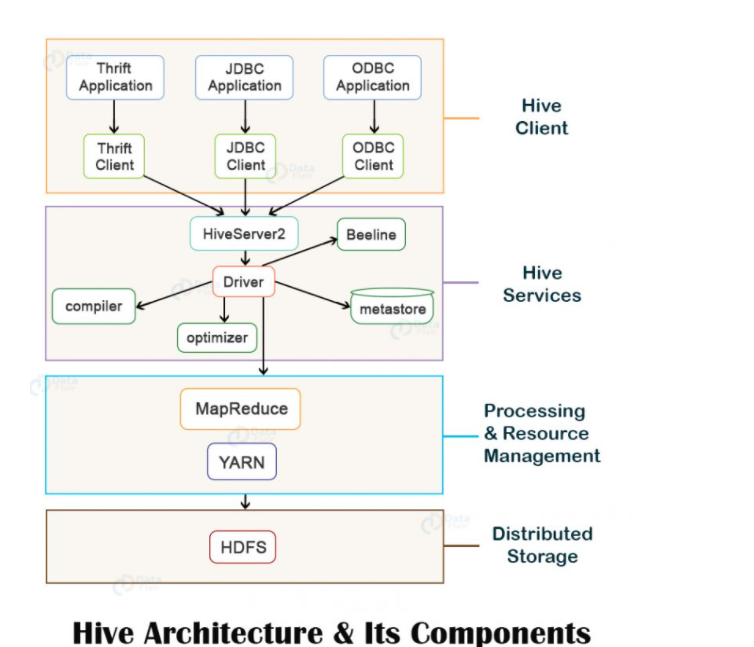
# hive

##### hive的架构



上图显示了Hive的体系结构及其主要组件。Hive的主要组件是：

1. Hive Client
2. Hive Services
3. Processing and Resource Management
4. Distributed Storage

### Hive的客户端

Hive支持使用JDBC，ODBC和Thrift驱动程序以任何语言（如Python，Java，C ++，Ruby等）编写的应用程序，以在Hive上执行查询。因此，可以轻松地以自己选择的任何语言编写hive客户端应用程序。

### Hive Services

为了执行所有查询，Hive提供了各种服务，例如Hive server2，Beeline等。

### Hive Driver

Hive Driver 接收用户通过shell提交的Hql语句，为查询创建session handles，并将查询结果发送到编译器。

### Hive Compiler

Hive编译器解析查询语句，使用存储在metastore中的元数据对不同的查询块和查询表达式进行语义分析和类型检查，并生成执行计划

编译器创建的执行计划是DAG，其中每个阶段都是 map/reduce 任务、对Hdfs的操作、元数据操作。

### Optimizer

优化器，Optimizer对执行计划进行转换操作，并拆分map/reduce任务来提高效率和可伸缩性。

### Execution Engine

执行引起在编译和优化步骤完成之后，使用hadoop按照依赖关系的顺序执行由编译器创建的执行计划。

### Metastore

Metastore是一个中央存储库，用于存储有关表和分区结构的元数据信息，包括列和列类型信息。

它还存储读/写操作所需的串行器和解串器信息，以及存储数据的HDFS文件。该元存储库通常是一个关系数据库。

Metastore提供了一个Thrift接口，用于查询和操作Hive元数据。

### HCatalog

HCatalog是Hadoop的表和存储管理层。它使使用不同数据处理工具（例如Pig，MapReduce等）的用户可以轻松地在网格上读取和写入数据。

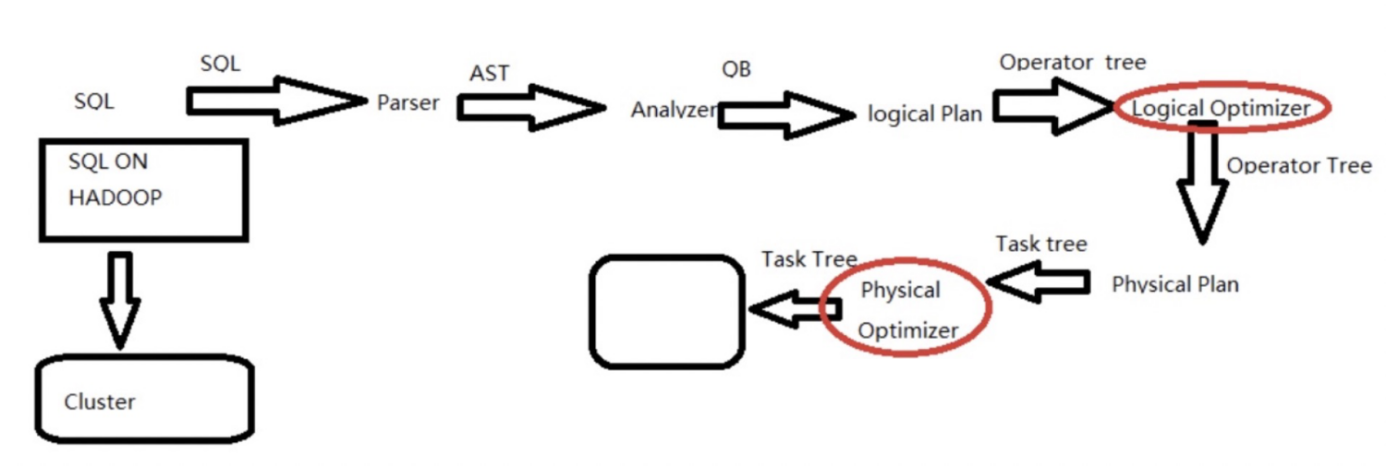
它建立在Hive Metastore的顶部，并将Hive Metastore的表格数据公开给其他数据处理工具。

### WebHCat

WebHCat是用于HCatalog的REST API。它是执行Hive元数据操作的HTTP接口。它为用户提供了运行Hadoop MapReduce（或YARN），Pig，Hive作业的服务。

##### hql执行流程和执行计划

hql执行流程



1.Parser：将sql解析为AST（抽象语法树），会进行语法校验，AST本质还是字符串

2.Analyzer：语法解析，生成QB（query block）

3.Logicl Plan：逻辑执行计划解析，生成一堆Opertator Tree

4.Logical optimizer:进行逻辑执行计划优化，生成一堆优化后的Opertator Tree

5.Phsical plan：物理执行计划解析，生成tasktree

6.Phsical Optimizer：进行物理执行计划优化，生成优化后的tasktree，该任务即是集群上的执行的作业

hql执行计划

通过explain参数开头执行hql查看计划结果，执行计划输出有三个部分：

·查询的抽象语法树

·计划不同阶段之间的依赖关系。

·每个阶段的描述

##### hive的优化

### 选用适当的文件格式

为了提高查询性能，ORC文件格式最合适，能够大幅度提升我们的查询性能。orc能够将原始数据大小最多减少 75%。因此，数据处理速度也会大幅度增加。与Text, Sequence 和 RC 格式相比，ORC显现出更好的性能。因此可以说，Hive在处理ORC格式数据时，可以提高性能。

### Hive分区

通过分区，数据集的各个列的所有条目被隔离并存储在各自的分区中，因此在编写查询以及从表中获取值的同时，仅需要查询所需要的分区，合理的使用分区能够减少查询结果集所产生的时间。

### 向量化

为了提高操作性能，可以使用向量化查询执行。（select、group by、where、join等操作）通过一次以1024行批量执行，而不是每次执行单个行来实现的。

向量化在HIve的0.13中引入，大大缩减了查询执行时间。并且可以通过两个参数轻松启用：

set hive.vectorized.execution = true

set hive.vectorized.execution.enabled = true

Hive 优化文章推荐：https://data-flair.training/blogs/hive-optimization-techniques/

##### hive的数据倾斜、及解决办法

**什么是数据倾斜**

由于数据分布不均匀，造成数据大量集中在一点，造成数据热点

**数据倾斜的现象**

在执行任务的时候，任务进度长时间维持在99%左右，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。数据倾斜的场景

1)、key分布不均匀

2)、业务数据本身的特性

3)、建表时考虑不周

4)、某些SQL语句本身就有数据倾斜

**数据倾斜的解决方案**

**Map端聚合**

--Map 端部分聚合，相当于Combiner

hive.map.aggr = true；

--有数据倾斜的时候进行负载均衡

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**

* 采用合理的Join
* 关于驱动表的取值，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 大量相同的特殊值
* 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。特殊情况处理

在业务逻辑优化效果的不大情况下，一些时候是可以将倾斜的数据单独拿出来处理。最后union回去

**典型的业务场景中出现数据倾斜**

**（1）空值产生的数据倾斜**

如日志中，常有信息丢失的问题，例如日志中的user\_id,如果取其中的user\_id和用户表中的user\_id关联，就会遇到数据倾斜的问题

-- user\_id 为空的不参与关联操作

select \* from log a join users b on a.user\_idisnotnulland a.user\_id = b.user\_id

union all

select \* from log a where a.user\_id is null

**（2）不同的数据类型关联产生数据倾斜**

用户表中user\_id 字段为 int ，log表中user\_id字段类型既有int类型也有string类型。当按照user\_id 进行两个表的join操作时，默认的Hash操作进行分配，这样会导致所有String类型的id的记录分配到一个Reducer中

-- 把不同类型的字段同意转换为String类型后 进行关联

select \* from users a left outer join logs b on a.user\_id = cast(b.user\_idas String);

**（3）比如在你拉数据的时候，某个节点上的数据分配配比过多。**

拉数据时自己多注意，避免崩溃

##### 你知道的关系型数据库和非关系型数据库有哪些？区别是什么？

讨论群成员【@风、@eliaukub 、@白小鹏、@朤 】非常感谢群友的提问

讨论结果：

关系型数据库：sql server、Oracle、mysql、db2、gp、access等等

非关系型数据库：redis、hbase、clickhouse、cassandra、es等等

关系型数据库和非关系型数据库的区别：

关系型数据库在数据的存储上通常以二维表的形式

非关系数据库可以通过html、xml等非二维表格形式存储

非关系型数据库在很多时候以复杂的关系来表达数据结构

redis 数据类型是 k-v，数据是基于内存。所以它很快。

MongoDB 是一个基于分布式文件存储的数据库。

咱们的mongodb批量有性能问题。不知道是啥问题，单条插入报错，看你ack级别了，就像kafka可以调整ack一样

芒果可以表嵌套，芒果完全可以替代mysql

芒果有readconcern和writeconcern非关系型，类似kafka的那种ack机制

clickhouse存粹的olap数据库吧

在传统关系型数据库遵循范式结构

而一些像neo4j的图数据库里则完全用一个表里的多个节点来阐述类型类似公司机构层级，或者是血缘关系，朋友关系等

而传统关系型数据库大多是循序第三范式，把关系节点拆解成行，通过parent\_id等方式来存储树形结构

redis则在嵌套这方面支持不够，它的定位就是为了做一个快速的缓存系统

类似neo4j这种存储单层网络模型，社交关系网这些，还有树形结构

redis则在嵌套这方面支持不够，它的定位就是为了做一个快速的缓存系统

芒果则是更趋近关系型数据库，支持事务，备份恢复，集群分片等分布式数据库

非关系型数据库就是因为关系型数据库在很多情况下不满足需求，或者性能无法满足，开始慢慢的演化

而类似clickhouse这种用于图表展示的比数据库，很多时候数据和数据之间没有任何关系

多半是以行式存储

真正的非关系型，说的其实是像clickhouse这种用来存储没有任何关系的数据

以前的芒果没有多段事务(类似kafka的ack)，在cap理论里只占了AP，也就是可用性和分区容错性，在后来的迭代中引入了ack机制，可以在AP和CP之间切换，在以前只能支持AP的时代，大多用于存储日志，而日志和日志之间没有直接关系

假如你数据库里有个结构层级关系的树形结构

那么你可以直接丢进mongodb做文档嵌套，实际上这个丢进去了虽然是丢进nosql数据库，但是它的最终目的还说为了存储关系型数据

nosql和sql并不是简单的用来区分关系型和非关系型数据库

具体还是要看用法和支持度

它有个演变的过程

所以很多人对这些概念其实有误区

总之核心就是，要区分关系型和非关系型，就要知道为什么有非关系型数据库

我们公司其他部门曾经在数据库里存储过用户关系结构和用户族谱，血缘关系

不过后来太慢了

所以才引进了neo4j

直接对节点的操作很方便

而要去mysql里把这个关系模型搭建出来，说白了关系型数据库顶多弄个parent\_id

也就是树形结构

关系型数据库支持树形结构已经比较困难了，大量数据下的遍历和修改都麻烦

假如冗余了path类似的字段，那更新一个树形节点可能导致很多数据都被更新

芒果支持读写事务，支持预写日志，redo之类的



芒果支持大文件存储，支持简单的mr

##### Hive如何避免小文件的产生,如何处理大量小文件？

### Hive如何避免小文件的产生

- 在使用Hive插入表中时，可以通过参数来控制文件大小

```Sql

set hive.merge.tezfiles=true;

set hive.merge.mapfiles=true;

set hive.merge.mapredfiles=true;

set hive.merge.size.per.task=128000000;

set hive.merge.smallfiles.avgsize=128000000;

-- 上面的参数设置对于MR 和 Tez引擎都适用。可以通过设置确保文件大小均小于或等于128MB

```

```Sql

hive.merge.mapfiles=true;（默认值为真） 是否合并Map输出文件

hive.merge.mapredfiles=false;（默认值为假）是否合并Reduce 端输出文件

hive.merge.size.per.task=256\*1000\*1000;（默认值为 256000000） 合并文件的大小

hive.merge.orcfile.stripe.level=true;(默认为真) 开启orc文件的合并

hive.merge.smallfiles.avgsize=16777216;(默认16000000) 作业的平均输出小于该值，启用一个mr任务合并小文件，前提是开启 mr两阶段文件合并

```

- 控制reducer的数量

```Sql

set hive.exec.reducers.bytes.per.reducer=<number>

set hive.exec.reducers.max=<number>

set mapreduce.job.reduces=<number>

-- 其中set mapreduce.job.reduces=number的方式优先级最高，set hive.exec.reducers.max=number优先级次之，set hive.exec.reducers.bytes.per.reducer=number 优先级最低。从hive0.14开始，一个reducer处理文件的大小的默认值是256M。

```

### 对于已经产生的小文件进行合并

- 一通过参数配置的方式将表中的数据写入到新表，然后再将数据覆盖/或修改表名

```Sql

set hive.merge.mapfiles=true;（默认值为真） 是否合并Map输出文件

set hive.merge.mapredfiles=false;（默认值为假）是否合并Reduce 端输出文件

set hive.merge.size.per.task=256\*1000\*1000;（默认值为 256000000） 合并文件的大小

set hive.merge.smallfiles.avgsize=16777216;(默认16000000) 作业的平均输出小于该值，启用一个mr任务合并小文件，前提是开启 mr两阶段文件合并

insert overwrit table new\_table

select \* from old\_table;

drop table old\_table;

alter table new\_table RENAME TO old\_table;

```

- 通过spark sql 的方式读取文件，然后通过减少Rdd的分区的方式达到合并小文件的目的

```scala

// 1。创建一个临时目录

// 2。将要通过Hadoop的Api计算出文件的大小。

// 3。文件大小/要合并后的文件大小 = 合并后的文件数量

// 4。备份原文件夹

// 4。spark 读取要合并的文件，并调整并行度：coalesce(合并后的文件数量)，然后写出到hive表路径

// 5。删除备份文件。

//代码自己实现

##### tez引擎的优点

Tez可以将多个有依赖的作业转换为一个作业，这样只需写一次HDFS，且中间节点较少，从而大大提升作业的计算性能。

Mr/tez/spark区别：

Mr引擎：多job串联，基于磁盘，落盘的地方比较多。虽然慢，但一定能跑出结果。一般处理，周、月、年指标。

Spark引擎：虽然在Shuffle过程中也落盘，但是并不是所有算子都需要Shuffle，尤其是多算子过程，中间过程不落盘  DAG有向无环图。兼顾了可靠性和效率。一般处理天指标。

Tez引擎：完全基于内存。  注意：如果数据量特别大，慎重使用。容易OOM。一般用于快速出结果，数据量比较小的场景

##### order by、sort by、distrbute by、cluster by，这四个的区别是什么？

1）Order By：全局排序，只有一个Reducer；

2）Sort By：分区内有序；

3）Distrbute By：类似MR中Partition，进行分区，结合sort by使用。

4） Cluster By：当Distribute by和Sorts by字段相同时，可以使用Cluster by方式。Cluster by除了具有Distribute by的功能外还兼具Sort by的功能。但是排序只能是升序排序，不能指定排序规则为ASC或者DESC。

在生产环境中Order By用的比较少，容易导致OOM。

在生产环境中Sort By+ Distrbute By用的多。

##### hive内外表的区别，各自的应用场景有哪些？

1、内部表与外部表的区别：

1.1 创建方式不同

（1）是否直接通过external创建

– create table（创建内部表）

– create external table(创建外部表）

1.2 删除

在导入数据到外部表，数据并没有移动到自己的数据仓库目录下，也就是说外部表中的数据并不是由它自己来管理的！而内部表则不一样。

* 在删除内部表时：内部表删除将表的元数据和数据同时删除。
* 在删除外部表时：外部表的元数据被删除，数据本身不删除。

2、应用场景：

2.1 外部表使用场景:

- 导入hdfs中的源数据 ，共享源数据的时候一定要记得使用外部表

2.2 内部表使用场景:

- 存放Hive处理的中间表、结果表

##### hive静态分区和动态分区，结合业务如何使用？

1、分区的意义：

使用分区技术，避免hive全表扫描，提升查询效率。

2、静态分区与动态分区

静态分区是手动的，动态分区是自动的

- 静态分区：需要知道分区数量和分区字段值并且插入数据的时候必须得知道该数据相对应的分区类型，并且还得一 一加载相应数据

- 动态分区 ：Hive会自动将数据按区域自动添加到临时分区表内,

**是否创建分区临时表？**

- 首先，是根据自己的业务场景决定的

- 如果你的原数据还有其他用处，建议你建一个临时分区表，这样不会破环你的原数据

- 如果你需要好几个动态分区表，你可以创建多个临时分区表，这样你就可以相对应的运行你的业务，达到多个目的。

3、应用场景

- 静态分区：

* 适用于小数据量并且分区数量比较少的情况；使用静态分区是自己定义数据，多用于增量表（不断增加表的内容）比如新闻表，每天都会变化增加，但需要事先知道有多少分区，每个分区都要手动插入。

- 动态分区：

* 不确定分区数量，数据量也不是很大，使用动态分区，以及在插入数据的分区字段是不确定的情况下。多用于每年 每月 每日 等 多用于全量导入
* 实际工作中趋向于使用动态分区！！！

##### union和union all的区别

1）union会将联合的结果集去重，效率较union all差

2）union all不会对结果集去重，所以效率高

##### 在项目中是否自定义过UDF、UDTF函数，以及用他们处理了什么问题，及自定义步骤？

1. 用UDF函数解析公共字段；用UDTF函数解析事件字段。
2. 自定义UDF：继承UDF，重写evaluate方法
3. 自定义UDTF：继承自GenericUDTF，重写3个方法：initialize(自定义输出的列名和类型)，process（将结果返回forward(result)），close

##### 公司有哪些主题域，数据域，是如何划分的?

公司中的数据域、主题域是如何划分的？

本话题是一个发散性的话题，那么在实际建设数据仓库中，我们还是会遇到各种问题。今天就一起聊聊主题域如何划分？

**什么是主题：**

数据仓库中的数据是面向主题组织的，主题是在较高层次上将企业信息系统中的数据进行综合、归类和分析利用的一个抽象概念，在逻辑意义上，它是对应企业中某一宏观分析领域所涉及的分析对象。例如 “销售分析”就是一个分析领域，因此这个数据仓库应用的主题就是“销售分析”。

简单说，一个主题对应一个分析对象。分析对象就是在决策、分析时重点关注的东西，这个东西其实是非常主观的，在不同的企业，或者企业的不同发展时期，所关注的点会不一样，从而影响有些主题可能存在或者不存在。一般来说，发展成熟的行业，主题的划分会比较固定。

数据仓库是面向主题的应用，主要功能是将数据综合、归类并进行分析利用。数据仓库模型设计除横向的分层外，通常还需要根据业务情况纵向划分主题域。主题域是业务对象高度概括的概念层次归类，目的是便于数据的管理和应用。

面向主题的数据组织方式，就是在较高层次上对分析对象数据的一个完整并且一致的描述，能刻画各个分析对象所涉及的企业各项数据，以及数据之间的联系。所谓较高层次是相对面向应用的数据组织方式而言的，是指按照主题进行数据组织的方式具有更高的数据抽象级别。与传统数据库面向应用进行数据组织的特点相对应，数据仓库中的数据是面向主题进行组织的。主题是根据分析的要求来确定的。这与按照数据处理或应用的要求来组织数据是不同的。

**什么是主题域：**

主题域：通常是联系较为紧密的数据主题的集合。比如销售分析，进销存分析都是主题，可根据业务的关注点，将这些数据主题划分到不同的主题域(即对某个主题进行分析后确定的主题的边界。)。主题域包含了某方面决策者关注的事物。一个主题域通常会覆盖多个业务部门，例如产品主题域涉及到销售、财务、物流、采购等部门。

主题域的确定必须由最终用户和数据仓库的设计人员共同完成的，而在划分主题域时，大家的切入点不同可能会造成一些争论、重构等的现象，考虑的点可能会是下方的某些方面：

**1、 按照业务或业务过程划分：**

**比如一个靠销售广告位置的门户网站主题域有：**

广告域，可以包含广告的库存，销售分析、内部投放分析，剩余资源赠送，资源利用率

客户域，客户在我们网站购买广告数据，竞争对手投放情况

会员域，会员的一系列情况，登录事实日志，呼叫中心通话记录，论坛发贴情况，网站访问事实，会员级的购买力，活跃度，最近关注的楼盘，城市，地域沿线，物业等，最近访问时间，近1个月访问次数，最近访问楼盘，行为指标(累计访问次数，平均停留时长，访问间隔天数，最近一个月累计访问次数）。

**比如传统制造行业，主题域有：**

生产企业域，包括企业的发货销售主题，业绩达率主题

经销商域，包括采购主题，库存主题，销售主题

活动域，包括营销活动费用主题

**比如电信行业常见的主题域有，每个主题域下存储与其业务特点相关的数据。**

业务域：

语音业务收入情况表：通话开始时间，通话结束时间，地域id ，通话类型id ，主叫，被叫，通话时长，扣费金额

 短信业务收入情况表：发送时间，地域id ，短信类型id ，发送方，接收方，短信条数，扣费金额

 客户域

客户基本信息表：客户编号，姓名，性别id ，年龄，固定电话，手机，邮箱，地域id… …

客户缴费信息表：客户编号，缴费电话，缴费方式，缴费时间，缴费金额

客户欠费信息表：客户编号，欠费号码，欠费时间，欠费金额

**2、 根据需求方划分：**

比如需求方为财务部，就可以设定对应的财务主题域，而财务主题域里面可能就会有员工工资分析，投资回报比分析等主题；

**3、 按照功能或应用划分：**

比如微信中的朋友圈数据域、群聊数据域等，而朋友圈数据域可能就会有用户动态信息主题、广告主题等；

**4、 按照部门划分：**

比如可能会有运营域、技术域等，运营域中可能会有工资支出分析、活动宣传效果分析等主题；

数据仓库面向在数据模型中已经定义好的公司的主要主题领域。典型的主题领域包括顾客、产品、订单和财务或是其他某项事务或活动。主题域是无法一次划分完整的，一般是一次先建立几个明确的主题，在大多数数据仓库的设计过程中都有一个主题域的选择过程。业务是一直发展的，因此设计之初不要想着一次把所有主题全部划分完整。我们可以遵循上面说的划分主题域的两个要点，后续采用迭代的方式补充。

**什么是数据域：**

关于数据域第一次接触时是在《大数据之路：阿里巴巴大数据实践》）一书中看到的：

数据域：指面向业务分析，将业务过程或者维度进行抽象的集合。其中，业务过程可以概括为一个个不可拆分的行为事件，在业务过程之下，可以定义指标; 维度是指度量的环境，如买家下单事件，买家是维度。为保障整个体系的生命力，数据域是需要抽象提炼，并且长期维护和更新的，但不轻易变动。在划分数据域时，既能涵盖当前所有的业务需求，又能在新业务进入时无影响地被包含进已有的数据域中和扩展新的数据域。数据域的划分工作可以在业务调研之后进行，需要分析各个业务模块中有哪些业务活动。

通常，您需要阅读各源系统的设计文档、数据字典和数据模型，研究逆向导出的物理数据模型。进而，可以进行跨源的主题域合并，跨源梳理出整个企业的数据域。

数据域可以按照用户企业的部门划分，也可以按照业务过程或者业务板块中的功能模块进行划分。例如A公司电商营销业务板块可以划分为如下数据域，数据域中每一部分都是实际业务过程经过归纳抽象之后得出的。例如：

|  |  |
| --- | --- |
| 数据域 | 业务过程 |
| 会员店铺域 | 注册、登录、装修、开店、关店 |
| 商品域 | 发布、上架、下架、重发 |
| 日志域 | 曝光、浏览、单击 |
| 交易域 | 下单、支付、发货、确认收货 |
| 服务域 | 商品收藏、拜访、培训、优惠券领用 |
| 采购域 | 商品采购、供应链管理 |

总而言之，切入的出发点逻辑不一样，就可以存在不同的划分逻辑。在建设过程中可采用迭代方式，不纠结于一次完成所有主题的抽象，可先从明确定义的主题开始，后续逐步归纳总结成自身行业的标准模型。

参考：

<https://dataclub.blog.csdn.net/article/details/110607231?utm_medium=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-7.control&depth_1-utm_source=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-7.control>

https://blog.csdn.net/qq\_43315928/article/details/101001813

https://help.aliyun.com/document\_detail/114443.html

##### hive元数据为什么使用Mysql？不使用默认的Derby？

在安装完成Hive之后默认是以Derby数据库作为元数据库，存储Hive有那些数据库，以及每个数据库中有哪些表，但是在实际生产过程中，并不是以derby作为Hive元数据库，都是以Mysql去替换derby。

原因主要是基于两点原因：

1.Derby数据库不支持并发，也就是只支持单线程操作，当有一个用户在对Hive进行操作时，其他用户则无法操作，导致整体效率性能较低。

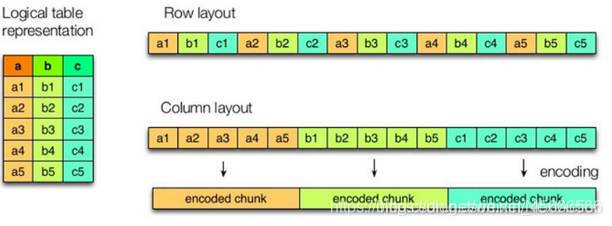
2.当在切换目录后，重新进入Hive会找不到原来已经创建的数据库和表。主要原因是比如第一次是在bin目录下进入Hive，那么在进入Hive后，会在bin目录下创建一个metaStore.db的目录，然后在此目录下会创建一个derby.log的日志文件，所有的元数据的信息都是存储在这个日志文件中。那么经过更换目录后，然后进入Hive就会存在一个问题，原有的元数据信息都是基于bin目录下创建的，所以在找数据库和表的信息时候，都是基于bin目录的，而此时就会存在找不到原有创建的数据库和表的信息，所以会将默认的Derby数据库进行更换为Mysql。

##### hive常见存储格式的区别和应用场景？

**引言**

Hive支持的存储数的格式主要有：TextFile、SEQUENCEFILE、ORC、PARQUET。其中，TEXTFILE、SEQUENCEFILE是基于行存储，ORC、PARQUET基于列存储。

**行存储和列存储**

* 上图中左边为逻辑表，右上为行存储，右下为列存储。
* **行存储特点**：查询满足条件的一整行数据的时候，列存储则需要去每个聚集的字段找到对应的每个列的值，行存储只需要找到其中一个值，其余的值都在相邻地方，所以此时行存储查询的速度更快。
* **列存储特点**：查询满足条件的一整列数据的时候，行存储则需要去每个聚集的字段找到对应的每个行的值，列存储只需要找到其中一个值，其余的值都在相邻地方，所以此时行存储查询的速度更快。另一方面，每个字段的数据类型一定是相同的，列式存储可以针对性的设计更好的设计压缩算法。
* 当查询结果为一整行的时候，行存储效率会高一些；当查询表中某几列时，列存储的效率会更高。
* 在对数据的压缩方面，列存储比行存储更有优势，所以列存储占用空间相对小一些。

**1、TextFile**

- 默认的文件格式，行存储。建表时不指定存储格式即为textfile，导入数据时把数据文件拷贝至hdfs不进行处理

- 优点：最简单的数据格式，便于和其他工具（Pig, grep, sed, awk）共享数据，便于查看和编辑；加载较快

- 缺点：耗费存储空间，I/O性能较低；Hive不进行数据切分合并，不能进行并行操作，查询效率低

- 应用场景：适用于小型查询，查看具体数据内容的测试操作

**2、sequencefile**

- 含有键值对的二进制文件，行存储

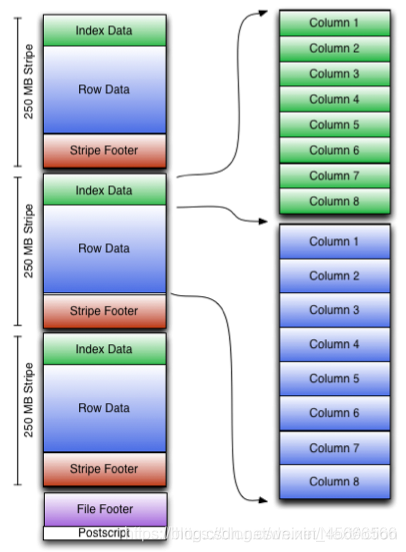
- 优点：可压缩、可分割，优化磁盘利用率和I/O；可并行操作数据，查询效率高

- 缺点：存储空间消耗最大；对于Hadoop生态系统之外的工具不适用，需要通过text文件转化加载

- 应用场景：适用于数据量较小、大部分列的查询。

**3、orc**

ORC文件由stripe、file footer、postscript三部分组成：

**3.1 Stripe**

一个ORC文件，可以包含多个Stripe，Stripe默认大小250M。

Stripe由Index Data、Row Data、Stripe Footer三部分组成:

- Index Data

轻量级索引，默认每隔1w行做一个索引，目的是记录某行的各个字段在Row Data中的偏移量,保存了该stripe上数据的位置信息，总行数等信息。

- Row Data

真正存储数据的部分，对每个列进行编码存储。

- Stripe Footer

存储stripe的元数据信息。

**3.2 File Footer**

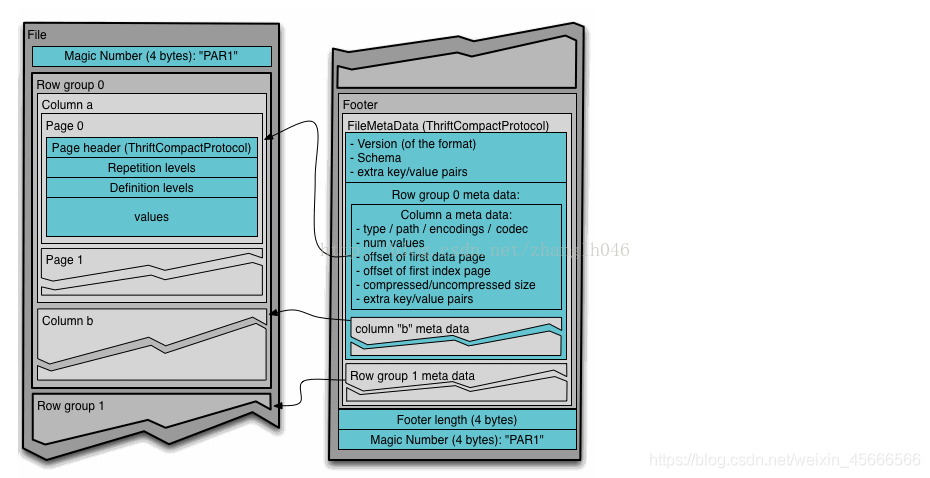
每个ORC文件中存有一个file footer，里面记录了各个stripe存储的行数，每个列的数据类型等信息。

**4、 parquet**

- parquet类似于orc，相对于orc文件格式，hadoop生态系统中大部分工程都支持parquet文件。

- 存储模式：按列存储，Parquet文件是以二进制方式存储的，不可以直接读取和修改的，文件是自解析的，文件中包括该文件的数据和元数据

- 存储结构：

 行组(Row Group)：按照行将数据物理上划分为多个单元，每一个行组包含一定的行数，在一个HDFS文件中至少存储一个行组，Parquet读写的时候会将整个行组缓存在内存中，所以如果每一个行组的大小是由内存大的小决定的

- 优点： Parquet能够很好的压缩和编码，有很好的查询性能，支持有限的模式演进。

- 缺点：写速度通常比较慢，不支持update、insert、delete、ACID

- 应用场景：适用于字段数非常多、无更新、只取部分列的查询

##### hive常见的建表方式有哪些？各自的使用场景是什么？

1、直接建表法

create table tableName(字段名称 字段类型 [comment '中文注释说明']，字段名称 字段类型， ....) row format delimited fields terminated by 'char分割符即列分割符' lines terminated by '行分割符';

应用场景：

直接进行 字段类型，字段备注，数据存储格式的自定义

2、抽取（as）建表法

create table new\_table\_name as select \* from table\_name;

应用场景：

中间逻辑处理的时候，进行建表，直接复制表的数据和结构,也就是建立一张临时表，方便以后的使用

3、like建表法

create table table\_name\_like like table\_name

应用场景：

只关注表结构，不需要数据

# 数仓

##### 在数据仓库中，数据同步有哪些方式？我们如何进行选择？

数据同步方法主要有全量同步策略、增量同步策略、新增及变化策略、特殊策略

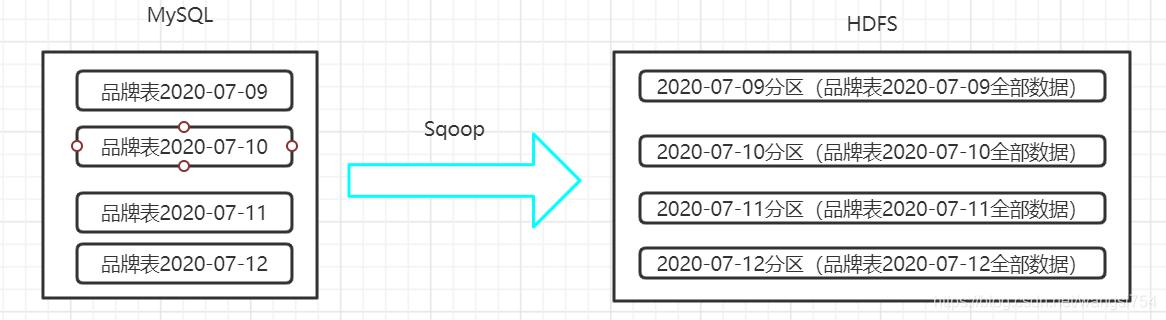
1.全量同步策略

应用场景：

1、首次拿到业务相关数据时且不关心主键的情况下，一般会将全部数据导入到一张表中。

2、针对业务需求： 每日全量，每天存储一份完整数据，作为一个分区。适用于表数据不大，且每天既有新数据插入，也会有旧数据修改的场景。

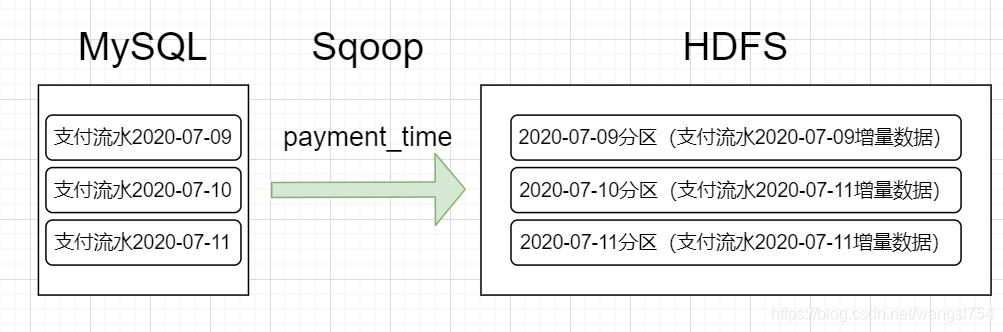
例如：编码字典表，品牌表，商品分类表，优惠表，活动表，商品表，加购表，SPU表等。

2、增量同步策略

应用场景：

1、一般在第一次全量建表以后，基本会选择增量同步策略，除非有所有数据需要更新之类的

2、每天存储一份增量数据，作为一个分区。适用于表数据量大，且每天只有新数据插入的场景。例如：退单表，订单状态表，支付流水表，订单详情表，商品评论表等。

3、新增及变化策略

每日新增及变化，就是存储创建时间和操作时间都是今天的数据。适用场景为表的数据量大，既会有新增，又会有变化。例如用户表、订单表、优惠券领用表等。

4、特殊策略

1、客观世界维度

没变化的客观世界的维度（比如性别，地区，民族，政治成分，鞋子尺码）可以只存一份固定值。

2、日期维度

日期维度可以一次性导入一年或若干年的数据。

3、地区维度

省份表、地区表。

[以上参考资料](https://blog.csdn.net/wangsl754/article/details/107237608)

补充：

关心主键并且只关心部分数据，我们可以使用select查询语句同步数据

a、研发落表，都有主键id——针对MySQL来说

需求：将每天的增量数据与目标数据进行合并，作为下一步开发的原数据

-- 创建临时表，同步过来数据加载到临时表

load data local inpath "文件路径 " overwrite into table tmp

-- 增量数据和目标数据合并：

insert overwrite table target select \* from tmp union all select \* from target a left join tmp b on a.id=b.id where b.id is null

-- 将临时表删除——节省内存

drop table if exists tmp;

b、比如用户针对一个订单发生多次改派,只关心终态

只需要取最后一条数据

select \* from (select \*,row\_number() over(partition by order\_id order by updatetime desc) rank from target) t where t.rank=1;

##### 拉链表的逻辑，使用场景，回滚

讨论群成员【@汉华国祚 】

拉链表记录一个事物从开始，一直到当前状态的所有变化的信息。即可以在所规定的时间粒度上体现数据完整的生命周期。

拉链表适用场景：

1 表的数据量很大。

2 表中的部分字段会被update更新操作。

3 需要查看某一个时间点或者时间段的历史快照信息。

4 表中的记录变化的比例和频率不大。

拉链表回滚：

假设恢复到t天之前的数据，即未融合t天数据之前的拉链表，假设标记的开始日期和结束日期分别为s、e，具体分析如下：

1 当t-1>e时，s数据、e数据在t天之前产生，保留即可

2 当t-1=e时，e数据在t天产生，需修改

3 当s<t<=e时，e数据在t+n天产生，需修改

4 当s>=t时，s数据、e数据在t+n天产生，删除即可

情景：拉链表dwd\_userinfo\_db,目前时间是2020-12-15，想回滚到2020-11-27,那么拉链表的状态得是2020-11-26。

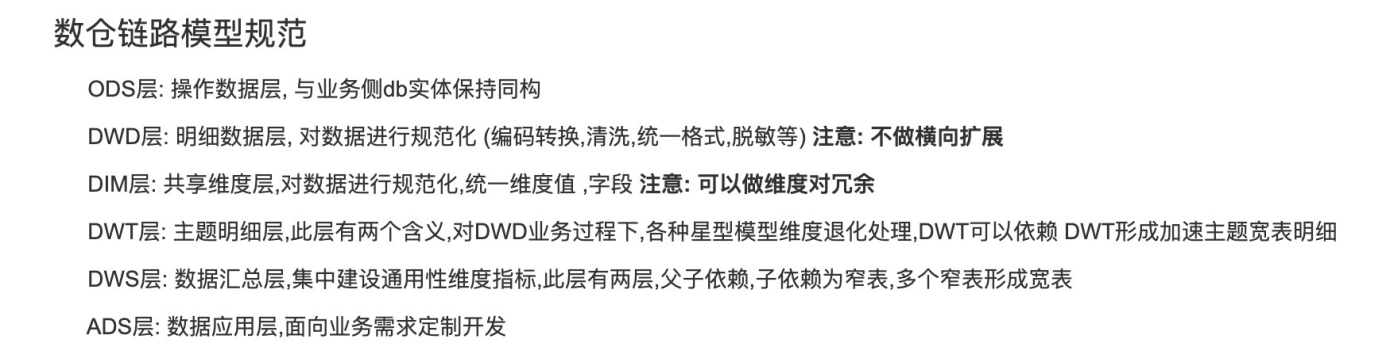
拉链表回滚：过滤starttime<=2020-11-26的数据，将endtime>2020-11-26的修改为9999-99-99

##### 建模工具是什么？

PowerDesigner/SQLYog/EZDML

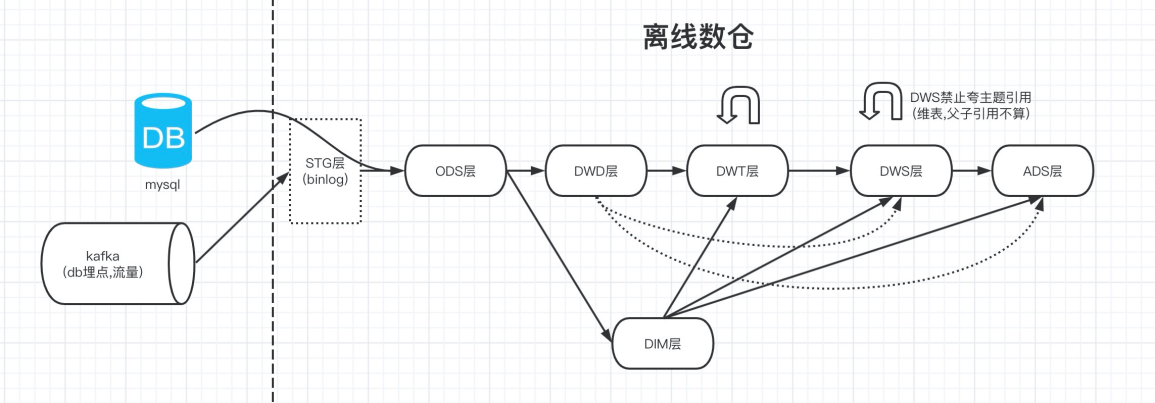
##### 数仓如何分层的？及每一层的作用？思考：为什么要这么分层？

数仓分层及每一层的作用：



为什么要这么分层？【思考】





不分层会产生的问题

a. 缺乏统一的业务和技术标准

如：开发规范、指标口径不统一

b. 缺乏统一数据质量监控

如：字段数据不完整和不准确等。

c. 业务知识体系散乱

如: 导致数据研发人员开发成本增加。

d. 数据架构不合理，数据层之间分工不明显

如: 数据流向混乱。

e. 缺失统一维度和指标管理

##### 什么是数仓？解决了什么问题，如果没有会怎样？

【文字版】

在小数据量的时代，企业的需求是建设各种应用来实现业务流程，此时的数据存储在关系型数据库中，范式模型也成为了业务系统的主要数据存储模型。

后来，企业逐渐有了数据分析的需求，数据量、数据的格式、来源发生了很大变化。虽然业务系统的范式模型也能满足一部分的分析需求，但是存在很多的缺陷，比如表关联复杂、查询效率低、一些需求需要跨系统取数等等，于是人们转而寻找更合适的数据模型来满足分析型应用的需求，在研究发展过程中，数仓的概念就诞生了。

类似于快递的集散中心，比如上海韵达的集散中心，全国各地的快递先汇总到这，暂存、根据地区分类、根据优先级确定顺序、分发到各个快递员，然后再送到手里，实际上这个仓库并不仅仅是存储，还有转换格式、分类聚合、分层的作用，所以用集散中心这个词更合适。既然是仓库，那必然要考虑数据怎么存，存什么样的格式，怎么取怎么用，这个就需要一套体系去组织和管理，这整套体系就形成了数仓的一系列理论。

总结大数据场景下企业面临的数据问题：多源异构、脏乱差的数据现象。

- 结构复杂

- 数据脏乱

- 理解困难

- 缺少历史

数据仓库，英文名称为DataWarehouse，可简写为DW或DWH。它出于分析性目的而创建。将不同来源的数据采集到统一的存储介质，对数据进行清洗、分析、分类、存储、计算、根据业务重组，为企业决策提供数据，为下游应用供数，实现数据赋能、数据驱动。

数仓的业务价值体现：

- 效果

- 诊断

- 预警

这里的企业决策，从业务的角度，其实分4块：

1. BI，就是报表，比如分组求topN。

**1. 即席查询。**

1. 随机、自定义查询数据库。即席：当场、随时，就是当场去查询。自己键入的SQL代码就是即席查询，相对于提前写好来说。将SQL代码写在java代码里，通过某个按钮去触发就不是即席查询，

2. 一般使用sql。

3. 并不是大数据领域独有的概念。

4. 并不意味着慢，也可以使用缓存、索引。秒级。

5. Kylin、druid、presto、impala是常见的即席查询工具。[四种即席查询工具比较](https://zhuanlan.zhihu.com/p/266695601)

6. 和实时数仓的区别

**2. 可视化**

2. ai

1. 为ai模型提供训练数据

2. 跑ai模型，为ai模型供数，比如为财务预警模型提供营业利润、无形资产比率、总利润增长率等数据

3. 下游web应用，将ADS层数据同步到下游web应用的业务数据库中

比如数据管控平台的指标查询。

运营系统、业务系统

4. 推荐系统和用户画像。

一般企业这2块算到大数据部门，数智，所以有些会要求会一些算法。

**\*\*==>数仓是一切大数据应用的基础。\*\***

【图片版】

