CEFET/RJ - CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA
CELSO SUCKOW DA FONSECA
Campus Nova Friburgo
Bacharelado em Sistemas de Informação
4º Período

Banco de Dados II

Compilação de Materiais

Gestão de Big Data
Universidade da Califórnia - San Diego

Tecnologias NoSQL
Universidade Nacional Autônoma do México

Análise de Big Data com SQL
Cloudera

A presente publicação é uma compilação dos materiais de aula dos professores:

- Glynn Durham (Cloudera)
- Ilkay Altintas (UC San Diego)
- Pilar Ángeles (UNAM)

Nova Friburgo, 01 de junho de 2023

Sumário

1. Big Data

1.1. Introdução

O que lançou a era do Big Data? Duas novas oportunidades contribuíram para o seu lançamento. As oportunidades são muitas vezes um sinal de mudança de tempos. Em 2013, um relatório influente de uma empresa chamada McKinsey afirmou que a área de ciência de dados foi o catalisador número um para o crescimento econômico. McKinsey identificou uma de nossas novas oportunidades que contribuíram para o lançamento da era do Big Data.

Uma crescente torrente de dados. Isso se refere à ideia de que os dados parecem estar vindo continuamente e a um ritmo rápido. Pense nisso, hoje você pode comprar um disco rígido para armazenar todas as músicas do mundo por apenas R$500,00. Essa é uma capacidade de armazenamento incrível em relação a qualquer forma anterior de armazenamento de música. Em 2010, havia 5 bilhões de celulares em uso. Você pode ter certeza de que há mais hoje e como eu tenho certeza que você entenderá, esses telefones e os aplicativos que instalamos neles são uma grande fonte de big data, que o tempo todo, todos os dias, contribui para o nosso núcleo. E o Facebook, que recentemente estabeleceu um recorde de ter um bilhão de pessoas fazendo *login* em um único dia, tem mais de 30 bilhões de pedaços de conteúdo compartilhados todos os meses. Bem, esse número é de 2013. Então, tenho certeza que é muito maior do que isso agora. Faz você pensar quantas ações no Facebook você fez no mês passado? Tudo isso leva a projeções de crescimento sério, com 40% em dados globais por ano e 5% em gastos globais de TI. Esses muitos dados certamente empurraram o campo da ciência de dados para começar a permanecer em si mesmo e no mundo dos negócios de hoje.

Mas, há algo mais contribuindo para o poder catalisador da ciência de dados. É chamado de computação em nuvem. Chamamos isso de computação sob demanda. A computação em nuvem é uma das maneiras pelas quais a computação agora se tornou algo que podemos fazer a qualquer hora e em qualquer lugar. Você pode se surpreender ao saber que alguns de seus aplicativos favoritos são de empresas que estão sendo administradas a partir de cafeterias. Esta nova capacidade, combinada com a nossa torrente de dados, nos dá a oportunidade de realizar análises de dados novas, dinâmicas e escaláveis, para nos dizer coisas novas sobre nosso mundo e nós mesmos. Resumindo, uma nova torrente de big data combinada com a capacidade de computação a qualquer momento, em qualquer lugar, esteve no centro do lançamento dessa nova era.

O que faz de Big Data algo valioso?

É a maneira pela qual o big data pode atender às necessidades humanas que o torna valorizado. Vejamos alguns exemplos dos aplicativos que o big data está nos permitindo imaginar e construir. **Big Data nos permite construir modelos melhores, que produzem resultados de maior precisão**. Estamos testemunhando abordagens extremamente inovadoras na forma como as empresas se comercializam e vendem produtos. Como os recursos humanos são gerenciados. Como os desastres são respondidos. E muitos outros aplicativos que evidenciaram dados baseados estão sendo usados para influenciar decisões.

O que exatamente isso significa? Aqui está um exemplo: a Amazon mantém algumas coisas dos usuários que permitem que eles personalizem o que eles me mostram. Buscam reduzir a enorme quantidade de opções que poderia oferecer ao usuário e exibir apenas o que ele procura, por exemplo, pratos de jantar. As empresas podem aproveitar a tecnologia para tomar decisões mais bem informadas que são realmente baseadas em sinais gerados por consumidores reais, como eu. Big Data permite que você ouça a voz de cada consumidor.

Muitas empresas, incluindo Walmart e Target, usam essas informações para personalizar suas comunicações com seus clientes, o que, por sua vez, leva a melhor atender às expectativas dos consumidores e clientes mais felizes. O que basicamente é dizer, big data permitiu o marketing personalizado. Os consumidores estão gerando dados publicamente acessíveis através de sites de mídia social, como Twitter ou Facebook.

Através desses dados, as empresas são capazes de ver seu histórico de compras, o que eles procuraram, o que eles assistiram, onde eles estiveram e o que eles estão interessados através de seus gostos e ações.

Vejamos alguns exemplos de como as empresas estão colocando essas informações para criar melhores campanhas de marketing e alcançar os clientes certos. Uma área com a qual todos estamos familiarizados são os motores de recomendação. Esses mecanismos aproveitam os padrões do usuário e os recursos do produto para prever a melhor combinação do produto para enriquecer a experiência do usuário.

Se você já fez compras na Amazon, sabe que recebe recomendações com base em sua compra. Da mesma forma, a Netflix recomenda que você assista novos programas baseados no seu histórico de visualização. Outra técnica que as empresas usam é a análise do sentimento, ou em termos simples, a avaliação da percepção em torno de eventos e produtos. Lembra dos pratos azuis que comprei na Amazon.com? Eu não só posso ler os comentários antes de comprá-los, eu também posso escrever uma avaliação do produto assim que eu receber meus pratos. Desta forma, outros clientes podem ser informados. Mas o mais importante, a Amazon pode acompanhar as avaliações de produtos e as tendências para um determinado produto. Neste caso, pratos azuis.

Por exemplo, eles podem avaliar se uma avaliação do produto é positiva ou negativa. Neste caso, enquanto a primeira revisão é negativa, as duas próximas avaliações são positivas. Uma vez que essas avaliações são escritas em inglês usando uma técnica chamada processamento de linguagem natural (PNL), e outros métodos analíticos, Amazon pode analisar a opinião geral de uma pessoa ou público sobre tal produto. É por isso que a análise do sentimento geralmente é referida como mineração de opinião.

Os canais de notícias são preenchidos com a análise de feeds do Twitter sempre que ocorre um evento de importância, como eleições. As marcas utilizam análise de sentimento para entender como os clientes se relacionam com seus produtos, de forma positiva, negativa e neutra. Isso depende muito do uso do processamento de linguagem natural.

Os dispositivos móveis são onipresentes e pessoas quase sempre carregam seus celulares com eles. Publicidade móvel é um enorme mercado para empresas. As plataformas utilizam os sensores em dispositivos móveis, como GPS, e fornecem anúncios baseados em localização em tempo real, oferecem descontos, com base neste dilúvio de dados.

Desta vez, vamos imaginar que eu comprei uma casa nova e eu acontece de estar em poucos quilômetros de um Home Depot, que vai me enviar cupons móveis sobre pintura, prateleiras e outras compras relacionadas com a casa me lembraria da Home Depot. Há uma grande chance de eu parar no Home Depot. Bingo! Quais tipos de big data são necessários para fazer isso acontecer? Definitivamente há a integração das minhas informações de consumidor e as bases de dados online e offline que incluem as minhas compras recentes. Mas o mais importante, os dados de geolocalização que se enquadram em um tipo maior de big data, big data espacial.

Vamos agora falar sobre como o comportamento global do consumidor pode ser usado para o desenvolvimento do produto. Estamos passando da personalização de marketing para o comportamento do consumidor como um todo. Toda empresa quer entender o comportamento coletivo de seus consumidores para capturar o cenário em constante mudança. Vários produtos de big data permitem isso desenvolvendo modelos para capturar o comportamento do usuário e permitir que as empresas direcionem o público certo para seus produtos. Ou desenvolva novos produtos para territórios inexplorados.

Por exemplo: após uma análise de suas vendas durante a semana, uma companhia aérea pode notar que seus voos da manhã estão sempre esgotados, enquanto seus voos da tarde correm abaixo da capacidade. Esta empresa pode decidir adicionar mais voos matinais com base em tal análise. Observe que eles não estão usando opções individuais do consumidor, mas usando todos os voos comprados sem considerar quem os comprou. Eles podem, no entanto, decidir prestar mais atenção à demografia desses consumidores usando big data para também adicionar voos semelhantes em outras regiões geográficas.

Com avanços rápidos na tecnologia de sequenciamento de genomas, a indústria de ciências biológicas está experimentando um enorme atrativo em big data biomédica. Estes dados biomédicos estão sendo usados por muitas aplicações em pesquisa e medicina personalizada. Você sabia que os dados genômicos são um dos maiores tipos de big data em crescimento? Entre 100 milhões e 2 bilhões de genomas humanos poderiam ser sequenciados até o ano 2025. Impressionante. Esta sequência de dados coletada exige entre 2 exabytes e 40 exabytes no armazenamento de dados. Em comparação, todo o YouTube requer apenas 1 a 2 exabytes por ano. Um exabyte é 10^{18} bytes. Ou seja, 18 zeros após 10.

Claro, a análise de tais volumes massivos de dados de sequência é cara. Pode levar até 10.000 trilhões de horas de CPU. Uma das aplicações biomédicas que esta quantidade de dados está habilitando é a medicina personalizada. Antes da medicina personalizada, a maioria dos pacientes sem um tipo específico e estágio de câncer recebeu o mesmo tratamento, que funcionava melhor para alguns do que para os outros. A pesquisa nesta área está possibilitando o desenvolvimento de métodos para analisar dados em larga escala para desenvolver soluções que se adaptam a cada indivíduo, e, portanto, pretendem ser mais eficazes.

Uma pessoa com câncer pode agora ainda receber um plano de tratamento padrão, como cirurgia para remover um tumor. No entanto, o médico também pode ser capaz de recomendar algum tipo de tratamento personalizado do câncer. Um grande desafio em aplicações biomédicas de big data, como muitos outros campos, é como podemos integrar muitos tipos de fontes de dados para obter mais problemas de insight.

Outra aplicação de big data vem de malha interconectada de grande número de sensores implantados em cidades inteligentes. Análise de dados gerados a partir de sensores em tempo real permite que as cidades ofereçam melhor qualidade de serviço aos habitantes. E reduza os efeitos indesejados, como poluição, congestionamento de tráfego, maior do que o custo ideal na prestação de serviços urbanos. Vamos tomar nossa cidade, San Diego. San Diego gera um enorme volume de dados de muitas fontes. Sensores de tráfego, satélites, redes de câmeras e muito mais. E se pudéssemos integrar e sintetizar esses fluxos de dados para fazer ainda mais pela nossa comunidade? Usando tal big data, podemos trabalhar para fazer de San Diego o protótipo da cidade digital. Não só para riscos que ameaçam a vida, mas para melhorar a nossa vida diária, como gerir o fluxo de tráfego de forma mais eficiente ou maximizar a poupança de energia, mesmo como veremos em seguida, incêndios florestais. Como resumo, big data tem um enorme potencial para habilitar modelos com maior precisão em muitas áreas de aplicação. E esses modelos altamente precisos estão influenciando e transformando os negócios.

1.2. Exemplos de aplicação

a. Salvando Vidas com Big Data

O Wildfire Analytics, que se divide em dois componentes - previsão e resposta. Por que isso é tão importante? Em maio de 2014, em San Diego, houve 14 incêndios queimando, até nove de uma só vez, que queimou um total de 26.000 hectares, 11.000 hectares, uma área pouco menos do que o tamanho da cidade de São Francisco. Seis pessoas ficaram feridas e uma pessoa morreu. E esses incêndios florestais resultaram em um custo total de mais de US $60 milhões em danos e combate a incêndios. Esses incêndios podem se tornar tão severos que nós realmente os chamamos de fogo. Embora não possamos controlar tais tempestades de fogo, algo que podemos fazer é chegar à frente deles, prevendo seu comportamento. É por isso que o gerenciamento de desastres de incêndios florestais em curso depende fortemente da compreensão de sua direção e taxa de propagação. Como esses incêndios fazem parte de nossas vidas, queríamos ver se podemos usar Big Data para monitorar, prever e gerenciar uma tempestade de fogo.

Por que o Big Data pode ajudar? Na verdade, a prevenção de incêndios selvagens e a resposta podem se beneficiar de muitos fluxos em nossa torrente de dados. Alguns fluxos são gerados por pessoas através de

dispositivos que possuem. Muitos vêm de sensores e satélites, coisas que medem fatores ambientais. E alguns vêm de dados organizacionais, incluindo mapas de área, melhores atualizações de serviço e bancos de dados de conteúdo de campo, que arquivam o quanto registra vegetação e outros tipos de combustível estão no caminho de um incêndio potencial.

O que torna isso um problema de Big Data? Porque novas abordagens e respostas podem ser tomadas se podemos integrar esses diversos fluxos de dados. Muitas dessas fontes de dados já existem há algum tempo. Mas o que falta no gerenciamento de desastres hoje é um sistema dinâmico integração de redes de sensores em tempo real, imagens de satélite, ferramentas de gerenciamento de dados em tempo real, ferramentas de simulação de incêndio selvagem, conectividade com centros de comando de emergência, e tudo isso antes, durante e depois de uma tempestade de fogo.

A integração de diversos fluxos e maneiras novas é realmente o que está impulsionando nossa capacidade de ver coisas novas e desenvolver análises preditivas, o que pode ajudar a melhorar nosso mundo. Quais são essas fontes diversas? Uma das fontes de dados mais importantes é a transmissão de dados de sensores a partir de estações meteorológicas e satélites, tais dados detectados incluem temperatura, umidade, pressão do ar. Também podemos incluir streaming de dados de imagem de câmeras e satélites de montanha nesta categoria.

Outra fonte de dados importante vem de instituições como o San Diego Supercomputer Center, que geram dados relacionados à modelagem de incêndios. Estes incluem mapas de perímetro de incêndio passados e atuais reunidos por as autoridades e mapas de combustível que nos falam sobre a vegetação, e outros tipos de combustível no caminho de um incêndio. Esses tipos de fontes de dados geralmente são estáticos ou atualizados em um ritmo lento, mas eles fornecem dados valiosos que são bem selecionados e verificados.

Uma grande parte dos dados sobre incêndios é realmente gerada pelo público em sites de mídia social como o Twitter, que suportam recursos de compartilhamento de fotos. Estas são as fontes de dados mais difíceis de simplificar durante um incêndio existente, mas elas podem ser muito valiosas uma vez integradas a outras fontes de dados. Imagine sintetizar todas as imagens no Twitter sobre um incêndio em curso ou verificando o sentimento público em torno dos limites de um incêndio.

Uma vez que você tenha acesso a essas informações na ponta dos dedos, há muitas coisas que você pode fazer com esses dados. Você pode simplesmente monitorá-lo, ou talvez você possa visualizá-lo. Mas não é até você reunir todos esses tipos diferentes de fontes de dados e integrá-los com análise em tempo real e modelagem preditiva que você pode realmente fazer contribuições na previsão e resposta a emergências de incêndios florestais. Então, agora, eu gostaria que você tomasse um momento e imagine como o Big Data pode ajudar com o combate a incêndios no futuro. Todos esses fluxos de dados se juntarão em telas 3D que podem mostrar todas as informações relacionadas, juntamente com previsões meteorológicas e incêndios, assim como a forma como os tornados são gerenciados hoje.

b. Usando Big Data para Ajudar Pacientes

Vejamos um segundo exemplo em que big data pode ter um grande impacto na salvação de vidas. Literalmente salvando vidas uma vida de cada vez: como melhorar a saúde humana através da pesquisa e prática de medicina de precisão? O que é medicina de precisão? É uma área emergente da medicina voltada para uma pessoa individual. Analisando sua genética, seu ambiente, suas atividades diárias para que se pode detectar ou prever um problema de saúde cedo, ajudar a prevenir doenças e, em caso de doença, fornecer a droga certa na dose certa que é adequada apenas para ela. Muito recentemente, a Casa Branca e o Instituto Nacional de Saúde aqui nos EUA declararam que ela é uma área prioritária para pesquisa e desenvolvimento para a próxima década.

Para que qualquer tecnologia tenha sucesso na vida real, precisamos não apenas um certo nível de maturidade da própria tecnologia, mas uma série de fatores propiciadores, incluindo ambiente econômico social, demandas do mercado, disponibilidade do consumidor, custo-eficácia, todos os quais devem trabalhar juntos.

Por que big data para medicina de precisão é importante agora? Um aspecto importante da medicina de precisão é utilizar o perfil genético de um indivíduo para o seu próprio diagnóstico e tratamento. Analisando o genoma humano, que detém a chave para a saúde humana está rapidamente se tornando mais acessível. O custo de hoje para sequenciar um genoma é menos do que 10% do que custou apenas em 2008. Mas, dados genômicos humanos são grandes.

Quão grande? Em um mundo perfeito, apenas os três bilhões de letras do seu genoma precisariam de cerca de 700 megabytes para armazenar. No mundo real, o que significa o tipo de dados gerados a partir de máquinas de sequenciamento do genoma, precisamos de 200 GB para armazenar um genoma. E leva agora cerca de um dia para sequenciar um genoma. Estamos finalmente começando a criar mais registros eletrônicos que podem ser armazenados e manipulados em mídia digital. A maioria dos consultórios médicos e hospitais agora usa sistemas eletrônicos de registro de saúde que contêm todos os detalhes da visita de um paciente e teste de laboratório.

Qual é o tamanho desses dados? Como um exemplo rápido o *Samaritan Medical Center Watertown New York* em 294 que *Community Hospital* relatou 120 terabytes a partir de 2013. O valor dos dados mais do dobro nos últimos dois anos. Então, claramente, apenas nos últimos dois anos, mudanças dramáticas prepararam a indústria de saúde para produzir e analisar grandes volumes de dados complexos de pacientes. Para resumir o que vimos até agora, os principais componentes dessas mudanças são:

- Custo reduzido de geração e análise de dados,
- Maior disponibilidade de armazenamento de dados barato e
- Aumento da digitalização de registros em papel anteriormente.

Mas precisamos de mais uma capacidade para avançar em direção à terra prometida das práticas de saúde individualizadas. Precisamos combinar vários tipos de dados produzidos por diferentes grupos em de forma significativa. A chave é a integração de vários tipos de fontes de dados. Dados de sensores, organizações e pessoas.

Vamos começar com os dados do sensor. Claro, equipamentos hospitalares digitais têm produzido dados de sensores há anos, mas era improvável que os dados tenham sido armazenados ou compartilhados, muito menos analisados retrospectivamente. Estes foram destinados para uso em tempo real, para informar os profissionais de saúde, e depois foram descartados. Agora temos muitos mais sensores e implantação. E muitos mais lugares que estão capturando e explicitamente coletando informações para serem armazenadas e analisadas.

Pegando um novo tipo de dados que está se tornando cada vez mais comum em nossas vidas diárias. Dispositivos de fitness estão em todos os lugares agora suas vendas dispararam nos últimos anos. Eles estão em pulseiras, relógios, sapatos e coletes, comunicando diretamente com o seu dispositivo móvel pessoal, rastreando várias variáveis de atividade, como pressão arterial, diferentes tipos de atividades, níveis de glicose no sangue, etc a cada momento. Seu objetivo é melhorar o bem-estar. Ao fazer você monitorar seu status diário e esperamos melhorar seu estilo de vida para se manter saudável. Mas os dados que eles geram podem ser informações médicas muito úteis porque esses dados são sobre o que acontece em sua vida normal e não apenas quando você vai ao médico. Quantos dados eles geram? O dispositivo chamado FitBit pode produzir vários gigabytes por dia.

Esses dados poderiam ser usados para economizar custos de saúde e afetar um estilo de vida mais saudável? Isso é um ponto de interrogação. É seguro adivinhar que esses dados por si só não conduziriam o sonho de medicina de precisão. Mas e se considerarmos integrá-lo com outras fontes de dados como registros eletrônicos de saúde ou um perfil genômico? Esta continua a ser uma pergunta em aberto. Esta é uma arena aberta para

pesquisas que meus colegas de roteiros estão fazendo. Também é uma área potencialmente significativa para o desenvolvimento de produtos e negócios.

Vejamos alguns exemplos de dados relacionados à saúde que estão sendo gerados por organizações. Muitos bancos de dados públicos, incluindo aqueles tratados e gerenciados pelo NCBI, o Centro Nacional de Informação Biotecnológica, foram criados para capturar os dados científicos básicos e o conhecimento para os humanos e outros organismos modelo nos diferentes blocos de construção da vida. Essas bases de dados carregam dados experimentais e computados que são necessários para observações para doenças não conquistadas como o câncer.

Além disso, muitos criaram bases de conhecimento como a Geneontologia e o Sistema Unificado de Linguagem Médica para reunir conhecimento humano em uma forma processável por máquina. Estes são apenas alguns exemplos de fontes de dados organizacionais e dados governamentais coletados por sistemas de saúde em todo o mundo também poderiam ser usados como uma fonte maciça de informação. Mas realmente algumas das mais interessantes e novas oportunidades parecem provavelmente vir da área de dados gerados por pessoas.

Aplicativos móveis de saúde é uma área que está crescendo significativamente. Existem agora aplicações para monitorizar a frequência cardíaca, a pressão arterial e testar os níveis de saturação de oxigénio. Apps, podemos dizer, registram dados de sensores, mas também são obviamente gerados a partir de pessoas. Mas há mais dados gerados por pessoas que são interessantes além das medidas de censura . Em 2015, o *Webby People's Voice Award* foi para um aplicativo que suporta meditação e atenção plena. Em vez de um dispositivo de detecção eletrônica, um humano indicaria quantos minutos por dia eles passaram meditando. Se eles interagem com o aplicativo que os lembra de estar atentos, então temos um comportamento gerado por humanos que não conseguimos obter de um sensor.

Existem hoje mais de 100.000 aplicativos de saúde no iTunes ou no Google Play. E, por algumas estimativas, o mercado de aplicativos de saúde móvel pode valer 27 bilhões de dólares até 2017. Então realmente somos vistos apenas o início de quais dados podem ser gerados aqui do que está sendo chamado de sensores humanos, mas para realmente entender onde o poder das pessoas geradas dados pode nos levar na era do big data para a saúde.

Vamos imaginar como as coisas estão agora. Em geral, um paciente vai ver seu médico e talvez seu médico pergunte se eles tiveram algum efeito colateral de seus medicamentos. A precisão e daí a qualidade dos dados que os pacientes fornecem neste tipo de configuração é muito baixa. Não que seja realmente culpa dos pacientes. Pode ter sido dias ou semanas atrás que eles experimentaram algo. Eles podem estar inseguros se algo que eles experimentaram foi realmente uma reação para então relatá-lo. E pode haver detalhes sobre exatamente quando eles tomaram um medicamento que são significativos, mas eles esqueceram disso depois do fato.

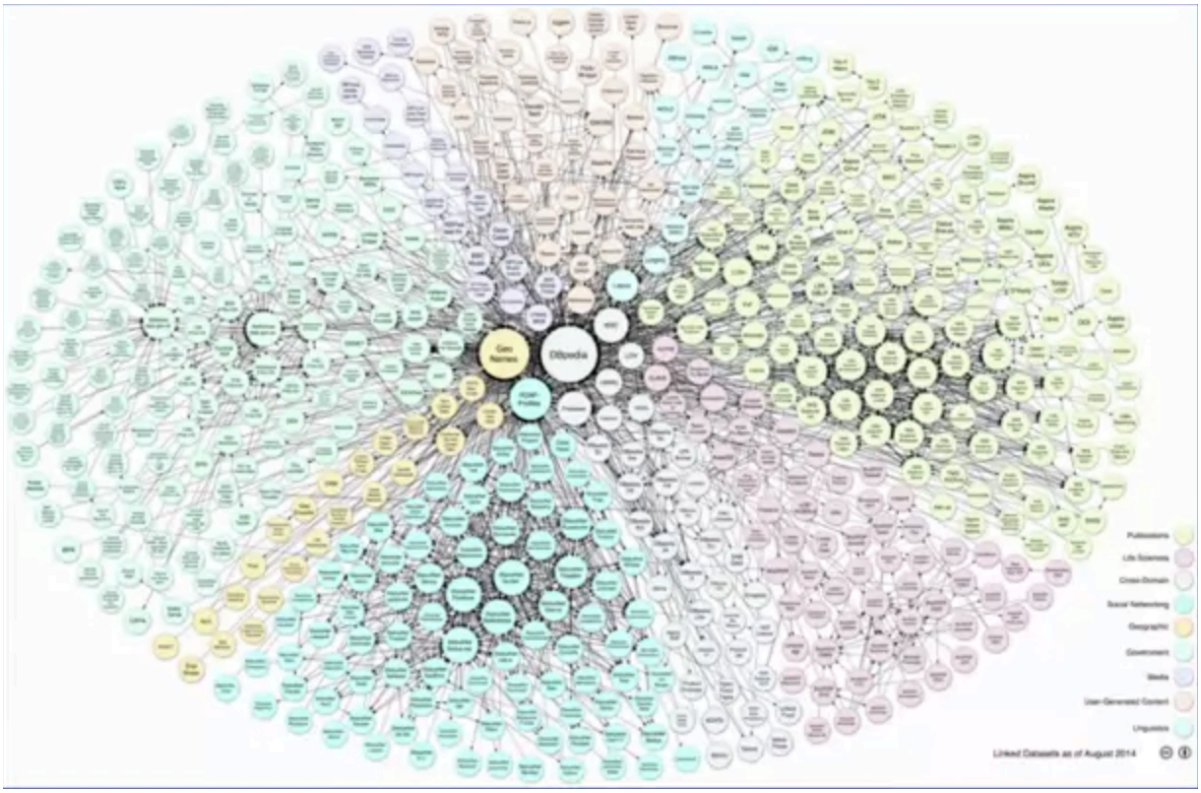
Hoje, as pessoas estão relatando reações e experiências que estão tendo. Estamos no Twitter, em sites de blog, grupos de suporte on-line, serviços de compartilhamento de dados on-line: estas são fontes de dados que nunca teve antes que podem ser usados para entender em uma taxa pessoal muito mais detalhada e o impacto das integrações de drogas são respostas a certos Se foram projetados para integrar registros médicos e hospitalares com informações sobre quando os medicamentos foram tomados e , em seguida, para aprofundar as mídias sociais ou coletar auto-relatos dos pacientes. Quem sabe que tipo de perguntas seremos capazes de responder? Ou novas perguntas que podemos fazer?

Material complementar

25 Fatos sobre Big Data: disponível em << https://www.slideshare.net/BernardMarr/big-data-25-facts/2-Every_2_dayswe_create_as >>

1.3. Características do Big Data - De onde vem o Big Data

O conceito de big data é que ele não é novo. A maioria das fontes de big data existia antes, mas a escala que usamos e as aplica hoje mudou. Basta olhar para esta imagem de dados de links abertos na Internet. Eu pensei que esta imagem era tão legal. Mostra não só que existem tantas fontes de dados, mas elas também estão conectadas.



O Big Data geralmente se resume a algumas variedades de dados gerados por máquinas, pessoas e organizações. Com dados gerados por máquina, nos referimos a dados gerados a partir de sensores em tempo real em máquinas industriais ou veículos que registram o comportamento do usuário online, sensores ambientais ou rastreadores de saúde pessoal e muitos outros recursos de dados de sentido.

O Grande Colisor de Hádrons gera 40 terabytes de dados a cada segundo durante experimentos. Mas dados gerados por humanos, nos referimos à grande quantidade de dados de mídia social, atualizações de status, tweets, fotos e mídias. Com dados gerados organizacionais nos referimos a tipos mais tradicionais de dados, incluindo informações de transações em bancos de dados e dados estruturados abertos armazenados em data warehouses.

Note que big data pode ser estruturado, semiestruturado ou não estruturado, que é um tópico sobre o qual falaremos mais tarde neste curso. Na maioria dos casos de uso comercial, qualquer fonte única de dados por conta própria não é útil. O valor real geralmente vem de combinar esses fluxos de fontes de big data entre si e analisá-los para gerar novos insights, que, em seguida, volta a ser big data em si. Uma vez que você tenha tais insights, ele então habilita o que chamamos de decisões e ações ativadas por dados. Vamos agora analisar esses diferentes tipos de big data com mais detalhes.

a. Big Data Gerado por Máquinas

Big data gerado por máquinas está em toda parte e há muito. Os grandes aviões exigem big data? Muito! Você sabia que um Boeing 787 produz meio terabyte de dados toda vez que voa? Realmente, quase todas as

partes do avião atualizam tanto o voo quanto a equipe de terra sobre seu status constantemente. De onde vem todos esses dados? Este é um exemplo de dados gerados por máquina provenientes de sensores.

Se você olhar para todas as fontes de big data, *machine data* é a maior fonte de big data. Além disso, é muito complexo. Em geral, chamamos máquinas que fornecem algum tipo de capacidade de detecção inteligentes. Você já se perguntou por que você chama seu celular de smartphone? Porque ele lhe dá uma maneira de rastrear muitas coisas, incluindo sua geolocalização, e conectá-lo a outras coisas.

Então, o que torna um dispositivo inteligente, inteligente? De modo geral, existem três propriedades principais de dispositivos inteligentes com base no que eles fazem com sensores e coisas que encapsulam. Eles podem se conectar a outros dispositivos ou redes, eles podem executar serviços e coletar dados de forma autônoma, o que significa por conta própria, eles têm algum conhecimento do ambiente. A disponibilidade generalizada dos dispositivos inteligentes e sua interconectividade levou a um novo termo a ser cunhado, A Internet das Coisas. Pense em um mundo de dispositivos inteligentes em casa, em seu carro, no escritório, cidade, áreas rurais remotas, o céu, até mesmo o oceano, todos conectados e todos gerando dados.

Vejamos um exemplo de um dispositivo que tem algumas dessas coisas nele. Um rastreador de atividades é um dispositivo ou aplicativo para monitoramento e rastreamento de métricas relacionadas ao fitness, como distância percorrida ou corrida, consumo de calorias e, em alguns casos, batimentos cardíacos e qualidade do sono. E se todos em Nova York usassem um rastreador de atividades? E se todos usassem vários? Não é tão incomum imaginar que este será o caso para muitas pessoas. Esses rastreadores de atividade permitem uma nova maneira de fazer a intervenção do paciente via medicina personalizada.

Da mesma forma, os sensores nos aviões geraram uma nova forma de olhar para a gestão de frotas e segurança de voo. Como resumo, as máquinas coletam dados 24 horas por dia, 7 dias por semana, via seus sensores integrados, tanto em balanças pessoais quanto industriais. E assim, eles são o maior de todas as fontes de big data.

Por que o Big Data gerado por máquinas é útil? Vamos voltar por um segundo para o nosso primeiro exemplo para os planos de big data gerados pela máquina. O que está produzindo todos esses dados no avião? Se você olhar para alguns dos sensores que contribuem para o meio terabyte de dados gerados em um avião, descobriremos que alguns deles provêm de acelerômetros que medem a turbulência. Há também sensores incorporados nos motores para temperatura, pressão, muitos outros fatores mensuráveis para detectar avarias do motor. A análise constante em tempo real de todos os dados coletados fornece ajuda monitoramento e detecção de problemas a 40.000 pés. Isso é aproximadamente 12.000 metros acima do solo. Chamamos esse tipo de processamento analítico *in situ*.

Antigamente, em sistemas tradicionais de gerenciamento de bancos de dados relacionais, os dados eram frequentemente movidos para o espaço computacional para processamento. No espaço de Big Data *In-Situ* significa trazer a computação para onde os dados estão localizados ou, neste caso, gerados. Um recurso chave desses tipos de notificações em tempo real é que elas habilitam ações em tempo real. No entanto, usar esse recurso exigiria que você abordasse seu aplicativo e seu trabalho de forma diferente. Se você estiver usando um rastreador de atividades, você provavelmente deve criar uma estratégia para como você irá incorporar o uso desses *gadgets* úteis em seu estilo de vida. Assim, se você está planejando incorporar insights orientados por Big Data em sua organização, você precisa definir uma nova estratégia e uma nova maneira de trabalhar.

A maioria das empresas centradas em Big Data atualizaram sua cultura para serem mais orientadas para ações em tempo real, refinando processos em tempo real para lidar com qualquer coisa, desde relações com clientes e detecção de fraudes, até monitoramento e controle do sistema. Além disso, esses volumes de dados em tempo real e operações analíticas que precisam de para ocorrer exigem um maior uso de sistemas de computação escaláveis, que precisam ser parte do planejamento de uma estratégia de Big Data organizacional.

Eles veem efeitos de tais mudanças também no sistema SCADA. SCADA significa Controle de Supervisão e Aquisição de Dados. SCADA é um tipo de sistema de controle industrial para monitoramento remoto e controle de processos industriais que existem no mundo físico, potencialmente incluindo vários locais, muitos tipos de sensores. Além de monitoramento e controle, o sistema SCADA pode ser usado para definir ações para redução de resíduos e melhoria da eficiência em processos industriais, incluindo os de fabricação e geração de energia, processos de infraestrutura pública ou privada, incluindo tratamento de água, petróleo e gasodutos e transmissão de energia elétrica e processos de instalação, incluindo edifícios, aeroportos, navios e estações espaciais. Eles podem até ser usados em aplicações de construção inteligente para monitorar e controlar sistemas de aquecimento, ventilação, ar condicionado como HVAC, acesso e consumo de energia.

Novamente, o gerenciamento desses processos uma vez que as tendências, padrões e anomalias são identificadas em tempo real precisa ser decidido no caso do Big Data. Como resumo, como o maior e mais rápido tipo de Big Data, os dados gerados por máquina podem habilitar de forma exclusiva ações em tempo real em muitos sistemas e processos. No entanto, uma mudança de cultura é necessária para sua computação e ação em tempo real.

b. Big Data Gerado por Pessoas - o Desafio não-estruturado

As pessoas estão gerando grandes quantidades de dados todos os dias através de suas atividades em vários sites de redes sociais como Facebook, Twitter e LinkedIn. Ou sites de compartilhamento de fotos online como Instagram, Flickr ou Picasa. E sites de compartilhamento de vídeo como o YouTube. Além disso, uma enorme quantidade de informações é gerada através de blogs e comentários, pesquisas na Internet, mais através de mensagens de texto, e-mail e através de documentos pessoais.

A maioria desses dados é pesada em texto e não estruturada, que não está em conformidade com um modelo de dados bem definido. Também podemos considerar esses dados como conteúdo com ocasionalmente alguma descrição anexada a ele. Essa atividade leva a um enorme crescimento de dados. Você sabia que em um único dia, os usuários do Facebook produzem mais dados do que bibliotecas de pesquisa acadêmica combinadas dos EUA? Vejamos alguns números de volume de dados diários semelhantes de algumas das maiores plataformas online. É incrível que alguns desses números estejam na faixa de petabytes para atividade diária. Um petabyte é mil terabytes. O tamanho total dos dados principalmente não estruturados gerados por humanos traz muitos desafios.

Company	Data Processed Daily
eBay	100 Petabytes (PB)
Google	100 PB
Facebook	30+ PB
Twitter	100 Terabytes(=.1PB)
Spotify	64 Terabytes

Dados não estruturados referem-se a dados que não estão em conformidade com um modelo de dados predefinido. Portanto, sem modelo de relação e sem SQL. É principalmente qualquer coisa que não armazenamos em um tradicional sistema de gerenciamento de banco de dados relacional . Considere um recibo de venda que você recebe de uma mercearia. Ele tem uma seção para uma data, uma seção para nome da loja, e uma seção para o valor total. Este é um exemplo de estrutura.

Humanos geram muitos dados não estruturados em forma de texto. Não existe um formato dado para isso. Veja todos os documentos que você escreveu com a mão até agora. Coletivamente, é um banco de dados não estruturados que você gerou pessoalmente. Na verdade, 80 a 90% de todos os dados em o mundo não é estruturado e esse número está crescendo rapidamente. Exemplos de dados não estruturados gerados por pessoas

incluem textos, imagens, vídeos, áudio, pesquisas na Internet e e-mails. Além de seu rápido crescimento, os principais desafios dos dados não estruturados incluem vários formatos de dados, como páginas da Web, imagens, PDFs, power point, XML e outros formatos que foram criados principalmente para consumo humano.

Pense nisso, embora eu possa classificar meu e-mail com data, remetente e assunto. Seria realmente difícil escrever um programa, categorizar todas as minhas mensagens de e-mail com base em seu conteúdo e organizá-las para mim de acordo outro desafio de dados gerados por humanos é o volume e geração rápida de dados, que é o que chamamos de velocidade. Basta ter um momento para estudar este gráfico informativo, e observar o que acontece em um minuto na internet, e considerar o quanto contribuir para isso. Além disso, a confirmação de dados não estruturados geralmente é demorada e dispendiosa.

Os custos e o tempo do processo de aquisição, armazenamento, limpeza, recuperação e processamento de dados não estruturados podem somar bastante e investimento antes que possamos começar a tirar valor desse processo. Pode ser muito difícil encontrar as ferramentas e pessoas para implementar tal processo e colher valor no final. Como resumo, embora haja uma enorme quantidade de dados gerados por pessoas, a maioria desses dados é desestruturada. Os desafios de trabalhar com dados não estruturados não devem ser considerados de ânimo leve. Em seguida, analisaremos como as empresas estão enfrentando esses desafios para obter insights. E assim, valor fora de trabalhar com as pessoas gerou dados.

Como Big Data gerado por Pessoas Está Sendo Usado?

Agora vamos analisar algumas das tecnologias emergentes para enfrentar esses desafios. E veja alguns exemplos que transformam dados não estruturados em insights valiosos. Embora dados não estruturados especialmente o tipo gerado por pessoas tem uma série de desafios. A boa notícia é que a cultura empresarial de hoje está mudando para enfrentar esses desafios e aproveitar ao máximo esses dados. Como se costuma dizer, um desafio é uma oportunidade perfeita. Este é certamente o caso do big data e esses desafios criaram uma indústria tecnológica própria. Esta indústria é principalmente centrada ou como diríamos, em camadas ou empilhadas, em torno de algumas estruturas fundamentais de big data de código aberto.

As ferramentas de big data precisam são projetadas do zero para gerenciar informações não estruturadas e analisá-las. A maioria dessas ferramentas é baseada em um framework de big data de código aberto chamado Hadoop. Hadoop foi projetado para suportar o processamento de grandes conjuntos de dados em um ambiente de computação distribuída. Esta definição já lhe daria uma dica de que aborda o primeiro desafio. Ou seja, o volume de informações não estruturadas. Hadoop pode lidar com grandes lotes de informações distribuídas, mas na maioria das vezes há uma necessidade de um processamento em tempo real de dados gerados por pessoas como atualizações do Twitter ou Facebook.

O monitoramento de conformidade financeira é outra área de nosso processamento central de tempo é necessário, em particular para reduzir os dados do mercado. Mídias sociais e dados de mercado são dois tipos do que chamamos de dados de alta velocidade. Storm e Spark são dois outros frameworks de código aberto que lidam com esses dados em tempo real gerados a uma taxa rápida. O Storm e o Spark podem integrar dados com qualquer tecnologia de armazenamento de dados ou banco de dados. Como já enfatizamos antes dados não estruturados não tem um modelo de dados relacional, então geralmente não cabe no modelo tradicional de data warehouse baseado em bancos de dados relacionais.

Data warehouses são repositórios centrais de dados integrados de uma ou mais fontes. Os dados que são armazenados em armazéns, são extraídos de várias fontes. Ele é transformado em uma forma estruturada comum e pode retardar isso no banco de dados central para uso por trabalhadores criando relatórios analíticos em toda a empresa. Esse processo Extract Transform Load é comumente chamado de ETL. Essa abordagem era bastante padrão em sistemas de dados corporativos até recentemente. Como você provavelmente notou, ele é bastante estático e não se encaixa bem com o mundo dinâmico de big data de hoje.

Então, como as empresas de hoje resolvem esse problema? Muitas empresas atualmente estão usando uma abordagem híbrida na qual seus dados estruturados menores permanecem em seus bancos de dados relacionais, e grandes conjuntos de dados não estruturados são armazenados em bancos de dados NoSQL na nuvem.

As tecnologias de dados NoSQL são baseadas em conceitos não relacionais e fornecem opções de armazenamento de dados tipicamente em nuvens de computação além das casas de taxa centradas de bancos de dados relacionais tradicionais. A principal vantagem de usar soluções NoSQL é sua capacidade de organizar os dados para acesso escalável para se adequar ao problema e objetivos relativos a como os dados serão usados. Por exemplo, se os dados serão usados em uma análise para encontrar conexões entre conjuntos de dados, então a melhor solução é um banco de dados gráfico.

Neo4j é um exemplo de um banco de dados gráfico. Graph networks é um tópico de interesse deste curso, vamos explicar em profundidade. Se os dados serão melhor acessados usando pares de valor de chave como um cenário de mecanismo de pesquisa, a melhor solução é provavelmente um banco de dados emparelhado de valor de chave dedicado. Redis é um exemplo de um banco de dados de valor chave. Estes e muitos outros tipos de sistemas NoSQL serão explicados mais adiante no curso.

Portanto, estamos confiantes de que existem tecnologias emergentes para desafios individuais para gerenciar dados não estruturados gerados por pessoas. Mas como se aproveita destes para gerar valor? Como vimos, big data deve passar por uma série de etapas antes de gerar valor. Ou seja, acesso a dados, armazenamento, limpeza e análise.

Uma abordagem para resolver esse problema é executar cada estágio como uma camada diferente. E use ferramentas disponíveis para ajustar o problema em questão, e dimensionar soluções analíticas para big data. Veremos ferramentas importantes que você pode usar para resolver seus problemas de big data além dos que você já viu.

Agora vamos dar um passo atrás e lembrar a nós mesmos qual era o valor. Lembre-se de como as empresas podem ouvir a voz real dos clientes usando big data? É este tipo de dados gerados que o habilitou. Análise de sentimento analisa mídias sociais e outros dados para encontrar se as pessoas se associam de forma positiva ou negativa aos seus negócios. As organizações estão utilizando o processamento de dados pessoais para entender as verdadeiras preferências de seus clientes. Agora vamos fazer um teste divertido para adivinhar quanto as empresas de dados do Twitter analisam todos os dias para medir o sentimento em torno de seus produtos. A resposta é 12 terabytes por dia. Para comparação, você precisaria ouvir continuamente por dois anos para terminar de ouvir 1 terabyte de música.

Outro exemplo de área de aplicação para dados gerados por pessoas é a modelagem e previsão do comportamento do cliente. Amazon, Netflix e muitas outras organizações, usam análises para analisar as preferências de seus clientes. Com base no comportamento do consumidor, as organizações sugerem melhores produtos aos clientes, e, por sua vez, têm clientes mais felizes e lucros maiores. Outra área de aplicação onde o valor vem na forma de impacto social e bem-estar social, é a gestão de desastres.

Como você viu no meu exemplo de incêndio selvagem, há muitos tipos de big data que podem ajudar na resposta a desastres. Dados na forma de fotos e tweets, ajuda a facilitar uma resposta coletiva a situações de desastre, como evacuações através a rota mais segura com base no feedback da comunidade através de mídias sociais. Há também redes que transformam o fornecimento de multidões e a análise de big data em ferramentas coletivas de resposta a desastres. A Rede Internacional de Mapeadores de Crise é a maior dessas redes e inclui uma comunidade internacional ativa de voluntários. Mapeadores de crise usam big data na forma de imagens aéreas e de satélite, mapas participativos e atualizações ao vivo do Twitter para analisar os dados usando plataformas geoespaciais, visualização avançada, simulação ao vivo de e modelos computacionais e estatísticos. Uma vez analisados os resultados são relatados para agências de resposta rápida e humanitárias na forma de aplicativos móveis e web.

Em 2015, logo após o terremoto do Nepal Crises Mappers fonte a análise de tweets e mídia convencional para acessar rapidamente os danos de desastres e necessidades e identificar onde a ajuda humanitária é necessária. Este exemplo é incrível e mostra como o big data pode ter enormes impactos para bem-estar social em momentos de necessidade. Você pode aprender mais sobre esta história no link a seguir. Como resumo, embora existam desafios ao trabalhar com pessoas não estruturadas geraram dados em uma escala e velocidade que os aplicativos exigem. Há também tecnologias e soluções emergentes que estão sendo usadas por muitos aplicativos para gerar valor a partir da rica fonte de informações.

c. Big Data Gerado por Organizações - Estruturado mas, muitas vezes, isolado

O último tipo de big data que discutiremos é big data gerado pelas organizações. Este tipo de dados é o mais próximo do que a maioria das empresas tem atualmente. Mas é considerado um pouco fora de moda, ou tradicional, em comparação com outros tipos de big data. No entanto, é pelo menos tão importante quanto outros tipos de big data.

Então, como as organizações produzem dados? A resposta para como uma organização gera dados é muito exclusiva para a organização e contexto. Cada organização tem práticas de operação distintas e modelos de negócios, que resultam em uma variedade de plataformas de geração de dados. Por exemplo, o tipo e a fonte de dados que um banco obtém é muito diferente de o que um fabricante de equipamentos de hardware recebe. Alguns tipos comuns de big data organizacional vêm de transações comerciais, cartões de crédito, instituições governamentais, e-commerce, bancos ou registros de ações, registros médicos, sensores, transações, cliques e assim por diante. Quase todos os eventos podem ser potencialmente armazenados.

As organizações armazenam esses dados para uso atual e futuro, bem como para análise do passado. Digamos que você seja uma organização que coleta transações de vendas. Você pode usar esses dados para reconhecimento de padrões para detectar produtos correlacionados, para estimar a demanda por produtos com probabilidade de aumentar em vendas e capturar atividades fraudulentas. Além disso, quando você conhece seu registro de vendas e pode correlacioná-lo com seus registros de marketing, você pode descobrir quais campanhas realmente causaram impacto. Você já está se tornando uma organização experiente em dados.

Agora pense em juntar seus dados de vendas com outros dados públicos abertos, como os principais eventos mundiais nas notícias. Você pode perguntar, foi marketing experiente ou uma consequência de eventos externos que desencadearam suas vendas? Usando análises adequadas, agora você pode criar inventários para corresponder ao crescimento e à demanda previstos. Além disso, as organizações criam e aplicam processos para registrar e monitoram eventos comerciais de interesse, como registrar um cliente, fabricar um produto ou fazer um pedido.

Esses processos coletam dados altamente estruturados que incluem transações, tabelas de referência e relacionamentos , bem como os metadados que definem seu contexto. Normalmente, os dados estruturados são armazenados em sistemas de gerenciamento de bancos de dados relacionais. No entanto, chamamos qualquer dado que esteja na forma de um registro localizado em um campo fixo ou dados estruturados de arquivo. Esta definição também inclui planilhas.

Como já mencionado, tradicionalmente esse tipo de dados altamente estruturados é a grande maioria do que a TI gerenciou e processou tanto em sistemas operacionais quanto de business intelligence. Vejamos os dados da transação de vendas em nosso exemplo anterior. Se você olhar para os dados na tabela relacional à direita. Como o nome estruturado sugere, a tabela é organizada para armazenar dados usando uma estrutura definida por um modelo. Cada coluna é marcada para nos dizer quais dados essa coluna pretende armazenar. Isto é o que chamamos de modelo de dados.

Um modelo de dados define cada uma dessas colunas e campos na tabela e define relacionamentos entre eles. Se você olhar para a coluna ID do produto, verá que ela inclui apenas identificadores que podem

potencialmente ser vinculados a outra tabela que define esses produtos. A capacidade de definir tais relações, facilmente fazer dados estruturados, ou neste caso bancos de dados relacionais, altamente adotado por muitas organizações. Existem linguagens comumente usadas como SQL, o Structured Query Language, para extrair dados de interesse dessas tabelas. Isso é referido como consultando os dados. No entanto, pode até ser um desafio integrar esses dados estruturados. Esta imagem nos mostra um contínuo de tecnologias para modelar, coletar e consultar dados não estruturados provenientes de componentes de software e de hardware dentro de uma organização.

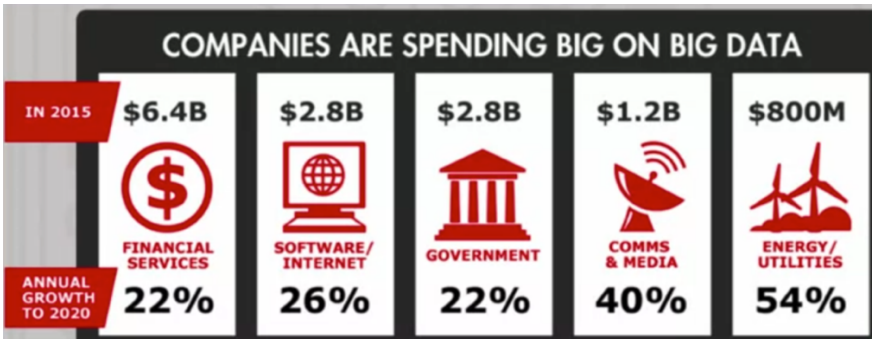
No passado, tais desafios levaram a que as informações fossem armazenadas no que chamamos de silos, mesmo dentro de uma organização. Muitas organizações tradicionalmente capturaram dados no nível do departamento, sem infraestrutura e política adequadas para compartilhar e integrar esses dados. Isso dificultou o crescimento do reconhecimento de padrões escaláveis para os benefícios de toda a organização. Porque nenhum sistema tem acesso a todos os dados que a organização possui. Cada conjunto de dados é compartimentalizado.

Se esses silos forem deixados intocados, as organizações correm o risco de ter descontextualizados, não sincronizado, e até mesmo conjuntos de dados invisíveis. As organizações estão percebendo os resultados prejudiciais desta rígida estrutura . E alterar políticas e infraestrutura para permitir o processamento integrado de todos os dados para benefício de toda a organização. As soluções baseadas em nuvem são vistas como soluções ágeis e de baixo consumo de capital nesta área. Como resumo, enquanto os dados organizacionais altamente estruturados são muito úteis e confiáveis, e, portanto, uma valiosa fonte de informação, organizações devem prestar especial atenção à separação dos silos de informação para fazer pleno uso de seu potencial.

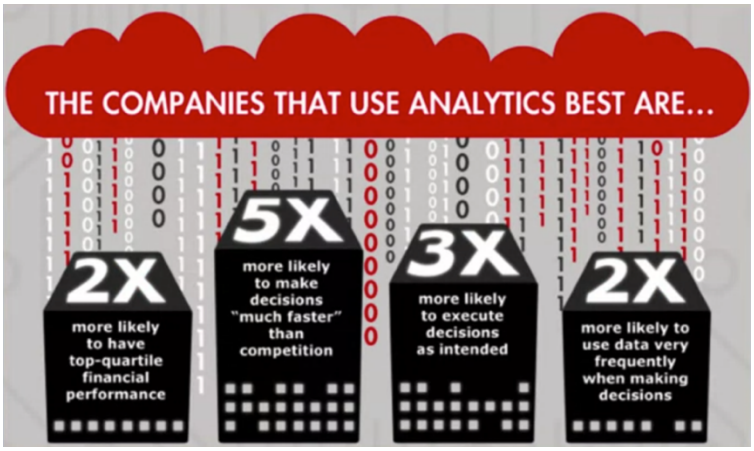
Dados gerados pela organização: Os benefícios vêm da combinação com outros tipos de dados.

Como algumas organizações se beneficiam de big data? Vamos olhar exemplos do mundo real para ver as vantagens que essas organizações estão tirando de big data. Uma dessas empresas é a UPS. A UPS entrega 16 milhões de entregas por dia. Eles recebem cerca de 40 milhões solicitações de rastreamento. Isto é enorme. Uma estimativa de quantos dados a UPS possui em suas operações é de 16 petabytes. Você consegue adivinhar quanto dinheiro a UPS pode economizar, reduzindo a rota de cada motorista por apenas uma milha? Se eles podem reduzir a distância percorrida por cada caminhão em até uma milha, UPS pode salvar um gritante $50 milhões por ano. É aqui que entra big data. Utilizando otimização complexa em grandes conjuntos de dados pode levar a otimizações de rota que não eram visíveis antes para a empresa. Big data, juntamente com processamento inteligente, permite a UPS para gerir milhares de otimizações de rota.

Vamos viajar da entrega do pacote ao domínio do varejo. Uma organização da área de compras no varejo que utiliza muito big data é o Walmart. O Walmart é uma grande organização que recebe 250 milhões de clientes em 10.000 lojas. Você sabia que eles coletam 2,5 petabytes de dados por hora? Eles coletam dados em tweets do Twitter, eventos locais, clima local, compras na loja, cliques on-line e muitos outros dados relacionados a vendas, clientes e produtos. Eles usam esses dados para encontrar padrões, como quais produtos são frequentemente comprados juntos, e qual é o melhor produto novo a introduzir em suas lojas, prever a demanda em no local específico, e para personalizar as recomendações do cliente. No geral, aproveitando o big data e análise, o Walmart manteve sua posição como um dos principais varejistas.



Os exemplos de UPS e Walmart foram apenas dois dos em um número de empresas que usam big data. O big data está produzindo resultados para empresas em todos os setores. Estudos preveem gastos com tecnologias de big data subindo drasticamente nos próximos cinco anos. Um estudo de Bane e Company sugere que os primeiros a adotar big data ganharam uma vantagem significativa do resto do mundo corporativo. Nos gráficos referenciados aqui, vemos que as empresas que usam análises têm o dobro da probabilidade estar no quartil superior do desempenho financeiro em seus setores. Cinco vezes mais chances de tomar decisões muito mais rapidamente que os pares do mercado. Três vezes mais probabilidade de executar decisões conforme o planejado. E duas vezes mais probabilidade de usar dados com muita frequência ao tomar decisões. Isso aponta para o crescimento e a demanda de pessoas e tecnologia centrada em torno de ou especializada em aplicações de big data. Em resumo, as organizações estão obtendo benefício significativo ao integrar práticas de big data em sua cultura e quebrando seus silos. Alguns dos principais benefícios para organizações são a eficiência operacional, melhor marketing resultados, lucros mais altos e maior satisfação do cliente.



1.4. Integração de Dados - A Chave do Sucesso

Seja qual for o seu aplicativo de big data, e os tipos de big data que você está usando o valor real virão da integração diferentes tipos de fontes de dados e da análise em escala. Então, como começamos a obter esse valor? Às vezes, basta, está olhando para os dados que você já coleta de uma maneira diferente. E isso pode significar uma grande diferença no seu retorno sobre o investimento. Esta nova história de junho de 2015 menciona que o Carnival Cruises está usando dados estruturados e não estruturados de uma variedade de fontes.

Carnival transforma em lucro usando técnicas de otimização de preços nos dados integrados. Para que você consiga essa história de sucesso, você precisará incluir a integração de dados em sua prática de big data. No entanto, existem alguns desafios únicos ao tentar integrar essas diversas fontes de dados e dimensionar as soluções.

Vamos tirar um momento para definir por que efeito da integração de dados é útil? Integração de dados significa reunir dados de diversas fontes e transformando-os em informações coerentes e mais úteis. Também

chamamos isso de conhecimento. O objetivo principal aqui é domar ou gerenciar mais tecnicamente dados e transformá-los em algo que você pode fazer uso de programaticamente. Um processo de integração de dados envolve muitas partes. Começa com a descoberta, o acesso e o monitoramento de dados e continua com a modelagem e a transformação de dados de uma variedade de fontes.

Mas por que precisamos de integração de dados em primeiro lugar? Vamos começar focando nas diferenças entre grandes conjuntos de dados provenientes de diferentes fontes. Você pode ter dados formatados de arquivo simples, dados de banco de dados relacional, dados de codificados em XML ou JSON, ambos comuns para dados gerados pela Internet. Estes diferentes formatos e modelos são úteis porque eles são projetados para expressar dados diferentes de maneiras únicas. De certa forma, diferentes formatos de dados e modelos tornam o big data mais útil e mais desafiador ao mesmo tempo. Ao integrar dados em formatos diferentes, você torna o produto final mais rico no número de recursos com os quais descrevem os dados. Por exemplo, integrando os dados do sensor ambiental e da câmera com dados do sistema de informação geográfica, como no meu aplicativo de previsão de fogo selvagem, posso usar os recursos de dados espaciais com dados não espaciais para executar simulações de incêndio com mais precisão. No passado, embora pudéssemos ver as imagens do fogo de câmeras do topo da montanha, assim como esta imagem, ainda éramos não capazes de dizer qual é a localização exata do incêndio automaticamente. Agora, quando um incêndio é detectado de uma câmera no topo da montanha, viewsheds são usados para estimar a localização do incêndio. Esta informação de localização pode ser introduzida no simulador de incêndio logo que seja detectada para prever o tamanho e a localização do incêndio na próxima hora com mais precisão e rapidez. Da mesma forma, eu posso usar dados em tempo real com conjuntos de dados de curva ocular, e usá-los todos juntos. Além disso, reunindo os dados e fornecendo acesso programável a eles, estou tornando cada conjunto de dados mais acessível. A integração de diversos conjuntos de dados reduz significativamente a complexidade geral dos dados no meu produto orientado por dados. Os dados se tornam mais disponíveis para uso e unificados como um sistema próprio.

Uma vantagem de tal integração não é mencionada frequentemente. Um sistema de dados simplificado e integrado pode aumentar a colaboração entre diferentes partes de seus sistemas de dados. Cada parte agora pode ver claramente como seus dados são integrados ao sistema geral. Incluindo os cenários do usuário e os processos de segurança e privacidade em torno dele. Em geral, ao integrar diversos fluxos de dados, você agrega valor ao seu big data e melhora o seu negócio mesmo antes de começar a analisá-lo. Em seguida, vamos nos concentrar nas dimensões da escalabilidade e discutir como podemos começar a enfrentar alguns desses desafios.

1.5. Os cinco Vs do Big Data

a. Volume

Volume é a dimensão de big data que se relaciona com o tamanho total do big data. Esse volume pode vir de grandes conjuntos de dados sendo compartilhados ou muitos pequenos pedaços de dados e eventos sendo coletados ao longo do tempo. A cada minuto são enviados 204 milhões de e-mails, 200.000 fotos são carregadas e 1,8 milhões de curtidas são gerados no Facebook. No YouTube, 1,3 milhão de vídeos são visualizados e 72 horas de vídeo são carregados.

Mas de quantos dados estamos falando? O tamanho e a escala de armazenamento para big data podem ser enormes. Você me ouviu dizer palavras que começam com peta, exa e yotta, para definir o tamanho, mas o que isso realmente significa? Para comparação, 100 megabytes terão algumas enciclopédias. Um DVD tem cerca de 5 GB e 1 TB teria cerca de 300 horas de vídeo de boa qualidade. Um negócio orientado a dados atualmente coleta dados na ordem de terabytes, mas petabytes estão se tornando mais comuns em nossas vidas diárias. O grande colisor de hádrons do CERN gera 15 petabytes por ano.

De acordo com as previsões de um relatório da IDC patrocinado por uma empresa de big data chamada EMC, dados digitais, crescerá em um fator de 44 até 2020. Este é um crescimento de 0,8 zetabytes, Em 2009 para 35,2 zettabytes em 2020. Um zettabyte é 1 trilhão de gigabytes, isso é 10^{21}. Os efeitos disso serão enormes! Pense em todo o tempo, custo, energia que será usado para armazenar e fazer sentido de tal quantidade de dados. A próxima era será yottabytes. 10^{24} e brontobytes, 10^{27}.

O que é realmente difícil de imaginar para a maioria de nós neste momento. Isso também é o que chamamos de dados em escala astronômica. A escolha de colocar a Via Láctea Galaxy no meio do círculo não é apenas para a estética. Isto é o que veríamos se fôssemos escalar 10^{21} vezes para o universo. Legal, não é? Por favor, consulte a leitura neste módulo chamado, o que significa escala astronômica, para um bom vídeo sobre os poderes de dez. Todos esses apontam para um crescimento exponencial no volume e no armazenamento de dados.

Qual é a relevância dessa quantidade de dados em nosso mundo? Lembra-se dos aviões coletando grandes dados? Nossa esperança, como passageiros, é que mais dados signifique melhor segurança de voo. A ideia é entender que empresas e organizações estão coletando e aproveitando grandes volumes de dados para melhorar seus produtos finais, seja segurança, confiabilidade, saúde ou governança. Em geral, nos negócios o objetivo é transformar esse número de dados em alguma forma de vantagem comercial.

A questão é como utilizamos volumes maiores de dados para melhorar a qualidade do nosso produto final? Apesar de uma série de desafios relacionados a ele. Há uma série de desafios relacionados aos grandes volumes de big data. O mais óbvio é, naturalmente, o armazenamento. À medida que o tamanho dos dados aumenta, faz a quantidade de espaço de armazenamento necessário para armazenar esses dados de forma eficiente. No entanto, também precisamos ser capazes de recuperar essa grande quantidade de dados rápido o suficiente, e movê-los para unidades de processamento em tempo hábil para obter resultados quando precisarmos deles.

Isso traz desafios adicionais, como rede, largura de banda, custo de armazenamento de dados. Armazenamento interno versus armazenamento em nuvem e coisas assim. Desafios adicionais surgem durante o processamento de dados tão grandes. A maioria dos métodos analíticos existentes não será dimensionada para tais somas de dados em termos de memória, processamento ou necessidades de E/S. Isso significa que o desempenho deles diminuirá. Você pode obter um bom desempenho para dados de centenas de clientes. Mas que tal dimensionar sua solução para 1.000 ou 10.000 clientes?

À medida que o volume aumenta o desempenho e o custo começam a tornar-se um desafio. As empresas precisam de uma estratégia holística para lidar com o processamento de dados em grande escala para seu benefício da maneira mais econômica. Avaliando as opções nas dimensões mencionadas aqui, é o primeiro passo quando se trata de aumentar continuamente o tamanho dos dados. Como um resumo, volume é a dimensão de big data relacionada ao seu tamanho e seu crescimento exponencial. Os desafios de trabalhar com volumes de big data incluem custo, escalabilidade, e desempenho relacionados ao armazenamento, acesso e processamento.

b. Variedade

Agora vamos falar sobre uma forma de escalabilidade chamada variedade. Neste caso, a escala não se refere à grandeza dos dados. Refere-se ao aumento da diversidade. Aqui está um mantra importante que você precisa pensar. Quando nós, como cientistas de dados, pensamos na variedade de dados, pensamos na complexidade de adicional que resulta de mais tipos de dados que precisamos armazenar, processar e combinar.

Antigamente, sempre pensávamos nos dados como tabelas. Essas tabelas podem estar em planilhas ou bancos de dados ou apenas arquivos, mas de alguma forma elas serão modeladas e manipuladas como linhas e colunas de tabelas. Agora, as tabelas ainda são realmente importantes e dominantes, no entanto, hoje uma

variedade muito maior de dados são coletados, armazenados e analisados para resolver problemas do mundo real.

Dados de imagem, dados de texto, dados de rede, mapas geográficos, simulações geradas por computador são apenas alguns dos tipos de dados que encontramos todos os dias. A heterogeneidade dos dados pode ser caracterizada ao longo de várias dimensões. Nós mencionamos quatro desses eixos aqui:

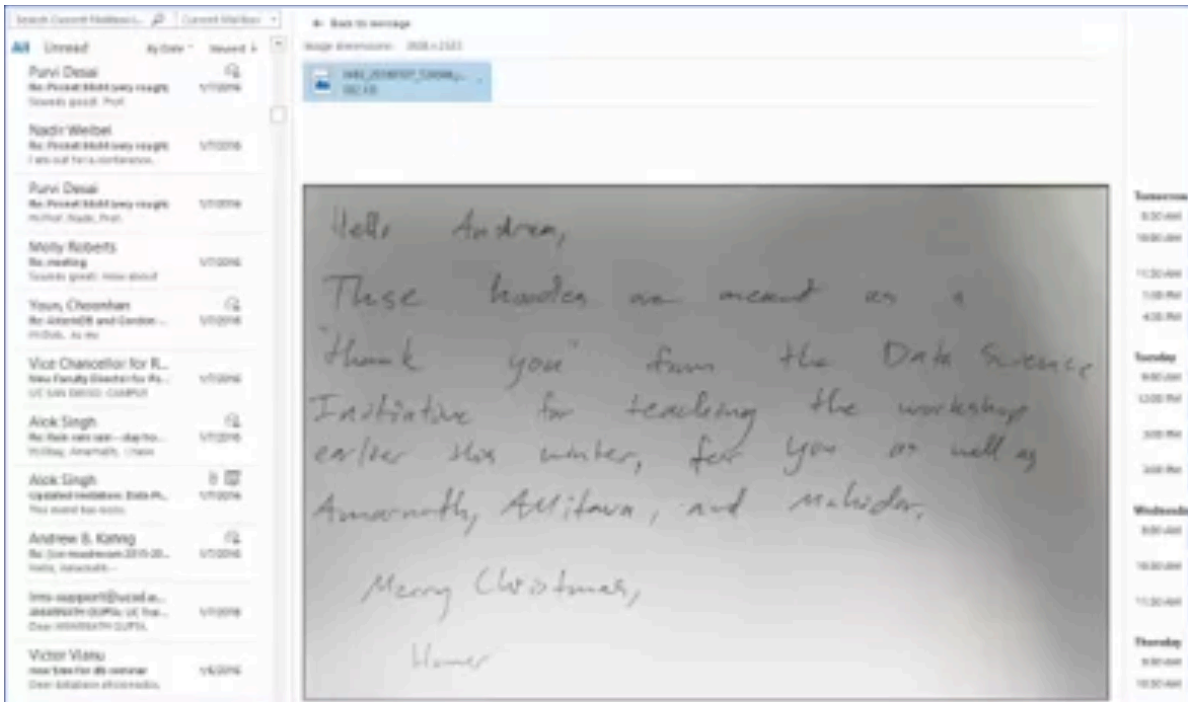
Variedade estrutural refere-se à diferença em a representação dos dados. Por exemplo, um sinal de eletrocardiograma é muito diferente de um artigo de jornal. Uma imagem de satélite de incêndios florestais da NASA é muito diferente dos tweets enviados por pessoas que estão vendo o fogo se espalhar.

Variedade de mídia refere-se ao meio em que os dados são entregues. O áudio de um discurso versus a transcrição do discurso pode representar a mesma informação em duas mídias diferentes. Objetos de dados, como vídeo de notícias, podem ter várias mídias. Uma sequência de imagens, um áudio e texto com legendas ocultas, todos os tempos sincronizados entre si.

A variedade semântica é mais bem descrita dois exemplos. Muitas vezes usamos diferentes unidades para quantidades que medimos. Às vezes também usamos medidas qualitativas versus quantitativas. Por exemplo, a idade pode ser um número ou representamos por termos como infantil, juvenil ou adulto. Outro tipo de variedade semântica vem de diferentes suposições de condições nos dados. Por exemplo, se realizarmos duas pesquisas sobre a renda em dois grupos diferentes de pessoas, podemos não ser capazes de comparar ou combiná-los sem saber mais sobre as próprias populações.

A variação e a disponibilidade assumem muitas formas. Para começar, os dados podem estar disponíveis em tempo real, como dados do sensor, ou podem ser armazenados, como registros de pacientes. Da mesma forma, os dados podem ser acessíveis continuamente, por exemplo a partir de uma câmera de tráfego. Versus intermitentemente, por exemplo , somente quando o satélite está sobre a região de interesse. Isso faz a diferença entre as operações que se pode fazer com os dados, especialmente se o volume dos dados for grande.

Não devemos pensar que um único objeto de dados, ou uma coleção de objetos de dados semelhantes, serão todos uniformes em si mesmos. E-mails, por exemplo, é uma entidade híbrida. Algumas dessas informações podem ser uma tabela, como mostrado aqui. Agora, o corpo do e-mail geralmente tem texto nele. No entanto,



alguns dos textos podem ter ornamentos em torno deles. Por exemplo, a parte destacada em amarelo representa algo chamado marcação no texto. E-mails contêm anexos. Estes são arquivos, ou imagens incorporadas, ou outros objetos multimídia que o mailer permite. Esta captura de tela do meu Outlook mostra a imagem de uma imagem digitalizada de uma nota manuscrita. Quando você recebe uma coleção de todos os emails de sua caixa de correio, ou que de uma organização, você verá que os remetentes e os receptores formam uma rede de comunicação.

Em 2001, houve um escândalo famoso em torno de uma empresa chamada Enron que se envolveu em práticas fraudulentas de relatórios financeiros. Sua rede de e-mail, parcialmente mostrada aqui, foi estudada pelo cientista de dados para encontrar padrões usuais e incomuns de conexões entre as pessoas na organização. Uma coleção de e-mails também pode ter sua própria semântica. Por exemplo, um e-mail não pode se referir, isso significa que não pode copiar ou encaminhar, um e-mail anterior. Finalmente, um servidor de e-mail é uma fonte de dados em tempo real. Mas um repositório de e-mail não é. Os e-mails e coleções de e-mails demonstram significativa variação interna na estrutura, mídia, semântica e disponibilidade?

Velocidade

Velocidade refere-se à velocidade crescente na qual big data é criado e à velocidade crescente na qual os dados precisam ser armazenados e analisados. O processamento de dados em tempo real para corresponder à sua taxa de produção à medida que é gerado é um objetivo específico da análise de big data. Por exemplo, esse tipo de recurso permite a personalização de anúncios nas páginas da Web que você visita com base em seu histórico recente de pesquisa, visualização e compras.

Se uma empresa não pode tirar proveito dos dados à medida que são gerados, ou na análise de velocidade deles é necessária, muitas vezes eles perdem oportunidades. A fim de construir um argumento para a importância desta dimensão de big data, vamos imaginar que estamos fazendo uma viagem. Você está procurando algumas informações melhores para começar a preparar a mala. Neste caso, quanto mais recente for a informação, maior será a sua relevância na decisão do que embalar. Você usaria informações meteorológicas do mês passado ou dados do ano passado neste momento? Ou, você usaria as informações meteorológicas desta semana, ontem ou melhor, hoje? Faz sentido obter as informações mais recentes sobre o tempo e processá-las de forma a facilitar as suas decisões. Se as informações forem antigas, não importa o quão precisas sejam.

Ser capaz de acompanhar a velocidade de big data e analisá-lo à medida que ele é gerado pode até impactar a qualidade de vida humana. Sensores e dispositivos inteligentes que monitoram o corpo humano podem detectar anormalidades em tempo real e desencadear ação imediata, potencialmente salvando vidas. Este tipo de processamento é o que chamamos de processamento em tempo real. O processamento em tempo real é bastante diferente do seu relativo remoto, o processamento em lote.

Processamento em lote era a norma até alguns anos atrás. Grandes quantidades de dados seriam alimentadas em máquinas grandes e processadas por dias de cada vez. Embora esse tipo de processamento ainda seja muito comum hoje, decisões baseadas em informações que têm poucos dias de idade podem ser catastróficas para algumas empresas.

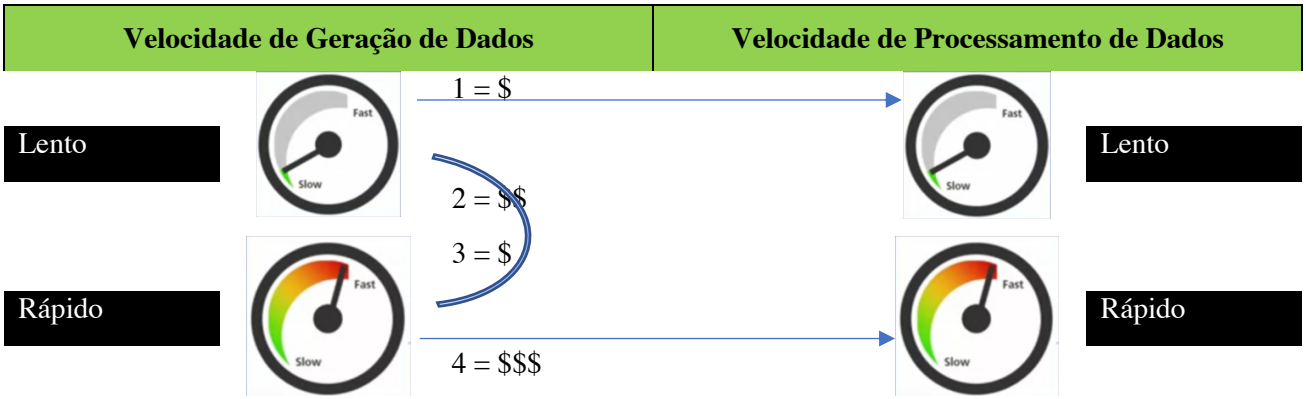
As organizações que tomam decisões sobre os dados mais recentes são mais propensas a atingir o alvo. Por esta razão, é importante combinar a velocidade de processamento com a velocidade de geração de informações e obter poder de decisão em tempo real. Além disso, o clima socioeconômico de hoje em dia requer decisões mais rápidas. Assim, não podemos esperar que todos os dados sejam produzidos pela primeira vez, então alimentados em uma máquina. Há muitas aplicações onde novas informações são streaming e precisa ser integrado com dados existentes para produzir decisões como planejamento de resposta de emergência em um tornado, ou decidir estratégias de negociação em tempo real, ou obter estimativas em publicidade. Temos que digerir pedaços de dados à medida que são produzidos e dar resultados significativos. À medida que mais dados

chegam, seus resultados precisarão se adaptar para refletir essa alteração na entrada. Decisões baseadas no processamento de dados já adquiridos, tais como processamento em lote, podem dar uma imagem incompleta. E, portanto, os aplicativos precisam de status em tempo real do contexto em mãos. Ou seja, análise de streaming.

Felizmente, com o evento de tecnologia de sensores baratos, celulares e mídias sociais, podemos obter as últimas informações a uma taxa muito rápida e em tempo real em comparação com o passado. Então, como você garante que correspondamos à velocidade das expectativas para obter insights de big data? Com a velocidade de big data. A taxa de geração, recuperação, ou processamento de dados é específica para o aplicativo. A necessidade de ações baseadas em dados em tempo real dentro de um business case é o que, no final, dita a velocidade da análise em relação a big data. Às vezes é necessária precisão de um minuto. Às vezes, meio dia.

Vamos olhar para esses quatro caminhos e discutir quando escolher o caminho certo para sua análise. Os sinais de dólar ao lado dos números neste exemplo indicam quão caro a operação é . Quanto mais dólares, maior o custo. Quando a pontualidade das informações processadas não desempenha nenhum papel na tomada de decisões, a velocidade com que os dados são gerados torna-se irrelevante. Em outras palavras, você pode aguardar o tempo necessário para processar dados. Dias, meses, semanas. E uma vez terminado o processamento, você vai olhar para os resultados e provavelmente compartilhá-los com alguém.

Quando a pontualidade não é um problema, você pode escolher qualquer um dos quatro caminhos. Você provavelmente escolherá o mais barato. Quando a pontualidade do resultado final é um problema decidir qual dos quatro caminhos escolher não é tão simples. Você terá que tomar uma decisão com base no custo do hardware, sensibilidade de tempo das informações, cenários futuros.



Em outras palavras, isso se torna uma questão orientada para os negócios. Por exemplo, se a velocidade for realmente importante a todo custo, você escolherá o caminho quatro. Como resumo, precisamos prestar atenção à velocidade de big data. Streaming data fornece informações sobre o que está acontecendo agora. Os dados de streaming têm velocidade, o que significa que são gerados a várias taxas. E a análise desses dados em tempo real dá agilidade e adaptabilidade para maximizar os benefícios que você deseja extrair.

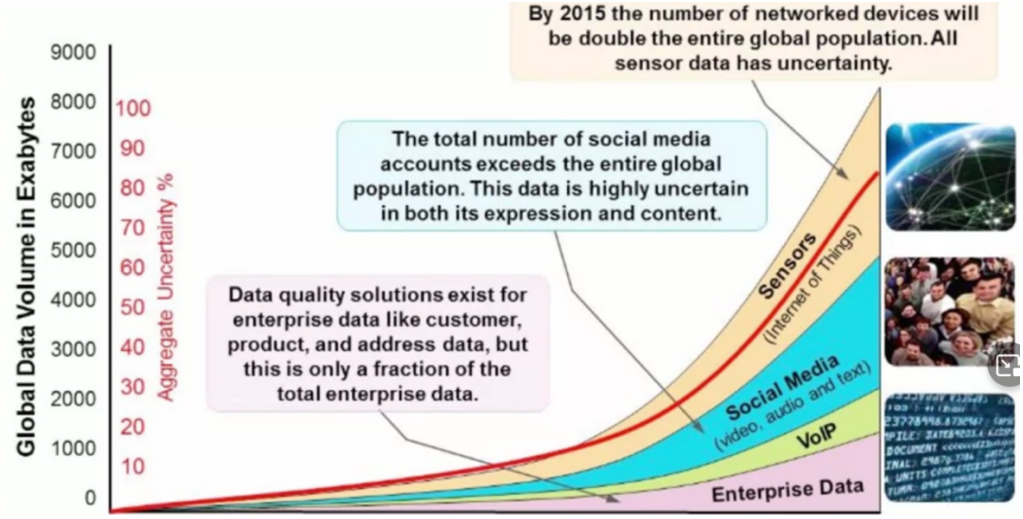
Veracidade

Veracidade de Big Data refere-se à qualidade dos dados. Às vezes é referido como validade ou volatilidade referindo-se ao tempo de vida dos dados. Veracidade é muito importante para tornar big data operacional. Porque big data pode ser poluído e incerto. Pode ser cheio de vieses, anormalidades e pode ser impreciso.

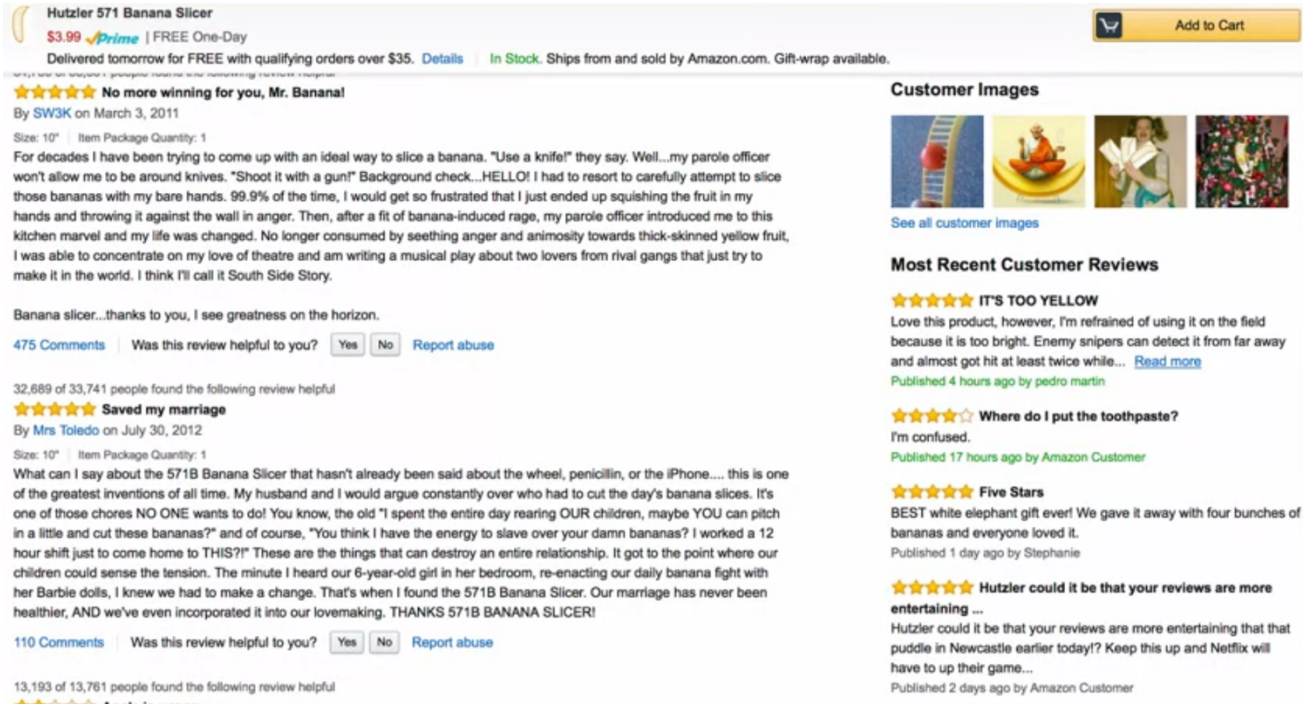
Os dados não têm valor se não forem precisos, os resultados da análise de big data são tão bons quanto os dados que estão sendo analisados. Isso geralmente é descrito na análise como lixo eletrônico igual a lixo para fora. Portanto, podemos dizer que, embora big data ofereça muitas oportunidades para tomar decisões ativadas por dados, a evidência fornecida pelos dados só é valiosa se os dados forem de uma qualidade satisfatória.

Há muitas maneiras diferentes de definir a qualidade dos dados. No contexto do big data, qualidade pode ser definida como uma função de um par de variáveis diferentes. Precisão dos dados, confiabilidade ou confiabilidade da fonte de dados. E como os dados foram gerados são todos fatores importantes que afetam a qualidade dos dados. Além disso, o quão significativo os dados são em relação ao programa que os analisa, é um fator importante e faz do contexto uma parte da qualidade.

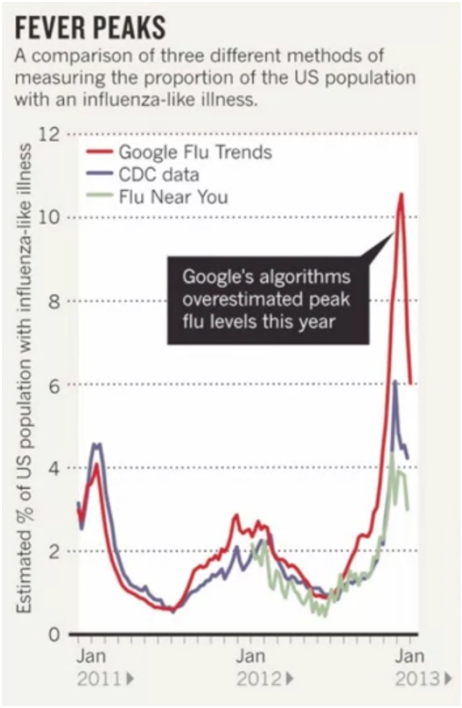
Neste gráfico de 2015, vemos os volumes de dados aumentando, começando com pequenas quantidades de dados corporativos para maiores, pessoas geraram voz sobre IP e dados de mídia social e dados de sensores gerados por máquina ainda maiores. Também vemos que a incerteza dos dados aumenta à medida que passamos de dados corporativos para dados do sensor. Isto é como esperávamos que fosse. Os dados empresariais tradicionais em armazéns têm soluções de qualidade padronizadas, como processos mestre para extração, transformação e carga dos dados que nos referimos como antes como ETL. À medida que as empresas começaram a incorporar pessoas menos estruturadas e não estruturadas e dados de máquinas em suas soluções de big data, os dados se tornam mais incertos. Há muitas razões para isso. Primeiro, dados não estruturados na Internet são imprecisos e incertos. Além disso, big data de alta velocidade deixa muito pouco ou nenhum tempo para ETL e, por sua vez, dificulta os processos de garantia de qualidade dos dados.



Vejamos essas avaliações de produtos para um cortador de banana na amazon.com. Um dos comentários de cinco estrelas diz que salvou seu casamento e comparou com as maiores invenções da história. Outro avaliador de cinco estrelas disse que seu oficial de condicional recomendou o fatiador, pois ele não tem permissão para estar perto de facas. Estes são obviamente falsos revisores. Agora pense em uma avaliação automatizada do produto passando por tais esplêndidas avaliações e estimando lotes de vendas para o cortador de banana e, por sua vez, sugerindo estocar mais do cortador no inventário. Amazon terá problemas.



Para um caso mais sério, vamos analisar o caso das tendências da gripe do Google de 2013. Para janeiro de 2013, o Google Friends realmente estimou quase o dobro de casos de gripe que foi relatado pelo CDC, os Centros de Controle e Prevenção de Doenças. A principal razão por trás disso foi que o Google Flu Trends usou um big data em a internet e não contabilizar adequadamente as incertezas sobre os dados. Talvez a atenção das notícias e das redes sociais prestada ao nível particularmente grave de gripe naquele ano tenha efetuado a estimativa. E resultou no que chamamos de uma estimativa acima. Este é um exemplo perfeito para quão imprecisos os resultados podem ser se apenas big data for usado na análise. Imagine o impacto econômico de fazer preparações para cuidados de saúde para o dobro da quantidade de casos de gripe. Isso seria enorme. O exemplo de tendências de gripe do Google também traz à tona a necessidade de ser capaz de identificar exatamente de onde vem big data que eles usaram.



Qual transformação big data passou por até o momento em que foi usado para uma estimativa? Isto é o que nos referimos como providência de dados. Assim como nos referimos a uma proveniência de artefatos. Como resumo, as crescentes torrentes de big data empurram para soluções rápidas para utilizá-lo em soluções analíticas. Isso cria desafios para manter o controle da qualidade dos dados. O que foi coletado, de onde veio, e como foi analisado antes de seu uso. Isto é semelhante a um artefato de arte tendo providência de tudo o que passou. Mas ainda mais complicado de alcançar com grandes volumes de dados chegando em variedades e velocidades.

Valência

Simplificando Valence refere-se a Conectividade. Quanto mais dados conectados forem, maior será a valência. O termo valência vem da química. Em química, falamos de elétrons centrais e elétrons de valência de um átomo. Os elétrons de valência estão na mais externa concha, têm o mais alto nível de energia e são

responsáveis pela ligação com outros átomos. Essa maior valência resulta em maior boding, que é maior conexão. Esta ideia é levada para a nossa definição do termo valência no contexto do big data.

Os itens de dados geralmente são conectados diretamente entre si. Uma cidade está conectada ao país a que pertence. Dois usuários do Facebook estão conectados porque são amigos. Um funcionário está conectado ao seu local de trabalho. Os dados também podem estar conectados indiretamente. Dois cientistas estão conectados, porque ambos são físicos. Para uma valência de coleta de dados mede a proporção de itens de dados realmente conectados para o número possível de conexões que podem ocorrer dentro da coleção.

O aspecto mais importante da valência é que a conectividade de dados aumenta ao longo do tempo. A série de gráficos de rede vem de um experimento social onde cientistas participantes de uma conferência foram convidados a conhecer outros cientistas que eles não conheciam antes. Depois de várias rodadas de reuniões, eles encontraram novas conexões mostradas por suas bordas vermelhas. Aumento de valência pode levar a um comportamento emergente de grupo em redes de pessoas, como a criação de novos grupos e coalizões que tenham valores e objetivos compartilhados. Um conjunto de dados de alta valência é mais denso.

Isso torna muitas críticas analíticas regulares muito ineficientes. Métodos analíticos mais complexos devem ser adotados para ter em conta a densidade crescente. Desafios mais interessantes surgem devido ao comportamento dinâmico dos dados. Agora há uma necessidade de modelar e prever como a valência de um conjunto de dados conectado pode mudar com o tempo e o volume. O comportamento dinâmico também leva ao problema da detecção de eventos, como explosões na coesão local em partes dos dados. E comportamento emergente em todo o conjunto de dados, como maior polarização em uma comunidade.

O sexto V: Valor

Embora possamos listar alguns outros aspectos com base no contexto, preferimos listar essas dimensões fundamentais cinco Vs. No entanto, no centro do desafio de big data está transformando todas as outras dimensões em valor comercial verdadeiramente útil. A ideia por trás do processamento de todo esse big data em primeiro lugar é trazer valor para o problema em questão.

Vamos explorar como dar os primeiros passos para começando a gerar valor de big data. Agora que vimos todos os caminhos, vamos nos concentrar em um exemplo de desafio de big data. Vamos imaginar agora que você faz parte de uma empresa chamada Eglence Inc. Um dos produtos da Eglence Inc é um jogo móvel altamente popular chamado Catch the Pink Flamingo. É um jogo multi-usuário onde os usuários têm que pegar tipos especiais de flamingos rosa que aparecem aleatoriamente no mapa do mundo em suas telas com base na missão que é atualizada aleatoriamente. O jogo é jogado por milhões de pessoas online em todo o mundo. Um dos objetivos do jogo é formar uma rede de jogadores para coletivamente cobrir o mapa do mundo com avistamentos de flamingo rosa e competir com outros grupos. Os usuários podem escolher seus grupos com base nas estatísticas do jogador. O site do jogo envia coisas legais grátis para usuários registrados. O registro exige que os usuários insiram informações demográficas como gênero, ano de nascimento, cidade, maior escolaridade e coisas assim. No entanto, a maioria dos usuários insere informações imprecisas sobre si mesmos, assim como a maioria de nós faz.

Para ajudar a melhorar o jogo, o jogo coleta dados de atividade de uso em tempo real de cada jogador e os alimenta para seus servidores de dados. Os jogadores deste jogo são entusiasticamente ativos nas redes sociais, e têm fortes associações com o jogo. Uma hashtag popular do Twitter para este jogo é, CatchThePinkFlamingo, que recebe mais de 200.000 menções em todo o mundo por dia. Há fortes comunidades de usuários que se reúnem via mídia social e se reúnem para jogar o jogo.

Agora, imagine-se como o arquiteto de soluções de big data da Fun Games Inc. Há definitivamente exemplos de todos os três tipos de fontes de dados neste exemplo. O aplicativo móvel gera dados para a análise da atividade do usuário. As conversas de jogadores no Twitter formam uma rica fonte de dados não estruturados

de pessoas. E os registros do cliente e do jogo são exemplos de dados que essa organização coleta. Este é um exemplo de big data desafiador onde todas as características de big data são representadas. Há grandes volumes de dados do jogador, jogo e Twitter, que também fala para a variedade de dados. Os fluxos de dados do aplicativo móvel, site e mídias sociais em tempo real, que podem ser definidos como dados de alta velocidade. A qualidade dos dados demográficos que os usuários inserem não é clara, e existem redes de jogadores que estão relacionados com o equilíbrio de big data.

1.6. Processando Big Data - Sistema de Arquivos distribuídos

A maioria de nós tem armários de arquivos físicos em nossos escritórios ou casas que nos ajudam a armazenar nossos documentos impressos. Todo mundo tem seu próprio método de organização de arquivos, incluindo a maneira como nós colocamos documentos semelhantes em um arquivo, ou a maneira como nós os classificamos em ordem alfabética ou de data. Quando os computadores saíram pela primeira vez, as informações e os programas foram armazenados em cartões perfurados. Estes cartões perfurados foram armazenados em armários de arquivos, assim como os armários de arquivos físicos hoje. É de onde vem o nome, sistema de arquivos.

A necessidade de armazenar informações em arquivos vem de uma necessidade maior de armazenar informações a longo prazo. Desta forma, a informação vive após o programa de computador, ou o que chamamos de processo, que a produziu termina. Se não tivermos arquivos, nosso acesso a essas informações não seria possível uma vez que um programa usasse ou produzisse. Mesmo durante o processo, talvez seja necessário armazenar grandes quantidades de informações que não podemos armazenar nos componentes do programa ou na memória do computador. Além disso, uma vez que os dados estão em um arquivo, vários processos podem acessar as mesmas informações, se necessário. Por todos esses motivos, armazenamos informações em arquivos em um disco rígido.

Há muitos desses arquivos, e eles são gerenciados pelo seu sistema operacional, como Windows ou Linux. Como o sistema operacional gerencia arquivos é chamado de sistema de arquivos. Como essas informações são armazenadas em unidades de disco tem alto impacto na eficiência e velocidade de acesso aos dados, especialmente no caso de big data. Embora os arquivos tenham endereços exatos para seus locais na unidade, referindo-se às unidades de dados de sequência desses blocos, isso é chamado de estrutura plana, ou construção hierárquica de registros de índice, isso é chamado de banco de dados. Eles também têm nomes simbólicos legíveis por humanos, geralmente seguidos por uma extensão. As extensões dizem que tipo de arquivo é, em geral. Programas e usuários podem acessar arquivos com seus nomes. O conteúdo de um arquivo pode ser numérico, alfabético, alfanumérico, ou executáveis binários.

A maioria dos usuários de computador trabalha em laptops pessoais ou em computadores desktop com um único disco rígido. Neste modelo, o usuário está limitado à capacidade de seu disco rígido. A capacidade de diferentes dispositivos varia. Por exemplo, enquanto seu telefone ou tablet pode ter uma capacidade de armazenamento na ordem de gigabytes, seu laptop pode ter um terabyte de armazenamento, mas e se você tiver mais dados? Alguns de vocês provavelmente tiveram problemas no passado, com ficando sem espaço no seu disco rígido. Uma maneira de resolver isso é ter um disco rígido externo e armazenar seus arquivos lá ou, você pode comprar um disco maior. Ambas as opções são um pouco complicadas, para copiar os dados para um novo disco, não é? Eles podem nem ser uma opção às vezes.

Agora imagine, você tem dois computadores e armazenando alguns de seus dados em um e o resto de seus dados em outro. A forma como organiza e particiona os seus dados entre estes computadores depende de si. Você pode querer armazenar seus dados de trabalho em um computador e seus dados pessoais em outro. Distribuir dados em vários computadores pode ser uma opção, mas levanta novos problemas. Nessa situação, você precisa saber onde encontrar os arquivos necessários, dependendo do que estiver fazendo. Você pode

encontrá-lo gerenciável, se for apenas seus dados. Mas agora imagine ter milhares de computadores para armazenar seus dados com grandes volumes e variedade. Não seria bom ter um sistema que possa lidar com o acesso aos dados e fazer isso por você?

Este é um caso que pode ser tratado por um sistema de arquivos distribuídos. Agora, vamos supor que existem racks desses computadores, muitas vezes até distribuídos em toda a rede local ou de área ampla, porque esses sistemas de arquivos, sistemas de arquivos distribuídos. Os conjuntos de dados, ou partes de um conjunto de dados, podem ser replicados nos nós de um sistema de arquivos distribuído. Como os dados já estão nesses nós, então a análise de partes dos dados é necessária de forma paralela de dados, computação pode ser movida para esses nós.

Além disso, os sistemas de arquivos distribuídos replicam os dados entre os racks e também computadores distribuídos por regiões geográficas. A replicação de dados torna o sistema mais tolerante a falhas. Isso significa que, se alguns nós ou um rack cair, há outras partes do sistema, os mesmos dados podem ser encontrados e analisados. Replicação de dados também ajuda a dimensionar o acesso a esses dados por muitos usuários. Muitas vezes, se os dados forem populares, muitos processos do leitor desejarão acesso a eles. Em uma replicação altamente paralelizada, cada leitor pode obter seu próprio nó para acessar e analisar dados. Isso aumenta o desempenho geral do sistema.

Note que um problema com essa replicação distributiva é, que é difícil fazer alterações nos dados ao longo do tempo. No entanto, na maioria dos sistemas de big data, os dados são gravados uma vez e as atualizações dos dados são mantidas como conjuntos de dados adicionais ao longo do tempo. Como resumo, um sistema de arquivos é responsável pela organização de o armazenamento de informações de longo prazo em um computador. Quando muitos computadores de armazenamento estão conectados através da rede, chamamos de sistema de arquivos distribuído. Os sistemas de arquivos distribuídos fornecem escalabilidade de dados, tolerância a falhas e alta simultaneidade por meio de particionamento e replicação de dados em muitos nós.

a. Computação Escalável pela Internet

A maior parte da computação é feita em um único nó de computação. Se a computação precisar de mais do que um nó ou processamento paralelo, como muitos problemas de computação científica, usamos computadores paralelos. Simplificando, um computador paralelo é um número muito grande de nós de computação única com capacidades especializadas conectadas a outra rede. Por exemplo, o Supercomputador Gordon aqui no *The San Diego Supercomputer Center*, tem 1.024 nós de computação com 16 núcleos cada igual a 16.384 núcleos de computação no total. Este tipo de computador especializado é bastante caro em comparação com seu primo mais recente, o cluster de commodities.

O termo, cluster de commodities, é frequentemente ouvido em conversas de big data. Clusters de commodities são computadores paralelos acessíveis com um número médio de nós de computação. Eles não são tão poderosos quanto os computadores paralelos tradicionais e são muitas vezes construídos a partir de nós menos especializados. Na verdade, os nós no cluster de commodities são mais genéricos em suas capacidades de computação. A comunidade de computação orientada a serviços através da Internet tem pressionado a computação para ser feita em clusters de commodities como computações distribuídas. E, por sua vez, reduzindo o custo da computação através da Internet.

Em clusters de commodities, os nós de computação são agrupados em racks conectados entre si através de uma rede rápida. Pode haver muitos desses racks em quantidades extensíveis. A computação em um ou mais desses clusters em uma rede local ou na internet é chamada de computação distribuída. Essas arquiteturas permitem o que chamamos de paralelismo de dados. No paralelismo de dados, muitos trabalhos que não compartilham nada podem funcionar em conjuntos de dados diferentes ou partes de um conjunto de dados. Este tipo de paralelismo às vezes é chamado como paralelismo de nível de trabalho. Grandes volumes e variedades

de big data podem ser analisados usando este modo de paralelismo, alcançando escalabilidade, desempenho e redução de custos.

Como você pode imaginar, há muitos pontos de falha dentro dos sistemas. Um nó ou um rack inteiro pode falhar a qualquer momento. A conectividade de um rack com a rede pode parar ou as conexões entre nós individuais podem quebrar. Não é prático reiniciar tudo sempre, se ocorrer falha. A capacidade de se recuperar de tais falhas é chamada de tolerância a falhas. Para tolerância a falhas de tais sistemas, surgiram duas soluções perfeitas. Ou seja, armazenamento de dados redundante e reinicialização de tarefas paralelas individuais com falha.

Os clusters de commodities são uma maneira econômica de alcançar escalabilidade paralela de dados para aplicativos de big data. Esses tipos de sistemas têm um maior potencial para falhas parciais. É esse tipo de computação distribuída que empurrou para uma mudança em direção a sistemas econômicos confiáveis e tolerantes a falhas para gerenciamento e análise de big data.

b. Modelos de Programação para Big Data

A computação escalável pela Internet viabiliza a escalabilidade de dados em paralelo para aplicativos de big data. Graças a clusters de commodities econômicos, juntamente com avanços em sistemas de arquivos distribuídos para mover a computação para dados, fornecem um potencial para realizar análises de big data escaláveis.

A próxima coisa que falaremos é como aproveitar desses avanços de infraestrutura. Quais são os modelos de programação certos? Um modelo de programação é uma abstração ou uma infraestrutura existente. É um conjunto de bibliotecas de tempo de execução abstratas e linguagens de programação que formam um modelo de computação. Este nível de abstração pode ser de baixo nível como na linguagem de máquina em computadores. Ou muito alto como em linguagens de programação de alto nível, por exemplo, Java.

Então podemos dizer, se a infraestrutura habilitadora para análise de big data é sistemas de arquivos distribuídos como mencionamos, então o modelo de programação para big data deve permitir a programabilidade das operações dentro de sistemas de arquivos distribuídos. O que queremos dizer com isso ser capaz de escrever programas de computador que funcionam eficientemente em cima de sistemas de arquivos distribuídos usando big data e tornando mais fácil lidar com todos os problemas potenciais.

Vamos descrever os requisitos para modelos de programação de big data. Em primeiro lugar, esse modelo de programação para big data deve suportar operações comuns de big data, como dividir grandes volumes de dados. Isso significa para particionamento e colocação de dados dentro e fora da memória do computador, juntamente com um modelo para sincronizar os conjuntos de dados mais tarde. O acesso aos dados deve ser alcançado de forma rápida.

Ele deve permitir distribuição rápida para nós dentro de um rack e estes são potencialmente, os nós de dados para os quais movemos o cálculo. Isso significa agendamento de muitas tarefas paralelas ao mesmo tempo. Ele também deve permitir a confiabilidade da computação e tolerância total contra falhas. Isso significa que ele deve habilitar replicações programáveis e recuperação de arquivos quando necessário. Deve ser facilmente escalável para as notas distribuídas onde os dados são produzidos.

Ele também deve permitir a adição de novos recursos para aproveitar os computadores distributivos e dimensionar para dados mais ou mais rápidos sem perder o desempenho. Isso é chamado de dimensionamento, se necessário. Uma vez que há uma variedade de tipos diferentes de dados, como documentos, gráficos, tabelas, valores-chave, etc.

Um modelo de programação deve permitir operações sobre um determinado conjunto desses tipos. Nem todos os tipos de dados podem ser suportados por um modelo específico, mas os modelos devem ser otimizados

para pelo menos um tipo. Está ficando um pouco complicado? Não tem que ser. Na verdade, aplicamos modelos semelhantes em nossas vidas diárias para tarefas diárias.

Vejamos o cenário em que você pode aplicar esse modelo sem saber. Imagine uma tarde pacífica de sábado. Você recebe um telefonema de um amigo e ele diz, que virão em sua casa em uma hora para o jantar. Parece que você esqueceu completamente que tinha convidado seus amigos para jantar... Então você diz que está ansioso para sua chegada e corre para a cozinha.

Como uma solução rápida, você decide cozinhar macarrão com um pouco de molho de tomate. Você precisa aproveitar a paralelização, de modo que o jantar esteja pronto quando seu convidado chegar, isso é dentro de uma hora. Você chama seu cônjuge e seus filhos adolescentes para ação na cozinha. Agora, você precisa dar a eles instruções para começar a picar os ingredientes para você. Mas no calor do momento, você acaba misturando as cebolas, tomates e pimentas. Em vez de classificá-los primeiro, você dá a todos um lote misturado aleatoriamente de diferentes tipos de vegetais. Eles são obrigados a usar seus poderes de computador para cortar os vegetais. Eles precisam garantir não misturar diferentes tipos de vegetais. Quando todos terminarem de cortar, você quer agrupar os vegetais por seus tipos. Você pede a cada ajudante para coletar itens do mesmo tipo, colocá-los em uma tigela grande e rotular esta tigela grande com a soma de pesos individuais dos tomates em uma tigela, pimentas em outra e as cebolas na terceira tigela. No final, você tem boas tigelas grandes com o peso total de cada vegetal rotulado nele. Seus ajudantes logo terminarão seu trabalho enquanto você está focado em coordenar suas ações e outras tarefas de jantar na cozinha, você pode começar a cozinhar seu macarrão.

O que você acabou de ver é um excelente exemplo de modelagem de big data em ação. Somente são realmente os dados processados por processadores humanos. Este cenário pode ser modelado por um modelo de programação comum para big data, intitulada MapReduce.

O MapReduce é um modelo de programação de big data que suporta todos os requisitos de modelagem de big data que mencionamos. Ele pode modelar o processamento de dados grandes, dividir complicações em diferentes tarefas paralelas e fazer uso eficiente de grandes clusters de commodities e sistemas de arquivos distribuídos. Além disso, ele resume os detalhes de paralelização, tolerância total, distribuição de dados, monitoramento e balanceamento de carga. Como um modelo de programação, foi implementado em alguns frameworks de big data diferentes.

Veremos mais detalhes sobre MapReduce e como sua implementação Hadoop funciona. Para resumir, os modelos de programação para big data são abstrações sobre sistemas de arquivos distribuídos. Os modelos de programação desejados para big data devem lidar com grandes volumes e variedades de dados. Suporta tolerância total e fornece funcionalidade de expansão horizontal. MapReduce é um desses modelos, implementado em uma variedade de estruturas, incluindo o Hadoop. Resumiremos o funcionamento interno da implementação do Hadoop a seguir.

2. Hadoop

Por que o Hadoop? Todos ouvimos dizer que os projetos relacionados ao Hadoop e neste ecossistema são ótimos para big data. Responderemos aos quatro Ws e um H sobre porque essa declaração é verdadeira. Antes de aprofundarmos os detalhes do Hadoop, vamos analisar as características do ecossistema Hadoop. O que há no ecossistema? Por que é benéfico? Onde é usado? Quem o usa? E como funcionam essas ferramentas?

As estruturas e aplicativos do ecossistema do Hadoop que descreveremos têm vários temas e objetivos abrangentes. Primeiro, eles fornecem escalabilidade para armazenar grandes volumes de dados em hardware de commodities. À medida que o número de sistemas aumenta, também aumenta a chance de falhas e falhas de hardware. Um segundo objetivo, suportado pela maioria das estruturas no ecossistema do Hadoop, é a capacidade de recuperar graciosamente desses problemas. Além disso, como já mencionamos antes, big data vem em uma variedade de formatos, como arquivos de texto, gráfico de redes sociais, dados de sensor de streaming e imagens. Um terceiro objetivo para o ecossistema Hadoop, então, é a capacidade de lidar com esses tipos de dados diferentes para qualquer tipo de dados. Você pode encontrar vários projetos no ecossistema que o suportam. Um quarto objetivo do ecossistema Hadoop é a capacidade de facilitar um ambiente compartilhado. Como mesmo clusters de tamanho modesto podem ter muitos núcleos, é importante permitir que vários trabalhos sejam executados simultaneamente. Por que comprar servidores apenas para deixá-los ociosos?

Outro objetivo do ecossistema Hadoop é fornecer valor para sua empresa. O ecossistema inclui uma ampla gama de projetos de código aberto apoiados por uma grande comunidade ativa. Estes projetos são gratuitos de uso e fácil de encontrar suporte para. A seguir, daremos uma olhada mais detalhada no ecossistema Hadoop. Primeiro, vamos explorar os tipos de projetos disponíveis e os tipos de recursos que eles fornecem. Em seguida, vamos dar uma olhada mais profunda nas três partes principais do Hadoop. O sistema de arquivos distribuído do Hadoop, ou HDFS. YARN, o agendador e o gerenciador de recursos. E MapReduce, um modelo de programação para o processamento de big data. Em seguida, discutiremos a computação em nuvem e os tipos de modelos de serviço que ela fornece. Também descreveremos situações em que o Hadoop não é a melhor solução. Este módulo conclui então com duas leituras envolvendo experiência prática com HDFS e MapReduce.

2.1. O Desafio da Escalabilidade

Em um projeto de Big Data, o processamento de grande volume de dados tem como um dos maiores desafios a escalabilidade da solução. Mas o que seria isso? Um sistema é considerado escalável se ele permanece com desempenho adequado, mesmo com um aumento significativo do número de usuários, de dados e/ou de recursos. A escalabilidade está relacionada a capacidade do sistema de gerir adequadamente os recursos computacionais disponíveis, monitorar a execução para verificar se há perdas de desempenho e criar mecanismos de resiliência que assegure a continuidade da operação, evitando interrupções.

Em projetos de Big Data, é fundamental planejar a arquitetura de forma a garantir a adequação da plataforma para o aumento de demanda. A plataforma deve assegurar alta disponibilidade, conseguindo se manter ativa mesmo na ocorrência de falhas, que provavelmente ocorrerão devido ao crescente número de dispositivos em operação. A escalabilidade pode ser abordada sob duas abordagens: escalabilidade vertical e escalabilidade horizontal.

Escalabilidade vertical - *Scale up*

Para ilustrar a abordagem de escalabilidade vertical, Marquesone (2016) faz uma analogia a construção de um edifício. A expansibilidade do sistema deve ser considerada desde os alicerces. Ou seja, o aumento de capacidade de processamento ou armazenamento está condicionada a infraestrutura disponível. Se por um lado este modelo impõe uma restrição de crescimento, deve-se considerar que a extensibilidade permitida ocorre

utilizando as melhores opções de compatibilidade, e, com isso, o sistema é mais confiável, embora as adaptações sejam mais caras e o aumento da capacidade seja limitado ao planejamento inicial.

Voltando à analogia da construção de um edifício, um prédio é uma opção interessante, pois os recursos comuns da infraestrutura podem ser compartilhados entre os usuários. Além disso, o espaço usado para a alocação dos recursos é muito menor se comparado ao espaço necessário para a construção de casas térreas com a mesma quantidade de cômodos. O acoplamento de cômodos traz benefícios, mas as mudanças realizadas no prédio afetam a todos. A escalabilidade nesta abordagem envolve a adição de processadores, pentes de memória e discos em um único servidor, que possui um limite de expansão pré-definido.

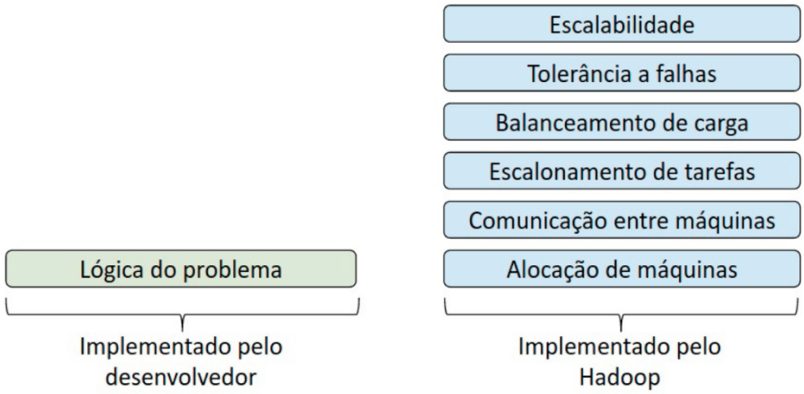
Escalabilidade horizontal - *Scale out*

Utilizando a analogia das edificações, a escalabilidade horizontal se assemelha a um condomínio de casas. A necessidade de espaço físico é maior para alocar novos recursos, porém as alterações não trazem implicações para a configurações prévia. Não há compartilhamento explícito de recursos entre servidores, sendo cada um deles um módulo independente. Desta forma, quando um novo recurso é adicionado, os serviços não ficam indisponíveis.

Esta abordagem de escalabilidade baseia-se no trabalho colaborativo - um cluster computacional irá dividir a carga de trabalho em atividades menores que serão executadas de forma distribuída entre os módulos independentes, ou nós. Neste caso, cada nó possui uma capacidade de processamento e armazenamento muito menor que o servidor adotado numa abordagem de escalabilidade vertical. O trabalho de vários nós, atuando de forma coordenada garante o sucesso da operação no modelo de escalabilidade horizontal. Se for detectada uma queda no desempenho ou aumento de usuários ou de recursos da aplicação, é possível adicionar novos servidores e redistribuir a carga de trabalho.

A escalabilidade horizontal permite que o desempenho da aplicação seja aperfeiçoado sob demanda. Os servidores independentes adicionados são mais simples e mais baratos quando comparados aos computadores de escala vertical e a capacidade de expansão possui uma perspectiva ilimitada, sobretudo em ambientes de computação em nuvem.

Embora apresente uma série de benefícios, a escalabilidade horizontal exige um sistema de processamento distribuído, trazendo uma camada de complexidade inerente aos sistemas de processamento paralelo. Se as questões de processamento paralelo não forem gerenciadas de forma eficiente, o desempenho será comprometido. Por essa questão, nasceu a necessidade de tecnologias que cuidassem do processamento paralelo, permitindo que os programadores focassem nas questões inerentes aos negócios. Surgiram assim as tecnologias de big data. Ou seja, os requisitos para o desenvolvimento do ambiente Hadoop são ilustrados na Figura abaixo:

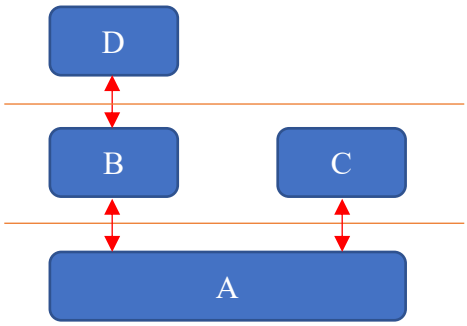


2.2. Ecossistema Hadoop

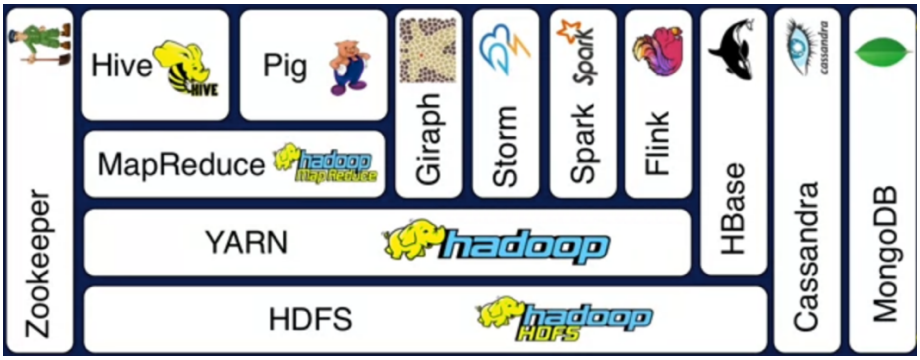
Como o movimento de código aberto de big data começou? Em 2004, o Google publicou um artigo sobre seu framework de processamento de interno chamado MapReduce. No ano seguinte, o Yahoo lançou uma implementação de código aberto baseada nesta estrutura chamada Hadoop. Nos anos seguintes, outras estruturas e ferramentas foram lançadas para a comunidade como projetos de código aberto. Essas estruturas forneceram novos recursos ausentes no Hadoop, como consultas SQL ou scripts de alto nível.

Hoje, existem mais de 100 projetos de código aberto para big data e esse número continua a crescer. Muitos dependem do Hadoop, mas alguns são independentes. Com tantas estruturas e ferramentas disponíveis, como aprendemos o que elas fazem? Podemos organizá-los com um diagrama de camadas para entender suas capacidades. Às vezes, também usamos o termo stack em vez de um diagrama de camadas. Em um diagrama de camada, um componente usa a funcionalidade ou os recursos dos componentes na camada abaixo dele. Normalmente, os componentes na mesma camada não se comunicam. E um componente nunca assume que uma ferramenta ou componente específico esteja acima dele.

No exemplo abaixo, o componente A está na camada inferior, quais componentes B e C usam. Componente D usa B, mas não C. E D não usam diretamente A. Vejamos um conjunto de ferramentas no ecossistema Hadoop como um diagrama de camadas. Este diagrama de camadas é organizado verticalmente com base na interface. Interfaces de baixo nível, portanto, armazenamento e agendamento, na parte inferior. E linguagens de alto nível e interatividade no topo.



O sistema de arquivos distribuídos do Hadoop, ou HDFS, é a base para muitas estruturas de big data, uma vez que fornece armazenamento escalonável e confiável. À medida que o tamanho dos seus dados aumenta, você pode adicionar o hardware ao HDFS para aumentar a capacidade de armazenamento, de modo que ele permita a expansão dos seus recursos. Hadoop YARN oferece agendamento flexível e gerenciamento de recursos sobre o armazenamento HDFS.O YARN é usado no Yahoo para agendar trabalhos em 40.000 servidores.



MapReduce é um modelo de programação que simplifica a computação paralela. Em vez de lidar com as complexidades de sincronização e agendamento, você só precisa dar MapReduce duas funções, mapear e

reduzir, como você ouviu antes. Este modelo de programação é tão poderoso que o Google já o usou para indexar sites. MapReduce assume apenas um modelo limitado para expressar dados.

Hive e Pig são dois modelos de programação adicionais no topo do MapReduce para aumentar a modelagem de dados do MapReduce com álgebra relacional e modelagem de fluxo de dados respectivamente. Hive foi criado no Facebook para emitir consultas de semelhantes ao SQL usando MapReduce em seus dados no HDFS. Pig foi criado no Yahoo para modelar programas baseados em fluxo de dados usando MapReduce. Graças à estabilidade do YARN para gerenciar recursos, não apenas para MapReduce, mas outros modelos de programação também.

Giraph foi construído para processar gráficos em larga escala de forma eficiente. Por exemplo, o Facebook usa Giraph para analisar os gráficos sociais de seus usuários. Da mesma forma, Storm, Spark e Flink foram criados para em tempo real e no processamento de memória de big data no topo do agendador de recursos do YARN e HDFS. O processamento na memória é uma maneira poderosa de executar aplicativos de big data ainda mais rápido, alcançando o melhor desempenho da 100x para algumas tarefas.

Às vezes, seus dados ou tarefas de processamento não são facilmente ou eficientemente representados usando o modelo de arquivo e diretório de armazenamento. Exemplos disso incluem coleções de valores-chave ou tabelas esparsas grandes. Projetos NoSQL, como Cassandra, MongoDB e HBase, lidam com esses casos. Cassandra foi criada no Facebook, mas o Facebook também usou o HBase para sua plataforma de mensagens.

Finalmente, executar todas essas ferramentas requer um sistema de gerenciamento centralizado para sincronização, configuração e para garantir alta disponibilidade. Zookeeper executa essas funções. Foi criado pelo Yahoo para disputar serviços nomeados após animais. Um dos principais benefícios do ecossistema Hadoop é que todas essas ferramentas são projetos de código aberto. Você pode baixar e usá-los gratuitamente. Cada projeto tem uma comunidade de usuários e desenvolvedores que responde perguntas, corrige bugs e implementa novos recursos. Você pode misturar e combinar para obter apenas as ferramentas necessárias para alcançar seus objetivos. Alternativamente, existem várias pilhas pré-construídas dessas ferramentas oferecidas por empresas como Cloudera, MAPR e Hortonworks. Essas empresas fornecem as pilhas de software principais gratuitamente e oferecem suporte comercial para ambientes de produção. Como resumo, o ecossistema Hadoop consiste em um número crescente de ferramentas de código aberto. Fornecendo oportunidades para escolher a ferramenta certa para as tarefas certas para melhorar o desempenho e reduzir custos. Vamos revelar algumas dessas ferramentas em mais detalhes e fornecer uma análise de quando usar qual no próximo conjunto de palestras.

2.3. Sistemas de Arquivos Distribuídos do Hadoop - HDFS

O Hadoop Distributed File System, um sistema de armazenamento para big data. Como uma camada de armazenamento, o sistema de arquivos distribuído Hadoop, ou a maneira como chamamos de HDFS. Serve como base para a maioria das ferramentas no ecossistema Hadoop. Ele fornece dois recursos que são essenciais para o gerenciamento de big data. Escalabilidade para grandes conjuntos de dados. E confiabilidade para lidar com falhas de hardware.

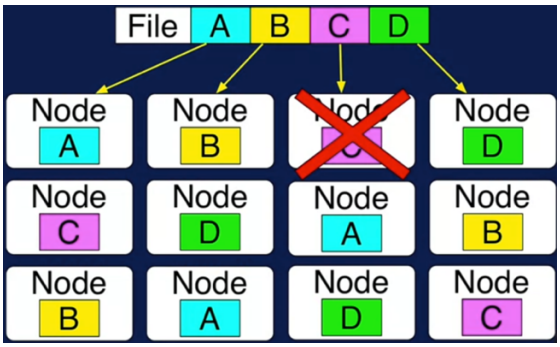
O HDFS torna transparentes ao usuário as questões complexas de distribuição dos dados. Ou seja, no momento em que solicitamos a escrita ou leitura dos dados, não precisamos saber como essas operações serão realizadas internamente, o que torna bem mais simples sua utilização. Por exemplo, para fazer o envio de um arquivo local para o HDFS, usamos comandos similares a sistemas UNIX:

```
$ hadoop fs -mkdir diretorio
$ hadoop fs -put $HOME/baseDeDados.csv diretorio
```

HDFS permite armazenar e acessar grandes conjuntos de dados. De acordo com a Hortonworks, um dos principais fornecedores de serviços Hadoop, HDFS mostrou escalabilidade de produção de até 200 petabytes e

um único cluster de 4.500 servidores. Com perto de um bilhão de arquivos e blocos. Se você ficar sem espaço, você pode simplesmente adicionar mais nós para aumentar o espaço. HDFS alcança escalabilidade por particionamento ou dividindo arquivos grandes em vários computadores. Isso permite o acesso paralelo a arquivos muito grandes, uma vez que os cálculos são executados em paralelo em cada nó onde os dados são armazenados. O tamanho típico do arquivo é de gigabytes a terabytes. O tamanho padrão do bloco, o tamanho de cada parte de um arquivo é 128 megabytes. Mas você pode configurar isso para qualquer tamanho.

Ao espalhar o arquivo por muitos nós, as chances são aumentadas de que um nó que armazena um dos blocos falhe. O que acontece a seguir? Perdemos as informações armazenadas no bloco C? HDFS foi projetado para tolerância total nesse caso. HDFS replica, ou faz uma cópia de blocos de arquivos em diferentes nós para evitar a perda de dados. Neste exemplo, o nó que falhou o bloco armazenado C. Mas bloco C foi replicado em dois outros nós no cluster. Por padrão, o HDFS mantém três cópias de cada bloco. Este é o fator de replicação padrão. Mas você pode alterá-lo globalmente para cada arquivo, ou por arquivo. HDFS também é projetado para lidar com uma variedade de tipos de dados alinhados com big data variedade.



Para ler um arquivo no HDFS, você deve especificar o formato de arquivo de entrada. Da mesma forma para gravar o arquivo, você deve fornecer o formato de arquivo de saída. HDFS fornece um conjunto de formatos para tipos de dados comuns. Mas isso é extensível e você pode fornecer formatos personalizados para seus tipos de dados. Por exemplo, arquivos de texto podem ser lidos. Linha por linha ou palavra de cada vez. Os dados geoespaciais podem ser lidos como vetores ou rasters. Formatos de dados específicos para dados geoespaciais ou outros formatos de dados específicos de domínio. Como formatos FASTA ou FASTQ para genômica de dados de sequência.

HDFS é composto por dois componentes. NameNode e DataNode. Eles operam usando um relacionamento mestre escravo. Onde o NameNode emite comentários para DataNodes no cluster. O NameNode é responsável pelos metadados. e DataNodes fornecem armazenamento em bloco. Geralmente há um NameNode por cluster, um DataNode no entanto, é executado em cada nó no cluster. Em algum sentido, o NameNode é o administrador ou o coordenador do cluster HDFS. Quando o arquivo é criado, o NameNode registra o nome, a localização na hierarquia de diretórios e outros metadados. O NameNode também decide quais nós de dados armazenar o conteúdo do arquivo e lembra esse mapeamento.

O DataNode é executado em cada nó no cluster. E é responsável por armazenar os blocos de arquivos. O nó de dados escuta comandos do nó de nome para criação de blocos, exclusão e replicação. A replicação fornece dois recursos principais. Tolerância a falhas e localidade de dados. Conforme discutido anteriormente, quando uma máquina no cluster tem uma falha de hardware, são duas outras cópias de cada bloco que são armazenadas nesse nó. Portanto, nenhum dado é perdido.

Replicação também significa que o mesmo bloco será armazenado em diferentes nós em o sistema que estão em diferentes localizações geográficas. Um local pode significar um rack específico ou um data center em uma cidade diferente. A localização é importante já que queremos mover a computação para os dados e não o contrário.

O fator de replicação padrão é três, mas você pode alterar isso. Um alto fator de replicação significa mais proteção contra falhas de hardware, e melhores chances de localidade de dados. Mas isso também significa que maior espaço de armazenamento é usado. Como resumo, o HDFS fornece armazenamento de big data escalável, particionando arquivos em vários nós. Isso ajuda a dimensionar a análise de big data para grandes volumes de dados. O aplicativo protege contra falhas de hardware e fornece localidade de dados quando movemos complicações analíticas para dados.

2.4. YARN - O Gerenciador de Recurso do Hadoop

YARN (Yet Another Resource Negotiator) é uma camada de gerenciamento de recursos que fica logo acima da camada de armazenamento HDFS. YARN interage com aplicativos e agende recursos para seu uso. YARN permite executar vários aplicativos em HDFS aumenta a eficiência de recursos e vamos além do mapa reduzir ou mesmo além do modelo de programação paralela de dados.

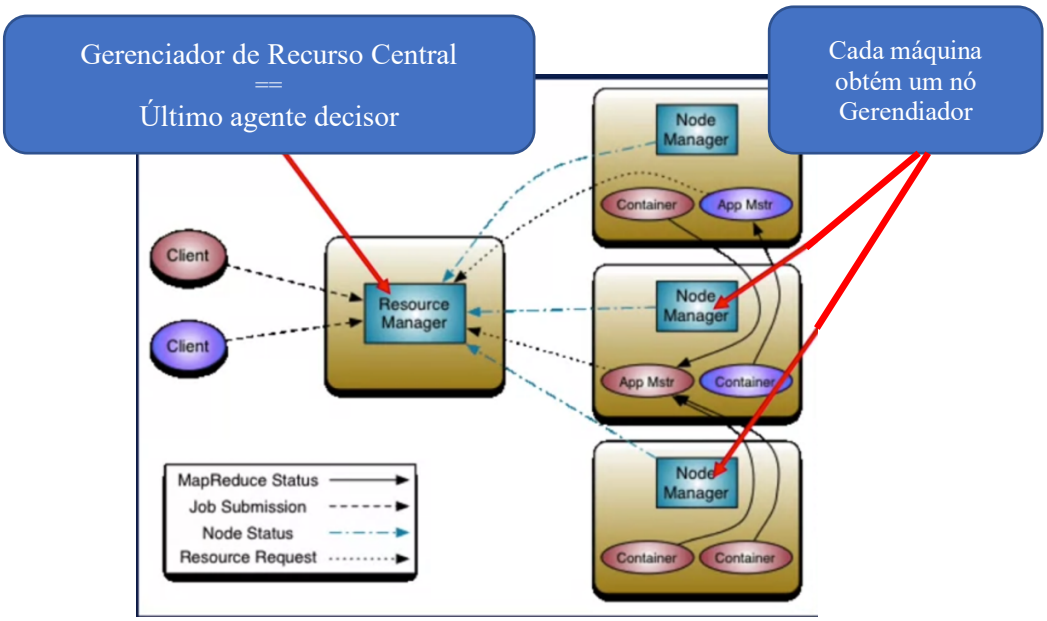
Quando o Hadoop foi criado, esse não era o caso. Na verdade, a pilha Hadoop original não tinha gerenciador de recursos. Estes dois diagramas, mostram, alguns de sua evolução nos últimos dez anos. Uma das maiores limitações do Hadoop 1.0, foi sua incapacidade de suportar aplicativos não baseados no MapReduce. Isso significava que, para aplicativos avançados, como análise de gráficos, exigia diferentes formas para modelagem, e para visualização de dados, você precisaria mover seus dados para outra plataforma. Isso dá muito trabalho se seus dados forem grandes.

A adição do YARN entre o HDFS e os aplicativos permitiu a criação de novos sistemas, com foco em diferentes tipos de aplicações de big data, como o Giraph para análise de dados de gráficos, o Storm para a análise de dados de streaming de e o Spark para análise na memória. YARN faz isso fornecendo uma estrutura padrão que oferece suporte ao desenvolvimento de aplicativos personalizado no ecossistema HADOOP.

O YARN permite extrair o máximo de benefícios de seus conjuntos de dados , permitindo que você use as ferramentas que você acha que são melhores para seu big data. Vamos dar uma olhada na arquitetura do YARN sem ser muito técnico. Nesta imagem, observe o gerenciador de recursos no centro e os gerenciadores de nós em cada um dos três nós à direita. O gerenciador de recursos controla todos os recursos e decide quem recebe o quê.O gerenciador de nó opera no nível da máquina e é responsável por uma única máquina. Juntos, o gerenciador de recursos e o gerenciador de nó formam a estrutura de computação de dados. Cada aplicativo obtém um mestre de aplicativo. Ele negocia recurso do Gerenciador de Recursos e ele fala com o Gerenciador de Node para obter suas tarefas concluídas. Observe os ovais rotulados Container O contêiner é uma noção abstrata que significa um recurso que é uma coleção de rede de disco de memória da CPU e outros recursos dentro da nota de computação para simplificar e ser menos preciso você pode pensar em um contêiner e a máquina. Analisamos as engrenagens essenciais do motor YARN para lhe dar uma ideia dos principais componentes do YARN.

Agora, quando você ouvir termos como Resource Manager, Node Manager e Container, você terá uma compreensão de quais tarefas eles são responsáveis. Aqui está um exemplo da vida real para mostrar a força Hadoop 2.0 sobre 1.0. Yahoo conseguiu executar quase duas vezes mais trabalhos por dia com o Yarn do que com o Hadoop 1.0. Eles também experimentaram um aumento substancial na utilização da CPU. Yahoo! até afirmou que a atualização para o YARN era igual em adicionando 1000 máquinas ao cluster de 2500 máquinas. Isso é grande.

O sucesso do YARN é evidente a partir de um crescimento explosivo de aplicativo diferente que o ecossistema Hadoop agora tem. Novo no fio? Você pode usar a ferramenta de sua escolha em vez de seu big data sem qualquer incômodo. Compare isso com o Hadoop 1.0 que foi limitado apenas ao MapReduce.



Yarn oferece muitas maneiras de aplicativos extrair valor dos dados. Ele permite que você execute muitos aplicativos distribuídos no mesmo cluster do Hadoop. Além disso, o YARN reduz a necessidade de mover dados e suporta maior utilização de recursos, resultando em custos mais baixos. É uma plataforma escalável que permitiu o crescimento de vários aplicativos sobre o HDFS, enriquecendo o ecossistema Hadoop.

2.5. MapReduce - Programação Simples para Grandes Resultados!

MapReduce é um modelo de programação para o ecossistema Hadoop. Ele depende do YARN para agendar e executar processamento paralelo sobre os blocos de arquivos distribuídos no HDFS. Existem várias ferramentas que usam o modelo MapReduce para fornecer uma interface de nível superior para outros modelos de programação.

Hive tem uma interface semelhante ao SQL que adiciona recursos que ajudam com a modelagem de dados relacionais. E Pig é uma linguagem de fluxo de dados de alto nível que adiciona recursos que ajudam na modelagem de mapas de processo.

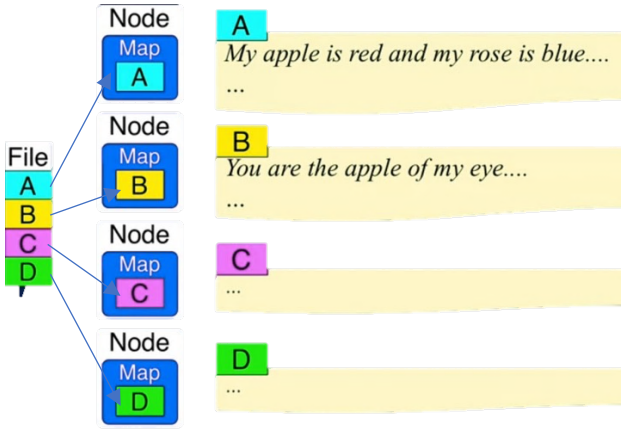
A programação paralela tradicional requer experiência em vários conceitos de computação e sistemas . Por exemplo, mecanismos de sincronização como bloqueios, semáforos, e monitores são essenciais. E usá-los incorretamente pode travar seu programa ou afetar gravemente o desempenho. Esta curva de aprendizagem alta torna difícil. Também é propenso a erros, já que seu código pode ser executado em centenas, ou milhares de nós, cada um com muitos núcleos. E qualquer problema relacionado a esses processos paralelos, precisa ser tratado pelo seu programa paralelo.

O modelo de programação MapReduce simplifica muito o código de execução em paralelo, já que você não precisa lidar com nenhum desses problemas. Em vez disso, você só precisa criar e mapear e reduzir tarefas, e você não precisa se preocupar com vários threads, sincronização ou problemas de simultaneidade. Então, o que é um programa MapReduce? Map e Reduce são dois conceitos baseados na programação funcional onde a saída da função é baseada exclusivamente na entrada. Assim como em uma função matemática, f(x) = y, y depende de x. Você fornece uma função, ou operação para um mapa, e reduz. E o tempo de execução executa-o sobre os dados.

Para o mapear, a operação é aplicada em cada elemento de dados. E em reduzir, a operação resume os elementos de alguma maneira. Usar o MapReduce em um exemplo tornará esses conceitos mais claros. “Olá

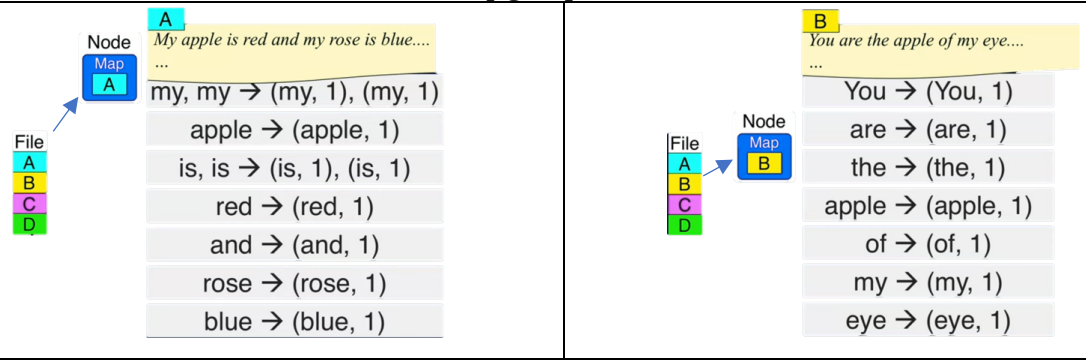
mundo" é um primeiro programa tradicional que você codifica quando você começa a aprender linguagens de programação. O primeiro programa a aprender, ou "Olá mundo" de MapReduce, é muitas vezes WordCount. WordCount lê um ou mais arquivos de texto e conta o número de ocorrências de cada palavra nesses arquivos. A saída será um arquivo de texto com uma lista de palavras e suas frequências de ocorrência nos dados de entrada.

Vamos examinar cada etapa do WordCount. Para simplificar estamos assumindo que temos um arquivo grande como entrada. Antes do WordCount ser executado, o arquivo de entrada é armazenado no HDFS. Como sabemos, HDFS particiona os blocos em vários nós no cluster. Nesse caso, quatro partições rotuladas, A, B, C e D. A primeira etapa no MapReduce é executar uma operação de mapa em cada nó. Como as partições de entrada são lidas a partir de HDFS, Map é chamado para cada linha na entrada. Vamos olhar para as primeiras linhas das partições de entrada, A e B, e começar a contar as palavras. A primeira linha, na partição no nó A, diz: "My apple is red and my rose is blue". Da mesma forma, a primeira linha, na partição B, diz, "You are the apple of my eye". Vamos agora ver o que acontece no primeiro nó de mapa para a partição A. Map cria um valor chave para cada palavra na linha que contém a palavra como a chave, e 1 como o valor.



Neste exemplo, a palavra "apple" é lida a partir da linha na partição A. Map produz um valor chave de (apple, 1). Da mesma forma, a palavra "my" é vista na primeira linha de A duas vezes. Então, os valores-chave de (my, 1), são criados. Note que o mapa vai para cada nó contendo um bloco de dados para o arquivo, em vez de os dados serem movidos para o mapa. Isso está movendo a computação para os dados. Vamos agora ver o que a mesma operação de mapa gera para a partição B. Uma vez que cada palavra ocorre apenas uma vez, uma lista de todas as palavras com um emparelhamento chave-valor cada é gerada.

Passo 1 - Map gera pares Chave-valor

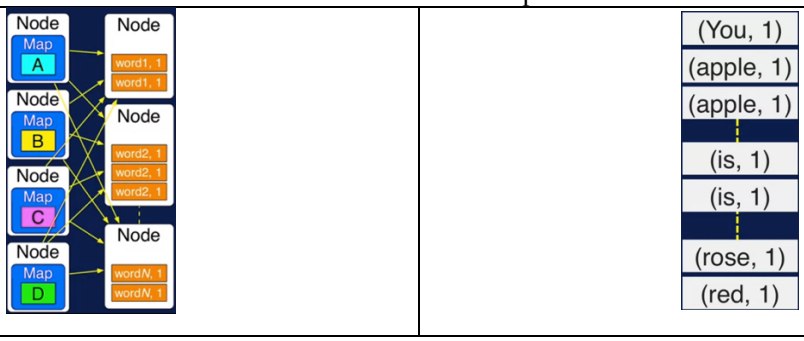


Observe as saídas do Map e cada par chave-valor associado a uma palavra. Em seguida, todos os valores-chave que foram saídas do mapa são classificados com base em sua chave. E os valores de chave, com a mesma palavra, são movidos ou embaralhados, para o mesmo nó. Para simplificar essa figura, cada nó tem apenas uma única palavra, em caixas laranja. Mas, em geral, um nó terá muitas palavras diferentes. Assim como o nosso exemplo das duas linhas nas partições A e B. Aqui vemos que, "you" e a "Apple", são atribuídos ao primeiro

nó. A palavra “is”, para o segundo nó. E as palavras, “rose” e “red”, para o terceiro. Embora, por simplicidade, desenhamos quatro nós de mapa e três nós embaralhados. O número de nós pode ser estendido tanto quanto o aplicativo exige.

Passo 2 - Ordenação

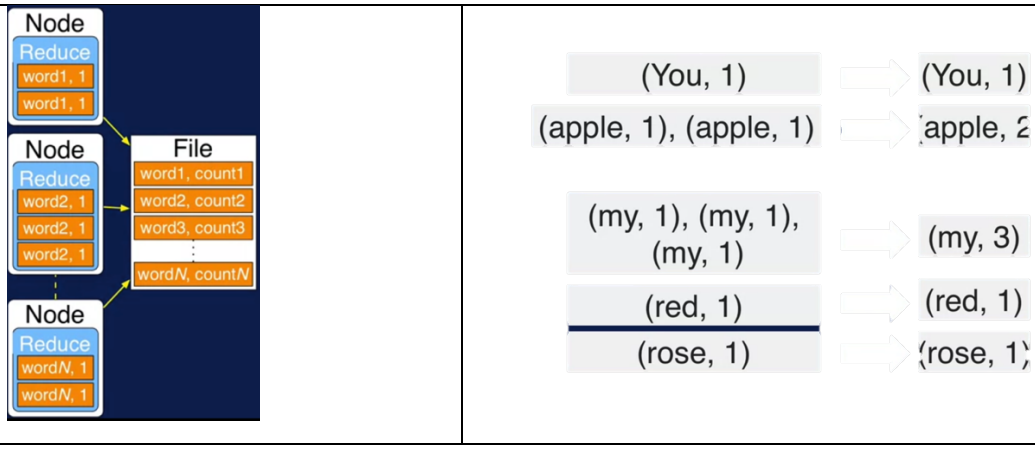
Pares com o mesmo valor são movidos para um mesmo nó



Em seguida, a operação de Reduce é executada nesses nós para adicionar valores para pares chave-valor com as mesmas chaves. Por exemplo, (apple, 1), e outro (apple, 1), torna-se (apple, 2). O resultado do Reduce é um único par de chaves para cada palavra que foi lida no arquivo de entrada. A chave é a palavra e o valor é o número de ocorrências.

Passo 3 - Reduzir

Adicionar valores com as mesmas chaves



Se olharmos para trás para o nosso exemplo WordCount, vemos que havia três etapas distintas. Ou seja, a etapa do mapa, a etapa aleatória e classificação e a etapa de redução. Embora, o exemplo WordCount seja bastante simples, representa um grande número de aplicativos aos quais esses três passos podem ser aplicados a fim de alcançar a escalabilidade paralela de dados.

Por exemplo, agora que você viu o aplicativo WordCount, considere alterar o algoritmo WordCount para indexar todos os URLs por palavras após um rastreamento na web. Isso significa que, em vez de apontar para um número, as chaves se referem a URLs. Após o Map, com esta nova função, que por sinal é chamada de função definida pelo usuário, a saída de shuffle e sort ficaria assim.

Agora, quando reduzimos os URLs, todos os URLs que mencionam a “Apple” ficariam assim. Esta é, de fato, uma das maneiras como um mecanismo de busca como o Google funciona. Então agora, se alguém veio para a interface construída para este aplicativo, para procurar a palavra “apple”, e entrou “apple”, seria fácil obter todas as URLs como a própria palavra. Não admira que o primeiro papel do MapReduce tenha sido produzido pelo Google.

Acabamos de ver como MapReduce pode ser usado em motores de busca além de contar as palavras e documentos. Embora seja possível adicionar muitos outros aplicativos, vamos parar aqui para uma discussão

geral sobre como os pontos do paralelismo de dados podem ser usados na pesquisa neste padrão de três etapas. Definitivamente há paralelização durante a etapa do Map. Esta paralelização é sobre a entrada, como cada partição é processada uma linha de cada vez. Para alcançar este tipo de paralelismo de dados devemos decidir em a granularidade de dados de cada concorrência paralela. Neste caso, será uma linha. Também vemos agrupamento paralelo de dados na fase aleatória e de classificação. Desta vez, a paralelização é sobre os produtos intermediários. Ou seja, os pares de chave-valor individuais. E depois do agrupamento dos produtos intermediários, a etapa de Reduce fica paralelizada para construir um arquivo de saída.

Você provavelmente notou que os dados são reduzidos para um conjunto menor em cada etapa. Esta visão geral nos deu uma ideia de quais tipos de tarefas que MapReduce é bom. Enquanto MapReduce se destaca em tarefas de lote independentes semelhantes às nossas aplicações, há certos tipos de tarefas que você não gostaria de usar MapReduce para. Por exemplo, se os seus dados são frequentemente alterados, MapReduce é lento, uma vez que lê todo o conjunto de dados de entrada de cada vez.

O modelo MapReduce requer que Map's e Reduce's executem independentemente um do outro. Isso simplifica muito seu trabalho como designer, já que você não precisa lidar com problemas de sincronização. No entanto, significa que os cálculos que têm dependências, não podem ser expressos com MapReduce.

Finalmente, MapReduce não retorna nenhum resultado até que todo o processo seja concluído. Ele deve ler todo o conjunto de dados de entrada. Isso o torna inadequado para aplicativos interativos onde os resultados devem ser apresentados ao usuário muito rapidamente, esperando um retorno do usuário. MapReduce esconde complexidades de programação paralela e simplifica muito a construção de aplicativos paralelos. Muitos tipos de tarefas adequadas para MapReduce incluem classificação de página do mecanismo de pesquisa e mapeamento de tópicos. Por favor, veja a leitura após esta palestra sobre como fazer molho de massa com MapReduce para outra aplicação divertida usando o modelo de programação MapReduce.

2.6. Quando Reconsiderar o Hadoop

O ecossistema Hadoop está crescendo em um ritmo acelerado. Isso significa que muitas coisas que foram difíceis, ou que não apoiaram, estão se tornando possíveis. Mas em que situações os principais recursos que tornam um problema Hadoop amigável?

Se você vir um crescimento em grande escala na quantidade de dados que irá abordar, provavelmente faz sentido usar o Hadoop. Quando você deseja acesso rápido aos seus dados antigos que, de outra forma, seriam utilizados em drives de fita para armazenamento de arquivamento, o Hadoop pode fornecer uma boa alternativa. Outros recursos amigáveis do Hadoop incluem cenários quando você deseja usar vários aplicativos no mesmo armazenamento de dados. Alto volume ou alta variedade também são ótimos indicadores para Hadoop como uma opção de plataforma. O processamento do conjunto de dados pequeno deve levantar as sobrancelhas. Você realmente precisa do Hadoop para isso? Cave mais fundo e descubra exatamente por que você deseja usar o Hadoop antes de seguir em frente.

Hadoop é bom para paralelismo de dados. Como você sabe, o paralelismo de dados é a execução simultânea da mesma função em vários nós através dos elementos de um conjunto de dados. Por outro lado, paralelismo de tarefas, como você vê neste gráfico, é a execução simultânea de muitas funções diferentes em vários nós através dos mesmos conjuntos de dados ou diferentes. Se o seu problema tiver paralelismo em nível de tarefa, você deve fazer mais uma análise sobre quais ferramentas você planeja implantar a partir do ecossistema Hadoop.

Quais são os benefícios precisos que essas ferramentas oferecem? Prossiga com cautela. Nem todos os algoritmos são escaláveis no Hadoop, ou redutível para um dos modelos de programação suportados pelo YARN. Portanto, se você estiver procurando implantar algoritmos de processamento de dados altamente acoplados proceda com cautela. Faça uma análise completa antes de usar o Hadoop.

Você está pensando em jogar fora suas soluções de banco de dados existentes e substituí-las pelo Hadoop? Pense novamente. Hadoop pode ser uma boa plataforma onde seus diversos conjuntos de dados podem pousar e são processados em uma forma digestível com seu banco de dados. Hadoop pode não ser a melhor solução de armazenamento de dados para o seu caso de negócios.

Avalie e prossiga com cautela. HDFS armazena dados em blocos de 128 megabytes ou mais, para que você tenha que ler um arquivo inteiro apenas para escolher uma entrada de dados. Isso torna um pouco mais difícil executar acesso aleatório a dados.

O ecossistema Hadoop está crescendo a um ritmo mais rápido do que nunca. Principalmente, consultas analíticas avançadas, tarefas sensíveis à latência e cibersegurança de dados confidenciais. Aqui, nós damos dicas para ferramentas que você pode querer olhar mais para entender os desafios que essas ferramentas precisam abordar.

Embora o Hadoop seja bom com escalabilidade de muitos algoritmos, é apenas um modelo e não resolve todos os problemas no gerenciamento e processamento de big data. Embora seja possível encontrar contra-exemplos, geralmente podemos dizer que a estrutura do Hadoop não é a melhor para trabalhar com pequenos conjuntos de dados, algoritmos avançados que exigem um tipo de hardware específico, paralelismo de nível de tarefa , substituição de infraestrutura ou acesso aleatório a dados.

2.7. O Desafio do Processamento

O volume de dados também traz implicações sobre como os dados serão processados. Os mecanismos desenvolvidos para aplicações que processam grandes volumes de dados, como o Hadoop MapReduce, oferecem escalabilidade e desempenho às soluções. Contudo, o Hadoop MapReduce não é uma solução adequada para todas as aplicações, pois foi desenvolvido para lidar com processamento em lote.

O processamento em lote refere-se ao processamento agendado de conjuntos de dados. Grandes massas de dados são coletadas por um período de tempo e agrupados para o processamento através de um job. Essa abordagem, conhecida como *um-para-muitos*, onde uma requisição é responsável pelo processamento de um grupo inteiro de dados. Os processamentos de transações diárias de cartões de crédito, o fechamento de folhas de pagamento, o processamento das declarações de imposto pela receita são exemplos de processamento em lote.

Conforme apresentado, além da variedade e volume de dados, Big Data também está relacionado à velocidade dos dados. Não apenas com a velocidade de geração, mas com a rapidez na geração de respostas - análise e consumo de dados. Portanto, há situações em que o processamento deve ser feito no momento em que a massa de dados chega ao sistema, em tempo real.

Em um processo comum de processamento em lote, as fases de coleta, armazenamento e processamento dos dados ocorrem em momentos distintos. O processo de captura e armazenamento de dados pode ter sido realizado dias, meses ou anos antes dos dados serem processados. Isto porque o fluxo de processamento não é contínuo, de forma que o processamento se encerra após computar todo o conjunto de dados definido no início do processamento. Por se tratar de um conjunto histórico de dados, é comum processar lotes de tamanhos massivos, podendo demorar minutos, horas ou até dias até que o trabalho seja completamente finalizado.

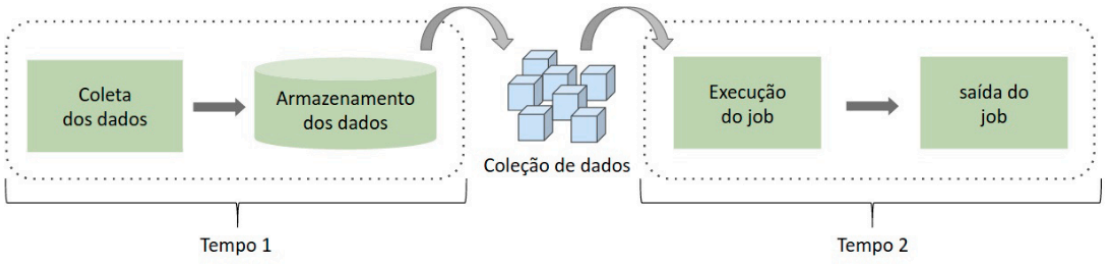


Figura 1 Fluxo de processamento em lote

O processamento em lote está relacionado ao aspecto da latência da aplicação: o tempo que o sistema demora para responder a um evento. Já a abordagem de processamento real, ou processamento próximo ao tempo real, estamos falando de processamento de baixa latência, na ordem de milissegundos ou segundos. Essa necessidade ocorre em uma série de aplicações de Big Data.

No cenário de processamento em tempo real, os dados são analisados assim que são gerados, criando a oportunidade de extrair informações imediatas sobre eles. Neste caso, o processo não é feito sobre um conjunto de dados, e sim sobre um item de dado apenas. Por exemplo, no processamento de dados oriundos de um sensor de temperatura, a cada informação recebida, deve-se imediatamente analisar se o item de dado obtido excede o valor definido como aceitável para a temperatura. Este tipo de processamento é chamado *um-para-um*, a requisição do processamento é individual para cada item de dado.

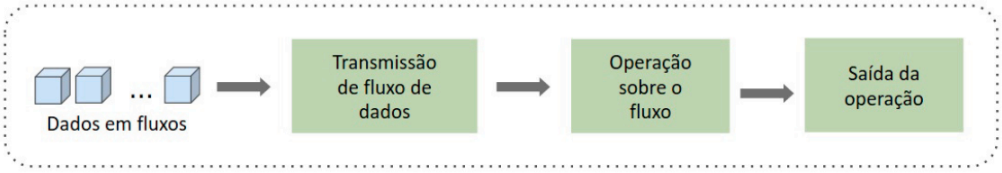


Figura 2 Fluxo de processamento em tempo real

Como deve ocorrer rapidamente, são atribuídas apenas pequenas manipulações a esses dados, como cálculos simples. Para que não haja gargalo de latência de I/O dos discos, uma estratégia muito comum é manter esses dados em memória até que sejam processados, e então sendo persistidos em um banco após o processamento. Esse tipo de processamento também permite realizar uma análise do dado recém-adquirido com os dados já persistidos no banco, porém, isso pode adicionar uma maior latência. São características do processamento em tempo real:

- **Baixa latência**: o tempo de entrega da resposta é crucial. A latência se acumula na medida em que os atrasos nas entregas das respostas a itens de dados ocorrem.
- **Consistência**: sistemas em tempo real devem ser capazes de lidar com imperfeições. Como os dados são processados assim que chegam da coleta, é normal que ocorram problemas, como atrasos, inversão de sequência, até mesmo perda de parte dos dados. A aplicação deve ser capaz de lidar com inconsistências.
- **Alta disponibilidade**: sistemas de processamento em tempo real podem sofrer grande impacto caso as etapas de coleta, transmissão e processamento de dados fiquem indisponíveis por um determinado tempo. A indisponibilidade de sistemas em tempo real pode ocasionar na perda de dados significantes para a aplicação.

Embora inúmeros fatores possam causar a indisponibilidade dos dados e não seja possível prever todos os problemas, existem mecanismos que buscam amenizar essas ocorrências. A maioria das aplicações adotam recursos de distribuição e replicação, na qual o serviço de processamento é dividido em um conjunto de máquinas, para que, caso uma venha falhar, outra máquina possa realizar o processamento.

Devido a esses requisitos, muitos dos recursos utilizados vão em direção ao uso de sistemas distribuídos. Isso ocorre principalmente no contexto de Big Data, em que o fluxo de dados ocorre em uma escala massiva. Para construir um ambiente computacional adequado para esse tipo de processamento, porém, é necessário verificar as necessidades específicas da aplicação.

Assim como temos o teorema CAP para as tecnologias de armazenamento de dados NoSQL, os autores Thomas Erl, Wajid Khattak e Paul Buhler apresentam no livro Big Data Fundamentals: Concepts, Drivers & Techniques o conceito SVC para sistemas distribuídos de processamento de dados em tempo real. São apresentadas 3 características desses sistemas: velocidade (S — Speed), consistência (C — Consistency) e volume de dados (V — Volume).

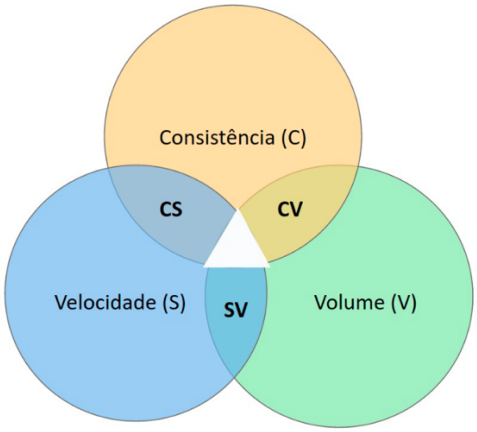


Figura 3 Teorema SCV

Conforme ilustrado na figura, assim como no teorema CAP, no teorema SCV somente duas das três características podem ser atendidas em conjunto. Um sistema que requer velocidade e consistência não consegue atender ao requisito volume de dados, visto que não terá tempo hábil para processar em poucos segundos um grande volume de dados.

Entretanto, se o volume de dados a ser processado é grande e a consistência deve ser mantida, então não é possível garantir a velocidade do processamento. Esse teorema pode ser utilizado como um guia para se definir as prioridades e limitações da aplicação a ser desenvolvida.

É comum que os mesmos dados processados em tempo real também sejam posteriormente persistidos em um disco rígido. Assim, permite-se a execução de análises mais complexas sobre eles, agora no modelo de processamento em lote.

Essa abordagem é muito utilizada em soluções que trabalham com o uso de aprendizado de máquina, pois na maioria dos casos essas soluções necessitam de um grande volume de dados histórico para a construção de modelos. Para isso, torna-se necessário uma infraestrutura que permita tanto a baixa latência do processamento das informações quanto tecnologias que ofereçam uma alta vazão para acesso aos dados.

Cenários de utilização do processamento em tempo real

Imagino que você esteja interessado em saber quando o processamento em tempo real é utilizado. A seguir, são apresentados alguns exemplos.

Dados da Web

Vivemos em um mundo cada vez mais digital, onde assistimos a um filme, ouvimos uma música, estudamos, jogamos e compramos pela Web. Por esse motivo, surge cada vez mais a necessidade de empresas

que oferecem esses serviços rastrearem os eventos ocorridos nesse contexto. Em que momento o cliente desistiu de uma compra? Quais produtos ele avaliou antes de decidir qual produto comprar? Qual o número de visitas no site? O serviço está funcionando corretamente?

Para responder perguntas como essas, os provedores desses serviços utilizam técnicas de processamento de registros de log. Com o advento de Big Data, esse processamento tende a ser cada vez mais automatizado, dada a imensa quantidade de registros a serem analisados. Essa análise de dados já ocorre há muitos anos, todavia, anteriormente ocorria de forma mais lenta. Porém, diante desse cenário atual cada vez mais competitivo, as empresas estão buscando cada vez mais agilizar o processo de extração de conhecimento dos dados. Por esse motivo, elas estão adotando tecnologias que permitam o processamento em tempo real, para assim agir imediatamente de acordo com o que foi observado.

Detecção de fraude

Com o processamento em tempo real, decisões podem ser tomadas no momento em que uma fraude é identificada ou até mesmo em momentos que a antecede, por meio de modelos preditivos. O processamento em tempo real tem sido então aplicado em inúmeras aplicações para identificação de fraude, tais como em ligações de números de emergência, em transações de cartão de crédito e no mercado de seguros.

Modelos são gerados por meio do rastreamento de inúmeras variáveis, como geolocalização, informações de crédito e perfil em redes sociais, que fornecem insights para identificar uma possível ação fraudulenta assim que um evento é gerado e um novo dado é recebido.

Redes sociais

Sabemos que as redes sociais têm sido uma fonte inesgotável de informações. O processamento em tempo real aplicado aos dados gerados dessas redes sociais tem sido cada vez mais significante. Um exemplo é a identificação atualizada de tendências.

A análise imediata das informações compartilhadas pelas redes permite gerar descobertas como pandemias de doenças em determinadas regiões. Ou seja, médicos e pacientes podem ter um panorama praticamente imediato da saúde da população em determinadas regiões.

Isso é um grande avanço, visto que em muitos casos se gastava dias para realizar essa verificação, fazendo com que os dados analisados já estivessem desatualizados. Esse é apenas um dos exemplos do benefício do processamento em tempo real de dados oriundos de redes sociais. Os dados gerados por humanos fornecem uma valiosa fonte de informação que, quando avaliados em tempo real, podem gerar conhecimentos imensuráveis.

3. Definindo Bancos de Dados, Tabelas e Colunas

Neste capítulo, você aprenderá a criar bancos de dados e tabelas no ambiente Hadoop. Lembre-se que com Hive e Impala, cada tabela tem dois componentes. Um deles é seus metadados que é armazenado no *metastore*, e dois seus dados que poderiam ser armazenados dizer no HDFS ou no S3. Discutiremos tudo sobre os metadados. Os metadados de uma tabela incluem os nomes das colunas e seu tipo de dados, e também coisas como onde os dados são armazenados e em que formato eles estão. Então, para criar uma tabela, você precisa especificar esses metadados e armazená-los no metastore.

É possível fazer isso usando as ações apontar e clicar no navegador de tabelas no Hue, para que você aprenda a fazer isso. Mas, em muitos casos, é melhor usar comandos SQL para criar bancos de dados e tabelas. Então, você aprenderá a usar as instruções **CREATE DATABASE** e **CREATE TABLE**.

Você deve lembrar que essas instruções estão em uma categoria de instruções SQL chamada *Data Definition Language* ou DDL. Cada SGBD suporta comandos DDL, mas há grandes diferenças no DDL entre diferentes sistemas gerenciadores. O que você aprenderá neste curso aplica-se apenas ao Hive e ao Impala.

A instrução CREATE DATABASE é muito simples e não há muito a aprender sobre isso. Então você vai passar a maior parte deste capítulo aprendendo sobre a instrução CREATE TABLE que tem muitas cláusulas e opções. Como parte da instrução CREATE TABLE, você aprenderá a formatação padrão e localização dos arquivos de dados para uma tabela, e como especificar quando você deseja algo diferente do padrão. Você aprenderá sobre alguns atributos avançados de tabela, incluindo o uso de Serializer Deserializers ou SerDES.

Às vezes, você pode cometer erros ao criar uma tabela ou mudar de idéia sobre alguns dos específicos. Assim, você aprenderá como fazer alterações nos esquemas de tabela e o que isso significa para os dados em si. Finalmente, você aprenderá um pouco mais sobre como o Hive e o Impala funcionam, incluindo como se mover entre os dois mecanismos quando estiver gerenciando ou trabalhando com as tabelas na metastore.

3.1. Criando Bancos de Dados e Tabelas

3.1.1 Criando bancos de dados e tabelas com Hue

No Hive e Impala, cada tabela pertence a um banco de dados específico. Os bancos de dados são úteis para organizar tabelas; usando vários bancos de dados, você pode ter tabelas com o mesmo nome em bancos de dados diferentes. Dessa forma, eles servem como ***namespaces***. Mesmo que você não use tabelas com o mesmo nome, separar os bancos de dados é útil para a organização e para restringir o acesso do usuário a dados que eles não precisam acessar ou não devem acessar.

Criar bancos de dados e tabelas usando o Table Browser do Hue é bastante fácil. À medida que você passa por essa leitura, ou depois de lê-la uma vez, use a VM para criar alguns bancos de dados e tabelas de teste. Você pode descartar (excluir) os bancos de dados e tabelas de teste quando terminar.

Para executar as etapas descritas nesta seção, você precisará usar o Hue na VM. Se você ainda não tiver a VM instalada e em execução, siga as instruções na leitura Baixando e Instalando a VM na primeira semana deste curso. Em seguida, abra o navegador da Web na VM e clique no link para Hue na barra de ferramentas de favoritos.

Criando um novo banco de dados

Você pode iniciar o processo de criação de um banco de dados acessando diretamente o Navegador de Tabelas ou usando o painel de origem de dados. Experimente os dois e veja qual você prefere.

Quando você cria um banco de dados usando esses métodos, Hive ou Impala cria o diretório **/user/hive/warehouse/databasename.db** (onde ***databasename*** é o nome que você digitou), a menos que um local diferente seja especificado. As tabelas criadas em um banco de dados serão colocadas como um subdiretório dentro desse diretório de banco de dados.

Usando o Browser Table [Navegador de Tabelas]

Entre no Table Browser clicando no menu de hambúrguer (três linhas horizontais) e escolhendo **Browsers > Tables**. O painel principal (centro) mostrará o banco de dados **default** para Hive e Impala. Para adicionar um novo banco de dados:

1. Clique em **Databases** no breadcrumbs na parte superior. (Veja a Figura 1 abaixo.)
2. Passe o mouse sobre o símbolo **+** na extrema direita; deve dizer **Create a new database**. Clique nesse símbolo.
3. Dê ao seu banco de dados um nome como **test** e (opcionalmente) uma descrição no campo apropriado.
4. Opcional: Se desejar armazenar os arquivos de banco de dados em um local diferente do HDFS (no S3, por exemplo), desmarque a caixa **Default Location** e forneça o local.
5. Clique em **Submit** (Enviar).
6. O pop-up Histórico de Tarefas aparecerá; o topo deve dizer **Creating database name** com uma linha verde embaixo. (Veja a Figura 2 abaixo.) Clique no **x** à direita para fechar a janela.
7. Verifique se a criação foi bem-sucedida clicando em **Databases** no breadcrumbs e observe que seu novo banco de dados agora aparece na lista. Se desejar, você pode verificar a estrutura do arquivo HDFS para ver se **/user/hive/warehouse/name.db** existe.
8. Para descartar (excluir) seu banco de dados de teste, marque a caixa ao lado dele e clique no botão **Drop** acima da lista e clique em **Yes**. Isso descartará o banco de dados e excluirá o diretório do HDFS. Tenha cuidado para não descartar nenhum banco de dados que contenha tabelas!

Figura 4

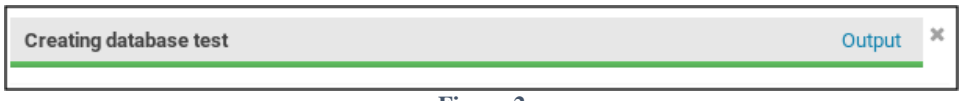


Figura 2

Usando o painel de fonte de dados

O painel da fonte de dados aparece com todas as áreas do Hue, para que você possa usar facilmente esse método sem a necessidade de alternar primeiro para o **Browser Table** (Navegador de tabela).

- Certifique-se de que o painel da fonte de dados esteja no modo de banco de dados (e não no modo de arquivos). O ícone do banco de dados (discos empilhados) deve ser azul. (Veja a Figura 3 abaixo.)
- Navegue até a fonte do Impala. (Veja a Figura 4 abaixo. Você também pode usar o Hive, e o banco de dados estará disponível para ambos os mecanismos.)

- Passe o mouse sobre o símbolo **+**; deve dizer **Create database**. Clique nesse símbolo. (Observação: a partir deste ponto, as etapas são as mesmas da seção anterior.)
- No painel principal, dê ao seu banco de dados um nome como **test** e (opcionalmente) uma descrição nos campos apropriados.
- Opcional: Se desejar armazenar os arquivos de banco de dados em um local diferente do HDFS (no S3, por exemplo), desmarque a caixa **Default Location** (Local Padrão) e forneça o local.
- Clique em **Submit** (Enviar). Como na seção anterior, o pop-up Histórico de Tarefas aparecerá. Clique no **x** para descartá-lo.
- Você estará agora no Table Browser, mas no banco de dados padrão. Verifique se a criação foi bem-sucedida clicando em Bancos de dados no breadcrumbs de navegação e observe que seu novo banco de dados agora aparece na lista. Você também pode navegar no painel da fonte de dados de volta à fonte Impala e ver a lista de bancos de dados lá.
- Para descartar seu banco de dados de teste, marque a caixa ao lado dele e clique no botão **Drop** (Soltar) acima da lista e, em seguida, clique em **Yes**.

Figura 3

Figura 4

Criando uma nova tabela vazia

Quando você cria uma tabela, Hive ou Impala cria o diretório **/user/hive/warehouse/tablename** ou, se estiver em um banco de dados diferente do padrão, **/user/hive/warehouse/databasename.db/tablename**. Este diretório estará vazio no início; você aprenderá em uma semana posterior do curso como preencher a tabela com dados carregando arquivos de dados nela.

Você aprenderá como descartar essas tabelas em uma lição posterior desta semana, então não se preocupe em descartar os exemplos que você criar agora.

Assim como na criação do banco de dados, você pode iniciar este processo com o painel de fonte de dados (Opção A, abaixo), ou pode ir diretamente para o **Table Browser** (Navegador de Tabelas) (Opção B). Decida o que você deseja experimentar com base em suas experiências de criação de bancos de dados de teste:

- Opção A: No painel da fonte de dados à esquerda, navegue até o banco de dados onde deseja colocar a tabela. Nesse caso, escolha o banco de dados **default** na origem do Impala. Passe o mouse sobre o símbolo **+**; deve dizer **`Create table`**. (Veja a Figura 5.) Clique nesse símbolo.

Figura 5

- Opção B: Entre no Table Browser clicando no menu hambúrguer (três linhas horizontais) e escolhendo **Browser > Tables**. O painel principal (centro) mostrará o banco de dados **default**. (Você pode alternar para um banco de dados diferente, se quiser, mas isso não é necessário para fins de teste. Permaneça no banco de dados **default**.) Novamente, passe o mouse sobre o símbolo **+** na extrema direita; deve dizer **Create a new table**. Clique nesse símbolo.

No painel principal, você tem a oportunidade de indicar um arquivo no HDFS que será o dado desta tabela. Isso pode ser muito útil, porque o Hue tentará adivinhar as colunas, incluindo seus tipos de dados, com base nesse arquivo. Você verá isso mais adiante neste curso, então, por enquanto, você criará uma tabela vazia sem dados.

1. Clique no campo Type e altere para Manually (Manualmente); em seguida, clique no botão **`Next`**.
2. Na área **`DESTINATION`**, dê um nome à sua tabela. Para este exemplo, use **default.test** ou apenas **test** se você está no banco de dados **default**.
3. Para **`PROPERTIES`**, você pode definir o formato do arquivo e o local de armazenamento, se desejar. Por enquanto, deixe os padrões (Formato de texto e Armazenar no local padrão marcados [*Store in Default location*]).
4. Em **`FIELDS`**, você pode especificar as colunas da sua tabela; clique em +Add Field [Adicionar campo] para cada coluna e especifique o nome e o tipo de dados. Dê à sua tabela de teste algumas colunas, como **id** e **title**. Você pode deixar o tipo de campo como **string** para cada um. Você aprenderá mais sobre como escolher tipos de dados posteriormente neste curso.
5. Clique em **`Submit`**. O pop-up Histórico de Tarefas aparecerá; a parte superior deve dizer **`Creating table name`** [Criando o nome da tabela] (para o caso de teste, o ***name*** é **default.test**) com uma linha verde embaixo. Clique no **x** à direita para fechar a janela.
6. Verifique se a criação foi bem-sucedida observando o painel de origem de dados. Navegue até o banco de dados (padrão do Impala), se necessário, e observe que sua nova tabela agora aparece na lista. Talvez seja necessário clicar no botão atualizar (duas setas curvas) para atualizar a exibição; escolha **Clear cache** [Limpar cache] se você fizer isso.
7. Para seu teste, verifique a estrutura do arquivo HDFS para ver se **/user/hive/warehouse/test/** existe. Este diretório estará vazio, porque a tabela não contém dados.

Criando uma nova tabela para consultar dados existentes

As etapas acima descreveram como criar uma tabela vazia, sem dados nela. Você também pode usar o Hue para criar uma tabela para consultar dados que já existem no HDFS.

Por exemplo, no diretório HDFS **/old/castles/**, há um arquivo chamado **castles.csv**. Revise este arquivo: Os campos são separados por vírgulas, os nomes das colunas (**name** e **country**) são fornecidos na linha do

cabeçalho e ambas as colunas contêm dados de cadeia de caracteres. Usando as etapas abaixo, você pode criar uma tabela para consultar esses dados.

1. Usando o painel de fonte de dados (no modo de banco de dados) ou o **table browser** [navegador de tabela], clique no símbolo **+** para criar uma nova tabela.
2. Clique no campo **`Type`** e altere para **`Manually`** [Manualmente]; em seguida, clique no botão **Next**.
3. Na área **`DESTINATION`**, dê um nome à sua table. Para sua tabela de exemplo, use **`default.castles`**.
4. Para **`PROPERTIES`**, você pode definir o formato do arquivo. O arquivo de exemplo é um arquivo de texto, portanto, deixe o formato padrão (**Text**) marcado.
5. Desmarque a caixa de seleção **`Store in Default location`** [Armazenar no local padrão]. Ao fazer isso, você verá um campo **`External location`** [Local externo] aparecer abaixo dele.
6. Insira o caminho do diretório HDFS (**/old/castles** para o exemplo) no campo **`External location`** [Local externo]. Alternativamente, você pode clicar no ícone **..** no lado direito e usar a caixa de diálogo para selecionar esta pasta. (Veja a Figura 6 abaixo.)
7. Logo abaixo, clique no ícone **`Extras`**, que se parece com três controles deslizantes. Você pode adicionar uma descrição opcional do local dos dados no campo **`Description`** (deixe em branco por enquanto). Você também pode especificar o caractere que separa os campos aqui marcando a caixa de seleção **`Custom char delimiters`** [Delimitadores de caracteres personalizados]. Marque a caixa e observe que três menus suspensos aparecem. Você pode especificar diferentes tipos de delimitadores aqui. Para este exemplo, você só precisa do menu suspenso **Field**. O separador de campo já está definido como **`Comma`** **[Vírgula] (,)** que é o separador de campo (delimitador) usado no arquivo **`castles.csv`**, portanto, mantenha essa seleção.
8. Em **FIELDS** [Campos], especifique as colunas desta tabela; clique em **+Add Field** para cada coluna e especifique o nome e o tipo de dados. Lembre-se de que o arquivo **castles.csv**, que contém os dados da tabela de exemplo, possui duas colunas: **name** e **country**. Adicione ambos (nessa ordem), ambos como tipos de **string**.
9. Clique em **Submit**. O pop-up Histórico de Tarefas aparecerá; o topo deve dizer **Creating table dbname.tablename** [Criando tabela dbname.tablename] com uma linha verde embaixo. Clique no **x** à direita para fechar a janela.
10. Verifique se a criação foi bem-sucedida observando o painel de origem de dados. Navegue até o banco de dados (padrão Impala), se necessário, e observe que a nova tabela (**castles**) agora aparece na lista. Talvez seja necessário clicar no botão atualizar (duas setas curvas) para atualizar a exibição; escolha **Clear cache** [Limpar cache] se você fizer isso.
11. Passe o cursor sobre o nome da tabela e clique no ícone **i** à direita do nome da tabela. Clique na guia **Sample** [Amostra] para verificar se os dados aparecem nos resultados da amostra.

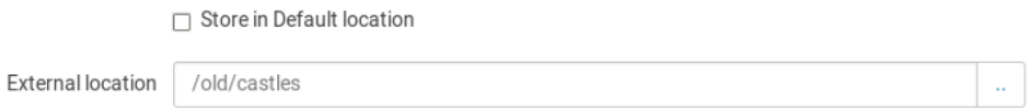


Figure 6

Limitações

Conforme descrito acima, o Hue oferece várias opções para criar bancos de dados e tabelas. No entanto, essas opções têm algumas limitações. Por exemplo, ao criar uma nova tabela para consultar dados existentes armazenados em arquivos de texto (como demonstrado acima), o Hue assume que os arquivos de dados possuem

linhas de cabeçalho. Atualmente, não há como especificar no Hue que os arquivos de dados não possuem linhas de cabeçalho.

Para superar essas e outras limitações do Hue, você pode usar comandos SQL para criar bancos de dados e tabelas. Além disso, usar comandos SQL (em vez de ações da interface de usuário do Hue) é uma maneira mais sistemática de executar tarefas como essa. Os comandos SQL podem ser roteirizados, automatizados e agendados. Ao salvar seus comandos SQL em um arquivo, você pode documentar efetivamente as etapas executadas e tornar as etapas reproduzíveis.

Aprenda agora como criar bancos de dados e tabelas executando instruções SQL.

3.1.2 Criando bancos de dados e tabelas com SQL

Você aprendeu como usar o Table Browser no Hue para criar bancos de dados e tabelas por meio de ações de apontar e clicar. Esse método às vezes é conveniente, mas geralmente é melhor usar comandos SQL para criar bancos de dados e tabelas.

As instruções SQL **CREATE DATABASE** e **CREATE TABLE** fornecem uma maneira mais sistemática de criar bancos de dados e tabelas. Esses comandos oferecem maior poder e flexibilidade e podem ser usados em outras interfaces Hive e Impala além do Hue. Usando esses comandos, você também tem a opção de criar scripts e automatizar a criação de bancos de dados e tabelas, tornando essas tarefas mais reprodutíveis.

À medida que você passa por essa leitura, ou depois de lê-la uma vez, use a VM para criar alguns bancos de dados e tabelas de teste. Você pode descartar os bancos de dados e as tabelas quando terminar.

Para executar as etapas descritas nesta leitura, você precisará usar o Hue na VM. Se você ainda não tiver a VM instalada e em execução, siga as instruções na leitura Baixando e Instalando a VM na primeira semana deste curso. Em seguida, abra o navegador da Web na VM e clique no link para Hue na barra de ferramentas de favoritos.

Criando um banco de dados

A criação do banco de dados é tão simples quanto a etapa 2 abaixo. A etapa 1 é apenas para levá-lo ao local onde você pode executar essa etapa, e a etapa 3 é a verificação do sucesso.

1. Primeiro, acesse o editor de consultas Impala na VM clicando no botão **Query** [Consulta]. Como o Hive e o Impala compartilham o metastore do Hive, ambos funcionarão, mas usar o editor de consulta do Impala é mais fácil. (Se você usar o Hive, o Impala não reconhecerá imediatamente o novo banco de dados. Você aprenderá mais sobre isso mais tarde nas lições desta semana. Por enquanto, use o Impala em vez do Hive.)
2. Digite e execute o comando: **`CREATE DATABASE test;`** Observação: isso criará um diretório no HDFS no local padrão: **`/user/hive/warehouse/test.db`**.
3. Verifique se a criação foi bem-sucedida usando o painel de fonte de dados à esquerda do painel do editor de consultas. Procure nos bancos de dados Impala ou Hive; você deverá ver seis bancos de dados (**`default, fly, fun, test, toy e wax`**). Se o **test** não aparecer, tente atualizar a tela usando o botão de atualização (duas setas curvas) na parte superior deste painel esquerdo. Se desejar, você pode verificar a estrutura do arquivo HDFS para ver se **`/user/hive/warehouse/test.db`** existe.
4. Para descartar seu banco de dados de teste, digite e execute o comando:

```
DROP DATABASE test;
```

Criando uma Tabela

Esta é apenas uma introdução rápida à instrução **`CREATE TABLE`**, fornecendo a estrutura mais básica. A próxima lição irá detalhar a instrução **`CREATE TABLE`** e cobrir cada uma das cláusulas nela. Não se preocupe com os detalhes ainda.

A etapa 3 abaixo é a etapa de criação real, que será a mesma para qualquer ferramenta que você usar para inserir comandos SQL. As etapas 1 e 2 são para ajudá-lo a navegar até a área apropriada no Hue, e a etapa 4 é a verificação do sucesso.

1. Clique no botão **Query** [Consulta] para acessar o editor de consultas Impala.
2. Verifique o banco de dados ativo e certifique-se de que é o que você deseja para sua nova tabela. Para testar, coloque-o no banco de dados **default**; se necessário, selecione **default** como o banco de dados ativo. (Veja a Figura 7 abaixo.)
3. Digite e execute este comando:

   ```
   CREATE TABLE test (col1 INT, col2 STRING);
   ```

   O nome da tabela vem depois de **`CREATE TABLE`** e, em seguida, uma lista dos nomes das colunas com seus tipos de dados. O comando abaixo cria uma tabela chamada test com duas colunas, uma coluna **integer** chamada **col1** e uma coluna **string** chamada **col2**. (Veja Notas abaixo.)
4. Verifique se a criação foi bem-sucedida usando o painel de fonte de dados à esquerda do painel do editor de consultas. Se o banco de dados selecionado não for o banco de dados **default** do Impala, navegue até ele e verifique se sua tabela **test** está listada. Talvez seja necessário atualizar a exibição clicando no botão atualizar (as duas setas curvas). Se desejar, você pode verificar a estrutura do arquivo HDFS para ver se **/user/hive/warehouse/test** existe.
5. Para descartar a tabela, digite e execute o comando:

   ```
   DROP TABLE test;
   ```

   O diretório da tabela e quaisquer dados que ele possa ter mantido também serão excluídos. Depois de descartar a tabela, verifique se **`/user/hive/warehouse/test`** não existe mais.

```
CREATE TABLE test(col1 INT, col2 STRING);
```

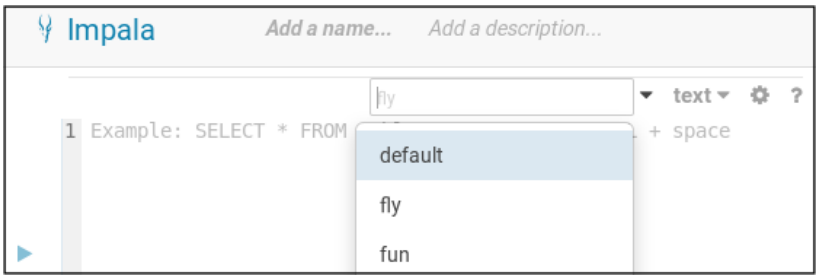


Figura 7

Observações:

A instrução **`CREATE TABLE`** na Etapa 3 cria um subdiretório no **`/user/hive/warehouse/directory`** no HDFS nomeado **test**. Se você colocar isso em um banco de dados diferente, como o banco de dados **fun**, o diretório de teste será um subdiretório do diretório desse banco de dados **`(/user/hive/warehouse/fun.db/`** para o banco de dados divertido).

Você também pode qualificar o nome da tabela com o nome do banco de dados: **CREATE TABLE default.test** .... Isso colocará a tabela chamada **test** no banco de dados **default**, independentemente de qual banco de dados estiver ativo. (Experimente!) Você pode achar esse método preferível e pule a etapa 2

3.1.3 Permissões para criar bancos de dados e tabelas

Na VM dessa especialização, você pode criar bancos de dados e tabelas. Isso requer permissões especiais e, em um ambiente real, talvez você não tenha essas permissões.

Os administradores de um ambiente do mundo real podem usar um comando DCL (Data Control Language), **`GRANT`**, para definir permissões para usuários com base em funções, grupos ou indivíduos. Como isso funciona depende de muitos fatores, incluindo quais ferramentas você está usando e como elas são configuradas. Os detalhes estão além do escopo deste curso, portanto, converse com seu administrador de TI se tiver dúvidas sobre o nível de permissões que você tem em seu ambiente de trabalho.

3.2. A Instrução CREATE TABLE

A instrução **CREATE TABLE** cria uma nova tabela e especifica suas características. Quando você executa um comando **CREATE TABLE**, Hive ou Impala adiciona a tabela ao **metastore** e cria um novo subdiretório no diretório **warehouse** no HDFS para armazenar os dados da tabela. A localização deste novo subdiretório depende do banco de dados no qual a tabela é criada.

Tabelas criadas em um banco de dados padrão são armazenadas em subdiretórios diretamente sob o diretório do **warehouse**. Tabelas criadas em outros bancos de dados são armazenadas em subdiretórios sob esses diretórios de banco de dados.

A sintaxe básica da instrução **CREATE TABLE**, deve ser familiar para qualquer pessoa que tenha criado tabelas em um banco de dados relacional. Depois de criar tabela, você opcionalmente especificar o nome do banco de dados. Em seguida, dê o nome da nova tabela e uma lista das colunas e seus tipos de dados.

```
CREATE TABLE dbname.tablename (col1 TYPE, col2 TYPE, …);
```

Se você omitir o nome do banco de dados, a nova tabela será criada no banco de dados atual. Se você estiver usando Hue, lembre-se de que o banco de dados atual é o que está selecionado no seletor de banco de dados ativo. Se você estiver usando a linha de comando, o banco de dados atual é o que você especificou quando você inicia para ser alinhado ou o shell Impala, ou o que você especificou no comando de uso mais recente na sessão. Se você não especificou um banco de dados atual de uma dessas maneiras, o banco de dados atual será o banco de dados **default**.

A lista de nomes de colunas e tipos de dados está entre parênteses, com cada par de tipo de nome dado como nome da coluna, espaço, tipo de dados. Os pares de tipos de nome são separados por vírgulas. Opcionalmente, você pode incluir quebras de linha após as vírgulas e espaços para recuar as linhas para dividir a lista entre várias linhas. Isso o torna mais legível quando há muitas colunas.

Os nomes das colunas e também os nomes das tabelas devem conter apenas caracteres alfanuméricos e sublinhados. Se quaisquer caracteres maiúsculos forem usados em nomes de colunas ou tabelas, eles serão convertidos em minúsculas. Em relação aos tipos de dados, você aprenderá mais sobre eles ao longo desta seção.

Para esta seção, não se preocupe com as especificidades dos tipos de dados. Por padrão Hive e Impala, crie o que são chamados de tabelas gerenciadas ou gerenciadas internamente. Quando você solta uma tabela gerenciada, o diretório de armazenamento de tabelas no sistema de arquivos é excluído e todos os arquivos de dados dentro desse diretório são excluídos.

Em alguns casos, convém evitar esse comportamento. Você pode fazer isso criando uma tabela não gerenciada ou gerenciada externamente. Para fazer isso, use a palavra-chave **EXTERNAL** como mostrado no

exemplo abaixo. Quando você solta uma tabela gerenciada externamente, os metadados da tabela são removidos do metastore, mas os dados da tabela permanecem no HDFS.

```
CREATE EXTERNAL TABLE tablename (
    col1 TYPE,
    col2 TYPE,
    …);
```

Uma tabela gerenciada externamente no Hive e Impala é diferente do que às vezes é chamado de tabela externa em sistemas de banco de dados relacionais. Então, se você tem alguma noção do que uma tabela externa é do mundo do banco de dados relacional, coloque isso de lado e lembre-se que com Hive e Impala, gerenciados externamente, significa que Hive e Impala deixarão os arquivos de dados no lugar se você apagar a tabela.

O Hive tem uma opção para criar o que é chamado de tabela temporária. Esta é uma tabela que é visível apenas para você, e somente na sessão atual. A tabela e todos os dados armazenados nela são excluídos no final da sua sessão atual. Você pode criar uma tabela temporária com o Hive adicionando a palavra-chave **TEMPORARY**, como mostrado no exemplo abaixo. Isso pode ser útil se você estiver usando um ambiente real e quiser testar uma instrução criar tabela antes de executá-la de verdade. No entanto, tabelas temporárias não são suportadas pelo Impala e eles têm algumas outras limitações. Portanto, neste curso, não estaremos usando tabelas temporárias.

```
CREATE TEMPORARY TABLE tablename (
    col1 TYPE,
    col2 TYPE,
    …);
```

Então essa é a sintaxe básica da instrução criar tabela para Hive e Impala. Novamente, se você já criou tabelas em um banco de dados relacional, ele deve parecer familiar para você. No entanto, existem três cláusulas opcionais que você pode usar em uma instrução criar tabela. Eles são exclusivos para motores SQL distribuídos como Hive e Impala. Estes são: a cláusula **ROW FORMAT**, a cláusula **STORED AS** e a cláusula de **LOCATION**.

A Cláusula ROW FORMAT

Como você provavelmente sabe, os arquivos de dados podem vir em diferentes formatos. Por exemplo, um arquivo pode ser delimitado por vírgula, o que significa que a vírgula (,) é usada para marcar (delimitar) quando o valor de uma coluna termina e o valor da próxima coluna começa. O caractere de tabulação é frequentemente usado, fornecendo um arquivo delimitado por tabulação.

Como parte da instrução **`CREATE TABLE`**, você pode especificar como os dados são delimitados em seus arquivos. Isso é feito usando a cláusula **`ROW FORMAT`**. A sintaxe para esta cláusula é:

```
ROW FORMAT DELIMITED
      FIELDS TERMINATED BY character
```

Por exemplo, considere esta linha de um arquivo de dados:

```
1,Data Analyst,135000,2016-12-21 15:52:03
```

Existem quatro campos aqui, separados por vírgulas: um id, um cargo, um salário e um postado em data e hora (**id, title, salary, posted**) quando os dados foram registrados. A instrução a seguir criará a tabela:

```
CREATE TABLE jobs (
      id INT,
      title STRING,
      salary INT,
      posted TIMESTAMP)
    ROW FORMAT DELIMITED
    FIELDS TERMINATED BY ‘,’;
```

A parte **ROW FORMAT DELIMITED** desta cláusula especifica que você está usando um delimitador. Você também precisa da parte **FIELDS TERMINATED BY**, para especificar qual delimitador você está usando. Nesse caso, o delimitador é a vírgula (,). Se o delimitador fosse o caractere de tabulação, a cláusula usaria **\t** entre aspas (porque **\t** é a sequência de escape que representa o caractere de tabulação):

```
ROW FORMAT DELIMITED
    FIELDS TERMINATED BY ‘\t’;
```

A parte **FIELDS TERMINATED BY** é uma parte da cláusula **ROW FORMAT**. Deve ser precedido por **ROW FORMAT DELIMITED**. A quebra de linha e o recuo usados nesses exemplos são opcionais.

Se você omitir a cláusula **ROW FORMAT,** Hive e Impala usarão o delimitador de campo padrão, que é o caractere ASCII Control+A. Este é um caractere não imprimível, portanto, quando você tenta visualizar esse caractere usando um editor de texto ou um comando **cat**, ele pode ser renderizado como um símbolo (como um retângulo com 0s e 1s) e outros caracteres podem se sobrepor a ele .

Experimente!

Nos exercícios a seguir, você pode ver como a cláusula **ROW FORMAT** determina o armazenamento de novos dados para uma tabela e como ela informa ao Hive e ao Impala como ler corretamente uma tabela em arquivos de dados existentes.

Na VM:

1. Faça login no Hue e vá para o editor de consultas Impala.
2. Faça o seguinte para criar uma tabela delimitada por vírgulas, preencha-a com uma linha de dados e observe o arquivo resultante no HDFS:
   a) Execute a instrução **CREATE TABLE**:

```
CREATE TABLE jobs (
    (id INT, title STRING, salary INT, posted TIMESTAMP)
  ROW FORMAT DELIMITED
      FIELDS TERMINATED BY ‘,’;
```

   b) Carregue uma linha de dados executando a instrução a seguir. (Observação: esta instrução não é uma boa maneira de adicionar muitos dados a uma tabela em um sistema de big data, por motivos descritos posteriormente no curso. Aqui, estamos adicionando apenas uma linha para fins de demonstração.)

```
INSERT INTO jobs
     VALUES (1,'Data Analyst',135000,'2016-12-21 15:52:03');
```

   c) Use o Navegador de arquivos ou o painel de fonte de dados no lado esquerdo (escolhendo o ícone de arquivos em vez do ícone do banco de dados) e localize o diretório **/user/hive/warehouse/jobs**. Se você não vir o subdiretório **jobs**, atualize a exibição clicando no botão atualizar (duas setas curvas). Encontre um arquivo com um nome que seja apenas uma sequência de letras e números e clique nesse arquivo.

d) Você pode ver o conteúdo do arquivo no painel principal. Observe que você tem uma linha de dados delimitada por vírgulas.

e) Observe que os valores de string e timestamp armazenados neste arquivo não estão entre aspas. As aspas foram usadas na instrução **INSERT** para incluir a **string** literal e os valores de timestamp de data/hora, mas Hive e Impala não armazenam essas aspas nos arquivos de dados da tabela.

3. Agora volte ao editor de consultas Impala e crie uma tabela delimitada por tabulações, preencha-a com os mesmos dados e compare o arquivo resultante no HDFS com o que você tinha para o arquivo delimitado por vírgulas.

4. Execute a instrução CREATE TABLE (as únicas diferenças são o nome da tabela e o caractere delimitador):

```
CREATE TABLE jobs_tsv
   (id INT, title STRING, salary INT, posted TIMESTAMP)
   ROW FORMAT DELIMITED
      FIELDS TERMINATED BY '\t';
```

5. Carregue uma linha de dados executando a instrução a seguir. (Observação: esta instrução não é uma boa maneira de adicionar muitos dados a uma tabela em um sistema de big data, por motivos descritos posteriormente no curso. Aqui, estamos adicionando apenas uma linha para fins de demonstração.)

```
INSERT INTO jobs_tsv
    VALUES (1,'Data Analyst',135000,'2016-12-21 15:52:03');
```

6. Use o Navegador de arquivos ou o painel de fonte de dados no lado esquerdo (escolhendo o ícone de arquivos em vez do ícone do banco de dados) e localize o diretório **/user/hive/warehouse/jobs_tsv**. Se você não vir o subdiretório **jobs_tsv**, atualize a exibição clicando no botão atualizar (duas setas curvas). Encontre um arquivo com um nome que seja apenas uma sequência de letras e números e clique nesse arquivo.

7. Você pode ver o conteúdo do arquivo no painel principal. Observe que, desta vez, você tem uma linha de dados delimitada por tabulação.

8. Observe que os valores de string e timestamp armazenados neste arquivo não estão entre aspas. As aspas foram usadas na instrução **INSERT** para incluir a **string** literal e os valores de timestamp de data/hora, mas Hive e Impala não armazenam essas aspas nos arquivos de dados da tabela.

9. Elimine as tabelas jobs e jobs_tsv. (Reveja as leituras de "Criando Bancos de Dados e Tabelas…" para saber como descartar tabelas, se necessário.)

10. Ao criar uma tabela para arquivos existentes, você desejará especificar como o arquivo já está delimitado. Execute as etapas a seguir para ver isso funcionando.

    a) Primeiro, examine os dados no diretório **/user/hive/warehouse/investors**. Este será o local padrão para uma tabela chamada **default.investors**. Ainda não há tabela para esses dados, então você criará uma. Observe que o arquivo é delimitado por vírgulas.

    b) Crie uma tabela de **investors** gerenciados externamente usando a instrução a seguir (que propositalmente não especifica o delimitador). ***É importante usar a palavra-chave*** **EXTERNAL**, para que você possa descartar a tabela sem excluir os dados.

```
CREATE EXTERNAL TABLE default.investors
    (name STRING, amount INT, share DECIMAL(4,3));
```

    c) Use Hue para olhar para a mesa. Ele tem apenas algumas linhas, então você pode usar **`SELECT * FROM investors;`** no editor de consultas ou você pode usar o painel de origem de dados para exibir os dados de amostra. Observe que a linha inteira acabou na coluna de **`name`**! Isso ocorre porque o comando usou o delimitador padrão, **`Control+A`**, em vez da vírgula.

d) Elimine a tabela digitando e executando o comando **DROP TABLE investors;** Verifique se o diretório **/user/hive/warehouse/investors** ainda existe e tem um arquivo nele. (Se você cometeu um erro e o diretório sumiu, veja abaixo!)

e) Agora, crie outra tabela de investidores usando a cláusula **ROW FORMAT** para especificar o delimitador:

```
CREATE EXTERNAL TABLE default.investors
   (name STRING, amount INT, share DECIMAL(4,3))
   ROW FORMAT DELIMITED
       FIELDS TERMINATED BY ',';
```

f) Use Hue para olhar para a mesa. Agora cada coluna deve ter valores. Guarde esta tabela, você a usará novamente mais tarde.

Se você excluiu acidentalmente seus dados:

Abra uma janela do Terminal. (Você pode fazer isso clicando no ícone na barra de menus que se parece com um computador.) Digite e execute o seguinte comando (em uma linha), que copiará o arquivo do disco local para o local apropriado no HDFS. Não inclua o $; esse é o prompt para indicar que este é um comando shell de linha de comando.

```
$ hdfs dfs -put
~/training_materials/analyst/scripts/static_data/default/investors /user/hive/war
ehouse/
```

A Cláusula STORED AS

Os dados de cada tabela são armazenados em arquivos. A cláusula **ROW FORMAT** especifica os delimitadores entre os valores das colunas nesses arquivos, mas existem diferentes formatos de arquivo que você pode usar. (Você aprenderá mais sobre esses formatos de arquivo na Semana 3 deste curso.) A cláusula **STORED AS** na instrução **CREATE TABLE** permite especificar qual formato de arquivo você deseja que uma nova tabela use. Para criar uma tabela que usa arquivos de dados existentes, você precisa corresponder ao formato do arquivo.

A sintaxe para esta cláusula é simplesmente:

```
STORED AS filetype
```

Por exemplo, uma instrução de criação de tabela pode ter esta aparência:

```
CREATE TABLE jobs
   (id INT, title STRING, salary INT, posted TIMESTAMP)
   STORED AS TEXTFILE;
```

O formato de arquivo padrão é **TEXTFILE** — formato de texto simples, que pode ser lido por humanos — então, neste exemplo, a cláusula **STORED AS** é opcional. Sem ele, Hive ou Impala ainda usariam **TEXTFILE** para o formato de arquivo. No entanto, outros formatos de arquivo (que geralmente parecerão sem sentido se você tentar visualizar os arquivos diretamente) devem ser especificados explicitamente.

Experimente!

Na VM:

1. Faça login no Hue e vá para o editor de consultas Impala.
2. Faça o seguinte para criar uma tabela, preencha-a com uma linha de dados e observe o arquivo resultante no HDFS:

a) Execute a seguinte instrução CREATE TABLE:

```
CREATE TABLE jobs_txt
      (id INT, title STRING, salary INT, posted TIMESTAMP)
    STORED AS TEXTFILE;
```

b) Carregue uma linha de dados executando a instrução a seguir.

```
INSERT INTO jobs_txt
   VALUES (1,'Data Analyst',135000,'2016-12-21 15:52:03');
```

c) Use o Browser File [Navegador de arquivos] ou o data source panel [painel de fonte de dados] (escolhendo o ícone de arquivos em vez do ícone do banco de dados) e localize o diretório **/user/hive/warehouse/jobs_txt**. Se você não vir o subdiretório **jobs_txt**, atualize a exibição clicando no botão atualizar (duas setas curvas). Encontre um arquivo com um nome que seja apenas uma sequência de letras e números e clique nesse arquivo.

d) Você pode ver o conteúdo do arquivo no painel principal. Observe que você pode ver claramente cada um dos valores adicionados à tabela.

3. Agora crie outra tabela usando um formato diferente e veja que o arquivo resultante parece diferente:

a) Execute a seguinte instrução **CREATE TABLE**, que configura a tabela para armazenar dados no formato **PARQUET**:

```
CREATE TABLE jobs_parquet
       (id INT, title STRING, salary INT, posted TIMESTAMP)
   STORED AS PARQUET;
```

b) Carregue uma linha de dados executando a instrução a seguir.

```
INSERT INTO jobs_parquet
   VALUES (1,'Data Analyst',135000,'2016-12-21 15:52:03');
```

c) Use o Browser File [Navegador de Arquivos] ou o Data source panel [painel de origem de dados] (escolhendo o ícone de arquivos em vez do ícone do banco de dados) e localize o diretório **/user/hive/warehouse/jobs_parquet**. Se você não vir o subdiretório **jobs_parquet**, atualize a exibição clicando no botão atualizar (duas setas curvas). Encontre um arquivo com um nome que seja apenas uma sequência de letras e números e clique nesse arquivo. Você receberá uma mensagem de erro informando que o Hue não pode ler o arquivo.

d) Abra uma janela do Terminal. (Você pode fazer isso clicando no ícone na barra de menus que se parece com um computador.) Digite e execute o seguinte comando, que mostrará o conteúdo do arquivo Parquet. (Não inclua o $; esse é o prompt para indicar que este é um comando shell de linha de comando, não uma consulta.) Observe que a saída inclui muitos caracteres não ASCII, portanto, você não pode realmente ler a maior parte.

```
$ hdfs dfs -cat /user/hive/warehouse/jobs_parquet/*
```

4. Elimine ambas as tabelas (**jobs_txt** e **jobs_parquet**), pois você não precisará de nenhuma delas novamente.

5. Agora tente criar uma tabela usando dados de um arquivo Parquet existente. Quando terminar, guarde esta tabela, porque você a usará novamente mais tarde. (Se você usar a palavra-chave **EXTERNAL** conforme indicado abaixo, então, descartar a tabela não excluirá os dados, portanto, você pode eliminá-la agora e voltar e recriar a tabela mais tarde, se desejar.)

a) Uma versão Parquet dos dados dos **investors** também é armazenada no HDFS, em **/user/hive/warehouse/investors_parquet** (que será o local padrão para uma tabela chamada **default.investors_parquet**). Examine o arquivo da mesma forma que você examinou o arquivo Parquet de jobs: Na janela Terminal, emita o comando

```
hdfs dfs -cat /user/hive/warehouse/investors_parquet/investors.parq
```

Novamente, você verá que não está realmente em formato legível por humanos.

b) Agora crie a tabela a partir do editor de consultas:

```
CREATE EXTERNAL TABLE default.investors_parquet
      (name STRING, amount INT, share DECIMAL(4,3))
  STORED AS PARQUET;
```

c) Use o painel de origem de dados ou execute uma consulta **`SELECT *`** para verificar se o conteúdo da nova tabela está correto. (Deve ser idêntica à tabela de outros **`investors`** que você criou na leitura "A cláusula ROW FORMAT".)

A Cláusula LOCATION

A menos que especificado de outra forma, Hive e Impala armazenam dados de tabela no diretório **warehouse**, que por padrão é o diretório HDFS **/user/hive/warehouse/**. No entanto, pode não ser onde você deseja armazenar alguns dados da tabela. Por exemplo, os dados podem já existir em outro lugar no HDFS e você pode não ter permissão para movê-los ou copiá-los. Se houver muitos dados, copiá-los para o diretório do warehouse seria ineficiente comparado a consultá-los em seu local atual. Mesmo que não haja muitos dados, mover uma cópia para o diretório do warehouse significa que a cópia ficará desatualizada assim que forem feitas alterações na versão original.

A cláusula **LOCATION** permite criar uma tabela cujos dados são armazenados em um diretório especificado, que pode estar fora do diretório do warehouse. A sintaxe é simplesmente:

```
LOCATION 'path/to/location/'
```

O exemplo mostrado abaixo cria uma nova tabela chamada jobs_training cujos dados residem no diretório HDFS /user/training/jobs/. O diretório de armazenamento especificado pode já existir, mas se não existir, Hive ou Impala o criará.

```
CREATE TABLE jobs_training
    (id INT, title STRING, salary INT, posted TIMESTAMP)
  LOCATION '/user/training/jobs_training/';
```

Para especificar um diretório de armazenamento fora do HDFS, você precisará de um caminho totalmente qualificado na cláusula **LOCATION**, incluindo um protocolo no início do caminho. Por exemplo, para especificar um diretório de armazenamento no Amazon S3, você normalmente precisará usar **LOCATION 's3a://bucket/folder/'**.

Não confunda isso com a palavra-chave EXTERNAL

Na prática, a palavra-chave **EXTERNAL** é frequentemente usada em conjunto com a cláusula **LOCATION** para especificar um diretório de armazenamento fora do diretório do warehouse. Mas quando você usa a palavra-chave **EXTERNAL**, isso não significa necessariamente que os dados são armazenados fora do diretório do warehouse. Na verdade, se você usar a palavra-chave **EXTERNAL**, mas não especificar um diretório com a cláusula **LOCATION**, o Hive e o Impala armazenarão os dados da tabela no diretório do warehouse. Por outro lado, se você usar a cláusula **LOCATION** sem a palavra-chave **EXTERNAL**, a eliminação da tabela poderá excluir os dados, mesmo que estejam armazenados fora do diretório do warehouse.

Portanto, pense em EXTERNO como significando gerenciado externamente (ou seja, gerenciado por algum software diferente do Hive ou Impala) e não significando armazenado externamente. Lembre-se de que a cláusula **LOCATION**, não a palavra-chave **EXTERNAL**, determina onde os dados da tabela são armazenados.

Experimente!

1. Se ainda não o fez, use o Hue File Browser ou o painel de fonte de dados para examinar o diretório **/user/hive/warehouse**. (Usando o painel de fonte de dados, você precisa clicar no ícone de arquivos em vez do ícone de banco de dados.) Você deve ver os diretórios para bancos de dados (com **.db** no final) e para quaisquer tabelas de teste que você criou no banco de dados **default** e não apagou. (Se você ainda não criou nenhuma tabela, ainda deverá ver alguns diretórios correspondentes a tabelas no banco de dados **default**, como **customers** e **emplooyees**.) Olhe dentro de um desses diretórios de tabela e observe se há um arquivo dentro. Se a tabela tiver dados, ela terá pelo menos um arquivo aqui.

   Em seguida, faça o seguinte para criar e examinar uma tabela cujos dados não vão para esse diretório padrão.

2. Vá para o editor de consultas Impala, selecione **default** como o banco de dados ativo e execute a seguinte instrução:

```
CREATE TABLE jobs_training
   (id INT, title STRING, salary INT, posted TIMESTAMP)
  LOCATION '/user/training/jobs_training/';
```

3. Use o painel da fonte de dados (para que você possa deixar o editor de consultas no painel principal) para localizar o diretório da tabela para esta nova tabela. Observe que não está em **/user/hive/warehouse/** como as outras tabelas no banco de dados padrão estão. Em vez disso, deve estar em **/user/training/**. Verifique se o novo diretório **jobs_training** está vazio, pois a tabela foi criada sem dados e nenhum foi inserido.

4. Insira alguns dados na nova tabela usando a seguinte instrução no editor de consultas:

```
INSERT INTO jobs_training
   VALUES (1,'Data Anaylist',135000,'2016-12-21 15:52:03');
```

5. Olhe novamente dentro do diretório **jobs_training** e verifique se um novo arquivo foi adicionado. Talvez seja necessário clicar no botão atualizar para atualizar a exibição. Você pode dar uma olhada no arquivo para ver que são os dados que você acabou de inserir.

6. Apague a tabela.

   Agora crie uma tabela para consultar alguns dados existentes localizados em um bucket do S3. Criamos um bucket e concedemos acesso somente leitura à VM. A mesma linha de dados que você está usando nesses testes é salva em um arquivo de texto delimitado no bucket do S3 chamado **training-coursera**, em uma pasta chamada **jobs**. O arquivo usa o delimitador padrão (Control+A) para que você não precise de uma cláusula **ROW FORMAT**.

7. Execute a seguinte instrução:

```
CREATE EXTERNAL TABLE jobs_s3
    (id INT, title STRING, salary INT, posted TIMESTAMP)
  LOCATION 's3a://training-coursera1/jobs/';
```

8. Verifique se a tabela tem uma linha de dados, usando o painel de origem de dados ou executando uma consulta **SELECT ***.

9. Apague a tabela. Observação: Normalmente, você precisa usar a palavra-chave **EXTERNAL** para garantir que não exclua dados acidentalmente ao descartar a tabela. Nesse caso, como você não tem acesso de gravação ao bucket, você não excluirá os dados mesmo que tenha esquecido a palavra-chave **EXTERNAL**. Ainda assim, é uma boa prática usar **EXTERNAL** para dados nos quais outras pessoas possam estar confiando também, para que você não corra o risco de destruir o trabalho deles, assim como o seu!

Criar atalhos de tabela

A seguir estão duas dicas para criar bancos de dados e tabelas.

Usando IF NOT EXISTS

Se você tentar criar um banco de dados ou tabela usando um nome para um que já existe, Hive ou Impala lançará um erro. Para evitar isso, você pode adicionar as palavras-chave **IF NOT EXISTS** à sua instrução **CREATE DATABASE** ou **CREATE TABLE**. A sintaxe é a seguinte:

```
CREATE DATABASE IF NOT EXISTS database_name;

CREATE TABLE IF NOT EXISTS table_name;
```

Quando você usa **IF NOT EXISTS**, Hive ou Impala não gerarão um erro se esse nome já estiver em uso. Em vez disso, eles não farão nada.

Isso é particularmente útil se você estiver usando um script para criar um banco de dados ou uma tabela. Se o script for executado várias vezes, o uso de **IF NOT EXISTS** permitirá que o script crie o banco de dados ou a tabela na primeira execução e também será concluído sem erros nas execuções subsequentes.

Você também pode usar **IF NOT EXISTS** junto com a palavra-chave **EXTERNAL** e junto com as outras cláusulas **CREATE TABLE** opcionais descritas nesta semana do curso.

Clonando uma tabela com LIKE

Se você precisar de uma nova tabela definida com exatamente a mesma estrutura de uma tabela existente, o Hive e o Impala facilitam muito a criação da nova tabela. Isso é chamado de clonagem de uma tabela e é feito usando a cláusula **LIKE**. A nova tabela terá as mesmas definições de coluna e outras propriedades da tabela existente, mas sem dados. A sintaxe é:

```
CREATE TABLE new_table_name LIKE existing_table_name;
```

O exemplo mostrado abaixo cria uma nova tabela vazia chamada **jobs_archived** com a mesma estrutura e propriedades da tabela existente chamada **jobs**.

```
CREATE TABLE jobs_archived LIKE jobs;
```

É possível especificar algumas das propriedades da tabela para a nova tabela incluindo as cláusulas apropriadas na instrução **CREATE TABLE … LIKE**. Das cláusulas abordadas neste curso, atualmente apenas as cláusulas **LOCATION** e **STORED AS** podem ser usadas. Se você precisar alterar outras propriedades, use **ALTER TABLE** após criar a tabela para definir essas propriedades.

Experimente!

Tente como as coisas funcionam se você criar um banco de dados usando **IF NOT EXISTS**. Você criará um banco de dados chamado **dig**; não abandone este, você o usará mais adiante.

1. Primeiro, crie um banco de dados chamado **dig** executando o comando:

```
CREATE DATABASE IF NOT EXISTS dig;
```

Verifique se agora você tem um banco de dados chamado **dig**.

2. Agora tente criar o banco de dados novamente usando **CREATE DATABASE dig;** (sem a frase **IF NOT EXISTS**). O que acontece?

3. Agora tente o comando original novamente e observe como a resposta é diferente:

```
CREATE DATABASE IF NOT EXISTS dig;
```

Agora tente clonar.

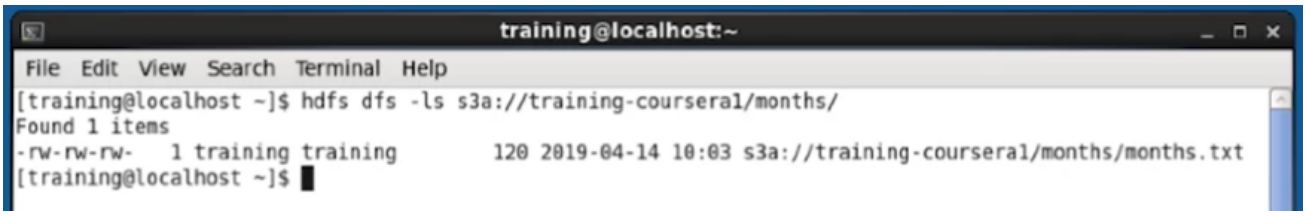
4. Clone uma das tabelas no banco de dados padrão usando a cláusula **LIKE**—qualquer uma das tabelas servirá, apenas certifique-se de usar um nome ligeiramente diferente para a nova tabela. (Por exemplo, você pode clonar a tabela de customers no banco de dados default e nomeá-la como **customer_clone**.)
5. Verifique se a estrutura (nomes das colunas com seus tipos de dados) é a mesma da tabela que você clonou. Verifique também se a nova tabela não contém dados e se a tabela original ainda possui seus dados.
6. Elimine a tabela clonada.

Usando diferentes Schemas no mesmo dado

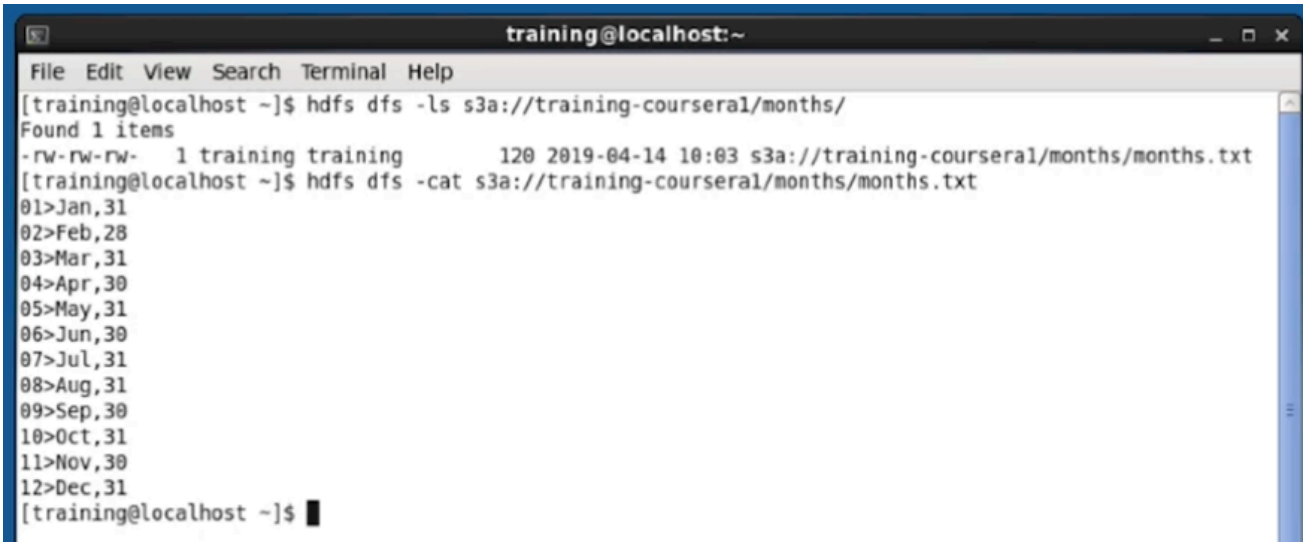
Se este tópico de criação de tabelas Hive e Impala é novo para você, você pode não entender imediatamente algumas das implicações em torno dele. Uma implicação que é especialmente difícil de entender é como o acoplamento solto de definições de tabela e os dados subjacentes torna o Hive e o Impala radicalmente diferentes de forma tradicional sistemas de banco de dados relacional.

Uma boa maneira de destacar essa diferença radical é demonstrar que você pode criar duas ou mais tabelas. Essa consulta os mesmos arquivos de dados sublinhados. Para demonstrar, usaremos um conjunto de dados armazenado em um arquivo de texto no S3, ele está no bucket chamado **training-coursera1**, em um subdiretório chamado **months**.

Eu vou o comando, **hdfs dfs -ls s3a://training-coursera1/months/** para listar os arquivos nesse diretório. O resultado mostra que há apenas um arquivo chamado **months.txt**.

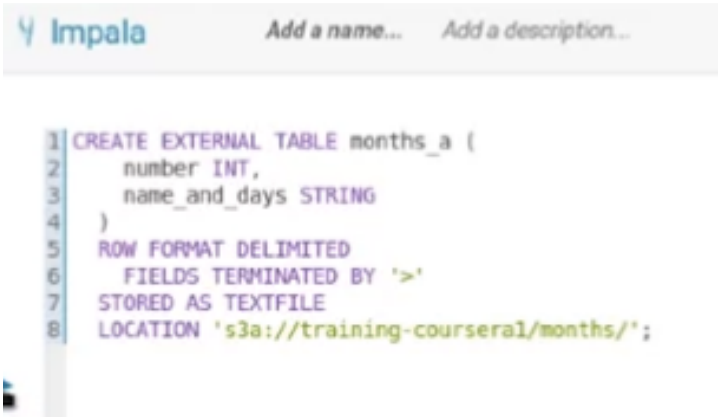


Executando um comando **hdfs dfs -cat** para imprimir o conteúdo desse arquivo, os dados resultantes descrevem os 12 meses do ano e parece que cada registro representa três valores. Primeiro, o número do mês, segundo o nome abreviado de três letras e terceiro, o número de dias nesse mês. Mas esse arquivo não usa delimitadores ou separadores de campo da maneira usual. Tem um sinal maior que, separando o primeiro e segundo campos. E uma vírgula, separando o segundo e terceiro campos, então eu gostaria

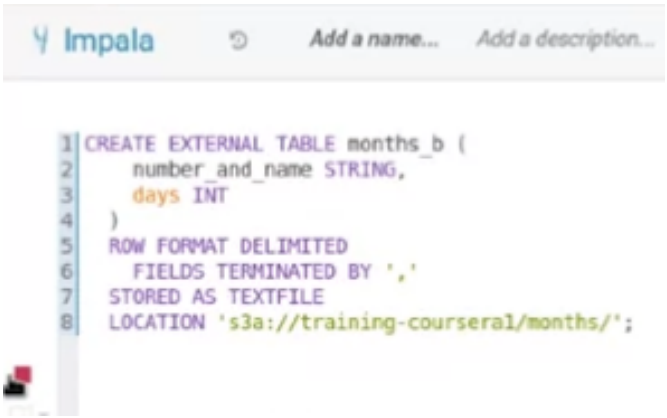


de criar uma tabela para consultar esses dados, mas não tenho certeza de qual delimitador especificar.

Então, para demonstrar que é possível criar mais de uma tabela em cima dos mesmos arquivos de dados, criarei duas tabelas, uma usando o sinal maior que como delimitador e a outra usando a vírgula. No editor de consulta Impala no Hue, vou executar uma instrução criar tabela para criar o primeiro deles. Os dados estão sendo gerenciados externamente, então usarei a palavra-chave externa, **CREATE EXTERNAL TABLE**. Vou nomeá-lo **month_a**. Para este primeiro, vou usar o sinal **>** como delimitador. Então as colunas serão o número do mês, que é um inteiro, isso é o que está no lado esquerdo do sinal maior que e no lado direito há uma concatenação de nome e dias, que podemos representar em uma string. Para especificar o delimitador, usarei **ROW FORMAT DELIMITED FIELDS TERMINATED BY ">"**. Os dados são armazenados em um arquivo de texto, então eu vou especificar, armazenado como arquivo de texto. Como o arquivo de texto é o formato de arquivo padrão, eu não preciso especificar isso, mas vou incluí-lo. E, finalmente, incluirei a cláusula de localização para especificar o diretório no S3, onde os dados são armazenados. Vou executar esta instrução para criar a tabela.



Agora vou modificar esta instrução para criar a segunda tabela. Vou fazer o nome da tabela **months_b**. Para esta tabela vou usar uma **vírgula** como delimitador. Assim, as colunas serão uma concatenação do número do mês e o nome do mês de três letras, que podemos representar em uma **string**. Isso é o que está à esquerda da vírgula, e à direita está o número de dias no mês, que é um inteiro. Na cláusula de **ROW FORMAT**, vou mudar o delimitador para uma **vírgula** e vou deixar tudo o resto como está. Vou executar esta instrução para criar a segunda tabela. Agora temos duas tabelas chamadas **months_a** e **months_b**, que consultam os mesmos arquivos de dados subjacentes, mas que têm esquemas diferentes.



Então, quando você consulta essas duas tabelas, você obtém colunas de resultados diferentes, mesmo que os dados por trás das duas tabelas sejam idênticos. A coluna com os valores concatenados juntos não é muito útil neste formulário. Mas você poderia usar o Hive ou Impala construídas em funções de string para extrair as partes dele.

Por exemplo, no Impala, você pode executar uma consulta que usa a função **split_part** para retornar as duas partes de cada nome e valor de dias, nos lados esquerdo e direito da vírgula, como duas colunas separadas nomeadas nome e dias. A função **split_part** não está disponível no Hive, mas há duas funções embutidas semelhantes no Hive chamado **split** e **substring_index**.

Você poderia usar um desses em vez disso. Vou deixar apagar essas duas tabelas, não precisaremos mais delas.

3.3. CREATE TABLE - Técnicas Avançadas

Especificando TBLPROPERTIES

Anteriormente, você aprendeu sobre a estrutura da instrução **CREATE TABLE** e sobre três cláusulas importantes que você pode usar para especificar como e onde os dados da tabela são armazenados. A cláusula **ROW FORMAT**, a cláusula **STORED AS** e a cláusula **LOCATION.** Essas três cláusulas são opcionais, assim como a palavra-chave **EXTERNAL**.

Mas você os usará frequentemente para substituir os comportamentos padrão do Hive e Impala ao criar uma nova tabela. Há outra cláusula opcional que você provavelmente não usará com tanta frequência, mas você ainda deve saber sobre ela. É a cláusula **TBLPROPERTIES**.

Esta cláusula permite que você defina algumas propriedades especiais para a tabela que você está criando. Por exemplo, se os dados da tabela estiverem em arquivos que incluem um cabeçalho de coluna na primeira linha, você poderá definir **TBLPROPERTIES ('skip.header.line.count'='1')**, como mostrado aqui. Então vamos pular a primeira linha. Esteja avisado, porém, que isso irá ignorar a primeira linha em cada arquivo no diretório de dados da tabela, não apenas no primeiro arquivo.

```
CREATE EXTERNAL TABLE dbname.tablename (col1 TYPE, col2 TYPE, …)
    ROW FORMAT …
    STORED AS …
    LOCATION …
    TBLPROPERTIES('skip.header.line.count'='1');
```

Se seus dados estão em vários arquivos, como muitas vezes é com big data, então você precisa de cada arquivo para ter a linha de cabeçalho nele. Caso contrário, Hive e Impala não lerão os dados na primeira linha desses outros arquivos.

Além disso, alguns sistemas, como o Apache Spark, ignorarão essa propriedade **skip.header.line.count** ao consultar uma tabela. Se você quiser saber mais sobre isso, você pode ler a entrada no sistema de rastreamento de problemas Apache Spark que descreve isso.

Usando Hive SerDes

O processamento de dados tradicional depende de dados armazenados em tabelas estruturadas com linhas e colunas bem definidas. No entanto, os dados geralmente não são estruturados dessa maneira. Muitos dados existem em arquivos de texto não estruturados ou semiestruturados, como arquivos de log, notas de formato livre em registros médicos eletrônicos, diferentes tipos de mensagens eletrônicas e análises de produtos. Esse tipo de dado pode fornecer informações valiosas, mas trabalhar com ele requer uma abordagem diferente.

O Hive fornece recursos para trabalhar com dados de texto não estruturados, dados semiestruturados em formatos como JSON e dados que não possuem delimitadores consistentes. Observe que os recursos discutidos nesta leitura são suportados apenas pelo Hive, não pelo Impala.

Lembre-se de que a cláusula **ROW FORMAT DELIMITED** é usada ao criar uma tabela com dados armazenados em arquivos de texto. Ao criar uma tabela usando esta cláusula, os dados nos arquivos de texto subjacentes devem ser organizados em linhas e colunas com delimitadores consistentes.

Hive SerDes

O Hive fornece interfaces chamadas SerDes que podem ler e gravar dados que não estão em um formato tabular estruturado. SerDe significa serializador/desserializador. A serialização é o processo de conversão de dados em bytes para que possam ser armazenados; desserializar é o processo inverso—decodificação ao ler o arquivo armazenado.

Você pode especificar o SerDe de uma tabela ao criar a tabela. Na verdade, cada tabela no Hive tem um SerDe associado a ela, quer você perceba ou não. Normalmente, o SerDe é configurado automaticamente para o padrão para um determinado formato de arquivo, com base nas cláusulas **ROW FORMAT** e **STORED AS**.

Para uma tabela armazenada como arquivos de texto, o SerDe padrão é chamado **LazySimpleSerDe**. O **LazySimpleSerDe** recebe esse nome porque usa uma técnica chamada inicialização lenta para melhor desempenho. Isso significa que ele não instancia objetos até que eles sejam necessários.

Além do **LazySimpleSerDe**, o Hive inclui vários outros SerDes integrados para trabalhar com dados em arquivos de texto. Se você quiser usar um desses, deverá especificar explicitamente o SerDe na instrução **CREATE TABLE**. A instrução inclui a cláusula **ROW FORMAT SERDE** seguida pelo nome completo da classe Java que implementa o SerDe, entre aspas.

Por exemplo, o Hive inclui o **OpenCSVSerde**, que pode processar dados em arquivos de valores separados por vírgula (CSV). Mas por que o Hive precisa de um CSV SerDe especial quando o SerDe padrão pode manipular arquivos de texto delimitados por vírgulas (quando você especifica **ROW FORMAT DELIMITED FIELDS TERMINATED BY ','**)? Embora o SerDe padrão do Hive possa funcionar com dados de texto simples delimitados por vírgulas, o formato CSV real permite coisas como vírgulas incorporadas em campos, campos entre aspas e campos ausentes. O SerDe padrão do Hive não suporta isso, mas o OpenCSVSerde sim. O OpenCSVSerde também suporta outros delimitadores, como os caracteres de tabulação e barra vertical.

O Experimente! A seção de leitura "**The ROW FORMAT Clause**" apresentou um exemplo em que você cria incorretamente uma tabela a partir de um arquivo delimitado por vírgula, porque o delimitador não foi especificado. Você corrigiu isso usando uma cláusula **ROW FORMAT DELIMITED**, mas também poderia ter corrigido usando o **OpenCSVSerde**:

```
CREATE EXTERNAL TABLE default.investors
   (name STRING, amount INT, share DECIMAL(4,3))
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde';
```

Dois outros exemplos são **RegexSerDe**, que identifica campos individuais em cada registro com base em uma expressão regular, e **JsonSerDe**, que processa dados no formato JSON. Veja a próxima leitura para saber mais sobre esses SerDes.

Além de usar o SerDes para ler e gravar dados de texto, o Hive também usa o SerDes para ler e gravar em formatos binários e colunares como Avro e Parquet, mas o Hive faz isso automaticamente e oculta os detalhes do usuário. Observe, no entanto, que o SerDe não é o mesmo que o tipo de arquivo, mas há uma conexão próxima - os tipos de arquivo exigem um SerDes específico para que o Hive possa ler e gravar esses tipos de arquivo. Você aprenderá mais sobre os tipos de arquivo nos materiais da próxima semana. Por enquanto, basta lembrar que o SerDes define processos para leitura de arquivos de dados.

Enquanto o **OpenCSVSerde** permite que você leia e grave arquivos usando a formatação especificada, alguns SerDes apenas lerão arquivos; você não pode usá-los para escrever. Você deve testar seu SerDes ou ler a documentação para determinar se a gravação de arquivos é suportada.

Experimente!

Faça o seguinte para criar uma tabela denominada túneis no banco de dados dig.

1. Examine os dados no arquivo **training_materials/analyst/data/tunnels.csv** no sistema de arquivos local da VM. Observe que é um arquivo de texto delimitado por vírgulas.
2. No editor de consulta do **Hive**, execute a instrução a seguir. Observe que ele usa a sintaxe **OpenCSVSerde** em vez da sintaxe **ROW FORMAT DELIMITED** que você usou na lição anterior. Você ainda pode usar essa outra sintaxe; isso serve para ilustrar que o **OpenCSVSerde** faz a mesma coisa. Além disso, você deve usar o Hive para executar a instrução abaixo, não o Impala.

```
CREATE TABLE dig.tunnels
      (terminus_1 STRING, terminus_2 STRING, distance SMALLINT)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde';
```

3. Em uma janela Terminal, execute a seguinte instrução (tudo em uma linha) para mover o **tunnels.csv** para o diretório da tabela. Não inclua o **$**; esse é o prompt para indicar que este é um comando shell de linha de comando, não uma consulta.

```
$ hdfs dfs -put ~/training_materials/analyst/data/tunnels.csv
/user/hive/warehouse/dig.db/tunnels/
```

Você aprenderá mais sobre essa declaração posteriormente.

4. Use o painel de origem de dados ou uma instrução Hive **`SELECT *`** para verificar se a tabela **tunnels** tem os dados, com valores nas colunas corretas. Lembre-se de que você pode consultar essa tabela apenas com o Hive, não com o Impala.

Trabalhando com dados não estruturados e semiestruturados

As seções abaixo descrevem dois SerDes para trabalhar com dados não estruturados e semiestruturados: **JsonSerDe** e **RegexSerDe**.

JsonSerDe

O **JsonSerDe** do Hive é útil para dados semiestruturados armazenados no formato JSON (JavaScript Object Notation). Para usar o SerDe, a instrução **CREATE TABLE** para a tabela deve incluir a cláusula **ROW FORMAT SERDE** seguida do nome totalmente qualificado da classe Java que implementa o **JsonSerDe**, entre aspas.

Por exemplo, alguns registros em um arquivo JSON podem ter esta aparência:

```
{"id":393930, "name":"Aleks Norkov", "city":"Berkeley", "state":"CA"}
{"name":"Christine Goldbaum", "city":"Boulder City", "state":"NV", "id":82800}
```

A instrução CREATE TABLE pode ficar assim:

```
CREATE TABLE subscribers
     (id INT, name STRING, city STRING, state STRING)
ROW FORMAT SERDE 'org.apache.hive.hcatalog.data.JsonSerDe';
```

A tabela incluiria os dados corretos, mesmo que os registros não estivessem organizados exatamente da mesma forma.

RegexSerDe

O **RegexSerDe** do Hive é útil para dados semiestruturados ou não estruturados. Ele identifica os campos dentro de cada registro com base em uma expressão regular (uma forma de descrever padrões em texto). Por exemplo, usando o **RegexSerDe**, o Hive pode ler diretamente um arquivo de log que não possui delimitadores consistentes. Assim como com os outros SerDes, a instrução **CREATE TABLE** para a tabela deve incluir a cláusula **ROW FORMAT SERDE** seguida do nome totalmente qualificado da classe Java que implementa o **RegexSerDe**, entre aspas. Para especificar a expressão regular, use **WITH SERDEPROPERTIES("input.regex"="regular expression")**. (Se você não estiver familiarizado com expressões regulares, veja este Tutorial de Expressões Regulares[1].)

Por exemplo, aqui estão dois registros de amostra de um arquivo de log. Os campos são separados por um espaço, mas também há espaços dentro da string de comentário citada. Isso dificulta o uso da cláusula **ROW FORMAT DELIMITED** usual, porque não seria capaz de distinguir os espaços entre aspas dos espaços delimitadores.

```
23/05/2016 19:45:19 312-555-7834 CHAMADA_RECEBIDA ""
23/05/2016 19:48:37 312-555-7834 RECLAMAÇÃO "Item danificado"
```

Você poderia fazer algum processamento dos dados para convertê-los, mas o **RegexSerDe** permite que você trabalhe diretamente com os dados brutos por meio do Hive. (Isso pode ser útil para arquivos de log que estão sendo gerados constantemente, adicionando os dados que você deseja usar.) Para fazer isso, use uma instrução **CREATE TABLE** como a seguinte.

```
CREATE TABLE calls (
      event_date STRING, event_time STRING,
      phone_num STRING, event_type STRING, details STRING)
```

[1] Tutorial de Expressões Regulares: https://ryanstutorials.net/regular-expressions-tutorial/

```
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
WITH SERDEPROPERTIES ("input.regex" =
              "([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*) \"([^\"]*)\"");
```

Dentro da expressão regular, os parênteses definem os grupos de captura. O valor de cada campo é o texto correspondente ao padrão dentro de cada par de parênteses. Os primeiros quatro campos capturam qualquer número de caracteres que não sejam espaços, com os campos separados por um espaço. O último campo começa após o caractere de aspas literal e captura qualquer número de caracteres sem aspas e termina antes do caractere de aspas seguinte. Os caracteres de aspas devem ser escapados com barras invertidas porque eles próprios estão dentro de uma string entre aspas.

Os cinco campos capturados tornam-se as cinco colunas da tabela: **event_date, event_time, phone_num, event_type e details**. Embora este exemplo use o tipo de dados **STRING** para todas as cinco colunas, o **RegexSerDe** oferece suporte a outros tipos de dados.

Observe que **RegexSerDe** é para desserialização (leitura), mas não suporta serialização (escrita). Você pode usar **LOAD DATA** para carregar um arquivo existente, mas **INSERT** não funcionará.

Experimente!

Na VM, tente criar algumas tabelas usando esses SerDes.

Primeiro, faça o seguinte para criar uma tabela usando um arquivo JSON para seus dados.

1. Examine o arquivo em **/user/hive/warehouse/subscribers/**. Estes são os mesmos dados de exemplo usados acima.
2. Crie a tabela de **subscribers**. No Hive, execute a seguinte instrução:

```
CREATE EXTERNAL TABLE default.subscribers
    (id INT, name STRING, city STRING, state STRING)
  ROW FORMAT SERDE 'org.apache.hive.hcatalog.data.JsonSerDe';
```

3. Execute uma instrução **SELECT *** para verificar se a tabela foi criada com os valores corretos em cada coluna. Observe que, embora as duas linhas forneçam os valores das colunas em ordens diferentes, cada linha tem os valores corretos (os dois IDs são **39390** e **82800**, por exemplo).
4. Opcional: Você pode descartar a tabela se quiser. Se você usou **EXTERNAL** conforme observado acima, a eliminação da tabela não excluirá os dados, portanto, você pode voltar e recriá-la mais tarde, se desejar.

Agora use o **RegExSerDe**. Os dados (veja o exemplo abaixo) possuem campos que não são delimitados; em vez disso, eles usam larguras fixas.

10309296107596201608290122 15Oakland CA94618

Field	Length
cust_id	**7**
order_id	**7**
order_dt	**8**
order_tm	**6**
city	**20**
state	**2**
zip	**5**

5. Opcional: Os dados de amostra estão em **/user/hive/warehouse/fixed**. (É apenas a linha fornecida acima.) Você pode dar uma olhada se quiser.

6. Dê uma olhada na expressão regular nesta instrução **CREATE TABLE** e veja como ela dividirá a linha de dados em sete campos. Em uma expressão regular, **\d** corresponde a qualquer dígito, um ponto (.) corresponde a qualquer caractere e **\w** corresponde a qualquer caractere de palavra (letras, números ou o caractere sublinhado). O primeiro \ em **\\d** e **\\w** é para escapar do segundo caractere \. Isso permite que o Hive saiba que este segundo \ é parte da expressão regular e não o início de uma sequência de escape representando um caractere especial.

```
CREATE EXTERNAL TABLE fixed
    (cust_id INT, order_id INT, order_dt STRING, order_tm STRING,
       city STRING, state STRING, zip STRING)
  ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
  WITH SERDEPROPERTIES ("input.regex" =
      "(\\d{7})(\\d{7})(\\d{8})(\\d{6})(.{20})(\\w{2})(\\ d{5})");
```

7. No Hive, execute a instrução acima.
8. Verifique os resultados usando o painel de origem de dados ou uma consulta **SELECT *.**
9. Opcional: Você pode descartar a tabela se quiser. Se você usou **EXTERNAL** conforme observado acima, a remoção da tabela não a excluirá, portanto, você pode voltar e recriá-la mais tarde, se desejar.

3.4. Gerenciando Tabelas Existentes

Você aprendeu a criar bancos de dados e tabelas. Mas e se você cometer um erro ao fazer isso? Por exemplo, e se você perceber, depois de criar uma tabela, que você a criou no banco de dados errado ou que uma das colunas precisa ser renomeada ou precisa ter um tipo de dados diferente? Ou se em uma data posterior, você precisar adicionar novas colunas a uma tabela existente, ou talvez descartar completamente uma tabela existente?

Estes são apenas alguns exemplos de casos em que você precisará modificar ou remover um banco de dados ou tabela. Nesta aula, você vai aprender como, usando as instruções ALTER e DROP.

Nesta seção, apresentaremos essas declarações de forma mais sistemática e descreveremos todas as implicações de usá-las. Por exemplo, o que acontece com os arquivos de dados de uma tabela quando você solta essa tabela?

Que alterações você deve fazer nos arquivos de dados se adicionar uma nova coluna a uma tabela? Muitas vezes, antes de modificar uma tabela, você deseja examiná-la para ver seu esquema existente e outras propriedades. Também vamos rever como fazer isso usando a declaração de descrição. Vamos introduzir uma variação na declaração descrita, e vamos introduzir outra instrução que é útil para ver como uma tabela foi criada.

Examinando a estrutura da tabela

Em contextos do mundo real, é comum que depois de criar uma tabela com Hive ou Impala, você perceba que os dados seriam melhor representados usando um esquema diferente ou que alguma outra propriedade da tabela precisa ser modificada. Portanto, muitas vezes você precisará fazer alterações nas tabelas. Você verá como fazer isso na leitura "Modificando Tabelas Existentes" mais adiante nesta lição.

Mas antes de modificar uma tabela, você deve examiná-la para ver seu esquema existente e outras propriedades. Existem dois comandos úteis para ajudá-lo a entender o estado de suas tabelas: **`DESCRIBE`** e **`SHOW CREATE TABLE`**. Além disso, **`SHOW CREATE TABLE`** é útil quando você precisa recriar tabelas em um ambiente diferente.

`DESCRIBE` (e **`DESCRIBE FORMATTED`**)

Você pode recuperar a instrução do utilitário **DESCRIBE**. Você pode usar a instrução **DESCRIBE** para ver quais colunas estão em uma tabela, executando o comando **DESCRIBE *tablename***. Os resultados mostram os nomes e tipos de dados de todas as colunas e, às vezes, um comentário para cada coluna.

Para ver informações mais detalhadas sobre uma tabela, use o comando **DESCRIBE FORMATTED**. Esse comando mostra detalhes adicionais, incluindo o formato do arquivo e o local de armazenamento dos arquivos de dados da tabela.

```
SHOW CREATE TABLE
```

Outra maneira de entender a estrutura e as propriedades de uma tabela é ver a instrução **CREATE TABLE** que a criou. Com Hive e Impala, você pode fazer isso usando a instrução **SHOW CREATE TABLE**.

Além disso, se você fez alterações no esquema ou em outras propriedades de uma tabela após criá-la, a saída da instrução **SHOW CREATE TABLE** refletirá todas essas alterações. Isso o torna particularmente útil para recriar uma tabela; em vez de emitir a instrução **CREATE TABLE** original, seguida por uma série de outras instruções para modificar a tabela, você pode usar SHOW **CREATE TABLE** para exibir uma única instrução **CREATE TABLE** para recriar a tabela em seu estado atual. Você pode copiar essa instrução **CREATE TABLE** e executá-la em um ambiente diferente que não compartilhe o mesmo **metastore**. Isso é especialmente útil ao migrar tabelas de um ambiente de desenvolvimento ou teste para um ambiente de produção.

Experimente!

Primeiro compare os resultados de uma instrução **DESCRIBE** com e sem a palavra-chave **FORMATTED**.

1. Execute os comandos a seguir no Hive e observe a diferença nos detalhes fornecidos. (Você deve usar o Hive porque a tabela **dig.tunnels** foi criada usando um Hive SerDe.)

```
DESCRIBE dig.tunnels;
DESCRIBE FORMATTED dig.tunnels;
```

Execute as etapas 2 e 3 para ver como você pode saber se uma tabela é gerenciada (internamente) ou não gerenciada (ou seja, gerenciada externamente).

2. Veja novamente os resultados **DESCRIBE FORMATTED** para **dig.tunnels**. Veja os resultados para Tipo de Tabela; o valor deve ser **MANAGED_TABLE**. Isso significa que ele foi criado sem a palavra-chave **EXTERNAL**.

3. Compare isso com uma tabela gerenciada externamente. Você pode executar isso no Hive ou Impala:

```
DESCRIBE FORMATTED default.investors;
```

Novamente, procure por **Table Type** e anote o valor para ele.

4. A tabela **dig.tunnels** foi criada com um SerDe. Observe novamente os resultados do comando **DESCRIBE FORMATTED** para essa tabela ou execute novamente o comando no Hive, se necessário. Procure a Biblioteca **SerDe** na coluna **col_name** e veja qual é o valor **data_type** para isso.

5. Embora você não tenha feito nenhuma modificação em uma tabela, tente a instrução **SHOW CREATE TABLE** com a tabela **default.investors**:

```
SHOW CREATE TABLE default.investors;
```

Tire algum tempo para rever o resultado. Você verá muitas coisas familiares (como palavras-chave **EXTERNAL** e **ROWS DELIMITED**); mas você provavelmente verá algumas coisas que não são tão familiares. Para esta tabela, você verá as configurações de **TBLPROPERTIES** que você não lembra da configuração e talvez não entenda. Essas são configurações que acontecem de forma invisível, por padrão. Não se preocupe com isso -

esta não é a instrução que você deveria ter usado para criar a tabela; em vez disso, é uma declaração possível que você pode usar para produzir a tabela exata que você tem atualmente.

Descartando bancos de dados e tabelas

Como você viu nas leituras "Criando Bancos de Dados e Tabelas…", você pode remover (descartar) bancos de dados e tabelas usando as instruções **`DROP DATABASE`** ou **`DROP TABLE`**. Assim como nas instruções **`CREATE`**, você pode descartar condicionalmente um banco de dados usando **`IF EXISTS`** na instrução. Isso evitará um erro caso o banco de dados ou a tabela não exista. A sintaxe é:

```
DROP DATABASE IF EXISTS database_name;
DROP TABLE IF EXISTS table_name;
```

Comportamento com tabelas gerenciadas ou não gerenciadas (externas)

Quando você descarta uma tabela gerenciada (ou seja, uma que você criou sem a palavra-chave **`EXTERNAL`**), o diretório de armazenamento da tabela será excluído. Isso significa que **você excluirá todos os dados dessa tabela**! Isso é verdade se o diretório de armazenamento da tabela estiver ou não no diretório de armazenamento do Hive. (A única exceção a essa regra é se o Hive ou Impala não tiver a permissão necessária para excluir os arquivos no diretório de armazenamento, por exemplo, se eles estiverem em um bucket do S3 ao qual Hive e Impala têm acesso apenas de leitura.)

No entanto, quando você descarta uma tabela não gerenciada (também chamada de gerenciada externamente), os dados no diretório de armazenamento da tabela não serão excluídos. Nesse caso, a Hive e a Impala entendem que esses dados se destinam a ser gerenciados fora de seu controle, não excluirá o diretório que contém os dados. (Isso é verdade mesmo se esse diretório estiver no diretório do armazém do Hive.)

Sempre tenha cuidado ao emitir uma instrução **`DROP TABLE`** e certifique-se de entender quais dados, se houver, serão perdidos.

Descartando um banco de dados que contém tabelas

Como um recurso de segurança, o Hive e o Impala lançarão um erro se você tentar descartar um banco de dados que contenha tabelas. Isso é para evitar a remoção não intencional de dados.

Você pode substituir esse recurso de segurança usando a palavra-chave **`CASCADE`** na instrução **`DROP DATABASE`**. A sintaxe é:

```
DROP DATABASE database_name CASCADE;
```

Use isso com muita cautela! Ele não apenas removerá o banco de dados, mas também todas as tabelas dentro dele, incluindo a exclusão dos dados de todas as tabelas gerenciadas no banco de dados.

Modificando tabelas existentes

Depois de criar uma tabela com Hive ou Impala, pode ser necessário modificar a definição da tabela. Isso pode ser porque houve um erro em sua instrução **`CREATE TABLE`**, ou porque a estrutura dos dados subjacentes foi alterada, ou talvez porque você precise usar uma convenção de nomenclatura diferente.

Para modificar uma definição de tabela, use a instrução **`ALTER TABLE`**. A sintaxe geral é

```
ALTER TABLE tablename ACTION parameters
```

A palavra-chave usada no lugar de **`ACTION`** depende do tipo de modificação que você deseja fazer. As seções a seguir apresentam algumas das modificações mais comuns. Existem outras possibilidades além das

apresentadas aqui. Você pode consultar a documentação (instrução Impala ALTER TABLE[2] ou Hive Language Manual DDL: Alter Table[3]) se quiser mais informações, mas neste ponto do curso, as opções apresentadas aqui provavelmente são suficientes.

Observação: às vezes é mais fácil simplesmente descartar sua tabela e recriá-la, em vez de fazer alterações na tabela, mas você deve ter cuidado para não excluir seus dados no processo. Nos RDBMSs tradicionais isso não é viável, mas nos sistemas de big data, os dados e metadados são separados. Se você criou uma tabela gerenciada externamente usando **`EXTERNAL`**, a eliminação dos dados é puramente uma operação de metadados e não afeta os dados. Se você tiver uma tabela gerenciada internamente, poderá torná-la gerenciada externamente primeiro (consulte a seção "Alterando para uma tabela não gerenciada (externa)" abaixo). Cabe a você decidir se prefere fazer as alterações em uma tabela ou soltá-la e recriá-la.

Renomeando uma Tabela

Para renomear uma tabela, use

```
ALTER TABLE nome_tabela_antiga RENAME TO nome_tabela_nova;
```

Por exemplo, isso renomeia a tabela customers para clients:

```
ALTER TABLE customers RENAME TO clients;
```

Quando você renomeia uma tabela, Hive ou Impala altera o nome da tabela no metastore e, se a tabela for gerenciada internamente, também renomeará o diretório da tabela no HDFS.

Movendo uma tabela para um banco de dados diferente

Para mover uma tabela para um banco de dados diferente, você também usa **`RENAME TO`** e especifica os nomes totalmente qualificados das tabelas antigas (existentes) e novas, incluindo os nomes do banco de dados:

```
ALTER TABLE old_database.tablename RENAME TO new_database.tablename;
```

Por exemplo, isso move a tabela existente chamada clientes do banco de dados default [padrão] para o banco de dados **dig**:

```
ALTER TABLE default.clients RENAME TO dig.clients;
```

Quando você move uma tabela para um banco de dados diferente, Hive ou Impala altera os metadados associados no **metastore** e, se a tabela for gerenciada, também moverá o diretório da tabela no HDFS para o subdiretório do banco de dados diferente.

Alterando o nome da coluna ou o tipo de dados

Para alterar o nome ou o tipo de dados de uma coluna, use

```
ALTER TABLE tablename CHANGE old_colname new_colname type;
```

Se você não estiver alterando o tipo de dados, ainda precisará fornecer o tipo. Se você não estiver alterando o nome da coluna, apenas o tipo de dados, repita o nome da coluna.

---

[2] Documentação ALTER TABLE do Impala:
https://www.cloudera.com/documentation/enterprise/latest/topics/impala_alter_table.html

[3] Documentação ALTER TABLE do Hive:
https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-AlterTable

Por exemplo, o seguinte altera a coluna **first_name** (do tipo **STRING**) na tabela de **employees** para **given_name** (mas mantém uma coluna **STRING**).

```
ALTER TABLE employees CHANGE first_name given_name STRING;
```

O exemplo a seguir altera o salário de **INT** para **BIGINT** sem alterar o nome da coluna:

```
ALTER TABLE employees CHANGE salary salary BIGINT;
```

Alterando a ordem das colunas (somente Hive)

Quando você cria uma tabela para dados existentes, suas colunas precisam ser fornecidas na mesma ordem em que aparecem nos arquivos de dados. Se você cometeu um erro e os colocou em uma ordem diferente na instrução **CREATE TABLE**, você pode corrigi-lo com a instrução **ALTER TABLE**.

Para alterar para onde uma coluna vai usando o Hive, use a palavra-chave **CHANGE** como se estivesse alterando o nome da coluna e adicione **AFTER *column*** ou **FIRST** no final.

Por exemplo, a tabela de **employees** na VM tem colunas nesta ordem: **empl_id, first_name, last_name, salary, office_id**. Suponha que você descubra que os dados do arquivo realmente listam o ID do escritório antes do salário do funcionário. O seguinte comando move **salary** para após a coluna **office_id**:

```
ALTER TABLE employees CHANGE salary salary INT AFTER office_id;
```

Se a coluna a ser movida precisar ser a primeira coluna (mais à esquerda), a instrução será:

```
ALTER TABLE tablename CHANGE col_name col_name col_type FIRST;
```

Observações

Esse recurso está disponível no Hive, mas não no Impala. Para Impala, consulte “Substituindo todas as colunas” para obter um método alternativo.

Você sempre precisa fornecer os nomes “antigos” e “novos” da coluna que está movendo, juntamente com seu tipo de dados, mesmo que esses detalhes não sejam alterados.

Isso não altera os arquivos de dados. Se você alterar a ordem das colunas para algo diferente da ordem nos arquivos, precisará recriar os arquivos de dados usando a nova ordem.

Adicionando ou removendo colunas

Você pode adicionar uma ou mais colunas ao final da lista de colunas usando **ADD COLUMNS**, ou (com Impala apenas) você pode excluir colunas usando **DROP COLUMN**. A sintaxe geral é

```
ALTER TABLE tablename ADD COLUMNS (col1 TYPE1, col2 TYPE2,… );
ALTER TABLE tablename DROP COLUMN colname;
```

Por exemplo, você pode adicionar uma coluna INT para bônus à tabela employees [de funcionários]:

```
ALTER TABLE employees ADD COLUMNS (bonus INT);
```

Ou você pode remover a coluna **office_id** da tabela de employees [funcionários]:

```
ALTER TABLE employees DROP COLUMN office_id;
```

Observações

DROP COLUMN não está disponível no Hive, apenas no Impala. No entanto, consulte "Substituindo todas as colunas" abaixo.

Você só pode descartar uma coluna de cada vez. Para descartar várias colunas, use várias instruções ou use o método para substituir colunas (veja abaixo).

Você não pode adicionar uma coluna no meio da lista e não no final. Você pode, no entanto, adicionar a coluna e alterar a ordem (veja acima) ou usar o método para substituir colunas (veja abaixo).

Assim como a alteração da ordem das colunas, isso não altera os arquivos de dados.

- Se a definição da tabela estiver de acordo com os arquivos de dados antes de você descartar qualquer coluna diferente da última, será necessário recriar os arquivos de dados sem os valores da coluna descartada.
- Se você descartar a última coluna, os dados ainda existirão, mas serão ignorados quando uma consulta for emitida.
- Se você adicionar colunas para as quais não existem dados, essas colunas serão **NULL** em cada linha.

Substituindo todas as colunas

Você também pode substituir completamente todas as colunas por uma nova lista de colunas. Isso é útil para descartar várias colunas ou se você precisar adicionar colunas no meio da lista. A sintaxe geral é

```
ALTER TABLE tablename REPLACE COLUMNS (col1 TYPE1, col2 TYPE2,… );
```

Isso remove completamente a lista de colunas existente e a substitui pela nova lista. Somente as colunas especificadas na instrução **ALTER TABLE** existirão e estarão na ordem que você fornecer.

Observação

Novamente, isso não altera os arquivos de dados, apenas os metadados da tabela, portanto, você desejará que a nova lista corresponda aos arquivos de dados ou precisará recriar os arquivos de dados para corresponder à nova lista.

Mudando para uma tabela não gerenciada (externa)

Se você criou uma tabela como uma tabela gerenciada (sem a palavra-chave **EXTERNAL**) e depois percebe que deseja que ela não seja gerenciada, você pode usar **ALTER TABLE** com TBLPROPERTIES para torná-la não gerenciada. Isso é particularmente útil se você deseja descartar uma tabela sem perder os dados.

A sintaxe geral é

```
ALTER TABLE tablename SET TBLPROPERTIES('EXTERNAL'='TRUE');
```

Observações

Ambos **EXTERNAL** e **TRUE** estão entre aspas e devem estar em letras maiúsculas, aqui.

Você também pode usar **SET TBLPROPERTIES** com outras propriedades que não foram definidas na criação.

Experimente!

Teste-os com a tabela de **investors**.

A primeira coisa que você fará é verificar se a tabela não é gerenciada (**external**) para que você possa descartar a tabela sem perder os dados, caso cometa um erro. Você pode então recriar a tabela usando o exercício da "The ROW FORMAT Clause" na lição "The CREATE TABLE Statement", onde você criou a tabela de **investors** pela primeira vez. (Você pode deixar essa alteração, não há necessidade de alterá-la de volta.) Se ela não for gerenciada, você deverá alterá-la.

1. No Hive ou Impala, execute o seguinte na tabela de **customers** para que você possa ver como é uma tabela gerenciada. Certifique-se de que o banco de dados **default** seja seu banco de dados ativo.

```
DESCRIBE FORMATTED customers;
```

Veja os resultados para **Tipo de Tabela**; o valor deve ser **MANAGED_TABLE**. Isso significa que ele foi criado sem a palavra-chave **EXTERNAL**.

2. Agora execute o mesmo comando na tabela de **investors** e observe qual é o valor para **Tipo de Tabela**. Se também for **MANAGED_TABLE**, execute o seguinte para alterá-lo para uma tabela não gerenciada e, em seguida, execute a instrução **DESCRIBE FORMATTED** novamente e verifique o valor para **Table Type**. (Se o valor não for **MANAGED_TABLE**, você não precisa executar este comando—embora não prejudique se você o fizer.)

```
ALTER TABLE investors SET TBLPROPERTIES('EXTERNAL'='TRUE');
```

Em seguida, tente alterar o nome.

3. Execute a seguinte instrução:

```
ALTER TABLE investors RENAME to dig.investors;
```

4. Verifique se o nome da tabela mudou no painel da fonte de dados (atualize a tela se necessário), ou executando **SELECT * FROM companies**;

5. Altere o nome de volta para **investors** e verifique a alteração.

Agora mova-o para o banco de dados **dig**.

6. Execute a seguinte instrução:

```
ALTER TABLE default.investors RENAME TO dig.investors;
```

7. Verifique se a tabela não está mais no banco de dados **default** e se está no banco de dados **dig**.

8. Verifique o diretório Hive **warehouse**. O subdiretório **investors** não foi movido para o diretório **dig.db** — ele ainda está no banco de dados **default**. Você sabe por quê? (Veja a seção abaixo para a resposta!)

9. Altere o diretório de volta ao **default** e verifique a alteração.

Altere a coluna **amount** para **holdings**.

10. Execute a seguinte declaração:

```
ALTER TABLE investors CHANGE amount holding INT;
```

11. Verifique se o nome da coluna mudou, usando o painel de fonte de dados ou executando o **DESCRIBE investors;**

12. Altere o nome da coluna de volta para **amount** e verifique a alteração.

Altere a coluna **amount** de **INT** para **BIGINT**.

13. Execute a seguinte instrução:

```
ALTER TABLE investors CHANGE amount amount BIGINT;
```

14. Verifique se o tipo da coluna foi alterado, usando o painel de fonte de dados ou executando **DESCRIBE investors;**

15. Altere o tipo de coluna de volta para **INT** e verifique a alteração.

Use Hive (não Impala) para colocar a coluna de **amount** no final em vez do meio e veja o efeito.

16. Primeiro dê uma olhada nos dados executando **SELECT * FROM investors;**
17. Execute a seguinte declaração:

```
ALTER TABLE investors CHANGE name name STRING AFTER amount;
```

18. Verifique se as colunas foram reordenadas nos metadados, mas não nos próprios dados, executando

```
SELECT * FROM investidores;
```

Observe que a **amount** é **NULL** em cada linha - isso ocorre porque os dados que estão sendo carregados são os dados de caracteres não numéricos que costumavam entrar em **name**. Como a **amount** é uma coluna **INT**, os dados não são válidos. Os dados inteiros destinados a **amount** são válidos para a coluna **name,** que é STRING, portanto, esses valores não são **NULL**. (O ponto aqui é que os dados em si não mudaram.)

19. Altere a ordem das colunas de volta executando

```
ALTER TABLE investors CHANGE name name STRING FIRST;
```

20. Verifique se as colunas voltaram ao normal usando a consulta **SELECT *.**

Use Impala para apagar uma coluna e adicioná-la de volta. (Você pode adicionar com Hive ou Impala, mas Hive não reconhece a palavra-chave **DROP COLUMN**, então para facilitar as coisas, use Impala para ambos.)

21. Primeiro apague a coluna **share**:

```
ALTER TABLE investors DROP COLUMN share;
```

22. Execute a consulta **SELECT *** para verificar se a coluna foi eliminada. Os valores nas outras colunas não foram alterados.
23. Adicione a coluna **share** de volta:

```
ALTER TABLE investors ADD COLUMNS (share DECIMAL(4,3));
```

24. Verifique se a coluna foi restaurada.

Finalmente, mude as colunas completamente!

25. Execute a seguinte declaração:

```
ALTER TABLE investors REPLACE COLUMNS (company STRING, holding BIGINT, share
DOUBLE);
```

26. Verifique se as colunas da tabela foram alteradas usando os **DESCRIBE investors;**
27. Restaure as colunas originais:

```
ALTER TABLE investors REPLACE COLUMNS
   (name STRING, amount INT, share DECIMAL(4,3));
```

28. Verifique se a tabela foi alterada novamente.

Considerações

O diretório da tabela de **investors** não foi movido quando movido para um banco de dados diferente porque é uma tabela não gerenciada. Assim como descartar a tabela não excluirá os dados, alterar o banco de dados não moverá os dados.

Tente mover **customers**, que é gerenciado, para o banco de dados **dig** e confirme se o diretório da tabela foi movido nesse caso. (Ou seja, **/user/hive/warehouse/** não deve mais ter o subdiretório **customers/**, mas **/user/hive/warehouse/dig.db/** deve tê-lo.) Talvez seja necessário atualizar a exibição. Certifique-se de mover a tabela de volta para o banco de dados **default** quando terminar.

3.5. Interoperabilidade Hive - Impala

Em geral, Hive e Impala trabalham bem juntos. Se você criar uma tabela usando o Hive, poderá consultá-la no Hive e Impala. Criando uma tabela no Impala, também é possível usar ambos os mecanismos para consultá-la.

Isso é em geral. Agora você já deveria ter lido sobre algumas técnicas que funcionam no Hive, mas não Impala. Como o comando **ALTER TABLE** para alterar a ordem das colunas. Há também alguns comandos que funcionam no Impala, mas não no Hive. Como o comando para apagar uma coluna. Há também algumas partes da sintaxe **CREATE TABLE**, que uma ou outra pode não suportar. Por exemplo, no Impala, você pode usar a cláusula **LIKE PARQUET** para criar uma tabela usando essas informações de esquema incorporadas dentro de um arquivo de parquet. Mas o Hive não suporta esta cláusula LIKE PARQUET.

Também é possível que uma tabela criada com um desses mecanismos não possa ser consultada pelo outro. Por exemplo, se você usar algum SerDe como este em uma instrução CREATE TABLE no Hive, então Impala não será capaz de consultar essa tabela. Assim, enquanto você pode confiar principalmente em tabelas disponíveis em ambos os motores, lembre-se que existem algumas coisas que exigem um motor específico. Se você receber mensagens de erro de um mecanismo, verifique se esse mecanismo suporta o que você está tentando fazer.

Lembre-se de que o Hive é tipicamente mais lento do que o Impala, mas o Hive é mais geral do que o Impala nos tipos de formatos de arquivo que ele suporta para tabelas. Impala é projetado para ser muito mais rápido e por isso é especializado no uso dos melhores formatos de arquivo para consultas rápidas, formatos como Apache Parquet.

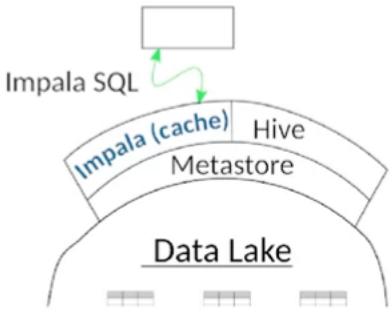
Quando você está no trabalho, muitas vezes você vai encontrar-se usando o Hive e sua grande variedade de SerDes disponíveis para ler dados em muitos formatos diferentes. Em seguida, Hive pode colocar esses dados em novas tabelas em um formato como Parquet para consulta rápida usando Impala.

Impala Metadata Refresh

Embora você geralmente possa usar Hive ou Impala para criar tabelas de consulta final, há uma diferença importante entre como Hive e Impala acessam a metastore. O Hive recupera metadados do metastore toda vez que ele cria uma consulta, mas o Impala não.

Impala armazena em cache metadados na memória para reduzir a latência da consulta. Esses metadados de cache Impala consiste na estrutura e locais das tabelas recuperadas do metastore, e também informações adicionais sobre arquivos de dados de tabela recuperados do sistema de armazenamento de dados, como HDFS ou S3.

O cache de metadados do Impala ajuda a retornar os resultados da consulta o mais rápido possível. Mas os metadados armazenados em cache podem ficar fora de sincronia com os metadados na metastore, e com os arquivos de dados armazenados. Isso acontece quando as alterações são feitas fora do Impala. Por exemplo, quando novas tabelas são criadas usando hive, quando os dados da tabela são importados usando o navegador da tabela do Hue, ou quando os dados da tabela são adicionados usando o comando HDFS. Quando alterações como essa ocorrem fora do Impala, é necessário atualizar o cache de metadados do Impala.



Existem diferentes maneiras de fazer isso dependendo de quais mudanças foram feitas fora do Impala. O comando **refresh** atualiza as informações que o Impala armazena em cache sobre o esquema de uma tabela específica e os locais e arquivos dessa tabela no sistema de armazenamento de dados.

External Metadata Change	Required Action	Effect on Local Caches
Table schema modified or new data added to a table	`REFRESH tablename;`	Reloads the metadata for one table immediately; reloads storage block locations for new data files only
New table added, or data in a table extensively altered, such as by HDFS balancing	`INVALIDATE METADATA tablename;`	Marks the metadata for a single table as stale; when the metadata is needed, all storage block locations are retrieved

Use este comando se você tiver alterado um esquema de tabela, como renomear uma coluna ou adicionar dados à tabela. A sintaxe é **`REFRESH tablename`**.

Mas se você adicionou uma nova tabela ao banco de dados de fora do Impala, então você precisará usar um comando diferente para atualizar o cache de metadados do Impala. O comando é **`INVALIDATE METADATA tablename`**. Esse comando faz com que Impala adicione todas as informações sobre essa nova tabela ao seu cache de metadados. Finalmente, você também pode usar metadados invalidados sem especificar uma tabela. Mas tenha cuidado ao usar isso, ele marcará os metadados para todas as tabelas como tale [cauda] e recarregará todos os metadados quando uma nova consulta for emitida.

Para um grande ambiente de produção com muitas tabelas, esta pode ser uma operação muito cara e pode levar muito tempo. Observe que quando o próprio Impala modifica o metastore ou qualquer arquivo armazenado, ele pode atualizar automaticamente os metadados do cache. Portanto, são apenas mudanças de fora do Impala que exigem que você use qualquer um desses comandos.

4. Tipos de Dados e de Arquivos

Neste ponto você já tem algum conhecimento de tipos de dados. Lembre-se de que cada coluna em uma tabela tem um tipo de dados específico. Você já usou alguns tipos de dados como STRINGS e INT em criar instruções de tabela ao longo do curso. Neste capítulo, descreveremos mais detalhes sobre estes e sobre os outros tipos de dados que você pode usar com Hive e Impala. Você aprenderá sobre os limites dos diferentes tipos de dados, o que eles se destinam e como escolher os certos ao criar uma tabela.

Em seguida, você aprenderá sobre os tipos de arquivos. Estes são os diferentes formatos de arquivo que você pode usar para armazenar os dados nas tabelas Hive e Impala. Você já aprendeu um pouco sobre formatos de arquivo e viu alguns exemplos de arquivos em formato de texto delimitado. Então, vamos abordar e revisar rapidamente o formato do arquivo de texto e, em seguida, entrar em mais detalhes sobre três outros formatos de arquivo populares, Avro, Hark e NRC.

Você aprenderá o que há de diferente nesses formatos de arquivo, ao escolher um deles em vez de arquivos de texto e como criar tabelas com arquivos em diferentes formatos. Também daremos uma rápida revisão de alguns outros formatos de arquivo que são menos utilizados.

4.1. Tipos de Dados

Tipos de dados inteiros

Hive e Impala suportam quatro tipos de dados inteiros: TINYINT, SMALLINT, INT e BIGINT. Estes representam números inteiros sem partes fracionárias.

Tipos inteiros maiores permitem que você represente intervalos maiores de números, conforme mostrado na tabela abaixo, mas processá-los requer mais memória, portanto, geralmente você deve usar o menor tipo inteiro que acomode todo o intervalo de valores em seus dados.

Integer Type	Range
TINYINT	-128 to 127
SMALLINT	-32,768 to 32,767
INT	-2,147,483,648 to 2,147,483,647 (approximately 2.1 billion)
BIGINT	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 (approximately 9.2 quintillion)

Ao contrário de alguns bancos de dados relacionais, todos os tipos inteiros no Hive e Impala são assinados; ou seja, eles podem representar valores inteiros positivos ou negativos. Não há tipos inteiros não assinados.

Tipos de dados decimais

Hive e Impala suportam três tipos de dados decimais: FLOAT, DOUBLE e DECIMAL. Esses tipos representam números que podem incluir partes fracionárias.

Os tipos FLOAT e DOUBLE representam números de ponto flutuante, que não possuem um número predeterminado de dígitos após o ponto decimal. DOUBLE oferece maior alcance e precisão do que FLOAT, mas processar DOUBLEs requer mais memória do que processar FLOATs, portanto, em geral, escolha FLOAT, a menos que você precise do intervalo ou precisão que DOUBLE oferece.

Por causa do sistema binário usado para armazenar números, os tipos de dados FLOAT e DOUBLE podem produzir imprecisões inesperadas, mesmo com aritmética aparentemente simples como 0,1 + 0,2. (Veja o Guia de Ponto Flutuante para mais informações sobre isso.)

DOUBLE é mais preciso porque DOUBLE usa 64 bits para armazenar cada número, enquanto FLOAT usa apenas 32 bits. (Portanto, DOUBLE tem o dobro do número de bits.) Isso significa que FLOAT normalmente tem precisão de até 7 dígitos, enquanto DOUBLE tem precisão de até 15 ou talvez 16 dígitos.

O tipo DECIMAL representa números com precisão e escala fixas. Ao criar uma coluna DECIMAL, você especifica a precisão, p, e a escala, s. Precisão é o número total de dígitos, independentemente da localização do ponto decimal. Escala é o número de dígitos após a casa decimal. Para representar o número 8,54 sem perda de precisão, você precisaria de um tipo DECIMAL com precisão de pelo menos 3 e escala de pelo menos 2.

A tabela abaixo ilustra a diferença de precisão usando resultados para π. Observe que o tipo DECIMAL(17,16) significa que há um total de 17 dígitos, com 16 deles após o ponto decimal.

Data Type	Result for π (bold are accurate)
FLOAT	**3.14159**7410125732
DOUBLE	**3.1415926535897793**1
DECIMAL(17,16)	**3.1415926535897932**

Usando o tipo DECIMAL, é possível representar números com maior precisão do que os tipos FLOAT ou DOUBLE podem representar. A precisão e escala máximas permitidas do tipo DECIMAL são ambas 38. (O Hive pode permitir valores maiores de precisão e escala, mas o Impala não os suporta.)

A tabela abaixo descreve o intervalo de DOUBLE, FLOAT e DECIMAL(38,0). Os intervalos descritos abaixo são o maior número negativo e o maior número positivo que cada tipo de dados pode representar.

Data Type	Range
FLOAT	-3.40282346638528860 * 10^38 to 3.40282346638528860 * 10^38
DOUBLE	-1.79769313486231570 x 10^308 to 1.79769313486231570 x 10^308
DECIMAL(38,0)	-10^38 + 1 to 10^38 - 1

Para representar moeda, você deve usar DECIMAL em vez de FLOAT ou DOUBLE; isso evita a perda de precisão, que normalmente é de suma importância com dados financeiros. Outra opção para moeda é usar um tipo inteiro para representar, por exemplo, o número de centavos, em vez de armazenar dólares com partes fracionárias.

Tipos de dados de cadeia de caracteres

Hive e Impala suportam três tipos de dados de caracteres: STRING, CHAR e VARCHAR. Esses tipos representam valores de texto alfanuméricos.

O tipo de dados STRING representa uma sequência de caracteres sem restrição de comprimento especificada.*

Se você está familiarizado com bancos de dados relacionais, provavelmente está mais acostumado com os tipos de caracteres CHAR e VARCHAR, que possuem um comprimento especificado.

O tipo CHAR representa sequências de caracteres de comprimento fixo, com um comprimento especificado preciso. Os valores maiores que o comprimento especificado são truncados. Os valores menores que o comprimento especificado são preenchidos com espaços. Se você atribuir o valor de 13 caracteres às regras do Impala! para uma coluna CHAR com comprimento 16, então Hive e Impala preencherão esse valor com três espaços para torná-lo 16 caracteres: Regras Impala!␣␣␣ (Os três símbolos mostrados neste exemplo representam espaços.)

O tipo VARCHAR representa sequências de caracteres com um comprimento máximo especificado.

Valores maiores que o máximo são truncados, mas valores menores que o máximo não são preenchidos com espaços. Se você tentar atribuir o valor de 13 caracteres às regras do Impala! em uma coluna VARCHAR com comprimento máximo de 10, então Hive e Impala truncarão esse valor para 10 caracteres, descartando os três últimos caracteres: Impala rul. No entanto, se o comprimento máximo for 13 ou mais, o valor armazenado será exatamente as regras do Impala! (sem espaços extras como você obteria com o tipo CHAR). A tabela aqui resume esses exemplos.

Data Type	Description	Value (attempting Impala rules!)
STRING	Any number of characters	**Impala rules!**
CHAR(10)	Exactly 10 characters	**Impala rul**
CHAR(16)	Exactly 16 characters	**Impala rules!␣␣␣**
VARCHAR(10)	At most 10 characters	**Impala rul**
VARCHAR(16)	At most 16 characters	**Impala rules!**

Com os tipos CHAR, os espaços à direita são ignorados nas comparações. Com os valores VARCHAR e STRING, quaisquer espaços à direita são considerados nas comparações. (Isso faz sentido, já que nenhum deles é preenchido automaticamente - espaços à direita não são considerados "preenchimento" nesses casos.)

Geralmente, você deve escolher STRING em vez de CHAR ou VARCHAR. O STRING oferece maior flexibilidade e facilidade de uso e, em alguns casos, o Hive e o Impala apresentam melhor desempenho e compatibilidade ao usar colunas STRING. Mas se você tiver uma necessidade específica de valores de string com comprimentos precisos ou com comprimentos máximos, poderá usar CHAR ou VARCHAR.

***Nota de rodapé: Limites reais de string**

Na verdade, existem limites práticos para o comprimento das strings, embora na maioria dos aplicativos do mundo real seja improvável que você os encontre. Por exemplo, no Impala, estas são as considerações para comprimentos de strings (retiradas do STRING Data Type na documentação do Impala da Cloudera):

O limite rígido para o tamanho de uma STRING e o tamanho total de uma linha é de 2 GB.

Se uma consulta tentar processar ou criar uma string maior que esse limite, ela retornará um erro ao usuário. O limite é de 1 GB em STRING ao gravar em arquivos Parquet.

As consultas que operam em strings com 32 KB ou menos funcionarão de maneira confiável e não causarão problemas significativos de desempenho ou memória (a menos que você tenha consultas muito complexas, muitas colunas e assim por diante).

O desempenho e o consumo de memória podem diminuir com strings maiores que 32 KB.

Isso varia um pouco de acordo com a versão do Impala que você está usando, portanto, se você estiver trabalhando com strings excepcionalmente grandes, verifique a documentação.

Outros tipos de dados

Além dos tipos de dados inteiros, decimais e caracteres, Hive e Impala suportam vários outros tipos de dados simples.

O tipo BOOLEAN representa um valor booleano, ou seja, um valor verdadeiro ou falso.

O tipo TIMESTAMP representa um instante no tempo. TIMESTAMPs podem representar valores com precisão de até nanossegundos. Eles são interpretados como sendo em UTC, ou Tempo Universal Coordenado, mas Hive e Impala fornecem funções para conversão para fusos horários locais.

O Hive (mas não o Impala) fornece um tipo DATE, representando um dia específico no formato AAAA-MM-DD, sem hora do dia. Com o Hive, um TIMESTAMP pode ser retirado da hora do dia por conversão para um tipo DATE.

Há também o tipo BINARY, que pode representar qualquer sequência de bytes brutos, e também é suportado apenas pelo Hive, não pelo Impala. Isso é análogo ao tipo VARBINARY em alguns bancos de dados relacionais. A tabela abaixo resume esses tipos.

Data Type	Description	Example Value
BOOLEAN	True or false	**true**
TIMESTAMP	Instant in time	2019-02-25 16:51:05
DATE (Hive only)	Date without time of day	2019-02-25
BINARY (Hive only)	Raw bytes	N/A

Hive e Impala também suportam tipos complexos (ARRAY, MAP e STRUCT).

4.2. Trabalhando com Tipos de Dados

Examinando tipos de dados

Se você não tiver certeza de quais tipos de dados são atribuídos a uma coluna—talvez seja uma tabela existente que outra pessoa criou, você acha que cometeu um erro ou simplesmente esqueceu como a definiu—há diferentes maneiras de obter essas informações .

Examine o esquema da tabela

Claro, você pode usar o **`Table Browser do Hue`**, ou o painel de fonte de dados à esquerda no Hue, para visualizar o esquema da tabela. Você pode visualizar os nomes das colunas junto com seus tipos de dados. Como você aprendeu na leitura da Semana 2, "Examinando a Estrutura da Tabela", você também pode usar os comandos **`DESCRIBE`** ou **`DESCRIBE FORMATTED`**. Ambos mostram quais colunas estão na tabela, com seus tipos de dados e, às vezes, comentários. **`DESCRIBE FORMATTED`** fornece um pouco mais de informações, incluindo o formato e a localização dos arquivos de dados da tabela.

O comando **`SHOW CREATE TABLE`** também pode ser usado para ver a definição de uma tabela. Você pode ler o comando CREATE TABLE resultante para ver quais são as colunas, incluindo quais são seus tipos de dados.

A função typeof no Impala

No Impala (mas não no Hive), você também pode usar a função typeof em uma instrução **`SELECT`** para obter o tipo de dados:

```
SELECT typeof(colname) FROM tablename LIMIT 1;
```

Se você não usar LIMIT 1, ele retornará uma linha para cada linha da tabela.

Esse método também é útil para ver qual tipo de dados uma expressão retorna. Examinar diretamente uma tabela, seja usando a interface gráfica do Hue ou usando um comando, não fornecerá essa informação. Em vez de colname na consulta acima, use a expressão de seu interesse. Se a expressão não envolver uma referência de coluna, o Impala permitirá que você deixe de fora a cláusula FROM.

Experimente!

Siga as etapas abaixo para praticar o uso da função typeof com o Impala. Se você quiser praticar o uso de **`DESCRIBE`** ou **`SHOW CREATE TABLE,`** consulte "Examinando a estrutura da tabela" na Semana 2.

Use a função **`typeof`** para localizar o tipo de dados da coluna **`list_price`** na tabela **`fun.games`**. (Use a instrução **`SELECT`** acima, substituindo **`colname`** e **`tablename`**.)

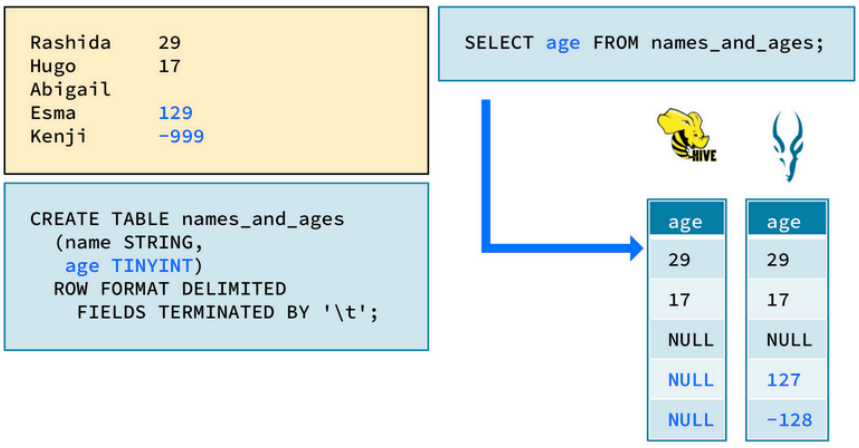
Alguns governos adicionam um imposto sobre vendas às compras de itens como jogos. O valor do imposto é uma porcentagem do preço pago. Por exemplo, um imposto sobre vendas de 7% tornaria o imposto de um jogo **`0,07 * list_price`**, e o custo final seria **`1,07 * list_price`**. Encontre o tipo de dados dessa expressão para os jogos na tabela **`fun.games`**.

O Impala permitirá que você omita a cláusula **`FROM`** quando não houver referência de coluna. Use a função typeof no Impala para encontrar o tipo de dados das seguintes expressões. (Tudo que você precisa é a expressão **`SELECT`**;)

a) 0,6
b) cos(0,6)
c) ceil(cos(0,6))
d) ceil(cos(0,6))/3

Valores fora do intervalo

Tanto o Hive quanto o Impala retornam **`NULL`** para valores **`DECIMAL`** que estão fora do intervalo para o tipo de dados de destino (como 23,63 em uma coluna **`DECIMAL(3,2)`**). Ambos também retornam **`-Infinity`** ou **`Infinity`** para os tipos **`FLOAT`** e **`DOUBLE`** quando o valor é muito grande e zero quando o valor é próximo de zero, mas muito pequeno para o intervalo do tipo de dados. (Esses valores zero podem ser renderizados de maneiras ligeiramente diferentes, por exemplo: 0, 0,0, -0 ou -0,0). No entanto, há uma diferença importante entre como o Hive e o Impala lidam com valores fora do intervalo em colunas inteiras: o Hive retorna **`NULL`** para valores inteiros fora do intervalo, enquanto o Impala retorna o valor mínimo ou máximo para o tipo de dados inteiro específico da coluna. O exemplo mostrado abaixo ilustra isso.



Neste exemplo, há uma tabela que inclui uma coluna chamada idade com o tipo de dados **`TINYINT`**. O tipo de dados **`TINYINT`** pode representar valores inteiros de **`-128`** a **`+127`**. Mas o arquivo de dados para esta tabela (representado pela caixa amarela) contém dois valores de idade errôneos que estão fora desse intervalo. O valor de idade para Esma é **`129`**, que está acima do valor máximo para o tipo **`TINYINT`**. O valor de idade para Kenji é **`-999`**, que está abaixo do valor mínimo para o tipo TINYINT.

Quando você consulta essa tabela com o Hive, ela retorna **`NULL`** para esses dois valores fora do intervalo, mas o Impala se comporta de maneira diferente. Impala retorna **`127`** para a idade de Esma (**`127`** é o valor

máximo para o tipo **`TINYINT`**). E Impala retorna **`-128`** para a idade de Kenji (**`-128`** é o valor mínimo para o tipo **`TINYINT`**).

Por que o Impala se comporta de maneira diferente do Hive nessa situação? Observe no exemplo mostrado aqui que a idade de Abigail está ausente no arquivo de dados no HDFS. Os resultados da consulta Impala permitem distinguir entre o valor ausente da idade de Abigail (**`NULL`**) e os valores fora do intervalo das idades de Esma e Kenji, enquanto os resultados da consulta Hive não permitem distinguir entre eles.

No entanto, uma implicação disso é que você deve escolher um tipo de dados cujos valores maiores e menores não ocorram em seus dados. Por exemplo, se seus dados contiverem números inteiros variando de **`-127`** a **`+126`**, não há problema em usar o tipo de dados **`TINYINT`**. Nesse caso, um valor de **`-128`** ou **`+127`** no resultado da consulta indicará inequivocamente um valor fora do intervalo. Mas se seus dados contiverem inteiros variando de **`-128`** a **`+127`**, você deve escolher **`SMALLINT`** para evitar ambiguidade entre os valores reais **`-128`** e **`+127`** e valores fora do intervalo.

4.3. Tipos de Arquivos

Lembre-se de que os dados nas tabelas Hive e Impala são armazenados em arquivos em um diretório no HDFS ou S3. Não há apenas um formato de arquivo fixo para esses dados de tabela. Hive e Impala suportam vários formatos de arquivo diferentes. Quando você cria uma tabela com Hive e Impala, você especifica qual formato usar no armazenado como Cláusula da instrução criar tabela.

Serão descritos alguns formatos de arquivo diferentes que Hive e Impala suportam. Nos concentraremos nos quatro que são usados com mais frequência; Textfiles, arquivos Avro, arquivos Parquet e arquivos ORC. Descreveremos as vantagens e desvantagens de usar cada um deles. Principalmente em termos de trade-offs, entre legibilidade humana e desempenho e também interoperabilidade com outras ferramentas com as quais você pode trabalhar.

Diferentes padrões para armazenamento, ingestão ou consulta podem tornar um formato uma escolha melhor do que outro formato. Com o big data, as diferenças podem ter um impacto significativo no desempenho. A escolha do formato depende em parte do mecanismo de consulta que você está usando. Textfiles, arquivos Avro e arquivos Parquet são compatíveis com Hive e Impala. Mas os arquivos ORC são compatíveis apenas com o Hive. Também darei uma visão geral rápida de alguns outros formatos de arquivo que são usados com menos frequência. Os formatos SequenceFile e RCFile.

Às vezes, a escolha de qual formato de arquivo usar quando você cria uma tabela é baseada simplesmente em qual formato de arquivo os dados já estão. Por exemplo, se você tiver um diretório HDFS ou um bucket S3 com arquivos Parquet já nele, então você pode criar uma tabela base Parquet para consultar os dados existentes com Hive e Impala sem movê-los ou copiá-los. Mas em outras situações, você terá a flexibilidade de escolher qual formato de arquivo usar e converterá dados nesse formato enquanto os carrega no diretório de armazenamento da tabela.

Você aprenderá mais sobre isso mais adiante. Por enquanto, lembre-se que é possível converter dados entre esses diferentes formatos de arquivo. Assim, você não ficará preso em um beco sem saída por sua escolha de formato de arquivo. No entanto, fazer a escolha certa quando você cria uma tabela pela primeira vez pode evitar a necessidade de converter os arquivos posteriormente, o que pode ser uma operação dispendiosa se os dados tiverem crescido muito.

4.3.1 Arquivos de texto

Arquivos de texto são o tipo de arquivo mais básico. Praticamente qualquer linguagem de programação pode ler e gravar dados em arquivos de texto, e muitos aplicativos que os analistas de dados usam podem

trabalhar com arquivos de texto delimitados por vírgulas e tabulações. Eles também são legíveis por humanos. Os valores são representados como strings de texto simples, para que você possa simplesmente abrir um editor de texto e visualizar os dados. Isso é útil quando você está investigando um problema.

No entanto, existem desvantagens em usar arquivos de texto. Quando usados para armazenar grandes quantidades de dados, os arquivos de texto são ineficientes. Representar valores numéricos como strings desperdiça muito espaço de armazenamento. É difícil representar dados binários como imagens em um arquivo de texto; isso requer o uso de técnicas como a codificação Base64. A conversão de dados entre representações de texto e seus tipos de dados nativos requer serialização e desserialização, o que diminui o desempenho.

No geral, os arquivos de texto oferecem excelente interoperabilidade, mas desempenho ruim.

4.3.2 Arquivos Avro

Apache Avro é uma estrutura de serialização de dados eficiente. O Avro define um formato de arquivo que usa codificação binária otimizada para armazenar dados com eficiência. (Para obter uma explicação sobre codificação binária, consulte "O que é codificação binária"[4] de wisegeek.com.) O formato Avro é amplamente suportado por ferramentas de big data e também foi projetado para funcionar em diferentes linguagens de programação e ferramentas fora do sistema típico de big data . Os arquivos Avro são adequados para armazenamento de dados de longo prazo.

Um arquivo de dados Avro inclui uma definição de esquema incorporada, que torna o arquivo *autodescritivo* — o próprio arquivo fornece informações sobre o que está no arquivo. O Avro também foi desenvolvido para lidar com a evolução do esquema. Isso significa que é possível adicionar, remover ou modificar colunas em uma tabela Hive ou Impala sem a necessidade de fazer alterações nos dados existentes armazenados em arquivos Avro. A estrutura Avro acomodará essas alterações de esquema, portanto, a tabela e os arquivos de dados existentes continuarão compatíveis, mesmo que seus esquemas não correspondam perfeitamente. (Para obter mais informações sobre a evolução do esquema, consulte "Como funciona a evolução do esquema"[5] na página vinculada. Observe que a maioria dos detalhes nesse artigo pertence aos esquemas Avro em geral, mas alguns detalhes são específicos ao uso do Avro no banco de dados Oracle NoSQL e, portanto, não são aplicáveis aos sistemas de big data apresentados neste curso.) No geral, o Avro oferece excelente interoperabilidade e excelente desempenho, tornando-o uma escolha popular para armazenamento de big data de uso geral.

4.3.3 Arquivos Parquet

Apache Parquet é um formato de arquivo *colunar* desenvolvido originalmente por engenheiros da Cloudera e Twitter. O Parquet foi inspirado em um projeto do Google chamado Dremel. Os formatos de arquivo colunares ou orientados a colunas organizam os dados por coluna, em vez de por linha. Isso os torna mais eficientes quando você precisa processar apenas uma ou algumas colunas. Por exemplo, veja a Figura 1 abaixo. As linhas (1, 2, 3) e colunas (A, B, C) dos dados são mostradas em uma estrutura tabular no lado esquerdo. As imagens do lado direito representam como esses dados podem ser armazenados em um formato de arquivo orientado por linha (superior) e um formato de arquivo orientado por coluna (inferior).

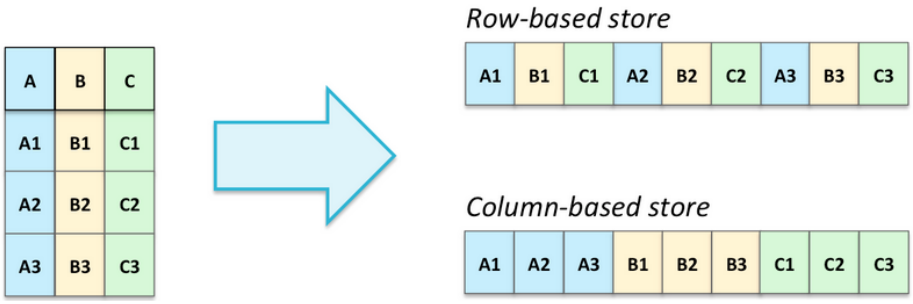
Quando os dados são armazenados em um formato orientado por linha, o arquivo organiza os dados sequencialmente primeiro por linha (a linha 1 consiste em A1, B1, C1). Quando os dados são armazenados

[4] O que é codificação binária: https://www.wisegeek.com/what-is-binary-encoding.htm

[5] Como funciona a evolução de esquemas:
https://docs.oracle.com/cd/E26161_02/html/GettingStartedGuide/schemaevolution.html#schemaevolutionhow

assim, o Hive e o Impala devem ler cada linha completa, mesmo que a consulta exija o processamento de apenas uma coluna.

Quando os dados são armazenados em um formato orientado a colunas, o arquivo organiza os dados sequencialmente primeiro por coluna (a coluna A possui os valores A1, A2, A3). Quando os dados são armazenados assim, o Hive e o Impala podem pular as colunas que não fazem parte da consulta e ler rapidamente apenas os valores das colunas que fazem parte da consulta. Isso melhora o desempenho da consulta, especialmente para tabelas com dezenas ou centenas de colunas.



O Parquet é amplamente suportado por ferramentas de big data, incluindo Hive e Impala, e foi projetado para funcionar em diferentes linguagens de programação. Assim como o Avro, o Parquet incorpora uma definição de esquema no arquivo e oferece suporte à evolução do esquema.

Parquet usa otimizações avançadas para armazenar dados de forma mais compacta e para acelerar as consultas. É mais eficiente quando os dados são carregados de uma só vez ou em grandes lotes; isso permite que o Parquet aproveite os padrões repetidos nos dados para armazená-los com mais eficiência. (Veja a figura abaixo.)

id	name	city
1	Alice	Palo Alto
2	Bob	Sunnyvale
3	Bob	Palo Alto
4	Bob	Berkeley
5	Carol	Berkeley

Parquet também usa compressão. A compactação codifica os dados de forma a ocupar menos espaço de armazenamento do que um arquivo descompactado, mas há um custo de tempo quando você lê ou grava o arquivo. Ou seja, a codificação para menos armazenamento significa mais tempo necessário para compactar (codificar) o arquivo antes de gravá-lo ou descompactar (decodificar) o arquivo depois de lê-lo.

No geral, o Parquet oferece excelente interoperabilidade e excelente desempenho, tornando-o uma escolha popular para armazenamento de dados em colunas.

4.3.4 Arquivos ORC

Apache ORC (Optimized Record Columnar) é um formato de arquivo originalmente desenvolvido por engenheiros da Hortonworks e do Facebook. (Hortonworks desde então se fundiu com Cloudera.). ORC é muito semelhante ao Parquet. ORC e Parquet foram projetados para atender a muitas das mesmas necessidades e, internamente, eles usam muitas das mesmas técnicas para obter um excelente desempenho.

ORC é frequentemente usado com o Hive. Para aproveitar certos recursos do Hive, você deve armazenar dados de tabela no formato de arquivo ORC. (Esses recursos não são abordados neste curso ou em outros cursos desta especialização.) No entanto, o ORC não é amplamente suportado por outras ferramentas. O Impala não pode consultar tabelas cujos dados estão armazenados em arquivos ORC.

No geral, o ORC oferece excelente desempenho, mas interoperabilidade limitada. Recomendamos escolher ORC quando você estiver usando recursos do Hive que o exijam. Tenha em mente que as tabelas do Hive que usam arquivos ORC não podem ser consultadas usando o Impala.

4.3.5 Outros tipos de arquivo

Os sistemas de big data às vezes usam outros formatos de arquivo além do texto, Avro, Parquet e ORC. Duas opções comuns são SequenceFiles eRCFiles. Geralmente, não recomendamos o uso desses formatos de arquivo, mas eles são descritos brevemente abaixo para que você esteja ciente deles.

Arquivos de sequência

O formato SequenceFile foi desenvolvido para sistemas de big data como alternativa aos arquivos de texto. SequenceFiles armazena pares de valores-chave em um formato de contêiner binário. Eles armazenam dados com mais eficiência do que arquivos de texto e podem armazenar dados binários como imagens.

No entanto, o formato SequenceFile está intimamente associado à linguagem de programação Java e não é amplamente suportado fora do ecossistema Hadoop.

No geral, SequenceFiles oferece bom desempenho, mas interoperabilidade ruim.

Arquivos RC

RCFile, que significa Record Columnar File, é um formato de arquivo colunar que foi desenvolvido para uso com o Hive. O RCFile também é suportado por algumas outras ferramentas, incluindo Impala, mas esse suporte é limitado. O formato RCFile armazena todos os dados como strings, o que é ineficiente.

No geral, o RCFile oferece baixo desempenho e interoperabilidade limitada.

O formato de arquivo ORC (descrito na leitura anterior) é uma versão aprimorada do RCFile com desempenho superior.

4.4. Trabalhando com Tipos de Arquivos

Com tantos formatos de arquivo diferentes, você pode se perguntar qual deles funcionará melhor para seus dados e caso de uso. Há vários fatores a serem considerados ao escolher um formato de arquivo. Por exemplo, qual é o padrão de consumo de dados? Em outras palavras, como os dados são carregados? O conjunto de dados será carregado de uma só vez ou novos registros serão adicionados continuamente?

Se os dados forem carregados de uma só vez, um formato de arquivo colunar como Parquet pode ser capaz de tirar proveito de padrões repetidos nos dados, para armazená-los de forma mais eficiente. Mas se ele está sendo carregado em lotes muito pequenos, Parquet pode ser menos eficiente do que outros formatos de arquivo.

Outro fator é que ferramentas você usará para trabalhar com os dados. Isso define a quantidade de interoperabilidade que você precisará. Por exemplo, você estará usando MapReduce, Hive, Impala, Spark ou qualquer outra ferramenta? Quais formatos essas ferramentas suportam?

Em geral, arquivos de texto e arquivos Parquet oferecem a melhor interoperabilidade. Ao usar esses formatos de arquivo, você terá mais flexibilidade em como processar os dados. No entanto, se você estiver

usando apenas Hive, não Impala, então o formato de arquivo ORC pode ser uma escolha melhor porque existem alguns recursos do Hive que exigem que os dados estejam em arquivos ORC.

Outra questão é, qual é a vida útil esperada dos dados? É temporário, como uma entrada para um trabalho que será excluído após a conclusão do trabalho? Ou você precisará reter os dados e ser capaz de lê-los daqui a alguns anos? Se os dados estão sendo armazenados por um longo tempo, então você deve escolher um formato de arquivo com boa interoperabilidade. Porque daqui a alguns anos você pode não estar usando as mesmas ferramentas que você está usando hoje. Também para armazenamento de dados a longo prazo, considere escolher um formato de arquivo que armazene as informações do esquema nos arquivos de dados. Dessa forma, você sempre poderá determinar os nomes e tipos de dados das colunas nos arquivos de dados. E considere escolher um formato de arquivo que suporte a evolução do esquema para que você possa adicionar, remover ou modificar colunas em uma tabela Hive ou Impala sem a necessidade de fazer alterações nos próprios arquivos de dados.

Geralmente, Avro e Parquet são boas escolhas para armazenamento de dados a longo prazo. Um pouco relacionado é a questão, quais são seus requisitos para tamanho de dados e desempenho de consulta? Se você está planejando armazenar uma grande quantidade de dados e precisa correlacionar eficientemente, então você deve escolher um formato de arquivo compactado que seja otimizado para desempenho de consulta. Os arquivos de parquet são tipicamente uma boa escolha para isso. A linha inferior é que não havia nenhum formato de arquivo único que seja melhor em todos os casos. O melhor formato depende dos seus dados e do que você está fazendo com eles. Use essas considerações junto com a forma como cada formato funciona para informar sua decisão.

4.4.1 Criando tabelas com arquivos Avro e Parquet

Quando você cria tabelas usando arquivos nos formatos Apache Avro ou Apache Parquet, há opções adicionais específicas para esses formatos de arquivo.

Apache Avro

Para criar uma tabela Avro (ou seja, uma tabela que usará o formato Avro), especifique **`STORED AS AVRO`**.

O Avro incorpora uma definição de esquema (nomes de coluna e seus tipos de dados) no próprio arquivo, mas você ainda deve fornecer uma definição de esquema em sua instrução **`CREATE TABLE`**. Existem três formas de fazer isso:

1. A maneira usual de especificar colunas e seus tipos de dados
2. Especificando um arquivo de esquema Avro
3. Fornecendo o esquema como uma string literal JSON

Para criar uma tabela Avro da maneira usual, forneça o nome e o tipo de dados de cada coluna, entre parênteses, com os pares nome-tipo separados por vírgulas:

```
CREATE TABLE company_email_avro (id INT, nome STRING, e-mail STRING)
    STORED AS AVRO;
```

Você pode evitar listar o esquema completo na instrução **`CREATE TABLE`** armazenando o esquema de tabela em um arquivo de esquema Avro separado e vinculando-o como uma propriedade de tabela. Em sua instrução **`CREATE TABLE`**, especifique **`TBLPROPERTIES ('avro.schema.url'='/path/to/file.avsc');`** em que **`file.avsc`** é um arquivo JSON que contém um esquema Avro.

Por exemplo, você pode criar a mesma tabela acima usando **`order_details.avsc`** com este conteúdo:

```
{"type":"registro",
     "name":"company_email_avro",
     "fields":[
           {"name":"id","type":["null","int"]},
           {"name":"name","type":["null","string"]},
           {"name":"email","type":["null","string"]}
     ]}
```

(As instâncias de "**null**" que precedem os nomes dos tipos de dados neste esquema indicam que as colunas podem conter valores **NULL**.)

Então a instrução **CREATE TABLE** poderia ficar assim:

```
CREATE TABLE company_email_avro
    STORED AS AVRO
    TBLPROPERTIES('avro.schema.url'='/path/to/company_email.avsc');
```

O caminho da URL pode apontar para um arquivo de esquema Avro no HDFS, conforme mostrado acima. Você também pode apontar para outros locais, como um servidor HTTP usando o protocolo **http://** ou um bucket do S3 usando o protocolo apropriado para sua configuração, como **s3a://**.

Em vez de usar o arquivo de esquema Avro, você pode colocar o JSON do esquema na instrução **CREATE TABLE** como uma propriedade de tabela usando '**avro.schema.literal**'. O seguinte também criará a mesma tabela dos exemplos anteriores:

```
CREATE TABLE company_email_avro
    STORED AS AVRO
    TBLPROPERTIES ('avro.schema.literal'=
     '{"type":"record",
         "name":"company_email_avro",
          "fields":[
               {"name":"id","type":["null","int"]},
               {"name":"name","type":["null","string"]},
               {"name":"email","type":["null","string"]}
       ]}');
```

Se o esquema JSON de um arquivo de dados Avro não estiver disponível para você, você poderá extraí-lo do arquivo de dados Avro usando o utilitário de linha de comando **avro-tools**. A sintaxe é:

```
avro-tools getschema /path/to/avro_data_file.avro
```

Este comando extrai o esquema do arquivo de dados Avro especificado e imprime a representação JSON do esquema na tela. Você pode então copiar e colar esse JSON na cláusula **TBLPROPERTIES** conforme descrito acima ou salvar esse JSON em um arquivo de texto simples usando a extensão **.avsc** e apontar para esse arquivo na cláusula **TBLPROPERTIES**, conforme descrito anteriormente. Este utilitário **avro-tools** é instalado na VM do curso.

Apache Parquet

Assim como nas tabelas Avro, você pode criar tabelas Parquet especificando **STORED AS PARQUET**. Você viu isso na semana 2 lendo sobre a cláusula **STORED AS**. Os arquivos Parquet também têm o esquema embutido no arquivo, mas a definição da tabela também precisa incluir a definição do esquema, por exemplo:

```
CREATE TABLE investors_parquet
      (name STRING, amount INT, share DECIMAL(4,3))
   STORED AS PARQUET;
```

Com arquivos Parquet, no entanto, Impala (não Hive) fornece um atalho: **`LIKE PARQUET '/path/to/file.parq'`**. Ao usar esse atalho, você pode obter o esquema diretamente de um arquivo Parquet, sem a necessidade de um arquivo de esquema separado, pois o Avro exige:

```
CREATE TABLE investors_parquet
  LIKE PARQUET '/users/hive/warehouse/investors_parquet/investors.parq'
  STORED AS PARQUET;
```

Observe que, neste caso, a tabela que está sendo criada usará o arquivo especificado como um de seus arquivos de dados, pois a localização do arquivo da tabela será **`/user/hive/warehouse/investors_parquet`** e o arquivo Parquet referenciado está nessa localização. Isso não precisa ser o caso; você pode usar um arquivo que existe em outro lugar que não seja onde os arquivos da tabela serão armazenados.

Observe também que isso não é o mesmo que clonar uma tabela usando **`LIKE tablename`**. (Consulte a leitura "**`CREATE TABLE Shortcuts`**" da Semana 2.) Usar **`LIKE tablename`** aponta para uma tabela existente e copia as informações de esquema dessa tabela do metastore; usando **`LIKE PARQUET '/path/to/file.parq'`** aponta para o caminho de um arquivo Parquet específico e copia as informações do esquema desse arquivo. Você pode usar o arquivo Parquet para uma tabela existente, mas nesse caso, provavelmente é melhor usar **`LIKE tablename`**.

A documentação do Cloudera para **`CREATE TABLE`** inclui as seguintes notas sobre o uso de **`LIKE PARQUET`**:

- Cada coluna na nova tabela tem um comentário informando que o tipo de coluna SQL foi inferido de um arquivo Parquet.
- O formato de arquivo da nova tabela é padronizado como texto, como acontece com outros tipos de instruções **`CREATE TABLE`**. Para que a nova tabela também use o formato Parquet, inclua a cláusula **`STORED AS PARQUET`** na instrução **`CREATE TABLE LIKE PARQUET`**.
- Se o arquivo de dados Parquet vier de uma tabela Impala existente, atualmente, quaisquer colunas **`TINYINT`** ou **`SMALLINT`** são transformadas em colunas **`INT`** na nova tabela. Internamente, o Parquet armazena esses valores como inteiros de 32 bits.

Experimente!

Tente criar as tabelas Avro de todas as três maneiras usando os comandos a seguir, descartando a tabela a cada vez antes de recriá-la de uma maneira diferente. **(Certifique-se de usar a palavra-chave `EXTERNAL` para não excluir o arquivo ao descartar a tabela.)**

1. Use da maneira usual:

```
CREATE EXTERNAL TABLE company_email_avro
       (id INT, name STRING, email STRING)
    STORED AS AVRO
    LOCATION 's3a://training-coursera2/company_email_avro/';
```

Use o comando **`DESCRIBE`** para inspecionar o esquema e, em seguida, verifique os dados de amostra usando seu método favorito para fazer isso (como o **`Sample`** do painel de origem de dados ou executando uma consulta **`SELECT *`**). Apague a tabela quando terminar.

2. Use o arquivo de esquema JSON em **`s3a://training-coursera2/company_email_avro.avsc`**. Primeiro, dê uma olhada no conteúdo desse arquivo (usando um comando **`hdfs dfs -cat`** ou **`aws s3 cp`** para conteúdo S3 - você não pode usar Hue para isso) e execute este comando:

```
CREATE EXTERNAL TABLE company_email_avro
```

```
    STORED AS AVRO
    LOCATION 's3a://training-coursera2/company_email_avro/'
    TBLPROPERTIES ('avro.schema.url'='
            s3a://training-coursera2/company_email_avro.avsc');
```

Use o comando **DESCRIBE** para inspecionar o esquema e, em seguida, verifique os dados de amostra usando seu método favorito para fazer isso (como o **Sample** do painel de origem de dados ou executando uma consulta **SELECT ***). Apague a tabela quando terminar.

3. Por fim, use o literal JSON:

```
CREATE EXTERNAL TABLE company_email_avro
    STORED AS AVRO
    LOCATION 's3a://training-coursera2/company_email_avro/'
    TBLPROPERTIES ('avro.schema.literal'=
       '{"type":"record",
           "name":"company_email_avro",
            "fields":[
               {"name":"id","type":["null","int"]},
               {"name":"name","type":["null","string"]},
               {"name":"email","type":["null","string"]}
            ]}');
```

Use o comando **DESCRIBE** para inspecionar o esquema e, em seguida, verifique os dados de amostra usando seu método favorito para fazer isso (como Amostra do painel de origem de dados ou executando uma consulta **SELECT ***). Apague a tabela quando terminar.

5. Gerenciando Datasets em Clusteres e Cloud Storage

Este capítulo é sobre o gerenciamento de datasets. Lembre-se de que nos clusters baseados no Hadoop são plataformas, os dados da tabela são armazenados separadamente dos metadados da tabela. Os metadados de uma tabela, que inclui a estrutura ou esquema de tabelas, são armazenados no metastore. Você pode criar e gerenciar a parte de metadados de uma tabela usando instruções **SQL** como **CREATE TABLE** e **ALTER TABLE**. Os dados de uma tabela são armazenados separadamente em um sistema de arquivos como HDFS ou S3.

Na maioria dos exemplos passados, os dados já foram carregados no sistema de arquivos para você e a tarefa em questão era definir os metadados para que você pudesse consultar esses arquivos de dados com Hive e Impala. Mas no mundo real, os dados nem sempre serão pré-carregados para você. Então, você aprenderá a carregar dados no HDFS e no S3.

Há mais de uma maneira de fazer isso. Assim, o conteúdo está dividido em quatro lições principais. Primeiro, vamos mostrar como carregar arquivos de dados no HDFS, o Hadoop Distributed File System. HDSF é onde Hive e Impala armazenam dados de tabela por padrão. Você pode carregar dados no HDFS usando o Hue ou usando a linha de comando. Vamos mostrar-lhe como fazer as duas coisas.

Em seguida, mostraremos como carregar arquivos de dados no Amazon S3, a plataforma de armazenamento em nuvem mais popular. Você também pode fazer isso usando o Hue, mas vamos nos concentrar na linha de comando. Mostraremos duas maneiras diferentes de fazê-lo a partir da linha de comando. Às vezes, os dados de origem que você deseja carregar não estão em arquivos, mas sim em uma tabela em um sistema de banco de dados relacional como MySQL ou PostgreSQL.

Existe uma ferramenta chamada Sqoop. Eles podem mover dados entre sistemas de banco de dados relacional e HDFS ou S3, vamos mostrar-lhe como usar isso. Finalmente, vamos mostrar como o Hive e o Impala podem ser usados para mover dados para tabelas executando comandos SQL. Apresentaremos várias instruções SQL que você pode usar para fazer isso e descreveremos como Hive e Impala normalmente exigem uma abordagem diferente para carregar dados do que sistemas de banco de dados relacionais.

Atualizando o Cache de Metadata do Impala após Carregar Dados

Lembre-se de que o rápido desempenho de consulta do Impala vem em parte do uso de um cache de metadados na memória do computador. Portanto, ele não precisa ir para o Metastore para procurar os metadados de uma tabela toda vez que uma consulta é executada.

Impala atualiza automaticamente ou atualiza seu cache de metadados quando você faz alterações com Impala. Mas você tem que dizer a ele para atualizar os metadados quando você faz alterações fora do Impala. Por exemplo, se você quiser alterar a ordem das colunas usando uma instrução Alter Table com após ou primeiro, você tem que fazer isso usando o Hive porque Impala não suporta estes após a nossa primeira palavra-chave.

Se você tentar imediatamente subconsulta a tabela no Impala, ele ainda usará o esquema antigo. Então, neste caso, você precisa dizer ao Impala para atualizar seu cache de metadados. Para fazer isso, execute um comando atualizado com Impala especificando o nome da tabela que você alterou. Em seguida, o cache de metadados do Impala será atualizado e ele usará o novo esquema de tabela. Agora, nos materiais desta semana, você não vai mudar as informações da coluna de uma tabela. Você estará carregando dados para as tabelas, o que envolve trabalhar com os arquivos que contêm os dados.

O cache de metadados do Impala também armazena informações sobre esses arquivos, como onde eles estão localizados e quantos eles estão. Existem diferentes maneiras de carregar os dados em um diretório de

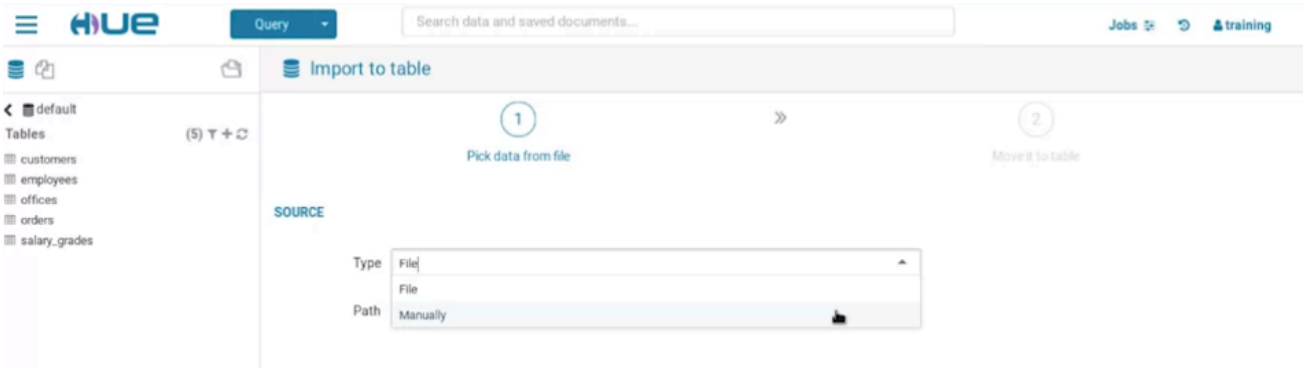
tabela e você não precisa usar Impala para fazê-lo. Se você carregar dados em um diretório de tabela de fora do Impala, Impala não estará ciente desses novos dados. Você pode consultar uma tabela e obter um resultado sem dados ou com dados incompletos. Para evitar isso, você precisa executar esse mesmo comando atualizado no Impala, sempre que você carregar novos dados em uma tabela de fora do Impala. Dessa forma, o cache de metadados do Impala saberá que os novos dados estão lá.

5.1. Carregando Arquivos no HDFS

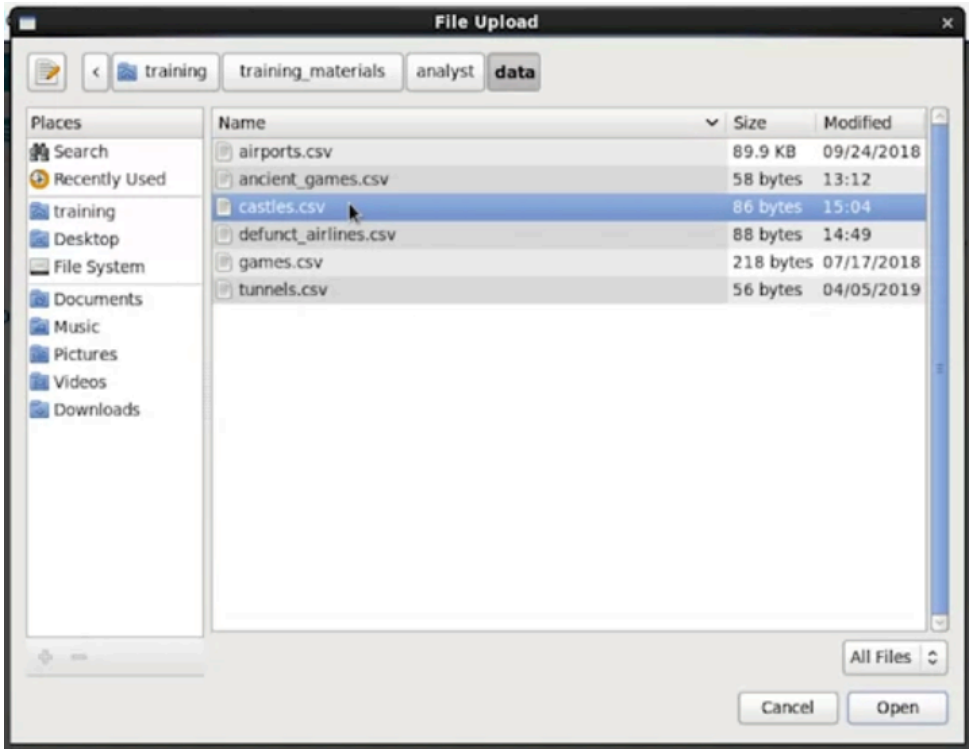
Carregando arquivos no HDFS com o Table Browser do Hue

O Hue inclui um navegador de tabelas (Table Browser) que você pode usar para navegar nas tabelas definidas no Metastore, e um navegador de arquivos que você pode usar para navegar nos diretórios e arquivos no HDFS. Você também pode usar essas duas interfaces para carregar arquivos de dados em um diretório de armazenamento de tabela no HDFS.

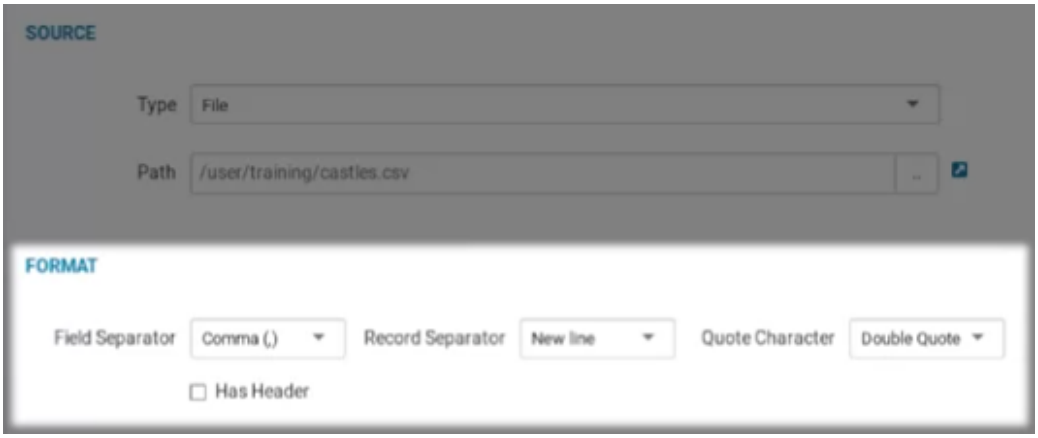
Para acessar o navegador de tabelas, clique no ícone de menu no canto superior esquerdo e, em seguida, em navegadores, clique em tabelas. Lembre-se de que o navegador de tabela permite que você crie uma nova tabela, você iniciar este processo clicando neste ícone de adição. Neste menu suspenso de tipo, se você selecionar a opção manualmente, então, na próxima etapa, o Hue oferece a opção de criar uma tabela vazia sem dados ainda ou de criar uma tabela gerenciada externamente para consultar alguns dados existentes que já estão no HDFS. Nesses casos, nenhum dado é carregado no HDFS.



Mas se você selecionar a opção de arquivo, o Hue criará uma tabela e moverá os dados para ela em uma única operação. Quando clicar neste seletor de caminho, o Hue abre esta escolha uma caixa de diálogo de arquivo. Aqui você tem duas opções. Um, você pode escolher um arquivo existente de algum lugar no HDFS. Se você fizer isso, você moverá esse arquivo para fora de seu local atual e para o diretório de armazenamento desta nova tabela. Dois, você pode fazer upload de um arquivo para este novo diretório de armazenamento de tabela a partir do seu sistema de arquivos local. Opção de upload: clique em carregar um arquivo. No meu sistema de arquivos local, navegue para **training/training_materiais/analyst/data**. Aqui selecionar o arquivo castles.csv e clicar em abrir.

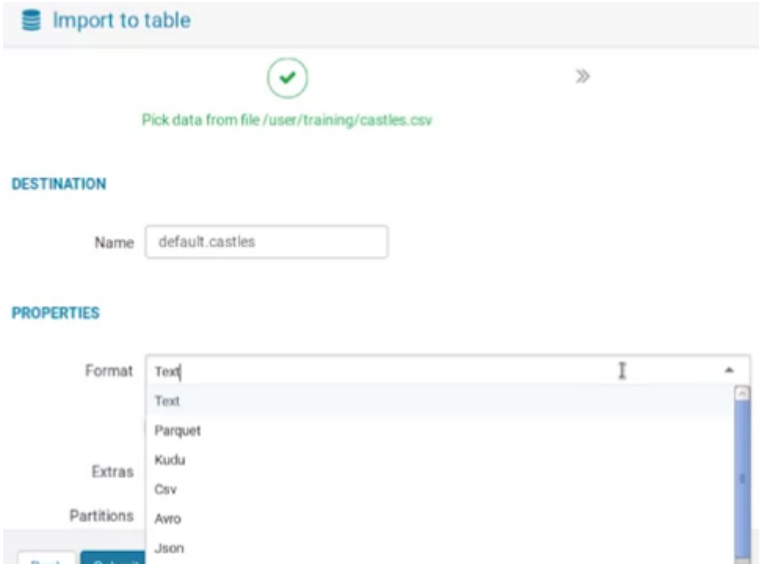


Hue, em seguida, carregue este arquivo para um local temporário no HDFS. Na seção de formato abaixo, você pode especificar o separador de campos. Em outras palavras, o delimitador. O separador de registros, esse é o caracter que marca o fim de um registro, e para o caracter de aspas. É comum a necessidade de mudar o separador de campo.



Para os outros dois, você geralmente pode manter os valores padrão. Este arquivo castles.csv tem uma linha de cabeçalho dando os nomes das colunas. Então eu vou marcar esta caixa de seleção tem cabeçalho. Você pode ver uma visualização dos dados como eles aparecerão quando você consultar esta tabela. Você pode usar essa visualização para confirmar se as configurações na seção de formato estão corretas. Parece correto, então eu vou clicar em **Next**.

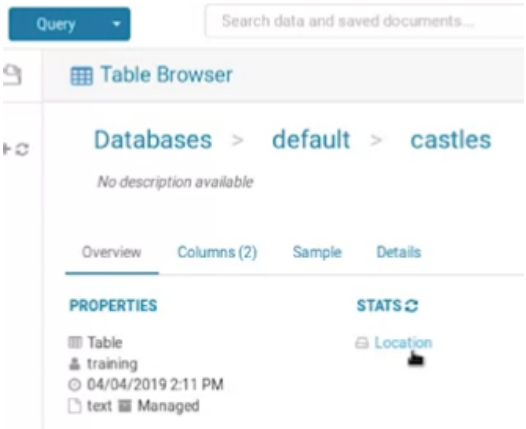
Nesta etapa, posso dar um nome à tabela e especificar em que banco de dados ela deve estar. Este campo de nome já está pré-preenchido com **default.castles**, o que significa que a tabela será nomeada castels e estará no banco de dados padrão. Vou manter como está. Em seguida, posso especificar mais algumas propriedades. Se os dados estivessem em algum outro formato como Parquet ou Avro, eu poderia usar esse formato para soltar



para especificar isso. Vou mantê-lo ajustado para texto. Vou manter esta caixa de seleção marcada. Dessa forma, este diretório de armazenamento de tabela será um subdiretório chamado **castles** no diretório do warehouse do Hive: **/user/hive/warehouse/castles**.

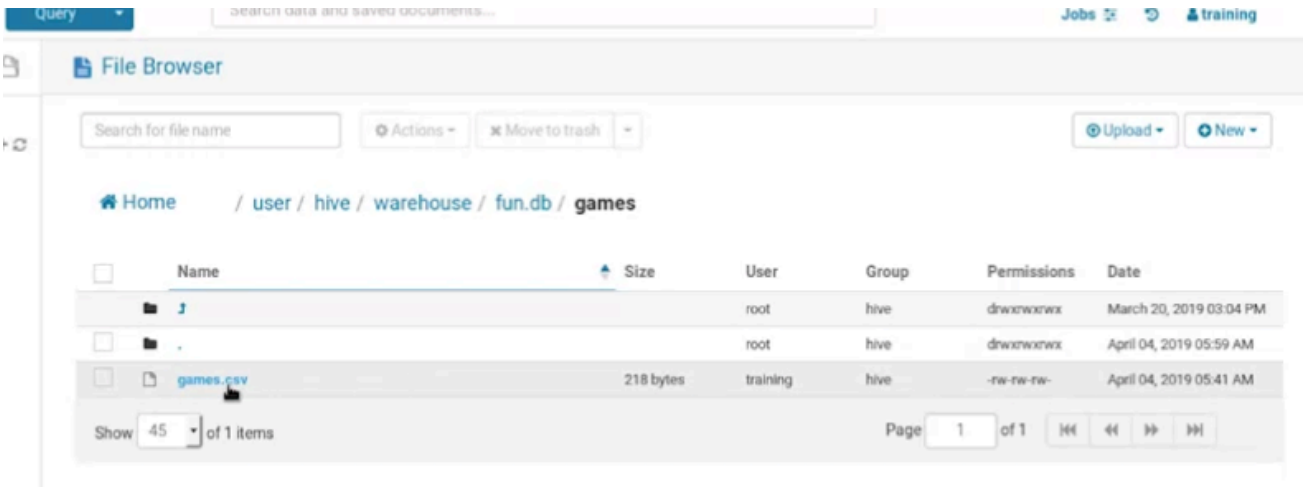
Os dados carregados irão para esse diretório no HDFS. Deixarei essas outras propriedades intocadas. Na seção de campos abaixo, você pode especificar o nome e o tipo de dados para cada coluna nesta tabela. Esses campos são pré-preenchidos usando os nomes da linha de cabeçalho no arquivo que enviei, e as melhores suposições do Hue sobre os tipos de dados das colunas. Você deve dar uma olhada nisso cuidadosamente para se certificar de que Hue adivinhou corretamente. Leve em consideração o que você sabe sobre os dados e o que você aprendeu sobre os diferentes tipos de dados. Neste caso, parece ok. Então eu vou clicar em **Submit**.

Em seguida, o Hue cria uma entrada para esta tabela no Metastore, cria o diretório de armazenamento para a tabela no HDFS e move o arquivo que eu enviei para este diretório de armazenamento. Você pode ver esse arquivo no diretório de armazenamento clicando neste link de localização. Aqui no diretório HDFS, **/users/hive/warehouses/castles** é o arquivo carregado.



No editor de consultas, posso atualizar o cache de metadados do Impala. Neste caso, eu preciso usar castelos de metadados invalidados em vez de castelos renovados porque castelos é uma nova tabela. Em seguida, posso consultar esta tabela. Assim, você pode usar o navegador de tabelas no Hue para carregar arquivos de dados no diretório de armazenamento HDFS de uma nova tabela no momento em que você cria a nova tabela.

Você também pode usar o navegador de tabelas para carregar arquivos de dados no diretório de armazenamento de uma tabela existente. Vou demonstrar isso agora. No navegador de tabelas, navegarei até uma tabela existente. Vou para o banco de dados **fun**, e aí, a tabela de games. Esta tabela tem cinco linhas descrevendo cinco jogos de tabuleiro populares. Vou clicar no link de localização para navegar no diretório de armazenamento desta tabela no HDFS.

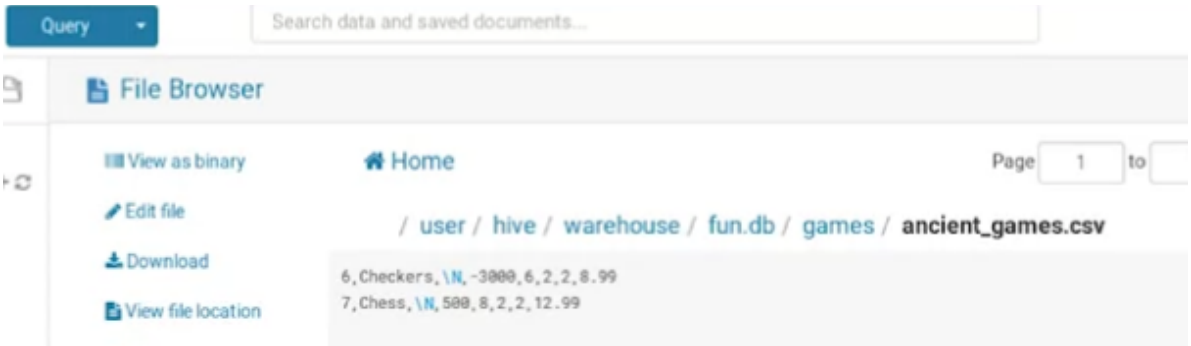


Você pode ver que há apenas um arquivo lá, games.csv contendo os dados para esses cinco jogos. Vou clicar no botão Voltar várias vezes para retornar ao navegador de tabelas. Há outro arquivo no sistema de arquivos local na VM chamado ancient_games.csv. Gostaria de adicionar os dados desse arquivo a esta tabela sem perder os dados que já estão nessa tabela.

Para fazer isso, vou clicar neste botão de importação de dados. O Hue abre esta caixa de diálogo de importação de dados. Aqui, como no exemplo anterior, você tem duas opções. Primeiro, você pode escolher um arquivo ou diretório existente de algum lugar no HDFS. Se você fizer isso, você moverá o arquivo ou diretório para fora de seu local atual e para o diretório de armazenamento desta tabela.

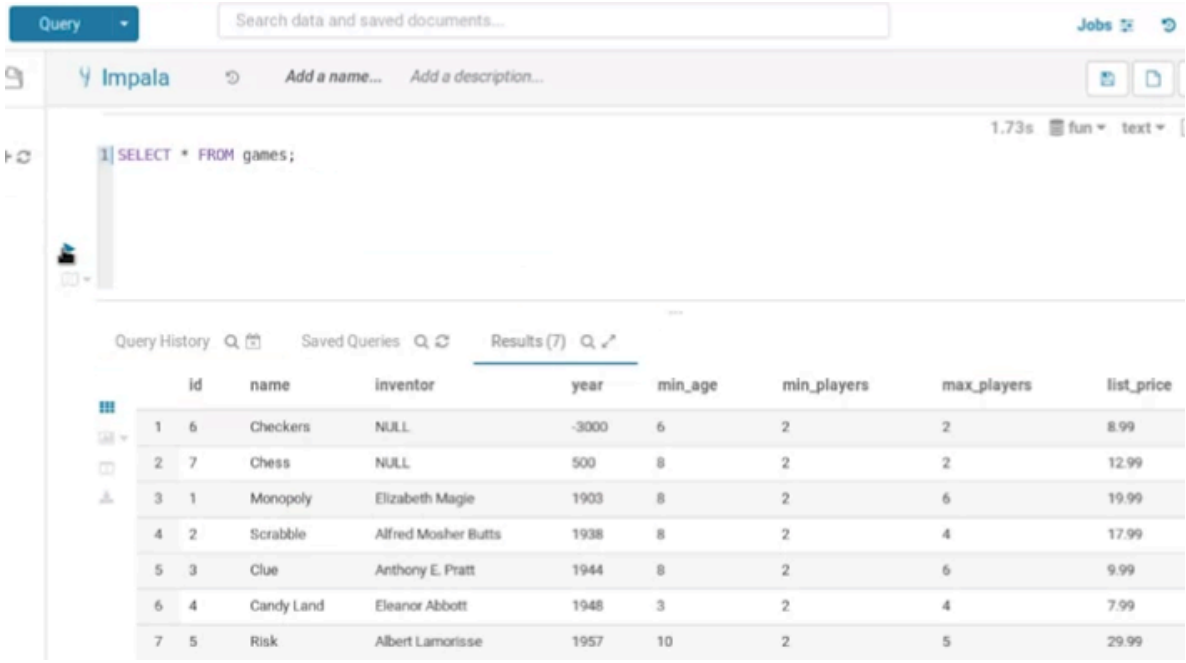
Dois, você pode fazer upload de um arquivo para este diretório de armazenamento de tabela a partir do seu sistema de arquivos local. Vou demonstrar novamente essa opção de upload. Desta vez, vou fazer o upload do arquivo **ancient_games.csv**. Depois de carregar este arquivo, vou selecioná-lo aqui. Não vou marcar esta caixa de seleção sobrescrever dados existentes. Marcar essa caixa faria com que o Hue removesse tudo neste diretório de tabela excluindo o arquivo existente games.csv, e isso não é o que eu quero fazer neste caso.

Vou clicar em enviar e aguardar a conclusão da tarefa. Quando terminar, clicarei no link de localização para navegar novamente para este diretório de armazenamento de tabela no HDFS. Agora, você pode ver que há dois arquivos lá; games.csv que estava lá antes, e ancient_games.csv que é novo. Este arquivo ancient_games.csv contém dois registros, representando **Checkers** e **Chess**. As colunas são exatamente as mesmas que no arquivo games.csv, e observe que há duas instâncias de **\N** indicando valores ausentes na terceira coluna.



No editor de consultas, posso selecionar o banco de dados financiado e, em seguida, atualizar os metadados do Impala para a tabela do jogo. O comando de atualização é suficiente neste caso, porque a tabela do games não é uma nova tabela, mas há novos dados nela. Portanto, este comando fará com que Impala perceba que novos dados.

Em seguida, posso consultar esta tabela. Nos resultados da consulta, você pode ver duas novas linhas representando **Checkers** e **Chess** junto com as cinco linhas antigas. Observe os valores nulos na terceira coluna, a coluna *inventor*, para **Checkers** e **Chess**. Ninguém sabe quem inventou esses jogos. Portanto, usando o navegador de tabelas do Hue, você pode carregar arquivos de dados no diretório de armazenamento para uma nova tabela ou uma tabela existente.



	id	name	inventor	year	min_age	min_players	max_players	list_price
1	6	Checkers	NULL	-3000	6	2	2	8.99
2	7	Chess	NULL	500	8	2	2	12.99
3	1	Monopoly	Elizabeth Magie	1903	8	2	6	19.99
4	2	Scrabble	Alfred Mosher Butts	1938	8	2	4	17.99
5	3	Clue	Anthony E. Pratt	1944	8	2	6	9.99
6	4	Candy Land	Eleanor Abbott	1948	3	2	4	7.99
7	5	Risk	Albert Lamorisse	1957	10	2	5	29.99

5.1.1 Mais sobre os comandos do shell do HDFS

Esta leitura fornece algumas informações adicionais sobre o HDFS e o Hadoop File System Shell que você deve conhecer.

O comando -mkdir

Um comando útil que não foi mencionado no vídeo anterior é o **`hdfs dfs -mkdir`**, que pode ser usado para criar um ou mais diretórios no HDFS. Este comando espera que o diretório pai do diretório que você está criando já exista, portanto, se você quiser usar este comando para criar diretórios aninhados, comece primeiro criando o diretório pai de nível mais alto, depois crie o subdiretório filho do próximo nível e em breve. Alternativamente, você pode usar a opção **-p** (abreviação de *parent* [pai]) para criar automaticamente quaisquer diretórios pai necessários do diretório especificado se eles ainda não existirem.

5.1.2 Mais opções de comando HDFS

Alguns dos comandos que você aprendeu neste curso podem ter opções de linha de comando adicionais que não foram descritas. Por exemplo, ao usar o comando quando usar **`hdfs dfs -rm`**, você pode especificar a opção **-r** (abreviação de recursivo) para excluir o diretório especificado e todos os arquivos e subdiretórios nele. A sintaxe é:

```
hdfs dfs -rm -r /caminho/para/diretório/
```

Seja extremamente cuidadoso ao usar esta opção -r! É fácil excluir inadvertidamente uma grande quantidade de dados ao especificar incorretamente o caminho do diretório.

Lista completa de comandos, opções e argumentos do shell do HDFS

Você pode ver uma lista completa dos comandos Hadoop File System Shell e as opções e argumentos que eles aceitam nesta página da web: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/FileSystemShell.html. Muitos deles você nunca precisará usar, mas este é um bom recurso para encontrar maneiras de fazer as coisas que você precisa. Ao examinar essa página, lembre-se de que **`hdfs dfs`** é o mesmo que **`hadoop fs`** - você pode usar qualquer um.

Alguns dos comandos nessa página estão relacionados a permissões e propriedade de arquivos, o que está além do escopo deste curso, mas consulte a leitura "Permissões do HDFS" se estiver interessado em saber mais sobre isso.

HDFS Paths

Neste curso, a maioria dos caminhos HDFS que você verá nos comandos hdfs dfs estão no formato **`/path/to/directory/`** (para um diretório) ou **`/path/to/directory/file.ext`** (para um arquivo). Isso é conhecido como um caminho não qualificado porque não especifica qual protocolo usar e não especifica qual instância específica do HDFS usar. Na maioria dos casos, é suficiente usar caminhos HDFS não qualificados, porque a configuração do seu ambiente de big data especificará qual protocolo e qual instância HDFS usar.

No entanto, em alguns casos, dependendo da configuração de seu ambiente de big data, pode ser necessário qualificar totalmente seus caminhos HDFS especificando o protocolo **`hdfs://`** e um nome de host indicando qual instância de HDFS usar. Para fazer isso, use o formulário: **`hdfs://hostname/path/to/directory/file.ext`** ou **`hdfs://hostname.domain/path/to/directory/file.ext`**. Pergunte ao administrador do sistema qual hostname e qual domínio (se houver) usar.

Você também pode especificar o protocolo **`hdfs://`** sem especificar um nome de host, usando o formulário: **`hdfs:///path/to/directory/file.ext`**. Observe as três barras após **`hdfs:`** As duas primeiras barras fazem parte do protocolo e a terceira barra é o início do caminho.

HDFS Trash

Observe que o HDFS possui um diretório "lixo" para arquivos excluídos recentemente, semelhante ao Lixo nos sistemas iOS ou Windows. Esta lixeira nem sempre está habilitada, então você deve verificar seu sistema para ver se está habilitado antes de assumir que você pode recuperar qualquer arquivo deletado!

O comando **`hdfs dfs -rm`** tem uma opção **`-skipTrash`** que você pode usar para ignorar a lixeira (se estiver habilitada) e excluir o arquivo imediatamente. Ao excluir arquivos grandes para liberar espaço no HDFS, considere usar essa opção para não precisar executar a etapa adicional de esvaziar a lixeira. Mas lembre-se que ao usar esta opção, os arquivos que você excluir não serão recuperáveis.

5.1.3 Encadeamento e script com comandos HDFS

Quando você trabalha muito com comandos HDFS, há duas técnicas que você provavelmente achará muito úteis: encadeamento e script.

Encadeando Comandos HDFS

Se você trabalhou muito com a linha de comando, pode estar familiarizado com o uso de pipes (|) para encadear comandos ou usar o redirecionamento para enviar resultados para um local diferente (como para um arquivo em vez de para a tela). Você também pode fazer isso com comandos de shell HDFS.

Você pode realizar muitas tarefas práticas combinando comandos de shell HDFS com canalização ou redirecionamento. Dois exemplos são descritos abaixo.

Visualizando as primeiras linhas de um arquivo

Para visualizar apenas as primeiras linhas de um arquivo de texto armazenado no HDFS, você pode canalizar a saída do comando **`hdfs dfs -cat`** para o comando **`head`**:

```
$ hdfs dfs -cat /caminho/para/arquivo.txt | head
```

Você pode ignorar a mensagem que diz "Não é possível gravar no fluxo de saída". Isso acontece porque o comando **`hdfs dfs -cat`** gera mais dados do que as entradas do comando **`head`**.

Carregando dados sem a linha de cabeçalho

Em vez de usar a propriedade da tabela **`skip.header.line.count`** para ignorar a linha de cabeçalho em arquivos de texto, você pode copiar tudo, exceto a linha de cabeçalho, ao colocar um arquivo no HDFS:

```
$ tail -n +2 arquivo_fonte.txt | hdfs dfs -put - /path/to/destination_file.txt
```

No comando acima, o caractere de hífen ou traço (-) após **`-put`** informa ao aplicativo de HDFS shell para obter a saída do comando **`tail`** antes do pipe e carregá-la no HDFS.

Para remover a linha de cabeçalho ao copiar um arquivo de texto na direção oposta (do HDFS para o sistema de arquivos local), você deve executar um comando diferente, desta vez usando o operador de redirecionamento de saída (>) para armazenar a saída do comando **`tail`** em um Arquivo:

```
$ hdfs dfs -cat /path/to/source_file.txt | tail -n +2 > destination_file.txt
```

Usando comandos em scripts

Você também pode usar comandos como este em scripts. Isso é particularmente útil quando você automatiza tarefas que envolvem o gerenciamento de arquivos no HDFS. Em scripts de shell, você pode usar comandos **`hdfs dfs`** exatamente como faria normalmente com outros comandos de shell. Scripts para outras linguagens, como Python e R, normalmente podem invocar comandos shell usando comandos específicos para a linguagem. Por exemplo, em Python você pode usar os módulos **`os`** ou **`subprocess`** e usar uma chamada como **`subprocess.call()`**. Em **R**, você pode usar **`system()`**.

5.1.4 Permissões HDFS

A segurança é uma consideração importante para qualquer sistema de computador, e os sistemas de big data não são exceção. Na VM deste curso, você tem todas as permissões necessárias para fazer os exercícios e exemplos deste curso, mas na sua empresa talvez você não consiga fazer tudo o que está aprendendo aqui. Por exemplo, a capacidade de criar bancos de dados, ou mesmo tabelas, geralmente é restrita para que o sistema de

uma empresa não fique sobrecarregado com itens redundantes ou desnecessários. Da mesma forma, a capacidade de descartar tabelas ou excluir ou mover arquivos pode ser restrita. Empresas maiores só permitirão que seus engenheiros de dados, administradores de banco de dados ou administradores de sistema façam esse trabalho.

Uma análise profunda dos problemas de permissão está além do escopo deste curso, mas os seguintes recursos podem fornecer informações adicionais, se você estiver interessado:

- Hadoop Security – Hadoop HDFS File Permissions do DWGeek é uma visão geral rápida das permissões de arquivo. Este é um bom lugar para começar, e então você pode decidir se quer ir mais fundo.
- Comandos HDFS, Permissões HDFS e Armazenamento HDFS é um capítulo de um livro, Expert Hadoop Administration: Managing, Tuning, and Securing Spark, YARN, and HDFS. A seção "Usuários e superusuários do HDFS" pode ser útil se você não estiver familiarizado com o conceito de superusuário.
- A documentação do Hadoop inclui o Guia de Permissões do HDFS, que provavelmente tem mais detalhes do que você gostaria, mas é um ótimo recurso se houver uma pergunta específica que você deseja responder.

5.2. Carregando Dados com Armazenamento em Nuvem

5.2.1 Outras Formas de Carregar Arquivos na S3

Existem outras maneiras de se conectar ao S3 e carregar dados, além do que mencionamos até agora. O método que você usar dependerá da política de sua empresa e de suas próprias preferências. Sinta-se à vontade para ler mais sobre os exemplos fornecidos através dos links aqui:

- Outras interfaces de linha de comando (CLIs), como s3cmd (consulte https://s3tools.org/s3cmd)
- Interfaces gráficas de usuário (GUIs), como CloudBerry Explorer ou CyberDuck
- Console de gerenciamento da AWS, uma interface direta da Amazon Web Services

5.2.2 Permissões S3

Assim como no HDFS, o gerenciamento de permissões é vital com o S3 — ainda mais porque, como um serviço de nuvem, o S3 pode, em muitos casos, ser acessado com mais facilidade por hackers e outros agentes mal-intencionados. Há muitas histórias sobre buckets do S3 sendo tornados públicos e tendo consequências indesejadas:

- Há um buraco em 1.951 buckets do Amazon S3
- Verizon e Viacom
- Não entre em pânico, Chicago, mas um erro de configuração do AWS S3 expôs 1,8 milhão de registros de eleitores

Vazamento em Bucket S3 atinge milhares de pessoas com autorização de segurança dos EUA

Se você usa o S3, certifique-se de entender os problemas de permissões e o que é permitido. A AWS fornece um guia sobre como definir permissões de acesso a buckets e objetos.

5.2.3 Valores ausentes

Ao trabalhar com colunas de cadeia de caracteres, é importante lembrar que uma cadeia vazia, também chamada de cadeia de comprimento zero, não é a mesma coisa que um valor ausente (**NULL**).

No entanto, quando os dados são armazenados em arquivos de texto delimitados por processos externos ao Hive e Impala, é comum que os valores ausentes sejam representados como *strings vazias*. Por exemplo, sua organização pode ter um processo de coleta de dados no qual os nomes e sobrenomes são armazenados em um arquivo CSV e alguns indivíduos podem deixar seus sobrenomes em branco. Portanto, as linhas do arquivo CSV resultante podem ficar assim: **`fname`**,**`lname`**

```
Bert,
Big,Bird
Count,von Count
Ernie,
Two-Headed,Monster
```

Observe as vírgulas à direita nas linhas em que os indivíduos não forneceram um sobrenome.

Se você carregar esse arquivo CSV no diretório de armazenamento de uma tabela Hive e Impala, os valores ausentes na coluna **`lname`** serão interpretados como strings vazias (`''`), não como valores ausentes (**`NULL`**).

Isso pode causar alguma confusão. Por exemplo, um analista pode usar Hive ou Impala para executar uma consulta nesta tabela, como

```
SELECT * FROM nomes WHERE lname IS NOT NULL;
```

Essa consulta retornaria *todas* as linhas. Isso pode desafiar a expectativa do analista de que retornaria apenas as linhas para Big Bird e Count von Count (porque as outras linhas parecem não ter sobrenomes). Mas o Hive e o Impala não tratam essas strings vazias como **`NULL`** por padrão.

No entanto, há uma propriedade de tabela que você pode definir para fazer com que Hive e Impala interpretem strings vazias em arquivos de dados de tabela como valores ausentes: a propriedade **`serialization.null.format`**.

Você pode definir essa propriedade em um **`CREATE TABLE`** usando esta cláusula **`TBLPROPERTIES`**:

```
TBLPROPERTIES ('serialization.null.format' = '')
```

Observe como o valor dessa propriedade é definido como `''` (uma string vazia ou de comprimento zero).

Você também pode definir essa propriedade em uma tabela existente usando uma instrução **`ALTER TABLE`**. Por exemplo:

```
ALTER TABLE names SET TBLPROPERTIES ('serialization.null.format' = '');
```

Se você definir essa propriedade de tabela como uma string vazia (''), as consultas Hive e Impala nessa tabela tratarão strings vazias em colunas de string de caracteres como se fossem valores ausentes verdadeiros (**`NULL`**).

O valor padrão deste **`serialization.null.format`** é **`'\N'`**. Esta é uma cadeia literal de dois caracteres que consiste em uma barra invertida seguida pela letra maiúscula N; não é um caractere especial.A propriedade **`serialization.null.format`** também determina como os valores **`NULL`** são representados nos arquivos de dados da tabela quando os dados são inseridos na tabela usando Hive ou Impala. Afeta como os valores ausentes são armazenados para colunas de todos os tipos de dados (não apenas tipos de cadeia de caracteres).

Para uma tabela específica, você deve decidir se os valores ausentes devem ser representados como **`\N`** ou como uma string vazia nos arquivos dessa tabela. Você nunca deve misturar representações diferentes nos arquivos para uma única tabela. Como prática recomendada, recomendamos o processamento de dados baseados em texto para garantir que os valores que você deseja interpretar como **`NULL`** pelo Hive e Impala sejam

representados como **\N** nos arquivos de dados antes de armazenar os arquivos no diretório da tabela. Mas em situações em que isso não é prático, o método descrito acima é uma solução eficaz.

Experimente!

As informações acima podem fazer mais sentido quando você as vê em ação, portanto, tente as etapas a seguir usando o Hue.

1. Para ver um exemplo de arquivo de dados que contém esse valor \N, observe o conteúdo do arquivo **/user/hive/warehouse/offices/offices.txt** no HDFS. Este é o arquivo que contém os dados da tabela de **offices** no banco de dados padrão. Observe o **\N** na quarta linha e terceira coluna deste arquivo, que representa um valor ausente de **state_province** para o escritório localizado em Cingapura (pesquise por Singapore). Consulte a tabela de **offices** com Hive ou Impala e observe que esse valor aparece como **NULL** no resultado da consulta.

Para o restante desta seção, você criará uma tabela e verá o efeito de definir a propriedade **serialization.null.format**.

2. Usando Hive ou Impala, crie uma tabela chamada **names** executando a seguinte instrução **CREATE TABLE**:

```
CREATE TABLE nomes (fname STRING, lname STRING)
    ROW FORMAT DELIMITED
       FIELDS TERMINATED BY ',';
```

3. No diretório da tabela no HDFS (**/user/hive/warehouse/names/**), crie um arquivo de texto e armazene as cinco linhas a seguir nele:

```
Bert,
Big,Bird
Count,von Count
Ernie,
Two-Headed,Monster
```

Você pode fazer isso com o navegador de arquivos do Hue ou com um comando **hdfs dfs**.

4. Se você estiver usando o Impala, execute o seguinte comando para forçar o Impala a atualizar seus metadados de arquivo para esta tabela:

```
REFRESH names;
```

5. Execute a seguinte consulta SQL nesta tabela:

```
SELECT * FROM names WHERE lname IS NOT NULL;
```

Observe como o conjunto de resultados inclui *todas* as linhas, incluindo as linhas nas quais **lname** está vazio.

6. Agora execute a seguinte instrução para dizer ao Hive e ao Impala para tratar strings vazias como valores ausentes para esta tabela:

```
ALTER TABLE names SET TBLPROPERTIES ('serialization.null.format' = '');
```

Se você estiver usando o Impala, atualize os metadados desta tabela novamente.

7. Execute a consulta SQL novamente para mostrar as linhas para as quais **lname IS NOT NULL**. Desta vez, observe como as linhas com sobrenomes vazios *não* são retornadas no conjunto de resultados.

8. Finalmente, elimine esta tabela **names**, porque você não precisará dela novamente.

5.2.4 Conjuntos de caracteres

Como você deve saber, os sistemas de computador usam conjuntos de caracteres definidos, que são coleções de caracteres, para quais são os caracteres permitidos. Exemplos incluem Unicode, ASCII, Extended ASCII e vários conjuntos de ISO.

A compatibilidade entre os conjuntos de caracteres que seus arquivos de dados usam e o conjunto de caracteres que seu sistema usa é uma consideração importante. Caracteres incompatíveis podem aparecer de forma estranha ou podem até fazer com que o sistema gere erros na leitura.

Se você estiver familiarizado com SQL em RDBMSs, talvez tenha visto a cláusula **`CHARACTER SET`** na instrução **`LOAD DATA`**, que permite especificar o conjunto de caracteres ao carregar os dados. Hive e Impala não têm essa cláusula em suas instruções **`LOAD DATA`**.

A documentação da Cloudera para o Impala diz isso sobre os conjuntos de caracteres usados pelo Impala:

Para suporte total em todos os subsistemas Impala, restrinja os valores de string ao conjunto de caracteres ASCII. Embora alguns dados de caracteres UTF-8 possam ser armazenados no Impala e recuperados por meio de consultas, não é garantido que as strings UTF-8 contendo caracteres não ASCII funcionem corretamente em combinação com muitos aspectos SQL.…
Para quaisquer aspectos de idioma nacional, como ordem de agrupamento ou interpretação de variantes ASCII estendidas, como codificações ISO-8859-1 ou ISO-8859-2, a Impala não inclui esses metadados com a definição da tabela. Se você precisar classificar, manipular ou exibir dados dependendo das características do idioma nacional dos dados de string, use a lógica no lado do aplicativo.

O Hive tem suporte total para o conjunto de caracteres UTF-8, mas nenhum outro. Do FAQ do usuário do Hive:

Você pode usar a string Unicode em dados/comentários, mas não pode usar para nome de banco de dados/tabela/coluna
Você pode usar a codificação UTF-8 para dados do Hive. No entanto, outras codificações não são suportadas (HIVE-7142 introduz a codificação para LazySimpleSerDe, no entanto, a implementação não está completa e [não] aborda todos os casos).

Tanto o Hive quanto o Impala também incluem funções de string (como **`encode()`** e **`decode()`** no Hive e **`base64encode()`** e **`base64decode()`** no Impala) que podem ajudar na transmissão de dados em caracteres diferentes de UTF-8 ou ASCII.

5.3. Usando Sqoop para Carregar Dados de Bancos de Dados Relacionais

5.3.1 Usando Sqoop para importar dados

Apache Sqoop é uma ferramenta de código aberto que foi originalmente criada na Cloudera. Seu nome vem da contração de “SQL to Hadoop”; ele move dados entre um sistema de gerenciamento de banco de dados relacional (RDBMS) e HDFS. Por exemplo, ele pode importar todas as tabelas de um banco de dados, apenas uma tabela ou apenas parte de uma tabela, como colunas ou linhas específicas. Ele também pode exportar dados do HDFS para um banco de dados relacional. Você verá mais sobre algumas dessas opções nas próximas leituras.

Sqoop é uma ferramenta de linha de comando que oferece vários comandos. O comando Sqoop **import** é usado para importar os dados em uma única tabela em um RDBMS para um diretório no HDFS. O exemplo a seguir importará todas as colunas da tabela **customers** no banco de dados da empresa no MySQL. No exemplo abaixo, o caractere **$** representa o prompt do shell (terminal) do sistema operacional e o caractere **** (barra invertida) é usado para continuar o comando em várias linhas.

```
$sqoop import \
    --connect jdbc:mysql://localhost/company \
    --username jdoe \
    --password bigsecret \
    --table customers
```

Este comando cria um subdiretório chamado **customers** no diretório inicial do usuário no HDFS e preenche esse subdiretório com arquivos contendo todos os dados da tabela **customers** no RDBMS. Por padrão, o Sqoop armazena os dados em arquivos de texto simples, onde cada linha do arquivo é um registro da tabela e os campos são separados por vírgulas. Esses padrões podem ser alterados adicionando opções, que são descritas na próxima leitura.

Por padrão, o Sqoop usa JDBC para se conectar ao banco de dados. No entanto, dependendo do banco de dados, pode haver um conector específico do banco de dados mais rápido disponível, que você pode usar usando a opção **--direct**.

Se a tabela cujos dados você está importando não tiver uma chave primária, você deverá especificar uma usando **--split-by *column***. Se você for dividir por uma coluna de string ou se a chave primária de uma tabela for uma coluna de string e estiver usando algumas versões mais recentes do Sqoop (a partir de 1.4.7), inclua a configuração

```
-Dorg.apache.sqoop.splitter.allow_text_splitter=true
```

imediatamente após o comando **import**.

Para importar todas as tabelas de um banco de dados para o HDFS, use o comando Sqoop **import-all-tables**. Este exemplo traz todas as tabelas do banco de dados da empresa para o HDFS.

```
$ sqoop import-all-tables \
    --connect jdbc:mysql://localhost/company \
    --username jdoe \
    --password bigsecret
```

Experimente!

A VM possui tabelas em um banco de dados MySQL. Embora eles já sejam importados para o metastore do Hive, faça o seguinte para reimportar um, apenas para praticar o uso do comando **import** do Sqoop.

1. Abra uma janela de terminal, se ainda não tiver uma, e execute o seguinte comando. A tabela **card_rank** que você está importando não tem uma chave primária, então você precisa especificar uma. O mais razoável é o **rank**, que é uma coluna de texto, então inclua o

   ```
   -Dorg.apache.sqoop.splitter.allow_text_splitter=true
   ```

   configuração também.

   ```
   $sqoop import \
       -Dorg.apache.sqoop.splitter.allow_text_splitter=true \
        --connect jdbc:mysql://localhost/mydb\
       --username training \
       --password training \
   ```

```
--table card_rank \
--split-by rank
```

2. Verifique se o HDFS agora tem **/user/training/card_rank**, com pelo menos um arquivo nele.

3. Revise um dos arquivos para ver como os campos são delimitados. (Você precisará disso quando criar a tabela.)

4. Embora tenha importado os dados, você ainda não tem uma tabela para eles. (A tabela **card_rank** existente está no banco de dados **fun** e seus dados estão em **/user/hive/warehouse/fun.db/card_rank**.) Execute uma instrução **CREATE TABLE** para criar uma tabela no banco de dados **default**, **default.card_rank**, com as seguintes colunas. Certifique-se de usar **ROW FORMAT** para especificar o delimitador e use uma cláusula **LOCATION** para especificar o local dos dados como **'/user/training/card_rank/'**.

name	type
rank	STRING
value	TINYINT

5. Execute uma consulta em sua nova tabela **default.card_rank** ou verifique os dados de amostra do painel de fonte de dados no Hue, para ver se sua tabela tem os dados que você importou. Observação: se você criou a tabela no Impala, não precisará atualizar os metadados, pois os dados estavam lá quando você criou a tabela.

6. Agora você pode eliminar a tabela **default.card_rank**. (Tenha cuidado para não deixar apagar a outra tabela **card_rank**.)

5.3.2 Mais opções de importação Sqoop

Existem muitas opções para importar tabelas usando Sqoop. Alguns dos mais comumente usados são dados abaixo; para saber mais, consulte a documentação em https://sqoop.apache.org. Certifique-se de usar a documentação para a versão que você está usando. Para ver qual versão do Sqoop você está usando, execute o comando **sqoop version**.

--target-dir especifica o diretório HDFS **/mydata/customers** como o local em que os dados serão salvos:

```
$sqoop import \
    --connect jdbc:mysql://localhost/company \
    --username jdoe \
    --password bigsecret \
    --tabela customers \
    --target-dir /mydata/customers
```

--warehouse-dir especifica o diretório pai do HDFS a ser usado para a importação; O Sqoop criará um subdiretório correspondente ao nome da tabela no diretório pai especificado, portanto, este exemplo também criará **/mydata/customers** no HDFS:

```
$sqoop import \
    --connect jdbc:mysql://localhost/company \
    --username jdoe \
    --password bigsecret \
    --table customers \
    --warehouse-dir /mydata
```

--fields-terminated-by especifica o delimitador entre colunas; neste exemplo, **\t** (tab) é o delimitador:

```
$sqoop import \
    --connect jdbc:mysql://localhost/company \
    --username jdoe \
    --password bigsecret \
    --table customers \
    --fields-terminated-by '\t'
```

--columns especifica colunas específicas a serem importadas, em vez de todas elas; aqui, **--columns** é usado para especificar as colunas **prod_id**, **name** e **price** da tabela de produtos, portanto, apenas essas três colunas serão importadas para o HDFS:

```
$sqoop import \
      --connect jdbc:mysql://localhost/company \
      --username jdoe \
      --password bigsecret \
      --table products \
      --columns "prod_id,name,price"
```

--where fornece um filtro para importar apenas linhas específicas da tabela; o exemplo a seguir usa um exemplo simples em que o valor na coluna de preço deve ser maior ou igual a 1.000, mas você pode usar expressões mais complexas, incluindo aquelas com **AND** e **OR**:

```
$sqoop import \
    --connect jdbc:mysql://localhost/company \
    --username jdoe \
    --password bigsecret \
    --table products \
    --columns "prod_id,name,price" \
    --where "price >= 1000"
```

5.3.3 Usando Sqoop para exportar dados

Você não apenas pode importar dados de um RDBMS para o Hadoop, mas também pode enviar dados de outra maneira, usando o comando **export** do Sqooq. Por exemplo, suponha que algumas recomendações de novos produtos tenham sido geradas após algum processamento no cluster Hadoop. Essas recomendações precisam ser exportadas para o banco de dados de back-end do site. Isso pode ser feito com o seguinte comando:

```
$ export sqoop \
    --connect jdbc:mysql://localhost/company \
    --username jdoe \
    --password bigsecret \
    --table product_recommendations \
    --export-dir /mydata/recommender_output
```

O argumento **--export-dir** especifica onde os dados a serem exportados estão localizados no HDFS; neste caso, está no diretório **/mydata/recommender_output**. A tabela de destino no RDBMS é identificada na opção **--table**; neste caso, a tabela será **product_recommendations**. Observe que isso apenas exporta os dados, não cria a tabela no RDBMS — a tabela de destino já deve existir.

Embora o Sqoop permita importar todas as tabelas para o HDFS usando um único comando, ele não possui um comando para exportar mais de uma tabela. A exportação deve ser feita em um diretório de tabela por vez.

Para obter mais opções, consulte a documentação em https://sqoop.apache.org. Certifique-se de usar a documentação para a versão que você está usando. Para ver qual versão do Sqoop você está usando, execute o comando **`sqoop version`**.

5.4. Usando Hive e Impala para Carregar Dados em Tabelas

5.4.1 Instruções SQL LOAD DATA

Uma maneira de carregar dados em uma tabela é executando uma instrução **`LOAD DATA INPATH`** com Hive ou Impala. Isso move os arquivos de dados especificados para o diretório de armazenamento da tabela no sistema de arquivos (HDFS ou S3).

O exemplo mostrado abaixo move o arquivo **`sales.txt`** do diretório HDFS **`/incoming/etl`** para o diretório da tabela denominada **`sales`**. Observe que esta instrução especifica o destino como um nome de tabela, não um diretório; Hive ou Impala usa metadados do metastore para determinar o diretório de armazenamento da tabela e move o arquivo para lá.

```
LOAD DATA INPATH '/incoming/etl/sales.txt' INTO TABLE sales;
```

O caminho de origem pode se referir a um arquivo, como neste exemplo, ou a um diretório, caso em que o Hive ou Impala moverá todos os arquivos desse diretório para a tabela. A instrução **`LOAD DATA INPATH`** adiciona os arquivos de origem a qualquer arquivo existente que já esteja no diretório da tabela - ou seja, não remove os arquivos existentes no diretório da tabela e, se houver alguma colisão de nome de arquivo, renomeia automaticamente os novos arquivos para que nenhum arquivo existente seja substituído. Em alguns casos, você pode querer excluir todos os arquivos de dados existentes no diretório da tabela antes de carregar novos arquivos de dados. Para fazer isso, use a palavra-chave **`OVERWRITE`**, conforme mostrado no exemplo abaixo. Esta opção é útil quando você precisa fazer um recarregamento completo de todos os dados de uma tabela.

```
LOAD DATA INPATH '/incoming/etl/sales.txt' OVERWRITE INTO TABLE sales;
```

A instrução **`LOAD DATA INPATH`** assume que os arquivos já estão em algum lugar em um sistema de arquivos acessível à sua instância do Hive ou Impala (como HDFS ou S3). Se não estiverem, primeiro você precisa carregar arquivos do seu sistema de arquivos local no HDFS ou S3, por exemplo, executando um comando **`hdfs dfs -put`** ou usando o Hue File Browser. Além disso, esse método move os arquivos em vez de copiá-los; os arquivos não existirão mais no diretório de origem depois que a instrução for executada.

Isso provavelmente soa muito como usar **`hdfs dfs -mv`**—e é, mas há algumas vantagens em usar **`LOAD DATA INPATH`**. Uma é que, se você estiver executando a instrução **`LOAD DATA INPATH`** com o Impala, o cache de metadados será atualizado automaticamente. Você não precisa executar um comando **`REFRESH`**; sua próxima consulta incluirá os novos dados. A outra vantagem é que, como a instrução renomeia todos os arquivos que são iguais aos arquivos que já existem no diretório, você não precisa se preocupar com o nome dos arquivos. Se você usar **`hdfs dfs -mv`** e houver um arquivo existente com o mesmo nome, o comando falhará e nenhuma alteração será feita—você terá que renomear um dos arquivos primeiro.

5.4.2 Instruções SQL INSERT

As instruções **`SQL INSERT`**, comuns para adicionar novas linhas de dados a tabelas em RDBMSs, podem ser usadas com Hive e Impala. Existem essencialmente duas versões, **`INSERT INTO`** (que adiciona arquivos sem alterar ou excluir arquivos existentes) e **`INSERT OVERWRITE`** (que substitui arquivos existentes por novos arquivos).

Se você fez o "Experimente!" seções na Semana 2, você as usou para adicionar uma única linha de dados a uma tabela que você criou. Eles também podem ser usados para adicionar várias linhas:

```
INSERT INTO tablename
   VALUES
    (row1col1value,row1col2value, … ),
    (row2col1value,row2col2value, … ),
     … ;
```

Isso pode ser útil para testar uma tabela, como examinar como ela armazena os dados, especialmente com um caso incomum ou extremo. No entanto, como esses exercícios observaram, em geral, esse é um *antipadrão* com Hive e Impala, e usá-lo para ingestão de dados em um ambiente de produção é uma prática ruim.

Em geral, os arquivos no HDFS são imutáveis — eles não podem ser modificados diretamente. (Um arquivo no HDFS pode ser excluído e pode ser substituído por uma nova versão do arquivo, mas em geral um arquivo no HDFS não pode ser modificado diretamente.) Dessa forma, a cada vez que você executa uma instrução **`INSERT`**, Hive ou Impala criam um novo arquivo no diretório de armazenamento da tabela para armazenar os novos valores de dados fornecidos na instrução. Portanto, inserir dados em pequenos lotes como esse faz com que o Hive ou Impala crie muitos arquivos pequenos no diretório de armazenamento da tabela. Isto é um problema.

O problema de *arquivos pequenos* é um problema comum em sistemas de big data. A maioria das ferramentas e estruturas executadas em um cluster Hadoop é projetada para funcionar com *arquivos grandes*, não com muitos arquivos pequenos. Portanto, se você tiver muitos arquivos pequenos (**qualquer coisa menor que 64 MB é considerada pequena**), operações como consultas no Hive e Impala se tornarão ineficientes e o desempenho da consulta será afetado negativamente. Cada comando **`INSERT`** cria um novo arquivo, e eles serão bem pequenos (para qualquer quantidade de dados que seja razoável adicionar usando um comando **`INSERT`**).

Uma solução corretiva para isso, quando você descobrir que tem muitos arquivos pequenos, é reescrever todo o conjunto de dados usando este comando:

```
INSERT OVERWRITE tablename SELECT * FROM tablename;
```

Você aprenderá mais sobre as instruções **`INSERT … SELECT`** na próxima leitura.

Observe que o problema de arquivos pequenos também pode surgir de outros usos. Por exemplo, as imagens geralmente são arquivos separados e não há uma maneira fácil de combiná-los em um arquivo, portanto, um grande número de imagens provavelmente será armazenado como um grande número de arquivos, potencialmente pequenos, dependendo da resolução e do tamanho do arquivo. a imagem. A solução corretiva descrita acima não ajuda muito nesse caso; existem outras maneiras de contornar esse problema, mas isso está além do escopo deste curso. Se você estiver interessado em ler mais sobre o problema de arquivos pequenos, consulte a postagem do blog Cloudera, The Small Files Problem.

Nota: Você pode ver a sintaxe **`INSERT INTO TABLE`** ou **`INSERT OVERWRITE TABLE`**. A palavra-chave **`TABLE`** é opcional. Você pode incluí-lo ou não, como quiser, mas é útil saber que ambas as sintaxes são válidas, caso você veja a que você decide não usar.

Você pode ler mais sobre como usar **`INSERT`** no Hive e Impala usando estes links:

- Hive LanguageManual UDF, inserindo dados em tabelas do Hive a partir de consultas
- Documentação do Impala da Cloudera, declaração INSERT

5.4.3 Instruções SQL INSERT ... SELECT e CTAS

Como você já deve saber, a instrução **SELECT** no SQL retorna um conjunto de resultados. Normalmente, você visualiza esse conjunto de resultados ou o armazena em um arquivo ou na memória do computador local, onde pode usá-lo para gerar um relatório ou visualização de dados. No entanto, você pode fazer mais com a instrução **SELECT** do que apenas visualizar os resultados ou armazená-los localmente. O conjunto de resultados de uma instrução **SELECT** tem a mesma estrutura básica de uma tabela, portanto, você pode usar um conjunto de resultados para materializar uma nova tabela. Como você aprenderá nesta leitura, é possível fazer isso com um único comando. Esse comando permite que você salve o resultado de uma consulta em uma tabela, para que você possa executar outra consulta posteriormente para analisar ou recuperar esse resultado.

A maneira de fazer isso é combinando um **INSERT** e um **SELECT** em um único comando. Usando uma instrução **INSERT**, você pode especificar o nome da tabela de destino, seguido por uma instrução **SELECT**. Esse tipo composto de instrução é conhecido como instrução **INSERT ... SELECT**. Hive ou Impala executa a instrução **SELECT** e salva os resultados na tabela especificada. Observe que a tabela de destino já deve existir. Se você deseja substituir qualquer dado existente, use **INSERT OVERWRITE**; se você quiser reter quaisquer dados existentes, use **INSERT INTO**.

Por exemplo, talvez você queira criar uma nova tabela, **chicago_employees**, que contenha apenas as linhas da tabela de **emplooyees** para funcionários do escritório de Chicago (**office_id = 'b'**). Depois de criar essa tabela (talvez usando **LIKE employees** no comando **CREATE TABLE**), o comando a seguir preencherá a tabela com as linhas desejadas. Quaisquer registros existentes anteriormente na tabela **chicago_employees** são excluídos.

```
INSERT OVERWRITE chicago_employees
    SELECT * FROM employees WHERE office_id='b';
```

As quebras de linha e o recuo usados nesses exemplos são opcionais. O recuo serve para mostrar que a instrução **SELECT** é realmente parte da instrução **INSERT**.

Como mencionado acima, uma instrução **INSERT … SELECT** requer que a tabela de destino já exista. Mas há uma instrução diferente que não requer isso: a instrução **CREATE TABLE AS SELECT** (CTAS). Uma instrução CTAS cria uma tabela e a preenche com o resultado de uma consulta, tudo em um comando.

Por exemplo, a instrução CTAS a seguir cria a tabela **chicago_employees** e carrega os mesmos dados que o comando **INSERT … SELECT** acima, mas faz tudo em um único comando.

```
CREATE TABLE chicago_employees AS
    SELECT * FROM employees WHERE office_id='b';
```

O CTAS combina efetivamente uma operação **CREATE TABLE** e uma operação **INSERT … SELECT** em uma única etapa. Os nomes das colunas e os tipos de dados da tabela recém-criada são determinados com base nos nomes e tipos das colunas consultadas na instrução **SELECT**.

No entanto, os outros atributos da tabela recém-criada, como o delimitador e o formato de armazenamento, não são baseados no formato da tabela que você está consultando; você deve especificar esses atributos, ou então a tabela recém-criada usará os padrões. Portanto, no exemplo aqui, como não há cláusula **ROW FORMAT**, Hive ou Impala usarão o delimitador de campo padrão, que é o caractere ASCII Control+A. Se você quisesse arquivos delimitados por vírgulas, precisaria incluir uma cláusula **ROW FORMAT**. Você deve colocar isso (e quaisquer outras cláusulas que especifiquem as propriedades da nova tabela de destino) antes da palavra-chave **AS**, conforme mostrado no exemplo abaixo.

```
CREATE TABLE chicago_employees
        ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
    AS
        SELECT * FROM employees WHERE office_id='b';
```

Com CTAS e **INSERT … SELECT**, você pode selecionar apenas algumas colunas para incluir na nova tabela listando apenas as colunas desejadas na lista **SELECT**. Por exemplo, como todos os registros estão no mesmo office, a coluna **office_id** não é realmente necessária. Você poderia usar:

```
CREATE TABLE chicago_employees AS
        SELECT empl_id, first_name, last_name, salary
            FROM employees
            WHERE office_id='b';
```

Observe que, se os dados na tabela de **employees** forem atualizados *após* a criação da tabela **chicago_employees**, os dados na tabela **chicago_employees** ficarão obsoletos - nenhum novo funcionário em Chicago será adicionado automaticamente à tabela **chicago_employees**. Portanto, se você estiver usando uma instrução CTAS ou **INSERT … SELECT** para criar uma tabela derivada que deseja manter atualizada com a tabela original, precisará configurar algum processo para mantê-la atualizada. Por exemplo, você pode agendar um trabalho que execute uma instrução **INSERT OVERWRITE … SELECT** para repovoar a tabela derivada todas as noites, para que os dados nunca fiquem mais de 24 horas obsoletos.

Experimente!

1. Use as informações acima para criar a tabela **chicago_employees** em duas etapas—crie a tabela usando **LIKE employees** (para que a coluna **office_id** seja incluída) e preencha-a com a instrução **INSERT … SELECT** acima. Verifique se a nova tabela possui apenas duas linhas, para Virginia e Luzja.
2. Elimine a tabela **chicago_employees** (excluindo os dados — exclua-a manualmente se você criou a tabela como **EXTERNAL**) e, em seguida, recrie-a sem a coluna **office_id**, usando uma instrução CTAS. Novamente, verifique se a nova tabela tem as duas linhas.

6. Analisando Big Data com SQL

6.1. Rodando instruções utilitárias em SQL

Como já vimos, o Hue permite que você veja quais bancos de dados existem, alternar para um determinado banco de dados, ver quais tabelas estão nele, e olhar para as colunas nessas tabelas, todas as ações de ponto e clique. Veremos aqui como para cada uma dessas tarefas, você pode escrever e executar uma instrução utilitária do SQL que faz a mesma coisa que a ação apontar e clicar. Então, se você pudesse fazer algo apontando e clicando, por que você iria querer escrever uma instrução SQL para fazê-lo? Bem, para começar, nem todas as interfaces SQL têm uma interface gráfica de usuário como Hue tem. Às vezes, a única maneira de executar tarefas simples como estas é inserindo e executando instruções SQL. Além disso, uma interface como o Hue permite que você execute algumas tarefas simples sem usar SQL. Mas como você verá que pode alcançar muitos mais tipos de tarefas inserindo e executando instruções SQL.

A primeira instrução utilitária SQL é SHOW DATABASES. Esta é muitas vezes a primeira instrução que você executaria ao se conectar a uma instância de um mecanismo SQL pela primeira vez. Ele informa quais bancos de dados existem.

Quando você está usando um mecanismo SQL, há sempre um banco de dados específico ao qual você está conectado. Isso é chamado de banco de dados atual ou o banco de dados ativo. Quando você faz login pela primeira vez no Hue e abre o Hive ou Impala Query Editor, o banco de dados atual é normalmente o nome padrão. Outros mecanismos SQL também têm bancos de dados específicos aos quais eles se conectam por padrão no início de uma nova sessão, mas geralmente não são nomeados padrão, e podem variar de usuário para usuário. Normalmente, se você estiver indo para trabalhar com uma tabela específica, você vai querer definir o banco de dados atual para o banco de dados que contém essa tabela. Para definir qual banco de dados é o banco de dados atual, você pode executar uma instrução USE. A instrução USE é muito simples, é apenas a palavra-chave USE seguido pelo nome de um banco de dados. Hue, que você usará ao longo deste curso na verdade não suporta a instrução USE. Em vez disso, no Hue, você sempre usa as ações de ponto e clique para definir o banco de dados atual. Logo acima do editor, há um seletor de banco de dados ativo. Você pode usar isso para ver o que é o banco de dados atual, agora é padrão, e para alterar o banco de dados atual.

Lembre-se de que um banco de dados no SQL é apenas um contêiner lógico para um grupo de tabelas. Então, depois de ver quais bancos de dados existir e alterar o banco de dados atual, muitas vezes o próximo passo é ver quais tabelas existem no banco de dados atual. Para fazer isso, execute a instrução SHOW TABLES.

Por fim, abordaremos a declaração DESCRIBE. Você pode usar a instrução DESCRIBE para ver quais colunas estão em uma tabela. A sintaxe é simples, seguindo a palavra-chave DESCRIBE, você coloca o nome da tabela cujas colunas você deseja ver. O resultado mostra as colunas pertencentes a uma tabela. Cada coluna também tem um tipo de dados e, opcionalmente, um comentário. Preste atenção nos nomes das colunas e a ordem em que eles estão.

6.2. Instrução SELECT

A instrução Select é a parte mais importante da linguagem SQL. As opções para o que você pode fazer com uma instrução Select são tão extensas, que Select forma sua própria categoria de instruções SQL chamadas queries. Na VM que estamos usando para este curso, você escreverá e executará instruções select no Hue, usando o Hive ou o Impala Query Editor. Essas consultas serão executadas no Hive ou Impala, e você visualizará os resultados em Hue.

No Impala Query Editor em Hue, insira e execute uma consulta simples para retornar todos os dados em uma tabela. O banco de dados atual é **Fun**. Eu sei que há uma tabela chamada **games** neste banco de dados.

Para retornar todas as colunas e todas as linhas desta tabela, eu entrarei na consulta, "**SELECT * FROM games**". O asterístico significa todas as colunas. Observe que o editor tem alguns recursos de preenchimento automático que sugerem nomes de tabelas e colunas de banco de dados disponíveis, e outra sintaxe de consulta.

Os dados que são retornados por uma instrução SQL são chamados de conjunto de resultados ou apenas o resultado. Este conjunto de resultados tem todas as cinco linhas e oito colunas da tabela de jogos. Você pode ver que esta tabela contém algumas informações sobre cinco jogos de tabuleiro diferentes, Monopoly, Scrabble, Clue, Candy Land e Risk. Você pode ver as colunas ID, nome, inventor, ano, idade mínima, min players, max players e preço sugerido. É importante pausar aqui por um minuto para falar sobre a ordem das colunas e linhas no conjunto de resultados. A ordem das colunas em um conjunto de resultados é determinada pela sua consulta, ou pela estrutura da tabela que você está consultando. Não há nada aleatório sobre a ordem da coluna.

No exemplo exibido, a consulta retornou todas as colunas. Assim, o conjunto de resultados mostrou todos eles. Sua ordem da esquerda para a direita foi determinada pela estrutura da tabela **games**, ID, nome, inventor e assim por diante. Para conferir, use "DESCRIBE games" para ver quais colunas estavam na tabela de jogos, que retornou os nomes da mesma coluna, na mesma ordem de cima para baixo, ID, nome, inventor e assim por diante. Eu também tenho a opção de especificar na instrução select quais colunas retornar, e em que ordem eu quero que elas sejam.

Quando você executa uma instrução select usando um mecanismo SQL distribuído, ***a ordem das linhas no conjunto de resultados é arbitrária e imprevisível***. Você pode executar exatamente a mesma consulta duas vezes em dados que não foram alterados, e obter as linhas em uma ordem diferente cada vez. O resultado geral será o mesmo, mas a ordem das linhas pode variar. Então, se você executar esta consulta, não se surpreenda se as linhas que você vê estiverem em uma ordem diferente de quando eu a executei. Isso é normal e esperado quando você está usando um mecanismo SQL distribuído. Mas você também não deve se surpreender se a ordem de as linhas em seus resultados são as mesmas que as minhas. Na nossa VM, esses mecanismos SQL distribuídos, Hives, e Impala, não são realmente distribuídos em vários computadores ou vários processadores. Configuramos a VM para usar apenas um processador em seu computador, e isso tira grande parte da aleatoriedade que faz com que as linhas sejam embaralhadas.

Expressões e operadores do SELECT

Uma instrução SELECT começa com a palavra-chave SELECT. A parte da instrução começando em no início com as palavras-chave SELECT e terminando antes da palavra-chave FROM é chamada de cláusula SELECT. Tudo o que vem depois a palavra-chave SELECT em esta cláusula é chamada de lista SELECT. Uma lista de seleção pode incluir referências de coluna, valores literal e o símbolo de asterisco, que significa todas as colunas. Uma lista de seleção também pode incluir expressões.

Uma expressão em SQL é uma combinação de valores literal, referências de coluna , operadores e funções. Considere a tabela games, no banco de dados fun, que tem uma coluna chamada **list_price**. Digamos que eu queira devolver os nomes de estes jogos e seus preços sugeridos, mas com uma taxa de transporte de cinco dólares adicionado ao preço. Para fazer isso, eu usaria a consulta:

SELECT name, list_price +5 FROM games

Nesta lista SELECT, name é apenas uma referência de coluna, mas list_price + 5 é uma expressão. Consiste em uma referência de coluna, list_price, um operador, o sinal de +, e um valor literal, 5. Quando o mecanismo SQL avalia a expressão, ele retorna uma coluna de resultado na qual o valor em cada linha é o valor de list_price nessa linha + 5.

Aqui está outro exemplo de uma expressão usando uma função em vez de um operador.

SELECT name, round(list_price) FROM games

A função round arredonda os números decimais para o número inteiro mais próximo. Então, neste caso, ele arredonda cada list_price para o dólar mais próximo, 19.99 arredonda para 20, etc.

Essas expressões de exemplo eram muito simples, mas expressões em SQL podem ser arbitrariamente complexas, desde que sejam compostas de valores literal, referências de colunas , operadores e funções juntas de forma válida. Round é um exemplo de uma função embutida.

Operadores e funções	Descrição
+	Soma
-	Subtração
*	Multiplicação
/	Divisão
%	Módulo
round(num, casas_decimais)	Arredonda o num de acordo com o número de casas decimais definido.
ceil(num)	Arredonda o num para cima
floor(num)	Arredonda o num para baixo
ABS(num)	Retorna o valor absoluto de um número
Sqrt(num)	Raiz quadrada
Pow(base, potencia)	Eleva a base a potência
Rand()	Gera um número decimal pseudo-aleatório entre 0 <= num < 1
Strings	
Len(str)	Tamanho da string
Reverse(str)	Inverte uma string
Upper(str)	Converte para maiúsculo
Lower(str)	Converte para minúsculo

Palavra-chave DISTINCT

DISTINCT é uma palavra-chave que você pode inserir logo após a palavra-chave SELECT para modificar o que a instrução select faz. Ela elimina linhas redundantes em uma consulta.

Cláusula WHERE

A cláusula Where filtra as linhas de dados com base em uma ou mais condições. Normalmente, essas condições são testes de os valores em colunas especificadas para todas as linhas dos dados. Em outras palavras, **Where** leva todos os dados na tabela, testa que linhas atendem a alguns critérios, e retorna apenas essas linhas.

A cláusula Where não tem efeito sobre a qual as colunas são retornadas apenas em que linhas são retornadas. A cláusula Where é opcional. Se você executar uma instrução Select que tem uma cláusula Select e uma cláusula From, mas nenhuma cláusula Where, então você obtém um conjunto de resultados que tem tantas linhas quanto a tabela especificada na cláusula From.

A cláusula Where é onde você realmente começa a fazer análise de dados, porque você pode usá-lo para responder perguntas em a forma em que as linhas são essas condições verdadeiras. Estas condições são a pergunta que você está fazendo sobre os dados, e o resultado que você obtém de volta contém a resposta para essa pergunta pronta para você interpretar.

Por exemplo, olhando para a tabela games no banco de dados fun, você poderia perguntar, quais jogos são preços abaixo $10? Para responder a essa pergunta, você iria filtrar as linhas para retornar apenas os jogos onde list_price é menor que 10.

SELECT * FROM games WHERE price < 10

Trabalhando com valores ausentes

Quando você está filtrando dados com base em algumas condições, é importante considerar se as condições são conhecidas para todas as linhas ou não. A presença de valores ausentes ou desconhecidos nos dados, pode tornar impossível determinar se as condições são verdadeiras ou falsas. Por exemplo, se o preço de algum jogo de tabuleiro é desconhecido, então é impossível determinar se esse preço é inferior a $10.

A realidade é que muitos conjuntos de dados têm, e, como analista de dados, você precisará lidar com eles corretamente. Em SQL, um valor nulo é um valor que está ausente ou desconhecido. Isso é representado pela palavra-chave “null” em expressões SQL, e em conjuntos de resultados. O significado ou interpretação de valores nulos, o que eles realmente representam, pode diferir com base no contexto. Na tabela de escritórios, a linha que representa esses escritórios de Cingapura tem um valor nulo na coluna state_province. A razão pelo qual o valor é nulo, é que Singapura não é dividido em estados ou províncias. Então o nulo significa não aplicável. Na tabela de inventário, há um nulo na coluna do corredor e outro na coluna de preço.

Conjuntos de dados do mundo real muitas vezes têm muitos valores ausentes neles. Há muitos conjuntos de dados no mundo real que têm mais valores ausentes do que valores não ausentes. Estes são referidos como conjuntos de dados esparsos.

Valores nulos são na sua maioria simples para interpretar quando você pode identificá-los. Você pode ter algumas perguntas sobre o que eles significam, mas pelo menos você sabe que eles estão lá e você pode interpretar os resultados de acordo. No entanto, os valores nulos tornam-se mais de um problema complicado quando você inadvertidamente os filtra fora de seus resultados usando uma cláusula WHERE. Lembre-se que quando você usa uma cláusula WHERE, você especifica as condições de filtragem com uma expressão booleana, e somente as linhas para as quais a expressão booleana avalia como true são retornadas no conjunto de resultados. Até agora, assumimos que para cada linha, uma expressão booleana avaliará como verdadeiro ou falso. Mas se uma tabela tem valores nulos nele, então há uma terceira possibilidade, uma expressão booleana poderia avaliar como nulo. Isso porque, como eu disse no início deste vídeo, valores ausentes podem tornar impossível determinar se algumas condições são verdadeiras ou falsas.

Linhas nas quais a expressão booleana na cláusula WHERE é avaliada como null, são omitidas dos resultados assim como as linhas em que ela avalia como false. Este é um conceito realmente importante para entender, então eu vou repeti-lo. Quaisquer linhas em que a expressão na cláusula WHERE avalia como false e quaisquer linhas em que ela seja avaliada como nula, são filtradas excluídas do conjunto de resultados. Isso pode ter implicações importantes para interpretar os resultados da consulta.

Como analista de dados, você pode evitar muitos problemas e culpas por estar consciente da possibilidade de faltar valores, e frasear suas interpretações para explicar essa possibilidade. Também é importante verificar explicitamente para valores nulos e tratá-los em suas consultas. Além disso, sistemas de banco de dados relacional tradicionais, possuem o conceito de restrições nulas. Restrições nulas podem impedir que dados com valores nulos sejam carregados em o banco de dados em primeiro lugar, mas mecanismos SQL distribuídos como Hive e Impala, geralmente não suportam essas restrições. A maneira como eles trabalham faz com que seja impraticável. Portanto, muitas vezes você precisa assumir que qualquer coluna em qualquer tabela poderia ter valores nulos nela.

7. Tecnologias NoSQL

7.1. Introdução ao NoSQL

O modelo de banco de dados relacional do usuário foi considerado para resolver quase qualquer problema de gerenciamento e armazenamento de dados. Além disso, os principais sistemas de gerenciamento de banco de dados relacional absorveram propostas alternativas, como as bases de dados em documentos XML e bancos de dados orientados a objetos. Longe de substituir os sistemas de gerenciamento de banco de dados relacional, eles complementam e ampliam suas características funcionais.

Mas quando as limitações do sistema de gerenciamento de banco de dados relacional se tornam evidentes, e os desafios de armazenamento de dados não estruturado crescem, a ideia de ter um único sistema de gerenciamento de banco de dados relacional tradicional que resolve tudo foi questionada. É quando os bancos de dados NoSQL chegam a este cenário, onde os bancos de dados NoSQL diferem da tecnologia relacional em três áreas principais: modelos de dados, estrutura de dados e modelo de desenvolvimento.

No caso de modelos de dados, um banco de dados NoSQL permite que você crie um aplicativo sem precisar definir o esquema primeiro ao contrário de bancos de dados relacionais.

Em relação à estrutura de dados, os bancos de dados NoSQL são projetados para lidar com dados não estruturados , como texto, mídia social, postagem, vídeo ou e-mail, que compõe grande parte dos dados que existem hoje.

Em relação ao modelo de desenvolvimento, bancos de dados NoSQL são de código aberto, enquanto bancos de dados relacionais geralmente são de código fechado com taxas de licenciamento cozido no uso de seu software.

Com o NoSQL, você pode começar em um projeto sem qualquer investimento pesado em taxas de software antecipadamente. Agora, o que significa NoSQL? Bem, NoSQL foi popularizado para se referir a bancos de dados que não suportam o modelo relacional, e eles não usam a linguagem de consulta estruturada ou bancos de dados relacionais. O termo NoSQL agrupa todas essas tecnologias, como gráficos, documentos, texto etc. Que, por definição, são não-relacionais.

Uma definição mais precisa de NoSQL corresponde a todos os bancos de dados de nova geração que são não-relacionais, distribuídos, de código aberto, sem esquema e horizontalmente escaláveis. Resumindo, NoSQL, agora, não só SQL é uma categoria geral de sistema de gerenciamento de banco de dados que difere do sistema de gerenciamento de banco de dados relacional de maneiras diferentes. Eles não têm um esquema, eles não permitem junções, eles não tentam garantir as propriedades ACID de transação e eles dimensionam horizontalmente.

O movimento NoSQL também chamou a atenção de várias empresas que desenvolveram tecnologias NoSQL, como Facebook, que desenvolveram o sistema Cassandra, mais tarde usado pelo Twitter. Além disso, o Linkedin desenvolveu o Project Voldemort e o Ubuntu One, um sistema de armazenamento em nuvem sincronizado baseado no CouchDB. Além dos aplicativos da web, os bancos de dados NoSQL também oferecem suporte a atividades diversas, incluindo análise preditiva e sistemas transacionais não críticos, aqueles que não exigem as propriedades ACID.

7.1.1 Quais são os benefícios de usar um banco de dados NoSQL?

Em primeiro lugar, como os dados são armazenados sem esquema, eles são gravados maiores velocidades em um banco de dados NoSQL geralmente são rápidos e, em seguida, fornecem alto desempenho. Por exemplo, a tecnologia Hypertable é capaz de armazenar um trilhão de dados por dia. Outro exemplo é um software Google

BigTable, que pode processar 20 petabytes de dados em um dia. Outro benefício é que, se um banco de dados experimenta algum crescimento, é possível adicionar nós a um sistema distribuído é tal forma que ele fornece processamento e armazenamento economicamente, também conhecido como escalabilidade horizontal.

Os bancos de dados NoSQL têm os mecanismos necessários para a escalabilidade horizontal. Em termos de armazenamento, o modelo de armazenamento NoSQL é muito mais simples do que o esquema de tabela do modelo relacional. Isso afeta positivamente o desempenho, pois não há nenhum esquema definido para salvar dados. Além disso, o armazenamento NoSQL é mais flexível do que o esquema rígido de tabelas. Na realidade, bancos de dados NoSQL falta de um esquema fixo.

Quando os dados a serem armazenados não podem ser convertidos em tabelas do modelo relacional, é conveniente procurar uma solução pelo NoSQL. Os bancos de dados NoSQL são uma boa opção para os sistemas de informação que não exigem as propriedades AC. Portanto, sem uma gestão rigorosa de uma transação, o controle da simultaneidade não é feito bloqueando, mas existem outros mecanismos. Como a durabilidade não é um elemento crítico em algumas aplicações web, não é necessário gravar logs para cada transação. Exemplos de dados não sensíveis à durabilidade incluem cópias de dados de sessão de usuário e mensagens de texto em fóruns ou redes sociais.

7.1.2 Desvantagens de bancos de dados NoSQL

NoSQL tem uma linguagem de consulta não padronizada. Na ausência de uma linguagem de consulta padrão para bancos de dados NoSQL, é necessário aprender cada dialeto de consulta para um determinado sistema de gerenciamento de banco de dados. Isso é verdade mesmo dentro de cada categoria de armazenamento NoSQL.

NoSQL tem problemas em transações. Como resultado de não seguir as propriedades ACID, há um controle mais fraco sobre a consistência, durabilidade e isolamento das transações.

NoSQL tem problemas de integridade. Garantir a integridade dos dados requer programação extra manualmente e não é função do SGBD. Integridade é entendida do ponto de vista das restrições, do domínio, referencial de valor nulo, etc

Agora que sabemos as vantagens e desvantagens dos bancos de dados NoSQL, temos o critério para decidir quando usar sistemas de banco de dados relacional, e quando usar banco de dados NoSQL tecnologias.

A decisão sobre que tipo de SGBD deve ser usado depende de um conjunto de fatores, incluindo, mas não limitado a. O volume de dados a serem armazenados. A concorrência estimada. O número de operações que são feitas no banco de dados por unidade de tempo. A escalabilidade desejada do banco de dados. O grau de integridade e consistência que é desejado. A natureza dos dados a serem armazenados. Os tipos mais frequentes de operações que você deseja fazer com os dados.

Existem duas características principais em relação a essas bases de dados iguais e elas são o teorema da CAP e as propriedades de base. Por um lado, o teorema do CAP foi adotado por várias empresas na web e na comunidade NoSQL.

A sigla para CAP refere-se ao seguinte, C como consistência. Refere-se se um sistema está em um estado consistente após a execução de uma operação ou não. Um sistema distribuído é considerado consistente se, após uma operação de atualização por um nó, o resto dos nós ver a atualização em um recurso de dados compartilhado. Um como disponibilidade, isso significa que um sistema é projetado e implementado de tal forma que ele pode continuar sua operação se houver problemas de software ou hardware ou um nó falhar. P como tolerância de partição é a capacidade de um sistema para continuar sua operação na presença de partições de rede. Isso acontece se um conjunto de nós em uma rede perder a conectividade com outros nós. Tolerância

de partição também pode ser considerada como a capacidade de um sistema para adicionar e remover nós dinamicamente.

O máximo que você pode ter duas das três características em um sistema de dados compartilhado. Por outro lado, um banco de dados NoSQL segue um paradigma das propriedades BASE.

BASE significa basicamente disponível. Estado flexível, as informações expirarão, a menos que sejam atualizadas. E eventual consistência. As propriedades BASE podem ser resumidas da seguinte forma, uma aplicação funciona basicamente o tempo todo. Não precisa ser consistente o tempo todo, mas acabará por chegar a um estado conhecido.

As tecnologias NoSQL compreendem uma série de bancos de dados não relacionais. Os mais comuns são os bancos de dados de chave-valor, orientados a colunas, orientados a documentos e orientados a grafos.

No caso de um banco de dados chave-valor, é um sistema que armazena valores indexados por chaves. E então ele pode armazenar dados estruturados e não estruturados. A possibilidade de armazenar qualquer tipo de valor é chamado esquema-menos. Os valores de dados são armazenados como matrizes de bytes. O conteúdo não é importante para o banco de dados. E oferece alto desempenho, muito escalável, muito flexível e baixa complexidade.

A desvantagem dos bancos de dados de valor-chave é que eles não oferecem suporte a consultas complexas, porque eles apenas procuram uma chave. Exemplos de bancos de dados de chave-valor são Amazon, DynamoDB , Cassandra, Voldemort, RamCloud e Flare.

Em relação aos bancos de dados orientados a colunas, como o nome indica, eles armazenam os dados em colunas em vez de linhas. Ou seja, todos os atributos de uma única entidade de dados são armazenados para que cada um deles possa ser acessado como uma unidade. Exemplos de DBMS orientados para colunas incluem Sybase IQ, Cassandra, hyper table e baseado em idade. Os seguintes exemplos ilustram como o armazenamento orientado a coluna é comparado ao armazenamento de linha.

Bancos de dados orientados a colunas não são bons para consultas que exigiam apresentar todo o registro de uma entidade. Neste caso, a loja de linha é melhor. O armazenamento orientado a colunas é especialmente eficiente quando as leituras de dados são maciças e grava em algumas colunas. Isso ocorre porque em uma consulta, apenas os dados das colunas que interessam são obtidos, e não todas as colunas de um registro, o que aumenta a eficiência.

Existem alguns bancos de dados colunares que podem ser projetados como propriedades AC relacionais e de suporte. Um exemplo disso é. Nestes casos, bancos de dados orientados para colunas são amplamente utilizados em business intelligence. Há também documentos em bancos de dados que são comumente valor-chave onde um valor é armazenado como um campo binário com um formato que o DBMS pode entender. Este formato é muitas vezes um documento JSON, JavaScript Object Notation, mas pode ser XML ou qualquer outro. Muitas vantagens são que ele permite consultas muito avançadas sobre os dados. E restaura as relações entre os dados.

Existem alguns bancos de dados de documentos que não permitem operações de associação devido a problemas de desempenho. Exemplos de DBMS orientados a documentos incluem CouchDB, MongoDB, Cloudkit e bancos de dados XML, como DB2 PureXML.

O último tipo de maior parte do banco de dados SQL que mencionarei corresponde a bases orientadas a grafos. Se grafos representados como um conjunto de nós ou entidades interligadas por idade ou relacionamentos. Os grafos dão importância não apenas aos dados, mas às relações entre eles. Relacionamentos também podem ter atributos e consultas diretas podem ser feitas para relacionamentos, em vez de para os nós.

Sendo armazenado dessa forma, é muito mais eficiente navegar entre relacionamentos do que em um modelo relacional.

Obviamente, esses tipos de bancos de dados só são úteis quando a informação pode ser facilmente representada como uma rede. Entre as implementações mais utilizadas estão Neo4J, Hyperbase-DB e InfoGrid. O desempenho e a escalabilidade de um banco de dados orientado a grafos são variáveis, altamente flexíveis e altamente complexas de implementar.

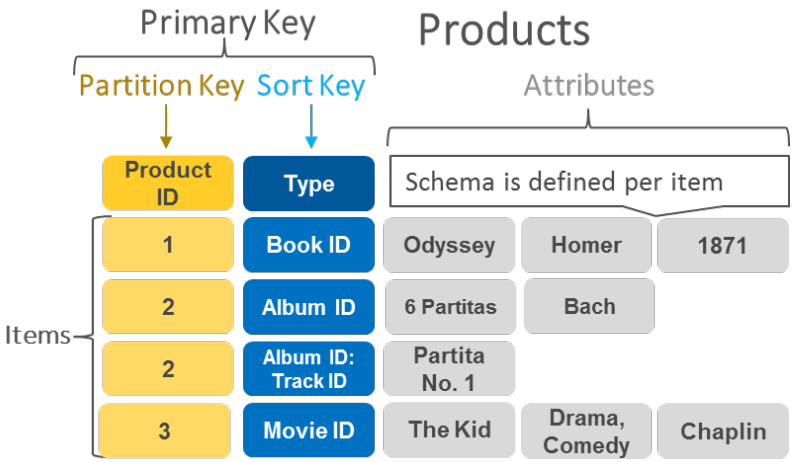
Agora que entendemos as principais características dos bancos de dados NoSQL, como decidir quais deles devemos utilizar. Bem, isso depende do desempenho, escalabilidade, flexibilidade, complexidade e funcionalidade que seu sistema de informações requer. Podemos entender a partir da tabela como estabelece uma comparação entre as diferentes alternativas de armazenamento de dados. Estas características são tendências e generalidades de tais sistemas, mas o desempenho e a escalabilidade dependem muito do tipo e frequência de operações que são aplicadas.

	Desempenho	**Escalabilidade**	**Flexibilidade**	**Complexidade**	**Funcionalidade**
Chave-valor	Alta	Alta	Alta	Nenhuma	Variável
Colunar	Alta	Alta	Moderada	Baixa	Mínimo
Documento	Alta	Variável	Alta	Baixa	Variável
Grafo	Variável	Variável	Alta	Alta	Teoria de grafos
Relacional	Variável	Variável	baixa	moderada	Álgebra relacional

7.2. Bancos de dados Chave-Valor (por AWS)

Um banco de dados de chave-valor é um tipo de banco de dados não relacional que usa um método de chave-valor simples para armazenar dados. Um banco de dados de chave-valor armazena dados como um conjunto de pares de chave-valor em que uma chave funciona como um identificador exclusivo. A chave e os valores podem ser qualquer coisa, desde objetos simples até objetos compostos complexos. Bancos de dados de chave-valor são altamente particionáveis e permitem escalabilidade horizontal que outros tipos de bancos de dados não conseguem alcançar. Por exemplo, o Amazon DynamoDB alocará partições adicionais a uma tabela se uma partição existente for preenchida até o limite de capacidade e mais espaço de armazenamento for necessário.

O diagrama a seguir mostra um exemplo de dados armazenados como pares de chave-valor no DynamoDB.



Casos de uso

Armazenamento de sessões

Um aplicativo orientado por sessão, como um aplicativo da Web, começa uma sessão quando o usuário faz login e fica ativo até que o usuário se desconecte ou a sessão expire. Durante esse período, o aplicativo armazena todos os dados relativos à sessão na memória principal ou em um banco de dados. Os dados da sessão podem incluir informações de perfil do usuário, mensagens, dados e temas personalizados, recomendações, promoções direcionadas e descontos. Cada sessão de usuário tem um identificador exclusivo. Os dados de sessão nunca são consultados por nada além de uma chave primária, então um armazenamento de chave-valor rápido é mais adequado para dados de sessão. Em termos gerais, os bancos de dados de chave-valor podem proporcionar menor sobrecarga por página do que bancos de dados relacionais.

Carrinho de compras

Na temporada de compras de fim de ano, um site de comércio eletrônico pode receber bilhões de pedidos em questão de segundos. Bancos de dados de chave-valor podem lidar com a escalabilidade de grandes quantidades de dados e volumes extremamente altos de mudanças de estado enquanto atendem a milhões de usuários simultâneos por meio do processamento e armazenamento distribuído. Bancos de dados de chave-valor também têm redundância incorporada, que podem lidar com a perda de nós de armazenamento.

Bancos de dados de chave-valor populares

Amazon DynamoDB

O Amazon DynamoDB[6] é um banco de dados não relacional que fornece performance confiável em qualquer escala. O serviço é um banco de dados totalmente gerenciado que pode operar em várias regiões e com vários mestres para oferecer latência consistente abaixo de 10 milissegundos e incorpora recursos de segurança, backup e restauração, além de armazenamento em cache de memória. No DynamoDB, um Item é composto por uma chave primária ou composta e um número flexível de atributos. Não há um limite explícito para o número de atributos associados a um item individual, mas o tamanho agregado de um item, incluindo todos os nomes e os valores de atributo, não pode ultrapassar 400 KB. Uma tabela é um conjunto de itens de dados, assim como uma tabela em um banco de dados relacional é um conjunto de filas. Cada tabela pode ter um número infinito de itens de dados.

Você pode começar a usar o DynamoDB em apenas 10 minutos com este tutorial detalhado[7]. Saiba mais sobre o DynamoDB e comece a usar[8] hoje mesmo.

7.3. Bancos de dados Colunar

Estritamente falando, um banco de dados de armazenamento colunar corresponde a um modelo binário-relacional. E tem suas origens durante a década de 1970. Sendo Chen, a primeira pessoa a mencioná-lo em seu papel transcendental Entity Relational Modeling para uma visão unificada de dados. O modelo relacional baseia-se no conceito matemático de relações.

O modelo Binário-Relacional é uma particularização do modelo relacional, com a restrição de que todas as relações são de grau 2. E é também um banco de dados colunar. Sybase IQ, Paraccel, Vertica, MonetDB e SAND Technologies são exemplos de bancos de dados colunares.

[6] https://aws.amazon.com/dynamodb/
[7] https://aws.amazon.com/getting-started/tutorials/create-nosql-table/
[8] https://aws.amazon.com/dynamodb/getting-started/

Para nós, o nome correto é bancos de dados binários relacionais. E o DBMS comercial que vamos falar é sub Sybase IQ porque tem muitos anos no mercado e é um banco de dados colunar muito forte gerenciado com SQL.

Uma tabela relacional tradicional pode ser decomposta em um modelo binário-relacional como é mostrado na figura.

Em bancos de dados colunares, há uma indexação implícita em cada coluna. Quero dizer, é um armazenamento vertical de dados, que é equivalente a cada coluna em cada tabela é um índice. Somente as colunas envolvidas em uma consulta são tocadas. Isso reduz drasticamente o número de operações de E/S ou de entrada e saída do sistema.

Um terço é levantado para cada coluna individualmente, em seguida, o processamento de carga e consulta são feitos em paralelo. Há uma compactação ao armazenar em colunas, porque em vez de armazenar dados horizontalmente, faça verticalmente e elimine duplicatas. Veja a figura a seguir.

Portanto, nos sistemas Data Warehouse ou OLAP, podemos esquecer o armazenamento horizontal relacional tradicional. Há mais benefícios no uso de bancos de dados colunares. Por exemplo, a capacidade de responder a mais consultas de negócios em menos tempo e com o mesmo hardware.

Existem experimentos que mostram as seguintes teorias sobre armazenamento colunar contra armazenamento linha. 30% de economia de espaço em disco, máximo de 88%. 30% de economia de espaço no data center de computação. 30% de economia em eletricidade devido a ter menos discos. 30% de economia em ar condicionado para ter que arrefecer menos discos. 30% menos armazenamento de backup, como fitas. 30% menos risco de ter menos discos suscetíveis a falhas e que podem parar o sistema.

Comparando colunas com bancos de dados de armazenamento de linha, os bancos de dados relacionais. Podemos dizer que a coluna economiza recursos de disco, tempo, memória e CPU quando consultas OLAP complexas são necessárias. Bancos de dados colunares não são bons para gravações transacionais e consultas porque eles exigem para unir todas as colunas a fim de retornar um registro inteiro. Isso leva tempo e recursos. Agora podemos ver como implementar um banco de dados colunar. No caso do SAP, o Sybase IQ suporta SQL. Como podemos ver no seguinte código e diagrama, onde podemos criar tabelas de modelo multidimensional para OLAP e executar algumas consultas OLAP.

Como podemos ver, um banco de dados colunar é bom para OLAP. Podemos usar um banco de dados colunar para implementar um data warehouse para consultas analíticas e obter vantagens em desempenho, compressão, capacidade de disco, etc.

7.3.1 Cassandra

Apache Cassandra é um sistema de gerenciamento de banco de dados NoSQL distribuído livre e de código aberto projetado para lidar com grandes quantidades de dados em muitos servidores de commodities, fornecendo alta disponibilidade sem ponto único de falha.

O Cassandra oferece suporte a clusters, abrangendo vários data centers com replicação sem mestre assíncrona, permitindo operações de baixa latência para nossos clientes. É uma base de dados altamente escalável, possivelmente consistente e distribuído de estruturas de chave-valor e colunar. Foi iniciado pelo Facebook e é um projeto Apache de código aberto escrito em Java, e depois um banco de dados multi-plataforma, que é um banco de dados que pode ser implementado sob diferentes sistemas operacionais.

Algumas vantagens do Cassandra para desenvolvimento web são: Cassandra é desenvolvido para ser um servidor distribuído, mas também pode ser executado como um nó simples. A escalabilidade horizontal adiciona novo hardware quando necessário, respostas rápidas mesmo que a demanda cresça, altas velocidades de

gravação para gerenciar volumes de dados incrementais, armazenamento distribuído, capacidade de alterar a estrutura de dados quando os usuários exigem mais funcionalidade, uma API simples e limpa para o seu linguagem de programação favorita, detecção automática de falhas, não há nenhum ponto único de falha, o que significa que cada nó sabe sobre os outros, é descentralizado, tolerante a falhas e permite o uso do Hadoop para usar o Map Reduce.

Algumas desvantagens são que Cassandra pode ter alguns problemas com; consultas ad-hoc, o que significa que estes quando o usuário tem que alterar consultas para ser adequado às suas necessidades, agregações ou uso de funções agregadas SQL, desempenho imprevisível, por exemplo, cadeias no tempo de consulta resposta.

Cassandra usa um protocolo Gossip, que é uma comunicação interna, para permitir a comunicação dentro de um anel, de modo que cada nó saiba sobre outros nós. Permite apoiar a descentralização e tolerância à partição. Cassandra é projetado para ser distribuído em várias máquinas que aparecem como uma máquina simples para os clientes. A estrutura mais externa de Cassandra é um cluster ou anel. Um nó tem uma réplica para diferentes intervalos de dados. Se algo der errado, uma réplica pode responder. O parâmetro replication_factor na criação de um KeySpace indica quantas máquinas no cluster receberão cópias dos mesmos dados.

Cassandra está mais focada na disponibilidade e na tolerância a falhas do que na inconsistência. Portanto, de acordo com a Apache Software Foundation, permite uma eventual consistência dada em milissegundos. Então Cassandra é AP. Para suportar principalmente a disponibilidade e tolerância de partições, seu sistema pode retornar dados imprecisos, mas o sistema estará sempre disponível mesmo na frente de uma partição de rede.

Descendente para dados distribuídos escaláveis, a Cassandra sacrifica uma semente para obter vantagens de desempenho, disponibilidade e gerenciamento operacional. Portanto, precisamos aprender alguns conceitos para entender como Cassandra funciona.

Uma coluna é composta por um nome, valor e timestamp. Um cluster corresponde a algumas máquinas que compõem uma instância de Cassandra. Eles podem conter vários keyspaces. Um keyspace é um namespace para um conjunto de ColumFamily, associado a um aplicativo. Normalmente vinculado a um banco de dados no modelo relacional. Uma família Columfamily contém várias colunas. Eles geralmente são vinculados a uma tabela no modelo relacional. Uma Supercolumn é um conjunto de colunas que eles próprios têm sub-colunas. As supercolunas não têm timestamps ao contrário das colunas. Uma supercoluna é análoga a um registro ou parte superior de um banco de dados relacional.

Em Cassandra, a unidade básica de armazenamento é uma coluna, embora existam super colunas, famílias de colunas e o keyspace. Os timestamps armazenam a hora da última atualização da coluna e são usados para resolução de conflitos. Um nome de coluna é análogo a um nome de atributo em uma tabela em um banco de dados relacional. Uma supercoluna é composta por uma matriz de várias colunas. Ele é especificado com um nome e um mapa ordenado de colunas.

As colunas que pertencem a uma super coluna são agrupadas usando um valor de pesquisa comum chamado Chave Raw. Em outras palavras, uma super coluna é um par chave-valor ninho de colunas. Os pares de chave-valor externos formam uma super coluna, enquanto pares internos corresponde às colunas. A figura mostra a estrutura de uma supercoluna. Nossa família de colunas contém colunas ou super colunas agrupadas que usam uma única chave bruta comum. Ele pode ser visto como um conjunto de pares de valor chave, onde as chaves são chaves brutas, e os valores um mapa de nomes de colunas. A figura apresenta visualmente a estrutura de uma família de colunas. Bem, podemos descansar por enquanto.

Cassandra gerencia colunas e família de colunas. Há mais elementos para revisar, por exemplo, um keyspace é um contêiner que armazena dados dos usuários do aplicativo. Os keyspaces podem ter associado uma ou mais famílias de colunas, embora nem sempre seja necessário que eles tenham famílias de colunas.

Os keyspaces exigem que alguns atributos sejam definidos, como nomes definidos pelo usuário, estratégias de replicação e outros. Um keyspace é análogo a um banco de dados em um modelo relacional, mas sem inter-relações. Um keyspace é um esquema de alto nível que contém famílias de colunas. Nas redes sociais, o status do usuário pode ser armazenado em uma família de super colunas. As postagens no Twitter ou tweets são uma loja em uma família de colunas.

Esses espaços exigem configuração de acordo com a consistência. Os atributos básicos que você pode associar a um keyspace são: o fator de replicação que indica quanto você deseja pagar o desempenho em favor da consistência. A estratégia de posicionamento de réplica, que indica como as réplicas são colocadas no anel, como SimpleStrategy, OldNetwork TopologyStrategy e NetworkTopologyStrategy. Você pode ler sobre essas estratégias no link a seguir.

Deve haver pelo menos uma família de colunas por keyspace. Lembre-se de que uma família de colunas é um contêiner de linhas contendo colunas. Dentro Cassandra, como a maioria dos bancos de dados NoSQL, o projeto requer que as consultas são modelo primeiro e, em seguida, a estrutura de dados em torno deles é definida. Podemos observar como bancos de dados relacionais diferem de Cassandra em termos de armazenamento.

Cassandra tem uma linguagem de consulta chamada CQL, que significa Cassandra Query Language. CQL oferece um mais do que perto de CQL nos dados do sensor é colocar tabelas contendo linhas e colunas. Por esse motivo, ao usar este documento, essas tabelas de termos mostra e colunas têm a mesma definição que eles têm no CQL. Alguns dos recursos que o CQL tem são o uso de tipos de dados, definição de dados, manipulação de dados, índices secundários, views materializadas, segurança, funções, operações aritméticas, suporte JSON e triggers.

7.4. Bancos de dados orientados a Documentos

MongoDB é um banco de dados de documentos. O nome vem da palavra inglesa “humongous”, que significa “enorme”. MongoDB salva estruturas de dados em documentos do tipo BSON, JSON binário, com seu esquema dinâmico, tornando a integração de dados em determinadas aplicações mais fácil e rápida.

MongoDB suporta outras consultas, indexação e replicação. No caso de outras consultas, MongoDB suporta pesquisa por campos, classificação, consultas e expressões regulares. As consultas podem retornar um campo específico do documento, mas também podem ser uma função JavaScript definida pelo usuário.

No caso de indexação, qualquer campo em um documento MongoDB pode ser indexado, assim como índices secundários são possíveis. No caso de replicação, MongoDB suporta o tipo de replicação master-slave, o que significa que o mestre pode executar, ler e escrever comentários, e o escravo pode copiar os dados do mestre, e só pode ser usado para leitura.

No caso de falhas de serviço com o mestre atual, o escravo tem a capacidade de escolher um novo mestre. MongoDB é mais adequado para sistemas de informação, como comércio eletrônico ou e-commerce, jogos, aplicativos móveis e gerenciamento de conteúdo, como armazenamento de comentários, votos, registro de usuários, perfis de usuários e sessões de dados.

MongoDB salva dados em documentos do tipo JSON com um esquema dinâmico chamado BSON, o que implica que não há nenhum esquema predefinido. Os elementos dos dados são chamados de documentos e são armazenados em coleções. Uma coleção pode ter um número indeterminado de documentos.

```
{
        name:       “sue”,                  ←       campo: value
        age:        26,                     ←       campo: value
        status:     “A”,                    ←       campo: value
        groups:     [ “news”, “sports” ]    ←       campo: value
}
```

Coleções são como tabelas, e documentos como linhas. Cada documento em uma coleção pode ter campos diferentes. A estrutura de um documento é simples e composta por pares chave-valor, semelhantes às matrizes associativas em uma linguagem de programação. Como valor, você pode usar números, cadeias de caracteres ou dados binários como imagens ou quaisquer outros pares chave-valor.

No MongoDB, um documento é um conjunto de campos chave-valor na sintaxe JSON. A figura apresenta um exemplo visual de um documento típico do MongoDB. Como podemos ver a partir da imagem, MongoDB fornece um armazenamento muito flexível para o nome, idade e status, e grupos de interesses de uma pessoa.

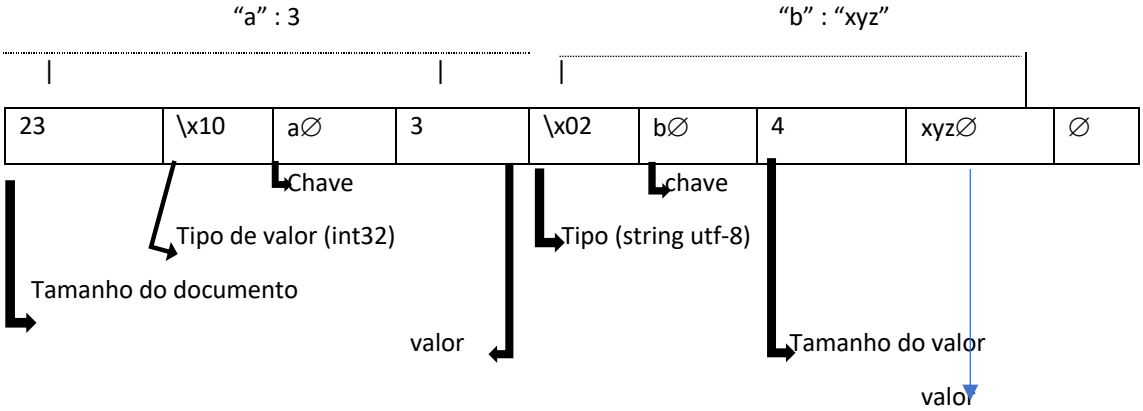
Também é possível crescer documentos e realizar correções, que estão ambos tendo que executar operações conjuntas caras. Os documentos são análogos à estrutura das linguagens de programação que associam chaves a valores, como tabelas de hash, dicionários e mapas.

Suponha que dois campos a e b, com valores três são xyz. A figura mostra um código JSON para atribuir valores a a e b, e as representações BSON correspondentes. Os documentos são armazenados em coleções, que são um conjunto de documentos relacionados que compartilham índices.

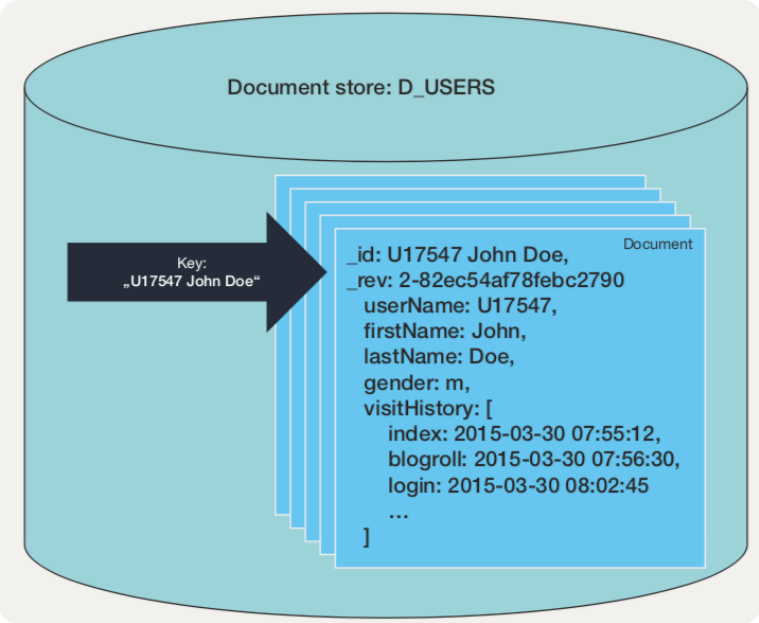
Json

```
{
“a”:        3,
“b”:        “xyz”
}
```

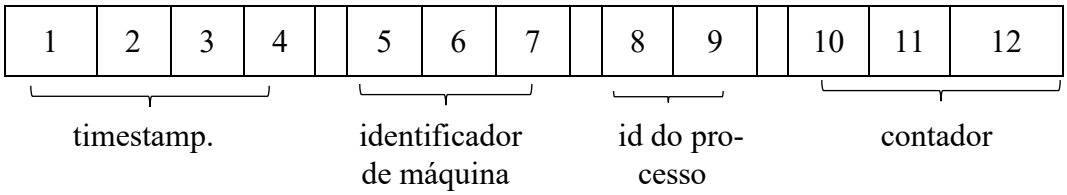
BSON



As coleções são análogas às tabelas do banco de dados relacional. A figura conceitualmente mostra uma coleção de MongoDB. O armazenamento de documentos oferece alto desempenho, escalabilidade de variáveis, alta flexibilidade e baixa complexidade de implementação.



Cada documento no MongoDB tem um ID. ID é o primeiro atributo do documento e o identificador interno está em valor hexadecimal. Neste exemplo, podemos ver que os primeiros quatro bytes são para um timestamp (data/hora), os próximos três bytes contêm o identificador correspondente, ao lado de polarização continha um identificador de processo, e os últimos três bytes são um contador.



MongoDB tem um número de mongo-tools. Alguns deles são brevemente explicados a seguir:

- Mongo binário é uma interface que permite aos desenvolvedores visualizar, inserir, excluir e atualizar dados em seu banco de dados. Ele também ajuda a replicar informações, configurar os fragmentos, desligar os servidores e executar JavaScript.
- Mongostat é uma ferramenta de linha de comando que resume uma lista de estatísticas de nossa instância MongoDB em execução.
- Mongotop é uma ferramenta de linha de comando para rastrear a quantidade de tempo que leva para ler ou gravar dados em uma instância.
- A ferramenta mongosniff fira tráfego de rede, indo e vindo do MongoDB.
- Os binários Mongo import e Mongo export são ferramentas de linha de comando que facilitam as operações de importação e exportação para o conteúdo de JSON, CSV ou TSV.
- Mongodump e mongorestore são ferramentas para criar uma exportação binária do conteúdo do banco de dados.

Práticas com MongoDB

Existem diferentes tipos de consultas que podem ser executadas no MongoDB.

- As **consultas de valor-chave** retornam resultados com base em um campo e um determinado valor.
- As **consultas de classificação** retornam resultados com base em desigualdades como maior que, menor que, igual a , entre dois valores, etc.

- As **consultas geoespaciais** retornam resultados com base na proximidade, como interseção e inclusão em um ponto , linha, círculo ou polígono.
- As **consultas de tipo texto** retornam resultados de acordo com palavras usando operadores booleanos como e, ou não.
- As **consultas de resumo** retornam resultados de médias , mínimos, máximos, totais, etc.
- **Consultas MapReduce** retornam resultados sobre os dados de todo o cluster quando as coleções são distribuídas.

Em uma tabela em um banco de dados relacional corresponde a uma coleção no MongoDB. Há uma instrução de inserção para alimentar o banco de dados Mongo. A instrução **find** permite obter informações de Mongo como selecionar em bancos de dados relacionais. Podemos estabelecer algumas condições, a fim de filtrar dados com **find** e a condição dentro de colchetes. Podemos limitar as saídas de consulta para um com a instrução find.

SQL	mongoDB
CREATE TABLE users (name VARCHAR(120), age NUMBER);	db.createCollection("users");
INSERT INTO users VALUES ("Bob", 32);	db.users.insert({name: "Bob", age: 32})
SELECT * FROM users	db.users.find({})
SELECT name, age FROM users	db.users.find({}, {name: 1, age: 1, _id:0})
SELECT name, age FROM users WHERE age = 33	db.users.find({age: 33}, {name: 1, age: 1, _id:0})
SELECT * FROM users WHERE age > 33	db.users.find({age: {$gt:33})
SELECT * FROM users WHERE age <=33	db.users.find({age: {$lte:33})

Os resultados são função distinta para obter valores de dados exclusivos. Esta tabela mostra as instruções mais comuns usadas no MongoDB, semelhantes ao SQL. Como você pode ver, há instruções para inserir, atualizar, excluir, criar Índice e mostrar o plano de consulta.

SQL	mongoDB
SELECT * FROM users LIMIT 1;	db.users.findOne()
SELECT DISTINCT name FROM users;	db.users.distinct ("name")
SELECT COUNT(*) FROM users	db.users.count()
SELECT COUNT(*) FROM users WHERE age >30	db.users.find({age: {$gt: 30}}).count()
SELECT COUNT(age) FROM users	db.users.find({age: {$exist: true} }).count()
UPDATE users SET age = 33 WHERE name = "Bob"	db.users.update({name: "Bob"}, {$set:{age:33} }, {multi: true})
UPDATE users SET age=33+2 WHERE name="Bob"	db.users.update({name: "Bob"}, {$inc: {age:2} }, {multi: true})
DELETE FROM users WHERE name = "Bob"	db.users.remove({name: "Bob"})
CREATE INDEX ON users (name ASC)	db.users.ensureIndex({name: 1})
CREATE INDEX ON users (name ASC, age DESC)	db.users.ensureIndex({name: 1, age: -1})
EXPLAIN SELECT * FROM users WHERE age =32	db.users.find({age: 32}).explain()

Embora MongoDB permita que dados binários sejam salvos em objetos BSON, seu limite de tamanho é 16 megabytes. GridFS é uma especificação para salvar arquivos grandes no MongoDB por um mecanismo para dividir de forma transparente um arquivo grande em vários documentos. Cada arquivo tem um objeto de

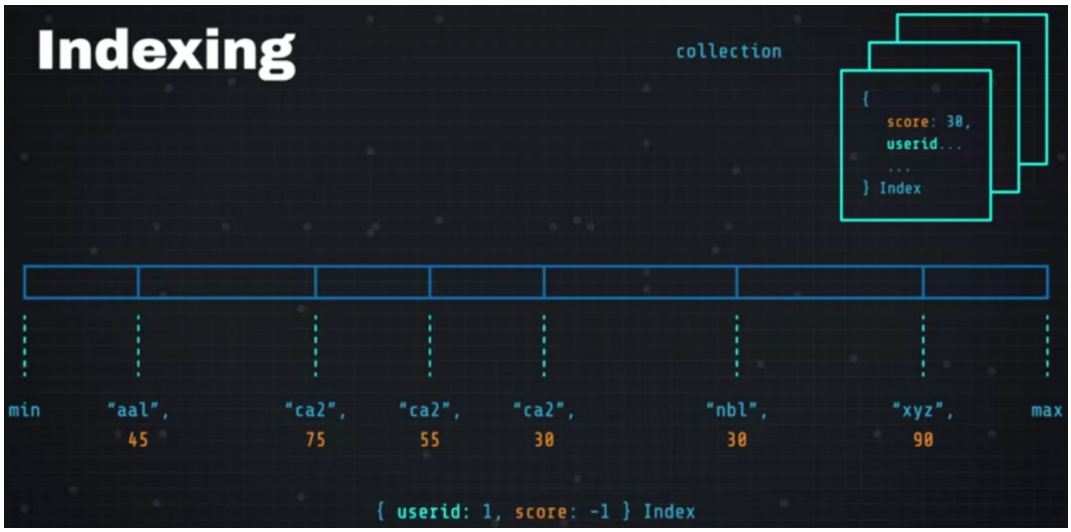
metadados na coleção de arquivos e um ou mais objetos de bloco na coleção de blocos. Se você precisa armazenar arquivos grandes no MongoDB, vá para o link 9mostrado para obter mais informações.

Índices no MongoDB permitem a rápida digitalização de documentos evitando o rote de toda a coleção usando um estruturas de dados especiais que armazenam apenas uma parte dele. Usando árvores B, o índice armazena o valor de um campo específico, ordem pelo valor desses campos. Os índices são definidos no nível de coleta e, se forem apropriados, MongoDB pode usá-los para limitar o número de documentos a serem inspecionados. Quando os critérios de pesquisa e a projeção de consulta incluem apenas os campos de índice, MongoDB pode retornar resultados diretamente do índice sem digitalizar qualquer outro documento. Um exemplo de índice e como usá-lo é mostrado na figura a seguir.

A consulta obtém um intervalo de documentos com valor de pontuação menor que 30 com os critérios de pesquisa. Uma projeção no MongoDB usa o valor um para campos que você deseja recuperar e zero para campos que você deseja excluir.

Índices compostos mantêm referências a vários campos dentro de uma coleção de documentos. A figura a seguir ilustra um exemplo de um índice composto por dois campos. Nesta figura, o índice é primeiro ordem crescente com base no campo userid e, em seguida, ordem decrescente pelo campo de pontuação. Índices compostos suportam consultas em qualquer prefixo dos campos de índice. Os prefixos de índice são o determinado subconjunto dos campos do conjunto de índices. Por exemplo, dado o seguinte índice. A sintaxe para criar um índice no Mongo shell é a seguinte. Onde **collection** é o nome da coleção, se o **valor 1** é atribuído, os valores do campo são ordenados **crescente** e se o **valor -1** a ordem é **decrescente**.

db.collection.ensureIndex ({field 1: 1 | -1, field 2: 1 | -1, … field n: 1 | -1 })



Aqui temos um exemplo de pesquisa de texto. Em primeiro lugar, inserimos cinco registros para as lojas de coleção, então precisamos criar um índice chamado texto e, finalmente, obtemos informações com encontrar indicando o índice de texto i procurando texto cafeteria Java que pesquisa.

```
db.stores.insert([
                { _id: 1, name: “Java Hut”, description: “Coffee and cakes” },
                { _id: 2, name: “Burger Buns”, description: “Gourmet hamburgers” },
                { _id: 3, name: “Coffee Shop”, description: “Just coffee” },
                { _id: 4, name: “Clothes Clothes Clothes”, description: “Discount clothing” },
                { _id: 5, name: “Java Shopping”, description: “Indonesian goods” }
        ] )
```

[9] http://www.mongodb.org/display/DOCS/GridFS

```
/*to allow text search over the name and description fields: */
db.stores.createIndex( { name: “text”, description: “text” } )
/* find all stores containing any terms from the list “coffee”, “shop”, and “java”: */
db.stores.find( {$text: { $search: “java coffee shop” } } )
```

Agora, vamos aprender como projetar e implementar um sistema de informação com um banco de dados de documentos. Existem várias estratégias para relacionar dados no MongoDB. A fim de projetar o banco de dados, de preferência decompõe a cardinalidade um-para-muitos em: **um para alguns**, **um para muitos** e **um para milhões**.

No caso de relacionamento **um para poucos**, use o modo incorporado ou desnormalizado. Um documento contém uma matriz de outros documentos. A vantagem é que podemos ler e gravar dados em um único acesso. A desvantagem é que ele não é escalável e não independente do acesso a dados incorporados. Aqui temos o nome, número de segurança social para uma pessoa Kate Monster e sua relação com dois endereços.

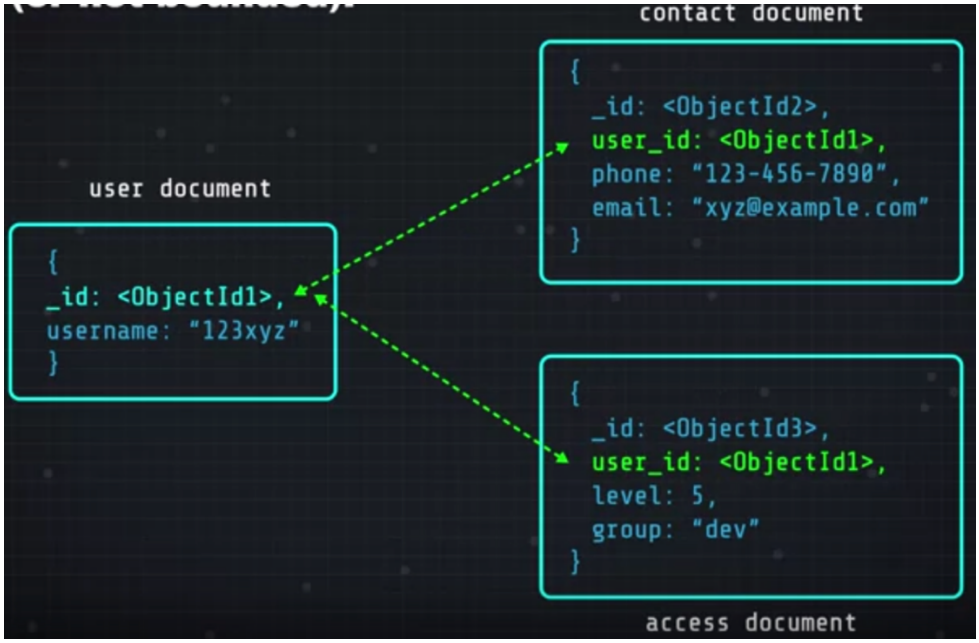
```
> db.person.findOne()
{
    name: “Kate Monster” ,
    ssn: “123-456-7890”,
    addresses: [
            {street: “123 Sesame St”, city: “Anytown”, cc: “USA” },
            {street: “123 Avenue Q”, city: “New York”, cc: “USA” }
    ]
}
```

No caso de uma relação **um-para-muitos**, use o modo de apontamento. Isso significa que os documentos contêm apontamentos com os IDs de outros documentos. A vantagem é que ele é moderadamente escalável. Existem leituras e atualizações independentes para ambos os documentos. Ele também suporta relacionamentos muitos-para-muitos. A principal desvantagem é que para associar dados, você tem que fazer uma junção manual que requer dois acessos ao banco de dados.

```
> db.parts.findOne()
{
    _id: ObjectID(“AAAA”) ,
    partno : “123-aff-456” ,
    name : “#4 grommet” ,
    qty : 94 ,
    cost : 0.94 ,
    price : 3.99
}
```

```
> db.products.findOne()
{
    name: “left-handed smoke shifter” ,
    manufacturer : “Acme Corp” ,
    catalog_number : 1234 ,
    parts : [  // array of references to Parts docs
      ObjectID(“AAAA”) , //ref to the #4 grommet
      ObjectID(“F17C”) ,
      ObjectID(“D2AA”)
      //etc
    ]
}
```

No caso da relação **um-para-milhões** ou não ligado, use o modo normalizado que é uma abordagem muito semelhante ao design em bancos de dados relacionais. Documentos filhos apontam para o ID do documento pai. A vantagem é que ele é altamente escalável. Ele permite leituras e atualizações independentes para ambos os documentos e também é apropriado para relacionamentos um-para-muitos. A desvantagem é que, para associar dados, você tem que fazer uma junção manual, o que requer dois acessos ao banco de dados.



A seguir mostra as principais características do MongoDB.

- MongoDB foi escrito em C ++
- Principal ponto: mantém algumas propriedades similares ao SQL (query e index)
- Licença AGPL (Apache)
- Protocolo: Customizado, binário (BSON)
- Replicação Master/Slave
- Fragmentação integrada
- Consultas são expressões em JavaScript
- Executa funções JavaScript no server side
- Usa arquivos mapeado na memória para armazenamento de dados
- Bom desempenho com funções
- Em sistemas 32 bits, limitaa a ~ 2.5GB
- Um banco de dados vazio ocupa 192MB
- GridFS para armazenamento de grandes volumes de dados + metadados
- Índice geospacial
- Melhor uso: se você necessitar de consultas dinâmicas. Se você preferir definir índices, não precisa de função Map/Reduce. Se você preferir bom desempenho em Banco de dados grandes.

7.5. Bancos de dados orientados a Grafos

Um grafo é representado como um conjunto de nós, que podem ser entidades interligadas por borda ou relacionamentos. Os grafos dão importância não apenas aos dados, mas também às relações entre nós. Relacionamentos também podem ter atributos e consultas diretas podem ser feitas para relacionamentos em vez de para os nós. Sendo armazenado dessa forma, é mais eficiente navegar entre relacionamentos do que em um modelo relacional. Obviamente, bancos de dados gráficos só são úteis quando as informações podem ser facilmente representadas como uma rede.

Entre as implementações de banco de dados de grafo mais utilizadas estão Neo4j, Hyperbase-DB e InfoGrid. O desempenho e a escalabilidade de um banco de dados orientado a gráficos são variáveis e altamente

complexos de implementar. No entanto, um banco de dados de grafo é altamente flexível em termos de uma estrutura.

Um exemplo deste gerenciador de banco de dados é Neo4j. Este tipo de banco de dados armazena grafos, e como já dissemos antes, um grafo é uma das estruturas de dados mais genéricas e flexíveis. Um grafo armazena dados em nós, que têm propriedades. Os nós são organizados em relacionamentos, que também possuem propriedades. Um rótulo agrupa nós em relacionamentos e/ou opção.

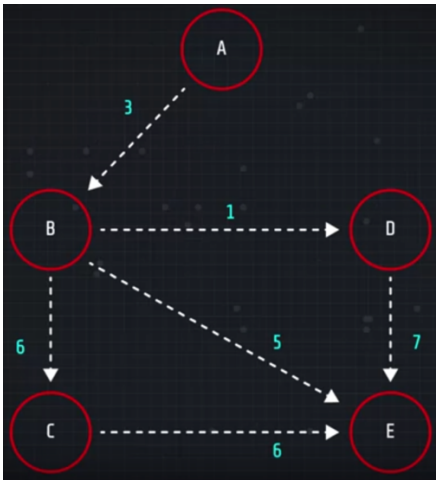
Uma busca navega em um grafo, que identifica caminhos que ordenam os nós. Índices mapeiam propriedades para nós ou relacionamentos. Um banco de dados de grafo gerencia um grafo e seus índices. Neo4j usa grafos para representar dados e as relações entre eles. Um grafo é formado por vértices ou denotado por círculos, e bordas mostradas por linhas de interseção.

Temos vários tipos de grafos.

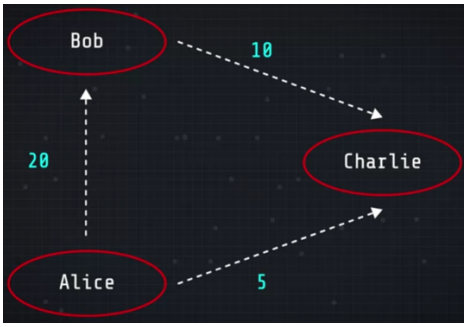
Grafos **não direcionados**, onde nós e relacionamentos são intercambiáveis, e a relação pode ser interpretada em qualquer sentido. Por exemplo, amizade, relacionamentos na rede social, Facebook.

Grafos direcionados, onde nós e relacionamentos não são bidirecionais por padrão. Por exemplo, relacionamentos no Twitter onde um usuário pode seguir determinados perfis sem que ele o siga.

Grafos ponderados, onde as relações entre nós têm algum tempo de avaliação numérica.



Grafos rotulados são usados quando diferentes vértices e relações entre eles são definidos e identificados por rótulos. Um exemplo de grafos rotulados é no Facebook onde podemos ter nós que encontramos por termos como amigo ou colega de trabalho e as relações como amigo ou parceiro.



O último tipo de grafos são **grafos de propriedade**, é um grafo com peso, com rótulos e onde podemos atribuir propriedades para ambos os nós e relacionamentos. Por exemplo, nome, idade, país de residência, nascimento. É um mais complexo porque contém várias das características que temos que cobrir antes como rótulos, peso, direções etc.

O grafo a seguir representa os principais conceitos que temos revisado até agora. Um banco de dados orientado a grafos é especialmente poderoso quando queremos dados conectados móveis. Ele usa um modelo baseado no grafo composto por nós e relações com propriedades. Qualquer domínio pode ser representado como grafo e, portanto, gerenciado por Neo4j.

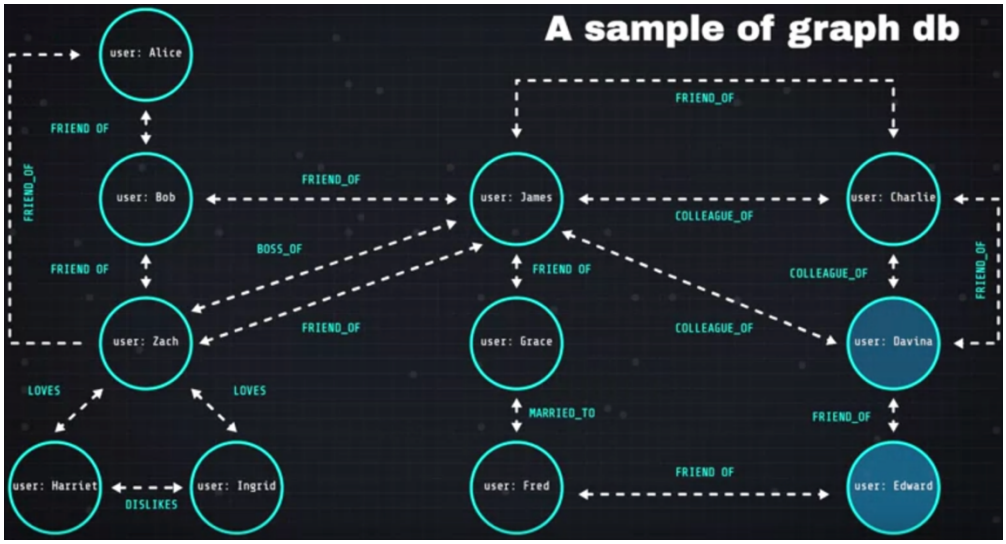
Neo4j é um banco de dados grafo que permite alto desempenho e alta disponibilidade durante as operações de leitura. Neo4j é um banco de dados sólido que dá suporte real para transações ACID, que é característica muito importante que os bancos de dados NoSQL podem oferecer. Este servidor é utilizável como uma biblioteca Java ou tem uma API REST. Uma API REST é um serviço web que significa Transferência de Estado

Representacional e usa HTTP como um meio para se comunicar com a fonte de dados e retornar recurso na forma de JSON.

7.5.1 Banco de dados Neo4J

Neo4j é excelente para modelar dados complexos e conectados. Para recomendações, inteligência de espécies, computação social, dados geoespaciais, administração de sistemas, logística, séries temporais e muito mais.

Aqui temos uma amostra de um banco de dados gráfico. Alice é amiga de Bob, que é amigo de Zach, e Zach também é amigo de Alice. Zach ama Harriet e Ingrid. Ingrid e Harriet não gostam uma da outra. Bob é amigo de James. James é o chefe e amigo do Zach. James é amigo de Grace, que é casado com Fred. James e Charlie são colegas e amigos. Charlie é colega de Davina, e Davina é amiga de Edward, que é amigo de Fred.



Para instalar e executar Neo4j, você precisa baixar neo4j-community do site. Descompacte e instale o arquivo no disco rígido. Inicie o servidor de acordo com seu sistema operacional e, em um navegador, vá para seu webadmin localhost, com 7474 como padrão. O navegador inclui Dashboard, navegador de dados, console, etc E também você pode usar REST para ir para o nó raiz, por exemplo.

Neo4j usa **cypher** como sua linguagem de consulta. **Cypher** é uma linguagem formal usada para preencher banco de dados e fazer solicitações. Ele usa índices e traversals, o que significa percorrer os nós do gráfico para obter dados.

Existem algumas cláusulas para criar e preencher seu gráfico de banco de dados.

A cláusula **create** cria nós e relacionamentos.

A cláusula **delete** exclui elementos de gráfico, nós, relacionamentos ou caminhos. Qualquer nó a ser excluído também deve ter todos os relacionamentos associados explicitamente excluídos.

detach delete, exclui um nó ou conjunto de nós. Todos os relacionamentos associados serão excluídos automaticamente.

set atualiza rótulos em nós e propriedades em nós e relacionamentos.

remove remove propriedades e rótulos de nós e relacionamentos.

foreach atualiza dados dentro de uma lista, sejam componentes de um caminho ou o resultado da agregação.

Para resolver consultas, temos as sentenças **Match** and **Where**.

A cláusula **match** especifica um padrão de nós e relacionamentos.

Where é a subcláusula para adicionar restrições para padrões em cláusulas **match** ou **optional match** ou para filtrar os resultados de uma cláusula **with**.

Considere o grafo a seguir. Criaremos dois nós, criamos frases onde Emil e Johan são duas pessoas que nasceram na Suécia. Também podemos criar a relação entre Emil e Johan com a frase de criação. No entanto, precisamos chegar a instâncias pessoais para Emil e Ian com correspondência e onde frases como condições, e, em seguida, criar a relação de nós entre Emil e Ian. Existem muitas linguagens de programação suportadas.

```
// criando os nós
CREATE (a:Person {name: “Emil”, from: “Sweden”, klout:99 })
CREATE (b:Person {name: “Johan”, from: “Sweden”, learn: surfing })

// criando relacionamentos
MATCH (a:Person), (b:Person)
WHERE a.name = “Emil” AND b.name = “Johan”
CREATE (a) - [r:KNOWS { since:2001 } ] -> (b)
RETURN type(r) , r.name
```

Você pode dar uma olhada no seguinte link para REST[10] ou API Java[11].

Podemos usar linguagem declarativa para percorrer um gráfico. Por exemplo, se você precisar procurar por uma pessoa chamada Emil, a seguinte consulta deve ser enviada.

```
MATCH (n:Person)
where n.name = “Emil”
return n;
```

A cláusula de **match** é especificar um padrão de nós e relacionamentos, como o seguinte, que é um padrão de nó único com uma pessoa de rótulo que irá atribuir correspondências à variável n.

A cláusula **where** é a restrição dos resultados. n.name = “Emil” permite comparar a propriedade do nome com o valor “Emil”.

[10] http://www.neo4j.org/develop/drivers e http://docs.neo4j.org/chunked/milestone/rest-api.html
[11] http://docs.neo4j.org/chunked/milestone/tutorials-java-embedded.html

E a cláusula **RETURN** é usada para solicitar um resultado específico.

A principal razão para usar **índices** em um banco de dados de grafo é encontrar o ponto de partida no gráfico o mais rápido possível. Depois disso, você confia em estruturas e relacionamentos in-graph para alcançar alto desempenho. Essas consultas de gráfico por si só não precisam de índices para executar rapidamente. Os índices podem ser adicionados a qualquer momento. Observe que levará algum tempo para um índice ficar online quando houver dados existentes.

Neste caso, queremos criar um índice para acelerar a busca de pessoas pelo nome no banco de dados.

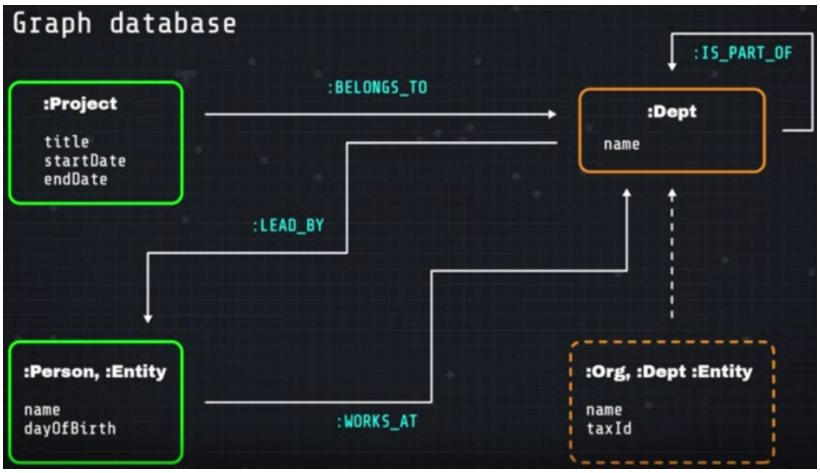
```
CREATE INDEX ON :Person(name)
```

Criamos o índice em: pessoa (nome).

Normalmente, você não especifica índices ao consultar dados. Eles serão usados automaticamente. Isso significa que podemos simplesmente procurar o nó Emil com MATCH. E o índice vai aparecer nos bastidores para aumentar o desempenho.

```
MATCH (n:Person { name: “Emil” } )
RETURN n;
```

No caso de relação e bancos de dados se você usar muitos para muitos relacionamentos você tem que introduzir uma tabela conjunta que contém chaves estrangeiras de ambas as tabelas participantes. Aumentar os custos de operação conjunta.



Cada nó no modelo de banco de dados de grafo diretamente e fisicamente contém uma lista de registros de relacionamentos que representam cada relacionamento com outros nós. Esses registros de relacionamentos Organizar por tipo e direção. Posso ter atributos adicionais. Sempre que você executar o equivalente a uma operação conjunta, o banco de dados apenas usa essa lista e tem acesso direto aos nós conectados. Eliminando a necessidade de uma pesquisa cara ou de muita computação.

Agora você pode definir consultas em execução para um grafo de exemplo chamado banco de dados de filmes. O modelo é composto por notas, filme e pessoa, e relações atuam, dirigidas e produzidas. Baixe o banco de dados do filme no link[12].

Modelo de dados:

```
(:Movie {title, released, ... })
(:Person {name, born, ... } )
```

[12] http://neo4j.com/developer/movie-database/#_the_dataset

(:Person) - [:ACTED_IN | :DIRECTED | :PRODUCED] -> (:Movie)

Instruções de Instalação do banco:

- Pare o servidor Neo4j e veja o arquivo de download.
- Substitua graph.db no caminho para os dados Neo.
- Inicie o servidor novamente com bin/neo4j start.
- Abra a interface web Neo4j no link http://localhost:7474
- E execute as seguintes consultas de exemplo e mostre os resultados.

Como um resumo de neo4j, podemos dizer que é um banco de dados orientado a grafos. Com total conformidade ACID, incluindo dados de longa duração, com nós como relacionamentos que podem ter metadados. Sua linguagem de consulta é chamado Cypher, e é baseado na correspondência de padrões integrados. Neo é melhor usado para dados de estilo grafo, ricos ou complexos, interligados. Por exemplo, para procurar estradas em relações sociais, ligações de transferência pública, mapas rodoviários ou portos de água destes.

7.6. Aplicações Intensivas em Dados

Vamos entender as diferentes arquiteturas fornecidas para projetar aplicativos confiáveis, sustentáveis e escaláveis com uso intensivo de dados. Os aplicativos com uso intenso de dados armazenam dados para que eles ou outro aplicativo possam encontrá-los novamente mais tarde em um banco de dados.

Os aplicativos colocam o resultado de uma operação dispendiosa na memória cache para acelerar as taxas. Eles permitem que os usuários pesquisem dados por palavra-chave ou filtrá-los de várias maneiras. Eles podem enviar uma mensagem para outro processo para ser tratado de forma assíncrona pelo processamento de fluxo e periodicamente triturar uma grande quantidade de dados acumulados através do processamento em lote.

Um exemplo de uma aplicação intensiva de dados pode ser uma experiência de jogo online. Este aplicativo é atribuído para gerenciar vários milhares de usuários simultâneos e pode escalar horizontalmente em vários pontos conforme necessário.

Sistemas tradicionais de gerenciamento de banco de dados foram utilizados para aplicativos com uso intenso de dados. No entanto, como os requisitos do sistema, volume e disponibilidade de dados estão aumentando a tarefa não é tão simples. Além disso, existem várias abordagens para o cache, várias maneiras de construir índices de pesquisa e assim por diante.

Ao construir um aplicativo, ainda precisamos descobrir quais ferramentas e quais abordagens são as mais apropriadas para a tarefa em questão e pode ser difícil combinar ferramentas quando você precisa fazer algo que uma única ferramenta não pode fazer sozinho. Portanto, é importante ter em mente algumas perguntas que devem ser respondidas ao projetar sistemas com uso intensivo de dados. Por exemplo:

- Como você garante que os dados permaneçam corretos e completos, mesmo quando as coisas dão errado internamente?
- Como você fornece um desempenho consistente aos clientes, mesmo quando partes do seu sistema estão degradadas?
- Como você escala para lidar com um aumento na carga?
- Como é uma boa API para o serviço?

Há muitos fatores que podem influenciar o projeto de um sistema de dados, como habilidades e experiência das pessoas envolvidas, dependências de sistemas legados, a escala de tempo para entrega, a tolerância de sua organização a diferentes tipos de risco, restrições regulatórias etc. Esses fatores dependem muito da situação.

Um aplicativo com uso intenso de dados deve ser confiável, escalável e mantido. Mesmo no caso de um incremento de carga de trabalho. A confiabilidade é uma característica importante dos aplicativos com uso intenso de dados. Confiabilidade é quando o sistema deve continuar a funcionar corretamente mesmo em face da adversidade. O aplicativo executa funções esperadas pelo usuário. Ele pode tolerar erros do usuário ou a execução de software de maneiras inesperadas. Seu desempenho é bom o suficiente para o caso de uso necessário sob cargas esperadas e volume de dados. O sistema impede qualquer acesso não autorizado.

No caso de escalabilidade à medida que um sistema cresce em volume de dados, volume de tráfego ou complexidade, deve haver maneiras razoáveis de lidar com esse crescimento. Escalabilidade é a capacidade de um sistema para lidar com o aumento da carga. Discutir escalabilidade significa considerar questões como, se o sistema crescer de uma maneira particular, quais são nossas opções para lidar com o crescimento? Como podemos adicionar recursos de computação para lidar com a carga adicional? Um aplicativo de dados intensivo deve funcionar bem no caso de aumentar a carga de trabalho. A carga de trabalho depende da arquitetura do sistema. Por exemplo, o número de solicitações por segundo para o nosso servidor web, a proporção de leituras para gravações em um banco de dados, o número de usuários ativos simultaneamente em uma sala de bate-papo ou a taxa de acertos em um cache ou qualquer outra coisa.

Um aplicativo com uso intensivo de dados deve ser mantido. Quero dizer, ao longo do tempo, muitas pessoas diferentes trabalharão no sistema, engenharia e operações, ambos mantêm o comportamento atual e adaptando o sistema a novos casos de uso, e todos eles devem ser capazes de trabalhar nele de forma produtiva.

Existem três princípios de design de sistemas de software que os ajudam a ser mantidos.

- Operabilidade, é quando uma equipe de operação torna mais fácil manter o sistema funcionando sem problemas.
- Simplicidade, torna mais fácil para novos engenheiros entender o sistema removendo o máximo de complexidade possível do sistema. No caso de operabilidade, uma boa equipe de operações deve ser responsável pelo seguinte e muito mais.
- Evolubilidade, facilitam os engenheiros a fazer alterações no sistema no futuro, adaptando-o a casos de uso imprevistos à medida que os requisitos mudam. Monitorando a integridade do sistema e restaurando rapidamente o serviço se ele entrar em um estado ruim. Rastreando a causa de problemas como falhas do sistema ou desempenho degradado. Manter o software e as plataformas atualizados, incluindo patches de segurança. Manter controle sobre como diferentes sistemas afetam uns aos outros, de modo que uma mudança problemática possa ser evitada antes que ela cause danos. Antecipar problemas futuros e resolvê-los antes que ocorram.

Aplicativos de software devem ser tão simples quanto possível. Pequenos projetos de software podem ter código deliciosamente simples e expressivo, mas à medida que os projetos se tornam maiores, eles geralmente se tornam muito complexos e difíceis de entender. Essa complexidade retarda todos os que precisam trabalhar no sistema. Aumentar o custo de manutenção.

Um aplicativo deve estar preparado para evoluir porque os requisitos do sistema mudam o tempo todo. Como, novos fatos, casos de uso anteriormente imprevistos emergem, prioridades de negócios mudam, usuários solicitam novos recursos, novas plataformas substituem plataformas antigas, mudança de requisitos legais ou regulamentares, crescimento do sistema força mudanças arquitetônicas, etc.

7.6.1 Diferentes Sistemas de Informação

Vamos verificar como os diferentes sistemas de informação devem ser implementados para serem confiáveis, sustentáveis e escaláveis. Agora, detalharei alguns problemas relacionados a aplicativos transacionais com dados estruturados, aplicativos analíticos com dados estruturados, aplicativos analíticos com

dados não estruturados e aplicativos de streaming com alto volume de dados não estruturados em tempo real. Explicarei como esses sistemas de informação devem ser implementados para serem confiáveis, sustentáveis e escaláveis.

No caso de aplicativos transacionais com dados estruturados, eles podem ser implementados sob diferentes arquiteturas. Em primeiro lugar, eles podem residir em um sistema de banco de dados relacional paralelo. Lembre-se, que o paralelismo divide um grande problema em muitos menores para serem resolvidos em paralelo. Isso é útil em aplicativos que precisam processar um número extremamente grande de transações por segundo da ordem de milhares de transações por segundo. Podemos ver uma arquitetura típica de um sistema de banco de dados paralelo.

Em segundo lugar, os aplicativos transacionais são implementados em um cluster. Eles podem implicar conjuntos separados de nós para clientes, servidores de banco de dados e processamento SQL, bem como servidor dedicado e software cliente para tarefas de gerenciamento. Essa arquitetura aumentou drasticamente a capacidade de processamento líquido disponível, reduz custos, possibilita um equilíbrio mais equilibrado e oferece gerenciamento de dados mais escalável e confiável.

Em terceiro lugar, esses aplicativos transacionais podem ser implementados em um banco de dados relacional na memória. No caso de aplicações analíticas onde os dados estruturados que podem ser implementados em um banco de dados colunar, em banco de dados in-memory ou em um banco de dados colunar in-memory.

No caso de um banco de dados colunar, o SAP Sybase IQ tem sido usado para business intelligence, data warehousing e dados marcados. Cada tabela relacional original é dividida em colunas, cada coluna é iniciada em seu próprio segmento. Cada segmento é comprimido individualmente, o que resulta em alta taxa de compressões. Chamar sucesso em apenas algumas colunas em contraste com uma consulta estrela selecionada não precisa se esquivar de segmentos de outras colunas. Um banco de dados na memória dentro de um sistema de gerenciamento de banco de dados é aquele que depende principalmente da memória principal para armazenamento de dados do computador.

Na memória, os bancos de dados são mais rápidos do que os bancos de dados do disco de extinção porque o acesso ao disco é mais lento do que o Os algoritmos de otimização interna são mais simples e executam menos instruções da CPU. O acesso a dados na memória elimina o tempo de busca ao consultar os dados, o que proporciona um desempenho mais rápido e previsível do que o disco.

No caso de aplicações analíticas, pode ser colunar. Se queremos o melhor de ambas as partes, podemos implementar aplicação analítica com dados estruturados em um banco de dados colunar in-memory. O banco de dados em memória colunar precisa de menos memória principal, onde é armazenamento quente do que um banco de dados tradicional usa para esses cache na memória principal. SAP hana dá a capacidade de manter os dados na memória ou no disco em um formato colunar. Os dados não são duplicados. A noção de diário dinâmico ajuda os usuários a escolher memória para dados quentes e disco para dados quentes, ajudando a atingir o equilíbrio de desempenho de preço certo. Nesse caso, precisamos analisar não só a estrutura, mas também os dados não estruturados, um sistema de banco de dados de um modelo não seria a melhor opção.

Em primeiro lugar, um banco de dados multimodal in-memory pode armazenar e gerenciar qualquer tipo de dados. Podemos ver no vídeo a seguir uma arquitetura SAP hana, onde qualquer tipo de dispositivo para aplicativos são integrados, suportando dados espaciais, texto , Hadoop, dados não estruturados, dados transacionais, etc

Em segundo lugar, podemos usar uma estrutura Hadoop para suportar big data e data mining. Apache Mahout fornece análise através de clustering, classificação ou tarefa de mineração de dados recomendação, a fim de analisar big data no HDFS. Isso exigirá streaming de aplicativos analíticos com alto volume ou dados

não estruturados em tempo real, então precisamos analisar se a arquitetura realmente se adapta às nossas especificações. Por exemplo, se considerarmos usar o Hadoop para big data e SAMOA, yarn, Spark ou qualquer outro processador de dados de streaming para suportar streaming de dados, precisamos verificar se o processo não foi um tempo de resposta de falha, degradando a expectativa em tempo real.

Outra arquitetura que pode suportar os aplicativos de streaming com alto volume de dados não estruturados em tempo real é um banco de dados multi-modelo in-memory, que pode armazenar e gerenciar gráficos, documentos, dados espaciais, colunas e dados de estrutura. Esta semana, aprendemos diferentes arquiteturas para projetar aplicativos escaláveis, sustentáveis e pouco confiáveis com uso intensivo de dados.