# 小小Hive，拿下拿下

原创 PowerData-独倾 PowerData 2023-09-22 17:31 发表于江苏

收录于合集

#大数据组件 17 #数据仓库 2


---

PowerData

PowerData数据之力社区官方公众号

18篇原创内容

公众号


---

# 作者介绍

本文由PowerData独倾贡献
姓名：李洪荣
花名：独倾
微信：JunQingXiao49
年龄：00后
工作经验：大三学生
学习内容：数据中台、数据组件、数据仓库等
自我介绍：大数据萌新一枚，请多多指教。

全文共 16412 个字，建议阅读 58 分钟

## 1. 说下Hive是什么?跟数据库区别?

### 什么是Hive

Hive是基于Hadoop的一个数据仓库工具，可以将结构化的数据文件映射为一张表，并提供类SQL查询功能。本质是：将HQL转化成MapReduce程序。

### Hive和数据库的区别

#### 查询语言

由于SQL被广泛的应用在数据仓库中，因此，专门针对Hive的特性设计了类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。

#### 数据存储位置

Hive 是建立在 Hadoop 之上的，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。

### 数据更新

由于Hive是针对数据仓库应用而设计的，而数据仓库的内容是读多写少的。因此，Hive中不建议对数据的改写，所有数据都是在加载的时候确定好的。而数据库中的数据通常是需要经常进行修改的，因此可以使用INSERT INTO ..... VALUES添加数据，使用UPDATE.....SET修改数据。

### 索引

HIVE在加载数据的过程中不会对数据进行任何处理，甚至不会对数据进行扫描，因此也没有对数据中的某些key建立索引。Hive要访问数据中满足条件的特定值时，需要暴力扫描整个数据，因此访问延时较高。由于MapReduce的引入，Hive可以并行访问数据，因此即使没有索引，对于大数据的访问，Hive仍然可以体现出其优势。数据库中，通常会针对一个或几个列建立索引，因此对于少量的特定条件的数据的访问，数据库可以有很高的效率，较低的延迟。反观Hive，则不适合在线数据查询。

### 执行

Hive中大多数查询的执行是通过 Hadoop 提供的 MapReduce 来实现的。而数据库通常有自己的执行引擎。

### 执行延迟

Hive在查询数据的时候，由于没有索引，需要扫描整个表，因此延迟较高。另外一个导致Hive执行延迟高的因素是MapReduce框架。由于MapReduce本身具有较高的延迟，因此在利用MapReduce执行Hive查询时，也会有较高的延迟。数据库的执行延迟较低，更适合执行小规模的数据。

### 可扩展性

由于Hive是建立在Hadoop之上的，因此Hive的可扩展性是和Hadoop的可扩展性是一致的。而数据库由于 ACID 语义的严格限制，扩展行非常有限。目前最好的并行数据库 Oracle 在理论上的扩展能力也只有100台左右。

#### 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。

## 2. 说下为什么要使用Hive?Hive的优缺点?Hive的作用是什么?

### 为什么要使用Hive?

- Hive是Hadoop生态系统中比不可少的一个工具，它提供了一种SQL(结构化查询语言)方言，可以查询存储在Hadoop分布式文件系统（HDFS）中的数据或其他和Hadoop集成的文件系统，如MapR-FS、Amazon的S3和像HBase（Hadoop数据仓库）和Cassandra这样的数据库中的数据。
- 大多数数据仓库应用程序都是使用关系数据库进行实现的，并使用SQL作为查询语言。Hive降低了将这些应用程序转移到Hadoop系统上的难度。凡是会使用SQL语言的开发人员都可以很轻松的学习并使用Hive。如果没有Hive，那么这些用户就必须学习新的语言和工具，然后才能应用到生产环境中。另外，相比其他工具，Hive更便于开发人员将基于SQL的应用程序转移到Hadoop中。如果没有Hive，那么开发者将面临一个艰巨的挑战，如何将他们的SQL应用程序移植到Hadoop上。

### Hive优缺点

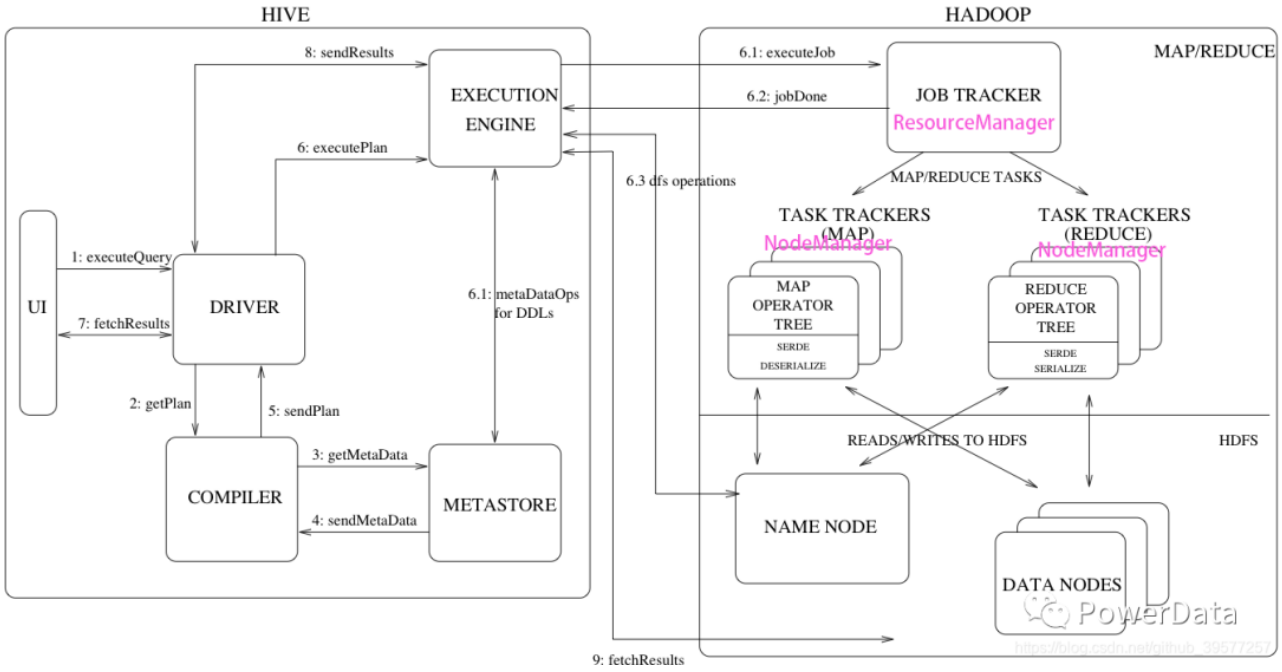
#### 优点

- 操作接口采用类SQL语法，提供快速开发的能力（简单、容易上手）。
- 避免了去写MapReduce，减少开发人员的学习成本。
- Hive的执行延迟比较高，因此Hive常用于数据分析，对实时性要求不高的场合。
- Hive优势在于处理大数据，对于处理小数据没有优势，因为Hive的执行延迟比较高。
- Hive支持用户自定义函数，用户可以根据自己的需求来实现自己的函数。

### 缺点

- Hive的HQL表达能力有限
- 迭代式算法无法表达
- 数据挖掘方面不擅长，由于MapReduce数据处理流程的限制，效率更高的算法却无法实现。
- Hive的效率比较低
- Hive自动生成的MapReduce作业，通常情况下不够智能化
- Hive调优比较困难，粒度较粗
- Hive不是一个完整的数据库。Hadoop以及HDFS的设计本身约束和局限性地限制了Hive所能胜任的工作。其中最大的限制就是Hive不支持记录级别的更新、插入或者删除操作。但是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。同时，因为Hadoop是面向批处理的系统，而MapReduce任务（job）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。传统数据库中在秒级别可以完成的查询，在Hive中，即使数据集相对较小，往往也需要执行更长的时间。##Hive的作用
- Hive是由Facebook开源用于解决海量结构化日志的数据统计工具。
- Hive是基于Hadoop的一个数据仓库工具，可以将结构化的数据文件映射为一张表，并提供类SQL查询功能。
- Hive的本质是将HQL转化成MapReduce程序
- Hive处理的数据存储在HDFS
- Hive分析数据底层的实现是MapReduce
- 执行程序运行在Yarn上

# 3. Hive架构




如图所示，Hive的主要组件是：

- UI ：用户向系统提交查询和其他操作的用户界面。
- Driver ：接收查询的组件（Component）。这个组件实现了session句柄的概念，并提供了在JDBC/ODBC接口上执行和获取模型化的API。
- Compiler ：解析查询的组件，对不同的查询块（query blocks）和查询表达式(query expressions)进行语义分析，最终在表(元数据表)的帮助下生成执行计划，并从Metastore中查找分区元数据。
- MetaStore ：存储仓库中各种表和分区的所有结构化信息的组件，包括列和列类型信息，读取和写入数据所需的序列化程序和反序列化程序，以及存储数据的相应HDFS文件。
- Execution Engine ：执行编译器创建的执行计划（execution plan）的组件。该计划是一个stage的DAG。执行引擎管理计划的这些不同stage之间的依赖关系，并在适当的系统组件上执行这些stage。
- 上图中显示了典型的查询如何在系统中流动（flows）。

    1. UI调用Driver的执行接口（step 1）。

2. Driver为查询创建会话句柄(session handle)，并将查询发送到Compiler以生成执行计划（step 2）。
3. Compiler从Metastore获取必要的元数据（step 3 and 4），此元数据用于查询树中的表达式进行类型检查以及基于查询谓词修剪分区（prune partitions）。
4. 由Compiler生成的计划（step 5）是一个stage的DAG，每个stage 是一个map/reduce job、元数据操作或一个 HDFS上的操作，对于map/reduce阶段，计划包含map运算树（在mappers上执行的运算树）和reduce运算树（用于需要reducers的操作）。
5. 执行引擎(Execution engine)将这些stage提交给适当的组件（step 6，6.1，6.2和6.3），在每个Task（mapper/reducer）中与表或中间输出相关联的反序列化用于从HDFS文件中读取行，这些行通过关联的运算树传递。
6. 生成输出后，它将通过序列化程序写入临时HDFS文件（如果操作不需要reduce，则会在mapper中发生），临时文件用于向计划的后续map/reduce阶段提供数据。对于DML(Data Manipulation Language)操作，最终临时文件将移动到表的位置，此方案用于确保不读取脏数据（文件重命名是HDFS中的原子操作）。对于查询，执行引擎直接作为来自Driver的提取调用的一部分从HDFS读取临时文件内容（step 7,8和9）。

## 4. Hive内部表和外部表的区别?

### 数据存储位置

- 内部表：内部表的数据存储在Hive数据仓库目录中，通常位于HDFS(Hadoop Distributed File System)上的/user/hive/warehouse目录下，由Hive完全管理。当创建内部表时，Hive会在指定的数据仓库目录中创建相应的数据存储目录，并将表的数据直接存放在这些目录中。
- 外部表：外部表的数据存储在用户指定的位置，可以是HDFS上的任意路径，也可以是本地文件系统或其他支持的存储系统。Hive仅在元数据中维护外部表的结构信息，不对

数据的存储位置和文件管理负责。

### 数据管理方式

- 内部表：由于Hive完全管理内部表的数据，它会在表被删除时，同时删除表对应的数据。这意味着删除内部表将导致表数据的彻底丢失。内部表适用于需要完全由Hive管理和控制的数据。
- 外部表：外部表的数据由用户自行管理，Hive仅维护元数据。如果删除外部表，只会删除元数据而不会影响存储在外部表位置的数据。这种特性使得外部表适用于对数据有更细粒度控制，希望在删除表时保留数据的情况。

### 数据的持久性

- 内部表：内部表的数据在被加载到表中后会持久保存，并且只有在显式删除表时才会被删除。在重启Hive或重新加载元数据后，内部表的数据会保留。
- 外部表：外部表的数据在加载到表中后并不一定被持久保存，因为外部表的数据是由用户管理的。如果数据源是临时性的，那么在会话结束或Hive重启后，外部表的数据可能会丢失。

### 数据的导入

- 内部表：可以使用INSERT语句向内部表中插入数据，Hive会将数据存储在内部表的数据目录中。
- 外部表：数据可以通过多种方式加载到外部表中，例如通过LOAD DATA语句从本地文件系统或其他数据源加载数据。在加载数据时，只是将数据的元数据信息添加到外部表中，实际数据保留在外部表的位置。

### **ALTER**操作

- 内部表：对于内部表，可以使用ALTER TABLE语句更改表的属性，例如更改列名、添加/删除分区等。

- 外部表：对于外部表，ALTER TABLE语句仅允许更改表的一些元数据信息，例如重命名表、更改列的注释等，但不能更改表的存储位置或数据本身。

总的来说，内部表适用于需要由Hive完全管理和控制数据的情况，而外部表适用于希望自行管理数据的情况，保留数据并在删除表时不影响数据的情况。选择内部表还是外部表取决于具体的数据管理需求和数据的生命周期。

## 5. 为什么内部表的删除，就会将数据全部删除，而外部表只删除表结构?为什么用外部表更好?

- 创建表时：创建内部表时，会将数据移动到数据仓库指向的路径；若创建外部表，仅记录数据所在的路径，不对数据的位置做任何改变。
- 删除表时：在删除表的时候，内部表的元数据和数据会被一起删除， 而外部表只删除元数据，不删除数据。这样外部表相对来说更加安全些，数据组织也更加灵活，方便共享源数据。###外部表的优点
- 外部表不会加载数据到Hive的默认仓库（挂载数据），减少了数据的传输，同时还能和其他外部表 共享数据。
- 使用外部表，Hive不会修改源数据，不用担心数据损坏或丢失。
- Hive在删除外部表时，删除的只是表结构，而不会删除数据。

## 6. Hive建表语句?创建表时使用什么分隔符?

- 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]
[TBLPROPERTIES (property_name=property_value, …)]
[AS select_statement]
```

## 字段解释说明

1. CREATE TABLE 创建一个指定名字的表。如果相同名字的表已经存在，则抛出异常；用户可以用
2. IF NOT EXISTS 选项来忽略这个异常。
3. EXTERNAL 关键字可以让用户创建一个外部表，在建表的同时可以指定一个指向实 际数据的路径
4. （LOCATION），在删除表的时候，内部表的元数据和数据会被一起删除，而外 部表只删除元数据，不删除数据。
5. COMMENT：为表和列添加注释。
6. PARTITIONED BY：创建分区表
7. CLUSTERED BY：创建分桶表
8. SORTED BY：不常用，对桶中的一个或多个列另外排序

- 用户在建表的时候可以自定义SerDe或者使用自带的SerDe。如果没有指定ROW FORMAT或者ROW FORMAT DELIMITED，将会使用自带的SerDe。在建表的时候，用户还需要为表指定列，用户在指定表的列的同时也会指定自定义的SerDe，Hive通过SerDe确定表的具体的列的数据。SerDe是Serialize/Deserilize的简称，hive使用Serde进行行对象的序列与反序列化。
- STORED AS：指定存储文件类型 常用的存储文件类型：SEQUENCEFILE（二进制序列文件）、TEXTFILE（文本）、RCFILE（列式存储格式文件）。
- 如果文件数据是纯文本，可以使用 STORED AS TEXTFILE。如果数据需要压缩，使用 STORED AS SEQUENCEFILE。

1. LOCATION：指定表在HDFS上的存储位置。
2. AS：后跟查询语句，根据查询结果创建表。
3. LIKE：允许用户复制现有的表结构，但是不复制数据。

## 分隔符

- 我们在建表的时候就指定了导入数据时的分隔符，建表的时候会有三种场景需要考虑：正常建表(default)；指定特定的特殊符号作为分隔符；使用多字符作为分隔符。
- 正常建表，采用默认的分隔符
- hive 默认的字段分隔符为ascii码的控制符\001,建表的时候用fields terminated by '\001',如果要测试的话， 造数据在vi 打开文件里面，用ctrl+v然后再ctrl+a可以输入这个控制符\001。按顺序，\002的输入方式为ctrl+v，ctrl+b。以此类推。
- 指定特定的特殊符号作为分隔符

```
CREATE TABLE test(id int, name string ,tel string) ROW FORMAT DELIMITED FIELDS TERMINATED BY
```

- 上面使用了'\t'作为了字段分隔符，'\n'作为换行分隔符。如果有特殊需求，自己动手改一下这两个符号 就行了。
- 使用多字符作为分隔符
- 假设我们使用【##】来作为字段分隔符，【\n】作为换行分隔符，则这里有两个方法：使用MultiDelimitSerDe的方法来实现：

```
CREATE TABLE test(id int, name string ,tel string) ROW FORMAT SERDE 'org.apache.hadoop.hive.
```

- 使用RegexSerDe的方法实现：

```
CREAET TABLE test(id int, name string ,tel string) ROW FORMAT SERDE 'org.apache.hadoop.hive.
```

## 7. Hive删除语句外部表删除的是什么?

- 外部表只删除元数据，不删除数据

## 8. Hive数据倾斜以及解决方案

### 什么是数据倾斜

- 数据倾斜主要表现在，mapreduce程序执行时，reduce节点大部分执行完毕，但是有一个或者几个reduce节点运行很慢，导致整个程序的处理时间很长，这是因为某一个key的条数比其他key多很多(有时是百倍或者千倍之多)，这条Key所在的reduce节点所处理的数据量比其他节点就大很多，从而导致某几 个节点迟迟运行不完。

### 数据倾斜的原因一些操作

- 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。
- 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，非常慢。
- group by 维度过小，某值的数量过多处理某值的reduce非常耗时。
- 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

### 现象

- 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。

### 数据倾斜的解决方案

- 参数调节

```
hive.map.aggr = true
```

Map 端部分聚合，相当于Combiner

```
hive.groupby.skewindata=true
```

- 有数据倾斜的时候进行负载均衡，当选项设定为true，生成的查询计划会有两个MR Job。第一个MR Job 中，Map 的输出结果集合会随机分布到Reduce中，每个Reduce做部分聚合操作，并输出结果，这样处理的结果是相同的Group By Key有可能被分发到不同的Reduce中，从而达到负载均衡的目的；第二个MR Job再根据预处理的数据结果按照Group By Key分布到Reduce中（这个过程可以保证相同的Group By Key 被分布到同一个Reduce中），最后完成最终的聚合操作。

### SQL语句调节

- 如何join：关于驱动表的选取，选用join key分布最均匀的表作为驱动表，做好列裁剪和filter操作，以达到两表做join的时候，数据量相对变小的效果。

- 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。
- count distinct大量相同特殊值：count distinct时，将值为空的情况单独处理，如果是计算count distinct，可以不用处理，直接过滤，在最后结果中加1。如果还有其他计算，需要进行group by，可以先将值为空的记录单独处理，再和其他计算结果进行union。
- 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。

## 典型的业务场景

### 空值产生的数据倾斜

- 场景：如日志中，常会有信息丢失的问题，比如日志中的 user_id，如果取其中的 user_id 和 用户表中的user_id 关联，会碰到数据倾斜的问题。

```
解决方法一：user_id为空的不参与关联
select * from log a
join users b
on a.user_id is not null
and a.user_id = b.user_id
union all
select * from log a
where a.user_id is null;
解决方法二：赋与空值分新的key值
select *
from log a
left outer join users b
on case when a.user_id is null then concat('hive',rand() ) else a.user_id end = b.user_id;
```

- 结论：方法2比方法1效率更好，不但io少了，而且作业数也少了。解决方法一中log读取两次，jobs是2。解决 方法二job数是1。这个优化适合无效id (比如 -99 , '', null 等) 产生

的倾斜问题。把空值的key变成一个字符串加上随机数，就能把倾斜的数据分到不同的reduce上，解决数据倾斜问题。

### 不同数据类型关联产生数据倾斜

- 场景：用户表中user_id字段为int，log表中user_id字段既有string类型也有int类型。当按照user_id进行两 个表的Join操作时，默认的Hash操作会按int型的id来进行分配，这样会导致所有string类型id的记录都分 配到一个Reducer中。

```
解决方法：把数字类型转换成字符串类型
select * from users a
left outer join logs b
on a.usr_id = cast(b.user_id as string);
```

### 小表不小不大，怎么用 **map join** 解决倾斜问题

- 使用map join解决小表（记录数少）关联大表的数据倾斜问题，这个方法使用的频率非常高，但如果小表很大，大到map join会出现bug或异常，这时就需要特别的处理。例如：

```
select * from log a
left outer join users b
on a.user_id = b.user_id;
```

- users表有600w+的记录，把users分发到所有的map上也是个不小的开销，而且map join不支持这么大的小表。如果用普通的join，又会碰到数据倾斜的问题。

```
解决方法：
select /+mapjoin(x)/* from log a
left outer join (
select /+mapjoin©/d.*
from ( select distinct user_id from log ) c
join users d
on c.user_id = d.user_id
) x
on a.user_id = b.user_id;
```

## 9. Hive如果不用参数调优，在map和reduce端应该做什么

### map阶段优化

- Map阶段的优化，主要是确定合适的map数。那么首先要了解map数的计算公式

```
num_reduce_tasks = min[${hive.exec.reducers.max},
(${input.size}/${hive.exec.reducers.bytes.per.reducer})]
```

- mapred.min.split.size: 指的是数据的最小分割单元大小；min的默认值是1B
- mapred.max.split.size: 指的是数据的最大分割单元大小；max的默认值是256MB
- dfs.block.size: 指的是HDFS设置的数据块大小。个已经指定好的值，而且这个参数默认情况下hive是识别不到的。
- 通过调整max可以起到调整map数的作用，减小max可以增加map数，增大max可以减少map数。需要提醒的是，直接调整mapred.map.tasks这个参数是没有效果的。

### reduce阶段优化

- reduce阶段，是指前面流程图中的reduce phase（实际的reduce计算）而非图中整个reduce task。Reduce阶段优化的主要工作也是选择合适的reduce task数量, 与map优化

不同的是，reduce优化时，可以直接设置mapred.reduce.tasks参数从而直接指定reduce的个数。

```
num_reduce_tasks = min[${hive.exec.reducers.max},
(${input.size}/${hive.exec.reducers.bytes.per.reducer})]
```

- hive.exec.reducers.max ：此参数从Hive 0.2.0开始引入。在Hive 0.14.0版本之前默认值是999；而从Hive 0.14.0开始，默认值变成了1009，这个参数的含义是最多启动的Reduce个数
- hive.exec.reducers.bytes.per.reducer ：此参数从Hive 0.2.0开始引入。在Hive 0.14.0版本之前默认值是1G(1,000,000,000)；而从Hive 0.14.0开始，默认值变成了256M(256,000,000)，可以参见HIVE-7158和HIVE-7917。这个参数的含义是每个Reduce处理的字节数。比如输入文件的大小是1GB，那么会启动4个Reduce来处理数据。
- 也就是说，根据输入的数据量大小来决定Reduce的个数，默认Hive.exec.Reducers.bytes.per.Reducer为1G，而且Reduce个数不能超过一个上限参数值，这个参数的默认取值为999。所以我们可以调整
- Hive.exec.Reducers.bytes.per.Reducer来设置Reduce个数。
- 注意：Reduce的个数对整个作业的运行性能有很大影响。如果Reduce设置的过大，那么将会产生很多小文件，对NameNode会产生一定的影响，而且整个作业的运行时间未必会减少；如果Reduce设置的过小，那么单个Reduce处理的数据将会加大，很可能会引起OOM异常。
- 如果设置了 mapred.reduce.tasks/mapreduce.job.reduces 参数，那么Hive会直接使用它的值作为Reduce的个数；
- 如果mapred.reduce.tasks/mapreduce.job.reduces的值没有设置（也就是-1），那么Hive会根据输入文件的大小估算出Reduce的个数。根据输入文件估算Reduce的个数可能未必很准确，因为Reduce的输入是Map的输出，而Map的输出可能会比输入要小，所以最准确的数根据Map的输出估算Reduce的个数。

# 10. Hive的用户自定义函数实现步骤与流程

## 如何构建UDF？

### 用户创建的UDF使用过程

1. 继承UDF或者UDAF或者UDTF，实现特定的方法；
2. 将写好的类打包为jar，如hivefirst.jar；
3. 进入到Hive外壳环境中，利用add jar /home/hadoop/hivefirst.jar注册该jar文件；
4. 为该类起一个别名，create temporary function mylength as 'com.whut.StringLength'，这里注意UDF只是为这个Hive会话临时定义的；
5. 在select中使用mylength()。

### 函数自定义实现步骤

1. 继承Hive提供的类

```
org.apache.hadoop.hive.ql.udf.generic.GenericUDF
org.apache.hadoop.hive.ql.udf.generic.GenericUDTF
```

2. 实现类中的抽象方法
3. 在 hive 的命令行窗口创建函数添加 jar

```
add jar linux_jar_path
```

4. 创建 function

```
create [temporary] function [dbname.]function_name AS class_name;
```

5. 在 hive 的命令行窗口删除函数

```
drop [temporary] function [if exists] [dbname.]function_name;
```

## 自定义UDF案例

- 需求:自定义一个UDF实现计算给定字符串的长度，例如：

```
hive(default)> select my_len("abcd");
4
```

1. 导入依赖

```
<dependencies>
 <dependency>
  <groupId>org.apache.hive</groupId>
  <artifactId>hive-exec</artifactId>
  <version>3.1.2</version>
 </dependency>
</dependencies>
```

2. 创建一个类，继承于Hive自带的UDF

```
 * 自定义 UDF 函数，需要继承 GenericUDF 类
 * 需求: 计算指定字符串的长度
 */
public class MyStringLength extends GenericUDF {
 /**
 *
 *@param arguments 输入参数类型的鉴别器对象
 * @return 返回值类型的鉴别器对象
 *@throws UDFArgumentException
 */
 @Override
 public ObjectInspector initialize(ObjectInspector[] arguments) throws UDFArgumentExcep
  // 判断输入参数的个数
```

```
        if(arguments.length !=1) {
         throw new UDFArgumentLengthException("Input Args Length Error!!!");
        }
      // 判断输入参数的类型

        if(!arguments[0].getCategory().equals(ObjectInspector.Category.PRIMITIVE)
          ) {
         throw new UDFArgumentTypeException(0,"Input Args Type Error!!!");
        }
      //函数本身返回值为 int，需要返回 int 类型的鉴别器对象
        return PrimitiveObjectInspectorFactory.javaIntObjectInspector;
      }

      /**
      * 函数的逻辑处理
      *@param arguments 输入的参数
      *@return 返回值
      *@throws HiveException
      */
      @Override
      public Object evaluate(DeferredObject[] arguments) throws HiveException {
        if(arguments[0].get() == null) {
          return 0;
        }
        return arguments[0].get().toString().length();
      }

      @Override
      public String getDisplayString(String[] children) {
        return "";
      }
    }
```

3. 打成jar包上传到服务器

```
/opt/module/data/myudf.jar
```

4. 将jar包添加到hive的classpath

```
hive (default)> add jar /opt/module/data/myudf.jar;
```

5. 创建临时函数与开发好的java class关联

6. 即可在hql中使用自定义的函数

```
hive (default)> select ename,my_len(ename) ename_len from emp;
```

## 11. Hive的三种自定义函数是什么?它们之间的区别?

- UDF：用户自定义函数，user defined function。单行进入，单行输出。UDF 操作作用于单个数据行，并且产生一个数据行作为输出。大多数函数都属于这一类（比如数学函数和字符串函数）。
- UDTF：用户自定义表生成函数。user defined table-generate function，单行输入，多行输出。UDTF 操作作用于单个数据行，并且产生多个数据行，一个表作为输出。
- UDAF：用户自定义聚合函数。user defined aggregate function，多行进入，单行输出。UDAF 接受多个输入数据行，并产生一个输出数据行。像COUNT和MAX这样的函数就是聚集函数。

## 12. Hive的cluster by、sort bydistribute by、orderby区别?

- order by：全局排序，一个reducer；
- sort by：分区内排序；
- distribute by：控制map结果的分发，相同值会被分发到同一个map；
- cluster by：当distribute by和sort by用的同一个字段，可以用这个关键字，不能指定排序顺序。

## 13. Hive分区和分桶的区别

- 分区针对的是数据的存储路径；分桶针对的是数据文件。
- 分区可以提高查询效率，实际上 hive 的一个分区就是 HDFS 上的一个目录，目录里放着属于该分区的数据文件。
- 分区提高了数据的查询效率，同时还能将数据隔离开，但是并非所有数据能形成合理的分区。hive可以将数据进行分桶，不同于分区是针对存储路径进行分类，分桶是在数据文件中对数据进行划分的一种技术。分桶是指定某一列，让该列数据按照哈希取模的方式随机、均匀地分发到各个桶文件中。

## 14. Hive的执行流程

1. (执行查询操作)Execute Query:命令行或Web UI之类的Hive接口将查询发送给Driver(任何数据库驱动程序，如JDBC、ODBC等)以执行。
2. (获取计划任务)Get Plan:Driver:借助查询编译器解析查询，检查语法和查询计划或查询需求
3. (获取元数据信息)Get Metadata:编译器将元数据请求发送到Metastore(任何数据库)。
4. (发送元数据)Send Metadata:Metastore将元数据作为对编译器的响应发送出去。
5. (发送计划任务)Send Plan:编译器检查需求并将计划重新发送给Driver。到目前为止，查询的解析和编译已经完成
6. (执行计划任务)Execute Plan:Driver将执行计划发送到执行引擎。
7. (执行Job任务)Execute Job:在内部，执行任务的过程是MapReduce Job。执行引擎将Job发送到ResourceManager,ResourceManager位于Name节点中，并将job分配给datanode中的NodeManager。在这里，查询执行MapReduce任务。
8. (元数据操作)Metadata Ops:在执行的同时，执行引擎可以使用Metastore执行元数据操作。
9. (拉取结果集)Fetch Result:执行引擎将从datanode上获取结果集。
10. (发送结果集至driver)Send Results:执行引擎将这些结果值发送给Driver。

11. (driver将result发送至interface)Send Results:Driver将结果发送到Hive接口。

## 15. Hive SQL转化为MR的过程?

1. 进入程序，利用Antlr框架定义HQL的语法规则，对HQL完成词法语法解析，将HQL转换为为AST（抽象语法树）；
2. 遍历AST，抽象出查询的基本组成单元QueryBlock（查询块），可以理解为最小的查询执行单元；
3. 遍历QueryBlock，将其转换为OperatorTree（操作树，也就是逻辑执行计划），可以理解为不可拆分的一个逻辑执行单元；
4. 使用逻辑优化器对OperatorTree（操作树）进行逻辑优化。例如合并不必要的ReduceSinkOperator，减少Shuffle数据量；
5. 遍历OperatorTree，转换为TaskTree。也就是翻译为MR任务的流程，将逻辑执行计划转换为物理执行计划；
6. 使用物理优化器对TaskTree进行物理优化；
7. 生成最终的执行计划，提交任务到Hadoop集群运行。

## 16. Hive的存储引擎和计算引擎

存储引擎：

- HDFS（Hadoop Distributed File System）：Hive 默认使用 HDFS 作为数据存储引擎。HDFS 是 Hadoop 提供的分布式文件系统，能够在大规模集群上存储和处理数据。
- Apache HBase：HBase 是一个分布式的、面向列的 NoSQL 数据库，Hive 可以通过 HBase 存储引擎来访问和查询 HBase 中的数据。

- Amazon S3：Hive 可以通过 Amazon S3 存储引擎来访问和查询存储在 Amazon S3 上的数据。

计算引擎：

- MapReduce：Hive 默认使用 MapReduce 作为计算引擎。MapReduce 是 Hadoop 提供的一种分布式计算框架，可以并行处理大规模数据集。
- Tez：Tez 是一个基于 Hadoop 的高性能数据处理框架，它可以替代 MapReduce 作为 Hive 的计算引擎，提供更快的查询执行速度。
- Spark：Hive 可以通过 Spark 计算引擎来执行查询。Spark 是一个快速的、通用的大数据处理引擎，可以在内存中进行数据处理，提供更高的性能。

## 17. Hive的文件存储格式都有哪些

文件存储格式有：**TextFile**、**SequenceFile**、**RCFile**、**ORCFile**。

- TextFile：默认格式，存储方式为行存储，数据不做压缩，磁盘开销大，数据解析开销大，可以结合GZIP和BZIP使用，但是使用这种方式，压缩的文件不支持split，Hive不对数据进行切分，从而无法对数据进行并行操作。
- SequenceFile：这是由Hadoop API提供的一种二进制文件支持，其具有使用方便，可分割，可压缩的特点，它支持三种压缩格式：None，Record，Block。Record压缩率低，一般使用Block。
- RCFile：按行分块，每块按列存储。RCFile不支持任意方式的数据写入，仅提供了一种追加接口。
- ORCFile：按行分块，每块按列存储，压缩快，快速列存取。

# 18. Hive中如何调整Mapper和Reducer的数目

- 当map或reduce的计算很复杂、单个map的执行时间很长，且hive分配的map数或reduce比较少，集群还有大量计算资源没有利用的情况时，可以通过增大map数或reduce数，来提高任务并发，缩短任务计算时长，提高计算效率。

## hive on mr

### 如何调整**map**数

- InputFormat 接口按照某个策略将输入数据且分成若干个 split，以便确定 Map Task 的个数即 Mapper 的个数，在 MapReduce 框架中，一个 split 就意味着需要一个 Map Task。
- 当hive.input.format=org.apache.hadoop.hive.ql.io.HiveInputFormat时，hive会先计算splitSize ，然后通过splitSize、任务输入的数据量大小和文件数来共同决定split数量，即map数。

```
splitSize = max{minSize,min{goalSize,blockSize}}
```

- minSize：是mapreduce.input.fileinputformat.split.minsize决定的 InputFormat的最小长度。
- goalSize：该值由 totalSize/numSplits来确定 InputSplit 的长度，它是根据用户的期望的 InputSplit 个数计算出来的。numSplits 为用户设定的 Map Task 的个数，默认为1，可通过mapreduce.job.maps设置。totalSize是hive任务处理的数据量大小。
- blockSize：HDFS 中的文件存储块block的大小，可以通过dfs.blocksize查看大小。
- 由上公式可知，在org.apache.hadoop.hive.ql.io.HiveInputFormat接口下，主要是mapreduce.input.fileinputformat.split.minsize和mapreduce.job.maps来决定map数。

- 当hive.input.format=org.apache.hadoop.hive.ql.io.CombineHiveInputFormat时，主要是如下四个参数起作用：

```
mapreduce.input.fileinputformat.split.minsize
mapreduce.input.fileinputformat.split.maxsize
mapreduce.input.fileinputformat.split.minsize.per.rack
mapreduce.input.fileinputformat.split.minsize.per.node
```

- 这里切分的逻辑比较复杂，主要的流程大致如下：

  1. 首先处理每个Datanode的blockInfo，先按照>=maxsplitsize来切分split，剩余的再按照blockinfo>=minSplitSizeNode切分，其余的等和rack的其余blockinfo进行合并。
  2. 其次对每个Rack进行处理：先按照>=maxsplitsize来切分split，剩余的再按照blockinfo>=minSplitSizeRack切分，其余的等和overflow的其余blockinfo进行合并。
  3. 对于overflow blockInfo直接根据maxsplitsize来进行切分。

### 如何调整**reduce**数

- hive on mr模式下reduce数主要受如下两个参数影响：

```
hive.exec.reducers.bytes.per.reducer --每个reduce处理的数量量
hive.exec.reducers.max  --hive任务最大reduce个数
reducer数 = min(hive.exec.reducers.max,max(1,totalsize/hive.exec.reducers.bytes.per.reducer)
```

## hive on tez

### 如何调整**map**数

- 在运行hive on tez时会发现跟hive on mr的map数差异会比较大，主要原因在于 Tez 中对 inputSplit 做了 grouping 操作，将多个 inputSplit 组合成更少的 groups，然后为每个 group 生成一个 mapper 任务，而不是为每个inputSplit 生成一个mapper 任务。可以通过调整如下参数来调整grouping数：

```
tez.grouping.min-size
tez.grouping.max-size
```

如何调整**reduce**数

- tez on tez模式下reduce数主要受如下两个参数影响：

```
hive.exec.reducers.bytes.per.reducer --每个reduce处理的数量量
hive.exec.reducers.max  --hive任务最大reduce个数
hive.tez.auto.reducer.parallelism
hive.tex.min.partition.factor
hive.tez.max.partition.factor
reducer数 = Max(1, Min(hive.exec.reducers.max, ReducerStage estimate/hive.exec.reducers.bytes
```

## 19. 介绍下知道的Hive窗口函数，举一些例子

- 窗口排序函数：ROW_NUMBER()、RANK()、DENSE_RANK()；
  - ROW_NUMBER()函数，生成的排序序号从1开始，不存在相同序号。
  - RANK()函数，生成的排序序号从1开始，序号相同有空位。
  - DENSE_RANK()函数，生成的排序序号从1开始，序号相同不会留下空位。
- 窗口聚合函数：SUM()、MIN()、MAX()、AVG()；
  - 搭配窗口子句使用， 如：select 字段1，字段2, ..., sum(字段3) over( partition by 字段1 order by 字段2 rows between unbounded preceding and current row) as 新字段1
  - 新字段1含义为计算当前分组中，从第一行到当前行字段3的和。其中ROWS BETWEEN叫窗口子句， 其中CURRENT ROW表示当前行，UNBOUNDED PRECEDING表示前面的起点，UNBOUNDED FOLLOWING表示后面的终点，当没有写窗口子句时，语义为从第一行到当前行。

- LAG():LAG(col，n，DEFAULT) 用于取窗口内列col往前第N行的值，列名，往前取n行（可选，默认为1），第三个参数为默认值NULL

- LEAD():LEAD(col, n, DEFAULT) 用于去窗口内列col往后第n行的值，列名，往后取n行（可选，默认为1），第三个参数当列col往后取n行为NULL时，取该默认值，若不指定，则为NULL。

## 20. Hive的count的用法

```
count（*）：所有行进行统计，包括NULL行

count（1）：所有行进行统计，包括NULL行

count（column）：对column中非NULL进行统计

count（distinct column）：对column中非NULL进行去重统计

count（distinct col1,col2,…）：对col1、col2,…多个字段同时去重并统计。

count(CASE WHEN plat=1 THEN u ELSE NULL END),

count(DISTINCT CASE WHEN plat=1 THEN u ELSE NULL END),

count(CASE WHEN (type=2 OR type=6) THEN u ELSE NULL END),

count(DISTINCT CASE WHEN (type=2 OR type=6) THEN u ELSE NULL END)
```

## 21. Hive的union和unionall的区别

- union 会将联合的结果集去重
- union all 不会对结果集去重

## 22.

## Hive的join操作原理，leftjoin、right join、inner join、outer join的异同?

- INNER JOIN（内连接）：INNER JOIN 返回两个表中满足连接条件的交集。只有在连接条件成立的情况下，才会返回匹配的行。结果集中的行数取决于两个表中满足连接条件的交集。
- LEFT JOIN（左连接）：LEFT JOIN 返回左表中的所有行，以及右表中满足连接条件的匹配行。如果右表中没有匹配的行，则返回 NULL 值。结果集中的行数取决于左表的行数。
- RIGHT JOIN（右连接）：RIGHT JOIN 返回右表中的所有行，以及左表中满足连接条件的匹配行。如果左表中没有匹配的行，则返回 NULL 值。结果集中的行数取决于右表的行数。
- FULL OUTER JOIN（全外连接）：FULL OUTER JOIN 返回左表和右表中的所有行，无论是否满足连接条件。如果某个表中没有匹配的行，则返回 NULL 值。结果集中的行数取决于右表的行数。

## 23. Hive如何优化join操作

### Common join

如果不指定MapJoin或者不符合MapJoin的条件，那么Hive解析器会将Join操作转换成Common Join,即：在Reduce阶段完成join。整个过程包含Map、Shuffle、Reduce阶段。

- Map阶段

  - 读取源表的数据，Map输出时候以Join on条件中的列为key，如果Join有多个关联键，则以这些关联键的组合作为key;
  - Map输出的value为join之后所关心的(select或者where中需要用到的)列；同时在value中还会包含表的Tag信息，用于标明此value对应哪个表；
  - 按照key进行排序
- Shuffle阶段
  - 根据key的值进行hash,并将key/value按照hash值推送至不同的reduce中，这样确保两个表中相同的key位于同一个reduce中
- Reduce阶段
  - 根据key的值完成join操作，期间通过Tag来识别不同表中的数据。

## mapjoin（用于优化小表与大表join）

- mapJoin的意思就是，当链接的两个表是一个比较小的表和一个特别大的表的时候，我们把比较小的table直接放到内存中去，然后再对比较大的表格进行map操作(执行mr过程的map操作)。join就发生在map操作的时候，每当扫描一个大的table中的数据，就要去去查看小表的数据，哪条与之相符，继而进行连接。这里的join会在map阶段完成，仅仅是在内存就进行了两个表的join，并不会涉及reduce操作。map端join的优势就是在于没有shuffle，从而提高效率。在实际的应用中，
- hive自动识别

```
--默认值为true，自动开户MAPJOIN优化
set hive.auto.convert.join=true;
--默认值为2500000(25M),通过配置该属性来确定使用该优化的表的大小，如果表的大小小于此值就会被加载进内存
set hive.mapjoin.smalltable.filesize=2500000;
```

这样设置，hive就会自动的识别比较小的表，继而用mapJoin来实现两个表的联合。

- 自己指定

```
select /*+ mapjoin(a) */      --指定将a表放入内存中
        a.*,
        b.*
from table1 a join table12 b
```

### Bucket-MapJoin（用于优化大表与大表join）

- 两个表join的时候，小表不足以放到内存中，但是又想用map side join这个时候就要用到bucket Map join。其方法是两个join表在join key上都做hash bucket，并且把你打算复制的那个（相对）小表的bucket数设置为大表的倍数。这样数据就会按照key join，做hash bucket。小表依然复制到所有节点，Map join的时候，小表的每一组bucket加载成hashtable，与对应的一个大表bucket做局部join，这样每次只需要加载部分hashtable就可以了。
- 前提条件

```
set hive.optimize.bucketmapjoin = true;
--一个表的bucket数是另一个表bucket数的整数倍
--列 == join列
--必须是应用在map join的场景中
--注意：如果表不是bucket的，则只是做普通join。
```

### SMB Join（用于优化大表与大表join）

- smb是sort merge bucket操作是Bucket-MapJoin的优化，首先进行排序，继而合并，然后放到所对应的bucket中去，bucket是hive中和分区表类似的技术，就是按照key进行hash，相同的hash值都放到相同的buck中去。在进行两个表联合的时候。我们首先进行分桶，在join会大幅度的对性能进行优化。也就是说，在进行联合的时候，是table1中的一小部分和table1中的一小部分进行联合，table联合都是等值连接，相同的key都放到了同一个bucket中去了，那么在联合的时候就会大幅度的减小无关项的扫描。
- 进行相应的设置

```
--写入数据强制分桶
set hive.enforce.bucketing=true;
```

```
--写入数据强制排序
set hive.enforce.sorting=true;
--开启bucketmapjoin
set hive.optimize.bucketmapjoin = true;
--开启SMB Join
set hive.auto.convert.sortmerge.join=true;
set hive.auto.convert.sortmerge.join.noconditionaltask=true;
--小表的bucket数=大表bucket数
--Bucket 列 == Join 列 == sort 列
--必须是应用在bucket mapjoin 的场景中
```

### LEFT SEMI JOIN(代替 in)

- LEFT SEMI JOIN 本质上就是 IN/EXISTS 子查询的表现，是IN的一种优化。
- LEFT SEMI JOIN 的限制是， JOIN 子句中右边的表只能在 ON 子句中设置过滤条件，在 WHERE 子句、SELECT 子句或其他地方都不行。
- 因为 left semi join 是 in(keySet) 的关系，遇到右表重复记录，左表会跳过，而 join 则会一直遍历。这就导致右表有重复值得情况下 left semi join 只产生一条，join 会产生多条。
- left semi join 是只传递表的 join key 给 map 阶段，因此left semi join 中最后 select 的结果只许出现左表。因为右表只有 join key 参与关联计算了。

### LEFT ANTI JOIN(代替 not in)

- LEFT ANTI JOIN 本质上就是NOT IN/EXISTS 子查询的表现，是NOT IN的一种优化。
- 当on条件不成立时，才返回左表中的数据，保留在结果集中。

## 24. Hive的mapjoin

- 在Map 端进行join，其原理是broadcast join，即把小表作为一个完整的驱动表来进行join操作。通常情况下，要连接的各个表里面的数据会分布在不同的Map中进行处理。即同一个Key对应的Value可能存在不同的Map中。这样就必须等到 Reduce中去连接。要使MapJoin能够顺利进行，那就必须满足这样的条件：除了一份表的数据分布在不同的Map中外，其他连接的表的数据必须在每个Map中有完整的拷贝。Map Join会把小表全部读入内存中，在Map阶段直接拿另外一个表的数据和内存中表数据做匹配 (这时可以使用Distributed Cache将小表分发到各个节点上，以供Mapper加载使用)，由于在map时进行了join操作，省去了reduce运行的效率也会高很多。

## 25. Hive Shuffle的具体过程

- 在 Shuffle 阶段，Hive 会将 Map 阶段的输出结果进行重新分区，以便将具有相同 Key 的数据发送给同一个 Reduce 任务进行处理。Shuffle 阶段的执行过程包括以下几个步骤：
  - Partition：根据 Reduce 任务的数量和 Map 阶段输出结果的 Key 进行分区，将具有相同 Key 的数据发送到同一个分区中。
  - Sort：对每个分区内的数据进行排序，以便在 Reduce 阶段进行合并和聚合操作时能够更高效地处理数据。
  - Combiner：可选的步骤，用于在 Map 阶段进行一些局部的合并和聚合操作，以减少数据传输量和网络开销。

## 26. Hive有哪些保存元数据的方式，都有什么特点?

- 内嵌模式：将元数据保存在本地内嵌的derby数据库中，内嵌的derby数据库每次只能访问一个数据文件，也就意味着它不支持多会话连接。
- 本地模式：将元数据保存在本地独立的数据库中（一般是mysql），这可以支持多会话连接。

- 远程模式：把元数据保存在远程独立的mysql数据库中，避免每个客户端都去安装mysql数据库。
- 内存数据库derby，占用空间小，但是数据存于内存，不稳定；
- mysql数据库，数据存储模式可自己设置，持久化好，查看方便。

## 27. Hive SQL实现查询用户连续登陆，讲讲思路

- 查询连续登录：使用 Hive SQL 的窗口函数和分析函数来查询用户的连续登录情况。具体步骤如下：

```
WITH login_rank AS (
  SELECT
    user_id,
    login_time,
    ROW_NUMBER() OVER (PARTITION BY user_id ORDER BY login_time) AS row_num,
    LAG(login_time) OVER (PARTITION BY user_id ORDER BY login_time) AS prev_l
  FROM
    login_log
)
SELECT
  user_id,
  MIN(login_time) AS start_time,
  MAX(login_time) AS end_time
FROM
  login_rank
WHERE
  DATEDIFF(login_time, prev_login_time) = 1
GROUP BY
  user_id;
```

1. 使用窗口函数 ROW_NUMBER() 对登录日志表按照用户ID和登录时间进行排序，并为每个用户的登录记录分配一个连续的行号。
2. 使用分析函数 LAG() 来获取前一行的登录时间。
3. 使用条件判断，如果当前登录时间与前一行的登录时间相差一天，则表示用户连续登录。
4. 最后，根据连续登录的条件进行筛选，可以使用子查询或者临时表来实现。代码如下

## 28. Hive的开窗函数有哪些

- 窗口函数
  - lag(col,n,val)：查询当前行前边第n行数据，如果没有默认为val
  - lead(col,n,val)：查询当前行后边第n行数据，如果没有默认为val
  - first_value(col,true/false)：查询当前窗口第一条数据，第二个参数为true，跳过空值
  - last_value(col,true/false)：查询当前窗口最后一条数据，第二个参数为true，跳过空值
- 排名开窗函数（RANK、DENSE_RANK、ROW_NUMBER、NTILE）
  - 排名开窗函数可以单独使用 Order by 语句，也可以和 Partition By 同时使用
  - Partition By 用于将结果集进行分组，开窗函数应用于每一组
  - Order By 指定排名开窗函数的顺序，在排名开窗函数中必须使用 Order By 语句
- 聚合开窗函数（SUM、AVG、MAX、MIN、COUNT）
  - 聚合开窗函数只能使用 Partition By 子句，Order By 不能与聚合开窗函数一同使用

## 29. - rank()排序相同时会重复，总数不变。

- dense_rank()排序相同时会重复，总数会减少。
- row_number()排序相同时不会重复，会根据顺序排序。

## 30.

## Hive count(distinct)有几个reduce，海量数据会有什么问题

- count(distinct)只有1个reduce。因为使用了distinct和count(full aggreates)，这两个函数产生的mr作业只会产生一个reducer，而且哪怕显式指定set mapred.reduce.tasks=100000也是没用的。
- 当使用count(distinct)处理海量数据（比如达到一亿以上）时，会使得运行速度变得很慢，因为出现了很严重的数据倾斜。

## 31. HQL：行转列、列转行

- 什么是行转列：把一行中不同列的数据放在同一列当中或者把多行中不同行的数据转为一列

### 行转列函数

- CONCAT(string a string b...) 返回输入字符串拼接后的结果，可以输入多个字符串。
- CONCAT_WS(separator,str1,str2,...) 第一个参数separator作为其他列的分隔符，其他参数必须为字符串或者字符串数组 分隔符为NULL 如果后面的参数存在 null 或者空字符串则跳过这些参数把同一行，不同列的合并为一个值
- COLLECT_SET（a）对一个基本数据类型字段进行去重汇总，产生array类型的字段

- COLLECT_LIST（a）对一个基本数据类型字段进行不去重汇总，产生array类型的字段

## 33. 分析函数中加Order By和不加Order By的区别？

- 当为排序函数，如row_number(),rank()等时，over中的order by只起到窗口内排序作用。
- 当为聚合函数，如max，min，count等时，over中的order by不仅起到窗口内排序，还起到窗口内从当前行到之前所有行的聚合，不加则整个分区聚合。

## 34. Hive优化方法

- 配置fetch抓取：修改配置文件hive.fetch.task.conversion为more，修改之后全局查找和字段查找以及limit都不会触发MR任务。
- 善用本地模式：大多数的Hadoop Job要求Hadoop提供完整的可扩展性来触发大数据集，不过有时候hive的输入数据量非常的小，这样的情况下可能触发任务的事件比执行的事件还长，我们就可以通过本地模式把小量的数据放到本地上面计算。
- Group by：默认情况下，map阶段同一key的数据会发给一个reduce，当一个key数据过大就会倾斜，并不是所有的聚合操作都需要reduce端完成，很多聚合操作都可以现在map端进行部分聚合，最后在reduce端得出最终结果。
    - 开启在map端聚合：hive.map.aggr = true。
    - 在map端进行聚合操作的条目数：hive.groupby.mapaggr.checkinterval = 100000。
    - 有数据倾斜的时候进行负载均衡：hive.groupby.skewindata = true。
- 行列过滤：列处理：只拿自己需要的列，如果有，尽量使用分区过滤。行处理：在分区裁剪的时候，当使用外关联的时候，先完全关联再过滤。
- 动态分区调整：(慎用)
    - 开启动态分区：hive.exec.dynamic.partition=true。

- 设置为非严格模式：hive.exec.dynamic.partiton.mode = nostrict。
- 实操：创建分区表、加载数据到分区表中、创建目标分区、设置动态分区、查看目标分区表的分区情况。

- 小文件进行合并：在map执行之前合并小文件，减少map的数量，设置 set hive.input.format = org.apache.hadoop.hive.ql.io.CombineHiveInputFormat。
- 调整map的数量和reduce的数量。
- 并行执行：在Hive中可能有很多个阶段，不一定是一个阶段，这些阶段并非相互依赖的。然后这些阶段可以并行完成，设置并行：set hive.exec.parallel = true。
- JVM的重用，缺点是task槽会被占用，一直要等到任务完成才会释放。

## 35. Hive表字段换类型怎么办

- 若表中无数据，则使用alter table change column命令直接修改

```
alter table mini_program_tmp_month_on_month change column month_on_month_reading month_on_mo
```

- 若表中有数据，上述方法会导致数据无法展示。则通过下列方法进行修改

```
alter table mini_program_result rename to mini_program_resul_copy;
```

```
create table mini_program_result (
user_id  string,
msisdn   string,
imsi     string)
partitioned by (city string,sdate string)
row format delimited fields terminated by ',';
```

```
set hive.exec.dynamic.partition=true;
set hive.exec.dynamic.partition.mode=nonstrick;
insert into table mini_program_result partition(city,sdate)
select
    t.user_id,
    t.msisdn,
    t.imsi,
    t.city,
    t.sdate
from mini_program_resul_copy ;
```

- 原来的数据按照分区动态插入
- 创建一个和原来表结构相同表
- 更改表名

## 36. parquet文件优势

- parquet文件简介
  - Apache Parquet是Apache Hadoop生态系统的一种免费的开源面向列的数据存储格式。它类似于Hadoop中可用的其他列存储文件格式，如RCFile格式和ORC格式。
  - Apache Parquet 是由 Twitter 和 Cloudera 最先发起并合作开发的列存项目，也是 2010 年 Google 发表的 Dremel 论文中描述的内部列存格式的开源实现。和一些传统的列式存储（C-Store、MonetDB 等）系统相比，Dremel/Parquet 最大的贡献是支持嵌套格式数据（Nested Data）的列式存储。嵌套格式可以很自然的描述互联网和科学计算等领域的数据，Dremel/Parquet “原生”的支持嵌套格式数据减少了规则化、重新组合这些大规模数据的代价。
  - Parquet 的设计与计算框架、数据模型以及编程语言无关，可以与任意项目集成，因此应用广泛。目前已经是 Hadoop 大数据生态圈列式存储的事实标准。
- 行存和列存的区别

| A | B | C |
| --- | --- | --- |
| A1 | B1 | C1 |
| A2 | B2 | C2 |
| A3 | B3 | C3 |

PowerData



- 由此可知，行存适用于数据整行读取场景，而列存更适用于读取部分列数据（统计分析等）的场景。
- 在面向列的的存储中，每列的数据存储在一起：
- 在面向行的存储中，每列的数据依次排成一行，存储方式如下：
- 下图是拥有A/B/C三个字段的表

- parquet文件的优势
  - 更高的压缩比：列存使得更容易对每个列使用高效的压缩和编码，降低磁盘空间。（网上的case是不压缩、gzip、snappy分别能达到11/27/19的压缩比）。
  - 更小的IO操作：使用映射下推和谓词下推，只读取需要的列，跳过不满足条件的列，能够减少不必要的数据扫描，带来性能的提升并在表字段比较多的时候更加明显。

## 37. Hive的数据中含有字段的分隔符怎么处理？

- Hive 默认的字段分隔符为 Ascii 码的控制符\001（^A），建表的时候用 fields terminated by '\001' 。
- 注意：如果采用\t 或者\001 等为分隔符， 需要要求前端埋点和 JavaEE 后台传递过来的数据必须不能出现该分隔符， 通过代码规范约束。
- 一旦传输过来的数据含有分隔符， 需要在前一级数据中转义或者替换 （ETL）。通常采 用 Sqoop 和 DataX 在同步数据时预处理。

## 38. 如何创建二级分区表？

```
create table dept_partition2 (
deptno int,    -- 部门编号
dname string, -- 部门名称
)
partitioned by (day string, hour string)
row format delimited fields terminated by '\t ';
```


【技术分享】元数据与数据血缘实现思路

Apache Doris 夺命 30 连问！(上)

Apache Doris 夺命 30 连问！(中)

Apache Doris 夺命 30 连问！(下)

利用 flink retract 计算长生命周期下跨业务状态指标

走进电力大数据

我们是由一群数据从业人员，因为热爱凝聚在一起，以开源精神为基础，组成的PowerData数据之力社区。

如果你也想要加入学习，可关注下方公众号后点击“加入我们”，与PowerData一起成长！

PowerData

PowerData数据之力社区官方公众号

18篇原创内容

公众号


- END -

收录于合集 #大数据组件 17

上一篇 · Alluxio 开源数据编排系统源码完整解析（下篇）

喜欢此内容的人还喜欢

Mysql事务和锁

PowerData