(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 104615750 B
(45)授权公告日 2017.11.03

(21)申请号 201510075765.6

(22)申请日 2015.02.12

(65)同一申请的已公布的文献号
申请公布号 CN 104615750 A

(43)申请公布日 2015.05.13

(73)专利权人 中国农业银行股份有限公司
地址 100005 北京市东城区建国门内大街69号

(72)发明人 朱浩

(74)专利代理机构 北京集佳知识产权代理有限公司 11227
代理人 王宝筠

(51)Int.Cl.
G06F 17/30(2006.01)

(56)对比文件
CN 103559272 A,2014.02.05,
CN 1848111 A,2006.10.18,
CN 101452486 A,2009.06.10,
CN 101458707 A,2009.06.17,
US 6073129 A,2000.06.06,

审查员 周循

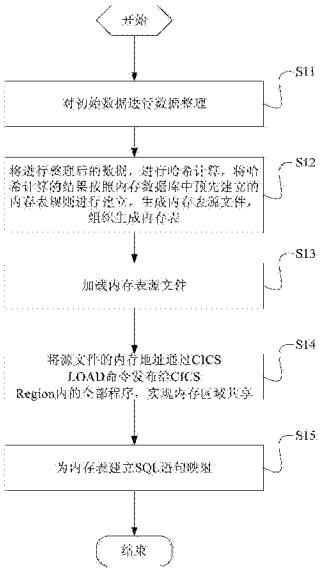
权利要求书3页 说明书12页 附图7页

(54)发明名称

一种主机系统下的内存数据库的实现方法

(57)摘要

本申请公开了一种主机系统下的内存数据库的实现方法，该方法包括：对初始数据进行数据整理；将进行整理后的数据进行哈希计算，将哈希计算的结果按照内存数据库中预先建立的内存表规则进行建立，生成内存表源文件，组织生成内存表；加载内存表源文件，设定下载指令CICS LOAD，当CICS Region内的程序调用内存数据库时，通过CICS LOAD将所述内存表源文件的入口地址发送给所述程序，为内存表建立SQL语句映射。从上述过程可以看出，本申请实施例公开的内存数据库实现方法中，内存表中的索引和数据均是在内存中构建结构，访问过程中减少了内存与磁盘数据交换之间的性能开销，因此，访问性能要高于内存数据库，因此能够进一步的满足应用对访问热表时的性能要求。



CN 104615750 B

1.一种主机系统下的内存数据库实现方法,其特征在于,该方法包括:

对初始数据进行数据整理;

将进行整理后的数据进行哈希计算,将哈希计算的结果按照内存数据库中预先建立的内存表规则进行建立,生成内存表源文件,组织生成内存表;其中,所述预先建立的内存表规则为:内存表为N+1级哈希结构,前N级只存储下一级哈希表所在位置,第N+1级用于存储数据;所述内存表的第N+1级存储的数据包括目标数据和离散数据,所述目标数据是基于初始数据构成,所述离散数据由动态新增的数据构成;

加载所述内存表源文件;

设定下载指令CICS LOAD,当客户信息控制系统区域CICS Region内的程序调用内存数据库时,通过CICS LOAD将所述内存表源文件的入口地址发送给所述程序,以作为该程序解析获取内存表地址的依据;

为所述内存表建立结构化查询语言SQL语句映射。

2.根据权利要求1所述的方法,其特征在于,所述数据整理为:将所述初始数据导入预先建立的物理表内。

3.根据权利要求1所述的方法,其特征在于,按照内存数据库中预先建立的内存表规则进行建立,生成内存表源文件的过程包括:

将所述整理后的数据进行N+1级哈希计算,将得到的N+1级哈希表按照所述内存表规则,将第N+1级用于存储数据,其余N级用于存储索引,生成相应的数据结构;

组织数据结构按照预先设定的形式生成内存表的高级语言源文件。

4.根据权利要求3所述的方法,其特征在于,所述索引中的各级哈希表包含有Offset和Num两个参数,其中,Offset代表下一级哈希表与内存的相对地址,Num代表下一级哈希表内的元素数量,当某一级哈希表中包含有H1个元素时,那该级哈希表以集合的形式进行表述为:

{{Offset1,Num1},{Offset2,Num2},...,{OffsetH1,NumH1}}。

5.根据权利要求4所述的方法,其特征在于,所述目标数据包括多个数据块,所述数据块包括:单条记录的访问结果VAL及访问该条记录时的索引字符串Key。

6.根据权利要求5所述的方法,其特征在于,所述目标数据还包括:指针Pointer,用于指向所述存储到离散数据中的新增数据。

7.根据权利要求6所述的方法,其特征在于,所述内存表中的起始地址为4KB的倍数,通过调整所述索引中各级哈希表的偏移量使得任一连续数据存储在同一虚拟页。

8.根据权利要求7所述的方法,其特征在于,所述SQL语句包括:访问、更新、删除、插入和游标。

9.根据权利要求8所述的方法,其特征在于,所述内存表的访问过程包括:

步骤一:定义用于索引的字符串为Key;

步骤二:根据拆分集合SkSet内的元素Sk_1划分Key,得到字符串集合,从所述集合中获得用于第一级哈希索引的字符串,定义为Key1;

步骤三:确定索引中的第一哈希级为当前待访问哈希级,所述Key1为当前待索引字符串;

步骤四:调用哈希函数,对所述当前待索引字符串进行哈希计算,并获得当前哈希值;

步骤五：获得哈希因子集合HSet内的与所述当前待索引字符串对应的元素，确定对应的当前级哈希表索引为：当前哈希值对所述当前待索引字符串对应的元素的取余结果；

步骤六：判断当前哈希级表索引的参数中的Num值是否等于0，如果是，则记录不存在，访问失败，否则，进入步骤七；

步骤七：当所述当前哈希级表索引具有下一哈希级时，确定当前哈希级表索引的下一级哈希表为当前待访问哈希级，并确定字符串集合中的下一字符串为当前待索引字符串，返回执行步骤四，直到索引表访问结束或中断，执行步骤八；

步骤八：根据当前哈希级表索引确定目标数据中与其对应的数据，遍历所述数据，并顺序比较所述索引字符串Key的值，如果匹配，则访问成功，否则进入步骤九；

步骤九：判断当前结构Pointer指针指向的位置，沿着指针开始逐步比较离散数据，如果匹配，则访问成功，否则，访问失败。

10.根据权利要求8所述的方法，其特征在于，所述内存表的删除操作包括：

访问待删除目标记录；

当访问到所述待删除目标记录后，将该待删除目标记录对应的最后一级哈希表内的Num值自动减1；

当所述待删除目标记录存在于离散数据区时，直接调用客户信息控制系统CICS命令FreeMain释放内存，并设定Pointer的指针指向所述待删除目标记录的上一条记录；

当所述待删除目标记录不存在于离散数据区时，将其对应的VAL的值清空为零。

11.根据权利要求8所述的方法，其特征在于，所述内存表的插入过程包括：

当主机采用联机类交易处理状态时，所述新增内存表数据的过程包括：

设置标志位的状态为更新状态，调用预先建立的内存表锁保护程序，所述标志位为预先在内存区域中保留的预设个数的字节；

判断目标数据区域内是否存在已被清空的记录项，若存在，则将待插入数据填充至所述记录项中VAL项中，并将所述记录项对应的Num的值进行累加操作，若不存在，则CICS命令GETMAIN申请内存空间，将所述待插入数据添加至该内存空间，并将原始的最后一项离散数据的Pointer指针指向新增记录处；

设置所述标志位的状态为完成插入。

12.根据权利要求8所述的方法，其特征在于，所述内存表的更新过程包括：

当主机采用联机类交易处理状态时，所述更新内存表数据的过程包括：

访问待更新目标记录；

当访问到所述待更新目标记录后，设置标志位的状态为更新状态，调用预先建立的内存表锁保护程序，所述标志位为预先在所述内存区域中保留的预设个数的字节；

更新目标记录中的VAL项的数据，重新计算索引的值，然后更新索引记录；

设置所述标志位的状态为完成更新。

13.根据权利要求11或12所述的方法，其特征在于，所述标志位为：

预先在共享内存中保留的预设个数的字节，当所述字节为1时，则表明内存表正处于更新或插入数据的状态，不可读取。

14.根据权利要求13所述的方法，其特征在于，所述内存表锁保护程序为：

用于占用系统的时间片而预先建立的循环程序，其中，所述循环程序的周期为主机时

钟周期的倍数，所述循环程序的循环次数，根据所述访问操作对应的最长时钟周期而决定。

15.根据权利要求8所述的方法，其特征在于，当主机采用批量类交易处理状态时，所述插入或更新所述内存表数据的过程包括：

将待更新或待插入数据记录到预先建立的物理表中；

利用批量更新程序从所述物理表中读取各条记录；

将所述各条记录中的数据存储于预先设定高级语言源程序内；

运行所述程序生成更新后的内存表。

一种主机系统下的内存数据库的实现方法

技术领域

[0001] 本申请涉及数据处理技术领域，特别涉及一种主机系统下的内存数据库的实现方法。

背景技术

[0002] 目前，大型商业银行仍主要以IBM主机系统，处理行内交易，在银行业应用的主机系统受限于IBM平台，故广泛应用的是关系型数据库管理系统DB2数据库。但是，类似于DB2等关系型数据库最擅长处理的是集合而非单条记录。当处理相同数量的记录时，逐条循环处理和批量处理之间的效率差异呈几何单位。

[0003] 而实际应用中，为了提高小表访问的效率，DB2提供了缓冲池bufferpool来籍此减少对磁盘进行I/O访问时的开销。

[0004] 但是从目前DB2数据库的应用现状来看，其数据访问效率并不能满足应用对访问热表时的性能要求。

发明内容

[0005] 有鉴于此，本申请提供一种主机系统下的内存数据库实现方法，以解决现有技术中的数据库的数据访问效率并不能满足应用对访问热表时的性能要求的问题。技术方案如下：

[0006] 一种主机系统下的内存数据库实现方法，该方法包括：

[0007] 对初始数据进行数据整理；

[0008] 将进行整理后的数据进行哈希计算，将哈希计算的结果按照内存数据库中预先建立的内存表规则进行建立，生成内存表源文件，组织生成内存表；

[0009] 加载所述内存表源文件；

[0010] 设定下载指令CICS LOAD，当客户信息控制系统区域CICS Region内的程序调用内存数据库时，通过CICS LOAD将所述内存表源文件的入口地址发送给所述程序，以作为该程序解析获取内存表地址的依据；

[0011] 为所述内存表建立结构化查询语言SQL语句映射。

[0012] 优选的，所述数据整理为：将所述初始数据导入预先建立的物理表内。

[0013] 优选的，所述预先建立的内存表规则为：内存表为N+1级哈希结构，其中，前N级只存储下一级哈希表所在位置，即索引，第N+1级用于存储数据。

[0014] 优选的，所述内存表的第N+1级存储的数据包括，目标数据和离散数据，其中所述目标数据是基于初始数据构成，所述离散数据由动态新增的数据构成。

[0015] 优选的，按照内存数据库中预先建立的内存表规则进行建立，生成内存表源文件的过程包括：

[0016] 将所述整理后的数据进行N+1级哈希计算，将得到的N+1级哈希表按照所述内存表规则，将第N+1级用于存储数据，其余N级用于存储索引，生成相应的数据结构；

[0017] 组织数据结构按照预先设定的形式生成内存表的高级语言源文件。

[0018] 优选的，所述索引中的各级哈希表包含有Offset和Num两个参数，其中，Offset代表下一级哈希表与内存的相对地址，Num代表下一级哈希表内的元素数量，当某一级哈希表中包含有Hl个元素时，那该级哈希表以集合的形式进行表述为：

[0019] {{Offset1,Num1},{Offset2,Num2},...,{OffsetHl,NumHl}}。

[0020] 优选的，所述目标数据包括多个数据块，所述数据块包括：单条记录的访问结果VAL及访问该条记录时的索引字符串Key。

[0021] 优选的，所述目标数据还包括：指针Pointer，用于指向所述存储到离散数据中的新增数据。

[0022] 优选的，所述内存表中的起始地址为4KB的倍数，通过调整所述索引中各级哈希表的偏移量使得任一连续数据存储在同一虚拟页。

[0023] 优选的，所述SQL语句包括：访问、更新、删除、插入和游标。

[0024] 优选的，所述内存表的访问过程包括：

[0025] 步骤一：定义用于索引的字符串为Key；

[0026] 步骤二：根据拆分集合SkSet内的元素Sk1划分Key，得到字符串集合，从所述集合中获得用于第一级哈希索引的字符串，定义为Key1；

[0027] 步骤三：确定索引中的第一哈希级为当前待访问哈希级，所述Key1为当前待索引字符串；

[0028] 步骤四：调用哈希函数，对所述当前待索引字符串进行哈希计算，并获得当前哈希值；

[0029] 步骤五：获得哈希因子集合HSet内的与所述当前待索引字符串对应的元素，确定对应的当前级哈希表索引为：当前哈希值对所述当前待索引字符串对应的元素的取余结果；

[0030] 步骤六：判断当前哈希级表索引的参数中的Num值是否等于0，如果是，则记录不存在，访问失败，否则，进入步骤七；

[0031] 步骤七：当所述当前哈希级表索引具有下一哈希级时，确定当前哈希级表索引的下一级哈希表为当前待访问哈希级，并确定字符串集合中的下一字符串为当前待索引字符串，返回执行步骤四，直到索引表访问结束或中断，执行步骤八；

[0032] 步骤八：根据当前哈希级表索引确定目标数据中与其对应的数据，遍历所述数据，并顺序比较所述索引字符串Key的值，如果匹配，则访问成功，否则进入步骤九；

[0033] 步骤九：判断当前结构Pointer指针指向的位置，沿着指针开始逐步比较离散数据，如果匹配，则访问成功，否则，访问失败。

[0034] 优选的，所述内存表的删除操作包括：

[0035] 访问待删除目标记录；

[0036] 当访问到所述待删除目标记录后，将该待删除目标记录对应的最后一级哈希表内的Num值自动减1；

[0037] 当所述待删除目标记录存在于离散数据区时，直接调用客户信息控制系统CICS命令FreeMain释放内存，并设定Pointer的指针指向所述待删除目标记录的上一条记录；

[0038] 当所述待删除目标记录不存在于离散数据区时，将其对应的VAL的值清空为零。

[0039] 优选的，所述内存表的插入过程包括：

[0040] 当主机采用联机类交易处理状态时，所述新增内存表数据的过程包括：

[0041] 设置标志位的状态为更新状态，调用预先建立的内存表锁保护程序，所述标识位为预先在所述内存区域中保留的预设个数的字节；

[0042] 判断目标数据区域内是否存在已被清空的记录项，若存在，则将所述待插入数据填充至所述记录项中VAL项中，并将所述记录项对应的Num的值进行累加操作，若不存在，则CICS命令GETMAIN申请内存空间，将所述待插入数据添加至该内存空间，并将原始的最后一项离散数据的Pointer指针指向新增记录处；

[0043] 设置所述标志位的状态为完成插入。

[0044] 优选的，所述内存表的更新过程包括：

[0045] 当主机采用联机类交易处理状态时，所述更新内存表数据的过程包括：

[0046] 访问待更新目标记录；

[0047] 当访问到所述待更新目标记录后，设置标志位的状态为更新状态，调用预先建立的内存表锁保护程序，所述标识位为预先在所述内存区域中保留的预设个数的字节；

[0048] 更新目标记录中的VAL项的数据，重新计算索引的值，然后更新索引记录；

[0049] 设置所述标志位的状态为完成更新。

[0050] 优选的，所述标志位为：

[0051] 预先在共享内存中保留的预设个数的字节，当所述字节为1时，则表明内存表正处于更新或插入数据的状态，不可读取。

[0052] 优选的，所述内存表锁保护程序为：

[0053] 用于占用系统的时间片而预先建立的循环程序，其中，所述循环程序的周期为主机时钟周期的倍数，所述循环程序的循环次数，根据所述访问操作对应的最长时钟周期而决定。

[0054] 优选的，当主机采用批量类交易处理状态时，所述插入或更新所述内存表数据的过程包括：

[0055] 将待更新或待插入数据记录到预先建立的物理表中；

[0056] 利用批量更新程序从所述物理表中读取各条记录；

[0057] 将所述各条记录中的数据存储于预先设定高级语言源程序内；

[0058] 运行所述程序生成更新后的内存表。

[0059] 本申请实施例公开的内存数据库的实现方法中，内存数据库中的索引和数据均是存储于内存表中，访问过程中减少了内存与磁盘数据交换之间的性能开销，因此，访问性能要高于数据库访问，因此能够进一步的满足应用对访问热表时的性能要求。

附图说明

[0060] 为了更清楚地说明本申请实施例中的技术方案，下面将对实施例描述中所需要使用的附图作简单地介绍，显而易见地，下面描述中的附图仅仅是本申请的一些实施例，对于本领域普通技术人员来讲，在不付出创造性劳动性的前提下，还可以根据这些附图获得其他的附图。

[0061] 图1是本申请实施例公开的一种主机系统下的内存数据库的实现方法流程图；

[0062] 图2是本申请实施例公开的Load Module文件的数据段实例；

[0063] 图3是本申请实施例公开的内存表的结构示意图；

[0064] 图4是本申请实施例公开的内存表优化方法流程图；

[0065] 图5是本申请实施例公开的内存表查找过程的流程图；

[0066] 图6是本申请实施例公开的内存表删除过程的流程图；

[0067] 图7是本申请实施例公开的内存表插入过程的流程图；

[0068] 图8是本申请实施例公开的内存表更新过程的流程图；

[0069] 图9是本申请实施例公开的变更内存表过程的流程图。

具体实施方式

[0070] 下面将结合本申请实施例中的附图，对本申请实施例中的技术方案进行清楚、完整地描述，显然，所描述的实施例仅仅是本申请一部分实施例，而不是全部的实施例。基于本申请中的实施例，本领域普通技术人员在没有做出创造性劳动前提下所获得的所有其他实施例，都属于本申请保护的范围。

[0071] 目前，大型商业银行仍主要以IBM主机系统，处理行内交易。基于此种现状，本发明提出了一种主机系统下的内存数据库的实现方法，以解决现有的数据访问效率并不能满足应用对访问热表时的性能要求的问题。

[0072] 本申请实施例公开的一种主机系统下的内存数据库的实现方法流程如图1所示，包括：

[0073] 步骤S11：对初始数据进行数据整理。

[0074] 初始数据是被导入内存数据库的目标数据，如热表数据等，还做为用于构建内存表的基础数据。数据整理过程是将初始数据整合后存入物理表；

[0075] 所述物理表是一张用于收集、存储预加载至内存中的小表或记录的静态表。在生成内存数据库之前，所有的初始数据都是离散存储的，数据整理的过程就是将用于生成内存数据库的离散数据存储到预先建立的物理表中，由于归集的目标记录的各字段在长度、类型上均可能不一致，因此，本申请实施例中将物理表定义为如下表1所示结构，包含三个字段，且均为char型，并占用足够长度，而所能支持的最大记录条数则需根据应用的需求以及客户信息控制系统区域CICS Region支持的最大内存而定。

[0076] 表1物理表结构

[0077]

	键(Key)	Val(值)
字节数	Char[*]	Char[*]
是否主键	主键	否

[0078] 步骤S12：将进行整理后的数据，进行哈希计算，将哈希计算的结果按照内存数据库中预先建立的内存表规则进行建立，生成内存表源文件，组织生成内存表；

[0079] 整理后的数据，即添加入物理表中的数据记录。

[0080] 所述预先建立的内存表规则为，内存表为N+1级哈希结构，其中，前N级只存储下一级哈希表所在位置，即索引，第N+1级用于存储数据。

[0081] 所述预先建立的内存表规则还包括，所述内存表的第N+1级存储的数据包括目标

数据和离散数据，其中所述目标数据为所述数据整理时导入的初始数据，所述离散数据为动态新增的数据。

[0082] 设定所述N+1级哈希表的拆分集合为SkSet{Sk1,Sk2,...,Skn}，其中Ski(i＝1,2,...n)，代表从Key中截取一段连续片段的起始位置，且当Ski所指片段长度超过Key中实际长度时，进行字符补充处理，定义哈希因子集合HSet{H1,H2,...,Hn}，其中元素Hi(i＝1,2,...n)代表每一级计算哈希值时的取余因子，且所述每一级取余因子逐级递减。所述拆分集合SkSet与所述Hset内的各元素，由所述整理后的初始数据动态决定。

[0083] 构建内存表时，如果仅将热表数据简单导入内存，并通过顺序遍历或二分法比较字符串实现访问，并不能充分发挥内存操作的性能。哈希算法是提升访问效率最直接、有效的方法。然而，全散列哈希表尽管访问效率最高，但却过度的占用内存资源，生产上并不允许。因此本实施例中，内存表按照上述方式进行定义。在本申请中，如果是顺序查找或二分查找，算法时间复杂度为O(n)，其中n为条目数目，执行效率很低。而本步骤中，将目标集合进行多级哈希划分，该方法的时间复杂度为O(m)，其中m为最后一级表内条目数，而m值由于上述哈希算法的高离散性，其值远远小于n，使得访问效率获得大幅提升。

[0084] 并且，该种结构实质已通过SkSet和HSet两个集合固化了表结构，降低动态过程中，实时增减数据时造成的性能开销。

[0085] 本申请公开的内存数据库实现方法中，将进行整理后的数据进行哈希计算，将哈希计算的结果按照内存数据库中预先建立的内存表规则建立内存表，生成内存表源文件的过程包括：

[0086] 将所述整理后的数据进行N+1级哈希计算，将得到的N+1级哈希表按照所述内存表规则，将第N+1级用于存储数据，其余N级用于为索引，生成相应的数据结构；

[0087] 组织数据结构按照预先设定的形式生成内存表的高级语言源文件，如下所示为，C语言源文件，其中buffer代表程序的数据段，main函数代表程序的代码段；

```
static (const) char buffer[] = {
        0x1,0x2,.....
    };
```

[0088]
```
void main(void)
{
  return;
}
```

[0089] PC机、服务器上被广泛使用的Windows、Linux系统，它们通常自行提供了API函数来设置共享内存，而它们的实现方式通常是利用的进程间通信实现。然而，主机系统是IBM提供的商用大型机，上面安装着Z/OS系统，且配备CICS中间层等来进行协议转化、程序部署。由于CICS Region的影响，主机系统在CICS Region中定义了多个交易，而上述的共享内存机制并不适用于Z/OS系统。可执行文件的格式与目标处理器平台以及编译系统息息相关，不同系统平台下往往具备不同的格式，但却同出一脉，如Linux平台中的ELF格式、

Windows平台中的PE格式等。可执行程序包含链接后产生的汇编指令的二进制机器码、数组信息以及链接时用于地址分配的符号表、字符串等信息，这些信息根据它们的属性以节(Section)或段(Segment)的形式存储在各种给定长度的区域内，一般包括用于存储二进制指令信息的代码段(.text)；用于存储已初始化全局静态变量和局部静态变量的数据段(.data)；用于存储const类型的只读数据的只读数据段(.rodata)；用于存储未初始化的全局变量和局部静态变量的临时数据段(.bss)；用户自定义段。Load Module是IBM Z/OS系统下可执行文件，尽管目前IBM并未提供其明确的文件结构，但经过一系列验证，它依然由上述各段组成，如代码段B_TEXT、静态数据段@STATIC等。

[0090] 最常见的内存共享方式动态连接库DLL，Z/OS上依然提供支持，但由于加载过程中，每个交易会进行一次数据备份，因此，需要将数据段注入程序的代码段中。这样所有的交易进行共享数据访问时，并未出现副本，减少系统资源开销。基于此原理，构建了上述代码段。

[0091] 步骤S13：加载内存表源文件；

[0092] 内存表是用于将多个应用场景下，频繁访问的小表或记录，如币种表等，直接驻留于CICS Region内，被交易之间共享，以达到提升访问效率的目标。因此，需要将其程序加载并保存在内存区域中，并进行共享。

[0093] PC机、服务器上被广泛使用的Windows、Linux系统，它们通常自行提供了API函数来设置共享内存，而它们的实现方式通常是利用的进程间通信实现。然而，主机系统是IBM提供的商用大型机，上面安装着Z/OS系统，且配备CICS中间层等来进行协议转化、程序部署。由于CICSRegion的影响，主机系统在CICS Region中定义了多个交易，而上述的共享内存机制并不使用于Z/OS系统。

[0094] 在上述实施例中，加载所述内存表源文件的过程为，在主机平台下，调用IBM配套的编译器、汇编器、链接器将上述代码段生成可执行文件，即Load Module。

[0095] 如图2以一个Load Module文件的数据段为例，其中，虚线框体内的四个字节0x2c0ac，代表数据段总长度，而由于程序内的数据段是多个子段组成，且IBM为了尽可能减少可执行程序的大小，采用了数据压缩技术，即如果一个子段内存在一定规模的，连续的数据，那么将对其进行压缩，并拆分为多个节。为此，下图中虚线和实线框体分别代表子段内的一个独立。如虚线框体中的0x0400005c，其中十六进制‘04’代表数据是按照4字节进行对齐，‘5c’代表当前节未被压缩前占用的字节数。虚线框体内的“0x00000218”代表当前节所在子段占用的字节数，而虚线色框体内的“0x0000004c”代表当前节实际占用的字节数，这也就说明该段连续数据后面还需补充0x10个空字节。实线框体代表另一个节，与虚线框体不同的是，其中‘0x00000067’代表当前节之前还需要补充0x67个空字节。

[0096] 代码段B_TEXT在格式上与数据段相同，但不需要进行数据压缩，并添加了函数Label信息，而在本发明中非系统相关的部分只存在main函数，即main函数为本发明中所述Load Module的唯一入口因此，在将数据段拷贝至代码段时，只对main函数所在节进行扩充和覆盖，具体过程如下：

[0097] 展开所述数据段中所述进行压缩后的部分，计算获取数据段的总长度的值；

[0098] 修改所述main函数所在节中代表数据段总长度区域的值，覆盖为所述数据段总长度的值；

[0099] 将所述展开后的数据段的数据，覆盖所述代码段内的数据；

[0100] 由于所述代码段的原始长度，可能小于所述展开后数据段的长度，因此覆盖结束后，需修改数据段的位置及长度。

[0101] 修改完成后，Z/OS操作系统进行加载时，可直接将内存表加载入代码段中。

[0102] 步骤S14：设定下载指令CICS LOAD，当CICS Region内的程序调用内存数据库时，通过CICS LOAD将所述内存表源文件的入口地址发送给所述程序，以作为该程序解析获取内存表地址的依据；

[0103] 通过上述步骤，实现内存共享。共享内存的目的是为了开辟一段可供同一个CICS Region内交易共同访问的内存区域。Z/OS、CICS中分配共享内存存在多种手段，如GETMAIN等。然而，GETMAIN只能用于从堆内申请指定大小的内存空间，但却难以将申请到的内存地址发布给REGION内的其它交易。本实施例中的LOAD命令及其参数如下所示，其中PROGRAM用于指定程序名，SET用于接收返回所申请内存的地址，LENGTH与FLENGTH用于接收所分配内存的长度。ENTRY为所加载程序的入口地址。HOLD标志位用于指明所加载的程序在交易结束后是否释放。首次调用该命令时，交易会将目标程序加载入内存中，而后续再次调用时，如果程序未发生变化则不在进行加载，实现共享内存地址的对外公布。

| LOAD PROGRAM(name) |_SET(ptr-ref)_|

[0104] |_LENGTH(data-area)__|_FLENGTH(data-area)_|

|_ENTRY(ptr-ref)|

|_HOLD_|

[0105] CICS LOAD命令仅是将一个已在中间层定义过的程序，加载入当前的CICS Region内，操作系统析文件结构后将数据段、程序段分别装载入各自的内存区域，因此，Load API命令返回的即是main函数的地址，或原始数据段的入口地址。

[0106] 步骤S15：为内存表建立建立结构化查询语言SQL语句映射。

[0107] 本实施例公开的内存表实现方法中，为了实现对内存表的操作，为其建立了访问、更新、删除、插入等SQL语句，并将其映射为函数。

[0108] 从上述过程可以看出，本申请实施例公开的内存表实现方法中，内存表中的索引和数据均是在内存中构建结构，访问过程中减少了内存与磁盘数据交换之间的性能开销，因此，访问性能要高于内存数据库，因此能够进一步的满足应用对访问热表时的性能要求。

[0109] 并且，本实施例中采用多级哈希计算实现了多级索引机制，相比常规可见的顺序查找或二分查找等具备更高的访问性能。

[0110] 下面以三级哈希表为例，对内存表的结构进行阐述，如图3所示。其中BASE_ADDR代表内存表在内存中的起始地址，索引由前两级哈希表组成，Offset、Num参数均是成对出现，其中Offset代表下一级哈希表于内存的相对地址，Num代表下一级哈希表内的元素数量。假设第一级哈希表中，存在H1个元素，那么第一级哈希表可以下述集合的形式进行表述.

[0111] {{Offset1,Num1},{Offset2,Num2},...,{OffsetH1,NumH1}}；

[0112] Offseti{i＝1,2,...,H1}为该元素指向的下一级哈希表，即第二级哈希表在内存

地址中相对于BASE_ADDR的偏移量。Numi{i＝1.2,...,H1}为第二级哈希表内元素数目。

[0113] 如图3中所示实例，第三级哈希表即为内存表中的目标数据和离散数据部分，它在结构上并不同于索引部分的结构，它由多块结构化的数据组成。其中，VAL代表单条记录的访问结果，Key代表访问该条记录时所用的索引字符串。此处存储Key是因为，在多级哈希中，我们并不能保证，不同的Key最后计算出来的哈希值是否确定不相同，即一定存在哈希碰撞，因此，需要利用Key值进行一次比较校验。Num代表子集合中元素数，每个子集合中还存在指针Pointer用于指向离散数据区域，离散数据内的Pointer用于将离散数据组织起来，这是因为，新增数据记录是利用GETMAIN函数动态分配，内存地址并不连续，所以利用Pointer指向所述存储到离散数据中的新增数据。

[0114] 除上述设计以外，在32位Z/OS操作系统中，虚拟页的大小为4KB，为了进一步提升主机缓存Cache预取的效率，本申请还提出了如图4所示的优化步骤：

[0115] 步骤S41：确保内存表的起始地址BASE_ADDR为4KB的倍数；

[0116] 步骤S42：通过调整索引中各级哈希表的偏移量使得任一连续数据存储在同一虚拟页。

[0117] 而本实施例中，则是通过修改前两级哈希表内的Offset变量，确保任一连续的数据，分布在同一虚拟页上，以减少数据Cache换页的次数。

[0118] 本申请公开的内存表实现方法中，为内存表建立的SQL语句映射包括访问、更新、删除、插入和游标等，下面对每一种SQL语句的实现方法进行详细阐述。

[0119] 如图5所示，为本申请实施例公开的内存表访问过程的流程图，其中包括：

[0120] 步骤S51：定义用于索引的字符串为Key；

[0121] 步骤S52：根据拆分集合SkSet内的元素Sk_1划分Key，得到字符串集合，从集合中获得用于第一级哈希索引的字符串Key1；

[0122] 以3级哈希表为例，该字符串集合中包含(Key1;Key2)

[0123] 步骤S53：确定索引中的第一哈希级为当前待访问哈希级，Key1为当前待索引字符串；

[0124] 步骤S54：调用哈希函数，对当前待索引字符串进行哈希计算，并获得当前哈希值；

[0125] 步骤S55：获得哈希因子集合HSet内的与当前待索引字符串对应的元素，确定对应的当前级哈希表索引为：当前哈希值对当前待索引字符串对应的元素取余的结果；

[0126] 假设结果为2，则当前级哈希表索引为$\{Offset_2,Num_2\}$。

[0127] 步骤S56：判断当前哈希级表索引的参数中的Num值是否等于0，如果是，则记录不存在，访问失败，否则，进入步骤S57；

[0128] 判断Num_2是否为0，也就是说，判断该索引指向的下一级哈希表中的元素个数是否为0，为0，则说明下一级哈希表中没有元素，相应的，也就不存在要访问的数据，所以访问失败。如果不为0，则证明有可能存在要访问的数据，需要进一步的确定。

[0129] 步骤S57：判断当前哈希级表索引是否具有下一哈希级，若是，则执行步骤S58，若否，则执行步骤S59；

[0130] 步骤S58确定当前哈希级表索引的下一级哈希表为当前待访问哈希级，并确定字符串集合中的下一字符串为当前待索引字符串，返回执行步骤S54；

[0131] 如果此时当前哈希级表索引不是最后一级索引，则证明后续还有索引，则按照上

面的过程，访问当前哈希级表索引指向的下一级哈希表。如果当前哈希级表索引不具有下一哈希级，则说明其本身为最后一级索引，因此，索引表的访问结束，进入对数据的访问。如果突发中断则直接进入对数据的访问。

[0132] 步骤S59：根据当前哈希级表索引确定目标数据中与其对应的数据，遍历数据，并顺序比较索引字符串Key的值；

[0133] 步骤S510：判断数据是否与索引字符串Key值匹配，如果匹配，则访问成功，否则进入步骤S511；

[0134] 步骤S511：确定当前结构Pointer指针指向的位置，沿着指针开始逐步比较离散数据；

[0135] 步骤S512：判断离散数据是否与索引字符串Key值匹配，如果匹配，则访问成功，否则访问失败。

[0136] 从上述的访问过程可以看出，本申请实施例公开的方案，采用逐级递减的方式查找目标数据，提高了数据访问的效率。

[0137] 在内存表的操作中，访问操作是最为基础的操作，其他的操作均为在成功执行访问操作后的进一步操作。

[0138] 如图6所示，为内存表的删除操作过程，包括：

[0139] 步骤S61：访问待删除目标记录；

[0140] 步骤S62：当访问到待删除目标记录后，将该待删除目标记录对应的最后一级哈希表内的Num值自动减1；

[0141] 步骤S63：判断待删除目标记录是否存在于离散数据区，若是，则执行步骤S64，若否，则执行步骤S65；

[0142] 步骤S64：直接调用CICS命令FreeMain释放内存，并设定Pointer的指针指向待删除目标记录的上一条记录；

[0143] 步骤S65：将其对应的VAL的值清空为零。

[0144] 该删除过程中，除了删除数据本身以外，必须同时修改与该数据记录对应的索引的Num参数值，否则会导致后续的操作无法正常进行。

[0145] 插入流程与主机当前所调用的交易类型有关，当采用联机类交易处理状态时，该过程如图7所示包括：

[0146] 步骤S71：设置标志位的状态为更新状态，调用预先建立的内存表锁保护程序，标识位为预先在内存区域中保留的预设个数的字节；

[0147] 步骤S72：判断目标数据区域内是否存在已被清空的记录项，若存在，执行步骤S73，若否，则执行步骤S74；

[0148] 步骤S73：将待插入数据填充至记录项中VAL项中，并将记录项对应的Num的值进行累加操作；

[0149] 步骤S74：CICS命令GETMAIN申请内存空间，将待插入数据添加至该内存空间，并将原始的最后一项离散数据的Pointer指针指向新增记录处；

[0150] 步骤S75：设置存储标志位的状态为完成插入。

[0151] 在同样的主机环境下，内存表的更新过程如图8所示，包括：

[0152] 步骤S81：访问待更新目标记录；

[0153] 步骤S82：当访问到待更新目标记录后，设置标志位的状态为更新状态，调用预先建立的内存表锁保护程序，标识位为预先在内存区域中保留的预设个数的字节；

[0154] 步骤S83：更新目标记录中的VAL项的数据，重新计算索引的值，然后更新索引记录；

[0155] 步骤S84：设置标志位的状态为完成更新。

[0156] 上述两个操作的具体实现过程，能够实现当用户提出向内存表中提出插入或更新数据的需求时，可实时的向内存表中更新数据。而由于多笔交易在CICS Region内是并发执行的，当一个用户更新内存表数据时，可能导致其它用户的访问操作出现异常。为此，动态更新方案下，本申请设置了标志位以及内存表锁保护程序。该标志位为预先在共享内存中保留的预设个数的字节，例如四个，当该四个字节为1时，则表明内存表正处于更新或插入数据的状态，不可读取。

[0157] 由于并发操作的影响，当主机设置标志位为1时，某些交易的访问操作还未结束，则主机自动调用一个预先建立的循环程序，以占用系统的时间片，待交易访问完成后，再执行插入或者更新操作。该循环程序经过编译后，生成3条逻辑运算指令，1条分支指令，根据主机处理器的体系结构，这段代码占用的硬件时钟周期为主机时钟周期的倍数，至多6个cycle，而本申请根据每次访问操作对应的最长时钟周期假设为MAX_CYCLE，那么可将Time_SPLICE设置为MAX_CYCLE*2/6，即可确保并发时钟要求。

[0158] #define Time_Splice MAX_CYCLE*2/6

[0159] for(int i＝0;i<Time_Splice;i++)

[0160] {

[0161] }

[0162] 当采用批量类交易处理状态时，更新内存表内的数据或者向内存表中插入数据都可以看做是对内存表的变更，而变更内存表的过程如图9所示，包括：

[0163] 步骤S91：将待更新或待插入数据记录到预先建立的物理表中；

[0164] 当有应用发起变更内存表的请求时，将对内存表的变更首先作用在物理表中，即，在物理表中更新与内存表对应的数据，或者插入新的数据。一旦提出更新物理表的变更请求，调起后备程序，将物理表内的数据将重新导入内存。

[0165] 步骤S92：利用批量更新程序从物理表中读取各条记录；

[0166] 步骤S93：将各条记录中的数据存储于预先设定高级语言源程序内；

[0167] 此处的高级语言可以为C语言或者汇编语言等。

[0168] 步骤S94：运行程序生成更新后的内存表。

[0169] 从上述情况来看，无论是插入还是更新，都先将要插入或者更新的数据添加或者更新至物理表，然后将物理表作为一个整体，重新生成内存表，全部更新至内存表内。无需针对每一次插入或者更新的需求对内存表进行操作。不会向内存表的离散数据区增加数据，全部的数据均加入目标数据区。

[0170] 联机整理方案中各应用均具备修改共享内存的权限，而批量整理方案，尽管更新内存时流程较多，但其优势却在于屏蔽了应用对共享内存的进行写操作，而读操作并不会造成系统级的破坏。

[0171] 除上述操作外，还包括游标操作，为了实现游标操作，本申请提出的技术方案中构

建了上述三个函数，void cstart(const char*Key)；int cursor(const char*Cond,...)；void cend(void)。其中，cstart函数根据Key的长度、Skset内的元素，判断将Key切割为k个子片段，也就是说访问到哈希表的第k级，并将所在表项在内存表中的偏移量存储于全局静态变量_curosr_start内；cursor函数读取_curosr_start并根据cond判断当前记录，执行结束前，将_curosr_start设置为下一条记录的偏移量，如果全部的记录都已经遍历完成，那么cursor函数返回0，否则为1；cend函数用

[0172] 于将_curosr_start置为-1。如下是使用游标函数的一个示例。

[0173]
```
Cstart();
for (;;)
{
    int flag = Cursor(,,,);
    if (flag == 0)
        break;
}
  Cend();
```

[0174] 除上述SQL语句外，其他的SQL语句在此不再一一描述。

[0175] 本申请实施例公开的主机系统下的内存数据库的实现方法，内存表中的索引和数据均是在内存中构建结构，访问过程中减少了内存与磁盘数据交换之间的性能开销，因此，访问性能要高于内存数据库，因此能够进一步的满足应用对访问热表时的性能要求。

[0176] 需要说明的是，本说明书中的各个实施例均采用递进的方式描述，每个实施例重点说明的都是与其他实施例的不同之处，各个实施例之间相同相似的部分互相参见即可。对于装置类实施例而言，由于其与方法实施例基本相似，所以描述的比较简单，相关之处参见方法实施例的部分说明即可。

[0177] 最后，还需要说明的是，在本文中，诸如第一和第二等之类的关系术语仅仅用来将一个实体或者操作与另一个实体或操作区分开来，而不一定要求或者暗示这些实体或操作之间存在任何这种实际的关系或者顺序。而且，术语“包括”、“包含”或者其任何其他变体意在涵盖非排他性的包含，从而使得包括一系列要素的过程、方法、物品或者设备不仅包括那些要素，而且还包括没有明确列出的其他要素，或者是还包括为这种过程、方法、物品或者设备所固有的要素。在没有更多限制的情况下，由语句“包括一个……”限定的要素，并不排除在包括要素的过程、方法、物品或者设备中还存在另外的相同要素。

[0178] 为了描述的方便，描述以上装置时以功能分为各种单元分别描述。当然，在实施本申请时可以把各单元的功能在同一个或多个软件和/或硬件中实现。

[0179] 通过以上的实施方式的描述可知，本领域的技术人员可以清楚地了解到本申请可借助软件加必需的通用硬件平台的方式来实现。基于这样的理解，本申请的技术方案本质上或者说对现有技术做出贡献的部分可以以软件产品的形式体现出来，该计算机软件产品可以存储在存储介质中，如ROM/RAM、磁碟、光盘等，包括若干指令用以使得一台计算机设备

(可以是个人计算机,服务器,或者网络设备等)执行本申请各个实施例或者实施例的某些部分所述的方法。

[0180] 以上对本申请所提供的一种主机系统下的内存数据库的实现方法进行了详细介绍,本文中应用了具体个例对本申请的原理及实施方式进行了阐述,以上实施例的说明只是用于帮助理解本申请的方法及其核心思想;同时,对于本领域的一般技术人员,依据本申请的思想,在具体实施方式及应用范围上均会有改变之处,综上所述,本说明书内容不应理解为对本申请的限制。

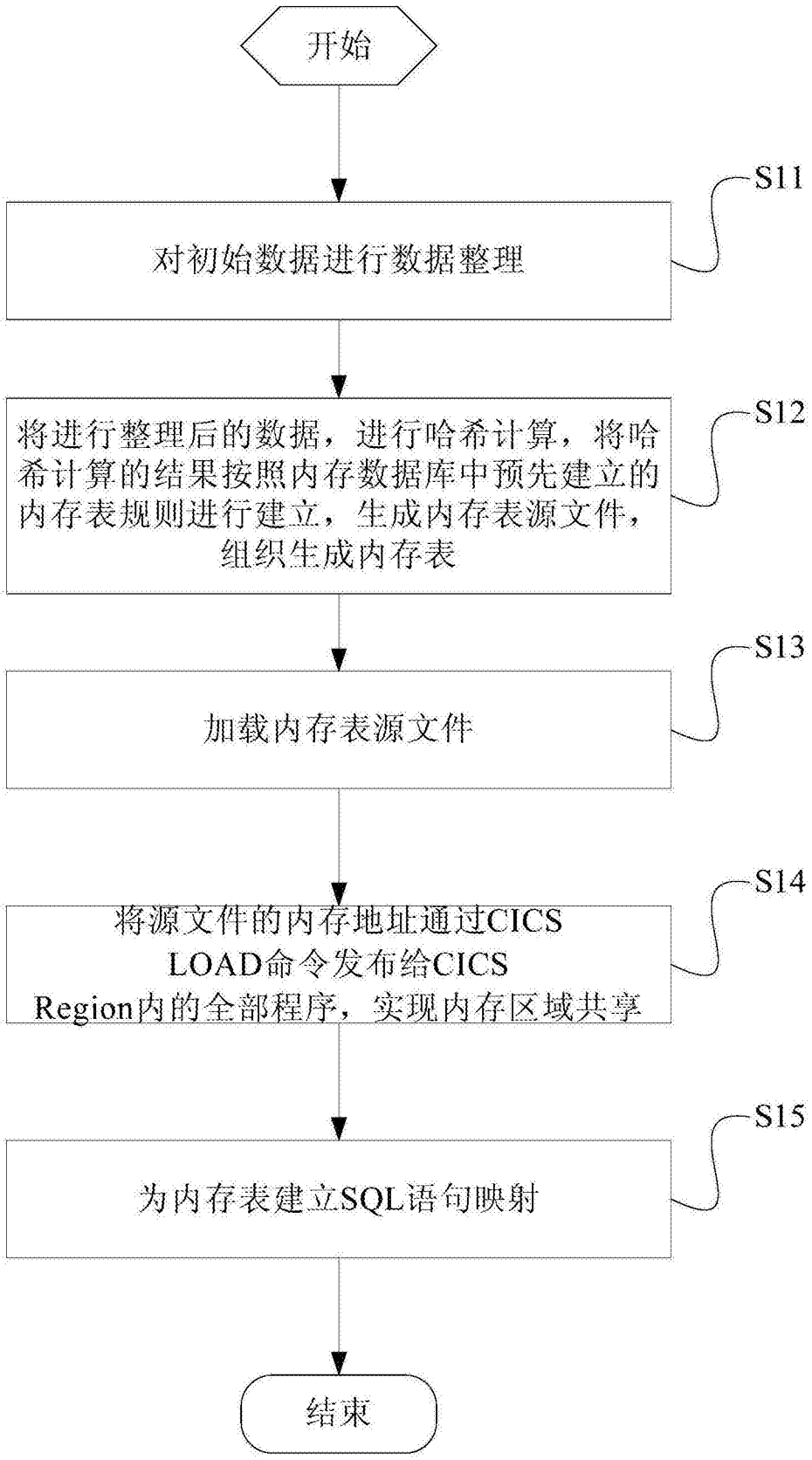

开始
S11
对初始数据进行数据整理
S12
将进行整理后的数据，进行哈希计算，将哈希计算的结果按照内存数据库中预先建立的内存表规则进行建立，生成内存表源文件，组织生成内存表
S13
加载内存表源文件
S14
将源文件的内存地址通过CICS LOAD命令发布给CICS Region内的全部程序，实现内存区域共享
S15
为内存表建立SQL语句映射
结束

图1

```
TRANSACTION: CMM PROGRAM:SCLOAD1 TAS\K:0034537 APPLID:CIJ2B5AI
1AC282F0 000000 0002C0AC 00000000 0400005C 00000218
1AC28300 000010 00000000 0000004C FFFFFFFF 00000001
1AC28310 000020 00003008 00000000 00000000 00000001
1AC28320 000030 00003048 00000001 00003088 00000002
1AC28330 000040 000030C8 00000001 000031D8 00000001
1AC28340 000050 00003368 00000002 00003258 00000002
1AC28350 000060 00003218 0400000FC 00000218 00000067
1AC28360 000070 0000008D 02000034 78000000 00000000
1AC28370 000080 00000000 02000035 88000000 01000036
1AC28380 000090 98000000 02000036 D8000000 01000037
1AC28390 0000A0 E8000000 02000038 28000000 01000039
1AC283A0 0000B0 38000000 01000039 78000000 01000039
1AC283B0 0000C0 B8000000 01000039 F8000000 0400003A
1AC283C0 0000D0 38000000 0100003C 58000000 0100003C
1AC283D0 0000E0 98000000 00000000 00000000 0200003C
1AC283E0 0000F0 D8000000 0100003D E8000000 0100003E
```

图2

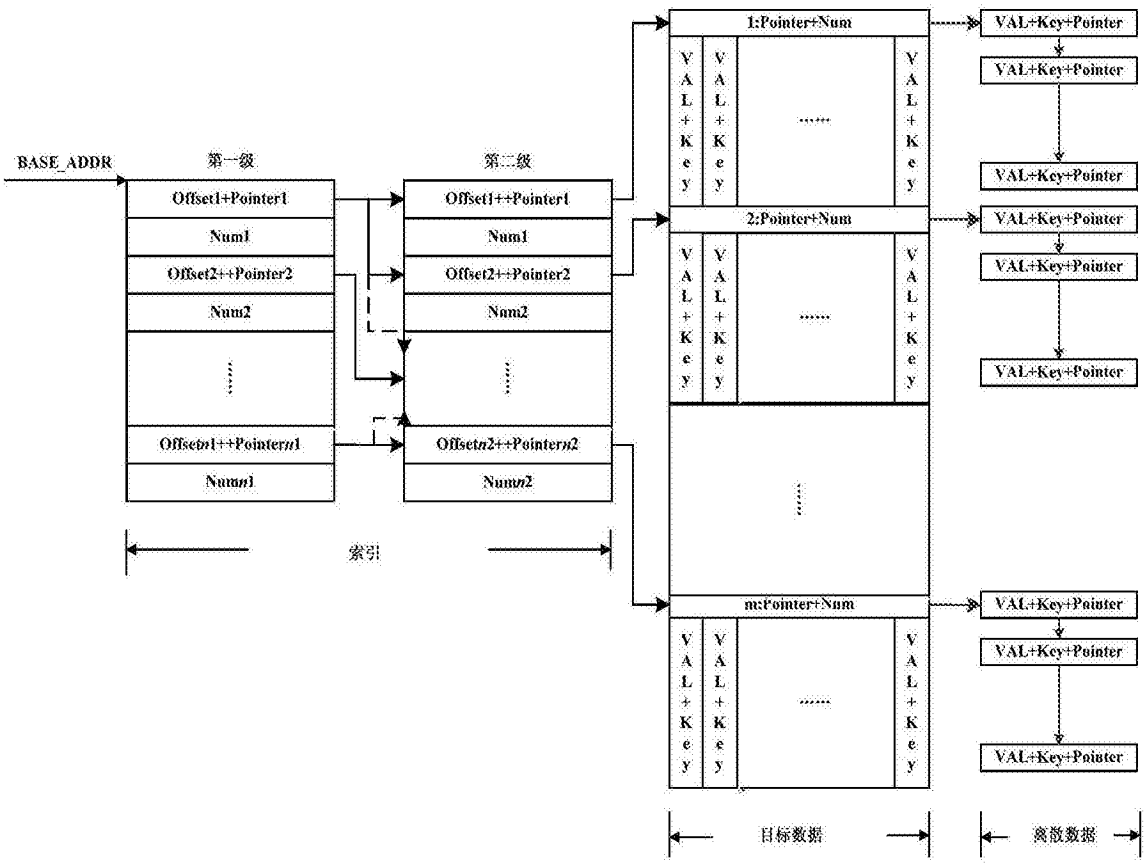


图3

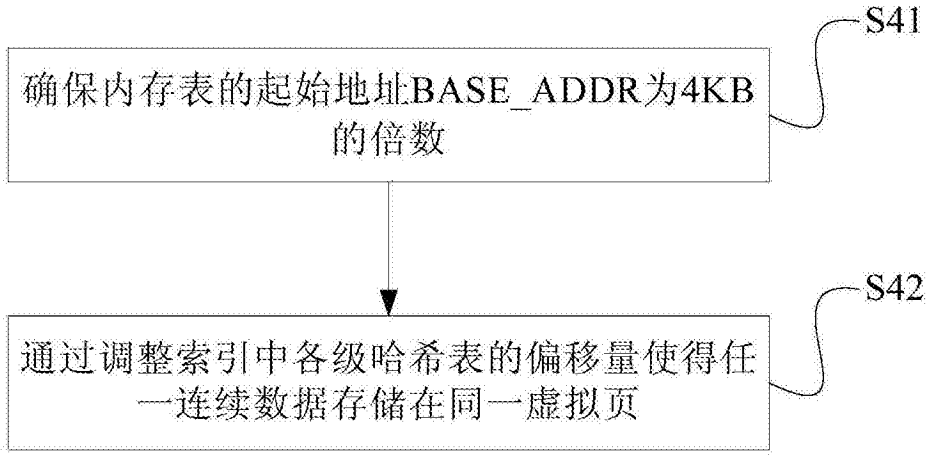

S41
确保内存表的起始地址BASE_ADDR为4KB的倍数
S42
通过调整索引中各级哈希表的偏移量使得任一连续数据存储在同一虚拟页

图4

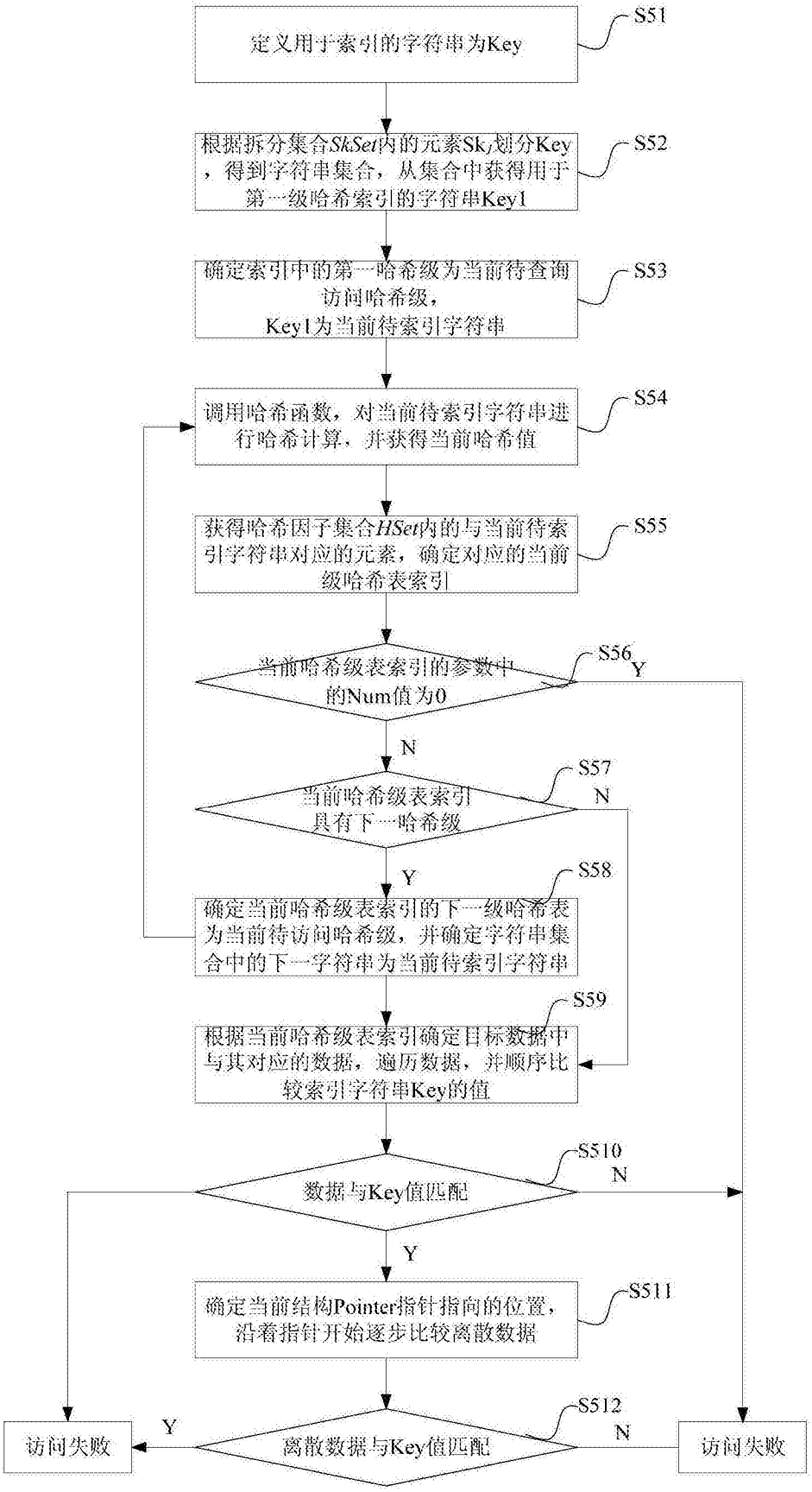

定义用于索引的字符串为Key
S51
根据拆分集合SkSet内的元素Sk划分Key，得到字符串集合，从集合中获得用于第一级哈希索引的字符串Key1
S52
确定索引中的第一哈希级为当前待查询访问哈希级，Key1为当前待索引字符串
S53
调用哈希函数，对当前待索引字符串进行哈希计算，并获得当前哈希值
S54
获得哈希因子集合HSet内的与当前待索引字符串对应的元素，确定对应的当前级哈希表索引
S55
当前哈希级表索引的参数中的Num值为0
S56
Y
N
当前哈希级表索引具有下一哈希级
S57
N
Y
S58
确定当前哈希级表索引的下一级哈希表为当前待访问哈希级，并确定字符串集合中的下一字符串为当前待索引字符串
S59
根据当前哈希级表索引确定目标数据中与其对应的数据，遍历数据，并顺序比较索引字符串Key的值
数据与Key值匹配
S510
N
Y
确定当前结构Pointer指针指向的位置，沿着指针开始逐步比较离散数据
S511
访问失败
Y
离散数据与Key值匹配
S512
N
访问失败

图5

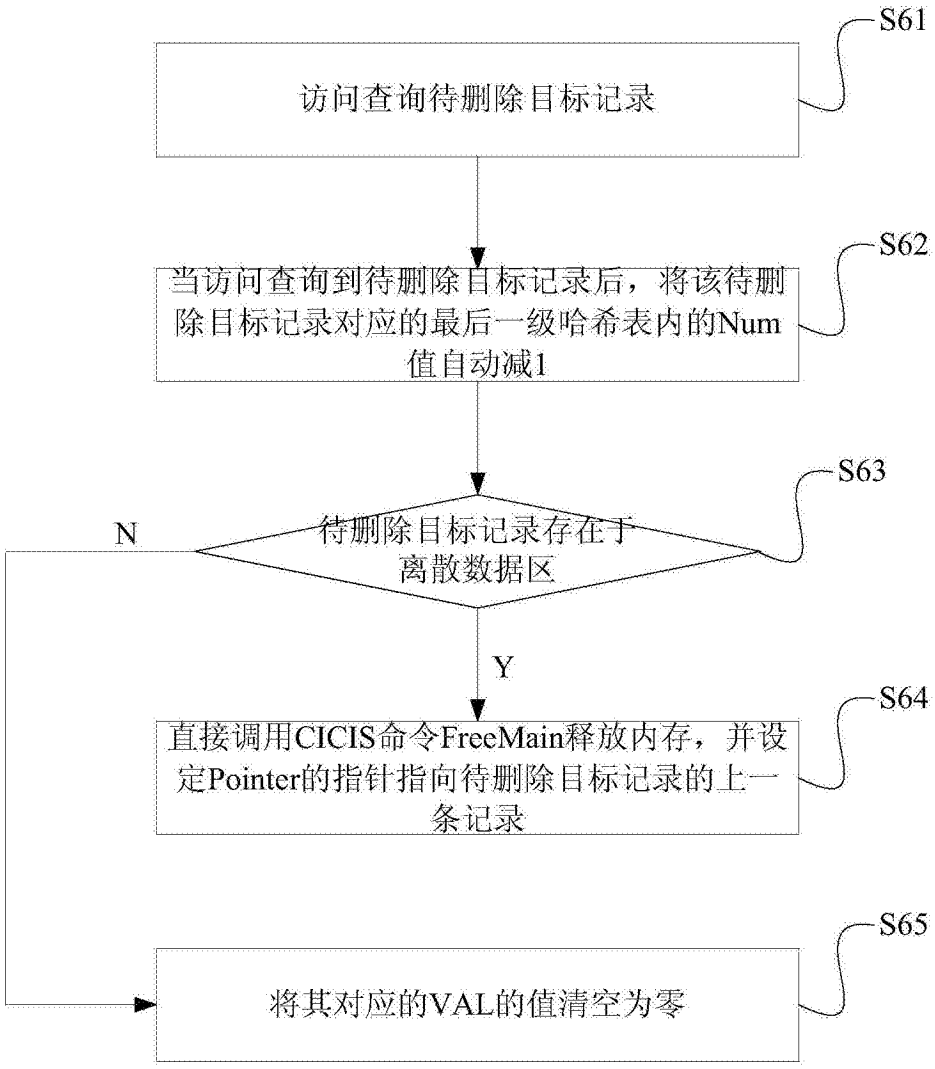

S61
访问查询待删除目标记录
S62
当访问查询到待删除目标记录后，将该待删除目标记录对应的最后一级哈希表内的Num值自动减1
S63
待删除目标记录存在于离散数据区
N
Y
S64
直接调用CICIS命令FreeMain释放内存，并设定Pointer的指针指向待删除目标记录的上一条记录
S65
将其对应的VAL的值清空为零

图6

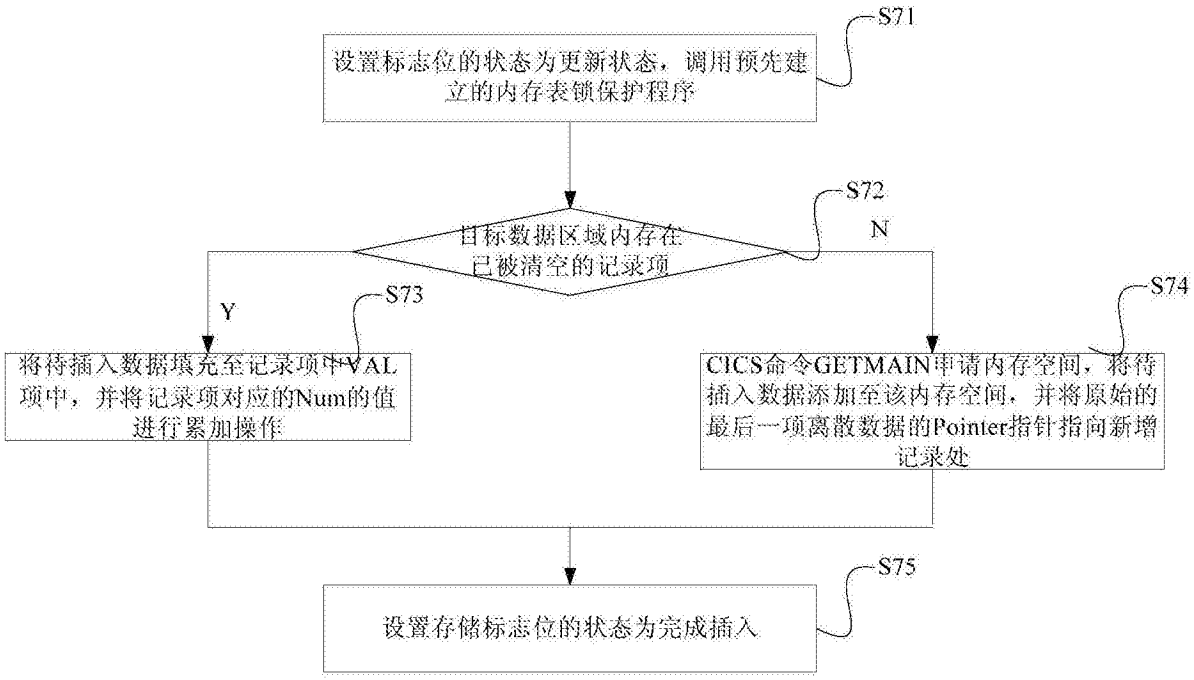

S71
设置标志位的状态为更新状态，调用预先建立的内存表锁保护程序
S72
目标数据区域内存在已被清空的记录项
Y
N
S73
将待插入数据填充至记录项中VAL项中，并将记录项对应的Num的值进行累加操作
S74
CICS命令GETMAIN申请内存空间，将待插入数据添加至该内存空间，并将原始的最后一项离散数据的Pointer指针指向新增记录处
S75
设置存储标志位的状态为完成插入

图7

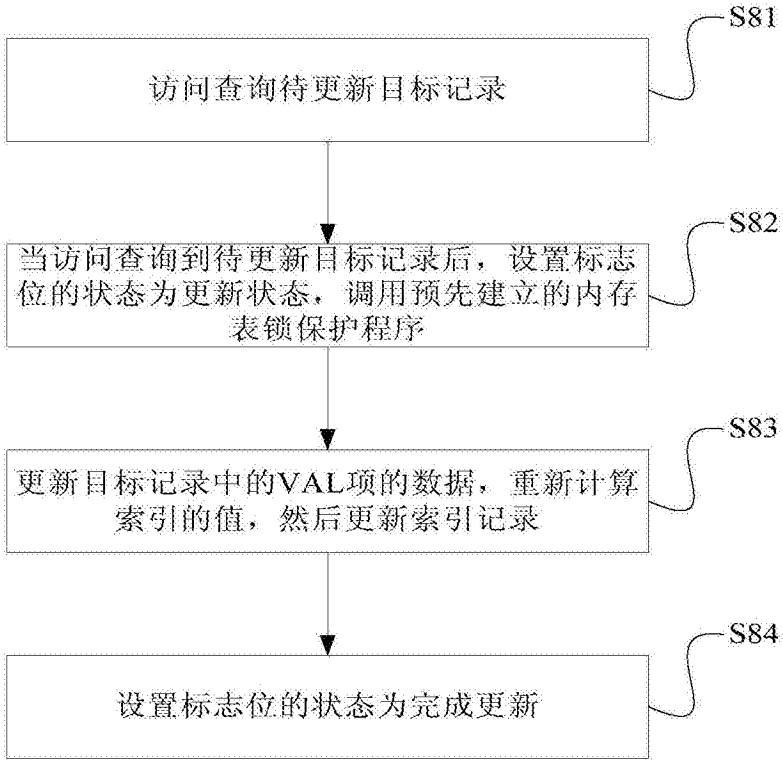

S81
访问查询待更新目标记录
S82
当访问查询到待更新目标记录后，设置标志位的状态为更新状态，调用预先建立的内存表锁保护程序
S83
更新目标记录中的VAL项的数据，重新计算索引的值，然后更新索引记录
S84
设置标志位的状态为完成更新

图8

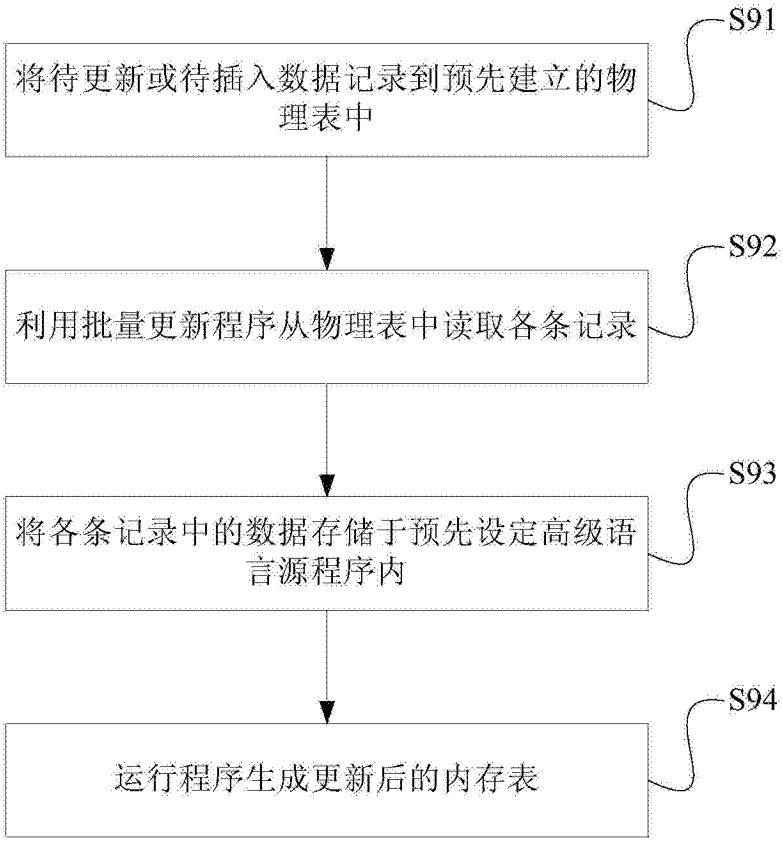

S91
将待更新或待插入数据记录到预先建立的物理表中
S92
利用批量更新程序从物理表中读取各条记录
S93
将各条记录中的数据存储于预先设定高级语言源程序内
S94
运行程序生成更新后的内存表

图9