Spark企业级大数据项目实战 第5课

DATAGURU专业数据分析社区

法律声明

【声明】本视频和幻灯片为炼数成金网络课程的教学资料，所有资料只能在课程内使用，不得在课程以外范围散播，违者将可能被追究法律和经济责任。

课程详情访问炼数成金培训网站

http://edu.dataguru.cn

炼数成金逆向收费式网络课程

- Dataguru（炼数成金）是专业数据分析网站，提供教育，媒体，内容，社区，出版，数据分析业务等服务。我们的课程采用新兴的互联网教育形式，独创地发展了逆向收费式网络培训课程模式。既继承传统教育重学习氛围，重竞争压力的特点，同时又发挥互联网的威力打破时空限制，把天南地北志同道合的朋友组织在一起交流学习，使到原先孤立的学习个体组合成有组织的探索力量。并且把原先动辄成千上万的学习成本，直线下降至百元范围，造福大众。我们的目标是：低成本传播高价值知识，构架中国第一的网上知识流转阵地。
- 关于逆向收费式网络的详情，请看我们的培训网站 http://edu.dataguru.cn

本课内容

- Hbase基础
- kafka每条消息的partition、offset等获取
- Hbase实现Exactly-once语义
- Sql on Hbase（Hive、Spark、Phoenix）
- Phoenix整合CDH（源码修改、编译）
- Phoenix整合Spark、Phoenix Query Server（JDBC）

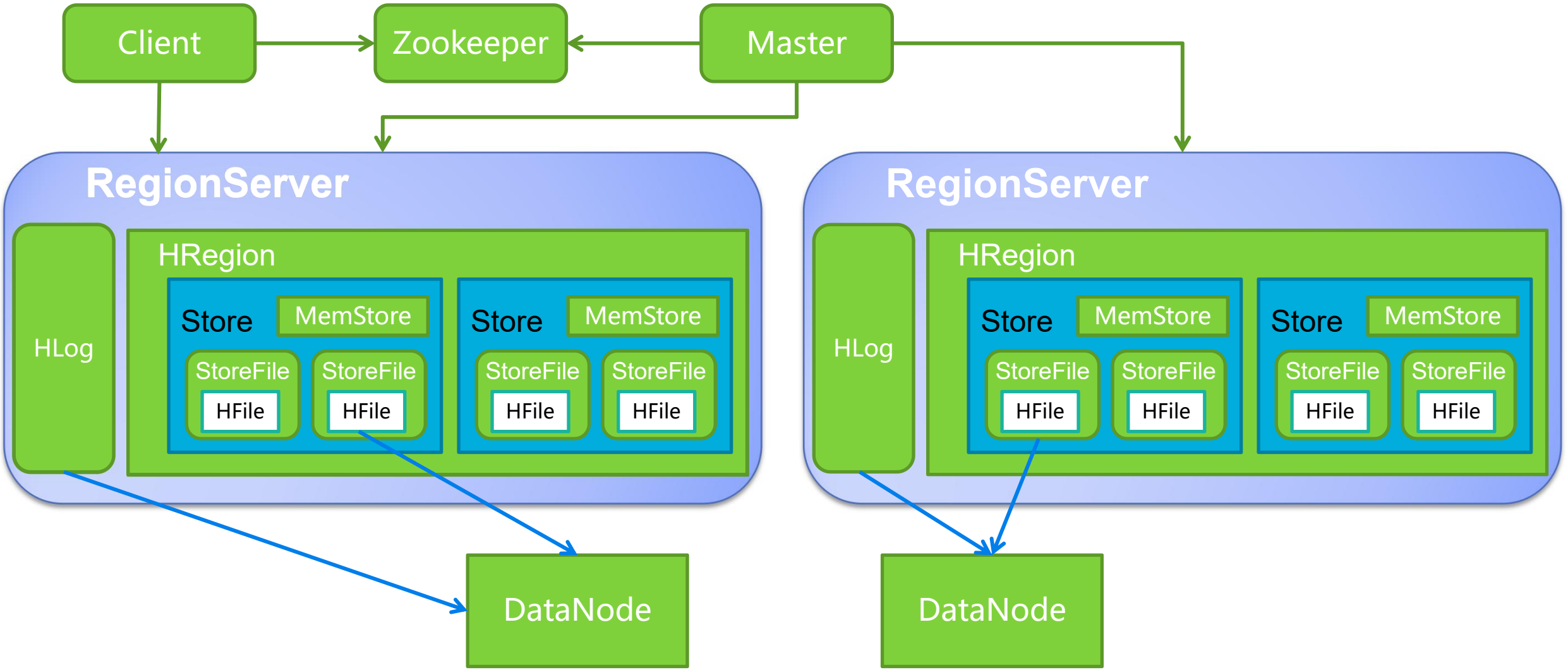
Hbase是什么？

1. HBase是一个分布式的、面向列的开源数据库
2. 源于 Fay Chang 所撰写的Google论文“Bigtable：一个结构化数据的分布式存储系统”
3. 适合于非结构化数据存储的数据库
4. 数据存储于HDFS之上， 有很好的备份机制。
5. 线性扩展

Hbase集群的角色

Hbase集群是主从架构。
主：一个或多个HMaster
从：多个RegionServer

Hbase架构图


Client
Zookeeper
Master
RegionServer
HLog
HRegion
Store
MemStore
StoreFile
HFile
Store
MemStore
StoreFile
HFile
StoreFile
HFile
RegionServer
HLog
HRegion
Store
MemStore
StoreFile
HFile
StoreFile
HFile
Store
MemStore
StoreFile
HFile
StoreFile
HFile
DataNode
DataNode

Hbase组件功能

Zookeeper

1. 保证HMaster节点的HA （ephemeral node）
2. 保存RegionServer节点的信息， 监控RegionServer的上下线， 异常宕机等 （ephemeral node）
3. 维护Hbase的元数据表的位置信息

HMaster

1. DDL操作
2. 记录region在哪台region server上， 负责region的分配和负载均衡。
3. 负责split后新的region的分配。
4. 新的regionserver上线， 负责region的迁移。
5. regionserver下线， 负责失效regionserver上region的迁移

HMaser失效， 不会影响数据的读写。

RegionServer

1. 负责Client向HDFS写数据
2. 管理region， region 的split等。

Zookeeper安装

Hbase安装

Hbase数据模型

表名： testorder

rowkey	timestamp	列族：info			列族：orders	
		name	age	sex	orderid	price
00001	t1	xiao1	22	男	02018010022	21
00001	t2	xiao11	23	男	02018010066	34
00001	t3	xiao22	24	男	02018010077	56
00001	t4	xiao134	25	男	02018010088	78
00002	t3	xiao2	34	女	02018010055	13
00003	t4	xiao3	34	男	02018010065	16

NameSpace
RowKey
Column Family(列族) / Column
TimeStamp / Cell

Hbase数据模型

NameSpace:

可以理解成关系型数据库中的用户、hive中的数据库。

访问执行NameSpace下的表： 名称空间: 表名， 比如default:test 就是default名称空间下的test表

Hbase默认有两个名称空间，分别是hbase（存放hbase元数据表）和default（默认）。

RowKey：

可以类比关系数据库中的主键。

rowkey的最大长度是64kb。

数据存储时候， 按照rowkey的字典序存储。

可能会出现存储热点。

Column Family（列族）：

一个列族可以管理很多列。 列表示为 列族名:列名， 如info:age , 表示info这个列族下的age列。

在建表时， 必须要先指定列族，但是不需要指定列名。 也就是说， 列族是表schema的一部分， 但是列不是schema的部分， 列可以在插入数据时动态添加。

create 'test2','info1','info2','info3'

Hbase数据模型

TimeStamp、Cell：

Cell： 通过rowkey和列唯一确定一个Cell。 一个Cell保存着这个rowkey的多个版本， 如下图所示。
TimeStamp：每个版本通过timestamp时间戳唯一标识。

Hbase提供两种数据版本管理方式： 1. 保留N个版本。 2. 数据的保留时间。

create 'spark_kafka_offsets', {NAME=>'offsets', TTL=>2592000} 保留7天的数据

create 'test1',{NAME=>'info1',VERSIONS=>3},{NAME=>'info2',VERSIONS=>2}

格式：create '表名',{NAME='列族1',VERSIONS=版本号},{NAME='列族2',VERSIONS=版本号}

rowkey	timestamp	列族：info			列族：orders	
		name	age	sex	orderid	price
00001	t1	xiao1	22	男	02018010022	21
00001	t2	xiao11	23	男	02018010066	34
00001	t3	xiao22	24	男	02018010077	56
00001	t4	xiao134	25	男	02018010088	78
00002	t3	xiao2	34	女	02018010055	13
00003	t4	xiao3	34	男	02018010065	16

rowkey=00001、info为name的Cell

Hbase基本操作

- Hbase Shell
- Java Api

Hbase数据读写流程

数据读流程：
1. Client通过zookeeper定位的Hbase元数据表（hbase:meta）。（老版本还有 --ROOT--表）
2. 从hbase:meta定位数据分布在哪些RegionServer上
3. 从RegionServer， 将内存（MemStore）和磁盘（StoreFile）中的数据合并后，返回Client

数据写流程：
1. Client向Region Server发起写请求（Put、Delete）
2. Region Server首先将数据写入WAL(HLog)中。 顺序写， 没有寻道的时间 600M/s
3. Region Server将数据写入内存(MemStore)
4. Client写成功
5. 内存中的数据定期写入磁盘(StoreFile)， 这个过程就是Flush

Hbase数据读写流程

什么时候触发Flush：
1. 内存(MemStore)中的数据达到阈值(默认64M)， 将内存中的数据写入StoreFile（HDFS）
2. 删除内存和HLog中历史数据
3. 在HLog中打标记点。

数据Compact（合并）、Split：
1. 当一个Region的Store中StoreFile文件数量达到一定阈值（默认4个）， 会触发Compact（合并）操作， 将多个StoreFile合并成一个大的StoreFile。 Compact会对Cell的历史版本进行删除和数据合并。
2. 当一个Region的数据达到256M， 会进行split操作， 即将一个region拆分成两个region， 拆分完成后， 老的region下线。 新的两个region根据负载情况分配到对应的regionServer。

从一张表的存储理解Hbase架构

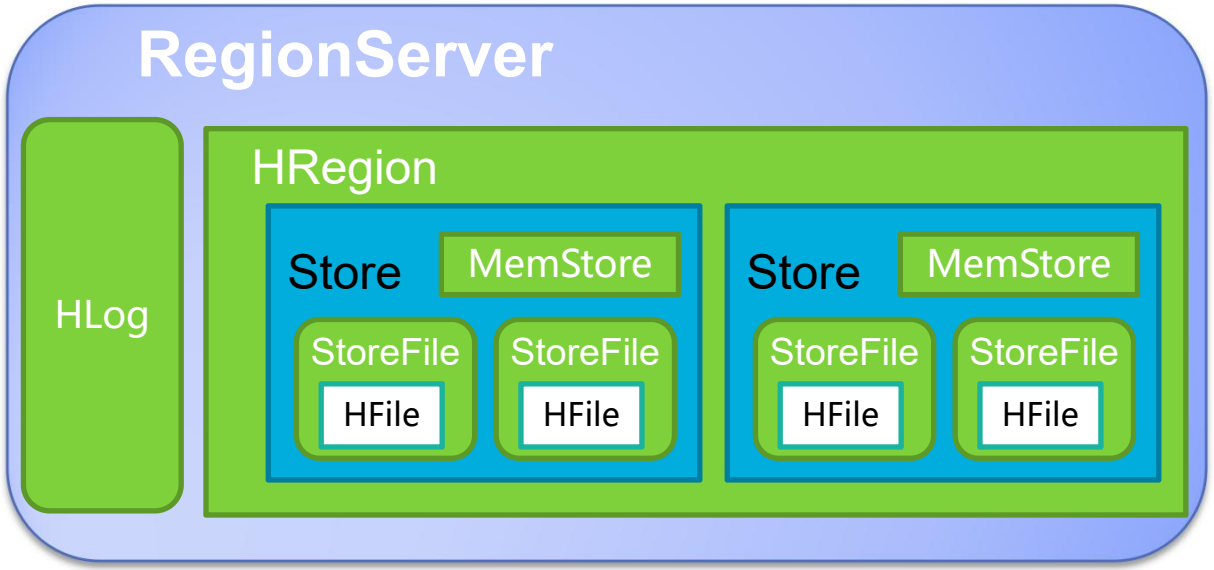
基于一张表：

1. 一个表的Region对应HDFS的一个目录

2. Store和列族一一对应。

在HDFS上， 一个列族对应一个HDFS目录，
在Region目录下面。

3. 一个Store对应多个StoreFile和一个MemStore



4. 数据写，先写日志， 然后存入MemStore， 返回Client写成功。

5. MemStore中数据大小达到阈值， 数据将Flush到StoreFile。

Split和Compact的影响？？

6. 当Store中的文件数量达到阈值， 会执行Compact(合并)操作

7. 当一个Region的StoreFile的大小达到阈值， 这个表的Region会进行split。

Hbase 列族数量不能太多

1. Flush会产生更多IO
   Flush的最小单元是region， 也就是说一个region中的某个列族做Flush操作， 其他的列族也会Flush。
   对每个列族而言， 每次Flush都会产生一个文件， 频繁Flush必然会产生更多的StoreFile，
StoreFile数量增多又会产生更多的Compact操作。 Flush和Compact都是很重的IO操作。

2. Split操作可能会导致数据分布不均匀
   Split的最小单元是region， 如果这个region有两个列族A、B。
   列族A有100亿条记录，列族B有100条记录。
   如果最终Split成20个region， 那么列族B的100条记录会分布到20个region上， 扫描列族B的性能低下。

Hbase二级索引

Hbase数据访问方式：
1. 通过rowkey（单个或者rowkey正则等）
2. 全表扫描

如果通过非rowkey字段， 只能全表扫描， 性能低下。

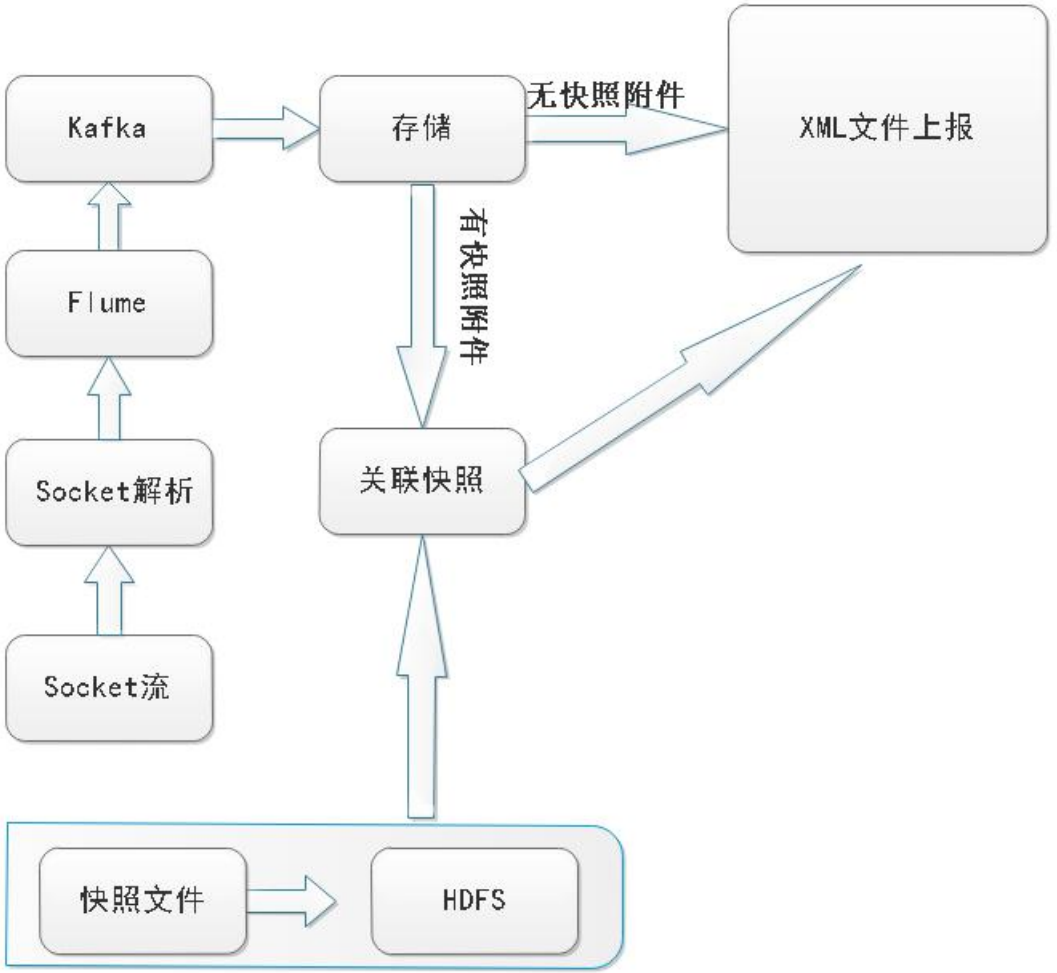
重构Hbase
Hbase + Solr
Hbase + Elasticsearch
Phoenix

Hbase实现Exactly-once语义

幂等写入（ idempotent writes）

rowkey设计

Kafka每条消息的partition、offset等获取



Kafka每条消息的partition、offset等获取

Hbase Rowkey： md5(topic + "|" + groupName + "|" + part) + String.format("%020d", java.lang.Long.valueOf(offset))

先保存数据-》 保存offset

```
var kafkaStream : InputDStream[(String, Int, Long, String)] = null

val messageHandler = (mmd : MessageAndMetadata[String, String]) => (mmd.topic,
mmd.partition, mmd.offset, mmd.message())

kafkaStream = KafkaUtils.createDirectStream[String, String, StringDecoder, StringDecoder,
(String, Int, Long, String)](ssc, kafkaParams, fromOffsets, messageHandler)
```

Hbase实现Exactly-once

```scala
// 每条消息:  (topic, partition, offset, message)
val streaming = createNewDirectKafkaStream(ssc, kafkaParams, Set(topic), groupName)
streaming.foreachRDD(rdd=>{
  if(!rdd.isEmpty()){
    rdd.map(x=>ParseUtils.parseMsg(x, groupName)).foreachPartition(p=>{
      val hbaseConf = HBaseConfiguration.create()
      hbaseConf.set("hbase.zookeeper.quorum", "spark1234")
      hbaseConf.set("hbase.zookeeper.property.clientPort", "12181")
      val conn = ConnectionFactory.createConnection(hbaseConf)
      val table = conn.getTable(TableName.valueOf(hTableName))
      import scala.collection.JavaConversions._
      table.put(seqAsJavaList(p.toSeq))
    })
  }

  saveOffsets(rdd.asInstanceOf[HasOffsetRanges].offsetRanges, groupName)
}
```

SQL on Hbase

1. Hbase on Hive

```
CREATE EXTERNAL TABLE hbase_test(key string, topic string, part string, offset string, srcip string, srcport string)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES ("hbase.columns.mapping" = ":key,info:topic, info:part, info:offset, info:srcip, info:srcport")
TBLPROPERTIES("hbase.table.name" = "mylog");
```

2. Hbase on Spark

3. Phoenix

Phoenix

1. Phoenix整合CDH（源码修改、编译）

2. Phoenix安装和使用

3. Phoenix整合Spark

4. Phoenix Query Server（JDBC）

Phoenix二级索引

1. Covered Indexes

通过索引就可以返回要查询的所有数据， 而不需要再通过所以去查询数据表。
索引的字段必须包含需要查询的所有的列（select的列和where条件的列）

```
CREATE INDEX my_index ON my_table (v1,v2) INCLUDE(v3)
```

2. Functional Indexes

函数索引： 通过在字段创建函数表达式的方式查询

```
CREATE INDEX UPPER_NAME_IDX ON EMP (UPPER(FIRST_NAME||' '||LAST_NAME))
SELECT EMP_ID FROM EMP WHERE UPPER(FIRST_NAME||' '||LAST_NAME)='JOHN DOE'
```

3. Global Indexes

全局索引， read heavy场景。 写性能差

```
CREATE INDEX myindex  ON test(id);
SELECT  id  FROM test WHERE id='12345';       // 能使用到索引
SELECT  id, name FROM test WHERE id='12345'; // 不能使用到索引
```

4. Local Indexes

局部索引： CREATE LOCAL INDEX MYINDEX ON test (id);

Thanks

FAQ时间