

**Instituto Federal do Amazonas****Professor:** *Sergio Augusto C. Bezerra***Disciplina:** Banco de Dados (Modelagem e Implementação em Banco de Dados)

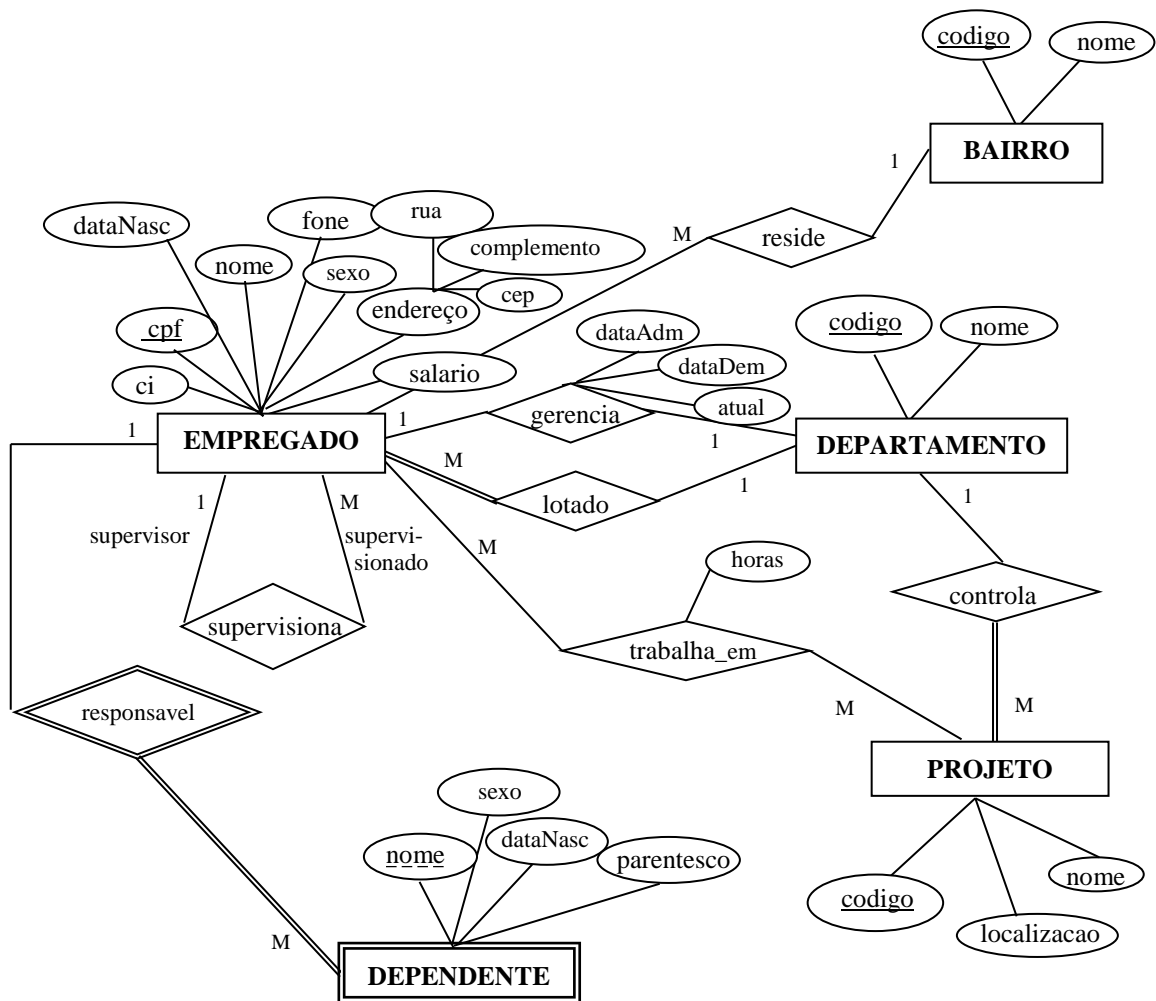
Nesta apostila é abordado um exemplo de uma aplicação de Banco de Dados, chamada EMPRESA, onde os conceitos do Modelo Entidade-Relacionamento (MER), Modelo Relacional e SQL são explorados.

## 1. Descrição do Sistema

O banco de dados EMPRESA controla os empregados, os departamentos e os projetos de uma empresa. Suponha que, após a fase de coleta de dados e análise dos requisitos, os projetistas do banco de dados declararam a seguinte descrição do “minimundo” – a parte da empresa a ser representada no banco de dados:

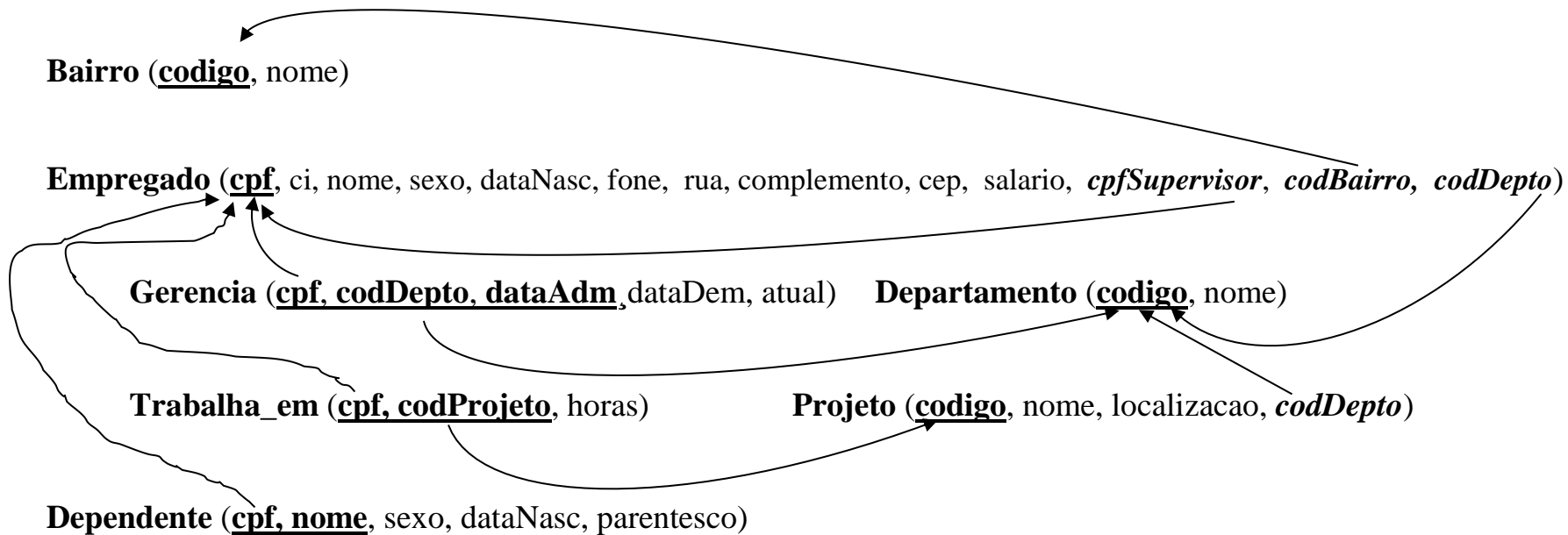
1. A empresa é organizada em departamentos. Cada departamento possui um nome único, um código único e um determinado empregado que gerencia o departamento. Acompanhamos o período que o empregado gerencia o departamento.
2. Um departamento controla um determinado número de projetos, cada um deles possuindo um nome único, um código único e uma única localização.
3. O nome, CPF (Cadastro de Pessoa Física), CI (Carteira de Identidade), endereço (rua, complemento, cep, bairro), salário, sexo, data de nascimento e o telefone de cada empregado deve ser registrado. Um empregado deve ser lotado em um departamento, mas pode trabalhar em diversos projetos, que não são necessariamente controlados pelo mesmo departamento. Acompanhamos o número de horas por semana que um empregado trabalha em cada projeto. Também temos que acompanhar o supervisor direto de cada empregado.
4. Desejamos acompanhar os dependentes de cada empregado para fins de seguridade social. Mantemos para cada dependente o nome, sexo, data de nascimento e grau de parentesco com empregado.

## 2. Modelo Entidade-Relacionamento



### 3. Mapeamento do MER para o Modelo Relacional

Segue abaixo as relações que surgiram após o mapeamento do banco de dados EMPRESA.



#### *Comentário sobre algumas restrições do sistema EMPRESA*

- Todo bairro será identificado por um código, não podendo existir bairros com os mesmos nomes.
- Todo empregado será identificado pelo CPF. O sistema deverá verificar a existência de um determinado empregado antes mesmo de cadastrá-lo no banco de dados. Não é obrigatório que o sistema vincule o empregado a algum departamento ou projeto no momento do cadastro. Um empregado não pode ser supervisor de si mesmo, mas pode ser supervisor e supervisionado ao mesmo tempo.

## 4. Dicionário de Dados

Legenda:      ➡ chave primária  
                  § chave estrangeira

### Bairro

Atributo	Tipo	Tamanho	Nulo	Descrição
codigo ➡	smallint	2	Não	Código do bairro (valor inicial 100)
nome	varchar	50	Não	Não devem existir duplicações

### Empregado

Atributo	Tipo	Tamanho	Nulo	Descrição
cpf ➡	varchar	11	Não	Cadastro de Pessoa Física
ci	varchar	10	Não	Carteira de Identidade
nome	varchar	50	Não	Nome do Empregado
sexo	char	1	Não	M: masculino e F: feminino
dataNasc	varchar	8	Não	Data de nascimento
fone	varchar	8	Sim	Número do telefone
rua	varchar	50	Não	
complemento	varchar	50	Sim	
cep	int	4	Sim	
salario	float	4	Não	Provento bruto
cpfSupervisor §	varchar	11	Sim	Não pode ser o próprio empregado
codBairro §	smallint	2	Não	Código do bairro
codDepto §	smallint	1	Não	Código do departamento. <i>Default</i> [0] para empregado com departamento indefinido.

### Departamento

Atributo	Tipo	Tamanho	Nulo	Descrição
codigo ➡	smallint	1	Não	Código do departamento
nome	varchar	50	Não	Nome do departamento

### Gerência

Atributo	Tipo	Tamanho	Nulo	Descrição
cpf ➡	varchar	11	Não	CPF do empregado que gerencia.
codDepto ➡	smallint	1	Não	Código do departamento gerenciado.
dataAdm ➡	varchar	8	Não	Data de admissão do gerente.
dataDem	varchar	8	Sim	Data de demissão do gerente.
atual	char	1	Não	Indica o atual gerente (Sim[S] Não[N])

### Projeto

Atributo	Tipo	Tamanho	Nulo	Descrição
codigo ➡	smallint	2	Não	Código do projeto. Valor inicial (100)
nome	varchar	200	Não	Nome do projeto.
localizacao	varchar	50	Não	Local onde estar sendo desenvolvido o projeto.
codDepto §	smallint	1	Não	Código do departamento responsável.

**Trabalha\_em**

Atributo	Tipo	Tamanho	Nulo	Descrição
cpf ↔	varchar	11	Não	CPF do empregado alocado no projeto.
codProjeto ↔	smallint	2	Não	Código do projeto.
horas	tinyint	1	Não	Tempo de dedicação ao projeto por semana.

**Dependente**

Atributo	Tipo	Tamanho	Nulo	Descrição
cpf ↔	varchar	11	Não	CPFdo empregado responsável.
nome ↔	varchar	50	Não	Nome do dependente.
sexo	char	1	Não	Sexo do dependente.
dataNasc	char	8	Não	Data de nascimento.
parentesco	char	1	Não	Grau de parentesco (cônjuge[C]   filho[F])

## 5. Trabalhando com SQL

### 5.1 Conceitos de Catálogos e Esquemas em SQL2

Um **esquema em SQL** é identificado através de um *nome de esquema*, e inclui um identificador de autorização para identificar o usuário ou a conta que possui o esquema, bem como **descritores** para cada *elemento* no esquema. Os **elementos** do esquema incluem tabelas, restrições, visões, domínios e outros componentes.

#### Sintaxe:

```
CREATE SCHEMA <nome_esquema> AUTHORIZATION <identificador de autorização>;
```

Exemplo: **CREATE SCHEMA** EMPRESA **AUTHORIZATION** SBEZERRA;

Além do conceito de esquema, a SQL2 utiliza o conceito de **catálogo** – uma chamada coleção de esquemas num ambiente SQL. Um catálogo sempre contém um esquema especial chamado INFORMATION\_SCHEMA (ESQUEMA\_DE\_INFORMAÇÕES) que fornece informações sobre todos os descritores de elementos de todos os esquemas no catálogo para usuários autorizados. Assim como a integridade referencial, as restrições de integridade podem ser definidas entre relações somente se elas existirem em esquemas dentro do mesmo catálogo. Esquemas dentro do mesmo catálogo podem compartilhar certos elementos, tais como definições de domínio.

### 5.2 Criando tabelas. Tipos de dados e Restrições na SQL2.

A SQL utiliza os termos tabela, linha (registro) e coluna para a relação, tupla, e atributo, respectivamente. O comando **CREATE TABLE** é utilizado para especificar uma nova relação dando a ela um nome e especificando seus atributos e restrições. Os atributos são primeiramente especificados, e a cada atributo é dado um nome, um tipo de dado para especificar seu domínio de valores e qualquer restrição de atributo tal como NOT NULL (NÃO NULO) ou NULL(NULO). As restrições de chave, de integridade da entidade e de integridade referencial podem ser especificadas – dentro da instrução CREATE TABLE – depois que os atributos forem declarado ou podem ser adicionadas posteriormente utilizando o comando ALTER TABLE.

**Sintaxe:**

```

CREATE TABLE <nome_tabela>
    (<nome_atributo1> <tipo_atributo1> <NOT NULL>,
     <nome_atributo2> <tipo_atributo2> <NOT NULL>,
     . . .
     <nome_atributoN> <tipo_atributoN> <NOT NULL>,
     <regras de integridade 1>
     . . .
     <regras de integridade M>);

```

Os **tipos de dados** disponíveis para atributos incluem dados numéricos (números inteiros de vários tamanhos como INTEGER ou INT e SMALLINT), string de caracteres (CHAR(n) ou VARCHAR(n), onde n é o número de caracteres), strings de bits, data e hora.

Na SQL2, é possível especificar o tipo de dado de cada atributo diretamente, mas de forma alternativa, um domínio pode ser declarado, e o nome do domínio é utilizado. Isso torna mais fácil alterar o tipo de dado para um domínio que seja utilizado por inúmeros atributos em um esquema, e melhora a capacidade de leitura do esquema. Por exemplo, podemos criar um domínio tipoCpf através da seguinte instrução:

```
CREATE DOMAIN tipoCpf AS char(11);
```

Agora podemos utilizar tipoCpf no lugar de char(11) para outros atributos que precisam de um domínio que inclua valores de um CPF. Um domínio também pode ser uma especificação padrão opcional, através de uma cláusula DEFAULT para atributos, como iremos discutir posteriormente.

**Especificando Restrições e Valores Default na SQL2.** Quando um valor para um determinado atributo for obrigatório especifique NOT NULL, caso contrário pode-se especificar NULL, lembrando que para uma chave primária o NOT NULL será obrigatório. Também é possível definir um *valor default* para um atributo, acrescentando a cláusula **DEFAULT <valor>** a uma definição de atributo. O valor default é incluído em qualquer nova tupla, se um valor explícito não for fornecido para aquele atributo. Caso nenhuma cláusula de default seja especificada, o valor de default (!) para o default é NULL.

Seguindo as especificações do atributo (ou coluna), outras restrições de tabela podem ser especificadas em uma tabela, incluindo as de chave e integridade referencial. A cláusula PRIMARY KEY (CHAVE PRIMÁRIA) especifica um ou mais atributos que compõem a chave primária de uma relação. A cláusula UNIQUE (ÚNICA) especifica chaves alternadas (ou secundárias). A integridade referencial é especificada na cláusula FOREIGN KEY (CHAVE ESTRANGEIRA).

Notemos que uma restrição de integridade pode ser violada quando tuplas são inseridas ou excluídas, ou quando um atributo de chave estrangeira é modificado. Na SQL2, o projetista do esquema pode especificar a ação a ser tomada se uma restrição de integridade referencial for violada. As opções incluem SET NULL, CASCADE e SET DEFAULT. Uma opção deve ser qualificada com ON DELETE (NA EXCLUSÃO) ou ON UPDATE (NA ATUALIZAÇÃO).

Mas para um melhor entendimento sobre estas ações, didaticamente prefiro que o aluno se detenha primeiro, em aprender, como: construir um banco de dados; construir e alterar tabelas; especificar restrições de domínio, restrições de tabela (*primary key*, *unique* e *foreign key*), onde o aluno mesmo implementará as ações a serem tomadas quando uma restrição de integridade referencial for violada; fazer consultas envolvendo inclusão, exclusão e alteração de registros. De posse desses conhecimentos o aluno estará apto a abstrair com categoria os detalhes de restrições.

**Exemplos:**

- 1) Criar as tabelas **Bairro**, **Departamento**, **Empregado** e **Gerencia** conforme o dicionário de dados. Há, não esqueça antes de:
  - a. Criar uma consulta nova e salvá-la com o nome de *CriarTabelasBDEmpresa.sql*.
  - b. Descrever um cabeçalho, identificando o objetivo da consulta, qual o autor e, em que data esta consulta foi criada ou alterada. Que isto seja uma prática em todas as consultas que você irá construir.

--CONSULTA PARA CRIAR AS TABELAS DO BANCO DE DADOS EMPRESA

--Autor: Sérgio Augusto C. Bezerra

--Data: 11/02/2017

```
CREATE TABLE Bairro(
    codigo          smallint      not null,
    nome            varchar(50)    not null,
    primary key     (codigo));
```

```
CREATE TABLE Departamento(
    codigo          smallint      not null,
    nome            varchar(50)    not null,
    primary key     (codigo),
    unique          (nome));
```

```
CREATE TABLE Empregado(
    cpf             varchar (11)    not null,
    ci              varchar (10)    not null,
    nome            varchar(50)     not null,
    sexo            char             not null,
    dataNasc        varchar (8)     not null,
    fone            varchar (8)     null,
    salario         float            not null,
    rua             varchar(50)      not null,
    complemento     varchar(50)      null,
    cep             int              null,
    cpfSupervisor   varchar (11)    null,
    codBairro       smallint         not null,
    codDepto        smallint         not null,
    primary key     (cpf),
    foreign key     (cpfSupervisor)  References Empregado(cpf),
    foreign key     (codBairro)       References Bairro(codigo),
    foreign key     (codDepto)        References Departamento(codigo));
```

```
CREATE TABLE Gerencia(
    cpf          varchar (11)  not null,
    codDepto     smallint      not null,
    dataAdm      varchar (8)   not null,
    dataDem      varchar (8)   null,
    atual        char          not null,
    primary key  (cpf, codDepto, dataAdm),
    foreign key  (cpf)          References Empregado(cpf),
    foreign key  (codDepto)     References Departamento(codigo));
```

- 2) Vamos escrever a tabela **Trabalha\_em** e tentar criá-la. Escreva alguma observação caso você note algo de anormal.

```
CREATE TABLE Trabalha_em(
    cpf          varchar (11)  not null,
    codProjeto   smallint      not null,
    horas        tinyint       not null,
    primary key  (cpf, codProjeto),
    foreign key  (cpf)          References Empregado(cpf),
    foreign key  (codProjeto)   References Projeto(codigo));
```

**Obs:** \_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

- 3) Criar as tabelas **Projeto** e **Dependente**.

### 5.3 Excluindo tabelas

#### Sintaxe:

**DROP TABLE** <nome da tabela>;

#### Exemplos:

- 4) Vamos excluir a tabela Dependente.

**drop table** Dependente;

- 5) Vamos excluir a tabela Projeto.

**drop table** Projeto;

**Obs:** \_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

### 5.4 Adicionando e Excluindo Colunas

Adicionando ou Excluindo colunas são duas formas de modificar uma tabela. Veja sintaxe abaixo:

#### Sintaxe:

```
ALTER TABLE    <nome da tabela>
{ | [ALTER COLUMN <nome da coluna>]
  | { ADD
      { <nome e tipo da coluna>
        { [NULL | NOT NULL]
      }
    | { DROP COLUMN <nome da coluna>;
```



- 6) Vamos inserir um atributo chamado **celular** na tabela Empregado.

**alter table** Empregado **add** celular varchar (8);

- 7) Vamos excluir um atributo chamado **celular** na tabela Empregado.

**alter table** Empregado **drop column** celular ;

## 5.5 Inserindo Registros em tabelas

### Sintaxe:

```
insert into <nome da tabela> <lista de atributos>
values      <lista de valores>;
```

### Exemplos:

- 8) Vamos escrever e executar as inserções abaixo para a tabela **Bairro e Departamento**. Salve as consultas como **BairroInserirGeral.sql** e **DepartamentoInserirGeral.sql**.

**insert into** Bairro  
**values** (100, 'ADRIANÓPOLIS');

Codigo	Nome
101	ARMANDO MENDES
102	ALEIXO
103	SÃO JOSÉ
104	JORGE TEIXEIRA
105	NOVO ALEIXO

**insert into** Departamento  
**values** (100, 'CONTABILIDADE');

Codigo	Nome
101	ADMINISTRAÇÃO
102	ENGENHARIA CIVIL
103	ENGENHARIA ELÉTRICA



9) Vamos escrever e executar as inserções abaixo para a tabela **Empregado**. Salve as consultas como ***EmpregadoInserirGeral.sql***.

**insert into** Empregado

**values** ('1111111111', '1010101010', 'ALISON MARTINS', 'M', '20031980', '2331111', 1800.00, 'RUA A Q.R N.19.',  
'ENFRETE A VIDEO LOCADORA AMANHECHER', 69011111, NULL, 100, 101);

Um valor NULO para o  
cpf do supervisor.

cpf	ci	nome	sexo	nasc.	telefone	salário	rua	comp.	cep	cpf-sup	bairro	depto
222222222-22	202020202-0	ANDRÉ LUIZ	M	25051978	2442222	2050.35	AV. BORBA, 555.		69022-222		105	103
333333333-33	303030303-0	TELMA CORREA	F	12121980	3453333	1680.50	AV. AÇAI, 333.	AP. 10	69033-333	222222222-22	103	103
444444444-44	404040404-0	ANNE OLIVEIRA	F	28031982	2394040	1600.00	RUA T, Q-M, 40.		69000-444	111111111-11	103	101

Como o segundo registro a ser inserido na tabela Empregado não possui valores para os atributos ***complemento*** e ***cpf do supervisor***, poderíamos proceder da seguinte forma:

**insert into** Empregado (cpf, ci, nome, sexo, dataNasc, fone, salario, rua, cep, codBairro, codDepto)

**values** ('2222222222', '2020202020', 'ANDRÉ LUIZ', 'M', '25051978', '2442222', 2050.35, 'AV. BORBA, 555.',  
69022222, 105, 103);

ou

**insert into** Empregado

**values** ( '2222222222', '2020202020', 'ANDRÉ LUIZ', 'M', '25051978', '2442222', 2050.35, 'AV. BORBA, 555.', '',  
69022222, null, 105, 103);

Um valor em branco para  
complemento.

## 5.6 Atualizando Registros

Para realizar alterações em um ou mais registros de uma tabela, é utilizado o comando **update** de acordo com a sintaxe abaixo:

### Sintaxe:

```
update <nome da tabela>
set    <atributo1> = <expressão1> , <atributo2> = <expressão2>
where  <condição>;
```

### Exemplo:

10) Considere a seguinte declaração:

“Atualize o salário de ANNE OLIVEIRA (cpf: 444444444-44) para R\$ 1680,50”.

```
update Empregado
set    salario = 1680.50
where  cpf = '444444444444';
```

## 5.7 Eliminando Registros

Para se eliminar um ou mais registros em uma tabela, utiliza-se o comando **delete**, onde a:

### Sintaxe:

```
delete from <nome da tabela>
where <condição>;
```

### Exemplo:

11) Considere a seguinte declaração:

“Excluir os empregados que trabalham no departamento 103 e possuem salário acima de R\$ 2000.00.”

```
delete from Empregado
where codDepto = 103 and
      salario > 2000.00;
```

*Nota:*

---



---



---



---

## 5.8 Consultas Básicas

A SQL possui uma instrução básica para recuperar informações de um banco de dados: a instrução **SELECT**. A forma básica desta instrução, às vezes chamada de mapeamento ou de bloco de **select-from-where**, é formada pelas três causas **SELECT**, **FROM** e **WHERE** e possui a:

**Sintaxe:**

```

select <lista de atributos>
from <lista de tabelas>
where <condições>;

```

**Exemplos:**

12) Recupere o nome e a data de nascimento de todos os empregados.

**Q1:**

```

select nome, dataNasc
from Empregado;

```

**Lidando com Nomes de Atributos Ambíguos e Renomeando (Aplicando Alias [Apelidos])**

Na SQL, o mesmo nome pode ser utilizado para dois (ou mais) atributos, contanto que os atributos estejam em *diferentes relações*. Se este for o caso, e uma consulta se referir a dois ou mais atributos com o mesmo nome, nós devemos qualificar o nome do atributo com o nome da relação, para evitar ambigüidade. Isto é realizado dando um prefixo do nome da relação ao nome do atributo e separando os dois através de um ponto. Veja a Consulta 2 (Q2) para ilustrar.

13) Recuperar o nome e o endereço (rua, complemento, cep, bairro) de todos os empregados.

**Q2:**

```

select Empregado.nome, rua, complemento, cep, Bairro.nome
from Empregado, Bairro
where codBairro = código;

```

Ambigüidade também surge no caso de consultas que se referem duas vezes à mesma relação, como no exemplo a seguir.

14) Liste o nome dos supervisores e de seus respectivos supervisionados.

**Q3:**

```

select S.nome, E.nome
from Empregado as S, Empregado as E
where S.cpf = E.cpfSupervisor;

```

Neste caso, nos é permitido declarar nomes de relações alternativos *S* e *E*, chamados de *alias* (apelidos) ou variáveis de tuplas, para a relação EMPREGADO. Um apelido pode vir em seguida à palavra-chave *AS*, conforme mostrado acima em Q3, ou pode diretamente seguir o nome da relação – por exemplo, escrevendo em *from Empregado S, Empregado E*. Também é possível renomear os atributos da relação dentro da consulta em SQL2. Veja a abaixo Q3 modificada.

```

select S.nome as Supervisor, E.nome as Supervisionado
from Empregado as S, Empregado as E
where S.cpf = E.cpfSupervisor;

```

## Exercícios

1) Vamos inserir mais registros para as tabelas, observe sempre o nome da consulta onde os dados foram salvos.

- Adicionar mais registros em **Empregado** (*EmpregadoInserirGeral.sql*).

cpf	ci	nome	sexo	nasc.	fone	salário
55555555-55	505050505-0	CRISTINA SOFIA	F	20/04/1979	2499-5050	2400,00
66666666-66	606060606-0	WILLIAM DE SOUZA	M	12/08/1978	2488-6060	2500,00
77777777-77	707070707-0	ELISANGELA DA SILVA	F	17/10/1977	2489-7070	1700,00

Continuação dos dados para Empregado:

cpf	rua	complemento	cep	cpf do supervisor	bairro	departamento
55555555-55	RUA MANA-CAPURU	AP. 202	69050-555		JORGE TEIXEIRA	ENGENHARIA CIVIL
66666666-66	RUA IPIRANGA		69060-666		ALEIXO	CONTABILIDADE
77777777-77	RUA AUTAZES	ATRÁS DO SHOPPING G. CIRCULAR	69079-777	66666666-66	JORGE TEIXEIRA	CONTABILIDADE

- Atualize o BD Empresa para que tenhamos a seguinte configuração para gerentes dos departamentos (*GerenteInserirGeral.sql*).

Gerente	Departamento	Data de Admissão
ALISON MARTINS	ADMINISTRAÇÃO	25/04/2001
CRISTINA SOFIA	ENGENHARIA CIVIL	18/07/2002
ANDRÉ LUIZ	ENGENHARIA ELÉTRICA	23/02/1998
WILLIAM DE SOUZA	CONTABILIDADE	03/06/1999

- Atualize o BD Empresa para incluir os **projetos**, bem como, os departamentos responsáveis e os trabalhadores envolvidos (*ProjetoInserirGeral.sql*).

Código	Projeto	Localização	SETOR	EMPREGADO	HORAS
100	SISTEMATIZAR A CONTABILIDADE NA EMPRESA	MANAUS	CONTABILIDADE	WILLIAM DE SOUZA ELISANGELA DA SILVA	4 4
101	PONTE 1 – KM3 DO CAREIRO	CAREIRO	ENG. CIVIL	CRISTINA SOFIA	4
102	MANUTENÇÃO ELÉTRICA DAS TURBINAS DA HIDRELÉTRICA DE BALBINA	PRESIDENTE FIGUEIREDO	ENG. ELÉTRICA	ANDRÉ LUIZ TELMA CORREA CRISTINA SOFIA	8 8 4

- Adicionar ao BD Empresa os dependentes dos empregados especificados abaixo (*DependenteInserirGeral.sql*).

Cpf do Empregado	Dependente	Sexo	Data de Nascimento	Parentesco
22222222-22	ANDRÉ LUIZ JUNIOR	M	12/11/2001	FILHO
	IVETE SILVA	F	22/12/1980	CÔNJUGE
33333333-33	LUIZA CORREA	F	13/05/2002	FILHO
77777777-77	JOSÉ MARIA	M	02/01/1970	CÔNJUGE

2) Construa consultas para os seguintes problemas:

- Faça uma relação dos bairros.
- Faça uma relação dos departamentos existentes na empresa.
- Recupere o nome e o telefone dos empregados que trabalham no departamento de ENGENHARIA ELÉTRICA.
- O nome dos empregados que possuem salário maior ou igual a R\$ 1.800,00.
- Relação do código e nome de todos os projetos, bem como o nome do departamento que controla estes projetos.
- O nome e o telefone do gerente de cada departamento.
- O nome, sexo, data de nascimento e o grau de parentesco dos dependentes dos empregados.
- O nome dos empregados que estão envolvidos em algum projeto.
- O nome e quantidade de horas dos empregados envolvidos no projeto “SISTEMATIZAÇÃO DA CONTABILIDADE NA EMPRESA”.
- Recuperar o nome, o endereço (rua, complemento, cep, bairro) e telefone de todos os empregados envolvidos em algum projeto controlado pelo departamento de ENGENHARIA ELÉTRICA.

### A Cláusula WHERE Não Especificada e o Uso de Asterisco (\*)

A falta de uma cláusula WHERE indica que não há nenhuma condição na seleção de tuplas (registros em tabelas); portanto, todas as tuplas da relação (tabelas em SQL) especificadas na cláusula FROM estão qualificadas e estão selecionadas para o resultado da consulta. Se mais de uma relação estiver especificada na cláusula FROM e não existir nenhuma cláusula WHERE, então o PRODUTO CRUZADO ou PRODUTO CARTESIANO – todas as possíveis combinações de tuplas – dessas relações é selecionado.

15) Selecione o cpf e o nome de todos os empregados (Q4) e todas as combinações possíveis entre EMPREGADO e DEPARTAMENTO (cpf e nome dos empregados, e nome dos departamentos) (Q5)

Q4:

```
select cpf, nome
from Empregado;
```

Q5:

```
select cpf, E.nome, D.nome
from Empregado E, Departamento D;
```

O que você percebeu no resultado da Consulta 5??????????????????

Para recuperar todos os atributos de uma tabela, não precisamos relacionar explicitamente os nomes dos atributos na cláusula SELECT, basta especificar um *asterisco* (\*), que corresponde a *todos os atributos*. Veja os exemplos abaixo:

16) Liste todos os atributos da tabela Dependente (Q6).

17) Liste todos os atributos em Empregado e Departamento, mas apenas do departamento de CONTABILIDADE (Q7).

Q6:

```
select *
from Dependente;
```

Q7:

```
select *
from Empregado E, Departamento D
where E.codDepto = D.codigo and D.nome = 'CONTABILIDADE';
```

### Tabelas como Conjuntos na SQL

A SQL geralmente trata uma tabela não como um conjunto, mas como um multiconjunto; tuplas repetidas podem aparecer mais de uma vez numa tabela e no resultado de uma consulta. A SQL não elimina automaticamente tuplas repetidas no resultado das consultas, pelas seguintes razões:

- A eliminação de tuplas repetidas é uma operação onerosa. Um modo de implementá-la é primeiramente ordenar as tuplas e então eliminar as repetidas.
- O usuário pode desejar ver as tuplas repetidas no resultado de uma consulta.
- Quando uma função de agregação é aplicada às tuplas, na maioria dos casos não desejamos eliminar as repetidas.

- Se efetivamente desejamos eliminar tuplas duplicatas utilizamos a palavra-chave DISTINCT na cláusula SELECT.

18) Recupere o salário de todos os empregados (Q8) e todos os valores diferentes de salários (Q9).

Q8:

```
select salario
from Empregado;
```

Q9:

```
select distinct salario
from Empregado;
```

A SQL incorporou diretamente algumas das operações de conjuntos da álgebra relacional. Existe uma operação de união de conjuntos (**UNION**), e na SQL2, existem também operações de diferenças entre conjuntos (**NOT IN**) e interseção de conjuntos (**IN**). As relações resultantes dessas operações de conjuntos são conjuntos de tuplas; ou seja, tuplas duplicadas são eliminadas no resultado. Uma vez que essas operações de conjunto se aplicam somente a relações compatíveis para união, devemos nos assegurar de que duas relações nas quais aplicamos a operação possuam os mesmos atributos e que estes apareçam na mesma ordem em ambas as relações.



19) Recuperar o nome de todos os empregados que trabalham em algum projeto ou que tenham algum dependente (**Q10**).

**Q10:**

```
(select e.nome
from empregado e, trabalha_em t
where e.cpf=t.cpf )
union
(select e.nome
from empregado e, dependente d
where e.cpf=d.cpf);
```

20) Recuperar o nome dos empregados que não participam de nenhum projeto.

**Q11:**

```
select nome
from empregado
where cpf NOT IN
      (select e.cpf
       from empregado e, trabalha_em t
       where e.cpf=t.cpf);
```

21) Recuperar o nome dos gerentes que participam de algum projeto.

**Q12:**

```
select e.nome
from empregado e, gerencia g
where e.cpf=g.cpf and e.nome IN
      (select nome
       from empregado e, trabalha_em t
       where e.cpf=t.cpf)
order by nome;
```

### Comparações de Substring, Operadores Aritméticos e Ordenamento

- O operador LIKE permite condições de comparação somente em partes de uma string de caracteres. Linhas parciais são especificadas, utilizando dois caracteres reservados: “%” substitui um número arbitrário de caracteres e o *underscore* ( ) substitui um único caracter.

22) Recupere o nome de todos os empregados que possuem o sobrenome SILVA.

**Q13:**

```
select nome
from empregado
where nome like '%Silva%';
```

23) Encontre todos os empregados que nasceram durante a década de 1980.

**Q14:**

```
select nome
from empregado
where dataNasc like '____198_';
```

- Os operadores padrão da aritmética para soma (+), subtração (-), multiplicação (\*) e divisão (/) podem ser aplicados a valores numéricos ou atributos com domínio numéricos.

24) Mostre os salários resultantes de todos os empregados se fosse dado um aumento de 20% (a saída deverá conter: nome, salário normal e salário com aumento do empregado)

**Q15:**

```
select nome, salario, 1.2*salario as 'Novo Salário'
from empregado;
```

- Outro operador de comparação é o BETWEEN (ENTRE).

25) Recupere o nome e o salário de todos os empregados que ganham um salário entre 1600 e 2000 reais. A condição (salario BETWEEN 1600 and 2000) é equivalente a (salario >= 1600 and salario <= 2000).

**Q16:**

```
select nome, salario
from empregado
where salario between 1600 and 2000;
```

- A SQL permite aos usuários ordenar as *tuplas* no resultado de uma consulta pelos valores de um ou mais atributos, utilizando a cláusula ORDER BY (ORDENAR POR), veja a consulta Q15. A ordem padrão é a ordem ascendente de valores. Podemos especificar a palavra-chave **DESC** se desejarmos uma ordem descendente de valores. A palavra-chave **ASC** pode ser utilizada para especificar explicitamente a ordem ascendente.

## 5.9 Funções Agregadas

São funções que realizam uma operação matemática em uma coluna. O MySQL implementa algumas funções agregadas: *count()*, *count(distinct)*, *min()*, *max()*, *avg()*, *sum()* e *std()*.

### • Função *count*

É utilizada para contar o número de ocorrências de todos os valores não-nulos em uma coluna.

26) Calcular ou contar o número de empregados da empresa.

**Q17:**

```
select count(*)
from empregado;
```

### • Função *count(distinct)*

É utilizada para contar as ocorrências não-nulas únicas de uma coluna ou colunas.

27) Insira um nome de bairro duplicado ao final da tabela bairro. Depois faça uma consulta para contar o número de bairros não duplicados.

Q18:

```
-- Inserindo um bairro com o mesmo nome, mas com código diferente.
-- Isto será possível porque a tabela bairro não tem o campo nome
-- definido como unique, unique(nome), como tem em departamento.
insert into bairro values(106,'Novo Aleixo');
```

```
select count(distinct nome)
from bairro;
```

- Função *max*

Retorna o valor mais alto da coluna.

28) Qual o maior salário da empresa?

Q19:

```
select max(salario)
from empregado;
```

- Função *min*

Retorna o valor mais baixo da coluna.

29) Qual o menor salário da empresa?

Q20:

```
select min(salario)
from empregado;
```

- Função *avg*

Retorna o valor médio da coluna designada.

30) Qual o valor média do salário de um trabalhador da empresa?

Q21:

```
select avg(salario)
from empregado;
```

- Função *sum*

Retorna a soma dos valores de uma coluna designada.

31) Qual o gasto total com salários dos empregados na empresa?

Q22:

```
select sum(salario)
from empregado;
```

32) Qual a quantidade de horas que a empregada Cristina Sofia reserva em projetos na empresa?

Q23:

```
select sum(horas)
from empregado e, trabalha_em t
where e.cpf = t.cpf and e.nome = 'Cristina Sofia';
```

- Função *std*

Retorna o desvio padrão de uma coluna designada.

33) Qual o desvio padrão dos salários dos empregados da empresa?

**Q24:**

```
select std(salario)
from empregado;
```

## 5.10 Cláusula Group By

A cláusula *group by* permite agrupar um conjunto de resultados finito.

34) Faça uma lista da quantidade de empregados da empresa por bairro

**Q25:**

```
select b.nome, count(e.nome)
from bairro b, empregado e
where b.codigo = e.codBairro
group by b.nome;
```

35) Faça uma listagem da quantidade de horas dos empregados da empresa que trabalham em algum projeto.

**Q26:**

```
select e.nome, sum(horas)
from empregado e, trabalha_em t
where e.cpf = t.cpf
group by e.nome ;
```

36) Refazendo a consulta anterior para que possa ser destacado em ordem decrescente de quantidade de horas e renomeando as colunas, respectivamente, para “Empregado” e “Somatório de horas em projeto”, tem-se:

**Q27:**

```
select e.nome 'Empregado', sum(horas) 'Somatório de horas em projeto'
from empregado e, trabalha_em t
where e.cpf = t.cpf
group by e.nome
order by sum(horas) desc;
```

## 5.11 Cláusula Having

37) Faça uma listagem contendo o nome do empregado por departamento que possuem maior salário

**Q28:**

```
select d.nome , e.nome, e.salario
from empregado e, departamento d
where e.coddepto = d.codigo
group by d.nome
having max(e.salario);
```

38) Faça uma listagem contendo o nome do empregado por departamento que possuem salário maior que R\$ 2000.

**Q29:**

```
select d.nome , e.nome, e.salario
from empregado e, departamento d
where e.coddepto = d.codigo
group by d.nome
having e.salario > 2000;
```

## 5.12 Stored Procedure

### Stored Procedure

A sintaxe a seguir refere-se à criação de um *stored procedure*.

#### Sintaxe:

```
DELIMITER $$
CREATE PROCEDURE sp_name (modo parâmetro tipo, ...)
BEGIN
    -- Corpo do Procedimento
END $$
DELIMITER;
```

- *sp\_name*: nome do stored procedure, que devem ser observadas as mesmas regras para definição de variáveis, onde não se pode iniciar com número ou caracteres especiais, salvo o underline, “\_”.
- *modo*: sinaliza a forma como os parâmetros serão tratados pelo procedimento:
  - *IN*: que o parâmetro é somente para trabalhar a entrada de dados, sendo uma porta de entrada para o SGBD.
  - *OUT*: que o parâmetro é apenas de saída, sendo uma porta de saída do SGBD.
  - *INOUT*: que o parâmetro é tanto de entrada quanto de saída, sendo uma porta ao mesmo tempo de entrada e saída do SGBD.
- *tipo*: é o tipo de dado do parâmetro fornecido pelo SGBD como *char*, *int*, dentre outros.

O comando DELIMITER é um ponto importante a ser considerado. O MySQL por padrão utiliza o sinal de “;” como delimitador de comandos, separando as instruções a serem executadas. Entretanto, no corpo de um *stored procedure* será necessário separar algumas instruções internas utilizando tal sinal, como consequência disso uma alteração no delimitador padrão deve ser realizada, neste caso para “\$\$. Vale ressaltar, no entanto, que ao final do *stored procedure* deve-se restaurar o valor padrão inicial do MySQL.

Depois de criado o *stored procedure*, então para executá-lo basta usar a palavra reservada CALL, a saber: *CALL sp\_name(parâmetro1, parâmetro2, ...);*

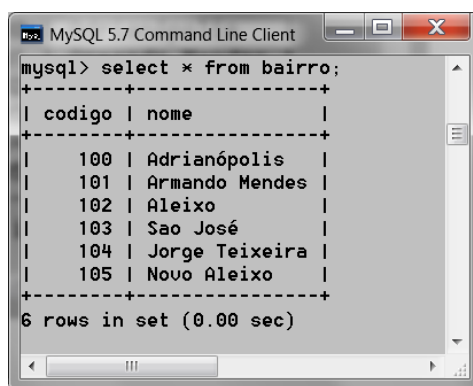
A seguir serão apresentados alguns exemplos de como criar e usar *stored procedures* no banco de dados *empresa*.

39) Crie um *stored procedure* para inserir o código e nome de um determinado bairro.

**Q30:**

```
DELIMITER $$
USE `empresa2017`$$
DROP PROCEDURE IF EXISTS sp_BairroInserir$$
CREATE PROCEDURE sp_BairroInserir (IN codigo smallint, IN nome varchar(50))
BEGIN
    insert into Bairro values(codigo, nome);
END $$
DELIMITER;
```

A tabela a seguir mostra uma fotografia do estado atual da tabela Bairro antes de utilizarmos Q30.



```
mysql> select * from bairro;
```

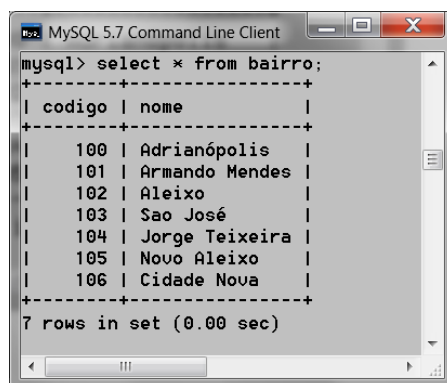
codigo	nome
100	Adrianópolis
101	Armando Mendes
102	Aleixo
103	Sao José
104	Jorge Teixeira
105	Novo Aleixo

6 rows in set (0.00 sec)

Para executar o `sp_BairroInserir` no próprio MySQL basta chamá-lo em um terminal ou a partir de uma consulta SQL, como a seguir:

```
mysql> CALL sp_BairroInserir(106, 'Cidade Nova');
```

Observe que antes o banco de dados empresa tinha 6 bairros, mas a partir da chamada de `sp_BairroInserir` um novo registro foi adicionado, passando a tabela Bairro a conter 7 registros.



```
mysql> select * from bairro;
```

codigo	nome
100	Adrianópolis
101	Armando Mendes
102	Aleixo
103	Sao José
104	Jorge Teixeira
105	Novo Aleixo
106	Cidade Nova

7 rows in set (0.00 sec)

40) Crie um *stored procedure* para retornar a quantidade de bairros existentes.

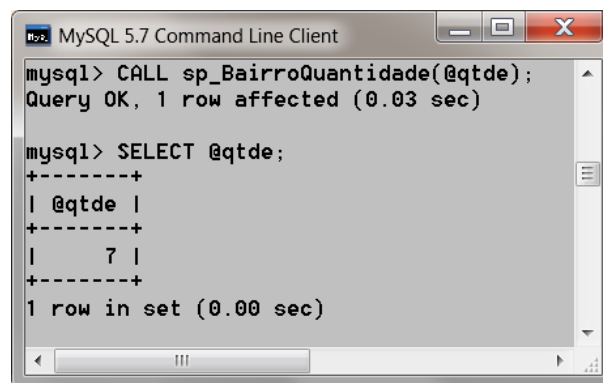
**Q31:**

```

DELIMITER $$
USE `empresa2017` $$
DROP PROCEDURE IF EXISTS sp_BairroQuantidade $$
CREATE PROCEDURE sp_BairroQuantidade (OUT quantidade smallint)
BEGIN
    SELECT COUNT(*) INTO quantidade FROM Bairro;
END $$
DELIMITER ;

```

O procedimento `sp_BairroQuantidade` tem como objetivo retornar a quantidade de registros da tabela `Bairro` em empresa, que para isso, por meio da palavra reservada `INTO`, atribui a variável de saída `quantidade` o valor 7. Claro, isso tudo dentro de `sp_BairroQuantidade`. Já para chamá-lo, como na figura a seguir, foi necessário adicionar uma variável antecedida por um `@` (`@qtde`) em virtude deste procedimento possuir um parâmetro definido com o modo `OUT`, indicando que o SGBD retornará um resultado por meio desta variável. `@qtde` é um ponteiro que referencia o parâmetro `quantidade` definido dentro de `sp_BairroQuantidade`.



41) Crie um *stored procedure* para retornar o cpf do supervisor de um determinado empregado.

**Q32:**

```

DELIMITER $$
USE `empresa2017` $$
DROP PROCEDURE IF EXISTS sp_SuperEmpregado $$
CREATE PROCEDURE sp_SuperEmpregado (INOUT cpfSuper varchar(11))
BEGIN
    SELECT cpfSupervisor INTO cpfSuper FROM Empregado where cpf = cpfSuper;
END $$
DELIMITER ;

```

Para usar `sp_SuperEmpregado` deve-se proceder como a seguir:

```

mysql> set @cpf = '33333333333';
mysql> call sp_SuperEmpregado(@cpf);
mysql> select @cpf;

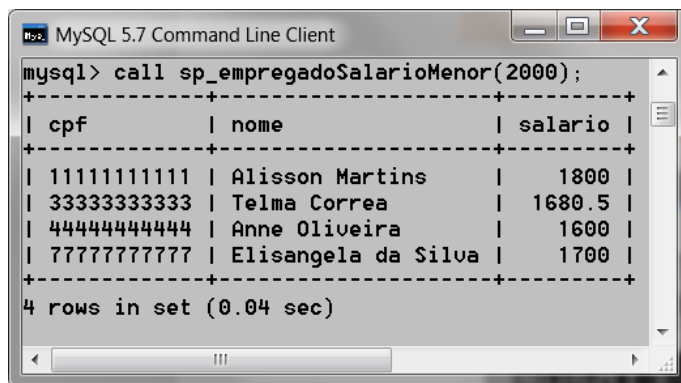
```

42) Crie um *stored procedure* para retornar o cpf, o nome e o salário dos empregados que possuem salário menor que um determinado valor especificado pelo usuário.

**Q33:**

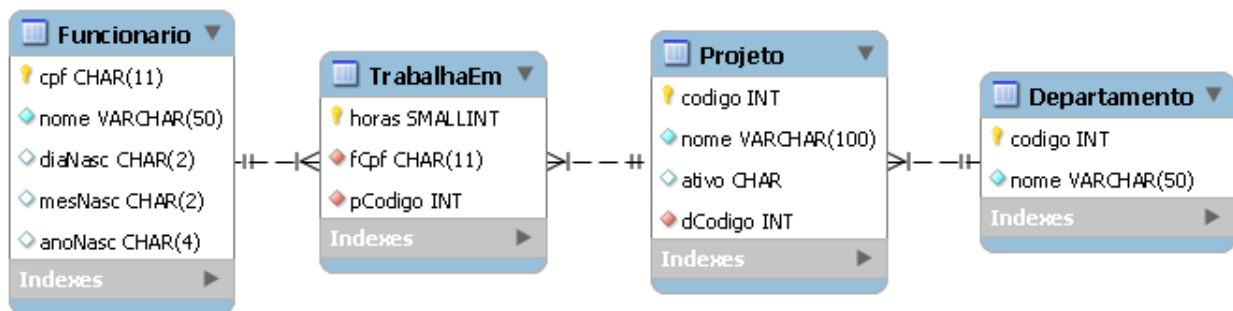
```
USE `empresa2017`;
DROP procedure IF EXISTS `sp_empregadoSalarioMenor`;
DELIMITER $$
USE `empresa2017`$$
PROCEDURE `sp_empregadoSalarioMenor`(in valor float)
BEGIN
    SELECT cpf, nome, salario FROM Empregado WHERE salario < valor;
END$$
```

Para usar `sp_empregadoSalarioMenor` deve-se proceder como a seguir:



## Exercícios

Com base no Diagrama a seguir que mostra o Modelo Físico do Banco de Dados do Sistema de Empresa Simples (SiES), então construa um **STORED PROCEDURE (SP)** para cada uma das questões a seguir.



**Observação:** no atributo **ativo**, tem-se 'S' para sim e 'N' para não.

- 1) Inserir dados para um novo funcionário.
- 2) Listar (retornar) o código e o nome do departamento conforme o código do projeto informado.
- 3) Atualizar o nome de um determinado funcionário conforme o seu CPF informado.
- 4) Retornar o nome de todos os funcionários e total de horas (somar todas as horas em todos os projetos) trabalhadas em projetos ativos.
- 5) Criar uma lista dos funcionários, contendo nome e dia, quando informado o mês.



6) Excluir um determinado projeto conforme o seu código informado. Assuma que este tipo de ação faz com que automaticamente sejam excluídos quaisquer vínculos entre funcionário e projeto, ou seja, instâncias em *TrabalhaEm* sejam também automaticamente excluídas.

7) Retornar a data de nascimento de um determinado funcionário quando informado seu CPF.

Função para concatenar: *CONCAT*. Quando usada, por exemplo, para concatenar ***primeiroNome*** e ***ultimoNome*** de uma pessoa, supondo que existam dois atributos com estes nomes em uma tabela ***Pessoa***.

```
SELECT CONCAT(primeiroNome, ultimoNome) as nome FROM Pessoa
```

8) Criar uma lista contendo o nome dos funcionários e o nome dos projetos ativos dos quais estes participam.

9) Retornar o código e o nome completo do projeto ativo quando for informado apenas parte do nome deste projeto.