Hive — SQL-движок над MapReduce

<https://habr.com/ru/company/dca/blog/283212/>

Классические СУБД, такие как Postgres, MySQL или Oracle не имеют такой гибкости в масштабировании при обработке больших массивов данных и при достижении объема большего дальнейшая поддержка становится большой головной болью.   
  
  
Собственно, [Apache Hive](https://hive.apache.org/) был придуман для того чтобы объеденить два этих достоинства:

* Масштабируемость MapReduce
* Удобство использования SQL для выборок из данных.

Архитектура

Hive представляет из себя движок,  который превращает SQL-запросы в цепочки map-reduce задач.  Движок включает в себя такие компоненты, как Parser(разбирает входящие SQL-запрсоы), Optimimer(оптимизирует запрос для достижения большей эффективности), Planner (планирует задачи на выполнение) Executor(запускает задачи на фреймворке MapReduce.  
  
Для работы hive также необходимо хранилище метаданных. Дело в том что SQL предполагает работу с такими объектами как база данных, таблица, колонки, строчки, ячейки и тд. Поскольку сами данные, которые использует hive хранятся просто в виде файлов на hdfs — необходимо где-то хранить соответствие между объектами hive и реальными файлами.   
  
В качестве metastorage используется обычная реляционная СУБД, такая как MySQL, PostgreSQL или Oracle. 

Command line interface

Для того чтобы попробовать работу с hive проще всего воспользоваться его командной строкой. Современная утилита для работы с hive называется **beeline**(привет нашим партнёрам из одноименного оператора :) ) Для этого на любой машине в hadoop-кластере (см. [наш туториал по hadoop](https://habrahabr.ru/company/dca/blog/268277/)) с установленным hive достаточно набрать команду. 

$beeline

Далее необходимо установить соединение с hive-сервером: 

beeline> !connect jdbc:hive2://localhost:10000/default root root

Connecting to jdbc:hive2://localhost:10000/default

Connected to: Apache Hive (version 1.1.0-cdh5.7.0)

Driver: Hive JDBC (version 1.1.0-cdh5.7.0)

Transaction isolation: TRANSACTION\_REPEATABLE\_READ

0: jdbc:hive2://localhost:10000/default>

root root — в данном контексте это имя пользователя и пароль. После этого вы получите командную строку, в которой можно вводить команды hive.  
  
Также иногда бывает удобно не вводить sql-запросы в командную строку beeline, а предварительно сохранить и редактировать их в файле, а потом выполнить все запросы из файла. Для этого нужно выполнить beeline с параметрами подключения к базе данных и параметром -f указывающим имя файла, содержащего запросы:

beeline -u jdbc:hive2://localhost:10000/default -n root -p root -f sorted.sql

Data Units

При работе с hive можно выделить следующие объекты которыми оперирует hive:

1. База данных
2. Таблица
3. Партиция (partition)
4. Бакет (bucket)

Разберем каждый из них подробнее: 

База данных

**База данных** представляет аналог базы данных в реляционных СУБД. База данных представляет собой **пространство имён**, содержащее таблицы. Команда создания новой базы данных выглядит следующим образом: 

CREATE DATABASE|SCHEMA [IF NOT EXISTS] <database name>

Database и Schema в данном контексте это одно и тоже. Необязательная добавка **IF NOT EXISTS** как не сложно догадаться создает базу данных только в том случае если она еще не существует.   
  
Пример создания базы данных: 

CREATE DATABASE userdb;

Для переключения на соответствующую базу данных используем команду USE:

USE userdb;

Таблица

Таблица в hive представляет из себя аналог таблицы в классической реляционной БД. Основное отличие — что данные hive’овских таблиц хранятся прост в виде обычных файлов на hdfs. Это могут быть обычные текстовые csv-файлы, бинарные sequence-файлы, более сложные колоночные paruqet-файлы и другие форматы. Но в любом случае данные, над которыми настроена hive-таблица очень легко прочитать и не из hive.   
  
Таблицы в hive бывают двух видов:   
  
**Классическая таблица**, данные в которую добавляются при помощи hive. Вот пример создания такой таблицы ([источник примера](http://www.tutorialspoint.com/hive/hive_create_table.htm)): 

CREATE TABLE IF NOT EXISTS employee ( eid int, name String,

salary String, destination String)

COMMENT 'Employee details'

ROW FORMAT DELIMITED

FIELDS TERMINATED BY '\t'

LINES TERMINATED BY '\n'

STORED AS TEXTFILE;

Тут мы создали таблицу, данные в которой будут храниться в виде обычных csv-файлов, колонки которой разделены символом табуляции.  После этого данные в таблицу можно загрузить. Пусть у нашего пользователя в домашней папке на hdfs есть (напоминаю, что загрузить файл можно при помощи **hadoop fs -put**) файл sample.txt вида:

1201 Gopal 45000 Technical manager

1202 Manisha 45000 Proof reader

1203 Masthanvali 40000 Technical writer

1204 Kiran 40000 Hr Admin

1205 Kranthi 30000 Op Admin

Загрузить данные мы сможем при помощи следующей команды: 

LOAD DATA INPATH '/user/root/sample.txt'

OVERWRITE INTO TABLE employee;

После hive **переместит** данныe, хранящемся в нашем файле в хранилище hive. Убедиться в этом можно прочитав данные напрямую из файла в хранилище hive в hdfs:

[root@quickstart ~]# hadoop fs -text /user/hive/warehouse/userdb.db/employee/\*

1201  Gopal       45000    Technical manager

1202  Manisha     45000    Proof reader

1203  Masthanvali 40000    Technical writer

1204  Kiran       40000    Hr Admin

1205  Kranthi     30000    Op Admin

Классические таблицы можно также создавать как результат select-запроса к другим таблицам: 

0: jdbc:hive2://localhost:10000/default> CREATE TABLE big\_salary as SELECT \* FROM employee WHERE salary > 40000;

0: jdbc:hive2://localhost:10000/default> SELECT \* FROM big\_salary;

+*-----------------+------------------+--------------------+-------------------------+--+*

| big\_salary.eid  | big\_salary.name  | big\_salary.salary  | big\_salary.destination  |

+*-----------------+------------------+--------------------+-------------------------+--+*

| 1201            | Gopal            | 45000              | Technical manager       |

| 1202            | Manisha          | 45000              | Proof reader            |

+*-----------------+------------------+--------------------+-------------------------+--+*

Кстати говоря, SELECT для создания таблицы в данном случае уже запустит mapreduce-задачу.   
  
**Внешняя таблица**, данные в которую загружаются внешними системами, без участия hive. Для работы с  внешними таблицами при создании таблицы нужно указать ключевое слово **EXTERNAL**, а также указать путь до папки, по которому хранятся файлы:

CREATE EXTERNAL TABLE IF NOT EXISTS employee\_external ( eid int, name String,

salary String, destination String)

COMMENT 'Employee details'

ROW FORMAT DELIMITED

FIELDS TERMINATED BY '\t'

LINES TERMINATED BY '\n'

STORED AS TEXTFILE

LOCATION '/user/root/external\_files/';

После этого таблицей можно пользоваться точно так же как и обычными таблицами hive. Самое удобное в этом, что вы можете просто скопировать файл в нужную папочку в hdfs, а hive будет автоматом подхватывать новые файлы при запросах к соответствующей таблице. Это очень удобно при работе например с логами. 

Партиция (partition)

Так как hive представляет из себя движок для трансляции SQL-запросов в mapreduce-задачи, то обычно даже простейшие запросы к таблице приводят к полному сканированию данных в этой таблицы. Для того чтобы избежать полного сканирования данных по некоторым из колонок таблицы можно произвести партиционирование этой таблицы. Это означает, что данные относящиеся к разным значениям будут физически храниться в разных папках на HDFS.   
  
Для создания партиционированной таблицы необходимо указать по каким колонкам будет произведено партиционирование:

CREATE TABLE IF NOT EXISTS employee\_partitioned ( eid int, name String,

salary String, destination String)

COMMENT 'Employee details'

PARTITIONED BY (birth\_year int, birth\_month string)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY '\t'

LINES TERMINATED BY '\n'

STORED AS TEXTFILE;

При заливке данных в такую таблицу необходимо явно указать, в какую партицию мы заливаем данные: 

LOAD DATA INPATH '/user/root/sample.txt' OVERWRITE

INTO TABLE employee\_partitioned

PARTITION (birth\_year=1998, birth\_month='May');

Посмотрим теперь как выглядит структура директорий: 

[root@quickstart ~]# hadoop fs -ls /user/hive/warehouse/employee\_partitioned/

Found 1 items

drwxrwxrwx   - root supergroup          0 2016-05-08 15:03 /user/hive/warehouse/employee\_partitioned/birth\_year=1998

[root@quickstart ~]# hadoop fs -ls -R /user/hive/warehouse/employee\_partitioned/

drwxrwxrwx   - root supergroup          0 2016-05-08 15:03 /user/hive/warehouse/employee\_partitioned/birth\_year=1998

drwxrwxrwx   - root supergroup          0 2016-05-08 15:03 /user/hive/warehouse/employee\_partitioned/birth\_year=1998/birth\_month=May

-rwxrwxrwx   1 root supergroup        161 2016-05-08 15:03 /user/hive/warehouse/employee\_partitioned/birth\_year=1998/birth\_month=May/sample.txt

Видно, что структура директорий выглядит таким образом,  что каждой партиции соответствует отдельная папка на hdfs. Теперь, если мы будем запускать какие-либо запросы, у казав в условии WHERE ограничение на значения партиций — mapreduce возьмет входные данные только из соответствующих папок.   
  
В случае External таблиц партиционирование работает аналогичным образом, но подобную структуру директорий придется создавать вручную.   
  
Партиционирование очень удобно например для разделения логов по датам, так как правило любые запросы за статистикой содержат ограничение по датам. Это позволяет существенно сократить время запроса.