



Bancos de Dados



Bancos de Dados

Autor: Gabriel Felippe

Website: akiradev.netlify.app

Introdução

- Um **banco de dados** é uma **coleção organizada de dados**, geralmente armazenados e acessados eletronicamente a partir de um sistema de computador.



Introdução

- Quando os **bancos de dados** são mais complexos, eles geralmente são desenvolvidos usando **técnicas de design e modelagem formais**.
- O **database management system** (DBMS) é o software que interage com os usuários finais, aplicativos e o próprio banco de dados para capturar e analisar os dados.
- O software **DBMS** também abrange os recursos básicos fornecidos para administrar o banco de dados. A soma total do banco de dados, do **DBMS** e dos aplicativos associados pode ser referido como um "sistema de banco de dados".

Introdução

- Os cientistas da computação podem classificar os sistemas de gerenciamento de banco de dados (DBMS) de acordo com os modelos de banco de dados que eles suportam.
- Os bancos de dados relacionais tornaram-se dominantes na década de 1980, estes que modelam os dados como linhas e colunas em uma série de tabelas, e a grande maioria usa SQL para escrever e consultar dados.
- Na década de 2000, os bancos de dados não relacionais se tornaram populares, chamados de NoSQL, porque usam diferentes linguagens de consulta (query languages).

Introdução

- A seguir temos o exemplo da instrução **SELECT**, que é usada para selecionar dados de uma tabela de banco de dados:

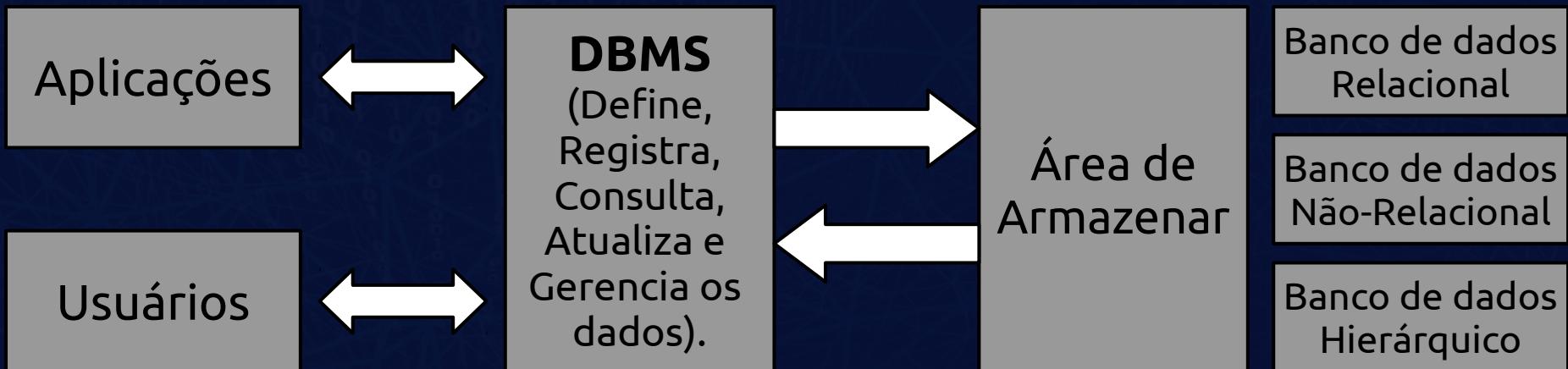
```
mysql> select * from filmes LIMIT 11;
+----+-----+-----+-----+
| id | titulo          | ano_lancamento | genero |
+----+-----+-----+-----+
| 1  | Forrest Gump   | 1994           | Drama   |
| 2  | Gladiator       | 2000           | Action   |
| 3  | Titanic         | 1997           | Romance |
| 4  | The Godfather  | 1972           | Crime   |
| 5  | Inception       | 2010           | Adventure |
| 6  | Pulp Fiction   | 1994           | Crime   |
| 7  | The Matrix      | 1999           | Cyberpunk |
| 8  | Johnny Mnemonic| 1995           | Cyberpunk |
| 9  | Dark City       | 1998           | Science Fiction |
| 10 | They Live       | 1988           | Science Fiction |
| 11 | The Truman Show| 1998           | Comedy Drama |
+----+-----+-----+-----+
11 rows in set (0.00 sec)
```

Terminologia e Visão Geral

- Formalmente, um "**banco de dados**" refere-se a um conjunto de dados relacionados e à forma como estes são organizados.
- O acesso a esses dados é geralmente fornecido por um "**sistema de gerenciamento de banco de dados**" (DBMS) que consiste em um conjunto integrado de software de computador que permite aos usuários interagir com um ou mais bancos de dados e fornece acesso a todos os dados contidos no banco de dados (embora possam existir restrições que limitam o acesso a dados específicos).

Terminologia e Visão Geral

- O **DBMS** fornece várias funções que permitem a entrada, o armazenamento e a recuperação de grandes quantidades de informações e fornece maneiras de gerenciar como essas informações são organizadas.

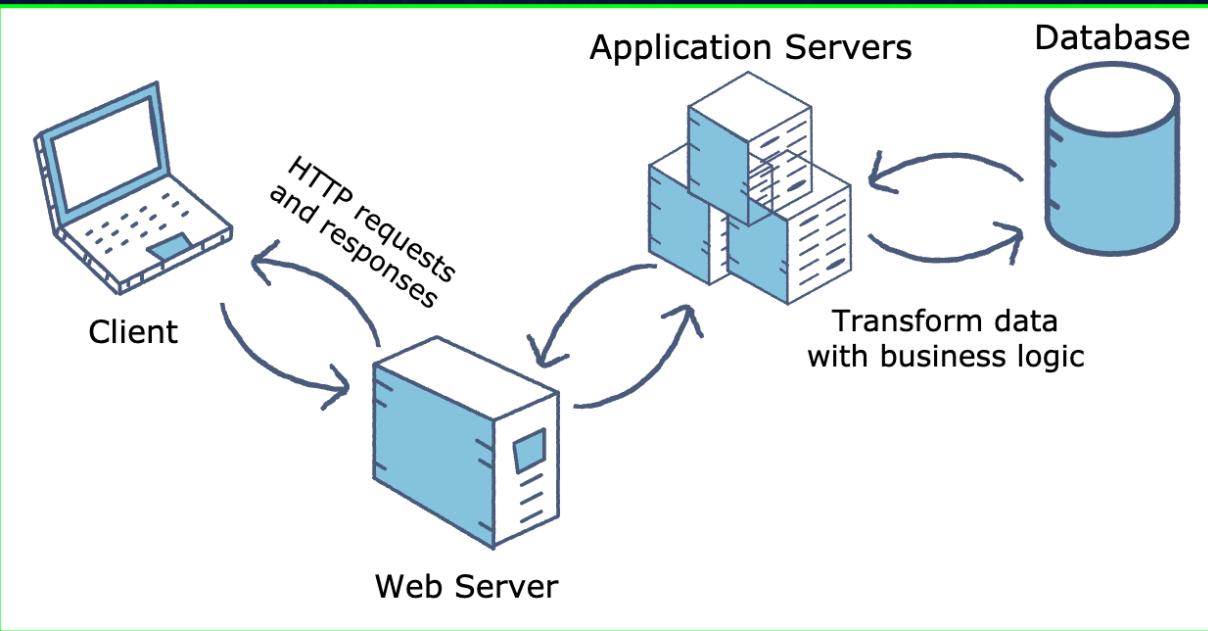


Terminologia e Visão Geral

- Os DBMS's existentes fornecem várias funções que permitem o gerenciamento de um banco de dados e seus dados, que podem ser classificados em quatro grupos funcionais principais:
 - **Definição:** Criação, modificação e remoção de definições que definem a organização dos dados.
 - **Atualização:** Inserção, modificação e exclusão dos dados reais.
 - **Recuperação:** Fornecimento de informações em uma forma diretamente utilizável ou para processamento posterior por outros aplicativos. Os dados recuperados podem ser disponibilizados em um formato basicamente igual ao que são armazenados no banco de dados ou em um novo formato obtido pela alteração ou combinação de dados existentes no banco de dados.
 - **Administração:** Registrando e monitorando usuários, reforçando a segurança dos dados, monitorando o desempenho, mantendo a integridade dos dados, lidando com o controle de simultaneidade e recuperando informações que foram corrompidas por algum evento, como uma falha inesperada do sistema.

Terminologia e Visão Geral

- Fisicamente, os servidores de banco de dados são **computadores dedicados** que mantêm os bancos de dados reais e executam apenas o DBMS e o software relacionado.

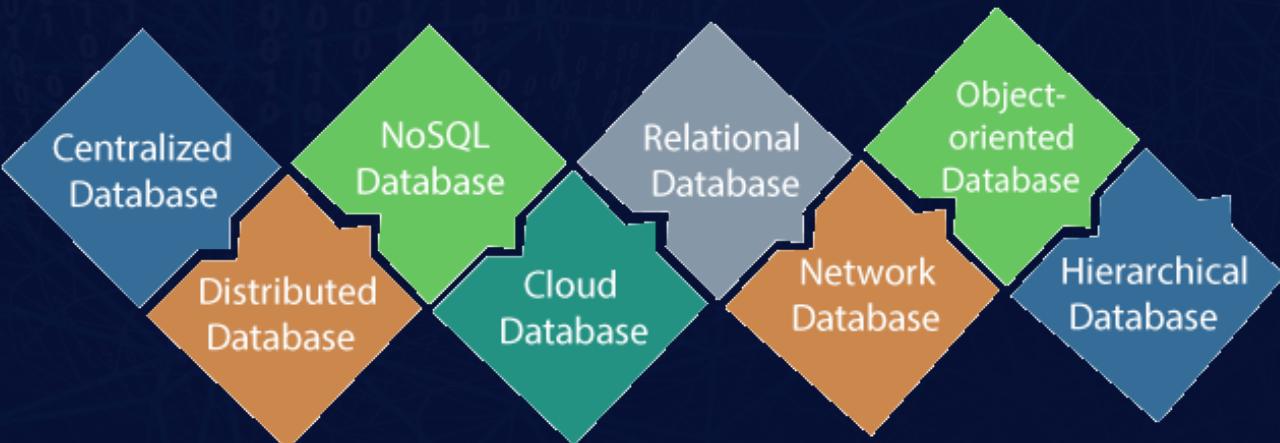


Terminologia e Visão Geral

- Os servidores de banco de dados geralmente são computadores com **multiprocessadores**, com memória generosa e arrays de **disco RAID** usados para armazenamento estável.
- Aceleradores de hardware de banco de dados, conectados a um ou mais servidores por meio de um canal de alta velocidade, também são usados em ambientes de processamento de transações de grande volume.
- Os DBMS's são encontrados no coração da maioria dos aplicativos de banco de dados. Os DBMS's podem ser construídos em torno de um **kernel multitarefa** personalizado com suporte de rede integrado, mas os DBMS's modernos geralmente contam com um sistema operacional padrão para fornecer essas funções.

Terminologia e Visão Geral

- Bancos de dados e DBMS's podem ser categorizados de acordo com o modelo de banco de dados que eles suportam (como relacional ou XML), o tipo de computador em que são executados (de um cluster de servidor a um telefone celular), a linguagem de consulta usada para acessar o banco de dados (como SQL ou XQuery) e sua engenharia interna, que afeta o desempenho, a escalabilidade, a resiliência e a segurança.



História

- Os tamanhos, capacidades e desempenho dos bancos de dados e seus respectivos DBMS's aumentaram em ordens de magnitude.
- Esses aumentos de desempenho foram possibilitados pelo progresso da tecnologia nas áreas de **processadores**, **memória de computador**, **armazenamento** de computador e **redes de computadores**.
- O conceito de banco de dados foi possibilitado pelo surgimento de meios de armazenamento de acesso direto, como **discos magnéticos**, que se tornaram amplamente disponíveis em meados da década de 1960; os sistemas anteriores dependiam do armazenamento sequencial de dados em fita magnética.

História

- ❑ O desenvolvimento subsequente da tecnologia de banco de dados pode ser dividido em três eras com base no modelo ou estrutura de dados:
 - ❑ Navegacional,
 - ❑ SQL/relacional
 - ❑ Pós-relacional.
- ❑ Os dois principais modelos de dados de navegação iniciais eram o **modelo hierárquico** e o modelo **CODASYL** (modelo de rede).
- ❑ Eles foram caracterizados pelo uso de ponteiros (geralmente endereços de disco físico) para seguir as relações de um registro para outro.

História

- O **modelo relacional**, proposto pela primeira vez em 1970 por **Edgar F. Codd**, partiu dessa tradição ao insistir que os aplicativos deveriam pesquisar dados por conteúdo, em vez de seguir links.
- O modelo relacional emprega conjuntos de tabelas, cada uma usada para um tipo diferente de entidade.
- Somente em meados da década de 1980 o hardware de computação se tornou poderoso o suficiente para permitir a ampla implantação de sistemas relacionais (SGBD's mais aplicativos).

História

- ❑ No início da década de 1990, no entanto, os sistemas relacionais dominavam em todos os aplicativos de processamento de dados em grande escala e, a partir de 2018, eles continuam dominantes: **IBM DB2**, **Oracle**, **MySQL** e **Microsoft SQL Server** são os DBMS's mais pesquisados.
- ❑ A linguagem de banco de dados dominante, SQL padronizada para o modelo relacional, influenciou as linguagens de banco de dados para outros modelos de dados.
- ❑ Os bancos de dados de objetos foram desenvolvidos na década de 1980 para superar a inconveniência da **object-relational impedance mismatch**, o que levou à criação do termo "pós-relacional" e também ao desenvolvimento de bancos de dados híbridos de **objeto-relacional**.

História

- A próxima geração de bancos de dados pós-relacionais no final dos anos 2000 tornou-se conhecida como bancos de dados **NoSQL**, introduzindo armazenamentos rápidos de **valores-chave** e bancos de dados **orientados a documentos**.
- Uma "próxima geração" concorrente, conhecida como bancos de dados **NewSQL**, tentou novas implementações que mantiveram o modelo relacional/SQL ao mesmo tempo em que buscava corresponder ao alto desempenho do NoSQL em comparação com os DBMS's relacionais disponíveis comercialmente.

Década de 1960, Navigational DBMS

- A introdução do termo **banco de dados** coincidiu com a disponibilidade de armazenamento de acesso direto (discos e tambores) a partir de meados da década de 1960.
- O termo representava um contraste com os sistemas baseados em fita do passado, permitindo o uso interativo compartilhado em vez do diário **batch processing**.
- O Oxford English Dictionary cita um relatório de 1962 da System Development Corporation da Califórnia como o primeiro a usar o termo "banco de dados" em um sentido técnico específico.

Década de 1960, Navigational DBMS

- À medida que os computadores cresceram em velocidade e capacidade, vários sistemas de banco de dados de uso geral surgiram; em meados da década de 1960, vários desses sistemas entraram em uso comercial.
- O interesse por um padrão começou a crescer, e Charles Bachman, autor de um desses produtos, o Integrated Data Store (IDS), fundou o Database Task Group dentro do CODASYL, o grupo responsável pela criação e padronização do COBOL.
- Em 1971, o Database Task Group entregou seu padrão, que geralmente ficou conhecido como abordagem CODASYL, e logo vários produtos comerciais baseados nessa abordagem entraram no mercado.

Década de 1960, Navigational DBMS

- A abordagem **CODASYL** ofereceu aos aplicativos a capacidade de navegar em um conjunto de dados vinculado que foi formado em uma grande rede.
- Os aplicativos podem encontrar registros por um dos três métodos:
 1. Uso de uma chave primária (conhecida como chave CALC, normalmente implementada por **hashing**).
 2. Relacionamentos de navegação (chamados conjuntos) de um registro para outro.
 3. Scanning de todos os registros em uma ordem sequencial.

Década de 1960, Navigational DBMS

- Sistemas posteriores adicionaram **B-trees** para fornecer caminhos de acesso alternativos.
- Muitos bancos de dados CODASYL também adicionaram uma linguagem de consulta declarativa para usuários finais (diferente da API de navegação).
- No entanto, os bancos de dados CODASYL eram complexos e exigiam treinamento e esforço significativos para produzir aplicativos úteis.
- A **IBM** também tinha seu próprio DBMS em 1966, conhecido como **Information Management System (IMS)**.

Década de 1960, Navigational DBMS

- ❑ IMS foi um desenvolvimento de software escrito para o **programa Apollo** no **System/360**.
- ❑ O IMS era geralmente semelhante em conceito ao CODASYL, mas usava uma hierarquia estrita para seu modelo de navegação de dados, em vez do modelo de rede do CODASYL.
- ❑ Ambos os conceitos tornaram-se mais tarde conhecidos como bancos de dados de navegação devido à forma como os dados eram acessados: o termo foi popularizado pela apresentação do Prêmio Turing de Bachman em 1973, *The Programmer as Navigator*.
- ❑ O IMS é classificado pela IBM como um **banco de dados hierárquico**. Os bancos de dados TOTAL da IDMS e da Cincom Systems são classificados como bancos de dados de rede.

Década de 1970, Relational DBMS

- **Edgar F. Codd** trabalhou na IBM em San Jose, Califórnia, em um de seus escritórios secundários que estava principalmente envolvido no desenvolvimento de sistemas de disco rígido.
- Ele estava insatisfeito com o modelo de navegação da abordagem CODASYL, notadamente a falta de um mecanismo de "busca".
- Em 1970, ele escreveu uma série de artigos que delinearam uma nova abordagem para a construção de banco de dados que culminou no inovador *A Relational Model of Data for Large Shared Data Banks*.

Década de 1970, Relational DBMS

- Neste artigo, ele descreveu um novo sistema para armazenar e trabalhar com grandes bancos de dados.
- Em vez de os registros serem armazenados em algum tipo de lista vinculada de registros de forma livre como em CODASYL, a ideia de Codd era organizar os dados como uma série de "tabelas", cada tabela sendo usada para um tipo diferente de entidade.
- Cada tabela conteria um número fixo de colunas contendo os atributos da entidade.

Década de 1970, Relational DBMS

- Uma ou mais colunas de cada tabela foram designadas como uma **chave primária** pela qual as linhas da tabela puderam ser identificadas de forma única; as referências cruzadas entre as tabelas sempre usaram essas chaves primárias, em vez de endereços de disco, e as consultas uniriam as tabelas com base nessas relações de chave, usando um conjunto de operações baseado no sistema matemático de **cálculo relacional** (do qual o modelo tira seu nome).
- A divisão dos dados em um conjunto de tabelas (ou relações) normalizadas visava garantir que cada "fato" fosse armazenado apenas uma vez, simplificando as operações de atualização.
- As tabelas virtuais chamadas *views* podem apresentar os dados de maneiras diferentes para diferentes usuários, mas as *views* não podem ser atualizadas diretamente.

Década de 1970, Relational DBMS

- Codd usou termos matemáticos para definir o modelo: relações, tuplas e domínios em vez de **tabelas, linhas e colunas**.

Tabela			
Linha			
	Coluna		

Década de 1970, Relational DBMS

- O uso de **chaves primárias** (identificadores orientados ao usuário) para representar relacionamentos entre tabelas, em vez de endereços de disco, teve duas motivações principais.
- Do ponto de vista da engenharia, ele permitiu que as tabelas fossem realocadas e redimensionadas sem a custosa reorganização do banco de dados.
- Mas **Codd** estava mais interessado na diferença na semântica: o uso de identificadores explícitos tornava mais fácil definir **operações de atualização** com definições matemáticas claras e também permitia que **operações de consulta** fossem definidas em termos da disciplina estabelecida de cálculo de predicado de primeira ordem; como essas operações têm propriedades matemáticas claras, é possível reescrever consultas de maneiras comprovadamente corretas, que é a base da otimização de consulta.
- Não há perda de expressividade em relação aos modelos hierárquico ou de rede, embora as conexões entre as tabelas não sejam mais tão explícitas.

Década de 1970, Relational DBMS

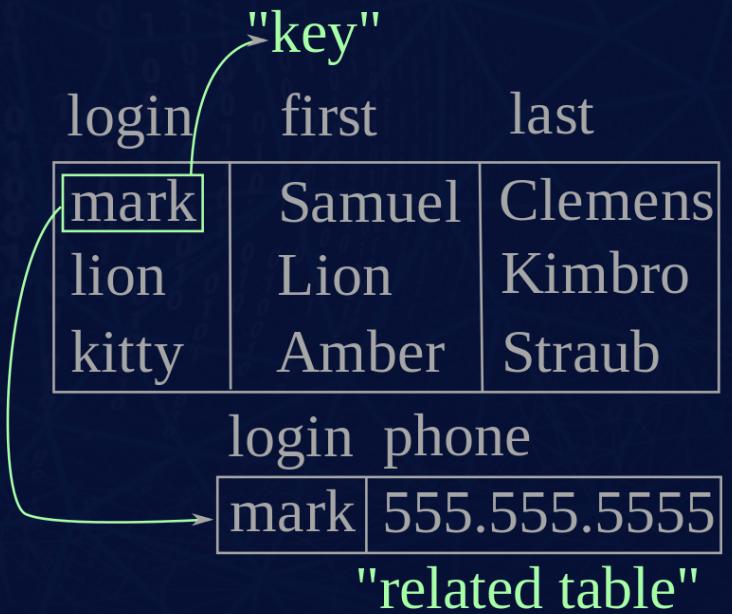
- ❑ Nos **modelos hierárquico e de rede**, os registros podiam ter uma estrutura interna complexa. Por exemplo, o histórico de salários de um funcionário pode ser representado como um "grupo recorrente" no registro do funcionário.
- ❑ No **modelo relacional**, o processo de normalização fez com que essas estruturas internas fossem substituídas por dados mantidos em várias tabelas, conectadas apenas por chaves lógicas.

Década de 1970, Relational DBMS

- Por exemplo, um uso comum de um sistema de banco de dados é rastrear informações sobre usuários, seu nome, informações de login, vários endereços e números de telefone.
- Na abordagem de navegação, todos esses dados seriam colocados em um único registro de comprimento variável.
- Na abordagem relacional, os dados seriam normalizados em uma tabela de usuário, uma tabela de endereço e uma tabela de número de telefone (por exemplo).
- Os registros seriam criados nessas tabelas opcionais apenas se o endereço ou os números de telefone fossem realmente fornecidos.

Década de 1970, Relational DBMS

- No **modelo relacional**, os registros são "vinculados" por meio de chaves virtuais não armazenadas no banco de dados, mas definidas conforme necessário entre os dados contidos nos registros.



Década de 1970, Relational DBMS

- Além de identificar linhas/registros usando identificadores lógicos em vez de endereços de disco, Codd mudou a maneira como os aplicativos reuniam dados de vários registros.
- Em vez de exigir que os aplicativos coletem dados, um registro por vez, navegando pelos links, eles usariam uma linguagem de consulta declarativa que expressasse quais dados eram necessários, em vez do caminho de acesso pelo qual eles deveriam ser encontrados.
- Encontrar um caminho de acesso eficiente aos dados tornou-se responsabilidade do sistema de gerenciamento de banco de dados, e não do programador do aplicativo.
- Esse processo, denominado otimização de consultas, dependia do fato de as consultas serem expressas em termos de lógica matemática.

Década de 1970, Relational DBMS

- O artigo de Codd foi escolhido por duas pessoas em Berkeley, Eugene Wong e Michael Stonebraker.
- Eles começaram um projeto conhecido como INGRES usando fundos que já haviam sido alocados para um projeto de banco de dados geográfico e alunos programadores para produzir código.
- Começando em 1973, o INGRES entregou seus primeiros produtos de teste que geralmente estavam prontos para uso generalizado em 1979.
- O INGRES era semelhante ao Sistema R de várias maneiras, incluindo o uso de uma "linguagem" para acesso a dados, conhecida como QUEL. Com o tempo, o INGRES mudou para o padrão SQL emergente.

Década de 1970, Relational DBMS

- A própria IBM fez um teste de implementação do modelo relacional, **PRTV**, e um de produção, **Business System 12**, ambos agora descontinuados. A empresa Honeywell escreveu **MRDS** para **Multics**, e agora existem duas novas implementações: **Alphora Dataphor** e **Rel**. A maioria das outras implementações de DBMS geralmente chamadas de relacionais são, na verdade, DBMS's SQL.
- Em 1970, a Universidade de Michigan iniciou o desenvolvimento do Sistema de Gerenciamento de Informações **MICRO** baseado em D.L. Child's Set-Theoretic Data model.
- O **MICRO** foi usado para gerenciar conjuntos de dados muito grandes pelo Departamento do Trabalho dos EUA, pela Agência de Proteção Ambiental dos EUA e por pesquisadores da Universidade de Alberta, da Universidade de Michigan e da Wayne State University. Ele foi executado em computadores mainframe IBM usando o **Michigan Terminal System**. O sistema permaneceu em produção até 1998.

Abordagem Integrada

- Nas décadas de 1970 e 1980, foram feitas tentativas de construir sistemas de banco de dados com hardware e software integrados. A filosofia subjacente era que essa integração proporcionaria melhor desempenho a um custo menor. Os exemplos foram **IBM System/38**, a primeira oferta da Teradata, e a máquina de banco de dados Britton Lee, Inc.
- Outra abordagem de suporte de hardware para gerenciamento de banco de dados era o acelerador **CAFS** da **ICL**, um controlador de disco de hardware com recursos de pesquisa programáveis. No longo prazo, esses esforços foram geralmente malsucedidos porque as máquinas de banco de dados especializadas não conseguiam acompanhar o rápido desenvolvimento e progresso dos computadores de uso geral.
- Assim, a maioria dos sistemas de banco de dados hoje em dia são sistemas de software executados em hardware de uso geral, usando armazenamento de dados de computador de uso geral. No entanto, essa ideia ainda é perseguida para determinados aplicativos por algumas empresas como **Netezza** e Oracle (**Exadata**).

Final da Década de 1970, SQL DBMS

- A IBM começou a trabalhar em um sistema de protótipo vagamente baseado nos conceitos de Codd como System R no início dos anos 1970.
- A primeira versão ficou pronta em 1974/5, e o trabalho então começou em sistemas multi-table nos quais os dados podiam ser divididos de forma que todos os dados para um registro (alguns dos quais são opcionais) não precisassem ser armazenados em um único grande "pedaço".
- As versões multiusuário subsequentes foram testadas por clientes em 1978 e 1979, quando uma linguagem de consulta padronizada SQL foi adicionada.
- As ideias de Codd estavam se estabelecendo como viáveis e superiores ao CODASYL, levando a IBM a desenvolver uma verdadeira versão de produção do System R, conhecida como SQL/DS e, mais tarde, Database 2 (DB2).
- O banco de dados Oracle de Larry Ellison começou de uma perspectiva diferente, com base nos documentos da IBM sobre o System R. Embora as implementações do Oracle V1 tenham sido concluídas em 1978, não foi até a versão 2 do Oracle quando Ellison venceu a IBM no mercado em 1979.

Final da Década de 1970, SQL DBMS

- Stonebraker passou a aplicar as lições do INGRES para desenvolver um novo banco de dados, Postgres, que agora é conhecido como PostgreSQL. O PostgreSQL é freqüentemente usado para aplicativos globais de missão crítica (os registros de nomes de domínio .org e .info o usam como seu principal armazenamento de dados, assim como muitas grandes empresas e instituições financeiras).
- Na Suécia, o artigo de Codd também foi lido e o Mimer SQL foi desenvolvido a partir de meados da década de 1970 na Universidade de Uppsala. Em 1984, este projeto foi consolidado em uma empresa independente.
- Outro modelo de dados, o modelo entidade-relacionamento, surgiu em 1976 e ganhou popularidade para o design de banco de dados, pois enfatizava uma descrição mais familiar do que o modelo relacional anterior. Posteriormente, os constructos entidade-relacionamento foram adaptados como um constructo de modelagem de dados para o modelo relacional, e a diferença entre os dois tornou-se irrelevante.

Década de 1980, em Desktop

- A década de 1980 marcou o início da era da **computação desktop**.
- Os novos computadores capacitaram seus usuários com planilhas como **Lotus 1-2-3** e software de banco de dados como **dBASE**.
- O produto **dBASE** era leve e fácil de entender para qualquer usuário de computador.
- C. Wayne Ratliff, o criador do **dBASE**, afirmou: **dBASE** era diferente de programas como BASIC, C, FORTRAN e COBOL porque muito do trabalho difícil já havia sido feito.
- A manipulação de dados é feita pelo **dBASE** em vez de pelo usuário, para que o usuário possa se concentrar no que está fazendo, em vez de ter que mexer com os detalhes complicados de abrir, ler e fechar arquivos e gerenciar a alocação de espaço. **dBASE** foi um dos títulos de software mais vendidos na década de 1980 e no início da década de 1990.

Década de 1990, Orientação a Objetos

- A década de 1990, junto com o aumento da **programação orientada a objetos**, viu um crescimento na forma como os dados em vários bancos de dados eram tratados.
- Programadores e designers começaram a tratar os dados em seus bancos de dados como **objetos**.
- Isso quer dizer que, se os dados de uma pessoa estivessem em um banco de dados, os atributos dessa pessoa, como endereço, número de telefone e idade, agora eram considerados pertencentes a essa pessoa em vez de dados estranhos.
- Isso permite que as relações entre os dados sejam relações com objetos e seus atributos e não com campos individuais.

Década de 1990, Orientação a Objetos

- O termo "object-relational impedance mismatch" descreveu a inconveniência de traduzir entre objetos programados e tabelas de banco de dados.
- Bancos de dados de objetos e bancos de dados de objetos relacionais tentam resolver esse problema fornecendo uma linguagem orientada a objetos (às vezes como extensões de SQL) que os programadores podem usar como alternativa ao SQL puramente relacional.
- No lado da programação, as bibliotecas conhecidas como object-relational mappings (ORM's) tentam resolver o mesmo problema.

Década de 2000, NoSQL e NewSQL

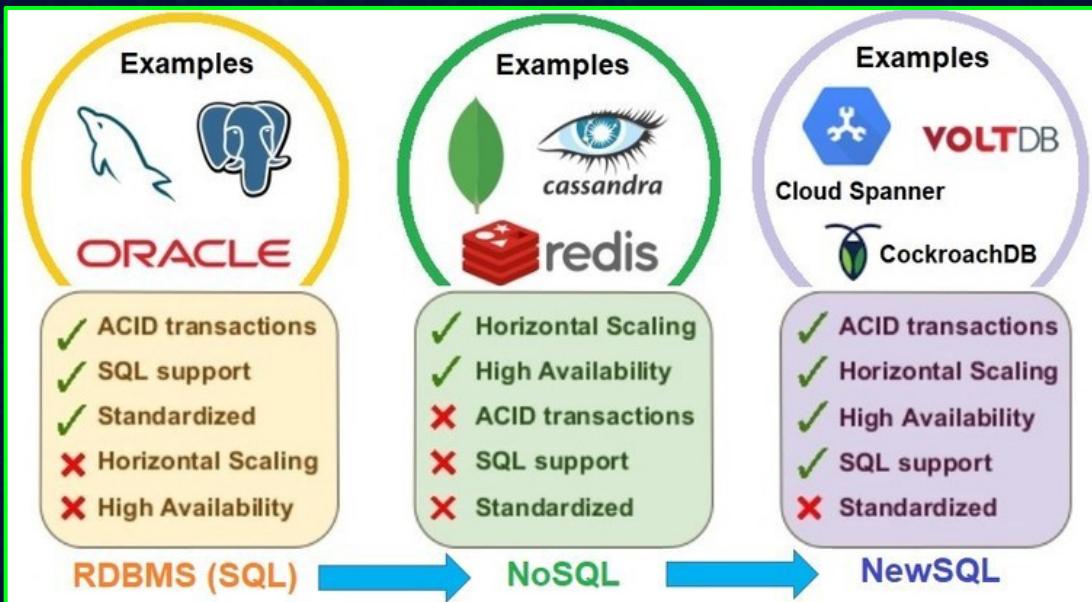
- Os bancos de dados XML são um tipo de banco de dados orientado a documentos estruturado que permite a consulta com base em atributos de documentos XML.
- Os bancos de dados XML são usados principalmente em aplicativos onde os dados são convenientemente vistos como uma coleção de documentos, com uma estrutura que pode variar de muito flexível à altamente rígida: os exemplos incluem artigos científicos, patentes, livros, declarações de impostos e registros pessoais.

Década de 2000, NoSQL e NewSQL

- Os bancos de dados **NoSQL** são frequentemente muito rápidos, não requerem esquemas de tabela fixos, evitam operações de junção armazenando dados **desnormalizados** e são projetados para **escalar horizontalmente**.
- Nos últimos anos, tem havido uma forte demanda por bancos de dados amplamente distribuídos com alta tolerância de partição, mas de acordo com o **CAP theorem**, é impossível para um **sistema distribuído** fornecer simultaneamente garantias de consistência, disponibilidade e tolerância de partição.
- Um sistema distribuído pode satisfazer quaisquer duas dessas garantias ao mesmo tempo, mas não todas as três. Por esse motivo, muitos bancos de dados **NoSQL** estão usando o que é chamado de **eventual consistency** para fornecer garantias de disponibilidade e tolerância de partição com um nível reduzido de consistência de dados.

Década de 2000, NoSQL e NewSQL

- ❑ **NewSQL** é uma classe de bancos de dados relacionais modernos que visa fornecer o mesmo desempenho escalonável de sistemas **NoSQL** para cargas de trabalho de processamento de transações online (leitura e gravação), enquanto ainda usa SQL e mantém as garantias **ACID** de um sistema de banco de dados tradicional.



Casos de Uso

- Os bancos de dados são usados para apoiar as operações internas das organizações e para sustentar as interações online com clientes e fornecedores.
- Os bancos de dados são usados para armazenar informações administrativas e dados mais especializados, como dados de engenharia ou modelos econômicos.
- Os exemplos incluem sistemas computadorizados de **biblioteca**, sistemas de **reserva de voos**, sistemas computadorizados de **inventário de peças** e muitos sistemas de **gerenciamento de conteúdo** que armazenam sites como coleções de páginas da Web em um banco de dados.

Classificação

- Uma forma de classificar bancos de dados envolve o tipo de seu **conteúdo**, por exemplo: objetos bibliográficos, documentos-texto, estatísticos ou multimídia.
- Outra forma é por sua **área de aplicação**, por exemplo: contabilidade, composições musicais, filmes, bancos, manufatura ou seguros.
- Uma terceira maneira é por algum **aspecto técnico**, como a estrutura do banco de dados ou tipo de interface.
- As seções a seguir listam alguns dos adjetivos usados para caracterizar diferentes tipos de bancos de dados.

In-Memory Database

- Um banco de dados na memória (**in-memory database**) é um banco de dados que reside principalmente na memória principal, mas normalmente é protegido por armazenamento de dados de computador não volátil.
- Os bancos de dados de memória principal são mais rápidos do que os bancos de dados de disco e, portanto, são freqüentemente usados onde o tempo de resposta é crítico, como em equipamentos de rede de telecomunicações.

Active Database

- Um banco de dados ativo (**active database**) inclui uma arquitetura orientada a eventos que pode responder às condições dentro e fora do banco de dados.
- Os possíveis usos incluem monitoramento de segurança, alertas, coleta de estatísticas e autorização.
- Muitos bancos de dados fornecem recursos de banco de dados ativos na forma de **database triggers**.

Cloud Database

- Um banco de dados em nuvem (**cloud database**) depende da tecnologia de nuvem.
- Tanto o banco de dados quanto a maior parte de seu DBMS residem remotamente, "na nuvem", enquanto seus aplicativos são desenvolvidos por programadores e, posteriormente, mantidos e usados por usuários finais por meio de um **navegador da web** e **API's** abertas.

Data Warehouses

- Os armazéns de dados (**data warehouses**) arquivam dados de bancos de dados operacionais e, frequentemente, de fontes externas, como empresas de pesquisa de mercado.
- O warehouse se torna a fonte central de dados para uso por gerentes e outros usuários finais que podem não ter acesso aos dados operacionais.
- Por exemplo, os dados de vendas podem ser agregados a totais semanais e convertidos de códigos de produto internos para usar **UPCs** para que possam ser comparados com dados **ACNielsen**.
- Alguns componentes básicos e essenciais do armazenamento de dados incluem a extração, análise e mineração de dados, transformação, carregamento e gerenciamento de dados para torná-los disponíveis para uso posterior.

Document-Oriented Database

- Um banco de dados orientado a documentos (**document-oriented database**) é projetado para armazenar, recuperar e gerenciar informações orientadas a documentos ou semiestruturadas.
- Bancos de dados orientados a documentos são uma das principais categorias de bancos de dados **NoSQL**.

Graph Database

- Um banco de dados de grafo (**graph database**) é um tipo de banco de dados **NoSQL** que usa estruturas de grafo com nós, arestas e propriedades para representar e armazenar informações.
- Os bancos de dados de grafos gerais que podem armazenar qualquer grafo são diferentes dos bancos de dados de grafos especializados, como **triplestores** e bancos de dados de rede (**network databases**).

Operational Databases

- Os bancos de dados operacionais (**operational databases**) armazenam dados detalhados sobre as operações de uma organização. Eles normalmente processam volumes relativamente altos de atualizações usando transações.
- Os exemplos incluem bancos de dados de clientes que registram informações de contato, crédito e demográficas sobre os clientes de uma empresa, bancos de dados pessoais que contêm informações como salário, benefícios, dados de habilidades sobre funcionários, sistemas **enterprise resource planning** que registram detalhes sobre componentes de produtos, estoque de peças e bancos de dados de finanças que controlam o dinheiro, a contabilidade e as transações financeiras da organização.

Parallel Database

- Um banco de dados paralelo (**parallel database**) busca melhorar o desempenho por meio da paralelização para tarefas como carregamento de dados, construção de índices e avaliação de consultas.
- As principais arquiteturas de DBMS paralelos que são induzidas pela arquitetura de hardware subjacente são:
 - **Shared memory architecture**: onde vários processadores compartilham o espaço de memória principal, bem como outro armazenamento de dados.
 - **Shared disk architecture**: em que cada unidade de processamento (normalmente consistindo em vários processadores) tem sua própria memória principal, mas todas as unidades compartilham o outro armazenamento.
 - **Shared-nothing architecture**: onde cada unidade de processamento tem sua própria memória principal e outro armazenamento.

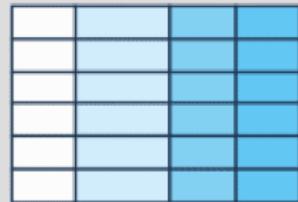
Outros Bancos de Dados

- Probabilistic databases empregam fuzzy logic para fazer inferências a partir de dados imprecisos.
- Real-time databases processam transações com rapidez suficiente para que o resultado volte e seja acionado imediatamente.
- Um spatial database pode armazenar os dados com recursos multidimensionais. As consultas sobre esses dados incluem consultas baseadas em localização, como "Onde fica o hotel mais próximo na minha área?".
- Um temporal database possui aspectos de tempo integrados, por exemplo, um modelo de dados temporais e uma versão temporal de SQL. Mais especificamente, os aspectos temporais geralmente incluem tempo válido e tempo de transação.
- Um terminology-oriented database se baseia em um banco de dados orientado a objetos, geralmente customizado para um campo específico.

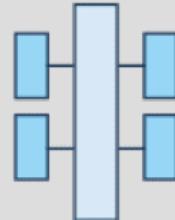
SQL vs NoSQL

SQL

Relational

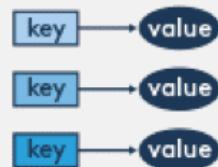


Analytical (OLAP)

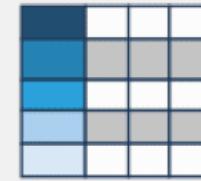


NoSQL

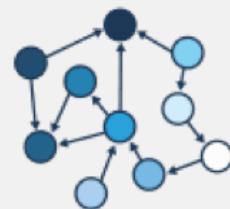
Key-Value



Column-Family



Graph

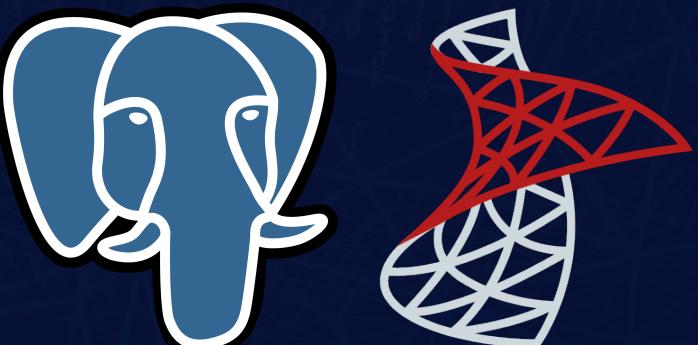


Document



Database Management System (DBMS)

- Connolly e Begg definem o **Database Management System** (DBMS) como um "sistema de software que permite aos usuários definir, criar, manter e controlar o acesso ao banco de dados".
- Exemplos de DBMS's incluem: MySQL, PostgreSQL, Microsoft SQL Server, banco de dados Oracle e Microsoft Access.



Database Management System (DBMS)

- A funcionalidade fornecida por um DBMS pode variar enormemente.
- A funcionalidade principal é o armazenamento, recuperação e atualização de dados. Codd propôs as seguintes funções e serviços que um DBMS de propósito geral totalmente desenvolvido deve fornecer:
 - Armazenamento, recuperação e atualização de dados;
 - Catálogo acessível ao usuário ou dicionário de dados que descreve os metadados;
 - Suporte para transações e simultaneidade;
 - Instalações para recuperar o banco de dados caso seja danificado;
 - Suporte para autorização de acesso e atualização de dados;
 - Suporte ao acesso de locais remotos;
 - Aplicar restrições para garantir que os dados no banco de dados obejam certas regras;

Database Management System (DBMS)

- Também é de se esperar que o DBMS forneça um conjunto de utilitários para os fins que possam ser necessários para administrar o banco de dados com eficácia, incluindo utilitários de importação, exportação, monitoramento, desfragmentação e análise.
- A parte central do DBMS que interage entre o banco de dados e a interface do aplicativo, às vezes chamada de **database engine**.
- Freqüentemente, os DBMS's terão parâmetros de configuração que podem ser ajustados estaticamente e dinamicamente, por exemplo, a quantidade máxima de memória principal em um servidor que o banco de dados pode usar. A tendência é minimizar a quantidade de configuração manual.

Database Management System (DBMS)

- Os primeiros DBMS's multiusuário normalmente permitiam que o aplicativo residisse no mesmo computador com acesso por meio de terminais ou software de emulação de terminal.
- A arquitetura **cliente-servidor** foi um desenvolvimento em que o aplicativo residia em um desktop cliente e o banco de dados em um servidor permitindo que o processamento fosse distribuído.
- Isso evoluiu para uma arquitetura **multitier** que incorpora servidores de aplicativos e servidores da web com a interface do usuário final por meio de um navegador da web com o banco de dados conectado apenas diretamente à camada adjacente.

Database Management System (DBMS)

- Um DBMS de uso geral fornecerá **application programming interfaces** (API) públicas e, opcionalmente, um processador para linguagens de banco de dados, como **SQL**, para permitir que os aplicativos sejam escritos para interagir com o banco de dados.
- Um DBMS de propósito especial pode usar uma API privada e ser especificamente personalizado e vinculado a um único aplicativo.
- Por exemplo, um sistema de e-mail executando muitas das funções de um DBMS de uso geral, como inserção de mensagem, exclusão de mensagem, tratamento de anexos, pesquisa de lista de bloqueio, associação de mensagens a um endereço de e-mail e assim por diante, no entanto, essas funções são limitadas ao que é necessário para lidar com email.

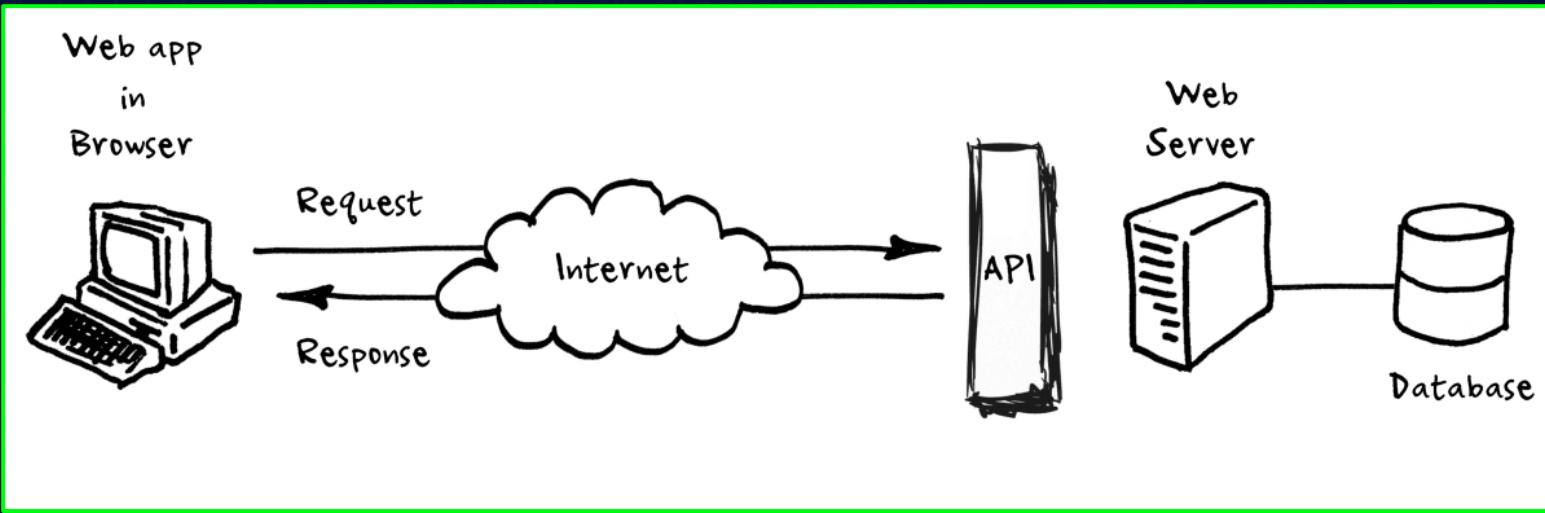
Aplicação

- A **interação externa** com o banco de dados será por meio de um programa aplicativo que faz interface com o DBMS.
- Isso pode variar de uma ferramenta de banco de dados que permite aos usuários executar consultas SQL textualmente ou graficamente, a um site que usa um banco de dados para armazenar e pesquisar informações.



Application Program Interface (API)

- Um programador codificará as interações com o banco de dados (às vezes referido como uma fonte de dados) por meio de uma **application program interface** (API) ou por meio de uma linguagem de banco de dados. A API ou idioma específico escolhido precisará ser suportado pelo DBMS, possivelmente indiretamente por meio de um pré-processador ou uma API de ponte. Algumas APIs pretendem ser independentes do banco de dados, sendo o **ODBC** um exemplo comumente conhecido. Outras APIs comuns incluem **JDBC** e **ADO.NET**.



Linguagens de Banco de Dados

- Linguagens de banco de dados são linguagens de propósito especial, que permitem uma ou mais das seguintes tarefas, às vezes distinguidas como sublinguagens:
 - **Data Control Language (DCL)**: controla o acesso aos dados;
 - **Data Definition Language (DDL)**: define os tipos de dados, como criação, alteração ou eliminação de tabelas e os relacionamentos entre eles;
 - **Data Manipulation Language (DML)**: executa tarefas como inserir, atualizar ou excluir ocorrências de dados;
 - **Data Query Language (DQL)**: permite buscar informações e computar informações derivadas.

Linguagens de Banco de Dados

- As linguagens de banco de dados são específicas para um determinado modelo de dados. Exemplos notáveis incluem:
 - **SQL** combina as funções de definição de dados, manipulação de dados e consulta em uma única linguagem. Foi uma das primeiras linguagens comerciais para o modelo relacional, embora se afaste em alguns aspectos do modelo relacional descrito por Codd (por exemplo, as linhas e colunas de uma tabela podem ser ordenadas). SQL se tornou um padrão do American National Standards Institute (ANSI) em 1986 e da International Organization for Standardization (ISO) em 1987.
 - **OQL** é um padrão de linguagem de modelo de objeto (do Object Data Management Group). Isso influenciou o design de algumas das linguagens de consulta mais recentes, como JDOQL e EJB QL.
 - **XQuery** é uma linguagem de consulta XML padrão implementada por sistemas de banco de dados XML, como MarkLogic e eXist, por bancos de dados relacionais com capacidade XML, como Oracle e DB2, e também por processadores XML em memória, como Saxon.
 - **SQL/XML** combina XQuery com SQL.

Linguagens de Banco de Dados

- Uma linguagem de banco de dados também pode **incorporar recursos** como:
 - Configuração específica de DBMS e gerenciamento de mecanismo de armazenamento;
 - Cálculos para modificar os resultados da consulta, como contagem, soma, média, ordenação, agrupamento e cross-referencing;
 - Aplicação de restrições (por exemplo, em um banco de dados automotivo, permitindo apenas um tipo de motor por carro);
 - Versão da application programming interface da linguagem de consulta, para conveniência do programador.

Armazenamento

- O armazenamento de banco de dados é o contêiner da materialização física de um banco de dados.
- Compreende o nível interno (físico) na arquitetura do banco de dados.
- Ele também contém todas as informações necessárias (por exemplo, **metadados**, "dados sobre os dados" e **estruturas de dados internas**) para reconstruir o nível conceitual e o nível externo do nível interno quando necessário.

Armazenamento

- O ato de colocar os dados em armazenamento permanente é geralmente responsabilidade do **database engine** conhecido como "storage engine".
- Embora normalmente acessado por um DBMS por meio do sistema operacional subjacente (e muitas vezes usando os **sistemas de arquivos** dos sistemas operacionais como intermediários para o layout de armazenamento), as propriedades de armazenamento e definição de configuração são extremamente importantes para a operação eficiente do DBMS e, portanto, são mantidos de perto por administradores de banco de dados.
- Um DBMS, enquanto em operação, sempre tem seu banco de dados residindo em vários tipos de armazenamento (por exemplo, memória e armazenamento externo).
- Os dados do banco de dados e as informações adicionais necessárias, possivelmente em grandes quantidades, são codificados em bits.

Armazenamento

- Os dados normalmente residem no armazenamento em estruturas que parecem completamente diferentes da aparência dos dados nos níveis conceitual e externo, mas de maneiras que tentam otimizar (o melhor possível) a reconstrução desses níveis quando necessário por usuários e programas, também quanto ao cálculo de tipos adicionais de informações necessárias dos dados (por exemplo, ao consultar o banco de dados).
- Alguns DBMS's suportam a especificação de qual **codificação de caracteres** foi usada para armazenar dados, portanto, várias codificações podem ser usadas no mesmo banco de dados.
- Várias estruturas de armazenamento de banco de dados de baixo nível são usadas pelo mecanismo de armazenamento para serializar o modelo de dados para que ele possa ser gravado no meio de escolha. Técnicas como indexação podem ser usadas para melhorar o desempenho. O armazenamento convencional é orientado por linha, mas também existem bancos de dados orientados por coluna e de correlação.

Armazenamento

- Freqüentemente, a **redundância de armazenamento** é empregada para aumentar o desempenho.
- Um exemplo comum é o armazenamento de **materialized views**, que consistem em visualizações externas ou resultados de consulta frequentemente necessários.
- O armazenamento de tais visualizações economiza o dispendioso cálculo delas cada vez que são necessárias.
- As desvantagens das **materialized views** são a sobrecarga incorrida ao atualizá-los para mantê-los sincronizados com seus dados de banco de dados atualizados originais e o custo de redundância de armazenamento.

Armazenamento

- Ocasionalmente, um banco de dados emprega **redundância de armazenamento** por **database objects replication** (com uma ou mais cópias) para aumentar a disponibilidade de dados (tanto para melhorar o desempenho de vários acessos simultâneos de usuário final a um mesmo objeto de banco de dados, quanto para fornecer resiliência em caso de falha parcial de um banco de dados distribuído).
- As atualizações de um objeto replicado precisam ser sincronizadas nas cópias do objeto.
- Em muitos casos, todo o banco de dados é replicado.

Segurança

- A **segurança do banco de dados** trata de todos os vários aspectos da proteção do conteúdo do banco de dados, seus proprietários e usuários.
- Ele varia de proteção de uso intencional de banco de dados não autorizado a acessos de banco de dados não intencionais por entidades não autorizadas (por exemplo, uma pessoa ou um programa de computador).

Segurança

- O **controle de acesso ao banco de dados** trata de controlar quem (uma pessoa ou um determinado programa de computador) tem permissão para acessar quais informações no banco de dados. As informações podem incluir objetos de banco de dados específicos (por exemplo, tipos de registro, registros específicos, estruturas de dados), certos cálculos sobre certos objetos (por exemplo, tipos de consulta ou consultas específicas), ou usando caminhos de acesso específicos para o primeiro (por exemplo, usando índices específicos ou outras estruturas de dados para acessar informações). Os controles de acesso ao banco de dados são definidos por pessoal autorizado especial (pelo proprietário do banco de dados) que usa interfaces do DBMS de segurança protegidas dedicadas.

Segurança

- A **segurança dos dados** impede que usuários não autorizados vejam ou atualizem o banco de dados. Usando senhas, os usuários têm acesso a todo o banco de dados ou subconjuntos chamados "subschemas". Por exemplo, um banco de dados de funcionários pode conter todos os dados sobre um funcionário individual, mas um grupo de usuários pode ser autorizado a visualizar apenas os dados da folha de pagamento, enquanto outros têm permissão para acessar apenas o histórico de trabalho e dados médicos. Se o DBMS fornece uma maneira de entrar e atualizar interativamente o banco de dados, bem como interrogá-lo, esse recurso permite o gerenciamento de bancos de dados pessoais.

Segurança

- A **segurança de dados** em geral lida com a proteção de pedaços específicos de dados, tanto fisicamente (ou seja, contra corrupção, destruição ou remoção), ou a interpretação deles, ou partes deles em informações significativas (por exemplo, por examinar as sequências de bits que eles compõem, concluindo números de cartão de crédito válidos específicos).
- Alterar e acessar os registros de log de quem acessou quais atributos, o que foi alterado e quando foi alterado. Os serviços de registro permitem uma **auditoria forense do banco de dados**, posteriormente, mantendo um registro das ocorrências e alterações de acesso.
- Às vezes, o código no nível do aplicativo é usado para registrar alterações, em vez de deixar isso para o banco de dados. O monitoramento pode ser configurado para tentar detectar violações de segurança.

Transações

- As **transações de banco de dados** podem ser usadas para introduzir algum nível de tolerância a falhas e integridade de dados após a recuperação de uma falha.
- Uma transação de banco de dados é uma unidade de trabalho, normalmente encapsulando uma série de operações em um banco de dados (por exemplo, ler um objeto de banco de dados, gravar, etc.), uma abstração suportada em banco de dados e também em outros sistemas. Cada transação tem limites bem definidos em termos de quais execuções de programa/código são incluídas nessa transação (determinado pelo programador da transação por meio de comandos de transação especiais).
- A sigla **ACID** descreve algumas propriedades ideais de uma transação de banco de dados: **atomicidade, consistência, isolamento** e **durabilidade**.

Migração

- Um banco de dados construído com um DBMS não é portátil para outro DBMS (ou seja, o outro DBMS não pode executá-lo). No entanto, em algumas situações, é desejável migrar um banco de dados de um DBMS para outro. Os motivos são principalmente econômicos (diferentes DBMS's podem ter diferentes custos totais de propriedade), funcionais e operacionais (diferentes DBMS's podem ter diferentes recursos). A migração envolve a transformação do banco de dados de um tipo de DBMS para outro. A transformação deve manter (se possível) o aplicativo relacionado ao banco de dados (ou seja, todos os programas de aplicativos relacionados) intacto. Assim, os níveis conceituais e de arquitetura externa do banco de dados devem ser mantidos na transformação. Pode ser desejável que também alguns aspectos do nível interno da arquitetura sejam mantidos. Uma migração de banco de dados complexa ou grande pode ser um projeto complicado e caro (único) por si só, que deve ser levado em consideração na decisão de migrar. Isso apesar do fato de que podem existir ferramentas para ajudar na migração entre DBMS's específicos. Normalmente, um fornecedor de DBMS fornece ferramentas para ajudar a importar bancos de dados de outros DBMS's populares.

Construindo, Mantendo e Ajustando

- Depois de projetar um banco de dados para um aplicativo, o próximo estágio é construir o banco de dados. Normalmente, um DBMS de uso geral apropriado pode ser selecionado para ser usado com esse propósito. Um DBMS fornece as **interfaces de usuário** necessárias para serem usadas pelos administradores de banco de dados para definir as estruturas de dados do aplicativo necessárias dentro do respectivo modelo de dados do DBMS. Outras interfaces de usuário são usadas para selecionar os parâmetros de DBMS necessários (como relacionados à segurança, parâmetros de alocação de armazenamento, etc.).

Construindo, Mantendo e Ajustando

- Quando o banco de dados está pronto (todas as suas estruturas de dados e outros componentes necessários são definidos), ele é normalmente preenchido com os dados do aplicativo inicial (inicialização do banco de dados, que normalmente é um projeto distinto; em muitos casos, usando interfaces DBMS especializadas que suportam a inserção em massa) antes de torná-lo operacional. Em alguns casos, o banco de dados se torna operacional enquanto está vazio de dados do aplicativo e os dados são acumulados durante sua operação.

Construindo, Mantendo e Ajustando

- Depois que o banco de dados é criado, inicializado e preenchido, ele precisa ser mantido. Vários parâmetros do banco de dados podem precisar de alteração e o banco de dados pode precisar ser ajustado (**tuning**) para melhor desempenho; as estruturas de dados do aplicativo podem ser alteradas ou adicionadas, novos programas de aplicativos relacionados podem ser escritos para adicionar à funcionalidade do aplicativo, etc.

Backup e Restauração

- Às vezes, é desejável trazer um banco de dados de volta a um estado anterior (por muitas razões, por exemplo, casos em que o banco de dados é encontrado corrompido devido a um erro de software, ou se ele foi atualizado com dados errados). Para conseguir isso, uma operação de **backup** é feita ocasionalmente ou continuamente, onde cada estado de banco de dados desejado (ou seja, os valores de seus dados e sua incorporação nas estruturas de dados do banco de dados) é mantido em **arquivos de backup dedicados** (existem muitas técnicas para fazer isso de forma eficaz). Quando é decidido por um administrador de banco de dados trazer o banco de dados de volta a este estado (por exemplo, especificando este estado por um ponto desejado no tempo em que o banco de dados estava neste estado), esses arquivos são usados para **restaurar** esse estado.

Recursos Diversos

- Outros recursos do DBMS podem incluir:
 - Logs do banco de dados: isso ajuda a manter um histórico das funções executadas.
 - Componente gráfico para a produção de gráficos e tabelas, especialmente em um sistema de data warehouse.
 - Otimizador de consulta: executa a otimização de consulta em cada consulta para escolher um plano de consulta eficiente (uma ordem parcial (árvore) de operações) a ser executado para calcular o resultado da consulta. Pode ser específico para um storage engine específico.
 - Ferramentas ou hooks para design de banco de dados, programação de aplicativo, manutenção de programa de aplicativo, análise e monitoramento de desempenho de banco de dados, monitoramento de configuração de banco de dados, configuração de hardware DBMS (um DBMS e banco de dados relacionado podem abranger computadores, redes e unidades de armazenamento) e mapeamento de banco de dados relacionado (especialmente para um DBMS distribuído), alocação de armazenamento e monitoramento de layout de banco de dados, migração de armazenamento, etc.

Design e Modelagem

- A primeira tarefa de um designer de banco de dados é produzir um modelo de dados conceitual que reflita a estrutura das informações a serem mantidas no banco de dados.
- Uma abordagem comum para isso é desenvolver um **modelo de entidade-relacionamento**, geralmente com o auxílio de ferramentas de desenho.
- Outra abordagem popular é a **Unified Modeling Language**.
- Um modelo de dados bem-sucedido refletirá com precisão o possível estado do mundo externo que está sendo modelado: por exemplo, se as pessoas puderem ter mais de um número de telefone, ele permitirá que essas informações sejam capturadas.

Design e Modelagem

- Projetar um bom modelo de dados conceituais requer um bom entendimento do **domínio do aplicativo**; normalmente envolve fazer perguntas profundas sobre aspectos de interesse para uma organização, como "um cliente também pode ser um fornecedor?", ou "se um produto é vendido com duas formas diferentes de embalagem, são eles o mesmo produto ou produtos diferentes?", ou "se um avião voa de Nova York a Dubai via Frankfurt, esse voo é um ou dois (ou talvez até três)?".
- As respostas a essas perguntas estabelecem definições da terminologia usada para entidades (clientes, produtos, voos, segmentos de voo) e seus relacionamentos e atributos.

Design e Modelagem

- A produção do modelo de dados conceitual às vezes envolve a entrada de **processos de negócios** ou a **análise do fluxo de trabalho** na organização.
- Isso pode ajudar a estabelecer quais informações são necessárias no banco de dados e quais podem ser deixadas de fora.
- Por exemplo, pode ajudar na hora de decidir se o banco de dados precisa conter dados históricos e também dados atuais.

Design e Modelagem

- Tendo produzido um modelo de dados conceitual com o qual os usuários estão satisfeitos, a próxima etapa é traduzi-lo em um **schema** que implementa as estruturas de dados relevantes dentro do banco de dados.
- Esse processo costuma ser chamado de design lógico de banco de dados e a saída é um **modelo de dados lógico** expresso na forma de um **schema**.
- Considerando que o modelo conceitual de dados é (pelo menos em teoria) independente da escolha da tecnologia de banco de dados, o modelo lógico de dados será expresso em termos de um modelo de banco de dados particular suportado pelo DBMS escolhido.

Design e Modelagem

- O modelo de banco de dados mais popular para bancos de dados de uso geral é o modelo relacional ou, mais precisamente, o modelo relacional representado pela linguagem **SQL**.
- O processo de criação de um design lógico de banco de dados usando este modelo usa uma abordagem metódica conhecida como **normalização**. O objetivo da normalização é garantir que cada "fato" elementar seja registrado apenas em um lugar, para que as inserções, atualizações e exclusões mantenham consistência automaticamente.

Design e Modelagem

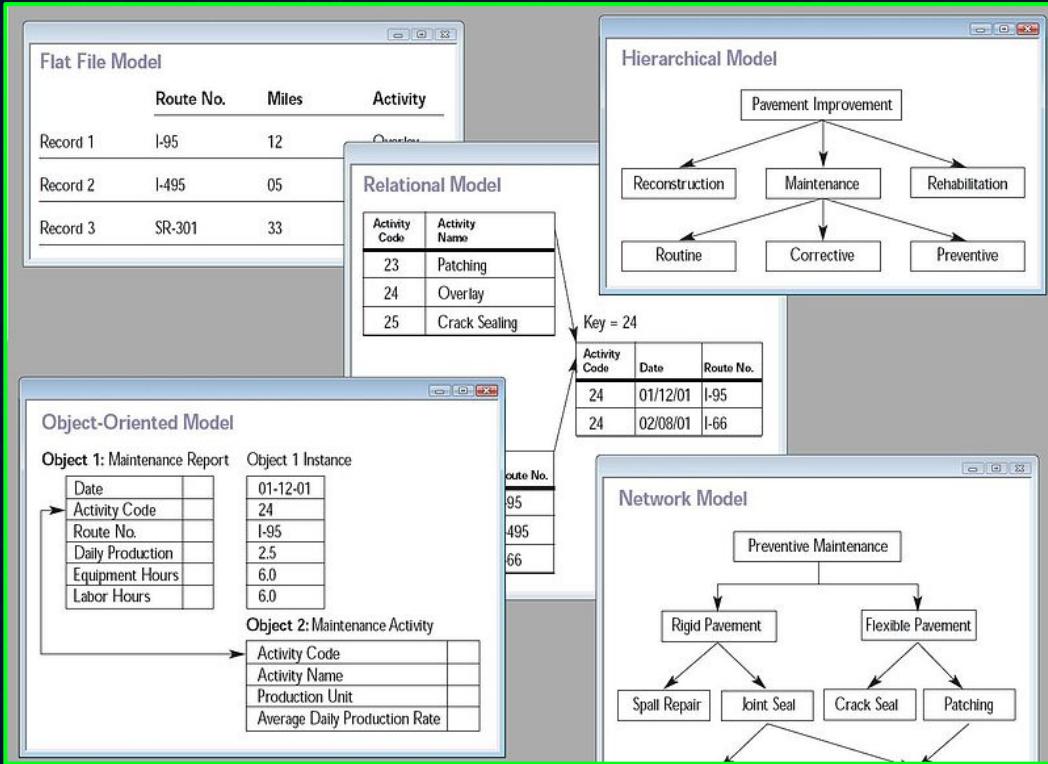
- O estágio final do projeto do banco de dados é tomar as decisões que afetam o desempenho, a escalabilidade, a recuperação, a segurança e assim por diante, que dependem do DBMS específico. Isso geralmente é chamado de design de banco de dados físico e a saída é o **modelo de dados físico**. Um objetivo principal durante esse estágio é a **independência de dados**, o que significa que as decisões tomadas para fins de otimização de desempenho devem ser invisíveis para os usuários finais e aplicativos. Existem dois tipos de independência de dados: Independência de dados físicos e independência de dados lógicos. O design físico é orientado principalmente pelos requisitos de desempenho e requer um bom conhecimento da carga de trabalho esperada e dos padrões de acesso, e um conhecimento profundo dos recursos oferecidos pelo DBMS escolhido.
- Outro aspecto do design de banco de dados físico é a segurança. Envolve a definição de controle de acesso a objetos de banco de dados, bem como a definição de níveis e métodos de segurança para os próprios dados.

Modelos

- Um **modelo de banco de dados** é um tipo de modelo de dados que determina a estrutura lógica de um banco de dados e, fundamentalmente, determina como os dados podem ser armazenados, organizados e manipulados.
- O exemplo mais popular de um modelo de banco de dados é o **modelo relacional** (ou a aproximação SQL de relacional), que usa um formato baseado em tabela.

Modelos

- A seguir temos uma imagem de cinco tipos de modelos de banco de dados:



Visualizações

- Um sistema de gerenciamento de banco de dados (DBMS) fornece três visualizações dos dados do banco de dados:
 1. O **nível externo** define como cada grupo de usuários finais vê a organização dos dados no banco de dados. Um único banco de dados pode ter qualquer número de visualizações no nível externo.
 2. O **nível conceitual** unifica as várias visualizações externas em uma visualização global compatível. Ele fornece a síntese de todas as visualizações externas. Está fora do escopo de vários usuários finais de banco de dados e é bastante interessante para desenvolvedores de aplicativos de banco de dados e administradores de banco de dados.

Visualizações

3. O **nível interno** (ou nível físico) é a organização interna dos dados dentro de um DBMS. Está preocupado com custo, desempenho, escalabilidade e outras questões operacionais.

Ele lida com o layout de armazenamento dos dados, usando estruturas de armazenamento como índices para aprimorar o desempenho.

Ocasionalmente, ele armazena dados de visualizações individuais (**materialized views**), calculadas a partir de dados genéricos, se houver justificativa de desempenho para tal redundância.

Ele equilibra todos os requisitos de desempenho das visualizações externas, possivelmente conflitantes, em uma tentativa de otimizar o desempenho geral em todas as atividades.

Visualizações

- Embora haja normalmente apenas uma visualização conceitual (ou lógica) e física (ou interna) dos dados, pode haver qualquer número de visualizações externas diferentes. Isso permite que os usuários vejam as informações do banco de dados de uma maneira mais relacionada aos negócios, em vez de de um ponto de vista técnico de processamento. Por exemplo, o departamento financeiro de uma empresa precisa dos detalhes de pagamento de todos os funcionários como parte das despesas da empresa, mas não precisa de detalhes sobre os funcionários que sejam do interesse do departamento de recursos humanos. Assim, departamentos diferentes precisam de visões diferentes do banco de dados da empresa.

Visualizações

- A **arquitetura de banco de dados de três níveis** está relacionada ao conceito de independência de dados, que foi uma das principais forças motrizes iniciais do modelo relacional. A ideia é que as alterações feitas em um determinado nível não afetam a visualização em um nível superior. Por exemplo, as mudanças no nível interno não afetam os programas aplicativos escritos usando interfaces de nível conceitual, o que reduz o impacto de fazer mudanças físicas para melhorar o desempenho.

Visualizações

- A **visão conceitual** fornece um nível de indireção entre **interno** e **externo**. Por um lado, ele fornece uma visão comum do banco de dados, independente de diferentes estruturas de visão externa e, por outro lado, abstrai os detalhes de como os dados são armazenados ou gerenciados (nível interno). Em princípio, todos os níveis, e até mesmo todas as visualizações externas, podem ser apresentados por um modelo de dados diferente.
- Na prática, geralmente um determinado DBMS usa o mesmo modelo de dados para os níveis externo e conceitual (por exemplo, modelo relacional). O nível interno, que está oculto dentro do DBMS e depende de sua implementação, requer um nível de detalhe diferente e usa seus próprios tipos de tipos de estrutura de dados.
- Separar os níveis externo, conceitual e interno foi uma característica importante das implementações do modelo de banco de dados relacional que dominam os bancos de dados do século XXI.

Pesquisa

- A tecnologia de banco de dados tem sido um tópico de pesquisa ativo desde 1960, tanto na academia quanto nos grupos de pesquisa e desenvolvimento de empresas (por exemplo, IBM Research). A atividade de pesquisa inclui teoria e desenvolvimento de protótipos.
- Tópicos de pesquisa notáveis incluíram: modelos, o conceito de transação atômica e técnicas de controle de simultaneidade relacionadas, linguagens de consulta e métodos de otimização de consulta, RAID e muito mais.
- A área de pesquisa de banco de dados tem vários periódicos acadêmicos dedicados (por exemplo: **ACM Transactions on Database Systems-TODS**, **Data and Knowledge Engineering-DKE**) e conferências anuais (por exemplo: **ACM SIGMOD**, **ACM PODS**, **VLDB**, **IEEE ICDE**).

Structured Query Language (SQL)

- SQL (**Structured Query Language**) é uma linguagem específica de domínio usada em programação e projetada para gerenciar dados mantidos em um sistema de gerenciamento de banco de dados relacional (RDBMS).
- É particularmente útil no tratamento de dados estruturados, ou seja, dados que incorporam relações entre entidades e variáveis.
- SQL é uma linguagem padrão para armazenar, manipular e recuperar dados em bancos de dados.
- SQL se tornou um padrão do American National Standards Institute (**ANSI**) em 1986 e da International Organization for Standardization (**ISO**) em 1987.

Structured Query Language (SQL)

- ❑ SQL pode executar consultas em um banco de dados.
- ❑ SQL pode recuperar dados de um banco de dados.
- ❑ SQL pode inserir registros em um banco de dados.
- ❑ SQL pode atualizar registros em um banco de dados.
- ❑ SQL pode excluir registros de um banco de dados.
- ❑ SQL pode criar novos bancos de dados.
- ❑ SQL pode criar novas tabelas em um banco de dados.

Structured Query Language (SQL)

- SQL pode criar **stored procedures** em um banco de dados.
- SQL pode criar visualizações em um banco de dados
- SQL pode definir permissões em tabelas, procedimentos e visualizações.
- Embora SQL seja um padrão ANSI/ISO, existem diferentes versões da linguagem SQL: No entanto, para serem compatíveis com o padrão ANSI, todos eles suportam pelo menos os comandos principais (como **SELECT**, **UPDATE**, **DELETE**, **INSERT**, **WHERE**) de maneira semelhante.

SQL em um Website

- Para construir um site que mostre dados de um banco de dados (SQL), você precisará de:
 - Um programa de banco de dados **RDBMS** (ou seja, MS Access, SQL Server, MySQL, etc).
 - Usar uma linguagem de script **server-side**, como PHP, Python, Java, JavaScript, etc.
 - Usar SQL para obter os dados que você deseja.
 - Usar **HTML/CSS** para estilizar a página e apresentar os dados.

RDBMS

- RDBMS significa **Relational Database Management System**.
- O RDBMS é a base do SQL e de todos os sistemas de banco de dados modernos, como MS SQL Server, IBM DB2, Oracle, MySQL e Microsoft Access.
- Os dados no RDBMS são armazenados em objetos de banco de dados chamados **tabelas**. Uma tabela é uma coleção de entradas de dados relacionadas e consiste em **colunas** e **linhas**.
- Cada tabela é dividida em entidades menores chamadas **campos**.
- Um **campo** é uma **coluna** em uma tabela projetada para manter informações específicas sobre cada registro na tabela.
- Um **registro**, também chamado de **linha**, é cada entrada individual que existe em uma tabela. Um registro é uma entidade horizontal em uma tabela.
- Uma coluna é uma entidade vertical em uma tabela que contém todas as informações associadas a um campo específico em uma tabela.

Entidades da Tabela

Tabela Pessoas

Coluna

SQL Tutorial

- Um dos bancos de dados mais populares é o MySQL.
- Para poder experimentar os exemplos de código neste tutorial, você deve ter o MySQL instalado em seu computador.
- Você pode baixar um banco de dados MySQL gratuito em:
- <https://www.mysql.com/downloads/>



SQL Tutorial

- Vamos iniciar o MySQL digitando o seguinte comando em nosso terminal:

```
mysql -u root -p
```

- Observe que neste exemplo eu estou usando o usuário **root**, digite a senha que você configurou durante a instalação. Com o banco de dados inicializado, podemos obter ajuda com o seguinte comando:

```
> help
```

- A instrução **CREATE DATABASE** é usada para criar um novo banco de dados SQL:

```
> CREATE DATABASE website;
```

SQL Tutorial

- Certifique-se de ter privilégios de administrador antes de criar qualquer banco de dados. Depois de criar um banco de dados, você pode verificá-lo na lista de bancos de dados com o seguinte comando:

```
> SHOW DATABASES;
```

- A instrução DROP DATABASE é usada para eliminar um banco de dados SQL existente.

```
> DROP DATABASE website;
```

- Tenha cuidado antes de descartar um banco de dados. A exclusão de um banco de dados resultará na perda de informações completas armazenadas no banco de dados!
- Certifique-se de ter privilégios de administrador antes de eliminar qualquer banco de dados. Depois que um banco de dados é eliminado, você pode verificá-lo na lista de bancos de dados com o seguinte comando SQL: **SHOW DATABASES;**

SQL Tutorial

- Digamos que você deseja gerar o backup de um único banco de dados, execute o seguinte comando, que irá gerar o backup do banco de dados **website** com estrutura e dados no arquivo **website.sql**.

```
mysqldump -u root -p website > /home/akira/Documentos/website.sql
```

- A instrução **CREATE TABLE** é usada para criar uma nova tabela em um banco de dados:

```
CREATE TABLE nome_tabela (  
    coluna1 datatype,  
    coluna2 datatype,  
    coluna3 datatype,  
);
```

Os parâmetros da **coluna** especificam os nomes das colunas da tabela.

O parâmetro **datatype** especifica o tipo de dados que a coluna pode conter (por exemplo: varchar, integer, date, etc).

SQL Tutorial

- Para criarmos uma tabela devemos selecionar o banco de dados que vamos trabalhar:

```
> USE website;
```

- O exemplo a seguir cria uma tabela chamada **Pessoas** que contém quatro colunas: **Nome**, **Idade**, **Endereco**, **Email**:

```
CREATE TABLE Pessoas (
    Nome varchar(255),
    Idade int,
    Endereco varchar(255),
    Email varchar(255)
);
```

SQL Tutorial

- A coluna **Idade** é do tipo **int** e conterá um número inteiro.
- As colunas **Nome**, **Endereco** e **Email** são do tipo **varchar** e conterão caracteres, e o comprimento máximo para esses campos é de 255 caracteres.
- A tabela vazia **Pessoas** ficará assim:

Nome	Idade	Endereco	Email

A tabela vazia **Pessoas** agora pode ser preenchida com dados com a instrução SQL **INSERT INTO**.

SQL Tutorial

- A instrução **DROP TABLE** é usada para eliminar uma tabela existente em um banco de dados:

```
> DROP TABLE Pessoas;
```

- Tenha cuidado antes de eliminar uma tabela. Excluir uma tabela resultará na perda de todas as informações armazenadas na tabela!
- A instrução **TRUNCATE TABLE** é usada para excluir os dados de uma tabela, mas não a própria tabela:

```
> TRUNCATE TABLE Pessoas;
```

SQL Tutorial

- A instrução **ALTER TABLE** é usada para adicionar, excluir ou modificar colunas em uma tabela existente.
- A instrução **ALTER TABLE** também é usada para adicionar e eliminar várias restrições em uma tabela existente.

- O seguinte SQL adiciona uma coluna **Sobrenome** à tabela **Pessoas**:

```
> ALTER TABLE Pessoas ADD Sobrenome varchar(255);
```

- Para excluir uma coluna em uma tabela, use a seguinte sintaxe

```
> ALTER TABLE Pessoas DROP COLUMN Sobrenome;
```

- Para alterar o tipo de dados de uma coluna em uma tabela, use a seguinte sintaxe:

```
> ALTER TABLE Pessoas MODIFY COLUMN Endereco varchar(350);
```

SQL Tutorial

- As restrições (**constraints**) podem ser especificadas quando a tabela é criada com a instrução **CREATE TABLE** ou depois que a tabela é criada com a instrução **ALTER TABLE**.

```
CREATE TABLE nome_tabela (
    coluna1 datatype restrição,
    coluna2 datatype restrição,
    coluna3 datatype restrição,
    coluna3 datatype restrição,
    coluna3 datatype restrição,
    ...
);
```

SQL Tutorial

- ❑ As restrições SQL são usadas para especificar regras para os dados em uma tabela.
- ❑ As restrições são usadas para limitar os tipos de dados que podem entrar em uma tabela.
- ❑ Isso garante a precisão e confiabilidade dos dados na tabela. Se houver qualquer violação entre a restrição e a ação de dados, a ação será abortada.
- ❑ As restrições podem ser no nível da coluna ou no nível da tabela. As restrições de nível de coluna se aplicam a uma coluna e as restrições de nível de tabela se aplicam a toda a tabela.

SQL Tutorial

- As seguintes restrições são comumente usadas em SQL:
 - NOT NULL - Garante que uma coluna não pode ter um valor NULL.
 - UNIQUE - Garante que todos os valores em uma coluna sejam diferentes.
 - PRIMARY KEY - Uma combinação de NOT NULL e UNIQUE. Identifica exclusivamente cada linha em uma tabela.
 - FOREIGN KEY - Impede ações que podem destruir links entre tabelas.
 - CHECK - Garante que os valores em uma coluna satisfaçam uma condição específica.
 - DEFAULT - Define um valor padrão para uma coluna se nenhum valor for especificado.
 - CREATE INDEX - Usado para criar e recuperar dados do banco de dados muito rapidamente.

SQL Tutorial

- Vamos agora recriar a tabela **Pessoas** com algumas restrições:

```
CREATE TABLE Pessoas (
    Id int PRIMARY KEY NOT NULL,
    Nome varchar(255) NOT NULL,
    Idade int NOT NULL,
    Endereco varchar(255),
    Email varchar(255) UNIQUE
);
```

- Para vermos as tabelas em nosso banco de dados, podemos usar o comando:

```
> SHOW TABLES;
```

SQL Tutorial

- O incremento automático permite que um número exclusivo seja gerado automaticamente quando um novo registro é inserido em uma tabela.
- Normalmente, esse é o campo de chave primária que desejamos que fosse criado automaticamente toda vez que um novo registro fosse inserido.
- A seguinte instrução SQL define a coluna `Id` como um campo de chave primária de incremento automático na tabela `Pessoas`:

```
CREATE TABLE Pessoas (
    Id int NOT NULL AUTO_INCREMENT,
    Nome varchar(255) NOT NULL,
    Idade int NOT NULL,
    Endereco varchar(255),
    Email varchar(255) UNIQUE,
    PRIMARY KEY (id)
);
```

SQL Tutorial

- Ao inserirmos um novo registro na tabela **Pessoas**, NÃO teremos que especificar um valor para a coluna **Id** (um valor único será adicionado automaticamente):

```
INSERT INTO Pessoas  
        (Nome, Idade, Endereco, Email)
```

```
VALUES  
        ('Gabriel', 30, 'Rua X', 'gabriel@gmail.com'),  
        ('Miguel', 22, 'Rua Z', 'miguel@gmail.com'),  
        ('Rafael', 25, 'Rua Y', 'rafael@yahoo.com'),  
        ('Maria', 20, 'Rua R', 'maria@yahoo.com'),  
        ('Julia', 18, 'Rua F', 'julia@google.com'),  
        ('Sofia', 18, 'Rua S', 'sofia@gmail.com');
```

SQL Tutorial

- Nossa tabela ficará então da seguinte forma:

Id	Nome	Idade	Endereço	Email
1	Gabriel	30	Rua X	gabriel@gmail.com
2	Miguel	22	Rua Z	miguel@gmail.com
3	Rafael	25	Rua Y	rafael@yahoo.com
4	Maria	20	Rua R	maria@yahoo.com
5	Julia	18	Rua F	julia@google.com
6	Sofia	18	Rua F	sofia@gmail.com

SQL Tutorial

- A instrução **SELECT** é usada para selecionar dados de um banco de dados.
- Os dados retornados são armazenados em uma tabela de resultados, chamada conjunto de resultados.
- A seguinte instrução SQL seleciona as colunas **Nome** e **Idade** da tabela **Pessoas**:

```
> SELECT Nome, Idade FROM Pessoas;
```

- A seguinte instrução SQL seleciona todas as colunas da tabela **Pessoas**:

```
> SELECT * FROM Pessoas;
```

- A instrução **SELECT DISTINCT** é usada para retornar apenas valores distintos (diferentes):

```
> SELECT DISTINCT Idade FROM Pessoas;
```

SQL Tutorial

- A cláusula **WHERE** é usada para filtrar registros.
- É usado para extrair apenas os registros que atendem a uma condição especificada.
- A cláusula WHERE não é usada apenas em instruções **SELECT**, ela também é usada em **UPDATE**, **DELETE**, etc.
- A seguinte instrução SQL seleciona todas as **Pessoas** com **Idade** superior a 22 anos:

```
> SELECT * FROM Pessoas Where Idade > 22;
```

- A seguinte instrução SQL seleciona todas as **Pessoas** que possuem o **Nome** 'Gabriel':

```
> SELECT * FROM Pessoas Where Nome = 'Gabriel';
```

- A seguinte instrução SQL seleciona todas as **Pessoas** cujo **Id** é menor ou igual a 3:

```
> SELECT * FROM Pessoas Where Id <= 3;
```

SQL Tutorial

- A cláusula **WHERE** pode ser combinada com os operadores **AND**, **OR** e **NOT**.
- Os operadores **AND** e **OR** são usados para filtrar registros com base em mais de uma condição:
 - O operador **AND** exibe um registro se todas as condições separadas por **AND** forem verdadeiras.
 - O operador **OR** exibe um registro se alguma das condições separadas por **OR** for verdadeira.
 - O operador **NOT** exibe um registro se as condições NÃO forem verdadeiras.
- Na instrução a seguir vamos selecionar todas as pessoas cujo **Nome** é 'Rafael' e **Idade** é superior a 22 anos:

```
> SELECT * FROM Pessoas Where Idade > 22 AND Nome = 'Rafael';
```

- A seguinte instrução seleciona todas as pessoas que não possuem 30 anos de idade:

```
> SELECT * FROM Pessoas Where NOT Idade = 30;
```

SQL Tutorial

- ❑ A palavra-chave **ORDER BY** é usada para ordenar o conjunto de resultados em ordem crescente ou decrescente.
- ❑ A palavra-chave **ORDER BY** ordena os registros em ordem crescente por padrão. Para ordenar os registros em ordem decrescente, use a palavra-chave **DESC**.
- ❑ A instrução a seguir selecionará todas as **Pessoas** ordenadas por **Idade**:

```
> SELECT * FROM Pessoas ORDER BY Idade;
```

- ❑ A instrução a seguir selecionará todas as **Pessoas** ordenadas por **Idade** de forma decrescente:

```
> SELECT * FROM Pessoas ORDER BY Idade DESC;
```

SQL Tutorial

- Um campo com valor **NULL** é um campo sem valor.
- Se um campo em uma tabela for opcional, é possível inserir um novo registro ou atualizar um registro sem adicionar um valor a este campo. Em seguida, o campo será salvo com um valor **NULL**.
- Observação: um valor **NULL** é diferente de um valor zero ou de um campo que contém espaços. Um campo com um valor **NULL** é aquele que foi deixado em branco durante a criação do registro!
- Não é possível testar os valores NULL com operadores de comparação.
- Teremos que usar os operadores **IS NULL** e **IS NOT NULL**.
- A seguir iremos selecionar todas as pessoas cuja **Idade** é **NULL**:

```
> SELECT * FROM Pessoas WHERE Idade IS NULL;
```

- Não teremos nenhum resultado, pois não temos nenhuma Idade **NULL**. Em seguida vamos selecionar todas as Pessoas suja idade não é **NULL**:

```
> SELECT * FROM Pessoas WHERE Idade IS NOT NULL;
```

SQL Tutorial

- A instrução **UPDATE** é usada para modificar os registros existentes em uma tabela.
- Observação: Tenha cuidado ao atualizar os registros em uma tabela! Observe a cláusula **WHERE** na instrução **UPDATE**. A cláusula **WHERE** especifica quais registros devem ser atualizados. Se você omitir a cláusula **WHERE**, todos os registros na tabela serão atualizados!
- A instrução a seguir irá alterar o **Nome** da pessoa cujo **Id** é 2:

```
> UPDATE Pessoas SET Nome = 'Samuel' WHERE Id = 2;
```
- Tenha cuidado ao atualizar os registros. Se você omitir a cláusula **WHERE**, TODOS os registros serão atualizados! Neste caso, todas as pessoas ficarão com o **Endereco** 'Rua'.

```
> UPDATE Pessoas SET Endereco = 'Rua';
```

SQL Tutorial

- ❑ A instrução **DELETE** é usada para excluir registros existentes em uma tabela.
- ❑ Observação: Tenha cuidado ao excluir registros em uma tabela! Observe a cláusula **WHERE** na instrução **DELETE**. A cláusula **WHERE** especifica quais registros devem ser excluídos. Se você omitir a cláusula **WHERE**, todos os registros na tabela serão excluídos!
- ❑ A seguinte instrução SQL exclui da tabela **Pessoas** a pessoa cujo **Id** é 2:

```
> DELETE FROM Pessoas WHERE id = 2;
```

- ❑ É possível excluir todas as linhas de uma tabela sem excluir a tabela. Isso significa que a estrutura, os atributos e os índices da tabela ficarão intactos. A seguinte instrução SQL exclui todas as linhas da tabela **Pessoas**, sem excluir a tabela:

```
> DELETE FROM Pessoas;
```

SQL Tutorial

- A palavra-chave **LIMIT** é usada para especificar o número de registros a serem retornados.
- A seguinte instrução SQL seleciona os três primeiros registros da tabela **Pessoas**:

```
> SELECT * FROM Pessoas LIMIT 3;
```

- A função **MIN()** retorna o menor valor da coluna selecionada.
- A função **MAX()** retorna o maior valor da coluna selecionada.
- As instruções a seguir selecionam a menor idade e a maior idade da tabela **Pessoas**:

```
> SELECT MIN(Idade) FROM Pessoas;
```

```
> SELECT MAX(Idade) FROM Pessoas;
```

SQL Tutorial

- A função **COUNT()** retorna o número de linhas que correspondem a um critério especificado.

- A instrução a seguir contará o número de nomes na tabela **Pessoas**:

```
> SELECT COUNT(Nome) FROM Pessoas;
```

- A função **AVG()** retorna o valor médio de uma coluna numérica.

- A instrução a seguir retornará a média de **Idade** da tabela **Pessoas**:

```
> SELECT AVG(Idade) FROM Pessoas;
```

- A função **SUM()** retorna a soma total de uma coluna numérica.

- A instrução a seguir retornará a soma de todas as idades da tabela **Pessoas**:

```
> SELECT SUM(Idade) FROM Pessoas;
```

SQL Tutorial

- O operador **LIKE** é usado em uma cláusula **WHERE** para pesquisar um padrão especificado em uma coluna.
- Existem dois **wildcards** geralmente usados em conjunto com o operador **LIKE**:
 - O sinal de porcentagem (%) representa zero, um ou vários caracteres.
 - O sinal de sublinhado (_) representa um único caractere.
- A seguinte instrução SQL seleciona todas as **Pessoas** com o **Nome** começando com "G":

```
> SELECT * FROM Pessoas WHERE Nome LIKE 'G%';
```
- A seguinte instrução SQL seleciona todas as **Pessoas** com o **Nome** terminando com "el":

```
> SELECT * FROM Pessoas WHERE Nome LIKE '%el';
```

SQL Tutorial

- A seguinte instrução SQL seleciona todas **Pessoas** com um **Nome** que têm "ri" em qualquer posição:

```
> SELECT * FROM Pessoas WHERE Nome LIKE '%ri%';
```

- A seguinte instrução SQL seleciona todas as **Pessoas** com o **Nome** que termina com "ia" e tem pelo menos 5 caracteres de comprimento:

```
> SELECT * FROM Pessoas WHERE Nome LIKE '____ia%';
```

- A seguinte instrução SQL seleciona todas **Pessoas** com um **Nome** que começa com "M" e termina com "l":

```
> SELECT * FROM Pessoas WHERE Nome LIKE 'M%l';
```

SQL Tutorial

□ O operador **IN** permite que você especifique vários valores em uma cláusula **WHERE**.

□ O operador **IN** é uma abreviação para várias condições **OR**.

□ A seguinte instrução SQL seleciona todas **Pessoas** com **Nome** “Gabriel” ou “Rafael”:

```
> SELECT * FROM Pessoas WHERE Nome IN ('Gabriel','Rafael');
```

□ A seguinte instrução SQL seleciona todas **Pessoas** cujo **Nome** não é “Gabriel” e “Rafael”:

```
> SELECT * FROM Pessoas WHERE Nome NOT IN ('Gabriel','Rafael');
```

□ A seguinte instrução SQL seleciona todas **Pessoas** cuja **Idade** não é 18, 20 e 22 anos:

```
> SELECT * FROM Pessoas WHERE Idade NOT IN (18,20,22);
```

SQL Tutorial

- O operador **BETWEEN** seleciona valores dentro de um determinado intervalo. Os valores podem ser números, texto ou datas.

- O operador **BETWEEN** é inclusivo: os valores inicial e final são incluídos.

- A seguinte instrução SQL seleciona todas **Pessoas** com **Idade** entre 18 e 25 anos:

```
> SELECT * FROM Pessoas WHERE Idade Between 18 AND 25;
```

- A seguinte instrução SQL seleciona todas **Pessoas** com **Idade** que não está entre 18 e 25 anos:

```
> SELECT * FROM Pessoas WHERE Idade NOT Between 18 AND 25;
```

- A seguinte instrução SQL seleciona todas **Pessoas** com **Id** entre 10 e 13:

```
> SELECT * FROM Pessoas WHERE Id BETWEEN 10 AND 13;
```

SQL Tutorial

- Os aliases SQL são usados para dar a uma tabela ou coluna de uma tabela um nome temporário.
- Os apelidos costumam ser usados para tornar os nomes das colunas mais legíveis.
- Um alias existe apenas para a duração da consulta.
- Um alias é criado com a palavra-chave AS.
- A seguinte instrução SQL cria dois aliases, um para a coluna Nome e outro para a coluna Endereco:

```
> SELECT Nome as N, Endereco as E FROM Pessoas;
```

- A seguinte instrução SQL cria um alias chamado "Info" que combina duas colunas (Endereco e Email):

```
> SELECT Nome, CONCAT(Endereco, ',', Email) AS Info FROM Pessoas;
```

SQL Tutorial

- Os comentários são usados para explicar seções de instruções SQL ou para evitar a execução de instruções SQL.
- Os comentários de uma única linha começam com `--`.
- Qualquer texto entre `--` e o final da linha será ignorado (não será executado).
- O exemplo a seguir usa um comentário de uma única linha como explicação:

```
> -- Seleciona todos os Nomes:  
> SELECT Nome FROM Pessoas;
```

- O exemplo a seguir usa um comentário de uma única linha para ignorar o final de uma linha:

```
> SELECT * FROM Pessoas -- WHERE Nome = 'Gabriel';
```

SQL Tutorial

- Os comentários de várias linhas começam com /* e terminam com */.
- Qualquer texto entre /* e */ será ignorado.
- O exemplo a seguir usa um comentário de várias linhas como explicação:

```
> /* Seleciona todas as colunas  
> e todos os registros  
> da tabela Pessoas:  
> SELECT * FROM Pessoas;
```

- O exemplo a seguir usa um comentário de várias linhas para ignorar instruções:

```
> /* SELECT * FROM Pessoas;  
> SELECT Nome FROM Pessoas; */
```

SQL Tutorial

- Para mais detalhes você pode visitar os seguintes tutoriais:



<https://www.sqltutorial.org/>

<https://www.w3schools.com/sql/>

NoSQL

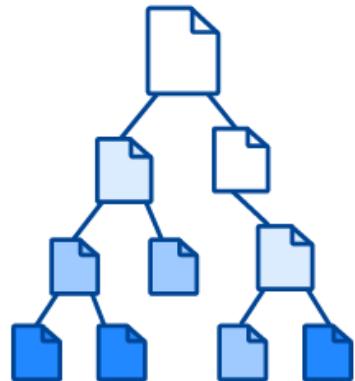
- Um banco de dados **NoSQL** (originalmente referindo-se a "não SQL" ou "não relacional") fornece um mecanismo para armazenamento e recuperação de dados que são modelados em meios diferentes das relações tabulares usadas em bancos de dados relacionais. Esses bancos de dados existem desde o final dos anos 1960, mas o nome "NoSQL" foi cunhado apenas no início do século 21, devido às necessidades das empresas da Web 2.0. Os bancos de dados NoSQL são cada vez mais usados em big data e aplicativos da web em tempo real. Os sistemas NoSQL às vezes também são chamados de "Não apenas SQL" para enfatizar que eles podem suportar linguagens de consulta semelhantes a SQL ou ficar ao lado de bancos de dados SQL em arquiteturas persistentes poliglotas.

NoSQL

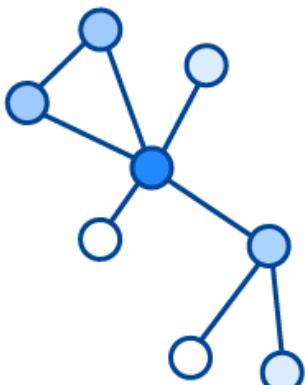
- As motivações para a abordagem NoSQL incluem: simplicidade de design, escalonamento "horizontal" mais simples para clusters de máquinas (o que é um problema para bancos de dados relacionais), controle mais preciso sobre a disponibilidade e limitação da object-relational impedance mismatch.
- As estruturas de dados usadas por bancos de dados NoSQL (por exemplo, **par de valores-chave**, **wide column**, **grafo** ou **documento**) são diferentes daquelas usadas por padrão em bancos de dados relacionais, tornando algumas operações mais rápidas em NoSQL.
- A adequação específica de um determinado banco de dados NoSQL depende do problema que ele deve resolver.
- Às vezes, as estruturas de dados usadas pelos bancos de dados NoSQL também são vistas como "mais flexíveis" do que as tabelas de banco de dados relacionais.

Estruturas de Dados NoSQL

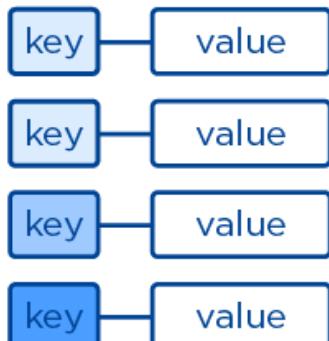
Document



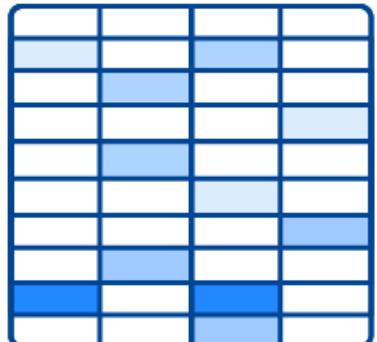
Graph



Key-Value



Wide-column



Tipos e Exemplos

- Existem várias maneiras de classificar bancos de dados NoSQL, com diferentes categorias e subcategorias, algumas das quais se sobrepõem. O que se segue é uma classificação básica por modelo de dados, com exemplos:
 - **Wide column**: Azure Cosmos DB, Accumulo, Cassandra, Scylla, Hbase.
 - **Document**: Azure Cosmos DB, Apache CouchDB, ArangoDB, BaseX, Clusterpoint, Couchbase, eXist-db, IBM Domino, MarkLogic, MongoDB, OrientDB, Qizx, RethinkDB.
 - **Key-value**: Azure Cosmos DB, Aerospike, Apache Ignite, ArangoDB, Berkeley DB, Couchbase, Dynamo, FoundationDB, InfinityDB, MemcacheDB, MUMPS, Oracle NoSQL Database, OrientDB, Redis, Riak, SciDB, SDBM/Flat File dbm, ZooKeeper.
 - **Graph**: Azure Cosmos DB, AllegroGraph, ArangoDB, InfiniteGraph, Apache Giraph, MarkLogic, Neo4J, AgensGraph, OrientDB, Virtuoso

Key-value Store

- Os armazenamentos de **valores-chave** usam o array associativo (também chamada de mapa ou dicionário) como seu modelo de dados fundamental.
- Neste modelo, os dados são representados como uma coleção de pares de valores-chave, de forma que cada chave possível apareça no máximo uma vez na coleção.
- O modelo de valor-chave é um dos modelos de dados não triviais mais simples, e modelos de dados mais ricos são frequentemente implementados como uma extensão dele.

Key		Value
City	→	Denver
State	→	Colorado
Country	→	USA

Key-value Store

- O modelo de **valor-chave** pode ser estendido a um modelo ordenado discretamente que mantém as chaves em ordem lexicográfica. Essa extensão é computacionalmente poderosa, pois pode recuperar com eficiência intervalos de chaves seletivas.
- Os armazenamentos de valores-chave podem usar modelos de consistência que variam de consistência eventual a **serializability**. Alguns bancos de dados oferecem suporte ao pedido de chaves. Existem várias implementações de hardware e alguns usuários armazenam dados na memória (RAM), enquanto outros em solid-state drives (SSD) ou rotating disks (também conhecido como hard disk drive (HDD)).

Document Store

- ❑ O conceito central de armazenamento de documentos é o de "**documento**". Embora os detalhes dessa definição difiram entre os bancos de dados orientados a documentos, todos eles assumem que os documentos encapsulam e codificam dados (ou informações) em alguns formatos ou codificações padrão.
- ❑ As codificações em uso incluem **XML**, **YAML** e **JSON** e formas binárias como **BSON**. Os documentos são endereçados no banco de dados por meio de uma chave única que representa aquele documento. Outra característica definidora de um banco de dados orientado a documentos é uma API ou linguagem de consulta para recuperar documentos com base em seu conteúdo.

Document Store

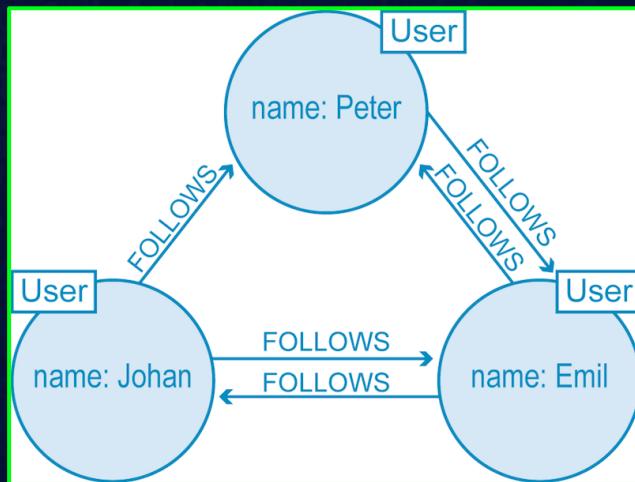
- Diferentes implementações oferecem diferentes maneiras de organizar e/ ou agrupar documentos:
 - Collections
 - Tags
 - Non-visible metadata
 - Directory hierarchies
- Em comparação com bancos de dados relacionais, as coleções podem ser consideradas análogas a tabelas e documentos análogos a registros.
- Mas eles são diferentes: cada registro em uma tabela tem a mesma sequência de campos, enquanto os documentos em uma coleção podem ter campos completamente diferentes.

Wide-column Store

- Um armazenamento de **colunas largas** (ou armazenamentos de registros extensíveis) é um tipo de banco de dados NoSQL.
- Ele usa tabelas, linhas e colunas, mas ao contrário de um banco de dados relacional, os nomes e o formato das colunas podem variar de linha para linha na mesma tabela.
- Um armazenamento de coluna larga pode ser interpretado como um armazenamento de valor-chave bidimensional.

Graph

- Os bancos de dados de **grafos** são projetados para dados cujas relações são bem representadas como um grafo que consiste em elementos conectados por um número finito de relações. Exemplos de dados incluem relações sociais, ligações de transporte público, mapas rodoviários, topologias de rede, etc.



Perfomance

- Ben Scofield classificou diferentes categorias de bancos de dados NoSQL da seguinte forma:

Data Model	Performance	Scalability	Flexibility	Complexity	Functionality
Key-Value	high	high	high	none	variable
Wide-Column	high	high	moderate	low	minimal
Document Oriented	high	variable	high	low	variable
Graph	variable	variable	high	high	graph theory
Relational	variable	variable	low	moderate	relational algebra

MongoDB

- ❑ MongoDB é um programa de banco de dados orientado a documentos cross-platform.
- ❑ Classificado como um programa de banco de dados NoSQL, o MongoDB usa documentos do tipo JSON com schemas opcionais.
- ❑ MongoDB é desenvolvido pela MongoDB Inc.



Redis

- Redis (Remote Dictionary Server) é um armazenamento de estrutura de dados in-memory, usado como um banco de dados de valor-chave distribuído in-memory, cache e message broker, com durabilidade opcional.
- O Redis oferece suporte a diferentes tipos de estruturas de dados abstratas, como strings, listas, mapas, conjuntos, conjuntos ordenados, HyperLogLogs, bitmaps, streams e spatial indices.



redis

Apache Cassandra

- O Apache Cassandra é um sistema de gerenciamento de banco de dados NoSQL gratuito e de código aberto, distribuído e de **wide-column store**, projetado para lidar com grandes quantidades de dados em muitos servidores de commodities, fornecendo alta disponibilidade sem um único ponto de falha. O Cassandra oferece suporte para clusters que abrangem vários datacenters, com asynchronous masterless replication, permitindo operações de baixa latência para todos os clientes.



Couchbase Server

- O **Couchbase Server**, originalmente conhecido como Membase, é um pacote de software de banco de dados orientado a documentos NoSQL de código aberto, distribuído (**shared-nothing architecture**) e otimizado para aplicativos interativos.
- Esses aplicativos podem servir a muitos usuários simultâneos, criando, armazenando, recuperando, agregando, manipulando e apresentando dados.
- Em suporte a esses tipos de necessidades de aplicativo, o Couchbase Server foi projetado para fornecer acesso a um valor-chave ou documento **JSON** fácil de dimensionar com baixa latência e alto rendimento sustentado. Ele foi projetado para ser agrupado a partir de uma única máquina para implementações em grande escala, abrangendo várias máquinas.



Big Data

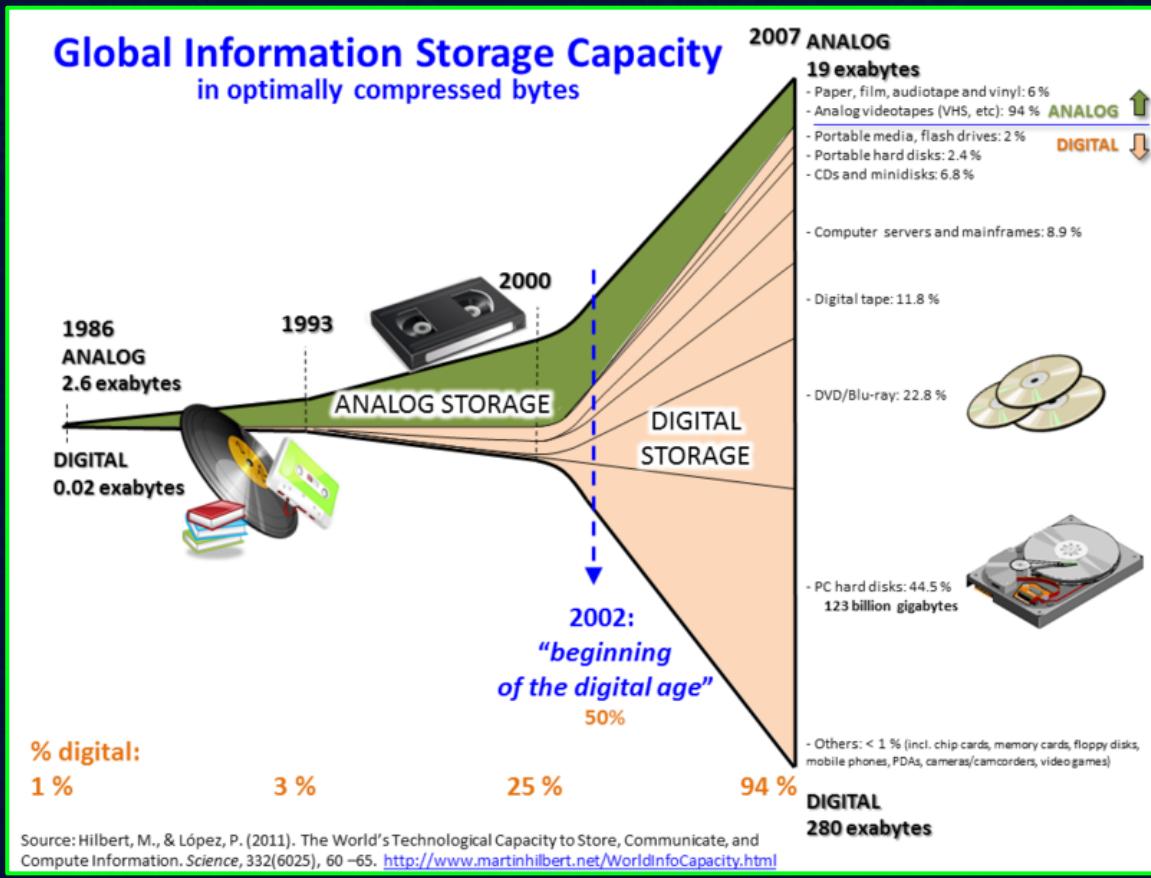
- Big data é um campo que trata de maneiras de analisar, extrair sistematicamente informações ou de outra forma lidar com conjuntos de dados que são muito grandes ou complexos para serem tratados por software processamento de dados tradicional.
- Dados com muitos campos (colunas) oferecem maior poder estatístico, enquanto dados com maior complexidade (mais atributos ou colunas) podem levar a uma maior taxa de descoberta falsa.
- Os desafios da análise de big data incluem captura de dados, armazenamento de dados, análise de dados, pesquisa, compartilhamento, transferência, visualização, consulta, atualização, privacidade de informações e fonte de dados.
- Big data foi originalmente associado a três conceitos principais: volume, variedade e velocidade.
- A análise de big data apresenta desafios na amostragem e, portanto, antes permitia apenas observações e amostragens.

Big Data

- O uso atual do termo big data tende a se referir ao uso de **predictive analytics, user behavior analytics** ou certos outros métodos de análise de dados avançados que extraem valor de big data.
- "Há poucas dúvidas de que as quantidades de dados agora disponíveis são realmente grandes, mas essa não é a característica mais relevante deste novo ecossistema de dados."
- A análise de conjuntos de dados pode encontrar novas correlações para "detectar tendências de negócios, prevenir doenças, combater o crime e assim por diante".

Big Data

Crescimento e digitalização da capacidade global de armazenamento de informações.



Big Data

- O tamanho e o número de conjuntos de dados disponíveis aumentaram rapidamente à medida que os dados são coletados por dispositivos como dispositivos móveis, dispositivos baratos e numerosos com sensores de informação na **Internet of Things**, antenas (sensoriamento remoto), logs de software, câmeras, microfones, radio-frequency identification (RFID) e wireless sensor networks.
- A capacidade tecnológica per capita mundial de armazenar informações praticamente dobrou a cada 40 meses desde a década de 1980; a partir de 2012, todos os dias 2.5 exabytes (2.5×2^{60} bytes) de dados são gerados.
- Com base em uma previsão de relatório do IDC, o volume de dados global foi previsto para crescer exponencialmente de 4.4 zetabytes para 44 zetabytes entre 2013 e 2020. Em 2025, o IDC prevê que haverá 163 zetabytes de dados.

Big Data

- Os **sistemas de gerenciamento de banco de dados relacional** e os pacotes de software estatístico de desktop usados para visualizar dados costumam ter dificuldade em processar e analisar big data.
- O processamento e a análise de big data podem exigir "software maciçamente paralelo rodando em dezenas, centenas ou mesmo milhares de servidores".
- O que se qualifica como "big data" varia de acordo com os recursos de quem o analisa e de suas ferramentas. Além disso, os recursos de expansão tornam o big data um alvo móvel. Para algumas organizações, enfrentar centenas de gigabytes de dados pela primeira vez pode desencadear a necessidade de reconsiderar as opções de gerenciamento de dados.

Big Data

- O termo big data está em uso desde a década de 1990, com alguns dando crédito a [John Mashey](#) por popularizar o termo.
- Big data geralmente inclui conjuntos de dados com tamanhos além da capacidade das ferramentas de software comumente usadas para capturar, curar, gerenciar e processar dados dentro de um tempo decorrido tolerável.
- A filosofia de big data engloba dados não estruturados, semiestruturados e estruturados, no entanto, o foco principal está nos dados não estruturados.
- Big data requer um conjunto de técnicas e tecnologias com novas formas de integração para revelar insights de conjuntos de dados que são diversos, complexos e em grande escala.
- Uma definição de 2018 afirma "Big data é onde as ferramentas de computação paralela são necessárias para lidar com os dados" e observa: "Isso representa uma mudança distinta e claramente definida na ciência da computação usada, por meio de teorias de programação paralela, e perdas de algumas das garantias e capacidades feitos pelo modelo relacional de Codd."

Características Big Data

- Big data pode ser descrito pelas seguintes características:
 - **Volume:** A quantidade de dados gerados e armazenados. O tamanho dos dados determina o valor e o insight potencial e se eles podem ser considerados big data ou não. O tamanho do big data geralmente é maior do que terabytes e petabytes.
 - **Variedade:** O tipo e a natureza dos dados. As tecnologias anteriores, como RDBMS's, eram capazes de lidar com dados estruturados com eficiência e eficácia. No entanto, a mudança no tipo e na natureza de estruturado para semiestruturado ou não estruturado desafiou as ferramentas e tecnologias existentes. As tecnologias de big data evoluíram com a intenção principal de capturar, armazenar e processar os dados semiestruturados e não estruturados (variedade) gerados com alta velocidade (velocity) e enormes em tamanho (volume).

Características Big Data

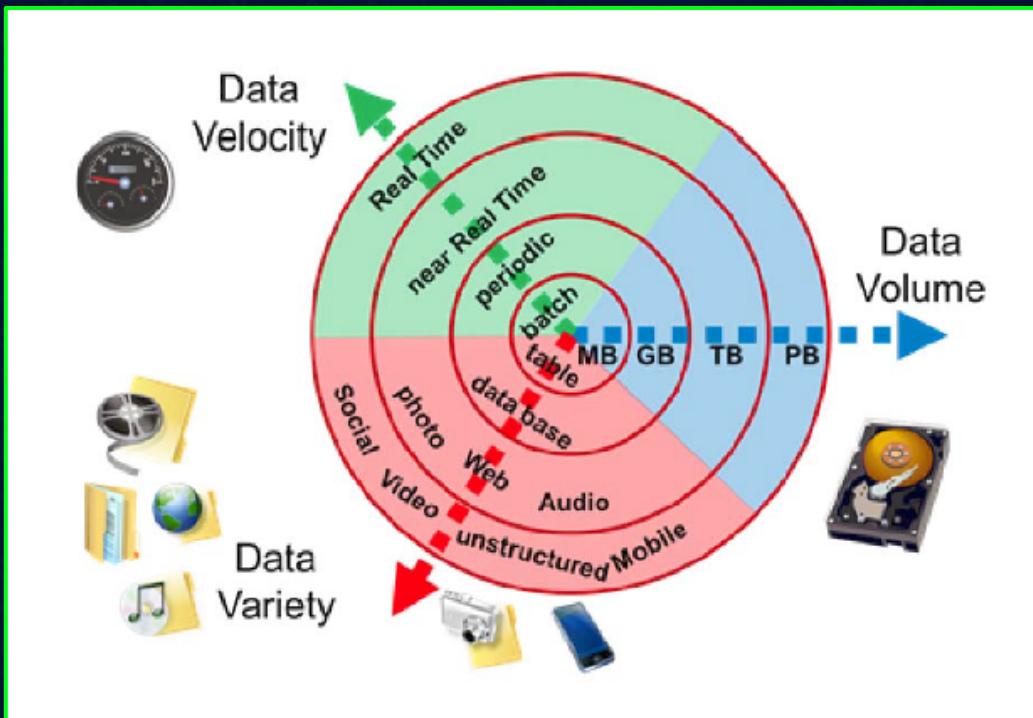
- Big data pode ser descrito pelas seguintes características:
 - **Velocidade**: A velocidade com que os dados são gerados e processados para atender às demandas e desafios que se encontram no caminho de crescimento e desenvolvimento. O big data geralmente está disponível em tempo real. Em comparação com pequenos dados, o big data é produzido de forma mais contínua. Dois tipos de velocidade relacionados ao big data são a frequência de geração e a frequência de manipulação, gravação e publicação.
 - **Veracidade**: A veracidade ou confiabilidade dos dados, que se refere à qualidade dos dados e ao valor dos dados. O big data não deve ser apenas grande, mas também confiável para agregar valor na análise. A qualidade dos dados capturados pode variar muito, afetando uma análise precisa.

Características Big Data

- Big data pode ser descrito pelas seguintes características:
 - **Valor:** O valor em informações que pode ser alcançado pelo processamento e análise de grandes conjuntos de dados. O valor também pode ser medido por uma avaliação das outras qualidades do big data. O valor também pode representar a lucratividade das informações recuperadas da análise de big data.
 - **Variabilidade:** A característica da mudança de formatos, estrutura ou fontes de big data. Big data pode incluir dados estruturados, não estruturados ou combinações de dados estruturados e não estruturados. A análise de big data pode integrar dados brutos de várias fontes. O processamento de dados brutos também pode envolver transformações de dados não estruturados em dados estruturados.

Características Big Data

- O gráfico a seguir apresenta o crescimento das principais características de big data (**volume**, **velocidade** e **variedade**):



Arquitetura Big Data

- Os repositórios de big data existem em muitas formas, geralmente construídos por empresas com necessidades especiais.
- A **Teradata Corporation** em 1984 comercializou o sistema DBC 1012 de processamento paralelo. Os sistemas Teradata foram os primeiros a armazenar e analisar 1 terabyte de dados em 1992.
- As unidades de disco rígido tinham 2.5 GB em 1991, portanto a definição de big data evolui continuamente de acordo com a **lei de Kryder**.
- A Teradata instalou o primeiro sistema baseado em RDBMS de classe petabyte em 2007. Em 2017, havia algumas dezenas de bancos de dados relacionais Teradata de classe petabyte instalados, o maior dos quais com mais de 50 PB. Os sistemas até 2008 eram dados relacionais 100% estruturados. Desde então, Teradata adicionou tipos de dados não estruturados, incluindo XML e JSON.

Arquitetura Big Data

- Em 2000, a Seisint Inc. (atualmente conhecida como **LexisNexis Risk Solutions**) desenvolveu uma plataforma distribuída baseada em C++ para processamento e consulta de dados conhecida como **HPCC Systems** platform.
- Este sistema particiona, distribui, armazena e entrega automaticamente dados estruturados, semiestruturados e não estruturados em diversos servidores de commodities.
- Os usuários podem escrever pipelines de processamento de dados e consultas em uma linguagem de programação de fluxo de dados declarativa chamada ECL.
- Os analistas de dados que trabalham em ECL não precisam definir data schemas antecipadamente e podem, em vez disso, focar no problema específico em questão, remodelando os dados da melhor maneira possível à medida que desenvolvem a solução.
- Em 2011, o **HPCC Systems** platform se tornou open-source sob a licença Apache v2.0.

Arquitetura Big Data

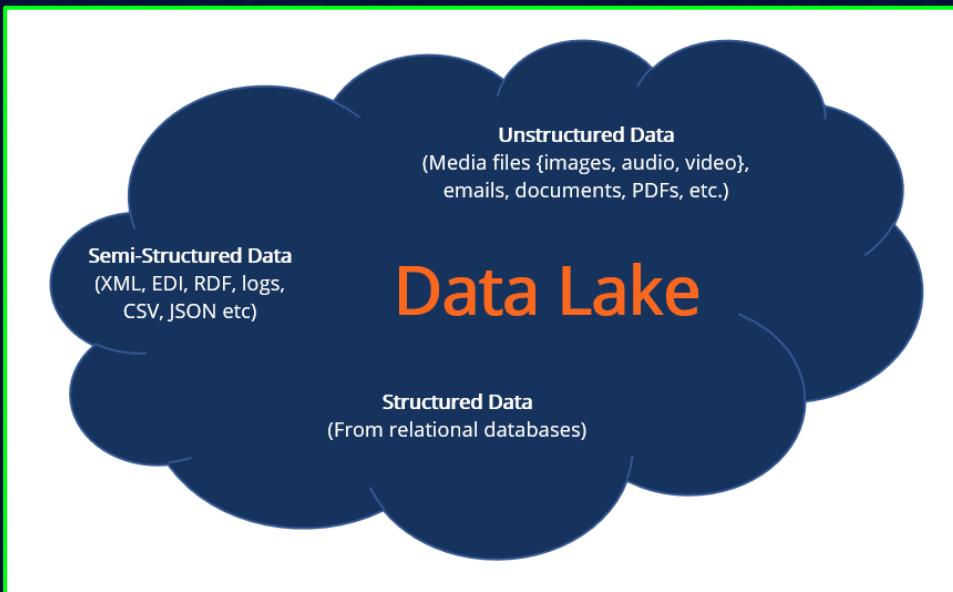
- O CERN e outros experimentos de física coletaram conjuntos de big data por muitas décadas, geralmente analisados por meio de computação de alto rendimento (high-throughput computing), em vez das map-reduce architectures normalmente representadas pelo movimento atual de "big data".
- Em 2004, o Google publicou um artigo sobre um processo chamado **MapReduce** que usa uma arquitetura semelhante.
- O conceito **MapReduce** fornece um modelo de processamento paralelo e uma implementação associada foi lançada para processar grandes quantidades de dados. Com o **MapReduce**, as consultas são divididas e distribuídas em nós paralelos e processadas em paralelo (a etapa "map").
- Os resultados são então reunidos e entregues (a etapa "reduce").

Arquitetura Big Data

- O framework **MapReduce** teve muito sucesso, então outros queriam replicar o algoritmo.
- Portanto, uma implementação do framework **MapReduce** foi adotada por um projeto de código aberto Apache denominado "**Hadoop**".
- O Apache Spark foi desenvolvido em 2012 em resposta às limitações do paradigma **MapReduce**, pois adiciona a capacidade de configurar muitas operações (não apenas mapear seguido de redução).
- Estudos em 2012 mostraram que uma arquitetura de múltiplas camadas era uma opção para resolver os problemas que o big data apresenta. Uma **arquitetura paralela distribuída** distribui dados em vários servidores; esses ambientes de execução paralela podem melhorar drasticamente as velocidades de processamento de dados. Esse tipo de arquitetura insere dados em um DBMS paralelo, que implementa o uso de frameworks **MapReduce** e **Hadoop**. Esse tipo de estrutura visa tornar o poder de processamento transparente para o usuário final, usando um servidor de aplicativos front-end.

Data Lake

- O **data lake** permite que uma organização mude seu foco do controle centralizado para um modelo compartilhado para responder às mudanças na dinâmica do gerenciamento de informações. Isso permite a segregação rápida de dados no **data lake**, reduzindo assim o tempo de sobrecarga.



Tecnologias Big Data

- Um relatório de 2011 do **McKinsey Global Institute** caracteriza os principais componentes e ecossistema de big data da seguinte forma:
 - Técnicas de análise de dados, como **A/B testing, machine learning e processamento de linguagem natural.**
 - Tecnologias de big data, como **business intelligence, computação em nuvem e bancos de dados.**
 - Visualização, como tabelas, gráficos e outras exibições dos dados.
- Big data multidimensional também pode ser representado como cubos de dados **OLAP** ou, matematicamente, **tensors.**

Tecnologias Big Data

- **Array Database Systems** foram desenvolvidos para fornecer armazenamento e suporte a consultas de alto nível neste tipo de dados.
- As tecnologias adicionais que estão sendo aplicadas a big data incluem efficient tensor-based computation, como multilinear subspace learning, bancos de dados de massively parallel-processing (MPP), aplicativos baseados em pesquisa, data mining, distributed file systems, distributed cache (por exemplo, burst buffer e Memcached), distributed databases, infraestrutura baseada em nuvem e HPC (aplicativos, recursos de armazenamento e computação) e a Internet.
- Embora muitas abordagens e tecnologias tenham sido desenvolvidas, ainda é difícil realizar o machine learning com big data.
- O programa de Análise de Dados Topológicos da **DARPA** busca a estrutura fundamental de conjuntos de dados massivos e em 2008 a tecnologia tornou-se pública com o lançamento de uma empresa chamada "**Ayasdi**".

Tecnologias Big Data

- Os profissionais de big data analytics processos geralmente são hostis ao armazenamento compartilhado mais lento, preferindo o direct-attached storage (**DAS**) em suas várias formas, desde solid state drive (**SSD**) a disco **SATA** de alta capacidade integrado em nós de processamento paralelo. A percepção das arquiteturas shared storage - storage area network (**SAN**) e network-attached storage (**NAS**) - é que elas são relativamente lentas, complexas e caras. Essas qualidades não são consistentes com sistemas big data analytics que prosperam através do desempenho do sistema, infraestrutura de commodity e baixo custo.

Aplicações Big Data

- O big data aumentou tanto a demanda de especialistas em gerenciamento de informações que Software AG, Oracle Corporation, IBM, Microsoft, SAP, EMC, HP e Dell gastaram mais de US\$15 bilhões em empresas de software especializadas em gerenciamento e análise de dados.
- Em 2010, esse setor valia mais de US\$100 bilhões e crescia quase 10% ao ano: cerca de duas vezes mais rápido que os negócios de software como um todo.
- As economias desenvolvidas usam cada vez mais tecnologias com uso intensivo de dados.
- Existem 4.6 bilhões de assinaturas de telefones celulares em todo o mundo, e entre 1 bilhão e 2 bilhões de pessoas acessando a Internet.
- Entre 1990 e 2005, mais de 1 bilhão de pessoas em todo o mundo entraram na classe média, o que significa que mais pessoas se tornaram mais alfabetizadas, o que por sua vez levou ao crescimento da informação.

Governo e Big Data

- A utilização e a adoção de big data em processos governamentais permitem eficiências em termos de custo, produtividade e inovação, mas não vem sem suas falhas.
- A análise de dados geralmente requer que várias partes do governo (central e local) trabalhem em colaboração e criem processos novos e inovadores para alcançar o resultado desejado.
- Uma organização governamental comum que faz uso de big data é a National Security Administration ([NSA](#)), que monitora as atividades da Internet constantemente em busca de padrões potenciais de atividades suspeitas ou ilegais que seu sistema pode detectar.
- [Civil registration and vital statistics](#) (CRVS) coleta todos os status dos certificados, desde o nascimento até o óbito. CRVS é uma fonte de big data para governos.

Governo e Big Data

- Dados espaciais - ou **big data from space** - é um termo usado para descrever as câmeras e as informações dos sensores coletadas por equipamentos de monitoramento espacial (satélites) e o processo de extrapolar padrões a partir deles usando software analítico.



Considerações Finais

“You can have data without information, but you cannot have information without data.” **Daniel Keys Moran**



Referências

- ❑ <https://en.wikipedia.org/wiki/Database>
- ❑ <https://en.wikipedia.org/wiki/MongoDB>
- ❑ <https://en.wikipedia.org/wiki/Redis>
- ❑ https://en.wikipedia.org/wiki/Couchbase_Server
- ❑ https://en.wikipedia.org/wiki/Apache_Cassandra
- ❑ https://en.wikipedia.org/wiki/Big_data
- ❑ <https://en.wikipedia.org/wiki/NoSQL>