华信培训

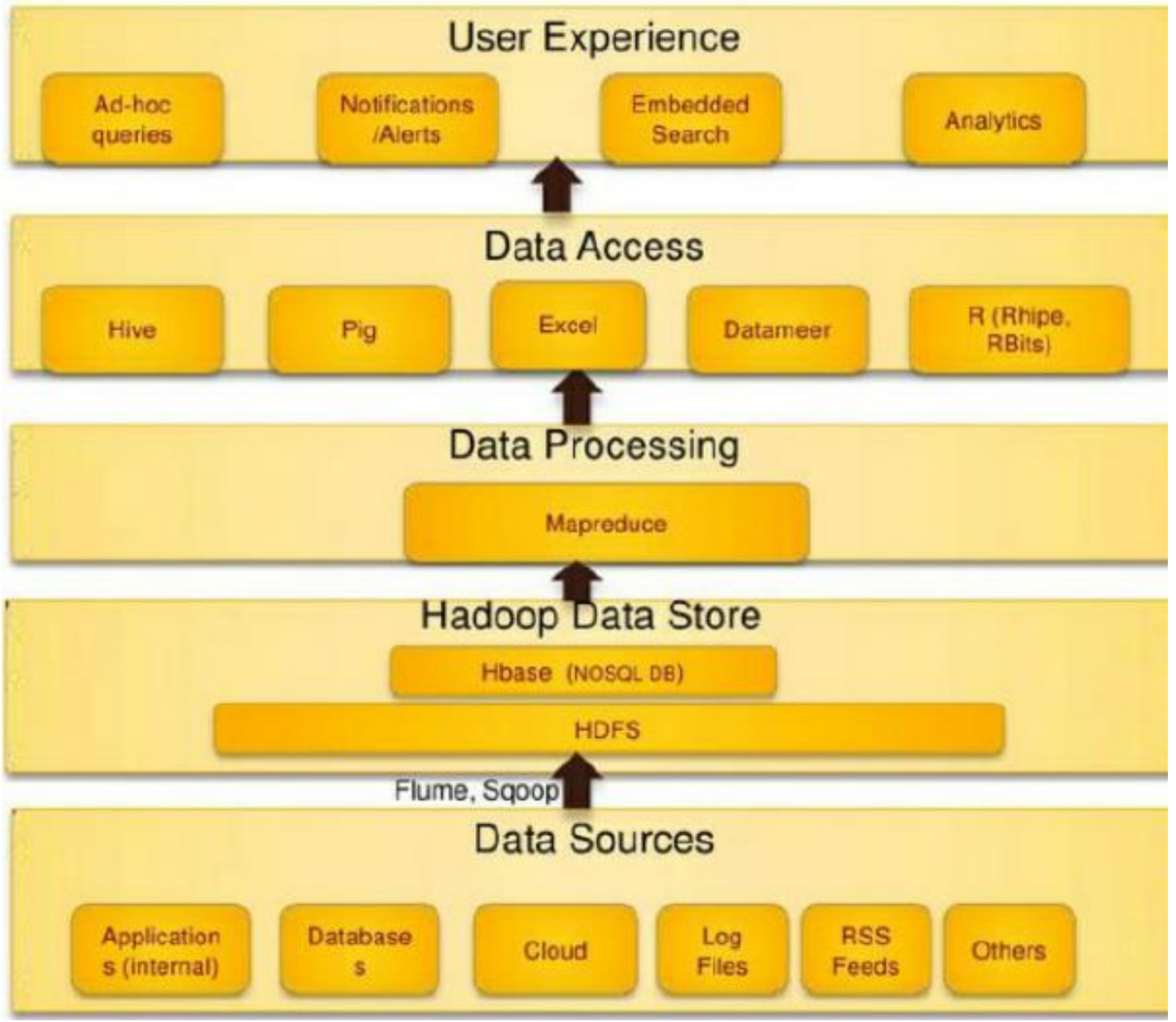
Hive概述

- Hive是基于Hadoop的数据仓库平台，由Facebook贡献，Hive提供了一系列的工具，可以用来进行数据提取、转化、加载（ETL），这是一种可以存储、查询和分析存储在 Hadoop 中的大规模数据的机制。
- Hive 定义了简单的类SQL 查询语言，称为 HQL，hive设计目的是让SQL技能良好，但Java技能较弱的分析师可以查询海量数据。
- Hive是SQL解析引擎，它将SQL语句转译成M/R Job，可以认为hive是HQL到MR的语言翻译器。Hive中数据计算使用MR，数据存储使用HDFS 。
- Hive的HQL表达能力有限，有些复杂运算还需要编写MR程序，hive允许熟悉 MapReduce 开发人员开发自定义的 mapper 和 reducer 来处理内建的 mapper 和 reducer无法完成的复杂的分析工作。
- Hive不是一个完整的数据库，Hadoop以及HDFS的设计本身约束和限制性地限制了Hive，使Hive不支持记录级别的更新或者删除操作。但用户可以通过查询生成新表或者将查询结果导入到文件中。
- Hadoop是一个面向批处理的系统，而MapReduce任务的启动过程需要消耗较长的时间，所以Hive查询延时比较严重。

Hadoop的缺点

- 过于底层
  - 不够灵活，受语言约束笨重不堪
  - 数据操作很不清晰，代码量大，难维护
  - 常见操作如排序、连接繁琐，且不高效
- 缺乏高级抽象
  - 程序关注点是如何实现，而不是要实现什么
  - 大量时间浪费在调试无用的细节上
- 常见的操作是对原有的数据进行不断地转化、综合、分析
  - 简化转化步骤
  - 存在常见的模式，例如排序、分组等
  - 希望将关注点提高到统计分析上
  - 提高重用性
- 在WordCount例子中：
  - SELECT word, count(*) FROM data GROUP BY word;
  - SELECT word, count(*) counter FROM data GROUP BY word HAVING counter >= 10 ORDER BY counter;

Hadoop生态图中的Hive



Hive优缺点

- 优点
  - Hive 使用类SQL 查询语法, 最大限度的实现了和SQL标准的兼容，大大降低了传统数据分析人员学习成本；
  - 使用JDBC 接口/ODBC接口，开发人员更易开发应用；
  - 以MR 作为计算引擎、HDFS 作为存储系统，解决了传统的关系型数据库在大数据处理上的瓶颈。
  - 统一的元数据管理（Derby、MySql等），并可与Pig 、Presto 等共享；
  - 并行计算，充分利用集群的CPU计算资源、存储资源，处理大规模数据集。
- 缺点
  - Hive的HQL表达的能力有限
  - 迭代式算法无法表达
  - 有些复杂运算用HQL不易表达
  - Hive效率较低
  - Hive自动生成MapReduce作业，通常不够智能；
  - HQL调优困难，粒度较粗
  - 可控性差

应用场景

- 适用场景
  - 海量数据的存储处理
  - 数据挖掘
  - 海量数据的离线分析
- 不适用场景
  - 复杂的机器学习算法
  - 复杂的科学计算
  - 联机交互式实时查询

Hive和Hadoop

- Hive 构建在 Hadoop 之上
- HQL 中对查询语句的解释、优化、生成查询计划是由 Hive 完成的
- 所有的数据都是存储在 Hadoop 中
- 查询计划被转化为 MapReduce 任务，在 Hadoop 中执行
- Hadoop和Hive都是用UTF-8编码的

Hive和RDBMS比较

	Hive	RDBMS
查询语言	HQL	SQL
数据存储	HDFS	Raw Device or Local FS
数据更新	不支持	支持
索引	新版本有，但较弱	有
执行	MapReduce	Executor
执行延迟	高	低
可扩展性	高	低
数据规模	大	小

Hive设计特征

- Hive 做为Hadoop 的数据仓库处理工具，它所有的数据都存储在Hadoop 兼容的文件系统中。Hive 在加载数据过程中不会对数据进行任何的修改，只是将数据复制或移动到HDFS 中Hive 设定的目录下，因此，Hive 不支持对数据的改写，所有的数据都是在加载的时候确定的。Hive 的设计特点如下：
  - 支持索引，加快数据查询。
  - 不同的文件格式，Hive 中默认有三个文件格式 TextFile，SequenceFile 以及 RCFile。
  - 将元数据保存在关系数据库中，减少了在查询中执行语义检查时间。
  - 可以直接使用存储在Hadoop 文件系统中的数据。
  - 内置大量用户函数UDF，也支持用户扩展UDF 函数来完成内置函数无法实现的操作。
  - 类SQL的查询方式，将SQL 查询转换为MapReduce 的job 在Hadoop集群上执行。

集群的规划设计

- 完成JDK安装配置
- 完成hadoop安装配置
- 下载文件
  - apache-hive-1.2.1-bin.tar.gz
  - mysql-connector-java-5.1.39.zip
- 集群中部署3个节点：

IP	hostname	充当的角色
192.168.149.128	Master	nn/RM/Hive/metastore/mysql
192.168.149.129	Slave1	dn/NM/Hive
192.168.149.131	Slave2	dn/NM/hive

安装模式

安装模式介绍：

Hive官网上介绍了Hive的3种安装方式，分别对应不同的应用场景。

1、内嵌模式（元数据保村在内嵌的derby，允许一个会话链接，尝试多个会话链接时会报错）

2、本地模式（本地安装mysql 替代derby存储元数据）

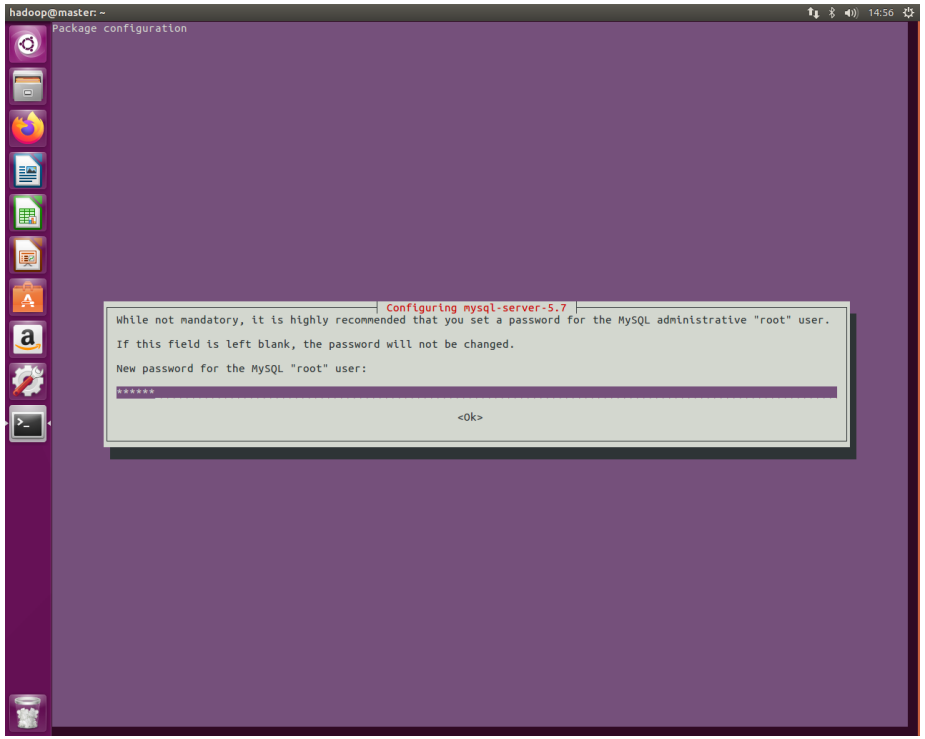
3、远程模式（远程安装mysql 替代derby存储元数据）

Hive的安装步骤--本地模式

1. 安装Mysql，并进行配置
2. 创建hive用户，并分配权限
3. 下载hive包，并解压到工作目录
4. 配置hive文件，设置环境变量、指出使用本地Mysql数据库，以及连接协议、账号、口令等
5. 把mysql-connector-java-x.x.x.jar复制到hive的lib目录下
6. 初始化mysql数据库
7. 启动hive
8. 验证hive是否安装成功

Hive的安装步骤--安装mysql

- 安装服务器：sudo apt-get install mysql-server
- 指定密码：123456



Hive的安装步骤--安装mysql

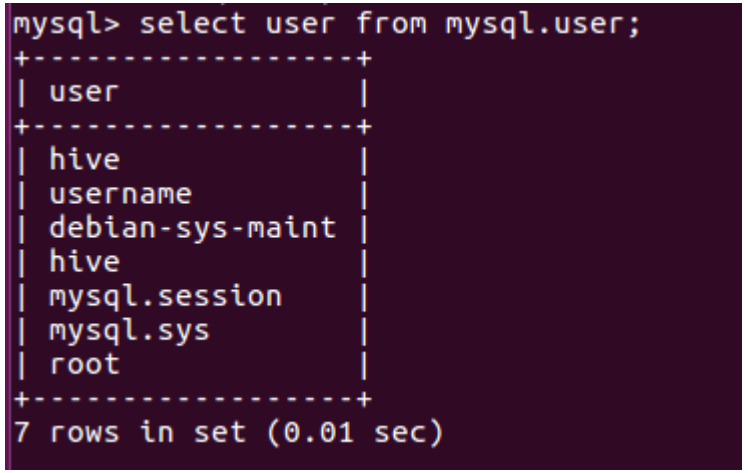
- 安装客户端
  - sudo apt-get install mysql-client
  - sudo apt-get install libmysqlclient-dev
- 修改配置文件/etc/mysql/mysql.conf.d/mysqld.cnf
  - # bind-address = 127.0.0.1

```
adoop@master:/etc/mysql/mysql.conf.d$ sudo vi mysqld.cnf
```

```
# Instead of skip-networking the default is now to listen only on
# localhost which is more compatible and is not less secure.
#bind-address           = 127.0.0.1
```

Hive的安装步骤--安装mysql

- 创建用户并设置远程登录权限
  - mysql -hlocalhost -uroot -p123456
  - GRANT ALL PRIVILEGES ON *.* TO 'hive'@'%' IDENTIFIED BY '123456'
  - GRANT ALL PRIVILEGES ON *.* TO 'hive'@'localhost' IDENTIFIED BY '123456'
  - FLUSH PRIVILEGES



Hive的安装步骤--Hive配置

- 上传apache-hive-1.2.1-bin.tar.gz，解压后重命名为hive-1.2.1
- MySQL的驱动包mysql-connector-java-5.1.39-bin.jar，将解压出来的jar放入hive 的lib目录下
- 配置hive环境变量

```
sudo vi /etc/profile
export HIVE_HOME=/home/hadoop/hive-1.2.1
export PATH=$PATH:$HIVE_HOME/bin:$PATH
export CLASSPATH=$CLASSPATH:$HIVE_HOME/lib
source /etc/profile
```

- 修改/conf/hive-env.sh文件

```
HADOOP_HOME=/home/hadoop/hadoop-2.7.2
export HIVE_CONF_DIR=/home/hadoop/hive-1.2.1/conf
```

Hive的安装步骤--HDFS存储配置

- Hive配置文件里要用到HDFS的一些路径，需要手动创建如下路径：
  - hdfs dfs -mkdir -p /hive/warehouse
  - hdfs dfs -mkdir -p /hive/logs
  - hdfs dfs -mkdir -p /hive/tmp
  - hdfs dfs -chmod 733 /hive/warehouse
  - hdfs dfs -chmod 733 /hive/logs
  - hdfs dfs -chmod 733 /hive/tmp

```
hadoop@master:~$ hdfs dfs -ls /hive
Found 3 items
drwxr-xr-x   - hadoop supergroup          0 2019-12-28 16:56 /hive/logs
drwxr-xr-x   - hadoop supergroup          0 2019-12-28 16:56 /hive/tmp
drwxr-xr-x   - hadoop supergroup          0 2019-12-28 16:56 /hive/warehouse
```

```
hadoop@master:~$ hdfs dfs -ls /hive
Found 3 items
drwx-wx-wx   - hadoop supergroup          0 2019-12-28 16:56 /hive/logs
drwx-wx-wx   - hadoop supergroup          0 2019-12-28 16:56 /hive/tmp
drwx-wx-wx   - hadoop supergroup          0 2019-12-28 16:56 /hive/warehouse
```

- 创建本地的目录
  - mkdir -p /home/hadoop/hive-1.2.1/hivedata/logs

Hive的安装步骤--配置hive-site.xml

- cp hive-default.xml.template hive-site.xml

<property><name>javax.jdo.option.ConnectionURL</name>
<value>jdbc:mysql://master:3306/metastore?createDatabaseIfNotExist=true</value></property>
<property><name>javax.jdo.option.ConnectionDriverName</name>
<value>com.mysql.jdbc.Driver</value></property>
<property><name>javax.jdo.option.ConnectionUserName</name>
<value>hive</value></property>
<property><name>javax.jdo.option.ConnectionPassword</name>
<value>123456</value></property>
<property><name>hive.metastore.warehouse.dir</name>
<value>/hive/warehouse</value></property>
<property><name>hive.exec.scratchdir</name><value>/hive/tmp</value>
</property>

Hive的安装步骤--配置log4j

- 创建配置文件：
- cp hive-exec-log4j.properties.template hive-exec-log4j.properties
- cp hive-log4j.properties.template hive-log4j.properties
- 修改两个文件中的配置:
  - hive.log.dir=/home/hadoop/hive-1.2.1/logs
  - log4j.appender.EventCounter=org.apache.hadoop.log.metrics.EventCounter

启动hive

- 确保hdfs启动

- 初始化mysql数据库：schematool --dbType mysql -initSchema

问题.ls: cannot access '/home/hadoop/spark-2.0.2/lib/spark-assembly-*.jar': No such file or directory

原因.spark升级到spark2以后，原有lib目录下的大JAR包被分散成多个小JAR包，原来的spark-assembly-*.jar已经不存在

解决.打开hive的安装目录下的bin目录，找到hive文件

修改成：sparkAssemblyPath=`ls ${SPARK_HOME}/jars/*.jar`

启动hive

问题. Sat Dec 28 18:24:00 CST 2019 WARN: Establishing SSL connection without server's identity verification is not recommended. According to MySQL 5.5.45+, 5.6.26+ and 5.7.6+ requirements SSL connection must be established by default if explicit option isn't set. For compliance with existing applications not using SSL the verifyServerCertificate property is set to 'false'. You need either to explicitly disable SSL by setting useSSL=false, or set useSSL=true and provide truststore for server certificate verification

解决.修改hive-site.xml配置文件

- <name>javax.jdo.option.ConnectionURL</name>
- <value>jdbc:mysql://localhost:3306/hive?createDatabaseIfNotExist=true&characterEncoding=UTF-8&useSSL=false</value>

启动hive

- 启动hive:hive

问题. Exception in thread "main" java.lang.RuntimeException: java.lang.IllegalArgumentException: java.net.URISyntaxException: Relative path in absolute URI: ${system:java.io.tmpdir%7D/$%7Bsystem:user.name%7D

解决.新建/hive-1.2.1/hivedata/tmp,修改hive-site.xml

<property><name>system:java.io.tmpdir</name>
<value>/home/hadoop/hive-1.2.1/hivedata</value></property>
<property><name>system:user.name</name>
<value>tmp</value></property>

- 测试hive

  hive> create TABLE test(id INT, name string); (观察hdfs下路径)

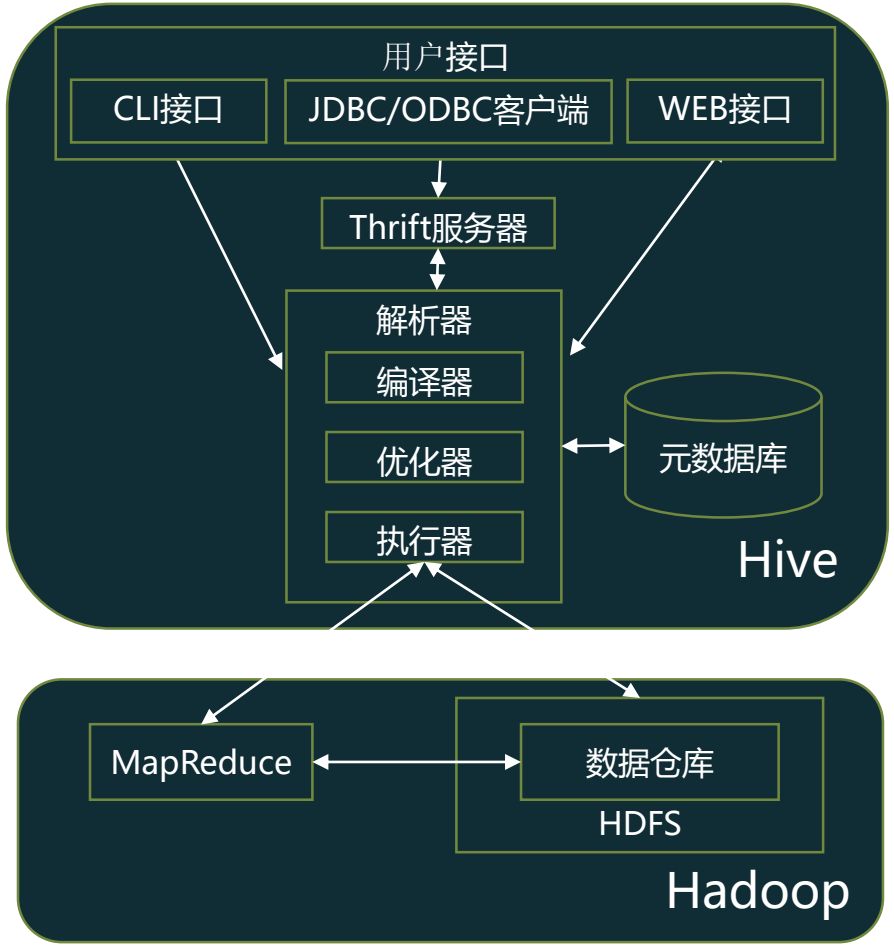
  hive> show tables;

hive远程模式安装部署

- 将master上的hive-1.2.1目录复制到其他节点上
- 按照master上的配置修改其他节点上的.profile文件，添加hive的配置
- 修改hive-size.xml文件，删除如下的配置：
  - javax.jdo.option.ConnectionURL
  - javax.jdo.option.ConnectionDriverName
  - javax.jdo.option.ConnectionUserName
  - javax.jdo.option.ConnectionPassword
- 修改hive-size.xml文件，添加如下的配置：
  - <property>
    <name>**hive.metastore.uris**</name>
    <value>thrift://192.168.149.128:9083</value>
    </property>
- master启动hdfs,hive服务端程序：master**:~$ hive --service metastore**
- 客户端直接使用hive命令即可
- 客户端/hive1.2.1/bin/hive文件，properties->permissions->execute

Hive架构

- 用户接口
  - ✓ 包括 CLI，JDBC/ODBC，WebUI
- 元数据库
  - ✓ 元数据用于存放Hive库的基础信息，它存储在关系数据库中，如 mysql、derby。元数据包括：数据库信息、表的名字，表的列和分区及其属性，表的属性，表的数据所在目录等。
- 解析器
  - ✓ 编译器、优化器、执行器
- Hadoop
  - ✓ 用 MapReduce 进行计算，用 HDFS 进行存储



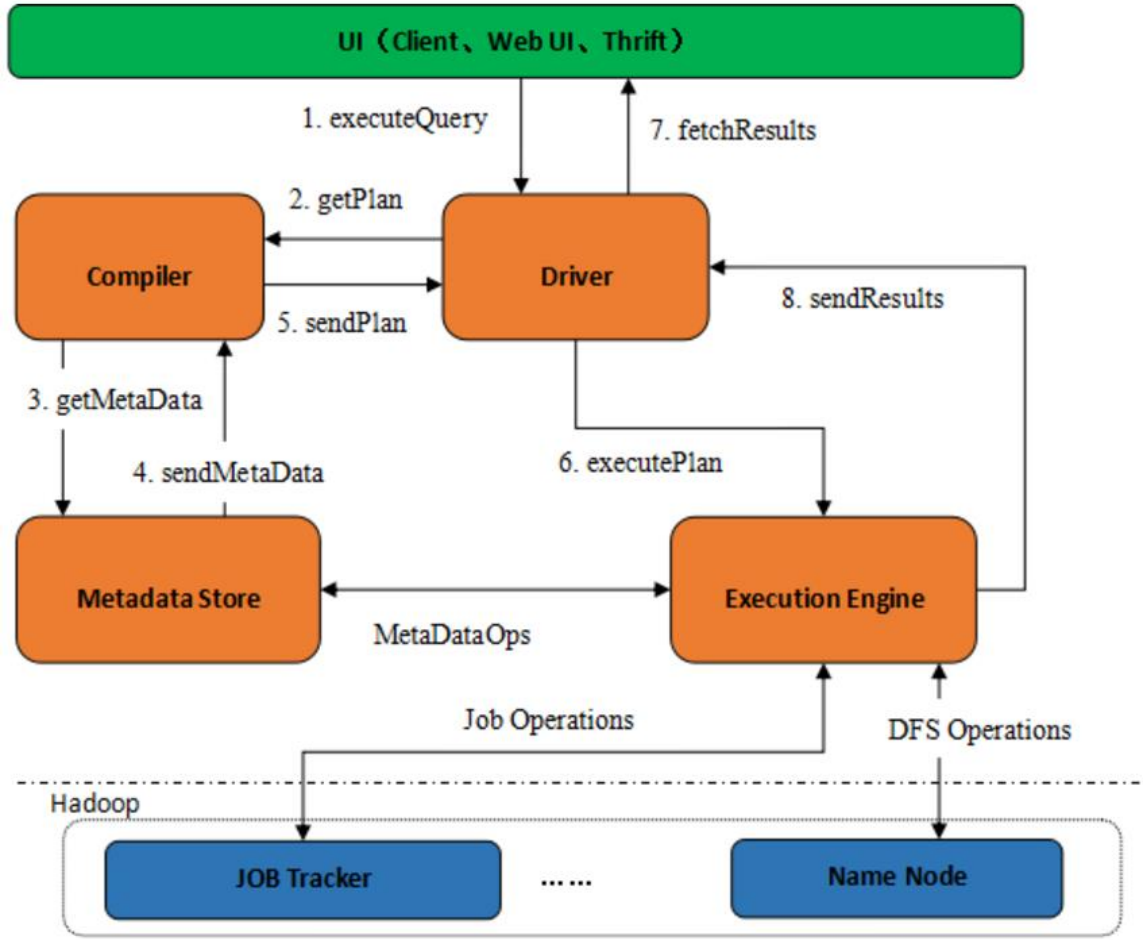
Hive的运行机制

① 用户通过用户接口连接Hive,发布Hive SQL

② Hive解析查询并制定查询计划

③ Hive将查询转换成MapReduce作业

④ Hive在Hadoop上执行MapReduce作业



metastore是hive元数据的集中存放地。
metastore默认使用内嵌的derby数据库作为存储引擎
Derby引擎的缺点：一次只能打开一个会话
使用Mysql作为外置存储引擎，多用户同时访问

Hive数据存储

Hive QL是类SQL，和SQL高80%以上的相似度，高扩展，不支持DELETE, UPDATE，不支持TRANSACTION

- Hive的数据存储基于Hadoop HDFS
- Hive没有专门的数据存储格式
- 存储结构主要包括：数据库、文件、表、视图、索引
- Hive默认可以直接加载文本文件（TextFile），还支持SequenceFile、RCFile
- 创建表时，指定Hive数据的列分隔符与行分隔符，Hive即可解析数据

用户接口主要有三个：CLI，JDBC/ODBC和 WebUI

1. CLI，即Shell命令行
2. JDBC/ODBC 是 Hive 的Java，与使用传统数据库JDBC的方式类似
3. WebGUI是通过浏览器访问 Hive

- Hive将元数据存储在数据库中(metastore)，目前只支持 mysql、derby。Hive 中的元数据包括表的名字，表的列和分区及其属性，表的属性（是否为外部表等），表的数据所在目录等
- 解释器、编译器、优化器完成 HQL 查询语句从词法分析、语法分析、编译、优化以及查询计划（plan）的生成。生成的查询计划存储在 HDFS 中，并在随后由 MapReduce 调用执行
- Hive 的数据存储在 HDFS 中，大部分的查询由 MapReduce 完成（包含 * 的查询，比如 select * from table 不会生成 MapRedcue任务）

Hive的数据库和表

- hive中数据库的概念本质上是表的一个目录或者命名空间。
  查看包含的数据库：show databases;
  创建一个数据库：create database wdc_db [cascade];
  切换数据库：use wdc_db;
  查看当前数据库：select current_database();
  删除数据库：drop database wdc_db;
  退出：quit;

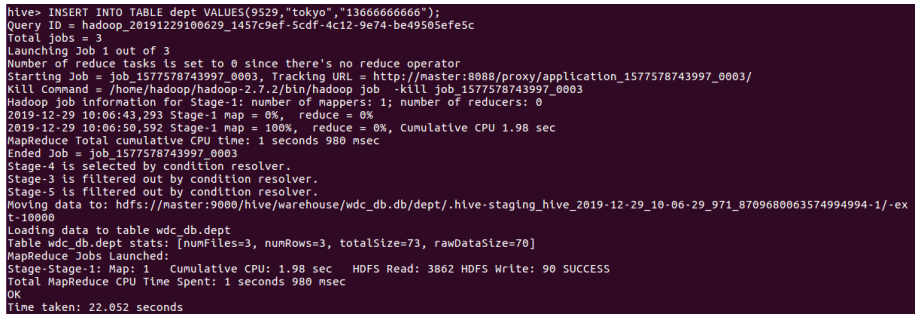
- Hive的表分为外部表和内部表
- 内部表： Hive既管理元数据也管理实际数据，不可以与其他工具共享数据，删除表时，数据也被删除
  表目录会创建在hdfs下/hive/warehouse/下的库名（xxx.db）目录下
- 外部表： Hive只管理元数据，可以与其他工具共享数据，删除表时，数据不被删除
  外部表会根据建表时LOCATION关键字指定的路径创建表目录。如果指定的目录存在，什么都不做。
  如果没有指定LOCATION，则位置与内部表相同。

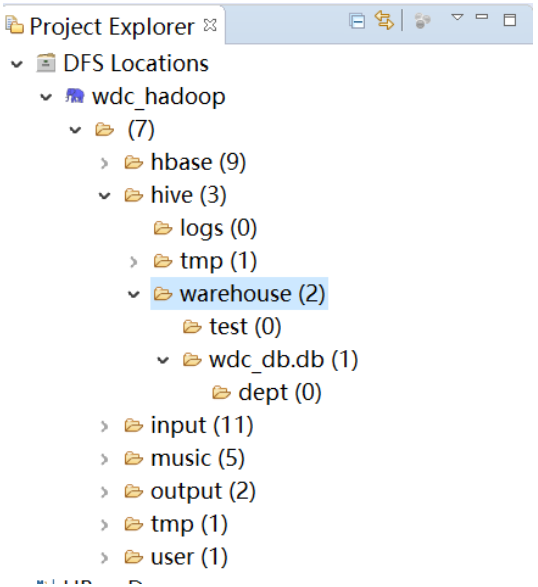
内部表和外部表的创建，差别就在两个关键字：EXTERNAL LOCATION

Hive的DDL及DML

- 创建表
  - CREATE TABLE if not exists dept(dept_no int,addr string,tel string)

  ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' ;
  - create table dept_tmp like dept;



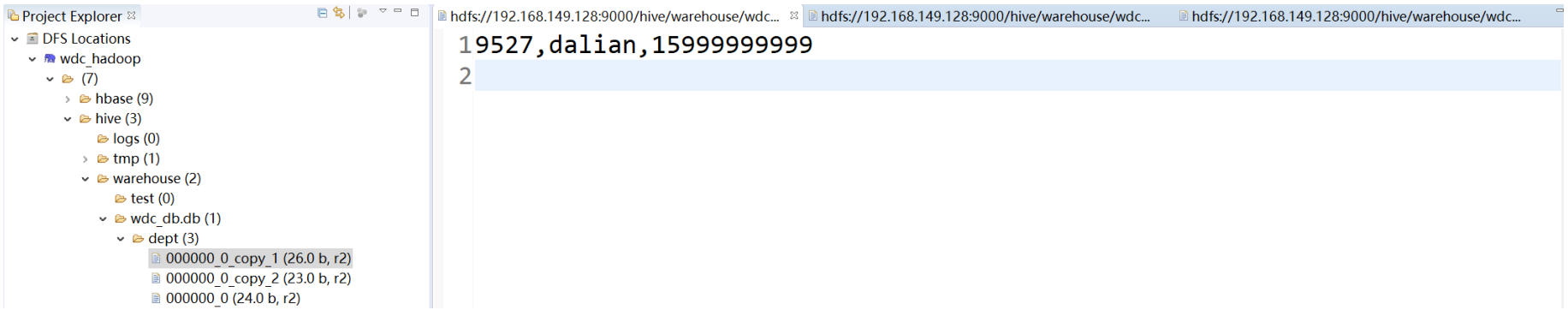


- 插入数据

  INSERT INTO TABLE dept VALUES(9527,"dalian","15999999999");

  INSERT INTO TABLE dept VALUES(9528,"shanghai","13888888888");

  INSERT INTO TABLE dept VALUES(9529,"tokyo","13666666666");



Hive的DDL及DML

- 显示所有的数据表：SHOW TABLES;

```
hive> show tables;
OK
dept
Time taken: 0.044 seconds, Fetched: 1 row(s)
```

- 查看表结构：DESC tablename;

```
hive> desc dept;
OK
dept_no                 int
addr                    string
tel                     string
Time taken: 0.201 seconds, Fetched: 3 row(s)
```

desc extended wdc_db.dept;

```
hive> desc extended wdc_db.dept;
OK
dept_no                 int
addr                    string
tel                     string

Detailed Table Information      Table(tableName:dept, dbName:wdc_db, owner:hadoop, createTime:1577584700, lastAccessTime:0, retention
:0, sd:StorageDescriptor(cols:[FieldSchema(name:dept_no, type:int, comment:null), FieldSchema(name:addr, type:string, comment:null),
FieldSchema(name:tel, type:string, comment:null)], location:hdfs://master:9000/hive/warehouse/wdc_db.db/dept, inputFormat:org.apache.
hadoop.mapred.TextInputFormat, outputFormat:org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat, compressed:false, numBuckets:
-1, serdeInfo:SerDeInfo(name:null, serializationLib:org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe, parameters:{field.delim=,, se
rialization.format=,}), bucketCols:[], sortCols:[], parameters:{}, skewedInfo:SkewedInfo(skewedColNames:[], skewedColValues:[], skewe
dColValueLocationMaps:{}), storedAsSubDirectories:false), partitionKeys:[], parameters:{totalSize=73, numRows=3, rawDataSize=70, COLU
MN_STATS_ACCURATE=true, numFiles=3, transient_lastDdlTime=1577585211}, viewOriginalText:null, viewExpandedText:null, tableType:MANAGE
D_TABLE)
Time taken: 0.14 seconds, Fetched: 5 row(s)
```

Hive的DDL及DML

- 增加列：alter table dept add columns(name string);

```
hive> alter table dept add columns(name string);
OK
Time taken: 1.523 seconds
```

- 删除列：alter table test REPLACE columns(name STRING,num INT,address STRING);

```
hive> desc test;
OK
id                      int
name                    string
Time taken: 0.136 seconds, Fetched: 2 row(s)
hive> alter table test REPLACE columns(name STRING,num INT,address STRING);
OK
Time taken: 0.159 seconds
hive> desc test;
OK
name                    string
num                     int
address                 string
Time taken: 0.111 seconds, Fetched: 3 row(s)
```

- 删除表：drop table test;

```
hive> drop table test;
OK
Time taken: 0.849 seconds
hive> show tables;
OK
Time taken: 0.047 seconds
```

基本数据类型

数据类型	所占字节	开始支持版本
TINYINT	1byte，-128 ~ 127	
SMALLINT	2byte，-32,768 ~ 32,767	
INT	4byte,-2,147,483,648 ~ 2,147,483,647	
BIGINT	8byte,-9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807	
BOOLEAN		
FLOAT	4byte单精度	
DOUBLE	8byte双精度	
STRING		
BINARY		从Hive0.8.0开始支持
TIMESTAMP		从Hive0.8.0开始支持
DECIMAL		从Hive0.11.0开始支持
CHAR		从Hive0.13.0开始支持
VARCHAR		从Hive0.12.0开始支持
DATE		从Hive0.12.0开始支持

复杂类型包括ARRAY,MAP,STRUCT,UNION，这些复杂类型是由基础类型组成的。

复合数据类型

```
CREATE TABLE student
(id INT,name STRING,favors ARRAY<STRING>,scores MAP<STRING, FLOAT>);
```

- hive记录中默认分隔符

分隔符	描述	语句
\n	分隔行	LINES TERMINATED BY '\t'
^A	分隔字段(列)，显示编码使用\001	FIELDS TERMINATED BY '\001'
^B	分隔复合类型中的元素，显示编码使用\002	COLLECTION ITEMS TERMINATED BY '\002'
^C	分隔map元素的key和value，显示编码使用\003	MAP KEYS TERMINATED BY '\003'

复合类型

- Struct内部的数据可以通过DOT（.）来存取，例如，表中一列c的类型为STRUCT{a INT; b INT}，我们可以通过c.a来访问域a
- create table student_test(id int, info struct<name:string, age:int>) row format delimited fields terminated by ',' collection items terminated by ':';

- Array中的数据为相同类型，例如，假如array A中元素['a','b','c']，则A[1]的值为'b'
- create table class_test(name string, student_id_list array<int>) row format delimited fields terminated by ',' collection items terminated by ':';

- Map访问指定域可以通过["指定域名称"]进行，例如，一个Map M包含了一个group-》gid的kv对，gid的值可以通过M['group']来获取
- create table employee_test(id string, perf map<string, int>) row format delimited fields terminated by '\001' collection items terminated by '\002' map keys terminated by '\003';

数据加载--本地

- Hive不支持一条一条的用insert语句进行插入操作，也不支持update的操作。数据是以从其他表查询或load的方式，加载到建立好的表中。数据一旦导入，则不可修改。
- LOAD DATA [LOCAL] INPATH 'filepath' [OVERWRITE] INTO TABLE tablename [PARTITION (partcol1=val1,partcol2=val2 ...)]
- LOCAL
  - 指定local,表示从本地文件系统加载（文件会被拷贝到HDFS中）
  - 不指定local,表示从HDFS中加载数据
- OVERWRITE 表示是否覆盖表中数据
- LOAD DATA LOCAL INPATH "/home/hadoop/hive-testdata/stu_datas.txt" INTO TABLE student_test;
- select info.name from student_test;

```
hive> LOAD DATA LOCAL INPATH "/home/hadoop/hive-testdata/stu_datas.txt" INTO TABLE student_test;
Loading data to table wdc_db.student_test
Table wdc_db.student_test stats: [numFiles=1, totalSize=50]
OK
Time taken: 0.66 seconds
hive> select * from student_test;
OK
9527    {"name":"wdc","age":29}
9528    {"name":"wec","age":28}
9529    {"name":"wsc","age":27}
9530    {"name":"wxc","age":26}
Time taken: 0.121 seconds, Fetched: 4 row(s)
```

数据加载--hdfs

- hdfs dfs -mkdir /hive_testdata
- hdfs dfs -put /home/hadoop/class_datas.txt /hive_testdata

```
hadoop@slave2:~$ hdfs dfs -put /home/hadoop/class_datas.txt /hive_testdata
hadoop@slave2:~$ hdfs dfs -ls /hive_testdata
Found 1 items
-rw-r--r--   2 hadoop supergroup         33 2019-12-29 17:52 /hive_testdata/class_datas.txt
```

- hdfs dfs -cat /hive_testdata/class_datas.txt

```
hadoop@slave2:~$ hdfs dfs -put /home/hadoop/class_datas.txt /hive_testdata
hadoop@slave2:~$ hdfs dfs -ls /hive_testdata
Found 1 items
-rw-r--r--   2 hadoop supergroup         33 2019-12-29 17:52 /hive_testdata/class_datas.txt
hadoop@slave2:~$ hdfs dfs -cat /hive_testdata/class_datas.txt
001,9527:9528:9529
002
003,9530hadoop@slave2:~$
```

- LOAD DATA INPATH "/hive_testdata/class_datas.txt" INTO TABLE class_test;

```
hive> LOAD DATA INPATH "/hive_testdata/class_datas.txt" INTO TABLE class_test;
Loading data to table wdc_db.class_test
Table wdc_db.class_test stats: [numFiles=1, totalSize=33]
OK
Time taken: 0.829 seconds
hive> select * from class_test;
OK
001     [9527,9528,9529]
002     NULL
003     [9530]
Time taken: 0.163 seconds, Fetched: 3 row(s)
```

数据加载--本地默认分隔符

- LOAD DATA LOCAL INPATH "/home/hadoop/hive-testdata/employee_datas.txt" INTO TABLE employee_test;

```
@slave1: ~/hive-testdata
9527^Asalary^C5000^Bage^C25^Bdept^C10
```

```
hive> LOAD DATA LOCAL INPATH "/home/hadoop/hive-testdata/employee_datas.txt" INTO TABLE employee_test;
Loading data to table wdc_db.employee_test
Table wdc_db.employee_test stats: [numFiles=1, totalSize=32]
OK
Time taken: 0.555 seconds
hive> select * from employee_test;
OK
9527    {"salary":5000,"age":25,"dept":10}
Time taken: 0.109 seconds, Fetched: 1 row(s)
hive> select perf['salary'] from employee_test;
OK
5000
Time taken: 0.167 seconds, Fetched: 1 row(s)
```

内部表&外部表

未被external修饰的是内部表（managed table），被external修饰的为外部表（external table）；

区别：

- 内部表数据由Hive自身管理，从本地文件系统或HDFS中copy文件到Hive数据仓库，外部表数据由HDFS管理，直接管理HDFS中的文件，而不将文件拷入Hive数据仓库
- 内部表数据存储的位置是hive.metastore.warehouse.dir（默认：/user/hive/warehouse），外部表数据的存储位置由自己制定（如果没有LOCATION，Hive将在HDFS上的/user/hive/warehouse文件夹下以外部表的表名创建一个文件夹，并将属于这个表的数据存放在这里）；
- 删除内部表会直接删除元数据（metadata）及存储数据；删除外部表仅仅会删除元数据，HDFS上的文件并不会被删除；
- 对内部表的修改会将修改直接同步给元数据，而对外部表的表结构和分区进行修改，则需要修复（MSCK REPAIR TABLE table_name;）

创建外部表

- 上传HDFS数据文件external_table.dat：hdfs dfs -put /home/hadoop/hive-testdata/external_table.dat /hive_testdata
- 创建表：create external table external_table (key string) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' location '/hive_testdata';

```
hive> create external table external_table (key string) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' location '/hive_testdata';
OK
Time taken: 1.229 seconds
hive> show tables;
OK
external_table
Time taken: 0.166 seconds, Fetched: 1 row(s)
```

- 查看数据：select * from external_table;select count(*) from external_table;
- 删除表：drop table external_table;

```
hive> select * from external_table;
OK
wdc
wec
wsc
wxc
wlc
Time taken: 0.519 seconds, Fetched: 5 row(s)
hive> drop table external_table;
OK
Time taken: 0.182 seconds
```

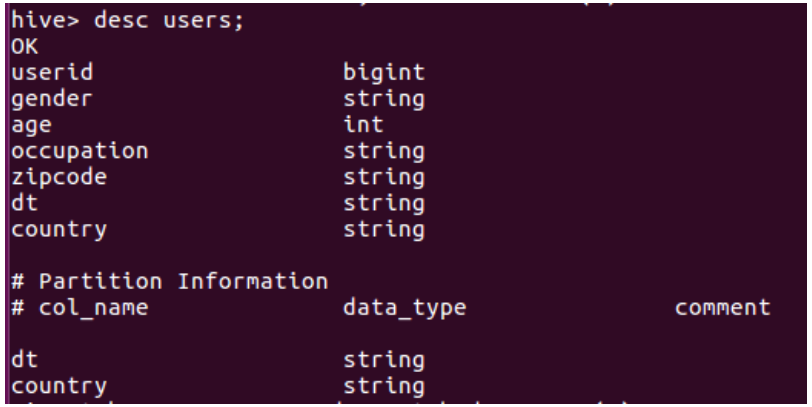
```
hadoop@slave2:~$ hdfs dfs -cat /hive_testdata/external_table.dat
wdc
wec
wsc
wxc
wlc
```

分区表

- 为了对表进行合理的管理以及提高查询效率，Hive可以将表组织成“分区”。
- 分区可以理解为分类，通过分类把不同类型的数据放到不同的目录下。
- 分类的标准就是分区字段，可以一个，也可以多个。
- 分区表的意义在于优化查询。查询时尽量利用分区字段。如果不使用分区字段，就会全部扫描。

```
CREATE TABLE users(
UserID BigInt,Gender String,Age Int,Occupation String,Zipcode String)
PARTITIONED BY (dt String,country STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
```

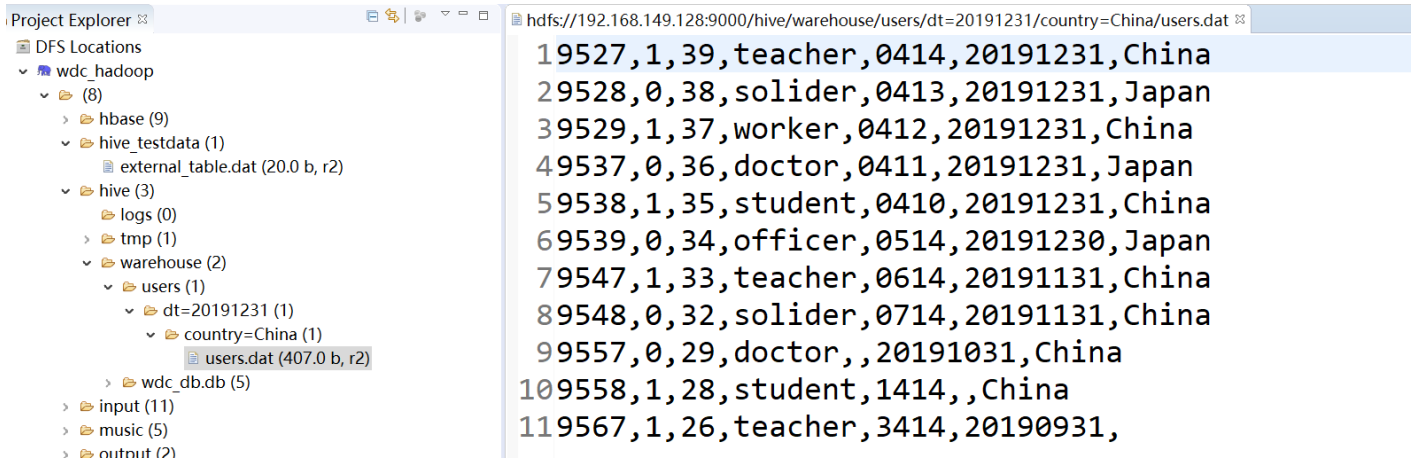
（注：在创建分区表时，partitioned字段可以不再字段列表中。生成的表中自动就会有该字段。）



分区表

- 数据装入的时候需要显示指定分区值：load data local inpath '/home/hadoop/hive-testdata/users.dat' into table users PARTITION(dt=20191231,country='China');

```
hive> load data local inpath '/home/hadoop/hive-testdata/users.dat' into table users
    > PARTITION(dt=20191231,country='China');
Loading data to table default.users partition (dt=20191231, country=China)
Partition default.users{dt=20191231, country=China} stats: [numFiles=1, numRows=0, totalSize=407, rawDataSize=0]
OK
Time taken: 4.717 seconds
hive> select * from users;
OK
9527    1       39      teacher 0414    20191231        China
9528    0       38      solider 0413    20191231        China
9529    1       37      worker  0412    20191231        China
9537    0       36      doctor  0411    20191231        China
9538    1       35      student 0410    20191231        China
9539    0       34      officer 0514    20191231        China
9547    1       33      teacher 0614    20191231        China
9548    0       32      solider 0714    20191231        China
9557    0       29      doctor          20191231        China
9558    1       28      student 1414    20191231        China
9567    1       26      teacher 3414    20191231        China
```



分区表

- 增加分区
  - 用 ALTER TABLE tbl_name ADD PARTITION …来向一个表中增加分区
  - alter table users add PARTITION(dt=20191131,country='China');
  - alter table users add PARTITION(dt=20191231,country='Japan');
  - alter table users add PARTITION(dt=20191231,country='US');
- 删除分区:
  - alter table users drop if exists partition (dt=20191231,country='US') ;

分区表

- 查询分区

只扫描country= ‘China’ 分区中的文件，而不用扫描其它的文件，从而能够提高查询效率

SELECT UserID, dt, Zipcode FROM users WHERE country='China’;

- 外部表也可以进行分区，这是管理大型生产数据集最常见的场景

create external table xxx{

… …

} partitioned by(year int,month int,day int… …)

桶表

- 对于每一个表或者分区， Hive可以进一步组织成桶。Hive采用对列值哈希，然后除以桶的个数求余的方式决定该条记录存放在哪个桶当中。在Hive中可以把数据按照桶的方式存放在不同文件中。
- 把表（或者分区）组织成桶（Bucket）优点：获得更高的查询处理效率，通过桶也可以提高取样的效率

CREATE TABLE bucketed_users(UserID Int,Gender string,Age Int,Occupation string,Zipcode string) CLUSTERED BY (UserID) INTO 4 BUCKETS ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';

- 要向分桶表中填充成员，需要set hive.enforce.bucketing 属性设置为 true，Hive 就知道用表定义中声明的数量来创建桶。

INSERT OVERWRITE TABLE bucketed_users SELECT UserID,Gender, Age,Occupation,Zipcode FROM users;

- 桶在物理上实际就是文件系统中的表目录中的一个文件。可以通过SORTED BY (UserID ASC)排序。

桶表

- 桶表的抽样查询

  select * from bucketed_users tablesample(bucket 2 out of 4 on userid);

- tablesample是抽样语句

  语法解析：TABLESAMPLE(BUCKET x OUT OF y)

  y必须是table总bucket数的倍数或者因子。hive根据y的大小，决定抽样的比例。

  例如，table总共分了64份，当y=32时，抽取(64/32=)2个bucket的数据，当y=128时，抽取(64/128=)1/2个bucket的数据。x表示从哪个bucket开始抽取。

  例如，table总bucket数为32，tablesample(bucket 3 out of 16)，表示总共抽取（32/16=）2个bucket的数据，分别为第3个bucket和第（3+16=）19个bucket的数据。

HiveQL查询

- select dept_no from dept where dept_no>9527 order by dept_no desc;
- select count(*) from dept;
- select sum(id) from student_test group by info.age limit 3;
- select id ,info.age+3 from student_test;
- 数组列：select name, student_id_list from class_test; select name,student_id_list[2] from class_test;
- map ：select id,perf['salary'],perf['age'],perf['dept']from employee_test;
- struct：select id ,info.name from student_test;
- 通过查询语句向表中插入数据：
- CREATE TABLE users_t(UserID BigInt,Gender String,Age Int,Occupation String,Zipcode String) PARTITIONED BY (dt String,country STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
- insert overwrite table users_t partition(dt='20191231',country) select u.userid,u.gender,u.age,u.occupation u.zipcode,u.country from users u where gender='1';

HiveQL查询

- 动态分区装载数据
- hive>set hive.exec.dynamic.partition=true;
- hive>set hive.exec.dynamic.partition.mode=nostrict;
- hive>set hive.exec.max.dynamic.partitions.pernode=1000;
- insert overwrite table users_t partition(dt,country) select u.userid,u.gender,u.age,u.occupation

  ,u.zipcode,u.dt,u.country from users u where gender='1’;
- 导出数据：insert overwrite local directory '/home/hadoop/data' ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' select * from users_t1;
- 空值判断：select * from dept where name is null;
- 获取时间： select current_date(); select current_timestamp(); select unix_timestamp();
- 对指定的值进行显示类型转换: select current_timestamp(),cast(unix_timestamp() as float);
- cast(0.2 as float),cast(date as date),cast(timestamp as date),cast(string as date),cast(date as string)
- 算术运算：取整round(3.14159),round(3.14159,2),floor(3.1415),ceil(3.1415),随机rand()
- 字符操作：长度length(name),连接concat('w','dc'),截位substr(name,2),substr(name,1,2),大写upper(),去空格trim(name)
- 聚合函数：count(*),sum(salary),avg(salary),min(salary),max(salary)

HiveQL查询

- 多表关联：select ct.*,st.* from class_test ct join student_test st on array_contains(ct.student_id_list,st.id);

  select ct.*,st.* from class_test ct,student_test st where array_contains(ct.student_id_list,st.id);
- Order by和Sort by
- order by会对查询结果集执行一个全局排序，所有的数据都通过一个reducer进行处理。
- sort by会在每个reducer中对数据进行排序，保证每个reducer的输出数据都是有序的，以提高后面进行的全局排序的效率。

  select userid,gender,age,occupation,zipcode from bucketed_users order by userid desc limit 5;

  select userid,gender,age,occupation,zipcode from bucketed_users sort by userid desc limit 5;
- hive中（distribute by + “表中字段”）关键字控制map输出结果的分发,相同值的map输出会发到一个reduce节点去处理。

  sort by为每一个reducer产生一个排序文件，他俩一般情况下会结合使用。

  select * from bucketed_users distribute by userid sort by userid desc;
- 多表关联

  select ct.*,st.* from class_test ct,student_test st where array_contains(ct.student_id_list,st.id);

  select ct.*,st.* from class_test ct join on student_test st on array_contains(ct.student_id_list,st.id);
- 抽样调查

  select * from bucketed_users tablesample(20 percent);

HiveQL—delete和update

- hive-site.xml

  <property><name>hive.support.concurrency</name><value>true</value></property>

  <property><name>hive.enforce.bucketing</name><value>true</value></property>

  <property><name>hive.exec.dynamic.partition.mode</name><value>nonstrict</value></property>

  <property><name>hive.txn.manager</name><value>org.apache.hadoop.hive.ql.lockmgr.DbTxnManager</value></property>

  <property><name>hive.compactor.initiator.on</name><value>true</value></property>

  <property><name>hive.compactor.worker.threads</name><value>1</value></property>

  <property> <name>hive.in.test</name><value>true</value></property>

- 表的存储格式必须是ORC（STORED AS ORC）；
- 表必须进行分桶（CLUSTERED BY (col_name, col_name, ...) INTO num_buckets BUCKETS）；
- Table property中参数transactional必须设定为True（tblproperties('transactional'='true')）；
- create table if not exists test_orc(name string,gender string,id BIGINT)clustered by (id) into 8 buckets STORED AS ORC TBLPROPERTIES ('transactional'='true');

HiveQL—delete和update

```
OK
test_orc.name   test_orc.gender test_orc.id
wlc     0       24
wwc     1       25
wssc    0       26
wsc     1       27
wec     0       28
wdc     1       29
wdc     1       23
Time taken: 0.101 seconds, Fetched: 7 row(s)
hive (wdc_db)> update test_orc set name ='wdcccc' where id =23;
Query ID = hadoop_20200112000412_b9ea4886-f724-428b-bc04-d048c0dc9a1d
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 8
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1578751990634_0008, Tracking URL = http://master:8088/proxy/application_1578751990634_0008/
Kill Command = /home/hadoop/hadoop-2.7.2/bin/hadoop job  -kill job_1578751990634_0008
Hadoop job information for Stage-1: number of mappers: 8; number of reducers: 8
2020-01-12 00:04:21,190 Stage-1 map = 0%,  reduce = 0%
2020-01-12 00:04:48,220 Stage-1 map = 13%,  reduce = 0%, Cumulative CPU 12.63 sec
2020-01-12 00:04:50,627 Stage-1 map = 63%,  reduce = 0%, Cumulative CPU 20.4 sec
2020-01-12 00:04:52,767 Stage-1 map = 75%,  reduce = 0%, Cumulative CPU 21.74 sec
2020-01-12 00:04:54,904 Stage-1 map = 88%,  reduce = 0%, Cumulative CPU 22.52 sec
2020-01-12 00:04:55,960 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 23.76 sec
2020-01-12 00:05:12,643 Stage-1 map = 100%,  reduce = 37%, Cumulative CPU 28.25 sec
2020-01-12 00:05:13,670 Stage-1 map = 100%,  reduce = 38%, Cumulative CPU 28.9 sec
2020-01-12 00:05:17,780 Stage-1 map = 100%,  reduce = 71%, Cumulative CPU 32.77 sec
2020-01-12 00:05:18,824 Stage-1 map = 100%,  reduce = 79%, Cumulative CPU 33.76 sec
2020-01-12 00:05:19,855 Stage-1 map = 100%,  reduce = 96%, Cumulative CPU 35.67 sec
2020-01-12 00:05:20,874 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 36.12 sec
MapReduce Total cumulative CPU time: 36 seconds 120 msec
Ended Job = job_1578751990634_0008
Loading data to table wdc_db.test_orc
Table wdc_db.test_orc stats: [numFiles=57, numRows=7, totalSize=16283, rawDataSize=0]
MapReduce Jobs Launched:
Stage-Stage-1: Map: 8  Reduce: 8   Cumulative CPU: 36.12 sec   HDFS Read: 91423 HDFS Write: 1112 SUCCESS
Total MapReduce CPU Time Spent: 36 seconds 120 msec
OK
row__id _c1     gender  id
Time taken: 69.787 seconds
hive (wdc_db)> select * from test_orc;
OK
test_orc.name   test_orc.gender test_orc.id
wlc     0       24
wwc     1       25
wssc    0       26
wsc     1       27
wec     0       28
wdc     1       29
wdcccc  1       23
Time taken: 0.298 seconds, Fetched: 7 row(s)
```

```
hive (wdc_db)> delete from test_orc where id between 26 and 30;
Query ID = hadoop_20200112001759_09126706-9657-4bb1-b577-8527a3e34acf
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 8
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1578751990634_0009, Tracking URL = http://master:8088/proxy/application_1578751990634_0009/
Kill Command = /home/hadoop/hadoop-2.7.2/bin/hadoop job  -kill job_1578751990634_0009
Hadoop job information for Stage-1: number of mappers: 8; number of reducers: 8
2020-01-12 00:18:07,503 Stage-1 map = 0%,  reduce = 0%
2020-01-12 00:18:27,079 Stage-1 map = 13%,  reduce = 0%, Cumulative CPU 4.12 sec
2020-01-12 00:18:28,156 Stage-1 map = 25%,  reduce = 0%, Cumulative CPU 4.53 sec
2020-01-12 00:18:29,236 Stage-1 map = 38%,  reduce = 0%, Cumulative CPU 6.73 sec
2020-01-12 00:18:30,274 Stage-1 map = 50%,  reduce = 0%, Cumulative CPU 10.85 sec
2020-01-12 00:18:32,363 Stage-1 map = 63%,  reduce = 0%, Cumulative CPU 15.12 sec
2020-01-12 00:18:34,451 Stage-1 map = 75%,  reduce = 0%, Cumulative CPU 15.91 sec
2020-01-12 00:18:35,517 Stage-1 map = 88%,  reduce = 0%, Cumulative CPU 18.36 sec
2020-01-12 00:18:38,628 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 19.01 sec
2020-01-12 00:18:49,012 Stage-1 map = 100%,  reduce = 13%, Cumulative CPU 20.5 sec
2020-01-12 00:18:50,064 Stage-1 map = 100%,  reduce = 38%, Cumulative CPU 23.82 sec
2020-01-12 00:18:52,134 Stage-1 map = 100%,  reduce = 58%, Cumulative CPU 26.07 sec
2020-01-12 00:18:53,165 Stage-1 map = 100%,  reduce = 79%, Cumulative CPU 28.4 sec
2020-01-12 00:18:54,216 Stage-1 map = 100%,  reduce = 91%, Cumulative CPU 29.99 sec
2020-01-12 00:18:56,264 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 32.09 sec
MapReduce Total cumulative CPU time: 32 seconds 90 msec
Ended Job = job_1578751990634_0009
Loading data to table wdc_db.test_orc
Table wdc_db.test_orc stats: [numFiles=61, numRows=3, totalSize=18426, rawDataSize=0]
MapReduce Jobs Launched:
Stage-Stage-1: Map: 8  Reduce: 8   Cumulative CPU: 32.09 sec   HDFS Read: 92497 HDFS Write: 2607 SUCCESS
Total MapReduce CPU Time Spent: 32 seconds 90 msec
OK
row__id
Time taken: 58.264 seconds
hive (wdc_db)> select * from test_orc;
OK
test_orc.name   test_orc.gender test_orc.id
wlc     0       24
wwc     1       25
wdcccc  1       23
Time taken: 0.112 seconds, Fetched: 3 row(s)
hive (wdc_db)>
```

Hive视图

使用视图可以降低查询的复杂度

- 视图的创建：create view v_users as select userid,gender,age from bucketed_users where age >35;
- 视图的删除：drop view if exists v_users;

```
hive> create view v_users as select userid,gender,age from bucketed_users where age >35;
OK
Time taken: 0.284 seconds
hive>
```

```
hive> select * from v_users;
OK
9528    0       38
9537    0       36
9529    1       37
9527    1       39
Time taken: 0.331 seconds, Fetched: 4 row(s)
```

```
hive> drop view v_users;
OK
Time taken: 0.3 seconds
```

Hive索引

使用Hive索引，其主要功能就是避免第一轮mr任务的全表扫描，而改为扫描索引表。如果索引表本身很大，其开销仍然很大，在集群资源充足的情况下，可以忽略使用hive下的索引。

- Hive1.2.1版本目前支持的索引类型有CompactIndexHandler和Bitmap。
- CompactIndexHandler压缩索引通过将列中相同的值的字段进行压缩从而减小存储和加快访问时间。
- Bitmap位图索引作为一种常见的索引，如果索引列只有固定的几个值，那么就可以采用位图索引来加速查询，利用位图索引可以方便的进行AND/OR/XOR等各类计算。
- 创建索引：create index dept_index on table dept(addr) as 'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler' with deferred rebuild in table dept_index_table;

  as 子句指定了索引处理器,就是一个实现了索引接口的java类。

  in table子句指定了在表中保存索引数据。

  with deferred rebuild：索引呈现空白状态，不包含任何数据
- 显示索引：show formatted index on dept;
- 删除索引：drop index if exists dept_index on dept;

Hive--文件格式

- Hive 中常用的有三种文件格式 TextFile、SequenceFile 以及 RCFile
- TextFile：默认的文件格式，文本文件格式便于和其他工具共享数据，便于查看和编辑，但文本文件存储占用空间较大。
- SequenceFile：是Hadoop提供的一种二进制文件格式，是Hadoop支持的标准文件格式，可以节约存储空间，也可以提高I/O性能。
- RCfile：hive支持的另一种高效的二进制文件格式， RCfile是一种行列存储相结合的存储方式，先将数据按行分块再按列式存储，保证同一条记录在一个块上，避免读取多个块，有利于数据压缩和快速进行列存储。

Hive--Sequence file

- HDFS和MR主要针对大数据文件来设计，在小文件处理上效率低。针对小文件可以选择Sequence file存储格式,将这些小文件包装起来,将整个文件作为一条记录,可以获取更高效率的储存和处理，避免多次打开关闭流耗费计算资源。
- Sequence file存储格式也可以将一个文件划分成多个块，采用一种可分割的方式对块进行压缩。
- 在hive中使用Sequence file存储格式
  - create table sequence_file_table(id int,name string) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' stored as sequencefile;
- SequenceFile采用了3种压缩方式：
  - None
  - Record（默认）
  - Block（压缩性能最好）
- 可以在hadoop的mapred-site.xml文件中，或者在hive的hive-site.xml中定义压缩方式。
  - mapred.output.compression.type

Hive--RCfile

- RCFile（Record Columnar File）存储结构遵循的是"先水平划分，再垂直划分"的设计理念，它结合了行存储和列存储的

优点。首先，RCFile保证同一行的数据位于同一节点，因此元组重构的开销很低。其次，像列存储一样，RCFile能够利用列维度的数据压缩，并且能跳过不必要的列读取。

- 创建RCFile存储格式的表
    - CREATE TABLE rc_file_table(key string,value string)ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.columnar.ColumnarSerDe' STORED AS RCFile;
- 其它
- ORCFile存储方式：数据按行分块 每块按照列存储 ，压缩快 快速列存取，效率比rcfile高,是rcfile的改良版本，相比RC能够更好的压缩，能够更快的查询，但还是不支持模式演进。
- Parquet存储方式能够很好的压缩，有很好的查询性能，支持有限的模式演进。但是写速度通常比较慢。这中文件格式主要是用在Cloudera Impala上面的。

Hive--SerDe

- SerDe 是 “Serializer and Deserializer.”的缩写，Hive 使用 SerDe和FileFormat进行行内容的读写
- HDFS文件 --> InputFileFormat --> <key, value> --> Deserializer --> 行对象
- 行对象 --> Serializer --> <key, value> --> OutputFileFormat --> HDFS文件
- 对象的序列化主要有两种用途：对象的持久化和对象数据的网络传送

```
CREATE [EXTERNAL] TABLE [IF NOT EXISTS] table_name
[(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]
[ROW FORMAT SERDE row_format]
[STORED AS file_format]
[LOCATION hdfs_path]
```

Hive--UDF

- User defined Function的简写，意思是用户自定义方法。在使用Hive shell时，可以使用sum()、count()等函数，但是有的业务场合希望有其它更多的函数。这时，就需要自定义一些函数，也就是UDF。
  - UDF（用户自定义函数）：操作单个数据行，产生单个数据行；
  - UDAF（用户自定义聚合函数）：操作多个数据行，产生一个数据行。
  - UDTF（用户自定义表生成函数）：操作一个数据行，产生多个数据行。
- 查看当前Hive会话中所加载的所有函数名称：show functions;
- 查看函数说明
  - desc function concat;
  - desc function extended concat;
- UDF函数可以直接应用于select语句，对查询结构做格式化处理后，再输出内容。

Hive--UDF创建

- 新建Java Application
- 导入hive/lib下的所有包,导入hadoop-common.jar包
- 代码上传打包， hive> add jar /home/hadoop/NationUDF.jar;

```
hive> add jar /home/hadoop/NationUDF.jar;
Added [/home/hadoop/NationUDF.jar] to class path
Added resources: [/home/hadoop/NationUDF.jar]
```

- hive 下创建临时函数： hive> create temporary function getNation as 'NationUDF';【注意包名】
- 测试：hive> select dept_no,getNation(addr),tel from dept;

```
hive> select dept_no,getNation(addr),tel from dept;
OK
9527    大连    15999999999
9528    上海    13888888888
9529    东京    13666666666
8527    其它    13666666666
8528    本溪    13666666666
8529    其它    13666666666
7529    其它    13666666666
7528    其它    13666666666
7527    其它    13666666666
```

Hive--高级函数

- lateral view用于和split, explode等UDTF一起使用，它能够将一行数据拆成多行数据，在此基础上可以对拆分后的数据进行聚合。lateral view首先为原始表的每行调用UDTF，UTDF会把一行拆分成一或者多行，lateral view再把结果组合，产生一个支持别名表的虚拟表。
- select name,student_id from class_test lateral view explode(student_id_list) student_id_123 as student_id ;
- collect相关的函数有collect_list和collect_set。
- 它们都是将分组中的某列转为一个数组返回，不同的是collect_list不去重而collect_set去重。
- select tel ,collect_list(dept_no) from dept group by tel;

Hive--性能优化

- 执行本地模式，避免执行MR
  - select * or select field1,field2
  - limite 10
  - where语句中只有分区字段
  - 使用本地set hive.exec.mode.local.auto=true;
- 本地模式设置方式：
  - set mapred.job.tracker=local;
  - set hive.exec.mode.local.auto=true;
- 下面两个参数是local mr中常用的控制参数:

  hive.exec.mode.local.auto.inputbytes.max默认134217728设置local mr的最大输入数据量,当输入数据量小于这个值的时候会采用local mr的方式

  hive.exec.mode.local.auto.input.files.max默认是4设置local mr的最大输入文件个数,当输入文件个数小于这个值的时候会采用local mr的方式

- hive是如何将查询转化为MapReduce的?
- EXPLAIN的使用
  - hive对sql的查询计划信息解析：EXPLAIN SELECT COUNT(1) FROM class_test;
- EXPLAIN EXTENDED
  - 显示详细扩展查询计划信息： EXPLAIN EXTENDED SELECT COUNT(1) FROM class_test;

Hive--性能优化

- 开启并行计算,增加集群的利用率
  - set hive.exec.parallel=true
- 设置严格模式后，以便某些查询在严格模式下无法执行
  - set hive.mapred.mode=strict | nostrict;
  - strict可以禁止三种类型的查询：
    - 强制分区表的where条件过滤
    - Order by语句必须使用limit
    - 限制笛卡尔积查询
- 调整mapper和reducer的数量
  - 太多map导致启动产生过多开销
  - 按照输入数据量大小确定reducer数目，
    - set mapred.reduce.tasks= 默认3
    - dfs -count /分区目录/*
    - hive.exec.reducers.max设置阻止资源过度消耗
- JVM重用
  - 小文件多或task多的业务场景
  - set mapred.job.reuse.jvm.num.task=10
  - 会一直占用task槽

Hive--性能优化

- order by 语句：　是全局排序
- sort by 语句：　是单reduce排序
- distribute by语句：是分区字段排序;
- cluster by语句：
- 可以确保类似的数据的分发到同一个reduce task中，并且保证数据有序防止所有的数据分发到同一个reduce上，导致整体的job时间延长
- cluster by语句的等价语句：
- distribute by col sort by col ASC

Hive--性能优化

- 数据倾斜（skew）使得某个reducer要处理的数据特别多，而其他reducer要处理的很少,任务进度长时间维持在99%（或100%）
- 情景：

关键词	情形	后果
Join	其中一个表较小， 但是key集中	分发到某一个或几个Reduce上的数据远高于平均值
	大表与大表，但是分桶的判断字段0值或空值过多	这些空值都由一个reduce处理，灰常慢
group by	group by 维度过小， 某值的数量过多	处理某值的reduce灰常耗时
Count Distinct	某特殊值过多	处理此特殊值的reduce耗时

- 原因： key分布不均匀，业务数据本身的特性，建表时考虑不周，某些SQL语句本身就有数据倾斜
- 任务进度长时间维持在99%（或100%），查看任务监控页面，发现只有少量（1个或几个）reduce子任务未完成。因为其处理的数据量和其他reduce差异过大。
- 单一reduce的记录数与平均记录数差异过大，通常可能达到3倍甚至更多。 最长时长远大于平均时长。

Hive--性能优化

- hive.map.aggr=true（开启map端combiner)
- hive.groupby.skewindata=true (有数据倾斜的时候进行负载均衡)
- 这个设置可以将顶层的聚合操作放在Map阶段执行，从而减轻清洗阶段数据传输和Reduce阶段的执行时间，提升总体性能。缺点：该设置会消耗更多的内存。
- 选用join key分布最均匀的表作为驱动表
  - 驱动表最右边：查询表表的大小从左边到右边依次增大
  - 标志机制：显示的告知查询优化器哪张表示大表/*+streamtable(table_name)*/
- 大小表Join：使用map join让小的维度表（1000条以下的记录条数） 先进内存。在map端完成reduce.
- 大表Join大表：把空值的key变成一个字符串加上随机数，把倾斜的数据分到不同的reduce上，由于null值关联不上，处理后并不影响最终结果。
- group by维度过小：采用sum() group by的方式来替换count(distinct)完成计算。
- 特殊情况特殊处理：在业务逻辑优化效果的不大情况下，有些时候是可以将倾斜的数据单独拿出来处理。最后union回去。

华信培训