

# 凑齐龙珠，召唤kylin

李易非 熊苗 林新迪

在本次作业中，我们需要利用 kylin 平台进行两个 sql 语句的查询，在我们的后续实验中发现两个语句以及表格需要略微修改为

```
1 select avg(turnoverratio) from stocks where trade-settlement>=0 group by
   name, valid_date;
2 select avg(changepercent), sum(volume) from stocks inner join sales on
   stocks.code = sales.code and stocks.valid_date=sales.valid_date group by
   stocks.valid_date
```

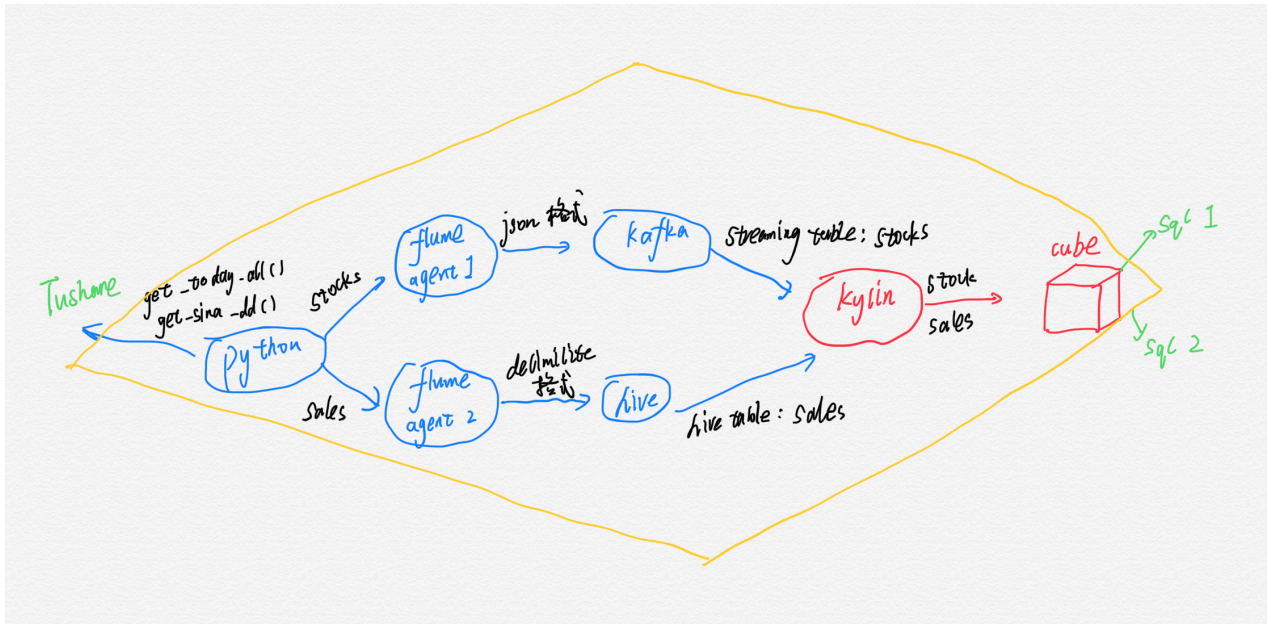
首先 `date` 为 sql 保留字，但是 kylin 在接受 kafka 流数据创建 streaming table 时并不对列名有这些要求，这会导致上述 sql 语句不可能被执行，所以我们将 `date` 改为 `valid_date`

其次，kylin 系统并不支持笛卡尔积，我们在创建 lookup table 的过程中也仅能设置 inner join / left join，所以我们需要把 sql 语句的笛卡尔积修改为 **a inner join b on condition** 的形式；

第三，作业要求中，应该是需要 sales, stocks 两张表均以 python -> flume -> kafka -> kylin 的形式输入 kylin 之中（kafka 流将在 kylin 中创建 streaming table），而在 kylin 中对 *streaming table* 和 *hive table* 是区别对待的，具体的说，**streaming table 的 lookup table 必须是 hive table**，所以我们必须将其中一个表格直接传入 hive，我们的解决方案为把 **stocks 作为 streaming table, sales 作为 hive table**；

最后，`ts.get_today_all()` 作为 stocks 表，`ts.get_sina_dd()` 作为 sales 表，因为 `ts.get_sina_dd()` 需要指定股票代码 `code`，我们在这里采用了将 `ts.get_today_all()` 中的所有 `code` 都运行一遍上述语句的方法，但问题在于 `ts.get_sina_dd()` 的 **primary key 并不是 code + date 而是 code + date + time**，而在 kylin 中创建 lookup table，需要 lookup 的两个维度 `code`，`date` 是主键，所以直接将 `ts.get_sina_dd()` 生成的表作为 lookup table 是 **不可能的**。由于上述原因，我们仅保留每个 `ts.get_sina_dd()` 返回的第一条记录插入表格，让 `code + date` 成为 sales 表的 primary key；

讨论清楚了上述问题，本次作业的逻辑就变得清晰了：



接下来，我们将介绍本次作业需要的诸多环境的安装，由于 kylin 背靠 hdfs, hbase, hive, spark; kafka 与 hbase 又需要 zookeeper 的支持，我们在配置环境过程中也遇到了诸多不便，详细记录如下。

## 1、环境配置

最终我们在 host0 + slave[1-4] 的集群环境配置如下：

- hadoop 2.7.6
- hbase 1.2.11
- hive 1.2.2
- kafka 2.1.1 (compatible with Scala 2.12)
- zookeeper 3.5.5
- kylin 2.6.2 (compatible with hbase 1.x)
- python 2.7

### zookeeper 集群环境配置

特别注意的一点在于，之前服务器环境中已有 *hbase*，*hbase* 也是要依赖 zookeeper 的，但是 *hbase* 发行版中集成了一个 zookeeper，现在我们该换自己的 zookeeper，否则二者会产生冲突，方法也很简单，只需要在 `hbase-site.xml` 中配置 `HBASE_MANAGES_ZK=false` 即可；

主要参考 [官方文档](#)

第一步，下载并解压 zookeeper

```
1 wget http://mirror.bit.edu.cn/apache/zookeeper/zookeeper-3.5.5/apache-
  zookeeper-3.5.5-bin.tar.gz
2 tar -xzvf apache-zookeeper-3.5.5-bin.tar.gz
3 mv apache-zookeeper-3.5.5-bin /usr/local/zookeeper
4 export ZOOKEEPER_HOME=/usr/local/zookeeper
5 cd $ZOOKEEPER_HOME
```

第二步，修改 zookeeper 配置 *conf/zoo.zfg*

```
1 tickTime=2000
2 dataDir=/var/lib/zookeeper
3 clientPort=2181
4 initLimit=5
5 syncLimit=2
6 server.1=host0:2888:3888
7 server.2=slave1:2888:3888
8 server.3=slave2:2888:3888
9 server.4=slave3:2888:3888
10 server.5=slave4:2888:3888
```

第三步，通过 *scp* 将整个 zookeeper 文件夹分发给 *slave[1-4]*

第四步，在各个服务器上创建 *dataDir* 指定的路径 */var/lib/zookeeper*，并在其中依据 *zoo.cfg* 设置的 *server id* 创建 *myid* 文件，比如 *host0* 可通过 `echo 1 > /var/lib/zookeeper/myid` 的方式创建 *myid* 文件；

第五步，启动 zookeeper 服务，检查是否正常，需要我们去各个服务器调用 */usr/local/zookeeper/bin/zkServer.sh start*，之后可通过 */usr/local/zookeeper/zkServer.sh status* 查看当前 *server* 在 zookeeper 集群中是 *follower* / *leader*

```
1 [root@host0 bin]$ ./zkServer.sh status
2 ZooKeeper JMX enabled by default
3 Using config: /usr/local/zookeeper/bin/../conf/zoo.cfg
4 Client port found: 2181. Client address: localhost.
5 Mode: leader
6
7 root@slave1:~# /usr/local/zookeeper/bin/zkServer.sh status
8 ZooKeeper JMX enabled by default
9 Using config: /usr/local/zookeeper/bin/../conf/zoo.cfg
10 Client port found: 2181. Client address: localhost.
11 Mode: follower
```

## kafka 集群配置

主要参考 [官方文档](#)

第一步，下载并解压 kafka

```
1 wget http://mirrors.tuna.tsinghua.edu.cn/apache/kafka/2.1.1/kafka_2.12-2.1.1.tgz
2 tar -xzvf kafka_2.12-2.1.1.tgz
3 mv kafka_2.12-2.1.1 /usr/local/kafka
4 export KAFKA_HOME=/usr/local/kafka
5 cd $KAFKA_HOME
```

第二步，将 `/usr/local/kafka` 通过 `scp` 命令发送给各个 slave 服务器，修改 `conf/server.properties`，设置正确的 `broker_id`，比如 `host0` 的 `broker id` 为 0，`slave1` 的 `broker id` 为 1，为了方便后续的 topic 管理，**强烈建议** 在其中也加入 `delete.topic.enable=true` 字段；

第三步，进行 topic 的创建，并运行 producer 和 consumer 检查 kafka 是否正常工作；

```
1 [root@host0 kafka]$ bin/kafka-topics.sh --create --zookeeper
  localhost:2181 --replication-factor 3 --partitions 1 --topic stocks
2 Created topic "stocks".
3 [root@host0 kafka]$ bin/kafka-topics.sh --describe --zookeeper
  localhost:2181 --topic stocks
4 Topic:stocks PartitionCount:1 ReplicationFactor:3 Configs:
5   Topic: stocks Partition: 0 Leader: 2 Replicas: 2,3,4 Isr: 2,3,4
6 [root@host0 kafka]$ bin/kafka-console-producer.sh --broker-list
  localhost:9092 --topic stocks
7 >hello, it is yeef
8 >Nice to meet you all!
9 ^C
10 [root@host0 kafka]$ bin/kafka-console-consumer.sh --bootstrap-server
  localhost:9092 --from-beginning --topic stocks
11 hello, it is yeef
12 Nice to meet you all!
13 ^CProcessed a total of 2 messages
```

```
root@slave4: ~ (ssh)
Connection to slave4 closed.
[root@host0 kafka]$ bin/kafka-console-producer.sh --broker-list localhost:9092 --topic kafka-test
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/local/kafka/libs/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
>hello
>I am fine
>thank you
>it is good
>I love you
>I love you too
>hhhh\
>this is a good
>I love China
>

root@slave4: /usr/local/kafka (ssh)
Welcome to Ubuntu 16.04 LTS (GNU/Linux 4.4.0-21-generic x86_64)

 * Documentation:  https://help.ubuntu.com/
New release '18.04.2 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Sat Jun  8 10:09:27 2019 from 10.173.32.17
root@slave4:~# cd /usr/local/kafka/
root@slave4:/usr/local/kafka# bin/kafka-server-start.sh -daemon config/server.properties
root@slave4:/usr/local/kafka# exit
logout
Connection to slave4 closed.
[root@host0 kafka]$ ls
bin  config  libs  LICENSE  logs  NOTICE  site-docs
[root@host0 kafka]$ bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --from-beginning --topic kafka-test
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/local/kafka/libs/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
hello
I am fine
thank you
it is good
I love you
I love you too
hhhh\
this is a good
I love China
[]
```

## kylin 环境配置

主要参考 [官方文档](#)。

在安装 kylin 之前，一定要依据官方文档要求，检查 hbase, hadoop, hive 等是否版本兼容

- Hadoop: 2.7+, 3.1+ (since v2.5)
- Hive: 0.13 - 1.2.1+
- HBase: 1.1+, 2.0 (since v2.5)
- Spark (可选) 2.3.0+
- Kafka (可选) 1.0.0+ (since v2.5)
- JDK: 1.8+ (since v2.5)
- OS: Linux only, CentOS 6.5+ or Ubuntu 16.0.4+

我们初期卡了很久的一个问题其实就在于 hive 版本过高, purge 掉 hive 的所有 hdfs 存储, mysql metastore 存储, 重新下载 hive 1.2.2 问题解决;

第一步, 下载 kylin 并解压

```
1 wget http://mirror.bit.edu.cn/apache/kylin/apache-kylin-2.6.2/apache-kylin-2.6.2-bin-hbase1x.tar.gz
2 tar -xzvf apache-kylin-2.6.2-bin-hbase1x.tar.gz
3 mv apache-kylin-2.6.2-bin-hbase1x /usr/local/kylin
4 export KYLIN_HOME=/usr/local/kylin
5 cd $KYLIN_HOME
```

第二步, 启动 hadoop, zookeeper, kafka, hbase, kylin (顺序有要求)

```
1 [root@host0 bin]$ jps
2 3601 ResourceManager
3 23633 Jps
4 3891 JobHistoryServer
5 19476 QuorumPeerMain
6 3430 SecondaryNameNode
7 3208 NameNode
8 45963 HMaster
9 23452 RunJar
```

23452 RunJar jps 进程即为 kylin 进程;

```

2019-06-07 21:28:50,306 INFO [main] zookeeper.ZooKeeper:438 : Initiating client connection, connectString=host0:2181,slave1:2181,slave2:2181,slave3:2181,slave4:2181 sessionTimeout=90000 watcher=hconnection-0x5b0680870x0, quorum=host0:2181,slave1:2181,slave2:2181,slave3:2181,slave4:2181, baseZNode=/hbase
2019-06-07 21:28:50,323 INFO [main-SendThread(slave1:2181)] zookeeper.ClientCnxn:975 : Opening socket connection to server slave1/10.173.32.12:2181. Will not attempt to authenticate using SASL (unknown error)
2019-06-07 21:28:50,329 INFO [main-SendThread(slave1:2181)] zookeeper.ClientCnxn:852 : Socket connection established to slave1/10.173.32.12:2181, initiating session
2019-06-07 21:28:50,337 INFO [main-SendThread(slave1:2181)] zookeeper.ClientCnxn:1235 : Session establishment complete on server slave1/10.173.32.12:2181, sessionId = 0x200001498f50003, negotiated timeout = 40000
2019-06-07 21:28:50,650 DEBUG [main] hbase.HBaseConnection:306 : HTable 'kylin_metadata' already exists
2019-06-07 21:28:50,898 DEBUG [main] hbase.HBaseConnection:180 : Using the working dir FS for HBase: hdfs://host0:9000
2019-06-07 21:28:50,898 INFO [main] hbase.HBaseConnection:257 : connection is null or closed, creating a new one
2019-06-07 21:28:50,900 INFO [main] zookeeper.RecoverableZooKeeper:120 : Process identifier=hconnection-0xce5a68e connecting to ZooKeeper ensemble=host0:2181,slave1:2181,slave2:2181,slave3:2181,slave4:2181
2019-06-07 21:28:50,900 INFO [main] zookeeper.ZooKeeper:438 : Initiating client connection, connectString=host0:2181,slave1:2181,slave2:2181,slave3:2181,slave4:2181 sessionTimeout=90000 watcher=hconnection-0xce5a68e0x0, quorum=host0:2181,slave1:2181,slave2:2181,slave3:2181,slave4:2181, baseZNode=/hbase
2019-06-07 21:28:50,901 INFO [main-SendThread(slave1:2181)] zookeeper.ClientCnxn:975 : Opening socket connection to server slave1/10.173.32.12:2181. Will not attempt to authenticate using SASL (unknown error)
2019-06-07 21:28:50,902 INFO [main-SendThread(slave1:2181)] zookeeper.ClientCnxn:852 : Socket connection established to slave1/10.173.32.12:2181, initiating session
2019-06-07 21:28:50,907 INFO [main-SendThread(slave1:2181)] zookeeper.ClientCnxn:1235 : Session establishment complete on server slave1/10.173.32.12:2181, sessionId = 0x200001498f50004, negotiated timeout = 40000
2019-06-07 21:28:50,926 INFO [close-hbase-conn] hbase.HBaseConnection:136 : Closing HBase connections...
2019-06-07 21:28:50,927 INFO [close-hbase-conn] client.ConnectionManager$HConnectionImplementation:1726 : Closing zookeeper sessionId=0x200001498f50004
2019-06-07 21:28:50,930 INFO [close-hbase-conn] zookeeper.ZooKeeper:684 : Session: 0x200001498f50004 closed
2019-06-07 21:28:50,930 INFO [main-EventThread] zookeeper.ClientCnxn:512 : EventThread shut down
2019-06-07 21:28:50,941 INFO [close-hbase-conn] client.ConnectionManager$HConnectionImplementation:2155 : Closing master protocol: MasterService
2019-06-07 21:28:50,941 INFO [close-hbase-conn] client.ConnectionManager$HConnectionImplementation:1726 : Closing zookeeper sessionId=0x200001498f50003
2019-06-07 21:28:50,945 INFO [close-hbase-conn] zookeeper.ZooKeeper:684 : Session: 0x200001498f50003 closed
2019-06-07 21:28:50,945 INFO [main-EventThread] zookeeper.ClientCnxn:512 : EventThread shut down

A new Kylin instance is started by root. To stop it, run 'kylin.sh stop'
Check the log at /usr/local/kylin/logs/kylin.log
Web UI is at http://host0:7070/kylin

```

我们可以访问 `host0:7070/kylin` (host0 要替换为公网 ip) 进行 kylin 网页端访问，默认 user 为 ADMIN, 密码为 KYLIN



Sign in

Username

Password

Login Issue?

接着，我们可以运行官方提供的 `sample.sh` 来初步感受 cube 的建立，其中也有一个 streaming table 的例子，`[root@host0 bin]$ ./sample.sh`，这将创建一个 `learn_kylin` 的 project，其中有两个 cube 的例子，一个是普通的 cube 的搭建，另一个为 streaming cube 的构建；

```

sample_cube/metadata to kylin_metadatahbase
Metadata restored from /usr/local/kylin/sample_cube/metadata
2019-06-07 21:43:05,135 INFO [close-hbase-conn] hbase.HBaseConnection:136 : Closing HBase connections...
2019-06-07 21:43:05,135 INFO [close-hbase-conn] client.ConnectionManager$HConnectionImplementation:2155 : Closing master protocol: MasterService
2019-06-07 21:43:05,136 INFO [close-hbase-conn] client.ConnectionManager$HConnectionImplementation:1726 : Closing zookeeper sessionId=0x100001b0cf70004
2019-06-07 21:43:05,141 INFO [close-hbase-conn] zookeeper.ZooKeeper:684 : Session: 0x100001b0cf70004 closed
2019-06-07 21:43:05,141 INFO [main-EventThread] zookeeper.ClientCnxn:512 : EventThread shut down
Sample cube is created successfully in project 'learn_kylin'.
Restart Kylin Server or click Web UI => System Tab => Reload Metadata to take effect

```

Jobs Slow Queries

Cube Name:  Jobs in: LAST ONE WEEK | ALL | NEW | PENDING | RUNNING | STOPPED | FINISHED | ERROR | DISCARDED

Job Name	Cube	Progress	Last Modified Time	Duration	Actions
BUILD CUBE - kylin_sales_cube - 20120101000000_20150101000000 - CST 2019-06-07 21:45:43	kylin_sales_cube	100%	2019-06-07 13:56:38 UTC	10.60 mins	Action

如果可以成功 build 上述 sample cube, 并在 insight 页面进行简单的 sql 查询, 我们就可以确定 kylin 的环境搭建成功👏

## 2、作业流程

### 2.1 stock streaming table

在本次作业中, 我们将 stock 作为 kylin 中的 streaming table, 通过 python->thrift->flume->kafka->kylin 的逻辑流程将其输入 kylin。

首先, 我们需要在 kafka 中创建 stocks 对应的 topic, 同时开启 consumer 进程, 确认我们的信息正常

```

1 bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 3 --partitions 1 --topic stocks
2 [root@host0 kafka]$ bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --from-beginning --topic stocks

```

接着, 开启 flume 进程, conf 文件如下:

```

1 # 给agent命名
2 a1.sources = r1
3 a1.sinks = k1
4 a1.channels = c1
5
6 # 配置source
7 a1.sources.r1.type=thrift
8 a1.sources.r1.bind=0.0.0.0

```













Streaming Table And Cluster Info

**Kafka Setting**

Topic

**Cluster-1**

ID	Host	Port	Actions
0	host0	9092	 
1	slave1	9092	 
2	slave2	9092	 
3	slave3	9092	 
4	slave4	9092	 

[+ Broker](#)

[+ Cluster](#)

**Parser Setting**

Parser Name

[← Prev](#) [Submit](#)

Table Schema:STOCK\_NEW

Columns | Extend Information | Streaming Cluster | Access | Snapshot [✕ Unload Table](#)

Columns

ID	Name	Data Type	Cardinality	Comment
1	CODE	varchar(256)		
2	NAME	varchar(256)		
3	CHANGEPERCENT	decimal(19,4)		
4	TRADE	decimal(19,4)		
5	OPEN	decimal(19,4)		
6	HIGH	decimal(19,4)		
7	LOW	decimal(19,4)		
8	SETTLEMENT	decimal(19,4)		
9	VOLUME	integer		
10	TURNROVERRATIO	decimal(19,4)		
11	AMOUNT	integer		
12	PER	decimal(19,4)		
13	PB	decimal(19,4)		
14	MKTCAP	decimal(19,4)		
15	NMC	decimal(19,4)		
16	VALID_DATE	date		
17	QUERY_TIME	timestamp		

## 2.2 sales hive table

在本次作业中，我们将 *sales* 作为 kylin 中的 hive table，并在后续过程中作为 *stocks* 的 lookup table，通过 python->thrift->flume->hive->kylin 的逻辑流程将其输入 kylin。

传入 hive 的过程如上次作业描述的流程，不做赘述；

### Table Schema:SALES

ID	Name	Data Type	Cardinality	Comment
1	CODE	varchar(256)	55	
2	NAME	varchar(256)	58	
3	TRADE_TIME	varchar(256)	17	
4	PRICE	double	54	
5	VOLUME	integer	58	
6	PREPRICE	double	57	
7	TYPE	varchar(256)	2	
8	VALID_DATE	date	1	

## 2.3 定义 data model

在本次作业中，我们有两个 sql 语句需要执行，第一个语句仅需要在 *stocks* 表上，第二个语句需要把 *sales* 表作为 lookup table，基于上述需求，我们构建一个 *stocks left join sales* 的 data model

The screenshot shows the 'Model Designer' interface with a progress bar at the top indicating the current step is 'Data Model'. Below the progress bar, the 'Fact Table' is identified as 'STOCKS\_3'. A 'Join Tables' section contains a table with the following data:

ID	Table Alias	Table Name	Table Kind	Join Type	Join Condition
1	SALES	DEFAULT.SALES	Normal	left	STOCKS_3.CODE = SALES.CODE STOCKS_3.VALID_DATE = SALES.VALID_DATE

将 *code*, *name*, *valid\_date*, *settlement*, *trade* 定义为 dimension，将 *changepercent*, *ratio*, *volume* 定义为 volume

The screenshot shows the 'Model Designer' interface with the progress bar at the 'Dimensions' step. Below the progress bar, a table lists the dimension and measure definitions for the fact table 'STOCKS\_3':

ID	Table Alias	Columns
1	STOCKS_3	["CODE"; "NAME"; "SETTLEMENT"; "VALID_DATE"; "TRADE"]
2	SALES	["CODE"; "VALID_DATE"]

Model Designer

ID	Column
1	STOCKS_3.CHANGEPERCENT
2	STOCKS_3.TURNOVERRATIO
3	STOCKS_3.VOLUME

← Prev Next →

并且按 *valid\_date* 属性进行 partition, 按天分割

Model Designer

Partition

Partition Date Column: STOCKS\_3.VALID\_DATE

Date Format: yyyy-MM-dd

Filter

Filter

## 2.4 定义 cube

基于上述 *stock\_sale\_left\_join\_model*, 我们进行 cube 构建

cube 的 dimension 直接沿用 model 中的 dimension, 并将 lookup table 中的两个 dimension 定义为 derived 类型, 提高 cube 效率

Cube Designer

ID	Name	Table Alias	Type	Column
1	CODE	STOCKS_3	normal	CODE
2	NAME	STOCKS_3	normal	NAME
3	TRADE	STOCKS_3	normal	TRADE
4	SETTLEMENT	STOCKS_3	normal	SETTLEMENT
5	VALID_DATE	STOCKS_3	normal	VALID_DATE
6	CODE	SALES	derived	["CODE"]
7	VALID_DATE	SALES	derived	["VALID_DATE"]

cube 的 measures 有三个 measure 的 sum 以及系统默认的 count measure, 提高 sql 执行效率

Cube Designer

Name	Expression	Parameters	Return Type
_COUNT_	COUNT	__Value:1, Type:constant	bigint
SUM_CHANGEPERCENT	SUM	__Value:STOCKS_3.CHANGEPERCENT, Type:column	decimal(19,4)
SUM_TURNOVERRATIO	SUM	__Value:STOCKS_3.TURNOVERRATIO, Type:column	decimal(19,4)
SUM_VOLUME	SUM	__Value:STOCKS_3.VOLUME, Type:column	float

在 advanced setting 中，添加聚合组 *trade, settlement*，提高 cube 构建效率，在 rowkey 中，将用于 filter 的条件 *trade, settlement* 放在前面，提高 cube 构建效率

### Aggregation Groups

Visit [aggregation group](#) for more about aggregation group.

ID	Aggregation Groups	Max Dimension Combination: 0
1	Includes	["STOCKS_3.TRADE", "STOCKS_3.SETTLEMENT"]
	Mandatory Dimensions	["STOCKS_3.TRADE", "STOCKS_3.SETTLEMENT"]
	Hierarchy Dimensions	
	Joint Dimensions	

### Rowkeys ⓘ

**Important:** Dimension's position on HBase rowkey is critical for performance. You can drag and drop to adjust the sequence. In short, put filtering dimension before non-filtering dimension, and put high cardinality dimension before low cardinality dimension.

ID	Column	Encoding	Length	Shard By
1	STOCKS_3.TRADE	dict[v1]		false
2	STOCKS_3.SETTLEMENT	dict[v1]		false
3	STOCKS_3.CODE	dict[v1]		false
4	STOCKS_3.NAME	dict[v1]		false
5	STOCKS_3.VALID_DATE	date[v1]		false

cube 构建引擎选择 mapreduce，尽管理论上 spark 要快，但是实践中发现 mapreduce 更快，很可能是因为我们的 spark 配置沿用原始，没有进行 case by case 的调优

### Cube Engine ⓘ

Engine Type : MapReduce

## Cube Designer



Model Name	stock_sale_left_join_model
Cube Name	stock_sale_left_join_cube
Fact Table	DEFAULT.STOCKS_3
Lookup Table	1
Dimensions	7
Measures	4

Description

← Prev

## 2.5 build cube and sql

直接在 action 中选择 build 即可，如果之前一切正常的话，构建在 3min 内即可完成

Job Name	Cube	Progress	Last Modified Time	Duration	Actions
BUILD CUBE - stock_sale_left_join_cube - 0_3703 - CST 2019-06-09 11:46:40	stock_sale_left_join_cube	100%	2019-06-09 03:49:25 UTC	2.53 mins	Action

接着就可以在 sql 页面进行我们的 sql 查询了！

```
1 select avg(turnoverratio) from stocks_3 where trade-settlement>=0 group by name, valid_date
```

New Query | Saved Queries | Query History

```
1 select avg(turnoverratio) from stocks_3 where trade-settlement--0 group by name, valid_date
```

Tips: Ctrl+Shift+Space or Alt+Space(Windows), Command+Option+Space(Mac) to list keywords in query box.

Project: tushare ✓ LIMIT 50000 Submit

### Results

1 ✓ x | Status: All

Query String Start Time: 2019-06-09 03:52:14 UTC Duration: 2.26s Rerun Save

Status: Success

Project: tushare

Cubes: CUBE[name=stock\_sale\_left\_join...]

Results (486)

Visualization | Export

EXPR\$0
4.5109
0
1.0718
1.0264
2.9131
1.0571
1.0571
1.0571

```
1 select avg(changepercent), sum(volume) from stocks_3 inner join sales on stocks_3.code = sales.code and stocks_3.valid_date=sales.valid_date group by stocks_3.valid_date
```

The screenshot shows a query execution interface. At the top, there are tabs for 'New Query', 'Saved Queries', and 'Query History'. The query editor contains the following SQL query:

```
1 select avg(changepercent), sum(volume) from stocks_3 left join sales on stocks_3.code = sales.code and stocks_3.valid_date=sales.valid_date group by stocks_3.valid_date
```

Below the query editor, there is a 'Project' dropdown set to 'tushare', a 'LIMIT' input field set to '50000', and a 'Submit' button. A tip below the query editor reads: 'Tips: Ctrl+Shift+Space or Alt+Space(Windows), Command+Option+Space(Mac) to list keywords in query box.'

The 'Results' section shows a status of 'Success' and a 'Project' of 'tushare'. The 'Query String' is displayed, along with 'Start Time: 2019-06-09 03:54:15 UTC' and 'Duration: 0.29s'. There are 'Rerun' and 'Save' buttons. The 'Results (1)' section shows a table with two columns: 'EXPR\$0' and 'EXPR\$1'. The first row contains the values '-2.4324158790...' and '41280984515'. There are also 'Visualization' and 'Export' buttons.

### 3、问题汇集

#### kylin start 速度相当慢，且集群极易集体宕机

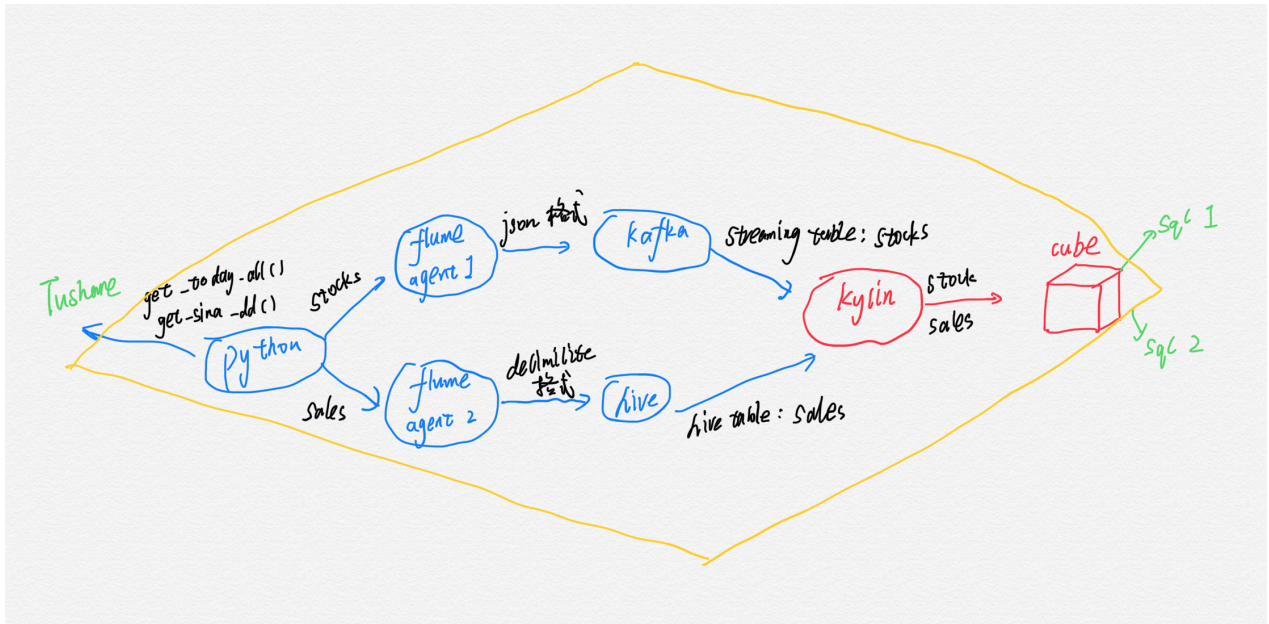
kylin 官方文档中提到集群需要至少 4核 32g 的配置，我一开始不信邪，还是 2核 4g 瞎玩，后来把 host0 改成了 4核16g，其余 slave 还是维持原有配置，勉强可以完成本次作业；

#### kylin web 端打不开

翻 log 查报错，发现是 tomcat 出的问题，但最终发现其实是 hive 版本不对导致的问题，人为 purge 掉所有之前 hive 在 hdfs, mysql 中的记录，重新下载 hive 1.2.2 问题成功解决；

#### kylin 并不支持 streaming table lookup streaming table

一开始我们将 stocks, sales 都定义为了 kafka streaming table, 但发现 streaming table 仅支持和 hive table 进行 lookup, 所以最终将 sales 改为输入 hive 的逻辑



### date 是 mysql 关键字

这个错误很坑，因为完全看不懂他的报错是为什么，后来灵光乍现，发现了这个问题，把 date 改为了 valid\_date，还是希望 kylin 可以做一个 sql 保留字检查，也提了一个 jira issue。