SQL

Structured Query Language *(Linguagem de Consulta Estruturada) é uma linguagem de pesquisa declarativa para banco de dados relacional. Muitas das características originais da SQL foram inspiradas na álgebra relacional. SQL foi desenvolvida originalmente no início dos anos 70 nos laboratórios da IBM em San Jose, dentro do projeto System R, que tinha por objetivo demonstrar a viabilidade da implementação do modelo relacional proposto por E. F. Codd. O nome original da linguagem era SEQUEL, acrônimo para "Structured English Query Language" (Linguagem de Consulta Estruturada), vindo daí o fato de, até hoje, a sigla, em inglês, ser comumente pronunciada "síquel" ao invés de "és-kiú-él", letra a letra. No entanto, em português, a pronúncia mais corrente é a letra a letra: "ésse-quê-éle"*

Versão 1.1

Apostila destinada ao curso com carga horária de 20 (vinte) horas

Sumário

1. Introdução

Devido a carência de literatura destinada ao ensino de Banco de Dados e SQL para os estudantes, elaboramos a presente apostila, que é utilizada para estabelecer um conjunto de conhecimentos determinados e introduzir o estudante no mundo dos SGBD[1] e da Linguagem SQL.

A primeira versão da linguagem SQL, chamada SEQUEL[2], surgiu em 1974 nos laboratórios da IBM na Califórnia/EUA. Entre 1976 e 1977 foi revisada e ampliada, mais tarde seu nome foi alterado para SQL[3].

Devido ao sucesso da nova forma de consulta e manipulação de dados dentro de um ambiente de banco de dados, sua utilização tornou-se cada vez maior. Diversos SGBD utilizam a SQL como linguagem padrão para o acesso às bases de dados. Entre eles podemos citar:

- DB2 da IBM
- ORACLE e MySQL da Oracle Corporation
- POSTGRES da Postgres
- SYBASE da Sybase INC
- SQL Server da Microsoft
- Ingres da Computer Associates

Em 1982 o ANSI[4] tornou a SQL a linguagem padrão para a manipulação de dados em ambiente relacional. A linguagem SQL apresenta vários enfoques:

- **Linguagem interativa de consulta (query AdHoc)** – Através de comandos SQL os usuários podem montar consultas poderosas, sem a necessidade da criação de um programa, podendo utilizar softwares para a montagem de relatórios.
- **Linguagem de programação para acesso às bases de dados** – Comandos SQL podem ser embutidos nas linguagens de programação (como por exemplo C, C++, Java, Visual Basic) e acessar os dados armazenados em uma base de dados relacional.
- **Linguagem de administração de banco de dados** – O Administrador do Banco de Dados (DBA[5]) pode utilizar comandos SQL para realizar diversas tarefas relacionadas com a manutenção dos objetos no banco de dados.
- **Linguagem de consulta em ambiente cliente/servidor** – Os softwares sendo processados nos computadores dos clientes podem se utilizar de comandos SQL

1 Sistema Gerenciador de Banco de Dados
2 Structured Query English Language
3 Structured Query Language
4 American National Standard Institute
5 DataBase Administrator

para se comunicarem, através de uma rede, com um SGBD sendo processado em uma máquina servidora.

- **Linguagem para bancos de dados distribuídos** – A SQL é também a linguagem padrão para a manipulação de dados em uma base de dados distribuída.

A SQL está dividida em:

- **Linguagem de definição de dados (DDL)** – Permite ao usuário a definição da estrutura e organização dos dados armazenados e de suas relações existentes.
- **Linguagem de manipulação de dados (DML)** – Permite a um usuário, ou a um programa de aplicação, a execução de operações de inclusão, remoção, seleção ou atualização de dados previamente armazenados na base de dados.

SQL é a mesma coisa que SQL Server?

Não. **SQL** é uma linguagem universal para trabalhar com banco de dados relacionais, **SQL Server** é um Software de Banco de Dados da Microsoft.

Definições

Segundo Korth, "*Banco de Dados pode ser definido como uma coleção de dados inter-relacionados, cujo conteúdo informativo representa a qualquer instante, o estado de uma determinada aplicação*". E segundo Navathe, "*Banco de Dados é uma coleção de dados interligados por uma semântica comum*".

Banco de Dados Relacional

O modelo de dados relacional representa os dados contidos em um Banco de Dados através de relações. Essas relações contêm informações sobre as entidades representadas e seus relacionamentos[6].

Banco de Dados Orientado a Objetos

Representam os dados como coleções que obedecem propriedades. São modelos geralmente conceituais dispondo de pouquíssimas aplicações reais. Cada objeto tem características próprias (atributos) com ações próprias (métodos)[7].

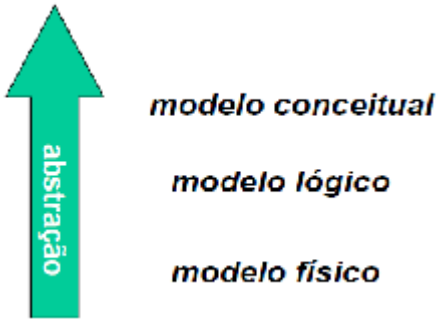
Abstração de Dados

Abstração é a habilidade de concentrar nos aspectos essenciais de um contexto qualquer, ignorando características menos importantes, ou seja, são propriedades comuns de um conjunto de objetos, omitindo os detalhes.

O sistema de banco de dados tem seu lado abstrato para o usuário, ou seja, para o usuário que vai usar o banco de dados não importa qual unidade de armazenamento, não importa

6 SILVESTRE, Eduardo. Banco de dados relacional, 2002.
7 SILVESTRE, Eduardo. Banco de dados orientado a objetos, 2002.

seu tamanho, ou banco de dados onde vai ser inserido seus dados, o importante é que na hora que for preciso procurar ou realizar uma consulta todos os dados estejam disponíveis.



A abstração é dividida em três níveis a seguir descritos:

1. **Modelo conceitual**: define quais os dados que serão armazenados e seu relacionamento.
2. **Modelo lógico**: é vista pelo usuário e opera os comandos de manipulação e operação dos dados. Os dados são apresentados como seriam percebidos por usuários finais.
3. **Modelo físico**: é o nível mais baixo de abstração, em que define efetivamente de que maneira os dados serão armazenados.

Integridade dos dados

Auxilia no processo de definição da integridade dos dados, protegendo contra corrupções e inconsistências geradas por falhas do sistema de computação, ou por erros nos programas de aplicação.

Múltiplas visões dos dados

Permite ao criador do banco de dados levar diferentes visões dos dados aos diferentes usuários.

Definição dinâmica dos dados

Através da linguagem SQL pode-se alterar, expandir ou incluir, dinamicamente, as estruturas dos dados armazenados, com máxima flexibilidade.

Base de Dados

BD ou Banco de dados é um programa de computador utilizado para organizar e guardar dados. Este conceito é parte integrante dos chamados Sistemas de Informação, onde os dados são coletados, processados, organizados e armazenados.

A parte de coleta, tratamento e processamento dos dados é feito por softwares específicos como por exemplo ERP, CRM, *Web Service*, sites, entre outros. Porém, o armazenamento e organização desses dados são realizados em um BD, podendo ser Oracle, SQL Server, MySQL ou outro. Deste modo, um BD pode ser usado para referenciar um software, como:

BD Oracle, BD SQL Server, entre outros; E neste caso eles são chamados de SGDB[8]. Mas pode ser usado para referenciar uma base de dados, que nada mais é do que o produto dos SGDB.

A maioria dos BD atuais são relacionais, pois suas estruturas são formadas por tabelas e do relacionamento entre as tabelas. Tanto um BD robusto como o Oracle ou pequenos como o HyperSQL adota este mesmo padrão, o que torna possível a integração de dados entre os diversos bancos.

Um BD é composto pelas seguintes partes:

- **Dicionário de Dados**: contém o esquema do BD, suas tabelas, índices, forma de acesso e relacionamentos existentes.
- **Gerenciador de Acesso ao Disco**: O SGBD utiliza o Sistema Operacional para acessar os dados armazenados em disco, controlando o acesso concorrente às tabelas do BD. O Gerenciador controla todas as pesquisas solicitadas pelos usuários no modo interativo, os acessos do compilador DML, os acessos feitos pelo Processador do BD ao Dicionário de Dados e também aos próprios dados.
- **Compilador DDL (*Data Definition Language*)**: processa as definições do esquema do Banco de Dados, acessando quando necessário o Dicionário de Dados do BD.
- **Processador do BD**: manipula requisições ao próprio BD em tempo de execução. É o responsável pelas atualizações e integridade do BD.
- **Processador de Pesquisas dos Usuários**: analisa as solicitações, e se estas forem consistentes, aciona o Processador do BD para acesso efetivo aos dados.

SGBD

Sistema de Gerenciamento de Bancos de Dados é um software com recursos específicos de forma a facilitar a manipulação das informações dos BD e o desenvolvimento de programas aplicativos. Como por exemplo Oracle, Ingres, Interbase ou MySQL.

Um SGBD não contém apenas os dados em si, mas armazena completamente toda a descrição dos dados, seus relacionamentos e formas de acesso.

Consulta

Em um SGBD é fornecida ao usuário somente uma representação conceitual dos dados, o que não inclui maiores detalhes sobre sua forma de armazenamento real. O chamado MER[9] é um tipo de abstração utilizada para fornecer esta representação conceitual.

Neste modelo, um esquema das tabelas, seus relacionamentos e suas chaves de acesso são exibidas ao usuário. Utilizando este modelo podemos então “navegar” pelos dados, e buscar as informações que nos interessa, para isso procedemos **pesquisas** (consultas) no SGBD.

8 Sistemas Gerenciados de Banco de Dados
9 Modelo de Entidade-Relacionamento

Redundância

Quando as aplicações estiverem realmente imunes a mudanças na estrutura de armazenamento ou na estratégia de acesso aos dados, podemos dizer que esta regra foi atingida. A redundância se faz com a eliminação da duplicidade das informações contidas no BD.

Inconsistência

Um SGBD deve permitir que cada usuário visualize os dados de forma diferente daquela existente previamente no BD. Uma visão consiste de um subconjunto de dados do BD, necessariamente derivados dos existentes no BD. A característica de inconsistência é fornecer dados ou permitir gravar dados inválidos, por exemplo: uma data de nascimento com o valor superior a data atual.

Integridade

Um SGBD deve gerenciar a integridade referencial definida em seu esquema, sem precisar em tempo algum, do auxílio do programa aplicativo. Desta forma, o DB deve ter ao menos uma instrução que permita a gravação de uma série modificações simultâneas e uma instrução capaz de cancelar um série modificações.

Regras de Nomenclatura

Ao trabalhar com SQL é comum utilizarmos *PascalCasing* nos casos de variáveis, nomes de objetos, colunas e outras definições. Já no caso de palavras chaves seque-se o padrão maiúsculo em todo o comando. Veja o exemplo de como esta regra é aplicada:

```
SELECT nome, sobrenome, idade
FROM clientes
WHERE cidade = 'Guarulhos'
```

Note que em SQL as strings são delimitadas por apóstrofo, ou aspas simples, e não por aspas duplas. Comandos SQL não utilizam como delimitador o ENTER e sim o espaço, ou seja, não existe marca de fim de comando, o processador de consultas separa cada palavra e consulta a tabela de comandos, o que for diferente da tabela de comandos é considerado parâmetro ou objeto.

Ou seja, objetos que utilizem espaço em branco ou tenham o mesmo nome que uma instrução precisam ser delimitados por colchetes. Veja o exemplo abaixo de uma tabela com espaços e com palavra reservada:

```
/*
Faz consulta na tabela de clientes
Trazendo os dados mais importantes
*/
SELECT name, lastname, [order] -- Dados principais
FROM [client national] -- Tabela de clientes
WHERE [state] = 'DF' -- Estado desejado
```

Foi necessário utilizar os colchetes nas palavras **order** e **state** por serem palavras reservadas, bem como no nome da tabela por conter espaços.

DICA: Evite utilizar as palavras reservadas e espaços em nomes de objetos criados. Comentários podem ser feitos em bloco ou em linha. Comentários em linha são definidos por "--" (dois traços seguidos) e a linha de comando precisa continuar após, como o exemplo anterior. Comentários em bloco são delimitados por "/*" e finalizados em "*/".

DER

Antes de falar sobre os conceitos, vamos conhecer a história do **Diagrama de Entidade e Relacionamento**. Tudo começou quando o Dr. Peter Chen em 1976 propôs uma forma de representar as Entidades e seus Relacionamento para os projetos de banco de dados. Isso forneceu uma nova e importante percepção dos conceitos de dados. "O DER proposto pelo Dr. Chen possibilitava ao projetista concentrar-se apenas na utilização dos dados sem se preocupar com estrutura lógica de tabelas" (DEVMEDIA, 2010). Por esse motivo, o DER é utilizado pelo projeto conceitual para modelar os conceitos de um BD de forma independente do SGDB.

Sendo assim, o **DER**[10] é um modelo conceitual no qual descrevemos o BD, utilizamos símbolos gráficos para representar os requisitos dos usuários.

- **Retângulos** – representam os conjuntos de entidades
- **Elipses** – representam os atributos
- **Losangos** – representam os conjuntos de relacionamentos
- **Linhas** – unem os atributos aos conjuntos de entidades e os conjuntos de entidades aos conjuntos de relacionamentos
- **Elipses duplas** – atributos multi valorados

Definições

Os comandos SQL são separados em três famílias:

1. **DDL – Data Definition Language** – Utilizada para criar, deletar e alterar os objetos como views, databases, stored procedures, entre outros.
2. **DCL – Data Control Language** – Permite controlar a segurança de dados definindo quem pode acessar cada operação em cada objeto.
3. **DML – Data Manipulation Language** – Comandos de manipulação das tabelas.

DDL	DCL	DML
CREATE ALTER DROP	GRANT REVOKE DENY	SELECT INSERT UPDATE

10 O software **brModelo** para realização do DER pode ser baixado gratuitamente no endereço eletrônico http://www.sis4.com/brmodelo/

		DELETE

Entidade

O primeiro conceito do DER é o conceito de entidade. Mas, o que é uma Entidade? É algo que possui uma existência distinta e separada, real ou imaginária. Ou seja, uma entidade é um objeto no mundo real que pode ser identificado de forma única em relação aos outros objetos.

Por exemplo, suponha que uma empresa pede que desenvolva um software para gerenciar seus funcionários. Durante a fase de entrevistas, perguntamos os desejos para o novo software (as regras de negócio). Uma possível resposta da empresa pode ser: informações sobre empregados e o departamento associado a cada um deles. Por tal resposta, podemos identificar algumas entidades no nosso DER, tais como:

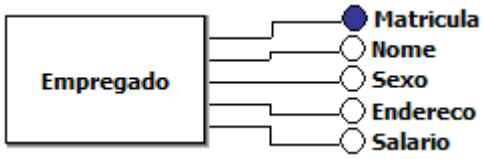
```
Empregado e Departamento
```

Conjunto de Entidades

Depois de identificar tais entidades, é necessário modelá-las no DER. Para tanto, devemos desenhar retângulos e escrever o nome da entidade no centro deles. Na figura seguinte, temos um exemplo de duas entidades: **Empregado** e **Departamento**.

Atributo

Note que, até o momento, as entidades não guardam informação alguma. Elas apenas representam um objeto existente no cenário da empresa. Para guardar informação, devemos definir os atributos para cada uma das entidades. Sendo assim, os atributos são propriedades particulares que descrevem cada entidade. São partes específicas de uma determinada entidade. São as informações que caracterizam a entidade.



Por exemplo, podem ser os atributos para a entidade **Empregado**: matrícula, nome, sexo, endereço e salário. Cada entidade possui valores específicos para seus atributos que pode diferir ou ser iguais aos valores dos atributos de outras entidades de um mesmo tipo de entidade.

Valor de um Atributo

Chamamos valor de um atributo ao conteúdo que um atributo pode ter. Fernando Anselmo, Maria Célia, Carlos Matheus poderiam ser os valores da entidade **Empregado**.

Domínio de um Atributo

É o conjunto de valores que um atributo pode assumir. Por exemplo, M ou F é considerado o domínio do atributo **sexo** da entidade **Empregado**.

Tipos de Atributos de uma Entidade

- **Único** – Cada entidade tem um valor diferente para este atributo. A matrícula de um **Empregado** é um atributo único porque não existe outro empregado com o mesmo número de matrícula.
- **Não-Único** – Quando o valor pode se repetir em várias entidades. Por exemplo, mais de um **Empregado** pode ter o mesmo **salário**.
- **Obrigatório** – Quando tem que existir um valor para este atributo em toda entidade. Por exemplo, todo empregado deve ter um **nome**.
- **Simples** – Quando possui um domínio simples. Por exemplo, o atributo **sexo** tem um domínio simples pois é formado pelo conjunto (único) das letras F e M.
- **Composto** – Quando possui mais de um domínio simples. Por exemplo, o **endereço** de um empregado é formado pelos domínios simples do Logradouro, do Bairro, da Cidade, do Estado e do CEP.
- **Uni valorado** – Quando tem um único valor para cada entidade. Por exemplo, cada empregado possui um único número de **matrícula**.
- **Multi valorado** – Quando pode ter mais de um valor para cada entidade. Por exemplo, um empregado pode ter mais de um telefone, do trabalho e da residência.
- **Derivado** – Quando o seu conteúdo depende do conteúdos de outros atributos. Por exemplo, o **salário** de um empregado depende do cargo que este empregado possui.
- **Não derivado** – Quando ele não pode ser obtido a partir de outros atributos. Por exemplo, **nome** de um empregado.
- **Identificador** – É o atributo ou atributos que identificam uma entidade de um tipo de entidade de maneira única. Por exemplo, a **matrícula** do empregado.
- **Não Identificador** – Quando o identificador não identifica por si só um entidade dentro de um tipo de entidades. Por exemplo, o **nome** não identifica o empregado dentro to tipo de entidade Empregado.

Observamos com os exemplos que cada atributo pode ser de vários tipos, por exemplo, o **matrícula** é um atributo único, obrigatório, uni valorado, não derivado e identificador.

Tipos de Dados

Os tipos de dados variam entre os diferentes RDBMS são padronizados pelo ANSI-92, mas alguns bancos podem utilizar nomenclaturas diferentes. Esses são os tipos mais comuns:

Tipo Comum	MS-SQL	Definição

binary varying	varbinary	Até 8.000 bytes, ocupação variável
character(n)	char(n)	Até 8.000 caracteres, ocupação fixa
character varying(n)	varchar(n)	Até 8.000 caracteres, ocupação variável
dec	decimal	-10^38 a 10^38, tamanho fixo com casa decimal fixa
float[(n)] de n = 1-7	real	-3.40E + 308 a 3.40E +308
float[(n)] de n = 8-15	float	-1.79E + 308 a 1.79E +308
integer	int	-2.147.483.648 a 2.147.483.648
blob	image	Até 2 GB de dados binários
clob	text	Até 2 GB de caracteres
national character(n)	nchar(n)	Até 4.000 caracteres, ocupação fixa (ocupa 2 bytes por caractere)
national character varying(n)	nvarchar(n)	Até 4.000 caracteres, ocupação variável (ocupa 2 bytes por caractere)
national text	ntext	Até 1 GB de caracteres (ocupa 2 bytes por caractere)
numeric	decimal	-10^38 a 10^38, tamanho fixo com casa decimal fixa

Tipos variáveis são melhores para textos pois ocupam apenas o espaço digitado, enquanto os tipos fixos ocupam espaço em branco quando não digitado. Alguns tipos de dados tem o tamanho definido na criação, como por exemplo, os tipos char, varchar e nvarchar.

Tipos Definidos pelo Usuário

Alem dos tipos principais temos também os tipos definidos pelo usuário que são baseados nos tipos do sistema. Por exemplo, o tipo CEP é uma variação de 9 caracteres, levando-se em conta o traço, como o exemplo abaixo:

```
sp_addtype 'CEP', 'char(9)'
```

É importante definir no inicio da criação os tipos a serem usados e os que podem ser criados para não ter variações em um mesmo dado, como por exemplo o CEP pode ser utilizado como 8 dígitos ou 8 caracteres. Criar o tipo CEP limita possíveis confusões.

Domínio

É o conjunto de valores permitidos para um atributo. Exemplo: Conjunto de valores do atributo Sexo do funcionário: M ou F; Conjunto de valores do atributo Nome aluno: 40 caracteres alfanuméricos Conjunto de valores do atributo salário: inteiro maior que 5000.

Relação

Representa a associação entre os elementos de duas entidades. Exemplo: João pertence ao Departamento de Banco de Dados, onde João é um elemento do conjunto de valores do atributo Nome do empregado da entidade Empregado.

Chaves

Chave primária, chave estrangeira e chave candidata são conceitos importantes na modelagem de dados, pois implementam restrições que garantirá ao futuro banco de dados a integridade dos dados.

Na análise de entidade e na identificação dos relacionamentos, teremos que definir as chaves que irão impor as restrições de integridade no banco de dados.

Chave primária (ou Primary Key)

Atributo ou combinação de atributos que possuem a propriedade de identificar de forma única uma linha da tabela. Corresponde a um atributo determinante. Cada tabela deve incluir um campo ou conjunto de campos que identifique de forma exclusiva, cada registro armazenado na tabela. Essas informações são chamadas de chave primária da tabela.

Desta forma, com a chave primária cria-se uma identificação única, e permite uma total segurança para que aplicações possam acessar, alterar, excluir dados sem correr o risco de apagar ou alterar dois campos da tabela ao mesmo tempo.

Chave estrangeira (ou Foreign Key)

Ocorre quando um atributo de uma relação for chave primária em outra relação. Por exemplo:

Tabela Produto

- **codigo_produto**
- produto
- categoria
- preco_data
- quantidade
- descricao

Tabela ItemDoPedido

- numero_pedido
- **codigo_produto**
- quantidade

Nessas tabelas temos dois casos de chaves primária e estrangeira. Observe que o **codigo_produto** consta nas duas tabelas. Em **Produto** é o campo identificador, ou seja, cada produto será exclusivo, portanto, é uma chave primária. Já em **ItemDoPedido** o campo pode constar várias vezes e aqui será uma chave estrangeira.

Portanto as tabelas ficarão assim:

Tabela Produtos

- codigo_produto (chave primária)
- produto
- categoria
- preco_data
- quantidade
- descricao

Tabela ItensDoPedido

- numero_pedido (chave primária)
- codigo_produto (chave estrangeira)
- quantidade

Chaves Candidatas

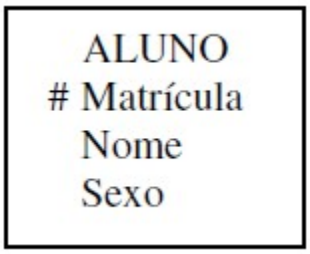
Ocorrem antes da definição de uma relação que existe mais de uma combinação de atributos possuindo a propriedade de identificação única. Por exemplo, Matrícula, CPF, RG, Número do Titulo de Eleitor

Entidades Fortes/Fracas

Como já vimos, chamamos de entidade, qualquer coisa real ou abstrata, de um determinado ambiente, sobre a qual precisamos guardar informações.

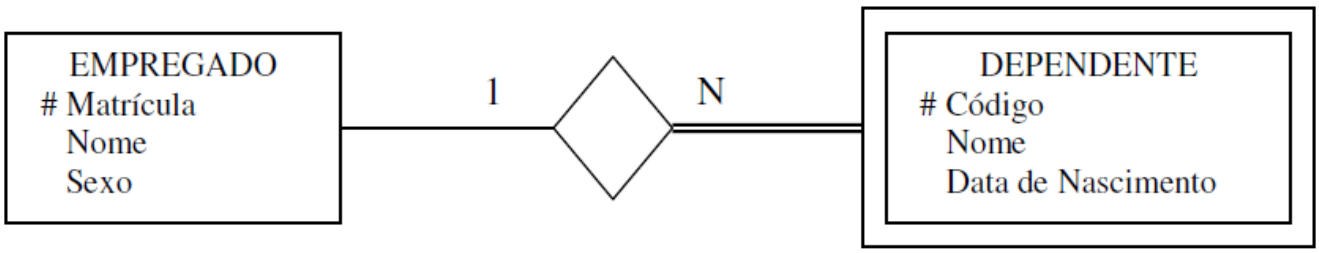
Entidade Primária (ou Forte)

Existe por si mesma. Sua identificação completa é feita pelos seus próprios atributos.



Entidade Dependente (ou Fraca)

Sua identificação não pode ser realizada unicamente por seus próprios atributos. Para sua identificação completa precisamos de atributos de outra entidade.



Vantagens e Desvantagens da Linguagem SQL

Com o que vimos até o presente momento, podemos apontar as seguintes vantagens no uso da linguagem SQL:

- Independência de fabricante
- A linguagem SQL é adotada por praticamente todos os SGBD relacionais existentes no mercado, além de ser uma linguagem padronizada (ANSI). Com isso, pelo menos em tese, posso mudar de SGBD sem me preocupar em alterar os programas de aplicação.
- Portabilidade entre plataformas de hardware e software
- Pode ser utilizada tanto em máquinas Intel rodando Windows, passando por workstations RISC rodando UNIX, até mainframes rodando sistemas operacionais

proprietários.

- Redução dos custos com treinamento, as aplicações podem se movimentar de um ambiente para o outro sem que seja necessária uma reciclagem da equipe de desenvolvimento.
- Usa inglês estruturado de alto nível
- O SQL é formado por um conjunto bem simples de sentenças em inglês, oferecendo um rápido e fácil entendimento.
- Permite consultas interativas
- Permite aos usuários acesso fácil e rápido aos dados a partir de um front end que permita a edição e a submissão de comandos SQL.

Porém, existem também algumas desvantagens no uso da linguagem SQL:

- Falta de ortogonalidade nas expressões, funções embutidas, variáveis indicadoras, referência a dados correntes, constante NULL e conjuntos vazios.
- Definição formal da linguagem após sua criação.
- Discordância com as linguagens hospedeiras (geralmente procedurais e orientadas para registros e não para conjuntos).
- Falta de algumas funções.
- Não dá suporte a alguns aspectos do modelo relacional (*join* explícitos, domínios)

2. Modelo Relacional

Definições

O **Modelo Relacional** foi introduzido por Edgar Frank Codd (1970) e tornou-se um padrão para aplicações comerciais, devido a sua simplicidade e desempenho. É um modelo formal, bastante representativo e ao mesmo tempo bastante simples, foi o primeiro modelo de dados descrito teoricamente.

Um dos SGBD precursores que implementaram esse modelo foi o System R (IBM), cuja sua história foi vista na nossa primeira aula, baseado em seus conceitos surgiram: DB2 (IBM), SQL-DS (IBM), Oracle, Informix, Ingres, Sybase, entre outros.

O Modelo Relacional representa os dados num Banco de Dados como uma coleção de relações e seus relacionamentos. Cada relação contém um nome e um conjunto de atributos com seus respectivos nomes. Informalmente, as relações do Modelo Relacional são também chamadas de tabelas pela maioria dos desenvolvedores.

Relação

É a forma como os objetos que compõem a realidade se relacionam. É o tipo de ocorrência existente entre entidades.

O relacionamento entre entidades em um diagrama ER é representado através de um losango que liga as entidades relacionadas. A figura acima ilustra um exemplo de relacionamento entre as entidades Empregado e Departamento.

Note que para identificar como cada entidade se relaciona com as demais é importante realizar algumas perguntas durante a fase de levantamento de requisitos. Por exemplo, para identificar os relacionamentos no nosso caso hipotético da empresa XPT você poderia perguntar: Como um empregado está relacionado com um departamento? Um empregado pode trabalhar em apenas um departamento? Um departamento possui mais de um empregado? Dependendo do tipo de resposta fornecida, um relacionamento poderá ser definido de três formas:

- um-para-um
- um-para-muitos
- muitos-para-muitos

Agora, vamos conhecer um pouco mais sobre cada forma de relacionamento.

Relacionamento um-para-um

O relacionamento um-para-um é usado quando uma entidade A se relaciona com apenas uma entidade B e vice-versa.

Representado pelo sinal: 1:1

No exemplo da Figura 8, temos o relacionamento Empregado (entidade) e Departamento (entidade). Esse é um relacionamento um-para-um porque a entidade Empregado gerencia um Departamento e um Departamento é gerenciado por apenas um Empregado. Você deve observar que o fato do Departamento ser gerenciado por apenas um Empregado é uma restrição da empresa que você realizou a entrevista na fase de levantamento de requisitos. Outras empresas podem ter vários empregados gerenciando um departamento. O importante nesse momento é entender que os tipos de relacionamentos determinam como as entidades se relacionam. Cada tipo de relacionamento define uma restrição diferente para as entidades envolvidas.

Relacionamento um-para-muitos

O relacionamento um-para-muitos é usado quando uma entidade A pode se relacionar com uma ou mais entidades B.

Representado pelo sinal: 1:N

No exemplo da Figura 9, temos o relacionamento Empregado (entidade) e Dependentes (entidade). Esse é um relacionamento um-para-muitos porque a entidade Empregado possui vários Dependentes. Por outro lado, cada Dependente possui apenas um Empregado.

Relacionamento muitos-para-muitos

O relacionamento muitos-para-muitos é usado quando várias entidades A se relacionam com várias entidades B.

Representado pelo sinal: N:N ou N:M

No exemplo da figura acima, temos o relacionamento Empregado (entidade) e Projeto (entidade), é um relacionamento muitos-para-muitos porque a entidade Empregado trabalha em vários (M) Projetos. Por outro lado, cada projeto possui (N) empregados.

Tupla

As tuplas representam os valores de uma tabela. A figura a seguir mostra a tabela Empregado preenchida com valores hipotéticos. Note que as colunas da tabela representam os atributos, enquanto as linhas representam as tuplas. Se uma tabela não tiver tuplas, ela estará vazia, ou seja, sem dados. Desse modo, quando efetuarmos uma busca no Google, recebemos como resposta as tuplas do banco de dados do Google que estão relacionadas de alguma forma com o texto procurado. Informalmente, as tuplas são também chamadas de registros pelos desenvolvedores.

Matricula	Nome	Sexo	Endereco	Telefone
1	Nelio	M	Rua das Na...	8888-1555
2	Jose	M	Rua das ca...	8888-9999
3	Maria	F	Rua das Ala...	9999-7444

Depois de aprender o significado dos conceitos Tabela, Atributo, Domínio e Tuplas, é importante agora aprender como utilizar uma ferramenta para definir tais conceitos no Modelo Relacional.

Atributo

Atributos são todas as informações que existem em uma tabela. Essas informações são chamadas informalmente de campos. **Exemplo**: Nome, CPF, Rua, Bairro, Telefones, CEP, Data de Nascimento etc.

Nulo

Quando é permitido que um atributo não possua um valor determinado, este valor é considerado **nulo**. Lembre-se que o termo **nulo** é diferente do termo **vazio**, sendo que este primeiro deriva da inexistência de um valor e o segundo é aplicado exclusivamente a campos do **tipo texto**.

Domínio

Todo atributo para armazenar as informações de uma tabela deve ter um domínio definido. O domínio representa todos os valores possíveis que um atributo pode receber. Por exemplo, o atributo Telefone pode receber um conjunto de número com oito dígitos. Por outro lado, o atributo Nome pode receber um conjunto de cadeias de caracteres que representa o nome de uma pessoa.

Devemos escolher quais serão os tipos de dados de cada atributo/coluna. Estes tipos de dados devem levar em conta o tipo de SGBD escolhido[11].

11 Os seguintes exemplos são referentes do Microsoft SQL Server 2000

Números exatos

- **Inteiros**: números inteiros da forma 0, 1, 1000, 10.000.000, -1, -300, etc. Estes podem ser divididos em inteiros pequenos, médios, grandes, cada um ocupando um determinado espaço de armazenamento. Exemplos:
  - **bigint**: de -2^63 (-9.223.372.036.854.775.808) até 2^63-1 (9.223.372.036.854.775.807). Ocupa 8 bytes;
  - **int**: de -2^31 (-2.147.483.648) até 2^31 - 1 (2.147.483.647). Ocupa 4 bytes;
  - **smallint**: de -2^15 (-32.768) até 2^15 - 1 (32.767). Ocupa 2 bytes;
  - **tinyint**: de 0 até 255. Ocupa 1 byte;
- **bit**: inteiro que pode valer 0 ou 1. Cada conjunto de 8 campos deste ocupa 1 byte, de 16 ocupa 2 bytes e assim em diante;
- **decimal e numeric**: podemos definir o máximo de dígitos e o máximo de dígitos após a vírgula (decimais). Varia de -10^38 +1 até 10^38 –1;

Dados monetários

- **money**: de -2^63 (-922.337.203.685.477,5808) até 2^63 - 1 (+922.337.203.685.477,5807), com precisão de 10.000 valores decimais. Ocupa 8 bytes;
- **smallmoney**: de -214.748,3648 through +214.748,3647, com precisão de 10.000 valores decimais. Ocupa 4 bytes;

Números aproximados

- **float**: dados de ponto flutuante de -2,23E - 308 até 2,23E + 308, onde define-se o expoente. Pode ocupar 4 ou 8 bytes;
- **real**: dados de ponto flutuante de -3.40E + 38 até 3.40E + 38, onde defini-se o expoente. Pode ocupar 4 ou 8 bytes;

Datas

- **datetime**: data de 01/01/1753 até 31/12/9999, com precisão de milésimos de segundo. Ocupa 8 bytes;
- **smalldatetime**: data de 01/01/1900 até 06/06/2079, com precisão de minutos. Ocupa 4 bytes;

Texto

- **char**: tamanho fixo de uma cadeia de caracteres, com máximo de 8000 caracteres. Ocupa n bytes onde n é o tamanho da cadeia. Exemplo: se armazenamos o texto "João", este ocupará n bytes;
- **varchar**: tamanho variável de uma cadeia de caracteres, com limite de 8000. Ocupa o tamanho em bytes do registro. Exemplo: se armazenamos o texto "João", este

ocupará 4 bytes;

- **text**: tamanho variável de uma cadeia de caracteres, limitado a 2^31 - 1 (2.147.483.647).

Texto Unicode (cadeias de texto UNICODE)

- **nchar**: tamanho fixo de uma cadeia de caracteres UNICODE, com máximo de 4000 caracteres. Ocupa n bytes onde n é o tamanho da cadeia. Exemplo: se armazenamos o texto "João", este ocupará n bytes;
- **nvarchar**: tamanho variável de uma cadeia de caracteres UNICODE, com limite de 4000. Ocupa o tamanho em bytes do registro. Exemplo: se armazenamos o texto "João", este ocupará 4 bytes;
- **ntext**: tamanho variável de uma cadeia de caracteres UNICODE, limitado a 2^30 - 1 (1.073.741.823).

Strings Binárias

- **binary**: dados binários de tamanho fixo, com máximo de 8.000 bytes;
- **varbinary**: dados binários de tamanho variável, com máximo de 8.000 bytes;
- **image**: dados binários de tamanho variável, com máximo de 2^31 - 1 (2.147.483.647) bytes;

Outros tipos

- **cursor**: referência de um cursor;
- **sql_variant**: armazena vários tipos de dados suportados pelo SQL Server, exceto text, ntext, timestamp, e sql_variant. Máximo de 8.016 bytes;
- **table**: tipo especial de dados que armazena o resultado de uma tabela, para processamento posterior;
- **timestamp**: tipo de dados que gera números binários automaticamente, únicos em um banco de dados. Ocupa 8 bytes;
- **uniqueidentifier**: um identificador único global (globally unique identifier – GUID).

Desse modo, o domínio de um atributo define qual o tipo de dado e o formato que o dado pode ser armazenado por aquele atributo. Por exemplo, o formato do atributo Data de Nascimento é “dd/mm/aaaa”. O formato do atributo CEP é “nnnnn-nnn”. Assim, o formato descreve como o dado será exibido para o usuário do sistema.

Modelo Relacional

O primeiro conceito do MER, ou **Modelo Entidade-Relacionamento** é o conceito de entidade. Mas, o que é uma Entidade? É algo que possui existência distinta e separada, real ou imaginária. Ou seja, uma entidade é um objeto no mundo real que pode ser identificado de forma única em relação aos outros objetos.

O Modelo Relacional representa as entidades dispostas em num BD, seus atributos e seus relacionamentos. Cada relação contém um nome e um conjunto de atributos com seus respectivos nomes.

Cardinalidade

A cardinalidade é um conceito importante para ajudar a definir o relacionamento, ela define o número de ocorrências em um relacionamento.

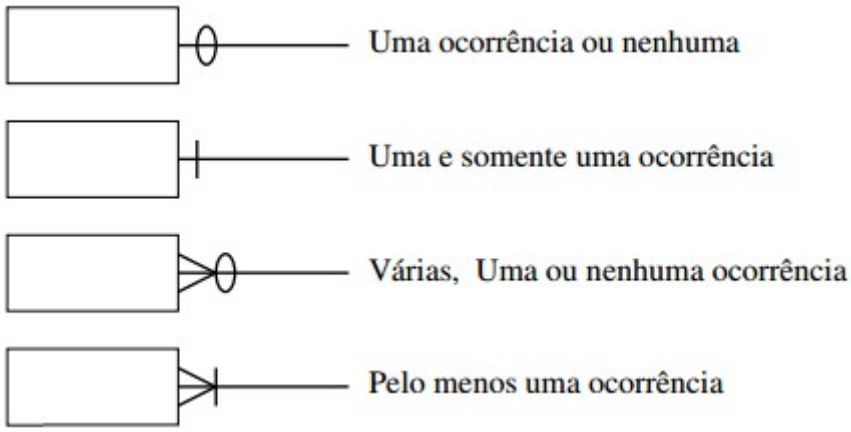
Para determinarmos a cardinalidade, deve-se fazer algumas perguntas relativa ao relacionamento em ambas às direções. Por exemplo, dado um relacionamento entre Departamento e Empregado, pode-se fazer as seguintes perguntas:

```
Pergunta: Um departamento possui quantos empregados?
Resposta: No mínimo 1 e no máximo N
Pergunta: Um empregado está alocado em quantos departamentos?
Resposta: No mínimo em 1 e no máximo em 1
```

De acordo com as respostas acima, temos que a cardinalidade expressa no relacionamento conforme a seguinte figura.

Grado

Ou Grau, são as restrições identificadas na quantidade de ocorrências de uma entidade, que pode estar relacionada a uma ocorrência de outra entidade. São representados graficamente por símbolos especiais colocados nas extremidades da linha que representa um relacionamento



Esquema

Em qualquer modelo de dados é importante distinguir entre a descrição da base de dados e a base de dados propriamente dita. A descrição de uma base de dados é chamada **Esquema do BD**. É especificado durante o projeto da base de dados, sendo que a expectativa de mudanças não é grande. A forma de visualização de um esquema é chamado de Diagrama do Esquema. Muitos modelos de dados têm certas convenções para, diagramaticamente, mostrar esquemas especificados no modelo.

Instância

Já os dados atualmente existentes em uma base de dados podem mudar com relativa frequência. Os dados da base de dados em um particular momento do tempo são chamados **Instâncias do BD** (ou Ocorrências ou Estados).

O **Esquema** é algumas vezes chamado de **Base Intencional** e uma instância é chamada de **Base Extensional do Esquema**.

Chave

São utilizadas para identificar um registro específico, geralmente feito através de um valor inteiro único. Um número artificialmente gerado é a melhor forma de garantir a unicidade da chave, pois poucas informações reais possuem essa propriedade.

Inter-relação

Dizemos que o Modelo possui as estruturas efetivamente inter-relacionadas, permitindo que o Processo de Negócio seja gerenciado de maneira integrada, e isso envolve diversas funções de diversas áreas como um processo único, que é visualizado por todos os envolvidos “de ponta a ponta”.

Os Processos de Negócio são considerados de maneira explícita e inteira. Uma organização com visão horizontal tem consciência dos Processos de Negócio que utiliza e que estão embutidos nas suas rotinas.

Álgebra Relacional

A álgebra relacional é uma linguagem de consulta procedural. Ela consiste em um conjunto de operações que tomam uma ou duas tabelas como entrada e produzem uma nova tabela como resultado. Essas operações baseiam-se na teoria dos conjuntos[12].

Seleção

Pode ser entendida como a operação que filtra, seleciona as linhas de uma tabela, realizando também uma projeção, e opera em um conjunto de dados, sendo portando uma operação unária.

É utilizada para selecionar um subconjunto de tuplas numa relação que satisfaça uma determinada condição.

12 As tabelas correspondem aos conjuntos

Projeção

Pode ser entendida como uma operação para filtrar as colunas de uma tabela de nosso BD ou uma tabela resultante de uma outra operação relacional executada. Por operar em apenas um conjunto de entrada, a projeção é classificada como uma operação unária.

A operação de projeção é utilizada para selecionar determinadas colunas de uma relação. A operação é executada em apenas uma relação e o resultado é uma nova relação contendo apenas os atributos selecionados, eliminando-se as duplicidades.

Produto Cartesiano

Utiliza a mesma notação de operação matemática de dois conjuntos, tendo como resultado do produto cartesiano de duas tabelas uma terceira tabela contendo as cominações possíveis entre os elementos das tabelas originais.

Essa tabela resultante possui um número de colunas que é igual à soma do número de colunas das tabelas iniciais e um número de linhas igual ao produto do número de linhas das duas tabelas

Composição ou Junção

Essa operação interage com o modelo relacional, ou seja trabalha com o modelo de relações entre tabelas realizando um produto cartesiano, combinando as linhas e somando as colunas de duas tabelas, só que partindo de campos comuns de ambas para realizar essa “seleção relacional”.

União

É uma operação binária, ou seja cria uma tabela a partir de duas outras tabelas união compatíveis levando as linhas comuns e não comuns a ambas. As informações duplicadas aparecerão somente uma vez no resultado.

Uma característica é que somente é possível utilizar este operador caso as tabelas de origem possuam compatibilidade de união, ou seja, as tabelas devem ser equivalentes e gerem o mesmo tipo de resultado.

Intersecção

É uma operação binária, ou seja, cria uma tabela a partir de duas outras tabelas levando sem repetição as linhas, que pertençam a ambas as tabelas presentes na operação. A relação criada pela operação de intersecção é o resultado de todas as tuplas que pertençam a ambas as relações presentes na operação.

Diferença

Essa operação permite encontrarmos linhas que estão em uma tabela mas não estão em outra. É uma operação primitiva que requer duas tabelas compatíveis, ou seja, estruturalmente idênticas. O resultado é uma tabela que possui todas as linhas que existem

na primeira tabela e não existem na segunda.

Divisão

Essa operação produz como resultado a projeção de todos os elementos da primeira tabela que se relacionam com todos os elementos da segunda tabela. Essa operação também pode ser obtida através de outras operações de álgebra relacional.

Integridade

O objetivo primordial de um SGBD é o de garantir a integridade de dados, para garantir a integridade de um banco de dados, os SGBD oferecem o mecanismo de restrições de integridade. Uma restrição de integridade é uma regra de consistência de dados que é garantida pelo próprio SGBD.

Algumas dessas restrições podem ser fornecidas pelo SGBD e automaticamente executadas. Outras restrições podem ser verificadas pelos programas de atualização ou durante a entrada de dados.

Restrições

As restrições de Integridade podem ser:

- Restrição de Domínio
- Restrição de Chave
- Restrição de Integridade de Entidade
- Restrição de Integridade Referencial
- Restrição de Integridade Semântica

Esse último tipo de restrições são chamadas de restrições semânticas (ou regras do negócio). São exemplo de restrições semânticas: Um empregado do departamento denominado “Finanças” não pode ter a categoria funcional “Engenheiro”. Um empregado não pode ter um salário maior que seu superior imediato.

Integridade Referencial

A chave primária e a integridade da relação são especificadas individualmente em cada relação. A integridade referencial é uma restrição estabelecida entre duas relações com o objetivo de manter a consistência dos dados entre as tuplas dessas relações.

De um modo simples, pode-se dizer que a integridade referencial determina que uma linha numa relação A que está referida a uma outra relação B tem que obrigatoriamente estar ligada a uma linha existente na relação B.

A associação entre as duas relações faz-se através da **Chave Estrangeira**, que é um atributo, ou grupo de atributos, que desempenham o papel de chave primária noutra relação,

servindo portanto de elo de ligação entre elas. Dependendo das situações é possível que uma chave estrangeira aceite valores nulos, o que significa que essa linha não está referida à chave da outra relação.

Data/Hora

Podem ser os tipos:

- **date**: data de calendário com ano, mês e dia. Exemplo: ‘2001-04-25’
- **time**: hora do dia em horas, minutos e segundos. Exemplos: ’11:07:14’, ’11:07:14.125’
- **timestamp** – data e hora. Exemplo: ‘2008-10-20 11:07:14.125’

Binários Longos

Tipos de dados para objetos grandes. Por exemplo fotos, vídeos, arquivos.

- **clob** – para caracteres (**text** para o banco Postgres)
- **blob** – para objetos binários interpretados por outra aplicação

Quando o resultado é um objeto grande, a consulta (query) devolve um apontador ou exporta os dados.

Enumerações

Conhecido pelo tipo ENUM. Um ENUM é um nome do domínio está firmemente acoplado. O processo de verificação ou não o registando de um ENUM nome do domínio é idêntico para o cessionário dos correspondentes é comumente chamado de "validação".

3. Normalização

Normalização é uma técnica de projeto amplamente utilizada no desenho de bancos de dados relacionais. É um processo sistemático através do qual uma tabela relacional não normalizada é transformada em um conjunto de tabelas normalizadas, que representem da melhor forma possível uma realidade a ser modelada.

Um conceito básico usado para a normalização é o conceito de dependência funcional. O processo de Normalização passa pelas seguintes etapas:

- O documento ou arquivo a ser normalizado é representado na forma de uma tabela não normalizada
- A tabela vai sendo decomposta em tabelas normalizadas.

A normalização dá-se em três passos principais, passando normalmente pelas formas normais ou FN, sendo que uma FN é um conjunto de regras que uma tabela deve obedecer que destinam-se a eliminar as redundâncias de dados.

1ª Forma Normal

Uma relação se encontra na 1FN se todos os domínios de atributos possuem apenas valores atômicos (simples e indivisíveis), e que os valores de cada atributo na tupla seja um valor simples. Assim sendo todos os atributos compostos devem ser divididos em atributos atômicos.

Características da 1FN:

- Eliminar as colunas duplicadas de uma mesma tabela
- Criar tabelas separadas para cada grupo de dados relacionados e identificar cada linha com uma ou mais colunas como únicas (IDs, Códigos, etc). Ou seja, todos os atributos da relação estiverem baseados em um domínio simples, não contendo grupos ou valores repetidos

Passagem à 1FN

Gerar uma única tabela com colunas simples, a chave primária: id de cada tabela aninhada. Por exemplo:

```
Fornecedor(codigo, status_cidade, cidade, codigo_peca, quantidade)
```

A tabela ao lado anteriormente possuía valores multi valorados para o campo CODIGO_PECA (P1,P2,P3,P4,P5) e o campo QUANTIDADE (300,200,400,200,100). Resultando nas seguintes tabelas:

```
Fornecedor(codigo, status_cidade, cidade)
CidadePeca(codigo_fornecedor, codigo_peca, quantidade)
```

2ª Forma Normal

Uma relação se encontra na segunda forma normal quando estiver na primeira forma normal e todos os atributos que não participam da chave primária são dependentes desta.

Assim devemos verificar se todos os atributos são dependentes da chave primária e retirar-se da relação todos os atributos de um grupo não dependente que dará origem a uma nova relação, que conterá esse atributo como não chave. Desta maneira, na segunda forma normal evita inconsistências devido a duplicidades.

Características da 2FN:

- Estar na 1FN
- Cada atributo não-chave for dependente da chave primária inteira, isto é, cada atributo não-chave não poderá ser dependente de apenas parte da chave. Ou seja, todos os seus atributos que não façam parte de alguma chave candidata devem ser determinados unicamente por qualquer chave candidata da tabela.

As informações do campo CIDADE e STATUS_CIDADE não são totalmente dependentes da chave primária total, por isso foi criada a tabela FORNECEDOR para guardar esses dados, tendo como chave primária o CODIGO.

```
Fornecedor(codigo, status_cidade, cidade)
```

Note que ainda há o problema de relacionar o Status com a Cidade, pois deve existir obrigatoriamente um fornecedor nela.

3ª Forma Normal

Uma relação estará na 3FN, quando estiver na 2FN e todos os atributos que não participam da chave primária são dependentes desta porém não transitivos. Assim devemos verificar se existe um atributo que não depende diretamente da chave, retirá-lo criando uma nova relação que conterá esse grupo de atributos, e defina com a chave, os atributos dos quais esse grupo depende diretamente.

Características da 3FN:

- Estar na 2FN
- Eliminar as colunas que não possuem dependência funcional com as chaves primárias. todos os atributos que não são chave sejam mutuamente independentes, isto é, que não existam funções que definam um ao outro. Portanto, sempre a chave por inteiro deve definir toda a tabela.

Foi criada a tabela CIDADE com os campos NOME_CIDADE como chave primária. Desta forma, a tabela FORNECEDOR só precisa guardar o Código Fornecedor (chave) e o valor da Cidade, que se relaciona com esta tabela.

```
Fornecedor(codigo, status_cidade)
```

Um esquema de dados nessa situação pode facilmente lidar com o projeto de um banco de

dados para uma empresa inteira.

Terceira Forma Normal de Boyce-Codd (3.5 FNBC)

Esta FN é chamada de 3.5, pois Boyce e Codd adicionaram mais um requisito a 3FN:

- Estar na 3FN (comum)
- Não existir dependência funcional dentro da chave primária

4ª Forma Normal

Na grande maioria dos casos, as entidades normalizadas até a 3FN são fáceis de entender, atualizar e de se recuperar dados. Mas às vezes podem surgir problemas com relação a algum atributo não chave, que recebe valores múltiplos para um mesmo valor de chave. Esta nova dependência recebe o nome de dependência multi valorada que existe somente se a entidade contiver no mínimo três atributos.

Características da 4FN:

- Estar na 3FN
- Não existir dependências multi valoradas

Uma entidade que esteja na 3FN também estará na 4FN, se ela não contiver mais do que um fato multi valorado a respeito da entidade descrita, em um relacionamento muitos-para-muitos, entidades independentes não podem ser armazenadas na mesma tabela. A 4FN lida com este problema, criando tabelas de relacionamentos apenas com chaves primárias e estrangeiras, removendo entradas duplicadas em outras tabelas.

Por exemplo, considere a seguinte tabela:

```
FuncionarioHabilidadeIdioma(matricula, habilidade, idioma)
```

Esta poderá ser transformada em duas outras tabelas:

```
FuncionarioHabilidade(matricula, habilidade)
FuncionarioIdioma(matricula, idioma)
```

5ª Forma Normal

A 5FN afirma que deve ser possível reconstruir a tabela original a partir das tabelas em que ela foi dividida. O benefício de aplicar esta regra é assegurar que nenhuma coluna estranha ao contexto foi criada e que todas as tabelas criadas são tão grandes quanto necessariamente precisam ser.

Características da 5FN:

- Estar na 4FN
- Quando o conteúdo deste mesmo registro não puder ser reconstruído (junção) a partir de outros registros menores, extraídos deste registro principal.

É uma boa prática aplicar estas regras, mas, a menos que esteja trabalhando num esquema de dados muito grande, provavelmente não precisará delas. Um registro não pode ser reconstruído a partir de vários tipos de registros menores. Exemplo:

- Os vendedores representam as empresas.
- As empresas fazem produtos.
- Os vendedores vendem os produtos.

Exemplo de um caso mais geral (permite qualquer combinação):

```
Relacional(vendedor, empresa, produto)
```

Podemos reconstruir essa tabela relacional do seguinte modo:

```
VendedorEmpresa(vendedor, empresa)
EmpresaProduto(empresa, produto)
VendedorProduto(vendedor, produto)
```

4. Comandos de Manutenção e Consulta

A manutenção do banco de dados é feita em dois grupos claros: Objetos do BD e Tuplas das Entidades. Usaremos como exemplo um clube de tratamento da forma física, selecionado, conceituado e imaginário chamado **Visual Spa**

A tabela HOSPEDE contém as informações sobre os sócios (tabela 1):

NOME	GENERO	BIOTIPO	ALTURA
MIGUEL	M	M	1,67
JOSIEL	M	M	1,72
RAQUEL	F	G	1,65
LUCIANA	F	G	1,80
JOANA	F	M	1,65
EMANUEL	M	M	1,78

Tabela 1

E a tabela QUARTO a seguir é um exemplo da lista de todos os hóspedes que chegaram ou saíram no período entre 01 e 15/01/2010.

NOME	QUARTO	CHEGADA	SAIDA	DESCONTO
MIGUEL	4	01-01-2010	08-01-2010	0,20
JOSIEL	2	01-01-2010	15-01-2010	0,10
RAQUEL	1	01-01-2010	15-01-2010	
LUCIANA	3	01-01-2010	08-01-2010	0,10
JOANA	5	01-01-2010	15-01-2010	
EMANUEL	6	01-01-2010	15-01-2010	0,12
MIGUEL	3	09-01-2010	15-01-2010	
LUCIANA	4	09-01-2010	15-01-2010	

Tabela 2

Base de Dados: CREATE, ALTER e DROP

Use a instrução **CREATE** para definir um novo objeto (tabela, índice, Banco de Dados ou outros), seus campos e restrições de seus campo.

***Sintaxe**: CREATE [tipo] <nome>*

A seguir o código para a criação do Banco de Dados:

```
CREATE DATABASE visualspa
```

A seguir o código necessário a criação da tabela HOSPEDE e QUARTO:

```
CREATE TABLE hospede (
 nome VARCHAR(25) NOT NULL,
 genero VARCHAR(1),
 biotipo VARCHAR(1),
 altura NUMERIC(5,2),
 PRIMARY KEY (nome)
```

```
)
CREATE TABLE quarto (
 nome VARCHAR(25) NOT NULL,
 quarto INT NOT NULL,
 chegada DATE NOT NULL,
 saida DATE,
 desconto NUMERIC(5,2),
 PRIMARY KEY (nome, quarto)
)
```

Utilizamos a instrução **ALTER** para modificar a estrutura de um objeto depois de ter sido criado, ou seja, este comando permite modificar objetos no banco de dados.

***Sintaxe**: ALTER <tipo> <nome> (ADD/DROP nome_atributo1 <tipo> [NOT NULL], nome_atributoN <tipo> [NOT NULL])*

A seguir o código para a modificação da tabela QUARTO, adicionando a chave de ligação entre esta tabela e HOSPEDE:

```
ALTER TABLE quarto ADD CONSTRAINT fkQuarto FOREIGN KEY (nome) REFERENCES hospede
(nome) ON UPDATE CASCADE ON DELETE CASCADE
```

Use a instrução **DROP** para remover um objeto depois de criado, ou seja, este comando permite a eliminação de objetos no banco de dados.

***Sintaxe**: DROP <tipo> <nome>*

A seguir o código para a eliminação da tabela QUARTO (não o execute pois necessitaremos dessas entidades para os exercícios posteriores):

```
DROP TABLE quarto
```

Registros: INSERT, UPDATE e DELETE

A inserção dos registros nas entidades é realizada com o comando **INSERT**, que possui a seguinte sintaxe básica:

***Sintaxe**:* INSERT INTO [tabela] (lista de colunas) VALUES (lista de valores)

A lista de colunas logo após o nome da tabela é opcional, se forem informadas as colunas chamamos de insert referencial e se não forem informadas as colunas de insert posicional. A vantagem do modelo referencial sobre o modelo posicional, é que neste ultimo todas as colunas da tabela precisam constar no values para encaixar com as colunas, alem de correrrmos o risco de alterarem a lista de colunas e os dados ficarem invertidos.

Os seguintes códigos SQL adicionam um novo hospede:

```
INSERT INTO hospede (nome, genero, biotipo, altura)
      VALUES ('MIGUEL', 1, 1, 1.67)
```

Adicione, conforme as tabelas 1 e 2 vistas no tópico anterior os registros de HOSPEDE e QUARTO. A modificação dos registros nas entidades é realizada com o comando **UPDATE**, que possui a seguinte sintaxe básica:

```
UPDATE [tabela] SET [campo1 = valor1] [, campoN = valorN] WHERE [condição]
```

Apesar da cláusula WHERE constar no UPDATE, é opcional. Entretanto, sem essa opção o comando modificará todos os registros. O exemplo abaixo é o recomendado, pois inclui a definição para um filtro de modificação:

```
UPDATE hospede SET biotipo = 2 WHERE nome = 'MIGUEL'
```

A exclusão dos registros nas entidades é realizada com o comando **DELETE**, que possui a seguinte sintaxe básica:

```
DELETE FROM [tabela] WHERE [condição]
```

Apesar da cláusula WHERE constar no comando DELETE, é opcional. Entretanto, sem essa opção o comando eliminará todos os registros. O exemplo abaixo é o recomendado, pois inclui a definição para um filtro de exclusão:

```
DELETE FROM hospede WHERE nome = 'MIGUEL'
```

Consultas

A consulta dos registros nas entidades é realizada com o comando **SELECT**, que possui a seguinte sintaxe básica:

Sintaxe: SELECT <lista_de_colunas> FROM <nome_tabela>

Por exemplo, podemos utilizar o seguinte comando para mostrar todos os hóspedes cadastrados:

```
SELECT * FROM hospede
```

O exemplo utiliza o coringa `*` para selecionar as colunas na ordem em que foram criadas. A instrução SELECT, como pudemos observar seleciona um grupo de registros de uma (ou mais) tabela(s). No caso a instrução FROM nos indica a necessidade de pesquisarmos tais dados apenas na tabela HOSPEDE.

A cláusula **DISTINCT** elimina duplicidades, significando que somente relações distintas serão apresentadas como resultado de uma pesquisa.

```
SELECT DISTINCT nome FROM hospede
```

A cláusula **ORDER BY** modificará a ordem de apresentação do resultado da pesquisa ascendente usando o termo “asc” (por padrão) ou descendente usando o termo “desc”.

```
SELECT nome, genero FROM hospede ORDER BY nome
```

Operadores Aritméticos

São quatro os operadores aritméticos que podem formar expressões:

+ adição

- subtração

* multiplicação

/ divisão

Na consulta a seguir, usamos o operador de adição para adicionar 0.05 ao valor corrente de DESCONTO a todos os hospedes que já possuem descontos. A cláusula SELECT contém duas expressões: NOME e a expressão formada pelo operador aritmético, DESCONTO + 0.05 mostrando um novo nome para esta coluna (novo_desconto). A tabela resultado deverá conter duas colunas, uma para cada expressão:

```
SELECT nome, desconto + 0.05 novo_desconto FROM quarto WHERE desconto IS NOT NULL
```

A consulta abaixo pode ser usada para determinar por quantos dias os hóspedes ficaram no **Spa**. Este número é obtido subtraindo-se a data de chegada de saída.

```
SELECT nome, saida - chegada tempo FROM quarto
```

Funções Agregadas

A característica marcante das funções agregadas é que produzem um único valor a partir de uma coluna inteira de dados. Portanto, enquanto qualquer outro tipo de expressão retorna um valor para cada linha, as funções agregadas retornam um valor que representa um agregado dos valores referentes às várias linhas. Por esta razão, são também chamadas de funções de colunas.

Existem cinco funções agregadas. São elas:

- AVG(argumento) – Retorna a média dos valores do argumento
- MAX(argumento) – Retorna o maior valor do argumento
- MIN(argumento) – Retorna o menor valor do argumento
- SUM(argumento) – Retorna o somatório dos valores do argumento
- COUNT(argumento) – Retorna a número de linhas do argumento

As funções agregadas normalmente usam como argumento um nome de coluna ou uma expressão que tenha um nome de coluna como componente, mas podemos usá-las com qualquer expressão numérica ou de datas.

A consulta a seguir lê todos os valores da coluna DESCONTO e fornece os percentuais médio, máximo e mínimo de desconto oferecidos aos hóspedes do Visual Spa. Como as função AVG, MAX, MIN e SUM ignoram valores nulos, o valor AVG (média) é realmente o desconto médio apenas daqueles hóspedes que obtiveram algum desconto:

```
SELECT AVG(desconto), MAX(desconto), MIN(desconto) FROM quarto
```

Outro exemplo:

```
SELECT MIN(desconto) * AVG(desconto) FROM quarto
```

Vejamos alguns exemplos:

- Mostrar o menor e o maior desconto para o quarto:

```
SELECT MIN(desconto), MAX(desconto) FROM quarto
```

- Mostrar a quantidade total de descontos para um determinado quarto.

```
SELECT SUM(desconto) FROM quarto WHERE nome = 'MIGUEL'
```

- Qual a média dos descontos dos Quartos

```
SELECT AVG(desconto) FROM quarto
```

- Quantos quartos apresentam um desconto superior a 10%

```
SELECT COUNT(*) FROM quarto WHERE desconto > 0.10
```

Agregação com a cláusula GROUP BY

A cláusula GROUP BY reúne diferentes linhas do resultado de uma consulta em conjuntos de acordo com as colunas nela mencionadas, chamadas colunas formadoras de grupos. As linhas são "agrupadas" de duas formas, ou em dois aspectos.

A consulta abaixo exemplifica a primeira forma, na qual todas as linhas que contêm o mesmo valor na primeira coluna especificada são exibidas em grupos no resultado. Neste caso, a coluna formadora de grupo é QUARTO, e todas as linhas que tenham o mesmo valor aparecerão juntas no resultado:

```
SELECT * FROM quarto GROUP BY quarto
```

Caso existam linhas em branco para a coluna QUARTO, ou seja, com um valor nulo, também seriam agrupadas. E as linhas são agrupadas da mesma maneira para cada coluna formadora de grupos subsequente, embora isto não esteja aparente no exemplo dados pois só contém duas colunas. Existem as seguintes funções agregadas — AVG, SUM, MAX, MIN e COUNT — que devem ser usadas com a cláusula GROUP BY pois geram um único valor.

A cláusula GROUP BY também pode ser usada em consultas contendo uma cláusula WHERE. Neste caso, a GROUP BY é codificada depois da cláusula WHERE. Por exemplo, a consulta abaixo exibe quantos hóspedes ocuparam os quartos depois do dia 01 de janeiro por quarto:

```
SELECT SUM(quarto) FROM quarto WHERE chegada > '2010-01-01' GROUP BY quarto
```

Agregação com a cláusula HAVING

A cláusula HAVING nos permite estreitar a área de atuação da cláusula GROUP BY da mesma forma que a cláusula WHERE estreita a área de atuação da cláusula SELECT, ou seja, através de uma condição de pesquisa.

Com o exemplo abaixo, criaremos uma nova tabela, **PESO** que será usada nos exemplos deste tópico. Criamos uma tabela através da declaração CREATE TABLE abaixo. Esta tabela se destina a registrar as pesagens periódicas dos hóspedes do Visual Spa.

```
CREATE TABLE peso (
 nome VARCHAR(25) NOT NULL,
 quando DATE,
 peso DECIMAL(4,1),
 PRIMARY KEY (nome, quando))
```

Após a inclusão dos dados, a tabela PESO terá a seguinte aparência:

NOME	QUANDO	PESO
MIGUEL	01-01-2010	66.4
MIGUEL	02-01-2010	66.2
RAQUEL	01-01-2010	86.5
RAQUEL	02-01-2010	86.1
RAQUEL	03-01-2010	85.8
JOANA	01-01-2010	66.7
JOANA	02-01-2010	66.3
LUCIANA	01-01-2010	67.5

Tabela 3

A seguinte tabela contém os registros de peso dos hóspedes por dia. A consulta a seguir indica os hóspedes que tiveram uma diferença entre seu peso mínimo e seu peso máximo.

```
SELECT nome, MAX(peso) - MIN(peso) diferenca FROM peso
GROUP BY nome
HAVING MAX(peso) - MIN(peso) > 0.0
```

Passemos para um outro exemplo. Esta consulta mostra os hospedes que realizaram mais de duas pesagens:

```
SELECT nome, COUNT(nome) FROM peso GROUP BY nome HAVING COUNT(nome) > 2
```

Ordenando os Dados

A cláusula **ORDER BY** permite classificar as linhas do resultado alfabética e numericamente, em ordem crescente ou decrescente. O padrão é a ordem crescente. A cláusula ORDER BY deve ser sempre a última cláusula da consulta.

No exemplo a seguir, obtemos os nomes dos hóspedes classificados em ordem decrescente através da palavra chave **DESC** na cláusula ORDER BY depois do nome da coluna a ser ordenada. Usamos a palavra-chave DISTINCT para suprimir as linhas duplicadas:

```
SELECT DISTINCT nome FROM quarto ORDER BY nome DESC
```

As classificações são, por padrão, efetuadas em ordem crescente, a não ser que seja indicada a palavra-chave DESC após o nome da coluna. Entretanto, podemos explicitar a ordem crescente para uma determinada coluna, usando-se a palavra-chave **ASC** após o nome da coluna na cláusula ORDER BY.

Produto Cartesiano

Podemos utilizar os seguintes Operadores Lógicos:

- AND para realizar uma conjunção de instruções
- OR para realizar uma disjunção de instruções
- NOT para realizar uma negação de instruções

Por exemplo. Quais são os hóspedes que têm o biotipo igual a 1 **e** o gênero igual a 'M'

```
SELECT nome FROM hospede WHERE biotipo = 1 AND genero = 'M'
```

Outro exemplo. Quais são os hóspedes que têm o biotipo igual a 1 **ou** o gênero igual a 'M'

```
SELECT nome FROM hospede WHERE biotipo = 1 OR genero = 'M'
```

Outro exemplo. Quais são os hóspedes que não têm o gênero igual a 'M"

```
SELECT nome FROM hospede WHERE NOT (genero = 'M')
```

Composição (JOIN)

Este é o comando mais comum utilizado e sua característica é só trazer registros que contenham correspondência nas outras tabelas utilizadas. Podemos listar todos os hóspedes e os quartos aos quais estes pertencem, com o seguinte código:

```
SELECT h.nome, q.quarto FROM hospede h
      INNER JOIN quarto q ON h.nome = q.nome
```

Outro modo que podemos realizar é mostrar todos os hóspedes, mesmo que não existam quartos associados a este:

```
SELECT h.nome, q.quarto FROM hospede h
      LEFT JOIN quarto q ON h.nome = q.nome
```

Ou o inverso, queremos ver todos os quartos, mesmo que estes não contenham nenhum hóspede:

```
SELECT h.nome, q.quarto FROM hospede h
      RIGHT JOIN quarto q ON h.nome = q.nome
```

Concluímos então que as instruções LEFT JOIN e RIGHT JOIN são utilizadas para resolver a não correspondência de dados entre diferentes tabelas.

O **NATURAL JOIN** e o **STRAIGHT_JOIN** fazem exatamente a mesma coisa que o **INNER JOIN** em questão de resultado, porém com algumas particularidades:

- **NATURAL JOIN**: não será necessário identificar quais colunas serão comparadas, pois será realizada uma comparação entre os campos com mesmo nome.

```
SELECT h.nome, h.altura, p.peso FROM hospede h NATURAL JOIN peso p
```

- **STRAIGHT_JOIN**: faz com que a tabela a esquerda seja lida primeiro, isso é utilizado quando o otimizador do JOIN coloca as tabelas em ordem errada. Isto é muito pouco utilizado.

```
SELECT h.nome, h.altura, p.peso FROM hospede h
   STRAIGHT_JOIN peso p ON h.nome = p.nome
```

Obs.:

1 – É possível substituir o **ON** por **USING** quando o nome das colunas nas duas tabelas for idêntico.

```
SELECT h.nome, h.altura, p.peso FROM hospede h INNER JOIN peso p USING(nome)
```

2 – O uso do INNER é opcional.

```
SELECT h.nome, h.altura, p.peso FROM hospede h JOIN peso p USING(nome)
```

Predicados de Procura

Predicados Relacionais

São aqueles que relacionam duas expressões segundo um operador. Os seguintes operadores são suportados:

Operador	Significado
=	Igual
! =	Diferente
<>	Diferente
>	Maior
! >	Não maior (menor ou igual)
<	Menor
! <	Não menor (maior ou igual)
>=	Maior ou igual
<=	Menor ou igual

Um predicado relacional segue a seguinte sintaxe: <expressão> <operador> <expressão>. Logo abaixo encontra-se um exemplo utilizando operadores relacionais. Neste exemplo, queremos obter todos os quartos cujo desconto é maior que 10%.

```
SELECT * FROM quarto WHERE desconto >= 0.10
```

Predicado BETWEEN

Compara um valor com uma faixa de valores. Os limites da faixa estão inclusos. Possui a seguinte sintaxe: <expressão> | not | between <expressão> and <expressão>

No exemplo seguinte recuperamos todas os quartos, cujo desconto está entre 10% e 20%:

```
SELECT nome, quarto FROM quarto WHERE desconto BETWEEN 0.10 AND 0.20
```

Predicado NULL

Utilizado para testar os valores nulos. Verifica, por exemplo, se colunas não contém nenhum valor armazenado. Possui a seguinte sintaxe: <nome da coluna> is | not | NULL. Caso seja necessário conhecer os quartos não possuem qualquer desconto, utilizaríamos a seguinte instrução:

```
SELECT nome, quarto FROM quarto WHERE desconto IS NULL
```

É possível utilizar o operador **not** juntamente com a palavra-chave is (do modo, *is not NULL*) para, por exemplo, montar condições de colunas que não possuem valores nulos.

Predicado LIKE

O predicado LIKE procura por strings que se encontram dentro de um determinado padrão. O predicado LIKE só pode ser usado com tipos de dados CHAR e VARCHAR.

A sintaxe para o predicado é seguinte:

<nome da coluna> | not | like < padrão de pesquisa >

O padrão de pesquisa trata-se de uma string a ser comparada com a coluna especificada. Este padrão pode ser formatado com a utilização de coringas, permitindo a procura de substrings. Os dois caracteres coringas existentes são o símbolo porcentagem '%' e o undescore '_'.

%	Equivale a zero ou mais caracteres
_	Equivale a um caractere qualquer

Os exemplos abaixo mostram a utilização do predicado LIKE:

- Verificar os hóspedes que possuem o final de sua descrição a palavra "EL" (p.ex. MIGU**EL**).

```
SELECT * FROM hospede WHERE NOME LIKE '%EL'
```

- Verificar os hóspedes que possuem o início de sua descrição a palavra "MIG" (p.ex. **MIG**UEL).

```
SELECT * FROM hospede WHERE NOME LIKE 'MIG%'
```

- Verificar os hóspedes que possuem em sua descrição a palavra "GU" (p.ex. MI**GU**EL).

```
SELECT * FROM departamento WHERE localidade LIKE '%GU%'
```

Predicado EXISTS

É utilizado para testar se o conjunto de linhas resultantes de uma consulta é vazio. Sintaxe:

| not | exists < subselect >

O exemplo seguinte mostra a utilização do predicado **exists**. Neste caso desejamos todos os nomes dos hospedes que tenham pelo menos uma reserva. Para fazermos isto, utilizamos a seguinte lógica: para cada Hóspede, pesquisamos em Quarto se reservou algo.

```
select NOME from HOSPEDE where exists
      (select * from QUARTO where HOSPEDE.NOME = QUARTO.NOME)
```

Predicado IN

O predicado in compara um valor com um conjunto de valores. Este conjunto de valores pode ser uma lista ou o resultado de um subselect. Sintaxe:

<expressão> | not | in < subselect >

ou

<expressão> | not | in < lista de valores >

Os seguintes exemplos demonstram o uso do predicado in:

- Selecionar todos os quartos que não possuem os descontos de 10% e 20%.

```
select * from QUARTO where DESCONTO not in (10,20)
```

- Selecionar todos os quartos reservados pelos hóspedes que possuem a altura superior a 2 metros.

```
select * from QUARTO where NOME in
      (select NOME from HOSPEDE where ALTURA > 2)
```

União

Cria uma consulta união, que combina os resultados de duas ou mais consultas ou tabelas independentes, é importante que ambas consultas retornem os mesmos tipos de dados na sequencia.

Sintaxe: [TABLE] consulta1 UNION [ALL] [TABLE] consulta2 [UNION [ALL] [TABLE] consultaN [...]]

Podemos mesclar os resultados de duas ou mais consultas, tabelas e instruções SELECT, em qualquer combinação, em uma única operação UNION.

Subconsultas

Uma **subconsulta** é uma consulta que faz parte de uma outra consulta para auxiliar esta, como exemplo, "criar uma consulta para verificar o peso atual de cada hospede baseado na última pesagem realizada por este".

Para montarmos a consulta principal precisamos associar a tabela **hospede** com a tabela **peso**, entretanto, o peso mostrado é o primeiro registro de cada hospede:

```
SELECT h.nome, p.peso FROM hospede h INNER JOIN peso p USING(nome) GROUP BY nome;
```

Podemos obtermos as últimas datas para cada hospede, podemos utilizar como base a seguinte consulta:

```
SELECT nome, MAX(quando) FROM peso GROUP BY nome;
```

A ideia de uma subconsulta é unir duas ou mais consultas para produzir um único resultado:

```
SELECT h.nome, p.peso FROM hospede h INNER JOIN peso p USING(nome)
WHERE p.quando = (SELECT MAX(quando) FROM peso WHERE nome = p.nome)
GROUP BY h.nome;
```

Podemos também utilizar subconsultas para simplificar o uso de comandos JOIN, por exemplo, desejamos saber quantas pesagens cada hóspede realizou. Utilizamos o seguinte SQL:

```
SELECT h.nome, COUNT(p.peso) FROM hospede h LEFT JOIN peso p USING(nome) GROUP BY
nome;
```

Para obtermos um resultado similar, podemos utilizar uma subconsulta.

```
SELECT h.nome, (SELECT COUNT(*) FROM peso WHERE nome = h.nome) FROM hospede h;
```

Trabalhando com Views

Muitas das consultas podem ser derivadas de várias tabelas e gerar um trabalho considerado para montar todos os JOIN's, utilizar determinados índices para obter uma boa performance da consulta. Sabemos também que a cada momento podemos ter que re-escrever pedaços de uma consulta e conseguir os mesmos resultados de antes é uma tarefa séria, já que existem várias formas para se escrever uma mesma consulta, levando em conta a ordem em que as tabelas aparecem, a ordem dos campos entre outros detalhes.

Tudo isso implica em ganho ou perda de performance. De fato, quando temos um grande trabalho de ajuste de performance em uma consulta, podemos rapidamente transformar esta consulta em uma **View** (visão), que permanecerá armazenada no servidor de bancos de dados em forma de tabela, para que possamos usá-la todas as vezes que esta seja necessária, desta forma aproveitaremos todo o trabalho realizado.

Uma *View* é um objeto que pertence a um banco de dados, baseada em declarações de consulta, retornando uma determinada visualização de dados de uma ou mais tabelas. Esses objetos são chamados por vezes de "virtual tables", formada a partir de outras tabelas que por sua vez são chamadas de "based tables" ou ainda outras *Views*. E alguns casos, as *Views* são atualizáveis e podem ser alvos de declaração INSERT, UPDATE e DELETE, que na verdade modificam sua "based tables".

Os benefícios da utilização de *Views*, além dos já citados, são:

- Uma *View* pode ser utilizada, por exemplo, para retornar um valor baseado em um identificador de registro;
- Pode ser utilizada para promover restrições em dados para aumentar a segurança dos mesmos e definir políticas de acesso em nível de tabela e coluna. Podem ser configurados para mostrar colunas diferentes para diferentes usuários do banco de dados;
- Pode ser utilizada com um conjunto de tabelas que podem ser unidas a outros conjuntos de tabelas com a utilização de JOIN´s ou UNION.

Criando Views

Para definir *Views* em um banco de dados, utilizamos a declaração CREATE VIEW, a qual possui a seguinte sintaxe:

```
CREATE [OR REPLACE] [ALGORITHM = algorithm_type] VIEW
 VIEW view_name [(column_list)] AS select_statement
 [WITH [CASCADED | LOCAL] CHECK OPTION]
```

view_name: é o nome que damos ao objeto View que criarmos. Esse nome poderá ser não qualificado, quando criado no banco de dados corrente ou totalmente qualificado quando criarmos em um banco de dados que não está definido no contexto atual (db_name.view_name).

column_list: recurso para que possamos sobrescrever os nomes das colunas que serão

recuperadas pela declaração SELECT;

select_statement: é a sua declaração SELECT e indica a forma na qual você deseja que os dados sejam retornados. Tal declaração deverá indicar a forma a qual você deseja retornar os dados, podendo ser utilizado funções, JOIN's e UNION. Podem ser utilizadas quaisquer tabelas ou *Views* contidas no servidor de bancos de dados *MySQL*, observando a questão de nomes totalmente qualificados ou não.

OR REPLACE: pode ser utilizada para substituir uma *View* de mesmo nome existente no banco de dados ao qual ela pertence. Pode-se utilizar ALTER TABLE para o mesmo efeito;

ALGORITHM: essa cláusula define qual algoritmo interno utilizar para processar a *View* quando a mesma for invocada. Estes podem ser UNDEFINED (cláusula em branco), MERGE ou TEMPTABLE.

WITH CHECK OPTION: todas as declarações que tentarem modificar dados de uma *View* definida com essa cláusula serão revisadas para checar se as modificações respeitarão a condição WHERE, definida no SELECT da *View*.

Ao criar uma *View*, um arquivo com extensão ".frm", com o mesmo nome desta também é criado no diretório do banco de dados, sob o diretório de dados do MySQL (datadir). Para iniciarmos com a mão na massa, definiremos uma *View* simples para armazenar a consulta vista na seção anterior.

```
CREATE VIEW vw_ultimaPesagem AS
 SELECT h.nome, MAX(p.quando), MAX(q.saida) FROM hospede h
 INNER JOIN peso p ON (h.nome = p.nome)
 INNER JOIN quarto q ON (h.nome = q.nome)
 GROUP BY h.nome
```

A partir desse momento, observe que um comando SHOW TABLES, veremos que uma tabela adicional foi criada, a **vw_ultimaPesagem**. Para uma conceituação mais ampla, uma *View* é um mapeamento lógico de várias tabelas contidas em um ou mais bancos de dados que por sua vez estão em um servidor MySQL. No caso da *View*, temos uma tabela virtual baseada em uma tabela chamada de base (hospede).

Sua utilização é como uma tabela comum, então podemos aplicar a seguinte instrução:

```
SELECT * FROM vw_ultimaPesagem
```

Outros Exemplos de Views

Views podem ser constituídas facilmente, como vimos anteriormente. Para termos *Views* que podem receber declarações de atualização tais como UPDATE e DELETE, que de fato alteram as tabelas base ("based tables"), temos que ter alguns cuidados na criação do objeto de visualização.

Analisemos a seguinte situação: "criar uma *View* para contar quantas pesagens cada hospede realizou, mostrando todos os hóspedes mesmo que este não tenha realizado nenhuma pesagem". Para esta *View* podemos utilizar o seguinte comando.

```
CREATE VIEW vw_quantidadePesagem AS
```

```
SELECT h.nome, COUNT(peso) quantidade FROM hospede h
LEFT JOIN peso p USING(nome)
GROUP BY nome
```

Para eliminar uma View, utilizamos o comando:

```
DROP VIEW [nome_da_view]
```

5. Exercícios de Fixação

Projeto Treinamento

1. Criar uma base de dados chamada **PROJETO01**.
2. Criar a tabela **FUNCIONARIO** para comportar a seguinte estrutura:

MATRICULA	NOME	DTCONTRATO	SALARIO
ARC01	RAFAEL	04-01-2010	780.40
ARC02	RAQUEL	11-01-2010	660.20
ARC03	SAMAEL	04-01-2010	860.50
ARC04	ISMAEL	11-01-2010	860.50
ARC05	RAQUEL	18-01-2010	660.20

Tabela 4

3. Criar a tabela **CURSO** para comportar a seguinte estrutura:

CODIGO	DESCRICAO
C01	SQL Básico
C02	SQL Avançado
CO3	Linguagem PL/SQL

Tabela 5

4. Criar a tabela **ESPECIALIZACAO** para comportar a seguinte estrutura:

MATRICULA	CURSO	DTCURSO
ARC01	C01	04-01-2010
ARC01	C02	11-01-2010
ARC01	C03	18-01-2010
ARC02	C02	11-01-2010
ARC02	C03	18-01-2010
ARC03	C01	04-01-2010
ARC03	C03	18-01-2010

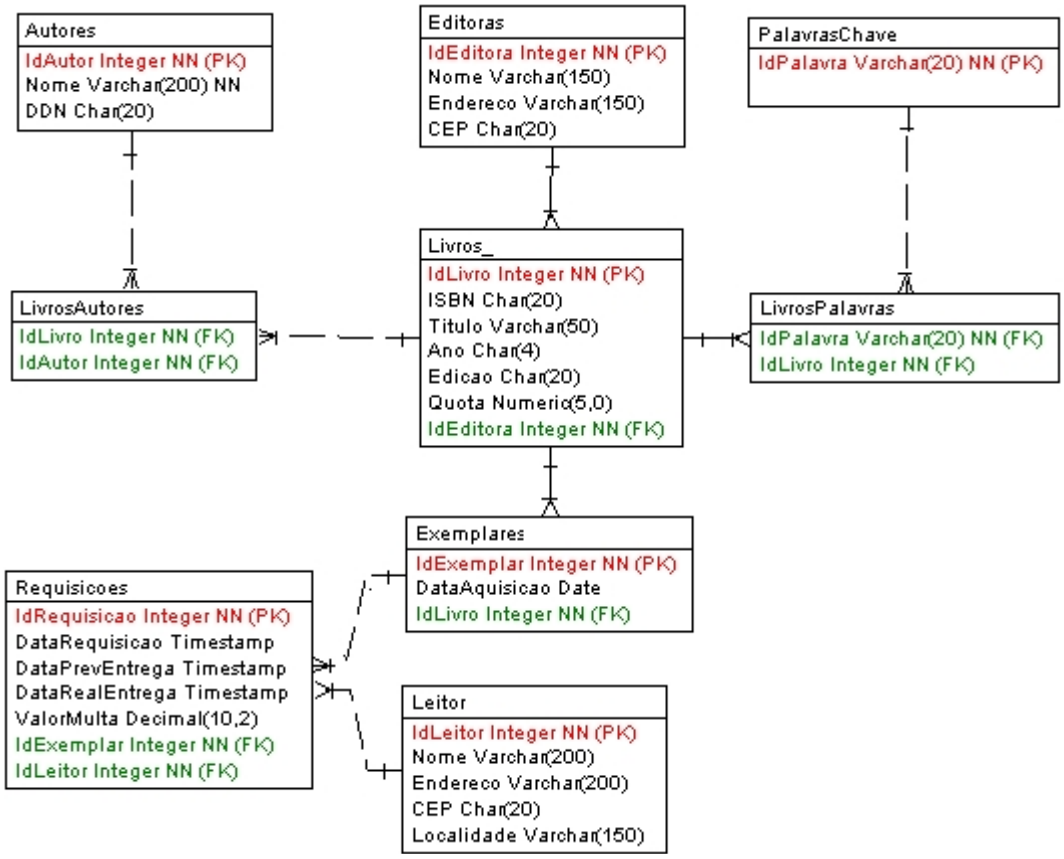
Tabela 6

5. Inserir os dados conforme as estruturas apresentadas.
6. Criar uma consulta que mostre quais funcionários não realizaram nenhum curso.
7. Criar uma consulta que mostre quais funcionários realizou todos os cursos.
8. Prover os seguintes aumentos de salário:
    1. Para o funcionário que realizou 1 curso aumento de 10%.
    2. Para o funcionário que realizou 2 cursos aumento de 15%.
    3. Para o funcionário que realizou 3 cursos aumento de 40%.
9. Eliminar os funcionários que não realizaram nenhum curso.

10. Criar um **SCRIPT** para realizar todas estas ações descritas.

Projeto Alexandria

1. Criar uma base de dados chamada **PROJETO02**.
2. Criar as tabelas para este projeto de acordo com o seguinte modelo:



3. Criar registros para popular este banco de dados.
4. Escrever comandos SQL que dão resposta às seguintes questões:
    1. Quais os códigos dos exemplares do livro com o título "Código de Da Vinci"?
    2. Quais os títulos dos exemplares, ordenados alfabeticamente, requisitados no mês de Janeiro de 2004?!
    3. Que códigos de exemplares é que não foram requisitados em Janeiro de 2004?
    4. Código, nome dos autores e quantidades de livros que escreveram livros para a Editora "FCA"?
    5. Quais os valores das multas pagas no mês de Outubro de 2004 em livros contendo a palavra-chave "Informática"?!

6. Que Leitores requisitaram todos os livros da Editora "Centro Atlântico"?
7. Quais os livros para os quais a biblioteca adquiriu pelo menos um exemplar no mesmo ano da sua edição.
8. Quais os códigos e nomes de Leitores que nunca tiveram uma multa por entrega de livro fora de prazo?!
9. Quantas requisições foram efetuadas em Fevereiro de 2004?
10. Qual o somatório total das multas pagas em 2003?
11. Qual o valor máximo de uma multa paga em cada ano?
12. Qual o período médio de requisição de um livro?

Projeto Empresa

1. Criar uma base de dados chamada **PROJETO03**.
2. Criar as tabelas para este projeto, e defina os tipos de cada campo:
   - FUNCIONARIO (matricula, nome_func, anos_servico, salario, cod_orgao)
   - ORGAO (cod_orgao, nome_orgao)
   - DEPARTAMENTO (cod_depto, localizacao)
   - DIVISAO (cod_div)
   - SUBORDINADO (cod_depto, cod_div)
   - GERENCIA (mat_ger, cod_orgao)
   - FUNC_COMUM (mat_func, cargo)
   - PROJETO (cod_proj, nome_proj, inicio, fim)
   - ALOCA (cod_proj, mat_func, data_alocacao)
   - COORDENA (cod_projeto, mat_coord, data_alocacao)
   - SUPERVISIONA (mat_superior, mat_func)
   - PROJ_TECNICO (cod_proj, area_pesquisa)
3. Criar registros para popular este banco de dados.
4. Escrever comandos SQL para resolver às seguintes questões:
   1. Listar todos os dados da tabela Funcionários ordenados por matrícula.
   2. Fornecer o número total de empregados da companhia.
   3. Listar os nomes dos funcionários que tenham entre 10 e 12 anos de serviço (inclusive).

4. Listar o nome, a matrícula e o salário de todas as pessoas cujo salário não esteja entre R$ 1.000,00 e R$ 3.000,00.
5. Listar os nomes dos funcionários com exatamente 5, 8 ou 13 anos de serviço, ou cujo valor de anos de serviço seja nulo.
6. Listar os nomes dos funcionários que tenham AN ou ON como o segundo e terceiros caracteres de seu nome.
7. Listar a matrícula, nome e salário de todas as pessoas em ordem alfabética de nome.
8. Listar os anos de serviço, matrícula, nome e salário em ordem decrescente de anos de serviço, e dentro de cada ano em ordem decrescente de salário.
9. Listar o nome do departamento, a matrícula, o nome e o salário dos funcionários em ordem decrescente de salário em seu departamento.
10. Listar os nomes dos gerentes de divisão e o número e o nome da respectiva divisão.
11. Listar matrícula, nome, anos de serviço de todos os gerentes de departamentos que recebem salários maiores que R$ 2.000,00 ou que possuam matrículas maiores que 30, e que tenham mais de 6 anos de serviço.
12. Listar os nomes dos gerentes de divisão que tenham ND ou LA como o segunda e terceira letra de seu nome.
13. Listar a matrícula, nome, cargo, anos de serviço e salário de todas as pessoas com 4 ou mais anos de serviço ou que ganhem salário superior a R$ 500,00.
14. Listar o nome do departamento, matrícula, nome e salário de todos os gerentes em ordem decrescente de salário dentro de departamento.
15. Listar o nome dos funcionários, o nome de seu departamento e o nome de seus respectivos supervisores ordenado por matrícula de funcionário.
16. Listar a matrícula, nome de todos os supervisores e os nomes de seus respectivos subordinados, ordenado por matrícula do supervisor e do subordinado.
17. Listar o nome e a matrícula dos funcionários que não são gerentes.
18. Listar as matrículas, os nomes e os departamentos de todos os empregados que gerenciam algum departamento.
19. Listar o código, o nome e as datas de todos os projetos técnicos.
20. Listar a média salarial por anos de serviço dos funcionários da companhia
21. Listar a média salarial de cada departamento.
22. Listar para cada número de anos de serviço o número de empregados e seu salário médio, mas somente para aqueles grupos que possuam mais de duas pessoas.

23. Listar para cada supervisor, o seu nome e o número de subordinados que este possui.
24. Listar o salário médio dos empregados por local e cargo para grupos de mais de um empregado.
25. Listar o número e o nome das divisões, o código e o nome dos respectivos departamentos subordinados a esta, ordenados por números das divisões.
26. Listar todos os atributos de projeto em ordem crescente de código
27. Descobrir qual a média de ganhos totais e quantos empregados são considerados para cálculo dessa média.
28. Listar o nome, a matrícula e o salário de todas as pessoas que não ganhem salário inferior a R$ 1.000,00.
29. Listar matrícula, nome, anos de serviço e salário de todas as pessoas com mais de 6 anos de serviço e que, ou seja gerente de departamento ou ganhe salário superior a R$ 3.000,00
30. Listar a matrícula, o nome e o salário de todas as pessoas em ordem decrescente de salário.
31. Listar o nome, a matrícula e o departamento de todas as pessoas que não trabalhem nos departamentos 10 e 20.
32. Listar a localização, a matrícula e o nome de todos os empregados do departamento de compra e venda.
33. Listar a matrícula, o nome e o salário de todos os coordenadores de projeto que não ganhem salário inferior a R$ 1.500,00.
34. Listar o nome e a localização de todos os departamentos subordinados à divisão "administração".
35. Listar o nome e a matrícula de todos os funcionários alocados no projeto “holografia”.
36. Listar o nome e a matrícula de todos os gerentes de departamento e o respectivo nome do departamento gerenciado e da divisão a que são subordinados.
37. Listar o salário médio para os departamentos que possuam salário médio superior à média da companhia.
38. Listar o menor, o maior e a média de salários de cada departamento que possua salário médio superior a R$ 1.000,00.
39. Listar o salário médio dos empregados por departamento e local para grupos de mais de um empregado.
40. Listar o salário médio dos empregados por local para grupos de mais de um empregado.
41. Listar a matrícula e os nomes dos coordenadores, e matrícula dos funcionários

que são supervisionados por eles.

42. Listar o nome e a matrícula dos funcionários que são vendedores.
43. Listar todos os funcionários que trabalham na Zona Sul.
44. Listar o nome e a matrícula de todos os empregados que trabalham na divisão "finanças".
45. Listar departamento, matrícula, nome e anos de serviço de todos os funcionários indicando aqueles com 10 anos de serviço. Ordene por matrícula dentro de departamento.
46. Listar os dados de departamento com o maior salário médio.
47. Listar a matrícula, o nome e o salário dos funcionários que ganham salário superior os do seu respectivo gerente.
48. Listar a matrícula, nome, cargo, anos de serviço e salário dos funcionários que não estão alocados a projetos, com 4 ou mais anos de serviço e que ganhem salário superior a R$ 500,00.
49. Listar a matrícula e a quantidade de projetos dos coordenadores que coordenam mais de um projeto.
50. Listar a matrícula e o nome do gerente do funcionário de cargo (Auxiliar Administrativo) que ganha o maior salário.
51. Listar a matrícula e o nome do funcionário de cargo (Auxiliar Administrativo) que ganha o maior salário. Liste também o nome do seu gerente.

Projeto Vendas

1. Criar uma base de dados chamada **PROJETO04**.
2. Defina as tabelas e seus campos para este projeto.
3. Criar registros para popular este banco de dados.
4. Escrever comandos SQL para resolver às seguintes listagens:
    1. Nome, endereço, CEP e UF de todos clientes que não pertencem do estado de DF.
    2. Nome, CGC, estado e cidade do cliente que possui a identificação 7121.
    3. Dados do cliente que possui o identificação 260.
    4. Dados dos vendedores que possui a faixa de comissão assalariado.
    5. Dados dos vendedores que possuem salário maior ou igual a 2.400,00.
    6. Dados dos vendedores no qual sua faixa de comissão seja diferente do tipo "Comissionário".
    7. Dados dos pedidos no qual o prazo de entrega seja igual ou menor 20.

8. Dados dos pedidos do cliente que possui a identificação 720.
9. Dados dos produtos cujo à unidade do produto seja em metros.
10. Dados dos produtos no qual o seu valor unitário seja maior ou igual 1,20.
11. Nome, salário fixo, comissão dos vendedores que possuem sua comissão igual a "C" e o salário maior ou igual a 1.780,00.
12. Nome dos vendedores que possuem o salário diferente de 2.490,00.
13. Unidade, descrição, valor unitário dos Produtos que contenha sua unidade em metros e valor maior ou igual a 0,73.
14. Nome, endereço, cidade, CEP, UF dos Clientes que moram no DF e possuem o CEP entre 70000 e 70200.
15. Pedidos que não tenham prazo de entrega igual a 30 dias.
16. Nome e endereço e CEP, UF, cidade dos Clientes que moram no DF ou em TO.
17. Vendedores que não possuem sua faixa de comissão entre A e B.
18. Código, nome, endereço, CGC do Cliente que começam com a letra S.
19. Produtos que tenham valor unitário na faixa de 1.55 até 6.18.
20. Pedidos que não tenham o número igual 131.
21. Nome, endereço, cidade, UF, CEP do Cliente que residam em avenidas.
22. Código, valor unitário, unidade e sua descrição dos Produtos que não tenham seu produto unitário na faixa de comissão 0,50 a 2,79.
23. Pedidos, prazo de entrega dos Pedidos que ficam entre 91 e 137.
24. Números e quantidades dos produtos que possuem sua quantidade entre 5 e 37.
25. Descrição e o valor unitário dos produtos que tenham a unidade em metros, e em ordem crescente de valor unitário.
26. Novo salário fixo dos vendedores de faixa de comissão C calculando com base no reajuste de 58% acrescido dr R$ 239,00 de bonificação e ordenar pelo nome do vendedor.
27. Menor salário dos vendedores e após listar o maior mostrando seu nome.
28. Quantidade total pedido para o produto "queijo" de código 25 nos itens dos pedidos.
29. Médias dos salários dos vendedores.
30. Clientes com seus pedidos.
31. Quantidade de clientes não possuem pedidos.
32. Vendedores que emitiram pedidos com prazo de entrega superiores a 8 dias e tenham salários igual ou maiores que R$ 970,00.

33. Clientes que tem o prazo de entrega maior que 7 dias para o produto "Chocolate".
34. Vendedores que venderam o produto "Vinho" em quantidade superior ou igual a 1 litro.
35. Clientes que fizeram pedidos para o vendedor "Jesiel".
36. Clientes da cidade de "Brasília" ou de "Sobradinho" que tiveram seus pedidos atendidos pelo vendedor "Raphael".