Day19_view & Phoenix操作 & HBase调优

大数据-张军锋 Day19 rowKey的设计 二级索引 Phoenix

数据库中view的作用

> 在数据库管理系统当中，view是描述数据库中信息的一种方式。若要将数据项按某种特定的序列排列、突出某些数据项，或者只显示特定的数据项，这些都可以通过view(视图)来实现。对于任何数据库来说，可能有一些视图需要定义。与拥有少量数据项的数据库相比，拥有很多数据项的数据库可能有更多的视图。就像虚拟表一样，视图本身并不真正的存储信息，但仅仅只是从一个或多个已经存在的表中将数据取出。虽然很无常，一个视图能通过存储其查询标准，而被重复的访问。

view 应用场景：限制数据查看权限，保存复杂的sql

```
create view ad_limit as
select * from ad where user_id < 50
```

`select * from ad_limit` 等同于 `select * from (select * from ad where user_id < 50)`

注意：view 不保存数据，保存的是sql

phoenix sql语句操作HBase

```
select * from system.catalog where table_schem='SYSTEM'and table_name='STATS'and column_name='P
HYSICAL_NAME';
-- 创建表必须添加主键
create table phoenix_user(
    user_id integer not null primary key
    ,username varchar(20)
    ,age integer
    ,birthday varchar(20)
)

-- 查询数据
select * from phoenix_user

-- 插入数据
upsert into phoenix_user (user_id,username,age,birthday) values(1,'张三',12,'2012-01-02');
upsert into phoenix_user (user_id,username,age,birthday) values(2,'李四',12,'2012-04-02');
upsert into phoenix_user (user_id,username,age,birthday) values(3,'王五',12,'2012-07-02');

-- 修改数据,只能根据主键进行修改
upsert into phoenix_user(user_id,age) values(1,25);
upsert inrto phoenix_user(user_id,age) values()

-- 删除数据 年龄大于20的删除1
delete from phoenix_user where age > 20
```

java API操作phoenix

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Random;

/**
* 项目名称：phoenixTest
* 类名称：PhoenixJdbcTest
* 类描述：phoenix jdbc 操作
* @author Allen
*/
public class PhoenixJdbcTest {

    public static final String URL = "jdbc:phoenix:master,slaver1,slaver2:2181";
    public static final String DRIVER_CLASS = "org.apache.phoenix.jdbc.PhoenixDriver";
    public static final String USER_NAME = "root";
    public static final String PASSWORD = "";

    private Connection connection;

    public PhoenixJdbcTest() {
        try {
            Class.forName(DRIVER_CLASS);
            this.connection = DriverManager.getConnection(URL, USER_NAME, PASSWORD);
        } catch (Exception e) {
            System.out.println("connection exception ......");
            e.printStackTrace();
        }
    }

    /**
    * findData 查找数据
    * @param @throws Exception 参数
    * @return void 返回类型
    * @Exception 异常对象
    * @author Allen
    */
    public void findData() throws Exception {
        Statement statement = connection.createStatement();
        String sql = "select * from phoenix_user";
        ResultSet resultSet = statement.executeQuery(sql);
        while (resultSet.next()) {
            System.out.println("user_id:" + resultSet.getInt("user_id") + "\tusername:" + resul
tSet.getString(2)
                    + "\tage:" + resultSet.getInt(3) + "\tbirthday:" + resultSet.getString(4));
        }
    }

    /**
    * insertData 通过sql查询数据到hbase上
    * @param @throws Exception 参数
    * @return void 返回类型
    * @Exception 异常对象
    * @author Allen
    */
    public void insertData() throws Exception {
        String sql = "upsert into phoenix_user (user_id,username,age,birthday) value
s(?,?,?,?)";
        PreparedStatement prepareStatement = connection.prepareStatement(sql);
        Random random = new Random();
```

```
            for (int i = 0; i < 10; i++) {
                prepareStatement.setInt(1, 14 + i);
                prepareStatement.setString(2, "user" + i);
                prepareStatement.setInt(3, random.nextInt(10) + 1);
                prepareStatement.setString(4, "2014-0" + random.nextInt(10) + "-" + random.nextIn
t(30) + 1);
                prepareStatement.addBatch();
            }
            prepareStatement.executeBatch();
            // phoenix 的 connection自动提交默认是关闭的，这点与其他数据库是不一样的
            prepareStatement.getConnection().commit();
        }

        /**
        * cleanUp 关闭资源
        * @param  参数
        * @return void 返回类型
        * @Exception 异常对象
        * @author Allen
        */
        public void cleanUp() {
            if (connection != null) {
                try {
                    connection.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
        }

        public static void main(String[] args) throws Exception {
            PhoenixJdbcTest pTest = new PhoenixJdbcTest();
            pTest.findData();
//          pTest.insertData();
            pTest.cleanUp();
        }
    }
```

HBase协处理器

协处理器的作用：hbase创建二级索引比较麻烦，对于排序，求和，计算这些简单的功能实现非常麻烦，为了降低难度提出了协处理器的概念。

> 协处理器的特性
>
> - 允许用户执行region级的操作，使用类似触发器的功能
> - 允许扩展现有的RPC协议引入自己的调用
> - 提供一个非常灵活的、可用于建立分布式服务的数据模型
> - 能够自动化扩展、负载均衡、应用请求路由
>
> 协处理器提供了两大类Observer、endPoint

1. Observer（观察者）
   该类是与RDMS中的触发器类似。回调函数在一些特定的事件发生时被调用。
   事件包括：用户产生的事件或者服务端内部产生的事件。
   协处理器框架提供的接口如下：
   a、RegionObserver：用户可以通过这种处理器来处理数据修改事件，它们与表的Region紧密关联。region级的操作。对应的操作是：put/delete/scan/get
   b、MasterObserver：可以用作管理或DDL类型的操作，是集群级的操作。对应的操作是：创建、删除、修改表。
   c、WALObserver：提供控制WAL的钩子函数。
   Observer定义好钩子函数，服务端可以调用。

    1. endPoint(终端)
       该类的功能类似RDMS中的存储过程。将用户的自定义操作添加到服务器端，endPoint可以通过添加远程过程调用来扩展RPC协议。用户可以将自定义完成某项操作代码部署到服务器端。例如：服务器端的计算操作。
       当二者结合使用可以决定服务器端的状态。

> 以创建二级索引为例

1. 创建项目，添加hbase依赖，在项目中定义observe类，继承BaseRegionObserver类，重写方法实现监听出发功能
2. 项目发成jar，放到hdfs上
3. 把协处理器添加到表bd14:order_item上，实现监听

```
import java.io.IOException;
import java.util.List;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.Cell;
import org.apache.hadoop.hbase.CellUtil;
import org.apache.hadoop.hbase.TableName;
import org.apache.hadoop.hbase.client.Connection;
import org.apache.hadoop.hbase.client.ConnectionFactory;
import org.apache.hadoop.hbase.client.Durability;
import org.apache.hadoop.hbase.client.Put;
import org.apache.hadoop.hbase.client.Table;
import org.apache.hadoop.hbase.coprocessor.BaseRegionObserver;
import org.apache.hadoop.hbase.coprocessor.ObserverContext;
import org.apache.hadoop.hbase.coprocessor.RegionCoprocessorEnvironment;
import org.apache.hadoop.hbase.regionserver.wal.WALEdit;

/**
* 项目名称：coprocessorTest
* 类名称：SecondaryIndexAutoUpdate
* 类描述：使用observe的coprocessor来自动更新order_item二级索引数据
* create 'bd14:order_item','i'
* create 'bd14:order_item_subtotal_index','r'
* 把这个协处理器添加到order_item表上，索引自动更新到order_item_subtotal_index里
*
* @author Allen
*/
public class SecondaryIndexAutoUpdate extends BaseRegionObserver{
    // 在表数据被put之前执行索引表数据的添加
    // put 待被插入到bd14:order_item表中的put对象

    @Override
    public void prePut(ObserverContext<RegionCoprocessorEnvironment> e, Put put, WALEdit edit,
Durability durability)
            throws IOException {
        // 参数put中就是要保存进bd14:order_item的put对象，从中获取subtotal和rowkey，保存数据到bd14:orde
r_item_subtotal_index
        List<Cell> subtotalCell = put.get("i".getBytes(), "subtotal".getBytes());
        if(subtotalCell != null && subtotalCell.size() > 0){
            RegionCoprocessorEnvironment environment = e.getEnvironment();
            Configuration conf = environment.getConfiguration();

            // 通过获取conf建立与hbase之间的链接
            Connection connection = ConnectionFactory.createConnection(conf);
            Table table = connection.getTable(TableName.valueOf("bd14:order_item_subtotal_inde
x"));

            // 构建要保存索引的put对象
            Put indexPut = new Put(CellUtil.cloneValue(subtotalCell.get(0)));
            indexPut.addColumn("r".getBytes(), put.getRow(), null);
            table.put(indexPut);
            table.close();
        }
    }

}
```

4. 创建表并为表添加协处理器 `alter 'bd14:order_item','coprocessor'=>'hdfs:///cop.jar|top.xiesn.coprocessor.SecondaryIndexAutoUpdate|1001|'`
5. 将数据添加到order_item表中 put `'bd14:order_item','wxd','i:subtotal','1002'`
6. 查询索引表中的数据 `scan 'bd14:order_item_subtotal_index'`

AggregationClient

> 这个类主要是做聚合操作的，下面以统计表中的行键个数为例来说明这个类的使用方式

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.TableName;
import org.apache.hadoop.hbase.client.Scan;
import org.apache.hadoop.hbase.client.coprocessor.AggregationClient;
import org.apache.hadoop.hbase.client.coprocessor.LongColumnInterpreter;

/**
* 项目名称：hbasetest
* 类名称：RowCountAggregation
* 类描述：hbase统计行键
* @author Allen
*/
public class RowCountAggregation {

    public static Configuration CONF = HBaseConfiguration.create();
    public AggregationClient aggregationClient;
    public RowCountAggregation() {
        aggregationClient = new AggregationClient(CONF);
    }

    public void getRowCount() throws Throwable{
        Scan scan = new Scan();
        scan.addFamily("i".getBytes());
        long count = aggregationClient.rowCount(TableName.valueOf("bd14:order_item".getByte
s()), new LongColumnInterpreter(), scan);
        System.out.println(count);
    }

    public static void main(String[] args) throws Throwable {
        RowCountAggregation rowCountAggregation = new RowCountAggregation();
        rowCountAggregation.getRowCount();
    }

}
```

HBase调优

client调优

1. hbase.client.write.buffer：写缓存大小，默认为2M，推荐设置为6M，单位是字节，当然不是越大越好，如果太大，则占用的内存太多；
2. hbase.client.scanner.caching：scan缓存，默认为1，太小，可根据具体的业务特征进行配置，原则上不可太大，避免占用过多的client和rs的内存，一般最大几百，如果一条数据太大，则应该设置一个较小的值，通常是设置业务需求的一次查询的数据条数，比如：业务特点决定了一次最多100条，则可以设置为100
3. 通过scan取完数据后，记得要关闭ResultScanner，否则RegionServer可能会出现问题（对应的Server资源无法释放）

服务端调优

1. hbase.regionserver.handler.count：该设置决定了处理RPC的线程数量，默认值是10，通常可以调大，比如：150，当请求内容很大（上MB，比如大的put、使用缓存的scans）的时候，如果该值设置过大则会占用过多的内存，导致频繁的GC，或者出现OutOfMemory，因此该值不是越大越好。
2. hbase.hregion.max.filesize ：配置region大小，0.94.12版本默认是10G，region的大小与集群支持的总数据量有关系，如果总数据量小，则单个region太大，不利于并行的数据处理，如果集群需支持的总数据量比较大，region太小，则会导致region的个数过多，导致region的管理等成本过高，如果一个RS配置的磁盘总量为3T*12=36T数据量，数据复制3份，则一台RS服务器可以存储10T的数据，如果每个region最大为10G，则最多1000个region，如此看，94.12的这个默认配置还是比较合适的，不过如果要自己管理split，则应该调大该值，并且在建表时规划好region数量和rowkey设计，进行region预建，做到一定时间内，每个region的数据大小在一定的数据量之下，当发现有大的region，或者需要对整个表进行region扩充时再进行split操作，一般提供在线服务的hbase集群均会弃用hbase的自动split，转而自己管理split。
3. hbase.hregion.majorcompaction：配置major合并的间隔时间，默认为1天，可设置为0，禁止自动的major合并，可手动或者通过脚本定期进行major合并，有两种compact：minor和major，minor通常会把数个小的相邻的storeFile合并成一个大的storeFile，minor不会删除标示为删除的数据和过期的数据，major会删除需删除的数据，major合并之后，一个store只有一个storeFile文件，会对store的所有数据进行重写，有较大的性能消耗。
4. hbase.hstore.compactionThreshold：HStore的storeFile数量>= compactionThreshold配置的值，则可能会进行compact，默认值为3，可以调大，比如设置为6，在定期的major compact中进行剩下文件的合并。
5. hbase.hstore.blockingStoreFiles：HStore的storeFile的文件数大于配置值，则在flush memstore前先进行split或者compact，除非超过hbase.hstore.blockingWaitTime配置的时间，默认为7，可调大，比如：100，避免memstore不及时flush，当写入量大时，触发memstore的block，从而阻塞写操作。

对于调优的方式有很多，我也是从网络上摘录的，如有更多的需要，请自行搜索