**hive笔记整理**

**了解**

|  |
| --- |
| Hive是基于Hadoop的一个数据仓库工具(离线)，可以将结构化的数据文件映射为一张数据库表，并提供类SQL（HQL）查询功能。  **Hive的本质：将SQL转化为MapReduce程序。 1）Hive处理的数据存储在HDFS 2）Hive分析数据底层的实现是MapReduce**  **3）执行程序运行在Yarn上** |

**Hive的启动**

|  |
| --- |
| 1:hadoop集群的机器配置时间同步  2:启动hdfs start-dfs.sh  3:启动MySQL service mysqld start  4:进入hive |

**加载数据**

|  |
| --- |
| 命令格式：**load data [local] inpath ‘path’ into table tName;**  hive (default)> load data local inpath '/root/表名.log' into table 表名; #本地磁盘文件  hive (default) > load data inpath '/表名.log' into table表名; # hdfs的文件  hive (default)> load data local inpath '/root/t\_order.log' overwrite into table t\_order; # 覆盖数据  如果加载的是本地的文件，那么相当于复制文件到表对应的数据目录下。  如果加载的是hdfs的文件，那么相当于移动文件。  如果想覆盖原有的数据： overwrite into |

**建表语句**

|  |
| --- |
| 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 row\_format]  [STORED AS file\_format]  [LOCATION hdfs\_path]  简单的建表语句为：  create table 表名(字段名1 数据类型,字段名2 数据类型，...)  row format delimited  fields terminated by '分隔符'; |

**内部表**

|  |
| --- |
| 内部表(MANAGED\_TABLE)：表目录按照hive的规范来部署，位于hive的仓库目录/user/hive/warehouse中  Hive中，表的Metadata信息全部存储在MySQL中。而存储在MySQL中的数据是没有校验过程的。  也就是说，你可以创建一个逻辑错误的表，两个表指向同一个HDFS文件也是有可能的。  所以，在使用CREATE TABLE的时候，LOCATION必须要指定成Path/表名  例：create table tName( id string )  row format delimited fields terminated by ‘,‘  location ‘/xxx/tName’;  加上location之后，表对应的数据目录是指定的目录。 |

**外部表**

|  |
| --- |
| 创建表时，通过external关键字来区分内部表和外部表。  通过location 关键字来指定表对应的结构化数据的存储位置。（不指定就是默认的）  例：create external table t\_product(pid string,pname string,price float,pnum int)  row format delimited fields terminated by ','  location '/prologs';  **外部表和内部表的特性差别：**  内部表的目录在hive的仓库目录中 VS 外部表的目录由用户指定  如果删除内部表，表没了，数据也没了。  如果删除外部表，表没了，数据还在。外部表建表时需使用关键字external 。 |

**分区表**

|  |
| --- |
| **分区表的实质是**：在表目录中为数据文件创建分区子目录，以便于在查询时，MR程序可以针对指定的分区子目录中的数据进行处理，缩减读取数据的范围，提高效率！  例：create table t\_adlogs(session string,mac string,appversion string,url string)  partitioned by (year string ,month string,day string)  row format delimited fields terminated by ',';  分区的字段可以当成正常的查询字段。  分区的字段，不能在创建表的字段中出现。  **加载数据：**  load data local inpath '/root/ad20181010.logs' into table t\_adlogs partition(year="2018",month="10",day="11");  **查询： 可以直接指定分区的字段进行过滤**  select \* from t\_adlogs where year = 2018 and month = 10 and day = 11;  **为分区表，添加一个分区，使用外部的一个数据源。**  **数据源不改变。**  alter table t\_adlogs add partition(year="2018",month=10,day=12) location '/adlogs';  **多个分区字段示例**  **建表：**  create table t\_partition(id int,name string,age int)  partitioned by(department string,sex string,howold int)  row format delimited fields terminated by ',';  **导数据**：  load data local inpath '/root/p1.dat' into table t\_partition partition(department='xiangsheng',sex='male',howold=20); |

**数据表的其他常见操作**

**1，克隆表**

根据已有的表来创建新表

hive (default)> create table t\_order\_cp like t\_order;

语法：create table 已存在的表 like 新表

只会创建新的表，表结构和克隆的表结构一样，但是没有数据。

**2，深度克隆表**

|  |
| --- |
| hive (default)> create table t\_order\_cp2 as select oid ,create\_time from t\_order; |

克隆2个字段，然后所有的数据也会复制到新表中。

创建表 + hadoop fs -put 数据

创建表 load

创建表 insert … select …

|  |
| --- |
| hive (default)> insert into tmp\_t\_price select oid,price from t\_order; |

**覆盖数据：**

|  |
| --- |
| hive (default)> insert overwrite table tmp\_t\_price select oid,price from t\_order; |

**数据类型**

|  |
| --- |
| **1:array**  **2:map 3:struct** |

**函数（配合select使用）**

|  |  |
| --- | --- |
| **常用函数**  **1：日期功能**  **from\_unixtime（bigint unixtime [，string format]）** 将unix epoch（1970-01-01 00:00:00 UTC）的秒数转换为表示当前系统时区中该时刻的时间戳的字符串，格式为“1970-01-01 00:00： 00" 。  **unix\_timestamp（字符串日期）** yyyy-MM-dd HH:mm:ss使用默认时区和默认语言环境将格式的时间字符串转换为Unix时间戳（以秒为单位），如果失败则返回0  **unix\_timestamp（字符串日期，字符串模式）** 将给定模式的时间字符串（转换为Unix时间戳（以秒为单位），如果失败则返回0 **to\_date（字符串时间戳）**  返回时间戳字符串的日期部分（pre-Hive 2.1.0）：to\_date（“1970-01-01 00:00:00”）=“1970-01-01”。从Hive 2.1.0开始，返回一个日期对象。  **current\_timestamp;** 返回值类型：timestamp，获取当前的时间戳(详细时间信息)  **current\_date;**  返回值类型：date，获取当前的日期  **datediff（string enddate，string startdate）** 返回从startdate到enddate的天数：datediff（'2009-03-01'，'2009-02-27'）= 2。**date\_add（日期/时间戳/字符串 startdate，tinyint / smallint / int天）**添加开始日期的天数：date\_add（'2008-12-31'，1）='2009-01-01'。 **date\_sub（日期/时间戳/字符串 startdate，tinyint / smallint / int天）**减去开始日期的天数  **date\_format（日期/时间戳/字符串ts，字符串fmt）** 将日期/时间戳/字符串转换为字符串值，格式为日期格式fmt指定的格式。  **2: 转换函数**  **Cast(str,as 新的类型）** 将表达式expr的结果转换为<type>  **3：数学数算**   |  | | --- | | round() 四舍五入 cell() 向上取整 floor() 向下取整 sqrt（） 求平方根  abs（） 绝对值 **greatest(..) 单行函数，求最大值**   **least(3,5,6) ; 求多个输入参数中的最小值（单行）**  **max()分组聚合函数，最大值（单列） min()分组聚合函数，最小值 （单列** |   **4：字符串(索引从1开始） get\_json\_object(string json\_string, string path)** 基于指定的json路径从json字符串中提取json对象，并返回提取的json对象的json字符串。如果输入的json字符串无效，它将返回null。注意：json路径只能包含字符[0-9a-z\_]，即没有大写或特殊字符。此外，键\*不能以数字开头。  **locate(string substr, string str[, int pos])** 从URL返回指定的部分。partToExtract的有效值包括HOST，PATH，QUERY，REF，PROTOCOL，AUTHORITY，FILE和USERINFO。此外，通过提供密钥作为第三个参数，可以提取QUERY中特定键的值  **concat\_ws（字符串SEP，字符串A，字符串B ...）** 字符串拼接，但使用自定义分隔符SEP  **concat(string A, string B...)**  拼接字符串  **split(string str, string regex)** 切割字符串  **reverse(String str)**  字符串反转  **substr(string str, int start)**  截取子串  **substring\_index（字符串A，字符串delim，int count）** 在分隔符delim的计数出现之前返回字符串A的子字符串。如果count为正数，则返回最终分隔符左侧的所有内容（从左侧开始计算）。如果count为负数，则返回最终分隔符右侧的所有内容（从右侧开始计算）。在搜索delim时，Substring\_index执行区分大小写的匹配。  **substr(string, int start, int len)** 返回从起始位置开始的长度为len的字节数组的子字符串或切片。  **ltrim(string A)** 左边去空格  **rtrim(string A)** 右边去空格  **regexp\_replace(string A, string B,string C)**  将字符串A中的符合java正则表达式B的部分替换为c(oracle不支持）  **regexp\_extract(string subject, string pattern, int index)**将字符串subject按照pattern正则表达式的规则拆分，返回index指定的字符。返回使用模式提取的字符串。。  **find\_in\_set(string str, string strList)** 返回str在strlist第一次出现的位置，strlist是用逗号分割的字符串。如果没有找该str字符，则返回0  **str\_to\_map（text [，delimiter1，delimiter2]）**字符串转为map格式，使用两个分隔符将文本拆分为键值对。Delimiter1将文本分成KV对，Delimiter2分割每个KV对。对于delimiter1，默认分隔符为'，'，对于delimiter2，默认分隔符为'：'。 **5：条件控制 1：if语句 语法：if(表达式，语句1，语句2） 理解：当表达式结果为true时，执行语句1，当表达式结果为false时，执行语句2  2：case when语句 case 字段 when 值1 then 返回值1**  **when 值2 then 返回值2**  **else 返回值3**  **end as 别名**  **或者**  **case**  **when 表达式1 then 返回值1**  **when 表达式2 then 返回值2**  **else 返回值3**  **end as 别名 6：集合函数**  **array(...) 构造一个...类型的数组**  **array\_contains(Array<T>, value) 返回boolean值，是否包含**  **sort\_array(Array<T>) 返回排序后的数组**  **size(Array<T>) 返回一个集合的长度，int值**  **size(Map<K.V>) 返回一个map的元素个数，int值**  **size(array<T>) 返回一个数组的长度,int值**  **map\_keys(Map<K.V>) 返回一个map字段的所有key，结果类型为：数组**  **map\_values(Map<K.V>) 返回一个map字段的所有value，结果类型为：数组 7：常见分组聚合函数**  sum(字段) : 求这个字段在一个组中的所有值的和  avg(字段) ： 求这个字段在一个组中的所有值的平均值  max(字段) ：求这个字段在一个组中的所有值的最大值  min(字段) ：求这个字段在一个组中的所有值的最小值  disctinct ：去重  **count()：求一个组中的满足某条件的数据条数！只要count中的表达式的返回值不是null，都会进行计算。**  **collect\_set() :将某个字段在一组中的所有值形成一个集合（数组）返回，去重。**  **collect\_list() :将某个字段在一组中的所有值形成一个集合（数组）返回，不去重。**  with rollup：逐层统计指定字段 with cube：统计指定字段的所有组合（包括NULL）  grouping sets：只统计指定字段 **8:行转列与列转行**  **列转行：**  **例： select id,name,collect\_list(subject)from t13 group by id,name; 不带去重功能**  **select id,name,collect\_set(subject)from t13 group by id,name; 带去重功能**  **表生成函数**  **炸裂函数 explode 生成一张表 将集合/数组‘炸开’**  **横向视图 表生成函数 lateral view**  **例：创建一张表，用于存储列转行后的结果数据**  **create table t\_exp as select id,name,collect\_set(subject) as cs from t13 group by id,name;**  **生成一张表**  **select explode(cs) tmp from t\_exp;**  **行转列**  **select id,name,sub from (select \* from t\_exp lateral view explode(cs) tmp as sub ) t/tmp;**  **理解：**  **lateral view 相当于两表在join，该join只在同一行的数据之间进行**  **将explode生成的表字段与lateral view进行拼接，通过共同部分（标签）**  **例：hive完成wordcount案例：**  **Select word,count(1) as cnts from**  **(select explode(split(sentence,’’)) as word from t\_wc) tmp group by word order by cnts desc;**  **9：用户自定义函数**  **(1)udf(user defined function)： 自定义函数，特点是输入一行，输出一行（一对一）**  **(2)udaf(user defined aggregation function)：自定义聚合函数，特点是输入多行，输出一行（多对一）**  **(3)udtf(User-Defined Table-Generating Functions):自定义拆分函数，特点是输入一行，输出多行（一对多） (1)UDF开发和使用步骤 创建函数流程  a、自定义一个Java类  b、继承UDF类  c、重写evaluate方法  d、打成jar包  e、在hive执行add jar方法 (临时使用) f、在hive执行创建模板函数  g、hql中使用**  **(2)UDAF开发和使用步骤**  **a、顶层UDAF类继承 org.apache.hadoop.hive.ql.udf.generic.GenericUDAFEvaluator里面编写嵌套类evaluator实现UDAF的逻辑。**  **b、实现resolver :resolver负责类型检查，操作符重载resolver通常继承**  **org.apache.hadoop.hive.ql.udf.GenericUDAFResolver2，但是更建议继承AbstractGenericUDAFResolver，**  **隔离将来hive接口的变化。**  **GenericUDAFResolver和GenericUDAFResolver2接口的区别是后面的允许evaluator实现可以访问更多的信息，**  **例如DISTINCT限定符，通配符FUNCTION(\*)。**  **c、实现evaluator :evaluator真正实现UDAF的逻辑**  **所有evaluators必须继承抽象类 org.apache.hadoop.hive.ql.udf.generic.GenericUDAFEvaluator。予类必须实现它的一些抽象方法，实现UDAF的逻辑。**  **d、打成jar包**  **e、在hive执行add jar方法 (临时使用) f、在hive执行创建模板函数**  **g、hql中使用**  **(3)UDTF开发和使用 a、 继承org.apache.hadoop.hive.ql.udf.generic.GenericUDTF,实现initialize, process, close三个方法。**  **b、UDTF首先会调用initialize方法，此方法返回UDTF的返回行的信息（返回个数，类型）。**  **c、初始化完成后，会调用process方法,真正的处理过程在process函数中，在process中，每一次forward()调用产生一行；**  **如果产生多列可以将多个列的值放在一个数组中，然后将该数组传入到forward()函数。 最后close()方法调用，对需要清理的方法进行清理。**  **d、打成jar包  e、在hive执行add jar方法 (临时使用)**  **f、在hive执行创建模板函数**  **g、hql中使用**  **注意：UDTF有两种使用方法，一种直接放到select后面，一种和lateral view一起使用**  **10：窗口分析函数**  **11：left semi join on :左半连接**  ***Left semi join ：相当于join连接两个表后产生的数据中的左半部分，ON 子句中设置右表的过滤条件，在 WHERE 子句、SELECT 子句或其他地方过滤都不行。左半连接，select 和 where 子句不能引用到右边表字段。***  **hive中不支持exist/IN子查询，可以用left semi join来实现同样的效果：**  s**elect**  **a.name as aname,**  **a.numb as anumb**  **from t\_a a**  **left semi join t\_b b**  **on a.name=b.name;**  **12：sort by 局部排序**  **13：hive启动**  ***A:交互式shell启动***  bin/hive  ***B:hive服务使用***  **1:启动服务端:** bin/hiveserver2  *后台启动：nohup bin/hiveserver2 1>/dev/null 2>&1 &*  **2：启动客户端**  **启动成功后，可以在别的节点上用beeline去连接**   * **方式（1）**   **[root@hdp-02 hive-1.2.1]# bin/beeline 回车，进入beeline的命令界面**  **输入命令连接hiveserver2**  **beeline> !connect jdbc:hive2://hdp-02:10000**  **（hadoop01是hiveserver2所启动的那台主机名，端口默认是10000）**   * **方式（2）**   **启动时直接连接：**  **bin/beeline -u jdbc:hive2://c701:10000 -n root** |