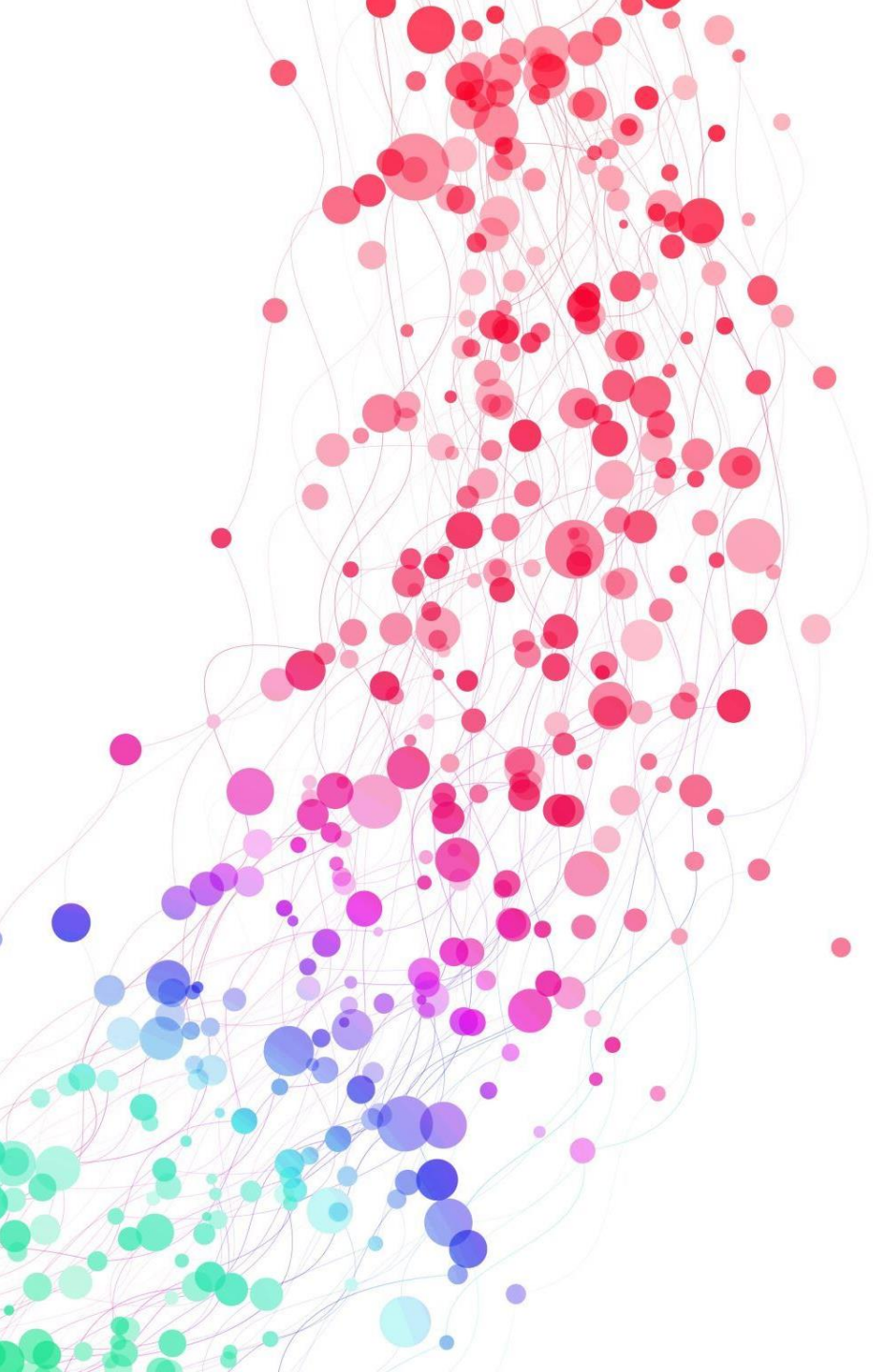

Banco de Dados para Automação de Testes

PROF MSC RANDERSON MELVILLE

O que é um banco de dados?

Um **banco de dados** é uma coleção organizada de dados que podem ser facilmente acessados, gerenciados e atualizados.

Ele armazena informações de maneira estruturada para que possam ser recuperadas e manipuladas de forma eficiente.

Tipos e importãncia

Relacionais: Organizam dados em tabelas que podem ser relacionadas entre si. Exemplos: MySQL, PostgreSQL, Oracle.

Não relacionais (NoSQL): Armazenam dados de forma mais flexível, como documentos, chave-valor ou grafos. Exemplos: MongoDB, Redis, Cassandra.

Importância dos bancos de dados em aplicações modernas:

Facilitam o armazenamento e recuperação de grandes volumes de dados.

São essenciais para o funcionamento de sistemas de informação, aplicativos web, e-commerce, entre outros.

Garantem integridade e segurança dos dados.

SGBDs

Exemplos de SGBDs (Sistemas de Gerenciamento de Banco de Dados) populares:

MySQL: Banco de dados relacional amplamente utilizado em aplicações web.

PostgreSQL: Um SGBD avançado e open-source que suporta SQL e consultas complexas.

MongoDB: Um banco de dados NoSQL orientado a documentos, ideal para grandes volumes de dados não estruturados.

Bancos de Dados Relacionais

Tabelas, Linhas e Colunas:

Tabelas: Estrutura básica em bancos de dados relacionais que organiza dados em linhas e colunas.

Linhas: Representam um registro individual de dados dentro de uma tabela.

Colunas: Representam os atributos ou campos dos dados, como "Nome", "Idade", "Endereço", etc.

Chave Primária e Chave Estrangeira:

Chave Primária (Primary Key): Um identificador único para cada registro em uma tabela. Garante que não existam duplicatas.

Chave Estrangeira (Foreign Key): Um campo que cria uma relação entre duas tabelas, referenciando a chave primária de outra tabela. Garante a integridade referencial entre os dados.

SQL (Structured Query Language):

Introdução ao DDL (Data Definition Language): Conjunto de comandos usados para definir e modificar a estrutura de banco de dados.

- CREATE: Cria novas tabelas, bancos de dados ou índices.
- ALTER: Modifica a estrutura de uma tabela existente.
- DROP: Remove tabelas ou bancos de dados.

DML (Data Manipulation Language): Conjunto de comandos usados para manipular os dados dentro das tabelas.

- SELECT: Consulta dados de uma ou mais tabelas.
- INSERT: Insere novos dados em uma tabela.
- UPDATE: Atualiza dados existentes em uma tabela.
- DELETE: Remove dados de uma tabela.

Noções Básicas de SQL para Testes

Como criar consultas simples para validação de dados:

Utilizar comandos SQL para verificar se os dados armazenados no banco estão corretos e consistentes.

As consultas mais comuns são **SELECT** para visualizar dados, **INSERT** para adicionar novos registros, **UPDATE** para modificar registros existentes e **DELETE** para remover registros.

SQL

Operações mais comuns:

SELECT: Recupera dados de uma tabela com base em critérios específicos. Exemplo: `SELECT * FROM Clientes WHERE Idade > 18;`

INSERT: Insere um novo registro em uma tabela. Exemplo: `INSERT INTO Pedidos (ClienteID, Valor) VALUES (1, 100.50);`

UPDATE: Atualiza os valores de um ou mais registros em uma tabela. Exemplo: `UPDATE Clientes SET Nome = 'Carlos' WHERE ClienteID = 1;`

DELETE: Remove um ou mais registros de uma tabela. Exemplo: `DELETE FROM Pedidos WHERE PedidoID = 10;`

Testes com Bancos de Dados

Qual o papel dos bancos de dados em automação de testes?

Testes de integração: Verificam se o sistema interage corretamente com o banco de dados, garantindo que as operações de leitura/escrita de dados funcionam corretamente.

Verificação de dados persistidos (Data Assertions): Testes que validam se os dados no banco de dados correspondem aos dados esperados após uma operação.

Testes de performance: Avaliam a eficiência e a velocidade das consultas em cenários de grande volume de dados.

Verificação de transações: Testes que garantem que as transações no banco de dados são concluídas com sucesso e que a integridade dos dados é mantida.

Estratégias de Testes com Banco de Dados

Testes de Integração:

- Garantem que a comunicação entre a aplicação e o banco de dados funcione corretamente.
- Exemplo: Testar se, após uma operação de "Compra", o banco de dados atualiza o estoque corretamente.

Testes de Performance:

- Avaliam a rapidez das consultas SQL e a capacidade do banco de dados de lidar com grandes volumes de dados.
- Exemplo: Testar quanto tempo uma consulta complexa leva para ser executada em uma base de dados com milhões de registros.

Verificação de Migrations:

- Testes que garantem que as alterações na estrutura do banco de dados (como a adição de novas colunas ou tabelas) são aplicadas corretamente e não causam erros.

Automação de Testes com Banco de Dados

Integração de automação de testes com bancos de dados:

- Utilizar ferramentas e frameworks para criar testes automatizados que interajam diretamente com o banco de dados.
- **JDBC (Java Database Connectivity)**: Um API Java que permite a execução de consultas SQL a partir de uma aplicação Java.
- **PyMySQL (Python)**: Biblioteca Python para interagir com bancos de dados MySQL.
- **Active Record (Ruby)**: Biblioteca usada no Ruby on Rails que facilita a interação entre o código Ruby e o banco de dados relacional.

Criando testes que interagem diretamente com o banco:

- Exemplo: Escrever testes que inserem um novo registro no banco de dados e verificam se ele foi persistido corretamente.

Ferramentas e bibliotecas úteis:

JUnit + DBUnit: JUnit é um framework de testes Java, e DBUnit permite executar testes em bancos de dados.

TestContainers: Biblioteca que permite rodar bancos de dados isolados em containers Docker durante os testes.

Docker: Ferramenta para criar ambientes de banco de dados em containers para testes.

Dicas e Boas Práticas

Evitar testes dependentes de banco de dados:

Sempre que possível, use mocks ou stubs para simular a interação com o banco de dados durante os testes, evitando dependências externas que possam causar flutuações nos resultados.

Manter os dados de teste consistentes:

Assegure-se de que os dados de teste sejam criados e removidos corretamente antes e depois de cada teste, para evitar interferências entre testes.

Dicas e Boas práticas

Criar dados de teste específicos:

Gere dados de teste que representem cenários reais e específicos, evitando dados genéricos. Isso ajuda a garantir que os testes sejam mais eficazes.

Ferramentas para criação de cenários de dados de teste:

Faker: Biblioteca que gera dados de teste falsos como nomes, e-mails, números de telefone, etc.

FactoryBoy: Biblioteca Python usada para criar objetos de teste (fixtures) de forma automatizada.