



Banco de Dados

Esta apostila é dirigida a estudantes que queiram ter uma visão unificada de modelagem de dados, projeto de bancos de dados relacionais e programação na linguagem SQL. Traz vários exemplos e exercícios para que o estudante possa iniciar seus estudos na área de Banco de Dados.



Sobre este material

Esta apostila tem como objetivo auxiliar os estudantes de escolas técnicas, na aprendizagem da disciplina Banco de Dados, presente nos cursos técnicos da área de Informática e Tecnologia da Informação. Esta apostila não substitui os livros, sua finalidade é criar um roteiro resumido do ensino-aprendizagem realizado em sala de aula.

Este material foi construído a partir de slides de aulas, apostilas, tutoriais, anotações, dicas e demais fontes de dados, obtidos a partir de pesquisas em sites de buscas na Internet. Além disto, este material tem como referência a seguinte bibliografia:

DAMAS, Luís. SQL, Structured query language; tradução Elvira Maria Antunes Uchôa. 6ª Edição, Rio de Janeiro: LTC, 2007.

DAVIS, Michele E; PHILIPS, Jon A, Aprendendo PHP e MySQL. Tradução da 6ª Edição, Rio de Janeiro: Alta Books, 2008.

MANZANO, José Augusto N.G. MySQL 5.5 interativo: guia essencial de orientação e desenvolvimento. 1ª edição, São Paulo: Érica, 2011.

Sumário

1. Conceitos gerais.....	4
1.1. O Modelo relacional	5
1.2. Aplicações de banco de dados	6
2. Modelagem de dados	9
2.1. Modelo Entidade-Relacionamento	10
2.2. Cardinalidade do relacionamento.....	12
2.3. Exercícios práticos	14
2.3.1. Clínica veterinária.....	15
2.3.2. Locadora de veículo.....	15
2.3.3. Ambiente empresarial	15
2.3.4. Área comercial.....	15
2.3.5. Empresa de construção civil.....	16
2.3.6. Sistema acadêmico.....	16
2.3.7. Companhia aérea	17
3. Modelo relacional e comandos SQL para definição de dados (DDL) ...	18
3.1. Linguagem SQL.....	19
3.1.1. Linguagem SQL: Comandos DDL.....	21
3.1.2. Linguagem SQL: chaves primárias e chaves estrangeiras	27
3.2. Normalização.....	34
3.2.1. 1ª Forma Normal.....	34
3.2.2. 2ª Forma Normal.....	34
3.2.3. 3ª Forma Normal.....	35
3.3. Exercícios práticos.....	36
3.3.1. Condomínio.....	36
3.3.2. Biblioteca	36
3.3.3. Operadora de Turismo.....	37

3.3.4.	Sistema escolar.....	38
3.3.5.	Sistema bancário	38
3.3.6.	Notas fiscais	39
3.3.7.	Inscrição	39
3.3.8.	Paciente	39
4.	SQL: Comandos para manipulação de dados (DML)	40
4.1.	Comando INSERT	40
4.2.	Comando UPDATE	41
4.3.	Comando DELETE	43
4.4.	Comando SELECT.....	44
4.4.1.	Comando SELECT: ligação entre tabelas e subconsultas.....	50
4.5.	Exercícios práticos.....	55
5.	Segurança na transação e acesso a dados	59
5.1.	Segurança no acesso a dados (Comandos DCL).....	59
5.2.	Segurança na transação de dados	62
6.	Acesso a dados com PHP - MySQL	65
7.	Acesso a dados com Visual C# - SQL Server.....	73

1. Conceitos gerais

Um banco de dados ou base de dados (sua abreviatura é BD, em inglês DB, *database*) são conjuntos de dados com uma estrutura regular que tem como objetivo organizar uma informação. Um banco de dados normalmente agrupa informações utilizadas para um mesmo fim de forma que possam representar coleções de informações que se relacionam de forma que crie um sentido. São de vital importância para empresas, e há duas décadas se tornaram a principal peça dos sistemas de informação.

Um banco de dados é uma coleção de dados relacionados. Entende-se por dado, toda a informação que pode ser armazenada e que apresenta algum significado dentro do contexto ao qual ele se aplica. Por exemplo, num sistema bancário, uma pessoa é identificada pelo seu cpf (cliente). Em um sistema escolar a pessoa é identificada pelo seu número de matrícula (aluno). Em um sistema médico a pessoa (paciente) é identificada pelo número do plano de saúde ou cartão SUS.

A lista telefônica é um exemplo de banco de dados. Nela percebemos que todos os dados referentes a uma pessoa estão na mesma linha, a isso chamamos de registros. O tipo ou categoria da informação (nome, telefone, etc.) sobre uma pessoa está separada em colunas, as quais chamamos campos. Uma lista de compras, lista telefônica, lista de contatos são exemplos de banco de dados presentes em nosso dia-dia.

Um banco de dados informatizado é usualmente mantido e acessado por meio de um software conhecido como Sistema Gerenciador de Banco de Dados (SGBD), que e muitas vezes o termo banco de dados é usado como sinônimo de SGDB. Um SGBD - Sistema de Gerenciamento de Banco de Dados é uma coleção de programas que permitem ao usuário definir, construir e manipular Bases de Dados para as mais diversas finalidades.

Um banco de dados pode ser local, quer dizer utilizável em uma máquina por um usuário, ou repartida, quer dizer que as informações são armazenadas em máquinas distantes e acessíveis por rede. A vantagem essencial da utilização dos bancos de dados é a possibilidade de poder ser acessada por vários usuários, simultaneamente.

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

1.1. O Modelo relacional

O modelo relacional é uma teoria matemática criada por Edgar Frank Codd em 1970 para descrever como as bases de dados devem funcionar. O Modelo relacional revelou-se ser o mais flexível e adequado ao solucionar os vários problemas que se colocam no nível da concepção e implementação da base de dados. A estrutura fundamental do modelo relacional é a relação (**tabela**). Uma relação é constituída por um ou mais atributos (**campos**) que traduzem o tipo de dados a armazenar. Cada instância do esquema (**linha**) é chamada de tupla (**registro**).

O modelo relacional implementa estruturas de dados organizadas em relações ou tabelas. Porém, para trabalhar com essas tabelas, algumas restrições precisaram ser impostas para evitar aspectos indesejáveis, como: Repetição de informação, incapacidade de representar parte da informação e perda de informação. Essas restrições são: integridade referencial, chaves e integridade de junções de relações.

As ilustrações abaixo apresentam exemplos de tabelas sob o modelo relacional.

MÉDICO		
Código do Médico	Nome	Especialidade
1	Mauricio de Nassau	Cardiologia
2	Jorge Amado	Dermatologista
3	Paulo Coelho	Cardiologia
4	Willian Bonner	Neurologista

PACIENTE	
Código do Paciente	Nome
1	Jabes Ribeiro
2	Vane do Renascer
3	Geraldo Simões
4	Capitão Azevedo

MÉDICO			
Código do Médico	Código do Paciente	Data	Hora
1	2	21/01/2013	14:25
1	3	21/01/2013	15:45
2	1	12/04/2012	09:25
4	3	27/05/2012	11:15

Em resumo e de acordo com a arquitetura ANSI / SPARC os Bancos de dados relacionais consistem de três componentes:

1. Uma coleção de estruturas de dados, formalmente chamadas de relações, ou informalmente tabelas, compondo o nível conceitual;
2. Uma coleção dos operadores, a álgebra e o cálculo relacionais, que constituem a base da linguagem SQL; e
3. Uma coleção de restrições da integridade, definindo o conjunto consistente de estados de base de dados e de alterações de estados.

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

Nos bancos de dados relacionais os relacionamentos entre as tabelas não são codificados explicitamente na sua definição. Em vez disso, se fazem implicitamente pela presença de atributos chave. As bases de dados relacionais permitem aos utilizadores (incluindo programadores) escreverem consultas (queries), reorganizando e utilizando os dados de forma flexível e não necessariamente antecipada pelos projetistas originais.

Esta flexibilidade é especialmente importante em bases de dados que podem ser utilizadas durante décadas, tornando as bases de dados relacionais muito populares no meio comercial.

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

1.2. Aplicações de banco de dados

Bancos de dados são usados em muitas aplicações, desde sistemas simples para controlar o estoque de material de uma loja a sistemas avançados como sistemas bancários e segurança pública.

Um aplicativo de banco de dados é um tipo de software exclusivo para gerenciar um banco de dados. Aplicativos de banco de dados abrangem uma vasta variedade de necessidades e objetivos, de pequenas ferramentas como uma agenda, até complexos sistemas empresariais para desempenhar tarefas como a contabilidade. O termo "Aplicativo de Banco de dados" usualmente se refere a softwares que oferecem uma interface para o banco de dados. O software que gerencia os dados é geralmente chamado de sistema gerenciador de banco de dados (SGBD) ou (se for embarcado) de "database engine".

Exemplos de aplicativos de banco de dados (SGBD) são Microsoft SQL Server, Oracle, MySQL, PostgreSQL, Firebird, etc. Os SGBD tem sete características operacionais elementares sempre observadas, que passaremos a listar:



Figura 1 - Sistemas gerenciadores de banco de dados - SGBD

Característica 1: Controle de Redundâncias - A redundância consiste no armazenamento de uma mesma informação em locais diferentes, provocando inconsistências. Em um Banco de Dados as informações só se encontram armazenadas em um único local, não existindo duplicação descontrolada dos dados.

Característica 2: Compartilhamento dos Dados - O SGBD deve incluir software de controle de concorrência ao acesso dos dados, garantindo em qualquer tipo de situação a escrita/leitura de dados sem erros.

Característica 3: Controle de Acesso - O SGBD deve dispor de recursos que possibilitem controlar e definir o acesso de cada usuário. Assim um usuário poderá realizar qualquer tipo de acesso, outros poderão ler alguns dados e atualizar outros e outros ainda poderão somente acessar um conjunto restrito de dados para escrita e leitura.

Característica 4: Interfaceamento - Um Banco de Dados deverá disponibilizar formas de acesso gráfico, em linguagem natural, em SQL ou ainda via menus de acesso.

Característica 5: Esquematização - Um Banco de Dados deverá fornecer mecanismos que possibilitem a compreensão do relacionamento existentes entre as tabelas e de sua eventual manutenção.

Característica 6: Controle de Integridade-Um Banco de Dados deverá impedir que aplicações ou acessos pelas interfaces possam comprometer a integridade dos dados.

Característica 7: Backups - O SGBD deverá apresentar facilidade para recuperar falhas de hardware e software.

Em nosso curso, voltado à construção e manipulação prática dos Bancos de Dados, e não de sua construção teóricas, apenas citaremos os aspectos básicos da construção teórica, de forma a facilitar ao estudante o relacionamento que existe entre Análise de Sistemas e Banco de Dados. Neste curso abordaremos os seguintes tópicos:

1 - Modelagem de Dados: modelo entidade-relacionamento

2 - Implementação e manipulação de banco de dados através da linguagem SQL:

- DML - Data Manipulation Language - trabalha com linhas (ex: comando SELECT);
- DDL - Data Definition Language - trabalha com obectos (ex: comando CREATE TABLE)
- DCL - Data Control Language - trabalha com utilizadores (ex: comando REVOKE);

3 - Integração de banco de dados com linguagem de programação através do desenvolvimento de um exemplo prático de integração de uma aplicação desenvolvida através de uma linguagem de programação (C#, PHP) com um banco de dados desenvolvido em um SGBG(SQL Server, MySQL)

2. Modelagem de dados

A modelagem de dados é a criação de uma estrutura de dados eletrônica (banco de dados) que representa um conjunto de informações. Esta estrutura permite ao usuário recuperar dados de forma rápida e eficiente. O objetivo é incluir dados em uma estrutura que possibilite transformar os dados originais em vários tipos de saídas como formulários, relatórios, etiquetas ou gráficos.

Essa capacidade de transformar informações caracteriza as operações de banco de dados e é a chave de sua utilidade. Um Banco de Dados – BD, representa uma coleção de dados que possui algum significado e objetiva atender a um conjunto de usuários. Por exemplo, um catálogo telefônico pode ser considerado um BD. Sendo assim, um BD não necessariamente está informatizado.

Quando resolvemos informatizar um BD, utilizamos um programa especial para realizar essa tarefa, o SGBD – Sistema Gerenciador de Banco de Dados. Em um SGBD relacional, enxergamos os dados armazenados em uma estrutura chamada tabela. Neste modelo, as tabelas de um BD são relacionadas, permitindo assim que possamos recuperar informações envolvendo várias delas. Observe o exemplo abaixo:

CLIENTES		
Código	Nome	Data de nascimento
1	Regilan Meira Silva	13/02/1983
2	Aline Araujo Freitas	27/08/1986
3	Joaquim José Pereira da Silva	12/05/1967
4	Maria Aparecida Gomes da Costa	06/01/1995

TELEFONES		
Código	Numero	Tipo
1	(73)9158-9683	Celular
1	(71)3458-5112	Residencial
3	(73)8874-9681	Celular
4	(77)8841-2563	Celular

Podemos verificar que a tabela **CLIENTES** está relacionada com a tabela Telefones. Note que o cliente Regilan Meira Silva possui dois telefones: um celular e um residencial. A cliente Aline Araujo Freitas possui um telefone celular, Maria Aparecida Gomes da Costa possui um celular e Joaquim José Pereira da Silva não possui telefone. Tal constatação é verificada após comparar a coluna CÓDIGO da tabela CLIENTES com a coluna CÓDIGO da tabela TELEFONES. A coluna CÓDIGO é utilizada para fazer o relacionamento entre as tabelas.

Entretanto, para que possamos implementar, de forma correta, um BD utilizando algum SGBD, temos que passar por uma fase intermediária – e não menos importante - chamada modelagem de dados.

Quando estamos aprendendo a programar e desenvolver os algoritmos, em geral dividimos esta tarefa em três fases:

1. Entendimento do problema;
2. Construção do algoritmo;
3. Implementação (linguagem de programação).

Em se tratando de banco de dados não é muito diferente:

1. Entendimento do problema;
2. Construção do modelo ER – entidade e relacionamento;
3. Implementação (SGBD).

Entender determinado problema nem sempre é uma tarefa fácil, principalmente se você não está familiarizado com a área de atuação de seu cliente, e/ou o cliente não sabe descrever sobre o problema a qual você foi contratado para resolver.

2.1. Modelo Entidade-Relacionamento

Antes da implementação em um SGBD, precisamos de uma descrição formal da estrutura de um banco de dados, de forma independente do SGBD. Essa descrição formal é chamada modelo conceitual. Podemos comparar o modelo conceitual com o pseudocódigo/português estruturado em algoritmos, na qual construímos os algoritmos independentes de que linguagem de programação iremos desenvolver nossos programas.

Costumamos representar um modelo conceitual através da abordagem **entidade-relacionamento (ER)**. Nesta abordagem construímos um diagrama, chamado **diagrama entidade-relacionamento (DER)**. Observe abaixo o diagrama que originou as tabelas CLIENTES e TELEFONES:

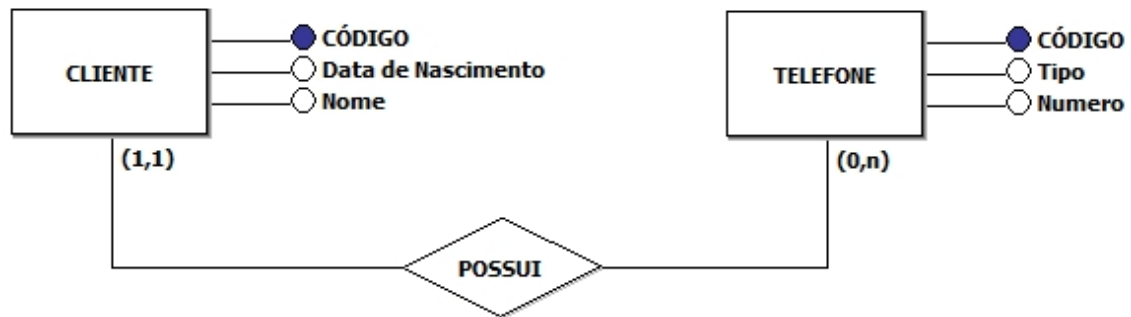


Figura 2 - DER Cliente x Telefone

Entidade pode ser entendida como uma “coisa” ou algo da realidade modelada onde deseja-se manter informações no banco de dados (BD). No exemplo anterior, as tabelas Clientes e Telefones são as entidades no modelo Entidade-Relacionamento. Outro exemplo seria em um sistema escolar, algumas entidades podem ser os alunos, professores, horário, disciplinas e avaliações. Note que uma entidade pode representar tanto objetos concretos (alunos), quanto objetos abstratos (horário). A entidade é representada por um retângulo. Uma entidade se transformará em uma tabela no modelo físico de banco de dados.

Observe outro exemplo abaixo:

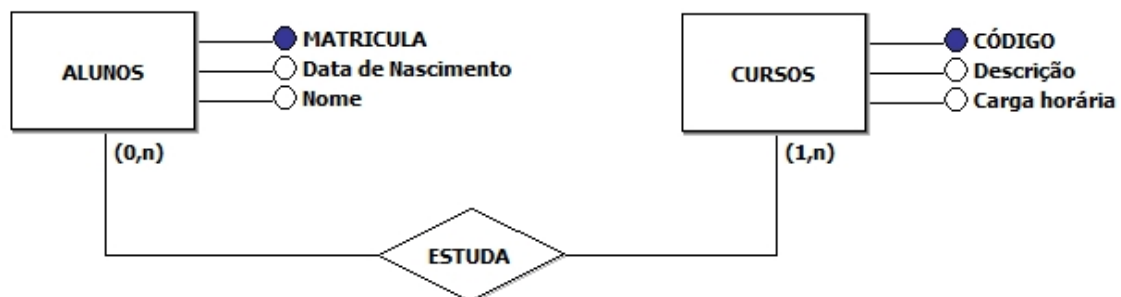


Figura 3 - DER Alunos x Curso

A entidade ALUNO representa todos os estudantes sobre as quais se deseja manter informações no BD. Relacionamento é um conjunto de associações entre entidades. O relacionamento é representado por um **LOSANGO** e o **nome do relacionamento (POSSUI,ESTUDA)**. Esse losango é ligado por linhas aos retângulos que representam as entidades participantes do relacionamento.

Um relacionamento pode envolver ocorrências de uma mesma entidade. Neste caso, estamos diante de um auto-relacionamento. Observe o exemplo:

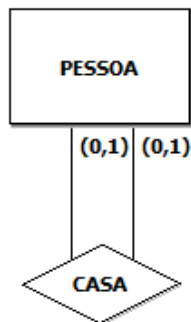


Figura 4 - DER: Auto relacionamento

Neste caso, CASA é um relacionamento que envolve duas ocorrências da entidade PESSOA, ou seja **PESSOA casa com PESSOA**.

2.2. Cardinalidade do relacionamento

Observe o modelo abaixo que representa um relacionamento entre Titular e Dependente e em seguida, considere os seguintes questionamentos:



Figura 5 - DER Titular x Dependente

- Um TITULAR pode não ter DEPENDENTES?
- Um DEPENDENTE pode ter mais de um TITULAR associado ?
- Determinado TITULAR pode possuir mais de um DEPENDENTE?
- Pode existir DEPENDENTE sem algum TITULAR associado?

As respostas desses questionamentos dependem do problema sendo modelado. Para que possamos expressar essas ideias no modelo, é necessário definir uma propriedade importante do relacionamento - **sua cardinalidade**.

A cardinalidade é um número que expressa o comportamento (número de ocorrências) de determinada entidade associada a uma ocorrência da entidade em questão através do relacionamento.

Existem dois tipos de cardinalidade: mínima e máxima. A cardinalidade máxima, expressa o número máximo de ocorrências de determinada entidade, associada a uma ocorrência da entidade em questão, através do relacionamento. A cardinalidade mínima, expressa o número mínimo de ocorrências de determinada entidade associada a uma ocorrência da entidade em questão através do relacionamento. Usaremos a seguinte convenção para expressar a cardinalidade:

Cardinalidade (Mínimo, Máximo)

Observe as cardinalidades mínima e máxima representadas no modelo abaixo:



Figura 6 – DER Titular x Dependente: cardinalidade

Para fazermos a leitura do modelo, partimos de determinada entidade e a cardinalidade correspondente a essa entidade é representada no lado oposto. Em nosso exemplo, a cardinalidade (0:N) faz referência a TITULAR, já a cardinalidade (1:1), faz referência a DEPENDENTE. Isso significa que:

- Uma ocorrência de empregado pode não estar associada a uma ocorrência de dependente ou pode estar associada a várias ocorrências dele (determinado empregado pode não possuir dependentes ou pode possuir vários);
- Uma ocorrência de dependente está associada a apenas uma ocorrência de empregado (determinado dependente possui apenas um empregado responsável).

Observação: Na prática, para as cardinalidades máximas, costumamos distinguir dois tipos: 1 (um) e N (cardinalidades maiores que 1). Já para as cardinalidades mínimas, costumamos distinguir dois tipos: 0 (zero) e 1 (um).

Atributo é uma característica relevante associada a cada ocorrência de entidade ou Relacionamento. Na figura 2, onde é apresentado o DER entre CLIENTE e TELEFONE, verificamos a presença de vários atributos: CLIENTE(Código, Nome, Data de Nascimento) e em TELEFONE (Código, Número, Tipo)

Para deixarmos o modelo de entidade e relacionamentos mais preciso, é necessário que haja uma forma de distinguir uma ocorrência da entidade das demais ocorrências da mesma entidade. Sendo assim, cada entidade deve possuir um identificador. Há várias formas de identificarmos entidades. Observe o modelo abaixo:

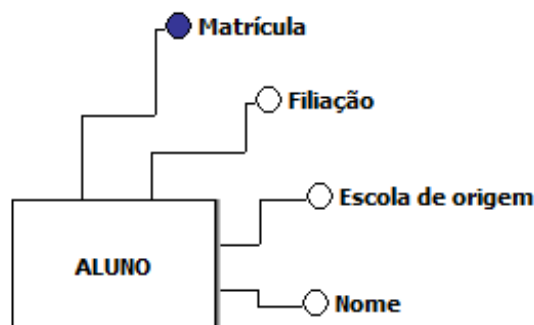


Figura 7 – Atributos da entidade ALUNO

Neste caso, a entidade aluno possui um único identificador (**Matrícula**). Em outras palavras, cada aluno deve possuir uma matrícula diferente. Existem situações onde é necessário mais de um atributo para identificar determinada entidade. Observe que para diferenciar um atributo identificador dos demais atributos, este aparece preenchido em azul.

Imagine uma biblioteca onde os livros ficam armazenados em prateleiras. Estas prateleiras encontram-se organizadas em corredores. Dessa forma, para identificar uma prateleira é necessário conhecer seu número, além do número do corredor correspondente. Observe o modelo abaixo:

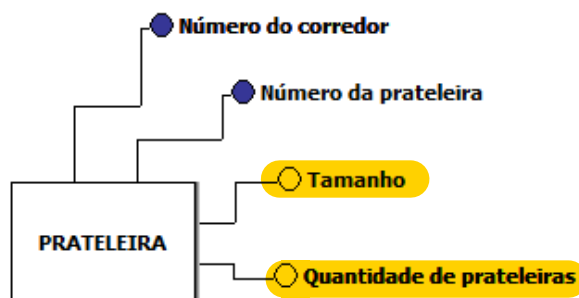


Figura 8 - Atributos da entidade Prateleira

Vimos que o identificador de entidade corresponde a um conjunto de atributos e relacionamentos cujos valores diferenciam cada ocorrência de entidade. No caso de relacionamentos, em geral a identificação ocorre através das ocorrências das entidades que fazem parte dele.

O diagrama Entidade-Relacionamento pode ser elaborado de forma manual ou utilizando softwares editores de Diagrama. Alguns destes aplicativos são:

- brModelo 2.0
- MySQL Workbench 5.2
- Open ModelSphere 3.1
- Devgems Data Modeler 1.4

Recomendamos o uso do brModelo 2.0 em virtude de ser um software brasileiro, gratuito e independente de qualquer SGBD. Com o brModelo 2.0 podemos criar diagramas entidade-relacionamento, modelagem relacional (com tabelas) e até gerar o código para implementação física do banco de dados em algum SGBD.

2.3. Exercícios práticos

Para facilitar o entendimento e ilustrar os elementos que definem um diagrama ER, serão apresentados alguns problemas com o seu propósito e as necessidades que este deve atender. Os cenários apresentados descrevem uma aplicação real de um sistema de banco de dados, sendo assim várias entidades,

atributos e relacionamentos devem ser respeitadas a fim de que o sistema funcione da forma esperada. Para isto, será criado para cada situação, um modelo ER que descreva todas as particularidades expostas, servindo de base para ilustrar os conceitos que envolvem a modelagem de um banco de dados relacional.

2.3.1. Clínica veterinária

O objetivo é desenvolver um modelo de dados para um hospital veterinário. Cada cliente pode possuir um ou vários animais em tratamento. Cada animal pode estar sofrendo de uma ou várias enfermidades. Os casos mais simples são resolvidos, geralmente, por um único veterinário, entretanto podem ocorrer casos em que um animal é atendido por mais de um veterinário.

2.3.2. Locadora de veículo

Elaborar o M.E.R., bem como identificar os atributos de cada entidade e relacionamentos, para uma Locadora de Automóveis, sabendo-se que:

- Para cada veículo locado é necessário saber , a marca, modelo, descrição, cor, placa e outros;
- Para cada contrato é necessário saber, data, preço da diária, o veículo, o cliente e o valor do seguro;
- Para cada cliente é necessário saber, o nome, endereço, cpf e outros dados pessoais;
- Para as manutenções dos veículos é necessário saber, por modelo o custo médio mensal.

2.3.3. Ambiente empresarial

Elaborar o M.E.R., bem como identificar os atributos de cada entidade e relacionamentos, para um ambiente empresarial, composto de departamentos, sabendo-se que:

- Cada departamento possui um código, nome, sigla e um chefe;
- Os chefes de departamento estão divididos em categorias e, para a empresa, é importante saber a data em que foi assumida cada chefia;
- Os empregados da empresa estão ligados a um departamento e a cada um deles está associado matrícula, nome, sexo, telefone, dependentes, data de admissão e cargo;
- Os empregados são alocados em projetos e a informação data de alocação deve ser mantida; e
- Cada projeto é caracterizado por um número, nome e horas previstas.

2.3.4. Área comercial

Uma empresa deseja informatizar sua Área Comercial. Após levantamento junto às áreas envolvidas, as seguintes necessidades foram identificadas:

- Para os produtos comercializados pela empresa é necessário saber o seu código, nome, custo, preço e a família de produtos a que pertence, bem como os preços e quantidades por pedido de venda. Para estes é necessário ter informações como número, data, cliente e as faturas geradas;
- Para os clientes é necessário ter o código, nome, endereço, pedidos de venda, faturas e duplicatas; e
- Para as faturas o número, data, duplicatas geradas e a transportadora dos produtos constantes na mesma. Quanto às duplicatas, necessita-se o número, data de vencimento, cliente e o portador (banco).

Fazer o diagrama entidade relacionamento e identificar os atributos de cada entidade e de cada relacionamento.

2.3.5. Empresa de construção civil

Um empresa de Construção Civil deseja informatizar sua Área de Pessoal, bem como o de Administração de Obras. Para atender estas necessidades o seguinte foi detectado:

- Todos os empregados tem um cargo e existe um plano de carreira para os mesmos;
- Os empregados estão lotados em órgãos, cada um deles tendo um chefe. Dentre os critérios para promoção dos empregados, destacam-se o da titulação e a participação em cursos profissionalizantes, com suas respectivas datas;
- Para todas as obras executadas, previamente são definidas as atividades que serão executadas, com seus respectivos cronogramas por participante, seu custo orçado, seu prazo de execução e o responsável pela mesma. Ainda são definidos os materiais a serem utilizados por atividade, independente do executor, e os custos previstos;
- A empresa definiu a necessidade de manter-se um histórico dos preços praticados pelos seus fornecedores; e
- Para uma melhor administração das obras, foi definido que semanalmente deveria ser emitido um relatório, demonstrando por obra o seu custo orçado e o real até a data, bem como o cronograma de execução, apontando os desvios existentes, se for o caso.

2.3.6. Sistema acadêmico

Sobre a área de Controle Acadêmico de uma Universidade temos as seguintes informações:

- É composta por diversos departamentos, os quais podem oferecer diversos cursos;
- Uma disciplina é oferecida por um único departamento;
- Todo aluno está matriculado em um único curso;
- Uma mesma disciplina pode constar do currículo de diversos cursos; e
- Todo professor está lotado em um departamento e pode ministrar diversas disciplinas.

Fazer o diagrama entidade relacionamento e identificar os atributos de cada entidade e de cada relacionamento.

2.3.7. Companhia aérea

Considere a descrição de um sistema de venda de passagens aéreas dada abaixo e desenhe um diagrama ER de acordo com as seguintes informações:

- Para um passageiro são registrados o número do documento de identidade e o seu nome.
- Um avião é caracterizado por um número de série único e por um modelo.
- Um assento é uma posição única em um avião, identificada por um código. Um assento é da classe econômica ou executiva.
- Um avião possui vários assentos e pode ser usado em vários vôos.
- Um vôo é identificado por um número e utiliza: um avião, um piloto, um aeroporto de partida e outro de chegada.
- Um passageiro pode reservar um assento em um vôo

3. Modelo relacional e comandos SQL para definição de dados (DDL)

O modelo relacional é um modelo de dados, adequado a ser o modelo de um Sistema Gerenciador de Banco de Dados (SGBD), que se baseia no princípio em que todos os dados estão guardados em tabelas (representação bi-dimensional de dados composta de linhas e colunas).

Tornou-se um padrão de fato para aplicações comerciais, devido a sua simplicidade e desempenho.

Toda a Informação de um banco de dados relacional é armazenada em TABELAS, que no modelo entidade-relacionamento são chamadas de ENTIDADES. Por exemplo, uma entidade "Alunos", no modelo relacional será uma tabela de nome ALUNO onde serão armazenadas informações sobre os diversos alunos.

Sobre cada um dos alunos podem ser armazenadas diversas informações tais como: Nome, RG, Matricula, Rua, Bairro, Telefone, CEP, Sexo, Estado Civil, etc.

Essas diversas características de cada Aluno são os "Atributos" da entidade Aluno, no modelo entidade-relacionamento. No modelo relacional os atributos são chamados de campos da tabela Aluno. Abaixo a representação da tabela Aluno no modelo relacional.

ALUNO				
matricula	nome	sexo	estado_civil	telefone
20121234	Luis Felipe Scolari	M	Casado	(81)5623-5475
20121235	Fernando Henrique Cardoso	M	Viúvo	(11)4126-1856
20111233	Dilma Rouseff	F	Divorciada	(33)2354-8962
20121236	Paulo Henrique Ganso	M	Solteiro	(11)2653-4123

A criação de tabelas e atributos em um banco de dados relacional, deve seguir regras, assim como acontece nas linguagens de programação para a criação de variáveis. Entre estas regras destacam-se:

- Nomes de tabelas devem ser únicos no banco de dados;
- Nomes de atributos devem ser únicos em uma tabela
- Não deverão existir espaços em nomes de tabelas e atributos
- Não deverá usar acentos e caracteres especiais para nomear tabelas e atributos.

Considerando a tabela Aluno, podemos observar que ela é composta de 5 colunas ou campos, a qual damos o nome de cada um dos atributos. Um nome de atributo deve ser único em uma tabela e dizer exatamente o tipo de informação que ele representa (caractere, número, data, etc.).

Uma coluna (atributo) não segue um ordenamento específico. O nome de uma coluna deve expressar exatamente o que armazena e sempre que possível utilizar prefixos padronizados: Cod_Dept, Nome_Funcionario, Qtde_Estoque.

NUNCA UTILIZAR ACENTOS GRÁFICOS E CARACTERES DA LÍNGUA PORTUGUESA, COMO EXEMPLO “Ç” PARA NOMEAR ATRIBUTOS E TABELAS

Podemos verificar também que a tabela Aluno possui quatro registros e cada registro representa um conjunto de valores. A este relacionamento damos o nome de registro, linha ou ainda Tupla;

Verificamos também que cada linha da tabela é única e possui um atributo identificador (**Num_Matrícula**). Este atributo identificador é chamado de chave primária. Uma chave primária nunca deverá ser repetido, ou seja, no caso da tabela Alunos nunca acontecerá de um aluno ter matrícula igual a outro aluno.

Regras:

- Em uma tabela não devem existir linhas duplicadas;
- As linhas de uma tabela não seguem uma ordem específica.

A tabela Aluno possui cinco campos (atributos). Para cada campo existe um conjunto de valores permitidos chamado domínio daquele atributo:

- Para o campo matricula o domínio é o conjunto de números naturais;
- Para o campo nome o domínio é qualquer nome válido;
- Enquanto que para sexo o domínio são os mnemônicos M ou F.

3.1. Linguagem SQL

Para a criação de banco de dados, tabelas e atributos em um SGBD, utilizaremos a linguagem SQL que é composta de comandos de manipulação, definição e controle de dados. A SQL estabeleceu-se como linguagem padrão de Banco de Dados Relacional.

SQL apresenta uma série de comandos que permitem a definição dos dados, chamada de DDL (Data Definition Language), composta entre outros pelos comandos Create, que é destinado a criação do Banco de Dados, das Tabelas que o compõe, além das relações existentes entre as tabelas. Como exemplo de comandos da classe DDL temos os comandos Create, Alter e Drop.

Os comandos da série DML (Data Manipulation Language), destinados a consultas, inserções, exclusões e alterações em um ou mais registros de uma ou mais tabelas de maneira simultânea. Como exemplo de comandos da classe DML temos os comandos Select, Insert, Update e Delete.

Uma subclasse de comandos DML, a DCL (Data Control Language), dispõe de comandos de controle como Grant e Revoke.

A Linguagem SQL tem como grandes virtudes sua capacidade de gerenciar índices, sem a necessidade de controle individualizado de índice corrente, algo muito comum nas linguagens de manipulação de dados do tipo registro a registro. Outra característica muito importante disponível em SQL é sua capacidade de construção de visões, que são formas de visualizarmos os dados na forma de listagens independente das tabelas e organização lógica dos dados.

Outra característica interessante na linguagem SQL é a capacidade que dispomos de cancelar uma série de atualizações ou de as gravarmos, depois de iniciarmos uma seqüência de atualizações. Os comandos Commit e Rollback são responsáveis por estas facilidades.

Devemos notar que a linguagem SQL consegue implementar estas soluções, somente pelo fato de estar baseada em Banco de Dados, que garantem por si mesmo a integridade das relações existentes entre as tabelas e seus índices.

Para trabalhar com a linguagem SQL, antes de tudo é necessário ter um SGBD instalado com suporte a linguagem SQL. Neste curso, os exemplos serão desenvolvidos baseado em dois SGBD:

- SQL Server Express 2012
- MySQL Community Server 5.5.29

De acordo com a Microsoft, SQL Server é um servidor de banco de dados abrangente, pronto para as cargas de trabalho corporativas mais exigentes, com altos níveis de desempenho, disponibilidade e segurança. O Microsoft SQL Server, apresenta uma versão gratuita, chamada de SQL Server Express. Este SGBD fornece um repositório de dados confiável e avançado para sites leves e aplicativos de área de trabalho. Informações e download do SQL Server estão disponíveis no link: <https://www.microsoft.com/sqlserver/pt/br/default.aspx>

De acordo com a Oracle, o MySQL Community Edition é uma versão livre para download do banco de dados de código aberto mais popular do mundo, que é suportado por uma comunidade ativa de desenvolvedores de código aberto e entusiastas. Informações e download do SQL Server estão disponíveis no link: <http://www.mysql.com/downloads/mysql/>

Recomendados que o leitor realize o download e instale as ferramentas acima (uma ou as duas), antes de iniciar a próxima parte deste estudo. Para ambiente Windows, a instalação destas ferramentas é bastante simples, baseadas em janelas (Next...Next...Finish ou Próximo...Próximo...Concluir). É recomendável

durante a instalação definir uma senha para o usuário administrador do banco de dados (no MySQL este usuário é conhecido como **root** e no SQL Server como **sa**).

3.1.1. Linguagem SQL: Comandos DDL

A maioria dos SGBDs (inclusive o SQL Server e o MySQL) disponibiliza de ferramentas gráficas (conforme será verificado em sala de aula) que permitem a criação de Bancos de Dados, mas é possível criar o próprio Banco de Dados a partir de um comando SQL. Neste material, iremos apenas estudar a implementação e manipulação de banco de dados a partir de comandos SQL. Durante as aulas e atividades em sala de aula, usaremos as ferramentas visuais.

O primeiro comando a ser estudado é o **CREATE DATABASE**. Este comando permite criar um banco de dados no SGBD escolhido.

```
CREATE DATABASE nome_do_banco_de_dados
```

Um exemplo para o comando seria a criação do banco de dados IFBA Ilhéus, que será usado para o gerenciamento acadêmico do Campus. Veja o exemplo:

```
CREATE DATABASE ifba_ilheus
```

OBS: Recomendamos o uso de caracteres minúsculos para nomes de banco de dados, tabelas e campos, em virtude de alguns sistemas operacionais e SGBD serem case-sensitive. Adotar um padrão, neste caso, minúsculo, facilitará a organização e manipulação dos trabalhos.

O comando **DROP DATABASE** permite remover um determinado Banco de Dados, apagando todas as tabelas e estruturas associadas e, conseqüentemente, todos os dados existentes nelas. **Recomendamos cuidado ao utilizar este comando.**

```
DROP DATABASE nome_do_banco_de_dados
```

O comando abaixo remove toda a estrutura e dados do banco de dados **ifba_ilheus**.

```
DROP DATABASE ifba_ilheus
```

O comando **CREATE TABLE** é o principal comando DDL da linguagem **SQL**. A criação de tabelas é realizada em SQL utilizando este comando. Sua sintaxe básica é a seguinte:

```
CREATE TABLE nome_da_tabela(  
    campo1 Tipo,  
    campo2 Tipo,  
    campo3 Tipo)
```

Um exemplo para o comando **CREATE TABLE** seria a criação da tabela **Empregado**, com os campos: **matricula** nome, **data** de nascimento e **salario**.

```
CREATE TABLE empregado(  
    matricula INTEGER,  
    nome VARCHAR(50) ,  
    data_nasc DATE,  
    salario FLOAT)
```



matricula	nome	data_nasc	salario

Em SQL, cada campo (coluna) de uma tabela deverá possuir um nome e um tipo de dados, quem em geral são agrupados em 3 categorias:

- Caracteres (Strings)
- Numéricos
- Tempo e Data

Os tipos de dados podem ser agrupados na tabela abaixo:

TIPO DE DADOS	DESCRIÇÃO
CHAR (n)	Tipo de dados STRING com comprimento fixo (n > 0)
CHAR	Um único caractere
VARCHAR (n)	Tipo de dados STRING com comprimento variável (n > 0)
BIT	Valores True/False ou 0/1
NUMERIC ou NUMERIC (n) ou	Valor numérico constituído por n dígitos e sinal e com d casas decimais

NUMEROC (n,d)	
DATETIME	Um valor de data ou hora entre os anos 100 e 9999.
INTEGER ou INT	Inteiro
SMALLINT	Pequeno Inteiro
FLOAT	Nº de Dupla Precisão
DATE	Datar
TIME	Hora

Estes tipos de dados citados são padrões em banco de dados que utilizam SQL, porém a maioria dos fabricantes de SGBD disponibiliza outros tipos de dados, como exemplo o *MONEY* (para valores monetários), *COUNTER* (para números inteiros incrementados automaticamente pelo banco de dados).

Vejamos os exemplos a seguir:

- Escrever um comando de SQL que permita criar uma tabela com o nome Caixa_Postal, capaz de armazenar um inteiro de até quatro dígitos e uma string com 45 caracteres

```
CREATE TABLE Caixa_Postal(
Codigo NUMERIC,
Localidade VARCHAR(45))
```

- Escrever um comando de SQL que permita criar uma tabela com o nome Pessoa, com os seguintes atributos: ID, Nome, Idade, Salario, Telefone e Código Postal)

```
CREATE TABLE Pessoa(
codigo INTEGER,
nome VARCHAR (45) ,
idade INTEGER,
salario NUMERIC(10,2) ,
telefone VARCHAR (12) ,
codigo_postal VARCHAR (9))
```

Para execução do comando CREATE TABLE é necessário indicar qual o nome da tabela e, para cada uma das colunas, o nome da coluna e o tipo de dados. No entanto, podem ser indicadas as características próprias de cada uma das

colunas, tais como: Que valores podem admitir? Qual o valor padrão? O campo representa um atributo identificador (chave primária)?

Na tabela caixa postal, apresentada anteriormente, estamos admitindo que a tabela é composta por duas colunas(código e localidade) e que qualquer uma delas pode admitir valores nulos(ou seja, o usuário poderá informar dados vazios para os campos).

Se quisermos que uma coluna não admita valores nulos, usamos a cláusula **NOT NULL**. Caso um valor não seja inserido em uma coluna o valor padrão (default) armazenado nela é **NULL**. No entanto, é possível associar um outro valor default através da cláusula **DEFAULT**. Se quisermos, por exemplo, que a localidade padrão (default) se chame Ilhéus então teremos que fazer o seguinte:

```
CREATE TABLE caixa_postal(  
  codigo NUMERIC NOT NULL,  
  localidade VARCHAR(45) DEFAULT 'ILHÉUS')
```

O código acima está enfatizando que o campo **codigo** da tabela Caixa Postal, **NÃO ACEITA** valores nulos, ou seja, o campo **codigo** deverá possuir SEMPRE, qualquer valor diferente de vazio. A localidade padrão é Ilhéus, desta forma, caso não seja informado nenhum valor para localidade, este será preenchido com o valor Ilhéus.

Na criação de tabelas, também podem ser definidas o uso de **RESTRIÇÕES**. Restrições são regras a que os valores de uma ou mais colunas devem obedecer. Por exemplo, o conteúdo da coluna Sexo só poderá conter os valores “F” ou “M”, a coluna Idade não poderá conter valores negativos, o salário não poderá ser inferior ao salário mínimo (R\$ 622,00). A utilização de restrições é a única garantia que temos de que os dados existentes nas colunas estão de acordo com as regras especificadas no projeto do Banco de Dados.

Existem alguns tipos distintos de restrições que se podem aplicar a colunas:

- Constraint NOT NULL
- Constraint CHECK
- Constraint UNIQUE
- Constraint PRIMARY KEY
- Constraint REFERENCES

A constraint **CHECK** permite realizar a validação dos dados inseridos na coluna, através da especificação de uma condição. São admitidos apenas os dados cujo resultado da avaliação da condição seja verdadeiro.

Exemplos com a constraints CHECK:

```
CREATE TABLE Dados_Pessoais
(
  codigo NUMERIC NOT NULL,
  nome CHAR(60) CHECK(nome NOT LIKE '%Regilan%'),
  idade INTEGER NOT NULL CHECK(Idade >= 0 AND Idade <= 150),
  sexo CHAR CHECK (SEXO IN('M', 'F')),
  tempo_servico INTEGER CHECK(Tempo_Servico >= 0)
)
```

A constraint **UNIQUE** indica que os valores dessa coluna não podem se repetir. Em uma tabela podem existir tantas colunas UNIQUE quantas forem necessárias.

Veja o exemplo:

```
CREATE TABLE Dados_Pessoais
(
  codigo NUMERIC NOT NULL,
  nome CHAR(60) UNIQUE,
  cpf CHAR(15) UNIQUE,
  tempo_servico INTEGER CHECK(Tempo_Servico >= 0)
)
```

Considere o seguinte exemplo: Criar um banco de dados Clínica com as seguintes características:

Tabela Medicos

- Atributo CRM: caractere, único e não vazio
- Atributo Nome: caractere e não vazio
- Atributo Idade: inteiro e não poderá ser maior que 23 e menor que 70
- Atributo Especialidade: caractere e não poderá possuir especialização em Ortopedia

Tabela Paciente

- Atributo CPF: caractere e único,
- Atributo Nome: caractere e não vazio
- Atributo Doença: caractere e não poderá ter valores como fratura e torção

Para elaborar o banco de dados proposto, primeiramente é necessário criar um novo banco de dados, para isto usaremos o comando CREATE DATABASE:

```
CREATE DATABASE CLINICA
```

Em seguida, criaremos a tabela Médicos com as restrições indicadas:

```
CREATE TABLE medicos
(
 .crm CHAR(15) NOT NULL UNIQUE,
  nome CHAR(100) NOT NULL,
  idade INTEGER CHECK(idade > 23 AND idade < 70),
  especialidade CHAR(50) CHECK(especialidade NOT LIKE
'%ORTOPEDIA%')
)
```

Por fim criaremos a tabela Paciente com as restrições indicadas

```
CREATE TABLE paciente(
  cpf CHAR(15) UNIQUE,
  nome CHAR(100) NOT NULL,
  doenca CHAR(50) CHECK(doenca NOT LIKE '%FRATURA%' AND
'%TORÇÃO%')
)
```

Antes de verificar o funcionamento das constraints PRIMARY KEY e REFERENCES, veremos o comando DROP e ALTER TABLE. O comando DROP TABLE permite remover uma determinada tabela de um Banco de Dados, e conseqüentemente, todos os dados existentes nela.

A sintaxe do comando é

```
DROP TABLE nome_da_tabela
```

Exemplo:

```
DROP TABLE Dados_Pessoais
```

O comando ALTER TABLE permite alterar a estrutura de uma tabela. Com o comando é possível adicionar uma nova coluna, modificar uma coluna já existente ou eliminar uma coluna.

Veja a sintaxe dos comandos:

```
ALTER TABLE Nome_Da_Tabela ADD Nome_Coluna Tipo_Coluna
```

```
ALTER TABLE Nome_Da_Tabela MODIFY Nome_Coluna Tipo_Coluna
```

```
ALTER TABLE Nome_Da_Tabela DROP Nome_Coluna Tipo_Coluna
```

Exemplos:

```
ALTER TABLE Fornecedor ADD Fax CHAR(15)
```

```
ALTER TABLE Fornecedor MODIFY Logradouro CHAR(60)
```

```
ALTER TABLE Funcionario MODIFY Salario NOT NULL
```

```
ALTER TABLE Funcionario MODIFY Matricula UNIQUE
```

```
ALTER TABLE Fornecedor DROP COLUMN Caixa_Postal
```

OBSERVAÇÃO: Alguns SGBDs podem não disponibilizar a opção MODIFY. Em caso de dúvida consulte o Manual de Referência do SGBD escolhido.

3.1.2. Linguagem SQL: chaves primárias e chaves estrangeiras

No modelo relacional a única forma de relacionar dados que existem em uma tabela com dados que existem em outra tabela é através de atributos comuns.

Assim, vão existir atributos especiais (chaves estrangeiras) que servem para fazer a ligação com outras tabelas, onde esses mesmos atributos são usados para identificar, unicamente, cada uma das linhas (chaves primárias). Os tipos de chaves podem ser:

- **Chave Primária:** chave que identifica cada tupla(linha)
- **Chave Estrangeira:** atributo ou conjunto de atributos de uma relação, que é chave primária em outra relação.

Uma chave primária (atributo identificador) é uma coluna ou um grupo de colunas que assegura a unicidade das linhas dentro de uma tabela. Uma chave primária que tenha mais de uma coluna é chamada de chave primária composta. Chaves primárias são geralmente indicadas pela sigla PK (primary key)

Exemplos de Chaves Primárias:

- Produto (**codigo_produto (PK)**, Descrição, ...)
- Veiculo (**chassi (PK)**, Cor, Tipo, ...)
- Usuario (**login (PK)**, Senha, ...)
- Conta_Bancaria(**agencia (PK)**,**conta_corrente (PK)**, Titular)

Para criação de chaves primárias, algumas regras são definidas, conforme a seguir:

- Valores de chave primária não podem ser nulos (not null):
- Valores de chaves primarias não podem ser nulos porque uma linha sem chave primaria não se distingue de outras linhas da mesma tabela;
- Colunas com chaves primarias não podem ter valores duplicados
- Um valor duplicado é um valor exatamente igual a outro de uma ou mais linhas da mesma coluna na mesma tabela;
- Se a coluna possui valores duplicados esta não pode servir de identificador da linha;

Recomendações:

1. Selecione chaves primarias absolutamente disciplinada e que permaneçam únicas;
2. Selecione chaves primarias que não tenham qualquer tendência a alterações;
3. Se possível seleciona chaves primarias simples;
4. De preferência a colunas numéricas para chaves primarias;
5. Selecione chaves primarias que sejam familiares;
6. Se não houver nenhuma coluna com chave primaria candidata, utilize chaves primarias atribuídas pelo sistema (autonumeração).

Uma chave estrangeira é uma coluna ou grupo de colunas que pode ou não ser chave primária da tabela em questão, mas, com certeza é chave primária de outra tabela. Uma chave estrangeira formada por mais de uma coluna é chamada de chave estrangeira composta. Chaves estrangeiras são indicadas pela sigla FK (**Foreign Key**)

Exemplos de Chave Estrangeira:

Dependentes (codigo_dependente, **codigo_socio(FK)**)

Pedido (nota_fiscal, **codigo_cliente(FK)**)

Como estudamos, toda entidade (tabela) deverá possuir um atributo único e não nulo que identifique um registro. Esse atributo é chamado de chave primária ou primary key. A constraint primary key é equivalente às cláusulas NOT NULL + UNIQUE juntas. Em uma tabela deve-se existir apenas uma constraint **primary key**, enquanto podem existir variadas ocorrências da cláusula **UNIQUE**.

Veja os exemplos abaixo para a criação de Primary Key:

```
CREATE TABLE dados_pessoais
(
  nome CHAR(60) UNIQUE,
  cpf CHAR(15),
  tempo_servico INTEGER CHECK(tempo_servico >= 0)
  Primary Key(CPF)
)
```

```
CREATE TABLE Comissao
(
  matricula_funcionario INTEGER,
  codigo_compra INTEGER,
  valor_compra NUMERIC(10,2) CHECK(valor_compra >= 1000)
  Primary Key(matricula_funcionario,codigo_compra)
)
```

A constraint **REFERENCES** permite fazer a validação das chaves estrangeiras. Isto é, não se podem inserir nos campos referenciados como chaves estrangeira valores que não existam na tabela onde os campos são chave primária. Veja exemplo na qual o campo codigo_postal é uma chave estrangeira da tabela POSTAL, para isso foi usado o comando REFERENCES, onde o campo **codigo_postal** da tabela PESSOA refere-se ao campo **codigo** da tabela POSTAL:

```
CREATE TABLE PESSOA
(
  nome CHAR(60) UNIQUE,
  cpf CHAR(15),
  idade INTERGER,
  telefone CHAR(15),
  codigo_postal NUMERIC(10) REFERENCES Postal(Codigo),
  Primary Key(CPF)
)
```

Quando estudamos sobre o modelo entidade-relacionamento, vimos que em um banco de dados, precisamos de alguma maneira para representar estes relacionamentos da vida Real, em termos das tabelas e de seus atributos. Isto é possível com a utilização de "Relacionamentos entre tabelas", os quais podem ser de três tipos:

- Um para Um
- Um para Vários
- Vários para Vários

O relacionamento Um para Um (1:1) existe quando os campos que se relacionam são ambos do tipo Chave Primária, em suas respectivas tabelas. Cada um dos campos não apresenta valores repetidos. Na prática existem poucas situações onde utilizaremos um relacionamento deste tipo.

Como exemplo do relacionamento Um para Um, vamos imaginar uma escola com um Cadastro de Alunos na tabela Alunos, destes apenas uma pequena parte participa da Banda da Escola. Por questões de projeto do Banco de Dados, podemos criar uma Segunda Tabela "Alunos da Banda", a qual se relaciona com a tabela Alunos através de um relacionamento do tipo Um para Um. Desta forma cada aluno somente é cadastrado uma vez na Tabela Alunos e uma única vez na tabela Alunos da Banda. Poderíamos utilizar o Campo Matrícula do Aluno como o Campo que relaciona as duas Tabelas. Veja exemplo no modelo relacional para este caso:

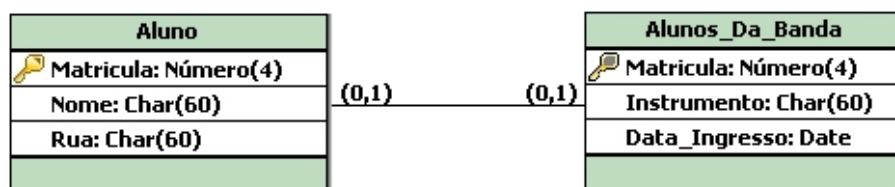


Figura 9 - Exemplo de relacionamento 1:1

A codificação em SQL para o relacionamento entre Aluno e Alunos_Da_Banca seria implementado da seguinte forma:

```
CREATE TABLE alunos
(
    matricula NUMERIC(4),
    nome CHAR(60),
    rua CHAR(60),
    Primary Key(matricula)
)
```

```
CREATE TABLE alunos_da_banda
(
matricula NUMERIC(4) REFERENCES alunos(matricula) ,
instrumento CHAR(60) ,
data_ingresso DATE,
Primary Key(matricula)
)
```

O relacionamento Um para Vários (1:N) é o tipo de relacionamento mais comum entre duas tabelas. Ocorre quando uma das tabelas (o lado um do relacionamento) possui um campo que é a Chave Primária e a outra tabela (o lado vários) se relaciona através de um campo cujos valores relacionados podem se repetir várias vezes. Veja exemplo no modelo entidade-relacionamento para esta situação



Figura 10 - Exemplo de relacionamento 1:N

No exemplo apresentado, há um relacionamento entre a tabela Clientes e Pedidos. Cada Cliente somente é cadastrado uma única vez na tabela de Clientes (por isso o campo Código do Cliente, na tabela Clientes, é uma chave primária, indicando que não podem ser cadastrados dois clientes com o mesmo código), portanto a tabela Clientes será o **lado um do relacionamento**. Ao mesmo tempo **cada cliente pode fazer diversos pedidos, por isso que o mesmo Código de Cliente poderá aparecer várias vezes na tabela Pedidos**, tantas vezes quantos forem os pedidos que o Cliente tiver feito. Desta forma temos um relacionamento do tipo **Um para Vários** entre a tabela Clientes e Pedidos, através do campo Código do Cliente, indicando que um mesmo Cliente pode realizar diversos (vários) pedidos. Abaixo a representação deste relacionamento através do modelo relacional.



Figura 11 - Exemplo de relacionamento 1:N no modelo relacional

A codificação em SQL para o relacionamento entre Cliente e Pedido seria implementado da seguinte forma:

```
CREATE TABLE cliente
(
  cpf NUMERIC(4) ,
  nome CHAR(60) ,
  telefone CHAR(15) ,
  email CHAR(60) ,
  Primary Key(cpf)
)
```

```
CREATE TABLE pedido
(
  numero_pedido NUMERIC(4) ,
  cpf NUMERIC(4) REFERENCES cliente(cpf) ,
  data DATE,
  valor NUMERIC(4,
  Primary Key(numero_pedido)
)
```

O relacionamento Vários para Vários acontece quando nos dois lados do relacionamento, os valores poderiam se repetir. Vamos considerar o caso entre Produtos e Pedidos. Posso ter Vários Pedidos nos quais aparece um determinado produto, além disso, vários Produtos podem aparecer no mesmo Pedido. Esta é uma **situação em que temos um Relacionamento do Tipo Vários para Vários.**

Na prática não é possível implementar um relacionamento deste tipo, devido a uma série de problemas que seriam introduzidos no modelo do banco de dados. Por exemplo, na tabela Pedidos teríamos que repetir o Número do Pedido, Nome do Cliente, Nome do Funcionário, Data do Pedido, etc para cada item do Pedido.

Para evitar este tipo de problema um relacionamento do tipo **Vários para Vários** é quebrado em **dois relacionamentos do tipo Um para Vários**. Isso é feito através da criação de uma nova tabela, a qual fica com o lado Vários dos relacionamentos.

No nosso exemplo vamos criar a tabela Detalhes do Pedido, onde ficam armazenadas as informações sobre os diversos itens de cada pedido, aí **ao invés de termos um relacionamento do tipo Vários para Vários, teremos dois relacionamentos do tipo um para vários**, conforme mostrado no modelo relacional a seguir.

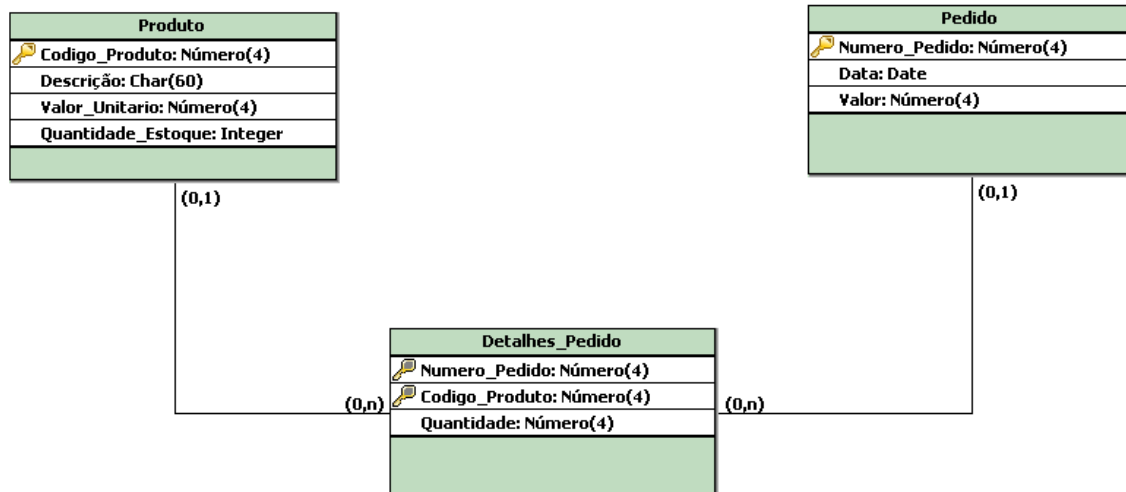


Figura 12 - Exemplo de relacionamento N:N no modelo relacional

A codificação em SQL para este relacionamento seria implementado da seguinte forma:

```

CREATE TABLE produto
(
  codigo_produto NUMERIC(4) ,
  descricao CHAR(60) ,
  valor_unitario NUMERIC(4) ,
  quantidade_estoque NUMERIC(4) ,
  Primary Key(codigo_produto)
)
  
```

```

CREATE TABLE pedido
(
  numero_pedido NUMERIC(4) ,
  data DATE ,
  valor NUMERIC(4) ,
  Primary Key(numero_pedido)
)
  
```

```

CREATE TABLE detalhes_pedido
(
  codigo_produto NUMERIC(4) REFERENCES
produto(Codigo_Produto) ,
  numero_pedido NUMERIC(4) REFERENCES pedido(codigo_pedido) ,
  quantidade NUMERIC(4) ,
  Primary Key(codigo_produto,numero_pedido)
)
  
```

3.2. Normalização

Normalização é o processo que permite a simplificação da estrutura de um banco de dados de modo que esta se apresente em um ótimo estado.

Esse processo é bastante simples e tem por principal objetivo remover grupos repetidos de informações. Normalmente, ao normalizar aumenta-se o número de tabelas e o número total de campos.

A teoria da normalização é baseada no conceito de Formas Normais, que são regras aplicada as estruturas das tabelas com o objetivo de minimizar ou eliminar a redundância de informações. Embora existam seis formas normais, normalmente se considera que um esquema de banco de dados está em um bom nível quando ele se encontra na terceira forma normal.

3.2.1. 1ª Forma Normal

Diz-se que uma relação está na primeira forma normal quando:

- Não contém atributos multivalorados (grupos de atributos)
- Não contém grupos repetitivos

Exemplo:

Fatura (**NumeroFatura (PK)**, CodigoCliente, NomeCliente, Endereco, CodigoProduto, DescricaoProduto, Preco, Quantidade)

Para fazer a passagem para a 1ª FN será necessário: Separar a tabela Fatura em duas tabelas: **Fatura e Itens**.

- Fatura (**NumeroFatura**, CodigoCliente, NomeCliente, Logradouro, Bairro, Cidade, Cep)
- Itens (**NumeroFatura (PK)**, **CodigoProduto(PK)**, DescricaoProduto, Preco, Quantidade)

3.2.2. 2ª Forma Normal

Diz-se que uma relação está na segunda forma normal quando:

- Está na 1FN
- Todos os atributos não chave dependem da totalidade da chave.

Exemplo:

Fatura (**NumeroFatura**,CodigoCliente, NomeCliente, Logradouro, Bairro, Cidade, Cep)

Itens (**NumeroFatura (PK)**, **CodigoProduto(PK)**, DescricaoProduto, Preco, Quantidade)

A tabela Itens não se encontra na 2FN porque os atributos **DescricaoProduto**, **Preco**, **Quantidade** não dependem de **CodigoFatura**, mas só de **CodigoProduto**.

Para fazer a passagem para a 2ª FN será necessário: Separar a tabela Itens em duas Tabelas: Itens e Produto.

Relação na 2ª FN:

- **Fatura (NumeroFatura (PK)**,CodigoCliente, NomeCliente, Logradouro, Bairro, Cidade, Cep)
- **Itens (NumeroFatura (PK)**, **CodigoProduto (PK)**, Quantidade)
- **Produto (CodigoProduto (PK)**, DescricaoProduto,Preco)

3.2.3. 3ª Forma Normal

Uma relação está na 3FN se:

- Está na 2ª FN
- **Todos os seus atributos não chave NÃO são identificados por um outro também não chave.**

Exemplo:

- **Fatura (NumeroFatura (PK)**,CodigoCliente, NomeCliente, Logradouro, Bairro, Cidade, Cep)
- **Itens (NumeroFatura (PK)**, **CodigoProduto (PK)**, Quantidade)
- **Produto (CodigoProduto (PK)**, DescricaoProduto,Preco)

A tabela Fatura não se encontra na 3FN porque os atributos NomeCliente, Logradouro, Bairro, Cidade, Cep são identificados por CodigoCliente e não por CodigoFatura.

Para passar para a 3 FN deve-se separar a tabela Fatura em duas tabelas: Fatura e Cliente.

Relação na 3ª FN:

- **Fatura (NumeroFatura (PK), CodigoCliente)**
- **Cliente (CodigoCliente (PK), NomeCliente, Logradouro, Bairro, Cidade, Cep)**
- **Itens (NumeroFatura (PK), CodigoProduto (PK), Quantidade)**
- **Produto (CodigoProduto (PK), DescricaoProduto (PK), Preco)**

3.3. Exercícios práticos

Para cada situação apresentada a seguir, escreva o script de criação do banco de dados utilizando os comandos SQL-DDL.

3.3.1. Condomínio

Construa um banco de dados para uma Administradora de Condomínios conforme as características abaixo:

- Um condomínio é formado por diversas unidades habitacionais.
- Cada unidade habitacional pertence a um condômino, o qual pode ser proprietário de várias unidades.
- Cada unidade pode ser alugada e tem que pertencer a um único condomínio.
- Toda pessoa (Condômino ou Locatário) possui um código, um nome e um endereço.
- Toda unidade possui um código (que a identifica no condomínio), a metragem, o número de cômodos e um endereço.
- Um condomínio é identificado por um código, um nome e um endereço.
- Entre os condôminos de um condomínio, um é o síndico. O período de vigência de um síndico em um condomínio é importante no processo.

3.3.2. Biblioteca

Construa um banco de dados para um sistema de controle e gerenciamento de empréstimos de livros de uma biblioteca acadêmica conforme as características abaixo:

- A biblioteca dispõe de livros, também denominados títulos. Estes possuem nome, autores e editoras. Cada título pertence a uma área de conhecimento e possui um código único de identificação.
- Cada título possui vários exemplares. Cada exemplar possui um código único de identificação.

- Cada título pode ter vários autores e um mesmo autor pode ter escrito vários títulos. Um autor possui código, nome, telefone e endereço.
- As editoras possuem código, nome, telefone e endereço.
- As áreas de conhecimento possuem código e uma descrição.
- Usuários, que podem ser alunos, professores ou funcionários, tomam livros emprestados por uma semana. A data de empréstimo é importante no processo. Cada usuário possui um código, nome, telefone e endereço.
- Cada título possui várias palavras-chave e uma palavra-chave pode estar ligada a vários títulos. Uma palavra-chave possui código e descrição.

3.3.3. Operadora de Turismo

Desenvolva um banco de dados completo, com o máximo de abstrações capaz de exprimir a situação descrita abaixo:

A Adrenalina Turismo & Aventura Ltda. é uma operadora de Turismo ecológico cujo ponto principal é a construção de nossos Pacotes Turísticos. Para que isto seja possível nosso Setor de Marketing constantemente mantém atualizada uma lista de hotéis. Para cada hotel, mantemos informações sobre, localização, número de quartos, classificação Embratur, existência de serviços de alimentação e valor da hospedagem.

O Setor de Marketing procura manter contato permanente com as diversas operadoras de transporte existentes no mercado, seja por via aérea ou terrestre. De todas, guarda informações sobre nome da operadora, trajeto, número de assentos disponíveis. Para as operadoras aéreas necessita ter informações sobre restrições de volume e taxas adicionais de embarque. Para as operadoras terrestres o número mínimo de assentos a serem utilizados para que o serviço esteja disponível.

O acompanhamento dos prestadores de serviços que proporcionam as diversas atividades a serem ofertadas nos Pacotes é de vital importância. Por conta disso, o Setor de Marketing tem sempre atualizado o nome do prestador do serviço, contato telefone, e-mail, CNPJ, nome e descrição do(s) serviço(s) e valor adotado.

O Setor Administrativo periodicamente monta e atualiza o arquivo de Pacotes a serem ofertados ao público através das informações sobre o hotel a ser utilizado, meios de transporte, serviços disponíveis, data e condições de pagamento e número de vagas. Cada Pacote utiliza uma única opção de hospedagem e pode oferecer diversas opções de atividades. Para cada atividade de um pacote deve-se informar a data e a hora inicial e final, o serviço que será prestado e, se for necessário, um transporte. Os hotéis, serviços e operadoras de transporte podem ser utilizados em diversos Pacotes.

O cliente pode realizar sua reserva determinando o pacote desejado, indicando o número de pessoas que irá constar em sua reserva e a forma de pagamento.

3.3.4. Sistema escolar

Forneça um banco de dados completo, com o máximo de abstrações capaz de exprimir a situação descrita abaixo:

Suponha que você tenha sido contratado para desenvolver um sistema para controlar os cursos livres oferecidos por uma escola. Em conversa com a diretora você apurou as seguintes informações:

A diretora da escola é responsável por registrar os professores que a escola contrata, os cursos que a escola oferece com as suas respectivas durações e por montar as turmas que serão oferecidas, estabelecendo: hora início e hora final da aula, o dia da semana em que a aula ocorrerá (as aulas são semanais), a data inicial para as aulas, o nível da turma (básico, médio, avançado), o número máximo de vagas.

Após a montagem da turma, o coordenador fará a alocação do professor na turma. Uma turma terá um único professor.

Sempre que solicitado por um aluno, a secretária deverá lhe fornecer a relação das turmas ativas, cada uma com seu respectivos horário, dia da aula, professor e disponibilidade de vagas.

Para que a secretária matricule um aluno em uma turma é preciso que ele tenha sido previamente cadastrado e que haja vaga na turma. Ao fazer a matrícula em uma determinada turma, o aluno receberá um comprovante contendo o seu nome e mais o nome, o dia, o horário e a duração do curso.

Sempre que solicitado pela direção ou pela coordenação, o sistema deverá lhe fornecer: (a) A relação de alunos matriculadas em um determinado curso, contendo: nome e duração do curso e mais nome, endereço e telefone do aluno; (b) As turmas que estão programadas para iniciar em um determinado período, contendo: nome do curso, nível da turma, duração do curso, data de início da turma e nome do professor; (c) Os professores que não estão alocados em nenhuma turma em andamento.

3.3.5. Sistema bancário

Desenvolva um banco de dados com o máximo de abstrações capaz de exprimir a situação descrita abaixo:

A construção de uma base de dados organizada com o objetivo de prover informações a usuários conhecedores de música clássica deverá refletir as seguintes características:

- Cada orquestra é catalogada contendo o seu nome, cidade, país e data correspondentes à sua criação;
- Orquestras executam sinfonias, as mais variadas. Os profundos conhecedores de música são capazes até de selecionar a orquestra que melhor desempenha uma determinada sinfonia. De cada uma sinfonia, é possível saber o seu nome, o compositor e a data de sua criação;
- Orquestras são constituídas de músicos, os mais variados, de acordo com a sua função dentro da mesma: maestro, flautista, etc. Cada músico é catalogado contendo: nome do músico, identidade, nacionalidade e data de nascimento. Um músico só pode pertencer a uma orquestra;
- Músicos tocam sinfonias, porém em alguns casos, alguns músicos podem mudar de função segundo a sinfonia (por exemplo, um violinista pode virar maestro). A data em que um músico apresenta uma determinada sinfonia também é importante no contexto e deve ser registrada.

3.3.6. Notas fiscais

Aplicar as Formas Normais cabíveis para a situação abaixo:

NotasFiscais (**NumNF (PK)**, Serie, Data Emissao, CodigoCliente, NomeCliente, Endereco cliente, CPFCliente, CodigoMercadoria, DescricaoMercadoria, QuantidadeVendida, PrecVenda, TotalVenda e TotalGeralNota).

OBS: Cada nota pode ter mais do que uma mercadoria.

3.3.7. Inscrição

Aplicar as Formas Normais cabíveis para a situação abaixo:

Inscricao (**Codigo Aluno (PK)**, NomeAluno, Telefone, AnoAdmissao, CodigoDisciplina, NomeDisciplina, NomeCurso, DataMatricula).

3.3.8. Paciente

Aplicar as Formas Normais cabíveis para a situação abaixo:

Paciente (**num_paciente (PK)**, nome_paciente, num_quarto, descricao_quarto, num_comodos_quarto, cod_medico, nome_medico, fone_medico).

4. SQL: Comandos para manipulação de dados (DML)

A linguagem SQL é basicamente dividida em três tipos de comandos:

SQL = DDL + DML + DCL

- **DDL** (definição de dados): Comandos: CREATE, DROP, ALTER
- **DML** (manipulação de dados): Comandos: SELECT, INSERT, UPDATE e DELETE
- **DCL** (controle de dados): Comandos: GRANT e REVOKE

Os comandos de manipulação de dados (DML) em SQL são representados por:

- **INSERT:** permite a inclusão de novas linhas nas tabelas
- **UPDATE:** altera os valores de dados já cadastrados
- **DELETE:** remove dados já cadastrados
- **SELECT:** usado para consultar o BD e retornar dados que satisfazem a determinada expressão em um comando

4.1. Comando INSERT

O comando INSERT permite inserir uma linha de dados na tabela e possui a seguinte sintaxe abaixo:

```
INSERT INTO NOME DA TABELA (coluna1, coluna2, coluna3) VALUES  
(valor1, valor2, valor3)
```

Exemplos:

```
INSERT INTO Cliente (codigo, nome, sexo)  
VALUES ("200810", "Regilan Meira", "Masculino")
```

```
INSERT INTO Disciplina (codigo, nome, ementa)  
VALUES ("01", "Banco de Dados", "DER, Modelo Relacional, SQL")
```

4.2. Comando UPDATE

O comando UPDATE é usado para mudar valores de linhas de dados que já foram cadastrados anteriormente e que obedecem a determinados critérios, especificados em condições. Este comando pode alterar mais de uma linha ao mesmo tempo, caso mais de uma linha obedeça a determinada condição. As condições podem também ser representadas utilizando os operadores: AND, OR e NOT

O comando UPDATE, contém a cláusula WHERE, de forma a restringir o conjunto dos registros que serão processados pelo comando. Se não for colocada a cláusula WHERE no comando UPDATE, as alterações serão realizadas em todos os registros da tabela.

Sintaxe:

```
UPDATE NOME DA TABELA  
SET coluna1 = valor1, coluna2 = valor2  
WHERE condições
```

Exemplos:

```
UPDATE Avaliacao SET media = 10
```

```
UPDATE Avaliacao SET media = 10  
WHERE nome_aluno = "João"
```

```
UPDATE Compras SET preco = 105,  
forma_pagamento = "Cartão de Crédito"  
WHERE numero_compra = "2008708"
```

Para melhor exemplificar o funcionamento do comando UPDATE, veremos as seguintes situações:

Situação 01: Aumentar o salário de todos os funcionários em 10%

Como se pretende aumentar o salário de todos os elementos da tabela FUNCIONÁRIO, o comando UPDATE não usará a cláusula WHERE.

```
UPDATE Funcionario SET salario = salario * 1.1
```

Situação 02: Aumentar o salário do funcionário Regilan Meira e adicionar 1 ano ao tempo de serviço.

Nessa situação, estamos restringindo a atualização para o funcionário REGILAN MEIRA, sendo assim faz-se necessário o uso da cláusula WHERE.

```
UPDATE Funcionario SET salario = salario * 1.1, idade =  
idade + 1  
WHERE nome = "Regilan Meira"
```

Situação 03: Adicionar o prefixo 55 ao telefone de todos os hospedes que residem no Brasil

Nessa situação, estamos restringindo a atualização para indivíduos Brasileiros, sendo assim faz-se necessário o uso da cláusula WHERE.

```
UPDATE Hospedes SET Telefone = "55" + Telefone  
WHERE pais = "Brasil"
```

Situação 04: Adicionar R\$ 150 no salário das mulheres que possuem filhos.

Nessa situação, estamos restringindo a atualização dos dados para duas condições. Sendo assim, utilizaremos a cláusula **WHERE** e o operador **AND**.

```
UPDATE Funcionarios SET Salario = Salario + 150  
WHERE Sexo = "F" and Filhos > 0
```

Situação 05: Adicionar R\$ 150 no salário das mulheres que possuem filhos, ou homens que são casados.

Nessa situação, utilizaremos a cláusula **WHERE** , juntamente com o operador **AND** e **OR**.

```
UPDATE Funcionarios SET Salario = Salario + 150  
WHERE (Sexo = "F" and Filhos > 0) OR (Sexo = "M" and  
EstadoCivil = "Casado")
```

Situação 06: Conceder desconto de 5% nos preços dos veículos que possuírem cor diferente de preto e branco.

Nessa situação, utilizaremos a cláusula **WHERE** , juntamente com o operador **AND**.

```
UPDATE Veiculos SET Preco = Preco - Preco * 0.05  
WHERE Cor <> "Branco" AND Cor <> "Preto"
```

Situação 07: Conceder desconto de 5% nos preços dos produtos a base de leite.

Nessa situação, utilizaremos o operador LIKE. O operador LIKE permite fazer comparações de partes da string. Para isso utilizaremos dois curingas (“ % “)

```
UPDATE Produto SET Preço = Preço - Preço * 0.05
WHERE Nome Like "Leite%"
```

4.3. Comando DELETE

O comando DELETE é usado para remover linhas de uma tabela. Este comando pode remover mais de uma linha ao mesmo tempo, caso mais de uma linha obedeça a uma certa condição. As condições podem ser representadas utilizando os operadores AND, OR e NOT.

O comando DELETE, contém a cláusula WHERE, de forma a restringir o conjunto dos registros que serão processados pelo comando. Se não for colocada a cláusula WHERE no comando DELETE, serão apagados todos os registros de uma tabela.

Assim como no comando UPDATE, podemos utilizar os operadores relacionais (>, >=, <, <=, =, <>, like) e os operadores lógicos (AND, OR) para especificar as condições de exclusão de dados.

Sintaxe:

```
DELETE FROM NOME DA TABELA
WHERE <condições>
```

Exemplos:

```
DELETE FROM ESCOLA
```

```
DELETE FROM ESCOLA WHERE ALUNO = "TIAGO PEREIRA"
```

```
DELETE FROM PRODUTOS
WHERE NOME Like "LEITE%"
```

```
DELETE FROM CLIENTES
WHERE QuantidadeCompras <= 3
```

4.4. Comando SELECT

O comando SELECT é usado para consultar o banco de dados e retornar dados recuperados que satisfazem a determinada condição expressa no comando.

Sua sintaxe é representada da seguinte forma:

```
SELECT <lista de atributos>
FROM NOME DA TABELA
WHERE <condições>
```

Exemplos:

```
SELECT codigo,aluno,media
FROM NOTAS
WHERE aluno = "Tiago"
```

```
SELECT matricula,nome,responsavel,data_nascimento,cpf,rg,
endereco,codigo_curso,observacoes FROM ALUNOS
WHERE nome = "Camila"
```

```
SELECT * FROM ALUNOS
WHERE nome = "Camila"
```

Obs: o símbolo * na clausula SELECT indica que deverá ser selecionado todos os campos de uma tabela.

Para melhor exemplificar o funcionamento do comando SELECT, veremos as seguintes situações:

Situação 01: Escrever o comando SQL que permite obter o RG, Nome e o Código Postal de todos os clientes registrados no banco de dados.

```
SELECT Rg, Nome, CodigoPostal
FROM Cliente
```

Situação 02: Selecionar todos os dados de todos os pacientes cadastrados no Hospital

Nesta situação usaremos o curinga(*) , ao invés de escrever todos os campos da tabela Paciente no comando SQL.

```
SELECT *  
FROM Pacientes
```

Situação 03: Selecionar o ID, Nome, Idade e o Salário de todos os Funcionários com Idade entre 30 e 40 anos

Nesta situação usaremos o operador WHERE, juntamente com operadores lógicos e relacionais.

```
SELECT Id, Nome, Idade, Salario  
FROM Funcionario  
WHERE Idade >= 30 AND Idade <= 40
```

Situação 04: Selecionar o ID, Nome, Idade e o Salário de todos os Funcionários cuja a idade não está entre 30 e 40 anos.

Nesta situação usaremos o operador WHERE, juntamente com operadores lógicos (AND e NOT) e relacionais.

```
SELECT Id, Nome, Idade, Salario  
FROM Funcionario  
WHERE Not (Idade >= 30 AND Idade <= 40)
```

Situação 05: Selecionar todos os indivíduos que possuem sobrenome “Silva”

Nesta situação usaremos o operador Like e o curinga “%”

```
SELECT *  
FROM Pessoa  
WHERE Nome Like "%Silva%"
```

Situação 06: Selecionar a quantidade de votos dos Partidos: PT, PSDB, PSB e DEM nas eleições de 2012.

```
SELECT QuantidadeVotos  
FROM Votos  
WHERE (Partido = "PT" OR Partido = "PSDB" OR Partido =  
"PSB" OR Partido = "DEM") AND AnoEleicao = 2012
```

Podemos incluir no comando SQL uma cláusula que permita solicitar a ordenação dos resultados fornecidos por um comando SELECT. A ordenação é feita da seguinte forma: primeiro os dígitos, depois os caracteres maiúsculos e por último os caracteres minúsculos.

Representação:

0 < 1 < ... < 8 < 9 < ... < A < B < ... < Z < ... a < b < ... < z

A ordenação pode ser realizada através da cláusula **ORDER BY** no comando SELECT. Esta cláusula aparece sempre posicionada no final do comando SELECT.

Veja a sintaxe:

```
SELECT Campo1 , Campo2 , Campo3  
FROM Tabela  
WHERE Condição  
ORDER BY Campo ASC | DESC
```

- **ASC** indica ordenação ASCendente
- **DESC** indica ordenação DESCendente

Para melhor compreensão, veremos os exemplos abaixo:

Situação 07: Selecionar todos os dados da tabela Pessoa, ordenado pela Idade.

```
SELECT *  
FROM Pessoa  
ORDER BY Idade
```

OBS: Quando a ordenação for ASCendente não é necessário incluir a cláusula ASC em ORDER BY, já que a ordenação padrão é ascendente

Situação 08: Selecionar o Nome e o Salário de todas as pessoas, ordenando o resultado pelo Salário, de tal forma que os maiores salários fiquem no topo da lista.

Nesta situação usaremos a cláusula DESC de maneira que os maiores salários fiquem no topo da lista.

```
SELECT Nome, Salario
FROM Pessoa
ORDER BY Salario DESC
```

Situação 09: Resolver o mesmo problema anterior, mas só para as pessoas com Cargo de Assistente Administrativo.

Nesta situação usaremos a cláusula WHERE, antes da ordenação.

```
SELECT Nome, Salario
FROM Pessoa
WHERE Cargo = "Assistente Administrativo"
ORDER BY Salario DESC
```

Situação 10: Selecionar todos os dados da tabela Pessoa, ordenado pela Idade e pelo Salário.

```
SELECT *
FROM Pessoa
ORDER BY Idade, Salario
```

Situação 11: Selecionar da tabela Pessoa o Nome e a Idade que irão ter daqui a dois anos. O resultado deverá vir pela nova idade.

```
SELECT Nome, Idade + 2 as Nova_Idade
FROM Pessoa
ORDER BY Idade + 2
```

Situação 12: Selecionar na tabela Vendas, a Nota Fiscal, o campo Valor, o montante do Imposto (17%) do Valor, e o Valor total com Imposto.

É possível também incluir na cláusula de ordenação o número da posição, dentre as colunas a serem apresentadas, da coluna ou colunas pelas quais se pretende ordenar o resultado.

```
SELECT Nota_Fiscal, Valor, Valor * 0.17 As Imposto, Valor +
Valor * 0.17 as Valor_Total
FROM VENDAS
ORDER BY 1, 4
```

OBS: 1 está se referindo ao campo Nota_Fiscal, 4 está se referindo ao valor total da venda(valor + imposto)

Situação 13: Selecionar o conjunto das Localidades existente na tabela Postal. OBS: Não exibir localidades iguais.

A cláusula **DISTINCT** permite eliminar repetições de linhas no resultado de um comando **SELECT**. A cláusula **DISTINCT** só pode ser colocada imediatamente depois do **SELECT**.

```
SELECT DISTINCT Localidade
FROM Postal
```

É possível incluir em comandos **SELECT** as funções de agregação. As funções de agregação (funções estatísticas) têm por objetivo obter informações sobre o conjunto de linhas especificados na cláusula **WHERE** ou sobre grupos de linhas indicados na cláusula **GROUP BY**.

As funções disponíveis são:

- **COUNT**: número de linhas
- **MAX**: o maior valor da coluna
- **MIN**: o menor valor da coluna
- **SUM**: soma de todos os valores da coluna
- **AVG**: média de todos os valores da coluna

Para melhor compreensão veremos as seguintes situações:

Situação 14: Quantos Funcionários existem e quantos têm telefone fixo.

```
SELECT COUNT(Nome) AS Total_Funcionario, COUNT(Telefone) as
Total_Funcionario_Com_Telefone FROM Funcionarios
```

Situação 15: Quantos Funcionários existem em nosso Banco de Dados

```
SELECT COUNT(*) AS Total
FROM Funcionarios
```

Situação 16: Qual o maior salário e a menor idade dos funcionários da empresa.

```
SELECT MAX(Salario) AS Maior_Salario, MIN(Idade) AS
Menor_Idade FROM Funcionarios
```

Situação 17: Qual o menor e maior valor de comissão superior a 1000 e inferior a 10000 na tabela Venda

```
SELECT MAX(Comissao) AS Maior_Comissao, MIN(Comissao) AS  
Menor_Comissao  
FROM Venda  
WHERE Comissao >= 1000 AND Comissao <= 10000
```

Situação 18: Qual é o valor total das comissões a pagar?

```
SELECT SUM(Comissao) AS Total_Comissao  
FROM Venda
```

Situação 19: Qual é o salário médio dos Funcionários com mais de 40 anos.

```
SELECT AVG(Salario) AS Media_Salario  
FROM Funcionario  
WHERE Idade > 40
```

O comando SELECT também permite agrupar resultados. As cláusulas de agrupamento estão relacionadas com as funções de agregação, e são úteis no tratamento de informações de forma agrupada. As cláusulas **GROUP BY** e **HAVING** fazem parte do comando SELECT e são representadas de acordo com a sintaxe a seguir:

```
SELECT Campos  
FROM Tabela  
WHERE Condição  
GROUP BY ...  
HAVING ...
```

A cláusula GROUP BY divide o resultado de um SELECT em um grupo de resultados que serão processados pelas funções de agregação.

Situação 20: Mostrar a quantidade de CARROS vendidos agrupados pelo Modelo.

```
SELECT Modelo_Carro, Count(Nota_Fiscal) AS  
Quantidade_Carros_Vendidos  
FROM Vendas  
GROUP BY Modelo_Carro
```

Situação 21: Mostrar para cada funcionário o valor total das vendas realizadas.

```
SELECT Nome, Sum(Valor) AS Total_Vendas
FROM Vendas
GROUP BY Nome
```

A cláusula **HAVING** faz restrições ao nível dos grupos que são processados. É comum surgir a dúvida sobre **WHERE** ou **HAVING**. A diferença entre **HAVING** e **WHERE** é que a cláusula **WHERE** é usada para restringir os registros a serem considerados na seleção. A Cláusula **HAVING** restringe os grupos que foram formados depois da aplicação da cláusula **WHERE**.

Situação 22: Mostrar para cada funcionário o valor total das vendas superiores a 80000

```
SELECT Nome, Sum(Valor) AS Total_Vendas
FROM Vendas
GROUP BY Nome
HAVING Sum(Valor) > 80000
```

Situação 23: Selecionar a quantidade de cada produto vendido em uma compra, ordenados em ordem decendente.

```
SELECT Codigo_Produto,Count(Codigo_Produto) AS QUANTIDADE
FROM Compra
GROUP BY Codigo_Produto
ORDER BY 2 DESC
```

4.4.1. **Comando SELECT: ligação entre tabelas e subsconsultas**

A ligação entre tabelas (join) permite extrair, através de um único **SELECT**, informações contidas em diferentes tabelas.

A junção entre tabelas é feita colocando-se na cláusula **FROM**, as tabelas que pretende-se juntar. Veja a sintaxe:

```
SELECT Campo1, Campo2, Campo3, CampoN
FROM Tabela1,Tabela2
```

A estrutura mostrada acima representa o produto cartesiano entre as tabelas, que associa a cada linha de uma tabela, o conjunto de linhas de outra tabela.

O **Inner Join** (ligação entre tabelas), ocorre quando se juntam duas ou mais tabelas, ligando-as através da Chave Primária de uma e da Chave Estrangeira da outra. Num Inner Join são exibidos apenas os registros em que exista ligação entre as tabelas. A junção de duas ou mais tabelas, é feita através das chaves estrangeiras, na cláusula WHERE.

Considere o seguinte comando:

```
SELECT Pessoa.Nome,Postal.Cod_Postal,Postal.Localidade
FROM Pessoa,Postal
WHERE Pessoa.Cod_Postal = Postal.Codigo
```

O comando acima retornará o Nome da Pessoa (tabela Pessoa), o código Postal e a Localidade (tabela Postal), quando o código Postal que existe na tabela PESSOA for igual ao código Postal existente na tabela Postal.

Ou seja, só será mostrado os dados da tabela Postal quando eles forem relacionados com a tabela Pessoa.

Para melhor compreensão veremos as seguintes situações:

Situação 24: Selecionar as localidades das pessoas que não tem telefone.

```
SELECT Postal.Localidade
FROM Pessoa,Postal
WHERE Pessoa.Cod_Postal = Postal.Codigo AND Pessoa.Telefone
IS NULL
```

OBS: A utilização do nome da tabela antes do nome da coluna não é obrigatória (Pessoa.Telefone). **Somente quando uma coluna, tem o mesmo nome em várias tabelas, é necessário essa identificação.**

Situação 25: Selecionar a data da venda, o valor e o nome do vendedor.

A tabela Venda e Vendedor estão relacionadas através das chaves estrangeiras. (Venda.CPFVendedor = Vendedor.CPF)

```
SELECT Venda.Data,Venda.Valor,Vendedor.Nome
FROM Venda,Vendedor
WHERE Venda.CPFVendedor = Vender.CPF
```

Situação 26: Em um clube, existe sócios titulares e dependentes. Selecionar o nome e a data de nascimento de todos os dependentes. Incluir na consulta o nome do titular de cada um dos dependentes. Ordenar o resultado pelo nome dos Dependentes.

A tabela Titular e Dependente estão relacionadas através das chaves estrangeiras. (Titular.Matricula = Dependente.MatriculaTitular)

```
SELECT
Titular.Nome,Dependente.Nome,Dependente.Data_Nascimento
FROM Titular,Dependente
WHERE Titular.Matricula = Dependente.MatriculaTitular
ORDER BY Dependente.Nome
```

É possível ainda em alguns sistemas, escrever a consulta acima usando o comando **INNER JOIN**.

```
SELECT
Titular.Nome,Dependente.Nome,Dependente.Data_Nascimento
FROM Titular INNER JOIN Dependente ON Titular.Matricula =
Dependente.MatriculaTitular
ORDER BY Dependente.Nome
```

Situação 27: Selecionar todos os produtos vendidos pelo vendedor de nome Fulano de Tal

Essa relação envolve 4 tabelas: Venda, Itens_Da_Venda, Produto e Vendedor. Na consulta SELECT deve-se especificar todos os relacionamentos entre as tabelas.

```
SELECT Produto.Nome
FROM Venda,Itens_Da_Venda,Produto,Vendedor
WHERE Vendedor.Nome = 'Fulano de Tal'
AND Venda.MatriculaVendedor = Vendedor.Matricula
AND Venda.Codigo = Itens_Da_Venda.Codigo_Venda
AND Itens_Da_Venda.Codigo_Produto = Produto.Codigo
```

Utilizando SQL, podemos ainda utilizar o comando de união (UNION) que permite juntar o conteúdo de dois ou mais comandos SELECT.

Situação 28: Em uma base de dados, o cadastro de todos os alunos, professores e funcionários estão localizados em tabelas diferentes. Como obter o CPF e o Nome, de todos os indivíduos cadastrados na base dados?

Para obter os dados solicitados no slide anterior, será necessário criar 3 comandos SELECT e usar o comando UNION para unir as consultas. Veja o resultado:

```
SELECT cpf,nome FROM Professor
UNION
SELECT cpf,nome FROM Funcionario
UNION
SELECT cpf,nome FROM Aluno
```

Em uma união (UNION), o número de campos a serem selecionados em cada um dos comandos SELECT tem de ser IGUAL. O nome das colunas apresentado no resultado é o nome das colunas selecionadas na primeira instrução SELECT. Além disso, são eliminadas do resultado as linhas duplicadas, do mesmo modo que no DISTINCT, a não ser que seja utilizado UNION ALL.

Situação 29: Obter o CPF e o Nome de todos os Funcionários, Professores e Alunos cuja a cidade natal seja Ilhéus. Por fim ordenar o resultado pela coluna Nome.

Cada comando SELECT pode conter sua própria cláusula WHERE. Porém, só poderá existir uma única cláusula ORDER BY, que estará localizada no último comando SELECT, e será aplicada a todo o resultado. Veja a resolução:

```
SELECT cpf,nome FROM Professor WHERE naturalidade = 'Ilhéus'
UNION
SELECT cpf,nome FROM Funcionario WHERE naturalidade =
'Ilhéus'
UNION
SELECT cpf,nome FROM Aluno WHERE naturalidade = 'Ilhéus'
ORDER BY nome
```

O operador INTERSECT permite juntar o resultado de dois comandos SELECT, apresentando as linhas que resultam de ambos os comandos. As linhas duplicadas são eliminadas, a não ser que seja utilizado INTERSECT ALL.

Situação 30: Selecionar o nome, cpf e data de nascimento de todos os funcionários que são alunos da universidade onde trabalham. Exibir os dados ordenados de forma decrescente pela coluna nome.

```
SELECT nome,cpf,data_nascimento FROM Funcionario
INTERSECT
SELECT nome,cpf,data_nascimento FROM Aluno
ORDER BY nome DESC
```

O operador **EXCEPT** retorna todas as linhas presentes no resultado da consulta1, mas que não estão presentes no resultado da consulta2 (às vezes isto é chamado de diferença entre duas consultas).

As linhas duplicadas são eliminadas a não ser que seja utilizado EXCEPT ALL.

Situação 31: Selecionar o nome, cpf e data de nascimento de todos os funcionários que não são alunos da universidade onde trabalham. Exibir os dados ordenados de forma decrescente pela coluna nome.

```
SELECT nome,cpf,data_nascimento FROM Funcionario
EXCEPT
SELECT nome,cpf,data_nascimento FROM Aluno
ORDER BY nome DESC
```

O comando SQL permite o uso de subconsultas. Uma subconsulta consiste em um SELECT dentro de outro SELECT.

Considere o seguinte problema:

Situação 32: Qual é o nome da Pessoa com menor salário na empresa?

Para conseguir resolver a consulta acima, será necessário resolver dois problemas:

1. Qual é o valor do menor salário?
2. Qual o nome da pessoal a que esse salário corresponde?

Para resolver o primeiro problema (Qual é o valor do menor salário?) podemos usar a função de agregação MIN.

```
SELECT MIN(salario) From Funcionario
```

Com base no valor retornado acima, podemos realizar a consulta para o outro problema (Qual o nome da pessoal a que esse salário corresponde?). Considere que a primeira consulta resultou um salário de R\$ 698.

```
SELECT Nome FROM Funcionario
WHERE salario = 698
```

Podemos obter o mesmo resultado através de comandos SELECT encadeados.

```
SELECT Nome FROM Funcionario
WHERE salario = (SELECT MIN(salario) FROM Funcionario)
```

Com o comando acima, o salário de cada um dos indivíduos existentes na tabela Pessoa é comparado diretamente com o resultado obtido do comando SELECT interior.

Sendo assim, primeiro é executado a consulta interior (SELECT MIN(salario) FROM Funcionarios), e em seguida a consulta exterior pode ser executada pelo resultado que o SELECT interior devolveu.

Situação 33: Qual o nome das pessoas cujo salário é menor que a soma de suas comissões durante o mês de Janeiro/2013.

```
SELECT Nome, Salario FROM Funcionario
WHERE salario < (SELECT SUM(valor) FROM Comissao WHERE
Mês = 1 and ano = 2013 and Comissao.CPF = Funcionario.CPF)
```

Na situação apresentada acima, o sentido da execução é de fora para dentro, ou seja, o SELECT exterior envia o Salário a fim de ser comparado com o total das Comissões associadas ao CPF a que pertence o Salário.

Quando o SELECT interior depende dos dados que lhe são fornecidos pelo SELECT exterior, damos o nome de Consulta Relacionada.

4.5. Exercícios práticos

a) Considere as seguintes tabelas:

- **Curso (Codigo (PK), Nome, NumeroVagas)**
- **Instrutor (Codigo (PK), Nome, Apelido, Fone, Celular).**
- **Horario (Codigo (PK), Sala, Hora)**
- **Ministrado (DataCurso (PK), CodigoHorario (PK,FK), CodigoCurso (FK) , CodigoInstrutor (FK))**

Através de consultas SQL, faça:

- Mostre todas as tabelas do seu BD
- Insira pelo menos 5 (cinco) registros em cada tabela
- Adicione o campo Autorizado char(1) na tabela cursos
- Mostre o nome de todos os instrutores
- Mostre todos os campos da tabela horário
- Mostre o nome do curso ordenado pela quantidade de vagas

- Mostre o nome do curso onde o número de vagas seja maior que 30, ordenado pelo número de vagas em decendente
- Apague todos os horários da sala número 15
- Coloque todos os cursos da sala 1 para a sala 14
- Crie uma tabela chamada cidades com os campos codigo e nome
- Mostre todos os dados da tabela instrutores
- Apague a tabela criada anteriormente
- Mostre a quantidade de vagas totais
- Mostre a média de vagas
- Mostre o menor número de vagas
- Mostre o maior número de vagas

b) Considere o esquema relacional abaixo:

- **Animal (codigo (PK), nome, espécie, raça).**
- **Vacina (tipo (PK), nome, preço, fornecedor).**
- **Vacinacao (tipo (PK, FK), codigo (PK, FK)).**

Elabore os seguintes comandos em SQL:

- Recuperar o código dos animais que são da espécie gato ou que recebam vacina do tipo 2.
- Recuperar o tipo das vacinas que o animal de código 20 não foi vacinado.
- Recuperar o nome das vacinas tomadas pelo animal de código 30.
- Recupere todos os tipos de vacinas com preço maior que 20 e o código (tipo) seja menor que 10.
- Recuperar o nome das vacinas tomadas por todos os animais.
- Obter os dados dos animais cujo nome seja iniciado com a letra R.
- Informar quantas vacinas foram aplicadas em animais cuja raça seja labrador.
- Obter o nome e o preço da vacina mais barata cadastrada no banco de dados.

c) Considerando o banco de dados abaixo:

- **Gerente (Código (PK), Nome, NumDepto)**
- **Empregado (Código (PK), Nome, Endereço, Telefone, Data_Admissão, NumDepto, Salário)**
- **Projeto (Número (PK), Nome, Descrição, NumDepto)**
- **Departamento (Número (PK), Nome, Localização)**
- **Trabalha_Em (EmpCod (PK, FK), NumProj (PK, FK), Total_Horas_Semanais)**

Formule as seguintes consultas em SQL:

- Informe o nome e a localização de todos os departamentos.
- Obtenha todos os dados dos empregados com salário superior ou igual a 1000 Reais
- Obtenha o total de horas semanais dedicadas pelo empregado de código 'E323' ao projeto de código 'P55'.
- Encontre o nome e a descrição de todos os projetos desenvolvidos pelo departamento de Marketing.
- Obtenha o nome do gerente do projeto 'P07'.
- Obtenha os códigos dos empregados que estão participando de projetos em desenvolvimento no Departamento de Processamento de Dados.
- Obtenha o nome e o salário dos funcionários com participação acima de 13 horas semanais em, pelo menos, um projeto desenvolvido em um departamento localizado no centro da cidade.
- Obtenha o total de horas semanais dedicadas pelo departamento 'D01' ao projeto 'P07'.

d) Considerando o banco de dados abaixo, formule as seguintes consultas em SQL:

- **Professor (Código (PK), Nome, Salario)**
- **Aluno (Matrícula (PK), Nome, DataNasc, Endereco)**
- **Disciplina (Codigo (PK), Nome, HorasSemanais)**
- **Leciona (CodigoProf (PK, FK), CódigoDiscip (PK, FK))**
- **Matriculadoem (Matricula (PK, FK), CodigoDiscip (PK,FK), Nota)**

Expresse as consultas abaixo em SQL:

- Obtenha o nome de cada disciplina que possua, no máximo, quatro horas de aula por semana.
- Recupere todas as informações a respeito dos professores que possuam salários superiores a R\$ 2000,00
- Obtenha os nomes e os endereços dos alunos nascidos no dia 14 de abril de 1970.
- Forneça as matrículas de todos os alunos que cursam a disciplina de nome "Cálculo II" e tiraram nota menor ou igual a 4.0.
- Informe a identidade e o salário dos professores que trabalhem em disciplinas com mais de 10 horas por semana.

- Obtenha os nomes dos professores das disciplinas cujo os alunos residam em Itabuna e obteve nota menor ou igual a 7.0.
- Obtenha os dados do aluno que obteve a maior nota na disciplina de nome “Lógica”.
- Obtenha os nomes dos professores das disciplinas cuja nota não tenha sido cadastrada no Banco de Dados.
- Qual a quantidade de professores que trabalham mais de 20 horas semanais na Instituição?
- Obtenha o nome dos professores que não estejam lecionando nenhuma disciplina neste período

5. Segurança na transação e acesso a dados

5.1. Segurança no acesso a dados (Comandos DCL)

Para controlar o acesso à informação, podemos criar um conjunto de regras para acesso ao SGDB, bem como conceder/eliminar uma determinada permissão ou privilégio a um usuário. Os comandos de SQL para definir mecanismos de segurança são conhecidos como DCL, que quer dizer **Data Control Language**, ou linguagem de controle de dados.

Os comandos utilizados são:

```
CREATE; DROP e ALTER USER (MYSQL) / LOGIN (SQL SERVER)
```

```
GRANTE e REVOKE.
```

Os bancos de dados permitem a criação e a manutenção de usuários através de comandos da linguagem SQL.

O SQL Server e o MYSQL possuem comandos diferentes para a criação de usuário em seus SGBDS.

No MYSQL e outros SGBD que usam PL-SQL, a sintaxe básica é:

```
CREATE USER Nome_do_usuario  
IDENTIFIED BY Senha_do_usuario
```

Situação 01: Criar no MYSQL um usuário com o nome: *fabio* e a senha: *123456*

```
CREATE USER fabio  
IDENTIFIED BY '123456'
```

Para criar usuário no SQL Server, usamos a seguinte sintaxe Transact-SQL:

```
CREATE LOGIN Nome_do_usuario  
WITH PASSWORD = Senha_do_usuario
```

Situação 02: Criar no SQL Server, um usuário com nome: *fabio* e senha: *123456*.

```
CREATE Login fabio  
WITH PASSWORD = '123456'
```

Depois de criado o usuário, é necessário ainda mapear esse usuário para o login criado, assim seu usuário conseguirá se logar no SQL Server e entrar no banco de dados desejado:

```
CREATE USER fabio FROM LOGIN fabio;
```

Se for necessário alterar alguma das características do usuário, como a senha, poderá ser utilizado o comando ALTER USER/LOGIN, cuja sintaxe é:

MYSQL

```
ALTER USER Nome_do_usuario  
IDENTIFIED BY Senha_do_usuario
```

SQL Server

```
ALTER USER Nome_do_usuario  
WITH PASSWORD = Senha_do_usuario
```

Para eliminar um usuário utilizamos o comando DROP USER/LOGIN, cuja sintaxe é:

MYSQL

```
DROP USER Nome_do_usuario
```

SQL Server

```
DROP LOGIN Nome_do_usuario
```

Além da segurança de dados em nível de usuário, podemos atribuir privilégios diferentes a cada um dos usuários do SGBD. Para isso usamos o comando **GRANT**, cuja sintaxe é:

```
GRANT Privilegio ON Objeto TO Usuario
```

Os privilégios que se podem atribuir são:

PRIVILÉGIO	DESCRIÇÃO
SELECT/SELECT (Lista)	Permite selecionar todas as colunas de uma tabela ou colunas específicas em listas
INSERT/INSERT (Lista)	Permite inserir registros em todas as

	colunas da tabela ou valores na lista das colunas de uma tabela
UPDATE/UPDATE (Lista)	Permite alterar conteúdos em todas as colunas da tabela ou alterar o conteúdo da lista das colunas de uma tabela
DELETE	Permite apagar registros de uma tabela
REFERENCES/REFERENCES (Lista)	Permite referenciar todas as colunas de uma tabela ou a lista das colunas de uma tabela
EXECUTE	Permite a execução de Stored Procedure
ALL	Permite conceder todos os privilégios.

Situação 04: Conceder privilegio de inserir,remover e atualizar dados da tabela Funcionário para o usuário gerente_rh

```
GRANT INSERT,DELETE,UPDATE ON Funcionario TO gerente_rh
```

Situação 05: Conceder privilegio de atualizar os dados dos campos Localidade da tabela Endereços para o usuário fabio.

```
GRANT UPDATE(Localidade)ON Enderecos TO fabio
```

Situação 06: Conceder privilegio de exibir os dados nome, salário e matricula da tabela Funcionário para o usuário fabio.

```
GRANT SELECT(Nome,Salario,Matricula) ON Funcionario TO fabio
```

O comando **REVOKE** permite retirar privilégios concedidos através do comando **GRANT**. A sua sintaxe é:

Exemplos:

```
REVOKE INSERT, UPDATE ON Funcionario FROM gerente_rh
```

```
REVOKE UPDATE,DELETE ON Enderecos FROM fabio
```

```
REVOKE ALL ON Produtos FROM fabio
```

5.2. Segurança na transação de dados

Quando um aplicativo tenta executar um comando em um banco de dados, este comando pode não ser executado de maneira correta, por alguns dos simples motivos abaixo:

- A sintaxe não estava correta.
- Erro de em apenas uma parte do comando
- Um dos recursos do banco de dados ficou indisponível durante a execução do comando, não permitindo que todo o comando seja executado.
- Fatores externos ao SGBD como queda de energia, travamento do servidor, etc.

Para ilustrar o conceito de transação, usaremos a transferência de dinheiro de uma conta para outra. Suponha que o indivíduo João pretenda transferir R\$ 1000,00 da sua conta bancária para a conta do indivíduo Pedro. Uma transferência corresponde a um depósito em uma conta e a retirada, do mesmo valor em outra conta.

O conjunto de comandos SQL a serem executados para realizar esta transação seria constituído por dois comandos UPDATE.

```
UPDATE Tabela_Conta  
SET Saldo = Saldo + 1000  
WHERE Conta = 'Conta Pedro'
```

```
UPDATE Tabela_Conta  
SET Saldo = Saldo - 1000  
WHERE Conta = 'Conta João'
```

O primeiro comando creditaria o montante na conta destino (Pedro) e o segundo comando faria a retirada do mesmo montante na conta origem

PROBLEMA: O que aconteceria se existisse algum tipo de problema e apenas o primeiro UPDATE fosse executado?

RESPOSTA: Considerando que apenas o 1º Update fosse executado, o Pedro ficaria com mais R\$ 1000,00 na sua conta, enquanto o João ficaria com seu saldo inalterado.

Para resolver esse tipo de problema, usamos a noção de **TRANSAÇÃO**.

Uma transação consiste em uma unidade lógica de trabalho em que todas as operações são realizadas ou nenhuma das operações é realizada. Sendo assim, se alguma falha se verificar entre os dois comandos UPDATE, independente da ordem pela qual são executadas, as duas contas ficarão inalteradas.

O início de uma transação pode ser indicado através de um comando específico: **BEGIN TRANSACTION**.

Para finalizar uma transação usamos os comandos **COMMIT** ou **ROLLBACK**.

Quando uma transação é realizada, os dados alterados não são escritos diretamente no banco de dados. São colocados em um repositório até que o usuário aplique o comando **COMMIT**, tornando definitiva todas as alterações realizadas, ou o comando **ROLLBACK**, ignorando todas as alterações realizadas desde o início da transação.

O conceito de transação só se aplica aos comandos que fazem a manipulação de dados, isto é, aos comandos INSERT, UPDATE e DELETE.

Abaixo um exemplo de código utilizando uma transação no SQL Server:

```
BEGIN TRANSACTION
COMANDO SQL (INSERT, UPDATE, DELETE)
IF @@ERROR <> 0
ROLLBACK
ELSE
COMMIT
```

Situação 07: Criar uma transação que atualiza o salário de todos os funcionários de uma empresa em 5%.

```
BEGIN TRANSACTION
UPDATE funcionario SET salario = salario + salario*0.05
IF @@ERROR <> 0
ROLLBACK
ELSE
COMMIT
GO
```

Neste exemplo, iniciamos a transação com o **BEGIN TRANSACTION**. Imagine que a tabela Funcionário possui 5000 registros que satisfaçam a condição e que por algum motivo no registro 3571 houve um problema.

Neste caso, a variável do SQL Server chamada @@ERROR vai guardar o código do erro, o comando ROLLBACK será executado e os 3571 registros já atualizados serão automaticamente voltados ao seu estado anterior.

Mas se tudo correr bem, o comando COMMIT é executado e todos os 5000 registros que atenderam à condição serão atualizados com êxito.

6. Acesso a dados com PHP - MySQL

Para apresentar o acesso a dados, usaremos com exemplo uma aplicação em PHP/MySQL.

A Base de Dados mais comum para a linguagem php é o MySQL. Podemos utilizar também outras bases de dados como, Oracle,SQLS Server, Postgres, etc.

Para o PHP interagir com uma base de dados SQL, existem três comandos básicos que devem ser utilizados:

- Um que faz a ligação com o servidor da base de dados
- Um que seleciona a base de dados a ser utilizada
- Um que executa um comando SQL.

OBS: Em geral, a maioria das linguagens e ambiente de programação utiliza os três procedimentos descritos acima para acesso a dados.

A função **mysql_connect**, é responsável por abrir uma conexão com um servidor MySQL.

Se a conexão for bem sucedida, um link para o banco de dados é retornado, caso contrário a função retorna **FALSE**.

A função **mysql_connect** recebe três parâmetros: **server**, **username**, **password**.

Estabelecendo conexão PHP -> MySQL:

```
<?php

$link = mysql_connect('localhost' , 'root', 'ad$14B' );

if (!$link ) {
    echo 'Não foi possível conectar: ' . mysql_error();
}

echo 'Conexão bem sucedida' ;
mysql_close( $link);

?>
```

Depois de conectar a uma base de dados, o próximo passo é selecionar qual banco de dados utilizar. Para isso, usamos a função: **mysql_select_db**

Esta função recebe dois parâmetros: o nome do banco de dados e a conexão com o banco de dados (opcional).

OBS: Caso não seja especificado a conexão do banco de dados, o padrão é a conexão da última chamada da função: **mysql_connect**

Selecionando o Banco de Dados:

```
<?php

$db = mysql_select('empresa');

if (!$db) {
    echo 'Não foi possível selecionar a base de dados ' .
mysql_error();
}
echo 'Banco de Dados selecionado' ;

?>
```

Após efetuar a conexão e selecionar a base de dados, podemos usar a função **mysql_query** para efetuar as consultas SQL

Exemplo:

```
<?php
if (mysql_select_db("empresa",$link))
{
    $resultado = mysql_query("select * from tabela");
}
else
{
    echo("O banco de dados não pode ser selecionado.")
}
?>
```

O PHP retornará os resultados de uma consulta em um vetor. Dependendo de como o resultado for obtido será um vetor cujos índices são numéricos ou o nome dos campos.

Utilizamos a função **mysql_fetch_array** para obter todas linhas de uma query:

```
$linha = mysql_fetch_array($resultado)
```

O código a seguir exibe em uma página web os resultados de cada linha de dados após a execução de uma consulta SQL.

```
<?php
while($linha = mysql_fetch_array($result)){
echo($linha["nome_do_campo"]."<br>");
}
?>
```

Para executar um comando de manutenção (**INSERT**, **UPDATE**, **DELETE**), utilizamos o mesmo procedimento para o **SELECT**, porém não é necessário obter os resultados da consulta em um array.

Veja exemplo:

```
$nome="lucas";
$consulta="INSERT into amigos (nome) VALUES ($nome)";
$resultado=mysql_query($consulta,$conect);
if(! $resultado){
echo "Falha ao executar o comando: " . mysql_error();
}
else{
echo "Dados inseridos com sucesso.";
}
```

Existem diversos tipos de entradas: rádio, checkbox, área de textos, botões, caixas de textos, etc.

Os formulários de entrada de dados são desenvolvidos em HTML e se localizam dentro da tag <form>.Veja exemplo:

```
<form action="gravar.php" method="post" >
<label>Nome</label>
<input type="text" name="nome" size="50" </input>
<input type="checkbox" name="sexo" value="casado"</input>
</form>
```

Para poder enviar as informações, seu formulário deve conter um botão "submit", isso se consegue através do comando:

```
<input type="submit" value="Texto do Botão">
```

Todos os campos que serão tratados no script PHP devem conter o parâmetro "NAME", caso contrário, os dados não serão passados para o script PHP.

O exemplo completo desta aplicação está disponível a seguir:

PÁGINA: cadastrar_aluno.php

Este arquivo .php contém o formulário para preenchimento de dados para cadastrar um aluno.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />
<title>Cadastrar Aluno</title>
</head>
<style type="text/css">
body
{
font-family:arial;
}
ul,li{list-style-type:none;
padding-left:10px;
}
fieldset
{
width:350px;
}
</style>

<body>
<form action="cadastrar_aluno_gravar.php" method="post">
<fieldset>
<legend>Cadastrar Aluno</legend>
<ul>
<li><label>Nome: </label><br /><input type="text"
name="nome" size="45"/></li>
<li><label>Data de Nascimento: </label><br /><input
type="text" name="datanascimento" size="20"/></li>
<li><label>Sexo: </label><br /><input type="radio"
name="sexo" size="45" value="m"/>Masculino<input type="radio"
name="sexo" size="45" value="f"/>Feminino</li>
<li><label>Curso: </label><br /><select name="curso">
<?php

//cria um link de conexão com o MYSQL. É passado para a
função 3 parâmetros, o primeiro o nome do servidor, em seguida o
nome do usuário e por último a senha, que neste caso está vazia
$link = mysql_connect("localhost" , "root", "" );
```

```

        //se o link com o servidor for realizado, será selecionado
o banco de dados
        if ($link ) {
            //seleciona o banco de dados usado como exemplo: conexasopho
            $db = mysql_select_db("conexasopho");
            //se o banco de dados for selecionado corretamente, será
executado o comando SQL para buscar os dados da tabela curso
            if ($db) {
                //executa o comando SQL para buscar os dados da tabela
curso e gravar na variavel resultado.
                $resultado = mysql_query("select * from curso");
                //a repetição abaixo é usada para buscar cada linha de
dados gerado pelo resultado da consulta. Enquanto houver linha de
dados, os dados serão lidos
                //Para cada linha, é gerado uma option para o select criado
antes de ser chamado o código PHP. Cada option, o valor do
atributo value será a coluna idCurso de uma linha de dados, e o
nome exibido em uma option será a coluna curso de uma linha de
dados.
                while($linha = mysql_fetch_array($resultado)){
                    ?>
                    <option value="<?php echo($linha["idcurso"]);?>"><?php
echo($linha["curso"]);?></option>
                    <?php
                    }

                }
            }
            //fecha a conexão.
            mysql_close( $link);
            ?>
            </select></li>
            <li><br /><input type="submit" value="Gravar dados" />
            </ul>
            </fieldset>
            </form>
            </body>
            </html>

```

PÁGINA: cadastrar_aluno_gravar.php

Este arquivo .php contém o código de acesso ao banco de dados com o script para gravação dos dados informados no formulário web para cadastrar um aluno.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />
<title>Cadastrar Aluno</title>
<style type="text/css">
body
{
font-family:arial;
}
</style>
</head>

<body>
<?php
//cria um link de conexão com o MYSQL. É passado para a
função 3 parâmetros, o primeiro o nome do servidor, em seguida o
nome do usuário e por último a senha, que neste caso está vazia
$link = mysql_connect("localhost" , "root", "" );
//se o link com o servidor for realizado, será selecionado
o banco de dados
if ($link ) {
//seleciona o banco de dados usado como exemplo: conexasphp
$db = mysql_select_db("conexasphp");
//se o banco de dados for selecionado corretamente, será
executado o comando SQL para gravar no banco de dados os dados
vindos do formulário cadastrar_aluno.php
if ($db) {
//as quatro variáveis a seguir recebem os valores vindos do
formulário anterior que são passados via método POST
$nome=$_POST["nome"];
$sexo=$_POST["sexo"];
$datanascimento=$_POST[datanascimento];
$curso=$_POST[curso];
//consulta SQL se inserção. OS campos que são varchar no
banco são passados como string, para isso usamos as aspas
simples
```

```

        $consulta="INSERT into aluno
(nome,datanascimento,sexo,curso) VALUES
('$nome',$datanascimento,'$sexo',$curso)";
        //executa o comando e guarda o retorno na variavel
resultado. Por último é testado o tipo de retorno(true ou false)
        $resultado=mysql_query($consulta,$link);
        if(!$resultado){
            echo "Falha ao executar o comando: " . mysql_error();
        }
        else{
            echo "Dados inseridos com sucesso.";
        }
    }
}

?>
</body>
</html>

```

PÁGINA: `exibir_aluno.php`

Este arquivo .php contém o código de acesso ao banco de dados com o script para seleção de dados e apresentação dos dados dos alunos em uma tabela html.

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />
<title>Alunos Matriculados</title>
<style type="text/css">
body
{
font-family:arial;
}
</style>
</head>

<body>

<table>
<caption>ALUNOS MATRICULADOS</caption>
<tr>
<th>Nome</th>

```



```

<th>Curso</th>
</tr>
<?php

    //cria um link de conexão com o MYSQL. É passado para a
    função 3 parâmetros, o primeiro o nome do servidor, em seguida o
    nome do usuário e por último a senha, que neste caso está vazia
    $link = mysql_connect("localhost" , "root", "" );
    //se o link com o servidor for realizado, será selecionado
    o banco de dados
    if ($link ) {
        //seleciona o banco de dados usado como exemplo: conexaophp
        $db = mysql_select_db("conexaophp");
        //se o banco de dados for selecionado corretamente, será
        executado o comando SQL para buscar os dados da tabela curso
        if ($db) {
            //executa o comando SQL para buscar os dados da tabela
            curso e gravar na variável resultado.
            $resultado = mysql_query("select aluno.nome,curso.curso
            from aluno,curso where curso.idcurso = aluno.curso");
            //a repetição abaixo é usada para buscar cada linha de
            dados gerado pelo resultado da consulta. Enquanto houver linha de
            dados, os dados serão lidos
            //Para cada linha, é gerado uma nova linha na tabela de
            exibição dos dados. Cada coluna da linha, receberá uma das
            colunas do resultado da consulta em cada linha de dados.
            while($linha = mysql_fetch_array($resultado)){
                ?>
                <tr>
                <td><?php echo($linha["nome"]);?></td>
                <td><?php echo($linha["curso"]);?></td>
                </tr>
                <?php
                }
            }

            ?>
        </body>
    </html>

```

7. Acesso a dados com Visual C# - SQL Server

Para iniciar o acesso a dados com o Visual C# o primeiro passo é definir qual o SGBD que será utilizado e também a biblioteca de dados, que dependerá do SGBD definido.

Exemplos:

- Access: OleDb
- SQL Server: SqlClient
- My SQL: MySQLClient

No caso do MySQL, Postgres e outros SGBD que não são nativos da plataforma .Net, será necessário fazer o download do conector ao SGBD. Após a instalação do conector específico, o acesso a dados é realizado da mesma forma que será mostrado nesta apostila.

Para apresentar o acesso a dados, iremos realizar um cadastro de contato que será composto dos seguintes dados:

- Código (Chave Primária)
- Nome
- E-mail
- Celular

Após a criação do banco de dados e da tabela contato no SQL Server, será necessário criar um formulário para cada campo da tabela Contato. Como o campo código é uma chave primária e será gerado automaticamente, não é necessário criar esse campo no formulário.

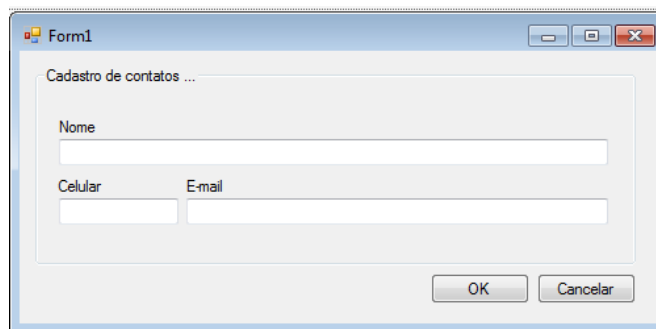


Figura 13 - Cadastro de contatos

Neste exemplo apresentado na apostila, será mostrado o acesso de dados de forma manual, ou seja, escrevendo linha por linha de código. O Visual C# pode facilitar este acesso, utilizando controles disponíveis na toolbox -> data, mas para melhor compreensão dos mecanismos de acesso a dados, utilizaremos apenas linha de comando para apresentação. Na primeira linha de código da aplicação devemos importar a biblioteca a ser utilizada para acesso a dados, neste caso o SQL Client.

```
using System.Data.SqlClient;
```

Em seguida devemos incluir os seguintes objetos de acesso a dados:

- O primeira que será uma objeto do tipo **Connection**, que armazenará as informações referente a string de conexão
- O segundo será do tipo **Command** que será responsável pela execução do comando SQL.

OBS: Estes objetos deverão ser criados no início da classe do formulário, para que todos os eventos criados possam utilizar estes objetos, que nestes casos são GLOBAIS para o formulário ao qual eles foram criados.

```
SqlConnection conexao = new SqlConnection("string de conexão do banco de dados");  
  
SqlCommand comando = new SqlCommand();
```

A string de conexão do banco de dados, mostrado no objeto SqlConnection (ou connection string) é um código utilizado pelo sistema para se conectar ao banco de dados. Cada banco de dados tem a sua própria string. No SQL Server, a string de conexão tem o seguinte formato:

```
Data Source=Endereço do Servidor SQL Server;Initial  
Catalog=nome do banco de dados;User Id=nome do usuário do banco  
de dados;Password=senha do usuário;
```

Em seguida vamos programar o evento Click() do botão OK que irá inserir os valores informados nas caixas de texto do formulário nos respectivos campos da tabela Contato.

O código de acesso a dados ficará dentro de uma estrutura de tratamento de exceção chamada: **try...catch...finally**

Dentro da estrutura **try** ficará o código que insere o registro no banco de dados.

Dentro da estrutura **catch** ficará o código caso algum erro aconteça nos comandos presente na estrutura **try**

Por último ainda existe **finally** que independente de erro ou não será sempre executado.

Primeiro vamos atribuir ao atributo comando o atributo conexão. Dessa forma o comando SQL presente no atributo COMANDO, será executado no banco de dados informado no atributo CONEXAO.

```
comando.Connection = conexao;
```

Depois devemos atribuir qual o comando SQL a ser executado. Veja no exemplo abaixo onde é representada a estrutura para o comando INSERT.

```
comando.CommandText = "INSERT INTO Contato(Nome,Celular,Email)";  
comando.CommandText += "values ('" + txtNome.Text + "', '" + txtCelular.Text + "  
''";  
comando.CommandText += "'" + txtEmail.Text + "')";
```

No código mostrado acima, observe o seguinte:

```
INSERT INTO Contato(Nome,Celular,Email)
```

O trecho representa a primeira parte do comando SQL para inserção, onde está sendo inserido os campos Nome,Celular,Email na tabela Contato

```
values ('" + txtNome.Text + "', '" + txtCelular.Text + "',  
'" + txtEmail.Text + "')
```

O trecho acima representa os valores que serão inseridos, no qual estão separados pela sintaxe " + campo do formulário + "

O próximo passo é composto de 3 operações

- Abrir o banco de dados
- Executar o comando
- Fechar o banco de dados

Abaixo veja o código para realizar essas operações:

```
conexao.Open();  
comando.ExecuteNonQuery();  
conexao.Close();
```

O código representado acima é válido para os comandos de:

- Inserir: INSERT
- Atualizar: UPDATE
- Excluir: DELETE

Veja abaixo um exemplo para atualizar um registro no banco de dados a partir do Visual Basic:

```
comando.CommandText = "UPDATE Contato set Nome = '" +  
txtNome.Text + "' where Codigo = '" + Convert.ToInt32(codigo) +  
" '";
```

Veja abaixo o comando para excluir um registro a partir do Visual Basic

```
comando.CommandText = "Delete From Contato where Codigo =  
'" + Convert.ToInt32(codigo) + "'";
```

Para fazer as operações de CONSULTA ou SELECT, será necessário além um objeto Conexao, um objeto do tipo Adapter e um outro objeto do tipo dataset

O objeto Adapter é o responsável por executar um comando de consulta, e o objeto DataSet é o responsável por guardar os valores retornados

Além dos objetos de conexão será necessário um componente visual chamado **DataGridView** que é responsável por apresentar os dados que foram retornados

pela Consulta ao Banco de Dados. O componente DataGridView está localizado na toolbar -> data. Veja abaixo como ficará o formulário.

The screenshot shows a Windows form titled 'Form1'. It has a section titled 'Cadastro de contatos ...' with three text boxes: 'Nome', 'Celular', and 'E-mail'. Below these are 'OK' and 'Cancelar' buttons. At the bottom, there is a section titled 'Contatos cadastrados' containing a DataGridView. The DataGridView has four columns: an empty column, 'Nome', 'Celular', and 'E-mail'. The first row has an asterisk (*) in the first column and empty cells in the others. The rest of the grid is shaded grey.

Figura 14 - Formulário com Datagridview

Para preencher o datagridview criaremos uma função que preenche o datagridview com todos os registros da tabela contato. Antes de codificar a função, devemos declarar mais dois objetos: um **dataset** e um **adapter**. O **dataset** e o **adapter** deverão estar declarados no mesmo local dos objetos connection e command, para que também sejam GLOBAIS ao formulário.

A declaração do objeto adapter e do objeto dataset deve ser como apresentado abaixo:

```
SqlDataAdapter adapter = new SqlDataAdapter();  
DataSet ds = new DataSet();
```

A função **preencheGrid** deverá ser criada conforme o código a seguir. A decisão de criar a função **preencheGrid** é pela razão de em qualquer necessidade de preencher o grid, é só realizar a chamada desta função, não sendo necessário criar novo código para cada nova visualização da Grid.

```

private void preencheGrid()
{
    ds = new DataSet();
    adapter = new SqlDataAdapter("SELECT Nome,Celular,Email from
Contato", conexao);
    adapter.Fill(ds, "Contato");
    dataGridView1.DataSource = ds;
    dataGridView1.DataMember = "Contato";
}

```

A função criada tem a responsabilidade de preencher o **dataset** com a consulta SQL presente no adapter. Em seguida é atualizada as propriedades **DataSource** e **DataMember** do dataset

- DataSource: fonte de dados(dataset)
- DataMember: tabela do banco de dados(contato)

Depois de criada a função é chamada no evento Load do formulário e no evento click do botão OK, para que a cada novo contato cadastrado, o grid seja atualizado.

The screenshot shows a Windows application window titled "Form1". Inside the window, there is a section titled "Cadastro de contatos ...". This section contains three text input fields: "Nome" (with the value "Dilma Rousseff"), "Celular" (with the value "9566-8874"), and "E-mail" (with the value "dilma@planalto.gov.br"). Below these fields are two buttons: "OK" and "Cancelar".

Below the input fields, there is a section titled "Contatos cadastrados" which contains a data grid. The grid has four columns: "Nome", "Celular", and "E-mail". The grid contains five rows of data:

	Nome	Celular	E-mail
	Regilan	8856-9568	regilan@hotmail....
	Aline	8856-9871	line@gmail.com
▶	Fernando	9985-1485	
	Dilma Rousseff	9566-8874	dilma@planalto.g...
*			

Figura 15 - Formulário de cadastro de contato

O código completo da aplicação é mostrado a seguir.

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

using System.Data.SqlClient;

namespace WindowsFormsApplication4
{
    public partial class Form1 : Form
    {
        SqlConnection conexao = new SqlConnection("string de conexão do banco de
dados");

        SqlCommand comando = new SqlCommand();

        SqlDataAdapter adapter = new SqlDataAdapter();

        DataSet ds = new DataSet();

        private void preencheGrid()
        {
            ds = new DataSet();
            adapter = new SqlDataAdapter("SELECT Nome,Celular,Email from
Contato", conexao);
            adapter.Fill(ds, "Contato");
            dataGridView1.DataSource = ds;
            dataGridView1.DataMember = "Contato";
        }

        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            preencheGrid();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            comando.Connection = conexao;
            comando.CommandText = "INSERT INTO Contato(Nome,Celular,Email)";
            comando.CommandText += "values ('" + txtNome.Text + "', '" +
txtCelular.Text + "',";
            comando.CommandText += "'" + txtEmail.Text + "')";

            conexao.Open();
            comando.ExecuteNonQuery();
            conexao.Close();

            preencheGrid();
        }
    }
}

```



```
}  
  }  
}
```