Testes de Integração

Aula 05 — O banco de verdade

Aula 05 · 11/11/2026 · Indicadores 1, 3

UniSenac Centro Universitário RS
Senac
Fecomércio
IFEP
Sesc
Sindicatos Empresariais

O mapa de hoje

- **Bloco 1** — o teste com dublê que passou com o SQL errado
- **Bloco 2** — um banco só para os testes, limpo antes de cada um
- **Bloco 3** — alterar, apagar, e o banco recusando o que não pode
- **Bloco 4** — mão na massa: testes de integração no CRUD **de vocês**

> Hoje o pgAdmin volta. Abram o PostgreSQL de vocês e confiram que lembram a senha do usuário `postgres` — o teste vai precisar dela.

Retomando a Aula 04

- Por que o dublê entra pelo construtor, e não é criado dentro da classe?
- O WhatsApp caiu no primeiro cliente. Como o `side_effect` mostrou isso sem derrubar o WhatsApp de verdade?
- O que **não** se mocka?

Bloco 1

Devemos mockar o CRUD?

O balcão pediu: achar o jogo pelo título

```
def buscar_por_titulo(conexao, titulo):
    cursor = conexao.cursor()
    try:
        cursor.execute(
            "SELECT id_jogo, titulo, genero, ano_lancamento, preco_diaria "
            "FROM jogo WHERE titlo = %s",
            (titulo,),
        )
        linha = cursor.fetchone()
        if linha:
            return _linha_para_jogo(linha)
    except pg.DatabaseError as e:
        print(e); conexao.rollback()
    return None
```

O teste, do jeito da aula passada

O banco é uma dependência. Então, um dublê no lugar da conexão:

```
def test_busca_pelo_titulo_com_cursor_falso():
    conexao = Mock()
    cursor = conexao.cursor.return_value
    cursor.fetchone.return_value = (1, "Zelda", "Aventura", 2017, Decimal("8.00"))

    jogo = buscar_por_titulo(conexao, "Zelda")

    assert jogo.titulo == "Zelda"
```

Palpite antes de rodar: verde ou vermelho?

⚠ Verde. E o SQL?

- O teste passou.
- Agora leiam o SQL do slide anterior com calma: `WHERE titlo = %s`. A coluna se chama `titulo`.
- O PostgreSQL recusaria esse comando na primeira linha. O dublê **nunca leu o SQL** — ele devolve a linha que a gente mandou devolver, para qualquer comando.
- O teste conferiu a única coisa que não podia dar errado, e deixou passar a única que podia.

Quando o trabalho é falar com o banco

- O `CobradorDeAtrasos` tem **regra** — quem atrasou, quando avisar. O banco é só de onde vêm os dados. Ali o dublê serve.
- O `buscar_por_titulo` não tem regra nenhuma. O trabalho **inteiro** dele é conversar com o banco.
- Se a função existe para falar com outro sistema, o único teste que prova que ela funciona é fazê-la falar com ele **de verdade**.

GLOSSÁRIO

`unit test`	teste de unidade — a regra, isolada do resto, com dublê no que é de fora
`integration test`	teste de integração — o código junto do sistema real com que ele conversa

Checagem do Bloco 1

Para cada uma, teste de unidade (com dublê, se precisar) ou teste de integração?

- `multa(dias_atraso)`
- o `CobradorDeAtrasos`, que decide quem avisar
- `criar_jogo(conexao, ...)`, que faz o `INSERT`
- `deletar_jogo(conexao, id)`

E: por que o teste com cursor falso passou com `titlo` no SQL?



`forms.gle/CYwhtJvLJ6hstC5q8`

Intervalo

10 minutos

Bloco 2

Um banco só para os testes

Tirar o CRUD do notebook

No `crud.ipynb` da UC2, a conexão era criada numa célula e usada por todas as funções:

```
conexao = pg.connect(host="localhost", dbname="locadora", ...)   # célula 5
cursor = conexao.cursor()

def criar_jogo(titulo, genero, ano_lancamento, preco_diaria): ...
```

- O teste não tem como dizer "use **outro** banco": a conexão está presa lá dentro.
- E o menu com `input()` no fim do notebook trava qualquer execução automática.
- O conserto é o mesmo da aula passada: a dependência entra **pela porta da frente**.

```
def criar_jogo(conexao, titulo, genero, ano_lancamento, preco_diaria): ...
```

Um banco que pode ser apagado

1. No pgAdmin: **Databases** → **Create** → **Database…**, nome `locadora_teste`
2. Com o `locadora_teste` selecionado, abram o **Query Tool** e rodem o `locadora-demo.sql` da UC2 — ele cria as tabelas
3. No `conftest.py` do site, troquem `SUA_SENHA` pela senha de vocês
4. Ponham `crud_jogo.py`, `conftest.py` e o arquivo de teste na mesma pasta

> **Nunca** o banco do projeto. O teste apaga as tabelas antes de cada execução — apontado para o banco errado, ele apaga os dados de vocês.

`conftest.py` — a conexão de cada teste

```python
@pytest.fixture
def conexao():
    conexao = pg.connect(**BANCO_DE_TESTE)
    cursor = conexao.cursor()
    cursor.execute("TRUNCATE jogo, plataforma RESTART IDENTITY CASCADE")  # zera ANTES
    conexao.commit()
    yield conexao       # o teste roda aqui, começando do zero
    conexao.close()     # e isto, quando ele termina
```

GLOSSÁRIO

`conftest.py`	o arquivo de preparação — o `pytest` lê sozinho, pelo nome
`yield`	entregar — "use isto; quando terminar, eu continuo daqui"
`TRUNCATE ... CASCADE`	esvaziar a tabela, e junto as que apontam para ela

O primeiro teste de integração

```python
def test_cria_e_busca_pelo_id(conexao):
    criado = criar_jogo(conexao, "Zelda", "Aventura", 2017, "8.00")

    achado = buscar_jogo(conexao, criado.id_jogo)

    assert achado.titulo == "Zelda"
    assert achado.preco_diaria == Decimal("8.00")
```

- O cenário agora é montado **no banco**: o jogo é gravado de verdade, antes da busca.
- Façam o mesmo para o `buscar_por_titulo`. Palpite: e agora, verde ou vermelho?

⚠ Erro comum — o teste que só passa na primeira vez

> Limpar com `rollback` no fim do teste parece natural. Mas o `criar_jogo` já deu `commit()` lá dentro — e o que foi gravado, o `rollback` não desfaz.

Limpeza	1ª rodada	2ª rodada
`rollback` no fim	3 vermelhos	4 vermelhos
`TRUNCATE` no começo	verde	verde

- O "Zelda" de um teste continua no banco no teste seguinte, e o `UNIQUE` do título recusa o segundo.
- Teste bom roda **duas vezes seguidas** com o mesmo resultado. Rodem duas vezes antes de entregar.

Checagem do Bloco 2

Uma colega escreveu esta fixture, e o CRUD dela dá `commit()` em toda função:

```python
@pytest.fixture
def conexao():
    conexao = pg.connect(**BANCO_DO_PROJETO)
    yield conexao
    conexao.rollback()
```



forms.gle/96XRKBVXSFj7af1M8

- Que **dois** problemas ela tem?
- Reescreva as linhas que precisam mudar

Intervalo

10 minutos

Bloco 3

Alterar, apagar e ser recusado

Depois do `DELETE`, a busca não acha

```python
def test_apaga_jogo(conexao):
    criado = criar_jogo(conexao, "Zelda", "Aventura", 2017, "8.00")

    assert deletar_jogo(conexao, criado.id_jogo) is True
    assert buscar_jogo(conexao, criado.id_jogo) is None
```

- Não basta a função dizer que apagou. Quem confere é uma **busca** no banco, depois.
- O mesmo vale para o `UPDATE`: altera, e depois busca para ver o valor novo.

E o jogo que tem plataforma?

O Zelda roda no Switch. O cenário é montado no banco, com o vínculo:

```
    cursor = conexao.cursor()
    cursor.execute("INSERT INTO plataforma (nome) VALUES ('Switch') RETURNING id_plataforma")
    id_plataforma = cursor.fetchone()[0]
    cursor.execute("INSERT INTO jogos_plataforma (id_jogo, id_plataforma) VALUES (%s, %s)",
                   (criado.id_jogo, id_plataforma))
    conexao.commit()

    assert deletar_jogo(conexao, criado.id_jogo) is True
```

Palpite: o `deletar_jogo` do `crud_jogo.py` apaga esse jogo? Pensem na chave estrangeira da Aula 11 da UC2.

O banco recusando o que não pode

```python
def test_banco_recusa_preco_negativo(conexao):
    assert criar_jogo(conexao, "Zelda", "Aventura", 2017, "-1.00") is None

def test_banco_recusa_titulo_repetido(conexao):
    criar_jogo(conexao, "Zelda", "Aventura", 2017, "8.00")
    assert criar_jogo(conexao, "Zelda", "RPG", 2020, "5.00") is None
```

- As constraints que vocês escreveram na UC2 — `CHECK`, `UNIQUE`, `NOT NULL`, `REFERENCES` — também se testam.
- Se alguém apagar o `CHECK` da tabela um dia, este teste fica vermelho no mesmo dia.

Checagem do Bloco 3

- Escreva o teste: atualizar o preço de um id que **não existe** devolve `False`
- No teste do jogo com plataforma, por que o `INSERT` do vínculo vai no **teste**, e não no `conftest.py`?
- Que constraint do banco do **seu** projeto você testaria primeiro? Por quê?



`forms.gle/mhn7ux2p4EQXhToZA`

Intervalo

10 minutos

MÃO NA MASSA

O CRUD de vocês, no banco de teste

1. **Parte A:** testes de integração para o `crud_jogo.py` do site — criar, buscar pelo id e pelo título, alterar, apagar com e sem plataforma. Achem os defeitos e consertem
2. **Parte B:** tirem do notebook o CRUD da entidade principal do projeto de vocês, para um `.py` com a conexão por parâmetro
3. Criem o banco `<projeto>_teste` com o `.sql` de criação de vocês, e um `conftest.py` que esvazie as tabelas
4. Um teste para cada uma das 4 operações, e um em que o banco **recusa** um dado inválido
5. Rodem a suíte inteira **duas vezes seguidas**

MÃO NA MASSA

O combinado desta entrega

GLOSSÁRIO

`test_grupoNN_aulaNN.py` `test_grupo03_aula05.py`

O que eu confiro: Os testes rodam no banco **de teste**, nunca no do projeto · Cada operação conferida por uma **busca depois**, não só pelo `True` que a função devolve · Pelo menos um teste em que o banco **recusa** · A suíte verde **duas vezes seguidas** · No fim do arquivo, o **registro**: que teste achou que defeito, e o que mudou

> Eu passo de mesa em mesa. Deixem o painel Testing aberto e o pgAdmin no banco de teste — eu vou pedir para rodar duas vezes na minha frente.

PARA CASA · OPCIONAL

Antes da próxima aula

Testar em casa: termine os testes das 4 operações do CRUD do grupo. Rode duas vezes seguidas.

Tarefa opcional: escolha a tabela do projeto que tem chave estrangeira e escreva o teste de apagar um registro que ainda é citado por outra tabela. O CRUD de vocês trata isso?

Para treinar: `crud_jogo.py` e `conftest.py` estão no site, na página da aula.

Para ver acontecer: demonstrações da Aula 05 — ver o cursor falso aprovar o SQL que o banco recusa, e rodar a suíte duas vezes com rollback e com TRUNCATE — e oito perguntas com resposta na hora.

Para a próxima aula

Vocês testaram regra, dublê e banco. Como mostrar o que foi testado — e o que falhou — para quem **não lê código**?

UniSenac Centro Universitário RS
Senac
Fecomércio
IFEP
Sesc
Sindicatos Empresariais