Design e Desenvolvimento de Bancos de Dados
— *PSET 1* —

Prof. Abrantes Araújo Silva Filho

Data de Entrega: 28/05/2023

LEIA COM ATENÇÃO!

Um *Problem Set* (PSET) é um conjunto de problemas e tarefas difíceis (alguns extremamente difíceis) que o forçarão a estudar e realmente compreender a matéria[1]. É **essencial** que você inicie o PSET o mais rápido possível... deixar para começar um PSET faltando um ou dois dias da data de entrega significa que você não conseguirá terminar. Também é **fundamental** que você, em caso de dúvidas ou dificuldades, participe das monitorias e discuta suas dúvidas com o monitor.

Sumário

[1] Para maiores informações sobre os PSETs, leia a Seção 5.3 do *Syllabus* da disciplina.

1 Instruções

Este PSET é uma das atividades **pontuadas** que, conforme detalhado no *Syllabus* (Seções 5 e 6), terão peso de 45% na nota final da disciplina (ou do bimestre). Por favor siga todas as instruções a seguir para a realização e entrega deste PSET.

Este PSET tem quatro grandes objetivos:

1. Fazer com que você aprenda sobre sistemas de controles de versões, em especial o Git (https://git-scm.com/) e o serviço web GitHub (https://github.com);
2. Fazem com que você aprenda sobre Markdown, em geral, e sobre a versão GitHub Flavored Markdown;
3. Fazer com que você aprenda a implementar projetos lógicos de bancos de dados utilizando o PostgreSQL, o Sistemas de Gerenciamento de Bancos de Dados (SGBD) *open source* mais avançado que existe hoje em dia;
4. Fazer com que você reflita sobre diversos problemas que podem ocorrer se o projeto lógico está mal preparado; e
5. Fazer com que você aprenda a Structured Query Language (SQL) em nível básico e intermediário.

Este PSET deve ser feito de **forma individual**, ou seja: cada aluno deve ser responsável por escrever sua própria resposta, estar preparado para demonstrar que esteve envolvido em todos os aspectos do PSET, estar preparado para demonstrar que você foi o criador de todos os códigos preparados.

Atenção para as regras de Integridade Acadêmica!

Por favor, **LEIA ATENTAMENTE** as regras sobre Integridade Acadêmica do *Syllabus* (Seção 7), em especial a "**Política sobre trabalho colaborativo**" (Seção 7.1), para você saber **o que é aceitável fazer ou não** nesta disciplina. Atividades que quebrarem essas regras terão a nota zerada e os alunos encaminhados à coordenação da UVV para aplicação das penalidades previstas (exceto quando o aluno utilizar a "**Cláusula de Arrependimento**", Seçao 7.2 do *Syllabus*).

As atividades deste PSET devem ser realizadas, preferencialmente, na **Máquina Virtual** da disciplina, que já tem o ambiente todo configurado, os softwares instalados e está pronta para uso. A máquina virtual pode ser baixada no site do Computação Raiz (https://www.computacaoraiz.com.br). Caso você opte por não utilizar a máquina virtual, terá de instalar todos os softwares em seu próprio computador e configurar o ambiente por conta própria (é difícil, mas não impossível).

Antes de começar, leia integralmente o PSET para ter uma noção geral do que será exigido, para estimar o grau de dificuldade que você terá, e para estimar quantas

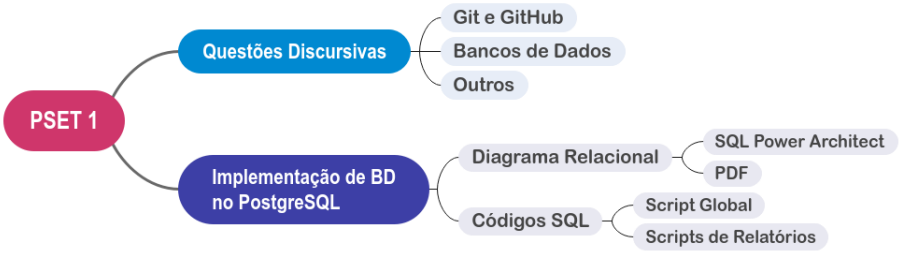
horas por dia você precisará se dedicar ao PSET. Em caso de dúvidas, entre em contato com o professor ou com os monitores o mais rápido possível.

Lembre-se: deixar para começar um PSET faltando apenas um ou dois dias da data de entrega significa que você não conseguirá terminar! Comece cedo, trabalhe um pouco todo dia, mantenha-se atento à data de entrega e tire suas dúvidas com os monitores e/ou professor. Não ultrapasse a data de entrega!

1.1 Visão geral do PSET

De modo geral neste PSET você trabalhará com dois tipos de tarefas: a) responder questões discursivas sobre a matéria; e b) implementar um banco de dados no PostgreSQL, a partir de um diagrama relacional, e escrever códigos SQL para gerar diversos relatórios. A Figura 1 ilustra as tarefas do PSET:

Figura 1: Visão geral do PSET 1



As questões discursivas serão apresentadas ao longo de todo o PSET e devem ser respondidas de acordo com as seguintes regras:

- As respostas devem ser **MANUSCRITAS**, ou seja, respostas impressas ou digitadas não serão aceitas (o aluno ficará com nota zero nas perguntas discursivas);
- As respostas devem ser escritas em **PAPEL ALMAÇO PAUTADO A4** (respostas entregues em outro tipo ou formato de papel não serão recebidas e/ou corrigidas, e o aluno ficará com nota zero nas perguntas discursivas);
- As respostas devem ser escritas utilizando-se **LÁPIS ESCURO** (2B, 4B, 6B). Responsas à caneta são aceitáveis desde que não existam rasuras (a existência de rasuras levará ao desconto de pontos). O uso de lápis de cor para desenhar e/ou ilustrar as respostas é permitido;
- Respostas escritas com o papel almaço do lado errado ou de cabeça para baixo não serão aceitas (o aluno ficará com nota zero nas perguntas discursivas);
- Escreva um **CABEÇALHO** na primeira página do papel almaço, contendo exatamente o seguinte:
  - Banco de Dados, PSET 1

  - Seu nome completo
  - Sua matrícula
  - Sua turma
  - Seu e-mail
- As respostas devem estar em **LETRA LEGÍVEL**, seguindo-se a **NORMA CULTA** da língua portuguesa[2]. Erros gramaticais podem acarretar o desconto de pontos. Respostas ilegíveis, de acordo com o julgamento do professor, podem acarretar na anulação da questão; e
- A apresentação das respostas é importante! Respostas mal organizadas, bagunçadas, rasuradas, sujas, manchadas ou amassadas sofrerão penalidades de pontos.

2 Git e GitHub

2.1 Git

O **Git** (https://git-scm.com) é um dos **Sistema de Controle de Versões** mais utilizados e importantes hoje em dia. Praticamente nenhum desenvolvedor trabalha sem utilizar um bom VCS (do inglês, *Version Control System*).

Um VCS é um software que nos permite controlar todas as alterações em um código fonte (ou demais documentos), nos permite criar ramos (*branches*) separados de codificação para a correção de bugs ou testes de novas funcionalidades, nos permite saber quem, quando e porquê uma determinada alteração no código foi feita e, principalmente, mantém um histórico recuperável de todo o código fonte, desde que a primeira linha de código foi escrita.

Sua primeira atividade neste PSET é aprender mais sobre os VCS, em geral, e sobre o Git, em especial. Você deve fazer o seguinte:

- Leia sobre os VCS na Wikipedia[3] ou procure vídeos no YouTube;
- Procure entender o que são, o que fazem e qual a importância dos diversos VCS;
- Entenda o que são VCS centralizados e distribuídos, comerciais e *open-source*;
- Visite o site do Git[4] e leia diversas páginas do site, procurando entender o Git e suas vantagens;
- Faça o download do livro "Pro Git", de Scott Chacon & Ben Straub, que está disponível integralmente no site do Git. **Leia os capítulos 1 e 2!**, isso é fundamental: para aprender a usar o Git com eficiência, você deve ler esses dois

[2] Você não é obrigado a seguir o novo acordo ortográfico da língua portuguesa, pode escrever de acordo com qualquer boa gramática publicada antes de 1990.

[3] https://en.wikipedia.org/wiki/Version_control

[4] https://git-scm.com

capítulos iniciais do livro![5] No site do Git existe uma versão online traduzida para Português caso você não leia em inglês[6]; e

- Procure vídeos ou tutoriais sobre o Git na internet, para aprender.

2.2 GitHub

Agora que você já sabe e domina o básico do Git, está na hora de aprender a utilizar um dos maiores provedores de servidores Git na internet, o **GitHub** (https://github.com).

Sua próxima atividade é aprender o que é o GitHub: O que ele faz? Para que serve? Como utilizar? GitHub é a mesma coisa que Git? É gratuito ou pago? Faça o seguinte:

- Procure vídeos e tutoriais na internet que ensinem como utilizar o GitHub.
- Visite o GitHub do professor (https://github.com/abrantesasf) e veja alguns dos repositórios públicos que estão lá desde 2012. Note que vários repositórios foram arquivados no Cofre Ártico de Código do GitHub!
- Conheça o Programa de Arquivamento do GitHub (https://archiveprogram.github.com/) e assista (somente em inglês) ao vídeo do Cofre Ártico de Código (https://archiveprogram.github.com/arctic-vault/).

Atenção: algumas entregas deste PSET serão realizadas através de um repositório público no GitHub, que deverá ser criado seguindo-se as regras abaixo:

1. Crie um usuário no GitHub (ou use algum que você já possua);
2. Crie um repositório público com o nome `uvv_bdl_turma`, por exemplo:
   - `uvv_bdl_cc1ma`
   - `uvv_bdl_cc1mb`
   - `uvv_bdl_cc1mc`
   - `uvv_bdl_cc1md`
   - `uvv_bdl_cc1n`
   - `uvv_bdl_si1n`
3. Dentro do repositório crie um diretório chamado de `pset1`. As entregas deste PSET serão feitas nesse diretório;
4. Seu repositório (e qualquer subdiretório) devem estar documentados adequadamente através de arquivos Markdown (ver a seguir); e
5. Ser repositório não pode, em hipótese alguma, ser bagunçado ou não documentado. Lembre-se que seus repositórios GitHub são o seu portfólio de divulgação para futuros clientes e empregadores.

[5] O resto você deve ler à medida que necessitar de funcionalidades mais avançadas em seu trabalho.
[6] E se esse for o seu caso, já passou da hora de estudar inglês: **tudo na computação depende de inglês**.

2.3 GitHub Flavored Markdown

O **Markdown** é uma sintaxe para a formatação de arquivos escritos em texto puro que, posteriormente, podem ser convertidos para HTML.

A partir do Markdown puro (https://daringfireball.net/projects/markdown/) o GitHub adicionou algumas funcionalidades sintáticas adicionais e criou o **GitHub Flavored Markdown**, que é uma versão do Markdown específica para documentar páginas e repositórios no GitHub.

Sua próxima tarefa é aprender a utilizar arquivos Markdown para documentar tudo que você está fazendo durante a resolução deste PSET: você deve documentar o repositório, documentar os subdiretórios, documentar respostas, enfim, tudo que você escrever nos repositórios (exceto os códigos-fontes e demais arquivos de um projeto, como o leiame.txt, etc.) deverá estar no formato Markdown.

Aprenda sobre o GitHub Flavored Markdown em:

- Basic Writing and Formatting Syntax[7]
- Working With Advanced Features[8]
- GitHub Flavored Markdown Spec[9]

Formate o arquivo "`README.MD`" de seu projeto e, usando o que você aprendeu sore o GitHub Markdown, ajuste o título, faça um cabeçalho com seu nome, o nome do professor e da monitora, e crie uma descrição geral. Documente todos os diretórios e subdiretórios de seu repositório!

2.4 Interfaces gráficas para Git e GitHub

O poder do Git está na linha de comando: se você dominar o uso do Git pela linha de comando, terá sempre à disposição tudo que você precisa para usar o Git e o GitHub com eficiência.

Entretanto, alguns usuários iniciantes podem se beneficiar de interfaces mais "amigáveis"[10], tais como:

- GitHub Desktop (https://desktop.github.com): é uma interface gráfica gratuita, para Windows e Mac, desenvolvida pelo próprio GitHub;
- TortoiseGit (https://tortoisegit.org): é uma "extensão" gratuita para o explorador de arquivos do Windows e cria uma integração muito interessante entre o Git, GitHub e Windows Explorer;

[7] https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax

[8] https://docs.github.com/en/get-started/writing-on-github/working-with-advanced-formatting

[9] https://github.github.com/gfm/

[10] Há controvérsias se as interfaces gráficas são mais amigáveis mesmo… eu, por exemplo, acho que elas são confusas e mais atrapalham do que ajudam.

- GitKraken (https://www.gitkraken.com): é uma interface gráfica comercial (mas tem uma versão grauita), para Windows, Linux e Mac. É considerada por muitos como uma das melhores interfaces para o Git e GitHub;
- SmartGit (https://www.syntevo.com/smartgit): é uma interface gráfica comercial (não tem versão gratuita), para Windows, Linux e Mac. Também é considerada como uma das melhores interfaces para o Git e GitHub.

As interfaces citadas acima são apenas algumas das centenas de interfaces disponíveis. Lembre-se: você pode optar por uma interface gráfica mas o verdadeiro poder do Git está na linha de comando.

2.5 Questões discursivas sobre Git e GitHub

Responda às seguintes questões discursivas, nas folhas de papel almaço:

Questão 1: O que são sistemas de controles de versões? Por que são importantes?

Questão 2: Qual a diferença entre o Git e o GitHub? Como eles estão relacionados? É possível usar um sem o outro?

Questão 3: O Git é um sistema distribuído de controle de versões. O que significa isso?

Questão 4: Faça um esquema do fluxo de trabalho do Git, explicando em detalhes o que é o diretório de trabalho, a área de *staging* e o repositório. Como os arquivos se movem entre essas áreas? Como os três estados principais dos arquivos (*commited*, *modified* e *staged*) se relacionam com essas áreas?

3 Implementação do banco de dados no PostgreSQL

3.1 Tudo em scripts!

Nesta seção você fará a implementação de um projeto lógico de banco de dados que simula uma pequena rede de lojas, as "Lojas UVV". Esse banco de dados foi disponibilizado pela Oracle, para treinamento em SQL, e foi levemente modificado para se adequar às necessidades de nosso curso.

A implementação do banco de dados será feita no PostgeSQL[11], o sistema de gerenciamento de bancos de dados *open source* mais avançado que existe atualmente. Algumas regras:

- Tudo que você fizer deve ser feito através de um **script SQL** que você colocará em seu repositório Git/GitHub;

[11] https://www.postgresql.org

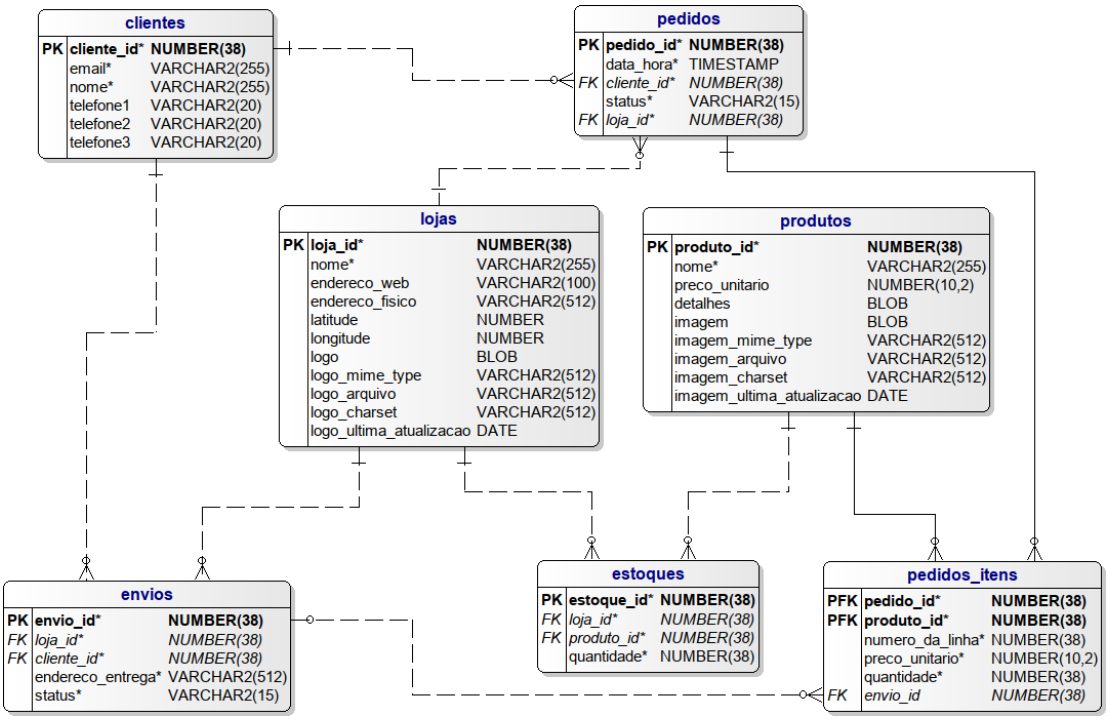
- Esse script deve estar **PRONTO PARA SER EXECUTADO**, de forma integral e **SEM NENHUM ERRO OU INTERRUPÇÃO**. Se o script apresentar erros durante a execução, seu PSET receberá nota zero (tenha certeza de testar o script várias vezes para confirmar que ele funciona sem erros);
- O script SQL deve ser **COMENTADO ADEQUADAMENTE** para que eu saiba o que você está fazendo. Pesquise sobre a maneira correta de comentar um script SQL (há comentários de linha e comentários de blocos, você deverá usar os dois);
- O script SQL deve estar **ORGANIZADO**, seguindo uma seqüência lógica, e bem formatado;
- Descubra como executar o script SQL no PostgreSQL, através da linha de comando (terminal Linux);
- Você deve criar um **único script SQL** para a implementação do banco de dados, com o nome de "`turma_matricula_postgresql.sql`", por exemplo:
  - `ccln_202212345678_postgresql.sql`
  - `siln_202287654321_postgresql.sql`
  - `cclma_202212348765_postgresql.sql`
  - `cclmc_202311148765_postgresql.sql`
- Você pode utilizar softwares de projetos de bancos de dados (como o SQL Power Architect[12] ou o DbSchema Community Edition[13]) para gerar os scripts SQL, mas lembre-se: os scripts gerados por esses softwares geralmente precisam de ajustes finais para ficarem com tudo necessário; também precisam receber comentários para indicar o que cada parte do script faz; não se esqueça de verificar se todas as chaves e checagens de validação estão presentes e funcionando; também não se esqueça de inserir comentários informativos no dicionário de dados (para o banco de dados, para cada tabela e para cada coluna dentro de cada tabela);
- Lembre-se: o seu repositório Git/GitHub é público, qualquer pessoa na internet terá acesso ao seu conteúdo. Não faça trabalhos desorganizados, bagunçados ou desleixados: o GitHub é uma excelente vitrine de projetos para recrutadores, e você não vai querer passar uma má impressão, correto?
- Você terá que pesquisar muita coisa específicas dos SGBD por conta própria, então não deixe para começar este PSET faltando poucos dias da data de entrega: não haverá tempo hábil para você terminar!

[12] https://bestofbi.com/architect-download/
[13] https://dbschema.com/

3.2 Banco de dados: Lojas UVV

O projeto lógico que você implementará no PostgreSQL é o projeto do banco de dados "Lojas UVV" utilizado na 1ª avaliação bimestral. A Figura 2 é o diagrama relacional desse banco de dados:

Figura 2: Lojas UVV



O diagrama acima está disponível como um arquivo de imagem anexo ao PSET, para sua comodidade. Por favor note que o diagrama acima está utilizando tipos de dados do Oracle e você implementará no PostgreSQL. É possível utilizar os mesmos tipos de dados?

Sua primeira tarefa é usar o SQL Power Architect para replicar exatamente o projeto lógico ilustrado acima (mantendo até mesmo erros, se for o caso), no padrão do PostgreSQL.

1. Use o diagrama do projeto (acima e no arquivo de imagem em anexo) e recrie o diagrama relacional no SQL Power Architect, para o PostgreSQL;
2. Crie comentários para o dicionário de dados, incluindo um comentário geral para o banco de dados, um comentário para cada tabela, e um comentário para cada coluna de cada tabela. Você **deve criar seus próprios comentários** para documentar tudo. Obs.: quando você inclui no projeto do SQL Power Architect os comentários adequados (tabelas e colunas, exceto o banco de

dados), esses comentários já serão incluídos automaticamente no script SQL gerado;

3. Coloque o arquivo do projeto, no formato do SQL Power Architect, (com a extensão final "architect") em seu repositório GitHub para avaliação, com o nome de "`turma_matricula_postgresql.architect`"; e
4. Exporte um documento PDF, a partir do SQL Power Architect, com o seu diagrama, e também coloque no repositório GitHub (o nome deve ser o mesmo do arquivo do projeto, apenas com a extensão PDF).

3.3 Questões discursivas sobre o projeto lógico

Agora que você já conseguiu reproduzir o projeto lógico ilustrado na Figura 2, responda às seguintes questões discursivas nas folhas de papel almaço:

Questão 5: Quais erros você consegue encontrar nesse projeto? Erros nas colunas? Erros nas cardinalidades? Erros na identificação dos relacionamentos? Se você encontrar erros no projeto, identifique quais são os erros e explique o que está errado (atenção: não altere o projeto).

Questão 6: Quais tabelas do projeto representam relacionamentos do tipo N:N? Identifique as tabelas e explique porque é um relacionamento N:N; Se não houver relacionamentos N:N, explique se isso seria um erro.

Questão 7: Na tabela "`pedidos_itens`" a coluna identificada com o nome de "`numero_da_linha`" não faz parte da PK composta. Isso está certo ou errado? Ocorreria alguma limitação à funcionalidade do banco de dados ao manter essa coluna fora da PK?

Questão 8 Por que a tabela "`pedidos_itens`" faz relacionamentos identificados com as tabelas `pedidos` e `produtos`, e um relacionamento não identificado com a tabela `envios`?

Questão 9: Qual é o único tipo de relacionamento que pode guardar dados? Por quê? Existe algum relacionamento assim nesse projeto? Se não existir, você sugeriria trocar alguns dos relacionamentos existentes para melhorar o projeto?

Questão 10: Algumas colunas estão utilizando o tipo de dados "BLOB". Que tipo de dado é esse? Quando sua utilização é adequado? O uso desse tipo de dados no projeto é correto? Explique.

3.4 Implementação no PostgreSQL

O PostgreSQL (https://www.postgresql.org) é o SGBD relacional *open-source* mais avançado que existe hoje em dia. Apesar dele ser mais "difícil" que usar do que o MariaBD/MySQL, ele é excelente para estudar e aprender sobre bancos de dados pois ele têm funcionalidades avançadas que não são encontradas em muitos SGBD comerciais.

Sua tarefa é descobrir, por conta própria, como usar o PostgreSQL e como implementar o modelo lógico descrito na Seção 3.2 nesse SGBD. Leia a documentação no site do PostgreSQL, assista a vídeos no YouTube ou peça ajuda para os monitores. O importante é que você aprenda a utilizar o PostgreSQL!

Atenção: você pode utilizar algumas interfaces gráficas que já estão prontas na máquina virtual para trabalhar com o PostgreSQL e testar todos os comandos SQL que serão necessários, tais como o PgAdmin ou o DBeaver, mas o resultado final a ser publicado em seu repositório GitHub é o script SQL que eu devo executar em meu computador. Esse script SQL deve ter **todos os comandos**, na ordem correta e com comentários adequados, para que eu possa simplesmente rodar o script e avaliar o trabalho que você fez.

Estude a documentação do PostrgreSQL e faça o seguinte:

1. Vamos começar com uma tarefa de administração de sistemas Linux, a atualização do sistema. Abra um terminal Linux e torne-se o superusuário (root) com o comando:

   ```
   [computacao@dbserver2 ~]$ su -
   Last login: Tue Mar  7 00:04:34 -03 2023 on pts/0

   [>>> ROOT <<<@dbserver2 ~]#
   ```

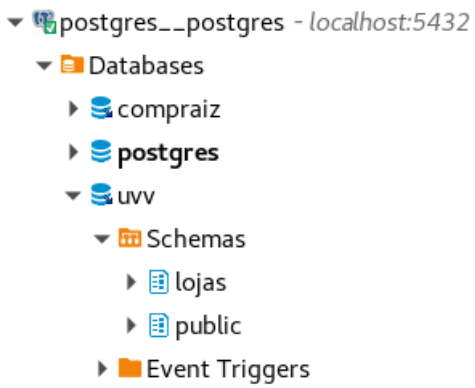
   Como o usuário "computacao" tem permissão para se tornar usuário "root", o Linux não solicita a senha e já mostra diretamente o prompt de superusuário (root). Nesse prompt, digite o seguinte comando:

   ```
   [>>> ROOT <<<@dbserver2 ~]# yum -y check-update; yum -y update
   ```

   Depois de algum tempo (e muitas mensagens esquisitas na tela) o sistema Linux (e diversos outros softwares) serão atualizados (é uma boa prática de segurança manter o sistema sempre atualizado). Se, durante a atualização, o Kernel do Linux foi atualizado, é necessário reiniciar a máquina virtual para que o novo Kernel seja carregado. Faça isso agora: reinicie a máquina virtual e depois continue. Parabéns por ter concluído sua primeira tarefa de administração de sistemas Linux!

2. Crie um usuário específico para ser o "dono" do banco de dados que será criado (utilize o seu nome). Essa é uma boa prática de programação: nós nunca utilizamos os usuários administradores do SGBD para criar bancos de dados de produção. Consulte a documentação do comando `CREATE USER` e não se esqueça de que esse usuário deve ter a permissão de criar bancos de dados, criar outras roles e ter uma senha criptografada.
3. Depois de criar o usuário, faça login no PostgreSQL com esse usuário e crie o banco de dados "uvv" para a implementação do projeto do banco de dados do recursos humanos. Consulte o comando `CREATE DATABASE` na documentação do PostgreSQL e utilize como parâmetros os seguintes valores:
   - nome do banco de dados: uvv
   - owner: o usuário que você criou
   - template: template0
   - encoding: UTF8
   - lc_collate: pt_BR.UTF-8
   - lc_ctype: pt_BR.UTF-8
   - allow_connections: true
4. Agora que você criou um banco de dados, faça uma conexão a esse banco de dados com o usuário que você criou (e que é o dono do banco de dados). Depois de se conectar, crie o esquema "lojas" para armazenar todos os objetos que serão criados com a implementação do projeto lógico. Consulte a documentação do comando `CREATE SCHEMA` para saber como criar o esquema "lojas" e autorizar o seu usuário (aquele que você criou no item 2) a utilizar esse esquema.

   Note que, no PostgreSQL, todo banco de dados recém-criado possui um esquema chamado "public". É nesse esquema público que os objetos do projeto lógico serão criados se não especificarmos algo diferente. Como queremos separar o projeto Lojas UVV de outros projetos que faremos no futuro dentro desse banco de dados chamado "uvv" criaremos o esquema "lojas" para esse projeto. Se você fez tudo direito, deve ver alguma coisa assim (se estiver usando o DBeaver):



Por que essa confusão toda, com bancos de dados e esquemas? Para dar flexibilidade! Nós podemos nos referir a qualquer objeto de qualquer esquema dentro de um banco de dados através da notação de pontos da seguinte maneira: "banco.esquema.objeto". Por exemplo, se quisermos nos referir à tabela "produtos" dentro do esquema "lojas" no banco de dados "uvv", basta escrever: "uvv.lojas.produtos". Mas por que eu iria querer fazer isso? Porque utilizando um esquema eu posso ter diferentes tabelas "produtos" no mesmo banco de dados, cada uma em um esquema diferente!

5. Ajustando o esquema padrão: agora que você já criou o esquema "lojas", perceba que o esquema que é utilizado por padrão no PostgreSQL é o esquema "public", justamente o que nós não queremos usar agora. Faça uma confirmação de que o esquema "public" realmente é o seu padrão com o comando "`SELECT CURRENT_SCHEMA();`". Você verá o esquema público como o padrão. Precisamos mudar isso para o esquema "lojas", caso contrário todo objeto que for criado sem uma qualificação expressa do esquema será criado dentro do esquema público.

   No PostgreSQL a ordem dos esquemas é definida por um parâmetro chamado de `search_path`. Para você ver esse parâmetro, execute o comando "`SHOW SEARCH_PATH;`": você verá algo como: "$user", public. Temos que alterar o search_path para o usuário que você criou. Faça isso com o comando a seguir:

   ```
   SET SEARCH_PATH TO lojas, "$user", public;
   ```

   Para confirmar que o search_path foi alterado, execute de novo o comando "`SHOW SEARCH_PATH;`": você verá que o esquema "lojas" agora aparece na frente de todos e, por isso, ele será o esquema padrão.

   Mas o comando anterior tem uma desvantagem: ele é temporário, ou seja, se você sair do PostgreSQL e voltar depois, o esquema "lojas" não continuará no search_path. Para isso temos que tornar essa mudança permanente para seu usuário, executando o comando abaixo:

   ```
   ALTER USER <nome do usuário>
   SET SEARCH_PATH TO lojas, "$user", public;
   ```

   ATENÇÃO: algumas interfaces gráficas (DBeaver, RazorSQL, PgAdmin e outras) podem ter problemas para identificar o esquema padrão por causa de bugs (sim, isso já ocorreu antes). Então ao utilizar interfaces gráficas, tenha certeza de que você está conectado ao banco de dados correto e ao esquema correto! Utilize os comandos abaixo sempre que precisar:

   - `SHOW SEARCH_PATH;`
   - `SELECT CURRENT_SCHEMA();`
   - `SET SEARCH_PATH TO ...`
   - `ALTER USER ... SET SEARCH_PATH TO ...`

6. Agora que você já está com o usuário, o banco de dados e o esquema prontos, é hora de implementar o projeto lógico do banco de dados Lojas UVV! Volte a se conectar no banco de dados "uvv" e implemente todo o projeto da Figura 2 no PostgreSQL! Para isso:
   - Crie as tabelas com todos os campos, com os tipos de dados e restrições de NOT NULL corretas. ATENÇÃO: você deve incluir comentários nos metadados para descrever todas as tabelas e descrever todas as colunas em todas as tabelas. Consulte os comandos `COMMENT ON TABLE` e `COMMENT ON COLUMN` na documentação do PostgreSQL (você terá que analisar o projeto Lojas UVV e determinar quais os comentários necessários).
   - Crie as PK de cada tabela (cuidado com as PK compostas);
   - Crie os relacionamentos entre as tabelas (das FK para as PK);
   - Crie quaisquer outras restrições de checagem que você achar importante na implementação do projeto, tais como: preço unitário do produto não pode ser negativo; quantidades não podem ser negativas, etc. É você que vai definir todas as restrições de checagem que achar necessárias, com exceção de duas: as restrições da coluna "status" dos pedidos e "status" dos envios devem ser as seguintes:
     - O "status" dos pedidos só pode aceitar os seguintes valores: CANCELADO, COMPLETO, ABERTO, PAGO, REEMBOLSADO e ENVIADO; e
     - O "status" dos envios só pode aceitar os seguintes valores: CRIADO, ENVIADO, TRANSITO e ENTREGUE.

     Outra restrição de checagem que você deve criar é uma restrição específica na coluna "endereco_fisico" na tabela "lojas": você deve criar uma restrição que garanta que pelo menos uma das colunas de endereço (web ou físico) estejam preenchidas, ou seja: apesar dessas colunas não serem obrigatórias, pelo menos uma deles deve estar preenchida. Como fazer isso com uma restrição de checagem?
7. Depois de implementar totalmente o projeto lógico, é hora de você inserir os dados que serão utilizados para gerar os relatórios na próxima seção do PSET. Felizmente você não precisará digitar todos os comandos de inserção por conta própria, basta colocar em seu SCRIPT os comandos INSERT que estão no arquivo anexo.
8. Quanto acabar tudo, prepare o script SQL a ser colocado no GitHub conforme as instruções anteriores. Esse script deve conter todos os comandos utilizados, na ordem correta, incluindo os comandos para a criação de todas as tabelas, chaves e restrições, e também deve incluir todos os comandos de inserção de dados em todas as tabelas! Note que o script deve ser organizado, comentado e pronto para que eu execute em meu computador para verificar e avaliar seu trabalho!

3.5 Questões sobre a implementação no PostgreSQL

Agora que você já conseguiu implementar o projeto lógico no PostgreSQL, responda às seguintes questões discursivas nas folhas de papel almaço:

Questão 11: Qual a diferença entre **banco de dados**, **usuário** e **esquema** no PostgreSQL?

Questão 12: Por que um **esquema** é importante?

Questão 13: Se você não definir um esquema específico, onde os objetos do banco de dados (tabelas, relacionamentos, dados, etc.) serão gravados? Isso é bom ou ruim? Por que?

Questão 14: Agora que você já implementou o projeto no PostgreSQL, tem alguma sugestão de melhoria a fazer para o projeto? Como ele poderia ser melhorado?

3.6 Implementação no MariaBD (opcional)

Agora que você já implementou tudo no PostgreSQL, que tal mostrar que aprendeu de fato? Se você tiver tempo e quiser estudar mais, uma tarefa opcional que você pode fazer é implementar o mesmo banco de dados Lojas UVV, agora usando o MariaDB! Obs.: essa tarefa não é obrigatório e não valerá pontos para o PSET (mas, quem sabe, se você estiver "pendurado" precisando de notas, essa tarefa opcional pode te ajudar!).

Descubra como criar o banco de dados "uvv", como criar seu usuário, como criar o esquema "lojas" e como implementar o projeto lógico da Figura 2 no MariaBD. É possível fazer no MariaBD/MySQL tudo o que o fizemos no PostgreSQL? É muito importante que você consulte a documentação do MariaBD.

Ao final de tudo, prepare um script SQL semelhante ao que você fez para o PostgreSQL, só que agora específico para o MariaBD: o script deve contar todos os comandos, na ordem correta, para que eu crie as tabelas, restrições, chaves, relacionamentos e faça a inserção de dados de forma automática no meu computador para eu avaliar seu trabalho. Coloque esse script no seu repositório GitHub.

LEMBRE-SE: os scripts SQL devem ser organizados e bem comentados!

3.7 Implementação no SQL Server (opcional)

Que tal aproveitar o embalo e implementar o mesmo projeto no SQL Server, da Microsoft? Ele também está na máquina virtual! Se você tiver tempo e quiser estudar mais, descubra como fazer a implementação no SQL Server. Depois prepare um script SQL e coloque no seu repositório GitHub também!

Novamente, essa tarefa é opcional e não vale para a nota oficial do PSET (mas pode te ajudar no futura se estiver "pendurado" precisando de nota).

3.8 Questões finais de implementação

Com tudo o que você já viu sobre o Oracle, o MariaDB e o PostgreSQL, responda à pergunta abaixo:

Questão 15: Faça uma comparação dos SGBD que você utilizou (Oracle, MariaDB e PostgreSQL). Quais as vantagnes e desvantagens de cada um? Quem tem a melhor documentação?

4 Relatórios SQL

Agora que você já está com o banco de dados pronto no PostgreSQL, chegou a hora de trabalhar gerando relatórios com SQL! Seu chefe precisa de diversos relatórios SQL para gerenciar e tomar decisões sobre os produtos e clientes das Lojas UVV. Utilize seus conhecimentos para ajudar seu chefe, preparando os relatórios solicitados a seguir.

(oculto)

5 Como entregar o PSET?

(oculto)