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.

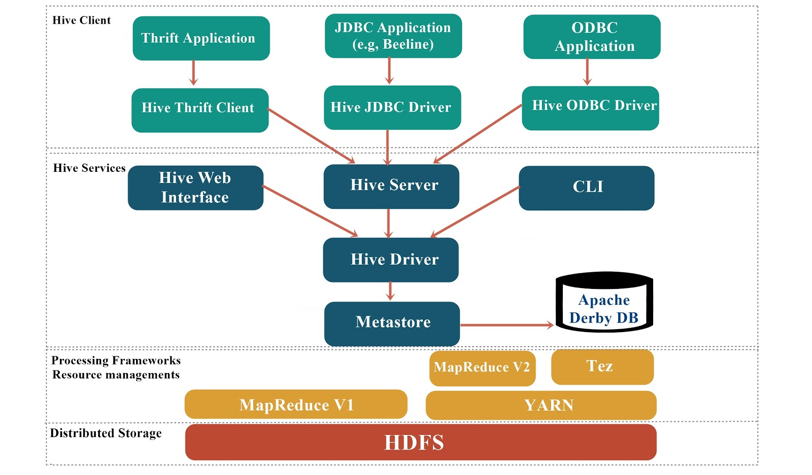
## Características

* 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 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](https://www.edureka.co/blog/wp-content/uploads/2016/11/Hive-Architecture-Hive-Tutorial-Edureka-1.png)

## 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.

### 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.

### Comandos

Criar banco de dados:

CREATE DATABASE IF NOT EXISTS [nome\_banco\_de\_dados]

LOCATION [caminho no HDFS];

Visualizar bancos de dados:

SHOW DATABASES;

Definição do banco de dados:

DESCRIBE DATABASE [nome\_banco\_de\_dados];

Excluir banco de dados:

DROP DATABASE [nome\_banco\_de\_dados];

Alterar banco de dados:

ALTER DATABASE [nome\_banco\_de\_dados] [parâmetros];

## View

Funcionam como Views em SGBDS: uma referência lógica de uma consulta, de certa forma algo como salvar uma query, para ser acessada como uma tabela física tradicional. Muito utilizada para consultas pre-estabelecidas por usuários e para controlar acesso a campos.

Criar View:

CREATE VIEW <nome\_da\_view> AS SELECT \* FROM <nome\_tabela>;

## SerDe

Abreviação de Serializer / Deserializer, permite que o Hive leia os dados de uma tabela e os grave no HDFS em qualquer formato personalizado, tais quais Avro, ORC, RegEx, Parquet, CSV e JsonSerDe.

### Formatos de arquivos – Posicional

Tomando como exemplo o arquivo posicional abaixo, em algum diretório no HDFS:

00001cliente\_01PF

00002cliente\_02PJ

00003cliente\_03PF

É possível criar uma tabela para lê-lo corretamente com através do código:

CREATE EXTERNAl TABLE tabela\_posicional (conta string, nome string, tipo\_pessoa string)

ROW FORMAT SERDE 'org.apache.hadoop.hive.contrib.serde2.RegexSerDe'

WITH SERDEPROPERTIES ('input.regex' ='(.{5})(.{10})(.{2})')

LOCATION '/user/hive/tabela\_posicional';

### Formatos de arquivos - Delimitado

Tomando como exemplo o arquivo delimitado por vírgulas abaixo, em algum diretório no HDFS:

1,cliente\_01,PF

2,cliente\_02,PJ

3,cliente\_03,PF

É possível criar uma tabela para lê-lo corretamente com através do código:

CREATE EXTERNAL TABLE tabela\_delimitada (conta string, nome string, tipo\_pessoa string)

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

LOCATION '/user/hive/tabela\_delimitada';

### Formatos de arquivos – JSON

JSON é um modelo para armazenamento de dados com capacidade de estruturar informações de uma forma bem mais compacta do que os arquivos sequenciais ou delimitados.

A tabela de exemplo criada com o código abaixo:

CREATE EXTERNAL TABLE tabela\_json (campo string) STORED AS TEXTFILE LOCATION '/user/hive/tabela\_json';

Pode ser carregada com o seguinte arquivo de exemplo:

{"conta":1,"cliente":{"nome":"cliente\_1","tipo\_pessoa":"PF"}}

{"conta":2,"cliente":{"nome":"cliente\_2","tipo\_pessoa":"PJ"}}

{"conta":3,"cliente":{"nome":"cliente\_3","tipo\_pessoa":"PF"}}

Os dados podem ser consultados através de uma query como esta:

SELECT get\_json\_object(campo, "$.conta") as conta, get\_json\_object(campo, "$.cliente.nome") as nome, get\_json\_object(campo, "$.cliente.tipo\_pessoa") as tipo\_pessoa FROM tabela\_json;

### 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 meenores 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;

Adicionando partição:

ALTER TABLE table\_part

ADD PARTITION(ano=2021)

LOCATION '/user/hive/tabela\_part/particao/ano=2021';

Excluindo partição:

ALTER TABLE table\_part

DROP PARTITION(ano=2021);

### Extração de dados Hive - HDFS

Extrair dados do Hive para HDFS:

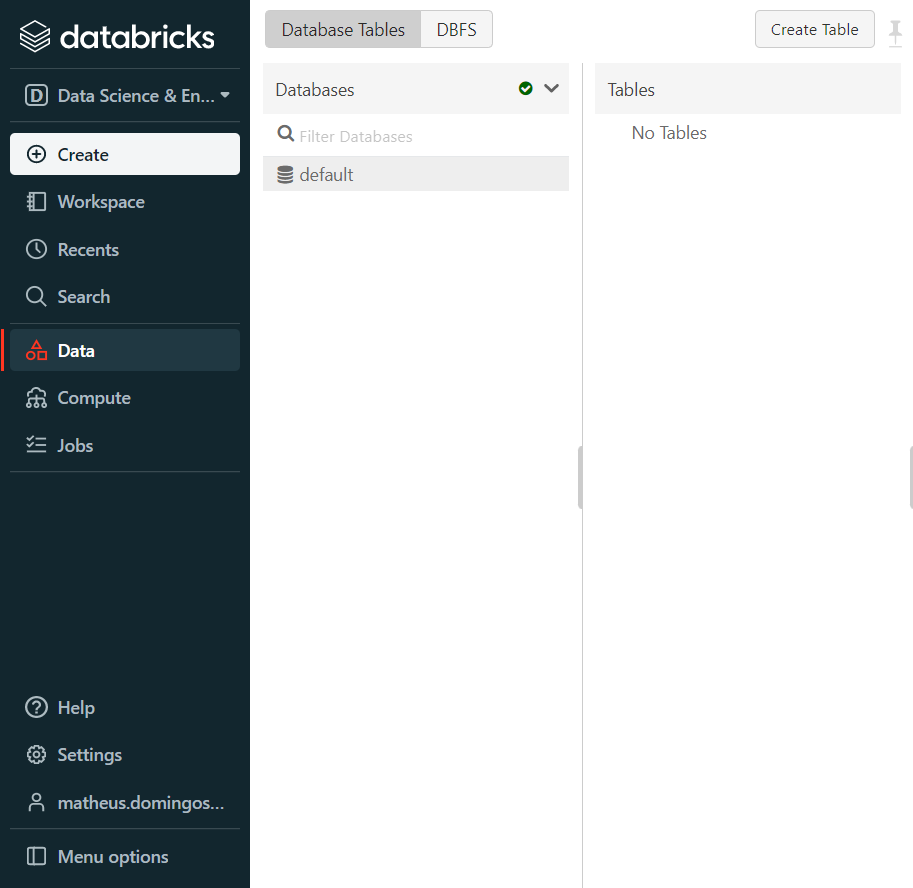
INSERT OVERWRITE DIRECTORY ‘/user/hive/output’

SELECT \* FROM table\_part;

## 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.

Bancos de dados e tabelas aqui criados podem ser visualizados no menu Data à esquerda.



### 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](https://community.cloud.databricks.com/)

[Documentação oficial](https://docs.databricks.com/)

[Databricks na AWS](https://aws.amazon.com/pt/quickstart/architecture/databricks/)

[Linguagens compatíveis com Databricks](https://docs.microsoft.com/pt-br/azure/databricks/languages/)

[Comandos para manipulação do DBFS](https://docs.databricks.com/dev-tools/databricks-utils.html)

[Schema vs Database](https://docs.databricks.com/spark/latest/spark-sql/language-manual/sql-ref-syntax-ddl-create-schema.html)

Livro - [Programming Hive: Data Warehouse and Query Language for Hadoop](https://www.oreilly.com/library/view/programming-hive/9781449326944/)