**A Arquitetura de Três-Esquemas (Ansi Sparc)**

O objetivo da arquitetura de três-esquemas é separar o usuário da aplicação do banco de dados físico. Nessa arquitetura, os esquemas podem ser definidos por três níveis:

* **O nível interno** tem um esquema interno, que descreve a estrutura de armazenamento físico do banco de dados. Esse esquema utiliza um modelo de dado físico e descreve os detalhes complexos do armazenamento de dados e caminhos de acesso ao banco de dados.
* **O nível conceitual**possui um esquema conceitual, que descreve a estrutura de todo o banco de dados para a comunidade de usuários. O **[esquema conceitual]** oculta os detalhes das estruturas de armazenamento físico e se concentra na descrição de entidades, tipos de dados, conexões, operações de usuários e restrições. Geralmente, um modelo de dados representacional é usado para descrever o esquema conceitual quando o sistema de banco de dados for implementado. Esse esquema de implementação conceitual é normalmente baseado em um projeto de esquema conceitual em um modelo de dados de alto nível.
* **O nível externo** ou visão (view) abrange os esquemas externos ou visões de usuários. Cada esquema externo descreve a parte do banco de dados que um dado grupo de usuários tem interesse e oculta o restante do banco de dados desse grupo. Como no item anterior, cada esquema externo é tipicamente implementado usando-se um modelo de dados representacional, possivelmente baseado em um projeto de esquema externo em um modelo de dados de alto nível.

A arquitetura de três-esquemas é uma ferramenta conveniente com a qual o usuário pode visualizar os níveis do esque­ma em um sistema de banco de dados. A maioria dos SGBDs NÃO separa os três níveis completamente, mas suporta a arquitetu­ra de três-esquemas de alguma forma. Alguns SGBDs incluem detalhes do nível físico no esquema conceituai. Na maioria dos SGBDs que oferece suporte a diferentes visões de usuários, os esquemas externos são especificados no mesmo modelo de dados que descreve a informação no nível conceituai. Alguns SGBDs permitem que diferentes modelos de dados sejam usados nos níveis conceituai e externo. A **independência de dados física** por si não implica em atendimento à **independência de dados lógica**. Determinado SGBD pode aplicar cada um dos conceitos de forma individual e, inclusive, parcialmente. Os mapeamentos **conceitual/interno** e **externo/conceitual** são a chave para a independência de dados física e lógica, respectivamente. Um sistema fornece independência de dados física se os usuários e programas de usuários forem imunes a mudanãs na estrutura física do banco de dados armazenado. De modo semelhante, um sistema proporciona independência de dados lógica se os usuários e os programas de usuários também forem imunes a mudanças na estrutura lógica do banco de dados. [Date, 2003].

**Tipos de banco de dados**

* **Hierárquico**: Os dados estão estruturados em hierarquias ou árvores as árvores são percorridas desde a raiz até ao nó que contém o registro pretendido, para efetuado o acesso ao dado. Adequado para acesso sequencial a dados, principalmente em casos em que, pela sua natureza, os dados estão organizados de forma sequencial. **Cada filho só pode ter um pai.** O relacionamento entre um registro pai e registros filhos é de 1:N no máximo. Problemas: não é possível consultar um produto enquanto ele não estiver associado a uma encomenda. Para consulta um produto, é obrigatório consultar antes o cliente e a encomenda. A estrutura de árvore é difícil de manter quando o volume de dados aumenta.
* **Rede**: É uma extensão do modelo hierárquico. Elimina o conceito de hierarquia. Cada registro filho pode ser ligado a mais de um registro pai, criando conexões bastante complexas. No modelo em rede os registros estão organizados em grafos. É possível representar relacionamentos: 1:1, 1:n e n:n. Torna a pesquisa mais rápida e mais flexível, pois não depende de um único nó raiz como vetor de inicialização de pesquisa. Ao contrário do Modelo Hierárquico, em que qualquer acesso aos dados passa pela raiz, no modelo em rede é possível acessar qualquer nó da rede. Problemas: Ainda apresenta os mesmos problemas com relação ao projeto de estrutura do modelo hierárquico. Qualquer alteração feita em uma classe de dados implica na criação de uma nova estrutura para suportar aquela alteração. Independência dos dados fica prejudicada
* **Relacional:** Aumentar a independência de dados nos sistemas gerenciadores de banco de dados. Prover um conjunto de funções apoiadas em álgebra relacional para armazenamento e recuperação de dados. Permitir processamento dedicado exclusivo. Representa os dados num BD como uma coleção de tabelas (relações). A tabela é composta por linhas (**tuplas**) e colunas (**atributos**).

O **modelo hierárquico** -> os dados são estruturados em hierarquias ou árvores. Os nós das hierarquias contêm ocorrências de registros, onde cada registro é uma coleção de campos (atributos), cada um contendo apenas uma informação. O registro da hierarquia que precede a outros é o **registro-pai**, os outros são chamados de **registros-filhos**. Uma **ligação** é uma **associação** entre dois registros. O relacionamento entre um registro-pai e vários registros-filhos possui cardinalidade **1:N**.  
  
O **modelo em redes** -> surgiu como uma extensão ao modelo hierárquico, eliminando o conceito de hierarquia e permitindo que um mesmo registro estivesse envolvido em várias associações. No **modelo em rede**, os registros são organizados em **grafos** onde aparece um único tipo de associação (set) que define uma relação **1:N** entre 2 tipos de registros: **proprietário** e **membro**.

A integridade de uma transação depende de 4 propriedades, conhecidas como ACID.

**Atomicidade -** Todas as ações que compõem a unidade de trabalho da transação devem ser concluídas com sucesso, para que seja efetivada. Se durante a transação qualquer ação que constitui unidade de trabalho falhar, a transação inteira deve ser desfeita (rollback). Quando todas as ações são efetuadas com sucesso, a transação pode ser efetivada e persistida em banco (commit).

**Consistência -** Todas as regras e restrições definidas no banco de dados devem ser obedecidas. Relacionamentos por chaves estrangeiras, checagem de valores para campos restritos ou únicos devem ser obedecidos para que uma transação possa ser completada com sucesso.  
  
**Isolamento -** Cada transação funciona completamente à parte de outras estações. Todas as operações são parte de uma transação única. O principio é que nenhuma outra transação, operando no mesmo sistema, possa interferir no funcionamento da transação corrente (é um mecanismo de controle). Outras transações não podem visualizar os resultados parciais das operações de uma transação em andamento (ainda em respeito à propriedade da atomicidade).  
  
**Durabilidade -** Significa que os resultados de uma transação são permanentes e podem ser desfeitos somente por uma transação subsequente. Por exemplo: todos os dados e status relativos a uma transação devem ser armazenados num repositório permanente, não sendo passíveis de falha por uma falha de hardware.

**Objeto persistente** é aquele que é armazenado no banco de dados. (persistir é salvar)

Há dois mecanismos para deixar um objeto persistente( pra deixar o objeto salvo):

1. Nomeação( naming) : consiste em dar ao objeto um nome persistente único pelo qual ele poderá ser recuperado futuramente. (dá ao objeto um nome que já foi salvo e que não é repetido).

2. Alcançabilidade( reachability): consiste em tornar persistente o objeto ao qual se possa alcançar a partir de um objeto persistente. ( pega um objeto salvo e alcança/associa( "junta") a um objeto não salvo...pronto o não salvo passará a ficar salvo).

**Banco de Dados Distribuídos (N camadas)**: Nesta arquitetura, a informação está distribuída em diversos servidores. Como exemplo, observe a abaix**o. Cada servidor atua como no sistema cliente-servidor, porém as consultas oriundas dos aplicativos são feitas para qualquer servidor indistintamente**. Caso a informação solicitada seja mantida por outro servidor ou servidores, o sistema encarrega-se de obter a informação necessária, de maneira transparente para o aplicativo, que passa a atuar consultando a rede, independente de conhecer seus servidores. Exemplos típicos são as bases de dados corporativas, em que o volume de informação é muito grande e, por isso, deve ser distribuído em diversos servidores. Porém, não é dependente de aspectos lógicos de carga de acesso aos dados, ou base de dados fracamente acopladas, em que uma informação solicitada vai sendo coletada numa propagação da consulta numa cadeia de servidores. A característica básica é a existência de diversos programas aplicativos consultando a rede para acessar os dados necessários, porém, sem o conhecimento explícito de quais servidores dispõem desses dados.

**Plataformas centralizadas**: Na arquitetura centralizada, existe um computador com grande capacidade de processamento, o qual é o hospedeiro do SGBD e emuladores para os vários aplicativos. Esta arquitetura tem como principal vantagem a de permitir que muitos usuários manipulem grande volume de dados. **Sua principal desvantagem está no seu alto custo**, pois exige ambiente especial para mainframes e soluções centralizadas.

**Sistemas de Computador Pessoal - PC**: Os computadores pessoais trabalham em sistema stand-alone, ou seja, fazem seus processamentos sozinhos. No começo esse processamento era bastante limitado, porém, com a evolução do hardware, tem-se hoje PCs com grande capacidade de processamento. Eles utilizam o padrão Xbase e quando se trata de SGBDs, funcionam como hospedeiros e terminais. Desta maneira, possuem um único aplicativo a ser executado na máquina. A principal vantagem desta arquitetura é a simplicidade.

**Banco de Dados Cliente-Servidor**: Na arquitetura Cliente-Servidor, o cliente (front\_end) executa as tarefas do aplicativo, ou seja, fornece a interface do usuário (tela, e processamento de entrada e saída). O servidor (back\_end) executa as consultas no DBMS e retorna os resultados ao cliente. Apesar de ser uma arquitetura bastante popular, são necessárias soluções sofisticadas de software que possibilitem: o tratamento de transações, as confirmações de transações (commits), desfazer transações (rollbacks), linguagens de consultas (stored procedures) e gatilhos (triggers). **A principal vantagem desta arquitetura é a divisão do processamento entre dois sistemas, o que reduz o tráfego de dados na rede.**

Há dois tipos de redundância de dados: a redundância controlada e a redundância não controlada.

A **redundância controlada** acontece quando o software tem conhecimento da múltipla representação da informação e garante a sincronia entre as diversas representações. Para o usuário, tudo funciona como se o sistema utilizasse apenas uma base de dados. Esse tipo de redundância é geralmente utilizado para possibilitar uma maior performance ao sistema ou aumentar sua confiabilidade.  
A **redundância não controlada** acontece quando a responsabilidade pela manutenção e sincronia das diversas representações de uma mesma informação fica a cargo do usuário e não do software.

**Independência de dados**

Vimos três níveis de abstração pelos quais o banco de dados pode ser visto. 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 dos dados:

* **Independência física de dados**: é a habilidade de modificar o esquema físico sem a necessidade de reescrever os programas aplicativos. As modificações no nível físico são ocasionalmente necessárias para melhorar o desempenho;
* **Independência lógica de dados:** é a habilidade de modificar o esquema conceitual sem a necessidade de reescrever os programas aplicativos. As modificações no nível conceitual são necessárias quando a estrutura lógica do banco de dados é alterada (por exemplo, a adição de contas de bolsas de mercado num sistema bancário).

A independência lógica dos dados é mais difícil de ser alcançada do que a independência física, porém os programas são bastante dependentes da estrutura lógica dos dados que eles acessam.

O conceito de independência dos dados é similar em muitos aspectos ao conceito de tipos abstratos de dados em modernas linguagens de programação. Ambos escondem detalhes de implementação do usuário. Isto permite ao usuário concentrar-se na estrutura geral em vez de detalhes de baixo nível de implementação. A independência física de dados não garante que também temos a independência lógica de dados.

**Redundância de dados**

**Normalização**

* reduz redundância e as chances de inconsistência
* MAU desempenho para consultas

**Desnormalização**

* melhorar desempenho das consultas
* foco na entrega da informação
* perde-se a garantia de consistência dos dados, posto que haverá um banco bem maior
* **BOM desempenho para consultas**

***"Segundo o Navathe para consultas é bom ter uma redundância controlada o que melhora o desempenho.*** A segunda diretriz é consistente, de certo modo, com as demais declarações da primeira diretriz. Podemos ver também a necessidade de uma abordagem mais formal para avaliar se um modelo cumpre essas diretrizes.***É importante observar que essas diretrizes, às vezes, precisam ser violadas de modo a melhorar o desempenho de certas consultas."***

A **redundância controlada** acontece quando o software tem conhecimento da múltipla representação da informação e garante a sincronia entre as diversas representações. Para o usuário, tudo funciona como se o sistema utilizasse apenas uma base de dados. Esse tipo de redundância é geralmente utilizado para possibilitar uma maior performance ao sistema ou aumentar sua confiabilidade.  
A **redundância não controlada** acontece quando a responsabilidade pela manutenção e sincronia das diversas representações de uma mesma informação fica a cargo do usuário e não do software.

**As regras de Codd**

* **Regra Zero:** Todas as regras baseiam-se na noção de que para que um Banco de Dados seja considerado Relacional, ele deve utilizar os recursos relacionais exclusivamente para seu gerenciamento.
* **Regra 1: Informação** – Todas as informações de um BDR devem ser representadas logicamente como valores de coluna em linhas dentro das tabelas.
* **Regra 2: Garantia de Acesso** – Deve-se garantir que todos os valores de uma tabela possam ser acessados por meio de uma combinação de nome de tabela, valor de chave primária e nome de coluna.
* **Regra 3: Tratamento Sistemático de Nulos** – Os nulos devem ser representados e tratados de modo sistemático, independente do tipo de dados.
* **Regra 4: Catálogo On-Line Dinâmico com Base no Modelo Relacional** – Os metadados devem ser armazenados e gerenciados como dados comuns, ou seja, em tabelas no interior do BD. Esses dados devem estar disponíveis aos usuários autorizados, utilizando a linguagem relacional padrão do BD.
* **Regra 5: Sublinguagem Ampla de Dados** – O BDR pode suportar várias linguagens. No entanto deve suportar uma linguagem declarativa bem definida com suporte para definição de dados, definição de visualização, manipulação de dados (interativa ou por programa), restrições de integridade, autorização e gerenciamento de transações (iniciar, comprometer e desfazer).
* **Regra 6: Atualização de Visualização** – Qualquer visualização que teoricamente possa ser atualizada deve ser por meio do sistema.
* **Regra 7: Inserção, atualização e exclusão de alto nível** – O BD deve dar suporte à configuração do nível de inserções, atualizações e exclusões. Ou seja, a capacidade de manipular um conjunto de dados através de um comando, deve-se estender às operações de Linguagem de Manipulação de Dados (DML) como insert, update e delete.
* **Regra 8: Independência Física de Dados** – Aplicativos e recursos ad hoc não são afetados logicamente quando os métodos de acesso ou as estruturas de armazenamento físico são alterados.
* **Regra 9: Independência Lógica de Dados** – Aplicativos e recursos ad hoc não são afetados logicamente quando de alterações de estruturas de tabela que preservem os valores originais da tabela (alteração da ordem ou inserção de colunas). Alterações nas relações e nas views causam pouco ou nenhum impacto nas aplicações.
* **Regra 10: Independência de Integridade**– Deve ser possível que todas as restrições de integridade relacional sejam definidas na linguagem relacional e armazenadas no catálogo de sistema, não no nível da aplicação. As aplicações não devem ser afetadas quando ocorrer mudanças nas restrições de integridade.
* **Regra 11: Independência de Distribuição** – Os usuários finais e aplicativos não conhecem nem são afetados pela localização dos dados (BD Distribuídos VS. BD Locais).
* **Regra 12: Não transposição das Regras**– Se o sistema dá suporte a acesso de baixo nível aos dados, não deve haver um modo de negligenciar as regras de integridade do BD.