Anotações - Banco de dados (MySQL)

Modelagem

Obrigatoriedade e Cardinalidade de banco de dados

A modelagem de banco de dados envolve a definição de entidades, atributos, relacionamentos e restrições para representar a estrutura e o comportamento do sistema que será suportado pelo banco de dados. Dois conceitos importantes nesse contexto são a obrigatoriedade (ou opção de participação) e a cardinalidade.

Obrigatoriedade (Opção de Participação):

Obrigatório (1)
Indica que a participação de uma entidade no relacionamento é obrigatória. Por exemplo, se uma entidade "Cliente" está relacionada com uma entidade "Pedido" de forma obrigatória, isso significa que cada pedido deve estar associado a um cliente.

Opcional (0 ou 1)
Indica que a participação da entidade no relacionamento é opcional. Pode haver casos em que a participação é permitida, mas não é obrigatória. No exemplo anterior, se a entidade "Cliente" estiver relacionada com "Pedido" de forma opcional, isso significa que um pedido pode ou não estar associado a um cliente.

Cardinalidade

Um para Um (1:1):
Indica que cada instância de uma entidade está associada a uma e apenas uma instância da outra entidade, e vice-versa.

Um para Muitos (1:N)
Indica que cada instância de uma entidade está associada a uma ou várias instâncias da outra entidade, mas cada instância dessa outra entidade está associada a uma e apenas uma instância da primeira entidade.
Muitos para Muitos (M:N) Indica que cada instância de uma entidade pode estar associada a várias instâncias da outra entidade, e vice-versa.
OBSERVAÇÃO:
As obrigatoriedades e cardinalidades corretas dependem das regras de negócio específicas do sistema que está sendo modelado. Ao criar um modelo de banco de dados, é importante entender completamente os requisitos e as relações entre as entidades para tomar decisões precisas sobre a obrigatoriedade e a cardinalidade.

Além disso, a normalização e a integridade referencial são aspectos importantes a serem considerados na modelagem de banco de dados para garantir consistência e eficiência.

Query

Diferença entre Projeção,Seleção e Junção

Projeção

Tudo aquilo que se quer ver na tela (seja de uma tabela ou não).

```
SELECT * FROM endereco;            <= Este vem de uma tabela;

SELECT 2 + 2 AS soma;              <= Já este não vem de uma tabela;

SELECT 2 + 2 AS soma,nome,NOW()    <= concatena-se os dois tipos;
FROM cliente;
```

Seleção

Filtrar subconjuntos (colunas) de um conjunto total (tabela). Onde sua principal cláusula de seleção é o **`WHERE`**.

```
SELECT nome,sexo,email             <= PROJEÇÃO;
FROM cliente
WHERE sexo = 'F';                  <= SELEÇÃO;
```

Junção

Quando se junta informações de tabelas distintas. (Também chamada de Querry Completa).

```
SELECT nome,sexo,bairro,cidade                  <= PROJEÇÃO;
FROM cliente
    INNER JOIN endereco                         <= JUNÇÃO;
    ON `IdCliente`= `Id_Cliente`
WHERE sexo = 'F';                               <=SELEÇÃO;
```

OBSERVAÇÃO:

Na maioria das vezes pode ser necessário a criação de um Query que mostre elementos de mais de duas tabelas, quando isso ocorre é necessária a criação de uma **query dupla,tripla** e assim por diante.

```
SELECT cliente.nome, cliente.sexo, endereco.bairro, endereco.cidade,
telefone.Tipo, telefone.numero
FROM cliente
INNER JOIN endereco
ON cliente.IdCliente = endereco.Id_Cliente
INNER JOIN telefone
ON cliente.IdCliente = telefone.Id_Cliente;
```

OBSERVAÇÃO - 2:
Com objetivo de diminuir as queries, pode-se utilizar pontilhamento para abreviar nomes de tabelas.

```
SELECT c.nome, c.sexo, e.bairro, e.cidade, t.Tipo, t.numero
FROM cliente c
INNER JOIN endereco e
ON c.IdCliente = e.Id_Cliente
INNER JOIN telefone t
ON c.IdCliente = t.Id_Cliente;
```

Tipos de Sintaxe MySQL

Manipulação de dados (**DML**)

A manipulação de dados (DML) é um conjunto de comandos SQL usados para inserir, atualizar, excluir e consultar dados em um banco de dados. Os comandos DML são responsáveis por todas as alterações feitas nos dados, seja criando novos dados, atualizando dados existentes ou excluindo dados.

Alguns dos comandos DML mais comuns são:

INSERT: Insere novos dados em uma tabela.

```
INSERT INTO clientes (nome, email) VALUES ('João Silva',
'joao.silva@example.com');
```

UPDATE: Atualiza dados existentes em uma tabela.

```
UPDATE clientes SET email = 'joao.silva@new.com' WHERE id = 1;
```

DELETE: Exclui dados de uma tabela.

```
DELETE FROM clientes WHERE id = 1;
```

SELECT: Consulta dados de uma tabela

```
SELECT * FROM clientes;.
```

Definição de dados (**DDL**)

A definição de dados (DDL) é um conjunto de comandos SQL usados para criar, alterar e excluir objetos de banco de dados, como tabelas, índices, views e procedimentos armazenados. Os comandos DDL são responsáveis por definir a estrutura do banco de dados.

Alguns dos comandos DDL mais comuns são:

CREATE: Cria um novo objeto de banco de dados.

```
CREATE TABLE clientes (
id int PRIMARY KEY,
nome varchar(255),
email varchar(255)
);
```

ALTER: Altera um objeto de banco de dados existente.

```
ALTER TABLE produto
MODIFY valor VARCHAR(50);
```

DROP: Exclui um objeto de banco de dados existente.

```
ALTER TABLE produto

DROP COLUMN peso;
```

Controle de dados (**DCL**)

O controle de dados (DCL) é um conjunto de comandos SQL usados para conceder e revogar privilégios a usuários e grupos. Os comandos DCL são responsáveis por controlar o acesso a dados e ao banco de dados.

Alguns dos comandos DCL mais comuns são:

GRANT: Concede privilégios a usuários e grupos.

```
GRANT SELECT ON clientes TO joao;
```

REVOKE: Revoga privilégios de usuários e grupos

```
REVOKE SELECT ON clientes FROM joao;.
```

Controle de transação(**TCL**)

O controle de transação (TCL) é um conjunto de comandos SQL usados para gerenciar transações. As transações são um conjunto de operações que devem ser executadas como uma unidade atômica. Os comandos TCL são responsáveis por garantir a integridade dos dados durante as transações.

Alguns dos comandos TCL mais comuns são:

COMMIT: Confirma uma transação.

```
COMMIT;
```

ROLLBACK: Desfaz uma transação.

```
ROLLBACK;
```

SAVEPOINT: Cria um ponto de salvamento em uma transação.

```
SAVEPOINT ponto_de_salvamento;
```

OBSERVAÇÃO: Os comandos DML, DDL, DCL e TCL são essenciais para o gerenciamento de bancos de dados. Cada tipo de comando tem um papel específico a desempenhar na manipulação e segurança dos dados.

SubQuery (INNER QUERY) - Exemplo

/*MOSTRE O CLIENTE QUE POSSUI A MENOR IDADE E SEU NOME*/

1 - QUAL QUERY MOSTRA A MENOR IDADE?

```
SELECT MIN(idade) FROM cliente;
```

2 - ENTÃO É SÓ COLOCAR ESSA QUERY DENTRO DA QUE MOSTRARÁ O NOME COM AJUDA DO **WHERE**

```
SELECT nome,idade FROM cliente
WHERE idade = (SELECT MIN(idade) FROM cliente);
```

Utilizando **JOIN**

```
SELECT c.nome, c.idade

FROM cliente c

JOIN (

    SELECT AVG(idade) AS idade_media

    FROM cliente) AS subquery

 ON c.idade > subquery.idade_media;
```

Criação de tabelas e relações na ordem correta

Uma boa prática é criar as tabelas primeiro, com todas as restrições necessárias, incluindo chaves primárias e chaves estrangeiras. Depois, as relações entre as tabelas podem ser criadas usando a instrução ALTER TABLE.

Ao criar as tabelas primeiro, é possível garantir que todas as restrições necessárias estejam presentes antes de tentar criar as relações. Isso evita erros e inconsistências.

Aqui está um exemplo de como criar as tabelas e depois as relações com ALTER TABLE:

SQL

```
-- Cria a tabela `clientes`

CREATE TABLE clientes (
```

```
  id INT PRIMARY KEY,

  nome VARCHAR(100) NOT NULL,

  email VARCHAR(100) NOT NULL

);



-- Cria a tabela `produtos`

CREATE TABLE produtos (

  id INT PRIMARY KEY,

  nome VARCHAR(100) NOT NULL,

  preco DECIMAL(10,2) NOT NULL

);



-- Cria a relação entre as tabelas

ALTER TABLE clientes ADD CONSTRAINT fk_clientes_produtos
FOREIGN KEY (id) REFERENCES produtos (id);
```

Neste exemplo, as tabelas `clientes` e `produtos` são criadas primeiro. Depois, a relação entre as tabelas é criada usando a instrução ALTER TABLE. A restrição FOREIGN KEY `fk_clientes_produtos` garante que o valor da coluna `id` na tabela `clientes` seja igual ao valor da chave primária da tabela `produtos`.

Outra boa prática é usar a opção `ON DELETE CASCADE` na restrição FOREIGN KEY. Essa opção garante que, se uma linha for excluída da tabela referenciada, todas as linhas relacionadas na tabela de referência também sejam excluídas. Isso ajuda a garantir a integridade referencial.

Aqui está um exemplo de como usar a opção `ON DELETE CASCADE`:

```
SQL
-- Cria a restrição FOREIGN KEY com a opção ON DELETE CASCADE

ALTER TABLE clientes ADD CONSTRAINT fk_clientes_produtos FOREIGN KEY
(id) REFERENCES produtos (id) ON DELETE CASCADE;
```

Neste exemplo, se uma linha for excluída da tabela `produtos`, todas as linhas relacionadas na tabela `clientes` também serão excluídas.

Triggers