Este tutorial mostra como instalar e executar o Oracle Database Express Edition (XE) via Docker, usando o Ubuntu no WSL2 dentro do Windows 11, com uma imagem oficial da Oracle. Essa configuração é ideal para fins de estudo, testes e desenvolvimento local, sem a necessidade de instalar o Oracle diretamente no sistema operacional.

> **Nota:** Este tutorial não aborda a instalação ou configuração do WSL2 nem do Ubuntu LTS. Estamos considerando que você já tenha o Ubuntu e Docker rodando no WSL2.

Criação do Volume

No Docker, volumes são usados para armazenar dados de forma persistente fora do ciclo de vida dos containers. No caso do Oracle XE, isso significa garantir que os dados do banco não se percam ao reiniciar o ou remover o container. O comando abaixo cria um volume nomeado chamado `oracle-xe-data`, que utilizado para armazenar os arquivos do banco Oracle de forma durável:

```
docker volume create oracle-xe-data
```

Esse volume será montado no container posteriormente, garantindo que os dados persistiam entre reinicialização e recriações do ambiente.

> **Importante:** Quando usado no contexto WSL2(Windows Subsystem for Linux2), esse volume garante que os dados não serão perdidos no mesmo após reiniciar o WSL2 ou próprio Windows. Ou seja, a base de dados continuará intacta após o sistema ser desligado ou reiniciado.

Criando o Contêiner

Após criar o volume para persistência dos dados, o próximo passo é iniciar o container do Oracle XE. Isso é feito por meio do comando `docker run`, que define diversas configurações importantes para o ambiente do banco de dados, como nome do container, portas de acesso, senha do usuário administrador e o volume a ser utilizado. Esse comando também garante que o banco de dados seja reiniciado

automaticamente sempre que o Docker (ou o sistema, como o WSL2) for reiniciado. Abaixo está o comando completo que realiza essa configuração:

```
docker run -d \
  --name oracle-xe \
  --restart unless-stopped \
  -p 1521:1521 \
  -p 5500:5500 \
  -e ORACLE_PWD=MinhaSenha123 \
  -v oracle-xe-data:/opt/oracle/oradata \
  container-registry.oracle.com/database/express:21.3.0-xe
```

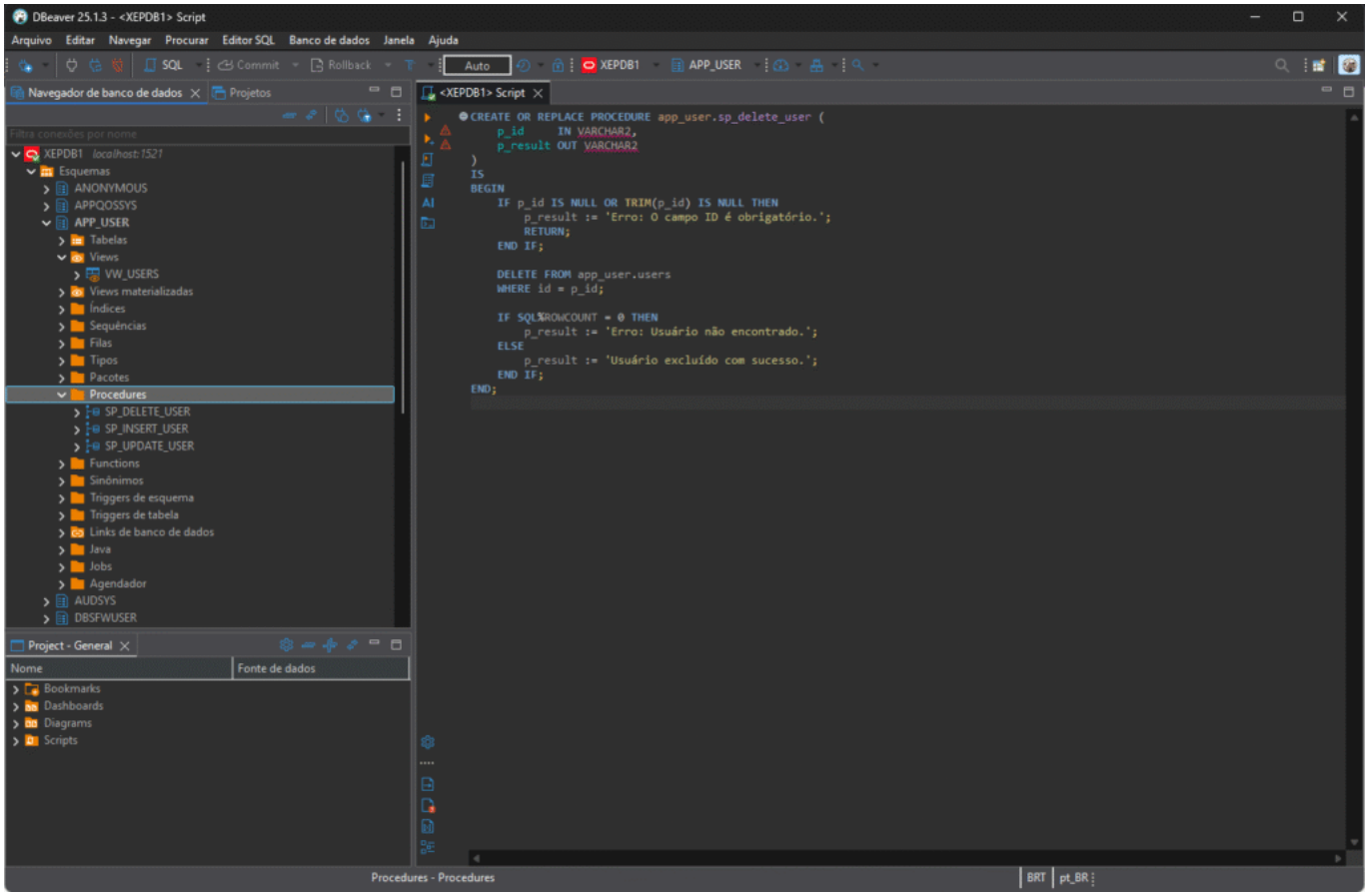
O que esse comando faz:

- `-name oracle-xe`: Nome do container;
- `-p 1521:1521`: Expõe a porta padrão do Oracle;
- `-p 5500:5500`: Acesso ao Enterprise Manager (Gui web);
- `-e ORACLE_PWD=...`: Define a senha do usuário **system**;
- `-v oracle-xe-data:/opt/oracle/oradata`: Monta o volume nomeado `oracle-xe-data` no diretório onde o Oracle XE armazena os dados. Isso garante que as informações do banco sejam preservadas mesmo após reiniciar o container, o WSL2 ou o sistema operacional.
- `Última linha`: Define a imagem **Oracle XE 21c**.

Conectando ao Oracle XE com o DBeaver Community

O **DBeaver Community** é uma ferramenta gráfica gratuita e de código aberto utilizada para acessar e gerenciar bancos de dados. Compatível com uma variedade de SGBDs (incluindo **Oracle**, MySQL, PostgreSQL, SQL Server, entre outros), essa ferramenta oferece uma interface intuitiva que facilita a execução de comandos SQL, criação de objetos no banco (como tabelas, procedures e views), além de visualização e edição de dados.

No contexto deste material, utilizamos o **DBeaver** para conectar ao **Oracle XE** rodando dentro **WSL2** via **Docker**, permitindo uma interação simples e eficaz com o banco de dados diretamente da interface gráfica.



Configurando a conexão DBeaver para acessar o Oracle XE

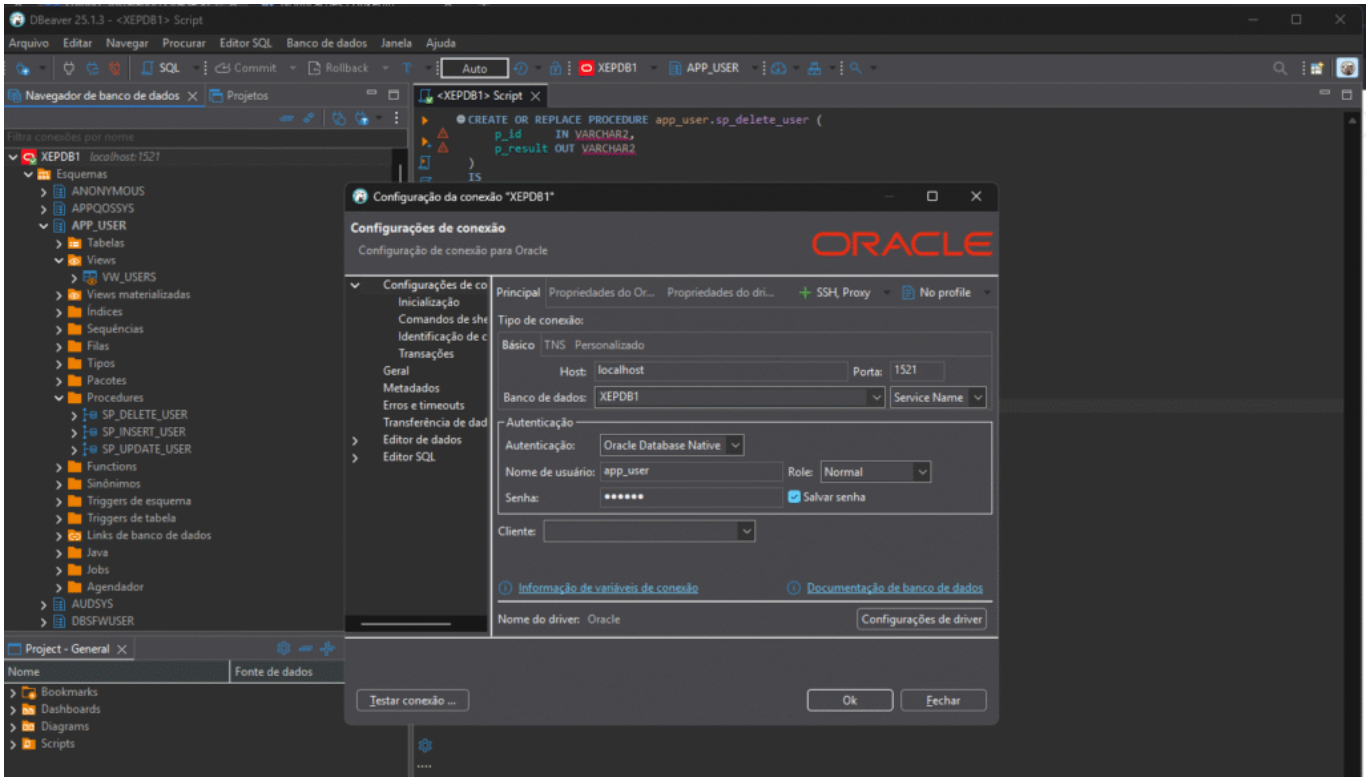
Para que o **DBeaver** consiga se comunicar corretamente com o **Oracle XE** executando em um container **Docker** (via **WSL2**), é necessário configurar manualmente os parâmetros de conexão. essa etapa informa ao **DBeaver** onde o banco está rodando, quais portas estão expostas, qual serviço utilizar e quais credenciais devem ser fornecidas.

Passo a passo:

1. Abra o DBeaver;
2. Clique em Arquivo > Nova Conexão (ou no ícone de nova conexão);
3. Na lista de banco de dados, selecione Oracle e clique em Avançar;
4. Preencha os campos de conexão:
    1. Host: localhost;
    2. Porta: 1521;
    3. Serviço (Service): XEPDB1;
    4. Usuário: system (ou outro usuário criado, como app_user);

5. Senha: a senha definida como parâmetro `ORACLE_PWD` no container (ex: `MinhaSenha123`).

5. Clique em “Testar Conexão” para validar a comunicação com o banco;
6. Se solicitado, permita o download do driver Oracle JDBC;
7. Clique em Concluir. A conexão será adicionada à barra lateral esquerda, pronta para ser utilizada.



Visualizando os containers de forma personalizada

O comando abaixo permite listar apenas as informações mais úteis dos containers em execução, com uma visualização mais limpa e organizada:

```
docker ps --format "table {{.ID}}\t{{.Names}}\t{{.CreatedAt}}\t{{.Status}}\t{{.Ports}}"
```

Esse comando usa a flag `--format`, que permite personalizar as colunas exibidas. Neste caso, estamos pedindo para mostrar os seguintes campos:

- `{{.ID}}`: O identificador único do container;

- `{{.Names}}`: O nome que você deu ao container (exemplo: oracle-xe);
- `{{.CreatedAt}}`: A data e hora em que o container foi criado;
- `{{.Status}}`: O estado atual do container (exemplo: “Up 5 minutes”);
- `{{.Ports}}`: Quais portas estão xpostas/mapeadas para o host.

A palavra `table` no início instrui o Docker a exibi os dados em formato tabular (com cabeçalho), e os `\t` inserem tabulações entre as colunas para melhorar a legibilidade. Esse formato é ideal para uso no terminal durante o desenvolvimento, especialmente quando você quer rapidamente visualizar o status de containers importantes sem excesso de informações.

Por que criar um usuário dedicado para o acesso da aplicação?

Ao trabalhar com banco de dados Oracle(ou qualquer outro), uma boa prática essencial é nunca usar usuários administradores como SYSTEM ou SYS em aplicações. Esses usuários possuem permissões elevadas e são voltados para administração do banco, não para uso em aplicações do dia a dia.

Criar um usuário dedicado, como `app_user`, traz várias vantagens:

- **Segurança**: Limita os acessos ao estritamente necessário, evitando alterações acidentais em objetos críticos do banco;
- **Organização**: Os objetos (tabelas, views, procedures) da aplicação ficam concentrados em um schema próprio;
- **Facilidade de manutenção**: Caso o acesso precise ser revogado, migrado ou auditado, tudo está isolado por usuário;
- **Boas práticas de arquitetura**: Em ambientes profissionais, é comum um banco de dados ter vários usuários(schemas), cada um representando um serviço ou módulo da aplicação.

O comando abaixo cria um novo usuário no banco com a senha `app123` e concede permissões básicas para o usuário se conectar ao banco e criar objetos como tabelas, view e procedimentos.

```
-- Criação do usuário
CREATE USER app_user IDENTIFIED BY app123;

-- Permissões básicas para criar e usar tabelas
GRANT CONNECT, RESOURCE TO app_user;

GRANT CREATE VIEW TO app_user;
```

Você pode se conectar com esse usuário no DBeaver ou via backend .NET usando a seguinte string de conexão:

```
"User Id=app_user;Password=app123;Data Source=localhost:1521/XEPDB1;"
```

Abaixo a tabela de armazenamento de usuários:

```
CREATE TABLE users (
    id VARCHAR2(36) PRIMARY KEY,
    name VARCHAR2(100) NOT NULL,
    email VARCHAR2(150) NOT NULL
);

INSERT INTO users (id, name, email)
VALUES ('550e8400-e29b-41d4-a716-446655440000', 'Alice', 'alice@example.com');
```

Regras de Negócio no Banco de Dados

Em sistemas modernos, é comum que as regras de negócio sejam implementadas na camada do backend, onde frameworks e linguagens de programação oferecem maior flexibilidade e controle. No entanto ainda existem cenários em que parte ou toda a lógica é mantida no banco de dados, especialmente em sistemas legados, ambientes internos mais controlados ou soluções com baixa complexidade, onde centralizar a lógica no banco pode facilitar o controle de integridade e reduzir a dependência de outras camadas. Nesta seção, veremos um exemplo prático e didático de como encapsular regras de negócio diretamente no **Oracle** usando **PL/SQL**, com stored procedures responsáveis por operações como inserção, atualização, exclusão e consulta de dados.

Inserção com PL/SQL

A stored procedure `sp_insert_user` demonstra como implementar a inserção de dados com validação diretamente no banco de dados, centralizando a regra de negócio no Oracle. Embora essa abordagem não seja comum em arquiteturas modernas voltadas à alta escalabilidade, ela ainda pode ser útil em sistemas

internos, legados ou como exercício didático para ilustrar o uso do PL/SQL para controle de integridade e regras no lado do banco.

```
CREATE OR REPLACE PROCEDURE app_user.sp_insert_user (
    p_id     IN VARCHAR2,
    p_name   IN VARCHAR2,
    p_email  IN VARCHAR2,
    p_result OUT VARCHAR2
) AS
BEGIN
    -- Validações básicas
    IF p_id IS NULL OR TRIM(p_id) IS NULL THEN
        p_result := 'O campo ID é obrigatório.';
        RETURN;
    ELSIF p_name IS NULL OR TRIM(p_name) IS NULL THEN
        p_result := 'O campo Nome é obrigatório.';
        RETURN;
    ELSIF p_email IS NULL OR TRIM(p_email) IS NULL THEN
        p_result := 'O campo E-mail é obrigatório.';
        RETURN;
    END IF;

    -- Inserção no banco
    INSERT INTO app_user.users (id, name, email)
    VALUES (p_id, p_name, p_email);

    p_result := 'Usuário inserido com sucesso.';

EXCEPTION
    WHEN OTHERS THEN
        p_result := 'Erro: ' || SQLERRM;
END;
```

Alteração:

```
CREATE OR REPLACE PROCEDURE app_user.sp_update_user (
    p_id     IN VARCHAR2,
```

```
    p_name   IN VARCHAR2,
    p_email  IN VARCHAR2,
    p_result OUT VARCHAR2
)
IS
BEGIN
    -- Validações básicas
    IF p_id IS NULL OR TRIM(p_id) IS NULL THEN
        p_result := 'Erro: O campo ID é obrigatório.';
        RETURN;
    ELSIF p_name IS NULL OR TRIM(p_name) IS NULL THEN
        p_result := 'Erro: O campo Nome é obrigatório.';
        RETURN;
    ELSIF p_email IS NULL OR TRIM(p_email) IS NULL THEN
        p_result := 'Erro: O campo E-mail é obrigatório.';
        RETURN;
    END IF;

    -- Atualização no banco
    UPDATE app_user.users
    SET name = p_name,
        email = p_email
    WHERE id = p_id;

    -- Verifica se algum registro foi atualizado
    IF SQL%ROWCOUNT = 0 THEN
        p_result := 'Erro: Usuário não encontrado.';
    ELSE
        p_result := 'Usuário atualizado com sucesso.';
    END IF;
END;
```

Exclusão:

```
CREATE OR REPLACE PROCEDURE app_user.sp_delete_user (
    p_id     IN VARCHAR2,
    p_result OUT VARCHAR2
)
IS
BEGIN
    IF p_id IS NULL OR TRIM(p_id) IS NULL THEN
        p_result := 'Erro: O campo ID é obrigatório.';
```

```
        RETURN;
    END IF;

    DELETE FROM app_user.users
    WHERE id = p_id;

    IF SQL%ROWCOUNT = 0 THEN
        p_result := 'Erro: Usuário não encontrado.';
    ELSE
        p_result := 'Usuário excluído com sucesso.';
    END IF;
END;
```

View:

```
CREATE OR REPLACE VIEW app_user.vw_users AS
SELECT
    id,
    name,
    email
FROM
    app_user.users;
```