

UNIVERSIDADE ESTADUAL DO OESTE DO PARANÁ – UNIOESTE
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS – CAMPUS CASCAVEL
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA AGRÍCOLA

APOSTILA DE PGADMIN E POSTGRESQL – LABORATÓRIO DE TOPOGRAFIA
E GEOPROCESSAMENTO (GEOLAB) E LABORATÓRIO DE MECANIZAÇÃO E
AGRÍCULTURA DE PRECISÃO (LAMAP)

GIUVANE CONTI

CASCAVEL

2019

LISTA DE FIGURAS

Figura 1. Teoria de conjuntos.	9
Figura 2. Teoria de conjuntos.	9
Figura 3. Representação de bancos de dados relacionais.	9
Figura 4. Exemplo de consulta SQL.....	10
Figura 5. Instalação PostgreSQL no Ubuntu.....	10
Figura 6. Instalação de pacotes complementares no Ubuntu.....	10
Figura 7. Alterando arquivo de configuração do PostgreSQL server no Ubuntu.....	11
Figura 8. Habilitando acesso remoto ao servidor PostgreSQL no Ubuntu.....	11
Figura 9. Download do instalador do PostgreSQL para Windows.	12
Figura 10. Versões de instalação do PostgreSQL para Windows.	12
Figura 11. Criando um servidor no PostgreSQL.....	13
Figura 12. Criando um banco de dados de manutenção no servidor PostgreSQL criado.....	14
Figura 13. Criando um banco de dados para o usuário.....	14
Figura 14. Definindo atributos do novo banco de dados criado.	15
Figura 15. Estrutura do banco de dados criado.....	16
Figura 16. Ferramenta 'Query Tool' para execução de comandos SQL.	17
Figura 17. Barra de ferramentas do Query Tool.....	18
Figura 18. Executando um comando SQL no PgAdmin.	19
Figura 19. Criando um novo usuário.	19
Figura 20. Definindo papéis de um novo usuário.	20
Figura 21. Consulta SQL que exibe informações disponíveis em arquivos de configuração do PostgreSQL.	22
Figura 22. Consulta SQL que exibe o local de armazenamento de arquivos de configuração do PostgreSQL.	22
Figura 23. Script de criação das tabelas cidade e clima.....	25
Figura 24. Exclusão da tabela clima.	27
Figura 25. Inserindo linhas (registros) na tabela cidade.	27
Figura 26. Inserindo linhas (registros) na tabela clima.	28
Figura 27. Consulta SQL básica na tabela clima.....	29
Figura 28. Consulta SQL com expressão aritmética.	29
Figura 29. Consulta SQL com restrições de linhas.	30
Figura 30. Consulta SQL com junção de tabelas com WHERE.	31
Figura 31. Consulta SQL com junção de tabelas com JOIN.....	31
Figura 32. Consulta SQL com função de agregação.....	32

Figura 33. Consulta SQL com função de agregação 'Max' e Subconsulta.	32
Figura 34. Consulta SQL com função de agregação e clausula GROUP BY.	33
Figura 35. Consulta SQL com função de agregação, clausula GROUP BY e restrição de agrupamento HAVING.	34
Figura 36. Atualizando registros de uma tabela.	34
Figura 37. Excluindo registros de uma tabela.	35
Figura 38. Criação de uma View (Visão).	35
Figura 39. Consulta SQL em uma View (Visão).	36
Figura 40. Sistema Universal Transverso de Mercator (UTM).	38
Figura 41. Requisição de arquivos espaciais via internet (WFS).	39
Figura 42. Requisição de mapas via internet (WMS).	40
Figura 43. Requisição de um catalogo de arquivos espaciais via internet (WCS).	40
Figura 44. Topologia em consultas espaciais.	41
Figura 45. Exemplo de consulta espacial.	41
Figura 46. Banco de dados com extensão espacial habilitada.	42
Figura 47. Tipos de dados espaciais disponibilizados pelo PostgreSQL/PostGIS.	43
Figura 48. Conexão na ferramenta PostGIS Shapefile Import/Export Manager.	44
Figura 49. Importando shapefile para banco de dados PostgreSQL/PostGIS.	45
Figura 50. Importação do shapefile para o banco de dados realizado com sucesso. ..	46
Figura 51. tabela 'paradas' com dados do shapefile já no banco de dados.	46
Figura 52. Exemplo de planilha para ser importada para o banco de dados PostgreSQL/PostGIS.	47
Figura 53. Opção para criação de novo script.	48
Figura 54. Tabela horários, utilizada na migração da planilha.	48
Figura 55. Opções para realizar a migração da planilha	49
Figura 56. Consulta SQL para testar migração.	49
Figura 57. Consulta SQL espacial utilizando a função ST_Within	50
Figura 58. Consulta SQL espacial utilizando a função ST_DWithin	51

LISTA DE QUADROS

Quadro 1. Tipos de dados numéricos.	23
Quadro 2. Tipos de dados Caractere.	24
Quadro 3. Tipos de dado data e hora.	24

SUMÁRIO

1. INTRODUÇÃO.....	7
2. MANIPULANDO O <i>POSTGRES</i> QL E O <i>PGADMIN</i>	8
2.1 ARQUITETURA.....	8
2.2 <i>STRUCTURE QUERY LANGUAGE (SQL)</i>	8
2.3 INSTALAÇÃO E CONFIGURAÇÃO.....	10
2.3.1 Linux.....	10
2.3.2 Windows.....	11
2.4 CONCEITOS	12
2.5 UTILIZANDO O <i>PgAdmin</i>	12
2.5.1 Criando um banco de dados.....	13
2.5.2 Executando comando <i>SQL</i>	16
2.6 CONFIGURAÇÃO DE USUÁRIOS E DE CONEXÕES.....	19
2.7 CRIAÇÃO DE UM BANCO DE DADOS POR LINHA DE COMANDO.....	21
2.8 CRIAÇÃO DE UM SCHEMA POR LINHA DE COMANDO	21
2.9 ARQUIVOS DE CONFIGURAÇÕES.....	21
3. TIPOS DE DADOS	23
3.1 TIPO NUMÉRICO.....	23
3.2 TIPO CARACTERE	23
3.3 TIPO DATA E HORA	24
3.4 TIPO BOOLEANO	24
4. MANIPULAÇÃO DE TABELAS.....	25
4.1 CRIANDO ESTRUTURAS DE TABELAS	25
4.2 ALTERANDO ESTRUTURAS DE TABELAS.....	26
4.3 EXCLUINDO TABELAS.....	27
4.4 INSERÇÃO DE LINHAS EM TABELAS	27
4.5 CONSULTANDO DADOS DE TABELAS.....	28
4.6 JUNÇÕES ENTRE TABELAS.....	30
4.7 FUNÇÕES DE AGREGAÇÃO	32
4.8 ATUALIZAÇÕES E EXCLUSÕES DE LINHAS.....	34
4.9 VISÕES.....	35
5. <i>POSTGIS</i>	36
5.1 CRIANDO UM AMBIENTE DE ANÁLISE ESPACIAL	36
5.2 DADO ESPACIAL.....	37

5.3	FORMATO RASTER X VETORIAL	37
5.4	SISTEMA DE COORDENADAS	37
5.5	ARQUIVO DE DADOS ESPACIAIS	38
5.6	CONCEITOS TOPOLÓGICOS	40
5.7	CONSULTAS ESPACIAIS	41
5.8	CRIANDO BASES DE DADOS ESPACIAIS	42
5.9	IMPORTANDO DADOS ESPACIAIS PARA O <i>POSTGRESQL/POSTGIS</i>	43
5.10	IMPORTANDO PLANILHAS PARA O <i>POSTGRESQL/POSTGIS</i>	46
5.11	<i>QUERIES</i> (CONSULTAS) ESPACIAIS NO <i>POSTGRESQL/POSTGIS</i>	49
6.	REFERÊNCIAS	52

1. INTRODUÇÃO

O *PostgreSQL* é um Sistema Gerenciador de Banco de Dados (SGBD) objeto-relacional. Foi desenvolvido pelo Departamento de Ciência da Computação da Universidade da Califórnia em Berkeley (POSTGRES, 2019).

O *PostgreSQL* segue o padrão da indústria para linguagens e consulta, SQL:2003. Por ser uma aplicação de código-fonte aberto uma equipe de desenvolvedores da Internet mantém o *PostgreSQL*. Assim, os usuários têm acesso ao código-fonte e contribuem com correções, aprimoramentos e sugestões para novos recursos. (STONES e MATTHEW, 2006).

Disponibiliza funcionalidades como: comandos complexos; chaves estrangeiras; gatilhos; visões; integridade transacional; controle de simultaneidade multiversão. Também possui suporte para: tipos de dados; funções; operadores; funções e agregação; métodos de índice e; linguagens procedurais (POSTGRES, 2019).

SQL (Structure Query Language) é uma linguagem declarativa, onde, através de uma sintaxe é possível dizer ao computador o que se deseja e a máquina decide a forma correta de chegar ao resultado. (POSTGRES, 2019).

Esta apostila tem o objetivo de fornecer uma introdução ao *PostgreSQL/PostGIS*, à alguns conceitos de bancos de dados, e à linguagem *SQL*. Os exemplos práticos utilizados nessa disciplina foram baseados no material oficial do *PostgreSQL* e no curso de (Ferreira, 2019).

Todos os arquivos com scripts e mapas estão disponíveis em um repositório aberto via GitHub¹.

¹ <https://github.com/giuvane/apostila-pgeagri>

2. MANIPULANDO O *POSTGRES*SQL E O *PGADMIN*

O *PgAdmin* é um software (IDE) utilizada para administração do Sistema Gerenciador de Banco de Dados (SGBD) *PostgreSQL*. Foi desenvolvida pela própria equipe de desenvolvimento do *PostgreSQL* (Documentação oficial). Para esta apostila será utilizada a versão 4 do software *PgAdmin*.

2.1 ARQUITETURA

O *PostgreSQL* utiliza o modelo cliente-servidor. Uma sessão do *PostgreSQL* possui os seguintes processos, ou seja, programas trabalhando de forma colaborativa (POSTGRES, 2019):

- **Processo servidor:** Este processo gerencia os arquivos de banco de dados, recebe conexões dos processos cliente com o banco de dados, e executa as ações desejadas pelos processos cliente. Este programa servidor é chamado de *postmaster*.
- **Cliente do usuário (*front-end*):** São os processos que desejam executar operações no banco de dados. Podem ser uma ferramenta no modo caractere, pode possuir interface gráfica, pode ser um servidor *web* que acessa o banco de dados e exibe as informações acessadas em uma página *web*.

Aplicações cliente-servidor podem estar alocados em locais (hospedeiros) diferentes. Desta forma a comunicação entre cliente-servidor é efetuada com uma conexão de rede, pelo protocolo TCP/P. O servidor *PostgreSQL* tem a capacidade de tratar várias conexões simultâneas de clientes. Para cada conexão com um cliente é gerado um novo processo chamado *fork* (Documentação oficial).

2.2 *STRUCTURE QUERY LANGUAGE (SQL)*

A linguagem *SQL* utiliza teoria de conjuntos em sua lógica. Um conjunto é uma representação matemática de coleções de elementos, geralmente tem características comuns entre si e podem ter diversas relações internas e externas (Figura 1).

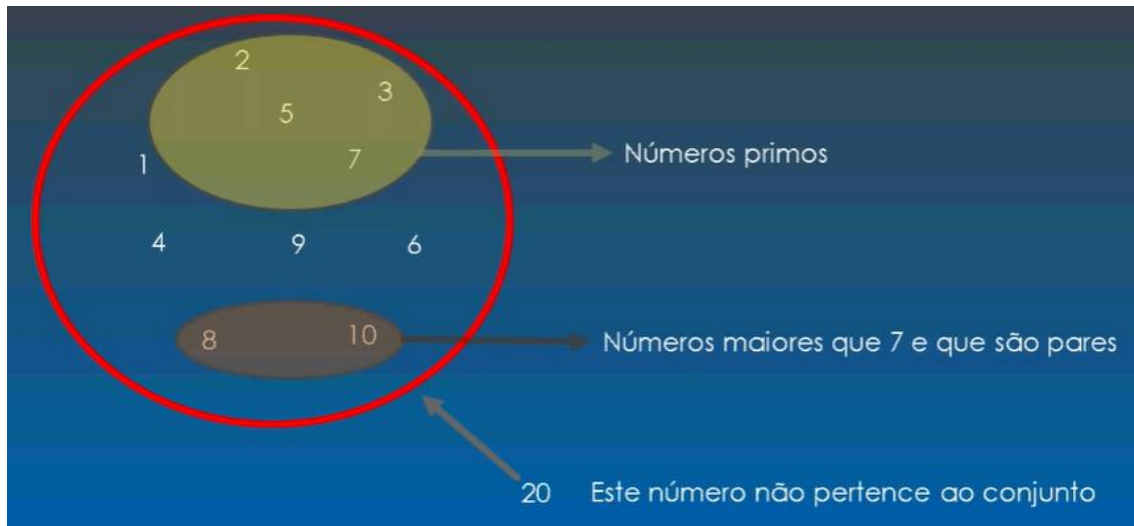


Figura 1. Teoria de conjuntos.

Em SQL as entidades partilham de relacionamentos, baseados na teoria de conjuntos, conforme Figura 2.



Figura 2. Teoria de conjuntos.

Bancos de dados relacionais trabalham tabelas (entidades), as quais possuem campos. Analogicamente a Figura anterior, onde foi representado Clientes, Vendas e Produtos. Em um banco de dados relacional, seria representado conforme Figura 3.

idCliente	Nome	idVenda	Cliente	Produto	idProduto	Descrição
10	João	200	10	101	101	Panela
20	Maria	300	10	102	102	Televisão
30	Marta	400	20	102	103	Geladeira

Figura 3. Representação de bancos de dados relacionais.

Quem comprou televisão, na linguagem SQL? Esta consulta SQL é vista na Figura 4.

1. Quem comprou televisão?	Select Clientes.nome From Clientes, Produtos,Vendas Where Produtos.Descrição = "Televisão" and Produtos.idProduto = Vendas.idVenda and Clientes.idClientes = Vendas.idClientes
----------------------------	---

Figura 4. Exemplo de consulta SQL.

2.3 INSTALAÇÃO E CONFIGURAÇÃO

Neste tópico será abordada a instalação e configuração do *PostgreSQL* server e da ferramenta gráfica utilizada nesta apostila, o *PgAdmin*. A instalação será descrita tanto para ambiente Linux quanto para ambiente Windows.

Para efetuar a instalação do *PostgreSQL* não é necessário nenhum acesso de superusuário (*root*).

2.3.1 Linux

Nesta apostila foi utilizada a distribuição *Ubuntu* do sistema operacional *Linux*. Para realizar a instalação do *PostgreSQL* no *Ubuntu*, é necessário executar o comando apresentado na Figura 5.

```
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

giuvane@giuvane-VirtualBox:~$ sudo apt-get install postgresql
```

Figura 5. Instalação *PostgreSQL* no *Ubuntu*

Ao executar este comando, algumas bibliotecas importantes do *PostgreSQL* serão instaladas, são elas: *libpq5*, *libsensors4*, *libxslt1*, *posrgre-server* e *postgres-client*, dentre outras.

Caso tenha interesse em instalar o *PostgreSQL* com alguns complementos, como a extensão espacial *PostGIS* ou o *R*. O comando a ser executado é descrito na Figura 6:

```
giuvane@giuvane-VirtualBox:~$ sudo apt-get install postgresql-server-dev-all
```

Figura 6. Instalação de pacotes complementares no *Ubuntu*

Caso seja necessário alterar alguma configuração do servidor *PostgreSQL* instalado (por exemplo, alterar o número máximo e conexões simultâneas permitida), é necessário acessar o arquivo de configuração do *PostgreSQL* server, através do comando *vim* (Figura 7)

```
giuvane@giuvane-VirtualBox:~$ vim /etc/postgresql/9.5/postgresql.conf
```

Figura 7. Alterando arquivo de configuração do *PostgreSQL* server no *Ubuntu*

Caso seja necessário habilitar o acesso remoto ao servidor do *PostgreSQL*, é necessário inserir a *tag* de configuração no arquivo ***postgresql.conf***, na área “*Customized options*” ou seja, Opções Customizadas deste arquivo (Figura 8)


```
#-----
# CUSTOMIZED OPTIONS
#-----
# Add settings for extensions here
listen_addresses = '*'
```

Figura 8. Habilitando acesso remoto ao servidor *PostgreSQL* no *Ubuntu*

Com este comando adicionado ao arquivo ***postgresql.conf***, o servidor *PostgreSQL* permitirá o acesso remoto de qualquer endereço IP. Por padrão de instalação, o servidor *PostgreSQL* aceita apenas acessos que estejam na mesma rede, ou seja, na mesma *LAN (Local Area Network)*.

2.3.2 Windows

Para a instalação do *PostgreSQL* em ambiente *Windows*, é necessário efetuar o download do instalador (arquivo executável.exe) no link (<https://www.postgresql.org/download/>) e selecionar a opção “*Windows*”, conforme demonstrado na Figura 9.

Downloads


PostgreSQL Core Distribution

The core of the PostgreSQL object-relational database management system is available in several source and binary formats.

Binary packages

Pre-built binary packages are available for a number of different operating systems:

- BSD
 - FreeBSD
 - OpenBSD
- Linux
 - Red Hat family Linux (including CentOS/Fedora/Scientific/Oracle variants)
 - Debian GNU/Linux and derivatives
 - Ubuntu Linux and derivatives
 - SuSE and OpenSuSE
 - Other Linux
- macOS
- Solaris
- **Windows**

Figura 9. *Download* do instalador do *PostgreSQL* para *Windows*.

Após clicar na opção “*Windows*”, será apresentada uma tabela com várias versões de distribuição do *PostgreSQL*. Para realizar o *download* do instalador, basta clicar em “*Download the installer*” e selecionar a versão desejada (Figura 10).

PostgreSQL Database Download					
PostgreSQL Version	Linux x86-64	Linux x86-32	Mac OS X	Windows x86-64	Windows x86-32
11.3	N/A	N/A	Download	Download	N/A
10.8	Download	Download	Download	Download	Download
9.6.13	Download	Download	Download	Download	Download
9.5.17	Download	Download	Download	Download	Download
9.4.22	Download	Download	Download	Download	Download
9.3.25 (Not Supported)	Download	Download	Download	Download	Download

Figura 10. Versões de instalação do *PostgreSQL* para *Windows*.

A instalação é simples, basta selecionar a pasta destino no qual você deseja instalar o servidor *PostgreSQL* e *software PgAdmin*. Ao instalar o servidor *PostgreSQL* será definida uma senha para o superusuário ‘*postgres*’. Este superusuário tem acesso ilimitado aos recursos do servidor e bancos de dados criados subsequentemente.

A instalação do suporte espacial *PostGIS* pode ser efetuado neste momento, ou futuramente, após ter o *PostgreSQL* e o *PgAdmin* instalados.

2.4 CONCEITOS

O *PostgreSQL* é um *SGBD* para gerenciar dados armazenados em **relações**. **Relação** é um termo matemático utilizado para representar uma tabela. Outras formas de organização de dados seriam arquivos e diretórios, utilizado em Sistemas Operacionais (SO). Cada **tabela**, possui uma coleção de **linhas**. As **linhas** de uma tabela possuem, por padrão, o mesmo conjunto de **colunas** nomeadas, e cada **coluna** é representada por um tipo de dado específico. **Colunas** possuem uma ordem fixa nas linhas (Documentação oficial).

2.5 UTILIZANDO O *PgAdmin*

O *PgAdmin* é um programa cliente que envia e recebe *SQL* ao *PostgreSQL*. É possível exibir resultados e navegar dentre estes resultados. Também é possível acessar muitos servidores *PostgreSQL* e um servidor *PostgreSQL* pode ser acessado por muitos clientes *PgAdmin* ao mesmo tempo.

O instalador do *PostgreSQL* em ambiente *Windows* já traz nativamente a ferramenta *PgAdmin* instalada juntamente o *Postgre server*. Em ambientes *Linux*, é necessário realizar a instalação do *PgAdmin* separadamente.

O endereço oficial do *PgAdmin* é o (<https://www.pgadmin.org/>).

2.5.1 Criando um banco de dados

Para realizar a criação de um novo banco de dados, clique com o botão direito no “Servers”, que se encontra em *Browser* do *PgAdmin 4*, conforme é visto na Figura 11.

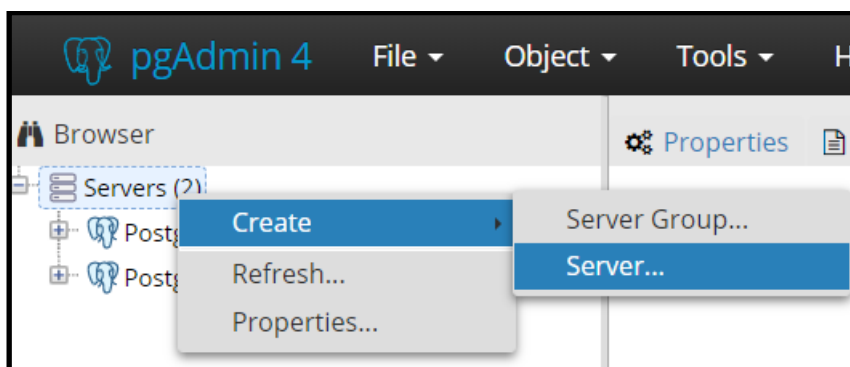


Figura 11. Criando um servidor no *PostgreSQL*.

Agora basta inserir o nome do servidor que se deseja criar, o nome do banco de dados de manutenção que o servidor irá utilizar e informar o usuário e senha de acesso (Figura 12).

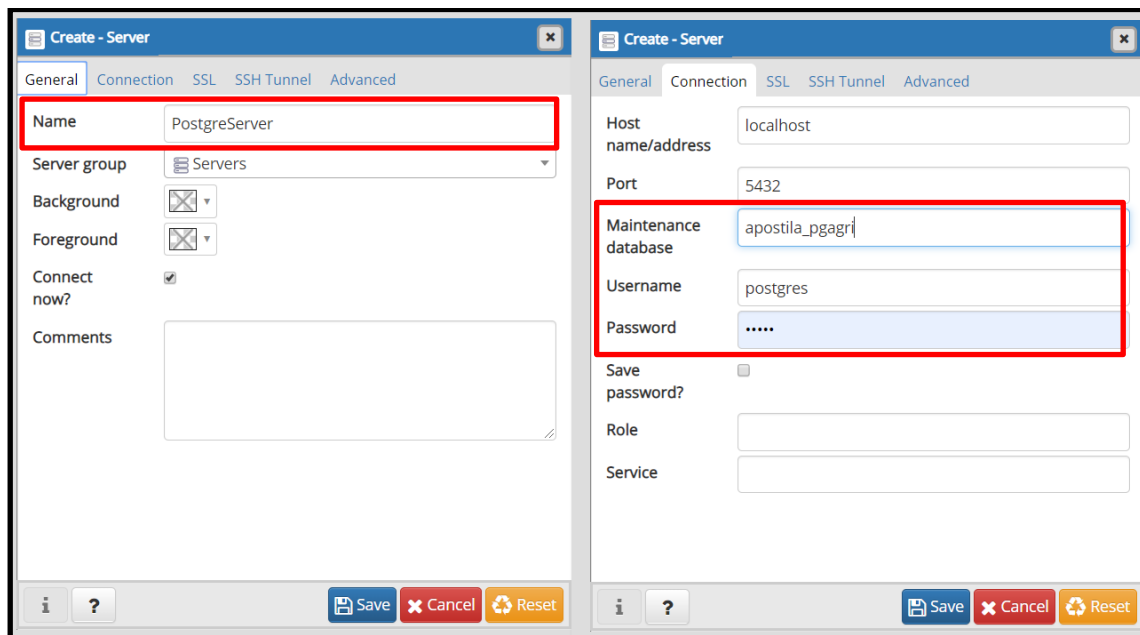


Figura 12. Criando um banco de dados de manutenção no servidor *PostgreSQL* criado.

Para realizar a criação de um novo banco de dados, este sim para ser utilizado por usuários, clique com o botão direito no “*Databases*” do servidor *PostgreSQL* instalado, seleciona as opções: *Create -> Database*, conforme apresentado na Figura 13.

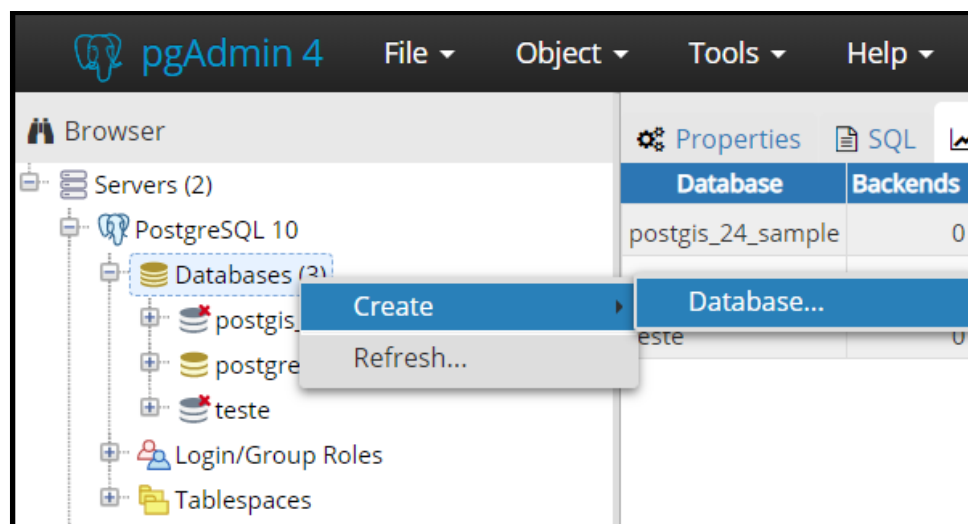


Figura 13. Criando um banco de dados para o usuário.

Na tela de criação de um novo banco de dados, insira o nome da nova base de dados que deseja que seja criação (Figura 14). A opção *Owner* especifica a qual usuário esta base de dados que será criada irá pertencer. As demais abas (*Definition*,

Security, *Parameters* e *SQL*) podem ser utilizadas com a configuração default sugerida pelo *PgAdmin*.

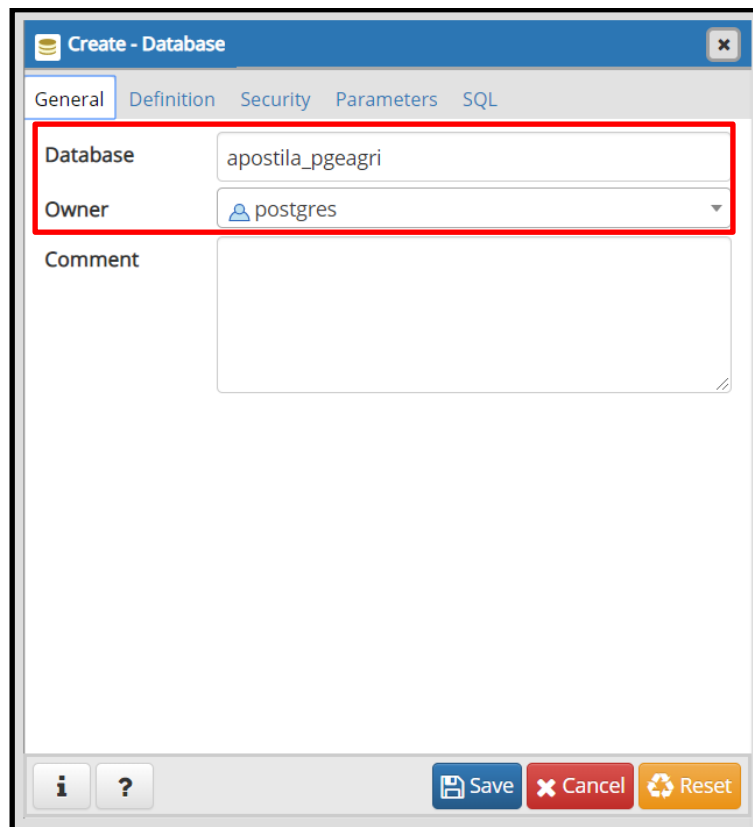


Figura 14. Definindo atributos do novo banco de dados criado.

Após clicar no botão “Save”, clique com o botão direito em *Databases* e posteriormente na opção “Refresh”. O banco de dados ‘*apostila_pgeagri*’ será exibido na lista de bancos de dados do servidor *PostgreSQL*, no nó “*Databases*” (Figura 15).

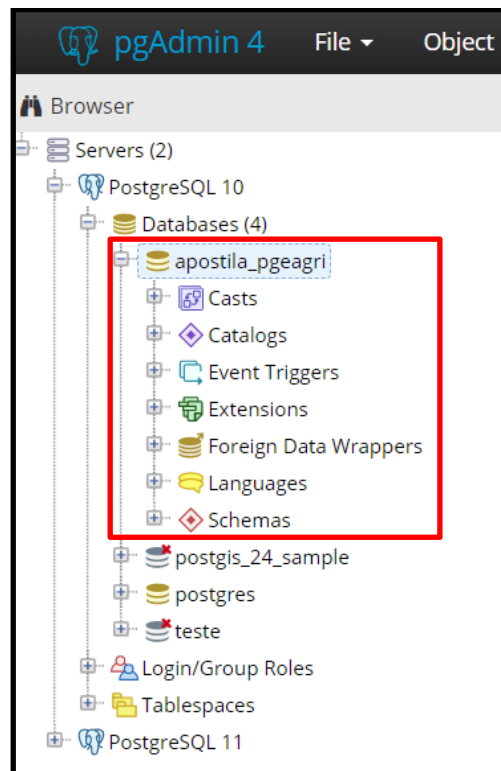


Figura 15. Estrutura do banco de dados criado.

2.5.2 Executando comando SQL

Para executar comandos SQL no *PgAdmin 4* é necessário utilizar uma ferramenta chamada “*Query Tool*”. Esta ferramenta já se encontra embutida junto ao *PgAdmin*, por *default*, após sua instalação.

Para abrir uma nova janela de consulta (*Query Tool*), basta clicar em *Tools -> Query Tool*, conforme apresentado na Figura 16.

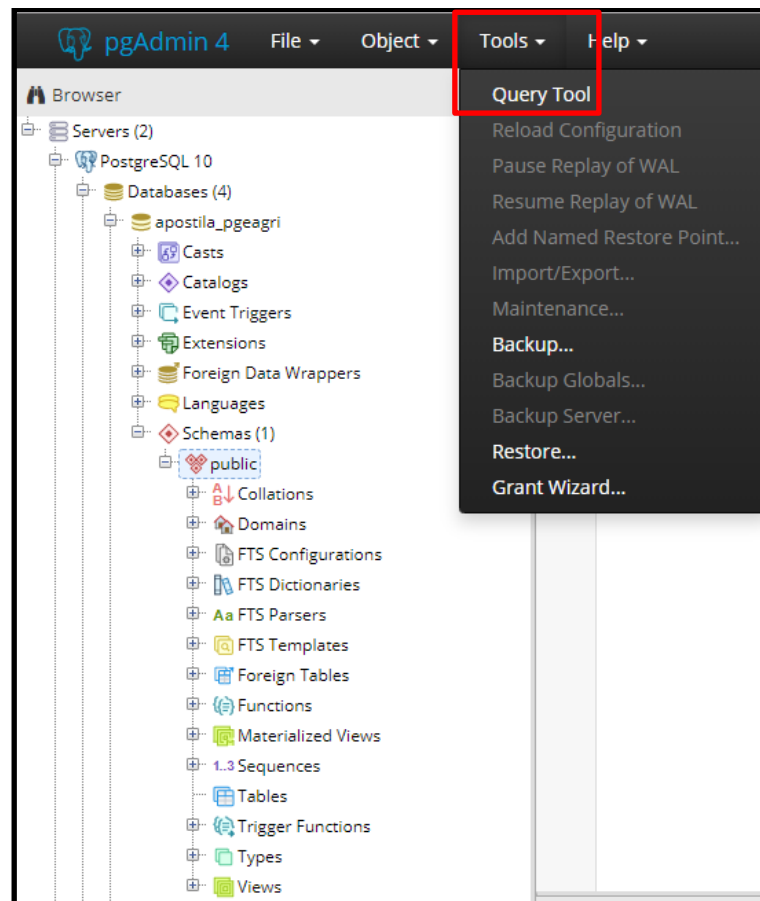


Figura 16. Ferramenta 'Query Tool' para execução de comandos SQL.

Os comandos descritos nesta apostila, serão todos executados via *Query Tool*. A Figura 17 apresenta a barra de tarefas disponibilizada pela ferramenta e suas várias funcionalidades.

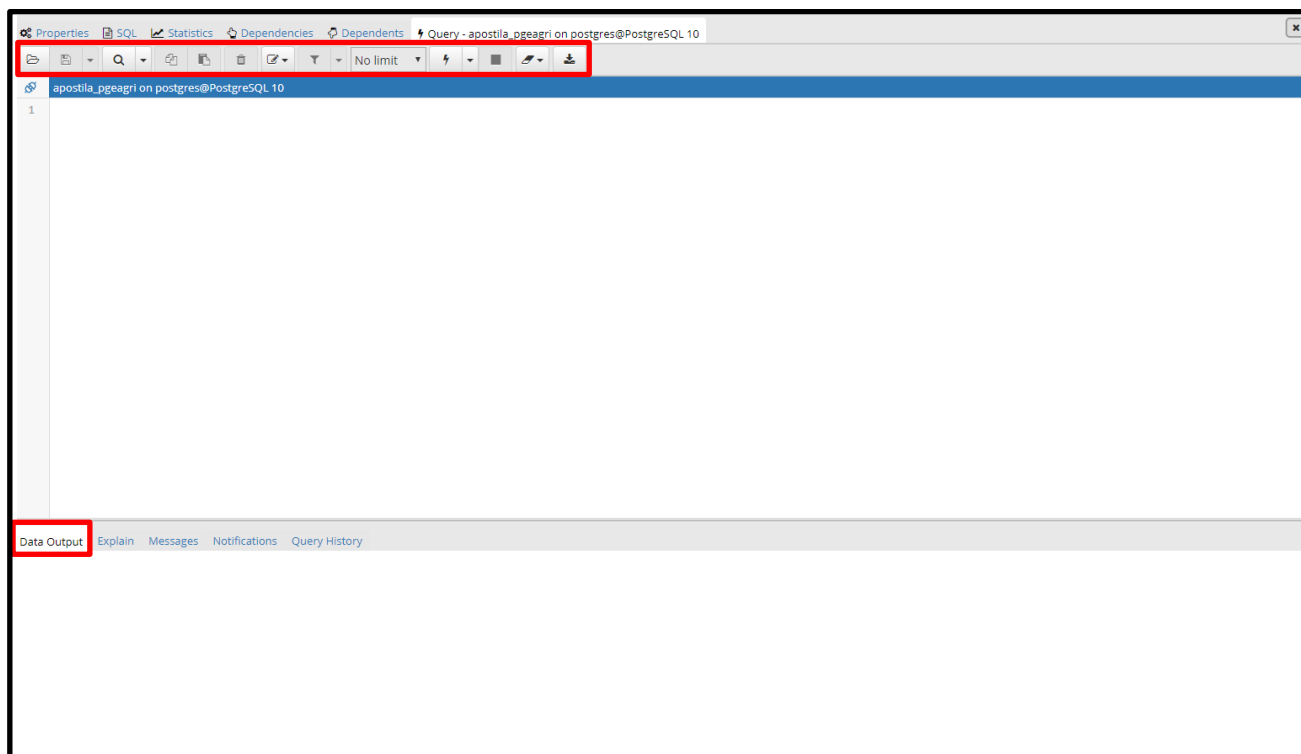


Figura 17. Barra de ferramentas do Query Tool.

Esta barra de tarefas disponibiliza várias funcionalidades, como: Abrir um arquivo SQL; Salvar um novo arquivo SQL; Buscar determinada informação em um arquivo SQL; Indentar um bloco de código selecionado; Comentar/Descomentar linhas e blocos de código; Executar/Atualizar um comando SQL; Limpar arquivo e exportar para arquivo CSV.

Os resultados de comandos SQL de consulta são exibidos na aba “*Data Output*”. As abas “*Explain*”, “*Messages*”, “*Notifications*” e “*Query History*” apresentam informações adicionais, dependendo do comando SQL executado.

A linha destacada em fundo azul, apresenta o status de conexão. Nesta imagem representa a seguinte informação: O *Query Tool* está conectado ao banco de dados “apostila_pgeagri” via usuário “postgres” no servidor *PostgreSQL PostgreSQL 10*.

Um exemplo de execução de comando SQL simples é apresentado na Figura 18, onde é realizada a apresentação da hora atual. Para realizar a execução basta clicar no botão *Execute/Refresh* da barra de tarefas ou pressionar F5.

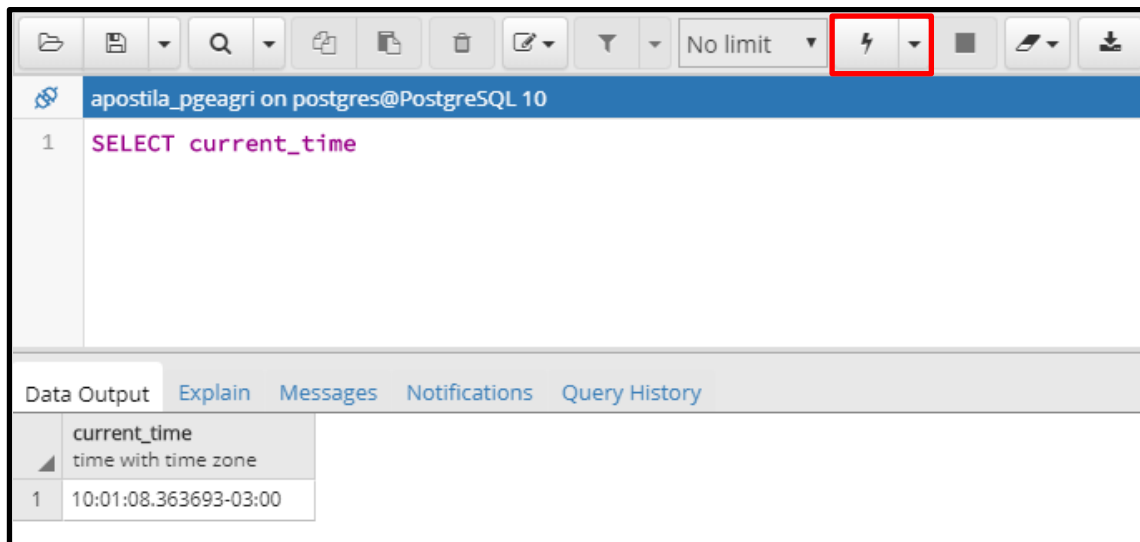


Figura 18. Executando um comando SQL no PgAdmin.

2.6 CONFIGURAÇÃO DE USUÁRIOS E DE CONEXÕES

Por padrão, o PostgreSQL já cria um superusuário, chamado 'postgres'. Este usuário e sua senha já foram definidos na hora de instalação do servidor PostgreSQL. À partir deste superusuário é possível criar novos usuários para o PostgreSQL. Ao criar usuários é possível definir usuários comuns, como até mesmo novos superusuários.

Para consultar qual é o usuário atual conectado, utiliza-se o comando:

```
SELECT current_user;
```

Para criar um usuário e definir seus privilégios, pelo PgAdmin, basta clicar com o botão direito no servidor PostgreSQL e selecionar a opção "Login/Group Role...". A opção é apresentada na Figura 19.

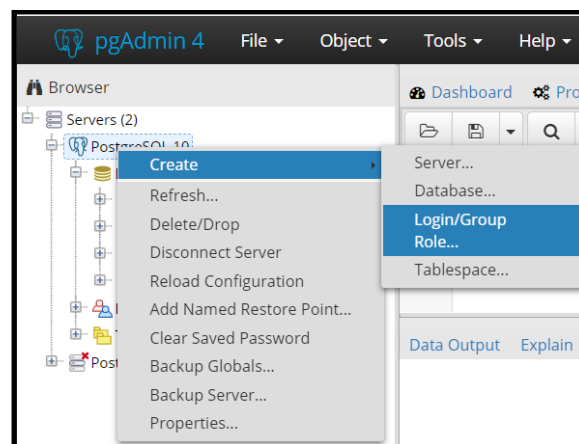


Figura 19. Criando um novo usuário.

Na aba *Privileges* é possível definir os privilégios que serão dados ao usuário (Figura 20).

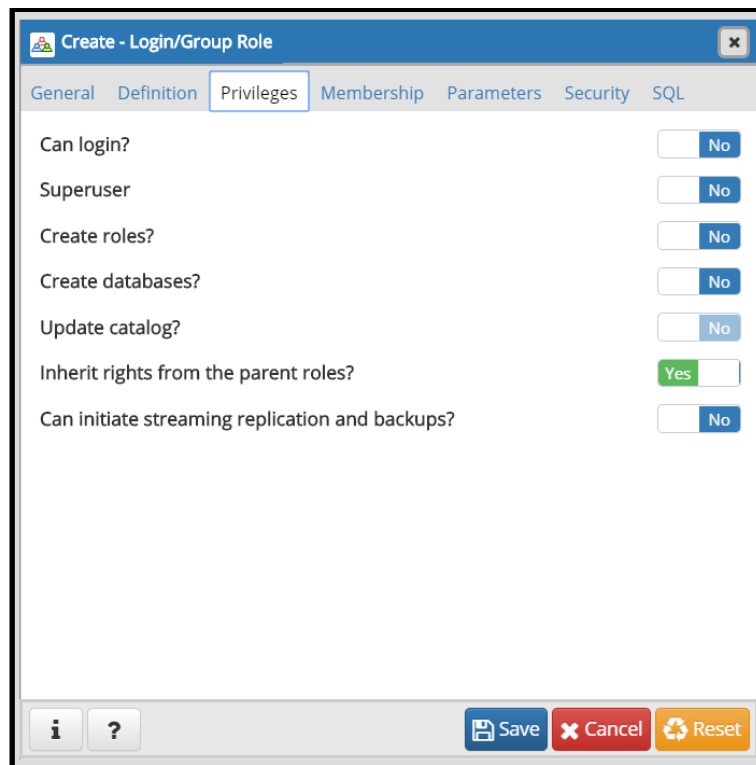


Figura 20. Definindo papéis de um novo usuário.

É possível definir se o novo usuário será um superusuário, se poderá criar outros papéis, se poderá criar *database*, dentre outras opções.

Por linha de comando é possível criar ou remover um usuário, com os seguintes comandos:

```
CREATE USER nome_do_usuario PASSWORD password;
DROP USER nome_do_usuario;
```

Existem vários privilégios distintos aos quais é possível atribuir a um usuário, tais como: *SELECT*, *INSERT*, *UPDATE*, *DELETE*, *RULE*, *REFERENTES*, *TRIGGER*, *CREATE*, *TEMPORARY*, *EXECUTE*, *USAGE* e *ALL PRIVILEGES*. O direito de modificar ou remover um objeto é sempre um privilégio apenas do seu dono (ou de superusuários). É utilizado o comando *GRANT* para conceder privilégios, e o comando *REVOKE* para revogar privilégios.

Por exemplo, imagine que 'pgeagri' seja um usuário existente no servidor PostgreSQL e será atribuído a este usuário o privilégio de atualizar a tabela clima. O seguinte comando concede este privilégio:

```
GRANT UPDATE ON clima TO pgeagri;
```

Para revogar este privilégio ao usuário 'pgeagri', se utiliza o seguinte comando:

```
REVOKE UPDATE ON clima FROM pgeagri;
```

A mesma lógica segue para os demais comandos de privilégios.

2.7 CRIAÇÃO DE UM BANCO DE DADOS POR LINHA DE COMANDO

Um banco de dados é a representação física dos dados, ou seja, o arquivo físico dos dados, no qual está armazenado em dispositivos periféricos. Neste arquivo são armazenados dados de diversos sistemas, disponíveis para consultas, atualizações e exclusões por parte dos usuários que acessam o banco de dados.

Comando para criação de um banco de dados:

```
CREATE DATABASE 'nome_do_banco_de_dados';
```

2.8 CRIAÇÃO DE UM SCHEMA POR LINHA DE COMANDO

Um *schema* é uma coleção de objetos. Estes objetos são organizados em estruturas lógicas ao qual realizam consultas diretamente nos dados que estão armazenados em um banco de dados. Estes *schemas* são constituídos por tabelas, *views*, tipos, *triggers*, funções, domínios, etc. Todo banco de dados criado, por padrão, apresenta um *schema* nomeado como "*public*", mas é possível criar novos *schemas* para um mesmo banco de dados.

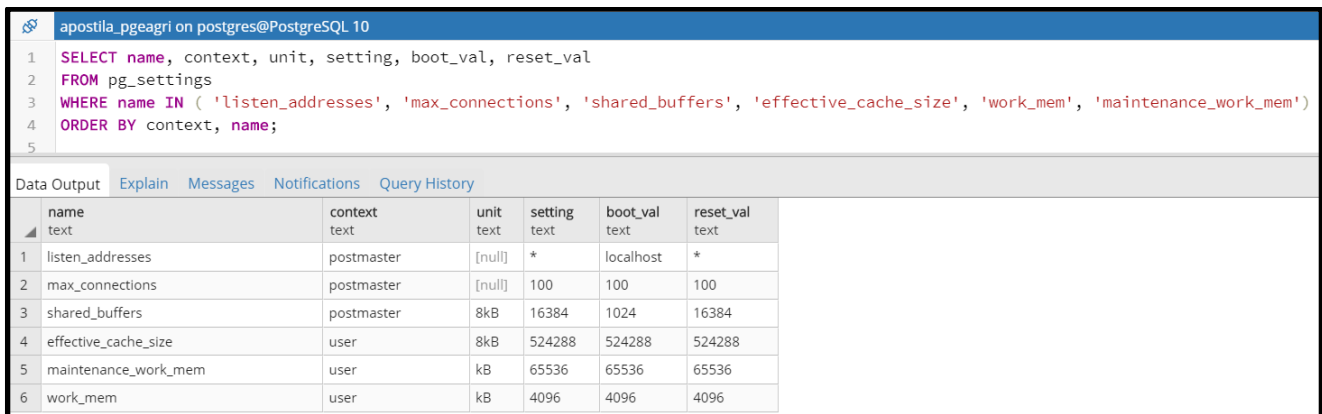
Comando para criação de um *schema*:

```
CREATE SCHEMA 'nome_do_schema';
```

2.9 ARQUIVOS DE CONFIGURAÇÕES

O servidor do *PostgreSQL* possui uma série de arquivos de configuração, aos quais controlam operações básicas em uma instância do *PostgreSQL*, tais como: *postgresql.conf*; *pg_hba.conf*; e *pg_ident.conf*.

É possível executar consultas SQL para visualizar algumas destas configurações disponibilizadas por estes arquivos de configuração. Por exemplo: Realizar uma consulta que exiba os endereços aos quais o *PostgreSQL* possa se conectar, o número máximo de conexões permitidos, o tamanho do *buffer* compartilhado que está configurado, o tamanho efetivo estimado do cache que estará livre, o tamanho de memória limite para operações e memória total alocada para operações (Figura 21).



The screenshot shows a PostgreSQL query editor window titled 'apostila_pgeagri on postgres@PostgreSQL 10'. The query is as follows:

```

1 SELECT name, context, unit, setting, boot_val, reset_val
2 FROM pg_settings
3 WHERE name IN ( 'listen_addresses', 'max_connections', 'shared_buffers', 'effective_cache_size', 'work_mem', 'maintenance_work_mem')
4 ORDER BY context, name;
5

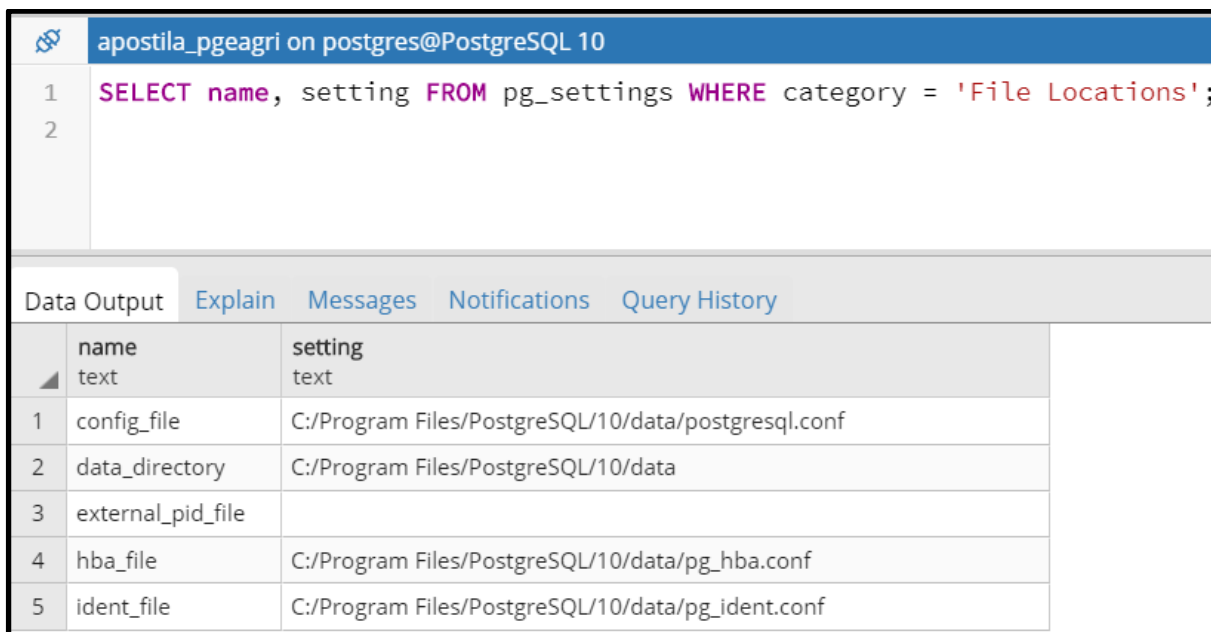
```

The results are displayed in a table with the following columns: name, context, unit, setting, boot_val, and reset_val. The data is as follows:

	name text	context text	unit text	setting text	boot_val text	reset_val text
1	listen_addresses	postmaster	[null]	*	localhost	*
2	max_connections	postmaster	[null]	100	100	100
3	shared_buffers	postmaster	8kB	16384	1024	16384
4	effective_cache_size	user	8kB	524288	524288	524288
5	maintenance_work_mem	user	kB	65536	65536	65536
6	work_mem	user	kB	4096	4096	4096

Figura 21. Consulta SQL que exibe informações disponíveis em arquivos de configuração do *PostgreSQL*.

Outro exemplo (Figura 22): Exibir o local onde estes arquivos de configuração estão armazenados dentro do servidor *PostgreSQL*:



The screenshot shows a PostgreSQL query editor window titled 'apostila_pgeagri on postgres@PostgreSQL 10'. The query is as follows:

```

1 SELECT name, setting FROM pg_settings WHERE category = 'File Locations';
2

```

The results are displayed in a table with the following columns: name, setting. The data is as follows:

	name text	setting text
1	config_file	C:/Program Files/PostgreSQL/10/data/postgresql.conf
2	data_directory	C:/Program Files/PostgreSQL/10/data
3	external_pid_file	
4	hba_file	C:/Program Files/PostgreSQL/10/data/pg_hba.conf
5	ident_file	C:/Program Files/PostgreSQL/10/data/pg_ident.conf

Figura 22. Consulta SQL que exibe o local de armazenamento de arquivos de configuração do *PostgreSQL*.

3. TIPOS DE DADOS

Este tópico apresentará alguns dos tipos de dados utilizados pelo SGBD *PostgreSQL*.

3.1 TIPO NUMÉRICO

O *PostgreSQL* disponibiliza vários tipos específicos para armazenamento de atributos do tipo numérico, cada um com um função e tamanho de armazenamento em disco diferente, o Quadro 1 apresenta os tipos disponíveis e o seu tamanho de armazenamento (POSTGRES, 2019).

Quadro 1. Tipos de dados numéricos.

Fonte: Adaptado de Postgres, 2019.

Nome do tipo	Tamanho de armazenamento
Smallint	2 bytes
Integer	4 bytes
Bigint	8 bytes
Decimal	variável
Numeric	variável
Real	4 bytes
double precision	8 bytes
Serial	4 bytes
Bigserial	8 bytes

Para o armazenamento de números inteiros, são utilizados os tipos *smallint*, *integer* e *bigint*. Números inteiros não possuem a parte fracionária. O tipo usual para armazenar valores inteiros é o *Integer*, pois oferece equilíbrio entre a faixa de valores e espaço utilizado em seu armazenamento (POSTGRES, 2019).

Para armazenamento de aritmética binária de ponto flutuante, onde é necessário precisão simples e dupla, se utilizam os tipos *real* e *double precision* (POSTGRES, 2019).

3.2 TIPO CARACTERE

Para armazenamento de caracteres o *SQL* define dois tipos básicos *varying(n)* e *character(n)*, onde *n* é um número inteiro positivo. Estes tipos podem armazenar

cadeias de caracteres com até n caracteres de comprimento (Quadro 2). No *PostgreSQL* as notações *varchar(n)* e *char(n)* são apelidos para *varying(n)* e *character(n)*. Além destes, o *PostgreSQL* suporta um tipo mais geral, chamado *text*, que armazena cadeias de caractere de qualquer comprimento (POSTGRES, 2019).

Quadro 2. Tipos de dados Caractere.

Fonte: Adaptado de Postgres, 2019.

Nome do tipo	Descrição
<i>character varying(n)</i> , <i>varchar(n)</i>	comprimento variável com limite
<i>character(n)</i> , <i>char(n)</i>	comprimento fixo, completado com brancos
<i>text</i>	comprimento variável não limitado

3.3 TIPO DATA E HORA

O Quadro 3 indica os tipos de campos disponíveis para armazenamentos de data e hora no *PostgreSQL*. *time*, *timestamp* e *interval* aceitam uma valor opcional de precisão p , onde é especificado o número de dígitos fracionários presente no campo de segundo (POSTGRES, 2019).

Quadro 3. Tipos de dado data e hora.

Fonte: Adaptado de Postgres, 2019.

Tipo	Descrição	Armazenamento	Mais cedo	Mais tarde	Resolução
<i>timestamp (p) [without time zone]</i>	tanto data quanto hora	8 bytes	4713 AC	1465001 DC	1 microssegundo / 14 dígitos
<i>timestamp (p) [with time zone]</i>	tanto data quanto hora	8 bytes	4713 BC	AD 1465001	1 microssegundo / 14 dígitos
<i>interval (p)</i>	intervalos de tempo	12 bytes	-178000000 anos	178000000 anos	1 microssegundo
<i>date</i>	somente datas	4 bytes	4713 AC	32767 DC	1 dia
<i>time (p) [without time zone]</i>	somente a hora do dia	8 bytes	00:00:00.00	23:59:59.99	1 microssegundo
<i>time (p) [with time zone]</i>	somente a hora do dia	12 bytes	00:00:00.00+12	23:59:59.99-12	1 microssegundo

3.4 TIPO BOOLEANO

O *PostgreSQL* disponibiliza o tipo padrão booleano do *SQL*, no qual possui apenas dois estados: verdade ou falso. Os valores literais válidos para o estado verdade são: *TRUE*; 't'; 'true'; 'y'; 'yes' e; '1'. Os valores literais válidos para o estado falso são: *FALSE*; 'f'; 'false'; 'n'; 'no' e; '0'.

4. MANIPULAÇÃO DE TABELAS

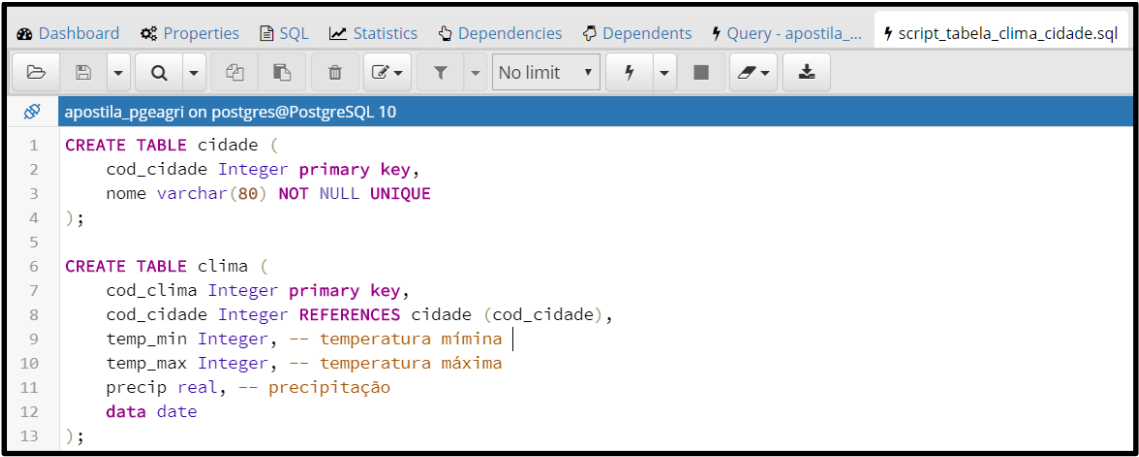
Neste tópico será abordada as linguagens *DDL* (*Data Definition Language*) e *DML* (*Data Manipulation Language*) do *SQL*. Onde *DDL* trata os comandos para criação de estruturas de dados dentro de um banco de dados e o *DML* trata os comandos para realizar consultas, inserções, atualizações e remoções de dados dentro de um banco de dados já criado via *DDL*.

4.1 CRIANDO ESTRUTURAS DE TABELAS

Para a criação de uma tabela é necessário especificar os nomes das colunas e seus respectivos tipos de dados (citados na seção 2.3 Conceitos).

Como exemplo serão criadas duas tabelas: *Clima* e *Cidade*.

A tabela *Cidade* será criada com os campos (colunas) *cod_cidade* e *nome* e a tabela *Clima* será criada com campos (colunas) *cod_clima*, *cod_cidade*, *temp_min*, *temp_max*, *precip* e *data*. O comando *SQL* para a criação desta tabela é apresentado na Figura 23.



```

1 CREATE TABLE cidade (
2     cod_cidade Integer primary key,
3     nome varchar(80) NOT NULL UNIQUE
4 );
5
6 CREATE TABLE clima (
7     cod_clima Integer primary key,
8     cod_cidade Integer REFERENCES cidade (cod_cidade),
9     temp_min Integer, -- temperatura mínima |
10    temp_max Integer, -- temperatura máxima
11    precip real, -- precipitação
12    data date
13 );
  
```

Figura 23. Script de criação das tabelas cidade e clima

É possível notar que o campo *nome* da tabela *cidade*, aceitará uma cadeia de caracteres (*string*) de tamanho máximo 80, ou seja, 80 caracteres, o registro inserido

nunca poderá ser nulo (restrição *NOT NULL*) neste campo e o valor será único (restrição *UNIQUE*), ou seja, não haverá duas linhas (registros) com o mesmo valor.

Os campos *temp_min* e *temp_max* da tabela *clima* aceitarão valores do tipo inteiro. O campo *precip* aceitará valores reais. O campo *date* irá armazenar uma data.

Os campos *cod_clima* na tabela *clima* e *cod_cidade* na tabela *cidade* identificados como '*primary key*' identificam que estes campos são as chaves primárias das tabelas. Isso quer dizer que, estas tabelas não irão aceitar dois registros com a mesma chave primária, ou seja, a linha (registro) com um determinado valor de chave primária será único e nunca poderá ser nulo.

O campo *cod_cidade* da tabela *clima* é uma restrição do tipo *foreign key*, ou seja, chave estrangeira, este valor deve corresponder a algum valor já de chave primária já existente na tabela *cidade*. A chave estrangeira é importante dentro de um SGBD relacional para manter a integridade referencial entre duas tabelas relacionadas.

Espaços em branco, tabulações e novas linhas podem ser utilizados livremente nos comandos *SQL*. Sendo assim, é possível utilizar um alinhamento livre, como na Figura x. O mesmo comando poderia ter sido escrito em uma mesma linha. O compilador do *PostgreSQL* irá desconsiderar estes caracteres e levará em consideração apenas as palavras reservadas da linguagem e os nomes corretos atribuídos ao comando desejado (neste caso, nome de tabela e nomes de campos). Dois hifens (--) indicam comentário, sendo assim, todo texto após os dois hifens é ignorado até o final da linha.

A linguagem *SQL* não diferencia letras maiúsculas de minúsculas em suas palavras chave e em seus identificadores.

4.2 ALTERANDO ESTRUTURAS DE TABELAS

O *PostgreSQL* disponibiliza um conjunto de comandos para realizar modificações em tabelas já existentes. É possível adicionar e remover colunas, adicionar e remover restrições, alterar o tipo de um dado de uma coluna, mudar o nome de uma coluna ou até mesmo mudar o nome de uma tabela.

Por exemplo, para alterar a tabela *clima*, adicionando um campo descrição do tipo *text* com restrição *NOT NULL*, utiliza-se o comando:

```
ALTER TABLE clima ADD COLUMN descrição text NOT NULL;
```

As variações para alterar a estrutura de uma tabela mudam pouco. Por exemplo, para remover o campo descrição criado, utiliza-se o comando:

ALTER TABLE clima *DROP COLUMN* descrição;

4.3 EXCLUINDO TABELAS

E exclusão de uma tabela é realizada através do comando SQL *'Drop'*, seguido da identificação da tabela na qual se deseja excluir, como na Figura 24.

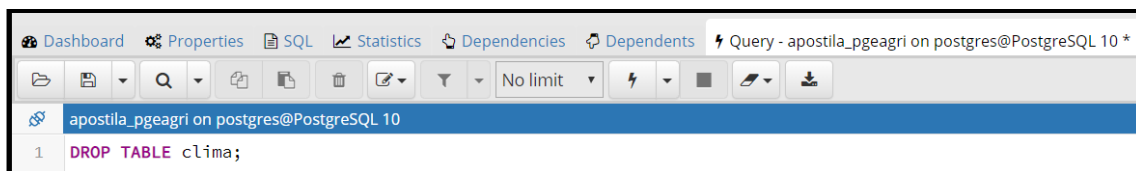


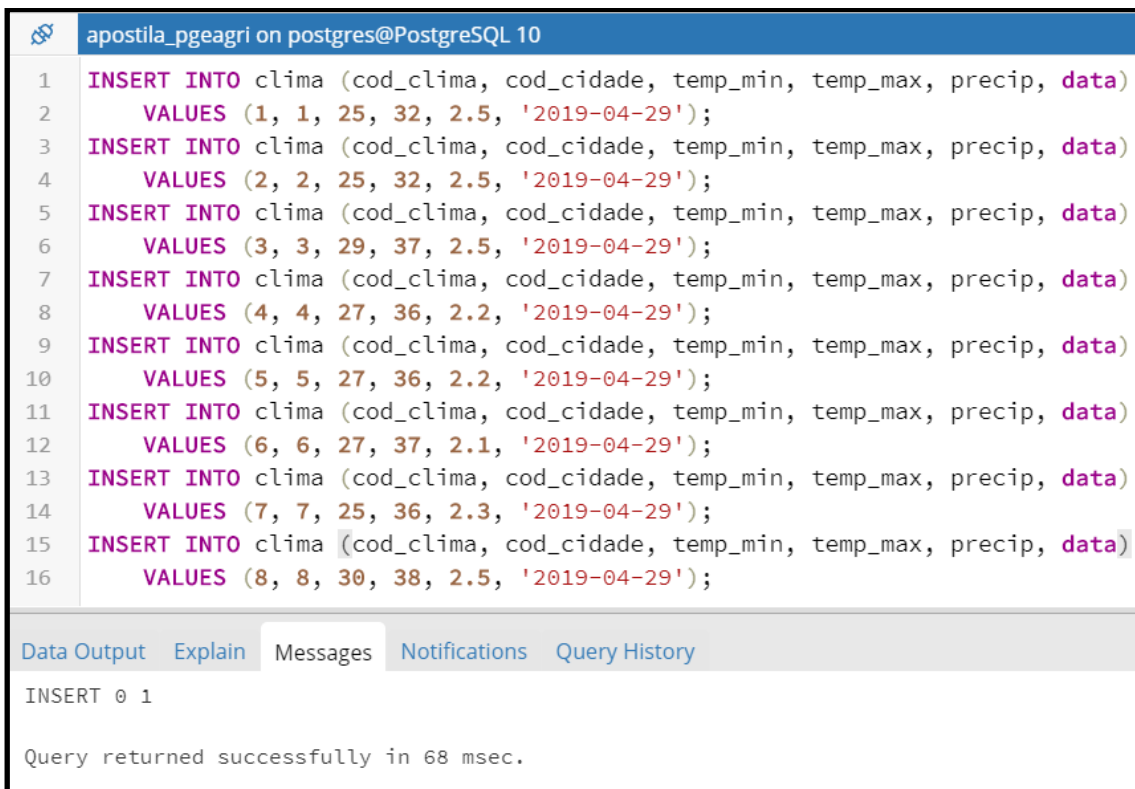
Figura 24. Exclusão da tabela clima.

4.4 INSERÇÃO DE LINHAS EM TABELAS

Para realizar a inserção de dados, ou seja, linhas (também chamados de registros), em tabelas criadas é utilizado o comando SQL *'Insert'*. A inserção de dados nas tabelas Cidade e Clima (criadas anteriormente), são apresentadas nas Figura 25 e Figura 26.



Figura 25. Inserindo linhas (registros) na tabela cidade.



```

apostila_pgeagri on postgres@PostgreSQL 10
1  INSERT INTO clima (cod_clima, cod_cidade, temp_min, temp_max, precip, data)
2    VALUES (1, 1, 25, 32, 2.5, '2019-04-29');
3  INSERT INTO clima (cod_clima, cod_cidade, temp_min, temp_max, precip, data)
4    VALUES (2, 2, 25, 32, 2.5, '2019-04-29');
5  INSERT INTO clima (cod_clima, cod_cidade, temp_min, temp_max, precip, data)
6    VALUES (3, 3, 29, 37, 2.5, '2019-04-29');
7  INSERT INTO clima (cod_clima, cod_cidade, temp_min, temp_max, precip, data)
8    VALUES (4, 4, 27, 36, 2.2, '2019-04-29');
9  INSERT INTO clima (cod_clima, cod_cidade, temp_min, temp_max, precip, data)
10   VALUES (5, 5, 27, 36, 2.2, '2019-04-29');
11 INSERT INTO clima (cod_clima, cod_cidade, temp_min, temp_max, precip, data)
12   VALUES (6, 6, 27, 37, 2.1, '2019-04-29');
13 INSERT INTO clima (cod_clima, cod_cidade, temp_min, temp_max, precip, data)
14   VALUES (7, 7, 25, 36, 2.3, '2019-04-29');
15 INSERT INTO clima (cod_clima, cod_cidade, temp_min, temp_max, precip, data)
16   VALUES (8, 8, 30, 38, 2.5, '2019-04-29');

Data Output Explain Messages Notifications Query History
INSERT 0 1
Query returned successfully in 68 msec.

```

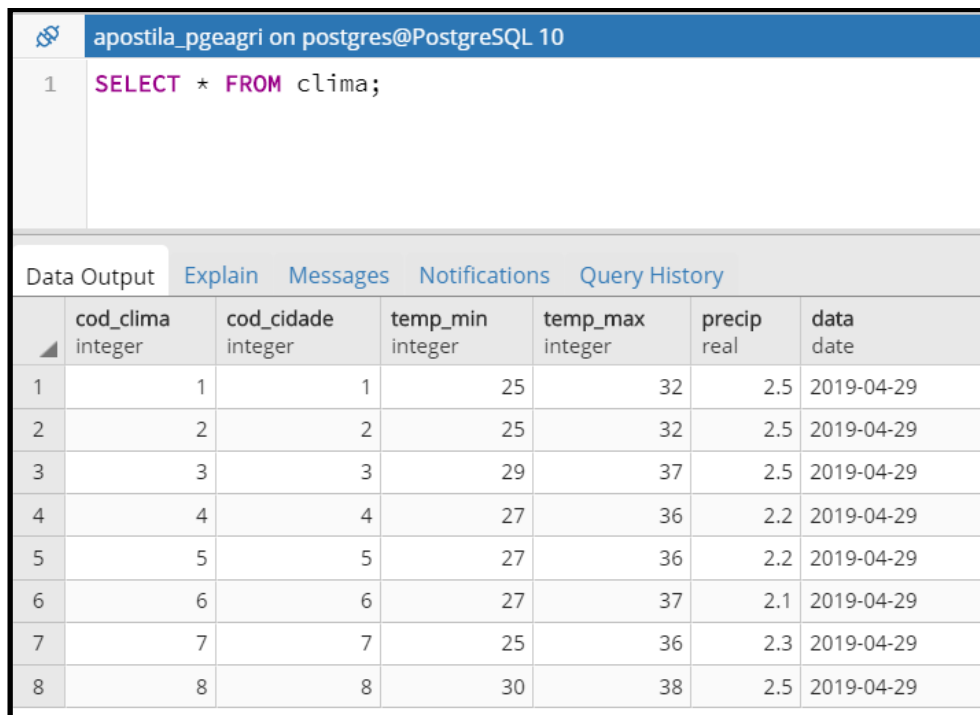
Figura 26. Inserindo linhas (registros) na tabela clima.

Na Figura 25 foram adicionados os valores inteiros para a chave primária (campo `cod_clima`) e o nome da cidade (campo `nome`), com cadeiras de caracteres identificados entre " (aspas).

Na Figura 26, além dos valores para a chave primária (`cod_clima`), no campo `cod_cidade` (chave estrangeira) foi atribuído um valor já existente na tabela cidade. Nos campos `temp_min` e `temp_max` foram adicionados valores inteiros. No campo `precip` foi necessário utilizar um valor do tipo ponto flutuante (com casas decimais), pois seu tipo é o tipo real. Para o campo `data` foi utilizada a formatação padrão do *SQL* para o tipo *date*, onde é identificado por 'YYYY-MM-DD', ou seja 'ANO-MÊS-DIA'.

4.5 CONSULTANDO DADOS DE TABELAS

Para realizar a recuperação de dados armazenados em uma tabela, é utilizado o comando *SQL* *Select*. Este comando é dividido em **lista de seleção**, **lista de tabelas** e **restrições**. Por exemplo, a Figura 27 apresenta um caso onde se desejou selecionar todos os campos (lista de seleção), da tabela clima (lista de tabelas), sem nenhuma restrição. O * identifica que foram recuperados todos os campos (colunas) da tabela clima.



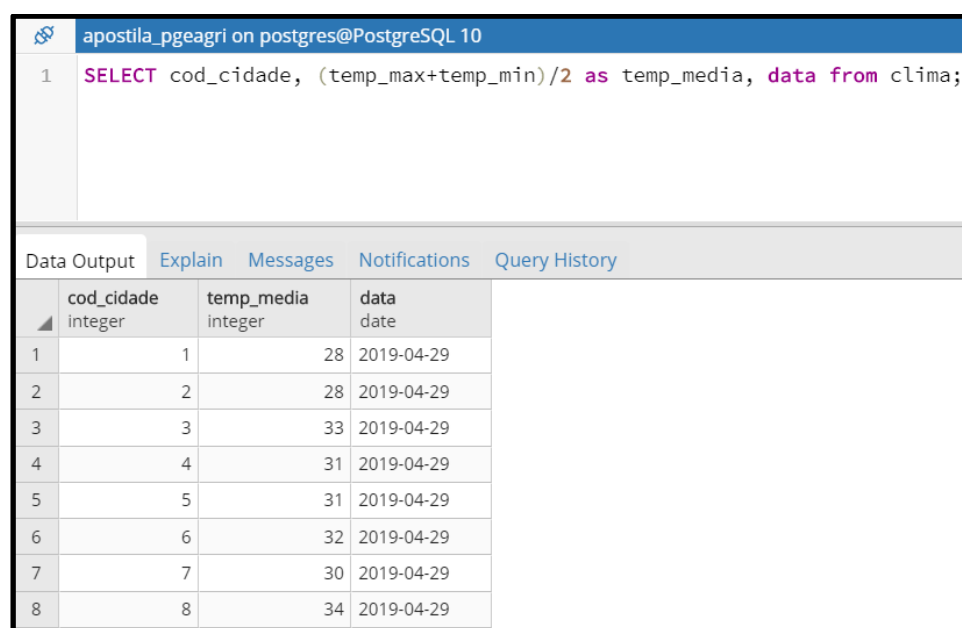
The screenshot shows a PostgreSQL query editor with the title 'apostila_pgeagri on postgres@PostgreSQL 10'. The query entered is `SELECT * FROM clima;`. Below the query, there are tabs for 'Data Output', 'Explain', 'Messages', 'Notifications', and 'Query History'. The 'Data Output' tab is selected, displaying a table with 8 rows and 7 columns: `cod_clima` (integer), `cod_cidade` (integer), `temp_min` (integer), `temp_max` (integer), `precip` (real), and `data` (date).

	cod_clima integer	cod_cidade integer	temp_min integer	temp_max integer	precip real	data date
1	1	1	25	32	2.5	2019-04-29
2	2	2	25	32	2.5	2019-04-29
3	3	3	29	37	2.5	2019-04-29
4	4	4	27	36	2.2	2019-04-29
5	5	5	27	36	2.2	2019-04-29
6	6	6	27	37	2.1	2019-04-29
7	7	7	25	36	2.3	2019-04-29
8	8	8	30	38	2.5	2019-04-29

Figura 27. Consulta SQL básica na tabela clima.

É possível adicionar expressões às consultas SQL. Por exemplo, imagina que seja necessário saber a temperatura média dos registros (linhas) da tabela clima. Para isso uma expressão aritmética pode ser adicionada à consulta (Figura 28).

Foram inseridas mais linhas (registros) na tabela clima para elucidar melhor os exemplos apresentados.



The screenshot shows the same PostgreSQL query editor with a new query: `SELECT cod_cidade, (temp_max+temp_min)/2 as temp_media, data from clima;`. The 'Data Output' tab is selected, displaying a table with 8 rows and 4 columns: `cod_cidade` (integer), `temp_media` (integer), and `data` (date).

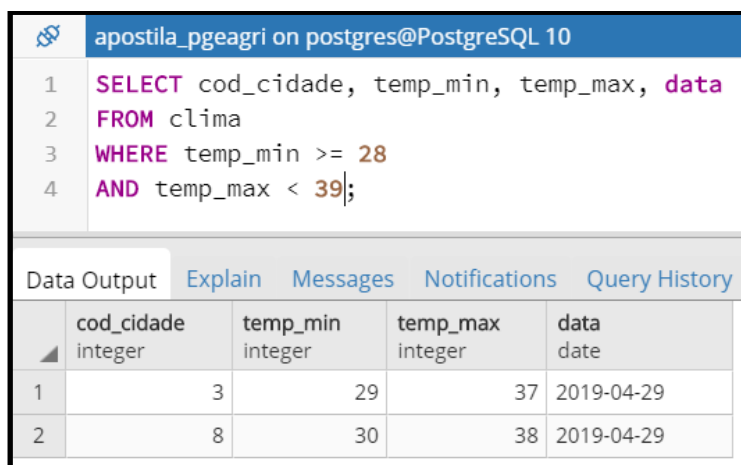
	cod_cidade integer	temp_media integer	data date
1	1	28	2019-04-29
2	2	28	2019-04-29
3	3	33	2019-04-29
4	4	31	2019-04-29
5	5	31	2019-04-29
6	6	32	2019-04-29
7	7	30	2019-04-29
8	8	34	2019-04-29

Figura 28. Consulta SQL com expressão aritmética.

O comando 'as' identifica que, neste campo consultado, um apelido 'temp_media' será apresentado após a execução da consulta, em sua coluna de saída. Esta cláusula é opcional.

Para adicionar restrições a uma consulta SQL, é utilizado o comando 'Where'. Este comando contém expressões booleanas (verdadeiro ou falso), e somente são retornadas linhas as quais esta expressão booleana for satisfeita. Para isto, são permitidos operadores booleanos usuais de linguagens de programação e lógica matemática (AND, OR e NOT).

A Figura 29 apresenta uma consulta SQL com restrições, onde são consultados apenas as linhas com temperatura mínima maior ou igual a 28 graus e temperatura máxima menor que 39 graus.



The screenshot shows a PostgreSQL query editor window titled 'apostila_pgeagri on postgres@PostgreSQL 10'. The query is as follows:

```

1 SELECT cod_cidade, temp_min, temp_max, data
2 FROM clima
3 WHERE temp_min >= 28
4 AND temp_max < 39;

```


Below the query, there are tabs for 'Data Output', 'Explain', 'Messages', 'Notifications', and 'Query History'. The 'Data Output' tab is selected, showing the following results:

	cod_cidade integer	temp_min integer	temp_max integer	data date
1	3	29	37	2019-04-29
2	8	30	38	2019-04-29

Figura 29. Consulta SQL com restrições de linhas.

4.6 JUNÇÕES ENTRE TABELAS

As consultas podem acessar várias linhas de uma ou mais tabelas ao mesmo tempo. Consultas que acessam várias linhas de tabelas diferentes, de uma vez só vez, são chamadas de *junções (joins)*. Como exemplo, suponha que seja necessário listar todas as linhas de clima, junto com o nome de suas respectivas cidades associadas. Para isto, é necessário comparar a coluna *cod_cidade* da tabela *clima* (chave estrangeira) com a coluna *cod_cidade* da tabela *cidade* (chave primária). Desta forma, selecionando os pares de linha onde estes valores são correspondentes. A Figura 30 apresenta este exemplo na prática.



apostila_pgeagri on postgres@PostgreSQL 10

1

SELECT cid.nome, cli.temp_min, cli.temp_max, cli.data

2

FROM cidade **as** cid, clima **as** cli

3

WHERE cid.cod_cidade = cli.cod_cidade

Data Output

Explain

Messages

Notifications

Query History

	nome character varying (80)	temp_min integer	temp_max integer	data date
1	Santa Helena	25	32	2019-04-29
2	Cascavel	25	32	2019-04-29
3	Foz do Iguaçu	29	37	2019-04-29
4	Medianeira	27	36	2019-04-29
5	Matelândia	27	36	2019-04-29
6	Vera Cruz do Oeste	27	37	2019-04-29
7	Missal	25	36	2019-04-29
8	São Miguel do Iguaçu	30	38	2019-04-29

Figura 30. Consulta SQL com junção de tabelas com *WHERE*.

Esta mesma consulta poderia ser escrita utilizando o comando '*Join*', conforme é visto na Figura 31.

apostila_pgeagri on postgres@PostgreSQL 10

```
1 SELECT cid.nome, cli.temp_min, cli.temp_max, cli.data
2 FROM cidade as cid
3 JOIN clima as cli ON (cid.cod_cidade = cli.cod_cidade)
```

Data Output

Explain

Messages

Notifications

Query History

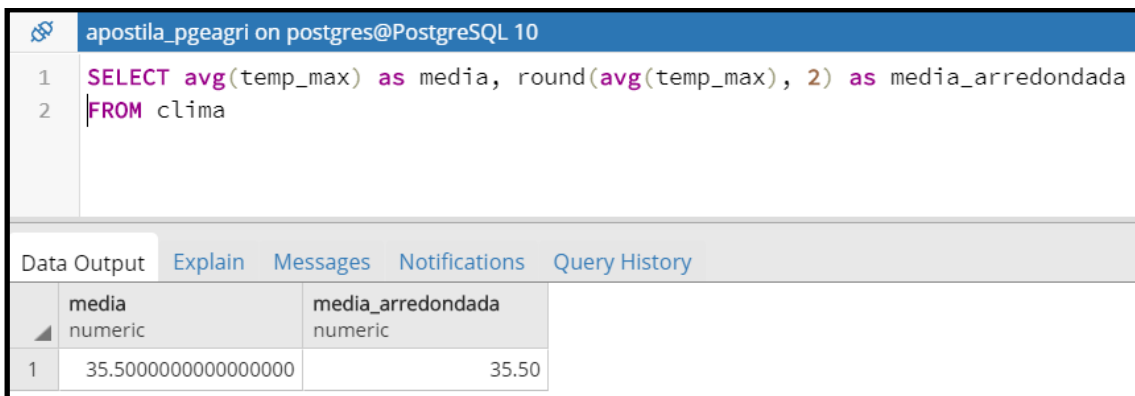
	nome character varying (80)	temp_min integer	temp_max integer	data date
1	Santa Helena	25	32	2019-04-29
2	Cascavel	25	32	2019-04-29
3	Foz do Iguaçu	29	37	2019-04-29
4	Medianeira	27	36	2019-04-29
5	Matelândia	27	36	2019-04-29
6	Vera Cruz do Oeste	27	37	2019-04-29
7	Missal	25	36	2019-04-29
8	São Miguel do Iguaçu	30	38	2019-04-29

Figura 31. Consulta SQL com junção de tabelas com *JOIN*.

4.7 FUNÇÕES DE AGREGAÇÃO

Uma função de agregação computa um único resultado para várias linhas de entrada. Por exemplo, é possível utilizar funções de agregação para contar (count), somar (sum), calcular média (avg), o valor máximo (max) e o valor mínimo (min) para um conjunto de linhas (registros).

Para representar, imagine que seja necessário recuperar o valor médio de temperaturas máximas entre todas as linhas da tabela Clima, a Figura 32 apresentar este exemplo implementado.



The screenshot shows a PostgreSQL query editor with the following SQL query:

```
1 SELECT avg(temp_max) as media, round(avg(temp_max), 2) as media_arredondada
2 FROM clima
```

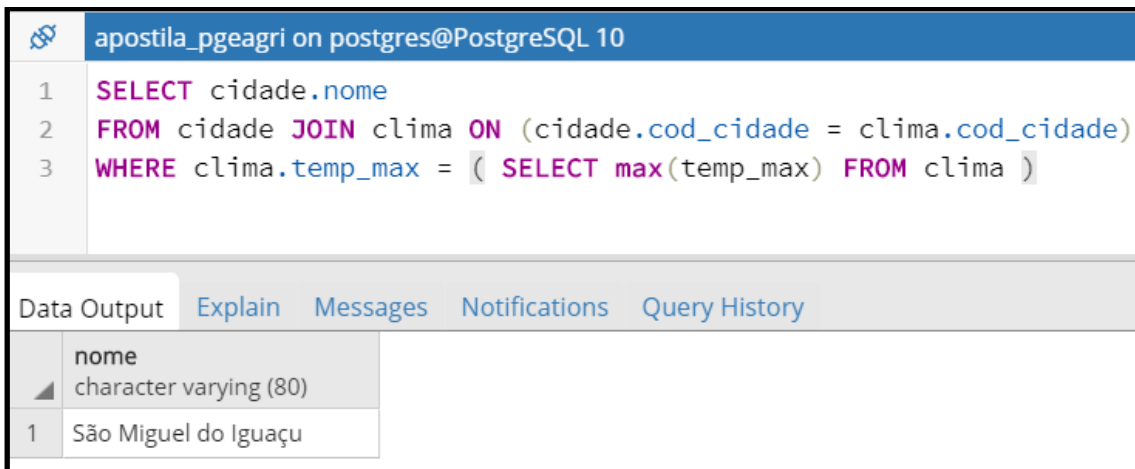
Below the query, the 'Data Output' tab is selected, showing the results of the query:

	media numeric	media_arredondada numeric
1	35.5000000000000000	35.50

Figura 32. Consulta SQL com função de agregação.

A segunda coluna retornada, apresenta o valor médio com um arredondamento de 2 casas decimais. Este arredondamento foi aplicado utilizando a função de linha 'Round'.

Agora imagine que seja necessário encontrar o lugar, ou os lugares, onde foram registradas as maiores temperaturas, utilizando a função agregação 'Max' (Figura 33).



The screenshot shows a PostgreSQL query editor with the following SQL query:

```
1 SELECT cidade.nome
2 FROM cidade JOIN clima ON (cidade.cod_cidade = clima.cod_cidade)
3 WHERE clima.temp_max = ( SELECT max(temp_max) FROM clima )
```

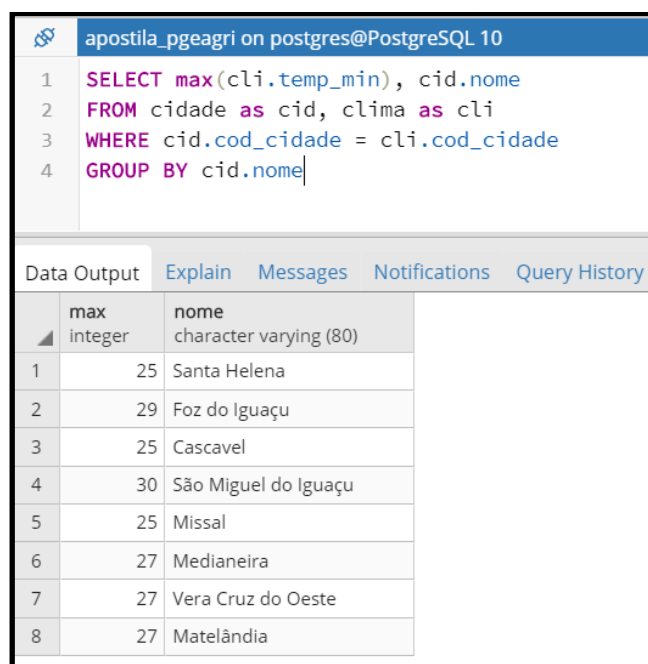
Below the query, the 'Data Output' tab is selected, showing the results of the query:

	nome character varying (80)
1	São Miguel do Iguaçu

Figura 33. Consulta SQL com função de agregação 'Max' e Subconsulta.

Foi utilizada a junção de tabelas para realizar a junção de consulta entre as tabelas clima e cidade. Na clausula *WHERE* (restrição de dados da consulta) onde verifica a comparação entre o maior valor registrado de temperaturas, foi necessário utilizar um ‘*Subselect*’ (Subconsulta) para realizar a comparação, isso porque funções de agregação não podem ser utilizadas diretamente em uma clausula *WHERE*. Uma subconsulta é uma ação independente, que calcula sua agregação em separado do que está acontecendo na consulta externa.

Agregações também são úteis se utilizadas combinadas com a clausula *GROUP BY*. A clausula *GROUP BY* pode concentrar a agregação em informações comuns em uma consulta, por exemplo, a Figura 34 apresenta a maior temperatura mínima observada em uma determinada cidade.



The screenshot shows a PostgreSQL query editor window titled 'apostila_pgeagri on postgres@PostgreSQL 10'. The query is as follows:

```

1 SELECT max(cli.temp_min), cid.nome
2 FROM cidade as cid, clima as cli
3 WHERE cid.cod_cidade = cli.cod_cidade
4 GROUP BY cid.nome

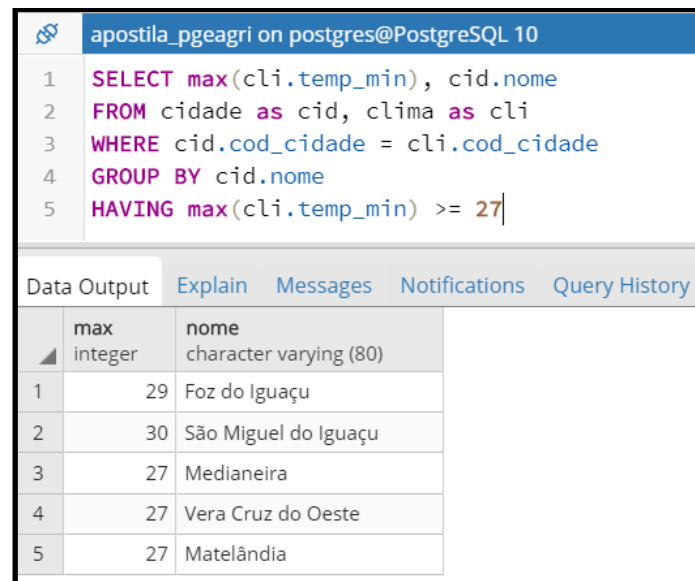
```

Below the query editor, there is a tabbed interface with 'Data Output' selected. The results are displayed in a table with two columns: 'max' (integer) and 'nome' (character varying (80)).

	max integer	nome character varying (80)
1	25	Santa Helena
2	29	Foz do Iguaçu
3	25	Cascavel
4	30	São Miguel do Iguaçu
5	25	Missal
6	27	Medianeira
7	27	Vera Cruz do Oeste
8	27	Matelândia

Figura 34. Consulta SQL com função de agregação e clausula *GROUP BY*.

As linhas agrupadas podem ser filtradas utilizando a clausula *HAVING*. A Figura 35 apresenta o mesmo exemplo executado anteriormente, porém desta vez apresentando apenas as cidades que tiveram suas máximas temperaturas mínimas iguais ou maiores que 27 graus.



```

1 SELECT max(cli.temp_min), cid.nome
2 FROM cidade as cid, clima as cli
3 WHERE cid.cod_cidade = cli.cod_cidade
4 GROUP BY cid.nome
5 HAVING max(cli.temp_min) >= 27

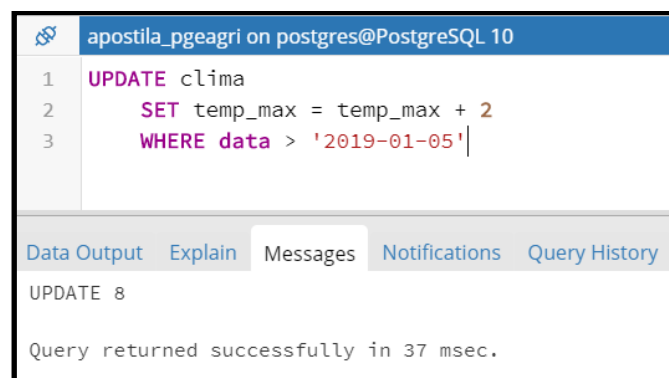
```

	max integer	nome character varying (80)
1	29	Foz do Iguaçu
2	30	São Miguel do Iguaçu
3	27	Medianeira
4	27	Vera Cruz do Oeste
5	27	Matelândia

Figura 35. Consulta SQL com função de agregação, clausula *GROUP BY* e restrição de agrupamento *HAVING*.

4.8 ATUALIZAÇÕES E EXCLUSÕES DE LINHAS

As linhas existentes podem ser atualizadas através do comando SQL *'Update'*. Imagine o seguinte cenário, todas as temperaturas máximas persistidas na tabela Clima após 05/01/2019 estão incorretas. É necessário atualizar todos os registros aumentar seus valores em 3 graus. A Figura 36 apresenta o comando SQL que realiza está tarefa.



```

1 UPDATE clima
2 SET temp_max = temp_max + 2
3 WHERE data > '2019-01-05'

```

	max integer	nome character varying (80)
1	29	Foz do Iguaçu
2	30	São Miguel do Iguaçu
3	27	Medianeira
4	27	Vera Cruz do Oeste
5	27	Matelândia

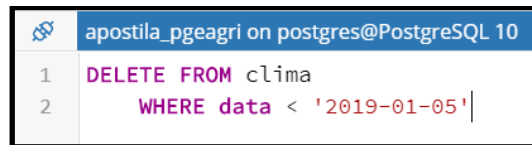
UPDATE 8

Query returned successfully in 37 msec.

Figura 36. Atualizando registros de uma tabela.

Na aba *'Messages'* foi indicado que os 8 registros nos quais se enquadravam na restrição de data foram atualizados.

Agora imagine que seria necessário excluir todos os registros da tabela clima anteriores a data de 05/01/2019. A Figura 37 apresenta este comando SQL.



```

apostila_pgeagri on postgres@PostgreSQL 10
1 DELETE FROM clima
2 WHERE data < '2019-01-05'

```

Figura 37. Excluindo registros de uma tabela.

É importante sempre verificar se o comando SQL está sendo executado com clausula WHERE, visto que, caso seja executado sem *WHERE*, todos os registros da tabela são excluídos.

4.9 VISÕES

Imagine que a consultas citada anteriormente envolvendo clima e cidade seja particularmente importante para uma aplicação, e por ser muito utilizada dentro desta aplicação, não é interessante que ela seja digitada a todo momento, pois ela apresenta uma certa complexidade quanto a junção de tabelas.

Para isto é possível criar uma *view* (visão) baseada na consulta, onde é possível atribuir um nome a esta visão e, futuramente ela pode ser referenciada como se fosse uma tabela comum. A ideia é que uma visão criada, funcione como um ‘atalho’ para uma consulta maior, que sua implementação seja complexa. A Figura 38 apresenta a criação desta visão.

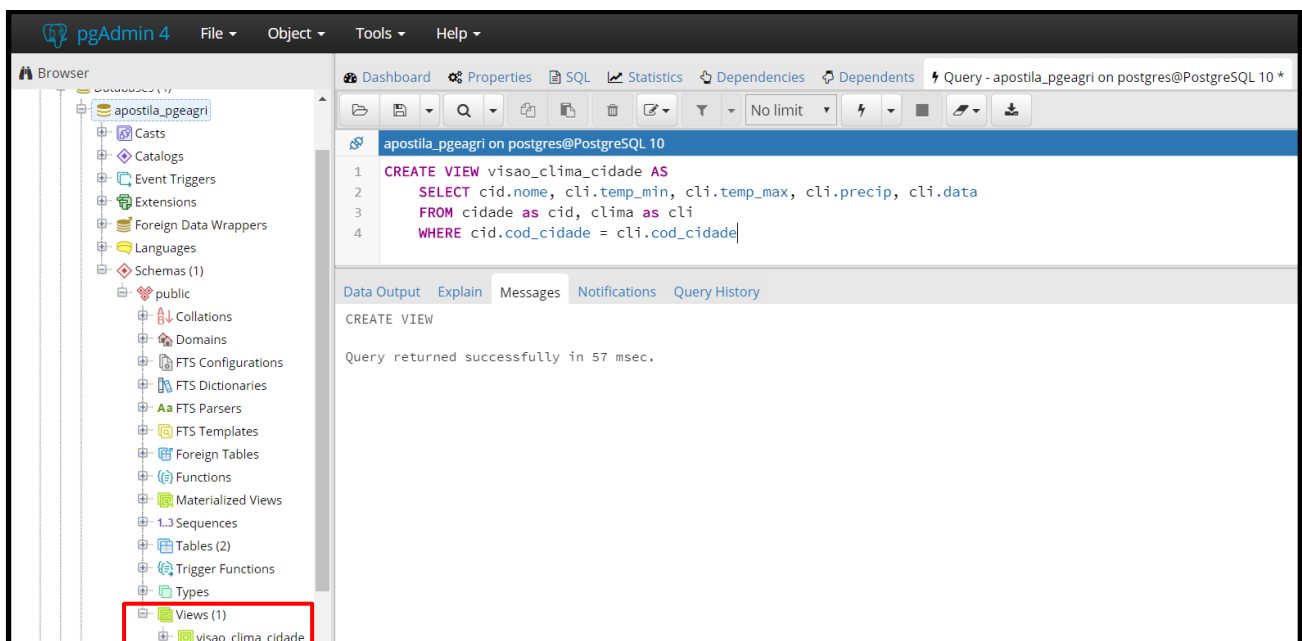
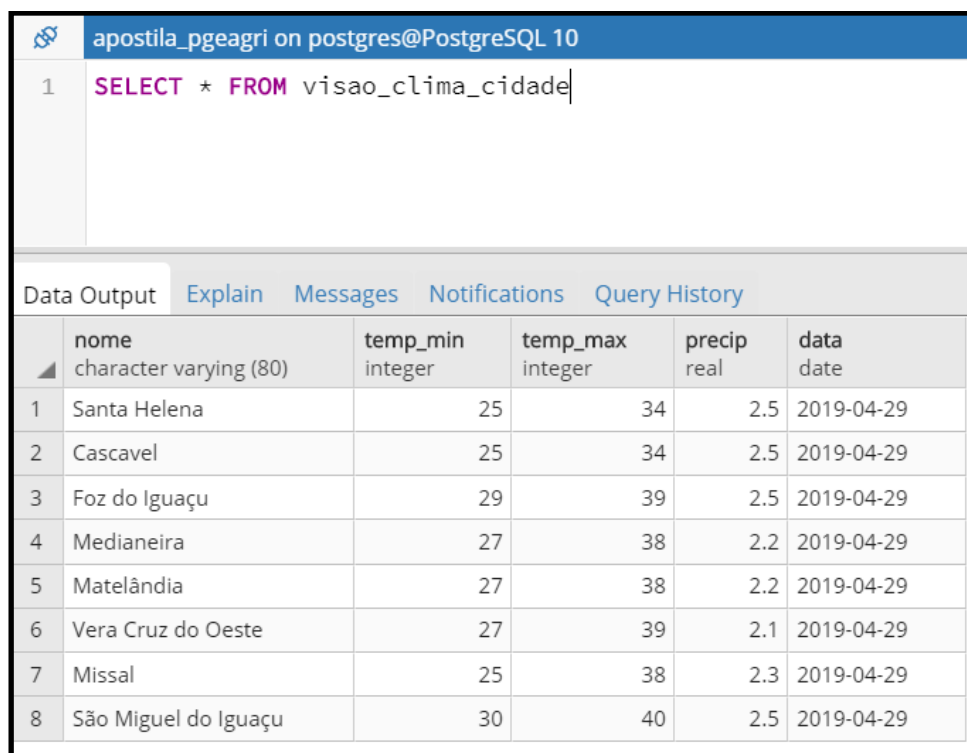


Figura 38. Criação de uma *View* (Visão).

A visão criada pode ser visualizada hierarquicamente dentro do *software PgAdmin*, dentro do nó “Views”, conforme visto na Figura x. Agora, com a visão criada, é possível realizar uma consulta SQL utilizando seu nome como referência (Figura 39).



The screenshot shows the PgAdmin interface with a SQL query executed in a view named 'visao_clima_cidade'. The query is 'SELECT * FROM visao_clima_cidade'. The results are displayed in a table with 6 columns: 'nome', 'temp_min', 'temp_max', 'precip', and 'data'. The table contains 8 rows of data representing weather information for different locations on 2019-04-29.

	nome character varying (80)	temp_min integer	temp_max integer	precip real	data date
1	Santa Helena	25	34	2.5	2019-04-29
2	Cascavel	25	34	2.5	2019-04-29
3	Foz do Iguaçu	29	39	2.5	2019-04-29
4	Medianeira	27	38	2.2	2019-04-29
5	Matelândia	27	38	2.2	2019-04-29
6	Vera Cruz do Oeste	27	39	2.1	2019-04-29
7	Missal	25	38	2.3	2019-04-29
8	São Miguel do Iguaçu	30	40	2.5	2019-04-29

Figura 39. Consulta SQL em uma View (Visão).

5. POSTGIS

O *PostGIS* é uma extensão espacial gratuita e de código fonte livre, do *PostgreSQL*. Adiciona um suporte para objetos espaciais às bases de dados relacionais manipuladas pelo *PostgreSQL*, sendo possível armazenar dados do tipo geográfico/espacial em uma base de dados relacional (POSTGIS, 2019).

5.1 CRIANDO UM AMBIENTE DE ANÁLISE ESPACIAL

Antes de se iniciar um trabalho ou projeto que caracterize o uso e análise de dados espaciais, é necessário a configuração de um ambiente mínimo para a realização do mesmo. É necessário tecnologias para armazenar dados, de maneira organizada (SGBD), soluções para processar e visualizar de maneira fácil e intuitiva a informação. Alguns recursos geralmente utilizados são:

- **Dados:** geralmente se trabalha com arquivos do tipo planilhas, arquivos csv e arquivos espaciais, nos formatos *kml*, *shapefile*, *mapinfo*, dentre outros.
- **Servidores de mapa:** *OpenStreetMap*, *GoogleMaps*, *Bing Maps*
- **SGBD:** *PostgreSQL* e *PostGIS*
- **Processamento de dados:** *QGIS*, *PostgreSQL* e *PostGIS*
- **Visualização de informação:** Sistemas de Informação Geográficas (SIG), um exemplo seria o *Software QuantumGIS* (QGIS)

Nesta apostila será trabalhado com arquivos *csv* e *shapefile*. À partir destes arquivos serão carregados seus dados espaciais para o *PostgreSQL/PostGIS*.

5.2 DADO ESPACIAL

O que diferencia uma informação espacial de uma informação normal, é a preocupação com sua localização no mundo, ou seja, no espaço (DRUCK et al., 2004). Alguns detalhes são importantes e levados em consideração em uma informação espacial. Imagine dados espaciais representados em uma imagem aérea, algumas informações relevantes devem ser consideradas, como: data; posição (latitude e longitude); elevação e a altitude do ponto de visão.

5.3 FORMATO RASTER X VETORIAL

O armazenamento de imagens com informações espaciais pode ser armazenado em dois formatos: raster (matricial) e vetorial (DRUCK et al., 2004).

Raster: a representação é feita através de uma matriz (linhas e colunas), onde cada celular (posição na matriz) corresponde a um atributo analisado e pode ser localizado entre o cruzamento das linhas e colunas (MOREIRA, 2011).

Vetorial: possui a localização e os atributos gráficos de cada objeto representado por, pelo menos, um par de coordenadas. As entidades podem ser apresentadas na forma de pontos, linhas (arcos e elementos lineares similares) e polígonos (áreas) (MOREIRA, 2011).

5.4 SISTEMA DE COORDENADAS

De forma geral, é possível afirmar que um sistema de coordenadas é uma ferramenta matemática utilizada para localizar um objeto no espaço. O sistema de coordenadas mais conhecido é o cartesiano. Neste sistema as distâncias são medidas em graus (distância angular), a longitude é a distância, em graus, de um ponto o meridiano de Greenwich, a latitude é a distância, em graus, de um ponto até a Linha do Equador (FERREIRA, 2019).

O Sistema Universal Transverso de Mercator (UTM) é um sistema de coordenadas baseado no plano cartesiano (com eixo x e y) e usa o metro (m) como unidade de medida para medir e determinar a posição de um objeto (Figura 40). Diferente de coordenadas geodésicas, o sistema UTM não acompanha a curvatura da Terra, sendo assim suas coordenadas são chamadas de coordenadas planas. No UTM, o mundo é dividido em 60 fusos, onde cada um se estende por 6 graus de longitude. Cada fuso é gerado à partir de uma rotação do cilindro de forma que o meridiano de tangência divide o fuso em duas partes iguais de 3 graus de amplitude (FERREIRA, 2019).

É importante levar em consideração que cada mapa é feito à partir de um sistemas de coordenadas, onde o mesmo é armazenado em cada arquivo gerado. Quando for utilizar um mapa é importante saber em qual sistema de coordenadas ele foi gerado.

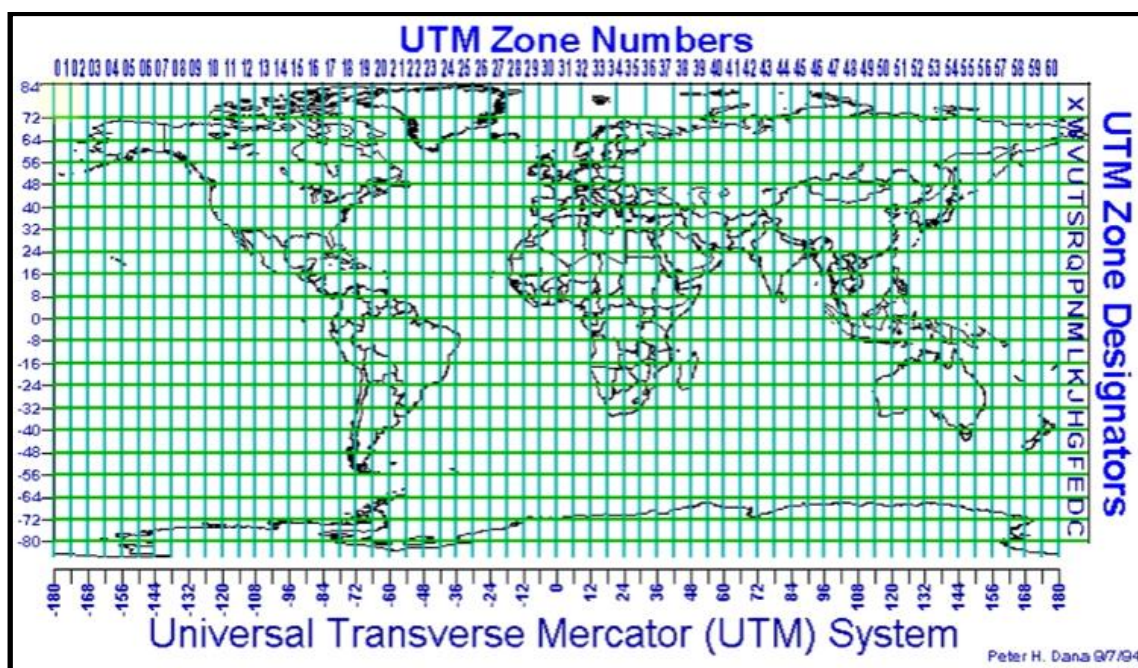


Figura 40. Sistema Universal Transverso de Mercator (UTM).

Fonte: Adaptado de Ferreira, 2019.

5.5 ARQUIVO DE DADOS ESPACIAIS

Arquivos de dados espaciais são diferentes de mapas e imagens raster. Dados podem ser armazenados em SGBDs ou arquivos avulsos. Ferramentas SIG podem processar estes tipos de dados, por exemplos o QGIS. O QGIS pode pegar um arquivo de dados espaciais (ou informações armazenadas em banco de dados) processá-lo e renderizar uma imagem. Esta imagem gerada pode ser arquivo vetorial ou raster (QUEIROZ; FERREIRA, 2006).

Um arquivo de dados espaciais ou informações geográficas recuperadas de SGBDs possuem em sua composição 2 componentes: Informações cartográficas (sistema de coordenadas utilizado); e Objetos Geográficos (conjunto de coordenadas e atributos).

Dentre os arquivos vetoriais, os formatos mais comuns de serem utilizados são: Shapefile, Mapinfo e KML. Já no formato raster, é comum de se utilizar arquivos do tipo do tipo GeoTiff. Todos estes tipos de formatos de arquivos espaciais podem ser persistidos (gravados) em *PostgreSQL/PostGIS*. Também é possível trabalhar com servidores de mapas (*OGC*, *WCS*, *WMS* e *WFS*). Estes são utilizados para facilitar o compartilhamento e aquisição de dados geográficos, visto que este, principalmente no Brasil, é um grande problema.

A *OGC (Open Geospatial Consortium)* foi criada para padronizar os formatos de dados geográficos para serem manipulados e compartilhados. Dentre eles, existem 3 tipos de criados pela *OGC* que são amplamente utilizados: *Web Feature Service (WFS)*; *Web Map Service (WMS)*; e *Web Coverage Service (WCS)* (QUEIROZ; FERREIRA, 2006).

Com o *WFS* você pode recuperar o arquivo de dados para ser trabalhado, via requisição na Internet (Figura 41).

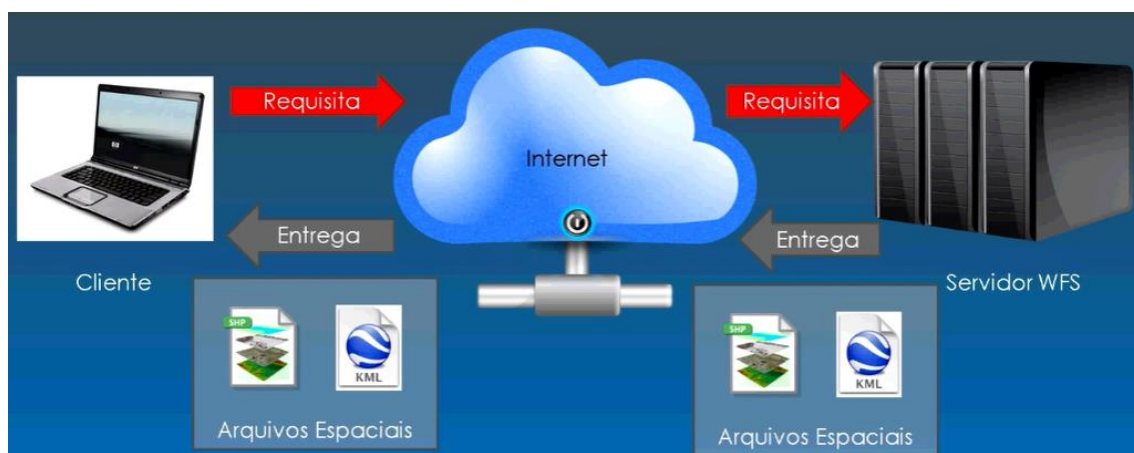


Figura 41. Requisição de arquivos espaciais via internet (*WFS*).

Fonte: Adaptado de Ferreira, 2019.

O *WMS* não trabalha com arquivo de dado espacial, ele trabalha diretamente com o mapa de dados espacial (Figura 42).

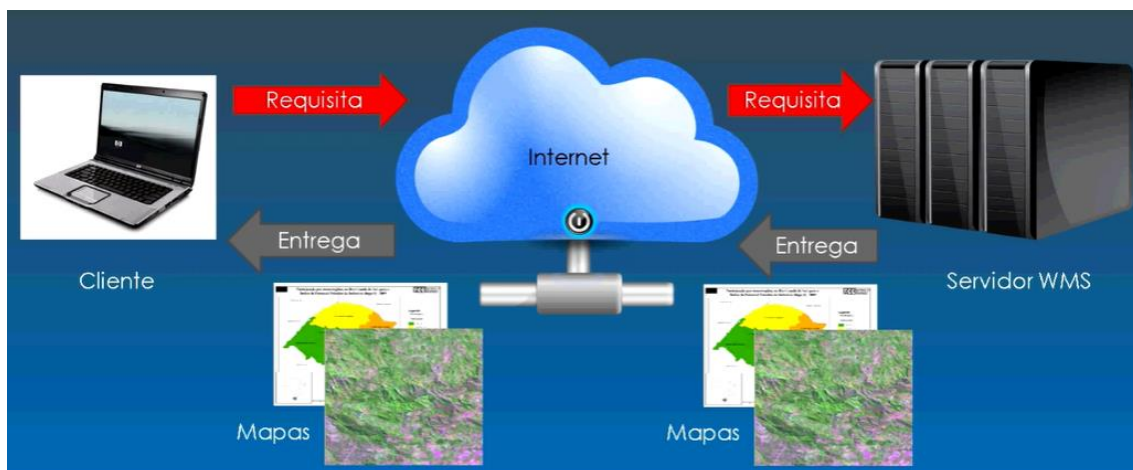


Figura 42. Requisição de mapas via internet (WMS).

Fonte: Adaptado de Ferreira, 2019.

O WCS tem um conceito um pouco mais interessante. Ele retorna um catalogo dos arquivos espaciais disponíveis, similar as “páginas amarelas” de listas telefônicas (Figura 43).

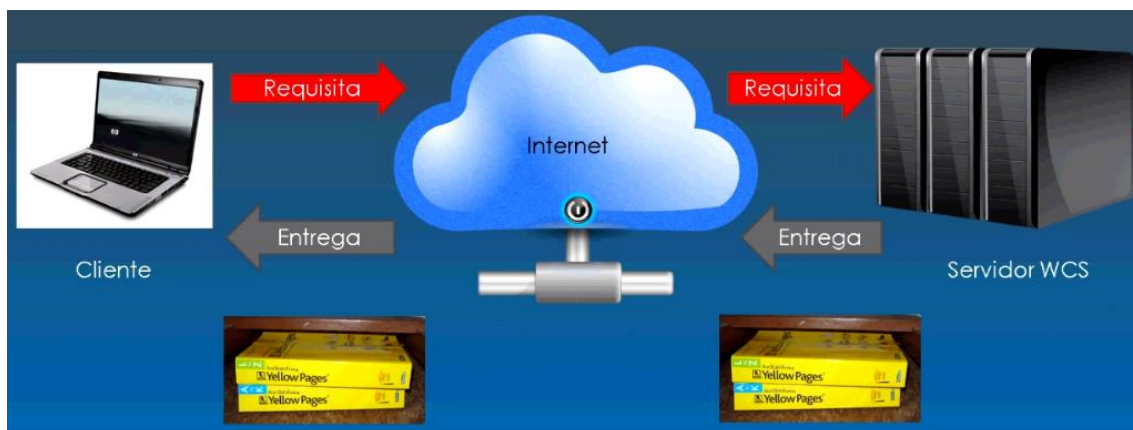


Figura 43. Requisição de um catalogo de arquivos espaciais via internet (WCS).

Fonte: Adaptado de Ferreira, 2019.

5.6 CONCEITOS TOPOLÓGICOS

A topologia é um conceito importante utilizado na lógica de consultas espaciais, como pode ser visto na Figura 44, o conceito de “fora”, “toca” e “dentro” são utilizados em consultas, tendo em vista que seu resultado será verdadeiro ou falso, caso você pergunte se o círculo está “fora”, ou se o círculo “toca”, ou se o círculo está “dentro” do retângulo (DRUCK et al., 2004).

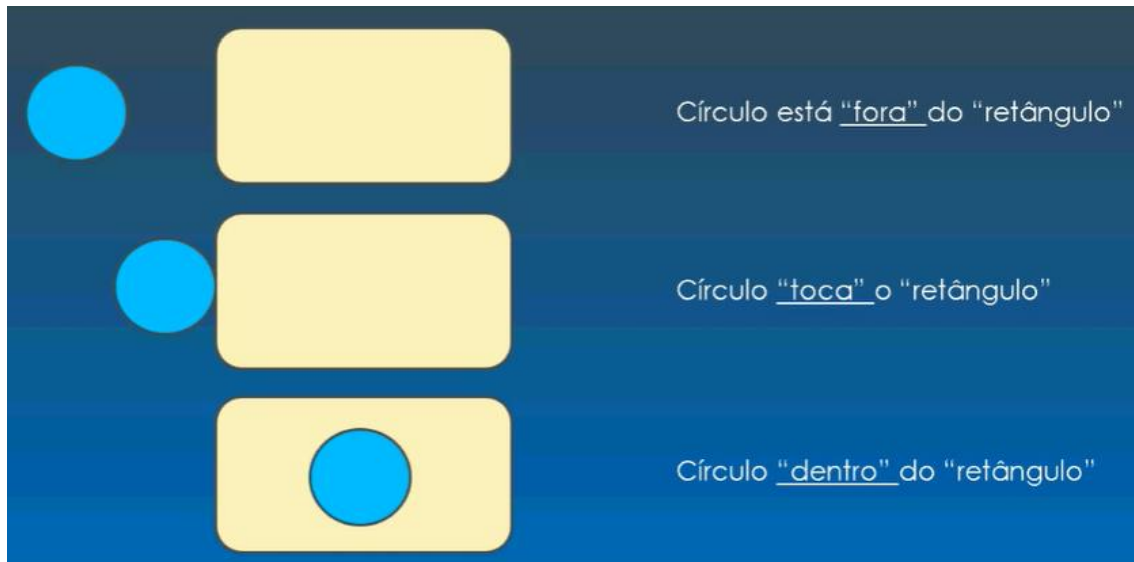


Figura 44. Topologia em consultas espaciais.

Fonte: Adaptado de Ferreira, 2019.

Em dados geográficos, esta mesma lógica pode ser utilizada em consultas espaciais. Por exemplo: é possível verificar se feições geográficas estão contidas umas em outras, se elas se cruzam ou se estão fora.

5.7 CONSULTAS ESPACIAIS

Consultas espaciais são consultas SQL, que além de elementos alfanuméricos, elas trabalham com conceitos topológicos, como por exemplo dentro, fora e toca. Imagina que seja necessário recuperar todos os nomes dos estados que cruzam o rio Amazonas. A Figura 45 apresenta a consulta SQL espacial necessária para atender esta questão.

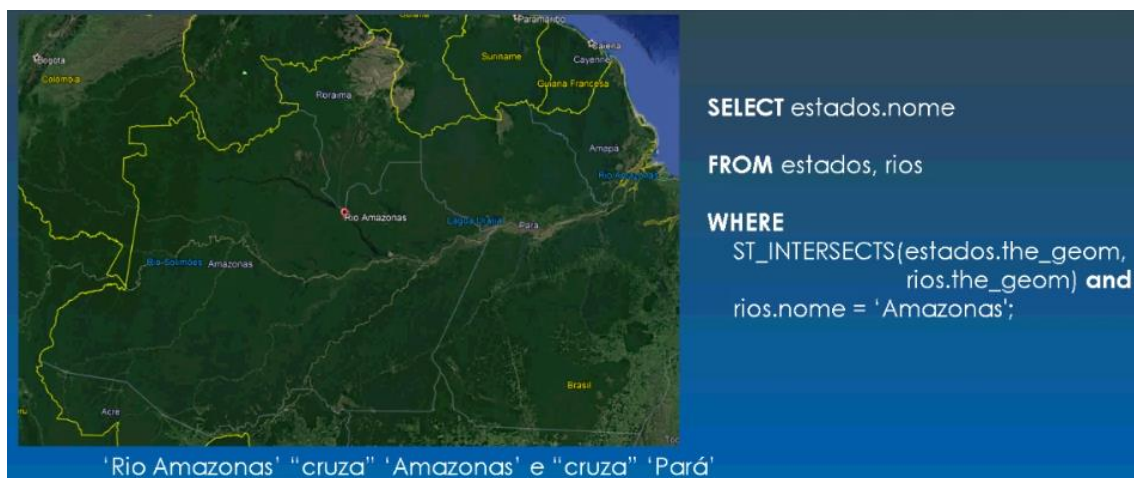


Figura 45. Exemplo de consulta espacial.

Fonte: Adaptado de Ferreira, 2019.

5.8 CRIANDO BASES DE DADOS ESPACIAIS

A criação de uma base de dados espacial dentro do SGBD *Postgres* é feita em paralelo com *PostgreSQL* e sua extensão espacial, o *PostGIS*. A criação de uma base de dados espacial apresenta uma série de vantagens, tais como: Segurança; Gestão da informação; Velocidade no processamento; Cruzamento de dados; e Escalabilidade na capacidade de acessos. Não é interessante que arquivos espaciais fiquem isolados em computadores.

Para a criação da base de dados espacial será utilizada a ferramenta *PgAdmin* 4. A criação de uma base de dados no *PostgreSQL* já foi apresentada na seção 2.5.1. O procedimento aqui será o mesmo, porém iremos adicionar a extensão geográfica a base de dados criada. Para isso em uma *Query Tool*, escreva o seguinte comando:

```
CREATE EXTENSION postgis;
```

Este comando informa ao banco de dados que a base de dados criada obterá suporte para realizar comandos, processamentos e suporte a dados geográficos. A Figura 46 apresenta a extensão adicionada e visualizada na barra de ferramentas do *PgAdmin*. O banco de dados “apostila_pgeagri_espacial” agora possui a extensão *postgis* adicionada aos seus recursos.

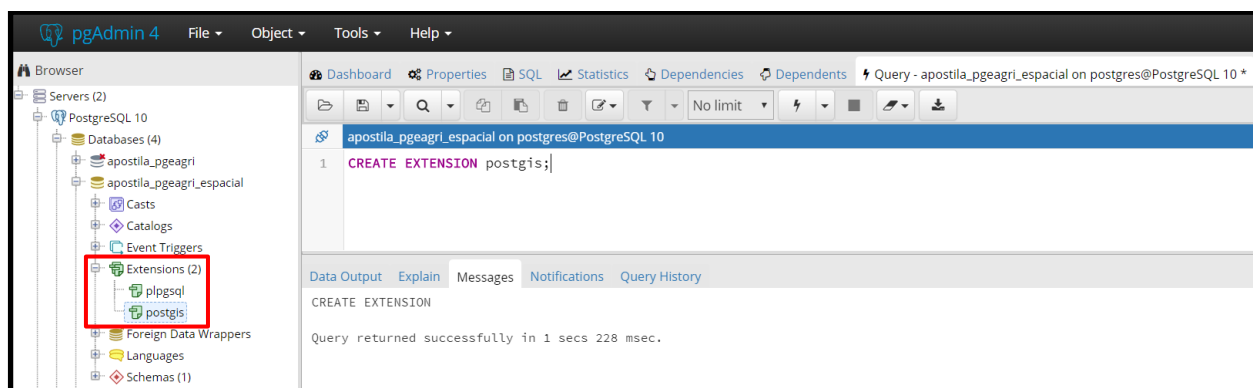


Figura 46. Banco de dados com extensão espacial habilitada.

Dentro do nó “*Types*” dentro do nó “*public*” é possível verificar, que agora foram disponibilizados vários tipos geográficos para serem trabalhos dentro do banco de dados ‘*apostila_pgeagri_espacial*’ (Figura 47).

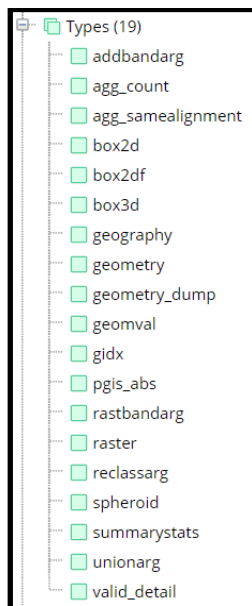


Figura 47. Tipos de dados espaciais disponibilizados pelo *PostgreSQL/PostGIS*.

5.9 IMPORTANDO DADOS ESPACIAIS PARA O *POSTGRESQL/POSTGIS*

Com a importação de um arquivo espacial para o *PostgreSQL/PostGIS*, é possível aproveitar todas as vantagens associadas ao SGBD e disponíveis para serem aplicadas em pesquisas espaciais.

Para importar arquivos do tipo *shapefile* para o *PostgreSQL/PostGIS*, é utilizado uma ferramenta disponibilizada pelo próprio SGBD chamada *PostGIS Shapefile Import/Export Manager*.

Nesta ferramenta, o primeiro passo é realizar a conexão com uma base de dados, informando seu usuário e senha, conforme Figura 48. Foi utilizada a base de dados “apostila_pgeagri_espacial”. Na opção “*Log Window*” é apresentada uma mensagem de validação da conexão com a base de dados.

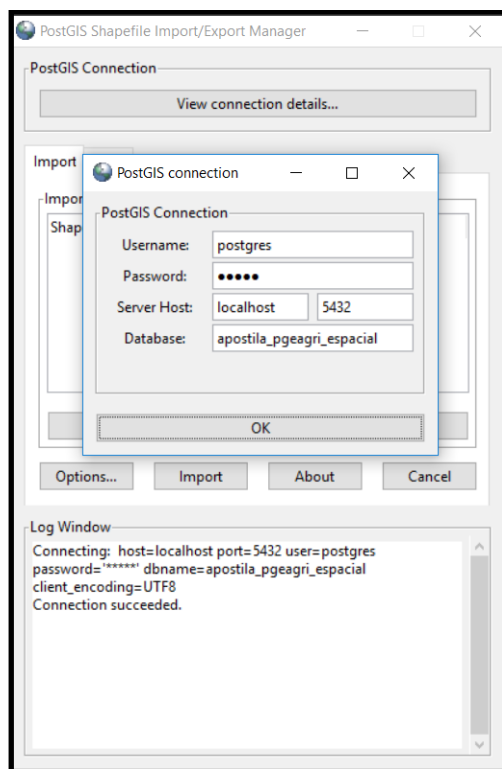


Figura 48. Conexão na ferramenta *PostGIS Shapefile Import/Export Manager*.

Com a conexão validada, o próximo passo é clicar no botão “*Add File*” para adicionar o *shapefile* que terá suas informações adicionadas ao banco de dados (Figura 49).

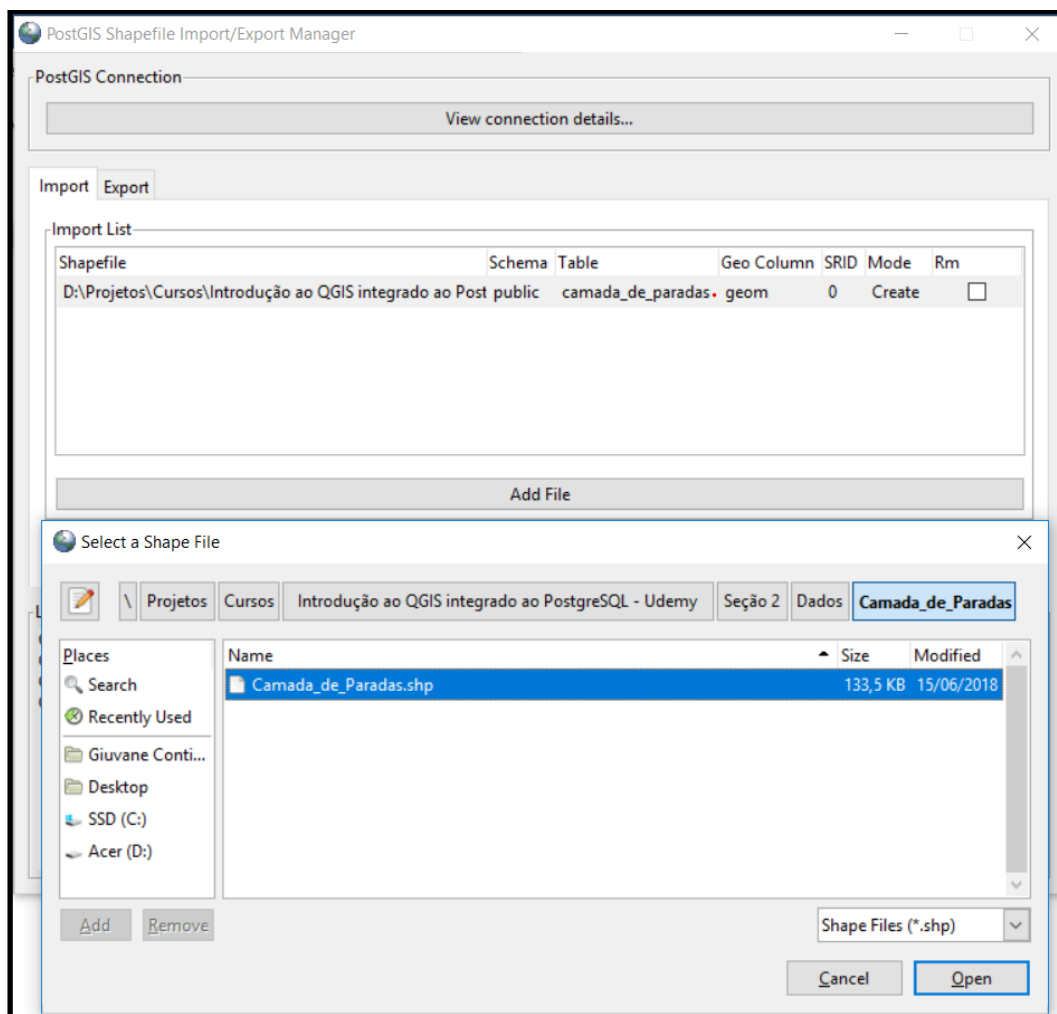


Figura 49. Importando *shapefile* para banco de dados *PostgreSQL/PostGIS*.

Na Figura 49, o *shapefile* selecionado será apresentado na listagem, a opção *Shapefile* indica a pasta de onde ele será carregado. O *Schema* indica em qual *schema* do banco de dados *PostgreSQL* ele será importado. A opção *Table* indica qual é o nome de tabela que será criado para este *shapefile* no *PostgreSQL*. *Geo Column* indica o tipo de coluna que será gerado. *SRID* é uma opção muito importante, nesta opção será inserido o código de coordenadas de projeção espacial. Mapas *shapefile* normalmente possuem um arquivo de extensão *.prj* (projeção), onde a projeção utilizada está armazenada. Baseado nesta projeção é possível recuperar este código e inserir no campo *SRID*, vários sites disponibilizam esta informação. O arquivo utilizado neste exemplo é a SIRGAS 2000 / UTM zone 23S, o código *SRID* para este é o 31983.

Com todas informações inseridas e editadas, basta clicar em *Import*. Para não ocorrer falhar, é importante verificar se no caminho onde se encontra o arquivo não existem letras com acentos. Outro detalhe é a codificação utilizada pelo arquivo, o padrão utilizado pelo programa é o UTF-8, porém alguns arquivos de dados irão exigir

outro tipo de codificação, para alterar o tipo de codificação, basta clicar em “Options...”. A Figura 50 apresenta a importação ocorrendo com sucesso.

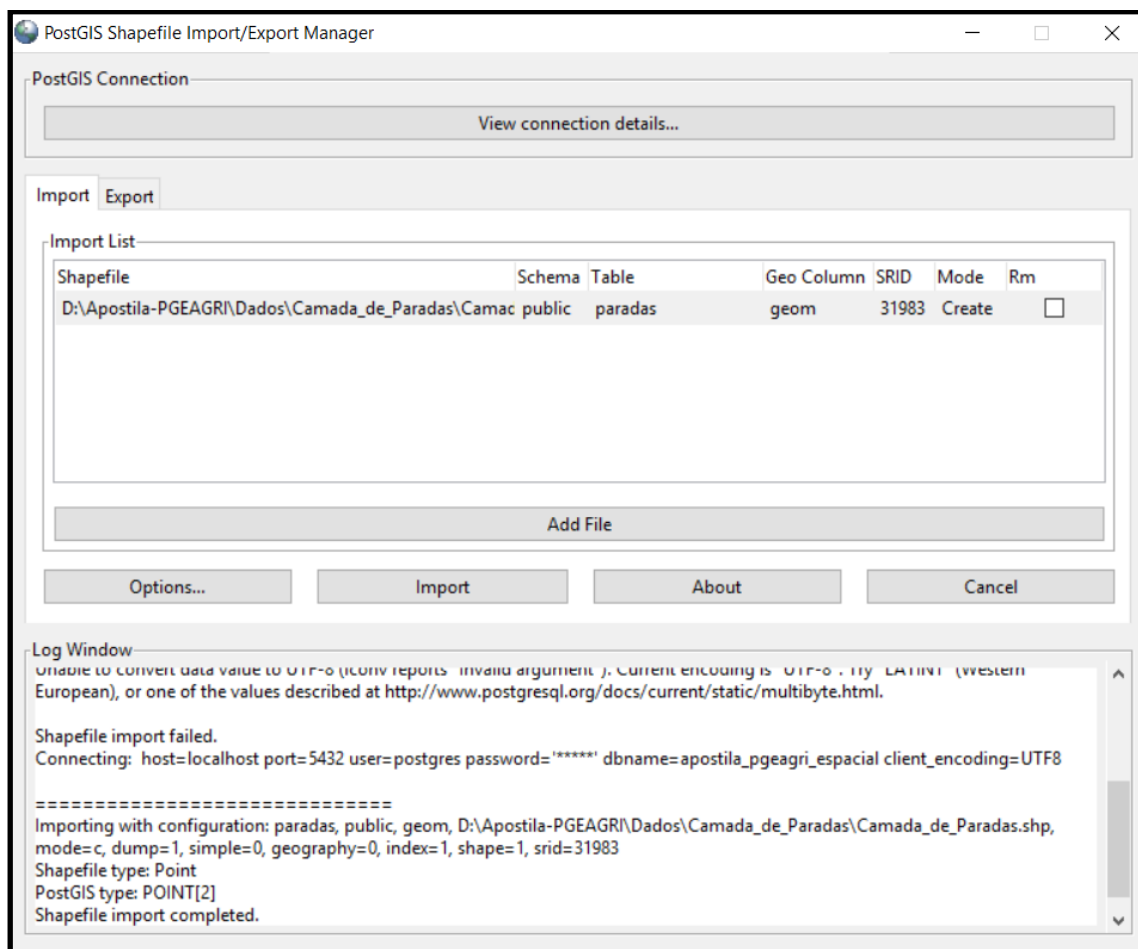


Figura 50. Importação do *shapefile* para o banco de dados realizado com sucesso.

Agora a tabela “paradas” já está disponível na base de dados “apostila_pgeagri_especial”, conforme apresentado na Figura 51.

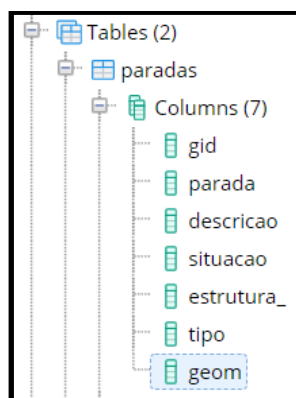


Figura 51. tabela 'paradas' com dados do *shapefile* já no banco de dados.

5.10 IMPORTANDO PLANILHAS PARA O POSTGRESQL/POSTGIS

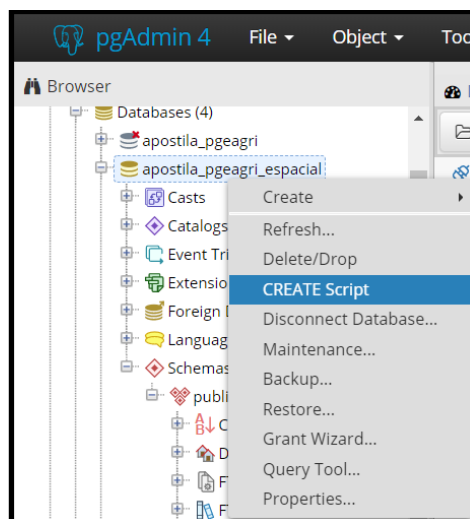


Figura 53. Opção para criação de novo script.

No *script* será criada uma tabela chamada horários, com os mesmos campos contidos na planilha (Figura 54).

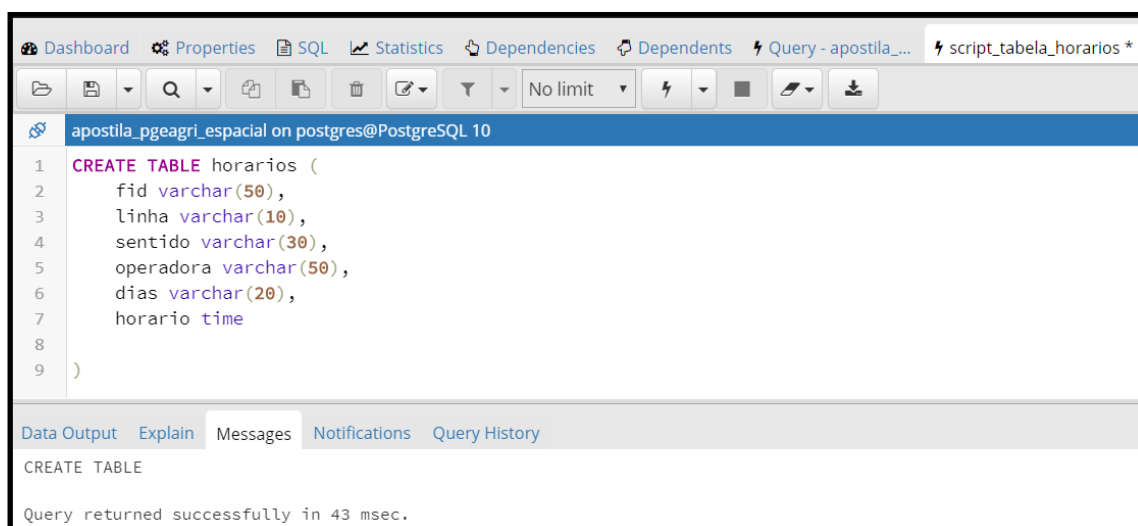


Figura 54. Tabela horários, utilizada na migração da planilha.

Com a tabela criada, agora vamos fazer a importação dos dados contidos na planilha, para isto clique com o botão direito na tabela criada (horários), selecione a opção “*Import/Export*” e selecione a planilha com os dados. Feito isto, basta selecionar e inserir as informações referentes a importação (Figura 55).

Figura 55. Opções para realizar a migração da planilha

A opção “*Header*” habilitada indica que a planilha possui uma linha de cabeçalho e a opção “*Delimiter*” indica qual é o caractere que separa cada dado nas linhas que irão para a tabela criada. Basta clicar em OK e os dados serão importados para a tabela *horários*. Para testar, realize uma consulta básica na tabela *horários* e verifique quais informações são recuperadas (Figura 56).

<

Figura 56. Consulta SQL para testar migração.

5.11 QUERIES (CONSULTAS) ESPACIAIS NO POSTGRESQL/POSTGIS

Em consultas espaciais, é possível utilizar vários operadores em cláusulas *WHERE* aos quais utilizam a geometria de cada tabela criada na base de dados. Por exemplo, imagine que dois arquivos *shapefiles* foram importados para uma base de

dados no *PostgreSQL/PostGIS*, um possui todas as paradas de ônibus (pontos) de Brasília e foi importado para a tabela “parada”. O outro possui os setores censitários (polígonos) de Brasília, ao qual foi importado para a tabela “setores censitários”.

Agora imagine que seja necessário consultar estas paradas de ônibus (tabela parada) e unir aos setores censitários (tabela setorescensitarios). Para realizar essa junção, levando em consideração feições geográficas, seria utilizado a consulta espacial apresentada na Figura 57. Note que a função geográfica *ST_Within* verifique a geometria das duas tabelas e retorna apenas as paradas que apresentam geometria dentro do setor censitário.

The screenshot shows a PostgreSQL query editor interface. The query is as follows:

```

1 SELECT s.gid, s.cd_geocodi, p.gid, p.parada
2 FROM paradas as p, setorescensitarios as s
3 WHERE ST_Within(p.geom, s.geom)
4 ORDER BY s.gid, s.cd_geocodi, p.gid, p.parada

```

Below the query, the results are displayed in a table with the following columns: *gid* (integer), *cd_geocodi* (character varying (20)), *gid* (integer), and *parada* (character varying (254)).

	gid integer	cd_geocodi character varying (20)	gid integer	parada character varying (254)
1	16	530010805250000	2098	4359
2	19	530010805250000	2099	4360
3	21	530010805090000	79	1109
4	22	530010805090000	112	1110
5	28	530010805070043	490	1889
6	30	530010805070045	450	1864
7	30	530010805070045	454	1868
8	31	530010805070046	455	1869
9	32	530010805070047	574	1944
10	33	530010805070048	477	1870
11	34	530010805070049	449	1863
12	38	530010805070053	452	1866
13	40	530010805080032	3674	5349

Figura 57. Consulta SQL espacial utilizando a função ***ST_Within***.

Assim como este exemplo apresentado na Figura 55, o *PostGIS* disponibiliza um manual completo com todas operações disponibilizadas para serem aplicadas unindo tabelas com informações geográficas.

Para o próximo exemplo, imagine duas tabelas, também importadas à partir de arquivos *shapefile*, escolas e linhas, onde a tabela escolas representa todas escolas cadastradas em Brasília e linhas representa as linhas de ônibus cadastradas.

Será realizada uma consulta onde, se deseja saber quais linhas passar a, pelo menos, 60 metros da escola. A consulta é apresentada na Figura 58. Note que neste exemplo se utilizou a função *ST_DWithin*.

Dashboard Properties SQL Statistics Dependencies Dependents consulta_paradas consulta_escolas

apostila_pgeagri_especial on postgres@PostgreSQL 10

```

1 SELECT e.cod_seec, e.nome, l.linha
2 FROM escolas as e, linhas as l
3 WHERE ST_Dwithin(e.geom, l.geom, 60)
4 ORDER BY e.cod_seec, e.nome, l.linha

```

Data Output Explain Messages Notifications Query History

	cod_seec numeric	nome character varying (254)	linha character varying (254)
1	3000846.0000	CEF 01 DE BRASILIA	0.113
2	3000846.0000	CEF 01 DE BRASILIA	0.150
3	3000846.0000	CEF 01 DE BRASILIA	0.153
4	3000846.0000	CEF 01 DE BRASILIA	0.160
5	3000846.0000	CEF 01 DE BRASILIA	0.172
6	3000846.0000	CEF 01 DE BRASILIA	0.174
7	3000846.0000	CEF 01 DE BRASILIA	0.185
8	3000846.0000	CEF 01 DE BRASILIA	0.186
9	3000846.0000	CEF 01 DE BRASILIA	0.224
10	3000846.0000	CEF 01 DE BRASILIA	0.300
11	3000846.0000	CEF 01 DE BRASILIA	0.302
12	3000846.0000	CEF 01 DE BRASILIA	0.306
13	3000846.0000	CEF 01 DE BRASILIA	0.310
14	3000846.0000	CEF 01 DE BRASILIA	0.314

Figura 58. Consulta SQL espacial utilizando a função **ST_DWithin**.

6. REFERÊNCIAS

DRUCK, S.; CARVALHO, M.S.; CÂMARA, G.; MONTEIRO, A.V.M. **Análise Espacial de Dados Geográficos**. Brasília, EMBRAPA, 2004.

FERREIRA, E. O. **Ambiente de Geoprocessamento**. 2019. Disponível em: <<https://sites.google.com/view/simulacaomobilitade/p%C3%A1gina-inicial/cursos/>>.

MOREIRA, M. A. **Fundamentos do Sensoriamento Remoto e Metodologias de Aplicação**. 4ª edição ed. Viçosa, MG: Editora UFV, 2011.

POSTGIS. **PostGIS 2.5.3 dev Manual**. The PostGIS Development Group. Disponível em: <<https://postgis.net/docs/>>.

POSTGRES. **PostgreSQL 11.4 Documentation**. The PostgreSQL Global Development Group. 2019. Disponível em: <<https://www.postgresql.org/docs/11/preface.html>>.

QUEIROZ, G. R.; FERREIRA, K. R. **Tutorial sobre Banco de Dados Geográficos**. GeoBrasil: Instituto Nacional de Pesquisas Espaciais (INPE). 2006.

STONES, R.; MATTHEW, N. **Beginning databases with PostgreSQL: from novice to professional**. [S.l.]: Apress, 2006.