[19] 中华人民共和国国家知识产权局

[51] Int. Cl.
G06F 17/30 (2006.01)
G06Q 10/00 (2006.01)

[12] 发明专利申请公布说明书

[21] 申请号 200810159786.6

[43] 公开日 2009 年 4 月 8 日

[11] 公开号 CN 101404013A

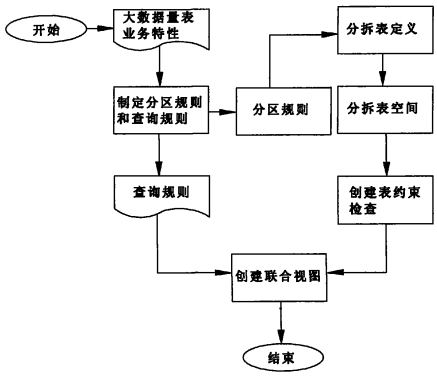
[22] 申请日 2008.11.13
[21] 申请号 200810159786.6
[71] 申请人 山东浪潮齐鲁软件产业股份有限公司
地址 250014 山东省济南市历下区山大路 224 号
[72] 发明人 宋智强
[74] 专利代理机构 济南信达专利事务所有限公司
代理人 姜 明
权利要求书 1 页 说明书 5 页 附图 1 页

[54] 发明名称

数据库大数据量表存储和查询方法

[57] 摘要

本发明提供一种数据库大数据量表存储和查询方法，该方法通过数据库物理设计和逻辑设计来实现海量数据表的存储“分区”，而不依赖于数据库本身是否具备分区技术；该方法根据日结帐表所特有的一些特性设计合理的规则，根据规则来对表定义进行“分区”设计，使记录在向数据库插入过程中自动“分发”到不同存储“分区”中；“分区”之后的日结帐表根据规则设计来创建联合视图，通过联合视图提供查询功能，这样实现的日结帐表在数据查询上会有明显的优势，而对于表空间存在大小限制的数据库，也不会因为日结帐表数据量庞大而导致数据库表空间达到上限而无法解决。



1．数据库大数据量表存储和查询的方法，其特征在于，根据企业中大数据量的表特性进行分区规则设计，通过设计好的分区规则，实现数据库表的逻辑设计和物理设计，从而实现大数据量表的分区存储以及分区查询的目的，具体实现步骤如下：

1）企业中日结帐表按照月份分区规则设计，日结帐表都存在日期列，而且对于日结帐表的查询方式是一个完整的月份，分区规则按照月份来制定；

2）在定义好分区规则之后，对于表定义按照分区规则划分成多份，按月份的划分定义成12个表，每个表都定义有特定的表空间，数据存储在不同特定的表空间中；

3）在定义好的表的基础上，增加表检查约束定义，约束条件为月份，通过检查约束实现记录增加的时候自动存储到不同表空间中；

4）分区表的定义完成之后，定义联合视图，通过联合视图联合所有分区表实现大数据量日结帐表的分区存储；

5）通过修改联合视图的定义，让每个分区表都增加相应的查询条件，这样就实现了分区查询的目的。

2、根据权利要求1所述的方法，其特征在于利用通用数据库定义以及设计规则来实现数据库表分区存储以及分区查询的目的，不依赖于任何数据库深层次的技术。

数据库大数据量表存储和查询方法

技术领域

本发明涉及计算机应用技术领域，具体地说是一种数据库大数据量表存储和查询的实现方法。

背景技术

在很多企业的业务系统中都存在“日清日结”或者“日清月结”的功能，这样会方便企业对当天或者整个月商品的库存以及销售情况进行盘点；在这一过程中，一般会生成大量的数据记录来存储到数据库相应的日结帐表中，随着时间的累积，这样的表会越来越庞大，在一些相对集中的企业数据库中数量级会达到亿级以上。

在数据库中如果存在记录数上亿的表，会给数据的查询带来比较大的麻烦，除了查询 SQL 本身的响应时间会越来越慢，主机系统在 CPU、内存资源的使用上也会越来越高，更严重的是会使跟这些表相关的业务功能无法进行，在并发要求比较高的数据库系统中，可能会给系统带来大量的数据库锁，并影响整个数据库系统，导致数据库的崩溃；而对于表空间存在大小限制的数据库，庞大数据量的表也是一个比较棘手的问题。

在解决大数据量表存储问题方面，比较常用的方式是在经过一段时间之后，对表中的数据进行转移，也就是从最初的表中将数据迁移到另一个或多个存储历史过渡数据的表中，也有直接将历史数据清除的做法；这两种做法都存在一定问题，将数据隔一段时间转移到历史过渡表中，一方面需要考虑历史过渡表怎么维护的问题，在转移数据上也需要花费很多精力，另外就是对于历史过渡数据的查询比较麻烦，查询性能也不会很好；如果将数据直接清空也将面临历史数据查询的问题。上面的方式多数时候都不具备自动维护的特点，都是在维护人员人工操作的情况下进行的。

发明内容

本发明的数据库大数据量表存储和查询方法，是根据企业中大数据量的表特性进行分区规则设计，通过设计好的分区规则，实现数据库表的逻辑设计和物理设计，从而实现大数据量表的分区存储以及分区查询的目的，具体实现步骤如下：

1）企业中日结帐表按照月份分区规则设计，日结帐表都存在日期列，而且对于日结帐表的查询方式是一个完整的月份，分区规则按照月份来制定；

2）在定义好分区规则之后，对于表定义按照分区规则划分成多份，按月份的划分定义成 12 个表，每个表都定义有特定的表空间，数据存储在不同特定的表空间中；

3）在定义好的表的基础上，增加表检查约束定义，约束条件为月份，通过检查约束实现记录增加的时候自动存储到不同表空间中；

4）分区表的定义完成之后，定义联合视图，通过联合视图联合所有分区表实现大数据量日结帐表的分区存储；

5）通过修改联合视图的定义，让每个分区表都增加相应的查询条件，这样就实现了分区查询的目的。

利用通用数据库定义以及设计规则来实现数据库表分区存储以及分区查询的目的，不依赖于任何数据库深层次的技术。

本发明的有益效果是：

1、通过企业中大数据量表的一些特性，来实现表的逻辑分区设计，从而实现大数据量表物理上的分区实现。

2、采用一些通用的数据库定义方法来解决大数据量表的分区实现问题，而不依赖数据库是否具备分区技术，大多数数据库产品都能够实现。

3、不需要花太多精力考虑数据存储维护的问题，表空间的限制相当于扩大到原来的 12 倍，避免了数据库表空间维护给系统运维人员带来的麻烦。

4、解决数据库大数据量表的查询 SQL 的响应时间随时间增长变得越来越慢的问题，在这种实现效果下，12 年之后才会相当于原方式 1 年之后的查询响应时间。

5、整个实现方式对于客户端工具和程序来说，相当于一个“黑盒子”，不需要开发人员做什么调整，在以后的维护当中也不需要系统维护人员做人工的干预。

附图说明

附图 1 是数据库大数据量表存储和查询的方法流程结构示意图。

具体实施方式

本发明的方法是是通过合理的分区规则来解决大数据量表的问题；不需要维护人员做过多的维护工作，更不需要定期进行历史数据的迁移工作。

企业中的日结帐表一般都是根据特定算法，来记录当天商品的销售、库存等财务报表数据的情况；汇总的目的一方面企业需要知道当天销售过程中的成本以及获得的利润情况，另一方面一些具体的人员，例如客户经理需要及时了解每个客户的当天的订货情况；这种日结帐表除了能够提供某一天或者某一段时间的报表查询数据之外，还将作为月结算的原始数据，那么这样来看日结帐表对系统来说真正有意义的都是不超过一个完整月份的数据。这样我们就可以根据这个规则对日结账表进行按月分区实现，实现分区之后日结帐表对应 12 个表。

具体实现的技术方案如下，假设某企业中存在一个客户商品日账表（SD_ITEM_CUST_DAY），

该表是用来存储每天该企业的所有客户定购商品情况的，我们现在想象一下在一个拥有庞大客户群的企业，比如拥有 20 万个客户，而每个客户每天至少定购 10 种商品，这样算下来这个表一天增加的记录是 200 万条，如果在一年当中把一些假期去掉，按剩下 200 天算，那一年下来这个表中存储的记录为 4 亿条记录，这是相当庞大的一个数字。现在我们看这个表原来的定义：

```
CREATE TABLE "SD_ITEM_CUST_DAY"  (
"COM_ID" VARCHAR(30) NOT NULL ,
"SALEORG_ID" VARCHAR(30) NOT NULL ,
"DATE1" CHAR(8) NOT NULL ,
"CUST_ID" VARCHAR(30) NOT NULL ,
"SLSMAN_ID" VARCHAR(30) NOT NULL WITH DEFAULT '0' ,
"ITEM_ID" VARCHAR(30) NOT NULL ,
"QTY_SOLD" DECIMAL(18,2) WITH DEFAULT 0 ,
"AMT_SOLD_WITH_TAX" DECIMAL(18,2) WITH DEFAULT 0)
IN "DATA_QUERY";
```

首先将该表逻辑定义上按月份划分成 12 个表，例如：

```
SD_ITEM_CUST_DAY01、SD_ITEM_CUST_DAY02...
 CREATE TABLE "SD_ITEM_CUST_DAY01"  (
"COM_ID" VARCHAR(30) NOT NULL ,
"SALEORG_ID" VARCHAR(30) NOT NULL ,
"DATE1" CHAR(8) NOT NULL ,
"CUST_ID" VARCHAR(30) NOT NULL ,
"SLSMAN_ID" VARCHAR(30) NOT NULL WITH DEFAULT '0' ,
"ITEM_ID" VARCHAR(30) NOT NULL ,
"QTY_SOLD" DECIMAL(18,2) WITH DEFAULT 0 ,
"AMT_SOLD_WITH_TAX" DECIMAL(18,2) WITH DEFAULT 0)
  IN "DATA_QUERY";
 ......
```

在定义划分好之后需要解决两方面问题，分别是分区存储和分区查询如何实现的问题。针对分区存储实现的问题，可以进一步细分为两个问题，一是如何将物理存储按照月份实现自动存储，二是如何对客户端或者程序员来说操作这 12 个表仍然跟 “分区”之前没有区别，这样客户端的工具及程序就不用去做任何的调整。

为了解决分区存储的第一个问题，我们在定义这 12 个表的时候每个表都指定了一个特定的表空间，另外特别增加一个特定的检查约束，检查约束的目的是保证表存储的时候按照我们所设想的每个月的数据存储在特定月份的表中。实现如下：

```
CREATE TABLE "SD_ITEM_CUST_DAY01"  (
"COM_ID" VARCHAR(30) NOT NULL ,
"SALEORG_ID" VARCHAR(30) NOT NULL ,
"DATE1" CHAR(8) NOT NULL ,
```

```
"CUST_ID" VARCHAR(30) NOT NULL ,
"SLSMAN_ID" VARCHAR(30) NOT NULL WITH DEFAULT '0' ,
"ITEM_ID" VARCHAR(30) NOT NULL ,
"QTY_SOLD" DECIMAL(18,2) WITH DEFAULT 0 ,
"AMT_SOLD_WITH_TAX" DECIMAL(18,2) WITH DEFAULT 0)
IN "DATA_QUERY01";
ALTER TABLE "SD_ITEM_CUST_DAY01"
ADD CONSTRAINT "ITEMCUSTDAYCHECK01" CHECK
 (SUBSTR(DATE1,5,2) = '01');
 ……
```

为了解决分区存储的第二个问题我们需要将这 12 个表创建成一个视图，通过联合定义使 12 个表对客户端及程序员表现为一个表，实现如下：

```
CREATE VIEW SD_ITEM_CUST_DAY AS
(
   SELECT * FROM SD_ITEM_CUST_DAY01 UNION ALL
   SELECT * FROM SD_ITEM_CUST_DAY02 UNION ALL
   SELECT * FROM SD_ITEM_CUST_DAY03 UNION ALL
   SELECT * FROM SD_ITEM_CUST_DAY04 UNION ALL
   SELECT * FROM SD_ITEM_CUST_DAY05 UNION ALL
   SELECT * FROM SD_ITEM_CUST_DAY06 UNION ALL
   SELECT * FROM SD_ITEM_CUST_DAY07 UNION ALL
   SELECT * FROM SD_ITEM_CUST_DAY08 UNION ALL
   SELECT * FROM SD_ITEM_CUST_DAY09 UNION ALL
   SELECT * FROM SD_ITEM_CUST_DAY10 UNION ALL
   SELECT * FROM SD_ITEM_CUST_DAY11 UNION ALL
   SELECT * FROM SD_ITEM_CUST_DAY12  );
```

在分区存储的问题解决之后，我们面临的是查询性能问题，因为我们不仅需要解决存储表空间数据量大的问题，还要解决在查询上分区之后的性能明显比分区之前好的问题。假使我们现在有一个查询 SQL：select CUST_ID, ITEM_ID, SUM(QTY_SOLD) from SD_ITEM_CUST_DAY where DATE1=' 20080908' ，如果按照上面的联合视图的方式去查询，最终的结果会是下面 12 条 SQL 的查询结果的一个联合结果集，

```
select CUST_ID,ITEM_ID,SUM(QTY_SOLD) from SD_ITEM_CUST_DAY01 where DATE1=' 20080908' ;
selectCUST_ID,ITEM_ID, SUM(QTY_SOLD) from SD_ITEM_CUST_DAY02 where DATE1=' 20080908' ;
selectCUST_ID,ITEM_ID, SUM(QTY_SOLD) from SD_ITEM_CUST_DAY03 where DATE1=' 20080908' ;
selectCUST_ID,ITEM_ID, SUM(QTY_SOLD) from SD_ITEM_CUST_DAY04 where DATE1=' 20080908' ;
selectCUST_ID,ITEM_ID, SUM(QTY_SOLD) from SD_ITEM_CUST_DAY05 where DATE1=' 20080908' ;
selectCUST_ID,ITEM_ID, SUM(QTY_SOLD) from SD_ITEM_CUST_DAY06 where DATE1=' 20080908' ;
selectCUST_ID,ITEM_ID, SUM(QTY_SOLD) from SD_ITEM_CUST_DAY07 where DATE1=' 20080908' ;
selectCUST_ID,ITEM_ID, SUM(QTY_SOLD) from SD_ITEM_CUST_DAY08 where DATE1=' 20080908' ;
selectCUST_ID,ITEM_ID, SUM(QTY_SOLD) from SD_ITEM_CUST_DAY09 where DATE1=' 20080908' ;
selectCUST_ID,ITEM_ID, SUM(QTY_SOLD) from SD_ITEM_CUST_DAY10 where DATE1=' 20080908' ;
selectCUST_ID,ITEM_ID, SUM(QTY_SOLD) from SD_ITEM_CUST_DAY11 where DATE1=' 20080908' ;
select CUST_ID, ITEM_ID, SUM(QTY_SOLD) from SD_ITEM_CUST_DAY12 where
```

DATE1=’20080908’;

这并不是我们想要的结果，这样的查询效率肯定很差，而且会比“分区”之前更差。我们实现的目的是想让这个SQL只会去查询SD_ITEM_CUST_DAY09一个表，即执行下面SQL：

select CUST_ID, ITEM_ID, SUM(QTY_SOLD) from SD_ITEM_CUST_DAY09 where DATE1=’20080908’;

为了实现这样的效果我们将原来SD_ITEM_CUST_DAY这个联合视图的定义重新调整一下，实现方式如下：

```
CREATE VIEW SD_ITEM_CUST_DAY AS(
SELECT * FROM SD_ITEM_CUST_DAY01 WHERE DATE1>='20080101' AND DATE1 <= '20080131' UNION ALL
SELECT * FROM SD_ITEM_CUST_DAY02 WHERE DATE1>='20080201' AND DATE1 <= '20080231' UNION ALL
SELECT * FROM SD_ITEM_CUST_DAY03 WHERE DATE1>='20080301' AND DATE1 <= '20080331' UNION ALL
SELECT * FROM SD_ITEM_CUST_DAY04 WHERE DATE1>='20080401' AND DATE1 <= '20080431' UNION ALL
SELECT * FROM SD_ITEM_CUST_DAY05 WHERE DATE1>='20080501' AND DATE1 <= '20080531' UNION ALL
SELECT * FROM SD_ITEM_CUST_DAY06 WHERE DATE1>='20080601' AND DATE1 <= '20080631' UNION ALL
SELECT * FROM SD_ITEM_CUST_DAY07 WHERE DATE1>='20080701' AND DATE1 <= '20080731' UNION ALL
SELECT * FROM SD_ITEM_CUST_DAY08 WHERE DATE1>='20080801' AND DATE1 <= '20080831' UNION ALL
SELECT * FROM SD_ITEM_CUST_DAY09 WHERE DATE1>='20080901' AND DATE1 <= '20080931' UNION ALL
SELECT * FROM SD_ITEM_CUST_DAY10 WHERE DATE1>='20081001' AND DATE1 <= '20081031' UNION ALL
SELECT * FROM SD_ITEM_CUST_DAY11 WHERE DATE1>='20081101' AND DATE1 <= '20081131' UNION ALL
SELECT * FROM SD_ITEM_CUST_DAY12 WHERE DATE1>='20081201' AND DATE1 <= '20081231' );
```

做上面修改之后，我们就实现了分区查询的效果。

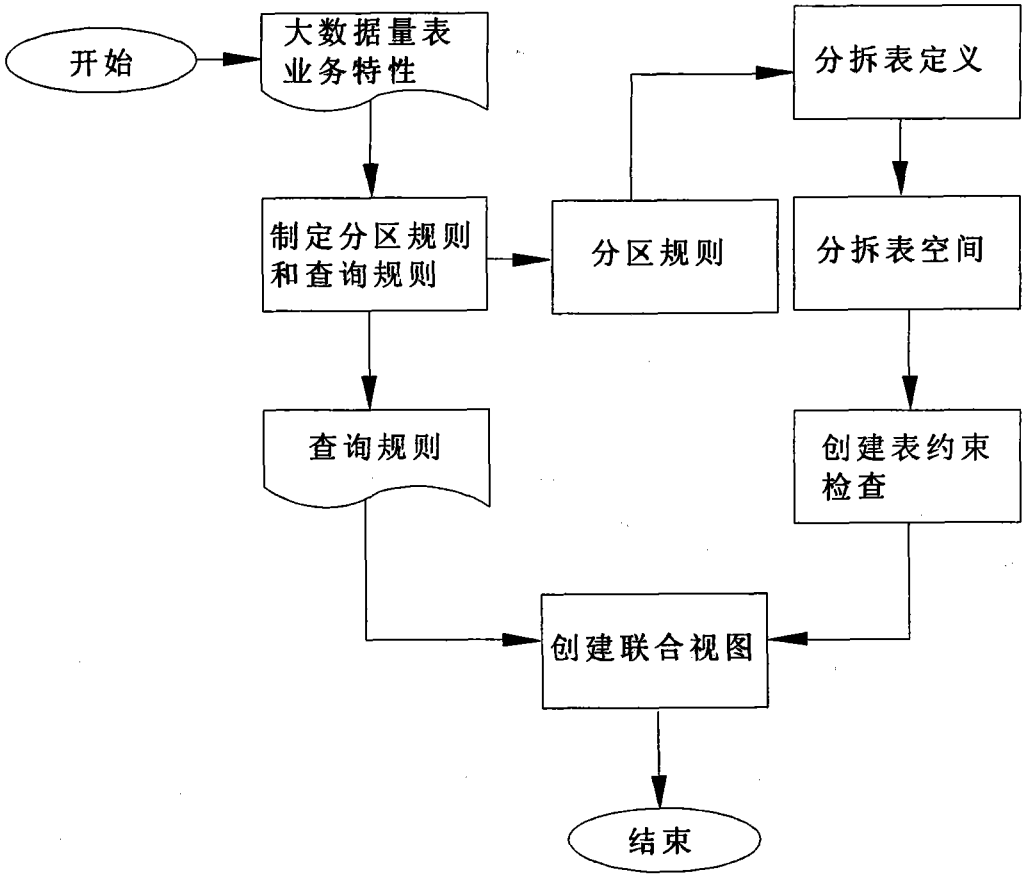

开始
大数据量表
业务特性
制定分区规则
和查询规则
查询规则
分区规则
分拆表定义
分拆表空间
创建表约束
检查
创建联合视图
结束

图 1