# Hive

## hive定义

是一个数据仓库基础工具在Hadoop中用来处理结构化数据。它架构在[Hadoop](http://www.yiibai.com/hadoop/" \t "_blank)之上，总归为大数据，并使得查询和分析方便。并提供简单的sql查询功能，可以将sql语句转换为MapReduce任务进行运行。

术语“大数据”是大型数据集，其中包括体积庞大，高速，以及各种由与日俱增的数据的集合。使用传统的数据管理系统，它是难以加工大型数据。因此，Apache软件基金会推出了一款名为Hadoop的解决大数据管理和处理难题的框架。

Hadoop是一个开源框架来存储和处理大型数据在分布式环境中。它包含两个模块，一个是MapReduce，另外一个是Hadoop分布式文件系统（HDFS）。

区别传统文件系统：NTFS、ext4、HFS+分别是windows，Linux，mac os的日志式文件系统

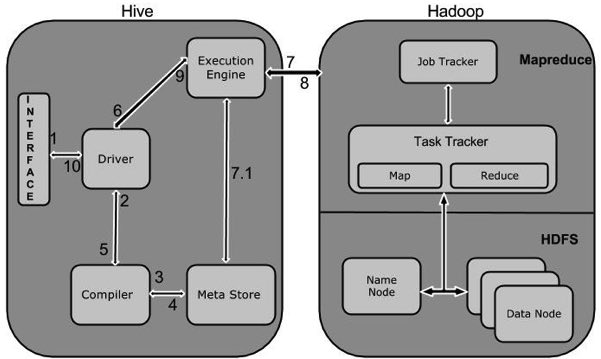
传统的文件系统是单机的，不能横跨不同的机器。HDFS（Hadoop Distributed FileSystem）的设计本质上是为了大量的数据能横跨成百上千台机器，但是你看到的是一个文件系统而不是很多文件系统。比如你说我要获取/hdfs/tmp/file1的数据，你引用的是一个文件路径，但是实际的数据存放在很多不同的机器上。你作为用户，不需要知道这些，就好比在单机上你不关心文件分散在什么磁道什么扇区一样。HDFS为你管理这些数据

有多种方法来执行MapReduce作业：

* 传统的方法是使用Java MapReduce程序结构化，半结构化和非结构化数据。
* 针对MapReduce的脚本的方式，使用Pig来处理结构化和半结构化数据。
* Hive查询语言（HiveQL或HQL）采用Hive为MapReduce的处理结构化数据。

Hive是一个数据仓库基础工具在Hadoop中用来处理结构化数据

## Hive和Hadoop框架的交互方式



下表定义Hive和Hadoop框架的交互方式：

|  |  |
| --- | --- |
| **Step No.** | **操作** |
| 1 | **Execute Query**  Hive接口，如命令行或Web UI发送查询驱动程序（任何数据库驱动程序，如JDBC，ODBC等）来执行。 |
| 2 | **Get Plan**  在驱动程序帮助下查询编译器，分析查询检查语法和查询计划或查询的要求。 |
| 3 | **Get Metadata**  编译器发送元数据请求到Metastore（任何数据库）。 |
| 4 | **Send Metadata**  Metastore发送元数据，以编译器的响应。 |
| 5 | **Send Plan**  编译器检查要求，并重新发送计划给驱动程序。到此为止，查询解析和编译完成。 |
| 6 | **Execute Plan**  驱动程序发送的执行计划到执行引擎。 |
| 7 | **Execute Job**  在内部，执行作业的过程是一个MapReduce工作。执行引擎发送作业给JobTracker，在名称节点并把它分配作业到TaskTracker，这是在数据节点。在这里，查询执行MapReduce工作。 |
| 7.1 | **Metadata Ops**  与此同时，在执行时，执行引擎可以通过Metastore执行元数据操作。 |
| 8 | **Fetch Result**  执行引擎接收来自数据节点的结果。 |
| 9 | **Send Results**  执行引擎发送这些结果值给驱动程序。 |
| 10 | **Send Results**  驱动程序将结果发送给Hive接口。 |

-> Hive是基于Hadoop的一个数据仓库工具；它是MapReduce的一个封装，底层就是MapReduce程序；

-> Hive可以将结构化的数据文件（eg：按照各字段分类的数据）映射成一张虚表，并提供类SQL查询功能；

-> 有了Hive后我们就不用再写麻烦的MapReduce程序了

3. 元数据库Meta store

（1）Meta store是Hive数据库中的一个库，用于存储处理数据的元数据，包括：

表名、表所属的数据库（默认是default）、表的拥有者、列/分区字段、表的类型（是否是外部表）、表的数据所在目录等。

（2）Meta store默认存储在Hive自带的derby数据库中，但因为derby无法实现并发，实际工作中一般将derby替换为mysql；

（3）Hive中处理的数据分两部分存放：

① 处理的结构化数据，存储在HDFS中

② 表的元数据存储在元数据库mysql中

四、Hive和数据库的区别

（1）数据存储位置不同

Hive中处理的结构化数据存储在HDFS中，元数据存储在mysql的Meta store中； 数据库将数据保存在块设备或本地文件系统中；

（2）数据更新

Hive是针对数据仓库设计的，主要用于读，所有的数据在加载时已经确定好，适合处理静态数据；

传统sql数据库通常是实时进行修改的，增删改查，适合处理动态数据；

（3）执行机制

Hive大多数查询的执行是通过Hadoop提供的MapReduce实现的；

数据库通常是用自己的执行引擎引擎innodb；注：MySQL早期版本执行引擎是MyIsam。5.6以前默认的执行引擎是innodb。 MySQL8.0彻底道别MyIsam

（4）执行延迟

Hive因为没有索引、利用MapReduce框架执行查询，所以Hive本身的延迟较高；

传统sql数据库的延迟较低，但是不太适合处理PB级别以上海量数据；

处理海量数据时，Hive的优势就显出来了

## Hive常见的应用场景

(1)日志分析：大部分互联网公司使用hive进行日志分析，包括百度、淘宝等。

(2)海量结构化数据离线分析

Hive适合的是什么场景呢？数据仓库。**基于Hadoop做一些数据清洗（ETL）、报表、数据分析**。本质上来说，它还是一个面向读的、面向分析的SQL工具

原文链接：https://www.yiibai.com/hive/

## hive元数据

hive使用MySQL当作元数据库，MySQL这个元数据库，保存的是hive表的描述信息，描述了有哪些database，table以及表有多少列，每一列是什么类型，还有描述表的数据保存在HDFS的什么位置。

hive是一个数据仓库工具（存储数据并分析数据，分析数据仓库中的数据量很大，一般要分析很长的时间）

*Hive——元数据表含义*

1、VERSION   -- 查询版本信息

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Field** | **Type** | **Comment** |
|  | VER\_ID | bigint(20) | ID主键 |
|  | SCHEMA\_VERSION | varchar(127) | Hive版本 |
|  | VERSION\_COMMENT | varchar(255) | 版本说明 |

2、DBS　　　　 -- 存储Hive中所有数据库的基本信息

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Field** | **Type** | **Comment** |
|  | DB\_ID | bigint(20) | 数据库ID |
|  | DESC | varchar(4000) | 数据库描述 |
|  | DB\_LOCATION\_URI | varchar(4000) | 数据库HDFS路径 |
|  | NAME | varchar(128) | 数据库名 |
|  | OWNER\_NAME | varchar(128) | 数据库所有者用户名 |
|  | OWNER\_TYPE | varchar(10) | 所有者角色 |

3、DATABASE\_PARAMS　　--该表存储数据库的相关参数，在CREATE DATABASE时候用

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Field** | **Type** | **Comment** |
|  | DB\_ID | bigint(20) | 数据库ID |
|  | PARAM\_KEY | varchar(180) | 参数名 |
|  | PARAM\_VALUE | varchar(4000) | 参数值 |

4、TBLS　　--该表中存储Hive表、视图、索引表的基本信息

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Field** | **Type** | **Comment** |
|  | TBL\_ID | bigint(20) | 表ID |
|  | CREATE\_TIME | int(11) | 创建时间 |
|  | DB\_ID | bigint(20) | 数据库ID |
|  | LAST\_ACCESS\_TIME | int(11) | 上次访问时间 |
|  | OWNER | varchar(767) | 所有者 |
|  | RETENTION | int(11) | 保留字段 |
|  | SD\_ID | bigint(20) | 序列化配置信息 |
|  | TBL\_NAME | varchar(128) | 表名 |
|  | TBL\_TYPE | varchar(128) | 表类型 |
|  | VIEW\_EXPANDED\_TEXT | mediumtext | 视图的详细HQL语句 |
|  | VIEW\_ORIGINAL\_TEXT | mediumtext | 视图的原始HQL语句 |

5、TABLE\_PARAMS　　--该表存储表/视图的属性信息。

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Field** | **Type** | **Comment** |
|  | TBL\_ID | bigint(20) | 表ID |
|  | PARAM\_KEY | varchar(256) | 属性名 |
|  | PARAM\_VALUE | varchar(4000) | 属性值 |

6、SDS　　--该表保存文件存储的基本信息，如INPUT\_FORMAT、OUTPUT\_FORMAT、是否压缩等。

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Field** | **Type** | **Comment** |
|  | SD\_ID | bigint(20) | 存储信息ID |
|  | CD\_ID | bigint(20) | 字段信息ID |
|  | INPUT\_FORMAT | varchar(4000) | 文件输入格式 |
|  | IS\_COMPRESSED | bit(1) | 是否压缩 |
|  | IS\_STOREDASSUBDIRECTORIES | bit(1) | 是否以子目录存储 |
|  | LOCATION | varchar(4000) | HDFS路径 |
|  | NUM\_BUCKETS | int(11) | 分桶数量 |
|  | OUTPUT\_FORMAT | varchar(4000) | 文件输出格式 |
|  | SERDE\_ID | bigint(20) | 序列化类ID |

7、SERDS      --该表存储序列化使用的类信息

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Field** | **Type** | **Comment** |
|  | SERDE\_ID | bigint(20) | 序列化类配置ID |
|  | NAME | varchar(128) | 序列化类别名 |
|  | SLIB | varchar(4000) | 序列化类 |

8、SERDE\_PARAMS      --该表存储序列化的一些属性、格式信息,比如：行、列分隔符

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Field** | **Type** | **Comment** |
|  | SERDE\_ID | bigint(20) | 序列化类配置ID |
|  | PARAM\_KEY | varchar(256) | 属性名 |
|  | PARAM\_VALUE | varchar(4000) | 属性值 |

9、COLUMNS\_V2      --该表存储表对应的字段信息。

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Field** | **Type** | **Comment** |
|  | CD\_ID | bigint(20) | 字段信息ID |
|  | COMMENT | varchar(256) | 字段注释 |
|  | COLUMN\_NAME | varchar(128) | 字段名 |
|  | TYPE\_NAME | varchar(4000) | 字段类型 |
|  | INTEGER\_IDX | int(11) | 字段顺序 |

10、PARTITIONS      --该表存储表分区的基本信息。

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Field** | **Type** | **Comment** |
|  | PART\_ID | bigint(20) | 分区ID |
|  | CREATE\_TIME | int(11) | 分区创建时间 |
|  | LAST\_ACCESS\_TIME | int(11) | 最后一次访问时间 |
|  | PART\_NAME | varchar(767) | 分区名 |
|  | SD\_ID | bigint(20) | 分区存储ID |
|  | TBL\_ID | bigint(20) | 表ID |

11、PARTITION\_KEYS      --该表存储分区的字段信息。

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Field** | **Type** | **Comment** |
|  | TBL\_ID | bigint(20) | 表ID |
|  | PKEY\_COMMENT | varchar(4000) | 分区字段说明 |
|  | PKEY\_NAME | varchar(128) | 分区字段名 |
|  | PKEY\_TYPE | varchar(767) | 分区字段类型 |
|  | INTEGER\_IDX | int(11) | 分区字段顺序 |

12、PARTITION\_KEY\_VALS      --该表存储分区字段值。

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Field** | **Type** | **Comment** |
|  | PART\_ID | bigint(20) | 分区ID |
|  | PART\_KEY\_VAL | varchar(255) | 分区字段值 |
|  | INTEGER\_IDX | int(11) | 分区字段值顺序 |

13、PARTITION\_PARAMS      --该表存储分区的属性信息。

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Field** | **Type** | **Comment** |
|  | PART\_ID | bigint(20) | 分区ID |
|  | PARAM\_KEY | varchar(256) | 分区属性名 |
|  | PARAM\_VALUE | varchar(4000) | 分区属性值 |

其他不常用的元数据表：

DB\_PRIVS：数据库权限信息表。通过GRANT语句对数据库授权后，将会在这里存储。

IDXS：索引表，存储Hive索引相关的元数据

INDEX\_PARAMS：索引相关的属性信息。

TAB\_COL\_STATS：表字段的统计信息。使用ANALYZE语句对表字段分析后记录在这里。

TBL\_COL\_PRIVS：表字段的授权信息

PART\_PRIVS：分区的授权信息

PART\_COL\_STATS：分区字段的统计信息。

PART\_COL\_PRIVS：分区字段的权限信息。

FUNCS：用户注册的函数信息

FUNC\_RU：用户注册函数的资源信息

# MapReduce, Hadoop, Hive

大数据本身是个很宽泛的概念，Hadoop生态圈（或者泛生态圈）基本上都是为了处理超过单机尺度的数据处理而诞生的。你可以把它比作一个厨房所以需要的各种工具。锅碗瓢盆，各有各的用处，互相之间又有重合。你可以用汤锅直接当碗吃饭喝汤，你可以用小刀或者刨子去皮。但是每个工具有自己的特性，虽然奇怪的组合也能工作，但是未必是最佳选择。

大数据，首先你要能存的下大数据。

传统的文件系统是单机的，不能横跨不同的机器。HDFS（Hadoop Distributed FileSystem）的设计本质上是为了大量的数据能横跨成百上千台机器，但是你看到的是一个文件系统而不是很多文件系统。比如你说我要获取/hdfs/tmp/file1的数据，你引用的是一个文件路径，但是实际的数据存放在很多不同的机器上。你作为用户，不需要知道这些，就好比在单机上你不关心文件分散在什么磁道什么扇区一样。HDFS为你管理这些数据。

存的下数据之后，你就开始考虑怎么处理数据。虽然HDFS可以为你整体管理不同机器上的数据，但是这些数据太大了。一台机器读取成T上P的数据（很大的数据，比如整个东京热有史以来所有高清电影的大小甚至更大），一台机器慢慢跑也许需要好几天甚至好几周。对于很多公司来说，单机处理是不可忍受的，比如微博要更新24小时热博，它必须在24小时之内跑完这些处理。那么我如果要用很多台机器处理，我就面临了如何分配工作，如果一台机器挂了如何重新启动相应的任务，机器之间如何互相通信交换数据以完成复杂的计算等等。这就是MapReduce / Tez / Spark的功能。MapReduce是第一代计算引擎，Tez和Spark是第二代。MapReduce的设计，采用了很简化的计算模型，只有Map和Reduce两个计算过程（中间用Shuffle串联），用这个模型，已经可以处理大数据领域很大一部分问题了。

那什么是Map什么是Reduce？

MapReduce是一种编程模型，用于大规模数据集（大于1TB）的并行运算。概念"Map（映射）"和"Reduce（归约）

考虑如果你要统计一个巨大的文本文件存储在类似HDFS上，你想要知道这个文本里各个词的出现频率。你启动了一个MapReduce程序。Map阶段，几百台机器同时读取这个文件的各个部分，分别把各自读到的部分分别统计出词频，产生类似

（hello, 12100次），（world，15214次）等等这样的Pair（我这里把Map和Combine放在一起说以便简化）；这几百台机器各自都产生了如上的集合，然后又有几百台机器启动Reduce处理。Reducer机器A将从Mapper机器收到所有以A开头的统计结果，机器B将收到B开头的词汇统计结果（当然实际上不会真的以字母开头做依据，而是用函数产生Hash值以避免数据串化。因为类似X开头的词肯定比其他要少得多，而你不希望数据处理各个机器的工作量相差悬殊）。然后这些Reducer将再次汇总，（hello，12100）＋（hello，12311）＋（hello，345881）= （hello，370292）。每个Reducer都如上处理，你就得到了整个文件的词频结果。

这看似是个很简单的模型，但很多算法都可以用这个模型描述了。

Map＋Reduce的简单模型很暴力，虽然好用，但是很笨重。第二代的Tez和Spark除了内存Cache之类的新feature，本质上来说，是让Map/Reduce模型更通用，让Map和Reduce之间的界限更模糊，数据交换更灵活，更少的磁盘读写，以便更方便地描述复杂算法，取得更高的吞吐量。

有了MapReduce，Tez和Spark之后，程序员发现，MapReduce的程序写起来真麻烦。他们希望简化这个过程。这就好比你有了汇编语言，虽然你几乎什么都能干了，但是你还是觉得繁琐。你希望有个更高层更抽象的语言层来描述算法和数据处理流程。于是就有了Pig和Hive。Pig是接近脚本方式去描述MapReduce，Hive则用的是SQL。它们把脚本和SQL语言翻译成MapReduce程序，丢给计算引擎去计算，而你就从繁琐的MapReduce程序中解脱出来，用更简单更直观的语言去写程序了。

有了Hive之后，人们发现SQL对比Java有巨大的优势。一个是它太容易写了。刚才词频的东西，用SQL描述就只有一两行，MapReduce写起来大约要几十上百行。而更重要的是，非计算机背景的用户终于感受到了爱：我也会写SQL！于是数据分析人员终于从乞求工程师帮忙的窘境解脱出来，工程师也从写奇怪的一次性的处理程序中解脱出来。大家都开心了。Hive逐渐成长成了大数据仓库的核心组件。甚至很多公司的流水线作业集完全是用SQL描述，因为易写易改，一看就懂，容易维护。

自从数据分析人员开始用Hive分析数据之后，它们发现，Hive在MapReduce上跑，真慢！（流水线作业集也许没啥关系，比如24小时更新的推荐，反正24小时内跑完就算了。但是数据分析，人们总是希望能跑更快一些。比如我希望看过去一个小时内多少人在充气娃娃页面驻足，分别停留了多久，对于一个巨型网站海量数据下，这个处理过程也许要花几十分钟甚至很多小时。而这个分析也许只是你万里长征的第一步，你还要看多少人浏览了跳蛋多少人看了拉赫曼尼诺夫的CD，以便跟老板汇报，我们的用户是猥琐男闷骚女更多还是文艺青年／少女更多。你无法忍受等待的折磨。

于是Impala，Presto，Drill诞生了（当然还有无数非著名的交互SQL引擎，就不一一列举了）。三个系统的核心理念是，MapReduce引擎太慢，因为它太通用，太强壮，太保守，我们SQL需要更轻量，更激进地获取资源，更专门地对SQL做优化，而且不需要那么多容错性保证（因为系统出错了大不了重新启动任务，如果整个处理时间更短的话，比如几分钟之内）。这些系统让用户更快速地处理SQL任务，牺牲了通用性稳定性等特性。如果说MapReduce是大砍刀，砍啥都不怕，那上面三个就是剔骨刀，灵巧锋利，但是不能搞太大太硬的东西。

Impala，Presto，Drill这些计算引擎系统，说实话，一直没有达到人们期望的流行度。因为这时候又两个异类被造出来了。他们是Hive on Tez / Spark和SparkSQL。它们的设计理念是，MapReduce慢，但是如果我用新一代通用计算引擎Tez或者Spark来跑SQL，那我就能跑的更快。而且用户不需要维护两套系统。这就好比如果你厨房小，人又懒，对吃的精细程度要求有限，那你可以买个电饭煲，能蒸能煲能烧，省了好多厨具。

上面的介绍，基本就是一个数据仓库的构架了。底层HDFS，上面跑MapReduce／Tez／Spark，在上面跑Hive，Pig。或者HDFS上直接跑Impala，Drill，Presto。这解决了中低速数据处理的要求。

那如果我要更高速的处理呢？

如果我是一个类似微博的公司，我希望显示不是24小时热博，我想看一个不断变化的热播榜，更新延迟在一分钟之内，上面的手段都将无法胜任。于是又一种计算模型被开发出来，这就是Streaming（流）计算。Storm是最流行的流计算平台。流计算的思路是，如果要达到更实时的更新，我何不在数据流进来的时候就处理了？比如还是词频统计的例子，我的数据流是一个一个的词，我就让他们一边流过我就一边开始统计了。流计算很牛逼，基本无延迟，但是它的短处是，不灵活，你想要统计的东西必须预先知道，毕竟数据流过就没了，你没算的东西就无法补算了。因此它是个很好的东西，但是无法替代上面数据仓库和批处理系统。

还有一个有些独立的模块是KV Store，比如Cassandra，HBase，MongoDB以及很多很多其他的（多到无法想象）。所以KV Store就是说，我有一堆键值，我能很快速滴获取与这个Key绑定的数据。比如我用身份证号，能取到你的身份数据。这个动作用MapReduce也能完成，但是很可能要扫描整个数据集。而KV Store专用来处理这个操作，所有存和取都专门为此优化了。从几个P的数据中查找一个身份证号，也许只要零点几秒。这让大数据公司的一些专门操作被大大优化了。比如我网页上有个根据订单号查找订单内容的页面，而整个网站的订单数量无法单机数据库存储，我就会考虑用KV Store来存。KV Store的理念是，基本无法处理复杂的计算，大多没法JOIN，也许没法聚合，没有强一致性保证（不同数据分布在不同机器上，你每次读取也许会读到不同的结果，也无法处理类似银行转账那样的强一致性要求的操作）。但是它就是快。极快。

每个不同的KV Store设计都有不同取舍，有些更快，有些容量更高，有些可以支持更复杂的操作。必有一款适合你。

除此之外，还有一些更特制的系统／组件，比如Mahout是分布式机器学习库，Protobuf是数据交换的编码和库，ZooKeeper是高一致性的分布存取协同系统，等等。

有了这么多乱七八糟的工具，都在同一个集群上运转，大家需要互相尊重有序工作。所以另外一个重要组件是，调度系统。现在最流行的是Yarn。你可以把他看作中央管理，好比你妈在厨房监工，哎，你妹妹切菜切完了，你可以把刀拿去杀鸡了。只要大家都服从你妈分配，那大家都能愉快滴烧菜。

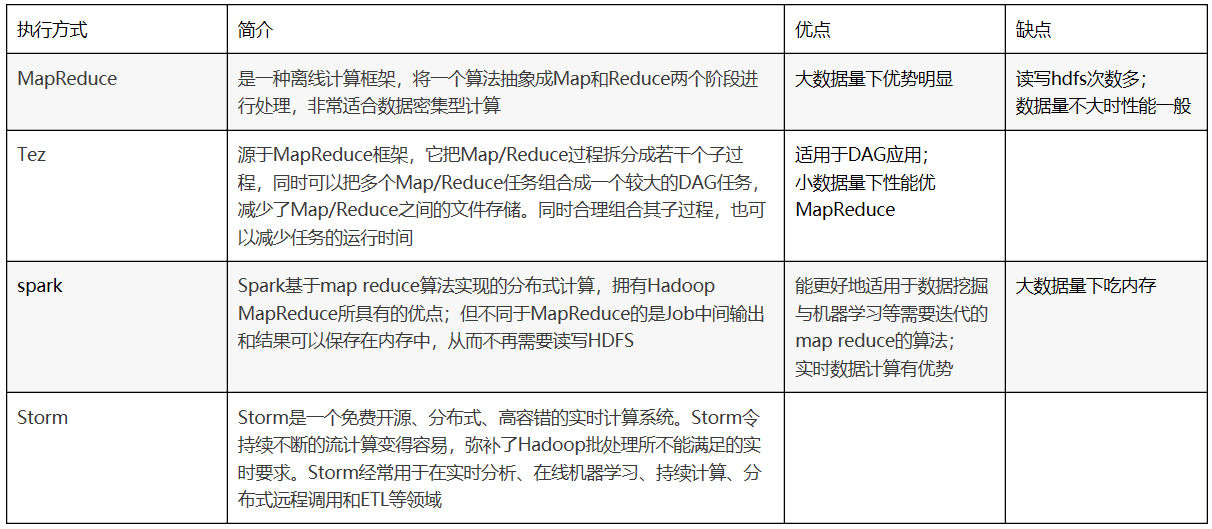
你可以认为，大数据生态圈就是一个厨房工具生态圈。为了做不同的菜，中国菜，日本菜，法国菜，你需要各种不同的工具。而且客人的需求正在复杂化，你的厨具不断被发明，也没有一个万用的厨具可以处理所有情况，因此它会变的越来越复杂。

附：Hadoop是基础，其中的HDFS提供文件存储，Yarn进行资源管理。在这上面可以运行MapReduce、Spark、Tez等计算框架。

MapReduce:是一种离线计算框架，将一个算法抽象成Map和Reduce两个阶段进行处理，非常适合数据密集型计算。

Spark:Spark是UC Berkeley AMP lab所开源的类Hadoop MapReduce的通用的并行计算框架，Spark基于map reduce算法实现的分布式计算，拥有Hadoop MapReduce所具有的优点；但不同于MapReduce的是Job中间输出和结果可以保存在内存中，从而不再需要读写HDFS，因此Spark能更好地适用于数据挖掘与机器学习等需要迭代的map reduce的算法。

# Mapre，Tez, spark, storm 计算框架对比



## MapReduce

|  |  |  |  |
| --- | --- | --- | --- |
| MapReduce | 是一种离线计算框架，将一个算法抽象成Map和Reduce两个阶段进行处理，非常适合数据密集型计算 | 大数据量下优势明显 | 读写hdfs次数多； 数据量不大时性能一般 |

## Tez

源于MapReduce框架，它把Ｍap/Reduce过程拆分成若干个子过程，同时可以把多个Ｍap/Reduce任务组合成一个较大的DAG任务，减少了Ｍap/Reduce之间的文件存储。同时合理组合其子过程，也可以减少任务的运行时间。适用于DAG应用；小数据量下性能优MapReduce。

## spark

Spark基于map reduce算法实现的分布式计算，拥有Hadoop MapReduce所具有的优点；但不同于MapReduce的是Job中间输出和结果可以保存在内存中，从而不再需要读写HDFS 能更好地适用于数据挖掘与机器学习等需要迭代的map reduce的算法；实时数据计算有优势。大数据量下吃内存。

## Storm

|  |  |
| --- | --- |
| Storm | Storm是一个免费开源、分布式、高容错的实时计算系统。Storm令持续不断的流计算变得容易，弥补了Hadoop批处理所不能满足的实时要求。Storm经常用于在实时分析、在线机器学习、持续计算、分布式远程调用和ETL等领域 |

————————————————

版权声明：本文为CSDN博主「ronaldo\_liu2018」的原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/sinat\_17697111/article/details/88851929

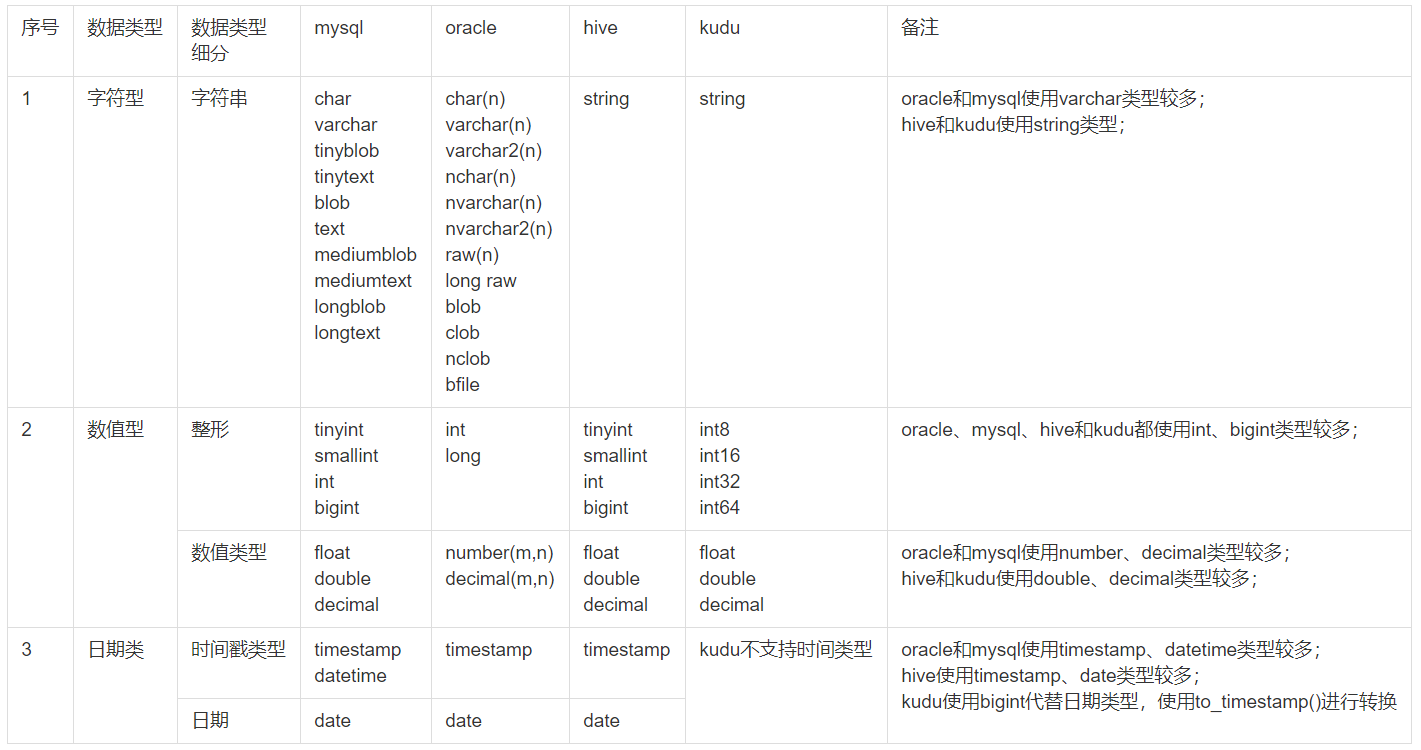
————————————————

版权声明：本文为CSDN博主「刘喵酱」的原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接及本声明。

原文链接：<https://blog.csdn.net/qq_36617639/article/details/80324982>

# HiveQL

## hive,mysql,oracle数据类型对比



# hive,mysql对比

两者的比较：

Hive采用了类SQL的查询语言HQL（hive query language）。除了HQL之外，其余无任何相似的地方。Hive是为了数据仓库设计的。

存储位置：Hive在Hadoop上hdfs文件系统里；Mysql将数据存储在设备或本地系统中；

数据更新：Hive不支持数据的改写和添加，是在加载的时候就已经确定好了；Mysql数据库可以CRUD；

索引：Hive无索引，每次扫描所有数据，底层是MR，并行计算，适用于大数据量；MySQL有索引，适合在线查询数据；

执行：Hive底层是MarReduce；MySQL底层是执行引擎；

可扩展性：Hive：大数据量，慢慢扩；MySQL:相对就很少了

# 对比sql和hive hql区别：

Hive-sql与SQL基本上一样，因为当初的设计目的，就是让会SQL不会编程MapReduce的也能使用Hadoop进行处理数据。大胆使用SQL的，如果遇到不对的，再查。

## 用法上的区别

1. HQL不支持行级别的增、改、删，所有数据在加载时就已经确定，不可更改。

2. 不支持事务。

3. 支持分区存储。

## hive下的SQL特点

1.不支持等值连接，一般使用left join、right join 或者inner join替代。

2.不能智能识别concat(‘;’,key)，只会将‘；’当做SQL结束符号。

 3.不支持INSERT INTO 表 Values（）, UPDATE, DELETE等操作

 4.HiveQL中String类型的字段若是空(empty)字符串, 即长度为0, 那么对它进行IS NULL的判断结果是False，使用left join可以进行筛选行。

 5.不支持 ‘< dt <’这种格式的范围查找，可以用dt in(”,”)或者between替代。

作者：MichalLiu  
链接：https://www.jianshu.com/p/0696a52226bb  
来源：简书  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

## 执行顺序差异

### msyql语句执行顺序

代码写的顺序：

select ... from... where.... group by... having... order by..

代码的执行顺序：

from... where...group by... having.... select ... order by...

### hive 语句执行顺序

大致顺序

from... where.... select...group by... having ... order by...

(HAVING语句通常与GROUP BY语句联合使用，用来过滤由GROUP BY语句返回的记录集。

HAVING语句的存在弥补了WHERE关键字不能与聚合函数联合使用的不足。

语法：

SELECT column1, column2, ... column\_n, aggregate\_function (expression)FROM tablesWHERE predicatesGROUP BY column1, column2, ... column\_nHAVING condition1 ... condition\_n;

同样使用本文中的学生表格，如果想查询平均分高于80分的学生记录可以这样写：

SELECT id, COUNT(course) as numcourse, AVG(score) as avgscore

FROM student

GROUP BY id

HAVING AVG(score)>=80;

在这里，如果用WHERE代替HAVING就会出错)

## 功能函数用法差异

# 附录

## Apache软件基金会

支持开源软件项目的非盈利性组织

### 主要成果：

Subversion

Subversion已经正式由Apache接管。

[**ActiveMQ**](https://baike.baidu.com/item/ActiveMQ)

免费开源由java编写符合JMS1.1标准的消息

[](https://baike.baidu.com/pic/apache/8512995/0/7d9893105f5c3138203f2e4b?fr=lemma&ct=single)ActiveMq管理界面

中间件。

另外，它也支持通过除java语言外的语言的使用

**Ant**

这个太出名了。标准的批处理工具。是一套基于java的程序打造工具

**Commons**

一些常用的工具类库，包括common-pool,dbcp,[fileupload](https://baike.baidu.com/item/fileupload),Common-beans等。

**Excalibur**

它的主要产品是一个由java写成的，名字叫做Fortress（要塞） 的轻量级的可嵌入式反向控制容器。

[**iBATIS**](https://baike.baidu.com/item/iBATIS)

并入的一个项目，是ORM的一个很流行的工具

[**Geronimo**](https://baike.baidu.com/item/Geronimo)

是Apache软件基金会为了创造一个兼容j2ee的容器，而整理出来的一个新成果

[**Jakarta**](https://baike.baidu.com/item/Jakarta)

许多Java子项目的集成，[tomcat](https://baike.baidu.com/item/tomcat" \t "_blank),[ant](https://baike.baidu.com/item/ant)等就是从这里孵化出去的。

**James**

是一套用java开发的邮件、[新闻组](https://baike.baidu.com/item/%E6%96%B0%E9%97%BB%E7%BB%84" \t "_blank)、[消息服务器](https://baike.baidu.com/item/%E6%B6%88%E6%81%AF%E6%9C%8D%E5%8A%A1%E5%99%A8)。它使用avalon的组件框架。目前支持 SMTP, POP3 和 NNTP 很快也会支持 IMAP

**Logging**

基于java的可靠，快速，扩展性强的日志工具

[**Maven**](https://baike.baidu.com/item/Maven)

是一套java开发的工程综合管理工具。它基于工程对象模型(POM)的理念

**Portals**

门户产品

[**Struts**](https://baike.baidu.com/item/Struts)

一套通过servlets和jsp来搭建web应用的MVC框架

[**Tomcat**](https://baike.baidu.com/item/Tomcat)　用量最大的免费的Java服务器

[**Hadoop**](https://baike.baidu.com/item/Hadoop)[1-2]

现今最为著名的大数据神器，包括分布式存储和计算框架。

# MySQL执行引擎对比与SQL优化

## 下面对比InnoDB和MyISAM的区别：

1.InnoDB：

优点：InnoDB是一个事务型的存储引擎，有行级锁定和外键约束，提供了对数据库ACID事务的支持，并且实现了SQL标准的四种隔离级别，设计目标是处理大容量数据库系统；

缺点：不支持全文索引，而且它没有保存表的行数，当SELECT COUNT(\*) FROM TABLE时需要扫描全表；

适用场景：经常更新的表，适合处理多重并发的更新请求，需要事务、外键；

插播下MySQL的4个事务隔离级别：

隔离级别 读数据一致性 脏读 不可重复读 幻读

未提交读(read uncommitted) 最低级别，不读物理上顺坏的数据 是 是 是

已提交读(read committed) 语句级 是 是 是

可重复读(Repeatable red) 事务级 是 否 是

可序列化(Serializable) 最高级别，事务级 否 否 否

 查看数据库默认的隔离级别（我的是8.0版本）：

show global variables WHERE variable\_name='transaction\_isolation'

2. MyISAM：

优点：支持全文类型索引，索引和记录分开存储，并存储了表的行数，所以select count(\*)效率很高（不加where）；

缺点：不支持数据库事务，更新操作需要锁定整个表，不支持行级锁和外键；

适用场景：经常读取数据的场合，更新操作少；

## SQL优化：

语句优化，要尽量避免全表扫描，首先应考虑在 where 及 order by 涉及的列上建立索引；

语句优化，用select field\_1,field\_2,field\_3... form table代替select \* from table；

语句优化，使用连接（JOIN）代替子查询；

语句优化，使用联合（UNION）代替手动创建的临时表；

建表优化，尽量减少字段宽度，使用ENUM存储固定数据，例如性别；

语句优化，避免在 where 子句中对字段进行 null 值判断，否则将导致引擎放弃使用索引而进行全表扫描；

建表优化，字段选用优先级，整形>date,time>enum,char>varchar>blob,text；

语句优化，尽量避免在 where 子句中使用 != 或 <> 操作符，否则将引擎放弃使用索引而进行全表扫描；

语句优化，尽量避免在 where 子句中使用in 和 not in，否则会导致全表扫描，连续的数值用between代替，以及使用exists；

语句优化，尽量使用全文检索代替like查询；

语句优化，Update语句尽量只set需要修改的字段，否则带来没必要的开销；

语句优化，数据量大时（1万以上）尽量避免使用，游标；

持续更新中。。。

原文链接：https://blog.csdn.net/qq\_38258310/article/details/94468604