**名词解释**

**Segment** 一个时间区间的构建结果

**Cuboid** 是cube的一个切面，64个bit构成，按照rowkey顺序，每一位标识某一个维度是否包含在该切面内 A B C D 1 1 0 0 13509

**Base Cuboid** 包含所有需要预计算的维度的切面

**Cube的设计**

**Model**

* 定义了一个星型或雪花模型:
  + 事实表，维表，以及它们的Join关系
  + 维度列的集合
  + 度量列的集合
  + 其它信息，如分区等

### 数据模型可以为多个Cube重用

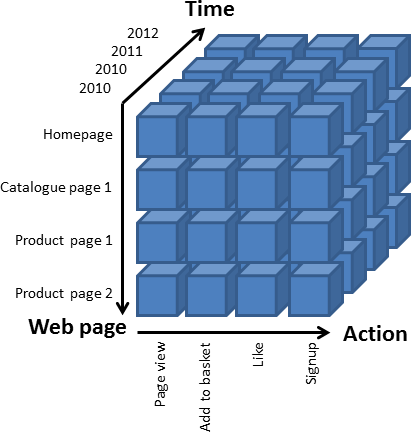
### 建模前首先要Load Table

**坑：hive不支持非等值连接**

**事实表和维表的关联用维表主键，避免多个字段关联**

**Cube**

### 设置维度和度量关系

* Cube主要有两部分组成
  + 维度
    - 分析数据的角度，如时间，地点
  + 度量
    - 测量的数据，如用户数，销售额

### SQL中的维度和度量

* + Group By, Where Condition => 维度

Count(), SUM(), Max()…=> 度量

### 维度对Cube的复杂度起主要作用

* + 一个N维的full Cube，有2N -1个Cuboid
  + Cuboid的大小，由其各个维度的基数决定
  + 查询都是按维度条件

减少cuboid数量，权衡在线聚合（时间）和离线预聚合（时间空间）

### 衍生维度

* 聚合组（强制、层级、联合维度）

衍生(Derived)维度

* + 当**维表**上的维度可以从PK推导时，可以声明这些维度为Derived
  + 注意：推导的维度基数变化不建议过大，如User\_Id -> Gender

聚合组

### 对维度分组

* + 将经常一起使用的维度放入聚合组 2N+M+L =》 2N + 2M + 2L
  + 在聚合组内使用Mandatory, Hierarchy和Joint等方法进一步降维
* 强制(Mandatory)维度
  + 一定或总是出现在where条件的维度
  + 声明一个维度为mandatory后，将只计算包含此维度的组合
  + 可以将组合数减少为原来的1/2
* 层级(Hierarchy)维度
  + 将有层级关系的若干维度，声明它们为Hierarchy关系
  + 观察维度是否有层级关系，如国家，省，市
  + 这M个维度的组合数，将从2M减少为M+1
  + 当层级数>=3，效果明显
* 组合(Joint)维度
  + 哪些维度组合总是会一起出现
  + 总是一起出现的维度，可以定义Joint
  + 低基数维度，可以使用Joint组合在一起(建议6个以下的维度组合，count(distinct ....)值小于10w)

M个维度使用joint，可以将计算组合数从2M减少为2

举例计算一个聚合组的coboid数量

A B C D E F G H 原本有 28 -1

Mandatory A ------------1

Hierarchy B C D ------------3

Joint E F -----------2

不包含在以上的 G H ----------4

Cuboid数量 1\*3\*2\*4 =24个

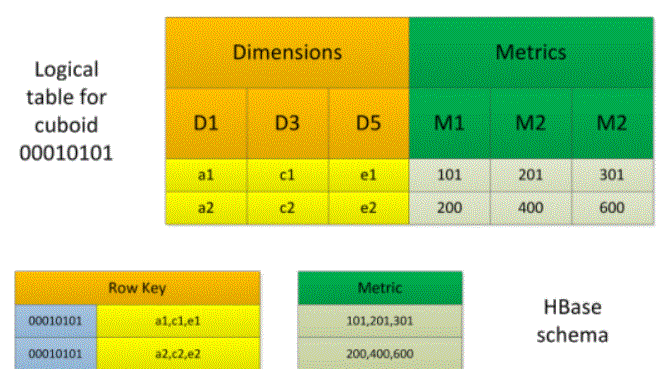
A B C 命中11100000

AD ACD ABCD A ABD 命中111100000

**控制单个聚合组cuboid数量小于256个**

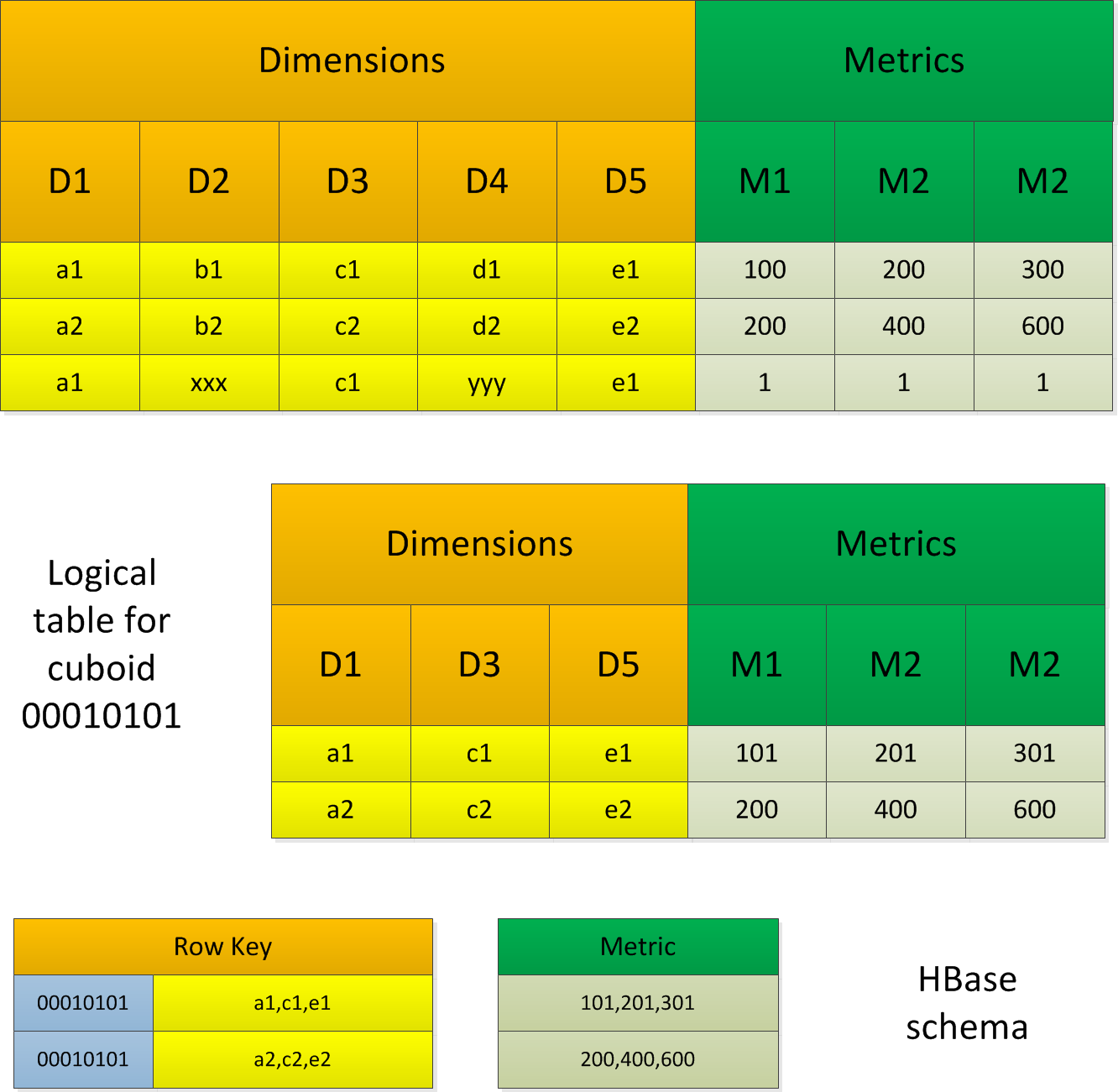
Rowkey

* + key 包括有prefix（Cuboid ID）+维度信息
  + value 是cuboid信息：度量信息



### Kylin会将查询条件，转换成Hbase Scan的start\_key和end\_key

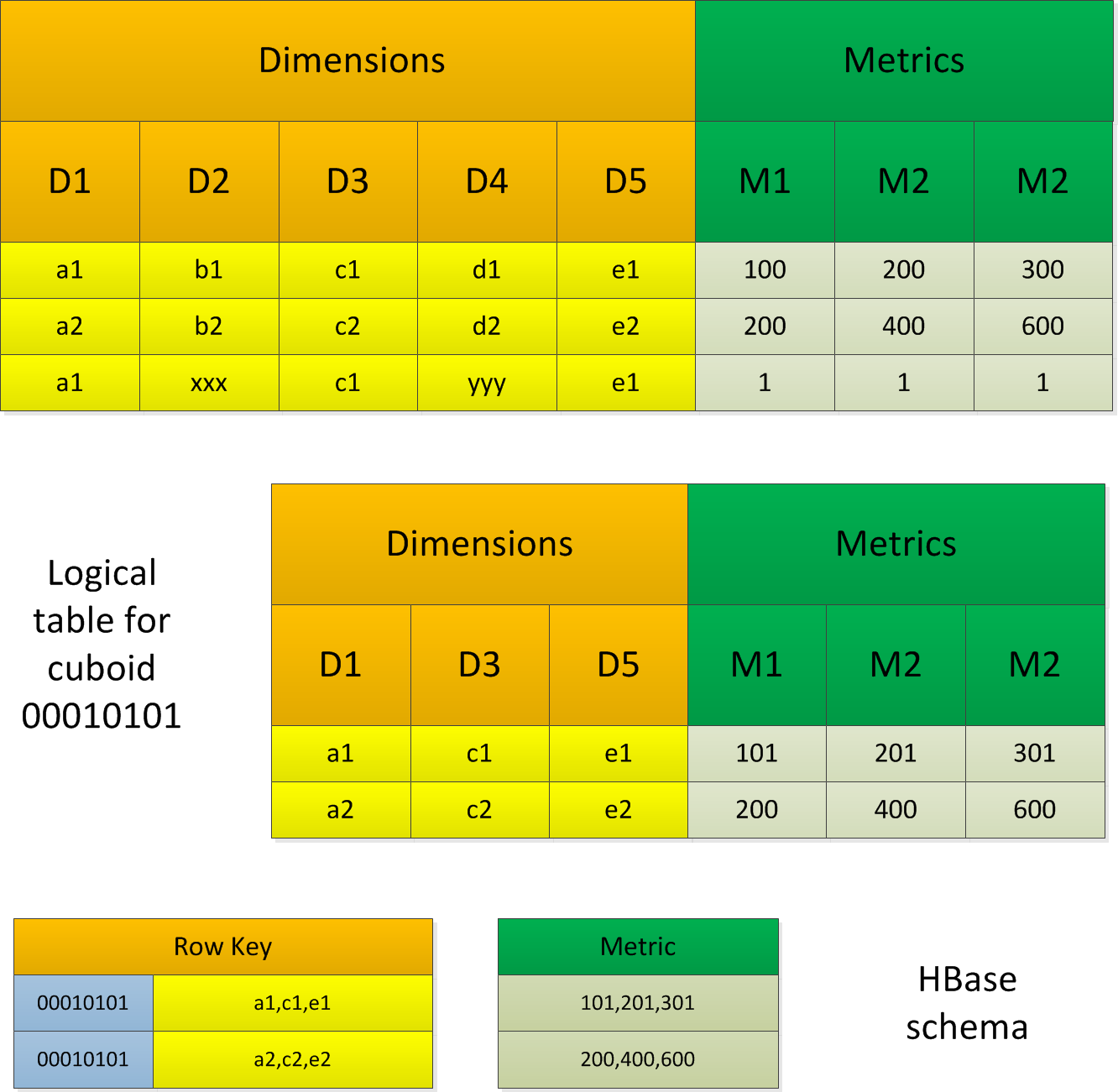
Dimension在rowkey中的次序，对查询性能有显著的影响

建议的优先次序：Mandatory > 常用于filter维度>基数大的维度>其他维度

* 对维度值编码，可以起到
  + 将值转成长度一致的字节，可以紧凑拼接
  + 通常可以起到压缩的作用

Dict编码：使用字典将长的值映射成短的ID；适合中低基数的维度，默认推荐编码

Fixed\_Length编码：将值补齐或截断成固定长度；适合超高基维度，比如身份证号



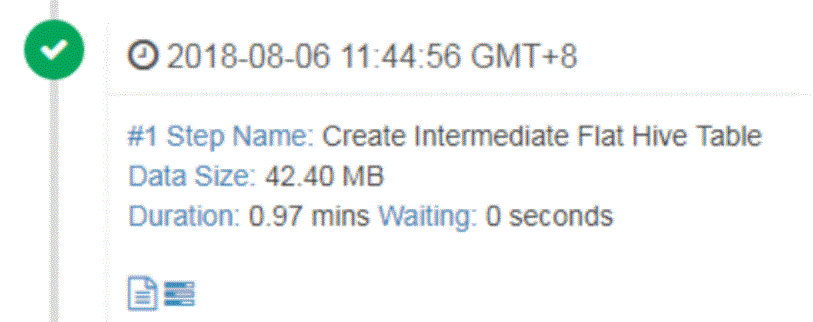
**Cube的构建**

构建的实质其实是其它Bi工具说的load data to DW，不同的是KYLIN有预计算

根据时间的增量构建或选定Segment的刷新构建，已有的segment无法只刷新一部分

构建的整个过程讲解

* 打平表 (Hive Job)



* 均匀分布平表 (Hive Job)

kylin.source.hive.redistribute-flat-table = true

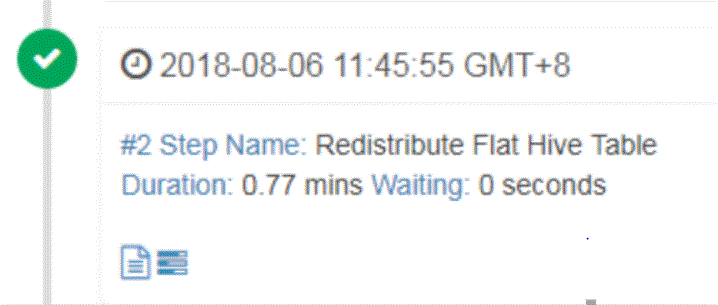
kylin.source.hive.keep-flat-table = false

kylin.engine.mr.mapper-input-rows 默认100w

通过设置reducer的数量，把平表分成几个文件，间接的控制了下一步base cuboid 的mr起的mapper数量

# distribute by rand()

如果有设置shardby字段，这里就是根据那些字段distribute



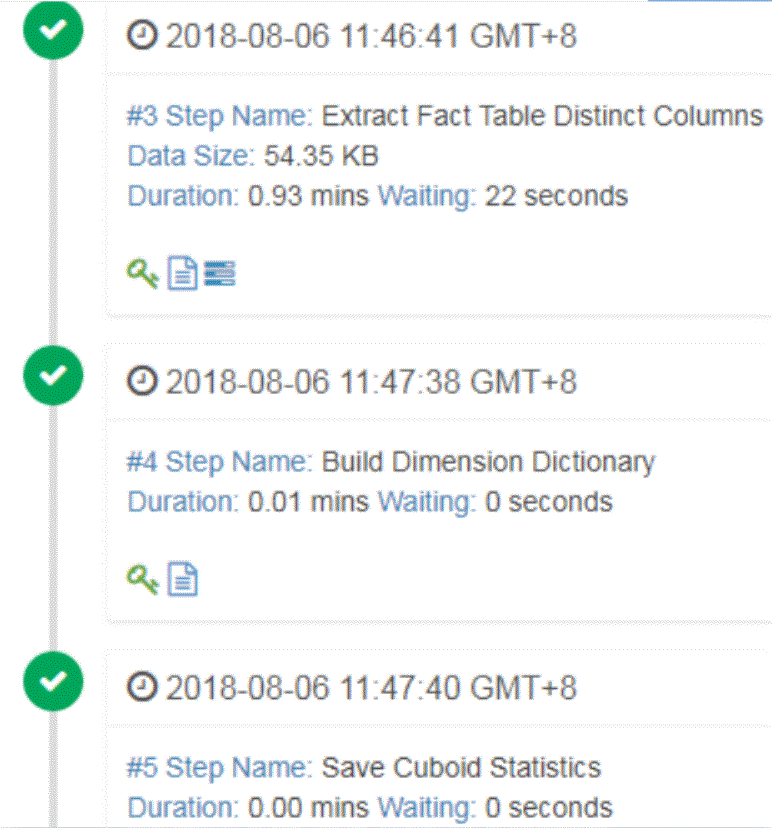
* 统计每列上的值，创建字典 (MR Job)

kylin.engine.mr.build-dict-in-reducer=true

Build Dimension Dictionary步骤中，dict极限值为2G  2147483648

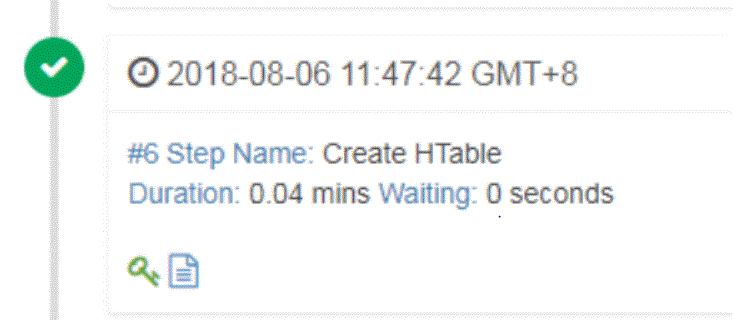
实践中，对于维度做Dict编码，需要基数约小于2000w，而且这一步消耗本地内存，太大可能导致jvm内存溢出

* 保存Cuboid统计信息（HLL）预估每个预计算的cuboid有多少行



hbase建表 region预分区

kylin.storage.hbase.region-cut-gb = 5G



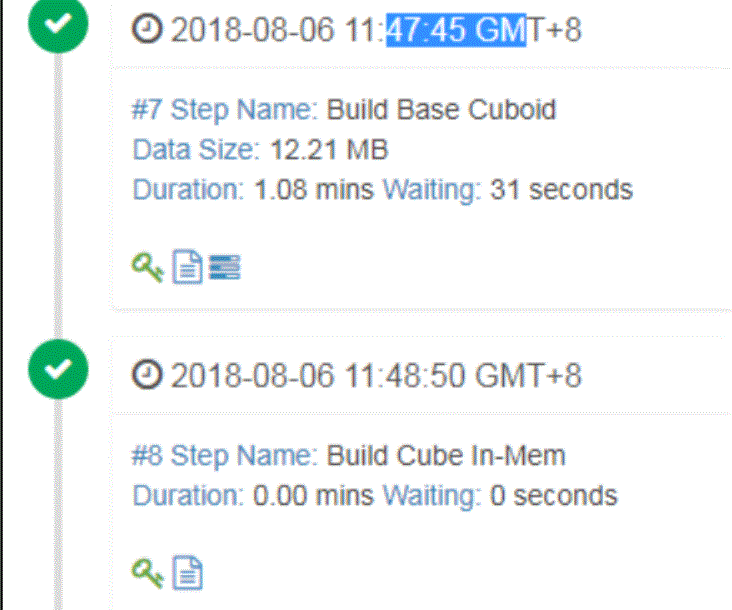
构建base Cuboid

mr调优的参数在hive\_conf\_job.xml中

kylin.engine.mr.reduce-input-mb = 500

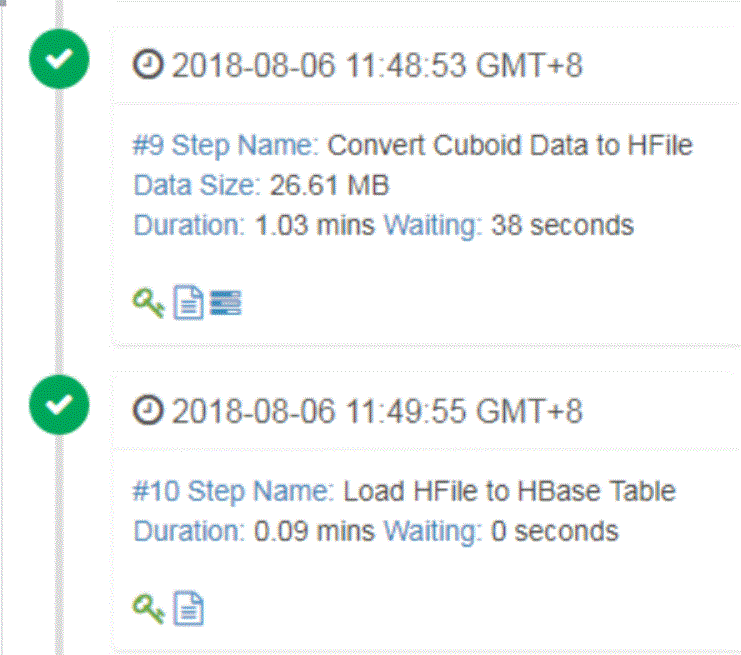
kylin.cube.algorithm=layer

In-Mem Cubing已废弃，Build Cube IN-Mem什么也不做

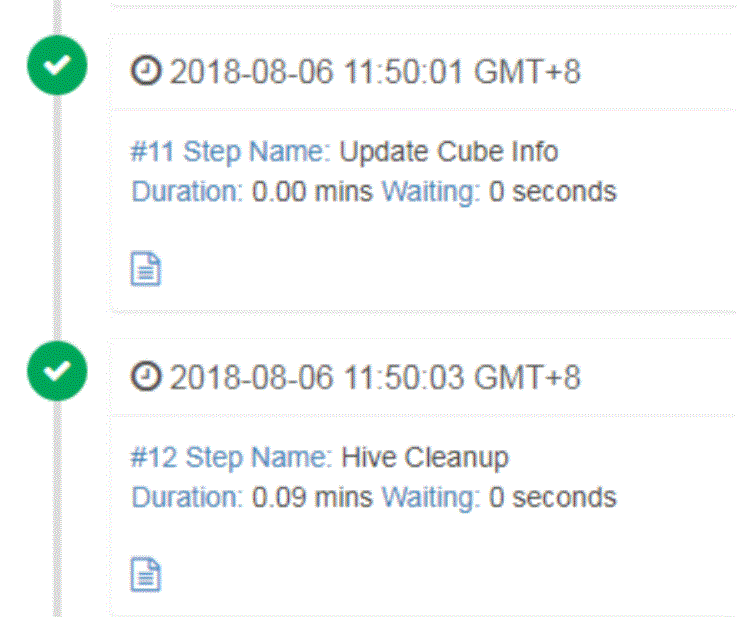


每一层mr的结果文件是sequence file，把它转成hbase的hfile，然后用bulk load加载到hbase中

kylin.storage.hbase.hfile-size-gb = 2G



更新cube元数据，清理构建中的垃圾（并没有清除中间过程在hdfs上生成的一些文件，需要用清理脚本来清理）



Layer构建的举例讲解

假设有四个维度 A B C D

必选 A

层级没有

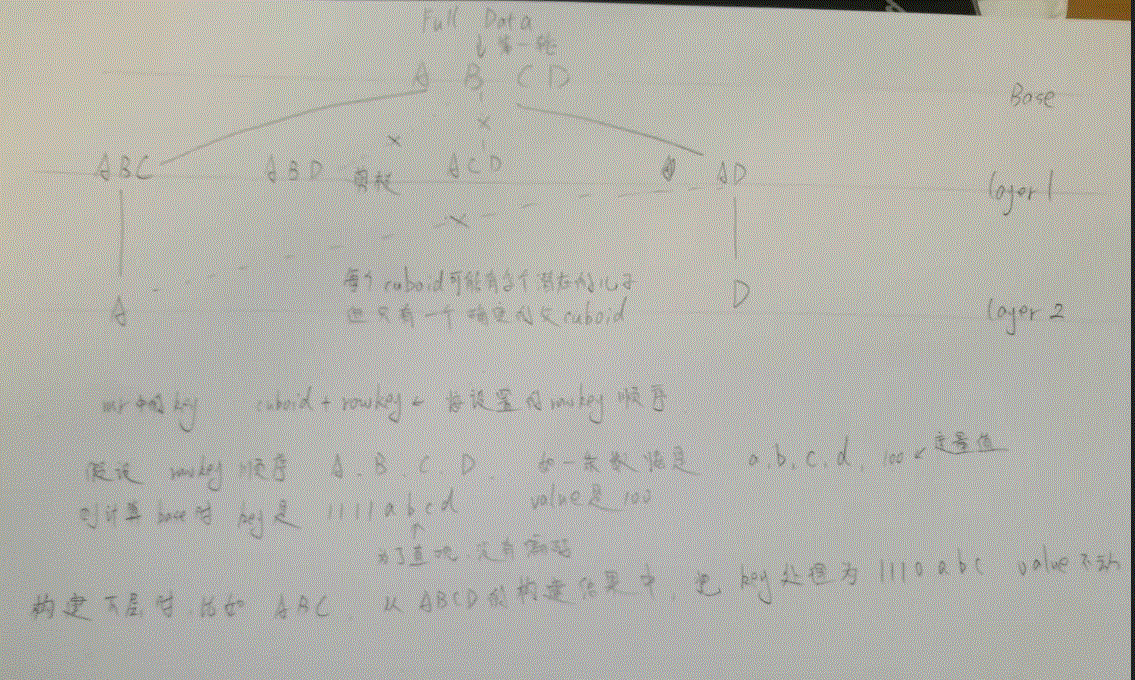
联合 B C

举例：

1111ABCD 100

1110ABC 100

1001AD 100



局限性：

Cuboid用的是long，所以目前最多支持63个维度

Cube优化

1. 利用聚合组控制cuboid数量
2. Derived维度
3. rowkey的排序，优化查询性能
4. Shardby

* 一般为高基维列，更适合均分
* 同一Cuboid中的不同行根据shardby维度的值，分配到不同的分区中
* 提速 where shardby列 =‘xxx’（只扫某个分区）

关键参数及调优

所有kylin参数都在conf/kylin.properties中以键值对形式保存

Cube级可覆盖参数：

重写默认的 kylin.properties：

Cube 相关的参数可以在任意 Cube 级别进行自定义

比如：kylin.cube.algorithm

kylin.storage.hbase.region-cut-gb

重写默认的 Hadoop job conf :

Mr job加前缀kylin.engine.mr.config-override.

Hive job加前缀kylin.source.hive.config-override.

比如：kylin.engine.mr.config-override.mapreduce.map.memory.mb=8192

kylin.source.hive.config-override.mapreduce.job.queuename=myQueue

**Cube构建参数**

kylin.cube.aggrgroup.max.combination 4096

kylin.job.retry 1

kylin.job.concurrent.max.limit 10

kylin.engine.mr.reduce-input-mb 500

MR Job启动前会依据输入预估Reducer接收数据的总量，再除以该参数得出Reducer的数目

kylin.engine.mr.min-reducer-number 1

kylin.engine.mr.max-reducer-number 500

**查询参数**

kylin.storage.partition.max-scan- bytes 3GB

查询服务器为每条查询投入的内存上限。如超过Kylin会抛出异常以保护查询服务器不会耗尽内存

kylin.storage.hbase.coprocessor- mem-gb 3.0

Coprocessor为每条查询投入的内存上限。如超过， 则数据会溢出到硬盘缓存，降低速度

kylin.query.cache-enabled true

kylin.query.cache-threshold-duration 2000 （millisecond）

只有执行较慢的查询才会被缓存

kylin.query.cache-threshold-scan-count 10240

扫描记录较多的查询才会被缓存

**hive关键参数优化**

1. Hive源表建议使用增量刷新列做分区，避免打平表全表扫描
2. Hive查询开启mapper side join(默认已开启)，开启mapper side join，Hive会将较小的维表加载到mapper的内存中，从而大大提速与fact表的join 操作，加速打平表

hive\_job\_conf.xml中

hive.auto.convert.join true

hive.auto.convert.join.noconditionaltask true

hive.auto.convert.join.noconditionaltask.size 100000000

1. 开启snappy压缩，默认没开

mapreduce.map.output.compress.codec org.apache.hadoop.io.compress.SnappyCodec

mapreduce.output.fileoutputformat.compress.codec org.apache.hadoop.io.compress.SnappyCodec

mapreduce.output.fileoutputformat.compress.type BLOCK

1. 禁用小文件合并

Kylin会通过redistribute的方式来进行合并，因而Hive的合并是多余的

hive.merge.mapfiles false

hive.merge.mapredfiles false

## kylin.properties中定义的参数是启动Kylin时默认加载的，需要重新启动Kylin才能生效

## kylin\_hive\_conf.xml是实时提交给Yarn的配置，即使不重启也会生效

Kylin\_hive\_conf.xml优先级高于kylin.properties

**Mr关键参数优化**

**资源分配**

例如服务器24核，128GB内存，平均每个核5.3GB，除去系统内存， 为单个task分配4GB可以获得最大并发

mapreduce.map.memory.mb

mapreduce.map.java.opts

mapreduce.task.io.sort.mb 2047

mapreduce.reduce.memory.mb

mapreduce.reduce.java.opts

mapreduce.reduce.shuffle.input.buffer.percent 0.5

JVM堆内存的多少比例可以用于存放map任务的输出结果

**开启压缩，减少IO**

mapreduce.map.output.compress true

mapreduce.map.output.compress.codec org.apache.hadoop.io.compress.SnappyCodec

mapreduce.output.fileoutputformat.compress true

mapreduce.output.fileoutputformat.compress.codec org.apache.hadoop.io.compress.SnappyCodec

**利用队列资源隔离**

mapreduce.job.queuename

**其他**

mapreduce.job.reduce.slowstart.completedmaps 0.95

mapreduce.map.maxattempts 2

mapreduce.reduce.maxattempts 2

日常维护

* 元数据备份工具：
  + $KYLIN\_HOME/bin/metastore.sh backup
  + 该工具把元数据以文件形式下载到本地

$KYLIN\_HOME/meta\_backups/meta\_2018\_06\_10\_20\_24\_50

* + 前往该目录，就可看到备份下来的元数据文件
* 元数据恢复工具
  + $KYLIN\_HOME/bin/metastore.sh restore /path/to/metadata
  + 例如：$KYLIN\_HOME/bin/metastore.sh restore

$KYLIN\_HOME/meta\_backups/meta\_2018\_06\_10\_20\_24\_50

## 元数据清理工具：

* + $KYLIN\_HOME/bin/metastore.sh cleanup [--delete true]