**Kostenloses eBook**

# LERNEN

# hive

Free unaffiliated eBook created from
**Stack Overflow contributors.**

#hive

# Inhaltsverzeichnis

# Über

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

# Kapitel 1: Erste Schritte mit Bienenstock

## Bemerkungen

- Hive ist ein Data Warehouse-Tool, das auf Hadoop basiert.
- Es bietet eine SQL-ähnliche Sprache zum Abfragen von Daten.
- Wir können fast alle SQL-Abfragen in Hive ausführen. Der einzige Unterschied besteht darin, dass im Backend ein Map-Reduction-Job ausgeführt wird, um das Ergebnis vom Hadoop-Cluster abzurufen. Aus diesem Grund benötigt Hive manchmal mehr Zeit, um die Ergebnismenge abzurufen.

## Examples

### Word Count Beispiel in Hive

**Docs-Datei (Eingabedatei)**

Mary hatte ein kleines Lamm

sein Vlies war schneeweiß

und überall wohin Mary ging

das Lamm würde sicher gehen.

**Hive-Abfrage**

```
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;
```

**Ausgabe der word_counts Tabelle in Hive**

Mary, 2

hatte, 1

a, 1

wenig, 1

Lamm, 2

seine, 1

Vlies, 1

war 2

weiß, 1

als, 1

Schnee, 1

und 1

überall 1

dass 1

ging, 1

die, 1

sicher, 1

bis 1

geh, 1

## Installation von Hive (Linux)

Laden Sie zunächst die neueste stabile Version von https://hive.apache.org/downloads.html herunter

-> Entpacken Sie jetzt die Datei mit

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

-> Erstellen Sie ein Verzeichnis in / usr / local / with

**$ sudo mkdir / usr / local / hive**

-> Verschieben Sie die Datei mit zu root

**$ mv ~ / downloads / hive-2.xy / usr / local / hive**

-> Bearbeiten Sie die Umgebungsvariablen für hadoop und bienenstock in .bashrc

**$ gedit ~ / .bashrc**

so was

export 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/* :.

-> Starten Sie hadoop, falls es noch nicht läuft. Stellen Sie sicher, dass es läuft und sich nicht im abgesicherten Modus befindet.

**$ hadoop fs -mkdir / user / hive / warehouse**

Das Verzeichnis "Warehouse" ist der Ort, an dem die Tabelle oder Daten gespeichert werden, die sich auf hive beziehen.

**$ hadoop fs -mkdir / tmp**

Das temporäre Verzeichnis „tmp“ ist der temporäre Speicherort für das Zwischenergebnis der Verarbeitung.

-> Legen Sie die Berechtigungen zum Lesen / Schreiben für diese Ordner fest.

**$ hadoop fs -chmod g + w / user / hive / warehouse**

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

-> Starten Sie HIVE mit diesem Befehl in der Konsole

**$ Bienenstock**

## Hive-Installation mit externem Metastore unter Linux

**Voraussetzungen:**

1. Java 7
2. Hadoop (Informationen zur Installation von Hadoop finden Sie hier )
3. Mein Server und Client

**Installation:**

Schritt 1: Laden Sie das neueste Hive-Archiv von der Downloadseite herunter .

Schritt 2: Extrahieren Sie das heruntergeladene Archiv ( ***Annahme:*** *Das Archiv wird in $ HOME heruntergeladen* )

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

Schritt 3: Aktualisieren Sie die Umgebungsdatei ( `~/.bashrc` )

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

Quelldatei die Datei, um die neuen Umgebungsvariablen festzulegen.

```
source ~/.bashrc
```

Schritt 4: Laden Sie den JDBC-Connector für MySQL von hier herunter und extrahieren Sie ihn.

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

Das extrahierte Verzeichnis enthält die Connector-JAR-Datei `mysql-connector-java-abcjar` . Kopieren Sie es in der `lib` von `$HIVE_HOME`

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

**Aufbau:**

Erstellen Sie die Hive-Konfigurationsdatei `hive-site.xml` im `hive-site.xml $HIVE_HOME/conf/` und fügen Sie die folgenden Eigenschaften für den Metastore hinzu.

```
<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>
```

Aktualisieren Sie die Werte von "Benutzername" und "Kennwort" von MySQL entsprechend in den

Eigenschaften.

**Erstellen Sie das Metastore-Schema:**

Die Metastore-Schemaskripts stehen unter `$HIVE_HOME/scripts/metastore/upgrade/mysql/`

Melden Sie sich bei MySQL an und beschaffen Sie das Schema.

```
mysql -u username -ppassword

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

**Metastore starten:**

```
hive --service metastore
```

Um es im Hintergrund auszuführen,

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

**HiveServer2 starten:** (ggf. verwenden)

```
hiveserver2
```

Um es im Hintergrund auszuführen,

```
nohup hiveserver2 metastore &
```

**Hinweis:** Diese ausführbaren Dateien sind unter `$HIVE_HOME/bin/` verfügbar.

**Verbinden:**

Verwenden Sie entweder `hive` , `beeline` oder hue , um eine Verbindung mit hive herzustellen.

Hive-CLI ist veraltet, es wird empfohlen, Beeline oder Hue zu verwenden.

**Zusätzliche Konfigurationen für Farbton:**

Aktualisieren Sie diesen Wert in `$HUE_HOME/desktop/conf/hue.ini`

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

Erste Schritte mit Bienenstock online lesen: https://riptutorial.com/de/hive/topic/1099/erste-schritte-mit-bienenstock

# Kapitel 2: Anweisung einfügen

## Syntax

- **Standardsyntax:**
- INSERT OVERWRITE TABLE tablename1 [PARTITION (partcol1 = val1, partcol2 = val2 ...) [WENN NICHT EXISTS]] select_statement1 FROM from_statement;
- INSERT INTO TABLE tabellenname1 [PARTITION (partcol1 = val1, partcol2 = val2 ...)] select_statement1 FROM from_statement;
- INSERT INTOABLE tabellenname1 [PARTITION (partcol1 = val1, partcol2 = val2 ...)] (z, y) select_statement1 FROM from_statement;
- **Hive-Erweiterung (mehrere Einfügungen):**
- FROM from_statement
  INSERT OVERWRITE TABLE tabellenname1 [PARTITION (partcol1 = val1, partcol2 = val2 ...) [IF NOT EXISTS]] select_statement1
  [INSERT OVERWRITE TABLE tablename2 [PARTITION ... [IF NOT EXISTS]] select_statement2]
  [INSERT INTO TABLE tabellenname2 [PARTITION ...] select_statement2] ...;
- FROM from_statement
  INSERT INTO TABLE tabellenname1 [PARTITION (partcol1 = val1, partcol2 = val2 ...)] select_statement1
  [INSERT INTO TABLE tabellenname2 [PARTITION ...] select_statement2]
  [INSERT OVERWRITE TABLE tablename2 [PARTITION ... [IF NOT EXISTS]] select_statement2] ...;
- **Hive-Erweiterung (Einfügen dynamischer Partitionen):**
- INSERT OVERWRITE TABLE Tabellenname PARTITION (partcol1 [= val1], partcol2 [= val2] ...) select_statement FROM from_statement;
- INSERT INTO TABLE Tabellenname PARTITION (partcol1 [= val1], partcol2 [= val2] ...) select_statement FROM from_statement;

## Bemerkungen

**Überschreiben einfügen**
Mit einer Überschreibanweisung zum Einfügen werden alle vorhandenen Dateien in der Zieltabelle oder -partition gelöscht, bevor neue Dateien basierend auf der verwendeten select-Anweisung hinzugefügt werden. Beachten Sie, dass bei Strukturänderungen an einer Tabelle oder an der zum Laden der Tabelle verwendeten DML die alten Dateien manchmal nicht gelöscht werden.

Beim Laden in eine Tabelle mit dynamischer Partitionierung werden nur die durch die select-Anweisung definierten Partitionen überschrieben. Vorhandene Partitionen im Ziel bleiben erhalten und werden nicht gelöscht.

**einfügen in**
Eine Einfügung in eine Anweisung hängt neue Daten in einer Zieltabelle an, die auf der verwendeten select-Anweisung basiert.

# Examples

## Überschreiben einfügen

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

## In die Tabelle einfügen

INSERT INTO wird an die Tabelle oder Partition angehängt, wobei die vorhandenen Daten erhalten bleiben.

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

Wenn eine Tabelle partitioniert ist, können wir diese Partition statisch einfügen, wie unten gezeigt.

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

Wenn eine Tabelle partitioniert ist, können wir sie wie unten gezeigt dynamisch in diese Partition einfügen. Um dynamische Partitionseinsätze durchzuführen, müssen Sie die folgenden Eigenschaften festlegen.

```
    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;
```

Mehrere Einfügungen in eine Tabelle.

Hive-Erweiterung (mehrere Einfügungen):

```
    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'
```

Anweisung einfügen online lesen: https://riptutorial.com/de/hive/topic/1744/anweisung-einfugen

# Kapitel 3: Benutzerdefinierte Aggregatfunktionen (UDAF)

## Examples

### UDAF bedeutet Beispiel

- Erstellen Sie eine Java-Klasse, die `org.apache.hadoop.hive.ql.exec.hive.UDAF` erweitert. `org.apache.hadoop.hive.ql.exec.hive.UDAF` Sie eine innere Klasse, die `UDAFEvaluator` implementiert
- Implementiere fünf Methoden
    - `init()` - Diese Methode initialisiert den Evaluator und setzt seinen internen Zustand zurück. Wir verwenden new Column () im folgenden Code, um anzuzeigen, dass noch keine Werte aggregiert wurden.
    - `iterate()` - Diese Methode wird jedes Mal aufgerufen, wenn ein neuer Wert aggregiert wird. Der Evaluator sollte seinen internen Zustand mit dem Ergebnis der Aggregation aktualisieren (wir machen eine Summe - siehe unten). Wir geben true zurück, um anzuzeigen, dass die Eingabe gültig war.
    - `terminatePartial()` - Diese Methode wird aufgerufen, wenn Hive ein Ergebnis für die Teilaggregation wünscht. Die Methode muss ein Objekt zurückgeben, das den Aggregationsstatus einkapselt.
    - `merge()` - Diese Methode wird aufgerufen, wenn Hive eine Teilaggregation mit einer anderen kombiniert.
    - `terminate()` - Diese Methode wird aufgerufen, wenn das Endergebnis der Aggregation benötigt wird.

```
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;
```

Benutzerdefinierte Aggregatfunktionen (UDAF) online lesen:
https://riptutorial.com/de/hive/topic/5137/benutzerdefinierte-aggregatfunktionen--udaf-

# Kapitel 4: Benutzerdefinierte Funktionen (UDFs) verwalten

## Examples

### Hive UDF-Erstellung

Um eine UDF zu erstellen, müssen wir die UDF-Klasse ( `org.apache.hadoop.hive.ql.exec.UDF` ) erweitern und die auswertende Methode implementieren.

Sobald UDF erfüllt ist und JAR erstellt ist, müssen wir dem Hive-Kontext jar hinzufügen, um eine temporäre / permanente Funktion zu erstellen.

```
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;
```

### Hive UDF, um die angegebene Zeichenfolge zu trimmen.

```
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;
 }
}
```

Exportieren Sie die obigen Dateien in eine JAR-Datei

Gehen Sie zur Hive-CLI und fügen Sie den UDF-JAR hinzu

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

Stellen Sie sicher, dass sich JAR im Hive-CLI-Klassenpfad befindet

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

Temporäre Funktion erstellen

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

UDF-Ausgabe

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

Benutzerdefinierte Funktionen (UDFs) verwalten online lesen:
https://riptutorial.com/de/hive/topic/4949/benutzerdefinierte-funktionen--udfs--verwalten

# Kapitel 5: Benutzerdefinierte Tabellenfunktionen (UDTFs)

## Examples

### UDTF-Beispiel und Verwendung

Benutzerdefinierte Tabellenfunktionen, dargestellt durch **die** Schnittstelle **org.apache.hadoop.hive.ql.udf.generic.GenericUDTF** . Diese Funktion ermöglicht die Ausgabe mehrerer Zeilen und Spalten für eine einzelne Eingabe.

Wir müssen die folgenden Methoden überschreiben:

```
 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
      }
}
```

Packen Sie den Code in jar und müssen Sie dem Hive-Kontext jar hinzufügen.

```
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
```

Benutzerdefinierte Tabellenfunktionen (UDTFs) online lesen:
https://riptutorial.com/de/hive/topic/6502/benutzerdefinierte-tabellenfunktionen--udtfs-

# Kapitel 6: Bienenstock-Erstellung durch Sqoop

## Einführung

Wenn wir einen Hive-Metastore für unseren HDFS-Cluster haben, kann Sqoop die Daten in Hive importieren, indem eine CREATE TABLE-Anweisung generiert und ausgeführt wird, um das Layout der Daten in Hive zu definieren. Das Importieren von Daten in Hive ist so einfach wie das Hinzufügen der Option --hive-import in die Sqoop-Befehlszeile.

## Bemerkungen

Der Import von Daten direkt aus RDBMS in HIVE kann viel Zeit sparen. Wir können auch eine Freiform-Abfrage (eine Verknüpfung oder eine einfache Abfrage) ausführen und diese in einer Tabelle unserer Wahl direkt in Hive einfügen.

--hive-import teilt Sqoop mit, dass das endgültige Ziel Hive und nicht HDFS ist.

--hive-table-Option hilft beim Importieren der Daten in die von uns gewählte Tabelle in der Struktur, andernfalls wird sie als die aus RDBMS importierte Quelltabelle benannt.

## Examples

### Hive-Import mit Zieltabellenname in Hive

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

Bienenstock-Erstellung durch Sqoop online lesen:
https://riptutorial.com/de/hive/topic/10685/bienenstock-erstellung-durch-sqoop

# Kapitel 7: Dateiformate in HIVE

## Examples

### SEQUENCEFILE

Speichern Sie Daten in SEQUENCEFILE, wenn die Daten komprimiert werden müssen. Sie können mit Gzip oder Bzip2 komprimierte Textdateien direkt in eine als TextFile gespeicherte Tabelle importieren. Die Komprimierung wird automatisch erkannt und die Datei wird während der Abfrageausführung schnell dekomprimiert.

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

### ORC

Das ORC-Dateiformat (Optimized Row Columnar) bietet eine äußerst effiziente Möglichkeit, Hive-Daten zu speichern. Es wurde entwickelt, um die Einschränkungen der anderen Hive-Dateiformate zu überwinden. Die Verwendung von ORC-Dateien verbessert die Leistung, wenn Hive Daten liest, schreibt und verarbeitet. ORC-Dateien können einfache Indizes und Bloom-Filter enthalten.

Siehe: https://cwiki.apache.org/confluence/display/Hive/LanguageManual+ORC

ORC ist ein empfohlenes Format zum Speichern von Daten in der HortonWorks-Distribution.

```
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"
              )
```

So ändern Sie eine Tabelle, sodass neue Partitionen der Tabelle als ORC-Dateien gespeichert werden:

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

Ab Hive 0.14 können Benutzer eine effiziente Zusammenführung kleiner ORC-Dateien `CONCATENATE` indem sie `CONCATENATE` Befehl `CONCATENATE` für ihre Tabelle oder Partition ausgeben. Die Dateien werden auf der Stripe-Ebene zusammengefügt, ohne dass eine erneute Registrierung erforderlich ist.

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

## PARKETT

Säulenspeicherformat für Parkett in Hive 0.13.0 und höher. Parkett wurde von Grund auf für komplexe verschachtelte Datenstrukturen entwickelt und verwendet den im Dremel-Papier beschriebenen Algorithmus zur Datensatzreduzierung und -montage. Wir glauben, dass dieser Ansatz der einfachen Abflachung von verschachtelten Namensräumen überlegen ist.

Parkett unterstützt sehr effiziente Kompressions- und Codierungsschemata. Mehrere Projekte haben gezeigt, wie sich die richtige Auswirkung der Anwendung des richtigen Kompressions- und Codierungsschemas auf die Daten auswirkt. Parkett ermöglicht die Festlegung von Kompressionsschemata auf Spaltenebene und ist zukunftssicher, so dass beim Erfassen und Implementieren weitere Codierungen hinzugefügt werden können.

Parkett wird als Dateiformat mit Impala-Tabellen in Cloudera-Distributionen empfohlen.

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

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

## AVRO

Avro-Dateien werden in Hive 0.14.0 und höher unterstützt.

Avro ist ein Remote-Prozeduraufruf- und Datenserialisierungs-Framework, das innerhalb des Hadoop-Projekts von Apache entwickelt wurde. Es verwendet JSON zur Definition von Datentypen und Protokollen und serialisiert Daten in einem kompakten Binärformat. Sein Hauptzweck ist Apache Hadoop, wo es sowohl ein Serialisierungsformat für persistente Daten als auch ein Drahtformat für die Kommunikation zwischen Hadoop-Knoten und von Client-Programmen zu den Hadoop-Diensten bereitstellen kann.

Spezifikation des AVRO-Formats: 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');
```

Die folgende Syntax kann auch ohne Schemadatei verwendet werden.

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

In den Beispielen oben ist die Klausel von `STORED AS AVRO` äquivalent zu:

```
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'
```

## Textdatei

TextFile ist das Standarddateiformat, es sei denn, der Konfigurationsparameter hive.default.fileformat hat eine andere Einstellung. Wir können eine Tabelle mit den Feldnamen in unserer Textdatei mit Trennzeichen erstellen. Nehmen wir zum Beispiel an, unsere CSV-Datei enthält drei Felder (ID, Name, Gehalt) und wir möchten eine Tabelle mit dem Namen "Mitarbeiter" erstellen. Wir werden den folgenden Code verwenden, um die Tabelle in Hive zu erstellen.

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

Nun können wir eine Textdatei in unsere Tabelle laden:

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

Anzeigen des Inhalts unserer Tabelle in hive, um zu überprüfen, ob die Daten erfolgreich geladen wurden:

```
SELECT * FROM employees;
```

Dateiformate in HIVE online lesen: https://riptutorial.com/de/hive/topic/4513/dateiformate-in-hive

# Kapitel 8: Daten in Hive exportieren

## Examples

### Exportfunktion im Bienenstock

**Exportieren von Daten aus der Employee-Tabelle in / tmp / ca_employees**

INSERT OVERWRITE LOCAL DIRECTORY '/ tmp / ca_employees' SELECT Name, Gehalt, Adresse FROM Mitarbeiter WHERE se.state = 'CA';

**Exportieren von Daten aus der Employee-Tabelle in mehrere lokale Verzeichnisse, basierend auf bestimmten Bedingungen**

Die folgende Abfrage zeigt, wie ein einzelnes Konstrukt verwendet werden kann, um Daten basierend auf bestimmten Kriterien in mehrere Verzeichnisse zu exportieren

```
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';
```

Daten in Hive exportieren online lesen: https://riptutorial.com/de/hive/topic/6530/daten-in-hive-exportieren

# Kapitel 9: Erstellen Sie eine Datenbank- und Tabellenanweisung

## Syntax

- CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [Datenbankname.] Tabellenname

  [(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, ...) - (Hinweis: Verfügbar in Struktur 0.10.0 und höher)] ON ((col_value, col_value, ...), (col_value, col_value, ...), ...) [ALS VERZEICHNIS GESPEICHERT] [[ROW FORMAT row_format] [STORED AS file_format] | GESPEICHERT VON 'storage.handler.class.name' [WITH SERDEPROPERTIES (...)]] [LOCATION hdfs_path] [TBLPROPERTIES (property_name = property_value, ...)] [AS select_statement];

- CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [Datenbankname]. Tabellenname LIKE bestehende_Tabelle_oder_Ansichtsname [LOCATION hdfs_path];

- data_type: primitiv_type, array_type, map_type, struct_type, union_type

- primitive_type: TINYINT, SMALLINT, INT, BIGINT, BOOLEAN, FLOAT, DOUBLE, STRING, BINARY, TIMESTAMP, DECIMAL, DECIMAL (Präzision, Skala), DATE, VARCHAR, CHAR

- array_type: ARRAY <data_type>

- map_type: MAP <primitiver_Typ, Daten_Typ>

- struct_type: STRUCT <col_name: data_type [COMMENT col_comment], ...>

- union_type: UNIONTYPE <Datentyp, Datentyp, ...>

- row_format: DELIMITED [FELDER BEENDET DURCH char [ABGEBOGEN DURCH char] [Sammlungsgegenstände bei char] [KARTENSCHLÜSSEL BEFINDET DURCH char] [LINES TER by DURCH char] [NULL DEFINED AS char]
  SERDE-Serienname [WITH SERDEPROPERTIES (Eigenschaftsname = Eigenschaftswert, Eigenschaftsname = Eigenschaftswert, ...)]

- Dateiformat:: SEQUENCEFILE, TEXTFILE, RCFILE, ORC, PARQUET, AVRO, INPUTFORMAT input_format_classname OUTPUTFORMAT output_format_classname

- CREATE (DATABASE | SCHEMA) [IF NOT EXISTS] Datenbankname [COMMENT Datenbankkommentar] [LOCATION hdfs_path] [WITH DBPROPERTIES (Eigenschaftsname = Eigenschaftswert, ...)];

# Bemerkungen

Beim Arbeiten mit Tabellen und Datenbanken in HIVE. Nachfolgende Punkte können nützlich sein.

- Wir können die Datenbank mit `use database;` wechseln `use database;` Befehl
- Um die aktuelle Arbeitsdatenbank kennen zu lernen, können Sie `SELECT current_database()`
- Um die DDL `SHOW CREATE TABLE tablename` die für die Anweisung create table verwendet wird, können Sie den Tabellennamen `SHOW CREATE TABLE tablename`
- Um alle Tabellenspalten `DESCRIBE tablename` , verwenden Sie `DESCRIBE tablename` , um erweiterte Details `DESCRIBE FORMATTED tablename` . B. die verwendeten Standortserver und andere. DESCRIBE kann auch als DESC abgekürzt werden.

# Examples

## Tabelle erstellen

Eine **verwaltete** Tabelle mit Partition erstellen und als Sequenzdatei speichern. Es wird angenommen, dass das Datenformat in den Dateien durch `Ctrl-A (^A)` feldbegrenzt und durch Zeilenumbruch getrennt wird. Die folgende Tabelle wird im Hive-Warehouse-Verzeichnis erstellt, das in value für den Schlüssel `hive.metastore.warehouse.dir` in der Hive-Konfigurationsdatei `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;
```

Erstellen einer **externen** Tabelle mit Partitionen und als Sequenzdatei gespeichert. Es wird angenommen, dass das Datenformat in den Dateien durch `ctrl-A` und durch Zeilenumbruch durch Zeilenumbrüche begrenzt wird. Die folgende Tabelle wird an dem angegebenen Ort erstellt und ist praktisch, wenn bereits Daten vorhanden sind. Die Verwendung einer externen Tabelle hat den Vorteil, dass wir die Tabelle löschen können, ohne die Daten zu löschen. Wenn wir zum Beispiel eine Tabelle erstellen und erkennen, dass das Schema falsch ist, können wir die Tabelle sicher löschen und mit dem neuen Schema neu erstellen, ohne sich um die Daten zu sorgen. Ein weiterer Vorteil besteht darin, dass wir andere Werkzeuge wie pig in denselben Dateien verwenden. Wir können sie auch nach dem Löschen der Tabelle weiter verwenden.

```
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>';
```

Erstellen Sie eine Tabelle mit select-Abfrage und **füllen Sie die** Ergebnisse aus der Abfrage aus. Diese Anweisungen werden als **CTAS (Create Table As Select) bezeichnet** .

CTAS besteht aus zwei Teilen: Der SELECT-Teil kann eine beliebige von HiveQL unterstützte SELECT-Anweisung sein. Der CREATE-Teil des CTAS entnimmt das resultierende Schema aus dem SELECT-Teil und erstellt die Zieltabelle mit anderen Tabelleneigenschaften wie SerDe und Speicherformat.

CTAS hat diese Einschränkungen:

- Die Zieltabelle kann keine partitionierte Tabelle sein.
- Die Zieltabelle kann keine externe Tabelle sein.
- Die Zieltabelle kann keine List-Bucketing-Tabelle sein.

```
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;
```

Tabelle erstellen wie:

Mit der **LIKE-Form von CREATE TABLE** können Sie eine vorhandene Tabellendefinition genau kopieren (ohne ihre Daten zu kopieren). Im Gegensatz zu CTAS erstellt die folgende Anweisung eine neue Tabelle, deren Definition in allen Details außer dem Tabellennamen genau mit der vorhandenen Tabelle übereinstimmt. Die neue Tabelle enthält keine Zeilen.

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

## Datenbank erstellen

An einem bestimmten Ort eine Datenbank erstellen. Wenn wir keinen Speicherort für die Datenbank angeben, wird dies im Warehouse-Verzeichnis erstellt.

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

## Hive ACID-Tabellenerstellung.

ACID-Tabellen werden seit Version 0.14 unterstützt. Die folgende Tabelle unterstützt UPDATE / DELETE / INSERT

Folgende Konfigurationsänderungen sind in hive-site.xml erforderlich.

```
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
```

Derzeit wird nur die orc-Datei unterstützt.

Tabelle zum Erstellen von Anweisungen

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

## HIVE_HBASE-Integration

Die Hive-Hbase-Integration wird seit den folgenden Versionen unterstützt. Bienenstock: 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");
```

Hinweis: Die erste Spalte sollte die Schlüsselspalte sein.

## Erstellen Sie eine Tabelle mit den vorhandenen Tabelleneigenschaften.

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

Erstellen Sie eine Datenbank- und Tabellenanweisung online lesen: https://riptutorial.com/de/hive/topic/3328/erstellen-sie-eine-datenbank--und-tabellenanweisung

# Kapitel 10: Indizierung

## Examples

### Struktur

```
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 (...)]
```

**Beispiel:**

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

**Index ändern**

ALTER INDEX Indexname ON Tabellenname [PARTITION (...)] REBUILD

**Index löschen**

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

Wenn in CREATE INDEX WITH DEFERRED REBUILD angegeben ist, ist der neu erstellte Index zunächst leer (unabhängig davon, ob die Tabelle Daten enthält).

Mit dem Befehl ALTER INDEX REBUILD kann die Indexstruktur für alle Partitionen oder eine einzelne Partition erstellt werden.

Indizierung online lesen: https://riptutorial.com/de/hive/topic/6365/indizierung

# Kapitel 11: SELECT-Anweisung

## Syntax

- SELECT [ALL | DISTINCT] select_expr, select_expr, select_expr,….
- FROM Tabellenreferenz
- [WO wo_bedingung]
- [GROUP BY col_list]
- [Mit der Bedingung]
- [ORDER BY col_list]
- [LIMIT n]

## Examples

### Alle Zeilen auswählen

`SELECT` wird verwendet, um Datenzeilen aus einer Tabelle abzurufen. Sie können angeben, welche Spalten abgerufen werden sollen:

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

Oder benutze einfach *, um alle Spalten zu erhalten:

```
SELECT *
FROM Employees;
```

### Wählen Sie bestimmte Zeilen aus

Diese Abfrage wird alle Spalten aus der Tabelle zurückzukehren `sales` , wo die Werte in der Spalte `amount` größer als 10 und die Daten in der `region` Spalte in „US“.

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

Sie können *reguläre Ausdrücke verwenden* , um die gewünschten Spalten auszuwählen. Die folgende Anweisung werden die Daten aus der Spalte erhalten `name` und alle Spalten mit dem Präfix beginnen `address` .

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

Sie können auch das Schlüsselwort `LIKE` (kombiniert mit dem Zeichen '%') verwenden, um Zeichenfolgen abzugleichen, die mit einer bestimmten Teilzeichenfolge beginnen oder enden. Die folgende Abfrage wird wieder alle Zeilen , in denen die Spalte `city` beginnt mit „Neu“

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

Sie können das Schlüsselwort `RLIKE` , um *reguläre* Java- *Ausdrücke zu verwenden* . Die folgende Abfrage wird wieder Zeilen , die `name` die Worte „smith“ oder „Sohn“ enthalten.

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

Sie können Funktionen auf die zurückgegebenen Daten anwenden. Der folgende Satz gibt alle Namen in Großbuchstaben zurück.

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

Sie können verschiedene mathematische Funktionen , Erfassungsfunktionen , Typumwandlungsfunktionen , Datumsfunktionen , Bedingungsfunktionen oder Stringfunktionen verwenden .

Um die Anzahl der in result angegebenen Zeilen zu begrenzen, können Sie das Schlüsselwort `LIMIT` . Die folgende Anweisung gibt nur zehn Zeilen zurück.

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

## Wählen Sie: Ausgewählte Spalten projizieren

### Beispieltabellenstruktur (zB Mitarbeiter)

| Spaltenname | Datentyp |
|---|---|
| ICH WÜRDE | INT |
| F_Name | STRING |
| L_Name | STRING |
| Telefon | STRING |
| Adresse | STRING |

### Projizieren Sie alle Spalten

Verwenden Sie Platzhalter `*` , um alle Spalten zu projizieren. z.B

```
Select * from Employee
```

### Projekt ausgewählte Spalten (zB ID, Name)

Verwenden Sie den Namen der Spalten in der Projektionsliste. z.B

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

### Verwerfen Sie eine Spalte aus der Projektionsliste

Zeigt alle Spalten außer einer Spalte an. z.B

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

### Spalten mit übereinstimmendem Muster verwerfen

Lehnt alle Spalten ab, die dem Muster entsprechen. zB Alle Spalten mit `NAME`

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

SELECT-Anweisung online lesen: https://riptutorial.com/de/hive/topic/4133/select-anweisung

# Kapitel 12: Tabellenerstellungsskript mit Beispieldaten

## Examples

### Datums- und Zeitstempeltypen

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

Minimale und maximale Datenwerte:

```
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');
```

### Textarten

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

Beispieldaten:

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

### Numerische Typen

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

Minimale und maximale Datenwerte:

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

### Gleitkommazahltypen

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

Minimale und maximale Datenwerte:

```
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);
```

### Boolesche und binäre Typen

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

Beispieldaten:

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

## Hinweis:

- Bei Boolean wird es intern als wahr oder falsch gespeichert.
- Bei Binärdateien werden Base64-codierte Werte gespeichert.

### Komplexe Typen

# ARRAY

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

Erstellen Sie `data.csv` mit Daten:

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

`data.csv` Sie die `data.csv` im `data.csv /tmp` und laden Sie diese Daten in Hive

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

Oder Sie können diese CSV in HDFS sagen, zB bei `/tmp` . Laden Sie Daten von CSV in HDFS mit

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

---

# KARTE

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

Datei `data.csv` :

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

Daten in den Bienenstock laden:

```
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 '&';
```

Datei `data.csv` :

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

Daten in den Bienenstock laden:

```
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 '&';
```

Datei `data.csv` :

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

Daten in den Bienenstock laden:

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

---

Tabellenerstellungsskript mit Beispieldaten online lesen:
https://riptutorial.com/de/hive/topic/5067/tabellenerstellungsskript-mit-beispieldaten

# Credits

| S. No | Kapitel | Contributors |
|---|---|---|
| 1 | Erste Schritte mit Bienenstock | Bhavesh, Community, franklinsijo, johnnyaug, NeoWelkin |
| 2 | Anweisung einfügen | Jared, johnnyaug, Venkata Karthik |
| 3 | Benutzerdefinierte Aggregatfunktionen (UDAF) | dev ツ, Venkata Karthik |
| 4 | Benutzerdefinierte Funktionen (UDFs) verwalten | Ashok, Panther, Venkata Karthik |
| 5 | Benutzerdefinierte Tabellenfunktionen (UDTFs) | Venkata Karthik |
| 6 | Bienenstock-Erstellung durch Sqoop | NeoWelkin |
| 7 | Dateiformate in HIVE | agentv, Alex, Community, johnnyaug, leftjoin, Mzzzzzz, NeoWelkin, tomek, Venkata Karthik |
| 8 | Daten in Hive exportieren | Prem Singh Bist |
| 9 | Erstellen Sie eine Datenbank- und Tabellenanweisung | CodingInCircles, dev ツ, goks, Panther, Venkata Karthik |
| 10 | Indizierung | Prem Singh Bist |
| 11 | SELECT-Anweisung | Ambrish, dev, Jaime Caffarel, johnnyaug |
| 12 | Tabellenerstellungsskript mit Beispieldaten | dev ツ |