

## **1. Introdução: Dados, Informação e conhecimento**

Dados, informação e conhecimento, lidamos com esses conceitos o tempo todo, seja em casa, nas empresas, escolas, igreja, etc. Ouvimos muitos termos relacionados como processamento de dados, sistemas de informação, gestão de conhecimento, arquitetura da informação, coleta de dados, base de conhecimentos, entre outros. Mas qual a diferença entre Dados, Informação e Conhecimento?

### **1.1. Dados**

Dados são códigos que constituem a matéria prima da informação, ou seja, é a informação não tratada. Os dados representam um ou mais significados que isoladamente não podem transmitir uma mensagem ou representar algum conhecimento.

Em uma pesquisa eleitoral por exemplo, são coletados dados, isto é, cada participante da pesquisa fornece suas opiniões e escolhas sobre determinados candidatos, mas essas opiniões não significam muita coisa no âmbito da eleição. Só depois de ser integrada com as demais opiniões é que teremos algo significativo.

### **1.2. Informações**

Informações são dados tratados. O resultado do processamento de dados são as informações. As informações têm significado, podem ser tomadas decisões ou fazer afirmações considerando as informações.

No exemplo da pesquisa eleitoral, os pesquisadores retêm dados dos entrevistados, mas quando inseridos nos sistemas e processados produzem informações e essas informações diz que tem mais chance de ser eleito, entre outras.

Desta forma podemos dizer que as informações é o conjunto de dados que foram processados, seja por meio eletrônico, mecânico ou manual e que produziu um resultado com significado.

### **1.3. Conhecimento**

O conhecimento vai além de informações, pois ele além de ter um significado tem uma aplicação.

Conhecimento é o ato ou efeito de abstrair ideia ou noção de alguma coisa, como por exemplo: conhecimento das leis; conhecimento de um fato (obter informação); conhecimento de um documento; termo de recibo ou nota em que se declara o aceite de um produto ou serviço; saber, instrução ou cabedal científico (homem com grande conhecimento).

As informações são valiosas, mas o conhecimento constitui um saber. Produz ideias e experiências que as informações por si só não será capaz de mostrar. Se informação é dado trabalhado, então conhecimento e informação trabalhada.

## **2. O que é um banco de dados?**

Bancos de dados, (ou bases de dados), são conjuntos de dados com uma estrutura regular que organizam informação. Um banco de dados normalmente agrupa informações utilizadas para um mesmo fim.

Um banco de dados é usualmente mantido e acessado por meio de um software conhecido como Sistema Gerenciador de Banco de Dados (SGBD). Muitas vezes o termo banco de dados é usado como sinônimo de SGBD.

O modelo de dados mais adotado hoje em dia é o modelo relacional, onde as estruturas têm a forma de tabelas, compostas por linhas e colunas.

Resumindo, um banco de dados é uma coleção de dados relacionados. Entende-se por dado, toda a informação que pode ser armazenada e que apresenta algum significado implícito dentro do contexto ao qual ele se aplica. Por exemplo, num sistema bancário, uma pessoa é identificada pelo seu CPF (cliente). Em um sistema escolar a pessoa é identificada pelo seu número de matrícula (aluno). Além disso, os dados que serão armazenados em cada situação podem diferir consideravelmente.

## **3. Modelo Relacional**

O modelo relacional é uma teoria matemática desenvolvida por Edgar Frank Codd para descrever como as bases de dados devem funcionar. Embora esta teoria seja a base para o software de bases de dados relacionais, muito poucos sistemas de gestão de bases de dados seguem o modelo de forma restrita, e todos têm funcionalidades que violam a teoria, desta forma

variando a complexidade e o poder. A discussão se esses bancos de dados merecem ser chamados de relacional ficou esgotada com tempo, com a evolução dos bancos existentes.

De acordo com a arquitetura ANSI / SPARC em três níveis, os Bancos de dados relacionais consistem de três componentes:

- uma coleção de estruturas de dados, formalmente chamadas de relações, ou informalmente tabelas, compondo o nível conceitual;
- uma coleção dos operadores, a álgebra e o cálculo relacionais, que constituem a base da linguagem SQL; e
- uma coleção de restrições da integridade, definindo o conjunto consistente de estados de base de dados e de alterações de estados. As restrições de integridade podem ser de quatro tipos:
  - domínio (ou tipo de dados),
  - atributo,
  - relações,
  - restrições de base de dados.

De acordo com o Princípio de Informação: toda informação tem de ser representada como dados; qualquer tipo de atributo representa relações entre conjuntos de dados.

Nos bancos de dados relacionais os relacionamentos entre as tabelas não são codificados explicitamente na sua definição. Em vez disso, se fazem implicitamente pela presença de atributos chave. As bases de dados relacionais permitem aos utilizadores (incluindo programadores) escreverem consultas (queries), reorganizando e utilizando os dados de forma flexível e não necessariamente antecipada pelos projetistas originais. Esta flexibilidade é especialmente importante em bases de dados que podem ser utilizadas durante décadas, tornando as bases de dados relacionais muito populares no meio comercial.

Um dos pontos fortes do modelo relacional de banco de dados é a possibilidade de definição de um conjunto de restrições de integridade. Estas definem os conjuntos de estados e mudanças de estado consistentes do banco de dados, determinando os valores que podem e os que não podem ser armazenados.

#### **4. Aplicações de bancos de dados**

Bancos de dados são usados em muitas aplicações, enquanto atravessando virtualmente a gama inteira de software de computador. Bancos de dados são o método preferido de armazenamento para aplicações multiusuárias grandes onde a coordenação entre muitos usuários é necessária. Até mesmo usuários individuais os acham conveniente, entretanto, e muitos programas de correio eletrônico e os organizadores pessoais estão baseado em tecnologia de banco de dados standard.

#### **5. Aplicativo de Banco de Dados**

Um Aplicativo de Banco de dados é um tipo de software exclusivo para gerenciar um banco de dados. Aplicativos de banco de dados abrangem uma vasta variedade de necessidades e objetivos, de pequenas ferramentas como uma agenda, até complexos sistemas empresariais para desempenhar tarefas como a contabilidade.

O termo "Aplicativo de Banco de dados" usualmente se refere a softwares que oferecem uma interface para o banco de dados. O software que gerencia os dados é geralmente chamado de sistema gerenciador de banco de dados (SGBD) ou (se for embarcado) de "database engine".

Exemplos de aplicativos de banco de dados são Firebird, Microsoft Access, dBASE, FileMaker, MySQL, PostgreSQL, Microsoft SQL Server, Informix e Oracle.

#### **6. Descrição do Banco de dados Acadêmico**

O banco de dados descrito representado abaixo é de um pequeno sistema escolar, onde existem basicamente dois componentes que são os alunos matriculados na instituição, bem como as notas obtidas por eles em todas avaliações realizadas durante um período escolar.

Uma vez definido o escopo da aplicação, ou seja, o seu propósito, o próximo passo é identificar os elementos que a constituem, e por consequência definir todos os dados relevantes para cada item existente. Esses elementos são comumente chamados de entidades, e que por questões de facilidade, são representadas por tabelas.

Matrícula	Nome	Série	Turma	Telefone	Data de Nascimento
1	José da Silva	Oitava	1	(31) 3666-9090	05-10-192
2	Ana Maria	Sétima	1	(21) 1234-4567	12-11-1983
3	Paulo Simon	Quinta	1	(31) 8890-7654	17-04-1984
4	Carla Beatriz	Sexta	1	(45) 9946-8989	30-07-1979
5	Ana Paula	Oitava	2	(62) 7878-0909	22-01-1980
6	Joana Prado	Quinta	2	(35) 8878-0099	06-05-1986

Tabela 1: Definição de ENTIDADE alunos

Matrícula	Data do Teste	Ponto
1	25-03-2004	5.5
2	25-03-2004	6
3	25-03-2004	8
4	25-03-2004	10
5	25-03-2004	7.8
6	25-03-2004	4.6
1	18-05-2004	7.2
2	18-05-2004	9.5
6	18-05-2004	10

Tabela 2: Definição de PONTUAÇÕES

A segunda entidade identificada no problema são as pontuações obtidas por cada aluno. Vale ressaltar que durante um período letivo poderão existir várias avaliações, geralmente em datas diferentes, onde deverão ser armazenados os resultados de todos os alunos para cada um destes testes. A tabela 2 descreve a entidade pontuações, que serve para o propósito exposto anteriormente.

Cada entidade é representada por uma tabela, sendo que neste universo de discussão ou modelo, existem apenas duas tabelas e um relacionamento entre elas, já que cada entidade aluno está ligada à entidade pontuação. Em aplicações mais complexas, poderão existir inúmeras tabelas e relacionamentos de forma a permitir a representação do problema abordado.

Atributos definem uma entidade

Cada entidade, no exemplo alunos e pontuações, é representada por uma tabela, que por sua vez são constituídas de linhas e colunas. Cada coluna representa um fragmento de dado e o conjunto de todas as colunas constitui a entidade propriamente dita. No contexto de banco de dados cada coluna é chamada de atributo e uma entidade será formada por um ou vários atributos.

Um atributo define uma característica da entidade, no exemplo aluno é constituído de seis atributos, que são o número de matrícula, nome, a série que está cursando, a sua turma, o seu telefone residencial e a data de nascimento. O atributo matrícula possui um papel importante no modelo servindo como identificador único para cada aluno. Em um banco de dados caso ocorram registros com valores idênticos não será possível determinar um contexto que os identifiquem unicamente. Por isso deve existir uma chave ou atributo que identifique unicamente cada registro. Ao observar a Tabela 1, percebe-se que não há dois alunos cadastrados com o mesmo número de matrícula. Portanto, este é o atributo chave da entidade, utilizado para a pesquisa de um registro nesta tabela.

A entidade pontuações necessita identificar o aluno, a data da avaliação e a pontuação atingida pelo aluno. Neste caso, como cada aluno é identificado unicamente pela sua matrícula, este atributo será inserido na tabela de pontuações para permitir associar o aluno à nota registrada, conforme visto na Tabela 2.

Percebe-se que cada atributo possui um conjunto de valores válidos e aceitáveis, que é definido como domínio do atributo. Todas informações vistas na tabela são textuais, isto é, sequências de letras e números, mas é notório que o conjunto de dados contido em cada coluna é diferente umas das outras. No caso de matrícula do aluno, o domínio dos dados é o conjunto dos números inteiros positivos, já que para cada aluno é atribuído um código numérico que denota a ordem em que este foi matriculado na escola. Ou seja, o texto contido nesta coluna é formado por uma combinação de números, portanto não existem letras.

## 7. Modelagem de dados: modelo conceitual, modelo lógico e físico

A modelagem de dados é uma técnica usada para a especificação das regras de negócios e as estruturas de dados de um banco de dados. Ela faz parte do ciclo de desenvolvimento de um sistema de informação e é de vital importância para o bom resultado do projeto. Modelar dados consiste em desenhar o sistema de informações, concentrando-se nas entidades lógicas e nas dependências lógicas entre essas entidades.

Modelagem de dados ou modelagem de banco de dados envolve uma série de aplicações teóricas e práticas, visando construir um modelo de dados consistente, não redundante e perfeitamente aplicável em qualquer SGBD moderno.

A modelagem de dados está dividida em:

### 7.1 Modelo conceitual

A modelagem conceitual baseia-se no mais alto nível e deve ser usada para envolver o cliente, pois o foco aqui é discutir os aspectos do negócio do cliente e não da tecnologia. Os exemplos de modelagem de dados vistos pelo modelo conceitual são mais fáceis de compreender, já que não há limitações ou aplicação de tecnologia específica.

A arquitetura desta abstração se dá em três níveis. O mais externo, o nível de visões do usuário, descreve partes do banco que serão visualizadas pelos usuários. No nível intermediário, tem-se o nível conceitual (ou lógico), que descreve quais os dados estão armazenados e seus relacionamentos. Finalmente, no nível mais baixo, está o nível físico, descrevendo a forma como os dados estão realmente armazenados.

O termo negócio refere-se ao problema em questão que se deseja realizar uma modelagem. A intenção de armazenar informações de alunos numa escola, por exemplo, sugere que o negócio seja "Registro acadêmico". Num outro exemplo, o negócio "Instituição financeira" poderá ser modelado tendo dados de correntistas e saldos.

Razões para a criação do modelo conceitual:

- Descreve exatamente as informações necessárias ao negócio. Para a modelagem, todas as regras do negócio deverão ser conhecidas e, cabe ao projetista, traduzi-las em informações relevantes ao banco;
- Facilita a discussão, seja entre o projetista e o usuário ou entre o projetista e sua equipe de trabalho;
- Ajuda a prevenir erros do futuro sistema;
- Uma forma de documentar o sistema ideal. Um sistema é ideal quando todas suas informações estão modeladas de acordo com certas condições;
- É a base para o projeto físico do banco de dados.

A abordagem utilizada aqui será a representação de dados no modelo relacional, utilizando-se, para tal, o Modelo de Entidade-Relacionamento (MER). O MER é um modelo baseado na percepção do mundo real, que consiste em um conjunto de objetos básicos chamados de entidades e nos relacionamentos entre esses objetos.

Foi proposto por Peter Chen, em 1976, como uma ferramenta de projeto de banco de dados. O MER apresenta como contribuições um maior grau de independência de dados que os modelos convencionais (de redes e hierárquico) e uma unificação de representação destes modelos, através do formalismo gráfico do Diagrama de Entidade-Relacionamento (DER).

São características do MER:

- Modela regras de negócio e não a implementação. A modelagem é dos dados requeridos para o negócio, baseado nas funcionalidades do sistema atual ou a ser desenvolvido. Para modelar um negócio, é necessário conhecer em detalhes sobre do que se trata.
- Possui uma sintaxe robusta, bem definida;
- Técnica amplamente difundida e utilizada. Atualmente, a maioria dos bancos de dados disponíveis no mercado utiliza a abordagem relacional como modelo de dados;
- Diagramas fáceis de entender e alterar.

Os objetivos de uma modelagem entidade-relacionamento são:

- Obter todas as informações requeridas sobre o negócio antes de sua implementação, tornando claras suas dependências;
- Dentro do possível, uma informação aparecer apenas uma vez no banco de dados. Uma modelagem que prevê o armazenamento de uma mesma informação em dois locais

diferentes, deixa o sistema vulnerável quanto a possibilidade destas informações não serem as mesmas. No caso de uma inconsistência dos dados, qual delas deverá ser descartada?

- Facilitar o projeto do banco de dados, possibilitando a especificação de sua estrutura lógica.

## 7.2 Entidade e Instância

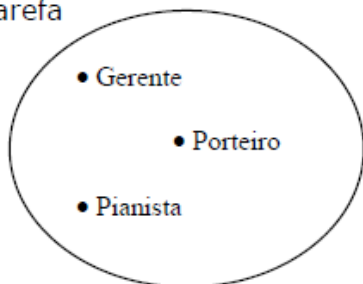
Num MER, uma entidade é um objeto, real ou abstrato, de relevância para o negócio. É uma categoria de ideias que são importantes ao negócio, as quais devem ser traduzidas em informação. Dois importantes aspectos de uma entidade é que possui instâncias e estas instâncias também são de interesse ao negócio.

Pode-se considerar que uma instância identifica individualmente uma entidade. O quadro abaixo mostra alguns exemplos de entidades e instâncias:

Entidade	Instância
Pessoa	João, José, Antônio
Produto	Prego 12x12, File de Peixe Merluza
Tipo de produto	Plástico, papel, madeira
Tarefa	Professor, pianista, gerente
Versão do documento	1.2, 10.5

Observa-se que uma entidade possui várias instâncias e que cada instância está relacionada a uma entidade. Uma entidade representa um conjunto de instâncias que interessam ao negócio. Segue o exemplo a seguir:

Tarefa



Em termos físicos, uma entidade será uma *tabela* do BD e cada instância será uma *linha* (ou *registro* ou *tupla*). Exemplo:

Tabela Tarefa		Entidade
Código	Descrição	
405	Gerente	Instâncias
564	Porteiro	
321	Pianista	

## 7.3 Atributo e Domínio

Um atributo também representa algo significativo ao negócio. Um atributo é uma propriedade de uma entidade. É uma porção de informação que descreve, quantifica, qualifica, classifica e especifica uma entidade. Normalmente, uma entidade possui vários atributos. Interessa, em termos de modelagem conceitual, que estes atributos representem informações relevantes ao negócio. Atributos possuem valores (um número, um caractere, uma data, uma imagem, um som, etc), chamados de tipos de dados ou formato. Para um atributo particular, todas suas instâncias possuem os mesmos formatos. O quadro abaixo apresenta exemplos de entidades, instâncias e atributos.

Entidade	Instância	Atributo
Empregado	João, Antônio	Nome, idade, tamanho pé, dependentes, cidade
Carro	Escort, Gol	Cor, preço, modelo
Tarefa	Gerente	Código, Depto, Valor/hora, descrição

Algumas questões:

- O atributo Idade, da entidade Empregado, não parece ser uma boa escolha. O ideal seria Data Nascimento, ficando o cálculo da idade quando necessário. O armazenamento da informação idade é de difícil, senão impossível, atualização;
- O atributo Tamanho pé, de Empregado, dependerá das regras de negócio. Imaginando que a finalidade de entidade Empregado seja a de armazenar dados sobre funcionários numa



empresa que forneça uniforme de trabalho, o atributo é coerente. Se a empresa não possui esta política, ele é desnecessário;

- Uma importante decisão precisa ser tomada em relação a armazenar uma informação como um atributo ou uma entidade. O atributo Cidade, de Empregado, terá a cidade na qual o empregado reside. Se Cidade fosse uma entidade, alguns possíveis atributos seriam População, Área e Data Fundação. Aqui, novamente a escolha passa pelas regras de negócio. Normalmente, uma informação será um atributo se for de natureza atômica e será uma entidade quando possuir informações que possam (ou necessitem) ser relacionadas a outras entidades.

Em termos físicos, os atributos serão as *colunas* de uma tabela do BD. Exemplo:

Tabela Empregado

Nome	Idade
João	32
Antônio	25

Colunas

**Atributo monovalorado ou atômico:** assume um único valor, num certo instante de tempo, para cada instância. Exemplo: Nome de Empregado

**Atributo composto:** formado por um ou mais subatributos. Exemplo: Cidade, de Empregado. Cidade pode ser composto pelo nome da cidade e o estado.

**Atributo multivalorado:** assume diversos valores. Seu nome, normalmente, é no plural. Exemplo: Dependentes de Empregado.

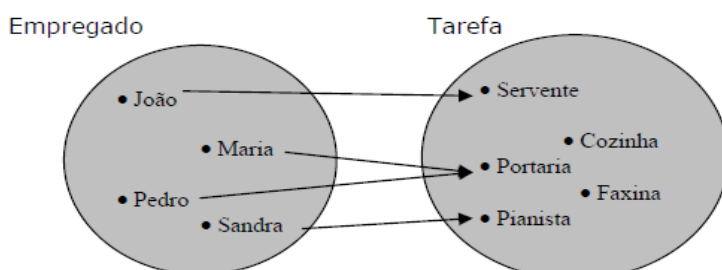
**Atributo determinante:** identifica cada entidade como única. Exemplo: Código, em Tarefa.

**Domínio de um atributo:** conjunto de valores possíveis para um atributo. Exemplo: Idade deve estar ente 18 e 60, Estado deve ser "RS", "RJ", "SP", etc.

## 7.4 Relacionamentos

É uma estrutura que indica uma associação entre duas ou mais entidades. Alguns exemplos:

Entidade	Relacionamento	Entidade
Empregado Tarefa	Exerce É exercida por	Tarefa Empregado
Produto Tipo Produto	É classificado por um É uma classificação de	Tipo Produto Produto
Pessoa Reserva	Faz É feita por	Reserva Pessoa



- Todos os empregados exercem tarefas;
- Nenhum empregado tem mais de uma tarefa;
- Nem todas as tarefas são exercidas por empregados;
- Algumas tarefas são exercidas por mais de um empregado.

O exemplo anterior serviu para que sejam apresentadas algumas questões referentes aos relacionamentos entre duas entidades. São elas:


1. Todo empregado DEVE (ou PODE) ter uma tarefa?

2. Toda tarefa DEVE (ou PODE) ser exercida por um empregado?
3. Um empregado pode exercer UMA ou MAIS tarefas? Ou então: Uma tarefa pode ser exercida por UM ou MAIS empregados?

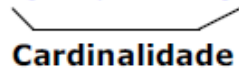
Para as duas primeiras, a análise do relacionamento é de existência (obrigatório => DEVE ou opcional => PODE), enquanto que a terceira trata da cardinalidade do relacionamento (um-para-um, um-para-vários, vários-para-um ou vários-para-vários).

Um relacionamento entre duas entidades E1 e E2 deve ser lido da seguinte forma:

Cada E1 {deve / pode} nome\_do\_relacionamento {um / vários} E2



**Existência**



**Cardinalidade**

### 7.5 Existência (Obrigatório x Opcional)

Um relacionamento pode ser obrigatório ou não. Se ele existe, diz-se que é obrigatório. Se não existe, é opcional. A existência ou não de um relacionamento é identificada pelas palavras DEVE e PODE, por exemplo. Sem levar em conta a cardinalidade, as possíveis combinações da relação de existência entre as entidades Empregado e Tarefa são (observar que a coluna "Pode/Deve" é determinante da existência do relacionamento):

E1	Pode Deve	Nome do relacionamento	E2	Obrigatório Opcional
Cada Empregado	Deve	Exercer	Uma/várias Tarefas	Obrigatório
Cada Tarefa	Deve	Ser exercida por	Um/vários Empregados	Obrigatório
Cada Empregado	Podem	Exercer	Uma/várias Tarefas	Opcional
Cada Tarefa	Podem	Ser exercida por	Um/vários Empregados	Opcional

### 7.6 Exercícios para fixação:

- 1-) Crie uma entidade, diferentes das abordadas em aula, e defina pelo menos 5 atributos
- 2-) Crie duas entidades, diferentes das abordadas em aula, defina pelo menos 5 atributos para cada, e ao fim relacione-as.

**Observação:** Este exercício deverá ser realizado em grupos e verificaremos em aula.

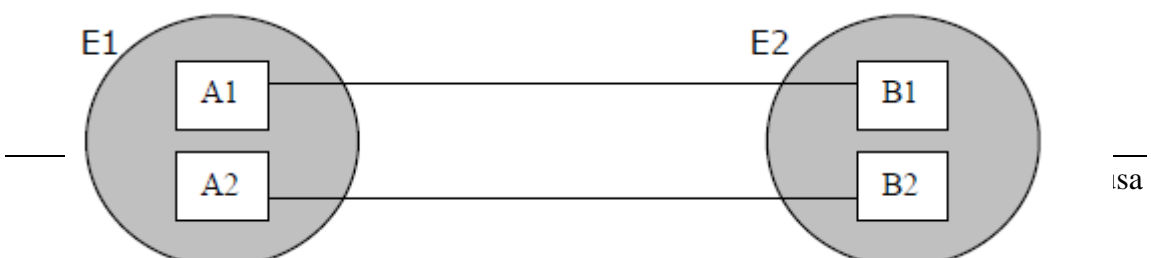
### Cardinalidade

É o número de entidades que podem estar associadas. Os relacionamentos binários podem ser: um-para-um (1:1), um-para-vários (1:N) ou vários-para-vários (N:N).

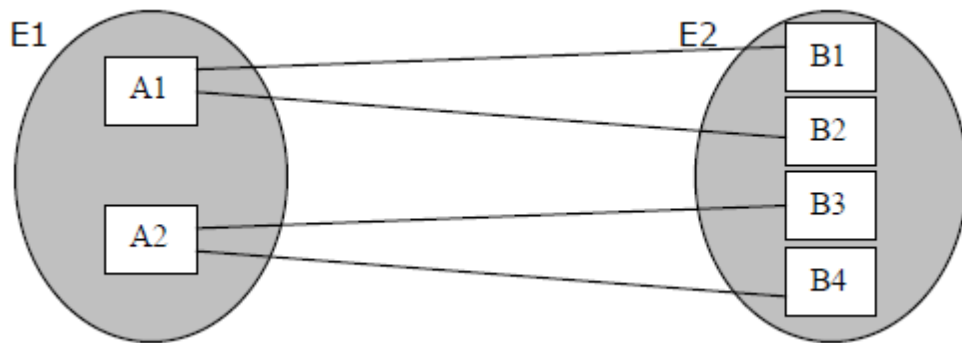
Sem levar em conta a existência, as possíveis combinações para a cardinalidade do relacionamento entre as entidades Empregado e Tarefa são (observar que a coluna de E2 é determinante para a cardinalidade):

E1	Pode Deve	Nome do relacionamento	E2	Cardinalidade
Cada Empregado	Deve/pode	Exercer	Uma Tarefa	Um-para-um 1:1
Cada Tarefa	Deve/pode	Ser exercida por	Um Empregado	
Cada Empregado	Deve/pode	Exercer	Uma Tarefa	Um-para-vários 1:N
Cada Tarefa	Deve/pode	Ser exercida por	Vários Empregados	
Cada Empregado	Deve/pode	Exercer	Várias Tarefas	Vários-para-vários N:N
Cada Tarefa	Deve/pode	Ser exercida por	Vários Empregados	

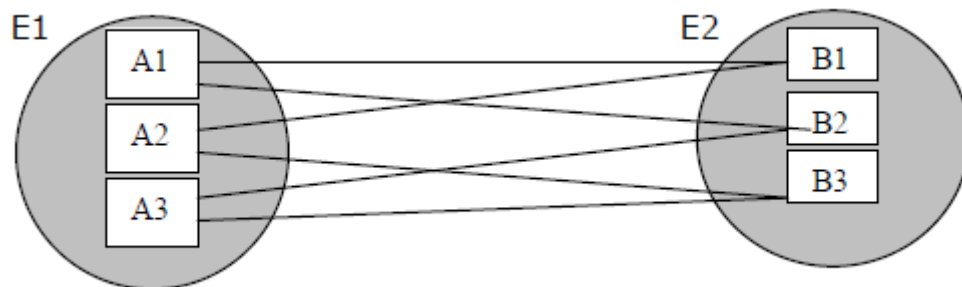
A cardinalidade um-para-um (1:1) ocorre quando uma instância de E1 está associada no máximo a uma instância de E2 e uma instância de E2 está associada no máximo a uma instância de E1.



A cardinalidade um-para-vários (1:N) ocorre quando uma instância de E1 está associada a qualquer número de instâncias de E2, enquanto que uma instância de E2 está associada no máximo a uma instância de E1.



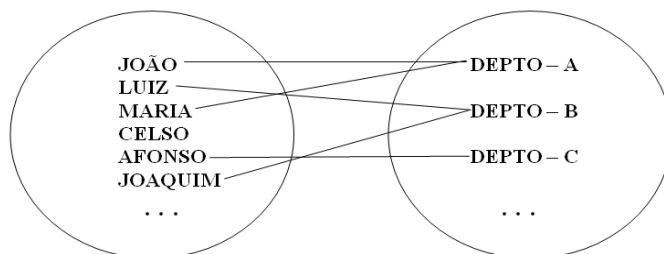
está associada a qualquer número de instâncias de E2 e uma instância de E2 está associada a qualquer número de instâncias de E1.



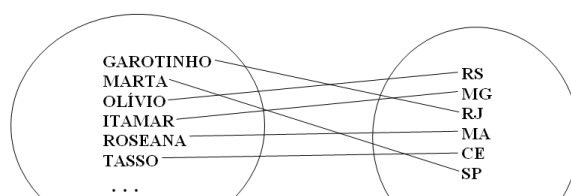
### 7.7 Exercícios para fixação 2:

Informe a cardinalidade para as situações a seguir

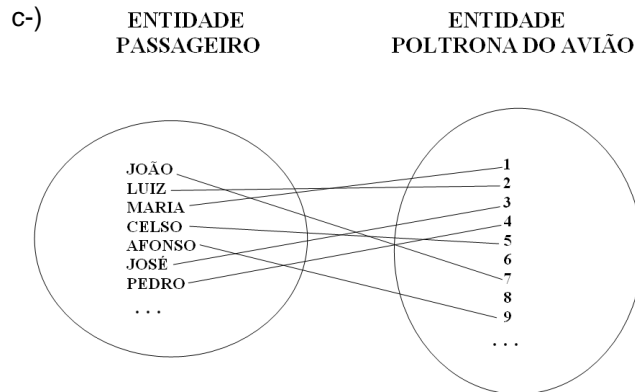
- a)
- | ENTIDADE  | ENTIDADE     |
|-----------|--------------|
| EMPREGADO | DEPARTAMENTO |



- b)
- | ENTIDADE   | ENTIDADE |
|------------|----------|
| GOVERNADOR | ESTADO   |







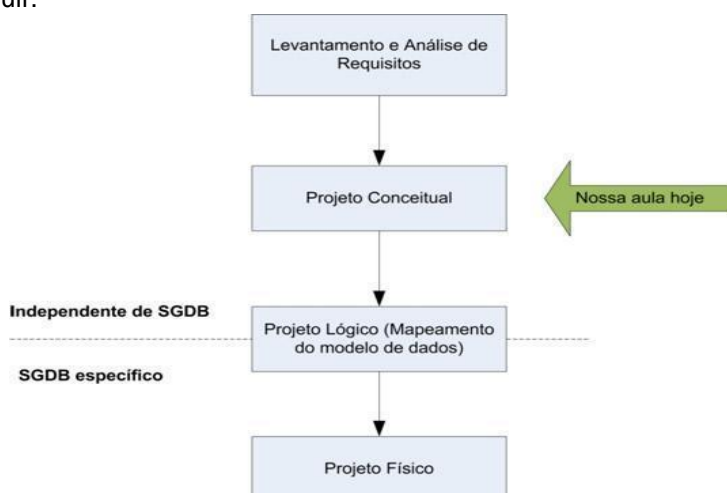
## 8. Modelo de Entidade e Relacionamento

### 8.1 História

O modelo de entidades e relacionamentos é um modelo conceitual onde descrevemos o nosso banco de dados. Representamos esse modelo por um diagrama de Entidade e Relacionamento (ER).

A estrutura lógica geral de um banco de dados pode ser expressa graficamente por um Diagrama de Entidade-Relacionamento. É uma representação gráfica do modelo ou parte do modelo. De acordo com o grau de complexidade do negócio ou o nível de detalhamento impresso pelo projetista do banco, um modelo pode ser representado por vários diagramas.

Mas, antes de falar sobre os conceitos, vamos conhecer a história do Modelo de Entidade e Relacionamento. Tudo começou quando o Dr. Peter Chen em 1976 propôs o modelo Entidade-Relacionamento (ER) para projetos de banco de dados. Isso deu uma nova e importante percepção dos conceitos de modelos de dados. O modelo ER proposto pelo Dr. Peter possibilitava ao projetista concentrar-se apenas na utilização dos dados sem se preocupar com estrutura lógica de tabelas. Por esse motivo, o modelo ER é utilizado pelo projeto conceitual para modelar os conceitos do banco de dados de forma independente de SGDB como vemos na figura a seguir.



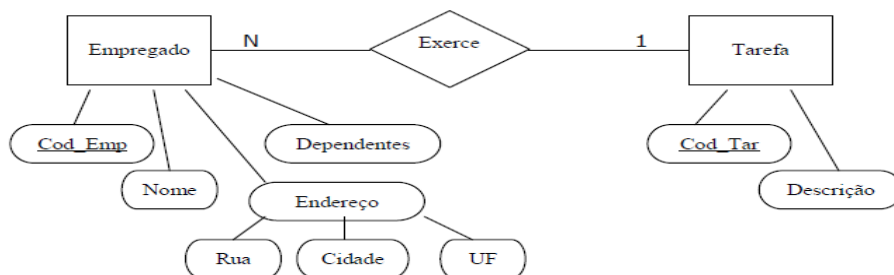
Professor Doutor Peter Chen

Cientista da computação americano  
 Página oficial: <http://www.csc.lsu.edu/~chen/>

Um DER utiliza-se de um número de elementos gráficos. Infelizmente, não existe uma padronização na representação do diagrama. Serão apresentados dois padrões: um deles segundo Peter Chen e o outro uma pequena variação de um desenvolvido pela Oracle.

Os componentes do Diagrama de Entidade-Relacionamento, de acordo com Peter Chen são:

- Retângulos: representam as entidades
- Elipses: representam os atributos
- Losangos: representam os relacionamentos
- Linhas: ligam atributos a entidades e entidades a relacionamentos.

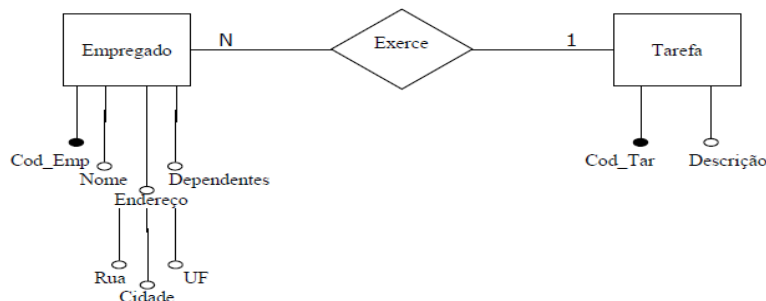


EXEMPLO DE DER SEGUNDO PETER CHEN

Observa-se o seguinte:

- São duas as entidades: Empregado e Tarefa
- Atributos da entidade Empregado:
  - Cod\_Emp (determinante - está sublinhado)
  - Nome (monovalorado)
  - Dependentes (multivalorado)
  - Endereço (composto)
  - Rua (monovalorado)
  - Cidade (monovalorado)
  - UF (monovalorado)
- Atributos da entidade Tarefa:
  - Cod\_Tar (determinante - está sublinhado)
  - Descrição (monovalorado)
- O relacionamento entre Empregado e Tarefa possui cardinalidade 1:n

Uma variação do diagrama anterior é apresentada a seguir. Nota-se que os atributos determinantes não mais são sublinhados, pois possuem símbolos diferentes dos demais, e adotaremos este modelo para nossas aulas.



#### 0.2 Exemplo para fixação na prática em aula

##### Variação 1:

Banco de dados de um hospital onde possuem médicos. Criem os atributos determinantes para as duas entidades, mais os que julgarem necessário, por fim relacione as duas entidades e determine a cardinalidade. **Lembre-se que o hospital precisará pelo menos de um médico cadastrado.**

### Variação 2:

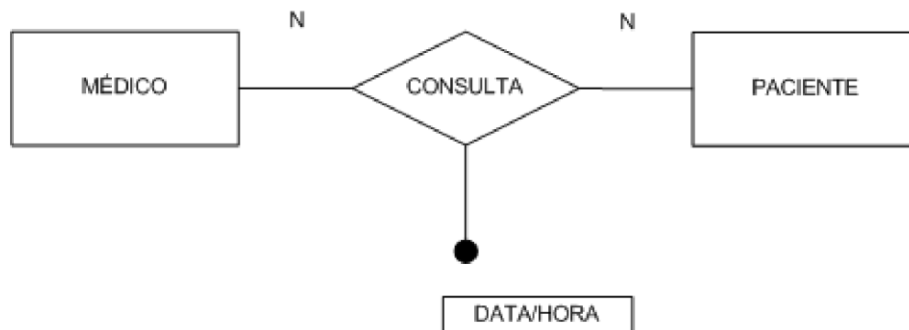
Levando em consideração o mesmo banco já criado, acrescente a entidade pacientes que serão atendidos pelos médicos, nesta nova entidade, crie os atributos necessários (determinantes e não determinantes) e por fim relacione com a entidade médico e determine a cardinalidade. **Lembre-se que um médico pode atender nenhum, um ou vários pacientes.**

### Variação 3:

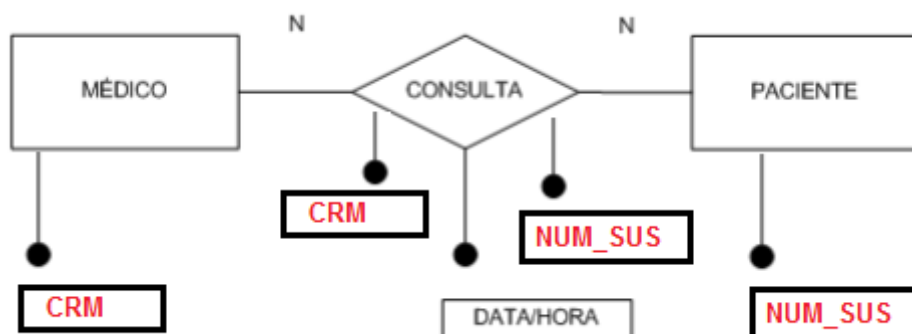
Ainda levando em consideração o mesmo banco já criado, acrescente a entidade medicamento, que serão fornecidos aos pacientes atendidos, nesta nova entidade, crie os atributos necessários (determinantes e não determinantes) e por fim relacione com a entidade paciente e determine a cardinalidade. **Lembre-se que um paciente pode ou não receber medicamentos.**

Há casos onde pode ser necessário relacionar ocorrências de mesmas entidades mais de uma vez. Por exemplo, em um modelo de consultas médicas, determinado paciente pode realizar consultas mais de uma vez com o mesmo médico.

Neste caso, podemos utilizar um atributo identificador no relacionamento (data/hora).



Isto também chamará de tabela de relacionamento, ou seja, precisamos ter os atributos determinantes tanto da entidade médico como da entidade paciente no relacionamento, para todo o relacionamento de N para N, o que denominamos em banco de dados como atributo estrangeiro ou chave estrangeira e assim o mesmo se tornará uma tabela. Veja:



Para exercitarmos este conceito vamos praticar as variações de acordo com o seguinte problema proposto:

### Variação 1:

Um banco de dados de uma loja, primeiramente quer cadastrar os seus Clientes e Fornecedores, crie os atributos (principalmente o(s) determinante(s)) necessários para cada Entidade.

### Variação 2:

Levando em consideração o banco já criado, agora iremos acrescentar os Produtos desta loja, é claro que estes produtos são vendidos a clientes, crie os atributos necessários para a Entidade, e os atributos do relacionamento, não se esqueçam dos atributos estrangeiros e da cardinalidade.

### Variação 3:

Neste mesmo banco implemente o relacionamento de Produto com Fornecedor, que chamará Fornece que deverá ser uma tabela, defina os atributos do relacionamento, e a cardinalidade.

### Variação 4:

Crie mais uma Entidade e seu respectivo Relacionamento.

### Usando uma ferramenta de apoio

A partir deste momento iremos utilizar uma ferramenta para ajudar a modelar melhor os dados.

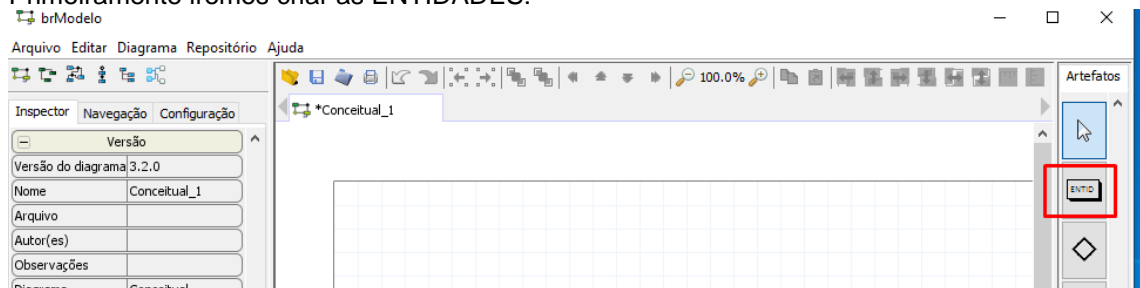
O brModelo (<http://www.sis4.com/brModelo/download.html>) desenvolvida em Java e é uma Ferramenta freeware voltada para ensino de modelagem em banco de dados relacional.

Para utilizarmos a ferramenta, exemplificaremos com um estudo de caso de um sistema bancário simplificado.

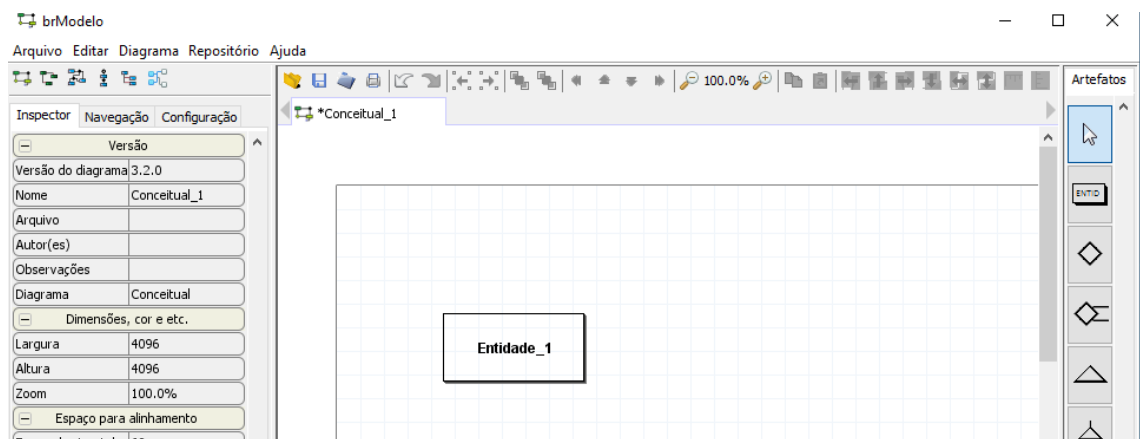
### 8.3 Sistema bancário simplificado

Um **banco** possui **nome** e **código**, este é composto de **agências** que possuem **número** e **endereço**. Cada agência possui **clientes** com **nome**, **CPF**, **telefone** e **endereço** que podem ter **contas** do tipo corrente ou especial. A **conta** deve ter **número** e **saldo**; a **conta do tipo especial** possui **limite especial**. Cada cliente pode indicar outro cliente ao banco e também adquirir um título de capitalização da agência, chamado **CAP**, que possui diversos **valores de investimento**, cada um possui um **código**. Os clientes que adquirirem o CAP ganham um brinde especial e também passam a ter um tratamento diferenciado através de um **gerente** especial (representado com **nome** e **CPF**) que a agência possui.

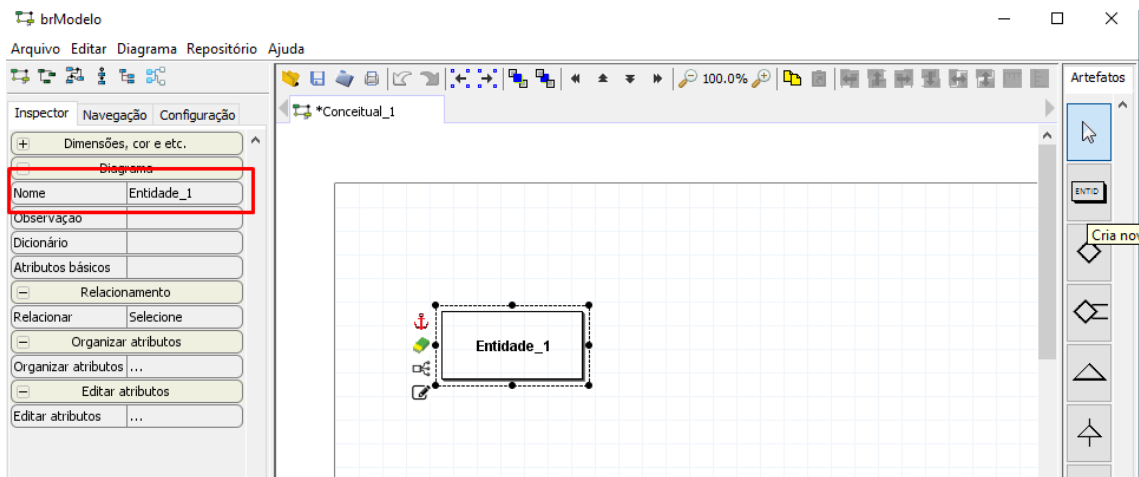
Primeiramente iremos criar as ENTIDADES:



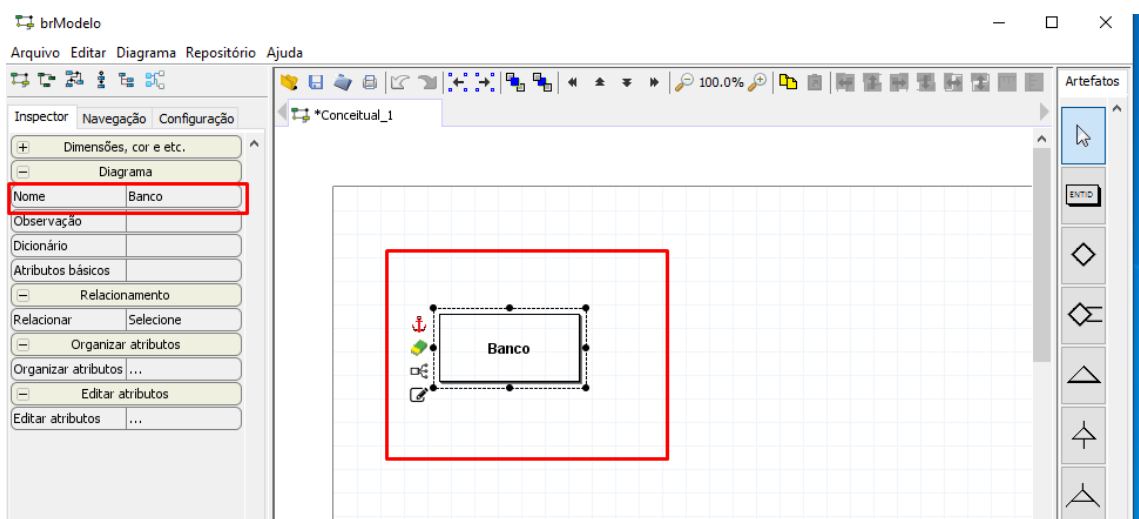
Clicar no ícone à esquerda **criar entidade** e clique na área branca.



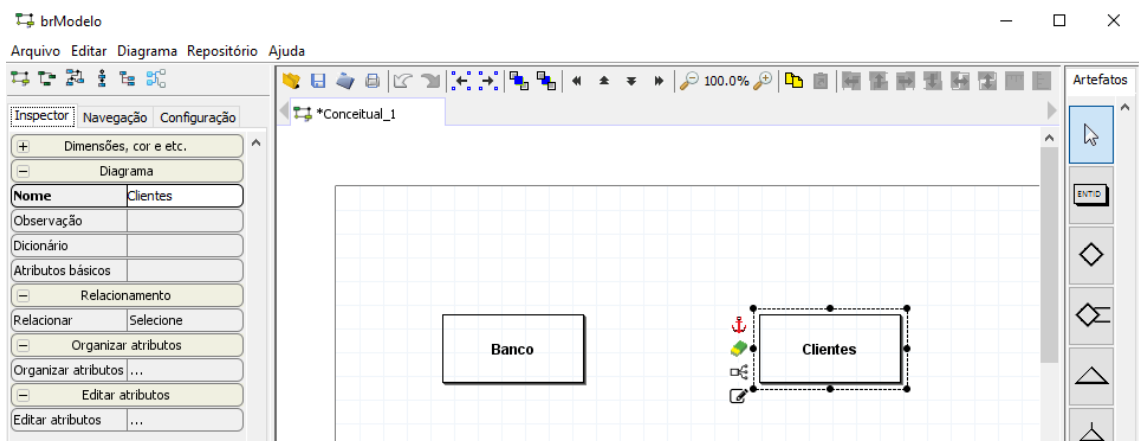
Clicar na Entidade criada para alterar seu nome.



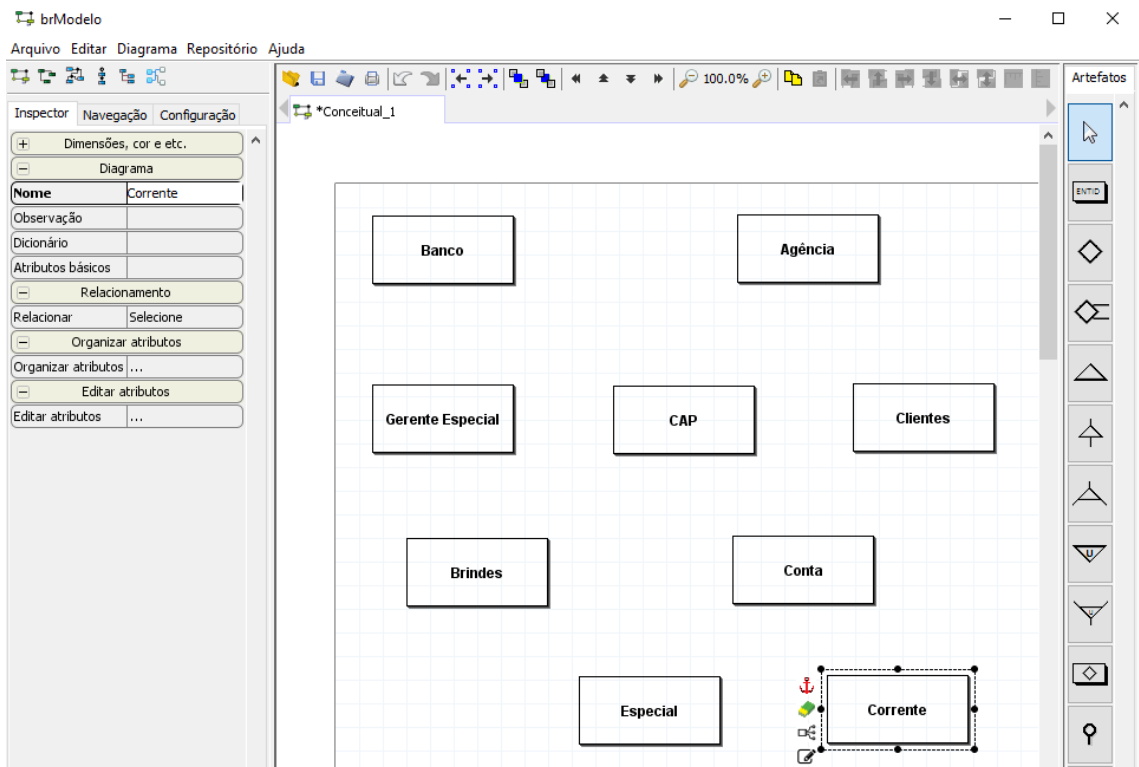
Na parte lateral esquerda veremos uma janela de propriedades, clique em cima de nome e altere o nome.



Criamos uma Entidade chamada Banco.

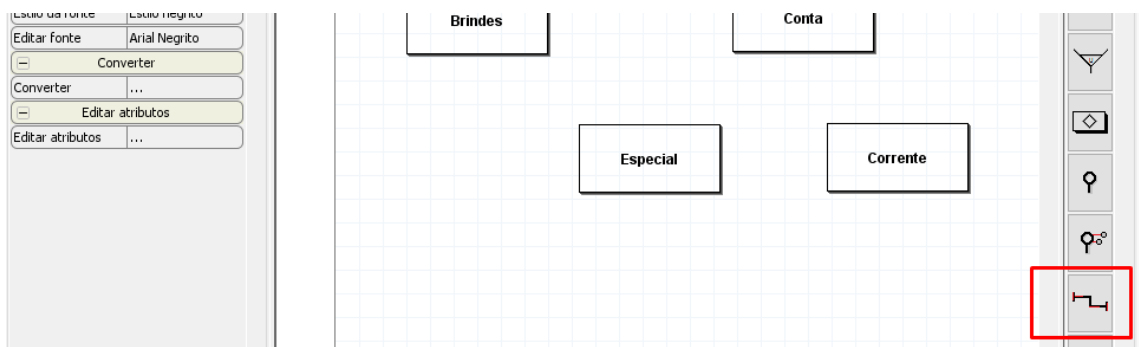


Criamos outra Entidade Clientes e vamos criar todas as outras existentes no estudo de caso.

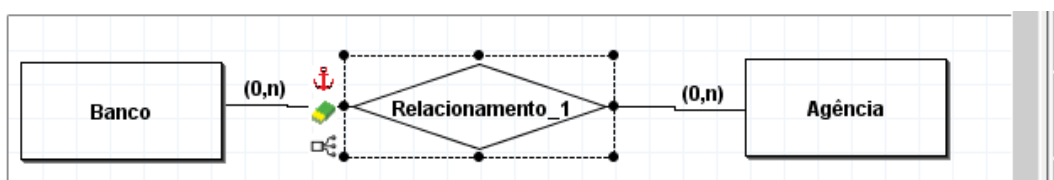


Faremos isto para todas as Entidades do nosso modelo.

**Após termos todas as Entidades, iremos criar os relacionamentos:**

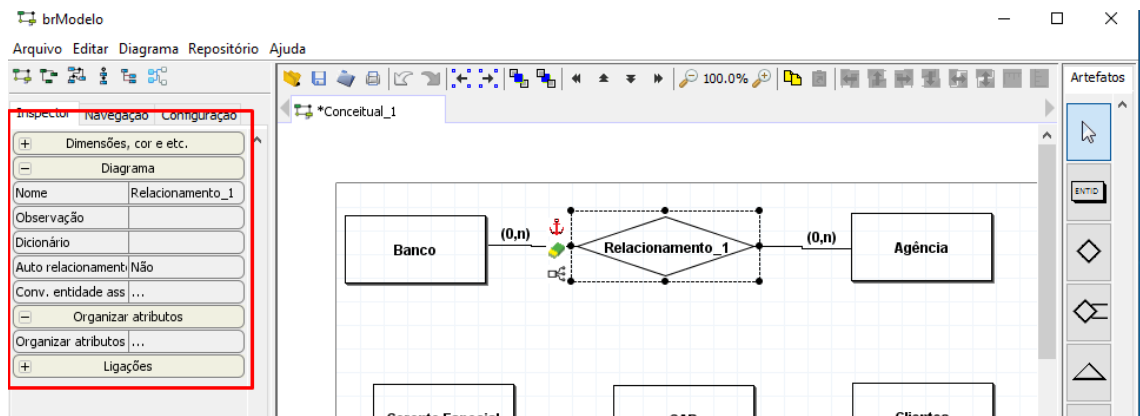


Criando o relacionamento com as Entidades Banco e Agência. Clique no menu à direita no ícone **Criar nova integração entre dois artefatos** e clicamos nas Entidades que se relacionam. Iremos clicar em Banco e Agência.

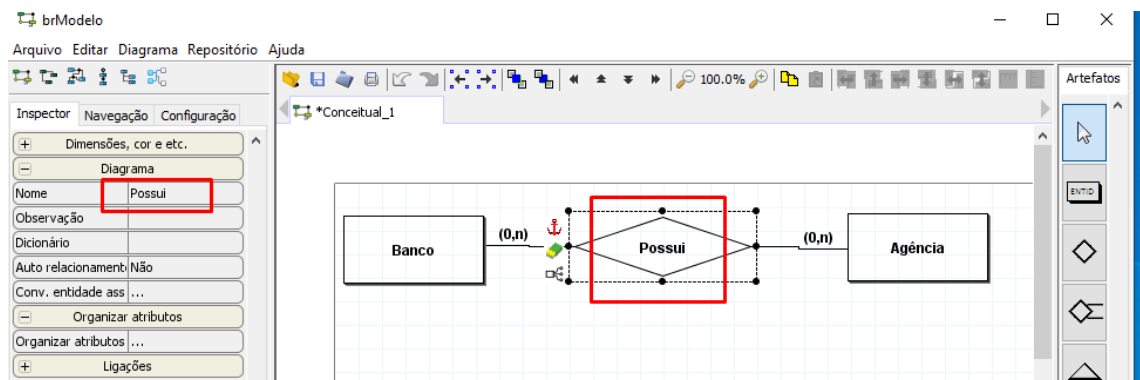


Clicamos sobre a relação para habilitar a janela de propriedade.

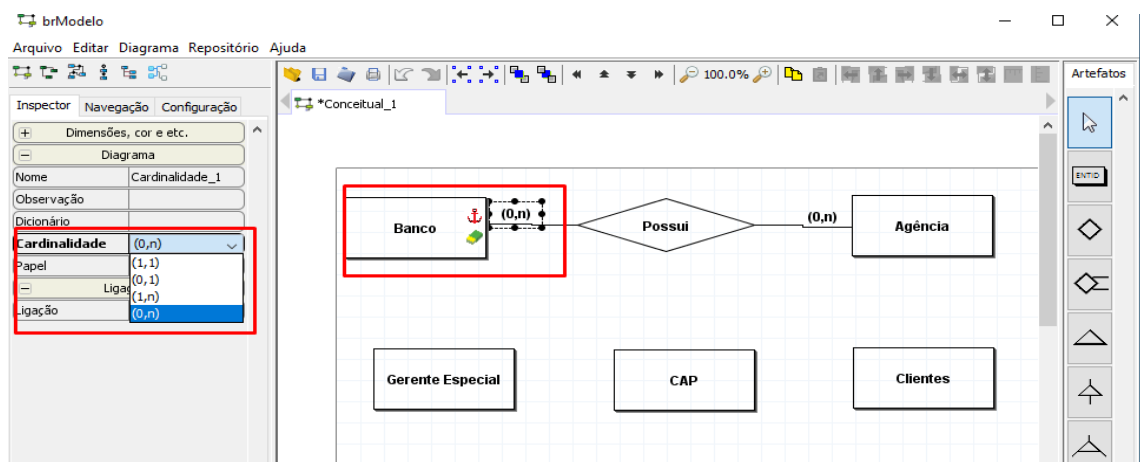




Clicamos em nome e alteramos o nome.

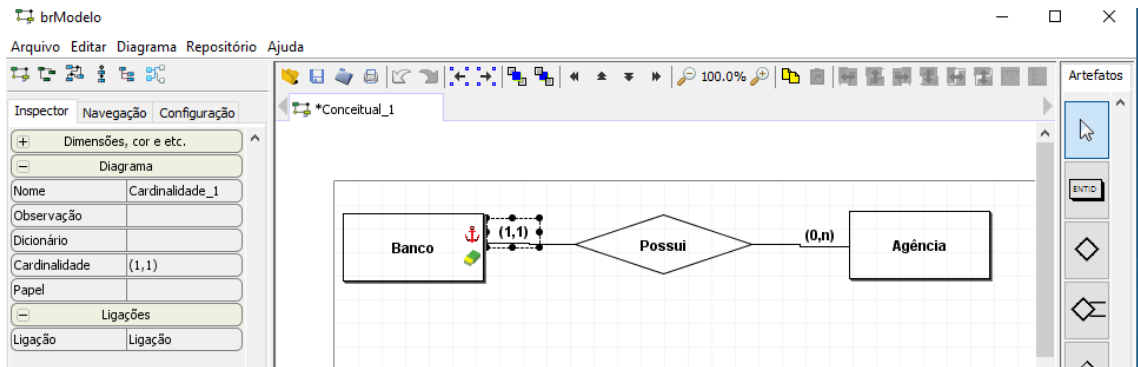


Colocamos o verbo Possui.

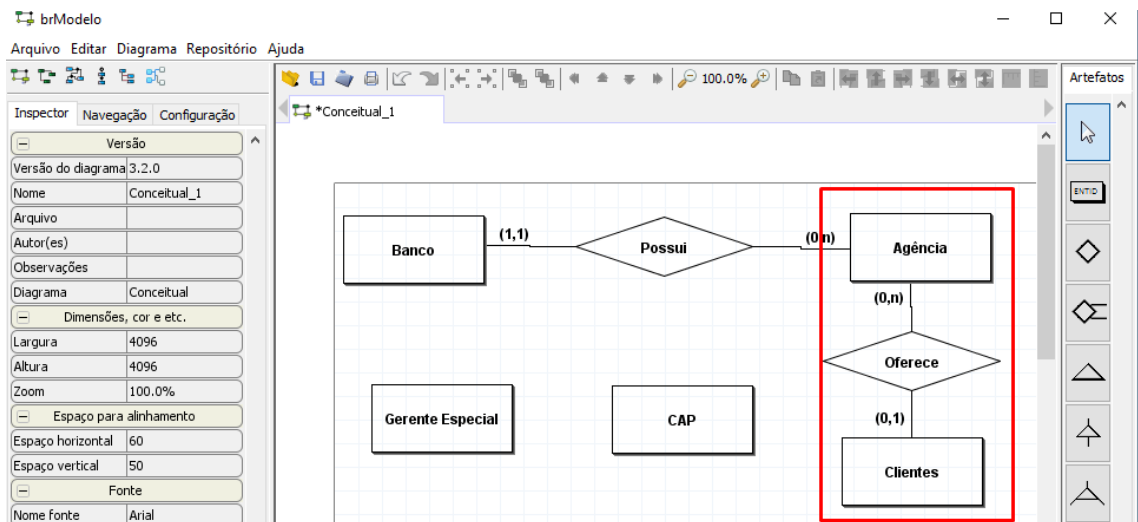


Agora iremos alterar as cardinalidade.

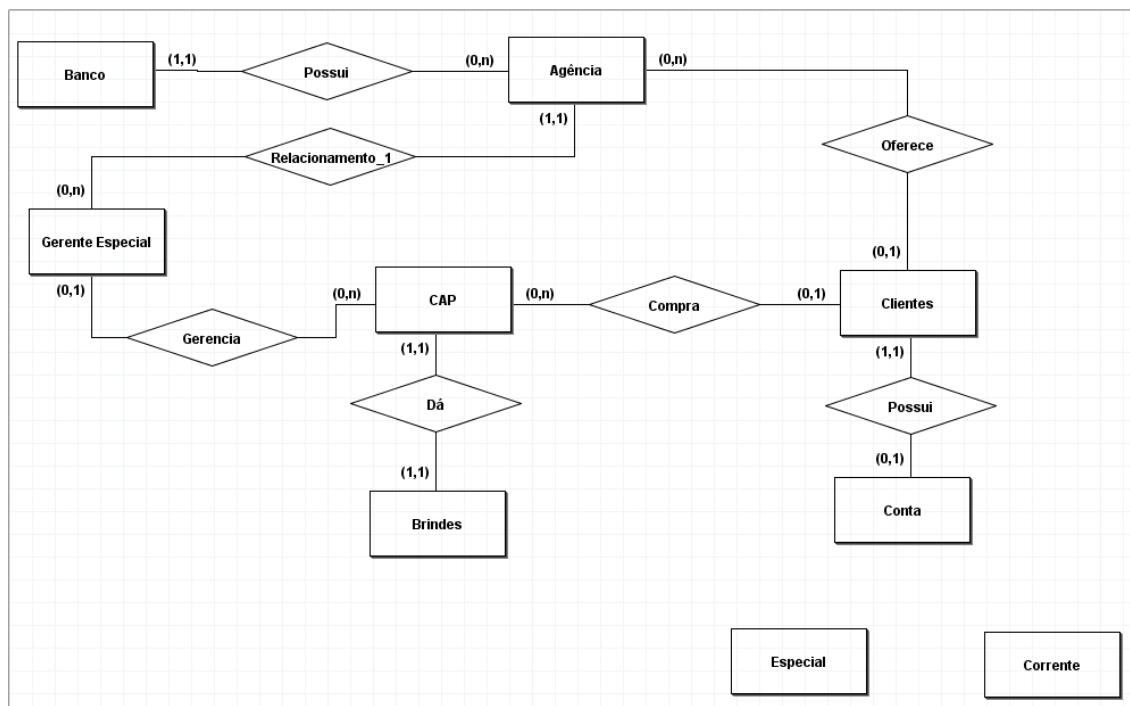
Na janela de propriedades à esquerda, clicamos em Cardinalidade e selecionamos a opção desejada.



Temos então o nosso primeiro relacionamento criado com a cardinalidade.



Outro relacionamento criado entre as Entidades Agência e Clientes.

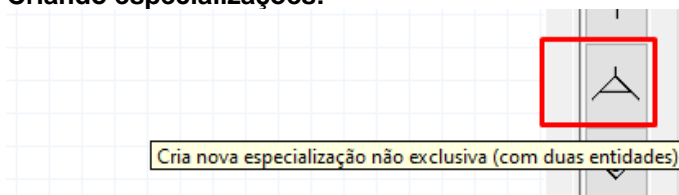


Criamos todos os relacionamentos.

### 8.3.1 Generalização e especialização

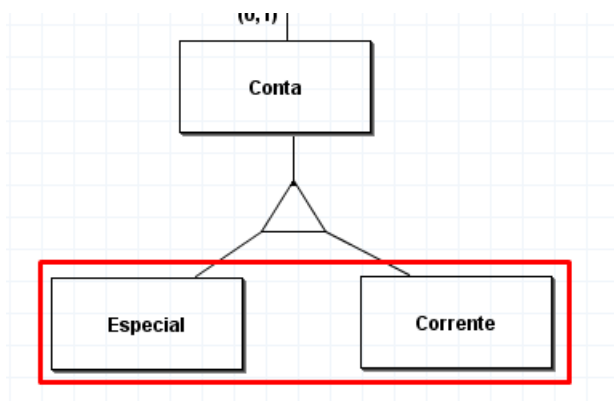
Existem casos em que um conjunto entidade pode ser dividido em categorias, cada qual com atributos específicos, é o caso que iremos utilizar na Entidade Conta que dará origem a duas categorias Especial e Corrente.

**Criando especializações:**

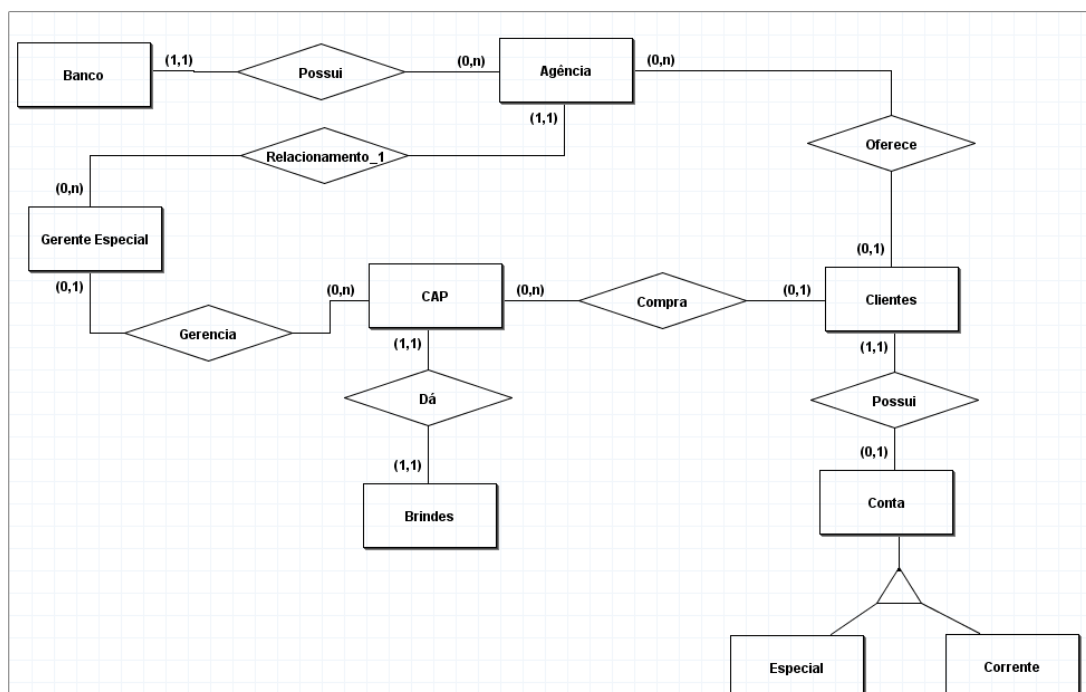


Clicando no ícone à esquerda em **Criar nova especialização não exclusiva (com duas entidades)** no menu de opções à direita.

Clique em cima de conta e temos criado Conta\_A e Conta\_B. Clique em cima de cada uma e altere o nome na janela de propriedades da direita.



Altere o nome e coloque Conta Especial e Conta Corrente. Criamos as Especializações.

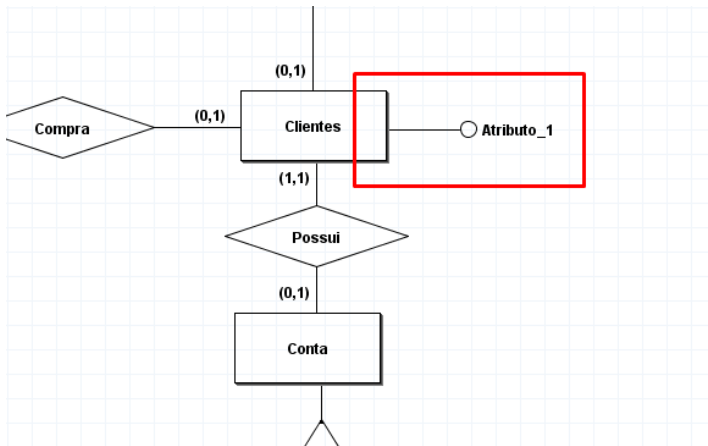


Modelo organizado.

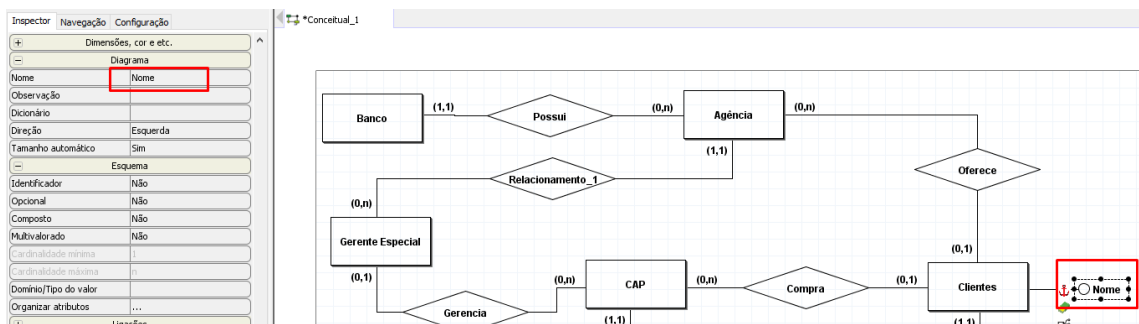
**Agora iremos colocar os atributos:**



Clicar no menu da direita no ícone **Cria novo atributo**.

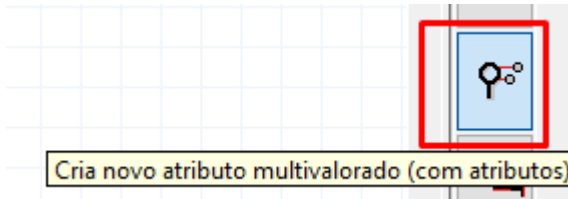


Clicar na Entidade Clientes.

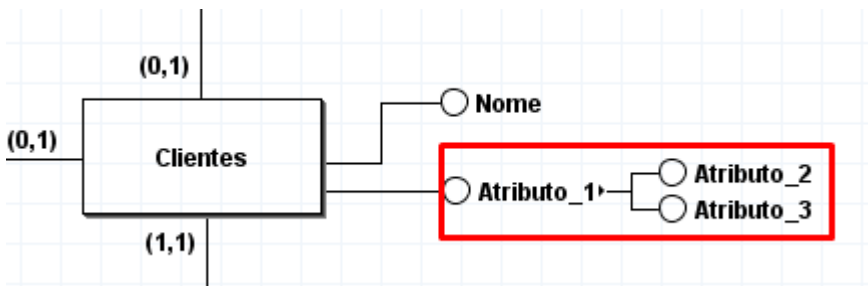


Clicar em cima do atributo e clicar na janela de propriedades do lado esquerdo em cima de nome e alterar o nome. Foi criado o atributo Nome.

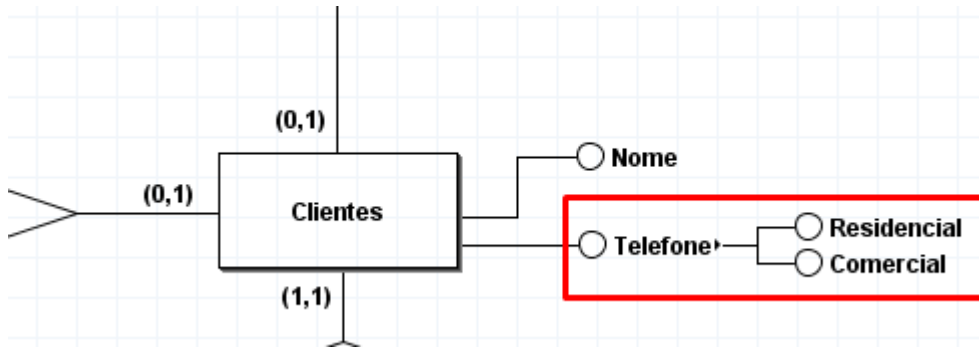
**Vamos criar o atributo telefone. Ele é um atributo multivalorado:**



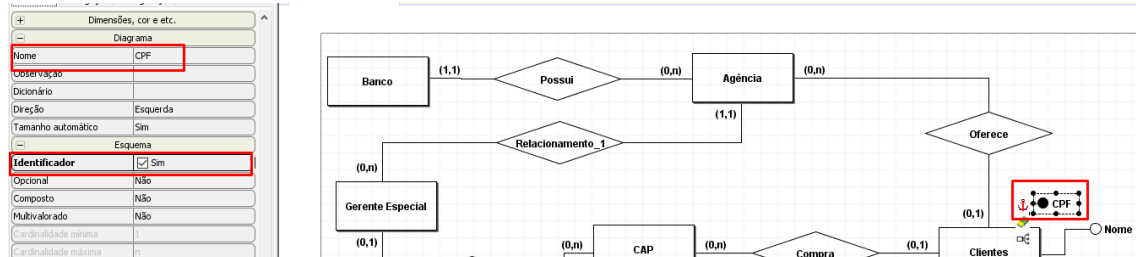
Clicar no menu à direita no ícone **Cria novo atributo multivalorado (com atributos)**.



Clicamos na Entidade Clientes e o atributo foi colocado. Clicamos novamente em cima do atributo para alterar o nome.



Vamos colocar o atributo CPF. Ele é um atributo identificador.

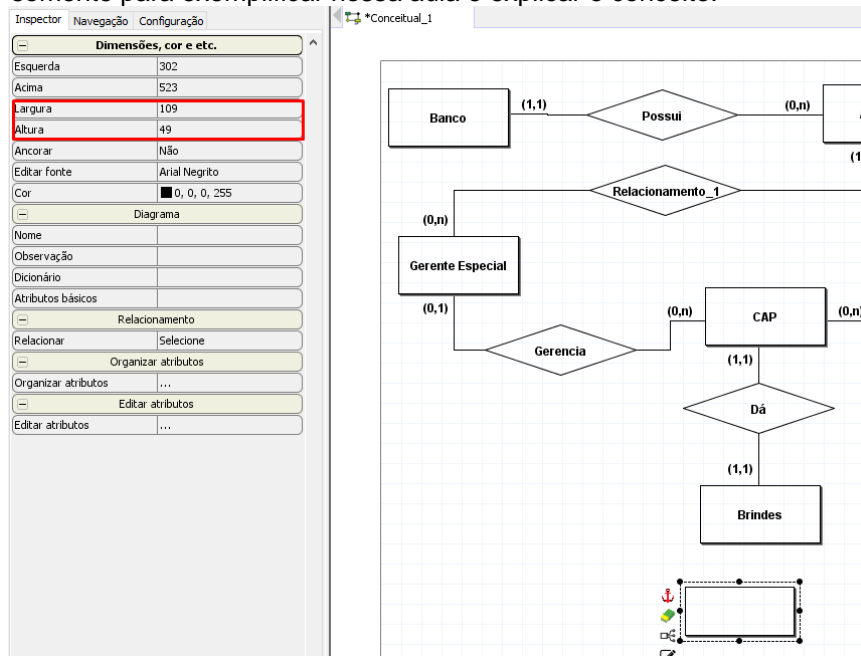


Clicar no menu da esquerda no check **Identificador**.

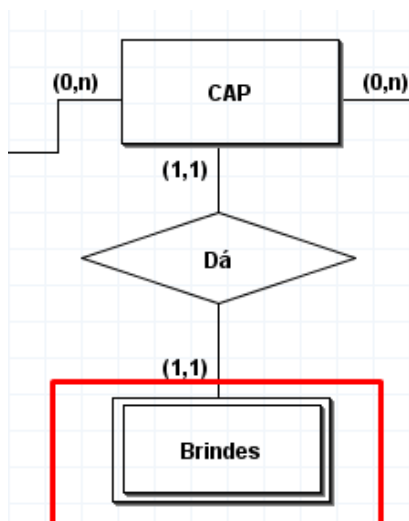
### 8.3.2 Criaremos agora uma entidade fraca:

Uma **entidade fraca** não possui sequer identidade própria, sendo sua chave primária composta pela chave estrangeira proveniente da entidade dona concatenada a um identificador de si própria (que pode repetir para diferentes instâncias da entidade dona).

A Entidade Brinde é uma entidade fraca, mas o BR Modelo não possui uma representação, então, criaremos outra Entidade e alteraremos sua altura e sua largura. Como disse em nossas aulas, a ideia é que a mesma tenha sua chave, pois como se trata de uma Entidade, mas vamos usar somente para exemplificar nossa aula e explicar o conceito.



Alteramos a Largura para 93 e Altura para 55



Colocamos a nova entidade em cima da entidade Brindes e na janela de propriedades do lado direito, no nome não escrevemos nada.

Temos agora a representação de uma entidade fraca.

### 9. História e Implementações do SQL

No início dos anos 70, o trabalho produtivo do colega de pesquisa da IBM Dr. E. F. Codd levou ao desenvolvimento de um produto modelo de dado relacional chamado SEQUEL ou Linguagem de Consulta em Inglês Estruturado (em inglês: *Strutucured English Query Language*). SEQUEL ultimamente se transformou em SQL ou Linguagem de Consulta Estruturada (em inglês: *Structured Query Language*).

IBM, junto com outros fornecedores de banco de dados relacionais, queria um método padronizado para acessar e manipular dados em um banco de dados relacional. Embora IBM tenha sido a primeira a desenvolver a teoria de banco de dados relacional, a Oracle foi a primeira a comercializar a tecnologia. Através do tempo, SQL se provou popular o suficiente no mercado de trabalho para atrair a atenção do *American National Standards Institute* (ANSI), que lançou padrões para SQL em 1986, 1989, 1992, 1999, 2003 e 2006. Este texto cobre o padrão ANSI 2003, pois o padrão 2006 lida com elementos do SQL fora do escopo dos comandos descritos neste livro. (Em essência, o padrão SQL2006 descreve como o XML deveria ser usado no SQL.)

Desde 1986, várias linguagens concorrentes permitiram programadores e desenvolvedores a acessarem e manipularem dados relacionais. No entanto, poucos foram fáceis de aprender ou aceitos universalmente como SQL. Programadores e administradores agora têm o benefício de serem capazes de aprender uma única linguagem que, com pequenos ajustes, é aplicável a uma vasta variedade de plataformas de banco de dados, aplicações e produtos, como exemplos temos:

- MySQL;
- Oracle Database 11g;
- PostgreSQL;
- SQL Server 2008 da Microsoft;
- Informix.

### 10. Breve História do MySQL

O MySQL surgiu a partir da necessidade da equipe que criou o SGBD, de utilizar algum mecanismo que permitisse a conexão de tabelas criadas na linguagem SQL para um determinado fim. A princípio, o grupo iria utilizar o mSQL(Mini SQL), mas logo perceberam que esta ferramenta não era rápida o suficiente para atender às necessidades do projeto. O jeito foi criar uma solução própria. Nascia o MySQL.

O MySQL foi criado por Michael Widenius na companhia suíça TcX. Por volta de 1979 Michael desenvolveu um banco de dados chamado UNIREG, "sendo rescritos em várias linguagens desde então". Em 1994, a empresa TcX começou o desenvolvimento de aplicações baseadas na Web, tendo como base o banco UNIREG, porém esse banco possuía muito "overhead" para obter sucesso em uma aplicação para geração de páginas dinâmicas na Web. Então a empresa TcX começou a procurar por outro banco o mSQL(Mini SQL), uma ferramenta baseada em SQL mas com características pobres não possuindo por exemplo suporte a índices, e com desempenho inferior ao UNIREG.



Foi então que o desenvolvedor do banco UNIREG contatou o David Hughes criador do mSQL(Mini SQL), para saber do interesse dele em unir os dois bancos. Sendo positivo o interesse de David, a empresa TcX resolveu desenvolver um novo banco, mas mantendo ao máximo a compatibilidade com mSQL(Mini SQL). TcX foi esperta o suficiente para não reinventar o que já estava bem feito, ela construiu seu servidor baseado na estrutura que já estava montada do UNIREG e utilizou grande número de utilitários escritos para mSQL(Mini SQL) e fez API's para o novo servidor praticamente iguais ao mSQL(Mini SQL). Como resultado usuários do mSQL(Mini SQL) que decidissem mudar para o novo servidor da TcX, teriam apenas que fazer pequenas e simples mudanças nos códigos existentes.

Então foi em maio de 1995 que, definitivamente, a primeira versão do MySQL foi lançada. Um dos parceiros da TcX sugeriu a distribuição do servidor na Internet, o objetivo disso era a utilização de um modelo pioneiro desenvolvido por Aladdin Peter Deutsch. O resultado foi um maior flexibilidade em sem "copyright", que fez do MySQL mais difundido gratuitamente do que mSQL(Mini SQL).

Um fato importante a ser destacado sobre o MySQL é que esse SGBD também possui uma licença comercial, isto é, paga. Neste caso, é possível obter suporte diferenciado dos desenvolvedores.

Vale ressaltar também que, em fevereiro de 2008, o MySQL foi comprado pela Sun Microsystems, que pagou a quantia de 1 bilhão de dólares pela aquisição.

## 11. O Banco de Dados MySQL

O MySQL é um dos sistemas de gerenciamento de banco de dados mais populares que existe e, por ser otimizado para aplicações Web, é amplamente utilizado na internet. É muito comum encontrar serviços de hospedagem de sites que oferecem o MySQL e a linguagem PHP, justamente porque ambos trabalham muito bem em conjunto.

Outro fator que ajuda na popularidade do MySQL é sua disponibilidade para praticamente qualquer sistema operacional, como Linux, FreeBSD (e outros sistemas baseados em Unix), Windows e Mac OS X. Além disso, o MySQL é um software livre sob licença GPL (General Public License), o que significa que qualquer um pode estudá-lo ou alterá-lo conforme a necessidade.

Entre as características técnicas do SGBD MySQL, estão:

- Alta compatibilidade com linguagens como PHP, Java, Python, C#, Ruby e C/C++;
- Baixa exigência de processamento (em comparação como outros SGBD);
- Vários sistemas de armazenamento de dados (database engine), como MyISAM, MySQL Cluster, CSV, Merge, InnoDB, entre outros;
- Recursos como transactions (transações), conectividade segura, indexação de campos de texto, replicação, etc;
- Instruções em SQL como indicam o nome.

### 11.1

### Utilização

Uma característica fundamental do MySQL, quicá na origem do seu sucesso, é ser desenvolvido em código aberto e funcionar num grande número de sistemas operacionais: Windows, Linux, FreeBSD, BSDI, Solaris, Mac OS X, SunOS, SGI, etc.

É reconhecido pelo seu desempenho e robustez e também por ser multitarefa e multiusuário. A própria Wikipédia, usando o programa MediaWiki, utiliza o MySQL para gerenciar seu banco de dados, demonstrando que é possível utilizá-lo em sistemas de produção de alta exigência e em aplicações sofisticadas.

No passado (até à versão 3.x), devido a não possuir funcionalidades consideradas essenciais em muitas áreas, como stored procedures, two-phase commit, subselects, foreign keys ou integridade referencial, era frequentemente considerado um sistema mais "leve" e para aplicações menos exigentes, sendo preterido por outros sistemas como o PostgreSQL.

### 11.2 Características do MySQL

- Banco de dados de código aberto e gratuito;
- As tabelas criadas podem ter tamanho de até 4 GB;
- Suporta diferentes plataformas: Win32, Linux, FreeBSD, Unix, etc;
- Suporte às API's das Seguintes linguagens: PHP, Perl, C,C++,Java, Pynthon, etc;
- Suporte a múltiplos processadores;
- Um sofisticado sistema de senhas criptografadas flexível e Seguro.
- Suporte à ODBC, você pode facilmente conectar o Access a um banco de dados do MySQL;
- Suporta até 16 índices por tabela;

- Código fonte escrito em C e C++ e testado com uma variedade de diferentes compiladores;
- O Cliente conecta no MySQL através de conexões TCP/IP;
- Capacidade para manipular bancos com até 50 milhões de registros;
- Reduz a administração, engenharia e a sustentação custa por até 50%.

### 11.3 O Que o MySQL Faz de Melhor

- Aplicações Web;
- Aplicações de nível corporativo;
- Suporte a código fonte aberto;
- Requisitos de sistema baixo;
- Tabelas com tamanho grande;
- Estabilidade.

### 11.4 Segurança no MySQL

O MySQL possui componentes de segurança contra ameaças externas como crackers e outros, e também proteger os dados dos próprios usuários. O mysql apresenta vários níveis de segurança em relação ao acesso. Todas as informações de segurança estão armazenadas no banco mysql.

A filosofia de segurança em banco de dados refere-se a fornecer ao usuário apenas o que é essencial para o seu trabalho.

### 11.5 O MySQL é Gratuito?

Pessoas confundem "free" com "grátis" o que é comum aqui no Brasil. Mas em se tratando de software este "free" é de open source e não gratuito. Para poder utilizar o MySQL sob a licença GPL (*General Public License*) e não precisar pagar, o produto desenvolvido precisa ser GPL (*General Public License*) também, senão, orientamos a compra da licença comercial, com baixo custo, sendo comercializada por servidor, sem limites de usuários e processadores e ainda com garantia perpétua de atualização de versão para o resto da vida.

Empresas Que o Utilizam:

AOL

Departamento De Census dos E. U.

Dow Jones

EarthLink

Ericsson

Google

Hewlett-Packard

Instrumentos De Texas

Lucent

Lufthansa

NASA

Siemens

Steaks De Omaha

Suzuki

Tempo Inc.

### 11.6 Qual o Tamanho Que as Tabelas do MySQL Podem ter?

No MySQL versão 3.23 o tamanho máximo foi expandido até 8 milhões de terabytes. Com este tamanho de tabela maior permitido, o tamanho máximo efetivo das tabelas para o banco de dados MySQL é normalmente limitado pelas restrições do sistema operacional quanto ao tamanho dos arquivos, não mais por limites internos do MySQL.

### 11.7 O Logo

O logo do golfinho do MySQL foi desenhado pela empresa finlandesa Finnish Advertising Agency Priority em 2001. O golfinho foi escolhido como o símbolo para o banco de dados MySQL pois o animal é esperto, rápido e ágil. O nome do golfinho é Sakila. Esse nome foi escolhido pelos fundadores da MySQL AB em uma enorme lista de nomes sugeridos pelos usuários numa campanha chamada de "Name the Dolphin" realizada por ideia da empresa.

O nome vencedor foi enviado por Ambrose Twebaze, um desenvolvedor de software Open Source da Swaziland (África). De acordo com Ambrose, "Sakila" é um nome feminino, que tem suas raízes no Siswati, dialeto local de Swaziland.



### 11.8 Principais Tipos de Dados

**INT:** Representa um valor inteiro. Pode ser com sinal ou sem sinal

**REAL:** Representa um valor com ponto flutuante. Oferece uma grande precisão e uma extensa faixa de valores

**CHAR(n):** Representa um valor caractere com tamanho fixo.

**TEXT:** Representa um valor para caractere com tamanho variável

**DATE:** Representa um valor de data padrão. Formato: YYYY-MM-DD (2001-01-01)

**TIME:** Representa um valor de tempo padrão. Armazena a hora de um dia independente de uma data particular. Formato : hh:mm:ss (06:00:00)

### 11.9 Compreendendo os tipos de dados do MySQL

Os tipos de dados que pode ter um campo, podem-se agrupar em três grandes grupos:

1. Tipos numéricos
2. Tipos de Data
3. Tipos de Cadeia

#### 1) Tipos numéricos:

Existem tipos de dados numéricos, que se podem dividir em dois grandes grupos, os que estão em vírgula flutuante (com decimais) e os que não.

**Int:** número inteiro com ou sem sinal. Com sinal a margem de valores válidos é desde -2147483648 até 2147483647.

**Float:** número pequeno em vírgula flutuante de precisão simples. Os valores válidos vão desde -3.402823466E+38 até -1.175494351E-38,0 até desde 175494351E-38 até 3.402823466E+38. (é o tipo para ser usado em números reais, preço, e números que usa decimal, ex: 10,32).

#### 2) Tipos data:

Na hora de armazenar datas, há que ter em conta que MySQL não verifica de uma maneira estrita se uma data é válida ou não. Simplesmente comprova que o mês está compreendido entre 0 e 12 e que o dia está compreendido entre 0 e 31.

**Date:** tipo data armazena uma data. A margem de valores vai desde o 1 de Janeiro de 1001 ao 31 de dezembro de 9999. O formato de armazenamento é de ano-mes-dia.

**Time:** armazena uma hora. A margem de horas vai desde -838 horas, 59 minutos e 59 segundos. O formato de armazenamento é 'HH:MM:SS'.

#### 3) Tipos de cadeia:

**VarChar(n):** armazena uma cadeia de longitude variável. A cadeia poderá conter desde 0 até 255 caracteres.

### 12. MySQL Workbench

O MySQL Workbench é uma ferramenta gráfica para modelagem de dados, integrando criação e designer que pretende ser uma evolução do já famoso DBDesigner4. A ferramenta possibilita trabalhar diretamente com objetos *schema*, além de fazer a separação do modelo lógico do catálogo de banco de dados.

Toda a criação dos relacionamentos entre as tabelas pode ser baseada em chaves estrangeiras. Outro recurso que a ferramenta possibilita é realizar a engenharia reversa de esquemas do banco de dados, bem como gerar todos os scripts em SQL.

Ferramenta de modelagem repleta de recursos MySQL Workbench apresentar opções que não o deixam para trás em comparação aos outros programas para modelagem de dados. A ferramenta faz sincronia dos modelos de banco de dados, importa e exporta modelos do DBDesigner4, bem como exporta scripts SQL.

A modelagem dos seus bancos de dados pode assumir níveis conceituais, lógicos e físicos. MySQL Workbench apresenta uma arquitetura extensível, sendo possível visualizar a representação de tabelas, funções, entre outros.

### 13. Comandos Básicos do SQL

Iremos ver os comandos básicos do SQL, utilizados pela maioria dos bancos de dados, inclusive o MySQL.

Observe que todo comando SQL termina com um ; (**ponto e vírgula**).

#### 13.1 Comando Create Database

Este comando permite a criação do banco de dados.

##### Sintaxe:

```
CREATE DATABASE < nome_db >;
```

onde:

· **nome\_db** - indica o nome do Banco de Dados a ser criado.

##### Exemplo:

```
Create database focus;
```

#### 13.2 Comando Drop Database

Este comando permite a exclusão do banco de dados. Caso existam tabelas criadas no banco, as mesmas serão apagadas.

##### Sintaxe:

```
DROP DATABASE < nome_db >;
```

onde:

· **nome\_db** - indica o nome do Banco de Dados a ser criado.

##### Exemplo:

```
Drop database focus;
```

#### 13.3 Comando Use

Este comando serve para selecionarmos o banco de dados a ser utilizado.

##### Sintaxe:

```
USE < nome_db >;
```

onde:

· **nome\_db** - indica o nome do Banco de Dados a ser criado.

##### Exemplo:

```
Use focus;
```

#### 13.4 Comando Create Table

Este comando permite a criação de tabelas no banco de dados.

##### Sintaxe:

```
CREATE TABLE < nome_tabela > (  
    nome_atributo1 < tipo > [ NOT NULL ],  
    nome_atributo2 < tipo > [ NOT NULL ],  
    .....  
    nome_atributoN < tipo > [ NOT NULL ]  
) ;
```

onde:

- **nome\_tabela** - indica o nome da tabela a ser criada.
- **nome\_atributo** - indica o nome do campo a ser criado na tabela.
- **tipo** - indica a definição do tipo de atributo ( integer(n), char(n), ... ).

**Exemplo:**

```
Create table alunos (  
    Id_aluno INT(3) NOT NULL,  
    nome VARCHAR(40) NOT NULL,  
    endereco VARCHAR(50) NOT NULL,  
    turma VARCHAR(20) NOT NULL,  
    PRIMARY KEY (Id_aluno)  
);
```

### 13.4.1 CREATE TABLE (com Cláusula DEFAULT)

A cláusula DEFAULT permite definir um valor padrão para um campo, que será utilizado caso não seja informado nenhum valor para esse campo na inserção de um registro na tabela.

**Sintaxe:**

```
sexo char(1) default 'M'
```

No exemplo acima, caso o campo "sexo" da tabela não seja preenchido com um valor durante a inserção de um registro, será assumido o valor 'M' para o campo. Para campos do tipo NUMÉRICO, o valor DEFAULT é escrito sem aspas. Exemplo:

```
salario decimal(10,2) default 0,
```

**Exemplo:**

```
CREATE TABLE clientes(  
    cpf integer unsigned not null,  
    nome varchar(100) not null,  
    data_nascimento date not null,  
    sexo char(1) default 'M',  
    salario decimal(10,2) default 0,  
    profissao varchar(30),  
    primary key(cpf)  
);
```

### 14.5 Comando Drop Table

Este comando elimina a definição da tabela, seus dados e referências.

**Sintaxe:**

```
DROP TABLE < nome_tabela > ;
```

**Exemplo:**

```
Drop table alunos;
```

### 14.6 Comando Insert

Este comando insere os dados em tabelas.

**Sintaxe:**

```
INSERT INTO < nome_tabela > (<campos_data_tabela>) VALUES(<valores_campos>);
```

**Exemplo:**

```
insert into tbl_alunos (ra, nome, data_nasc, turma) values(1, 'MARCOS COSTA',  
'1981-02-06', '1DSN');
```

### 13.7 Comando Select

Consulta os dados em tabelas.

**Sintaxe:**

SELECT \* FROM < nome\_tabela > WHERE <condição>;

**Exemplo:**

**SELECT \* FROM tbl\_alunos WHERE ra = 1;**

### 13.8 Comando Update

Atualiza os dados em tabelas.

**Sintaxe:**

UPDATE < nome\_tabela > SET <campos> WHERE <condição>;

**Exemplo:**

**UPDATE tbl\_alunos SET nome = "ANTONIO JOSÉ DA SILVA" WHERE ra = 1;**

### 13.9 Comando Delete

Deleta os dados em tabelas (CUIDADO).

**Sintaxe:**

DELETE < nome\_tabela > WHERE <condição>;

**Exemplo:**

**DELETE FROM tbl\_alunos WHERE ra = 1;**

### 13.10 Comando Alter

Este comando permite inserir/eliminar atributos nas tabelas já existentes.

**Sintaxe:**

ALTER TABLE < nome\_tabela > ADD / DROP (  
     nome\_atributo1 < tipo > [ NOT NULL ],  
     nome\_atributoN < tipo > [ NOT NULL ]  
 );

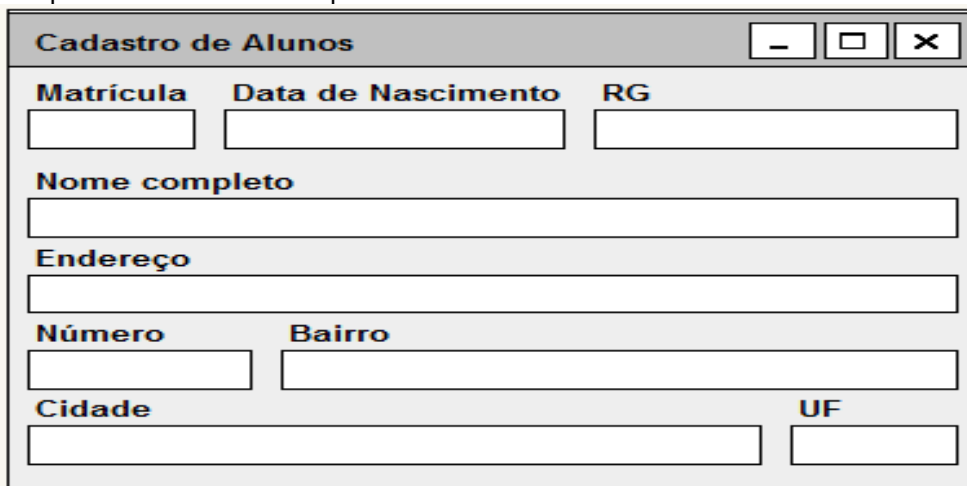
**Exemplos:**

**Alter table alunos ADD COLUMN turno char(10) NOT NULL;**

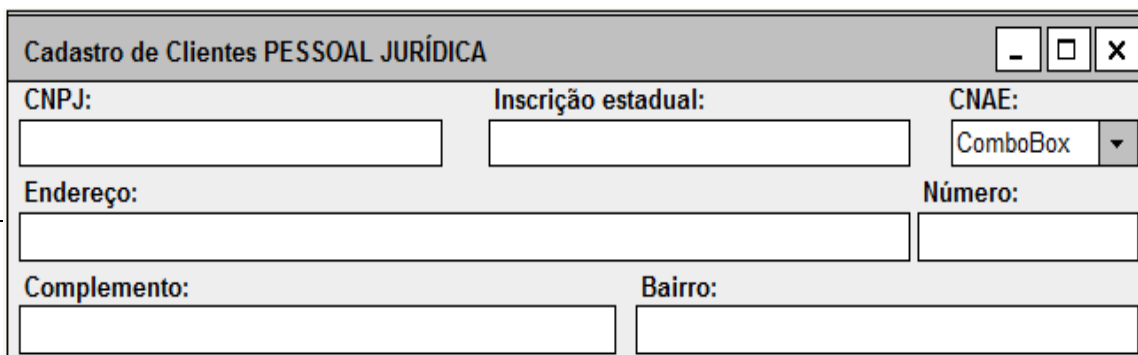
**Alter table alunos DROP COLUMN turno;**

## 14. Exercícios para fixação de conteúdo

1-) De acordo com o Layout de Tela, crie uma tabela e defina a chave primária compreendendo os dados apresentados:



2-) De acordo com o Layout de Tela, crie uma tabela e defina a chave primária compreendendo os dados apresentados:





**3-)** Vamos agora fazer uma atividade juntos de análise, conforme diz abaixo sobre uma Escola de Música:

Uma escola de música precisa manter uma base de dados organizada com o objetivo de prover informações sobre músicos, orquestras, sinfonias e instrumentos:

- Cada orquestra é catalogada contendo o seu nome, cidade, país e data correspondentes à sua criação;

- Orquestras executam sinfonias, as mais variadas. Os profundos conhecedores de música são capazes até de selecionar a orquestra que melhor desempenha uma determinada sinfonia. De cada uma sinfonia, é possível saber o seu nome, o compositor e a data de sua criação;

- Orquestras são constituídas de músicos, os mais variados, de acordo com a sua função dentro da mesma: maestro, flautista, etc. Cada músico é catalogado contendo: nome do músico, identidade, nacionalidade e data de nascimento. Um músico só pode pertencer a uma orquestra;

- Músicos tocam sinfonias, porém em alguns casos, alguns músicos podem mudar de função segundo a sinfonia (por exemplo, um violinista pode virar maestro). A data em que um músico apresenta uma determinada sinfonia também é importante no contexto.

- Cada músico pode ser apto a tocar vários instrumentos, mas em cada sinfonia toca apenas um instrumento, pois depende de sua função na sinfonia.

## **17. O MySQL WorkBench.**

No decorrer desse curso veremos uma série de recursos de gerenciamento ligados ao Sistema Gerenciador de Banco de Dados Relacionais (SGBDR) “**Mysql**”, esses recursos visam desde a manutenção dos bancos de dados e suas tabelas através de recursos como “**triggers**”, “**stored procedures**”, “**views**”, “**index**” recursos esses fundamentais para a boa “**saúde**” de um banco de dados no que se refere a consistência dos dados e performance, passando também pela parte de segurança referente a criação e gestão de usuários e atribuição e revogação de direitos a esses usuários além de formas de realizar backups das bases de dados existentes.

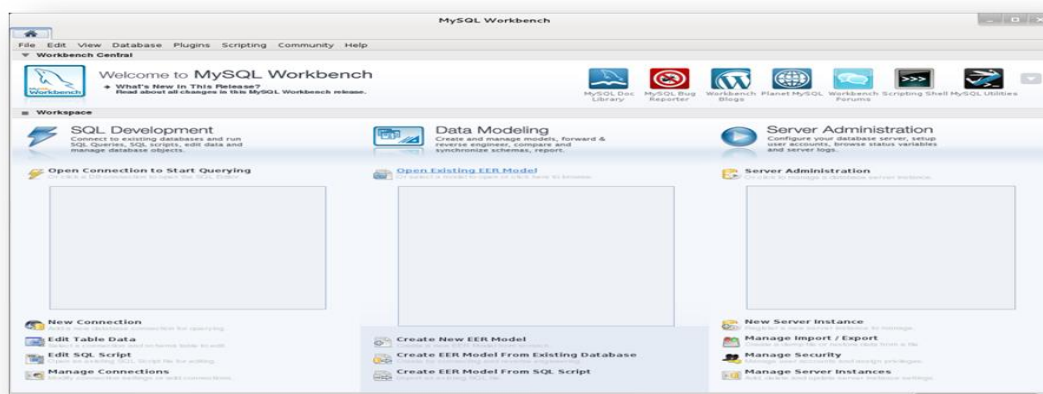
Tais tarefas podem se mostrar desafiadoras ao “**Database Administrator**” (DBA) profissional responsável por tais tarefas, logo se faz necessário o uso de uma ou mais ferramentas que permita um alto grau de produção para esse profissional. Existe uma gama enorme de ferramentas capazes de fornecer esse auxílio para os mais diversos SGBDR corporativos existentes hoje no mercado. No nosso caso estamos trabalhando com o “**MySQL**”

e vamos utilizar uma “**Integrated Development Environment**” (IDE) ou “**Ambiente Integrado de Desenvolvimento**” conhecido como “**MySQL Workbench**”.

Essa ferramenta reúne dentro de si recursos divididos em três grupos básicos são eles:

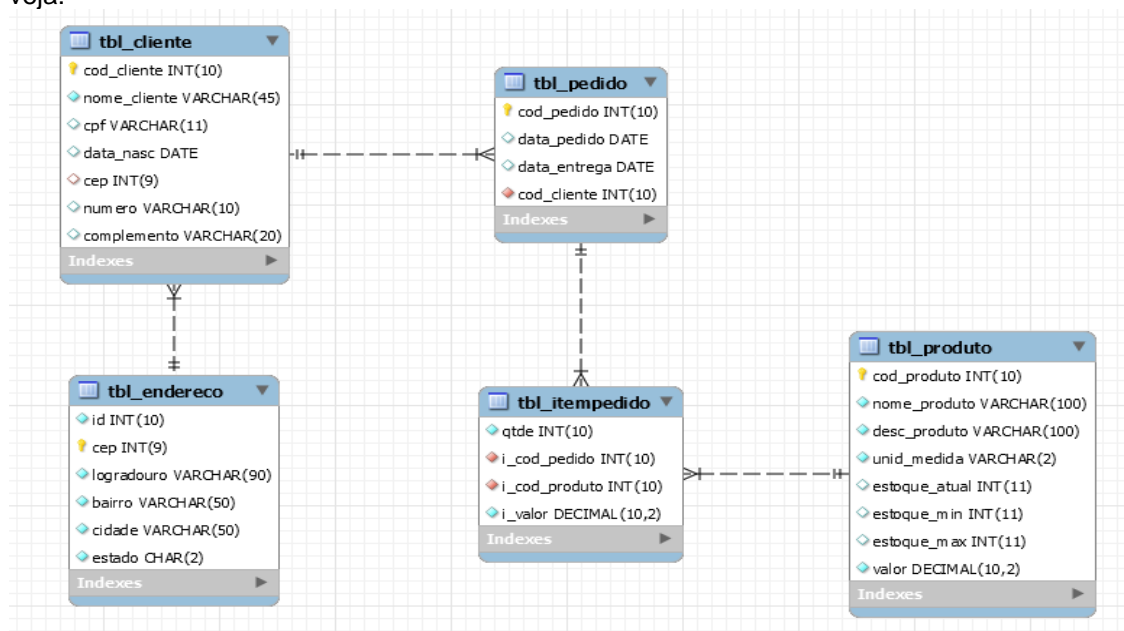
- **SQL Development** – Reúne ferramentas ligadas a codificação de comandos **SQL Data Manipulation Language** ou **Linguagem de Manipulação de Dados (DML)**, **Data definition Language** ou **Linguagem de Definição de Dados (DDL)**, **Data Control Language** ou **Linguagem de controle de Dados (DCL)**, **Data Transaction Language** ou **Linguagem de Transação de Dados (DTL)** e **Data Query Language** ou **Linguagem de Consulta de dados (DQL)**.
- **Data Modeling** – Reúne ferramentas ligadas a modelagem de banco de dados permitindo a criação de diagramas/modelos de entidade relacionamento, além disso é possível converter esses diagramas em projetos físicos (bancos de dados) através de interface gráfica e podemos também realizar processos de engenharia reversa onde podemos “apontar” para um projeto físico de um determinado banco de dados e converter esse em diagramas/modelos de entidade relacionamento isso é muito útil principalmente quando estamos trabalhando em projetos que foram legados (já existiam e nos foram passados a título de continuação) e por alguma razão qualquer a equipe anterior não documentou a base de dados ou até mesmo essa documentação foi perdida.
- **Server Administrator** – Reúne ferramentas uteis para a gerencia de backups e da “saúde” das bases de dados criadas na instancia do “**MySQL**” que se está gerenciando, permite por exemplo verificar quantas conexões estão ativas nas bases de dados existentes, verificar a “saúde” da memória principal do servidor que hospeda a instancia do “**MySQL**”, o tráfego de dados, índice de eficiência das chaves, criar e associar a usuários diretos ou remover usuários e direitos, realizar backups entre outras ações.

A imagem abaixo ilustra a tela inicial do “**MySQL WorkBench**” onde podemos escolher entre os grupos de ferramentas citadas:



## 18. SQL

Para nossa aula dar mais efetividade vamos criar essa base de dados como referência, veja:



Mas iremos fazer esse banco de dados através de comandos SQL, primeiramente vamos organizar as coisas, iremos criar um Script SQL chamado “bd\_vendas DDL” onde irá conter a criação do banco de dados e as tabelas mostradas no DER anterior.

- Criando e habilitando a base de dados:

```

/*
CRIAÇÃO DO BANCO DE DADOS BD_VENDAS - TLBD III
*/
create database bd_vendas;

-----

/*
HABILITANDO O BANCO DE DADOS PARA USO
*/
use bd_vendas;
  
```

- Criando a tabela de produtos:

```

/*
CRIAÇÃO DA TABELA DE PRODUTOS
*/
create table tbl_produto (
  cod_produto int unsigned auto_increment,
  nome_produto varchar(100) not null,
  desc_produto varchar(100) not null,
  unid_medida varchar(2) not null,
  estoque_atual int default 0,
  estoque_min int default 0,
  estoque_max int default 0,
  valor decimal(10,2) not null,
  primary key (cod_produto));
  
```

- Criando a tabela de endereços para armazenar os CEPs:

```

/*
  CRIAÇÃO DA TABELA DE ENDEREÇO - CEP
*/
create table tbl_endereco (
  id int(10) not null,
  cep int(9) not null,
  logradouro varchar(90) not null,
  bairro varchar(50) not null,
  cidade varchar(50) not null,
  estado char(2) not null,

  constraint pk_endereco primary key (cep)
);

```

- Criando a tabela de clientes:

```

/*
  CRIAÇÃO DA TABELA DE CLIENTES
*/
create table tbl_cliente (
  cod_cliente int unsigned auto_increment,
  nome_cliente varchar(45) not null,
  cpf varchar(11) default '',
  data_nasc date,
  cep int(9) default 0,
  numero varchar(10) default '',
  complemento varchar(20) default '',

  primary key (cod_cliente),
  constraint foreign key fk_clienccep (cep) references tbl_endereco(cep)
);

```

- Criando a tabela de pedidos:

```

/*
  CRIAÇÃO DA TABELA DE PEDIDOS
*/
create table tbl_pedido (
  cod_pedido int unsigned auto_increment,
  data_pedido date,
  data_entrega date,
  cod_cliente int unsigned not null,
  primary key (cod_pedido),
  constraint fk_cliente foreign key (cod_cliente)
  references tbl_cliente(cod_cliente));

```

- Criando a tabela de itens do pedido:

```

/*
  CRIAÇÃO DA TABELA DE ITENS DO PEDIDO
*/
create table tbl_itempedido (
  qtde int unsigned not null,
  i_cod_pedido int unsigned not null,
  i_cod_produto int unsigned not null,
  i_valor decimal(10,2) not null,
  constraint fk_pedido1
  foreign key (i_cod_pedido)
  references tbl_pedido (cod_pedido),
  constraint fk_tbl_produto1
  foreign key (i_cod_produto)
  references tbl_produto(cod_produto));

```

Feito isso salvem esse arquivo e mantenha sempre com vocês em nossas aulas, pois iremos incrementar mais tabelas no decorrer do curso.

Caso queiram para verificar se a construção do seu banco de dados ficou igual a mencionada inicialmente, façam a engenharia reversa (*Reverse Engineer*) no menu *DATABASE* para verificar se ficou igual.

### 18. Popular as tabelas

Iremos agora inserir dados nas tabelas para que possamos dar seguimento aos nossos estudos, mas a primeira pergunta que vem a nossa mente é: “Por qual tabela devo começar?”. Para essa resposta precisamos analisar o DER e verificar quais tabelas não possuem dependências de chaves estrangeiras e começar pelas mesmas, assim as tabelas de endereço e produto devem ser as primeiras a serem populadas, pois são tabelas que fornecem suas chaves para o preenchimento de outras, se fizermos ao contrário, a inserção dará erro.

Para isso, vamos criar outro Script SQL chamado “bd\_vendas DML”, onde iremos colocar todos os *INSERTs* que iremos fazer para popular essas tabelas, esse também deverá acompanhar os senhores para nossas aulas.

Para darmos início, vamos popular a tabela de endereço com os CEPs, vou disponibilizar um Script com esses *inserts* com as cidades de Taboão da Serra e Embu das Artes, darão cerca de 2000 registros nessa tabela.

Na sequência vamos inserir os produtos:

```

insert into tbl_produto (nome_produto, desc_produto, unid_medida,
estoque_atual,estoque_min, estoque_max,valor) values
('Arroz', 'Arroz agulhinha tipo 1','SC', 10,2,20, 12.50),
('Feijão', 'Feijão carioquinha com casca','SC', 25,5,60, 7.50),
('Macarrão', 'Macarrão Adria espaguete','PC', 50,10,80, 5.50),
('Óleo', 'Óleo Lisa','LT', 15,10,45, 6.50),
('Vinagre', 'Vinagre Castelo','GR', 30,10,50, 7.89),
('Batata', 'Batata lavada','KG', 100,50,200, 4.50),
('Tomate', 'Tomate vermelho','KG', 80,8,160, 6.90),
('Cebola', 'Cebola com casca','KG', 50,5,100, 6.99),
('Leite', 'Leite Leco','CX', 25,10,90, 2.50),
('Café', 'Café do Ponto','SC', 500,100,200, 11.50);

```

Feito isso, podemos realizar as inserções dos clientes, pois sem os dados da tabela de CEPs não seria possível chegar a essa etapa. **Só lembrando que para a inserção dessa tabela, o CEP informado deverá estar cadastrado na tabela de CEPs.**

```

/*
Clientes
*/
insert into tbl_cliente(nome_cliente,cpf,data_nasc,cep,numero,complemento) values
('Marcos Costa de Sousa','12345678901','1981-02-06',6768100,'1525','apto 166C'),
('Zoroastro Zoando','01987654321','1989-06-15',6757190,'250',''),
('Idelbrandolância Silva','54698721364','1974-09-27',6753001,'120',''),
('Cosmólio Ferreira','41368529687','1966-12-01',6753020,'25','apto 255 F'),
('Conegunda Prado','54781269501','1950-10-06',6753020,'50','apto 166C'),
('Brogundes Asmônio','41256398745','1940-05-10',6753400,'100',''),
('Iscruência da Silva','12457965823','1974-11-25',6803040,'5',''),
('Zizafânio Zizando','54123698562','1964-08-14',6803140,'25',''),
('Ricuerda Zunda','21698534589','1934-10-14',6803045,'123',''),
('Aninoado Zinzão','25639856971','1976-12-25',6803070,'50','');

```

### Exercício

“Populem” as tabelas de acordo com o MER desenvolvido por vocês, com uma massa de dados consistente e concisa seguindo os seguintes critérios:

- 1º - Criem no mínimo 5 pedidos;
- 2º - Para cada pedido, pelo menos 3 itens em cada;
- 3º - Para evidenciar, em um arquivo .doc enviem:
  - INSERT da tabela de pedido;
  - INSERT da tabela de itempedido;

### 19. Transactions em banco de dados

Transação ou *Transaction* é uma única unidade de trabalho processada pelo Sistema de Gerenciamento de Banco de Dados (SGBD). Imagine que precisamos realizar em uma única transação duas operações de manipulação de dados (DML, do inglês Data Manipulation Language), como *INSERT*, *DELETE* e *UPDATE*. Estas operações só podem se tornar permanentes no banco de dados se todas forem executadas com sucesso. Em caso de falhas em uma das duas é possível cancelar a transação, porém todas modificações realizadas durante a mesma serão descartadas, inclusive a que obteve sucesso. Assim, os dados permanecem íntegros e consistentes.

Podemos caracterizar as transações conforme abaixo:

**O *BEGIN TRANSACTION*** indica onde ela deve começar, então os comando SQL a seguir estarão dentro desta transação.

**O *COMMIT TRANSACTION*** indica o fim normal da transação, o que tiver de comando depois já não fará parte desta transação. Neste momento tudo o que foi manipulado passa fazer parte do banco de dados normalmente e operações diversas passam enxergar o que foi feito.

**O *ROLLBACK TRANSACTION*** também fecha o bloco da transação e é a indicação que a transação deve ser terminada, mas tudo que tentou ser feito deve ser descartado porque alguma coisa errada aconteceu e ela não pode terminar normalmente. Nada realizado dentro dela será perdurado no banco de dados.

Ao contrário do que muita gente acredita *rollback* no contexto de banco de dados não significa reverter e sim voltar ao estado original. Um processo de reversão seria de complicadíssimo à impossível.

A maioria dos comandos SQL são transacionais implicitamente, ou seja, ele por si só já é uma transação. Você só precisa usar esses comandos citados quando precisa usar múltiplos comandos.

## 20. Comandos DML

Vamos neste momento alternar nossas aulas de acordo com o que foi visto até agora, neste capítulo iremos ver *UPDATE*, *INSERT*, *DELETE* e principalmente o *SELECT*, nos aprofundando na DQL, que é uma ramificação da DML, tudo isso para melhor fixação do conteúdo, que é vital neste momento do nosso curso. Apesar de termos as tabelas criadas e populadas conforme aula anterior, neste primeiro momento iremos nos ater em trabalhar somente com a tabela de Clientes **TBL\_CLIENTE** e Endereço **TBL\_ENDERECO**, em seguida passamos para as outras tabelas até finalizarmos fazendo os todos os procedimentos vistos com as chaves estrangeiras criadas nestas tabelas.

### 20.1 – SELECT

De acordo com a tabela “populada”, podemos retirar informações ricas da mesma, para isso iremos ver a DQL, que nos ajudará muito nesta extração de informações.

#### 20.1.1 – CLÁUSULA DISTINCT

A cláusula *DISTINCT* permite que os dados repetidos de uma tabela sejam exibidos uma única vez, vamos observar o exemplo abaixo:

```
select distinct bairro
from tbl_endereco;
```

Neste caso somente serão mostrados os bairros distintos existentes na **TBL\_ENDERECO**, ou seja, aparecerá no resultado 202 linhas, então podemos dizer que a tabela de endereço da nossa base de dados possui 202 bairros diferentes em seu cadastro.

Como qualquer *SELECT*, podemos refinar mais a consulta, por exemplo se quisermos ver a quantidade distintas de bairro por município por exemplo:

```
select distinct bairro
from tbl_endereco
where cidade = 'Taboão da Serra';
```

Nesse outro exemplo filtramos pelo município de Taboão da Serra, onde foram apresentadas 98 linhas, ou seja, em Taboão da Serra a nossa tabela possui 98 bairros diferentes.



### 20.1.2 – CLÁUSULA LIMIT

A cláusula LIMIT é usada para determinar um limite para a quantidade e para a faixa de linhas que podem ser retornadas, vamos neste nosso exemplo mostrar as linhas referentes aos clientes ISCRUÊNCIA e ZIZAFÂNIO observar o exemplo abaixo:

```
select *  
from tbl_cliente  
limit 6,2
```

Observe que o primeiro parâmetro de limite especificado (6) refere-se ao deslocamento, ou seja, ao ponto inicial, ao passo que o segundo parâmetro (2) refere-se a quantidade de linhas a serem retornadas.

Quando um único parâmetro é informado, este se refere a quantidade de linhas retornadas.

### 20.1.3 – CLÁUSULA ORDER BY

A cláusula ORDER BY pode ser usada juntamente com a cláusula LIMIT quando for importante que as linhas sejam retornadas de acordo com uma determinada ordem, observar o exemplo abaixo:

```
select *  
from tbl_cliente  
order by nome_cliente  
limit 6,2;
```

Esta ordem também poderá ser ASCENDENTE ou DESCENDENTE dependendo da minha necessidade, veja:

```
select *  
from tbl_cliente  
order by nome_cliente desc  
limit 6,2;
```

Ou

```
select *  
from tbl_cliente  
order by nome_cliente asc  
limit 6,2;
```

Quando não informamos a classificação depois da cláusula ORDER BY por default é considerado ASC.

### 20.1.4 – CLÁUSULA WHERE

A cláusula WHERE serve para determinar quais os dados de uma tabela que serão afetados pelos comandos SELECT, UPDATE e DELETE, podemos dizer então que esta cláusula determina o escopo de uma consulta a algumas0 linhas, realizando uma filtragem dos dados que estejam de acordo com as condições definidas, vamos observar o exemplo abaixo:

```
select *  
from tbl_cliente  
where cpf = '12345678901';
```

Neste caso trará os dados do cliente "MARCOS COSTA DE SOUSA" que possui o CPF 12345678901.

```
select *  
from tbl_endereco  
where bairro = 'Jardim Caner'
```

Neste outro caso os dados de endereço que são do Bairro JD CANER serão mostrados.

```
select *  
from tbl_endereco  
where cidade = 'Taboão da Serra'
```

Neste outro caso os dados de endereço que são da Cidade de TABOÃO DA SERRA serão mostrados.

#### 20.1.5 – OPERADORES AND E OR

Os operadores lógicos AND e OR são utilizados junto com a cláusula WHERE quando for necessário utilizar mais de uma condição de comparação, vamos observar o exemplo abaixo:

```
select * from tbl_endereco  
where estado = 'SP'  
and cidade = 'Taboão da Serra'
```

Neste caso todos endereços da UF de SP e a CIDADE de TABOÃO DA SERRA serão mostrados.

```
select * from tbl_endereco  
where estado = 'SP'  
or cidade = 'Taboão da Serra';
```

Neste caso todos os endereços da UF de SP ou da CIDADE de TABOÃO DA SERRA serão mostrados, em nosso exemplo irão aparecer todos os dados cadastrados, em caso de uma base de dados de todo o Brasil por exemplo, se existisse a cidade de Taboão da Serra na Bahia por exemplo, seria mostrada no resultado.

#### 20.1.6 – OPERADOR IN

O operador IN permite checar se o valor de uma consulta está presente em uma lista de elementos.

```
select * from tbl_endereco  
where bairro in ('Jardim Kuabara', 'Jardim Pazini');
```

Neste caso todos os Endereços dos BAIRROS Jardim Kuabara e Jardim Pazini serão mostrados.

Com o NOT antecedido do IN, faz a operação inversa, ou seja não trará os endereços dos BAIRROS Jardim Kuabara e Jardim Pazini.

```
select * from tbl_endereco  
where bairro not in ('Jardim Kuabara', 'Jardim Pazini');
```

#### 20.1.7 – OPERADOR LIKE

O operador LIKE é empregado quando a base para realizar pesquisa forem as colunas que estão no formato char ou varchar, vamos observar o exemplo abaixo:

```
select * from tbl_cliente  
where nome_cliente like 'M%'
```

Neste caso todos os clientes que possuem a letra M no início do seu nome serão mostrados.

```
select * from tbl_cliente  
where nome_cliente like '%M%';
```

Neste caso todos os clientes que possuem a letra M em qualquer parte do seu nome serão mostrados.

```
select * from tbl_cliente  
where nome_cliente like '%M';
```

Neste caso todos os clientes que possuem a letra M no fim do seu nome serão mostrados.

Com o NOT antecedido do LIKE, faz a operação inversa, ou seja não trará os clientes dos nas situações acima, veja:.

```
select * from tbl_cliente  
where nome_cliente not like 'M%';
```

Neste caso todos os clientes que possuem a letra M no início do seu nome não serão mostrados.

```
select * from tbl_cliente  
where nome_cliente not like '%M%';
```

Neste caso todos os clientes que possuem a letra M em qualquer parte do seu nome não serão mostrados.

```
select * from tbl_cliente  
where nome_cliente not like '%M';
```

Neste caso todos os clientes que possuem a letra M no fim do seu nome não serão mostrados.

```
select * from tbl_cliente  
where nome_cliente like 'M_R_S%';
```

Neste caso todos os clientes que possuem na 1ª posição de seus nomes letra M, na 3ª posição a letra R e na 5ª posição a letra S e não importando o restante do nome serão mostrados, nesse caso adivinhem o nome de quem aparecerá!!!

#### 20.1.8 – OPERADOR BETWEEN

O operador BETWEEN tem a função de permitir a consulta de uma determinada faixa de valores. Este operador permite checar se o valor de uma coluna encontra-se em um determinado intervalo, vamos observar o exemplo abaixo:

```
select * from tbl_produto  
where estoque_min between 10 and 100
```

Neste caso serão mostrados todos os registros que possuem o campo ESTOQUE\_MIN entre 10 e 100.

Outro exemplo, podemos usar o NOT BETWEEN

```
select * from tbl_produto  
where estoque_min not between 10 and 100
```

Neste caso é feito o contrário do exemplo dado, ele trará os registro que NÃO estejam com o campo ESTOQUE\_MIN entre 10 e 100.

#### 20.1.9 – CLÁUSULA COUNT

A cláusula COUNT serve para contarmos quantas ocorrências existe um determinado SELECT, vamos observar o exemplo a seguir:

```
select count(*) from tbl_produto  
where estoque_min between 10 and 100
```

Neste caso é realizada a contagem de quantas ocorrências existem de acordo com o campo ESTOQUE\_MIN entre 10 e 100, ele mostrará o número 6.

#### 20.1.10 – CLÁUSULA GROUP BY

A cláusula *GROUP BY* serve para agrupar diversos registros com base em uma ou mais colunas da tabela, vamos observar o exemplo abaixo:

```
select count(*), unid_medida
from tbl_produto
group by unid_medida
```

Neste caso serão mostrados os registros agrupados pelo campo UNID\_MEDIDA, aparecerão CX = 1, GR = 1, KG = 3, LT = 1, PC = 1 e SC = 3.

```
select count(*), month(data_nasc)
from tbl_cliente
where year(data_nasc) > 1930
group by month(data_nasc)
order by 2
```

Neste caso será mostrada a quantidade de ocorrências existentes de acordo o mês do campo DATA\_NASC da tabela TBL\_CLIENTE, pois estamos contando e agrupando pelo mês, com o ano do mesmo campo maior que 1930.

```
select count(*) as qtde, month(data_nasc) as mes,
year(data_nasc) as ano
from tbl_cliente
where year(data_nasc) > 1930
group by month(data_nasc), year(data_nasc)
order by 2
```

Neste caso será mostrada a quantidade de ocorrências existentes igualmente ao caso anterior, só com a diferença de colocarmos o ano e agrupando o mesmo, isso nos trará mais uma linha no resultado.

### 20.2 Funções agregadas

Podemos resumir as informações dos dados de uma tabela através de funções agregadas, que são aplicadas sobre as colunas de uma tabela. Devemos passar como argumento de entrada uma coluna que precisamos extrair umas simples informação da mesma, um único resultado, para que assim, não seja preciso analisar informação por informação, com uso da nossa força bruta.

Você sabe como obter um valor resultante de uma simples soma dos valores de uma coluna específica?

Sabe extrair qual é o maior ou menor valor entre muitos em um conjunto?

Sabe quantas linhas uma tabela possui?

Sabe fazer a média entre um conjunto de valores?

Iremos ver 4 diferentes funções agregadas que geram diferentes tipos de resultado, são elas:

- SUM(): computa e retorna o resultado da soma de todos os valores de uma coluna informada;
- AVG(): retorna a média dos valores de uma coluna;
- MIN(): encontra o menor valor dentre aqueles contidos na coluna de entrada;
- MAX(): encontra o maior dentre todos os valores de um conjunto;

#### 20.2.1 Função SUM()

A função soma computa a soma dos valores de uma coluna. Essa função serve para manipularmos dados numéricos. O tipo numérico do dado pode ser *integer* e *decimal*. O resultado da função *SUM()* será do mesmo tipo da coluna que está sendo utilizada por esta função, vamos observar o exemplo abaixo:

```
select sum(estoque_min) as soma_estoque
from tbl_produto;
```

Neste caso será mostrada a soma do campo ESTOQUE\_MIN existente na tabela.

### 20.2.2 Função AVG()

A função `AVG()` computa a média entre os valores de uma coluna de dados. Assim como a função `SUM()`, esta função deve ser aplicada sobre dados numéricos. Porque esta função direciona a soma dos valores, da coluna informada, para uma coluna temporária e divide o resultado da soma pela quantidade de valores que foram usadas na soma. Esse resultado final pode ser de um tipo diferente do tipo da coluna usada. Por exemplo, se você aplicar `AVG` em uma coluna onde só contenha inteiros, o resultado será *decimal*, isso porque ocorre uma divisão antes de finalizar o processo desta função, vamos observar o exemplo abaixo:

```
select avg(estoque_max) as media_qtde_max
from tbl_produto;
```

Neste caso será mostrada a média do campo ESTOQUE\_MAX existente na tabela.

### 20.2.3 Função MIN()

A função `MIN` serve para obtermos o valor mínimo existente em uma determinada coluna, vamos observar o exemplo abaixo:

```
select min(valor) as vlr_min_preco
from tbl_produto;
```

Neste caso será mostrado o mínimo valor do campo VALOR existente na tabela.

### 20.2.4 Função MAX()

A função `MAX` serve para obtermos o valor máximo existente em uma determinada coluna, vamos observar o exemplo abaixo:

```
select max(valor) as vlr_max_preco
from tbl_produto;
```

Neste caso será mostrado o máximo valor do campo VALOR existente na tabela.

### 20.2.5 Operações matemáticas no SELECT

Podemos realizar “contas” no próprio `SELECT` para agilizar o processo de consulta, vejamos os exemplos:

```
select qtde * i_valor as valor_produto
from tbl_item_pedido
where i_cod_produto = 2
and i_cod_pedido = 1
```

Neste caso será mostrado o resultado da multiplicação dos dados das colunas VALOR e QTDE.

## 20.3 Joins de tabelas

Quando queremos relacionar mais de uma tabela no `SELECT`, podemos contar com alguns tipos de `JOINS` que veremos a seguir:

### 20.3.1 Inner Join

É o método de junção mais conhecido e, pois retorna os registros que são comuns às duas tabelas, precisamos fazer a amarração das tabelas com as chaves respectivas, veja os exemplos:

```
select *
from tbl_cliente as c
inner join tbl_pedido as p
on c.cod_cliente = p.cod_cliente
```

Ou

```
select *
from tbl_cliente as c, tbl_pedido as p
where c.cod_cliente = p.cod_cliente
```

Ambos darão o resultado:

cod_cliente	nome_cliente	cpf	data_nasc	cep	numero	complemento	cod_pedido	data_pedido	data_entrega	cod_cliente
1	Marcos Costa de Sousa	12345678901	1981-02-06	6768100	1525	aoto 166C	1	2019-01-01	2019-01-04	1
2	Zoroastro Zoando	01987654321	1989-06-15	6757190	250		2	2019-01-05	2019-01-07	2
3	Idelbrandolâncio Silva	54698721364	1974-09-27	6753001	120		3	2019-01-12	2019-01-15	3
5	Coneunda Prado	54781269501	1950-10-06	6753020	50	aoto 166C	4	2019-01-15	2019-01-16	5
7	Iscrência da Silva	12457965823	1974-11-25	6803040	5		5	2019-01-20	2019-01-22	7

### 20.3.2 Left Join

Tem como resultado todos os registros que estão na tabela A (mesmo que não estejam na tabela B) e os registros da tabela B que são comuns à tabela A, precisamos fazer a amarração das tabelas com as chaves respectivas, veja os exemplos:

```
select *
from tbl_cliente as c left join tbl_pedido as p
on c.cod_cliente = p.cod_cliente
```

O resultado será esse:

cod_cliente	nome_cliente	cpf	data_nasc	cep	numero	complemento	cod_pedido	data_pedido	data_entrega	cod_cliente
1	Marcos Costa de Sousa	12345678901	1981-02-06	6768100	1525	aoto 166C	1	2019-01-01	2019-01-04	1
2	Zoroastro Zoando	01987654321	1989-06-15	6757190	250		2	2019-01-05	2019-01-07	2
3	Idelbrandolâncio Silva	54698721364	1974-09-27	6753001	120		3	2019-01-12	2019-01-15	3
5	Coneunda Prado	54781269501	1950-10-06	6753020	50	aoto 166C	4	2019-01-15	2019-01-16	5
7	Iscrência da Silva	12457965823	1974-11-25	6803040	5		5	2019-01-20	2019-01-22	7
4	Cosmólio Ferreira	41368529687	1966-12-01	6753020	25	aoto 255 F	NULL	NULL	NULL	NULL
6	Broundes Asmônio	41256398745	1940-05-10	6753400	100		NULL	NULL	NULL	NULL
8	Zizafânio Zizundo	54123698562	1964-08-14	6803140	25		NULL	NULL	NULL	NULL
9	Ricuerda Zunda	21698534589	1934-10-14	6803045	123		NULL	NULL	NULL	NULL
10	Aninoado Zinzão	25639856971	1976-12-25	6803070	50		NULL	NULL	NULL	NULL

### 20.3.3 Right Join

Teremos como resultado todos os registros que estão na tabela B (mesmo que não estejam na tabela A) e os registros da tabela A que são comuns à tabela B, precisamos fazer a amarração das tabelas com as chaves respectivas, veja os exemplos:

```
select *
from tbl_pedido as p right join tbl_cliente as c
on p.cod_cliente = c.cod_cliente
```

O resultado será:

cod_pedido	data_pedido	data_entrega	cod_cliente	cod_cliente	nome_cliente	cpf	data_nasc	cep	numero	complemento
1	2019-01-01	2019-01-04	1	1	Marcos Costa de Sousa	12345678901	1981-02-06	6768100	1525	aoto 166C
2	2019-01-05	2019-01-07	2	2	Zoroastro Zoando	01987654321	1989-06-15	6757190	250	
3	2019-01-12	2019-01-15	3	3	Idelbrandolâncio Silva	54698721364	1974-09-27	6753001	120	
4	2019-01-15	2019-01-16	5	5	Coneunda Prado	54781269501	1950-10-06	6753020	50	aoto 166C
5	2019-01-20	2019-01-22	7	7	Iscrência da Silva	12457965823	1974-11-25	6803040	5	
NULL	NULL	NULL	NULL	4	Cosmólio Ferreira	41368529687	1966-12-01	6753020	25	aoto 255 F
NULL	NULL	NULL	NULL	6	Broundes Asmônio	41256398745	1940-05-10	6753400	100	
NULL	NULL	NULL	NULL	8	Zizafânio Zizundo	54123698562	1964-08-14	6803140	25	
NULL	NULL	NULL	NULL	9	Ricuerda Zunda	21698534589	1934-10-14	6803045	123	
NULL	NULL	NULL	NULL	10	Aninoado Zinzão	25639856971	1976-12-25	6803070	50	

## 20.4 Exercício prático

Elaborem um banco de dados que contemple situação problema por vocês, seguindo os seguintes critérios:

- Grupo de até 4 pessoas;
- Fazer o DER (Diagrama de Entidade e Relacionamento)
- Fazer o banco físico em SQL;
- “Popular” as tabelas de forma que se consiga tirar informações;
- Realizar pelo menos 15 *queries* para extração de informações desse banco.

Ao final o mesmo deverá ser apresentado com a solução criada.

## 21. VIEWS (Visualizações).

### 21.1 Introdução.

“Views” também conhecidas em português como visualizações nada mais são do que tabelas virtuais que são criadas a partir de seleções de dados, essas tabelas virtuais reúnem apenas os campos de uma ou mais tabelas físicas que são listados em uma instrução “SELECT” associada a “VIEW”, diferente das tabelas físicas “reais” que são armazenadas em disco e são

permanentes, ou seja se o servidor de dados for reinicializado tanto as tabelas quanto o seus dados permanecerão em disco as “**VIEWS**” são, como já foi dito virtuais, logo seus dados permanecem armazenados na memória principal do servidor de dados, isso traz algumas vantagens a mais óbvia é o acesso extremamente rápido a esses dados, logo podemos afirmar que as “**VIEWS**” são um recurso altamente recomendado em casos onde essas seleções de dados são muito requisitadas o caso mais comum de aplicação é em relatórios, e sendo consideradas como “tabelas” tudo que vimos em DQL pode ser usado em VIEWS.

Abaixo podemos ver a sintaxe de como criar uma “**view**”:

**CREATE VIEW** <NOME\_DA\_VIEW> **AS** <CONSULTA>;

Para praticarmos um pouco vamos criar três “**views**” em nossa base de dados “**VENDAS**” é necessário que a massa de dados que pedi no início do semestre esteja feita. Uma vez feito isso vamos criar três “**views**”, observe as imagens abaixo:

A primeira “**view**” lista clientes e pedidos:

```
#1ª VIEW - RELAÇÃO DE CLIENTE COM PEDIDO
create view vw_cliped as
  select c.cod_cliente as codigo, c.nome_cliente as nome, p.cod_pedido as pedido,
         p.data_pedido as data_requisicao
  from tbl_cliente c, tbl_pedido p
  where c.cod_cliente = p.cod_cliente;
```

A segunda “**view**” lista clientes, pedidos e itens dos pedidos:

```
#2ª VIEW - RELACIONA CLIENTES, PEDIDOS E ITEM DOS PEDIDOS
create view vw_clipedprod as
  select c.cod_cliente as codigo, c.nome_cliente as nome, p.cod_pedido as pedido,
         p.data_pedido as data_requisicao,
         i.i_cod_produto as produto, i.qtde
  from tbl_cliente c, tbl_pedido p, tbl_itempedido i
  where c.cod_cliente = p.cod_cliente
         and i.i_cod_pedido = p.cod_pedido;
```

A terceira “**view**” lista clientes, pedidos, itens dos pedidos e descrição dos produtos:

```
#3ª VIEW - RELACIONA CLIENTES, PEDIDOS, ITENS DOS PEDIDOS E PRODUTOS
create view vw_prodtudototal as
  select c.cod_cliente as codigo, c.nome_cliente as nome, p.cod_pedido as pedido,
         p.data_pedido as data_requisicao,
         i.i_cod_produto as produto, pr.nome_produto as descricao, i.qtde,
         pr.valor, i.qtde * pr.valor as totalcomprado
  from tbl_cliente c, tbl_pedido p, tbl_itempedido i, tbl_produto pr
  where c.cod_cliente = p.cod_cliente
         and i.i_cod_pedido = p.cod_pedido
         and i.i_cod_produto = pr.cod_produto;
```

Para visualização de “**VIEWS**” usamos o comando SELECT convencional, e para apagar uma “**VIEW**” usamos o comando DROP VIEW.

## 21.2 VIEWS com posição consolidada

Uma VIEW consolidada nada mais é que uma consulta que agrupa informações de vários registros de uma só vez, bastante utilizada para relatórios.

A view irá mostrar os produtos dos pedidos já calculados (quantidade X valor unitário), vejam o seguinte exemplo para explicar:



```
create view vw_consolidavenda (cod_pedido, cod_produto, valor) as
select p.cod_pedido,
       i.i_cod_produto, i.qtde * pr.valor
from   tbl_pedido p, tbl_itempedido i, tbl_produto pr
where  i.i_cod_pedido = p.cod_pedido
       and i.i_cod_produto = pr.cod_produto;
```

Na sequencia se fizermos um *SELECT* nesta view teremos:

	COD_PEDIDO	COD_PRODUTO	VALOR
	11	1	22.50
	11	2	9.60
	11	3	2.50
	12	4	17.50
	13	5	18.00

Outro exemplo seria agrupar o total dos pedidos, vejam:

```
create view vw_vendatotal (cod_pedido, total_valor) as
select p.cod_pedido, sum(i.qtde * pr.valor)
from   tbl_pedido p, tbl_itempedido i, tbl_produto pr
where  i.i_cod_pedido = p.cod_pedido
       and i.i_cod_produto = pr.cod_produto
group by p.cod_pedido;
```

Foram agrupados os pedidos com seus respectivos totais.

### Exercícios para fixação:

De acordo com o DER já confeccionado para nossa aula, criem as seguintes views:

- Que enxerguem os dados do cliente (código e nome) e pedidos (número do pedido, data do pedido e data de entrega), onde a data do pedido seja superior a 30/01/2014;
- Que enxerguem os dados do cliente (código do cliente e nome), dados do pedido (código do pedido, data do pedido e data da entrega), os dados do item do pedido (quantidade e código do produto), onde a quantidade destes produtos sejam maiores de 25;
- Que enxerguem os dados do pedido (código do pedido, código do cliente), os dados do item do pedido (quantidade, código do produto e descrição do produto);
- Que enxerguem os produtos reajustados em 11,2 %, onde deverá ser mostrado o código e a descrição do produto, o valor atual e o valor reajustado.

## 22. Triggers (Gatilhos)

### 22.1 Introdução

Uma “**trigger**” também conhecida em português como “**gatilho**” é uma sub-rotina alocada dentro de uma base de dados assim como os procedimentos armazenados (**stored procedure**), porem diferentes desses que precisam ser chamados para serem executados uma “**trigger**” é executada automaticamente (**disparada**) perante uma ação sofrida em uma tabela especifica dentro da base de dados onde a “**trigger**” foi criada, esse disparo pode ocorrer antes ou depois da ação em questão, logo podemos afirmar que “**triggers**” são criadas em um determinado banco de dados ficando associadas a uma tabela especifica e que sua execução depende de uma ação especifica na tabela em questão podendo a execução da “**trigger**” ser antes ou depois da ação que tabela sofre.

A sintaxe de um “**trigger**” pode ser observada abaixo:

**DELIMITER** [define um caractere delimitador]

**CREATE TRIGGER** <nome> <momento> <evento> ON <tabela>

## FOR EACH ROW

### BEGIN

<ação a ser executada pela trigger>

### END

Para tornar o entendimento mais fácil vamos criar uma **“trigger”** na base de dados **“BD\_VENDAS”**, a ideia da mesma é que sempre que um registro na tabela **“TBL\_CLIENTE”** for inserido será gravado na tabela chamada de **“TBL\_LOG”** o usuário logado no banco de dados, assim como a data e a hora em que a inclusão ocorreu para que esses dados possam ser usados posteriormente para uma possível auditoria.

Para tanto vamos antes de qualquer coisa criar a tabela **“TBL\_LOG”** que vai armazenar esses dados de auditoria a imagem abaixo traz o código de criação da tabela:

```
• create table tbl_log(
    id_log int not null auto_increment primary key,
    usuario varchar(50) not null,
    dt_log date not null,
    hora time not null
);
```

Agora vamos criar a **“trigger”**, essa por sua vez deve saber o momento em que deve ser executada sendo esses antes (**BEFORE**) ou depois (**AFTER**) de uma ação, além disso, devemos também definir em qual tabela a **“trigger”** vai atuar nesse caso, a tabela **“TBL\_CLIENTE”** e, seguida devemos definir qual ação será responsável pela execução da **“trigger”** no caso uma ação de **“DELETE”**, porém vale ressaltar aqui que poderia ser qualquer outra ação que a tabela possa vir a sofrer, observe a imagem abaixo:

```
• delimiter $
  create trigger trg_log before delete
  on tbl_cliente
  for each row
  begin
    insert into tbl_log
      (usuario, dt_log, hora)
    values (user(), curdate(), curtime());
  end $
```

## 22.2. Exibindo triggers.

Assim como os procedimentos armazenados também podemos listar as **“triggers”** de nossa base de dados, observe as imagens abaixo:

```
show triggers from bd_vendas;
```

Lista todas as **“triggers”** de uma base de dados específica, nesse caso a base de dados **“BD\_VENDAS”** e novamente o usuário que está operando deve ter esse direito.

### Excluindo triggers.

Assim como os procedimentos armazenados nos podemos também excluir **“triggers”** que tenham sido criadas, observe a imagem abaixo:

```
drop triggers trg_log;
```

**Exercícios para fixação:**

- a) Modifique a tabela tbl\_log acrescentando um campo onde armazene o tipo de operação realizada, sendo: "INSERÇÃO", "ATUALIZAÇÃO" ou "EXCLUSÃO" e outro campo que armazene a tabela que está sendo realizadas as ações.
- b) De acordo com o exercício A crie uma trigger que ao atualizar e antes de qualquer ação na tabela de Pedidos;
- c) De acordo com o exercício A crie uma trigger que ao excluir e antes de qualquer ação na tabela de Produtos;
- d) De acordo com o exercício A crie uma trigger que ao inserir e depois de qualquer ação na tabela de Clientes.

Essa atividade deverá ser desenvolvida em duplas e apresentada até a próxima aula.