Hive

**一、hive介绍**

**1. 背景**

（1）mapreduce学习成本高，开发成本高（一条HQL语句可以实现mapreduce数十行语句）

（2）mapreduce在实际中的计算问题基本都是结构化数据；针对结构化数据最好的分析工具是SQL

2. 什么是Hive

（1）由facebook实现并开源，贡献给了apache基金会

（2）是基于hadoop的数据仓库工具

- hive底层是hadoop，相当于hadoop上层的一个应用组件

- 数据仓库和数据库的区别

1）概念上

——数据库是用于管理精细化数据，一般情况下用于存储计算结果数据，并且分库、分别进行存储；

——数据仓库更像通常说的数据打包过程，里面存储的数据并没有细化区分，仅仅是一个数据管理的概念

2）用途上

——数据库通常用于联机事务处理OLTP(On-Line Transaction Processing)，就是通常的增删改处理

——数据仓库通常用于联机分析处理OLAP(On-line Analytical Processing)，通常用于查询

——hive不支持删除和修改操作，但是支持插入

3）使用上

——数据库使用的是标准SQL语句，分布式数据库hbase使用的是非标准化SQL语句（NoSQL）

——数据仓库使用的是方言版SQL，即HQL

4）模式上

——数据库本质上是写模式的数据库，hbase是非严格模式

——数据仓库通常是读模式

（3）可以将结构化的数据映射为一张数据库表（二维表）

- 结构化数据：通常是行列结构数据，每一行上的列比较规整

- 将文本中的每一行数据映射为表的每一条数据，每一列数据映射为表的字段

（4）并提供HQL（Hive SQL）查询功能

（5）底层数据是存储在hdfs上

- 在hive上建的表仅仅是对hdfs上的结构化数据进行映射管理

- 真实数据仍然在hdfs上，hive上不会存储数据，仅仅是一个管理数据的作用

- hive想要管理hdfs上的数据，就要建立一个关联关系：关联hive表与hdfs上的数据路径

- 这种关联是依赖于一个元数据库记录的，一般这个元数据库采用的是关系型数据库

- 真实生产中一般使用mysql作为hive的源数据库，但hive中内置的默认元数据库是derby，实际应用需要将其改成mysql

- hive元数据库记录的信息就是hive中表与hdfs数据路径的映射关系，以及hive表属性（内部表/外部表/视图）和表中的字段信息（字段类型和字段的顺序）

- 所以在HIVE中，元数据非常重要，元数据一旦丢失，hive中的库和表都会丢失

（6）hive本质是将SQL语句转换为mapreduce任务运行，使不熟悉mapreduce的用户可以方便地利用HQL处理和计算hdfs上的结构化数据

- hive库中存储了很多编好的map和reduce模板，用户在执行HQL语句的时候，它会被编译成mapreduce任务后执行

- hive实质上就是hadoop上的一个工具，数据存储在hdfs上，计算使用的是mapreduce

- 可以节省开发成本

（7）hive主要适用于离线的批处理

3. hive的特点

（1）优点

- 可扩展性强，并且是横向扩展(继承自hdfs)

- 纵向：在单台机器上扩展，受摩尔定律限制；横向：节点上扩展，理论上没有上限

- 延展性强：hive支持自定义函数

- 有良好的容错性（继承自hdfs）

- 适合海量数据离线统计分析

- 传统数据库一般不支持大数据，但查询速度比hive快（实时查询）

（2）缺点

- 不支持删改(delete,updata)操作，2.3.2版本开始支持插入操作

- hive的查询延时严重

- 因为hive的查询最终转换成mapreduce任务，mapreduce任务本身执行很慢

- 传统数据库通常支持实时查询

- hive不支持事务

- 因为hive做的最多的是查询，而增删改才需要绑定事务，所以没有必要支持事务

二、hive的架构：四层

1. 用户接口层

- 命令行客户端CLI，hive操作最常用的方式

- JDBC\ODBC方式，实质是用javaAPI连接数据库，也比较常用

- web ui，开发时候基本上不用

2. 元数据库：存储hive表与hdfs数据的映射关系

- 一般选用传统关系型数据库，企业用的最多时mysql

- 连接到数据库(mysql、derby)的模式分为三种：单用户模式、多用户模式、远程服务模式

- 元数据包括database、表名、表的列和类型、存储空间、分区、表数据所在目录等

3. thift服务：用来提供跨编程语言的服务

- 可以用各种编程语言操作hive

- hive本身可以解析sql语言，通过thift，可以用其他编程语言如java\rubi\c语言操作hive

4. drive驱动层

1）驱动器：驱动sql语句运行，及sql语句解析为mapreduce程序，最终将mapreduce程序提交到hadoop

2）编译器：将sql语句通过自带的map\reduce模板，编译为mapreduce程序

- 首先生成一个逻辑执行计划，再生成一个物理执行计划，并提交给hadoop

3）优化器：编译过程中有sql转化的mr程序存在包含关系，重复操作，优化器将重复操作过滤掉

三、hive的数据组织格式

1. 最外层的是数据库database

2. 表：hive中表分为四种

- 内部表（管理表）/外部表：

- 是两个相对的概念，不可能有一个表既是内部表、又是外部表

- 主要区别就是删除表的时候是否删除数据：内部表删除表的时候会删除数据和元数据，外部表删除表时不会删除数据、只会删除元数据

- 一般情况下存储公共数据的表要存放成外部表

- 分区表：作用是提升查询的整体性能

- 不同于hadoop中的分区：

- hive存储海量数据，查询会存在瓶颈，所以查询时一定要注意避免全表扫描select \* from table;

- 为了提升查询性能，就出现了分区表：将数据按照用户的业务存储在不同的目录下

- 存储时指定的不同目录，叫做分区字段

- 示例：select \* from table where city=beijing；执行时只会对指定分区扫描，不会全表扫描

- 实际工作中，一般以日期作为分区字段，一般一天存在一个目录文件夹

- 分桶表：

- 类似于hadoop中的分区，基于hash算法计算，将余数不同的输出到不同文件中

- 分区表是认为划分的，分桶表是程序自动决定的，只能指定桶的个数（类似hadoop分区的个数），数据分到哪个桶是程序决定的

- 分桶表作用：

—— 提升join的性能

—— 提升数据样本的抽取效率：直接拿一个桶中的数据作为样本数据

3. 视图view

- hive中的视图仅仅相当于一个sql语句的快捷方式或别名，因为在hive中只有逻辑视图，不存在物理视图

- 物理视图：是将sql语句的执行结果存储在视图中

4. 数据存储

- hive中的原始数据存储在hdfs, 与数据存在mysql中

四、hive的安装

1. 准备工作

- 保证hadoop可以正常运行

- 保证mysql可以正常运行

2. 安装模式：使用自带的derby数据库，几乎没人用

- 下载hive并解压

- 进入hive/bin，进行元数据库初始化：./schematool -dbType derby -initSchema

- 使用：启动hive: ./hive

- show databases; 显示现有数据库

- create database tests; 新建tests数据库

- use tests; 使用tests数据库

- create table test01; 在数据库tests中创建表test01

- show tables; 查看现有表

- exit; 退出hive

- 使用derby数据库的优点：搭建简单，容易上手

- derby数据库的缺点：

- 单用户的，用户只能在当前目录下执行操作

- 用户切换目录的时候，需要重新初始化元数据库，初始化完成后会在当前目录生成连两个文件

- derby.log日志信息

- metastore\_db 元数据信息

- 在当前目录启动hive，会加载当前目录的元数据库

- 造成在不同目录下启动hive的时候，元数据信息不能共享，要想元数据信息共享只能手动拷贝

- 多用户模式不适用

3. 安装模式：使用mysql作为元数据库，需要安装mysql(用yum或rpm方式pkgp)

- 安装完成：启动/etc/init.d/mysql start；停止/etc/init.d/mysql stop

- 配置root用户

- delete from user where user=''; 删除所有空用户

- update user set host='%' where host='localhost'; 将本地访问权限改为所有人员访问权限

- update user set authentication\_string=PASSWORD('xxxxxxx') where user='root'; 修改root用户登录密码

- flush privileges; 刷新权限设置

- 重启mysql : service mysql restart

- root登录: mysql - root -p （输入新设密码登录）

- show databases; 查看数据库

- use mysql; 使用mysql数据库

- show tables；查看所有表

- select \* from user； 查看user表

- create database test; 创建数据库test

- use test; 进入test数据库；

- create table test01(id int); 创建一张名为test01的表，只有一个字段id，类型为int

4. 安装hive

- 下载、上传、解压

- 将解压后的hive安装包移动到usr/local目录下，并更名为hive

- 配置/hive/conf目录下：

- 修改hive-default.xml，这个文件很大，建议创建一个hive-site.xml，并加入如下配置信息

- <name>javax.jdo.optioin.ConnectionURL</name> 源数据库的连接url

<value>jdbc:mysql://localhost:3306/myhive?createDatabaseIfNotExist=true</value>

- <name>javax.jdo.option.ConnectionDriverName</name> mysql驱动

<value>com.mysql.jdbc.Driver</value>

- <name>javax.jdo.option.ConnectionUserName</name> mysql用户名

<value>root</value>

- <name>javax.jdo.option.Con nectionPassword</name> mysql密码

<value>root</value>

- 初始化源数据库：hive1.0版本不需要，直接运行启动即可；2.0版本需要

- 将mysql的驱动包下载并放到/hive/lib目录下

- . /schematool -dbType mysql -initSchema

- 启动hive:

- 先启动hdfs服务：start-all.sh

- 还没有配置环境变量，在bin/执行脚本hive

- 如果启动hive显示command not found, 可以尝试source /etc/profile

- show databases; create database mytest; use mytest; create table tt(id int);

- insert into table tt values(1) ; 在tt表中插入一条数据1; 因为调用mapreduce操作，插入数据效率很低(54.87s)

- mysql模式下，所有用户共享原数据

- 默认情况下，原数据放在hdfs上的user/hive/warehouse（hadoopm: 50070/browse the file system）

- hdfs dfs -ls /user/hive/warehouse/ 用来察看数据仓库里文件的linux命令

- 自定义存储目录：修改hive/conf/hive-site.xml文件

- 目录下面的 .db文件为数据库文件，下面存放表

- 执行insert语句，每执行一次，都会生成一个单独的小文件，所以不建议使用insert(效率低，小文件）

- 配置环境变量：vi /etc/profile

- export HIVE\_HOME=/usr/local/hive； $HIVE\_HOME/bin

- source /etc/profile

5. 是否有必要在每个节点都安装hive

- hive是一个操作hadoop的客户端，没有必要每台机器都装，经常使用hive的节点安装即可

- mysql元数据库一定相同，否则各客户端获取的数据会不同，所以源数据库配置必须要相同（同一个mysql地址下的同一个库）

- 所以hive不存在集群模式，下面各节点都是客户端

**五、hive操作**

1. 简单操作：在hives上建立一个表，与上传数据关联

- 建立库： create database mytest;

- 建表：

- create table salary(id int,name string, department string, gender string, age int, job string, timeIn string, education string)

> row format delimited fields terminated by ',' 指定列分隔符

> lines terminated by '\n' 指定行分隔符

- 查看表信息：desc salary;

- 加载数据：从本地加载

- load data local inpath '/srv/ftp/salary.csv' into table salary;

- load操作相当于将本地文件上传到hdfs上hive数据存储目录下user/hive/warehouse

- 如果直接把文件直接上传到hive数据存储目录下，也可以查询到

2. 元数据库

- myhive/DBS：数据库的描述信息，每当hive中创建一个数据库，就在这个表中插入一条数据

- 保存信息：数据库在hdfs上的地址、数据库的名称和创建者

- myhive/TBLS：表的描述信息，每当在hive上创建一个表，就在这个表中插入一条数据

- 保存的信息：创建者、创建时间、名称、类型

- manage-table管理表：即内部表（相对外部表）

- myhive/COLUMNS\_V2: 存储表的列信息，每当表中添加一个列，就在这个表中添加一条数据

- 保存的信息：对应每张表的列名、类型、创建表时每列的先后顺序

- 修改这表中的列名，hive中表的列名也会被修改，说明hive中表的信息是从元数据库中加载的

- 注意：元数据虽然可以手动修改，但是一定不要修改，因为有可能引起源数据表中的原始数据发生改变

3. hive的beeline方式连接

（1）三种连接方式：

- cli客户端连接：又分为两种方式

- 以客户端命令方式进入，前提是必须在hive安装的节点上

- 这种方式是启动服务的同时，进入客户端

- beeline方式(hiveserver2方式)连接: 实际开发中这种方式用的比较多

- hiveserver2方式：将hive服务启动为一个后台进程(类似一个hive的服务端)，客户机就可以进行连接

- jdbc连接的时候，前提是保证hive服务的启动

- jdbc

- webui：基本不用

（2）hiveserver2

- 这种方式必修修改配置文件，允许远程连接

- 修改配置文件：/usr/local/hadoop/etc/hadoop# vi hdfs-site.xml

- <name>dfs.webhdfs.enabled</name> 开启hdfas远程连接的服务

<value>true</value>

- 修改配置文件：/usr/local/hadoop/etc/hadoop# vi core-site.xml

- <name>hadoop.proxyuser.root.hosts</name> 可以通过哪些主机远程连接

<value>\*</value> \* 表示允许所有主机远程访问

- <name>hadoop.proxyuser.root.groups</name> 允许哪些用户在访问

<value>\*</value> \* 表示允许所有用户组访问

- 上述配置中，原来用的是hadoop而不是root，但启动hiveserver2的时候一直挂在那里不动，改成root则运行正常了

- 把修改的配置文件发送到其他主机节点上

- 修改完成配置文件，需要重启集群（hdfs和yarn）

- 启动hiveserver2: usr/local/hive/bin文件夹下运行hiveserver2

- 这种方式是挂起启动，启动后一直挂在主机页面

- 使用nohup(no hang up) hiveserver2启动，表示后台启动，不会挂在主机页面上

- 后台启动需要将日志保存下来：0-标准输入，1-标准输出，2-错误输出

- nohup hiveserver2 2>>~/error.log 1>>~/std.log

- 1>>~/std.log表示将标准输出1的日志存入根目录下的std.log文件

- 标准输出1用处不大，可以保存在~/dev/null文件(可以丢弃的文件)： - 1>>~/dev/null

- 2>>~/error.log表示将错误输出2的日志存入根目录下error.log文件

- 在远程主机上启动beeline: usr/local/hive/bin文件夹下运行binline，之后运行 !connect jdbc:hive2://hadoopm:10000

- hive和mysql的连接方式

- hive连接的url: jdbc: hive2: //hadoopm:10000

- mysql连接的url: jdbc: mysql: //localhost: 3306

- 如果core-site.xml内部配置用的root(而非hadoop)， 则运行命令后根据提示输入username为root, password直接回车即可

- 如果core-site.xml内部配置用的hadoop（而非root），则运行命令后使用的是安装hive的用户名和密码

**六、hive的sql语句**

**1. DDL**

（1）库操作

- 创建库：create database [if not exists] 库名;

- [if not exists]是为了避免报错，同样适用于表操作

- 删除库:

- drop database [if exists] 库名 [restrict];

- [if exists]是为了避免报错，同样适用于表操作

- restrict是默认（严格模式，只能删除空库）

- drop database 库名 cascade; 强制删除非空的库（空库也可以删除）

- 察看库的列表信息:

- show databases;

- show databases like 's\*'; 察看库名s开头的库

- 使用库: use 库名；

- 察看正在使用的库： select current\_database();

- 没有使用的时候，默认是default库

- 察看库信息: desc database 库名;

- 修改库： 基本不用

（2）表操作

1) 创建表

create [external] table [if not exists] table\_name

[(col\_name data\_type [comment col\_comment], .....)]

[conmment 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]

- [external] 是否创建外部表，加上这个关键字是外部表，不加创建的是内部表

- [if not exists]: 同创建库

- [(col\_name data\_type [comment col\_comment], .....)] 建表字段名+字段类型【+字段描述信息】

- 例如create table stu(id int comment 'decribe student ID')

- [conmment table\_comment] 描述表信息

- 例如create table stu(id int comment 'decribe student ID') comment 'describe ...'

- [partitioned by (col\_name data\_type [comment col\_comment], ...)] 指定分区字段

- col\_name data\_type：指定分区字段。注意：分区字段（类型）不是建表的字段（类型）

- [clustered by (col\_name, col\_name, .....)[sorted by (col\_name [asc|desc], ...)] into num\_buckets buckets]

- clustered by (col\_name, col\_name, .....) 指定分桶（类似分区）字段，分桶字段一定是建表语句中的字段

- sorted by (col\_name [asc|desc], ...) 指定在某一个分桶内是否需要排序以及排序规则(升序|降序)

- 排序字段一定是即建表语句中的字段

- into num\_buckets buckets 指定分桶个数

- [row format row\_format] 指定分隔符

- delimited fields terminated by ‘’指定列分隔符

- lines terminated by ‘’ 指定行分隔符

- [stored as file\_format] 指定最终表数据的存储格式，常用格式如下：

- textfile： 文本格式，默认格式

- rcfile: 行列结合的格式

- parquet: 压缩格式

- [location hdfs\_path] 指定hive表在hdfs中的存储路径

- 不指定则默认在配置的路径下存储（如果没有配置则存储在/user/hive/warehouse下），

- 指定则存储在指定的路径

* 创建内部表
  + create table if not exists manage\_table(id int, age int) row format delimited fields terminated by ',';
  + 可以通过mysql中的myhive数据库TBLS表中查看类型
* 创建外部表
  + create external table if not exists external\_table(id int, age int, name string) row format delimited fields terminated by ',';
  + 可以通过mysql中的myhive数据库TBLS表中查看类型
* 创建分区表
  + 分区表存的是不同的目录(按分区的名字)，所以分区表在添加数据之前要先添加分区
    - Alter table ptn\_table add [if not exists] partition(city=’beijing’);
  + Create [external] table ptn\_table(id int, name string, age int) partitioned by (city string) row format delimited fields terminated by ',';
  + 注意：分区字段一定不是建表字段
* 创建分桶表
  + 分桶表类似hadoop中分区的概念，默认分区是按照hash进行分区的
  + Create table buk\_table(id int, name string, age int) cluster by (id) sorted by (age desc) into 4 buckets row format delimited fields terminated by ',';
    - cluster by (id)：相当于指定map端发送的key， id.hashCode()
    - into 4 buckets：相当于指定numreducetask，相当于id.hashCode() % num\_buskets
  + 分桶字段一定是建表语句中的字段

- 建立复制表：like关键字

* + Create table copy\_table01 like ex\_table; ex\_table是一张外部表
  + 复制出来的表是否外部表，跟复制时是否指定external关键字有关
  + 复制表只能复制表结构，并不复制表数据
* 建立查询表：c-t-a-s
  + Create table as select … from …
  + 建立出来的是否外部表，跟复制时是否指定external关键字有关
* 建表的注意事项：
  + 如果建立的表要从hdfs加载数据，则必须建成外部表，因为内部表删除时会同时删除元数据和原始数据，而hdfs上的数据是公共数据，这样可能造成公共数据丢失，按照这些数据做好的业务模块数据运行异常
  + 建立外部表一定要指定存储路径，且存储路径要指向hdfs原始数据的存储路径，因为外部表从hdfs加载数据时是移动操作，原始数据移动后，会造成别人使用这些数据制定的业务逻辑所需的数据丢失

2）查看表

* Show tables;
* Show tables in 库名； 察看某一个库中的所有表
* Show tables like ‘s\*’; 查看以s开头的表

3）查看表信息

- desc 表名； 查看表的字段信息

- desc formatted 表名； 格式化显示表的详细信息

- desc extended 表名； 显示表的详细信息

4）删除表

- drop table if exists表名；

- 如果是从hdfs上加载数据建内部表，当内部表删除表的时候，元数据会被删除，同时hdfs上的原始数据会一并删除；

- 如果从hdfs上加载数据建外部表，删除表的时候元数据会被删除，但hdfs上的原始数据不会一并删除。

- 所以，从hdfs上加载数据创建的表，必须建成外部表，因为hdfs上原始存储的数据一般为公共数据

5）清空表数据（表结构保留）

- truncate table 表名;

6）修改表

- 修改表名字

* + Alter table 原表名 rename to 新表名;

- 修改表字段信息

* + 添加字段：alter table 表名 add columns (字段1 类型1, 字段2 类型2…)，添加的新字段必须指定类型
  + 修改字段：

- 修改字段名和类型：alter table 表名 change 原字段 新字段 新字段类型

- alter table copy\_table01 change age ages string;

- 修改表字段的时候，注意字段类型之间的匹配：只能小类型转大类型(如int-string)

* + 替换表的列
    - alter table 表名 replace columns (字段1 类型1, 字段2 类型2, …)

- 修改表分区信息

* + 添加分区
    - Alter table 表名 add if not exists partition (分区字段值)
      * Alter table ptn\_table add if not exists partition (city=’beijing’);
      * 添加分区的过程中，还可以指定分区的存储路径
        + Alter table ptn\_table add if not exists partition (city=’nanjing’) location ‘/user/nanjing’;
        + 否则分区将存储在表路径下面的分区路径中（默认）
  + 修改分区
    - 修改分区存储路径
      * Alter table 表名 set partition(分区字段值) location ‘路径’;
        + Alter table ptn\_table set partition(city=’beijing’) location ‘/user/beijing’;
  + 删除分区
    - Alter table 表名 drop if exists partition (分区字段值)
      * Alter table ptn\_table drop if exists partition (city=’nanjing’);

7）其他辅助命令

- 查看建表语句： show create table 表名;

- 查看分区：一定针对分区表

* + Show partitions 表名；查看表的分区信息
  + Show partitions 表名 partition(分区字段值); 查看表中某一个分区的信息

- 查看库的信息

* + Desc database 库名;
  + Desc database extended 库名；
* 查询属性值
  + Set hive.exec.dynamic.partition.mode;
* 设置hive运行的本地模式
  + Set hive.exec.mode.local.auto=true; （=false为集群模式）

**2. DML**

**（1）数据加载：**

1）load方式

- load data inpath(从hdfs加载) / local inpath(本地加载) ‘路径’ into table 表名

- 从本地加载相当于复制，将本地的文件复制到hive的默认路径下

- 从hdfs加载：

- 先将文件上传到hdfs

- hadoop fs –put 本地文件路径 hdfs路径

- 例如：hadoop fs –put /srv/ftp/salary.txt /upload/

- 将hdfs的数据加载到hive的表中

- 例如: load data inpath /upload/salary.txt into table salary;

- 这个操作是移动操作，因为hive创建的表是存储在hdfs上，原来从本地上传的文件也是在hdfs上，同在hdfs上没有必要存两份，因此hive加载数据到表的操作，实际上是将上传到hdfs上的文件移动到hive默认的存储目录之下

- 从hdfs加载数据，一定要指定存储路径（location…要指向原数据的存储路径），因为这个操作是移动操作，一旦移动则使用这个数据做好的业务逻辑的数据会丢失

2）insert方式：

- 单条数据插入：insert into table 表名 values(数据值);

- 会先建立一个临时表，稍后再将临时表中的数据拷贝的表中

- 效率低

- 多条数据单重插入：从一个表中查询，将查询结果全部插入到指定的表中

- insert into table 表名 select \* from 表名 where 条件语句

- insert into table salary select \* from xxx where age>20;

- 多重插入：一次插入多个表，但是对基表只扫描一次，减少基表的扫描次数

- from 基表表名

insert into table 表1 select …

insert into table 表2 select …

* hive> from salary

> insert into table salary\_query02 select \* where age>35

> insert into table salary\_query03 select \* where age<=35;

**（2）分区表的数据插入**

**-** 分为静态和动态分区

- 静态分区适用于分区比较少、且分区数量可以确定的情况下

- 动态分区适用于分区个数多，或分区个数不能确定的情况下

- 加载数据前必须先添加分区：alter table 表名 add partition(分区字段值)

- 注意：分区字段一定不是原表的已有字段

- 可以在建表的时候添加分区

- 向分区表加载数据一定要指定加载到哪个分区中

- load data local inpath 路径 into table 分区表名 partition(分区字段值);

- 但是，分区表的数据加载一般不使用load方式，因为没有数据检测（不管数据是否加载正确都不会报错）

- 单条插入：insert into table 表名 partition(分区字段值) values(插入值) 不建议这种方式

- 多条插入：insert into table 表名 partition(分区字段值) select \* from 基表名 where 查询条件;

- 动态分区：

- 上面的方式属于静态添加分区：在加载数据之前手动添加分区

- 当分区个数不确定的时候要使用动态分区：可以根据数据自动进行分区，不需要提前添加分区，而是加载数据时进行分区处理。

- 将分区模式改成非严格模式：

- set hive.exec.dynamic.partition=true; 是否开启动态分区（2.x默认开启，1.x版本默认false）

- set hive.exec.dynamic.partition.mode=nonstict; 设置动态分区的模式为“非严格”

- 严格模式strict：至少要手动添加一个静态分区

- 非严格模式：完全动态添加分区

- insert into table 表名 partition(分区字段) select 查询字段,..分区字段 from salary;

- 必须把分区字段放在最后，否则会默认按最后一个字段分区

- 多重分区：多个分区字段

- insert into table salary\_ptn partition(department,age) select id,name,…,education,department,age from salary;

- hive.2x的目录结构是按照多个分区字段的顺序进行划分的

- hive.1x进行多重分区，必须手动指定第一级分区

- 分区字段在使用的时候是作为普通字段使用（查询语句），只是在建表和加载数据的时候不一样

**（3）分桶表的数据插入**

- 分桶类似hadooop中的分区概念

- load方式：分桶表不允许用load方式加载数据，只能用insert方式

- insert方式

- 先建立分桶表

- create table if not exists salary\_buk(id int,name string,department string,gender string,age int,job string,timein string,education string)

> clustered by (age) sorted by (gender) into 3 buckets

> row format delimited fields terminated by ',';

- 加载数据

- insert into table salary\_buk select \* from salary; 以下是输入的日志记录

- In order to change the average load for a reducer (in bytes):

set hive.exec.reducers.bytes.per.reducer=<number> 每个reducer的吞吐量，默认256M

- In order to limit the maximum number of reducers:

set hive.exec.reducers.max=<number> reducetask的最大执行个数，默认1009

- In order to set a constant number of reducers:

set mapreduce.job.reduces=<number> 设置reducetask实际执行个数，默认-1（代表未设置使用默认值1）

* 分桶表的分表数按照设置的分桶个数(clustered by (age) sorted by (gender) into 3 buckets)分为3张表
  + - 分桶方式为按：分桶字段值(age值)%分桶数(3)的结果（分别为0，1，2）分表

**（4）数据导出**

- 语法：insert overwrite [local] directory directory1 select\_statement;

- 单重导出导出到本地insert overwrite local directory ‘本地路径’ select …;

- insert overwrite local directory ‘/srv/ftp/salary01.txt’ select \* from salary where age>=30;

- 多重导出

- from salary

Insert overwrite local directory /srv/ftp/salary01.txt’select \* where age>=30

Insert overwrite local directory /srv/ftp/salary02.txt’select \* where age<30;

- 导出到dfs不需要加local 关键字

**（5）查询语句**

1）一般查询语法：select … from … (where …) (group by …) (order by …) (limit …)

2）注意：

- 在hive中要尽量避免select \* from，属于全表扫描查询，因为hdfs存储是分布式查询，全表扫描效率低

- 支持select、union all、join(left,right,full-join)、like、where、having、各种聚合函数，支持json解析

- 支持自定义函数查询

- UDF(User Defined Function): 自定义函数，一对一解析函数（进去一条数据，解析出来一条数据）

- UDAF(User Defined Aggregate Function):自定义聚合函数，多对一，类似与sum\avg\count函数

- UDTF(User Defined Table Function): 自定义表函数，一对多，如列转行（一行拆开成多列explode函数）

- 不支持update和delete

- 虽然支持in/exists(老版本不支持)，但效率低，hive推荐使用半连接（semi join）方式来代替，这样效率更高

- 内连接：两个表中都有的字段才能连接上

- 外连接：

- 左外连接：以左表为基表，左表有的全部关联；右表有的字段会关联上，左表有右表没有的字段显示null

- … left outer join b on a.id=b.id

- 右外连接：与左外连接相反

- … right outer join … on …

- 全外连接：两个表合并，左表有的关联左表的，右表有的关联右表的，没有的显示null

- … full outer join … on …

- 半连接semi join：只取其中的某一个表

- 左半连接left semi join: 显示在右表中存在的左表的数据

- 右半连接right semi join: 显示左表中存在的右表的数据

- 不支持case…when…

3）查询语法：

- select [all|distinct] select\_condition1, select\_condition2, …

from table\_name a

[join table\_other b on a.id=b.id]

[where where\_condition]

[group by col\_list [having\_condition]]

[cluster by col\_list | [distribute by col\_list] [sort by col\_list | order by col\_list [asc|desc]]]

[limit number]

- [all|distinct]: 显示所有/去重

- select\_condition：查询的字段

- from table\_name a | table\_other b：查表的两张表名，a、b为给表起的别名

- [join table\_other b on a.id=b.id]： 关联条件（a.id=b=id）

- [where where\_condition]: 过滤条件

- [group by col\_list [having\_condition]]：分组条件

- 排序相关字段

- cluster by col\_list：先分桶 + 再排序

- 等于distribute by + sort by： 在分桶的基础上，每个桶中进行排序

- cluster by 不能与sort by同时使用

- 当distribute by与sort by字段相同时，才能用cluster by代替；

-当distribute by与sort by字段不相同时,要单独使用distribute by和sort by

- distribute by col\_list: 只分桶，不排序

- 类似于分桶，按照排序字段和reducetask个数进行分桶（取余），但是并不排序

- sort by col\_list [asc|desc]: 局部排序

- 在每一个reducetask中进行排序，字段中的某一个值最终到哪个reducetask的排序结果输出中是随机的

- 当reducetask个数为1时与order by的结果一样，相当于全局排序

- 设置reudcetask个数：set mapreduce.job.reduces=<number> ，默认值是-1（表示1个reducetask）

- order by col\_list [asc|desc]：全局排序,不给排序规则的情况下按默认(升序)

- 建表语句中的clustered by\sorted by\partitioned by 与查询语句中的order by\sort by\cluster by没有关系

- 分桶表的作用：

- hive中的表都是大表（表1和表2），关联时大表1中的每个数据都要全表扫描大表2的数据，效率低

- 将两个表都进行分桶：按照相同的字段进行分桶，且分桶的数量成倍数关系

- 分桶数量相同时，两张大表取余结果相同的都在同一个分桶表，关联时只需要进行相应的分桶表扫描即可

- 分桶数量成倍数关系是原理类似，只不过是大表的分桶表之间的一对多的关系

- 分桶可以极大提升jion效率

- 提高抽样的性能：抽样按照散列形式，分桶按照hash算法，分桶表中的数据呈散列形式

3. hive的fetch过程：抓取

- 在执行hql语句的时候，一旦要执行mapreduce(mr)，效率降低，这是hive内部设置一些语句不执行mr程序，数据直接从本地抓取，就是fetch过程

- 什么时候执行mr：属于hive的优化，但是各个版本的优化方式不同

- hive2.x版本中hive的执行引擎支持spark和tez，但默认是mr

- 官网文档的相关内容

|  |
| --- |
| hive.fetch.task.conversion  * Default Value: minimal in Hive 0.10.0 through 0.13.1, more in Hive 0.14.0 and later * Added In: Hive 0.10.0 with [HIVE-2925](https://issues.apache.org/jira/browse/HIVE-2925); default changed in Hive 0.14.0 with [HIVE-7397](https://issues.apache.org/jira/browse/HIVE-7397)   Some select queries can be converted to a single FETCH task, minimizing latency. **Currently the query should be single sourced not having any subquery and should not have any aggregations or distincts (which incur RS – ReduceSinkOperator, requiring a MapReduce task), lateral views and joins.**  Supported values are none, minimal and more.   1. none:  Disable hive.fetch.task.conversion (value added in Hive 0.14.0 with [HIVE-8389](https://issues.apache.org/jira/browse/HIVE-8389))   - 所有的hql语句都要转成mr   1. minimal:  SELECT \*, FILTER on partition columns (WHERE and HAVING clauses), LIMIT only   - select \* 不走mr  - 在分区表中的过滤语句（where, having）  - 分页limit语句  - 剩下的都走mr程序  2. more:  SELECT, FILTER, LIMIT only (including TABLESAMPLE, virtual columns). "more" can take any kind of expressions in the SELECT clause, including UDFs.(UDTFs and lateral views are not yet supported – see [HIVE-5718](https://issues.apache.org/jira/browse/HIVE-5718).)  - 所有的单独的select语句  - 所有的过滤（where, having）  - 所有的limit分页 |

**4. join补充**

**1）注意：**

- join只支持等值连接

- select a.\* b.\* from a join b on a.id=b.id

- 不等值连接map的key设置太复杂

- join只支持and连接，不支持or连接

- select a.\* b.\* from a join b on a.id=b.id and a.name=b.name

- or连接map的key设置太复杂

- join支持多表连接

- 当多表关联是同一个字段时，只需要一个mr

- 例如：a表: id, name; b表: id, age; c表: id, sex

- select a.\* b.\* c.\* from a join b on a.id=b.id join c on a.id=c.id

- 当多表关联不同字段,会转化成多个mr

- 例如：a表: id, name; b表: id, age; c表：name, sex;

- select a.\* b.\* c.\* from a join b on a.id=b.id join c on a.name=c.name

2）join的连接方式

- 内连接inner join，默认方式

- 两个表中都有的字段才能连接上

- select a.\*, b.\* from table1 a inner join table2 b on a.id=b.id;

- 外连接：以join为分界线，左边的是左表，右边的是右表

- 左外连接：以左表为基表，左表有的全部显示；左表有右表没有显示null；右表有左表没有的字段不显示

- select a.\*, b.\* from table1 a left outer join table2 b on a.id=b.id;

- 右外连接：与左外连接相反

- select a.\*, b.\* from table1 a right outer join table2 b on a.id=b.id;

- 全外连接：两个表合并，左表有右表没有,或右表有左表没有的显示null

- select a.\*, b.\* from table1 a full outer join table2 b on a.id=b.id;

- 半连接

- 用来替代in\exists语句: select id from table1 a where id in (1,2,5)

- 左半连接

- select a.\* from table1 a left semi join table2 b on a.id=b.id

- 只select左表，查找的是左表id在右表中id存在的数据，返回的是左表中存在的数据

- 右半连接：和左半连接相反

**七、hive高级**

**1. hive的数据类型**

（1）hive的数据类型

- 简单数据类型

- tinyint

- smallint

- int

- string

- bigint

- 复杂数据类型：基础类型的组合

- array数组：由一系列相同数据类型的元素组成，这些元素可以通过下标来访问

- 建表的分隔符语句：COLLECTION ITEMS TERMINATED BY ',';

- map类型：包含key-value键值对，可以通过key来访问元素

- 建表的分隔符语句：COLLECTION ITEMS TERMINATED BY ',' MAP KEYS TERMINATED BY ':';

- struct：可以包含不同数据类型的元素。这些元素可以通过”点语法”的方式来得到所需要的元素，比如user是一个STRUCT类型，那么可以通过user.address得到这个用户的地址

- 建表的分隔符语句：COLLECTION ITEMS TERMINATED BY ',';

（2）示例：

Id address score info

001 henan,xinjiang,Beijing yuwen:45,shuxue:56 18,男,playing

- create external table information

(id int, address array<string>, score map<string, int>, info struct<age:int, sex:string, hobby:string>)

row format delimited fields terminated by ‘\t’ 列的分隔符

collection items terminated by ‘,’ 列内集合元素的分隔符：数据、map、struct

map keys terminated by ‘:’ 列内元素内的分隔符：map

lines terminated by ‘\n’; 行分隔符

- 指定分隔符按照先列分隔符后行分隔符、列分隔符由外向内的顺序：

- 先指定列的分隔符

- 再指定列内元素的分隔符

- 再指定列内各元素内部的分隔符

- 行分隔符放在最后

**2. hive的视图：**

（1）创建视图，类似ctas语句

- create view 视图名 as 查询语句

- create view my\_view as select \* from salary where age>30;

（2）查看视图

- show tables; 显示表和视图列表

- show views；查看视图列表

- desc 视图名; 显示视图信息

- desc formatted 视图名：显示视图详细信息

- Table Type:VIRTUAL\_VIEW 虚拟试图

（3）删除视图

- drop view 视图名;

（4）注意事项：

- hive中的视图仅仅存储了sql语句的快捷方式

- hive中只有逻辑视图，没有物化视图

- hive中的视图只支持查询操作，不支持增insert删delete改updata

- hive中的视图保存

- 元数据是存在MYHIVE/TBLS

- 只保存了sql语句和元数据信息，没有保存执行结果（/user/hive/warehouse/没有相应的文件存储）

- hive中的视图在查询视图的时候才执行

**3. hive的内置函数（共计271个）**

**（1）概述：分为三类DUF\DUAF\UDTF**

- 用户自定义函数：一对一

- 用户自定义聚合函数：多对一

- 用户自定义表函数：一对多

**（2）查看内置函数**

- show functions; 察看内置函数（271个）

- desc function 函数名； 察看函数描述信息

- desc function extended 函数名; 察看函数详细信息

（3）ceil/floor函数

- floor(x): 返回不大于x的最大整数

- select floor(-5) from src limit 1;

- ceil(x): 返回不小于x的最小整数

**（4）字符串常用函数**

- substring(str, pos[, len]) 字符串截取：从源字符串str的pos位置开始截取长度len的字符串

- 截取都是从左向右，不论pos为正或负

- pos：截取开始的位置，可以为负值

- len参数如果省略则从pos位置截取到字符串末尾

- instr(str, substr): 从str中查找子串substr，返回子串在str第一次出现的起始位置索引

- 索引是从1开始计数

- 没有找到返回0

- concat（str1，str2，…strn）：字符串拼接

- 会按照传参的顺序拼接

- concat\_ws(sep, [string | array(string)]+)

- 用分隔符sep拼接字符串string或字符串组成的数组

- array(n1,n2,…nn): 生成数组[n1,n2,…nn]

- n1-nn都为数值型，则生成的数组元素都为数值型

- n1-nn为数值和字符串混合型，生成的数组元素都为字符串

- map(key0, value0, key1, value1...)：生成键值对构成的字典

- 默认奇数为是key,偶数位是value

**（5）数据处理常用函数**

- NVL(value,default\_value): value如果为null则返回默认值default\_value，否则返回value值

- IF(expr1,expr2,expr3): 如果表达式expr1为true,返回expr2,否则返回expr3

- explode(a): 把数组分成多行，或将map分成多个行列（多行两列）

- 参数a为array或map

- 当同时查询炸裂字段和普通字段时，需要使用横向虚拟视图

- select name,ct.value from table\_arr lateral view explode(city) ct as value;

- lateral view explode(city) : 炸裂后的横向虚拟视图

- ct为横向视图别名

- value为炸裂后的视图中每列(字段)的别名（array只有一列，map有两列（key,value））

- ct.value：查询name和对应的炸裂后的每列的值(map要查询ct.key, ct.value)

**（6）分组排序函数row\_number/rank/dense\_rank**

1）这三个函数用法类似

2）row\_number() over (分组排序规则): 用于顺序添加排序号

- 分组规则由partition by 指定；排序规则由order by 指定。排序一定要，分组可以不要

- 用于业务需求中求分组的topN，例如求每个部门年龄最大的：

Create table salary\_rownum as select id,name,department,gender,age,job,row\_number() over (partition by department order by age) as index from salary;

select \* from salary\_rownum where index=1; 求每个部门年龄最小的（要求年龄最大者，排序按照降序即可）

- row\_number()：按分组（上例中是按部门）添加排序号，从1开始，顺序添加

- 分组规则用partition by; 排序规则用order by。这种方式较常用

- 可以以分组用group by; 排序用sort by

2）rank()/ dense\_rank() 用于排序

- 用法：rank()/dense\_rank() over (分组排序规则)

- 示例：按年龄排序

- select id,name,department,gender,age,rank()/dense\_rank() over (order by age desc) as index from salary;

- rank()排序结果类似: 1 1 3 3 5 6 7 7 9 …… 计数排名

- dense\_rank()排序结果类似: 1 1 2 2 3 4 5 5 6 ……

- 按分组排序: select id,name,department,gender,age,rank()/dense\_rank() over (partition by department order by age desc) as index from salary;

4. hive的自定义函数

要用java编程和eclipse

5. Json解析：目标是解析value值

（1）get\_json\_object函数：获取json对应字段的值

- get\_json\_object(json\_txt, path)

- json\_txt：json格式的字符串

- path: 需要解析的字段的路径

- $: 取根对象，第一级节点

- .: 取子对象，用来访问下一级节点

- []: 取数组

- \*: 通配符for []

- 示例：{[‘movie’:1193,’rate’:5, ‘timestamp’:9700982, ‘uid’:1]}

- select get\_json\_object(line,‘$.[0].movie’) from 表名;

- 求评分次数最高的三部电影

- select get\_json\_object(t.line, ‘$.movie’) as movieid, count(\*) as total from rating as t

group by get\_json\_object(t.line, ‘$.movie’)

order by total desc limit 3;

group by执行在select之前，所以别名movieid不可用；order by在select后执行，所以别名可用

6. python脚本方式查询

（1）linux自带python：运行命令which python, 获取python路径: /usr/bin/python

- 编写python脚本：vi 路径 mypython.py

- 运行：python 文件名.py

（2）脚本使用select transform(脚本解析的字段) using ‘python 脚本名.py’from 表名;

- 要让hive找到脚本，要将脚本放到classpath:

- 在hive下执行命令：add file 脚本所在路径/脚本名.py;

- 察看文件：list files;

- 语法

- transform（需要解析的原数据库中的字段）using ‘python python脚本’ as (python解析出来的字段)

7. hive中分隔符

（1）hive的分隔符只能是单字节的，类似”::”分隔符识别为 : 和 空

- 默认的解析库只能解析单字节的

（2）使用多字节分隔符

1）方式一：修改序列化库，修改成正则表达式库

- 语句：

row format

serde ‘org.apache.hadoop.hive.serde2.RegexSerde’ 指定序列化类库（正则表达式库）

with serdeproperties(‘input.regex’=’(.\*)\\|\\|(.\*)’,’output.format.string’=’%1%s %2$s’)

- ‘input.regex’=’(.\*)[\\|\\|(.\*)](file:///\\|\\|(.*))’,’ 指定输入的正则表达式

- output.format.string’=’%1%s %2$s’ 指定输出的正则表达式，定义怎么从正则表达式中抽取结果

2）方式二：修改hive的底层源码，修改org.apache.hadoop.hive.serde2.lazy.LazySimpleSerde的源码

- 不建议，一般公司中也不会这样修改

**8. hive的shell操作**

**（1）进入hive之前的操作**

- hive [-hiveconf x=y]\* [<i filename>]\* [<-f filename> | <-e query-string>] [-S]

- -i 从文件初始化 HQL

- -e 从命令行执行指定的HQL

- -e后面跟一个sql语句

- 例如hive –e ‘show databases’

- 可以在不进入hive客户端的条件下，执行hive语句；

- -f 执行配置文件脚本

- hive -f 文件路径/脚本文件名.conf

- -v 输出执行的HQL语句到控制台

- -p <port> connect to Hive Server on port number

- -hiveconf x=y 设置hive/Hadoop的配置变量

- hive –hiveconf key=value;

- 例如：hive –hiveconf mapred.reduce.tasks=3;

- -S 表示以不打印日志的形式执行命名操作

**（2）进入hive之后的操作**

- quit/exit： 退出hive

- set key=value; 设置属性；set key; 查询属性；set -v: 显示hive的所有属性值

- add file: 添加脚本文件, 添加到hive的classpath下

- add jar jarname: 添加jar包，添加到hive的classpath下

- list file/list jar：显示class下的所有文件/jar包

- dfs [dfs cmd] 在hive中调用hdfs命令

- hdfs命令：hadoop fs –命令 hive调用hdfs命令：dfs –命令；

- hive数据就是在hdfs上，可以将hive理解成hadoop的一个客户端

- source file: 执行sql脚本文件（.sql）

- source 路径/文件名.sql

- 需要批量执行sql语句，可以把所有sql语句写入一个脚本，批量执行

- explain 查询语句; 查询后面sql语句的执行计划（步骤）

**（3） hive的配置信息设置**

- 四种方式

- hive/conf/hive-site.xml中配置

- 进入hive时，hive –hiveconf key=value 方式，或hive –f 路径/配置脚本文件名.conf

- 进入hive后，用set key=value 方式

- 加载顺序：1—>2—>3

- 生效顺序：最后加载的生效

**八、hive优化**

**1. 排序的选择：合理选择排序方式**

- order by 全局排序

- sort by 局部排序

- distribute by 分桶而不排序

- cluster by = distribute by + sort by ,条件是两者的字段相同，不是同一个字段只能分开始用

**2. 笛卡尔积：hive中慎用**

- 笛卡尔积没有关联条件，转化为mr的时候，所有数据都到一个redecetask中，性能很低

- 如果非要用笛卡尔积（如大表关联小表），额外加一个关联键(两个表都添加，而且关联规则要一致)，取决于到底使用几个reducetask

**3. 合理设计maptask数量**

- maptask过多，说明小文件过多，会造成container启动和销毁的时间远大于运行计算的时间，不划算

- maptask过少，造成map端的并行度不高，造成资源不合理利用

- 最好128M一个，数据不要过小

- 原始数据进行小文件合并

- 当mr之间存在串联，一定要控制上一个mr程序输出的reducetask个数

- jvm重用：一个container可以运行多个maptask(运行完毕一个task不销毁)

- 参数set mapred.job.reuse.jvm.num.tasks=5 设置一个container中运行多少个maptask任务（默认值1）

- 可以提升整体运行性能，本质上减少container开启和销毁的时间

**4. reducetask的合理设置**

- reducetask决定的是map端出来的数据的走向，reducetask越多，说明reduce端的并行度越高，整体性能越高

- 经验值：节点数\*0.95(100个节点，设置95个)，手动设置

- set hive.exec.reducer.max; 默认1009，经验值为0.95 \* 集群中datanode个数，这个经验值是上限，在达不到上限时，根据map端输出的数据量和业务实际需求去决定开启的reducetask数量

**5. in/exist尽量不要使用**

- 没有关联条件，导致数据最终全部到一个reducetask，性能低

- 使用left semi join：因为半连接右关联条件，可以设置多个reducetask，并行度提高

- hive擅长做有关联条件的关联，可以找到关联的key，并行度可以提高

**6. 能使用多重查询（插入）的时候不要使用单重查询（插入）**

- 单重：插入一次扫描一次基表

- insert into table 表名 select …… from 表名；

- 多重：扫描一次，插入多次；减少扫描基表的次数

- from 基表

insert into 表名1 select …

insert into 表名2 select …

**7. 合理利用分区**

- 什么时候需要分区：当表数据比较大，为了避免全表扫描，使用分区

- 作用：减少数据扫描范围

- 设计分区时，一般选择频繁过滤查询的字段作为分区字段

**8. 合理设计分桶**

- 当数据有join需求或抽样需求的时候，设计分桶

- 分桶类似于mr中分区的概念，最终结果是输出到不同文件

- 作用：

- 提升join的性能，大表关联大表的时候可以转成小表关联小表

- 提升抽样的性能，hash算法进行分桶，每个桶的数据是足够散列（分散均匀），可以直接用某个桶的数据作为抽样数据

- 抽样语法：tablesample (bucket x out of y)

- y: 分为几个桶簇（一组桶：一个或多个桶）

- x: 抽样时从哪一个桶簇开始抽样

- 例如分为3个桶和3个桶簇，每个桶簇包含1个桶 tablesample (bucket 1 out of 3)

- select \* from 表名 tablesample(1 out of bucket 3)

- 一般取样的时候，桶簇个数是桶的因子或倍数关系

**9. join优化：能使用map端的join，不使用reduce端的join**

- 总体原则：

- 先做过滤后再进行join操作，最大限度减少参与join的数据量

- 小表join后，最好启动mapjoin

- join on的条件相同的话，最好放入一个job,并且join表的排序顺序从小到大

- a表:100;b表:1000;c表:10000 join a on b on c

- 使用join语句时，应将条目少的表/子查询放在join操作符的左边（作为左表）

**10. group by优化**

- 建表时，尽量使用orc、parquet这些列式存储格式

- 因为列式存储的表，每列数据在物理上是存在一起的，hive查询时会只遍历需要的列数据，减少处理的数据量

**11. 本地模式执行mapreduce**

- 当hive查询处理数据量较少的时候，优先选择本地模式运行（不走集群），减少资源消耗，提升执行速度

- set hive.exec.mode.local.auto=true; 开启本地模式，默认是false

**12. 将小文件进行合并**

- 文件数过多会给HDFS带来压力，而且会影响处理效率，可以通过合并map和reduce的结果文件来降低影响：

- set hive.input.format=org.apache.hadoop.hive.ql.io.CombineHiveInputFormat; 执行map前进行小文件合并，常用

- set hive.merge.mapfiles=true; 在map only的任务结束时合并小文件

- set hive.merge.mapredfiles=true; 在mapreduce任务结束时合并小文件

- set hive.merge.size.per.task=256\*1000\*1000; 设置合并文件的大小（最大合并到256M，超过则会生成另一个文件）

- set mapred.max.split.size=256000000; 设置每个map最大的分割大小

- set mpared.min.split.size.per.node=1; 设置一个节点上切片的最小量

**九、hive中的数据倾斜**

**1. 什么是数据倾斜：数据分布不均匀**

- 底层执行引擎是MapReduce，hive数据倾斜实际上是Mapreduce的数据倾斜

- map端是逻辑分片，数据量小，倾斜小；

- reduce端针对所有map输出结果的，数据量大，如果并行度不高，容易产生数据倾斜

**2. 数据倾斜产生的原因归根结底是分区没有分号，也就是map端的key设计**

- 示例：

- 淘宝女性客户数量远大于男性客户，如果按性别分区，处理女性分区的mapreduce时间会远大于男性分区的时间，产生数据倾斜

- 电信流量数据按地区分区，北京vs西藏（北京人口稠密，西藏人口稀疏），数据也容易产生数据倾斜

- 大数据编程中，不怕数据量大，就怕数据倾斜，严重影响效率

- key的设置要做到每个分区的数据量相对均衡，有效减小数据倾斜

**3.哪些操作容易造成数据倾斜**

**（1）最容易造成数据倾斜的是join操作**

- 转换mr时候key的设置就是关联条件

**（2）场景一：网站采集日志信息的时候，join关联时有null值**

- 所有的null值最终会分到同一个分区中，null值特别多，会造成有null关联的那个分区数据特别多，产生数据倾斜

- select \* from user join log on user.id=log.userid

- 如何解决

- null值不参与关联，用的较多

- select \* from user join log on user.id=log.userid

and log.userid is not null 关联的时候去掉

union all select \* from log where userid is null; 将null值单独拼接进来，不丢失数据

- 给所有null值加一个随机数（用hive的内置函数）

- 可以散列null，不会都分到一个分区中，降低数据倾斜

**（3）场景二：关联的时候数据类型不统一**

- user: id int; log: userid string

- 默认的hash操作会按照小类型（int类型）的id进行分配，导致所有的string类型id被分到同一个reducer中

- 解决方案：把数字类型id转换成string类型id

- select \* from user a left outer join log b on b.userid=cast(a.userid as string)

**（4）场景三：大小表关联**

- map端join效率高，并行度高，不容易产生，所以使用mapjoin解决小表关联大表产生的数据倾斜

- set hive.auto.convert.join；map端关联是否开启，默认true开启。Hive.1x版本默认是false

- set hive.mapjoin.smalltable.filesize; 设置map端join的小表的大小，只要小表不超过这个值，默认都走map端join；如果最小表大于设置的值，默认走reduce端的join

- 默认25000000（25M）,可以随意设置，只要不超过内存容量

- 在进行关联操作的时候，两个表

- 大表和大表关联，切分大表，切在25M范围内，分别使用mapjoin

- 大表都拆成小表，转成小表关联小表：建表时建分桶表，两个表的分桶数量一样或倍数关系，分桶字段必须是关联字段

- 拆分其中一个大表，转成大表关联小表：

- 方式1：大表抽取数据，写sql，加过滤条件：select \* from a join (select … from b where <id<)

- 方式2：分区建表，分区字段为关联字段，在进行关联时拿每个分区分别进行关联

- 强制走mapjoin: select /\*+mapjoin(需要加到缓存中的表（a）)\*/ a.\*, b.\* from a join b on a.id=b.id;

- 只要小表的大小你的缓存可以承受就可以

**4. 如何判断程序发生数据倾斜**

- 日志中：reduce进度一直卡在某一个进度，耗时很长

**十、hive的执行原理**

**1. hive执行过程**

- 将sql语句进行编译成mr—>优化，去掉重复操作—>执行mr

- 编译过程是将hql转换成一组操作符（select、join、group by、load。。。），操作符是hive最小的处理单元

- 操作符有哪些

- tablescanOperator: 表扫描

- reducesinkOperator: 创建发送到reduce端的键值对

- joinOperator：关联两份数据

- selectOperator：选择输出列

- filesinkOperator: 建立结果数据，输出至文件

- filterOperator: 过滤输入数据

- groupbyOperator：group by语句

- mapjoinOperator: /\*+mapjoin(输入缓存的table)\*/

- limitOperator: limit语句

- unionOperator: union语句

**2. join操作**

-