
AGENDA 2

- INTRODUÇÃO AO SGBD
- IMPLEMENTAÇÃO DE BANCO DE DADOS
- INTERFACE DE COMANDO



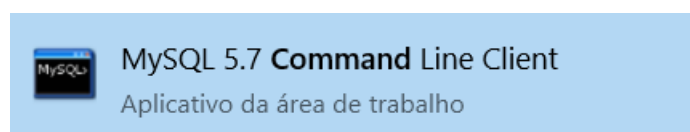
Na agenda anterior sobre Banco de Dados, você foi apresentado à Linguagem SQL, instalamos o SGBD MySQL e a ferramenta WorkBench, além de revisarmos seus conhecimentos sobre os Modelos Conceitual e Lógico.

Em relação aos SGBDs, é sempre importante lembrar que é um software utilizado para gerir Bases de Dados, permitindo criar, modificar, eliminar tabelas, além de inserir, alterar, excluir e consultar seus dados, tendo como principais características, garantir a segurança e integridade dos dados, seu compartilhamento e a recuperação de falhas (backup).

Agora vamos partir para a implementação do Banco de Dados, ou seja, o desenvolvimento do Projeto Físico, para isso aplicaremos instruções da Linguagem SQL no SGBD MySQL.

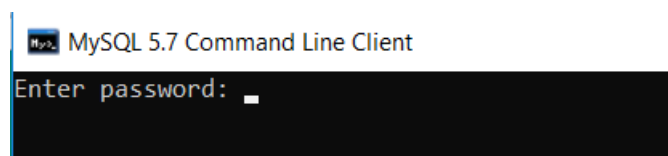
Assim como na agenda anterior, antes de cada comando será apresentada a sua **sintaxe**. Vale ainda lembrar que em linguagem de programação, quando falamos de **sintaxe**, nos referimos à **forma de escrever código fonte (palavras reservadas, comandos, recursos diversos)**. Os conteúdos entre os símbolos < > ou [] encontrados na **sintaxe** significam que os mesmos devem ser **substituídos** ou são **opcionais**, respectivamente. Vamos em frente!!!

Para iniciarmos, acesse a interface de comando do SGBD MySQL:

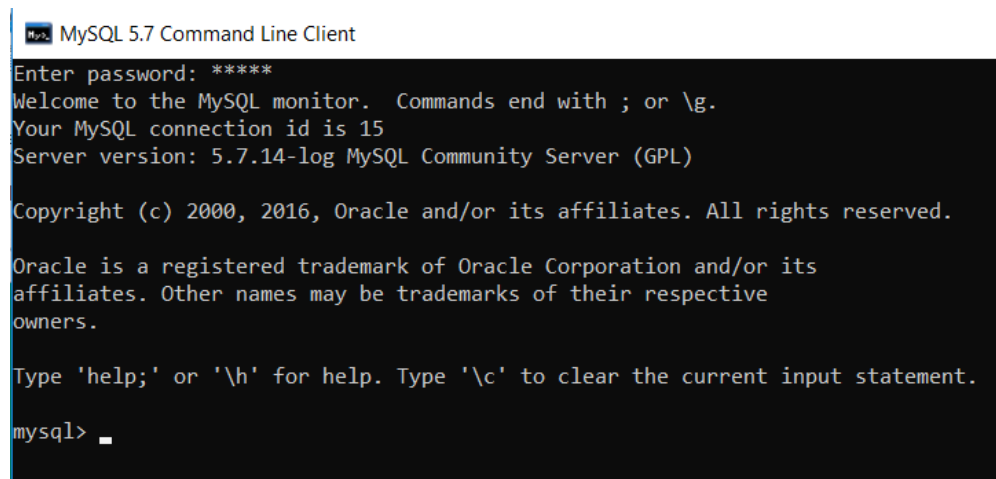


Obs.: a partir da próxima agenda utilizaremos a interface gráfica Workbench.

Será apresentada a seguinte interface:



Informe a senha definida na instalação do MySQL e tecle <ENTER>:



Tudo pronto!!! Vamos começar???

Para criar um banco de dados, utilize o comando `create database` ou `create schema`:

Sintaxe:

```
create database <nome_do_banco_de_dados>;
```

```
create schema <nome_do_banco_de_dados>;
```

Exemplo:

```
create database escola;
```

```
mysql> create database escola;
Query OK, 1 row affected (0.03 sec)

mysql>
```

```
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| escola      |
| mysql       |
| performance_schema |
| sakila      |
| sys         |
| world       |
+-----+
```

Agora que já criamos o banco, vamos criar as Tabelas, para isso utilizaremos uma parte do SQL chamada DDL (Data Definition Language), Linguagem de Definição de Dados, que possui comandos para definição e alteração de estruturas do banco de dados.

Para criar uma tabela, utilize o comando `create table`:

Veja a sintaxe:

```
create table <nome_da_tabela> (
  campo_1 tipo(<tamanho>) [<padrão>] [<restrição>]
    [<comentário>],
  .
  .
  campo_n tipo(<tamanho>) [<padrão>] [<restrição>]
    [<comentário>],
  [<constraint <nome da restrição> primary key (<campo(s) da tabela)>],
  [<constraint <nome da restrição> foreign key (<campo da tabela>)
    references <tabela de origem> (<campo origem>)],
  [<constraint <nome da restrição> unique key (<campo(s) da tabela)>]);
```

A imagem a seguir explica a composição da sintaxe da cláusula `create table`.

Aplicação:

```
create table produto
```

nome da tabela

```
(
```

```
    prod_id    int(8)
```

tipo e tamanho do campo

```
    comment 'identificador do código do produto',
```

```
    prod_descr varchar(50)
```

```
    comment 'nome do produto',
```

comentário de acordo com a função do campo

```
    prod_sigla varchar(20) not null
```

nome do campo

```
    comment 'identificação abreviada do produto',
```

restrição

```
    prod_preco double(17,2) default 0
```

valor padrão

```
    comment 'preço do produto',
```

```
    constraint pk_produto primary key (prod_id),
```

nome da restrição

```
    constraint uk_produto_prod_descr
```

campo onde será aplicada a restrição

```
        unique key (prod_descr)
```

```
);
```



Sobre chave única, é importante que você não confunda com chave primária, pois ela é utilizada quando necessitamos definir que um campo ou um conjunto de campos não podem ter seu conteúdo repetido, mas sem que ele ou eles sejam definidos como chave primária.

A seguir alguns exemplos de criação de tabelas:

Exemplo 1:

```
create table funcionario (  
    fnumero int(10) unsigned auto_increment  
        comment 'identificador do funcionario',  
    fnome varchar(80) not null  
        comment 'nome do funcionário',  
    endereco varchar(80) not null  
        comment 'endereço do funcionário',  
    salario double(10,2) default 0  
        comment 'quantidade do produto em estoque',
```

```

supernumero int(10)
    comment 'identificador do funcionário supervisor',
dnumero int(5) not null
    comment 'identificador do departamento',
constraint pk_funcionario primary key (fnumero)
);

```

Você pode estar se perguntando, onde estão as chaves estrangeiras da tabela funcionário? Não se preocupe, elas serão implementadas mais à frente. Ainda nesse primeiro exemplo, você pode perceber que foram inseridas algumas cláusulas como: **unsigned**, **auto_increment**, **not null** e **default**. Vejam os seus significados:

AUTO_INCREMENT - permite que um número único seja gerado quando um novo registro é inserido em uma tabela. Em MySQL, a palavra **AUTO_INCREMENT**, inicia com o valor 1, e se incrementa de 1 em 1.

DEFAULT - Define um valor padrão que é adicionado quando nenhum outro valor é passado.

NOT NULL - Cada linha deve conter um valor para essa coluna, valores nulos não são permitidos.

UNSIGNED - Usado para tipos numéricos, limita os dados armazenados a números positivos e zero. Por exemplo, quando queremos bloquear a inserção de valores negativos em uma coluna utilizamos o parâmetro **UNSIGNED**.

Vale ressaltar ainda a cláusula **COMMENT**, que permite que o desenvolvedor comente os campos da tabela. Veja este exemplo:

Na base de dados **escola**, digite os comandos a seguir, na linha de comando do MySQL:

```

mysql> use escola;
Database changed

```

Após a seleção do banco de dados, digite o comando para criação da tabela **funcionario**.

```

mysql> create table funcionario (
->     fnumero int(10) unsigned auto_increment
->     comment 'identificador do funcionario',
->     fnome varchar(80) not null
->     comment 'nome do funcionário',
->     endereco varchar(80) not null
->     comment 'endereço do funcionário',
->     salario double(10,2) default 0
->     comment 'quantidade do produto em estoque',
->     supernumero int(10)
->     comment 'identificador do funcionário supervisor',
->     dnumero int(5) not null
->     comment 'identificador do departamento',
->     constraint pk_funcionario primary key (fnumero)
-> );

```

```
mysql> show tables;
+-----+
| Tables_in_escola |
+-----+
| funcionario       |
+-----+
```

Para obter maiores detalhes da tabela utilize o comando **describe**.

```
describe funcionario;
```

```
mysql> describe funcionario;
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| fnumero    | int(10) unsigned | NO   | PRI | NULL    | auto_increment |
| fnome      | varchar(80)      | NO   |     | NULL    |                |
| endereco    | varchar(80)      | NO   |     | NULL    |                |
| salario     | double(10,2)     | YES  |     | 0.00    |                |
| supernumero | int(10)          | YES  |     | NULL    |                |
| dnumero     | int(5)           | NO   |     | NULL    |                |
+-----+-----+-----+-----+-----+-----+
```

No próximo exemplo, utilizaremos o conceito de **chave estrangeira**, relacionando as tabelas departamento e funcionário e de **chave única**, não permitindo que o nome do departamento seja repetido, mesmo não sendo chave primária.

Exemplo 2:

```
create table departamento (
    dnumero int(5) auto_increment
        comment 'identificador do departamento',
    dnome varchar(50) not null
        comment 'nome do departamento',
    fnumero int(10) unsigned
        comment 'identificador do funcionário gerente',
    dataini date
        comment 'data de início do gerenciamento',
    constraint pk_departamento primary key (dnumero),
    constraint fk_depto_fnumero
        foreign key (fnumero)
        references funcionario (fnumero),
    constraint uk_dnome_depto
        unique key (dnome)
);
```

campo da tabela
definido como
chave estrangeira

tabela e campo de
origem da chave
estrangeira

Lembre-se de que uma chave estrangeira deve ter o mesmo tipo e tamanho do campo da tabela de origem. No exemplo anterior, o campo **fnumero** da tabela **departamento**, tem o mesmo tipo e tamanho do campo **fnumero** da tabela **funcionario**.

```
mysql> create table departamento (  
->     dnumero int(5) auto_increment  
->     comment 'identificador do departamento',  
->     dnome varchar(50) not null  
->     comment 'nome do departamento',  
->     fnumero int(10) unsigned  
->     comment 'identificador do funcionário gerente',  
->     dataini date  
->     comment 'data de início do gerenciamento',  
->     constraint pk_departamento primary key (dnumero),  
->     constraint fk_depto_fnumero  
->     foreign key (fnumero)  
->     references funcionario (fnumero),  
->     constraint uk_dnome_depto  
->     unique key (dnome)  
-> );
```

Alteração da Tabela

Para alterar uma tabela, não se pode utilizar o comando **create table**, porque ela já existe! O comando utilizado para alterar a estrutura de uma tabela é o **alter table**.

Para alterar a estrutura, incluindo um campo, utilize:

Sintaxe:

```
alter table <nome_da_tabela>  
    add campo tipo(tamanho) [padrão] [restrição]  
    [comentário];
```

A exemplo a seguir explica a composição da sintaxe da cláusula **alter table**, quando da alteração de uma tabela para a inclusão de um campo.

Exemplo:

```
alter table funcionario  
    add email varchar(80)  
    comment 'e-mail do cliente';
```

tabela que será alterada

parte da instrução que define
as características do campo
que será incluído

```
mysql> alter table funcionario
-> add email varchar(80)
-> comment 'e-mail do cliente';
Query OK, 0 rows affected (0.96 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> describe funcionario;
+-----+-----+-----+-----+-----+-----+
| Field      | Type                | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| fnumero    | int(10) unsigned    | NO   | PRI | NULL    | auto_increment |
| fnome      | varchar(80)         | NO   |     | NULL    |                |
| endereco   | varchar(80)         | NO   |     | NULL    |                |
| salario    | double(10,2)        | YES  |     | 0.00    |                |
| supnumero  | int(10)             | YES  |     | NULL    |                |
| dnumero    | int(5)              | NO   |     | NULL    |                |
| email      | varchar(80)         | YES  |     | NULL    |                |
+-----+-----+-----+-----+-----+-----+
```

Para alterar a estrutura, modificando um campo utilize:

Sintaxe:

```
alter table <nome_da_tabela>
    modify campo tipo(tamanho) [padrão] [restrição];
```

O exemplo a seguir explica a composição da sintaxe da cláusula **alter table**, quando da alteração de uma Tabela para modificar alguma das características de um campo, neste exemplo, o tamanho.

Exemplo:

```
alter table funcionario
    modify email varchar(100);
```

parte da instrução que define o que vai ser alterado na tabela

```
mysql> alter table funcionario
-> modify email varchar(100);
Query OK, 0 rows affected (0.18 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> describe funcionario;
+-----+-----+-----+-----+-----+-----+
| Field      | Type                | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| fnumero    | int(10) unsigned    | NO   | PRI | NULL    | auto_increment |
| fnome      | varchar(80)         | NO   |     | NULL    |                |
| endereco   | varchar(80)         | NO   |     | NULL    |                |
| salario    | double(10,2)        | YES  |     | 0.00    |                |
| supnumero  | int(10)             | YES  |     | NULL    |                |
| dnumero    | int(5)              | NO   |     | NULL    |                |
| email      | varchar(100)        | YES  |     | NULL    |                |
+-----+-----+-----+-----+-----+-----+
```



Antes de qualquer alteração na estrutura de uma tabela, é importante realizar uma análise dos conteúdos existentes nos campos. Uma alteração sem critério poderá causar perda de informação como, por exemplo, a diminuição do tamanho de um campo.

Para alterar a estrutura excluindo um campo, utilize:

Sintaxe:

```
alter table <nome_da_tabela>
    drop campo;
```

O exemplo a seguir explica a composição da sintaxe da cláusula **alter table**, quando da alteração de uma tabela para a exclusão de um campo.

Exemplo:

```
alter table funcionario
    drop email;
```

parte da instrução que define o que vai ser excluído na tabela

```
mysql> alter table funcionario
-> drop email;
Query OK, 0 rows affected (0.79 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> describe funcionario;
```

Field	Type	Null	Key	Default	Extra
fnumero	int(10) unsigned	NO	PRI	NULL	auto_increment
fnome	varchar(80)	NO		NULL	
endereco	varchar(80)	NO		NULL	
salario	double(10,2)	YES		0.00	
supernumero	int(10)	YES		NULL	
dnumero	int(5)	NO		NULL	

Você se lembra que a tabela **funcionario** tem a chave estrangeira “departamento do funcionário” (**dnumero**)?

Imagine que no momento de criação da tabela **funcionario** você tenha se esquecido de fazer o relacionamento com a tabela **departamento**, precisando incluir a **chave estrangeira** na estrutura da tabela **funcionario** que já está pronta! Como faria?

Para isso, deve ser utilizado o comando **alter table**.

Analise a sintaxe a seguir:

Sintaxe:

```
alter table <nome_da_tabela>
    add constraint <nome_restrição>
        foreign key <campo>
        references <tabela_origem> (<campo_origem>);
```

O exemplo a seguir explica a composição da sintaxe da cláusula **alter table**, quando da alteração de uma tabela para a inclusão de uma restrição do tipo **chave estrangeira**.

Exemplo:

```
alter table funcionario
    add constraint fk_func_dnumero
    foreign key (dnumero)
    references departamento (dnumero);
```

parte da instrução que define as características da restrição que será incluída

```
mysql> alter table funcionario
-> add constraint fk_func_dnumero
-> foreign key (dnumero)
-> references departamento (dnumero);
```

Exclusão da Tabela

Caso você precise excluir uma tabela, deverá utilizar o comando **drop table**, conforme a sintaxe a seguir:

Sintaxe:

```
drop table <nome_da_tabela>;
```

O exemplo a seguir explica a composição da sintaxe da cláusula **drop table**, quando da exclusão de uma tabela da base de dados.

Exemplo:

```
drop table funcionario;
```

Nome da tabela que será excluída do base de

Aqui também vale lembrá-lo que a execução do comando **drop table** sem nenhum critério ou análise mais profunda, pode causar a perda permanente de dados, no exemplo acima, todos os registros da tabela **funcionario** serão perdidos.

Agora é com você!!!

Implemente o Banco de Dados de uma empresa bancária no SGBD MySQL, utilizando os comandos de DDL da Linguagem SQL, baseado no Modelo Lógico a seguir, mapeado na agenda anterior:

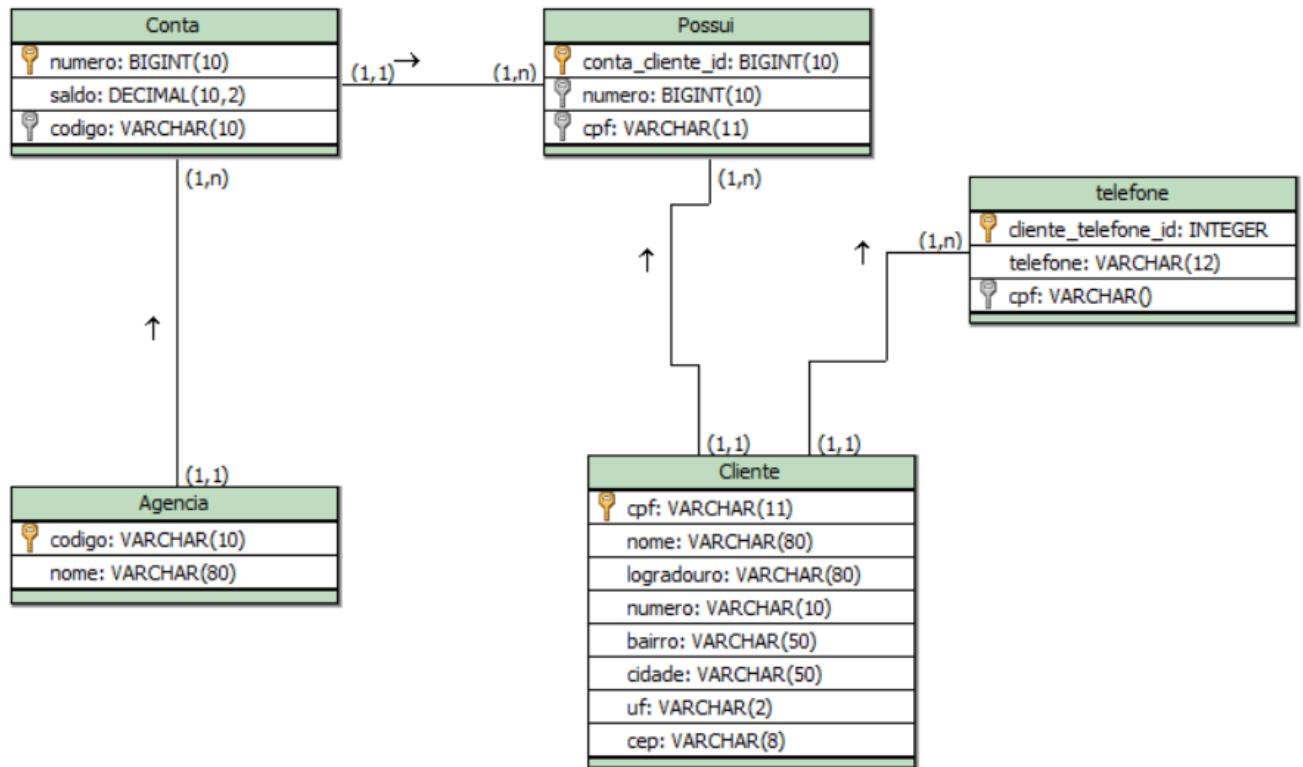


Imagem 2 – Projeto Lógico, desenvolvido no aplicativo brModelo

Agencia = {codigo, nome}

Conta = {numero, saldo, codigo}

Cliente = {cpf, nome, logradouro, numero, bairro, cidade, uf, CEP}

Conta = {numero, saldo, codigo}

Possui = {conta_cliente_id, numero, cpf}

Telefone = {conta_telefone_id, telefone, cpf}

O que achou? Foi difícil? Tenho certeza que não! Vamos ver!!!

Como o contexto é sobre uma empresa bancária, vamos criar o banco de dados definindo o nome como **banco**.

```
create database banco;
```

```
mysql> create database banco;
Query OK, 1 row affected (0.20 sec)

mysql> show databases;
+-----+
| Database |
+-----+
| information schema |
| banco      |
| mysql      |
| performance_schema |
| sakila     |
+-----+
```

Obs.: caso já tenha criado o banco de dados na agenda anterior, apenas selecione-o.

Baseado no Projeto Lógico, vamos criar as estruturas, uma dica neste momento é criar primeiro as estruturas que não possuem **chaves estrangeiras**, que é o caso de **agencia** e **cliente**.

Não esqueça de selecionar o banco de dados antes de executar o SQL para criação das estruturas.

```
use banco;
```

```
create table cliente (
    cpf varchar(11),
    nome varchar(80),
    logradouro varchar(80),
    numero varchar(10),
    bairro varchar(50),
    cidade varchar(50),
    uf varchar(2),
    cep varchar(8),
    constraint pk_cliente primary key (cpf)
);
```

```
create table agencia (
    codigo varchar(10),
    nome varchar(80) not null,
    constraint pk_agencia primary key (codigo)
);
```

```
mysql> use banco;
Database changed
mysql> create table cliente (
  -> cpf varchar(11),
  -> nome varchar(80),
  -> logradouro varchar(80),
  -> numero varchar(10),
  -> bairro varchar(50),
  -> cidade varchar(50),
  -> uf varchar(2),
  -> cep varchar(8),
  -> constraint pk_cliente
  -> primary key (cpf)
  -> );
Query OK, 0 rows affected (0.33 sec)
```

```
mysql> create table agencia (
  -> codigo varchar(10),
  -> nome varchar(80) not null,
  -> constraint pk_agencia
  -> primary key (codigo)
  -> );
Query OK, 0 rows affected (0.18 sec)
```

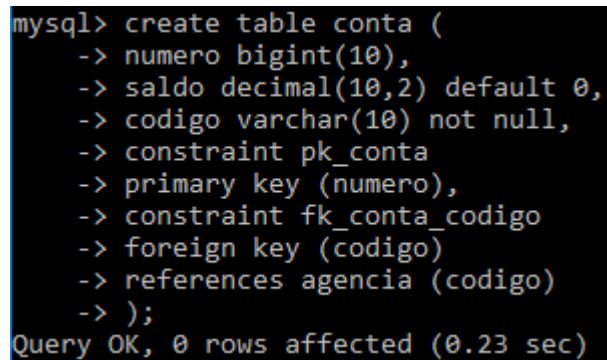
Chegou a vez das estruturas que possuem **chaves estrangeiras** (FKs), a estrutura **possui** tem duas e uma delas vem da estrutura **conta** que ainda não foi criada, por esse motivo vamos deixá-la por último.

```
create table cliente_telefone (
  cliente_telefone_id int(10) auto_increment,
  telefone varchar(12) not null,
  cpf varchar(11) not null,
  constraint pk_cliente_telefone
    primary key (cliente_telefone_id),
  constraint fk_ct_cpf
    foreign key (cpf)
    references cliente (cpf),
  constraint uk_ct_fone_cpf
    unique key (telefone, cpf)
);
```

```
mysql> create table cliente_telefone (
  -> cliente_telefone_id int(10) auto_increment,
  -> telefone varchar(12) not null,
  -> cpf varchar(11) not null,
  -> constraint pk_cliente_telefone
  -> primary key (cliente_telefone_id),
  -> constraint fk_ct_cpf
  -> foreign key (cpf)
  -> references cliente (cpf),
  -> constraint uk_ct_fone_cpf
  -> unique key (telefone, cpf)
  -> );
Query OK, 0 rows affected (0.44 sec)
```

Observe que além da PK (chave primária), na estrutura `cliente_telefone` definimos uma UK (chave única) para os campos `telefone` e `cpf`, já que o mesmo `cpf` não poderá ser vinculado ao mesmo `telefone` mais de uma vez e alteramos o nome para que fique mais fácil de identificar de quem serão os telefones armazenados nesse estrutura.

```
create table conta (  
    numero bigint(10),  
    saldo decimal(10,2) default 0,  
    codigo varchar(10) not null,  
    constraint pk_conta  
        primary key (numero),  
    constraint fk_conta_codigo  
        foreign key (codigo)  
        references agencia (codigo)  
);
```



```
mysql> create table conta (  
-> numero bigint(10),  
-> saldo decimal(10,2) default 0,  
-> codigo varchar(10) not null,  
-> constraint pk_conta  
-> primary key (numero),  
-> constraint fk_conta_codigo  
-> foreign key (codigo)  
-> references agencia (codigo)  
-> );  
Query OK, 0 rows affected (0.23 sec)
```

Agora está faltando ainda a estrutura `possui`, vamos criá-la com um nome mais sugestivo, deixando bem claro sua finalidade, `conta_cliente`, baseado no relacionamento entre as estruturas `conta` e `cliente`, assim como fizemos com a estrutura `cliente_telefone`.

```
create table conta_cliente (  
    conta_cliente_id bigint(10) auto_increment,  
    numero bigint(10) not null,  
    cpf varchar(11) not null,  
    constraint pk_conta_cliente  
        primary key (conta_cliente_id),  
    constraint fk_cc_numero  
        foreign key (numero)  
        references conta (numero),  
    constraint fk_cc_cpf  
        foreign key (cpf)  
        references cliente (cpf),  
    constraint uk_cc_numero_cpf  
        unique key (numero, cpf)  
);
```

```
mysql> create table conta_cliente (  
  -> conta_cliente_id bigint(10) auto_increment,  
  -> numero bigint(10) not null,  
  -> cpf varchar(11) not null,  
  -> constraint pk_conta_cliente  
  -> primary key (conta_cliente_id),  
  -> constraint fk_cc_numero  
  -> foreign key (numero)  
  -> references conta (numero),  
  -> constraint fk_cc_cpf  
  -> foreign key (cpf)  
  -> references cliente (cpf),  
  -> constraint uk_cc_numero_cpf  
  -> unique key (numero, cpf)  
  -> );  
Query OK, 0 rows affected (0.37 sec)
```

Na estrutura `conta_cliente`, também criamos uma **UK** (chave única), já que um cliente não poderá ter mais de um vínculo com a mesma conta.

Pronto!!! Com toda a estrutura criada, o projeto está finalizado!

Para ver todas as tabelas que você criou, utilize o comando `show tables;`

```
mysql> show tables;  
+-----+  
| Tables_in_banco |  
+-----+  
| agencia          |  
| cliente          |  
| cliente_telefone |  
| conta            |  
| conta_cliente    |  
+-----+
```

É isso aí!!! Vamos agora finalizar essa agenda colocando a mão na massa.