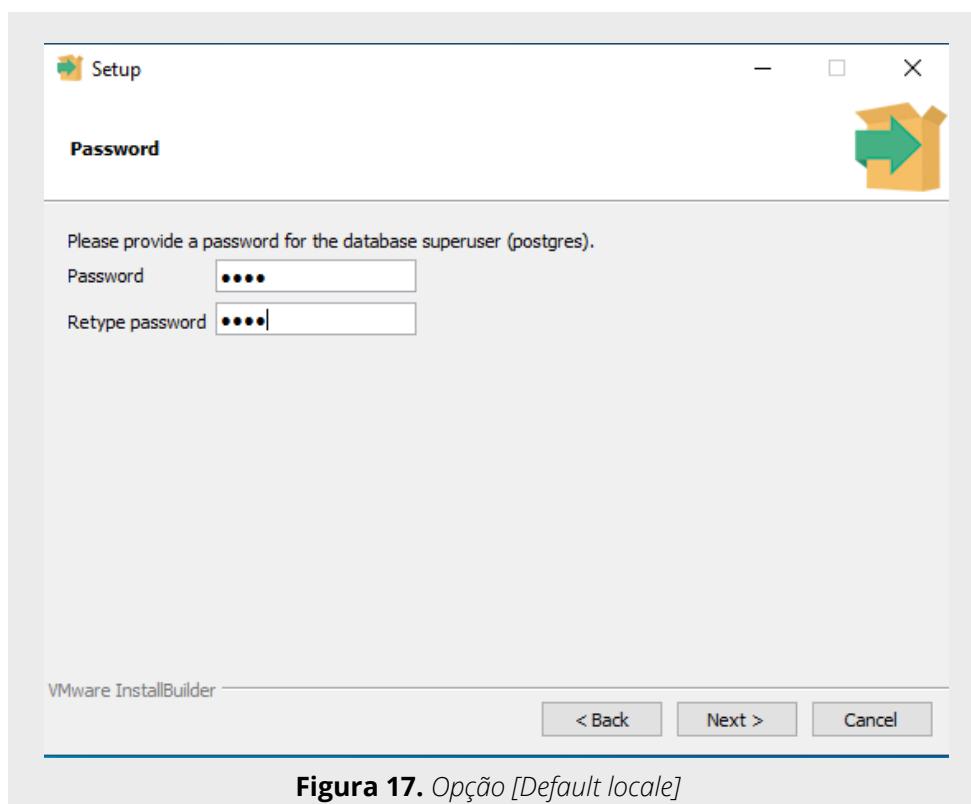


Figura 17. Opção [Default locale]

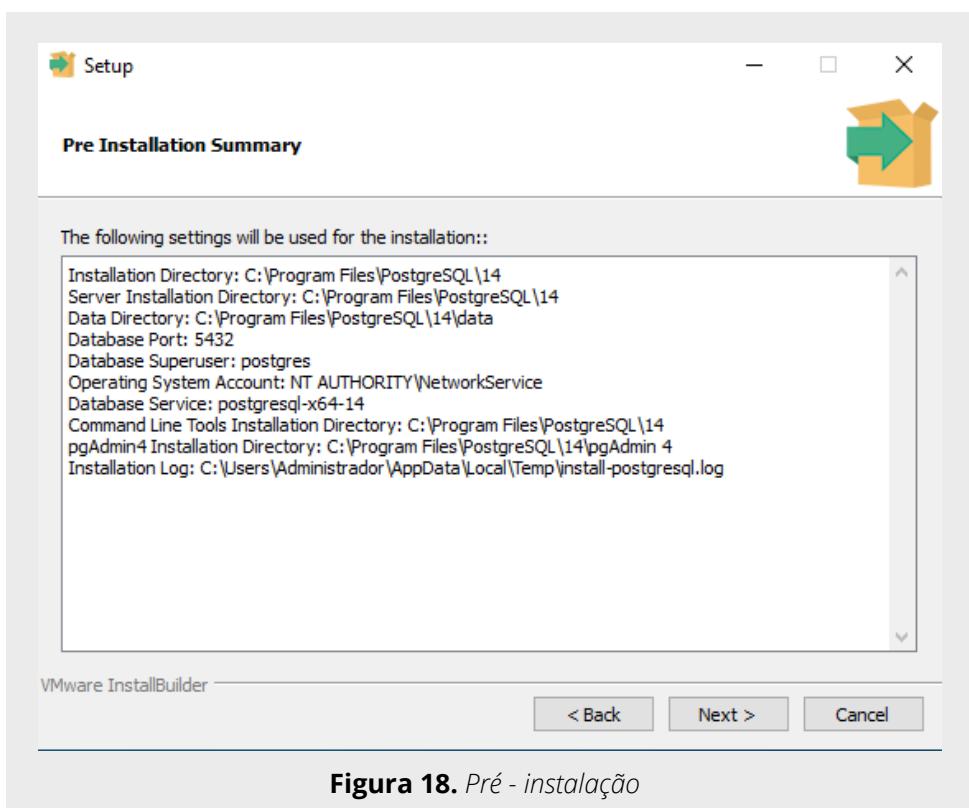


Figura 18. Pré - instalação

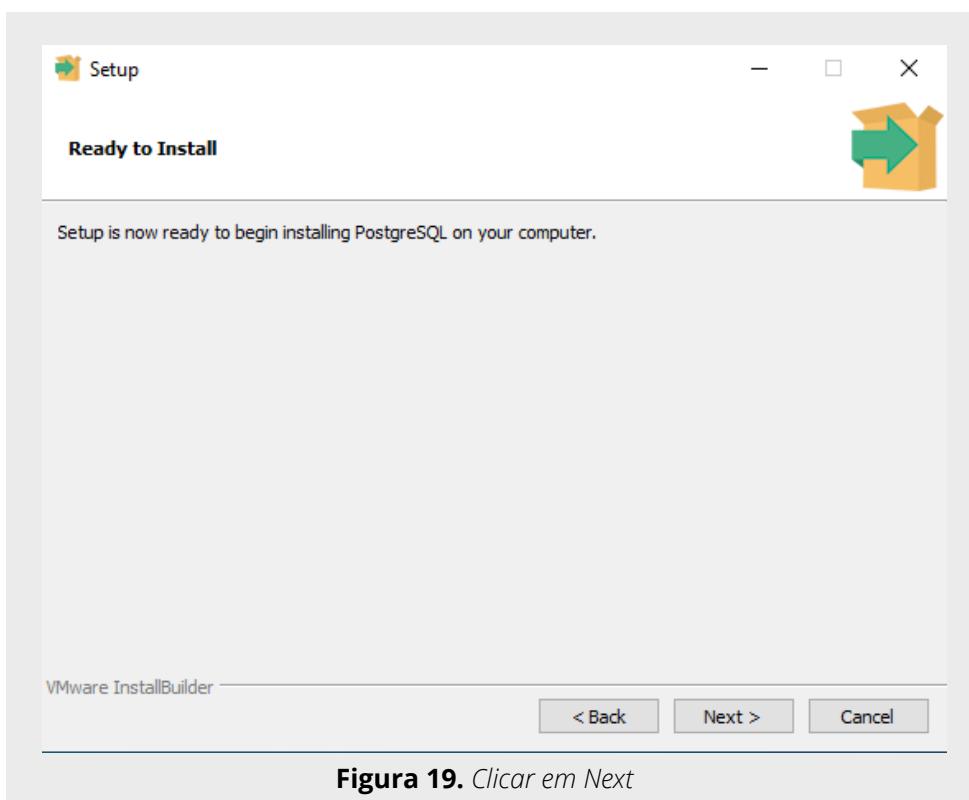


Figura 19. Clicar em Next

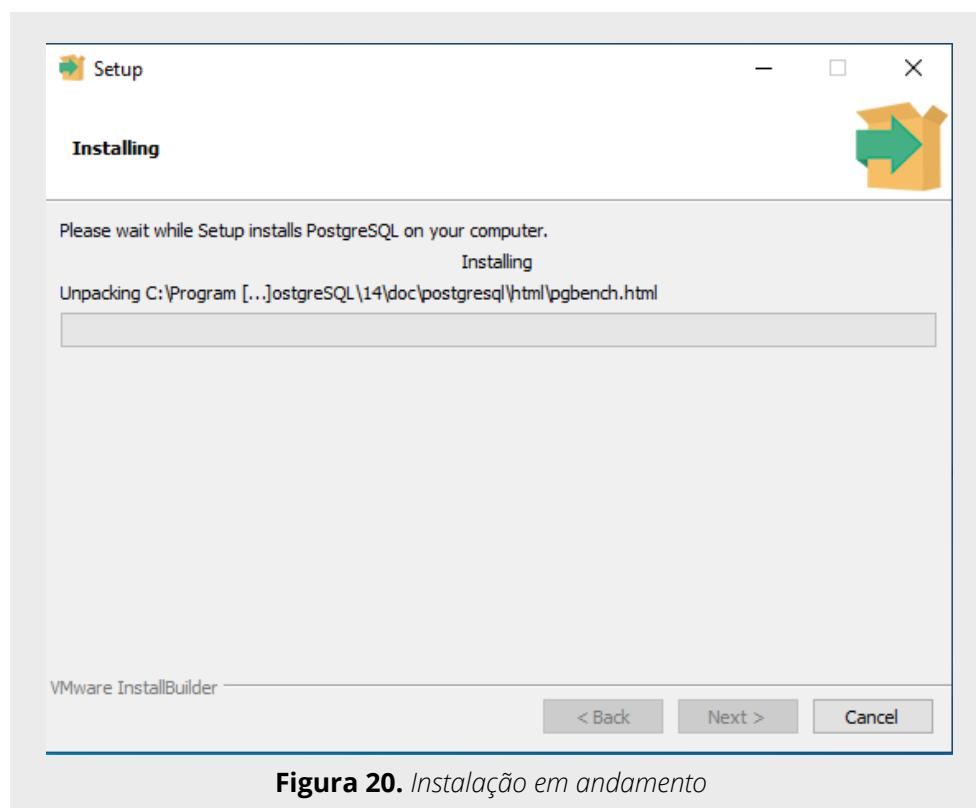


Figura 20. Instalação em andamento

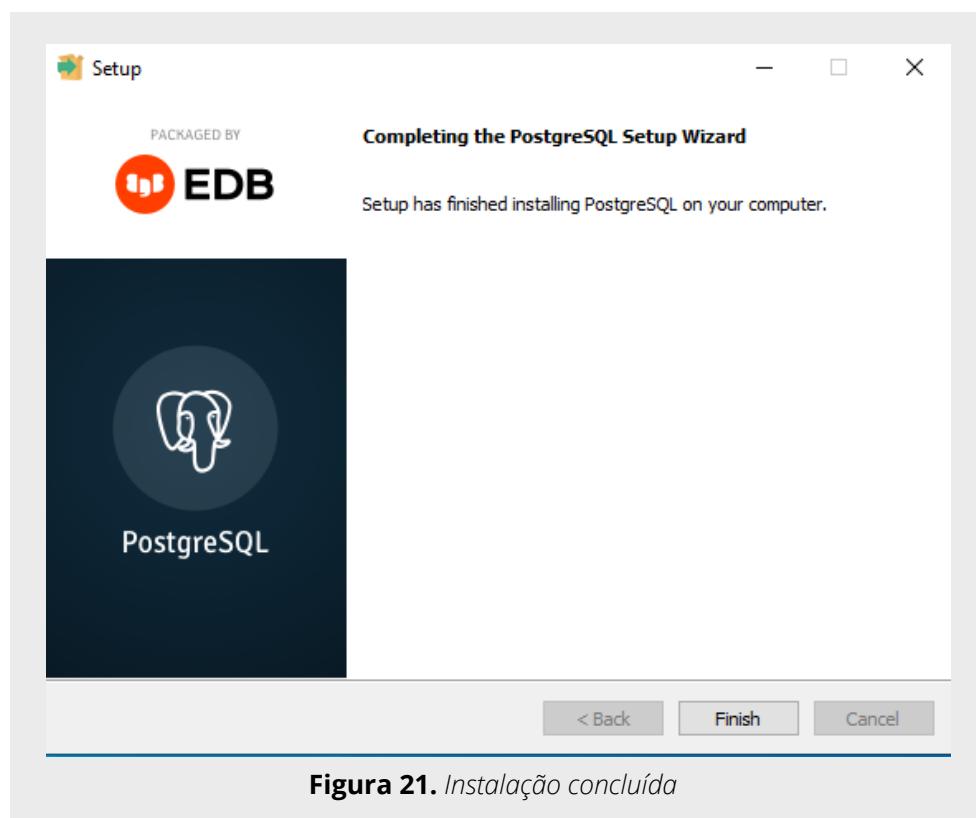


Figura 21. Instalação concluída

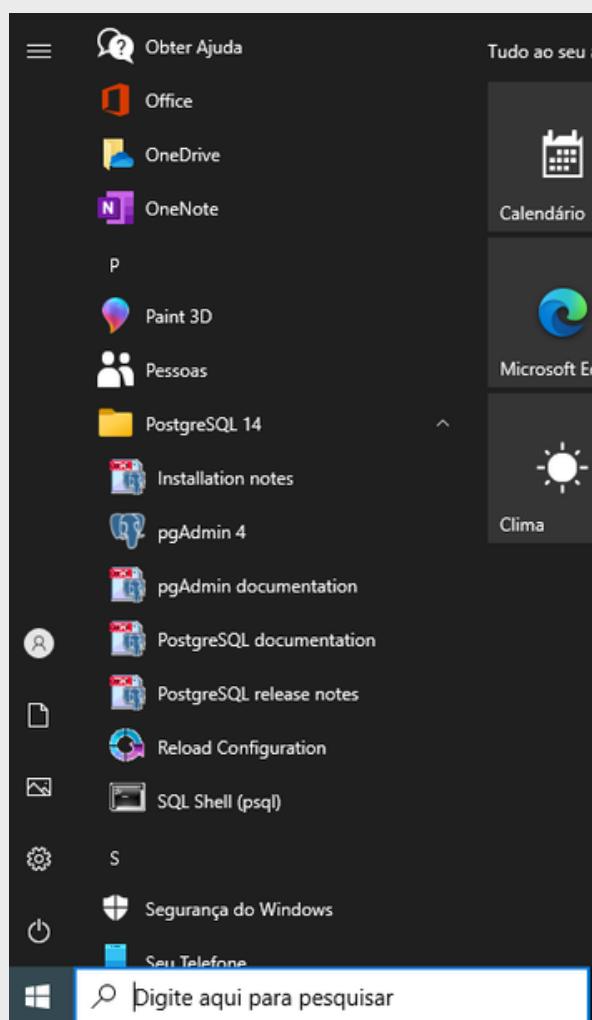


Figura 22. Abrir o pgAdmin 4 por meio do menu Iniciar

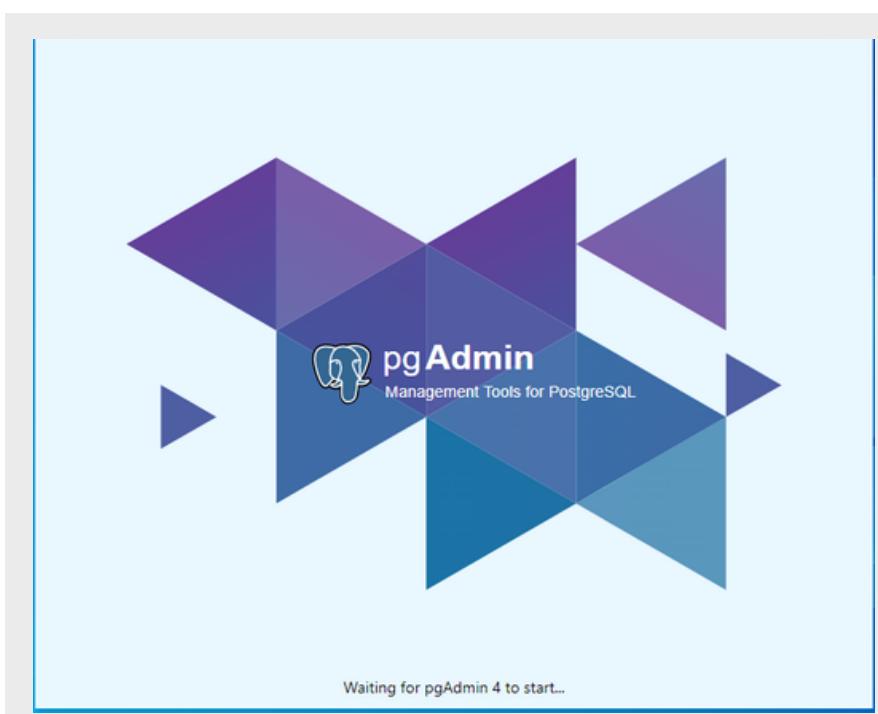


Figura 23. pgAdmin abrindo – no primeiro acesso há uma certa demora

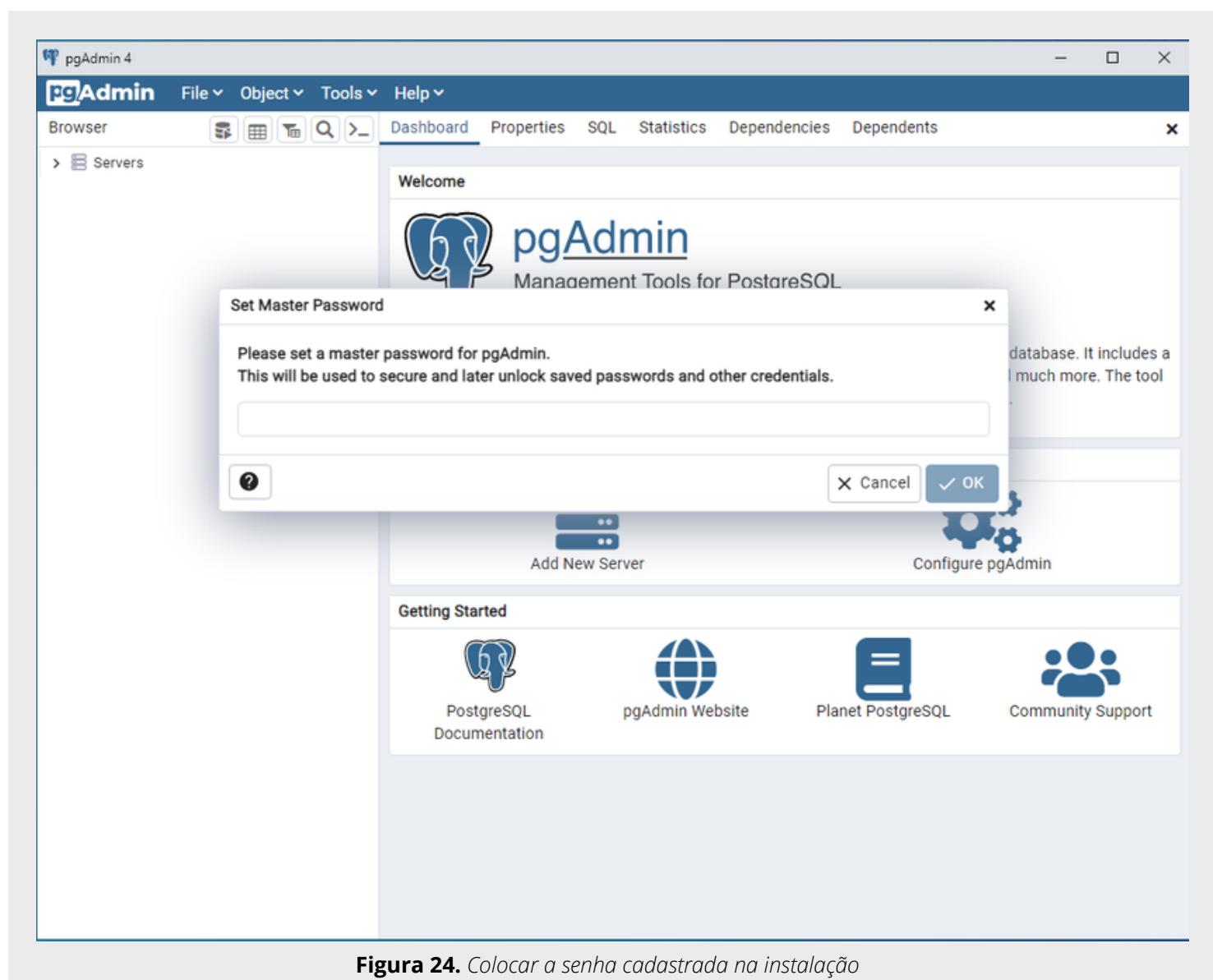


Figura 24. Colocar a senha cadastrada na instalação



Figura 25. Tela inicial do pgAdmin

5.1. Conhecendo o PgAdmin

O pgAdmin 4 suporta todos os recursos do PostgreSQL, desde a escrita de consultas SQL simples até o desenvolvimento de bancos de dados complexos. Ele é projetado para consultar um banco de dados ativo (em tempo real), permitindo que você fique atualizado com as modificações e implementações.

Os recursos do pgAdmin 4 incluem:

- detecção automática e suporte para objetos descobertos em tempo de execução;
- uma ferramenta de consulta SQL ao vivo com edição direta de dados;
- suporte para consultas administrativas;
- um editor SQL de realce de sintaxe;
- interfaces gráficas redesenhasadas;
- poderosas caixas de diálogo e ferramentas de gerenciamento para tarefas comuns;
- comportamento responsivo e sensível ao contexto;
- mensagens de erro de suporte;
- dicas úteis;
- ajuda on-line e informações sobre como usar as caixas de diálogo e ferramentas do pgAdmin.

Quando o pgAdmin é aberto, a interface apresenta uma barra de menus e uma janela dividida em dois painéis: o controle de árvore do *navegador* no painel esquerdo e um navegador com guias no painel direito.

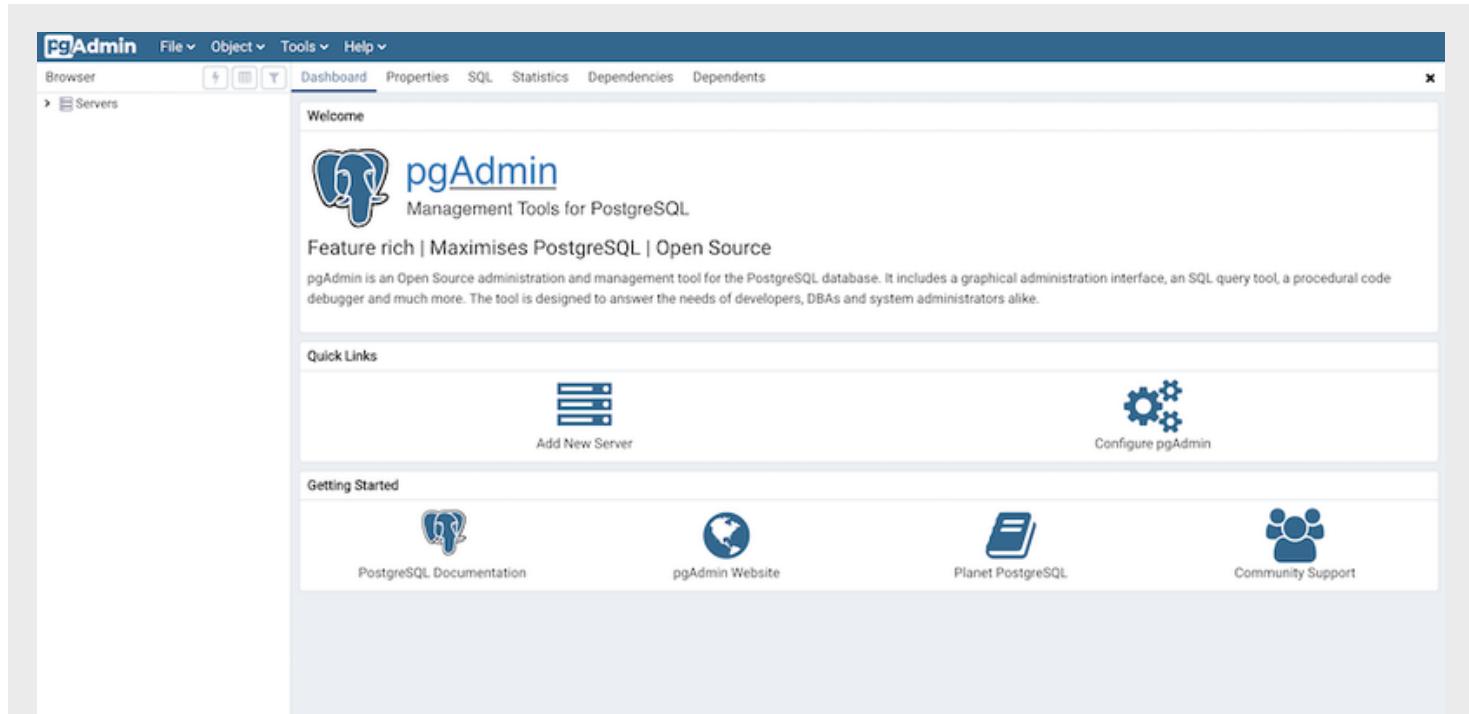


Figura 26. Tela inicial do pgAdmi

Selecione um ícone no painel *Links rápidos* na guia Painel para:

- Clique no botão *Adicionar novo servidor* para abrir a caixa de diálogo Criar - Servidor para adicionar uma nova definição de servidor.
- Clique no botão *Configurar pgAdmin* para abrir a caixa de diálogo *Preferências* para personalizar seu cliente pgAdmin.

Links no painel Introdução abrem uma nova guia do navegador que fornece informações úteis para usuários do PostgreSQL:

- Clique no link *Documentação do PostgreSQL* para navegar até a página Documentação do projeto de código aberto do PostgreSQL; uma vez no site do projeto, você pode revisar os manuais das versões atualmente suportadas do servidor PostgreSQL.
- Clique no link do *site pgAdmin* para navegar até o site do projeto pgAdmin. O site pgAdmin apresenta notícias sobre lançamentos recentes do pgAdmin e outras informações do projeto.
- Clique no link *Planet PostgreSQL* para navegar até o agregador de blogs para blogs relacionados ao PostgreSQL.
- Clique no link *Community Support* para navegar até a página *Community* no site do projeto de código aberto PostgreSQL; esta página fornece informações sobre como obter suporte para recursos do PostgreSQL.

• BARRA DE MENU

A barra de menu pgAdmin fornece menus suspensos para acesso a opções, comandos e utilitários. A barra de menus exibe as seguintes seleções: *File*, *Object*, *Tools** e *Help*. As seleções podem estar esmaecidas, o que indica que estão desabilitadas para o objeto atualmente selecionado no controle de árvore pgAdmin.

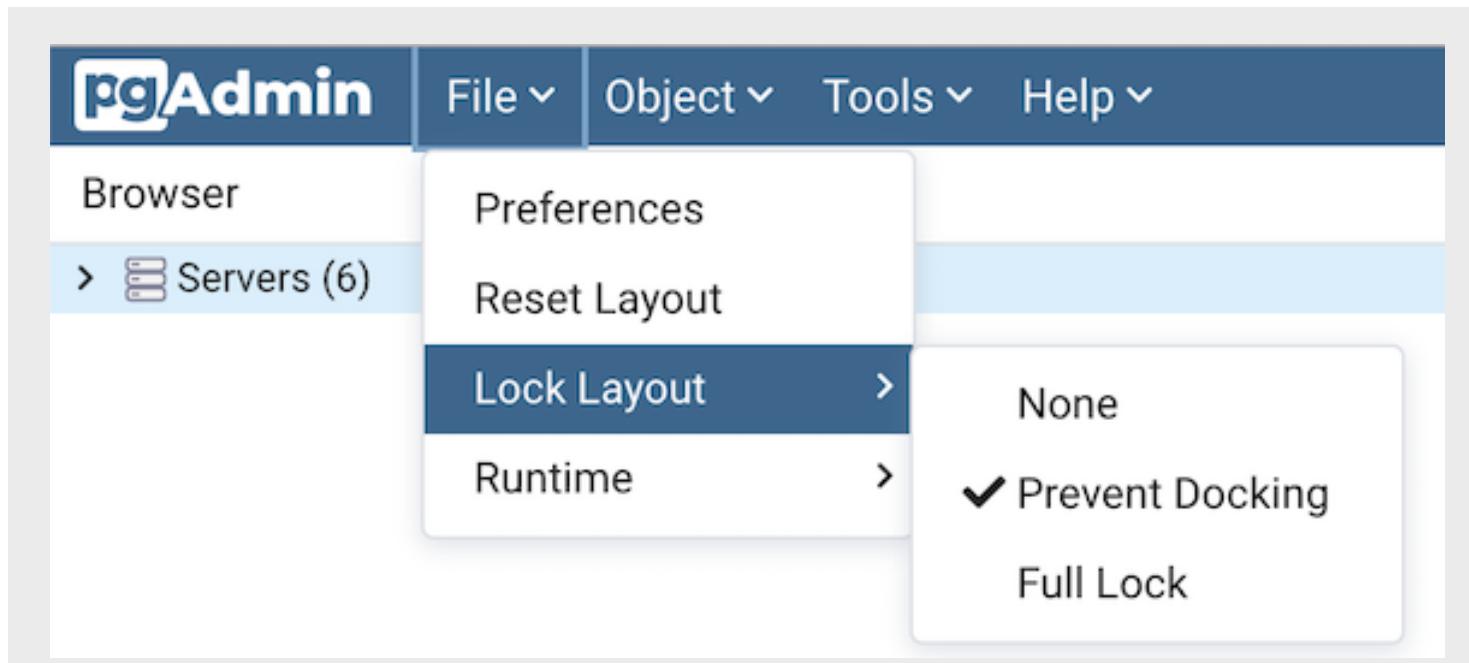


Figura 27. Use o menu Arquivo para acessar as seguintes opções:

Opção	Ação
Preferências	Clique para abrir a caixa de diálogo Preferências para personalizar as configurações do pgAdmin.
Redefinir Layout	Se você modificou a área de trabalho, clique para restaurar o layout padrão.
Esquema de Bloqueio	Carregue para abrir um submenu para selecionar o nível para bloquear a disposição da interface do utilizador. Isto também pode ser alterado a partir das preferências de visualização do navegador
Tempo de Execução	Clique para abrir um submenu para configurar, visualizar log e configurar zoom. Visível apenas quando o pgAdmin 4 é executado no modo desktop.

- **MENU OBJECT**

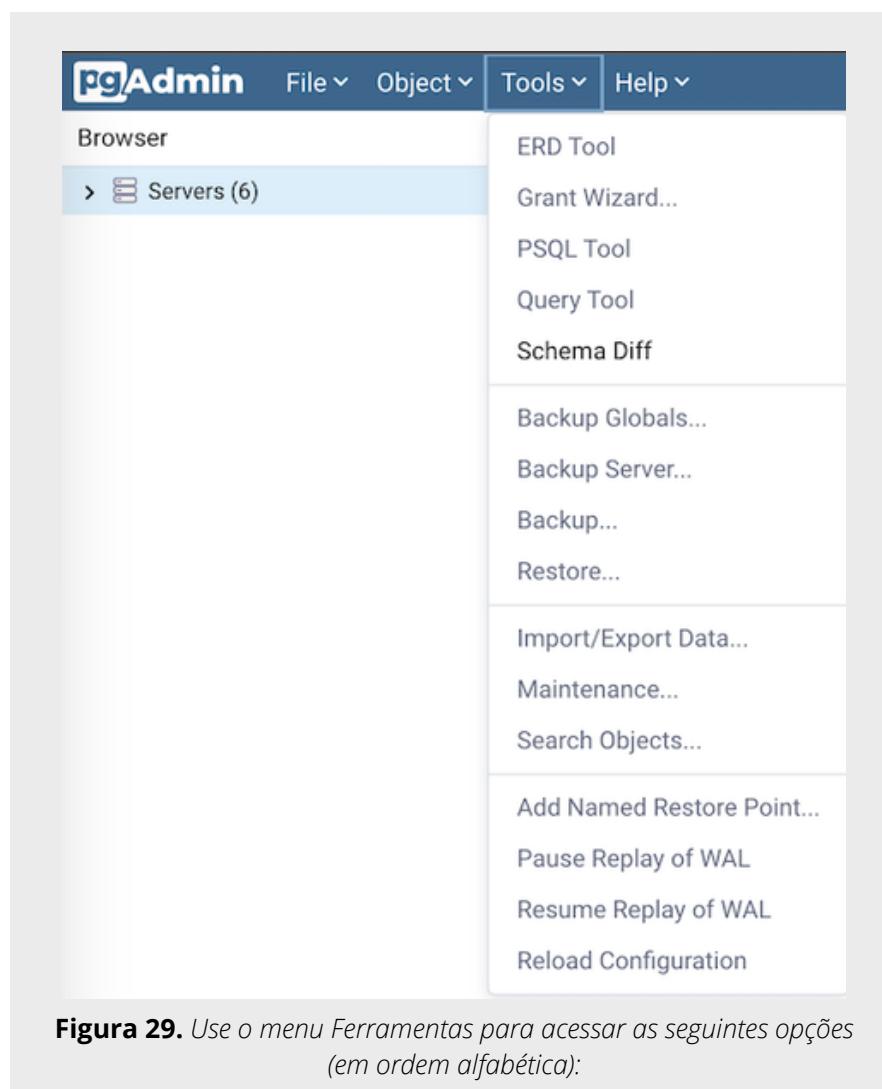


Figura 28. O menu Objeto é sensível ao contexto. Use o menu Objeto para acessar as seguintes opções (em ordem alfabética):

Opção	Ação
Mudar senha	Clique para abrir a caixa de diálogo Alterar senha... para alterar sua senha.
Limpar senha salva	Se você salvou a senha do servidor do banco de dados, clique para limpar a senha salva. Ative somente quando a senha já estiver salva.
Limpar senha do túnel SSH	Se você salvou a senha do túnel ssh, clique para limpar a senha salva. Ative somente quando a senha já estiver salva.
Conectar Servidor	Clique para abrir a caixa de diálogo Conectar ao Servidor para estabelecer uma conexão com um servidor.
Crio	Clique em <i>Criar</i> para acessar um menu de contexto que fornece seleções sensíveis ao contexto. Sua seleção abre uma caixa de diálogo <i>Criar</i> para criar um novo objeto.
Excluir/Remover	Clique para excluir o objeto atualmente selecionado do servidor.
Desconectar do Servidor	Clique para se desconectar do servidor atualmente selecionado.
Soltar cascata	Clique para excluir o objeto selecionado no momento e todos os objetos dependentes do servidor.
Propriedades	Clique para revisar ou modificar as propriedades do objeto selecionado no momento.
Atualizar	Clique para atualizar o objeto atualmente selecionado.
Remover Servidor	Clique para remover o servidor atualmente selecionado.
Scripts	Clique para abrir a ferramenta de consulta para editar ou visualizar o script selecionado no menu suspenso.
Gatilho(s)	Clique para <i>Desativar</i> ou <i>Ativar</i> gatilho(s) para a tabela selecionada no momento. As opções são exibidas no menu suspenso.

Truncar	Clique para remover todas as linhas de uma tabela (<i>Truncate</i>), para remover todas as linhas de uma tabela e suas tabelas filhas (<i>Truncate Cascade</i>) ou para remover todas as linhas de uma tabela e reiniciar automaticamente as sequências pertencentes às colunas (<i>Truncate Restart Identity</i>). As opções são exibidas no menu suspenso.
Ver dados	Clique para acessar um menu de contexto que oferece várias opções para visualização de dados (veja abaixo).
Gerar ERD	Clique para abrir a ferramenta ERD com diagrama gerado automaticamente para o banco de dados selecionado. Esta opção está disponível apenas quando um banco de dados é selecionado. As opções são exibidas no menu suspenso.

- **O MENU FERRAMENTA**



Opção	Ação
Ferramenta ERD	Clique para abrir a ferramenta ERD e começar a projetar seu banco de dados.
Assistente de concessão...	Clique para acessar a ferramenta Assistente de concessão.
Ferramenta PSQL	Clique para abrir a Ferramenta PSQL e iniciar o PSQL no contexto atual do banco de dados.
Ferramenta de Consulta	Clique para abrir a Ferramenta de consulta para o objeto selecionado no momento.

Diferença de Esquema	Clique para abrir o Schema Diff e comece a comparar dois bancos de dados ou dois esquemas.
Backup Globals...	Clique para abrir a caixa de diálogo Backup Globals... para fazer backup de objetos de cluster.
Servidor de backup...	Clique para abrir a caixa de diálogo Backup Server... para fazer backup de um servidor.
Cópia de segurança...	Clique para abrir a caixa de diálogo Backup... para fazer backup dos objetos do banco de dados.
Restaurar...	Clique para acessar a caixa de diálogo Restaurar para restaurar arquivos de banco de dados de um backup.
Importar/Exportar dados...	Clique para abrir a caixa de diálogo Importar/Exportar dados... para importar ou exportar dados de uma tabela.
Manutenção...	Clique para abrir a caixa de diálogo Manutenção... para VACUUM, ANALYZE, REINDEX ou CLUSTER.
Pesquisar objetos...	Clique para abrir os Objetos de Pesquisa... e comece a pesquisar qualquer tipo de objeto em um banco de dados.
Adicionar ponto de restauração nomeado	Clique para abrir a caixa de diálogo Adicionar ponto de restauração nomeado... para obter um instantâneo pontual do estado atual do servidor.
Pausar repetição de WAL	Clique para pausar a reprodução do log WAL.
Retomar a reprodução do WAL	Clique para retomar a reprodução do log WAL.
Recarregar configuração...	Clique para atualizar os arquivos de configuração sem reiniciar o servidor.
Gerente de armazenamento	Clique para abrir o Storage Manager para carregar, excluir ou baixar os arquivos de backup.

- **O MENU AJUDA**

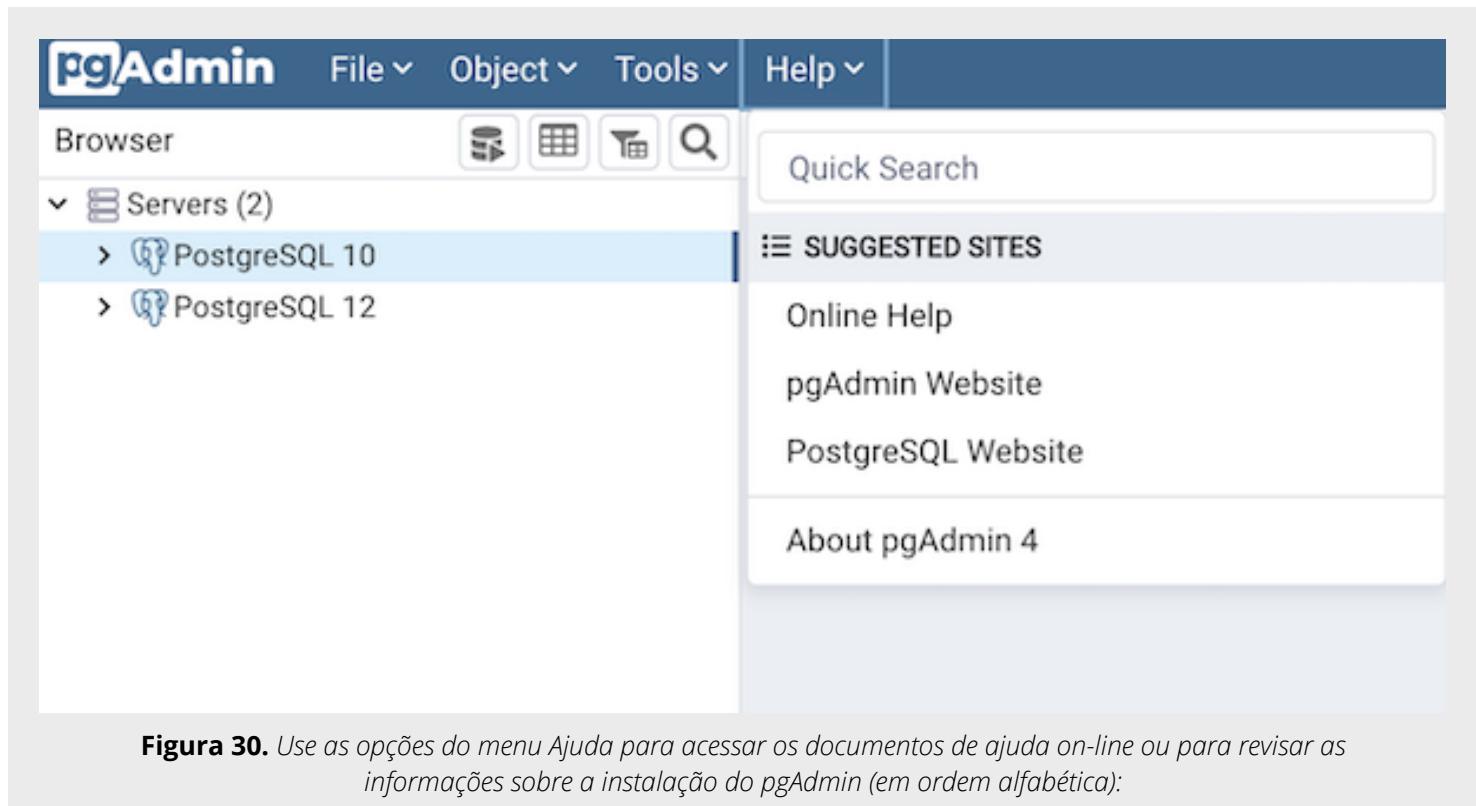


Figura 30. Use as opções do menu Ajuda para acessar os documentos de ajuda on-line ou para revisar as informações sobre a instalação do pgAdmin (em ordem alfabética):

Ação	Opção
Pesquisa Rápida	<p>Digite suas palavras-chave no campo <i>Pesquisa Rápida</i>. Digitar pelo menos três caracteres exibirá todas as possibilidades de correspondência em <i>itens de menu</i> e os documentos relevantes em <i>artigos de ajuda</i>. Clique nas opções em <i>itens de menu</i> para executar a ação de uma funcionalidade ou objeto específico. Clique em qualquer um dos <i>artigos da ajuda</i> para abrir a ajuda desse tópico com o texto destacado em uma janela separada.</p> <p>Nota: se alguma das opções nos itens do menu estiver desativada, esse tópico fornecerá informações por meio do ícone de <i>informações</i>.</p>
Sobre pgAdmin 4	Clique para abrir uma janela na qual você encontrará informações sobre o pgAdmin; isso inclui a versão atual e o usuário atual.

Ação	Opção
Ajuda on-line	Clique para abrir o suporte de documentação para usar os utilitários, ferramentas e diálogos do pgAdmin. Navegue (na guia recém-aberta) documentos de ajuda no painel esquerdo do navegador ou use a barra de pesquisa para especificar um tópico.
Site pgAdmin	Clique para abrir o site <i>pgAdmin.org</i> em uma janela do navegador.
Site PostgreSQL	Clique para acessar a documentação principal do PostgreSQL hospedada no site do PostgreSQL. O site também oferece guias, tutoriais e recursos

- **BARRA DE FERRAMENTAS** 

A barra de ferramentas do pgAdmin fornece botões de atalho para recursos usados com frequência, como *Exibir dados* e a *Ferramenta de consulta*, que são usados com mais frequência no programa. Esta barra de ferramentas é visível no painel do navegador. Os botões são ativados/desativados com base no nó do navegador selecionado.



Figura 31. Barra de Ferramentas

- Use o botão *Ferramenta de Consulta* para abrir a Ferramenta de Consulta no contexto atual do banco de dados.
- Use o botão *Visualizar* dados para visualizar/editar os dados armazenados em uma tabela selecionada.
- Use o botão *Linhas Filtradas* para acessar o pop-up *Filtro de Dados* para aplicar um filtro a um conjunto de dados para visualização/edição.
- Use o botão *Pesquisar* objetos para acessar a caixa de diálogo de objetos de pesquisa. Ele ajuda você a pesquisar qualquer objeto de banco de dados.
- Use o botão *PSQL Tool* para abrir o PSQL no contexto atual do banco de dados.

- **NAVEGADOR COM ABAS** 

O painel direito da janela *pgAdmin* apresenta uma coleção de guias que exibem informações sobre o objeto atualmente selecionado no controle de árvore *pgAdmin* na janela esquerda. Selecione uma guia para acessar informações sobre o objeto realçado no controle de árvore.

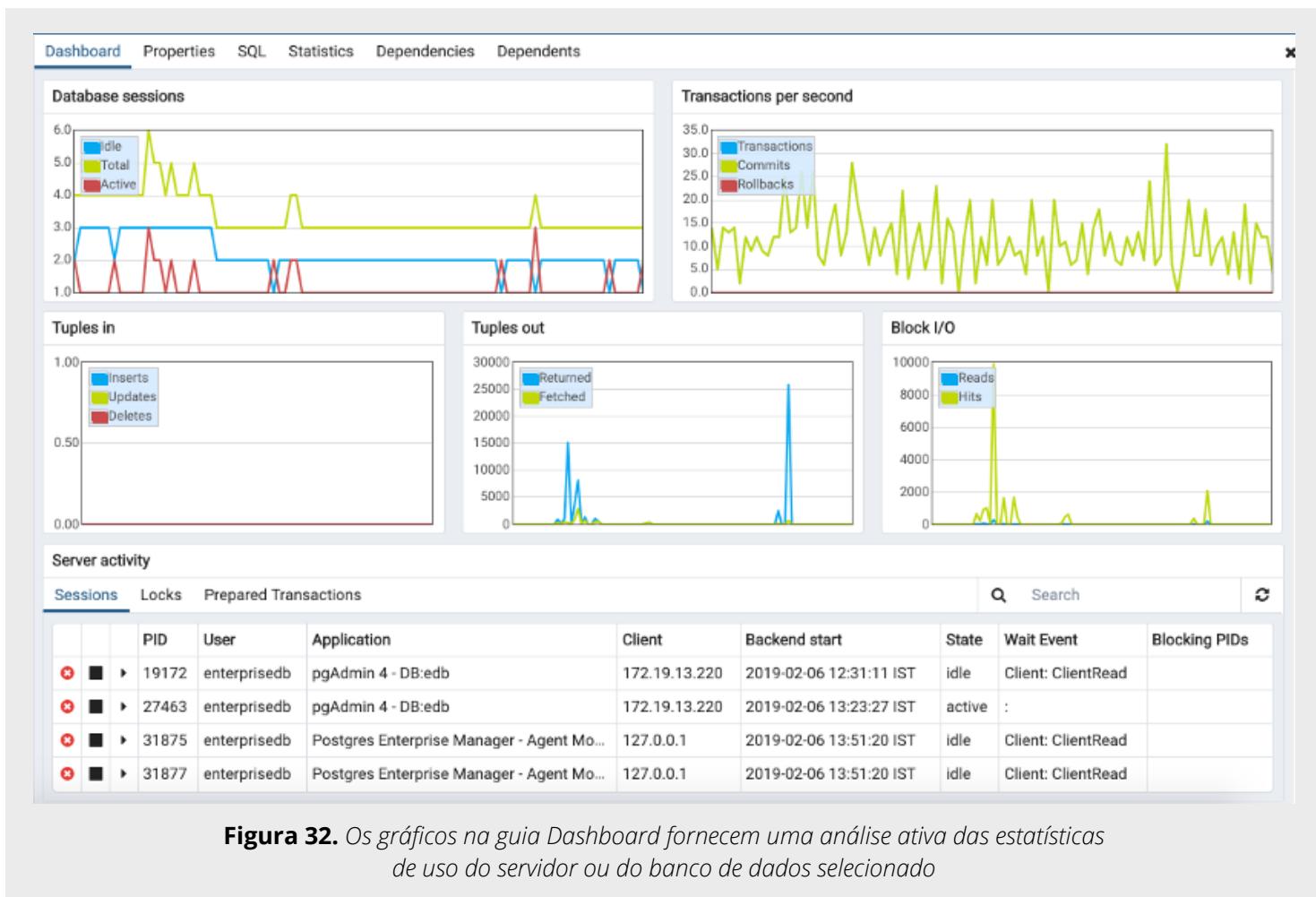


Figura 32. Os gráficos na guia Dashboard fornecem uma análise ativa das estatísticas de uso do servidor ou do banco de dados selecionado

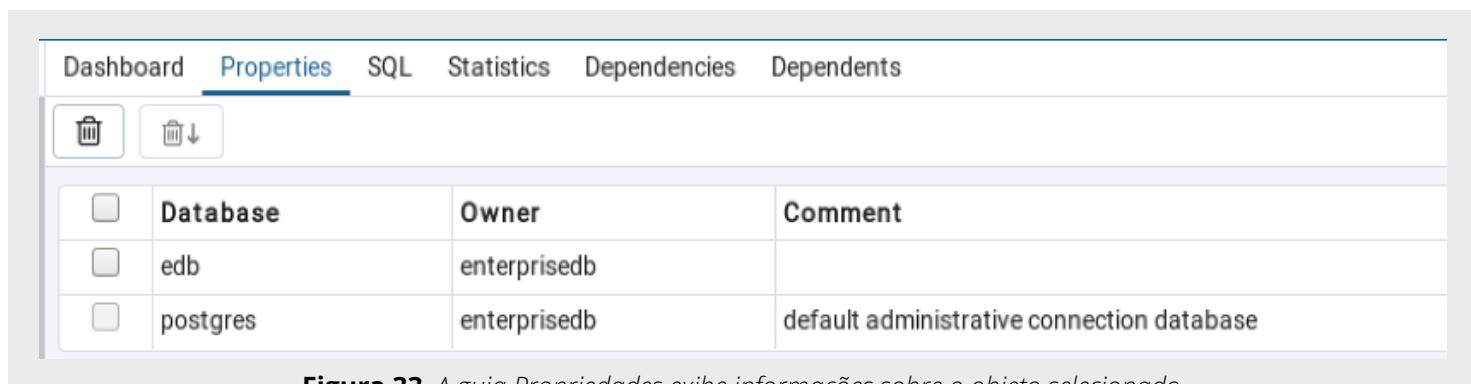
- O gráfico Sessões do servidor ou Sessões do banco de dados exibe as interações com o servidor ou banco de dados.
- O gráfico Transações por segundo exibe as confirmações, reversões e o total de transações por segundo que estão ocorrendo no servidor ou no banco de dados.
- O gráfico Tuplas no gráfico exibe o número de tuplas inseridas, atualizadas e excluídas no servidor ou no banco de dados.
- O gráfico de saída de tuplas exibe o número de tuplas buscadas e retornadas do servidor ou no banco de dados.
- O gráfico Block I/O exibe o número de blocos lidos do sistema de arquivos ou buscados no cache de buffer (mas não no cache do sistema de arquivos do sistema operacional) para o servidor ou banco de dados.

O painel *Atividade do servidor* exibe informações sobre sessões, bloqueios, transações preparadas e configuração do servidor (se aplicável). As informações são apresentadas em tabelas sensíveis ao contexto. Use os controles localizados acima da tabela para:

- Clique no botão *Atualizar* para atualizar as informações exibidas em cada tabela.
- Insira um valor na caixa *Pesquisar* para restringir o conteúdo da tabela a uma ou mais sessões que satisfaçam os critérios de pesquisa. Por exemplo, você pode inserir um ID de processo para localizar uma sessão específica ou um estado de sessão (como ocioso) para localizar todas as sessões que estão em estado ocioso.

Você pode usar ícones na tabela *Sessões* para revisar ou controlar o estado de uma sessão:

- Use o ícone *Terminar* (localizado na primeira coluna) para interromper uma sessão e removê-la da tabela. Antes que o servidor encerre a sessão, será solicitado que você confirme a sua seleção.
- Use o ícone *Cancelar* (localizado na segunda coluna) para encerrar uma consulta ativa sem fechar a sessão. Antes de cancelar a consulta, o servidor solicitará que você confirme sua seleção. Ao cancelar uma consulta, o valor exibido na coluna *State* da tabela será atualizado de *Active* para *Idle*. A sessão permanecerá na tabela até que seja encerrada.
- Use o ícone *Detalhes* (localizado na terceira coluna) para abrir a guia *Detalhes*. Essa guia exibe informações sobre a sessão selecionada.



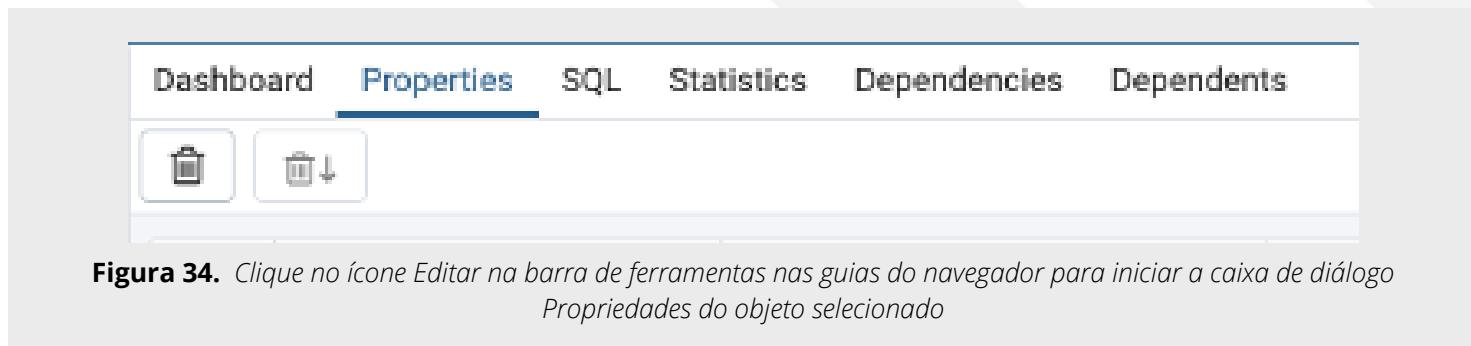
The screenshot shows the PgAdmin interface with the 'Properties' tab selected. The toolbar has buttons for 'Delete' and 'Delete & cascade'. The table lists databases:

	Database	Owner	Comment
<input type="checkbox"/>	edb	enterprisedb	
<input type="checkbox"/>	postgres	enterprisedb	default administrative connection database

Figura 33. A guia *Propriedades* exibe informações sobre o objeto selecionado

Clique no ícone *Excluir* na barra de ferramentas na guia do navegador para excluir os objetos selecionados no painel *Propriedades*.

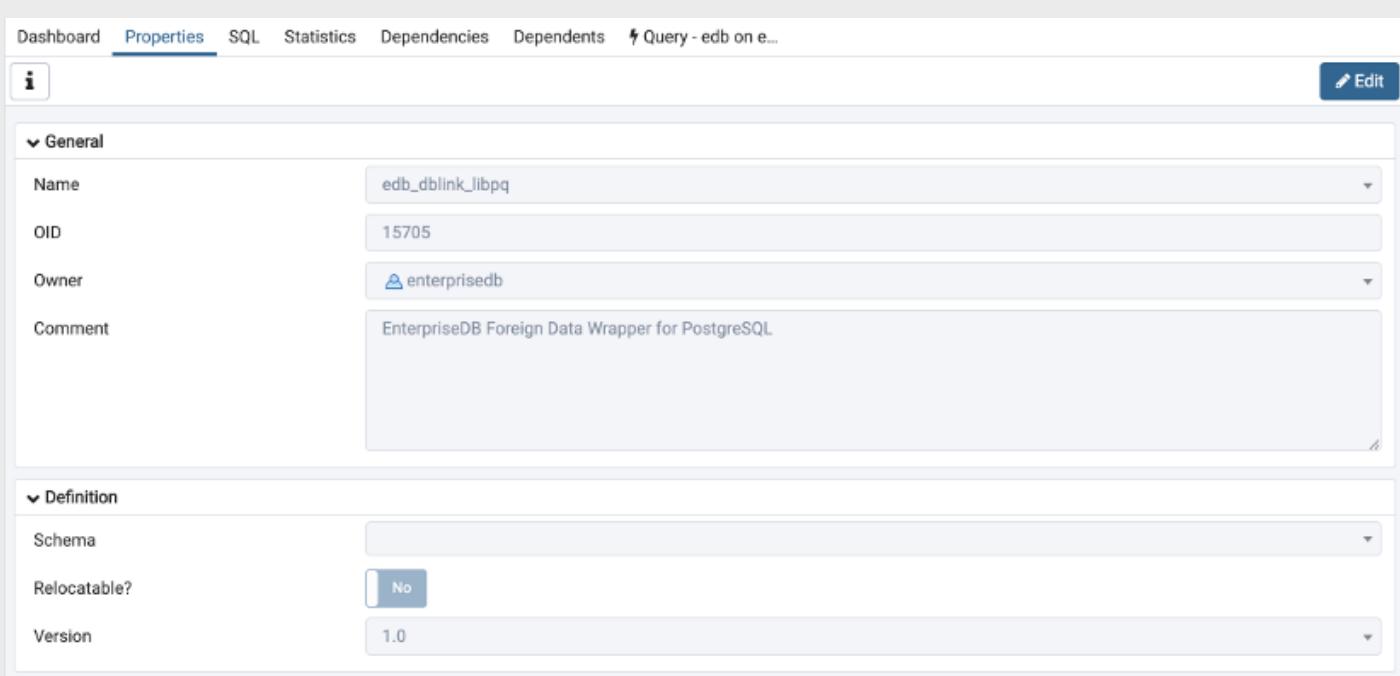
Clique no ícone *Soltar Cascata* na barra de ferramentas na guia do navegador para excluir os objetos selecionados e todos os objetos dependentes no painel *Propriedades*.



The screenshot shows the PgAdmin interface with the 'Properties' tab selected. The toolbar has buttons for 'Delete' and 'Delete & cascade'.

Figura 34. Clique no ícone *Editar* na barra de ferramentas nas guias do navegador para iniciar a caixa de diálogo *Propriedades* do objeto selecionado

Para preservar quaisquer alterações na caixa de diálogo *Propriedades*, clique no ícone *Salvar*; suas modificações serão exibidas na guia *Propriedades atualizadas*.



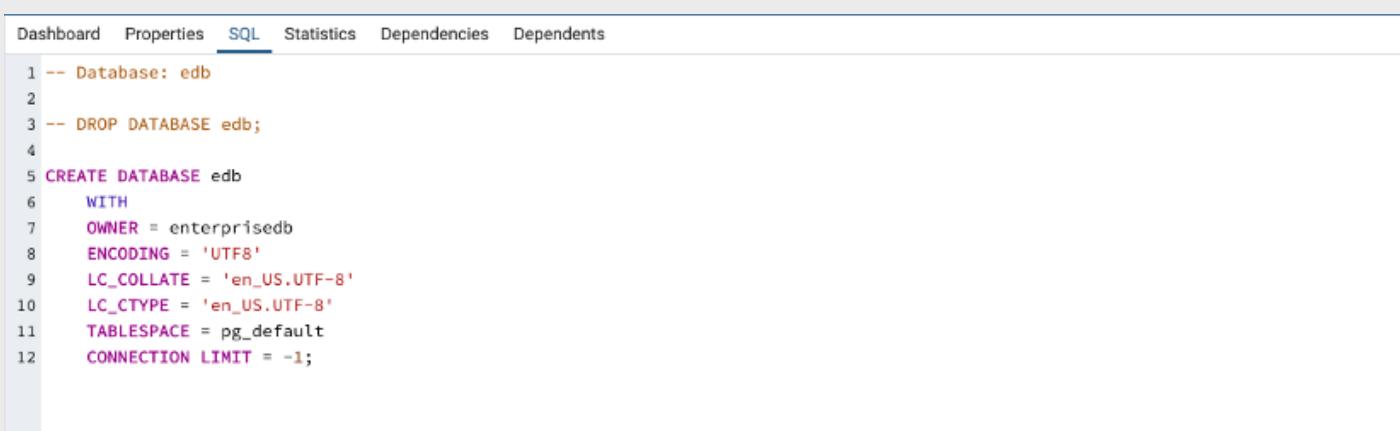
General

Name	edb_dblink_jibpq
OID	15705
Owner	enterpriseDB
Comment	EnterpriseDB Foreign Data Wrapper for PostgreSQL

Definition

Schema	
Relocatable?	No
Version	1.0

Figura 35. Os detalhes sobre o objeto destacado no controle de árvore são exibidos em um ou mais painéis recolhíveis. Você pode usar a seta à esquerda de cada rótulo de painel para abrir ou fechar um painel



```

1 -- Database: edb
2
3 -- DROP DATABASE edb;
4
5 CREATE DATABASE edb
6   WITH
7     OWNER = enterpriseDB
8     ENCODING = 'UTF8'
9     LC_COLLATE = 'en_US.UTF-8'
10    LC_CTYPE = 'en_US.UTF-8'
11    TABLESPACE = pg_default
12    CONNECTION LIMIT = -1;
  
```

Figura 36. A guia SQL exibe o script SQL que criou o objeto realçado e, quando aplicável, uma instrução SQL (comentada) que *DROP* o objeto selecionado. Você pode copiar as instruções SQL para o editor de sua escolha usando atalhos de recortar e colar.

Dashboard	Properties	SQL	<u>Statistics</u>	Dependencies	Dependents	Query - e...
Statistics						Value
Backends			5			
Xact committed			92095209			
Xact rolled back			36			
Blocks read			5269146			
Blocks hit			15431026716			
Tuples returned			22491999300			
Tuples fetched			6469655717			
Tuples inserted			13529			
Tuples updated			152			
Tuples deleted			98			
Last statistics reset			2018-12-21 14:30:08.829322+05:30			
Tablespace conflicts			0			
Lock conflicts			0			
Snapshot conflicts			0			
Bufferpin conflicts			0			
Deadlock conflicts			0			
Temporary files			0			
Size of temporary files			0 bytes			
Deadlocks			0			
Block read time			0			
Block write time			0			
Size			18 MB			

Figura 37. Estatísticas

A guia *Estatísticas* exibe as estatísticas coletadas para cada objeto no controle de árvore; as estatísticas exibidas na tabela variam de acordo com o tipo de objeto selecionado. Clique em um cabeçalho de coluna para classificar a tabela pelos dados exibidos na coluna; clique novamente para inverter a ordem de classificação. A tabela a seguir lista algumas das estatísticas disponíveis:

Painel	Descrição
PID	O ID do processo associado à linha.
Do utilizador	O nome do usuário que possui o objeto.
Base de dados	exibe o nome do banco de dados.
Back-Ends	exibe o número de conexões atuais com o banco de dados.
Início do back-end	A hora de início do processo de back-end.
Xact Comprometido	exibe o número de transações confirmadas no banco de dados na última semana.

Painel	Descrição
Xact revertido	exibe o número de transações revertidas na última semana.
Blocos lidos	exibe o número de blocos lidos da memória (em megabytes) na última semana.
Blocos atingidos	exibe o número de blocos atingidos no cache (em megabytes) na última semana.
Tuplas Devolvidas	exibe o número de tuplas retornadas na última semana.
Tuplas Buscadas	exibe o número de tuplas buscadas na última semana.
Tuplas inseridas	exibe o número de tuplas inseridas no banco de dados na última semana.
Tuplas atualizadas	exibe o número de tuplas atualizadas no banco de dados na última semana.
Tuplas excluídas	exibe o número de tuplas excluídas do banco de dados na última semana.
Última redefinição de estatísticas	exibe a hora da última redefinição de estatísticas para o banco de dados.
Conflitos de tablespace	exibe o número de consultas canceladas devido ao conflito de recuperação com tablespaces descartados no banco de dados.
Bloquear conflitos	exibe o número de consultas canceladas devido ao conflito de recuperação com bloqueios no banco de dados.
Conflitos de instantâneo	exibe o número de consultas canceladas devido ao conflito de recuperação com instantâneos antigos no banco de dados.
Conflitos de buffer	exibe o número de consultas canceladas devido ao conflito de recuperação com buffers fixados no banco de dados.

Painel	Descrição
Arquivos temporários	exibe o número total de arquivos temporários, incluindo aqueles usados pelo coletor de estatísticas.
Tamanho dos arquivos temporários	exibe o tamanho dos arquivos temporários.
Impasses	exibe o número de consultas canceladas devido a um conflito de recuperação com deadlocks no banco de dados.
Bloquear o tempo de leitura	exibe o número de milissegundos necessários para ler os blocos lidos.
Bloquear tempo de gravação	exibe o número de milissegundos necessários para escrever os blocos lidos.
Tamanho	exibe o tamanho (em megabytes) do banco de dados selecionado.

Dashboard	Properties	SQL	Statistics	Dependencies	Dependents	
Type					Name	Restriction
❖ Schema					public	normal

Figura 38. Dependências

A guia *Dependências* exibe os objetos dos quais o objeto selecionado atualmente depende. Se uma dependência for eliminada, o objeto atualmente selecionado no controle de árvore pgAdmin será afetado. Para garantir a integridade de toda a estrutura do banco de dados, o servidor de banco de dados garante que você não descarte accidentalmente objetos dos quais outros objetos dependem; você deve usar o comando `DROP CASCADE` para remover um objeto com uma dependência.

A tabela *Dependências* exibe as seguintes informações:

- O campo *Tipo* especifica o tipo de objeto pa;
- O campo *Nome* especifica o nome de identificação do objeto pai;
- O campo *Restrição* descreve a relação de dependência entre o objeto atualmente selecionado e o pai:
 - Se o campo for *auto*, o objeto selecionado pode ser descartado separadamente do objeto pai e será descartado se o objeto pai for descartado;
 - Se o campo for *internal*, o objeto selecionado foi criado durante a criação do objeto pai e será descartado se o objeto pai for descartado;
 - Se o campo for *normal*, o objeto selecionado pode ser descartado sem descartar o objeto pai;
 - Se o campo estiver em *branco*, o objeto selecionado é exigido pelo sistema e não pode ser descartado.

Dashboard	Properties	SQL	Statistics	Dependencies	<u>Dependents</u>
Type	Name			Restriction	
✓ Check	public.spatial_ref_sys_srid_check			auto	
⚠ Primary Key	public.spatial_ref_sys_pkey			auto	
✓ Check	public.spatial_ref_sys_srid_check			normal	

Figura 39. Dependentes

A guia *Dependentes* exibe uma tabela de objetos que dependem do objeto atualmente selecionado no navegador *pgAdmin*. Um objeto dependente pode ser descartado sem afetar o objeto atualmente selecionado no controle de árvore *pgAdmin*.

- O campo *Tipo* especifica o tipo de objeto dependente;
- O campo *Nome* especifica o nome de identificação do objeto dependente;
- O campo *Banco de dados* especifica o banco de dados no qual o objeto reside.

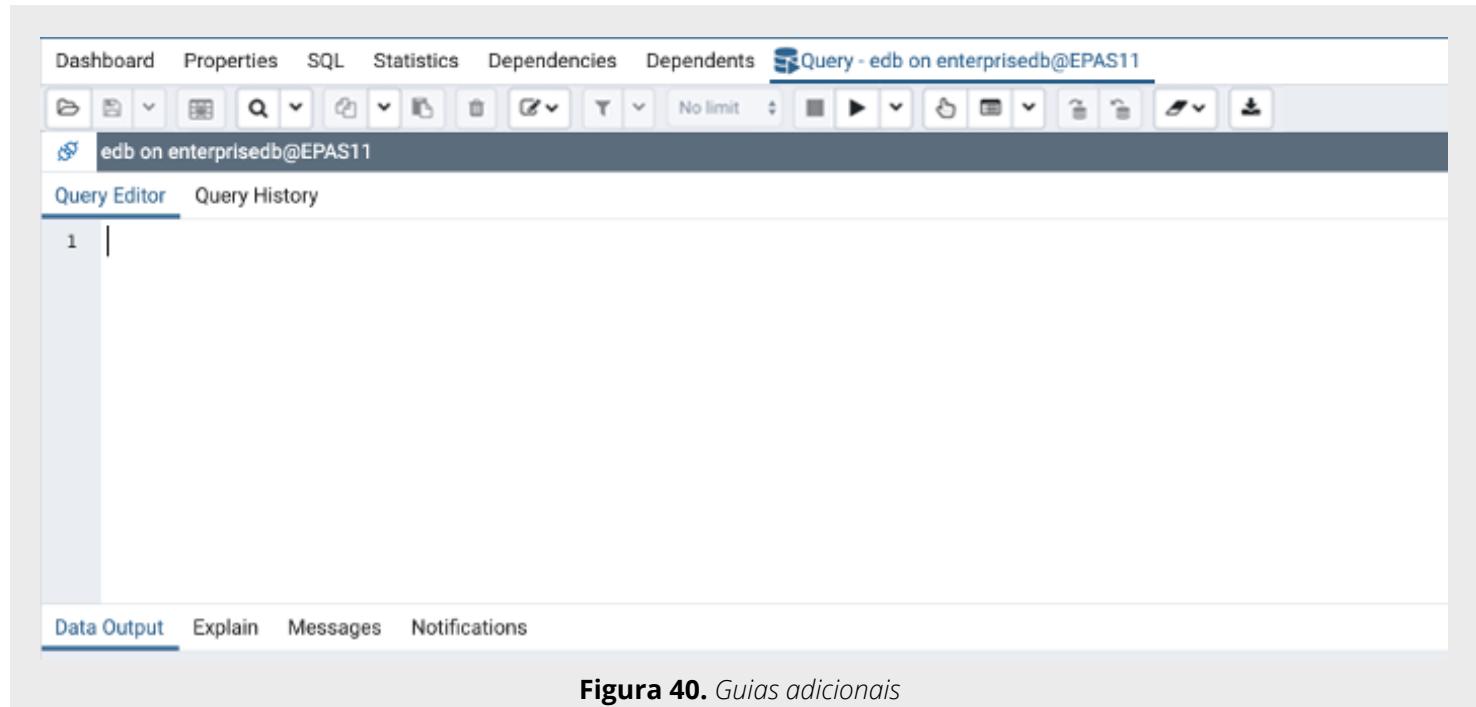


Figura 40. Guias adicionais

Guias adicionais são abertas quando você acessa a funcionalidade estendida oferecida pelas ferramentas pgAdmin (como a ferramenta de consulta, o depurador ou o editor SQL). Use o ícone fechar (X) localizado no canto superior direito de cada guia para fechá-la quando terminar de usar a ferramenta. Assim como as guias permanentes, essas guias podem ser reposicionadas na janela do cliente pgAdmin.

Por padrão, cada vez que você abre uma ferramenta, o pgAdmin abre uma nova aba do navegador. Você pode controlar esse comportamento modificando o nó *Exibir* da caixa de diálogo *Preferências para cada ferramenta*. Para abrir a caixa de diálogo *Preferências*, selecione *Preferências* no menu *Arquivo*.

• CONTROLE DE ÁRVORE

O painel esquerdo da janela principal exibe um controle de árvore (o controle de árvore pgAdmin) que fornece acesso aos objetos que residem em um servidor.

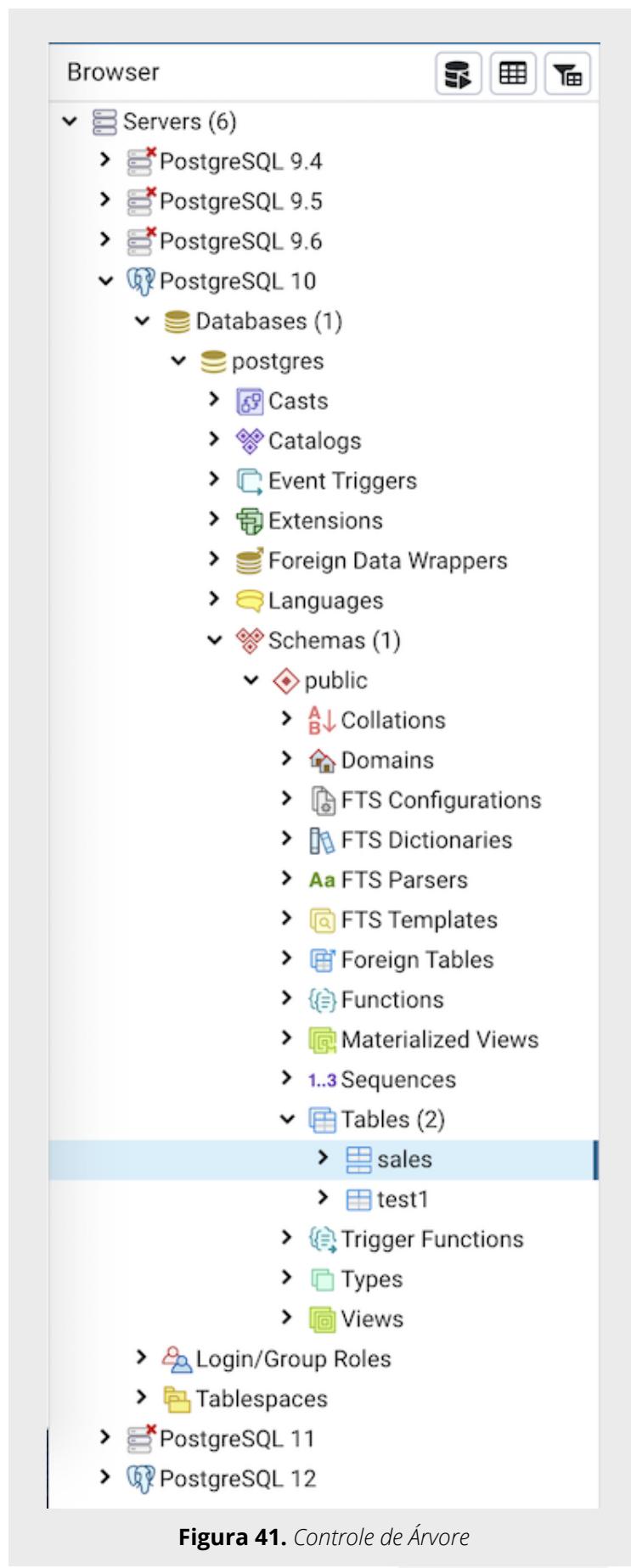


Figura 41. Controle de Árvore

Você pode expandir os nós no controle de árvore para exibir os objetos de banco de dados que residem em um servidor selecionado. O controle de árvore se expande para uma exibição hierárquica:

- Use o sinal de mais (+) à esquerda de um nó para expandir um segmento do controle de árvore;
- Clique no sinal de menos (-) à esquerda de um nó para fechá-lo.

Você também pode **arrastar e soltar** determinados objetos na Ferramenta de consulta, o que pode economizar tempo na digitação de nomes longos de objetos. O texto que contém o nome do objeto será totalmente qualificado com esquema. Aspas duplas serão adicionadas se necessário. Para funções e procedimentos, o nome da função junto com os nomes dos parâmetros será colado na Ferramenta de consulta.

Acesse menus sensíveis ao contexto clicando com o botão direito do mouse em um nó do controle de árvore para executar tarefas comuns. Os menus exibem opções que incluem uma ou mais das seguintes seleções (as opções aparecem em ordem alfabética):

Opção	Ação
Adicionar ponto de restauração nomeado	Clique para criar e inserir o nome de um ponto de restauração.
Cópia de segurança...	Clique para abrir a caixa de diálogo <i>Backup...</i> para fazer backup dos objetos do banco de dados.
Backup globais...	Clique para abrir a caixa de diálogo <i>Backup Globals...</i> para fazer backup de objetos de cluster.
Conectar servidor...	Clique para abrir a caixa de diálogo <i>Conectar ao Servidor</i> para estabelecer uma conexão com um servidor.
Crio	Clique para acessar um menu de contexto que fornece seleções sensíveis ao contexto. Sua seleção abre uma caixa de diálogo <i>Criar</i> para criar um novo objeto.
CRIAR Script	Clique para abrir a <i>Ferramenta de consulta</i> para editar ou visualizar o script CREATE.
Depuração	Clique para abrir a ferramenta <i>Debug</i> ou para selecionar <i>Set breakpoint</i> para parar ou pausar a execução de um script.

Opção	Ação
Excluir/Remover	Clique para excluir o objeto atualmente selecionado do servidor.
Desconectar banco de dados...	Clique para encerrar uma conexão de banco de dados.
Soltar Cascata	Clique para excluir o objeto selecionado no momento e todos os objetos dependentes do servidor.
Depuração	Clique para acessar a ferramenta <i>Depurador</i> .
Assistente de concessão	Clique para acessar a ferramenta <i>Assistente de concessão</i>
Manutenção...	Clique para abrir a caixa de diálogo <i>Manutenção...</i> para VACUUM, ANALYZE, REINDEX ou CLUSTER.
Propriedades...	Clique para revisar ou modificar as propriedades do objeto selecionado no momento.
Atualizar...	Clique para atualizar o objeto atualmente selecionado.
Recarregar configuração...	Clique para atualizar os arquivos de configuração sem reiniciar o servidor.
Restaurar...	Clique para acessar a caixa de diálogo <i>Restaurar</i> para restaurar arquivos de banco de dados de um backup.
Ver dados	Use a opção <i>Exibir Dados</i> para acessar os dados armazenados em uma tabela selecionada com a guia <i>Saída de Dados</i> da <i>Ferramenta de Consulta</i> .

Os menus sensíveis ao contexto associados a *Tabelas* e nós de *Tabela* aninhados fornecem opções de exibição adicionais (as opções aparecem em ordem alfabética):

Opção	Ação
Importar/Exportar dados...	Clique para abrir a caixa de diálogo <i>Importar/Exportar...</i> para importar ou exportar dados da tabela selecionada.
Limpar estatísticas	Clique para redefinir as estatísticas da tabela selecionada.
Scripts	Clique para abrir a ferramenta de consulta para editar ou visualizar o script selecionado no menu suspenso.
Truncar	Clique para remover todas as linhas de uma tabela.
Truncar Cascata	Clique para remover todas as linhas de uma tabela e suas tabelas filhas.
Ver as primeiras 100 linhas	Clique para acessar uma grade de dados que exibe as primeiras 100 linhas da tabela selecionada.
Ver últimas 100 linhas	Clique para acessar uma grade de dados que exibe as últimas 100 linhas da tabela selecionada.
Ver todas as linhas	Clique para acessar uma grade de dados que exibe todas as linhas da tabela selecionada.
Ver linhas filtradas...	Clique para acessar o pop-up <i>Filtro de Dados</i> para aplicar um filtro a um conjunto de dados.

- **ATALHOS DE TECLADO** ☺

Atalhos de teclado são fornecidos no pgAdmin para permitir acesso fácil a funções específicas. Atalhos alternativos podem ser configurados em Arquivo > Preferências, se desejado.

- **Janela principal do navegador** ☺

Ao usar a janela principal do navegador, os seguintes atalhos de teclado estão disponíveis:

Atalho para todas as plataformas	Função
Alt+Shift+F	Abra o menu Arquivo
Alt+Shift+O	Abra o menu Objeto
Alt+Shift+L	Abra o menu Ferramentas
Alt+Shift+H	Abra o menu Ajuda
Alt+Shift+B	Concentre a árvore do navegador
Alt+Shift+[Mover o painel com guias para trás
Alt+Shift+]	Mover o painel com guias para frente
Alt+Shift+Q	Abra a Ferramenta de Consulta no banco de dados atual
Alt+Shift+V	Exibir dados na tabela/exibição selecionada
Alt+Shift+C	Abra o menu de contexto
Alt+Shift+N	Criar um objeto
Alt+Shift+E	Editar propriedades do objeto
Alt+Shift+D	Excluir o objeto
Alt+Shift+G	Depuração direta

- **Abas de Diálogo** 

Use os atalhos abaixo para navegar pelas guias nas caixas de diálogo:

Atalho para todas as plataformas	Função
Control+Shift+[Guia de diálogo para trás
Control+Shift+]	Guia de diálogo para frente

- **Controles de Grade de Propriedades** 

Use os atalhos abaixo ao trabalhar com controles de grade de propriedades:

Atalho para todas as plataformas	Função
Control+Shift+A	Adicionar linha na grade
Aba	Mover o foco para o próximo controle
Shift+Tab	Mover o foco para o controle anterior
Retornar	Escolha o item selecionado em uma caixa de combinação

- **Editores SQL** ☺

Ao usar os editores SQL de realce de sintaxe, os seguintes atalhos estão disponíveis:

Atalho (Windows/Linux)	Atalho (MAC)	Função
Alt + Esquerda	Opção + Esquerda	Mover para o início da linha
Alt + Direita	Opção + Direito	Mover para o fim da linha
Ctrl + Alt + Esquerda	Cmd + Opção + Esquerda	Mover uma palavra para a esquerda
Ctrl + Alt + Direita	Cmd + Opção + Direita	Mover uma palavra para a direita
Ctrl+ /	Cmd + /	Comente o código selecionado (Inline)
Ctrl+ .	Cmd+ .	Descomente o código selecionado (Inline)
Ctrl + Shift + /	Cmd + Shift + /	Código de comentário/descomentário (Bloquear)
Ctrl + um	Cmd + a	Selecionar tudo
Ctrl + c	Cmd + c	Copiar o texto selecionado para a área de transferência
Ctrl + r	Cmd + r	Refazer a última edição desfeita
Ctrl + v	Cmd + v	Colar texto da área de transferência
Ctrl + z	Cmd + z	Desfazer última edição
Aba	Aba	Recuar texto selecionado

Atalho (Windows/Linux)	Atalho (MAC)	Função
Shift + Tab	Shift + Tab	Remover recuo do texto selecionado
Alt + g	Opção + g	Saltar (para linha:coluna)
Ctrl + Espaço	Ctrl + Espaço	Autocompletar
Ctrl + f	Cmd + f	Achar
Ctrl + g	Cmd + g	Encontre o próximo
Ctrl + Shift + g	Cmd + Shift + g	Encontrar anterior
Ctrl + Shift + f	Cmd + Shift + f	Substituir

- **Ferramenta de consulta** ↴

Ao usar a Ferramenta de consulta, os seguintes atalhos estão disponíveis:

Atalho (Windows/Linux)	Atalho (MAC)	Função
F5	F5	Executar consulta
F6	F6	Salvar alterações de dados
F7	F7	EXPLICAR consulta
Shift + F7	Shift + F7	ANALISAR consulta
F8	F8	Executar consulta ao arquivo CSV
<chave de acesso> + o	<chave de acesso> + o	Abrir arquivo
<chave de acesso> + s	<chave de acesso> + s	<chave de acesso> + s
<chave de acesso> + n	<chave de acesso> + n	Localizar opção suspensa

Atalho (Windows/Linux)	Atalho (MAC)	Função
<chave de acesso> + c	<chave de acesso> + c	Copiar linha(s)
<chave de acesso> + p	<chave de acesso> + p	Colar linha(s)
<chave de acesso> + d	<chave de acesso> + d	Excluir linha(s)
<chave de acesso> + f	<chave de acesso> + f	Caixa de diálogo Filtrar
<chave de acesso> + i	<chave de acesso> + i	Opções de filtro suspensas
<chave de acesso> + r	<chave de acesso> + r	Limite de linha
<chave de acesso> + q	<chave de acesso> + q	Cancelar consulta
<chave de acesso> + l	<chave de acesso> + l	Limpar menu suspenso
<chave de acesso> + x	<chave de acesso> + x	Executar opção suspensa
<chave de acesso> + t	<chave de acesso> + t	Exibir status de conexão
<chave de acesso> + y	<chave de acesso> + y	Copiar SQL no painel de histórico

- **Depurador** 

Ao usar o Depurador, os seguintes atalhos estão disponíveis:

Atalho (Windows/Linux)	Atalho (MAC)	Função
<chave de acesso> + i	<chave de acesso> + i	Entrar
<chave de acesso> + o	<chave de acesso> + o	Passar por cima
<chave de acesso> + c	<chave de acesso> + c	Continuar/Reiniciar
<chave de acesso> + t	<chave de acesso> + t	Alternar ponto de interrupção
<chave de acesso> + x	<chave de acesso> + x	Limpar todos os pontos de interrupção
<chave de acesso> + s	<chave de acesso> + s	Pare
Alt + Shift + q	Opção + Shift + q	Insira ou edite valores na grade

- **Navegação por Aba Interna e Painel** 

Ao usar a Ferramenta de consulta e o Depurador, os seguintes atalhos estão disponíveis para navegação no painel interno:

Atalho (Windows/Linux)	Atalho (MAC)	Função
Alt + Shift +]	Alt + Shift +]	Mover para a próxima guia em um painel
Alt + Shift + [Alt + Shift + [Mover para a guia anterior em um painel
Alt + Shift + Tab	Alt + Shift + Tab	Mover entre os painéis internos

- **Chave de Acesso** 

<accesskey> depende do navegador e da plataforma. A tabela a seguir lista as chaves de acesso padrão para navegadores compatíveis.

Atalho (Windows/Linux)	Atalho (MAC)	Função	
Internet Explorer	Alt	Alt	
Safari	Alt		Ctrl + Opção

6. Introdução ao PostgreSQL DDL

O PostgreSQL DDL é o acrônimo para Data Definition Language. O Idioma de Definição de Dados é usado para lidar com as descrições e esquemas do banco de dados, e é usado para definir e modificar a estrutura dos dados. Com a ajuda do Idioma de Definição de Dados, decidimos como os dados devem ser armazenados no banco de dados. Podemos realizar operações no banco de dados criando um novo banco de dados, alterando o banco de dados existente e removendo, truncando ou renomeando o banco de dados existente.

Como funciona a Declaração DDL no PostgreSQL?

O Idioma de Definição de Dados é usado para definir, bem como modificar, a estrutura dos dados. O que significa que os comandos do Idioma de definição de dados podem ser usados para criar, soltar ou modificar tabelas dentro de um mesmo banco de dados.

Lista de declaração DDL. O Idioma de Definição de Dados consiste nas seguintes declarações:

- CREATE;
- ALTER;
- TRUNCATE;
- DROP.

6.1. CREATE

O CREATE é usado para criar objetos (*database, table, index, views, store procedure, function, e triggers*).

Sintaxe:

```
CREATE TABLE <table_name>
(column_name_1 datatype,
column_name_2 datatype,
.
.
column_name_n datatype
);
```

Exemplo:

```
CREATE TABLE student
(
student_id INT,
student_name VARCHAR(30),
student_age INT,
student_address VARCHAR(25),
student_phone_no VARCHAR(20)
);
```

6.2. ALTER

A instrução ALTER permite modificar os objetos de banco de dados existentes. Podemos alterar ou modificar a estrutura do banco de dados usando uma instrução ALTER.

Podemos executar as seguintes operações usando uma instrução ALTER:

1. Adicione uma coluna na tabela;
2. Drop a coluna existente;
3. Altere o tipo de dados da coluna.

a. Considere a seguinte sintaxe de instrução ALTER para adicionar uma PRIMARY KEY em uma coluna de uma tabela existente.

Sintaxe:

```
ALTER TABLE
<tableName>
ADD PRIMARY KEY (<columnName>);
```

Exemplo:

```
ALTER TABLE
student
ADD PRIMARY KEY (student_id);
```

b. Considere a seguinte sintaxe de instrução ALTER para definir NÃO NULL em uma coluna na tabela existente.

Sintaxe:

```
ALTER TABLE
<tableName>
ALTER COLUMN
<columnName>
SET NOT NULL;
```

Exemplo:

```
ALTER TABLE
student
ALTER COLUMN
student_name
SET NOT NULL;
```

c. Considere a seguinte sintaxe de declaração ALTER para excluir NÃO NULL em uma coluna na tabela existente.

Sintaxe:

```
ALTER TABLE
<tableName>
ALTER COLUMN
<columnName>
DROP NOT NULL;
```

Exemplo:

```
ALTER TABLE
student
ALTER COLUMN
student_name
DROP NOT NULL;
```

d. Considere a seguinte sintaxe de instrução ALTER para adicionar uma nova coluna na tabela existente.

Sintaxe:

```
ALTER TABLE
<tableName>
ADD <columnName data-type>;
```

Exemplo:

```
ALTER TABLE
student
ADD email varchar(50);
```

Considere a seguinte sintaxe de instrução ALTER para renomear um nome de coluna existente para o novo nome da coluna.

Sintaxe:

```
ALTER TABLE
<tableName>
RENAME COLUMN
<oldColumnName> TO <newColumnName>;
```

Exemplo:

```
ALTER TABLE
student
RENAME COLUMN
Email TO email_id;
```

Considere a seguinte sintaxe de instrução ALTER para remover uma coluna existente da tabela.

Sintaxe:

```
ALTER TABLE
<tableName>
DROP COLUMN
<columnName>;
```

Exemplo:

```
ALTER TABLE
student
DROP COLUMN
Email_id;
```

Podemos renomear os objetos do banco de dados. A instrução RENAME é usada para renomear as tabelas no banco de dados.

Considere a seguinte sintaxe de declaração RENAME para renomear a coluna existente da tabela.

Sintaxe:

```
ALTER TABLE <oldName>
RENAME
TO
<newName>;
```

Exemplo:

```
ALTER TABLE
student
RENAME TO
studentInfo;
```

6.3. TRUNCATE

Podemos remover todas as linhas de uma tabela, incluindo todos os espaços alocados para as linhas.

Considere a seguinte sintaxe de instrução TRUNCATE para remover todas as linhas da tabela.

Sintaxe:

```
TRUNCATE TABLE  
<tableName>;
```

Exemplo:

```
TRUNCATE TABLE studentInfo;
```

6.4. DROP

Podemos excluir objetos de banco de dados usando a instrução DROP, como uma tabela, índice ou visualização etc.

Considere a seguinte sintaxe de instrução DROP para remover toda a estrutura do objeto do banco de dados.

Sintaxe:

```
DROP TABLE  
<tableName>;
```

OR

```
DROP DATABASE  
<databaseName>;
```

Exemplo:

```
DROP TABLE studentInfo;
```

7. Data Modification Language - DML

O aplicativo a se modelar é um aplicativo semelhante ao Twitter, por exemplo, no qual temos usuários e esses podem enviar mensagens curtas, seguir outros usuários e, em seguida, podem ver o que esses enviaram.

Começando com algo simples, tem-se:

```
(  
    userid  bigserial primary key,  
    uname   text not null,  
    nickname text,  
    bio     text,  
    picture  text,  
  
    unique(uname)  
);  
  
create table tweet.follower  
(  
    follower bigint not null references tweet.users(userid),  
    following bigint not null references tweet.users(userid),  
  
    primary key(follower, following)  
);  
  
create table tweet.message  
(  
    messageid bigserial primary key,  
    userid   bigint not null references tweet.users(userid),  
    datetimestamp timestamp not null default now(),  
    message   text not null,  
    favs      bigint,  
    rts       bigint,  
    location  point,  
    lang      text,  
    url       text  
);
```

- **Insert Into**

Dado o modelo anterior de tweets, o primeiro elemento de que se precisa são os usuários. Veja como criar os primeiros usuários:

```
insert into tweet.users (userid, uname, nickname, bio)
    values (default, 'Theseus', 'Duke Theseus', 'Duke of Athens.');
```

O comando insert aceita a inclusão de várias linhas na mesma instrução, conforme abaixo:

```
insert into tweet.users (uname, bio)
    values ('Egeus', 'father to #Hermia.'),
           ('Lysander', 'in love with #Hermia.'),
           ('Demetrius', 'in love with #Hermia.'),
           ('Philostrate', 'master of the revels to Theseus.'),
           ('Peter Quince', 'a carpenter.'),
           ('Snug', 'a joiner.'),
           ('Nick Bottom', 'a weaver.'),
           ('Francis Flute', 'a bellows-mender.'),
           ('Tom Snout', 'a tinker.'),
           ('Robin Starveling', 'a tailor.'),
           ('Hippolyta', 'queen of the Amazons, betrothed to Theseus.'),
           ('Hermia', 'daughter to Egeus, in love with Lysander.'),
           ('Helena', 'in love with Demetrius.'),
           ('Oberon', 'king of the fairies.'),
           ('Titania', 'queen of the fairies.'),
           ('Puck', 'or Robin Goodfellow.'),
           ('Peaseblossom', 'Team #Fairies'),
           ('Cobweb', 'Team #Fairies'),
           ('Moth', 'Team #Fairies'),
           ('Mustardseed', 'Team #Fairies'),
           ('All', 'Everyone speaking at the same time'),
           ('Fairy', 'One of them #Fairies'),
           ('Prologue', 'a play within a play'),
           ('Wall', 'a play within a play'),
           ('Pyramus', 'a play within a play'),
           ('Thisbe', 'a play within a play'),
           ('Lion', 'a play within a play'),
           ('Moonshine', 'a play within a play');
```

- **Insert Into ... Select**

A instrução de *inserção* também pode usar uma consulta como fonte de dados. Pode-se, por exemplo, preencher a tabela *tweet.follower* com pessoas que são conhecidas por se amarem de seu campo *bio*.

Primeiro, precisa-se tirar esses dados dos campos previamente inseridos, que é a fonte de dados nesse caso.

```
select users.userid as follower,
       users.uname,
       f.userid as following,
       f.uname
  from tweet.users
 join tweet.users f
    on f.uname = substring(users.bio from 'in love with #?(.*).')
 where users.bio ~ 'in love with';
```

A *substring* aqui retorna apenas o grupo de correspondência de expressão regular, que passa a ser o nome de quem o usuário criado anteriormente ama. A consulta, então, resulta no seguinte:

Follower	Uname	Following	Uname
3	Lysander	13	Hermia
4	Demetrius	13	Hermia
13	Hermia	3	Lysander
14	Helena	4	Demetrius

(4 rows)

Agora, tem-se que inserir o seguidor e seguir os dados na tabela *tweet.follower*, é claro. Como o inserção no comando sabe ler sua entrada a partir do resultado de uma instrução selecionada, é muito fácil de fazer. Veja:

```
insert into tweet.follower
  select users.userid as follower,
         f.userid as following
    from tweet.users
   join tweet.users f
      on f.uname = substring(users.bio from 'in love with #?(.*).')
 where users.bio ~ 'in love with';
```

Agora, sobre essas fadas seguindo sua rainha e rei:

```
with fairies as
(
  select userid
    from tweet.users
   where bio ~ '#Fairies'
)
insert into tweet.follower(follower, following)
  select fairies.userid as follower,
         users.userid as following
    from fairies cross join tweet.users
   where users.bio ~ 'of the fairies';
```

Desta vez, até se tem a oportunidade de usar um *cross join*, já que se quer produzir todas as diferentes combinações de uma *fada* com seus súditos reais.

Aqui está o que se tem configurado em termos de seguidores agora:

```
select follower.uname as follower,
       follower.bio as "follower's bio",
       following.uname as following

  from tweet.follower as follows

  join tweet.users as follower
    on follows.follower = follower.userid

  join tweet.users as following
    on follows.following = following.userid;
```

E aqui está o que já foi configurado:

Follower	Follower's Bio	Following
Hermia	daughter to Egeus, in love with Lysander.	Lysander
Helena	in love with Demetrius.	Demetrius
Demetrius	in love with #Hermia	Hermia

Follower	Follower's Bio	Following
Lysander	in love with #Hermia	Hermia
Peaseblossom	Team #Fairies	Oberon
Cobweb	Team #Fairies	Oberon
Moth	Team #Fairies	Oberon
Mustardseed	Team #Fairies	Oberon
Peaseblossom	Team #Fairies	Titania
Cobweb	Team #Fairies	Titania
Moth	Team #Fairies	Titania
Mustardseed	Team #Fairies	Titania

(12 rows)

- **Update**

A instrução *Update* é usada para substituir os valores existentes no banco de dados. Seu aspecto mais importante está em seu comportamento de concorrência, pois permite substituir os valores existentes enquanto outros usuários estão trabalhando simultaneamente com o banco de dados.

No PostgreSQL, todos os recursos de concorrência são baseados no MVCC, e no caso da declaração de *Update* significa que internamente o PostgreSQL está fazendo tanto uma *inserção* dos novos dados quanto uma *exclusão* dos antigos. As colunas do sistema PostgreSQL *xmin* e *xmax* permitem o rastreamento de visibilidade das linhas para que a instrução simultânea tenha um instantâneo consistente do conjunto de dados do servidor o tempo todo.

Como o bloqueio de linha é feito por tupla no PostgreSQL, uma declaração de *atualização* só bloqueia outra *atualização, exclui ou seleciona para instrução de atualização* que visa à(s) mesma(s) linha(s).

Anteriormente, foram criados alguns usuários sem um *apelido*, e talvez seja hora de remediar isso, atribuindo-lhes seu *nome* como apelido por enquanto.

```
begin;

  update tweet.users
    set nickname = 'Robin Goodfellow'
    where userid = 17 and uname = 'Puck'
  returning users.*;

commit;
```

Aqui escolhe-se o id 17 da tabela de uma forma manual. A ideia é mostrar como atualizar campos em uma única tupla de uma *chave primária*. Em muitos casos, o código deste aplicativo obteve o *id* anteriormente e o injetou na consulta de atualização.

E graças à cláusula *de retorno*, pode-se ver o que foi feito:

Userid	Uname	Nickname	Bio	Picture
17	Puck	RobinGoodfellow	or Robin Goodfellow	☒

(1 row)

Como você pode ver, na consulta anterior não só foi usado o *campo-chave principal*, mas também, como é uma chave, foi adicionado o valor real, que é o objeto de interesse. Se as informações tivessem sido coladas erroneamente, a *atualização* não encontraria linhas correspondentes e isso afetaria zero tuplas.

Agora há outro caso de uso para essa dupla verificação: a concorrência. Sabe-se que o apelido *robin goodfellow* se aplica a *Puck*. E se alguém atualizou o *nome do Puck* enquanto a declaração de atualização estava sendo executada? Com essa verificação dupla, sabe-se exatamente que um dos seguintes é verdadeiro:

- 1.Ou a outra declaração veio em primeiro lugar e o nome não é *Puck* e nenhuma linha foi atualizada.
- 2.A outra declaração virá mais tarde e apenas uma linha foi atualizada (que sabe-se que é user id 17, e chamado *Puck*).

Pense nesse truque ao lidar com a concorrência no código do seu aplicativo, e ainda mais quando você estiver corrigindo alguns dados do console para uma correção única. Em seguida, use sempre um bloco de transações explícito para que você possa verificar o que aconteceu e emitir uma *reversão*; quando não é o que você pensou.

Também se pode *atualizar* várias linhas ao mesmo tempo. Suponha-se que se quer adicionar um apelido padrão a todos esses personagens:

```
update tweet.users
  set nickname = case when uname ~ ' '
                      then substring(uname from '[^ ]* (.*)')
                      else uname
                    end
  where nickname is null
returning users.*;
```

E agora todos recebem um apelido adequado, computado a partir de seu nome de usuário com o truque fácil e prático que você pode ver na consulta. A principal informação a ser lembrada nessa consulta é que você pode usar os dados existentes em sua instrução *UPDATE*.

Agora, quem são os usuários do Twitter?

```
select uname, nickname, bio
  from tweet.users
order by userid;
```

Uname	Nickname	Bio
Theseus	Duke Theseus	Duke of Athens.
Egeus	Egeus	father to #Hermia.
Lysander	Lysander	in love with #Hermia.
Demetrius	Demetrius	in love with #Hermia.
Philostrate	Philostrate	master of the revels to Theseus.
Peter Quince	Quince	a carpenter.
Snug	Snug	a joiner.
Nick Bottom	Bottom	a weaver.
Francis Flute	Flute	a bellows-mender.
Tom Snout	Snout	a tinker.

Uname	Nickname	Bio
Robin Starveling	Starveling	a tailor.
Hippolyta	Hippolyta	queen of the Amazons, betrothed to Theseus.
Hermia	Hermia	daughter to Egeus, in love with Lysander.
Helena	Helena	in love with Demetrius.
Oberon	Oberon	king of the fairies.
Titania	Titania	queen of the fairies.
Puck	Robin Goodfellow	or Robin Goodfellow.
Peaseblossom	Peaseblossom	Team #Fairies
Cobweb	Cobweb	Team #Fairies
Moth	Moth	Team #Fairies
Mustardseed	Mustardseed	Team #Fairies
All	All	Everyone speaking at the same time
Fairy	Fairy	One of them #Fairies
Prologue	Prologue	a play within a play
Wall	Wall	a play within a play
Pyramus	Pyramus	a play within a play
Thisbe	Thisbe	a play within a play

Uname	Nickname	Bio
Lion	Lion	a play within a play
Moonshine	Moonshine	a play within a play

(29 rows)

Inserindo alguns Tweets

```

<PLAYSUBT>A MIDSUMMER NIGHT'S DREAM</PLAYSUBT>

<ACT><TITLE>ACT I</TITLE>

<SCENE><TITLE>SCENE I. Athens. The palace of THESEUS.</TITLE>
<STAGEDIR>Enter THESEUS, HIPPOLYTA, PHILOSTRATE, and
Attendants</STAGEDIR>

<SPEECH>
<SPEAKER>THESEUS</SPEAKER>
<LINE>Now, fair Hippolyta, our nuptial hour</LINE>
<LINE>Draws on apace; four happy days bring in</LINE>
<LINE>Another moon: but, O, methinks, how slow</LINE>
<LINE>This old moon wanes! she lingers my desires,</LINE>
<LINE>Like to a step-dame or a dowager</LINE>
<LINE>Long withering out a young man revenue.</LINE>
</SPEECH>

<SPEECH>
<SPEAKER>HIPPOLYTA</SPEAKER>
<LINE>Four days will quickly steep themselves in night;</LINE>
<LINE>Four nights will quickly dream away the time;</LINE>
<LINE>And then the moon, like to a silver bow</LINE>
<LINE>New-bent in heaven, shall behold the night</LINE>
<LINE>Of our solemnities.</LINE>
</SPEECH>
```

Para que os personagens da reprodução twitem suas linhas, é necessário escrever um simples analisador XML para o formato e usar o comando de inserção SQL. Extraído do código usado para inserir os dados, aqui está a consulta de inserção:

```
insert into tweet.message(userid, message)
  select userid, $2
    from tweet.users
   where users.uname = $1 or users.nickname = $1
```

Como o texto da peça usa nomes como <SPEAKER>QUINCE</SPEAKER> e foram inseridos nomes reais no banco de dados, deve-se combinar o conteúdo XML do jogo com o *uname* ou o campo de *apelidos*.

Agora que os dados estão carregados, pode-se dar uma olhada no início da peça em SQL.

```
select uname, message
  from tweet.message
    left join tweet.users using(userid)
order by messageid limit 4;
```

E sim, agora pode-se ver, por exemplo, Shakespeare tuitando:

Uname	Message
Theseus	<ul style="list-style-type: none"> Now, fair Hippolyta, our nuptial hour ← Draws on apace; four happy days bring in ← Another moon: but, O, methinks, how slow ← This old moon wanes! she lingers my desires, ← Like to a step-dame or a dowager ← Long withering out a young man revenue. ←
Hippolyta	<ul style="list-style-type: none"> Four days will quickly steep themselves in night; ← Four nights will quickly dream away the time; ← And then the moon, like to a silver bow ← New-bent in heaven, shall behold the night ← Of our solemnities. ←

Uname	Message
Theseus	<pre> Go, Philostrate, Stir up the Athenian youth to merriments; Awake the pert and nimble spirit of mirth; Turn melancholy forth to funerals; The pale companion is not for our pomp. Hippolyta, I woo'd thee with my sword, And won thy love, doing thee injuries; But I will wed thee in another key, With pomp, with triumph and with revelling. </pre>
Egeus	<pre> Happy be Theseus, our renowned duke!</pre>

- **Delete**

A instrução de exclusão permite marcar tuplas para remoção. Dada a implementação do MVCC pelo PostgreSQL, não seria sábio remover a tupla do disco no momento da exclusão:

1. Primeiro, a transação pode *reverter* e ainda não se sabe;
2. Em segundo lugar, outras transações simultâneas só podem ver a exclusão após o *commit*, não assim que a declaração for feita.

Assim como na declaração de atualização, a parte mais importante da declaração *de exclusão* tem a ver com a concorrência. Mais uma vez, a principal razão pela qual se usa um RDBMS é para que não se tenha que resolver os problemas de concorrência no código do aplicativo feito, em vez disso, pode-se concentrar-se em oferecer uma melhor experiência ao usuário.

Diga-se que, por engano, foram adicionados personagens de outra peça, e não se quer ter que lidar com eles. Primeiro, inserindo-os:

```

insert into tweet.users (uname, bio)
values ('CLAUDIUS', 'king of Denmark.'),
       ('HAMLET', 'son to the late, and nephew to the present king'),
       ('POLONIUS', 'lord chamberlain.'),
       ('HORATIO', 'friend to Hamlet'),
       ('LAERTES', 'son to Polonius'),
       ('LUCIANUS', 'nephew to the king');

```

A sintaxe de exclusão é bastante simples:

```
begin;

delete
  from tweet.users
  where userid = 22 and uname = 'CLAUDIUS'
returning *;

commit;
```

E como de costume, graças à cláusula de retorno, sabe-se exatamente o que acabou-se de marcar para a exclusão:

Userid	Uname	Nickname	Bio	Picture
22	Claudius	¤	king of Denmark.	¤

(1 row)

Agora também se pode *excluir* mais de uma linha com o mesmo comando - tudo depende do que foi combinado. Como os novos caracteres inseridos por engano não tinham um papel na peça de onde foram inseridas as mensagens criadas, então pode-se usar uma *anti-join* para excluí-las com base nessas informações:

```
begin;

with deleted_rows as
(  

  delete
    from tweet.users
    where not exists
      (
        select 1
          from tweet.message
          where userid = users.userid
      )
  returning *
)
select min(userid), max(userid),
  count(*),
  array_agg(uname)
from deleted_rows;

commit;
```

E, como esperado, tem-se um bom resumo do que foi feito. Essa deve ser agora a sua sintaxe padrão para qualquer exclusão que você pretenda executar interativamente em qualquer banco de dados, certo?

Min	Max	Count	Array_Agg
41	45	5	{HAMLET, POLONIUS, HORATIO, LAERTES, LUCIANUS}

(1 row)

- **Excluindo todas as linhas: Truncate**

PostgreSQL adiciona às instruções *DML* o comando *truncate*. Internamente, é considerado um *DDL* em vez de um *DML*. É uma maneira muito eficiente de limpar uma tabela de todo o seu conteúdo de uma só vez, pois não segue o sistema MVCC por tupla e simplesmente removerá os arquivos de dados em disco.

Observe que o comando *truncate* ainda está em conformidade com o MVCC:

```
select count(*) from foo;

begin;
truncate foo;
rollback;

select count(*) from foo;
```

Supondo que não haja atividade simultânea em seu sistema ao executar os comandos, ambas as consultas de contagem retornam naturalmente ao mesmo número.

- **Excluir, mas manter algumas linhas**

Ao limpar um conjunto de dados, pode acontecer que você queira remover a maior parte do conteúdo de uma tabela. Pode ser uma tabela de registros, uma trilha de auditoria que expirou ou algo assim. Como foi visto anteriormente ao usar PostgreSQL, marque as tuplas como não sendo mais visíveis e, em seguida, o *Vacuum* faz o levantamento pesado no fundo. É, então, mais eficiente criar uma tabela contendo apenas as novas linhas e trocá-la com a tabela antiga:

```
begin;

create table new_name (like name including all);

insert into new_name
    select <column list>
        from name
    where <restrictions>;

drop table name;
alter table new_name rename to name;

commit;
```

No caso geral, assim que você remover a *maioria* das entradas da sua tabela, esse método será mais eficiente. O problema com ele é o nível de bloqueio necessário para executar a *tabela de queda* e as declarações de *tabela de alteração*.

Esses *DDL* requerem um *bloqueio exclusivo de acesso* e bloquearão qualquer tráfego de leitura e gravação para ambas as tabelas enquanto elas são executadas. Se você não tem horas lentas ou mesmo fora do horário de expediente, então pode não ser viável para você usar este truque.

A parte boa sobre *excluir* e *aspirar* é que essas funções podem correr no meio de qualquer tráfego simultâneo, é claro.

8. Referências Bibliográficas

ALVES, William Pereira. **Banco de Dados [BV:MB]**. 1^a ed. São Paulo: Érica, 2014.

BALIEIRO, R. **Banco de Dados [BV:RE]**. 1^a ed. Rio de Janeiro: SESES, 2015.

ELMASRI, R.; NAVATHE, Shamkant B. **S. Sistemas de Banco de Dados [BV:PE]**. 7. ed. São Paulo: Pearson, 2018.

FONSECA, Cleber Costa da. **Implementação de banco de dados. Banco de Dados [BV:RE]**. 1^a ed. Rio de Janeiro: SESES, 2016.

HEUSER, C. **Projeto de Banco de Dados [BV:MB]**. 6. ed. Porto Alegre: ARTMED, 2009.

MACHADO, Felipe N. R. **Banco de Dados - Projeto e Implementação [BV:MB]**. 3. ed. São Paulo: Érica, 2014.

NETO, Geraldo H. **MODELAGEM DE DADOS [BV:RE]**. 1^a ed. Rio de Janeiro: SESES, 2015.

PUGA, Sandra; FRANÇA, Edson; GOYA, Milton. **Banco de Dados: implementação em SQL, PL/SQL e Oracle 11g [BV:PE]**. 1^a ed. São Paulo: Pearson, 2013.

RAMAKRISHNAN, R. **Sistemas de gerenciamento de banco de dados [BE:MB]**. 3. ed. Porto Alegre: McGraw-Hill, 2008.



Digital College

ENSINO DE HABILIDADES DIGITAIS

digitalcollege.com.br