03-Hive.md 11.6 KB

Семинар по Hive (МВС-2017)

1. Запуск оболочки Hive

Существует 3 основных вида запуска задач в Hive. Проверим их работу на команде `SHOW DATABASES` (показывает список существующих баз данных Hive).

- запуск с помощью интерактивной оболочки:

```
$ hive
hive> SHOW DATABASES;
```

- запуск внешней команды:

```
$ hive -e 'SHOW DATABASES'
```

- запуск внешнего файла:

```
$ echo 'SHOW DATABASES' > sh_db.sql # запись в файл
hive -f sh_db.sql
```

Hive shell также позволяет запускать shell-команды внутри оболочки Hive. *После '**!**' не должно быть пробелов, после команды должна ставиться '**;**'*. Попробуем получить кол-во виртуальных ядер на клиенте:

```
hive> !nproc;
```

Составные команды вроде `cat file | grep 'key' | tee new_file` hive shell не поддерживает. Также можно из Hive shell работать с hdfs.

```
hive> dfs -ls;
```

2. Создание базы данных

- Создадим тестовую БД.

```
hive> create database <YOUR_USER>_test location '/user/<YOUR_USER>/test_metas
```

`Hive metastore` - это реляционная БД, которая находится в HDFS. Она хранит метаинформацию о таблицах. При создании базы нужно указать **полный путь** к metastore.

- Если вы указали неверное название базы или LOCATION, базу можно удалить:

```
hive> drop database if exists <YOUR_USER>_test cascade;
```

Слово `CASCADE` отвечает за удаление базы вместе с её содержимым.

- Вывод информации о БД

```
hive -e 'DESCRIBE DATABASE <YOUR_USER>_test'
```

3. Создание таблиц

Создадим таблицу в тестовой базе. Для исходных данных используем датасет "Подсети" (`/data/subnets/variant1`):

- IP-адрес,
- маска подсети, в которой он находится.

```
ADD JAR /opt/cloudera/parcels/CDH/lib/hive/lib/hive-contrib.jar;
USE <YOUR_USER>_test;
DROP TABLE IF EXISTS Subnets;

CREATE EXTERNAL TABLE Subnets (
    ip STRING,
    mask STRING
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY  '\t'
STORED AS TEXTFILE
LOCATION '/data/subnets/variant1';
```

Пояснения

1. `ADD JAR` - подключение Jar'ника с классами Hive.
2. `USE ...` - подключение к базе данных. Без этой строки таблицы будут создаваться в базе "default". Также можно вместо `USE` использовать аргумент `--database` при запуске запроса.

3. `EXTERNAL` - существует 2 типа таблиц: managed и external. External-таблицы работают с внешними данными не изменяя их, а managed позволяют их изменять.
4. `STORED AS` здесь выбирается формат хранения таблицы. Для External-таблиц формат должен совпадать с форматом хранения данных. Для managed рекомендуется использовать сжатые форматы хранения (RCFile, AVRO и т.д.).

Записываем код в файл, сохраняемся и запускаем:

```
hive -f my_query.hql
```

Проверим, как создалась таблица (выведем первые 10 строк):

```
hive --database <YOUR_USER>_test -e 'SELECT * FROM Subnets LIMIT 10'
```

4. Партиционирование

Создадим партиционированную таблицу из таблицы Subnets. Информация о каждой партиции хранится в отдельной HDFS-директории внутри metastore.

```
ADD JAR /opt/cloudera/parcels/CDH/lib/hive/lib/hive-contrib.jar;

SET hive.exec.dynamic.partition.mode=nonstrict;

USE <YOUR_USER>_test;
DROP TABLE IF EXISTS SubnetsPart;

CREATE EXTERNAL TABLE SubnetsPart (
    ip STRING
)
PARTITIONED BY (mask STRING)
STORED AS TEXTFILE;


INSERT OVERWRITE TABLE SubnetsPart PARTITION (mask)
SELECT * FROM Subnets;
```

Уже здесь вы можете увидеть, что запрос транслируется в MapReduce-задачу. Чтоб убедится в этом, можно зайти на ApplicationMaster UI: http://mipt-master.atp-fivt.org:8088 (http://mipt-master.atp-fivt.org:8088) Видим 1 MapReduce Job, в которой имеется только Map-стадия.

Проверить получившиеся партиции:

```
hive --database <YOUR_USER>_test -e 'SHOW PARTITIONS SubnetsPart'
```

5. Парсинг входных данных с помощью регулярных выражений

1. Создаём новую таблицу.

```
add jar /opt/cloudera/parcels/CDH/lib/hive/lib/hive-contrib.jar;
add jar /opt/cloudera/parcels/CDH/lib/hive/lib/hive-serde.jar;
USE <YOUR_USER>_test;
DROP TABLE IF EXISTS SerDeExample;

CREATE EXTERNAL TABLE SerDeExample (
    ip STRING,
    date STRING,
    request STRING,
    responseCode STRING
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.contrib.serde2.RegexSerDe'
WITH SERDEPROPERTIES (
    "input.regex" = '^(\\S*)\\t.*$'
)
STORED AS TEXTFILE
LOCATION '/data/user_logs/user_logs_S';

select * from SerDeExample limit 10;
```

Получаем на выходе:

```
135.124.143.193 NULL    NULL    NULL
247.182.249.253 NULL    NULL    NULL
135.124.143.193 NULL    NULL    NULL
222.131.187.37  NULL    NULL    NULL
 ...
```

Видим, что получилось «откусить» регуляркой первое поле. Остальные пока NULL'ы.

2. Пробуем откусить следующее поле. Меняем регулярное выражение (3 табуляции ставятся только в этом случае, все остальные поля разделены одной табуляцией).

```
"input.regex" = '^(\\S*)\\t\\t\\t(\\S*)\\t.*$'
```

Сохраняемся, запускаем: `$ hive –f myQuery.sql` Получаем:

```
135.124.143.193 20150601013300  NULL    NULL
247.182.249.253 20150601013354  NULL    NULL
135.124.143.193 20150601013818  NULL    NULL
222.131.187.37  20150601013957  NULL    NULL
```

```
...
```

3. Чтобы данные распарсились правильно, нужно воспользоваться таким regex:

```
"input.regex" = '^(\\S*)\\t\\t\\t(\\d{8})\\S*\\t(\\S*)\\t(\\S*)\\t.*$'
```

> **Задача.** Добавьте его в запрос и выполните, чтоб убедится, что данные распарсились правильно.

Примечание. Библиотека SerDe из `hive.contrib` имеет одну особенность. На выход после парсинга она выдаёт только строки тогда как в датасете имеются и числа. В случае, если в таблице есть числовые столбцы рекомендуется использовать `org.apache.hadoop.hive.serde2.RegexSerDe`. Эта реализация SerDe корректно парсит большинство типов данных Hive, но работает только для десериализации (т.е. для чтения данных). Синтаксис заросов при использовании этой версии SerDe почти не отличается. Единственное отличие - в пути к классу, который прописываем в `ROW FORMAT SERDE`.

Вывод информации про таблицу:

```
hive -S --database <YOUR_USER>_test -e 'DESCRIBE SerDeExample'
```

Новый аргумент: `-S` отключает логи, информацию о времени выполнения и т.д. Остаётся только результат запроса.

```
ip                      string                  from deserializer
date                    string                  from deserializer
request                 string                  from deserializer
responsecode            string                  from deserializer
```

6. Практика

> **Задача 0.** Посчитать кол-во различных масок подсети.

Решение.

```
ADD JAR /opt/cloudera/parcels/CDH/lib/hive/lib/hive-contrib.jar;
USE <YOUR_USER>_test;

SELECT COUNT(DISTINCT mask)
FROM Subnets;
```

С помощью ключевого слова `EXPLAIN` можно вывести план запроса. Там будет показано в какие MapReduce-Job'ы будет транслироваться запрос.

```
STAGE DEPENDENCIES:
  Stage-1 is a root stage
  Stage-0 depends on stages: Stage-1

STAGE PLANS:
  Stage: Stage-1
    Map Reduce
      Map Operator Tree:
          TableScan
            ...
      Reduce Operator Tree:
        Group By Operator
          ...

  Stage: Stage-0
    Fetch Operator
      ...
```

Если зайти на ApplicationMaster Web UI [http://mipt-master.atp-fivt.org:8088 (http://mipt-master.atp-fivt.org:8088)] можно увидеть, что запрос действительно транслируется в MapReduce-задачи.

> **Задача 1.** Посчитать кол-во адресов, имеющих маску 255.255.255.128.
>
> **Задача 2.** Посчитать среднее кол-во адресов по маскам. По каждой задаче выведите план запроса и посчитайте по нему кол-во MapReduce Job.

7. Hive streaming

Существует 2 основных способа использования внешних скриптов в Hive Streaming.

- использование команды,
- подключение внешних скриптов.

> **Пример.** Вывести 1-й октет IP-адресов из таблицы Subnets.

Решение 2-мя способами: /home/velkerr/seminars/mcs17_hive1/6-1-streaming_example

7.1. Отладка скриптов для Streaming

Внешние скрипты могут быть достаточно сложными, поэтому удобно сначала их отладить.

```
hive --database <YOUR_USER>_test -e 'SELECT * FROM SerDeExample LIMIT 10' | ./<yo
```

7.2. Практика

Входные данные - всё та же таблица подсетей (Subnets).

> **Задача 4.** Заменить в логах дату 20150601 на сегодняшнее число, используя Streaming.
>
> **Задача 5.** Перевести IP-адреса в численное представление. Для быстрого перевода можно воспользоваться таким Python-кодом: `struct.unpack("!I", socket.inet_aton(ip)`
>
> По каждой задаче выведите план запроса и посчитайте по нему кол-во MapReduce Job.