大数据技术之Kylin

(作者：杨晓凯)

版本：V1.0

# 第1章 概述

## 1.1 Kylin定义

Apache Kylin是一个开源的分布式分析引擎，提供Hadoop/Spark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，最初由eBay Inc开发并贡献至开源社区。它能在亚秒内查询巨大的Hive表。

## 1.2 Kylin特点

Kylin的主要特点包括支持SQL接口、支持超大规模数据集、亚秒级响应、可伸缩性、高吞吐率、BI工具集成等。

1）标准SQL接口：Kylin是以标准的SQL作为对外服务的接口。

2）支持超大数据集：Kylin对于大数据的支撑能力可能是目前所有技术中最为领先的。早在2015年eBay的生产环境中就能支持百亿记录的秒级查询，之后在移动的应用场景中又有了千亿记录秒级查询的案例。

3）亚秒级响应：Kylin拥有优异的查询相应速度，这点得益于预计算，很多复杂的计算，比如连接、聚合，在离线的预计算过程中就已经完成，这大大降低了查询时刻所需的计算量，提高了响应速度。

4）可伸缩性和高吞吐率：单节点Kylin可实现每秒70个查询，还可以搭建Kylin的集群。

5）BI工具集成

Kylin可以与现有的BI工具集成，具体包括如下内容。

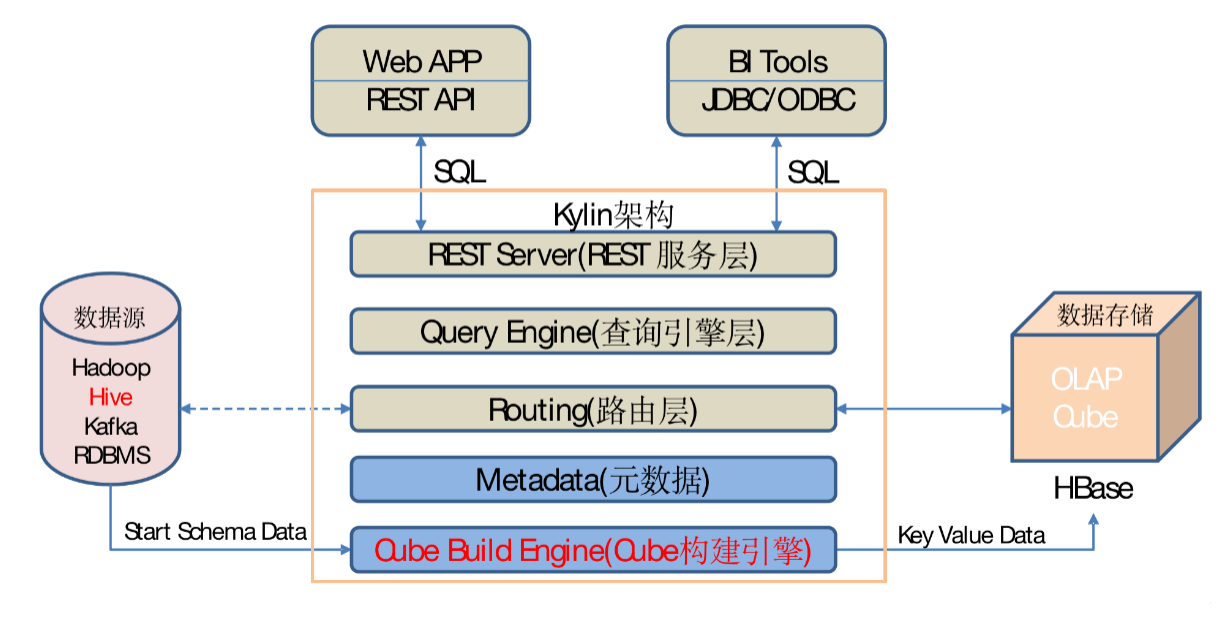
ODBC：与Tableau、Excel、PowerBI等工具集成

JDBC：与Saiku、BIRT等Java工具集成

RestAPI：与JavaScript、Web网页集成

Kylin开发团队还贡献了**Zepplin**的插件，也可以使用Zepplin来访问Kylin服务。

## 1.3 Kylin架构



1）REST Server

REST Server是一套面向应用程序开发的入口点，旨在实现针对Kylin平台的应用开发工作。 此类应用程序可以提供查询、获取结果、触发Cube构建任务、获取元数据以及获取用户权限等等。另外可以通过Restful接口实现SQL查询。

2）查询引擎（Query Engine）

当Cube准备就绪后，查询引擎就能够获取并解析用户查询。它随后会与系统中的其它组件进行交互，从而向用户返回对应的结果。

3）Routing

负责将解析的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，这部分查询可以在秒级设置毫秒级完成，而且还有一些操作使用过的查询原始数据（存储在Hadoop的HDFS中通过Hive查询）。这部分查询延迟较高。

4）元数据管理工具（Metadata）

Kylin是一款元数据驱动型应用程序。元数据管理工具是一大关键性组件，用于对保存在Kylin当中的所有元数据进行管理，其中包括最为重要的Cube元数据。其它全部组件的正常运作都需以元数据管理工具为基础。Kylin的元数据存储在hbase中。

5）任务引擎（Cube Build Engine）

这套引擎的设计目的在于处理所有离线任务，其中包括Shell脚本、Java API以及Map Reduce任务等等。任务引擎对Kylin当中的全部任务加以管理与协调，从而确保每一项任务都能得到切实执行并解决其间出现的故障。

## 1.4 Kylin工作原理

Apache Kylin的工作原理本质上是MOLAP（Multidimension On-Line Analysis Processing）Cube，也就是多维立方体分析。是数据分析中非常经典的理论，下面对其做简要介绍。

### 1.4.1 维度和度量

维度：即观察数据的角度。比如员工数据，可以从性别角度来分析，也可以更加细化，从入职时间或者地区的维度来观察。维度是一组离散的值，比如说性别中的男和女，或者时间维度上的每一个独立的日期。因此在统计时可以将维度值相同的记录聚合在一起，然后应用聚合函数做累加、平均、最大和最小值等聚合计算。

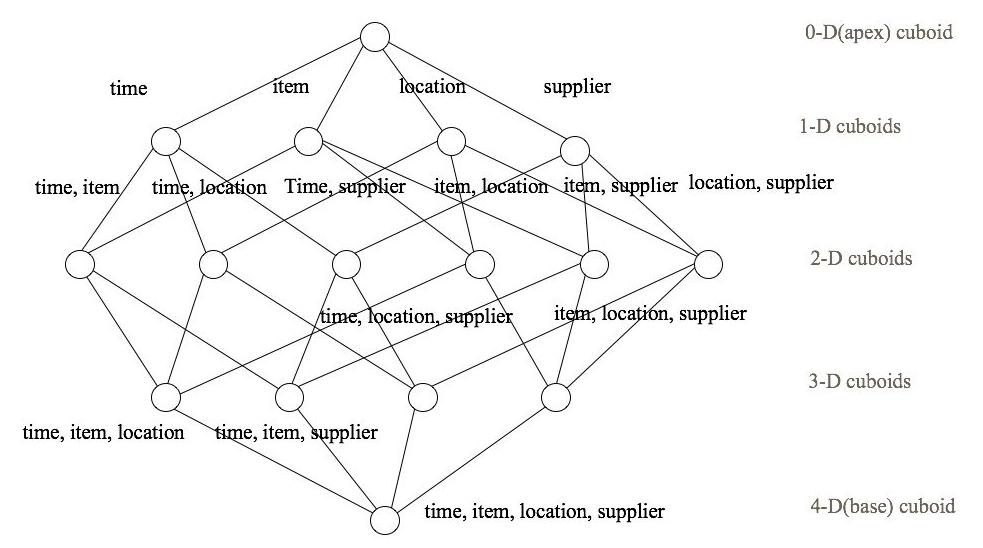
度量：即被聚合（观察）的统计值，也就是聚合运算的结果。比如说员工数据中不同性别员工的人数，又或者说在同一年入职的员工有多少。

### 1.4.2 Cube和Cuboid

有了维度跟度量，一个数据表或者数据模型上的所有字段就可以分类了，它们要么是维度，要么是度量（可以被聚合）。于是就有了根据维度和度量做预计算的Cube理论。

给定一个数据模型，我们可以对其上的所有维度进行聚合，对于N个维度来说，组合的所有可能性共有2n种。对于每一种维度的组合，将度量值做聚合计算，然后将结果保存为一个物化视图，称为Cuboid。所有维度组合的Cuboid作为一个整体，称为Cube。

下面举一个简单的例子说明，假设有一个电商的销售数据集，其中维度包括时间[time]、商品[item]、地区[location]和供应商[supplier]，度量为销售额。那么所有维度的组合就有24 = 16种，如下图所示：



一维度（1D）的组合有：[time]、[item]、[location]和[supplier]4种；

二维度（2D）的组合有：[time, item]、[time, location]、[time, supplier]、[item, location]、[item, supplier]、[location, supplier]3种；

三维度（3D）的组合也有4种；

最后还有零维度（0D）和四维度（4D）各有一种，总共16种。

注意：每一种维度组合就是一个Cuboid，16个Cuboid整体就是一个Cube。

### 1.4.3 核心算法

Kylin的工作原理就是对数据模型做Cube预计算，并利用计算的结果加速查询：

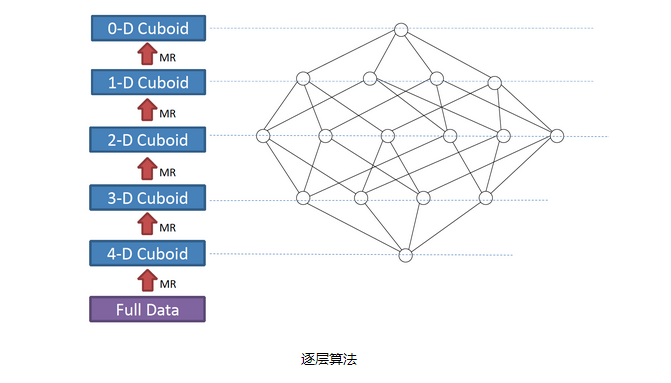
1）指定数据模型，定义维度和度量；

2）预计算Cube，计算所有Cuboid并保存为物化视图；

预计算过程是Kylin从Hive中读取原始数据，按照我们选定的维度进行计算，并将结果集保存到Hbase中，默认的计算引擎为MapReduce，可以选择Spark作为计算引擎。一次build的结果，我们称为一个Segment。构建过程中会涉及多个Cuboid的创建，具体创建过程由kylin.Cube.algorithm参数决定，参数值可选 auto，layer 和 inmem， 默认值为 auto，即 Kylin 会通过采集数据动态地选择一个算法 (layer or inmem)，如果用户很了解 Kylin 和自身的数据、集群，可以直接设置喜欢的算法。

3）执行查询，读取Cuboid，运行，产生查询结果。

#### 1.4.3.1 逐层构建算法（layer）



我们知道，一个N维的Cube，是由1个N维子立方体、N个(N-1)维子立方体、N\*(N-1)/2个(N-2)维子立方体、......、N个1维子立方体和1个0维子立方体构成，总共有2^N个子立方体组成，在逐层算法中，按维度数逐层减少来计算，每个层级的计算（除了第一层，它是从原始数据聚合而来），是基于它上一层级的结果来计算的。比如，[Group by A, B]的结果，可以基于[Group by A, B, C]的结果，通过去掉C后聚合得来的；这样可以减少重复计算；当 0维度Cuboid计算出来的时候，整个Cube的计算也就完成了。

每一轮的计算都是一个MapReduce任务，且串行执行；一个N维的Cube，至少需要N+1次MapReduce Job。

**算法优点：**

1）此算法充分利用了MapReduce的能力，处理了中间复杂的排序和洗牌工作，故而算法代码清晰简单，易于维护；

2）受益于Hadoop的日趋成熟，此算法对集群要求低，运行稳定；在内部维护Kylin的过程中，很少遇到在这几步出错的情况；即便是在Hadoop集群比较繁忙的时候，任务也能完成。

**算法缺点：**

1）当Cube有比较多维度的时候，所需要的MapReduce任务也相应增加；由于Hadoop的任务调度需要耗费额外资源，特别是集群较庞大的时候，反复递交任务造成的额外开销会相当可观；

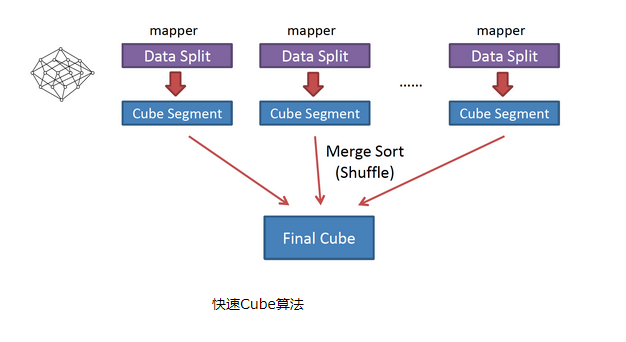
2）此算法会对Hadoop MapReduce输出较多数据; 虽然已经使用了Combiner来减少从Mapper端到Reducer端的数据传输，所有数据依然需要通过Hadoop MapReduce来排序和组合才能被聚合，无形之中增加了集群的压力;

3）对HDFS的读写操作较多：由于每一层计算的输出会用做下一层计算的输入，这些Key-Value需要写到HDFS上；当所有计算都完成后，Kylin还需要额外的一轮任务将这些文件转成HBase的HFile格式，以导入到HBase中去；

总体而言，该算法的效率较低，尤其是当Cube维度数较大的时候。

#### 1.4.3.2 快速构建算法（inmem）

也被称作“逐段”(By Segment) 或“逐块”(By Split) 算法，从1.5.x开始引入该算法，利用Mapper端计算先完成大部分聚合，再将聚合后的结果交给Reducer，从而降低对网络瓶颈的压力。该算法的主要思想是，对Mapper所分配的数据块，将它计算成一个完整的小Cube 段（包含所有Cuboid）；每个Mapper将计算完的Cube段输出给Reducer做合并，生成大Cube，也就是最终结果；如图所示解释了此流程。



与旧算法相比，快速算法主要有两点不同：

1） Mapper会利用内存做预聚合，算出所有组合；Mapper输出的每个Key都是不同的，这样会减少输出到Hadoop MapReduce的数据量；

2）一轮MapReduce便会完成所有层次的计算，减少Hadoop任务的调配。

# 第2章 Kylin环境搭建

## 2.1 安装地址

1）官网地址

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

2）官方文档

<http://kylin.apache.org/cn/docs/>

3）下载地址

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

## 2.2 安装部署

1）将apache-kylin-2.5.1-bin-hbase1x.tar.gz上传到Linux

2）解压apache-kylin-2.5.1-bin-hbase1x.tar.gz到/opt/module

[bawei@hadoop102 sorfware]$ tar -zxvf apache-kylin-2.5.1-bin-hbase1x.tar.gz -C /opt/module/

注意：需要在/etc/profile文件中配置HADOOP\_HOME，HIVE\_HOME，HBASE\_HOME并将其对应的sbin（如果有这个目录的话）和bin目录配置到Path，最后需要source使其生效。

3）启动

[bawei@hadoop102 kylin]$ bin/kylin.sh start

启动之后查看各个节点进程：

--------------------- hadoop102 ----------------

3360 JobHistoryServer(MR的历史服务，必须启动)

31425 HMaster

3282 NodeManager

3026 DataNode

53283 Jps

2886 NameNode

44007 RunJar

2728 QuorumPeerMain

31566 HRegionServer

--------------------- hadoop103 ----------------

5040 HMaster

2864 ResourceManager

9729 Jps

2657 QuorumPeerMain

4946 HRegionServer

2979 NodeManager

2727 DataNode

--------------------- hadoop104 ----------------

4688 HRegionServer

2900 NodeManager

9848 Jps

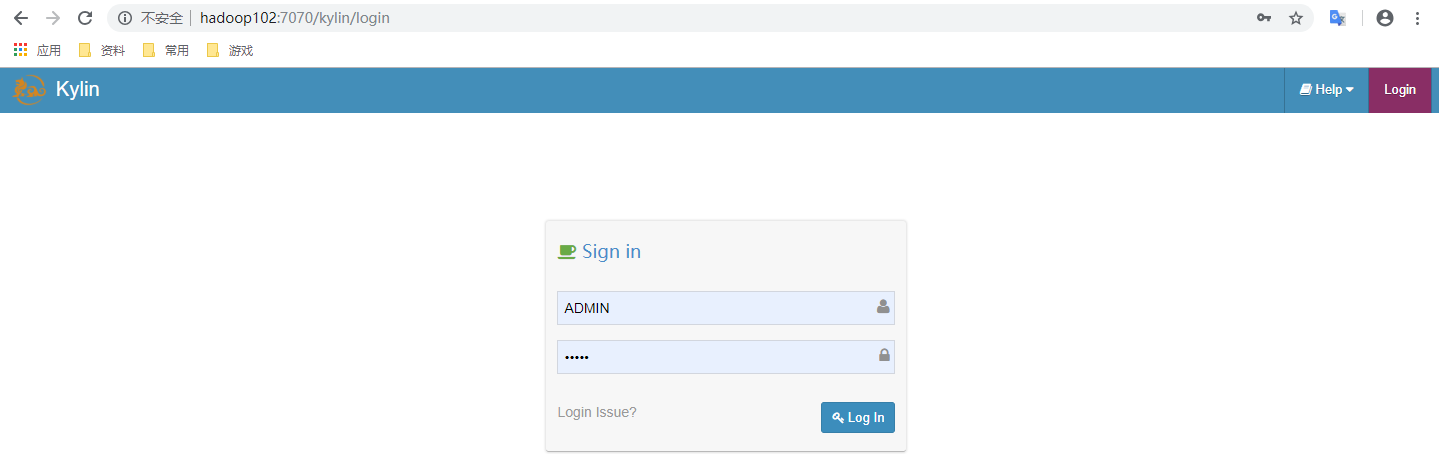
2636 QuorumPeerMain

2700 DataNode

2815 SecondaryNameNode

注意：启动Kylin之前要保证HDFS，YARN，ZK，HBASE相关进程是正常运行的。

在<http://hadoop102:7070/kylin>查看Web页面



用户名为：ADMIN，密码为：KYLIN（系统已填）

4）关闭

[bawei@hadoop102 kylin]$ bin/kylin.sh stop

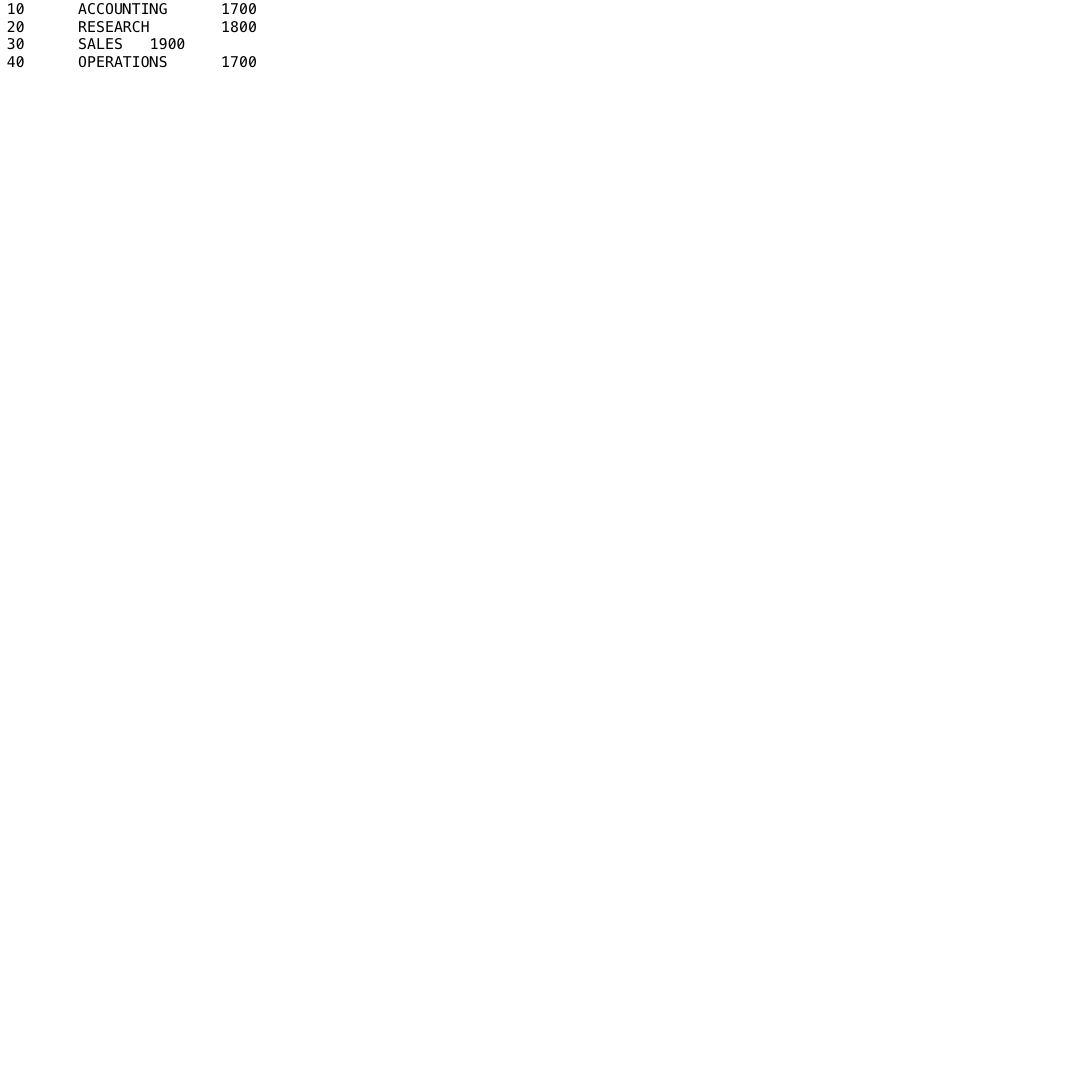
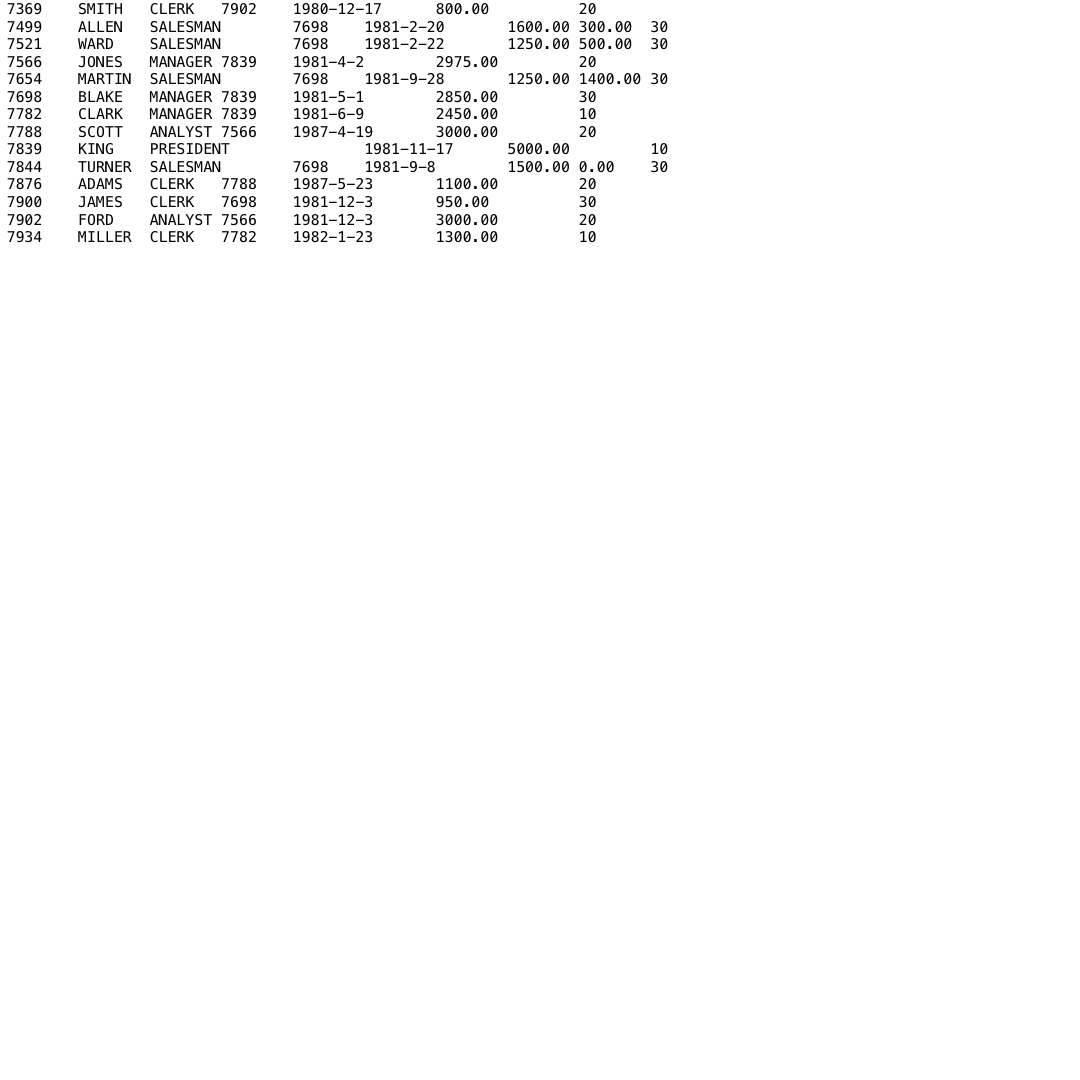
# 第3章 快速入门

需求：实现按照维度（工作地点）统计员工信息

## 3.1 数据准备

在Hive中创建数据，分别创建部门和员工外部表，并向表中导入数据。

（1）原始数据

（2）建表语句

创建部门表

create external table if not exists default.dept(

deptno int,

dname string,

loc int

)

row format delimited fields terminated by '\t';

创建员工表

create external table if not exists default.emp(

empno int,

ename string,

job string,

mgr int,

hiredate string,

sal double,

comm double,

deptno int)

row format delimited fields terminated by '\t';

（3）查看创建的表

hive (default)> show tables;

OK

tab\_name

dept

emp

（4）向外部表中导入数据

导入数据

hive (default)> load data local inpath '/opt/module/datas/dept.txt' into table default.dept;

hive (default)> load data local inpath '/opt/module/datas/emp.txt' into table default.emp;

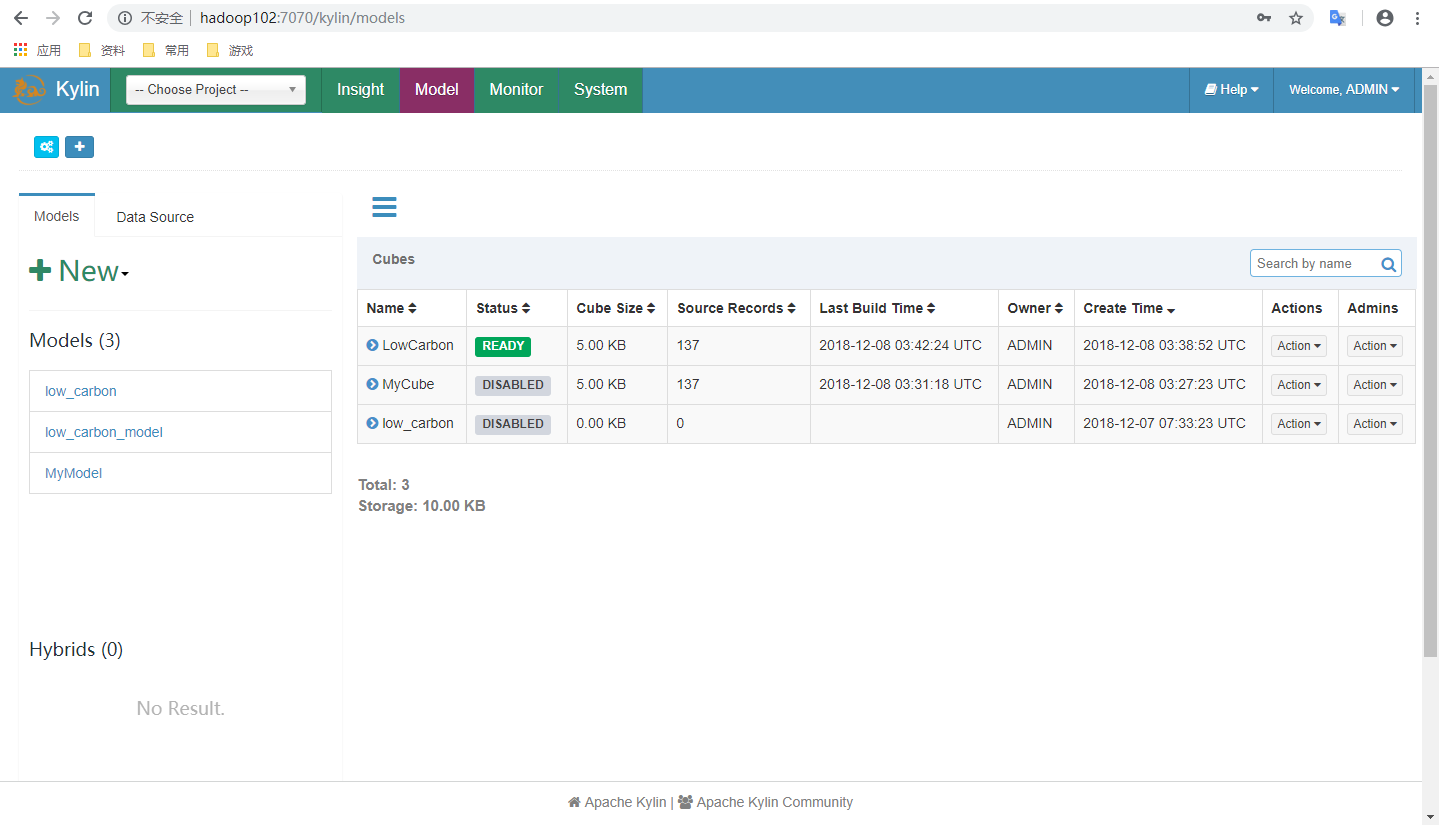
查询结果

hive (default)> select \* from emp;

hive (default)> select \* from dept;

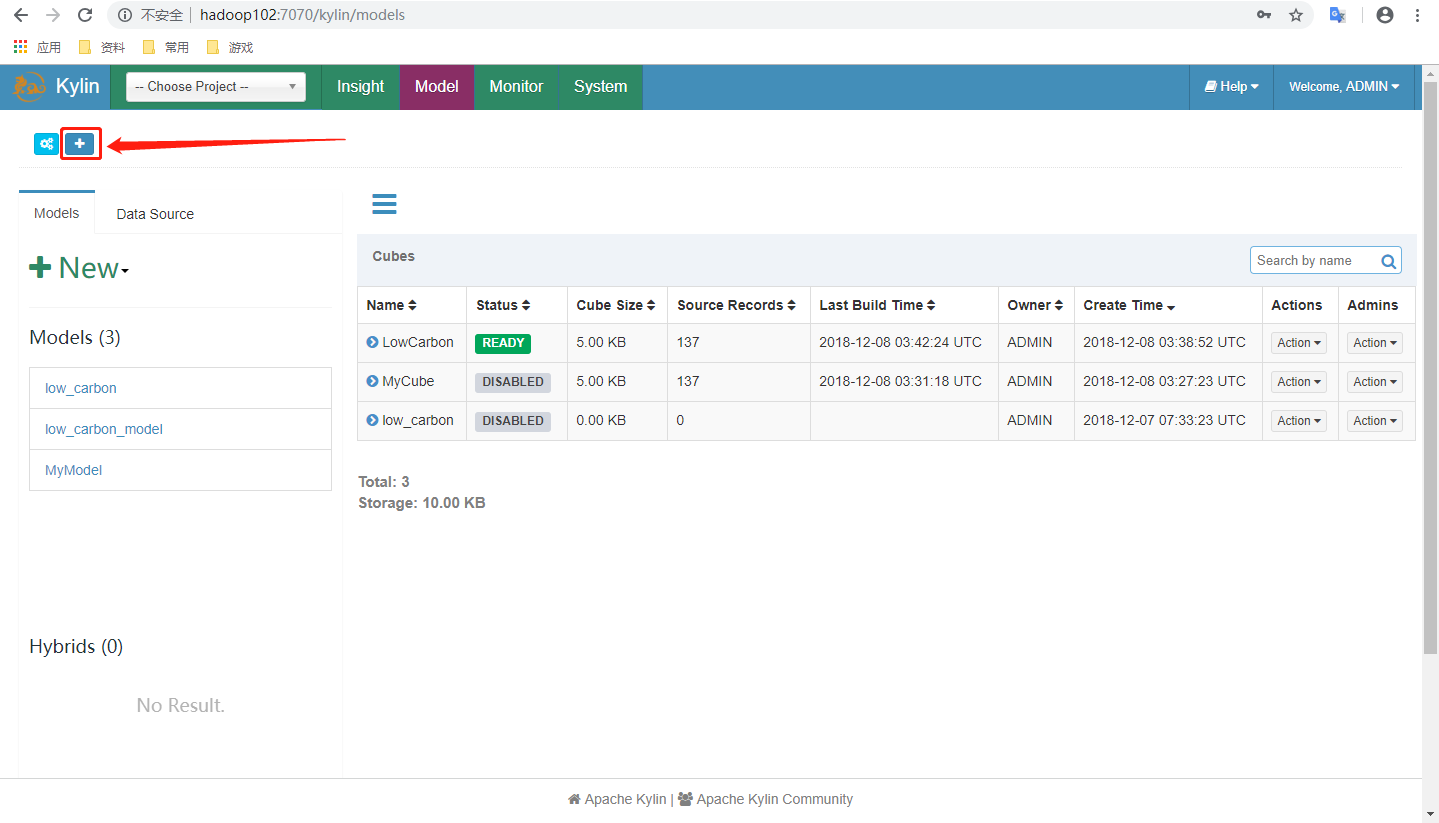
## 3.2 创建项目

### 3.2.1 登录系统

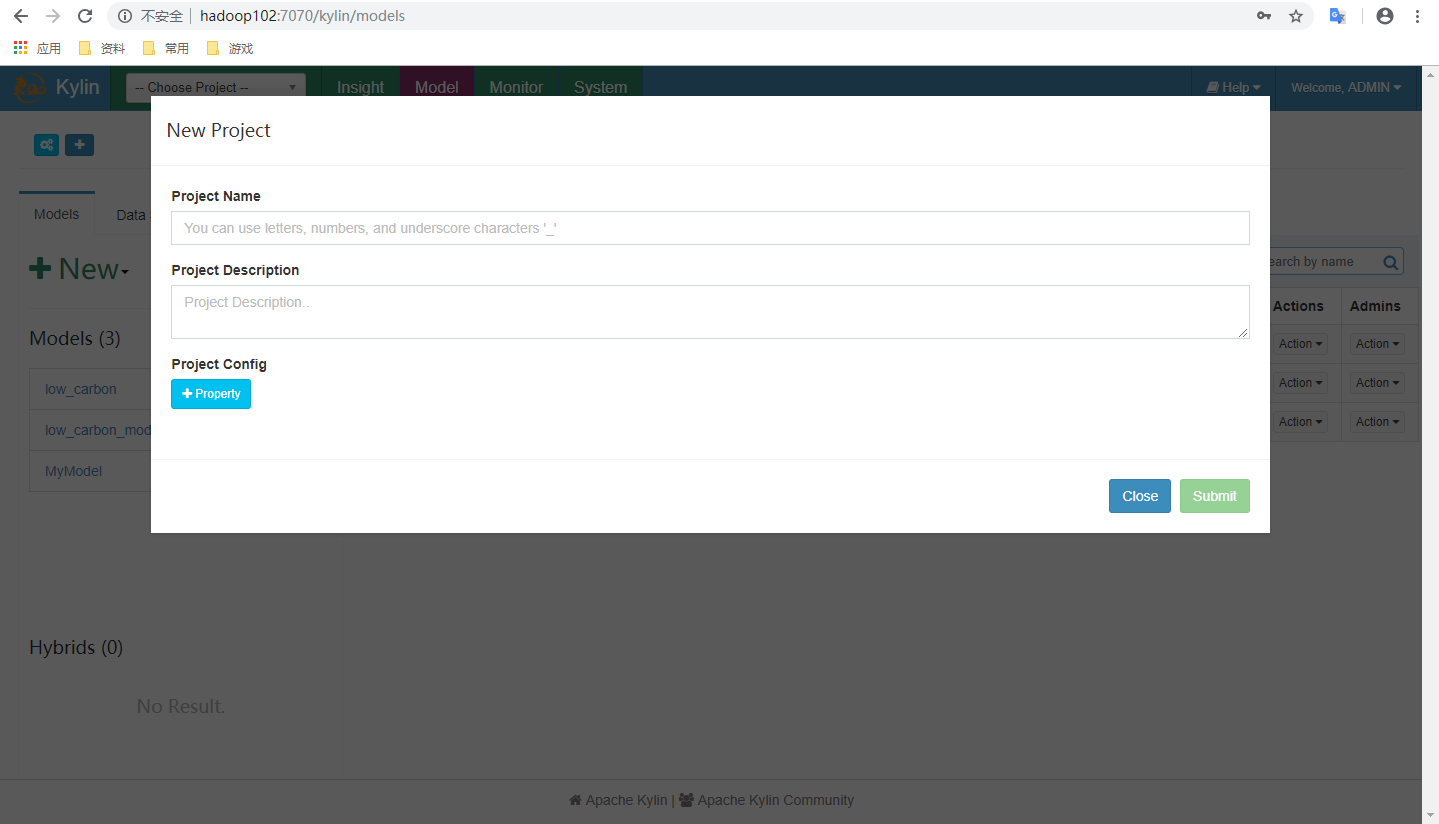


### 3.2.2 创建工程

1）点击图上所示“+”号

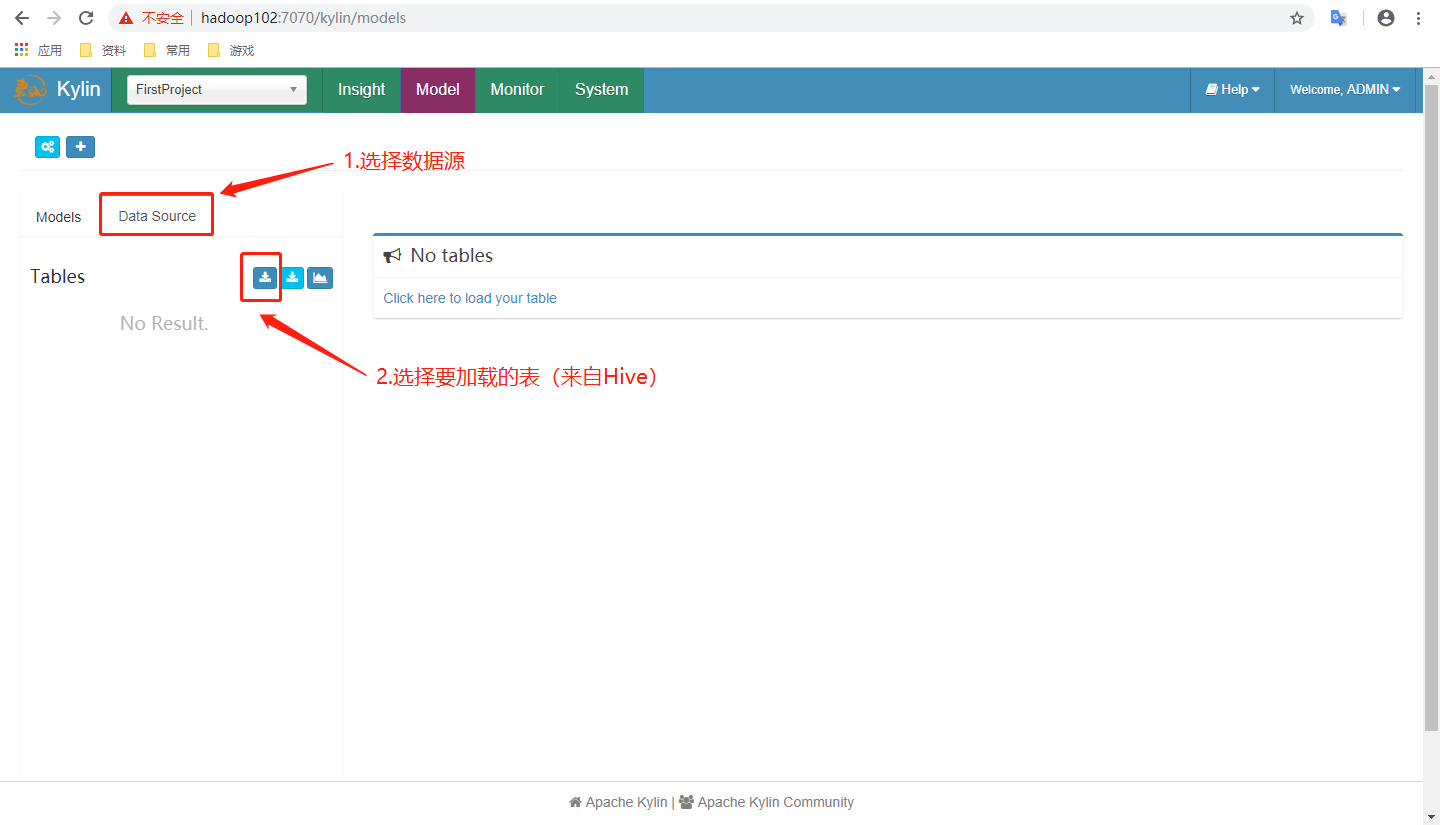


2）填入项目名及描述点击Submit

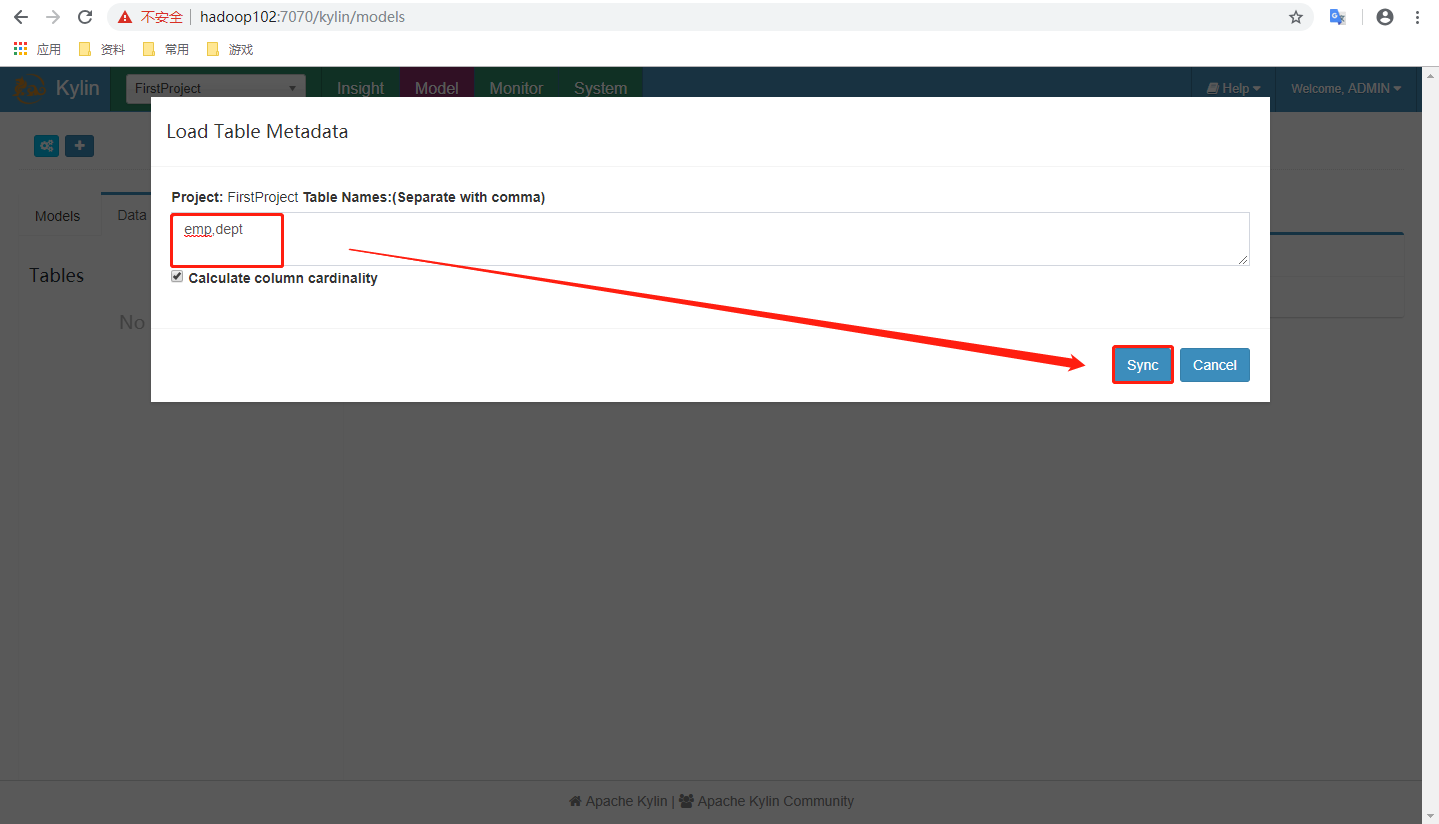


### 3.2.3 选择数据源

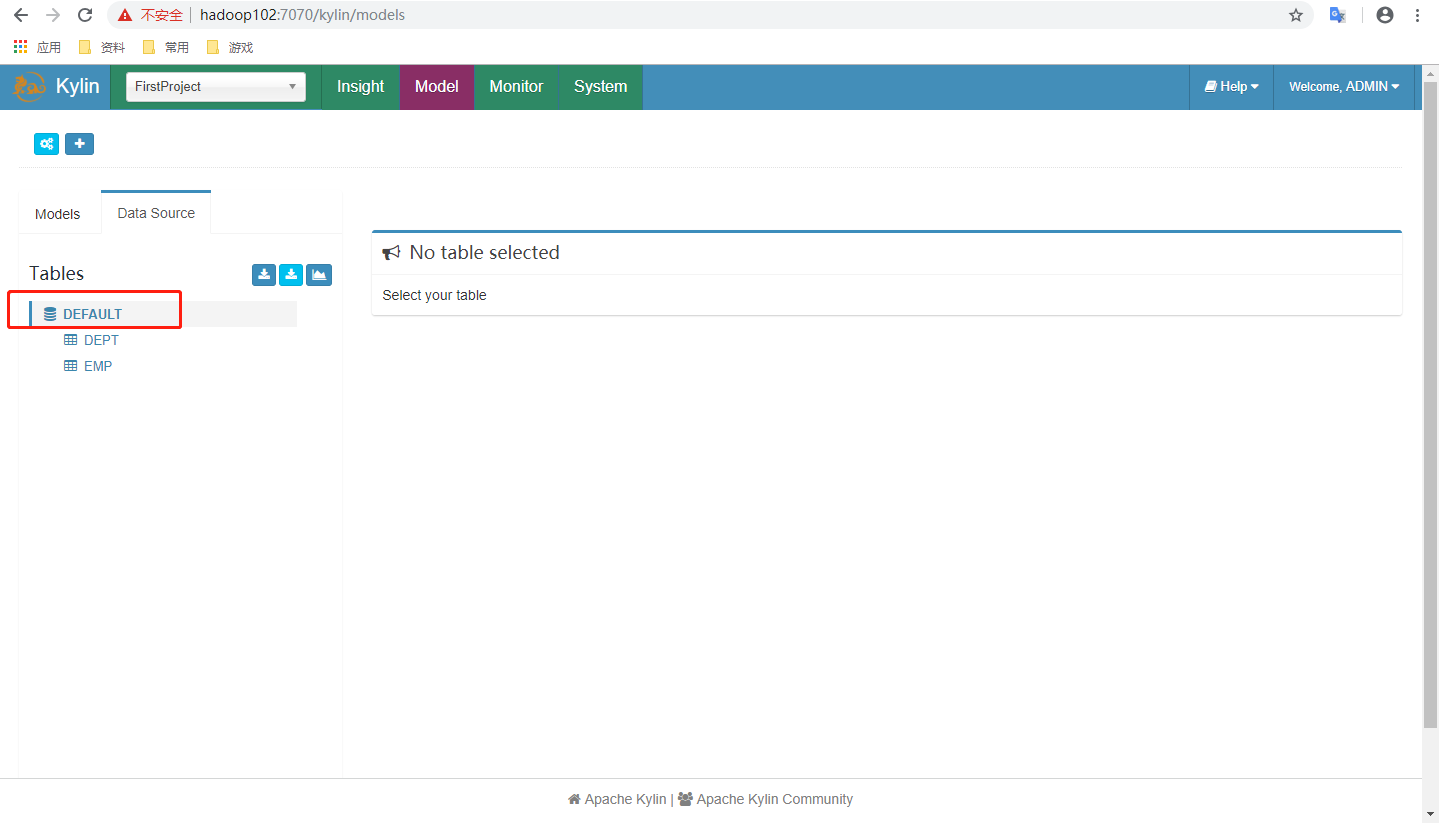
1）选择加载数据源方式



2）输入要作为数据源的表

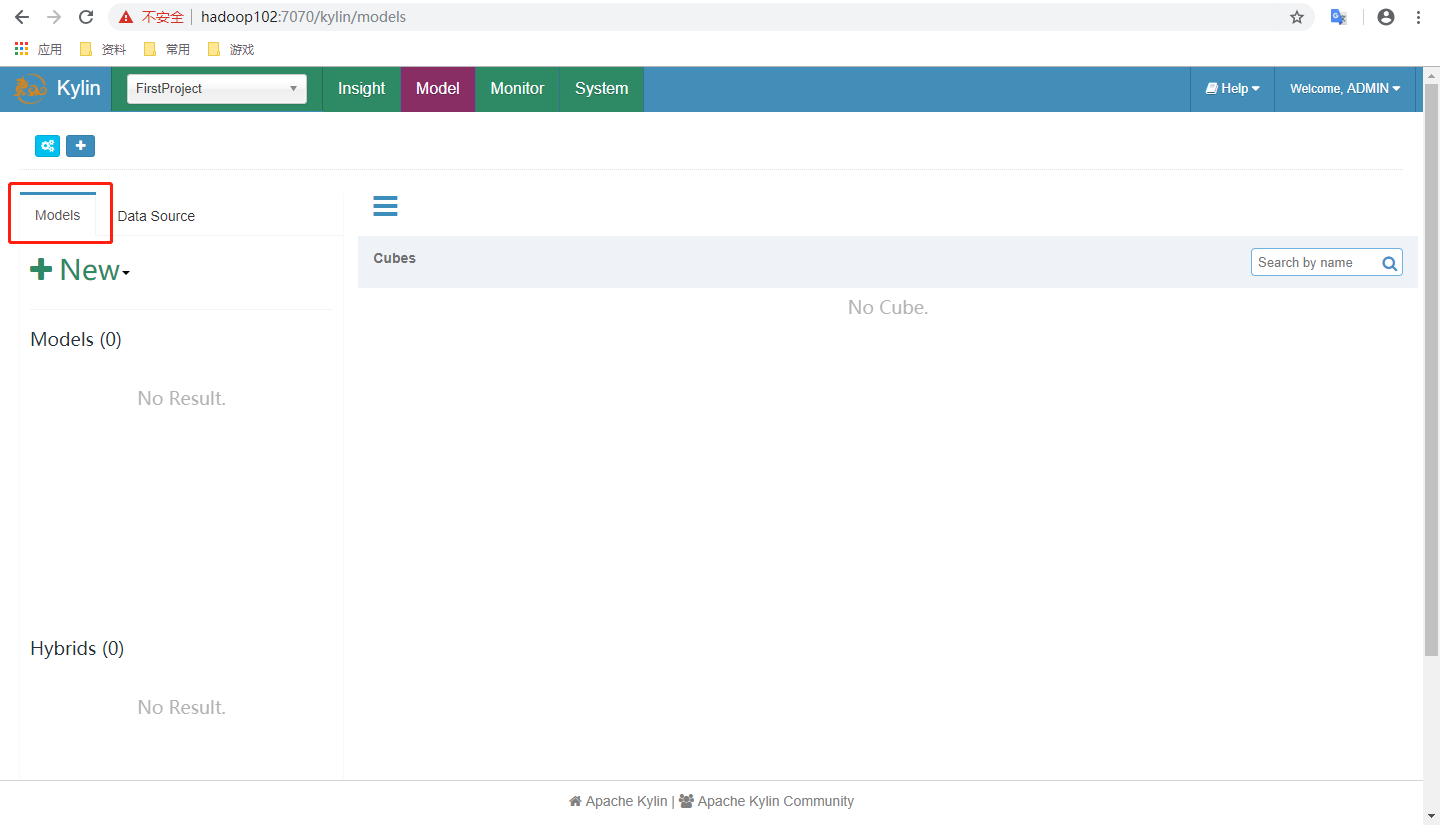


3）查看数据源

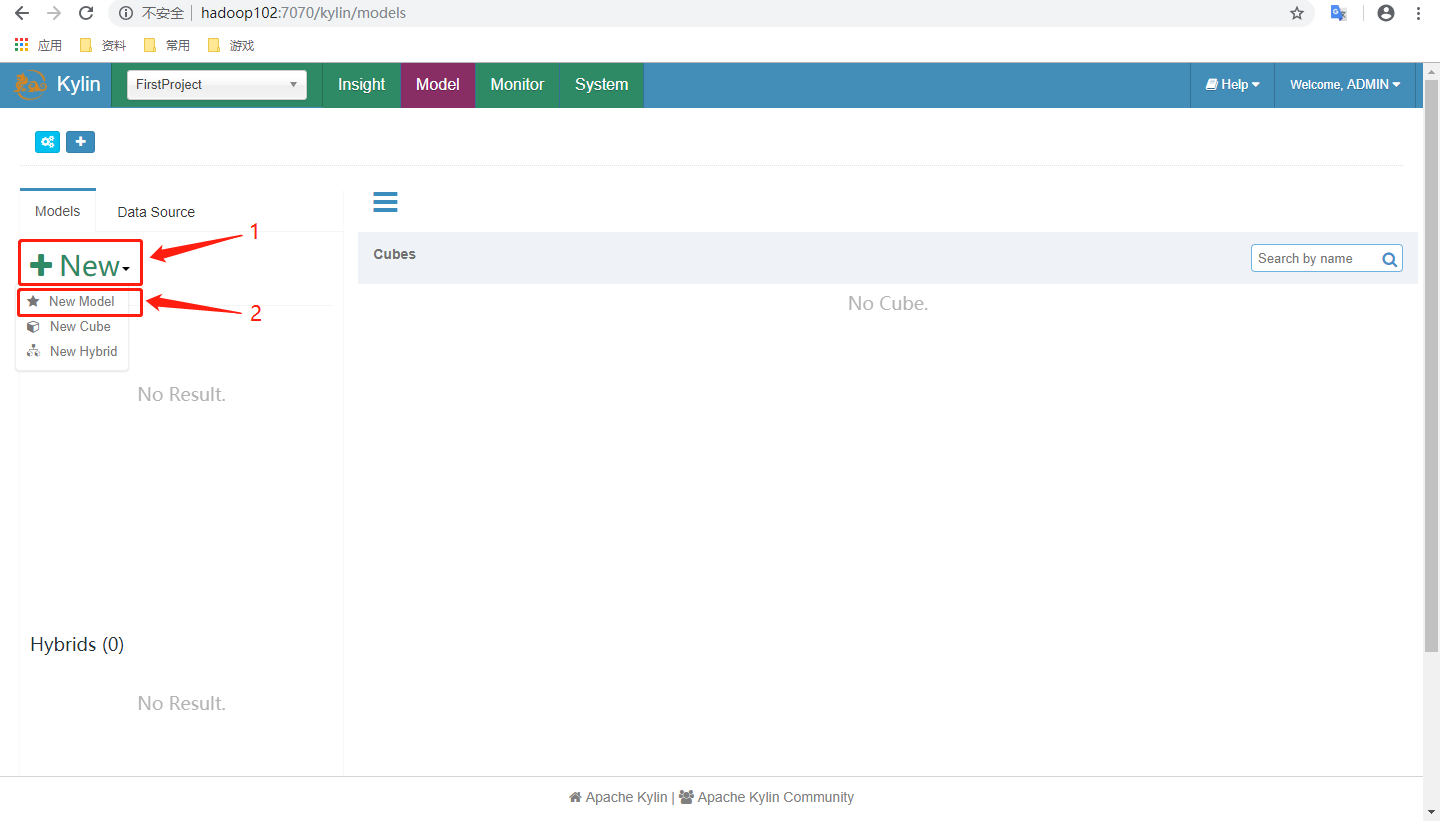


## 3.3 创建Model

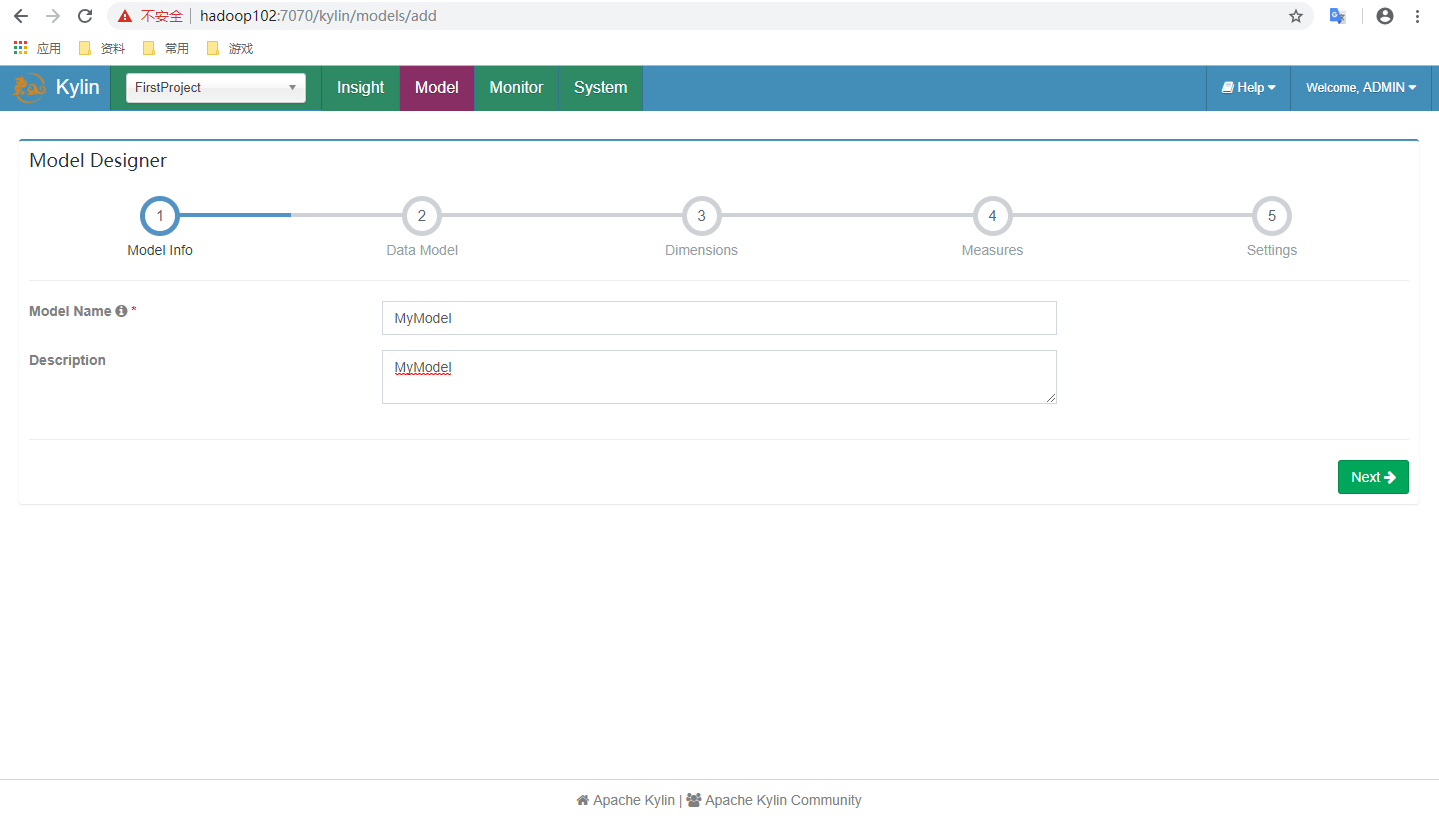
1）回到Models页面



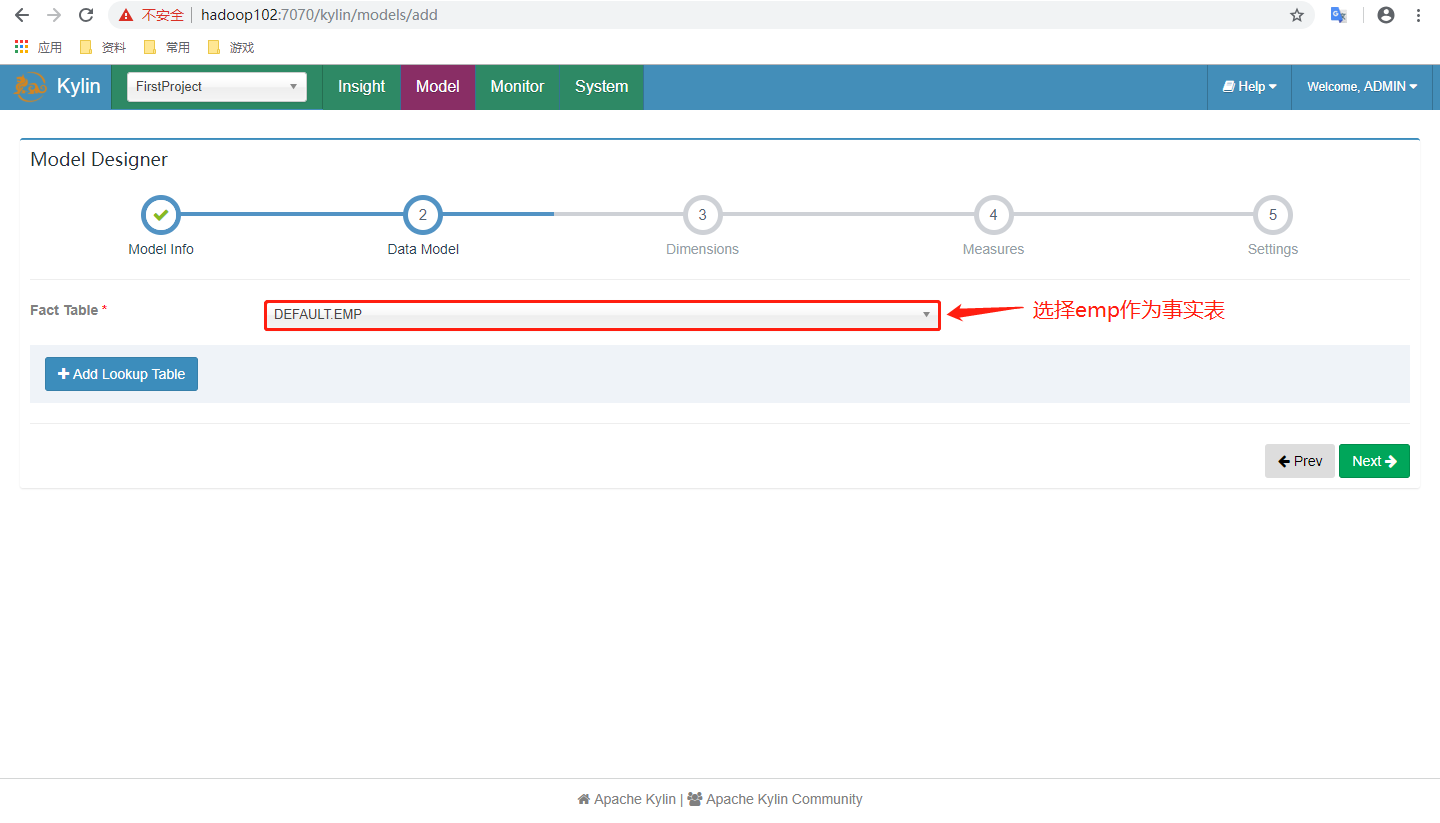
2）点击New按钮后点击New Model



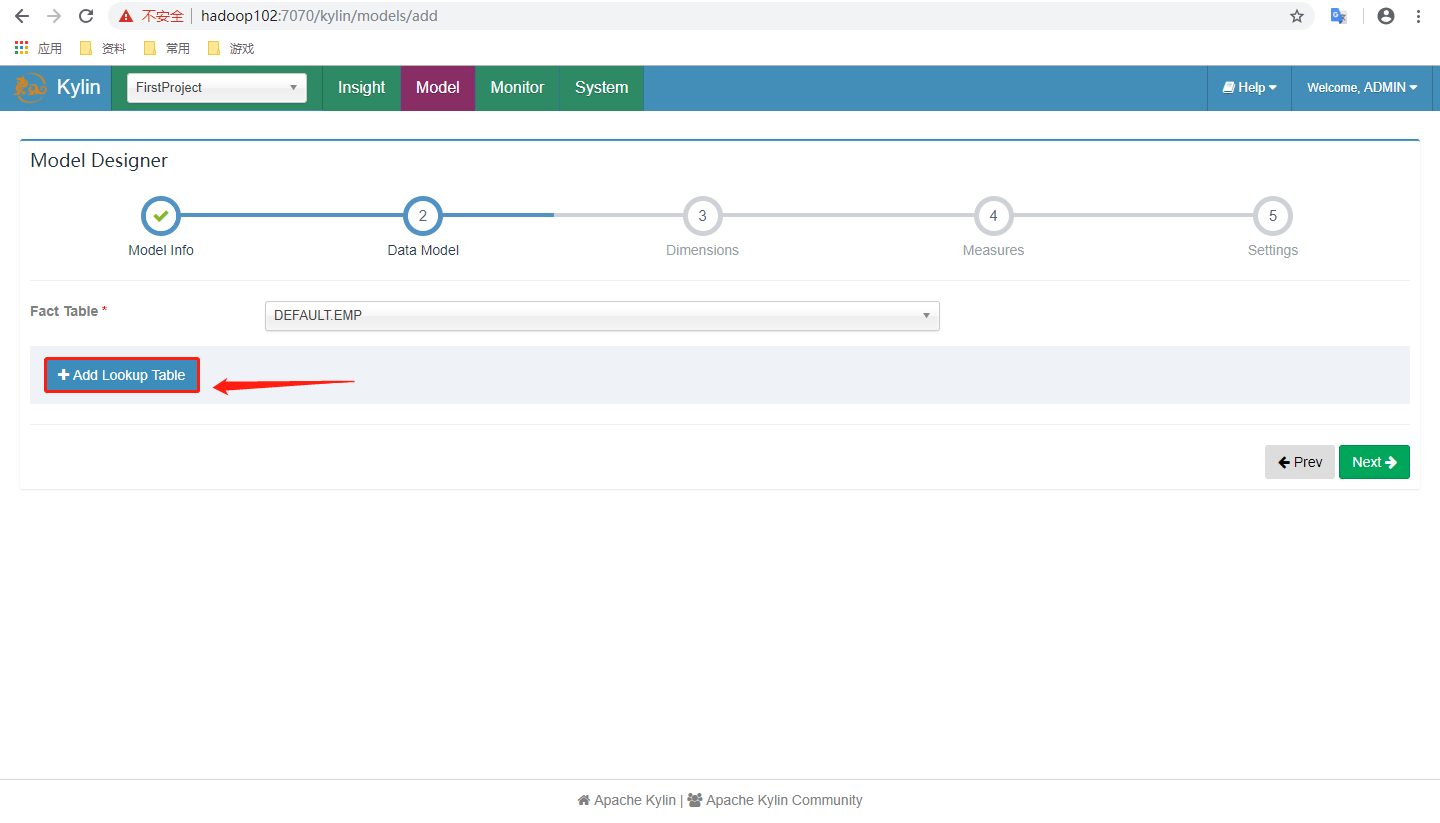
3）填写Model名称及描述后Next



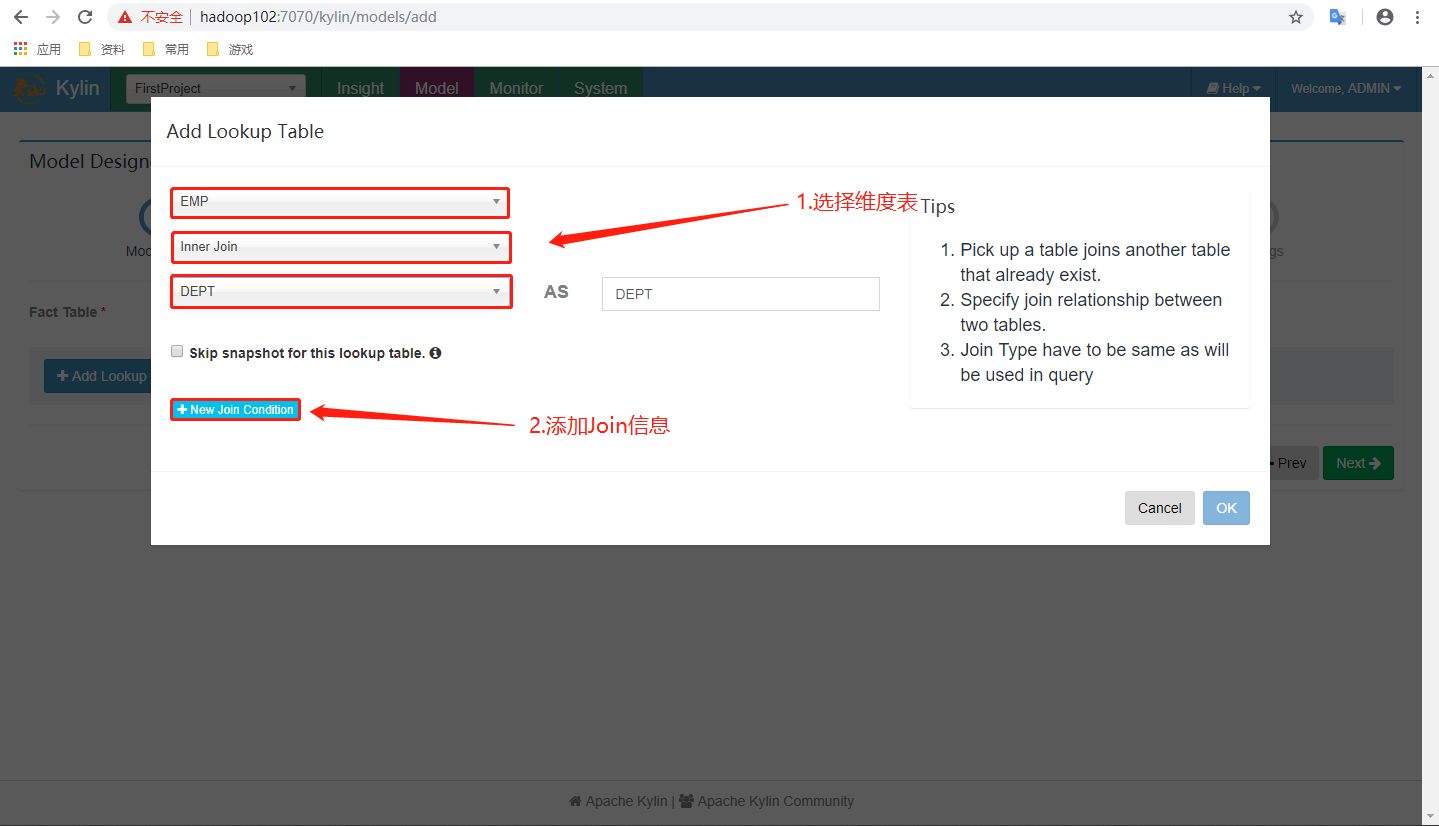
4）选择事实表

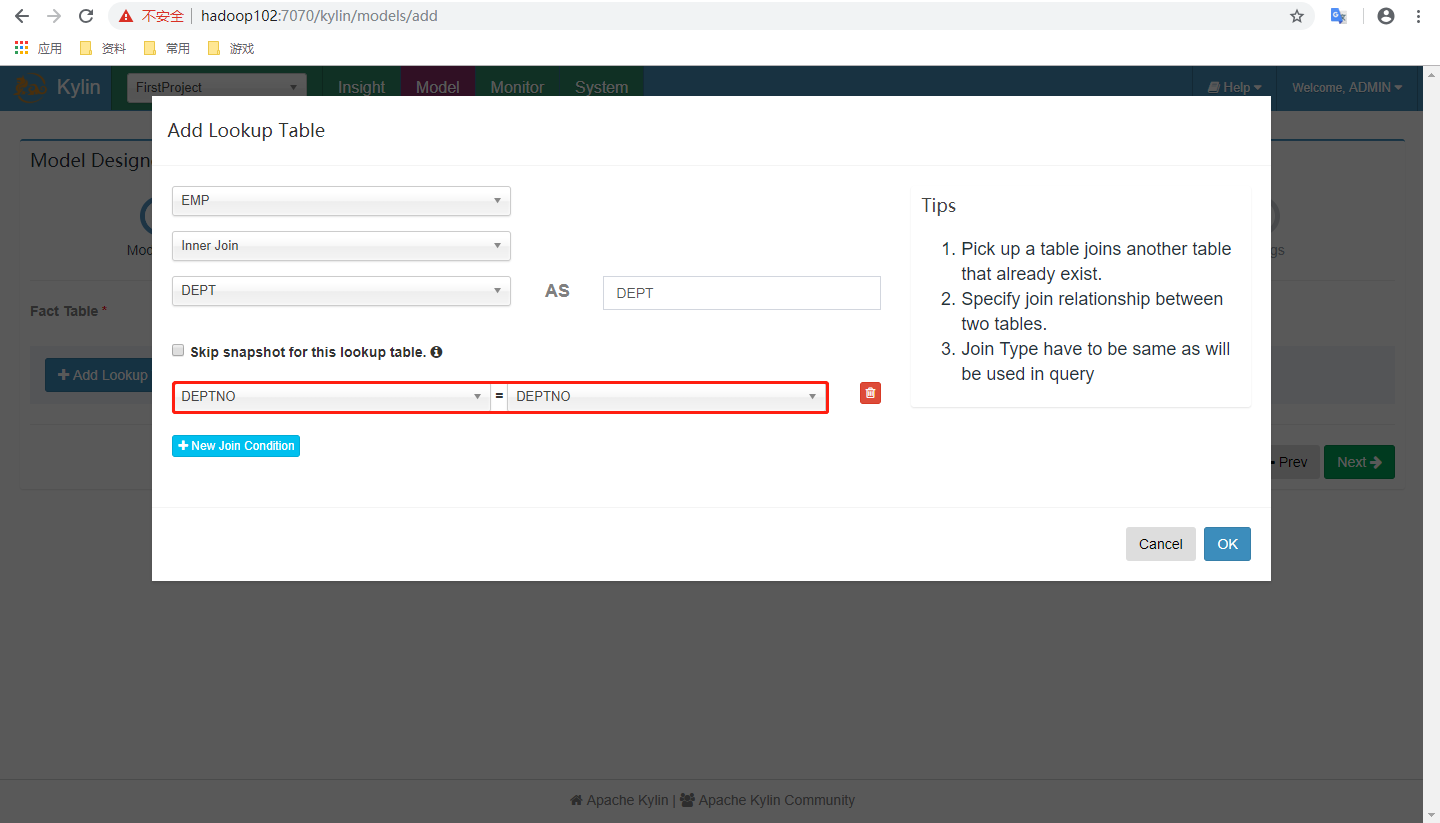


5）添加维度表

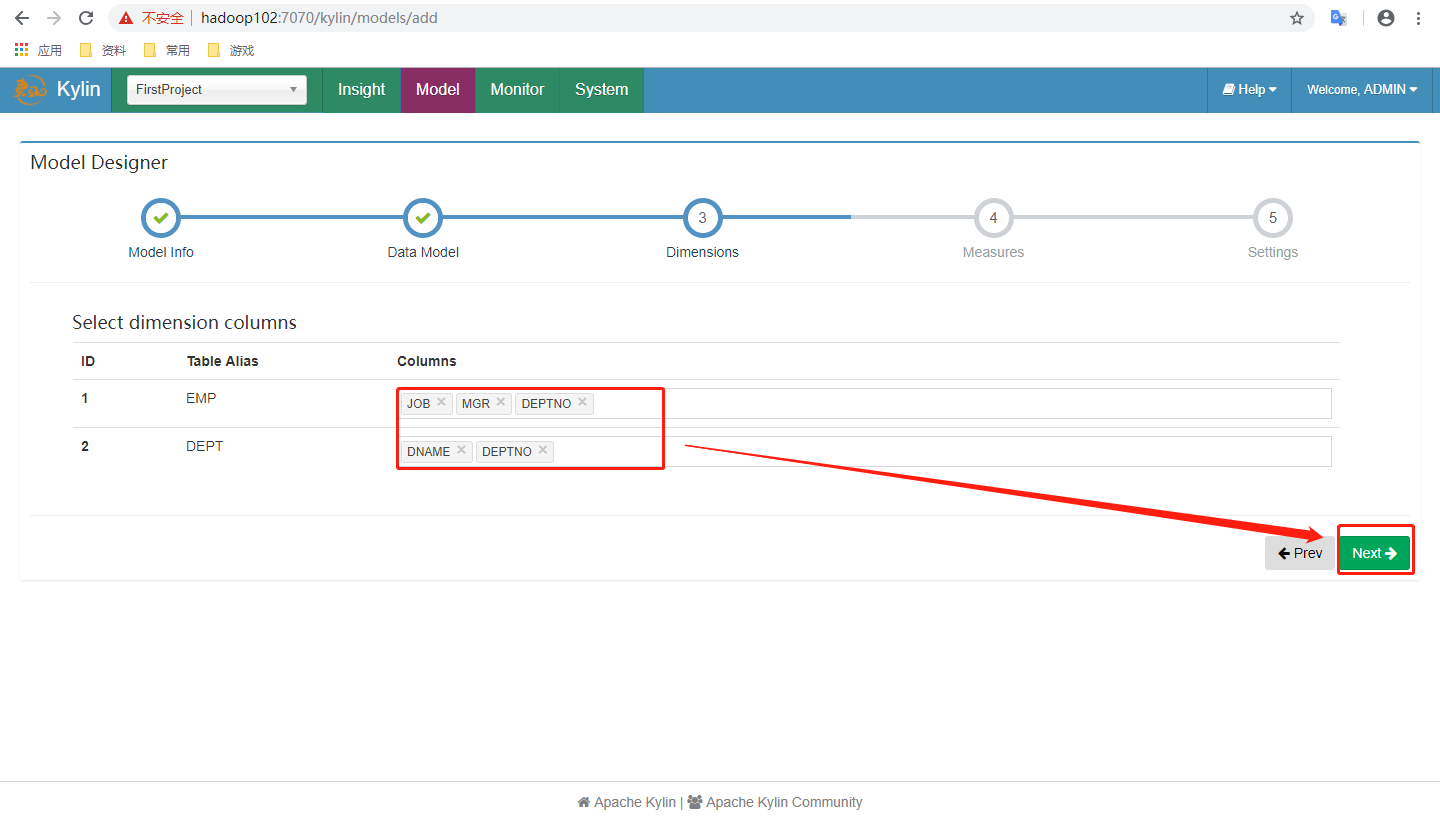


6）选择添加的维度表及join字段

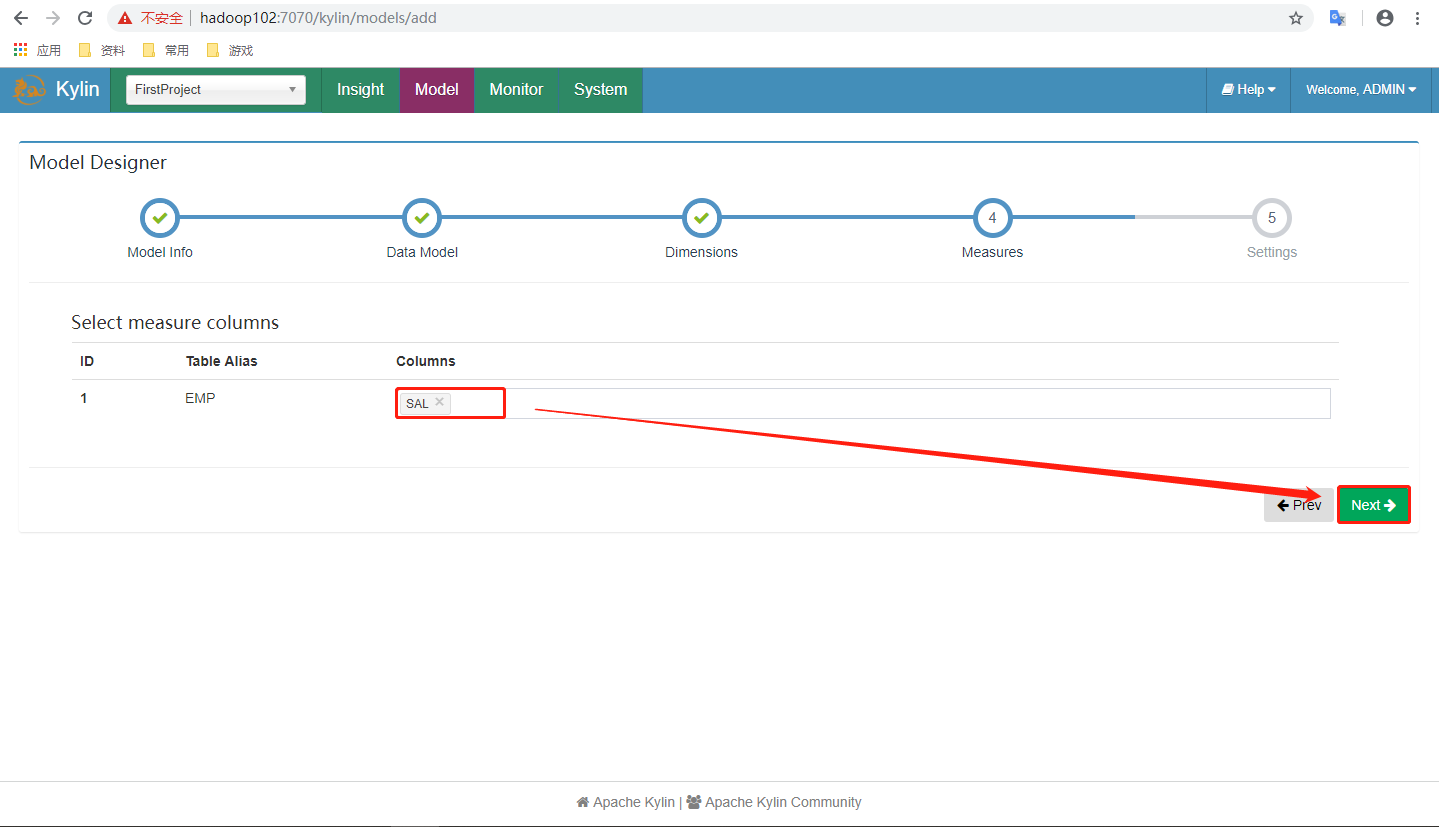




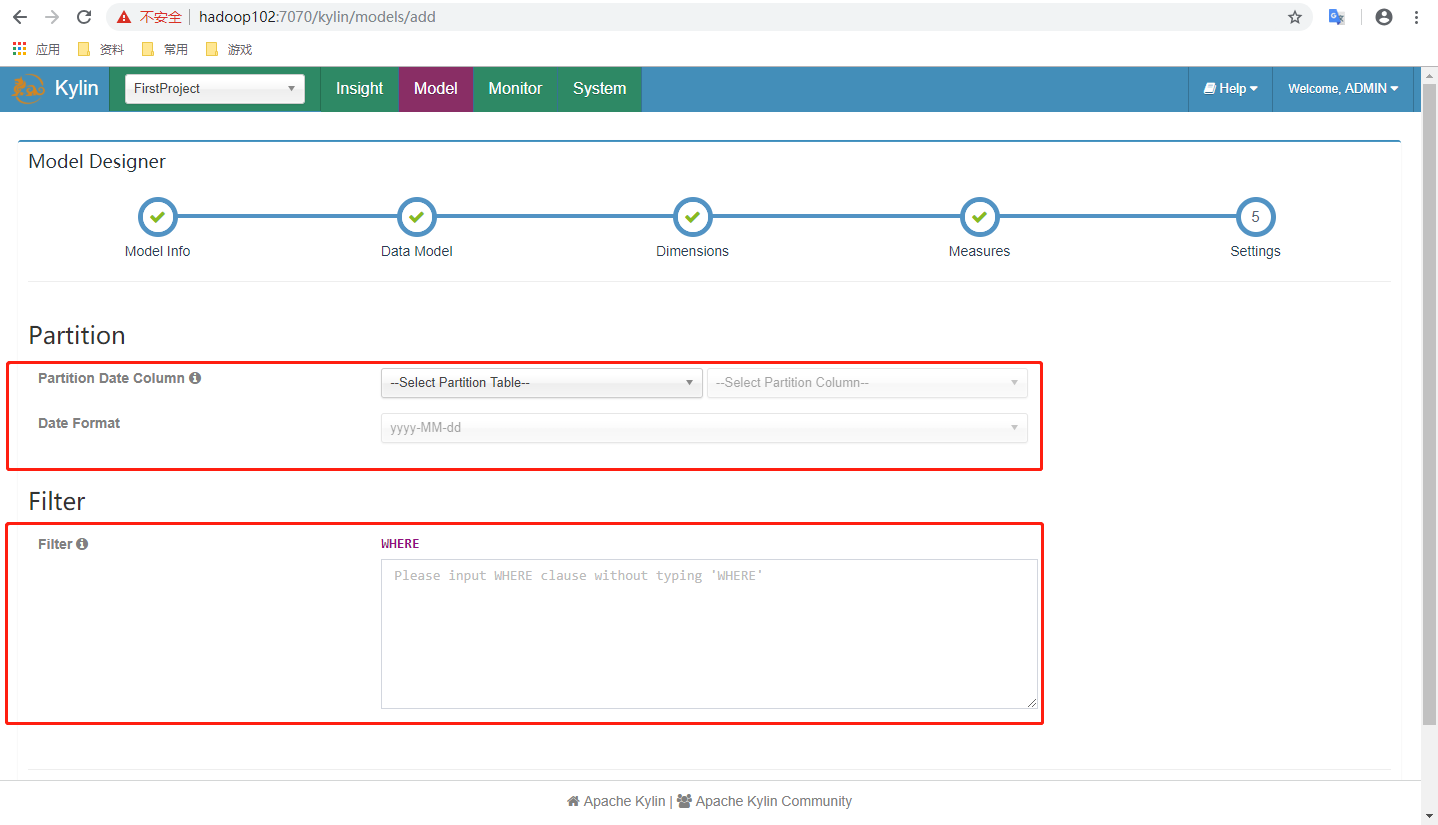
7）选择维度信息



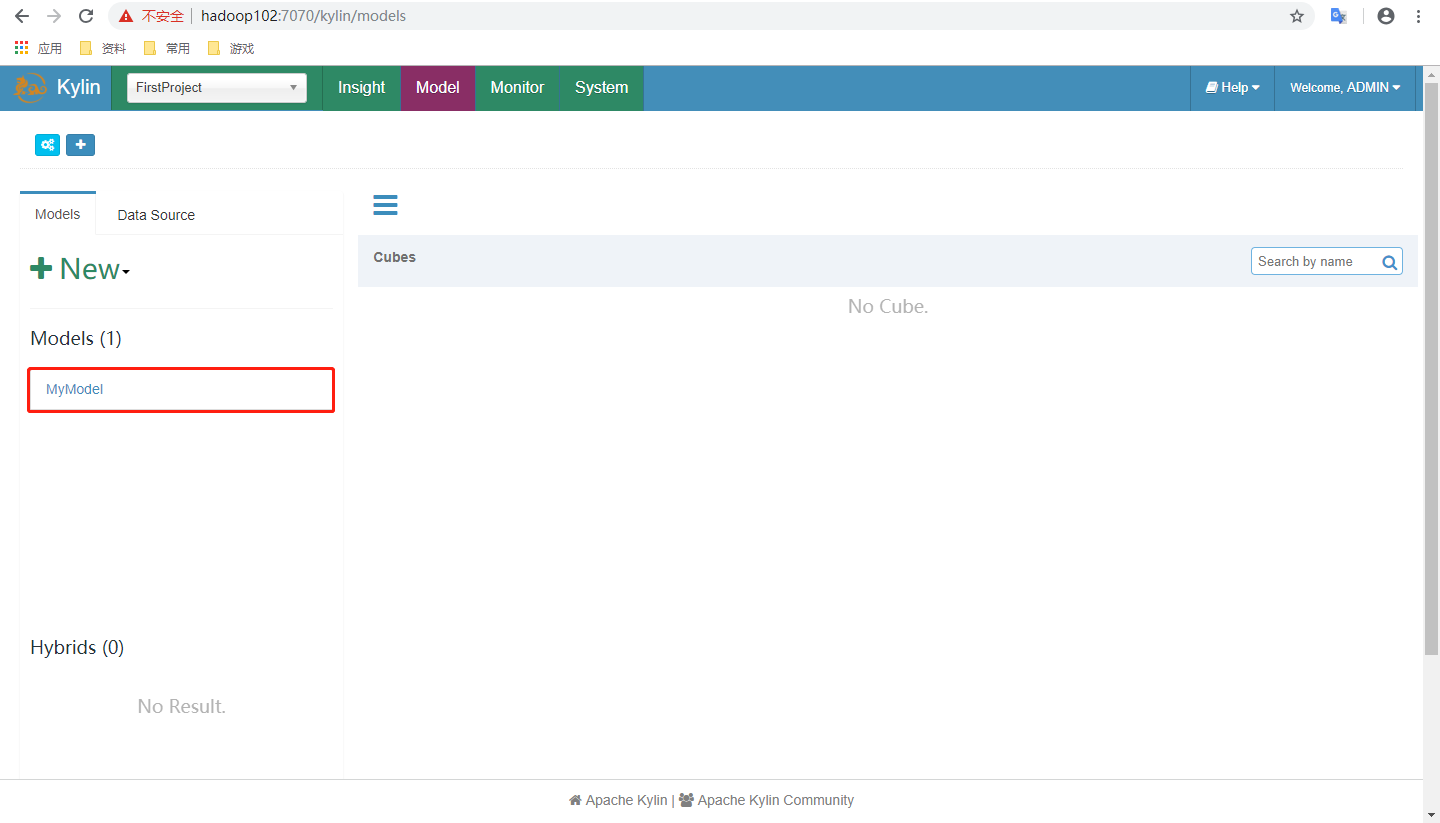
8）选择度量信息



9）添加分区信息及过滤条件之后“Save”

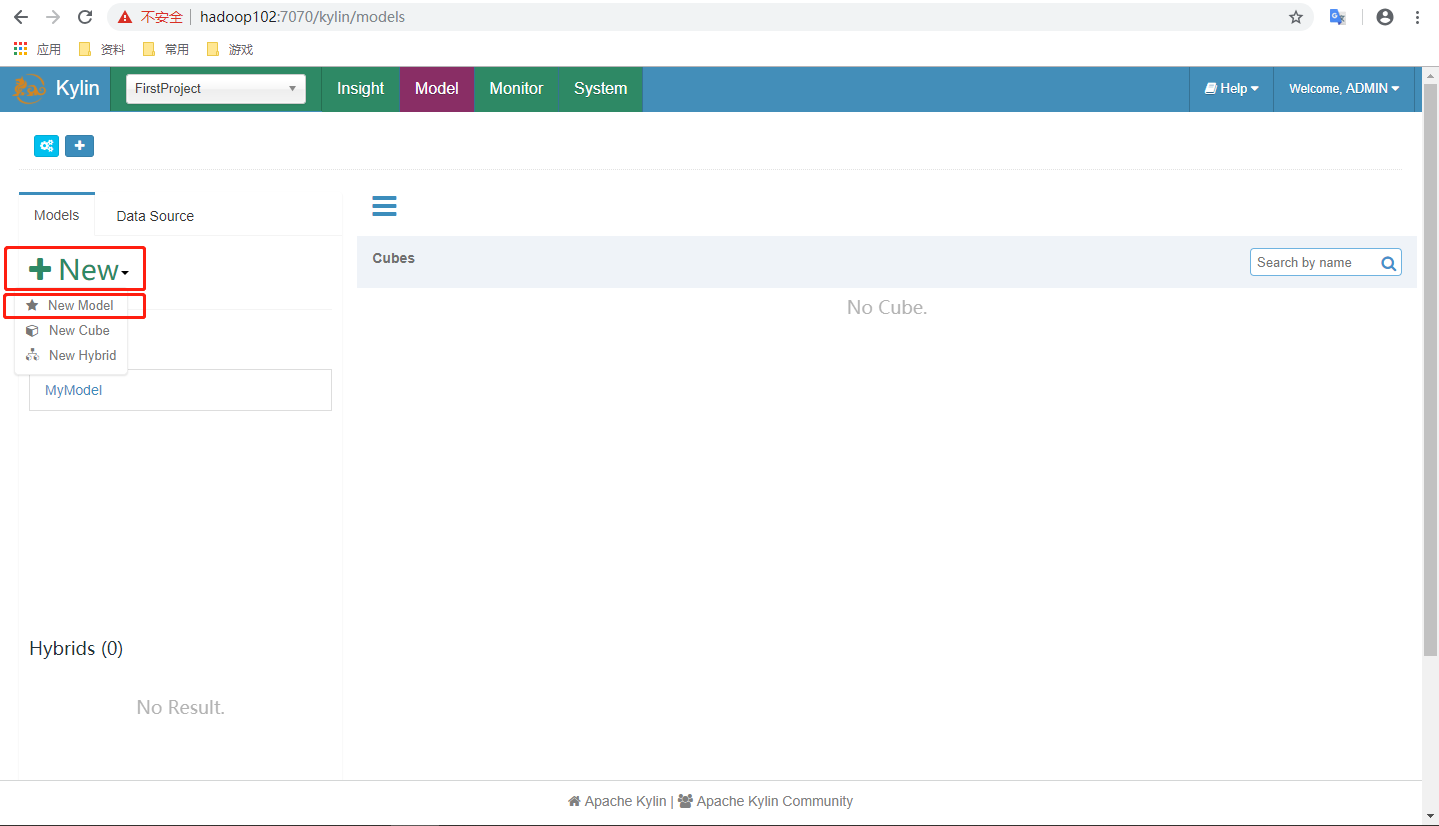


10）创建Model完成

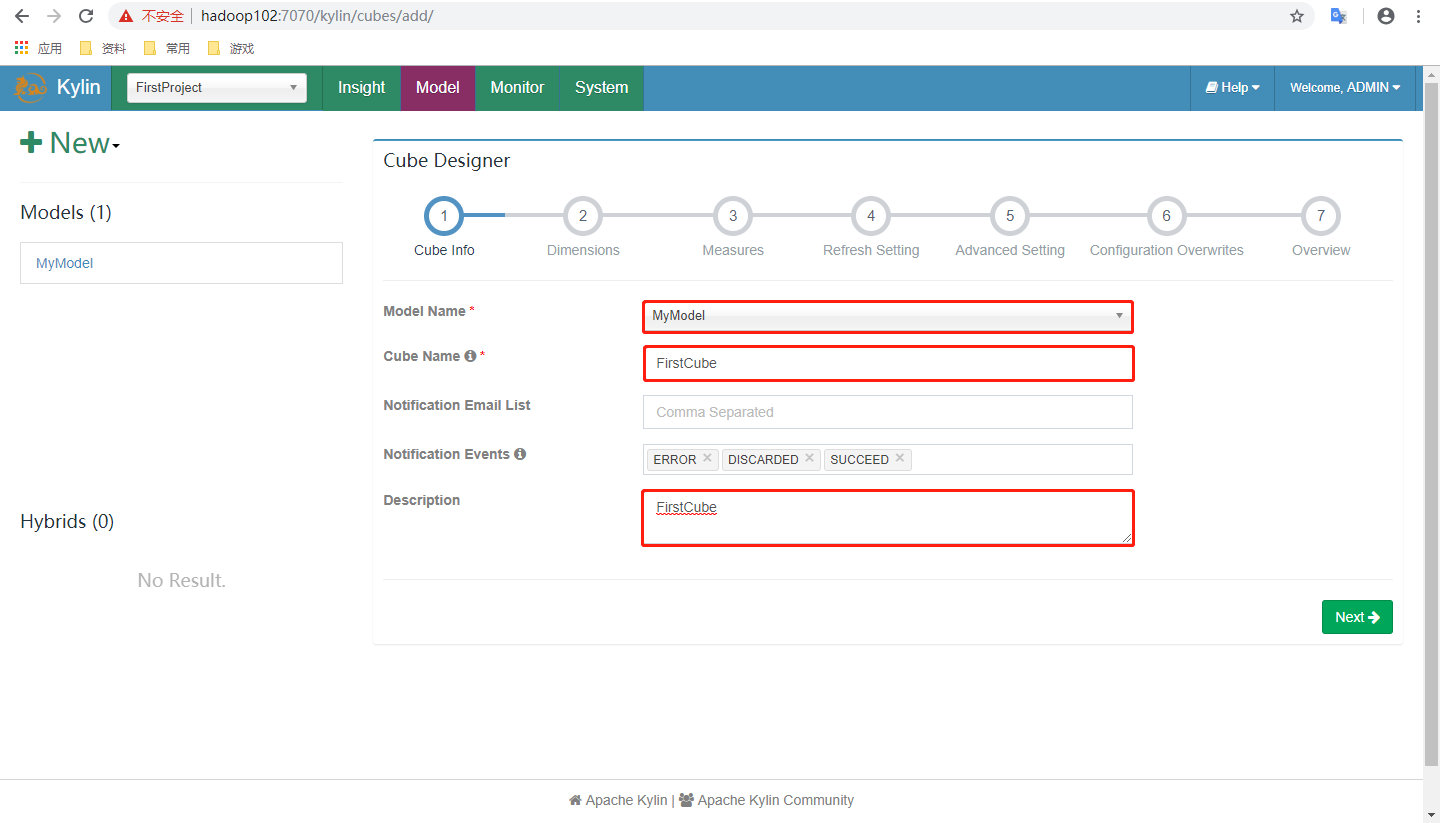


## 3.4 创建Cube

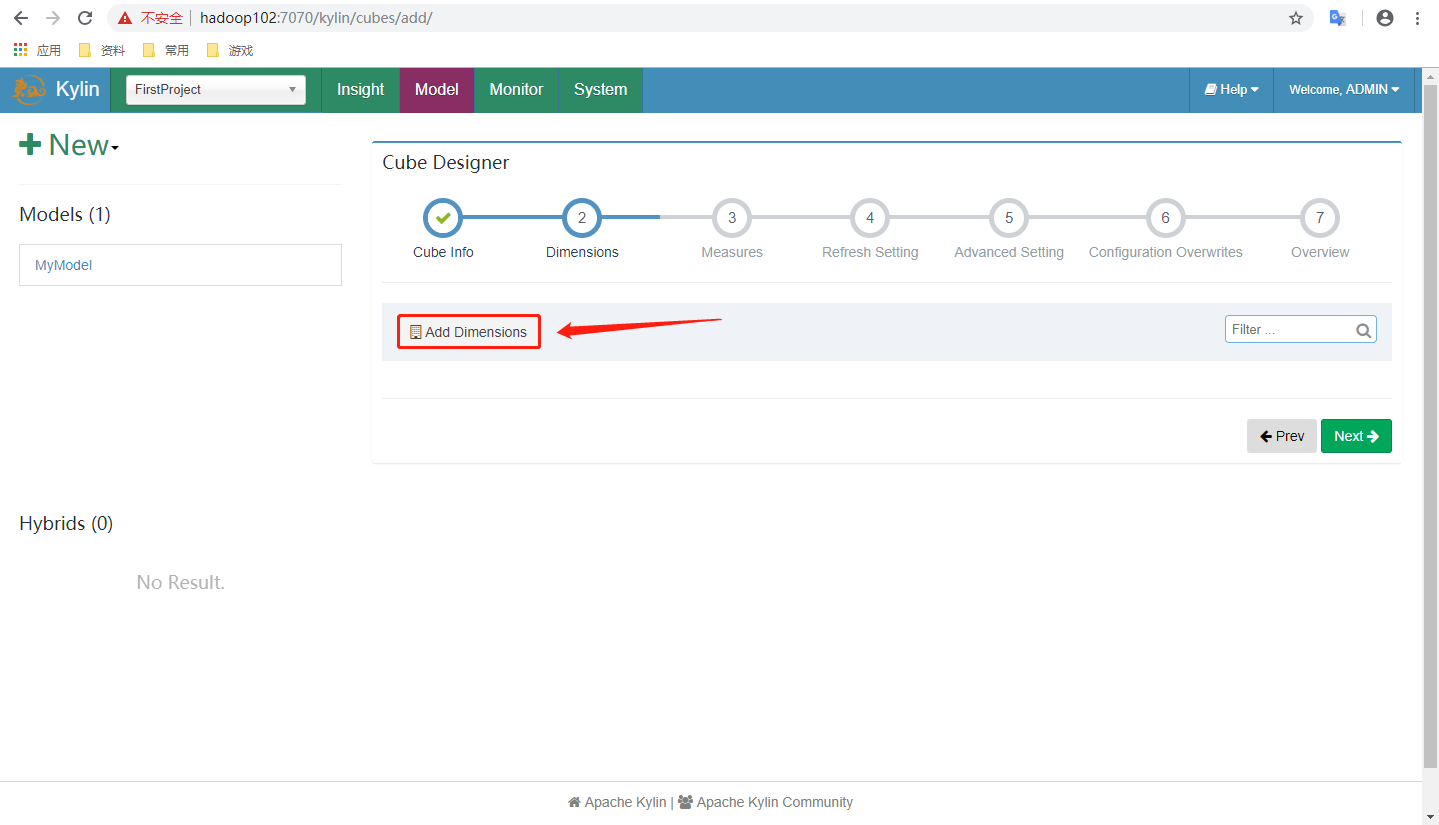
1）点击New按钮然后选择New Cube

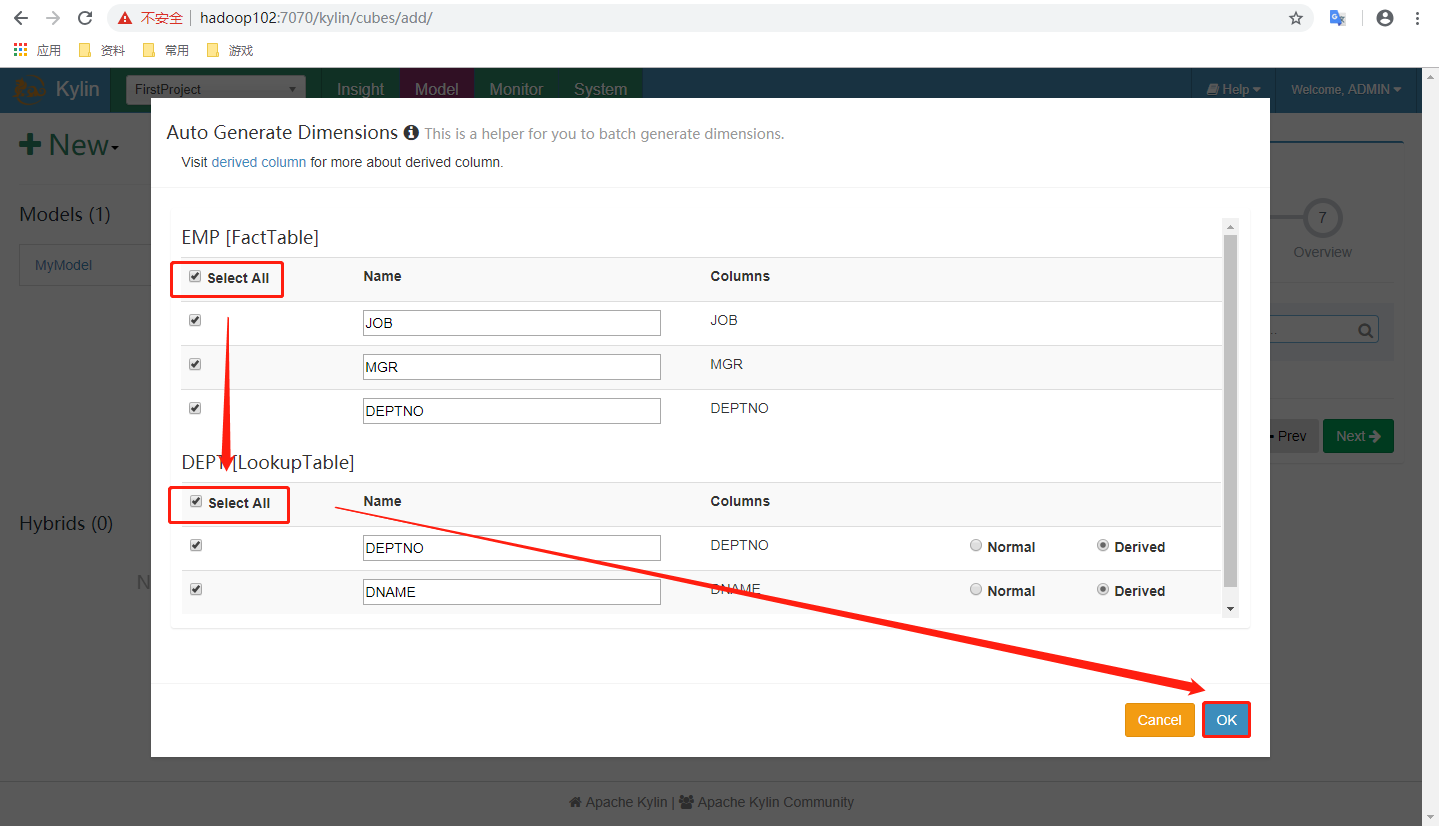


2）选择Model及填写Cube Name

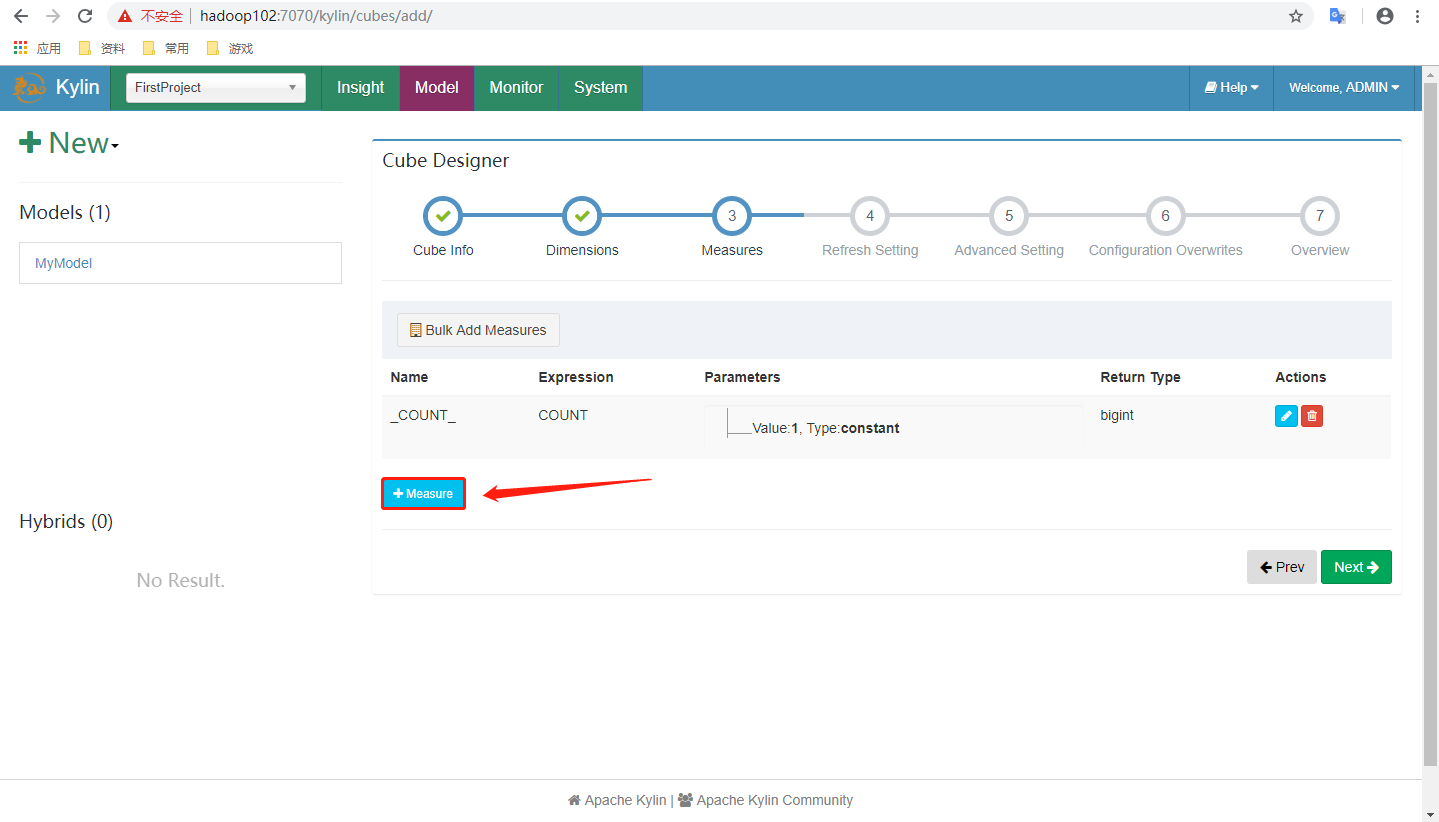


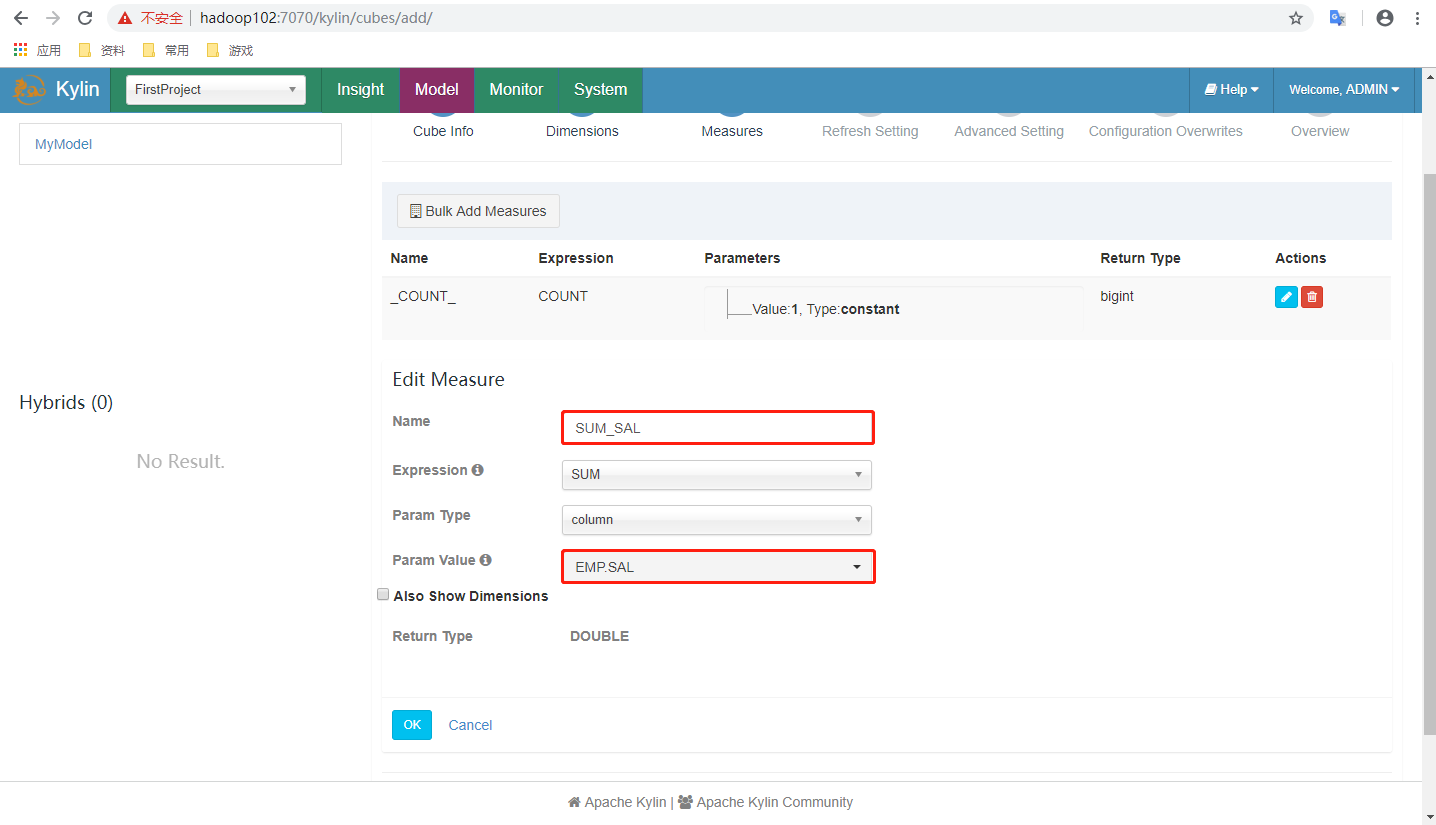
3）添加维度



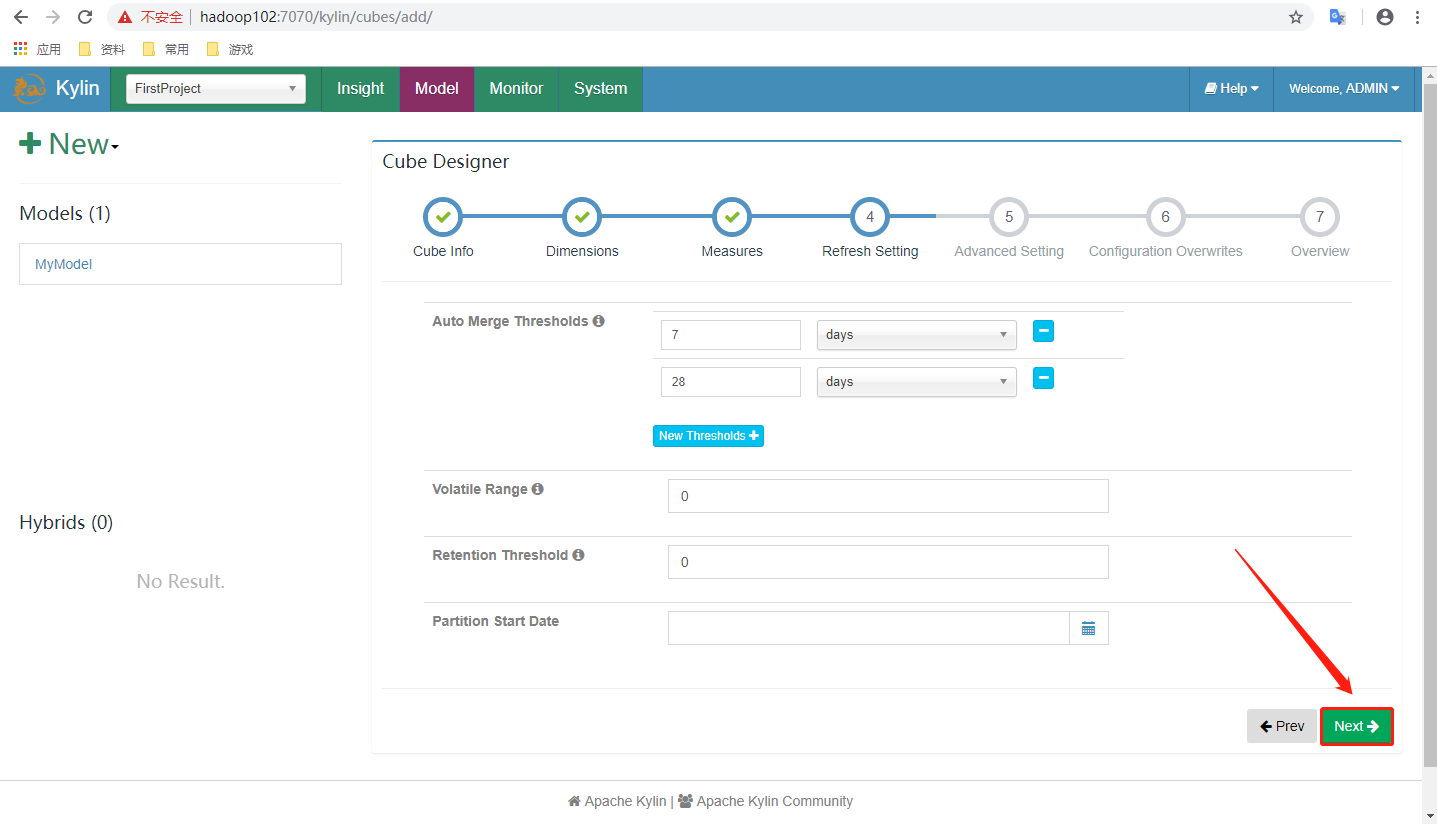


4）添加需要做预计算的内容

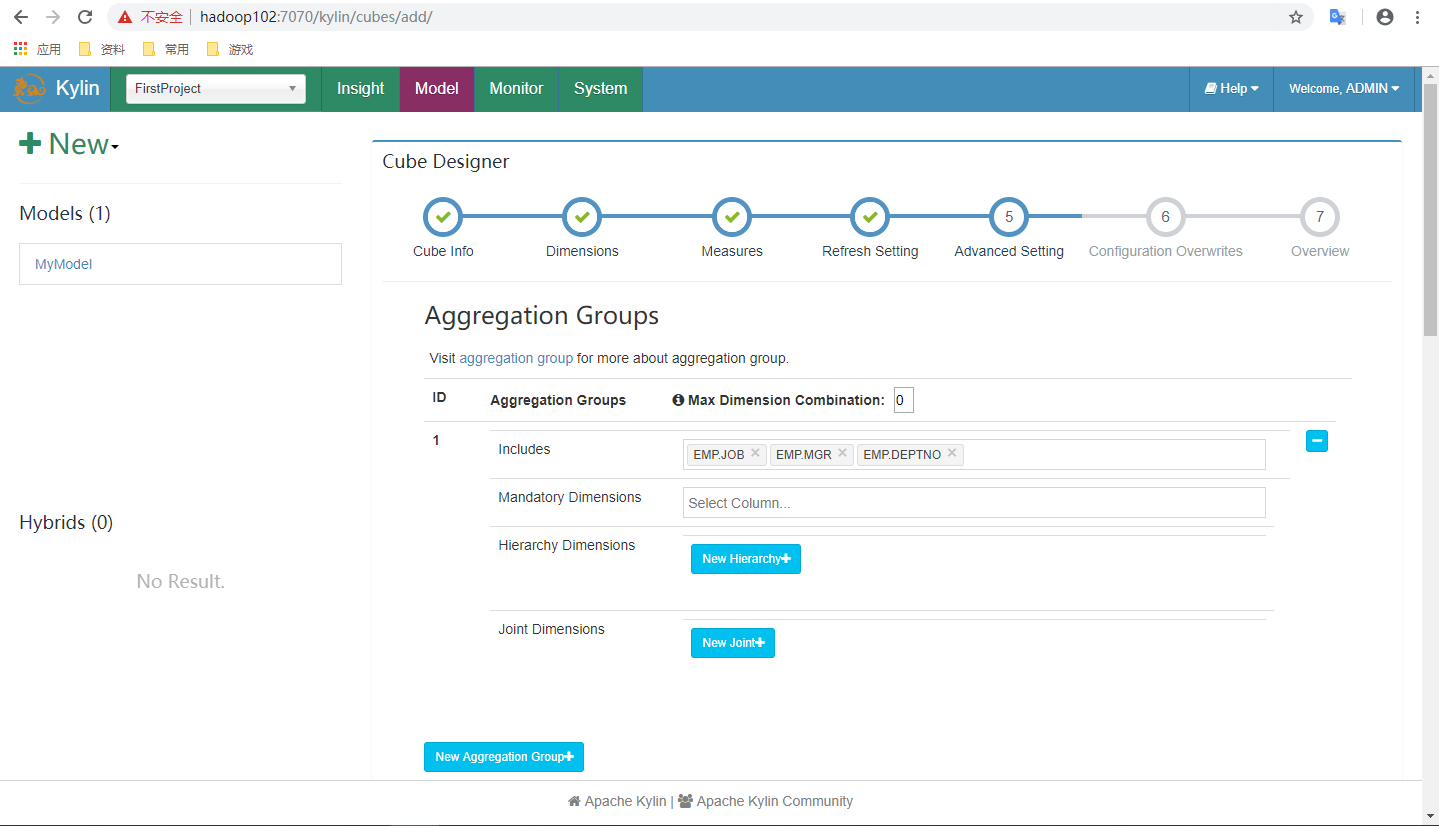




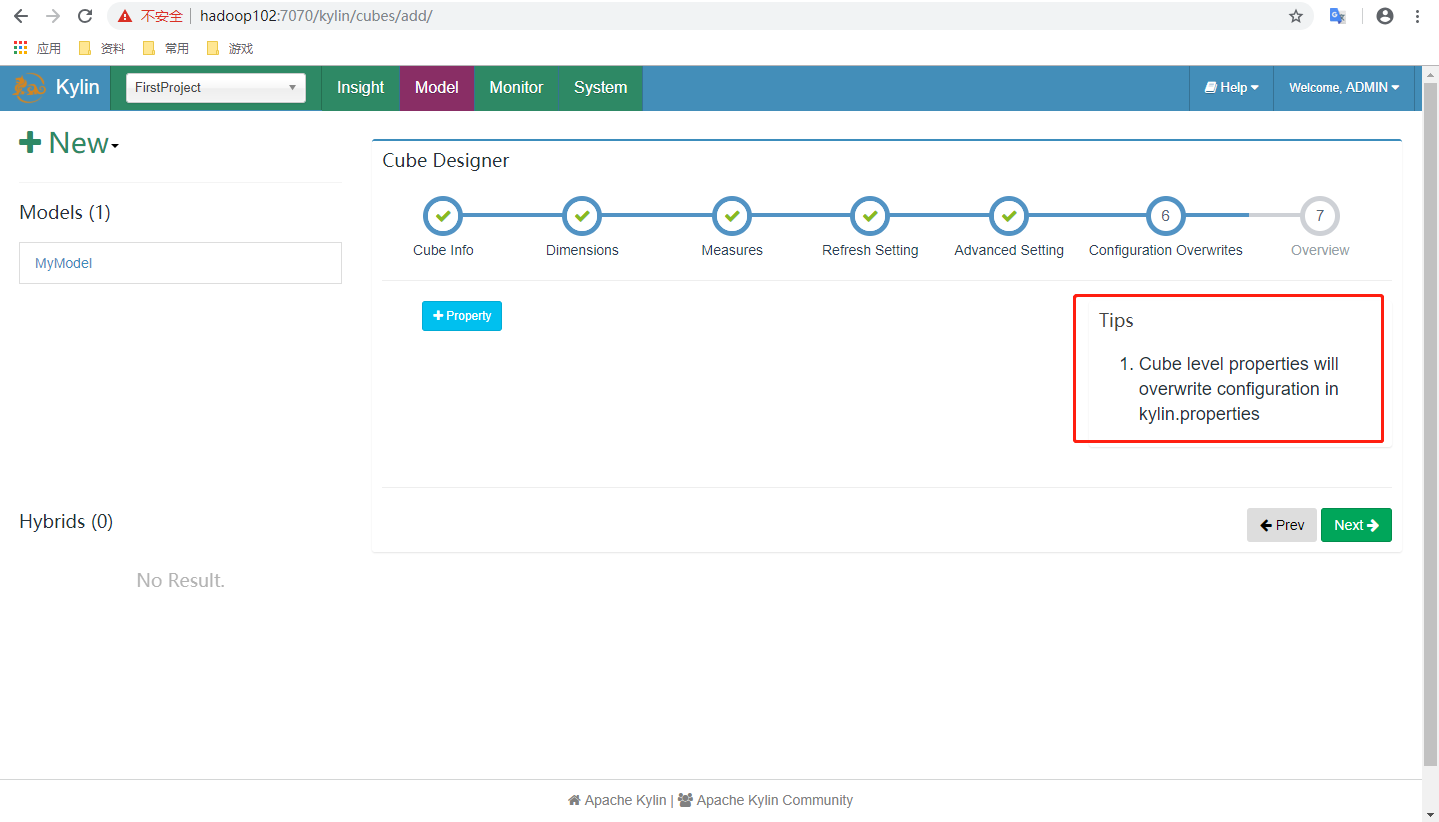
5）动态更新相关（默认）



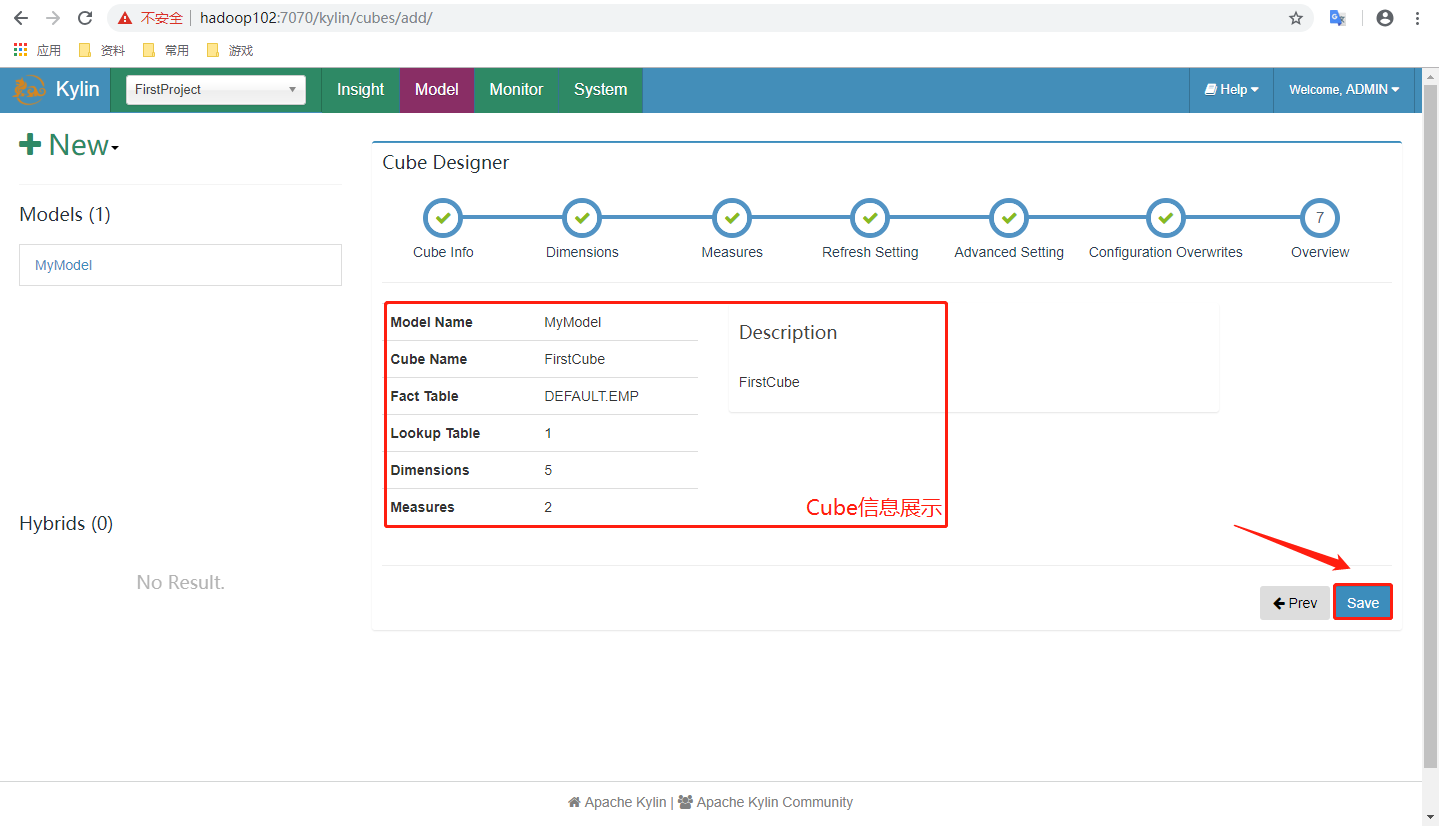
6）高阶模块（默认）



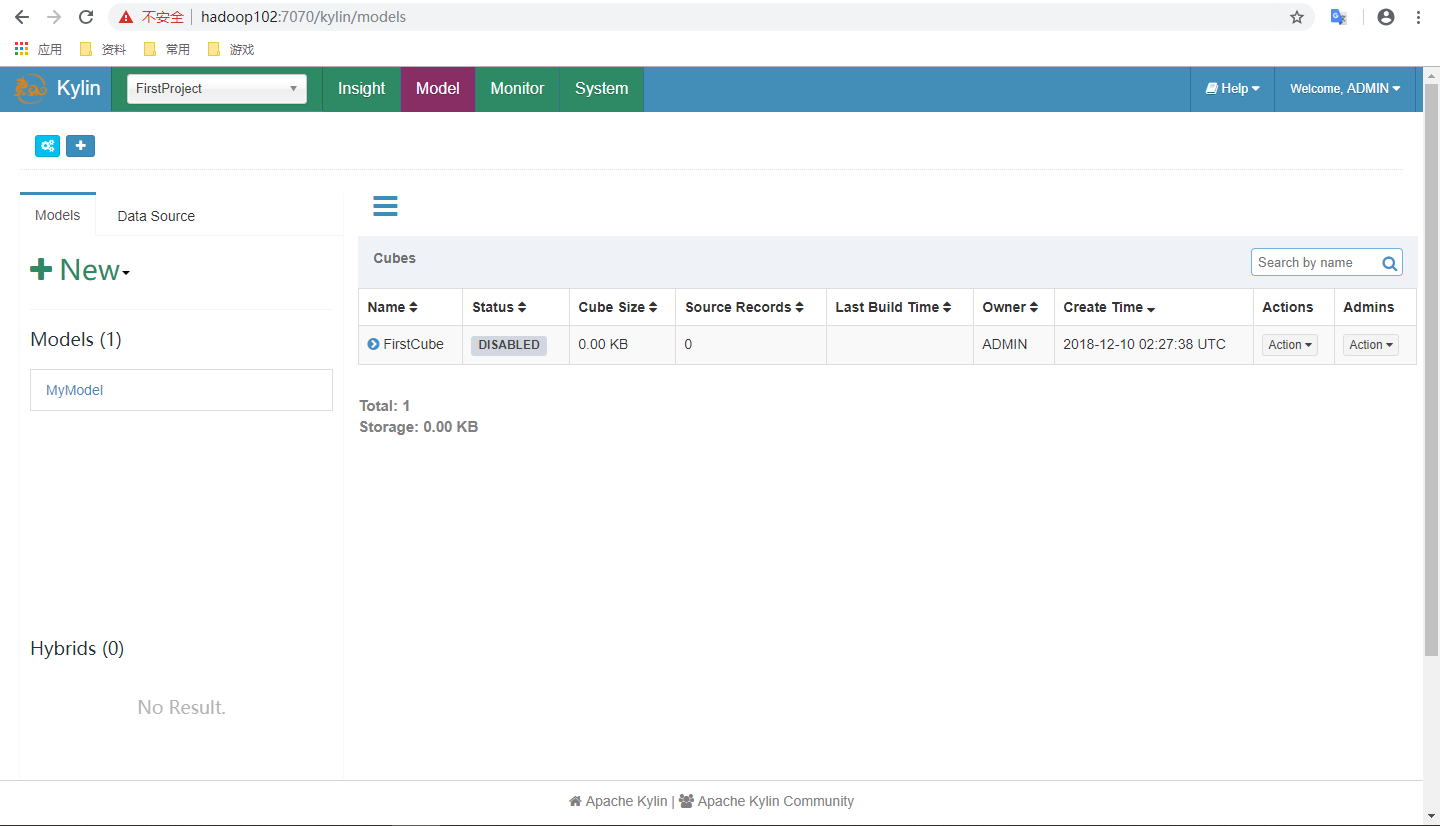
7）需要修改的配置



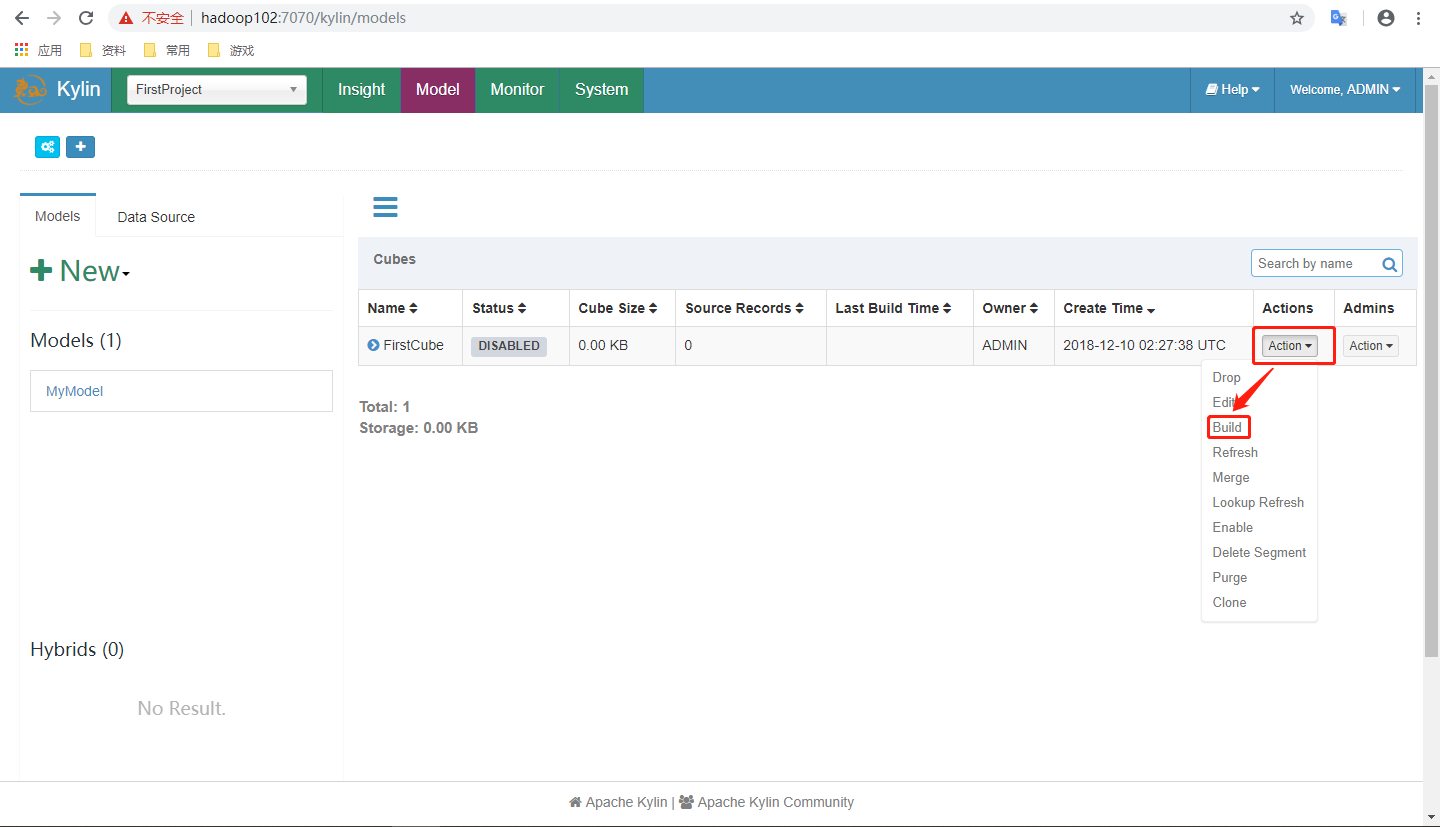
8）Cube信息展示



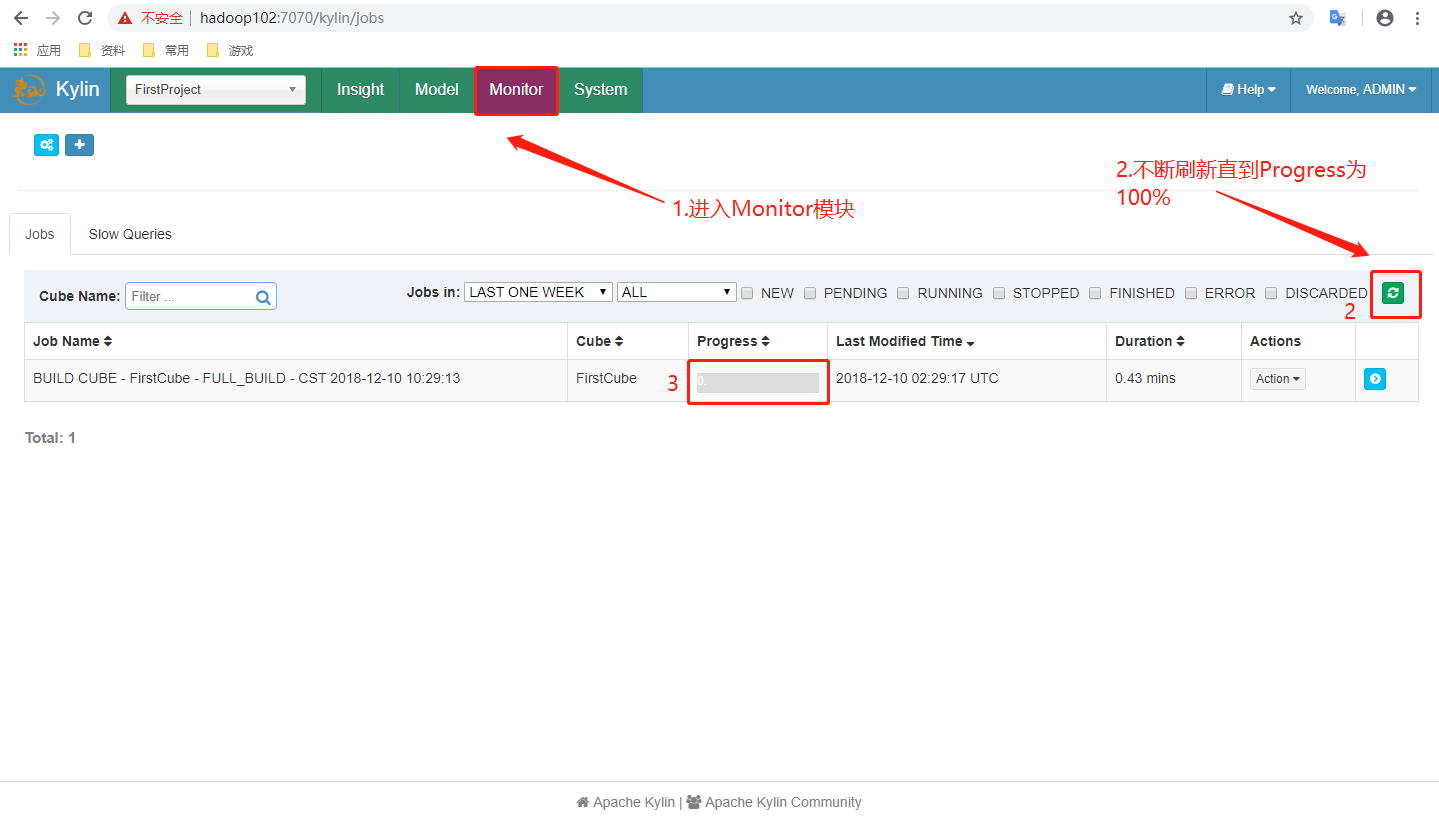
9）Cube配置完成



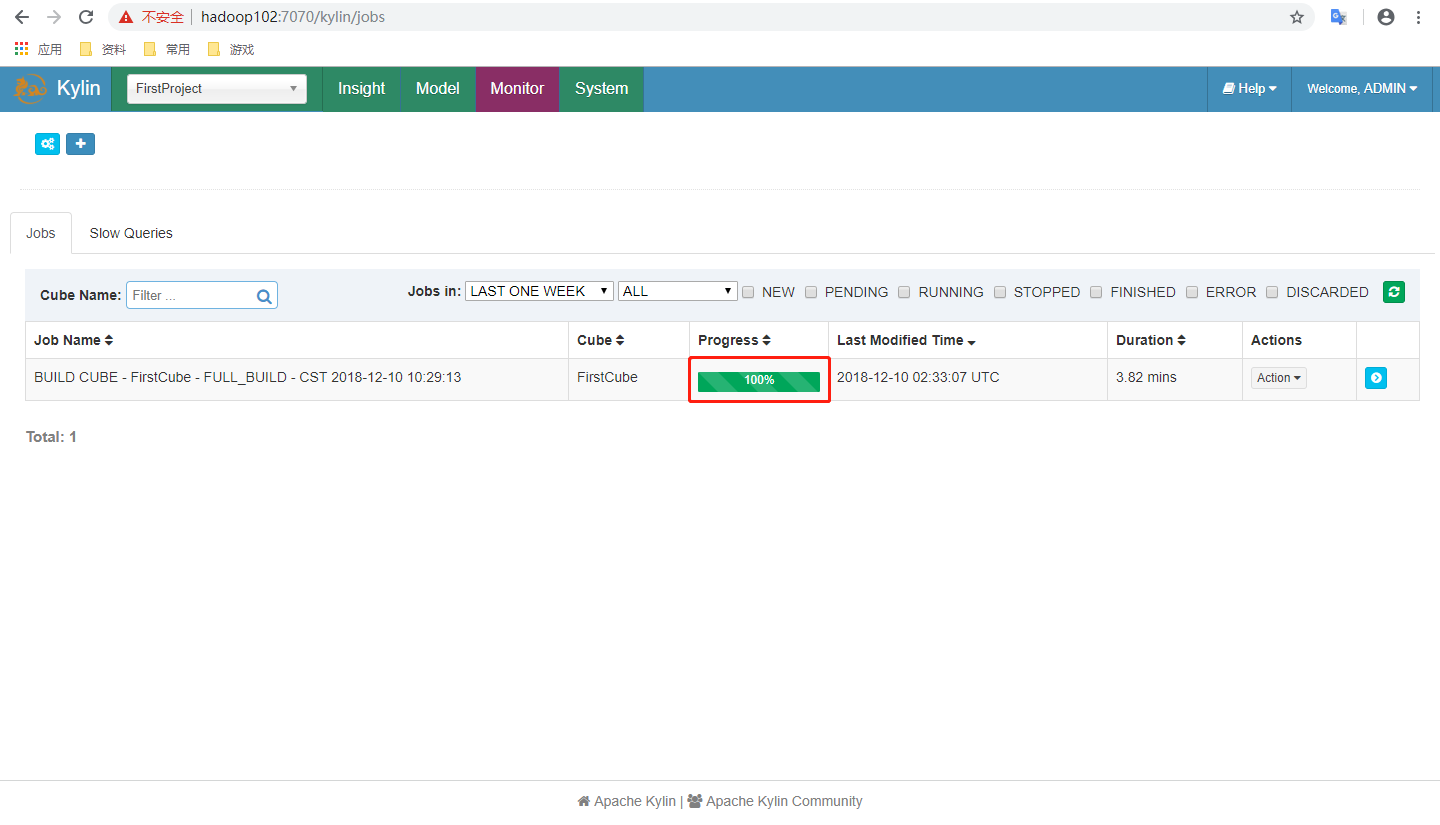
10）触发预计算



11）查看Build进度



12）构建Cube完成



## 3.5 Hive和Kylin性能对比

需求：根据部门名称[dname]统计员工薪资总数[sum（sal）]

### 3.5.1 Hive查询

hive> select dname,sum(sal) from emp e join dept d on e.deptno = d.deptno group by dname;

Query ID = bawei\_20181210104140\_4931b735-5bad-4a4f-bce6-67985b8fe30a

Total jobs = 1

SLF4J: Class path contains multiple SLF4J bindings.

… …

… …

Stage-Stage-2: Map: 1 Reduce: 1 Cumulative CPU: 3.95 sec HDFS Read: 13195 HDFS Write: 48 SUCCESS

Total MapReduce CPU Time Spent: 3 seconds 950 msec

OK

ACCOUNTING 3750.0

RESEARCH 10875.0

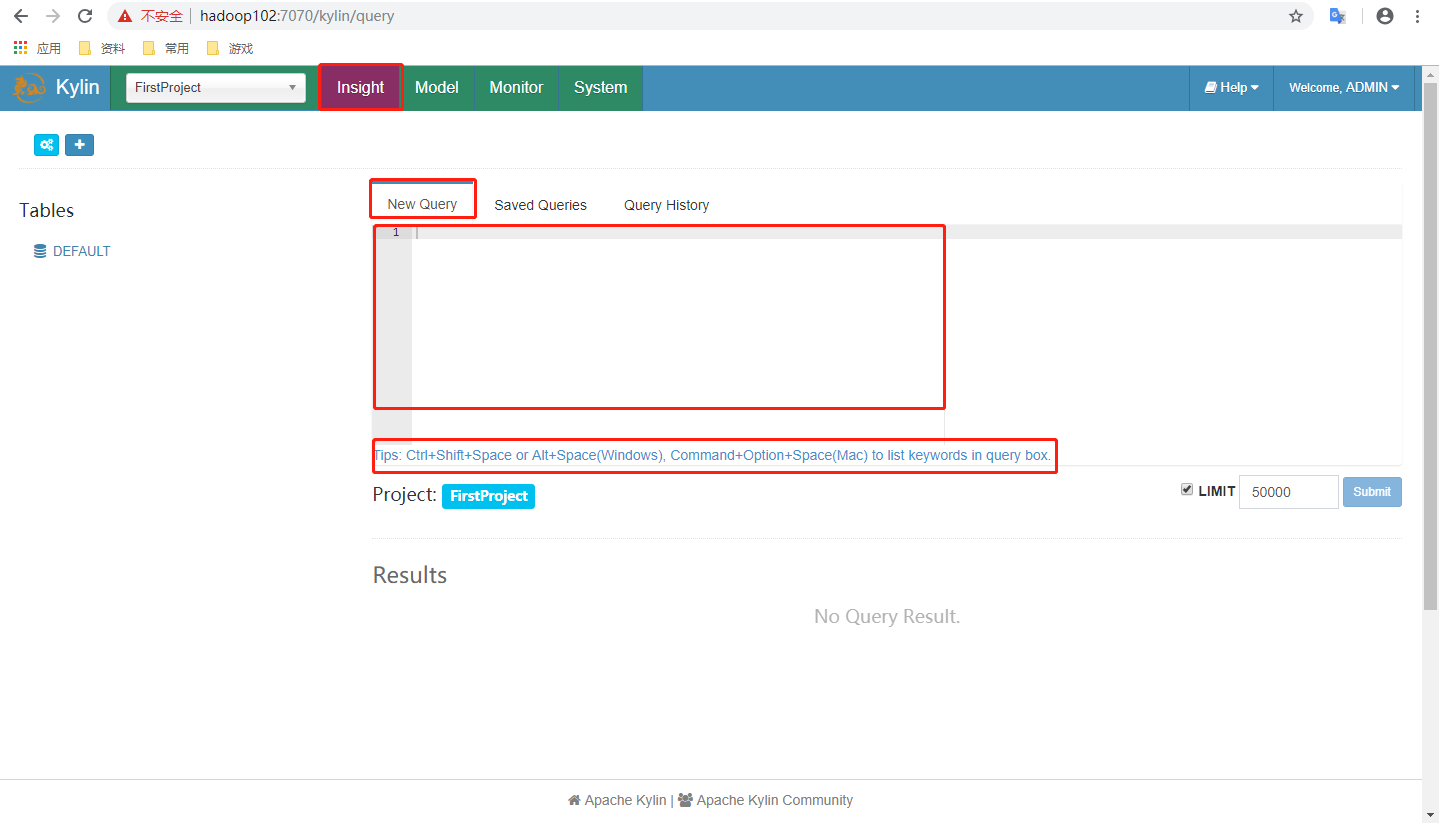
SALES 9400.0

Time taken: 23.893 seconds, Fetched: 3 row(s)

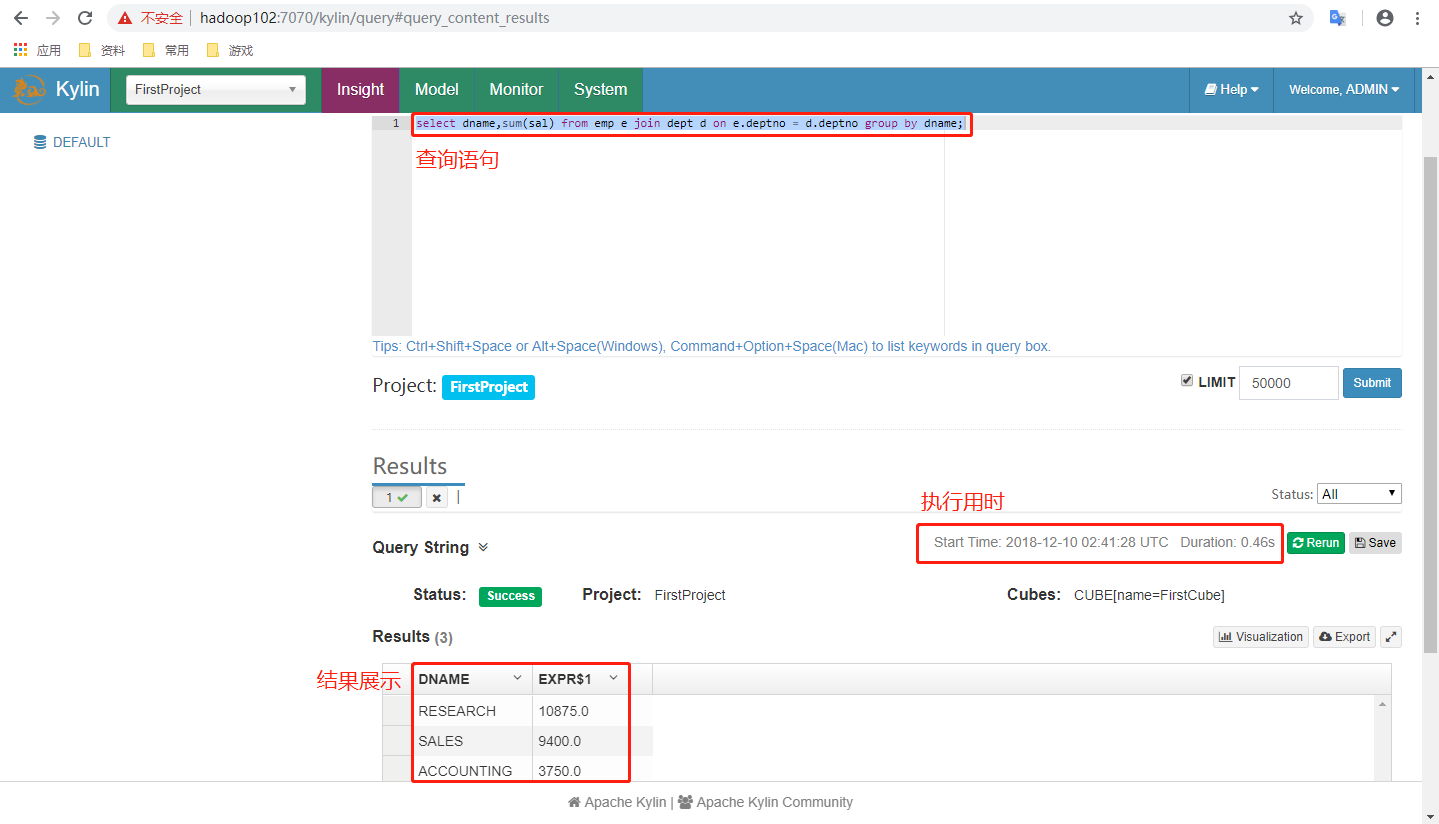
hive>

### 3.5.2 Kylin查询

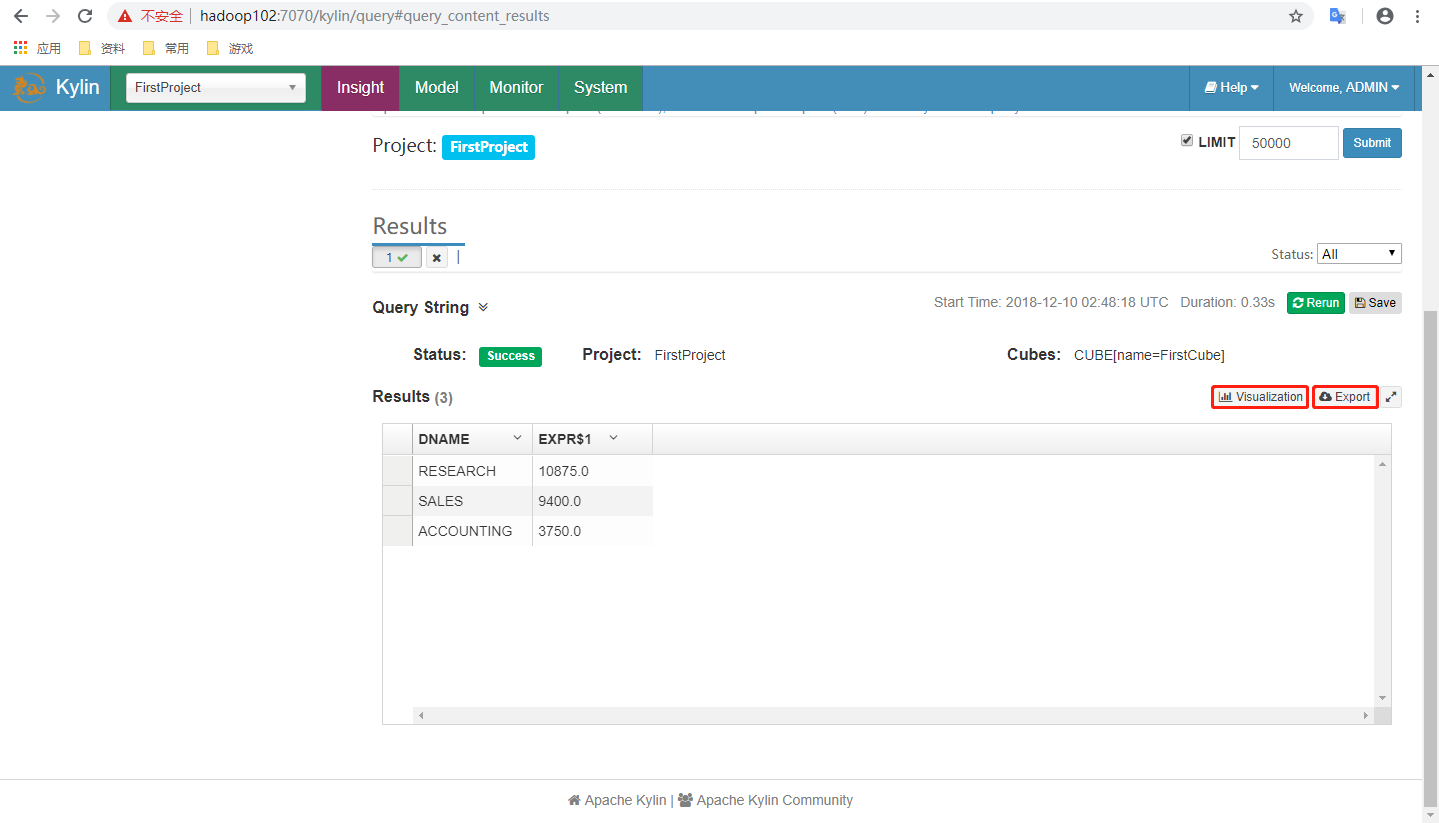
1）进入Insight页面



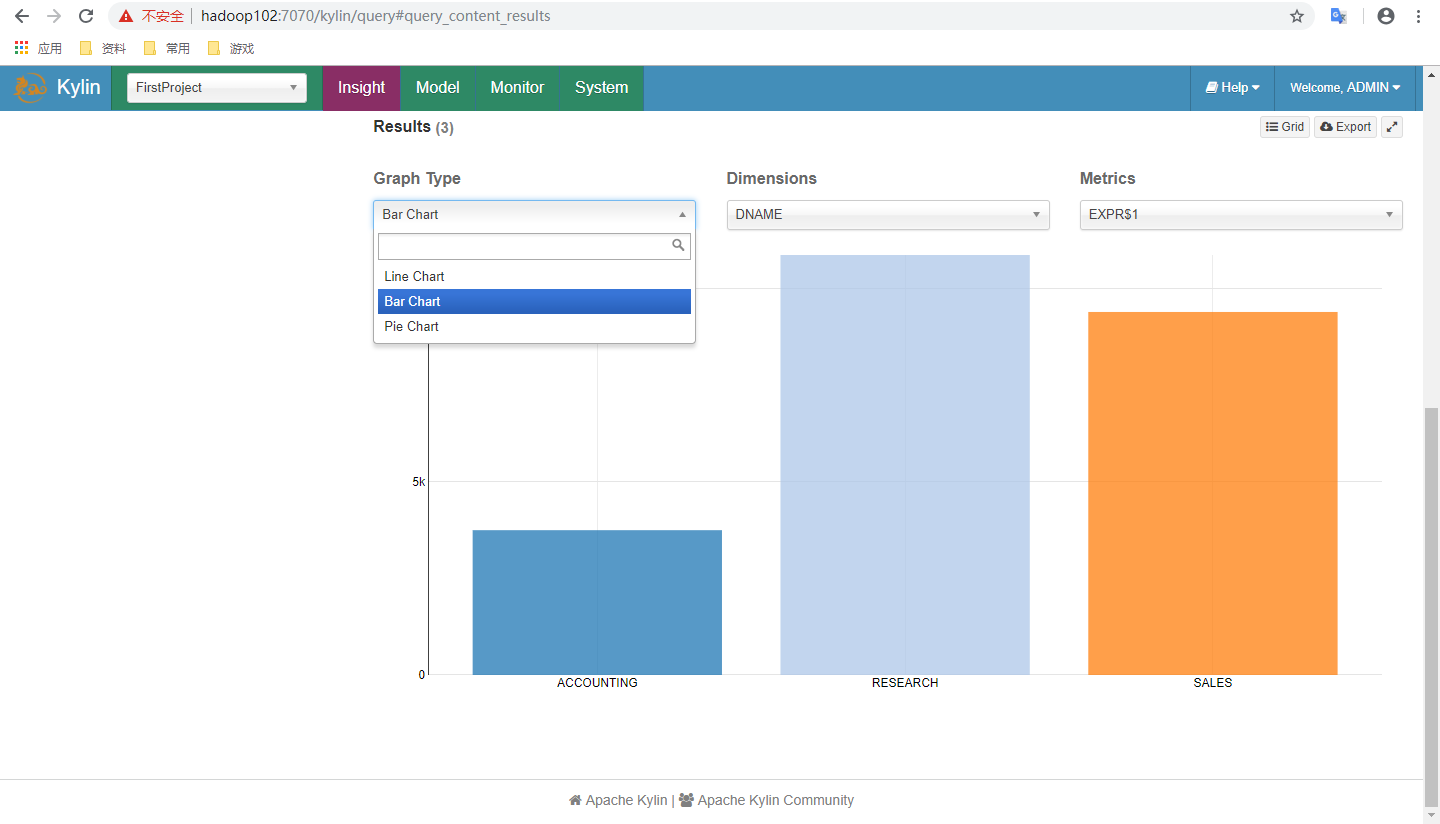
2）在New Query中输入查询语句并Submit



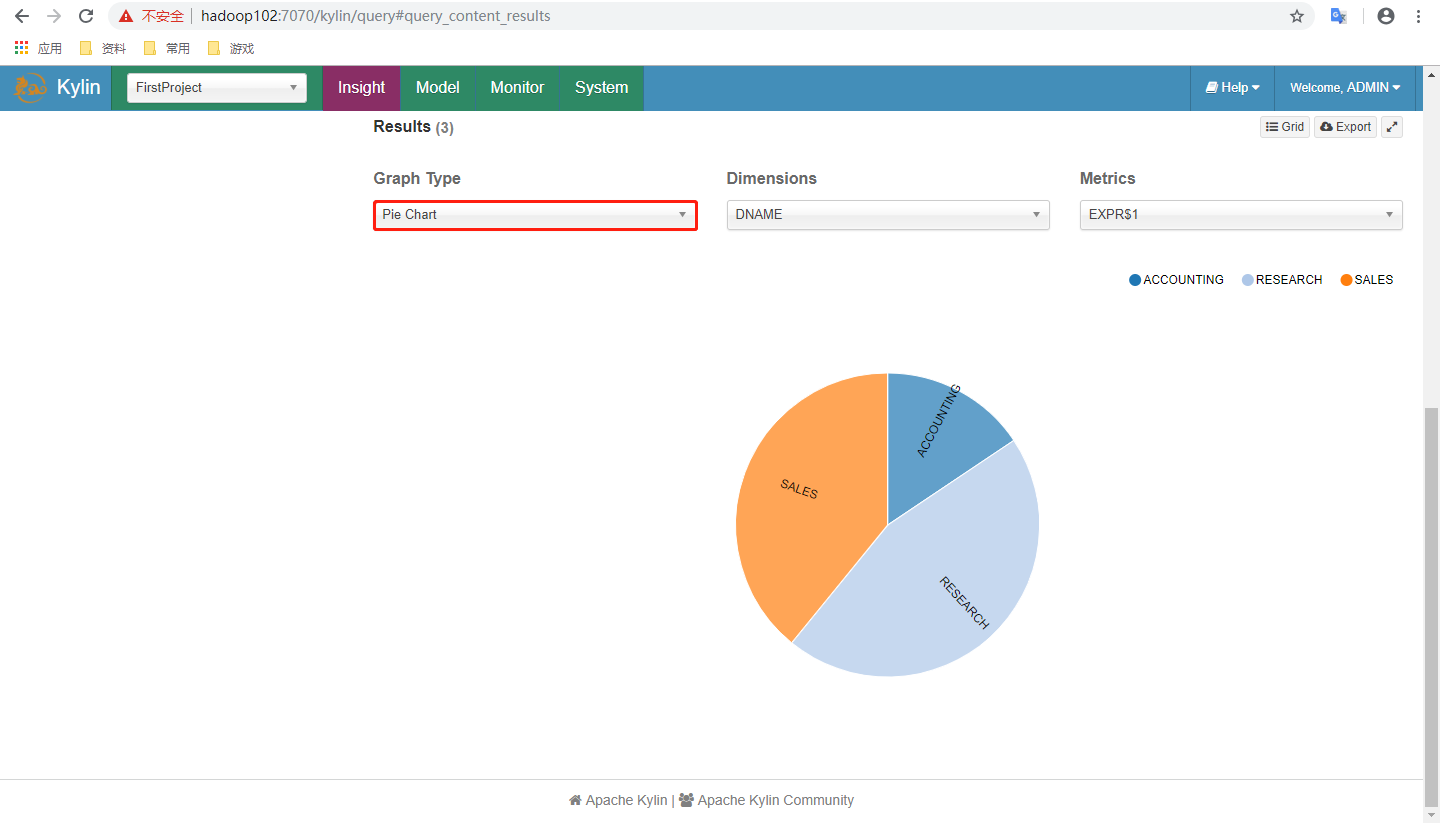
3）数据图表展示及导出



4）图表展示之条形图



4）图表展示之饼图



# 第4章 可视化

可以与Kylin结合使用的可视化工具很多，例如：

ODBC：与Tableau、Excel、PowerBI等工具集成

JDBC：与Saiku、BIRT等Java工具集成

RestAPI：与JavaScript、Web网页集成

Kylin开发团队还贡献了Zepplin的插件，也可以使用Zepplin来访问Kylin服务。

## 4.1 JDBC

1）新建项目并导入依赖

<dependencies>

<dependency>

<groupId>org.apache.kylin</groupId>

<artifactId>kylin-jdbc</artifactId>

<version>2.5.1</version>

</dependency>

</dependencies>

2）编码

package com.bawei;

import java.sql.\*;

public class TestKylin {

public static void main(String[] args) throws Exception {

//Kylin\_JDBC 驱动

String KYLIN\_DRIVER = "org.apache.kylin.jdbc.Driver";

//Kylin\_URL

String KYLIN\_URL = "jdbc:kylin://hadoop102:7070/FirstProject";

//Kylin的用户名

String KYLIN\_USER = "ADMIN";

//Kylin的密码

String KYLIN\_PASSWD = "KYLIN";

//添加驱动信息

Class.forName(KYLIN\_DRIVER);

//获取连接

Connection connection = DriverManager.getConnection(KYLIN\_URL, KYLIN\_USER, KYLIN\_PASSWD);

//预编译SQL

PreparedStatement ps = connection.prepareStatement("SELECT sum(sal) FROM emp group by deptno");

//执行查询

ResultSet resultSet = ps.executeQuery();

//遍历打印

while (resultSet.next()) {

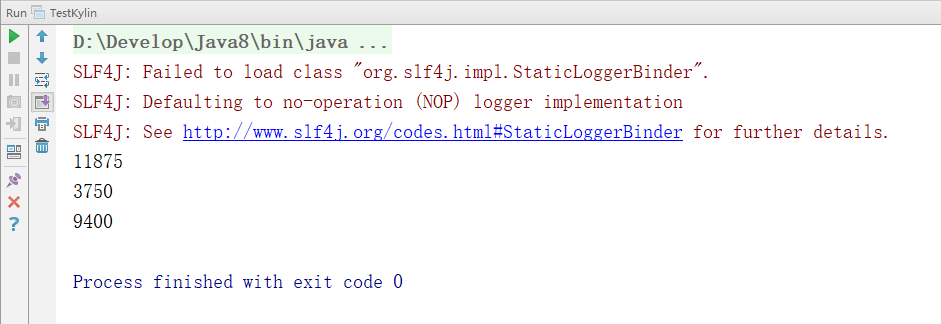
System.out.println(resultSet.getInt(1));

}

}

}

3）结果展示



## 4.2 Zeppelin

### 4.2.1 Zeppelin安装与启动

1）将zeppelin-0.8.0-bin-all.tgz上传至Linux

2）解压zeppelin-0.8.0-bin-all.tgz之/opt/module

[bawei@hadoop102 sorfware]$ tar -zxvf zeppelin-0.8.0-bin-all.tgz -C /opt/module/

3）修改名称

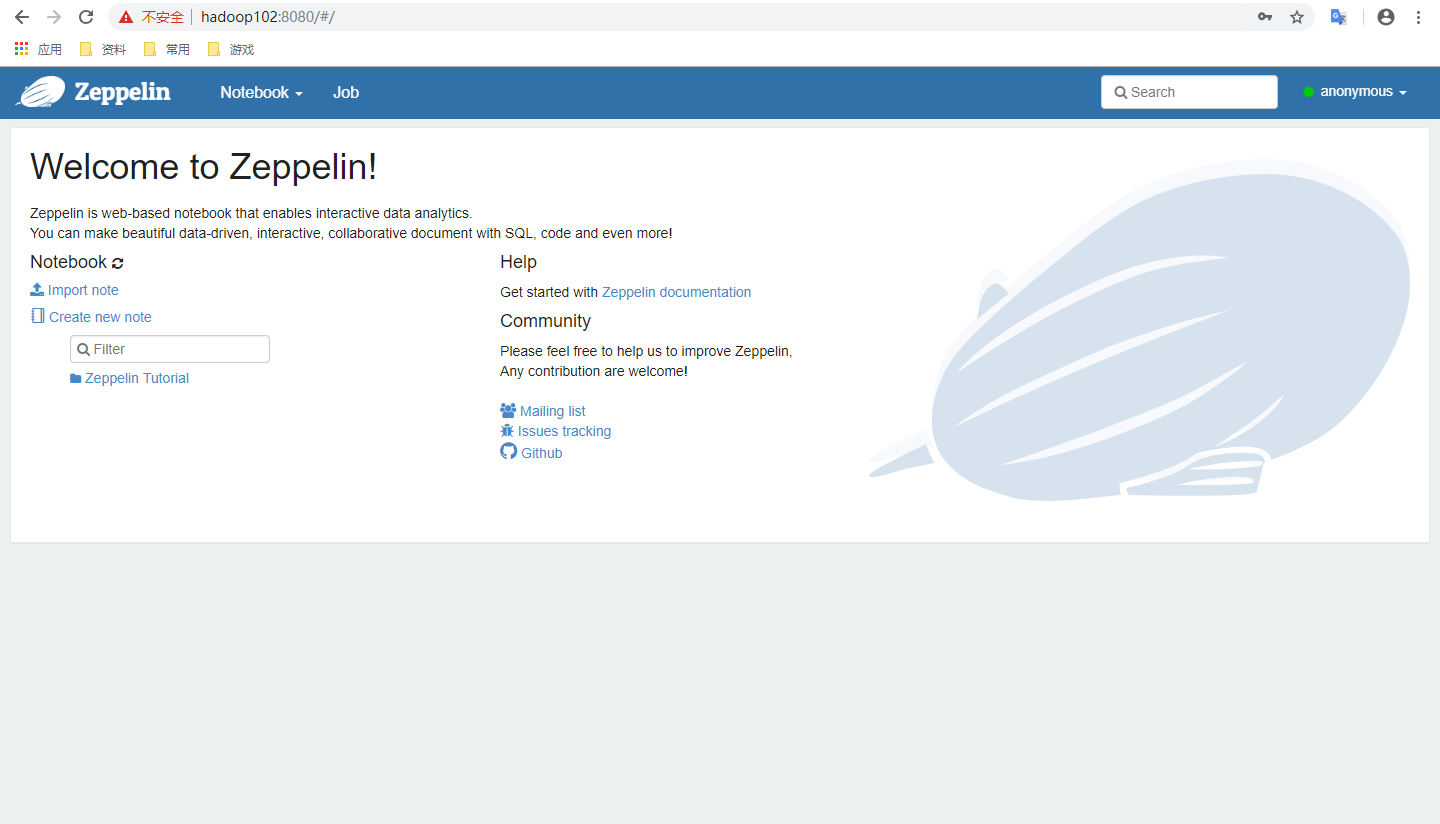
[bawei@hadoop102 module]$ mv zeppelin-0.8.0-bin-all/ zeppelin

4）启动

[bawei@hadoop102 zeppelin]$ bin/zeppelin-daemon.sh start

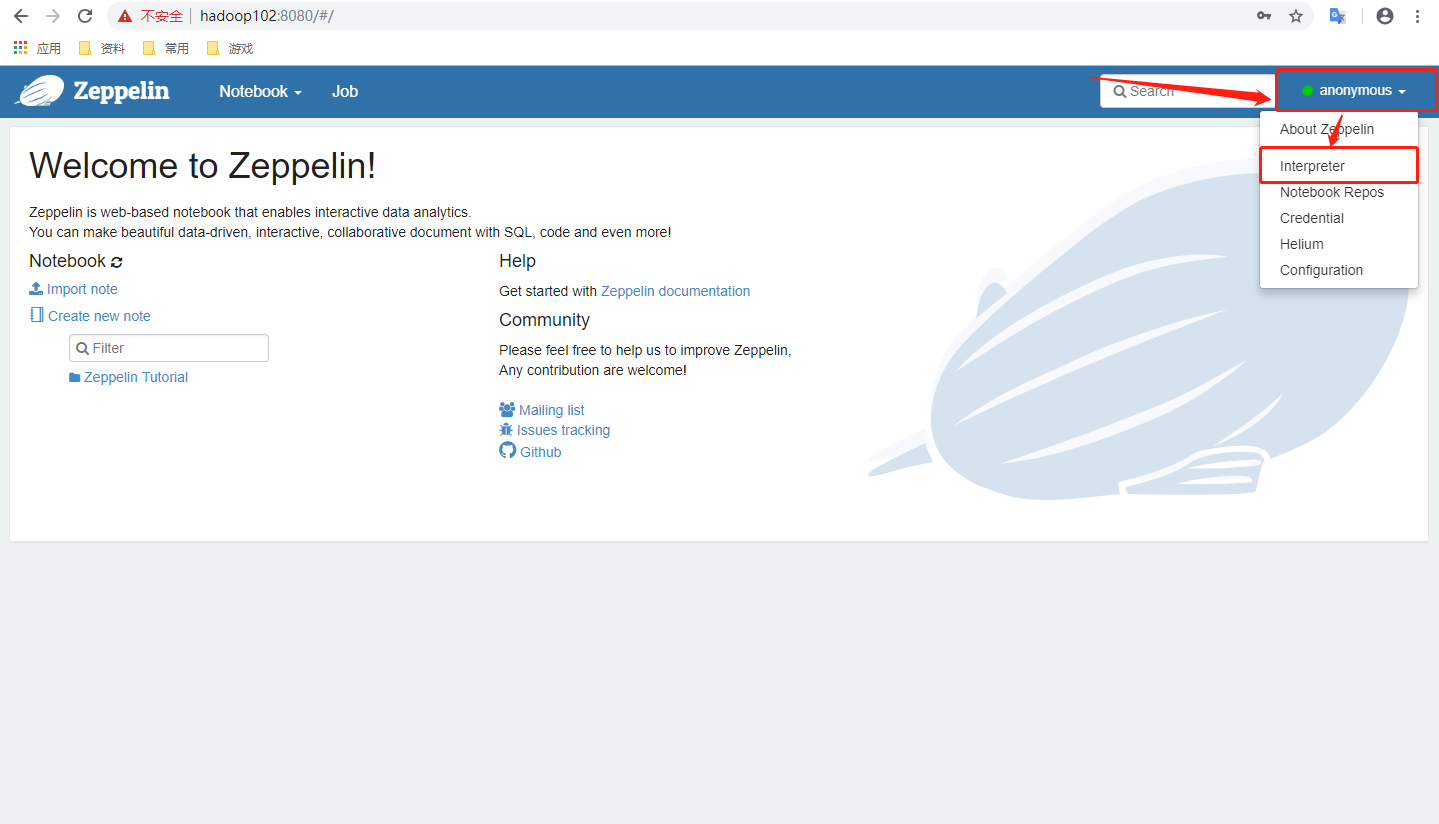
可登录网页查看，web默认端口号为8080

http://hadoop102:8080

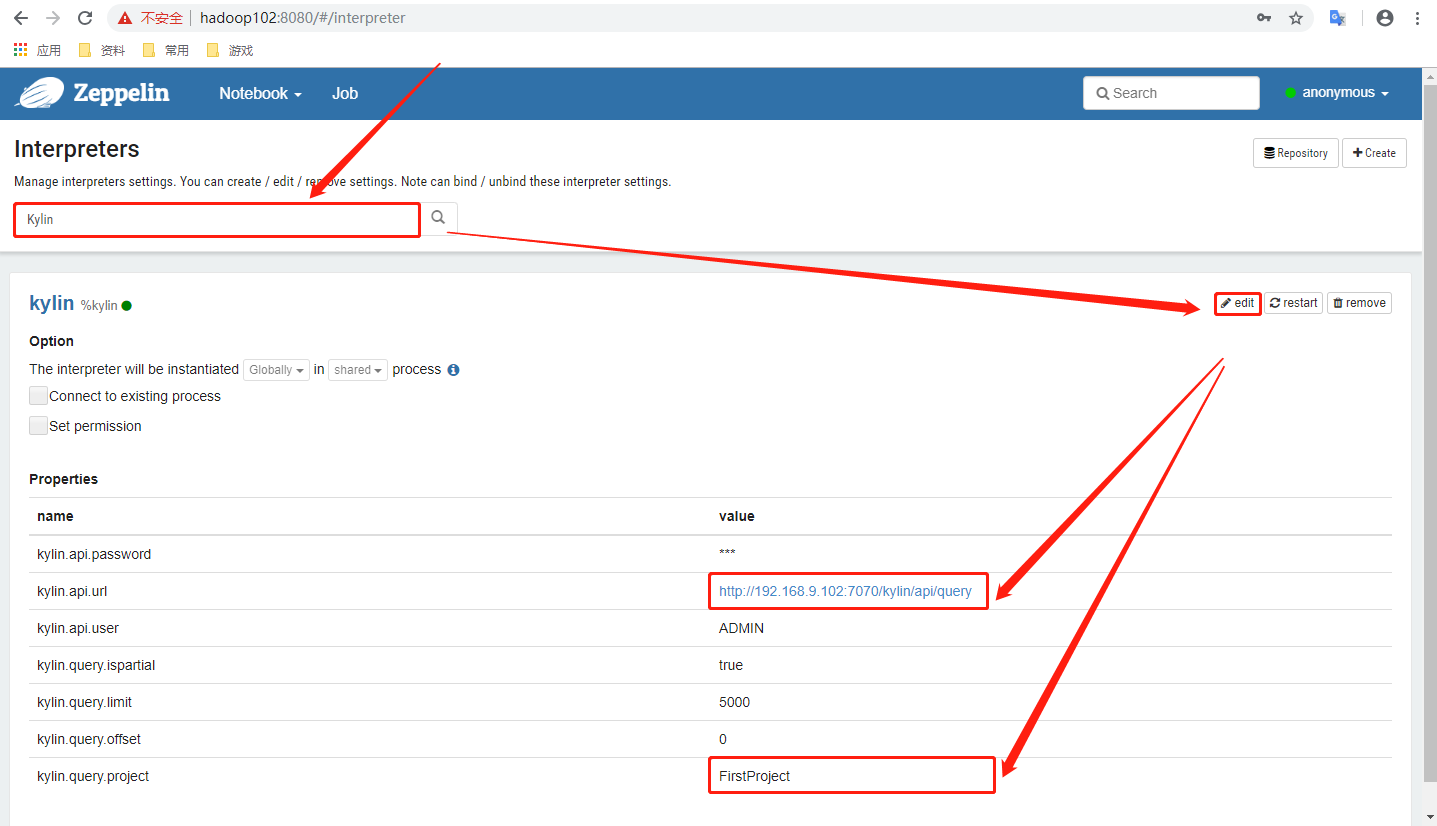


### 4.2.2 配置Zepplin支持Kylin

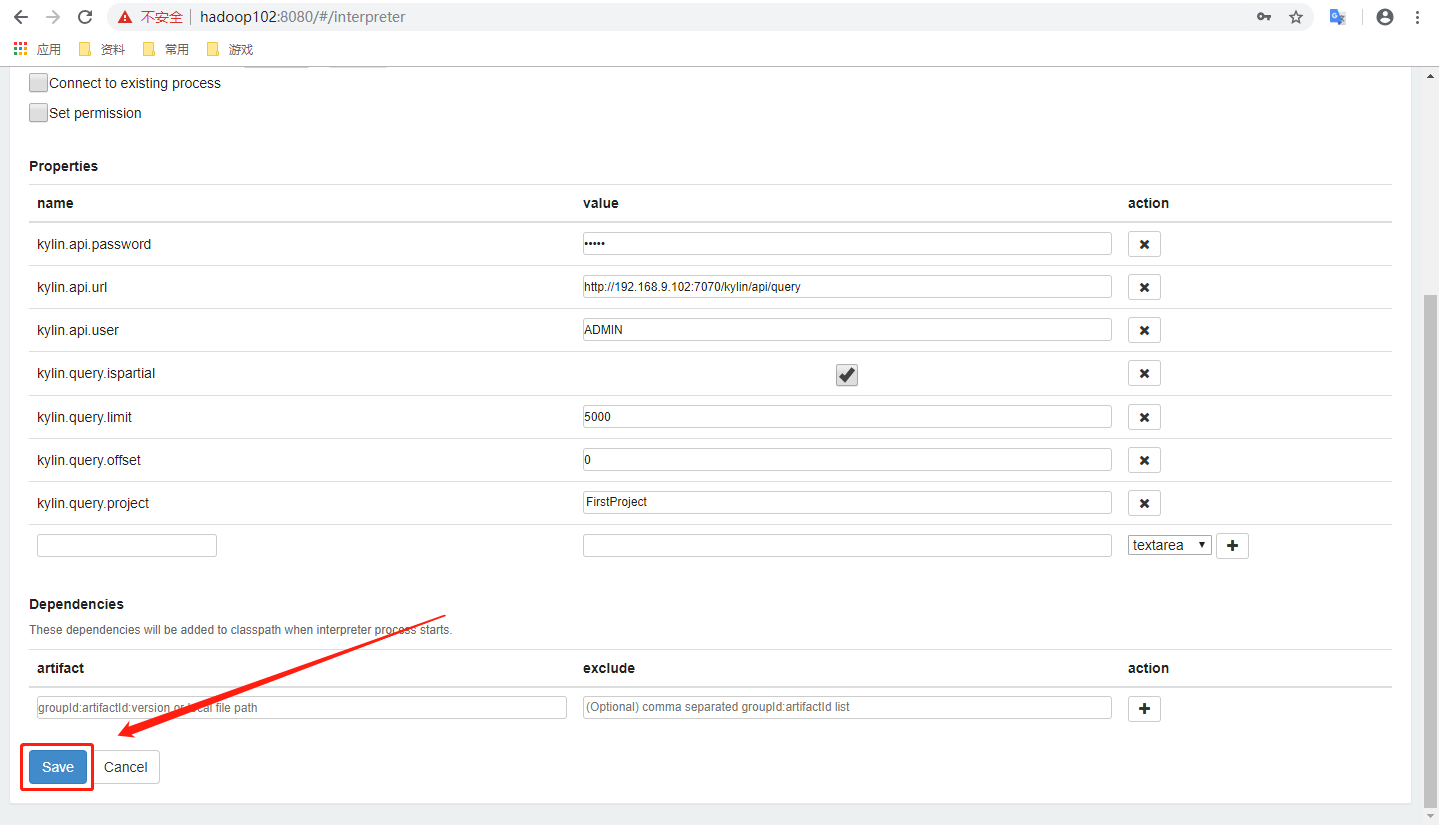
1）点击右上角anonymous选择Interpreter



2）搜索Kylin插件并修改相应的配置



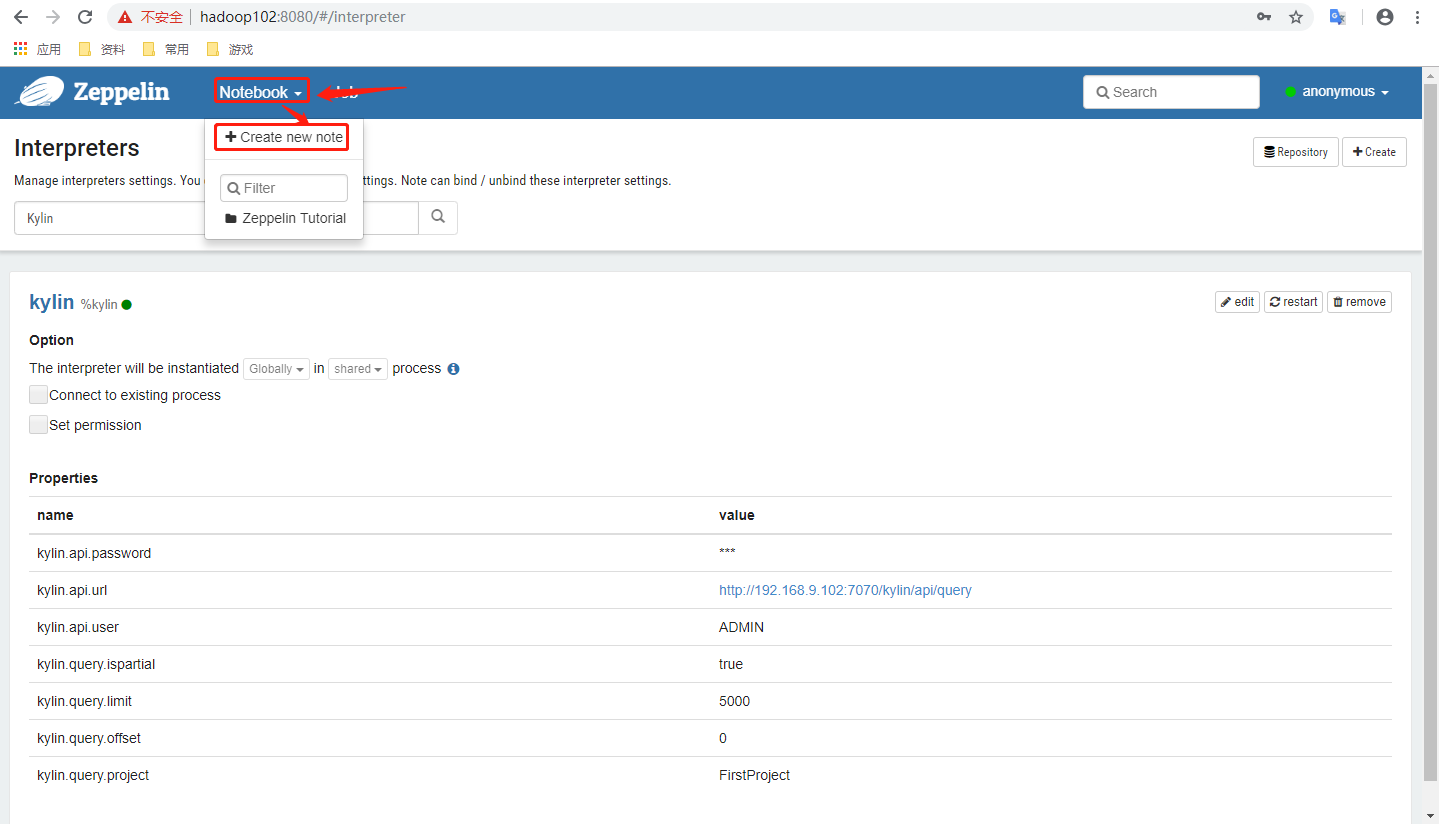
3）修改完成点击Save完成



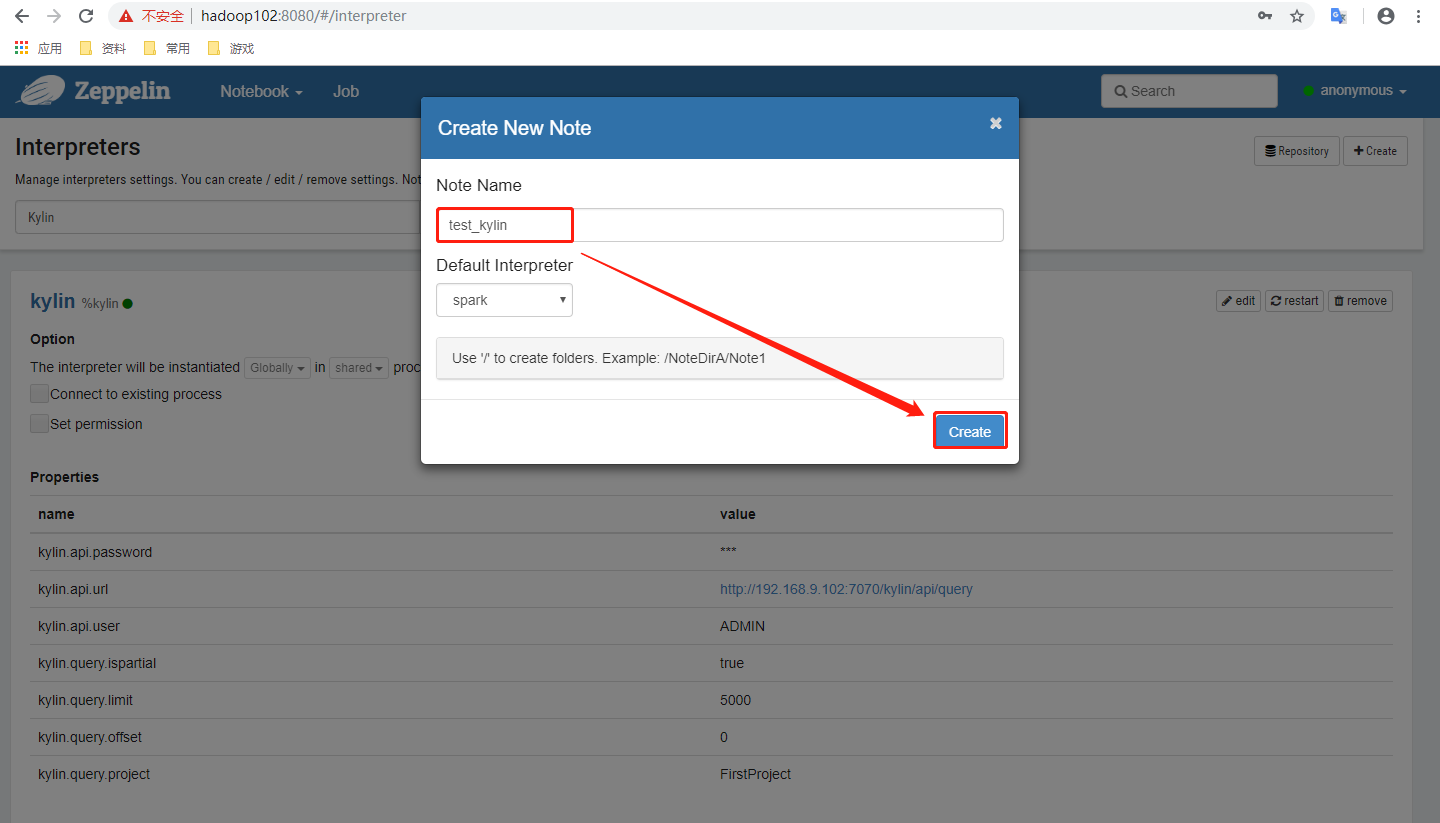
### 4.3.3 案例实操

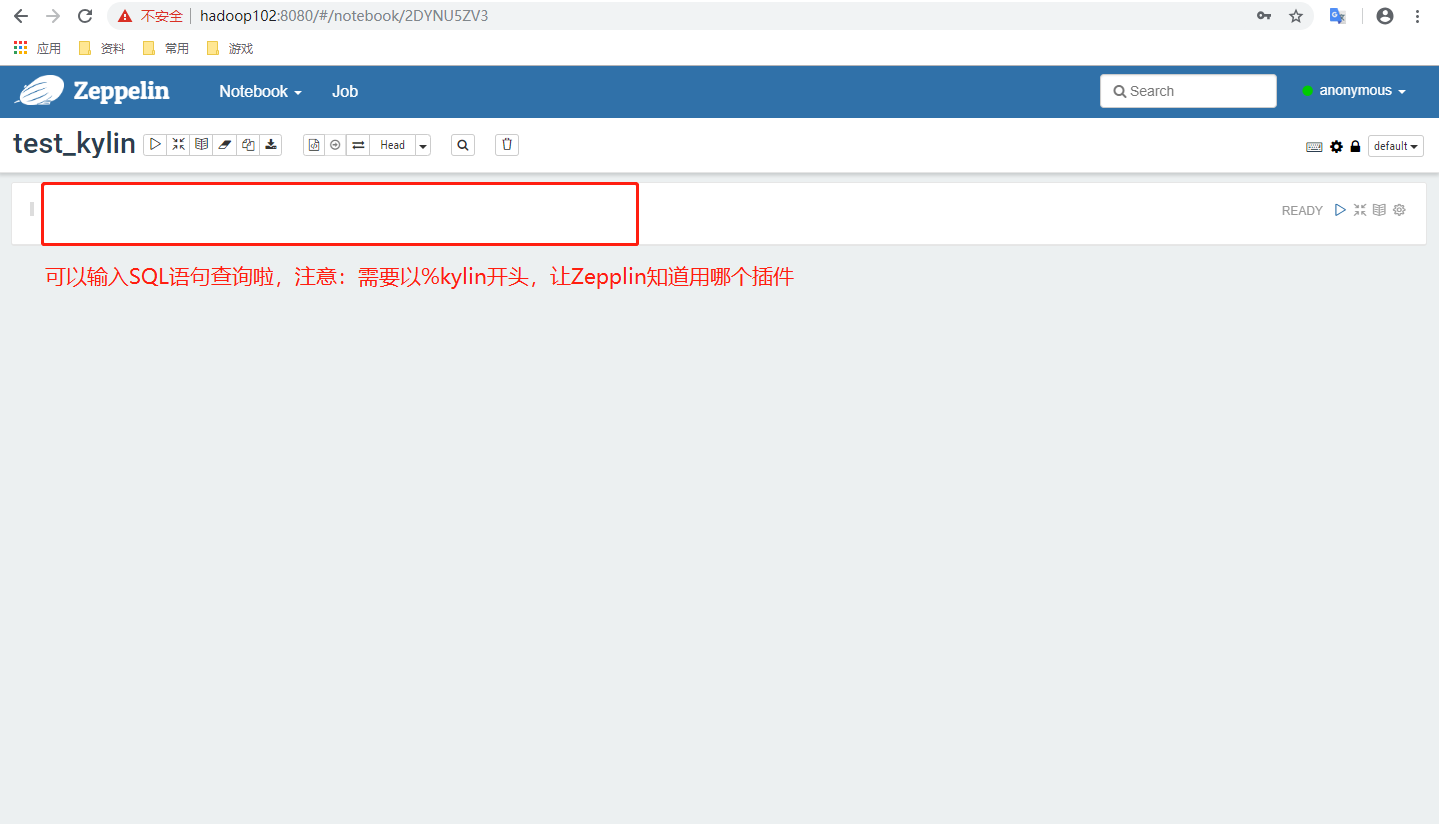
需求：查询员工详细信息，并使用各种图表进行展示

1）点击Notebook创建新的note

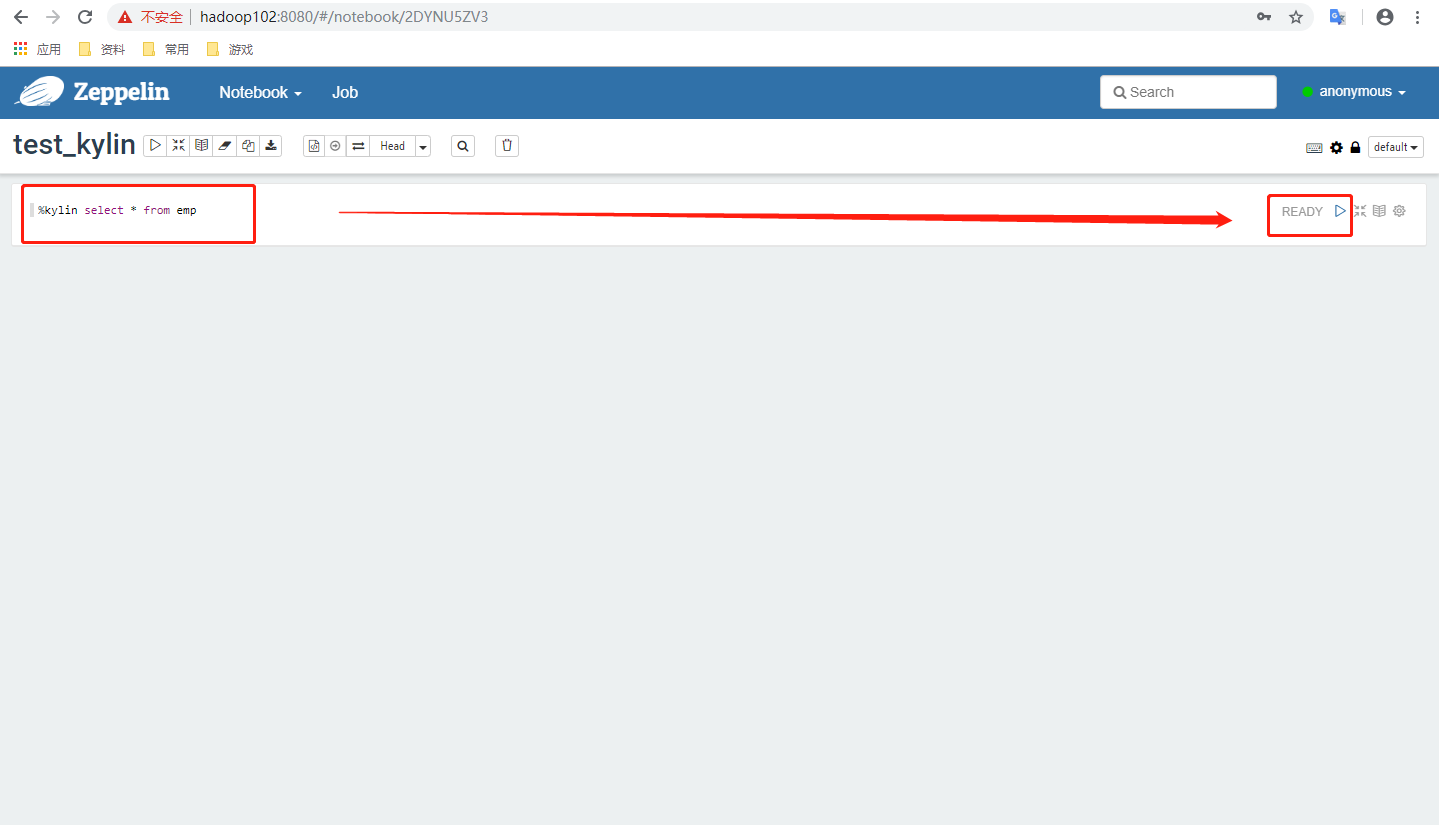


2）填写Note Name点击Create

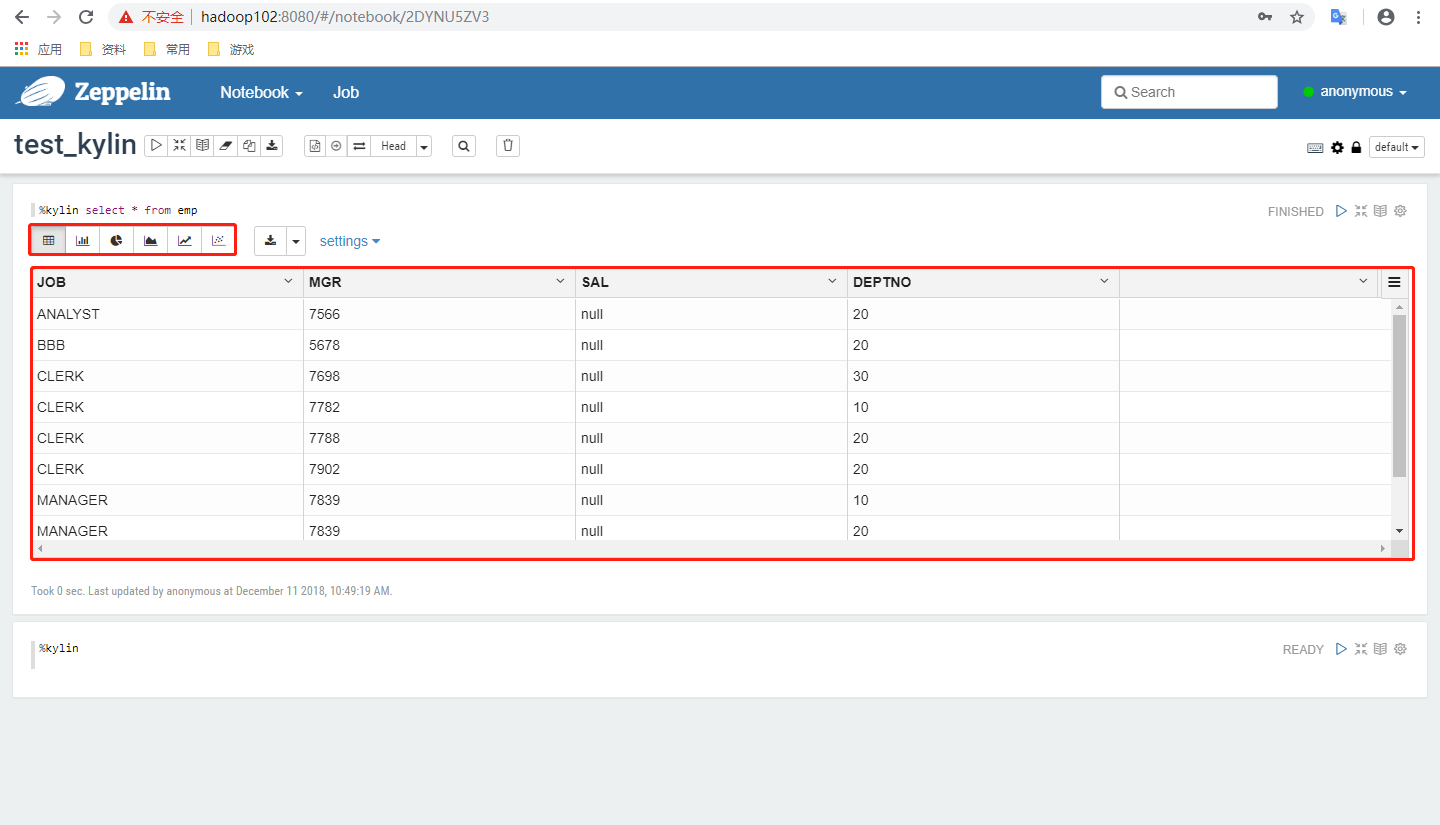




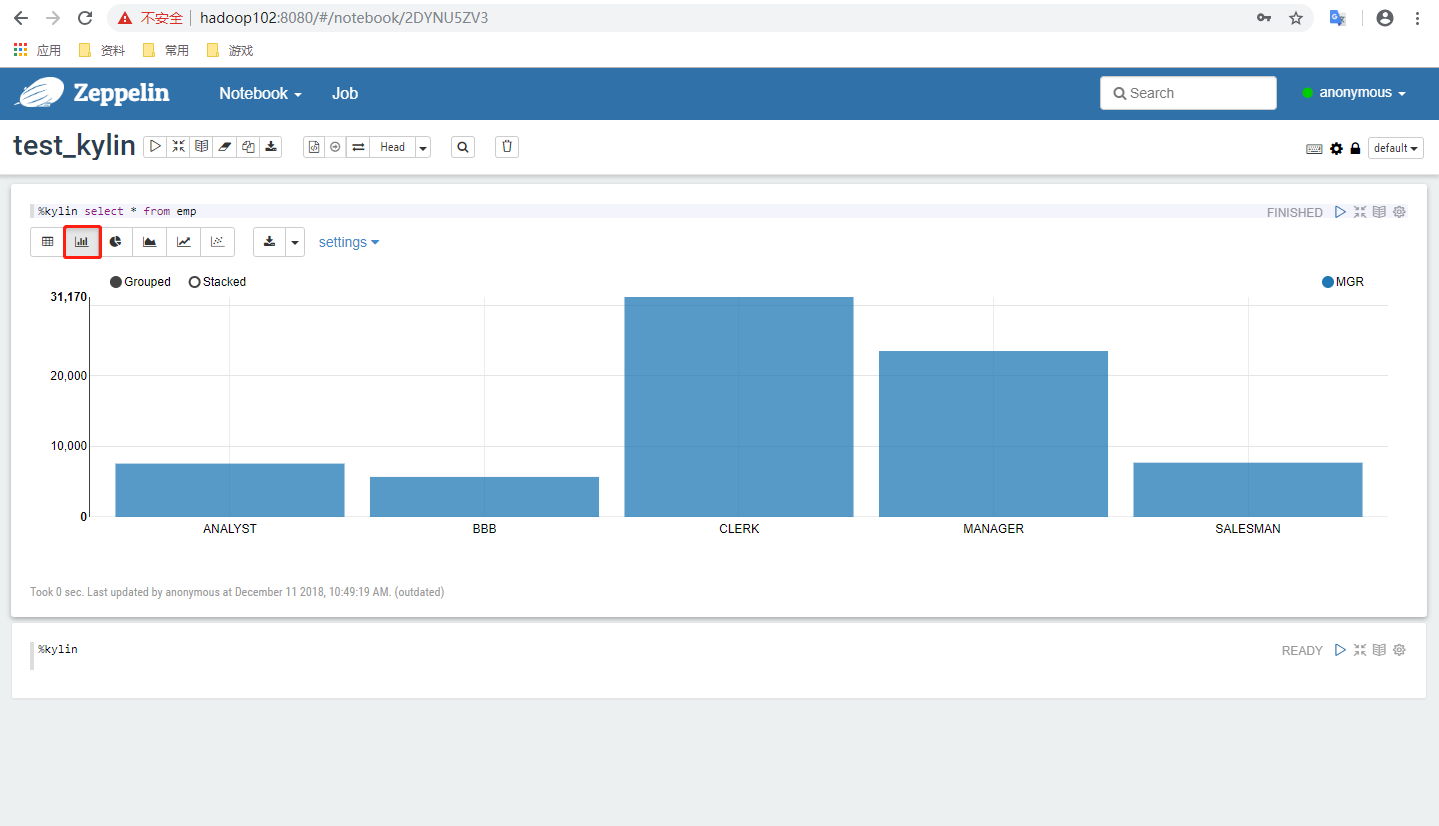
3）执行查询

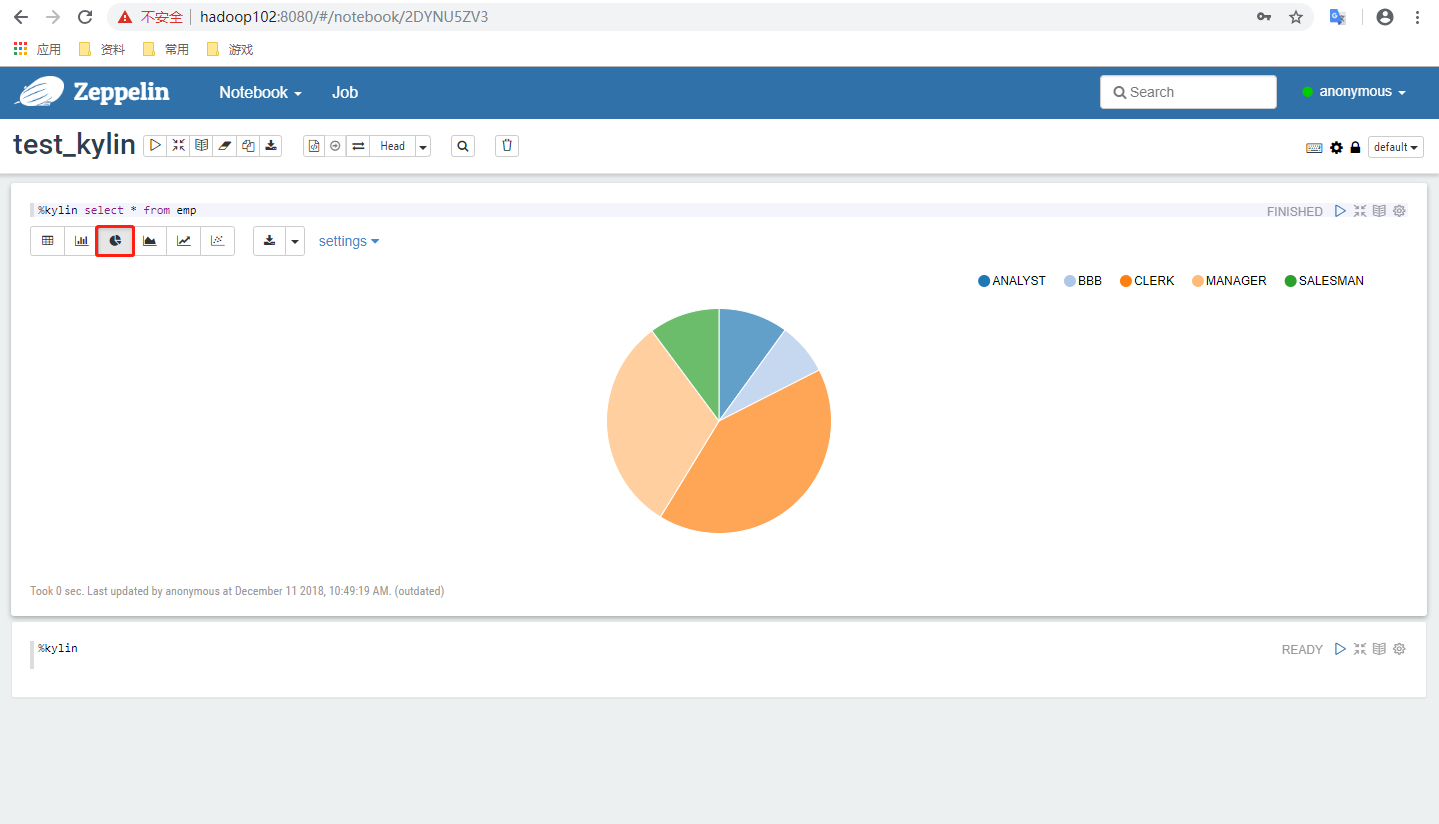


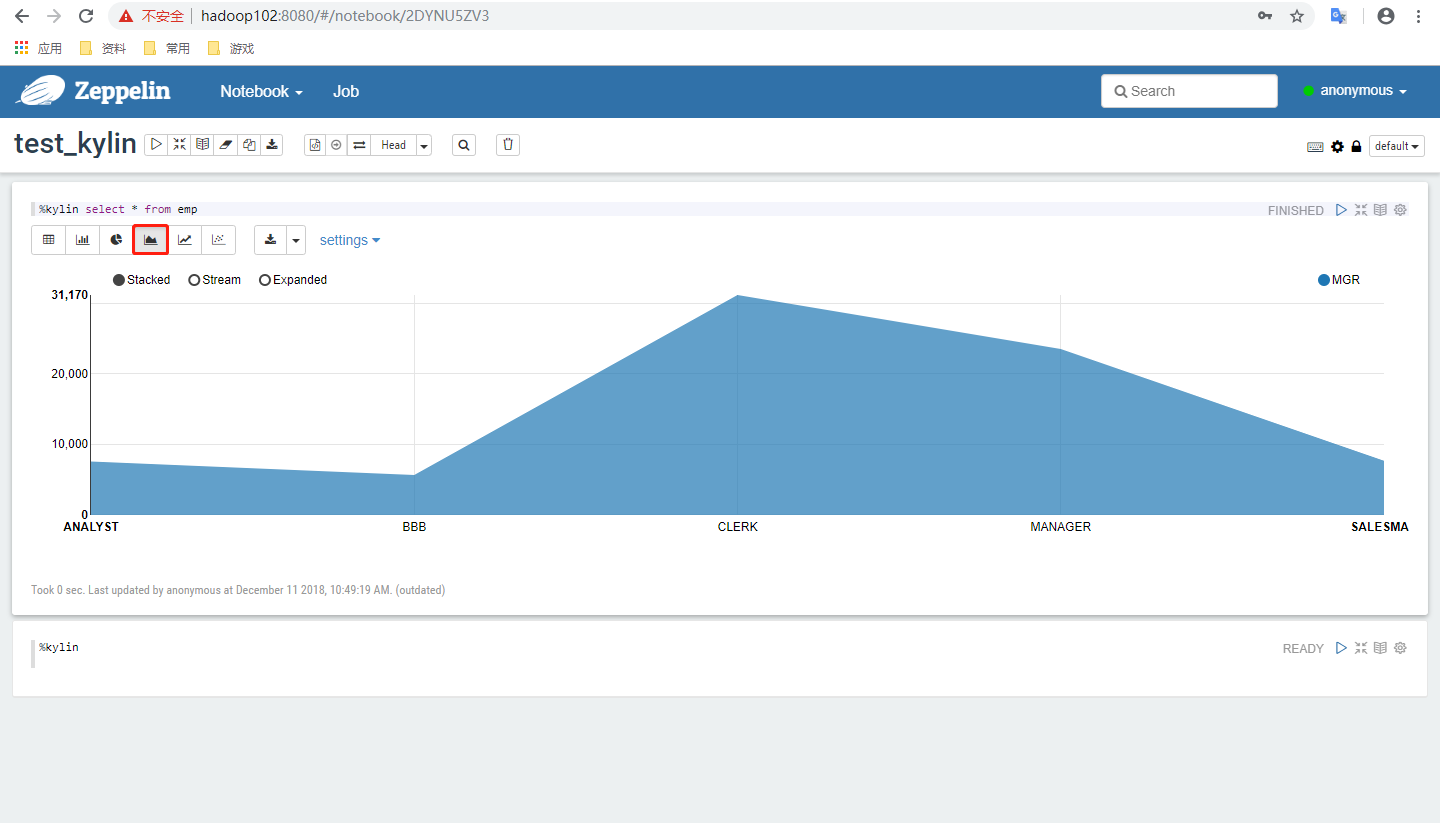
4）结果展示

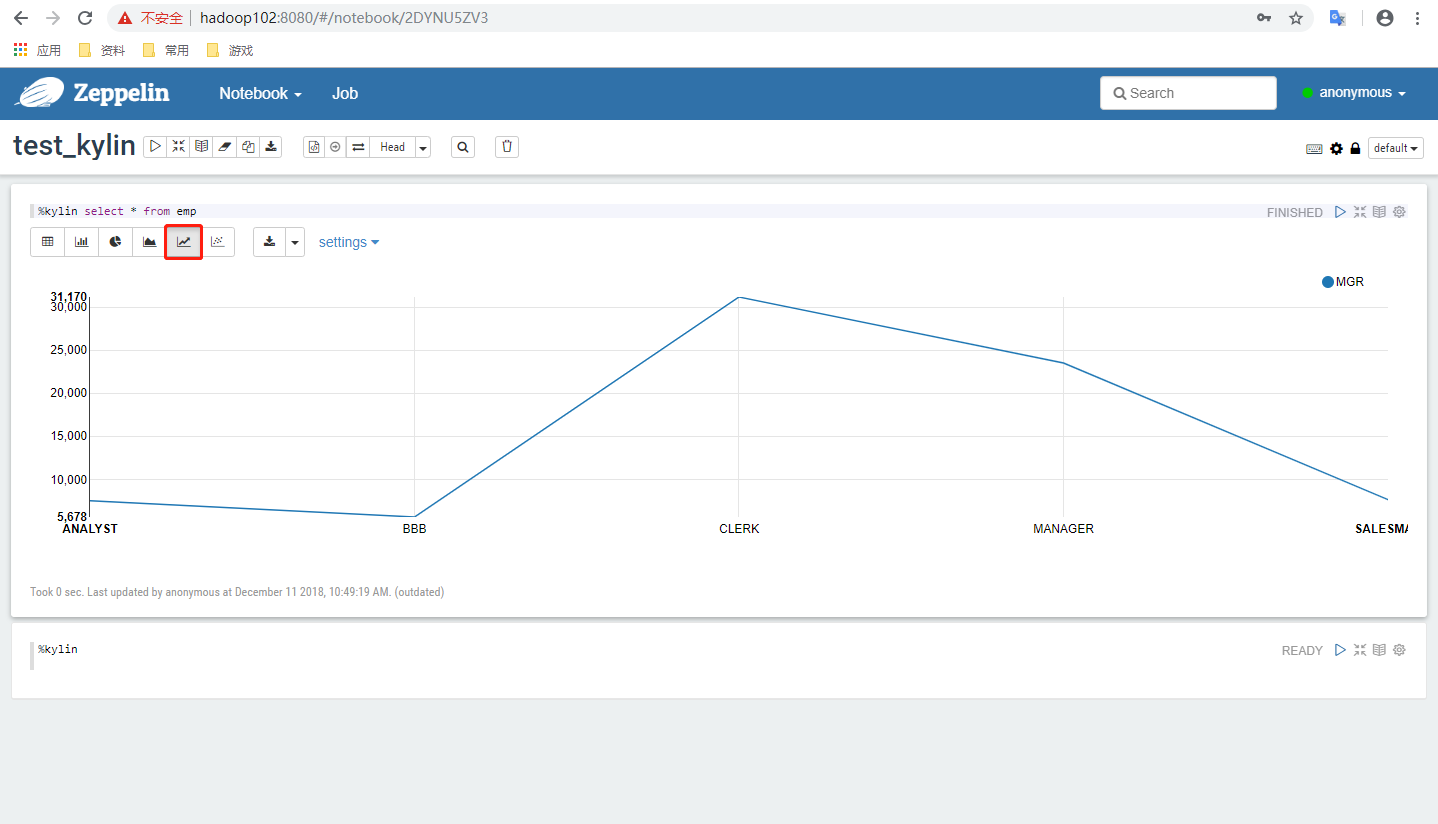


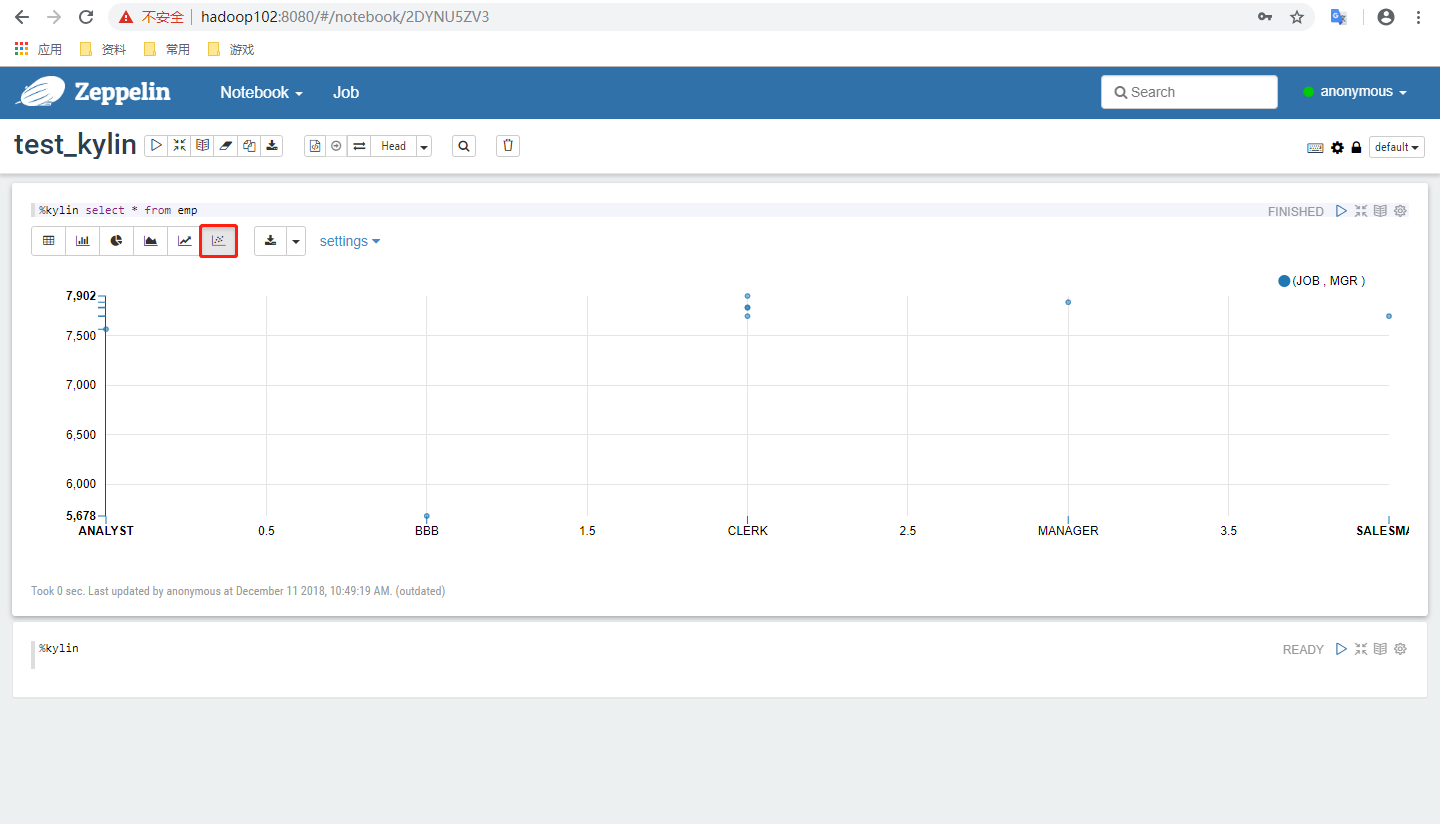
5）其他图表格式











# 第5章 Cube构建优化

从之前章节的介绍可以知道，在没有采取任何优化措施的情况下，Kylin会对每一种维度的组合进行预计算，每种维度的组合的预计算结果被称为Cuboid。假设有4个维度，我们最终会有24 =16个Cuboid需要计算。

但在现实情况中，用户的维度数量一般远远大于4个。假设用户有10 个维度，那么没有经过任何优化的Cube就会存在210 =1024个Cuboid；而如果用户有20个维度，那么Cube中总共会存在220 =1048576个Cuboid。虽然每个Cuboid的大小存在很大的差异，但是单单想到Cuboid的数量就足以让人想象到这样的Cube对构建引擎、存储引擎来说压力有多么巨大。因此，在构建维度数量较多的Cube时，尤其要注意Cube的剪枝优化（即减少Cuboid的生成）。

## 5.1 找到问题Cube

### 5.1.1 检查Cuboid数量

Apache Kylin提供了一个简单的工具，供用户检查Cube中哪些Cuboid 最终被预计算了，我们称其为被物化（Materialized）的Cuboid。同时，这种方法还能给出每个Cuboid所占空间的估计值。由于该工具需要在对数据进行一定阶段的处理之后才能估算Cuboid的大小，因此一般来说只能在Cube构建完毕之后再使用该工具。目前关于这一点也是该工具的一大不足，由于同一个Cube的不同Segment之间仅是输入数据不同，模型信息和优化策略都是共享的，所以不同Segment中哪些Cuboid被物化哪些没有被物化都是一样的。因此只要Cube中至少有一个Segment，那么就能使用如下的命令行工具去检查这个Cube中的Cuboid状态：

bin/kylin.sh org.apache.kylin.engine.mr.common.CubeStatsReader CUBE\_NAME

CUBE\_NAME：想要查看的Cube的名字

例如：

[bawei@hadoop102 kylin]$ bin/kylin.sh org.apache.kylin.engine.mr.common.CubeStatsReader FirstCube

… …

… …

Statistics of FirstCube[FULL\_BUILD]

Cube statistics hll precision: 14

Total cuboids: 7

Total estimated rows: 51

Total estimated size(MB): 3.027915954589844E-4

Sampling percentage: 100

Mapper overlap ratio: 1.0

Mapper number: 1

Length of dimension DEFAULT.EMP.JOB is 1

Length of dimension DEFAULT.EMP.MGR is 1

Length of dimension DEFAULT.EMP.DEPTNO is 1

|---- Cuboid 111, est row: 10, est MB: 0

|---- Cuboid 011, est row: 9, est MB: 0, shrink: 90%

|---- Cuboid 001, est row: 3, est MB: 0, shrink: 33.33%

|---- Cuboid 010, est row: 7, est MB: 0, shrink: 77.78%

|---- Cuboid 101, est row: 9, est MB: 0, shrink: 90%

|---- Cuboid 100, est row: 5, est MB: 0, shrink: 55.56%

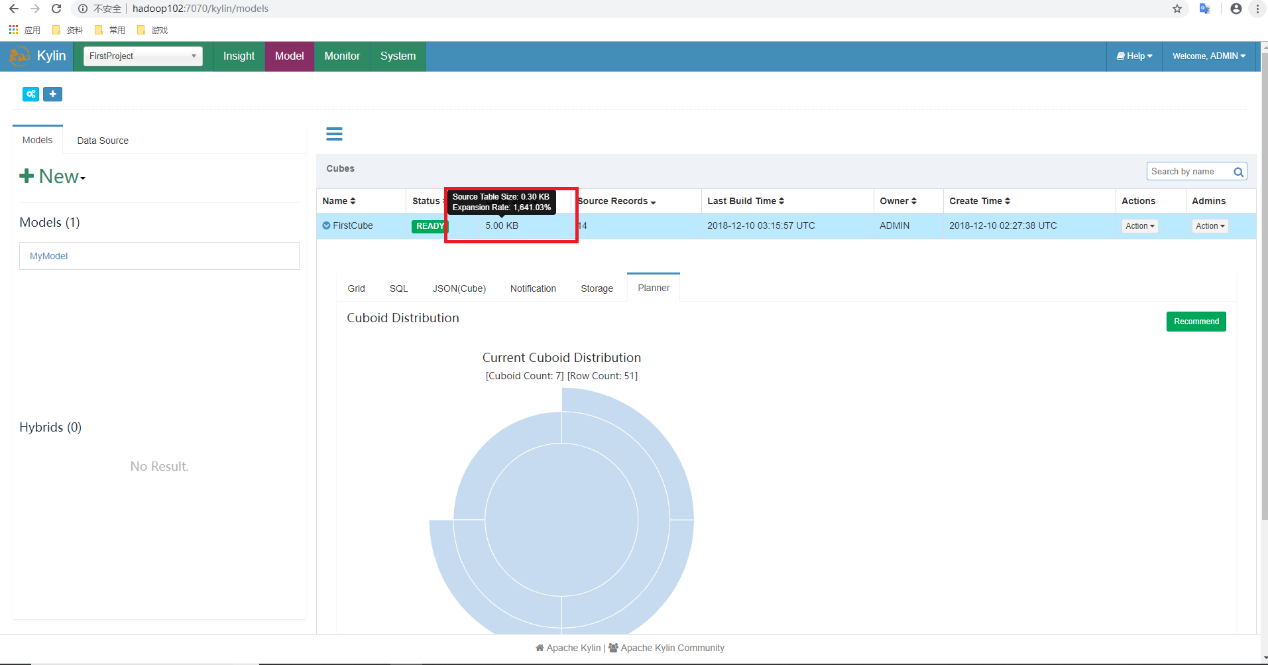
|---- Cuboid 110, est row: 8, est MB: 0, shrink: 80%

从分析结果的下半部分可以看到，所有的Cuboid及它的分析结果都以树状的形式打印了出来。在这棵树中，每个节点代表一个Cuboid，每个Cuboid都由一连串1或0的数字组成，如果数字为0，则代表这个Cuboid中不存在相应的维度；如果数字为1，则代表这个Cuboid中存在相应的维度。除了最顶端的Cuboid之外，每个Cuboid都有一个父亲Cuboid，且都比父亲Cuboid少了一个“1”。其意义是这个Cuboid就是由它的父亲节点减少一个维度聚合而来的（上卷）。最顶端的Cuboid称为Base Cuboid，它直接由源数据计算而来。

每行Cuboid的输出中除了0和1的数字串以外，后面还有每个Cuboid 的的行数与父亲节点的对比（Shrink值）。所有Cuboid行数的估计值之和应该等于Segment的行数估计值，每个Cuboid都是在它的父亲节点的基础上进一步聚合而成的，因此从理论上说每个Cuboid无论是行数还是大小都应该小于它的父亲。在这棵树中，我们可以观察每个节点的Shrink值，如果该值接近100%，则说明这个Cuboid虽然比它的父亲Cuboid少了一个维度，但是并没有比它的父亲Cuboid少很多行数据。换而言之，即使没有这个Cuboid， 我们在查询时使用它的父亲Cuboid，也不会有太大的代价。那么我们就可以对这个Cuboid进行剪枝操作。

### 5.1.2 检查Cube大小

还有一种更为简单的方法可以帮助我们判断Cube是否已经足够优化。在Web GUI的Model页面选择一个READY状态的Cube，当我们把光标移到该Cube的Cube Size列时，Web GUI会提示Cube的源数据大小，以及当前Cube的大小除以源数据大小的比例，称为膨胀率（Expansion Rate），如图所示。



一般来说，Cube的膨胀率应该在0%~1000%之间，如果一个Cube的膨胀率超过1000%，那么Cube管理员应当开始挖掘其中的原因。通常，膨胀率高有以下几个方面的原因。

1）Cube中的维度数量较多，且没有进行很好的Cuboid剪枝优化，导致Cuboid数量极多;

2）Cube中存在较高基数的维度，导致包含这类维度的每一个Cuboid占用的空间都很大，这些Cuboid累积造成整体Cube体积变大;

因此，对于Cube膨胀率居高不下的情况，管理员需要结合实际数据进行分析，可灵活地运用接下来介绍的优化方法对Cube进行优化。

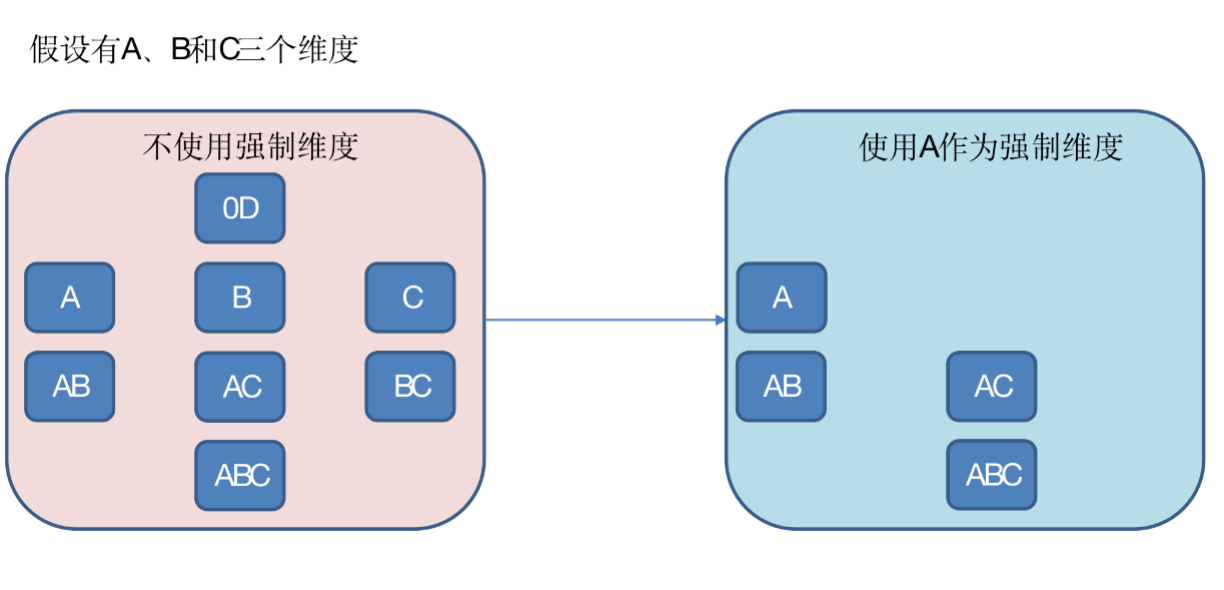
## 5.2 优化构建

### 5.2.1 使用聚合组

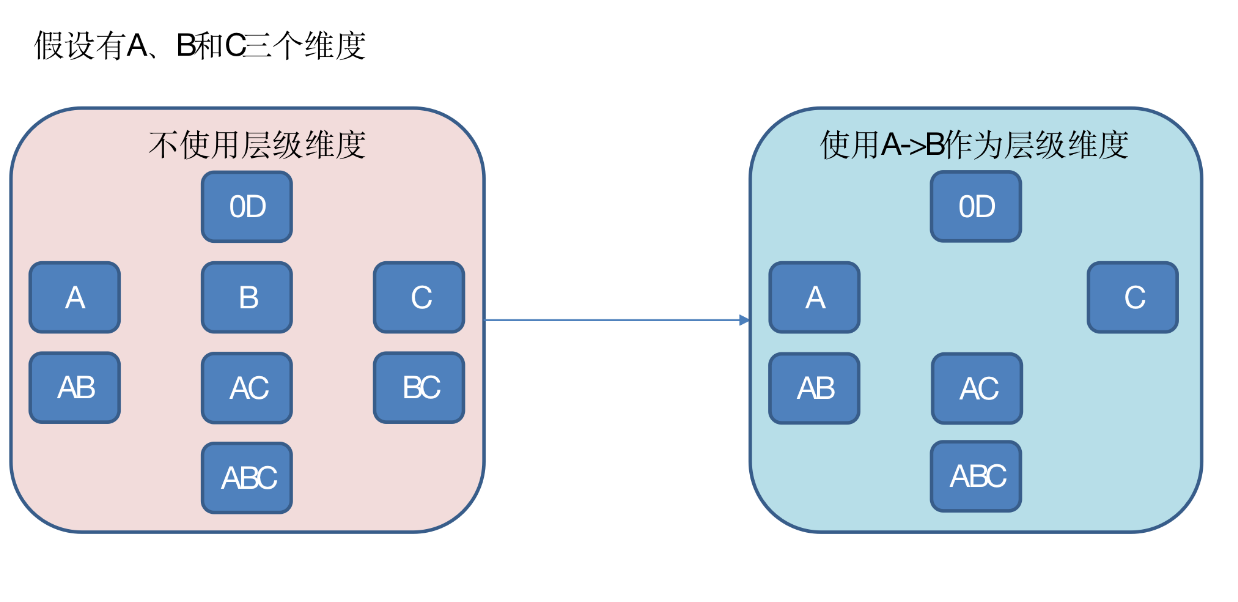
聚合组（Aggregation Group）是一种强大的剪枝工具。聚合组假设一个Cube的所有维度均可以根据业务需求划分成若干组（当然也可以是一个组），由于同一个组内的维度更可能同时被同一个查询用到，因此会表现出更加紧密的内在关联。每个分组的维度集合均是Cube所有维度的一个子集，不同的分组各自拥有一套维度集合，它们可能与其他分组有相同的维度，也可能没有相同的维度。每个分组各自独立地根据自身的规则贡献出一批需要被物化的Cuboid，所有分组贡献的Cuboid的并集就成为了当前Cube中所有需要物化的Cuboid的集合。不同的分组有可能会贡献出相同的Cuboid，构建引擎会察觉到这点，并且保证每一个Cuboid无论在多少个分组中出现，它都只会被物化一次。

对于每个分组内部的维度，用户可以使用如下三种可选的方式定义，它们之间的关系，具体如下。

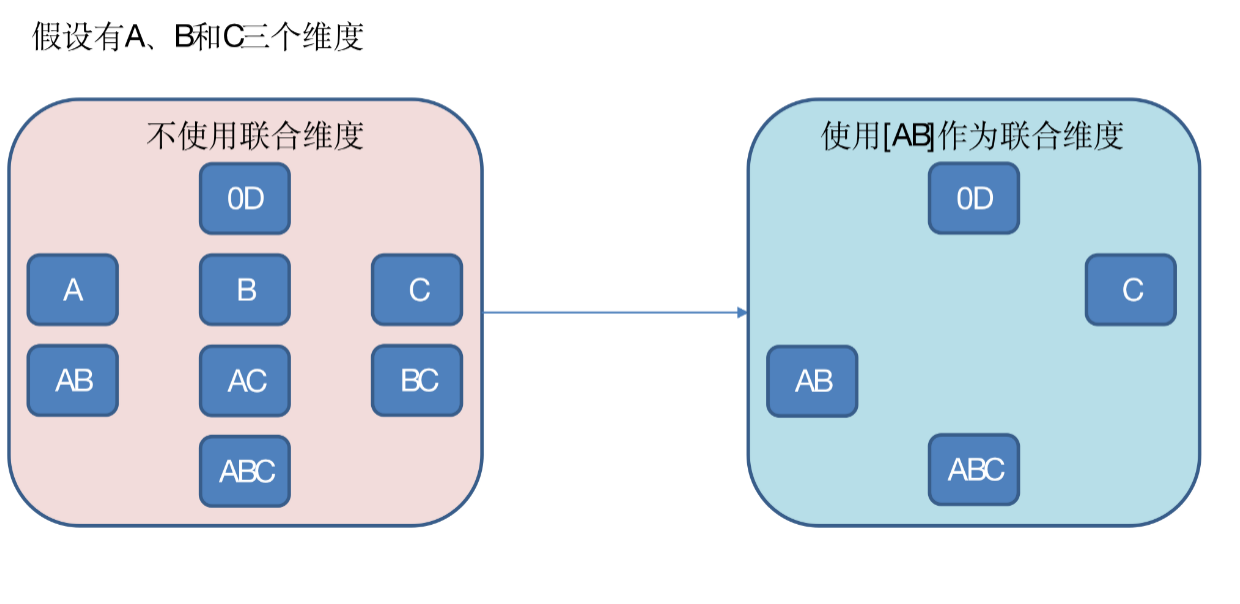
1）强制维度（Mandatory），如果一个维度被定义为强制维度，那么这个分组产生的所有Cuboid中每一个Cuboid都会包含该维度。每个分组中都可以有0个、1个或多个强制维度。如果根据这个分组的业务逻辑，则相关的查询一定会在过滤条件或分组条件中，因此可以在该分组中把该维度设置为强制维度。



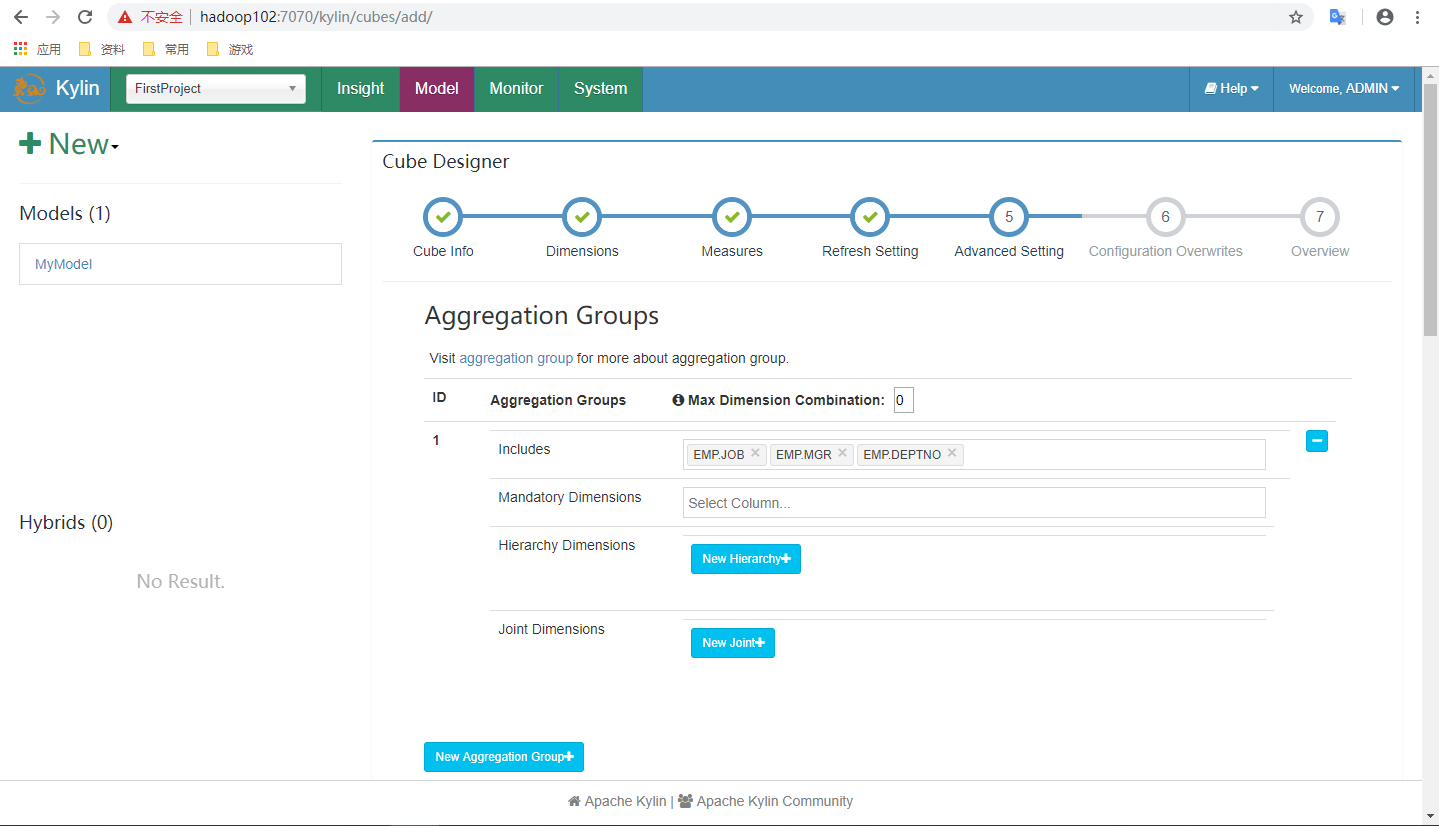
2）层级维度（Hierarchy），每个层级包含两个或更多个维度。假设一个层级中包含D1，D2…Dn这n个维度，那么在该分组产生的任何Cuboid中， 这n个维度只会以（），（D1），（D1，D2）…（D1，D2…Dn）这n+1种形式中的一种出现。每个分组中可以有0个、1个或多个层级，不同的层级之间不应当有共享的维度。如果根据这个分组的业务逻辑，则多个维度直接存在层级关系，因此可以在该分组中把这些维度设置为层级维度。



3）联合维度（Joint），每个联合中包含两个或更多个维度，如果某些列形成一个联合，那么在该分组产生的任何Cuboid中，这些联合维度要么一起出现，要么都不出现。每个分组中可以有0个或多个联合，但是不同的联合之间不应当有共享的维度（否则它们可以合并成一个联合）。如果根据这个分组的业务逻辑，多个维度在查询中总是同时出现，则可以在该分组中把这些维度设置为联合维度。



这些操作可以在Cube Designer的Advanced Setting中的Aggregation Groups区域完成，如下图所示。



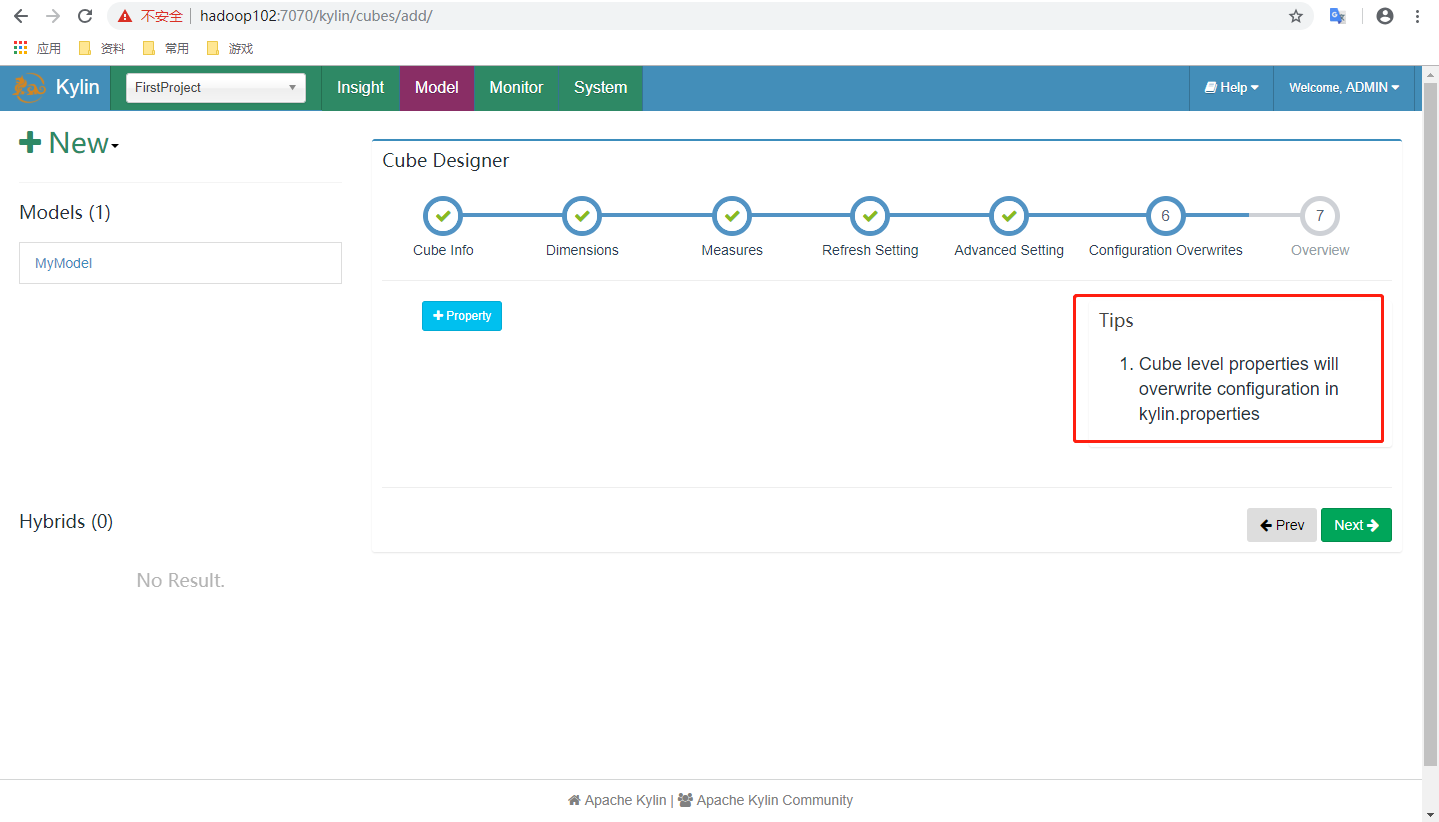
聚合组的设计非常灵活，甚至可以用来描述一些极端的设计。假设我们的业务需求非常单一，只需要某些特定的Cuboid，那么可以创建多个聚合组，每个聚合组代表一个Cuboid。具体的方法是在聚合组中先包含某个Cuboid所需的所有维度，然后把这些维度都设置为强制维度。这样当前的聚合组就只能产生我们想要的那一个Cuboid了。

再比如，有的时候我们的Cube中有一些基数非常大的维度，如果不做特殊处理，它就会和其他的维度进行各种组合，从而产生一大堆包含它的Cuboid。包含高基数维度的Cuboid在行数和体积上往往非常庞大，这会导致整个Cube的膨胀率变大。如果根据业务需求知道这个高基数的维度只会与若干个维度（而不是所有维度）同时被查询到，那么就可以通过聚合组对这个高基数维度做一定的“隔离”。我们把这个高基数的维度放入一个单独的聚合组，再把所有可能会与这个高基数维度一起被查询到的其他维度也放进来。这样，这个高基数的维度就被“隔离”在一个聚合组中了，所有不会与它一起被查询到的维度都没有和它一起出现在任何一个分组中，因此也就不会有多余的Cuboid产生。这点也大大减少了包含该高基数维度的Cuboid的数量，可以有效地控制Cube的膨胀率。

### 5.2.2 并发粒度优化

当Segment中某一个Cuboid的大小超出一定的阈值时，系统会将该Cuboid的数据分片到多个分区中，以实现Cuboid数据读取的并行化，从而优化Cube的查询速度。具体的实现方式如下：

构建引擎根据Segment估计的大小，以及参数“kylin.hbase.region.cut”的设置决定Segment在存储引擎中总共需要几个分区来存储，如果存储引擎是HBase，那么分区的数量就对应于HBase中的Region数量。kylin.hbase.region.cut的默认值是5.0，单位是GB，也就是说对于一个大小估计是50GB的Segment，构建引擎会给它分配10个分区。用户还可以通过设置kylin.hbase.region.count.min（默认为1）和kylin.hbase.region.count.max（默认为500）两个配置来决定每个Segment最少或最多被划分成多少个分区。



由于每个Cube的并发粒度控制不尽相同，因此建议在Cube Designer 的Configuration Overwrites（上图所示）中为每个Cube量身定制控制并发粒度的参数。

假设将把当前Cube的kylin.hbase.region.count.min设置为2，kylin.hbase.region.count.max设置为100。这样无论Segment的大小如何变化，它的分区数量最小都不会低于2，最大都不会超过100。相应地，这个Segment背后的存储引擎（HBase）为了存储这个Segment，也不会使用小于两个或超过100个的分区。我们还调整了默认的kylin.hbase.region.cut，这样50GB的Segment基本上会被分配到50个分区，相比默认设置，我们的Cuboid可能最多会获得5倍的并发量。