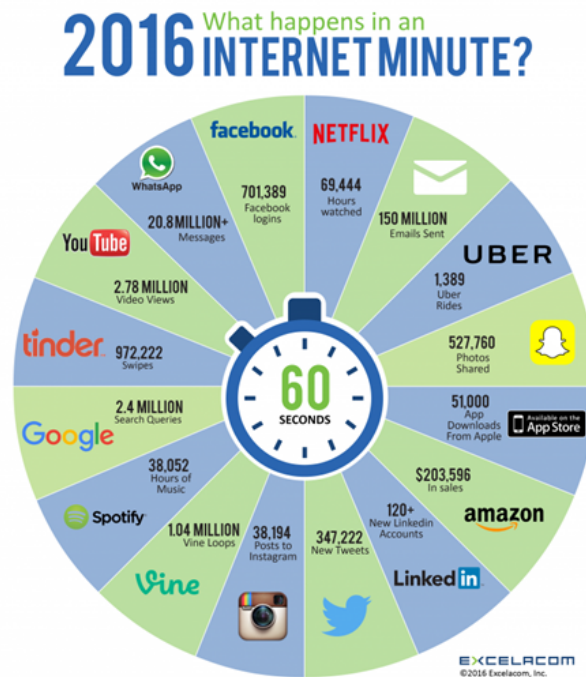


Conceitos de Data & Analytics I

1. Big Data

Em artigo de 2016, foram apresentados dados estatísticos espantosos para a época da quantidade de dados nas plataformas escaláveis que dominavam o mundo até então. O nome desse artigo era: “*What Happens in an Internet Minute in 2016?*” - em português, “*O que acontece em um minuto na internet em 2016*” [22]. Uma imagem deste artigo resume seu conteúdo. Segue abaixo:



Em 2019, no artigo [23] os números foram atualizados, conforme a imagem abaixo:

2019 This Is What Happens In An Internet Minute



Já em 2021, no artigo [24] os números foram novamente atualizados, e o resultado foi este:

2021 This Is What Happens In An Internet Minute



Podemos perceber que o número de dados aumenta significativamente, por exemplo, o WhatsApp saiu de cerca de 20 milhões para mais de 40 milhões (em 2019) e para quase 70 milhões em 2021.

De acordo com o glossário do **Gartner**, o termo *Big Data* significa um grande volume de dados, de alta velocidade e/ou de alta variedade que exigem formas inovadoras e econômicas de processamento de informações e que permitem uma melhor percepção, tomada de decisões e automação de processos. Em resumo, Big Data é um conjunto de dados maior e mais complexo, especialmente de novas fontes de dados [7]. Esses enormes volumes de dados podem ser usados para resolver problemas de negócios que as empresas não conseguiriam resolver antes.

O conceito de Big Data tomou maiores proporções com a definição de Doug Laney do *Gartner* com os 3 Vs:

- **Volume:** a quantidade de dados importa. Com big data, você terá que processar grandes volumes de dados estruturados ou não. Podem ser dados de valor desconhecido, como *feeds* de dados do *Twitter*, fluxos de cliques em uma página da web ou em um aplicativo para dispositivos móveis, ou ainda um equipamento habilitado para sensores. Para algumas empresas, isso pode utilizar dezenas de *terabytes* de dados. Para outras, podem ser centenas de *petabytes*.
- **Velocidade:** os dados são recebidos numa grande velocidade e precisam ser administrados em tempo hábil. Alguns produtos inteligentes habilitados para internet operam em tempo real ou quase em tempo real e exigem avaliação e ação em tempo real.
- **Variedade:** refere-se aos vários tipos de dados disponíveis. Tipos de dados tradicionais foram estruturados e se adequam perfeitamente a um banco de dados relacional. Com o aumento de big data, os dados vêm em novos tipos de dados não estruturados.

Tipos de dados não estruturados e semiestruturados, como texto, áudio e vídeo exigem um pré-processamento adicional para obter significado e dar suporte a *metadados*.

Com o passar dos anos surgiram mais 2 Vs:

- **Variabilidade:** os fluxos de dados podem ser altamente inconsistentes com picos periódicos. por exemplo, um assunto que vira tendência nas redes sociais. Todos os dias ocorrem picos de dados devido eventos particulares e estes podem ser mais difíceis de gerenciar e manipular.
- **Complexidade:** os dados têm origem de múltiplas tecnologias, o que torna difícil ligá-los, combiná-los, limpá-los e transformá-los entre sistemas. No entanto, é necessário conectar e correlacionar relações, hierarquias e ligações múltiplas, ou então pode-se rapidamente perder o controle sobre esses dados.

Neste artigo [7], a Oracle conta um pouco de como surgiu a necessidade de Big Data que foi transcrito para esse trabalho. Por volta de 2005, as pessoas começaram a perceber a quantidade de usuários de dados gerados pelo Facebook, YouTube e outros serviços on-line. O *Hadoop* (uma estrutura de código aberto criada especificamente para armazenar e analisar grandes conjuntos de dados) foi desenvolvido no mesmo ano. O NoSQL também começou a ganhar popularidade durante esse período.

O desenvolvimento de estruturas de código aberto, como o Hadoop, (e, mais recentemente, o Spark) foi essencial para o crescimento do big data, porque elas tornaram o trabalho com big data mais fácil e seu armazenamento mais barato. Nos anos seguintes, o volume de big data disparou. Usuários ainda estão gerando grandes quantidades de dados, mas não são somente humanos que estão fazendo isso. Com o advento da Internet das Coisas (*IoT*), mais objetos e dispositivos estão conectados à internet, reunindo dados sobre padrões de uso do cliente e desempenho do produto. O surgimento do *machine learning* produziu ainda mais dados.

Apesar da evolução do big data, sua utilidade ainda está no começo. A computação em nuvem expandiu ainda mais as possibilidades do big data. A nuvem oferece uma escalabilidade verdadeiramente elástica, na qual os desenvolvedores podem simplesmente criar clusters *ad-hoc* para testar um subconjunto de dados.

De acordo com a SAS [8], a importância do big data não gira em torno da quantidade de dados que a empresa tem, mas do que a empresa faz com eles. Pode-se obter dados de várias fontes e analisá-los para encontrar respostas que permitem: reduzir custos; economizar tempo; desenvolver novos produtos e otimizar ofertas; tomar decisões mais inteligentes. Quando combina-se big data com inteligência analítica, pode-se realizar tarefas corporativas como por exemplo:

- Determinar a causa de falhas, problemas e defeitos quase que em tempo real;
- Gerar cupons no ponto de venda com base nos hábitos de compra do cliente;
- Recalcular carteiras de riscos completas em minutos;
- Detectar comportamentos fraudulentos antes que eles afetem sua organização.

2. Ciência de Dados

Ciência de dados é uma área interdisciplinar voltada para o estudo e a análise de dados, estruturados ou não, que visa a extração de conhecimento ou insights para possíveis tomadas de decisão, de maneira similar à mineração de dados. Ciência de dados alia big data e machine learning, além de técnicas de outras áreas interdisciplinares como estatística, economia, engenharia e outros subcampos da computação como: banco de dados e análise de agrupamentos (cluster analysis). A ciência de dados é um campo que já existe há 30 anos, porém ganhou mais destaque nos últimos anos devido a alguns fatores como: o surgimento e popularização do Big Data e o desenvolvimento de áreas como o machine learning. A ciência de dados pode, por exemplo, transformar essa grande quantidade de dados brutos em insights de negócios, e com isso, auxiliar empresas em tomadas de decisões para atingir melhores resultados [9].

De acordo com o glossário do Gartner, a função de cientista de dados é crítica para organizações que buscam extrair insights de informações em iniciativas de “big data” e requer uma ampla combinação de habilidades que podem ser melhor atendidas numa equipe, por exemplo: Colaboração e trabalho em equipe são necessários para trabalhar com stakeholders de negócios visando entender os problemas de negócios. A figura abaixo apresenta a relação entre as áreas envolvidas.



Figura - Relação de perfis na ciência de dados

3.Diferença entre papéis envolvidos em Big Data

Cientista de Dados

O Cientista de Dados é o profissional com formação básica bem sólida com conhecimento nas áreas de ciência da computação, estatística, modelagem, análises, matemática, machine learning, banco de dados, business intelligence, ciência social ou física, entre outros. A quantidade de conhecimentos que este profissional deve adquirir é muito vasta e em diversas áreas das exatas e algumas humanas. Por este motivo decidiu-se por criar perfis de Cientista de Dados para as áreas de maior interesse do profissional, removendo boa parte da diversidade exigida e facilitando a estruturação de trilhas de estudos. Baseado em estudos, definiu-se 3 tipos de profissionais:

- **Desenvolvedor:** profissionais focados em problemas técnicos e no gerenciamento dos dados. Claramente escrevendo código, provavelmente código em produção, no seu dia-a-dia. São pessoas com muitas habilidades técnicas, tendo graduação em Ciência da Computação, Engenharia da Computação e afins.
- **Estatístico:** profissionais focados em problemas estatísticos. Normalmente começaram em projetos de pesquisa acadêmicos em ciência física ou social. São pessoas que foram ou são Estatísticos e que normalmente passaram muito tempo em Universidades fazendo mestrado e/ou doutorado.
- **Businessperson:** profissionais focados na empresa e em como projetos de dados podem gerar lucros. São pessoas que se classificam como líderes ou empreendedores e possuem certas habilidades técnicas, possivelmente tendo graduação em exatas. São profissionais generalistas, que atuam conjunto com todos os outros perfis, com habilidades de comunicação e integração entres os perfis e também entre os Engenheiros de Dados, Arquitetos de Soluções e Desenvolvedores.

Arquiteto de Soluções para Dados

De acordo com o Open Group, o arquiteto de soluções tem por objetivo tornar o negócio simples - "making simple to business". A Arquitetura da Solução é uma prática de definição e descrição de uma arquitetura de um sistema no contexto de uma solução específica e pode englobar descrição do sistema inteiro ou só de parte do sistema.

Para um Arquiteto de Soluções focado em Dados, é necessário o conhecimento amplo de diversos conceitos como por exemplo Business Intelligence, Data Warehouse, Data Lake, Big Data, Ciência de Dados, em linguagens de programação como SQL, Python, Java, em ferramentas como por exemplo as presentes no Ecossistema Hadoop e em técnicas englobando dados como por exemplo Batch, ETL, Streaming, CDC, SCD entre outras.

Como a arquitetura deve ser desenhada para um cliente, precisa-se considerar a estrutura atual do cliente e entender o objetivo futuro com a solução desejada. Por este motivo precisa-se conhecer o que está disponível de ser utilizado para a necessidade do cliente como estado atual da solução na infraestrutura do cliente e possíveis evoluções como infraestruturas em nuvens como AWS, Microsoft Azure e Google.

Engenheiro de Dados

O Engenheiro de Dados é um profissional com amplo e profundo conhecimento sobre a implantação de componentes tecnológicos. Entre as tarefas esperadas estão a criação de metodologias de desenvolvimento específicas para clientes contemplando todo o ciclo de vida dos dados do Data Lake em conjunto com as decisões arquiteturais definidas no projeto, ser o responsável por buscar ferramentas e soluções para deploy e sustentação de Data Lakes, conhecer e sugerir melhores práticas e trabalhar em conjunto com o Arquiteto de Soluções em tempo de definição e em conjunto com os desenvolvedores em tempo de implementação.

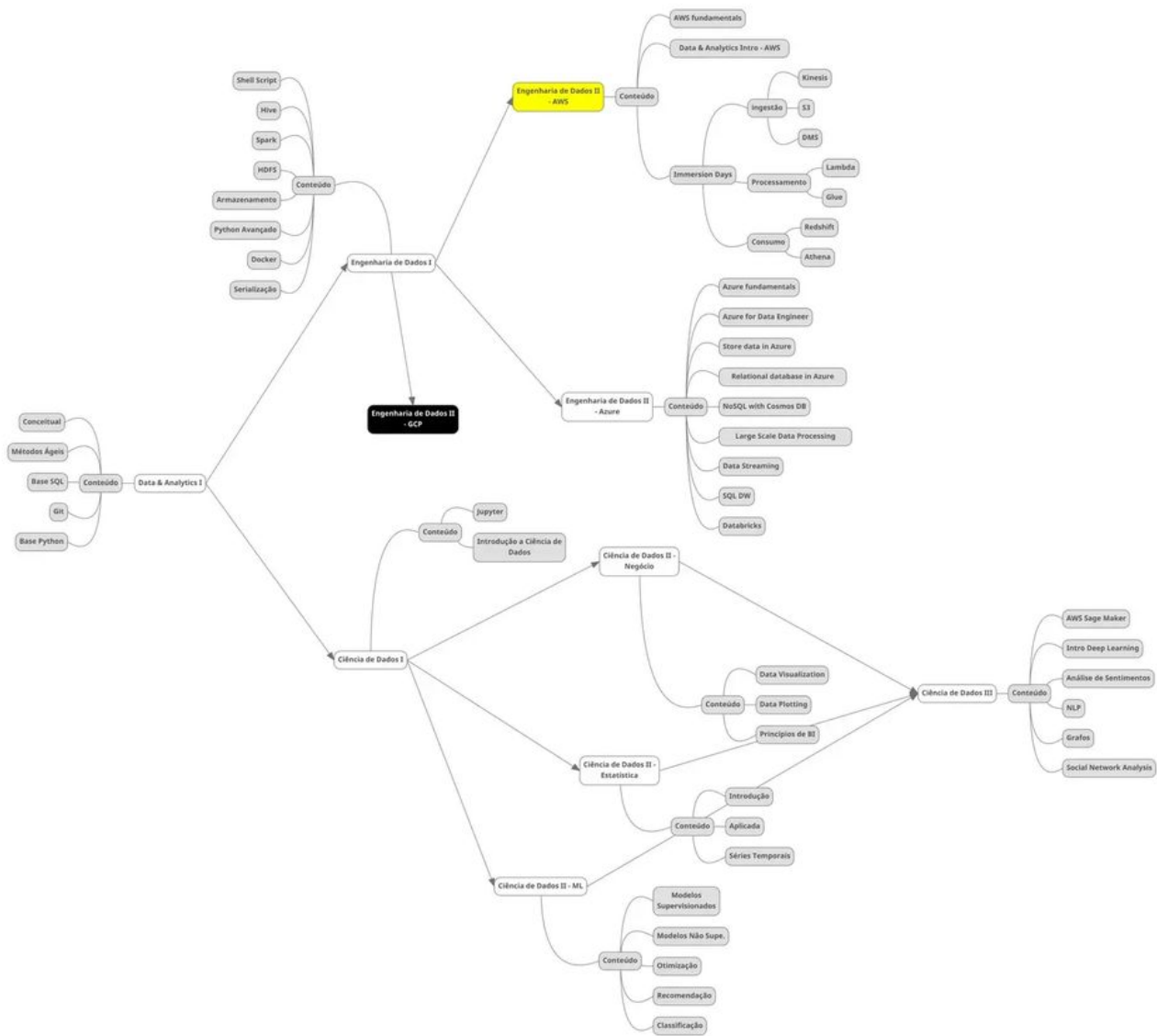
Desenvolvedores

Profissional com habilidades de codificação de sistemas que irá trabalhar em conjunto com o Engenheiro de dados e com o Arquiteto de Soluções. Junto ao arquiteto com a visão do todo, trazendo necessidades e angústias do cliente e sugerindo mudanças no desenho da solução. Junto ao Engenheiro de Dados na implementação da solução e apresentando sugestões de melhoria de processos previamente definidos.

O desenvolvedor deve possuir os conhecimentos comuns de soluções de dados que todos os profissionais desta área precisam. No contexto utilizado pela Compasso, os desenvolvedores precisam dominar as linguagens de programação Python e SQL, as ferramentas Apache Spark, Hive e Elasticsearch e os serviços Serverless dos provedores de nuvem.

Trilhas na Compasso:

As trilhas foram divididas de acordo com os perfis. Há uma trilha que deve ser de conhecimento de todos os perfis, chamada de trilha Conceito. Abaixo apresentam-se os detalhes para as trilhas.



4. Tipos de Dados (estruturado, semi e não estruturado)

Literal [↗](#)

Você sabe o que é um valor literal? É um valor isolado, seja ele numérico, textual ou alfanumérico (letras, símbolos e números). Abaixo tem-se uma lista de alguns exemplos de valores literais entre aspas e o que significam após um símbolo hífen (-):

1. "A" – uma letra ou um caractere.

"56" – um número

"meu papel hoje será" – um texto

"Leandro" – um texto

"-62,13" – um número

Um formulário de cadastro, por exemplo, reúne vários campos que o usuário preencherá com dados literais: nome, idade, e-mail, endereço, telefone etc. Depois que os dados forem cadastrados, eles serão armazenados de forma inter-relacionada, e assim os dados brutos transformam-se em dados processados. Por exemplo, **informações** de uma pessoa.

Quando os **dados são processados**, relacionados/agrupado ou interpretados, adicionando-se sobre eles contexto e importância, eles passam a representar **informações**.

A palavra Leandro para os computadores é apenas uma sequência de letras quando interpretada como dado bruto, mas é um nome quando considerada como informação ou dado processado.

Assim, dados e informações são semelhantes quando queremos comunicar valores que trafegam em um canal de comunicação ou que estão armazenados em um dispositivo. Por exemplo, na frase: “meu HD queimou e perdi todos os dados que estavam nele”. Se o termo dados fosse substituído por informações teria o mesmo significado.

Porém, são diferentes quando queremos falar de diferentes estágios da obtenção e interpretação de dados. “Consegui recuperar vários dados brutos, mas não todas as informações”.

Dado

O termo no singular ou no plural de forma isolada pode deixar dúvidas, por isso neste texto prefere-se escrever **dados digitais** para deixar bem claro que não há relação com dados usados em jogos de tabuleiro. Esse dado, em inglês, chama-se *dice*.

Estes dados digitais, em inglês, chamam-se **data** (não confundir com data em português que, em inglês, é *date*).

O dado representa a menor partícula de uma informação, assim como um átomo representa a menor partícula de uma matéria. Portanto, um dado pode ser simplesmente um caractere, um texto, um número ou uma combinação de ambos, mas pode ser mais do que isso também.

Pense um aplicativo de conversa como o Whats App. Quais tipos de dados você pode enviar para um destinatário?

- Um texto com ou sem símbolos (emojis)
- Um áudio
- Uma foto tirada na hora
- Uma imagem da galeria (que pode ser um gráfico, uma ilustração ou mesmo uma foto)
- Um vídeo (feito na hora ou armazenado)
- Sua localização
- Um contato
- Um arquivo

Todos são exemplos de dados, por isso eles podem ser simples valores literais ou agrupados em arquivos (de imagem, de vídeo, de áudio etc).

Informação

Será que o simples processamento de dados caracteriza uma informação?

Por exemplo, se você receber um texto como “xk!f@f#” fará algum sentido? Obviamente não, pois tem-se apenas um texto sem sentido. Mas o dado foi processado, no sentido de ser convertido em formato adequado para trafegar de um dispositivo para o outro.

Agora, se vier a mensagem recebida for algo como “cheguei e estou lhe esperando aqui na frente do seu trabalho”? Aí é uma informação, pois é um dado textual provido de sentido, no caso, de comunicar uma mensagem em português ao destinatário.

O mesmo raciocínio pode ser feito nos demais itens. De um áudio cheio de ruídos ou sem som não se extrai informação. Uma foto totalmente borrada e desfocada também não. O oposto das situações é claramente uma informação e assim por diante com os outros tipos de dados citados.

Dessa forma, em algumas situações, os dados podem ser processados, e mesmo assim não resultarem em informações. Em outras situações, o processamento faz toda diferença. Um código de barra é um número e, portanto, um dado bruto. Quando ele é adotado como forma de controle de produtos, como em um supermercado, ele representa uma informação: o produto. Caso não exista nenhum produto associado ao código, ainda assim ele representa um significado: “produto não cadastrado”.

Portanto **informação** requer um processamento ou uma interpretação (ou análise) de dados brutos. No caso do código de barra, é um processamento de fato, uma consulta ao banco de dados com a entrada sendo o código cuja saída são os dados do produto. No caso dos dados do WhatsApp, os dados dependem de interpretação e análise para serem informações.

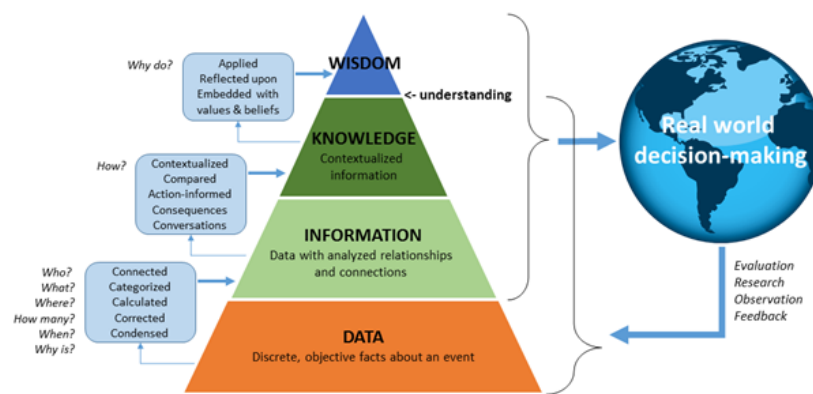
Dados versus Informações [↗](#)

Se você trabalha com dados e informações ou pensa em trabalhar, como ser Cientista de Dados ou trabalhar nas áreas de Mineração de dados, *Business Intelligence*, *Machine Learning* e outras áreas ascendentes, saber essa informação é importante. Essa diferença é, na verdade, só o começo, pois subindo na escala de transformação temos ainda o conhecimento e a inteligência (ou sabedoria).

Conhecimento e Inteligência [↗](#)

Os dados correspondem ao nível mais baixo, depois de processados ou interpretados tem-se a informação e, então, quando ela é contextualizada, chega-se ao **conhecimento**. Outras características também mudam conforme navega-se na pirâmide abaixo. Os dados são fáceis de serem estruturados e coletados por equipamentos, de forma que são normalmente explícitos, quantificados e transferíveis. As informações requerem uma estruturação mais complexa, pois são dados com significados e agrupados. Já os conhecimentos são implícitos, de difícil estruturação e transmissão.

Portanto, conhecimento é uma informação contextualizada, mas normalmente também corresponde a uma informação relevante e valiosa. Existe ainda mais um nível, quando o conhecimento é transformado em **inteligência** pelo processo do aprendizado ou aplicação e avaliação dos conhecimentos, servindo como importante ferramenta para tomada de decisões.



Dados Estruturados e Não Estruturados [↗](#)

Existem 3 formas de classificar os dados de acordo com sua estrutura:

- Dados estruturados
- Dados semiestruturados
- Dados não estruturados



A imagem acima mostra uma diferença visual, sugerindo que os dados estruturados são organizados em um padrão fixo, enquanto os não estruturados seguem uma estrutura rígida. Os semiestruturados ficam entre os extremos: não são estruturados de forma rígida, mas também não são totalmente desestruturados.

Dados Estruturados

Dados estruturados são aqueles organizados e representados com uma **estrutura rígida**, a qual foi previamente planejada para armazená-los.

De forma prática, pense em um formulário de cadastro com os campos: **nome**, **e-mail**, **idade** e uma **pergunta** que admite como resposta **sim** ou **não**. O campo **nome** será um **texto**, uma sequência de letras com ou sem a presença de espaços em branco, que terá um limite máximo e não poderá conter números ou símbolos. O campo **e-mail** também terá o padrão **textual**, mas formado por uma sequência de caracteres (e não só letras, pois admitirá números e alguns símbolos) e terá que ter obrigatoriamente uma arroba. **Idade** é um campo que aceita apenas um **número** inteiro positivo, enquanto o campo referente a **pergunta** armazena um **valor binário** (pense em 1 bit, que pode ser 0 ou 1. Valor 0 para não, 1 para sim). Assim, cada campo possui um padrão bem definido, que representa uma estrutura rígida e um formato previamente projetado para ele. Pode-se destacar que:

- Os dados de um mesmo cadastro estão relacionados (dizem respeito a mesma pessoa). Em outras palavras, os dados estruturados de um mesmo bloco (registro) possuem uma **relação**.

Registros ou grupos de dados diferentes (como de pessoas diferentes), possuem diferentes valores, mas utilizam a mesma representação estrutural homogênea para armazenar os dados. Ou seja, possuem mesmo **atributos** (pense como sinônimo de campos no exemplo acima) e **formatos**, mas valores diferentes.

Então pode-se afirmar que **banco de dados** é um exemplo de dados estruturados. Além de banco de dados, pode-se afirmar também que um **formulário** de cadastro, mesmo que armazenado em outro recurso fora banco de dados (como em um arquivo), também é um exemplo de dados estruturados por conter campos definidos por uma estrutura rígida e previamente projetada, se enquadrando na definição.

Exemplos de Dados Estruturados

O exemplo mais típico de dados estruturados é um **banco de dados**. Nele, os dados são estruturados conforme a definição de um **esquema**, que define as tabelas com seus respectivos campos (ou atributos) e tipos (formato). O **esquema** pode ser pensado como uma meta-informação do **banco de dados**, ou seja, uma descrição sobre a organização dos dados que serão armazenados no banco. É exatamente como no exemplo do formulário que, normalmente, está interligado com um banco de dados.

Dados Não Estruturados

Qual é o oposto de uma estrutura rígida e previamente pensada? Uma estrutura flexível e dinâmica ou **sem estrutura definida**. Exemplo mais comum? Um documento.

Pense em um arquivo criado a partir de um editor de texto. Pode-se adicionar a ele quanto texto quiser, sem se preocupar com campos, restrições e limites. O arquivo pode conter também imagens, como gráficos e fotos, misturado com textos. Imagens, assim como vídeos ou arquivos de áudio, são também exemplos de dados não estruturados.

Assim, é fácil concluir que as redes sociais, as quais possuem um enorme volume de dados, como textos, imagens e vídeos criados diariamente por usuários, representam outro exemplo de dados não estruturados. Atualmente, mais de 80% do conteúdo digital gerado no mundo é do tipo não estruturado ([artigo](#)).

Exemplos de Dados Não Estruturados

Normalmente, basta pensar em uma situação de dados que não seguem estrutura para termos exemplos de dados não-estruturados, mas é preciso tomar um pouco de cuidado com essa análise.

Em computação, todo dado, seja ele um arquivo ou um campo rígido, terá que ter algum tipo de estrutura, mesmo que mínima. Um **arquivo** é um tipo de estrutura mínima, pois é a unidade básica de armazenamento de um sistema operacional, mas ela é genérica, pois

aceita diferentes tipos de dados. Em resumo, quase tudo cairá em um arquivo, mesmo porque um vídeo tem que gravar em arquivo seus dados com um codificador (codec), um áudio também e assim por diante. Portanto, na estrutura interna do arquivo, se ela existe e é rígida, ou não.

Assim, possivelmente, a maior parte dos arquivos que você pensar serão não-estruturados. Vamos aos exemplos:

- Textos diversos (páginas da internet, relatórios, documentos, e-mails, mensagens em aplicativos como Whats App, etc)
- Imagens (fotos, gráficos, ilustrações, desenhos, etc)
- Arquivos de áudio (música, streaming, etc)
- Arquivos de vídeo (filmes, seriados, feitos por usuários, etc)
- Redes sociais (blogs, Facebook, Twitter, Instagram, LinkedIn, etc)

Dados Semiestruturados

Apresentam uma representação heterogênea, ou seja, possuem estrutura, mas ela é flexível. Assim, ela facilita o controle por ter um pouco de estrutura, mas também permite uma maior flexibilidade.

Um exemplo típico é um arquivo em **XML** (eXtensible Markup Language, que significa, em português, linguagem de marcação estendida), o qual possui nós, que são rótulos de abertura e fechamento, este precedido com o símbolo “/”, com os dados inseridos entre os nós. Vamos exemplificar.

Imagine o seguinte texto bruto (digo, por estar em um editor de texto):

Nome: *Leandro Pinho Monteiro*

E-mail: leandro@mentalguild.com.br

Rua Siqueira Bueno, 1134, Mooca, São Paulo.

Agora, pensem nesses dados não-estruturados transpostos para um arquivo XML. O conteúdo do arquivo ficaria assim:

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <cadastro>
3   <nome>Leandro Pinho Monteiro</nome>
4   <e-mail>leandro@mentalguild.com.br</e-mail>
5   <endereço>Rua Siqueira Bueno, 1134, Mooca, São Paulo</endereço>
6 </cadastro>
```

A primeira linha serve para avisar que o arquivo contém uma estrutura XML e que usa codificação de caracteres unicode.

Outros exemplos de arquivos com dados semi-estruturados:

- JSON – Javascript Object Notation,
- RDF – Resource Description Framework,
- OWL – Web Ontology Language.



5. Banco de dados (RDBMS vs NoSQL)

Um banco de dados é uma coleção organizada de dados, geralmente armazenados e acessados eletronicamente de um sistema de computador. São coleções organizadas de dados que se relacionam de forma a criar algum sentido e dar mais eficiência durante uma pesquisa ou estudo.

O Sistema de Gerenciamento de Banco de Dados (SGBD), do inglês Database Management System (DBMS), é o software que interage com usuários finais, aplicativos e com o próprio banco de dados para capturar e analisar os dados. Os SGBDs podem ser classificados de acordo com o modelo do banco de dados, os modelos mais comuns são SGBDs Relacional e NoSQL.

*Obs.: Como curiosidade a DB-ENGINES mantém o ranking de utilização dos bancos de dados no [link](#).

Bancos de Dados Relacionais

Os sistemas bancos de dados Relacionais, do inglês *Relational Database Management System* (RDBMS), utilizam o modelo relacional de dados. Este modelo consiste de Tabelas, Colunas, Registros, Relacionamentos entre outros componentes. Abaixo tem-se um breve resumo destes componentes:

- **Tabelas:** estrutura de linhas e colunas para armazenamento de dados
- **Colunas (atributos):** atributos de uma tabela. São as características dos dados armazenados, como por exemplo: Nome, Data de nascimento, etc.
- **Registros (tuplas):** Um registro ou tupla é uma linha formada por uma ou mais colunas
- **Chaves:** as tabelas se relacionam através de chaves. Chave Primária é o identificador exclusivo de uma tabela. Chave estrangeira é a chave de relação entre uma tabela e outra.

Em bancos de dados relacionais é necessário estruturar os objetos antes de inserir os dados. Esta estruturação dos objetos, dá-se o nome de Modelagem. A Modelagem pode seguir um processo de Normalização, que simplifica grupos complexos de dados para evitar redundâncias e possibilitar um maior desempenho nas pesquisas[1].

Existem cinco estágios de normalização, 1º ao 5º. Para um banco de dados se encontrar em cada um desses estágios ou formas (denominadas formas normais), cada uma de suas tabelas deve atender a alguns pré-requisitos. Os pré-requisitos são cumulativos, isto é,

para alcançar a 3ª forma normal (3NF), um banco de dados precisa atender aos pré-requisitos das 1ª e 2ª formas normais, acrescidos dos requisitos exclusivos da 3NF.

Para conhecer mais sobre as Formas Normais sugiro ler o material [aqui](#). ➡

SGBDs relacionais respeitam as propriedades de transação **ACID** (Atomicidade, Consistência, Isolamento e Durabilidade - do inglês, *Atomicity, Consistency, Isolation, Durability*). Todas as transações que ocorrem num banco de dados relacional satisfazem as propriedades ACID. Resumidamente, cada uma dessas propriedades afirma:

- **Atomicidade:** garante que uma transação é realizada por completo ou não é realizada. Não existe transação parcial;
- **Consistência:** garante que somente dados válidos são armazenados de acordo com as regras pré-definidas no banco de dados;
- **Isolamento:** garante a proteção de que o dado de uma transação não afete o dado de outra transação;
- **Durabilidade:** garante que todos salvos não são perdidos.

Ainda em SGBDs relacionais, assim como qualquer outro sistema, a arquitetura tem um grande impacto no SGBD. Estes sistemas têm 2 possíveis arquiteturas **OLAP** (Processamento Analítico em Tempo Real, do inglês Online Analytical Processing) e **OLTP** (Processamento de Transações em Tempo Real, do inglês Online Transaction Processing).

- **OLAP:** otimizado para realização de seleção/extração de dados ou de grande volume de dados. Ideal para sistemas como de DW.
- **OLTP:** otimizado para registrar transações. Ideal para sistemas com interações com clientes ou sistemas transacionais.

A linguagem predominante num banco de dados relacional é o SQL (Linguagem de Consulta Estruturada, do inglês *Structured Query Language*). É uma linguagem simples e de fácil uso.

Bancos de dados NoSQL

Bancos de dados NoSQL (*Not only Structured Query Language*) é um termo genérico que representa os bancos de dados não relacionais. Com o objetivo de atender as necessidades de aplicações que necessitam de um desempenho superior ao oferecido por bancos de dados relacionais e realizar o dimensionamento da infraestrutura conforme a demanda que fez com que os bancos de dados NoSQL surgissem. Assim como os bancos relacionais, bancos de dados NoSQL respeitam propriedades específicas, chamadas de BASE (Basicamente Disponível, Estado Leve e Eventualmente consistente, do inglês Basically Available, Soft-State e Eventually Consistent).

- **Basicamente Disponível:** distribui os dados em diferentes repositórios tornando-os sempre disponíveis para as aplicações;
- **Estado Leve:** não é consistente o tempo todo o que deixa o funcionamento leve;
- **Eventualmente Consistente:** garante que em algum momento do tempo os dados estarão consistentes.

Uma aplicação funciona basicamente todo o tempo (Basicamente Disponível), não tem de ser consistente todo o tempo (Estado Leve) e o sistema torna-se consistente no momento devido (EventualmenteConsistente)[2].

Tipos de NoSQL

Os Bancos de dados NoSQL possuem diferentes tipos de bancos de dados com propósitos específicos. Todos os bancos de dados NoSQL se enquadram em um dos tipos abaixo e alguns acumulam mais de um tipo.

- **Banco de dados chave-valor (key-value):** usa chaves para armazenar valor. Valores podem ser objetos binários simples que vão de textos planos (simples *strings*) até vídeos. Exemplos: Redis, Amazon DynamoDB, Memcached e CosmosDB;
- **Banco de dados Orientado a Documentos:** similar ao key-value, exceto que o valor é um único documento, normalmente um XML, JSON ou BSON. Exemplos: MongoDB, CouchBase, Amazon DynamoDB e CosmosDB;
- **Banco de dados Orientado a Colunas:** similar ao key-value, porém armazena coluna e valor. Exemplos: HBase, Cassandra e CosmosDB;
- **Banco de dados Orientado a Grafos:** usa estruturas de grafos para armazenar, mapear e relacionar dados. Bancos de grafos não tem tabelas e colunas, tem sim nodos, vértices e relações. Levam vantagem em mineração de dados e reconhecimento de padrões. Exemplos: Neo4j, CosmosDB e OrientDB.

Para mais informações sobre os tipos de bancos de dados, sugerimos o DB-ENGINES [link](#). 🔗

Bancos de dados NoSQL x Relacional

Teorema CAP

O Teorema CAP ou também chamado de Teorema de Brewer, Eric Brewer, resumidamente afirma que é impossível para um sistema de armazenamento de dados distribuído fornecer simultaneamente mais de duas das três propriedades a seguir:

- **Consistência (*Consistency*)**: Cada leitura recebe a escrita mais recente ou uma mensagem de erro.
- **Disponibilidade (*Availability*)**: Cada requisição recebe uma resposta (sem erro) - sem a garantia de que contém a escrita mais recente.
- **Tolerância à Partição (*Partition tolerance*)**: O sistema continua operando apesar de um número arbitrário de mensagens serem descartadas (ou atrasadas) pela rede entre os nós.

NOTE: Nunca um sistema possuirá as 3 propriedades!

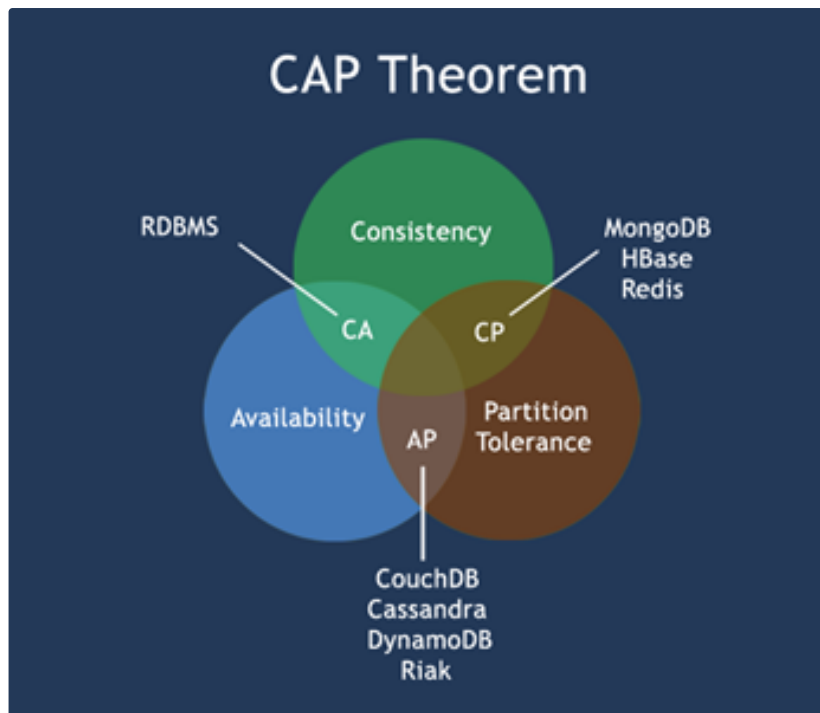


Figura - Representação do Teorema CAP

NoSQL x Relacional

E como os sistemas são diferentes, precisamos compará-los para estas diferenças ficarem mais visíveis. Abaixo tem-se uma tabela sintética com as principais diferenças observadas:

Característica	NoSQL	RDBMS
Escalabilidade	Horizontal	Vertical
Querying	SQL-like (ex: JSON)	SQL
Armazenamento	Plataformas de <i>storage</i> hierarquicamente distribuídas	Plataformas de <i>storage</i> tradicionais
Aplicações Transacionais	Melhor para transações leves	Melhor para transações complexas
Propriedades	BASE	ACID

Casos de Uso e Não-Uso

Quando se deve ou não usar cada uma das tecnologias.

NoSQL

- **Uso:**
 - Necessidade de ir além do SQL
 - Aplicações que precisam de flexibilidade de *schema*
 - Aplicações que precisam de Escalabilidade horizontal
 - Mais performance para grandes conjuntos de dados
- **Não-Uso:**
 - Necessidade de consistência o tempo todo (exemplo: numa consulta de saldo bancário). NoSQL possuem consistência variável
 - Não indicada para processos transacionais que necessitem de confirmação de transação com baixa latência. Bancos NoSQL possuem latência transacional.

RDBMS

- **Uso:**
 - Aplicações com necessidade de consistência transacional com baixa latência
 - Aplicações com transações complexas
- **Não-Uso:**
 - Aplicações que tem demanda variável acabam necessitando de muito HW para um banco de dados relacional. Devido a escalabilidade vertical
 - Aplicações que precisam de flexibilidade de *schema*

Como sugestão pode-se instalar diferentes tipos de bancos de dados e realizar testes simples para fixar o conhecimento.

Change Data Capture (CDC)

É um *Design Pattern* em bancos de dados utilizado para determinar os dados que mudaram para que a ação possa ser executada. Esta abordagem de integração tem como objetivo diminuir a necessidade de carregar muitos dados e reduzir janelas de cargas batch ou streaming/micro-batch. Pode ser utilizada para sincronizar sistemas ou replicar dados. Existem alguns métodos de implementação deste *pattern*, os principais são:

- Linhas com Data de Modificação: O objeto monitorado possui uma coluna que representa a hora ou minuto ou segundo da última alteração.
- Linhas com nº de versão do registro/linha: O objeto monitorado possui uma coluna que representa a versão da última alteração.
- Linhas com indicadores de status: O objeto monitorado possui uma coluna com o status indicador que a linha foi modificada.
- Tabelas com triggers: Quando alguma operação de *insert* ou *update* ou *delete* ocorrerem no objeto monitorado, uma trigger inclui essa informação numa tabela *shadow* (sombra). Nesta tabela pode ter a linha inteira modificada ou apenas a chave primária é armazenada, bem como o tipo de operação (inserir, atualizar ou excluir).
- Pesquisa em Logs Transacionais do banco de dados: Os bancos de dados transacionais armazenam todas as alterações em um log de transações para recuperar o estado do banco de dados, caso o banco de dados falhe por qualquer motivo. O CDC baseado em log aproveita esse aspecto do banco de dados transacional para ler as alterações do log do objeto monitorado.
- Eventos programados: Realiza uma alteração no código fonte do sistema, em pontos apropriados que se referem ao objeto monitorado, para fornecer o CDC de que os dados foram alterados.

6. Tipos de armazenamentos de dados

Existem diversos formatos que podem ser utilizados para armazenar dados em arquivos, como por exemplo o arquivo de texto de extensão TXT. Em se falando de armazenamento de dados, os formatos mais utilizados são TXT, CSV, XML, JSON, ORC, PARQUET e AVRO. Abaixo tem-se uma linha do tempo destes arquivos.

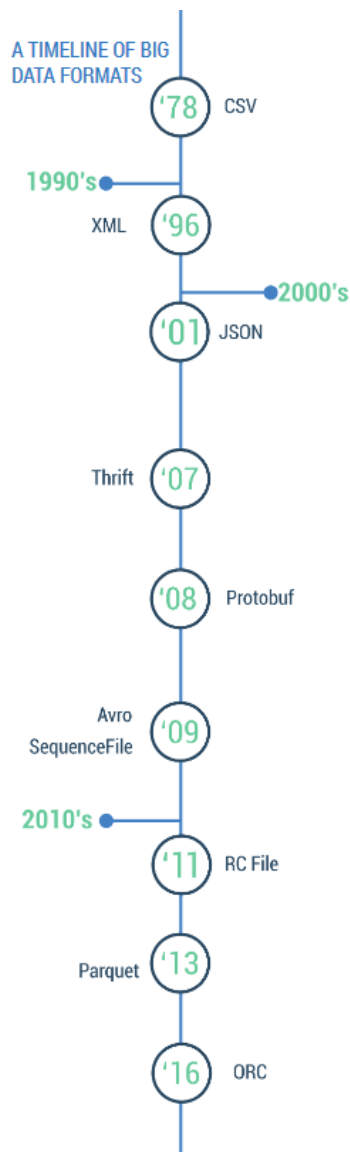


Figura – Nexla *whitepaper* “An Introduction to Big Data Formats: Understanding Avro, Parquet, and ORC”

TXT: A definição precisa do .txt formato não é especificado, mas normalmente coincide com o formato aceito pelo sistema terminal ou simples editor. Arquivos com a extensão. Txt extensão pode ser facilmente lido ou abertos por qualquer programa que lê texto e, por essa razão, são considerados universais (ou plataforma independente). Estes arquivos são facilmente editáveis, bastando alterar o que está escrito, com um simples delete seguido de nova escrita.

log1-7_14.55.txt - Notepad

File Edit Format View Help

```

Going to next goal at 55000 0
WEEK: 1799 SECONDS: 251731.358 X: 32.896391 Y: -117.200281 Heading: 178
WEEK: 1799 SECONDS: 251731.452 X: 32.896391 Y: -117.200281 Heading: 178
WEEK: 1799 SECONDS: 251731.550 X: 32.896391 Y: -117.200281 Heading: 178
WEEK: 1799 SECONDS: 251731.652 X: 32.896391 Y: -117.200281 Heading: 178
WEEK: 1799 SECONDS: 251731.751 X: 32.896391 Y: -117.200281 Heading: 178
WEEK: 1799 SECONDS: 251731.851 X: 32.896391 Y: -117.200281 Heading: 178
WEEK: 1799 SECONDS: 251731.956 X: 32.896390 Y: -117.200281 Heading: 178
WEEK: 1799 SECONDS: 251732.052 X: 32.896390 Y: -117.200281 Heading: 178
WEEK: 1799 SECONDS: 251732.152 X: 32.896390 Y: -117.200281 Heading: 178
WEEK: 1799 SECONDS: 251732.252 X: 32.896390 Y: -117.200281 Heading: 178
WEEK: 1799 SECONDS: 251732.350 X: 32.896390 Y: -117.200281 Heading: 178
WEEK: 1799 SECONDS: 251732.452 X: 32.896389 Y: -117.200281 Heading: 178
WEEK: 1799 SECONDS: 251732.555 X: 32.896389 Y: -117.200281 Heading: 178
WEEK: 1799 SECONDS: 251732.653 X: 32.896389 Y: -117.200281 Heading: 178
WEEK: 1799 SECONDS: 251732.751 X: 32.896388 Y: -117.200281 Heading: 178
WEEK: 1799 SECONDS: 251732.853 X: 32.896388 Y: -117.200281 Heading: 178
WEEK: 1799 SECONDS: 251732.953 X: 32.896388 Y: -117.200281 Heading: 178
WEEK: 1799 SECONDS: 251733.055 X: 32.896387 Y: -117.200280 Heading: 178

```

CSV: O *Comma-Separated Values* é um arquivo de texto delimitado que usa uma vírgula para separar valores. Um arquivo CSV armazena dados tabulares (números e texto) em texto sem formatação. Cada linha do arquivo é um registro de dados. Cada registro consiste em um ou mais campos, separados por vírgulas. O uso da vírgula como separador de campos é a fonte do nome para este formato de arquivo. O formato do arquivo CSV não é totalmente padronizado. A ideia básica de separar campos com vírgula é clara, mas essa ideia fica complicada quando os dados do campo também podem conter vírgulas ou mesmo quebras de linha incorporadas. As implementações de CSV podem não manipular esses dados de campo ou podem usar aspas para delimitar o campo. As aspas não resolvem tudo: alguns campos podem precisar de aspas incorporadas; portanto, uma implementação CSV pode incluir caracteres de escape ou sequências de escape. Além disso, o termo 'CSV' também é utilizado por alguns formatos que usam diferentes delimitadores de campo, por exemplo, ponto e vírgula. Isso inclui valores separados por tabulação e valores separados por espaço. Esses arquivos alternativos separados por delimitador geralmente recebem uma extensão .csv, apesar do uso de um separador de campo sem vírgula. Essa terminologia flexível pode causar problemas na troca de dados. Muitos aplicativos que aceitam arquivos CSV têm opções para selecionar o caractere delimitador e o caractere de cotação. O ponto e vírgula é frequentemente usado em alguns países europeus, como a Itália, em vez de vírgulas.

persons.csv - Notepad

File Edit Format View Help

```

Family Name,Given Name,VIAF ID
Ackersdijck,Willem Cornelis,17959345
Ade|lung,Friedrich von,22963658
Afzelius,Arvid August,49972119
Amerling,Karel,13331054
Anton,Karl Gottlob von,183632821
Arwidsson,Adolf Ivar,8184878
Asbjørnsen,Peter Christen,116587918
Attems,Heinrich,37665468
Atterbom,Per Daniel Amadeus,46819248
Balabin,Viktor Petrovich,44473845
Banks,Joseph,46830189
Beck,Friedrich,44338671
Becker,Reinhold von,42101066
Bernhart,Johann Baptist,69674335

```


XML: O *Extensible Markup Language* é um formato de arquivo texto que usa uma recomendação da W3C (*World Wide Web Consortium*) para gerar linguagens de marcação. O XML é um formato para a criação de documentos com dados organizados de forma hierárquica, frequentemente, em documentos de texto formatados, imagens vetoriais ou bancos de dados. Pela sua portabilidade, já que é um formato que não depende das plataformas de hardware ou de software, um banco de dados pode, através de uma aplicação, escrever em um arquivo XML, e um outro banco distinto pode ler então estes mesmos dados. Importantes princípios do XML:

- Separação do conteúdo da formatação
- Simplicidade e legibilidade, tanto para humanos quanto para computadores
- Possibilidade de criação de *tags* sem limitação
- Criação de arquivos para validação de estrutura (chamados DTDs)
- Interligação de bancos de dados distintos
- Concentração na estrutura da informação, e não na sua aparência

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE SLI SYSTEM "F:\Projects\Cadenus\XML\SLI.dtd">
<SLI>
  <ID>352189</ID>
  <SLAID>726549</SLAID>
  <FlowIdentification>
    <SourceAddress>217.9.64.200</SourceAddress>
    <DestinationAddress>217.9.64.210</DestinationAddress>
    <SourcePort>80</SourcePort>
    <DestinationPort>90</DestinationPort>
    <ProtocolNumber>10</ProtocolNumber>
  </FlowIdentification>
  <Performance>
    <One-Way-Delay>
      <Value>100ms</Value>
      <MeasurementPeriod>86400s</MeasurementPeriod>
      <ThresholdViolations>5</ThresholdViolations>
    </One-Way-Delay>
    <PacketLoss>
      <Value>0,001</Value>
      <MeasurementPeriod>86400s</MeasurementPeriod>
      <ThresholdViolations>8</ThresholdViolations>
    </PacketLoss>
    <Jitter>
      <Value>5ms</Value>
      <MeasurementPeriod>86400s</MeasurementPeriod>
      <ThresholdViolation>10</ThresholdViolation>
    </Jitter>
    <Throughput>
      <Value>1Mbs</Value>
      <MeasurementPeriod>86400s</MeasurementPeriod>
      <ThresholdViolations>4</ThresholdViolations>
    </Throughput>
  </Performance>
  <Reliability>
    <MeanDownTime>
      <Value>6000s</Value>
    </MeanDownTime>
    <TimeToRepair>
      <Value>3000s</Value>
    </TimeToRepair>
  </Reliability>
  <Sampling Type="Periodic"/>
</SLI>
```

JSON: O *JavaScript Object Notation* é um formato compacto, de padrão aberto independente, de troca de dados simples e rápida (*parsing*) entre sistemas, que utiliza texto legível a humanos, no formato atributo-valor (natureza auto-descritiva). Isto é, um modelo de transmissão de informações no formato texto, muito usado em *web services* que usa transferência de estado representacional (REST) e aplicações AJAX, substituindo o uso do XML. O JSON é um formato de troca de dados entre sistemas independente de linguagem de programação derivado do *JavaScript*. O JSON pode ser considerado concorrente da XML na área de troca de informações. Vejamos algumas das principais semelhanças e diferenças entre os modelos de marcação das informações:

Semelhanças:

- Representam informações no formato texto;
- Possuem natureza auto-descritiva;

- Ambos são capazes de representar informação complexa, difícil de representar no formato tabular (CSV). Alguns exemplos: objetos compostos (objetos dentro de objetos), relações de hierarquia, atributos multivalorados, *arrays*, dados ausentes, etc.
- Ambos podem ser utilizados para transportar informações em aplicações AJAX.
- Ambos podem ser considerados padrões para representação de dados. XML é um padrão W3C, enquanto JSON foi formalizado na RFC 4627.
- Ambos são independentes de linguagem. Dados representados em XML e JSON podem ser acessados por qualquer linguagem de programação, através de API's específicas.

Diferenças:

- Não é uma linguagem de marcação. Não possui *tags* de abertura e de fechamento;
- Representa as informações de forma mais compacta;
- Não permite a execução de instruções de processamento, enquanto é possível em XML.
- É tipicamente destinado para a troca de informações, enquanto XML possui mais aplicações. Por exemplo: existem bancos de dados no formato XML e estruturados em SGBD XML nativo.

```
[
  {
    "Name": "Yogurt Depot",
    "Id": 1,
    "Revenue": 2000,
    "Cost": 100,
    "Category": [ "dessert", "food", "yogurt" ],
    "Visits": [
      {
        "day": "Mon",
        "visit_count": 300
      },
      {
        "day": "Tue",
        "visit_count": 700
      }
    ],
    "City": "Tucson",
    "Stars Rated": [
      { "stars": "5", "customers Rated": 10 },
      { "stars": "4", "customers Rated": 8 },
      { "stars": "3", "customers Rated": 120 },
      { "stars": "2", "customers Rated": 6 },
      { "stars": "1", "customers Rated": 0 }
    ]
  },
  {
    "Name": "Corner Bakery",
    "Id": 2,
    "Revenue": 6100,
    "Cost": 120,
    "Category": [ "bakery", "food" ],
  }
]
```

AVRO: O Apache Avro é um formato baseado em linha que é altamente divisível. O recurso inovador e essencial do Avro é que o esquema viaja junto com os dados. **A definição de dados (metadados) é armazenada no formato JSON enquanto os dados são armazenados no formato binário**, minimizando o tamanho do arquivo e maximizando a eficiência.

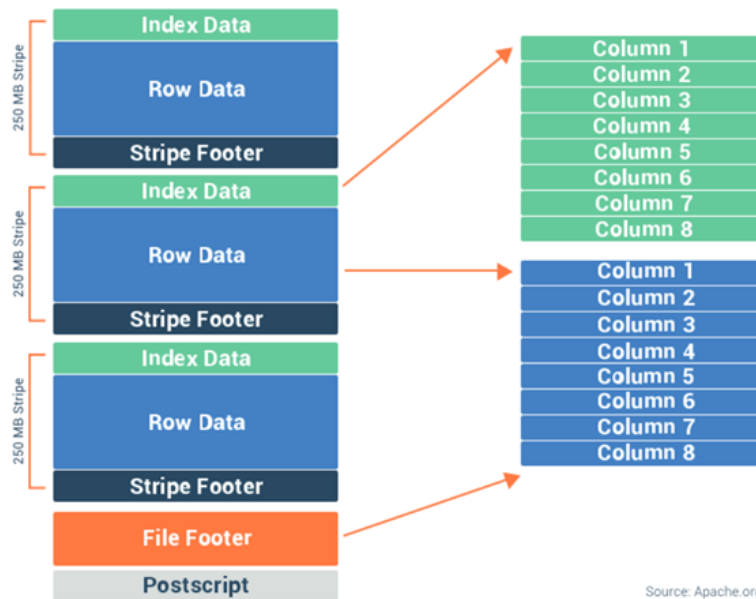
O Avro oferece suporte robusto à evolução do esquema, gerenciando campos adicionados, campos ausentes e campos que foram alterados. Isso permite que o software antigo leia os novos dados e o novo software leia os dados antigos - um recurso crítico se os dados tiverem potencial para mudar. Com a capacidade da Avro de gerenciar a evolução do esquema, é possível atualizar componentes de forma independente, em momentos diferentes, com baixo risco de incompatibilidade. Isso evita que os aplicativos precisem escrever

declarações if-else para processar diferentes versões de esquema e evita que o desenvolvedor tenha que olhar o código antigo para entender os esquemas antigos. Como todas as versões do esquema são armazenadas em JSON legível por humanos, é fácil entender todos os campos disponíveis. O Avro pode suportar muitas linguagens de programação diferentes. Como o esquema é armazenado em JSON enquanto os dados estão em binário, o Avro é uma opção relativamente compacta para armazenamento persistente de dados e transferência eletrônica. O Avro normalmente é o formato de escolha para cargas de trabalho com muita gravação, pois é fácil acrescentar novas linhas.

PARQUET: Lançado em 2013, o Parquet foi desenvolvido pela Cloudera e pelo Twitter (e inspirado no sistema de consultas Dremel do Google) para servir como um armazenamento de dados colunares otimizado para o Hadoop. Como os dados são armazenados por colunas, eles podem ser altamente compactados e divididos. Parquet é comumente usado com o Apache Impala, um banco de dados MPP do Hadoop. O Impala foi projetado para consultas de baixa latência e alta simultaneidade no Hadoop. Os metadados da coluna para um arquivo Parquet são armazenados no final do arquivo, o que permite gravação rápida em uma passagem. Os metadados podem incluir informações como tipos de dados, esquema de compactação/codificação usado (se houver), estatísticas, nomes de elementos e muito mais. O Parquet é especialista em analisar amplos conjuntos de dados com muitas colunas. Cada arquivo Parquet contém dados binários organizados por "grupo de linhas". Para cada grupo de linhas, os valores dos dados são organizados por coluna. Isso permite os benefícios de compactação. O Parquet é uma boa opção para cargas de trabalho com leitura pesada. Geralmente, a evolução do esquema no tipo de arquivo Parquet não é um problema e é suportada. No entanto, nem todos os sistemas que preferem o Parquet suportam a evolução do esquema de maneira facilitada. Por exemplo, considere um armazenamento colunar como Impala. É difícil para esse armazenamento de dados oferecer suporte à evolução do esquema, pois o banco **de dados precisa ter duas versões do esquema (antiga e nova) para uma tabela.**

ORC: O *Optimized Row Columnar* foi desenvolvido pela primeira vez na Hortonworks para otimizar o armazenamento e o desempenho no Apache Hive, resumidamente um data warehouse para consultas e análises no Hadoop. O Hive foi projetado para consultas e análises e usa a linguagem de consulta HiveQL (semelhante ao SQL). Os arquivos ORC são projetados para alto desempenho quando o Hive está lendo, gravando e processando dados. O ORC armazena dados em linha no formato colunar. Esse formato de coluna de linha é altamente eficiente para compactação e armazenamento. Ele permite o processamento paralelo num cluster, e o formato colunar permite pular colunas desnecessárias para um processamento e descompressão mais rápidos. Os arquivos ORC podem armazenar dados com mais eficiência sem compactação do que os arquivos de texto compactados. Como o Parquet, o ORC é uma boa opção para cargas de trabalho pesadas. Esse nível avançado de compactação é possível devido ao seu sistema de indexação. Os arquivos ORC contêm "faixas" (stripes) de dados. Essas faixas são os blocos de dados e independentes uma da outra, o que significa que as consultas podem pular para a faixa necessária conforme a necessidade. Dentro de cada faixa, o leitor pode focar apenas nas colunas necessárias. O arquivo de rodapé inclui estatísticas descritivas para cada coluna em uma faixa, como contagem, soma, min, máximo e se valores nulos estiverem presentes. O ORC foi projetado para maximizar o armazenamento e a eficiência da consulta. De acordo com a Apache Foundation, "o Facebook usa o ORC para salvar dezenas de petabytes em seu data warehouse e demonstrou que o ORC é significativamente mais rápido que o RC File ou Parquet." Semelhante ao Parquet, a evolução do esquema é suportada pelo formato de arquivo ORC, mas sua eficácia depende se o armazenamento de dados tem suporte. Avanços recentes foram feitos no Hive que permitem anexar colunas, conversão de tipos e mapeamento de nomes.

ORC FILE STRUCTURE



Source: Apache.org.
<https://cwiki.apache.org/confluence/display/Hive/LanguageManual+ORC>

Voltando à estrutura, compara-se o Avro, o Parquet e o ORC em relação ao suporte à divisão, compactação e evolução do esquema. A primeira determinação provavelmente será se seus dados são mais adequados para serem armazenados em linha ou coluna. Dados transacionais, dados no nível do evento e casos de uso para os quais você precisará alavancar muitas colunas são mais adequados para dados baseados em linhas. Se for esse o caso, o Avro é provavelmente a melhor escolha. O Avro geralmente é a opção para mais cargas de trabalho com muita gravação, pois seu formato baseado em linha é mais fácil de acrescentar (semelhante a uma estrutura tradicional de banco de dados). Mas se os dados são mais adequados para um formato colunar, a questão será Parquet ou ORC. Além da importância relativa do suporte à divisão, compactação e evolução de esquema, deve-se considerar sua infraestrutura existente. O ORC maximiza o desempenho no Hive. Ambos os formatos oferecem benefícios e provavelmente se resumirá a qual sistema você tem acesso e está mais familiarizado. Os formatos colunares são a opção para cargas de trabalho pesadas para leitura, devido aos ganhos de eficiência em termos de divisão e compactação discutidos neste documento.

BIG DATA FORMATS COMPARISON

	Avro	Parquet	ORC
Schema Evolution Support			
Compression			
Splitability			
Most Compatible Platforms	Kafka, Druid	Impala, Arrow Drill, Spark	Hive, Presto
Row or Column	Row	Column	Column
Read or Write	Write	Read	Read

Source: Nexla analysis, April 2018

7.Data Lake (Arquitetura Lambda)

Um Data Lake é um local central para armazenar todos os seus dados, independentemente de sua origem ou formato [6]. Data Lakes são alimentados com informações em sua forma nativa com pouco ou nenhum processamento realizado para adaptar a estrutura a um esquema corporativo. A estrutura dos dados coletados, portanto, não é conhecida quando é inserida no Data Lake, mas é encontrada somente por meio da descoberta, quando lida. Por isso uma grande vantagem de um Data Lake é a flexibilidade.

Atributo	DW	Data Lake
Schema	Schema-on-write	Schema-on-read
Escala	Escala para grandes volumes a um custo moderado	Escala para grandes volumes a um custo baixo
Métodos de Acesso	Acessado através de padrões SQL e ferramentas de BI	Acessado através de sistemas como SQL, programas criados por desenvolvedores e outros métodos
Workload	Suporta processamento batch, bem como milhares de usuários simultâneos realizando análises interativas	Suporta processamento batch, além de um recurso aprimorado sobre EDWs para suportar consultas interativas de usuários
Dado	Limpo	Bruto (Raw)
Complexidade	Integrações Complexas	Processamento Complexos
Custo/Eficiência	Uso eficiente de CPU/IO	Uso eficiente das capacidades de armazenamento e processamento a um custo muito baixo
Benefícios	<ul style="list-style-type: none"> • Transforma uma vez, usa muitas vezes • Dados limpos, seguros e protegidos • Fornece uma visão única da empresa dos dados de várias origens • Fácil de consumir os dados • Alta concorrência • Desempenho consistente • Rápidos tempos de resposta 	<ul style="list-style-type: none"> • Transforma a economia financeira do armazenamento de grandes quantidades de dados • Suporta Pig e HiveQL e outros frameworks de programação de alto nível • Escala para executar em dezenas de milhares de servidores • Permite o uso de qualquer ferramenta • Permite que a análise comece assim que os dados chegam • Permite o uso de conteúdo estruturado e não estruturado em um único storage • Suporta modelagem ágil, permitindo que os usuários alterem modelos, aplicativos e consultas (queries)

Atributos Chaves de um Data Lake

Um repositório para ser considerado um Data Lake deve ter pelo menos as três (03) características abaixo:

- Deve ser um único repositório compartilhado de dados. Normalmente armazenado em Hadoop Distributed File System (HDFS)
- Incluir capacidades de orquestração e agendamento de tarefas(jobs)
- Conter um conjunto de aplicativos ou de workflows para consumir, processar ou agir de acordo com os dados

Arquitetura Lambda

Primeiramente, arquitetura Lambda não tem nada a ver com o serviço da AWS. Arquitetura Lambda é um modelo de arquitetura BigData proposto por Nathan Marz [21]. Este modelo independe de soluções tecnológicas específicas para a ingestão, armazenamento e processamento dos dados, ou seja, é um modelo teórico. Marz reforça que não há uma única ferramenta que provem uma solução completa de BigData, é necessário utilizar uma variedade de ferramentas e técnicas para construir um sistema completo de Big Data. Por isso tem-se que reforçar a importância do desenho de arquitetura de uma solução.

Para que os dados sejam processados e entregues atingindo uma expectativa de tempo dos *stakeholders*, a arquitetura Lambda é dividida em três camadas: **batch layer**, **speed layer** e **serving layer** de acordo com a figura abaixo:

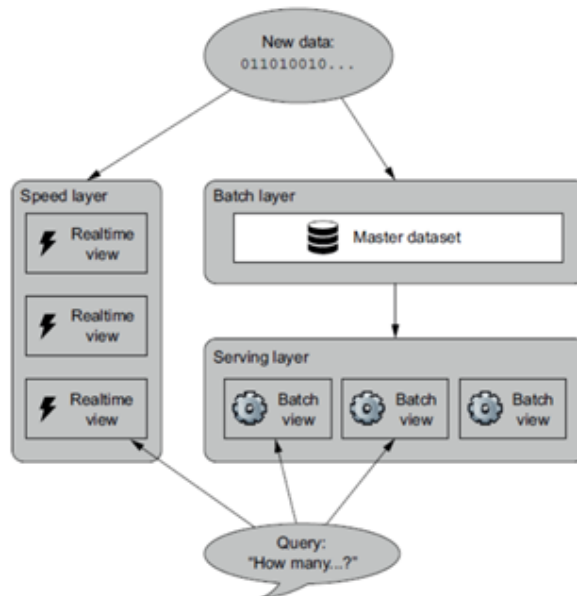


Figura - Arquitetura Lambda proposta por Marz.

O dado a ser processado é direcionado para duas camadas (batch e speed). Dentro da **batch layer** esse dado é armazenado de maneira atômica, ou seja, nada é atualizado ou sobrescrito. Caso exista a necessidade de uma mudança, uma nova versão do dado já alterada é armazenada e a anterior não é removida e continua sem mudanças. Isso permite que o dado em seu formato original sempre esteja disponível. Esses dados que estão na **batch layer** são então processados para gerar visualizações pré-calculadas com as informações ajustadas e organizadas de acordo com a necessidade de negócio.

Como a quantidade de dados armazenados a cada dia só cresce, e os dados da **serving layer** só são recebidos ao final do processamento da **batch layer**, o resultado é que cada vez o intervalo para a atualização no **serving layer** fica maior.

Visando compensar esse intervalo a **speed layer** foi criada. Essa camada, que recebe a mesma estrutura atômica de dados, irá processá-los em tempo real e disponibilizá-los para que os sistemas finais disponham dessas informações enquanto esperam pela **batch layer**.

A execução na **speed layer** é bem mais complexa uma vez que os dados precisam ser atualizados e agrupados de acordo com a necessidade do negócio, pois se fossem mantidos em sua forma original inviabilizaria o processamento. Porém essa camada somente precisa se preocupar com os dados que ainda *não foram entregues* pela **batch layer**, o que reduz imensamente a quantidade de dados a ser processada. Assim que o processamento da **batch layer** termina e uma nova versão é disponibilizada, esses dados na **speed layer** podem ser descartados.

Utilizando as três camadas dessa arquitetura é possível processar uma quantidade imensa de dados e mantê-los em sua estrutura original na **batch layer**, disponibilizar esses dados em visualizações pré-computadas (**serving layer**), compensar os intervalos da camada batch e continuar entregando as informações em tempo real (**speed layer**).





















Como a arquitetura propõe que os dados sejam armazenados de maneira atômica, o sistema fica protegido até mesmo de erro humano. Também é possível garantir uma visão consistente sem a necessidade de esperar até o final do processamento batch, se beneficiando da **speed layer** e ainda tornando tudo isso transparente aos sistemas finais por meio da **serving layer**.

First make it possible. Then make it beautiful. Then make it fast (Primeiro torne isso possível. Então faça bonito. Por fim faça isso rápido)

— Nathan Marz [21].

8.Referências

[1] Laudon, Kenneth; Jane (2011). Sistemas de Informação gerenciais

- [2]  [NoSQL, Base VS ACID e Teorema CAP](#) , acessado em 01/04/2019
- [3]  [Big Data Battle : Batch Processing vs Stream Processing](#) , acessado em 01/04/2019
- [4] <https://aws.amazon.com/pt/data-warehouse/>
- [5]  [Slowly Changing Dimensions](#)
- [6] LaPlante, Alice; Ben Sharma(2014). Architecting Data Lakes.
- [7]  [O que é Big Data? | Oracle Brasil](#)
- [8]  [Big Data: What it is and why it matters](#)
- [9]  [O que é Data Science? | Oficina da Net](#)
- [10]  [O que é mineração de dados?](#)
- [11]  [Difference of Data Science, Machine Learning and Data Mining - DataScienceCentral.com](#)
- [12]  [What are the differences between Data Science and Data Mining, are they same?](#)
- [13]  [Machine learning: o que é e qual sua importância?](#)
- [14]  [Deep learning: o que é e qual sua importância?](#)
- [15]  [What is an API \(Application Program Interface\)? | Webopedia](#)
- [16]  [What Is an API \(Application Program Interface\)?](#)
- [17]  [Enterprise Service Bus](#)
- [18]  [Zato | O que ESB e SOA são, afinal?](#)
- [19]  [What Is Messaging? \(The Java EE 6 Tutorial\)](#)
- [20]  [Integração por Mensageria \(Messaging\)](#)
- [21] https://thinksis.com/wp-content/uploads/2018/10/Nexla_Whitepaper_Introduction-to-Big-Data-Formats-Saket-Saurabh.pdf
- [22]  [What Happens in an Internet Minute in 2016?](#)
- [23]  [What Happens in an Internet Minute in 2019?](#)
- [24]  [What Happen in an Internet Minute - Bond High Plus](#)