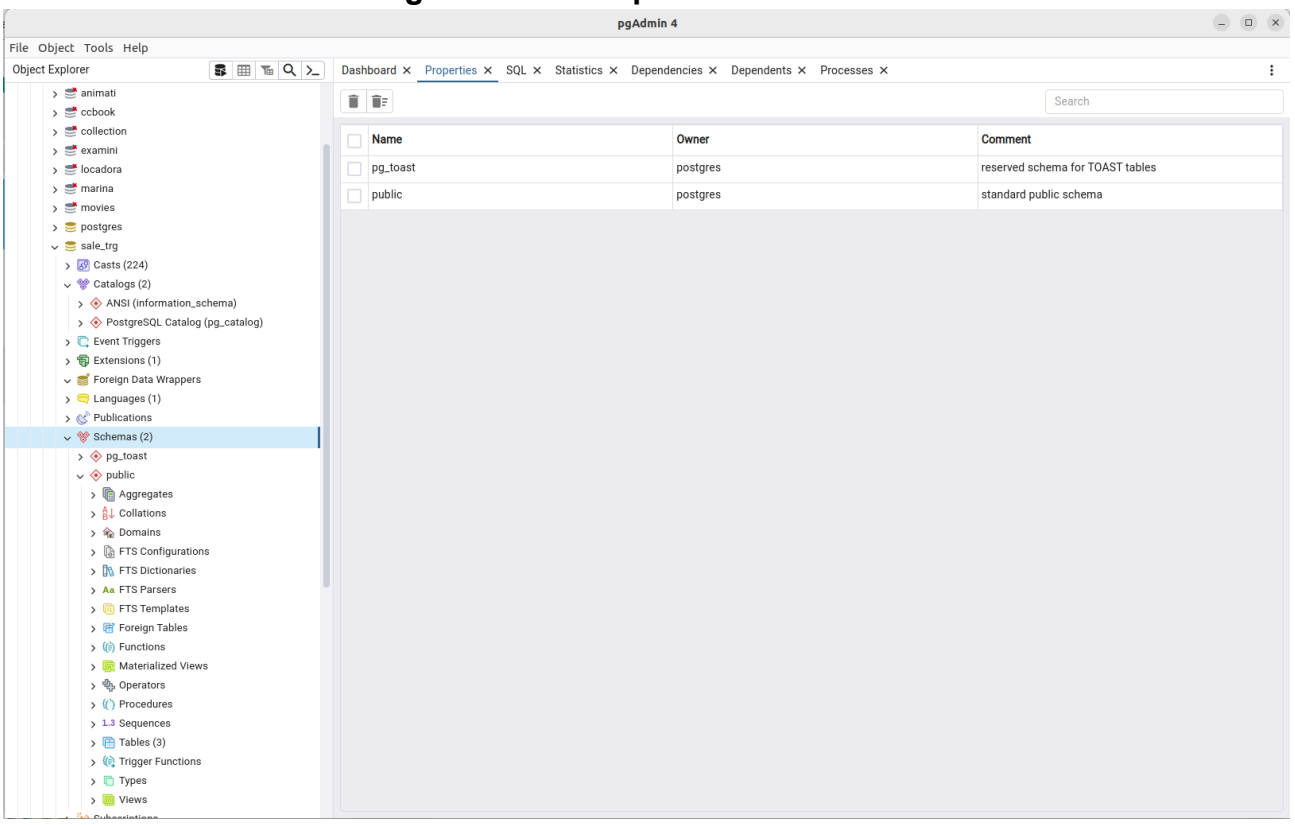
Objetos do PostgreSQL

O Sistema Gerenciador de Banco de Dados PostgreSQL possui várias estratégias para implementar a sua forma de gerenciar os dados. Podemos definir essas estratégias como estruturas de controle onde os dados estão armazenados. As estruturas utilizadas devem ser persistidas de alguma forma para que o SGBD possa recuperá-las quando necessário. A Figura 1 apresenta o primeiro nível de hierarquia que o PostgreSQL organiza seus dados. O nível mais alto da estrutura são os servidores (servidores locais são definidos como **localhost**). Dentro de servidores podemos ter (*i*) banco de dados (na Figura 1, vemos o banco de dados **sale_trg**); (*ii*) Tablespaces (na Figura 2 aparacem **pg_default** e **pg_global**); (*iii*) papeis de grupo; e (*iv*) papeis de conexão (login) (a Figura 2 apresenta alguns papeis com **postgres** (mais abaixo) e **denio** - esses papeis representam os usuários criados para acessarem os banco de dados).
Perceba, na Figura 2, que existe um servidor (`Medilaudo` com as configurações de conexão como host=192.168.1.65 port=5432). Isso indica que é possível se conectar em um servidor remoto. Para que uma instalação do PostgreSQL possa receber requisições externas é necessário alterar no arquivo postgres.conf a linha `listen_addresses='localhost'` para `listen_addresses='*'`, que indica que anteriormente o servidor "ouvia" (listen) apenas o servidor local e agora, "ouve" qualquer servidor ("*"). É possível especificar uma lista endereços IPs válidos para serem ouvidos.

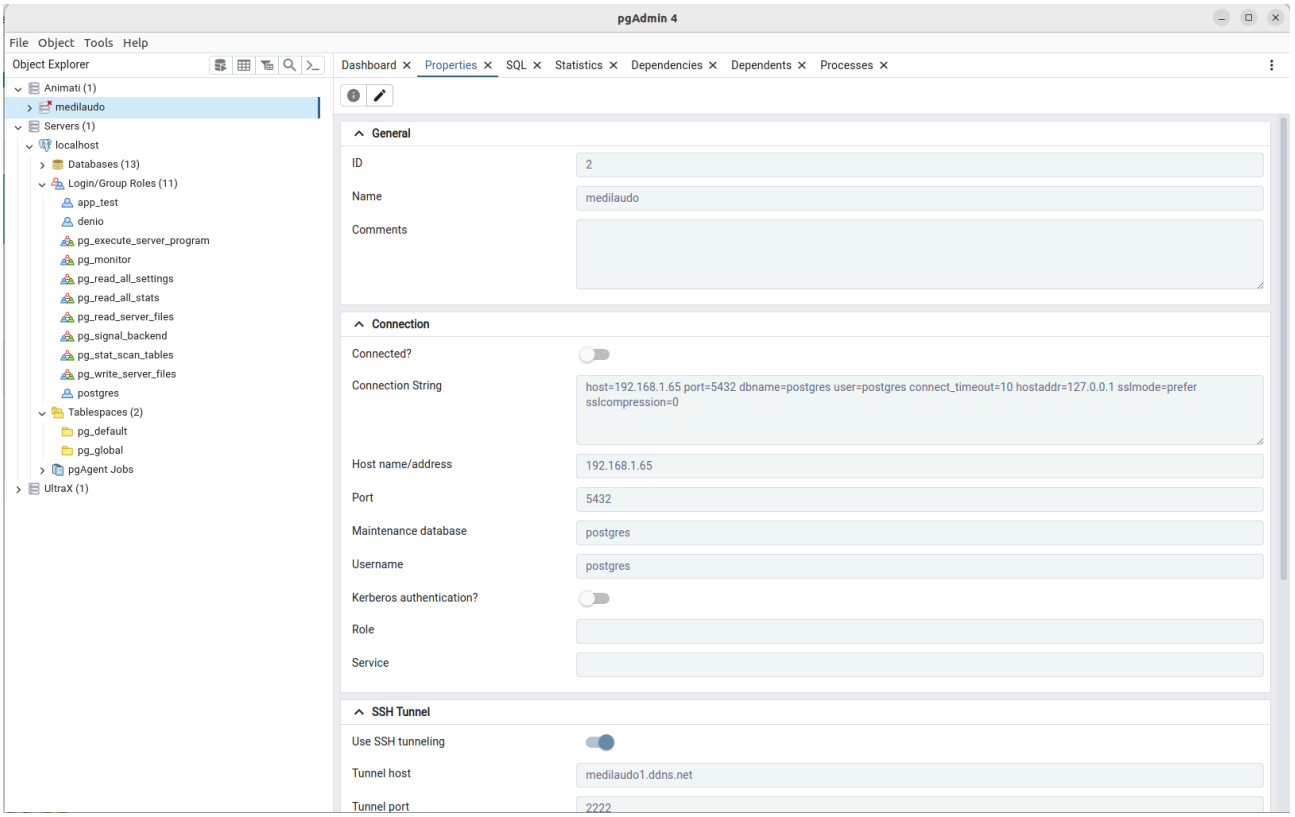
Figura 1 - Hierarquias no PGAdmin



A descrição acima é particular para o SGBD PostgreSQL mas a maioria dos SGBD seguem o mesmo raciocínio e cabe ao interessado em estudar outros SGBDs como os objetos são mantidos.
A Figura 1 apresenta a organização dos objetos do PostgreSQL (extraída do aplicativo PGAdmin 4). O objeto que nos interessa é o *Database* que na figura está selecionado o **sale_trg**. Dentro de um banco de dados temos: (*i*) catalogs (apresenta os catálogos criados no banco de dados – catálogos representam o dicionário de dados), na figura

aparecem os dois padrões do PostgreSQL: ANSI e PostgreSQL (pg_catalog) – este é o padrão do SGBD, o objeto pg_catalog é um esquema especial que organiza o dicionário de dados do PostgreSQL (esquemas serão vistos mais adiante); (*ii*) Casts (representam tipos criados pelo usuário – não estudaremos); (*iii*) Extensions (extensões do SGBD, o padrão deve ser a extensão para a linguagem plpgsql – linguagem de programação do PostgresQL); (*iv*) Foreign Data Wrappers (utilizando para acessar objetos remotos de outros tipos de SGBD – não estudaremos); (*v*) Languages (define as linguagens de programação que podem ser utilizadas pelo SGBD, a padrão é a plpgsql – que também aparece no grupo Extensions); e (*vi*) Schemas (objeto importante do PostgreSQL e será visto com mais detalhes – na Figura 1 existem dois esquemas: pg_toast e public – padrão).

Figura 2 - Tablespaces



O PostgreSQL se apoia em esquemas para organizar os dados do usuários (tabelas, visões, índices, etc.). Um banco de dados pode ter vários esquemas e esses são independentes entre si, ou seja, dois esquemas diferentes podem ter dois objetos com o mesmo nome. Os usuários acessam os esquemas que foram autorizados. O SGBD sempre terá um super usuário (o padrão do PostgreSQL é o postgres) que distribui e revoga privilégios para os outros usuários. A seguir são apresentados alguns dos objetos que estudaremos ao tratarmos de SQL Avançado.

Os estruturas utilizadas para armazenar os dados propriamente ditos podem ser classificadas em:

Tabelas: armazenam os dados das aplicações definidas e utilizadas pelos usuários. As tabelas são as estruturas mais importantes na visão da aplicação do banco de dados. As tabelas estão associadas a um esquema que está associado a um banco de dados.

Stored Procedures (procedimentos armazenados): a maioria dos SGBDs oferecem uma linguagem de programação embutida que permite implementar regras de negócio no próprio banco de dados. O nome comum dado para a linguagem é PL (procedure language), assim, cada SGBD nomeia sua linguagem geralmente utilizando a sigla PL: Oracle é PL-SQL, PostgreSQL é PGPLSQL e assim por diante. Os programas feitos em PL são armazenados no banco de dados e podem ser executados pelas aplicações. Existem dois tipos básicos de stored procedures (SP):

Tradicionais: são as SPs que atuam como funções ou procedimentos. Funcionam similar a uma função em C, o programador define se o procedimento retorna ou não um valor. Sintaxe genérica:

```
create [or replace] function <name> ([<parameters>])
   returns <type> as
$$ <block> $$ language plpgsql;
```

Onde `<block>` é definido como:

```
[Declare
   <variables>]
Begin
   <commands>
End;
```

Triggers (gatilhos): são tipos especiais de SPs que são executadas automaticamente quando ocorrem atualizações em tabelas do banco de dados. Elas são criadas em duas partes: na primeira é criada a SP que corresponde a trigger (onde são implementadas as regras) e na segunda é criada a trigger que executara a SP criada inicialmente.

A sintaxe abaixo apresenta os passos para implementar um trigger:

```
create [or replace] function <name_TG> ([<arguments>]) returns
trigger ...

create trigger <nome>
{ before | after } {update | delete | insert}
on <table> [ for [ each ] { row | statement } ]
execute procedure <name_TG> ( [arguments] )
```

A SP criada retorna um tipo especial (**trigger**), essa SP deve aparecer na criação da trigger propriamente dita.

Usuários: para acessar o banco de dados é necessário existir usuários no SGBD. A maioria dos SGBDs trata os usuários como objetos do próprio banco e eles são armazenados e tratados de forma independente dos outros objetos. Em PostgreSQL existe o conceito de **User** e **Role** que podemos considerar como sinônimos (apesar de significarem exatamente a mesma coisa). A diferença básica é que o usuário já herda o privilégio de conexão já para o papel (role) é necessário dar esse privilégio explicitamente. Sintaxe:

```
create user <name> password '<password>' with <privileges>;
create role <role name> '<password>' with login <privileges>;
```

Esquemas (schema): são formas de organizar os objetos dentro de um banco de dados. É um objeto do PostgreSQL utilizado para organizar outros componentes do banco (tabela, visões, SP). Ao se criar um banco, o esquema **public** é associado automaticamente a este banco e todos os objetos criados ficam abaixo do

esquema **public**. Como todos os objetos são criados implicitamente no esquema público, não é necessário acessar um objeto informando o esquema, assim, `select * from public.client` é igual a `select * from client` pois quando trabalhamos dentro um esquema não é necessário informar o esquema onde o objeto se encontra. O mesmo vale para criação e exclusão de objetos em um esquema, por exemplo, `create table myschema.client (a integer)` cria a tabela **client** no esquema **myschema** no banco de dados atual.

Na realidade, o PostgreSQL procura primeiramente um esquema com o nome do usuário conectado e em seguida o público.

Comandos relacionados a esquemas:

`show search_path` : mostra o esquema atual. O resultado desse comando, quando o esquema é o padrão, é `$user,public` que indica que o primeiro esquema a ser procurado é aquele com o nome do usuário (indicado pela variável *$user*) e o segundo é o *public*.

`set search_path to <schema>`: troca o esquema atual para <schema>

`create schema <schema name>`: cria um esquema no banco de dados atual.

`create schema <schema name> authorization <user name>`: cria um esquema no banco de dados atual para o usuário *user name*.

`alter user <user name> set search_path to <schemas>`: altera o esquema padrão do usuário *user name* para aqueles listados em *schemas*. Este comando vale para a próxima conexão do usuário.

`alter database <database name>  set  search_path =<schemas>`: altera o esquema padrão do banco de dados.

Importante ressaltar que se o comando:

```
revoke all privileges on schema public from group public;
```

for executado, os usuários criados de forma padrão não terão acesso a nenhum objeto do banco já que estão associados ao esquema *public* cujo os privilégios estão associados ao grupo *public*. Neste caso então, o usuário deve ser criado em um esquema pré-definido com privilégios de acesso. Exemplo:

`postgres=#create user teste password 'teste';` -- cria usuário teste (conexão implícita)

`postgres=#\c postgres teste` -- conecta teste no banco postgres (psql)

`postgres=>\d` -- mostrará todos os objetos do schema public

Se fizermos:

`postgres=#revoke all privileges on schema public from group public;`

`postgres=#\c postgres teste` -- conecta teste no banco postgres (psql)

`postgres=>\d` -- nenhum objeto será listado

Observe que na segunda parte do exemplo, todos os privilégios padrão do postgreSQL foram revogados, assim o usuário teste consegue apenas se conectar a um banco.

Para solucionar isso imagine que existe um esquema chamado desenv:

`postgres=#create schema desenv;` -- cria esquema desenv

`postgres=#grant usage on schema desenv to teste;` -- permissão para o esquema

`postgres=#\c postgres teste` -- nenhum objeto será listado

`postgres=>\d --` continua não aparecendo os objetos

`postgres=>show search_path; --` aparece $user$, public

`postgres=> set search_path to desenv;` -- esquema padrão desenv

`postgres=>\d` -- agora aparecerá os objetos do esquema desenv

Para evitar executar o comando set todas as vezes que conectar-se como o

usuário teste, o comando `alter user teste  set search_path to desenv` fará que o esquema padrão para o usuário `teste` seja `desenv`.

Banco de dados: é o objeto maior onde as tabelas, SP, visões, entre outros, estão subordinados. Em PostgreSQL, usuários e *tablespaces* (visto mais adiante) não estão subordinados a um banco de dados. A criação de um banco de dados é bastante simples, a única restrição é o usuário criador ter privilégios para criar banco de dados.

```
create database mydb;
```
-- cria o banco mydb com parâmetros padrão

```
create database mydb tablespace myts;
```
-- myts é a tablespace padrão de mydb

```
create database mydb template dbdefault;
```
-- mydb será criado como cópia do dbdefault

```
create database mydb connection limit=5;
```
-- máximo 5 usuários conectados ao mybd

Os parâmetros podem ser misturados, assim podemos criar o banco como:

```
create database mydb tablespace myts  connection limit=5  template dbdefault;
```

Os comandos `alter database <parâmetros>` e `drop database <nome>` alteram os parâmetros de um banco existente e excluem um banco existente, respectivamente.

Tablespace: é o objeto de um SGBD que armazena todos os outros objetos dos bancos de dados. Uma *tablespace* pode ser vista como um grande repositório onde todos os objetos estão armazenados. A estratégia de utilizar *tablespaces* para armazenar objetos faz com que os SGBD tenham controle de como organizar os objetos dentro de uma *tablespace* pois a mesma pode ser tratada como um dispositivo de armazenamento próprio do SGBD. Um banco de dados é associado a uma *tablespace*, bem como uma tabela. No PostgreSQL, a *tablespace* padrão é a **pg_default**. Pode-se criar novas *tablespaces* e associar o banco de dados e/ou outros objetos (tabelas, visões) as *tablespaces* criadas.

Sintaxes:

```
create tablespace <tablespace name> [owner <user name>] <location>
```

É necessário dar permissão ao usuário **postgres** para escrever e mudar as permissões de `<location>`, utiliza-se o comando `chown postgres.postgres <location>`

A localização da tablespace deve ser uma pasta a partir da raiz que não tenha outro dono “no meio do caminho”. Por exemplo, se o usuário aluno está na pasta /home/aluno e criar a pasta /home/aluno/data e executar o comando chown postgres.postgres /home/aluno/data, o comando (dentro do ambiente postgres)

```
create tablespace mytbsp '/home/aluno/data'
```
gerará o erro de permissão negada, pois o usuário postgres não é dono do /home/aluno (e nem é aconselhável fazê-lo ser).

Privilégios: privilégios são formas de acesso aos objetos do banco dadas aos usuários do banco. O dono do objeto tem, por padrão, todos os privilégios sobre o objeto criado. Depende do objeto, tem-se os privilégios sobre o mesmo. Para tabelas temos select, update, delete, insert, rule, references, entre outros. Para banco de dados temos create. Para funções temos execute, e assim por diante. A expressão `all privileges` indica todos os privilégios possíveis para aquele objeto.

O comando que concede privilégios é o `grant`.

```
postgres=#grant select on client to teste
```
-- teste agora pode fazer select em client

```
postgres=#grant all privileges on employee to teste
```
-- teste pode fazer qualquer operação na tabela employee

```
postgres=#grant execute on function calc() to teste -- teste pode fazer executar a função calc()
```

Os privilégios podem ser dados de tal forma que o usuário que os recebe pode concedê-los para outros usuário. Para tal, basta utilizar a opção WITH GRANT OPTION no fim do comando.

```
postgres=#grant insert on employee to teste with grant option
-- teste pode conceder o privilégio de insert sobre employee para outros usuários
```

O comando que retira os privilégios é o revoke. Funciona da mesma maneira que o grant porém ao invés de utilizar `to <user>` utiliza-se `from user`.

```
postgres=#revoke all privileges on employee from teste
-- teste não tem mais privilégios sobre a tabela employee
```

Para não conceder privilégios um a um para um grupo de usuário, pode-se criar um grupo, dar privilégios para este grupo e depois adicionar usuários ao grupo.

```
create group mygroup;
grant select, insert on employee to mygroup;
alter group my group add user user1, user2, user3.
```

Nos comandos acima, o grupo mygroup foi criado, em seguida recebeu os privilégios de inserção e consulta na tabela employee. Finalmente, os usuários user1, user2 e user3 foram adicionados ao grupo mygroup e automaticamente receberam os privilégio do grupo.

Visões são objetos importantes no banco de dados pois podem abstrair consultas complexas. Isso é conseguido pois as visões representam um select. A partir do momento que uma visão é criada, pode-se realizar consultas sobre as mesmas como se fosse tabelas normais do banco de dados. Alguns SGBDs permitem que sejam feitas inserções, atualizações e exclusões em visão (essas operações afetariam as tabelas utilizadas para construir a visão) mas nós não vamos estudar essa modalidade.

```
create view employeedepto as select e.name, d.name from employee e join
depto d on e.did=d.did;
```

ou

```
create view employeedepto (ename, dname) as select e.name, d.name from
employee e join depto d on e.did=d.did;
```

As visões podem ser materializadas (`materialized`) ou temporárias (`temporary` ou `temp`), essa última é excluída quando a sessão postgres for finalizada.

```
create materialized view employeedepto as select e.name, d.name from
employee e join depto d on e.did=d.did;

create view temporary employeedepto (ename, dname) as select e.name,
d.name from employee e join depto d on e.did=d.did;
```

pg_views armazena os dados sobre as visões.

Índices são recursos essenciais em banco de dados. Por meio deles, as consultas podem ser otimizadas e aceleradas

```
create [unique] index [concurrently] <name> on <table> [using <method>]
                                                       [include (columns)]
```

methods: btree (padrão) e hash

pg_indexes armazena dados sobre os índices.

O schema `information_schema` armazena objetos com informações sobre o esquema do banco conectado.

```
select * from information_schema.tables;
```

```
-- armazena VIEW e BASE_TABLE (table_type)
```