Projeto Pessoal: Sistema de Cadastro de Funcionários com MySQL e Modelagem Relacional!

Olá, rede! Tudo bem?

Espero que todos estejam bem! Continuando aqui com a modelagem do nosso banco de dados, Após a modelagem lógica, e a criação do diagrama de entidade-relacionamento (ER), agora estou avançando para a implementação do banco de dados com o modelo físico. Esse modelo é onde realmente detalhamos como os dados serão armazenados no banco, incluindo os tipos de dados, chaves primárias e estrangeiras, índices e a estrutura de tabelas que irão sustentar o sistema.

Fase 4 do Projeto: Implementação do Modelo Físico.

O modelo físico é fundamental para a construção de um banco de dados eficiente e de fácil manutenção. Aqui estão os principais pontos abordados:

Definição de Tipos de Dados: Para cada atributo das entidades, agora escolhemos os tipos de dados apropriados (por exemplo, VARCHAR, DECIMAL, DATE, CHAR, INT E ETC.).

Chaves Primárias e Estrangeiras: A modelagem inclui as chaves primárias para garantir a unicidade de cada registro (tuplas) e as chaves estrangeiras para estabelecer os relacionamentos entre as tabelas.

Respeitando as Formas Normais: Durante a criação do modelo físico, as formas normais foram respeitadas para garantir a eliminação de redundâncias e dependências indesejadas entre os dados.

Criação do banco e das tabelas

Chegamos à 4ª parte do projeto: O Modelo Físico!

Abaixo teremos a criação do banco de dados e as tabelas.

```
create database if not exists bd_empresa
DEFAULT CHARACTER SET utf8mb4
DEFAULT COLLATE utf8mb4_unicode_ci;
-- Criando o banco de dados bd_empresa com charset utf8mb4 para suportar todos os caracteres Unicode e collation utf8mb4_unicode_ci para ordenação e comparação de texto correta.

use bd_empresa; -- acessando o banco de dados criado.

-- criação das tabelas.
-- criação da tabela colaborador.

create table if not exists colaborador(
	idcolaborador int primary key auto_increment,
	nome varchar(45) not null,
	cpf varchar(15) not null, /*não deixei o cpf como único para que possa haver possibilidade de uma recontratação de colaborador.*/
	data_nascimento date not null,
	email varchar(35),
	sexo enum('M', 'F') not null,
```

```
        id_turno int,             -- Declarando a coluna de chave estrangeira
        id_departamento int, -- Declarando a coluna de chave estrangeira
        id_cargo int              -- Declarando a coluna de chave estrangeira
)default charset = utf8mb4;

-- criação da tabela endereco.
create table if not exists endereco(
        idendereco int primary key auto_increment,
        rua varchar(60) not null,
        cep varchar(8) not null,
        bairro varchar(30) not null,
        cidade varchar(30) not null,
        estado char(2) not null,
        numero int unsigned,
        id_colaborador int unique  -- Declarando a coluna de chave estrangeira
)default charset = utf8mb4;

-- criação da tabela telefone.
create table if not exists telefone(
        idtelefone int primary key auto_increment,
        numero varchar(15) not null,
        tipo enum('comerc', 'resid', 'celular') not null,
        id_colaborador int                    -- Declarando a coluna de chave estrangeira
)default charset = utf8mb4;

-- criação da tabela turno.
create table if not exists turno(
        idturno int primary key auto_increment,
        periodo enum('turno A', 'turno B', 'turno C', 'turno Admin') not null,
        hora_inicial enum('06:00', '14:00', '22:00') not null,
```

```
        hora_final enum('14:00', '22:00', '06:00', '17:00') not null
)default charset = utf8mb4;

-- criação da tabela salario.
create table if not exists salario(
        idsalario int primary key auto_increment,
        valor decimal(10,2) not null,
        id_colaborador int    -- Declarando a coluna de chave estrangeira
)default charset = utf8mb4;

-- criação da tabela estadocivil.
create table if not exists estadocivil(
        idestado_civil int primary key auto_increment,
        tipo enum('cadado(a)', 'solteiro(a)', 'divorciado(a)',
                'viuvo(a)',      'separado(a)', 'companheiro(a)') not null,
        id_colaborador int unique
)default charset = utf8mb4;

-- criação da tabela beneficio.
create table if not exists beneficio(
        idbeneficio int primary key auto_increment,
        vale_transporte enum('sim', 'nao') not null,
        vale_refeição enum('sim', 'nao') not null,
        id_colaborador int unique
)default charset = utf8mb4;

-- criação da tabela departamento.
create table if not exists departamento(
        iddepartamento int primary key auto_increment,
        nome_departamento enum('Desenvolvimento Software',
```

```
                    'Desenvolvimento BD', 'Engenharia',
                'Engenharia dados','Gerencia','RH') not null
)default charset = utf8mb4;

-- criação da tabela demissao.
create table if not exists demissao(
        iddemissao int primary key auto_increment,
        data_demissao date not null,
        motivo enum('Demissao sem justa causa',
                'Demissao por justa causa',
                'Pedido de demissao',
                'Demissão consensual',
                'Demissao indireta') not null,
        id_colaborador int unique          -- Declarando a coluna de chave estrangeira
)default charset = utf8mb4;

-- criação da tabela admissao.
create table if not exists admissao(
        idadmissao int primary key auto_increment,
        data_admissao date not null,
        id_colaborador int unique      -- Declarando a coluna de chave estrangeira
)default charset = utf8mb4;

-- criação da tabela cargo.
create table if not exists cargo(
idcargo int primary key auto_increment,
nome_cargo enum('Assitente TI', 'Programador jr', 'Programador pleno',
                'Programador senior', 'Analista banco dados',
                'Gerente', 'Coordenador', 'Assistente RH', 'Gerente RH',
                'Engenheiro banco dados') not null
```

)default charset = utf8mb4;

/***/
/************************Uma breve explicação das decisões tomadas*********************************/
/***/

Boas Práticas

create database if not exists bd_empresa

DEFAULT CHARACTER SET utf8mb4

DEFAULT COLLATE utf8mb4_unicode_ci;

Usando o IF NOT EXISTS ao criar bancos de dados:

Isso evita erros caso o banco de dados já exista. O uso de IF NOT EXISTS é uma boa prática para evitar falhas de execução caso estiver rodando o script várias vezes ou em diferentes ambientes (produção, desenvolvimento, etc.).

Organizar o nome do banco de dados:

É uma boa prática usar um nome de banco de dados claro e relacionado ao projeto, como bd_empresa neste caso, para facilitar a identificação.

Usando utf8mb4 como charset:

Fiz o uso do utf8mb4 sendo uma boa prática porque ele suporta todos os caracteres Unicode, incluindo emojis e outros símbolos que não são cobertos pelo utf8 tradicional.

Usando utf8mb4_unicode_ci como collation:

utf8mb4_unicode_ci é uma colação recomendada, pois é baseada nas regras Unicode para comparação de texto. Ela é case-insensitive (não diferencia maiúsculas de minúsculas) e trata corretamente caracteres acentuados e especiais, proporcionando maior precisão na comparação textual em sistemas que precisam lidar com múltiplos idiomas.

Existem outras colações possíveis, como utf8mb4_general_ci, mas a versão unicode_ci tende a ser mais precisa para comparação de texto em vários idiomas.

/***/

-- a criação das tabelas.

-- tabela colaborador.

Boas Práticas no Código

```
create table if not exists colaborador(
	idcolaborador int primary key auto_increment,
	nome varchar(45) not null,
	cpf varchar(15) not null, /*não deixei o cpf como único para que possa haver possibilidade de uma recontratação de colaborador.*/
	data_nascimento date not null,
	email varchar(35),
	sexo enum('M', 'F') not null,
	id_turno int,		-- Declarando a coluna de chave estrangeira
	id_departamento int, -- Declarando a coluna de chave estrangeira
	id_cargo int		-- Declarando a coluna de chave estrangeira
)default charset = utf8mb4;
```

Uso de IF NOT EXISTS:

O uso de IF NOT EXISTS é uma boa prática, pois evita que a tabela seja recriada caso já exista. Isso ajuda a prevenir erros se o script for executado múltiplas vezes ou em diferentes ambientes (desenvolvimento, produção, etc.).

Definir a chave primária:

Chave primária (idcolaborador): Definir uma chave primária (PRIMARY KEY) é uma boa prática, pois ela garante a unicidade dos registros e melhora a eficiência das operações de busca e referência. O uso de auto_increment é adequado, pois facilita a inserção de novos registros sem a necessidade de fornecer um identificador manualmente.

Escolher tipos de dados adequados:

nome: O tamanho do varchar(45) para nome é uma quantidade razoável para este campo.

email: O tamanho do varchar(35) é razoável para esses campos.

cpf: O tamanho do varchar(15) para esse é o tamanho adequado para armazenar o CPF, incluindo a máscara (por exemplo, XXX.XXX.XXX-XX).

data_nascimento: date para o tipo de dado date é adequado para armazenar datas de nascimento de colaboradores.

sexo: o uso de enum para sexo é uma boa prática, pois restringe os valores possíveis a 'M' e 'F', o que garante dados consistentes.

Adicionar not null em campos obrigatórios:

nome, data_nascimento, sexo: Esses campos são essenciais e por este motivo foram declarados com NOT NULL, o que impede que registros incompletos sejam inseridos na tabela.

email: O email não sofreu a restrição de NOT NULL isso porque não é algo obrigatório.

Chaves estrangeiras: as colunas id_turno, id_departamento, e id_cargo são chaves estrangeiras, conhecidas como foreign keys, e as mesmas serão criadas fora das tabelas, o que é uma abordagem que tem sido discutida como boas práticas de design de banco de dados.

Usar o charset adequado:

O uso de DEFAULT CHARSET = utf8mb4 também é conhecido como boa prática, pois permite a armazenagem de todos os caracteres Unicode (incluindo emojis e símbolos de várias línguas). Isso ajuda a garantir que o banco de dados possa armazenar dados de forma completa e sem problemas relacionados a diferentes idiomas ou caracteres especiais.

```
/*********************************************************************************************************/

-- tabela endereco.

Boas Práticas no Código

create table if not exists endereco(
        idendereco int primary key auto_increment,
        rua varchar(60) not null,
        cep varchar(8) not null,
        bairro varchar(30) not null,
        cidade varchar(30) not null,
        estado char(2) not null,
        numero int unsigned,
        id_colaborador int unique  -- Declarando a coluna de chave estrangeira
)default charset = utf8mb4;
```

Uso de IF NOT EXISTS:

O uso de IF NOT EXISTS é uma boa prática, pois evita a recriação da tabela caso ela já exista, prevenindo erros durante execuções repetidas do script.

Definição de chave primária:

A chave primária (PRIMARY KEY) é corretamente definida em idendereco, garantindo que os registros sejam identificados de forma única e melhorando a performance das operações de busca e manipulação de dados.

É importante observar o uso do not null, ou seja, são campos obrigatórios.

Uso de varchar com tamanhos adequados:

rua: varchar(60) o tamanho de 60 caracteres para a rua é adequado para o nosso caso.

cep: varchar(10) o CEP geralmente tem 8 caracteres (formato XXXXX-XXX), e o tamanho de 10 caracteres está ideal para esse campo.

bairro: varchar(30) está adequado para o nosso caso.

cidade: varchar(30) está adequado para o nosso caso.

estado: char(2) o uso de char(2) para armazenar a sigla do estado (como "SP" ou "RJ") é adequado, pois o Brasil possui apenas 26 estados e o Distrito Federal, então o campo é sempre fixo com 2 caracteres.

número: int unsigned o uso de int unsigned para o número da residência também é considerada uma boa prática, pois o número de uma residência nunca pode ser negativo. Isso ajuda a garantir dados válidos e economiza espaço.

Chave estrangeira (id_colaborador):

A coluna id_colaborador foi definida como única (unique), mas não foi explicitamente declarada como chave estrangeira, isso será tratado mais adiante, além disto, temos um unique o que significa que um colaborador só pode ter um endereço de residência.

Charset utf8mb4:

O uso de DEFAULT CHARSET = utf8mb4 mais uma vez uma boa prática para garantir o suporte a todos os caracteres Unicode, incluindo caracteres especiais, acentos, emojis, etc.

```
/*********************************************************************************************************/

-- criação da tabela telefone.

create table if not exists telefone(

        idtelefone int primary key auto_increment,

        numero varchar(15) not null,

        tipo enum('comerc.', 'resid.', 'celular') not null,
```

```
    id_colaborador int                  -- Declarando a coluna de chave estrangeira
)default charset = utf8mb4;
```

Boas Práticas no Código

Uso de IF NOT EXISTS:

O uso de IF NOT EXISTS é uma boa prática, pois evita erros caso a tabela já exista no banco de dados. Isso é útil durante execuções repetidas do script ou quando o ambiente de desenvolvimento.

Definir chave primária:

A definição de idtelefone como a chave primária (PRIMARY KEY) é essencial para garantir a unicidade de cada telefone e melhorar a performance de buscas e outras operações.

telefone: Uso de VARCHAR para o número de telefone.

numero: VARCHAR(15): com 15 caracteres são suficientes para armazenar números de telefone no Brasil, incluindo o DDD e possíveis símbolos como parênteses e traços por exemplo, (11) 98765-4321 outra observação muito importante o uso do not null ou seja são campos obrigatórios.

telefone: o uso de ENUM para o tipo de telefone:

tipo ENUM('comerc.', 'resid.', 'celular'): O uso de ENUM é adequado para tipos fixos de telefone.

Chave estrangeira para id_colaborador:

A coluna id_colaborador está indicada como uma chave estrangeira, mas não foi declarada explicitamente como tal no código, isso será tratado mais adiante.

Charset utf8mb4:

O uso de DEFAULT CHARSET = utf8mb4 mais uma vez uma boa prática para garantir o suporte a todos os caracteres Unicode, incluindo caracteres especiais, acentos, emojis e caracteres especiais.

```
/*************************************************************************************************************/

-- criação da tabela turno.

create table if not exists turno(

    idturno int primary key auto_increment,

    periodo enum('turno A', 'turno B', 'turno C', 'turno Admin') not null,

    hora_inicial enum('06:00', '14:00', '22:00') not null,

    hora_final enum('14:00', '22:00', '06:00', '17:00') not null
```

)default charset = utf8mb4;

Boas Práticas no Código

Uso de IF NOT EXISTS:

O uso de IF NOT EXISTS é uma boa prática, pois evita erros de recriação da tabela, permitindo que o script seja executado várias vezes sem problemas.

Definir chave primária:

A definição de idturno como a chave primária (PRIMARY KEY) é adequada, pois garante a unicidade dos registros e melhora o desempenho nas consultas que buscam dados específicos.

periodo: o uso de ENUM para periodo

ENUM para periodo: Usar ENUM para os diferentes períodos (turno A, B, C, Admin) isto é uma boa escolha, pois os valores são limitados e conhecidos, tornando os dados consistentes.

hora_inicial e hora_final:

A utilização de ENUM para horários fixos também é adequada no caso em que os horários são bem definidos e limitados, como neste caso.

Aqui vai apenas uma observação:

Para uma flexibilidade e expansão: Se no futuro precisássemos de mais horários ou se os horários precisarem de ajustes mais dinâmicos, talvez um campo TIME ou DATETIME fosse mais adequado. O ENUM pode ser rígido e dificultar alterações no futuro, caso os turnos ou horários se tornem mais variados, mas para o projeto em questão está sobe medida.

Charset utf8mb4:

O uso de DEFAULT CHARSET = utf8mb4 mais uma vez uma boa prática para garantir o suporte a todos os caracteres Unicode, incluindo caracteres especiais, acentos, emojis e caracteres especiais.

```
 /*****************************************************************************************************************/

-- criação da tabela salario.

create table if not exists salario(

        idsalario int primary key auto_increment,

        valor decimal(10,2) not null,

        id_colaborador int    -- Declarando a coluna de chave estrangeira

)default charset = utf8mb4;
```

Boas Práticas no Código

Uso de IF NOT EXISTS:

O uso de IF NOT EXISTS é uma boa prática para evitar erros durante a execução repetida do script, garantindo que a tabela só será criada se ela não existir.

Definição de chave primária:

A coluna idsalario foi definida como chave primária (PRIMARY KEY). Isso assegura que cada registro de salário tenha uma identificação única, melhorando a performance das operações de busca e manipulação de dados.

salário: o uso de DECIMAL para o valor do salário:

valor DECIMAL(10,2) foi uma escolha adequada para representar valores monetários, pois o tipo DECIMAL é exato, o que significa que não há problemas com arredondamento, que podem ocorrer com o tipo FLOAT ou DOUBLE.

Um exemplo para ficar mais claro o entendimento:

10: Define até 10 dígitos no total, o que é mais do que suficiente para valores de salários (por exemplo, até R$ 99.999.999,99).

2: Define 2 casas decimais para representar centavos, que é o formato esperado para valores monetários.

Chave estrangeira (id_colaborador):

A coluna id_colaborador é uma chave estrangeira que se refere à tabela colaborador, mas ela não está explicitamente definida como chave estrangeira no código, isso será tratado mais adiante.

Charset utf8mb4:

O uso de DEFAULT CHARSET = utf8mb4 mais uma vez uma boa prática para garantir o suporte a todos os caracteres Unicode, incluindo caracteres especiais, acentos, emojis e caracteres especiais.

```
/*********************************************************************************************************/

-- criação da tabela estadocivil.

create table if not exists estadocivil(

        idestado_civil int primary key auto_increment,

        tipo enum('cadado(a)', 'solteiro(a)', 'divorciado(a)',
```

```
            'viuvo(a)',      'separado(a)', 'companheiro(a)') not null,
        id_colaborador int unique
)default charset = utf8mb4;
```

Boas Práticas no Código

Uso de IF NOT EXISTS:

O uso de IF NOT EXISTS é uma boa prática para evitar erros de recriação da tabela, permitindo que o script seja executado várias vezes sem problemas.

Definir chave primária:

A coluna idestado_civil é definida como chave primária (PRIMARY KEY), o que é adequado para garantir que cada estado civil tenha uma identificação única.

tipo: o uso de ENUM para o tipo de estado civil:

ENUM para o campo tipo é uma boa prática, pois os valores possíveis são limitados e conhecidos. Isso garante que os valores sejam consistentes e evita erros de digitação ao inserir os dados.

Uso de UNIQUE em id_colaborador:

A coluna id_colaborador está definida como UNIQUE, o que significa que cada colaborador pode ter apenas um estado civil registrado na tabela. Isso foi feito para esta regra de negócios, ou seja, um colaborador só pode ter um estado civil por vez, além disso, é uma chave estrangeira que se refere à tabela colaborador, mas ela não está explicitamente definida como chave estrangeira no código, isso será tratado mais adiante.

Charset utf8mb4:

O uso de DEFAULT CHARSET = utf8mb4 é uma boa prática para garantir que o banco de dados suporte todos os caracteres Unicode, incluindo caracteres especiais, emojis e outros símbolos.

```
/*************************************************************************************************************/
-- criação da tabela beneficio.
create table if not exists beneficio(
        idbeneficio int primary key auto_increment,
        vale_transporte enum('sim', 'nao') not null,
        vale_refeição enum('sim', 'nao') not null,
        id_colaborador int unique
```

```
)default charset = utf8mb4;
```

Boas Práticas no Código

Uso de IF NOT EXISTS:

O uso de IF NOT EXISTS é uma boa prática, pois impede erros de recriação da tabela e garante que o script possa ser executado várias vezes sem problemas.

Definir chave primária:

A coluna idbeneficio é corretamente definida como chave primária (PRIMARY KEY), garantindo a unicidade dos registros e permitindo um acesso eficiente.

Uso de ENUM para os benefícios:

ENUM para vale_transporte e vale_refeicao esta escolha foi feita devido esta regra de negócio, pois esses campos têm apenas dois valores possíveis: 'sim' ou 'não'. Usar ENUM garante que os valores sejam consistentes e evita erros de digitação.

Chave única (UNIQUE) para id_colaborador:

A coluna id_colaborador é definida como UNIQUE, o que significa que um colaborador pode ter nesta regra de negócios apenas um conjunto de benefícios (vale-transporte e vale-refeição) por vez, além disso, é uma chave estrangeira que se refere à tabela colaborador, mas ela não está explicitamente definida como chave estrangeira no código, isso será tratado mais adiante.

Charset utf8mb4:

O uso de DEFAULT CHARSET = utf8mb4 é uma boa prática para garantir que o banco de dados suporte todos os caracteres Unicode, incluindo caracteres especiais, emojis e outros símbolos.

```
/*********************************************************************************************************/

-- criação da tabela departamento.

create table if not exists departamento(

        iddepartamento int primary key auto_increment,

        nome_departamento enum('Desenvolvimento Software',

                                'Desenvolvimento BD', 'Engenharia',

                        'Gerencia','RH') not null

)default charset = utf8mb4;
```

Boas Práticas no Código

Uso de IF NOT EXISTS:

O uso de IF NOT EXISTS é uma boa prática para evitar erros ao recriar a tabela, permitindo que o script seja executado várias vezes sem problemas.

Definir chave primária:

A coluna iddepartamento é corretamente definida como chave primária (PRIMARY KEY), garantindo a unicidade e permitindo um acesso eficiente.

Uso de ENUM para os departamentos:

O tipo ENUM para o campo nome_departamento é uma boa prática, isso porque os departamentos são valores fixos e limitados. O uso de ENUM garante que os valores serão consistentes e evita erros de digitação.

O uso de DEFAULT CHARSET = utf8mb4 é uma boa prática para garantir que o banco de dados suporte todos os caracteres Unicode, incluindo caracteres especiais, emojis e outros símbolos.

```
/*********************************************************************************************************/

-- criação da tabela demissao.

create table if not exists demissao(

	iddemissao int primary key auto_increment,

	data_demissao date not null,

	motivo enum('Demissao sem justa causa',

		'Demissao por justa causa',

		'Pedido de demissao',

		'Demissão consensual',

		'Demissao indireta') not null,

	id_colaborador int unique		-- Declarando a coluna de chave estrangeira

)default charset = utf8mb4;
```

Boas Práticas no Código

Uso de IF NOT EXISTS:

O uso de IF NOT EXISTS é uma boa prática, pois garante que o código não tente recriar a tabela se ela já existir, evitando erros de execução repetida do script.

Definir chave primária:

A coluna iddemissao está definida como chave primária (PRIMARY KEY), o que é uma boa prática para garantir a unicidade e facilitar as operações de busca e atualização.

data_demissao: temos o tipo date é adequado para armazenar datas de demissão dos colaboradores.

Uso de ENUM para o motivo da demissão:

O tipo ENUM para o campo motivo é uma boa prática, pois os valores possíveis são limitados e conhecidos, garantindo que apenas motivos válidos sejam inseridos.

Uso de UNIQUE em id_colaborador:

Chave estrangeira para id_colaborador:

A coluna id_colaborador está definida como UNIQUE, o que significa que um colaborador pode ter apenas um registro de demissão. Isso pode é uma boa prática para esta regra de negócios que é: um colaborador só pode ser demitido uma vez, além disso, é uma chave estrangeira que se refere à tabela colaborador, mas ela não está explicitamente definida como chave estrangeira no código, isso será tratado mais adiante.

Charset utf8mb4:

O uso de DEFAULT CHARSET = utf8mb4 é uma boa prática para garantir que o banco de dados suporte todos os caracteres Unicode, incluindo caracteres especiais, emojis e outros símbolos.

```
/*********************************************************************************************************/

-- criação da tabela admissao.

create table if not exists admissao(

        idadmissao int primary key auto_increment,

        data_admissao date not null,

        id_colaborador int unique      -- Declarando a coluna de chave estrangeira

)default charset = utf8mb4;
```

Boas Práticas no Código

Uso de IF NOT EXISTS:

O uso de IF NOT EXISTS é uma boa prática, pois garante que o código não tente recriar a tabela se ela já existir, evitando erros de execução repetida do script.

Definir chave primária:

A coluna idadmissao está corretamente definida como chave primária (PRIMARY KEY), garantindo a unicidade e facilitando a recuperação de dados relacionados a admissões específicas.

data_admissao: temos o tipo date é adequado para armazenar datas de admissão dos colaboradores.

Uso de UNIQUE em id_colaborador:

Chave estrangeira para id_colaborador:

A coluna id_colaborador está definida como UNIQUE, o que significa que um colaborador pode ter apenas um registro de admissao. Isso pode é uma boa prática para esta regra de negócios que é: um colaborador só pode ser admitido uma vez, além disso, é uma chave estrangeira que se refere à tabela colaborador, mas ela não está explicitamente definida como chave estrangeira no código, isso será tratado mais adiante.

Charset utf8mb4:

O uso de DEFAULT CHARSET = utf8mb4 é uma boa prática, pois isso permite que a tabela armazene todos os caracteres Unicode, incluindo caracteres especiais, emojis e outros símbolos.

```
/*********************************************************************************************************/

-- criação da tabela cargo.

create table if not exists cargo(

idcargo int primary key auto_increment,

nome_cargo enum('Assitente TI', 'Programador jr', 'Programador pleno',

                'Programador senior', 'Analista banco dados',

                'Gerente', 'Coordenador', 'Assistente RH', 'Gerente RH',

                'Engenheiro banco dados') not null

)default charset = utf8mb4;
```

Boas Práticas no Código

Uso de IF NOT EXISTS:

O uso de IF NOT EXISTS é uma boa prática, pois garante que o código não tente recriar a tabela se ela já existir, evitando erros de execução repetida do script.

Definir chave primária:

A coluna idcargo está corretamente definida como chave primária (PRIMARY KEY), garantindo a unicidade de cada cargo e facilitando operações de busca, inserção e manipulação de dados.

Uso de ENUM para nome_cargo:

O uso de ENUM para o campo nome_cargo está adequado isso porque o número de cargos é fixo e limitado. Isso garante que os cargos sejam consistentes e evita erros de digitação ou valores inválidos.

Apenas uma Observação: Caso precisássemos adicionar novos cargos no futuro ou modificar os cargos existentes, o uso de ENUM poderia ser limitante, pois para adicionar ou remover um cargo nós precisaríamos alterar a definição da tabela, o que pode ser mais difícil de gerenciar em um sistema em produção.

O que poderia ser feito: criar uma tabela separada para armazenar os cargos, permitindo adicionar ou remover cargos sem alterar a estrutura da tabela cargo. Isso pode ser mais flexível, especialmente se os cargos mudarem com frequência, porém para esta regra de negócio o modelo atual está atendendo bem nossas necessidades.

Usar utf8mb4 como charset para garantir compatibilidade com caracteres especiais e Unicode.

/***/

-- Definindo todas as chaves estrangeiras para garantir a integridade referencial

-- foreign keys da tabela colaborador, faz referencia a tabela turno

alter table colaborador

add constraint fk_colaborador_turno

foreign key (id_turno) references turno(idturno);

-- foreign keys da tabela colaborador, faz referencia a tabela departamento

alter table colaborador

add constraint fk_colaborador_departamento

foreign key (id_departamento) references departamento(iddepartamento);

-- foreign keys da tabela colaborador, faz referencia a tabela cargo

alter table colaborador

add constraint fk_colaborador_cargo

foreign key (id_cargo) references cargo(idcargo);

-- foreign keys da tabela endereco, faz referencia a tabela colaborador

alter table endereco

```
add constraint fk_endereco_colaborador
foreign key (id_colaborador) references colaborador(idcolaborador);


-- foreign keys da tabela telefone, faz referencia a tabela colaborador
alter table telefone
add constraint fk_telefone_colaborador
foreign key (id_colaborador) references colaborador(idcolaborador);


-- foreign keys da tabela demissao, faz referencia a tabela colaborador
alter table demissao
add constraint fk_demissao_colaborador
foreign key (id_colaborador) references colaborador(idcolaborador);


-- foreign keys da tabela admissao, faz referencia a tabela colaborador
alter table admissao
add constraint fk_admissao_colaborador
foreign key (id_colaborador) references colaborador(idcolaborador);


-- foreign keys da tabela salario, faz referencia a tabela colaborador
alter table salario
add constraint fk_salario_colaborador
foreign key (id_colaborador) references colaborador(idcolaborador);


-- foreign keys da tabela estado_civil, faz referencia a tabela colaborador
alter table estadocivil
```

```
add constraint fk_estadocivil_colaborador

foreign key (id_colaborador) references colaborador(idcolaborador);



-- foreign keys da tabela beneficio, faz referencia a tabela colaborador

alter table beneficio

add constraint fk_beneficio_colaborador

foreign key (id_colaborador) references colaborador(idcolaborador);


/*****************************************************************************************************************/
/************************Uma breve explicação das decisões tomadas*********************************/
/*****************************************************************************************************************/

 -- Para que serve esta tal de foreign key?
```

Uma foreign key (ou chave estrangeira) é um conceito fundamental em bancos de dados relacionais.

Ela é um campo (ou conjunto de campos) em uma tabela que faz referência à primary key (chave primária) de outra tabela. Em outras palavras, a foreign key serve para criar um relacionamento entre tabelas, garantindo que os dados de uma tabela estejam sempre relacionados de forma consistente com os dados de outra.

Abaixo estão as principais finalidades e benefícios de usar uma foreign key:

Manter a integridade referencial:

A foreign key garante que os valores de uma coluna em uma tabela correspondam a valores válidos na tabela referenciada. Por exemplo, evita que seja inserido um pedido com um código de cliente que não existe na tabela de clientes.

Criar relacionamentos entre tabelas:

Por meio das foreign keys, é possível representar relações entre diferentes entidades. Por exemplo, uma tabela de Pedidos pode ter uma foreign key que aponta para a tabela de Clientes, indicando a qual cliente cada pedido pertence.

Evitar dados órfãos:

Ajuda a prevenir registros "órfãos" — ou seja, registros que fazem referência a dados que não existem mais. Por exemplo, ao excluir um cliente, todos os pedidos relacionados podem ser automaticamente excluídos (dependendo das regras definidas), garantindo que a tabela de Pedidos não contenha registros inválidos.

Por que criar as foreign keys separadamente (fora da definição da tabela)?

Em alguns casos, é considerado uma boa prática criar as foreign keys separadamente após a criação das tabelas. Os motivos incluem:

Respeito à ordem de dependências:

Quando existe uma relação de dependência entre várias tabelas (por exemplo, tabela_1, tabela_2, e tabela_3), pode haver situações em que uma foreign key aponta para uma tabela que ainda não foi criada.

Por exemplo:

A tabela_1 depende da tabela_2;

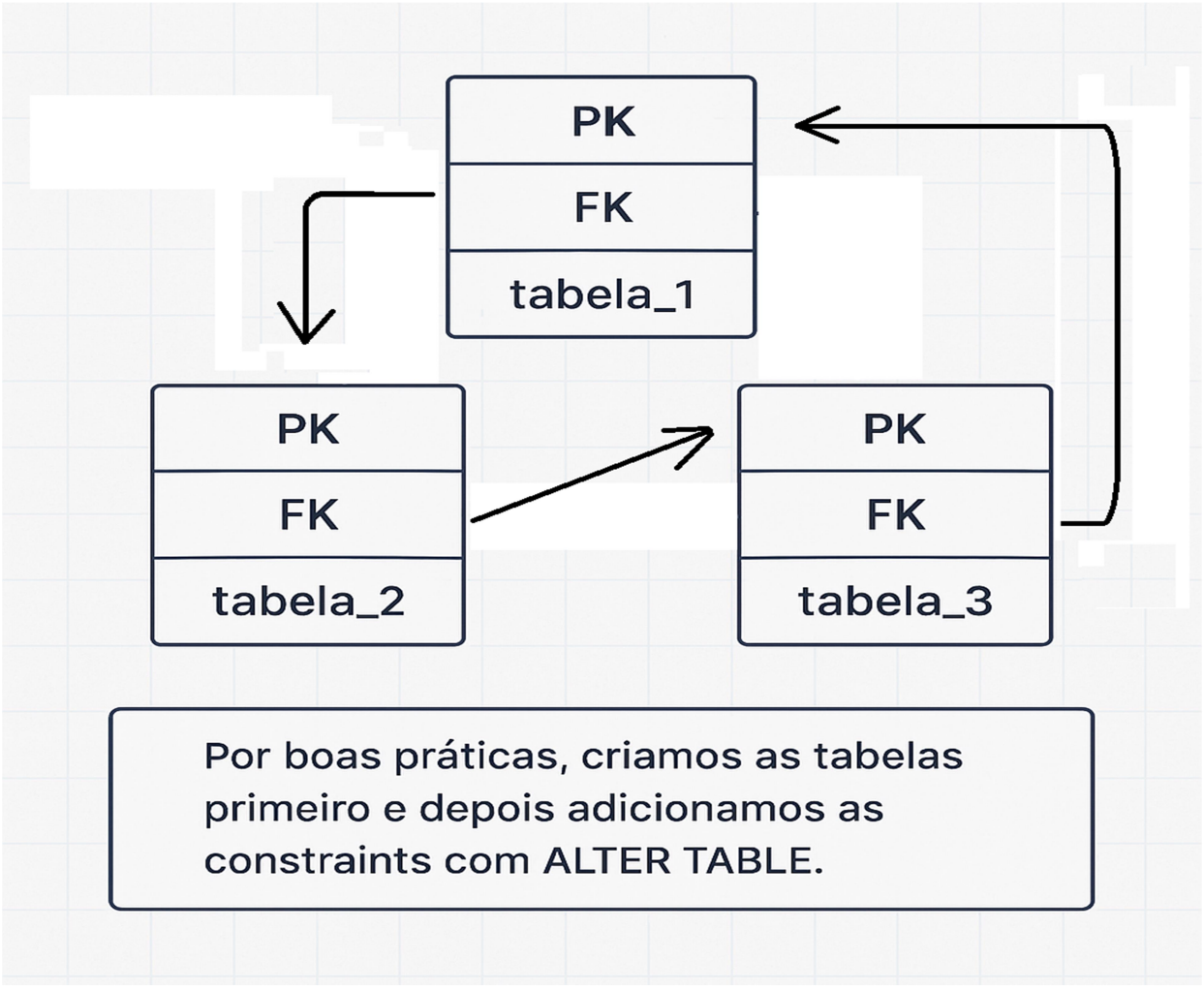
A tabela_2 depende da tabela_3;

E a tabela_3 volta a depender da tabela_1.

Isso cria um ciclo de dependência que impede a criação direta com foreign keys já na definição da tabela, pois o banco de dados exige que a tabela referenciada já exista.

Abaixo uma figura que facilitará o entendimento:

Facilidade de Manutenção e Flexibilidade:



Se você precisar alterar ou remover uma chave estrangeira em um banco de dados sem modificar diretamente a estrutura das tabelas, a definição de foreign keys separadas pode tornar o processo mais fácil e menos propenso a erros.

Essa abordagem é útil em bancos de dados dinâmicos ou grandes sistemas, onde exista a necessidade de ajustar os relacionamentos entre tabelas de forma flexível, sem causar grandes impactos na estrutura dos dados.

Desempenho e Escalabilidade:

Em grandes sistemas distribuídos ou em bancos de dados que exigem alta performance, a definição explícita de foreign keys dentro da tabela pode ser um obstáculo. Já criar as chaves estrangeiras de forma separada pode ajudar a melhorar o desempenho, pois o banco de dados pode não precisar verificar as restrições o tempo todo.

Por que definir as Foreign Keys por fora da tabela?

Há algumas vantagens e considerações em colocar as foreign keys fora da definição da tabela. Isso envolve principalmente flexibilidade no momento de manutenção do banco de dados e gestão das dependências. Aqui estão alguns pontos chave:

Vantagens de definir foreign keys fora da tabela:

Facilidade na modificação da estrutura:

Ao não definir as chaves estrangeiras diretamente nas tabelas durante a criação, fica mais fácil ajustar a estrutura das tabelas posteriormente. Você pode adicionar ou remover chaves estrangeiras com comandos ALTER TABLE, sem a necessidade de modificar a estrutura da tabela diretamente. Isso pode ser útil em ambientes de produção onde a tabela já contém dados, e modificar a estrutura da tabela pode ser mais complexo.

Mais controle sobre a integridade referencial:

Colocar as chaves estrangeiras fora das tabelas pode ser vantajoso quando você precisa de mais controle sobre o relacionamento entre as tabelas, especialmente se houver regras complexas de exclusão, atualização ou manipulação de dados relacionados. Por exemplo, você pode adicionar restrições específicas de ação (como ON DELETE CASCADE, ON UPDATE RESTRICT) que podem ser facilmente ajustadas sem precisar alterar a tabela inteira.

Desvantagens de definir foreign keys fora da tabela, pois é nem tudo são flores existem as desvantagens também abaixo alguma delas:

Potencial aumento da complexidade:

Gerenciar as dependências entre tabelas separadas pode aumentar a complexidade da administração do banco de dados, pois você precisará de scripts separados para definir ou modificar as chaves estrangeiras. Além disso, pode ser mais difícil entender rapidamente todas as relações entre as tabelas, especialmente em bancos de dados com muitas tabelas.

Consistência:

Quando você coloca as chaves estrangeiras fora das tabelas, pode ocorrer a situação de não garantir a consistência referencial durante o desenvolvimento ou manutenção do banco de dados. Isso ocorre especialmente caso esqueça de adicionar a chave estrangeira no futuro. Em comparação, se a chave estrangeira estiver diretamente na tabela, a integridade referencial é mais garantida.

Aumento de complexidade nos scripts de migração:

Quando você precisa migrar dados entre diferentes ambientes (como desenvolvimento, staging, e produção), o processo pode se tornar mais complexo se as foreign keys estiverem declaradas separadamente. Isso pode exigir scripts adicionais para garantir que as dependências sejam configuradas corretamente.

E ai qual o método escolher??? resposta: tudo depende!!!

Bem vejamos foreign key dentro das tabelas:

Isso é ideal quando queremos garantir a integridade referencial automaticamente e tiver um banco de dados com uma estrutura mais simples. Esse método também é mais indicado para projetos menores ou sistemas que não exigem uma manutenção constante da estrutura do banco de dados.

Foreign keys fora da tabela:

Esta abordagem é útil para sistemas mais complexos ou ambientes em que a estrutura do banco de dados pode ser frequentemente alterada. Colocar as foreign keys fora das tabelas proporciona maior flexibilidade, permitindo adicionar ou modificar chaves estrangeiras sem alterar a definição da tabela.

Então basicamente:

Foreign keys dentro da tabela:

Facilitam o controle de integridade referencial diretamente na definição da tabela, mas limitam flexibilidade caso seja necessário fazer mudanças na estrutura do banco.

Foreign keys fora da tabela:

Proporcionam mais flexibilidade e controle ao adicionar ou alterar as chaves estrangeiras sem modificar a estrutura das tabelas, mas podem aumentar a complexidade e o risco de inconsistências.

Aplicação no seu projeto:

No nosso projeto, estamos adotando a abordagem de definir as foreign keys fora das tabelas, utilizando um recurso conhecido como constraint. Essa escolha permite um controle mais flexível e a possibilidade de alterar os relacionamentos entre tabelas conforme necessário, sem interferir diretamente na estrutura das mesmas.

/**/

-- inserção de dados.

/*uma observação muito importante: para adicionar os registros temos que respeitar a ordem de dependência, ou seja, dentro da tabela colaborador consta 3 chaves estrangeiras, que se referem as chaves primárias das tabelas turno, departamento e cargo, sendo assim caso eu tente inserir os dados da tabela colaborador primeiro que os outros registros, os dados simplesmente não serão inseridos.*/

```
insert into turno(idturno, periodo, hora_inicial, hora_final)
values(null, 'turno B', '14:00', '22:00'),
        (null, 'turno A', '06:00', '14:00'),
        (null, 'turno Admin', '06:00', '17:00'),
        (null, 'turno B', '14:00', '22:00'),
        (null, 'turno A', '06:00', '14:00'),
        (null, 'turno B', '14:00', '22:00'),
        (null, 'turno A', '06:00', '14:00'),
        (null, 'turno Admin', '06:00', '17:00'),
        (null, 'turno B', '14:00', '22:00'),
        (null, 'turno A', '06:00', '14:00');
select * from turno;

insert into departamento(iddepartamento, nome_departamento)
values(null, 'Desenvolvimento Software'),
```

```
        (null, 'Desenvolvimento Software'),
        (null, 'Desenvolvimento Software'),
        (null, 'Desenvolvimento Software'),
        (null, 'Desenvolvimento Software'),
        (null, 'Desenvolvimento BD'),
        (null, 'Engenharia'),
        (null, 'Engenharia dados'),
        (null, 'RH'),
        (null, 'Gerencia');
select * from departamento;

insert into cargo(idcargo, nome_cargo)
values(null, 'Programador jr'),
        (null, 'Analista banco dados'),
        (null, 'Assitente TI'),
        (null, 'Programador jr'),
        (null, 'Programador jr'),
        (null, 'Programador senior'),
        (null, 'Analista banco dados'),
        (null, 'Programador jr'),
        (null, 'Assistente RH'),
        (null, 'Gerente');
select * from cargo;

insert into colaborador(idcolaborador, nome, cpf, data_nascimento, email, sexo, id_turno,
id_departamento, id_cargo)
values(null,"Breanna Moore","117-99-4967","1960-06-
07","lynndavenport@example.org","F",1,1,1),
    (null,"Alejandro Zimmerman","509-09-9072","1986-11-11","lpowers@example.net","M",2,2,2),
    (null,"Michelle Page","512-45-5206","1975-04-04","psawyer@example.com","F",3,3,3),
    (null,"Dr. Edward Lee","501-96-5261","1985-09-26","shermanjudith@example.net","M",4,4,4),
```

```
    (null,"Ryan Smith","783-94-8884","1990-05-04","dbates@example.com","M",5,5,5),
    (null,"Martin Castillo","722-95-3843","1983-08-25","robertmiller@example.com","M",6,6,6),
    (null,"Samuel Marquez","695-61-8259","1968-09-22","donna46@example.com","M",7,7,7),
    (null,"Brittany Hernandez","773-24-2166","1950-01-
25","johnanderson@example.com","M",8,8,8),
    (null,"Nicholas Walsh","321-23-1317","1989-10-18","hodgesaaron@example.net","M",9,9,9),
    (null,"Daniel Smith","582-88-2893","1984-02-06","zgrant@example.com","M",10,10,10);
select * from colaborador;


        /*id_turno int,                   -- Declarando a coluna de chave estrangeira
        id_departamento int, -- Declarando a coluna de chave estrangeira
        id_cargo int             -- Declarando a coluna de chave estrangeira
    */
insert into admissao(idadmissao, data_admissao, id_colaborador)
values(null,"1996-09-13",1),
    (null,"1990-08-28",2),
    (null,"2010-11-04",5),
    (null,"2010-03-09",6),
    (null,"1990-10-12",8),
    (null,"2014-09-17",10),
    (null,"1998-01-09",3),
    (null,"1992-12-23",4),
    (null,"2000-02-16",7),
    (null,"1988-12-25",9);
select * from admissao;



insert into demissao(iddemissao, data_demissao, motivo, id_colaborador)
values(null,"2000-06-09","Pedido de demissao",1),
          (null,"2001-02-16","Demissao por justa causa",2),
```

```
        (null,"2019-07-20","Demissao consensual",5),
        (null,"2015-05-06","Demissao por justa causa",6),
        (null,"1998-05-20","Demissao sem justa causa",8),
        (null,"2024-02-05","Demissao consensual",10);
select * from demissao;

insert into endereco(idendereco, rua, cep, bairro, cidade, estado, numero, id_colaborador)
values(null, "Parque Anhangabaú", "01007040", "CENTRO", "São Paulo", "SP", 70,1),
        (null, "Passarela Carlos Drummond de Andrade","01007060", "CENTRO", "São Paulo",
"SP", 71,2),
        (null, "Viaduto Doutor Eusébio Stevaux", "01007040", "CENTRO", "São Paulo", "SP",
73,3),
        (null, "Parque Anhangabaú", "01007040", "CENTRO", "São Paulo", "SP", 70,4),
        (null, "Largo Fernando Gallego", "01007070", "CENTRO", "São Paulo", "SP", 74,5),
        (null, "Rua São Benedito", "02315060", "Vila Progredior", "São Paulo", "SP", 45,6),
        (null, "Avenida Ipiranga", "01037000", "Centro", "São Paulo", "SP", 150,7),
        (null, "Rua dos Três Irmãos", "05698030", "Jardim Rio Bonito", "São Paulo", "SP", 33,8),
        (null, "Rua Martins Fontes", "01050000", "Centro", "São Paulo", "SP", 120,9),
        (null, "Avenida Paulista", "01310100", "Bela Vista", "São Paulo", "SP", 500,10);
select * from endereco;

insert into telefone(idtelefone, numero, tipo, id_colaborador)
values(NULL, '956657712', 'celular', 1),
        (NULL, '64785241', 'resid', 1),
        (NULL, '969412525', 'comerc', 1),
        (NULL, '985451236', 'celular', 2),
        (NULL, '996451236', 'celular', 3),
        (NULL, '996451521', 'celular', 4),
        (NULL, '64581263', 'resid', 5),
        (NULL, '956965231', 'celular', 5),
        (NULL, '996984236', 'celular', 6),
```

```
        (NULL, '945789696', 'celular', 7),
        (NULL, '996965336', 'celular', 9),
        (NULL, '996451236', 'resid', 9),
        (NULL, '978478533', 'celular', 8),
        (NULL, '978964513', 'celular', 10);
select * from telefone;

insert into salario(idsalario, valor, id_colaborador)
values(null, 1750.75, 1),
        (null, 2780.56, 2),
        (null, 1982.75, 3),
        (null, 2256.22, 4),
        (null, 1660.55, 5),
        (null, 3750.75, 6),
        (null, 12230.62, 7),
        (null, 13261.20, 8),
        (null, 2548.63, 9),
        (null, 18500.40, 10);
select * from salario;

insert into estadocivil(idestado_civil, tipo, id_colaborador)
values(null, 'cadado(a)', 1),
        (null, 'solteiro(a)', 2),
        (null, 'divorciado(a)', 3),
        (null, 'cadado(a)', 4),
        (null, 'divorciado(a)', 5),
        (null, 'cadado(a)', 6),
        (null, 'cadado(a)', 7),
        (null, 'cadado(a)', 8),
        (null, 'cadado(a)', 9),
```

```
        (null, 'viuvo(a)', 10);
select * from estadocivil;

insert into beneficio(idbeneficio, vale_transporte, vale_refeição, id_colaborador)
values(null, 'sim', 'sim',1),
        (null, 'nao', 'sim',2),
        (null, 'sim', 'nao',3),
        (null, 'sim', 'nao',4),
        (null, 'sim', 'sim',9),
        (null, 'nao', 'nao',10),
        (null, 'sim', 'sim',7),
        (null, 'sim', 'nao',5),
        (null, 'sim', 'sim',8),
        (null, 'sim', 'sim',6);
select * from beneficio;

-- select de todas as tabelas.
select * from colaborador;
select * from admissao;
select * from demissao;
select * from endereco;
select * from telefone;
select * from turno;
select * from salario;
select * from departamento;
select * from cargo;
select * from estadocivil;
select * from beneficio;
```

Nos próximos posts, será o momento da Execução de Consultas e Testes no Banco de Dados

Além disso, estou populando o banco com mais de 100 registros reais de exemplo — uma ótima forma de praticar consultas SQL na prática!!! Espero que gostem!!! Tudo estará disponível no meu GitHub!: https://github.com/Eder-Alan
Quase esqueci! Se você tem dicas ou já passou por projetos semelhantes em modelagem de dados com MySQL, vou adorar trocar experiências!

#BancoDeDados #SQL #Modelagem #SistemaDeGestao #DesenvolvimentoDeSistemas #Aprendizado #MySQL #DatabaseDesign #ERDiagram #SQLPractice #DevCommunity

/**/