***Apostila MySQL***

**CREATE DATABASE** var; **→** Comando utilizado para criar banco de dados.

**CREATE TABLE** var( )**;** → Comando utilizado para criar tabelas.

**DESCRIBE** var**;** or **DESC** var; → Comando utilizado para descrever a estrutura de toda a tabela.

**SHOW DATABASES**; → Comando utilizado para verificar todos os Bancos de Dados criados no sistema.

**USE** var**;** → Comando utilizado para entrar ou selecionar um Banco de Dados.

Apenas no terminal : **status**; → Comando utilizado para analisar as informações do Banco de Dados, inclusive qual Banco de Dados está aberto.

**SHOW TABLES**; → Comando utilizado para analisar quais tabelas estão criadas no Banco de Dados selecionado.

**DROP DATABASE** var; → Comando utilizado para apagar um Banco de Dados.

**CREATE DATABASE** var

**DEFAULT CHARACTER SET utf8**

**DEFAULT COLLATE utf8\_general\_ci** → Comando utilizado para criar um Banco de Dados e fazer o padrão de letras do Banco de Dados ser de uma coleção “UTF8”.

CREATE TABLE pessoas(

nome varchar(30),

nascimento,

sexo,

peso,

altura,

nacionalidade

)**default charset = utf8**; → Comando utilizado para configurar o conjunto de letras padrão da tabela criada para a coleção “UTF8”.

nome varchar(30) **not null**; → Comando utilizado para obrigar o usuário a digitar algo.

nacionalidade varchar(20) **default** var; → Comando utilizado para preencher um espaço vazio com um texto específico.

CREATE TABLE pessoas(

var **int not null auto\_increment,**

nome varchar(30) NOT NULL,

nascimento date,

sexo enum('M','F'),

peso decimal(5,2),

altura decimal(3,2),

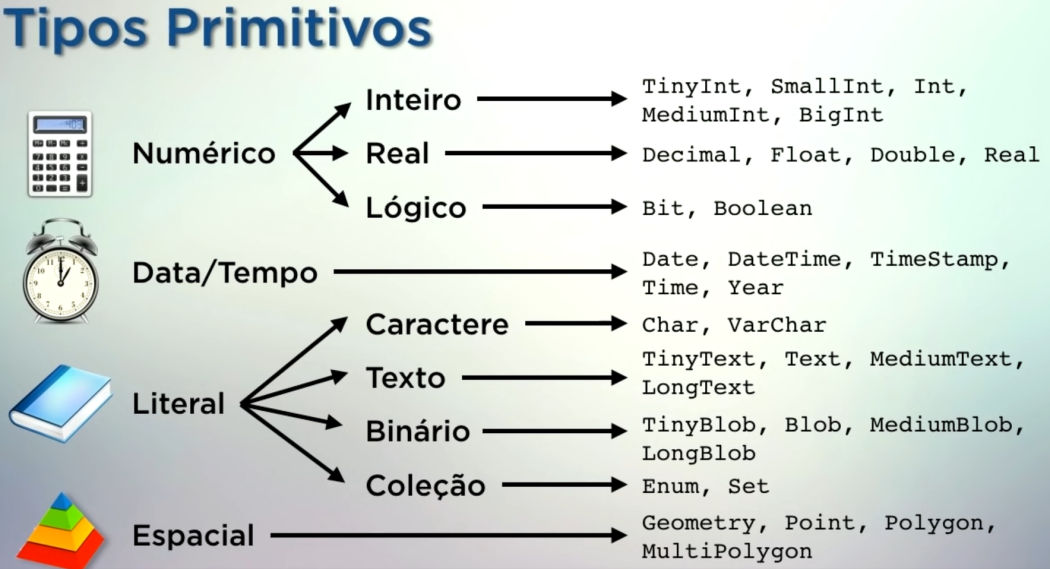
nacionalidade varchar(20) default 'Brasil',

**primary key** (var)

)default charset = utf8; →

Comando utilizado para fazer a criação da Chave Primária em uma tabela do Banco de Dados.

**Tipos Primitivos:**

****

**INSERT INTO** var

(nome, nascimento, sexo, peso, altura, nacionalidade)

**VALUES**

('Godofredo','1984-01-30','M','78.5','1.83','Brasil'); →

Comando utilizado para inserir dados as colunas de uma tabela.

**SELECT \* FROM** var; → Comando utilizado para analisar todos os dados de uma tabela.

**INSERT INTO** var **VALUES**

(**default**,'Creuza','1920-12-30','F','50.2','1.65',**default**); →

Caso a ordem de seleção dos dados nos campos da tabela seja sequencial, incluindo todos os dados é possível abreviar o código desta maneira.



**ALTER TABLE** var

**ADD COLUMN** var varchar(10); → Comando utilizado

para adicionar campos ( colunas ) nas tabelas.

**ALTER TABLE** var

**DROP COLUMN** var; → Comando utilizado para excluir uma

coluna criada na tabela.

alter table pessoas

add column profissao varchar(20) **after** nome; →

Comando utilizado para criar uma coluna na tabela em um lugar específico. No caso acima, na frente ou **after** ( depois ) da outra coluna mencionada no final do código.

alter table pessoas

add column codigo int **first**; → Comando utilizado para

criar colunas na tabela na primeira posição da tabela.

**ALTER TABLE** var

**MODIFY COLUMN** var var varchar(20); → Comando

utilizado para alterar os dados de alguma coluna já criada na tabela.

**ALTER TABLE** var

**CHANGE COLUMN** var var1 varchar(20) **not null**; →

Comando utilizado para fazer alterações em uma coluna de

uma tabela de uma maneira mais profunda.

**ALTER TABLE** var

**RENAME TO** var; → Comando utilizado para alterar o

nome de uma tabela.

CREATE TABLE **IF NOT EXISTS** gafanhotos ( teste int ) →

Comando utilizado para criar um uma tabela apenas se ela não existir.

CREATE TABLE GAFANHOTOS (

nome varchar(20) not null **unique**,

descricao text,

carga int,

totaulas int,

ano year,

) CHARSET utf8; → Comando utilizado para não deixar

que na seleção de dados seja informado dois dados com o mesmo valor no campo descrito.

CREATE TABLE GAFANHOTOS (

nome varchar(20) not null unique,

descricao text,

carga int **unsigned**,

totaulas int,

ano year,

) CHARSET utf8; → Comando utilizado para não permitir

armazenamento de números negativos no campo específico, fazendo a economia de pelo menos 1 byte a cada dado informado.

DROP TABLE **IF EXISTS** nome\_da\_tabela; → Comando

utilizado para apagar uma tabela caso ela exista.

**UPDATE** nome\_da\_tabela

**SET** coluna\_da\_tabela **=** dado

**WHERE ID =** numero\_do\_id → Comando

utilizado para alterar um dado específico em alguma tabela.

**UPDATE** nome\_da\_tabela

**SET COLUNA\_DA\_TABELA =** dado, coluna\_da\_tabela **=** dado, coluna\_da\_tabela **=** dado

**WHERE ID =** numero\_do\_id

**LIMIT 1;** →

Comando utilizado para alterar mais de um dado numa mesma linha da tabela.

**DELETE FROM** nome\_da\_tabela

**WHERE ID =** numero\_do\_id**;** → Comando utilizado para

apagar uma linha específica da tabela.

**DELETE FROM** nome\_da\_tabela

**WHERE ID IN** (‘dado’,’dado,’dado’) → Comando

utilizado para apagar várias linhas da tabela a partir de seus id específicos.

**DELETE FROM** nome\_da\_tabela

**WHERE** coluna\_da\_tabela **=** dado\_em\_comum**; →**

Comando utilizado para apagar várias linhas da tabela a partir de uma característica similar entre elas.

**TRUNCATE TABLE** nome\_da\_tabela**;** or  **TRUNCATE** nome\_da\_tabela**;** →

Comando utilizado para apagar todos os dados da tabela.

**SHOW CREATE TABLE** nome\_da\_tabela**; →** Comando

utilizado para ver o comando que foi usado para criar uma tabela já existente.

**SELECT \* FROM** nome\_da\_tabela

**ORDER BY** coluna\_da\_tabela →

Comando utilizado para ordenar uma tabela de acordo com uma coluna específica.

**SELECT \* FROM** nome\_da\_tabela

**ORDER BY** coluna\_da\_tabela **DESC** →

Comando utilizado para ordenar uma tabela de acordo com uma coluna específica em ordem decrescente.

**SELECT \* FROM** nome\_da\_tabela

**ORDER BY** coluna\_da\_tabela **ASC** →

Comando utilizado para ordenar uma tabela de acordo com uma coluna específica em ordem crescente.

**SELECT** coluna\_da\_tabela **,** coluna\_da\_tabela **FROM** nome\_da\_tabela →

Comando utilizado para verificar os dados de uma tabela apresentando como resultado colunas específicas.

SELECTcoluna\_da\_tabela **,** coluna\_da\_tabelaFROMnome\_da\_tabela

**ORDER BY** coluna\_da\_tabela **,** coluna\_da\_tabela

→

Comando utilizado para ordenar o resultado da consulta de uma tabela ordenando a partir de de duas colunas.

**Obs.:** Ele ordenada da primeira coluna até a última.

SELECTcoluna\_da\_tabela **,** coluna\_da\_tabelaFROMnome\_da\_tabela

**WHERE** nome\_da\_tabela operadordado\_específico

ORDER BYcoluna\_da\_tabela **,** coluna\_da\_tabela →

Comando utilizado para filtrar um dado da tabela a partir de um dado específico.

**Obs.:** O operador pode ser ( **= , > , < , <= , >= , !** ) .

SELECTcoluna\_da\_tabela **,** coluna\_da\_tabelaFROMnome\_da\_tabela

**WHERE** coluna\_da\_tabela\_númerica

**BETWEEN** dado\_específico\_de\_ínicio **AND** dado\_específico\_final

ORDER BYcoluna\_da\_tabela **,** coluna\_da\_tabela →

Comando utilizado para verificar os dados em uma consulta a partir de dados filtrados por uma coluna númerica com um ínicio e um final.

**Exemplo:**

**WHERE** ano **BETWEEN** 2014 **AND** 2016

SELECTcoluna\_da\_tabela **,** coluna\_da\_tabelaFROMnome\_da\_tabela

**WHERE** coluna\_da\_tabela\_númerica **IN (** dado\_especifico **,** dado\_especifico **)**

ORDER BYcoluna\_da\_tabela **,** coluna\_da\_tabela →

Comando utilizado para verificar dados em consulta em uma tabela a partir de dados específicos em uma coluna.

SELECTcoluna\_da\_tabela **,** coluna\_da\_tabelaFROMnome\_da\_tabela

**WHERE** coluna\_da\_tabela operador dado\_especifico condiçãocoluna\_da\_tabela operador dado\_especifico

ORDER BYcoluna\_da\_tabela **,** coluna\_da\_tabela →

Comando utilizado para verificar dados em consulta em uma tabela a partir de dados específicos em uma ou mais colunas.

**Exemplo:**

**WHERE** carga **>** 35 **or** totaulas **<** 30;

**SELECT \* FROM** nome\_da\_tabela

**WHERE** coluna\_da\_tabela **LIKE** ‘letras%’ **;** →

Comando utilizado para fazer uma consulta na tabela de dados de uma coluna específica que começam com letras específica.

**SELECT \* FROM** nome\_da\_tabela

**WHERE** coluna\_da\_tabela **not**  **LIKE** ‘letras%’ **;** →

Comando utilizado para fazer uma consulta na tabela de dados de uma coluna específica que não começam com letras específica.

**SELECT \* FROM** nome\_da\_tabela

**WHERE** coluna\_da\_tabela **LIKE** ‘%letras’ **;** →

Comando utilizado para fazer uma consulta na tabela de dados de uma coluna específica que terminam com letras específica.

**SELECT \* FROM** nome\_da\_tabela

**WHERE** coluna\_da\_tabela **not LIKE** ‘%letras’ **;** →

Comando utilizado para fazer uma consulta na tabela de dados de uma coluna específica que não terminam com letras específica.

**SELECT \* FROM** nome\_da\_tabela

**WHERE** coluna\_da\_tabela **LIKE** ‘%letras%’ **;** →

Comando utilizado para fazer uma consulta na tabela de dados de uma coluna específica que tenham letras específica.

**SELECT \* FROM** nome\_da\_tabela

**WHERE** coluna\_da\_tabela **not** **LIKE** ‘%letras%’ **;** →

Comando utilizado para fazer uma consulta na tabela de dados de uma coluna específica que não tenham letras específica.

**SELECT \* FROM** nome\_da\_tabela

**WHERE** coluna\_da\_tabela **LIKE** ‘letras%p\_’ **;** →

Comando utilizado para fazer uma consulta na tabela de dados de uma coluna específica que tenham letras específicas no começo e no final e depois da letra final tenham números ou caracteres.

**SELECT \* FROM** nome\_da\_tabela

**WHERE** coluna\_da\_tabela **LIKE** ‘letras(intervalo de letras)letras%’ **;** →

Comando utilizado para fazer uma consulta na tabela de dados de uma coluna específica que tenham letras específicas no começo, um intervalo de letras e após outra letra ou um conjunto de letras específicos.

SELECT coluna\_específica FROM nome\_da\_tabela

**WHERE SUBSTRING\_INDEX(**coluna\_específica**, ‘ ’, 1) = LIKE '%**letra\_específica**';** →

Comando utilizado para consultar dados onde suas primeiras instâncias terminam com uma letra específica.

**SELECT DISTINCT** coluna\_da\_tabela **FROM gafanhotos**

**ORDER BY** coluna\_da\_tabela**; →**

Comando utilizado para verificar cada dado único de colunas específicas em uma tabela.

**SELECT COUNT(\*) FROM** nome\_da\_tabela**; →**

Comando utilizado para verificar quantos registros existem em uma tabela.

**SELECT MAX(**coluna\_da\_tabela**) FROM** nome\_da\_tabela**; →**

Comando utilizado para verificar qual o valor máximo de uma coluna específica da tabela.

**SELECT MIN(**coluna\_da\_tabela**) FROM** nome\_da\_tabela**; →**

Comando utilizado para verificar qual o valor mínimo de uma coluna específica da tabela.

**SELECT SUM(**coluna\_da\_tabela**) FROM** nome\_da\_tabela**; →**

Comando utilizado para somar os valores de uma coluna específica da tabela.

**SELECT AVG(**coluna\_da\_tabela**) FROM** nome\_da\_tabela**; →**

Comando utilizado para tirar a média dos valores de uma coluna específica da tabela.

**SELECT**  **TIMESTAMPDIFF**(**YEAR**, coluna\_com\_as\_datas, **CURDATE**()) **AS** idade **FROM** nome\_da\_tabela; →

Comando utilizado para verificar a idade de uma pessoa.

**SELECT** primeira\_coluna\_específica, **COUNT(\*)** from nome\_da\_tabela

**GROUP BY** segunda\_coluna\_específica

**ORDER BY** pk(Primary Key)**;** →

Comando utilizado para verificar o total de registros de uma coluna específica na tabela, ordenando pelo número identificador “id”.

**SELECT** coluna\_específica, **COUNT(\*)** from nome\_da\_tabela

**GROUP BY** coluna\_específica**;**

**HAVING** coluna\_específica condição →

Comando utilizado para verificar o total de registros de uma coluna específica na tabela através de uma condição, ordenando pelo número identificador “id”.

**SELECT** coluna\_específica, **COUNT(\*)** from nome\_da\_tabela

**WHERE** segunda\_coluna\_específica condição

**GROUP BY** coluna\_específica**;**

**HAVING** coluna\_específica condição **( SELECT AVG(**coluna\_específica**) FROM** nome\_da\_tabela**);** →

Comando utilizado para verificar em uma tabela o total de registros de uma coluna específica, a partir do sinal de condição de outra coluna específica, agrupando os registros com todos os dados da primeira coluna mencionada, filtrando os registros quando os dados estiverem acima da média de dados da coluna mencionada.

**GRANT ALL PRIVILEGES** → Comando utilizado para gerar

permissão total de um usuário.

**GRANT** tipo\_de\_permissão**,** tipo\_de\_permissão **ON** nome\_do\_banco de\_dados**.\* TO ‘**nome\_do\_usuário**’@’**ip\_do\_usuário**’;** →

Comando utilizado para dar permissões específicas a um usuário.

**REVOKE ALL PRIVILEGES** → Comando utilizado para

revogar as permissões totais de um usuário.

**REVOKE** tipo\_de\_permissão**,** tipo\_de\_permissão **ON** nome\_do\_banco de\_dados**.\* TO ‘**nome\_do\_usuário**’@’**ip\_do\_usuário**’;** →

Comando utilizado para revogar permissões específicas de um usuário.

**SHOW GRANTS FOR ‘**nome\_do\_usuário**’@’**ip\_do\_usuário**’;** →

Comando utilizado para verificar as permissões de um usuário específico.

**FLUSH ALL PRIVILEGES** → Comando utilizado para

atualizar os privilégios após alterações.

**START TRANSACTION** → Comando utilizado para iniciar

uma transação.

**START** **TRANSACTION**;

INSERT INTO nome\_da\_tabela

(coluna\_da\_tabela)

VALUES

('dado');

COMMIT;

ROLLBACK; → Comando utilizado para iniciar uma

transação.

START TRANSACTION;

INSERT INTO nome\_da\_tabela

(coluna\_da\_tabela)

VALUES

('dado');

**COMMIT;**

ROLLBACK; → Comando utilizado para confirmar

uma transação.

START TRANSACTION;

INSERT INTO nome\_da\_tabela

(coluna\_da\_tabela)

VALUES

('dado');

COMMIT;

**ROLLBACK;**  → Comando utilizado para desfazer

uma transação caso algo de errado durante o processo.

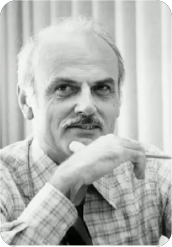
***História do Banco de Dados***

***( Modelo Relacional )***



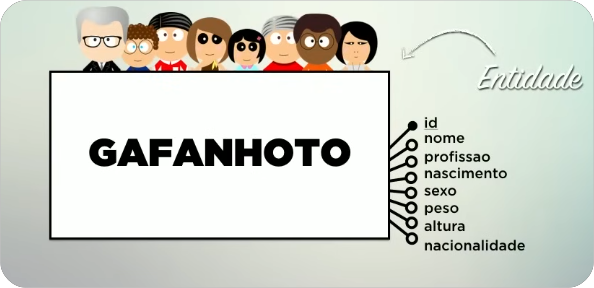
Na década de 60 o Governo dos Estados Unidos e a IBM fizeram um evento chamado **CODASYL** que foi onde nasceu a Linguagem de Programação **COBOL** e os conceitos de Banco de Dados.

Nessa época surgiu o **Modelo Hierárquico** e **Modelo em Rede**. Esses conceitos eram muito importantes na época, mas com o passar do tempo foram ficando para trás, pois se tornaram pouco eficientes. Então na década de 70 um dos engenheiros da IBM o Edgar F.Codd propôs a criação de um Banco de Dados um pouco diferenciado onde muito mais que ligações teriam relações entre eles; esse modelo se chamou **Modelo Relacional**.

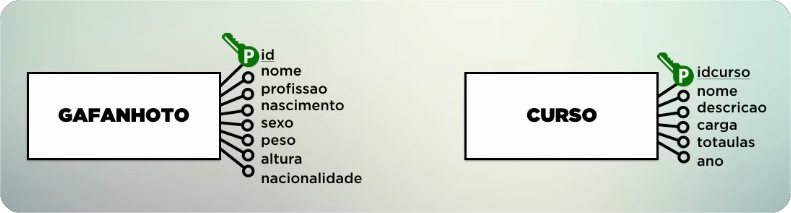


No Modelo Relacional começamos a ter relações entre Banco de Dados.

Em um Banco de Dados podemos por exemplo podemos cadastrar uma pessoa, e a partir disso temos um registro; assim acessando os dados dessa pessoa podemos verificar onde ela mora, que tipos de contas ela fez e o quanto isso afetou no meu estoque.

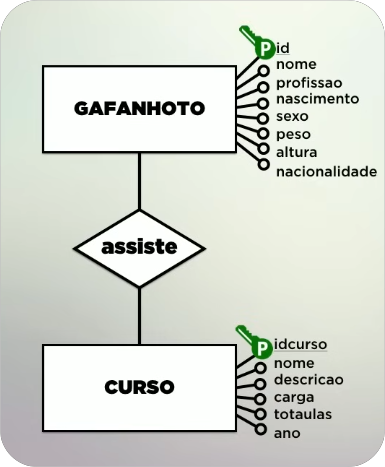


Podemos armazenar diversas pessoas dentro uma entidade ( tabela ), e dentro dessas entidades podemos armazenar diversos registros ( colunas ).



Dentro uma entidade temos sempre um atributo primário geralmente identificado por “ID”. Chamamos esse atributo de **Primary Key** ou Chave Primária.

As entidades também podem se relacionar, e existem diversas maneiras delas se relacionarem.

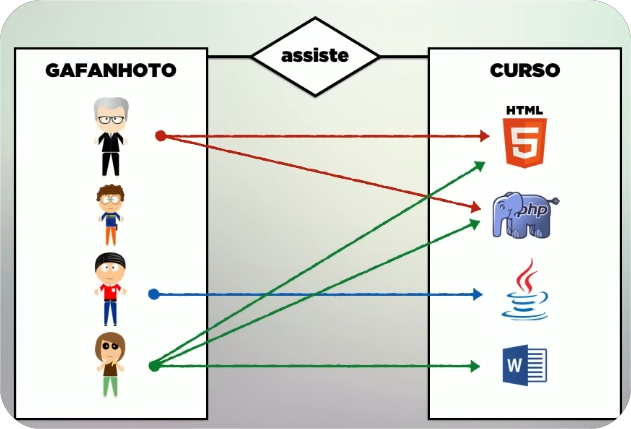


Aqui vamos dar foco a Relacionamentos Múltiplo, em específico Relacionamentos Duplos, com apenas 2 entidades no caso.

Damos o nome para essa estrutura lógica de **Diagrama E-R**, **Diagrama Entidade - Relacionamento** ou **DER**.

Sempre que ver uma entidade veja como um container, onde dentro deste temos certos tipos de dados que chamamos de **Campo**.

Cada entidade pode se relacionar com qualquer registro de outra tabela, e quantos registros ela precisar.

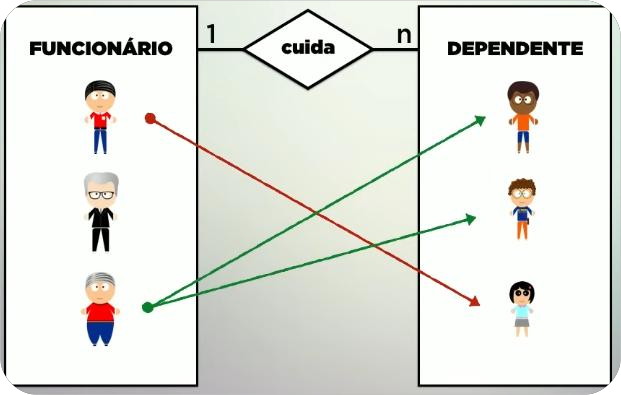


A quantidade de registros que uma entidade pode se relacionar em outra entidade é chamada de **cardinalidade**. E essas podem ser representadas por **1** ( uma) que é a cardinalidade simples ou **n** ( muitas ) que é cardinalidade múltiplas. Existem muitas outras cardinalidades, mas aqui vamos dar mais ênfase a estas.

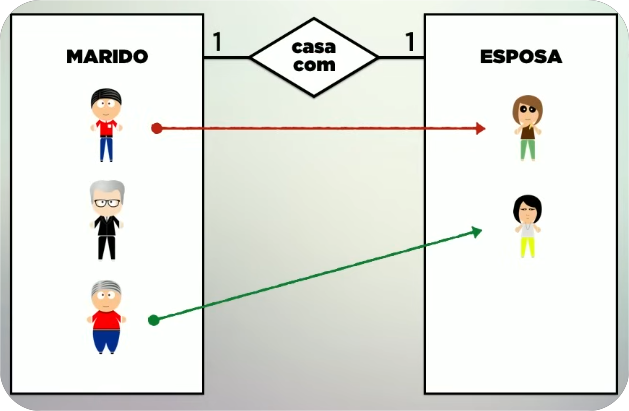
Podemos ter relacionamentos **many-to-many** ( muitos para muitos ).



Podemos ter também relacionamentos **many-to-one** ( muitos para um ) ou **one-to-many** ( um para muitos ).



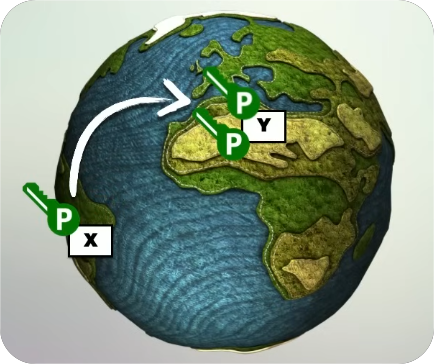
E por último podemos ter o relacionamento **one-to-one** ( um para um ).



Para fazer esses relacionamentos vamos necessitar de outro tipo de chave que se chama **Foreign Key** ou Chave Estrangeira.

Imagine o planeta terra. De um lado do planeta temos a entidade **X** e do outro lado do planeta temos a chave **Y**.

Se quisermos que a entidade X que tem uma chave **PK** ( Primary Key ) e Y que também tem uma chave **PK** ( Primary Key ) se relacione, precisamos que a chave de X crie uma cópia e deixe essa chave em Y, e essa cópia se chama **FK** ( Foreign Key ).

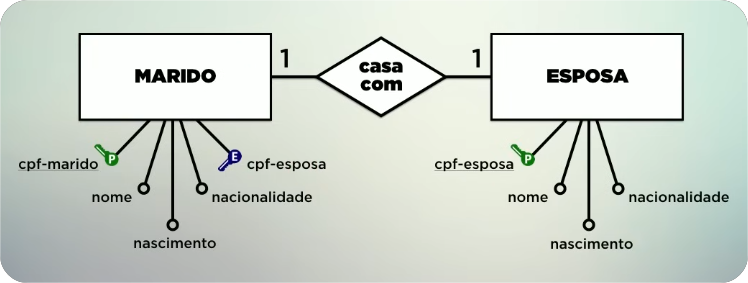


Para criar por exemplo um relacionamento one-to-one precisamos pensar que podemos juntar tudo em uma tabela só.

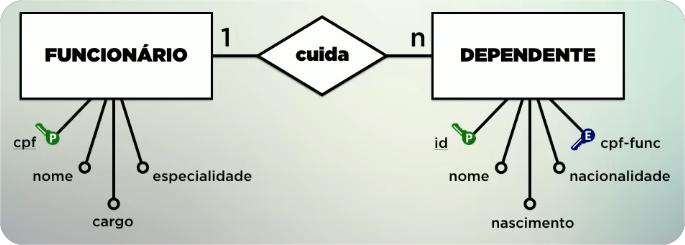
Para isso precisamos pensar primeiramente em uma entidade dominante.

Então pegamos a chave primária de uma das entidades e transferimos sua copia FK para a outra entidade.

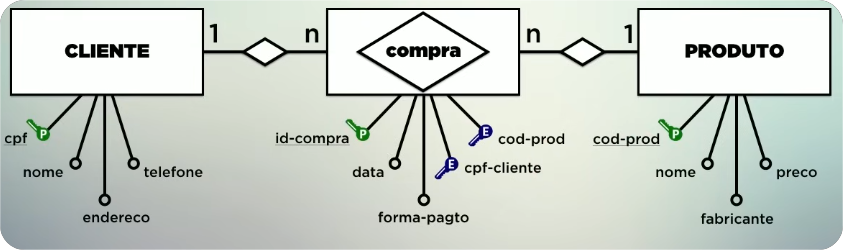
O nome da FK nesse caso será o mesmo nome da chave PK na entidade original.



Já para relacionamentos one-to-many ou many-to-one, fazemos da mesma maneira do relacionamento one-to-one porém a chave estrangeira seguirá outro padrão de nomenclatura.



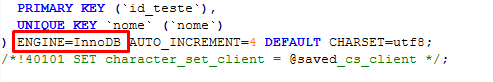
Para relacionamentos many-to-many faremos o relacionamento virar uma entidade e está se relacionará com ambas as outras duas entidades e após isso aplicamos a PK como em many-to-one.



A partir desse conhecimento agora podemos aprender outro conceito que se chama JOIN.

Quando criamos uma Tabela SQL, precisamos definir um **ENGINE** que é a máquina que irá criar registros.

Ao abrir por exemplo um Backup de um arquivo SQL, vamos nos deparar com essa configuração quase no final do código...



Percebemos que por padrão a **ENGINE** é a InnoDB, que além de uma máquina é um mecanismo de criação de tabelas.

O InnoDB foi criado pelo InnoBase que hoje pertence a Oracle, e este é o mecanismo que permiti criar tabelas com algumas características importantes, como suportar Chaves Estrangeiras ( FK ).

Você também pode escolher usar outros mecanismos como o **MyISAM**, **XtrapDB**.

Muitas empresas usavam o **MyISAM**, porém este não trazia características importantes em transações como o **ACID** ( Atomicidade, Consistência, Isolamento e Durabilidade ), então começou a ser implementado com mais frequência o **InnoDB** e o **XtrapDB** que suportam o ACID.

Toda transação precisar seguir as caracteríscas do ACID:

**Atomicidade →** Ela precisar atómica; ou seja, ela não pode ser dívida em sub-tarefas. A transação precisa ser feito por completo.

**Consistência →** O Banco de Dados precisa continuar sem erros após a transação. Ele não pode ter falhas, e não pode passar a violar nenhuma regra.

**Isolamento →** Quando as transações são feitas em multi-tarefas; ou seja, quando dois usuários fazem transações ao mesmo tempo, as transações não podem interferir uma na outra.

**Durabilidade →** As transações precisam durar o tempo que for necessário.

**ALTER** **TABLE** nome\_da\_tabela

**ADD** **FOREIGN** **KEY** **(**nome\_da\_coluna\_FK**)**

**REFERENCES** nome\_da\_tabela\_estrangeira **(**nome\_da\_coluna(PK)\_da\_tabela\_estrangeira**);**  → Comando

utilizado para transformar uma coluna específica em Chave Estrangeira, criando um relacionamento com outra tabela específica.

**INNER JOIN:**

**SELECT** \*

**FROM** nome\_da\_tabela

**INNER JOIN** nome\_da\_tabela estrangeira **ON** nome\_da\_tabela\_estrangeira**.**pk\_da\_tabela\_estrangeira = nome\_da\_tabela**.**nome\_da\_coluna\_FK; →

Comando utilizado para verificar os dados da Tabela Estrangeira referenciados a partir da Coluna Estrangeira criada na tabela principal. Aqui serão apresentadas apenas os campos da tabela principal que tem referência com a tabela estrangeira.

**OUTER JOIN ON:**

**SELECT** \*

**FROM** nome\_da\_tabela

**LEFT OUTER JOIN** nome\_da\_tabela estrangeira **ON** nome\_da\_tabela\_estrangeira**.**pk\_da\_tabela\_estrangeira = nome\_da\_tabela**.**nome\_da\_coluna\_FK; → Comando

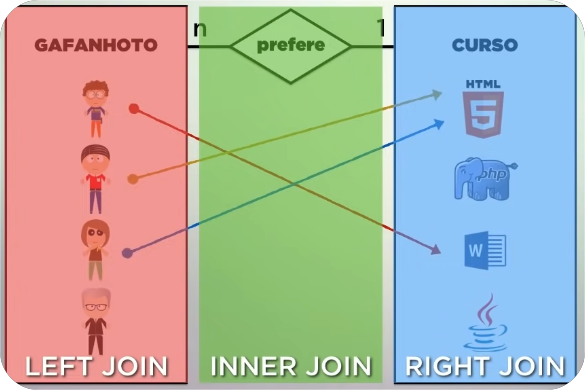
utilizado para verificar os dados da Tabela Estrangeira referenciados a partir da Coluna Estrangeira criada na tabela principal. Aqui serão apresentados todos os campos de ambas as tabelas referenciando a Tabela Principal.

**SELECT** \*

**FROM** nome\_da\_tabela

**RIGHT OUTER JOIN** nome\_da\_tabela estrangeira **ON** nome\_da\_tabela\_estrangeira**.**pk\_da\_tabela\_estrangeira = nome\_da\_tabela**.**nome\_da\_coluna\_FK; → Comando

utilizado para verificar os dados da Tabela Estrangeira referenciados a partir da Coluna Estrangeira criada na tabela principal. Aqui serão apresentados todos os campos de ambas as tabelas referenciando a Tabela Estrangeira.



**SELECT** \*

**FROM** nome\_da\_tabela\_primaria

**INNER** **JOIN** nome\_da\_tabela\_intermediaria

**ON** nome\_da\_tabela\_de\_referencia**.**id(FK\_da\_tabela\_primária) **=** nome\_da\_tabela\_intermediaria**.**id(PK)

**INNER** **JOIN** nome\_da\_tabela\_secundaria

**ON** nome\_da\_tabela\_intermediaria**.**id(FK\_da\_tabela\_secundaria) = nome\_da\_tabela\_secundaria**.**(PK); →

Comando utilizado para consultar os dados a partir de um relacionamento **many-to-many(muitos-para-muitos).**

**OBS.:** Nesse caso precisamos criar uma tabela intermediária que se relacionará com ambas as tabelas.