Aula 04: Modelagem de Dados e Junções

A modelagem de dados é o processo de organizar e estruturar os dados de uma forma que eles possam ser armazenados e gerenciados de maneira eficiente em um banco de dados. É como se estivéssemos ***desenhando o mapa de como os dados se relacionam*** entre si.

Vamos entender os conceitos-chave:

- **Chave Natural:** Um atributo que já existe no mundo real e pode ser usado como identificador único, como o CPF de uma pessoa ou o número de série de um produto.
- **Chave Candidata:** Uma ou mais colunas que podem ser candidatas a chaves primárias, ou seja, que podem identificar unicamente cada linha da tabela.
- **Chave Primária (PK - Primary Key):** É uma das chaves candidatas escolhida para ser o identificador único de cada linha na tabela. Ela **não pode ter valores duplicados** e **não pode ser nula**.
- **Chave Estrangeira (FK - Foreign Key):** É uma coluna em uma tabela que referencia a chave primária de outra tabela. Ela é o elo que conecta as tabelas e estabelece relacionamentos.
- **Cardinalidade:** Descreve a relação entre duas tabelas. Por exemplo, uma pessoa pode ter `1` ou `muitos` pedidos (`1` para `muitos`).

Também temos as **restrições de integridade**, que garantem que os dados no banco de dados sejam precisos e consistentes:

- **Integridade da Entidade:** Garante que a chave primária seja única e não nula.
- **Integridade Referencial:** Garante que a chave estrangeira em uma tabela corresponda a um valor de chave primária válido em outra tabela, impedindo a criação de "registros órfãos".

Atividade Guiada: Banco de Dados Relacional

Imagine uma loja virtual, um e-commerce. Precisamos de tabelas para `clientes` e `pedidos`, não apenas produtos que temos. Inicie o **Apache**, o **MySQL** e acesse o **PhpMyAdmin**.

1. Utilize o banco de dados `vendas_online` da aula anterior;

```
USE vendas_online;
```

2. Crie as tabelas `clientes` e `pedidos` e as relacione:

```
CREATE TABLE clientes (
    id_cliente INT PRIMARY KEY,
    nome VARCHAR(255),
    email VARCHAR(255)
);
```

```
CREATE TABLE pedidos (
    id_pedido INT PRIMARY KEY,
    data_pedido DATE,
    valor_total DECIMAL(10, 2),
    id_cliente INT,
    id_produto INT,
    quantidade INT,
    FOREIGN KEY (id_cliente) REFERENCES clientes(id_cliente),
    FOREIGN KEY (id_produto) REFERENCES produtos(id_produto)
);
```

Explicação:

- A tabela `clientes` tem `id_cliente` como sua chave primária.
- A tabela `pedidos` tem `id_pedido` como chave primária e `id_cliente` como **chave estrangeira**.
- `FOREIGN KEY (id_cliente) REFERENCES clientes(id_cliente)` cria a relação, garantindo que todo `id_cliente` na tabela `pedidos` exista na tabela `clientes`.

4. **Importe os dados** para ambas as tabelas (use arquivos CSV do Campus Digital).
5. **Apresente os dados criados** usando `SELECT * FROM clientes;` e `SELECT * FROM pedidos;`.

Se você só se lembrar de relacionar os pedidos depois de já ter criado as tabelas e importado os dados, não tem problema. Você não precisa começar do zero.

A solução é adicionar a **chave estrangeira** à sua tabela de `pedidos` usando o comando `ALTER TABLE`. Isso permite que você modifique a estrutura de uma tabela já existente:

```
-- Adicionar a chave estrangeira para a tabela clientes

TABLE pedidos

ADD CONSTRAINT fk_pedidos_clientes

FOREIGN KEY (id_cliente) REFERENCES clientes(id_cliente);
```

Junções (JOINs)

Agora que temos tabelas relacionadas, podemos combiná-las para obter informações mais completas. O `JOIN` nos permite combinar linhas de duas ou mais tabelas com base em uma coluna relacionada. A cláusula `ON` especifica a condição de junção.

Listar o nome dos clientes e a data de seus pedidos:

```
SELECT c.nome, p.data_pedido
FROM clientes c
JOIN pedidos p ON c.id_cliente = p.id_cliente;
```

Explicação:

- `FROM clientes c` e `JOIN pedidos p`: A letra `c` e `p` são apelidos (aliases) para as tabelas, o que facilita a leitura.
- `ON c.id_cliente = p.id_cliente`: Esta é a condição de junção. Ela nos diz para combinar as linhas onde o `id_cliente` da tabela `clientes` é igual ao `id_cliente` da tabela `pedidos`.

Atividade Prática: MySQL com Python (continuação)

1. Importe uma nova base de dados de sua escolha (utilize algum dataset sugerido nos sites da aula anterior);
2. Crie um novo arquivo Python;
3. Conecte-se ao seu novo banco de dados via Python;
4. Crie uma função para consultar a tabela;
5. Execute consultas para responder às seguintes perguntas, imprimindo os resultados:
    - Liste o nome de todos os elementos;.
    - Encontre o nome e algum valor quantitativo ligado a esses elementos;
    - Conte quantos elementos possuem algum filtro de categoria.
6. **Utilize outros arquivos .csv para a temática que você selecionou na aula anterior, relacione as tabelas e construa 3 novas consultas utilizando JOIN.**

Relacionando DataFrames com Python

Agora, vamos unir o que aprendemos sobre `JOIN` com Python. A biblioteca `Pandas` nos permite fazer junções:

```
import pandas as pd
import mysql.connector

def obter_dados_do_banco(query):
    # código da função da aula anterior

query_clientes = "SELECT * FROM clientes"
df_clientes = pd.DataFrame(obter_dados_do_banco(query_clientes), columns=['id_cliente',
'nome', 'email'])
print(df_clientes)
```

Crie outro DataFrame a partir de um arquivo CSV ou Excel com os pedidos.

```
df_pedidos = pd.read_csv('pedidos.csv')
print(df_pedidos)
```

Agora, vamos relacionar os dois DataFrames usando a função `merge`:

```
# Relacionando os DataFrames pela coluna 'id_cliente'
df_relacionado = pd.merge(df_clientes, df_pedidos, on='id_cliente', how='inner')

print(df_relacionado)
```

Explicação:

- `pd.merge(df_clientes, df_pedidos, ...)`: Esta função combina os dois DataFrames.
- `on='id_cliente'`: Especifica a coluna em comum para a junção.
- `how='inner'`: Similar ao `JOIN` em SQL, `inner` retorna apenas as linhas que têm valores correspondentes em ambas as tabelas.

Atividade Prática: MySQL com Python (continuação)

(…)

> **7. Relacione os DataFrames usando o `pd.merge()` e traga a visualização desse comando.**