Aspectos Básicos de Banco de Dados

Prof. Rogério Gonçalves Bittencourt, M.Sc.

Florianópolis, fevereiro de 2004.

Conteúdo

Introdução

Neste capítulo apresentaremos alguns termos (terminologia) e conceitos de grande importância.

Qual a Diferença Entre Dados, Informação e Conhecimento

DADOS - representação de fatos, conceitos ou instruções de maneiras formalizadas, adequadas para comunicação, interpretação ou processamento por pessoas ou meios automatizados.

INFORMAÇÃO - significado que pessoas associam aos dados através de convenções usadas em sua interpretação.

CONHECIMENTO - discernimento, critério, apreciação prática de vida, experiência.

Banco de Dados

"Um banco de dados é um conjunto de arquivos relacionados entre si" (Chu, 1983)

"Um banco de dados é uma coleção de dados operacionais armazenados, sendo usados pelos sistemas de aplicação de uma determinada organização" (C. J. Date, 1985)

"Um banco de dados é uma coleção de dados relacionais" (Elmasri & Navathe, 1989)

"Um banco de dados é um conjunto de dados armazenados, cujo conteúdo informativo representa, a cada instante, o estado atual de uma determinada aplicação" (Laender, 1990)

Baseado nas definições acima se pode deduzir então que Banco de Dados é:

- Coleção de dados relacionados;
- Coleção logicamente coerente de dados com algum significado inerente;
- Um BD está sempre associado a aplicações e a usuários que têm interesse nele.

Ex: Agenda de endereços

Sistema de Gerência de Banco de Dados (SGBD)

"O SGBD permite a *definição*, *construção* e *manipulação* do banco de dados para diversas aplicações".

DEFINIÇÃO do BD:
Envolve a especificação dos tipos de dados a serem armazenados no BD, mais a descrição de cada tipo de dado.

CONSTRUÇÃO do BD:
Processo de armazenar os dados em um meio controlado pelo SGBD.

MANIPULAÇÃO do BD:
Execução de operações de consulta e recuperação de dados específicos, além de atualização de dados para refletir, no BD, mudanças no mini-mundo sendo modelado. A manipulação inclui, também, a geração de relatórios a partir dos dados do BD.

Sistema de Banco de Dados

Sistema de Banco de Dados é um sistema de software composto pelos programas de aplicação, pelo SGBD e pelo BD, para um conjunto de aplicações de uma mesma organização.

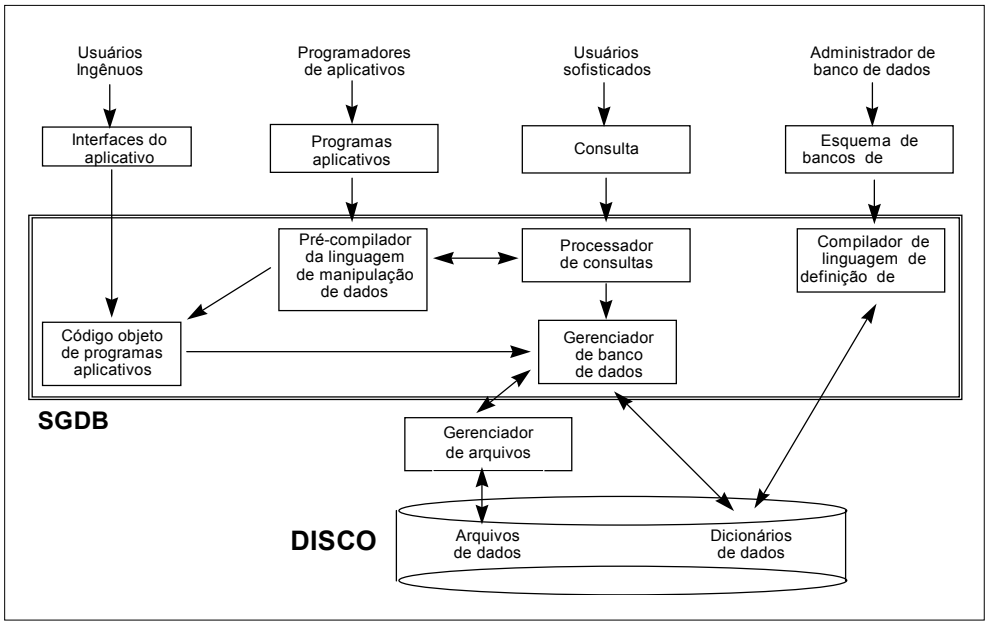
Programas de aplicação, colocado na definição acima, são programas que realizam funções da aplicação. EX.: cálculo das deduções e impostos, a partir da receita apurada, dos custos computados e da legislação em vigor. Eles também são os responsáveis pela garantia das restrições de integridade que não podem ser controladas pelo SGBD. Implementam interfaces e relatórios específicos. Acessam o BD através do SGBD para consulta e atualização dos dados da aplicação.

Resumindo:

SBD = BD + SGBD + PA

De acordo com (DATE,1985), um SBD é dividido em módulos que tratam de partes, em separado, cada uma das responsabilidades do sistema geral. Estes componentes fundamentais são:

- Gerenciador de Arquivos → que trata da alocação do espaço para armazenamento e das estruturas de dados utilizadas para representar a informação armazenada no disco;
- Gerenciador de Banco de Dados → fornece a interface entre os dados de baixo nível armazenados no disco e os programas aplicativos e de consulta submetidos ao sistema;
- Processador de Consultas → traduz as consultas escritas em uma linguagem de alto nível para instruções de baixo nível que o gerenciador do banco de dados entende;
- Pré-compilador DML → converte comandos DML embutidos em um aplicativo para chamadas de procedimento normal na linguagem hospedeira;
- Arquivos de Dados → armazenam banco de dados por si mesmos;
- Dicionário de Dados → é o componente responsável pelo armazenamento dos metadados sobre a estrutura do banco de dados. O dicionário de dados é bastante utilizado;



Porque Usar Banco de Dados

Sistema de Banco de Dados proporciona à empresa o ***controle centralizado*** de seus dados operacionais. Tal situação contrasta nitidamente com o que podemos encontrar em uma empresa que não utiliza um SGBD, onde cada aplicação dispõe de seus próprios arquivos de tal forma que os dados operacionais são muito dispersos, dificultando o controle sistemático. Isto implica que exista um DBA, isto é, um Administrador do Banco de Dados (*Database Administrator*, em inglês).

Vantagens do Controle Centralizado

Reduzir Redundância
Nos sistemas gerenciadores de arquivos, cada aplicação possui seus próprios arquivos. Este fato costuma provocar uma redundância considerável nos dados armazenados, causando desperdício de espaço de armazenamento.

Evitar Inconsistência
A inconsistência é conseqüência natural da redundância. Suponhamos que um certo fato do mundo real (o fato de que um fornecedor X fornece o item Y para a empresa) é representado por duas entradas distintas no banco de dados, e que o SGBD não tenha conhecimento da duplicidade (redundância não controlada). Ocorrerá que em determinado momento duas entradas não são concordantes. Diz-se, então, que o banco de dados é inconsistente.

Compartilhamento dos Dados
O compartilhamento não significa apenas que as aplicações existentes podem compartilhar os dados do banco de dados, mas também que novas aplicações podem ser desenvolvidas para operar sobre os mesmos dados armazenados.

Padronização
Pelo fato do controle centralizado, o SGBD pode assegurar que todos os padrões aplicáveis serão observados na representação dos dados.

Restrições de Segurança
O DBA (Adm. do Banco de Dados), detendo toda a autoridade sobre as dados operacionais, pode assegurar:

- Que os únicos meios de acesso ao banco de dados sejam realizados através de certos canais;
- Definir controles de segurança a adotar (principalmente para dados especiais);

- Estabelecer diferentes controles para cada tipo de acesso (recuperação, modificação, anulação, etc.), e para cada parte da informação no banco de dados.

Manter a Integridade
O problema da integridade é assegurar que os dados do banco de dados sejam corretos (íntegros), ou seja, as informações que compõe o BD têm que expressar exatamente o que foi informado, o BD não pode permitir que as informações se modifiquem incorretamente.

- **Integridade Referencial** - os registros de relacionamentos devem fazer referência a ocorrências de entidades existentes no banco de dados. Não deve haver relacionamento referenciando uma chave primária inexistente.
- **Integridade Transacional** - as transações efetuadas na base de dados devem ocorrer com segurança, completando-se ou não o procedimento, os dados devem se manter íntegros.

Equilibrar Necessidades Conflitantes
O DBA, tendo conhecimento das necessidades globais da empresa (em oposição às necessidades de um usuário individual) pode estruturar o sistema, a fim de proporcionar um serviço geral que seja "o melhor para a empresa".

Independência dos Dados
Independência de dados é um dos objetivos de um SGBD, e consiste na capacidade de isolar programas de aplicação das mudanças em estruturas de armazenamento (esquema físico), definição dos dados (esquema lógico) e das estratégias de acesso do BD. Um SGBD que ofereça independência de dados garante que programas continuem a rodar se os dados armazenados forem reorganizados para atender a outra aplicação prioritária. Aplicações baseadas em sistemas de arquivos dependem dos dados.

O Que se Pode Almejar Com o Uso de SBD

- Redundância controlada de dados
- Compartilhamento de dados por aplicações diversas
- Controle de autorização de acesso a dados
- Acesso a dados através de diferentes interfaces
- Modelagem de relacionamentos complexos entre dados
- Garantia de restrições de integridade da aplicação
- Garantia de consistência física dos dados
- Potencial para imposição de padrões (modelagem e programação)
- Flexibilidade na definição e manutenção dos dados
- Redução do tempo de desenvolvimento de aplicações

Modelos de Dados

Conjunto de conceitos que podem ser usados para descrever o BD. Divide-se em Modelos Conceituais, Modelos de Implementação e Modelos Físicos.

Modelos Conceituais: Provêem conceitos próximos aos percebidos por muitos usuários. Usam conceitos como entidades, atributos e relacionamentos.

EX: Modelo ER, Modelo OO

Modelos de Implementação: Tem conceitos que podem ser entendidos pelos usuários e não estão muito distantes da maneira como os dados são organizados fisicamente. São usados freqüentemente em SGBD's comerciais. Representam os dados usando estruturas de registro.

EX: Modelo Relacional, Modelo Rede, Modelo Hierárquico.

Modelos Físicos: Descrevem como os dados são armazenados representando informação como formato de registros, ordenação de registros, métodos de acesso.

Arquitetura de um SGBD

A arquitetura divide-se em três níveis gerais:

Nível Interno → Mais próximo do armazenamento físico, isto é, relaciona-se com a forma como os dados são armazenados. Emprega-se o Modelo de Dados Físico para descrever detalhes de armazenamento.

Nível Conceitual → Descreve a estrutura completa de um BD para uma comunidade de usuários. É uma descrição global do BD, que esconde detalhes da estrutura física de armazenamento. Pode-se empregar um modelo de alto nível (modelo conceitual) ou de implementação.

Nível Externo → Mais próximo dos usuários. É formado por um conjunto de visões de usuários ou esquemas externos. Cada visão descreve a parte do BD que um grupo de usuários está interessado. Pode ser empregado um modelo de alto nível (modelo conceitual) ou de implementação

A maioria dos SGBD's não separa os 3 níveis completamente. Alguns incluem detalhes de nível físico no nível conceitual.

Notar que os 3 esquemas são apenas descrições de dados. Os dados que realmente existem estão no nível físico.

Estamos agora prontos para delinearmos uma arquitetura para um sistema de banco de dados. Nosso propósito, ao apresentar esta arquitetura, é estabelecer a base sobre a qual possamos trabalhar nos capítulos subseqüentes. Esta base é extremamente útil para descrevermos os conceitos gerais do banco de dados e explicarmos a estrutura de sistemas de banco de dados específicos - mas, certamente, não queremos afirmar que todo o sistema se encaixa nesta estruturação, nem sugerir que esta arquitetura, em particular, estabelece a única base possível. Os sistemas "pequenos" (com base em micro), especialmente, não suportarão todos os aspectos da arquitetura. Entretanto, parece que ela ajustase razoavelmente bem à maioria dos sistemas (relacionais ou outros); ademais, em termos gerais, concorda com a proposição do "ANSI/SPARC Study Group on Data Base Management Systems" [2.1-2.4]. Optamos, contudo, por não seguir a terminologia ANSI/SPARC em todos os detalhes.

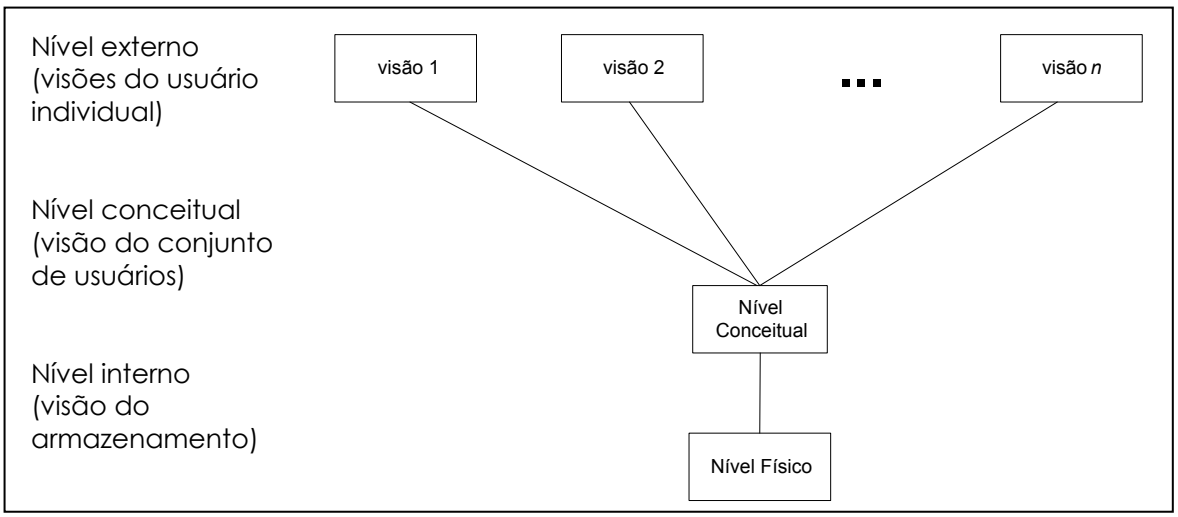
Os Três Níveis da Arquitetura

A arquitetura divide-se em três níveis gerais: interno, conceitual e externo. Em termos amplos:

- O nível *interno é* o mais próximo ao armazenamento físico - i.e. relaciona-se à forma como são realmente armazenados os dados;
- O nível *externo é* o mais próximo aos usuários - i.e., à forma como os dados são vistos pelos usuários individuais;
- O nível *conceitual é* o "nível de simulação", entre os dois outros.

Se o nível externo diz respeito às visões do usuário *individual, o* nível conceitual pode ser considerado a visão do *grupo* de usuários.

Em outras palavras, haverá muitas "visões externas" distintas, cada uma consistindo em uma representação mais ou menos abstrata de determinada parte do banco de dados e haverá precisamente uma "visão conceitual", que corresponde à representação abstrata do banco de dados em sua totalidade.' (Lembremo-nos que a maioria dos usuários não estará interessada em todo o banco de dados, mas somente numa parte restrita do mesmo.) Da mesma forma, haverá exatamente uma "visão interna", representando todo o banco de dados como armazenado de fato.



Os três níveis da arquitetura.

Um exemplo tornará estas idéias mais claras. A figura seguinte mostra a visão conceitual de um simples banco de dados sobre funcionários, a visão interna correspondente e as duas visões externas correspondentes, uma para um usuário de PL/1 e outra para o usuário de COBOL. O exemplo, na certa, é totalmente hipotético - não pretende simular qualquer sistema real - e muitos detalhes irrelevantes foram deliberadamente omitidos. Veja os esquemas abaixo:

```
        Externo (PL/1)                  Externo (COBOL)
        DCL    1 EMPP,                  01 EMPC.
               2 EMP # CHAR (6),        02 EMPNO PIC X (6)
               2 SAL FIXED BIN (31)     02 DEPTNO PIC X (4).

Conceitual
EMPLOYEE
EMPLOYEE_NUMBER          CHARACTER (6)
DEPARTAMENT_NUMBER       CHARACTER (4)
SALARY                          NUMERIC      (5)

Interno
STORED_EMP   LENGTH = 118
PREFIX  TYPE = BITE(6), OFFSET = O
EMP#    TYPE = BYTE(6), OFFSET = 6, INDEX = EMPX
DEPT#   TYPE = BYTE(4), OFFSET = 12
PAY     TYPE = FULLWORD, OFFSET = 16
```

Exemplos dos três níveis.

Interpretamos da seguinte maneira:

- Banco de dados contém, no nível conceitual, informações referentes ao tipo de entidade chamada EMPLOYEE. Cada EMPLOYEE tem um EMPLOYEE-NUMBER (seis caracteres), um

DEPARTMENT-NUMBER (quatro caracteres) e um SALARY (cinco dígitos decimais).

- Os funcionários estão representados, no nível interno, por um tipo de registro armazenado chamado STORED-EMP, com dezoito bytes de comprimento. O STORED-EMP contém quatro tipos de campos armazenados: um prefixo de seis bytes (contendo, provavelmente, informações de controle como sinalizadores ou ponteiros), e três campos de dados correspondendo às três propriedades dos funcionários' Os registros STORED_FMP são, adicionalmente, indexados no CAMPO EMP por um índice chamado EMPX.
- O usuário de PL/1 tem uma visão externa do banco de dados, na qual cada funcionário é representado por um registro PL/I, contendo dois campos (os números de departamento não são do interesse deste usuário, e por isto foram omitidos da visão). O tipo de registro é definido por uma declaração comum de estrutura PL/I, de acordo com as regras normais de PL/I.
- Do mesmo modo, o usuário de COBOL tem uma visão externa, na qual cada funcionário é representado por um registro COBOL, contendo, novamente, dois campos (desta vez foi omitido o de salário). O tipo de registro é definido por um registro comum COBOL, de acordo com as regras normais do COBOL.
- Observemos que objetos correspondentes podem ter nomes diferentes em cada nível. O número do funcionário, por exemplo, é chamado de EMP# na visão da PL/I, e de EMPNO na visão COBOL, como EMPLOYEE-NUMBER na visão conceitual e como EMP # (novamente) na visão interna. O sistema certamente deve estar a par das correspondências. Deve ser informado, por exemplo, que o campo COBOL EMPNO deriva do objeto conceitual EMPLOYEE-NUMBER que, por sua vez, é representado no nível interno pelo campo armazenado EMP#. Tais concordâncias, ou *mapeamentos,* não são mostradas no esquema acima.

Observação: Em um sistema relacional o primeiro nível, o nível conceitual, será definitivamente relacional, no sentido de que os objetos visíveis neste nível serão tabelas relacionais (os operadores também serão operadores relacionais, i.e., operadores que trabalham com tais tabelas). Uma determinada visão externa (segundo nível, nível externo) será igualmente relacional ou algo bem próximo; por exemplo, os registros PL/1 e COBOL da Fig. 1.3 podem ser considerados, respectivamente, como as representações PL/I e COBOL de (uma linha dentro de) uma tabela relacional. O terceiro nível, o nível interno, certamente não será sempre "relacional", desde que os objetos desse nível não serão exatamente tabelas relacionais (armazenadas) - ao contrário, serão a mesma espécie de objetos encontrados no nível interno de outros tipos de sistema (a saber, registros armazenados, ponteiros, índices, acessos hash etc.). De fato, a teoria

relacional como tal não tem nada a dizer sobre o nível interno (repetimos, sobre como o banco de dados aparece para o *usuário)*.

Examinaremos agora mais detalhadamente os três níveis da arquitetura, iniciando com o nível externo. A figura abaixo, à qual estaremos nos referindo neste capítulo, mostra os principais componentes da arquitetura e seus inter-relacionamentos.

Nível Externo

O nível externo é o nível do usuário individual. Um determinado usuário, como explicado no Capítulo I, tanto pode ser um programador de aplicações como um usuário de terminal on-line - i.e., um usuário final - de qualquer grau de sofisticação. O DBA é um caso especial importante. (Ao contrário dos usuários comuns, o DBA terá de se interessar pelos níveis conceitual e interno também. Vide as próximas duas seções.) Cada usuário tem uma *linguagem à* sua disposição:

- Esta linguagem, para o programador de aplicação, pode ser uma linguagem convencional de programação, como COBOL ou PL/I, ou uma linguagem de programação apropriada, específica do sistema em questão (sistemas NOMAD, Rdv/VMS ou dbase II).

- Para o usuário final, pode ser uma linguagem de consulta ou uma linguagem de propósitos especiais, talvez baseada em formulários ou menu, modelada às necessidades do usuário e suportada por um programa de aplicação on-line.

Para nossos propósitos, o que importa é sabermos que todas essas linguagens incluirão uma *sublinguagem de dados* - ou seja, um subconjunto de toda a linguagem, voltado para os objetivos e operações do banco de dados. A sublinguagem de dados (abreviada na Fig. 2.3 como DSL) é embutida na *linguagem hospedeira* correspondente, a qual proporciona os diversos recursos não-específicos de banco de dados, tais como variáveis locais (temporárias), operações computacionais, lógica if-then-else e assim por diante. Um determinado sistema pode suportar múltiplas linguagens hospedeiras e múltiplas sublinguagens de dados.

Observação: Embora seja conveniente diferenciar, na arquitetura, a sublinguagem de dados e linguagem hospedeira, as duas, em relação ao usuário, *podem ser* indistinguíveis. Se assim for, ou se só puderem ser separadas com dificuldades, dizemos que são "*unidades de maneira firme*". Se as linguagens são fáceis e claramente separadas, dizemos que *são "unidas de maneira indefinida*". A maioria dos sistemas de hoje suporta apenas a união indefinida. Um sistema firmemente unido propiciará um conjunto de recursos mais uniforme para o usuário, mas obviamente requer maior esforço por parte dos projetistas do

sistema (o que poderia explicar o que temos hoje). Entretanto, parece haver um movimento que, gradualmente, nos levará a dispor, nos próximos anos, de sistemas com união mais firme.

Em princípio, qualquer sublinguagem de dados é realmente uma combinação de pelo menos duas linguagens subordinadas: a *linguagem de definição de dados* (DDL), que possibilita a definição ou descrição dos objetos do banco de dados, e a *linguagem de manipulação de dados* (DML), que suporta a manipulação ou processamento desses objetos. Consideremos o usuário de PL/1 da **Fig. 2.2 de Seção 2.2**. A sublinguagem de dados para aquele usuário compõe-se dos aspectos de PL/1 utilizados para a comunicação com o DBMS. A parte DDL consiste nas construções declarativas do PL/I necessárias para declarar os objetos do banco de dados: a própria instrução DECLARE (DCL), certos tipos de dados PL/I, e possivelmente as extensões especiais para PL/I, para suportar novos objetos que não são tratados pelo PL/1 existente. A parte DML compõe-se das instruções executáveis do PL/I que transferem as informações de e para o banco de dados - podendo, novamente, incluir novas instruções especiais. (Observação: Os PL/I atuais de fato não incluem aspectos específicos de banco de dados. As instruções "DML", por isto, são apenas "CHAMADAS" para o DBMS. Eis por que sistemas PL/I, como muitos sistemas atuais, só proporcionam uma união muito indefinida entre a sublinguagem de dados e a hospedeira.)

Voltando à arquitetura: Já mencionamos que o usuário individual, por via de regra, só vai interessar-se por determinada parte do banco de dados; além disso, a visão que o usuário terá daquela parcela é, em geral, algo abstrato, em comparação à forma como os dados são fisicamente armazenados. Em termos ANSI/SPARC, a visão de determina do usuário é uma *visão externa.* A visão externa é, portanto, o conteúdo do banco de dados como visto por determinado usuário (ou seja, para aquele usuário, a visão externa é o banco de dados). Um usuário dó Departamento de Pessoal, por exemplo, pode ver o banco de dados como uma coleção de eventos de registros do departamento, e uma coleção de eventos de registros de empregados (podendo estar totalmente desinformado das ocorrências de registros de fornecedores e peças vistas pelos usuários do Departamento de Compras).

Portanto, uma visão externa consiste, em geral, em ocorrências múltiplas, de múltiplos tipos de *registro externo*'. Um registro externo *não* é necessariamente o mesmo que um registro armazenado. A sublinguagem de dados do usuário é definida em termos de registros externos; por exemplo, uma operação de "recuperação de registro" da DML irá recuperar um evento de registro externo, e não uma ocorrência de registro armazenado.

Cada visão externa é definida por meio de um *esquema externo,* que consiste, basicamente, em definições para cada um dos vários tipos de registro externo naquela visão externa. O esquema externo é descrito

usando-se a parte DDL da sublinguagem de dados do usuário. (Denomina-se, assim, a DDL, às vezes, de *DDL externa.) O* tipo de registro externo funcionário, por exemplo, pode ser definido como um campo de seis caracteres, número do funcionário, mais um campo de cinco dígitos decimais, 'salário', e assim por diante. Além disso, deve haver uma definição do *mapeamento* entre o esquema externo e o esquema conceitual fundamental (descrito na próxima seção).

Voltamo-nos agora para o nível conceitual.

Nível Conceitual

A *visão conceitual é* a representação de todo o conteúdo de informações do banco de dados, também (como a visão externa) um tanto abstrato quando comparada à forma como os dados são fisicamente armazenados, que também pode ser bem diferente da maneira como os dados são vistos por qualquer usuário em particular. A grosso modo, podemos dizer que a visão conceitual é a visão dos dados "como realmente são", e não como os usuários são forçados a vê-los devido às restrições (por exemplo) da linguagem ou do hardware utilizados pelos mesmos.

A visão conceitual consiste em ocorrências múltiplas de tipos múltiplos de *registros conceituais.* A mesma pode consistir, por exemplo, em uma série de ocorrências de registros de departamentos, mais um conjunto de ocorrências de registros de funcionários, ou de fornecedores de peças ... De um lado, um registro conceitual não é necessariamente o mesmo do que um registro externo e, de outro, nem o mesmo que um registro armazenado.

A visão conceitual é definida pelo *esquema conceitual* que inclui definições de todos os diversos tipos de registros conceituais. O esquema conceitual é escrito através de outra linguagem de definição de dados, a *DDL conceitual.* Para que se possa alcançar a independência de dados, essas definições da DDL conceitual não podem conter quaisquer considerações sobre a estrutura de armazenamento ou a estratégia de acesso - devem ser apenas definições das informações. Assim, não pode haver referência ao esquema conceitual para as representações do campo armazenado, seqüência de registro armazenado, indexação, acesso hash, ponteiros ou quaisquer outros detalhes de armazenamento/acesso. Se o esquema conceitual for realmente independente de dados, então os esquemas externos, que são definidos em termos do esquema conceitual, também serão necessariamente independentes de dados.

A visão conceitual, então, é a visão do conteúdo total do banco de dados, e o esquema conceitual é uma definição desta visão. Seria errado, porém, dizer-se que o esquema conceitual não é nada mais do que um conjunto de definições parecidas com simples definições de registros como encontrados, por exemplo, num programa COBOL. As definições no esquema conceitual devem incluir uma grande quantidade de aspectos, como controles de segurança e de integridade. Algumas autoridades na

matéria ainda vão mais além, e sugerem que o objetivo final do esquema conceitual é descrever toda a empresa - não somente os dados operacionais, mas também como são usados: como transcorre o fluxo em cada ponto da empresa, como é usado em cada ponto, como se aplicam à auditoria ou outros controles em cada ponto, e assim por diante [2.5]. Enfatizamos, no entanto, que hoje em dia nenhum sistema realmente suporta um nível conceitual que se aproxime deste grau de abrangência; na maioria dos sistemas existentes, o "esquema conceitual" é realmente pouco mais que uma simples união de todos os esquemas externos e individuais, com a provável adição de alguns controles de segurança e integridade. Mas tudo indica que os sistemas, no futuro, serão eventualmente mais sofisticados quanto ao suporte do nível conceitual.

Nível Interno

O terceiro nível da arquitetura é o nível interno. A *visão interna é* uma pequena representação de todo o banco de dados; consiste em ocorrências múltiplas de tipos múltiplos de *registros internos. O* "registro interno" é termo ANSI/SPARC para a estrutura que temos denominado de registro *armazenado* (termo que continuaremos a empregar). A visão interna é um tanto distante do nível físico, uma vez que não trabalha em termos de registros *físicos* (também chamados *páginas ou blocos),* nem de considerações de dispositivos específicos tais como cilindro ou comprimento de trilha. (A visão interna assume basicamente um espaço de endereçamento linear e infinito. Os detalhes de como este espaço de endereçamento é mapeado até o armazenamento físico são altamente específicos a cada sistema, e deliberadamente omitidos de arquitetura).

A visão interna é descrita por meio do *esquema interno,* que não só define os vários tipos de registros armazenados como também especifica os índices que existem, como os campos armazenados são representados, a seqüência física dos registros armazenados, e assim por diante. O esquema interno é preparado através de uma outra linguagem de definição de dados *- a DDL interna.*

Observação: Nesse livro, usaremos normalmente o termo "banco de dados armazenado" em vez de "visão interna", e o termo "definição de estrutura de armazenamento" em vez de "esquema interno".

Observamos, à parte, que, em certas situações excepcionais, os programas de aplicação - em particular, as aplicações de natureza "utilitária" - podem operar diretamente no nível interno, ao invés do nível externo. Não é necessário dizer que esta prática não é recomendável; representa um risco de segurança (posto que o controle de segurança não é observado) e um risco de integridade (uma vez que os controles de integridade também não são observados), e o programa, além disso, torna-se dependente de dados, em algumas ocasiões, porém esta poderá ser a única forma de se obter a função ou a performance necessária - assim

como o usuário de uma linguagem de programação de alto nível pode, ocasionalmente, precisar utilizar a linguagem de montagem (*assembler*) para satisfazer certos objetivos funcionais ou de desempenho.

Mapeamentos

O leitor poderá observar dois níveis de mapeamento na arquitetura, um entre os níveis externo e conceitual do sistema e um entre os níveis conceitual e interno. O ***mapeamento conceitual interno*** define a correspondência entre a visão conceitual e o banco de dados armazenado; especifica como os registros e campos conceituais são representados no nível interno. Se a estrutura do banco de dados armazenado for modificada - i.e., se for executada uma mudança na definição da estrutura armazenada -, o mapeamento conceitual/interno também deverá ser modificado **de** acordo, de forma que o esquema conceitual permaneça invariável. (O controle destas mudanças é da responsabilidade do DBA.) Em outras palavras, os efeitos dessas modificações devem ser isolados abaixo do nível conceitual, de maneira a preservar a independência de dados.

Um ***mapeamento externo conceitual*** define a correspondência entre uma determinada visão externa e a visão conceitual. As diferenças que podem existir entre estes dois níveis são similares àquelas que podem existir entre a visão conceitual e o banco de dados armazenado. Como exemplo, os campos podem ter tipos de dados diferentes, as denominações de campo e registro podem ser modificadas, campos conceituais múltiplos podem ser combinados num único campo externo (virtual) etc. Pode haver qualquer número de visões externas ao mesmo tempo; qualquer quantidade de usuários pode compartilhar de uma determinada visão externa; diferentes visões externas podem sobrepor-se. Alguns sistemas possibilitam que a definição de uma visão externa seja expressa em termos de outra (de fato, via mapeamento externo/externo), ao invés de exigirem sempre uma definição explícita do mapeamento a nível conceitual - um aspecto muito útil, quando diversas visões externas se relacionam entre si.

Linguagens de SGBD's

A linguagem de SGBD's é essencialmente SQL, que por sua vez é subdividido em grupos de comandos, de acordo com a função de cada comando. Infelizmente, estas subdivisões não são utilizadas por todas as implementações de SGBD's. Elas são enfatizadas pela ANSI (*American National Standards Institute*), mas muitos produtos SQL não as tratam separadamente na prática.

A Linguagem de Definição de Dados (ou DDL - *Data Definition Language*, ou também chamado pela ANSI como Linguagem de Definição de Esquema) consiste nos comandos que criam os objetos (tabelas, índices,

visões, e assim por diante) no banco de dados. Linguagem de Manipulação de Dados (DML - *Data Manipulation Language*) é um conjunto de comandos que determinam quais valores estão presentes nas tabelas em qualquer momento. A Linguagem de Controle de Dados (DCL - *Data Control Language*) consiste nas características que determinam se a um usuário é permitido executar uma ação particular. Isto é considerado parte da DDL pela ANSI. Elas (DDL, DML e DCL) não são linguagens diferentes por si só, mas divisões de comandos SQL em grupos de acordo com suas funções.

Então, resumindo,

DDL = Data Definition Language - Linguagem de definição de dados, possibilita a definição ou descrição dos objetos do BD.

DML = Data Manipulation Language - Linguagem de manipulação de dados, que suporta manipulação (acesso e alteração).

DCL = Data Control Language - Linguagem de controle de dados, possibilita a determinação das permissões que cada usuário terá sobre os objetos do BD (permissão de criação, consulta, alteração, etc.).

No SQL, por exemplo, são instruções DML:

SELECT - consulta (seleção)

UPDATE - atualização

DELETE - exclusão

INSERT - inclusão

DML's podem ser:

- **procedurais** - o usuário tem que especificar o que deseja e como pretende obter isso. Recuperam registros individuais do BD e processam cada registro separadamente. (devem ser embutidas em LPG's)
- **não procedurais** - são capazes de expressar operações complexas de maneira concisa. O usuário especifica apenas o que deseja, e permite que o sistema decida como obter isso. (pode ser embutida em uma LPG de propósito geral)

Independência de Dados

O Banco de Dados pode ser visto sob três níveis de abstração:

- Nível Físico → nível mais baixo de abstração, descreve como os dados estão realmente armazenados. Complexas estruturas de dados de baixo nível são descritas em detalhes.
- Nível Conceitual → descreve quais dados estão armazenados de fato no BD e as relações que existem entre eles. Aqui o BD inteiro é descrito em termos de um pequeno número de estruturas relativamente simples.
- Nível Visual → o mais alto nível de abstração descreve apenas parte do BD. Apesar do uso de estruturas mais simples do que no nível conceitual, alguma complexidade perdura devido ao grande tamanho do BD.

A habilidade de modificar a definição de um esquema em um nível sem afetar a definição de esquema num nível mais alto é chamada de ***independência de dados***. Existem dois níveis de independência de dados:

- ***Independência física de dados*** é a habilidade de modificar o esquema físico sem a necessidade de rescrever os programas aplicativos. As modificações no nível físico são ocasionalmente necessárias para aprimorar o desempenho.
- ***Independência lógica de dados*** é a habilidade de modificar o esquema conceitual sem a necessidade de rescrever os programas aplicativos. As modificações no nível conceitual são necessárias quando a estrutura lógica do BD é alterada.

Administrador de Banco de Dados

Uma das principais razões para empregar um SGBD é ter um controle central dos dados e dos programas de acesso a eles. A pessoa que tem esse controle sobre o sistema é chamada ***administrador do banco de dados* (DBA)**. O administrador do banco de dados (DBA) é a pessoa (ou grupo de pessoas) responsável pelo controle do sistema. As responsabilidades do DBA incluem o seguinte:

Decidir o conteúdo de informações do banco de dados →

Faz parte do trabalho do DBA decidir exatamente que informação manter no banco de dados - em outras palavras, deve identificar as entidades do interesse da empresa e a informação a registrar em relação a estas entidades.' Uma vez feito isto, o DBA deve então definir o conteúdo do banco de dados, descrevendo o esquema conceitual (usando o DDL conceitual). A forma objeto (compilado) daquele esquema é

utilizada pelo DBMS para responder às solicitações de acesso. A forma (não compilada) atua como documento de referência para os usuários do sistema.

Decidir a estrutura de armazenamento e a estratégia de acesso →

O DBA também deve decidir como os dados serão representados no banco de dados', e definir esta representação escrevendo a definição **da** estrutura de armazenamento (usando a DDL interna). Além disso, deve definir o mapeamento associado entre os níveis interno e conceitual. Na prática, tanto a DDL conceitual quanto a DDL interna - mais provavelmente a primeira - provavelmente incluirão os meios de definição desse mapeamento, mas as duas funções (a definição do esquema, a definição do mapeamento) devem ser claramente separadas. Tal como o esquema conceitual, o esquema interno e o mapeamento correspondente existirão não só na forma fonte como na forma objeto.

Servir de elo de ligação com usuários →

É função do DBA servir de elo de ligação com os usuários, a fim de garantir a disponibilidade dos dados de que estes necessitam, e preparar - ou auxiliá-los na preparação dos necessários esquemas externos, utilizando a DDL externa apropriada (como já mencionamos, um determinado sistema pode suportar diversas DDL's externas e distintas). E, ainda, também deve ser definido o mapeamento entre qualquer esquema externo e o esquema conceitual. Na prática, a DDL externa provavelmente incluirá os meios de especificação do mapeamento, mas o esquema e o mapeamento devem ser claramente separados. Cada esquema externo e o mapeamento correspondente existirão tanto na forma fonte como na forma objeto.

Definir os controles de segurança e integridade →

Os controles de segurança e de integridade, como já mencionado, podem ser considerados parte do esquema conceitual. A DDL conceitual incluirá os recursos para a especificação de tais controles. a Definir a estratégia de reserva e recuperação. A partir do momento em que a empresa começa efetivamente a basear-se em banco de dados, torna-se dependente do bom funcionamento deste sistema. Na eventualidade de danos à parte do banco de dados - causados, digamos, por erro humano, ou por falha no hardware, ou no sistema operacional de suporte -, é de suma importância fazer retornar os dados envolvidos com um mínimo de demora e com as menores conseqüências ao restante do sistema. Por exemplo, os dados que *não* sofreram danos não deverão ser afetados.' O DBA deve definir e implementar uma estratégia de recuperação apropriada envolvendo, por exemplo, o descarregamento

periódico do banco de dados na memória auxiliar de armazenamento e procedimentos para recarregá-lo, quando necessário.

Monitorar o desempenho e atender as necessidades de modificações →
O DBA deve organizar o sistema de tal maneira que obtenha "o melhor desempenho para a empresa"; e efetuar os ajustes adequados quanto às necessidades de modificações. Como já mencionamos, quaisquer mudanças nos detalhes de armazenamento e acesso devem ser acompanhadas de mudanças correspondentes na definição do mapeamento, a partir do nível conceitual, de forma que o esquema conceitual possa permanecer constante. O DBA precisa de diversos programas utilitários que o auxiliem nas tarefas de administração do BD. Estes são uma parte essencial de um sistema de banco de dados prático, embora não sejam mostrados na Fig. 1.3 da arquitetura. Abaixo estão alguns exemplos dos programas utilitários necessários.

Rotinas de Carga →
Para criar uma versão inicial do banco de dados a partir de um ou mais arquivos.

Rotinas de Despejo na Memória e Recuperação →
Despejar o banco de dados, auxiliar de armazenamento de dados, e recarregar o banco de dados a partir dessa cópia de segurança. Observação: As rotinas de carga mencionadas acima consistirão, na prática, no aspecto de "recuperação" das rotinas de despejo/recuperação na memória.

Rotinas de Reorganização →
Para rearrumar os dados no banco de dados, em vista de diversas razões de desempenho - por exemplo, agrupar os dados de certa maneira ou regenerar espaço ocupado por dados que se tornaram obsoletos.

Rotinas Estatísticas →
Para computar diversos desempenhos estatísticos, tamanhos de arquivos e distribuição de valores de dados.

Rotinas Analíticas →
Para analisar as estatísticas mencionadas. Uma das ferramentas mais importantes do DBA - de muitas maneiras e, de fato, o coração de todo o sistema - é *o dicionário de dados* (conhecido também como catálogo do sistema). O dicionário de dados pode ser considerado um banco de dados, um banco de dados de sistema. O conteúdo do dicionário pode ser considerado "dados sobre dados" (às vezes denominado "metadados") - ou

seja, *descrições* de outros objetos no sistema, ao invés de simples "dados em bruto".

Exercícios

1) Quais são os principais componentes dos sistemas de BD? Explique sucintamente a funcionalidade de cada uma.
2) Defina os termos:
3) Banco de Dados
4) Sistemas de Gerência de Bancos de Dados
5) Sistemas de Banco de Dados
6) Cite e explique algumas vantagens do uso de BD.
7) O que é Integridade Referencial e Transacional?
8) O que é Independência de Dados?
9) Descreva a arquitetura de SBD de 3 níveis.
10) Quais são os tipos de linguagens de SGBD? Para que serve cada uma?
11) Qual a diferença entre Independência Física e Lógica?
12) Cite e explique algumas responsabilidades do DBA.

MODELOS DE DADOS (MODELOS CONCEITUAIS E LÓGICOS)

Uma Coleção de ferramentas conceituais para descrição de dados, relacionamentos de dados, semântica de dados e restrições de consistência.

Classificação de Modelos de Dados

Os vários modelos de dados propostos atualmente, e encontrados na literatura, podem ser divididos em três (3) grandes grupos: modelos lógicos baseados em objetos (= Modelos Conceituais), modelos lógicos baseados em registros (= Modelos Lógicos ou Modelos de Implementação) e, modelos físicos de dados.

Modelos Lógicos Baseados Em Objetos - Modelos Conceituais

São usados na descrição de dados nos níveis conceitual e visual. Caracterizam-se pelo fato de fornecerem, de forma conveniente, capacidade de estruturação flexível e admitem restrições de dados para serem explicitamente especificados. Os mais conhecidos são:

Modelo Entidade-Relacionamento (MER);

Modelo Orientado a Objetos

O **Modelo ER** tem ganhado aceitação no projeto de BD e é bastante utilizado na prática.

O **Modelo OO** inclui muito dos conceitos no modelo ER, mas representa códigos executáveis assim como dados.

Modelos Lógicos Baseados Em Registro - Modelos Lógicos Ou De Implementação

São usados na descrição de dados nos níveis conceitual e visual. Em comparação aos modelos de dados baseados em objetos, ambos são usados para especificar a estrutura lógica geral do banco de dados e para fornecer uma descrição de alto nível da implementação.

São assim chamados porque o BD é estruturado em registros de formato fixo de diversos tipos. Cada tipo de registro define um número fixo de campos (atributos), e cada campo é usualmente de um tamanho fixo.

Desde o final dos anos 60, vários SGBD's comerciais foram construídos.....

Os três modelos mais aceitos são:

- Modelo Relacional;
- Modelo Rede;
- Modelo Hierárquico.

Modelo Relacional

O **Modelo Relacional** representa dados e relacionamentos entre dados por um conjunto de tabelas, cada uma tendo um número de colunas com nomes únicos. Veremos o Modelo Relacional com mais detalhes mais adiante.

nome	rua	cidade	número
Ana	Arvoredo	Porto Alegre	900
João	Norte	São Paulo	556
João	Norte	São Paulo	647
Carlos	Alameda	Florianópolis	801
Carlos	Alameda	Florianópolis	647

número	saldo
900	55
556	100.000
647	105.366
801	10533

Modelo Rede

Os dados no **Modelo Rede**, também conhecido como Modelo CODASYL ou DBTG, são representados por coleções de registros (como no Pascal ou PL/I), e os relacionamentos entre os dados são representados por elos, que podem ser vistos como ponteiros. Os registros no BD são organizados como coleções de gráficos arbitrários.

Um banco de dados rede consiste em uma coleção de registros que são conectados uns aos outros por meio de ligações. Um registro é em muitos aspectos similar a uma entidade no modelo ER. Cada registro é uma coleção de campos (atributos), cada um dos quais contendo apenas um

valor de dado. Uma ligação é uma associação entre precisamente dois registros. Assim, uma ligação pode ser vista como uma forma restrita (binária) de relacionamento no modelo ER.

Existem dois conceitos básicos de estruturas:

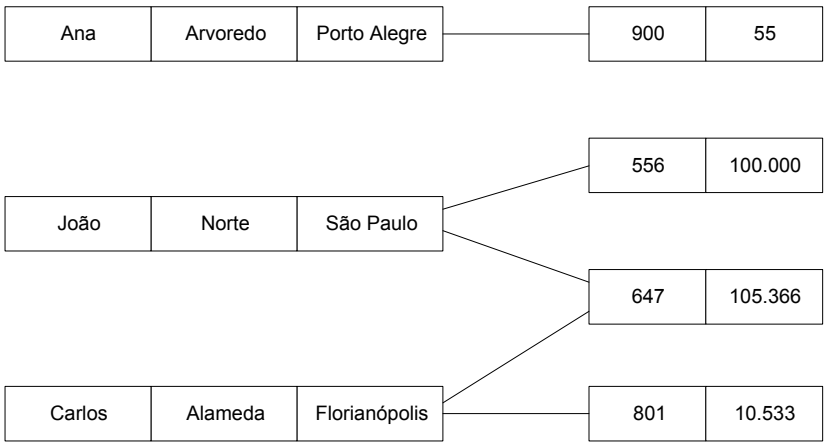
- registro (*"record type"*), onde cada registro descreve a estrutura de um grupo de registros que armazenaram o mesmo tipo de informação;
- ligação pai-filho (*"set type"*), onde cada ligação pai-filho é um relacionamento um-para-muitos (1:n) entre dois *"record types"*.

Cada registro representa alguma informação do mundo real sobre um grupo de itens, chamados itens de dados (ou atributos).

A figura abaixo mostra *"record types"* ESTUDANTE e CURSO, e um *"set type"* SE_MATRICULA entre eles, com ESTUDANTE como o registro-pai e CURSO como registro-filho. A representação diagramática mostra o uso de uma seta do registro pai para o registro filho, é chamada de **DIAGRAMA DE BACHMAN**, pois foi Charles Bachman que primeiro introduziu isto.

O conjunto de ocorrências inclui o registro-pai JOÃO e três registros-filhos correspondendo aos três cursos nos quais ele se matriculou. Um banco de dados pode conter muitas ocorrências da ligação pai-filho SE_MATRICULA, um por estudante. Note que se um estudante não é associado em qualquer curso, a ocorrência do registro ESTUDANTE ainda define um conjunto de ocorrências na qual existe um registro-pai e zero registros-filhos. A figura mostra os itens de dados contidos nos registros acima mencionados.

Sistemas de banco de dados rede permitem vetores, que são itens de dados que podem ter valores múltiplos dentro de um registro. Isto corresponde a um atributo multivalorado no modelo ER.



Exemplo de esquema de BD em rede – BATINI, 1992

Modelo Hierárquico

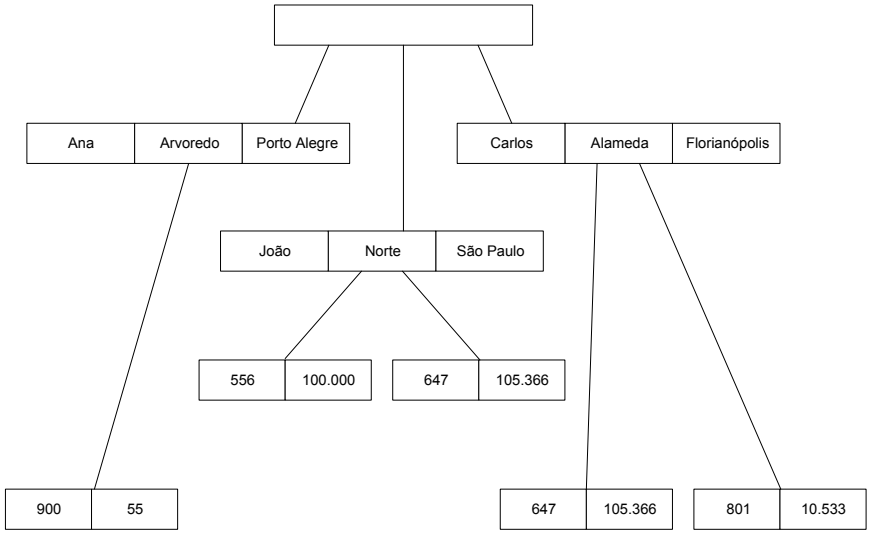
O primeiro SGBD comercialmente disponível e que se tornou popular era hierárquico, o BD da IBM, conhecido como IMS (*"Information Management System"*), que esteve no seu auge na década de 70. Como resultado, hoje existe uma grande base de dados que ainda utilizam a abordagem hierárquica.

O modelo hierárquico possui dois conceitos básicos de estruturas:

- Registro (*"record types"*), que é definido como um grupo de registros que armazenam informações de um mesmo tipo (no IMS, é chamado de segmento);
- Relacionamento pai-filho (*"parent-child relationship types"*), também chamado *"PCR type"* , é um relacionamento um-para-muitos entre um segmento-pai e um segmento-filho.

Um segmento (*"record type"*) possui muitas ocorrências, chamados registros. Uma ocorrência de um PCR consiste de um registro do segmento-pai e muitos (zero ou mais) ocorrências de um segmento-filho.

Um esquema de um BD hierárquico possui um número de hierarquias, onde cada hierarquia (esquema hierárquico) consiste de um número de segmentos e PCR's arranjados de maneira que forma uma árvore. A figura 2.4 mostra um esquema hierárquico com 2 segmentos e 1 PCR. Os PCR's são referidos como o par (segmento-pai, segmento-filho). Um registro pode somente possuir um segmento-pai. A figura mostra um possível estado de um BD IMS.



Esquema de um BD hierárquico – Heuser, 1995

Modelos Físicos de Dados

São usados para descrever dados no nível mais baixo. Existem poucos modelo físicos em uso. Dois dos mais conhecidos:

- Modelo Unificador;
- Estrutura de Memória.

Modelos Físicos de dados capturam aspectos da implementação de sistemas de BD.

Classificação de SGBD's

Quanto ao modelo de dados (principal critério):

- Relacional → representa o BD como coleção de tabelas;
- Hierárquico → representa o BD um conjunto de árvores;
- Rede → representa o BD como um grafo;
- Outros → OO, relacional extendido, semântico,...

Quanto ao número de usuários:

- mono-usuário;
- multi-usuário.

Quanto à distribuição dos dados e software:

- centralizado;
- distribuído.

Quando o SGBD é distribuído e multi-usuário, cada usuário percebe o BD como se ele fosse mono-usuário e centralizado (conceito de transparência em Sistemas Distribuídos).

Exercícios

1) Defina Modelo de Dados.
2) Como são classificados os Modelos de Dados?

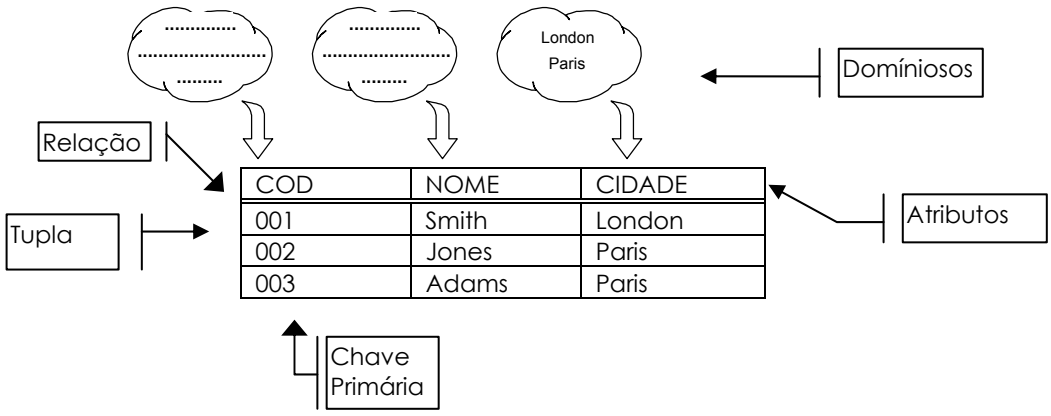
Modelo Relacional

Durante o período problemático dos modelos hierárquico e rede, Edward F. Codd (1970) propôs o *MODELO RELACIONAL*. A partir daí, diversas implementações de Bancos de Dados Relacionais (RDBS - SGBDR) começaram a aparecer.

Os Bancos de Dados Relacionais possuem diversas características importantes que resolvem muitos dos problemas dos outros modelos (hierárquico e rede). Ao contrário de seus antecessores, o Modelo Relacional não se baseia num paradigma de estruturação de dados particular, e sim em um fundamento matemático específico.

A visão de Banco de Dados mudou para um nível mais alto, concentrando seus princípios no modelo de negócios e não na implementação de estruturas complexas.

O Modelo Relacional representa o BD como uma coleção de ***relações***. Informalmente, uma relação é uma tabela ou arquivo simples.



COD	NOME	CIDADE
001	Smith	London
002	Jones	Paris
003	Adams	Paris

Domínio

A noção de domínio de um item de dado assemelha-se à noção de domínio de um conjunto na matemática. Um *Domínio* **D** é um conjunto de valores atômicos (indivisíveis). É o universo de valores permitido para um dado conjunto. Para tanto, é necessário o uso da DDL e da DCL do SGBD. Ainda, a definição de um domínio acarreta em certas operações e comparações de valor válidas para um item de dado.

Domínios podem ser simples, ou seja, possuem um único valor, como um inteiro, ou compostos, quando possuem vários valores, como uma data, que apresenta dia, mês e ano. Os valores dos itens de dados que

apresentam domínios compostos são abstraídos e tratados como um único valor. Uma data poderia ser manipulada como uma *string*, por exemplo.

O *Domínio* especifica o tipo do atributo:

- inteiros
- reais
- conjunto de cidades
- etc.

Atributo

Um *Atributo* representa o uso de um *Domínio* dentro de uma *Relação*. Vários *Atributos* podem pertencer a um mesmo *Domínio*.

Matematicamente, um atributo é o nome de um conjunto. Em uma tabela, representa uma coluna. Um atributo pode apresentar um valor condizente com o domínio associado a ele ou *null*. *Null* indica ausência de valor ou valor desconhecido.

Relação

Uma Relação dos *Domínios* ***D1***, ***D2***, ..., ***Dn*** compõe-se de:

- um conjunto fixo de ***atributos*** (esquema da relação) ***A1***, ***A2***, ..., ***An***, de forma que cada atributo ***Ai*** (i = *1*, *2*, ..., *n*) corresponda exatamente a um dos domínios básicos ***Di*** (colunas da tabela).
- um conjunto de ***tuplas*** (linhas da tabela)

Uma relação é vista como uma tabela no modelo relacional, composta por um cabeçalho e um corpo. O cabeçalho é formado por um conjunto fixo de atributos que possuem nomes distintos, para evitar ambigüidade na localização de um item de dado.

Uma Relação possui uma ***Cardinalidade*** (= número de *tuplas*, = linhas) e um ***Grau*** (= número de atributos, = colunas).

O conceito de relação é ligeiramente diferente do conceito de tabela. Relação equivale à noção de conjunto, ou seja, um agrupamento de elementos sem repetição. Tabela equivale a noção de coleção, ou seja, um agrupamento de elementos onde é permitida a repetição. SGBD's lidam, na prática, com o conceito de tabela.

"Uma relação **r** do esquema **R(A1, A2, ..., An)** é um conjunto de tuplas r={t1, t2, ..., tn}, cada tupla é uma lista de n valores t=<v1, v2, ..., vn> onde cada valor vi (i = 1, 2, ..., n) é um elemento do domínio dom(Ai) ou um valor nulo especial."

Ex.: ESTUDANTE

Matrícula	Nome	Endereço	
01	N1	E1	t1
------	------	Null	t2
...	...	...	...
------	------	------	tn

t1 = <01, N1, E1>

01 ∈ dom (matrícula)
N1 ∈ dom (nome)
E1 ∈ dom (endereço)

Chaves

O conceito de chaves é fundamental, pois permitem a identificação de tuplas em uma tabela, e permitem o estabelecimento de relacionamentos entre tabelas.

Tipos de Chaves:

- Chave Primária → Atributo ou combinação de atributos que permite a identificação única de uma tupla em uma relação. É uma ou mais colunas cujos valores distinguem uma linha das demais dentro de uma tabela. Uma chave primária não tem nenhuma ligação com a ordenação e com o acesso à tabela, pois, segundo CODD, declarar um atributo como chave primária e acessar a tabela por outro atributo, serve para manter duas restrições de integridade (vamos ver este conceito mais adiante).
- Chave Candidata → Em uma tabela, podemos identificar várias alternativas de identificador único, ou seja, vários atributos ou combinações de atributos que permitem a identificação única de uma tupla em uma relação. Estas chaves seriam várias *chaves candidatas* a serem chave primária. Toda chave primária é uma chave candidata, porém nem toda chave candidata é chave primária.
- Chave Alternativa → Do conjunto de chaves candidatas escolhemos somente uma para ser chave primária, as outras são chamadas de chaves alternativas.
- Chave Estrangeira → Atributo ou combinação de atributos de uma relação que são chaves primárias de outra relação. É uma coluna ou combinação de colunas, cujos valores aparecem necessariamente na chave primária de uma tabela. A chave estrangeira é o mecanismo que permite a implementação de relacionamentos em um BD Relacional.

Ex.: FUNCIONÁRIO (**COD#**, FNOME, ***DEPT#***, SALARIO)

DEPARTAMENTO (**DEPTNUM**, DEPTNOME)

Índices

Permitem a otimização da recuperação de tuplas em uma tabela, via um método de acesso. É um recurso físico, cujo principal objetivo está relacionado com a performance do sistema. Cada estrutura de índice está associada a uma chave de pesquisa particular.

Uma chave pode ser utilizada como índice, porém um índice não é necessariamente uma chave. Existem vários tipos de índice (que dependem do ambiente relacional que estamos trabalhando):

- Índices Primários → se um arquivo for organizado seqüencialmente, isto é, um arquivo seqüencial, o índice que possui uma chave de pesquisa que especifica esta ordem seqüencial, é chamado de índice primário. A chave de pesquisa de um índice primário é geralmente a chave primária. Os arquivos que são organizados em ordem seqüencial e possuem um índice primário, são chamados de arquivos seqüenciais-indexados.
- Índices Secundários → são os outros índices que possuem uma chave de pesquisa, e esta chave de pesquisa não especifica a ordem seqüencial do arquivo.

Dicionário de Dados

A estrutura de um banco de dados relacional é armazenada em um ***dicionário de dados*** ou ***catálogo do sistema***. O dicionário de dados é composto de um conjunto de relações, idênticas em propriedades às relações utilizadas para armazenar dados. Elas podem ser consultadas com as mesmas ferramentas utilizadas para consultar relações de tratamento de dados. Nenhum usuário pode modificar as tabelas de dicionário de dados diretamente. Entretanto, os comandos da linguagem de manipulação de dados que criam e destroem os elementos estruturais do banco de dados trabalham para modificar as linhas em tabelas de dicionários de dados.

Em geral, você encontrará os seguintes tipos de informações em um dicionário de dados:

- Definição de colunas que compõe cada tabela;
- Restrição de integridade imposta sobre relações;
- As informações de segurança (qual o usuário tem o direito de realizar o qual operação sobre qual tabela);
- Definições de outros elementos estruturais de um banco de dados, como visualizações e domínios definidos pelo usuário;

Quando um usuário e tentar acessar dados de qualquer maneira, um SGBD relacional primeiro dirige-se ao dicionário de dados para determinar se os elementos do banco de dados que o usuário solicitou fazem realmente parte do esquema. Além disso, o SGBD verifica se o usuário tem os direitos de acesso àquilo que ele está solicitando.

Quando um usuário tenta modificar dados, o SGBD também vai para o dicionário de dados afins de procurar restrições de integridade que podem ter sido colocadas na relação. Se os dados atendem às restrições, a modificação é permitida. Caso contrário, o SGBD retorna uma mensagem de erro e não faz a alteração.

Como todo acesso ao banco de dados relacional é feita através do dicionário de dados, dizemos que os SGBD's relacionais são baseados em um dicionário de dados.

Tabelas de Exemplo do Dicionário de Dados

As tabelas precisas que compõe um dicionário de dados dependem um pouco do SGBD. Nesta seção, você verá um exemplo de uma maneira típica em que um SGBD específico (SYBASE SQL ANYWHERE) organiza seu dicionário de dados.

O fecho do dicionário de dados é realmente uma tabela que documenta todas as tabelas do dicionário de dados. A partir dos nomes das tabelas do dicionário de dados, você provavelmente pode adivinhar que há tabelas para armazenar dados sobre tabelas básicas, suas colunas, seus índices e suas chaves estrangeiras.

creator	tname	dbspace	tabletype	ncols	Primary key
SYS	SYSTABLE	SYSTEM	TABLE	12	Y
SYS	SYSCOLUMN	SYSTEM	TABLE	14	Y
SYS	SYSINDEX	SYSTEM	TABLE	8	Y
SYS	SYSIXCOL	SYSTEM	TABLE	5	Y
SYS	SYSFOREIGNKEY	SYSTEM	TABLE	8	Y
SYS	SYSFKCOL	SYSTEM	TABLE	4	Y
SYS	SYSFILE	SYSTEM	TABLE	3	Y
SYS	SYSDOMAIN	SYSTEM	TABLE	4	Y
SYS	SYSUSERPERM	SYSTEM	TABLE	10	Y
SYS	SYSTABLEPERM	SYSTEM	TABLE	11	Y
SYS	SYSCOLPERM	SYSTEM	TABLE	6	Y

A tabela ***syscatalog*** descreve as colunas em cada tabela (incluindo as tabelas do dicionário de banco de dados). Na figura anterior, você pode ver uma parte de uma tabela ***syscolumns*** que descreve a tabela de itens de vendas da Lasers Only.

creator	cname	tname	coltype	nulls	lenth	InPrymaryKey	colno
DBA	item_numb	itens	integer	N	4	Y	1
DBA	title	itens	varchar	Y	60	N	2
DBA	distributor-numb	itens	integer	Y	4	N	3
DBA	release_date	itens	date	Y	6	N	4
DBA	retail_price	itens	numeric	Y	6	N	5

Lembre-se sempre de que essas tabelas de dicionário de dados têm a mesma estrutura e deve obedecer às mesmas regras que as tabelas básicas. Elas devem ter chave primária única não-nula e também devem por integridade referencial entre elas próprias.

Regras de Integridade Relacional

- Integridade da Entidade
- Integridade Referencial

As duas regras referem-se às chaves primárias e às chaves estrangeiras.

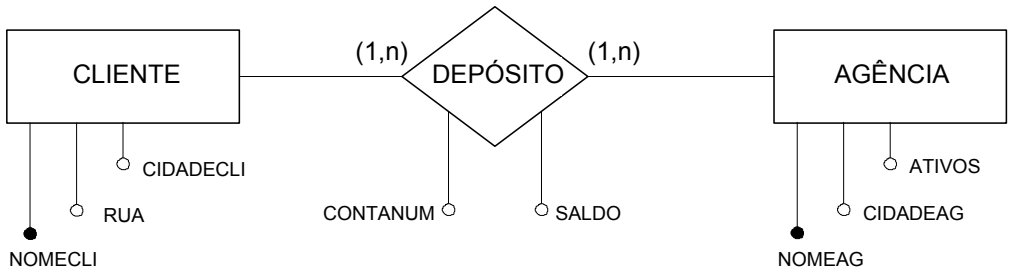
Regra de Integridade da Entidade:

Nenhum atributo de uma chave primária pode ter valor nulo.

Regra de Integridade Referencial:

Todas as chaves estrangeiras presentes em uma relação só podem ter valor igual a algum valor na relação onde ela é chave primária.

Por exemplo, vamos supor o seguinte Diagrama ER:



Este diagrama geraria as seguintes tabelas:

NOMECLI	RUA	CIDADECLI	NOMEAG
JOÃO	XYZ	WWWWW	DOWN
CARLOS	XXX	WWWWW	MIANUS
SILVA	XXX	WWWWW	MIANUS

CLIENTE

NOMEAG	CIDADEAG	ATIVOS
DOWN	YYYYYY	1000000
REDWOOD	ZZZZZZZZ	5000000
MIANUS	WWWWW	8000000

AGÊNCIA

NOMECLI	NOMEAG	CONTANUM	SALDO
JOAO	DOWN	155123	2000120,00
JOAO	REDWOOD	134556	255613,00
CARLOS	MIANUS	111111	125842,00
SILVA	MIANUS	548123	456982,00

DEPÓSITO

Implicações das Regras

1) O SGBD deve rejeitar uma operação que torne o BD não íntegro, ou;
2) O SGBD deve aceitar a operação, mas realizar outras operações compensatórias (se necessário).

Especificação de Banco de Dados Relacional

A especificação de uma base de dados relacional (chamada de esquema da base de dados) deve conter no mínimo a definição do seguinte:

- Tabelas que formam a base de dados
- Colunas que as tabelas possuem
- Restrições de integridade

Abaixo é mostrado o esquema correspondente às tabelas:

CODEMP	NOME	CODDEP	CATEGFUNC
.....	.....	.....	.....
.....	.....	.....	.....

EMP (CODEMP, NOME, CODDEP, CATEGFUNC)

CODDEP referencia DEPT

CODDEP	NOME
.....	.....
.....	.....

DEP (CODDEP, NOME)

Nesta notação, são listadas as tabelas e, para cada tabela, enumerados, entre parênteses, os nomes das colunas que compõe a tabela. As colunas que compõe a chave primária aparecem sublinhadas. Após a definição da tabela aparecem as definições das chaves estrangeiras que aparecem na tabela na forma:

<nome de coluna ch. Estrangeira> referencia <nome de tabela>

Ou, quando tratar-se de uma chave estrangeira composta por múltiplas colunas:

(<nome de coluna>$_1$, <nome de coluna>$_2$, ...) referencia <nome de tabela>

As 12 Regras de Codd

Em outubro de 1985, E. F. Codd publicou uma série de dois artigos no semanário da indústria de computação chamado *COMPUTER WORLD*. O primeiro artigo delimitava 12 critérios a que um banco de dados relacional deve obedecer integralmente. O segundo artigo comparava os atuais produtos para *mainframes* com esses 12 critérios, produzindo uma série de controvérsias sobre se os SGBD's devem ser teoricamente rigorosos ou se eles simplesmente devem trabalhar eficientemente.

Raros são os Bancos de Dados que se enquadrem em mais do que 10 destas regras.

As regras de Codd são:

Regra 1: as regras para informações

> *"Toda as informações em um banco de dados relacional são representadas explicitamente no nível lógico e exatamente de uma maneira - por valores em tabelas".*

O primeiro critério para bancos de dados Relacionais lida com a estrutura de dados que são utilizadas para armazenar dados e representar relacionamentos de dados. O objetivo dessa regra é fazer com que as relações (e tabelas de dimensionar esse) sejam as *únicas* estruturas de dados utilizadas em um banco de dados relacional. Portanto, produtos que requerem *links* codificados diretamente entre tabelas não são relacionais. A linha Independentemente de da sua posição com relação a esse argumento em particular há duas razões importantes por que é uma boa à idéia é criar um banco de dados e com nada além de tabelas.

- Relacionamentos lógicos são muito flexíveis. Em uma rede simples ou em um banco de dados hierárquico, os únicos relacionamentos que podem ser utilizados na recuperação são aqueles que foram predeterminados pelo projetista de banco de dados e que escreveu o esquema. Entretanto, uma vez que um banco de dados Relacional representa seus relacionamentos e por meio da correspondência com valores de dados, a operação de *join* pode ser utilizada para implementar relações instantaneamente, mesmo àquelas que um projetista de banco de dados talvez não tenha antecipado.

- Esquemas de banco de dados relacional são bem flexíveis. Você pode adicionar, modificar em remover relações individuais sem afetar o restante do esquema. O fato, contanto que não alterem a estrutura de tabelas que atualmente está sendo utilizada, você pode modificar o esquema de um banco de dados ao vivo. Entretanto, para modificar o esquema de uma rede simples o de um banco de dados e hierárquico, é preciso interromper todo o processamento do banco de dados e gerar novamente o todo o esquema. Em muitos casos, modificar o projeto de banco de dados também significa recriar todos os arquivos físicos (utilizando um processo de *dump*) para corresponder com o novo projeto.

Quando Codd originalmente escreveu suas regras, os bancos de dados não poderiam armazenar imagens. Hoje em dia, vários SGBD's oferecem a opção de ir armazenar imagens como BLOB's dentro do banco de dados ou como nomes de caminho ou OVFLS para arquivos de imagens que são armazenados fora do banco de dados. Tecnicamente, nomes de caminho ou *URLS* para arquivos externos são ponteiros para algo além de tabelas e, portanto, aparentemente fariam com que um SGBD viola-se essa regra. Entretanto, o espírito da regra é que relacionamentos entre entidades - os relacionamentos lógicos no banco de dados - são representados

correspondendo valores de dados, sem o uso de ponteiros de qualquer tipo para indicar conexões entre entidades.

Regra 2: a regra de acesso garantido

> *"Cada e todos os dados (valor atômico) em um banco de dados relacional têm a garantia de serem logicamente acessíveis pela reclassificação de uma combinação de nomes de tabelas, valor de chave primária e nome de coluna".*

Como a principal razão pela qual em seremos os dados em um banco de dados é para que possamos recuperá-lo os novamente, devemos estar certos de que poderemos recuperar todos dados possíveis.

Essa regra afirma que você deve conhecer apenas três aspectos para localizar um dado específico: o nome da tabela, o nome da coluna e a chave primária da linha.

Não há nenhuma regra nesse conjunto de doze regras e que afirme categoricamente que cada linha em uma relação de Evans ter uma chave primária única. Entretanto, uma relação não pode obedecer à regra de acesso garantido, a menos que ela tenha uma chave primária única. Sem uma chave primária única, você recuperará algumas linhas com o valor da chave primária utilizado em uma pesquisa, mas não necessariamente a linha exata desejada. Portanto, alguns dados talvez não estejam acessíveis sem a capacidade única de identificar linhas.

Antigos bancos de dados Relacionais não requeriam um qualquer chave primária. É possível criar é eu utilizar tabelas sem restrições de chave primária. Hoje em dia, entretanto, a SQL permite criar uma tabela sem especificação de uma chave primária, mas a maioria dos SGBD's não permite que você em si já dados nessa tabela.

Regra 3: tratamento sistemático de valores nulos

> *"Valores nulos (distintos da e string de caracteres vazia ou de uma string de caracteres em branco ou de qualquer outro número) são suportados completamente em um SGBD relacional para representar de maneira sistemática as informações ausentes, independentemente do tipo de dados".*

Como você sabe, nulo é um valor de banco de dados especial que significa " desconhecida". A presença dele em um banco de dados traz problemas especiais durante recuperação de dados. Por exemplo, considere o que acontece você tiver uma relação de funcionários que contém uma coluna para falar. Assuma que o salário é nulo em algumas

partes das linhas. O que deve acontecer se alguém consultar a tabela para todas as pessoas que ganham mais de US$ 60.000? As linhas com nulos devem ser recuperadas ou permanecer fora?

Quando SGBD avaliam nulo em relação ao critério lógico de valor de um salário maior que US$ 60.000, ele não pode afirmar se a linha contendo nulo cumprir com os critérios. Talvez compra; talvez não. Por essa razão, dizemos que bancos de dados relacionais utilizam lógica trivalorada. O resultado da avaliação de uma expressão lógica é verdadeiro, falso ou talvez.

Primeiro, com SGBD relacional deve armazenaram o mesmo valor de nulo em todas as colunas e linhas onde o usuário não insere explicitamente os valores de dados. O valor utilizado para o nulo deve ser o mesmo, independentemente do tipo de dados da coluna. Observa que o nulo não é o mesmo que um caractere de espaço em branco; ele tem seu próprio valor ASCII ou UNICODE distinto. Entretanto, na maioria das vezes quando você vê na tela uma tabela de resultados de uma consulta, nulos não aparecem como valores em branco.

Segundo, o SGBD deve ter alguma maneira conhecida e coerente de tratar desses nulos ao realizar consultas. Em geral, você descobrirá que linhas com nulos não são recuperadas por uma consulta como a do exemplo de salário maior que US$ 60.000, a menos que o usuário solicite explicitamente linhas com um valor de nulo. Atualmente, a maioria dos SGBD's relacionais obedece ao na tabela de verdade lógica trivalorada para determinar o comportamento de recuperação quando eles encontram nulos.

A inclusão de nulos em uma relação pode ser extremamente importante. Eles oferecem uma maneira consistente de extinguir entre dados válidos, como um 0, e dados ausentes. Por exemplo, faz muita diferença saber que o soldo de uma conta a pagar é 0 em vez de desconhecido. A conta com começaram número zero cancelar é algo que gostamos de ver; a conta com saldo desconhecido poder ser um grande problema.

Regra 4: catálogo on-line dinâmico baseado no modelo relacional

> *"A descrição de banco de dados é representada no nível lógico da mesma maneira que dados simples, de modo que usuários autorizados possam aplicar a mesma linguagem relacional que aplicam a dados regulares a sua interrogação".*

Uma vantagem da utilização de estruturas de dados idênticas para um dicionário de dados, como ocorre com tabelas de dados, é que você tem uma maneira consistente para acessar todos os elementos do banco de dados. Você precisa aprender apenas uma linguagem de consulta. Isso

também simplifica o próprio SGBD, uma vez que ele pode utilizar o mesmo mecanismo para tratar dados sobre o banco de dados (metadados) que ele utiliza para dados sobre a organização.

Regra 5: a regra da sublinguagem de dados abrangentes

> *"Um sistema relacional pode suportar várias linguagens e vários modos de utilização de terminal (por exemplo, modo de preenchimento de espaços em branco). Entretanto, há pelo menos uma linguagem cujas instruções podem ser descritos por alguma sintaxe bem definida, como string de caracteres, e que seja abrangente para suportar a todos os seguintes itens:*
>
> - *A definição dos dados*
> - *Definição da visualização*
> - *Manipulação de dados*
> - *Restrições de integridade*
> - *Limites de transação*

Um banco de dados relacional deve ter alguma linguagem que possa manter elementos estruturais do banco de dados, modificar em recuperar dados. Como a maioria dos atuais SGBD as relacionais utilizam SQL como sua principal linguagem de manipulação de dados, parece não haver nenhum problema aqui.

Entretanto, com SGBD que não suporta SQL, mas utiliza uma linguagem gráfica, tecnicamente não atenderia essa regra. Não obstante, a vários produtos hoje em dia cuja linguagem gráfica pode realizar o todas as tarefas que Codd estou sem uma sintaxe de linha de comando. Teoricamente, esses SGBD as talvez não seja "completamente relacionais", mas uma vez que eles possam realizar todas as tarefas relacionais necessárias, você não perde nada se não tiver a linguagem de linha de comando.

Regra 6: a regra da visualização de atualização

> *"Todas às vezes realizações que teoricamente são atualizáveis são também atualizáveis pelo sistema".*

Essa regra simplesmente significa que se uma visualização atender aos critérios de atualização, com SGBD deve ser capaz de tratar dessa atualização e propagar as atualizações de volta às tabelas básicas.

Regra 7: inserção de alto nível, atualização e exclusão

> *"A capacidade de tratar uma relação básica o uma relação derivada como um único operando é aplicada não apenas a recuperação de dados, mas também à inserção, atualização e exclusão de dados*".

Codd quis assegurar que o SGBD pudesse tratar ao mesmo tempo de múltiplas linhas de dados, especialmente quando os dados fossem modificados.

A linguagem SQL fornece essa capacidade para os atuais SGBD's relacionais. O que isso traz para você? Poder modificar mais de uma linha com um único comando simplificar lógica da manipulação de dados. Por exemplo, em vez de precisar ler linha por linha de uma relação para localizar as linhas para modificação, você pode especificar critérios lógicos que identificam as linhas a serem modificadas e deixar que o SGBD encontre as linhas para você.

Regra 8: independência de dados físicos

> *"Atividades de programas de aplicações e atividades de terminais permanecem logicamente intactas sempre que alguma alteração ocorrer na representação de armazenamento e o nos métodos de acesso*".

Um dos benefícios da utilização de um sistema de banco de dados, em vez de um sistema de processamento de arquivos, é que o SGBD isola usuário de detalhes de armazenamento físico.

Isso significa que você pode mover o banco de dados a partir de um volume de disco para outro, alterar o *layout* físico dos arquivos e assim por diante, sem causar nenhum impacto na maneira como os programas de aplicações e os usuários finais interagem com as tabelas do banco de dados.

A maioria dos SGBD's atuais oferece pouco controle sobre as estruturas de arquivos utilizadas para armazenar dados no disco. Portanto, na prática, independência de dados físicos significa que você pode mover o banco de dados e a partir de um volume de discos ou diretório para outro sem afetar as aplicações ou os usuários interativos. Com algumas exceções - em particular, SGBD's deu usuário final baseados no modelo dBase - a maioria dos SGBD's atuais fornecem independência de dados físicos.

Regra 9: independência de dados lógicos

> *"Programas de aplicações e atividades de terminais permanece logicamente intactos quando são feitas alterações de qualquer tipo as tabelas básicas que preservam informações que teoricamente permitem a estabilidade dos dados*".

Independência de dados lógicos é um pouco mais sutil que a independência de dados físicos. Em essência, significa que se você alterar o esquema, talvez adicionando ou removendo uma tabela ou adicionando uma coluna para uma tabela, as outras partes do esquema que não devem ser afetadas pela alteração permanecem inalteradas.

Como um exemplo, considere o que acontece quando você adiciona uma tabela a um banco de dados. Uma vez que as relações são logicamente independentes entre si, adicionar uma tabela não causa absolutamente nenhum impacto sobre qualquer tabela. Para obedecer à regra de independência de dados lógicos, com SGBD deve assegurar que de fato não há nenhum impacto sobre outras tabelas.

Por outro lado, se você escolher uma tabela do banco de dados, essa modificação não é *"de preservação de informações"*. É quase certo que os dados serão perdidos quando a tabela for removida. Portanto, não é necessário que aplicações e usuários interativos não serão afetados pela operação.

Regra 10: independência de integridade

> *"Restrições de integridade específicas para um banco de dados relacional em particular devem ser definido vez na sublinguagem de dados Relacionais e passíveis de serem armazenados no catálogo, não nos programas de aplicações"*.

No mínimo duas das seguintes restrições de integridade e devem ser suportadas:

1. Integridade de entidade: nenhum componente de uma chave primária tem permissão para ter um valor nulo
2. Integridade referencial: para cada valor de chave estrangeira não nulo e distinto em um banco de dados relacional, deve existir um valor de chave primária e que corresponda ao mesmo domínio.

Observe que essa regra requer que a declaração de restrições de integridade faça parte de qualquer que seja a linguagem utilizada para definir a estrutura do banco de dados. Além disso, qualquer tipo de restrição de integridade deve ser armazenado em um dicionário de dados que possa ser acessado enquanto o banco de dados é utilizado.

Quando a IBM lançou seu carro-chefe de banco de dados relacional – o DB/2 – uma entre duas reclamações dos usuários foi a falta de suporte para integridade referencial. Quanto a isso, a IBM e outros fornecedores de SGBD omitiram a integridade referencial, pois ela reduzia o desempenho. Todas as vezes que você modificar uma linha de dados, o SGBD deve ir ao dicionário de dados, procurar uma regra de integridade e realizar o teste indicado pela regra, tudo isso antes de realizar uma atualização. Uma verificação de integridade referencial de uma única coluna pode envolver dois ou mais acessos ao disco, que leva mais tempo do que simplesmente fazer a modificação diretamente na tabela básica.

Entretanto, sem a integridade referencial os relacionamentos em um banco de dados relacional tornam-se rapidamente inconsistentes. Portanto, consultas de recuperação não necessariamente recuperam todos os dados, pois referências cruzadas ausentes fazem com que *joins* omitam dados. Nesse caso, o banco de dados não será confiável e praticamente inutilizável. (Tudo bem, a IBM adicionou integridade referencial ao DB/2 bem rapidamente!)

Regra 11: independência de distribuição

> *"Um SGBD relacional tem independência de distribuição".*

Um banco de dados distribuído é um banco de dados onde os próprios dados são armazenados em mais de um computador. Portanto, o banco de dados é a união de todas as suas partes. Na prática, as partes não são únicas, mas contêm um grande volume de dados duplicados.

Em outras palavras, um banco de dados distribuído deve parecer ao usuário como uma banco de dados centralizado. As aplicações e os usuários interativos não precisam conhecer o local de armazenamento dos dados, incluindo a localização de múltiplas cópias dos mesmos dados.

Regra 11: regra da não-subversão

> *"Se um sistema relacional tiver uma linguagem de baixo nível (um único registro de cada vez), essa linguagem de baixo nível não pode ser utilizada para subverter ou pular as regras de integridade ou restrições expressas na linguagem relacional de nível mais alto (múltiplos registros de cada vez)".*

Muitos produtos de SGBD na década de 1980 tinham linguagens que poderiam acessar diretamente as linhas nas tabelas, separadamente da SQL, que opera em múltiplas linhas de cada vez. Essa regra afirma que não há nenhuma maneira de utilizar essa linguagem de acesso direto para evitar as restrições de integridade armazenadas no dicionário de dados. Todas as regras de integridade devem ser observadas.

Destas doze regras, pelo menos seis devem ser cumpridas para que o SGBD possa ser qualificado como completo relacionalmente.

Integridade

Controle de Integridade Semântica do Banco de Dados

O termo integridade é utilizado em banco de dados com o seguinte significado: precisão, correção, validade. O grande problema da integridade é o de assegurar que os dados no banco de dados se mantenham precisos, corretos, validos. Ou seja, o de preservar o banco de dados contra atualizações/inserções não válidas (controle de acesso).

Se o objetivo do controle de acesso ao BD é o de evitar que pessoas e/ou programas não autorizados leiam e/ou atualizem a base de dados, então é objetivo dos mecanismos que zelam pela integridade semântica do BD garantir que somente atualizações ***permitidas*** sejam executadas na base de dados.

Por "*permitidas*", entenda-se as modificações que mantenham a relação entre o mini-mundo (do usuário/da aplicação) e a representação deste no BD.

Requisitos (restrições) de Consistência (*Consistency Constraints*) dizem respeito à consistência dos dados em sua representação nos níveis mais baixos da arquitetura do SGBD.

Exemplo: no endereço de disco (cilindro, trilha, bloco físico) deve estar armazenado o bloco de dados lógico (página) correto (correspondente).

Em níveis mais altos da arquitetura do SGBD, o número de requisitos de consistência a serem preservados de multiplica.

Exemplo: os endereços de registros apontados por um arquivo de índice devem sempre refletir o estado atual do arquivo indexado.

Requisitos de Consistência dos níveis mais baixos da arquitetura do SGBD podem ser derivados durante seu projeto e não dependem de uma aplicação particular (são válidos para qualquer aplicação).

No nível mais alto da arquitetura (nível das estruturas de dados lógicas - modelo externo, dicionário de dados) devem ser levados em conta e assegurados os relacionamentos entre o mini-mundo no qual o usuário baseia seu trabalho e o "mapeamento" deste mini-mundo no BD. Este mapeamento é específico e dependente de cada aplicação.

A partir da definição, por parte do usuário, dos estados corretos e das transições de estado corretas no seu mini-mundo, o SGBD deve garantir que o BD reflita somente estes estados e que as operações sobre o BD reflitam somente estas transições.

A garantia da integridade semântica é a garantia de que o estado dos dados do BD está sempre coerente com a realidade para o qual o mesmo foi projetado e criado. Não basta ter um esquema com os dados eficientemente bem estruturados, se não existir nenhum controle sobre os valores dos mesmos (falta de confiabilidade). Se não existe gerenciamento de integridade semântica, pode-se ter violações de domínio (valores não condizentes com a semântica de certo atributo - por exemplo, um empregado com idade de 10 ou 100 anos), dados desconhecidos (ausência de valor em atributos significativos, como nome do empregado) e relacionamentos incorretos ou inexistentes (por exemplo, a ocorrência de situações proibidas, como departamentos sem gerente ou alguém sendo gerente de mais de um departamento).

Garantir integridade semântica é garantir estados corretos de um dado (verificado sempre após alguma atualização no BD) e transições de estado também válidas (às vezes, uma atualização no BD acarreta outras ações de atualização sobre outros dados - transparentes para a aplicação - de forma a manter a integridade do esquema).

Um subsistema de integridade semântica de um SGBD controla restrições especificadas sobre um esquema. Estas restrições (ou regras) são chamadas restrições de integridade (RI's). O gerenciamento destas restrições envolve três funções básicas:

- Especificação de RI's: deve ser possível especificar testes e ações para garantia de integridade. Para tanto, o SGBD deve suportar uma linguagem de especificação de restrições (DCL), cujos comandos são compilados e mantidos no catálogo do sistema (DD) para posterior consulta e modificação;
- Monitoramento de RI's: sempre que ocorrer uma operação de atualização sobre um dado, todas as RI's que dizem respeito ao mesmo devem ser consultadas no DD, para verificar se alguma delas não foi violada ou se algo mais (por exemplo, outra atualização ou a execução de um procedimento qualquer) deve ser realizado;
- Ações para garantia de RI's: quando alguma operação de atualização gera alguma inconsistência, devem ser tomadas ações como o impedimento da atualização (*rollback* da atualização) ou a execução de outros procedimentos para atualizar dados relacionados.

Assim sendo, em toda RI deve ser possível especificar:

- Para quais dados deve-se verificar a regra (alcançe da regra);

- Quando a regra deve ser verificada (antes, imediatamente após ou um tempo após a operação de atualização);
- Que ação deve ser tomada para garanti-la.

A existência de um subsistema de integridade semântica é extremamente vantajosa para as aplicações que utilizam um SGBD, pois existe total independência deste controle, ou seja, o gerenciamento de integridade fica a cargo do SGBD. As aplicações ficam liberadas desta pesada tarefa, sendo que estas apenas submetem operações ao SGBD e recebem indicação de O.K. ou sinalizações de violações e atitudes tomadas para garanti-las.

Classificação dos Requisitos de Integridade (RI)

Os requisitos de integridade podem ser classificados segundo os seguintes aspectos:

- Segundo o Seu Alcance (volume de informação associada à RI)

  a) Atinge um único atributo de uma relação.

  Ex.: ***e-nome*** só pode conter letras ou brancos (RI1)

  b) Atinge mais de um atributo de uma mesma tupla

  Ex: s-total ≤ s-orçamento (RI2)

  c) Atinge mais de uma tupla (um registro) do mesmo tipo (mesma relação)

  Ex.: ***e-no*** deve ser unívoco (cada tupla em **EMP** tem valor diferente em ***e-no***) (RI3)

  d) Atinge mais de uma tupla que podem ser de tipos diferentes

  Ex.: s-totsal (var_seção) = ∑ e-sal (i) ∧ e-sec (i) = var_seção (RI4)

- Segundo o Momento do Teste da Restrição

  a) **Instantânea** → sempre que o dado associado à RI for inserido ou atualizado, a RI deve ser testada.

  Ex.: ***e-no*** ≥ '0001' ∧ ***e-no*** ≤ '9999' (RI5)

b) **Postergada →** a RI só pode ser testada após uma série de operações sobre o BD (típico para RI's que atingem mais de uma tupla de relações diversas).

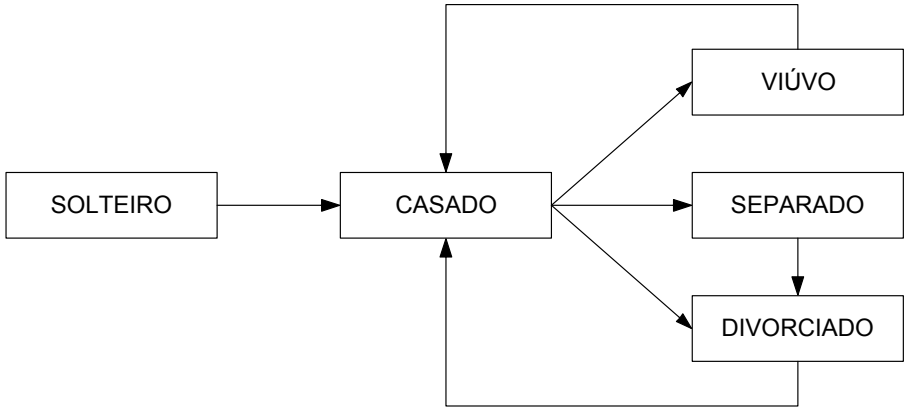
Ex.: O usuário deseja aumentar o salário de *alguns* empregados (da mesma seção). A RI4 só necessita ser testada ao final das atualizações. (performance!!)

RI's Que Regulamentam a Atualização de Valores

- RI's baseadas em Versões de Valores de Dados.
- Testes levam em conta a relação (valor_antigo, valor_novo).

Exemplo 1: O salário do empregado não pode ser rebaixado (RI6)
e-sal (novo_valor) ≥ ***e-sal*** (valor_atual)

Exemplo 2: As transições do estado civil do empregado devem respeitar o seguinte grafo de precedências:



RI's Que Devem Ser Testadas na Ocorrência de Eventos Externos

O momento do teste da RI é determinado pela aplicação.

Exemplo: A partir de três meses de sua data de ingresso na empresa, o salário do empregado deve refletir o salário básico estipulado para sua função (piso salarial da categoria).

(3 meses = período de experiência) (RI7)
(data_atual - e-dataing ≥ 90) ⇒ e-sal ≥ sal_base (e-func)

Métodos Utilizados no Suporte à Especificação de Restrições de Integridade

a) Oferecimento de mecanismos embutidos na linguagem de manipulação de dados (DML para a definição das RI's. (A DML pode ser procedural, (ex.: CODASYL) ou descritiva (ex.: SQL, QUEL, QBE)).

b) Especificação de RI associadas ao objeto e integradas à especificação do próprio objeto com base no conceito de tipos abstratos de dados (ADT) (proposta nova).

Definição e Teste de Restrições de Integridade

→ Em SGBD's "Mais Antigos" (Não Relacionais)

- Restrições de Integridade no DBTG (modelo em Rede):

  As RI's são como *procedures* em uma linguagem baseada em COBOL. O DBTG oferece uma cláusula de sua linguagem de declaração que associa operações sobre a base de dados com *procedures* que testam RI's.

  ON < command_list> CALL <procedure>

  Esta cláusula é associada a declarações de tipos de registros e sets:

  <command_list> C {insert, delete, modify, find,...}

  Exemplo: Teste de RI5 durante a inserção de um novo empregado:

```
PROCEDURE P1:
    IF (e_nr < '0001' OR e_nr > '9999')
    THEN GO TO TRATA_ERRO_RI5
END P1;
DEFINE RECORD_TYPE EMP
e_no : CHAR(4);
...
ON INSERT CALL P1
END;
```

→ Em SGBD's Relacionais

- RI's em SQL → definição com o uso da DML

- Alternativa 1: através da cláusula ASSERT

  ASSERT RI5 ON EMP:
  e-no BETWEEN '0001' AND '9999' IMMEDIATE
  ASSERT RI2 ON SEÇÃO:
  s-totsal ≤ s-orçam
  ASSERT RI4 ON SEÇÃO X:
  s-totsal = (SELECT SUM (***e-sal***) FROM EMP WHERE ***s-sec***=x.***s-no***)
  ASSERT RI6 ON EMP (***e-sal***):
  NEW ***e-sal*** ≥ OLD ***e-sal***

  RI's especificadas com ASSERT tem seu teste postergado até o fim do trabalho do usuário, a não ser que estejam associadas à cláusula IMMEDIATE.

- Alternativa 2: através do conceito de TRIGGER

  Permite que o sistema execute automaticamente uma série de operações quando uma determinada operação é executada pelo usuário (lembra DBTG).

  TRIGGER: Evento → Teste → Procedimento

  DEFINE TRIGGER T1:
  ON UPDATE EMP x (***e-sal***)
  (UPDATE SEÇÃO
  SET ***s-totsal*** = ***s-totsal*** + (NEW x.***e-sal*** - OLD x.***e-sal***)
  WHERE ***s-no*** = x.***e-sec***)

Exemplo de Manutenção de Restrição de Integridade Com e Sem TRIGGER (SQL)

- MANUTENÇÃO SEM USO DE TRIGGER (tudo programado pelo usuário)

  ASSERT RI4 ON SEÇÃO X: /* X representa cursor */
  s-totsal = (SELECT SUM (***e-sal***)
  FROM EMP
  WHERE ***e-sec*** = x.***s-NO***) /* postergado */
  Begin-Of-Transaction /* início do programa do usuário */
  UPDATE EMP
  SET ***e-sal*** = ***e-sal*** + ***e-sal*** * 0.1 /* aumento de 10% */
  WHERE ***e-sec*** = 'SETOR B'
  UPDATE SEÇÃO
  SET s-totsal = s-totsal + s-totsal * 0.1
  WHERE ***s-nr*** = 'SETOR B'
  End-Of-Transacton /* fim do programa do usuário */

No final do programa, a cláusula *ASSERT* é automaticamente avaliada.

- MANUTENÇÃO COM USO DE *TRIGGER* (atualização automática de *s-totsal)*

```
DEFINE TRIGGER T1:
ON UPDATE EMP x (e-sal)
(UPDATE SEÇÃO
SET s-totsal = s-totsal + (NEW x.e-sal - OLD x.e-sal)
WHERE s-no = x.e-sec)
Begin-Of-Transaction      /* início do programa do usuário */
UPDATE EMP
SET e-sal = 1.1 * e-sal
WHERE e-sec = 'Setor B'
```

  - ♦ Neste ponto do programa, o SGBD executa T1, mantendo RI4

```
End-Of-Transaction        /* fim do programa do usuário */
```

Outros programas podem fazer uso do mesmo *Trigger* T1.

***Triggers* Podem Causar Problemas**

- O programador deve conhecer todos os *triggers* declarados no SBD, caso contrário programará de forma redundante, podendo provocar inconsistências no BD;
- A definição de novos *triggers* pode implicar na alteração de programas de aplicação já existentes;
- Já que *triggers* podem ser ativados de dentro de outros *triggers*, sua definição em cadeia (e talvez recursiva) pode fugir ao controle de seus programadores e gerar inconsistências na base de dados;
- O usuário não tem como definir uma ordem de execução de *triggers* (SQL). Caso uma operação do usuário dispare mais de um *trigger*, o resultado dos primeiros a serem executados pode interferir no processamento dos próximos;
- O sistema não testa a semântica de um *trigger*;
- Posso declarar *triggers* conflitantes entre si (Um *trigger* pode fazer alguma coisa e outro desfazer).

Técnicas Para a Implementação de Controle Automático de RI's no SGBD

Existem duas alternativas:

1. As rotinas de controle de restrições de integridade constituem parte do código do SBD.

⇨ O controle é, portanto interpretativo e acontece em tempo de execução das operações sobre o BD. A cada operação, as RI's a ela associadas são testadas e, em caso de transgressão, um código de erro é transmitido ao programa do usuário.

2. O programa de aplicação é pré-compilado por um módulo especial que investiga cada operação que será executada sobre o BD.

⇨ Para cada uma destas operações, suas RI's associadas são indentificadas e o pré-compilador/processador, ou expande o programa com o código necessário ao teste das RI's (ex.: query-modification), ou conecta o programa (logicamente) a rotinas pré-compiladas que realizam estes testes (ex.: triggers).

Vantagens do Esquema Pré-Compilativo

1. A identificação das RI's que devem ser garantidas para cada operação a ser executada pelo programa é feita uma só vez.

→ Compare com o esquema interpretativo no seguinte caso:

```
WHILE Recebe_Tupla_Do_Usuário   /* via arquivo */
DO Insere_Tupla_Em_Relação_EMP
```

2. O pré-processador pode varrer todo o programa de aplicação, identificando grupos de comandos associados às mesmas RI's.

→ Sendo assim os respectivos testes podem ser inseridos após todos os comandos do grupo correspondente. Por exemplo:

```
Atualiza_Salário_Emp (e-no = 'E-1')
Remove_Tupla_Emp (e-no = 'E-2')
Atualiza_Tupla_Seção (s-totsal)
```

→ Testa_Ris:

s-totsal = SELECT SUM (***e-sal***) FROM EMP WHERE ***e-sec*** = ***s-no***
s-totsal ≤ s-orçam

3. No esquema interpretativo, o sistema vai analisando as operações do programa de aplicação conforme estas vão sendo executadas.

→ Sendo assim, possíveis transgressões que poderiam ser detectadas em tempo de compilação do programa só serão identificadas após a execução de várias operações.

Exemplo: imagine que a terceira operação do exemplo acima não conste do programa de aplicação.

→ As observações feitas, acima, só são válidas para interfaces de BD's embutidos em linguagens de programação de uso geral (ex.: Pascal, COBOL, C). Em interfaces para consultas "ad-hoc", as premissas podem ser outras.

→ É custoso para o SGBD identificar, automaticamente, o conjunto de operações que se relacionam com o mesmo conjunto de RI's.

→ Para o SGBD seria custoso verificar se existem RI's conflitantes. Para a aplicação seria difícil verificar se todas as RI's foram descritas.

→ As restrições de integridade mais complexas (envolvendo mais de uma relação e funções de agregação são mantidas pelos programas de aplicação ou através de triggers e "stored procedures". Os programadores de aplicação podem construir programas mais eficientes para situações específicas.

→ Mesmo que fosse possível implementar todo o controle de RI's no SGBD, o tempo de execução dos testes seria proibitivo para aplicações maiores (mais complexas).

Visões

Relações, no modelo relacional, são as tabelas físicas no banco de dados. As visões são relações virtuais derivadas das relações do banco de dados, ou seja, tabelas virtuais derivadas das tabelas físicas do banco de dados.

Uma visão é criada com o intuito de melhorar a segurança de acesso ao banco de dados, e gerar relações que melhor se adequam às necessidades de uma aplicação, para um determinado usuário (ou grupo de usuários).

Uma visão pode ser encarada como uma janela para um conjunto de dados mantidos em um BD. Considerando um BD relacional, por exemplo, uma visão seria uma relação virtual derivada a partir de dados de uma ou mais relações do esquema. Esta derivação é transparente para a aplicação que a manipula, ou seja, para a aplicação é como se os dados da visão fossem tabelas mantidas fisicamente no BD.

Visões são dinâmicas por definição, ou seja, sempre refletem o estado atual dos dados das relações das quais derivam. Caso novos dados sejam inseridos, ou alguns deles modificados diretamente nas tabelas, os mesmos serão naturalmente "vistos" por futuras consultas solicitadas às visões que têm acesso a estes dados.

Por exemplo:

a) Um funcionário do hospital não deve ter acesso a todos os dados pessoais de um paciente, somente ao seu código e nome;
b) Pode ser interessante vincular os dados de um médico aos dados de suas consultas

O principal cuidado quando se define uma visão é considerá-la passível de atualização ou não. Por *default*, toda visão é passível de consulta. Por outro lado, nem toda visão é atualizável. Se nenhum tipo de controle for feito explicitamente pelo projetista do BD ou implicitamente pelo SGBD, dados inconsistentes podem surgir no BD, em decorrência de uma atualização proveniente de uma visão. É necessário que toda visão atualizável atenda as seguintes premissas (supondo um BD relacional):

1) "Toda visão atualizável deve preservar a chave primária (CP) da relação da qual deriva."

2) "Toda visão atualizável deve conter atributos cujos valores tenham uma correspondência direta com valores de atributos presentes na relação da qual deriva."

3) "Toda visão atualizável deve ser derivada, preferencialmente, de apenas uma relação."

A premissa 1 garante que toda tupla modificada por uma visão tenha condições de ser identificada e modificada na relação física, sem problema de ambiguidade, ou seja, deve existir um mapeamento 1 para 1 de uma tupla da visão para uma tupla da relação. Quando isto não é garantido, tem-se vários problemas, como por exemplo, inclusões de tuplas com CP total ou parcialmente nula nas relações ou remoções de tuplas na visão sem saber exatamente quantas tuplas serão efetivamente removidas na relação.

A premissa 2 considera visões estatísticas, ou seja, visões que retornam certos cálculos associados a alguns atributos de uma relação. São visões tipicamente para consulta a informações derivadas do BD, não havendo sentido a realização de atualizações a partir delas. Isto porquê, ocorrendo atributos da visão que são cálculos a partir de atributos da relação, não há uma correspondência direta entre colunas da visão e da relação. Isto pode dificultar e até mesmo inviabilizar a atualização física do BD, ainda mais quando não se consegue determinar precisamente esta correspondência (por exemplo, um cálculo com várias variáveis, cada uma sendo um atributo de uma relação).

A premissa 3 considera problemas de efeitos indesejáveis de atualizações feitas a partir de visões, pelo fato da mesma agregar atributos de diversas relações. Por exemplo, a inclusão de uma nova tupla na visão se reflete na inclusão de tuplas em todas as relações envolvidas, podendo gerar violação de CP, se nem todos os atributos que compõem as chaves das relações envolvidas estiverem presentes. A atualização de uma tupla da visão acarreta atualizações em atributos de relações que podem violar restrições de CP e chave estrangeira (CE). Da mesma forma, por não haver uma correspondência 1 para 1 entre tuplas da visão e da relação, uma exclusão feita na visão pode ter o efeito indesejável de excluir mais dados do que o realmente esperado.

Ao nível de controle de atualização a partir de visões, a linguagem SQL permite a inclusão da cláusula *With Check Option* ao comando *Create View*, que impede a inclusão ou atualização de valores de atributos da visão que possam violar o seu predicado de definição. Por exemplo, se a visão enxerga apenas dados de pessoas maiores de idade, não é permitido cadastrar, a partir dela, uma pessoa com menos de 18 anos. Porém, este controle é insuficiente, considerando que, em algumas visões, os atributos que fazem parte do seu predicado de definição não são manipulados por ela. Assim, uma nova tupla inserida a partir dela gera *null* nestes atributos e, se nenhum controle explícito for implementado, esta nova tupla não será mais enxergada.

Vantagens

- Elas fazem com que o mesmo dado seja visto por diferentes usuários de diferentes formas (ao mesmo tempo);
- A percepção do usuário é simplificada;
- É óbvio que o mecanismo da visão possibilita aos usuários centrarem-se unicamente nos dados que lhes são importantes e ignorarem o resto. O que talvez não seja tão óbvio é que, pelo menos na recuperação (consulta), tal mecanismo também possa simplificar consideravelmente as operações de manipulação de dados feitas pelo usuário;
- Segurança automática para os dados ocultos → Dados ocultos são aqueles não visíveis através de determinada visão. Ficam claramente protegidos por meio desta visão específica. Assim, obrigar os usuários a acessar o banco de dados através de visões é um mecanismo simples, porém eficaz de controle de autorização.

Algumas Sugestões Importantes

- Impedir operações de atualização sobre visões;
- As tuplas de uma visão devem corresponder a tuplas que tenham condições de serem identificadas nas tabelas físicas das quais ela deriva;
- Cada atributo de uma visão deve corresponder a um atributo distinto e identificável de alguma das tabelas físicas das quais ela deriva.

Concluindo, visões são um mecanismo importante em BD's, uma vez que garantem independência lógica do esquema do BD, ou seja, permite que cada aplicação ou usuário manipule apenas os dados que lhe interessam. A visão também esconde (abstrai) o processo de busca destes dados, que pode ser complexo caso várias relações tenham que ser consultadas e combinadas para se obter a estrutura pretendida pela aplicação ou usuário. Outra importante vantagem de uma visão é ser um mecanismo de autorização de acesso, ou seja, disponibiliza para uma aplicação ou usuário apenas dados que o(a) mesmo(a) tem permissão.

Triggers

Um *trigger* foi definido inicialmente como "um procedimento pré-definido de um banco de dados, condicional ou incondicionalmente sucedido ou precedido de outras operações do banco de dados automaticamente" (K.P *Eswaran, "Specifications, Implementations and Interactions of a Trigger Subsystem in an Integrated Database System*," IBM *Research Report*, RJ1820, 1976). Isto significa, códigos procedurais ou uma seqüência de operações codificadas em uma mistura de SQL e comandos de programação. Um *trigger* é, então, uma lógica de processamento procedural, armazenada nos SGBD's e executada automaticamente pelos servidores de SGBD sob condições específicas. A palavra "automática" é muito importante - aplicações ou usuários não disparam *triggers*, eles são executados automaticamente quando as aplicações ou usuários realizam operações específicas sobre a base de dados. Um trigger tem os seguintes componentes:

- **Coação (situação):** a situação de integridade ou regra de negócio forçada pelo *trigger*, em outras palavras, é a proposta desta ferramenta. Na prática, aparece no cabeçalho do *trigger* e deve refletir-se no seu nome. Por exemplo, "Abertura de Balanço Positivo" requer que todas novas contas devem ter saldos positivos.
- **Evento:** uma situação específica ocorrendo na base de dados, que indica quando a situação deve ser forçado. O evento é dito para ser "realizado" quando a situação ocorre. O evento é especificado de duas maneiras:

    - Estado de alteração da base de dados e
    - Condição de atributo opcional usado para extrair algumas das situações alteradas.

Exemplo: somente inclui-se registro na tabela PLANO com valores positivos no atributo "saldo".

- **Ação:** uma *procedure* ou seqüência de operações procedurais que implementam uma lógica de processamento requerida obrigadas pela situação. Por exemplo, a ação deve forçar a regra de negócio que contas contábeis não podem ter saldo negativo na abertura de balanço. Isto pode ser feito pela rejeição da operação de inserção se o balanço de abertura é negativo, ou pela substituição do valor negativo por zero e sua inserção numa tabela de acontecimentos. A condição "*if*" implicada realça um outro ponto : operações de manipulação convencional em SGBD também são limitadas para implementar as ações requeridas. Eles devem ser estendidos com construções procedurais como repetições

(*while*, *repeat* e *for*) e comandos de condição (*if* e *case*). Um "*trigger*" é, portanto, um evento que dispara uma ação obrigado por uma situação.

Vantagens

A principal características é que eles são armazenados e executados no banco de dados. Segue algumas outras vantagens :

- **O *trigger* é sempre disparado quando o evento associado ocorre**

Desenvolvedores de aplicativos não tem a lembrança para incluir a funcionalidade em todas a aplicações e os usuários não podem desviar dos *triggers* através de ferramentas interativas. A maioria dos SGBD's tem alguns mecanismos para desviar dos *triggers*, ou por desativação temporária ou usando um "*trace point*". Este artifício somente pode ser usado por pessoas altamente responsáveis pelos seus atos.

- ***Triggers* são administrados cetralizadamente**

Eles são codificados, testados e, então, forçados para todas aplicações de acesso ao Banco de Dados. Os *triggers* são usualmente controlados e até controlados, por um DBA experiente. O resultado é que os "*triggers*" são implementados eficientemente.

- **A ativação central e o processamento de *triggers* adapta perfeitamente a arquitetura cliente-servidor**

Uma simples requisição de um cliente pode resultar em uma seqüência completa de verificações, e a execução de operações subsequentes sobre o banco de dados. Os dados e as operações não são trafegadas pela rede entre cliente e servidor.

Pelo motivo de que os "*triggers*" são tão poderosos, eles devem ser gerenciados muito bem e usados corretamente. *Triggers* ineficientes podem corromper a integridade dos dados.

Utilização de Triggers

Triggers são extremamente poderosos e podem ser usados para vários propósitos:

CONTROLE DE INTEGRIDADE → pode-se usar *triggers* para implementar integridade de domínio, integridade de atributos, integridade referencial e situações de integridade não-referencial.

REGRAS DE NEGÓCIO → utiliza-se *triggers* para centralizar obrigações de regras de negócios. Regras de negócios são situações invocadas sobre relacionamentos entre tabelas ou entre registros diferentes na mesma tabela.

Por exemplo, a soma dos totais de ITEM_PEDIDO devem adicionar ao total do registro de PEDIDO do correspondente pedido.

APLICAÇÃO LÓGICA → pode-se usar *triggers* para forçar lógicas de negócio de modo centralizado. Por exemplo, inserir automaticamente registros em ORDEM e ITEM_ORDEM quando o valor de QTDE_DISPONIVEL estiver abaixo, pedindo uma aquisição. Regras de negócios podem ser formalizadas e, atualmente, serem definidas de modo declarativo. Isto tudo se a sintaxe declarativa permitir, mas aplicações lógicas mais complexas em funcionalidade podem ser especificadas.

SEGURANÇA → utiliza-se *triggers* para verificar situações importantes de segurança sobre a base de dados. Quando uma operação é realizada sobre uma entidade "sensível", o *trigger* é disparado para verificar se esta operação é permitida para o usuário.

Por exemplo, pode-se somente inserir um registro na tabela de FUNCIONARIO se a coluna DEPARTAMENTO conter o valor deste próprio departamento. Em muitos sistemas, entretanto, não pode-se usar *triggers* para restringir a visualização dos dados pelos usuários. Geralmente, estes sistemas possuem ferramentas específicas otimizadas para atender estes casos de permissão de acesso às tabelas.

AUDITORIA → *triggers* podem incluir registros em tabelas de auditoria com a finalidade de registrar todas operações sobre tabelas importantes ao sistema. O problema desta abordagem é que muitas ações de *triggers* são feitas sob controles transacionais. Quando uma operação é retornada (*rollback*), todas as operações relacionadas em *triggers*, também, são recuperadas. Os *triggers* somente gravam os efeitos das operações concluídas com sucesso. Quando uma operação malsucedida é recuperada posteriormente, a entrada de auditoria desta operação também será retornada. A auditoria não contém a ameaça de perigo à violação da integridade dos dados ou à restrição de segurança.

REPLICAÇÃO DE DADOS → muitos representantes comerciais de SGBD's e consultores tem implementado replicadores usando *triggers* como mecanismo de gravação. Na prática, quando as tabelas replicadas são alteradas, os *triggers* disparam um registro de movimentação em tabelas de armazenamento (*buffer table*). Um servidor de replicação então propaga as operações efetuadas a partir das *buffer tables* para os vários Bancos de Dados de destino. Nesta situação, o controle transacional sobre os *triggers* é

extremamente utilizado, para que somente seja replicado transações completadas com sucesso.

O uso de *triggers* é limitado somente à funcionalidade provida pelo SGBD especificado e, é lógico, a sua imaginação e ao seu espírito inovador.

Atualmente, os *triggers* estão próximo ao servidor de Banco de Dados. Em muitos casos, tornando o serviço de banco de dados bastante "pesado". Para descarregar este serviço de *triggers* dos servidores, as novas versões de SGBD's estão isolando os *triggers* em um servidor específico, para receber tratamento específico.

Exemplo de *trigger* para obrigar a abertura de balanço na tabela PLANO. Implementado em CA-OpenIngres/Desktop 1.1

```
create table PLANO_CONTAS
 (num_conta   char (10) not null,
  desc_conta   char (10) not null,
  tipo_conta      char (1) not null,
  saldo_conta     decimal (10,2) not null,
  primary key (num_conta));

/* procedure stored FORCA_BALANCO */
/* situacao: obriga a entrada de valores positivos */


store FORCA_BALANCO
procedure FORCA_BALANCO static
parameters
 string       num_conta
 number       saldo_conta
local variables
 sql handle AcctHandle
actions
 on procedure startup
       begin
             call sqlconnect (AcctHandle)
             call    sqlprepare    (AcctHandle,    'update
PLANO_CONTAS    \
             set   saldo_conta   =   0   where   num_conta   =
:num_conta')
       end
 on procedure execute
       begin
             if (saldo_conta < 0)
                call sqlexecute (AcctHandle)
                call                      sqlgetmodified_rows
(AcctHandle,RowCount)
                if RowCount <= 0
                    return 20001
       end
 on procedure close
       begin
             call sqldisconnect (AcctHandle)
       end
```

```
create trigger FORCA_BALANCO_TRIGGER
 after insert on PLANO_CONTAS
 (execute      FORCA_BALANCO      (PLANO_CONTAS.num_conta,
PLANO_CONTAS.saldo_conta))
 for each row
```