FUNDAMENTOS DE BIG DATA

Maycon Viana Bordin

SAGAH
SOLUÇÕES EDUCACIONAIS INTEGRADAS

Hive: arquitetura e componentes

OBJETIVOS DE APRENDIZAGEM

Ao final deste texto, você deve apresentar os seguintes aprendizados:

- Reconhecer as dificuldades no desenvolvimento das soluções MapReduce.
- Explicar a arquitetura do Hive e a sua integração com os componentes do Hadoop.
- Descrever o processo de análise de dados por meio do Hive.

Introdução

Analisar grandes volumes de dados no Hadoop pode ser desafiador. É preciso ter conhecimentos em programação, além de estar familiarizado com a interface de programação de aplicações (API, do inglês *application programming interface*) do MapReduce e o seu funcionamento. O Hive simplifica a análise de dados no Hadoop, ao permitir o uso da Linguagem de Consulta Estruturada (SQL, do inglês Structured Query Language) para a realização de consultas em grandes volumes de dados.

Neste capítulo, você vai compreender as dificuldades no desenvolvimento de aplicações em MapReduce e como o Hive surgiu para facilitar essa tarefa. Você também vai aprender sobre a arquitetura do Hive e como ela se integra com os componentes do Hadoop. Por fim, você vai estudar o funcionamento do processo de análise de dados com o Hive.

Desenvolvimento com MapReduce

O **MapReduce** foi desenvolvido pelo Google e é um modelo de programação e uma implementação para processamento e geração de grandes *datasets* (DEAN; GHEMAWAT, 2008). Programas escritos nesse modelo funcional são automaticamente paralelizados e podem ser executados em grandes *clusters* de máquinas *commodity*.

O MapReduce é influenciado pela programação funcional e, como o nome sugere, é composto por duas fases: a fase de *map* e a fase de *reduce*. Ambas as fases possuem como entrada e saída pares de chave/valor. Fica a cargo do desenvolvedor especificar as funções de *map* e de *reduce*, bem como o tipo de dados para os pares de chave/valor em cada fase.

Na Figura 1, é possível visualizar uma aplicação de *wordcount*, que é uma aplicação com propósito demonstrativo muito utilizada em *frameworks* de *big data*. Ela consiste em contar o número de ocorrências de cada palavra contida em um *dataset* textual. A entrada da aplicação é um *dataset* com texto; a etapa de *map* vai receber pares de chave/valor compostos pelo número da linha como chave e o texto da linha como valor. A função de *map* vai quebrar o texto em palavras e enviar como saída pares de chave/valor compostos pela palavra como chave e um contador com valor 1.

Entre a etapa de *map* e *reduce*, ocorre uma operação chamada de *shuffle*. Essa etapa, de acordo com White (2015), é responsável por ordenar as saídas do *map* para que elas possam ser entregues para a etapa de *reduce*. O processo em si é mais complexo e envolve o particionamento das saídas do *map* para cada tarefa de *reduce*, bem como a cópia desses dados para as máquinas onde as tarefas de *reduce* vão executar o ordenamento desses dados. Quando os dados de todas as tarefas de *map* forem copiados, a tarefa de *reduce* pode iniciar.

Na etapa de *reduce*, a entrada são pares de chave/valor compostos pela palavra como chave e a lista de valores como valor. A função de *reduce*,

então, soma os valores e emite um novo par, com a palavra como chave e a soma dos valores como valor. No final, a saída será a frequência de todas as palavras do texto.

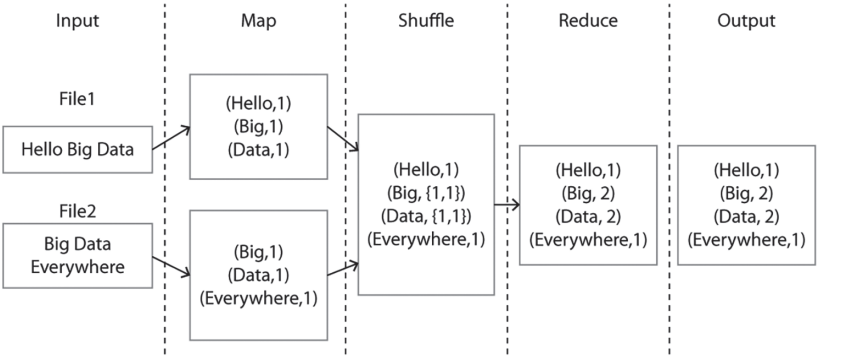

Figura 1. Exemplo do MapReduce para aplicação de *wordcount*.
Fonte: Adaptada de Siddesh, Hiriyannaiah e Srinivasa (2014).

Para White (2015), programas em MapReduce são simples, mas não o suficiente para permitirem a expressão de aplicações úteis. Em outras palavras, para desenvolver uma aplicação de análise de dados que seja útil, muito provavelmente será preciso criar diversos programas em MapReduce encadeados, cada um se alimentando do resultado do programa anterior. Esse processo é chamado de *workflow* e pode ser desenvolvido utilizando-se a API do MapReduce no Hadoop. Existem casos mais complexos, em que as dependências entre *jobs* formam uma DAG (*directed acyclic graph*), e é preciso informar as dependências existentes entre os *jobs*. Nesses casos mais complexos, é possível utilizar a API JobControl do Hadoop, ou utilizar orquestradores externos, como o Apache Oozie ou o Apache Airflow.

No intuito de facilitar o uso do poder do MapReduce com o Hadoop, diversas ferramentas foram desenvolvidas para abstrair as complexidades de programas MapReduce por meio de linguagens de alto nível. Dentre essas ferramentas, as mais notáveis são: Hive, Pig, Cascading, Crunch e Spark.

De volta à aplicação de *wordcount*, o exemplo oficial da documentação do Apache Hadoop (2020) é composto por 133 linhas de código escrito em Java. Em comparação, para contar palavras com o **Hive**, são necessárias apenas quatro linhas de código em Hive QL.

Exemplo

```
SELECT word, count(1) AS count FROM
   (SELECT explode(split(line, ' ')) AS word FROM
FILES) w
   GROUP BY word
   ORDER BY word;
```

Esse exemplo demonstra a simplicidade com que consultas podem ser feitas com o Hive, em comparação com o MapReduce. Entretanto, é importante ressaltar que sempre haverá casos de uso em que a programação será necessária, como em programas de aprendizado de máquina, por exemplo.

Fique atento

Atualmente, existem diversas alternativas ao MapReduce. As mais utilizadas dão suporte a linguagens similares ao SQL (também chamado de SQL-On-Hadoop), como é o caso do Presto, do Cloudera Impala, do Apache Drill e do Spark SQL, além de serem compatíveis com o ambiente Hadoop.

Arquitetura do Hive

O Hive foi criado pelo Facebook para permitir a análise de dados por meio do Hadoop, utilizando a linguagem SQL. Nas palavras dos criadores do *framework*, trata-se de uma ferramenta de *data warehouse* em cima do Hadoop. O Hive transforma consultas SQL em *jobs* que serão executados no *cluster* do Hadoop. Os dados são organizados em tabelas, permitindo, dessa forma, definir uma estrutura para os dados armazenados em sistemas de arquivos distribuídos, como o Hadoop Distributed File System (HDFS). Outros exemplos são: Amazon S3, Azure Data Lake Storage, Google Cloud Storage.

A Figura 2 ilustra os componentes do Hive, bem como a relação entre esses componentes. O Hive fornece três interfaces diferentes para que clientes se conectem, descritas a seguir.

- **Thrift:** permite a interação com o Hive utilizando qualquer linguagem de programação com suporte a esse protocolo.
- **Java Database Connectivity (JDBC):** fornece um *driver* JDBC de tipo 4 (Java puro) para conexão com o servidor do Hive. O *driver* utiliza uma implementação do Thrift para Java para se comunicar com o Hive.

- **Open Database Connectivity (ODBC):** permite que aplicações com suporte ao protocolo se conectem ao Hive. Também utiliza o protocolo Thrift para se comunicar com o Hive.

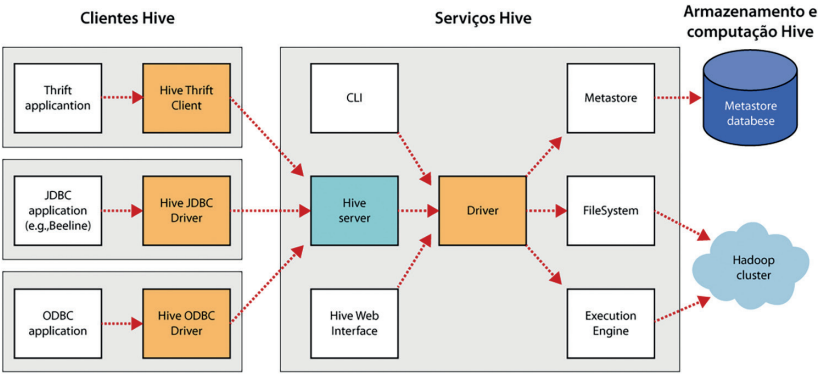

Figura 2. Componentes do Apache Hive.
Fonte: Adaptada de White (2015).

Os clientes se conectam ao servidor Hive (HiveServer2), o que permite, dessa forma, que eles submetam consultas para execução. Ele tem suporte à concorrência multicliente e à autenticação e executa como um serviço único composto por um serviço Hive baseado em Thrift (TCP ou HTTP) e um servidor *web* Jetty para a interface gráfica do Hive (APACHE HIVE, 2019).

A Figura 3 mostra em detalhes como é o fluxo de interação entre os componentes do Hive. Nela, é possível visualizar o *driver*, que é o componente que recebe as consultas e fornece APIs para execução e obtenção de resultados nos moldes das interfaces JDBC/ODBC. Ele envia a consulta para o compilador, que faz o *parse* da mesma, além de realizar uma análise semântica dos blocos de consulta e expressões. Por fim, o compilador gera um plano de execução com a ajuda dos metadados da tabela e das partições obtidos na *metastore*. Com o plano de execução em mãos, o *driver* o envia para a *engine* de execução.

A *metastore* é o repositório central de metadados do Hive e é dividida em duas partes: o serviço e o armazenamento dos dados. Por padrão, o serviço executa na mesma máquina virtual Java que o serviço do Hive e utiliza o banco de dados embutido Derby, para armazenamento dos metadados no disco local. O Hive suporta qualquer banco de dados com suporte ao JDBC para ser utilizado no armazenamento dos metadados, como o MySQL e o PostgreSQL. Na *metastore*, ficam armazenadas informações sobre (APACHE HIVE, 2019):

- *database*, que é um *namespace* para tabelas;
- tabelas, que contêm lista de colunas, *owner*, informações sobre o armazenamento (formato, local, compressão) e informações para serialização e deserialização (*serializer-deserializer*, ou SerDe); e
- partições, já que cada partição pode ter suas próprias colunas e informações de SerDe e armazenamento, permitindo que o esquema da tabela evolua sem afetar partições antigas.

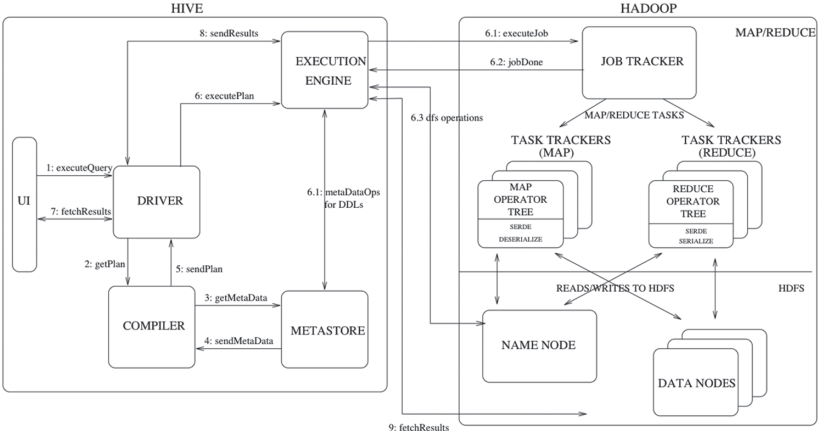

Figura 3. Fluxo de interação entre os componentes do Apache Hive.
Fonte: Apache Hive (2019, documento *on-line*).

Já a *engine* de execução é responsável por executar o plano de execução criado no compilador. Esse plano é uma DAG em estágios, e as dependências entre esses estágios são gerenciadas pela *engine* de execução, além de ela ser responsável pela execução dos mesmos. O Hive havia sido criado para utilizar o MapReduce como sua *engine* de execução, mas, atualmente, também é possível utilizar o Apache Tez e o Apache Spark como *engines* de execução. Ambos são *engines* genéricos, para execução de DAGs que oferecem maior flexibilidade e desempenho do que o MapReduce (WHITE, 2015).

Além dos componentes listados, existe ainda o otimizador, responsável por realizar mais transformações no plano de execução, com o intuito de melhorar o desempenho das consultas. No início, ele era um componente baseado em regras, como poda de colunas e *pushdown* de predicados (em-

purrar partes do SQL para serem executadas onde o dado está armazenado). Novas otimizações foram adicionadas ao longo do tempo, inclusive o uso de otimizações baseadas em custo (APACHE HIVE, 2019).

Long Live and Process (LLAP)

Long Live and Process (LLAP) é uma funcionalidade que foi adicionada no Hive 2.0 e que visa a melhorar o desempenho do Hive por meio das seguintes otimizações:

- *input/output* assíncrono e que leva em conta as rotações do disco rígido;
- *pre-fetching* (obter com antecedência) e *caching* de partes das colunas;
- operadores de *pipelines multi-threaded* e amigáveis com compiladores *just in time*.

O LLAP fornece um modelo híbrido de execução e é composto por um *daemon* de longa duração, que substitui as interações diretas com o *DataNode* do HDFS, além de ter uma *framework* baseada em DAG muito bem integrada. As funcionalidades de *caching* de dados, *pre-fetching*, partes do processamento de consultas e controle de acesso são deslocadas para esse *daemon*. Consultas menores e mais curtas são executadas diretamente pelo *daemon*, enquanto qualquer consulta mais pesada é executada pelos *containers* padrão do YARN (o gerenciador de recursos de *cluster* do Hadoop).

O LLAP é uma funcionalidade opcional do Hive e trabalha em conjunto com o processo de execução existente, apenas melhorando ele. O LLAP não é uma *engine* de execução — toda execução continua sendo agendada e monitorada pela *engine* de execução de forma transparente entre os nós do LLAP e os *containers* regulares. O suporte ao LLAP depende de cada *engine* de execução, sendo que, atualmente, existe suporte para o Tez, e não há planos para suportar o MapReduce (APACHE HIVE, 2019).

Hive 3

A versão 3 do Hive trouxe algumas melhorias, dentre elas a compatibilidade com a versão 3 do Hadoop. Das novas funcionalidades do Hive 3, as principais, de acordo com Leonard (2019), são as descritas a seguir.

- ***Views* materializadas:** o Hive 3 permite criar *views* que são materializadas, evitando a reexecução da consulta para construir a *view*. Ele oferece suporte à reconstrução incremental, atualizando a *view* com os dados que foram inseridos desde a última atualização, em vez de reconstruir a *view* completamente.
- **Novas *constraints*:** possibilidade de utilizar as *constraints UNIQUE e NOT NULL* em colunas de tabelas do Hive.
- **Novos conectores:** conector JDBC, permitindo a leitura de dados de bancos de dados externos por meio do Hive. Já o conector para Kafka permite ler dados de tópicos em tempo real.
- **ACID v2 (transações):** para tabelas no formato Apache ORC, agora é possível fazer ingestão em tempo real de dados, além de realizar operações de *insert*/*update*/*delete*. Também foi melhorado o desempenho em tabelas ACID; tabelas em formatos diferentes do Apache ORC agora têm suporte para *insert*/*select* com ACID e compatibilidade com armazenamento em nuvem, como o Amazon S3.

Saiba mais

Park (2018), em seu estudo, aplicou o *benchmark* TPC-DS, comparando as principais ferramentas de SQL-on-Hadoop, e demonstrou que o Hive 3 com LLAP habilitado conseguiu ser superior ao Hive 2 com Tez, ao Presto e até mesmo ao Spark 2.

Análise de dados com Hive

No Hive, a linguagem utilizada para consultar e manipular dados é o HiveQL, que é uma mistura dos dialetos do SQL-92, MySQL e Oracle. Ao longo do tempo, o suporte ao padrão SQL-92 tem melhorado, e existem ainda funções de padrões mais novos, como as funções de janela do SQL:2003. O HiveQL também possui algumas extensões fora dos padrões, como algumas funções emprestadas do MapReduce — por exemplo, o *multitable insert* e as cláusulas *TRANSFORM*, *MAP* e *REDUCE* (WHITE, 2015).

No Quadro 1, são detalhadas as principais funcionalidades do padrão SQL, comparando-as com as funcionalidades disponíveis no HiveQL. Por meio desse quadro, é possível saber como o Hive se equipara a bancos de dados relacionais com suporte completo ao SQL e em quais pontos o suporte ainda

é limitado, como é o caso das transações. Em outros casos, o HiveQL possui mais funcionalidades do que aquelas definidas no padrão do SQL.

Antes de abordarmos exemplos da linguagem HiveQL, precisamos compreender alguns conceitos básicos do Hive, descritos a seguir.

- **Tabelas:** de forma análoga às tabelas em bancos de dados relacionais, as tabelas do Hive permitem filtros, projeções, *joins* e *unions*. Todos os dados da tabela são armazenados em um sistema de arquivos (p. ex., HDFS), que pode ou não ser gerenciado pelo Hive (*internal table* e *external table*, respectivamente). As linhas de uma tabela são armazenadas em colunas com tipos de dados definidos.
- **Partições:** cada tabela pode ter uma ou mais chaves de partição, que determinam como os dados estão armazenados nos diretórios do sistema de arquivos. Por exemplo, uma tabela com chave de partição na coluna data terá dados de uma data específica armazenados no diretório <localização da tabela>/data=<data>. Isso permite que o Hive descarte parte dos dados que serão lidos com base nos predicados da consulta.
- ***Buckets*:** dados dentro de cada partição podem ser divididos em *buckets*, com base no *hash* de uma coluna da tabela. Cada *bucket* é armazenado como um arquivo no diretório da partição. A utilização de *buckets* permite que o sistema avalie com eficiência consultas que dependem de amostras de dados.

Por padrão, tabelas criadas serão gerenciadas pelo próprio Hive, sendo ele responsável por armazenar os dados no diretório de armazenamento dentro do sistema de arquivos e no formato de arquivos definido. Alternativamente, é possível criar tabelas externas para referenciar dados já existentes em uma localização fora do diretório de armazenamento do Hive.

Quadro 1: Comparação entre SQL e HiveQL

Funcionalidade	SQL	HiveQL
Updates	*UPDATE*, *INSERT*, *DELETE*	*UPDATE*, *INSERT*, *DELETE*
Transações	Suportado	Suporte limitado
Índices	Suportado	Suportado

(Continua)

(Continuação)

Funcionalidade	SQL	HiveQL
Tipos de dados	Inteiros, ponto flutuante, ponto fixo, texto e *strings* binárias, temporais	Booleanos, inteiros, ponto flutuante, ponto fixo, texto e *strings* binárias, temporais, *arrays*, *map*, *struct*
Funções	Centenas de funções internas	Centenas de funções internas
Inserts multitabela	Não suportado	Suportado
CREATE TABLE... AS SELECT	Não é valido no SQL-92, mas alguns bancos implementam	Suportado
SELECT	SQL-92	SQL-92; SORT BY para ordenação parcial, *LIMIT* para limitar o número de linhas retornado
Joins	SQL-92 ou variantes (*join* de tabelas na cláusula *FROM* e condição do *join* na cláusula *WHERE*)	*Inner joins*, *outer joins*, *semi joins*, *map joins*, *cross joins*
Subqueries	Em qualquer cláusula (correlacionada ou não correlacionada)	Nas cláusulas *FROM*, *WHERE* ou *HAVING* (*subqueries* não correlacionadas — não suportado)
Views	Atualizáveis (materializadas ou não materializadas)	Atualizáveis (materializadas ou não materializadas) (a partir do Hive 3)
Pontos de extensão	Funções definidas pelo usuário, *stored procedures*	Funções definidas pelo usuário, *scripts* MapReduce

Fonte: Adaptado de White (2015).

Para criar uma tabela interna no Hive e carregar dados para ela, podemos utilizar a cláusula *LOAD*. No exemplo a seguir, uma tabela chamada de *managed_table* é criada com uma coluna chamada *dummy* do tipo *string*. O segundo comando do exemplo carrega os dados do caminho informado para dentro da tabela gerenciada.

```
CREATE TABLE managed _ table (dummy STRING);
LOAD DATA INPATH '/user/tom/data.txt' INTO table
managed _ table;
```

Para deletar a tabela, utiliza-se o comando *DROP TABLE*, e, com isso, tanto os dados como os metadados da tabela são removidos. Já para uma tabela externa, a sua criação é um pouco diferente, como pode ser visto no exemplo a seguir. Além da inclusão da palavra-chave *EXTERNAL*, também é obrigatório informar onde os dados da tabela estão armazenados, por meio da palavra-chave *LOCATION*. É interessante notar que uma tabela externa do Hive pode ser criada mesmo que os dados ainda não existam na localização informada. Como ela é uma tabela que não é gerenciada pelo Hive, ele não faz nenhum tipo de verificação e também não é capaz de realizar alterações nos dados de tabelas externas.

```
CREATE EXTERNAL TABLE external _ table (dummy STRING)
 LOCATION '/user/tom/external _ table';
LOAD DATA INPATH '/user/tom/data.txt' INTO TABLE
external _ table;
```

Para deletar uma tabela externa, o comando é o mesmo que foi utilizado no exemplo anterior para a tabela interna. A diferença é que, quando uma tabela externa é deletada do Hive, apenas os metadados são removidos, e os dados continuam existindo, já que não são gerenciados pelo Hive. Essa é uma das maiores diferenças entre tabelas internas e externas (WHITE, 2015).

No Hive, existem duas dimensões do armazenamento de dados: o formato das linhas e o formato dos arquivos. O formato das linhas define como as linhas e os campos em uma determinada linha serão armazenados. Esse formato de linhas é definido no Hive por meio do SerDe. O SerDe é responsável por traduzir as linhas no formato do Hive para o formato de dados para armazenamento e vice-versa.

Já o formato de arquivos define o formato de *container* para os campos em uma linha. O formato mais simples é um arquivo de texto puro, mas existem outros formatos binários orientados a linha, como SequenceFile e Avro, e formatos orientados a coluna, como Record Columnar File, Optimized Row Columnar e *Parquet*.

Nas próximas subseções, serão apresentadas algumas das funcionalidades para consultar dados com o Hive.

Ordenação e agregação

A ordenação de dados no Hive pode ser feita utilizando-se a cláusula padrão *ORDER BY*. Por meio dessa cláusula, o Hive vai executar uma ordenação total e paralela da entrada (*Total Sort*). Quando não for necessário realizar uma ordenação global dos dados, é possível utilizar a cláusula *SORT BY*, que vai gerar um arquivo ordenado por *reducer* (WHITE, 2015).

Com o Hive, é possível ainda controlar para qual *reducer* uma linha em particular vai ir, método normalmente utilizado quando agregações subsequentes serão realizadas, o que pode otimizar a consulta. Nesses casos, a cláusula utilizada é a *DISTRIBUTE BY*. No exemplo a seguir, um *dataset* contendo temperaturas por ano é ordenado de modo que todas as linhas de um ano específico vão para a mesma partição do *reducer*.

```
FROM records2
SELECT year, temperature
DISTRIBUTE BY year
SORT BY year ASC, temperature DESC;
```

Caso as colunas do *SORT BY* sejam iguais às do *DISTRIBUTE BY*, é possível trocar essas duas cláusulas pelo *CLUSTER BY*.

Joins

Uma das operações que demonstra a facilidade de uso do Hive é o *join*, dada a complexidade envolvida em realizar essa operação utilizando o MapReduce (WHITE, 2015). O exemplo a seguir demonstra o mais simples de todos os *joins*, o *inner join*. Nesse exemplo específico, a tabela *sales* é unida com a tabela *things* através da cláusula *ON* pela coluna id de ambas. No Hive, apenas a equidade é permitida na cláusula *ON*, mas há suporte para *joins* com múltiplas colunas utilizando a palavra-chave *AND*.

Exemplo

```
SELECT sales.*, things.*
   FROM sales
JOIN things ON (sales.id = things.id);
```

Exemplo

```
SELECT station, year, AVG(max_temperature)
   FROM (
SELECT station, year, MAX(temperature) AS max_temperature
FROM records2
WHERE temperature != 9999 AND quality IN (0, 1, 4, 5, 9)
GROUP BY station, year
) mt
GROUP BY station, year;
```

No exemplo apresentado, a consulta calcula a temperatura média para cada ano e estação de medição. A *subquery* na cláusula *FROM* encontra a temperatura máxima para cada ano e estação de medição por meio da função *MAX*. Já a consulta externa acessa os resultados da *subquery* e utiliza a função *AVG* para calcular a média de temperatura por ano e estação de medição.

Views

Uma *view* é uma espécie de tabela virtual que é definida por meio de uma declaração de *SELECT*. *Views* são muito úteis para disponibilizar os dados para os usuários de uma forma diferente daquela armazenada em disco. *Views* podem agregar, simplificar e formatar dados de tabelas existentes, deixando mais conveniente o consumo dos dados ou facilitando o processamento dos mesmos em etapas posteriores (WHITE, 2015).

Até a versão 2 do Hive, *views* não eram materializadas. Isso significa que, toda vez que uma *view* é acessada, a declaração *SELECT* precisa ser executada para retornar os resultados para o usuário. Uma alternativa era a materialização manual por meio da expressão *CREATE TABLE... AS SELECT*. Já na versão 3 do Hive, existe a possibilidade de materializar *views*, permitindo que o seu resultado seja persistido em disco, evitando, assim, que a declaração de *SELECT* seja executada a cada acesso à *view*.

Utilizando o mesmo exemplo das *subqueries*, podemos criar uma *view* chamada *valid_records*, para representar parte da *subquery* na cláusula *FROM* apenas com os filtros. Podemos criar também outra *view* chamada *max_temperatures*, para representar a parte da consulta que calcula as temperaturas máximas por ano e estação de medição, utilizando a *view valid_records* na sua cláusula *FROM*.

Exemplo

```
CREATE VIEW valid _ records
   AS
SELECT *
FROM records2
WHERE temperature != 9999 AND quality IN (0, 1, 4, 5, 9);

CREATE VIEW max _ temperatures (station, year,
max _ temperature)
AS
SELECT station, year, MAX(temperature)
FROM valid _ records
GROUP BY station, year;
```

Exemplo

```
SELECT station, year, AVG(max _ temperature)
   FROM max _ temperatures
GROUP BY station, year;
```

O resultado da consulta é o mesmo que a execução da consulta com *subquery*, resultando no mesmo número de *jobs* do MapReduce.

Referências

APACHE HADOOP. *MapReduce tutorial*. California: Apache Software Foundation, 2020. Disponível em: https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html. Acesso em: 20 ago.2020.

APACHE HIVE. *Home*. California: Apache Software Foundation, 2019. Disponível em: https://cwiki.apache.org/confluence/display/Hive/Home. Acesso em: 20 ago. 2020.

DEAN, J.; GHEMAWAT, S. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, v. 51, n. 1, Jan. 2008. Disponível em: https://dl.acm.org/doi/10.1145/1327452.1327492. Acesso em: 20 ago. 2020.

LEONARD, G. Running Apache Hive 3, new features and tips and tricks. *In*: ADALTAS Blog. [*S. l.: s. n.*], 2019. Disponível em: https://www.adaltas.com/en/2019/07/25/hive-3--features-tips-tricks/. Acesso em: 20 ago. 2020.

PARK, S. Performance Evaluation of SQL-on-Hadoop Systems using the TPC-DS Benchmark. *In*: HIVE on Kubernetes. [*S. l.: s. n.*], 2018. Disponível em: https://mr3.postech.ac.kr/blog/2018/10/30/performance-evaluation-0.4/. Acesso em: 20 ago. 2020.

SIDDESH, G. M.; HIRIYANNAIAH, S.; SRINIVASA, K. G. Driving Big Data with Hadoop Technologies. In: RAJ, P.; DEKA, G. C. *Handbook of research on cloud infrastructures for big data analytics*. United States: IGI Global, 2014.

WHITE, T. *Hadoop:* the definitive guide. 4th. ed. Massachusetts: O'Reilly Media, 2015.

Leituras recomendadas

CAPRIOLO, E.; WAMPLER, D.; RUTHERGLEN, J. *Programming Hive:* data warehouse and query language for Hadoop. Massachusetts: O'Reilly Media, 2012.

DU, D. *Apache Hive Essentials.* Birmingham: Packt, 2015.

SHAW, S. *et al*. *Practical Hive:* a guide to Hadoop's data warehouse system. New York: Apress, 2016.

Fique atento

Os *links* para *sites* da *web* fornecidos neste capítulo foram todos testados, e seu funcionamento foi comprovado no momento da publicação do material. No entanto, a rede é extremamente dinâmica; suas páginas estão constantemente mudando de local e conteúdo. Assim, os editores declaram não ter qualquer responsabilidade sobre qualidade, precisão ou integralidade das informações referidas em tais *links*.

Conteúdo:

SAGAH
SOLUÇÕES EDUCACIONAIS INTEGRADAS