Бесплатная электронная книга

учусь

hive

Free unaffiliated eBook created from
Stack Overflow contributors.

#hive

Около

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: hive

It is an unofficial and free hive ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official hive.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

глава 1: Начало работы с улей

замечания

- Улей - это инструмент хранилища данных, построенный на вершине Hadoop .
- Он предоставляет SQL-подобный язык для запроса данных.
- Мы можем запускать почти все SQL-запросы в Hive, с той лишь разницей, что он запускает работу по сокращению карты на бэкэнд для получения результата из Hadoop Cluster. Из-за этого улей иногда занимает больше времени, чтобы получить результат.

Examples

Пример подсчета слов в улье

Файл Docs (входной файл)

У Мэри был маленький ягненок

его шерсть была белой, как снег

и везде, куда Мария пошла

ягненок обязательно поехал.

Уличный запрос

```
CREATE TABLE FILES (line STRING);

LOAD DATA INPATH 'docs' OVERWRITE INTO TABLE FILES;

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

Вывод таблицы word_counts в Hive

Мэри, 2

было, 1

а, 1

мало, 1

баранина, 2

его, 1

флис, 1

была, 2

белый, 1

а, 1

снег, 1

и, 1

везде, 1

что, 1

пошел, 1

, 1

Конечно, 1

к, 1

идти, 1

Установка Hive (linux)

Начните с загрузки последней стабильной версии с https://hive.apache.org/downloads.html

-> Теперь распакуйте файл с помощью

$ tar -xvf hive-2.xy-bin.tar.gz

-> Создать каталог в каталоге / usr / local / with

$ sudo mkdir / usr / local / hive

-> Переместить файл в корень с помощью

$ mv ~ / Downloads / hive-2.xy / usr / local / hive

-> Редактировать переменные среды для hadoop и hive в .bashrc

$ gedit ~ / .bashrc

как это

экспорт HIVE_HOME = / usr / local / hive / apache-hive-2.0.1-bin /

export PATH = $ PATH: $ HIVE_HOME / bin

export CLASSPATH = $ CLASSPATH: / usr / local / Hadoop / lib / * :.

export CLASSPATH = $ CLASSPATH: /usr/local/hive/apache-hive-2.0.1-bin/lib/* :.

-> Теперь, запустите hasoop, если он еще не запущен. И убедитесь, что он запущен, и он не находится в безопасном режиме.

$ hadoop fs -mkdir / пользователь / **улей** / **склад**

Каталог «склад» - это место для хранения таблицы или данных, относящихся к улью.

$ hadoop fs -mkdir / tmp

Временной каталог «tmp» является временным местом хранения промежуточного результата обработки.

-> Установить разрешения для чтения / записи в этих папках.

$ hadoop fs -chmod g + w / пользователь / **улей** / **склад**

$ hadoop fs -chmod g + w / user / tmp

-> Теперь активируйте HIVE с помощью этой команды в консоли

$ hive

Установка улья с использованием внешнего метастора в Linux

Предпосылки:

1. Java 7
2. Hadoop (см. Здесь для установки Hadoop)
3. Сервер и клиент Mysql

Монтаж:

Шаг 1: Загрузите последний архив из Hive с страницы загрузки .

Шаг 2: Извлеките загруженный tarball (***Предположение:*** *tarball загружается в $ HOME*)

```
tar -xvf /home/username/apache-hive-x.y.z-bin.tar.gz
```

Шаг 3: Обновите файл окружения (`~/.bashrc`)

```
export HIVE_HOME=/home/username/apache-hive-x.y.z-bin
export PATH=$HIVE_HOME/bin:$PATH
```

введите файл, чтобы установить новые переменные среды.

```
source ~/.bashrc
```

Шаг 4: Загрузите соединитель JDBC для MySQL из здесь и извлечь его.

```
tar -xvf mysql-connector-java-a.b.c.tar.gz
```

`mysql-connector-java-abcjar` каталог содержит jar-файл `mysql-connector-java-abcjar` .
Скопируйте его в `lib $HIVE_HOME`

```
cp mysql-connector-java-a.b.c.jar $HIVE_HOME/lib/
```

Конфигурация:

Создайте файл конфигурации `hive-site.xml $HIVE_HOME/conf/` и добавьте следующие свойства, связанные с метасторе.

```
<configuration>
   <property>
      <name>javax.jdo.option.ConnectionURL</name>
      <value>jdbc:mysql://localhost/hive_meta</value>
      <description>JDBC connect string for a JDBC metastore</description>
   </property>

   <property>
      <name>javax.jdo.option.ConnectionDriverName</name>
      <value>com.mysql.jdbc.Driver</value>
      <description>Driver class name for a JDBC metastore</description>
   </property>

   <property>
       <name>javax.jdo.option.ConnectionUserName</name>
       <value>mysqluser</value>
       <description>username to use against metastore database</description>
   </property>

   <property>
       <name>javax.jdo.option.ConnectionPassword</name>
       <value>mysqlpass</value>
       <description>password to use against metastore database</description>
   </property>

   <property>
       <name>datanucleus.autoCreateSchema</name>
       <value>false</value>
   </property>

   <property>
       <name>datanucleus.fixedDatastore</name>
       <value>true</value>
```

```
   </property>
</configuration>
```

Обновите значения MySQL «имя пользователя» и «пароль» соответственно в свойствах.

Создайте схему Metastore:

Скрипты схемы метастабильности доступны в разделе `$HIVE_HOME/scripts/metastore/upgrade/mysql/`

Войдите в MySQL и введите схему,

```
mysql -u username -ppassword

mysql> create database hive_meta;
mysql> use hive_meta;
mysql> source hive-schema-x.y.z.mysql.sql;
mysql> exit;
```

Запуск Metastore:

```
hive --service metastore
```

Чтобы запустить его в фоновом режиме,

```
nohup hive --service metastore &
```

Запуск HiveServer2: (используйте, если необходимо)

```
hiveserver2
```

Чтобы запустить его в фоновом режиме,

```
nohup hiveserver2 metastore &
```

Примечание. Эти исполняемые файлы доступны в разделе `$HIVE_HOME/bin/`

Подключение:

Используйте `hive` , `beeline` или Hue для соединения с Hive.

Hive CLI устарел, рекомендуется использовать Beeline или Hue.

Дополнительные конфигурации для оттенков:

Обновите это значение в `$HUE_HOME/desktop/conf/hue.ini`

```
[beeswax]
    hive_conf_dir=/home/username/apache-hive-x.y.z-bin/conf
```

Прочитайте Начало работы с улей онлайн: https://riptutorial.com/ru/hive/topic/1099/начало-работы-с-улей

глава 2: Вставить заявление

Синтаксис

- **Стандартный синтаксис:**
- INSERT OVERWRITE TABLE tablename1 [PARTITION (partcol1 = val1, partcol2 = val2 ...) [IF NOT EXISTS]] select_statement1 FROM from_statement;
- INSERT INTO TABLE tablename1 [PARTITION (partcol1 = val1, partcol2 = val2 ...)] select_statement1 FROM from_statement;
- INSERT INTO TABLE tablename1 [PARTITION (partcol1 = val1, partcol2 = val2 ...)] (z, y) select_statement1 FROM from_statement;
- **Расширение улья (несколько вставок):**
- FROM from_statement
  INSERT OVERWRITE TABLE tablename1 [PARTITION (partcol1 = val1, partcol2 = val2 ...) [ЕСЛИ НЕ СУЩЕСТВУЕТ]] select_statement1
  [INSERT OVERWRITE TABLE tablename2 [PARTITION ... [ЕСЛИ НЕ СУЩЕСТВУЕТ]] select_statement2]
  [INSERT INTO TABLE tablename2 [PARTITION ...] select_statement2] ...;
- FROM from_statement
  INSERT INTO TABLE tablename1 [PARTITION (partcol1 = val1, partcol2 = val2 ...)] select_statement1
  [INSERT INTO TABLE tablename2 [PARTITION ...] select_statement2]
  [INSERT OVERWRITE TABLE tablename2 [PARTITION ... [ЕСЛИ НЕ СУЩЕСТВУЕТ]] select_statement2] ...;
- **Расширение улья (динамические перегородки):**
- INSERT OVERWRITE TABLE tablename PARTITION (partcol1 [= val1], partcol2 [= val2] ...) select_statement FROM from_statement;
- INSERT INTO TABLE tablename PARTITION (partcol1 [= val1], partcol2 [= val2] ...) select_statement FROM from_statement;

замечания

вставить переписать
Оператор перезаписи вставки удаляет все существующие файлы в целевой таблице или разделе перед добавлением новых файлов на основе используемого оператора select. Обратите внимание, что при изменении структуры таблицы или в DML, используемой для

загрузки таблицы, иногда старые файлы не удаляются. При загрузке в таблицу с использованием динамического разбиения только разделы, определенные оператором select, будут перезаписаны. Любые ранее существовавшие разделы в целевом файле будут оставаться и не будут удалены.

вставлять в
Вставка в оператор добавляет новые данные в целевую таблицу на основе используемого оператора select.

Examples

вставить переписать

```
insert overwrite table yourTargetTable select * from yourSourceTable;
```

Вставить в таблицу

INSERT INTO добавит к таблице или разделу, сохранив существующие данные.

```
    INSERT INTO table yourTargetTable SELECT * FROM yourSourceTable;
```

Если таблица разделена, то мы можем вставлять ее в этот конкретный раздел статическим способом, как показано ниже.

```
    INSERT INTO TABLE yourTargetTable PARTITION (state=CA, city=LIVERMORE)
    select * FROM yourSourceTable;
```

Если таблица разделена, то мы можем вставить ее в этот раздел динамически, как показано ниже. Для создания динамических вставок разделов мы должны установить ниже свойства ниже.

```
    Dynamic Partition inserts are disabled by default. These are the relevant configuration
properties for dynamic partition inserts:
    SET hive.exec.dynamic.partition=true;
    SET hive.exec.dynamic.partition.mode=non-strict

    INSERT INTO TABLE yourTargetTable PARTITION (state=CA, city=LIVERMORE) (date,time)
    select * FROM yourSourceTable;
```

Несколько вставок из таблицы.

Расширение улья (несколько вставок):

```
    FROM table_name

    INSERT OVERWRITE TABLE table_one SELECT table_name.column_one,table_name.column_two
```

```
    INSERT OVERWRITE TABLE table_two SELECT table_name.column_two WHERE table_name.column_one
== 'something'
```

Прочитайте Вставить заявление онлайн: https://riptutorial.com/ru/hive/topic/1744/вставить-заявление

глава 3: Выписка SELECT

Синтаксис

- SELECT [ALL | DISTINCT] select_expr, select_expr, select_expr, ...
- FROM table_reference
- [WHERE where_condition]
- [GROUP BY col_list]
- [Имея условие]
- [ORDER BY col_list]
- [LIMIT n]

Examples

Выбрать все строки

`SELECT` используется для извлечения строк данных из таблицы. Вы можете указать, какие столбцы будут получены:

```
SELECT Name, Position
FROM Employees;
```

Или просто используйте * для получения всех столбцов:

```
SELECT *
FROM Employees;
```

Выберите конкретные строки

Этот запрос возвратит все столбцы из таблицы `sales` , где значение в столбце `amount` больше , чем 10 , и данных в `region` столбца в «США».

```
SELECT * FROM sales WHERE amount > 10 AND region = "US"
```

Вы можете использовать *регулярные выражения* для выбора столбцов, которые вы хотите получить. Следующий оператор получит данные из `name` столбца и всех столбцов, начинающихся с `address` префикса.

```
SELECT name, address.* FROM Employees
```

Вы также можете использовать ключевое слово `LIKE` (в сочетании с символом «%»), чтобы соответствовать строкам, которые начинаются или заканчиваются определенной подстрокой. Следующий запрос вернет все строки, где `city` столбца начинается с «New»

```
SELECT name, city FROM Employees WHERE city LIKE 'New%'
```

Вы можете использовать ключевое слово `RLIKE` для использования *регулярных выражений* Java. Следующий запрос будет возвращать строки, `name` столбцов которых содержат слова «кузнец» или «сын».

```
SELECT name, address FROM Employee WHERE name RLIKE '.*(smith|son).*'
```

Вы можете применить функции к возвращенным данным. Следующее предложение вернет все имя в верхнем регистре.

```
SELECT upper(name) FROM Employees
```

Вы можете использовать различные математические функции , функции сбора, функции преобразования типов, функции даты , условные функции или строковые функции .

Чтобы ограничить количество строк, заданных в результате, вы можете использовать ключевое слово `LIMIT` . Следующий оператор вернет только десять строк.

```
SELECT * FROM Employees LIMIT 10
```

Выбрать: выбранные столбцы проекта

Пример таблицы (например, Employee)

Название столбца	Тип данных
Я БЫ	INT
F_Name	STRING
L_Name	STRING
Телефон	STRING
Адрес	STRING

Проектировать все столбцы

Используйте wild card `*` для проецирования всех столбцов. например

```
Select * from Employee
```

Выбранные столбцы проекта (например, ID, Name)

Используйте имя столбцов в списке проекций. например

```
Select ID, Name from Employee
```

Отменить 1 столбец из списка проекций

Отображать все столбцы, кроме 1 колонки. например

```
Select `(ID)?+.+` from Employee
```

Отменить шаблон соответствия столбцам

Отклонить все столбцы, соответствующие шаблону. например Отклонить все столбцы, заканчивающиеся на `NAME`

```
Select `(.*NAME$)?+.+` from Employee
```

Прочитайте Выписка SELECT онлайн: https://riptutorial.com/ru/hive/topic/4133/выписка-select

глава 4: индексирование

Examples

Состав

```
CREATE INDEX index_name
ON TABLE base_table_name (col_name, ...)
AS 'index.handler.class.name'
[WITH DEFERRED REBUILD]
[IDXPROPERTIES (property_name=property_value, ...)]
[IN TABLE index_table_name]
[PARTITIONED BY (col_name, ...)]
[
 [ ROW FORMAT ...] STORED AS ...
 | STORED BY ...
]
[LOCATION hdfs_path]
[TBLPROPERTIES (...)]
```

Пример:

```
CREATE INDEX inedx_salary ON TABLE employee(salary) AS
'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler' WITH DEFERRED REBUILD;
```

Изменить индекс

ALTER INDEX index_name ON table_name [PARTITION (...)] REBUILD

Индекс падения

```
DROP INDEX <index_name> ON <table_name>
```

Если WITH WITH DEFERRED REBUILD указано в CREATE INDEX, то вновь созданный индекс изначально пуст (независимо от того, содержит ли таблица какие-либо данные).

Команда ALTER INDEX REBUILD может использоваться для построения структуры индекса для всех разделов или одного раздела.

Прочитайте индексирование онлайн: https://riptutorial.com/ru/hive/topic/6365/индексирование

глава 5: Пользовательские агрегированные функции (UDAF)

Examples

Пример UDAF

- Создайте класс Java, который расширяет `org.apache.hadoop.hive.ql.exec.hive.UDAF` Создайте внутренний класс, который реализует `UDAFEvaluator`
- Внедрение пяти методов
    - `init()` - Этот метод инициализирует оценщика и сбрасывает его внутреннее состояние. Мы используем новый столбец () в приведенном ниже коде, чтобы указать, что все значения еще не были агрегированы.
    - `iterate()` - этот метод вызывается каждый раз, когда создается новое значение для агрегирования. Оценщик должен обновить свое внутреннее состояние в результате выполнения агрегации (мы делаем сумму - см. Ниже). Мы возвращаем true, чтобы указать, что вход был действительным.
    - `terminatePartial()` - этот метод вызывается, когда Hive хочет получить результат для частичной агрегации. Метод должен возвращать объект, который инкапсулирует состояние агрегации.
    - `merge()` - этот метод вызывается, когда Hive решает объединить одну частичную агрегацию с другой.
    - `terminate()` - этот метод вызывается, когда необходим конечный результат агрегации.

```
public class MeanUDAF extends UDAF {
// Define Logging
static final Log LOG = LogFactory.getLog(MeanUDAF.class.getName());
public static class MeanUDAFEvaluator implements UDAFEvaluator {
/**
 * Use Column class to serialize intermediate computation
 * This is our groupByColumn
 */
public static class Column {
 double sum = 0;
 int count = 0;
 }
private Column col = null;
public MeanUDAFEvaluator() {
 super();
 init();
 }
// A - Initalize evaluator - indicating that no values have been
// aggregated yet.
public void init() {
```

```
 LOG.debug("Initialize evaluator");
 col = new Column();
 }
// B- Iterate every time there is a new value to be aggregated
 public boolean iterate(double value) throws HiveException {
 LOG.debug("Iterating over each value for aggregation");
 if (col == null)
 throw new HiveException("Item is not initialized");
 col.sum = col.sum + value;
 col.count = col.count + 1;
 return true;
 }
// C - Called when Hive wants partially aggregated results.
 public Column terminatePartial() {
 LOG.debug("Return partially aggregated results");
 return col;
 }
 // D - Called when Hive decides to combine one partial aggregation with another
 public boolean merge(Column other) {
 LOG.debug("merging by combining partial aggregation");
 if(other == null) {
 return true;
 }
 col.sum += other.sum;
 col.count += other.count;
 return true;
}
 // E - Called when the final result of the aggregation needed.
 public double terminate(){
 LOG.debug("At the end of last record of the group - returning final result");
 return col.sum/col.count;
 }
 }
}


hive> CREATE TEMPORARY FUNCTION <FUNCTION NAME> AS 'JAR PATH.jar';
hive> select id, mean_udf(amount) from table group by id;
```

Прочитайте Пользовательские агрегированные функции (UDAF) онлайн:
https://riptutorial.com/ru/hive/topic/5137/пользовательские-агрегированные-функции--udaf-

глава 6: Пользовательские функции пользователя Hive (UDF)

Examples

Создание UUF улья

Чтобы создать UDF, нам необходимо расширить класс UDF (`org.apache.hadoop.hive.ql.exec.UDF`) и реализовать метод оценки.

После выполнения UDF и создания JAR нам нужно добавить jar в контекст hive для создания временной / постоянной функции.

```
import org.apache.hadoop.hive.ql.exec.UDF;

class UDFExample extends UDF {

  public String evaluate(String input) {

    return new String("Hello " + input);
  }
}

hive> ADD JAR <JAR NAME>.jar;
hive> CREATE TEMPORARY FUNCTION helloworld as 'package.name.UDFExample';
hive> select helloworld(name) from test;
```

Улей UDF для обрезки данной строки.

```
package MyHiveUDFs;

import org.apache.commons.lang.StringUtils;
import org.apache.hadoop.hive.ql.exec.UDF;
import org.apache.hadoop.io.Text;

public class Strip extends UDF {

private Text result = new Text();
 public Text evaluate(Text str) {
 if(str == null) {
 return null;
 }
 result.set(StringUtils.strip(str.toString()));
 return result;
 }
}
```

экспортировать файл выше в jar

Перейдите в CLI Hive и добавьте UDF JAR

```
hive> ADD jar /home/cloudera/Hive/hive_udf_trim.jar;
```

Убедитесь, что JAR находится в интерфейсе Hive CLI Classpath

```
hive> list jars;
/home/cloudera/Hive/hive_udf_trim.jar
```

Создать временную функцию

```
hive> CREATE TEMPORARY FUNCTION STRIP AS 'MyHiveUDFs.Strip';
```

Выход UDF

```
 hive> select strip('   hiveUDF ') from dummy;
 OK
 hiveUDF
```

Прочитайте Пользовательские функции пользователя Hive (UDF) онлайн:
https://riptutorial.com/ru/hive/topic/4949/пользовательские-функции-пользователя-hive--udf-

глава 7: Пользовательские функции таблицы (UDTF)

Examples

Пример UDTF и использование

Пользовательские функции таблицы, представленные интерфейсом **org.apache.hadoop.hive.ql.udf.generic.GenericUDTF** . Эта функция позволяет выводить несколько строк и несколько столбцов для одного входа.

Мы должны переписать ниже методы:

```
 1.we specify input and output parameters
abstract StructObjectInspector initialize(ObjectInspector[] args)
                                                throws UDFArgumentException;

 2.we process an input record and write out any resulting records
abstract void process(Object[] record) throws HiveException;

3.function is Called to notify the UDTF that there are no more rows to process.
    Clean up code or additional output can be produced here.
abstract void close() throws HiveException;




import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import org.apache.hadoop.hive.ql.exec.UDFArgumentException;
import org.apache.hadoop.hive.ql.metadata.HiveException;
import org.apache.hadoop.hive.ql.udf.generic.GenericUDTF;
import org.apache.hadoop.hive.serde2.objectinspector.ObjectInspector;
import org.apache.hadoop.hive.serde2.objectinspector.ObjectInspectorFactory;
import org.apache.hadoop.hive.serde2.objectinspector.PrimitiveObjectInspector;
import org.apache.hadoop.hive.serde2.objectinspector.StructObjectInspector;
import
org.apache.hadoop.hive.serde2.objectinspector.primitive.PrimitiveObjectInspectorFactory;

public class NameParserGenericUDTF extends GenericUDTF {
      private PrimitiveObjectInspector stringOI = null;

       //Defining input argument as string.
      @Override
      public StructObjectInspector initialize(ObjectInspector[] args) throws
UDFArgumentException {
        if (args.length != 1) {
          throw new UDFArgumentException("NameParserGenericUDTF() takes exactly one
argument");
        }
```

```
        if (args[0].getCategory() != ObjectInspector.Category.PRIMITIVE
            && ((PrimitiveObjectInspector) args[0]).getPrimitiveCategory() !=
PrimitiveObjectInspector.PrimitiveCategory.STRING) {
          throw new UDFArgumentException("NameParserGenericUDTF() takes a string as a
parameter");
        }

        // input
        stringOI = (PrimitiveObjectInspector) args[0];

        // output
        List<String> fieldNames = new ArrayList<String>(2);
        List<ObjectInspector> fieldOIs = new ArrayList<ObjectInspector>(2);
        fieldNames.add("name");
        fieldNames.add("surname");
        fieldOIs.add(PrimitiveObjectInspectorFactory.javaStringObjectInspector);
        fieldOIs.add(PrimitiveObjectInspectorFactory.javaStringObjectInspector);
        return ObjectInspectorFactory.getStandardStructObjectInspector(fieldNames, fieldOIs);
      }

      public ArrayList<Object[]> processInputRecord(String name){
            ArrayList<Object[]> result = new ArrayList<Object[]>();

            // ignoring null or empty input
            if (name == null || name.isEmpty()) {
              return result;
            }

            String[] tokens = name.split("\\s+");

            if (tokens.length == 2){
                result.add(new Object[] { tokens[0], tokens[1] });
            }else if (tokens.length == 4 && tokens[1].equals("and")){
                result.add(new Object[] { tokens[0], tokens[3] });
                result.add(new Object[] { tokens[2], tokens[3] });
            }

            return result;
      }

      @Override
      public void process(Object[] record) throws HiveException {
        final String name = stringOI.getPrimitiveJavaObject(record[0]).toString();
        ArrayList<Object[]> results = processInputRecord(name);

        Iterator<Object[]> it = results.iterator();

        while (it.hasNext()){
            Object[] r = it.next();
            forward(r);
        }
      }

      @Override
      public void close() throws HiveException {
        // do nothing
      }
}
```

Упакуйте код в банку и добавьте jar в контекст hive.

```
hive> CREATE TEMPORARY FUNCTION process_names as 'jar.path.NameParserGenericUDTF';

Here we will pass input as full name and break it into first and last name.

hive> SELECT
   t.name,
   t.surname
FROM people
    lateral view process_names(name) t as name, surname;

Teena Carter
John Brownewr
```

Прочитайте Пользовательские функции таблицы (UDTF) онлайн:
https://riptutorial.com/ru/hive/topic/6502/пользовательские-функции-таблицы--udtf-

глава 8: Создание отчета о базе данных и таблицах

Синтаксис

- CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.] Имя_таблицы

  [(col_name data_type [COMMENT col_comment], ...)] [COMMENT table_comment] [PARTITIONED BY (col_name data_type [COMMENT col_comment], ...)] [CLUSTERED BY (col_name, col_name, ...) [SORTED BY (col_name [ASC | DESC], ...)] INTO num_buckets BUCKETS] [SKEWED BY (col_name, col_name, ...) - (Примечание: доступно в Hive 0.10.0 и новее)] ON ((col_value, col_value, ...), (col_value, col_value, ...), ...) [STORED AS DIRECTORIES] [[ROW FORMAT row_format] [STORED AS file_format] | STORED BY 'storage.handler.class.name' [WITH SERDEPROPERTIES (...)]] [LOCATION hdfs_path] [TBLPROPERTIES (property_name = property_value, ...)]
  [AS select_statement];

- CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.] Имя_таблицы LIKE existing_table_or_view_name [LOCATION hdfs_path];
- data_type: primitive_type, array_type, map_type, struct_type, union_type
- primitive_type: TINYINT, SMALLINT, INT, BIGINT, BOOLEAN, FLOAT, DOUBLE, STRING, BINARY, TIMESTAMP, DECIMAL, DECIMAL (точность, масштаб), DATE, VARCHAR, CHAR
- array_type: ARRAY <data_type>
- map_type: MAP <primitive_type, data_type>
- struct_type: STRUCT <col_name: data_type [COMMENT col_comment], ...>
- union_type: UNIONTYPE <data_type, data_type, ...>
- row_format: DELIMITED [FIELDS TERMINATED BY char [ESCAPED BY char]] [КОЛЛЕКЦИОННЫЕ ПУНКТЫ, ПРЕКРАЩЕННЫЕ char] [КЛЮЧИ КАРТЫ, ПРЕКРАЩЕННЫМИ char] [ЛИНИИ, ПРЕКРАЩЕННЫЕ char] [NULL DEFINED AS char] , SERDE serde_name [WITH SERDEPROPERTIES (property_name = property_value, property_name = property_value, ...)]
- file_format:: SEQUENCEFILE, TEXTFILE, RCFILE, ORC, PARQUET, AVRO, INPUTFORMAT input_format_classname OUTPUTFORMAT output_format_classname
- CREATE (DATABASE | SCHEMA) [IF NOT EXISTS] имя_базы_комментария [COMMENT database_comment] [LOCATION hdfs_path] [WITH DBPROPERTIES (property_name =

property_value, ...)];

замечания

При работе с таблицами и базами данных в HIVE. Ниже точки могут быть полезны.

- Мы можем переключать базу данных с `use database;` команда
- Чтобы узнать текущую рабочую базу данных, мы можем использовать `SELECT current_database()`
- Чтобы увидеть DDL, используемый для создания инструкции table, мы можем использовать `SHOW CREATE TABLE tablename`
- Чтобы увидеть все столбцы таблицы, используйте `DESCRIBE tablename` чтобы показать расширенные данные, такие как location serde used и другие `DESCRIBE FORMATTED tablename` . DESCRIBE также может быть сокращен как DESC.

Examples

Создать таблицу

Создание **управляемой** таблицы с разделом и сохранение в виде файла последовательности. Предполагается, что формат данных в файлах разделен по полю с помощью `Ctrl-A (^A)` и строки, разделенной символом новой строки. `hive.metastore.warehouse.dir` таблица создается в каталоге хранилища хранилища, указанном в значении для ключа `hive.metastore.warehouse.dir` в файле конфигурации `hive-site.xml` .

```
CREATE TABLE view
(time INT,
id BIGINT,
url STRING,
referrer_url STRING,
add STRING COMMENT 'IP of the User')
COMMENT 'This is view table'
PARTITIONED BY(date STRING, region STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\001'
STORED AS SEQUENCEFILE;
```

Создание **внешней** таблицы с разделами и сохранение в виде файла последовательности. Формат данных в файлах считается разделенным по полю `ctrl-A` и строковым разделителем по новой строке. Нижеследующая таблица создается в указанном месте и подходит, когда у нас уже есть данные. Одним из преимуществ использования внешней таблицы является то, что мы можем удалить таблицу без удаления данных. Например, если мы создаем таблицу и понимаем, что схема неверна, мы можем безопасно отказаться от таблицы и воссоздать ее с помощью новой схемы, не беспокоясь о данных. Другое преимущество заключается в том, что если мы используем другие инструменты, такие как свиньи в одних и тех же файлах, мы можем продолжать использовать их даже после

удаления таблицы.

```
CREATE EXTERNAL TABLE view
(time INT,
id BIGINT,
url STRING,
referrer_url STRING,
add STRING COMMENT 'IP of the User')
COMMENT 'This is view table'
PARTITIONED BY(date STRING, region STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\001'
STORED AS SEQUENCEFILE
LOCATION '<hdfs_location>';
```

Создание таблицы с использованием запроса выбора и **заполнения** результатов запроса, эти операторы известны как **CTAS (Create Table As Select)** .

В CTAS есть две части: SELECT-часть может быть любой инструкцией SELECT, поддерживаемой HiveQL. Часть CREATE CTAS берет результирующую схему из части SELECT и создает целевую таблицу с другими свойствами таблицы, такими как SerDe и формат хранения.

CTAS имеет следующие ограничения:

- Целевая таблица не может быть секционированной таблицей.
- Целевая таблица не может быть внешней таблицей.
- Целевая таблица не может быть таблицей балансировки списка.

```
CREATE TABLE new_key_value_store
ROW FORMAT SERDE "org.apache.hadoop.hive.serde2.columnar.ColumnarSerDe"
STORED AS RCFile
AS
SELECT * FROM page_view
SORT BY url, add;
```

Создать таблицу как:

Форма **LIKE CREATE TABLE** позволяет скопировать существующее определение таблицы точно (без копирования его данных). В отличие от CTAS, приведенная ниже инструкция создает новую таблицу, определение которой точно соответствует существующей таблице во всех деталях, отличных от имени таблицы. Новая таблица не содержит строк.

```
CREATE TABLE empty_page_views
LIKE page_views;
```

Создать базу данных

Создание базы данных в определенном месте. Если мы не укажем какое-либо место для базы данных, созданную в каталоге хранилища.

```
CREATE DATABASE IF NOT EXISTS db_name
COMMENT 'TEST DATABASE'
LOCATION /PATH/HDFS/DATABASE/;
```

Создание таблицы ACID улья.

Таблицы ACID поддерживаются с версии улья 0.14. Ниже таблицы поддерживает UPDATE / DELETE / INSERT

Ниже изменений конфигурации требуется в hive-site.xml.

```
 hive.support.concurrency = true
 hive.enforce.bucketing = true
 hive.exec.dynamic.partition.mode = nonstrict
 hive.txn.manager =org.apache.hadoop.hive.ql.lockmgr.DbTxnManager
 hive.compactor.initiator.on = true
 hive.compactor.worker.threads = 1
```

В настоящее время поддерживается только формат файла orc.

Оператор создания таблицы.

```
 create table Sample_Table(
 col1 Int,
 col2 String,
 col3 String)
 clustered by (col3) into 3 buckets
 stored as orc
 TBLPROPERTIES ('transactional'='true');
```

Интеграция HIVE_HBASE

Интеграция Hive-Hbase поддерживается с более низких версий. Улей: 0.11.0 HBase: 0.94.2 Hadoop: 0.20.2

```
CREATE TABLE hbase_hive
(id string,
 col1 string,
 col2 string,
 col3 int)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES
("hbase.columns.mapping" = ":key,cf1:col1,cf1:col2,cf1:col3")
TBLPROPERTIES ("hbase.table.name" = "hive_hbase");
```

Примечание: 1-й столбец должен быть ключевым столбцом.

Создайте таблицу, используя существующие свойства таблицы.

```
CREATE TABLE new_table_name LIKE existing_table_name;
```

Прочитайте Создание отчета о базе данных и таблицах онлайн:
https://riptutorial.com/ru/hive/topic/3328/создание-отчета-о-базе-данных-и-таблицах

глава 9: Создание таблицы улья через Sqoop

Вступление

Если у нас есть мета-магазин Hive, связанный с нашим кластером HDFS, Sqoop может импортировать данные в Hive, создав и выполнив инструкцию CREATE TABLE, чтобы определить макет данных в Hive. Импорт данных в Hive так же просто, как добавление опции -hive-import в вашу командную строку Sqoop.

замечания

Импорт данных непосредственно из РСУБД в систему «ВИЧ» может решить много времени. Также мы можем запустить запрос свободной формы (соединение или простой запрос) и заполнить его в таблице нашего выбора непосредственно в Hive.

--hive-import сообщает Sqoop, что конечным пунктом назначения является Hive, а не HDFS.

- опция «таблица» помогает импортировать данные в таблицу в выбранном нами улье, иначе она будет называться исходной таблицей, импортируемой из РСУБД.

Examples

Импорт куста с именем таблицы назначения в куст

```
$ sqoop import --connect jdbc:mysql://10.0.0.100/hadooptest
--username hadoopuser -P
--table table_name --hive-import --hive-table hive_table_name
```

Прочитайте Создание таблицы улья через Sqoop онлайн:
https://riptutorial.com/ru/hive/topic/10685/создание-таблицы-улья-через-sqoop

глава 10: Сценарий создания таблицы с примерными данными

Examples

Типы даты и времени

```
CREATE TABLE all_datetime_types(
  c_date date,
  c_timestamp timestamp
);
```

Минимальные и максимальные значения данных:

```
insert into all_datetime_types values ('0001-01-01','0001-01-01 00:00:00.000000001');
insert into all_datetime_types values ('9999-12-31','9999-12-31 23:59:59.999999999');
```

Типы текста

```
CREATE TABLE all_text_types(
  c_char char(255),
  c_varchar varchar(65535),
  c_string string
);
```

Пример данных:

```
insert into all_text_type values ('some ****&&&%%% char value ','some $$$$#####@@@@ varchar
value','some !!~~~++ string value' );
```

Числовые типы

```
CREATE TABLE all_numeric_types(
  c_tinyint tinyint,
  c_smallint smallint,
  c_int int,
  c_bigint bigint,
  c_decimal decimal(38,3)
);
```

Минимальные и максимальные значения данных:

```
insert into all_numeric_types values (-128,-32768,-2147483648,-9223372036854775808,-
99999999999999999999999999999999999.999);
insert into all_numeric_types values
(127,32767,2147483647,9223372036854775807,99999999999999999999999999999999999.999);
```

Числа с плавающей точкой

```
CREATE TABLE all_floating_numeric_types(
  c_float float,
  c_double double
);
```

Минимальные и максимальные значения данных:

```
insert into all_floating_numeric_types values (-3.4028235E38,-1.7976931348623157E308);
insert into all_floating_numeric_types values (-1.4E-45,-4.9E-324);
insert into all_floating_numeric_types values (1.4E-45,4.9E-324);
insert into all_floating_numeric_types values (3.4028235E38,1.7976931348623157E308);
```

Логические и двоичные типы

```
CREATE TABLE all_binary_types(
  c_boolean boolean,
  c_binary binary
);
```

Пример данных:

```
insert into all_binary_types values (0,1234);
insert into all_binary_types values (1,4321);
```

Замечания:

- Для boolean, внутренне он хранится как true или false.
- Для двоичного файла он будет хранить значение, закодированное base64.

Комплексные типы

ARRAY

```
CREATE TABLE array_data_type(
  c_array array<string>)
 ROW FORMAT DELIMITED
  FIELDS TERMINATED BY ','
  COLLECTION ITEMS TERMINATED BY '&';
```

Создайте `data.csv` с данными:

```
arr1&arr2
arr2&arr4
```

Поместите `data.csv` в `data.csv /tmp` и загрузите эти данные в Hive

```
LOAD DATA LOCAL INPATH '/tmp/data.csv' INTO TABLE array_data_type;
```

Или вы можете поместить этот CSV в HDFS в `/tmp` . Загружать данные из CSV на HDFS, используя

```
LOAD DATA INPATH '/tmp/data.csv' INTO TABLE array_data_type;
```

КАРТА

```
CREATE TABLE map_data_type(
  c_map map<int,string>)
 ROW FORMAT DELIMITED
  FIELDS TERMINATED BY ','
  COLLECTION ITEMS TERMINATED BY '&'
  MAP KEYS TERMINATED BY '#';
```

файл `data.csv` :

```
101#map1&102#map2
103#map3&104#map4
```

Загружать данные в улей:

```
LOAD DATA LOCAL INPATH '/tmp/data.csv' INTO TABLE map_data_type;
```

STRUCT

```
CREATE TABLE struct_data_type(
  c_struct struct<c1:smallint,c2:varchar(30)>)
 ROW FORMAT DELIMITED
  FIELDS TERMINATED BY ','
  COLLECTION ITEMS TERMINATED BY '&';
```

файл `data.csv` :

```
101&struct1
102&struct2
```

Загружать данные в улей:

```
LOAD DATA LOCAL INPATH '/tmp/data.csv' INTO TABLE struct_data_type;
```

UNIONTYPE

```
CREATE TABLE uniontype_data_type(
  c_uniontype uniontype<int, double, array<string>)
 ROW FORMAT DELIMITED
  FIELDS TERMINATED BY ','
  COLLECTION ITEMS TERMINATED BY '&';
```

файл `data.csv` :

```
0&10
1&10.23
2&arr1&arr2
```

Загружать данные в улей:

```
LOAD DATA LOCAL INPATH '/tmp/data.csv' INTO TABLE uniontype_data_type;
```

Прочитайте Сценарий создания таблицы с примерными данными онлайн:
https://riptutorial.com/ru/hive/topic/5067/сценарий-создания-таблицы-с-примерными-данными

глава 11: Форматы файлов в HIVE

Examples

SEQUENCEFILE

Храните данные в SEQUENCEFILE, если данные необходимо сжать. Вы можете импортировать текстовые файлы, сжатые Gzip или Bzip2, непосредственно в таблицу, хранящуюся как TextFile. Сжатие будет обнаружено автоматически, и файл будет распакован «на лету» во время выполнения запроса.

```
CREATE TABLE raw_sequence (line STRING)
STORED AS SEQUENCEFILE;
```

ORC

Формат файла Optimized Row Columnar (ORC) обеспечивает высокоэффективный способ хранения данных Hive. Он был разработан для преодоления ограничений других форматов файлов Hive. Использование файлов ORC повышает производительность, когда Hive считывает, записывает и обрабатывает данные. Файл ORC может содержать легкие индексы и фильтры цветения.

См. Https://cwiki.apache.org/confluence/display/Hive/LanguageManual+ORC

ORC - рекомендуемый формат для хранения данных в дистрибутиве HortonWorks.

```
CREATE TABLE tab_orc (col1 STRING,
                      col2 STRING,
                      col3 STRING)
STORED AS ORC
TBLPROPERTIES (
               "orc.compress"="SNAPPY",
               "orc.bloom.filter.columns"="col1",
               "orc.create.index" = "true"
              )
```

Чтобы изменить таблицу, чтобы новые разделы таблицы были сохранены в виде файлов ORC:

```
ALTER TABLE T SET FILEFORMAT ORC;
```

Начиная с Hive 0.14, пользователи могут запросить эффективное слияние небольших файлов ORC вместе, выпустив команду `CONCATENATE` на их таблицу или раздел. Файлы будут объединены на уровне полосы без ресериализатоина.

```
ALTER TABLE T [PARTITION partition_spec] CONCATENATE;
```

ПАРКЕТ

Формат столбчатого хранения паркета в Hive 0.13.0 и выше. Паркет построен с нуля со сложными вложенными структурами данных и использует алгоритм измельчения и сборки записей, описанный в документе Dremel. Мы считаем, что этот подход превосходит простое сглаживание вложенных пространств имен.

Паркет построен для поддержки очень эффективных схем сжатия и кодирования. Несколько проектов продемонстрировали влияние эффективности применения правильной схемы сжатия и кодирования к данным. Паркет позволяет использовать схемы сжатия для каждого столбца и надежно проверяется, чтобы добавить больше кодировок по мере их создания и реализации.

Рекомендуется использовать паркет. Формат файла с таблицами Impala в дистрибутивах Cloudera.

См .: http://parquet.apache.org/documentation/latest/

```
CREATE TABLE parquet_table_name (x INT, y STRING) STORED AS PARQUET;
```

AVRO

Файлы Avro поддерживаются в Hive 0.14.0 и более поздних версиях.

Avro - это схема удаленного вызова процедур и данных, разработанная в рамках проекта Hadoop от Apache. Он использует JSON для определения типов данных и протоколов и сериализует данные в компактном двоичном формате. Его основное использование - в Apache Hadoop, где он может обеспечить как формат сериализации для постоянных данных, так и формат проводов для связи между узлами Hadoop и клиентскими программами с услугами Hadoop.

Спецификация формата AVRO: https://avro.apache.org/docs/1.7.7/spec.html

```
CREATE TABLE kst
PARTITIONED BY (ds string)
STORED AS AVRO
TBLPROPERTIES (
  'avro.schema.url'='http://schema_provider/kst.avsc');
```

Мы также можем использовать синтаксис ниже, не используя файл схемы.

```
CREATE TABLE kst (field1 string, field2 int)
PARTITIONED BY (ds string)
STORED AS AVRO;
```

В приведенных выше примерах Предложение `STORED AS AVRO` эквивалентно:

```
ROW FORMAT SERDE
  'org.apache.hadoop.hive.serde2.avro.AvroSerDe'
STORED AS INPUTFORMAT
  'org.apache.hadoop.hive.ql.io.avro.AvroContainerInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.avro.AvroContainerOutputFormat'
```

Текстовый файл

TextFile - это формат файла по умолчанию, если параметр конфигурации hive.default.fileformat не имеет другого параметра. Мы можем создать таблицу на кусте, используя имена полей в нашем текстовом файле с разделителями. Например, наш файл csv содержит три поля (id, имя, зарплата), и мы хотим создать таблицу в улье под названием «сотрудники». Мы будем использовать приведенный ниже код для создания таблицы в улье.

```
CREATE TABLE employees (id int, name string, salary double) ROW FORMAT DELIMITED FIELDS
TERMINATED BY ',';
```

Теперь мы можем загрузить текстовый файл в нашу таблицу:

```
LOAD DATA LOCAL INPATH '/home/ourcsvfile.csv' OVERWRITE INTO TABLE employees;
```

Отображение содержимого нашей таблицы на улье, чтобы проверить, были ли данные успешно загружены:

```
SELECT * FROM employees;
```

Прочитайте Форматы файлов в HIVE онлайн: https://riptutorial.com/ru/hive/topic/4513/форматы-файлов-в-hive

глава 12: Экспорт данных в улей

Examples

Функция экспорта в улье

Экспорт данных из таблицы сотрудников в / tmp / ca_employees

INSERT OVERWRITE LOCAL DIRECTORY '/ tmp / ca_employees' SELECT имя, зарплата, адрес FROM сотрудников WHERE se.state = 'CA';

Экспорт данных из таблицы сотрудников в несколько локальных каталогов на основе определенного условия

В приведенном ниже примере показано, как единая конструкция может использоваться для экспорта данных в несколько каталогов на основе определенных критериев

```
FROM employees se INSERT OVERWRITE DIRECTORY '/tmp/or_employees' SELECT * WHERE se.cty = 'US'
and se.st = 'OR'
INSERT OVERWRITE DIRECTORY '/tmp/ca_employees' SELECT * WHERE se.cty = 'US' and se.st = 'CA'
INSERT OVERWRITE DIRECTORY '/tmp/il_employees' SELECT * WHERE se.cty = 'US' and se.st = 'IL';
```

Прочитайте Экспорт данных в улей онлайн: https://riptutorial.com/ru/hive/topic/6530/экспорт-данных-в-улей

кредиты

S. No	Главы	Contributors
1	Начало работы с улей	Bhavesh, Community, franklinsijo, johnnyaug, NeoWelkin
2	Вставить заявление	Jared, johnnyaug, Venkata Karthik
3	Выписка SELECT	Ambrish, dev, Jaime Caffarel, johnnyaug
4	индексирование	Prem Singh Bist
5	Пользовательские агрегированные функции (UDAF)	dev ヾ, Venkata Karthik
6	Пользовательские функции пользователя Hive (UDF)	Ashok, Panther, Venkata Karthik
7	Пользовательские функции таблицы (UDTF)	Venkata Karthik
8	Создание отчета о базе данных и таблицах	CodingInCircles, dev ヾ, goks, Panther, Venkata Karthik
9	Создание таблицы улья через Sqoop	NeoWelkin
10	Сценарий создания таблицы с примерными данными	dev ヾ
11	Форматы файлов в HIVE	agentv, Alex, Community, johnnyaug, leftjoin, Mzzzzzz, NeoWelkin, tomek, Venkata Karthik
12	Экспорт данных в улей	Prem Singh Bist