Hive概述

1、Hive的概念及架构

Hive 是建立在 Hadoop 上的数据仓库基础构架。

它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，它允许熟悉 SQL 的用户查询数据。同时，这个语言也允许熟悉 MapReduce 的开发者开发自定义的 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是SQL解析引擎**，它将SQL语句转译成Map/Reduce Job然后在Hadoop执行。

Hive的表其实就是HDFS的目录，按表名把文件夹分开。如果是分区表，则分区值是子文件夹，可以直接在 Map/Reduce Job里使用这些数据。**Hive相当于hadoop的客户端工具**，部署时不一定放在集群管理节点中，也可以放在某个节点上。

数据仓库，英文名称为**Data Warehouse**，可简写为DW或DWH。数据仓库，是为企业所有级别的决策制定过程，提供所有类型数据支持的战略集合。它出于分析性报告和决策支持目的而创建。为需要业务智能的企业，提供指导业务流程改进、监视时间、成本、质量以及控制。

Hive的版本介绍：0.13和.14版本，稳定版本，但是不支持更新删除操作。1.2.1和1.2.2 版本，稳定版本，为Hive2版本（是主流版本） 1.2.1的程序只能连接hive1.2.1 的hiveserver2

2、Hive与传统数据库比较

1.查询语言。类 SQL 的查询语言 HQL。熟悉 SQL 开发的开发者可以很方便的使用 Hive 进行开发。

2.数据存储位置。所有 Hive 的数据都是存储在 HDFS 中的。而数据库则可以将数据保存在块设备或者本地文件系统中。

3.数据格式。Hive 中没有定义专门的数据格式。而在数据库中，所有数据都会按照一定的组织存储，因此，数据库加载数据的过程会比较耗时。

4.数据更新。Hive 对数据的改写和添加比较弱化，0.14版本之后支持，需要启动配置项。而数据库中的数据通常是需要经常进行修改的。

5.索引。Hive 在加载数据的过程中不会对数据进行任何处理。因此访问延迟较高。数据库可以有很高的效率，较低的延迟。由于数据的访问延迟较高，决定了 Hive 不适合在线数据查询。

6.执行计算。Hive 中执行是通过 MapReduce 来实现的而数据库通常有自己的执行引擎。数据规模。由于 Hive 建立在集群上并可以利用 MapReduce 进行并行计算，因此可以支持很大规模的数据；对应的，数据库可以支持的数据规模较小。

3、Hive的数据存储格式

Hive的数据存储基于Hadoop HDFS。

Hive没有专门的数据文件格式，常见的有以下几种：TEXTFILE、SEQUENCEFILE、AVRO、RCFILE、ORCFILE、PARQUET。

TextFile:

TEXTFILE 即正常的文本格式，是Hive默认文件存储格式，因为大多数情况下源数据文件都是以text文件格式保存（便于查看验数和防止乱码）。此种格式的表文件在HDFS上是明文，可用hadoop fs -cat命令查看，从HDFS上get下来后也可以直接读取。 **TEXTFILE 存储文件默认每一行就是一条记录，可以指定任意的分隔符进行字段间的分割。但这个格式无压缩，需要的存储空间很大。**虽然可以结合Gzip、Bzip2、Snappy等使用，使用这种方式，Hive不会对数据进行切分，从而无法对数据进行并行操作。**一般只有与其他系统由数据交互的接口表采用 TEXTFILE 格式，其他事实表和维度表都不建议使用。**

RCFile:

Record Columnar的缩写。是Hadoop中第一个列文件格式。能够很好的压缩和快速的查询性能。通常写操作比较慢，比非列形式的文件格式需要更多的内存空间和计算量。**RCFile是一种行列存储相结合的存储方式。**首先，其将数据按行分块，保证同一个record在一个块上，避免读一个记录需要读取多个block。其次，块数据列式存储，有利于数据压缩和快速的列存取。

ORCFile:

Hive从0.11版本开始提供了ORC的文件格式，**ORC文件**不仅仅是**一种列式文件存储格式**，最重要的是**「有着很高的压缩比」**，并且**对于MapReduce来说是可切分（Split）的。**因此，**在Hive中使用ORC作为表的文件存储格式，不仅可以很大程度的节省HDFS存储资源，而且对数据的查询和处理性能有着非常大的提升**，因为ORC较其他文件格式压缩比高，查询任务的输入数据量减少，使用的Task也就减少了。**ORC能很大程度的节省存储和计算资源，但它在读写时候需要消耗额外的CPU资源来压缩和解压缩**，当然这部分的CPU消耗是非常少的。

Parquet:

通常我们使用关系数据库存储结构化数据，而**关系数据库中使用数据模型都是扁平式的**，遇到诸如List、Map和自定义Struct的时候就需要用户在应用层解析。但是在大数据环境下，通常数据的来源是服务端的埋点数据，很可能需要把程序中的某些对象内容作为输出的一部分，而每一个对象都可能是嵌套的，所以如果能够原生的支持这种数据，这样在查询的时候就不需要额外的解析便能获得想要的结果。

Parquet``的灵感来自于2010年Google发表的Dremel论文，文中介绍了一种支持嵌套结构的存储格式，并且使用了列式存储的方式提升查询性能。**Parquet仅仅是一种存储格式，它是语言、平台无关的，并且不需要和任何一种数据处理框架绑定。这也是parquet相较于orc的仅有优势：支持嵌套结构」。**Parquet 没有太多其他可圈可点的地方,比如他**不支持update操作(数据写成后不可修改),不支持ACID等.**

SEQUENCEFILE:

SequenceFile是Hadoop API 提供的一种二进制文件，它将数据以<key,value>的形式序列化到文件中。这种二进制文件内部使用Hadoop 的标准的Writable 接口实现序列化和反序列化。它与Hadoop API中的MapFile 是互相兼容的。Hive 中的SequenceFile 继承自Hadoop API 的SequenceFile，不过它的key为空，使用value 存放实际的值， 这样是为了避免MR 在运行map 阶段的排序过程。**SequenceFile支持三种压缩选择： NONE, RECORD, BLOCK。Record压缩率低，一般建议使用BLOCK压缩。SequenceFile最重要的优点就是Hadoop原生支持较好，有API**，但除此之外平平无奇，实际生产中不会使用。

AVRO:

Avro是一种用于支持数据密集型的二进制文件格式。它的文件格式更为紧凑，若要读取大量数据时，Avro能够提供更好的序列化和反序列化性能。并且**Avro数据文件天生是带Schema定义的**，所以它不需要开发者在API 级别实现自己的Writable对象。**Avro提供的机制使动态语言可以方便地处理Avro数据。**最近多个Hadoop 子项目都支持Avro数据格式，如Pig 、Hive、Flume、Sqoop和Hcatalog。

其中的**TextFile**、**RCFile**、**ORC**、**Parquet**为Hive**最常用的四大存储格式**它们的 存储效率及执行速度比较如下：ORCFile存储文件读操作效率最高，耗时比较（ORC<Parquet<RCFile<TextFile） ORCFile存储文件占用空间少，压缩效率高(ORC<Parquet<RCFile<TextFile)

4、Hive操作客户端

常用的客户端有两个：CLI，JDBC/ODBC

CLI，即Shell命令行

JDBC/ODBC 是 Hive 的Java，与使用传统数据库JDBC的方式类似。

1.**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；由解释器、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。**

2.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metastore

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，可以多用户同时访问`

二、Hive的基本语法

1、Hive建表语法

```
CREATE [EXTERNAL] TABLE [IF NOT EXISTS] table_name
  -- 定义字段名，字段类型
  [(col_name data_type [COMMENT col_comment], ...)]
  -- 给表加上注解
  [COMMENT table_comment]
  -- 分区
  [PARTITIONED BY (col_name data_type [COMMENT col_comment], ...)]
  -- 分桶
  [CLUSTERED BY (col_name, col_name, ...)
  -- 设置排序字段 升序、降序
  [SORTED BY (col_name [ASC|DESC], ...)] INTO num_buckets BUCKETS]
  [
   -- 指定设置行、列分隔符
   [ROW FORMAT row_format]
   -- 指定Hive储存格式：textFile、rcFile、SequenceFile 默认为：textFile
   [STORED AS file_format]

   | STORED BY 'storage.handler.class.name' [ WITH SERDEPROPERTIES (...) ]  (Note:  only
available starting with 0.6.0)
  ]
  -- 指定储存位置
  [LOCATION hdfs_path]
  -- 跟外部表配合使用，比如：映射HBase表，然后可以使用HQL对hbase数据进行查询，当然速度比较慢
  [TBLPROPERTIES (property_name=property_value, ...)]  (Note:  only available starting      with
0.6.0)

  [AS select_statement]  (Note: this feature is only available starting with 0.5.0.)
```

建表格式1：全部使用默认建表方式

```
create table students
(
    id bigint,
    name string,
    age int,
    gender string,
    clazz string
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
-- 必选，指定列分隔符
```

建表格式2：指定location （这种方式也比较常用）

```
create table students2
(
    id bigint,
    name string,
    age int,
    gender string,
    clazz string
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LOCATION '/input1';
-- 指定Hive表的数据的存储位置，一般在数据已经上传到HDFS，想要直接使用，会指定Location，
-- 通常Locaion会跟外部表一起使用，内部表一般使用默认的location
```

建表格式3：指定存储格式

```
create table students3
(
    id bigint,
    name string,
    age int,
    gender string,
    clazz string
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
STORED AS rcfile;
-- 指定储存格式为rcfile，inputFormat:RCFileInputFormat,outputFormat:RCFileOutputFormat，
-- 如果不指定，默认为textfile，
-- 注意：除textfile以外，其他的存储格式的数据都不能直接加载，需要使用从表加载的方式。
```

建表格式4：create table xxxx as select_statement(SQL语句) (这种方式比较常用)

注：

- 新建表不允许是外部表。
- select后面表需要是已经存在的表，建表同时会加载数据。
- 会启动mapreduce任务去读取源表数据写入新表

```
create table students4 as select * from students2;
```

建表格式5：create table xxxx like table_name 只想建表，不需要加载数据

```
create table students5 like students;
```

2、Hive加载数据

1.使用hdfs dfs -put 本地数据hive表对应的HDFS目录下

2.使用 load data inpath

从hdfs导入数据，路径可以是目录，会将目录下所有文件导入，但是文件格式必须一致

```
-- 将HDFS上的/input1目录下面的数据 移动至 students表对应的HDFS目录下
-- 注意是 移动! 移动! 移动!
load data inpath '/input1/students.txt' into table students;
-- 清空表
truncate table students;
```

从本地文件系统导入

```
-- 加上 local 关键字 可以将Linux本地目录下的文件 上传到 hive表对应HDFS 目录下 原文件不会被删除
load data local inpath '/usr/local/soft/data/students.txt' into table students;
-- overwrite 覆盖加载
load data local inpath '/usr/local/soft/data/students.txt' overwrite into table students;
```

3.create table xxx as SQL语句，表对表加载

```
create table test.aa as select * from test.bb
```

4.insert into table xxxx SQL语句 （没有as），表对表加载:

```
-- 将 students表的数据插入到students2
-- 这是复制 不是移动 students表中的表中的数据不会丢失
insert into table students2 select * from students;

-- 覆盖插入 把into 换成 overwrite
insert overwrite table students2 select * from students;
```

「注」：

1.如果建表语句没有指定存储路径，不管是外部表还是内部表，存储路径都是会默认在hive/warehouse/xx.db/表名的目录下。加载的数据如果在HDFS上会移动到该表的存储目录下。注意是移动，不是复制

2.删除外部表，文件不会删除，对应目录也不会删除

3、Hive 内部表（Managed tables）vs 外部表（External tables）

外部表和普通表的区别

1.外部表的路径可以自定义，内部表的路径需要在 hive/warehouse/目录下

2.删除表后，普通表数据文件和表信息都删除。外部表仅删除表信息

建表语句：

```
-- 内部表
create table students_internal
(
    id bigint,
    name string,
    age int,
    gender string,
    clazz string
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LOCATION '/input2';

-- 外部表
create external table students_external
(
    id bigint,
    name string,
    age int,
    gender string,
    clazz string
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LOCATION '/input3';
```

加载数据：

```
hive> dfs -put /usr/local/soft/data/students.txt /input2/;
hive> dfs -put /usr/local/soft/data/students.txt /input3/;
```

删除表：

```
hive> drop table students_internal;
Moved: 'hdfs://master:9000/input2' to trash at: hdfs://master:9000/user/root/.Trash/Current
OK
Time taken: 0.474 seconds
hive> drop table students_external;
OK
Time taken: 0.09 seconds
```

1.可以看出，删除内部表的时候，表中的数据（HDFS上的文件）会被同表的元数据一起删除；删除外部表的时候，只会删除表的元数据，而不会删除表中的数据（HDFS上的文件）

2.一般在公司中，使用外部表多一点，因为数据可以需要被多个程序使用，避免误删，通常外部表会结合location一起使用

3.外部表还可以将其他数据源中的数据 映射到 hive中，比如说：hbase，ElasticSearch…

4.设计外部表的初衷就是 让 表的元数据 与 数据 解耦

4、Hive 分区

分区表实际上是在表的目录下在以分区命名，建子目录；作用：进行分区裁剪，避免全表扫描，减少MapReduce处理的数据量，提高效率 。

一般在公司的hive中，所有的表基本上都是分区表，通常按日期分区、地域分区；分区表在使用的时候记得加上分区字段；分区也不是越多越好，一般不超过3级，根据实际业务衡量。

分区的概念和分区表：

分区表指的是在创建表时指定分区空间，实际上就是在hdfs上表的目录下再创建子目录。

在使用数据时如果指定了需要访问的分区名称，则只会读取相应的分区，避免全表扫描，提高查询效率。

建立分区表：

```
create external table students_pt1
(
    id bigint,
    name string,
    age int,
    gender string,
    clazz string
)
PARTITIONED BY(pt string)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
```

增加一个分区：

```
alter table students_pt1 add partition(pt='20210904');
```

删除一个分区：

```
alter table students_pt drop partition(pt='20210904');
```

查看某个表的所有分区

```
-- 推荐这种方式（直接从元数据中获取分区信息）
show partitions students_pt;

-- 不推荐
select distinct pt from students_pt;
```

往分区中插入数据：

```
insert into table students_pt partition(pt='20210902') select * from students;

load data local inpath '/usr/local/soft/data/students.txt' into table students_pt
partition(pt='20210902');
```

查询某个分区的数据：

```
-- 全表扫描，不推荐，效率低
select count(*) from students_pt;


-- 使用where条件进行分区裁剪，避免了全表扫描，效率高
select count(*) from students_pt where pt='20210101';


-- 也可以在where条件中使用非等值判断
select count(*) from students_pt where pt<='20210112' and pt>='20210110';
```

5、Hive动态分区

有的时候原始表中的数据里面包含了“日期字段 dt”，需要根据dt中不同的日期，分为不同的分区，将原始表改造成分区表。hive``默认不开启动态分区动态分区：根据数据中某几列的不同的取值 划分 不同的分区

```
-- 表示开启动态分区
hive> set hive.exec.dynamic.partition=true;

-- 表示动态分区模式：strict（需要配合静态分区一起使用）、nostrict

-- strict：insert into table students_pt partition(dt='anhui',pt) select ......,pt from
students;
hive> set hive.exec.dynamic.partition.mode=nostrict;

-- 表示支持的最大的分区数量为1000，可以根据业务自己调整
hive> set hive.exec.max.dynamic.partitions.pernode=1000;
```

建立原始表并加载数据

```
create table students_dt
(
    id bigint,
    name string,
    age int,
    gender string,
    clazz string,
    dt string
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
```

建立分区表并加载数据

```
create table students_dt_p
(
    id bigint,
    name string,
    age int,
    gender string,
    clazz string
)
PARTITIONED BY(dt string)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
```

使用动态分区插入数据

```
-- 分区字段需要放在 select 的最后，如果有多个分区字段 同理，
-- 它是按位置匹配，不是按名字匹配
insert into table students_dt_p partition(dt) select id,name,age,gender,clazz,dt from
students_dt;

-- 比如下面这条语句会使用age作为分区字段，而不会使用student_dt中的dt作为分区字段
insert into table students_dt_p partition(dt) select id,name,age,gender,dt,age from students_dt;
```

多级分区

```
create table students_year_month
(
    id bigint,
    name string,
    age int,
    gender string,
    clazz string,
```

```
    year string,
    month string
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';

create table students_year_month_pt
(
    id bigint,
    name string,
    age int,
    gender string,
    clazz string
)
PARTITIONED BY(year string,month string)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';

insert into table students_year_month_pt partition(year,month) select
id,name,age,gender,clazz,year,month from students_year_month;
```

6、Hive分桶

分桶实际上是对文件（数据）的进一步切分；Hive默认关闭分桶；

分桶的作用：在往分桶表中插入数据的时候，会根据 clustered by 指定的字段 进行hash分组 对指定的buckets个数 进行取余，进而可以将数据分割成buckets个数个文件，以达到数据均匀分布，可以解决Map端的“数据倾斜”问题，方便我们取抽样数据，提高Map join效率；**分桶字段** 需要根据业务进行设定

开启分桶开关

```
hive> set hive.enforce.bucketing=true;
```

建立分桶表

```
create table students_buks
(
    id bigint,
    name string,
    age int,
    gender string,
    clazz string
)
CLUSTERED BY (clazz) into 12 BUCKETS
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
```

往分桶表中插入数据

```
-- 直接使用load data 并不能将数据打散
load data local inpath '/usr/local/soft/data/students.txt' into table students_buks;

-- 需要使用下面这种方式插入数据，才能使分桶表真正发挥作用
insert into students_buks select * from students;
```

7、Hive连接JDBC

启动*hiveserver2*的服务

```
hive --service hiveserver2 &
```

新建*maven*项目并添加两个依赖

```
    <dependency>
        <groupId>org.apache.hadoop</groupId>
        <artifactId>hadoop-common</artifactId>
        <version>2.7.6</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.apache.hive/hive-jdbc -->
    <dependency>
        <groupId>org.apache.hive</groupId>
        <artifactId>hive-jdbc</artifactId>
        <version>1.2.1</version>
    </dependency>
```

编写*Java*通过*JDBC*连接*Hive*

```
import java.sql.*;

public class HiveJDBC {
    public static void main(String[] args) throws ClassNotFoundException, SQLException {
        Class.forName("org.apache.hive.jdbc.HiveDriver");
        Connection conn = DriverManager.getConnection("jdbc:hive2://master:10000/test3");
        Statement stat = conn.createStatement();
        ResultSet rs = stat.executeQuery("select * from students limit 10");
        while (rs.next()) {
            int id = rs.getInt(1);
            String name = rs.getString(2);
            int age = rs.getInt(3);
            String gender = rs.getString(4);
            String clazz = rs.getString(5);
            System.out.println(id + "," + name + "," + age + "," + gender + "," + clazz);
        }
        rs.close();
```

```
        stat.close();
        conn.close();
    }
}
```

三、Hive的数据类型

1、基本数据类型

数值型：

```
TINYINT — 微整型，只占用1个字节，只能存储0-255的整数。
SMALLINT- 小整型，占用2个字节，存储范围-32768 到 32767。
INT- 整型，占用4个字节，存储范围-2147483648到2147483647。
BIGINT- 长整型，占用8个字节，存储范围-2^63到2^63-1。
```

布尔型

```
BOOLEAN — TRUE/FALSE
```

浮点型

```
FLOAT- 单精度浮点数。
DOUBLE- 双精度浮点数。
```

字符串型

```
STRING- 不设定长度。
```

2、日期类型

时间戳 timestamp

日期 date

```
create table testDate(
    ts timestamp
    ,dt date
) row format delimited fields terminated by ',';

-- 2021-01-14 14:24:57.200,2021-01-11
```

时间戳与时间字符串转换

```
-- from_unixtime 传入一个时间戳以及pattern（yyyy-MM-dd）
-- 可以将 时间戳转换成对应格式的字符串
select from_unixtime(1630915221,'yyyy年MM月dd日 HH时mm分ss秒')

-- unix_timestamp 传入一个时间字符串以及pattern，
-- 可以将字符串按照pattern转换成时间戳
select unix_timestamp('2021年09月07日 11时00分21秒','yyyy年MM月dd日 HH时mm分ss秒');
select unix_timestamp('2021-01-14 14:24:57.200')
```

3、复杂数据类型

主要有三种复杂数据类型：Structs，Maps，Arrays

四、Hive HQL使用语法

SQL``语言可以分为5大类：

(1）DDL(Data Definition Language) 数据定义语言 用来定义数据库对象：数据库，表，列等。关键字：create，drap,alter等

(2）DML(Data Manipulation Language) 数据操作语言 用来对数据库中表的数据进行增删改。关键字：insert,delete,update等

(3）DQL(Data Query Language)数据查询语言 用来查询数据库表的记录（数据）。关键字：select,where 等

(4）DCL(Data Control Language) 数据控制语言 用来定义数据库的访问权限和安全级别，及创建用户。关键字：GRANT，REVOKE等

(5)TCL(Transaction ControlLanguage) 事务控制语言 T CL经常被用于快速原型开发、脚本编程、GUI和测试等方面，关键字: commit、rollback等。

1、HQL语法-DDL

```
-- 创建数据库
create database xxxxx;
-- 查看数据库
show databases;
-- 删除数据库
drop database tmp;
-- 强制删除数据库：
drop database tmp cascade;
-- 查看表：
SHOW TABLES;
-- 查看表的元信息：
    desc test_table;
    describe extended test_table;
    describe formatted test_table;
-- 查看建表语句：
show create table table_XXX
-- 重命名表：
    alter table test_table rename to new_table;
-- 修改列数据类型：
alter table lv_test change column colxx string;

-- 增加、删除分区：
    alter table test_table add partition (pt=xxxx)
```

```
alter table test_table drop if exists partition(...);
```

2、HQL语法-DML

```
where  -- 用于过滤，分区裁剪，指定条件

join  -- 用于两表关联，left outer join ，join，mapjoin（1.2版本后默认开启）

group by  -- 用于分组聚合，通常结合聚合函数一起使用

order by  -- 用于全局排序，要尽量避免排序，是针对全局排序的，即对所有的reduce输出是有序的

sort by  -- 当有多个reduce时，只能保证单个reduce输出有序，不能保证全局有序

cluster by = distribute by + sort by

distinct -- 去重
```

五、Hive HQL使用注意

1.count(1)、count(*)和count(字段名)执行效果上的区别

- count(*)包括了所有的列，相当于行数，在统计结果的时候，不会忽略列值为NULL
- count(1)包括了忽略所有列，用1代表代码行，在统计结果的时候，不会忽略列值为NULL
- count(列名)只包括列名那一列，在统计结果的时候，会忽略列值为空（这里的空不是只空字符串或者0，而是表示null）的计数，即某个字段值为NULL时，不统计。

2.HQL 执行优先级：`from``、where、 group by 、having、order by、join、select 、limit`

3.where 条件里不支持不等式子查询，实际上是支持 in、not in、exists、not exists

4.hive中大小写不敏感

5.在hive中，数据中如果有null字符串，加载到表中的时候会变成 null （不是字符串）如果需要判断 `null`，使用 某个字段名 `is null` 这样的方式来判断;或者使用 `nvl()` 函数，不能 直接 某个字段名 `== null`

6.使用explain查看SQL执行计划

六、Hive 的函数使用

1、Hive-常用函数

（1）关系运算

```
// 等值比较 = == <=>
// 不等值比较 != <>
// 区间比较：select * from default.students where id between 1500100001 and 1500100010;
// 空值/非空值判断：is null、is not null、nvl()、isnull()
// like、rlike、regexp用法
```

（2）数值计算

```
取整函数(四舍五入)：round
向上取整：ceil
向下取整：floor
```

（3） 条件函数

if：if(表达式,如果表达式成立的返回值,如果表达式不成立的返回值)

```
select if(1>0,1,0);
select if(1>0,if(-1>0,-1,1),0);
```

COALESCE

```
select COALESCE(null,'1','2'); -- 1 从左往右 一次匹配 直到非空为止
select COALESCE('1',null,'2'); -- 1
```

case when … then … else … end

```
select  score
        ,case when score>120 then '优秀'
              when score>100 then '良好'
              when score>90 then '及格'
        else '不及格'
        end as pingfen
from default.score limit 20;
--  注意条件的顺序
```

（4）日期函数

```
select from_unixtime(1610611142,'YYYY/MM/dd HH:mm:ss');

select from_unixtime(unix_timestamp(),'YYYY/MM/dd HH:mm:ss');

-- '2021年01月14日' -> '2021-01-14'
select from_unixtime(unix_timestamp('2021年01月14日','yyyy年MM月dd日'),'yyyy-MM-dd');
-- "04牛2021数加16逼" -> "2021/04/16"
select from_unixtime(unix_timestamp("04牛2021数加16逼","MM牛yyyy数加dd逼"),"yyyy/MM/dd");
```

(5) 字符串函数

```
concat('123','456'); -- 123456
concat('123','456',null); -- NULL

select concat_ws('#','a','b','c'); -- a#b#c
select concat_ws('#','a','b','c',NULL); -- a#b#c 可以指定分隔符，并且会自动忽略NULL
select concat_ws("|",cast(id as string),name,cast(age as string),gender,clazz) from students
limit 10;

select substring("abcdefg",1); -- abcdefg HQL中涉及到位置的时候 是从1开始计数
-- '2021/01/14' -> '2021-01-14'
select concat_ws("-
",substring('2021/01/14',1,4),substring('2021/01/14',6,2),substring('2021/01/14',9,2));

select split("abcde,fgh",","); -- ["abcde","fgh"]
select split("a,b,c,d,e,f",",")[2]; // c

select explode(split("abcde,fgh",",")); -- abcde
          --  fgh

-- 解析json格式的数据
select get_json_object('{"name":"zhangsan","age":18,"score":[{"course_name":"math","score":100},
{"course_name":"english","score":60}]}',"$.score[0].score"); -- 100
```

2、Hive-高级函数

（1）窗口函数（开窗函数）：用户分组中开窗

在sql中有一类函数叫做`聚合函数,例如sum()、avg()、max()等等,这类函数可以将多行数据按照规则聚集为一行,一般来讲聚集后的行数是要少于聚集前的行数的.

但是有时想要既显示聚集前的数据,又要显示聚集后的数据,这时我们便引入了窗口函数。（开创函数，我们一般用于分组中求 TopN问题）

```
-- 数据：
111,69,class1,department1
112,80,class1,department1
113,74,class1,department1
```

```
114,94,class1,department1
115,93,class1,department1
121,74,class2,department1
122,86,class2,department1
123,78,class2,department1
124,70,class2,department1
211,93,class1,department2
212,83,class1,department2
213,94,class1,department2
214,94,class1,department2
215,82,class1,department2
216,74,class1,department2
221,99,class2,department2
222,78,class2,department2
223,74,class2,department2
224,80,class2,department2
225,85,class2,department2
```

```
建表：
create table new_score(
    id  int
    ,score int
    ,clazz string
    ,department string
) row format delimited fields terminated by ",";
```

「row_number()：无并列排名」

```
-- 使用格式：
 select xxxx, row_number() over(partition by 分组字段 order by 排序字段 desc) as rn from tb group by xxxx
```

「dense_rank()：有并列排名，并且依次递增」

「rank()：有并列排名，不依次递增」

「percent_rank()：(rank的结果-1)/(分区内数据的个数-1)」

「cume_dist()：计算某个窗口或分区中某个值的累积分布。」

假定升序排序，则使用以下公式确定累积分布：小于等于当前值x的行数 / 窗口或partition分区内的总行数。

其中，x 等于 order by 子句中指定的列的当前行中的值。

「NTILE(n)：对分区内数据再分成n组，然后打上组号」「max()、min()、avg()、count()、sum()等函数：是基于每个partition分区内的数据做对应的计算」

窗口帧：用于从分区中选择指定的多条记录，供窗口函数处理

Hive 提供了两种定义窗口帧的形式：ROWS 和 RANGE。两种类型都需要配置上界和下界。
例如，ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW 表示选择分区起始记录到当前记录的所有行；
SUM(close) RANGE BETWEEN 100 PRECEDING AND 200 FOLLOWING 则通过 字段差值 来进行选择。
如当前行的 close 字段值是 200，那么这个窗口帧的定义就会选择分区中 close 字段值落在 100 至 400 区间的记录。
以下是所有可能的窗口帧定义组合。如果没有定义窗口帧，则默认为 RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW。
注意：窗口帧只能运用在max、min、avg、count、sum、FIRST_VALUE、LAST_VALUE这几个窗口函数上

```
-- 「测试」
SELECT id
     ,score
     ,clazz
     ,SUM(score) OVER w as sum_w
     ,round(avg(score) OVER w,3) as avg_w
     ,count(score) OVER w as cnt_w
FROM new_score
WINDOW w AS (PARTITION BY clazz ORDER BY score rows between 2 PRECEDING and 2 FOLLOWING);
```

```
-- 「测试2：」
select  id
        ,score
        ,clazz
        ,department
        ,row_number() over (partition by clazz order by score desc) as rn_rk
        ,dense_rank() over (partition by clazz order by score desc) as dense_rk
        ,rank() over (partition by clazz order by score desc) as rk
        ,percent_rank() over (partition by clazz order by score desc) as percent_rk
        ,round(cume_dist() over (partition by clazz order by score desc),3) as cume_rk
        ,NTILE(3) over (partition by clazz order by score desc) as ntile_num
        ,max(score) over (partition by clazz order by score desc range between 3 PRECEDING and
11 FOLLOWING) as max_p
from new_score;
```

```
Total MapReduce CPU Time Spent: 2 seconds 600 msec
OK
114     94      class1  department1     1       1       1       0.0     0.273   1       94
214     94      class1  department2     2       1       1       0.0     0.273   1       94
213     94      class1  department2     3       1       1       0.0     0.273   1       94
211     93      class1  department2     4       2       4       0.3     0.455   1       94
115     93      class1  department1     5       2       4       0.3     0.455   2       94
212     83      class1  department2     6       3       6       0.5     0.545   2       83
215     82      class1  department2     7       4       7       0.6     0.636   2       83
112     80      class1  department1     8       5       8       0.7     0.727   2       83
113     74      class1  department1     9       6       9       0.8     0.909   3       74
216     74      class1  department2     10      6       9       0.8     0.909   3       74
111     69      class1  department1     11      7       11      1.0     1.0     3       69
221     99      class2  department2     1       1       1       0.0     0.111   1       99
122     86      class2  department1     2       2       2       0.125   0.222   1       86
225     85      class2  department2     3       3       3       0.25    0.333   1       86
224     80      class2  department2     4       4       4       0.375   0.444   2       80
123     78      class2  department1     5       5       5       0.5     0.667   2       80
222     78      class2  department2     6       5       5       0.5     0.667   2       80
121     74      class2  department1     7       6       7       0.75    0.889   3       74
223     74      class2  department2     8       6       7       0.75    0.889   3       74
124     70      class2  department1     9       7       9       1.0     1.0     3       70
Time taken: 23.843 seconds, Fetched: 20 row(s)
```

「LAG(col,n)：往前第n行数据 LEAD(col,n)：往后第n行数据 FIRST_VALUE：取分组内排序后，截止到当前行，第一个值 LAST_VALUE：取分组内排序后，截止到当前行，最后一个值，对于并列的排名，取最后一个 测试3：」

```
select  id
        ,score
        ,clazz
        ,department
        ,lag(id,2) over (partition by clazz order by score desc) as lag_num
        ,LEAD(id,2) over (partition by clazz order by score desc) as lead_num
        ,FIRST_VALUE(id) over (partition by clazz order by score desc) as first_v_num
        ,LAST_VALUE(id) over (partition by clazz order by score desc) as last_v_num
        ,NTILE(3) over (partition by clazz order by score desc) as ntile_num
from new_score;
```

```
Total MapReduce CPU Time Spent: 2 seconds 310 msec
OK
114     94      class1  department1     NULL    213     114     213     1
214     94      class1  department2     NULL    211     114     213     1
213     94      class1  department2     114     115     114     213     1
211     93      class1  department2     214     212     114     115     1
115     93      class1  department1     213     215     114     115     2
212     83      class1  department2     211     112     114     212     2
215     82      class1  department2     115     113     114     215     2
112     80      class1  department1     212     216     114     112     2
113     74      class1  department1     215     111     114     216     3
216     74      class1  department2     112     NULL    114     216     3
111     69      class1  department1     113     NULL    114     111     3
221     99      class2  department2     NULL    225     221     221     1
122     86      class2  department1     NULL    224     221     122     1
225     85      class2  department2     221     123     221     225     1
224     80      class2  department2     122     222     221     224     2
123     78      class2  department1     225     121     221     222     2
222     78      class2  department2     224     223     221     222     2
121     74      class2  department1     123     124     221     223     3
223     74      class2  department2     222     NULL    221     223     3
124     70      class2  department1     121     NULL    221     124     3
Time taken: 24.295 seconds, Fetched: 20 row(s)
hive>
```

（2）*Hive* 行转列

使用关键字：lateral view explode

```
-- 建表：
create table testArray2(
    name string,
    weight array<string>
)row format delimited
fields terminated by '\t'
COLLECTION ITEMS terminated by ',';

-- 样例数据：
-- 孙悟空 "150","170","180"
-- 唐三藏 "150","180","190"
```

```
select name,col1  from testarray2 lateral view explode(weight) t1 as col1;
```

```
hive> select name,col1  from testarray2 lateral view explode(weight) t1 as col1;
OK
孙悟空  "150"
孙悟空  "170"
孙悟空  "180"
唐三藏  "150"
唐三藏  "180"
唐三藏  "190"
Time taken: 0.095 seconds, Fetched: 6 row(s)
```

```
select key from (select explode(map('key1',1,'key2',2,'key3',3)) as (key,value)) t;
```

```
hive> select key from (select explode(map('key1',1,'key2',2,'key3',3)) as (key,value)) t;
OK
key1
key2
key3
Time taken: 0.497 seconds, Fetched: 3 row(s)
```

```
select name,col1,col2  from testarray2 lateral view explode(map('key1',1,'key2',2,'key3',3)) t1
as col1,col2;
```

```
hive> select name,col1,col2  from testarray2 lateral view explode(map('key1',1,'key2',2,'key3',3))
t1 as col1,col2;
OK
孙悟空  key1    1
孙悟空  key2    2
孙悟空  key3    3
唐三藏  key1    1
唐三藏  key2    2
唐三藏  key3    3
Time taken: 0.084 seconds, Fetched: 6 row(s)
```

```
select name,pos,col1  from testarray2 lateral view posexplode(weight) t1 as pos,col1;
```

```
hive> select name,pos,col1  from testarray2 lateral view posexplode(weight) t1 as pos,col1;
OK
孙悟空  0       "150"
孙悟空  1       "170"
孙悟空  2       "180"
唐三藏  0       "150"
唐三藏  1       "180"
唐三藏  2       "190"
Time taken: 0.052 seconds, Fetched: 6 row(s)
hive>
```

（3）*Hive* 列转行

```
-- 数据：

孙悟空  150
孙悟空  170
孙悟空  180
唐三藏  150
唐三藏  180
唐三藏  190
```

```
-- 建表:
create table testLieToLine(
    name string,
    col1 int
)row format delimited
fields terminated by '\t';
```

「测试1:」

```
select name,collect_list(col1) from testLieToLine group by name;
```

「测试2:」

```
select  t1.name, collect_list(t1.col1)
from (
    select  name
            ,col1
    from testarray2
    lateral view explode(weight) t1 as col1
) t1 group by t1.name;
```

```
Total MapReduce CPU Time Spent: 2 seconds 570 msec
OK
唐三藏  ["\"150\"","\"180\"","\"190\""]
孙悟空  ["\"150\"","\"170\"","\"180\""]
Time taken: 22.648 seconds, Fetched: 2 row(s)
hive>
```

（4）*Hive*自定义函数*UserDefineFunction*

UDF：一进一出

创建maven项目，并加入依赖

```
<dependency>
     <groupId>org.apache.hive</groupId>
     <artifactId>hive-exec</artifactId>
     <version>1.2.1</version>
</dependency>
```

编写代码，继承org.apache.hadoop.hive.ql.exec.UDF，实现evaluate方法，在evaluate方法中实现自己的逻辑

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

public class HiveUDF extends UDF {
    // hadoop => #hadoop$
    public String evaluate(String col1) {
    // 给传进来的数据 左边加上 # 号 右边加上 $
        String result = "#" + col1 + "$";
        return result;
    }
}
```

打成jar包并上传至Linux虚拟机(小北路径：/usr/local/soft/jars/)\

在hive shell中，使用 add jar 路径将jar包作为资源添加到hive环境中

```
add jar /usr/local/soft/jars/HiveUDF2-1.0.jar;
```

使用jar包资源注册一个临时函数，fxxx1是你的函数名，'MyUDF'是主类名

```
create temporary function fxxx1 as 'MyUDF';
```

使用函数名处理数据

```
select fxx1(name) as fxx_name from students limit 10;
```

```
#施笑槐$
#吕金鹏$
#单乐蕊$
#葛德曜$
#宣谷芹$
#边昂雄$
#尚孤风$
#符半双$
#沈德昌$
#羿彦昌$
```

「UDTF：一进多出」

样例数据:

```
"key1:value1,key2:value2,key3:value3"

key1 value1

key2 value2

key3 value3
```

「方法一：使用 explode+split」

```
select split(t.col1,":")[0],split(t.col1,":")[1]
from (select
explode(split("key1:value1,key2:value2,key3:value3",",")) as col1) t;
```

「方法二：自定UDTF」

```
//自定义代码

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.StructObjectInspector;
import org.apache.hadoop.hive.serde2.objectinspector.primitive.PrimitiveObjectInspectorFactory;

import java.util.ArrayList;
```

```
public class HiveUDTF extends GenericUDTF {
    // 指定输出的列名 及 类型
    @Override
    public StructObjectInspector initialize(StructObjectInspector argOIs) throws
UDFArgumentException {
        ArrayList<String> filedNames = new ArrayList<String>();
        ArrayList<ObjectInspector> filedObj = new ArrayList<ObjectInspector>();
        filedNames.add("col1");
        filedObj.add(PrimitiveObjectInspectorFactory.javaStringObjectInspector);
        filedNames.add("col2");
        filedObj.add(PrimitiveObjectInspectorFactory.javaStringObjectInspector);
        return ObjectInspectorFactory.getStandardStructObjectInspector(filedNames, filedObj);
    }

    // 处理逻辑 my_udtf(col1,col2,col3)
    // "key1:value1,key2:value2,key3:value3"
    // my_udtf("key1:value1,key2:value2,key3:value3")
    public void process(Object[] objects) throws HiveException {
        // objects 表示传入的N列
        String col = objects[0].toString();
        // key1:value1  key2:value2  key3:value3
        String[] splits = col.split(",");
        for (String str : splits) {
            String[] cols = str.split(":");
            // 将数据输出
            forward(cols);
        }

    }

    // 在UDTF结束时调用
    public void close() throws HiveException {

    }
}
```

「SQL:」

```
select my_udtf("key1:value1,key2:value2,key3:value3");
```

「举例说明：」

```
字段：id,col1,col2,col3,col4,col5,col6,col7,col8,col9,col10,col11,col12 共13列

数据：a,1,2,3,4,5,6,7,8,9,10,11,12 b,11,12,13,14,15,16,17,18,19,20,21,22
c,21,22,23,24,25,26,27,28,29,30,31,32

转成3列：id,hours,value

例如：a,1,2,3,4,5,6,7,8,9,10,11,12 a,0时,1 a,2时,2 a,4时,3 a,6时,4
```

```
-- 建表:
create table udtfData(
    id string
    ,col1 string
    ,col2 string
    ,col3 string
```

```
    ,col4 string
    ,col5 string
    ,col6 string
    ,col7 string
    ,col8 string
    ,col9 string
    ,col10 string
    ,col11 string
    ,col12 string
)row format delimited fields terminated by ',';
```

「java代码：」

```
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.StructObjectInspector;
import org.apache.hadoop.hive.serde2.objectinspector.primitive.PrimitiveObjectInspectorFactory;

import java.util.ArrayList;

public class HiveUDTF2 extends GenericUDTF {
    @Override
    public StructObjectInspector initialize(StructObjectInspector argOIs) throws
UDFArgumentException {
        ArrayList<String> filedNames = new ArrayList<String>();
        ArrayList<ObjectInspector> fieldObj = new ArrayList<ObjectInspector>();
        filedNames.add("col1");
        fieldObj.add(PrimitiveObjectInspectorFactory.javaStringObjectInspector);
        filedNames.add("col2");
        fieldObj.add(PrimitiveObjectInspectorFactory.javaStringObjectInspector);
        return ObjectInspectorFactory.getStandardStructObjectInspector(filedNames, fieldObj);
    }

    public void process(Object[] objects) throws HiveException {
        int hours = 0;
        for (Object obj : objects) {
            hours = hours + 1;
            String col = obj.toString();
            ArrayList<String> cols = new ArrayList<String>();
            cols.add(hours + "时");
            cols.add(col);
            forward(cols);
        }
    }

    public void close() throws HiveException {

    }
}
```

「添加jar资源:」

```
add jar /usr/local/soft/HiveUDF2-1.0.jar;
```

「注册udtf函数：」

```
create temporary function my_udtf as 'MyUDTF';
```

「SQL:」

```
select id
       ,hours
       ,value from udtfData lateral view
my_udtf(col1,col2,col3,col4,col5,col6,col7,col8,col9,col10,col11,col12)
t as hours,value ;
```

「UDAF：多进一出」

3、Hive 中的wordCount

建表：

```
create table words(
    words string
)row
```

数据：

```
hello,java,hello,java,scala,python
hbase,hadoop,hadoop,hdfs,hive,hive
hbase,hadoop,hadoop,hdfs,hive,hive
```

```
hive> select * from words;
OK
hello,java,hello,java,scala,python
hbase,hadoop,hadoop,hdfs,hive,hive
hbase,hadoop,hadoop,hdfs,hive,hive
Time taken: 0.056 seconds, Fetched: 3 row(s)
```

```
select word,count(*) from (select explode(split(words,',')) word from words) a group by a.word;
```

```
Total MapReduce CPU Time Spent: 2 seconds 510 msec
OK
hadoop  4
hbase   2
hdfs    2
hello   2
hive    4
java    2
python  1
scala   1
Time taken: 25.47 seconds, Fetched: 8 row(s)
```

七、Hive 的Shell使用

第一种shell

```
hive -e "select * from test03.students limit 10"
```

第二种shell

```
hive -f hql文件路径
-- 将HQL写在一个文件里，再使用 -f 参数指定该文件
```