Pós-Graduação Lato Sensu

Curso de Especialização em Tecnologia Java

Revisão SQL

Professor Hugo Baker Goveia

hbakergoveia@hotmail.com

UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CAMPUS CORNÉLIO PROCÓPIO

1. Conceitos de Banco de Dados

Um banco de dados é uma coleção organizada de informações relacionadas. Ele é projetado para permitir o armazenamento, a recuperação, a modificação e a exclusão eficiente dessas informações.

Imagine um banco de dados como uma "caixa" onde você pode guardar diferentes tipos de informações. Essas informações podem ser sobre pessoas, produtos, transações, registros médicos, entre outros. Cada tipo de informação é armazenada em uma tabela, que é semelhante a uma planilha com linhas e colunas. Cada linha representa um registro individual, enquanto cada coluna seria um atributo específico desse registro.

O banco de dados permite que você defina relacionamentos entre diferentes tabelas. Por exemplo, você pode ter uma tabela de "Clientes" e uma tabela de "Pedidos". A tabela de "Pedidos" pode ter uma coluna que se refere ao ID do cliente, estabelecendo uma relação entre as duas tabelas. Esses relacionamentos ajudam a organizar e estruturar as informações de forma mais eficiente.

Além disso, um banco de dados oferece recursos para consultar, pesquisar e recuperar informações com base em critérios específicos. Você pode executar consultas para recuperar registros que atendam a certas condições ou combinar informações de várias tabelas para obter resultados mais complexos e personalizados.

Existem diferentes tipos de bancos de dados, como bancos de dados **relacionais** (onde as tabelas têm relacionamentos predefinidos) e bancos de dados **não relacionais** (que usam estruturas flexíveis para armazenar dados). Cada tipo tem suas próprias características e é adequado para diferentes tipos de aplicações.

Sendo assim, um banco de dados é um recurso computacional poderoso que possibilita armazenar, organizar e gerenciar informações de maneira eficiente, permitindo a recuperação rápida e confiável desses dados quando necessário. Ele desempenha um papel fundamental em muitas aplicações modernas, desde sistemas de gerenciamento de estoque até redes sociais e sistemas bancários.

“Conjunto de dados integrados que tem por objetivo atender a uma comunidade específica”
Heuser

“Um Banco de Dados é uma coleção estruturada de dados relacionados a alguns fenômenos reais que estamos tentando modelar.”
Ozsü

“Um Banco de Dados é uma coleção de dados relacionados, organizada e armazenada de forma a possibilitar fácil manipulação.”
Elmasri

2. Conceitos de SGBD

SGBD é a sigla para **Sistema Gerenciador de Banco de Dados**. É um software projetado para facilitar o armazenamento, o acesso e a manipulação de **bancos de dados**. O SGBD atua como uma camada intermediária entre os usuários e o banco de dados, fornecendo uma interface para interagir com os dados de forma eficiente e segura.

Os SGBDs desempenham várias funções essenciais para a gestão de um banco de dados como por exemplo:

- Definição de Estrutura: O SGBD permite definir a estrutura do banco de dados, incluindo a criação de tabelas, a especificação dos atributos de cada tabela e a definição de relacionamentos entre as tabelas.

- Armazenamento de Dados: O SGBD é responsável pelo armazenamento físico dos dados em dispositivos de armazenamento, como por exemplo discos rígidos. Ele gerencia como os dados são organizados, acessados e mantidos em memória.

- Manipulação de Dados: O SGBD permite usar uma linguagem de consulta, geralmente chamada de SQL (Structured Query Language), que permite aos usuários realizar operações de consulta, inserção, atualização e exclusão de dados no banco de dados. Essa linguagem facilita a interação com os dados, permitindo que os usuários realizem tarefas complexas com facilidade.

- Controle de Acesso: O SGBD gerencia o acesso aos dados, garantindo que apenas usuários autorizados possam visualizar, modificar ou excluir informações. Ele controla as permissões de acesso e protege os dados contra acessos não autorizados.

- Concorrência e Recuperação: Quando vários usuários acessam o banco de dados simultaneamente, o SGBD gerencia a concorrência para garantir que os dados permaneçam consistentes e que não ocorram conflitos. Além disso, ele oferece recursos de recuperação para garantir que o banco de dados possa se recuperar de falhas e evitar a perda de dados.

- Integridade e Consistência: O SGBD mantém a integridade dos dados, garantindo que as regras e restrições definidas para o banco de dados sejam obedecidas. Ele realiza verificações para garantir que os dados estejam consistentes e que não ocorram valores inválidos ou duplicados.

Existem vários SGBDs disponíveis, como MySQL, Oracle, Microsoft SQL Server, PostgreSQL, MongoDB e o **MariaDB** (adotado nessa disciplina). Cada um tem suas próprias características e é adequado para diferentes tipos de aplicações e necessidades.

Portanto, um SGBD é um software que gerencia todos os aspectos do banco de dados, desde sua criação e organização até a manipulação, acesso e segurança dos dados. Ele

desempenha um papel fundamental na administração eficiente e confiável dos bancos de dados em diversas aplicações no mundo atual.

3. Bancos de dados SQL e NoSQL

BANCO DE DADOS SQL (RELACIONAL):
Os bancos de dados SQL, também conhecidos como bancos de dados relacionais, seguem um modelo de dados tabular, onde os dados são organizados em tabelas com linhas e colunas. Eles são altamente estruturados e usam linguagem SQL (Structured Query Language) para consultar e manipular os dados.

Exemplos: MySQL, PostgreSQL, Oracle Database, Microsoft SQL Server e MariaDB.

Onde é geralmente usado?

- **Aplicações empresariais:** São comuns em sistemas de gestão empresarial (ERP), sistemas de RH, sistemas de contabilidade, etc.
- **Aplicações de comércio eletrônico:** Armazenamento de informações de produtos, pedidos, clientes, etc.
- **Sistemas de gerenciamento de conteúdo (CMS):** Armazenamento de texto, imagens e outros recursos.

Banco de Dados NoSQL (Não Relacional):
Os bancos de dados NoSQL (Not Only SQL) são projetados para dados não estruturados ou semiestruturados e não seguem o modelo tabular dos bancos de dados relacionais. Eles são mais flexíveis e podem acomodar diferentes tipos e modelos de dados como documentos, chave-valor e colunas. Flexibilidade que em termos de estrutura de dados, atua de forma que permite que os esquemas sejam alterados sem muita dificuldade.
NoSQL é mais facilmente escalável horizontalmente, o que o torna adequado para cargas de trabalho que precisam crescer rapidamente.

Consulta aos dados:
Os bancos de dados NoSQL (Not Only SQL) não usam a linguagem SQL (Structured Query Language) tradicional para consultar os dados. Em vez disso, eles usam abordagens e linguagens específicas ao modelo de dados que estão sendo utilizados. Cada tipo de banco de dados NoSQL tem sua própria linguagem ou interface para consultar e manipular dados.

Exemplos de bancos NoSQL e seus Tipos de linguagens:

- **Banco de Dados de Documentos (MongoDB):** Bancos de dados de documentos geralmente usam uma linguagem de consulta específica, como o MongoDB Query Language (MQL). MQL permite consultar documentos JSON-like armazenados no banco de dados. Esses documentos são armazenados em coleções, que são análogas às tabelas em bancos de dados relacionais.

Banco de Dados de Chave-Valor (Redis): Os bancos de dados de chave-valor geralmente fornecem comandos simples para acessar e manipular valores associados a chaves. No caso do Redis, temos comandos como GET, SET, DEL, entre outros.

Banco de Dados de Colunas (Apache Cassandra): Bancos de dados de colunas frequentemente têm uma linguagem de consulta própria, como o CQL (Cassandra Query Language), que é semelhante em alguns aspectos ao SQL, mas otimizado para a estrutura de dados de colunas.

Banco de Dados de Grafos (Neo4j): Bancos de dados de grafos geralmente possuem linguagens de consulta específicas para trabalhar com dados de grafo. Neo4j, por exemplo, usa a linguagem de consulta Cypher.

Onde é geralmente usado?

- **Aplicações web de alto tráfego:** São escaláveis horizontalmente, tornando-os adequados para armazenar dados de redes sociais, aplicativos móveis e outros sistemas de alto tráfego.
- **Análise de big data:** São usados para armazenar e processar grandes volumes de dados em aplicativos de análise.
- **Armazenamento de informações geoespaciais:** Podem ser usados para armazenar e consultar informações geográficas.

4. Escalabilidade do Banco de Dados

Tratando de escalabilidade do banco de dados, podemos encontrar duas abordagens diferentes para lidar com o aumento de carga ou demanda de um sistema. Eles se referem à capacidade de um sistema crescer e lidar com mais carga, essas abordagens recebem o nome de **Ecalabilidade Horizontal** e **Escalabilidade Vertical**.

4.1. Escalabilidade Horizontal

A escalabilidade horizontal envolve adicionar mais recursos (geralmente servidores ou máquinas) ao sistema para aumentar sua capacidade. Isso é feito distribuindo a carga de trabalho entre várias instâncias do sistema, é adequado para sistemas que precisam lidar com um grande volume de tráfego, pois permite que adicionar mais servidores conforme necessário.
No entanto, é importante mencionar que requer gerenciamento de recursos e coordenação entre as instâncias.

Exemplo: Em um sistema de comércio eletrônico, onde podemos ativar servidores web adicionais para distribuir o tráfego de usuários em um caso de aumento de acessos.

4.2. Escalabilidade Vertical

A escalabilidade vertical envolve aumentar os recursos em uma única instância do sistema, geralmente, adicionando mais potência de processamento, memória, armazenamento, etc., à máquina existente.
É adequado quando é preciso que um único servidor ou máquina lide com cargas de trabalho intensivas. No caso de sistemas que têm requisitos de desempenho específicos e previsíveis. Entretanto, há um limite para o quão verticalmente um sistema pode ser escalado, pois os recursos de uma única máquina têm limitações físicas

Exemplo: Em um banco de dados, você pode melhorar o desempenho verticalmente adicionando mais CPU, RAM ou armazenamento à máquina de banco de dados existente.

4.3. Combinando as duas abordagens

Uma abordagem híbrida é ponto interessante também, onde um sistema é escalado horizontalmente para lidar com cargas variáveis e, ao mesmo tempo, escalado verticalmente para melhorar o desempenho em máquinas individuais quando necessário. A escolha entre escalabilidade horizontal e vertical depende das necessidades específicas do projeto que vamos implantar e das características de carga de trabalho, acesso de usuários, etc.

5. Modelos de Dados

SBDs são construídos baseados em Modelos de dados. Os modelos de dados são representações conceituais e estruturais que descrevem como os dados são organizados e relacionados em um banco de dados. Eles fornecem uma estrutura para representar entidades, atributos e relacionamentos entre os dados.

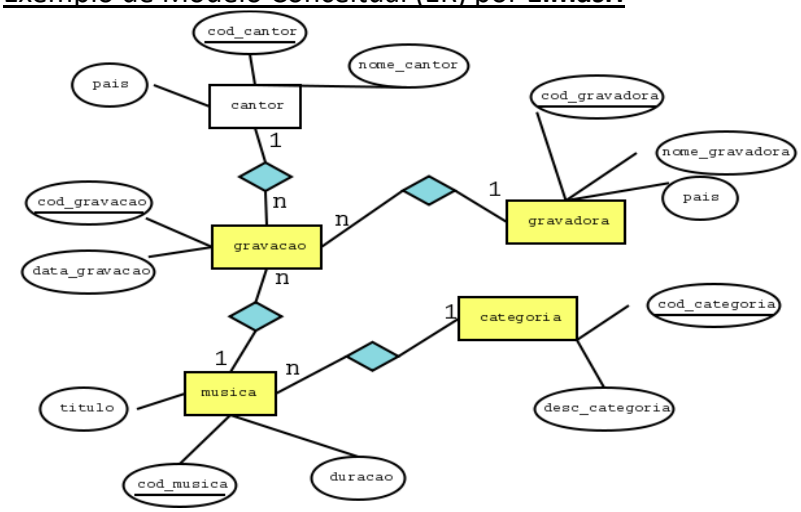
Os três principais tipos de modelos de dados são: modelos conceituais, modelos lógicos e modelos físicos.
Os **modelos conceituais** fornecem uma visão de alto nível dos dados e relacionamentos, os **modelos lógicos** detalham a estrutura lógica do banco de dados e os **modelos físicos** descrevem como os dados são armazenados e otimizados no nível de implementação. Cada um desses modelos desempenha um papel importante durante o processo de projeto e implementação de um sistema de banco de dados.

- **Modelos Conceituais:**

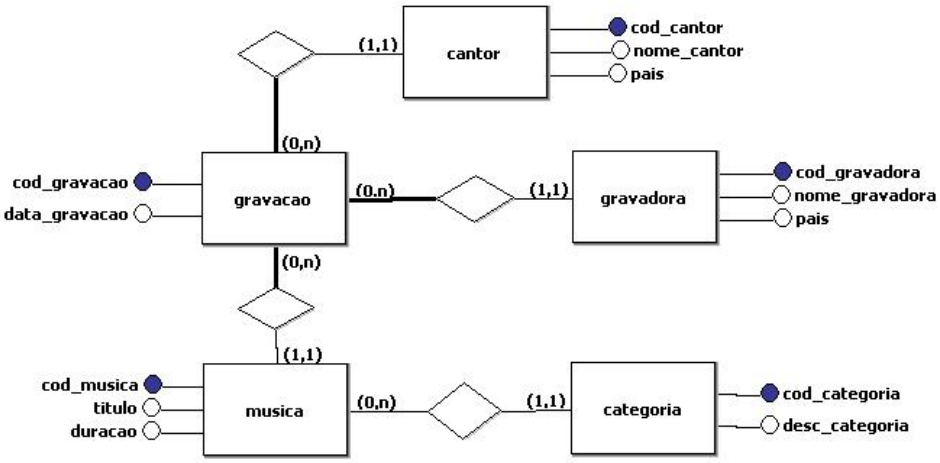
Os modelos conceituais são modelos de dados de alto nível que descrevem as principais entidades, relacionamentos e restrições em um nível abstrato. Eles são independentes de qualquer tecnologia específica de banco de dados e se concentram em capturar a visão conceitual do mundo real.
Um exemplo comum de modelo conceitual é o Diagrama de Entidade-Relacionamento (DER), que descreve as entidades (objetos) envolvidas em um sistema, os atributos que as definem e os relacionamentos entre elas. O DER usa símbolos como retângulos, losangos e linhas para representar entidades, atributos e relacionamentos, respectivamente.

Os modelos conceituais são úteis para entender a estrutura geral dos dados, identificar os principais componentes do sistema e estabelecer uma base para a criação de modelos lógicos.

Exemplo de Modelo Conceitual (ER) por **Elmasri**



Exemplo de Modelo Conceitual (ER) por **Heuser**



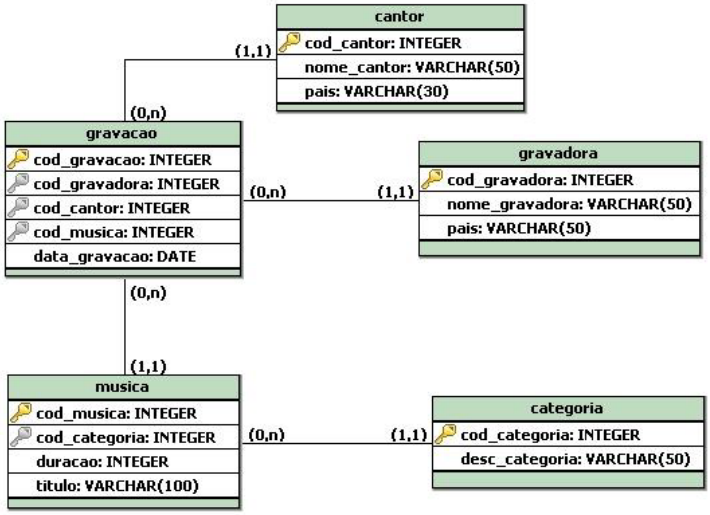
- **Modelos Lógicos:**

Os modelos lógicos são uma representação mais detalhada dos dados e descrevem como os dados são armazenados e manipulados em um nível mais próximo da implementação do banco de dados. Eles são especificados usando uma linguagem de modelagem de dados específica, como o modelo relacional.

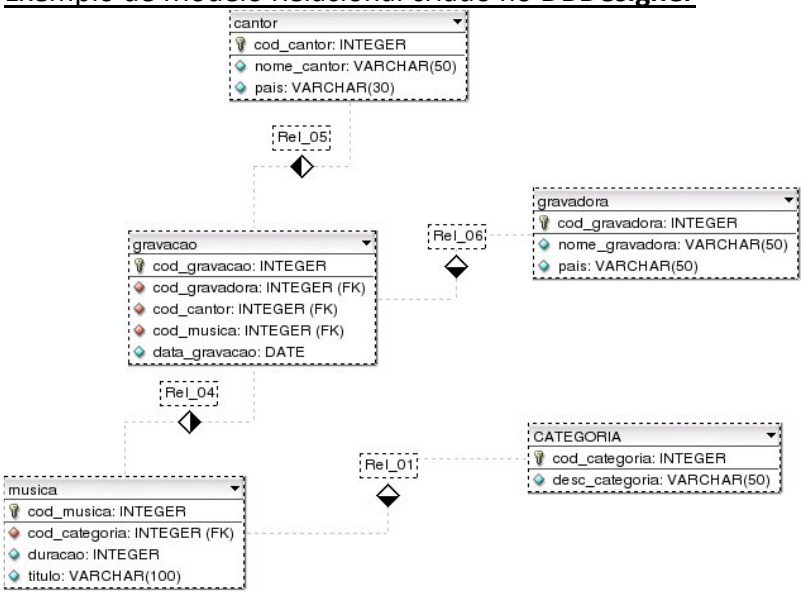
No modelo relacional, por exemplo, um modelo lógico é construído usando tabelas, onde cada tabela representa uma entidade e seus atributos, e os relacionamentos são estabelecidos por meio de chaves primárias e estrangeiras. Nesse nível, também são definidas as restrições de integridade e os tipos de dados a serem usados.

Os modelos lógicos são usados para projetar e implementar o banco de dados de acordo com as necessidades específicas da aplicação. Eles fornecem uma estrutura detalhada para armazenar os dados e são usados como base para a implementação física do banco de dados.

Exemplo de Modelo Relacional criado no **brModelo**



Exemplo de Modelo Relacional criado no **DBDesigner**



- **Modelos Físicos:**

Os modelos físicos descrevem como os dados são realmente armazenados em um nível de baixo nível. Eles são específicos para a plataforma de banco de dados e levam em consideração as características físicas do sistema de armazenamento, como a organização de arquivos, índices, partições e otimizações de desempenho.
No modelo físico, são considerados aspectos como o uso de estruturas de armazenamento eficientes, a distribuição de dados em discos e a indexação adequada

para acesso rápido aos dados. É nesse nível que as decisões sobre o uso de índices, partições e outras técnicas de otimização são tomadas.

Os modelos físicos são usados pelos administradores de banco de dados e pelos desenvolvedores para implementar efetivamente o banco de dados em um ambiente específico.

O modelo físico é a representação do banco de dados usando **SQL** (DDL - Data Definition Language), especificamente com as instruções CREATE TABLE. No modelo físico, você define a estrutura das tabelas, os tipos de dados dos atributos, as chaves primárias, as chaves estrangeiras e outras restrições.

O modelo físico é a etapa em que você traduz o modelo lógico, que descreve as entidades, atributos e relacionamentos, para uma representação concreta em um determinado SGBD. Ao criar as tabelas e definir os relacionamentos e as restrições, você está construindo o esquema físico do banco de dados.

Essa etapa é importante para implementar efetivamente o banco de dados em um ambiente específico, levando em consideração as características físicas da plataforma de banco de dados escolhida. Além das instruções CREATE TABLE, também podem ser utilizadas outras instruções do SQL, como CREATE INDEX, para criar índices que otimizam a recuperação dos dados.

Portanto, o modelo físico é uma representação mais detalhada e concreta do banco de dados, que considera a estrutura lógica definida no modelo lógico e a adapta às características físicas da implementação em um determinado SGBD.

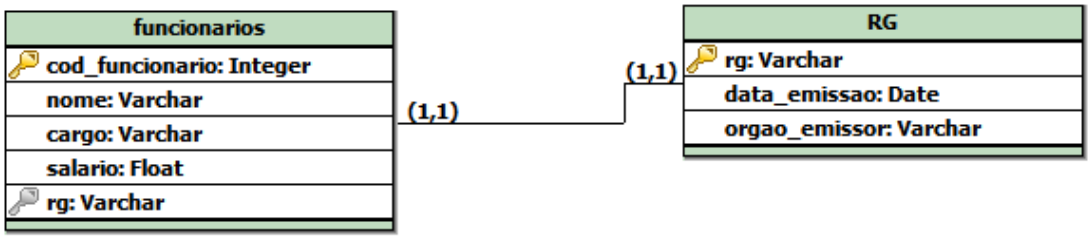
Exemplo de Modelo Físico

```sql
CREATE TABLE cantor (
    cod_cantor integer NOT NULL auto_increment,
    nome_cantor character varying(50),
    pais character varying(30),
    CONSTRAINT cantor_pkey PRIMARY KEY (cod_cantor)
);


CREATE TABLE categoria (
    cod_categoria integer NOT NULL auto_increment,
    desc_categoria character varying(50),
    CONSTRAINT categoria_pkey PRIMARY KEY (cod_categoria)
);


CREATE TABLE gravadora (
    cod_gravadora integer NOT NULL auto_increment,
    nome_gravadora character varying(50),
    pais character varying(50),
    CONSTRAINT gravadora_pkey PRIMARY KEY (cod_gravadora)
);


CREATE TABLE musica (
    cod_musica integer NOT NULL AUTO_INCREMENT,
    cod_categoria integer NOT NULL,
    duracao integer,
    titulo character varying(100),
    CONSTRAINT musica_pkey PRIMARY KEY (cod_musica),
    CONSTRAINT musica_cod_categoria_fkey FOREIGN KEY (cod_categoria) REFERENCES categoria(cod_categoria)
);

CREATE TABLE gravacao (
    cod_gravacao integer NOT NULL auto_increment,
    cod_gravadora integer NOT NULL,
    cod_cantor integer NOT NULL,
    cod_musica integer NOT NULL,
    data_gravacao date,
    CONSTRAINT gravacao_pkey PRIMARY KEY (cod_gravacao),
    CONSTRAINT gravacao_cod_cantor_fkey FOREIGN KEY (cod_cantor) REFERENCES cantor(cod_cantor),
    CONSTRAINT gravacao_cod_gravadora_fkey FOREIGN KEY (cod_gravadora) REFERENCES gravadora(cod_gravadora),
    CONSTRAINT gravacao_cod_musica_fkey FOREIGN KEY (cod_musica) REFERENCES musica(cod_musica)
);
```

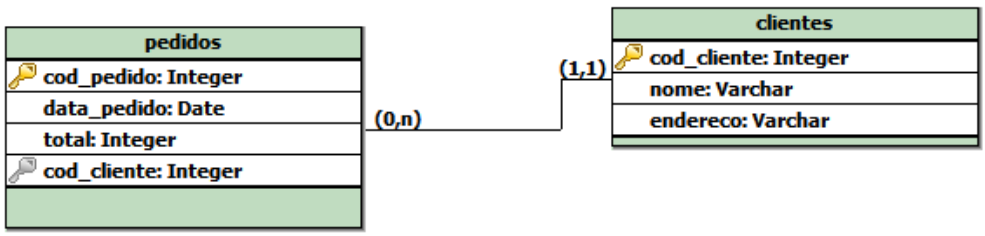
5.1. Relacionamentos entre tabelas

Relacionamentos entre tabelas são uma parte fundamental da modelagem de bancos de dados relacionais. Eles definem como as tabelas em um banco de dados estão relacionadas entre si.

- **One-to-One (Um-para-Um):**
  Neste tipo de relacionamento, uma linha (registro) em uma tabela está relacionada a uma única linha em outra tabela e vice-versa.
  Exemplo: Um banco de dados de funcionários, onde cada funcionário tem uma única carteira de identidade (RG) exclusiva associada a ele. Cada funcionário tem um único RG, e cada RG pertence a apenas um funcionário.
  No print abaixo podemos observar a definição da cardinalidade representada em (1,1) que indica no mínimo 1 e no máximo 1. Reproduzindo a situação de "um funcionário" pode ter apenas "um RG" cadastrado para ele no sistema.

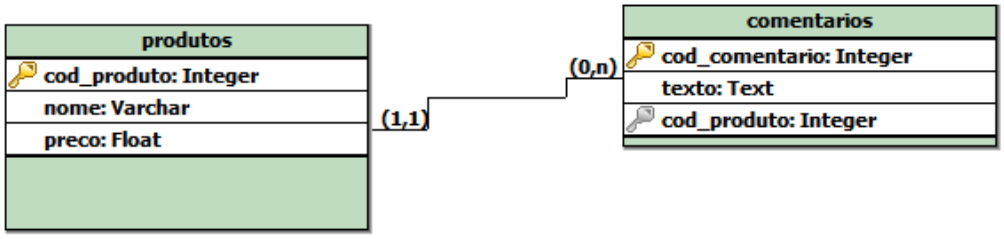


- **One-to-Many (Um-para-Muitos):**
  Neste tipo de relacionamento, uma linha em uma tabela está relacionada a várias linhas em outra tabela, mas as linhas na segunda tabela estão relacionadas a apenas uma linha na primeira tabela.
  Exemplo: Um banco de dados de clientes e pedidos, onde cada cliente pode fazer vários pedidos, mas cada pedido pertence a apenas um cliente.
  No print abaixo podemos observar o lado (0,n), onde indica que pode ter no mínimo 0 e no máximo n(muitos), esse é o lado "muitos" da relação e o lado (1,1) que é o lado "um" da relação. Reproduzindo a situação de que "um cliente" pode ter "muitos pedidos".

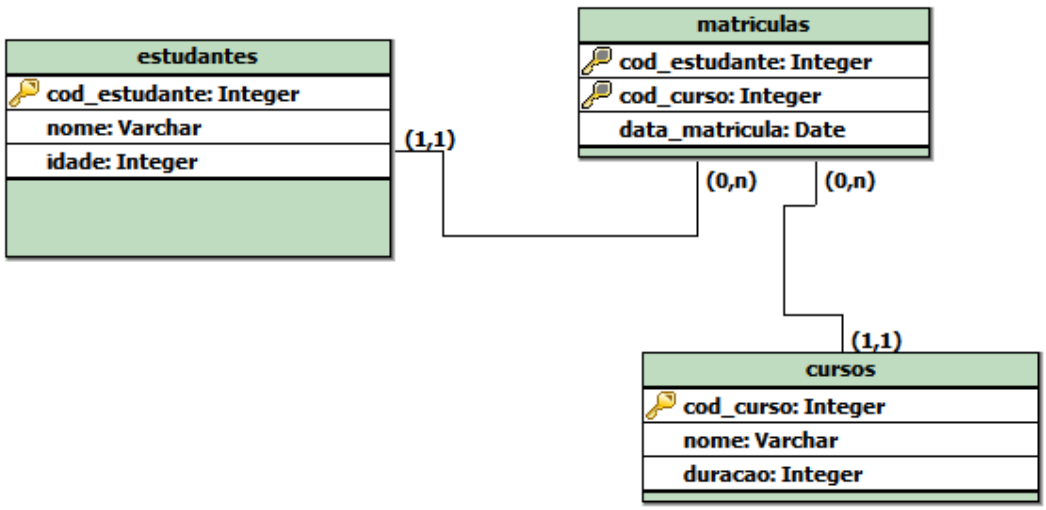


- **Many-to-One (Muitos-para-Um):**
  Este é o inverso do relacionamento "One-to-Many". Muitas linhas em uma tabela estão relacionadas a uma única linha em outra tabela.

Exemplo: Em um sistema de avaliação de produtos, muitos comentários podem estar associados a um único produto, mas um comentário pertence a apenas um produto.
No print abaixo podemos observar o lado (0,n), onde indica que pode ter no mínimo 0 e no máximo n(muitos), esse é o lado "muitos" da relação e o lado (1,1) que é o lado "um" da relação. Reproduzindo a situação de que "um produto" pode ter "muitos comentários".



- **Many-to-Many (Muitos-para-Muitos):**
  Neste tipo de relacionamento, muitas linhas em uma tabela estão relacionadas a muitas linhas em outra tabela. Para implementar isso em um banco de dados relacional, geralmente é necessário uma tabela intermediária (tabela de junção/pivô) que conecta as duas tabelas.
  Exemplo: Um banco de dados de estudantes e cursos, onde muitos estudantes podem estar matriculados em muitos cursos e vice-versa.
  No print abaixo podemos observar a existência da tabela "matrículas" que é onde armazena o relacionamento entre "estudantes" e "cursos", permitindo que dessa forma exista um relacionamento de "Muitos-Para-Muitos". Em "estudantes" o lado (0,n) da linha que liga "estudantes->matriculas" indica que "um estudante pode estar matriculado em nenhum ou vários cursos" e o lado (1,1) dessa mesma linha indica que uma matrícula pertence a um estudante. A mesma ideia se aplica entre "matriculas" e "cursos". Reproduzindo a situação de que "um estudante" pode ter "muitas matrículas" em "muitos cursos" e vice versa.



6. SQL (Structured Query Language)

SQL (Structured Query Language) é uma linguagem de programação especializada para gerenciamento de banco de dados relacionais. Ela foi projetada para permitir a manipulação, definição e controle de bancos de dados relacionais.

O SQL é uma linguagem declarativa, o que significa que você especifica o que deseja que o banco de dados faça e não precisa se preocupar com os detalhes de como isso será realizado internamente. Ele fornece um conjunto de comandos e instruções para executar diversas operações em bancos de dados, como consultas, inserções, atualizações e exclusões de dados.

CATEGORIAS DE COMANDOS SQL:

- **DDL (Data Definition Language):** Esses comandos são usados para definir a estrutura do banco de dados, criando, alterando ou excluindo objetos como tabelas, índices e restrições. Exemplos de comandos DDL são CREATE TABLE, ALTER TABLE e DROP TABLE.

- **DML (Data Manipulation Language):** Esses comandos são usados para manipular os dados dentro das tabelas do banco de dados. Eles permitem a inserção, atualização e exclusão de registros. Exemplos de comandos DML são INSERT, UPDATE e DELETE.

- **DQL (Data Query Language):** Esses comandos são usados para recuperar dados do banco de dados por meio de consultas. A principal instrução DQL é o SELECT, que permite especificar as condições e as colunas desejadas para a consulta.

- **DCL (Data Control Language):** Esses comandos são usados para controlar as permissões de acesso aos objetos do banco de dados. Comandos DCL são usados para conceder ou revogar privilégios a usuários e definir a segurança do banco de dados. Exemplos de comandos DCL são GRANT e REVOKE.

- **TCL (Transaction Control Language):** Esses comandos são usados para gerenciar transações no banco de dados, garantindo a consistência e a integridade dos dados. Comandos TCL são usados para iniciar, confirmar ou reverter transações. Exemplos de comandos TCL são COMMIT e ROLLBACK.

Além dessas categorias, existem outras instruções e cláusulas do SQL que permitem realizar operações avançadas, como junções, agregações, ordenações e subconsultas(subqueries).

6.1. CREATE TABLE e SCHEMA

```sql
CREATE SCHEMA apostilaUTFPR;
USE apostilaUTFPR;

CREATE TABLE especialidades (
  cod_especialidade integer NOT NULL AUTO_INCREMENT,
  nome_especialidade VARCHAR(50),
  PRIMARY KEY (cod_especialidade)
);

CREATE TABLE medicos (
  cod_medico integer NOT NULL auto_increment,
  nome_medico VARCHAR(50),
  crm VARCHAR(10),
  cod_especialidade INT,
  PRIMARY KEY (cod_medico),
  FOREIGN KEY(cod_especialidade) REFERENCES especialidades(cod_especialidade)
);
```

De acordo com o exemplo acima temos:

- **CREATE SCHEMA** é um comando para criar um esquema (ou schema) em um banco de dados. Um esquema é uma estrutura lógica que permite organizar e agrupar objetos relacionados, como tabelas, visões, procedimentos armazenados, funções, entre outros.

  Um esquema fornece um namespace para objetos, ajudando a evitar conflitos de nomes entre os objetos do banco de dados. Além disso, os esquemas podem ser usados para controlar permissões de acesso a objetos, permitindo que determinados usuários ou grupos acessem apenas objetos específicos em um esquema.

  Existe também a instrução CREATE DATABASE, qual a diferença para CREATE SCHEMA?
  No MySQL e em alguns outros sistemas de gerenciamento de banco de dados, como o MariaDB, **não há diferença prática entre as instruções CREATE SCHEMA e CREATE DATABASE.** Ambas as instruções são usadas para criar um novo banco de dados.

  Embora a sintaxe seja ligeiramente diferente, a funcionalidade é a mesma. No MySQL e no MariaDB, a instrução CREATE SCHEMA é uma alternativa para criar um banco de dados, e ambas podem ser usadas indistintamente para criar um novo banco de dados.

  A escolha entre CREATE SCHEMA e CREATE DATABASE geralmente se resume à preferência pessoal ou a requisitos específicos de um determinado sistema de gerenciamento de banco de dados. Em última análise, ambas as instruções têm o mesmo efeito de criar um novo banco de dados no sistema.

Mas em alguns sistemas de gerenciamento de banco de dados (SGBDs), as instruções CREATE SCHEMA e CREATE DATABASE podem ter papéis diferentes.

Por exemplo:
PostgreSQL: No PostgreSQL, a instrução CREATE SCHEMA é usada para criar um novo esquema dentro de um banco de dados existente. Um esquema é uma estrutura lógica que agrupa objetos relacionados, como tabelas, visões e funções. Já a instrução CREATE DATABASE é usada para criar um novo banco de dados separado.

Oracle Database: No Oracle Database, a instrução CREATE SCHEMA é usada para criar um novo usuário e um novo esquema associado a esse usuário. A instrução CREATE DATABASE é usada para criar um novo banco de dados separado.

Microsoft SQL Server: No SQL Server, as instruções CREATE SCHEMA e CREATE DATABASE são usadas de forma similar ao MySQL e MariaDB, ou seja, ambas podem ser usadas para criar um novo banco de dados.

- **USE apostilaUTFPR** é uma instrução utilizada para especificar o banco de dados com o qual você deseja interagir ou no qual deseja executar comandos subsequentes.
  Ao utilizar essa instrução você está informando ao sistema de gerenciamento de banco de dados que deseja estabelecer o contexto de trabalho para o banco de dados chamado "apostilaUTFPR". Isso significa que todos os comandos subsequentes serão executados nesse banco de dados específico, a menos que você altere explicitamente o contexto novamente com uma nova instrução "USE". Essa instrução é útil quando você possui vários bancos de dados no seu sistema e deseja direcionar suas operações para um banco de dados específico. Ela permite que você altere o contexto do banco de dados sem precisar fornecer o nome completo do banco de dados em cada comando subsequente.

- **CREATE TABLE** define que será criado uma tabela com o nome “especialidades” por exemplo e dentro dos parênteses nós temos as colunas de nossa tabela, onde é informado o tipo de dados, como VARCHAR, Integer, entre outros. Também nesse momento é definido a chave primária através do PRIMARY KEY e se há um relacionamento entre as tabelas através do FOREIGN KEY.

6.2. INSERT

Instrução SQL utilizada para inserir um novo registro na tabela. Abaixo temos um exemplo:

```
INSERT INTO especialidades (cod_especialidade, nome_especialidade)
VALUES (1, 'Cardiologia');
```

No exemplo acima estamos inserindo um novo registro na tabela "especialidades". A instrução "INSERT INTO" indica que queremos inserir dados na tabela especificada. Em seguida, especificamos os nomes das colunas entre parênteses: "cod_especialidade" e "nome_especialidade".

A cláusula "VALUES" é usada para especificar os valores que queremos inserir. No exemplo acima, estamos inserindo o valor 1 para a coluna " cod_especialidade " e 'Cardiologia' para a coluna " nome_especialidade ".

6.3. UPDATE

Essa instrução é usada para modificar um registro existente na tabela.

```
UPDATE especialidades
SET nome_especialidade = 'Ortopedia'
WHERE cod_especialidade = 1;
```

A instrução acima começa com o nome da tabela que desejamos atualizar, seguido pela cláusula SET, que especifica os campos que queremos modificar e seus respectivos novos valores.

No exemplo acima, estamos atualizando o valor da coluna "nome_especialidade" para 'Ortopedia' na tabela "especialidades". A cláusula WHERE é usada para filtrar os registros a serem atualizados. Nesse caso, estamos indicando que queremos atualizar o registro onde o valor da coluna "cod_especialidade" é igual a 1.

Ao executar essa instrução, o registro correspondente à especialidade com o ID 1 na tabela "especialidades" será atualizado, substituindo o valor do campo "nome_especialidade" por 'Ortopedia'.

6.4. DELETE

A instrução delete é utilizada para remover registros da tabela. A cláusula FROM especifica a tabela da qual queremos excluir registros.

```
DELETE FROM especialidades
WHERE cod_especialidade = 1;
```

No exemplo acima, estamos usando a cláusula WHERE para filtrar os registros a serem excluídos. Nesse caso, estamos indicando que queremos excluir o registro onde o valor da coluna "cod_especialidade" é igual a 1.

Ao executar essa instrução, o registro correspondente à especialidade com o ID 1 na tabela "especialidades" será removido da tabela.

É importante ter cuidado ao usar a instrução DELETE, pois ela remove permanentemente os registros da tabela.

6.5. SELECT

A instrução SELECT é uma das principais instruções no SQL e é usada para recuperar dados de uma ou mais tabelas em um banco de dados. Ela permite que você especifique quais colunas deseja selecionar, quais tabelas deseja consultar, bem como aplicar filtros e realizar operações de agrupamento, ordenação e junção de dados.

A estrutura básica da instrução SELECT é a seguinte:

```
SELECT coluna1, coluna2, ...
FROM tabela
WHERE condição;
```

- A cláusula SELECT especifica as colunas que você deseja recuperar os dados. Você pode listar as colunas separadas por vírgula ou usar o asterisco (*) para selecionar todas as colunas da tabela, como por exemplo "**Select * from especialidades**"

- A cláusula FROM indica a tabela (ou tabelas) da qual você deseja recuperar os dados. Você pode consultar uma única tabela ou combinar várias tabelas usando junções.

- A cláusula WHERE é opcional e é usada para aplicar condições de filtragem aos registros selecionados. Você pode usar operadores lógicos (como igual, maior que, menor que, etc.) e operadores de comparação (como AND, OR, NOT) para definir as condições.

A instrução SELECT também pode ser estendida com outras cláusulas, como:

- **ORDER BY:** permite ordenar os resultados em ordem ascendente (ASC) ou descendente (DESC) com base em uma ou mais colunas.
- **GROUP BY:** permite agrupar os resultados com base em uma ou mais colunas e realizar operações de agregação, como SUM, COUNT, AVG, etc.
- **JOIN:** permite combinar dados de várias tabelas com base em uma condição de junção.

Um exemplo de **select:**

```
SELECT nome_medico
FROM medicos
WHERE salario > 10000;
```

Na instrução acima estamos selecionando a coluna "nome_medico" da tabela "médicos" e retornando todos os registros onde o salário do médico é maior que R$ 10.000,00, de forma que atenda à condição especificada na cláusula WHERE.

6.6. FUNÇÕES AGREGADAS

São funções que operam em um conjunto de valores e retornam um único valor resumido com base nos dados selecionados. Essas funções são aplicadas em colunas que contêm valores numéricos, como soma, média, contagem, valor mínimo e valor máximo.

As funções agregadas são frequentemente usadas em combinação com a cláusula GROUP BY, onde os resultados são agrupados com base em uma ou mais colunas. Isso permite que as funções agregadas sejam aplicadas a grupos específicos de registros, em vez de operar em todos os dados da tabela.

Sendo muito úteis quando é preciso obter informações resumidas sobre seus dados, como totais, médias, contagens, etc. Elas permitem realizar cálculos e obter insights estatísticos sobre conjuntos de dados específicos em uma tabela.

SUM: retorna a soma de todos os valores de uma coluna numérica informada na instrução.

```
SELECT SUM(salario) AS Total_Salarios
FROM medicos;
```

Acima temos um exemplo onde estamos utilizando a instrução SELECT para recuperar a **soma dos salários** de todos os médicos da tabela "medicos". A função **SUM** é aplicada à coluna "salario", e o resultado é renomeado como "Total_Salarios" usando a cláusula **AS**.

AVG: retorna a média dos valores em uma coluna numérica.

```
SELECT AVG(salario) AS Media_Salarios
FROM medicos;
```

Acima temos um exemplo onde é retornado a média dos salários dos médicos através da função **AVG** que é aplicada à coluna “salario”, e o resultado é renomeado como “Media_Salarios” usando a cláusula **AS**.

COUNT: retorna o número total de registros em uma coluna ou o número de valores não nulos.

```
SELECT COUNT(*) AS TotalMedicos
FROM medicos;
```

Nesse script acima, a instrução SELECT COUNT(*) é usada para contar o número total de registros (médicos) na tabela "medicos". A função COUNT(*) retorna a contagem de todas as linhas da tabela.
A cláusula AS TotalMedicos é usada para renomear o resultado da contagem para "TotalMedicos".

MIN: retorna o valor mínimo em uma coluna.

```
SELECT MIN(salario) AS Menor_Salario
FROM medicos;
```

No exemplo acima, a função agregada **MIN(salario)** é usada para encontrar o menor valor na coluna "salario" da tabela "medicos".
A cláusula AS Menor_Salario é usada para renomear o resultado da função agregada como "Menor_Salario".
Dessa forma conseguimos obter o menor salário pago a algum médico na tabela "medicos".

MAX: retorna o valor máximo em uma coluna.

```
SELECT MAX(salario) AS Maior_Salario
FROM medicos;
```

O exemplo acima utiliza a função MAX para obter o maior salário pago a algum médico.
Nesse script, a função agregada MAX(salario) é usada para encontrar o maior valor na coluna "salario" da tabela "medicos".

A cláusula AS Maior_Salario é usada para renomear o resultado da função agregada como "Maior_Salario".

6.7. CREATE VIEW

A função CREATE VIEW no MariaDB (e em outros sistemas de gerenciamento de bancos de dados relacionais) é usada para criar uma "visão" virtual de uma ou mais tabelas existentes. Uma view é uma consulta SQL armazenada que pode ser tratada como uma **tabela virtual**.
É importante entender que ela não armazena dados fisicamente, mas em vez disso, apresenta uma representação específica dos dados existentes nas tabelas subjacentes do SQL usado para cria-la.

A criação de uma view pode ser útil por vários motivos:

- **Simplificação:** Podemos criar uma view para simplificar consultas complexas ou frequentemente usadas, ocultando detalhes da estrutura da tabela subjacente.

- **Segurança:** As views podem ser usadas para limitar o acesso a determinados dados, mostrando apenas as informações necessárias para os usuários.

- **Abstração de Dados:** As views permitem que os desenvolvedores e usuários finais vejam os dados de uma maneira que faça mais sentido para eles.

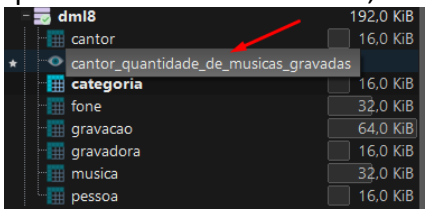
Sintaxe para criação de uma view:

```
CREATE VIEW nome_da_visao AS
SELECT coluna1, coluna2
FROM tabela
WHERE condição;
```

Exemplo de uma criação de view:

```
CREATE VIEW cantor_quantidade_de_musicas_gravadas AS
SELECT c.cod_cantor, c.nome_cantor, COUNT(*) AS musicas_gravadas
FROM gravacao g, cantor c
WHERE c.cod_cantor=g.cod_cantor
GROUP BY c.cod_cantor;
```

Após executar o script, podemos observar que a view aparece como se fosse uma “tabela virtual” junto com as tabelas de nosso banco de dados. No exemplo o banco de dados que temos se chama “dml8”, mas poderia ser qualquer nome.



Se executarmos um SELECT na view que acaba de ser criada, nós podemos observar que ela retorna o resultado para nós. A view age como uma camada de abstração sobre os dados originais.

```
12 SELECT * FROM cantor_quantidade_de_musicas_gravadas;
13
```

cantor (10r × 3c)

cod_cantor	nome_cantor	musicas_gravadas
1	Marisa Monte	4
2	Coldplay	4
3	U2	3
4	Djavan	1
5	Laura Pausini	1
6	Roberto Leal	1
8	Legiao Urbana	4
9	Cazuza	2
10	Tom Jobim	4
11	Frank Sinatra	1