

Modelagem de Dados e SQL

UNIDADE 1 - INTRODUÇÃO AO BANCO DE DADOS	5
1.1 INTRODUÇÃO	5
UNIDADE 2 - CONCEITOS E ARQUITETURAS - SGBD.....	6
2.1 AS LINGUAGENS PARA MANIPULAÇÃO DE DADOS:	6
2.2 CLASSIFICAÇÃO DOS SGBDs	6
2.3 O MODELO DE REDE 3-3-1	6
2.4 O MODELO HIERÁRQUICO 3-3-2:	7
2.5 O MODELO RELACIONAL 3-3-3:	7
UNIDADE 3 - MODELAGEM DE DADOS UTILIZANDO O MODELO ENTIDADE RELACIONAMENTO (ER)	10
3.1 ELEMENTOS DO MODELO ENTIDADE-RELACIONAMENTO	10
3.2 TIPOS E INSTÂNCIAS DE RELACIONAMENTO:	12
3.3 RESUMO DOS OBJETOS GRÁFICOS:.....	16
3.4 MODELO ENTIDADE RELACIONAMENTO ESTENDIDO:	17
3.5 ESPECIALIZAÇÃO	18
3.6 GENERALIZAÇÃO:.....	19
3.7 “LATTICE” OU MÚLTIPLA HERANÇA:	21
3.8 DICAS PARA A ELABORAÇÃO DE UM DIAGRAMA E-R:.....	22
UNIDADE 4 - O MODELO RELACIONAL	24
4.1 O MODELO RELACIONAL.....	24
4.2 DOMÍNIOS, TUPLAS, ATRIBUTOS E RELAÇÕES.....	24
4.3 ATRIBUTO CHAVE DE UMA RELAÇÃO	24
4.4 CHAVE PRIMÁRIA	26
4.5 CHAVE CANDIDATA	27
4.6 CHAVE ESTRANGEIRA	27
4.7 MAPEAMENTO DO MODELO ENTIDADE RELACIONAMENTO PARA O MODELO RELACIONAL	27
4.8 NORMALIZAÇÃO DE DADOS	31
UNIDADE 5 - SQL (STRUCTURED QUERY LANGUAGE)	35
5.1 INTRODUÇÃO	35
5.2 DML (DATA MANIPULATION LANGUAGE):.....	35
5.3 DCL (DATA CONTROL LANGUAGE):	35
5.4 TIPOS DE DADOS:	36
5.5 EXPRESSÕES E OPERADORES:	36
5.6 TIPOS DE OPERADORES	37
UNIDADE 6 - CRIAÇÃO DE TABELAS.....	39
6.1 TUTORIAL:.....	39
6.2 CRIANDO TABELAS	39
6.3 PARÂMETROS DE CAMPOS NA CRIAÇÃO DE TABELAS:	40
6.4 VALORES NULOS	40
6.5 CHAVE PRIMÁRIA	40
6.6 CHAVE ESTRANGEIRA:	41
6.7 INTEGRIDADE REFERENCIAL:	41
6.8 ON UPDATE:.....	41
6.9 ON DELETE:	42
6.10 REMOVENDO TABELAS.....	43
UNIDADE 7 - COMANDOS PARA ALTERAR TABELAS	44
7.1 ALTERAÇÃO DE TABELAS CRIADAS.....	44

7.2	ADICIONAR COLUNAS A UMA TABELA – ADD.....	44
7.3	REMOVER COLUNAS DE UMA TABELA – DROP.....	44
7.4	RENAMEANDO COLUNAS DE UMA TABELA – RENAME	44
7.5	MODIFICANDO COLUNAS DE UMA TABELA – MODIFY	44
UNIDADE 8 -	MODIFICANDO DADOS.....	46
8.1	INSERINDO DADOS.....	46
8.2	INSERINDO DADOS POR MEIO DE UM SELECT	46
8.3	EXCLUINDO INFORMAÇÕES	47
8.4	ATUALIZANDO DADOS DE UMA TABELA.....	47
UNIDADE 9 -	CONSULTA EM UMA ÚNICA TABELA SQL.....	49
9.1	SINTAXE BÁSICA DE CONSULTAS SQL	49
9.2	EXEMPLOS DE CONSULTAS SQL.....	49
9.3	SELEÇÃO DE TODOS OS REGISTROS COM COLUNAS ESPECÍFICAS.....	49
9.4	REDEFININDO O NOME DAS COLUNAS	50
9.5	SELEÇÃO DE REGISTROS COM EXIBIÇÃO DE STRINGS	51
UNIDADE 10 -	UNIDADE 10: FUNÇÕES COM STRINGS	53
10.1	FUNÇÕES COM STRINGS.....	53
10.2	SINTAXE DA FUNÇÃO DE CONCATENAÇÃO DE STRINGS	53
10.3	CONTROLE DE MAIÚSCULAS E MINÚSCULAS	53
10.4	SEGMENTANDO UMA STRING.....	54
UNIDADE 11 -	CRITÉRIOS DE CONSULTA – CLÁUSULA WHERE	56
11.1	CLÁUSULA WHERE.....	56
11.2	CONSULTAS SIMPLES COM CLÁUSULA WHERE	57
11.3	SELECIONANDO DADOS NÃO REPETIDOS	58
11.4	SELECIONANDO ALGUNS DADOS.....	59
UNIDADE 12 -	PREDICADOS	62
12.1	PREDICADOS LIKE E NOT LIKE:.....	62
12.2	PREDICADOS BETWEEN... AND E NOT BETWEEN... AND	63
12.3	PREDICADOS IN E NOT IN:.....	63
12.4	PREDICADO IS NULL E IS NOT NULL.....	64
12.5	SELEÇÃO COM OPERADORES LÓGICOS	65
UNIDADE 13 -	FUNÇÕES DE AGRUPAMENTO E ORDENAÇÃO.....	66
13.1	OBTENDO INFORMAÇÕES ESTATÍSTICAS	66
13.2	AGRUPAMENTO UTILIZANDO GROUP BY:.....	66
13.3	UTILIZANDO WHERE E GROUP BY:	67
13.4	SELEÇÃO DE REGISTROS COM AGRUPAMENTO PELA CLÁUSULA GROUP BY E HAVING:	68
13.5	ORDENAÇÃO DAS CONSULTAS	68
UNIDADE 14 -	CONSULTAS EM MÚLTIPLAS TABELAS	70
14.1	CONSULTAS EM MÚLTIPLAS TABELAS	70
14.2	JOIN.....	70
14.3	EQUIJOIN – JOIN DE IGUALDADE	71
14.4	EXERCÍCIOS SUGERIDOS	72
14.5	EQUIJOINS E OPERADORES LÓGICOS:.....	72
14.6	EQUIJOINS ENTRE MAIS DE DUAS TABELAS.....	73
14.7	REDEFINIÇÃO DO NOME DE TABELAS	73
14.8	OUTROS TIPOS DE JOINS - OUTER JOINS.....	73

14.9	OUTROS TIPOS DE JOINS - SELF JOINS	74
UNIDADE 15 -	TRABALHANDO COM DATAS.....	76
15.1	DATAS EM MYSQL.....	76
15.2	RECUPERANDO INFORMAÇÕES ESPECÍFICAS DE DATA.....	76
UNIDADE 16 -	PRINCIPAIS BANCOS DE DADOS	77
UNIDADE 17 -	MYSQL	78
17.1	INTRODUÇÃO	78
UNIDADE 18 -	O FRONT-END	79
18.1	ALGUNS EXEMPLOS.....	79
UNIDADE 19 -	CONECTANDO AO BANCO DE DADOS.....	80
19.1	ATRAVÉS DE LINGUAGENS DE PROGRAMAÇÃO	80
19.2	ATRAVÉS DO FRONT-END.....	81
UNIDADE 20 -	CONHECENDO A INTERFACE.....	84
20.1	PHPMYADMIN.....	84
20.2	MYSQL QUERY BROWSER	84
20.3	ACESSANDO UMA BASE DE DADOS.....	85
UNIDADE 21 -	CRIANDO UMA NOVA BASE DE DADOS	87
21.1	INTERFACE E COMANDO.....	87
UNIDADE 22 -	CRIANDO TABELAS	88
22.1	INTERFACE	88
UNIDADE 23 -	CRIANDO CONSULTAS	91
23.1	INTERFACE	91
UNIDADE 24 -	EXERCÍCIOS EXTRAS	93

Unidade 1 - Introdução ao Banco de Dados

1.1 Introdução

A tecnologia aplicada aos métodos de armazenamento de informações vem crescendo e gerando um impacto cada vez maior no uso de computadores, em qualquer área em que os mesmos podem ser aplicados.

Um “Banco de dados” pode ser definido como um conjunto de “dados” devidamente relacionados. Por “dados” podemos compreender como “fatos conhecidos” que podem ser armazenados e que possuem um significado implícito. Porém, o significado do termo “banco de dados” é mais restrito que simplesmente a definição dada acima. Um banco de dados possui as seguintes propriedades:

Banco de dados é uma coleção lógica coerente de dados com um significado inerente; uma disposição desordenada dos dados não pode ser referenciada como um banco de dados;

Banco de dados é projetado, construído e populado com dados para um propósito específico; um banco de dados possui um conjunto pré-definido de usuários e aplicações;

Banco de dados representa algum aspecto do mundo real, o qual é chamado de “minimundo”; qualquer alteração efetuada no minimundo é automaticamente refletida no banco de dados.

Banco de dados pode ser criado e mantido por um conjunto de aplicações desenvolvidas especialmente para esta tarefa ou por um “Sistema Gerenciador de Banco de Dados” (SGBD). Um SGBD permite aos usuários criarem e manipularem bancos de dados de propósito gerais. O conjunto formado por um banco de dados mais as aplicações que manipulam o mesmo é chamado de “Sistema de Banco de Dados”.

Banco de Dados: é uma coleção de dados inter-relacionados, representando informações sobre um domínio específico.

Exemplos: lista telefônica, controle do acervo de uma biblioteca, sistema de controle dos recursos humanos de uma empresa.

Sistema de Gerenciamento de Bancos de Dados (SGBD): é um software com recursos específicos para facilitar a manipulação das informações dos bancos de dados e o desenvolvimento de programas aplicativos.

Unidade 2 - Conceitos e Arquiteturas - SGBD

2.1 As Linguagens para Manipulação de Dados:

Para a definição dos esquemas conceituais e internos pode-se utilizar uma linguagem chamada DDL (Data Definition Language - Linguagem de Definição de Dados). O SGBD possui um compilador DDL que permite a execução das declarações para identificar as descrições dos esquemas e para armazená-las no catálogo do SGBD. A DDL é utilizada em SGBDs onde a separação entre os níveis internos e conceituais não é muito clara.

Em um SGBD em que a separação entre os níveis conceitual e interno são bem claras, é utilizado outra linguagem, a SDL (Storage Definition Language - Linguagem de Definição de Armazenamento) para a especificação do esquema interno. A especificação do esquema conceitual fica por conta da DDL.

Em um SGBD que utiliza a arquitetura três esquemas, é necessária a utilização de mais uma linguagem para a definição de visões, a VDL (Vision Definition Language - Linguagem de Definição de Visões).

Uma vez que o esquema esteja compilado e o banco de dados esteja populado, usa-se uma linguagem para fazer a manipulação dos dados, a DML (Data Manipulation Language - Linguagem de Manipulação de Dados).

2.2 Classificação dos SGBDs

O principal critério para se classificar um SGBD é o modelo de dados no qual é baseado. A grande maioria dos SGBDs contemporâneos é baseada no modelo relacional, alguns em modelos conceituais e alguns em modelos orientados a objetos. Outras classificações são:

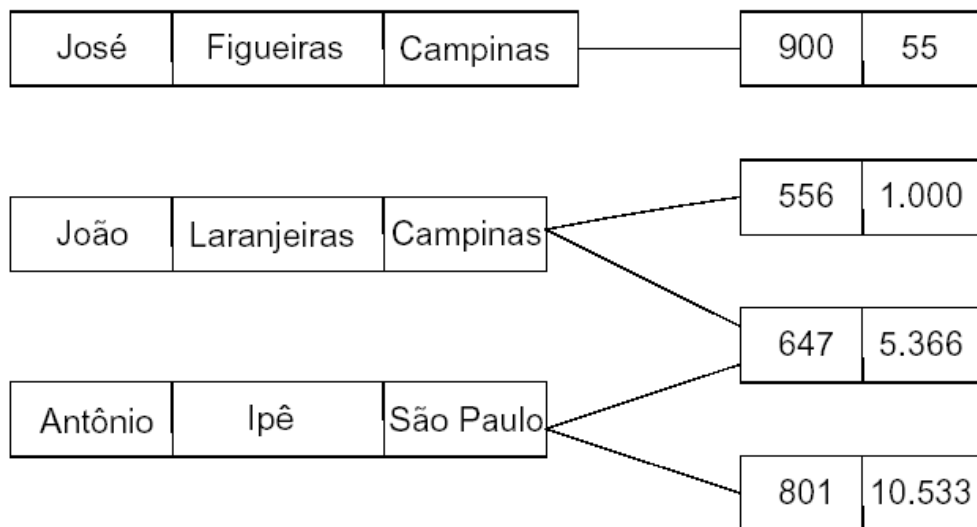
Usuários: um SGBD pode ser monousuário, comumente utilizado em computadores pessoais ou multiusuários, utilizado em estações de trabalho, minicomputadores e máquinas de grande porte;

Localização: um SGBD pode ser localizado ou distribuído; se ele for localizado, então todos os dados estarão em uma máquina (ou em um único disco) ou distribuído, onde os dados estarão distribuídos por diversas máquinas (ou diversos discos);

Ambiente: ambiente homogêneo é o ambiente composto por um único SGBD e um ambiente heterogêneo é o ambiente composto por diferentes SGBDs.

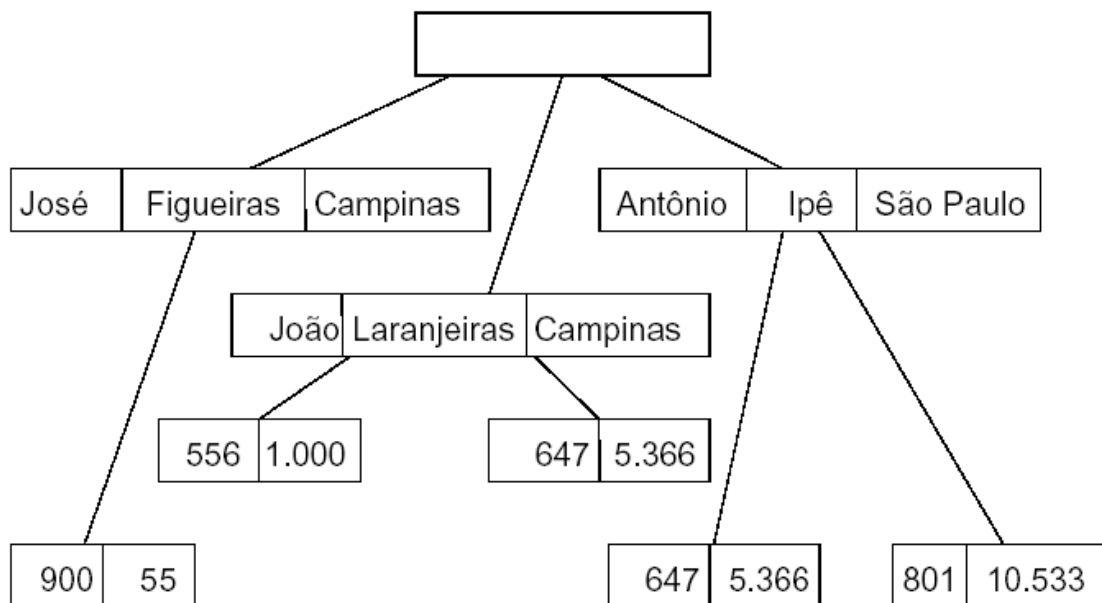
2.3 O Modelo de Rede 3-3-1

Os dados são representados por coleções de registros e os relacionamentos por elos.



2.4 O Modelo Hierárquico 3-3-2:

Os dados e relacionamentos são representados por registros e ligações, respectivamente. Os registros são organizados como coleções arbitrárias de árvores.



2.5 O Modelo Relacional 3-3-3:

Os meta dados são representados por tabelas, contendo linhas e colunas, onde são ligadas por colunas. Cada tabela contém uma ou mais colunas como identificadores.

Tabela Cliente (dados)

cód-cliente	nome	rua	cidade
015	José	Figueiras	Campinas
021	João	Laranjeiras	Campinas
037	Antônio	Ipê	São Paulo

Tabela Conta (dados)

nro-conta	saldo
900	55
556	1.000
647	5.366
801	10.533

Tabela Cliente-Conta
(relacionamento)

cód-cliente	nro-conta
015	900
021	556
021	647
037	647
037	801

Laboratório

Definir os seguintes termos:

Sistema de bancos de dados:

Banco de dados

Sistema de gerenciamento de banco de dados

Quais as vantagens e desvantagens da utilização de um sistema de banco de dados?

Descrever o modelo relacional de dados.

Definir os seguintes termos:

Linguagem de definição de dados;

Linguagem de manipulação de dados.

Unidade 3 - Modelagem de Dados Utilizando o Modelo Entidade Relacionamento (ER)

3.1 Elementos do Modelo Entidade-Relacionamento

Este é o modelo mais utilizado atualmente, devido principalmente, a sua simplicidade e eficiência. Baseiam-se na percepção do mundo real, que consiste em uma coleção de objetos básicos, chamados entidades e em relacionamentos entre esses objetos. Para identificar um modelo relacional, utilizamos o Diagrama de Estrutura de Dados.

Abaixo identificaremos todas as características do Modelo E-R

Entidade: objeto do mundo real, concreto ou abstrato e que possui existência independente.

O objeto básico tratado pelo modelo ER é a “entidade”. Uma entidade pode ser concreta, como uma caneta ou uma pessoa, ou abstrata, como um conceito ou uma sensação.

No banco de dados de uma empresa, por exemplo, são entidades: Funcionário, cliente, Departamento, etc. Cada entidade representa objetos com as mesmas características.

Um banco de dados, portanto, compreende uma coleção de conjuntos de entidades do mesmo tipo.

O símbolo que representa a entidade no modelo E-R é um retângulo como o nome da entidade escrito no seu interior, por exemplo:



FUNCIONARIO

Cada entidade possui um conjunto particular de propriedades que a descreve chamado “atributos”.

Atributos: São propriedades (características) que identificam as entidades. Cada entidade possui uma coleção de elementos de dados.

A cada atributo de uma entidade é associado um domínio de valores. Esses domínios podem ser um conjunto de números inteiros, números reais, cadeias de caracteres ou qualquer outro tipo de valor que o atributo pode assumir.

Os atributos são representados apenas pelo seu nome ligado à entidade por uma linha reta, por exemplo:

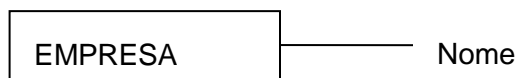


FUNCIONÁRIO

Nome

Podemos ter vários tipos de atributos: simples, composto, multivalorado e determinante. Vejamos as características de cada um deles:

Simples ou atômico: não possui qualquer característica especial. Por exemplo, o nome Da empresa é um atributo sem qualquer característica especial.



Composto: o seu conteúdo é formado por itens menores. O conteúdo de um atributo composto pode ser dividido em vários atributos simples.



Multivalorado: o seu conteúdo pode ser formado por mais de uma informação. É indicado colocando-se um asterisco procedendo ao nome do atributo, como no caso da empresa que pode possuir mais de um telefone.

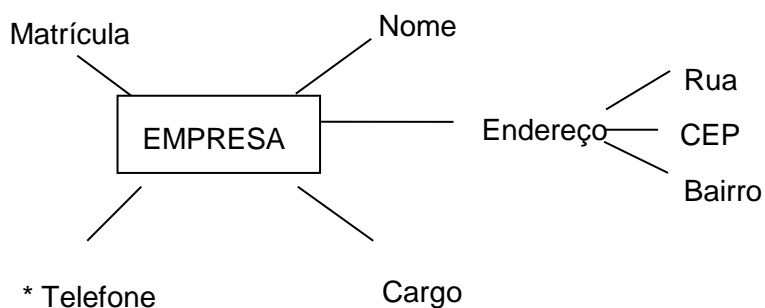


Determinante: o atributo determinante é aquele que define univocamente as instâncias de uma entidade, ou seja, é único para as instâncias de uma entidade. É indicado sublinhando-se o nome do atributo. No exemplo de uma empresa, o CNPJ é um atributo determinante, pois não podem existir duas empresas com o mesmo valor nesse atributo.



Vejamos um exemplo no qual se aplicam os conceitos vistos até aqui:

Uma empresa necessita armazenar os dados de seus funcionários, atualmente em um fichário. Os dados são a Matrícula, o Nome, o Endereço (Rua, CEP e Bairro), o Telefone (o funcionário pode ter mais de um) e o cargo.



Laboratório

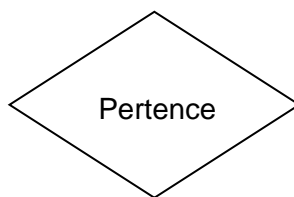
Identifique os atributos e o tipo do atributo nas seguintes entidades.

CLIENTES - FUNCIONARIOS - EMPRESA - PRODUTOS

3.2 Tipos e Instâncias de Relacionamento:

Além de conhecer detalhadamente os tipos entidade, é muito importante conhecer também os relacionamentos entre estes tipos entidades.

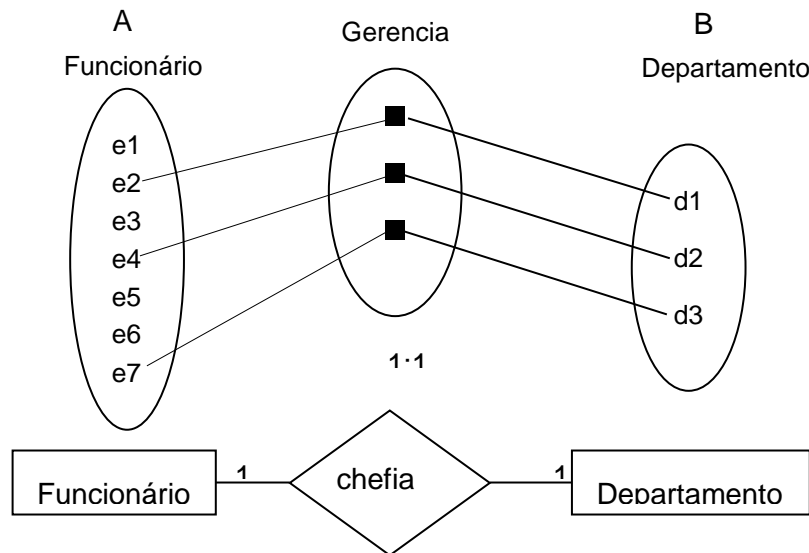
Relacionamento: é o tipo de ocorrência existente entre entidades. O Símbolo que representa o relacionamento no modelo E-R é um losango com o nome do relacionamento escrito no seu interior, como no exemplo a seguir:



Existem três tipos de relacionamentos entre entidades: um-para-um, um-para-vários e vários-para-vários.

Um-Para-Um: é quando uma entidade de A se relaciona com uma entidade de B.

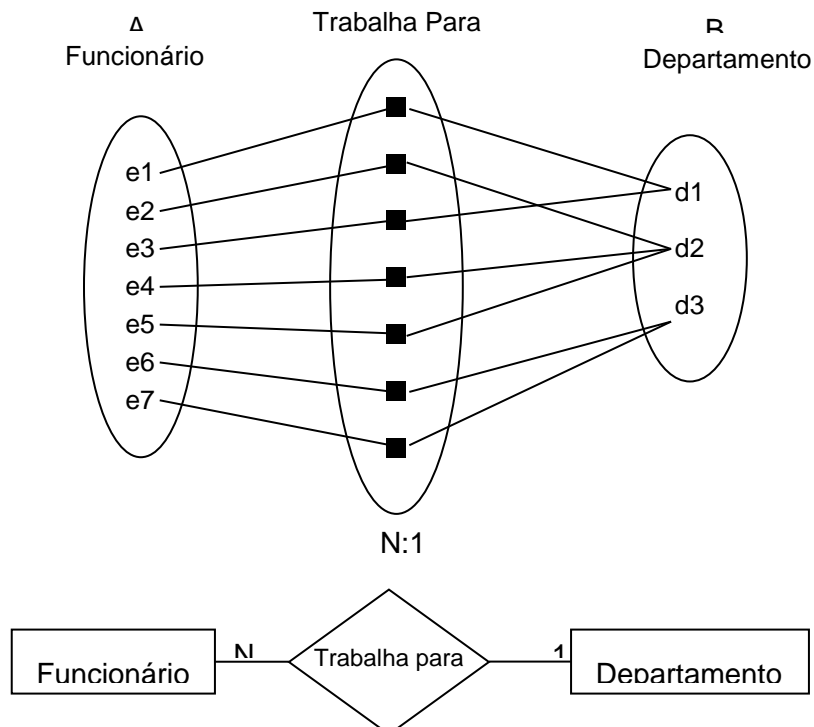
Como exemplo, podemos citar uma empresa que é dividida em departamentos e em cada departamento possui um único gerente e um funcionário-gerente só pode chefiar um único departamento.



Este é um caso particular, pois outra empresa poderia ter o mesmo gerente chefiando vários departamentos. Vale lembrar que os modelos de dados são utilizados num determinado contexto. Muda-se o contexto, muda-se também o modelo de dados.

Um-para-Vários: é quando cada entidade de (A) pode se relacionar com uma ou mais entidades de B. Quando se quer dizer uma ou mais de uma entidade, utiliza-se a letra N.

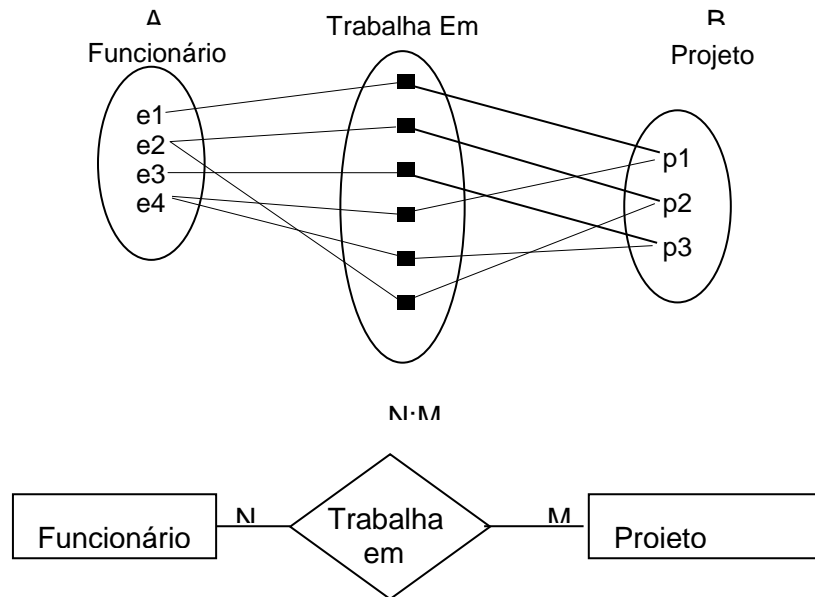
No exemplo, uma empresa dividida em departamentos, um funcionário só pode ser alocado em um único departamento, porém num departamento podem trabalhar vários funcionários.



Esta representação acaba resultando num relacionamento 1: N invertido.

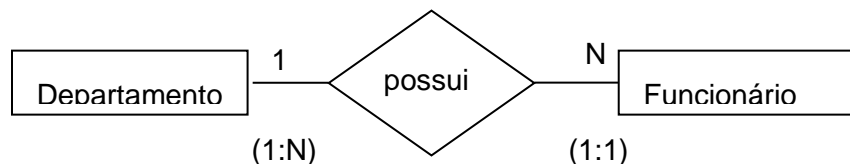
Vários-para-vários: é quando várias (N) entidade de A se relacionam com várias(M) entidade de B.

Exemplo para esse tipo de relacionamento é o que ocorre entre um funcionário e os projetos de uma empresa. Um funcionário pode trabalhar em vários projetos, e um projeto pode ser executado por mais de um funcionário.



Cardinalidade: define o número máximo de ocorrências em um relacionamento

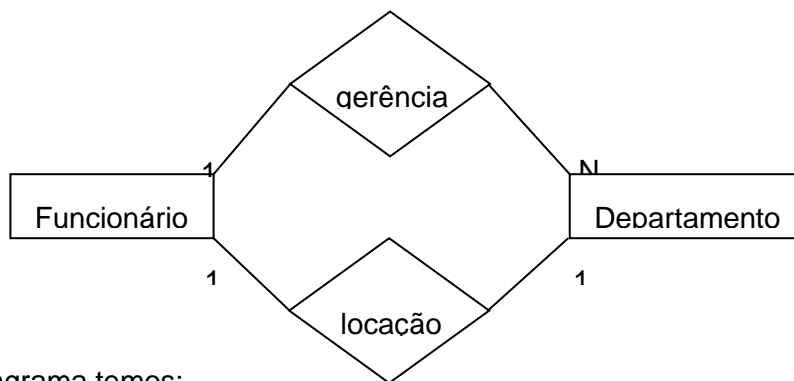
Para se determinar a cardinalidade, deve-se fazer a pergunta relativa ao relacionamento em ambas as direções. No exemplo a seguir, temos:



- Um departamento possui quantos funcionários? No mínimo 1 e no máximo N.
- Um funcionário está alocado em quantos departamentos? Em no mínimo 1 e no máximo 1.

No primeiro caso a cardinalidade é N, e no segundo é 1. Somando-se as cardinalidade, definimos o resultado final do relacionamento, observado nos modelos de dados (no caso 1: N).

Dois relacionamentos: uma entidade pode ter mais de um relacionamento com a outra entidade. A entidade departamento, por exemplo, pode ter uma relação de locação e outra de gerência com a entidade Funcionário. Nesse caso, existem dois relacionamentos entre as entidades.



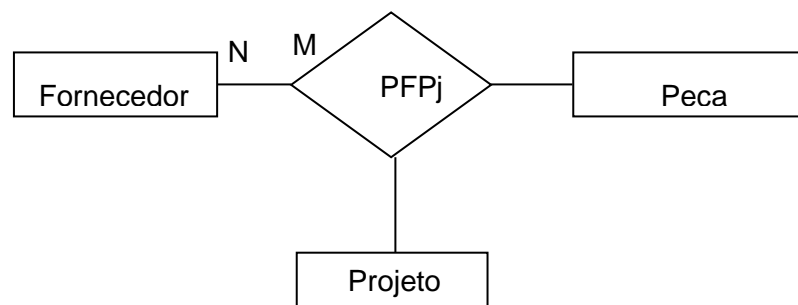
Traduzindo o diagrama temos:

- cada departamento da empresa possui vários (N) funcionários lotados nele.
- um departamento possui um único funcionário que ocupa o cargo de gerente.

Grau de um Relacionamento: O “grau” de um tipo relacionamento é o número de tipos entidade que participam do tipo relacionamento. O grau de um relacionamento é ilimitado, porém, a partir do grau três (ternário), a compreensão e a dificuldade de se desenvolver a relação corretamente se tornam extremamente complexas.

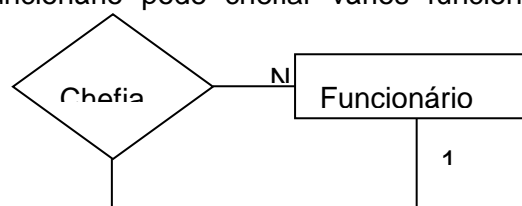
Relacionamento Ternário: alguns relacionamentos precisam ligar 3 entidades, como no exemplo que se segue.

Numa indústria, um fornecedor pode fornecer várias peças para vários projetos. Um projeto pode ter vários fornecedores para várias peças. E uma peça pode ter vários fornecedores para vários projetos.



Auto relacionamento: uma entidade pode se relacionar com ela mesma. Observe o exemplo:

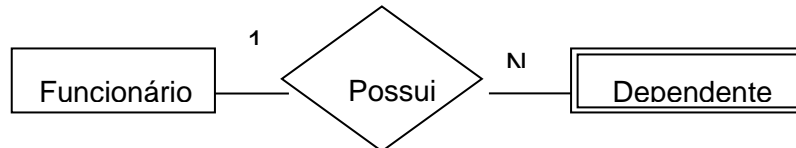
Numa empresa, um funcionário pode chefiar vários funcionários, porém ele também é um funcionário.



Entidade Dependente: uma entidade pode ter sua existência vinculada à existência de outra entidade.

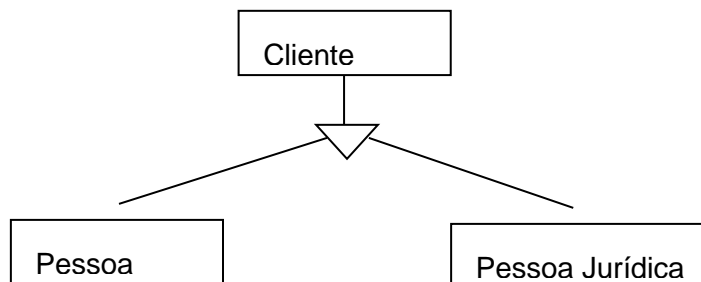
Uma empresa necessita armazenar os dados dos dependentes menores dos funcionários.

A entidade Dependente só existe porque existe a entidade Funcionário



O símbolo que representa a entidade dependente é um retângulo dentro de outro retângulo, com o nome da entidade escrito no seu interior.

Relacionamento É-Um (Generalização/Especialização): ocorre quando uma entidade com seus atributos englobam entidades especializadas com seus atributos específicos. No esquema a seguir, a entidade Cliente engloba seus atributos e também os atributos específicos da entidade Pessoa Física, assim como os de Pessoa Jurídica.

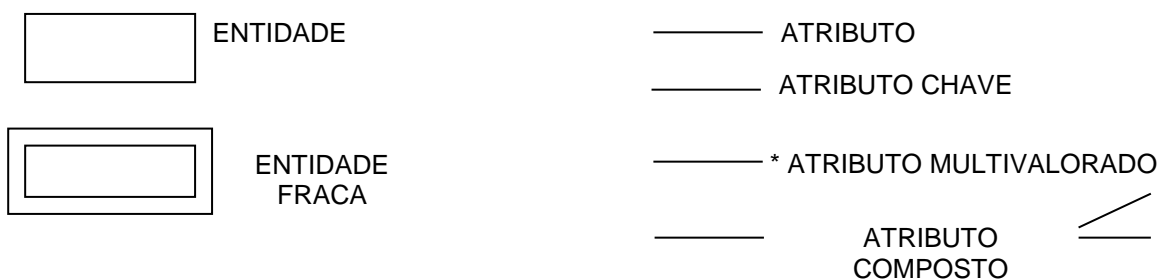


3.3 Resumo dos objetos gráficos:

O diagrama Entidade Relacionamento é composto por um conjunto de objetos gráficos que visa representar todos os objetos do modelo Entidade Relacionamento tais como entidades, atributos, atributos chaves, relacionamentos, restrições estruturais, etc.

O diagrama ER fornece uma visão lógica do banco de dados, fornecendo um conceito mais generalizado de como estão estruturados os dados de um sistema.

Os objetos que compõem o diagrama ER estão listados a seguir,



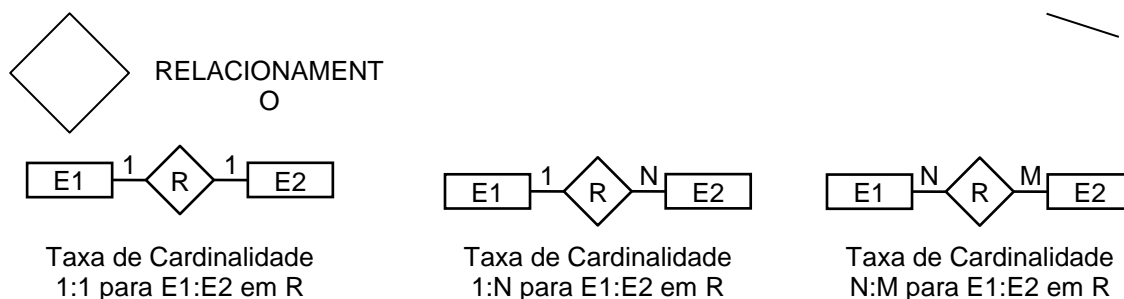


Figura - Objetos que Compõem o Diagrama ER

3.4 Modelo Entidade Relacionamento Estendido:

Os conceitos do modelo Entidade Relacionamento discutidos anteriormente são suficientes para representar logicamente a maioria das aplicações de banco de dados.

Porém, com o surgimento de novas aplicações, surgiu também a necessidade de novas semânticas para a modelagem de informações mais complexas. O modelo Entidade Relacionamento

Estendido (ERE) visa fornecer esta semântica para permitir a representação de informações complexas.

É importante frisar que embora o modelo ERE trate classes e subclasses, ele não possui a mesma semântica de um modelo orientado a objetos.

O modelo ERE engloba todos os conceitos do modelo ER mais os conceitos de subclasse, superclasse, generalização e especialização e o conceito de herança de atributos.

Subclasses, Superclasses e Especializações:

O primeiro conceito do modelo ERE que será abordado é o de subclasse de um tipo entidade. Como visto anteriormente, um tipo entidade é utilizado para representar um conjunto de entidades do mesmo tipo.

Em muitos casos, um tipo entidade possui diversos subgrupos adicionais de entidades que são significativas e precisam ser representadas explicitamente devido ao seu significado à aplicação de banco de dados. Leve em consideração o seguinte exemplo:

Para um banco de dados de uma empresa temos o tipo entidade empregado, o qual possui as seguintes características: nome, RG, CIC, número funcional, endereço completo (rua, número, complemento, CEP, bairro, cidade), sexo, data de nascimento e telefone (DDD e número); caso o (a) funcionário (a) seja um (a) engenheiro (a), então se deseja armazenar as seguintes informações: número do CREA e especialidade (Civil, Mecânico, Eletro/Eletrônico); caso o (a) funcionário (a) seja um (a) secretário (a), então se deseja armazenar as seguintes informações: qualificação (bi ou tri língua) e os idiomas no qual possui fluência verbal e escrita.

Se as informações número do CREA, especialidade, tipo e idiomas forem representadas diretamente no tipo entidade empregado estaremos representando informações de um conjunto limitados de entidades empregado para os todos os funcionários da empresa.

Neste caso, podemos criar duas subclasses do tipo entidade empregado: engenheiro e secretária, as quais irão conter as informações acima citadas. Além disto, engenheiro e secretária podem ter relacionamentos específicos.

Uma entidade não pode existir meramente como componente de uma subclasse. Antes de ser componente de uma subclasse, uma entidade deve ser componente de uma superclasse.

Isto leva ao conceito de herança de atributos; ou seja, a subclasse herda todos os atributos da superclasse. Isto porque a entidade de subclasse representa as mesmas características de uma mesma entidade da superclasse.

Uma subclasse pode herdar atributos de superclasses diferentes.

A figura abaixo mostra a representação diagramática do exemplo acima.

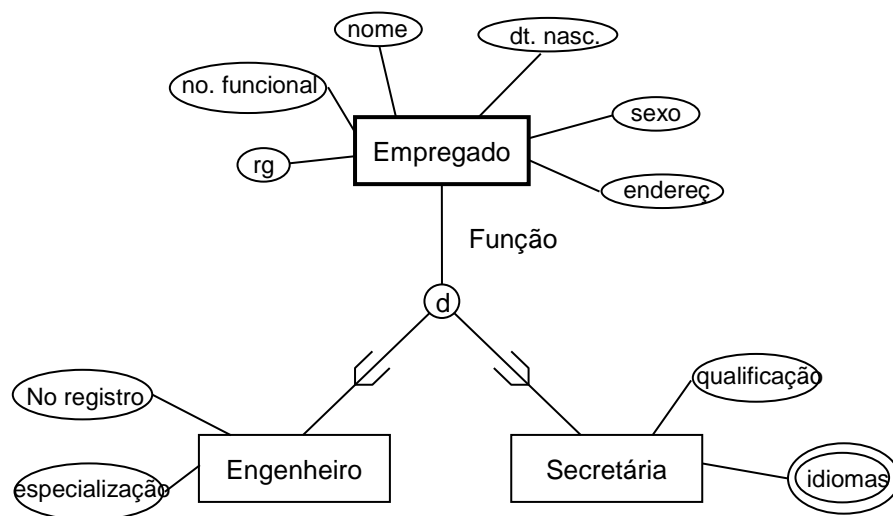


Figura - Representação de Superclasse e Subclasses

3.5 Especialização

Especialização é o processo de definição de um conjunto de classes de um tipo entidade; este tipo entidade é chamado de superclasse da especialização. O conjunto de subclasses é formado baseado em alguma característica que distingue as entidades entre si.

No exemplo da figura 12, temos uma especialização, a qual pode chamar de função. Veja agora no exemplo da figura 13, temos a entidade empregado e duas especializações.

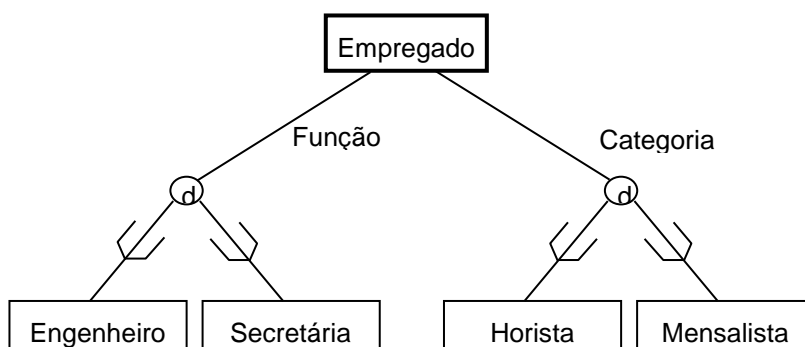


Figura - Duas Especializações para Empregado: Função e Categoria

Como visto anteriormente, uma subclasse pode ter relacionamentos específicos com outras entidades ou com a própria entidade que é a sua superclasse. Veja o exemplo da figura 14.

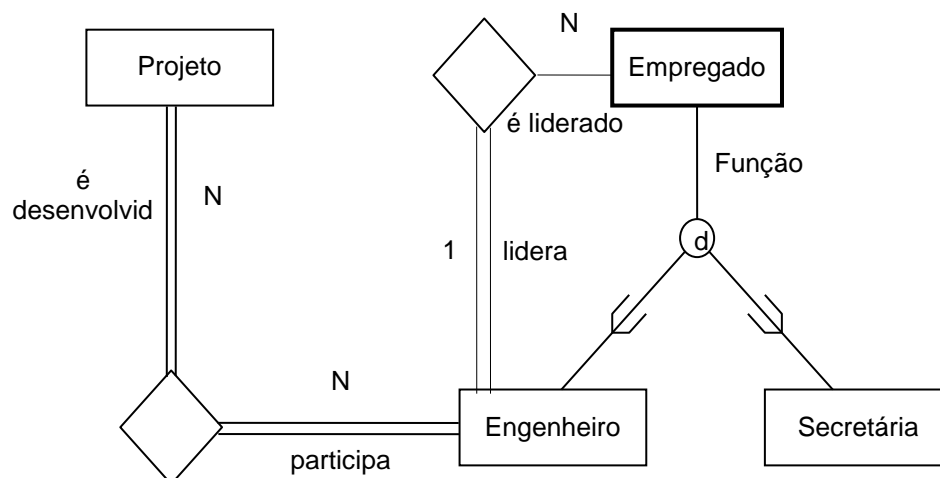


Figura - Relacionamentos Entre Subclasses e Entidades

O processo de especialização nos permite:

Definir um conjunto de subclasses de um tipo entidade;

Associar atributos específicos adicionais para cada subclasse;

Estabelecer tipos relacionamentos específicos entre subclasses e outros tipos entidades.

3.6 Generalização:

A generalização pode ser pensada como um processo de abstração reverso ao da especialização, no qual são suprimidas as diferenças entre diversos tipos entidades, identificando suas características comuns e generalizando estas entidades em uma superclasse.

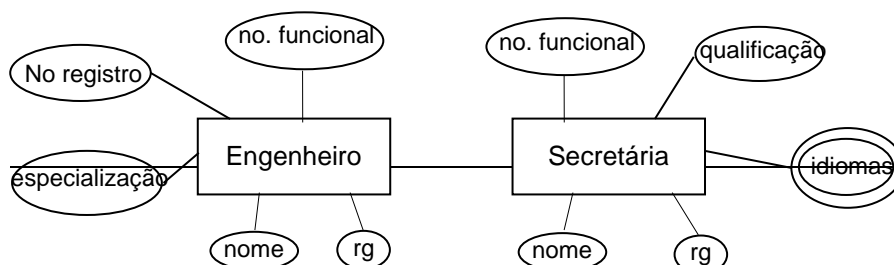


Figura - Tipos Entidades Engenheiro e Secretária

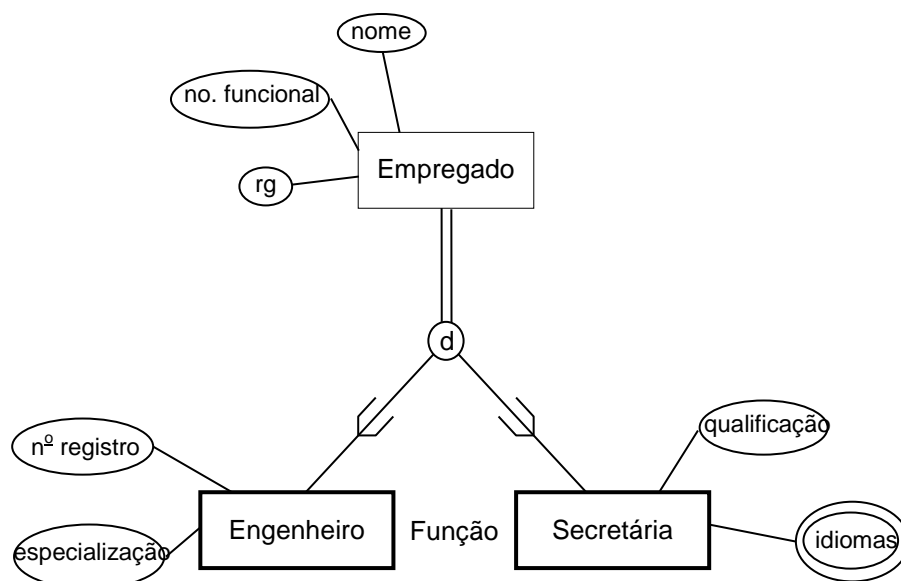


Figura - Generalização Empregado para os Tipos Entidades Engenheiro e Secretária

É importante destacar que existe diferença semântica entre a especialização e a generalização.

Na especialização, podemos notar que a ligação entre a superclasse e as subclasses é feita através de um traço simples, indicando participação parcial por parte da superclasse.

Analisando o exemplo da figura 12, é observado que um empregado não é obrigado a ser um engenheiro ou uma secretária. Na generalização, podemos notar que a ligação entre a superclasse e as subclasses é feita através de um traço duplo, indicando participação total por parte da superclasse. Analisando o exemplo da figura 16, é observado que um empregado é obrigado a ser um engenheiro ou uma secretária.

A letra d dentro do círculo que especifica uma especialização ou uma generalização significa disjunção.

Uma disjunção em uma especialização ou generalização indica que uma entidade do tipo entidade que representa a superclasse pode assumir apenas um papel dentro da mesma.

Analisando o exemplo da figura 13 temos duas especializações para a superclasse Empregado, as quais são restringidas através de uma disjunção. Neste caso, um empregado pode ser um engenheiro ou uma secretária e o mesmo pode ser horista ou mensalista.

Além da disjunção podemos ter um “overlap”, representado pela letra o. No caso do “overlap”, uma entidade de uma superclasse pode ser membro de mais que uma subclasse em uma especialização ou generalização. Analise a generalização no exemplo da figura 17. Suponha que uma peça fabricada em uma tornearia pode ser manufaturada ou torneada ou ainda, pode ter sido manufaturada e torneada.

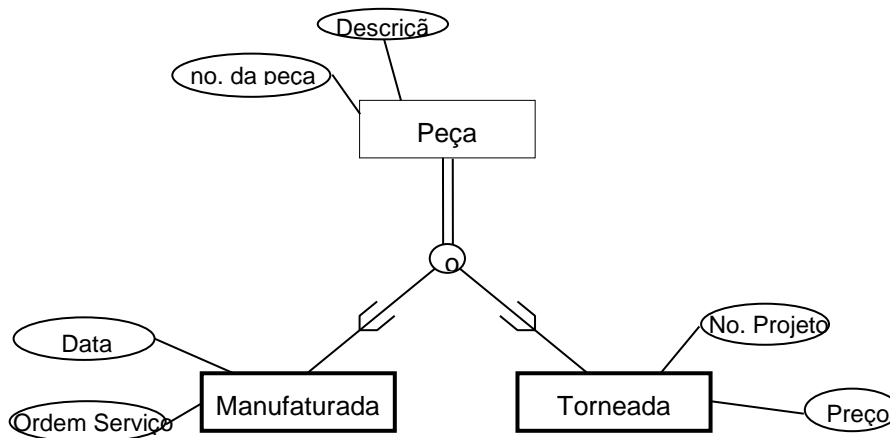


Figura - Uma Generalização com “Overlap”

3.7 “Lattice” ou Múltipla Herança:

Uma subclasse pode ser definida através de um “lattice”, ou múltipla herança, ou seja, ela pode ter diversas superclasses, herdando características de todas. Leve em consideração o seguinte exemplo:

Uma construtora possui diversos funcionários, os quais podem ser engenheiros ou secretárias. Um funcionário pode também ser assalariado ou horista. Todo gerente de departamento da construtora deve ser um engenheiro e assalariado.

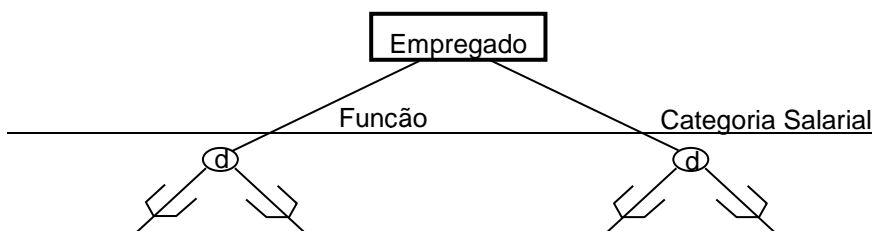


Figura 18 - Um “Lattice” com a Subclasse Gerente Compartilhada

Neste caso então, um gerente será um funcionário que além de possuir as características próprias de Gerente, herdará as características de Engenheiro e de Mensalista.

3.8 Dicas para a elaboração de um diagrama E-R:

A presença de um substantivo usualmente indica uma entidade;

A presença de um verbo é uma forte indicação de um relacionamento;

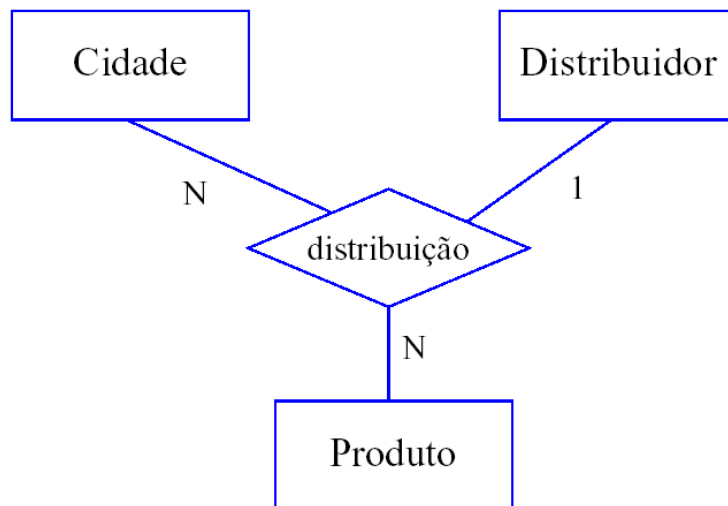
Um adjetivo, que é uma qualidade, é uma forte indicação de um atributo;

Um advérbio temporal, qualificando o verbo, é uma indicação de um atributo do relacionamento.

Laboratório

Etapas

- 1) Construa um diagrama E-R para um hospital com um conjunto de pacientes e um conjunto de médicos. Registros de diversos testes realizados são associados a cada paciente.
- 2) Construa um diagrama E-R para uma companhia de seguros de automóveis com um conjunto de clientes, onde cada um possui certo número de carros. Cada carro tem um número de acidentes associados a ele.
- 3) Explique a diferença entre uma entidade (conjunto entidade) e uma ocorrência (instância) de uma entidade.
- 4) O que é o papel de uma entidade em um relacionamento? Quando é necessário especificá-lo?
- 5) Construa um diagrama ER em que o conceito de entidade associativa seja utilizado.
- 6) Mostre como o diagrama abaixo pode ser representado apenas por relacionamentos binários.



7) Modifique o diagrama abaixo para especificar o seguinte:

- a) Um curso não pode estar vazio, isto é, deve possuir alguma disciplina em seu currículo.
- b) Um aluno, mesmo que não inscrito em nenhum curso, deve permanecer por algum tempo no banco de dados.
- c) Um aluno pode fazer mais de um curso.

8) Esboce o diagrama do exercício anterior na notação de Peter Chen, especificando as cardinalidades mínimas e máximas (min. Max).

Unidade 4 - O Modelo Relacional

4.1 O Modelo Relacional

Nesta unidade é apresentado o modelo relacional de forma teórica e a forma como o mesmo pode ser derivado do modelo Entidade-Relacionamento apresentado na unidade anterior. Assim, com o conteúdo visto na unidade anterior podemos modelar os dados de uma aplicação, através do modelo ER, e mapeá-los para o modelo relacional, de forma a implementar no banco de dados.

4.2 Domínios, Tuplas, Atributos e Relações

Um **domínio D** é um conjunto de valores atômicos, sendo que por atômico, podemos compreender que cada valor do domínio é indivisível. Durante a especificação do domínio é importante destacar o tipo, o tamanho e a faixa do atributo que está sendo especificado. Por exemplo:

Coluna	Tipo	Tamanho	Faixa
RG	Numérico	10,0	03000000-25999999
Nome	Caractere	30	a-z, A-Z
Salário	Numérico	5,2	00100,00-12999,99

Um **esquema de relação R**, denotado por $R(A_1, A_2, \dots, A_n)$, onde cada atributo A_i é o nome do papel desempenhado por um domínio **D** no esquema relação **R**, onde **D** é chamado domínio de A_i e é denotado por $dom(A_i)$. O grau de uma relação **R** é o número de atributos presentes em seu esquema de relação.

A **instância r** de um esquema relação denotado por $r(R)$ é um conjunto de n-tuplas:

$r = [t_1, t_2, \dots, t_n]$ onde os valores de $[t_1, t_2, \dots, t_n]$ devem estar contidos no domínio **D**. O valor **nulo** também pode fazer parte do domínio de um atributo e representa um valor não conhecido para uma determinada tupla.

4.3 Atributo Chave de uma Relação

Uma relação pode ser definida como um conjunto de tuplas distintas. Isto implica que a combinação dos valores dos atributos em uma tupla não pode se repetir na mesma tabela. Existirá sempre um subconjunto de atributos em uma tabela que garantem que não haverá valores repetidos para as diversas tuplas da mesma, garantindo que $t1[SC] \neq t2[SC]$.

SC é chamada de **superchave** de um esquema de relação. Toda relação possui ao menos uma superchave - o conjunto de todos os seus atributos. Uma **chave C** de um esquema de relação **R** é uma superchave de **R** com a propriedade adicional que removendo qualquer atributo **A** de **K**, resta ainda um conjunto de atributos **K'** que não é uma superchave de **R**. Uma chave é uma superchave da qual não se pode extrair atributos.

Por exemplo, o conjunto: *(RA, Nome, Endereço)* é uma superchave para estudante, porém, não é uma chave, pois se tirarmos o campo *Endereço* continuaremos a ter uma superchave. Já o conjunto *(Nome da Revista, Volume, Nº da Revista)* é uma superchave e uma chave, pois qualquer um dos atributos que retirarmos, deixará de ter uma superchave, ou seja, *(Nome da Revista, Volume)* não identifica uma única tupla.

Em outras palavras, uma superchave é uma **chave composta**, ou seja, uma chave formada por mais que um atributo.

Veja o exemplo abaixo:




Tabela DEPENDENTES				
RG Responsável	Nome Dependente	Dt. Nascimento	Relação	Sexo
10101010	Jorge	27/12/86	Filho	Masculino
10101010	Luiz	18/11/79	Filho	Masculino
20202020	Fernanda	14/02/69	Cônjuge	Feminino
20202020	Ângelo	10/02/95	Filho	Masculino

Quando uma relação possui mais que uma chave (não confundir com chave composta) - como, por exemplo, RG e CIC para empregados - cada uma destas chaves são chamadas de **chaves candidatas**. Uma destas chaves candidatas deve ser escolhida como **chave primária**.

Uma **chave estrangeira CE** de uma tabela **R₁** em **R₂** ou vice-versa, especifica um relacionamento entre as tabelas **R₁** e **R₂**.

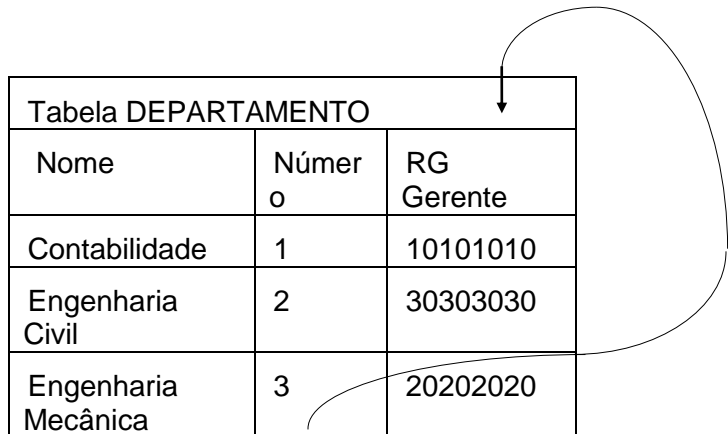


Tabela DEPARTAMENTO		
Nome	Número	RG Gerente
Contabilidade	1	10101010
Engenharia Civil	2	30303030
Engenharia Mecânica	3	20202020

Tabela EMPREGADO					
Nome	RG	CIC	Depto.	RG Supervisor	Salário
João Luiz	10101010	11111111	1	NULO	3.000,00
Fernando	20202020	22222222	2	10101010	2.500,00
Ricardo	30303030	33333333	2	10101010	2.300,00
Jorge	40404040	44444444	2	20202020	4.200,00
Renato	50505050	55555555	3	20202020	1.300,00

4.4 Chave Primária

É o atributo ou conjunto de atributos que identifica univocamente as tuplas (linhas) de uma tabela. Normalmente é o atributo determinante no modelo E-R e, da mesma forma, é sublinhado quando informado.

Muitas vezes, precisamos criar um campo chave, mesmo que a tabela tenha atributos determinantes.

Na tabela Fornecedor, criamos um campo determinante Código, denominado, portanto, chave primária (*PRIMARY KEY*).

4.5 Chave Candidata

É qualquer conjunto de atributos que consegue satisfazer ao critério de chave primária. Toda tabela tem pelo menos uma chave candidata. Quando existe mais de uma chave candidata, deve-se escolher para chave primária o campo que tenha a menor possibilidade de ter um valor nulo.

4.6 Chave Estrangeira

É o atributo de uma tabela que é chave primária de outra tabela.

A chave estrangeira (*FOREIGN KEY*) é utilizada sempre que uma entidade se relacionar com outra por meio de um relacionamento. Quando isso ocorre, o campo chave primária da tabela A é chave estrangeira na tabela B, ou vice-versa, dependendo o tipo de cardinalidade dos relacionamentos.

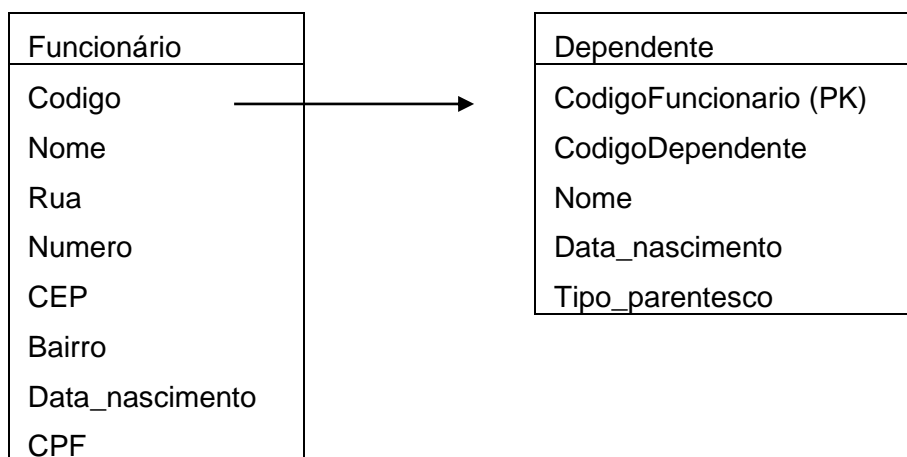
4.7 Mapeamento do Modelo Entidade Relacionamento para o Modelo Relacional

O mapeamento do modelo entidade relacionamento para o Modelo Relacional segue oito passos básicos, a saber:

1. Para cada entidade **E** no modelo ER é criada uma tabela **T₁** no Modelo Relacional que inclua todos os atributos simples de **E**; para cada atributo composto, são inseridos apenas os componentes simples de cada um; um dos atributos chaves de **E** deve ser escolhida como a chave primária de **T₁**;

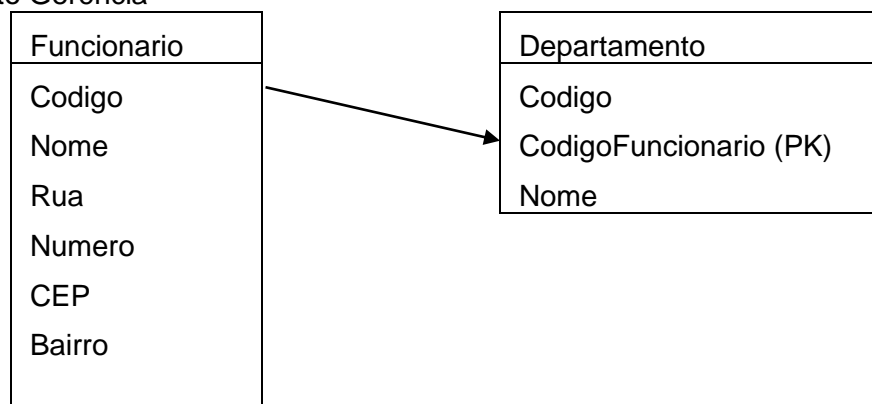
Funcionário
Código (PK)
Nome
Rua
Numero
CEP
Bairro
Data_nascimento
CPF

2. Para cada entidade fraca **EF** com entidade proprietária **E** no modelo ER, é criada uma tabela **T₁** no Modelo Relacional incluindo todos os atributos simples de **EF**; para cada atributo composto, são inseridos apenas os componentes simples de cada um; a chave primária desta relação **T₁** será composta pela chave parcial da entidade fraca **EF** mais a chave primária da entidade proprietária **E**;



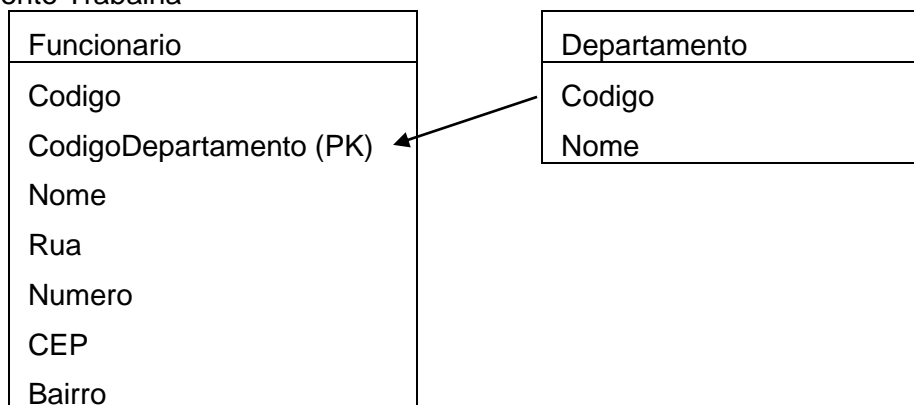
3. Para cada relacionamento regular com cardinalidade 1:1 entre entidades E_1 e E_2 que geraram as tabelas T_1 e T_2 respectivamente, devemos escolher a chave primária de uma das relações (T_1 , T_2) e inseri-la como chave estrangeira na outra relação; se um dos lados do relacionamento tiver participação total e outro parcial, então é interessante que a chave do lado com participação **parcial** seja inserida como chave estrangeira no lado que tem participação **total**;

Relacionamento Gerência

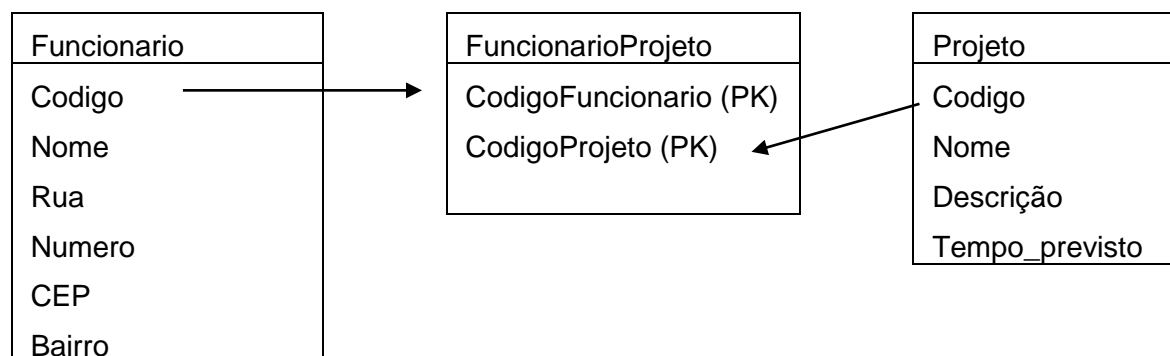


4. Para cada relacionamento regular com cardinalidade 1: N entre entidades E_1 e E_2 respectivamente e que geraram as tabelas T_1 e T_2 respectivamente, deve-se inserir a chave primária de T_1 como chave estrangeira em T_2 ;

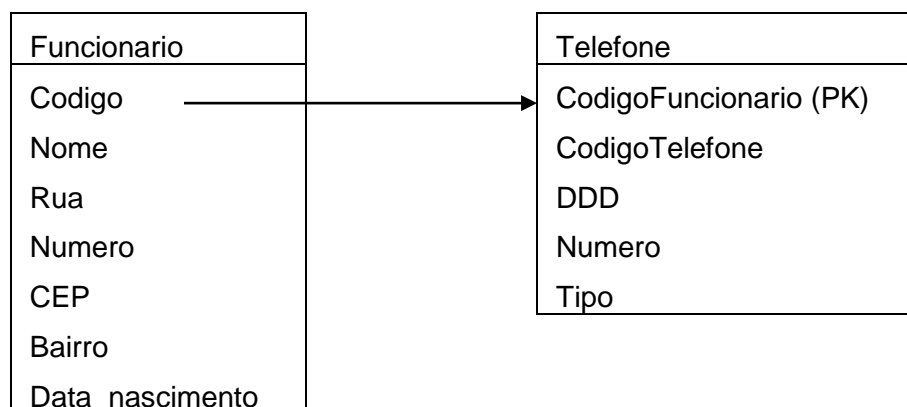
Relacionamento Trabalha



5. Para cada relacionamento regular com cardinalidade N: N entre entidades E_1 e E_2 , cria-se uma nova tabela T_1 , contendo todos os atributos do relacionamento mais o atributo chave de E_1 e o atributo chave de E_2 ; a chave primária de T_1 será composta pelos atributos chave de E_1 e E_2 ;



6. Para cada atributo multivalorado A_1 , cria-se uma tabela T_1 , contendo o atributo multivalorado A_1 , mais o atributo chave C da tabela que representa a entidade ou relacionamento que contém A_1 ; a chave primária de T_1 será composta por A_1 mais C ; se A_1 for composto, então a tabela T_1 deverá conter todos os atributos de A_1 ;



CPF

7. Para cada relacionamento n -ário, $n > 2$, cria-se uma tabela T_1 , contendo todos os atributos do relacionamento; a chave primária de T_1 será composta pelos atributos chaves das entidades participantes do relacionamento;

8. Converta cada especialização com m subclasses $\{S_1, S_2, \dots, S_m\}$ e superclasse SC , onde os atributos de SC são $\{c, a_1, a_2, \dots, a_n\}$ onde c é a chave primária de SC , em tabelas utilizando uma das seguintes opções:

8.1. Crie uma tabela T para SC com os atributos $A(T) = \{c, a_1, a_2, \dots, a_n\}$ e chave $C(T) = c$; crie uma tabela T_i para cada subclasse S_i , $1 \leq i \leq m$, com os atributos $A(T_i) = \{c\} \cup A(S_i)$, onde $C(T) = c$;

8.2. Crie uma tabela T_i para cada subclasse S_i , $1 \leq i \leq m$, com os atributos $A(T_i) = A(S_i) \cup \{c, a_1, a_2, \dots, a_n\}$ e $C(T_i) = c$;

8.3. Crie uma tabela T com os atributos

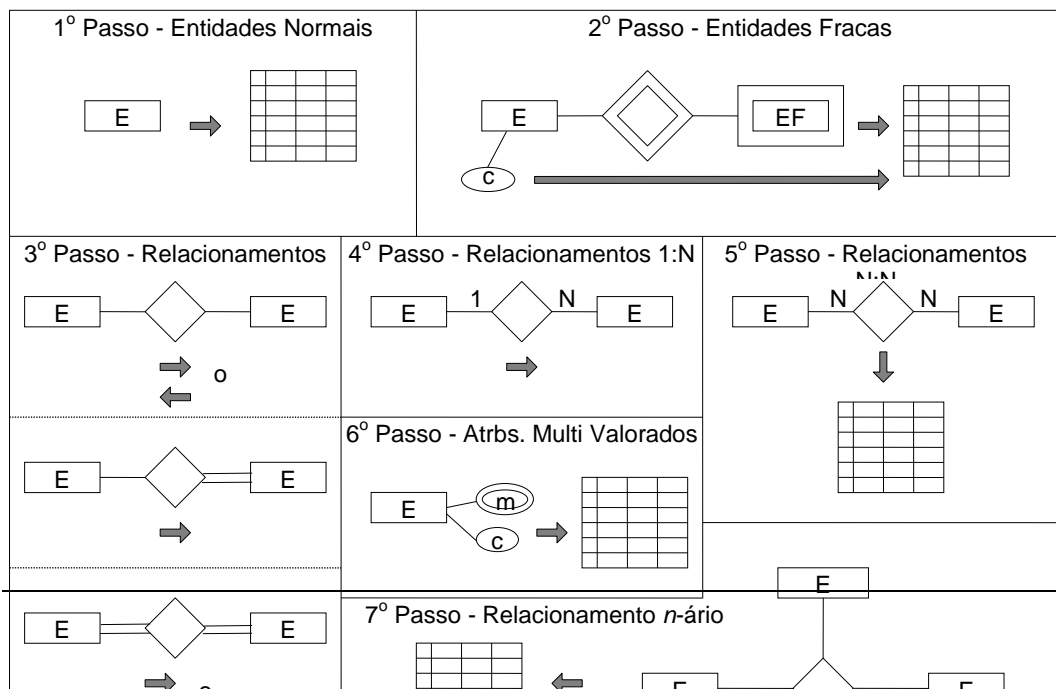
$A(T) = \{c, a_1, a_2, \dots, a_n\} \cup A(S_1) \cup \dots \cup A(S_m) \cup \{t\}$ e $C(T) = c$, onde t é um atributo **tipo** que indica a subclasse à qual cada tupla pertence, caso isto venha a ocorrer;

8.4. Crie uma tabela T com atributos

$A(T) = \{c, a_1, a_2, \dots, a_n\} \cup A(S_1) \cup \dots \cup A(S_m) \cup \{t_1, t_2, \dots, t_m\}$ e $C(T) = c$;

esta opção é para generalizações com “overlapping”, e todos os t_i , $1 \leq i \leq m$, é um atributo “booleano” indicando se a tupla pertence ou não à subclasse S_i ; embora funcional esta opção possa gerar uma quantidade muito grande de valores nulos;

Figura - Mapeamento para o Modelo Relacional



4.8 Normalização de Dados

Consiste em definir o formato conceitual adequado para as estruturas de dados identificados no projeto lógico do sistema, com o objetivo de minimizar o espaço utilizado pelos dados e garantir a integridade e confiabilidade das informações.

A normalização é feita, através da análise dos dados que compõem as estruturas utilizando o conceito chamado "Formas Normais (FN)". As FN são conjuntos de restrições nos quais os dados devem satisfazê-las. Exemplo pode-se dizer que a estrutura está na primeira forma normal (1FN), se os dados que a compõem satisfizerem as restrições definidas para esta etapa.

A normalização completa dos dados é feita, seguindo as restrições das quatro formas normais existentes, sendo que a passagem de uma FN para outra é feita tendo como base o resultado obtido na etapa anterior, ou seja, na FN anterior.

Para realizar a normalização dos dados, é primordial que seja definido um campo chave para a estrutura, campo este que garanta identificar apenas uma tupla da relação.

Primeira Forma Normal (1FN)

Uma tabela está na primeira forma normal se no domínio de seus atributos, isto é, na interseção de uma linha com uma coluna (célula) da tabela, só existirem valores atômicos.

Valor atômico é uma unidade indivisível, simples.

Assim, não pode ocorrer um domínio Endereço com subdomínios Cidade, Rua, Número, Complemento. Uma tabela normalizada teria estes campos separados.

Situação

Funcionario
Codigo
Nome
Endereco

Solução

Funcionario
Codigo
Nome
Cidade
Rua
Numero
Complemento

Bairro

Segunda Forma Normal (2FN)

Uma tabela está na segunda forma normal se ela estiver na primeira forma normal e se cada coluna não-chave depender totalmente da coluna-chave.

Consistem em retirar das estruturas que possuem chaves compostas (campo chave sendo formado por mais de um campo), os elementos que são funcionalmente dependentes de parte da chave. Podemos afirmar que uma estrutura está na 2FN, se ela estiver na 1FN e não possuir campos que são funcionalmente dependentes de parte da chave. Exemplo:

Situação

Venda
NumeroNF
CodigoMercadoria
DescricaoMercadoria
QuantidadeVendida
PreçoVenda
TotalVenda

Solução

Vendas
NumNF
CodigoMercadoria
Quantidade
Valor

Mercadoria
CodigoMercadoria
DescricaoMercadoria
PrecoVenda

Como resultado desta etapa, houve um desdobramento do arquivo de Vendas em duas estruturas, a saber:

Primeira estrutura (Arquivo de Vendas): Contém os elementos originais, sendo excluídos os dados que são dependentes apenas do campo Código da Mercadoria.

Segunda estrutura (Arquivo de Mercadorias): Contém os elementos que são identificados apenas pelo Código da Mercadoria, ou seja, independentemente da Nota Fiscal, a descrição e o preço de venda serão constantes.

Terceira Forma Normal (3FN)

Consistem em retirar das estruturas os campos que são funcionalmente dependentes de outros campos que não são chaves. Podemos afirmar que uma estrutura está na 3FN, se ela estiver na 2FN e não possuir campos dependentes de outros campos não chaves. Exemplo:

Situação

Solução

NotaFiscal	NotaFiscal	Cliente
NumeroNF	NumeroNF	Código
Serie	Serie	NomeCliente
DataEmissao	DataEmissao	CGCCliente
CodigoCliente	CodigoCliente	
NomeCliente	TotalGeral	
CGCCliente		
TotalGeral		

Como resultado desta etapa, houve um desdobramento do arquivo de Notas Fiscais, por ser o único que possuía campos que não eram dependentes da chave principal (Num. NF), uma vez que independente da Nota Fiscal, o Nome e CGC do cliente são inalterados. Este procedimento permite evitar inconsistência nos dados dos arquivos e economizar espaço por eliminar o armazenamento frequente e repetidas vezes destes dados. A cada nota fiscal comprada pelo cliente, haverá o armazenamento destes dados e poderá ocorrer divergência entre eles.

Depois das alterações:

Primeira estrutura (Arquivo de Notas Fiscais): Contém os elementos originais, sendo excluídos os dados que são dependentes apenas do campo Código do Cliente (informações referentes ao cliente).

Segundo estrutura (Arquivo de Clientes): Contém os elementos que são identificados apenas pelo Código do Cliente, ou seja, independente da Nota Fiscal, o Nome, Endereço e CGC dos clientes serão constantes.

Após a normalização, as estruturas dos dados estão projetadas para eliminar as inconsistências e redundâncias dos dados, eliminando desta forma qualquer problema de atualização e operacionalização do sistema. A versão final dos dados poderá sofrer alguma alteração, para atender as necessidades específicas do sistema, a critério do analista de desenvolvimento durante o projeto físico do sistema.

Resumo da Unidade

O **modelo relacional** foi criado por Codd em 1970 e tem por finalidade representar os dados como uma coleção de relações, onde cada relação é representada por uma **tabela**, ou falando de uma forma mais direta, um arquivo. Porém, um arquivo é mais restrito que uma tabela. Toda tabela pode ser considerada um arquivo, porém, nem todo arquivo pode ser considerado uma tabela.

Quando uma relação é pensada como uma tabela de valores, cada linha nesta tabela representa uma coleção de dados relacionados. Estes valores podem ser interpretados como fatos descrevendo uma instância de uma entidade ou de um relacionamento. O nome da tabela e das colunas desta tabela é utilizado para facilitar a interpretação dos valores armazenados em cada linha da tabela. Todos os valores em uma coluna são necessariamente do mesmo tipo.

Na terminologia do modelo relacional, cada tabela é chamada de **relação**; uma linha de uma tabela é chamada de **tupla**; o nome de cada coluna é chamado de **atributo**; o tipo de dado que descreve cada coluna é chamado de **domínio**.

IMPORTANTE

Antes de começar o conteúdo sobre SQL, fazer o exercício abaixo para servir de exemplo no próximo assunto.

* Criar um banco de dados que represente uma empresa.

Este banco deve ser capaz de informar:

- a) Dados sobre os funcionários;
- b) Quantos funcionários trabalham em um determinado setor;
- c) Qual é o salário médio dos funcionários.

Unidade 5 - SQL (structured query language)

5.1 Introdução

Quando os Bancos de Dados Relacionais estavam sendo desenvolvidos, foram criadas linguagens destinadas à sua manipulação. De todas, sem dúvida a que mais se popularizou foi a linguagem SQL.

A linguagem SQL permite a criação e manipulação de dados do banco de dados sem que o programador precise se preocupar com detalhes internos do banco, como a forma como os dados são armazenados e organizados internamente.

Foi criada para definir, modificar e consultar dados armazenados em banco de dados. Os grandes fabricantes de banco de dados criaram extensões próprias para a linguagem. Como exemplo, temos: Transact/SQL, Sysbase, Microsoft, PL/SQL, etc.

Para evitar problemas na manipulação do código, devido à quantidade de plataformas, um comitê foi criado para torná-la independente. Este comitê é o comitê da ANSI (American National Standards Institute).

A linguagem SQL não é procedural, como C, Basic, COBOL, Pascal, entre outras. Nestas linguagens o programador tem que dizer passo-a-passo o que o computador deve fazer.

O SQL é uma linguagem declarativa. Nesta linguagem o programador diz ao computador o que deve ser feito e este se encarrega de fazê-lo.

A linguagem SQL pode ser Interativa ou Embutida. A forma Interativa é usada diretamente para operar um banco de dados, através de uma ferramenta que permite a execução direta do código SQL. A forma Embutida introduz o código SQL “dentro” das linhas do programa, que através de variáveis é tratado adequadamente.

A linguagem SQL é subdividida em três grupos de comando de forma parecida com a estrutura de um banco de dados:

DDL (Data Definition Language):
Comandos responsáveis pela criação de objetos. Ex.: Create Table, Create View, Create Index.

5.2 DML (Data Manipulation Language):

Comando responsável pela alteração de dados. Ex.: Select, Update, Delete.

5.3 DCL (Data Control Language):

Comandos responsáveis pela segurança do banco de dados. Ex.: Grant, Revoke.

5.4 Tipos de Dados:

Na construção de uma tabela é necessário que os dados sejam definidos quanto à forma de armazenamento. As “variações” do SQL criaram outros tipos de dados, como por exemplo: armazenagem de imagens, filmes, vetores, entre outros. O padrão SQL ANSI reconhece apenas os tipos Text e Number.

A tabela abaixo demonstra estes dois tipos:

Texto		
Varchar	Caracteres de tamanho variável	Até 254 caracteres
Char	caracteres de tamanho fixo	Até 254 caracteres
Numérico		
Dec	Número decimal com casas definidas	Até 15 dígitos
Numeric	Número com precisão especificada	Até 22 dígitos
Int	Inteiro sem parte decimal	
Smallint	idêntico a Int com limite de tamanho	
Float	Número de ponto flutuante com base exponencial 10	
Real	Idêntico a Float exceto que não é usado um argumento para especificar o tamanho	
Double	Idêntico a Real, só que com precisão maior.	
Data e Hora		
Date	Armazena datas	
Time	Armazena horas	

5.5 Expressões e Operadores:

Os operadores têm por finalidade permitir uma flexibilidade maior na manipulação dos dados de um banco. Estes seguem precedências de utilização conjunta.

5.6 Tipos de Operadores

Multiplicação	*
Divisão	/
Subtração	-
Adição	+
Módulo	%

Aritméticos

Caracteres

Concatenação de campos	
------------------------	--

Comparação

Igualdade	=
Desigualdade	<>
Maior que	>
Menor que	<
Maior ou Igual a	>=
Menor ou Igual a	<=
Não menor que	!<
Não maior que	!>
Não Igual a	!=

Lógicos

NOT	Inverter o valor Booleano
AND	Verdadeiro se ambos forem verdadeiros
OR	Verdadeiro se for verdadeiro um dos lados
BETWEEN	Verdadeiro se estiver dentro da faixa
LIKE	Verdadeiro se operador encontrar um padrão
IN	Verdadeiro se algum item for verdadeiro
SOME	Verdadeiro se qualquer item for verdadeiro
ANY	Verdadeiro se algum item for verdadeiro
ALL	Verdadeiro se todo o conjunto for verdadeiro

Bitwise

&
!
^

Unários

+	Positivo
-	Bitwise NOT
~	Negativo

Unidade 6 - Criação de Tabelas

6.1 Tutorial:

Na construção de uma tabela é necessário que os dados sejam definidos quanto à forma de armazenamento. As "variações" do SQL criaram outros tipos de dados, como por exemplo: armazenagem de imagens, filmes, vetores, entre outros. O padrão SQL ANSI reconhece apenas os tipos Text e Number.

O tipo **char** só deve ser utilizado quando todas as entradas ocupam o tamanho definido, pois o banco de dados sempre aloca aquele tamanho. Já o tipo **Varchar** aloca um tamanho diferente no banco de dados para cada entrada, sendo mais recomendado quando cada entrada tem um tamanho diferente.

6.2 Criando Tabelas

Antes mesmo de executar a instrução SQL que irá criar as tabelas é fundamental um planejamento e um reconhecimento da estrutura que se propõe criar.

O comando Create Table tem por finalidade criar uma nova tabela no banco de dados.

Sintaxe:

```
Create Table [nome da tabela] ([nome do campo] [tipo de dado]
[tamanho]...).
```

Como todas as máquinas estão acessando uma única base de dados, se todos executarem o comando de criação de uma única tabela, apenas o primeiro será bem sucedido, e os outros retornarão um erro de tabela já existente. Assim, para testarmos os comandos, cada aluno deve criar uma tabela diferente, e para tanto iremos utilizar o nome de cada máquina. Sempre que for mencionada nos comandos a palavra "NOME_DE_SUA_MAUQUINA", substitua pelo nome da máquina que estiver utilizando.

Comando:

```
Create Table NOME_DE_SUA_MAUQUINA (
  Id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT
  Nome VARCHAR (20)
)
```

Este comando permitiu a criação de uma tabela que contém dois campos, um inteiro chamado Id e um campo VARCHAR chamado Nome.

Se utilizarmos o comando Select, podemos ver os campos e o conteúdo:

Comando:

```
Select * from NOME_DE_SUA_MAUQUINA
```

Resultado:

Id	Nome
-----	-----

Observe que, embora a tabela esteja vazia, aparecem os nomes dos dois campos criados, mostrando que a criação da tabela foi bem sucedida.

6.3 Parâmetros de Campos na Criação de Tabelas:

6.4 Valores Nulos

Pode-se definir se o campo receberá valores NULOS ou não. Quando os valores não são NULOS tornam-se obrigatórios, o que significa que não será possível inserir dados na tabela em que tais campos não tenham um valor.

6.5 Chave Primária

Por definição é o dado que diferencia uma linha de outra na tabela tornando-as únicas. Pode ser composta, ou seja, utiliza mais de uma coluna para compor um valor único.

Para um campo ser definido como Chave Primária, ele não poderá receber um valor NULO.

Por exemplo, muitas vezes criamos uma tabela com um campo de identificação (geralmente chamado de Id, Cod, Código ou outro nome semelhante). Este campo será utilizado em consultas para relacionar esta tabela com outras, como vimos nos exercícios com **Select**. É mais eficiente para certas consultas que este campo seja a chave primária, mesmo porque ele necessariamente será único.

Observe a sintaxe para criarmos a tabela novamente com o Id como sendo obrigatório e chave primária. Não se esqueça de remover a tabela, se já não o fez, antes de executar o comando:

Comando:

```
Create Table NOME_DE_SUA_MQUINA(  
Id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT  
Nome VARCHAR (20),  
PRIMARY KEY(Id)  
)
```


6.6 Chave Estrangeira:

Por definição é o campo de outra tabela que recebe os valores do campo da Chave Primária.

Esta ligação a um campo de **Chave Primária** possibilita que os dados mantenham a integridade referencial. Esta integridade significa que os dados de uma tabela só poderão ser inseridos de forma válida e correspondente.

Execute o comando `select * from pedidos` e confira que a tabela contém duas chaves estrangeiras. Veja se consegue descobrir quais são.

6.7 Integridade Referencial:

A Integridade Referencial é utilizada para garantir a Integridade dos dados entre as tabelas relacionadas. Por exemplo, considere um relacionamento do tipo Um-para-Vários entre a tabela Clientes e a tabela Pedidos (um cliente pode fazer vários pedidos). Com a Integridade Referencial, o banco de dados não permite que seja cadastrado um pedido para um cliente que ainda não foi cadastrado. Em outras palavras, ao cadastrar um pedido, o banco de dados verifica se o código do cliente que foi digitado já existe na tabela Clientes. Se não existir, o cadastro do pedido não será aceito. Com o uso da Integridade Referencial é possível ter as seguintes garantias (ainda usando o exemplo entre as tabelas Clientes e Pedidos):

Quando o Código de um cliente for alterado na Tabela Clientes, podemos configurar para o banco de dados atualizar, automaticamente, todos os Códigos do Cliente na Tabela Pedidos, de tal maneira que não fiquem Registros Órfãos, isto é, registros de Pedidos com um Código de Cliente para o qual não existe mais um correspondente na Tabela Clientes. Essa ação é conhecida como "Propagar atualização dos campos relacionados". Quando um Cliente for excluído da Tabela Clientes, podemos configurar para que o banco de dados exclua, automaticamente, na tabela Pedidos, todos os Pedidos para o Cliente que está sendo Excluído. Essa opção é conhecida como "Propagar exclusão dos registros relacionados".

6.8 ON UPDATE:

NO ACTION (RESTRICT) - quando o campo chave primária está para ser

atualizado a atualização é abortada caso um registro em uma tabela referenciada tenha um valor mais antigo. Este parâmetro é o default quando esta cláusula não recebe nenhum parâmetro.

Exemplo: ERRO Ao tentar usar "UPDATE clientes SET código = 5 WHERE código = 2."

Ele vai tentar atualizar o código para 5, mas como em pedidos existem registros do cliente 2 haverá o erro.

CASCADE (Em Cascata) - Quando o campo da chave primária é atualizado, registros na tabela referenciada são atualizados.

Exemplo: Funciona: Ao tentar usar "UPDATE clientes SET código = 5 WHERE código = 2. Ele vai tentar atualizar o código para 5 e vai atualizar esta chave também na tabela pedidos.

SET NULL - Quando um registro na chave primária é atualizado, todos os campos dos registros referenciados a este são setados para NULL.

Exemplo: UPDATE clientes SET código = 9 WHERE código = 5;

Na clientes o código vai para 5 e em pedidos, todos os campos cod_cliente com valor 5 serão setados para NULL.

SET DEFAULT - Quando um registro na chave primária é atualizado, todos os campos nos registros relacionados são setados para seu valor DEFAULT.

Exemplo: se o valor default do código de clientes é 999, então.

UPDATE clientes SET codigo = 10 WHERE codigo = 2. Após esta consulta o campo código com valor 2 em clientes vai para 999 e também todos os campos cod_cliente em pedidos.

6.9 ON DELETE:

NO ACTION (RESTRICT) - Quando um campo de chave primária está para ser deletado, a exclusão será abortada caso o valor de um registro na tabela referenciada seja mais velho. Este parâmetro é o default quando esta cláusula não recebe nenhum parâmetro.

Exemplo: ERRO em DELETE FROM clientes WHERE código = 2. Não funcionará caso o cod_cliente em pedidos contenha um valor mais antigo que código em clientes.

CASCADE - Quando um registro com a chave primária é excluído, todos os registros relacionados com aquela chave são excluídos.

SET NULL - Quando um registro com a chave primária é excluído, os respectivos campos na tabela relacionada são setados para NULL.

SET DEFAULT - Quando um registro com a chave primária é excluído, os campos respectivos da tabela relacionada são setados para seu valor DEFAULT.

6.10 Removendo Tabelas

Para remover uma tabela de um banco de dados deve-se utilizar o comando **Drop Table** seguido do nome da tabela:

Sintaxe:

Drop Table [nome da tabela]

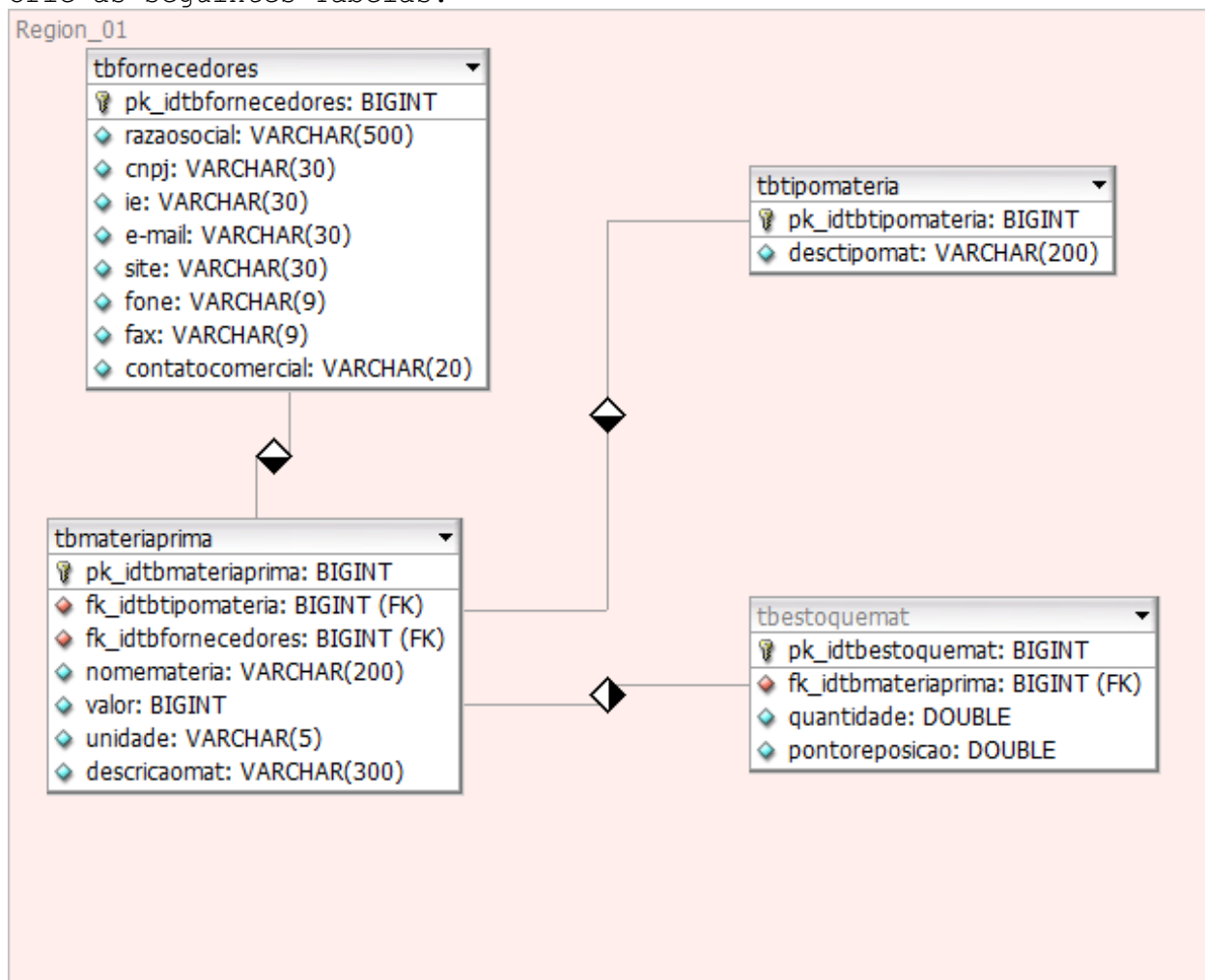
Execute o comando para remover a tabela criada anteriormente, para podermos criá-la com novos campos.

Comando:

Drop Table NOME_DE_SUA_MAQUINA

Laboratório

Crie as seguintes Tabelas:



Unidade 7 - Comandos para Alterar Tabelas

7.1 Alteração de Tabelas Criadas

Após a criação das tabelas pode ser necessário incluir ou remover colunas. Sempre é possível remover a tabela e criá-la novamente com a estrutura nova, mas com isto perderíamos todos os dados previamente cadastrados. Por isto da importância do comando ALTER TABLE.

Sintaxe:

```
ALTER TABLE [nome da tabela]
DROP [nome da coluna]
ADD [nome da coluna e tipos de dados]
RENAME [nome atual] [novo nome]
MODIFY [nome da coluna e tipos de dados]
```

7.2 Adicionar colunas a uma tabela – ADD

Este comando adiciona colunas a uma tabela. As colunas serão criadas da mesma forma que no comando CREATE TABLE.

Comando:

```
ALTER TABLE NOME_DE_SUA_MAUQUINA ADD Sobrenome VarChar(20)
```

7.3 Remover colunas de uma tabela – DROP

Remove uma coluna de uma tabela. Caso a coluna já possua dados digitados, eles também serão perdidos.

Comando:

```
ALTER TABLE NOME_DE_SUA_MAUQUINIA DROP Sobrenome
```

7.4 Renomeando colunas de uma tabela – RENAME

Renomeia uma coluna de uma tabela. Caso a coluna já possua dados digitados, eles não serão perdidos.

Comando:

```
ALTER TABLE NOME_DE_SUA_MAUQUINIA RENAME Sobrenome SegundoNome
```

7.5 Modificando colunas de uma tabela – MODIFY

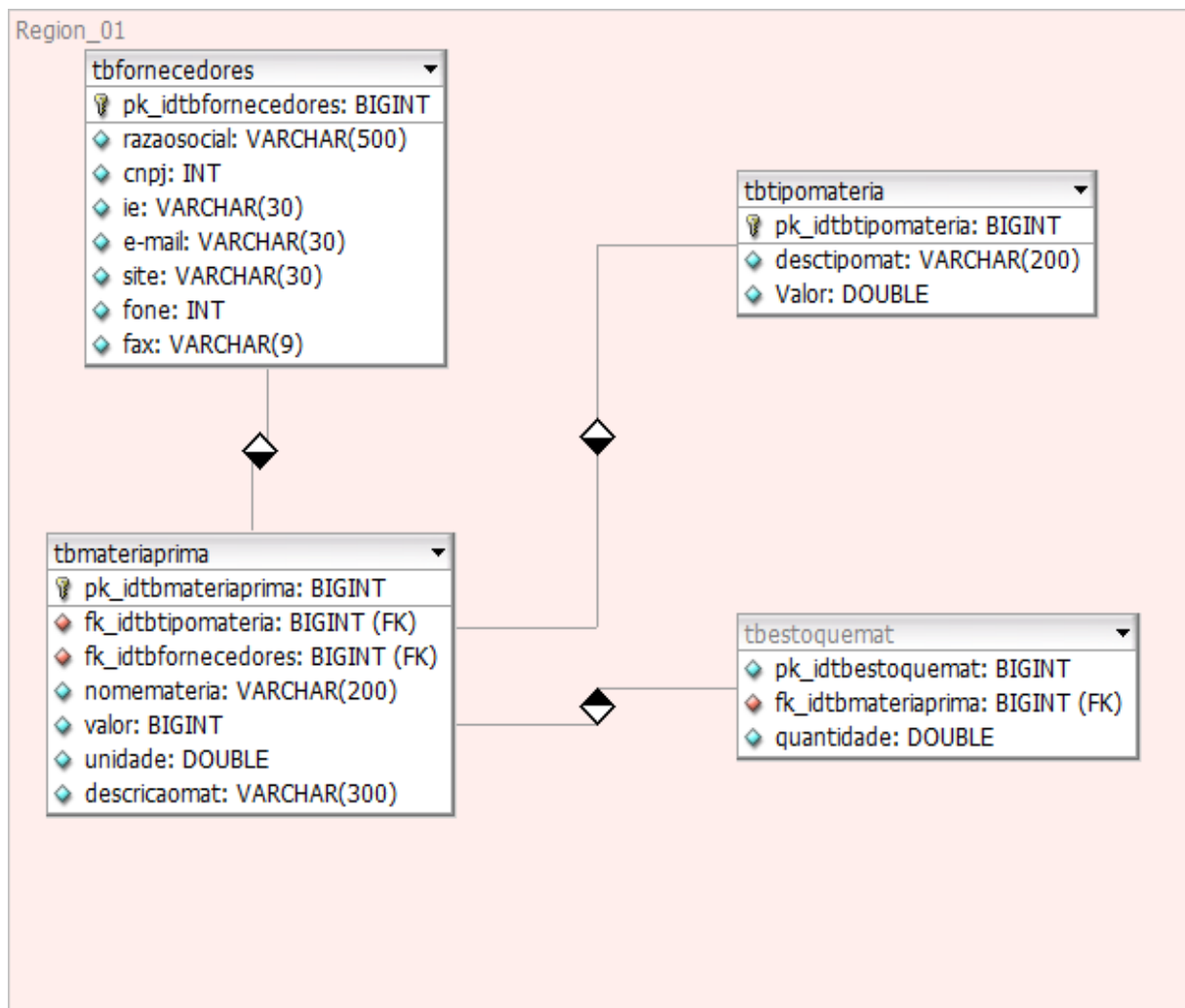
Modifica uma coluna de uma tabela. Caso a coluna já possua dados digitados, eles não serão perdidos.

Comando:

```
ALTER TABLE NOME_DE_SUA_MAQUINIA MODIFY SegundoNome varchar(30)
```

Laboratório

Modifique as tabelas do exercício anterior para que fiquem com o seguinte formato:



Unidade 8 - Modificando Dados

8.1 Inserindo Dados

Além dos comandos de consulta ao banco de dados, uma aplicação geralmente irá também inserir novos dados e atualizar os existentes;

Ao inserir dados em uma tabela, o administrador do banco de dados deverá conhecer qual a estrutura da tabela, bem como o tipo de campo. Sem estas especificações ele não conseguirá fazer a inserção, pois os dados serão rejeitados.

A sintaxe para inserir dados é mostrada abaixo:

Sintaxe:

```
Insert into [nome da tabela] (Campo1, Campo2, ...) values (Valor1, Valor2,...);
```

Vamos agora criar uma nova tabela e inserir um dado nela.

Comando de Criação de Tabela

```
Create Table NOME_DE_SUA_MAQUINA(Id INT, Nome VARCHAR (20))
```

Comando de Inserção de Dados

```
Insert Into NOME_DE_SUA_MAQUINA (Id, Nome) values (1, 'João')
```

Com esta instrução SQL obteremos o seguinte resultado:

Comando:

```
Select * from NOME_DE_SUA_MAQUINA;
```

Resultado:

Id	Nome
-----	-----
-	
1	João

8.2 Inserindo dados por meio de um SELECT

É possível fazer a inserção de dados através de uma instrução SQL com o comando Select, com todas as suas variações.

Esta possibilidade é muito útil, pois dependendo do caso não será necessário digitar os dados nesta tabela, uma vez que eles podem ser reproduzidos através de tabelas existentes no Banco de Dados.

Exemplificando:

Crie sua tabela com um único campo 'nome' (não se esqueça de dar um DROP TABLE antes):

Comando:

```
Create Table NOME_DE_SUA_MAQUINA(Nome VarChar (50))
```

Agora vamos colocar nesta tabela todos os nomes de funcionários da empresa:

Comando:

```
insert into NOME_DE_SUA_MAQUINA(Nome) select Nome from funcionarios
```

Confira executando um comando **SELECT * FROM NOME_DE_SUA_MAQUINA**. Se tudo ocorrer corretamente, o resultado será que você terá uma nova tabela com todos os nomes presentes na tabela funcionários.

8.3 Excluindo informações

Este comando remove linhas das tabelas. Podemos remover todo o conteúdo da tabela, ou definir as linhas a serem removidas utilizando a cláusula **WHERE**. A sintaxe é mostrada abaixo:

Sintaxe:

```
DELETE FROM [nome da tabela] WHERE [Condição];
```

Comando:

```
Delete from NOME_DE_SUA_MAQUINA where Nome = 'José'
```

Lembre-se que ao excluir uma linha de registro relacionada à exclusão poderá ser bloqueada pelo SQL devido as **PRIMARY KEY** e **FOREIGN KEY** existentes.

Para remover todos os itens de uma tabela basta não incluir a cláusula **WHERE**.

8.4 Atualizando dados de uma tabela

Através deste comando é possível atualizar um ou mais campos de uma tabela. A sintaxe é mostrada abaixo:

Sintaxe:

```
UPDATE [nome da tabela] SET [nome da coluna] = [novo valor]
```

É possível utilizar as cláusulas **FROM** e **WHERE** para determinar a atualização vinda de outras tabelas ou quais os critérios utilizados para a atualização.

Expressões poderão ser utilizadas para realizar as alterações nos campos.

Execute o comando **select * from produtos** antes e depois de rodar o comando abaixo:

Comando:

```
update produtos set preco = preco * 2
```

Observe que os valores podem ser diferentes, pois todos os alunos estão alterando a mesma tabela, neste caso.

Esta alteração está permitindo ao usuário aumentar em 100% o preço de todos os produtos. Esta alteração poderia ser feita utilizando-se uma restrição com a cláusula **Where**, para atualizar apenas um produto.

Comando:

```
update produtos set preco=20 where id = 2
```

Este comando define como 20 o preço do produto 2, ou seja, da carne.

Laboratório

Baseado nas tabelas criadas e as alterações realizadas nas tabelas das unidades anteriores realize os seguintes comandos:

Insira 3 registros em cada tabela

Delete 1 registro da tabela tbfornecedores
Atualize o valor da matéria prima em 20%
Insira 2 registros na tabela tbmateriaprima
Remova o ultimo registro da tabela tbestoque
Atualize a tabela tbestoque para um aumento de 50% na quantidade
Insira 2 registros na tbfornecedores
Delete 1 registro da tabela tbtipomateria
Insira 2 registros na tabela tbtipomateria
Reduza em 13% o valor da tabela tbmateriaprima

Unidade 9 - Consulta em uma única Tabela SQL

9.1 Sintaxe Básica de Consultas SQL

Em bancos de dados relacionais, temos a informação armazenada em tabelas, como visto anteriormente. Cada tabela contém um conjunto de informações armazenadas em linhas da tabela, cada linha representando uma entrada, sendo que as colunas representam os parâmetros associados àquela informação.

Uma das consultas mais simples que podemos realizar em SQL, portanto, é a consulta que retorna os parâmetros que desejamos de todas as entradas da tabela. Sua sintaxe é mostrada abaixo:

```
SELECT      [nome      do      campo1],      [nome      do      campo2],      ...  
FROM [nome da tabela]
```

O campo “nome da tabela” define a tabela que estamos consultando, enquanto os campos “nome do campoN” representam os parâmetros, e, portanto colunas, que desejamos recuperar.

9.2 Exemplos de Consultas SQL

Inicialmente faremos algumas consultas simples a uma única tabela. Nos exemplos desta unidade estaremos utilizando a tabela “transportadoras”, cujo conteúdo é mostrado abaixo. O instrutor poderá utilizar os mesmos exemplos ou criar junto com os alunos tabelas próprias, utilizando o banco de dados que for mais conveniente.

Obs.: Neste e nos demais exemplos de tabelas do banco de dados, a primeira linha especifica o nome do atributo ao qual cada coluna se refere.

Tabela transportadoras:

Nome	Telefone
Transportadora Fulano	(51) 2111-1666
Transportes Pedrão	(51) 0000-0000
Expresso Marte	(51) 1234-5678

9.3 Seleção de Todos os Registros com Colunas Específicas

Execute a consulta abaixo, ou siga as instruções do instrutor, executando uma consulta equivalente:

```
Select Nome from transportadoras
```

Executando a consulta acima na tabela Transportadoras, contendo os dados mostrados, teremos o seguinte resultado:

Nome

Transportadora Fulano
Transportes Pedrão
Expresso Marte

Na mesma tabela, também podemos consultar mais parâmetros, bastando especificá-los separados por vírgulas. Confira a consulta a seguir e o resultado correspondente:

```
Select Nome, Telefone from transportadoras
```

Resultado:

Nome	Telefone
-----	-----
Transportadora Fulano	(51) 3346-7300
Transportes Pedrão	(51) 0000-0000
Expresso Marte	(51) 1234-5678

9.4 Redefinindo o Nome das Colunas

Normalmente as colunas retornadas por uma consulta SQL possuem o nome da coluna respectiva na tabela do banco de dados. Algumas vezes, porém, podemos querer definir outros nomes para as colunas.

Sintaxe:

```
SELECT [nome do campo] as nome definido FROM [nome da tabela]
```

Comando:

```
select nome as transportadoras from transportadoras
```

Resultado:

Transportadoras

Transportadora Fulano
Transportes Pedrão
Expresso Marte

9.5 Seleção de Registros com Exibição de Strings

É possível selecionar registros de uma tabela exibindo um texto fixo para cada linha de registro

Sintaxe:

```
SELECT 'texto', [nome do campo], 'texto', [nome do campo] FROM [nome da tabela]
```

Comando:

```
select nome, ' ocupa o cargo de', cargo from funcionarios
```

Resultado:

Nome	'ocupa o cargo de'	Cargo
-----	-----	-----
José	ocupa o cargo de	Presidente
Maurício	ocupa o cargo de	Gerente de Vendas
Luís	ocupa o cargo de	Boy
Lígia	ocupa o cargo de	Telefonista

Consulta com Execução de Cálculos:

Com o SQL podemos efetuar cálculos entre elementos lidos do banco de dados e retorná-los na consulta. Para selecionar registros de uma tabela com a execução de cálculos matemáticos é necessário utilizar os operadores '+', '-', '/', '*' e '^'.

Sintaxe:

```
SELECT [nome do campo]*[nome do campo] as [nome da expressão] FROM [nome da tabela];
```

Para demonstrarmos este recurso do SQL, utilizaremos uma nova tabela, com dados numéricos. Novamente, o instrutor poderá utilizar estes mesmos dados ou criar exemplos próprios ao longo da aula:

Nome	Preço	Estoque
Leite	10	500
Carne	20	330
Pão	18.6	200
Suco	20	330

Comando:

```
select nome, preco*estoque as total from produtos
```

Resultado:

Nome	total
------	-------

-----	-----
Leite	5000
Carne	6600
Pão	37200
Suco	6600

Laboratório

Baseado nas tabelas criadas e as alterações realizadas nas tabelas das unidades anteriores exiba os seguintes resultados:

O fornecedor “x” tem a razão social “y”

A matéria prima “x” tem o valor de “y” e a quantidade “z”

A matéria prima “x” tem um valor total de “quantidade * valor”

A quantidade “x” no estoque não pode chegar a “20% da quantidade”

O tipo de matéria “x” tem o valor de “y” e com desconto fica em “80% do valor”

A matéria “x” é descrita como “y”

“12% do valor” são considerados custo da matéria prima “x”

Acesse o site “x” da empresa “y”

“1/3 do valor * quantidade” é destinado ao pagamento de imposto da matéria prima “y”

Após um aumento de 13% no estoque, a quantidade vai de “x” para “y”

Unidade 10 - Unidade 10: Funções com strings

10.1 Funções com Strings

Funções que tratam de strings permitem fazer uma série de manipulações de textos que são recuperadas das tabelas do banco de dados.

Nesta unidade apresentamos algumas das funções mais comumente utilizadas em situações reais, para o aluno se familiarizar com o conceito de funções de manipulação de strings. Na prática pode ser interessante consultar as funções que o banco de dados que estiver sendo utilizado fornece, quando for necessário manipular strings.

10.2 Sintaxe da Função de Concatenação de Strings

Para juntar duas ou mais strings existe uma função, chamada concat, mostrada a seguir:

Sintaxe:

```
concat( string1, string2,...)
```

O exemplo a seguir mostra o uso desta função para juntar o nome e sobrenome dos funcionários em uma única coluna. Observem o uso do espaço para garantir que o nome e sobrenome vão ter uma separação.

Exemplo:

```
select concat(Nome, ' ', Sobrenome) from funcionarios
```

10.3 Controle de Maiúsculas e Minúsculas

Duas funções permitem transformar o retorno de uma consulta em letras maiúsculas e minúsculas.

Lower

Transforma as letras em minúsculas na resposta da instrução SQL. Como exemplo:

```
Select Lower(Nome) from clientes;
```

Upper

Transforma as letras em maiúsculas na resposta da instrução SQL. Como exemplo:

```
Select Upper(NomeDoContato) from clientes;
```

10.4 Segmentando uma String

Alguns comandos permitem separar uma string, recuperando apenas determinada parte da informação que se deseja.

Substr

Separa uma parte da linha baseada na seguinte forma:

```
SUBSTR(String1, Valor1, Valor2),
```

Onde:

String1 é a string da qual vai ser recuperado um segmento

Valor1 é onde começa a ser feita a separação

Valor 2 é o número de caracteres que serão separados.

Exemplo:

```
Select Substring(Nome, 1, 3) from clientes
```

Left

Separa uma parte da linha baseada na seguinte forma:

```
LEFT (String1, Valor1)
```

Onde,

String1 é a string de origem

Valor1 é o número de caracteres que serão separados a partir da esquerda.

Exemplo:

```
Select Left(Nome, 1) from clientes;
```

Right

Separa uma parte da linha baseada na seguinte forma:

```
RIGHT (String1, Valor1)
```

Onde,

String1 é a string de origem

Valor1 é o número de caracteres que serão separados a partir da direita.

Exemplo:

```
Select Right(Nome, 1) from clientes;
```

Length(Len)

Retorna o tamanho da string:

```
LENGTH (String1)
```

Onde,

String1 é a string de origem

Exemplo:

```
Select LENGTH(Nome) from clientes;
```

Locate(Charindex)

Retorna a posição do texto procurado:

```
LOCATE (Valor1, String1)
```

Onde,

Valor1 é o texto procurado.

String1 é a string de origem

Exemplo:

```
Select LOCATE('João', Nome) from clientes;
```

Laboratório

Baseado nas tabelas criadas e as alterações realizadas nas tabelas das unidades anteriores exiba os seguintes resultados:

O fornecedor “apenas o 1º nome” é do tipo “final da razão social”

A matéria prima “em maiúscula” tem o valor de “y” e a quantidade “z”

A matéria prima iniciadas em “1º letra” tem um valor total de “quantidade * valor”

O tipo de matéria “em minúsculo” tem o valor de “y” e com desconto fica em “80% do valor”

A matéria “duas letras iniciais” é descrita como “em maiúsculo”

“12% do valor” são considerados custo da matéria prima “1 letra em maiúsculo,”.

Acesse o site “retirar o WWW” da empresa “y”

“1/3 do valor * quantidade” é destinado ao pagamento de imposto da matéria prima “em maiúscula”

A matéria prima “x” é formada por “quantidade de letras” letras

O fornecedor “x” tem um fax de “quantidade de letras” letras

Unidade 11 - Critérios de Consulta – Cláusula WHERE

11.1 Cláusula Where

Até o momento todas as consultas apresentadas retornaram todas as entradas das tabelas, restringindo unicamente os atributos que desejávamos recuperar. Em outras palavras, os comandos anteriores recuperavam algumas colunas, mas todas as linhas das tabelas.

Em muitos casos, desejaremos recuperar apenas uma ou algumas entradas de uma tabela específica, por exemplo, quando desejarmos recuperar dados sobre um cliente específico. Nestes casos, nossa consulta SQL terá uma cláusula Where, conforme sintaxe mostrada abaixo:

```
SELECT [nome do campo] FROM [nome da tabela] WHERE [nome do campo] [operador de comparação ou restrição] [critério de comparação ou restrição]
```

```
UPDATE [nome da tabela] SET [nome do campo] = [novo valor] WHERE [nome do campo] [operador de comparação ou restrição] [critério de comparação ou restrição]
```

```
DELETE FROM [nome da tabela] WHERE [nome do campo] [operador de comparação ou restrição] [critério de comparação ou restrição]
```

Uma consulta pode ter uma quantidade infinita de condições ou cláusulas, sendo que as mesmas devem ser separadas por um operador lógico (AND ou OR). Veja o exemplo abaixo:

```
SELECT [nome do campo] FROM [nome da tabela] WHERE [nome do campo] [operador de comparação ou restrição] [critério de comparação ou restrição] AND [nome do campo] [operador de comparação ou restrição] [critério de comparação ou restrição]
```

Ou

```
SELECT [nome do campo] FROM [nome da tabela] WHERE [nome do campo] [operador de comparação ou restrição] [critério de comparação ou restrição] OR [nome do campo] [operador de comparação ou restrição] [critério de comparação ou restrição]
```

Os campos que são utilizados nos critérios das consultas, não são obrigatoriamente os mesmo utilizados como saída de informação (para consultas de seleção) ou como objetos de alteração. Podendo, por exemplo, exibir o nome de produtos que tenha um preço inferior a um determinado valor. Veja o exemplo, sendo “nomeproduto” o campo a ser exibido e “preco” o campo que recebe a condição de exibição:

```
SELECT nomeproduto FROM produtos WHERE preco <= 2.5
```

Observe, também, que é possível a utilização de um mesmo campo, mais de uma vez como critério de uma consulta. Por exemplo, se desejar exibir os filmes lançados após o ano 2000 com a exceção do ano de 2005, o comando seria desenhado da seguinte forma.


```
SELECT nomefilme FROM filmes WHERE ano >= 2000 AND ano <> 2005
```

Estes critérios podem ser utilizados em qualquer tipo de consulta (seleção, exclusão ou atualização). Principalmente em consultas que realizam alterações, recomenda-se atentar para o uso quase que obrigatório da cláusula “where”, pois sua omissão resulta na execução da alteração em todo o conjunto de dados contemplados na sentença.

Vejamos a seguinte situação: para os funcionários com mais de 1 ano de tempo de trabalho a empresa deseja conceder um aumento de 15% em seus salários. Se executarmos uma consulta conforme exemplo abaixo, todos os funcionários receberam aumento independente do seu tempo na empresa:

```
UPDATE funcionarios SET salario = salario * 1.15
```

Para que apenas os funcionários que possuem o critério estabelecido pela empresa para receber o aumento, sejam efetivamente beneficiados, é necessário incluir uma cláusula. Observe a sintaxe correta abaixo.

```
UPDATE funcionarios SET salario = salario * 1.15 WHERE tempo >= 1
```

Em consultas de deleção o cuidado deve ser ainda maior, pois uma execução de uma sentença sem um critério de forma correta ou mesmo sem a existência de um, pode acarretar na perda significativa de dados importantes para a instituição.

Como regra geral recomenda-se iniciar a escrita de seu script SQL, para consultas de alterações, pelos critérios que as mesmas devem considerar, reduzindo assim, a chance de cometer qualquer tipo de equívoco, com consequências mais graves.

11.2 Consultas Simples com Cláusula Where

Vamos definir uma consulta bastante simples, restringindo a lista de clientes apenas aqueles cujo país é o Brasil. Utilizaremos a tabela clientes, mostrada a seguir:

Tabela Clientes

Nome	País
Empresa W	Brasil
Empresa A	Brasil
Empresa B	Brasil
Empresa C	Argentina
Empresa X	Brasil

Comando:

```
SELECT NOME FROM CLIENTES WHERE PAIS='BRASIL'
```

Resultado:

nome

Empresa A
Empresa B
Empresa X

O sinal de igualdade (=) foi utilizado porque o usuário sabia exatamente o que estava procurando. Verifique os operadores de comparação que estão no início da apostila.

Cabe também, ressaltar que devemos sempre atentar para o tipo de dado armazenado nos campos que serão utilizados nos critérios de consulta. Pois esta observação nos mostra que tipos de critérios podem ser utilizados (para datas, por exemplo) e qual a forma correta de montagem do critério ou seleção.

No exemplo da consulta acima, podemos perceber que o critério foi digitado entre aspas simples, pois o valor do campo é do tipo texto, a exemplo do aconteceria com um campo data (apenas devemos atentar para seu formato).

Para campos tipados como numéricos não há a necessidade, além de não ser recomendado, do uso de aspas simples. Entretanto, se estiver utilizando um banco de dados que você não tenha estruturado, utilize comandos (veremos nas próximas unidades) que exibam a estrutura da tabela, pois podemos encontrar campos com valores numéricos que tenham sido tipados como texto, o que pode atrapalhar a percepção do resultado final.

11.3 Selecionando Dados Não Repetidos

Algumas vezes não queremos recuperar todas as entradas de determinada tabela, mas apenas aquelas não repetidas. Podemos, por exemplo, querer listar todas as cidades em que a empresa tem clientes, ou todas as faixas de salário dos funcionários, e nestes casos não faz sentido repetir inúmeras vezes a mesma cidade ou faixa salarial apenas por ter mais de um dado com aquela informação.

Existe um comando para resolver esta questão, chamado **DISTINCT**, cuja sintaxe é mostrada abaixo:

```
SELECT DISTINCT [nome do campo] FROM [nome da tabela];
```

Executando a consulta abaixo, veremos que retornará os países em que existem clientes armazenados no banco de dados:

```
select distinct pais from clientes
```

Resultado:

País
Argentina
Brasil

Case o comando seja executado novamente, porém sem a cláusula DISTINCT e observe a termos a repetição dos países.

O comando DISTINCT realiza um agrupamento dos dados exibidos, portanto devemos sempre estar atentos aos campos que inserimos antes da expressão FROM, pois se os mesmos forem únicos, pode não ocorrer o agrupamento de forma correta.

No exemplo aqui citado os países, aparecerão apenas uma vez, pois somente a coluna “país” foi exibida, se fosse inserido o campo “nome”, por exemplo, haveria a duplicação dos mesmos.

Vejamos o caso na prática

```
select distinct nome, pais from clientes
```

Resultado:

Nome	País
	Brasil
Empresa A	Brasil
Empresa B	Brasil
Empresa C	Argentina
Empresa X	Brasil

11.4 Selecionando alguns dados

Em determinadas situações desejamos apenas obter alguns registros de uma consulta, pois apenas uma amostra de dados pode ser suficiente para chegarmos ao resultado esperado da sentença.

Esta sintaxe para construção de um limite dos resultados leva em consideração o banco de dados no qual as informações estão armazenadas, sejamos abaixo a sintaxe para Access/SQL SERVER e para MySQL.

Access/SQL SERVER:

```
SELECT TOP [valor] [opcional PERCENT] [nome do campo] FROM [nome da  
tabela] WHERE [nome do campo] [operador de comparação ou restrição]  
[critério de comparação ou restrição]
```

MySQL

```
SELECT [nome do campo] FROM [nome da tabela] WHERE [nome do campo]  
[operador de comparação ou restrição] [critério de comparação ou  
restrição] LIMIT [valor]
```

Se desejarmos, por exemplo, exibir apenas 5 clientes que sejam do país ‘Brasil’, devemos construir a consulta no SQL SERVER.

```
SELECT TOP 5 nome FROM clientes WHERE pais='brasil'
```

A utilização da cláusula de limite de registros é muito utilizada em testes de verificação do conteúdo de uma tabela. Também é amplamente aplicada em subconsultas (tema do curso avançado).

Laboratório.

Para a realização dos exercícios abaixo crie a seguinte tabela e insira os dados listados abaixo dela. Após execute uma consulta que exiba cada

CELULARES
Cod_Celular: BIGINT
Modelo: VARCHAR(40)
Marca: VARCHAR(40)
Camera: BOOL
MP3: BOOL
Bateria: FLOAT
Valor: FLOAT
Lancamento: DATE

Cod_Celular	Modelo	Marca	Camera	MP3	Bateria	Valor	Lancamento
1	Star	Samsung	Sim	Sim	3.2	600.00	2009-04-12
2	N97	Nokia	Sim	Sim	4.1	700.00	2010-01-20
3	IPhone	Apple	Sim	Sim	3.4	1.100	2007-12-30
4	Corby	Samsung	Sim	Sim	4.2	800.00	2008-06-25
5	N95	Nokia	Sim	Sim	2.7	950.00	2005-05-05
6	Jet	Samsung	Sim	Sim	3.3	650.00	2008-10-09
7	2160	Nokia	Não	Não	8.7	50.00	1997-03-04

Os dados de todos os celulares com mp3.

Liste modelo de todos os celulares da Samsung.

Liste nome de todos os modelos, junto com seu valor e marca.

Liste nome de todos os celulares, junto com seus valores, para todos os celulares que foram lançados há pelo menos de 2 anos.

Liste o modelo, marca e bateria do celular código 4.

Liste os celulares lançados a mais de 10 anos.

Liste todos os celulares com bateria com mais de 4 horas e menos de 6 horas

Liste 2 celulares com câmera

Liste as marcas de celular existentes

Liste celulares da marca “Nokia” com bateria de mais e 3 horas e com valor inferior a R\$ 700,00

Atualize em 10% o valor dos celulares da “Nokia”

Insira mais 4 modelos de celular

Delete 2 celulares da Samsung

Atualize a informação MP3 para NÃO em celulares com bateria inferior a 3 horas e valor superior a R\$ R\$ 500,00

Mude o nome do celular de código 7 para “MP15”

Atualize o valor dos celulares em 2% para celulares com câmera e valor inferior a R\$ 800,00

Atualize 3 celulares para câmera NÃO

Mude o valor dos celulares Samsung para 80% de seu valor atual

Insira 2 celulares com nomes novos utilizando os dados de outro celular

Delete todos os celulares com valor inferior a R\$ 300,00

Unidade 12 - Predicados

12.1 Predicados LIKE e NOT LIKE:

Na unidade anterior, vimos como selecionar apenas algumas entradas de uma tabela, com o uso da cláusula Where. Entretanto, apenas com os operadores de igualdade, que foi mostrado, e outros operadores aritméticos, como “<”, “>”, etc., não conseguiremos sempre selecionar o que desejamos. Em particular, quando desejamos selecionar entradas com base em um atributo de texto, precisamos de recursos mais poderosos. Por exemplo, podemos querer procurar por um cliente do qual não temos todo o nome, mas apenas um dos sobrenomes. Utilizar o operador ‘=’ não irá nos atender neste caso.

O predicado “LIKE” serve para este fim. Funciona como um tipo de pesquisador de sequências podendo ser auxiliado por um % que significa qualquer caractere. Pode ser precedido por um NOT que expressa a negação. Sua sintaxe é mostrada abaixo:

```
Select * from [nome da tabela] WHERE [campo analisado] {NOT  
LIKE} {LIKE} [valor ou texto];
```

Para entender a vantagem de utilizar os comandos LIKE e NOT LIKE vamos primeiro executar uma consulta com o operador de igualdade. Nesta apostila utilizaremos uma tabela bastante simples, com fins didáticos, sendo possível que o instrutor opte por construir exemplos mais complexos junto com a turma.

Tabela clientes:

Nome
João da Silva Borges
Pedro Álvaro
João Augusto

Comando:

```
select nome from clientes where nome = 'João'
```

Resultado:

nome

A resposta para esta instrução será uma tabela vazia. Apesar de existir um 'João' na tabela, o critério de igualdade exige o nome completo. A consulta apenas retornaria o cliente se fosse colocado o nome completo: 'João da Silva'

Assim, utilizar o sinal de igualdade nos retorna o resultado desejado somente se conhecermos o nome completo da pessoa. O comparativo lógico LIKE, permite selecionar por semelhança:

Comando:

```
select nome from clientes where nome like 'João%'
```

Resultado:

nome

João da Silva Borges
João Augusto

Da mesma forma como podemos pesquisar por informações que comecem por um determinado conjunto de caracteres, também podemos definir que certas letras estejam no final ou em qualquer posição do texto, como mostramos no exemplo a seguir, que retorna todos os contatos que tem a string 'Silva' no meio do nome.

Comando:

```
select nome from clientes where nome like '%silva%'
```

Resultado:

nome

João da Silva Borges

12.2 Predicados BETWEEN... AND e NOT BETWEEN... AND

O PREDICADO **Between... and..** define uma faixa de valores na qual o campo precisa estar incluído.

Sintaxe:

```
Select * from [nome da tabela] WHERE [campo da tabela] {NOT} BETWEEN  
[valor ou texto] AND [valor ou texto];
```

Comando:

```
select Nome, Estoque from produtos where estoque between 500 and 1000
```

Resultado:

Nome	Estoque
-----	-----
Leite	1000

12.3 Predicados IN e NOT IN:

O predicado IN compara um valor com uma lista ou coleção de valores. Lembre-se que os critérios do tipo texto devem ser escritos entre aspas simples.

Sintaxe:

```
Select * from [nome da tabela] WHERE [campo da tabela] {NOT} IN (valor ou  
texto, valor ou texto,...)
```

Comando:

```
select Nome, Pais from clientes where pais in ('Brasil', 'Argentina')
```

Resultado:

Nome	País
-----	-----
	Brasil
Empresa A	Brasil
Empresa B	Brasil
Empresa C	Argentina
Empresa X	Brasil

12.4 Predicado IS NULL e IS NOT NULL

Separa os registros NULOS. Quando precedido por um NOT, separa os registros com valor válido.

Sintaxe:

```
select * from [nome da tabela] WHERE [nome do campo] {IS NOT NULL} {IS NULL};
```

Este predicado pode ser útil para determinadas situações em que algum campo esteja *null* porque não foi processado ainda, por exemplo, numa compra que não foi finalizada, ou em um pedido que não foi entregue. Observe o exemplo abaixo, que é um exemplo típico de uma consulta utilizando este predicado para identificar pedidos ainda não enviados.

Comando:

```
select * from pedidos where DataEnvio Is Null
```

Analogamente, para verificarmos quais entradas possuem um valor de DataEnvio, e, portanto representam pedidos já enviados, podemos utilizar o comando abaixo:

Comando:

```
select * from pedidos where DataEnvio Is Not Null
```

O predicado ISNULL(COALESCE em MySQL) também pode ser utilizado na exibição de algum valor caso um campo esteja nulo.

Comando:

```
select ISNULL(taxa, '') Cobrança from pedidos
```


12.5 Seleção com Operadores Lógicos

Para selecionar registros de uma tabela com a determinação de critérios o usuário deverá utilizar a cláusula WHERE combinada com os operadores lógicos AND, OR e NOT. A sintaxe é mostrada abaixo:

Sintaxe:

```
SELECT [nome do campo] FROM [nome da tabela] WHERE [nome do campo]
[operador de comparação ou restrição] [critério de comparação ou restrição] AND
[nome do campo] [operador de comparação ou restrição] [critério de comparação ou
restrição] OR [nome do campo] [operador de comparação ou restrição] [critério de
comparação ou restrição]
```

Como exemplo, vamos selecionar todos os produtos cujo estoque seja maior que 500 e tenham um preço menor que 10.

Comando:

```
select Nome, Estoque, Preço from produtos where estoque > 500 and preço <
10
```

Neste exemplo, a cláusula AND obrigou que ambas as condições fossem atendidas. Podemos também definir que o item seja selecionado caso qualquer das condições seja atendida, como mostra o exemplo abaixo em que selecionamos os itens que tenham. preço menor que 10 ou um estoque maior que 1000.

Comando:

```
select Nome, Estoque, preço from produtos where preço < 10 or estoque >
1000
```

Laboratório.

Para a realização dos exercícios abaixo utilize a tabela criada no exercício anterior

Os dados de todos os celulares com mp3 não nulo.

Liste modelo de todos os celulares com nome contendo “k”.

Liste nome de todos os modelos, junto com seu valor e marca que seu valor esteja entre R\$ 300 e R\$ 500.

Liste nome de todos os celulares, junto com seus valores, para todos os celulares que não sejam da “Nokia”.

Liste o modelo, marca e bateria do celular código 4, 6, 2.

Liste os celulares lançados a mais de 10 anos e que sua bateria dure entre 4 e 6 horas.

Liste todos os celulares com bateria com mais de 4 horas e menos de 6 horas

Liste 2 celulares com câmera com valor nulo

Liste as marcas de celular existentes em letra maiúscula e sem valores nulos

Liste celulares da marca “Apple” com bateria de mais e 3 horas e com valor entre a R\$ 700,00 e R\$ 2.000,00

Delete todos os celulares com bateria com 1, 3 ou 4 horas de duração.

Unidade 13 - Funções de Agrupamento e Ordenação

13.1 Obtendo Informações Estatísticas

Muitas vezes não queremos recuperar os dados armazenados no banco de dados, mas sim informações estatísticas sobre os mesmos, como totais e médias de valores. Para tanto utilizamos as chamadas funções de agrupamento.

A sintaxe das funções de agrupamento é mostrada abaixo:

Sintaxe:

```
SELECT {ALL} {DISTINCT} Função([nome do campo]) FROM [nome da tabela]
```

As funções de agrupamento utilizadas no SQL são: **COUNT()**, **AVG()**, **MAX()**, **MIN()** e **SUM()**.

Abaixo mostramos um exemplo simples, mas bastante completo do uso de todas estas funções para processar cálculos em cima de uma tabela de produtos, e um exemplo de retorno.

Comando:

```
select SUM(Estoque) as 'Total no Estoque', MAX(Preco) as 'Preço mais caro', MIN(Preco) as 'Preço mais barato', AVG(Preco) as 'Preço médio', COUNT(Preco) as 'Total de Registros' from produtos
```

Exemplo de Resultado:

Total no Estoque	Preço mais caro	Preço mais barato	Preço médio	Total de Registros
-----	-----	-----	-----	-----
3600	31	5	16.0000	4

As funções agregadas não podem ser utilizadas dentro de uma cláusula WHERE, porem é possível utilizar esta cláusula para restringir o número de linhas que serão consideradas no cálculo da resposta, como mostra o exemplo abaixo:

Comando:

```
SELECT AVG(preco) from produtos where preco > 10
```

Nesta consulta acima o que estamos obtendo é a média entre todos os produtos cujo preço é maior que 10.

13.2 Agrupamento Utilizando GROUP BY:

A cláusula **GROUP BY** agrupa os registros de acordo com uma condição especificada. A sintaxe é mostrada abaixo:

```
SELECT [nome do campo] FROM [nome da tabela] GROUP BY [nome do campo]
```

Abaixo mostramos um exemplo em que agrupamos os clientes por países para obtermos uma listagem de todos os países de clientes.

Comando:

```
select Pais from clientes group by pais
```

Resultado:

Pais

Argentina
Brasil

O resultado acima poderia ser obtido também com o uso do comando **DISTINCT**. Na realidade, geralmente o uso do comando **GROUP BY** só é adequado em conjunto com funções de agrupamento, como mostramos no exemplo a seguir:

```
select count(*) as 'Total', pais from clientes group by pais
```

Resultado:

Total	País
-----	-----
1	Argentina
4	Brasil

Como agrupamos por países, conseguimos utilizar o comando de **COUNT** e descobrir quantas vezes cada um dos países aparece.

13.3 Utilizando WHERE e GROUP BY:

Podemos utilizar **WHERE** e **GROUP BY** juntos para selecionar registros de uma tabela, agrupando-os por um determinado campo. A sintaxe é apresentada abaixo:

```
SELECT [nome do campo] FROM [nome da tabela] WHERE [nome do campo]  
[operador de comparação ou restrição] [critério de comparação ou restrição]  
GROUP BY [nome do campo]
```

O comando abaixo mostra o uso de **COUNT** e **WHERE** para retornar o total de clientes com cargo do contato como sendo de gerente de vendas.

```
select count(*), pais from clientes where CargoContato = 'Gerente de
Vendas' GROUP BY pais
```

13.4 Seleção de registros com Agrupamento pela Cláusula GROUP BY e HAVING:

Enquanto a cláusula WHERE filtra os registros antes de agrupá-los, diminuindo assim a quantidade de registros gerados pela cláusula GROUP BY, a cláusula HAVING faz um filtro também, porém, permitindo a utilização de uma função agregada.

A sintaxe é mostrada abaixo:

```
SELECT [nome do campo] FROM [nome da tabela] GROUP BY [nome do
campo] HAVING Função Agregada([nome do campo])
```

Por exemplo, se quisermos ver o total de clientes de países cujo total for maior que 3, usaríamos o comando abaixo:

```
select count(*),pais from clientes group by pais having count(*) >=
3
```

Resultado:

Count(*)	País
-----	-----
4	Brasil

Como só existe uma referência a "Argentina", e portanto o total é menor que 3, não aparece no resultado.

Para entender melhor a diferença entre WHERE, GROUP BY e HAVING, é preciso compreender a definição de cada uma, mostrada abaixo:

a cláusula **WHERE** é utilizada para filtrar as linhas que resultam da consulta da tabela especificada pela cláusula **FROM**.

A cláusula **GROUP BY** é utilizada para agrupar as linhas filtradas por **WHERE**.

A cláusula **HAVING** é usada para filtrar as linhas do grupo criado por **GROUP BY**.

13.5 Ordenação das Consultas

Até agora não nos preocupamos com a ordem em que são mostrados os resultados de uma consulta, mas em situações reais normalmente vamos querer controlar esta ordenação. A cláusula **ORDER BY** ordena os registros em crescente e decrescente baseado em um campo da tabela, e sua sintaxe de uso é mostrada abaixo:

```
SELECT [nome do campo] FROM [nome da tabela] ORDER BY [nome do campo]
{ASC}{DESC}
```

Se não utilizarmos a cláusula **ORDER BY**, o resultado da instrução será baseado no índice da tabela analisada. Ao utilizarmos a cláusula estamos definindo a ordem, como no caso abaixo em que os produtos são mostrados em ordem alfabética:

Comando:

```
select Nome from produtos order by Nome
```

Não é necessário utilizar o complemento **ASC** para as ordenações crescentes, porém para as ordenações decrescentes deve-se utilizar **DESC**:

Laboratório.

Para a realização dos exercícios abaixo utilize a tabela criada no exercício anterior

Some os valores dos celulares.

Calcule o tempo médio de duração da bateria

Encontre o maior e o menor valor dos celulares.

Conte o número de celulares por Marca.

Encontre a média de valores por Marca.

Calcule a média de valor das Marcas de celular com mais de um exemplar.

Mostre as marcas de celular que tenham apenas 1 exemplar

Liste as marcas de celular existentes em letra maiúscula e sem valores nulos

Liste todos os celulares por em ordem decrescente de lançamento

Calcule a média de duração dos celulares com câmera

Unidade 14 - Consultas em Múltiplas Tabelas

14.1 Consultas em Múltiplas Tabelas

Conforme visto nos capítulos anteriores, de teoria de modelagem de dados, as tabelas do banco de dados têm relações entre si, e é essencial dominar os recursos da linguagem SQL que nos permitem criar consultas com base nestas relações.

O agrupamento de tabelas na SQL é muito mais do que uma “consulta”, é o fato de poder interligar duas ou mais tabelas relacionadas entre si de forma que possam ser recuperados os seus registros na resposta de uma instrução SQL.

Nem todos os bancos de dados utilizam a forma padrão de ANSI, existindo algumas variações na sintaxe do comando.

14.2 Join

Podemos definir Join como a junção de duas tabelas.

O resultado da união de duas tabelas será a multiplicação das linhas da primeira pela segunda tabela, ou seja, para cada linha da primeira tabela, todas as linhas da segunda serão repetidas.

Na prática esta situação não é muito comum, mas serve como base para entendermos o funcionamento de outros tipos de Join.

Sintaxe:

```
Select [nome da tabela1].[nome do campo], [nome da tabela2].[nome do campo] from [nome da tabela1], [nome da tabela2];
```

Observe a operação de JOIN entre duas tabelas pedido e clientes, sem a utilização de nenhuma restrição na união. Assumindo que existam 3 empresas e 3 pedidos, teremos um retorno com nova linha:

Comando:

```
select clientes.Nome, pedidos.Id as 'ID do pedido' from pedidos, clientes
```

Resultado:

Nome	ID do pedido
------	--------------

-----	-----
	1
	2
	3
Empresa A	1
Empresa A	2
Empresa A	3
Empresa B	1
Empresa B	2
Empresa B	3

O resultado é um produto cartesiano entre elas, em que o total de linhas será igual ao total de entradas de uma tabela multiplicado pelo total de entradas da outra.

Para obtermos um resultado que faça sentido, geralmente utilizamos a cláusula WHERE.

14.3 Equijoin – Join de Igualdade

Também conhecido como join simples ou inner join, o Equijoin fará a união entre duas ou mais tabelas através de uma cláusula WHERE, que ligará a chave primária de uma a chave estrangeira de outra.

Sintaxe:

```
Select [nome da tabela1].[nome do campo], [nome da tabela2].[nome do campo] from [nome da tabela1], [nome da tabela2] WHERE [nome da tabela1].[nome do campo] = [nome da tabela2].[nome do campo]
```

Vamos utilizar um inner join para obter uma lista dos produtos e suas respectivas categorias. Vamos assumir que em uma tabela produtos temos o código da categoria, e esta informação - a chamada chave estrangeira - nos permite descobrir dentro da tabela categoria o nome da categoria daquele produto.

Comando:

```
select produtos.Nome as 'Nome Produto', categorias.Nome as 'Categoria' from produtos, categorias where produtos.CodCategoria = categorias.id
```

Resultado:

Nome Produto	Categoria
--------------	-----------

-----	-----
Leite	Bebidas
Suco	Bebidas
Carne	Carnes
Pão	Outros Alimentos

14.4 Exercícios Sugeridos

Anteriormente foi apresentada a tabela clientes, com atributos Nome e País. Transforme esta informação em duas tabelas, Novo_Clientes e Países, definindo os atributos que fazem a relação entre as tabelas, e defina uma consulta SQL que retorne o mesmo resultado da tabela Clientes original, porém consultando as duas novas tabelas

Tabela Clientes

Nome	País
	Brasil
Empresa A	Brasil
Empresa B	Brasil
Empresa C	Argentina
Empresa X	Brasil

14.5 Equijoins e Operadores Lógicos:

É possível combinar em uma cláusula WHERE com equijoins os operadores lógicos: AND, OR e NOT. Isto será necessárias em boa parte das consultas SQL feitas em um sistema, pois geralmente queremos definir condições exatas para a recuperação de dados de diversas tabelas.

Sintaxe:

```
Select [nome da tabela1].[nome do campo], [nome da tabela2].[nome do campo]
from [nome da tabela1], [nome da tabela2] WHERE [nome da tabela1].[nome do campo] = [nome da tabela2].[nome do campo] AND [nome da tabelaN].[nome do campo] [Comparação] [valor ou 'texto'] OR [nome da tabelaN].[nome do campo] [Comparação] [valor ou 'texto'] NOT [nome da tabelaN].[nome do campo] [Comparação] [valor ou 'texto']
```

Como exemplo, vamos criar uma consulta em tabelas 'produtos' e 'categorias', selecionando produtos cujo preço é maior que 10 e ainda exibindo a categoria a que pertencem:

Comando:

```
select produtos.Nome as 'Nome Produto', categorias.Nome as 'Categoria' , Preco
from produtos, categorias where produtos.CodCategoria = categorias.id and produtos.preco > 10
```


14.6 Equijoins entre mais de duas tabelas

Na maioria dos bancos de dados relacionais as tabelas relacionam-se entre si em mais de um nível. Vamos definir uma consulta entre tabelas pedidos, produtos e categorias para exemplificar este recurso.

```
Select [nome da tabela1].[nome do campo], [nome da tabela2].[nome do campo], [nome da tabela3].[nome do campo] from [nome da tabela1], [nome da tabela2], [nome da tabela3] WHERE ...
```

Comando:

```
select pedidos.id as 'Id do Pedido', produtos.nome as 'Nome do Produto', categorias.nome as 'Categoria' from pedidos, produtos, categorias where produtos.codcategoria = categorias.id AND pedidos.idProduto = produtos.id
```

Em situações reais, muitas vezes serão criadas consultas entre diversas tabelas, geralmente com a ligação entre cada tabela e as demais sendo feita através de uma igualdade que ligue a chave primária de uma tabela com a chave estrangeira da outra.

14.7 Redefinição do Nome de Tabelas

Para ‘facilitar’ a vida do administrador da base de dados, não é necessário que este digite o nome das tabelas em sua totalidade. É possível criar um alias para cada tabela, diminuindo, assim, a quantidade de caracteres digitados. O exemplo abaixo ilustra esta técnica.

Comando:

```
select PE.id as 'Id do Pedido', PR.nome as 'Nome do Produto', C.nome as 'Categoria' from pedidos as PE, produtos as PR, categorias as C where PR.codcategoria = C.id AND PE.idProduto = PR.id
```

14.8 Outros Tipos de Joins - Outer Joins

O resultado de uma instrução SQL que utiliza **Equijoin**, exibe os resultados que são comuns entre os campos das duas tabelas. Neste método, ficam de fora as linhas que não encontram relação entre as duas tabelas. Porém esta informação de entradas que não aparecem num InnerJoin também pode ser útil. Por exemplo, para saber se existe algum Cliente que nunca tenha feito um pedido.

Left e Right Join

Para que a questão acima seja respondida, é necessário entender que existe um lado da relação que irá listar seus registros, baseado no **Left** ou **Right**.

Em algumas plataformas é utilizado o * (asterisco) para determinar o lado do outer join. Em outras é utilizado o (+) (sinal de adição entre parênteses). Ambos utilizados em locais diferentes. Na maioria dos casos é utilizado o comando LEFT JOIN que é aceito pelo MySQL e será utilizado nos exemplos abaixo.

Sintaxe do Left Join:

```
Select [nome da tabela1].[nome do campo], [nome da tabela2].[nome do campo] from [nome da tabela1], [nome da tabela2] WHERE [nome da tabela1].[nome do campo] * = [nome da tabela2].[nome do campo]
```

ou

```
Select [nome da tabela1].[nome do campo], [nome da tabela2].[nome do campo] from [nome da tabela1], [nome da tabela2] WHERE [nome da tabela1].[nome do campo] (+) = [nome da tabela2].[nome do campo]
```

ou

```
Select [nome da tabela1].[nome do campo], [nome da tabela2].[nome do campo] from [nome da tabela1] LEFT JOIN [nome da tabela2] ON [nome da tabela1].[nome do campo de relação] = [nome da tabela2].[nome do campo de relação]
```

Apenas a terceira sintaxe é aceita pelo MySQL, por exemplo.

Iremos inicialmente utilizar um comando LEFT JOIN para identificar clientes que não tem nenhum pedido. Para tanto teremos que restringir o retorno àquelas linhas em que não existe uma ligação com a tabela 'pedidos' (portanto as entradas da tabela 'pedidos' retornam null):

```
select clientes.nome from clientes left join pedidos on clientes.Id = IdCliente where pedidos.id is null
```

Observação: o comando RIGHT JOIN não é implementado em MySQL, mas invertendo-se as tabelas pode-se obter efeito equivalente.

14.9 Outros Tipos de Joins - Self Joins

Self Joins são relações estabelecidas entre uma mesma tabela, por intermédio de dois de seus campos. São utilizadas normalmente quando a tabela referencia a si mesmo para definir uma hierarquia, por exemplo entre funcionários. A única diferença para um Inner Join comum é que repetimos a mesma tabela com dois *alias* diferentes.

Abaixo é mostrada a sintaxe de um **self join**.

Sintaxe:

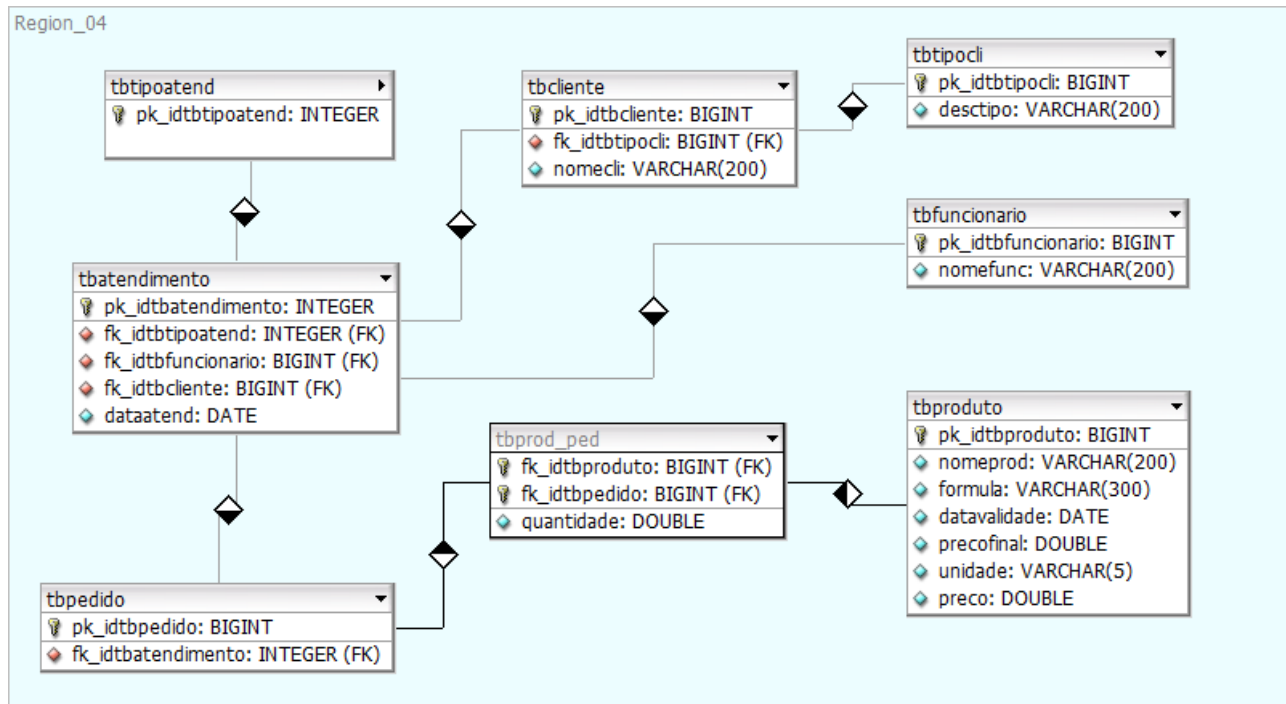
```
Select * from [nome da tabela].[nome do campo1], [nome da tabela].[nome do campo2] where [nome da tabela].[nome do campo1]= [nome da tabela].[nome do campo2];
```

Supondo uma tabela 'funcionarios' em que existe um campo que referencia a própria tabela, o campo 'chefe', que contém o Id do chefe de cada funcionário. Através de um self join podemos listar todos os funcionários que possuem um chefe, indicando quem é o mesmo:

```
select F2.Nome as 'funcionário', 'é subordinado a', F1.Nome as 'chefe' from funcionarios as F1, funcionarios as F2 where F2.Chefe = F1.Id
```

Laboratório

Crie a seguinte estrutura de tabelas, para executar os exercícios



- Liste o nome e a quantidade de atendimentos por cliente
- Liste o nome e a quantidade de atendimentos por funcionário
- Conte quantos produtos estão em cada pedido
- Calcule a média de valor dos pedidos de cada cliente
- Calcule o valor total de cada pedido
- Mostre a quantidade, total e média de pedidos por data
- Liste a média de pedidos que cada produto participou
- Liste o produto mais vendido por cada vendedor
- Mostre quantidade de pedidos por tipo de cliente e tipo de pedido
- Exiba todos os produtos que não tem pedido
- Liste os produtos por cliente que não foram solicitados por eles
- Liste quantos pedidos cada vendedor faz por data

Unidade 15 - Trabalhando com datas

15.1 Datas em MySQL

A forma de trabalhar com datas pode variar um pouco em diferentes bancos de dados. Aqui apresentamos recursos tendo como base o MySQL.

15.2 Recuperando Informações Específicas de Data

Diversas funções permitem recuperar informações específicas com base em uma data, como o dia, mês e ano de determinada data, como mostrado abaixo:

DAY OF MONTH

Seleciona o dia de uma data:

```
select DataPedido, dayofmonth(DataPedido) from pedidos
```

MONTH

Seleciona o mês de uma data:

```
select DataPedido, month(DataPedido) from pedidos
```

YEAR

Seleciona o ano de uma data:

```
select DataPedido, year(DataPedido) from pedidos
```

Formato das Datas

Para definir diretamente uma data, por exemplo para um cálculo ou para inserir no banco de dados, o formato é um texto (separado por aspas) na forma "ano-mês-dia".

O exemplo abaixo retorna o ano do dia 19 de maio de 1980:

Exemplo:

```
select year('1980-05-19')
```

Unidade 16 - Principais Bancos de Dados

O dinamismo da área de tecnologia da informação nos proporciona contatos cada vez mais frequentes com uma infinidade de modelos de banco de dados. A cada dia vemos a notícia do lançamento de uma nova plataforma de gerenciamento de dados.

No entanto alguns bancos de dados obtiveram maior notoriedade no mundo da informática. Dentre estes bancos cabe destacar o Microsoft Access, que foi um dos precursores dos SGBDs.

Sua inovação em termos de capacidade de armazenamento e dinamismo foi o principal responsável pelo seu enorme sucesso. Entretanto com a popularização da Internet, e por sua vez, da distribuição das informações em redes de grande alcance, suas limitações foram ficando evidentes. Abriu-se então espaço para os chamados “softwares livres”.

Esta nova realidade proporcionou o aparecimento do MySQL, PostgreSQL e tantos outros. Por sua maior capacidade de desempenho em um ambiente compartilhado, estes sistemas ganharam força na internet, sendo responsáveis por um grande mercado de sites pessoais e de pequeno-médias corporações.

Quando nos referimos a grandes corporações, que dependem criticamente do controle de seus dados, nos deparamos com a necessidade de bancos de dados que trabalhem, com recursos mais avançados, tais como replicação, serviços de integração, business intelligence, etc.

Neste contexto encontramos como grandes expoentes o SQL SERVER e o ORACLE além de outros com menor expressão. O SQL SERVER tem como grande diferencial ser da Microsoft o que faz com que sua integração com linguagens .Net ocorra de forma natural.

Nas próximas unidades de nosso material iremos trabalhar com 1 destes SGBDs. Trataremos das principais características e formas de manipulação do MYSQL.

Unidade 17 - MySQL



17.1 Introdução

O MySQL é mais um exemplo de SGBD que utiliza a linguagem SQL como forma de interagir com os dados armazenados. Obviamente, possui algumas sintaxes de comandos que mesmo provenientes da linguagem SQL são específicas deste banco. É um dos bancos de dados mais utilizados por empresas e usuários pessoais.

Podemos citar vários grandes exemplos de usuários deste sistema de gerenciamento de banco de dados, mas o maior case de sucesso, não poderia deixar de ser outro: o Google.

Este sistema teve seu lançamento na década de 1980, de lá para cá o número de usuários e pessoas responsáveis por sua manutenção e testes só tem aumentado. Além disso uma série de sistemas satélite vem sendo desenvolvidos para agregar cada vez mais funcionalidades e este bando.

Inicialmente o MySQL era um sistema que pertencia a uma empresa que o ofertava livremente, sem a intenção de gerar concorrência. No entanto a sua grande expansão fez com que grandes corporações como a SUN a adquirisse. Ironicamente a SUN foi adquirida pela ORACLE o que criou quase que uma “guerra fria” de bancos de dados, tendo a ORACLE de um lado e a Microsoft de outro.

Este grande ganho de mercado que o MySQL obteve tem como um dos grandes fatores a integração com uma das linguagens de programação de maior sucesso na Internet, o PHP. Esta configuração de uso do Linux(Sistema Operacional), MySQL, PHP e o Apache(Servidor), criou uma sigla conhecida como LAMP, que é uma das opções mais aceitas pelos usuários na hora de hospedar seus sites. Também é comum a visualização da sigla WAMP, que nada mais é do que troca do sistema operacional Linux pelo Windows.

Em seu princípio o MySQL era considerado um SGBD fraco e de poucos recursos. No entanto esta realidade mudou drasticamente nos últimos anos e ele já é utilizado na gestão de dados com grande volume de acesso como o Google e a Wikipédia além de ter sua utilização empregada em instituições que lidam com a segurança e precisão da informação como fatores determinantes de seu negócio. Podemos citar como exemplo a NASA e o exército Americano, além de exemplos nacionais como o banco Bradesco e tantas outras instituições financeiras.

Unidade 18 - O Front-End

18.1 Alguns Exemplos

Manipular os dados armazenados em um banco de dados requer o uso de ferramentas específicas para esta operação. Estes sistemas são os chamados fron-ends.

A instalação completa do MySQL acompanha o aplicativo desktop chamado MySQL Query Browser:



Este sistema é bem completo e oferece suporte a linguagem, bem como interface amigável ao banco em si.



Além desta possibilidade nativa, possuímos uma série de outros sistemas desktop, cabendo um destaque ao Navicat.

Entretanto como a maior popularização do MySQL se deu pelo grande casamento com a linguagem para web(PHP), não poderia deixar de existir um sistema nesta linguagem e que o mesmo fosse um dos mais utilizados.

Estamos nos referindo ao phpMyAdmin



Em nosso material vamos citar exemplos de manipulação de dados utilizando o phpMyAdmin bem como do MySQL Query Browser

Unidade 19 - Conectando ao Banco de Dados

19.1 Através de linguagens de programação

Um banco de dados tem por objetivo maior ser um local de armazenamento de informações para ser acessada a qualquer momento e preferencialmente de qualquer local. No entanto seu uso mais comum está ligado à integração com sistemas web e desktop.

A banco de dados MySQL, como já referido anteriormente, tem uma forte ligação com a linguagem PHP, mas pode, interagir com qualquer outro sistema independente de sua linguagem.

Para estabelecer esta conexão as linguagens se utilizam de scripts próprios.

Exemplo de script em PHP para conexão com MYSQL:

```
<?
// Este arquivo conecta um banco de dados MySQL - Servidor = localhost
$dbname="teste"; // Indique o nome do banco de dados que será aberto
$usuario=" "; // Indique o nome do usuário que tem acesso
$password=" "; // Indique a senha do usuário
//1º passo - Conecta ao servidor MySQL
if(!($id = mysql_connect("localhost",$usuario,$password))) {
    echo "Não foi possível estabelecer uma conexão com o gerenciador
MySQL. Favor Contactar
o Administrador.";
    exit;
}
//2º passo - Seleciona o Banco de Dados

if(!($con=mysql_select_db($dbname,$id))) {

    echo "Não foi possível estabelecer uma conexão com o gerenciador
MySQL. Favor Contactar
o Administrador.";
    exit;
}
?>
```

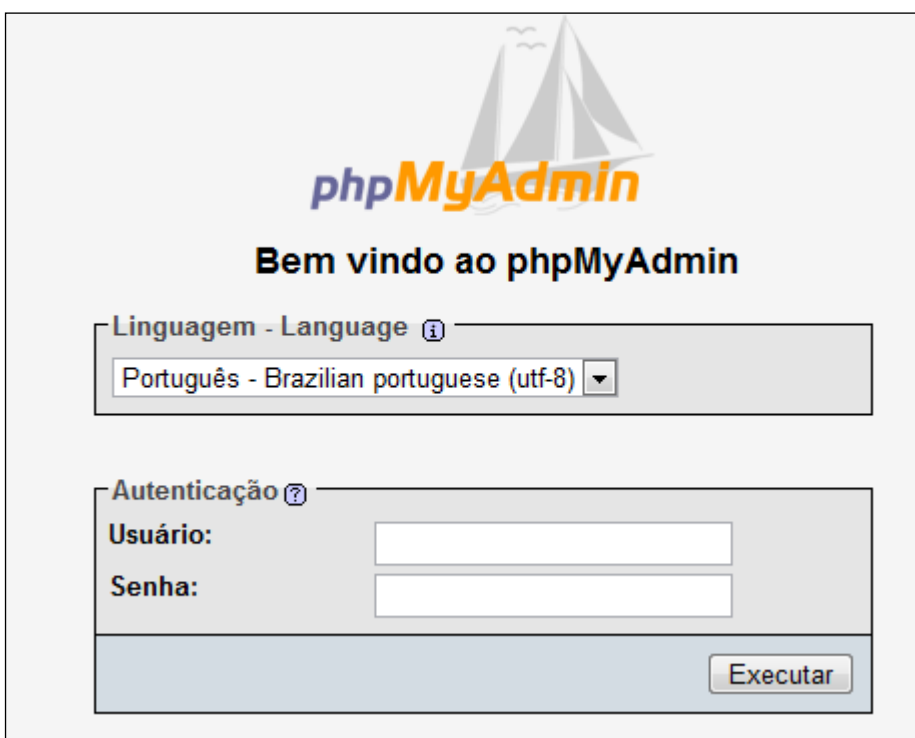
Além deste exemplo existe outras linguagem que possuem seu próprio script de conexão

19.2 Através do Front-End

Para acessarmos nosso banco de dados, com a intenção de realizar manutenções, extração de informações, gerenciamento de performance, criação de regras, etc. podemos fazê-lo de forma mais rápida e dinâmica através de nossos fron-ends.

O phpMyAdmin pode ser acessado através do endereço IP e porta fornecidos pelo provedor onde o site de encontra hospedado. Esta informação ou link, normalmente encontra-se disponíveis em painéis de controle para sites fornecidos por estes provedores de hospedagem.

Será exibida uma tela solicitando o usuário e senha que serão utilizados nesta conexão. Abaixo podemos ver um exemplo desta tela.

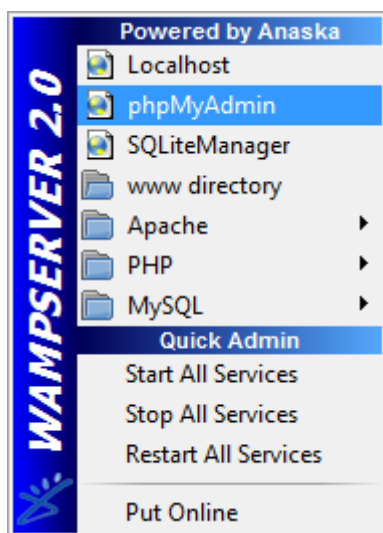


The image shows the phpMyAdmin login interface. At the top, there is a logo featuring a sailboat and the text 'phpMyAdmin'. Below the logo, it says 'Bem vindo ao phpMyAdmin'. There are two main sections: 'Linguagem - Language' with a dropdown menu showing 'Português - Brazilian portuguese (utf-8)', and 'Autenticação' with input fields for 'Usuário:' and 'Senha:'. An 'Executar' button is located at the bottom right of the authentication section.

No caso de instalação local basta digitar <http://localhost/phpmyadmin/> em seu navegador preferido, ou utilizar algum sistema para trabalhar com PHP, como por exemplo, o WAMP ou mesmo o easyPHP.

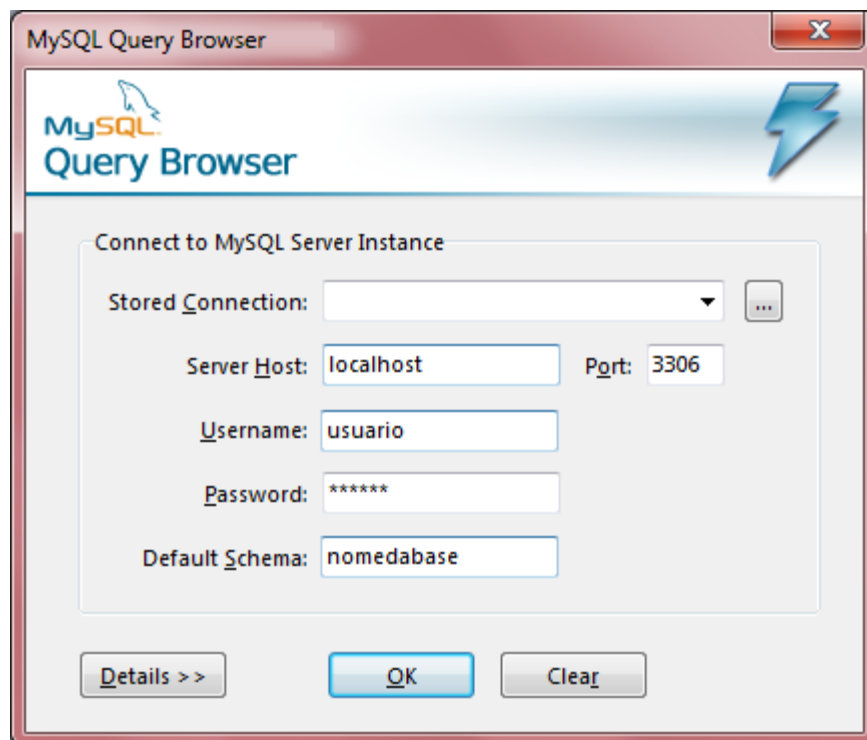
Estes sistemas permitem o acesso de forma mais rápida e sem a necessidade de informar usuário e senha a todo o momento

Vela uma tela de como acessar o phpMyAdmin pelo WAMP.



Por sua vez o MySQL Query Browser é um software desktop, portanto, fica instalado na máquina que irá acessar o banco. Este programa pode ser facilmente baixado através do site da MYSQL ou em local próprios para download de aplicativos, tais como o superdownloads e o baixaki.

A acessar este sistema a seguinte tela será exibida:



Nesta tela deverá ser informado;

- O IP e a porta de conexão(normalmente 3306) com o servidor(localhost se for localmente)
- O usuário(normalmente o mesmo do painel de controle ou root se for localmente)
- A senha(definida pelo usuário)
- Base de dados padrão(A base do projeto)

Unidade 20 - Conhecendo a Interface

20.1 phpMyAdmin

Ao acessar o phpMyAdmin nos é exibida um tela com as principais funcionalidades do bando de dados. Veja uma tela de exemplo abaixo:



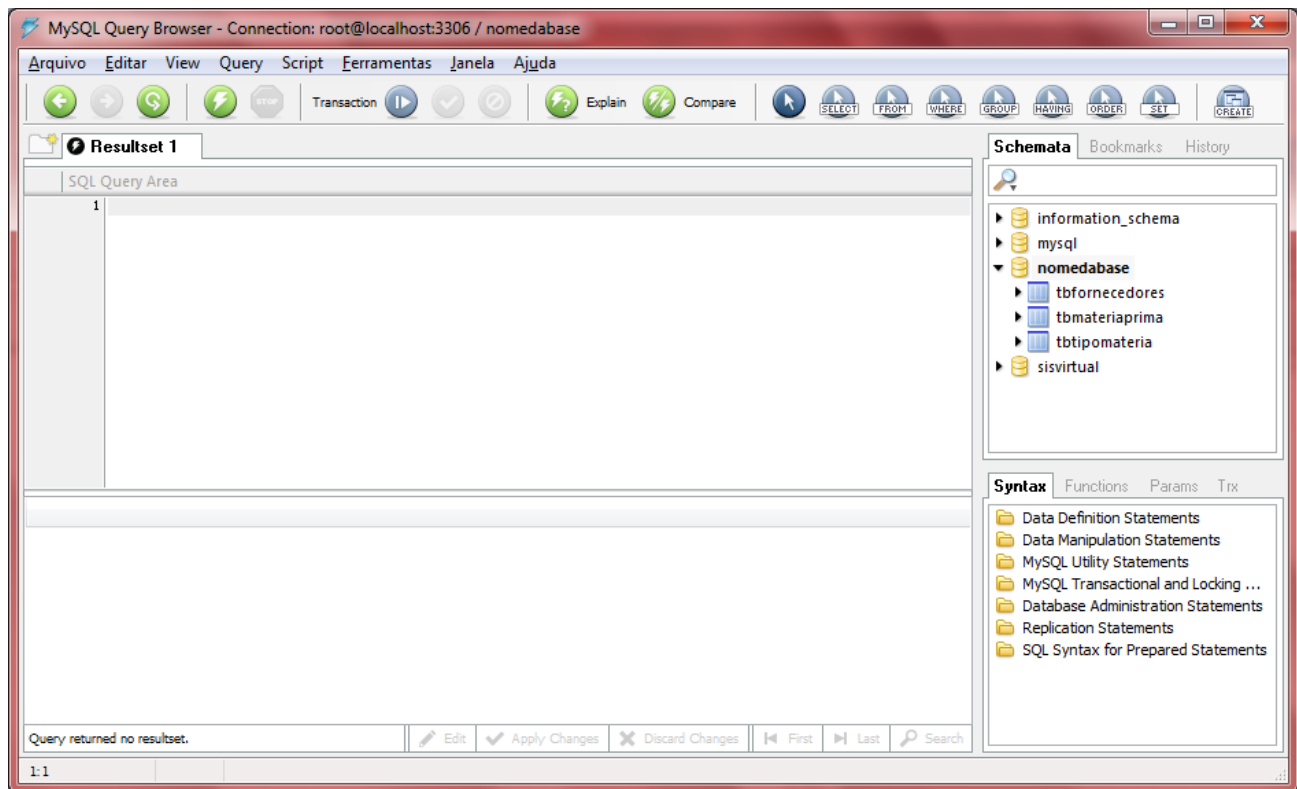
Esta tela nos mostra algumas informações sobre a configuração do servidor em que o banco está hospedado bem como estatísticas sobre o funcionamento do banco de dados

20.2 MySQL Query Browser

O MySQL Query Browser possui uma tela muito simples e intuitiva para sua utilização. Diferentemente do phpMyAdmin esta interface não exibe inicialmente informações sobre o servidor, mas sim, traz uma caixa de texto para a digitação de sentenças SQL.

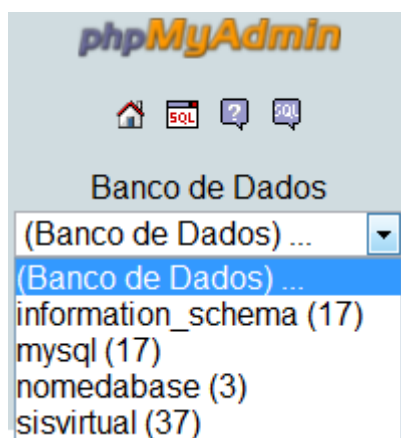
Também podemos observar à existência de botões que representam sintaxes da linguagem SQL, que são utilizados para tornar mais ágil a criação dos scripts.

Veja uma tela de exemplo deste layout inicial

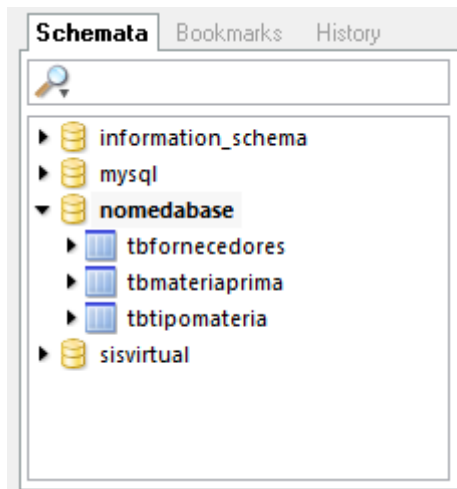


20.3 Acessando uma Base de Dados

No phpMyAdmin possuímos na parte superior esquerda da tela uma lista das bases de dados disponíveis, basta clicar sobre a mesma



No MySQL Query Browser localizamos estas mesmas opções no canto superior direito:




Unidade 21 - Criando uma nova Base de Dados

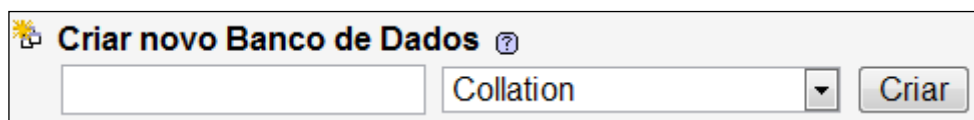
21.1 Interface e Comando

Para criar uma nova base de dados, bem como para a execução de qualquer consulta, deleção, ou outra atividade em nosso banco de dados, podemos utilizar de instruções SQL digitadas diretamente ou nos fazer valer de front-ends.

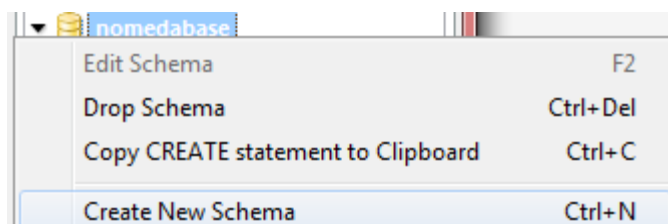
No caso da criação de uma nova base de dados utilizáramos a sentença:

```
CREATE DATABASE nomedatabase
```

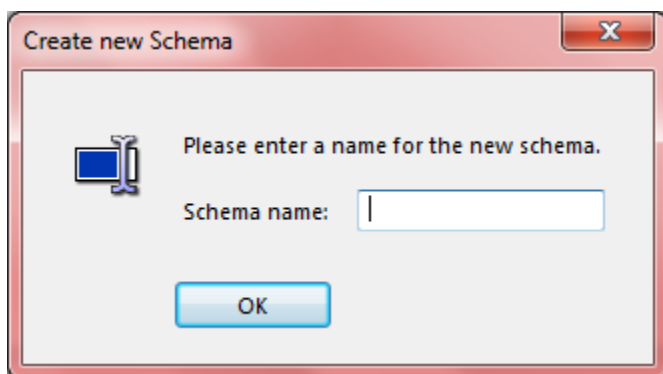
Pelo phpMyAdmin realizamos esta operação, estando na tela inicial (clicar no botão Home - ) basta digitar o nome da nova base e clicar em “Criar”



No MySQL Query Browser clicamos com o botão direito sobre uma base qualquer e escolhemos a opção “Create New Schema”



Por fim digitamos o nome do banco e clicamos em “OK”




Unidade 22 - Criando Tabelas

22.1 Interface

A criação de tabelas com a utilização de scripts já foi vista em unidades anteriores, então vamos nos deter na facilidade da criação de tabelas com a interação da interface dos front-ends

phpMyAdmin








Com a base de dados selecionada basta digitar o nome da nova tabela e o número de campos que a mesma terá (número de arquivos)

 Criar nova tabela no Banco de Dados **nomedabase**

Nome: Número de arquivos:

Logo após este passo será exibida uma tela para as devidas configurações:

Servidor: localhost ▶ **Banco de Dados: nomedabase** ▶ **Tabela: novatabela**

Campo	Tipo 	Tamanho/Definir ¹	Collation	Atributos	Nulo
<input type="text"/>	VARCHAR 	<input type="text"/>	<input type="text"/>	<input type="text"/>	not null 
<input type="text"/>	VARCHAR 	<input type="text"/>	<input type="text"/>	<input type="text"/>	not null 
<input type="text"/>	VARCHAR 	<input type="text"/>	<input type="text"/>	<input type="text"/>	not null 

Nome do campo






Tipo de dado

Tamanho do Campo

Formatação Campo

Valores Nulos

Opções numéricas ou update

Padrão ²	Extra						Comentários
<input type="text"/>	<input type="text"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="checkbox"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="checkbox"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="checkbox"/>	<input type="text"/>

Valor padrão

Autoincremento

Chave Primária

Índice

Único

Vazio

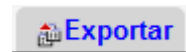
Texto Completo

Comentário

The image shows a database creation interface with three main sections at the top: 'Comentários da tabela:', 'Storage Engine:', and 'Collation:'. Below these are input fields and dropdown menus. Labels with leader lines point to these elements: 'Comentários' points to the text area under 'Comentários da tabela:'. 'Tipo de tabela' points to the 'InnoDB' dropdown under 'Storage Engine:'. 'Formatação' points to the empty dropdown under 'Collation:'. Below these sections is a row with three buttons: 'Salvar', 'Ou Adicionar', and 'Executar'. A text input field containing the number '1' is located between 'Ou Adicionar' and 'campo(s)'. Labels point to these elements: 'Salvar' points to the 'Salvar' button. 'Incluir campos' points to the text input field with '1'. 'Executar' points to the 'Executar' button.

O preenchimento das informações solicitadas nos campos, na realidade gera um script em SQL para a criação da referida tabela.

Podemos verificar os scripts criados para a criação de tabelas bem como para inserção do dados das mesmas através do botão “exportar”

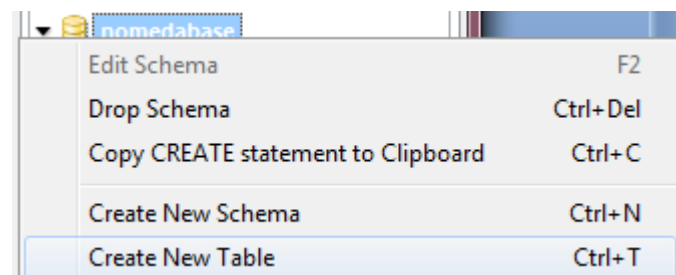


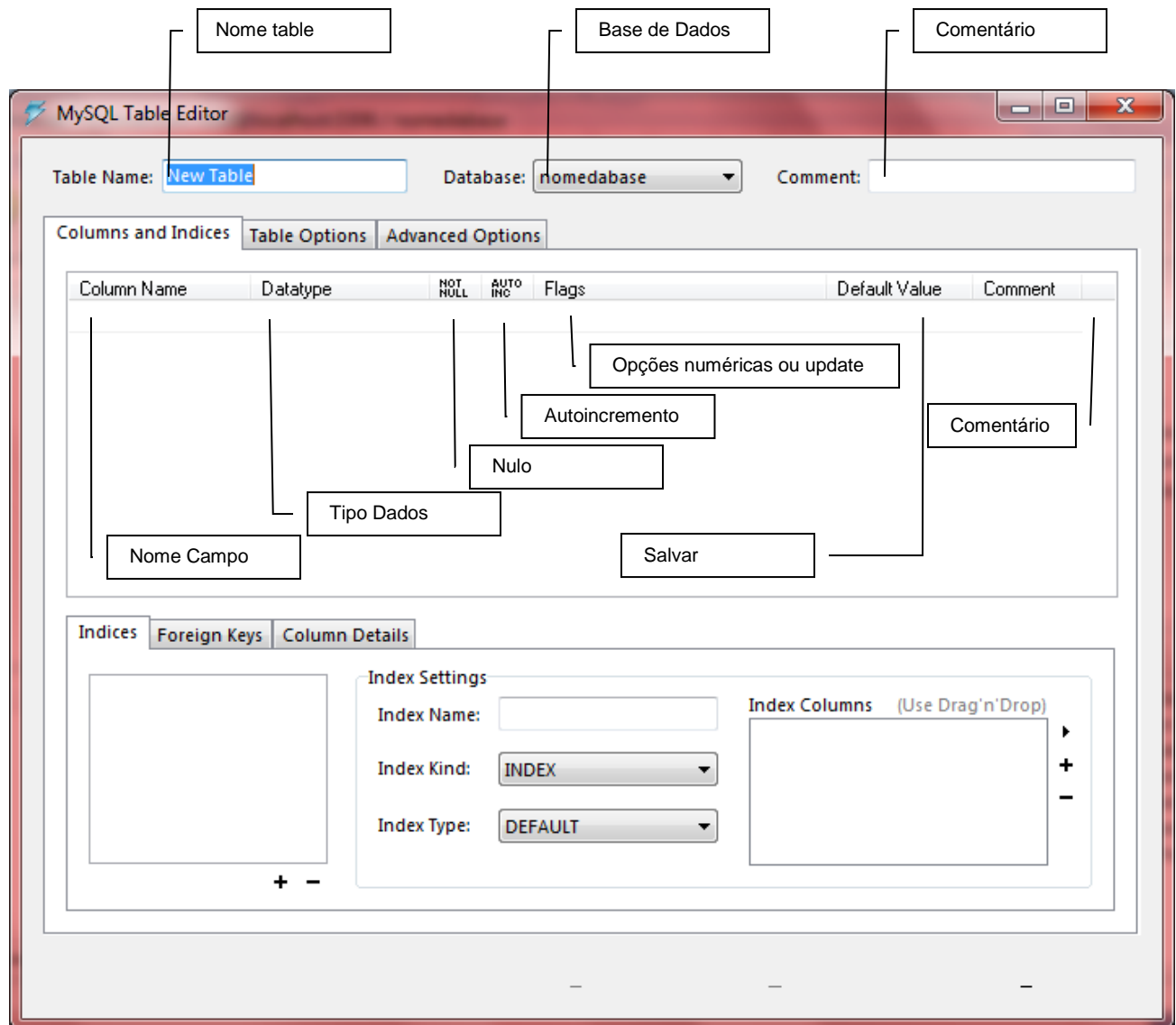
Esta opção é muito útil quando precisamos criar uma série de tabelas que desenvolvemos localmente para um servidor remoto.

Além disto este script, também é utilizado para a “instalação” de sistemas web baseado em banco de dados MySQL

MySQL Query Browser

Para o MySQL Query Browser o processo é simples, basta clicar com o botão direito sobre a base de dados na qual desejamos que a tabela seja criada.







Além da informações apontadas nas caixas de texto cabe destacar que na parte inferior da tela é exibida a chave primária selecionada bem como os índices. Além disso também encontramos mais detalhes sobre a configuração de cada campo.

Unidade 23 - Criando Consultas

23.1 Interface

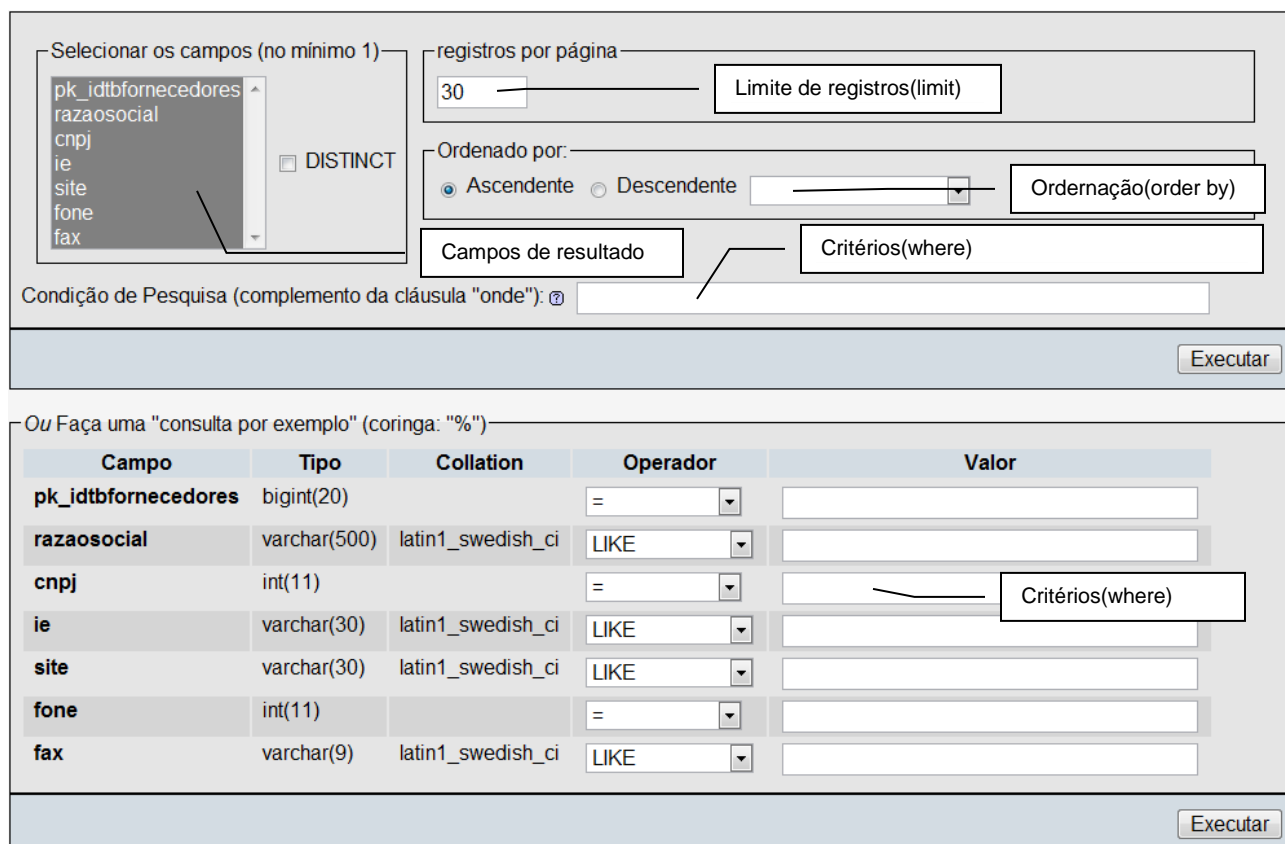
A exemplo da criação de bases de dados e tabelas, no caso das consultas, também podemos realizá-las por scripts ou via interface. Vejamos como isso funciona em nossos front-ends

phpMySQL

Com sua base de dados selecionada, a lista com as tabelas é exibida a direita. Ao lado de cada tabela temos o botão de suas propriedades(). Para a execução de um script direto em SQL basta clicar na aba  SQL.

Para utilizar um “assistente” na criação de suas consultas clique na aba  Procurar

A seguinte tela será exibida;



Selecionar os campos (no mínimo 1)

registros por página: 30 Limite de registros(limit)

Ordenado por: Ascendente Descendente Ordernação(order by)

Campos de resultado Critérios(where)

Condição de Pesquisa (complemento da cláusula "onde"): ?

Executar

Ou Faça uma "consulta por exemplo" (coringa: "%")

Campo	Tipo	Collation	Operador	Valor
pk_idtbfornecedores	bigint(20)		=	
razaosocial	varchar(500)	latin1_swedish_ci	LIKE	
cnpj	int(11)		=	
ie	varchar(30)	latin1_swedish_ci	LIKE	
site	varchar(30)	latin1_swedish_ci	LIKE	
fone	int(11)		=	
fax	varchar(9)	latin1_swedish_ci	LIKE	

Critérios(where)

Executar

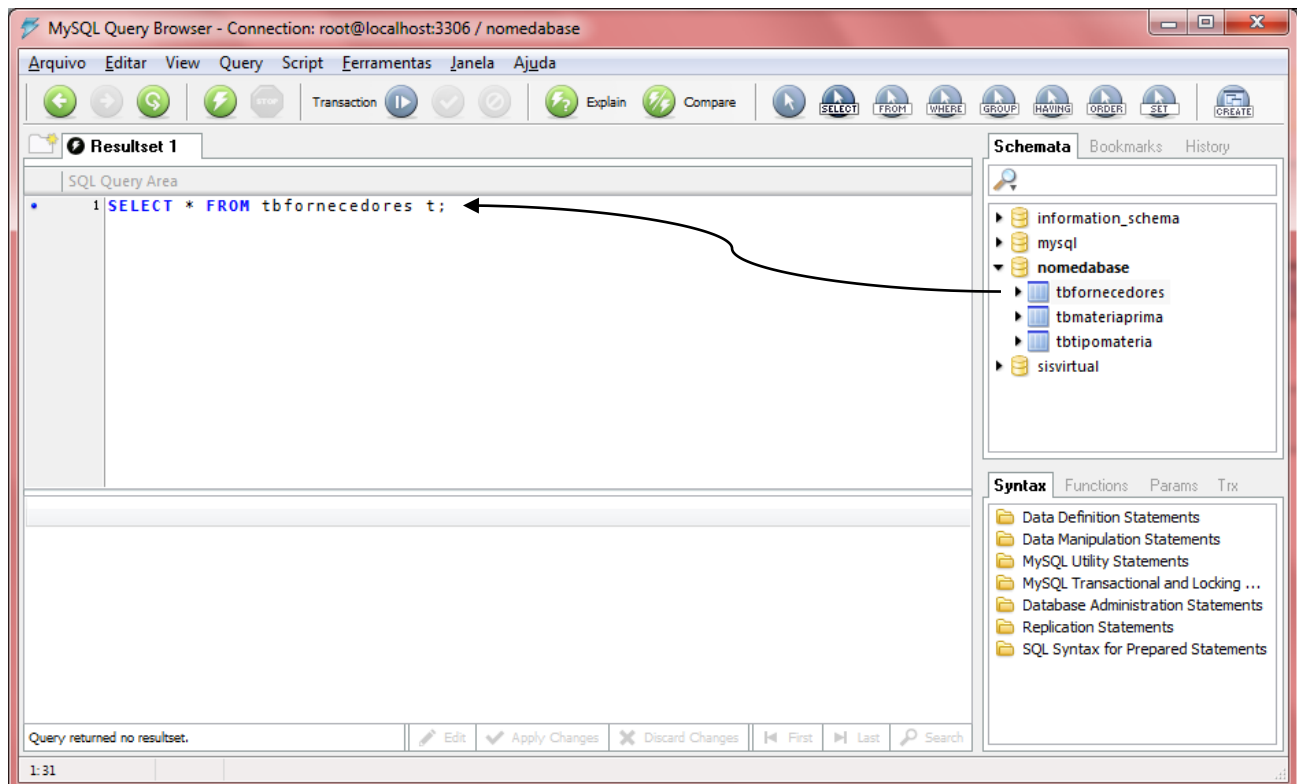
MySQL Query Browser

No Query Browser possuímos botões que representam instruções em SQL. A clicar nestes botões e após em um campo ou tabela a expressão é montada na tela de consulta.

Botões que representam o SQL



Ao arrastar uma tabela veja o resultado



Unidade 24 - Exercícios Extras

Exercício01

Uma loja de roupas deseja criar um cadastro com suas peças. Cada peça possui um código que a identifica, uma descrição, um preço-unitário e uma quantidade em estoque. Deseja também manter um cadastro de seus clientes com nome, telefone (o cliente pode ter mais de um, ou nenhum), e as peças que ele já comprou. A data em que o cliente comprou a peça é guardada.

Exercício02

Uma empresa bancária mantém um cadastro com os dados dos seus cliente(identidade, nome, endereço{ rua, CEP e bairro} e telefone { o cliente pode ter mais de um}) e de sua contas (número da conta e saldo). Um cliente pode ter mais de uma conta no banco e uma conta pode ser de mais de um cliente (conta conjunta). O banco mantém também um cadastro com as suas agências (código e nome), em que cada agência pode ter mais de uma conta.

Exercício03

Desenhe o Modelo E-R que descreva a situação da Empresa Acme Problemas Ltda., que apresenta a seguinte estrutura:

- a) cada empregado é representado a partir das seguintes informações básicas: código-empresa, nome, endereço;
- b) cada departamento é representado por meio de: nome, código departamento;
- c) cada empregado chefia um ou mais departamentos;
- d) cada item vendido é representado pelo seu nome, preço, fornecedor, número do modelo (dado pelo fornecedor) e número interno do item (dado pelo responsável pelo estoque);
- e) cada fornecedor é descrito pelo seu nome, endereço, itens fornecidos ao estoque e preço;
- f) os clientes cadastrados tem os seguintes dados: nome, endereço e telefone;
- g) os representantes da empresa junto aos clientes são empregados;

Exercício04

Um banco de dados, utilizado por uma faculdade, usa um sistema de cadastro de inscrição de disciplinas que contém as informações aluno e inscrição. As seguintes informações devem estar incluídas.

Aluno: código de aluno, nome do aluno, ano da admissão e telefone de contato;

Inscrição: código do aluno, código da disciplina, nome da disciplina, código do curso, nome do curso e data da matrícula.

Projete o banco de dados para esses dados. Faça quaisquer suposições razoáveis sobre as dependências envolvidas.

Exercício05

Você acabou de fundar sua empresa de consultoria: Beija-Flor Consultoria. Seu primeiro trabalho é desenvolver um sistema para cadastro de clientes. Você recebeu uma lista com os dados que deverão compor o sistema. Com base nesta lista, normalize a estrutura de dados de acordo com as formas normais.

Lista de informações que deverão compor o sistema cadastro de clientes:

Nome

Nome do Pai

Nome da Mãe

Endereço

Telefone1

Telefone2

Número do Fax

Número do Celular

Telefone do trabalho

Data de Nascimento

Naturalidade

Nacionalidade

Endereço de correspondência

Nome do filho 1

idade do filho 1

Nome do filho 2

idade do filho 2

Nome do filho 3

idade do filho 3

Nome do filho 4

idade do filho 4

Nome do Cônjuge

Número do CPF

Número da carteira de identidade

Exercício06

De acordo com as regras , normalize as estruturas abaixo.

- Relação de Programadores:
 - Numero da Matrícula
 - Nome do Programador
 - Setor
 - Nível (1,2,3)
 - Descrição do Nível (1 - Júnior, 2 - Pleno, 3 - Sênior)

- Programas
 - Código do Programa
 - Nome do Programa
 - Tempo Estimado
 - Nível de Dificuldade (1, 2 ou 3)
 - Descrição da Dificuldade (Fácil, Médio, Difícil)

Regras do negócio:

- Um programa pode ser feito por mais de um Programador;
- Um programador pode fazer um ou mais programas;
- O Nível de dificuldade do programa depende do tempo estimado para a elaboração do mesmo;

Exercício07

Você deve representar usando o modelo lógico a situação descrita a seguir:

O Departamento de Vendas da Indústria Beleza Ltda, após estudos de mercado, verificou que para atingir seus objetivos seria necessário adquirir frota de veículos próprios para motorizar seus vendedores. O mercado consumidor foi dividido em regiões de venda. Foram estabelecidos percursos de entrega abrangendo pontos estratégicos dessas regiões e vendedores foram designados para cobrir estes percursos. Um sistema deve ser construído para administração da nova sistemática de vendas adotada pela empresa. Após entrevistas com o gerente da área, foram obtidas as seguintes informações:

- cada região é identificada por um código;
- uma região é composta de vários pontos estratégicos;
- as regiões não têm pontos estratégicos em comum;
- o vendedor tem a responsabilidade de cobrir uma região;
- uma região pode ser coberta por vários vendedores;
- a cada dia, um veículo fica sob responsabilidade de um vendedor;
- um vendedor pode vender quaisquer itens ativos da tabela de produtos;
- o vendedor é responsável pela identificação de cada cliente consumidor na nota fiscal;
- a nota fiscal contendo identificação do vendedor, itens e quantidades vendidas é exigida para comprovação da venda.