尚硅谷大数据项目之电商数仓（即席查询）

(作者：尚硅谷大数据研发部)

版本：V4.0

第 1 章 Presto

1.1 Presto 简介

1.1.1 Presto 概念

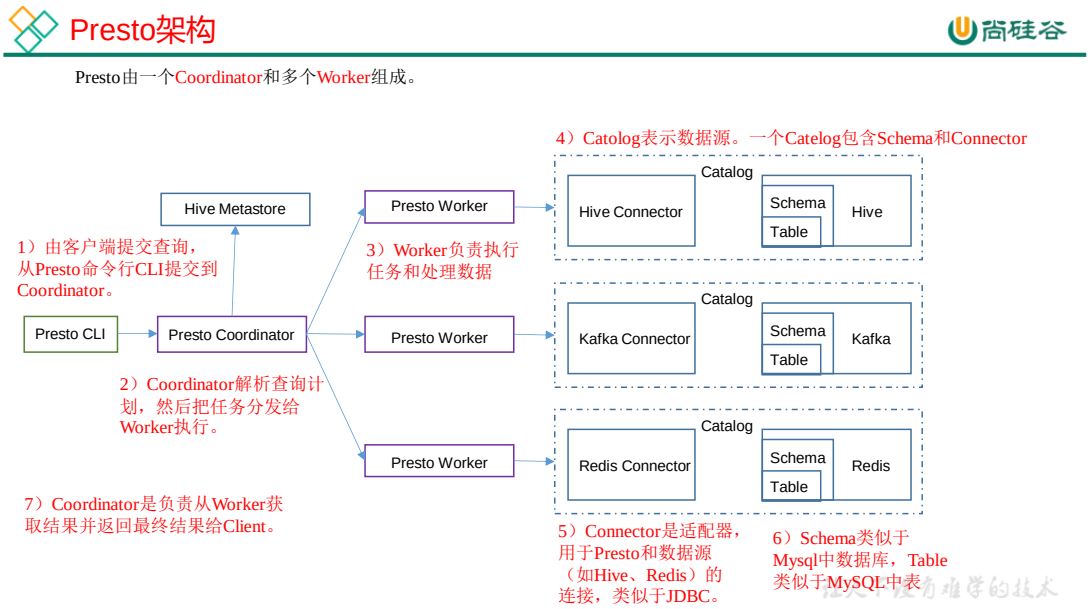
Presto概念

Presto是一个开源的分布式SQL查询引擎，数据量支持GB到PB字节，主要用来处理秒级查询的场景。

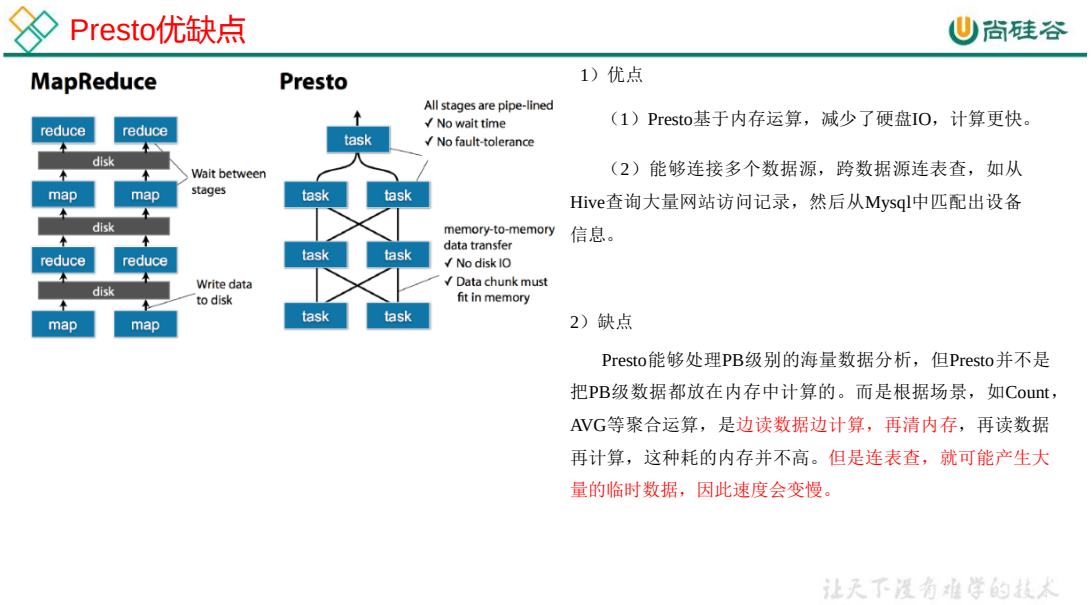
注意：虽然Presto可以解析SQL，但它不是一个标准的数据库。不是MySQL、Oracle的代替品，也不能用来处理在线事务（OLTP）。

让天下没有难学的技术

1.1.2 Presto 架构



1.1.3 Presto 优缺点



1.1.4 Presto、Impala 性能比较

https://blog.csdn.net/u012551524/article/details/79124532

测试结论：Impala 性能稍领先于 Presto，但是 Presto 在数据源支持上非常丰富，包括 Hive、图数据库、传统关系型数据库、Redis 等。

1.2 Presto 安装

1.2.1 Presto Server 安装

0）官网地址

https://prestodb.github.io/

1）下载地址

https://repo1.maven.org/maven2/com/facebook/presto/presto-server/0.196/presto-server-0.196.tar.gz

2）将 presto-server-0.196.tar.gz 导入 hadoop102 的/opt/software 目录下，并解压到/opt/module 目录

```
[atguigu@hadoop102 software]$ tar -zxvf presto-server-0.196.tar.gz -C /opt/module/
```

3）修改名称为 presto

```
[atguigu@hadoop102 module]$ mv presto-server-0.196/ presto
```

4）进入到/opt/module/presto 目录，并创建存储数据文件夹

```
[atguigu@hadoop102 presto]$ mkdir data
```

5）进入到/opt/module/presto 目录，并创建存储配置文件文件夹

```
[atguigu@hadoop102 presto]$ mkdir etc
```

6）配置在/opt/module/presto/etc 目录下添加 jvm.config 配置文件

```
[atguigu@hadoop102 etc]$ vim jvm.config
```

添加如下内容

```
-server
-Xmx16G
-XX:+UseG1GC
-XX:G1HeapRegionSize=32M
-XX:+UseGCOverheadLimit
-XX:+ExplicitGCInvokesConcurrent
-XX:+HeapDumpOnOutOfMemoryError
-XX:+ExitOnOutOfMemoryError
```

7）Presto 可以支持多个数据源，在 Presto 里面叫 catalog，这里我们配置支持 Hive 的数据源，配置一个 Hive 的 catalog

```
[atguigu@hadoop102 etc]$ mkdir catalog
[atguigu@hadoop102 catalog]$ vim hive.properties
```

添加如下内容

```
connector.name=hive-hadoop2
hive.metastore.uri=thrift://hadoop102:9083
```

8）将 hadoop102 上的 presto 分发到 hadoop103、hadoop104

```
[atguigu@hadoop102 module]$ xsync presto
```

9）分发之后，分别进入 hadoop102、hadoop103、hadoop104 三台主机的/opt/module/presto/etc 的路径。配置 node 属性，node id 每个节点都不一样。

更多 Java –大数据 –前端 –python 人工智能资料下载，可百度访问：尚硅谷官网

```
[atguigu@hadoop102 etc]$vim node.properties
node.environment=production
node.id=ffffffff-ffff-ffff-ffff-ffffffffffff
node.data-dir=/opt/module/presto/data

[atguigu@hadoop103 etc]$vim node.properties
node.environment=production
node.id=ffffffff-ffff-ffff-ffff-ffffffffffffe
node.data-dir=/opt/module/presto/data

[atguigu@hadoop104 etc]$vim node.properties
node.environment=production
node.id=ffffffff-ffff-ffff-ffff-ffffffffffffd
node.data-dir=/opt/module/presto/data
```

10）Presto 是由一个 coordinator 节点和多个 worker 节点组成。在 hadoop102 上配置成 coordinator，在 hadoop103、hadoop104 上配置为 worker。

（1）hadoop102 上配置 coordinator 节点

```
[atguigu@hadoop102 etc]$ vim config.properties
```

添加内容如下

```
coordinator=true
node-scheduler.include-coordinator=false
http-server.http.port=8881
query.max-memory=50GB
discovery-server.enabled=true
discovery.uri=http://hadoop102:8881
```

（2）hadoop103、hadoop104 上配置 worker 节点

```
[atguigu@hadoop103 etc]$ vim config.properties
```

添加内容如下

```
coordinator=false
http-server.http.port=8881
query.max-memory=50GB
discovery.uri=http://hadoop102:8881
```

```
[atguigu@hadoop104 etc]$ vim config.properties
```

添加内容如下

```
coordinator=false
http-server.http.port=8881
query.max-memory=50GB
discovery.uri=http://hadoop102:8881
```

11）在 hadoop102 的/opt/module/hive 目录下，启动 Hive Metastore，用 atguigu 角色

```
[atguigu@hadoop102 hive]$
nohup bin/hive --service metastore >/dev/null 2>&1 &
```

12）分别在 hadoop102、hadoop103、hadoop104 上启动 Presto Server

（1）前台启动 Presto，控制台显示日志

```
[atguigu@hadoop102 presto]$ bin/launcher run
[atguigu@hadoop103 presto]$ bin/launcher run
[atguigu@hadoop104 presto]$ bin/launcher run
```

更多 Java –大数据 –前端 –python 人工智能资料下载，可百度访问：尚硅谷官网

（2）后台启动 Presto

```
[atguigu@hadoop102 presto]$ bin/launcher start
[atguigu@hadoop103 presto]$ bin/launcher start
[atguigu@hadoop104 presto]$ bin/launcher start
```

13）日志查看路径/opt/module/presto/data/var/log

1.2.2 Presto 命令行 Client 安装

1）下载 Presto 的客户端

https://repo1.maven.org/maven2/com/facebook/presto/presto-cli/0.196/presto-cli-0.196-executable.jar

2）将 presto-cli-0.196-executable.jar 上传到 hadoop102 的/opt/module/presto 文件夹下

3）修改文件名称

```
[atguigu@hadoop102 presto]$ mv presto-cli-0.196-executable.jar
prestocli
```

4）增加执行权限

```
[atguigu@hadoop102 presto]$ chmod +x prestocli
```

5）启动 prestocli

```
[atguigu@hadoop102 presto]$ ./prestocli --server hadoop102:8881
--catalog hive --schema default
```

6）Presto 命令行操作

Presto 的命令行操作，相当于 Hive 命令行操作。每个表必须要加上 schema。

例如：

```
select * from schema.table limit 100
```

1.2.3 Presto 可视化 Client 安装

1）将 yanagishima-18.0.zip 上传到 hadoop102 的/opt/module 目录

2）解压缩 yanagishima

```
[atguigu@hadoop102 module]$ unzip yanagishima-18.0.zip
cd yanagishima-18.0
```

3）进入到/opt/module/yanagishima-18.0/conf 文件夹，编写 yanagishima.properties 配置

```
[atguigu@hadoop102 conf]$ vim yanagishima.properties
```

添加如下内容

```
jetty.port=7080
presto.datasources=atguigu-presto
presto.coordinator.server.atguigu-presto=http://hadoop102:8881
catalog.atguigu-presto=hive
schema.atguigu-presto=default
sql.query.engines=presto
```

4）在/opt/module/yanagishima-18.0 路径下启动 yanagishima

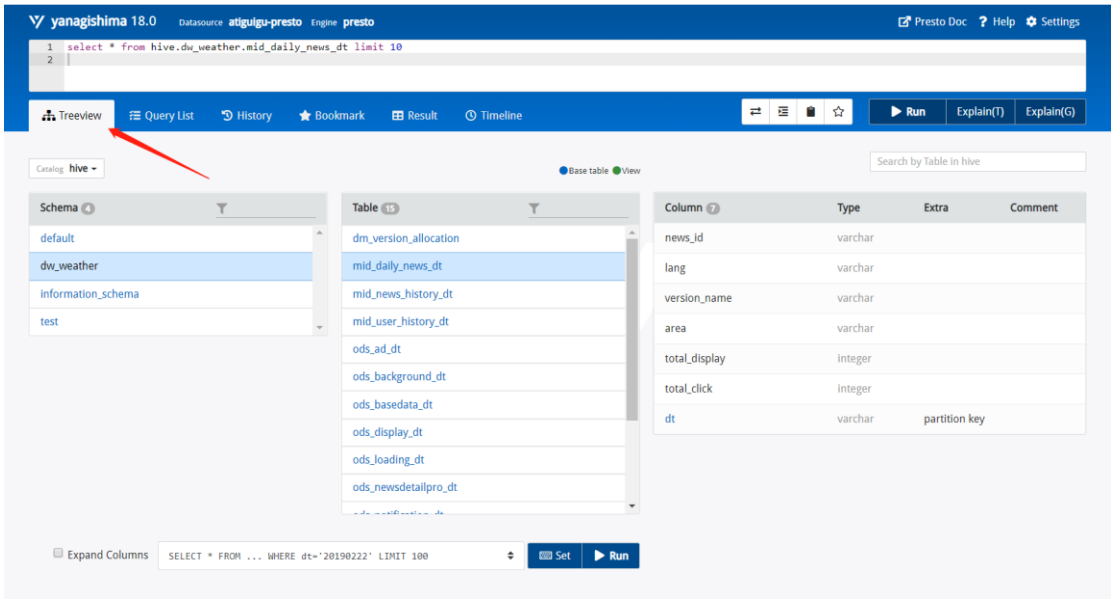
```
[atguigu@hadoop102 yanagishima-18.0]$
nohup bin/yanagishima-start.sh >y.log 2>&1 &
```

5）启动 web 页面

http://hadoop102:7080

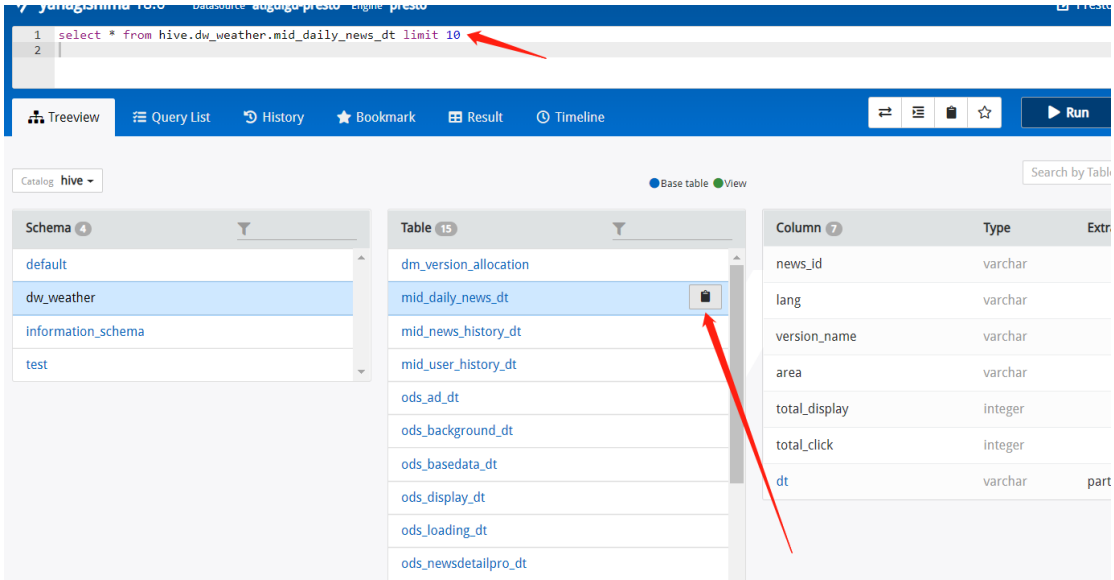
看到界面，进行查询了。

6）查看表结构

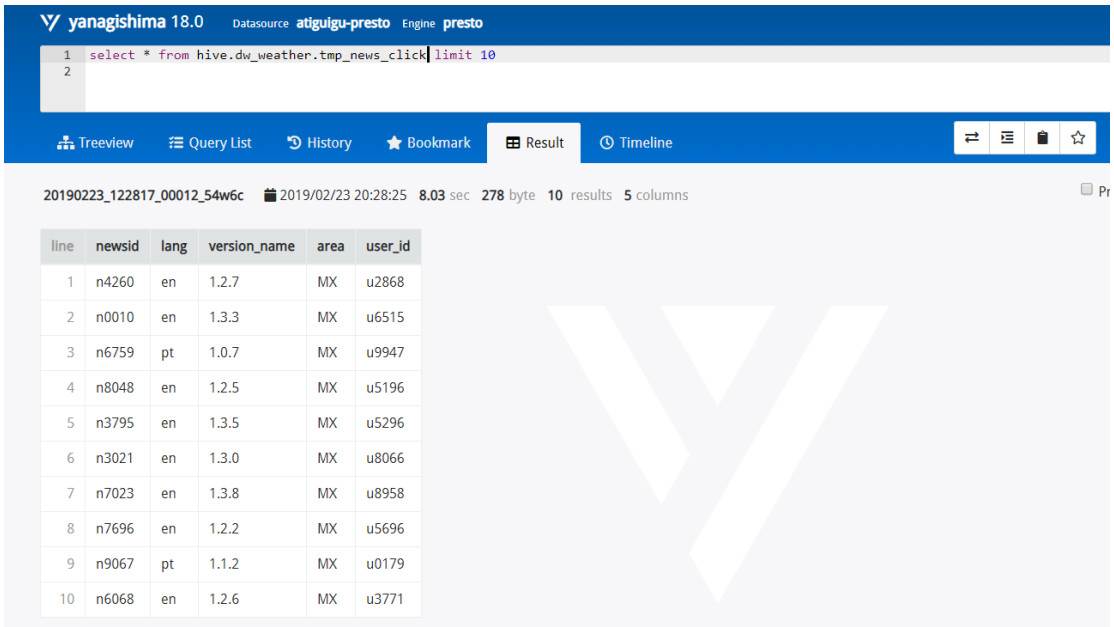


这里有个 Tree View，可以查看所有表的结构，包括 Schema、表、字段等。

比如执行 select * from hive.dw_weather.tmp_news_click limit 10，这个句子里 Hive 这个词可以删掉，是上面配置的 Catalog



每个表后面都有个复制键，点一下会复制完整的表名，然后再上面框里面输入 sql 语句，ctrl+enter 键执行显示结果



line	newsid	lang	version_name	area	user_id
1	n4260	en	1.2.7	MX	u2868
2	n0010	en	1.3.3	MX	u6515
3	n6759	pt	1.0.7	MX	u9947
4	n8048	en	1.2.5	MX	u5196
5	n3795	en	1.3.5	MX	u5296
6	n3021	en	1.3.0	MX	u8066
7	n7023	en	1.3.8	MX	u8958
8	n7696	en	1.2.2	MX	u5696
9	n9067	pt	1.1.2	MX	u0179
10	n6068	en	1.2.6	MX	u3771

1.3 Presto 优化之数据存储

1.3.1 合理设置分区

与 Hive 类似，Presto 会根据元数据信息读取分区数据，合理的分区能减少 Presto 数据读取量，提升查询性能。

1.3.2 使用列式存储

Presto 对 ORC 文件读取做了特定优化，因此在 Hive 中创建 Presto 使用的表时，建议采用 ORC 格式存储。相对于 Parquet，Presto 对 ORC 支持更好。

1.3.3 使用压缩

数据压缩可以减少节点间数据传输对 IO 带宽压力，对于即席查询需要快速解压，建议采用 Snappy 压缩。

1.4 Presto 优化之查询 SQL

1.4.1 只选择使用的字段

由于采用列式存储，选择需要的字段可加快字段的读取、减少数据量。避免采用*读取所有字段。

```
[GOOD]: SELECT time, user, host FROM tbl

[BAD]:  SELECT * FROM tbl
```

1.4.2 过滤条件必须加上分区字段

对于有分区的表，where 语句中优先使用分区字段进行过滤。acct_day 是分区字段，visit_time 是具体访问时间。

```
[GOOD]: SELECT time, user, host FROM tbl where acct_day=20171101

[BAD]:  SELECT * FROM tbl where visit_time=20171101
```

1.4.3 Group By 语句优化

合理安排 Group by 语句中字段顺序对性能有一定提升。将 Group By 语句中字段按照每个字段 distinct 数据多少进行降序排列。

```
[GOOD]: SELECT GROUP BY uid, gender

[BAD]:  SELECT GROUP BY gender, uid
```

1.4.4 Order by 时使用 Limit

Order by 需要扫描数据到单个 worker 节点进行排序，导致单个 worker 需要大量内存。如果是查询 Top N 或者 Bottom N，使用 limit 可减少排序计算和内存压力。

```
[GOOD]: SELECT * FROM tbl ORDER BY time LIMIT 100

[BAD]:  SELECT * FROM tbl ORDER BY time
```

1.4.5 使用 Join 语句时将大表放在左边

Presto 中 join 的默认算法是 broadcast join，即将 join 左边的表分割到多个 worker，然后将 join 右边的表数据整个复制一份发送到每个 worker 进行计算。如果右边的表数据量太大，则可能会报内存溢出错误。

```
[GOOD] SELECT ... FROM large_table l join small_table s on l.id
= s.id
[BAD] SELECT ... FROM small_table s join large_table l on l.id
= s.id
```

1.5 注意事项

1.5.1 字段名引用

避免和关键字冲突：MySQL 对字段加反引号`、Presto 对字段加双引号分割

当然，如果字段名称不是关键字，可以不加这个双引号。

1.5.2 时间函数

对于 Timestamp，需要进行比较的时候，需要添加 Timestamp 关键字，而 MySQL 中对 Timestamp 可以直接进行比较。

```
/*MySQL 的写法*/
```

更多 Java –大数据 –前端 –python 人工智能资料下载，可百度访问：尚硅谷官网

```
SELECT t FROM a WHERE t > '2017-01-01 00:00:00';

/*Presto 中的写法*/
SELECT t FROM a WHERE t > timestamp '2017-01-01 00:00:00';
```

1.5.3 不支持 INSERT OVERWRITE 语法

Presto 中不支持 insert overwrite 语法，只能先 delete，然后 insert into。

1.5.4 PARQUET 格式

Presto 目前支持 Parquet 格式，支持查询，但不支持 insert。

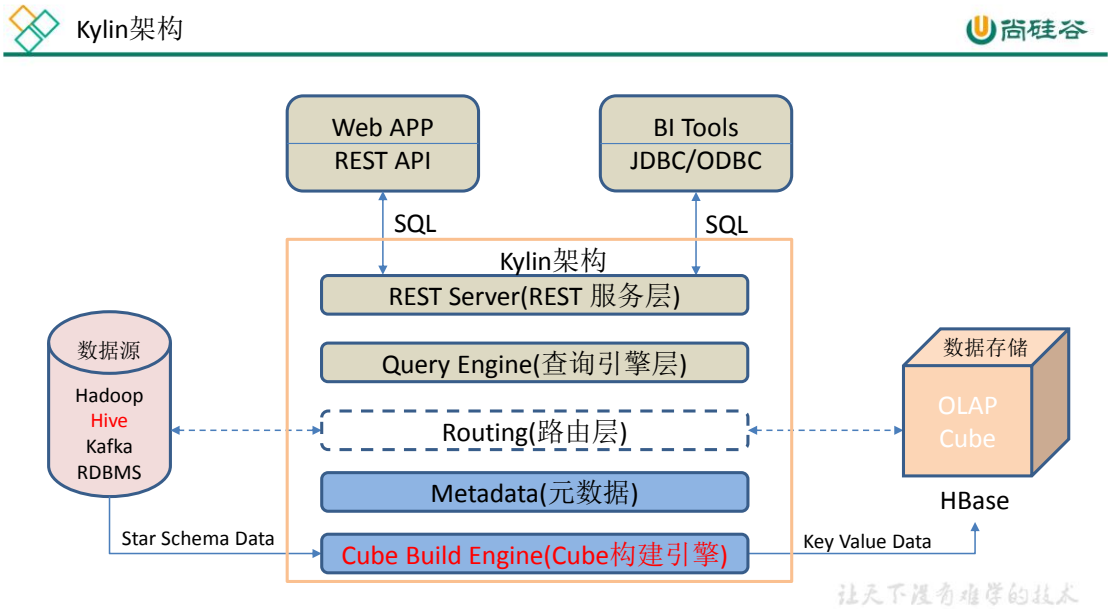
第 2 章 Kylin

2.1 Kylin 简介

2.1.1 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。

2.1.2 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。

更多 Java –大数据 –前端 –python 人工智能资料下载，可百度访问：尚硅谷官网

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）路由器（Routing）

在最初设计时曾考虑过将 Kylin 不能执行的查询引导去 Hive 中继续执行，但在实践后发现 Hive 与 Kylin 的速度差异过大，导致用户无法对查询的速度有一致的期望，很可能大多数查询几秒内就返回结果了，而有些查询则要等几分钟到几十分钟，因此体验非常糟糕。最后这个路由功能在发行版中默认关闭。

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。

2.1.3 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。

2.2 Kylin 安装

2.2.1 Kylin 依赖环境

安装 Kylin 前需先部署好 Hadoop、Hive、Zookeeper、HBase，并且需要在/etc/profile 中配置以下环境变量 HADOOP_HOME，HIVE_HOME，HBASE_HOME，记得 source 使其生效。

尚硅谷大数据技术之HBase.docx

2.2.2 Kylin 搭建

1）上传 Kylin 安装包 apache-kylin-3.0.2-bin.tar.gz

2）解压 apache-kylin-3.0.2-bin.tar.gz 到/opt/module

```
[atguigu@hadoop102 sorfware]$ tar -zxvf apache-kylin-3.0.2-bin.tar.gz -C /opt/module/

[atguigu@hadoop102 module]$ mv /opt/module/apache-kylin-3.0.2-bin /opt/module/kylin
```

2.2.3 Kylin 兼容性问题

修改/opt/module/kylin/bin/find-spark-dependency.sh，排除冲突的 jar 包

需要增加的内容：**! -name '*jackson*' ! -name '*metastore*'**

注意前后保留空格

```
spark_dependency=`find -L $spark_home/jars -name '*.jar' ! -name '*slf4j*' ! -name '*jackson*' ! -name '*metastore*' ! -name '*calcite*' ! -name '*doc*' ! -name '*test*' ! -name '*sources*' ''-printf '%p:' | sed 's/:$//'`
if [ -z "$spark_dependency" ]
then
    quit "spark jars not found"
else
    verbose "spark dependency: $spark_dependency"
    export spark_dependency
fi
echo "export spark_dependency=$spark_dependency" > ${dir}/cached-spark-dependency.sh
```

增加该内容，排除冲突的jar包

2.2.4 Kylin 启动

（1）启动 Kylin 之前，需先启动 Hadoop（hdfs，yarn，jobhistoryserver）、Zookeeper、Hbase

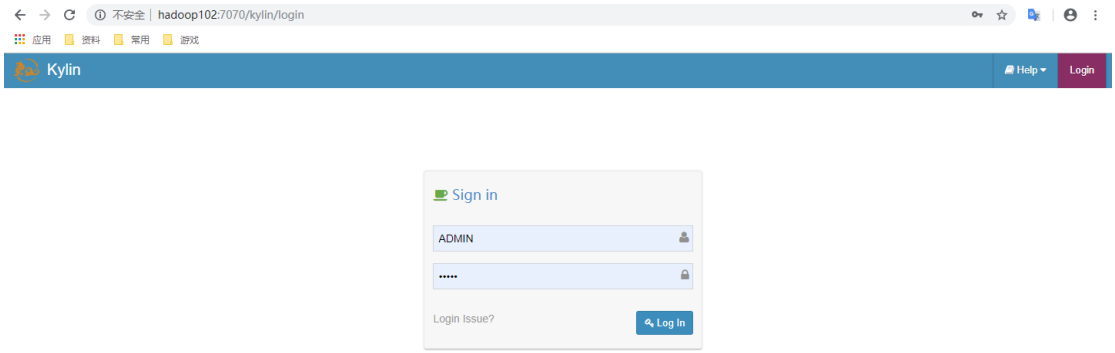
（2）启动 Kylin

```
[atguigu@hadoop102 kylin]$ bin/kylin.sh start
```

启动之后查看各个节点进程：

```
--------------------- hadoop102 ----------------
3360 JobHistoryServer
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
```

在 http://hadoop102:7070/kylin 查看 Web 页面



用户名为：ADMIN，密码为：KYLIN

5）关闭 Kylin

```
[atguigu@hadoop102 kylin]$ bin/kylin.sh stop
```

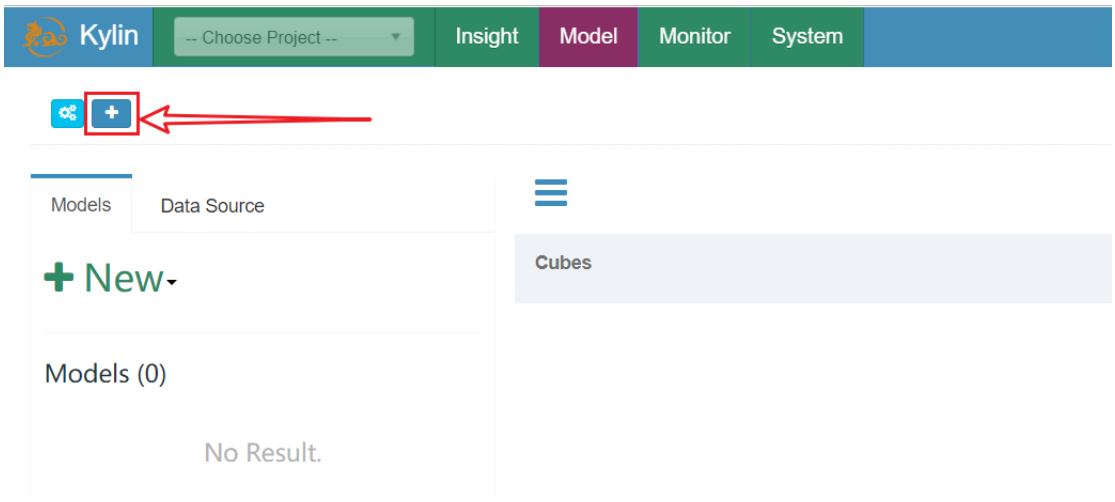
2.3 Kylin 使用

以 gmall 数据仓库中的 dwd_payment_info 作为事实表，dwd_order_info_his、dwd_user_info 作为维度表，构建星型模型，并演示如何使用 Kylin 进行 OLAP 分析。

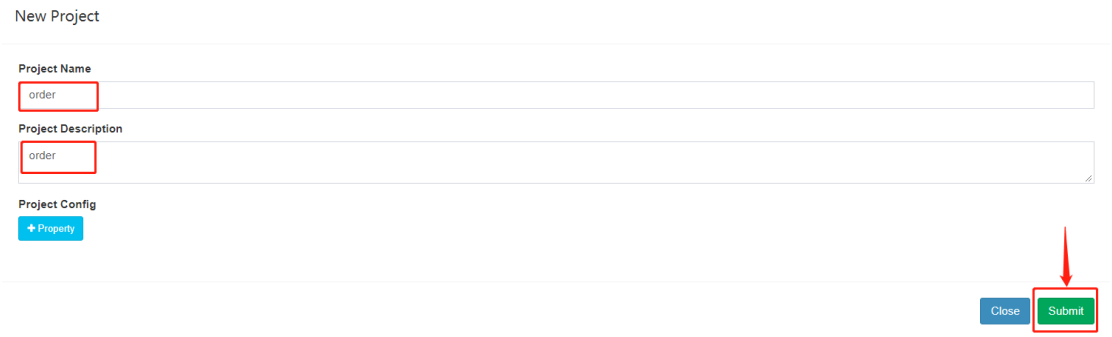
更多 Java –大数据 –前端 –python 人工智能资料下载，可百度访问：尚硅谷官网

2.2.1 创建工程

1）点击下图中的"+"。

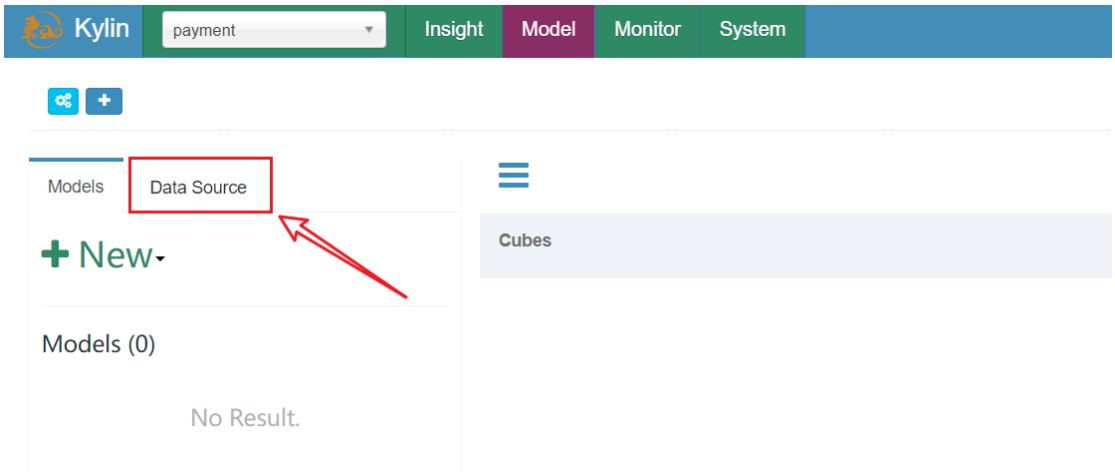


2）填写项目名称和描述信息，并点击 Submit 按钮提交。

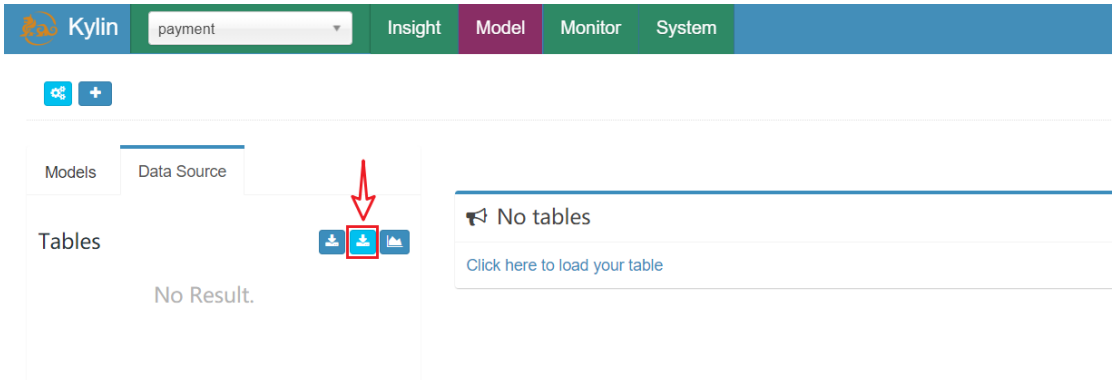


2.2.2 获取数据源

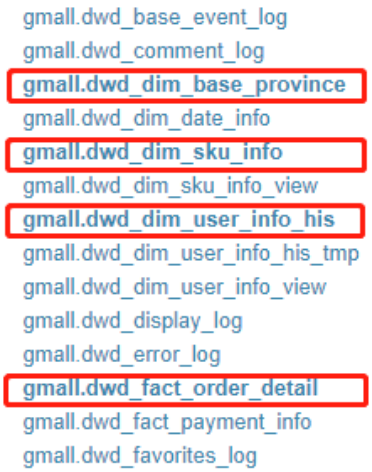
1）点击 DataSource



2）点击下图按钮导入 Hive 表

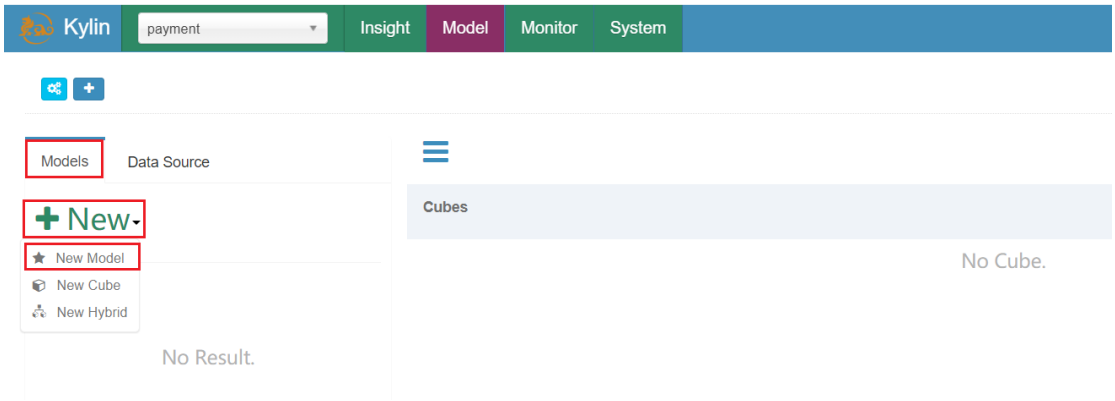


3）选择所需数据表，并点击 Sync 按钮

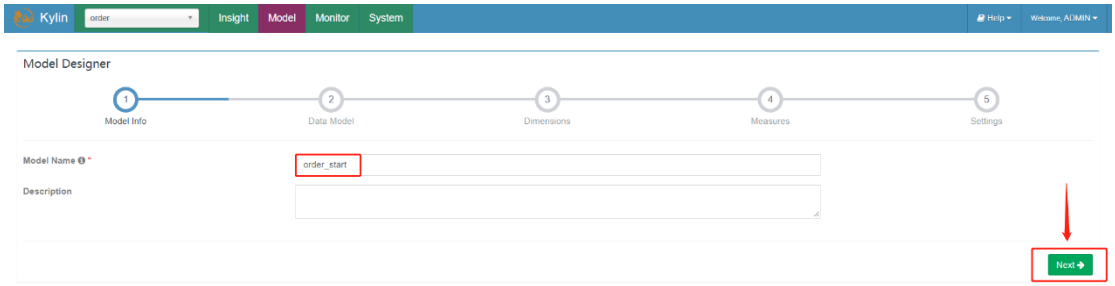


2.2.3 创建 model

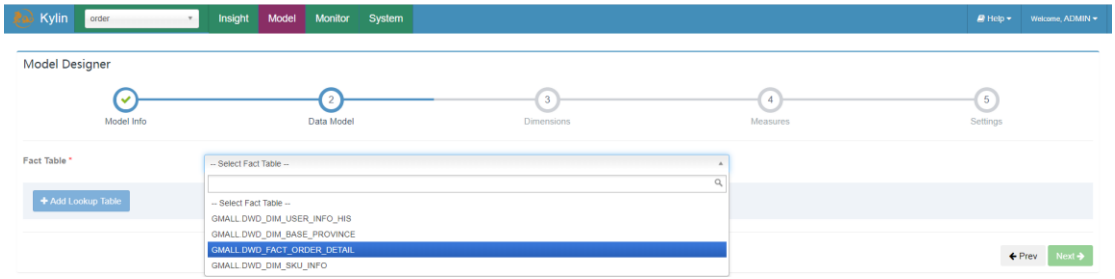
1）点击 Models，点击"+New"按钮，点击"★New Model"按钮。



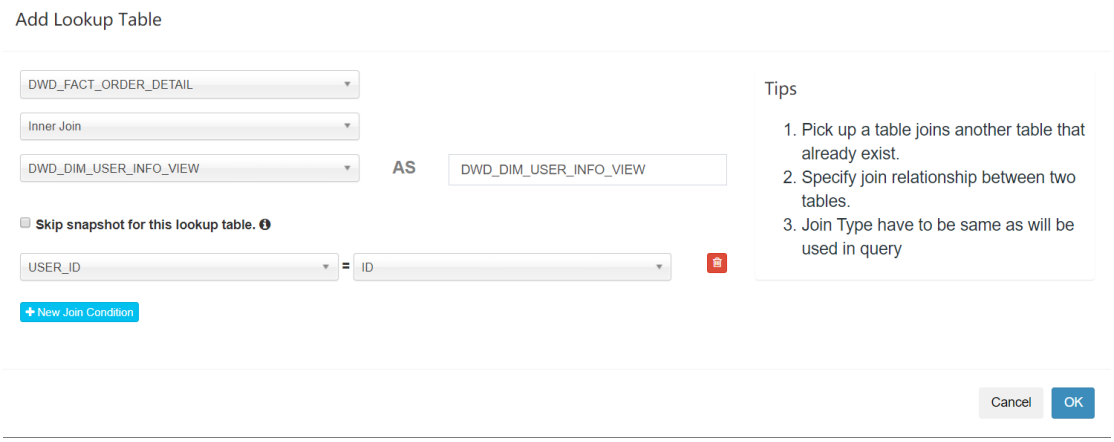
2）填写 Model 信息，点击 Next



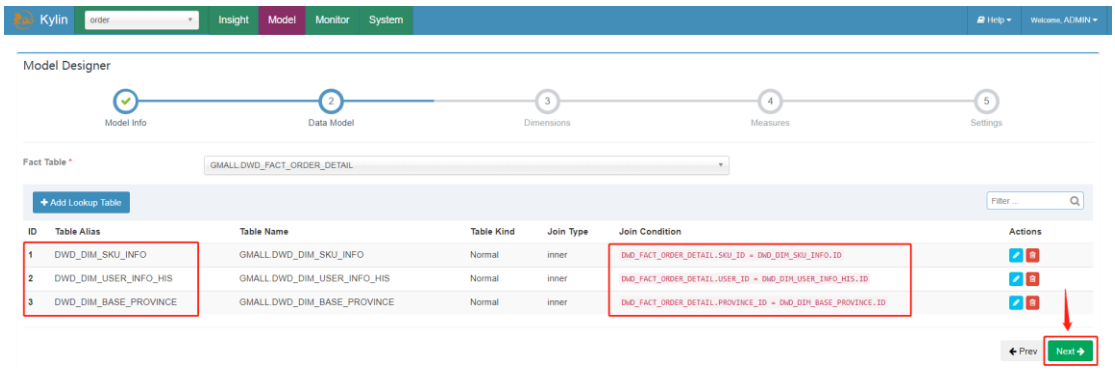
3）指定事实表



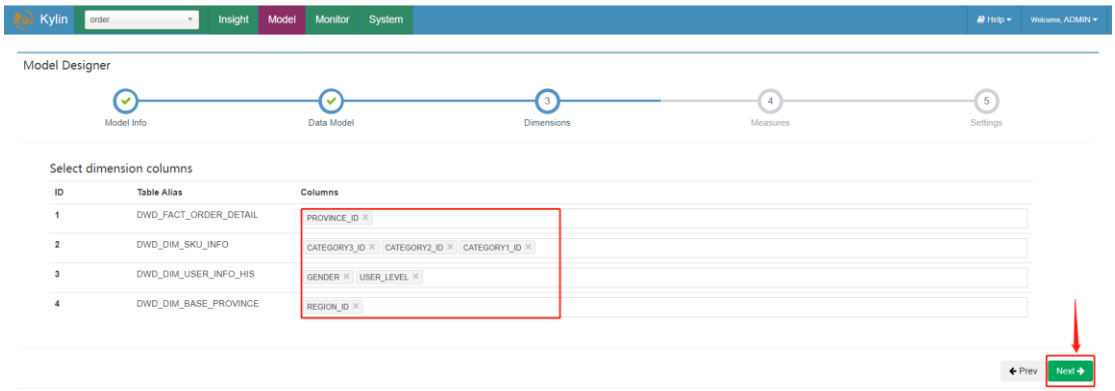
4）选择维度表，并指定事实表和维度表的关联条件，点击 Ok



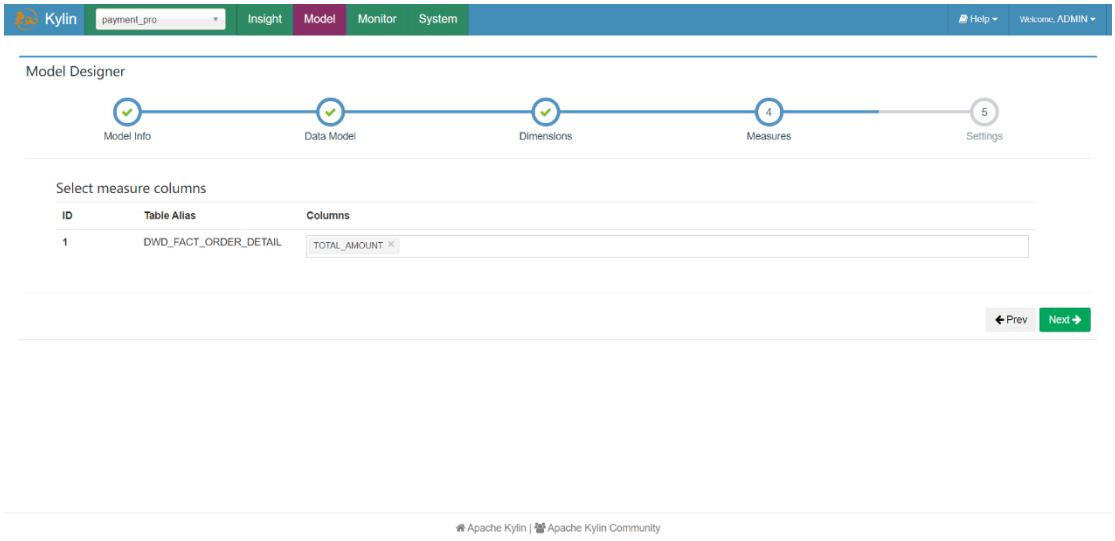
维度表添加完毕之后，点击 Next



5）指定维度字段，并点击 Next



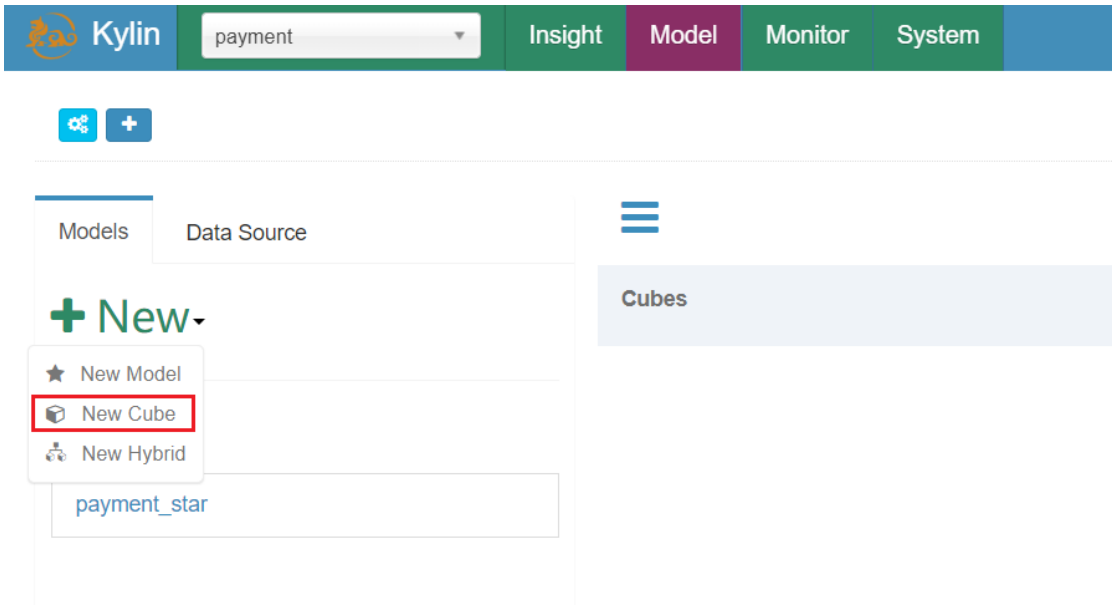
6）指定度量字段，并点击 Next



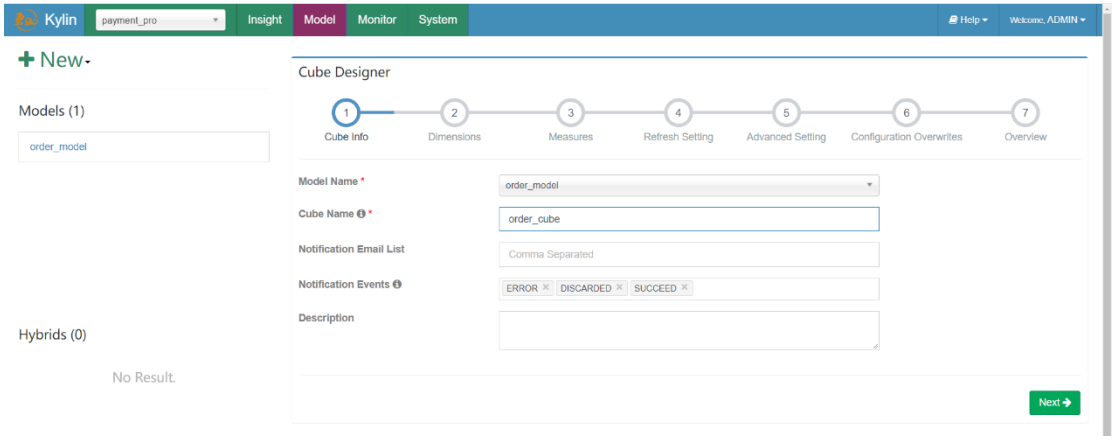
7）指定事实表分区字段（仅支持时间分区），点击 Save 按钮，model 创建完毕

2.2.4 构建 cube

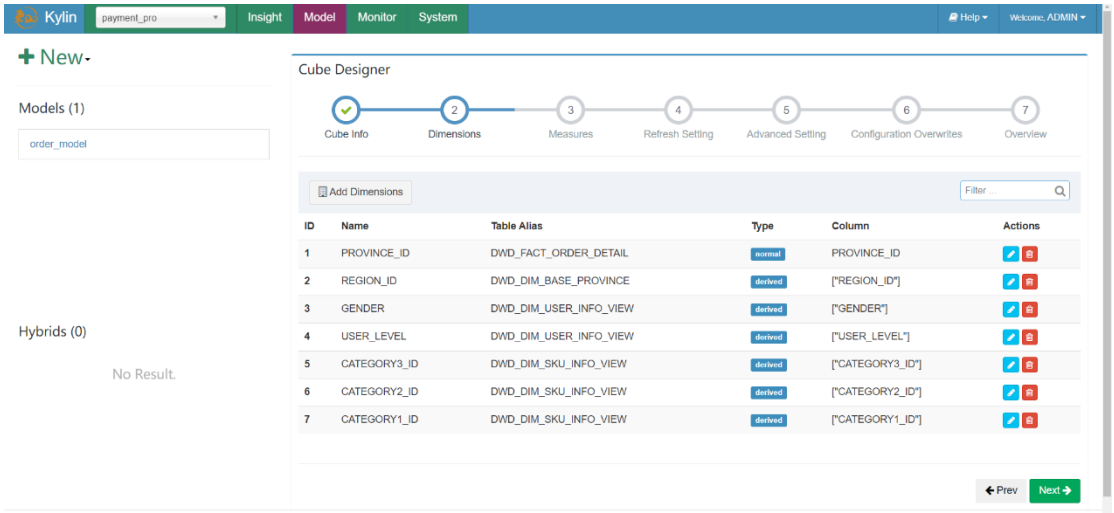
1）点击 new， 并点击 new cube



2）填写 cube 信息，选择 cube 所依赖的 model，并点击 next

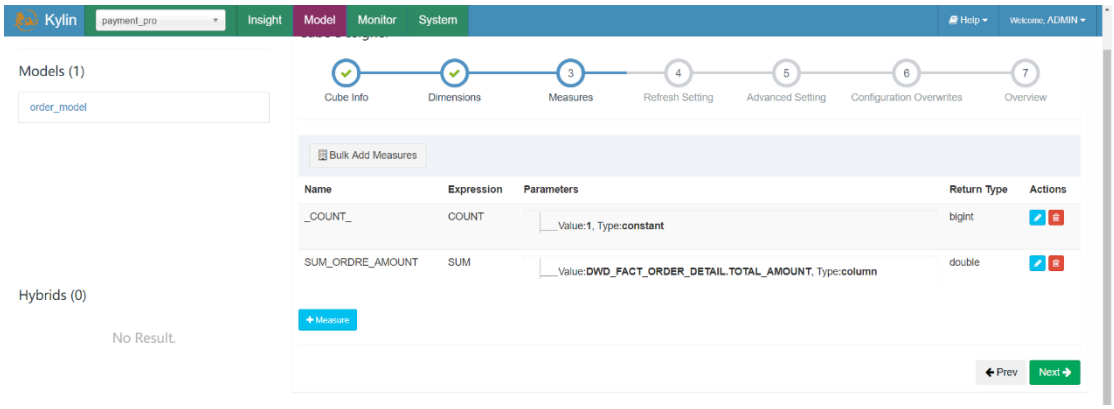


3）选择所需的维度，如下图所示

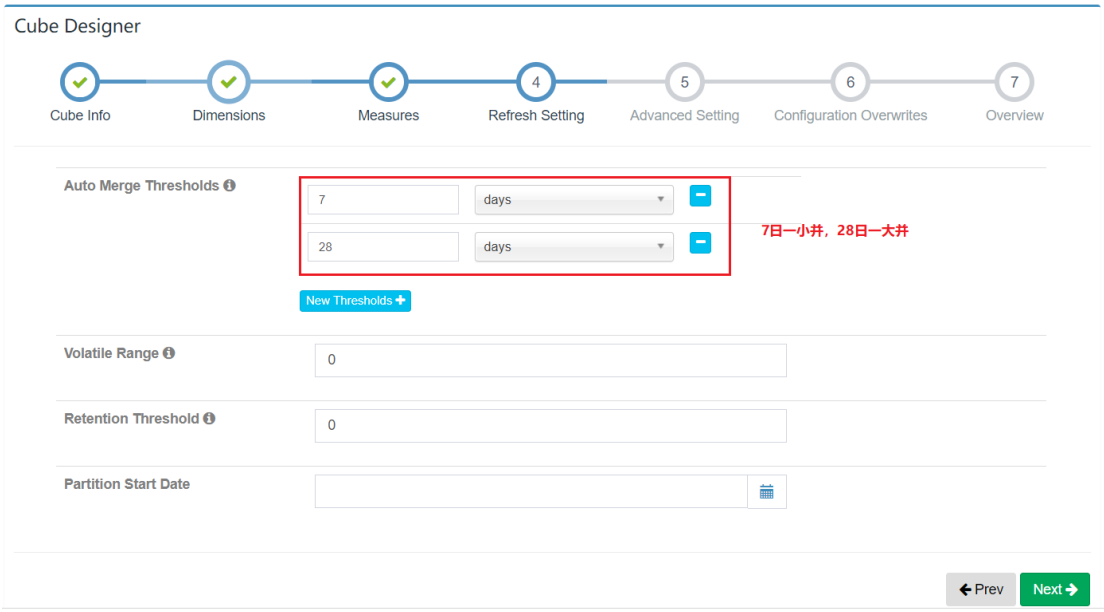


4）选择所需度量值，如下图所示

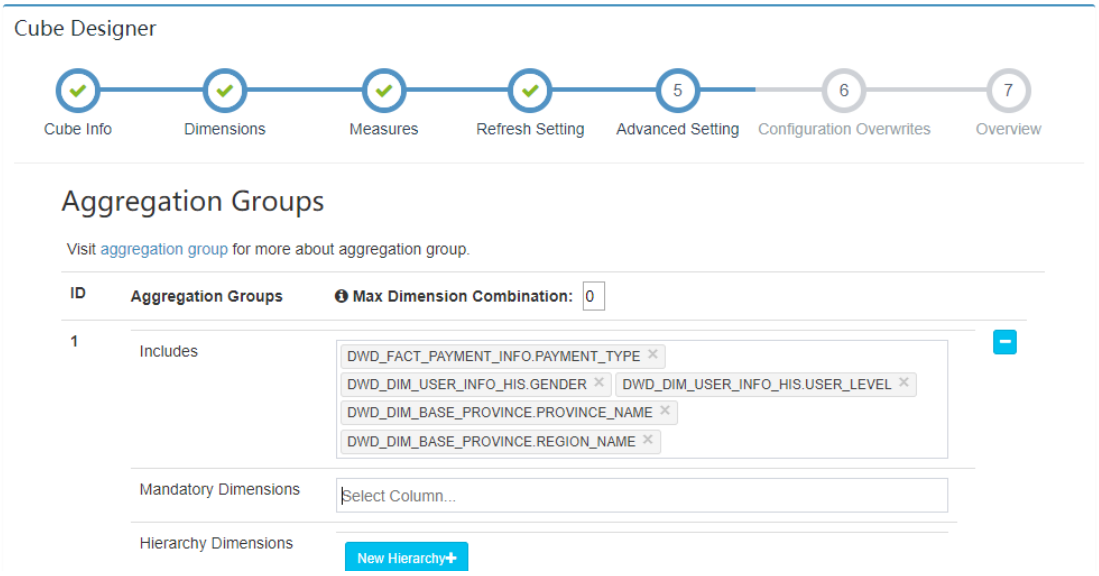
更多 Java –大数据 –前端 –python 人工智能资料下载，可百度访问：尚硅谷官网



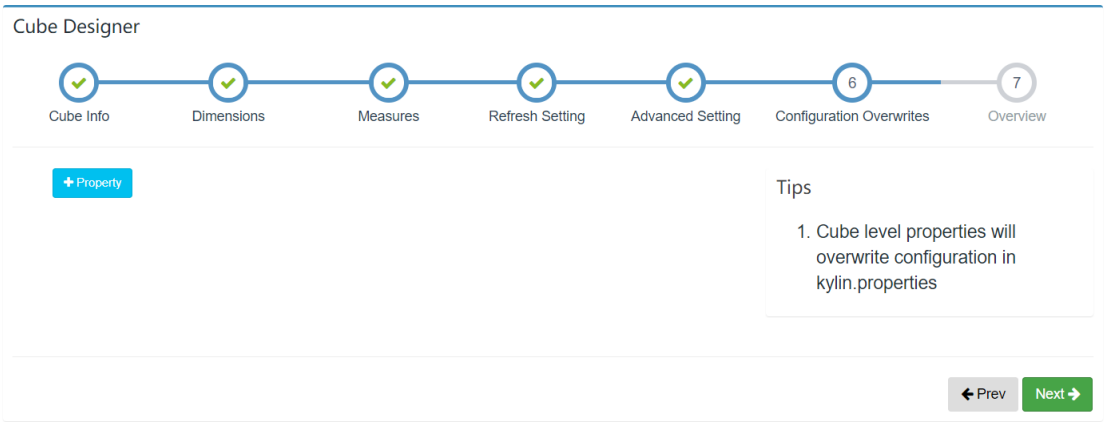
4）cube 自动合并设置，cube 需按照日期分区字段每天进行构建，每次构建的结果会保存在 Hbase 中的一张表内，为提高查询效率，需将每日的 cube 进行合并，此处可设置合并周期。



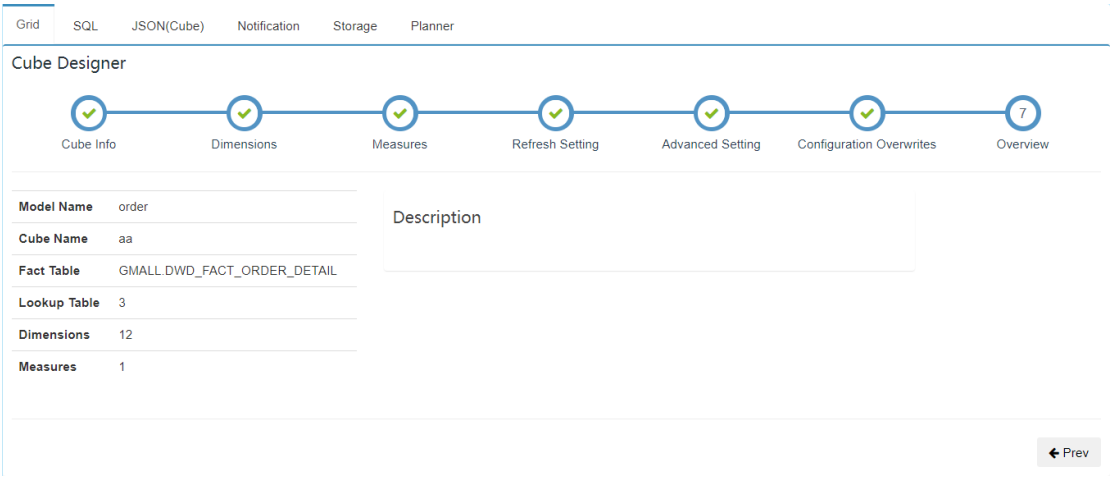
5）Kylin 高级配置（优化相关，暂时跳过）



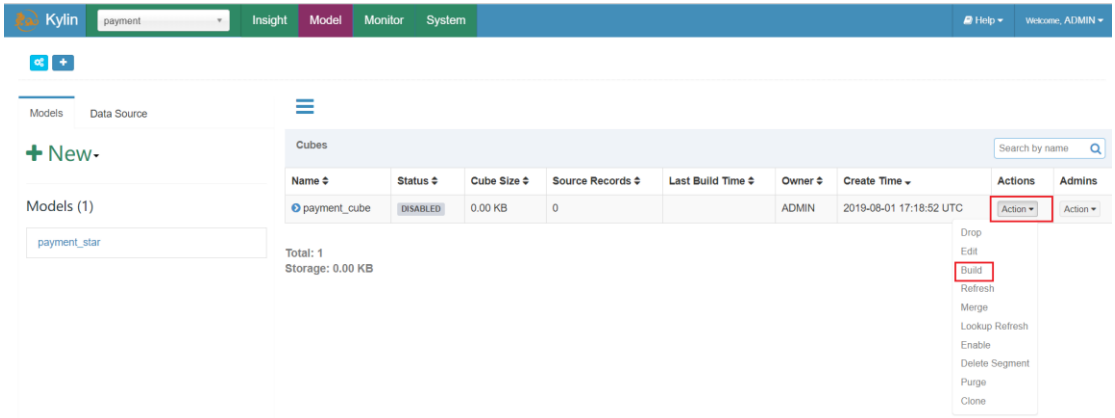
6）Kylin 相关属性配置覆盖



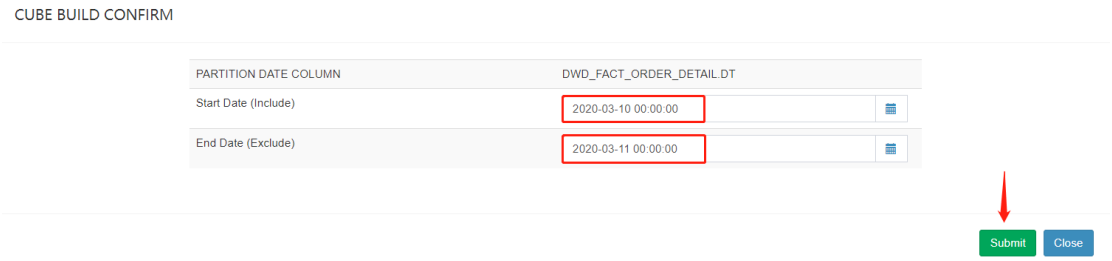
7）Cube 信息总览，点击 Save，Cube 创建完成



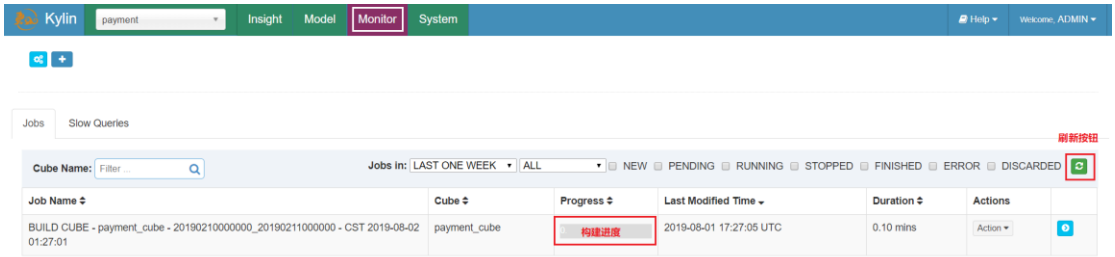
8）构建 Cube（计算），点击对应 Cube 的 action 按钮，选择 build



9）选择要构建的时间区间，点击 Submit



10）点击 Monitor 查看构建进度



2.2.5 使用进阶

1）每日全量维度表及拉链维度表重复 Key 问题如何处理

按照上述流程，会发现，在 cube 构建流程中出现以下错误

Output

```
org.apache.kylin.engine.mr.exception.HadoopShellException: java.lang.RuntimeException: Checking snapshot of TableRef[DWD_ORDER_INFO] failed.
        at org.apache.kylin.cube.cli.DictionaryGeneratorCLI.processSegment(DictionaryGeneratorCLI.java:103)
        at org.apache.kylin.cube.cli.DictionaryGeneratorCLI.processSegment(DictionaryGeneratorCLI.java:50)
        at org.apache.kylin.engine.mr.steps.CreateDictionaryJob.run(CreateDictionaryJob.java:73)
        at org.apache.kylin.engine.mr.MRUtil.runMRJob(MRUtil.java:92)
        at org.apache.kylin.engine.mr.common.HadoopShellExecutable.doWork(HadoopShellExecutable.java:63)
        at org.apache.kylin.job.execution.AbstractExecutable.execute(AbstractExecutable.java:164)
        at org.apache.kylin.job.execution.DefaultChainedExecutable.doWork(DefaultChainedExecutable.java:70)
        at org.apache.kylin.job.execution.AbstractExecutable.execute(AbstractExecutable.java:164)
        at org.apache.kylin.job.impl.threadpool.DefaultScheduler$JobRunner.run(DefaultScheduler.java:113)
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
        at java.lang.Thread.run(Thread.java:748)
Caused by: java.lang.IllegalStateException: The table: DWD_ORDER_INFO Dup key found, key=[1], value1=[1,651,3,1,1,2828202075,2019-02-10 08:10:13.0,2019-02-1
```

错误原因分析：

上述错误原因是 model 中的维度表 dwd_dim_user_info_his 为拉链表、dwd_dim_sku_info 为每日全量表，故使用整张表作为维度表，必然会出现订单表中同一个 user_id 或者 sku_id 对应多条数据的问题，针对上述问题，有以下两种解决方案。

方案一：在 hive 中创建维度表的临时表，该临时表中只存放维度表最新的一份完整的数据，在 kylin 中创建模型时选择该临时表作为维度表。

方案二：与方案一思路相同，但不使用物理临时表，而选用视图（view）实现相同的功能。

此处采用方案二：

（1）创建维度表视图

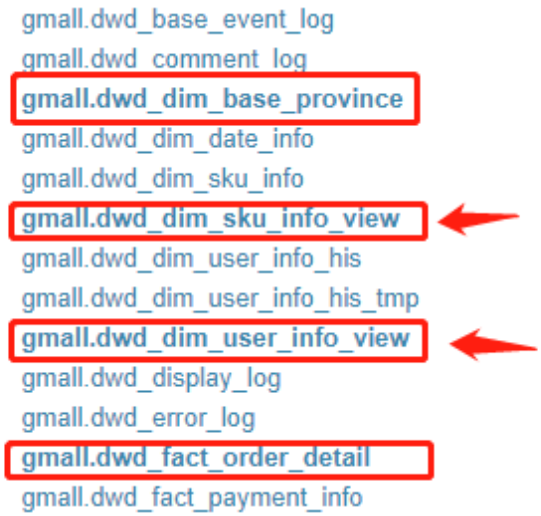
```
--拉链维度表视图
```

```
create view dwd_dim_user_info_his_view as select * from dwd_dim_user_info_his where end_date='9999-99-99';

--全量维度表视图
create view dwd_dim_sku_info_view as select * from dwd_dim_sku_info where dt=date_add(current_date,-1);

--当前情形我们先创建一个 2020-03-10 的视图
create view dwd_dim_sku_info_view as select * from dwd_dim_sku_info where dt='2020-03-10';
```

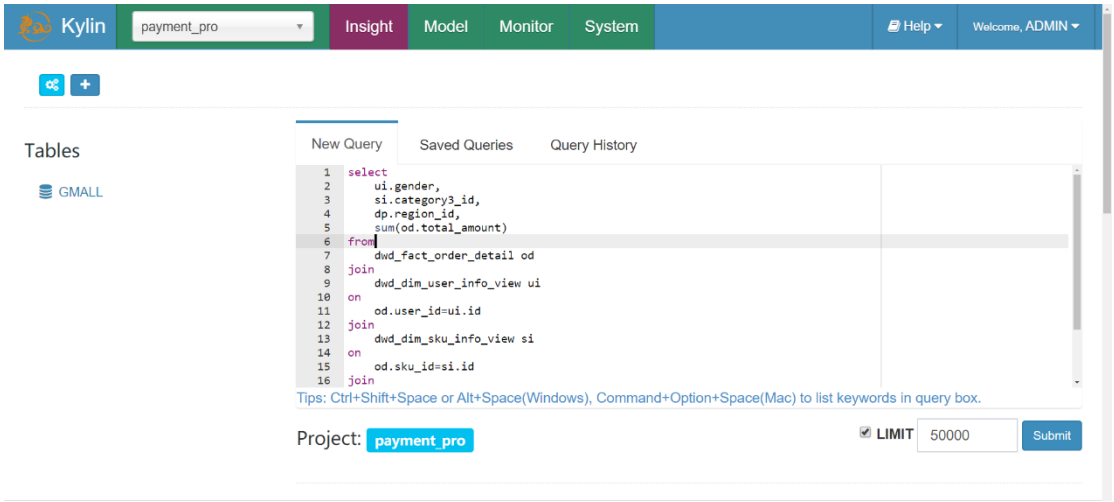
（2）在 DataSource 中导入新创建的视图，之前的维度表，可选择性删除。



（3）重新创建 model、cube。

（4）查询结果

```
select
    ui.gender,
    si.category3_id,
    dp.region_id,
    sum(od.total_amount)
from
    dwd_fact_order_detail od
join
    dwd_dim_user_info_view ui
on
    od.user_id=ui.id
join
    dwd_dim_sku_info_view si
on
    od.sku_id=si.id
join
    dwd_dim_base_province dp
on
    od.province_id=dp.id
group by
    ui.gender,si.category3_id,dp.region_id;
```



2）如何实现每日自动构建 cube

Kylin 提供了 Restful API，因次我们可以将构建 cube 的命令写到脚本中，将脚本交给 azkaban 或者 oozie 这样的调度工具，以实现定时调度的功能。

脚本如下:

```
#!/bin/bash
cube_name=order_cube
do_date=`date -d '-1 day' +%F`

#获取 00:00 时间戳
start_date_unix=`date -d "$do_date 08:00:00" +%s`
start_date=$(($start_date_unix*1000))

#获取 24:00 的时间戳
stop_date=$(($start_date+86400000))

curl -X PUT -H "Authorization: Basic QURNSU46S1lMSU4=" -H
'Content-Type: application/json' -d '{"startTime":'$start_date',
"endTime":'$stop_date',                    "buildType":"BUILD"}'
http://hadoop102:7070/kylin/api/cubes/$cube_name/build
```

2.4 Kylin Cube 构建原理

2.4.1 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。

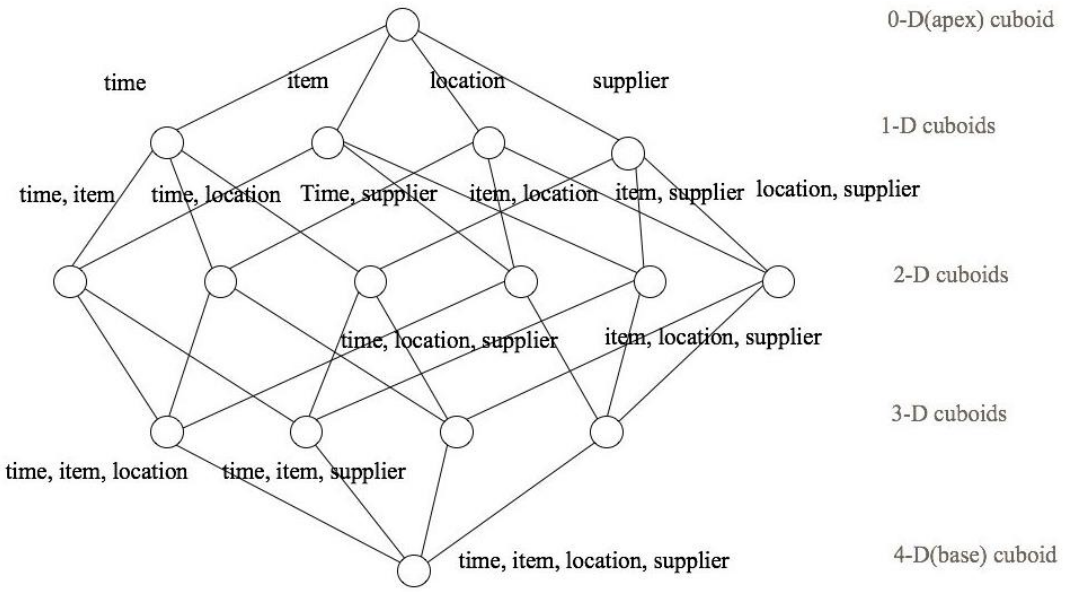
度量：即被聚合（观察）的统计值，也就是聚合运算的结果。比如说员工数据中不同性别员工的人数，又或者说在同一年入职的员工有多少。

2.4.2 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 2^n 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 2^4= 16 种，如下图所示：



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

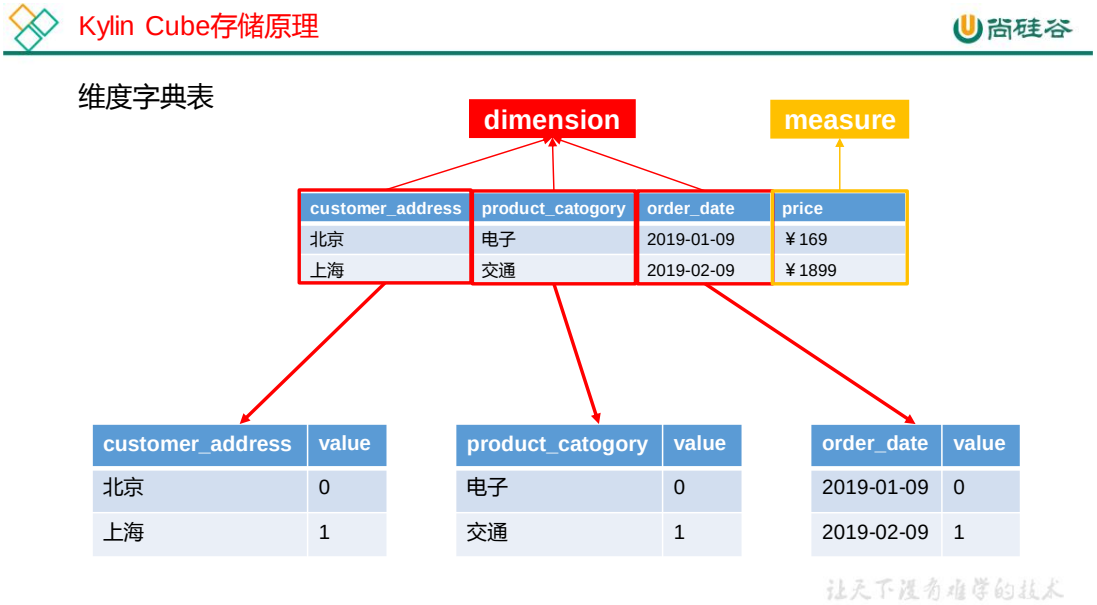
二维度（2D）的组合有：[time, item]、[time, location]、[time, supplier]、[item, location]、[item, supplier]、[location, supplier]3 种；

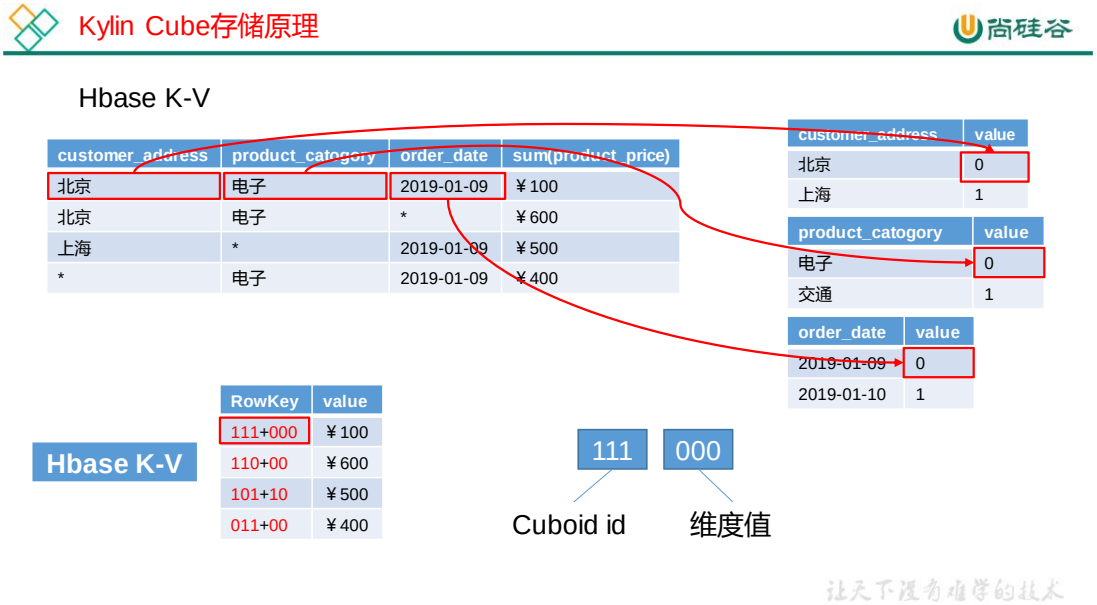
三维度（3D）的组合也有 4 种；

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。

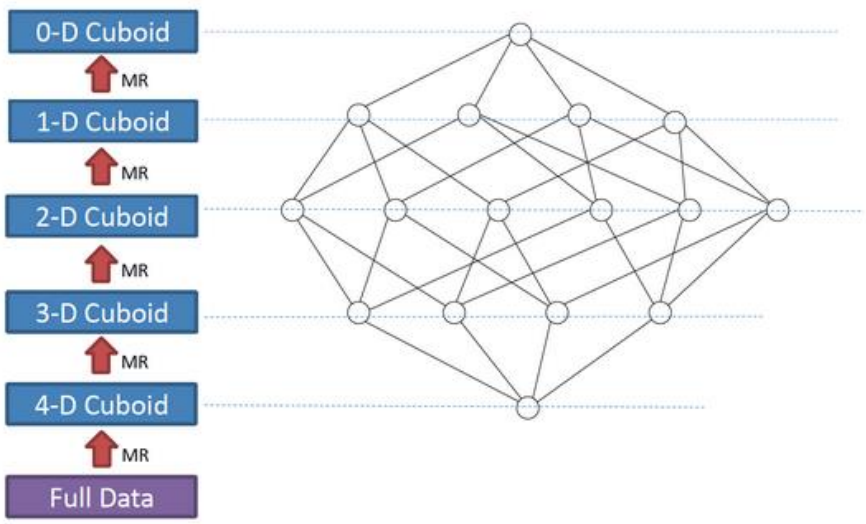
2.4.4 Cube 存储原理





2.4.3 Cube 构建算法

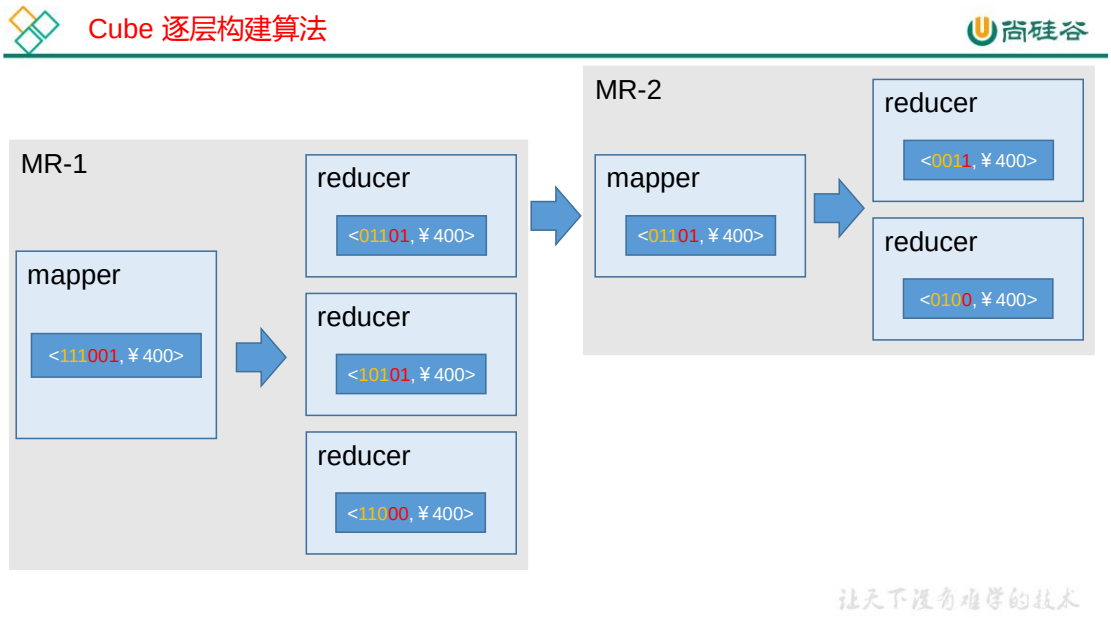
1）逐层构建算法（layer）



逐层算法

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 次 MapReduce Job。



算法优点：

1）此算法充分利用了 MapReduce 的优点，处理了中间复杂的排序和 shuffle 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，此算法非常稳定，即便是集群资源紧张时，也能保证最终能够完成。

算法缺点：

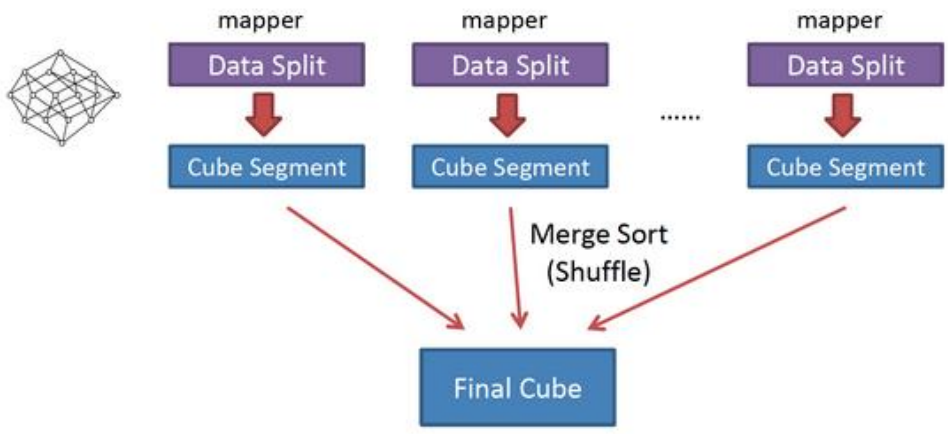
1）当 Cube 有比较多维度的时候，所需要的 MapReduce 任务也相应增加；由于 Hadoop 的任务调度需要耗费额外资源，特别是集群较庞大的时候，反复递交任务造成的额外开销会相当可观；

2）由于 Mapper 逻辑中并未进行聚合操作，所以每轮 MR 的 shuffle 工作量都很大，导致效率低下。

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。

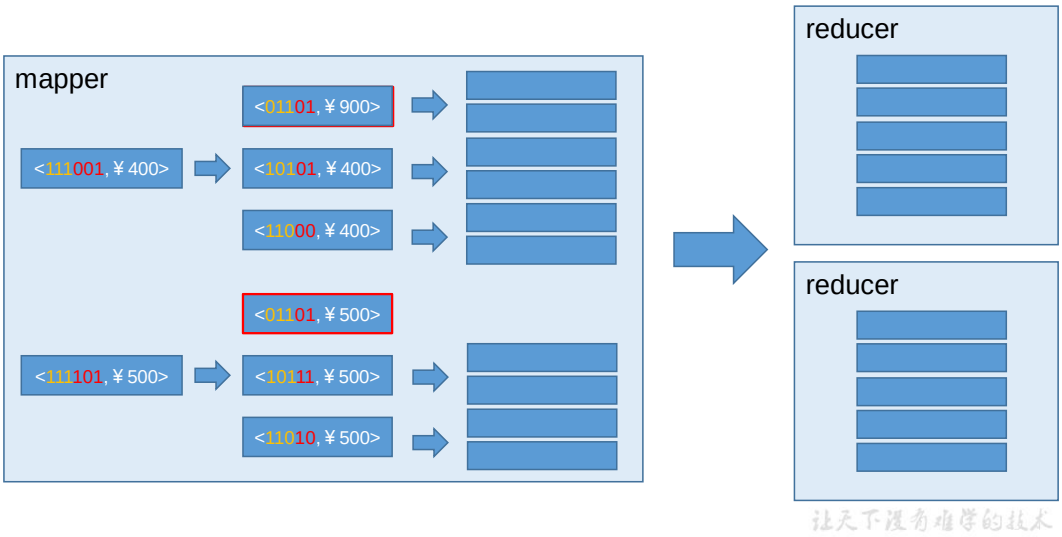
2）快速构建算法（inmem）



快速Cube算法

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，该算法的主要思想是，每个 Mapper 将其所分配到的数据块，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。

Cube 快速构建算法



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，Combiner 也不再需要；

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

2.5 Kylin Cube 构建优化

2.5.1 使用衍生维度（derived dimension）

衍生维度用于在有效维度内将维度表上的非主键维度排除掉，并使用维度表的主键（其实是事实表上相应的外键）来替代它们。Kylin 会在底层记录维度表主键与维度表其他维度之间的映射关系，以便在查询时能够动态地将维度表的主键“翻译”成这些非主键维度，并进行实时聚合。

衍生维度原理

fact

A_id	C
1	?
2	?
3	?
4	?

dim_A

id	B	E
1	北京	a
2	深圳	b
3	上海	c
4	杭州	a

需求：分析count(C)和E之间的关系

1.构建cube时用A_id代替E

A_id	count(C)
1	1
2	1
3	1
4	1

F(A) = E

A	E
1	a
2	b
3	c
4	a

2.查询时用E替换A_id

E	count(C)
a	1
b	1
c	1
a	1

3.聚合

查询结果

E	count(C)
a	2
b	1
c	1

让天下没有难学的技术

虽然衍生维度具有非常大的吸引力，但这也并不是说所有维度表上的维度都得变成衍生维度，如果从维度表主键到某个维度表维度所需要的聚合工作量非常大，则不建议使用衍生维度。

2.5.2 使用聚合组（Aggregation group）

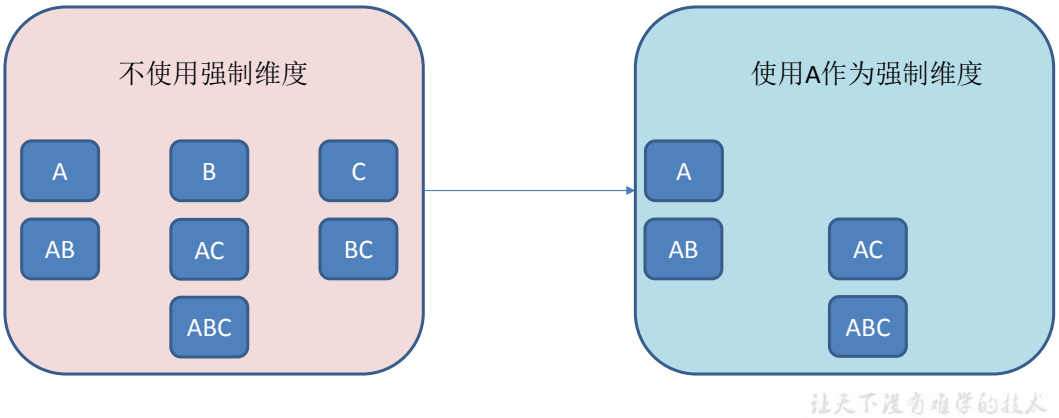
聚合组（Aggregation Group）是一种强大的剪枝工具。聚合组假设一个 Cube 的所有维度均可以根据业务需求划分成若干组（当然也可以是一个组），由于同一个组内的维度更可能同时被同一个查询用到，因此会表现出更加紧密的内在关联。每个分组的维度集合均是 Cube 所有维度的一个子集，不同的分组各自拥有一套维度集合，它们可能与其他分组有相同的维度，也可能没有相同的维度。每个分组各自独立地根据自身的规则贡献出一批需要被物化的 Cuboid，所有分组贡献的 Cuboid 的并集就成为了当前 Cube 中所有需要物化的 Cuboid 的集合。不同的分组有可能会贡献出相同的 Cuboid，构建引擎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

强制维度

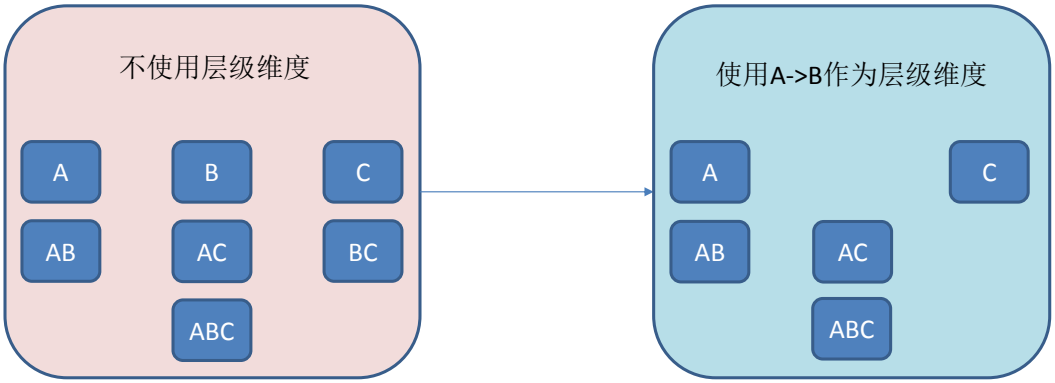
假设有A、B和C三个维度



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。

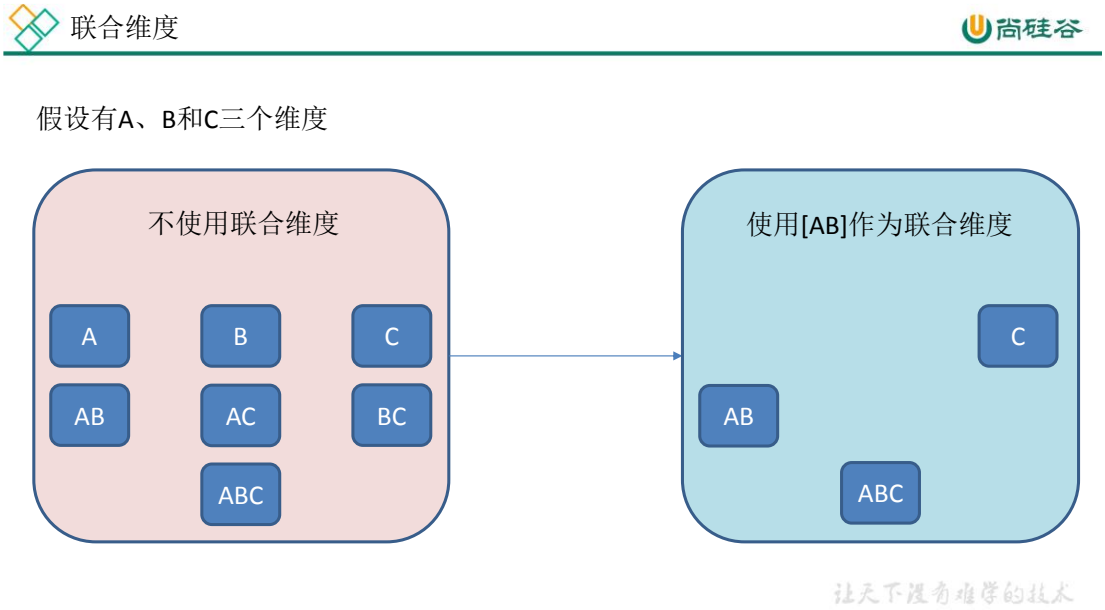
层级维度

假设有A、B和C三个维度

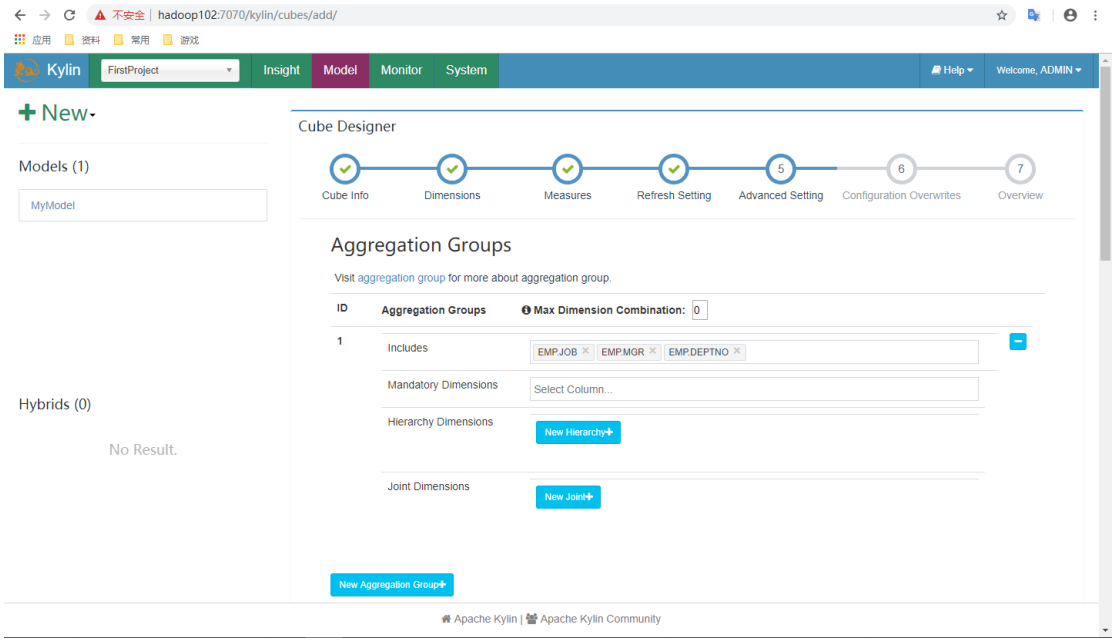


3）联合维度（Joint），每个联合中包含两个或更多个维度，如果某些列形成一个联合，那么在该分组产生的任何 Cuboid 中，这些联合维度要么一起出现，要么都不出现。每个分组中可以有 0 个或多个联合，但是不同的联合之间不应当有共享的维度（否则它们可以合并成一个联合）。如果根据这个分组的业务逻辑，多个维度在查询中总是同时出现，则可以在

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

更多 Java –大数据 –前端 –python 人工智能资料下载，可百度访问：尚硅谷官网

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。

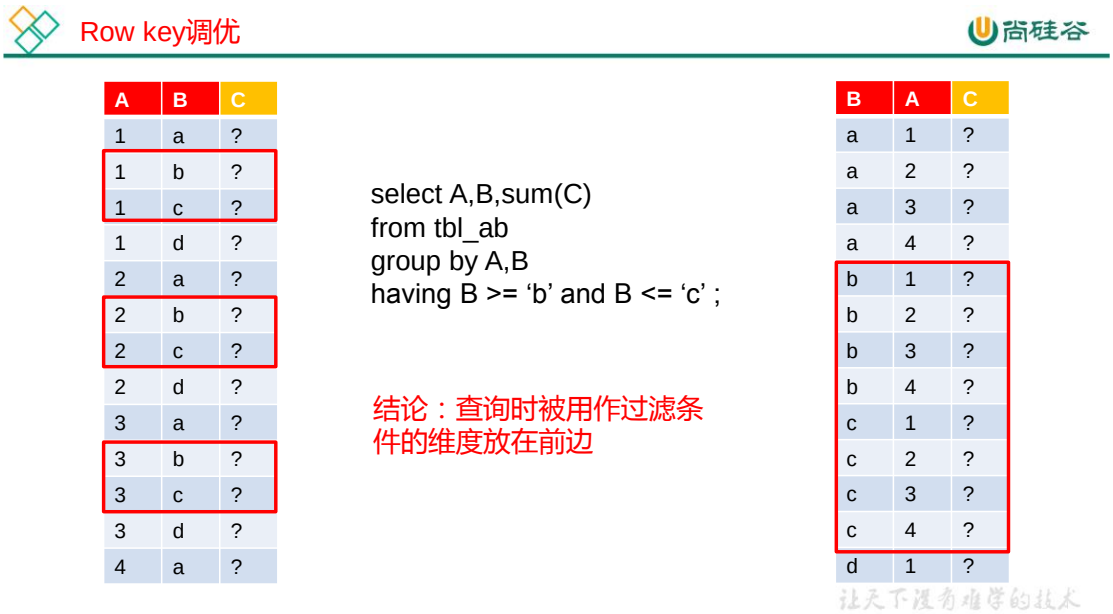
2.5.3 Row Key 优化

Kylin 会把所有的维度按照顺序组合成一个完整的 Rowkey，并且按照这个 Rowkey 升序排列 Cuboid 中所有的行。

设计良好的 Rowkey 将更有效地完成数据的查询过滤和定位，减少 IO 次数，提高查询速度，维度在 rowkey 中的次序，对查询性能有显著的影响。

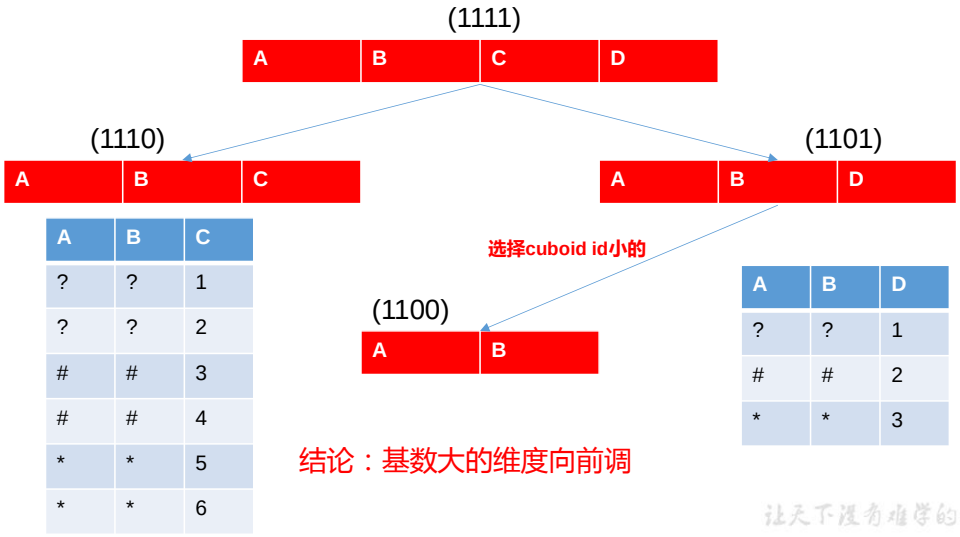
Row key 的设计原则如下：

1）被用作过滤的维度放在前边。



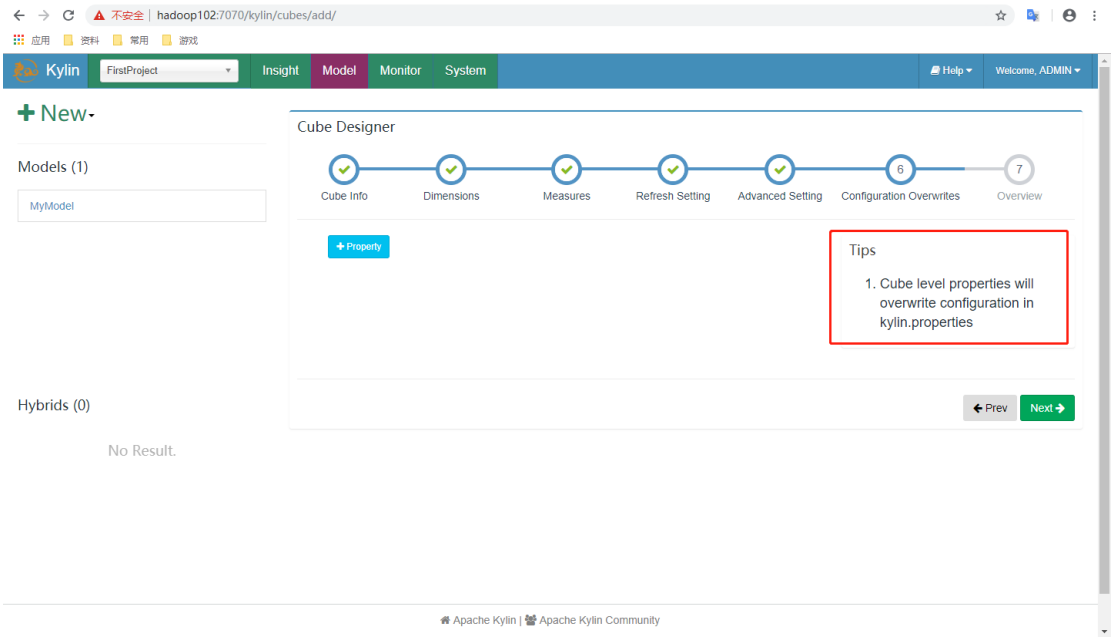
2）基数大的维度放在基数小的维度前边。

Row key调优



2.5.4 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。

2.6 Kylin BI 工具集成

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。

2.6.1 JDBC

1）新建项目并导入依赖

```
    <dependencies>
        <dependency>
            <groupId>org.apache.kylin</groupId>
            <artifactId>kylin-jdbc</artifactId>
            <version>3.0.2</version>
        </dependency>
    </dependencies>
```

2）编码

```
package com.atguigu;
```

```
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

2.6.2 Zepplin

1）Zepplin 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

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

（3）修改名称

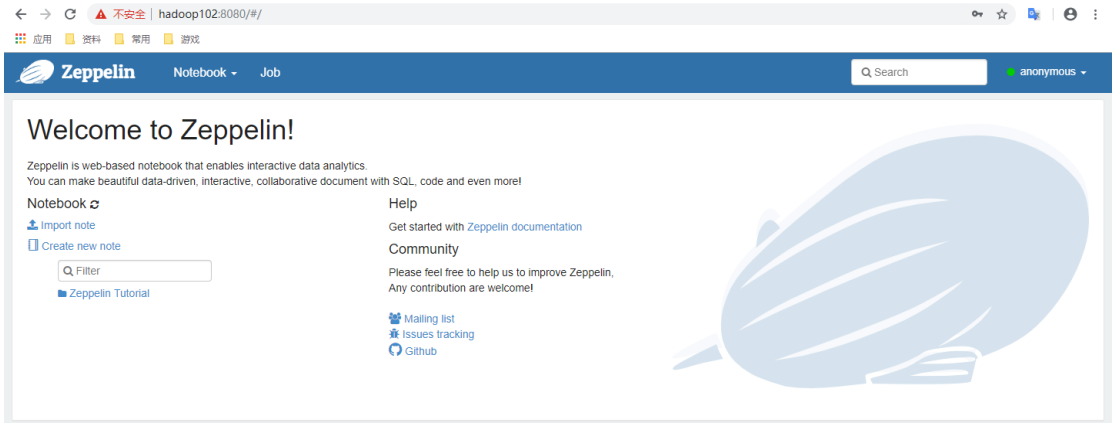
```
[atguigu@hadoop102 module]$ mv zeppelin-0.8.0-bin-all/ zeppelin
```

（4）启动

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

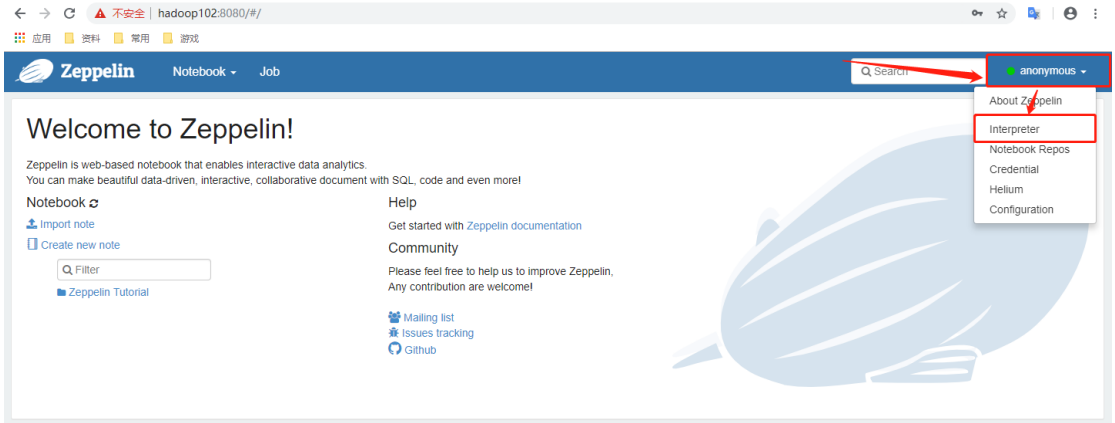
可登录网页查看，web 默认端口号为 8080

http://hadoop102:8080

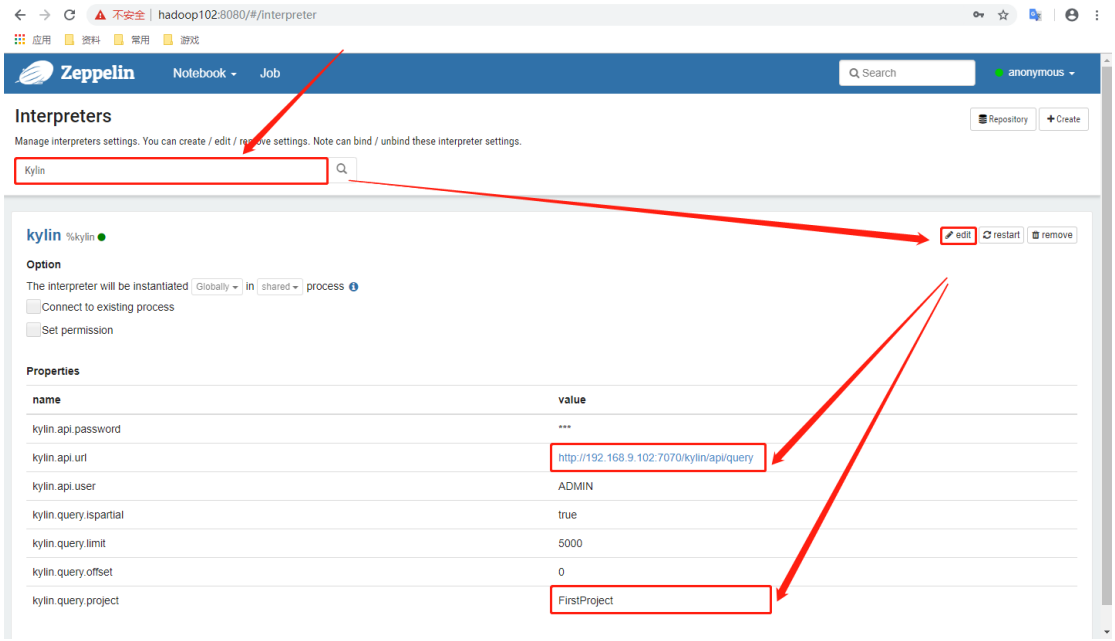


2）配置 Zepplin 支持 Kylin

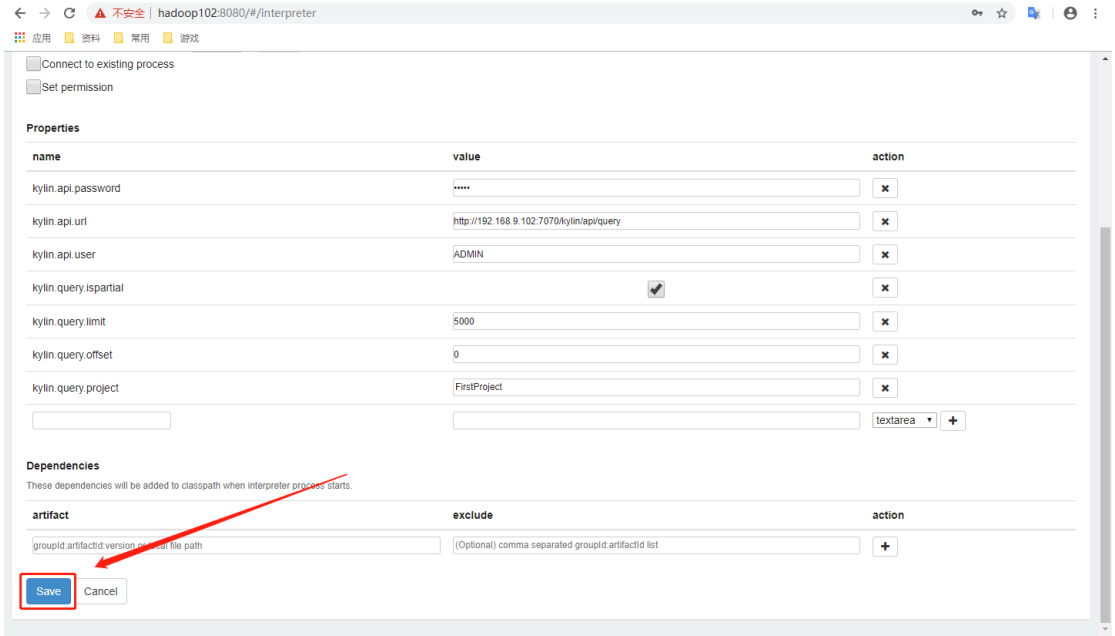
（1）点击右上角 anonymous 选择 Interpreter



（2）搜索 Kylin 插件并修改相应的配置



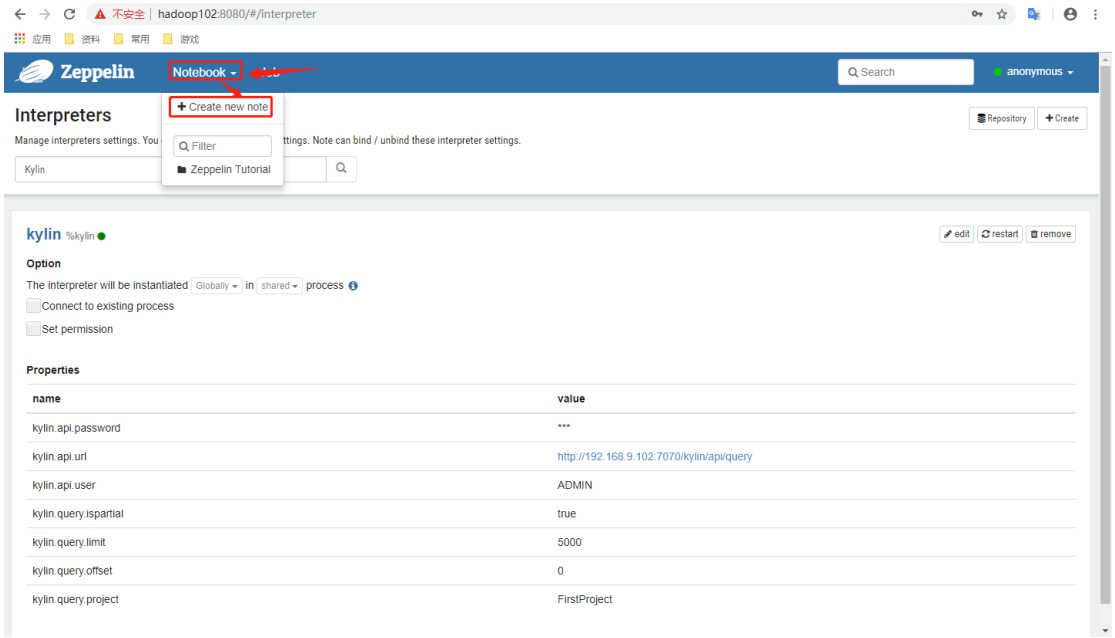
（3）修改完成点击 Save 完成



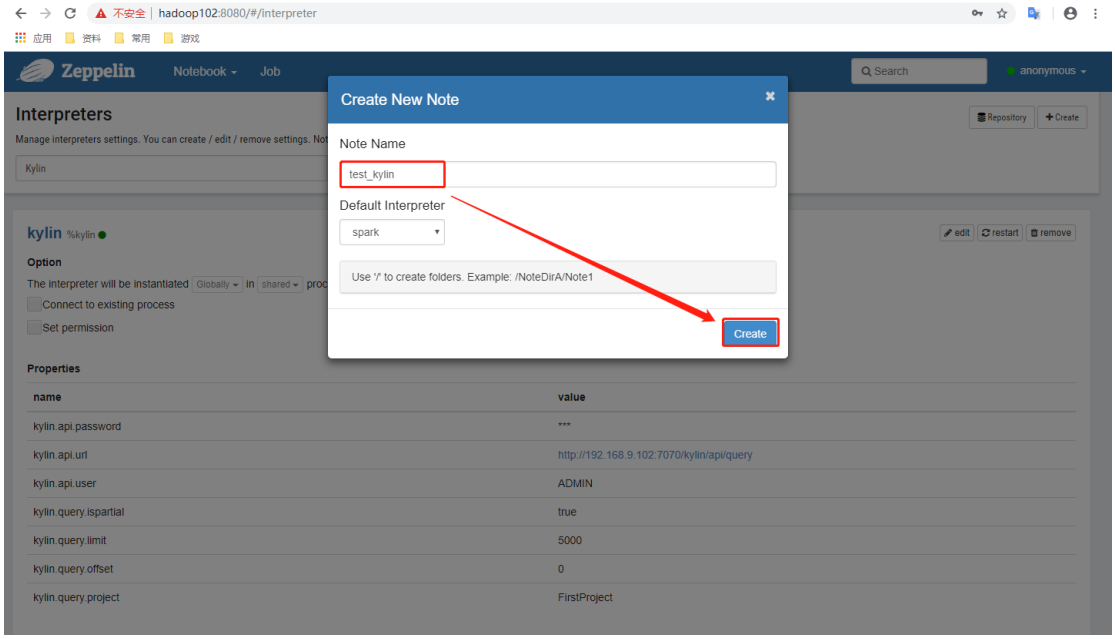
3）案例实操

需求：查询员工详细信息，并使用各种图表进行展示

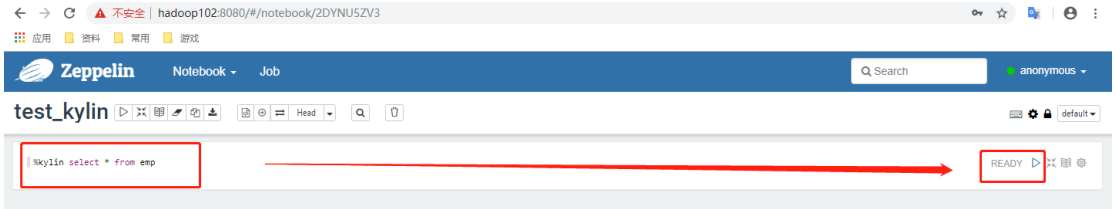
（1）点击 Notebook 创建新的 note



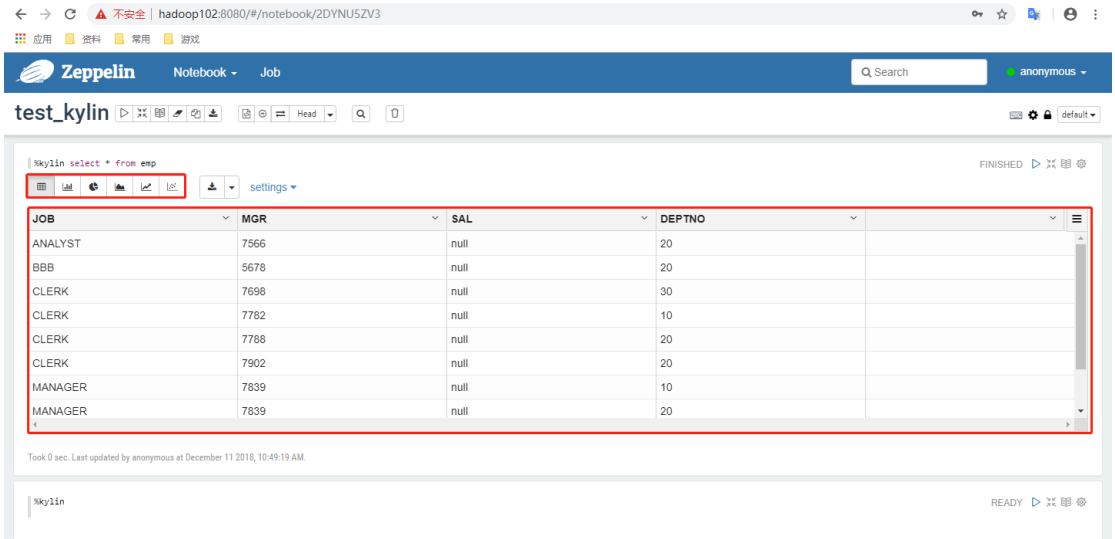
（2）填写 Note Name 点击 Create



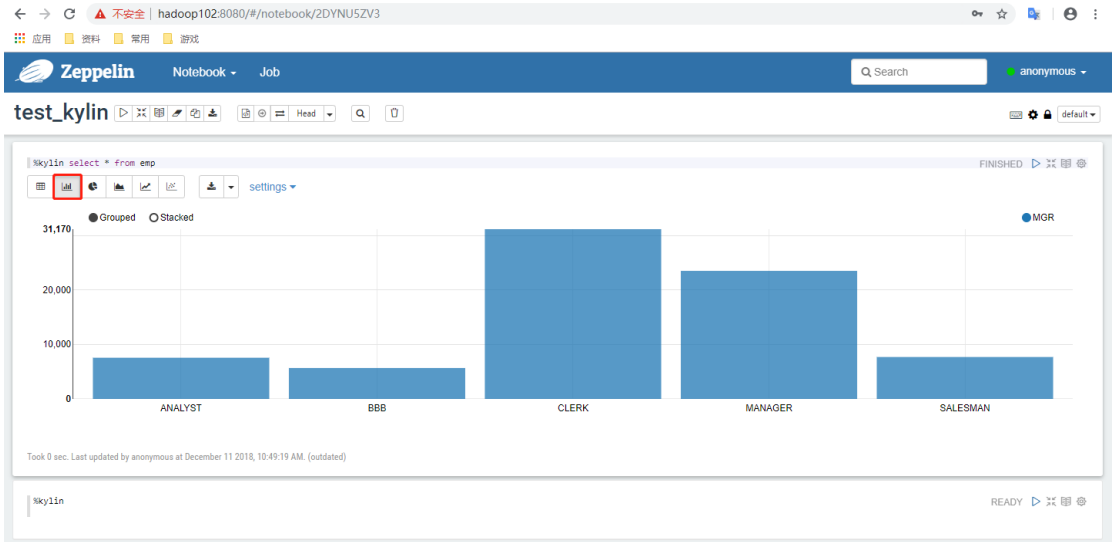
（3）执行查询

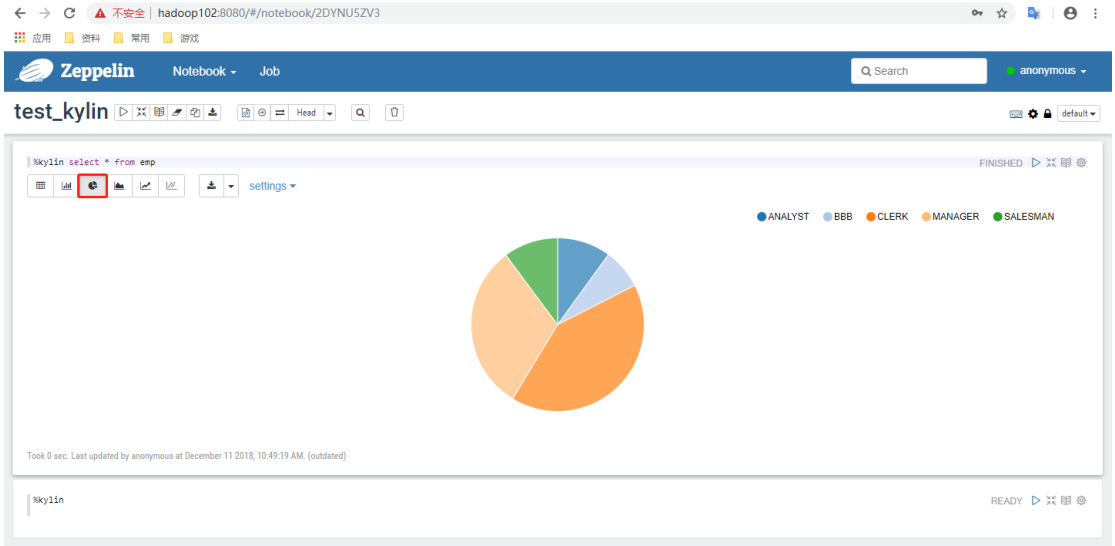


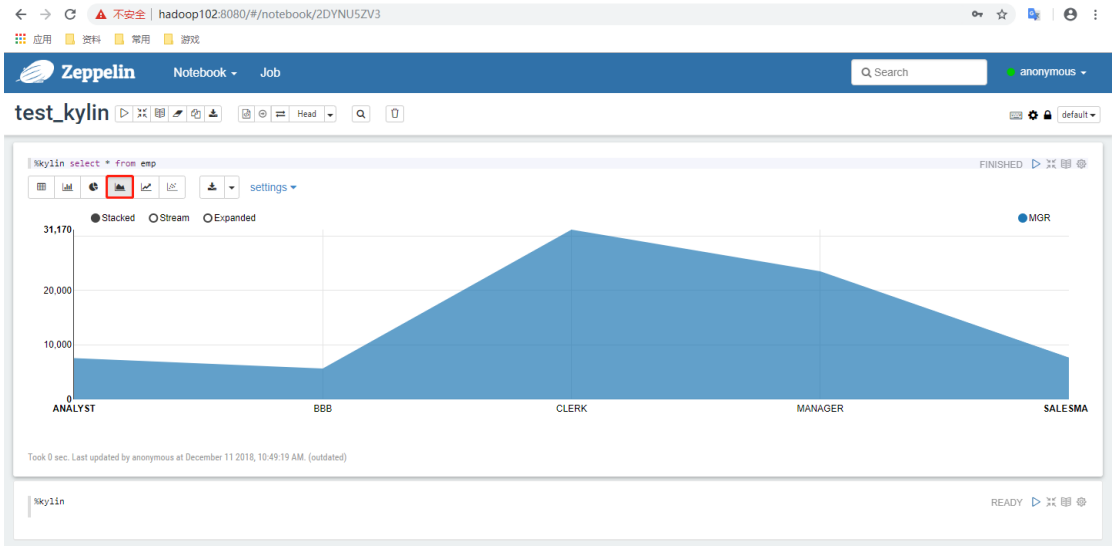
（4）结果展示

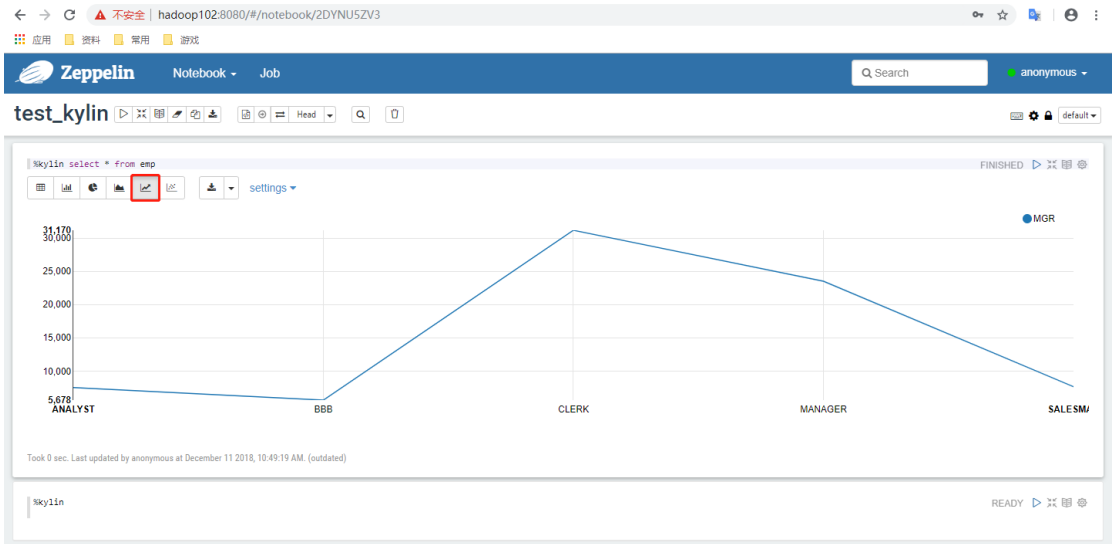


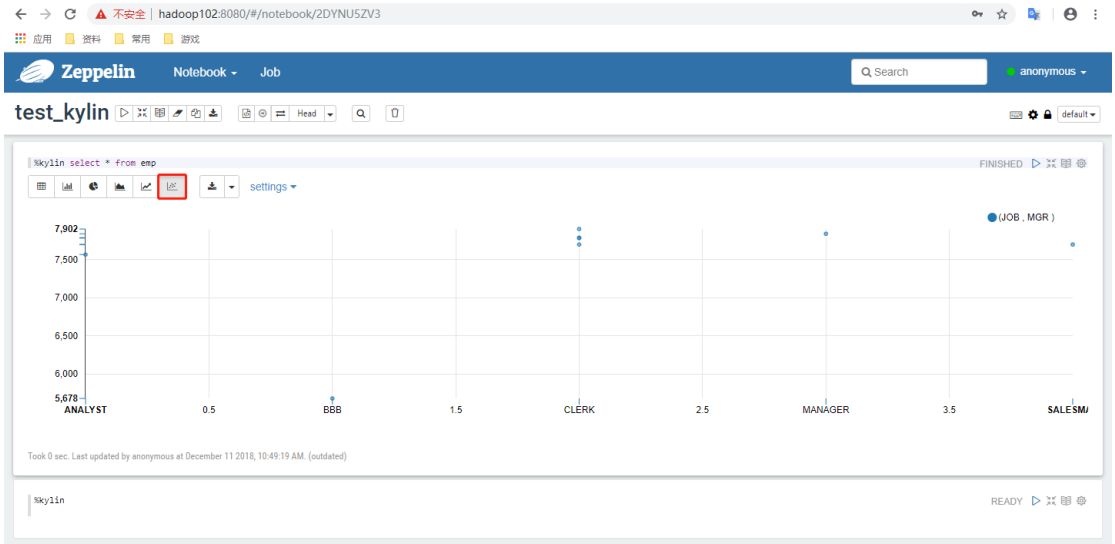
（5）其他图表格式




hadoop102:8080/#/notebook/2DYNU5ZV3
Zeppelin
Notebook
Job
Search
anonymous
test_kylin
%kylin select * from emp
FINISHED
settings
ANALYST
BBB
CLERK
MANAGER
SALESMAN
Took 0 sec. Last updated by anonymous at December 11 2018, 10:49:19 AM. (outdated)
%kylin
READY


hadoop102:8080/#/notebook/2DYNU5ZV3
Zeppelin
Notebook
Job
Search
anonymous
test_kylin
%kylin select * from emp
FINISHED
settings
Stacked
Stream
Expanded
MGR
31,170
20,000
10,000
0
ANALYST
BBB
CLERK
MANAGER
SALESMA
Took 0 sec. Last updated by anonymous at December 11 2018, 10:49:19 AM. (outdated)
%kylin
READY


hadoop102:8080/#/notebook/2DYNU5ZV3
Zeppelin
Notebook
Job
Search
anonymous
test_kylin
%kylin select * from emp
FINISHED
settings
MGR
31,170
30,000
25,000
20,000
15,000
10,000
5,678
ANALYST
BBB
CLERK
MANAGER
SALESM
Took 0 sec. Last updated by anonymous at December 11 2018, 10:49:19 AM. (outdated)
%kylin
READY


hadoop102:8080/#/notebook/2DYNU5ZV3
Zeppelin
Notebook
Job
Search
anonymous
test_kylin
Head
default
%kylin select * from emp
FINISHED
settings
(JOB , MGR)
7,902
7,500
7,000
6,500
6,000
5,678
ANALYST
0.5
BBB
1.5
CLERK
2.5
MANAGER
3.5
Took 0 sec. Last updated by anonymous at December 11 2018, 10:49:19 AM. (outdated)
%kylin
READY