



**Estratégia**  
CONCURSOS

Aula

**Tecnologia da Informação (parte VI) para Técnico Judiciário - TRT- PR**

Professor: Equipe Informática e TI, Thiago Rodrigues Cavalcanti, Victor Dalton

# AULA 00: Introdução e administração de banco de dados em PostgreSQL

## Sumário

Apresentação do professor.....	1
Motivação para o curso.....	2
Cronograma.....	4
PostgreSQL.....	5
1. Introdução ao PostgreSQL.....	5
1.1. O que é PostgreSQL?.....	5
1.1.1. Uma rápida história do PostgreSQL.....	6
1.2. Instalando o PostgreSQL.....	9
1.3. Conectando a um servidor Postgres.....	10
1.4. Arquivos de configuração para acesso remoto.....	12
1.5. Criando um banco de dados.....	14
2. Administração do Servidor.....	16
2.1. Distribuição dos arquivos em diretórios.....	16
2.2. Setup e operação do servidor.....	16
2.2.1. pgAdmin III.....	16
2.3. Controlando status do Servidor (pg_ctl).....	18
2.4. Catálogo do sistema.....	18
2.4.1. Pg_cast, pg_depend, pg_proc, pg_attrdef e pg_type.....	19
2.4.2. Pg_settings.....	20
2.5. Segurança da informação no Postgres.....	21
2.5.1. Autenticação do cliente.....	22
2.5.1.1. O arquivo pg_hba.conf.....	23
2.5.1.2. Métodos de autenticação.....	23
2.5.2. Papeis no banco de dados.....	24
2.6. Backup e Restore.....	26
Questões Comentadas.....	30
Considerações finais.....	35
Referências.....	35

## Apresentação do professor

Olá senhoras e senhores! Hoje daremos continuidade um conjunto de cursos relacionados a Banco de dados. É um prazer imenso fazer parte desta equipe de professores do Estratégia Concursos e ter a oportunidade de apresentar um pouco do meu conhecimento e experiência em concursos públicos! Gostaria, antes de começar de fato o conteúdo teórico desta aula, de me apresentar de forma rápida. Creio que seja importante para nos conhecermos.

Meu nome é Thiago, sou casado, tenho um filho de quatro anos. Sou cristão. Frequento a IPN – Igreja Presbiteriana Nacional. Sou formado em

Ciência da Computação pela UFPE. Tenho mestrado em engenharia de software pela mesma universidade. Frequento academia para manter a forma, mas meu hobby mesmo é pedalar! Decidi vender o carro e viver num desafio intermodal de transporte. Vou para o trabalho de bike sempre que possível!

Onde eu trabalho? No Banco Central do Brasil! Fruto de uma trajetória de dois anos de estudos diários. Aposentei as canetas em 2010. Trabalho com análise e modelagem de dados. Já participei do desenvolvimento de software, mas, desde 2014, estou em uma nova área dentro do Departamento de Informática (Deinf). Depois eu posso contar como está estruturado o Deinf de uma das principais autarquias da administração pública federal.

Minha mais recente experiência com dados, seja na administração ou modelagem, é parte de uma estratégia profissional de alinhar meu trabalho diário como servidor público com minha carreira paralela de professor de Banco de Dados (BD) e *Business Intelligence* (BI). A ideia é conseguir me especializar cada vez mais no tema nesta nova carreira dentro da TI, que o mercado está denominando de cientista dos dados.

Entrei no universo de ensino para concursos públicos há alguns anos. Desde 2012 tenho me dedicado especificamente a área de BD e BI. Adquiri experiência em cursos presenciais aqui em diversas partes do Brasil, também tenho gravado sistematicamente aulas on-line. É com essa bagagem que eu me apresento aos senhores como professor. A meta é sempre desenvolver um material completo, recheado de questões e com dicas exclusivas para ajudar você no seu objetivo: ser aprovado!

## **Motivação para o curso**

O concurso em questão neste curso é para o cargo de Analista Judiciário do Tribunal Regional do Trabalho do Paraná - TRT-PA. Trabalhar em um tribunal é sem dúvida o objetivo de muitos concurseiros. Vejamos algumas das características do cargo:

### **Salário:**

Remuneração inicial: R\$ 8.803,97 (oito mil, oitocentos e três reais e noventa e sete centavos)

### **Formação:**

Diploma de curso superior em Tecnologia da Informação, ou de qualquer outro curso superior com Pós-Graduação na área de Tecnologia da Informação, com carga horária mínima de 360 (trezentos e sessenta) horas/aula, devidamente registrados e fornecidos por Instituição de Ensino Superior reconhecida pelo Ministério da Educação. Excepcionalmente, mediante justificativa escrita do próprio candidato, poderá ser aceito certificado de colação

de grau ou certificado de conclusão da Pós-Graduação, acompanhados de histórico escolar.

### **Atribuições:**

Desenvolver projetos e sistemas de informática; documentar os sistemas; analisar e avaliar diagramas, estruturas e descrições de entradas e saídas de sistemas; sugerir as características e quantitativos de equipamentos necessários à utilização dos sistemas; analisar e avaliar as definições e documentação de arquivos, programas, rotinas de produção e testes de sistemas; identificar as necessidades de produção, alteração e otimização de sistemas; prestar suporte técnico e treinamento aos usuários de sistemas; analisar e avaliar procedimentos para instalação de base de dados, assim como definir dados a serem coletados para teste paralelo de sistemas; planejar e coordenar as atividades de manutenção dos sistemas em operação; elaborar projetos de páginas para internet e intranet; elaborar especificação técnica para subsidiar a aquisição de software e equipamentos de informática; propor padrões e soluções para ambientes informatizados; elaborar pareceres técnicos; redigir, digitar e conferir expedientes diversos e executar outras atividades de mesma natureza e grau de complexidade.

É importante salientar uma característica deste edital a carga horária semanal de trabalho. Segundo o edital os candidatos aos Cargos/Áreas/Especialidades do presente Concurso ficarão sujeitos à jornada de 40 (quarenta) horas semanais. Vejam, também, que o salário é bem interessante e o trabalho a ser desenvolvido no tribunal, após uma leitura rápida das atribuições do cargo, deve ser bastante gratificante para um profissional e TI. Mas antes você precisa ser aprovado na prova, e é para isso que estamos aqui: ajudar você a extrair o máximo do seu potencial!

Sendo assim, montamos um **curso teórico em PDF**, baseado, principalmente, nas questões anteriores da Fundação Carlos Chagas (FCC). Nosso objetivo contribuir para seu estudo agregando material teórico de qualidade com questões de provas anteriores. **Vamos juntos?**

**Observação importante:** este curso é protegido por direitos autorais (copyright), nos termos da Lei 9.610/98, que altera, atualiza e consolida a legislação sobre direitos autorais e dá outras providências.

Grupos de rateio e pirataria são clandestinos, violam a lei e prejudicam os professores que elaboram os cursos. Valorize o trabalho de nossa equipe adquirindo os cursos honestamente através do site Estratégia Concursos ;-)

**Observação importante II:** todo o conteúdo deste curso encontra-se completo em nossos textos escritos. As videoaulas visam reforçar o aprendizado, especialmente para aqueles que possuem maior facilidade de aprendizado com vídeos e/ou querem ter mais uma opção para o aprendizado.

## Cronograma

Para proporcionar uma visão geral do assunto e fornecer uma linha de ação para o estudo da matéria dividimos o curso em oito aulas, sendo esta a aula 00. A aula engloba a primeira parte da matéria de banco de dados PostgreSQL. Mostraremos alguns aspectos conceituais deste SGBD *Open Source* e em seguida apresentaremos algumas peculiaridades da administração deste banco de dados. As demais aulas, seguindo a ementa do curso, são apresentadas abaixo e estão distribuídas como se segue:

**Ementa do curso:** Linguagem PL/SQL. Conceitos e administração de banco de dados Oracle 11g e 12c, PostgreSQL e SQL Server.

Pois bem, e como serão distribuídas as nossas aulas?

**Aula 00:** Aula demonstrativa – PostgreSQL – Introdução e administração de banco de dados PostgreSQL

**Aula 01:** PostgreSQL – psql, usando o pgAdmin, peculiaridades do tipos de dados, objetos e SQL

**Aula 02:** Oracle 11g e 12c – Parte I - Características de banco de dados Oracle: Estrutura Lógica e Física, Objetos do Schema, Tipos de Dados, Estrutura de Memória

**Aula 03:** Oracle 11g e 12c – Parte II - Gerenciando o banco de dados Oracle: Estrutura de Processos, Modo de Inicialização e de Encerramento de Servidor, Backup

**Aula 04:** Linguagem – PL/SQL – Parte I – Introdução ao código PL/SQL, Identificadores, instruções executáveis.

**Aula 05:** Linguagem – PL/SQL – Parte II – Estruturas de controle, Cursores, Exceções

**Aula 06:** Linguagem – PL/SQL – Parte III – Procedures, functions e Packages, Triggers

**Aula 07:** Introdução a Sql Server

**Observação importante III:** o conteúdo deste curso não abrange todo o edital de Tecnologia da Informação, apenas os itens elencados no conteúdo do curso. Estão disponíveis outros cursos ministrados pelos Professores Fábio Alves, André Castro, Victor Dalton e Diego Carvalho, também no Estratégia Concursos, que também abrange outra parte do edital.

Definido o cronograma, vamos partir para o conteúdo da nossa aula demonstrativa.

## PostgreSQL

# 1. Introdução ao PostgreSQL

Queremos aqui contextualizar PostgreSQL dentro do arcabouço do Banco de Dados para que você possa se situar dentro do assunto. A nossa divisão do assunto foi pensada da seguinte forma: primeiro veremos o processo de administrações de dados utilizando o SGBD PostgreSQL (Aula 00), em seguida, analisaremos as peculiaridades dos comandos de manipulação e criação de dados (Aula 01). Neste segundo momento, trataremos também dos tipos de dados e das extensões do PostgreSQL à linguagem SQL/ANSI.

## 1.1. O que é PostgreSQL?

O banco de dados *open source* mais avançado do mundo! Isso mesmo! Do mundo! O PostgreSQL é um sistema de gerenciamento de banco de dados objeto-relacional (ORDBMS) baseado no POSTGRES versão 4.2, desenvolvido na Universidade de Berkeley na Califórnia (UCB) mais especificamente no departamento de ciência da computação. POSTGRES foi pioneiro em muitos conceitos que só se tornaram disponíveis em alguns sistemas de banco de dados comerciais posteriormente.

O PostgreSQL é um descendente de código aberto do programa original desenvolvido em Berkeley. Ele suporta grande parte dos comandos da linguagem padrão SQL/ANSI e oferece muitas características modernas, entre elas: consultas complexas, chaves estrangeiras, triggers, views atualizáveis, integridade transacional e controle de concorrência multiversão.

Além disso, o PostgreSQL pode ser estendido pelo usuário de várias maneiras, por exemplo, adicionando novos tipos de dados, funções, operadores, funções agregadas, métodos para criação índice, e linguagens procedurais.

É por causa da licença liberal que o PostgreSQL pode ser usado, modificado e distribuído por qualquer pessoa gratuitamente para qualquer fim, seja ele privado, comercial ou acadêmico.

### **1.1.1. Uma rápida história do PostgreSQL**

Primeiramente deixem-me justificar as próximas linhas. Você deve estar se perguntando, por que eu quero conhecer a história do PostgreSQL? A ideia é apresentar como as principais funcionalidades evoluíram ao longo do tempo, e, o mais importante, que você possa ir se familiarizando com os termos associados aos serviços providos por um SGBD.

Originalmente chamado de Postgres, foi criado na UCB por um professor de ciência da computação chamado **Michael Stonebraker**, que se tornou o CTO da Informix Corporation, empresa que viria ser comprada pela IBM. Em 1995, dois alunos de Ph.D. do laboratório de Stonebraker, Andrew Yu e Jolly Chen, substituíram linguagem de consulta PostQUEL do Postgres por um subconjunto estendido de funções do SQL. Eles renomearam o sistema para Postgres95.

Em 1996, Postgres95 saiu de dentro da academia e começou uma nova vida no universo open source fora do campus. Um grupo de desenvolvedores dedicados, de fora de Berkeley, viu o sistema como uma promessa, e se dedicaram ao seu desenvolvimento contínuo. Se valendo de contribuições enormes de tempo, habilidade, trabalho e conhecimento técnico, esse grupo global de desenvolvimento transformou radicalmente o Postgres.

Durante os anos subsequentes, trouxeram consistência e uniformidade para o código fonte, criando testes de regressão detalhados para a garantia de qualidade, criaram listas de discussão para relatórios de bugs, e inúmeros bugs foram corrigidos, e ainda, foram adicionadas novas funcionalidades incríveis. Eles ajustaram todo o sistema, preenchendo várias lacunas tais como documentação para desenvolvedores e usuários.

O fruto deste trabalho foi um novo banco de dados que ganhou uma **reputação sólida e estabilidade**. Com o início de sua nova vida no mundo *open source*, muitos novos recursos adicionados e vários aprimoramentos fizeram o sistema de banco de dados receber seu nome atual: PostgreSQL. "Postgres" ainda é usado como um apelido mais fácil de pronunciar.

O PostgreSQL começou sua contagem de versões pela **versão 6.0**, dando crédito a seus muitos anos anteriores de desenvolvimento. Com a ajuda de centenas de desenvolvedores de todo o mundo, o sistema foi alterado e melhorado em quase todas as áreas. Ao longo dos próximos quatro anos (versões 6.0-7.0), grandes melhorias e novas funcionalidades foram implementadas, tais como:



- **Multiversion Concurrency Control (MVCC).** O nível de bloqueio de tabela foi substituído por um sistema de controle de concorrência multiversão sofisticado, que permite que aos leitores continuar lendo dados consistentes durante a atividade de escrita, possibilitando backups on-line enquanto o banco de dados está em execução.

Antes de continuar tratando das melhorias feitas no Postgres, vamos nos concentrar para fazermos a questão abaixo:



**01) BANCA: CESPE ANO: 2014 ÓRGÃO: ANATEL PROVA: ANALISTA ADMINISTRATIVO - SUPORTE E INFRAESTRUTURA DE TI**

A respeito de banco de dados, julgue os itens que se seguem.

O PostgreSQL 9.3, ao gerenciar o controle de concorrência, permite o acesso simultâneo aos dados. Internamente, a consistência dos dados é mantida por meio do MVCC (multiversion concurrency control), que impede que as transações visualizem dados inconsistentes.

**Gabarito C.** O PostgreSQL fornece um rico conjunto de ferramentas para os desenvolvedores gerenciarem o acesso simultâneo aos dados. Internamente, a consistência dos dados é mantida por meio de um **modelo de concorrência multiversão** (MVCC). Isto significa que ao consultar um banco de dados cada transação enxerga uma fotografia dos dados (ou uma versão do banco de dados) no seu estado em algum tempo passado, independentemente do estado atual dos dados subjacentes.

Isso protege a transação de enxergar dados inconsistentes que poderiam ser causados por atualizações de transações simultâneas nas mesmas linhas de dados, fornecendo um isolamento da transação para cada sessão de banco de dados. O MVCC, por não utilizar as metodologias de bloqueio de sistemas de banco de dados tradicionais, minimiza a disputa de bloqueio, a fim de permitir que um bom desempenho em ambientes de multiusuários.

A principal vantagem de usar o modelo MVCC no controle de concorrência, em vez de bloqueios de itens de dados é que, no MVCC bloqueios adquiridos para consulta de dados (leitura) não conflitam com os bloqueios adquiridos para a gravação de dados, e assim a leitura nunca bloqueia a escrita, e a escrita nunca bloqueia a leitura. PostgreSQL mantém essa garantia mesmo quando fornecendo o nível mais rigoroso de isolamento da transação através do uso.

- **Recursos SQL importantes.** Muitas melhorias foram feitas SQL incluindo subconsultas, padronização, restrições, chaves primárias, chaves estrangeiras, identificadores entre aspas, tipo de coerção de string literal (*literal string type coercion*), tipo *casting*, entrada dos tipos inteiro binário e hexadecimal, entre outros.



- **Tipos internos (built-in) melhorados.** Foram adicionados novos tipos nativos, incluindo uma ampla gama de tipos de data/hora e tipos geométricos adicionais.
- **Speed.** Velocidade e desempenho tiveram grandes aumentos, na ordem de 20 a 40%, e tempo de inicialização foi reduzido em 80%.

Os quatro anos seguintes (versões 7.0 a 7.4) trouxeram o *Write-Ahead Log* (WAL), esquemas SQL, *prepared queries*, *outer joins*, consultas complexas, sintaxe do *join* do SQL92, TOAST, suporte a IPv6, *information schema* do SQL-padrão, *full-text index*, *auto-vacuum*, linguagens procedurais Perl/Python/TCL, melhor suporte a SSL, uma otimizador de consulta revisado, informações estatísticas sobre o banco de dados, maior segurança, *table functions*, e ainda, melhorias no log, melhorias significativas de velocidade, entre outras coisas. Uma pequena amostra de desenvolvimento intenso do PostgreSQL é refletida em suas notas de lançamento.

As versões 8.0 a 8.4 foram lançadas entre os anos de 2005 a 2009. Nestes quatro anos de desenvolvimento tivemos o incremento das seguintes funcionalidades: Servidor nativo Microsoft Windows, *savepoints*, *tablespaces*, manipulação de exceção em funções, a recuperação *point-in-time*, otimização de desempenho, commit em duas fases, o particionamento de tabelas, verificação de índice bitmap, bloqueio de linha compartilhada, papéis (roles), *online index builds*, *advisory locks*, *warm standby*, tuplas do tipo *heap-only*, *full-text search*, SQL/XML, tipo ENUM, tipo UUID, funções de janelas, parâmetros default para funções, permissões em nível de coluna, restauração de banco de dados paralelos, agrupamento por banco de dados, expressões de tabela comuns e consultas recursivas.

Por fim chegamos às últimas e mais recentes versões (9.0 a 9.4). Esses são os releases mais atuais do PostgreSQL. Por meio deles incrementos foram feitos ao sistema, vejamos quais foram inclusas melhorias de acordo com as versões na tabela abaixo:

9	Replicação de streaming binário interna (built-in), <i>hot standby</i> , suporte ao Windows 64-bits, gatilhos-por-coluna e execução trigger condicional, restrições de exclusão, blocos de código anônimos, parâmetros nomeados, regras de senha.
9.1	Replicação síncrona, agrupamentos por colunas, tabelas <i>unlogged</i> , indexação dos vizinhos mais próximos (k-NN), isolamento de instantâneo serializado (SSI), expressões de tabela comuns armazenáveis, integração com Linux-SE, extensões, tabelas anexadas SQL/Med (Wrappers para dados externos), gatilhos em views.
9.2	<i>Cascading replication streaming</i> , varreduras index-only, suporte a

	JSON nativo, a melhoria da gestão de bloqueio, range de tipos, ferramenta pg_receivexlog, índices GiST para espaço-particionado.
9.3	Tarefas de background customizáveis, checksums de dados, operadores JSON dedicados, LATERAL JOIN, pg_dump mais rápido, nova ferramenta de monitoramento de servidor <b>pg_isready</b> , características de triggers, funcionalidades para visões, tabelas estrangeiras graváveis, visões materializadas e melhorias de replicação.
9.4	Tipo de dados JSONB, instrução ALTER SYSTEM para alterar valores de configuração, atualização de visões materializadas sem bloqueio de leitura, o registro dinâmico para registro/start/ stop de processos em segundo plano, Logical API Decoding, melhorias índice GiN, suporte a Linux huge page e cache de recarga de banco de dados via pg_prewarm.

Você pode ter visto uma quantidade enorme de palavras acima cujo significado você ainda não tem conhecimento. Digo logo que não vamos detalhar a definição de todos eles. Mas posso garantir que passaremos pelos principais. Essa lista vai funcionar na hora da prova como uma memória auxiliar, você verá como seu cérebro funciona bem quando solicitarem informações a respeito das funcionalidades existentes ou não no PostgreSQL.

A partir de agora quando falarmos em PostgreSQL, estamos tratando da versão 9.4 que é a última versão com suporte disponível.

## 1.2. Instalando o PostgreSQL

Vamos então dar os primeiros passos para o entendimento. O PostgreSQL é um servidor de banco de dados SQL avançado, disponível em uma ampla gama de plataformas. Está disponível gratuitamente para usar, alterar ou redistribuir da forma como você quiser. Sua licença é de código aberto muito parecida com a licença BSD (Berkeley Software Distribution), embora diferente o suficiente ser conhecida como TPL (**The PostgreSQL License**).

O PostgreSQL já é usado por muitos pacotes de aplicativos diferentes, e por isso é possível que você já encontre ele instalado em seus servidores. Muitas distribuições Linux incluem PostgreSQL como parte da instalação básica, ou fornecem a opção de incluí-lo durante a instalação.

Se você ainda não tem uma cópia, ou não tem a versão mais recente, pode baixar o código-fonte ou baixar um dos pacotes binários para uma ampla variedade de sistemas operacionais a partir da seguinte URL:

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

Detalhes da instalação variam significativamente de uma plataforma para outra e não existem truques especiais ou receitas para mencionar. Basta seguir o guia de instalação, e você vai longe!

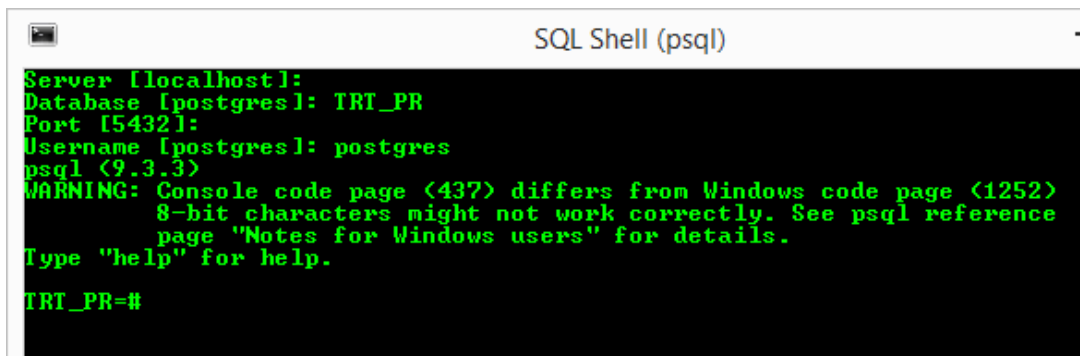
O processo de instalação envolve a cópia de todos os programas compilados em um diretório que irá servir como a base para todas as atividades do PostgreSQL. Ele também irá conter todos os programas, bancos de dados PostgreSQL, e arquivos de log. O diretório é normalmente chamado /usr/local/pgsql.

### 1.3. Conectando a um servidor Postgres

Ok! Instalamos nosso banco de dados e agora? Como faremos para utilizá-lo? Como devemos nos comunicar com ele? Conectar-se ao banco de dados é a primeira experiência da maioria das pessoas. Então, vamos tê-la, e corrigir quaisquer problemas que tivermos ao longo do caminho. Lembre-se que a conexão deve ser feita de forma segura, portanto, pode haver alguns obstáculos para garantir que os dados que desejamos acessar são seguros.

Para se conectar, precisamos ter um servidor PostgreSQL rodando no host, escutando uma porta. Nesse servidor devem existir um banco de dados e um usuário. O host deve permitir explicitamente conexões dos clientes que fornecem os dados para autenticação usando o método especificado pelo servidor. Por exemplo, especificar uma senha não irá funcionar se o servidor solicitou uma forma diferente de autenticação.

Vejam abaixo um exemplo de uma conexão usando o SQL Shell (psql). Nele utilizamos os seguintes dados: localhost para o servidor, TRT\_PR para o banco de dados, 5432 para a porta e postgres para o usuário.



```
SQL Shell (psql)
Server [localhost]:
Database [postgres]: TRT_PR
Port [5432]:
Username [postgres]: postgres
psql <9.3.3>
WARNING: Console code page (437) differs from Windows code page (1252)
8-bit characters might not work correctly. See psql reference
page "Notes for Windows users" for details.
Type "help" for help.
TRT_PR=#
```

**Dica:** O **psql** é um front-end baseado em terminal para PostgreSQL. Ele permite que você digite os comandos interativamente, envie para o PostgreSQL e veja os resultados da consulta. Alternativamente, sua entrada pode ser de um arquivo. Além disso, ele fornece um número de meta-comandos e diversas funcionalidades *shell-like* para facilitar a escrita de scripts e automatizar uma grande variedade de tarefas. É importante conhecê-lo!

O servidor de banco de dados PostgreSQL é classificado com cliente-servidor. O local onde sistema roda conhecido como o host. Podemos acessar o servidor PostgreSQL remotamente através da rede. No entanto, devemos especificar o host, que é um nome de host ou um hostaddr, que é um endereço IP. Podemos especificar o host como "localhost" se quisermos fazer uma conexão **TCP/IP** para o mesmo servidor. Muitas vezes, é melhor usar uma conexão socket Unix, que é tentada se o nome do host iniciar por barra (/).

Em qualquer sistema, pode haver mais de um servidor de banco de dados. Cada servidor de banco de dados escuta em exatamente uma porta de rede, que não pode ser compartilhada entre os servidores no mesmo host. O número da porta padrão para PostgreSQL é **5432**, o que foi registrado na Internet Assigned Numbers Authority (**IANA**), e é atribuída exclusivamente ao PostgreSQL. O número da porta pode ser usado para identificar exclusivamente um servidor de BD específico se existirem muitos.

Um servidor de banco de dados também é conhecido como um "cluster de banco de dados", porque o servidor PostgreSQL permite definir um ou mais bancos de dados em cada servidor. Cada solicitação de conexão deve identificar exatamente um banco de dados identificado por seu DBNAME. Quando você se conectar, você só será capaz de ver objetos do banco de dados criados dentro dessa base especificada na conexão.

### **Dica do professor**

Se você quiser confirmar que conectou ao servidor certo e da maneira certa, você pode executar alguns ou todos os seguintes comandos:

**SELECT inet\_server\_port();** - Exibe a porta na qual o servidor está escutando.

**SELECT current\_database();** - Mostra o banco de dados atual.

**SELECT current\_user;** - Mostra o ID do usuário atual.

**SELECT inet\_server\_addr();** - Mostra o endereço IP do servidor que aceitou a conexão.

A senha do usuário **não** é acessível usando SQL geral, por razões óbvias.

Vejamos então como esse assunto já foi cobrado em provas anteriores:

**02) BANCA: FCC ANO: 2012 ÓRGÃO: TCE-AM PROVA: ANALISTA TÉCNICO DE CONTROLE EXTERNO - TECNOLOGIA DA INFORMAÇÃO**

Sobre os fundamentos arquiteturais do banco de dados PostgreSQL, considere:

I. Utiliza um modelo cliente/servidor, consistindo de um processo servidor que gerencia os arquivos do banco de dados, controla as conexões dos clientes ao banco dados e efetua ações no banco de dados em favor dos clientes.

II. A aplicação cliente, que irá efetuar as operações no banco de dados, poderá ser de diversas naturezas, como uma ferramenta em modo texto, uma aplicação gráfica, um servidor web que acessa o banco de dados para exibir as páginas ou uma ferramenta de manutenção especializada.

III. A aplicação cliente pode estar localizada em uma máquina diferente da máquina em que o servidor está instalado. Neste caso, a comunicação entre ambos é efetuada por uma conexão TCP/IP. O servidor pode aceitar diferentes conexões dos clientes ao mesmo tempo.

Está correto o que se afirma em

A I, II e III.

B I e II, apenas.

C I e III, apenas.

D II e III, apenas.

E III, apenas.

**Gabarito C.** Observem que segundo o conteúdo apresentado acima podemos afirmar que as alternativas I e III estão corretas. Já alternativa II está falando de servidores de aplicação, também conhecidos como servidores web. Eles são os responsáveis por montar a página web e enviar para o cliente.

Existe uma variedade de distribuições possíveis para o PostgreSQL. Em muitas delas, você vai verificar que o acesso remoto é desabilitado por padrão, como medida de segurança. Vamos, a seguir, aprender a fazer ajustes nos arquivos de configuração para aceitação de conexões remotas.

### **1.4. Arquivos de configuração para acesso remoto**

Vamos agora conhecer alguns arquivos nos quais precisamos fazer alterações para admitirmos o acesso remoto. Primeiramente mostraremos quais são as alterações e cada arquivo, em seguinte teceremos os comentários teóricos explicativos a respeito de cada alteração.

- Adicionar/editar a seguinte linha no seu arquivo postgresql.conf:

- `listen_addresses = '*'`

- Adicionar a seguinte linha como a primeira linha do arquivo `pg_hba.conf`, para permitir o acesso a todos os bancos de dados para todos os usuários utilizando uma senha criptografada:

#	TYPE	DATABASE	USER	CIDR-ADDRESS	METHOD
	host	all	all	0.0.0.0/0	md5

O parâmetro **listen\_addresses** especifica qual o range de endereços IP pode escutar. Isso permite que você tenha mais de uma placa de rede (NICs) por sistema. Na maioria dos casos, nós queremos aceitar conexões em todas as NICs, por isso usamos "\*", que significa "todos os endereços IP".

O **pg\_hba.conf** contém um conjunto de **regras de autenticação** baseada no host. Cada regra é considerada em sequência até que uma regra possa ser disparada, ou a tentativa de acesso seja rejeitada especificamente, pelo uso do método **reject**. (falaremos dos métodos de acesso mais adiante).

A regra anterior significa que uma conexão remota específica é admitida para qualquer(all) usuário, ou qualquer(all) banco de dados, em qualquer endereço de IP e é solicitada a autenticar usando uma senha **md5** criptografada. Vejamos cada um dos parâmetros indicados:

**TYPE = host** significa uma conexão remota.

**DATABASE = all** significa "para todos os bancos de dados". Outros nomes devem corresponder exatamente a um nome de banco, exceto quando tiver um sinal de mais (+) como prefixo, neste caso queremos indicar um "*group role*" ao invés de um único usuário. Você também pode especificar uma lista de usuários, separados por vírgulas, ou usar o símbolo @ para incluir um arquivo com uma lista de usuários.

**USER = all** "para todos os usuários." Outros nomes devem corresponder exatamente, exceto quando prefixado com um sinal de mais (+), caso em que queremos dizer que estamos usando um "*group role*" ao invés de um único usuário. Da mesma forma que o DATABASE pode-se usar uma lista de usuários, separados por vírgulas ou usar o símbolo @ para incluir um arquivo com uma lista de usuários.

**CIDR-ADDRESS** consiste em duas partes: endereço IP/máscara de sub-rede. A máscara de subrede é especificada como os primeiros números (bits) do endereço IP que compõem a máscara. Assim /0 significa zero bits do endereço IP, de modo que todos os endereços IP serão comparados, por exemplo, 192.168.0.0/24 significaria igualar os primeiros 24 bits, portanto, qualquer endereço IP do formulário 192.168.0.x poderia corresponder.



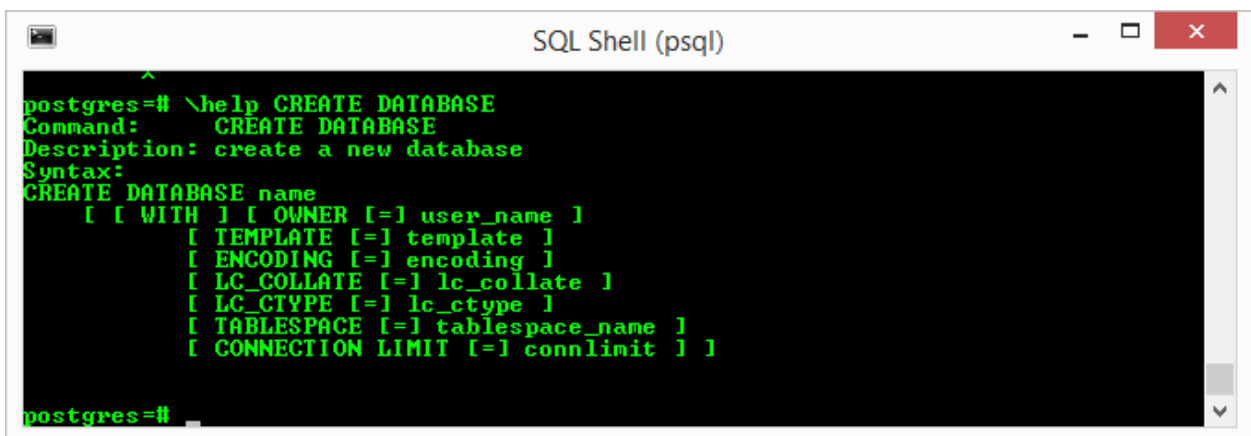
**Method = trust** efetivamente significa "sem autenticação". Outros métodos de autenticação incluem GSSAPI, SSPI, LDAP, RADIUS, e PAM. Conexões com o PostgreSQL também podem ser feitas usando SSL, no qual os certificados SSL do cliente em questão são utilizados para fornecer autenticação.

## 1.5. Criando um banco de dados

Agora que já aprendemos a ajustar o servidor para fazermos conexões remotas, vamos seguir em frente para entender os conceitos sobre alguns comandos ou aplicações oferecidas pelo PostgreSQL para criação das estruturas de objetos. Começaremos pelo **initdb**.

Usando a seguinte sintaxe: **initdb [option...] [--pgdata | -D] directory**, é possível criar um novo cluster de banco de dados. Este é uma coleção de bancos de dados que é gerenciada por uma instância do servidor. Utilizando o parâmetro PGDATA podemos especificar o diretório onde o cluster de banco de dados deve ser armazenado. Outra opção com o mesmo efeito é o uso do -D.

Veremos agora o comando CREATE DATABASE. É possível, utilizando a interface Shell (psql), rodar o comando \help e ver as características deste comando que serve basicamente para criação de uma nova base de dados. Observe a figura a seguir:



```
SQL Shell (psql)

postgres=# \help CREATE DATABASE
Command:      CREATE DATABASE
Description:  create a new database
Syntax:
CREATE DATABASE name
[ [ WITH ] [ OWNER [=] user_name ]
  [ TEMPLATE [=] template ]
  [ ENCODING [=] encoding ]
  [ LC_COLLATE [=] lc_collate ]
  [ LC_CTYPE [=] lc_ctype ]
  [ TABLESPACE [=] tablespace_name ]
  [ CONNECTION LIMIT [=] connlimit ] ]

postgres=#
```

Lembrando que para poder executar o comando é preciso ter o privilégio de CREATEDB. Por default, o novo banco de dados criado será basicamente um clone do banco de dados padrão do sistema denominado template1. É possível executar o comando usando o **wrapper createdb**. No Windows ele é um executável (.exe), disponível na pasta bin da instalação do PostgreSQL.

Abaixo apresentamos uma questão de prova que cobra este assunto. Usaremos a questão para detalharmos alguns aspectos do comando CLUSTER e das definições de LOCALE.



**03) BANCA: FCC ANO: 2013 ÓRGÃO: TRT - 12ª REGIÃO (SC) PROVA: ANALISTA JUDICIÁRIO - TECNOLOGIA DA INFORMAÇÃO**

Localização refere-se ao fato de uma aplicação respeitar as preferências culturais sobre alfabetos, classificação, formatação de números etc. PostgreSQL usa o padrão ISO C e POSIX fornecidos pelo sistema operacional do servidor para aplicar as regras de localização. O suporte à localização é automaticamente inicializado quando um cluster de banco de dados é criado usando o comando

- A create cluster.
- B create database.
- C initdb.
- D ccluster.
- E locale init.

**Gabarito C.** Vamos fazer alguns comentários sobre as alternativas. Na alternativa A temos o comando CREATE CLUSTER. Ele não existe dentro do rol de comandos do SGBD. O comando que existe é o **CLUSTER**. Ele é utilizado basicamente para reordenar uma tabela fisicamente de acordo com as informações de um dos índices. Vejamos a sintaxe do comando:

```
CLUSTER [VERBOSE] table_name [ USING index_name ]
```

Sobre as alternativas B e C acabamos de defini-las acima. A alternativa D tratar de um comando que não existe no Postgres. O utilitário (utility) que existe é **clusterdb** usado como **wrapper** para o comando CLUSTER que acabamos de falar.

Na letra E temos mais uma expressão que não existe na documentação do Postgres: "locale init". Mas sabemos que o suporte a *locale* é algo imprescindível dentro de qualquer SGBD. Vejamos então como o suporte a locale é tratado.

O suporte a *locale* refere-se ao fato das aplicações respeitarem características culturais como alfabeto, ordenação, formato de numeração ou moedas, etc. O PostgreSQL usa o padrão ISO C e POSIX fornecidos pelo sistema operacional do servidor para aplicar as regras de localização.

O suporte a *Locale* é automaticamente inicializado quando um cluster de banco de dados é criado usando o initdb. Ele irá inicializar o cluster de banco de dados com a definição do seu ambiente de execução por default, por isso, se o seu sistema já está definido para utilizar o idioma que você quer usar em seu cluster de banco de dados, então não há mais nada que você precise fazer. Se você quiser usar um Locale diferente (ou você não tem certeza de qual Locale está definido para sistema), você pode instruir initdb qual *locale* usar especificando a opção --locale.

Por exemplo:

```
initdb --locale = sv_SE
```

Neste exemplo para sistemas Unix definimos o idioma para Sueco (sv) que é falado na Suécia (SE).

## 2. Administração do Servidor

Esta parte do curso abrange os temas que são interessantes para o administrador de banco de dados PostgreSQL. Inclui detalhes da instalação do software, configuração do servidor, gerenciamento de usuários e bancos de dados, e tarefas de manutenção. Qualquer pessoa que execute um servidor PostgreSQL, mesmo para uso pessoal, mas especialmente em produção, deve estar familiarizado com os temas aqui abordados.

### 2.1. Distribuição dos arquivos em diretórios

Quando o PostgreSQL é instalado, ele cria em seu diretório raiz, normalmente **/usr/local/pgsql**. Este diretório mantém todos os arquivos necessários utilizados PostgreSQL divididos em vários subdiretórios:

**/bin** - programas de linha de comando do PostgreSQL, como **psql**.

**/data** - Arquivos de configuração e tabelas compartilhadas por todos os bancos de dados. Por exemplo, uma tabela **pg\_shadow** compartilhada por todos os bancos de dados.

**/data/base** - Um subdiretório é criado para cada banco de dados. Usando os comandos **du** e **ls**, os administradores podem exibir a quantidade de espaço em disco usado por cada banco de dados, tabela ou índice.

**/doc** - documentação do PostgreSQL.

**/include** - Inclui os arquivos usados por várias linguagens de programação.

**/lib** - Bibliotecas usadas por várias linguagens de programação. Este subdiretório também contém os arquivos usados durante a inicialização e exemplos e *templates* de arquivos de configuração que podem ser copiados para **/data** e modificados.

**/man** - manuais PostgreSQL páginas.

### 2.2. Setup e operação do servidor

Ferramentas de administração com interface gráfica são frequentemente solicitadas pelos administradores do sistema. PostgreSQL tem uma gama de opções de ferramentas. As duas opções mais populares são: pgAdmin III e phpPgAdmin. Apresentamos abaixo alguns aspectos da ferramenta pgAdmin.

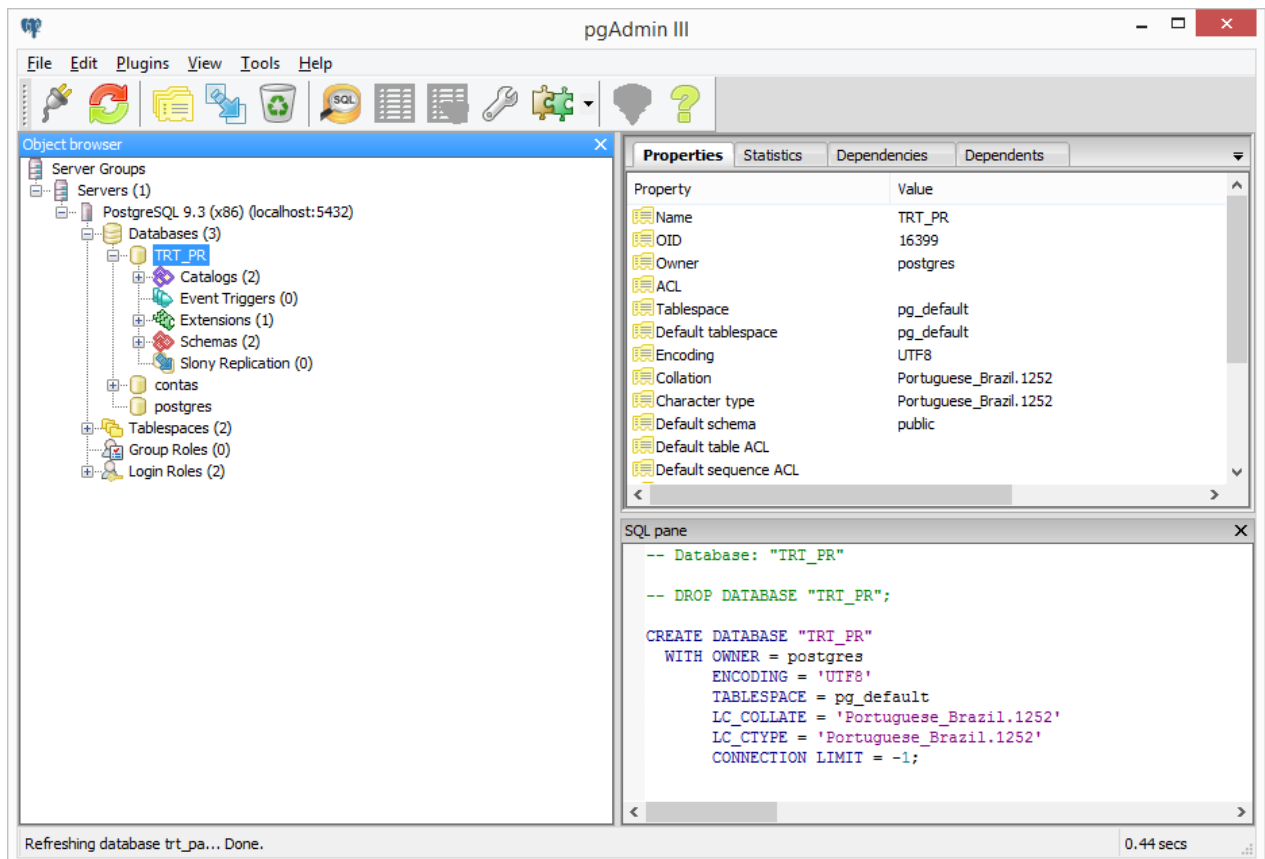
#### 2.2.1. pgAdmin III

O pgAdmin é uma plataforma para administração e desenvolvimento de banco de dados PostgreSQL. O aplicativo pode ser usado em Linux, FreeBSD,

Solaris, Mac OSX e plataformas Windows para gerenciar versões PostgreSQL 7.3 ou superior, executado em qualquer plataforma, bem como em versões comerciais e derivadas de PostgreSQL como Postgres Plus Advanced Server and Greenplum.

O pgAdmin é projetado para atender as necessidades de todos os usuários. Entre suas funcionalidades inclui escrever consultas SQL para desenvolver bases de dados complexas. A interface gráfica suporta todas as funcionalidades do PostgreSQL e torna fácil administração. O aplicativo também inclui um editor de **destaque de sintaxe SQL**, um editor de código para o server-side, um agente de agendamento de tarefas *SQL/batch/shell*, um suporte para o mecanismo de replicação *Slony-I* e muito mais. Uma conexão com o servidor pode ser feita usando TCP/IP ou socket no domínio Unix (nas plataformas \* nix), e pode ser criptografado usando SSL. Drivers adicionais não são necessários para se comunicar com o servidor de banco de dados.

O pgAdmin é desenvolvido por uma comunidade de especialistas em PostgreSQL de todo o mundo e está disponível em mais de uma dúzia de idiomas. Ele é um Software Livre liberado sob a licença do PostgreSQL. É possível visualizar a interface da plataforma na figura abaixo.



## 2.3. Controlando status do Servidor (pg\_ctl)

O `pg_ctl` é um utilitário para controlar o status de um cluster de banco de dados PostgreSQL, ou seja, iniciar, parar ou reiniciar o servidor back-end do PostgreSQL, ou ainda, exibir o status de um servidor em execução. Embora o servidor possa ser iniciado manualmente, o `pg_ctl` encapsula tarefas como redirecionar a saída do log. Ele também fornece opções convenientes para uma parada controlada.

Após iniciar o servidor usando comando `pg_ctl` podemos visualizar alguns arquivos que compõe a configuração do servidor:

- **postmaster.pid** - a existência deste arquivo no diretório de dados é usado para ajudar **pg\_ctl** a determinar se o servidor está sendo executado ou não.
- **postmaster.opts.default** - se esse arquivo existir no diretório de dados, o `pg_ctl` (no modo **start**) passará o conteúdo do arquivo como opções para o comando.
- **postmaster.opts** - se esse arquivo existir no diretório de dados, o `pg_ctl` (no modo **restart**) vai passar o conteúdo do arquivo como opções para o servidor PostgreSQL. O conteúdo deste arquivo também é exibido no modo de status.
- **postgresql.conf** - este arquivo, localizado no diretório de dados, é analisado para descobrir a porta apropriada para usar com `psql` quando o parâmetro `-w` é informado.

### Dica do professor

Ao entrar no `psql` o que você pode fazer? Executar comandos SQL de criação de objetos na base ou para manipulação de tabelas e outros objetos da base dados. Algumas dicas podem ajudar, digite:

`\h` - para ajuda com comandos SQL

`\?` - para obter ajuda sobre comandos internos

`\g` ou ponto e vírgula (;) - para executar consulta

`\copyright` - para termos de distribuição

`\q` - para sair

## 2.4. Catalogo do sistema

Os catálogos do sistema são o lugar onde um SGBD armazena os metadados de um esquema relacional, tais como informações sobre tabelas e colunas, e as informações sobre a estruturação interna. Os catálogos do sistema do PostgreSQL são tabelas regulares. Você pode eliminar e recriar as tabelas, colunas, adicionar valores de inserção e atualização nestas tabelas, contudo, é preciso ter cuidado para não danificar seu sistema.

Normalmente, não se deve alterar os catálogos do sistema à mão, há sempre comandos SQL para fazer isso. (Por exemplo, CREATE DATABASE insere uma linha para o **catálogo pg\_database**, e realmente cria o banco de dados no disco). Há algumas exceções para operações particularmente esotéricas, como a adição de métodos de acesso a índice. Passaremos aqui por algumas definições presentes no catálogo de sistema. Também levaremos em consideração o assunto cobrado em provas anteriores de concurso.

### 2.4.1. Pg\_cast, pg\_depend, pg\_proc, pg\_attrdef e pg\_type

Apresentaremos esses conceitos por meio da questão abaixo:



#### 04) BANCA: VUNESP ANO: 2013 ÓRGÃO: COREN-SP PROVA: ANALISTA - SISTEMAS

No sistema gerenciador de bancos de dados PostgreSQL (v. 9.1), o item do catálogo de sistema que armazena informações sobre valores default de colunas das tabelas é

- A pg\_casts
- B pg\_depend
- C pg\_proc
- D pg\_attrdef
- E pg\_type

**Gabarito D.** Nós vamos analisar primeiro o que cada uma das alternativas apresenta. Lembrando que todas as alternativas fazem parte do catálogo de sistemas. O **pg\_cast** grava o caminho de conversão entre os dados, tanto as possibilidades disponíveis nativamente no sistema, quanto aquelas que são definidas pelo usuário.

O **pg\_depend** armazena os relacionamentos de dependência entre os objetos do banco de dados. Esta informação permite ao comando DROP descobrir quais outros objetos devem ser removidos pelo DROP CASCADE ou impedir a remoção deles no caso de DROP RESTRICT.

O **pg\_proc** registra as informações sobre as funções e as procedures. A tabela contém dados para funções agregadas, bem como funções simples.

O **pg\_attrdef** armazena os valores default das colunas. A principal informação sobre colunas é armazenada em **pg\_attribute** (ou tabela do catálogo). Apenas as colunas que explicitamente especificam o valor default (quando a tabela é criada ou a coluna é adicionada) terão uma entrada aqui. Vejam que está é a nossa resposta. Vamos analisar agora a última alternativa.

O **pg\_type** grava informações sobre os tipos de dados. Os tipos básicos e tipos enum (escalares) são criados com CREATE TYPE, e os domínios com CREATE DOMAIN. Um tipo composto é criado automaticamente para cada tabela no banco de dados, para representar a estrutura de linha da tabela. Também é possível criar tipos compostos com CREATE TYPE AS.

### 2.4.2. Pg\_settings

A visão pg\_settings fornece acesso aos parâmetros de run-time do servidor. É essencialmente uma interface alternativa para os comandos SHOW e SET usados para manipular os parâmetros de sistema. Ela também fornece acesso a algumas informações sobre cada parâmetro que não estão diretamente disponíveis no comando SHOW, como valores mínimos e máximos.

A visão pg\_settings não pode ter linhas inseridas ou apagadas, mas pode ser atualizada. Uma atualização aplicada a uma linha da visão é equivalente a executar o comando SET no parâmetro. A alteração afeta apenas o valor utilizado pela **sessão atual**. Se uma atualização for emitida dentro de uma transação que é abortada mais tarde, os efeitos do comando UPDATE desaparecem quando a transação é revertida. Uma vez que a transação sofre COMMIT, os efeitos vão persistir até o final da sessão, a menos que substituída por outro comando UPDATE ou SET.

Vamos a seguir fazer uma questão a respeito do assunto.



**05) BANCA: FCC ANO: 2014 ÓRGÃO: TJ-AP PROVA: ANALISTA JUDICIÁRIO - BANCO DE DADOS - DBA**

Um dos itens da administração do sistema gerenciador de bancos de dados PostgreSQL (V.9.3.4) refere-se a gerenciar informações sobre os bancos de dados por ele controlados. O PostgreSQL contém algumas visões que auxiliam nessa tarefa, dentre elas, a visão pg\_settings que contém dados sobre

A os parâmetros run-time do servidor.

B estatísticas das tabelas do servidor.

C os usuários dos bancos de dados.

D a lista de bloqueios impostos.

E a lista das funções presentes no banco de dados.

**Gabarito A.** A questão pergunta sobre os dados que são armazenados na visão pg\_setting. Sabemos que são os dados de run-time, mais quais exatamente? Segue uma tabela com os parâmetros, tipos e descrição das informações contidas na visão.



Nome	Tipo	Descrição
name	text	Nome do parâmetro de configuração
setting	text	O valor atual do parâmetro
unit	text	Unidade implícita do parâmetro
category	text	Grupo lógico do parâmetro
short_desc	text	Uma breve descrição do parâmetro
extra_desc	text	Descrição adicional, mais detalhada do parâmetro
context	text	Contexto necessário para definir o valor do parâmetro.
vartype	text	Tipo de parâmetro (bool, enum, integer, real, ou string)
source	text	Fonte do valor do parâmetro atual
min_val	text	Valor mínimo permitido do parâmetro (null para valores não-numéricos)
max_val	text	Valor máximo permitido do parâmetro (null para valores não-numéricos)
enumvals	text[]	Os valores permitidos de um parâmetro de enum (NULL para valores não-ENUM)
boot_val	text	O valor do parâmetro assumido na inicialização do servidor, se o parâmetro não for definido de outro modo
reset_val	text	Valor que o RESET iria redefinir para o parâmetro na sessão atual

## 2.5. Segurança da informação no Postgres

Nesta parte da aula apresentamos funcionalidades ligadas à segurança de banco de dados Postgres. Podemos listar como formas de garantir a segurança dos dados: revogar o acesso do usuário a uma tabela, concessão do acesso de usuário a uma tabela, criação de um novo usuário, impedir temporariamente um usuário se conecte, remover um usuário sem deixar apagar os seus dados, verificar se todos os usuários têm uma senha segura, limitar os poderes de superusuário a usuários específicos, criar auditoria para comandos DDL, integração com LDAP, conectar-se usando SSL e criptografia de dados confidenciais. Antes de vermos algumas dessas questões vamos como segurança já foi cobrada em prova pelos CESPE.



**06) ANO: 2014 BANCA: CESPE ÓRGÃO: ANATEL PROVA: ANALISTA ADMINISTRATIVO - SUPORTE E INFRAESTRUTURA DE TI**

Julgue o item abaixo:

A conexão com o PostgreSQL 9.3 é realizada, por padrão, na porta TCP 5432. Uma das configurações de segurança permitida é o acesso por meio de SSL que é true, por padrão, e é aceito, neste caso, com o uso dos protocolos TCP, IP ou NTP.



**Gabarito E.** Vimos no início da aula que a porta padrão do PostgreSQL é a 5432. Até aqui a alternativa estaria correta.

O PostgreSQL possui a habilidade de usar SSL para proteger as conexões de banco de dados, criptografando todos os dados passados por essa conexão. Usando SSL fazemos com que seja muito mais difícil de capturar o tráfego de banco de dados, incluindo nomes de usuários, senhas e dados confidenciais que são trafegados entre o cliente e o servidor. Uma alternativa ao uso de SSL está em executar a conexão através de uma VPN (Virtual Private Network).

Este método de autenticação utiliza certificados de cliente SSL para realizar a autenticação. E, portanto, só está disponível para conexões SSL. Ao usar esse método de autenticação, o servidor irá requerer que o cliente forneça um certificado válido. Nenhum aviso de senha será enviada para o cliente. O atributo CN (COMMON NAME) do certificado será comparado com o nome do usuário do banco de dados solicitado, e se eles combinam o login será permitido. O mapeamento de nome de usuário pode ser usado para permitir que o CN seja diferente do nome do usuário do banco de dados.

Para obter ou gerar uma chave de servidor SSL, o par de certificados para o servidor, e armazená-los para o diretório de dados do banco de dados atual usamos os arquivos **server.key** e **server.crt**. Ele já pode ser criado em algumas plataformas. Por exemplo, no Ubuntu, postgres está configurado para suportar conexões SSL por padrão.

Vejam que a instalação do PostgreSQL não está configurada por default para conexões SSL apesar de possuir suporte nativo. Outro ponto é que embora seja possível usar SSL tanto para o protocolo IP quanto sobre NTP isso não é utilizado pelo PostgreSQL que concentra seu uso apenas sobre o TCP.

Para utilizar SSL o libpq lê o arquivo de configuração do OpenSSL para todo o sistema. Por padrão, esse arquivo é nomeado openssl.cnf e está localizado no diretório relatado pelo comando openssl version -d. Este padrão pode ser substituído, definindo a variável de ambiente OPENSSL\_CONF para o nome do arquivo de configuração desejado.

Apenas para informação libpq é a interface para o programador de aplicação escrita em C para PostgreSQL. libpq possui um conjunto de funções de biblioteca que permitem os programas clientes passarem comandos para o servidor PostgreSQL e para receber os resultados dessas consultas.

### **2.5.1. Autenticação do cliente**

A autenticação é o processo pelo qual o servidor de banco de dados estabelece a identidade do cliente e, por extensão, determina se o aplicativo cliente (ou o usuário que executa o aplicativo cliente) tem permissão para se conectar com o banco de dados que foi solicitado.

O PostgreSQL oferece diferentes métodos de autenticação para o cliente. O método utilizado para autenticar uma conexão cliente em particular pode ser selecionado com base no endereço de host do cliente, no banco de dados e nos de usuários.

### 2.5.1.1. O arquivo pg\_hba.conf

A autenticação do cliente é controlada por um arquivo de configuração, que tradicionalmente é chamado **pg\_hba.conf** e é armazenado no diretório de dados raiz do banco de dados (HBA significa **autenticação baseada em host**). Um arquivo pg\_hba.conf padrão é criado quando o diretório de dados é inicializado pelo initdb.

O formato geral do arquivo pg\_hba.conf é um conjunto de registros, um por linha. As linhas em branco são ignoradas, assim como qualquer texto após o **caractere de comentário #**. Os registros não podem ter mais de uma linha. Um registro é constituído por um número de campos que são separados por espaços e/ou tabulação. Os campos podem conter espaços em branco se o valor do campo for colocado entre aspas.

Cada registro especifica um **tipo de conexão**, uma faixa de endereço IP de cliente (se for relevante para o tipo de conexão), um nome de banco de dados, um nome de usuário, bem como o **método de autenticação** a ser usado para conexões que utilizam estes parâmetros. O primeiro registro com os dados: tipo de correspondência de conexão, endereço do cliente, banco de dados solicitado e nome de usuário é usado para executar a autenticação. Não há nenhuma possibilidade de múltiplas tentativas, ou seja, se um registro é escolhido e a autenticação falhar, os registros subsequentes não são considerados. Se nenhum registro coincide, o acesso é negado. Abaixo temos um exemplo do arquivo:

#	TYPE	DATABASE	USER	ADDRESS	METHOD
host	all		all	192.168.54.1/32	reject
host	all		all	0.0.0.0/0	gss

### 2.5.1.2. Métodos de autenticação

Apresentamos abaixo a lista de possibilidades de métodos de autenticação disponíveis no PostgreSQL.

**trust** - Permite a conexão incondicionalmente. Este método permite que qualquer pessoa que pode se conectar ao servidor de banco de dados PostgreSQL e se autenticar como o usuário que desejarem, sem a necessidade de senha ou qualquer outra autenticação.

**reject** - Rejeitar qualquer conexão incondicionalmente. Isso é útil para "filtragem" de certos hospedeiros de um grupo, por exemplo, uma linha de reject poderia bloquear um host específico para conexão, enquanto uma linha mais tarde permite que os hosts restantes possam se conectar.

**md5** - Exige que o cliente forneça uma senha double-MD5-hash para autenticação.

**password** - Exige que o cliente forneça uma senha não criptografada para autenticação. Uma vez que a senha é enviada em texto simples através da rede, não deve ser usado em redes não confiáveis.

**GSS** - Use GSSAPI para autenticar o usuário. Este método só está disponível para conexões TCP/IP.

**sspi** - Usa SSPI para autenticar o usuário. O método só está disponível no Windows.

**ident** - Obtém o nome do usuário do sistema operacional do cliente entrando em contato com o servidor e verifica se ele corresponde ao nome do usuário do banco de dados. A autenticação ident só pode ser utilizada em conexões TCP/IP. Quando especificado para conexões locais, peer authentication será utilizado em seu lugar.

**peer** - Obtém o nome de usuário do sistema operacional do cliente e verifica se ele corresponde ao nome do usuário solicitado banco de dados. Isto só está disponível para conexões locais.

**ldap** - Autentica usando um servidor LDAP.

**radius** - Autentica usando um servidor RADIUS.

**cert** - Autentica usando certificados de cliente SSL.

**pam** - autentica utilizando o serviço Pluggable Authentication Modules (PAM) fornecido pelo sistema operacional. Permissões e papeis

O PostgreSQL administra as permissões de acesso ao banco de dados utilizando o conceito de papéis. Podemos pensar em um papel como uma base de dados de usuário, ou um grupo de usuários da base de dados, dependendo de como o papel é configurado. Roles (papeis) podem possuir permissões sobre objetos de banco de dados (por exemplo, tabelas) e podem atribuir privilégios sobre os objetos a outros papéis visando controlar quem tem acesso a quais objetos.

O conceito de papéis une os conceitos de "usuários" e "grupos". Em versões do PostgreSQL anteriores a 8.1, os usuários e os grupos eram tipos de entidades distintas, mas agora há apenas papéis. Qualquer papel pode atuar como um usuário, um grupo, ou ambos. A seguir descreveremos como criar e gerenciar papéis e introduzimos o sistema de privilégios do SGBD.

## **2.5.2. Papeis no banco de dados**

Papéis de bancos de dados são conceitualmente completamente separados dos usuários do sistema operacional. Na prática, pode ser conveniente manter uma correspondência, mas isso não é necessário. Papéis de bancos de dados são globais através de uma instalação de um cluster de banco de dados (e não por

base de dados individual). Para criar um papel use o comando CREATE ROLE SQL:

```
CREATE ROLE nome;
```

O **nome** segue as regras usadas para identificadores SQL: sem caracteres especiais ou aspas duplas. (Na prática, normalmente você vai querer adicionar opções, como LOGIN, para o comando). Para remover uma função existente, use o comando análogo DROP ROLE:

```
DROP ROLE nome;
```

Por conveniência, o executáveis **createuser** e **dropuser** são fornecidos como *wrappers* em torno esses comandos SQL que podem ser chamados a partir da linha de comando shell.

Para determinar o conjunto de papéis existentes, você pode examinar o catálogo do sistema pg\_roles, por exemplo:

```
SELECT rolname FROM pg_roles;
```

O programa psql \du é um meta-comando útil para listar os papéis existentes. Abaixo temos um exemplo da criação de um ROLE e da execução do comando \du no psql. Logo em seguida faremos uma questão pra fixação do conteúdo.

```
postgres=# CREATE ROLE flavia LOGIN SUPERUSER;
CREATE ROLE
postgres=# \du
```

Role name	List of roles Attributes	Member of
flavia	Superuser	{ }
postgres	Superuser, Create role, Create DB, Replication	{ }
rchthiago	Superuser, Create role, Create DB	+ { }
	10 connections	+ { }
	Password valid until 2014-05-03 00:00:00-03	{ }
thiago		{ }

**Dica:** É uma boa prática para criar uma ROLE que tem os privilégios CREATEDB e CREATEROLE, mas não é um SUPERUSER, e depois usar essa ROLE para todo o gerenciamento de rotinas dos bancos de dados e papeis. Essa abordagem evita os perigos de operar como um superusuário em tarefas que realmente não necessitam dele.



**07) BANCA: VUNESP ANO: 2014 ÓRGÃO: SP-URBANISMO PROVA: ANALISTA ADMINISTRATIVO - TECNOLOGIA DA INFORMAÇÃO - INFRAESTRUTURA**

O comando a ser executado no sistema gerenciador de bancos de dados PostgreSQL (v. 9.3), para obter o conjunto de papéis presentes em um banco de dados, é:

- A SELECT oid FROM pg\_cast.
- B SELECT rolconfig FROM pg\_rules.
- C SELECT rolname FROM pg\_cast.
- D SELECT rolname FROM pg\_roles.
- E SELECT rolname FROM pg\_rules.

**Gabarito D.** Observem que a questão procura verificar seu conhecimento a respeito da tabela que armazena as ROLES em um banco de dados Postgres, qual seja, a tabela pg\_roles. Veja que estamos falando de ROLES (papeis) e não de RULES (regras).

Apenas para complementar nosso conhecimento a respeito de ROLES apresentamos abaixo a sintaxe completa do comando. Observe que a maioria dos parâmetros é de entendimento intuitivo.

```
CREATE ROLE name [ [ WITH ] option [ ... ] ]
```

As opções podem ser as seguintes:

```
SUPERUSER | NOSUPERUSER  
| CREATEDB | NOCREATEDB  
| CREATEROLE | NOCREATEROLE  
| CREATEUSER | NOCREATEUSER  
| INHERIT | NOINHERIT  
| LOGIN | NOLOGIN  
| REPLICATION | NOREPLICATION  
| CONNECTION LIMIT connlimit  
| [ ENCRYPTED | UNENCRYPTED ] PASSWORD 'password'  
| VALID UNTIL 'timestamp'  
| IN ROLE role_name [, ...]  
| IN GROUP role_name [, ...]  
| ROLE role_name [, ...]  
| ADMIN role_name [, ...]  
| USER role_name [, ...]  
| SYSID uid
```

## 2.6. Backup e Restore

Apresentamos a seguir três utilitários do PostgreSQL para backup e restauração de banco de dados. O pg\_dump que extrai um banco de dados PostgreSQL para um arquivo script ou outro arquivo. O pg\_dumpall cuja função é extrair um cluster de banco de dados PostgreSQL para um arquivo script. E o pg\_restore o qual restaura um banco de dados PostgreSQL a partir de um arquivo gerado pelo pg\_dump.

O **pg\_dump** é um utilitário para fazer backup de um banco de dados PostgreSQL. Faz backups consistentes mesmo se o banco de dados está sendo usado. O pg\_dump não bloqueia os outros usuários que acessam o banco de dados (leitura ou gravação).

Ele pode ter sua saída em formato de scripts ou arquivos. Dumps de script são arquivos de texto simples que contêm os comandos SQL necessários para reconstruir o banco de dados para o estado em que estava no momento em que foi salvo. Para restaurar a partir de um script, carregue o mesmo no psql. Os arquivos de script podem ser usados para reconstruir o banco de dados até mesmo em outras máquinas com outras arquiteturas; com algumas modificações, até em outros bancos de dados SQL.

Agora vamos falar do **pg\_dumpall** que é outro utilitário para fazer "dumping" de todos os bancos de dados do PostgreSQL em um arquivo script. Este contém comandos SQL que podem ser usados como entrada do psql para restaurar os bancos de dados. Ele faz isso chamando pg\_dump para cada banco de dados em um cluster. O pg\_dumpall também despeja os objetos globais que são comuns a todos os bancos de dados (**Obs: O pg\_dump não salva estes objetos**). Isto inclui atualmente informações sobre os usuários do banco de dados e grupos, *tablespaces* e propriedades, tais como permissões de acesso que se aplicam ao banco de dados como um todo.

Como o pg\_dumpall lê as tabelas de todos os bancos de dados, você provavelmente vai ter de se conectar como um superusuário do banco de dados, a fim de produzir uma "descarga" completa dos dados. Também será necessário o privilégio de superusuário para executar o script salvo, a fim possuir autorização para adicionar usuários e grupos, e para criar bases de dados.

Por fim vamos falar do **pg\_restore**, o último utilitário deste nosso bloco que serve para restaurar um banco de dados PostgreSQL a partir de um arquivo gerado pelo pg\_dump em qualquer um dos formatos. São executados os comandos necessários para reconstruir o banco de dados para o estado em que estava no momento em que foi salvo. Os arquivos de backup permitem que o pg\_restore seja seletivo sobre o que é restaurado, ou mesmo reordenar os itens antes de restaurar. Esses arquivos de exportação são projetados para serem portáteis entre arquiteturas.

O pg\_restore pode operar de dois modos. Se um nome de banco de dados for especificado, pg\_restore se conecta ao banco de dados e restaura o conteúdo do arquivo diretamente no banco de dados. Caso contrário, um script contendo os comandos SQL necessários para reconstruir o banco de dados é criado e gravado em um arquivo ou na saída padrão. Este script de saída é equivalente ao formato de saída de texto sem formatação do pg\_dump.



**08) BANCA: FCC ANO: 2014 ÓRGÃO: TRT - 13ª REGIÃO (PB) PROVA: ANALISTA JUDICIÁRIO - TECNOLOGIA DA INFORMAÇÃO**

Paulo utiliza o `pg_dump` do PostgreSQL para fazer cópia de segurança de um banco de dados. Normalmente faz cópias de segurança no formato `tar` e utiliza o `pg_restore` para reconstruir o banco de dados, quando necessário. O `pg_restore` pode selecionar o que será restaurado, ou mesmo reordenar os itens antes de restaurá-los, além de permitir salvar e restaurar objetos grandes. Certo dia Paulo fez uma cópia de segurança do banco de dados chamado `trt13` para o arquivo `tribunal.tar`, incluindo os objetos grandes. Paulo utilizou uma instrução que permitiu a seleção manual e reordenação de itens arquivados durante a restauração, porém, a ordem relativa de itens de dados das tabelas não pôde ser alterada durante o processo de restauração. Paulo utilizou, em linha de comando, a instrução

A `pg_dump -Ec -h trt13 > tribunal.tar`

B `pg_dump -Ft -b trt13 > tribunal.tar`

C `pg_dump -tar -a trt13 > tribunal.tar`

D `pg_dump -tar -c trt13 > tribunal.tar`

E `pg_dump -Fp -b trt13 > tribunal.tar`

**Gabarito B.** Para fazermos a questão é preciso entender alguns detalhes do comando `pg_dump`, vejamos que ele deu a dica na questão: (1) "permitir salvar e restaurar objetos grandes" e durante a restauração (2) "permitiu a seleção manual e reordenação de itens arquivados durante a restauração, porém, (3) a ordem relativa de itens de dados das tabelas não pôde ser alterada durante o processo de restauração." Vamos então entender o que significa.

Primeiramente para restaurar grandes objetos ou `large objects` é necessário usar o parâmetro `-b`. Este é o comportamento padrão, exceto quando `--schema`, `--table`, ou `--schema-only` é especificado. Nestes casos o `-b` só é útil para adicionar objetos grandes para dumps seletivos.

Quando usado com um dos formatos de ficheiros de arquivo e combinado com o `pg_restore`, o `pg_dump` fornece um mecanismo de arquivamento e transferência flexível. O `pg_dump` pode ser usado para fazer backup de um banco de dados inteiro, em seguida, o `pg_restore` pode ser usado para examinar o arquivo e/ou selecionar as partes do banco de dados que devem ser restauradas.

Os formatos de arquivo de saída mais flexíveis são o formato de "custom" (`-Fc`) e o formato de "diretório" (`-Fd`). Eles permitem que a seleção e a reordenação de todos os itens arquivados, suportam a restauração paralela, e são compactados por padrão. O formato de "diretório" é o único formato que suporta descargas paralelas.

Outros dois formatos podem ser usados. O plano (plain `-Fp`) cuja saída do arquivo é um script SQL em texto plano. O outro seria o `tar` (`-Ft`) que exporta



um arquivo em formato “.tar” adequado para entrada do pg\_restore. O formato tar é compatível com o formato de diretório. Extrair um arquivo tar formatado produz um arquivo no formato de diretório válido. No entanto, o tar-format **não suporta a compressão** e tem um limite de 8 GB para o tamanho das tabelas individuais. Além disso, **a ordem relativa de itens de dados da tabela não pode ser mudada** durante a restauração.

Veja que com essas informações podemos nos aventurar a montar nosso comando pg\_dump:

```
pg_dump -Ft -b trt13 > tribunal.tar
```

Veja que depois do -b definimos a base de dados sobre a qual vamos fazer o dump e em seguida definimos após o operador “>” o arquivo no qual os dados serão gravados. Gabarito confirmado na alternativa B.

Terminamos aqui o assunto teórico da nossa Aula 00. Apresentamos a seguir um conjunto de questões comentadas para fixação do assunto. Esperamos que você esteja curtindo o curso.

## Questões Comentadas

**09) BANCA: VUNESP ANO: 2013 ÓRGÃO: COREN-SP PROVA: ANALISTA - ADMINISTRADOR DE BANCO DE DADOS**

No sistema gerenciador de bancos de dados PostgreSQL (v. 9.1), uma forma de melhorar o desempenho (tuning), quando da inserção de grande quantidade de registros em uma tabela, é

A criar uma nova tabela com o mesmo nome da tabela original e inserir os registros nessa nova tabela.

B criptografar cada registro inserido na tabela com um protocolo de 56 bits.

C desabilitar o autocommit e executar apenas um commit ao final das inserções de registros.

D realizar, obrigatoriamente, um commit a cada novo registro inserido.

E realizar, simultaneamente a cada inserção de novo registro, o backup da tabela.

**Gabarito C.** Questão até certo ponto característica. Se você tem um conhecimento de transações com grandes quantidade de dados é possível algumas atitudes que melhoram o desempenho do sistema. Uma das possibilidades é desabilitar as integridades referencias (IRs), outra é reduzir o número de commits parciais dos dados. Quando tratamos de inserção de quando quantidade de dados é comum desabilitar o autocommit e só fazer um commit ao final.

Se você permitir que a cada inserção exista um commit, o PostgreSQL está fazendo um monte de trabalho adicional para cada linha que é adicionada. Um benefício adicional de fazer todas as inserções em uma transação é que, se a inserção de uma linha vier a falhar, então a inserção de todas as linhas inseridas até este ponto seriam revertidas, então você não vai ser preso a um conjunto parcial dos dados carregado.

**10) ANO: 2013 BANCA: VUNESP ÓRGÃO: COREN-SP PROVA: ANALISTA - SISTEMAS**

O comando do PL/pgSQL que permite obter o efeito de um comando, como por exemplo, os indicadores de status do sistema, é

A GET DIAGNOSTICS ...

B EXCEPTION DIAGNOSTICS ...

C MERGE DIAGNOSTICS ...

D NOTICE DIAGNOSTICS ...

E RAISE DIAGNOSTICS ...

**Gabarito A.** Existem várias formas de determinar o efeito de um comando. O primeiro método é usar o comando **GET DIAGNOSTICS**, que tem a forma:

GET DIAGNOSTICS variável = item [, ...];

Veja que este comando é o gabarito da nossa questão. Ele permite a recuperação dos indicadores de status do sistema. Cada item é uma palavra chave que identifica o valor de estado a ser atribuído à variável especificada. Os itens de status disponíveis atualmente são ROW\_COUNT, o número de linhas processadas pelo último comando SQL enviado para a máquina SQL, e RESULT\_OID, o OID da última linha inserida pelo comando SQL mais recente. Note que RESULT\_OID só é útil depois de um comando INSERT em uma tabela contendo OIDs. Vejamos um exemplo:

GET DIAGNOSTICS integer\_var = ROW\_COUNT;

O segundo método para determinar os efeitos de um comando é verificar a variável especial FOUND, que é do tipo boolean. FOUND é iniciada como falsa dentro de cada chamada de função PL/pgSQL. Ela é definida por cada um dos seguintes tipos de declarações:

A instrução SELECT INTO define FOUND como TRUE quando uma linha é atribuída, falso se nenhuma linha é retornada.

A instrução PERFORM define FOUND como TRUE quando produz (e despreza) uma ou mais linhas, falso se não produz nenhuma linha.

O UPDATE, INSERT e DELETE definem FOUND como TRUE quando pelo menos uma linha é afetada, falso se nenhuma linha é afetada.

A instrução FETCH define FOUND como verdade quando retorna uma linha, falso se nenhuma linha é retornada.

A instrução MOVE atribui a FOUND valor TRUE quando reposiciona com sucesso o cursor, FALSE contrário.

A instrução FOR define FOUND como TRUE quando interage uma ou mais vezes, caso contrário FALSE. Isso se aplica a todas as quatro variantes da instrução FOR.

Outras declarações PL/pgSQL não alteram o estado do FOUND. Note em particular que o EXECUTE muda a saída do GET DIAGNOSTICS, mas não altera o FOUND. FOUND é uma variável local dentro de cada função PL/pgSQL; quaisquer alterações afetam apenas a função atual.

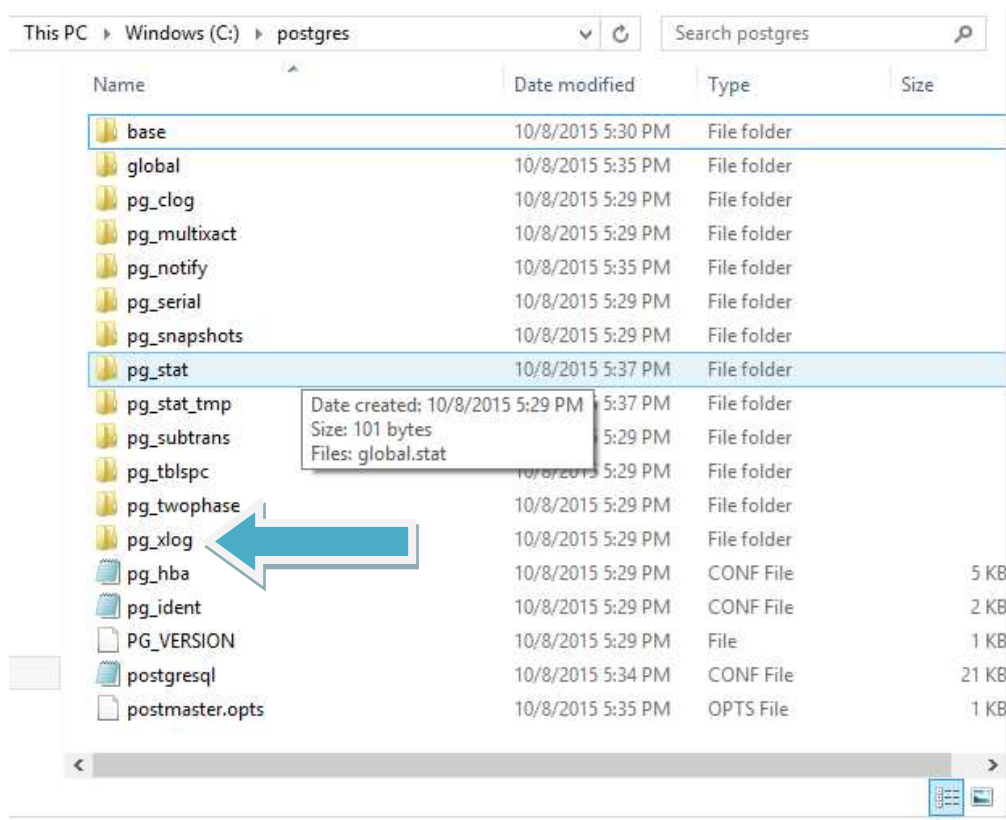


**11) ANO: 2015 BANCA: CESPE ÓRGÃO: CGE-PI PROVA: AUDITOR GOVERNAMENTAL - TECNOLOGIA DA INFORMAÇÃO**

Acerca de bancos de dados, julgue os itens a seguir.

No PostgreSQL 9.3, os arquivos de WAL (write-ahead logging), que armazenam as transações do SGBD na forma de segmentos de log, são gravados por padrão no diretório pg\_wal abaixo do diretório data.

**Gabarito E.** Essa questão é um pouco maldosa, força o candidato a conhecer detalhes da estrutura de pastas de uma cluster de banco de dados do PostgreSQL. Veja a figura abaixo com uma lista de pastas criadas pelas execução do comando initdb.



Observem que existe uma seta que aponta para a pasta pg\_xlog. É essa pasta responsável por armazenar os logs do WAL. Mas o que é WAL?

Write-ahead logging (WAL) é um método padrão para garantir a integridade dos dados. Resumidamente, o conceito central do WAL é que as alterações nos arquivos de dados (onde tabelas e índices residem) devem ser escritas somente após essas alterações serem registradas, ou seja, após os registros que descrevem as alterações serem liberados para armazenamento permanente.

Se seguirmos esse procedimento, nós não precisamos dar um flush das páginas de dados no disco a cada commit de transações, porque sabemos que, no caso de um acidente seremos capazes de recuperar o banco de dados usando o log. Quaisquer alterações que não foram aplicadas nas páginas de dados podem ser refeitas a partir dos registros de log. (Esta recuperação denomina-se roll-forward, também conhecido como REDO.)

**12) BANCA: FCC ANO: 2013 ÓRGÃO: TRT - 5ª REGIÃO (BA) PROVA: ANALISTA JUDICIÁRIO - TECNOLOGIA DA INFORMAÇÃO**

No PostgreSQL, a função utilizada para obter informações sobre arquivos é chamada

- A pg\_header\_info
- B pg\_file\_info
- C pg\_stat\_file
- D pg\_read\_file
- E pg\_file\_access.

**Gabarito C.** Vamos entender cada das alternativas acima. Quais informações cada uma das funções fornecem e sobre exatamente o que? As alternativas A,B e E não fazem parte do conjunto de funções utilizadas para acesso ao arquivos. As alternativas pg\_stat\_file e pg\_read\_file fazem parte das funções genérica de acesso a dados. Vamos então conhecê-las.

As funções mostradas fornecem acesso nativo a arquivos na máquina que hospeda o servidor. Somente os arquivos dentro do diretório de cluster de banco de dados e do log\_directory podem ser acessados. O uso dessas funções é restrito aos superusuários. São quatro as funções:

**pg\_ls\_dir** retorna todos os nomes do diretório especificado, exceto as entradas especiais "." e "..".

**pg\_read\_file** retorna parte de um arquivo de texto, iniciando no offset determinado, retornando na maioria os bytes definidos pelo length (ao menos que o fim do arquivo seja atingido primeiro). Se o offset for negativo, é relativo ao final do arquivo. Se offset e length são omitidos, o arquivo inteiro é retornado. Os bytes lidos do arquivo são interpretados como uma string na codificação (encoding) do servidor; um erro é lançado se eles não são válidos para esta codificação.

**pg\_read\_binary\_file** é semelhante ao *pg\_read\_file*, exceto pelo resultado que é um valor bytea; por conseguinte, o controle de codificação não é executado. Em combinação com a função *convert\_from*, esta função pode ser usado para ler um arquivo em uma codificação especificada:

```
SELECT convert_from(pg_read_binary_file('arquivo_em_utf8.txt'), 'UTF8');
```

**pg\_stat\_file** retorna um registro que contém o tamanho do arquivo, último timestamp de acesso, último timestamp de modificação, timestamp do último status do arquivo de mudança (apenas plataformas Unix), timestamp de criação do arquivo(somente para Windows), e um boolean indicando se é um diretório. Usos típicos incluem:

```
SELECT * FROM pg_stat_file('filename');
```

```
SELECT (pg_stat_file('filename')).modification;
```

Vejam que está última função que mais se adequa ao enunciado e, portanto, torna-se nossa resposta.

## Considerações finais

Chegamos, pois, ao final da aula demonstrativa!

A continuação deste assunto encontra-se na próxima aula. Espero reencontrar você, como um aluno efetivo.

Espero que tenha gostado! Desejo a todos bons estudos! E até breve!

Thiago Cavalcanti

## Referências

Fiz uma lista com alguns links de referências caso você queria se aprofundar um pouco.

- i. Para saber um pouco mais sobre administração de bancos de dados PostgreSQL - <http://www.postgresql.org/docs/9.4/static/admin.html>
- ii. Informações sobre o catálogo de dados do PostgreSQL - <http://www.postgresql.org/docs/9.1/static/catalogs.html>
- iii. Novas funcionalidades do PostgreSQL 9.4 - [https://wiki.postgresql.org/wiki/What's\\_new\\_in\\_PostgreSQL\\_9.4](https://wiki.postgresql.org/wiki/What's_new_in_PostgreSQL_9.4)
- iv. Informações do PostgreSQL na Wikipedia - <https://en.wikipedia.org/wiki/PostgreSQL>



# ESSA LEI TODO MUNDO CONHECE: PIRATARIA É CRIME.

Mas é sempre bom revisar o porquê e como você pode ser prejudicado com essa prática.



1 Professor investe seu tempo para elaborar os cursos e o site os coloca à venda.



2 Pirata divulga ilicitamente (grupos de rateio), utilizando-se do anonimato, nomes falsos ou laranjas (geralmente o pirata se anuncia como formador de "grupos solidários" de rateio que não visam lucro).



3 Pirata cria alunos fake praticando falsidade ideológica, comprando cursos do site em nome de pessoas aleatórias (usando nome, CPF, endereço e telefone de terceiros sem autorização).



4 Pirata compra, muitas vezes, clonando cartões de crédito (por vezes o sistema anti-fraude não consegue identificar o golpe a tempo).



5 Pirata fere os Termos de Uso, adultera as aulas e retira a identificação dos arquivos PDF (justamente porque a atividade é ilegal e ele não quer que seus fakes sejam identificados).



6 Pirata revende as aulas protegidas por direitos autorais, praticando concorrência desleal e em flagrante desrespeito à Lei de Direitos Autorais (Lei 9.610/98).



7 Concurseiro(a) desinformado participa de rateio, achando que nada disso está acontecendo e esperando se tornar servidor público para exigir o cumprimento das leis.



8 O professor que elaborou o curso não ganha nada, o site não recebe nada, e a pessoa que praticou todos os ilícitos anteriores (pirata) fica com o lucro.



Deixando de lado esse mar de sujeira, aproveitamos para agradecer a todos que adquirem os cursos honestamente e permitem que o site continue existindo.