

# **INTRODUÇÃO A BANCOS DE DADOS**

**José Luís Carneiro**

Salvador  
2004

# Índice

<b>APRESENTAÇÃO</b>	<b>3</b>
<b>SISTEMAS DE BANCOS DE DADOS</b>	<b>4</b>
Conceitos Iniciais	4
Componentes de um Sistema de Banco de Dados	5
Volatilidade em um Sistema de Banco de Dados	6
Entidades e Relacionamentos	6
Modelos de Dados	7
Por que usar um SGBD?	9
Independência de Dados	10
A arquitetura de um Sistema de Banco de Dados	11
Atribuições do Administrador de Banco de Dados	15
Funções de um Sistema Gerenciador de Banco de Dados	16
Arquitetura Cliente/Servidor	18
Processamento Distribuído	19
<b>BANCOS DE DADOS RELACIONAIS</b>	<b>22</b>
Relações	22
Visões	23
Definição formal de uma relação	24
Propriedades de uma relação	25
Chaves	25
Normalização	26
O Dicionário de Dados	29
Integridade	30
<b>LINGUAGEM DE ACESSO ESTRUTURADA (SQL)</b>	<b>32</b>
Características da SQL	32
Estrutura da SQL	33
Instruções SQL	35
Ferramentas oferecidas pela linguagem SQL	51
<b>TÓPICOS ADICIONAIS</b>	<b>57</b>
Transações	57
Concorrência	58
Segurança do Banco de Dados	61
Otimização	63
<b>BIBLIOGRAFIA</b>	<b>65</b>

# Apresentação

A maioria das atividades desenvolvidas em uma empresa envolve, de uma forma ou de outra, a manipulação de dados e informações. As informações de uma empresa muitas vezes valem mais que seu patrimônio físico.

Fosse em antiquados sistemas de arquivamento em papel, ou em sistemas computadorizados, o arquivamento e acesso aos dados da empresa sempre tiveram papel fundamental.

No mercado atual, altamente competitivo, esse papel tornou-se crucial: a ineficiência em recuperar dados e processar informações pode determinar o fracasso de uma empresa. Podemos concluir que os bancos de dados têm atualmente uma enorme importância. Esse curso pretende auxiliá-lo(a) a obter um conhecimento geral sobre bancos de dados, de forma a desempenhar as funções de Analista de Sistemas.

Veremos, inicialmente, uma introdução à teoria de bancos de dados com o objetivo de obter os conhecimentos necessários para um estudo aplicado utilizando o PostgreSQL, um dos sistemas gerenciadores de bancos de dados de baixo custo mais completos e confiáveis atualmente.

Essa apostila foi desenvolvida para oferecer uma noção básica de banco de dados sem a obrigatoriedade de um livro-texto. Entretanto, é imprescindível, para aqueles que desejam se aprofundar, adquirir um bom livro sobre o assunto. Existem vários, muitos deles já traduzidos para o português. Tomaremos como guia o livro “Introdução a Sistemas de Banco de Dados” de C. J. Date.

# Sistemas de Bancos de Dados

## Conceitos Iniciais

O curso é sobre “bancos de dados”. Mas o que significa “dado”?

Sabemos que a palavra derivou do latim “*datu*” e que um dos seus significados é “conceder”.

Ou seja, são **fatos fornecidos que descrevem uma característica de um objeto ou evento do mundo real** (na verdade o termo “objeto” não é o ideal, mas para o momento basta).

Podemos dizer que a data de nascimento de uma pessoa é um dado sobre ela.

Coloquialmente é comum utilizarmos “dado” e “informação” como sinônimos. Para efeito do nosso estudo, no entanto, vamos diferenciá-los. Enquanto dados são os componentes básicos a partir dos quais a informação é criada, consideraremos que **informação é um dado inserido num contexto**; uma conclusão a que podemos chegar analisando um ou mais dados apresentados. Voltando ao nosso exemplo, se consideramos a data de nascimento como um dado, devemos considerar a idade dessa pessoa como uma informação sobre ela (obtida a partir da data de nascimento).

Vale observar que há ocasiões em que a fronteira entre dados e informações é muito tênue. Enquanto um dado pode ser apenas um amontoado de caracteres incompreensíveis fora de um contexto, a informação comunica algo que outra pessoa entende. Logo, alguns dados são informação para quem os compreende: o nome de uma pessoa, “Yoko Ono” por exemplo, é apenas um dado para uns enquanto é uma informação para outros.

E o que viria a ser um banco de dados? **Um conjunto de dados relacionados entre si armazenados segundo uma determinada lógica de forma para que possam ser recuperados quando necessário**. Por esse raciocínio, um cadastro de clientes (contendo seu nome, data de nascimento e endereço) pode ser considerado um exemplo de banco de dados.

## Definição Simplista de “Sistema de Banco de Dados”

Grosso modo, um “Sistema de Banco de Dados” é a metodologia utilizada para manter e acessar os dados armazenados num banco de dados. É essa metodologia que irá definir que dados estão armazenados, em que formato, sua classificação, etc.

Podemos começar comparando-o a um fichário destinado a conter os dados que se deseja armazenar. Para ser útil, esse nosso fichário deve permitir algumas operações básicas:

1. Acréscimo de fichas e de pastas que as agruparão por afinidade.

2. Localização de um determinado dado contido em uma ou em várias fichas.
3. Atualização dos dados contidos em cada ficha.
4. Eliminação de fichas e pastas indesejadas.

Portanto, um sistema informatizado de banco de dados, em princípio, diminuiria o trabalho de manter esse fichário.

Realmente, os primeiros sistemas informatizados que utilizavam bancos de dados o faziam de forma bastante similar ao antigo armazenamento em armários e fichários. Cada departamento possuía o seu banco de dados, normalmente uma versão eletrônica de formulários em papel pré-existent, contendo exclusivamente os dados utilizados pelos funcionários daquele departamento.

Repetição de dados em bancos espalhados por vários departamentos (e às vezes dentro de um mesmo departamento) era comum e freqüentemente causava inconsistências entre os dados e a duplicação do trabalho de atualização já que os sistemas não se comunicavam entre si.

Os atuais sistemas de bancos de dados têm muito mais atribuições e vantagens que seus antepassados.

## Componentes de um Sistema de Banco de Dados

Observando um sistema mais moderno de banco de dados podemos destacar alguns componentes principais: *hardware*, *software*, usuários e dados.

O *hardware* dependerá da quantidade de dados a ser manipulada, do tempo de resposta e nível de disponibilidade desejados, dos requisitos de segurança e dos recursos disponíveis. É um equilíbrio delicado que envolve conhecimentos de equipamentos que fogem ao escopo deste curso.

O *software* (também conhecido Sistema de Gerenciamento de Bancos de Dados ou SGBD) servirá de interface entre os usuários e os dados armazenados, dispensando que a maioria dos usuários (os usuários finais) tenha conhecimento de detalhes técnicos de armazenamento. Dessa forma, ao usuário médio basta saber **como** solicitar as operações desejadas ao SGBD, que se encarregará de executá-las.

Os usuários podem ser divididos em três categorias:

1. Os **Usuários Finais** são aqueles que utilizarão o sistema de banco de dados rotineiramente. O objetivo final de um sistema de banco de dados é atender às suas necessidades da forma mais eficiente possível garantindo informação coerente,

consistente e segura, na hora e na forma desejada, com o mínimo de conhecimento técnico possível.

2. Os **Administradores de Bancos de Dados** (DBA – *database administrator*) e os **Administradores de Dados** (DA – *data administrator*) são as pessoas responsáveis por analisar as necessidades dos usuários finais e desenhar o banco de dados de forma a atendê-las. É responsabilidade do administrador de dados, entre outras coisas, determinar quais dados devem ser armazenados, por quanto tempo e quem terá acesso a eles. Ao DBA cabe implementar essas definições e manter o banco de dados sempre disponível.
3. Os **Programadores de Aplicações** são responsáveis por desenvolver (normalmente em linguagens de alto nível) os aplicativos que receberão as solicitações dos usuários finais. Esses aplicativos encaminharão as solicitações ao SGBD e trarão de volta o resultado. Esses aplicativos, hoje normalmente visuais, muitas vezes eliminarão dos usuários finais até mesmo a necessidade de aprenderem os comandos de operação do SGBD.

## Volatilidade em um Sistema de Banco de Dados

Durante sua operação, um sistema de banco de dados manipula diversos dados internos, muitas vezes criados pelo próprio sistema, como variáveis, acumuladores e resultados de consulta. Esses dados têm um alto grau de volatilidade, sendo criados e apagados continuamente.

Enquanto estes dados são criados e apagados de forma transparente para o usuário, os dados inseridos no banco de dados pelo usuário são persistentes, isto é, permanecem indefinidamente no banco de dados, sendo removidos somente com uma **solicitação expressa** do usuário.

Podemos então definir um banco de dados como **uma coleção de dados persistentes utilizados pelas aplicações de uma organização**.

## Entidades e Relacionamentos

Durante a análise dos dados para o desenho de um sistema de bancos de dados, podemos observar que existem determinados objetos (ou eventos) do mundo real sobre os quais desejamos armazenar informações.

De um professor podemos armazenar, por exemplo, seu nome, a disciplina que leciona e seu salário. De um aluno, as disciplinas a que assiste, suas notas e sua frequência. E de uma disciplina, suas turmas, sua carga horária e seus pré-requisitos. Dizemos que professores, alunos e disciplinas são as **entidades** básicas do nosso banco de dados e que os dados que desejamos armazenar são suas **propriedades**.

Podemos dizer que entidades são objetos (ou eventos) do mundo real que têm dados a serem armazenados e, portanto precisam ser representadas no banco de dados.

Ainda podemos observar, em muitos casos, que na vida real existem **relacionamentos** entre essas entidades que também devem ser representados no banco de dados: um professor leciona uma determinada disciplina; um aluno assiste a uma determinada disciplina; e uma disciplina tem determinados pré-requisitos (que normalmente são outras disciplinas).

Uma coisa que devemos salientar é que os próprios relacionamentos têm, muitas vezes, informações que são necessárias: o relacionamento “aluno-assiste-a-disciplina” gera a pergunta “Em que sala?”. Concluimos que os próprios relacionamentos têm propriedades a serem armazenadas, caracterizando-se também como entidades.

As propriedades que armazenamos em nosso banco de dados nos permitem formular afirmações sobre essas entidades. Essas afirmações podem ser exclusivamente verdadeiras ou falsas. Do ponto de vista lógico, chamamos essas afirmações de **proposições**. Ou seja, o dado que armazenamos na propriedade está correto (proposição verdadeira) ou está incorreto (proposição falsa).

Consideremos também que, exceto em caso de erro de entrada, os dados armazenados num banco de dados estão corretos, já que correspondem a fatos fornecidos pelo mundo real (obtidos da própria entidade observada).

Podemos agora deduzir um outro conceito para Banco de Dados:

“Se as entidades são objetos do mundo real que têm dados a serem armazenados, se estes dados podem ser avaliados como proposições (verdadeiras ou falsas) e se não faz sentido armazenar proposições falsas num banco de dados, podemos dizer que um **banco de dados é uma coleção de proposições verdadeiras**”.

## Modelos de Dados

Um modelo de dados é uma definição **abstrata** dos objetos representados por estes dados, dos relacionamentos desses objetos entre si e de um conjunto de operadores e regras que os

usuários finais utilizam para interagir com o banco de dados. Como esses dados estarão implementados não importa aos usuários finais, daí o termo “definição abstrata”.

Os bancos de dados que utilizam entidades e seus relacionamentos baseiam-se num modelo formal, fundamentado na lógica e na matemática, chamado de **Modelo Relacional de Dados**.

As características principais deste modelo são:

1. Os dados são representados por relações em que cada linha pode ser lida como uma proposição verdadeira. Para facilitar a compreensão neste momento, considere que uma relação é muito semelhante a uma tabela.
2. Oferece operadores relacionais que, aplicados às relações, permitem extrair novas relações a partir das primeiras. Temos por exemplo, o operador restrição que extrai um determinado conjunto de linhas, e o operador projeção que extrai um determinado conjunto de colunas. Obviamente podemos considerar um conjunto de linhas e colunas como uma nova relação.

O Modelo Relacional é o único que realmente podemos considerar um modelo real, visto que é o único que demonstrou, matematicamente, ser completo e consistente. Sobre a definição teórica então, foram desenvolvidos os sistemas de banco de dados.

Nos caso de outros “modelos”, ocorre o inverso: a definição veio para explicar o sistema de banco de dados já desenvolvido.

Desse modo, muitos dos sistemas não-relacionais utilizam artifícios de implementação (como ponteiros) para que possam funcionar. O Modelo Relacional não utiliza esses artifícios já que é funcional desde o âmbito teórico (note porém, que isso não signifique que um SGBDR não utilize ponteiros no nível físico; na verdade, é quase certo que utilize, mas esses detalhes de armazenamento estão ocultos do usuário).

Enquanto no modelo relacional de dados o usuário vê os dados como tabelas, nos sistemas não-relacionais os dados são apresentados de outras formas:

1. Na **Abordagem Hierárquica** (IMS da IBM), os dados são apresentados como coleções de registros e as ligações entre eles. Essas ligações obedecem a uma hierarquia, formando uma estrutura de árvore. Os operadores oferecidos ao usuário são ponteiros que permitem seu deslocamento na árvore. É mais indicado para representar estruturas hierárquicas como relacionamentos “um para um” e “um para muitos”, pois apresenta alto nível de redundância de informações e ineficiência em estruturas mais complexas como os relacionamentos “muitos para muitos”.
2. Na **Abordagem em Rede** ou CODASYL (IDMS da *Computer Associates* e TOTAL da *Cincom Systems*), os dados também são representados como coleções de registros e

ligações entre eles, embora não haja uma relação hierárquica definida. Como um registro pode ter qualquer quantidade de superiores e dependentes imediatos, representa relacionamentos “muitos para muitos” mais diretamente que a abordagem hierárquica, mas seu esquema de definição tende a se tornar extremamente complexo. Os operadores oferecidos também são ponteiros, e permitem o deslocamento na rede. Uma variação dessa abordagem é a utilização de listas invertidas, apresentada pelo ADABAS (*Adaptable Data Base System*) da *Software AG*.

Existe ainda a **Abordagem Orientada a Objeto**, que aplica os conceitos da Teoria de Orientação a Objetos aos sistemas de bancos de dados. Atualmente poucos são os SGBD que apresentam essas características, o que mais se aproxima é o Oracle9i.



Uma curiosidade sobre o Modelo Relacional é que, ao contrário dos sistemas não-relacionais, ainda não há um produto comercial que implemente todas as suas características.

## Por que usar um SGBD?

A essa altura, você deve estar se perguntando: “Mas por que eu deveria usar um sistema gerenciador de banco de dados?”. Existem vantagens óbvias e outras menos aparentes. As principais são:

1. Menos trabalho repetitivo.
2. Os dados consultados são mais atualizados, uma vez que a máquina trabalha muito mais rápido que o homem.
3. Várias aplicações podem trabalhar sobre os mesmos dados, evitando o re-trabalho de desenhar um novo banco de dados para cada aplicação e reduzindo a redundância de dados.

4. Menos inconsistência e maior integridade dos dados, já que com menos redundância, é menos provável que uma alteração numa ocorrência de um dado não seja realizada nas demais ocorrências.
5. Maior segurança dos dados. Basta protegermos um banco de dados contra falhas e acidentes. A segurança de acesso é proporcionada pelo SGBD.
6. Os dados podem ser padronizados. Por exemplo, quando houve a redução de dígitos de código telefônico de DDD, pudemos assegurar num banco de dados bem desenhado que todos os códigos de DDD haviam sido alterados.
7. Os atuais SGBD oferecem suporte a transações (um conjunto de operações unidas logicamente em torno de um único evento). Se uma operação não puder ser concluída, nenhuma das outras é efetuada. Isso ajuda a evitar inconsistências em caso de problemas que interrompam a atualização dos dados. Veremos mais sobre transações adiante.
8. Independência de dados. Um sistema de banco de dados constitui-se numa camada entre o usuário e a implementação dos dados. O que o usuário vê é uma **representação** dos dados armazenados. A forma como essa representação é implementada é de conhecimento apenas do DBA e do SGBD.

## Independência de Dados

De todas as vantagens de um SGBD, a independência de dados é a mais aparente para os administradores de dados, administradores de bancos de dados e programadores de aplicações. A independência de dados é a imunidade oferecida pelo SGBD das aplicações e usuários às mudanças ocorridas no banco de dados. Existem dois tipos de independência de dados:

1. **Independência Física** – Imunidade às alterações na representação física e no método de acesso aos dados. Dessa forma, os bancos de dados podem ser projetados da forma mais eficiente possível. Quando do desenvolvimento de novas aplicações, os dados modelados podem ser aproveitados sem alterações: o SGBD disponibiliza os dados no formato esperado por cada aplicação, independente do formato em que foram armazenados. E, caso seja necessário alterar o formato dos dados devido a mudanças nos requisitos, isso pode ser feito com o mínimo impacto sobre os sistemas envolvidos. Um bom exemplo dessas alterações foi a mudança nos campos de datas,

aumentando de dois para quatro o número de dígitos do ano, que causou tanta polêmica na virada do milênio.

2. **Independência Lógica** – Imunidade às mudanças na estrutura lógica do banco de dados. Enquanto a independência física permite alterações transparentes ao formato de armazenamento dos dados modelados, a independência lógica minimiza o impacto causado por uma possível alteração na modelagem dos dados. Se o administrador de dados verificar que uma propriedade não deveria pertencer a uma determinada entidade e sim a uma outra, ou que mais propriedades são necessárias, o DBA pode realizar a alteração sem ter que reescrever todas as aplicações envolvidas.

Por essa razão, quando falamos de SGBD, fazemos uma distinção entre campos, registros e arquivos **armazenados** e **lógicos** (ou **visualizados**). O dado que está armazenado não precisa ser igual ao dado que é visualizado por um usuário (ou por uma aplicação).

Como exemplo, um dado que aparentemente pertence a uma entidade pode ser o resultado da junção de dados de diversas entidades diferentes; ou um dado numérico pode estar armazenado em formato hexadecimal (por questões de eficiência) enquanto os usuários o vêem no formato decimal, mais natural para nós humanos.

Aproveitaremos para definir os termos “campo”, “registro” e “arquivo”, já que apareceram com certa frequência daqui pra frente.

Campo é a menor unidade de dado sobre uma entidade. É o armazenamento de uma de suas propriedades. Considerando a entidade “Professor”, o nome do professor seria um campo.

Um registro é uma coleção de campos relacionados entre si. Define uma instância (um exemplar) daquela entidade. Ainda considerando a entidade “Professor”, cada um dos professores seria uma **instância** dessa entidade, e para cada um deles haveria um registro.

Já um arquivo é a coleção de todas as instâncias de um mesmo tipo de registro. No nosso exemplo, todos os professores existentes no nosso banco de dados.

## A arquitetura de um Sistema de Banco de Dados

Em 1972 foi criado um grupo de estudos para definir uma arquitetura padrão para um sistema de banco de dados que atendesse às demandas teóricas. O nome desse grupo foi “*ANSI/X3/SPARC Study Group on Data Base Management Systems*”, e a arquitetura passou a ser conhecida como ANSI/SPARC.

Segundo essa arquitetura, um sistema de banco de dados pode possuir três níveis:

1. Nível Interno ou Físico – Responsável pelo armazenamento dos dados.

2. Nível Conceitual ou Lógico Comunitário – Serve de interface entre o primeiro e o terceiro níveis.
3. Nível Externo ou Lógico do Usuário – Responsável pela visualização dos dados por parte do usuário.

## O Nível Externo (ou Visão Externa)

O nível externo constitui-se na visão que o usuário tem do banco de dados; como os dados são visualizados por ele. Dessa forma, haverá tantas dessas visões quantas forem as formas distintas que os dados podem ser visualizados (usuários diferentes, aplicações diferentes, etc.). Por usuário, podemos entender tanto o próprio DBA (ou um desenvolvedor de aplicações) quanto um usuário final. Porém, observemos agora o usuário final: ele só estará interessado em uma determinada parte do banco de dados e, do seu ponto de vista, essa visão é o banco de dados completo.

O termo ANSI/SPARC para essa visão é “**visão externa**”. Ela independe da forma como os dados estão armazenados (**independência lógica** dos dados, como vimos).

A cada visão externa, corresponde um **esquema externo** que contém as definições de tipo dos dados que são disponibilizados.

A visão externa poderá ser acessada de diversas formas, inclusive por meio de ferramentas bastante amigáveis, muitas vezes visuais (normalmente voltadas para o usuário final).

O desenvolvedor, entretanto, utilizará uma linguagem de alto nível para produzir o aplicativo desejado. As linguagens para desenvolvimento de aplicações normalmente utilizarão um subconjunto de comandos para acessar e manipular o banco de dados.

Chamamos a esse subconjunto de “sublinguagem de dados” e à linguagem de alto nível que o utiliza, de linguagem “hospedeira”.

Dessa forma, enquanto a linguagem hospedeira é responsável pelos recursos de programação (como variáveis, estruturas de decisão e repetição), a sublinguagem de dados responsabiliza-se pelo acesso ao banco de dados. Atualmente a sublinguagem de dados mais utilizada é a SQL. Veremos mais detalhes sobre ela adiante.

Dizemos que a sublinguagem de dados está **embutida** na linguagem hospedeira e, pela facilidade com que podemos distinguir entre elas, podemos classificá-las como **fracamente acopladas** (quando são facilmente distinguíveis) ou **fortemente acopladas** (quando só podem ser distinguidas com dificuldade).

Como a linguagem SQL normalmente não oferece suporte ao acoplamento forte, atualmente a grande maioria das linguagens possui acoplamento fraco. Porém, isso só é verificado do ponto de vista do desenvolvedor. Do ponto de vista do usuário final, isso é transparente.

Qualquer que seja a “sublinguagem de dados”, ela divide-se em pelo menos duas partes:

1. **DDL** (*Data Definition Language*) – Instruções para definição de estruturas, esquemas e visões no banco de dados.
2. **DML** (*Data Manipulation Language*) – Instruções para manipulação dos dados no banco de dados, permitindo inclusão, alteração, exclusão e consulta a esses dados.

Veremos mais sobre DDL e DML, quando tratarmos sobre SQL.

## O Nível Conceitual (ou Visão Conceitual)

Enquanto a visão externa é uma representação do banco de dados como ele é visto por um determinado usuário, a visão conceitual é uma representação dos dados como eles “realmente são no mundo real”.

É descrita pelo **esquema conceitual** e serve de interface entre a visão externa e o formato de armazenamento dos dados, sendo a responsável pela **independência física** dos dados.

Provê uma definição teórica dos dados, sem as limitações técnicas de implementação (como características de *hardware* e *software*) ou mesmo limitações de acesso de cada usuário. Portanto, a visão conceitual deve compreender **todo** o banco de dados, inclusive suas restrições de integridade e segurança, sem mencionar detalhes de armazenamento.

Por essa razão, a visão conceitual é a mais próxima da definição teórica obtida no levantamento do banco de dados. Dessa forma, sendo a visão conceitual independente dos dados, a visão externa, baseada nela, também será.

Infelizmente não há, atualmente, nenhum sistema que atenda completamente a essas condições. Todos eles, em maior ou menor grau, poluem a visão conceitual com detalhes de implementação. Pode ser que no futuro existam sistemas comerciais que consigam tal grau de abstração, mas nos sistemas atuais, a visão conceitual é pouco mais que a reunião de todas as diferentes visões externas e mais algumas restrições de segurança e integridade. Longe do ideal teórico.

## O Nível Interno (ou Visão Interna)

A visão interna é a camada de mais baixo nível das três, sendo a que mais se aproxima dos dados armazenados. É descrita pelo **esquema interno** e nela são definidas, por exemplo, a

seqüência de armazenamento e os formatos de cada campo (o campo “X” deverá ser armazenado em decimal ou em binário?), os índices que serão utilizados e as otimizações necessárias ao bom funcionamento do banco de dados.

Entretanto, a visão interna não abrange definições físicas do banco de dados, não há preocupação com dispositivos de armazenamento, nem com número de cilindros ou trilhas. Essas definições estão restritas ao **Nível Físico**, responsável pela manipulação dos registros físicos, fora da arquitetura de banco de dados.

Vale observar que, no nível físico, não nos referimos a registros, mas a **blocos** (ou **páginas**) que são as menores unidades de informação manipuladas numa única operação de entrada e saída. Um bloco tem um tamanho fixo (normalmente 1 kb, 2 kb ou 4 kb) diferente para cada sistema de banco de dados, que não corresponde necessariamente ao tamanho de um registro definido no nível conceitual. Na verdade, dependendo do tamanho dos registros, um bloco pode conter vários deles ou apenas parte de um.

## Mapeamentos

Como pudemos observar, as definições de cada visão não são diretamente correspondentes entre si. Nem sempre as definições de uma visão correspondem às definições de uma outra visão (como as visões externas e a visão conceitual, por exemplo).

A correspondência (ou “tradução”) das definições de uma visão para outra é chamada de **mapeamento**. Dessa forma, existe um **mapeamento conceitual/interno** (responsável pela correspondência entre esses níveis) e diversos **mapeamentos externos/conceituais** (um para cada visão externa distinta).

Como o próprio nome diz, o mapeamento conceitual/interno especifica como os registros definidos no nível conceitual são representados no nível interno. Qualquer alteração na estrutura do banco de dados (mesmo a criação de um novo índice ou a otimização de uma pesquisa) refletirá numa mudança neste mapeamento de forma a manter a **independência de dados física** característica do nível conceitual.

Um mapeamento externo/conceitual realiza um trabalho semelhante, só que define a correspondência entre uma (determinada) visão externa e o nível conceitual. Portanto, existem diversos mapeamentos externos/conceituais, um para cada visão externa.

Quanto à mutabilidade, o comportamento é análogo: qualquer alteração no nível conceitual deve ser logo refletida em **todos** os mapeamentos externos/conceituais afetados por ela. Dessa

forma, as visões externas não precisam ser alteradas, mantendo a **independência de dados lógica**.

Existem ainda mapeamentos externos/externos que permitem que visões externas possam ser baseadas em outras visões externas, ao invés de serem baseadas obrigatoriamente na visão conceitual. Isso poupa trabalho quando temos diversas visões externas semelhantes, evitando definir todas elas a partir do nível conceitual. Esse tipo de mapeamento é comum em sistemas de banco de dados relacionais.

## Atribuições do Administrador de Banco de Dados

Relembrando, o administrador de dados é responsável pela definição do Projeto Lógico da empresa, por exemplo: quais entidades devem ser avaliadas, quais dados sobre elas devem ser armazenados, quem deve ter acesso a esses dados e quais as restrições de integridade que são aplicáveis. Por sua vez, o DBA possui o conhecimento técnico necessário para implementar essas definições em um SGBD.

Agora podemos definir mais claramente as atribuições do DBA:

1. **Definir o esquema conceitual:** Uma vez que o administrador de dados tenha definido o projeto lógico, contendo as definições conceituais do banco de dados como um todo, o DBA criará o esquema conceitual (utilizado na definição da visão conceitual).
2. **Definir o esquema interno:** Com a visão conceitual pronta, cabe ao DBA determinar como esses dados devem ser armazenados de forma a obter o melhor desempenho possível. Qual o melhor formato para cada campo? Qual o melhor índice para tratar os dados? Respostas a essas e a outras perguntas compõem o que chamamos de Projeto Físico do Banco de Dados (não confundir com Nível Físico!). Pronto o projeto físico, o DBA criará o esquema interno (que servirá de base para a visão interna) e o mapeamento conceitual/interno.
3. **Ligação com os usuários:** É papel do DBA prover aos usuários a infra-estrutura para que possam explorar o banco de dados. Para isso, o DBA é responsável pela definição (e implementação) das diversas visões externas, seus esquemas externos e os respectivos mapeamentos externos/conceituais. De forma similar, o DBA deve dar consultoria e suporte no desenvolvimento de aplicativos e na resolução de problemas relacionados ao banco de dados.
4. **Definir a política de descarga e recarga:** Devido à importância dos dados para a empresa e ao fato do DBA ser o responsável técnico pelo banco de dados, é sua

responsabilidade definir a política de cópias de segurança periódicas do banco de dados (descarga), seu armazenamento e eventual restauração (recarga) em caso de necessidade, da forma mais rápida e transparente possível aos usuários.

5. **Monitorar o desempenho do banco de dados e responder a requisitos de mudanças:** Como vimos anteriormente, cabe ao DBA manter o banco de dados com o melhor desempenho possível. É sua atribuição configurar o banco de dados de forma a otimizar esse desempenho. Podem ser necessárias reorganizações periódicas no banco de dados ou mesmo alterações em seu nível interno (e as conseqüentes alterações no mapeamento conceitual/interno) para adequá-lo a novas necessidades ou tecnologias. É responsabilidade do DBA coordenar essas paradas de manutenção.

## Funções de um Sistema Gerenciador de Banco de Dados

Agora podemos estimar com mais precisão o volume de trabalho desempenhado pelo sistema gerenciador de banco de dados.

Uma consulta simples que apenas identifique e exiba alguns registros em uma visão externa seria trabalhada em várias etapas:

- Primeiro a solicitação do usuário seria interpretada e analisada.
- Seria verificado então o esquema para identificar quais os campos necessários e os seus respectivos tipos de dados.
- Os campos conceituais envolvidos seriam identificados no mapeamento externo/conceitual correspondente.
- O processo continuaria da mesma forma, passando pelo esquema conceitual e o mapeamento conceitual/interno, até chegar ao esquema interno.
- Os registros internos necessários seriam então selecionados pelo SGBD e o processo seria reiniciado no sentido inverso efetuando todas as operações e conversões de tipo necessárias para devolver ao usuário, segundo as especificações da visão externa, os registros desejados no formato esperado.

Como dissemos antes, o SGBD serve de interface para o usuário, realizando todo o trabalho de manutenção do banco de dados de forma transparente. Dessa forma podemos considerar como funções de um SGBD:

1. Aceitar e “compreender” as diversas definições de dados (esquemas externos, conceitual e interno e os mapeamentos associados) de forma a realizar as conversões necessárias. Para isso é necessário um **interpretador/compilador de DDL**.

2. “Compreender” as solicitações de manipulação de dados dos usuários (inclusões, alterações de dados, exclusões e pesquisas). Para isso é necessário um **interpretador/compilador de DML**.
3. Otimizar as requisições de DML de forma a obter o melhor desempenho possível durante sua execução. Vale observar que existem dois tipos de solicitações DML:
  - 3.1. **Planejadas** – Solicitações rotineiras na empresa, normalmente em nível operacional, repetidas diversas vezes sem alterações (muitas vezes programadas). São previstas pelo DBA que ajustará o banco de dados (no nível interno) de forma serem executadas eficientemente.
  - 3.2. **Não Planejadas** – Solicitações que não são rotineiras na empresa, normalmente solicitações de pesquisa utilizadas como apoio ao processo de tomada de decisão. Por serem imprevisíveis pelo DBA, serão executadas da melhor maneira **possível** (não necessariamente a mais eficiente) pelo SGBD, de acordo com os projetos físico e lógico do banco de dados.
4. Como dito anteriormente, o SGBD deve supervisionar todo e qualquer acesso aos dados de forma a garantir que as restrições de integridade e de segurança estão sendo obedecidas.
5. Prover meios para **recuperação de transações**, isto é, no caso de impossibilidade de conclusão bem sucedida de uma transação, retornar o banco de dados ao estado anterior. Isso será mais explicado ao estudarmos transações.
6. Realizar o **controle de concorrência**, ou seja, gerenciar o acesso simultâneo de mais de um usuário aos mesmos dados. O objetivo é, quando num acesso simultâneo, um usuário alterar os dados armazenados, garantir que todos os demais usuários tenham acesso imediato às informações atualizadas. Isso evita a situação em que dois usuários acessam os mesmos dados num estado “A” e o primeiro deles altera os dados para um estado “B”, enquanto o segundo usuário, trabalhando agora com um estado “A” não correspondente à realidade, os altera para um estado “C”, comprometendo a integridade dos dados.
7. Disponibilizar um **Dicionário de Dados**. Nele estão armazenados **descritores** (ou **metadados**), que são informações necessárias ao funcionamento do banco de dados, como as definições de todos os tipos de dados utilizados, os mapeamentos e os esquemas (interno, conceitual e externo) e as restrições de segurança e integridade. Poderá ainda incluir informações como: os usuários e aplicações que acessam o

sistema, suas consultas customizadas e atividades pré-definidas de execução rotineira e automática.

Os “sistemas gerenciadores de bancos de dados” utilizados antigamente estavam mais próximos na verdade de **Sistemas Gerenciadores de Arquivos**. Ou seja, eles apenas administravam arquivos armazenados onde podiam realizar operações simples de busca e atualização. No entanto, comparando com os SGBD:

1. Não possuíam conhecimento interno da estrutura dos arquivos armazenados, por isso não tinham as mesmas conversões de tipo e independência (lógica e física) de dados que o SGBD.
2. Ofereciam pouco (ou nenhum) suporte a restrições de segurança e integridade.
3. Não ofereciam um dicionário de dados.
4. Não ofereciam integração e compartilhamento de arquivos. Cada arquivo era privativo do usuário ou da aplicação que o utilizava (daí a redundância de dados nos “bancos de dados” espalhados pelos diversos departamentos).
5. Não ofereciam suporte a transações. Se, durante uma atualização em várias etapas, uma dessas etapas não tivesse sucesso, teríamos que desfazer, manualmente cada uma das etapas já concluídas de forma a retornar o banco de dados à situação anterior.

## Arquitetura Cliente/Servidor

Uma vez que os sistemas de banco de dados existem basicamente para fornecer a infraestrutura necessária ao desenvolvimento e execução de aplicações de banco de dados, podemos analisá-la por uma divisão lógica de responsabilidades mais simples que a arquitetura ANSI/SPARC, possuindo apenas dois níveis:

O primeiro nível, composto pelas aplicações que acessarão o banco de dados, é chamado de **cliente**.

O segundo nível, composto pelo próprio SGBD, é chamado de **servidor**, pois é responsável por prover (servir) os clientes com a infra-estrutura necessária ao seu funcionamento (as funções de um SGBD como definição e manipulação de dados, controle de concorrência e restrições de segurança e integridade).

O nome dessa arquitetura é “**Arquitetura Cliente/Servidor**” e permite uma separação clara entre as duas principais partes de um sistema de banco de dados (clientes e servidores). Nessa arquitetura, é indiferente ao servidor a localização dos clientes, que podem ser executados tanto na mesma máquina que ele, quanto em máquinas distintas. Isso permite o

desenvolvimento do sistema de forma a executar cada parte em uma máquina diferente. Com isso, podemos somar às vantagens trazidas pelo SGBD, o processamento distribuído (como veremos logo adiante). O processamento é dividido entre o cliente e o servidor, diminuindo a carga e o tráfego de informações entre eles.

Do ponto de vista do servidor, os clientes são quaisquer aplicações que utilizem os seus serviços, podendo ser:

- Aplicações desenvolvidas pelos programadores de aplicações em linguagens de alto nível acopladas a uma sublinguagem de dados.
- Ferramentas, muitas vezes fornecidas pelo próprio fabricante do SGBD, para que o próprio usuário final possa interagir com o banco de dados sem necessidade de desenvolvimento elaborado, como geradores de relatórios, ferramentas case e de “*data ware house*”.
- Utilitários, obrigatoriamente fornecidos pelo fabricante do SGBD, que através de acesso de baixo nível (normalmente direto ao nível interno) realizam atividades de manutenção no banco de dados. Alguns exemplos dessas atividades são: rotinas de reorganização, carga inicial de dados, descarga e recarga, levantamento de estatísticas e otimização.

A quantidade (e qualidade) dos utilitários assim como das ferramentas para o usuário final, varia para cada SGBD. Essa avaliação, principalmente em relação às ferramentas para o usuário final, tem grande relevância no processo de escolha do SGBD a ser adotado.

## Processamento Distribuído

É o nome dado à conexão simultânea de mais uma máquina de forma que o processamento possa ser distribuído entre elas, melhorando o desempenho.

Existe também a expressão “**processamento paralelo**” que descreve uma situação bastante semelhante. No processamento paralelo a idéia central é desmembrar um problema grande em problemas menores que possam ser resolvidos simultaneamente. Várias máquinas **conjugadas** trabalhando de forma a simular o funcionamento de uma máquina de maior capacidade (*clustering*). Já no processamento distribuído, o objetivo é distribuir tarefas relacionadas entre máquinas **independentes** de forma a reduzir sobrecarga e gargalos.

No processamento distribuído apesar de serem interligadas, as tarefas têm sua execução independente. Dessa forma, enquanto no processamento paralelo as máquinas tendem a

manter proximidade física, no processamento distribuído isso não é imprescindível. Para nosso estudo de bancos de dados, vamos nos ater ao processamento distribuído.

Há muitas variedades de processamento distribuído em um sistema de banco de dados. A mais simples é quando o servidor (*back end*) é executado em uma máquina enquanto o cliente (*front end*) é executado em outra. De fato, o termo “cliente/servidor” deixou de ser relacionado estritamente à arquitetura para ser quase um sinônimo dessa variedade de processamento distribuído.

Algumas vantagens dessa variedade:

1. Aumento de desempenho já que as tarefas relacionadas ao cliente e o processamento no servidor são executados simultaneamente.
2. A máquina servidora pode ter uma configuração mais adequada, muitas vezes multiprocessada, com grande capacidade de memória e componentes redundantes, oferecendo melhor desempenho e segurança.
3. A máquina cliente também pode ser mais adequada, muitas vezes com a configuração mínima para executar suas tarefas, diminuindo os custos.
4. Várias máquinas clientes podem acessar o mesmo servidor simultaneamente compartilhando o banco de dados. Esse recurso é muito utilizado uma vez que um dos principais objetivos de um SGBD é compartilhar o banco de dados.
5. Uma máquina cliente pode acessar dados de vários servidores, possibilitando que os dados da empresa estejam distribuídos em mais de um servidor.

Esse último recurso é muito desejado uma vez que, normalmente as empresas têm os seus dados espalhados por muitas máquinas ao invés de ficarem concentrados num único servidor.

Esse acesso múltiplo pode ser feito de duas formas:

- O cliente só pode acessar um servidor de cada vez. Dessa forma não é possível combinar dados de mais de um servidor numa única operação e o usuário precisa conhecer a localização dos dados nos diversos servidores.
- O cliente pode acessar vários servidores simultaneamente, combinando dados de mais de um servidor numa única operação. Do ponto de vista do cliente, os servidores aparentam ser um único, não importando a localização dos dados.

A segunda forma caracteriza um **Sistema de Banco de Dados Distribuído**. Um sistema de banco de dados distribuído implica que um cliente deve conseguir acesso a diversos bancos de dados distintos, gerenciados por gerenciadores diferentes, em máquinas distintas, funcionando sob sistemas operacionais diversos, comunicando-se por meio de redes de computadores diferentes, de forma transparente.

Do ponto de vista do cliente, é como se todos os dados se originassem de um mesmo SGBD rodando em uma única máquina. Ou seja, **um sistema de banco de dados distribuído é um ótimo exemplo de independência lógica.**

Atualmente, essa é uma das tecnologias de banco de dados que mais recebe investimentos dos fabricantes de SGBD.

# Bancos de Dados Relacionais

Já vimos sobre os bancos de dados relacionais e algumas de suas características principais. Uma vez que ele é o único modelo de dados real e (principalmente) devido à sua importância comercial, concentraremos nossos estudos nesse modelo.

O modelo relacional foi publicado em 1969 por Edgar F. Codd, um matemático da IBM. O trabalho de Codd começou a servir de base para o trabalho de outros pesquisadores. Um grupo, em Berkeley, desenvolveu o SGBD Ingres enquanto outro grupo, na IBM, desenvolveu a primeira versão da linguagem SQL.

À medida que o tempo passou, o modelo relacional foi conquistando mais espaço no mercado e sendo aprimorado, a tal ponto que certas características ainda não foram totalmente implementadas.

Os Sistemas Gerenciadores de Banco de Dados Relacionais (SGBDR) aumentaram em quantidade. Existe hoje, desde opções de baixo custo até opções extremamente custosas e abrangentes. Mesmo assim, todas oferecem um alto nível de qualidade e profissionalismo. Escolher um SGBDR, hoje, é selecionar qual melhor se adapta (e atende) às necessidades do sistema de banco de dados.

## Relações

O modelo relacional é claramente baseado no conceito de matrizes, onde as linhas das matrizes seriam os registros e as colunas (das matrizes) seriam os campos. Entretanto, essa terminologia não é adequada a expressar os conceitos do modelo relacional.

Descreveremos agora os termos utilizados pelo modelo relacional, para chegarmos à definição de relações.

Como vimos anteriormente, **atributos são os elementos de dados que, juntos, descrevem uma entidade**. Considerando a entidade aluno, o nome, sexo e data de nascimento seriam atributos.

Cada atributo tem um conjunto de valores possíveis para ele, por exemplo, só há dois valores possíveis para sexo: masculino e feminino. **Domínio é o conjunto de valores possíveis para um atributo**.

Cada ocorrência da entidade representada pela relação (no exemplo, cada aluno) possui um conjunto de valores que o descrevem. **Tupla é o conjunto de valores que, juntos, descrevem um elemento (uma instância) da entidade**.

O modelo relacional tem esse nome porque representa os dados contidos em um banco de dados através de relações que contém informações sobre as entidades representadas e as interações entre elas (relacionamentos). Mas o que são relações?

Informalmente, **uma relação é o conjunto de tuplas que descreve todas as ocorrências, representadas no banco de dados, de uma determinada entidade ou evento do mundo real.**

Entretanto devemos observar que, segundo a terminologia aplicada ao Modelo Relacional, os valores existentes em uma relação num determinado momento são chamados de **Valores de Relação**, e as variáveis que os contém de **Variáveis de Relação**.

É muito comum utilizarmos o termo “relação” quando, na verdade, nos referimos a uma variável de relação. Em situações menos formais a troca não é tão problemática, mas como essa prática tende a causar uma certa confusão, é bom observarmos com cuidado o uso desses termos.

## Visões

Uma visão é um tipo especial de variável de relação, que é derivada da avaliação de uma expressão relacional aplicada sobre outras variáveis de relação nesse instante.

A definição de uma visão é armazenada no dicionário de dados e é percebida por um usuário como uma variável de relação igual às demais. Dizemos que as visões são **variáveis de relações virtuais** enquanto chamamos as variáveis de relação propriamente ditas de **variáveis de relações reais**.

As visões não são fisicamente armazenadas, portanto não são cópias dos dados das variáveis de relações reais. Qualquer alteração nos dados contidos numa visão será refletida instantaneamente nas variáveis de relações subjacentes.

As visões são a forma, prevista no modelo relacional, de proporcionar a independência de dados lógica, entretanto, não devemos confundir as visões do modelo relacional com as visões externas vistas na arquitetura ANSI/SPARC. O equivalente, no modelo relacional, às visões externas ANSI/SPARC seria o conjunto de diversas variáveis de relações (tanto reais quanto virtuais) e as suas definições formariam o “esquema externo”.

## Definição formal de uma relação

Uma relação é dividida em duas partes, uma parte abstrata que descreve os domínios que a compõem, chamada cabeçalho. E outra parte chamada corpo, concreta, com os dados que representam cada uma das instâncias da entidade descrita pela relação.

*Atenção! Apesar da semelhança com uma tabela (linhas, colunas, cabeçalho e corpo), uma **relação não é uma tabela**. Uma relação é uma definição puramente conceitual, não tendo uma representação física exata. Sua representação mais próxima no mundo real é uma tabela. Na verdade, uma tabela a é representação mais próxima de uma variável de relação. Por essa razão, os SGBDR implementam relações por meio de tabelas e, neste trabalho, quando precisarmos representar uma variável de relação, usaremos o mesmo artifício.*

Considere, como exemplo, a tabela (contendo a representação de algumas tuplas) abaixo:

CIDADE	UF	TEMP_MIN	TEMP_MAX
Salvador	BA	22,5	29,5
Vitória da Conquista	BA	18,2	27,3
Rio de Janeiro	RJ	20,8	43,0
Gramado	RS	-2,0	26,7
Pelotas	RS	-0,5	32,0

Dados fictícios

No cabeçalho estão descritos os atributos que compõem a relação (CIDADE, UF, TEMP\_MIN e TEMP\_MAX).

Essa relação apresenta quatro atributos, porém dois deles (TEMP\_MIN e TEMP\_MAX) compartilham o mesmo domínio (temperaturas de -30° a 60° Celsius). Um atributo precisa ser único numa relação, um domínio não. O número de atributos numa relação é chamado de **cardinalidade da relação**. Essa é uma relação de cardinalidade quatro.

O corpo da relação é composto pelas tuplas que compõem a relação.

A relação do nosso exemplo possui cinco tuplas. O número de tuplas em uma relação determina o **grau da relação**. Essa é uma relação de grau cinco.

Observemos agora uma definição mais formal do termo “relação” (C. J. Date):

“Dada uma coleção de  $n$  tipos ou domínios  $T_i$  ( $i = 1, 2, \dots, n$ ), não necessariamente todos distintos,  $r$  será uma **relação** sobre esses tipos se consistir em duas partes, um *cabeçalho* e um *corpo*, onde:

- O **cabeçalho** é um conjunto de  $n$  **atributos** da forma  $A_i:T_i$ , onde  $A_i$  (que devem ser todos distintos) são *nomes de atributos* de  $r$  e  $T_i$  são os *nomes de tipos* correspondentes ( $i = 1, 2, \dots, n$ ).

b. O **corpo** é um conjunto de  $m$  **tuplas**  $t$ , onde  $t$  é por sua vez um conjunto de componentes da forma  $A_i:vi$  no qual  $vi$  é um valor do tipo  $T_i$  – o *valor do atributo* correspondente ao atributo  $A_i$  da tupla  $t$  ( $i = 1, 2, \dots, n$ ).

Os valores  $m$  e  $n$  são chamados respectivamente **cardinalidade** e **grau** da relação  $r$ .”

## Propriedades de uma relação

Para que consideremos que um conjunto de atributos é uma relação, de acordo com o modelo relacional, este conjunto precisa apresentar algumas propriedades específicas:

1. As tuplas devem ser únicas – O corpo da relação é um conjunto de tuplas e os conjuntos (em matemática) não possuem itens duplicados.
2. As tuplas não são ordenadas – Também derivado do fato do corpo da relação ser um conjunto; os conjuntos (em matemática) não são ordenados.
3. Os atributos não são ordenados – De modo similar à propriedade anterior, não há obrigatoriedade que os atributos tenham alguma ordem específica.
4. Cada tupla contém apenas um valor para cada atributo – De acordo com a própria definição formal, um tupla é um conjunto de pares ordenados na forma  $A_i:vi$  ( $i = 1, 2, \dots, n$ ) não admitindo mais de um valor por atributo. Quando uma relação apresenta essa propriedade, dizemos que está na **primeira forma normal**. Veremos mais detalhes quando tratarmos de normalização.

## Chaves

Por definição, o modelo relacional não permite a existência de tuplas duplicadas, ou seja, tendo todos os elementos de uma tupla, podemos referenciá-la unicamente em uma variável de relação. Acontece que, normalmente, não sabemos todos os valores de uma tupla, conhecemos apenas alguns e desejamos obter os demais. Para atender a essa necessidade, o modelo relacional oferece o instrumento das chaves.

Seja uma variável de relação  $R$ . Na prática, é freqüente que um subconjunto  $K$  dos atributos de  $R$  tenha o caráter de unicidade. Dizemos que  $K$  é uma **Chave Candidata** de  $R$  se apresentar ambas as propriedades a seguir:

1. **Unicidade** – Não existe duas tuplas de  $R$  com o mesmo valor de  $K$ .
2. **Irreduzibilidade** – Nenhum subconjunto de  $K$  tem a propriedade de unicidade.

Dessa forma, toda variável de relação possui pelo menos UMA chave candidata. Dentre o conjunto de todas as chaves candidatas de uma variável de relação, o modelo relacional escolhe uma para ser eleita **Chave Primária**, aquela utilizada como item identificador das

tuplas. As demais chaves candidatas recebem o nome de **Chaves Alternativas**. Elas também identificam unicamente cada uma das tuplas e podem ser utilizadas em alguma circunstância especial para isto.

Já vimos que o modelo relacional não utiliza ponteiros para conectar duas relações entre si. Essa ligação é realizada pelo aparecimento, nos atributos de uma relação, de um elemento de ligação comum a ambas.

Sejam duas variáveis de relações R1 e R2. Para que a variável de relação R1 seja conectada à variável de relação R2, é necessário que um dos atributos de R2 referencie um atributo de R1, e que esse atributo seja uma Chave Candidata de R1.

Dessa forma, uma **Chave Estrangeira** é um conjunto de atributos de uma variável de relação, cujos valores correspondem obrigatoriamente aos valores de uma chave candidata de outra variável de relação conectada a ela.

## Normalização

Normalização é o processo ao qual devemos submeter uma relação para que possamos evitar redundâncias e tornar o projeto do nosso banco de dados mais eficiente.

## Dependência Funcional

Para que possamos compreender o processo de normalização de uma relação, precisamos do conceito de Dependência Funcional. Dizemos que um atributo A é dependente funcionalmente de um atributo B, quando o valor de A é determinado pelo valor de B.

Considere a variável de relação representada a seguir:

Gênero	Sigla	Título	Atores Principais	Emissora	Canal
Ficção	TOS	Star Trek	William Shatner Leonard Nimoy DeForest Kelley	USA	105
Ficção	TNG	Star Trek: A Nova Geração	Patrick Stewart Brent Spinner	USA	105
Ficção	DS9	Star Trek: Deep Space 9	Avery Brooks Nana Visitor Rene Auberjonois	USA	105
Ficção	VOY	Star Trek: Voyager	Kate Mulgrew Robert Picardo	USA	105
Ficção	ENT	Enterprise	Scott Bakula Jolene Blalock Connor Trinneer	AXN	103

Independente da existência da chave primária (Estilo–Sigla, as colunas sombreadas), o canal é determinado pela emissora de cada série. Dizemos portanto, que o atributo Canal **depende funcionalmente** do atributo Emissora.

## Formas Normais

Vimos que uma das propriedades de uma relação é estar na “primeira forma normal”, o estágio inicial de normalização. Entretanto, existem outros níveis de normalização. Quanto mais alto o nível, mais próximo da forma relacional pura estará o banco de dados. Estudiosos já identificaram diversos níveis de normalização (ou formas normais), entretanto os três primeiros são os mais utilizados, e bastam para o nosso estudo.

### Primeira Forma Normal (1FN)

Uma variável de relação estará na 1FN se e somente se, em todo valor válido dessa variável de relação, cada tupla contiver somente um valor para cada atributo. Ou seja, um atributo não poderá conter uma estrutura ou uma outra relação (valores múltiplos).

No nosso exemplo, o atributo Atores Principais apresenta mais de um valor para cada tupla. Para que a variável de relação esteja na 1FN, devemos criar uma segunda variável de relação contendo apenas os nomes dos autores e conectada à primeira por meio de uma relação “um-para-muitos”.

Sigla	Ator	Nome
TOS	001	William Shatner
TOS	002	Leonard Nimoy
TOS	003	DeForest Kelley
TNG	001	Patrick Stewart
TNG	002	Brent Spinner
DS9	001	Avery Brooks
DS9	002	Nana Visitor
DS9	003	Rene Auberjonois
VOY	001	Kate Mulgrew
VOY	002	Robert Picardo
ENT	001	Scott Bakula
ENT	002	Jolene Blalock
ENT	003	Connor Trinneer

### Segunda Forma Normal (2FN)

Uma variável de relação estará na 2FN se e somente se, ela estiver na 1FN e todo atributo não-chave for irreduzivelmente dependente da chave primária. Ou seja, caso a chave primária seja composta por mais de um atributo, nenhum dos atributos não-chave poderá depender de parte da chave primária.

Ainda no nosso exemplo, a chave primária é composta pelos atributos Gênero e Sigla, entretanto, os demais atributos dependem funcionalmente apenas do atributo Sigla. Para que a variável de relação esteja na 2FN, basta retirarmos o atributo Gênero da chave primária

deixando-o como um atributo não-chave normal. A chave primária passaria a ser apenas o atributo Sigla.

### Terceira Forma Normal (3FN)

Uma variável de relação estará na 3FN se e somente se, ela estiver na 2FN e todo atributo não-chave for dependente exclusivamente da chave primária. Ou seja, nenhum atributo poderá ser funcionalmente dependente de outro atributo que não pertença à chave primária. Vale ressaltar que, no caso de existirem chaves candidatas que não sejam a chave primária essa forma normal não é muito adequada, devendo ser substituída pela Forma Normal de Boyce-Codd (BCFN):

Uma variável de relação estará na BCFN se e somente se, os únicos determinantes são chaves candidatas.

Voltando ao nosso exemplo, o atributo Canal não depende da chave primária, e sim do atributo Emissora.. Para que a variável de relação esteja na 3FN, devemos criar uma terceira variável de relação contendo apenas os canais e emissoras, e conectada à primeira por meio de uma relação “um-para-muitos”.

<b>Emissora</b>	<b>Nome</b>	<b>Canal</b>
001	USA	105
002	AXN	103

### Regras práticas para normalização

Podemos resumir o processo de normalização na aplicação de algumas regras práticas que garantirão que o banco de dados atenderá às condições das 3 formas normais:

- Certifique-se que todos os atributos na variável de relação estejam relacionados entre si.
- Evite a redundância, dentro de cada tupla e dentro da variável de relação.
- Assegure-se que todo atributo em uma variável de relação dependa funcionalmente da chave primária e apenas dela.

Altere o formato das variáveis de relação desmembrando tuplas e variáveis de relação sempre que preciso.

## O Dicionário de Dados

Uma das funções do SGBD é fornecer o dicionário de dados. O dicionário de dados contém as definições de todos os objetos contidos no banco de dados, como os esquemas (interno, conceitual e os vários externos), o mapeamento externo/conceitual e o mapeamento conceitual/interno, as diversas variáveis de relações e seus índices, as relações de integridade e segurança e os usuários e suas permissões.

Essas informações são utilizadas pelo SGBD para realizar suas funções. Quando um usuário faz uma requisição, por exemplo, o SGBD utiliza as informações de usuários e restrições de segurança para permitir o acesso e, uma vez aceita a requisição, as definições dos índices e estruturas de armazenamento são utilizadas para otimizar requisição.

Uma vez que o Modelo Relacional é baseado na teoria de conjuntos, a maioria das operações realizadas nesse modelo são **não-procedurais** (ou **não-procedimentais**), ou seja, especifica-se o “que” e não o “como”. Dessa forma, especificamos os dados que desejamos, e cabe ao SGBD decidir a melhor maneira de obtê-los. A própria navegação entre os dados é transparente para o usuário. Dessa forma, o dicionário de dados é uma fonte de informações fundamental para que o SGBD realize essas tarefas da forma mais eficiente. Quando estudarmos sobre otimização, veremos mais detalhes sobre isso.

## O Símbolo NULL

Existem situações em que é necessário representar a ausência de um valor num banco de dados. Muitas vezes porque não conhecemos o valor para informá-lo (qual o tipo sanguíneo do professor Fulano?), outras porque o valor não se aplica (qual o número de filhos de uma criança recém-nascido?).

Para esses casos, o modelo relacional oferece a noção de NULL. NULL não é exatamente um valor, pode ser considerado um símbolo que representa o vazio. A tradução para o português seria “nulo”, entretanto a mais adequada às nossas necessidades seria “desconhecido”.

O modelo relacional determina que NULL não coincidirá com nenhum valor, nem mesmo com NULL. Dessa forma, não pode ser comparado a nada, qualquer comparação devolve um valor NULL como resposta.

Como tudo, NULL tem suas vantagens e desvantagens. A principal vantagem é que permite a representação da inexistência de valores, facilitando a criação de restrições que envolvam essa situação. Sua principal desvantagem é o aumento da complexidade na lógica de qualquer

desenvolvimento que utilize SQL , uma vez que passam a existir três resultados para a maioria dos testes lógicos (mais detalhes em Operadores Lógicos).

Existe muita controvérsia sobre a utilização do NULL, autores de renome como C. J. Date e o próprio E. F. Codd ainda não chegaram a um acordo sobre esse valor especial, discutindo sobre a validade de utilizá-lo no modelo relacional. Não temos a intenção de resolver essa questão neste trabalho. Apenas reconheceremos que esse símbolo existe (é implementado em todos os SGBDR) e tem várias utilidades.

Com os devidos cuidados quanto aos testes lógicos, é uma ferramenta muito útil. E como acontece com qualquer ferramenta, a decisão de utilizá-lo é prerrogativa do desenvolvedor (administrador de dados e DBA).

## Integridade

A integridade de um banco de dados refere-se à precisão ou correção dos dados nele armazenados. Garantimos essa integridade por meio de Restrições de Integridade que são armazenadas no dicionário de dados. O SGBD encarrega-se então de, a cada operação que acarrete alteração nos dados, verificar se a alteração solicitada viola as restrições de integridade. Em caso afirmativo, a operação é interrompida.

As restrições de integridade características da empresa proprietária do banco de dados são normalmente chamadas de Regras de Negócio, pois descrevem o funcionamento do negócio da empresa. São definidas pelo Administrador de Dados e evitam o armazenamento de dados em desacordo com a política da empresa. Por exemplo, se uma política da empresa determina que o desconto comercial máximo concedido a um cliente é de dez por cento, uma operação de atualização no banco de dados que informe um desconto de vinte por cento será interrompida.

Além das regras de negócio, existem restrições que visam a manter a integridade dos dados do banco de dados, independente da política da empresa. Podemos classificar essas restrições de integridade em restrições de: tipos de dados, integridade de entidade e integridade referencial.

## Restrições de Tipos de Dados

Evitam que dados incompatíveis com os tipos de dados selecionados para seus atributos (como datas inválidas) sejam aceitos pelo banco de dados.

## Restrições de Integridade de Entidade

Garantem que as chaves primárias tenham valores válidos, uma vez que o objetivo de uma chave primária é a identificação de uma tupla.

## Restrições de Integridade Referencial

Garantem a integridade das relações entre duas ou mais variáveis de relações, devendo uma chave estrangeira apontar para uma tupla existente.

Quando uma variável de relação está conectada a outra por meio de uma chave estrangeira, a inclusão (ou alteração) de dados numa tupla deve ser supervisionada para evitar que a chave estrangeira armazene dados inválidos.

Durante a exclusão de uma tupla apontada pelas chaves estrangeiras de tuplas de outras variáveis de relações, podem ser utilizadas três estratégias para manter a integridade:

- A operação será interrompida.
- As tuplas relacionadas serão excluídas em cascata. Essa estratégia deve ser utilizada com cuidado por razões óbvias, afinal ninguém quer dados sendo excluídos em seqüência do banco de dados.
- Os valores das chaves estrangeiras nas tuplas relacionadas serão alterados para apontarem para uma outra tupla ou para um valor padrão (podendo inclusive ser o valor NULL).

# Linguagem de Acesso Estruturada (SQL)

A especificação de E. F. Codd para o Modelo Relacional determinava que os usuários teriam acesso ao banco de dados através de uma linguagem de consulta e que ela seria a **única** forma de acesso aos dados.

Em 1974, cinco anos após a publicação do Modelo Relacional, foi iniciado na IBM o desenvolvimento do System/R, o primeiro protótipo de um SGBDR. Para o acesso, foi criada uma linguagem denominada SEQUEL. Mais tarde seu nome foi alterado para *Structured Query Language* (Linguagem de Consulta Estruturada) ou SQL. Atualmente a palavra “SQL” deixou de ser um acrônimo para ser o próprio nome da linguagem.

Na década de 80, a linguagem SQL foi escolhida pelo *American National Standards Institute* (ANSI) e pouco depois, pelo *International Standards Institute* (ISO), como a linguagem de consulta padrão para bancos de dados relacionais.

O primeiro padrão, publicado em 1989 pelas duas organizações, ficou conhecido como SQL1 ou SQL-89. Em 1992, esse padrão foi revisado e é o atual, conhecido como SQL/92, SQL-92 ou SQL2. Seu nome oficial agora é *International Standard Database Language SQL (1992)*. É esse padrão que veremos neste trabalho.

Um novo padrão (SQL3 ou SQL:1999) foi desenvolvido para ser uma “SQL orientada a objetos”, para atender ao acesso a bancos de dados orientados a objeto. Poucos SGBD implementam algumas de suas extensões (Oracle9i), e ainda não há nenhum que consiga implementar todas.

## Características da SQL

É uma linguagem simples e fácil de aprender. Os comandos são escritos quase em linguagem natural (inglês) permitindo que mesmo aqueles que não são programadores possam escrever suas consultas relativamente rápido. Não é necessário conhecer lógica de programação. As instruções são executadas separadamente, dessa forma, aprender SQL é aprender a sintaxe de cada instrução SQL e o que ela faz.

Por isso e por ter sido desenhada originalmente para ser uma sublinguagem de dados, a SQL não contém instruções de controle de fluxo, laços ou de redirecionamento de entrada e saída. O objetivo inicial era que a SQL dependesse de uma linguagem hospedeira que oferecesse tais recursos.

Com a introdução, em 1996, do recurso de Módulos Armazenados Persistentes (*Persistent Data Modules* – PSM) a SQL tornou-se mais poderosa. No entanto, o nosso estudo será concentrado no padrão SQL/92.

A SQL não adota, rigorosamente, o modelo relacional. Entre as diferenças podemos observar:

- O termo “tabela” substitui os termos “variável de relação” e “relação”.
- Os termos “coluna” e “campo” substituem o termo “atributo”.
- Os termos “linha” e “registro” substitui o termo “tupla”.
- Podem existir linhas duplicadas, enquanto no modelo relacional as tuplas devem ser únicas.
- A SQL não possui um número ilimitado de tipos de dados, ao contrário do modelo relacional.

Apesar disso, a SQL implementa o modelo relacional de maneira muito confiável.

A SQL é uma linguagem muito grande. O documento padrão da SQL/92 (ISO 9075) possui mais de 600 páginas. Faremos, portanto, um estudo superficial da linguagem, enfocando os pontos mais utilizados. Será o suficiente, porém, para executar as tarefas mais exigidas na criação e acesso a um banco de dados relacional.

## Estrutura da SQL

A SQL não é *case-sensitive*, ou seja, não diferencia letras maiúsculas ou minúsculas. Isso quer dizer que “SELECT” e “SeLeCt” têm o mesmo significado. Tradicionalmente escrevemos as palavras reservadas em maiúsculas e os demais nomes em minúsculas (exceto as constantes), mas isso não é obrigatório.

Utilizamos o caractere ponto-e-vírgula (“;”) como indicador de término de instrução.

Os nomes de campos, tabelas, visões e demais objetos devem conter apenas letras, números e alguns caracteres especiais, em particular o traço (“-”) e o sublinhado (“\_”). Espaços não são permitidos.

## Palavras Reservadas

A SQL/92 possui mais de 300 palavras reservadas (palavras que não podem ser utilizadas como nomes de campos ou de variáveis). Desse grande número, apenas algumas são utilizadas com mais frequência:

Palavras reservadas – SQL/92				
ABSOLUTE	ACTION	ADA	ADD	ALL
ALLOCATE	ALTER	AND	ANY	ARE
AS	ASC	ASSERTION	AT	AUTHORIZATION
AVG	BEGIN	BETWEEN	BIT	BIT_LENGTH
BOTH	BY	CASCADE	CASCADDED	CASE
CAST	CATALOG	CHAR	CHAR_LENGTH	CHARACTER
CHARACTER_LENGTH	CHECK	CLOSE	COALESCE	COLLATE
COLLATION	COLUMN	COMMIT	CONNECT	CONNECTION
CONSTRAINT	CONSTRAINTS	CONTINUE	CONVERT	CORRESPONDING
COUNT	CREATE	CROSS	CURRENT	CURRENT_DATE
CURRENT_TIME	CURRENT_TIMESTAMP	CURRENT_USER	CURSOR	DATE
DAY	DEALLOCATE	DEC	DECIMAL	DECLARE
DEFAULT	DEFERRABLE	DEFERRED	DELETE	DESC
DESCRIBE	DESCRIPTOR	DIAGNOSTICS	DISCONNECT	DISTINCT
DOMAIN	DOUBLE	DROP	ELSE	END
END-EXEC	ESCAPE	EXCEPT	EXCEPTION	EXEC
EXECUTE	EXISTS	EXTERNAL	EXTRACT	FALSE
FETCH	FIRST	FLOAT	FOR	FOREIGN
FORTRAN	FOUND	FROM	FULL	GET
GLOBAL	GO	GOTO	GRANT	GROUP
HAVING	HOURL	IDENTITY	IMMEDIATE	IN
INCLUDE	INDEX	INDICATOR	INITIALLY	INNER
INPUT	INSENSITIVE	INSERT	INT	INTEGER
INTERSECT	INTERVAL	INTO	IS	ISOLATION
JOIN	KEY	LANGUAGE	LAST	LEADING
LEFT	LEVEL	LIKE	LOCAL	LOWER
MATCH	MAX	MIN	MINUTE	MODULE
MONTH	NAMES	NATIONAL	NATURAL	NCHAR
NEXT	NO	NONE	NOT	NULL
NULLIF	NUMERIC	OCTET_LENGTH	OF	ON
ONLY	OPEN	OPTION	OR	ORDER
OUTER	OUTPUT	OVERLAPS	PAD	PARTIAL
PASCAL	POSITION	PRECISION	PREPARE	PRESERVE
PRIMARY	PRIOR	PRIVILEGES	PROCEDURE	PUBLIC
READ	REAL	REFERENCES	RELATIVE	RESTRICT
REVOKE	RIGHT	ROLLBACK	ROWS	SCHEMA
SCROLL	SECOND	SECTION	SELECT	SESSION
SESSION_USER	SET	SIZE	SMALLINT	SOME
SPACE	SQL	SQLCA	SQLCODE	SQLERROR
SQLSTATE	SQLWARNING	SUBSTRING	SUM	SYSTEM_USER
TABLE	TEMPORARY	THEN	TIME	TIMESTAMP
TIMEZONE_HOUR	TIMEZONE_MINUTE	TO	TRAILING	TRANSACTION
TRANSLATE	TRANSLATION	TRIM	TRUE	UNION
UNIQUE	UNKNOWN	UPDATE	UPPER	USAGE
USER	USING	VALUE	VALUES	VARCHAR
VARYING	VIEW	WHEN	WHENEVER	WHERE
WITH	WORK	WRITE	YEAR	ZONE

## Tipos de dados

A SQL não permite que os usuários definam os seus próprios tipos de dados, eles podem criar domínios que utilizem os tipos já existentes. Observem os mais comuns:

Tipos de dados mais comuns – SQL/92		
CHAR( <i>n</i> ) CHARACTER ( <i>n</i> )	<i>n</i> bytes	Cadeia de caracteres, com tamanho fixo de até 254 bytes.
VARCHAR ( <i>n</i> ) CHAR VARYING ( <i>n</i> ) CHARACTER VARYING ( <i>n</i> )	<i>n</i> bytes	Cadeia de caracteres, com tamanho variável de até 254 bytes.
LONG VARCHAR ( <i>n</i> )	Variável	Cadeia de caracteres, com tamanho não fixado maior que 254 bytes. É equivalente ao BLOB.
DATE	8 bytes	Armazena Datas. Possui dez posições e seu formato padrão é YYYY-MM-DD.
TIME	8 bytes	Armazena Horas. Possui oito posições e seu formato padrão é HH:MM:SS.
TIMESTAMP DATETIME	8 bytes	Armazena, simultaneamente, informação de data e hora.
INTEGER INT	4 bytes	Armazena números inteiros de até 10 dígitos. -2147483648 a +2147483647
SMALLINT	2 bytes	Armazena números inteiros de até 5 dígitos. -32.768 a +32.768
FLOAT ( <i>n</i> )	12 bytes	Armazena valores de ponto flutuante com precisão simples (7 dígitos). $1,175 \times 10^{-38}$ a $3,402 \times 10^{38}$
DOUBLE	16 bytes	Armazena valores de ponto flutuante com precisão dupla (15 dígitos). $2,225 \times 10^{-308}$ a $1,797 \times 10^{308}$
DECIMAL ( <i>p</i> , <i>s</i> )	2, 4 ou 8 bytes	Armazena valores com número fixo de casas decimais. <i>p</i> é o número dígitos para representar os inteiros e <i>s</i> é o número de dígitos para representar os decimais.

## Instruções SQL

Podemos dividir as instruções da linguagem SQL em três grupos:

1. *Data Definition Language* (DDL) – Comandos responsáveis pela criação dos próprio banco de dados, suas tabelas, índices, visões, etc.
2. *Data Control Language* (DCL) – Comandos relacionados à segurança do banco de dados, atribuindo privilégios de acesso aos usuários do banco de dados.
3. *Data Manipulation Language* (DML) – Comandos para acesso (consulta e alteração) aos dados do banco de dados.

Veremos algumas instruções de cada um desses três grupos nas três próximas seções.

Apesar de desenvolvida para ser portátil, é difícil utilizar um trecho de código em SQL escrito para um SGBD em outro, sem nenhuma alteração. Em parte, isso é consequência da existência de três padrões diferentes e do fato de cada SGBD acrescentar extensões proprietárias visando a otimizar o seu próprio funcionamento. Felizmente as alterações necessárias são, normalmente, simples e rápidas.

Neste trabalho manteremos a sintaxe mais próxima da SQL/92, exceto quando o PostgreSQL tiver uma sintaxe diferente. Neste caso, optaremos pela sintaxe mais simples. Quando possível, as cláusulas mais complexas (e menos utilizadas) serão omitidas.

## Notação utilizada

Ao descrevermos a sintaxe de um comando, utilizaremos a notação mais comum no meio da informática:

1. Palavras-chaves estarão em **NEGRITO**.
2. Parâmetros fornecidos pelo usuário estarão em *itálico*.
3. Parâmetros fornecidos pelo usuário, que podem ser divididos em elementos menores, estarão <destacados>.
4. Quando houver mais um parâmetro aplicável a uma determinada situação:
  - 4.1. As opções disponíveis estarão separadas por uma “|” (barra vertical).
  - 4.2. Os parâmetros obrigatórios estarão entre {chaves}. Apenas um desses parâmetros deve ser digitado.
  - 4.3. Os parâmetros opcionais estarão entre [colchetes].

## Instruções DDL

As instruções DDL são responsáveis pela criação, alteração e exclusão dos elementos do banco de dados. Não são executadas com frequência e quando são, demandam cuidado pois são permanentes, não oferecendo condição de desfazer (*rollback*) alguma alteração. São normalmente executados pelo DBA e por um seleto grupo de usuários.

### Create Database

```
CREATE DATABASE database;
```

Cria o banco de dados (quando o SGBD pode manipular mais de um banco de dados). Instrução não implementada na SQL/92, o SGBD trabalha somente com um banco de dados.

### Drop Database

```
DROP DATABASE database;
```

Exclui um banco de dados existente no SGBD. Uma vez executada essa instrução, nenhum dado do banco de dados excluído poderá ser recuperado. Instrução não implementada na SQL/92.

## Connect

**CONNECT** *database*;

Seleciona um banco de dados dentre os existentes no SGBD para trabalhar. Instrução não implementada na SQL/92.

## Create Table

**CREATE TABLE** *table*( *<coldef>* [, *<coldef>* | *<tconstraint>* ... ] );

Onde:

*<coldef>* = *col* { *<datatype>* | **COMPUTED** [ **BY** ] (*<expr>*) } [ **NOT NULL** ]  
 [ **DEFAULT** { *literal* | **NULL** | **USER** } ], [ *<colconstraint>* ]  
*<datatype>* = um dos tipos de dados definidos na SQL2.  
*<expr>* = Uma expressão SQL válida que resulte em um único valor.  
*<colconstraint>* = [ **CONSTRAINT** *constraint* ]  
 { **PRIMARY KEY**  
 | **UNIQUE**  
 | **REFERENCES** *othertable* [(*othercol* [, *othercol* ...])]   
 [ **ON DELETE** { **NO ACTION** | **CASCADE** | **SET NULL** | **SET DEFAULT** } ]  
 [ **ON UPDATE** { **NO ACTION** | **CASCADE** | **SET NULL** | **SET DEFAULT** } ]  
 | **CHECK** ( *condition* ) }  
*<tconstraint>* = [ **CONSTRAINT** *constraint* ]  
 { { **PRIMARY KEY** | **UNIQUE** } ( *col* [, *col* ...] )  
 | **FOREIGN KEY** ( *col* [, *col* ...] ) **REFERENCES** [ *othertable* ( *othercol* [, *othercol* ...] ) ]  
 [ **ON DELETE** { **NO ACTION** | **CASCADE** | **SET NULL** | **SET DEFAULT** } ]  
 [ **ON UPDATE** { **NO ACTION** | **CASCADE** | **SET NULL** | **SET DEFAULT** } ]  
 | **CHECK** ( *condition* ) }

Cria uma tabela no banco de dados corrente, definindo as colunas (campos), chaves primárias, chaves estrangeiras, índices alternativos e restrições.

### Observações:

- *table* é o nome da tabela a ser criada.
- *col* é o nome da coluna a ser criada. A definição das colunas de uma tabela é feita relacionando-as uma após a outra.
- *datatype* contém o tipo e tamanho dos campos definidos para a tabela, segundo o padrão SQL/92.
- **COMPUTED BY** – Campos calculados, onde o valor é obtido através de uma expressão que pode envolver outras colunas das tabelas (as colunas devem ser definidas **antes** da coluna calculada). O conteúdo de um campo calculado não é armazenado, e sim resolvido quando se pede ao SGBD.

- NOT NULL – Exige o preenchimento do campo, ou seja, é obrigatório que possua um conteúdo.
- DEFAULT – Preenche o campo com valores pré-definidos de acordo com o tipo do campo, caso não seja especificado o seu conteúdo no momento da inclusão de novas linhas. Os valores pré-definidos são:
  - Campos numéricos: Zero
  - Campos alfanuméricos: Caractere branco (*ASCII 32*)
  - Campo formato Date: Data corrente.
  - Campo formato Time: Horário corrente.
  - NULL: valores nulos
  - USER: o nome do usuário que está alterando a linha no momento
- CONSTRAINT – Define o nome para a restrição.
- PRIMARY KEY – Define qual a coluna que será a chave primária da tabela. Caso a tabela tenha mais de uma coluna como chave, elas deverão ser relacionadas entre parênteses e separadas por vírgulas.
- FOREIGN KEY – Define quais as colunas que serão chaves estrangeiras. Na cláusula REFERENCES deve ser especificada a tabela relacionada.
- ON DELETE – Esta cláusula define o comportamento quando houver a exclusão de uma linha nas tabelas referenciadas nas chaves estrangeiras, determinando se a exclusão deve ser propagada na tabela atual. As opções são:
  - NO ACTION – Não permite a exclusão de uma linha na tabela relacionada, se houver alguma linha na tabela atual referenciado-o. O mesmo que a opção “RESTRICT”.
  - CASCADE – Exclui todas as linhas da tabela atual que estão relacionados à linha excluída na tabela relacionada.
  - SET NULL – Atribui o valor NULL à chave estrangeira em todas as linhas da tabela atual que estejam relacionadas ao registro excluído.
  - SET DEFAULT – Atribui o valor DEFAULT à chave estrangeira em todas as linhas da tabela atual que estejam relacionadas à linha excluída.
- ON UPDATE – Esta cláusula define o comportamento quando houver alteração nas colunas referenciada nas chaves estrangeiras, determinando se a alteração deve ser propagada na tabela atual. As opções são as mesmas da cláusula ON DELETE.

- CHECK – Esta cláusula permite criar uma restrição de linha. Isto é, uma restrição que verifica a validade de uma expressão a cada tentativa de inclusão ou alteração de uma linha.

## Drop Table

**DROP TABLE** *table*;

Exclui a tabela do banco de dados corrente. Ao ser executada essa instrução, além da estrutura, serão excluídos os dados e os índices relacionados à tabela.

## Alter Table

**ALTER TABLE** *table* <operation> [, <operation> ... ];

Onde:

<operation> = { **ADD** <coldef>

| **ADD** <tconstraint>

| **ALTER** [ **COLUMN** ] *colname* <altcolclause>

| **DROP** *col*

| **DROP CONSTRAINT** *constraint* }

<altcolclause> = { **TO** *newcolname*

| **TYPE** *newdatatype*

| **POSITION** *newposition* }

<datatype> = um dos tipos de dados definidos na SQL2.

<expr> = Uma expressão SQL válida que resulte em um único valor.

<colconstraint> = [ **CONSTRAINT** *constraint* ]

{ **PRIMARY KEY**

| **UNIQUE**

| **REFERENCES** *othertable* [ ( *othercol* [, *othercol* ... ] ) ]

[ **ON DELETE** { **NO ACTION** | **CASCADE** | **SET NULL** | **SET DEFAULT** } ]

[ **ON UPDATE** { **NO ACTION** | **CASCADE** | **SET NULL** | **SET DEFAULT** } ]

| **CHECK** ( *condition* ) }

<tconstraint> = [ **CONSTRAINT** *constraint* ]

{ { **PRIMARY KEY** | **UNIQUE** } ( *col* [, *col* ... ] )

| **FOREIGN KEY** ( *col* [, *col* ... ] ) **REFERENCES** *othertable* [ ( *othercol* [,

*othercol* ... ] ) ]

[ **ON DELETE** { **NO ACTION** | **CASCADE** | **SET NULL** | **SET DEFAULT** } ]

[ **ON UPDATE** { **NO ACTION** | **CASCADE** | **SET NULL** | **SET DEFAULT** } ]

| **CHECK** ( *condition* ) }

Altera a estrutura de uma tabela acrescentando, alterando e retirando colunas ou restrições de integridade.

### Observações:

- ADD – Inclui uma nova coluna ou restrição.

- ALTER – Troca o nome da coluna, seu tipo de dado e posição.
- DROP – Exclui uma coluna ou restrição.

### Create View

```
CREATE VIEW view [ column [ , ... ] ]  
    AS SELECT expr [ AS colname ] [ , ... ]  
    FROM table  
    [ WHERE <condition> ];
```

Onde:

*<condition>* = Critérios de seleção de linhas a serem incluídas.

Cria uma visão no banco de dados corrente, definindo as colunas a serem exibidas e os critérios de seleção utilizados para selecionar as linhas exibidas na visão.

### Drop View

```
DROP VIEW view;
```

Exclui a visão do banco de dados corrente. Ao ser executada essa instrução apenas a estrutura da visão será excluída, os dados das tabelas subjacentes não serão afetados.

### Create Index

```
CREATE [ UNIQUE ] [ ASC[ENDING] | DESC[ENDING] ] INDEX index  
    ON table ( col [ , col ... ] );
```

Cria um índice de acesso para uma ou mais colunas em uma tabela. O uso de índices aumenta a velocidade de acesso aos dados em uma operação de seleção. Instrução não implementada na SQL/92.

#### Observações:

- UNIQUE impede a existência mais de uma linha com o mesmo valor na(s) coluna(s) especificadas.
- ASC[ENDING] classifica as linhas em ordem crescente. Opção padrão.
- DESC[ENDING] classifica as linhas em ordem decrescente.

### Drop Index

```
DROP INDEX index;
```

Exclui o índice do banco de dados. Instrução não implementada na SQL/92.

## Alter Index

**ALTER INDEX** *index* {**ACTIVE** | **INACTIVE** };

Ao contrário do esperado, a instrução ALTER INDEX não permite alterar um índice já existente acrescentando ou excluindo colunas. Ela permite ativar ou desativar um índice. É utilizada normalmente antes da inclusão de um grande número de linhas para aumentar a velocidade, já que quando inativo, o índice não é atualizado. Ao reativá-lo, o índice é atualizado de uma só vez de forma a corresponder à situação atual da tabela. Para adicionar ou remover colunas em um índice, utilize a instrução DROP INDEX para excluí-lo e então utilize a instrução CREATE INDEX para recriá-lo. Instrução não implementada na SQL/92.

## Instruções DCL

Devido à importância dos dados, muitas vezes não é desejável que qualquer usuário tenha acesso a eles. Por essa razão, é possível definir níveis de acesso num banco de dados, restringindo o acesso a determinadas tabelas a um número restrito de usuários. Ou ainda, limitar certas tarefas a um grupo específico de usuários: certos usuários teriam permissão apenas para consultar registros enquanto a outros seria dada permissão para alterar os dados armazenados.

### Create Group

**CREATE GROUP** *<groupname>* [ **WITH** [ **SYSID** *<gid>* ] [ **USER** *<username>* [, ... ] ] ];

Onde:

*<groupname>* = Nome de um grupo de usuários.

*<gid>* = Número de identificação do grupo no PostgreSQL. Se não informado, é utilizado o próximo número disponível (último existente mais um).

*<username>* = Usuários (já existentes) a serem incluídos no grupo.

Permite criar um grupo de usuários que receberá privilégios de acesso a determinados objetos do banco de dados. Somente o DBA tem permissão de criar, alterar e excluir grupos. Essa instrução não existe no padrão SQL/92. O padrão SQL/92 implementa a instrução ROLE que desempenha papel semelhante.

## Alter Group

**ALTER GROUP** *<groupname>* **ADD USER** *<username>* [, ... ];

ou

**ALTER GROUP** *<groupname>* **DROP USER** *<username>* [, ... ];

Onde:

*<groupname>* = Nome de um grupo de usuários.

*<username>* = Usuários a serem incluídos no grupo ou removidos dele.

Permite alterar os usuários que compõem um grupo. No caso de inclusão de usuários no grupo, os usuários devem ter sido criados previamente. Os usuários não sofrem nenhuma alteração exceto a saída do grupo.

## Drop Group

**DROP GROUP** *<groupname>*;

Onde:

*<groupname>* = Nome de um grupo de usuários.

Exclui um grupo do banco de dados. Os usuários não sofrem nenhuma alteração exceto a saída do grupo.

## Grant

**GRANT** *<privileges>* **ON** *<object\_list>* **TO** { **PUBLIC** | **GROUP** *usergroup* | *username* };

Onde:

*<privileges>* = { **ALL** [ **PRIVILEGES** ] | *<privilege\_list>* }

*<privilege\_list>* = **SELECT** | **INSERT** | **UPDATE** | **DELETE** | **RULE**

*<object\_list>* = Nomes de tabelas ou visões, separados por vírgulas.

Permite ao criador de um objeto (tabela ou visão) atribuir privilégios de acesso para um determinado usuário ou grupo de usuários. Nenhum usuário tem permissão sobre um determinado objeto do banco de dados, exceto quando permitido pelo criador do objeto. O criador de um objeto tem, por definição, acesso total ao objeto.

### Observações:

- Os privilégios listados devem ser separados por vírgulas.
- Os nomes dos objetos listados devem ser separados por vírgulas.
- As opções de privilégios a serem atribuídos são:
  - ALL – Acesso total à tabela ou visão.
  - SELECT – Permissão para acessar todas as colunas da tabela numa seleção
  - INSERT – Permissão para incluir dados em todas as colunas da tabela.
  - UPDATE – Permissão para atualizar dados em todas as colunas da tabela.

- DELETE – Permissão para eliminar linhas da tabela.
- RULE – Permissão para definir regras para a tabela.
- Os usuários que receberão os privilégios podem ser especificados um de cada vez (pelo seu *username*), por meio de grupos (GROUP) ou por meio da cláusula PUBLIC que atribui o privilégio a todos os usuários do banco de dados.

## Revoke

**REVOKE** *<privilege>* **ON** *<objectlist>* **FROM** { **PUBLIC** | **GROUP** *usergroup* | *username* };

Onde:

*<privilege>* = { **ALL** [ **PRIVILEGES** ] | *<privilege\_list>* }

*<privilege\_list>* = **SELECT** | **INSERT** | **UPDATE** | **DELETE** | **RULE**

*<objectlist>* = Uma lista com nomes de tabelas ou visões, separados por vírgulas.

Permite ao criador de um objeto (tabela ou visão) retirar privilégios de acesso para um determinado usuário ou grupo de usuários.

### Observações:

- Os privilégios listados devem ser separados por vírgulas.
- Os nomes dos objetos listados devem ser separados por vírgulas.
- As opções de privilégios a serem atribuídos são:
  - ALL – Impede todo o acesso à tabela ou visão.
  - SELECT – Impede o acesso às colunas da tabela numa seleção
  - INSERT – Impede a inserção de dados nas colunas da tabela.
  - UPDATE – Impede a atualização de dados nas colunas da tabela.
  - DELETE – Impede a exclusão de linhas da tabela.
  - RULE – Impede a definição de regras para a tabela.
- Os usuários que perderão os privilégios podem ser especificados um de cada vez (pelo seu *username*), por meio de grupos (GROUP) ou por meio da cláusula PUBLIC que retira o privilégio de todos os usuários do banco de dados.

## Instruções DML

As instruções DML são responsáveis pela manipulação dos dados de um banco de dados. Permitem a inclusão de novas linhas e a alteração e exclusão das existentes. São instruções executadas com bastante frequência e por um grande número de usuários e, devido ao suporte a transações, permitem que um comando seja desfeito caso ocorra algum problema.

## Insert

```
INSERT INTO table [ ( col [ , ... ] ) ]  
    { DEFAULT VALUES | VALUES ( expr [ , ... ] ) | SELECT query };
```

Insere nova(s) linha(s) em uma tabela ou visão. Os valores podem ser constantes especificadas ou o resultado de uma expressão especificada. É possível ainda a inclusão de um conjunto de linhas definido por meio de uma consulta SQL. As linhas deverão atender às restrições de integridade e segurança definidas para o banco de dados, caso contrário a instrução falhará.

### Observações:

- *table* é o nome da tabela que receberá as novas linhas.
- *col* é o nome da coluna que terá os valores definidos.
- *expr* é uma expressão válida, ou uma constante literal.
- *query* é uma consulta SQL que resultará num conjunto de linhas a serem incluídas.

## Update

```
UPDATE table SET col = expr [ , ... ]  
    [ WHERE condition ];
```

Altera os valores nas colunas especificadas para todas as linhas. Caso seja especificada uma condição na cláusula opcional WHERE, somente as linhas que satisfizerem a essa condição serão afetadas. Deverão aparecer na instrução todas as colunas que serão alteradas. Caso as alterações solicitadas não atendam às restrições de integridade e segurança, a instrução falhará.

### Observações:

- *table* é o nome da tabela que será alterada.
- *col* é o nome da coluna que terá os valores alterados.
- *expr* é uma expressão válida, ou uma constante literal.
- *condition* condição SQL que determinará quais linhas serão afetadas.

## Delete

```
DELETE FROM table  
    [ WHERE condition ];
```

Exclui linhas da tabela especificada. Caso seja especificada uma condição na cláusula opcional WHERE, somente as linhas que satisfizerem a essa condição serão afetadas.

**Observações:**

- *table* é o nome da tabela que será alterada.
- *condition* condição SQL que determinará quais linhas serão afetadas.

**Atenção:** Sem a cláusula WHERE, a instrução excluirá todas as linhas da tabela especificada.

## Begin Transaction

**BEGIN** [**WORK** | **TRANSACTION** ];

Inicia uma transação. Normalmente o SGBD considera cada instrução do usuário como uma transação isolada. Essa instrução indica que as instruções a seguir compõem uma única transação.

**Observações:**

- Tanto a cláusula WORK quanto a cláusula TRANSACTION são opcionais. Servem apenas para tornar a instrução mais inteligível para os usuários.

## Commit

**COMMIT** [**WORK** | **TRANSACTION** ];

Torna permanente e encerra a transação corrente. Todas as alterações feitas tornam-se visíveis para os demais usuários.

**Observações:**

- Tanto a cláusula WORK quanto a cláusula TRANSACTION são opcionais. Servem apenas para tornar a instrução mais inteligível para os usuários.
- Tem como sinônima, a instrução END.

## Rollback

**ROLLBACK** [**WORK** | **TRANSACTION** ];

Interrompe a transação atual, desfazendo todas as alterações ocorridas durante ela.

**Observações:**

- Tanto a cláusula WORK quanto a cláusula TRANSACTION são opcionais. Servem apenas para tornar a instrução mais inteligível para os usuários.
- Tem como sinônima, a instrução ABORT.

## Select

### SELECT [ALL | DISTINCT ]

```
<selectedcolumns> [ FROM <from_item> [, ... ]
[ WHERE condition ]
[ GROUP BY <columndef> [, ... ] ]
[ HAVING condition [, ... ] ]
[ { UNION [ ALL ] | INTERSECT [ ALL ] | EXCEPT [ ALL ] } <other_select> ]
[ ORDER BY <columndef> [ ASC | DESC ] ;
```

Onde:

<selectedcolumns> = { \* | ~~showncolumns~~ [, ... ] }

<showncolumns> = <columndef> [ AS outputname ]

<columndef> = { col | expr }

<from\_item> = { tablename [ [ AS ] alias ]

| ( other\_select ) [ AS ] alias

| <from\_item> <join\_type> <from\_item> [ ON join\_condition

| USING ( join\_column\_list ) ] }

Uma referência a uma tabela, o resultado de uma instrução SELECT ou o resultado de uma cláusula JOIN.

<join\_type> = { [ INNER ] JOIN | LEFT [ OUTER ] JOIN | RIGHT [ OUTER ] JOIN  
| FULL [ OUTER ] JOIN | CROSS JOIN }

<other\_select> = Uma outra instrução SELECT (sem as cláusulas GROUP BY, HAVING e ORDER BY) que será combinada com a primeira.

Essa instrução recupera dados de uma ou mais tabelas retornando-os em uma tabela resultante (mesmo que temporária). Essa tabela terá como colunas as expressões listadas em <selectedcolumns>. As linhas que satisfizerem à cláusula WHERE serão as linhas da nova tabela. Se não houver cláusula WHERE, todas as linhas serão selecionadas.

### Observações:

- *condition* é uma expressão que aplicada às linhas, resulte nos valores lógicos Verdadeiro ou Falso. Essa expressão será utilizada para selecionar as linhas que aparecerão na tabela gerada pela instrução.
- *col* é o nome da coluna que será incluída na seleção.
- *expr* é uma expressão SQL válida que resulte num valor.
- *condition* condição SQL que determinará quais linhas serão afetadas.
- *alias* é o nome alternativo dado a uma origem de dados (tabela ou resultado de instrução SELECT).
- DISTINCT – Elimina linhas duplicadas do resultado da instrução.
- ALL – Retorna todas as linhas no resultado da instrução, inclusive as linhas duplicadas. Opção padrão.

- FROM – Especifica a(s) tabela(s) que origina(m) as colunas selecionadas.
- WHERE – Determina, por meio de uma expressão lógica, quais linhas da(s) tabela(s) de origem devem ser exibidas no resultado da instrução.
- GROUP BY – Agrupa, em uma única linha, as linhas da tabela resultante que apresentarem o mesmo valor na coluna especificada.
- HAVING – Da mesma forma que a cláusula WHERE permite a filtragem das linhas a serem exibidas, a cláusula HAVING filtra os grupos (definidos na cláusula GROUP BY) que deverão ser exibidos.
- UNION – Calcula a **união** entre duas instruções SELECT, exibindo todas as linhas das duas instruções. As linhas duplicadas são omitidas, exceto quando a cláusula ALL é especificada.
- INTERSECT – Calcula a **interseção** entre duas instruções SELECT, exibindo apenas as linhas que são comuns às duas. As linhas duplicadas são omitidas, exceto quando a cláusula ALL é especificada.
- EXCEPT – Calcula a **diferença** entre duas instruções SELECT, exibindo as linhas retornadas pela primeira instrução, mas não as retornadas pela segunda.
- ORDER BY – Permite ordenar os dados apresentados na tabela resultante do SELECT. Essa ordenação pode ser de ascendente (ASC – opção padrão) ou descendente (DESC).

A instrução SELECT é uma das mais complexas da linguagem SQL, com várias combinações possíveis entre suas diversas cláusulas. Independente da complexidade, é uma das mais utilizadas, por isso, observaremos com mais detalhes algumas de suas cláusulas mais utilizadas:

### **Cláusula FROM**

Essa cláusula especifica uma ou mais tabelas de origem para a instrução SELECT. Se mais de uma tabela for especificada, o resultado é o produto cartesiano de todas as linhas em todas as colunas.

Uma vez que as instruções SQL geram tabelas como resultado, a origem pode ser também uma instrução SELECT separada por parênteses. Nesse caso, é necessário um *alias* para a tabela resultante desta instrução. Pode ainda, apresentar uma cláusula JOIN que combinará duas outras origens. Logo veremos mais sobre cláusulas JOIN.

### **Cláusula WHERE**

Esta é uma cláusula comum a muitas instruções SQL (como UPDATE, SELECT e DELETE), permite restringir a execução do comando por meio de critérios de seleção. Qualquer expressão lógica envolvendo os campos das tabelas é válida. Sua utilização é sempre a mesma, independente da instrução SQL.

### **Cláusula GROUP BY**

Uma instrução SELECT apresenta as linhas de forma analítica, muitas vezes desejamos consolidar esses dados para que possam ser manipulados mais facilmente. Essa é a função da cláusula GROUP BY. Perde-se a informação detalhada em nível de registros para ficarmos com as informações consolidadas (como subtotais num relatório).

### **Cláusula HAVING**

Caso seja necessário filtrar as informações consolidadas pela cláusula GROUP BY, devemos utilizar esta cláusula. Dessa forma podemos exibir somente os agrupamentos desejados. É o equivalente à cláusula WHERE aplicado a um GROUP BY.

### **Cláusula ORDER BY**

Uma vez que os dados armazenados em um banco de dados relacional não obedecem a nenhuma classificação, muitas vezes desejamos classificá-los antes de sua apresentação. Essa é a função da cláusula ORDER BY. Os campos constantes nesta cláusula devem obrigatoriamente aparecer na cláusula SELECT. Não necessária a existência de um índice no banco de dados, mas se ele existir, o comando será executado de forma mais rápida.

### **Uniões**

De forma equivalente à operação de UNIÃO da teoria de conjuntos, a cláusula UNION agrega resultados de duas instruções SELECT. Deve existir compatibilidade de colunas e as linhas duplicadas serão desprezadas. Sintaxe:

```
SELECT <select_expression1>  
UNION  
SELECT <select_expression2>;
```

### **Junções**

A operação de junção une duas tabelas com base em valores iguais numa coluna comum. Normalmente, isso é utilizado quando temos duas tabelas relacionadas com dados complementares e desejamos apresentar a informação de forma consolidada. No Modelo Relacional isso é representado pelas relações “um-para-um”, “um-para-muitos” e “muitos-para-muitos”.

Consideremos duas tabelas: uma contendo os dados das pacientes numa maternidade e outra contendo os dados dos seus filhos. Na segunda tabela, cada filho teria um campo com dados que permitissem identificar sua mãe de forma única (uma chave estrangeira).

Como na tabela das mães não constam dados dos filhos, caso desejemos um relatório apresentando cada mãe e seu(s) respectivo(s) filho(s), utilizaremos uma junção.

Podemos classificar as junções em:

#### Junção Interna (ou Natural)

É a junção mais simples, sua compreensão é intuitiva (daí o seu nome).

Dadas duas relações A e B com uma coluna comum Y, sua junção natural C apresentará as colunas (não duplicadas) de A e B, e suas linhas serão as linhas de A e B cujos valores Y forem iguais.. Um bom exemplo é o exemplo da maternidade dado há pouco.

Formalmente sua sintaxe é a seguinte:

```
SELECT {col [,col ... ] | * }
FROM <tableA> [ INNER ] JOIN <tableB>
[ ON <searchcondition>]
[ WHERE <searchcondition>;
```

No entanto, normalmente é implementada através do próprio comando SELECT, especificando as ligações envolvidas na cláusula FROM e as condições de ligação na cláusula WHERE.

#### Junção Externa

São menos utilizadas que as junções internas. Ao contrário delas, os valores sem correspondência na outra tabela são incluídos (apresentando o valor simbólico NULL).

Formalmente sua sintaxe é:

```
SELECT {col [,col ... ] | * }
FROM <tableA> { LEFT | RIGHT | FULL } [ OUTER ] JOIN <tableB>
[ ON <searchcondition>]
[ WHERE <searchcondition>;
```

De acordo com a especificação do usuário classificam-se em:

- *Left Outer Join* (Junção Externa à Esquerda) – Retornam todas as linhas da tabela da esquerda (*tableA*) e quaisquer linhas da tabela da direita (*tableB*) que atendam ao critério especificado na cláusula ON.
- *Right Outer Join* (Junção Externa à Direita) – Retornam todas as linhas da tabela da direita (*tableB*) e quaisquer linhas da tabela da esquerda (*tableA*) que atendam ao critério especificado na cláusula ON.
- *Full Outer Join* (Junção Externa Completa) – Combinação das duas anteriores. Retorna todas as linhas de ambas as tabelas, independente do critério de seleção.

## Lock

**LOCK** [ **TABLE** ] table IN [ **ROW** | **ACCESS** ] { **SHARE** | **EXCLUSIVE** } **MODE**;

ou

**LOCK** [ **TABLE** ] table IN **SHARE ROW EXCLUSIVE MODE**;

Solicita explicitamente o bloqueio de uma tabela inteira ou de linhas individuais durante uma transação.

### Observações:

- **ROW** – Bloqueia linhas individualmente.
- **ACCESS** – Bloqueia a estrutura da tabela. Opção padrão.
- **SHARE** – Permite que outras transações solicitem o bloqueio **SHARE**. Impede o bloqueio **EXCLUSIVE**.
- **EXCLUSIVE** – Bloqueio que impede a solicitação de bloqueio por outras transações. Opção padrão.

## Set Transaction Isolation Level

**SET TRANSACTION ISOLATION LEVEL** [ **READ COMMITTED** | **SERIALIZABLE** ];

Define o nível de isolamento da transação atual (que dados a transação pode ver quando há outras transações concorrentes). Não tem efeito em transações posteriores.

Pelo padrão SQL/92, essa instrução não pode ser executada com uma transação em andamento. Entretanto no PostgreSQL deve ser a primeira instrução após o início da transação (**BEGIN TRANSACTION**).

### Observações:

- **READ COMMITTED** – A transação só vê as alterações que foram gravadas (**COMMIT**) antes do seu início. Opção padrão.
- **SERIALIZABLE** – A transação atual só vê as alterações que foram gravadas (**COMMIT**) antes da primeira instrução **DML** desta transação.

## Explain

**EXPLAIN** [ **VERBOSE** ] *query*;

Exibe o plano de execução utilizado pelo otimizador do PostgreSQL para executar a instrução **DML** especificada. Essa instrução não existe no padrão SQL/92.

### Observações:

- **VERBOSE** – Exibe toda a representação interna da árvore de sintaxe abstrata utilizada na execução da instrução.

## Ferramentas oferecidas pela linguagem SQL

Além das instruções, a linguagem SQL oferece alguns outros elementos para complementar as instruções e permitir a manipulação dos dados. O conhecimento dessas ferramentas é tão importante quanto o conhecimento das instruções SQL propriamente ditas. Seguindo o nosso escopo (elementos mais utilizados na linguagem SQL), apresentaremos apenas os operadores e funções de uso mais comum.

### Operadores

A linguagem SQL possui lógicos, matemáticos, de manipulação de strings e de comparação. Estes últimos incluem ainda algumas construções especiais, como veremos adiante.

#### Operadores Lógicos

Todas as operações lógicas são feitas utilizando combinações de apenas três operadores:

- Operador binário representando o “E” lógico.
- Operador binário representando o “OU” lógico.
- Operador unário representando o “NÃO” lógico.

A linguagem SQL utiliza tabelas verdade com três valores lógicos (FALSO, VERDADEIRO e NULL). Observe as avaliações do valor especial NULL (em itálico):

P	Q	NOT P	P AND Q	P OR Q
VERDADEIRO	VERDADEIRO	FALSO	VERDADEIRO	VERDADEIRO
VERDADEIRO	FALSO	FALSO	FALSO	VERDADEIRO
VERDADEIRO	<i>NULL</i>	FALSO	<i>NULL</i>	VERDADEIRO
FALSO	VERDADEIRO	VERDADEIRO	FALSO	VERDADEIRO
FALSO	FALSO	VERDADEIRO	FALSO	FALSO
FALSO	<i>NULL</i>	VERDADEIRO	FALSO	<i>NULL</i>
<i>NULL</i>	VERDADEIRO	<i>NULL</i>	<i>NULL</i>	VERDADEIRO
<i>NULL</i>	FALSO	<i>NULL</i>	FALSO	<i>NULL</i>
<i>NULL</i>	<i>NULL</i>	<i>NULL</i>	<i>NULL</i>	<i>NULL</i>

#### Operadores de Comparação

São operadores binários que retornam valores lógicos de acordo com o resultado da comparação entre os valores. Os dois operandos devem ser de tipos de dados compatíveis, ou seja, comparáveis entre si (não faz sentido comparar um dado do tipo VARCHAR com um dado do tipo FLOAT). As comparações são feitas binariamente, da esquerda para a direita (exceto quando o uso de parêntesis altere a precedência). Comparações do tipo “1 < 2 < 3” não são válidas, pois são resolvidas como “1 < 2” (VERDADEIRO) e “VERDADEIRO < 3” que não faz sentido.

Operador	Descrição
<	Menor que
>	Maior que
<=	Menor ou igual a
>=	Maior ou igual a
=	Igual a
<> (ou !=)	Diferente de

Existem ainda as construções especiais a seguir que podem substituir uma comparação de intervalos com a mesma eficiência e são mais amigáveis:

Construção	É equivalente a
a BETWEEN x AND y	a >= x AND a <= y
a NOT BETWEEN x AND y	a < x OR a > y
a IN (x, y, z)	a = x OR a = y OR a = z

Quanto ao valor especial NULL, uma vez que não pode ser comparado pelas vias normais (qualquer comparação envolvendo NULL devolve NULL, lembra-se?), utilizamos outras duas construções especiais:

Construção	Substitui a comparação
expr IS NULL	expr = NULL
expr IS NOT NULL	expr <> NULL

Existe ainda um operador de comparação de *strings* utilizado normalmente na cláusula WHERE quando desejamos obter as linhas em que os valores de uma coluna sigam um determinado padrão.

O operador LIKE utiliza os caracteres especiais “%” (porcentagem) e “\_” (sublinhado), e segue a sintaxe abaixo:

```
SELECT {col [,col ... ] | *}
FROM <table>
WHERE <column> LIKE <pattern>;
```

O posicionamento dos caracteres porcentagem e sublinhado definem o padrão a ser procurado. O caractere de sublinhado significa que qualquer caractere naquela posição será considerado válido, enquanto o caractere de porcentagem significa que uma seqüência de caracteres naquela posição será considerada válida. Observe os exemplos abaixo:

Condição	Significado
WHERE NOME LIKE 'JOSE%'	Qualquer nome que comece com 'JOSE'
WHERE NOME LIKE '%CARNEIRO'	Qualquer nome terminado em 'CARNEIRO'
WHERE NOME LIKE ' _R%'	Qualquer nome em que a terceira letra seja 'R'.

## Operadores Matemáticos

O conjunto de operadores matemáticos oferecido pela linguagem SQL é bastante completo, abrangem inclusive operações binárias (“E”, “OU”, “NOT” e deslocamento binários). Apresentaremos apenas os mais utilizados:

Operador	Descrição
+	Soma
-	Subtração
*	Multiplicação
/	Divisão
^	Exponenciação
!	Fatorial

### Operadores para *Strings*

Basicamente existe apenas um operador para lidar com *strings* (cadeias de caracteres), o operador binário “||” que realiza a concatenação de caracteres. Ou seja:

“Postgre” || “SQL” resulta em “PostgreSQL”.

### Funções

As funções oferecidas pela linguagem SQL podem ser classificadas em sete grupos principais:

- Manipulação de *Strings*
- Matemáticas
- Estatísticas
- Formatação e Conversão de Tipos
- Data e Hora
- Geométricas

Neste trabalho, daremos mais atenção aos três primeiros grupos.

### Funções de manipulação de *Strings*

As funções de manipulação de *strings* são utilizadas tanto no tratamento de resultados de consultas e relatórios, quanto na definição regras de restrição e valores para campos. As mais comuns são:

Função	Descrição
ASCII( <i>text</i> )	Código ASCII do primeiro caractere
CHAR_LENGTH( <i>string</i> ) CHARACTER_LENGTH( <i>string</i> )	Comprimento da <i>string</i>
CHR( <i>integer</i> )	Caractere correspondente ao código ASCII
INITCAP( <i>text</i> )	Conversão das iniciais para maiúsculas
LOWER( <i>string</i> )	Conversão para minúsculas
LPAD( <i>stringx</i> , [ <i>substring</i> ] )	Preenchimento com caracteres à esquerda
LTRIM( <i>stringsubstring</i> )	Remoção de caracteres à esquerda
REPEAT( <i>stringx</i> )	Repetições da <i>string</i> especificada
RPAD( <i>stringx</i> , [ <i>substring</i> ] )	Preenchimento com caracteres à direita
RTRIM( <i>stringsubstring</i> )	Remoção de caracteres à direita
STRPOS( <i>stringsubstring</i> )	Localização de <i>substring</i>
SUBSTR( <i>stringx</i> , [ <i>y</i> ] )	Extração de <i>substring</i>
UPPER( <i>string</i> )	Conversão para maiúsculas

## Funções Matemáticas

Função	Descrição
ABS(x)	Valor absoluto
ACOS(x)	Arco cosseno
ASIN(x)	Arco seno
ATAN(x)	Tangente inversa
CBRT(x)	Raiz cúbica
CEIL(x)	Inteiro posterior
COS(x)	Cosseno
COT(x)	Cotangente
DEGREES(x)	Conversão de radianos para graus
EXP(x)	Exponenciação
FLOOR(x)	Inteiro anterior
LN(x)	Logaritmo Neperiano
LOG(x)	Logaritmo na base 10
LOG(x, y)	Logaritmo na base especificada
MOD(x, y)	Resto da divisão
PI()	Constante $\Pi$
POW(x, y)	Potenciação
RADIANS(x)	Conversão de graus para radianos
RANDOM()	Número aleatório entre 0 e 1
ROUND(x, y)	Arredondamento
SIN(x)	Seno
SQRT(x)	Raiz quadrada
TAN(x)	Tangente
TRUNC(x, y)	Trunca um número

## Funções Estatísticas

As funções estatísticas permitem agrupar os valores constantes numa tabela devolvendo um resultado único (como a soma ou a média dos valores encontrados). São muito úteis em relatórios e consultas de caráter gerencial.

Função	Descrição
AVG(x)	Média aritmética
COUNT(x)	Contagem de ocorrências encontradas
MAX(x)	O maior valor encontrado
MIN(x)	O menor valor encontrado
STDDEV(x)	Desvio padrão
SUM(x)	Somatório
VARIANCE(x)	Variância

Permitem, além da tradicional restrição com cláusula WHERE, a utilização dos parâmetros (ALL e DISTINCT) que determinam o conjunto de linhas a serem consideradas.

## Stored Procedures (Procedimentos Armazenados)

São basicamente programas pré-compilados que ficam armazenados no servidor e são executados por solicitação do cliente. A execução é restrita ao servidor. O cliente solicita a execução, aguarda o seu término e recebe de volta somente o seu resultado.

Suas principais vantagens são:

- Aumentam o desempenho em ambientes Cliente/Servidor pois diminuem o tráfego de informações entre os clientes e o servidor.
- Possibilitam um maior grau de independência de dados pois ocultam do usuário detalhes do banco de dados e do SGBD.
- Oferecem maior segurança. Os usuários podem não ter autorização para alterar determinados dados, mas serem autorizados a executar uma *stored procedure* que os atualiza.
- Criam uma camada contendo todas as regras de negócio da empresa, tornando-as independentes do cliente. Uma alteração nas regras de negócio não implicaria em alterações na camada de clientes.

Como exemplo de *stored procedures* podemos citar as operações em *batch* (lote) muito comuns em sistemas de grande porte e operações compostas por diversas operações menores, como a conciliação de contas.

Com os avanços tecnológicos, tanto nos sistemas operacionais quanto no desenvolvimento de aplicações, é cada vez mais comum a utilização das técnicas de multiprocessamento e bancos

de dados distribuídos. Nesses casos, a independência proporcionada pelo uso de *stored procedures* e *triggers* é fundamental.

### *Triggers* (Gatilhos)

Um *trigger* é um tipo especial de procedimento armazenado que será disparado (executado automaticamente pelo SGBD) na ocorrência de um determinado evento. Como eventos disparadores temos normalmente a execução de alguma operação que atualize os dados do banco de dados ou a violação de alguma regra de integridade.

Observe que, apesar de possível, é desaconselhada a utilização de *triggers* para a manutenção de integridade. Seu uso é mais indicado em operações com relação de consequência. Por exemplo, ao incluir os dados de uma fiscal de venda, um *trigger* pode já incluir pedidos para repor os itens vendidos.

# Tópicos Adicionais

## Transações

Uma transação é uma unidade lógica de trabalho. Um grupo de operações relacionadas que devem ser executadas como uma unidade. Caso uma operação não seja concluída com sucesso, nenhuma das outras deve ser executada e o banco de dados deve retornar ao estado anterior.

Consideremos a inclusão, no banco de dados, de uma nota fiscal contendo vários itens. Caso um dos itens não seja incluído (por qualquer razão), toda a inclusão da nota deve ser interrompida sob pena de criarmos inconsistências nos dados (o total da nota não será condizente com o somatório dos itens!).

## Recuperação de Transações

Como vimos, as instruções SQL responsáveis por implementar o controle de transações são:

- **BEGIN TRANSACTION** – Indica o início de uma transação. Deve ser registrada a situação atual do banco de dados e iniciado o registro das operações para, no caso de uma eventual falha, retornar o banco de dados a essa situação.
- **COMMIT** – Indica o término com sucesso de uma transação. As informações atuais podem ser tornadas permanentes.
- **ROLLBACK** – Indica o insucesso na execução de uma transação, e que SGBD deve “desfazer” todas as alterações ocorridas na transação.

Antes que cada operação seja executada, o SGBD deve registrá-la no arquivo de *log* (registro). Assim, se antes da execução de todas as operações que compõem a transação, um erro (previsto ou não) for observado, um **ROLLBACK** será executado trazendo o banco de dados de volta à situação anterior. Essa gravação antecipada tem o nome de *Write-ahead Logging* (Gravação Antecipada do Registro).

Em intervalos predeterminados, em geral um número preestabelecido de entradas no registro, o SGBD grava fisicamente a situação atual em disco e marca um *checkpoint* (ponto de checagem) no registro. Esse *checkpoint* fornece uma lista das transações que estavam em andamento naquele instante.

Caso haja uma falha no sistema (uma falta de energia por exemplo) que interrompa o processamento abruptamente, quando o SGBD é reinicializado:

1. São criadas duas listas *REDO* (refazer) e *UNDO* (desfazer). A lista *UNDO* contém todas as transações em andamento no momento do último *checkpoint*. A lista *REDO* começa vazia.
2. O SGBD pesquisa o registro a partir do último *checkpoint*. Toda vez que é encontrada uma instrução *BEGIN TRANSACTION*, essa transação é acrescentada à lista *UNDO*. Quando é encontrada uma instrução *COMMIT*, a transação correspondente é movida para a lista *REDO*. Finalmente, cada vez que é encontrada uma instrução *ROLLBACK*, a transação correspondente é retirada da lista *UNDO*.
3. Agora que o SGBD tem listas com todas as transações a serem desfeitas e todas as transações a serem refeitas, ele percorre as listas refazendo as transações constantes da lista *REDO* e desfazendo as transações constantes da lista *UNDO*.

Só depois da conclusão dessas atividades de recuperação de transações, o sistema aceitará alguma instrução.

## Concorrência

Apesar de a maioria dos sistemas de banco de dados serem multiusuários, não levamos isso em consideração nos nossos estudos até o momento. Em um sistema de banco de dados multiusuário, existe o problema da concorrência, ou seja o acesso simultâneo de mais de um usuário aos dados.

Os problemas mais comuns de concorrência são:

- A atualização perdida (*lost update*) – Uma transação A acessa uma tupla num momento  $T_1$ . Uma transação B acessa a mesma tupla no instante  $T_2$ . A transação atualiza a tupla no momento  $T_3$  com base nos dados lidos no momento  $T_1$ . A transação B atualiza a tupla no momento  $T_4$  com os dados lidos no momento  $T_2$  (os mesmos do momento  $T_1$ ). As alterações realizadas pela transação A são sobrescritas pelas alterações da transação B.
- A leitura suja (*dirty read*) – Uma transação A realiza uma alteração numa tupla num momento  $T_1$ . Uma transação B acessa a mesma tupla no instante  $T_2$  (antes que a transação A seja devidamente encerrada). A transação A é cancelada por meio de um *ROLLBACK* no momento  $T_3$ . A transação B estará operando com base em informações que não são mais válidas podendo inclusive realizar outras alterações (às vezes em outras tuplas) baseada nessas informações.

- A análise inconsistente – A transação A realiza o somatório das tuplas I e II no momento  $T_1$ . A transação B altera o valor da tupla I no momento  $T_2$ . A transação A adiciona ao somatório o valor da tupla III no momento  $T_3$ . O somatório retornado pela transação A está incorreto pois tomou por base um valor que foi alterado depois pela transação B. Esse caso normalmente é consequência de uma leitura suja.

## Bloqueios (*Locks*)

Uma maneira de evitar esses problemas é pela utilização de bloqueios (*locks*). Toda vez que uma transação precisar acessar uma ou mais tuplas e deseja garantir que os valores não serão alterados por outra transação, ela pode solicitar o bloqueio dessas tuplas.

Existem vários tipos de bloqueio, consideraremos apenas os mais simples:

- Bloqueio de Gravação (*write lock*) – Impede qualquer solicitação de bloqueio das tuplas por outra transação até que a transação A seja concluída (com ou sem sucesso).
- Bloqueio de Leitura (*read lock*) – Impede a solicitação de bloqueios de gravação por qualquer outra transação até que a transação A seja concluída (com ou sem sucesso). As solicitações de bloqueios de leitura são permitidas.

Uma transação que deseja ler uma tupla deve solicitar o bloqueio de leitura sobre ela. Caso deseje alterar uma tupla, a transação deve solicitar o bloqueio de escrita sobre ela. Essas solicitações são, em geral implícitas e, ao término da transação os bloqueios são liberados.

Se não for possível obter o bloqueio solicitado por uma transação, a transação entra em estado espera, permanecendo assim até que a solicitação seja concedida. Para evitar que uma determinada transação permaneça em estado de espera indefinidamente, o SGBD atende às solicitações na ordem “primeiro a chegar, primeiro a ser atendido”.

## Bloqueio de duas fases

O **teorema do bloqueio de duas fases** determina que uma transação deve possuir duas fases: uma fase inicial em que obtém os bloqueios necessários e uma fase final quando, após as alterações, libera cada um dos bloqueios obtidos. Este teorema determina ainda que após liberado um bloqueio, nenhum outro pode ser obtido. De fato, no padrão SQL/92 não há instruções de bloqueio, somente o SGBD pode bloquear alguma tupla.

Entretanto como as transações normalmente envolvem interação humana (entrada de dados), para poupar recursos e melhorar o desempenho, normalmente os SGBD permitem que as transações liberem bloqueios antes do término e obtenham outros e oferecem algum recurso de bloqueio explícito para que a segurança seja garantida pela aplicação.

### *Deadlocks*

Utilizando o recurso do bloqueio, podemos resolver os problemas de concorrência vistos anteriormente. Entretanto existe a possibilidade de um novo problema: o *deadlock* (espera indefinida de duas ou mais transações por um determinado evento; no caso, o bloqueio de uma determinada tupla). O ideal é que o SGBD seja capaz de detectar um *deadlock* e interrompê-lo, escolhendo uma das transações envolvidas e forçando um ROLLBACK.

Nem todos os SGBD são capazes de detectar de fato um *deadlock*. Alguns apenas utilizam o critério do tempo: uma transação que não realize qualquer trabalho num período predeterminado de tempo será considerada em *deadlock* e interrompida.

A transação interrompida precisa ser refeita. Alguns (poucos) SGBD detectam que a transação foi interrompida por *deadlock* e a refazem automaticamente. A maioria simplesmente devolve uma mensagem avisando o ocorrido. Se for esse o caso, a aplicação deve tomar as providências necessárias para refazer a transação. Qualquer que seja a solução, ela deve ser transparente para o usuário final.

## Níveis de Isolamento

Por definição as transações devem ser serializáveis, isto é, podem ser executadas em qualquer ordem. Executar a transação A e em seguida a transação B terá o mesmo resultado que executar a transação B e em seguida a transação A.

Para que as transações sejam serializáveis, o nível de isolamento deve ser máximo. O nível de isolamento de uma transação define qual o nível de interferência que transações concorrentes têm entre si. Portanto, para que as transações sejam serializáveis, as alterações efetuadas por uma transação são invisíveis às demais até a sua conclusão.

Como vimos, o padrão SQL/92 não oferece opções explícitas de bloqueio, mas oferece quatro níveis de isolamento, em ordem decrescente de isolamento:

- SERIALIZABLE – Isolamento máximo. A transação atual não visualiza as alterações feitas pelas demais. Permite que as transações sejam serializáveis.

- REPEATABLE READ – A transação atual não visualiza as alterações que ocorreram após o primeiro acesso aos dados.
- READ COMMITTED – A transação atual não visualiza as alterações que ainda não foram gravadas (COMMIT) pelas demais transações.
- READ UNCOMMITTED – A transação visualiza as alterações das demais transações, independente de haverem sido gravadas (COMMIT).

O PostgreSQL por outro lado, oferece apenas dois:

- SERIALIZABLE – Isolamento máximo. A transação atual não visualiza as alterações feitas pelas demais. Permite que as transações sejam serializáveis.
- READ COMMITTED – A transação atual não visualiza as alterações que ainda não foram gravadas (COMMIT) pelas demais transações. Opção padrão.

## Segurança do Banco de Dados

Mencionamos diversas vezes os termos “integridade” e “segurança”. É comum serem confundidos, mas os dois conceitos são bastante diferentes. Enquanto integridade se refere à proteção dos dados quanto à sua exatidão, segurança se refere à proteção quanto aos acessos (para consulta ou alteração) não autorizados.

A política de segurança de um banco de dados está diretamente ligada à política da empresa. Da mesma forma que o projeto do banco de dados, sua definição é atribuição do administrador de dados e sua implementação (e posterior manutenção) é atribuição do DBA.

A segurança de um banco de dados ocorre sob diversos aspectos (acesso físico, escolha de sistema operacional, dispositivos biométricos, etc.), porém daremos mais atenção ao aspecto da segurança oferecida pelo SGBD. De uma maneira geral, a segurança de um banco de dados é feita por meio da autorização ou negação do acesso de um usuário a um determinado objeto do banco de dados (uma tabela, visão, *stored procedure*, etc.).

Quando um usuário deseja acessar o banco de dados, deve se identificar fornecendo sua *userid* (identificação de usuário) e uma confirmação de identidade (senha, impressão digital ou de retina, etc.). De posse dessa identificação, o banco de dados recupera as definições de segurança do dicionário de dados e concede os privilégios ao usuário para aquela sessão de trabalho. Existem dois principais sistemas de controle de acesso: mandatário e discricionário.

## Controle Mandatário

Cada objeto tem um determinado nível de classificação e cada usuário, um nível de liberação. Existe um nível de liberação correspondente a cada nível de classificação.

- Um usuário só pode ter acesso de leitura a um objeto se seu nível de liberação é maior ou igual ao nível de classificação do objeto.
- Um usuário só pode excluir objetos que tenham um nível de classificação igual ou inferior ao seu nível de liberação.
- Todos os objetos atualizados (incluídos e alterados) por um usuário assumem automaticamente o nível de classificação correspondente ao seu nível de liberação. Evita-se assim que um usuário leia um dado exclusivo para o seu nível de liberação e grave-o num nível de classificação inferior, quebrando o esquema de segurança.

Esse sistema começou a receber mais atenção quando Departamento de Defesa norte-americano estabeleceu as exigências para aquisição de SGBD. É um sistema mais rígido e hierárquico, sendo mais utilizado em organizações militares e governamentais.

## Controle Discricionário

São concedidos, aos usuários, privilégios de acesso aos objetos do banco de dados. Esses privilégios podem ser distintos para objetos e usuários distintos. É um sistema mais flexível e o implementado pelo padrão SQL/92.

A atribuição de privilégios é realizada por meio de instruções SQL e a cada comando dado pelo usuário o SGBD verifica o seu acesso aos dados solicitados. Caso o usuário não tenha o privilégio solicitado, a ação padrão é simplesmente negar o acesso e devolver uma mensagem informando o ocorrido. O aplicativo então opta por simplesmente informar ao usuário a negativa, ou tomar ações mais drásticas como interromper a sessão do usuário ou avisar ao DBA.

## Outros recursos de segurança

Paralelamente aos sistemas de controle propriamente ditos, existem outros recursos de segurança que são aplicados dependendo do nível de segurança desejado:

- Visões – Um recurso muito simples, mas muito poderoso, da própria linguagem SQL é a utilização de visões. Simplesmente impedimos os usuários de terem

acesso direto às tabelas, obrigando-os a utilizarem visões, que apresentarão a eles apenas os dados desejados. Do ponto de vista dos usuários não autorizados, os demais dados simplesmente “não existem”.

- *Stored Procedures* – De modo semelhante ao uso de visões, limitamos o acesso direto dos usuários aos dados armazenados.
- Auditoria – O aplicativo (ou o SGBD, dependendo do fabricante) grava em uma tabela de auditoria todos os acessos ao banco de dados. Muitas vezes, só o fato de saber que suas ações serão registradas impede algum usuário mais ousado. De qualquer forma, em caso de necessidade, os usuários responsáveis pela segurança podem consultar a tabela de auditoria, utilizando sua linguagem padrão de consulta para obter os dados do acesso.
- Criptografia de Dados – Supomos até o momento que a tentativa de acesso seria utilizando os recursos do próprio SGBD. Pode ocorrer entretanto, que o invasor tente contornar o SGBD, acessando diretamente os arquivos do banco de dados ou espionando os dados que trafegam entre o cliente e o servidor. Para esses casos, a providência mais eficaz é utilizar algum sistema de criptografia. Existem diversos métodos de criptografia (é uma das ciências mais antigas da humanidade!), cada um com suas vantagens e desvantagens.

## Otimização

As instruções DML normalmente têm mais de uma forma de serem executadas. Considerando que maioria das instruções DML age sobre um grande número de linhas, e que muitas dessas instruções realizam diversas operações sobre esse mesmo conjunto de linhas, a ordem em que essas operações são executadas causa enorme diferença no tempo de resposta e nos recursos envolvidos para obter essa resposta.

Otimização é o processo de escolher o melhor modo de executar as operações necessárias para cumprir a instrução recebida. Todo SGBD utiliza algum tipo de otimização porém, nos sistemas não-relacionais essa otimização fica a cargo do usuário do sistema. Caso as escolhas feitas pelo usuário não sejam as mais indicadas, não há nada que o sistema possa fazer.

Os sistemas relacionais, por utilizarem uma linguagem não-procedural de alto nível (SQL) que especifica o “QUÊ” fazer em lugar do “COMO” fazer, têm a liberdade de otimizar as instruções recebidas da forma mais conveniente. Na verdade, os sistemas relacionais dependem de um sistema de otimização para terem uma eficiência razoável.

Essa dependência, que poderia ser um problema, termina sendo uma grande vantagem dos sistemas relacionais pelo simples fato que é quase certo que a otimização feita automaticamente pelo sistema seja melhor do que a otimização escolhida pelo usuário.

Em primeiro lugar, porque o sistema dispõe de informações (como a estrutura do banco de dados e suas estatísticas de utilização) que não são do conhecimento do usuário. E em segundo lugar, porque sendo o SGBD tem a seu favor o fato que o procedimento de otimização é repetitivo e maçante, muito mais adequado para uma máquina do que para um ser humano.

Em certas ocasiões pode ser interessante para o usuário seguir obrigatoriamente um plano definido por ele, independente de não ser o mais eficiente. Entretanto, o padrão SQL/92 não oferece essa possibilidade. O objetivo é sempre a eficiência máxima, independente da vontade do usuário.

Existem SGBDR que permitem que o plano de execução definido automaticamente pelo sistema seja substituído por outro definido pelo usuário, mas isso varia de produto para produto.

No caso do PostgreSQL, não é possível trocar o plano de execução. Porém o usuário pode consultar o plano escolhido pelo SGBD para, baseado nessas informações, criar as condições (como índices alternativos ou visões) que permitam forçar uma alteração nessa escolha de acordo com suas necessidades.

## Bibliografia

DATE, C. J. **Introdução a Sistemas de Bancos de Dados**. Rio de Janeiro: Campus, 2000.

Butzen, Fred; Forbes, Dorothy. **Linux – Bancos de Dados**. Rio de Janeiro: Ciência Moderna, 1997.

International Organization for Standardization (ISO). **Database Language SQL**. Documento ISO 9075:1992.

RED HAT, INC. **Red Hat Database: SQL Guide and Reference**. [on line] Disponível na INTERNET via URL: <http://www.redhat.com/docs/manuals/database>. Arquivo capturado em 02/07/2002.

Wirth, Niklaus. **Algoritmos e Estruturas de Dados**. Rio de Janeiro: Prentice/Hall do Brasil.