2. OS MODELOS DE DADOS

Este capítulo apresenta os modelos de dados, iniciando pelo modelo Hierárquico chegando até os modelos NoSQL.

Antes dos anos 60, dados eram armazenados e acessados em estruturas denominadas **Arquivos Indexados** VSAM (*Virtual Storage Access Method*).

Na década de 1960, surge o modelo de dados **Hierárquico**, estrutura única até então, onde os dados eram dispostos segundo uma hierarquia. Na mesma década, o comitê CODASYL (*Committee on Data Systems and Languages*), (TAYLOR, W.R., 1978), propõe ajustes no modelo de dados hierárquico, originando o modelo de dados em **Rede** (FRANK, R.L., 1978).

Na década de 1970, surge o modelo de dados **Relacional** cujo primeiro ajuste, denominado MER-X (Modelo Entidade Relacionamento Estendido), ocorreu na década de 1980, (SILBERSCHATZ, A. *et al*., 2006).

Na década de 1980, surge o modelo de dados **Orientado a Objetos** (KIM, W., 1990), inaugurando a família dos modelos de dados **pós-relacionais**.

Surge na década de 1990, o modelo de dados **Objeto-Relacional**, fruto da combinação dos conceitos encontrados no modelo de dados Relacional e no modelo de dados Orientado a Objetos. Na sequência, ainda na mesma década, 1990, modelos de dados em sua maioria, relacionais, armazenavam dados dispostos em arquivos do tipo XML (*Extensiveble Markup Language*), e por conta das características estruturais desses arquivos, passaram a ser denominados modelos de dados Semi-Estruturados (HEUSER, C.A., 2014).

Na década seguinte, 2000, também na condição de pós-relacionais, surgem os modelos de dados Não Estruturados, rotulados pela sigla **NoSQL** (*Not Only SQL*), e com eles, novos conceitos associados a consistência, disponibilidade e principalmente, escalabilidade de dados.

2.1. ARQUIVOS INDEXADOS

Arquivos indexados, são arquivos que possuem uma entrada auxiliar, denominada índice, que possui um tamanho bem menor que o arquivo de dados, o qual otimiza as buscas aleatórias de registros, com uma determinada chave.

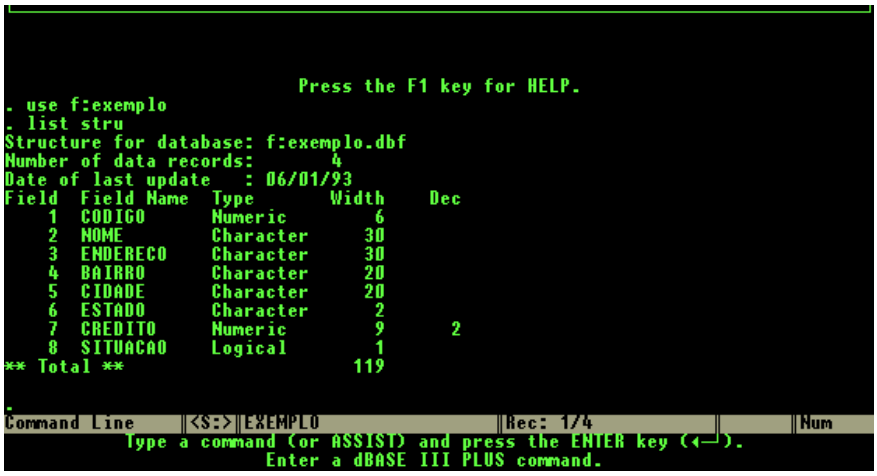
Um índice é um mapeamento de chaves, que é utilizada para fins de otimização na busca, permitindo uma localização mais rápida de um determinado registro em um arquivo, e eles também são armazenados em arquivos em disco.

Os índices primários seguem a mesma ordem do arquivo de dados, podendo ser baseados ou não na chave primária. Esse tipo de índice, podem apontar para blocos de registros, índices esparsos.

Esses tipos de índices tem um bom desempenho, pois ele tem menos registros do que o arquivo de dados, resultando em uma quantidade menor de bytes transferidos do disco rígido, implicando em menos movimentações do cabeçote de leitura e escrita, facilitando a busca.

Índices secundários não seguem a mesma ordem dos arquivos de dados, tendo as chaves de busca diferentes da chave primária. A chave pode ser qualquer composição de campos do arquivo de dados, como:

- Um campo;
- Uma parte dele;
- Uma combinação de campos;
- O resultado de um processamento com um ou mais campos.



Uso de índices em Arquivos DBASE (DBF) com Linguagem Clipper 5.1

```
-- Essa ação é feita somente no primeiro uso do arquivo ou quando é preciso uma reordenação

USE EXEMPLO
CREATE INDEX CODIGO TO INDCODIGO.NTX
CREATE INDEX NOME TO INDNOME.NTX

-- Abre o arquivo, seta o índice e faz busca para recuperar um registro apagado logicamente

USE EXEMPLO SET ON INDEX 1
FIND “0001”
IF DELETED( )
```

```
    RECALL( ) “REGISTRO RECUPERADO”
ENDIF

USE EXEMPLO SET ON INDEX 2
SET SOFTSEEK ON
SEEK “ALEX”
IF DELETED( )
    RECALL( ) “REGISTRO RECUPERADO”
ENDIF

-- Abre o arquivo, seta o índice e faz busca para recuperar um registro apagado logicamente

CODVAR := 0
@ 10,20 SAY “DIGITE O CÓDIGO:” GET CODVAR PICTURE “999999”
READ
FIND CODVAR // pesquisa o conteúdo da variável
IF FOUND( )
   DISPLAY COD,NOME,SALARIO
ENDIF
```

Inserção de um Registro em Arquivos DBASE (DBF) com Linguagem Clipper 5.1

```
CODVAR = 0
NOMEVAR = SPACE(35)
DATAVAR = CTOD(“ / / “)
// entrada de dados
@@ 08,10 SAY “CÓDIGO.....:” GET CODVAR PICTURE “999999”
READ
IF CODVAR = 0 // sai
   RETURN // retorna
ENDIF
USE EXEMPLO SET ON INDEX 1
SEEK CODVAR //  ou FIND pesquisa no índice o conteúdo da variável CODVAR
IF EOF( ) // não encontrou ou .NOT. FOUND()
   DBAPPEND( ) // cria um registro em branco e bloqueia
   @@ 10,10 SAY ”NOME.....:” GET NOMEVAR PICT “@!”
   @@ 20,10 SAY “DATA ADMISSÃO...:” GET DATAVAR
   READ
   // grava os dados no registro em branco
   REPLACE NOME WITH NOMEVAR
   REPLACE DTADM WITH DATAVAR
   COMMIT // atualiza fisicamente o registro
   REPLACE COD WITH CODVAR
   UNLOCK // libera o registro criado
ELSE
  @@ 21,20 SAY “** REGISTRO JÁ CADASTRADO **” WAIT “ “ // aguarda qq tecla
ENDIF
```

2.2. O MODELO DE DADOS HIERÁRQUICO

Neste modelo, os dados são organizados de forma descendente, haja vista a representação no formato árvore, e são definidos como uma coleção de registros conectados entre si por meio de ligações.

Cada registro desta coleção contém uma coleção de campos onde cada campo contém um único valor. Uma ligação conecta exclusivamente dois registros, ou seja, o relacionamento entre um registro-pai ou segmento pai, hierarquicamente superior, e um registro-filho ou segmento filho, hierarquicamente inferior, tem cardinalidade **1:N** (*um para muitos*), respectivamente.

A Figura 1 apresenta um exemplo da estrutura do modelo hierárquico de dados. Nesta Figura, observam-se as relações hierárquicas entre os diferentes níveis do modelo.

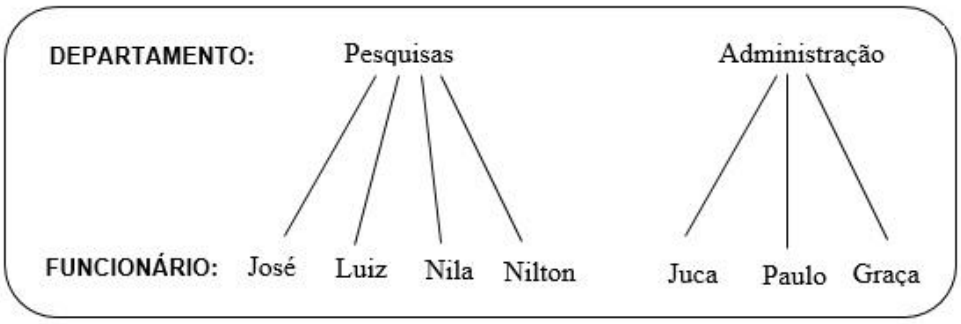


Figura 1. Estrutura do modelo hierárquico de dados, adaptado de ELMASRI (2005)

Porém, com o aumento crescente do volume dos dados e da complexidade dos sistemas, o modelo hierárquico de dados foi se mostrando inadequado, pois manter uma estrutura hierárquica de árvore se tornou tarefa trabalhosa, além da impossibilidade de se representar relacionamentos com cardinalidade **N:N** (*muitos para muitos*).

2.3. O MODELO DE DADOS EM REDES

Esse modelo foi utilizado pela primeira vez em 1964, por Charles Bachman, e possibilita cardinalidade **N:N** entre diferentes hierarquias.

Neste modelo, a pesquisa torna-se mais flexível e o desempenho melhor, quando comparados ao modelo hierárquico, por não depender de um único registro (nó) como inicializador da pesquisa. Porém, o modelo de dados em redes, estruturalmente, apresenta os mesmos problemas do modelo

hierárquico, onde uma alteração efetuada em determinada classe hierárquica de dados, implica na criação de nova estrutura para suportar esta nova classe, além de, da mesma forma que no modelo de dados hierárquico, o acesso aos dados ocorrer através de linguagens denominadas hospedeiras.

A Figura 2 destaca o diferencial do modelo em redes, em relação ao modelo hierárquico.

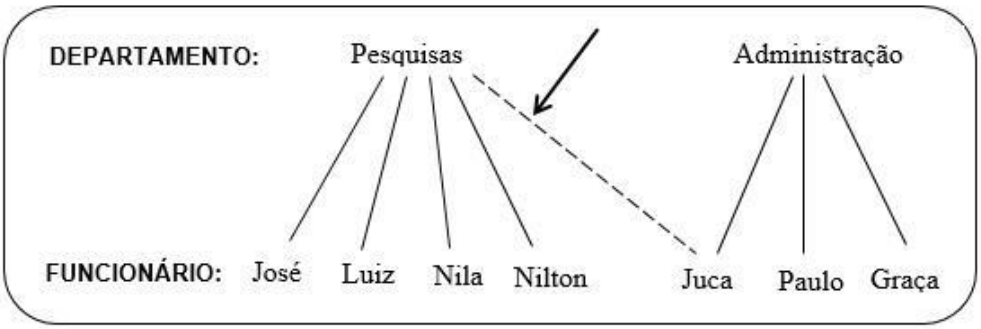


Figura 2. Estrutura do modelo de dados em rede, adaptado de ELMASRI (2005)

2.4. O MODELO DE DADOS RELACIONAL

O modelo relacional, criado na década de 1970, pelo matemático britânico Edgar Frank Codd (CODD, E.F., 1970) e posteriormente aprimorado por Christopher J. Date (DATE, C.J., 2004) foi desenvolvido com a finalidade de suprir deficiências estruturais dos modelos hierárquicos e de redes, principalmente as deficiências associadas a dificuldades de manipulação dos dados e também por inconsistências causadas pela não versatilidade cardinal das associações hierárquicas, ou seja, dependência dos dados.

O modelo relacional, através de regras fundamentadas da teoria matemática de conjuntos com o acréscimo de funções apoiadas na álgebra relacional, permite processamento *ad hoc* (processamento dedicado, exclusivo) quando da captura, processamento e disponibilização da informação (SILBERSCHATZ, A. et al., 2006).

Devido a sua simplicidade, flexibilidade e desempenho satisfatórios, o modelo de dados relacional mostrou-se apropriado principalmente na solução de problemas de cunho comercial. Este modelo de dados representa os dados, como uma coleção de entidades interligadas (tabelas).

As **entidades** (*retângulos*), interligadas por r**elacionamentos** (*losangos*), são compostas por **atributos** (*elipses*) que, necessariamente, contém o mesmo tipo de dado.

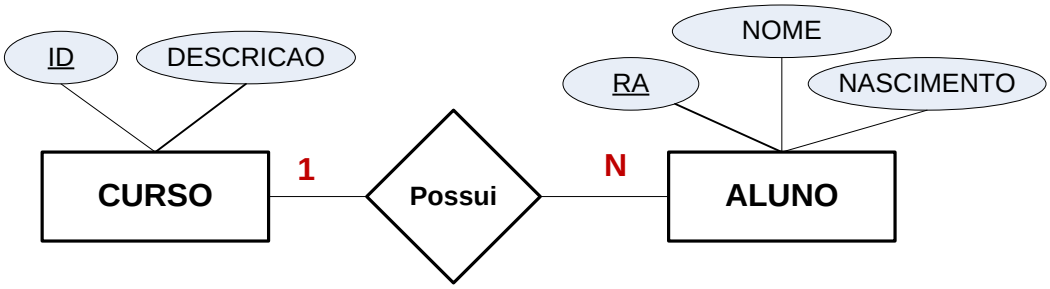
Regras oriundas da coexistência combinada de atributos e relacionamentos, ditas formas normais, garantem a integridade do dado através da eliminação de

redundâncias, evitando tuplas (linhas - conjunto sequencial de colunas) repetidas (SILBERSCHATZ, A. et al., 2012).

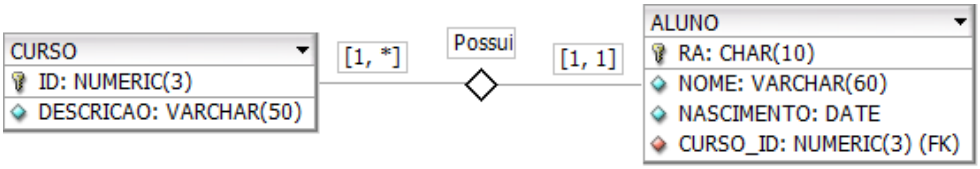
A Figura 3 mostra a estrutura básica do modelo de dados relacional, por meio do diagrama **Entidade-Relacionamento** (E-R).

Figura 3. Estrutura básica do modelo de dados relacional, NETO J.G., (2016).

Exemplo teórico de Diagrama E-R



Exemplo Teórico transformado em Diagrama E-R Prático



2.5. O MODELO DE DADOS ORIENTADO A OBJETOS

Quando criado, acreditava-se que o modelo de dados orientado a objetos, com o passar do tempo representaria grande parcela dos modelos de dados utilizados, por suprir na época, a então principal deficiência do modelo relacional, ou seja, representar de forma única dados de tipos distintos, dados complexos.

Representar um objeto complexo em um modelo relacional significa subdividir o objeto em um grande número de linhas e realizar diversas operações *join* (junção) na hora de reconstruir o objeto. Isto é ruim.

Representar um objeto complexo em um modelo objeto-relacional é difícil. Surge, então, uma nova alternativa para persistir objetos: são os sistemas gerenciadores de bancos de dados orientados a objetos (SGBDOO).

Eles implementam de forma natural os modelos orientados a objetos, isto é, os conceitos de classe, objeto, tipo, identidade, igualdade, encapsulamento, herança, agregação, método e polimorfismo. Além disso, possuem mecanismos especiais para controlar transações entre objetos, técnicas de armazenamento que facilitam a recuperação rápida de objetos complexos (*clustering*) e funções adicionais para coordenar atividades cooperativas, tais como mecanismos para avisar os usuários sobre mudanças de estado nos objetos e para notificar a disponibilidade dos objetos.

O acesso aos objetos é mais versátil quando se utiliza um SGBDOO, já que é possível fazê-lo de forma navegacional ou por meio de linguagens do tipo SQL. Usando SGBDRs, a única maneira de acessar os dados é através de linguagens de consulta (SQL). Porém, existem ferramentas (frameworks) que realizam o Mapeamento Objeto-Relacional (Tabela-Classes) em tempo de execução da aplicação. É o caso do Hibernate, utilizada pela ferramenta Eclipse, que por sua vez utiliza-se da linguagem Java.

O **Hibernate** é um framework para o mapeamento objeto-relacional escrito na linguagem Java, mas também é disponível em **.Net** com o nome **NHibernate**. Este framework facilita o mapeamento dos atributos entre uma base tradicional de dados relacionais e o modelo objeto de uma aplicação, mediante o uso de arquivos (XML) ou anotações Java. Hibernate é um software livre de código aberto distribuído com a licença LGPL (Licença Pública Geral Menor)

A **HQL** (Hibernate Query Language) é um dialeto SQL para o Hibernate. Ela é uma poderosa linguagem de consulta que se parece muito com a SQL, mas a HQL é totalmente orientada a objeto, incluindo os paradigmas de herança, polimorfismo e encapsulamento.

No Hibernate, pode-se escolher tanto usar a SQL quanto a HQL. Escolhendo a HQL, poderá executar os pedidos SQL sobre as classes de persistência do Java ao invés de tabelas no banco de dados.

Hoje, porém, verifica-se que os modelos de dados orientados a objetos são utilizados em aplicações específicas, enquanto que os modelos de dados relacionais continuam a sustentar os negócios tradicionais, onde estruturas de dados baseadas em relações são necessárias e suficientes.

O diagrama de classes da UML (*Unified Modeling Language*) é geralmente utilizado para representar o modelo de dados orientado a objetos.

Análogo ao modelo de dados relacional, representado por entidades e relacionamentos, o modelo orientado a objetos é representado por classes e atributos.

A Figura 4 apresenta a estrutura básica do modelo de dados orientado a objetos, por meio do diagrama de classes.

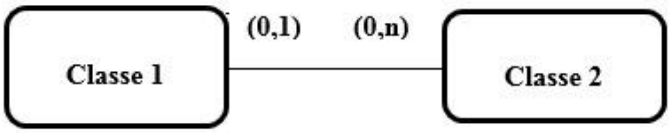


Figura 4. Estrutura básica do modelo de dados orientado a objetos

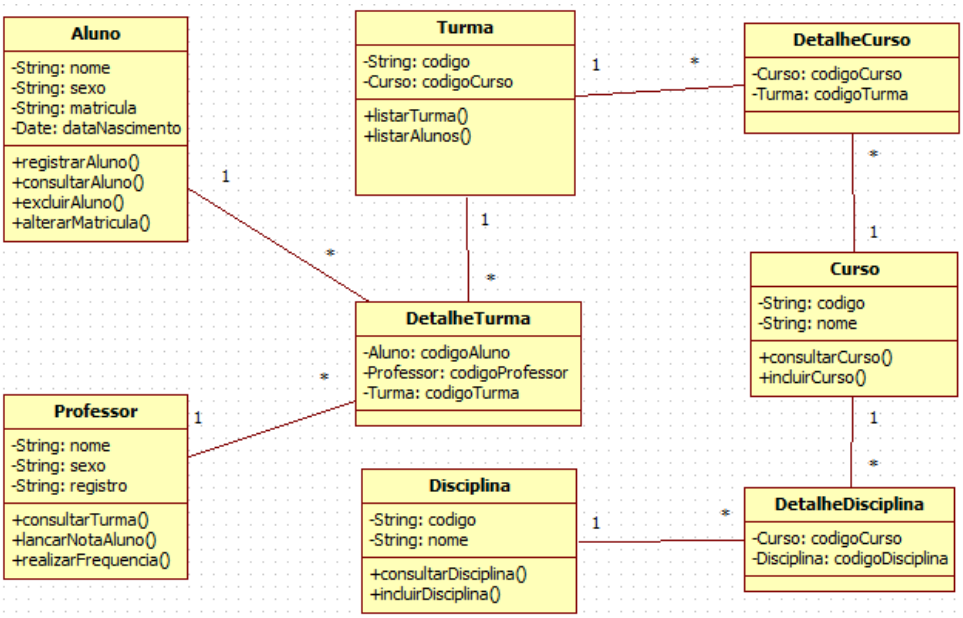
A conjunção dos tipos de dados distintos, segregados e representados por atributos no modelo relacional, tem no modelo orientado a objetos, concepção integrada, diferenciando-se da representação semântica, observada no modelo relacional.

No modelo de dados orientado a objetos, a manipulação ocorre por intermédio dos métodos definidos pelas classes, nas quais os objetos estão.

Hierarquicamente falando, os objetos são organizados em supertipos que por sua vez possuem tipos e subtipos. Assim, o relacionamento entre os objetos é possível desde que haja referências estabelecidas entre eles.

Porém, diferentemente do modelo relacional, fortemente fundamentado em teorias matemáticas consistentes, este modelo carece de fundamentação matemática formal (SILBERSCHATZ, A. et al., 2006).

Exemplo de um Diagrama de Classes



2.6. OS MODELOS DE DADOS NoSQL

Posteriores aos modelos de dados relacional e orientados a objetos, os modelos de dados **NoSQL** podem ser classificados como modelos relacionais fracamente normalizados, denominados MRNN (Modelos Relacionais Não Normalizados ou Não Relacionais) (SETZER, V.W., 2010).

Os modelos de dados NoSQL são divididos em quatro tipos: *Tipo Chave-valor, Tipo coluna, Tipo orientado a documentos e Tipo orientado a grafos*.

2.6.1 Modelo de dados do tipo chave/valor

Considerado o mais escalável dos modelos de dados pós-relacionais, os conceitos deste modelo de dados são intrínsecos aos demais modelos *NoSQL* e suporta maiores volumes de dados quando comparado aos demais modelos, por ser baseado em coleções de chaves únicas, com valores associados a estas chaves.

Neste modelo, existe uma tabela *hash* baseada em eventos *get (consulta – leitura)* e *put (atualiza)* que permite, por conta da estrutura simples, melhor desempenho quando comparado aos demais modelos NoSQL.

A Figura 5 apresenta a estrutura do modelo de dados tipo chave/valor.

Chave1	Valor1a
	Valor1b
	Valor1c

Figura 5. Estrutura do modelo de dados NoSQL, chave-valor, NETO J.G., (2016).

A Figura 6 apresenta um exemplo de aplicação para o modelo de dados NoSQL do tipo chave/valor.

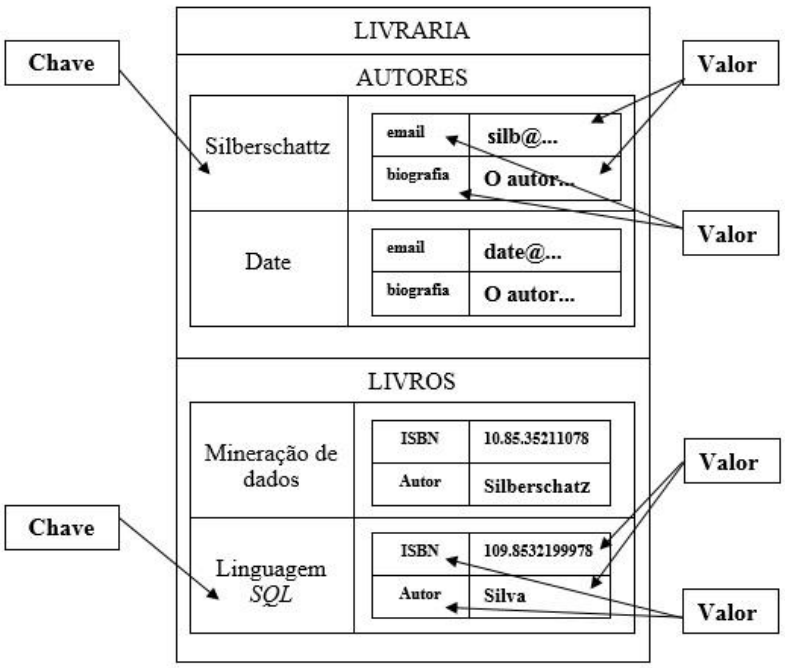


Figura 6. Exemplo de aplicação em *NoSQL*, chave-valor, NETO J.G., (2016).

2.6.2 Modelo de dados do tipo colunas

Neste modelo, existe um mapa indexado por três chaves (uma linha, uma coluna e um *timestamp*) que combinados, retornam uma *string* valorada.

A Figura 7 apresenta a estrutura do modelo de dados NoSQL, orientado a colunas.

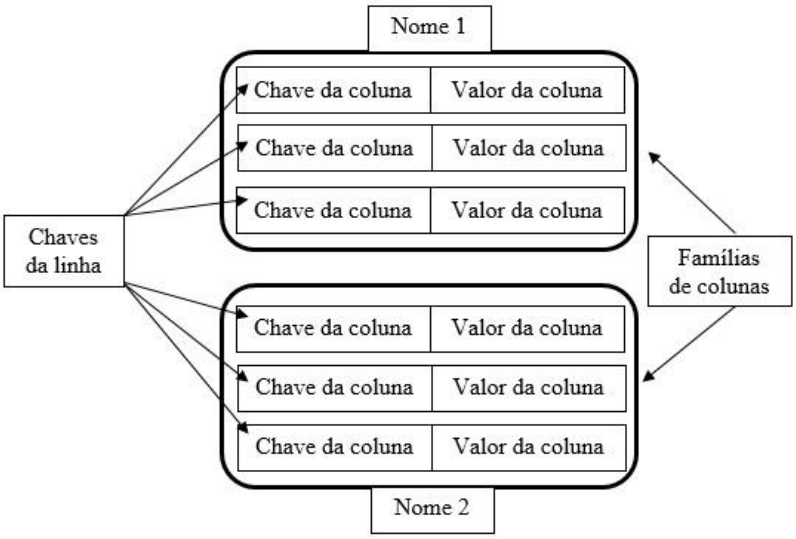


Figura 7. Estrutura do modelo NoSQL, orientado a colunas, adaptado de Fowler (2012).

A Figura 8 apresenta um exemplo de aplicação para o modelo de dados NoSQL, orientado a colunas.

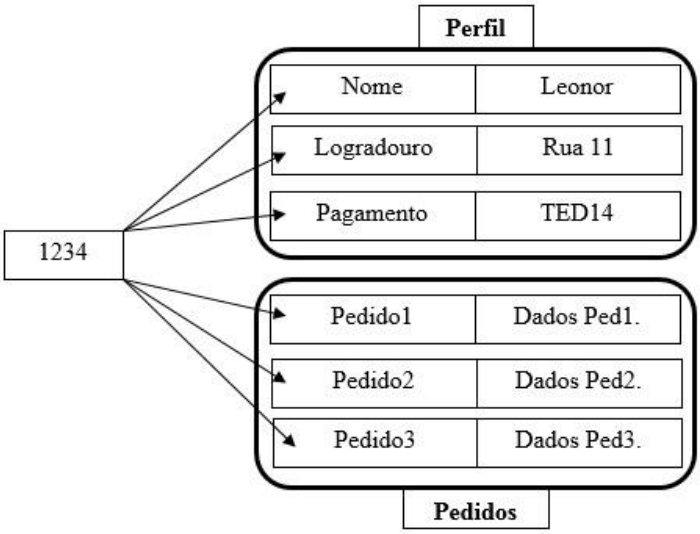


Figura 8. Exemplo de aplicação do modelo *NoSQL*, orientado a colunas adaptado de FOWLER (2012).

2.6.3 Modelo de dados do tipo orientado a documentos

Este modelo de dados é o que mais se aproxima do modelo relacional, quando se compara documentos deste modelo, com tuplas (linhas) do modelo relacional.

Neste modelo de dados, os documentos, unidades básicas de armazenamento, não precisam possuir estrutura pré-definida nem comum entre si, contendo apenas o conteúdo mais relevante do documento, além da possibilidade de um documento conter outros documentos aninhados.

A Figura 9 apresenta a estrutura do modelo de dados orientado a documentos.

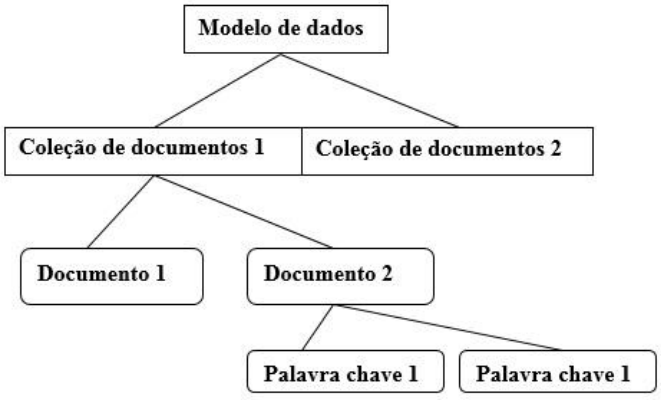


Figura 9. Estrutura do modelo *NoSQL*, orientado a documentos NETO J.G., (2016).

A Figura 10 apresenta um exemplo de aplicação para o modelo de dados NoSQL, orientado a documentos.

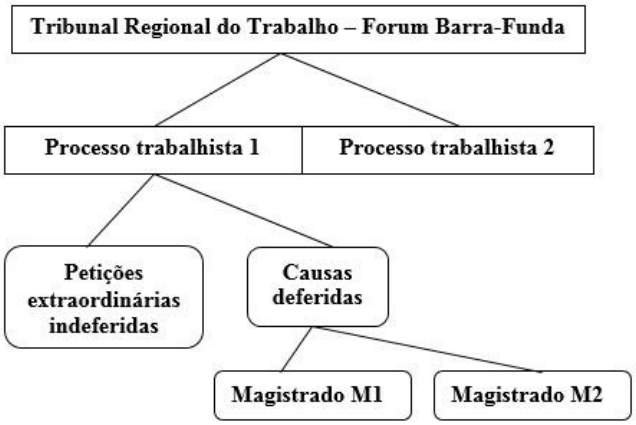


Figura 10. Exemplo de aplicação do modelo de dados *NoSQL*, orientado a documentos, NETO J.G., (2016).

2.6.4 Modelo de dados do tipo orientado a grafos

Neste modelo, os dados são armazenados em nós de um grafo e ligados pelas arestas a outros nós, conforme suas relações.

Diferentemente dos outros modelos NoSQL, neste modelo a consistência transacional é assegurada por propriedades *ACID*, logo, apresenta menor escalabilidade e desempenho.

Algumas bibliografias rotulam este modelo de dados como NewSQL, associando a ele características dos modelos NoSQL quanto ao desempenho e escalabilidade, e características do modelo relacional, quanto a consistência transacional.

Os bancos de dados **NewSQL** buscam promover a mesma melhoria de desempenho e escalabilidade dos sistemas NoSQL, não abrindo mão dos benefícios dos bancos de dados tradicionais, da linguagem SQL e das propriedades ACID. Mike Stonebreaker, fundador do VoltDB (um dos bancos de dados desse novo modelo), destacou a vantagem dos bancos de dados NewSQL por proporcionarem consultas em tempo real, além de maior capacidade de processamento. Segundo Mike, há um custo grande em não usar SQL, sendo exigido trabalho excessivo dos desenvolvedores para compensar sua ausência.

A Figura 11 apresenta a estrutura básica do modelo de dados NoSQL, do tipo grafos.

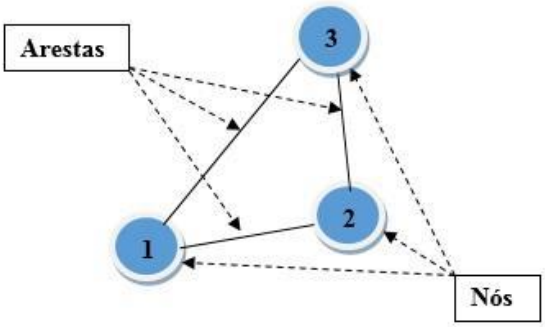


Figura 11. Estrutura do modelo de dados *NoSQL* , orientado a grafos, NETO J.G., (2016).

A Figura 12 apresenta um exemplo de aplicação para o modelo de dados NoSQL, orientado a grafos.

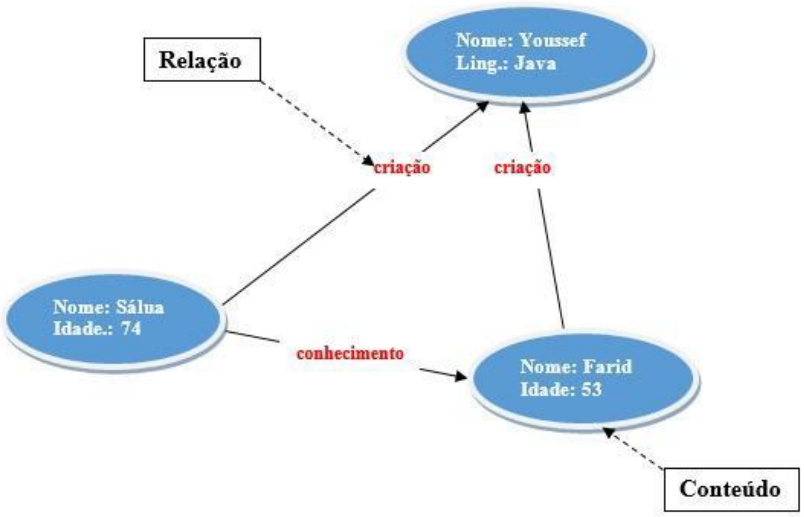


Figura 12. Exemplo de aplicação do modelo de dados *NoSQL*, orientado a grafos, NETO J.G., (2016).

2.7. COMPARATIVO ENTRE OS MODELOS DE DADOS

A Tabela 1 apresenta os modelos de dados, classificando-os quanto às principais demandas de suas evoluções, ou seja, complexidade de acesso, escalabilidade horizontal por aumento do volume de dados e facilidade em representar dados de diferentes estruturas.

Modelos	Complexidade	Escalabilidade	Representatividade
NoSQL	Variável / Baixa	Alta	Alta
Orientado a objetos	Variável	Variável/Baixa	Variável/Alta
Relacional	Variável	Variável	Variável/Baixa
Em redes	Alta	Mínima	Nula
Hierárquico	Alta	Mínima	Nula

Evolução no tempo

Tabela 1 – Os modelos de dados e suas demandas evolutivas, NETO J.G., (2016).

Analisando a Tabela 1, observa-se que, à medida que o tempo passa, os modelos de dados apresentam acessos menos complexos (mais amigáveis), maior escalabilidade horizontal e tendem a representar melhor dados com estruturas distintas.

A Tabela 2 apresenta o detalhamento dos modelos de dados *NoSQL*, comparando-os ao modelo de dados relacional. Neste comparativo, são consideradas funcionalidades tais como: desempenho em consultas envolvendo dados, estruturados ou não; escalabilidade por aumento significativo do volume de dados não estruturados; flexibilidade e complexidade nas formas de acesso; consistência do modelo; e fundamentação teórica estrutural.

Modelos	Desempenho	Escalabilidade	Flexibilidade	Complexidade	Consistência	Teoria
Orientado a grafos	Variável	Variável/Alta	Alta	Alta	Variável	Dos grafos
Orientado a documentos	Alto	Variável/Alta	Alta	Baixa	Baixa	Mínima
Tipo chave / valor	Alto	Alta	Alta	Mínima	Baixa	Mínima
Tipo colunas	Alto	Alta	Média	Baixa	Baixa	Mínima
Relacional	Variável	Variável	Baixa	Média	Alta	De conjuntos

Tabela 2 – Comparativo do modelo de dados relacional com os modelos de dados *NoSQL*, NETO J.G., (2016).

Analisando a Tabela 2, observa-se que os modelos NoSQL apresentam boa escalabilidade, e que esta característica é, na mesma proporção, observada no respectivo desempenho. Da mesma forma, existe uma relação direta entre a teoria que apoia o modelo de dados e a consistência dos dados abrigados neste modelo, fatores estes inversamente proporcionais ao desempenho.

Vale notar que no caso particular de modelos NoSQL orientados a grafo, algumas características tais como desempenho e escalabilidade se assemelham ao modelo de dados relacional, embora no modelo orientado a

grafos a representatividade de dados com diferentes estruturas (Tabela 1), seja maior.

A Figura 13 complementa a Tabela 2, comparando a complexidade de acesso (*front-end*) e o desempenho dos modelos de dados relacional e NoSQL, com o respectivo volume armazenado.

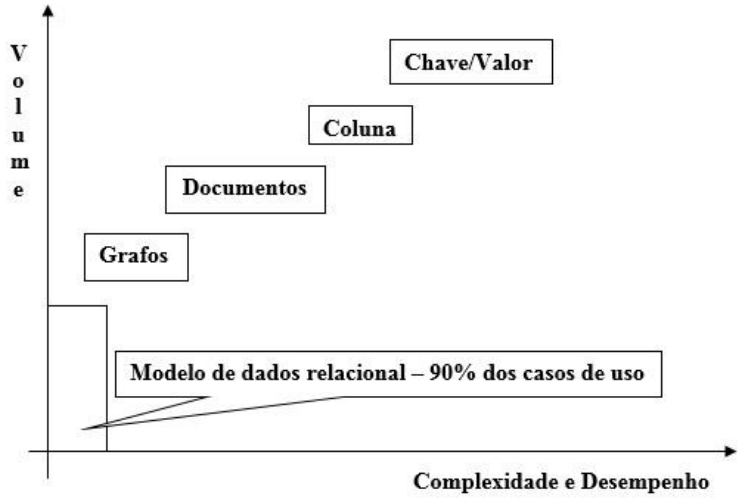


Figura 13. Comparativo dos modelos de dados *NoSQL*, adaptado de STEPPAT, N. (2011).

Analisando a Figura 13, observa-se que o modelo de dados relacional, embora atualmente suporte a maioria dos sistemas computacionais que acessam dados, mesmo tendo *interface* de acesso menos complexa, não representa o modelo de dados mais utilizado quando há a necessidade de armazenamento de grandes volumes de dados, dados estes em sua maioria, não estruturados.

Dados não estruturados tais como imagens, sons, mensagens contínuas, reportagens, páginas de navegadores ou a composição destes dados, são, no modelo de dados relacional, armazenados da mesma forma que, por exemplo, um valor numérico.

A Oracle criou em versões mais recentes de seus gerenciadores de bancos de dados (*10G*, *11G* e *12C*), apoiados em regras do modelo de dados relacional, *data-types* próprios para este fim, denominados BLOB (*Binary Large Object*) e CLOB (*Character Large Object*), coexistindo assim dados não estruturados, abrigados nestes *data-types* com dados estruturados, no mesmo modelo relacional, mas alerta sobre eventuais dificuldades associadas a estes tipos de dados no caso de procedimentos associados a salva-guarda/recuperação dos dados, manipulação de conteúdos compartilhados e sobretudo, desempenho em sistemas distribuídos, quando do aumento do volume de dados não estruturados armazenados nestes *data-types*.

Essa conjunção de fatores, demanda ao modelo relacional tempo adicional de processamento, tempo este proporcionalmente aumentado em acessos de grandes volumes de dados.

Modelos de dados mais recentes, originalmente projetados para dados distribuídos e de natureza complexa, como os modelos de dados NoSQL, apropriados para grandes volumes de dados, altamente escaláveis horizontalmente, disponibilizando os dados em tempos computacionais menos dilatados e principalmente, capazes de coexistir estruturas distintas, representam a tendência natural de evolução do modelo de dados relacional, cujo propósito na ocasião da sua concepção era abrigar e prover acesso a dados estruturados de forma bidimensional, haja vista nesta ocasião, a inexistência de dados não estruturados.

Referencialmente falando, observa-se nos modelos de dados NoSQL, formas particulares no tratamento dos dados, adequando seu uso à aplicação e não necessariamente utilizando-os sobre regras menos flexíveis, como as regras impostas pelo modelo relacional, adaptando neste último, a aplicação ao modelo de dados.

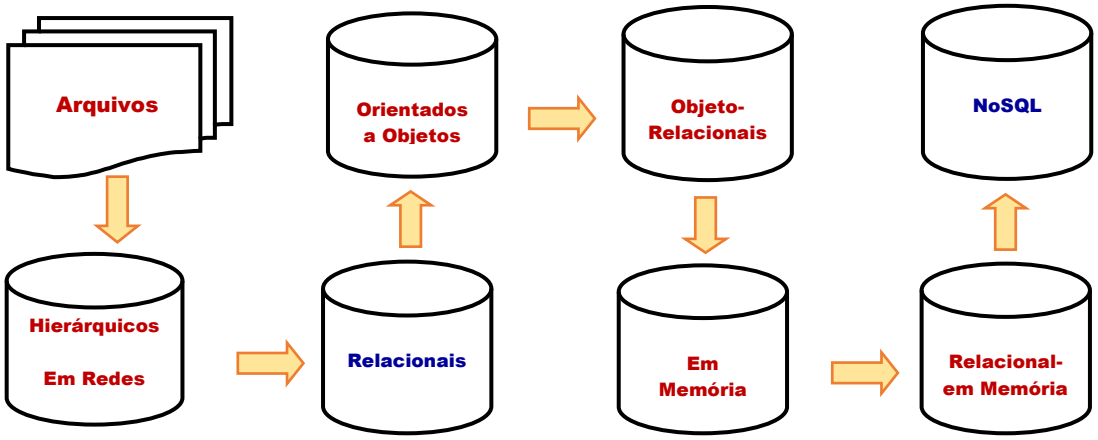
Com relação as transações, nos modelos de dados NoSQL, o conceito ACID é, na maioria das vezes, substituído pelos conceitos do teorema CAP (*Consistency, Availability e Partition Tolerance*) e pelo conceito *BASE* (*Basically Available, Soft State and Eventual Consistency*) onde, no primeiro conceito, demonstra ser impossível para um modelo de dados distribuído ser simultaneamente consistente, estar sempre disponível e intolerante a falhas, como sugere o modelo de dados relacional.

Assim, o teorema CAP sugere somente ser possível a combinação das três propriedades (Consistência, Disponibilidade e Tolerância a falhas no particionamento), duas a duas, ou seja, quando combinadas consistência e disponibilidade, combinação mais próxima dos modelos de dados relacionais, a tolerância a falhas em sistemas com dados distribuídos não é garantida. Quando combinadas disponibilidade com tolerância a falhas, a consistência não é assegurada, assim como a combinação de consistência e tolerância a falhas não assegura disponibilidade dos dados.

Por sua vez, regras do conceito BASE sugerem modelos de dados disponíveis sob demanda, sem a necessidade de estarem consistentes o tempo todo, possibilitando consistência eventual.

RESUMO DA EVOLUÇÃO

- **Sistemas de Processamento de Arquivos** - Clipper, Cobol, Dataflex, etc.
- **Sistemas Gerenciadores de Bancos de Dados** (Hierarquico – Rede – Relacional) - Visual Basic, Delphi, Centura, etc.
- **Sistemas Gerenciadores de Bancos de Dados Orientados a Objetos** - Java, etc.
- **Sistemas Gerenciadores de Bancos de Dados Objeto-Relacionais** - Java com Hibernate e C# com Microsoft Entity Framework (ORM – Object Relational Management).
- **Bancos de Dados em Memória** (IMDB - *In-Memory Databases*)
- Cogitasse uma **tecnologia hibrida** entre os SGBD e os IMDB
- **Bancos de Dados NoSQL**



OBS.: Não é uma sequência obrigatória para a migração de bases de dados