Aula 3 - Hive

Lucio Monteiro

Apache Hive

É um sistema de Data Warehouse de código aberto, usado originalmente para consultar e analisar grandes conjuntos de dados armazenados no Hadoop.

Criado pelo time de Infraestrutura de Dados do Facebook, o **Hive** utiliza uma linguagem chamada *HiveQL* (Hive Query Language), para transformar sentenças SQL em Jobs MapReduce executados no cluster Hadoop, solucionando o empasse em que estavam na época por ter analistas e desenvolvedores proficientes em SQL, mas com pouco conhecimento em JAVA para utilizar o MapReduce diretamente.

Características

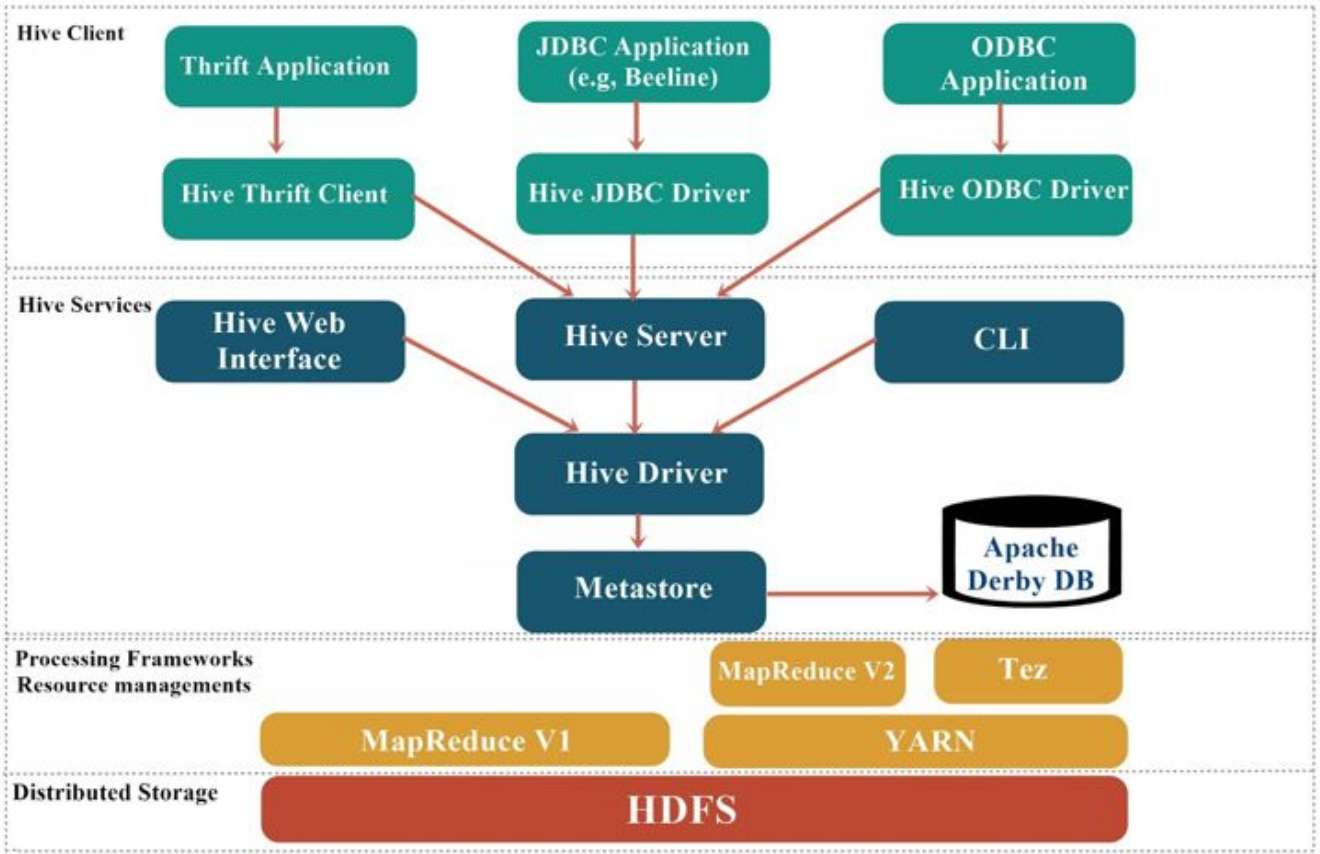
- Interpreta instruções *SQL* para jobs *MapReduce*.
- Lê dados de arquivos estruturados e semiestruturados no *HDFS*, e se baseia em metadados para simular tabelas de um banco de dados relacional.
- Não possui e nem é um *SGBD* (Sistema Gerenciador de Banco de Dados).
- Foi desenhado para melhor performance analisando grandes quantidades de dados que se encontram em clusters.
- *Data Warehouse* do *Apache Hadoop*.

Hive

Hive x RDBMS (Sistema de Gerenciamento de Banco de Dados Relacional)

	HIVE	RDBMS
Uso	Foco em analytics	Foco em on line ou analytics
Acesso	Batch	Batch e Interativo
Integridade	Baixa	Alta
Escalabilidade	Horizontal	Vertical
Armazenamento	Baixo custo por PB	PB?
Interface	HiveQL	SQL
Latência	Minutos ou mais	ms, ml ou segundos
Estrutura de dados	Não Estruturado	Estruturado

Arquitetura



Fonte: edureka

Componentes do Hive

Driver

Compila, otimiza, executa o HiveQL, e decide se executa a query local ou submeter em um job MapReduce.

Metastore

Armazena os metadados, interpretando os arquivos no *HDFS* como conteúdo de uma tabela. Armazena as informações de como as linhas e colunas são delimitadas dentro do arquivo (Hive Schema). Pode ser armazenado no MySQL, Oracle, Derby ou Postgresql.

HiveQL

Linguagem muito próxima do SQL (mais ainda do MySQL) que possibilita criação de bancos de dados e tabelas.

Hiveserver

Permite que conexões (Thrift, ODBC e JDBC) de outros componentes tenham comunicação com o **Hive**.

Componentes do Hive

CLI

Command Line Interface, linha de interface de comando, para acessar o shell do **Hive** via terminal do sistema operacional (SO).

Hiveserver2

Evolução do *hiveserver*, suporta autenticação e múltiplos usuários concorrentes.

Beeline

É a CLI para acessar o *hiveserver2*, utilizando conexão *JDBC*, também via terminal. Exemplo de uso do Beeline:

```
beeline -u "jdbc:hive2://localhost:10000/default"
```

Consultar tabelas:

```
show tables;
```

Checar comandos do Beeline:

```
beeline -help
```

Banco de Dados

Bancos de dados no **Apache Hive** são diretórios onde tabelas (subdiretórios) são armazenadas, uma espécie de catálogo portanto. Caso a cláusula `LOCATION` seja utilizada, o diretório será criado no caminho indicado, caso contrário, por padrão (padrão esse que pode ser alterado em arquivos de configuração) em */user/hive/warehouse*.

Tabela Gerenciada (Managed Table)

Também conhecida como tabela interna, tem o ciclo de vida de seus dados controlados pelo **Hive**, ou seja, quando excluímos a tabela os dados também são excluídos!

Se criada sem indicação de banco de dados, será alocada no `default`, cuja localização padrão é:

```
/user/hive/warehouse/NOME_TABELA
```

Caso tenha seu banco de dados indicado, o caminho padrão será:

```
/user/hive/warehouse/NOME_BANCO.db/NOME_TABELA
```

Tabela Externa (External Table)

Uma tabela externa é apenas uma referência a um diretório e arquivos, ou seja, o **Hive** não é dono dos dados. Neste caso, quando excluímos a tabela os dados não são excluídos, apenas seus metadados!

Com relação ao código, é preciso utilizar a palavra-chave `EXTERNAL` e indicar a localização dos dados através da cláusula `LOCATION`.

Formato de Arquivos

- Não existe um formato Hive

- Conector para vários formatos
  - Arquivos de texto com valores separados por vírgula e tabulação (CSV / TSV)
  - Sequence File
  - Parquet
  - ORC
  - AVRO
  - JSONFILE
  - Outros ...

Text File

- Arquivos de texto
- Formato padrão
  - Hive
  - Sqoop
- Facilidade para compartilhar os dados do Hadoop com outros sistemas externos
- Facilidade de edição manualmente
- Menos eficiente que outros formatos
- Exemplo
  - txt
  - csv
  - Estruturas de texto
    - xml
    - json

Sequence File

- Arquivo de Sequência do Hadoop
- Formado por pares chave e valor
- Armazena em formato binário
- Mais eficiente que o arquivo de texto
- Facilidade para compartilhar os dados com outras ferramentas do Hadoop

ORC File

- Optimized Row Colunnar File
- Substituiu o formato RC File
  - Mesmas características
- Compacta melhor os arquivos RC
  - Consultas mais rápidas
- Projetado para otimizar o desempenho no Hive
  - Formado por faixas
    - Grupo de dados de linha
  - Não usado para MapReduce não-Hive
    - Pig
    - Impala
    - Java

Avro

- Formado por serialização de dados com neutralidade de linguagem
- Armazenamento dos dados e metadados juntos
- Vantagem
  - Suporta MapReduce
  - Suporta evolução do esquema

Parquet

- Formato colunar
  - Formado por grupos de colunas
- Vantagem
  - Suporta MapReduce
    - Pig
    - Hive
    - Java
  - processamento
    - Impala
    - SparK
  - Suporta evolução do esquema
    - Hive
    - Impala

Compressão de Dados

- Armazenamento x Velocidade de leitura

- ZLIB (GZip)
  - Alta compressão, lento
- Snappy
  - Baixa compressão, rápido

Comandos - Informações BD e Tabelas

- Listar todos os BD
  - show database;
- Estrutura sobre o bd
  - desc database <nomeBD>;
- Listar as tabelas
  - show tables;
- Estrutura da tabela
  - desc <nomeTabela>;
  - desc formatted <nomeTabela>;
  - desc extended <nomeTabela>;

Comandos - Criação Banco de Dados

- Criar BD
  - create database <nomeBanco>;
- Local diferente do conf. Hive
  - create database <nomeBanco> location “/diretorio”;
- Adicionar comentário
  - create database <nomeBanco> comment “descrição”;
- Ex
  - create database test location “/user/hive/warehouse/test” comment “banco de dados para treinamento”
  - default
    - /user/hive/warehouse/test.db

Comandos - Tabela Interna e Externa

- Tabela interna
  - create table user(cod int, name string);
  - drop table
    - Apaga os dados e metadados
- Tabela externa
  - create external table e_user(cod int, name string) location '/user/
  - drop table
    - Usar para compartilhar os dados com outras ferramentas
    - Apaga apenas os metadados
    - Dados ficam armazenado no sistema de arquivos

Comandos - Tipos Dados Simples

- INT
- SMALLINT
- TINYINT
- BIGINT
- BOOLEAN
- FLOAT
- DOUBLE
- DECIMAL
- STRING
- VARCHAR
- CHAR

Comandos - Tipos Dados Complexos

- ARRAY
  - Lista de Elementos [‘Seg’, ‘Ter’, ‘Qua’, ‘Qui’, ‘Sex’]
- MAP
  - Tipo Chave-valor ‘nome’ ->’Rodrigo’
- STRUCT
  - Define os campos dos seus tipos de dados
  - STRUCT<col_name : data_type, ...
- UNION
  - Armazenar diferentes tipos de dados no mesmo local de memória
  - UNIONTYPE<data_type, data_type, ...>

Comandos - Opções de Leitura

- Definir delimitadores
  - row format delimited
  - fields terminated by ‘<delimitador>'
  - lines terminated by ‘<delimitador>’
  - Delimitadores: ‘qualquer coisa’, \b (backspace), \n (newline), \t (tab)
- Pular um número de linhas de leitura do arquivo
  - tblproperties("skip.header.line.count"=“<número de linhas");
- Definir Localização dos dados (Tabela externa)
  - location '/user/cloudera/data/client’;

Comandos - Criação de Tabela

- Tabela Externa

```
create external table user(
        id int,
        name String,
        age int
)
row format delimited
fields terminated by '\t'
lines terminated by '\n'
stored as textfile
location '/user/cloudera/data/client';
```

Comandos - Inserção de Dados

- Inserir dados
  - insert into table <nomeTabela> partition(<partition>='<value>') values(<campo>,<value>), (<campo>,<value>), (<campo>,<value>);
- Ex
  - insert into users values(10, 'Rodrigo'),(11,'Augusto');
  - insert into users partition(data=now()) values(10, 'Rodrigo'),(11,'Augusto');
  - insert into users select * from cliente;

Comandos - Carregamento Dados

- Carregar dados no sistema de arquivos local
  - hive> load data inpath <diretório> into table <nomeTabela>;
- Ex.
  - load data local inpath '/home/cloudera/data/test' into table alunos
  - load data inpath '/user/cloudera/data/test' overwrite into table alunos partition(id)

Comandos - Seleção Dados

- select * from <nometable>
  <where ...>
  <group by ... >
  <having ... >
  <order by ... >
  <limit n>;

- Ex
  - hive> **select** * from client **where** state=sp **group by** city **having** population > 100 **order by** client **limit** 10;

Comandos - Join

- Aceita apenas ANSI JOINS
- Inner Join, Left Outer, Right Outer, Full Outer
  - select * from a join b on a.valor = b.valor
  - select * from a,b where a.valor = b.valor
    - erro

Comandos - View - Consultas

- Salvar consultas
- Tratar como tabelas
- Objetos Lógicos
  - Esquema é fixo quando criado a View
  - Alterar tabela não altera a view
- Comando
  - create view <nomeView> as select * from nome_table;

Particionamento no Hive

Particionamento de tabelas no **Hive** funciona de forma similar a bancos relacionais. Trata-se de uma maneira de dividir uma tabela em partes menores com base nos valores de colunas específicas, como data, cidade e departamento. Cada tabela na seção pode ter uma ou mais chaves de partição para identificar uma partição específica.

Características:

- Particiona a tabela em diretórios no *HDFS*.
- Ganho de performance e eficiência.
- Evita a leitura da base toda.
- Muito utilizado em grandes volumes de dados.

O exemplo a seguir criará uma tabela chamada table_part particionada por ano:

Criar tabela particionada:

```
CREATE EXTERNAL TABLE table_part(nome string)

PARTITIONED BY (ano int)

ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' STORED AS TEXTFILE;
```

Verificar tabela:

```
DESCRIBE table_part;
```

Particionamento no Hive

Particionamento de tabelas no **Hive** funciona de forma similar a bancos relacionais. Trata-se de uma maneira de dividir uma tabela em partes menores com base nos valores de colunas específicas, como data, cidade e departamento. Cada tabela na seção pode ter uma ou mais chaves de partição para identificar uma partição específica.

Características:

- Particiona a tabela em diretórios no *HDFS*.
- Ganho de performance e eficiência.
- Evita a leitura da base toda.
- Muito utilizado em grandes volumes de dados.

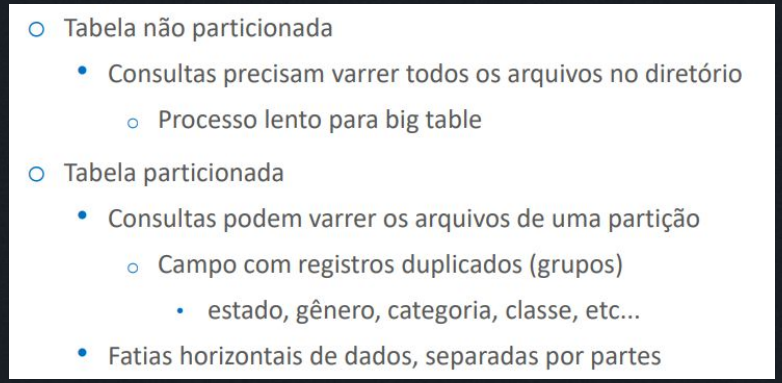


Tabela Particionada

- Definir parâmetros na criação da tabela
  - Partição
    - Um campo que não está na estrutura da tabela
    - Organizar os dados
    - Consultas SQL interpretarão como coluna
      - Select * from user where **cidade**=sp
    - Comando
      - partitioned by (<campo> <type>)
  - Buket
    - Quantidade que os dados serão divididos
    - Campo precisa estar na estrutura da tabela
    - Comando
      - clustered by (<campo>) into <qtd> buckets

- Tabela Particionada

```
create table user(
        id int,
        name String,
        age int
)
partitioned by (data String)
clustered by (id) into 4 buckets;
```

Particionamento Estático

- Você pode inserir os arquivos individualmente em uma tabela de partição
- Criar novas partições manualmente
- Comando
  - hive> alter table <nomeTabela> add partition(<partição>='<valor>');
- Ex.
  - Criar uma partição para cada dia
  - hive> alter table logs add partition(data='2019-21-02');

Particionamento Dinâmico

- Partições são criadas automaticamente nos tempos de carregamento
- Novas partições podem ser criadas dinamicamente
  - Baseada no valor da última coluna
    - Se a partição não existir, ela será criada
    - Se existir, os dados serão adicionados na partição
  - Ex.
    - hive> insert overwrite table user_cidade partition (cidade) select * from user;
- Por padrão, o particionamento dinâmico está desativado, para ativar
    - SET hive.exec.dynamic.partition = true;
    - SET hive.exec.dynamic.partition.mode = nonstrict ;

Partições

- Visualizar partições de uma tabela
  - hive> show partitions user;
- Excluir partições de uma tabela
  - hive> alter table user drop partition (city='SP');
- Alterar nome da partição de uma tabela
  - hive> alter table user partition city rename to partition state;

Estrutura dos Dados

- Dados estruturados e semi-estruturados
- Hierarquia dos dados
  - Database
    - Table
      - Partition - Coluna de armazenamento dos dados no sistema de arquivo (diretórios)
        - Bucket - Dados são divididos em uma coluna através de Hash

- Exemplo de caminho

/user/hive/warehouse/banco.db/tabela/data=010119/000000_0

Hive no Databricks

Assim como o *HDFS* do *Hadoop* serviu de base para serviços de plataformas em nuvem de grandes fornecedores, o **Hive** também, que pode ser notado no Databricks, na AWS (como no Gue Catalog), etc.

Banco de Dados

Como bem sabe, no **Hive** não há bancos de dados propriamente ditos, mas um conjunto de diretórios, arquivos e metadados registrados.

Em um notebook Databricks é possível, em uma célula SQL, utilizar comandos como:

Criar banco de dados:

```
CREATE DATABASE <nome_do_bd>;
```

Deletar banco de dados:

```
DROP DATABASE <nome_do_bd>;
```

Durante o processo de criação de um banco de dados também é possível utilizar a cláusula `LOCATION` que apontará o caminho onde o banco de dados será criado. Caso contrário, o caminho padrão é: */user/hive/warehouse*

Um outro aspecto interessante é que, diferente do SGBD Oracle, e assim como no MySQL, os conceitos de Database e Schema se misturam, sendo aqui no Databricks o comando *CREATE DATABASE* um alias para *CREATE SCHEMA*, que pode ser usado preferivelmente e encarado como boa prática.

Tabelas Gerenciadas

Para criar tabelas gerenciadas no Databricks basta utilizar o comando *CREATE TABLE* sem as cláusulas `LOCATION` e `EXTERNAL`. Seu diretório padrão será */user/hive/warehouse* e os registros inseridos respeitarão o comportamento visto anteriormente: se a tabela for apagada, eles também serão.

Tabelas Externas

No **Hive** dentro do Databicks também existe o conceito de tabelas externas, onde os dados não "pertencem" ao vive, ou seja, se a tabela for deletada, os arquivos se manterão lá, apenas metadados são perdidos.

Para indicar que a tabela a ser criada é externa é possível utilizar a cláusula `EXTERNAL` já vista, ou simplesmente utilizando a `LOCATION`, pois aqui não é possível definir um caminho para tabelas gerenciadas.

Formato de arquivos

Um comportamento interessante das tabelas do Databricks é que, por padrão, ao inserir registros nelas, os arquivos salvos estarão no formato *.parquet*.

Há uma cláusula chamada `USING` que pode ser usada durante a criação de tabelas e permite que diferentes formatos sejam utilizados, como no exemplo abaixo:

```
CREATE TABLE <nome_tabela> (<campos e tipos de dados>)USING CSV;
```

Além do CSV, também é possível trabalhar com *json*, *avro*, *parquet*, *orc* e *delta*, que abordaremos com mais profundidade futuramente.

Indicações e Bibliografia

Databricks Community

Documentação oficial

Databricks na AWS

Comandos para manipulação do DBFS

Schema vs Database

Livro - Programming Hive: Data Warehouse and Query Language for Hadoop

Obrig.ada