

---

# KYLIN 学习资料

## 1、关于 KYLIN

官网地址: <http://kylin.apache.org/>

**本质：拿空间换时间---《pre-build》**

通过预计算把用户需要查询的维度以及他们所对应的考量的值，存储在多维空间里。

当用户查询某几个维度的时候，通过这些维度条件去定位到预计算的向量空间，通过再聚合处理，快速返回最终结果给用户

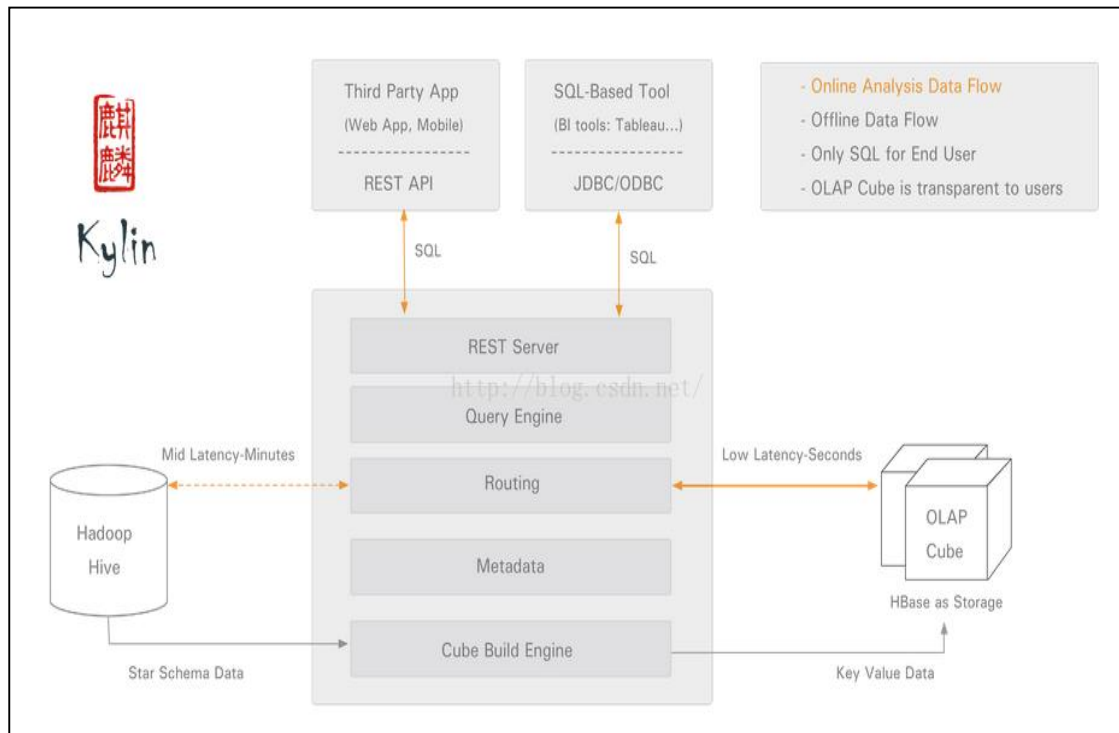
Apache Kylin™ 是一个开源的分布式分析引擎，提供 Apache Hadoop 之上的 SQL 查询接口及多维分析（OLAP）能力以支持超大规模数据。Apache Kylin™ 首创使用多维立方体预计算处理大数据查询，在 Hadoop 平台上提供亚秒级查询千亿记录的能力，提供标准 SQL 接口，查询性能比 Hive 快 100~1000 倍。其独创的稀疏立方体、压缩存储、微批处理构建等技术，很好的解决了大数据建立索引的指数级膨胀的难题，从而在查询速度上大大领先于其他基于 MPP(大规模并行计算)等技术的解决方案，可以很好的满足百亿规模以上超大数据集提供快速的、高并发标准 SQL 查询的业务需求，并通过其 ODBC、JDBC 驱动及 REST API 等与 BI 工具，前端可视化技术等无缝整合。Apache Kylin™ 也是第一个由中国人主导的 Apache 顶级项目，于 2015 年 11 月正式毕业成为 Apache 顶级项目。2015 年 9 月，Apache Kylin 与 Spark、HBase、Kafka 等并列荣膺 InfoWorld 2015 年 Bossie 最佳开源大数据工具奖。这也是国人项目第一次获得该国际大奖。Apache Kylin™ 在大数据分析领域应用广泛，获得了快速的推广。国内外一线的互联网，金融，电信等公司越来越多地采用 Apache Kylin™ 作为其大数据分析平台。

## 2、KYLIN 介绍

KYLIN 是 Ebay 开发的一套 OLAP 系统，与 Mondrian 不同的是，它是一个 MOLAP 系统，主要用于支持大数据生态圈的数据分析业务，它主要是通过预计算的方式将用户设定的多维立方体缓存到 HBase 中（目前还仅支持 hbase），这段时间对 mondrian 和 kylin 都进行了使用，发现这两个系统是时间和空间的一个权衡吧，mondrian 是一个 ROLAP 系统，所有的查询可以通过实时的数据库查询完成，而不会有任何的预计算，大大节约了存储空间的要求（但是会有查询结果的缓存，目前是缓存在程序内存中，很容易导致 OOM），而 kylin 是一个 MOLAP 系统，通过预计算的方式缓存了所有需要查询的数据结果，需要大量的存储空间（原数据量的 10+倍）。一般我们要分析的数据可能存储在关系数据库（mysql、oracle，一般是程序内部写入的一些业务数据，可能存在分表甚至分库的需求）、HDFS 上数据（结构化数据，一般是业务的日志信息，通过 hive 查询）、文本文件、excel 等。kylin 主要是对 hive 中的数据进行预计算，利用 hadoop 的 mapreduce 框架实现。而 mondrian 理论上可以支持任意的提供 SQL 接口数据，由于关系数据库一般会存在索引，所以即使使用 mondrian 去查询性能还是可以接受的，当前我们使用的 oracle 数据库，千万条级别的记录，查询可

以在分钟级别完成，但是对于 hive、这样的数据源查询就太慢了，慢得不可以接受。

## 2.1、KYLIN 系统架构



kylin 由以下几部分组成：

- REST Server：提供一些 restful 接口，例如**创建 cube**、**构建 cube**、**刷新 cube**、**合并 cube** 等 cube 的操作，project、table、cube 等元数据管理、用户访问权限、系统配置动态修改等。除此之外还可以通过该接口实现 SQL 的查询，这些接口一方面可以通过第三方的调用，另一方也被kylin的web界面使用。

- jdbc/odbc 接口：kylin 提供了 jdbc 的驱动，驱动的 classname 为 org.apache.kylin.jdbc.Driver，使用的 url 的前缀 jdbc:kylin:，使用 jdbc 接口的查询走的流程和使用 RESTful 接口查询走的内部流程是相同的。这类接口也使得 kylin 很好的兼容 tebleau（**可视化分析工具**）甚至 mondrian（一个用 Java 写成的 OLAP（**在线分析性处理**）引）。

- Query 引擎：kylin 使用一个开源的 Calcite（**Hadoop 中新型大数据查询引擎**）框架实现 SQL 的解析，相当于 SQL 引擎层。

- Routing：该模块负责将解析 SQL 生成的执行计划转换成 cube 缓存的查询，cube 是通过预计算缓存在 hbase 中，这部分查询是可以再秒级甚至毫秒级完成，而还有一些操作使用过查询原始数据（存储在 hadoop 上通过 hive 上查询），这部分查询的延迟比较高。

- Metadata：kylin 中有大量的元数据信息，包括 cube 的定义，星状模型的定义、job 的信息、job 的输出信息、维度的 directory 信息等等，元数据和 cube 都存储在 hbase 中，存储的格式是 json 字符串，除此之外，还可以选择将元数据存储在本地文件系统。

- Cube 构建引擎：这个模块是**所有模块的基础**，它负责预计算创建 cube，创建的过程是通过 hive 读取原始数据然后通过一些 mapreduce 计算生成 Htable

---

然后 load 到 hbase 中。

## 2.2、KYLIN 相关名词解释

Here are some domain terms we are using in Apache Kylin, please check them for your reference. They are basic knowledge of Apache Kylin which also will help to well understand such concept, term, knowledge, theory and others about Data Warehouse, Business Intelligence for analytics.

- **Data Warehouse:** a data warehouse (DW or DWH), also known as an enterprise data warehouse (EDW), is a system used for reporting and data analysis, [wikipedia](#)

- **Business Intelligence:** Business intelligence (BI) is the set of techniques and tools for the transformation of raw data into meaningful and useful information for business analysis purposes, [wikipedia](#)

- **OLAP:** OLAP is an acronym for online analytical processing

- **OLAP Cube:** an OLAP cube is an array of data understood in terms of its 0 or more dimensions, [wikipedia](#)

- **Star Schema:** the star schema consists of one or more fact tables referencing any number of dimension tables, [wikipedia](#)

- **Fact Table:** a Fact table consists of the measurements, metrics or facts of a business process, [wikipedia](#)

- **Lookup Table:** a lookup table is an array that replaces runtime computation with a simpler array indexing operation, [wikipedia](#)

- **Dimension:** A dimension is a structure that categorizes facts and measures in order to enable users to answer business questions. Commonly used dimensions are people, products, place and time, [wikipedia](#)

- **Measure:** a measure is a property on which calculations (e.g., sum, count, average, minimum, maximum) can be made, [wikipedia](#)

- **Join:** a SQL join clause combines records from two or more tables in a relational database, [wikipedia](#)

## 2.3、关键流程

在 kylin 中,最关键的两个流程是 cube 的预计算过程和 SQL 查询转换成 cube

---

的过程，cube 的构造可以分成 cube 的构建和 cube 的合并，首先需要创建一个 cube 的定义，包括设置 cube 名、cube 的星状模型结构，dimension 信息、measure 信息、设置 where 条件、根据 hive 中事实表定义的 partition 设置增量 cube，设置 rowkey 等信息，这些设置在 mondrian 中也是可以看到的，一个 cube 包含一些 dimension 和 measure，where 条件决定了源数据的大小，在 mondrian 中可以通过 view 实现。另外，kylin 还提供了增量计算的功能，虽然达不到实时计算的需求，但是基本上可以满足数据分析的需求。

查询解析过程主要是使用 Calcite 框架将用户输入的 SQL 解析并转换成对 hbase 的 key-value 查询操作以获取结果，但是经过使用发现它对 SQL 的支持是比较差的，所有的 SQL 不能使用 from A,B where xxx 之类的 join 方式，必须使用 inner (left、right) join on 的方式，否则解析就会出错，这就会导致 mondrian 生成的 SQL 压根不能使用 kylin 查询（因为 mondrian 生成的 SQL 是前面一种方式的），另外还有一个局限性就是发现只能对 cube 相关的表和列进行查询，例如根据维度进行 group by 查询定义的度量信息，而其他的查询也统统的返回错误，这点倒也不算是很大的问题，毕竟 cube 的模型已经定义，我们不太可能查询这个模型以外的东西。还有一点有点不能接受的是 **kylin 对于子查询的支持很弱**，测试发现查询的结果经常返回空（没有一行），而相同的查询在 hive 中是有结果的，这对于一些产品方需求支持不是很好，例如产品方可能需要查询年销售额大于 xx 的地区的每个月的销售总额。我们一般情况下会写出这样的 sql: `select month, sum(sales) from fact where location in (select location from fact group by year having sum(sales) > 1000) group by month;`前一段时间测试发现这种 SQL 对于关系数据库简直是灾难，因为 in 语句会导致后面的子查询没有缓存结果，而写成 `select month, sum(sales) from fact as A inner join (select location from fact group by year having sum(sales) > 1000) as B on A.location = B.location group by month;`可以提高性能，但是测试发现 kylin 返回的结果为空，而 kylin 对于 in 语句的查询时非常高效的（毕竟全部走缓存），那么我们就不得不首先执行子查询得到 location 集合，然后再写一个 SQL 使用 `where location in xxx`（kylin 对于使用 in 子句的查询支持还是相当棒的）的方式获得结果，这个应该是需要改进的地方吧。

## 2.4、Cube 模型

前面介绍了 cube 在创建过程中需要的设置，这里看一下每一个设置的具体含义吧，首先我们会设置 cube 名和 notification 列表，前者需要保证是全局唯一的，后者是一些 Email 用于通知 cube 的一些事件的发生。接着我们需要定义一个星状模型，和一般的数据仓库模型一样，**需要指定一个事实表和任意多个维度表**，如果存在维度表还需要指定事实表和维度表的关联关系，也就是 join 方式。接下来是定义 dimension，在定义 dimension 的时候可以选择 dimension 的类型，分为 Normal、Hierachy（层次化）以及 Derived，这个后面再进行介绍，dimension 的定义决定着 cube 的大小，也需要用户对原始的表非常了解。

接下来是定义 measure，kylin 会为每一个 cube 创建一个聚合函数为 count(1)的度量，它不需要关联任何列，用户自定义的度量可以选择 SUM、COUNT、DISTINCT COUNT、MIN、MAX，而每一个度量定义时还可以选择这些聚合函数的参数，可以选择常量或者事实表的某一列，一般情况下我们当然选择某一列。这里我们发现 kylin 并不提供 AVG 等相对较复杂的聚合函数（方差、平均差更没有了），主要是因为它需要基于缓存的 cube 做增量计算并且合并成新的 cube，而这些复杂的聚合函数并不能简单的对两个值计算之后得到新的值，例如需要增量合并的两个 cube 中某一个 key 对应的 sum 值分别为 A 和 B，那么合并之后的则

---

为 A+B,而如果此时的聚合函数是 AVG,那么我们必须知道这个 key 的 count 和 sum 之后才能做聚合。这就要求使用者必须自己想办法自己计算了。

定义完 measure 之后需要设置 where 条件,这一步是对原始数据进行过滤,例如我们设定销售额小于 xxx 的地区不在于本次分析范围之内,那么就可以在 where 条件里设定 location in xxx (子查询),那么生成的 cube 会过滤掉这些 location,这一步其实相当于对无效数据的清洗,但是在 kylin 中这个是会固化的,不容易改变,例如我今天希望将销售额小于 xx 的地区清洗掉,明天可能有想将年消费小于 xxx 的用户去除,这就需要每次都创建一个相同的 cube,区别仅仅在于 where 条件,他们之间会有很多的重复缓存数据,也会导致存储空间的浪费,但这也是 MOLAP 系统不可避免的,因此当过滤条件变化比较多时,更好的方案则是创建一个完整的 cube (不设置任何 where 条件),使用子查询的方式过滤掉不希望要的一些维度成员。

接下来的一步是设置增量 cube 信息,首先需要选择事实表中的某一个时间类型的分区列(貌似只能是按照天进行分区),然后再指定本次构建的 cube 的时间范围(起始时间点和结束时间点),这一步的结果会作为原始数据查询的 where 条件,保证本次构建的 cube 只包含这个闭区间时间内的数据,如果事实表没有时间类型的分区别或者没有选择任何分区则表示数据不会动态更新,也就不可以增量的创建 cube 了。

最后一步设置 rowkey,这一步的建议是看看就可以了,不要进行修改,除非对 kylin 内部实现有比较深的理解才能知道怎么去修改。当然这里有一个可以修改的是 mandatory dimension,如果一个维度需要在每次查询的时候都出现,那么可以设置这个 dimension 为 mandatory,可以省去很多存储空间,另外还可以对所有维度进行划分 group,不会组合查询的 dimension 可以划分在不同的 group 中,这样也会降低存储空间。

## 2.5、Dimension 介绍

在一个多维数据集合中,维度的个数决定着维度之间可能的组合数,而每一个维度中成员集合的大小决定着每一个可能的组合的个数,例如有三个普通的维度 A、B、C,他们的不同成员数分别为 10/100/1000,那么一个维度的组合有 2 的 3 次方个,分别是{空、A、B、C、AB、BC、AC、ABC},每一个成员我们称为 cuboid (维度的组合),而这些集合的成员组合个数分别为 1、10、100、1000、10\*100、100\*1000、10\*1000 和 10\*100\*1000。我们称每一个 dimension 中不同成员个数为 cardinality,我们要尽量避免存储 cardinality 比较高的维度的组合,在上面的例子中我们可以不缓存 BC 和 C 这两个 cuboid,可以通过计算的方式通过 ABC 中成员的值计算出 BC 或者 C 中某个成员组合的值,这相当于是时间和空间的一个权衡吧。

在 kylin 中存在的四种维度是为了减少 cuboid 的个数,而不是每一个维度是否缓存的,当前 kylin 是对所有的 cuboid 中的所有组合都进行计算和存储的,对于普通的 dimension,从上面的例子中可以看出 N 个维度的 cuboid 个数为 2 的 N 次方,而 kylin 中设置了一些维度可以减少 cuboid 个数,当然,这需要使用者对自己需要的维度十分了解,知道自己可能根据什么进行 group by。

### 2.5.1、Mandatory 维度

这种维度意味着每次查询的 group by 中都会携带的,将某一个 dimension 设置为 mandatory 可以将 cuboid 的个数减少一半,如下图:

- Mandatory dimension cuts cuboid combinations by half.

Normal Dimensions			A is Mandatory		
A	B	C	A	B	C
A	B	-	A	B	-
-	B	C	A	B	C
A	-	C	A	-	C
A	-	-	A	-	-
-	B	-	-	B	-
-	-	C	-	-	C
-	-	-	-	-	-

mandatory dimension

这是因为我们确定每一次 group by 都会携带 A，那么就可以省去所有不包含 A 这个维度的 cuboid 了。

## 2.5.2、Hierarchy 维度

这种维度是最常见的，尤其是在 mondrian 中，我们对于多维数据的操作经常会有上卷下钻之类的操作，这也就需要要求维度之间有层级关系，例如国家、省、城市，年、季度、月等。有层级关系的维度也可以大大减少 cuboid 的个数。如下图：

- Hierarchy dimension reduces combination from  $2^N$  to  $N+1$ .

Normal Dimensions			A->B->C is Hierarchy		
A	B	C	A	B	C
A	B	-	A	B	-
-	B	C	A	-	C
A	-	C	-	-	-
A	-	-	-	-	-
-	B	-	-	-	-
-	-	C	-	-	-
-	-	-	-	-	-

hierarchy dimension

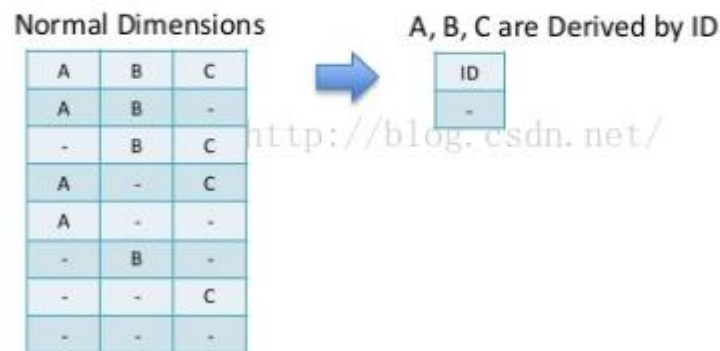
这里仅仅局限于 A/B/C 是一个层级，例如 A 是年份、B 是季度、C 是月份，那么查询的时候可能的组合只有年、xx 年的季度、xx 年 xx 季度的 xx 月，这就意味着我们不能再单独的对季度和月份进行聚合了，例如我们查询的时候不能使用 group by month，而必须使用 group by year, quart, month。如果需要单独的对 month 进行聚合，那么还需要再使用 month 列定义一个单独的普通维度。

## 2.5.3、Derived 维度



这类维度的意思是可推导的维度，需要该维度对应的一个或者多个列可以和维度表的主键是一一对应的，这种维度可以大大减少 cuboid 个数，如下图：

- **Derived dimension reduces combination from  $2^N$  to 2 at the cost of extra runtime aggregation.**



derived dimension

例如 timeid 是时间这个维度表的主键，也就是事实表的外键，时间只精确到天，那么 year、month、day 三列可以唯一对应着一个 time\_id，而 time\_id 是事实表的外键，那么我们可以指定 year、month、day 为一个 derived 维度，实际存储的时候可以只根据 timeid 的取值决定维度的组合，但这这就要求我们在查询的时候使用的 group by 必须指定 derived 维度集合中的所有列。

最后，简单介绍一下如何计算 cuboid 个数的，假设我们存在两个普通维度 brand、product，存在一个 hierarchy，包含四个维度分别为 year、quart、month 和 day，一个 derived 维度，指定 location 信息，包含 country、province 和 city 列，这相当于一共 9 个维度，但是根据上面的分析我们并不需要 512 分 cuboid。

第 0 层的 cuboid（不包含任何维度，不包含 group by），cuboid 的个数为 1，这个 cuboid 的成员个数也为 1；

第 1 层的 cuboid 包含一个维度，一共有 4 种组合（分别为 brand、product、year、location，因为 quart 是 hierarchy 的第二个层级，不能单独 group by，而 location 的三列可以视为一个整体），成员个数则有每一个维度的 cardinality；

第 2 层的 cuboid 有 7 种，分别为 {brand、product}、{brand、year}、{brand、location}、{product、year}、{product、location}、{year、location} 和 {year、quart}；

第 3 层的 cuboid 有 8 种，分别为 {brand、product、year}、{brand、product、location}、{product、year、location}、{brand、year、location}、{brand、year、quart}、{product、year、quart}、{location、year、quart}、{year、quart、month}；

第 4 层的 cuboid 有 8 种，分别为 {brand、product、year、location}、{brand、product、year、quart}、{brand、location、year、quart}、{product、location、year、quart}、{brand、year、quart、month}、{product、year、quart、month}、{location、year、quart、month}、{year、quart、month、day}

---

第 5 层的 cuboid 有 7 种, 分别为 {brand、product、year、quart、location}、{brand、product、year、quart、momth}、{brand、location、year、quart、month}、{product、location、year、quart、month}、{brand、year、quart、month、day}、{product、year、quart、month、day}、{location、year、quart、month、day}

第 6 层的 cuboid 有 5 种, 分别为 {brand、product、year、quart、month、location}、{brand、product、year、quart、momth、day}、{brand、location、year、quart、month、day}、{product、location、year、quart、month、day}

第 7 层的 cuboid 有 1 中, 为 {brand、product、year、quart、month、day、location}

所以一共 40 个 cuboid (kylin 计算的是 39 个, 应该没有把第 0 层的计算在内)。

## 2.6、增量 Cube

由于 kylin 的核心在于**预计算缓存数据**, 那么对于实时的数据查询的支持就不如 mondrian 好了, 但是一般情况下我们数据分析并没有完全实时的要求, 数据延迟几个小时甚至一天是可以接受的, kylin 提供了增量 cube 的接口, kylin 的实现是一个 cube (这里是指逻辑上的 cube) 中可以包含多个 segment, 每一个 segment 对应着一个物理 cube, 在实际存储上对应着一个 hbase 的一个表, 用户定义根据某一个字段进行增量 (目前仅支持时间, 并且这个字段必须是 hive 的一个分区字段), 在使用的时候首先需要定义好 cube 的定义, 可以指定一个时间的 partition 字段作为增量 cube 的依赖字段, 其实这个选择是作为原始数据选择的条件, 例如选择起始时间 A 到 B 的数据那么创建的 cube 则会只包含这个时间段的数据聚合值, 创建完一个 cube 之后可以再次基于以前的 cube 进行 build, 每次 build 会生成一个新的 segment, 只不过原始数据不一样了 (根据每次 build 指定的时间区间), 每次查询的时候会查询所有的 segment 聚合之后的值进行返回, 有点类似于 tablet 的存储方式, 但是当 segment 存在过多的时候查询效率就会下降, 因此需要在存在多个 segment 的时候将它们进行合并, 合并的时候其实是指定了一个时间区间, 内部会选择这个时间区间内的所有 segment 进行合并, 合并完成之后使用新的 segment 替换被合并的多个 segment, 合并的执行是非常迅速的, 数据不需要再从 HDFS 中获取, 直接将两个 hbase 表中相同 key 的数据进行聚合就可以了。但是有一点需要注意的是当合并完成之后, 被合并的几个 segment 所对应的 hbase 表并没有被删除。实际的使用过程中对于增量的 cube 可以写个定时任务每天凌晨进行 build, 当达到一个数目之后进行 merge (其实每次 build 完成之后都进行 merge 也是可以的)。

## 2.7、Cube 的词典树

Kylin 的 cube 数据是作为 key-value 结构存储在 hbase 中的, key 是每一个维度成员的组合值, 不同的 cuboid 下面的 key 的结构是不一样的, 例如 cuboid={brand, product, year} 下面的一个 key 可能是 brand='Nike', product='shoe', year=2015, 那么这个 key 就可以写成 Nike:shoe:2015, 但是如果使用这种方式的话会出现很多重复, 所以一般情况下我们会把一个维度下的所有成员取出来, 然后保存在一个数组里面, 使用数组的下标组合成为一个 key, 这样可以大大节省 key 的存储空间, kylin 也使用了相同的方法, 只不过使用了字典树 (Trie 树), 每一个维度的字典树作为 cube 的元数据以二进制的方式存储在 hbase 中, 内存中也



会一直保持一份。

## 总结:

以上介绍了 kylin 的整体框架以及部分的模块的流程, 由于之前主要是关注 cube 的一些操作, 例如创建、构建、合并等, 对于查询这一块了解的较少, 当然, 这一块是 kylin 的核心之一。接下来会从源代码的角度去看 kylin 是如何构建和 mergecube 的, 以及执行查询的流程。

## 3、KYLIN 搭建以及测试（单节点）

### 3.1、官方安装说明

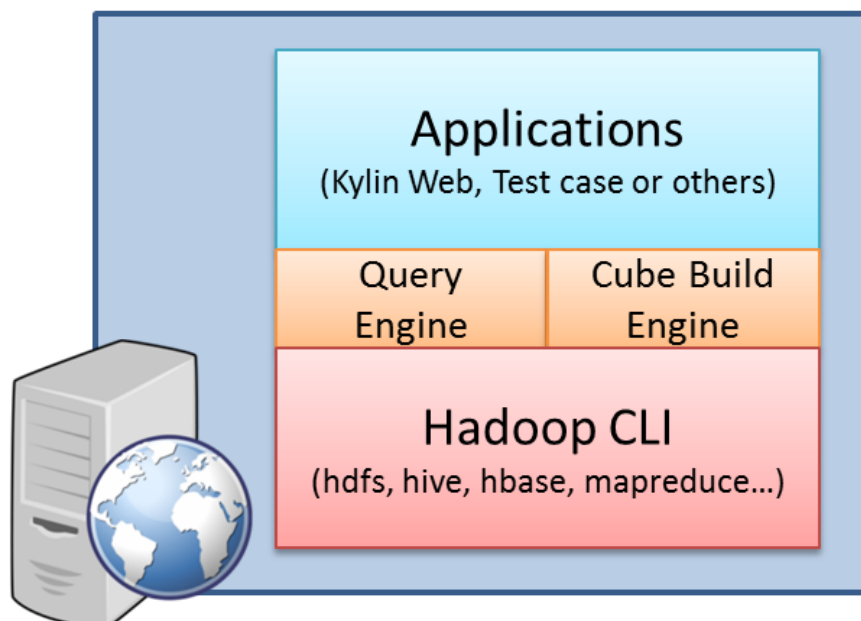
官方说法:

#### Installation Guide

##### Environment

Kylin requires a properly setup Hadoop environment to run. Following are the minimal request to run Kylin, for more detial, please check [Hadoop Enviroment](#).

It is most common to install Kylin on a Hadoop client machine, from which Kylin can talk with the Hadoop cluster via command lines including `hive`, `hbase`, `hadoop`, etc. The scenario is depicted as:



For normal use cases, the application in the above picture means Kylin Web, which contains a web interface for cube building, querying and all sorts of management. Kylin Web launches a query engine for querying and a cube build engine for building cubes. These two engines interact with the Hadoop components, like hive and hbase.

Except for some prerequisite software installations, the core of Kylin installation is accomplished by running a single script. After running the script, you will be able to build sample cube and query the tables behind the cubes via a unified web interface.

### Install Kylin

1. Download latest Kylin binaries at <http://kylin.apache.org/download>
2. Export KYLIN\_HOME pointing to the extracted Kylin folder
3. Make sure the user has the privilege to run hadoop, hive and hbase cmd in shell. If you are not so sure, you can run `bin/check-env.sh`, it will print out the detail information if you have some environment issues.
4. To start Kylin, run `bin/kylin.sh start`, after the server starts, you can watch logs/kylin.log for runtime logs;
5. To stop Kylin, run `bin/kylin.sh stop`

If you want to have multiple Kylin nodes running to provide high availability, please refer to [this](#)

After Kylin started you can visit <http://hostname:7070/kylin>. The default username/password is ADMIN/KYLIN. It's a clean Kylin homepage with nothing in there. To start with you can:

1. [Quick play with a sample cube](#)
2. [Create and Build a cube](#)
3. [Kylin Web Tutorial](#)

## 3.2、系统依赖要求

### Hadoop Environment

Kylin need run in a Hadoop node, to get better stability, we suggest you to deploy it a pure Hadoop client machine, on which it the command lines like `hive`, `hbase`, `hadoop`, `hdfs` already be installed and configured. The Linux account that running Kylin has got permission to the Hadoop cluster, including create/write hdfs, hive tables, hbase tables and submit MR jobs.

### Recommended Hadoop Versions

- Hadoop: 2.6 - 2.7
- Hive: 0.13 - 1.2.1
- HBase: 0.98 - 0.99, 1.x
- JDK: 1.7+

*Tested with Hortonworks HDP 2.2 and Cloudera Quickstart VM 5.1*

To make things easier we strongly recommend you try Kylin with an all-in-one sandbox VM, like [HDP sandbox](#). In the following tutorial we'll go with [Hortonworks Sandbox 2.1](#) and [Cloudera QuickStart VM 5.1](#).

To avoid permission issue in the sandbox, you can use its `root` account. The password for [Hortonworks Sandbox 2.1](#) is `hadoop`, for [Cloudera QuickStart VM 5.1](#) is `cloudera`.

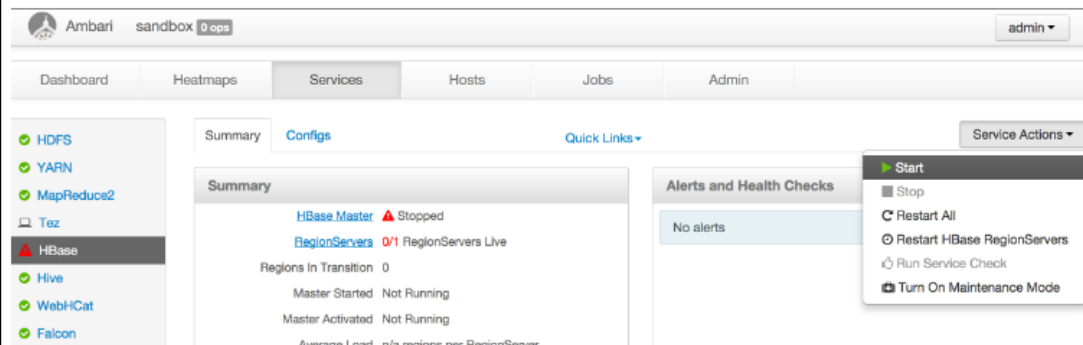
We also suggest you using bridged mode instead of NAT mode in Virtual Box settings. Bridged mode will assign your sandbox an independent IP address so that you can avoid issues like [this](#).

## Start Hadoop

Use ambari helps to launch hadoop:

```
ambari-agent start
ambari-server start
```

With both command successfully run you can go to ambari homepage at [http://your\\_sandbox\\_ip:8080](http://your_sandbox_ip:8080) (user:admin,password:admin) to check everything's status. By default hortonworks ambari disables Hbase, you need manually start the Hbase service at ambari homepage.



The screenshot shows the Ambari web interface for a sandbox environment. The 'Services' tab is selected, and the 'HBase' service is highlighted in the left sidebar. The 'Summary' section shows the HBase Master is 'Stopped' and RegionServers are in 'Transition'. The 'Service Actions' dropdown menu is open, showing options like 'Start', 'Stop', 'Restart All', 'Restart HBase RegionServers', 'Run Service Check', and 'Turn On Maintenance Mode'.

### Additional Info for setting up Hortonworks Sandbox on Virtual Box

Please make sure Hbase Master port [Default 60000] and Zookeeper [Default 2181] is forwarded to Host OS.

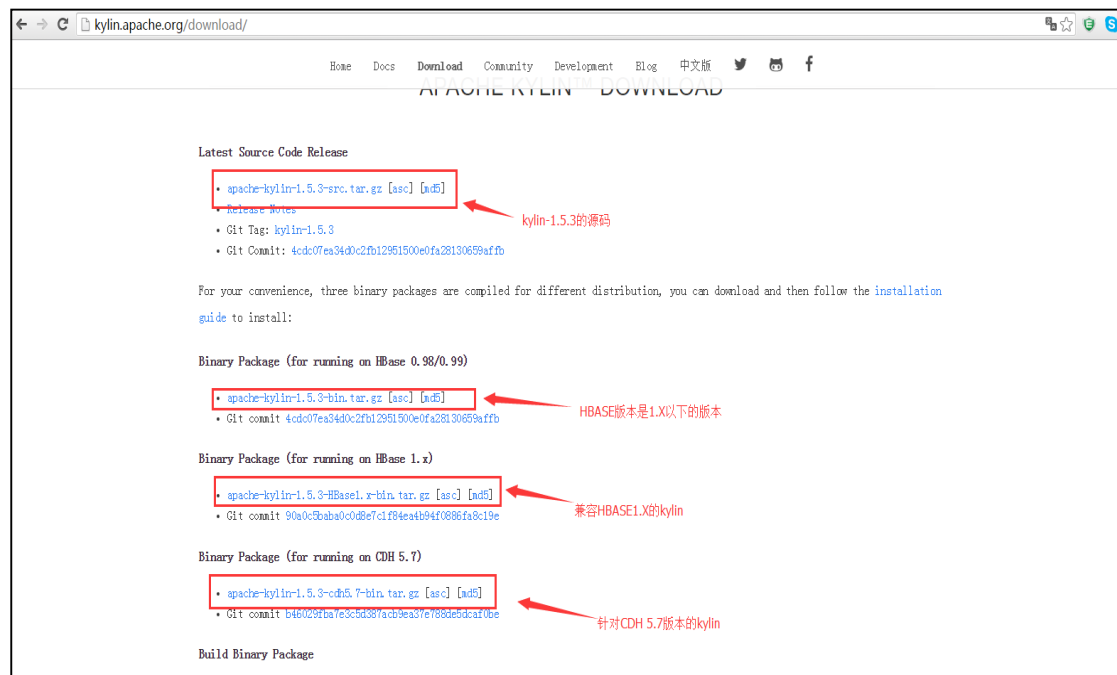
目前个人集群环境如下（基于 CDH 5.7）



## 3.3.、KYLIN 单节点安装

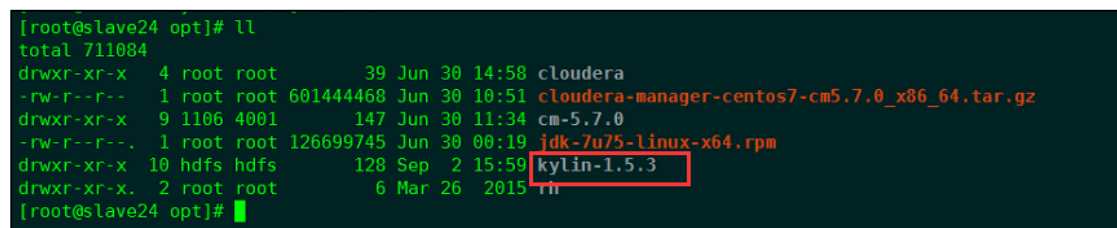
### 3.3.1、Kylin 下载

<http://kylin.apache.org/download/>

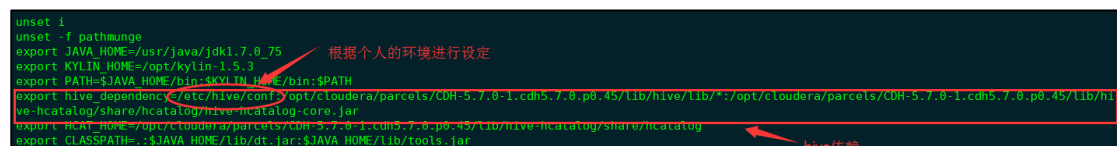


### 3.3.2、Kylin 安装

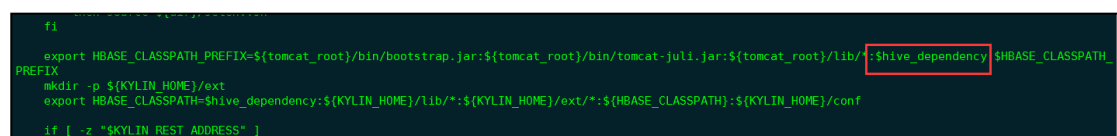
1) 解压 kylin 到要安装的集群节点



2) 配置 kylin 环境变量以及依赖的环境变量



3) 修改 bin 下的 kylin 启动文件加入环境变量中的 hive\_dependency 依赖



#### 4) 修改 Kylin 配置文件

```
[root@slave24 opt]# cd kylin-1.5.3/conf/
[root@slave24 conf]# vi kylin.properties
[root@slave24 conf]#
```

#### 5) 由于 kylin 用户都是 hbase 的，需要修改 kylin 的执行权限为 hdfs

chown -R hdfs.hdfs /usr/local/kylin

如若不修改的话默认的所属用户以及组为 root

在启动时会报错：权限不足，写入等权限不足

#### 6) 集群环境检查

执行 bin 下的 check-env.sh

如若环境无误只会显示 kylin 的 kylin\_home 信息

```
[root@slave24 bin]# ./check-env.sh
KYLIN_HOME is set to /opt/kylin-1.5.3
[root@slave24 bin]#
```

#### 7) 启动 kylin

进入 kylin 目录下执行：

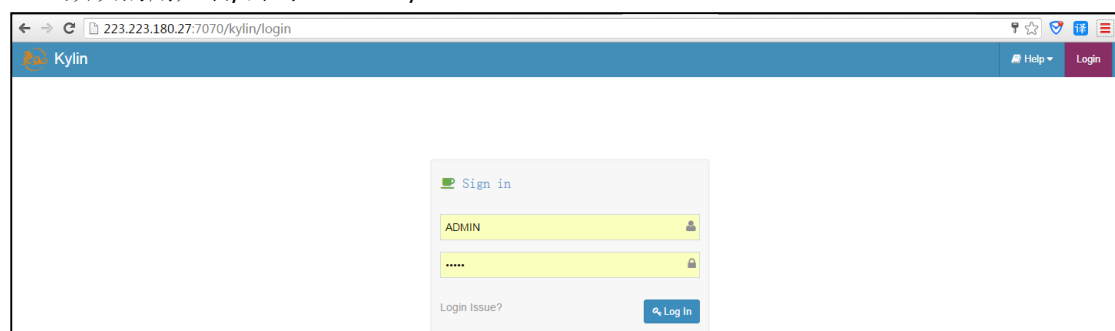
bin/kylin.sh start

启动成功后进入 8)，有问题再查看日志进行错误跟踪

#### 8) 进入 kylin 页面查看

地址为：<http://<安装 kylin 节点的机器 IP>:7070/kylin/login>

默认的用户名/密码：ADMIN/KYLIN



#### 9) 测试数据运行

- a) 停掉 kylin
- b) 在 bin 下执行:  
bin/sample.sh
- c) 查看生产的表 (hive+hbase)

Hive:

```
hive> show tables;  
OK  
customers  
kylin_cal_dt  
kylin_category_groupings  
kylin_intermediate_sample_cube_1_clone_20120101000000_20160831000000  
kylin_sales  
sample_0/  
sample_08  
t_hive  
web_logs  
Time taken: 2.321 seconds, Fetched: 9 row(s)  
hive>
```

默认hive生成该三张表

Hbase:

```
hbase(main):001:0> list  
TABLE  
KYLIN_56W60CON4T  
KYLIN_6JEQ398NDH  
KYLIN_7M5D1M0Q00  
KYLIN_G3YY0Y35TU  
KYLIN_G06IK3YUK0  
KYLIN_LS3NWC8SWT  
KYLIN_01WNSKZN45  
KYLIN_047IPS6PT0  
kylin_metadata  
kylin_metadata_acl  
kylin_metadata_user  
11 row(s) in 0.5836 seconds  
=> ["KYLIN_56W60CON4T", "KYLIN_6JEQ398NDH", "KYLIN_7M5D1M0Q00", "KYLIN_G3YY0Y35TU", "KYLIN_G06IK3YUK0", "KYLIN_LS3NWC8SWT", "KYLIN_01WNSKZN45", "KYLIN_047IPS6PT0", "kylin_metadata", "kylin_metadata_acl", "kylin_metadata_user"]  
hbase(main):002:0>
```

kylin元数据表

初始化的默认表

10) 再启动 kylin

11) 在界面可以看到

在界面按该操作进行操作

## Quick Start with Sample Cube

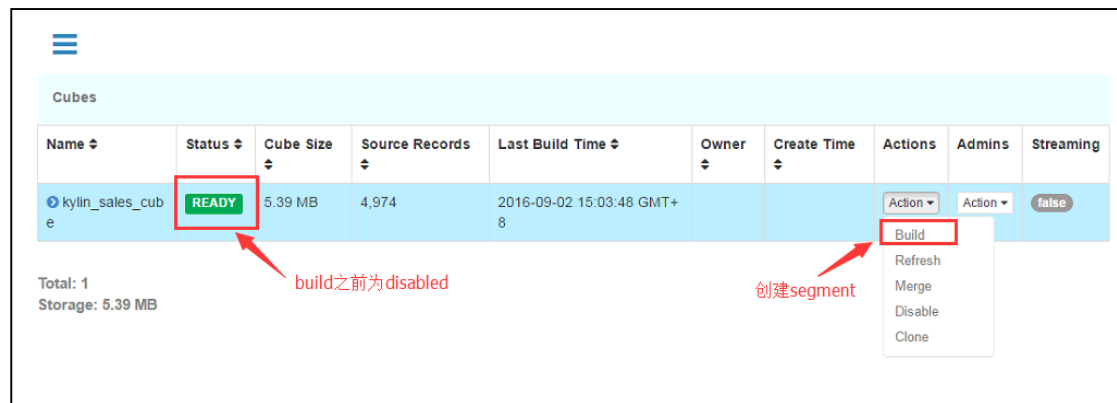
Kylin provides a script for you to create a sample Cube; the script will also create three sample hive tables:

1. Run `${KYLIN_HOME}/bin/sample.sh` ; Restart kylin server to flush the caches;
2. Logon Kylin web with default user ADMIN/KYLIN, select project "learn\_kylin" in the project dropdown list (left upper corner);
3. Select the sample cube "kylin\_sales\_cube", click "Actions" -> "Build", pick up a date later than 2014-01-01 (to cover all 10000 sample records);
4. Check the build progress in "Monitor" tab, until 100%;
5. Execute SQLs in the "Insight" tab, for example:  
`select part_dt, sum(price) as total_sold, count(distinct seller_id) as sellers from kylin_sales group by part_dt order by part_dt`
6. You can verify the query result and compare the response time with hive;

## What's next

You can create another cube with the sample tables, by following the tutorials.

成功后看到如下界面:



## 12) cube 创建后可以在 Monitor 界面进行进度查看

Job Name	Cube	Progress	Last Modified Time	Duration	Actions
kylin_sales_cube - 20120101000000_20120110021000 - BUILD - PDT 2016-06-23 20:07:30	kylin_sales_cube	100%	2016-06-23 19:47:51 PST	40.05 mins	Action
kylin_sales_cube - 20120101000000_20120110235500 - BUILD - PDT 2016-06-23 17:01:02	kylin_sales_cube	83.33%	2016-06-23 19:07:01 PST	179.13 mins	Action
kylin_sales_cube - 20120101000000_20120105063000 - BUILD - PDT 2016-06-23 10:03:40	kylin_sales_cube	77.78%	2016-06-23 16:00:22 PST	398.98 mins	Action
kylin_sales_cube - 20120101000000_20120202235500 - BUILD - PDT 2016-06-23 07:25:23	kylin_sales_cube	22.2%	2016-06-23 08:17:11 PST	109.38 mins	Action

## 13) cube 创建成功后可以进入 Insight 界面进行 sql 查询以及检查 hive 和 hbase 表变化

执行 sql:

```
select part_dt, sum(price) as total_sold, count(distinct seller_id) as sellers
from kylin_sales group by part_dt order by part_dt;
```

PART_DT	TOTAL_SOLD	SELLERS
2012-01-01	466.9037	12
2012-01-02	970.2347	17
2012-01-03	917.4138	14
2012-01-04	553.0541	10
2012-01-05	732.9007	18
2012-01-06	296.3882	9
2012-01-07	1184.1870	22
2012-01-08	541.7355	14

Build 成功后, hive 中建立了 3+n 个表, 如图所示 (3 个官网案例 hive 表, n 个 build 的 hive 表)



```
hive> show tables;
OK
kylin_cal_dt
kylin_category_groupings
kylin_intermediate_kylin_sales_cube_desc_20120101000000_20120105063000
kylin_intermediate_kylin_sales_cube_desc_20120101000000_20120110235500
kylin_intermediate_kylin_sales_cube_desc_20120101000000_20120202235500
kylin_sales
Time taken: 0.323 seconds, Fetched: 6 row(s)
hive>
```

Build 成功后, hbase 中建立了 1+n 个表, 如图所示(1 个元数据表, n 个 build 的 hbase 表)

```
hbase(main):029:0> list
TABLE
KYLIN_7932J1LD0I
KYLIN_JE63ERR0LR
KYLIN_KPVQ8J6VVK
KYLIN_O2EZ9WSOLO
KYLIN_TQVDJK5AP1
KYLIN_WJ510XZH4
kylin_metadata
tsnappy
8 row(s) in 0.0470 seconds

=> ["KYLIN_7932J1LD0I", "KYLIN_JE63ERR0LR", "KYLIN_KPVQ8J6VVK", "KYLIN_O2EZ9WSOLO", "KYLIN_TQVDJK5AP1", "KYLIN_WJ510XZH4", "kylin_metadata", "tsnappy"]
```

针对不同的 build 好的 segment 会在 hbase 中出现不同的 KYLIN\_XXXXX 表与其对应

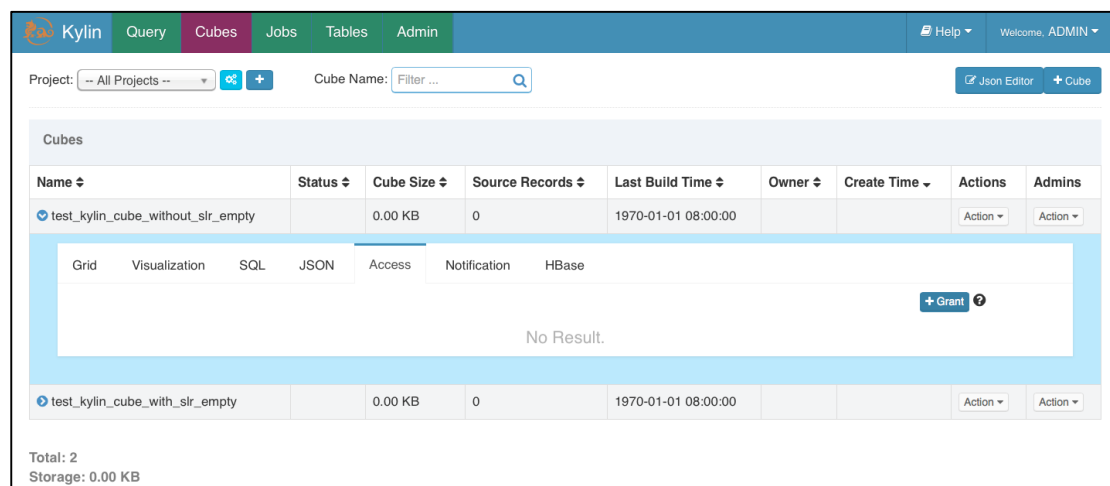
14) Project、Cube 的创建配置过程自行官网, 不再赘述



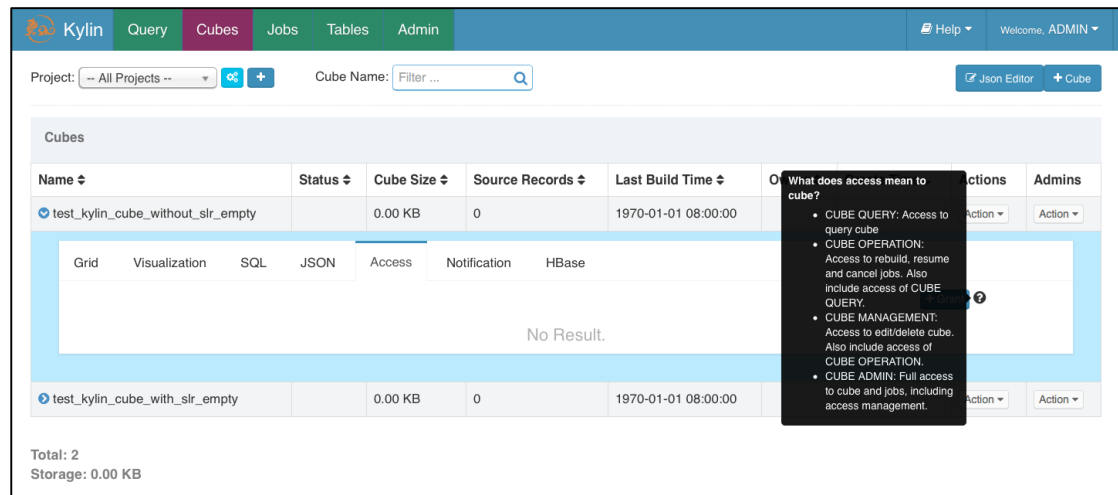
15) Cube 授权

在 Cubes 页面, 双击 cube 行查看详细信息。在这里我们关注 Access 标签。

点击+Grant 按钮进行授权。



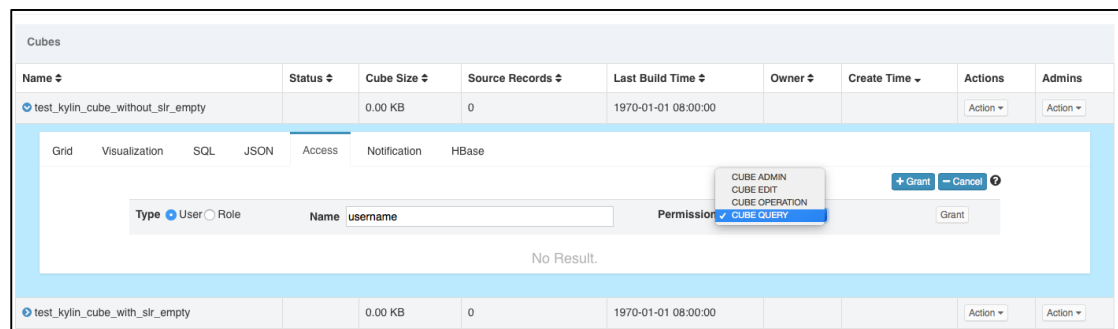
一个 cube 有四种不同的权限。将你的鼠标移动到?图标查看详细信息。



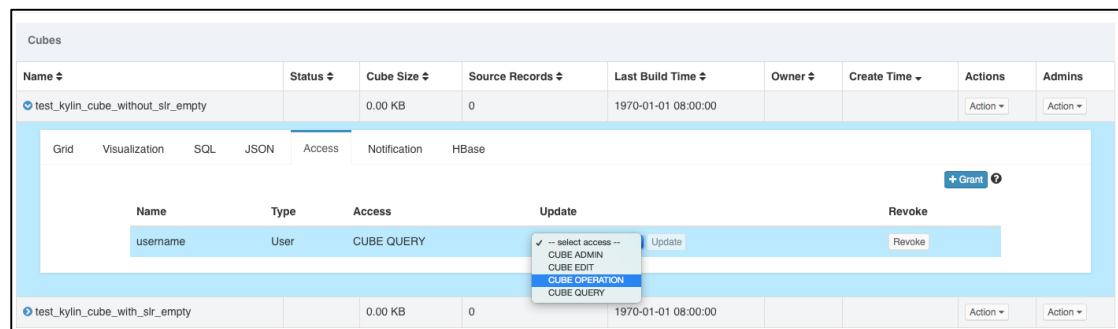
授权对象也有两种：User 和 Role。Role 是指一组拥有同样权限的用户。

## 1. 授予用户权限

- 选择 User 类型，输入你想要授权的用户的用户名并选择相应的权限。



-- 然后点击 **Grant** 按钮提交请求。在这一操作成功后，你会在表中看到一个新的表项。你可以选择不同的访问权限来修改用户权限。点击 **Revoke** 按钮可以删除一个拥有权限的用户。



## 2. 授予角色权限

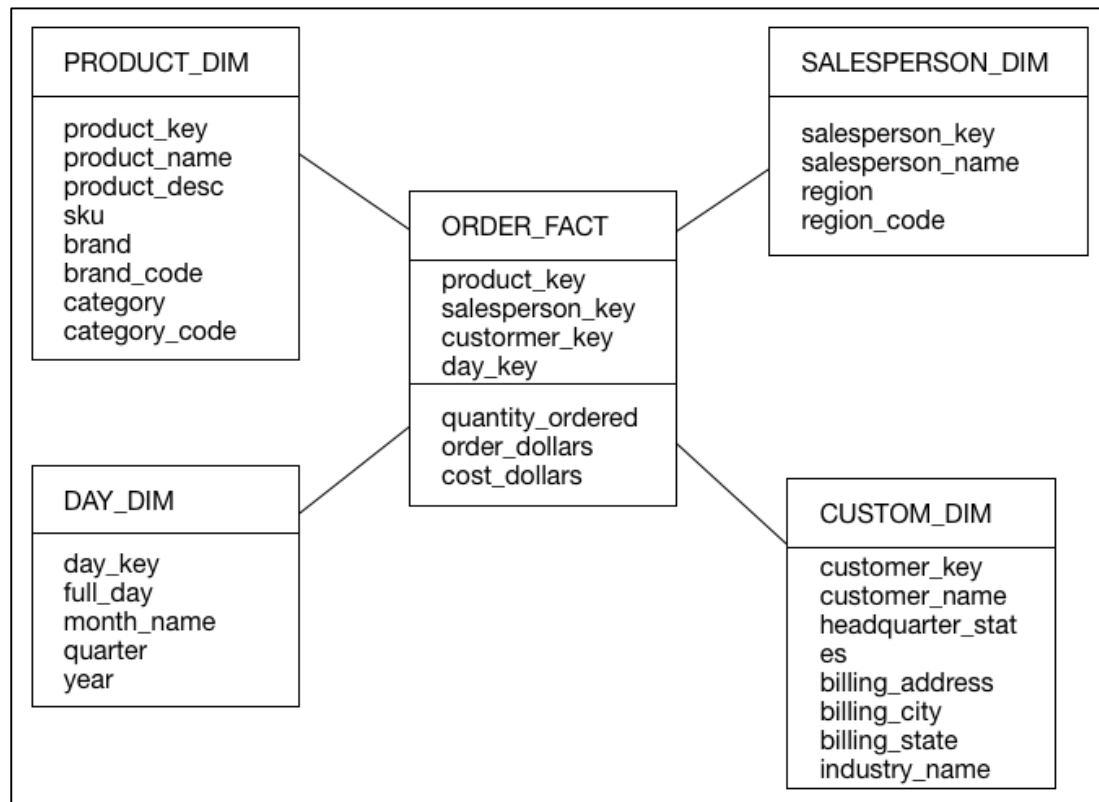
- 选择 Role 类型，通过点击下拉按钮选择你想要授权的一组用户并选择一个权限。
- 然后点击 **Grant** 按钮提交请求。在这一操作成功后，你会在表中看到一个新的表项。你可以选择不同的访问权限来修改组权限。点击 **Revoke** 按钮可以删除一个拥有权限的组。

## 总结：

至此，kylin 的单节点搭建以及 sample 测试数据的执行，简单的 project 创建和 cube 的创建已完成，实际情况还请自行搭建测试。

## 3.4、Cube 构建案例(仅做学习)

### 3.4.1、Hive 订单数据仓库构建



### 3.4.2、表的创建以及数据录入

#### 1) 创建事实表并插入数据

```
DROP TABLE IF EXISTS default.fact_order ;
```

```
create table default.fact_order (
    time_key string,
    product_key string,
    salesperson_key string,
    custom_key string,
    quantity_ordered bigint,
    order_dollars bigint,
    cost_dollars bigint
) ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```

```
load data local inpath '/root/kylinsample/fact_order.txt' overwrite into table
default.fact_order;
```

---

```
##load data local inpath '/root/kylinsample/fact_order.txt' into table default.fact_order;
```

**fact\_order.txt**

```
2016-05-01, pd001, sp001, ct001, 100, 2000, 1000
2016-05-01, pd001, sp002, ct002, 100, 2000, 1000
2016-05-01, pd001, sp003, ct002, 100, 2000, 1000
2016-05-01, pd002, sp002, ct002, 100, 2000, 1000
2016-05-01, pd003, sp003, ct001, 100, 2000, 1000
2016-05-01, pd001, sp003, ct001, 100, 2000, 1000
2016-05-01, pd001, sp002, ct001, 100, 2000, 1000
2016-05-01, pd001, sp003, ct002, 100, 2000, 1000
2016-05-01, pd002, sp001, ct001, 100, 2000, 1000
2016-05-01, pd003, sp001, ct001, 100, 2000, 1000
2016-05-01, pd004, sp001, ct001, 50, 1000, 600
2016-05-02, pd001, sp001, ct001, 50, 1000, 600
2016-05-02, pd001, sp002, ct002, 100, 2000, 1000
2016-05-02, pd001, sp003, ct002, 100, 2000, 1000
2016-05-02, pd002, sp001, ct001, 50, 1000, 600
2016-05-02, pd003, sp001, ct001, 50, 1000, 600
2016-05-02, pd004, sp001, ct001, 50, 1000, 600
2016-05-03, pd001, sp001, ct001, 50, 1000, 600
2016-05-03, pd001, sp002, ct002, 100, 2000, 1000
2016-05-03, pd001, sp003, ct002, 100, 2000, 1000
2016-05-04, pd002, sp001, ct001, 700, 14000, 10000
2016-05-04, pd003, sp001, ct001, 700, 14000, 10000
2016-05-04, pd004, sp001, ct001, 100, 2000, 1000
2016-05-05, pd001, sp001, ct001, 100, 2000, 1000
2016-05-05, pd001, sp002, ct002, 700, 14000, 10000
2016-05-05, pd001, sp003, ct002, 700, 14000, 10000
2016-05-05, pd002, sp001, ct001, 100, 2000, 1000
2016-05-05, pd003, sp001, ct001, 100, 2000, 1000
2016-05-05, pd004, sp001, ct001, 100, 2000, 1000
2016-05-06, pd001, sp001, ct001, 100, 2000, 1000
2016-05-06, pd001, sp002, ct002, 100, 2000, 1000
2016-05-06, pd001, sp003, ct002, 100, 2000, 1000
2016-05-07, pd002, sp001, ct001, 100, 2000, 1000
2016-05-07, pd003, sp001, ct001, 100, 2000, 1000
2016-05-07, pd004, sp001, ct001, 50, 1000, 600
2016-05-07, pd002, sp001, ct001, 100, 2000, 1000
2016-05-07, pd003, sp001, ct001, 100, 2000, 1000
2016-05-07, pd004, sp001, ct001, 50, 1000, 600
2016-05-08, pd001, sp001, ct001, 50, 1000, 600
2016-05-08, pd001, sp002, ct002, 100, 2000, 1000
```

---

2016-05-08, pd001, sp003, ct002, 100, 2000, 1000  
2016-05-08, pd001, sp001, ct001, 50, 1000, 600  
2016-05-08, pd001, sp002, ct002, 100, 2000, 1000  
2016-05-08, pd001, sp003, ct002, 100, 2000, 1000  
2016-05-08, pd001, sp001, ct001, 50, 1000, 600  
2016-05-08, pd001, sp002, ct002, 100, 2000, 1000  
2016-05-08, pd001, sp003, ct002, 100, 2000, 1000  
2016-05-09, pd002, sp001, ct001, 50, 1000, 600  
2016-05-09, pd003, sp001, ct001, 50, 1000, 600  
2016-05-09, pd004, sp001, ct001, 50, 1000, 600  
2016-05-09, pd001, sp001, ct001, 50, 1000, 600  
2016-05-09, pd002, sp001, ct001, 50, 1000, 600  
2016-05-09, pd003, sp001, ct001, 50, 1000, 600  
2016-05-09, pd004, sp001, ct001, 50, 1000, 600  
2016-05-09, pd001, sp001, ct001, 50, 1000, 600  
2016-05-09, pd001, sp002, ct002, 100, 2000, 1000  
2016-05-09, pd004, sp003, ct002, 100, 2000, 1000  
2016-05-09, pd002, sp001, ct001, 700, 14000, 10000  
2016-05-09, pd003, sp003, ct001, 700, 14000, 10000  
2016-05-09, pd004, sp003, ct001, 100, 2000, 1000  
2016-05-10, pd001, sp001, ct001, 100, 2000, 1000  
2016-05-10, pd001, sp002, ct002, 700, 14000, 10000  
2016-05-10, pd001, sp003, ct002, 700, 14000, 10000  
2016-05-10, pd002, sp001, ct001, 100, 2000, 1000  
2016-05-11, pd003, sp003, ct001, 100, 2000, 1000  
2016-05-11, pd004, sp001, ct001, 100, 2000, 1000  
2016-05-12, pd001, sp001, ct001, 100, 2000, 1000  
2016-05-12, pd004, sp002, ct002, 100, 2000, 1000  
2016-05-12, pd001, sp003, ct002, 100, 2000, 1000  
2016-05-12, pd001, sp001, ct001, 100, 2000, 1000  
2016-05-12, pd004, sp002, ct002, 100, 2000, 1000  
2016-05-12, pd001, sp003, ct002, 100, 2000, 1000  
2016-05-13, pd002, sp001, ct001, 100, 2000, 1000  
2016-05-13, pd003, sp001, ct001, 100, 2000, 1000  
2016-05-13, pd004, sp001, ct001, 50, 1000, 600  
2016-05-14, pd001, sp001, ct001, 50, 1000, 600  
2016-05-14, pd001, sp002, ct002, 100, 2000, 1000  
2016-05-14, pd001, sp003, ct002, 100, 2000, 1000  
2016-05-15, pd002, sp001, ct001, 50, 1000, 600  
2016-05-15, pd003, sp001, ct001, 50, 1000, 600  
2016-05-15, pd004, sp001, ct001, 50, 1000, 600  
2016-05-15, pd002, sp001, ct001, 50, 1000, 600  
2016-05-15, pd003, sp001, ct001, 50, 1000, 600  
2016-05-15, pd004, sp001, ct001, 50, 1000, 600

---

```
2016-05-15, pd002, sp001, ct001, 50, 1000, 600
2016-05-15, pd003, sp001, ct001, 50, 1000, 600
2016-05-15, pd004, sp001, ct001, 50, 1000, 600
2016-05-16, pd001, sp001, ct001, 50, 1000, 600
2016-05-16, pd001, sp002, ct002, 100, 2000, 1000
2016-05-16, pd001, sp003, ct002, 100, 2000, 1000
2016-05-16, pd001, sp001, ct001, 50, 1000, 600
2016-05-16, pd001, sp002, ct002, 100, 2000, 1000
2016-05-16, pd001, sp003, ct002, 100, 2000, 1000
2016-05-17, pd002, sp001, ct001, 700, 14000, 10000
2016-05-17, pd003, sp001, ct001, 700, 14000, 10000
2016-05-17, pd004, sp001, ct001, 100, 2000, 1000
2016-05-17, pd002, sp001, ct001, 700, 14000, 10000
2016-05-17, pd003, sp001, ct001, 700, 14000, 10000
2016-05-17, pd004, sp001, ct001, 100, 2000, 1000
2016-05-18, pd001, sp001, ct001, 100, 2000, 1000
2016-05-18, pd003, sp002, ct001, 700, 14000, 10000
2016-05-18, pd001, sp003, ct002, 700, 14000, 10000
2016-05-19, pd002, sp001, ct001, 100, 2000, 1000
2016-05-19, pd003, sp001, ct002, 100, 2000, 1000
2016-05-20, pd001, sp001, ct001, 100, 2000, 1000
2016-05-20, pd002, sp002, ct002, 100, 2000, 1000
2016-05-20, pd003, sp003, ct001, 100, 2000, 1000
2016-05-20, pd004, sp001, ct001, 100, 2000, 1000
2016-05-20, pd001, sp002, ct002, 100, 2000, 1000
2016-05-20, pd002, sp001, ct002, 100, 2000, 1000
```

## 2) 创建天维度表 dim\_day

```
DROP TABLE IF EXISTS default.dim_day ;
```

```
create table default.dim_day (
```

```
    day_key string,
    full_day string,
    month_name string,
    quarter string,
    year string
```

```
)ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
```

```
STORED AS TEXTFILE;
```

```
load data local inpath '/root/kylinsample/dim_day.txt' overwrite into table default.dim_day;
```

### dim\_day.txt

```
2016-05-01, 2016-05-01, 201605, 2016q2, 2016
2016-05-02, 2016-05-02, 201605, 2016q2, 2016
2016-05-03, 2016-05-03, 201605, 2016q2, 2016
```

---

2016-05-04, 2016-05-04, 201605, 2016q2, 2016  
2016-05-05, 2016-05-05, 201605, 2016q2, 2016  
2016-05-06, 2016-05-06, 201605, 2016q2, 2016  
2016-05-07, 2016-05-07, 201605, 2016q2, 2016  
2016-05-08, 2016-05-08, 201605, 2016q2, 2016  
2016-05-09, 2016-05-09, 201605, 2016q2, 2016  
2016-05-10, 2016-05-10, 201605, 2016q2, 2016  
2016-05-11, 2016-05-11, 201605, 2016q2, 2016  
2016-05-12, 2016-05-12, 201605, 2016q2, 2016  
2016-05-13, 2016-05-13, 201605, 2016q2, 2016  
2016-05-14, 2016-05-14, 201605, 2016q2, 2016  
2016-05-15, 2016-05-15, 201605, 2016q2, 2016  
2016-05-16, 2016-05-16, 201605, 2016q2, 2016  
2016-05-17, 2016-05-17, 201605, 2016q2, 2016  
2016-05-18, 2016-05-18, 201605, 2016q2, 2016  
2016-05-19, 2016-05-19, 201605, 2016q2, 2016  
2016-05-20, 2016-05-20, 201605, 2016q2, 2016

### 3) 创建售卖员的维度表 **salesperson\_dim**

DROP TABLE IF EXISTS default.dim\_salesperson ;

```
create table default.dim_salesperson (  
    salesperson_key string,  
    salesperson string,  
    salesperson_id string,  
    region string,  
    region_code string  
)ROW FORMAT DELIMITED FIELDS TERMINATED BY ','  
STORED AS TEXTFILE;
```

load data local inpath '/root/kylin-sample/dim\_salesperson.txt' overwrite into  
table default.dim\_salesperson;

#### **dim\_salesperson.txt**

sp001,hongbin,sp001,beijing,10086  
sp002,hongming,sp002,beijing,10086  
sp003,hongmei,sp003,beijing,10086

### 4) 创建客户维度 **custom\_dim**

DROP TABLE IF EXISTS default.dim\_custom ;

```
create table default.dim_custom (  
    custom_key string,  
    custom_name string,  
    custom_id string,
```



---

```
    headquarter_states string,
    billing_address string,
    billing_city string,
    billing_state string,
    industry_name string
)ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```

```
load data local inpath '/root/kylinsample/dim_custom.txt' overwrite into table
default.dim_custom;
```

#### **dim\_custom.txt**

```
ct001, custom_john, ct001, beijing, zgx-beijing, beijing, beijing, internet
ct002, custom_herry, ct002, henan, shlinjie, shangdang, henan, internet
```

#### **5) 创建产品维度表并插入数据**

```
DROP TABLE IF EXISTS default.dim_product ;
```

```
create table default.dim_product (
    product_key string,
    product_name string,
    product_id string,
    product_desc string,
    sku string,
    brand string,
    brand_code string,
    brand_manager string,
    category string,
    category_code string
)ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```

```
load data local inpath '/root/kylinsample/dim_product.txt' overwrite into table
default.dim_product;
```

#### **dim\_product.txt**

```
pd001, Box-Large, pd001, Box-Large-des, large1.0, brand001, brandcode001, brandmanager
001, Packing, cate001
pd002, Box-Medium, pd001, Box-Medium-des, medium1.0, brand001, brandcode001, brandmana
ger001, Packing, cate001
pd003, Box-small, pd001, Box-small-des, small1.0, brand001, brandcode001, brandmanager
001, Packing, cate001
pd004, Envelope, pd001, Envelope_des, large3.0, brand001, brandcode001, brandmanager001,
```

---

Pens, cate002

这样一个星型的结构表在 hive 中创建完毕, 实际上一个离线的数据仓库已经完成, 它包含一个主题, 即商品订单

### 3.4.3、Kylin 的 Project 创建与数据同步

#### Project 创建:

- 1、单击"Manage Project"
- 2、单击"New Project"
- 3、输入"Project Name", Warehouse\_01
- 4、Submit

#### 数据同步:

- 1、选择 Warehouse\_01,选择"Data Source" tab 页
- 2、单击"Load Hive Table"
- 3、输入需要同步的表  
"DEFAULT.FACT\_ORDER,DEFAULT.DIM\_DAY,DEFAULT.DIM\_PRODUCT,DEFAULT.DIM\_SALES  
PERSON,DEFAULT.DIM\_CUSTOM"
- 4、Sync

### 3.4.4、Kylin 的 Model 创建

- 1、选择"Models" tab 页,单击"New Model"
- 2、"Model Name"输入,Warehouse\_01\_Model
- 3、选择"Fact Table"为 DEFAULT.FACT\_ORDER;再 添加 Lookup Table;
- 4、选取每张表的哪些列字段作为 Dimensions

ID	Table Name	Columns
1	DEFAULT.FACT_ORDER	TIME_KEY PRODUCT_KEY SALESPERSON_KEY CUSTOM_KEY
2	DEFAULT.DIM_DAY	FULL_DAY
3	DEFAULT.DIM_PRODUCT	PRODUCT_NAME
4	DEFAULT.DIM_SALESPERSON	SALESPERSON
5	DEFAULT.DIM_CUSTOM	CUSTOM_NAME
- 5、选取 DEFAULT.FACT\_ORDER 表的哪些列字段作为 measures  
QUANTITY\_ORDERED ORDER\_DOLLARS COST\_DOLLARS
- 6、a.选取 "Partition Date Column"为 DEFAULT.FACT\_ORDER.TIME\_KEY,格式 yyyy-MM-dd  
b.对于"Filter"条件,由于没有要过滤的条件,故不填写
- 7、Save

### 3.4.5、Kylin 的 Cube 创建

- 1、选择"Models" tab 页,单击"New Cube “
- 2、Cube Info:

---

"Model Name"选择,WareHouse\_01\_Model

"Cube Name"输入,cube01

3、Dismensions:

单击"Auto Generator",依据情况选择维度的列,全选

4、Measures:

a.单击"+Measure",添加要聚合计算的度量,比如 sum(QUANTITY\_ORDERED)

b.Expression: SUM/MIN/MAX/COUNT/COUNT\_DISTINCT/TOP\_N/RAW

5、Refresh Setting:

a.Auto Merge Thresholds,自动合并阈值,7~28 days

b.Retention Threshold,保留天数,60

c.Partition Start Date,非常重要,是后面 build cube 的开始日期

6、Advanced Setting:

--Aggregation Groups:

a.Includes: TIME\_KEY ,PRODUCT\_KEY ,SALESPERSON\_KEY , CUSTOM\_KEY

b.Mandatory Dimensions: TIME\_KEY

c.Hierarchy Dimensions: PRODUCT\_KEY ,SALESPERSON\_KEY ,CUSTOM\_KEY

d.Joint Dimensions: 无

--Rowkeys:

TIME\_KEY ,PRODUCT\_KEY ,SALESPERSON\_KEY ,CUSTOM\_KEY 4 个字段为 dict 字典编码

7、Configuration Overwrites: 无

8、Overview:

保存 cube

### 3.4.6、Cube Build

1、选择 cube01,单击" Action" ,选择 Build

2、填写 End Date,Submit

3、单击" Monitor" ,观察 Job

4、等待 Process100% (Any Errors)

5、返回 cube01,查看 cube size 和 Source Records 等字段更新

### 3.4.7、Hive\*KYLIN 查询对比

```
1. 1.2016-05-01到2016-05-15期间的每天的订单数量,订单金额,订单成本
2.
3. Hive: 65.816 s
4. select fact.time_key, sum(fact.quantity_ordered), sum(fact.order_dollars), sum(fact.cost_dollars) from fact_order as fact
5. where fact.time_key >= "2016-05-01" and fact.time_key <= "2016-05-15"
6. group by fact.time_key order by fact.time_key;
7.
8. Kylin: 0.32s-->0.27s
9. select fact.time_key, sum(fact.quantity_ordered), sum(fact.order_dollars), sum(fact.cost_dollars) from fact_order as fact
10. where fact.time_key between '2016-05-01' and '2016-05-15'
11. group by fact.time_key order by fact.time_key
```

```

1. 2. 2016-05-01到2016-05-15期间的每天的产品的订单量
2.
3. Hive: 100.336s
4. select dday.full_day,dsp.product_name, sum(fact.quantity_ordered) from fact_order as fact
5. inner join dim_day as dday on fact.time_key = dday.day_key
6. inner join dim_product as dsp on fact.product_key = dsp.product_key
7. where dday.full_day >= "2016-05-01" and dday.full_day <= "2016-05-15"
8. group by dday.full_day,dsp.product_name
9. order by dday.full_day,dsp.product_name;
10.
11. Kylin:0.93s-->0.39s
12. select dday.full_day,dsp.product_name, sum(fact.quantity_ordered) from fact_order as fact
13. inner join dim_day as dday on fact.time_key = dday.day_key
14. inner join dim_product as dsp on fact.product_key = dsp.product_key
15. where dday.full_day >= '2016-05-01' and dday.full_day <= '2016-05-15'
16. group by dday.full_day,dsp.product_name
17. order by dday.full_day,dsp.product_name

```

The screenshot shows the Kylin web interface with the following details:

- Query String:**

```

1 select dday.full_day,dsp.product_name, sum(fact.quantity_ordered) from fact_order as fact
2 inner join dim_day as dday on fact.time_key = dday.day_key
3 inner join dim_product as dsp on fact.product_key = dsp.product_key
4 where dday.full_day >= '2016-05-01' and dday.full_day <= '2016-05-15'
5 group by dday.full_day,dsp.product_name
6 order by dday.full_day,dsp.product_name;

```
- Project:** Warehouse\_01
- Limit:** 50000
- Status:** Success
- Project:** Warehouse\_01
- Cubes:** cube01
- Results (46):**

FULL_DAY	PRODUCT_NAME	EXPRES2
2016-05-01	Box-Large	24600
2016-05-01	Box-Medium	8200
2016-05-01	Box-small	8200
2016-05-01	Envelope	2050
2016-05-02	Box-Large	10250
2016-05-02	Box-Medium	2050

## 4、KYLIN 优化

### 4.1、Kylin 对传统 MOLAP 的改进

计算 Cube 的存储代价以及计算代价都是比较大的，传统 OLAP 的维度爆炸的问题 Kylin 也一样会遇到。Kylin 提供给用户一些优化措施，在一定程度上能降低维度爆炸的问题：

#### 1. Cube 优化：

- Hierarchy Dimension
- Derived Dimension
- Aggregation Group

Hierarchy Dimension，一系列具有层次关系的 Dimension 组成一个 Hierarchy，比如年、月、日组成了一个 Hierarchy，在 Cube 中，如果不设置 Hierarchy，会有年、月、日、

---

年月、年日、月日 6 个 cuboid，但是设置了 Hierarchy 之后 Cuboid 增加了一个约束，希望低 Level 的 Dimension 一定要伴随高 Level 的 Dimension 一起出现。设置了 Hierachy Dimension 能使得需要计算的维度组合减少一半。

**Derived Dimension**，如果在某张维度表上有多个维度，那么可以将其设置为 **Derived Dimension**，在 Kylin 内部会将其统一用维度表的主键来替换，以此来达到降低维度组合的数目，当然在一定程度上 **Derived Dimension** 会降低查询效率，在查询时，Kylin 使用维度表主键进行聚合后，再通过主键和真正维度列的映射关系做一次转换，在 Kylin 内部再对结果集做一次聚合后返回给用户

**Aggregation Group**，这是一个将维度进行分组，以求达到降低维度组合数目的手段。不同分组的维度之间组成的 Cuboid 数量会大大降低，维度组合从 2 的  $(k+m+n)$  次幂至多能降低到 2 的  $k$  次幂加 2 的  $m$  次幂加 2 的  $n$  次幂。Group 的优化措施与查询 SQL 紧密依赖，可以说是为了查询的定制优化。如果查询的维度是夸 Group 的，那么 Kylin 需要以较大的代价从  $N$ -Cuboid 中聚合得到所需要的查询结果，这需要 Cube 构建人员在建模时仔细地斟酌。

## 2. 数据压缩:

Apache Kylin 针对维度字典以及维度表快照采用了特殊的压缩算法，对于 Hbase 中的聚合计算数据利用了 Hadoop 的 LZ0 或者是 Snappy，从而保证存储在 Hbase 以及内存中的数据尽可能的小。其中维度字典以及维度表快照的压缩考虑到 DataCube 中会出现非常多的重复的维度成员值，最直接的处理方式就是利用数据字典的方式将维度值映射成 ID，Kylin 中采用了 Trie 树的方式对维度值进行编码

## 3. distinct count 聚合查询优化:

Apache Kylin 采用了 HypeLogLog 的方式来计算 DistinctCount。好处是速度快，缺点是结果是一个近似值，会有一定的误差。在非计费等通常的场景下 DistinctCount 的统计误差应用普遍可以接受。

具体的算法可见 Paper，本文不再赘述：

<http://algo.inria.fr/flajolet/Publications/FIFuGaMe07.pdf>

# 4.2、Kylin 的 Hierarchies, Derived 维度方面配置优化详述

## Hierarchies:

理论上对于  $N$  维度，我们可以进行 2 的  $N$  次方的维度组合。然而对于一些维度的组合来说，有时是没有必要的。例如，如果我们有三个维度：continent, country, city，在 hierarchies 中，最大的维度排在最前面。当使用下钻分析时，我们仅仅需要下面的三个维度的组合：

group by continent

group by continent, country

group by continent, country, city

---

在这个例子中，维度的组合从 2 的 3 次方共 8 种减少到了 3 种，这是一个很好的优化，同样适合 YEAR,QUATER,MONTH,DATE 等场景。

如果我们设置 hierarchy 作为 H1,H2,H3，那么典型的场景应该是：

### A. Hierarchies on lookup table

Fact table (joins)Lookup Table

column1,column2,,,,, FK PK,,H1,H2,H3,,,,

### B. Hierarchies on fact table

Fact table

column1,column2,,,H1,H2,H3,,,,,

对于 scenario A,这是一个特殊的案例，PK 在 lookup 的表上，意外的成为了 hierarchies 的一部分。例如我们有一个日历的 lookup 表，cal\_dt 是 PK(primary key)：

#### A\*. Hierarchies on lookup table over its primary key

Lookup Table(Calendar)

cal\_dt(PK), week\_beg\_dt, month\_beg\_dt, quarter\_beg\_dt,,

对于 A\*这种案例，你应该使用“Derived Columns”这种优化方案。

### Derived Columns:

当一个或多个维度(必须是 lookup 表的维度，这些字段被称为“Derived”)能够从另一个中减少(通常是相关的 FK，被称为“host column”)，Derived column 就可以被使用。

例如，假如我们有一个 lookup 的表，我们使用 join 关联 fact 表，并且使用“where DimA=DimX”。在 Kylin 中需要注意，如果你选择 FK 为一个维度，那么相关的 PK 将自动可查询的，没有任何额外的开销。这重要的原因是 FK 和 PK 总是相同的，Kylin 能够首先在 FK 上使用 filters/groupby，并且使用 PK 透明地替换。这个表明如果我们想用 DimA(FK),DimX(PK),DimB,DimC 在我们的 Cube 中，我们能够安全地仅仅选择 DimA,DimB,DimC。

Fact table (joins)Lookup Table

column1,column2,,,,, DimA(FK)DimX(PK),,DimB, DimC

这里的维度 DimA(维度代表 FK/PK)有一个特殊的映射到 DimB。

dimA dimB dimC

1 a ?

2 b ?

3 c ?

4 a ?

在这里案例中，给定一个 DimA 的值，DimB 的值就确定了，因此我们说 DimB 能够从 DimA 获得(Derived)。当我们 build 一个 cube 包含 DimA 和 DimB，我们能够简单的包含 DimA，

---

并且标记 DimB 作为 Derived。Derived column(DimB)不会参与 cuboids 的生成:  
original combinations: --原始维度组合

ABC,AB,AC,BC,A,B,C

combinations when driving B from A: --使用 Derived 优化后的维度组合

AC,A,C

在运行时，例如“select count(\*) from fact\_table inner join loopup1 group by loopup1 .dimB”的案例中，它期待从包含 DimB 的 cuboid 中去获取查询结果。然而，DimB 因为使用了 Derived 优化，在 cuboids 没有结果。在这种情况下，我们修改执行计划，首先按照 DimA(its host column)进行 group by 操作，我们将获取中间的结果，比如：

DimA count(\*)

1 1  
2 1  
3 1  
4 1

然后，Kylin 将使用 DimB 的值替换 DimA 的值(因为他们的值都在 lookup 表中，Kylin 能够加载整个 lookup 表到内存中并且 build 一个他们的映射关系)，因而中间的结果为：

DimB count(\*)

a 1  
b 1  
c 1  
a 1

紧接着，运行 SQL 的引擎(calcite)将进一步的聚合中间结果为最终结果：

DimB count(\*)

a 2  
b 1  
c 1

这个步骤发生在 SQL 查询运行期间，也就是“at the cost of extra runtime aggregation”。

## 5、KYLIN 元数据备份

Kylin 组织它所有的元数据(包括 cube descriptions and instances, projects, inverted index description and instances,jobs, tables and dictionaries)作为一个层次的文件系统。

然而，Kylin 使用 HBase 来进行存储，而不是普通的文件系统。

我们可以从 Kylin 的配置文件 kylin.properties 中查看到：

```
## The metadata store in hbase
```

```
kylin.metadata.url=kylin_metadata@hbase
```

表示 Kylin 的元数据被保存在 HBase 的 kylin\_metadata 表中。



---

## 5.1、备份 Kylin 的元数据

### **./bin/metastore.sh backup**

这将备份元数据到本地目录 `KYLIN_HOME/metadata_backups` 下面，目录的命名格式为：

`KYLIN_HOME/meta_backups/meta_year_month_day_hour_minute_second`

比如我的 Kylin 的家目录为 `/var/lib/kylin/kylin`，那么备份数据的目录为：

`/var/lib/kylin/kylin/meta_backups/meta_2016_05_01_11_50_32`

我们来查看一下目录：

```
[kylin@SZB-L0023777kylin]$ cd /var/lib/kylin/kylin/meta_backups/meta_2016_05_01_11_50_32
```

```
[kylin@SZB-L0023777meta_2016_05_01_11_50_32]$ ll
```

```
total 44
```

```
drwxrwxr-x 2 kylin kylin 4096 May  1 11:50 cube
drwxrwxr-x 2 kylin kylin 4096 May  1 11:50 cube_desc
drwxrwxr-x 4 kylin kylin 4096 May  1 11:50 cube_statistics
drwxrwxr-x 6 kylin kylin 4096 May  1 11:50 dict
drwxrwxr-x 2 kylin kylin 4096 May  1 11:50 execute
drwxrwxr-x 2 kylin kylin 4096 May  1 11:50 execute_output
drwxrwxr-x 2 kylin kylin 4096 May  1 11:50 model_desc
drwxrwxr-x 2 kylin kylin 4096 May  1 11:50 project
drwxrwxr-x 2 kylin kylin 4096 May  1 11:50 table
drwxrwxr-x 2 kylin kylin 4096 May  1 11:50 table_exd
drwxrwxr-x 5 kylin kylin 4096 May  1 11:50 table_snapshot
```

## 5.2、恢复元数据

假如你的 Kylin 元数据挂掉了，那么我们就可以使用之前备份的数据进行恢复：

1. 首先 `reset` 当前 Kylin 的元数据存储，这将清理掉所有存储在 HBase 中的 Kylin 元数据，确保在此之前做过备份

### **./bin/metastore.sh reset**

2. 接着，上传备份的元数据到 Kylin 的元数据中

```
./bin/metastore.sh restore$KYLIN_HOME/meta_backups/meta_xxxx_xx_xx_xx_xx_xx
```

## 5.3、从 Kylin 元数据中清理掉无用的资源

随着时间的推移，有些资源，比如字典、表的快照等变得无用了(cube 的 segment 被删除或合并了)，但是他们仍然占用空间。可以执行如下命令查找和清理无用的元数据：

1. 首先，执行检查，这是安全的操作，不会修改任何内容：

### **./bin/metastore.sh clean**

---

将需要被删除的资源(resources)罗列出来

2. 接着，在上面的命令中，添加“--delete true”参数，这样就会清理掉哪些无用的资源。切记，在这个命令操作之前，一定要备份 Kylin 元数据：

```
./bin/metastore.sh clean --delete true
```

## 6、KYLIN 的中间存储清理(HDFS & HBase Tables)

Kylin 在创建 cube 过程中会在 HDFS 上生成中间数据。另外，当我们对 cube 执行 purge/drop/merge 时，一些 HBase 的表可能会保留在 HBase 中，而这些表不再被查询，尽管 Kylin 会做一些自动的垃圾回收，但是它可能不会覆盖所有方面，所以需要我们能够每隔一段时间做一些离线存储的清理工作。具体步骤如下：

1. 检查哪些资源需要被清理，这个操作不会删除任何内容：

```
export KYLIN_HOME=/path/to/kylin_home
```

```
${KYLIN_HOME}/bin/kylin.sh
```

```
org.apache.kylin.storage.hbase.util.StorageCleanupJob --delete false
```

2. 根据上面的输出结果，挑选一两个资源看看是否是不再需要的。接着，在上面的命令基础上添加“--delete true”选项，开始执行清理操作，命令执行完成后，中间的 HDFS 文和盒 HTables 表就被删除了。

---

## 7、KYLIN 的 RESTFUL API 使用

### 7.1、Build Cube with RESTful API

#### 1. Authentication

- Currently, Kylin uses [basic authentication](#).
- Add `Authorization` header to first request for authentication
- Or you can do a specific request by `POST http://localhost:7070/kylin/api/user/authentication`
- Once authenticated, client can go subsequent requests with cookies.

```
POST http://localhost:7070/kylin/api/user/authentication
```

```
Authorization:Basic xxxxD124xxxGFxxxSDF
Content-Type: application/json;charset=UTF-8
```

#### 2. Get details of cube.

- `GET http://localhost:7070/kylin/api/cubes?cubeName={cube_name}&limit=15&offset=0`
- Client can find cube segment date ranges in returned cube detail.

```
GET http://localhost:7070/kylin/api/cubes?cubeName=test_kylin_cube_with_slr&limit=15&offset=0
```

```
Authorization:Basic xxxxD124xxxGFxxxSDF
Content-Type: application/json;charset=UTF-8
```

#### 3. Then submit a build job of the cube.

- `PUT http://localhost:7070/kylin/api/cubes/{cube_name}/rebuild`
- For put request body detail please refer to [Build Cube API](#).
  - `startTime` and `endTime` should be utc timestamp.
  - `buildType` can be `BUILD`, `MERGE` or `REFRESH`. `BUILD` is for building a new segment, `REFRESH` for refreshing an existing segment. `MERGE` is for merging multiple existing segments into one bigger segment.
- This method will return a new created job instance, whose uuid is the unique id of job to track job status.

```
PUT http://localhost:7070/kylin/api/cubes/test_kylin_cube_with_slr/rebuild
```

```
Authorization:Basic xxxxD124xxxGFxxxSDF
Content-Type: application/json;charset=UTF-8
```

```
{
  "startTime": 0,
  "endTime": 1388563200000,
  "buildType": "BUILD"
}
```

#### 4. Track job status.

- `GET http://localhost:7070/kylin/api/jobs/{job_uuid}`
- Returned `job_status` represents current status of job.

#### 5. If the job got errors, you can resume it.

- `PUT http://localhost:7070/kylin/api/jobs/{job_uuid}/resume`

Eg:

```
log="/home/kylinbuild/cube_cront/logs/"$(getLastDay)"_cube_build.log";

curl -c cookiefile.txt -X POST -H "Authorization: Basic QURNSU46S11MSU4=" -H 'Content-Type: application/json' http://192.168.50.24:7070/kylin/api/user/authentication>>$log
curl -b cookiefile.txt -X PUT -H 'Content-Type: application/json' -d '{"startTime":'$st', "endTime":'$et', "buildType": "BUILD"}' http://192.168.50.24:7070/kylin/api/cubes/Sample_Cube_1/rebuild>>$log
```

log="/home/kylinbuild/cube\_cront/logs/"\$(getLastDay)"\_cube\_build.log";

curl -c cookiefile.txt -X POST -H "Authorization: Basic QURNSU46S11MSU4=" -H 'Content-Type: application/json' http://192.168.50.24:7070/kylin/api/user/authentication>>\$log

curl -b cookiefile.txt -X PUT -H 'Content-Type: application/json' -d '{"startTime":'\$st', "endTime":'\$et', "buildType": "BUILD"}' http://192.168.50.24:7070/kylin/api/cubes/Sample\_Cube\_1/rebuild>>\$log

curl -X PUT --user ADMIN:KYLIN -H "Content-Type: application/json;charset=utf-8" -d '{ "startTime": 1356998400000, "endTime": 1357084800000, "buildType": "BUILD"}' http://localhost:7070/kylin/api/cubes/kylin\_sales/rebuild

中间的时间处理可以用该界面工具去转化

<http://www.epochconverter.com/>

## 7.2、Use RESTful API in Javascript

Kylin security is based on basic access authorization, if you want to use API in your javascript, you need to add authorization info in http headers.

Example on Query API.

```
$.ajaxSetup({
  headers: { 'Authorization': "Basic eWFu*****X***ZA==", 'Content-Type': 'application/json;charset=utf-8' } // use your own authorization code here
});
var request = $.ajax({
  url: "http://hostname/kylin/api/query",
  type: "POST",
  data: '{"sql": "select count(*) from SUMMARY;", "offset": 0, "limit": 50000, "acceptPartial": true, "project": "test"}',
  dataType: "json"
});
request.done(function( msg ) {
  alert(msg);
});
request.fail(function( jqXHR, textStatus ) {
  alert( "Request failed: " + textStatus );
});
```

## Keypoints

1. add basic access authorization info in http headers.
2. use right ajax type and data synax.

## Basic access authorization

For what is basic access authorization, refer to [Wikipedia Page](#).

How to generate your authorization code (download and import “jquery.base64.js” from <https://github.com/yckart/jquery.base64.js>).

```
var authorizationCode = $.base64('encode', 'NT_USERNAME' + ":" + 'NT_PASSWORD');

$.ajaxSetup({
  headers: {
    'Authorization': "Basic " + authorizationCode,
    'Content-Type': 'application/json;charset=utf-8'
  }
});
```

## 7.3、Use RESTful API

This page lists all the RESTful APIs provided by Kylin; The base of the URL is [/kylin/api](#), so don't forget to add it before a certain API's path. For example, to get all cube instances, send HTTP GET request to “/kylin/api/cubes”.

- Query
  - [Authentication](#)
  - [Query](#)
  - [List queryable tables](#)
- CUBE
  - [List cubes](#)
  - [Get cube](#)
  - [Get cube descriptor \(dimension, measure info, etc\)](#)
  - [Get data model \(fact and lookup table info\)](#)
  - [Build cube](#)
  - [Disable cube](#)
  - [Purge cube](#)
  - [Enable cube](#)
- JOB
  - [Resume job](#)
  - [Discard job](#)
  - [Get job status](#)
  - [Get job step output](#)
- Metadata
  - [Get Hive Table](#)
  - [Get Hive Table \(Extend Info\)](#)
  - [Get Hive Tables](#)
  - [Load Hive Tables](#)
- Cache
  - [Wipe cache](#)

Eg:

#### Response Sample

```
{
  "userDetails":{
    "password":null,
    "username":"sample",
    "authorities":[
      {
        "authority":"ROLE_ANALYST"
      },
      {
        "authority":"ROLE_MODELER"
      }
    ],
    "accountNonExpired":true,
    "accountNonLocked":true,
    "credentialsNonExpired":true,
    "enabled":true
  }
}
```

#### Curl Example

```
curl -c /path/to/cookiefile.txt -X POST -H "Authorization: Basic XXXXXXXXX" -H 'Content-Type: application/json' http://<host>:<port>/kylin/api/user/authentication
```

If login successfully, the JSESSIONID will be saved into the cookie file; In the subsequent http requests, attach the cookie, for example:

```
curl -b /path/to/cookiefile.txt -X PUT -H 'Content-Type: application/json' -d '{"startTime":'1423526400000', "endTime":'1423526400', "buildType":"BUILD"}' http://<host>:<port>/kylin/api/cubes/your_cube/rebuild
```

这里只列出单个案例，其余的请到官网自行了解

[http://kylin.apache.org/docs15/howto/howto\\_use\\_restapi.html](http://kylin.apache.org/docs15/howto/howto_use_restapi.html)

## 8、KYLIN 的简单容错处理

Apache Kylin 对于 Cube 的最小存储单位为 data segment，类似于 Hive 的 partition，data segment 采用左闭右开区间表示，如[2015-11-01, 2015-11-02)表示含有 2015-11-01 这一天的数据。对于 Cube 数据的管理主要基于 data segment 粒度，大致分为 3 种操作：计算(build)、更新(refresh)、合并(merge)。对于一个具体产品来说，它的数据是需要每天例行计算到 cube 中，正常例行下，每天会生成 1 个 data segment，但可能会因为数据仓库的任务延迟，2 天或多天生成 1 个 segment。随着时间推移，一方面，大量的 data segment 严重影响了性能，另一方面，这也给管理带来了困难和麻烦。因此，对于 1 个 cube，我们按照 1 个自然月为 1 个 data segment，清晰且易管理。

假设我们有 1 个月 30 天的数据，共 23 个 data segment 数据片段，如：[2015-11-01, 2015-11-02)，[2015-11-02, 2015-11-04)，[2015-11-04, 2015-11-11)，[2015-11-11, 2015-11-12)，[2015-11-12, 2015-11-13)，。。。[2015-11-30, 2015-12-01)

**问题 1:** 假设因为数据有问题，需要回溯 2015-11-01 的数据，因为我们能够在 cube 中找到 [2015-11-01, 2015-11-02)这样一个 data segment，满足这个时间区间，于是，我们可以直接

界面操作或者 Rest API 启动这个 data segment 的 refresh 更新操作。

**问题 2:** 假设我们需要回溯 2015-11-02 到 2015-11-03 的数据，同理，可以找到一个符合条件的 data segment [2015-11-02, 2015-11-04)，然后 refresh 更新这个 data segment。

**问题 3:** 假设我们需要回溯 2015-11-01 到 2015-11-04 的数据，我们找不到直接满足时间区间的 data segment。于是我们有 2 种解决方案，第 1 种方案是分别依次 refresh 更新 [2015-11-01, 2015-11-02)， [2015-11-02, 2015-11-04) 这 2 个 data segment 实现；第 2 种方案是先合并 (merge)[2015-11-01, 2015-11-02)， (2015-11-02, 2015-11-04) 这两个 data segment，合并后得到 [2015-11-01, 2015-11-04) 这样 1 个 data segment，然后我们再拉取新数据后执行更新操作，即可满足需求。

**问题 4:** 假设我们需要刷新 2015-11-01~2015-11-30 这 1 个月的数据，需要将 23 个 data segment 合并成 [2015-11-01, 2015-12-01) 这 1 个 data segment，计 1 次操作。然后再执行 1 次更新操作，共 2 次操作即可完成需求

## 9、KYLIN 的 JDBC Driver

### Authentication

Build on Apache Kylin authentication restful service. Supported parameters:

- user : username
- password : password
- ssl: true/false. Default be false; If true, all the services call will use https.

### Connection URL format:

```
jdbc:kylin://<hostname>:<port>/<kylin_project_name>
```

- If “ssl” = true, the “port” should be Kylin server’s HTTPS port;
- If “port” is not specified, the driver will use default port: HTTP 80, HTTPS 443;
- The “kylin\_project\_name” must be specified and user need ensure it exists in Kylin server;

### 1. Query with Statement

```
Driver driver = (Driver) Class.forName("org.apache.kylin.jdbc.Driver").newInstance();

Properties info = new Properties();
info.put("user", "ADMIN");
info.put("password", "KYLIN");
Connection conn = driver.connect("jdbc:kylin://localhost:7070/kylin_project_name", info);
Statement state = conn.createStatement();
ResultSet resultSet = state.executeQuery("select * from test_table");

while (resultSet.next()) {
    assertEquals("foo", resultSet.getString(1));
    assertEquals("bar", resultSet.getString(2));
    assertEquals("tool", resultSet.getString(3));
}
```

此处未进行全部截图..



Eg:

```
1 package com.zp;
2
3 import java.sql.Connection;
4
5
6
7
8
9 public class KylinODBCTest {
10     public static void main(String[] args) {
11         try {
12             Driver driver = (Driver) Class.forName(
13                 "org.apache.kylin.jdbc.Driver").newInstance();
14             Properties info = new Properties();
15
16             info.put("user", "ADMIN");
17             info.put("password", "KYLIN");
18
19             Connection conn = driver.connect(
20                 "jdbc:kylin://192.168.170.70/KAP_Sample_1", info);
21             Statement state = conn.createStatement();
22             ResultSet res = state.executeQuery("select * from kylin_sales");
23
24             while (res.next()) {
25                 System.out.println(res.getDate("PART_DT"));
26             }
27         } catch (Exception e) {
28             e.printStackTrace();
29         }
30     }
31 }
32
```

此处改成自己的kylin机器IP即可

此处仅限测试，可以自行测试其余类型的字段输出

<terminated> KylinODBCTest [Java Application] E:\Program Files\Java\jre7\bin\javaw.exe (2016年9月6日 下午12:14:13)

2012-12-31  
2012-12-31  
2012-12-31  
2012-12-31  
2012-12-31

基本操作同一般的 JDBC 操作，请自行到官网查看了解

[http://kylin.apache.org/docs15/howto/howto\\_jdbc.html](http://kylin.apache.org/docs15/howto/howto_jdbc.html)

## 10、KYLIN 的版本升级

### Upgrade From Old Versions

#### Upgrade from 1.5.2 to v1.5.3

Kylin v1.5.3 metadata is compitible with v1.5.2, your cubes don' t need rebuilt, as usual, some actions need to be performed:

##### 1. Update HBase coprocessor

The HBase tables for existing cubes need be updated to the latest coprocessor; Follow [this guide](#) to update;

##### 2. Update conf/kylin\_hive\_conf.xml

From 1.5.3, Kylin doesn' t need Hive to merge small files anymore; For users who copy the conf/ from previous version, please remove the "merge" related properties in kylin\_hive\_conf.xml, including "hive.merge.mapfiles", "hive.merge.mapredfiles", and "hive.merge.size.per.task"; this will save the time on extracting data from Hive.

---

此处未进行全部截图..

官网有详细说明，在此不做阐述

[http://kylin.apache.org/docs15/howto/howto\\_upgrade.html](http://kylin.apache.org/docs15/howto/howto_upgrade.html)

## 11、KYLIN 的安全策略

### Enable Security with LDAP and SSO

#### Enable LDAP authentication

Kylin supports LDAP authentication for enterprise or production deployment; This is implemented with Spring Security framework; Before enable LDAP, please contact your LDAP administrator to get necessary information, like LDAP server URL, username/password, search patterns;

#### Configure LDAP server info

Firstly, provide LDAP URL, and username/password if the LDAP server is secured; The password in kylin.properties need be salted; You can run

“org.apache.kylin.rest.security.PasswordPlaceholderConfigurer AES your\_password” to get a hash.

```
ldap.server=ldap://<your_ldap_host>:<port>
ldap.username=<your_user_name>
ldap.password=<your_password_hash>
```

Secondly, provide the user search patterns, this is by LDAP design, here is just a sample:

```
ldap.user.searchBase=OU=UserAccounts,DC=mycompany,DC=com
ldap.user.searchPattern=(&{AccountName={0}})(memberOf=CN=MYCOMPANY-USERS,DC=mycompany,DC=com))
ldap.user.groupSearchBase=OU=Group,DC=mycompany,DC=com
```

If you have service accounts (e.g. for system integration) which also need be authenticated, configure them in ldap.service.\*; Otherwise, leave them be empty;

此处未进行全部截图..

官网有详细说明，在此不做阐述

[http://kylin.apache.org/docs15/howto/howto\\_ldap\\_and\\_sso.html](http://kylin.apache.org/docs15/howto/howto_ldap_and_sso.html)

---

## 12、KYLIN 的 ODBC 操作及和可视化的集成

官网有详细说明，在此不做详细阐述，只进行简单截图表示

### 12.1、ODBC 安装

<http://kylin.apache.org/docs15/tutorial/odbc.html>

#### Kylin ODBC Driver

We provide Kylin ODBC driver to enable data access from ODBC-compatible client applications.

Both 32-bit version or 64-bit version driver are available.

Tested Operation System: Windows 7, Windows Server 2008 R2

Tested Application: Tableau 8.0.4, Tableau 8.1.3 and Tableau 9.1

#### Prerequisites

1. Microsoft Visual C++ 2012 Redistributable
  - For 32 bit Windows or 32 bit Tableau Desktop: Download: [32bit version](#)
  - For 64 bit Windows or 64 bit Tableau Desktop: Download: [64bit version](#)
2. ODBC driver internally gets results from a REST server, make sure you have access to one

#### Installation

1. Uninstall existing Kylin ODBC first, if you already installed it before
2. Download ODBC Driver from [download](#).
  - For 32 bit Tableau Desktop: Please install KylinODBCDriver (x86).exe
  - For 64 bit Tableau Desktop: Please install KylinODBCDriver (x64).exe
3. Both drivers already be installed on Tableau Server, you properly should be able to publish to there without issues

### 12.2、Tableau 9 集成

[http://kylin.apache.org/docs15/tutorial/tableau\\_91.html](http://kylin.apache.org/docs15/tutorial/tableau_91.html)

#### Tableau 9

Tableau 9.x has been released a while, there are many users are asking about support this version with Apache Kylin. With updated Kylin ODBC Driver, now user could interactive with Kylin service through Tableau 9.x.

#### For Tableau 8.x User

Please refer to [Kylin and Tableau Tutorial](#) for detail guide.

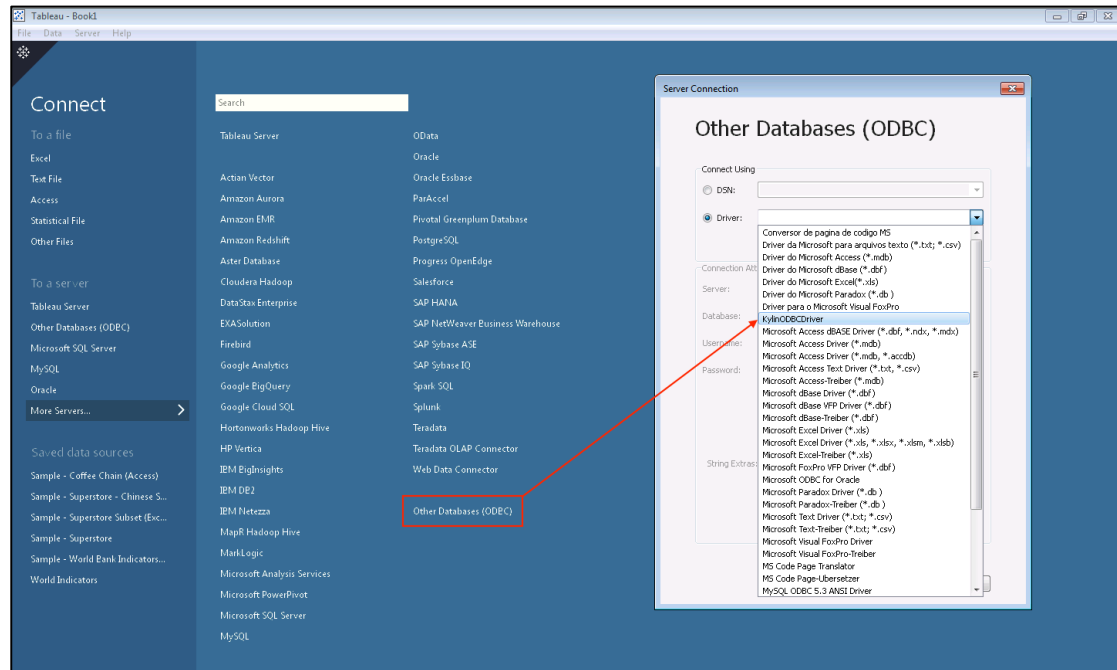
#### Install Kylin ODBC Driver

Refer to this guide: [Kylin ODBC Driver Tutorial](#).

Please make sure to download and install Kylin ODBC Driver **v1.5**. If you already installed ODBC Driver in your system, please uninstall it first.

#### Connect to Kylin Server

Connect Using Driver: Start Tableau 9.1 desktop, click **Other Database(ODBC)** in the left panel and choose KylinODBCDriver in the pop-up window.



## 12.3、MS Excel and Power BI 集成

<http://kylin.apache.org/docs15/tutorial/powerbi.html>

### MS Excel and Power BI

Microsoft Excel is one of the most famous data tool on Windows platform, and has plenty of data analyzing functions. With Power Query installed as plug-in, excel can easily read data from ODBC data source and fill spreadsheets.

Microsoft Power BI is a business intelligence tool providing rich functionality and experience for data visualization and processing to user.

Apache Kylin currently doesn't support query on raw data yet, some queries might fail and cause some exceptions in application. Patch KYLIN-1075 is recommended to get better look of query result.

Power BI and Excel do not support "connect live" model for other ODBC driver yet, please pay attention when you query on huge dataset, it may pull too many data into your client which will take a while even fail at the end.

#### Install ODBC Driver

Refer to this guide: [Kylin ODBC Driver Tutorial](#).

Please make sure to download and install Kylin ODBC Driver **v1.2**. If you already installed ODBC Driver in your system, please uninstall it first.

**注:**

以上内容摘自于官网相关文档、朋友的博客以及百度相关文档，只供个人学习，禁止非法用途，有问题可以联系本人：**58123441**（丶扎啤）