(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 103460208 B
(45)授权公告日 2017.04.05

(21)申请号 201280017308.6
(22)申请日 2012.03.02
(65)同一申请的已公布的文献号
申请公布号 CN 103460208 A
(43)申请公布日 2013.12.18
(30)优先权数据
13/082,829 2011.04.08 US
(85)PCT国际申请进入国家阶段日
2013.10.08
(86)PCT国际申请的申请数据
PCT/US2012/027417 2012.03.02
(87)PCT国际申请的公布数据
WO2012/138437 EN 2012.10.11
(73)专利权人 波音公司
地址 美国伊利诺伊州
(72)发明人 I·A·约翰逊
(74)专利代理机构 北京纪凯知识产权代理有限公司 11245
代理人 赵蓉民
(51)Int.Cl.
G06F 17/30(2006.01)
(56)对比文件
US 2009299987 A1,2009.12.03,
CN 1575464 A,2005.02.02,
CN 102142039 A,2011.08.03,
CN 1426003 A,2003.06.25,
xgy1522.数据仓库超级大表分区方式改变.《http://blog.csdn.net/xqy1522/article/details/5489823》.2010,第1-4页.
审查员 梁静静

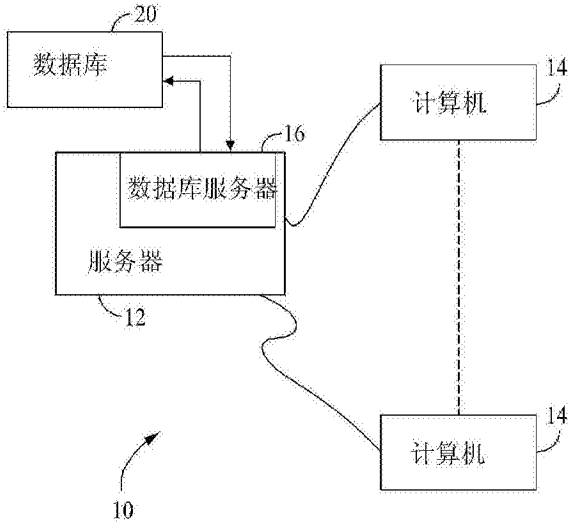
权利要求书2页 说明书40页 附图23页

(54)发明名称

用于将数据加载到时态数据仓库的方法和系统

(57)摘要

披露的系统包括时态数据仓库和可操作从而在该系统上运行的平台无关的时态数据仓库加载应用程序。加载应用程序将源自进入数据的时间戳数据与集合运算符的关系代数一起使用，从而识别在进入数据和先前存储在数据仓库内的数据之间的净变化并将其排序。加载应用程序以对数据仓库正常操作相对小的干扰将已识别并排序的净变化加载到数据仓库。优化方案，包括但不限于将工作负载不同分区到并行流是经元数据可选择的。



CN 103460208 B

1.一种用于将进入数据集加载到时态数据仓库的系统(10,22,700),所述系统包含:

存储装置(720),所述存储装置(720)包括时态数据仓库和进入数据集;以及

处理器单元(710),所述处理器单元(710)耦合到所述存储装置(720)并且经编程:

将所述进入数据集划分为包括第一分区和第二分区的多个分区,其中所述多个分区中的每个分区包括多个数据记录;

将所述第一分区输入到预加载表;

将所述第二分区输入到所述预加载表;以及

将所述预加载表应用到所述时态数据仓库;

其中进入数据包括源自源数据库(20)的数据的快照,并且所述处理器单元(710)经进一步编程:

检测在时态数据仓库中的有效数据记录不与在所述进入数据集中的数据记录关联;

基于所述检测执行所述有效数据记录的隐式删除;

确定与在所述进入数据集中的第一数据记录关联的最早源时间戳;以及

识别表示以下内容的一主关键字组:

与刚好在所述最早源时间戳之前的源时间戳关联的在所述时态数据仓库中的数据记录;以及

与比最早源时间戳晚的源时间戳关联的在所述时态数据仓库中的一个或更多数据记录;以及

基于已识别的主关键字组输入所述第一分区和所述第二分区,其中所述分区排除每个主关键字的所有较早历史数据记录。

2.根据权利要求1所述的系统(10,22,700),其中所述处理器单元(710)经编程至少部分通过将哈希函数应用到与至少一个数据记录关联的主关键字以便产生对应于所述至少一个数据记录的哈希值,从而将所述进入数据集划分为所述多个分区。

3.根据权利要求1所述的系统(10,22,700),其中所述处理器单元(710)经进一步编程在所述第一分区被预加载到所述预加载表之后,将所述第二分区输入到所述预加载表。

4.根据权利要求1所述的系统(10,22,700),其中所述处理器单元(710)经进一步编程在所述第一分区被输入到所述预加载表时,将所述第二分区输入到所述预加载表。

5.根据权利要求4所述的系统(10,22,700),其中所述处理器单元(710)经编程基于确定并行输入的当前量小于并行输入的预定最大量,在所述第一分区被输入到所述预加载表时将所述第二分区输入到所述预加载表。

6.根据权利要求1所述的系统(10,22,700),其中所述处理器单元(710)经编程至少部分通过以下措施输入所述第一分区和所述第二分区中的至少一个:

将所述第一分区和所述第二分区中的至少一个的数据记录输入到可变表,所述可变表对应于所述第一分区和所述第二分区中的至少一个;以及

将源自所述可变表的所述数据记录复制到所述预加载表。

7.根据权利要求1所述的系统(10,22,700),其中所述处理器单元(710)经进一步编程:

在所述第一分区中识别包括除时间戳之外的多个字段的数据记录,所述多个字段等于先前输入的数据记录的多个非关键字段;

在将所述第一分区输入到所述预加载表时,排除已识别的数据记录。

8.一种用于将多个数据记录加载到时态数据仓库的方法，所述方法包含：

由计算装置(16,24,700)将所述数据记录划分为包括第一分区和第二分区的多个分区；

由所述计算装置(16,24,700)将所述第一分区输入到预加载表；

由所述计算装置(16,24,700)将所述第二分区输入到所述预加载表；以及

由所述计算装置(16,24,700)将所述预加载表应用到所述时态数据仓库；

其中进入数据包括源自源数据库的数据的快照，并且所述方法进一步包括：

检测在时态数据仓库中的有效数据记录不与在所述进入数据集中的数据记录关联；

基于所述检测执行所述有效数据记录的隐式删除；

确定与在所述进入数据集中的第一数据记录关联的最早源时间戳；以及

识别表示以下内容的一主关键字组：

与刚好在所述最早源时间戳之前的源时间戳关联的在所述时态数据仓库中的数据记录；以及

与比最早源时间戳晚的源时间戳关联的在所述时态数据仓库中的一个或更多数据记录；以及

基于已识别的主关键字组输入所述第一分区和所述第二分区，其中所述分区排除每个主关键字的所有较早历史数据记录。

9.根据权利要求8所述的方法，其中所述第一分区和所述第二分区被并行输入。

10.根据权利要求8所述的方法，进一步包含确定并行输入的当前量小于并行输入的预定最大量，其中基于所述确定将所述第一分区和所述第二分区并行输入。

11.根据权利要求8所述的方法，进一步包含确定并行输入的当前量大于或等于并行输入的预定最大量，其中基于所述确定将所述第一分区和所述第二分区循序输入。

12.根据权利要求8所述的方法，其中由所述计算装置(16,24,700)将所述数据划分为多个分区包含：

将哈希函数应用到至少一个数据记录从而产生与所述至少一个数据记录关联的哈希值；以及

基于分区的预定量将模运算符应用到所述哈希值，从而确定与所述至少一个数据记录对应并关联的分区数目。

13.根据权利要求8所述的方法，进一步包含：

在所述第一分区中识别包括除时间戳之外的多个字段的所述数据记录，所述多个字段等于先前输入数据记录的多个非关键字段；

在将所述第一分区输入到所述预加载表时，排除已识别的数据记录。

用于将数据加载到时态数据仓库的方法和系统

技术领域

[0001] 本披露的领域一般涉及计算机数据仓库(CDW),并且更具体涉及用于时态规格化数据仓库的元数据驱动数据捕捉的方法和系统。

背景技术

[0002] 存在不借助于顺序法的用单个通用设计的进入数据的快速加载和时间序列变化量的需要。顺序法一般不是用于初始化和与较高量进入数据事件一起使用的有效措施。另外,具有无关于接口类型,有时减少检测在数据内的变动的密集预处理,和/或确保唯一有效时限以使得能够为每个目标表格产生候选行的加载设定的需要。最终,因为与数据存储关联的成本,所以具有识别所有类型的数据变动并避免在授权时间戳(有效时间)之后加载没有新内容的新数据行。这样的实践可以通过在时限内压缩连续重复行的数据帮助减少存储使用。

[0003] 当前,通常在大型外部应用服务器上运行的复杂定制数据加载程序是已在加载时态数据仓库的尝试中实施的解决方案。这样的程序通过主关键字串行处理并应用数据,这可以导致长运行时间和昂贵的、对目标表格相对干扰的更新。在一些实例中,为连续支持用户,两组目标表格使用并在加载完成时交换。然而在这样的系统中,通常已在数据库中的一些数据移除,与进入数据一起在应用服务器上外部处理并且重加载从而实现数据加载,这对网络和数据库进一步施压。其他已知现有解决方案也仅趋向于预期的解决方案而不是所有可能解决方案,在未预期情况(例如在主关键字内的有效时间连接点(time tie))下破坏、放弃加载或拒绝数据。

[0004] 其他已设想解决方案一般具有其他缺点。例如,由于开发成本,因此被硬编码从而接受特别类型进入数据并且提取目标模式(schemas)的设计是不希望的。进一步地,维护成本可以关于寻址主关键字或属性何时改变成接口的数据源、数据目标或方法。提取、变换和加载(ETL)工具用来在服务器上数据库之外执行工作是一个可能的解决方案,但低效并可以受网络流量影响。当使用在由数据仓库广泛使用的大规模并行处理(MPP)架构上使用外部的或每次一行的解决方案时,在已设想解决方案中效率损失是特别巨大的。同样,私有数据库工具需要专门知识并且不可移植到其他平台(例如Oracle PL/SQL)。这些解决方案对于可以近实时提出的较大量数据是低效的,不可能非侵入加载并且为初始化或大量数据需要不同编码以实现可接受的性能。

发明内容

[0005] 在一个方面中,提供用于将进入数据集加载到时态数据仓库(temporal data warehouse)的系统。该系统包括存储装置和耦合到存储装置的处理器单元。存储装置包括时态数据仓库和进入数据集。处理器单元经编程将进入数据集划分为包括第一分区和第二分区的多个分区(partition)。多个分区中的每个分区包括多个数据记录。处理器也经编程将第一分区输入到预加载表、将第二分区输入到预加载表并将预加载表应用到时态数据仓

库。

[0006] 在另一方面中，提供用于将进入数据集加载到时态数据仓库的方法。该方法包括由计算装置将数据记录划分为包括第一分区和第二分区的多个分区。第一分区和第二分区由计算装置输入到预加载表。预加载表由计算装置应用到时态数据仓库。

[0007] 在更另一方面中，提供计算机程序产品。计算机程序产品包括非暂时性计算机可读介质，其具有在其上实施的用于以净变化数据(net change data)加载到数据仓库的计算机可执行指令。当由至少一个处理器执行时，计算机可执行指令导致处理器将进入数据集划分为包括第一分区和第二分区的多个分区。多个分区中的每个分区包括多个数据记录。计算机可执行指令也导致将第一分区输入到预加载表、将第二分区输入到预加载表并将预加载表应用到数据仓库。

附图说明

[0008] 图1是计算机系统的简化框图。

[0009] 图2是计算机网络的框图。

[0010] 图3是图解示范变动数据捕捉过程的流程图。

[0011] 图4是图解示范分区加载过程的流程图。

[0012] 图5是图解示范数据应用过程的流程图。

[0013] 图6是与在图4中示出的步骤100关联的数据流图。

[0014] 图7是与在图4中示出的步骤101关联的数据流图。

[0015] 图8是与在图4中示出的步骤102关联的数据流图。

[0016] 图9是与在图4中示出的步骤103关联的数据流图。

[0017] 图10是与在图4中示出的步骤104关联的数据流图。

[0018] 图11是与在图4中示出的步骤105关联的数据流图。

[0019] 图12是与在图4中示出的步骤106关联的数据流图。

[0020] 图13是与在图4中示出的步骤107关联的数据流图。

[0021] 图14是与在图4中示出的步骤108关联的数据流图。

[0022] 图15是与在图4中示出的步骤109关联的数据流图。

[0023] 图16是与在图4中示出的步骤110关联的数据流图。

[0024] 图17是与在图4中示出的步骤111关联的数据流图。

[0025] 图18是与在图4中示出的步骤112关联的数据流图。

[0026] 图19是与在图5中示出的应用步骤202关联的数据流图。

[0027] 图20是与在图5中示出的应用步骤203关联的数据流图。

[0028] 图21是与在图5中示出的应用步骤204关联的数据流图。

[0029] 图22是与在图5中示出的应用步骤205关联的数据流图。

[0030] 图23是与在图5中示出的应用步骤206关联的数据流图。

[0031] 图24是示范计算装置的框图。

具体实施方式

[0032] 实施例在此关于变动数据捕捉(CDC)过程描述。如在此使用，“CDC”指代捕捉变动

并将其应用到时态数据仓库的过程。到CDC过程的输入、进入数据集可以已变换从而匹配目标仓库的数据模型(例如规格化的、业务的或自然的关键字),但没有时态处理例如时间排序、时态规格化和/或解析时态碰撞。进入数据集可以已经加载到数据库系统,以使其可由CDC过程直接访问。

[0033] 本披露可以在计算机码或机器可用指令的一般背景下描述,该计算机码或机器可用指令包括由计算机或其他机器例如个人数据助理或其他手持装置执行的计算机可执行指令例如程序模块。一般地,包括路线、程序、对象、部件、数据结构等的程序模块涉及执行特别任务或实施特别抽象数据类型的代码。本披露可以在各种系统配置中实践,包括手持装置、消费者电子设备、通用计算机、更专用的计算装置等。本披露也可以在分布式计算环境中实践,其中任务由通过通信网络链接的远程处理装置执行。

[0034] 所描述系统可操作从而关于一组进入数据自身和现有数据仓库分析可以称为进入数据集的该组进入数据,使用关系代数集运算符与已存储在数据仓库内的数据比较将净变化数据识别并排序,并向数据仓库应用更新。进入数据集包括可以表现源数据库的快照的多个数据记录(例如在时间上某点的在数据库中所有数据记录)和/或已对源数据库执行的多个消息或事务(例如插入、更新和/或删除)。

[0035] 为实现这样的方法,对应于数据仓库的软件代码例如结构化查询语言(SQL)代码可以在这里描述的软件构建(例如编译)时、在软件部署时和/或在元数据(例如数据库结构)修订时生成。每当时间数据加载到数据仓库时,已生成代码可以然后由系统执行。在一些实施例中,已生成代码由将已存储代码存储在数据库中的一个或更多已存储规程(例如在数据库中存储并由其执行的功能代码)产生。在数据加载期间,检索已生成语句并对进入数据执行。

[0036] 将进入数据加载到数据仓库的过程的性能例如执行时间和/或计算资源利用可以使用一个或更多最优化选项来改善。计算资源利用可以无限制包括处理器利用、存储器利用和/或网络利用。最优化选项可以包括例如将进入数据分区和分离处理每个分区,在将数据应用到目标表格之前将进入数据输入到可变表(volatile table),当进入数据不需要时从目标表格比较过滤历史,以及将数据时态地规格化的方法。

[0037] 在此描述的实施例涉及一般元数据驱动时态数据仓库加载设计,该设计包括产生数据加载代码的SQL代码生成器。当执行时,数据加载代码可以有效处理,并将任何量(初始加载、迁移、每日的、没小时的)与任何类型的源系统数据(拉动或推动,新或旧数据)加载到规格化时态数据仓库,基于在每个表格的主关键字中具有有效开始时间戳来识别净变化信息并将其排序到时态设计,并且仅使用set-SQL语句填充对应有效结束时间戳或等效时限,从而产生有效时限的等效。这样的过程有时共同称为变动数据捕捉(CDC)。

[0038] 已披露时态数据仓库加载设计通过关于一组进入数据自身并关于现有数据仓库分析该进入数据从而确定净变化数据来操作。适当有效时间排序(时态设计)然后分配并有效应用到新排序行,并且更新到仅使用ANSI SQL在目标数据仓库中定义时限的结束时间戳。该过程预生成SQL语句(例如插入和时态更新),并且当加载数据时完全在数据仓库数据库内检索并执行SQL。

[0039] 在此描述实施例的示范技术效果可以无限制包括(a)将进入数据集划分成包括第一分区和第二分区的多个分区,其中该多个分区的每个分区包括多个数据记录;(b)基于哈

希函数(hash function)和预定量的分区划分进入数据集;(c)将第一分区和第二分区循序或并行(例如同时)输入到预加载表;(d)将预加载表应用到时态数据仓库;(e)将分区输入到对应可变表(volatile table);(f)从可变表复制分区到预加载表;(g)识别第一分区中的数据记录,该数据记录包括等于先前输入数据记录的非关键字段的除时间戳之外的多个字段;(h)当输入第一分区到预加载表时排除已识别记录;(i)基于检测到在时态数据仓库中的有效数据记录不与在进入数据集中的数据记录关联,执行有效数据记录的隐删除(implicit delete);(j)确定与进入数据集中第一数据记录关联的最早源时间戳;(k)识别一主关键字组,该主关键字组表示与刚好在最早源时间戳之前的源时间戳关联的时态数据仓库中的数据记录,以及与在最早源时间戳之后的源时间戳关联的时态数据仓库中的一个或更多数据记录;以及(l)基于已识别组的主关键字输入第一和第二分区。

[0040] 实施例可以在下面关于特别应用来描述,例如存储关于材料清单(BOM)的信息和/或关于零件(例如机械设备零件)的信息的数据仓库。设想这样的实施例可应用于任何时态数据仓库。

[0041] 图1是包括服务器系统12和多个客户端子系统的示范系统10的简化框图,该多个客户端子系统也称为客户端系统14,连接到服务器系统12。如在下面更详细描述的计算机化建模和分组工具存储在服务器系统12中,并且可以由在客户端系统14(例如计算机)中的任何一个的请求者访问。如在图1中图解,客户端系统14是包括网络浏览器的计算机14,以使服务器系统12可使用互联网访问客户端系统14。客户端系统14通过包括网络例如局域网(LAN)或广域网(WAN)、拨号连接、电缆调制解调器和特殊高速ISDN线路的许多接口互连到互联网。客户端系统14可以是能够互连到互联网的任何装置,包括基于网络的电话、个人数字助理(PDA)或其他基于网络的可连接设备。数据库服务器16连接到含有关于各种事物的信息的数据库20,如在下面更详细描述。在一个实施例中,中心化数据库20存储在服务器系统12上,并且可以由在客户端系统14中的一个的潜在用户通过客户端系统14中的一个登录到服务器系统12上来访问。在可替换实施例中,数据库20远离服务器系统12存储并且可以是非中心化的。

[0042] 图2是系统22的示范实施例的扩展框图。系统22仅是合适计算环境的一个例子,并且不意图提出与本披露的使用或功能性的保护范围有关的任何限制。系统22不应解释为具有涉及在此图解部件中的任何一个或组合的任何依赖或需求。使用与在图1中使用相同的参考号在图2中识别等同于系统10的部件(在图1中示出)的在系统22中的部件。系统22包括服务器系统12和客户端系统1。服务器系统12进一步包括数据库服务器16、应用服务器24、网络服务器26、传真服务器28、目录服务器30和邮件服务器32。磁盘存储单元30(其包括数据库20)耦合到数据库服务器16和目录服务器30。服务器16、24、26、28、30和32在局域网(LAN)中耦合。另外,系统管理员的工作站38、用户工作站40和监控者的工作站42耦合到LAN36。可替换地,工作站38、40和42使用互联网链路耦合到LAN36,或通过内部网连接。在一些实施例中,数据服务器16耦合到其他装置例如目录服务器30不可访问的磁盘存储单元34。

[0043] 每个工作站38、40和42是具有网络浏览器的个人计算机。尽管在工作站执行的功能通常图解为在分别的工作站38、40和42执行,但这样的功能可以在耦合到LAN36的许多个人计算机中的一个执行。工作站38、40和42图解为仅与分离功能关联从而促进不同类型功

能的理解，该不同类型功能可以由具有到LAN36的访问的个人执行。

[0044] 服务器系统12经配置使用互联网提供商(ISP)互联网连接48通信耦合到包括雇员44的各种个人，并耦合到第三方例如客户/承包商46。在示范实施例中的通信图解为使用互联网执行，然而，任何其他广域网(WAN)型通信可以在其他实施例中利用，即系统和过程不限于使用互联网实践。另外并且除WAN50之外，局域网36可以代替WAN50使用。

[0045] 在示范实施例中，具有工作站54的任何授权个人可以访问系统22。客户端系统中的至少一个包括位于远程位置的管理员工作站56。工作站54和56是具有网络浏览器的个人计算机。同样，工作站54和56经配置与服务器系统12通信。此外，传真服务器28与包括使用电话链路的工作站56的远程定位客户端系统通信。传真服务器28经配置也与其他客户端系统和/或工作站38、40和42通信。

[0046] 利用图1和图2的系统，在不中断用户查询的情况下经规划小批量运行实现高效率和相对非侵入的近实时加载。过程基于标准ANSISQL，因此其可应用于任何数据库平台，对数据库管理系统(DBMS)能力起杠杆作用，特别对于大规模并行处理(MPP)架构提供超线性的可扩展性，并且不需要在外部服务器上的数据处理(例如，SQL可以从任何地方调用)。在一个实施例中，通过将主关键字定义和表名称用作参数，数据仓库加载在运行时间完全元数据驱动。另一优点是模式变动不需要变动数据捕捉系统的重编译或重启，并且操作元数据可以在任何时间变动(例如，显式或隐式删除(implicit delete)形式、分区的量和/或并行度的水平)。否则，任何接口类型可以适应于单个程序，并且在数据模型内的全部表格(有效时间包括在每个主关键字中)可以适应于单个程序。仅需要候选行作为输入(列+有效时间戳)，若有任何事物变动，不需要该变动的识别作为到变动数据捕捉系统的输入。对于快照接口，不需要删除的识别。在有效时间上的联结可以用在数据集和多个调用内并跨该数据集和多个调用的极短排序时间在主关键字内破坏。追溯和/或历史更新通过更新进入和现有数据的时间排序来执行。

[0047] 因为现有解决方案有时定制成接口类型并通常为每个表格中每列完全硬编码，所以上面提到的改善在现有解决方案上实现。另外，时态排序的现有途径是同时单行的，并且不是经set-SQL的全体(例如使用关系代数集运算符)。因此这些解决方案不与变动数据捕捉系统一样超线性缩放。例如，在此描述的实施例可以在小于处理100行所需要时间10倍的时间中处理1,000行。在示范实施例中，在处理期间没有数据从数据库移除，并且变动数据捕捉系统的调用形式可以是外部的(例如Perl)或内部数据库规程。

[0048] 所描述实施例促进减小并潜在消除与识别变动(例如插入、更新、删除、重编和/或历史更新)和将变动应用到时态数据仓库关联的开发成本，该时态数据仓库经开始-结束有效时间戳定义的时限保留历史。有效且非常可缩放的设计被描述，用set-SQL对DBMS引擎和架构起杠杆作用，不同于使用低效游标(同时一行)、外部数据加载服务器并生成关联网络流量的现有解决方案。当使用包括但不限于最终用户锁定法和经SQL修饰符使用时态历史的各种查询方法，经为效率最大化的非常迅速的set-SQL应用事务(在DBMS内最小化工作负载并最大化吞吐量的最终阶段和目标的相同结构)加载时，最低侵入设计允许连续查询。

[0049] 如在此进一步描述，实施例可以至少部分实施为SQL生成器的序列，该SQL生成器产生并存储SQL语句以便通过查询数据库编目(例如对于列名称和基础数据类型信息)和主关键字元数据表来加载数据。预生成SQL可以在运行时间对进入数据执行。下面描述的步骤

序列将候选行分析、准备并然后在单个有效事务中应用到目标数据库。这些步骤可以在带有访问的任何编程、描述或规程语言中实施，从而对数据库执行SQL发生器、取得导致的SQL语句，并且然后对数据库执行已取得状态。

[0050] 以下包括在此利用的某些术语和缩写词的定义。在线事务处理(OLTP)数据库是通常包括规格化数据库结构的基于事务的数据库。例如，与在所引用数据记录中的复制相反，在OLTP中的数据记录(例如在表格中的行)可以包括对另一数据记录(例如在另一表格中的行)的引用。进一步地，OLTP数据库可以强制引用完整性从而确保这样的引用有效(例如涉及现存数据记录和/或特别类型的数据记录)。

[0051] 主关键字(PK)是用于目标表(例如核心表、非核心表或派生层(derived layer))的如在数据建模工具中定义的完全主关键字。如在此使用，“非核心”表是在此表现为目标数据库层的规格化时态目标表。PK包括称为SOURCE_START_TS的源系统开始时间戳列(在数据库视图CDW_PK_COLS_V中可得)，其支持历史的保存。源时间戳表示在产生它的授权系统中行有效性时期的开始，并且在许多系统中可以称为产生或最后修改时间戳。在时态数据仓库中的有效时限可以表达为在此情况下包括SOURCE_START_TS并排除SOURCE_END_TS的表示时限的一对时间戳(例如，开始时间戳和结束时间戳)。

[0052] PK_latest是排除通常是在线事务处理系统业务关键字(在数据库视图CDW_PK_COLS_LATEST_V中可得)的SOURCE_START_TS的主关键字。

[0053] W_table是进入数据集变换的目标。在示范实施例中，W_table包括带有省略的都表示时态时期的2对标准开始-结束时间戳，但带有存在并称为SRC_START_TS的源系统时间戳的非核心表的复制。当选项ALL_VT(在下面更详细描述)设定成Y时，W_table的易失复制可以使用。

[0054] X_table是预加载表，加载到目标表的所有行的来源。X_table可以是目标表的复制，该复制带有存储分配动作(ETL指示符)的列和命名为src的代替来源的两个源时间戳的添加。

[0055] 目标是与带有唯一命名表格的单个数据库对应的一层计算机数据仓库，该表格表示数据仓库的范畴。非核心是目标的例子。其他潜在数据库表格层是核心(例如以第三范式或3NF完全整合)并被取得(例如预联合、聚合)。除以其他方式陈述之外，所有过程可以应用于这三层，其中所取得数据潜在源于非核心或核心表，但在调用该过程之前仍呈现为到W_table的输入。当选项ALL_VT(在下面更详细描述)设定成Y时，目标的易失复制可以使用。

[0056] ALL_VT选项表示系统是否应使用易失工作表。当ALL_VT停用(例如设定成N)时，系统使用在分段数据库中的两个已生成工作表(例如W_table和X_table)，该两个表格基于它们的目标副本。第三易失或非永久表格可以在这里描述的过程执行前用来加载W_table。除外部地用来变换数据集以便由CDC过程使用的任何其他表格之外，对于每个目标表，可以具有内建到分段区域数据库的这些表格的多达三个脚本生成的变量。这些的易失复制在ALL_VT启用(例如设定成Y)时产生。这三个表格是不可以有数据库建模工具直接建模的另外表格。相反，除了在运行时间构建的可变表之外，它们可以由脚本在构建时间构建并且包括在目标构建脚本中。

[0057] 在示范实施例中，脚本如在图1中示出产生并命名每个表格。

[0058] 表1

[0059]

CDC 共用表结构	表定义
W_table：用于除显式删除、目标右侧映射之外所有分段变形。	源自建模工具的目标非核心/已取得表定义——除放弃 SOURCE_END_TS、CDW_START_TS、CDW_END_TS，将 SOURCE_START_TS 重命名为 SRC_START_TS +做出选择
X_table：所有目标数据的直接来源并从 W_table 或潜在外部过程加载。CDC 的应用阶段仅从这里运行，添加或使用的 ETL 指示符代码是 I、O、U、D。CDC 代码将进入数据从 W_移动到 X_，除显式/级联删除之外（在 CDC 调用之前完成）。	源自建模工具的目标非核心/已取得表定义——除放弃 CDW_START_TS、CDW_END_TS，将 SOURCE_START_TS 重命名为 SRC_START_TS +做出选择，将 SOURCE_END_TS 重命名为 SRC_END_TS，将列 ETL_INDICATOR CHAR（1）添加到表格的末端
CDW_PK_COLS_LATEST_V（仅视图）	从建模工具主关键字加载，在 DATA_LAYER 上询问，通常仅使用目标，在 ETL 处理中取得
CDW_PK_COLS_V（在基表上的视图）	从建模工具主关键字加载，在 DATA_LAYER 上询问，通常仅使用目标，在 ETL 处理中取得

[0060] W_table是在调用CDC系统之前用于除显式删除之外所有分段变形的目标表。X_table是所有目标数据的直接来源，并从W_table或在显式或隐式删除的情况下经潜在外部过程加载。CDC的应用阶段在X_table中添加或使用ETL指示符代码，例如这里在别处定义的I、O、U、D。在将数据从W_table移动到X_table，并且在控制对目标数据库的变动的最终应用之前在X_table内进一步更新时，与CDC系统关联的代码初始化或设定。

[0061] 在示范实施例中，当ALL_VT启用时，脚本产生并访问如在表2中示出的表格，其中TNAME对应于实际目标表名称。

[0062] 表2

[0063]

表格	在步骤中的填充	在步骤（s）中的使用	内容
TNAME_VT	105	106-111	限于历史的 X 和 1 个先前行中 PK 的目标表的复制
TNAME_TVT	100	102-105（如果 ALL_VT = Y）	目标表的 UT（分区选择）
TNAME_WVT	101	103-104（如果 ALL_VT = Y）	W 表的 UT（分区选择）
TNAME_XVT	101	102，104-112（如果 ALL_VT = Y）	X 表的 UT（分区选择），在步骤 112 中 ins sel 到 X 表
TNAME_KVT	103	104（如果 NORMALIZE_LATEST = Y）	PK 的 X 表的 UT（分区选择），从而在步骤 104 中排除比较(无新信息）

[0064] 提取、变形和加载(ETL)操作可以涉及使用在表3中示出的ETL指示符。

[0065] 表3

[0066]

ETL 指示符	目标动作	新目标行结束 TS
I	插入新行	空（开放有效时间）
U	插入新行，将目标最后行结束时间戳（如果没有已终止）更新到在 x_table 中 PK 内的最早 U 行开始 ts。	1. 如果在 x_table 中 PK 内最后行则为空 或 2. 新 x_table 行的开始（即使为删除） （闭合有效时间）
O	插入是失序更新的新行，其中它不是在 PK_Latest 内的最后开始 ts，或其是最后行但其开始 ts 比在 PK 内的最后终止早。在任一情况下该行在 X_table 中或在加载到目标之后得到结束 ts（预终止）。	1. 将在 X_table 中的结束 ts 设定成在目标中 PK 内下个行的开始 ts 或 2.在加载到目标之后将结束 es 设定成最后失效期 （闭合有效时间）
D	用未终止结束时间戳或先前 X_table（如果刚好在前面）更新最后当前目标表格行，仅从 X_table 行的开始 ts 设定结束 ts。行不直接加载	空，除非在带有 X_table 中的较新行的批处理内更新 （闭合有效时间）

[0067] 如在上面示出，提取、变换和加载(ETL)指示符包括I、U、O和D，并且每个与一个或更多目标非核心表动作关联，例如加载新目标行或结束在现有行上的时间戳。对于ETL指示符I，非核心动作是新行的插入，并且新目标行结束时间戳是NULL(最后行，无限的周期有效性)直到被取代或逻辑删除。对于ETL指示符U，非核心动作是将新行，对非核心最后行结束时间戳的更新(如果没有已终止)插入到在X_table中主关键字(PK)内的最早U行开始时间戳。任何结束时间戳来自其他X_table记录。如果最后行在X_table中PK内或是下个X_table行的开始，则指示符U的新目标行结束时间戳是NULL。除时限空隙经逻辑删除来显式设定或以其他方式预先指定之外，行的结束时间戳或结束时期由在主关键字内随后行的开始时间戳暗示。因此新行的默认结束时间戳或有效期是“直到被取代”。

[0068] 对于ETL指示符O，非核心动作是新行的插入，该新行是失序更新，其中它不是在PK_Latest内的最后开始ts，或其是最后行但其开始ts比在PK内的最后终止早。在任一情况下该行在X_table中或在加载到目标之后与结束时间戳关联(即预终止)。对于指示符D(逻辑删除)，非核心动作是用未终止结束时间戳或先前X_table(如果刚好在前面)更新最后当前目标表格行，从X_table行的开始时间戳设定结束时间戳。行不直接加载。指示符D的新目标行结束时间戳初始为NULL，并且可以后来基于在X_table中的较新行更新。

[0069] 在示范实施例中，ETL CDC处理和代码生成器依靠预填充到分段表的主关键字元数据。产生两个视图从而提供包括SOURCE_START_TS的完整数据仓库主关键字，或通常是OLTP业务关键字的排除SOURCE_START_TS的该关键字的最后视图。另外，代码生成器依靠提供描述数据库结构(例如数据库、表格和/或列)的信息的标准数据库目录视图。可以仅实施为视图的第一视图可以命名为CDW_PK_COLS_LATEST_V。可以是在基础表上的视图的第二视图可以命名为CDW_PK_COLS_V。第一视图和第二视图都在建模工具中从主关键字加载，在DATA_LAYER上查询，并且在ETL处理中通常使用非核心的已取得层。

[0070] 变动数据捕捉过程基于已定义标准化方案过程操作从而经早先提到的2个视图构建工作表即W_table和X_table的一般形式，以及主关键字元数据的可用性。

[0071] 关于变动数据捕捉过程的功能概要，通过将源数据变换并从分段表加载到W_table(在分段数据库中)，每非核心的源-系统特定变换过程，已取得的与任何其他的关联数据加载在调用变动数据捕捉之前执行，除了显式删除消息之外。加载W_table的过程通常对于每个表格独立，但不可以基于为数据库定义的具体变换过程完全独立。

[0072] 在一个实施例中，W_table和X_table在为源系统运行的每个CDC开始之前为空。除显式删除之外，变动数据捕捉从W_table经X_table加载数据到计算机数据仓库(CDW)数据层(例如非核心)。在示范实施例中，该过程跨目标表并行化到可能范围并且没有相互依赖性。

[0073] 在示范实施例中，CDC系统在相对段的时间量(通常几秒或更少)中使用set-SQL在单数据库将阶段负载应用到每个目标表。这些事务跨表格跨目标表并行化到可能范围并且没有相互依赖性。在此描述的变动数据捕捉系统和方法涉及基于可以对时态准则起杠杆作用的合适查询访问方法，允许不干扰报告的CDW灵活迅速加载的小批量设计。没有数据库实用程序用来加载目标表。对于给定源系统或一组相关表，整个批量运行可以在将新批量运行初始化之前完成。即，数据加载的CDC相关部分在没有关于目标表的并行化或重叠的情况下执行。

[0074] 除由逻辑删除产生的缝隙之外，CDC系统仅更新现有目标行的两个标准结束时间戳(来源或有效的和事务的或ETL)，其中源结束时间戳(source_end_ts)通常仅设定成随后行的源系统指定删除时间戳(为删除)或源开始时间戳。这有效地更新有效时限，如果时期类型代替一对时间戳可用，则可以使用该时期类型不失一般性实施该有效时限。所有新数据(例如任何新关键字或非关键字属性值)导致新目标行。除了指定已取得表格在一些情况下可以整个刷新之外，没有其他CDW列可以在目标表中由CDC过程更新。

[0075] CDC系统确保从数据模型直接加载的每计算机数据仓库元数据的主关键字的唯一性。没有有效完整性约束经采用或需要从而实施。因此，被动检查脚本可以作为进一步确认来运行。CDC系统也确保时间戳范围有效。例如，系统可以检验结束源或有效时间戳大于或

等于开始时间戳，和/或在PK内，源或开始时间戳等于除删除之外的先前行的源结束时间戳，和/或源结束时间戳对于除逻辑删除之外的主关键字内最后行为空。相似功能性可以在代替一堆时间戳的时限数据类型上想象操作。

[0076] CDC系统填充表示两个时态时限的全部四个标准化时间戳，其中源开始时间戳为总是从源数据行填充的唯一时间戳(在W_table和X_table中命名为SRC_START_TS)。源结束时间戳属性也经由带有当前行的唯一内容或删除时间(如果已知，否则为当前时间)的主关键字内下个行的开始时间戳从该源获得，但当行表示最后信息时可以为空。

[0077] 两个CDW时间戳反映实际加载时间(事务时间)但可以为给定表的给定小批量运行标准化。例如，CDW时间戳可以在加载之前立即获得，并且设定为固定值从而允许容易识别在给定表的给定小批量中加载的所有行。CDC系统也在每个表格加载之后(在预先CDC中的W_table，如果ALL_VT停用则作为步骤104的部分的X_table，或在步骤112的最后重复的结束)收集或刷新统计量。变动数据捕捉系统也可以调用被动主关键字唯一性，并且如果不分离调用则调用每功能需求的外来关键字完整性检查。

[0078] 在CDC系统中，暗示的父子(parent-to-child)删除可以实施为占位符(place-holder)从而示出有待适应的过程流程。复杂分段变换与混合模型发布(例如用于一个表格的推送和快照)需要的的变化没有处理。如提到，任何显式和任何复杂隐式删除可以在CDC开始之前由源系统特定变换代码加载到X_table。CDC系统允许已删除记录恢复或“再生”，即使在相同批量中。这样的状况可以在注意到来源的开始时间戳大于或等于仅表示删除的先前记录的源结束时间戳时检测到。

[0079] 在示范实施例中，非核心新行计数等于非核心旧行计数+I+O+U计数。该系统可以将更新(O和U可以分离跟踪)计数，并且对其中结束时间戳不为空的X_table中O和U的计数确认这些计数。

[0080] 由代码生成器产生的预生成询问可以包括涉及数据层的状况，从而在不同层(例如非核心的、核心的或已获得的)中防止潜在重复表名称。在示范实施例中，不是指定CDW和SOURCE时间戳的时间戳列包括时区和六位精度。随着对代码生成器的适当变动，这些需求中的一个或六个可以省略。

[0081] 用于将进入数据加载到数据仓库的示范方法在下面关于特别处理步骤或操作来描述。以每元数据设定的在分区加载步骤(在图4中示出)内多个重复中并行运行多个分区的选项，分离模块基于可应用的预先需要来提供，其中每目标W_table或X_table一次调用。例如，在进入数据之间具有相互依赖性的情况下，在此描述的步骤100到112可以需要涉及源系统的所有W_table在处理的初始化之前完全加载。然而，CDC过程自身不引入这样的相互依赖性。

[0082] 在示范实施例中，每个步骤作为分离SQL语句执行，使得能够避免数据库性能恶化。进一步地，步骤110到112可以不包括在单个数据库事务中。相反，例如每个步骤或步骤的多个分组中的每个可以在分离事务中执行。在一些实施例中，整个加载过程在任何数据库错误的情况下放弃，但可以在信息消息和/或告警消息的情况下继续。应用步骤201-207可以作为多重单个请求的单个数据库事务运行，其中显式回退在任何错误上提出。所有CDC过程可以在新的小批量运行为给定源系统初始化之前完成。

[0083] 用于分段和目标数据库的数据库名称可以为每个CDW环境适当参数化。表名称基

于目标表名称参数化，并且分配到正确数据库名称(W_、X_和_X表格在分段数据库中，并且目标表可以在非核心的、核心的或已取得的数据库中)。注意一些例子表格不关于数据库受限。例如，W_和X_可以在STAGE中，而目标通常是NONCORE。

[0084] 适当的锁定修饰符由CDC代码生成器添加从而在目标表或共享永久段上将锁定内容最小化。在示范实施例中，在将应用阶段封装期间ETL不控制到目标数据的访问。共同数据仓库架构包括与外出数据库视图的“LOCK FOR ACCESS”等效的“乱读(dirty read)”。在事务内的上面描述的应用步骤的顺序经设定将该问题最小化。在可替换实施例中，定义如需要则等待事务完成的另外SQL视图。

[0085] 在示范实施例中，CDC过程控制在目标表水平，其中同步点控制在源系统水平，对应于根据ETL代码记述并结构化的工作。用于源系统中所有表格的应用步骤(每表格一个数据库事务)可以并行化到可能范围从而提供新数据的相容视图，并且将在查询的引用完整性问题最小化。应注意由于源开始时间戳是主关键字的部分并且在一些实施例中在父子之间变化，因此真实引用完整性不可以用常规约束强制。如可用，则时限数据类型的使用可以允许时态PK和外来关键字(FK)约束受强制。

[0086] 参考图3，其是将示范数据变动捕捉过程与可以希望的支持预先需要外部加载过程一起图解的流程图70，不同的候选行在步骤72中由外部过程(例如除CDC过程之外的过程)插入到W_table。在示范实施例中，使用表格-指定变换将所有限定源系统表格行(例如消息或快照)写入到W_table。可以使用INSERT SELECT设定逻辑。在步骤72的一个替换中，候选行插入到W_table并用一个“批量”运行的CDC代码的开始点或基线填充W_table。通过在SELECT语句中使用DISTINCT消除完整复制行。在步骤72中的插入之后，W_table含有如果历史由来源保留并提供则可以包括历史的已变换进入数据集的完全不同的快照。对于基于消息的接口，该步骤可以不包括删除消息。

[0087] 除在下面进一步描述的步骤101之外或与步骤101组合，在步骤74中显式删除从分段插入到X_table。在这样的情况下，分段/变换SQL代码将PK属性和“D”(删除)ETL指示符加载到X_table。在一个替换中，X_table在步骤74中处插入显式删除之前清除(例如清空)。在另一替换中，步骤74包括为不可以包括任何显式删除的快照型接口省略的显式删除的插入，除了当LOAD_TYPE=B时之外，在此情况下可以允许两者的组合。当隐式删除代替显式删除使用时，步骤74可以省略。

[0088] 在步骤76中，CDC过程等待76与源表格关联的加载工作完成。在任选步骤78中，当加载工作完成时，级联删除执行。对于一些表格(例如从属子表格(dependent children))，删除行写入到X_table。用于源系统的变换SQL过程直接加载PK、父src_start_ts、“D”的ETL指示符，以及终止行的源开始时间戳。在一个示范实施例中，步骤78的级联删除是以级联到子表格并且不显式提供的父删除的源变换为基础的CDC过程的任选预先需要。对于不是最后的目标行，这些删除可以在下面更详细描述的应用步骤206中执行。

[0089] 在任选步骤80中，执行父-子隐式删除。在一个替换中，响应于更新，为一个或更多表格(例如从属子表格)将删除行写入到X_table。用于源系统的变换SQL过程加载PK、父src_start_ts、“D”的ETL指示符，以及不与存储在src_end_ts中的该子表格一起发送的先前父行的源开始时间戳。在另一实施例中，该设计基于源系统接口(例如固有父TS)变化。在步骤80的一个替换中，父-子隐式删除是以暗示所有先前子行的删除的从属子行更新为基

础的CDC过程的任选预先需要。在一个可替换实施例中，对于不是最后的目标行，这些删除根据在下面披露的应用步骤206执行。在一些实例中，步骤72到82在CDC过程之前并且不容纳在这里描述的过程内。

[0090] 在步骤82中，CDC过程等待与源表格关联的工作完成。当这些工作完成时，在步骤84中产生数据库会话。步骤84可以包括在针对数据库会话的过程和/或线程中产生数据库会话，以使在一个会话中执行的操作不影响在另一会话中执行的操作。

[0091] 在步骤86中，已产生会话的量与SESSION_PARALLELISM比较。如果会话的量小于SESSION_PARALLELISM，则步骤84再次执行从而产生另一数据库会话。否则在步骤88中，CDC过程等待所有数据库会话完成处理。

[0092] 对于由步骤84产生的每个数据库会话，CDC过程在步骤90中确定是否任何分区可用于加载。如是，则在步骤92中使用由84产生的可用数据库会话执行分区加载，如在下面参考图4描述。在一个实施例中，步骤92提供分区加载从而执行将进入数据的一个分区输入到预加载表例如X_table。

[0093] 在一些实施例中，进入数据被划分成多个分区(例如，通过将NUM_PARTITIONS设定成大于1的值)，并且步骤92可以为每个分区执行。例如，进入数据可以关于PK不同地划分，并且基本均匀地使用元数据(例如，以分区大小的1%、5%或10%变化)。在一个实例中，分区的量可以由用户提供的参数NUM_PARTITIONS定义。将NUM_PARTITIONS设定成1的值可以通过导致进入数据集作为单个分区处理来有效停用进入数据集的分区。

[0094] 当NUM_PARTITIONS>1时，多个分区可以基于表示个别分区的执行并发性的希望程度的另一用户定义参数SESSION_PARALLELISM并行加载。可替换地，分区可以循序或串行加载。例如，设定SESSION_PARALLELISM等于1可以导致分区的循序处理。

[0095] 在步骤90，当CDC过程确定没有分区可用来在数据库会话中输入时，会话完成处理，并且执行在步骤88继续，其中CDC过程等待所有数据库会话完成。

[0096] 在步骤94中，在所有分区加载完成之后，已加载数据中的全部在一个数据库会话中应用，如关于图6描述。在一个图解例子中，NUM_PARTITIONS设定成5，并且SESSION_PARALLELISM设定成3。步骤84执行3次从而产生3个数据库会话。第一会话执行步骤92从而执行第一分区的加载，第二会话执行步骤92从而执行第二分区的加载，并且第三会话执行步骤92从而执行第三分区的加载。在该例子中假设分区大小基本相似，则第一数据库会话完成步骤92(关于第一分区)并执行步骤90，确定更多分区(即第四和第五分区)可用于加载。第一数据库会话执行步骤92从而执行第四分区的加载。第二数据库会话完成步骤92(关于第二分区)并执行步骤90，确定分区(即第五分区)可用于加载。第二数据库会话执行步骤92从而执行第五分区的加载。第三数据库会话完成步骤92(关于第三分区)，执行步骤90，确定没有分区可用于加载，并且前进到步骤88从而等待所有会话完成。相似地，第一和第二数据库会话都完成步骤92(分别关于第四和第五分区)，执行步骤90，确定没有分区可用于加载，并且前进到步骤88从而等待所有会话完成。随着所有数据库会话完成，CDC过程前进到步骤94从而在一个数据库会话中应用已加载分区。

[0097] 图4是图解示范分区加载过程的流程图98。在示范实施例中，由流程图98图解的过程为分区NUM_PARTITIONS中的每个循序和/或并行执行。在下面描述的步骤可以取决于一个或更多元数据参数，如由表4示出。

[0098] 表4

[0099]

步骤	LOAD_TYPE	ALL_VT	NORMALIZE_LATEST	NUM_PARTITIONS	TVT_FILTER_HISTORY
100	S|B	Y	n/a	N 的 1-N	Y|N
101	S|B	Y	n/a	N 的 1-N	n/a
102	S|B	Y|N	n/a	n/a	n/a
103	S|B	Y|N	Y	n/a	n/a
104	n/a	Y|N	Y|N	n/a	n/a
105	n/a	Y|N	n/a	n/a	n/a
106	n/a	Y|N	n/a	n/a	n/a
107	n/a	Y|N	n/a	n/a	n/a
108	n/a	Y|N	n/a	n/a	n/a
109	n/a	Y|N	n/a	n/a	n/a
110	n/a	Y|N	n/a	n/a	n/a
111	n/a	Y|N	n/a	n/a	n/a
112	S|B	Y	Y|N	n/a	n/a

[0100] 在示范实施例中，在表4中的字符值（例如“Y”或“S”）表示步骤仅在元数据参数等于字符值时执行。管状符号（“|”）表示在字符值之间的析取（“或”）关系，在此情况下步骤在元数据等于列出字符值中的任何时执行。进一步地，“n/a”表示元数据参数不可应用于过程步骤。在这样的情况下，该步骤可以无关于元数据参数的值执行。

[0101] 在步骤100中，当ALL_VT=Y并且NUM_PARTITIONS>1时，目标分区的加载为快照负载（例如LOAD_TYPE=S或B）执行。总之，步骤100需要大型表格，其中产生将表格逻辑分区到分离可变表格的成本比添加该步骤的成本更多地减小处理器使用和/占用的执行时间。在一个实施例中，步骤100可以在与步骤101相同的状况下调用。

[0102] 在步骤100和101中，使用HASHBUCKET(HASHROW())函数产生逻辑分区，该函数在示范实施例中基于排除源开始时间戳的主关键字列产生在一和一百万之间的整数。这提供相对均匀的分区（例如相似大小的分区）并且是将数据行确定性分配到不同分区的低成本（例如在计算资源方面）方法。MOD（模）函数对元数据参数NUM_PARTITIONS使用，其中作为余数的1到N是CURRENT_PARTITION的元数据值。代码生成器在SQL中为步骤100和101例示这些值，并且基于在CDW_CDC_PARM中的表参数将它们适当存储在CDW_CDC_SQL表中以便检索。

[0103] 步骤100

[0104] 在示范实施例中，步骤100使用通常值为N(否)的称为TVT_FILTER_HISTORY的历史过滤参数。在CD基于在W_table中的进入数据运行期间，当TVT_FILTER_HISTORY等于Y时，CDC系统精简在目标表中不需要的较早历史行。用于每个W_table PK的最早时间戳经查询并比较从而构建在已分区可变表上充当过滤器的一组目标表主关键字。每应用分区过滤器的W_table中主关键字使用较早源时间戳的WITH子句构建已取得表。这然后用来产生来自排除所有较早历史的目标表的需要的一组不同主关键字。

[0105] 在一个实施例中，集合包括比最早W_table行新的行，在最早W_table行之前的行，以及在没有已经包括的情况下的最后行，从而在步骤102中确保合适暗示的删除处理。在一个替换中，分区表达式在每个情况下应用。另外，因为进一步查询状况添加从而经分离查询状况从目标表提供匹配任何显式删除行主关键字的行，所以LOAD_TYPE=B可以影响该选项的操作。

[0106] 在另一示范实施例中，TVT_FILTER_HISTORY启用，导致后面步骤例如步骤104更低的计算资源利用。继续该实施例，TVT_FILTER_HISTORY可以在为相对巨大(例如含有数百万行)并具有相对大百分比的历史行(例如大于表内容的75%)的表格减小资源利用中是有效的。

[0107] 启用ALL_VT的一个优点是导致PK_LATEST用作主索引(PI)，在大规模并行处理(MPP)数据库系统上经哈希算法在一个或更多指定列上的实施分配数据的方法。有利地，该选项提供比在其中PI具有远少于PK的列的表格中小的失真的PK_Latest。另外，当ALL_VT启用时，系统从带有访问锁定修饰符的基础表读取数据从而避免在基础表视图中过滤的任何风险。改善的效率可以通过在从列出显式列名称的基础表读取的一个步骤中产生可变表(VT)来实现。这导致列保留“非空”属性，不同于从访问视图产生。在一个替换中，当NUM_PARTITIONS=1时，在该步骤中绕过哈希分区从而节省计算资源。例如，NUM_PARTITIONS可以为失真表设定成1，否则该失真表足够大到由与在相对小的分区中处理数据关联的计算资源的减少来补偿与分区关联的计算成本。

[0108] 步骤101

[0109] 在示范实施例中，当ALL_VT=Y并且NUM_PARTITIONS>1时，步骤101为快照负载执行(例如LOAD_TYPE=S或B)。即，该步骤可以为大型表格需要，对于大型表格，在分离可变表中产生表格的不同逻辑分区的成本比添加该步骤的成本更多地减小处理器使用和/占用的执行时间。在一个替换中，步骤101在与步骤100相同的状况下调用，并且完成为W表、X表和目标表构建可变表的过程，从而允许如希望则逻辑分区分离处理并在并行会话中处理。

[0110] 使用HASHBUCKET(HASHROW())函数产生逻辑分区，该函数在示范实施例中基于排除源开始时间戳的主关键字列产生在一和一百万之间的整数。这提供相对均匀的分区(例如相似大小的分区)并且是将数据行确定性分配到不同分区的低成本(例如在计算资源方面)方法。MOD(模)函数对元数据参数NUM_PARTITIONS使用，其中作为余数的1到N是CURRENT_PARTITION的元数据值。代码生成器在SQL中为该步骤和步骤100例示这些值，并且基于在CDW_CDC_PARM中的表参数将它们适当存储在CDW_CDC_SQL表中以便检索。

[0111] 产生X_table的空易失复制。对于LOAD_TYPE=B，第三SQL语句可以执行以从匹配当前哈希分区的X_table插入显式删除行(例如ETL_INDICATOR=“D”)。除了在该情况下之外，没有其他行从X_table读取，否则该情况对于LOAD_TYPE=“S”在CDC开始完全为空，或对于

LOAD_TYPE="B"仅用显式删除来填充。

[0112] 步骤102

[0113] 在示范实施例中，步骤102在快照负载上执行(例如LOAD_TYPE=S或B)，并且在加载类型S和B之间在步骤102中可以没有代码不同。即，用于显式删除的X_table行的构建可以在元数据的完整快照可用时调用，并且为其存在不取决于父表的表格调用。这些后面的情况是父子暗示删除。在一些实施例中，没有对快照接口的替换，例如使用修改时间戳仅加载已变动行是实际的时候，使用步骤102。

[0114] 在一个实施例中，步骤102包括隐式删除步骤。通过检测在非核心(结束时间戳为空)中是有效行并且不再在进入快照中的最后主关键字(PK_latest)来确定删除；因此推断自从最后数据馈送以来其已在源系统中删除。在一个实施例中，当前数据库时间戳插入到SRC_START_TS以便由应用步骤(例如应用步骤202)使用。例如，单个应用步骤可以使用当前数据库时间戳执行隐式和显式删除。该时间戳在目标表中变为结束时间戳。由于没有源自源系统的触发或删除时间，因此当前时间戳在源系统中用作推断删除时间。

[0115] 步骤103

[0116] 在示范实施例中，当NORMALIZE_LATEST=Y时，步骤103为快照负载执行(例如LOAD_TYPE=S或B)，其中在加载类型S和B之间没有代码不同。该步骤可以在例如表格具有充分量的无内容新行时调用。特别地，该步骤仅消除带有比最后有效行更新的SOURCE_START_TS的每主关键字的下个连续行，并且全部其他非关键字属性相同。例如，步骤103可以是有效的，其中大百分比的W_table行表示带有较新时间戳并且没有新属性的当前目标表行。

[0117] 步骤103的添加可以导致比仅使用步骤104低的成本的过程，其识别变动并将未变动较新行的主关键字加载到名为table_KVT的可变表，进而仅在步骤104中使用以将行排除于更复杂的完全排序和比较之外。由于步骤104不需要完全比较，因此该途径的计算资源节省可以是充分的。

[0118] 步骤104

[0119] 在示范实施例中，步骤104以源自W_table的候选行加载到X_table，该候选行在除源时间戳的最后3个数字之外的至少一个属性上与目标表行(当ALL_VT=N时)或在VT中存储的已过滤目标行(当ALL_VT=Y时)不同。这样的行初始编码为ETL标识符"I"，并且如需要则SOURCE START TS以一个微秒唯一排序。该过程允许非关键字属性变动更新到目标表(其中对应的1毫秒添加到源开始TS)，例如在错误中先前删除的最后记录的重激活。这用有效时间解析唯一性违反从而确保仅加载不同的进入数据集行。

[0120] 例如，若工作故障导致需要重运行更新W_table中的列而不是源开始时间戳的工作，则变动数据捕捉系统检测新的非关键字属性并以带有唯一的排序源开始时间戳将新行插入到非核心。在任何情况下，考虑进入的和现有的数据，用于给定主关键字的任何新源开始时间戳也导致提供非关键字属性的新行如有的话则不同于该主关键字的刚好之前的行。

[0121] 在一些实施例中，步骤104的时间戳重排序部分省略(例如，如果源系统保证唯一业务关键字不包括源开始时间戳)。最小时间增量例如一微秒添加到随后行的源开始时间戳，该随后行具有无进一步排序完成的相等主关键字，依靠在PK内的定序函数，例如row_number()函数的等效。

[0122] 时间戳重排序用来初始保证唯一主关键字(带有时间戳)因此确保一对一行分配

的更新过程。由于没有不同的非关键字属性，因此排序的行中的一些可以随后删除（见于步骤107）。由于保留最早的该行，这将引入新排序的可能性最小化（例如，最早行没有添加到它的时间）。当X_table不是可变表（例如ALL_VT停用）时，在步骤104中在X_table上收集或刷新统计量促进实现最优加载性能。

[0123] 在示范实施例中，步骤104的操作基于最优化选项变化。例如，当ALL_VT启用时，步骤104可以接收可变表格而不是常规的或永久的表格并对其操作。进一步地，当NORMALIZE_LATEST=Y时，另外的子查询在SQL语句的末尾添加从而排除源自在步骤103中填充的_KVT可变表的行，如在上面描述。这避免在该步骤中检测到导电一组大得多的较新但未变动行的高成本的自连结。NORMALIZE_LATEST可以连同ALL_VT一起启用。

[0124] 步骤105

[0125] 在示范实施例中，步骤105在先前步骤中以候选插入行加载到X_table之后执行。因此选择查询不需要将在W_table中的进入行和在X_table中的删除行联合从而确定在当前运行中涉及的该主关键字组。当进入W_table行在加载之前消除时，特别是在快照加载的情况下，该途径可以导致相对小的可变表。

[0126] 相似于步骤104，当ALL_VT启用时，步骤105可以接受可变表名称并对其操作。进一步地，ALL_VT选项可以影响导致的可变表的主索引。当ALL_VT=N时，可以使用与W_table和X_table主索引匹配的主索引。相反，当ALL_VT=Y时，使用的主索引可以是排除SOURCE_START_TS的主关键字从而匹配其他三个可变表。

[0127] 步骤105可以促进充分性能最优化，从而通过使用存储在临时表中的受限子组的目标表行减小在步骤106-110中执行的分析，特别是步骤107的时态规格化的成本。改善可以对于仅将新候选行发送到W_table的推送接口和/或当广阔历史在目标表中存在时显著。性能改善的两个方面是可能的。第一，CDC系统可以仅为在W_table和X_table中含有的主关键字（排除源开始时间戳）考虑目标表行。对于净变化或推送接口，加载越频繁，该步骤可以在减小分析步骤的成本中越有效。联结的数据量的数量可以至少减小到百分之一（例如，每存在的加载主关键字1%）。

[0128] 第二，CDC系统可以将源自目标表的该PK行的时限限于如有的话在最早进入W_table和X_table源开始时间戳之前的行和所有随后行。即，CDC系统可以忽略分析步骤中不需要的历史行，为每个PK排除比到最早W_table或X_table的先前行早的行。由于一些中间较新目标行也可以是不需要的，因此该最优化不可以确定行的最小数目。时态排序仅需要刚好在进入行之前或之后的行。时态过滤的该途径预期在相对低的计算成本提供充分益处。历史的重编是一般少见的。因此新的进入数据通常比所有先前已存储数据更新。因此该步骤通常仅读取在上面描述的第一性能改善方面中选择的每PK的最后当前行。

[0129] 步骤105可以填充在每次运行清除的永久表或可应用于涉及的DBMS的任何形式的临时表。不失一般性，以从用户账户卷（已经巨大从而支持涉及的连结）分派的空间，选择易失临时表。在没有数据库目录影响或产生表许可的需要的情况下，该表以在目标表上选择语句的输出被定义并自动具体化。

[0130] 在一些实施例中，CDC系统假设当NUM_PARTITIONS=1（没有分区完成）时，步骤100任何随后执行（例如下次CDC运行）使用分离数据库会话，该会话确保可变表及其内容毁坏并因此无错误允许产生表命令。当使用分区（例如NUM_PARTITIONS>1）时，步骤111放弃该表

格从而在单个会话中允许重复迭代，如在下面描述。

[0131] 步骤106

[0132] 在示范实施例中，步骤106为插入候选(例如排除删除)将在X_table和非核心之间的副本主关键字排序。在一些实施例中，在X_table内排序可以由步骤104执行。当ALL_VT启用时，步骤106可以接受可变表名称并对其操作。

[0133] 通过添加比在否则未使用的六个子第二时间戳数字中最后三个数字中含有的最大序列大的值开始，CDC系统确保主关键字唯一并且跨现有和预期数据行排序。较新的小批量加载每次接受新时间戳，并且如果时间戳的显著部分是唯一的则潜在表示最后记录。

[0134] 步骤106可以是步骤107和之后的预先需要，并且由于如果带有副本主关键字的数据记录具有新内容则排序到唯一时间戳，因此消除主关键字相等情况。删除记录可以排除。另外，时间戳的“主干(stem)”(例如几乎最后三个排序数字)可以用于随后的“分组依据(group by)”操作，从而允许仅以时间戳区分的多个主关键字值排序。

[0135] 步骤107

[0136] 在示范实施例中，当ALL_VT启用时，步骤107可以接受可变表名称并对其操作。当与由在主关键字内的源开始时间戳分类的刚好先前的行比较时，步骤107删除不含除源开始时间戳之外的新关键字或非关键字属性的候选W_table行。该步骤表示普遍称为时态规格化的过程的压缩单元。

[0137] 计算资源可以在包括相同数据的行加载多于一次时浪费。因此，步骤104实施时态时限压缩单元。然而，仍可以希望记录数据从“A”到“B”然后回到“A”中的任何变动。因此，排除开始时间戳的每个不同行的第一实例维持在X_table中。更具体地，在X_table和非核心表的联合内，如果PK_latest相同，并且如果当由在PK_latest内的源开始时间戳分类时除时间戳之外的所有列与先前的行相同，则检测在X_table内的数据。

[0138] 在一些实施例中，启用ALL_VT可以充分减小步骤107的计算资源利用，特别是在有限数目的输入行(例如带有频繁加载的推送接口)和带有广阔历史的目标表(例如包括部分修订)的情况下。例在该审慎乘积连结中的改善可以达到若干数量级。例如，处理器利用和/或存储器利用可以减小。进一步地，由于计算资源利用环境，因此步骤107的占用时间也可以减小。

[0139] 在两个或更多相同连续行(例如在PK和属性上相同)的情况下，CDC系统可以确保在当前最后终止目标行的结束之后最新行开始并且较早候选行在最后目标行的时限内开始时，不是全部都作为冗余删除。该状况可以称为“重激活”情况，其可以例如在源系统暂时逻辑删除数据并然后在较新时间戳的情况下恢复数据时发生。向终止行的结束时间戳添加一毫秒允许即使在所有其他属性匹配先前行，提供最小时态粒度的情况下，CDC系统仍开始新行。具体地，另外连结(代号为表C)可以包括在步骤107中，使用在线分析处理(OLAP)查询从而即使逻辑删除仍寻找最新源开始时间戳，并将开始日期(或年份2500，如果没有这样的行存在)和结束日期返回从而与最后X_table行比较。如果下个最后行(B.)在X表中但容纳在最后C表目标行的时限中并因此B行在该步骤中删除，则添加到SQL语句的逻辑可以防止放弃最后X_table行(A.)。在示范实施例中，C.连结的额外计算成本最小。

[0140] 由于计算数据仓库将时态有效性存储为开始-结束时间戳范围或时期，因此知道假如相同进入行是在非核心中的最后行，则该相同进入行实际上仍然不是新信息。相似地，

知道一旦结束时间戳在进入数据的情况下分配，则在W_table中的两个相同行具有连续但不同的开始时间也不是新信息，最早行的开始时间已经在开始和结束时间戳之间的时期中捕捉该内容。在示范实施例中，步骤107在X_table里面容纳历史更新并移除邻接副本。如果表格具有许多列，特别是在需要在比较的任一侧上检查空值的情况下，则与步骤107关联的SQL语句可以相对大。尽管一个普遍利用的系统包括每语句一兆字节(1MB)的限制，但其他工具可以施加较小的大小限制，这可以需要多个步骤将非关键字属性比较分解到已减小复杂度或大小的执行单元。当比较任选列中的全部(一般全部非PK的列)时，空保护经Coalesce函数提供。row_number函数的使用依靠由步骤106确保的在X_table和非核心之间的不同源开始TS。

[0141] 步骤108

[0142] 在示范实施例中，当ALL_VT启用时，步骤108可以接受可变表名称并对其操作。步骤108是为候选插入行(“I”)更新提取、变换和加载(ETL)指示符的两个步骤中的第一个，在此情况下对于较新更新从“I”到“U”。如在此使用，术语“较新的”指代具有主关键字，包括源开始时间戳的数据记录，该数据记录晚于在非核心中相同PK_latest内最后行的主关键字，即使标记为删除。若每个ETL标识符表示新内容并且不在步骤107中移除，则步骤108可以更新在主关键字内多于一个X_table行的ETL指示符。在示范实施例中，仅最后有效非核心行的结束时间戳在应用阶段(例如在下面的应用步骤202)中更新，该应用阶段仅找到每PK的最早“U”行从而将其开始时间戳作为最后非核心行的结束时间戳来应用。当具有X_table中每PK多于一个行设定成“U”从而反映几乎最后行插入到预终止的目标时，在下面描述的步骤110可以提供结束时间戳。

[0143] 步骤109

[0144] 在示范实施例中，当ALL_VT启用时，步骤109可以接受可变表名称并对其操作。步骤109允许新历史行添加到计算机数据仓库。该步骤是为候选插入行(“I”或“U”)更新ETL指示符的两个步骤中的第二个，在此情况下对于到“较早”数据的更新从“I”或“U”到“0”。存在带有“0”的ETL指示符的“早”更新的两个情况：1.源开始时间戳在PK_latest内最后非核心行之前，即使标记为删除，其也称为失序更新；以及2.不符合情况1，因此开始时间戳比在非核心中PK_latest中的任何行更新，但开始时间戳也小于在非核心中的最后结束时间戳。即，该行是较新更新，但由于后面的终止日期已在非核心中，因此一旦输入则已逻辑删除并且标记为终止。通过定义，由于该行的开始时间戳比最后非核心开始时间戳更新，因此该行不是最后行的删除，并且已经标记为“U”。

[0145] 步骤110

[0146] 在示范实施例中，当ALL_VT启用时，步骤110可以接受可变表名称并对其操作。当具有X_table中每主关键字多于一个行设定成“U”从而反映几乎最后行插入到预终止的目标时，步骤110提供结束时间戳。步骤110设定没有指定变为非核心中最后行的所有预终止新行的结束时间戳。带有ETL指示符“0”的所有行需要结束时间戳，并且仅带有不是X_table和非核心中最后的ETL指示符“U”的这些行非核心也得到等于下个行的开始时间的结束时间戳。在应用阶段(例如在下面描述的应用阶段204)中，通过定义，“0”行从随后的非核心行得到其结束时间戳。该步骤可以通过联合或预期算子的使用在单个SQL语句中完成。

[0147] 步骤111

[0148] 在示范实施例中，当ALL_VT启用时，步骤111可以接受可变表名称并对其操作。步骤111设定行的精确开始时间戳为在X_table中的所有删除行（“D”ETL指示符）逻辑删除，并且将该值存储在该行的src_end_ts列中。这为应用阶段（例如在下面描述的阶段206）提供精确的完整主关键字，从而将通过寻找源自非核心的先前行和删除应用到的X表行将终止的单个非核心行定位。最后行和其他预终止X_table行可以更新，但不需要该更新，并且其他步骤可以终止这些行。删除行的源结束时间戳是终止行的源开始时间戳，并且变为当时存在的行的结束时间戳。在示范实施例中，当预先CDC步骤（例如在图3中示出的在步骤92之前的步骤）确定子表格的级联和暗示删除时，步骤111维持引用完整性。步骤111由此促进确保父记录具有应用到它们的对应历史删除，不需要预先CDC过程在调用CDC之前确定精确时间戳。

[0149] 步骤112

[0150] 在示范实施例中，仅当LOAD_TYPE=S（快照加载）或B（带有显式或隐式删除的快照加载）并且ALL_VT=Y（使用可变表格）时，步骤112执行。步骤112将结果数据从易失X_table加载到实际（例如永久的或“实体的”）X_table，允许并行会话使用会话特定可变表处理表数据的唯一分区。

[0151] 当LOAD_TYPE=B时，步骤112首先删除已为当前分区加载到X_table的显式删除。这些显式删除在步骤101中加载到_XVT表并且在当前运行期间处理。这可以首先完成从而防止在X_table中的副本行并移除未处理显式删除。例如，在步骤100期间CDC可以在目标表中排序并设定匹配时间戳。

[0152] 如在图3中示出，当NUM_PARTITIONS>1时，在执行应用步骤之前使用匹配步骤、分区和会话的预生成SQL，为从1到NUM_PARTITIONS的每个分区重复（例如每SESSION_PARALLELISM参数串行或并行）步骤100到112。在错误的情况下，由流程图70图解的过程（在图3中示出）可以停止。在一些实施例中，在实际X_table上的统计量在最后分区加载中收集（例如，其中CURRENT_PARTITION=NUM_PARTITIONS）。

[0153] 图5是图解示范数据应用过程的流程图200。在示范实施例中，在下面描述的应用步骤201-207为每个目标表在单个数据库事务内一起执行，其中每目标表一次起动。若错误检查和行计数阵列合适管理，则步骤201-207中的全部都可以组合成单个多语句请求。如果步骤个别提交并且步骤遇到错误，则应用步骤201-207的执行可以放弃，并且没有进一步的SQL语句可以提交。在这样的情境下，整个事务取消或“回滚”。取决于源变换需求，在开始任何应用步骤201-207之前，CDC过程可以等待所有源表格在步骤100到112中完成。应用步骤可以为所有可应用表格并行执行从而在连续查询访问期间将目标数据库的引用完整性最大化。

[0154] 应用步骤201

[0155] 在示范实施例中，应用步骤201开始数据库事务，其中与直到应用步骤207，在下面描述的结束变换步骤的所有随后步骤关联的SQL语句全部应用，或在SQL语句内任何地方有错误的情况下完全不应用。这促进在有效状况例如每PK_latest至多一个源结束时间戳、至多一个有效行下提出目标表，除非该行已逻辑删除。

[0156] 应用步骤202

[0157] 应用步骤202更新每PK_latest一个最后有效行（源结束时间戳为空），从而将两个

结束时间戳(源和CDW)设定成为ETL指示符“U”将行标记成逻辑删除，由于一个或更多较新行的到达，因此这导致最后非核心时间戳终止。所有删除或指示符“D”的处理在应用步骤206中发生。在步骤202中应用的状况包括变为结束时间的X_table源开始时间戳至少与非核心开始时间一样大(例如时期>0)。

[0158] 应用步骤203

[0159] 在示范实施例中，应用步骤203是将新行插入到非核心的唯一应用步骤。除删除之外的所有ETL指示符导致新的非核心行(I、O和U)。行可以预终止(例如，由于具有值的X_table源结束时间戳列)，或不预终止。当在可以分配事务的或CDW时间戳的任何步骤中的时候，该值表示应用阶段的当前时间戳，通常在应用步骤之前确定并且在每个应用步骤中一贯使用，因此恒定的开始CDW时间戳也在目标表上唯一识别CDC的调用。

[0160] 应用步骤204

[0161] 当在应用步骤203中插入的新行(“O”行的一个情况)具有最后源开始时间戳，但该时间戳比已经在非核心中的最后源结束时间戳早时，应用步骤204校正先前非核心行的结束时间戳。这是已终止或过期行接收带有比现有终止时间戳小的开始时间戳的失序更新的“O”行相对少见的情况。

[0162] 通过包括对X_table上最后主关键字的子查询从而避免目标表完全连结到其自身，当表格具有广阔历史时与潜在巨大处理器应用和潜在另外失真关联的操作，应用步骤204的计算资源利用可以减小。

[0163] 应用步骤205

[0164] 应用步骤205为用ETL指示符“O”标记的行调用。步骤205将所有X_table“O”行连结到非核心表从而如有刚好先前的非核心行的话则确定这些行，并然后用作为结束时间戳的X_table行的开始时间戳更新这些行，以及更新CDW结束时间戳。应用步骤205为失序新行完成将源时间戳扶梯步进(ladder stepping)的过程，从而在源时间戳中提供不同的非重叠时限。当由在主关键字的剩余部分内的源开始时间戳分类时，除标记为逻辑删除的任何行之外，源开始时间戳是刚好之前行(如有的话)的源结束时间戳。

[0165] 应用步骤206

[0166] 应用步骤206为用ETL指示符“D”标记的行调用，并且应用到源自非核心或在批量中新加载行的历史和当前行。该过程将在非核心中的现有结束时间戳更新到删除行的开始时间戳(例如ETL指示符=“D”)。对于直接插入到X_table的行(例如父-子暗示删除)，构建行的预先CDC过程确保结束时间戳仍大于开始时间戳，并小于或等于随后行的源开始时间戳。

[0167] 为确保一对一连结，由于在X_table中的源开始时间戳已经用来记录在目标表中变为行的结束时间戳的删除事件的源时间戳，因此非核心目标行的开始时间戳存储在X_table中的src_end_ts列中。在步骤206中实施的该最终状况适应最后非核心行的逻辑删除并适应历史删除，从而确保当更新结束时间戳时新的结束时间戳较小(例如，可以仅缩短行的寿命或时期，不将其增长，从而确保在PK_latest内没有跨行的源时间戳开始和结束时期的重叠)。

[0168] 应用步骤207

[0169] 最终应用步骤207提出用于结束数据库事务的SQL语句，若没有错误发生，则该SQL语句使自从先前开始事务以来所有先前语句的事务范畴结束。如果没有已经执行，则此时

统计量可以在目标表上收集或刷新。

[0170] 图6-23是进一步解释与所描述变动数据系统关联的步骤中的每个的数据流图。例如，图6是与从非核心数据302加载进入数据的分区的步骤100关联的数据流图300。根据哈希函数通过从非核心数据302选择数据记录的一部分来产生306可变表304。进一步地，在启用历史过滤的情况下(例如TVT_FILTER_HISTORY=Y)，产生306可变表304可以包括基于在W_table308中的数据省略源自非核心数据302的数据记录。另外，当历史过滤启用并且LOAD_TYPE=B时，X_table310可以用作到步骤100的输入。

[0171] 以下是当TVT_FILTER_HISTORY=N，并且LOAD_TYPE=S时与步骤100关联的伪码的例子。

[0172]
```
Step 100 - Pseudo Code (TVT_FILTER_HISTORY= N,
LOAD_TYPE = S):

Create volatile table _TVT as
Select * from target table
Where HASHBUCKET(HASHROW(PK Latest)) MOD
NUM_PARTITIONS
= CURRENT_PARTITION
Primary Index PK Latest;
```

[0173] 以下是当TVT_FILTER_HISTORY=Y，并且LOAD_TYPE=S时与步骤100关联的伪码的例子。

[0174]
```
Step 100 - Pseudo Code (TVT_FILTER_HISTORY= Y,
LOAD_TYPE = S):
```

WITH W table minimum primary key and src start TS where HASHBUCKET(HASHROW(PK

Latest)) MOD NUM_PARTITIONS = CURRENT_PARTITION

Create volatile table _TVT as

Select * from target table

Where full primary key in (

Select newer rows in target table than derived W table with hash partition

[0175] Union

Select latest older row in target table relative to derived W table with hash partition

Union

Select latest row from target table)

Where HASHBUCKET(HASHROW(PK Latest)) MOD NUM_PARTITIONS

= CURRENT_PARTITION

Primary Index PK Latest;

[0176] 以下是当TVT_FILTER_HISTORY=Y,并且LOAD_TYPE=B时与步骤100关联的伪码的例子。

Step 100 - Pseudo Code (TVT_FILTER_HISTORY= Y, LOAD_TYPE = B):

WITH W table minimum primary key and src start TS where HASHBUCKET(HASHROW(PK

Latest)) MOD NUM_PARTITIONS = CURRENT_PARTITION

Create volatile table _TVT as

[0177] Select * from target table

Where full primary key in (

Select newer rows in target table than derived W table with hash partition

Union

Select latest older row in target table relative to derived W table with hash partition

Union

Select latest row from target table)

Or PK_Latest in (select PK_Latest from X table where ETL_Indicator is D from hash partition)

[0178] Where HASHBUCKET(HASHROW(PK Latest)) MOD NUM_PARTITIONS

= CURRENT_PARTITION

Primary Index PK Latest;

[0179] 图7是涉及进入数据的分区从W_table322加载的步骤101的数据流图。可变表_WVT324和_XVT326产生328。根据哈希函数通过从W_table322选择数据记录的一部分来加载_WVT表324。当LOAD_TYPE=B时，源自X_table330的“D”行加载332到可变表_XVT326。

[0180] 以下是当LOAD_TYPE=S时与步骤101关联的伪码的例子。

Step 101 - Pseudo Code (LOAD_TYPE = S):

Create volatile table _WVT as

Select * from W table

Where HASHBUCKET(HASHROW(PK Latest)) MOD NUM_PARTITIONS

[0181] = CURRENT_PARTITION

Primary Index PK Latest;

Create volatile table _XVT as

X table with no data

Primary Index PK Latest;

[0182] 以下是当LOAD_TYPE=B时与步骤101关联的伪码的例子。

Step 101 - Pseudo Code (LOAD_TYPE = B):

Create volatile table _WVT as

[0183] Select * from W table

Where HASHBUCKET(HASHROW(PK Latest)) MOD NUM_PARTITIONS

```
= CURRENT_PARTITION
Primary Index PK Latest;

Create volatile table _XVT as
X table with no data
Primary Index PK Latest;
```

[0184]

```
Insert into _XVT (PK, 4 row marking columns)
Select PK, 4 row marking columns from X table
Where HASHBUCKET(HASHROW(PK Latest)) MOD
NUM_PARTITIONS
= CURRENT_PARTITION
AND ETL_INDICATOR = 'D'
```

[0185] 图8是与为显式删除构建X_table行的步骤102关联的数据流图340。图示340图解如果源自非核心数据342的行不再在W_table344中出现，则假设其已从来源删除。行以源自非核心数据342的最后主关键字(PK_latest)、当前时间戳和ETL_Indicator“D”插入346到X_table342，其中“当前”非核心主关键字不在W_table344中。这是其中表格不取决于父表格的简单情况。

[0186] 以下是与步骤102关联的伪码的例子。

[0187] Step102-Pseudo Code:

[0188] Insert into X_table

[0189] Select[*,Current_Timestamp,'D']from target

[0190] WHERE PK-Latest NOT IN

[0191] (Select[PK_Latest]from W-table);

[0192] 图9是与步骤103关联的数据流图360。在W_table362内的数据记录与在非核心数据364内的数据记录比较从而识别366未变动较新行的主关键字。带有相同内容的较早进入数据行的这些主关键字存储在可变表_KVT368中。

[0193] 以下是与步骤103关联的伪码的例子。

[0194] Step103-Pseudo Code:

[0195] Create VT of W table full PK's to exclude in step104

[0196] Select full PK from W table

[0197] Where full PK in(

[0198] Select earliest full PK row from W table joined to target table

[0199] Where target table is latest row and W table is next newest row

[0200] And all attributes of both rows are identical excluding source TS

[0201] 图10是涉及为新且变动的记录构建X_table行、非不同的完整主关键字的序列以及在X_table382上统计量的收集的步骤104的数据流图380。如果至少一个列不同于在非核

心数据386中的所有其他行，这允许新数据历史加载到X_table，则行插入384到X_table382。通过选择所有W_table388行减去(SQL except)所有非核心386行，默认ETL指示符设定成1。变动数据捕捉系统和主关键字完整性需要这样步骤，如果在数据中受保证则不总是需要这样步骤。一微秒为PK_latest内第二到第N行添加到W_table388中的src_start_ts时间戳。相似地，为任何时间戳排序保留的源时间戳的最后三个次第二数字从W_table和非核心之间的变动比较排除。

[0202] 以下是与步骤104关联的伪码的例子。

```
Step 104 - Pseudo Code:

Insert into X_table
Select [*] from W_table  -- 1 microsecond sequencing added to start TS
```

[0203]

```
Where * not in (  -- exclude microsecond sequencing when selecting start TS
Select [*] from target);  -- exclude ns sequencing when selecting start TS
Collect statistics on X_table;
```

[0204] 进一步地，当NORMALIZE_LATEST=Y时，_KVT表390可以由主步骤103用主关键字填充，如在上面描述。在这样的情境下，步骤104排除(例如不插入384到X_table382)与在_KVT表390中出现的主关键字关联的数据记录。

[0205] 图11是与步骤105关联的数据流图。与在X_table404中出现的PK_Latest关联的非核心数据402内的数据记录基于在非核心数据402中的源开始时间戳被滤除，并且插入406到易失目标表408。

[0206] 以下是与步骤105关联的伪码的例子。

```
Step 105 - Pseudo Code:

Insert into Target_Table_VT (create volatile temporary table via select statement)
Select * from Target Table where
```

[0207]

```
PK_Latest in (select PK_Latest from X_table)
And (SOURCE_START_TS >= MIN SRC_START_TS in X table for that exact PK
OR SOURCE_START_TS is MAX for PK < MIN SRC_START_TS in X)
```

[0208] 图12是与X_table422行的排序的步骤106关联的数据流图，该排序进而更新现有非核心424行。步骤106的意图是通过将最大非核心时间戳(TS_microseconds)添加到相同PK内带有另外相等开始时间戳(排除最后3个次第二数字)的所有X_table“I”行，更新与X_table422关联的源开始时间戳。如果是新的，则用于主关键字(PK)的已排序(在步骤104中)

行426被接收，具有与现有行相同的时间戳(TS)，确保在非核心行424之后新行落入序列。

[0209] 以下是与步骤106关联的伪码的例子。

```
Step 106 - Pseudo Code:

UPDATE X-alias
FROM X-table X-alias
  , ( SELECT
    PK_Latest
    , CAST ( MAX (src_start_ts) as char(23) ) F23C
    , substring ( cast (max(source_start_ts) as char(26) ) from 24
for 3 ) + 1 L3C
    , substring ( cast (source_start_ts as char(32) ), from 27 for
6 ) + 1 L3C
    FROM target
    GROUP BY PK_Latest ,F23C,TSTZ ) QQQ
SET SRC_START_TS
  = F23C || SUBSTRING(CAST((L3C / 1000 +
  (SUBSTRING(cast(xpm.src_start_ts as char(26)) FROM 24
FOR 3))/1000) AS DEC(4,3)) FROM
  4 FOR 3)
WHERE X-alias.PK_Latest = QQQ.PK_Latest
    AND CAST ( X-alias.src_start_ts AS CHAR(23) ) =
QQQ.F23C
    AND X-alias.ETL_INDICATOR = 'I' ;
```

[0210]

[0211]

[0212] 图13是与在例如X_table442和非核心表444的联合，以及另外地X_table446和非核心表448的联合内放弃连续冗余X_table行的步骤107关联的数据流图。两个联合在主关键字上连结从而将行排序，并允许关于所有非关键字属性复制并然后连结到最新目标表PK的连续行的检测。步骤107表示当包括相同数据的行加载多于一次并且实施时态时限压缩单元时资源浪费的确认。然而，仍希望记录从“A”到“B”然后回到“A”的数据中的任何变动。因此，排除开始时间戳的每个不同行的第一实例维持在X_table450中。更具体地，在X_table和非核心表的联合内，如果PK_latest相同，并且如果当由在PK_latest内的源开始时间戳分类时除时间戳之外的所有列与先前行相同，则删除在X_table450内的数据。

[0213] 在示范实施例中，包括X_table452和非核心表454的另外连结(代号为表C)从而即使逻辑删除仍寻找最新源开始时间戳，并将开始日期(或年份2500，如果没有这样的行存在)和结束日期返回从而与最后X_table行比较

[0214] 以下是与步骤107关联的伪码的例子。

Step 107 - Pseudo Code:

[0215]
```
Delete from X_table where PK IN (
(Select PK from
    (Select A.* from
      (Select *, table_source, Row_Number() from X_table union noncore
        partition by PK_Latest Order by SRC_START_TS to create Row_Number) A
      INNER JOIN (Select *, table_source, Row_Number() from X_table union noncore
        partition by PK_Latest Order by SRC_START_TS to create Row_Number) B
    Where A.PK_Latest = B.PK_Latest
      and B.Row_Number = A.Row_Number - 1
```

[0216]
```
      and all non-key attribute are the same (values equal or both null)
    )
    AND A.Table Source = 'X'
    Left Outer Join (Select PK_Latest, If not null then Source Start TS else Year 2500,
    SOURCE_END_TS
      From Target partition PK_Latest and present newest Source Start ts) C
    ON A.PK_Latest = C.PK_Latest
    WHERE ( B.Table Source = 'X'
      AND A.Time period is newer than the latest target row
```

[0217] 图14是与标记X_table462的行的步骤108关联的数据流图460，该行更新到在非核心数据464内的当前行。在步骤108中，为更新466X_table，ETL_Indicator在更新现有非核心“当前”行的“I”行上设定成“U”，其中进入源时间戳大于在非核心表中的最后源时间戳。在这里描述的应用步骤202中，这些“U”行中的最早“U”行的开始时间戳在主关键字内用来终止最后非核心行。

[0218] 以下是与步骤108关联的伪码的例子。

Step 108 - Pseudo Code:

[0219]
```
UPDATE X_tbl
```

```
FROM X_TABLE X_tbl
   , (select PK_Latest
   , max(src_start_ts) src_start_ts
   from target
   group by PK_Latest ) NC_tbl
SET ETL_INDICATOR = 'U'
WHERE X_tbl.PK_Latest = NC_tbl.PK_Latest
   AND X_tbl.SRC_START_TS > NC_tbl.SRC_START_TS
   AND X_tbl.ETL_INDICATOR = 'I' ;
```

[0221] 图15是图解标记X_table482中的行的步骤109的数据流图480，这些行更新到在非核心数据484中的“历史”行。流图480涉及失序应用的更新。在步骤109中，为更新486在X_table482内的数据，ETL_Indicator在先前设定成I或U的行中设定成“0”以便现有非核心“历史”行的更新。在这样的行中，一旦考虑整个X_table和非核心表行，则进入源时间戳小于在非核心行中的最后源开始时间戳。这通过将源自非核心和X_table的时间戳比较组合从而实现每PK_latest的整体最大值来实现。可替换地，进入源时间戳小于最后源结束时间戳，因此该行应预终止。这些“0”行更新非核心数据中的结束时间戳从而校正历史序列。这些更新失序应用，因此大多数在步骤110中得到下个行的结束时间戳。其他在应用步骤204中得到结束时间戳。

[0222] 以下是与步骤109关联的伪码的例子。

```
Step 109 - Pseudo Code:

UPDATE X_tbl
FROM X_TABLE X_tbl
   , (select PK_Latest
   , max(src_end_ts) max_end_ts
   , max(src_start_ts) max_start_ts
   from target
   group by PK_Latest ) Max_tbl
SET ETL_INDICATOR = 'O'
WHERE X_tbl.PK_Latest = Max_tbl.PK_Latest
      AND ( (X_tbl.SRC_START_TS < Max_tbl.MAX_END_TS
        OR ( X_tbl.SRC_START_TS <
Max_tbl.MAX_START_TS ) )
      AND X_tbl.ETL_INDICATOR IN ( 'I', 'U') ;
```

[0223]

[0224]

[0225] 图16是图解已经在非核心中或在X_table中更新的X_table行(“0”或“U”)终止的步骤110的数据流图500。在X_table行中的结束时间戳设定成先前行(在非核心或X_table

中)的开始时间戳。其中ETL_Indicator设定成“0”的行允许加载历史更新。这些是进入行，该进入行不是在其主关键字内的最后行，即它们已经联合506跨X_table502和非核心504更新。它们基于其结束时间戳插入到非核心数据作为历史行。

[0226] 以下是与步骤110关联的伪码的例子。

```
Step 110 - Pseudo Code:

Update X-tbl
FROM X-table X-tbl
   , ( Select AAA.PK-Latest, min(BBB.START_TS) as END_TS
   From ( Select PK
      From X-table ) AAA,
      ( Select PK
      From X-table
UNION
      Select PK
      From target ) BBB
      Where BBB.PK_Latest = AAA.PK_Latest
      And BBB.START_TS > AAA.START_TS
      Group By AAA.PK
   ) QQQ
SET END_TS = QQQ.END_TS
WHERE X-table.PK = QQQ.PK
   and X-table.ETL_Indicator IN ('O', 'U');
```

[0227]

[0228] 图17是图解通过寻找逻辑删除应用到的最后非核心行的时间戳，为所有删除行(“D”的ETL_Indicator)提供完整关键字(带有源开始时间戳)的步骤111的数据流图520。更具体地，基于删除时间戳，结束时间戳在X_table行上设定成刚好先前的X_table行或非核心行的开始时间戳。在变动数据捕捉之前，有待逻辑删除的这些行另外是已经准备好的父子暗示或级联删除行。这样的情况在应用步骤206中执行。

[0229] 以下是与步骤111关联的伪码的例子。

Step 111 - Pseudo Code:

```
Update X-tbl
FROM X-table X-tbl
  , ( Select AAA.PK-Latest, min(BBB.START_TS) as END_TS
  From ( Select PK
    From X-table ) AAA,
    ( Select PK, Max Start TS
    From X-table
UNION
    Select PK, Max Start TS
    From target ) BBB
    Where BBB.PK_Latest = AAA.PK_Latest
    And BBB.START_TS > AAA.START_TS
    Group By AAA.PK, BBB.Max Start TS
  ) QQQ
SET END_TS = QQQ.END_TS
WHERE X-table.PK = QQQ.PK and X-table.Start TS < QQQ.Start TS
  and X-table.ETL_Indicator = 'D';
```

[0230]

[0231] 图18是图解数据记录从_XVT表544插入到X_table546的步骤112的数据流图。在数据已插入542到X_table546之后，步骤112包括放弃548为分区产生的所有可变表。在某些情况下(例如当NORMALIZE_LATEST=N并且LOAD_TYPE=B时)，步骤112可以包括在X_table546中删除550在当前分区中的带有"D"的ETL指示符的行。

[0232] 以下是当NORMALIZE_LATEST=N并且LOAD_TYPE=S时与步骤112关联的伪码的例子。

Step 112 - Pseudo Code (NORMALIZE_LATEST = N, LOAD_TYPE = S):

```
INSERT INTO X-table SELECT * FROM _XVT;

DROP TABLE _XVT;
DROP TABLE _WVT;
DROP TABLE _TVT;
DROP TABLE _VT;

COLLECT STATISTICS X-table;  (last partition only)
```

[0233]

[0234] 以下是当NORMALIZE_LATEST=Y并且LOAD_TYPE=S时与步骤112关联的伪码的例子。

Step 112 - Pseudo Code (NORMALIZE_LATEST = Y, LOAD_TYPE = S):

```
INSERT INTO X-table SELECT * FROM _XVT;

DROP TABLE _XVT;
DROP TABLE _WVT;
DROP TABLE _TVT;
DROP TABLE _VT;
DROP TABLE _KVT

COLLECT STATISTICS X-table;  (last partition only)
```

[0236] 以下是当NORMALIZE_LATEST=N并且LOAD_TYPE=B时与步骤112关联的伪码的例子。

Step 112 - Pseudo Code (NORMALIZE_LATEST = N, LOAD_TYPE = B):

[0237]

```
DELETE FROM X-table
WHERE ETL_INDICATOR = 'D'
AND
HASHBUCKET(HASHROW(USAGE_INSTANCE_NUM_ID))
MOD NUM_PARTITIONS = 0;

INSERT INTO X-table SELECT * FROM _XVT;
DROP TABLE _XVT;
DROP TABLE _WVT;
DROP TABLE _TVT;
DROP TABLE _VT;

COLLECT STATISTICS X-table;  (last partition only)
```

[0238]

[0239] 第一应用步骤，步骤201确保直到应用步骤207，END TRANSACTION的所有随后SQL语句全部应用，或在语句内任何地方有错误的情况下完全不应用。这对于将目标表留在有效状况(例如，每PK_latest至多一个SOURCE_END_TS、至多一个有效行，除非该行已逻辑删除)是必需的。

[0240] 以下是与应用步骤201关联的伪码的例子。

[0241] Step 201 Pseudo Code:

[0242] STARTTRANSACTION

[0243] 图19是图解应用步骤202的数据流图600，应用步骤202是已更新非核心602行的先前版本的终止。为更新非核心行，结束时间戳从空值更新604到源自X_table606的最后主关键字的最早后继行的开始时间戳，其中ETL指示符设定成“U”。该步骤覆盖为更新设定最后非核心行的结束时间戳，其中一个行的结束时间戳是下个行的开始时间戳。

[0244] 以下是与应用步骤202关联的伪码的例子。

Step 202 - Pseudo Code

[0245]
```
UPDATE noncore
SET SOURCE_END_TS = MIN(X-table.SRC_START_TS)
   CDW_END_TS = current timestamp for table
      WHERE noncore.PK_Latest = X-table.PK_Latest
        AND SOURCE_END_TS IS NULL
        AND                X-table.SRC_START_TS                >=
```

[0246]
```
noncore.SOURCE_START_TS
        AND X-table.ETL_INDICATOR = 'U'
        AND src_start_ts is the earliest within the PK_Latest;
```

[0247] 图20是图解应用步骤203的数据流图620，应用步骤203是为ETL指示符I、0和U将新行插入622到非核心624。除为删除标记的行之外，所有进入行都加载。具有I的ETL_Indicator的行和具有U的ETL_Indicator的一些行变为最后行，剩余U和所有0行预终止。当用于来源的结束时间戳在X_table626中不为空(在大多数情况下的ETL指示符0和U)的时候，情况语句用来将结束时间戳设定成固定的当前时间戳。除删除之外所有进入行都加载。I行和每主关键字一个U行可以变为最后(无结束时间戳)，而另外的U行预终止。

[0248] 以下是与应用步骤203关联的伪码的例子。

[0249] Step203-Pseudo Code

[0250] INSERT into noncore

[0251] Select*from X-table

[0252] Where ETL_Indicator='I','0'or'U';

[0253] 图21是图解应用步骤204的数据流图640，应用步骤204是在非核心数据642中新插入的“0”行应从先前行继承较晚的终止日期时的更新。这是其中行已经删除(终止644)但接收失序更新的情况，该失序更新在删除发生之前但在逻辑删除行的开始时间之后发生。由于新插入行具有最后的源开始时间戳，因此其应得到较晚的结束时间戳。应用步骤204可以仅从X_table646过滤“PK”。

[0254] 以下是与应用步骤204关联的伪码的例子。

Step 204 - Pseudo Code

[0255]
```
UPDATE NC_Tbl
FROM
```

```
        noncore NC_Tbl,
        ( SELECT PK_Latest MAX ( SOURCE_END_TS ) MAX_END_TS
          FROM noncore
          WHERE ( PK_Latest in ( SELECT PK_Latest FROM X_table ) )
        GROUP BY PK_Latest ) Max_NC
      SET SOURCE_END_TS = Max_NC.MAX_END_TS,
        CDW_END_TS = current timestamp for table
      WHERE NC_Tbl.PK_Latest = Max_NC.PK_Latest
        AND NC_Tbl.SOURCE_START_TS < Max_NC.MAX_END_TS
        AND NC_Tbl.SOURCE_END_TS IS NULL ;
```

[0256]

[0257] 图22是图解应用步骤205的数据流图660，应用步骤205是已经终止但具有在应用步骤203期间此后立即插入的“丢失”更新664的非核心662行上的结束时间戳的更新。在应用步骤205中，由于插入“0”行，因此“0”行从X_table666使用从而校正现在具有不同后继行的行的结束时间戳。

[0258] 以下是与应用步骤205关联的伪码的例子。

```
Step 205 - Pseudo Code

UPDATE NC_Tbl
FROM noncore NC_Tbl ,
( SELECT NC_Tbl.PK_Latest
  , X_Tbl.SRC_START_TS SRC_END_TS
  , MAX ( NC_Tbl.SOURCE_START_TS )
```

[0259]

```
SOURCE_START_TS
    FROM ( SELECT PK_Latest
      , SRC_START_TS
      FROM X-Table
      WHERE ETL_INDICATOR = 'O' ) X_Tbl
    , ( SELECT PK_Latest
      , SOURCE_START_TS
```

```
        FROM noncore ) NC_Tbl
      WHERE NC_Tbl.PK_Latest = X_Tbl.PK_Latest
      AND          NC_Tbl.SOURCE_START_TS          <
X_Tbl.SRC_START_TS
      GROUP BY NC_Tbl.PK_Latest , X_Tbl.SRC_START_TS
[0260] ) QQQ
    SET SOURCE_END_TS = QQQ.SRC_END_TS,
      CDW_END_TS = current timestamp for table
    WHERE NC_Tbl.PK_Latest = QQQ.PK_Latest
      AND          NC_Tbl.SOURCE_START_TS          =
QQQ.SOURCE_START_TS ;
```

[0261] 图23是图解应用步骤206的数据流图680，应用步骤206是由逻辑删除引起的非核心行的终止。对于“D”的ETL_Indicator，源开始时间戳保存到在X_table682中的源结束时间戳从而提供完整目标主关键字，以及更新684结束时间戳从而删除时间戳值。

[0262] 以下是与应用步骤206关联的伪码的例子。

```
Step 206 - Pseudo Code

UPDATE NC_Tbl
FROM noncore NC_Tbl, x table X_Tbl
Set NC_Tbl source end ts = X tbl source start ts,
  NC_Tbl.cdw end ts = current timestamp for table
[0263] Where NC_Tbl.PK_latest = X_Tbl.PK_latest
  And NC_Tbl.source_start_ts = X_Tbl.src_end ts -- ensures
1-to-1
  And X_Table.ETL_Indicator is 'D'
  And NC_Tbl source end ts is null or greater than X table start
ts
```

[0264] 以下是与应用步骤207关联的伪码的例子。

[0265] Step207-Pseudo Code

[0266] END TRANSACTION

[0267] Evaluate whether to refresh noncore statistics

[0268] 在示范实施例中，所有变动数据捕捉(CDC)过程(在上面描述的步骤和应用步骤)在任何随后小批量数据加载的该部分开始写入到W_table和/或X_table，或为给定源系统或一组目标表调用CDC之前完成。同样，整个加载过程没有串行化，仅到W_table和/或X_table的写入与CDC的调用。

[0269] 在上面描述的实施例用来以支持每次10或15分钟的小批量调度的足够效率，从任何源系统无修改加载任何量的数据到连续可用的时态规格化数据仓库。这些实施例可以用

较少的数据库特定调整(例如列出列的目录表的名称),在支持ANSI SQL-2003标准(或带有一些转换的较低标准)的任何关系数据库上用来加载保留历史并且不需要有效强制引用完整性的时态规格化数据仓库。此外,特别是关于将载荷分区的已实施元数据最优化参数的使用促进减小容许数据加载的每加载过程的计算资源成本充分减小,从而容许数据在增加的潜伏期加载,在使用对称多处理(SMP)架构而不是在数据仓库中普遍使用的昂贵得多的大规模并行处理(MPP)架构的商品计算机服务器上加载。实施例在一组候选行内并在这些行和目标数据库之间操作,允许在主关键字内的多个行立即处理、排序并且如有关于时间间隔的邻接副本则压缩。

[0270] 因此在至少一个实施例中,提供填充时态规格化数据仓库的方法,包括关于一组进入数据自身和现有数据仓库分析该组进入数据,使用关系代数集运算符(set-SQL)将净变化数据识别并排序,并且当数据保持在数据仓库自身内的时候向数据仓库应用输入和时态更新。为完成在上面描述的方法,软件代码可以动态生成从而执行数据插入和时态更新,并且已生成代码然后执行。另外,邻接数据压缩到最小数目的时限,并且在唯一时间戳内的微秒级序列按需要生成并维持。

[0271] 在至少一个实施例中提供系统,其包括数据仓库、接收有待存储到数据仓库的多组进入数据的容量,以及可操作从而将在已接收数据对先前存储在数据仓库中的数据的净变化识别并排序的排序单元。系统可以包括可操作从而将邻接数据压缩到最小时限的压缩单元、生成执行数据仓库更新的代码的自动编码器单元和执行已生成代码的执行单元中的一个或更多。排序单元可操作从而利用关系代数集运算符使用广泛接受的ANSI标准结构化查询语言(SQL)数据操纵语言(DML)识别并排序在该实例中实施的数据。

[0272] 广泛认识到更大的成功公司移至具有用于如果不是全部则是大部分的分析需要的单个或小数目的规格化时态数据仓库,替代数百个操作数据存储和数据集市(data marts)。该范例已为完全新型的有效且相对非干扰的加载软件产生充分需要。已描述实施例为时态数据仓库提供开发和维持成本的显著减小,提供在数据加载服务器使用上的巨大成本避免,并且完全避免加载时期期间支持连续查询可用性的目标数据库的第二复制的任何需要(第二复制可以仍为其他原因利用)。由于时态数据仓库使用全球地增长,并且没有当前技术提供相似能力,同时也唯一使得能够实现从连续可用数据的简单复制跨无论是一个或更多主题范围或整个企业的数据仓库范畴支持多类型分析需要(例如操作的、战术的、战略的)的数据仓库战略,因此已描述实施例可应用于任何数据库平台。

[0273] 已描述系统和方法支持单个规格化数据仓库的近实时的、最小干扰的加载,这进而使得能够在单个系统中以规格化到所需要最小时限的全时态历史实现经适当安全性和授权控制的到所有数据的单个复制的近立即外部访问。

[0274] 在示范实施例中,进入数据集的分区是任选的。分区可以独立处理,以使执行过程不从多于一个分区访问数据。相反,处理每个分区的结果可以累积到X_table以便在单个应用步骤(未并行化或分区)中使用,如在上面进一步描述。

[0275] 一些实施例在输入进入数据集中实施易失的(例如非永久的)表格。一组完整可变表可以例如在分区被指定时使用。进一步地,数据可以在计算机数据仓库(CDW)中的现有数据和进入数据集之间规格化(例如时态地规格化),无论使用常规的、永久的("实体的")表格并且无分区或使用分区和实体表的易失复制,每分区一组(例如每分区四个虚拟表)。

[0276] 本发明的实施例可以使用一个或更多计算装置执行，例如数据服务器16和/或应用服务器24(在图2中示出)。图24是示范计算装置700的框图。在示范实施例中，计算装置700包括通信构造705，该通信构造705在处理器单元710、存储器715、永久存储720、通信单元725、输入/输出(I/O)单元730和呈现接口例如显示器735之间提供通信。另外或可替换地，呈现接口可以包括音频装置(未示出)和/或能够向用户传达信息的任何装置。

[0277] 处理器单元710为可以加载到存储器715的软件执行指令。处理器单元710可以是一组一个或更多处理器或可以包括处理器核心，取决于特别实施。进一步地，处理器单元710可以使用一个或更多异构处理器系统实施，其中主处理器与副处理器一起在单芯片上存在。在另一实施例中，处理器单元710可以是含有相同类型的多个处理器的同构处理器系统。

[0278] 存储器715和永久存储720是存储装置的例子。如在此使用，存储装置是能够基于暂时和/或基于永久存储信息的硬件的任何部分。存储器715可以是无限制例如随机访问存储器和/或任何其他合适易失性或非易失性存储装置。永久存储720可以采取各种形式，取决于特别实施，并且永久存储720可以含有一个或更多部件或装置。例如，永久存储720可以是硬盘驱动器、闪存存储器、可重写光盘、可重写磁带和/或以上的一些组合。由永久存储720使用的介质也可以是可移除的。无限制例如可移除硬盘驱动器可以用于永久存储720。

[0279] 存储装置例如存储器715和/或永久存储720可以经配置存储数据以便与在此描述的过程一起使用。例如，存储装置可以存储计算机可执行指令、可执行软件部件(例如数据加载部件和/或数据仓库部件)、从数据源接收的数据、配置数据(例如最优化选项)和/或适合与在此描述的方法一起使用的任何其他信息。

[0280] 通信单元725在这些例子中提供与其他计算装置或系统的通信。在示范实施例中，通信单元725是网络接口卡。通信单元725可以提供使用实体或无线通信链接提供通信。

[0281] 输入/输出单元730使得能够实现数据与连接到计算机装置700的其他装置的输入和输出。无限制例如输入/输出单元730可以通过用户输入装置例如键盘和/或鼠标为用户输入提供连接。进一步地，输入/输出单元730可以向打印机发送输出。显示器735提供向用户显示信息的机构。例如，呈现接口如显示器735可以显示图形用户接口。

[0282] 用于操作系统和应用程序或程序的指令位于永久存储720上。这些指令可以加载到存储器715以便由处理器单元710执行。不同实施例的过程可以由处理器单元710使用位于存储器例如存储器715中的计算机实施指令和/或计算机可执行指令执行。这些指令在此称为可以由处理器单元710中的处理器读取并执行的程序代码(例如目标代码和/或源代码)。在不同实施例中的程序代码可以在不同的实体或有形计算机可读介质例如存储器715或永久存储720上实施。

[0283] 程序代码740以功能形式位于选择性可移除的非暂时计算机可读介质745上，并且可以加载到或转移到计算装置700以便由处理器单元710执行。程序代码740和计算机可读介质745在这些例子中形成计算机程序产品750。在一个例子中，计算机可读介质745可以是有形形式例如光或磁盘，其插入或放入驱动器或是永久存储720的部分的其他装置，以便转移到存储装置例如是永久存储720的部分的硬盘驱动器。以有形形式，计算机可读介质745也可以采取连接到计算装置700的永久存储的形式，例如硬盘驱动器、拇指驱动器或闪存存储器。计算机可读介质745的有形形式也称为计算机可记录存储介质。在一些实例中，计算

机可读介质745可以是不可移除的。

[0284] 可替换地，程序代码740可以从计算机可读介质745通过到通信单元725的通信链接和/或通过到输入/输出单元730的连接转移到计算装置700。通信链接和/或连接在图解例子中可以是实体的或无线的。计算机可读介质也可以采取非有形介质的形式，例如含有程序代码的通信链接或无线传输。

[0285] 在一些图解实施例中，程序代码740可以从另一计算装置或计算机系统经由网络下载到永久存储720以便在计算装置700内使用。例如，存储在服务器计算装置中的计算机可读存储介质中的程序代码可以从服务器经由网络下载到计算装置70。提供程序代码740的计算装置可以是服务器计算机、工作站、客户端计算机或能够存储并传输程序代码740的一些其他装置。

[0286] 程序代码740可以组织成功能上相关的计算机可读部件。每个部件可以包括计算机可执行指令，当由处理器单元710执行时该计算机可执行指令导致处理器单元710执行在此描述的操作中的一个或更多。

[0287] 用于计算装置700的在此图解的不同部件不意味着向其中可以实施不同实施例的方式提供架构限制。不同图解实施例可以在包括除为计算装置700图解的部件或代替这些部件的部件的计算机系统中实施。例如，在图24中示出的其他部件可以从示出的图解例子变化。

[0288] 作为一个例子，在计算装置700中的存储装置是可以存储数据的任何硬件设备。存储器715、永久存储720和计算机可读介质745是有形形式的存储装置的例子。

[0289] 在另一例子中，总线系统可以用来实施通信构造705并可以包括一条或更多总线，例如系统总线或输入/输出总线。当然，总线系统可以使用在附装到总线系统的不同部件或装置之间提供数据转移的任何合适类型的架构实施。另外，通信单元可以包括用来传输和接收数据的一个或更多装置，例如调制解调器或网络适配器。进一步地，存储器可以是无限制例如存储器715或缓存，例如可以在通信构造715中存在的接口和存储器控制器集线器中发现的缓存。

[0290] 上面描述的实施例提供用于将进入数据集加载到时态数据仓库的系统。这样的系统包括存储装置和耦合到存储装置的处理器单元，存储装置包括时态数据仓库和进入数据集。在至少一个实施例中的处理器经编程将进入数据集划分为包括第一分区和第二分区的多个分区，其中该多个分区中的每个分区包括多个数据记录，将第一分区输入到预加载表、将第二分区输入到预加载表并将预加载表应用到时态数据仓库。

[0291] 在另一实施例中，处理器单元经编程至少部分通过将哈希函数应用到与至少一个数据记录关联的主关键字从而产生对应于该至少一个数据记录的哈希值，将进入数据集划分为多个分区。在另一实施例中，处理器单元经进一步编程在第一分区预加载到表格之后，将第二分区输入到预加载表。在更另一实施例中，处理器单元经进一步编程在第一分区输入到预加载表时，将第二分区输入到预加载表。在这样的实施例中，处理器单元经编程基于确定并行输入的当前量小于并行输入的预定最大量，在第一分区输入到预加载表时将第二分区输入到预加载表。

[0292] 在可替换实施例中，处理器单元经编程至少部分通过将分区的数据记录输入到对应于分区的可变表，并将源自可变表的数据记录复制到预加载表，输入第一分区和第二分

区中的至少一个。

[0293] 在更其他实施例中，处理器单元经进一步编程在第一分区中识别包括除时间戳之外的多个字段的数据记录，该多个字段等于先前输入数据记录的非关键字段，并且在将第一分区输入到预加载表时排除已识别数据记录。进入数据可以包括源自源数据库的数据的快照，并且在这样的实施例中处理器单元经进一步编程检测在时态数据仓库中的有效数据记录不与在进入数据集中的数据记录关联，并且基于所述检测执行有效数据记录的隐式删除。处理器单元可以经进一步编程确定与在进入数据集中的第一数据记录关联的最早源时间戳，并且识别表示以下内容的一主关键字组：在时态数据仓库中的数据记录，该数据记录与刚好在最早源时间戳之前的源时间戳关联，以及在时态数据仓库中的一个或更多数据记录，该一个或更多数据记录与比最早源时间戳晚的源时间戳关联，以及基于已识别主关键字组输入第一分区和第二分区。

[0294] 另外，在此描述的实施例提供用于将多个数据记录加载到时态数据仓库的方法。这样的方法包括由计算装置将数据记录划分为包括第一分区和第二分区的多个分区，由计算装置将第一分区输入到预加载表，由计算装置将第二分区输入到预加载表，以及由计算装置将预加载表应用到时态数据仓库。在方法的某些实施例中，第一分区和第二分区并行输入。方法可以进一步包括确定并行输入的当前量小于并行输入的预定最大量，其中基于所述确定将第一分区和第二分区并行输入。方法可以进一步包括确定并行输入的当前量大于或等于并行输入的预定最大量，其中基于所述确定将第一分区和第二分区循序输入。

[0295] 方法的可替换实施例考虑由计算装置将数据划分为多个分区包含将哈希函数应用到至少一个数据记录从而产生与该至少一个数据记录关联的哈希值，并且基于分区的预定量将模运算符应用到哈希值，从而确定与至少一个数据记录对应并关联的分区数目。

[0296] 方法的实施例也可以包括在第一分区中识别包括除时间戳之外的多个字段的所述数据记录，该多个字段等于先前输入数据记录的非关键字段，并且在将第一分区输入到预加载表时，排除已识别数据记录。

[0297] 上面描述的实施例可以进一步表征为包含非暂时计算机可读介质的计算机程序产品，该非暂时计算机可读介质具有在其上实施的用于将净变化数据加载到数据仓库的计算机可执行指令。当由至少一个处理器执行时，计算机可执行指令导致处理器将进入数据集划分为包括第一分区和第二分区的多个分区，其中该多个分区中的至少一个分区包括多个数据记录，将第一分区输入到预加载表、将第二分区输入到预加载表并将预加载表应用到数据仓库。

[0298] 在进一步的实施例中，计算机可执行指令导致至少一个处理器将第一分区和第二分区相互并行输入。而在其他实施例中，计算机可执行指令进一步导致至少一个处理器将并行输入的当前量与并行输入的预定最大量比较，在当前量小于并行输入的最大量时将第二分区和第一分区并行输入，并且在当前量大于或等于并行输入的最大量时在第一分区的输入之后输入第二分区。

[0299] 在更其他实施例中，计算机可执行指令导致至少一个处理器至少部分通过将第一分区的数据记录输入到对应于第一分区的第一可变表，将第二分区的数据记录输入到对应于第二分区的第二可变表，并将第一可变表和第二可变表的数据记录复制到预加载表，输入第一分区和第二分区。

[0300] 在更其他实施例中，进入数据集包括源自源数据库的数据的快照，并且计算机可执行指令进一步导致至少一个处理器检测在时态数据仓库中的有效数据记录不与在进入数据集中的数据记录关联，并且基于所述检测执行有效数据记录的隐式删除。

[0301] 本书面描述使用例子披露包括最优模式的本发明，并且也使得任何本领域技术人员能够实践本发明，包括制作并使用任何装置或系统并执行任何所包括方法。本发明的专利范围由权利要求定义，并且可以包括对本领域技术人员发生的其他例子。如果这样的其他例子具有不与权利要求的文字语言不同的结构元件，或它们包括与权利要求的文字语言没有实质不同的等效结构元件，则意图将它们包括在权利要求的保护范围内。

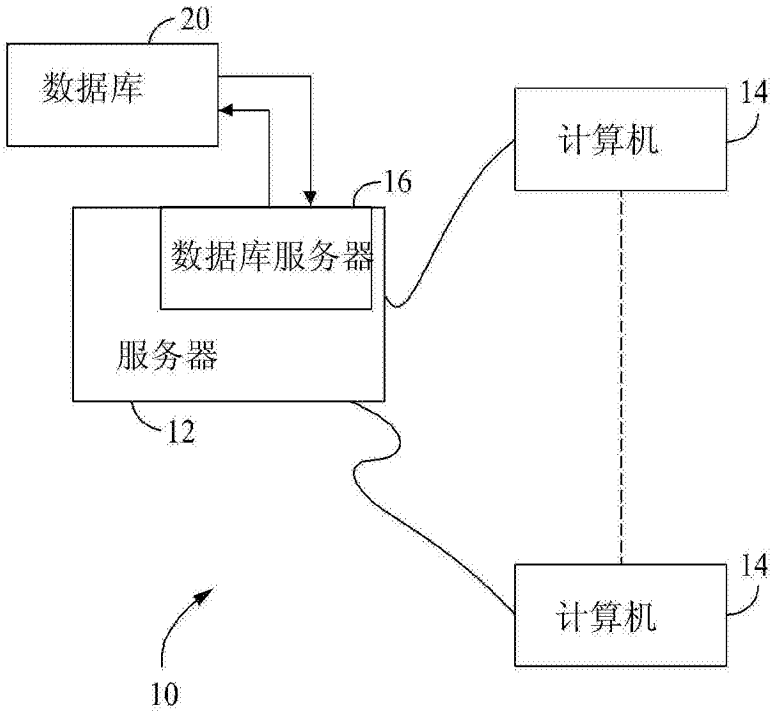


图1

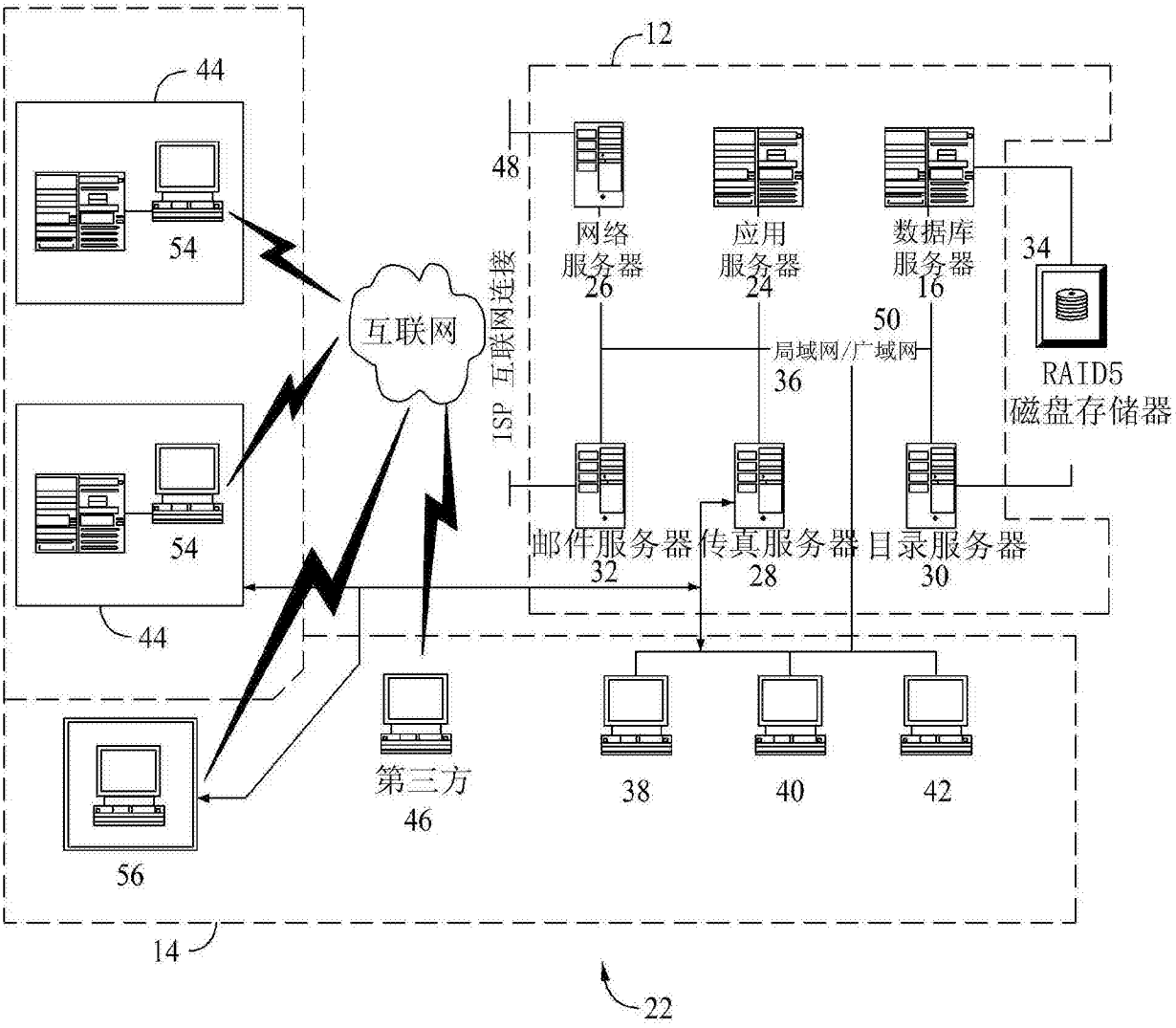

12
44
54
48
网络
服务器
26
应用
服务器
24
数据库
服务器
16
34
互联网
ISP 互联网连接
局域网/广域网
36
50
RAID5
磁盘存储器
54
邮件服务器
32
传真服务器
28
目录服务器
30
44
第三方
46
38
40
42
56
14
22

图2

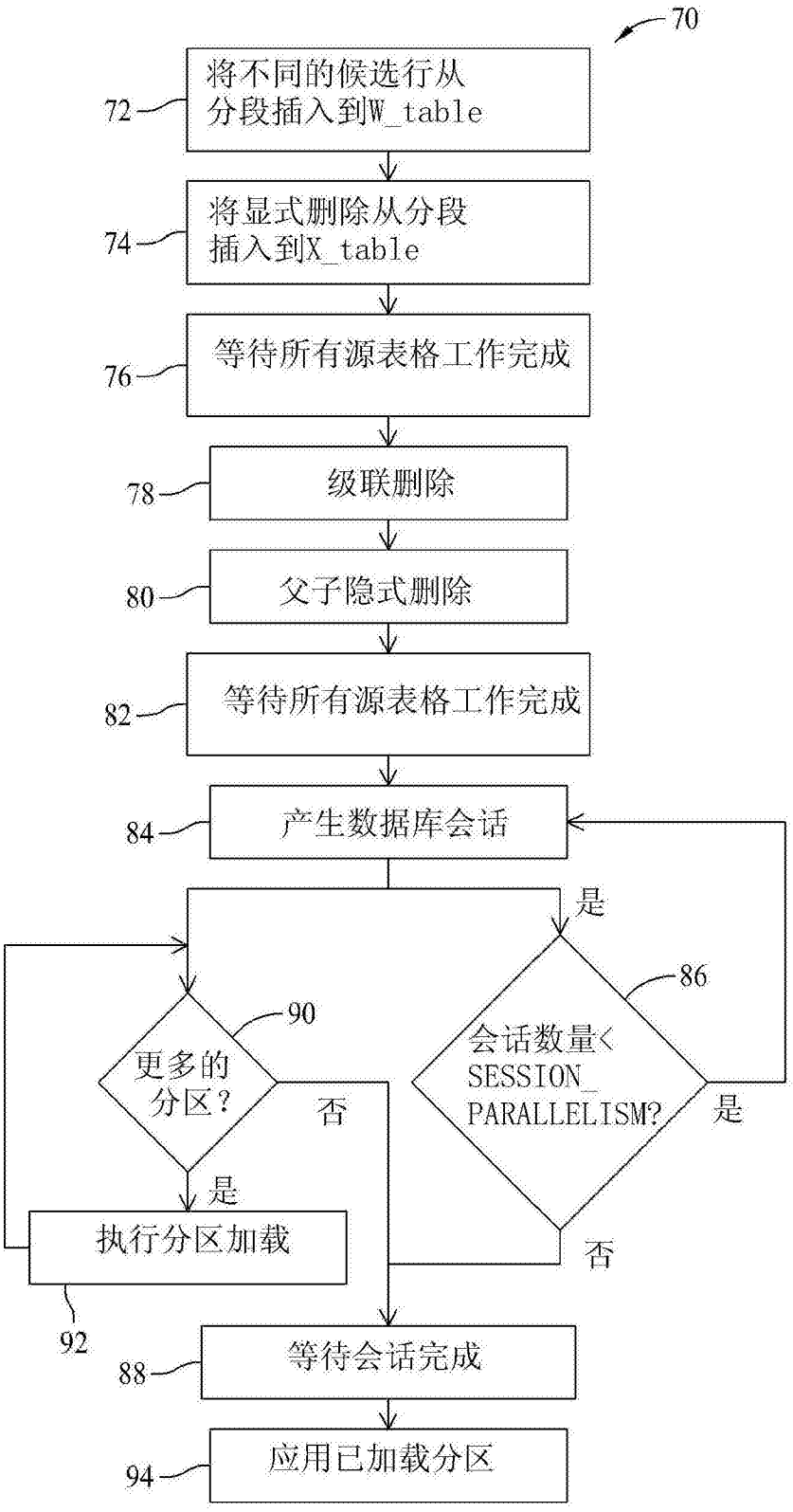

70
72
将不同的候选行从分段插入到W_table
74
将显式删除从分段插入到X_table
76
等待所有源表格工作完成
78
级联删除
80
父子隐式删除
82
等待所有源表格工作完成
84
产生数据库会话
是
86
会话数量< SESSION_ PARALLELISM?
是
否
90
更多的分区?
否
是
执行分区加载
92
88
等待会话完成
94
应用已加载分区

图3

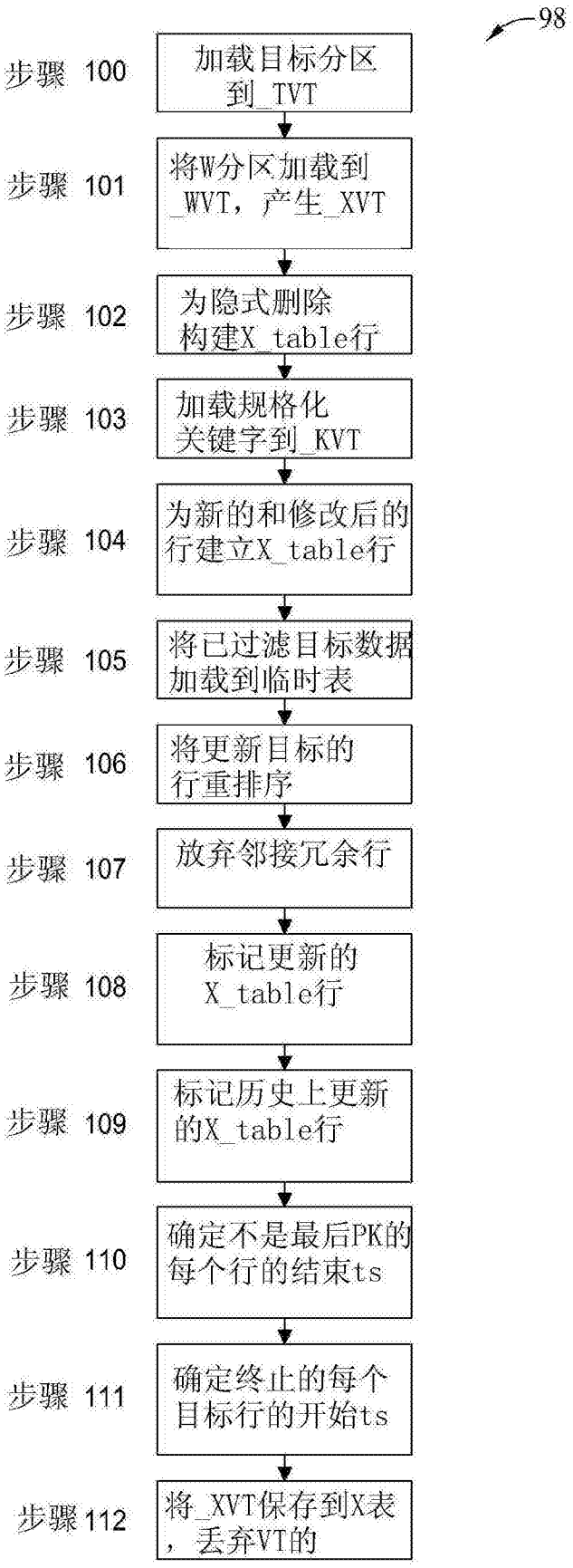

98
步骤 100
加载目标分区到_TVT
步骤 101
将W分区加载到_WVT，产生_XVT
步骤 102
为隐式删除构建X_table行
步骤 103
加载规格化关键字到_KVT
步骤 104
为新的和修改后的行建立X_table行
步骤 105
将已过滤目标数据加载到临时表
步骤 106
将更新目标的行重排序
步骤 107
放弃邻接冗余行
步骤 108
标记更新的X_table行
步骤 109
标记历史上更新的X_table行
步骤 110
确定不是最后PK的每个行的结束ts
步骤 111
确定终止的每个目标行的开始ts
步骤 112
将_XVT保存到X表，丢弃VT的

图4

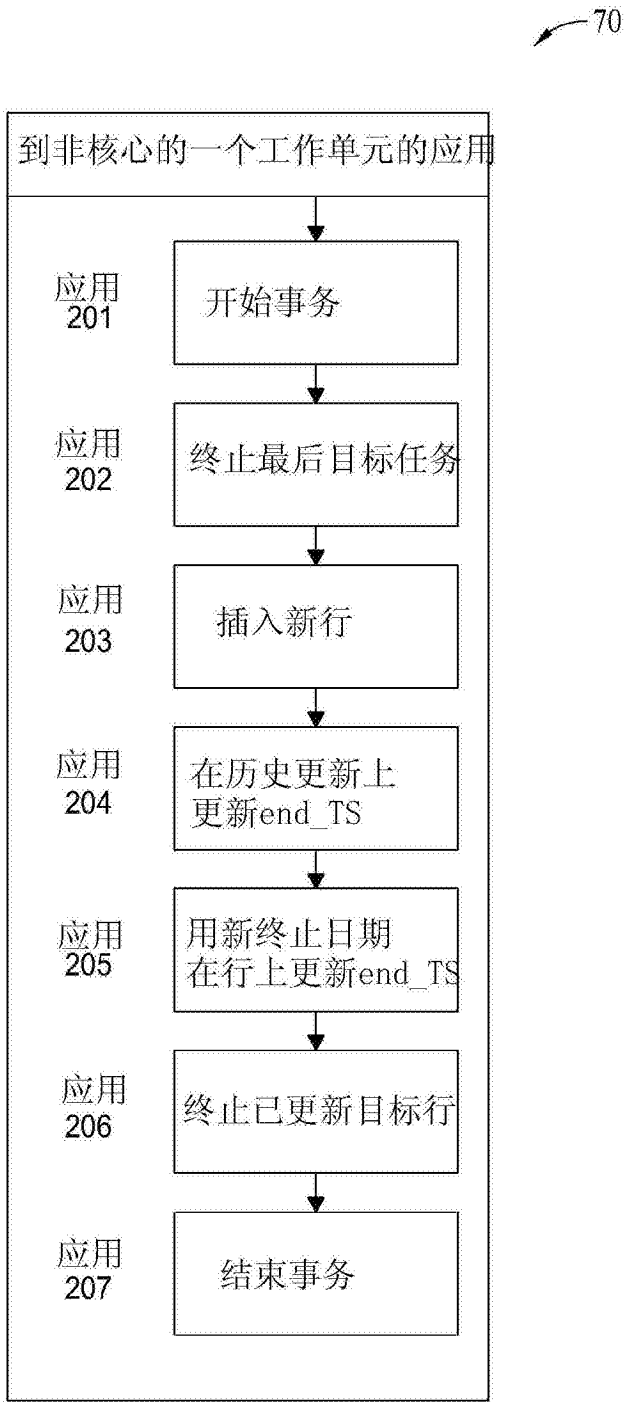

70
到非核心的一个工作单元的应用
应用 201
开始事务
应用 202
终止最后目标任务
应用 203
插入新行
应用 204
在历史更新上 更新end_TS
应用 205
用新终止日期 在行上更新end_TS
应用 206
终止已更新目标行
应用 207
结束事务

图5

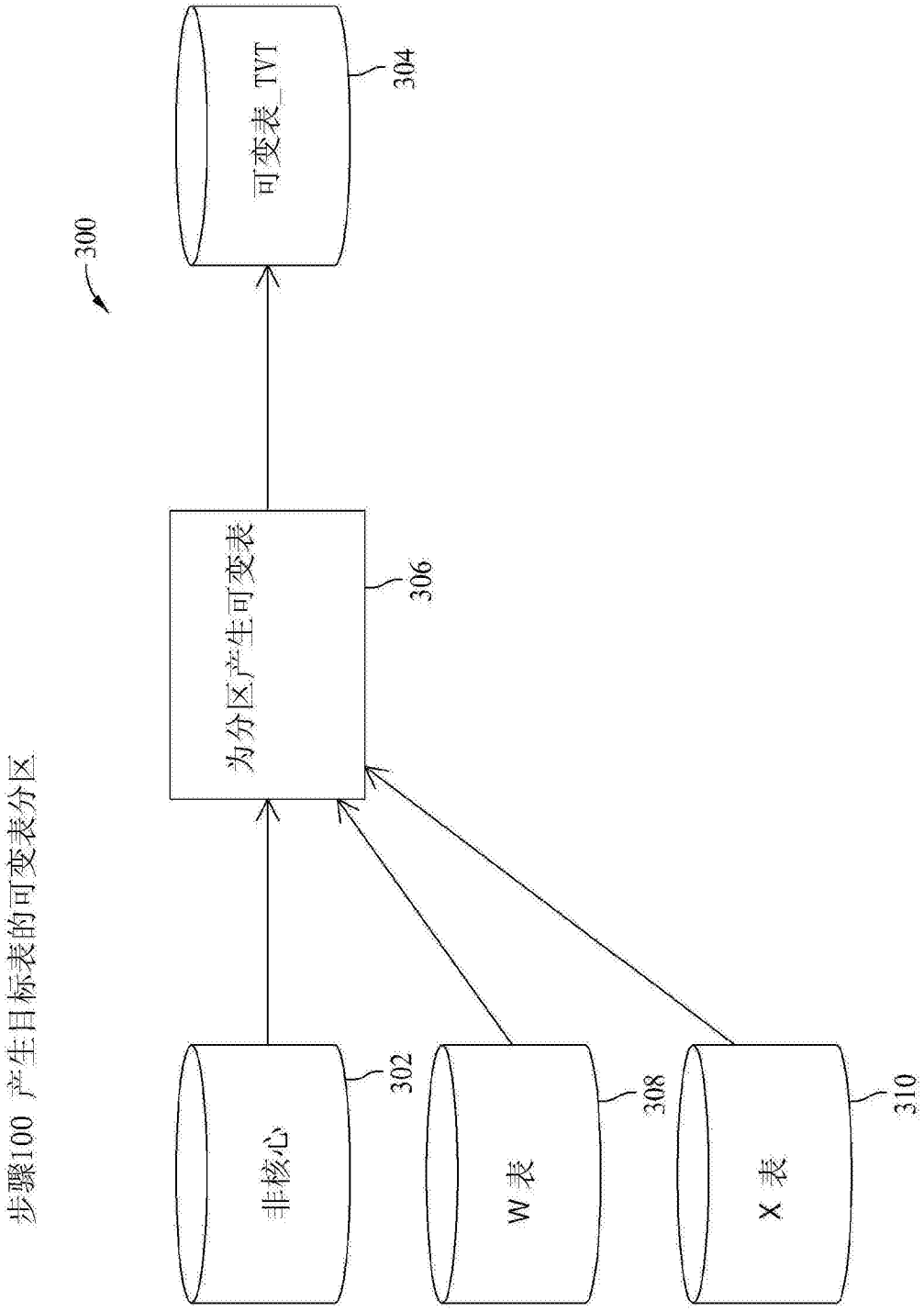

步骤100 产生目标表的可变表分区
300
可变表_TVT
304
为分区产生可变表
306
非核心
302
W 表
308
X 表
310

图6

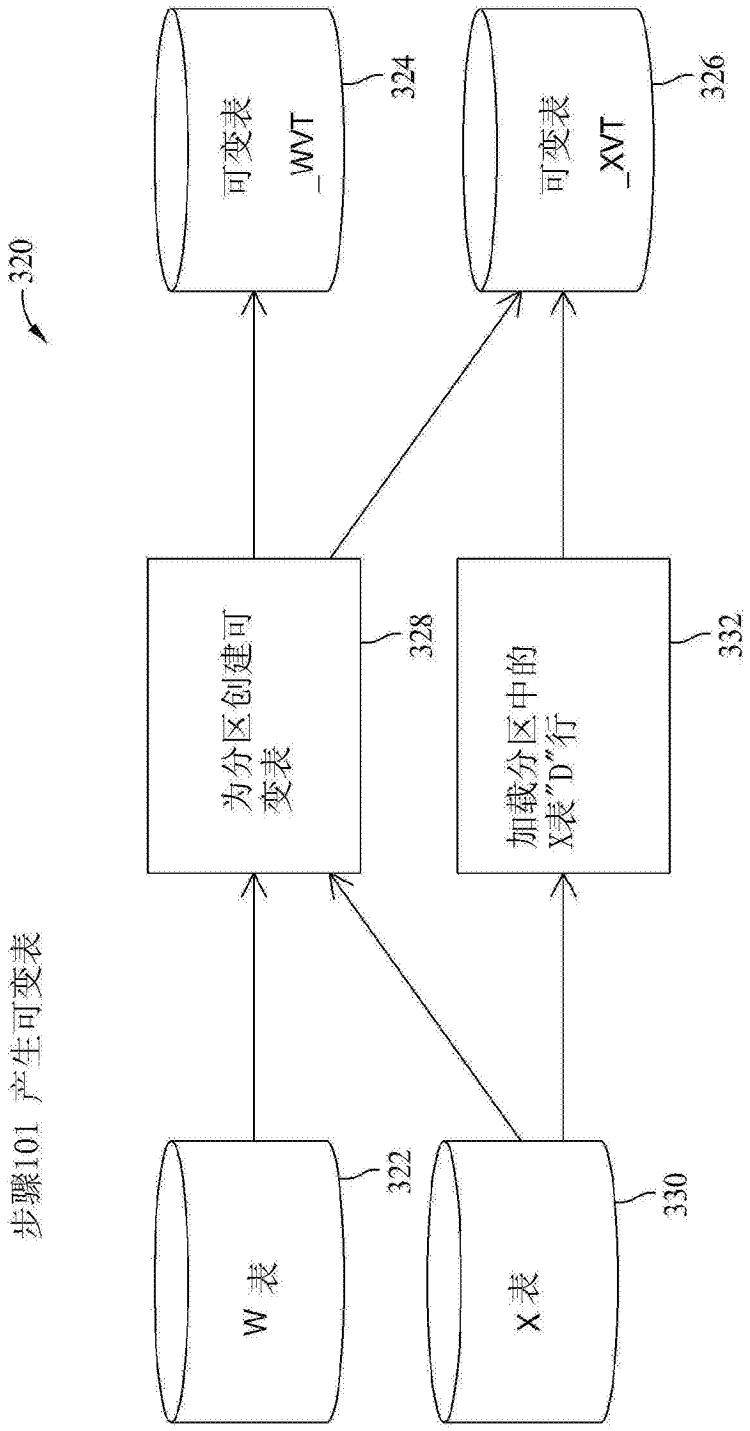

320
步骤101 产生可变表
W 表
322
X 表
330
为分区创建可变表
328
加载分区中的X表"D"行
332
可变表_WVT
324
可变表_XVT
326

图7

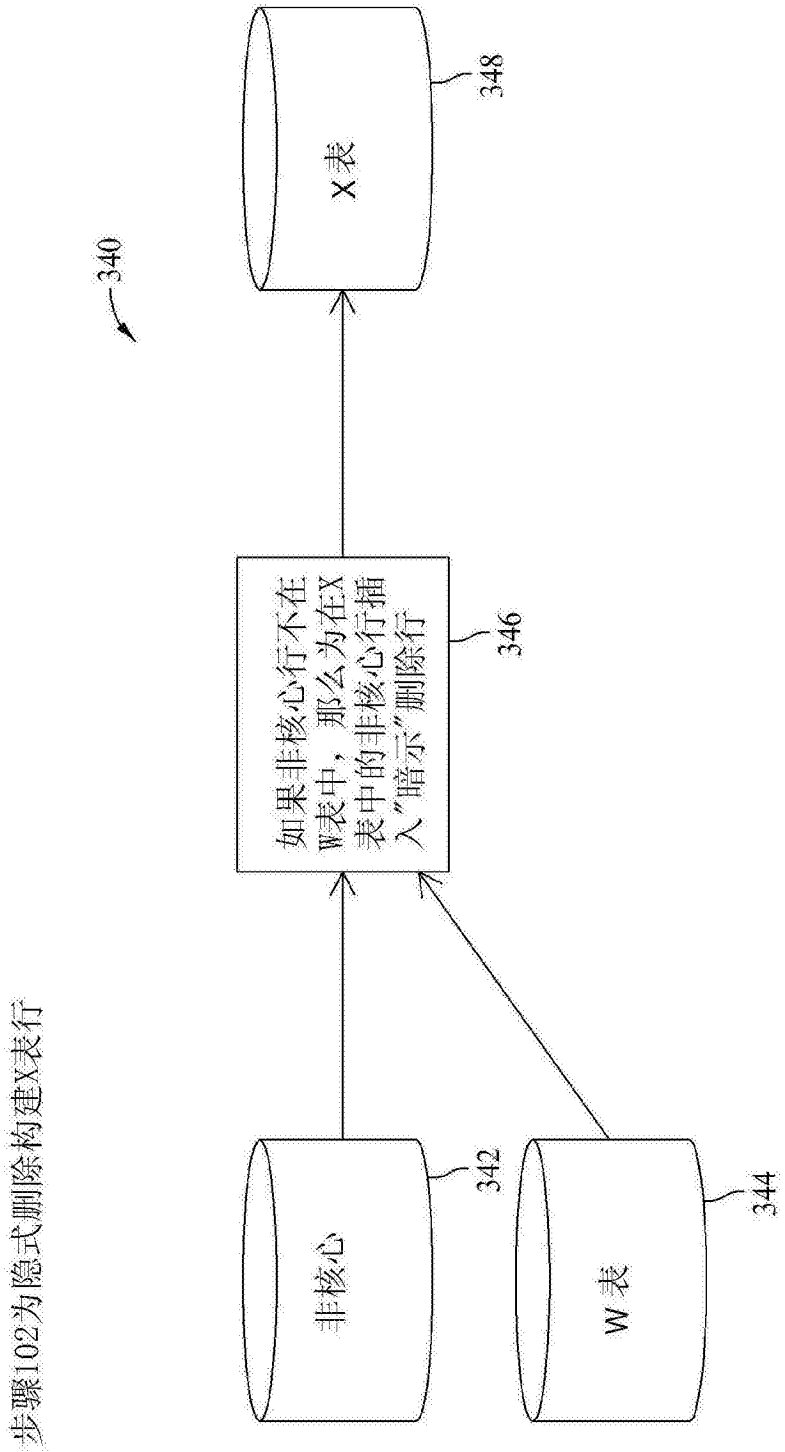

步骤102为隐式删除构建X表行
340
非核心
342
W 表
344
如果非核心行不在W表中，那么为在X表中的非核心行插入"暗示"删除行
346
X表
348

图8

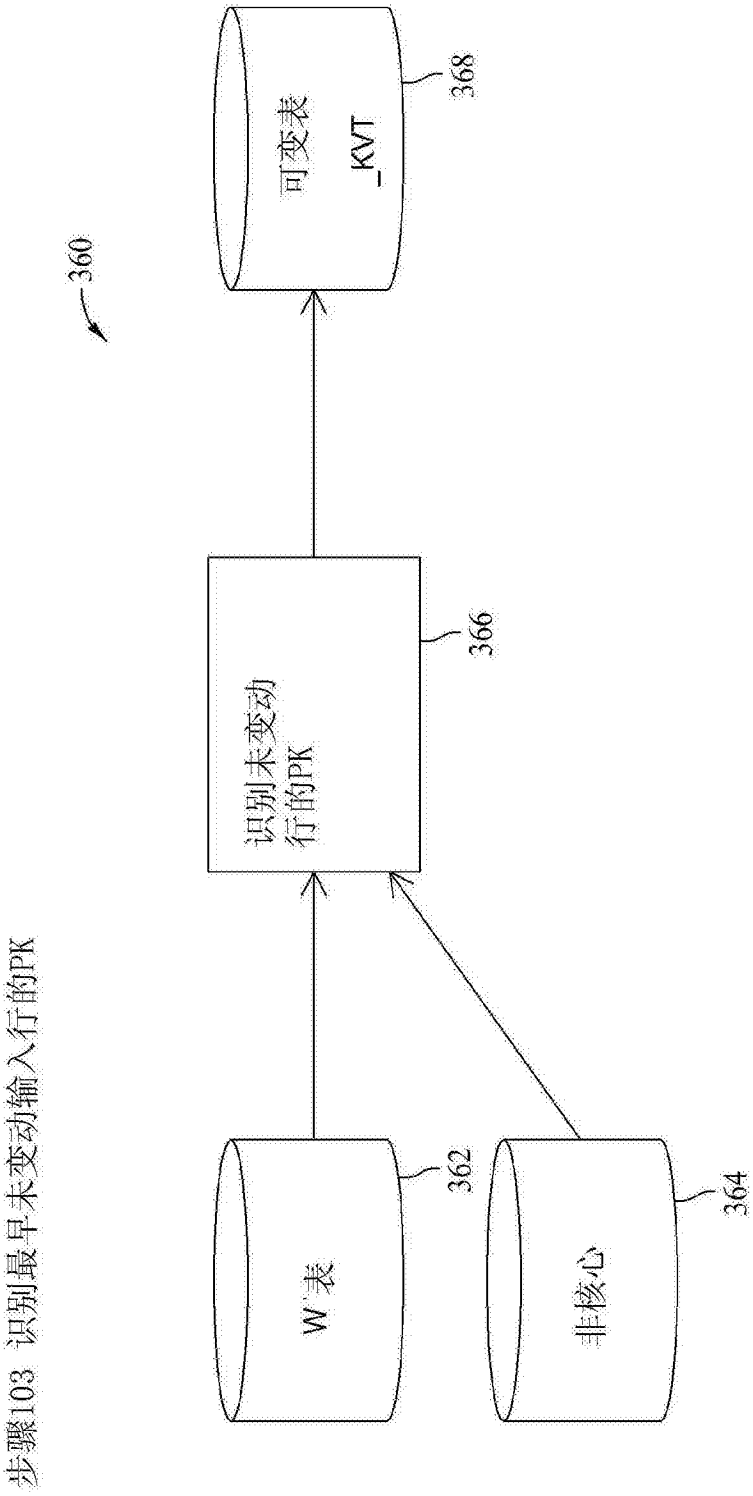

步骤103 识别最早未变动输入行的PK
360
W 表
362
非核心
364
识别未变动
行的PK
366
可变表
_KVT
368

图9

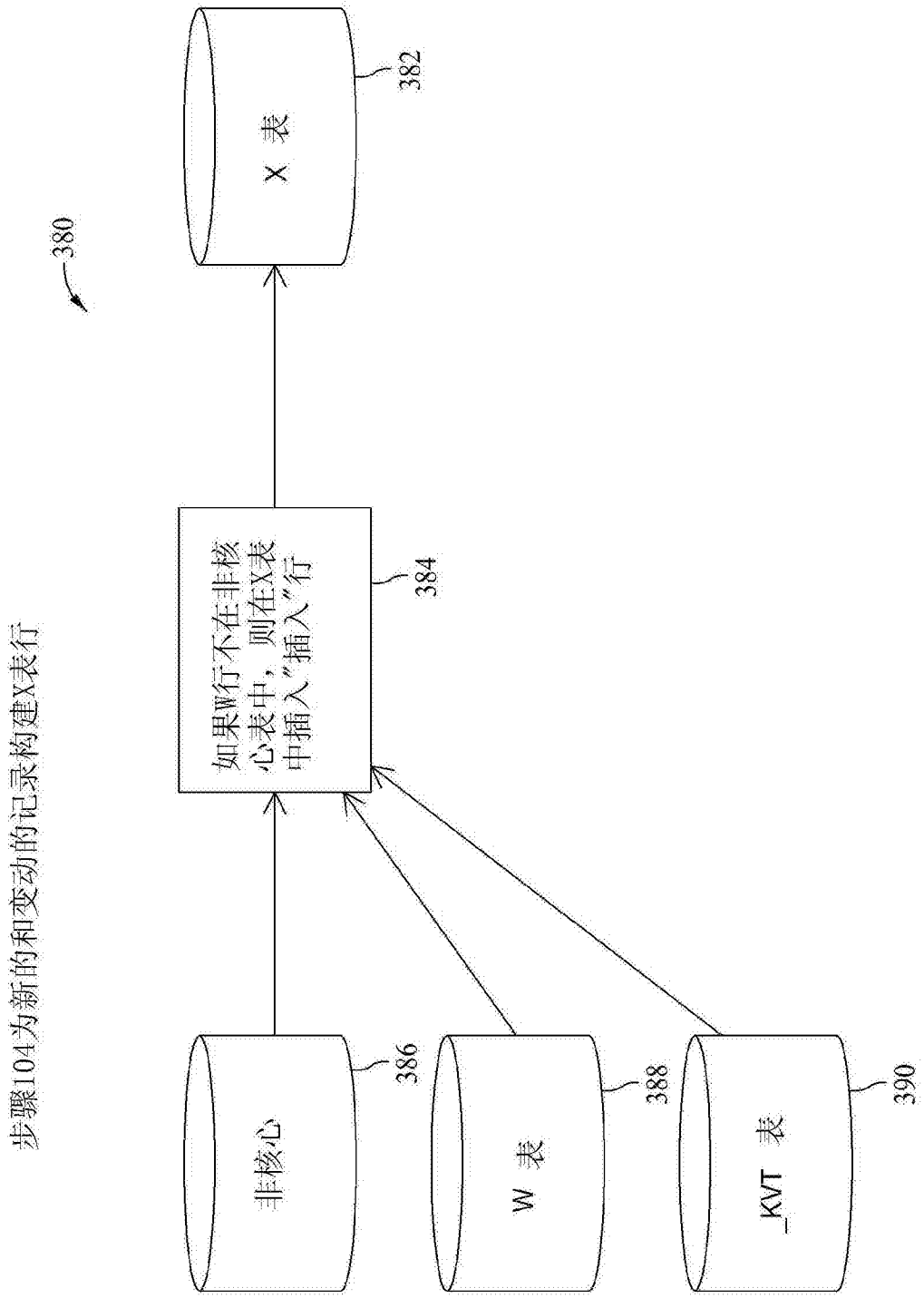

步骤104为新的和变动的记录构建X表行
380
X 表
382
如果W行不在非核心表中，则在X表中插入”插入”行
384
非核心
386
W 表
388
_KVT 表
390

图10

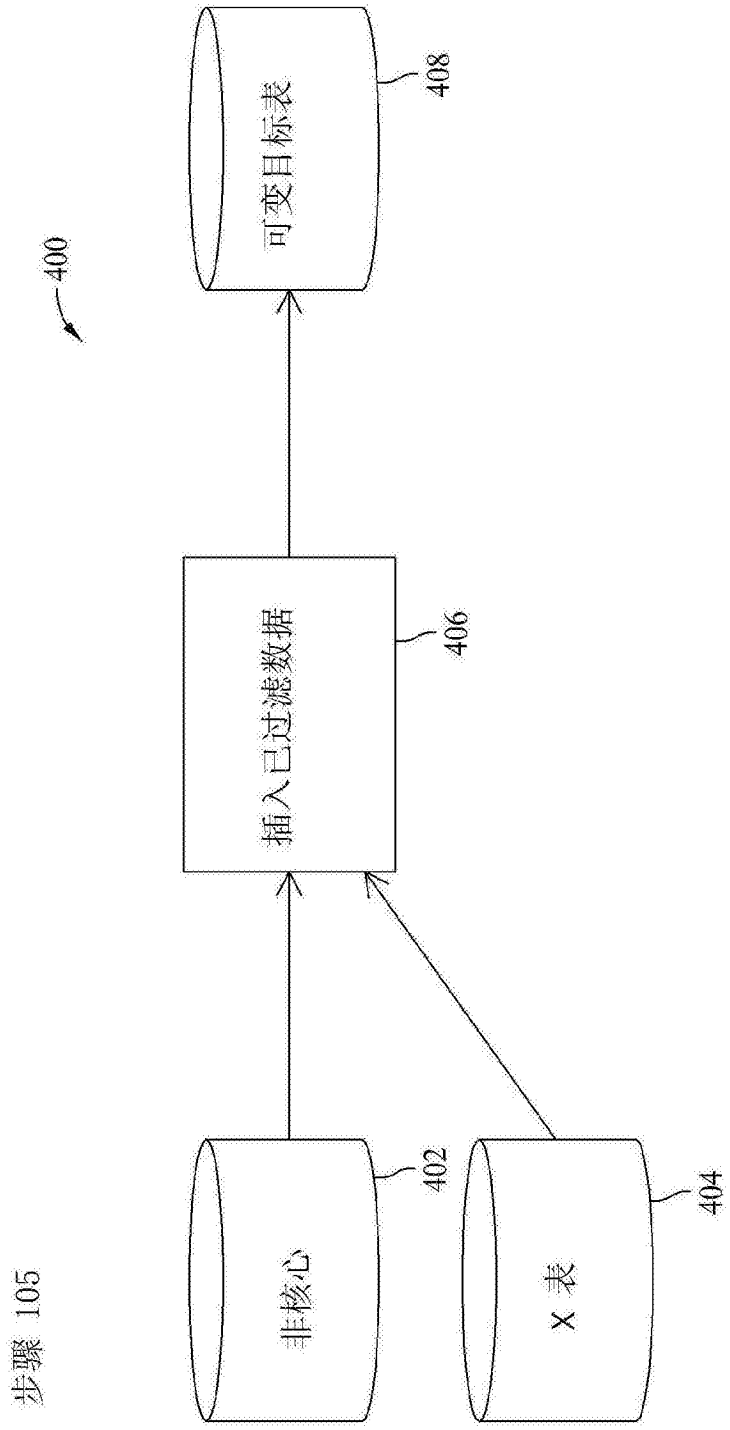


图11

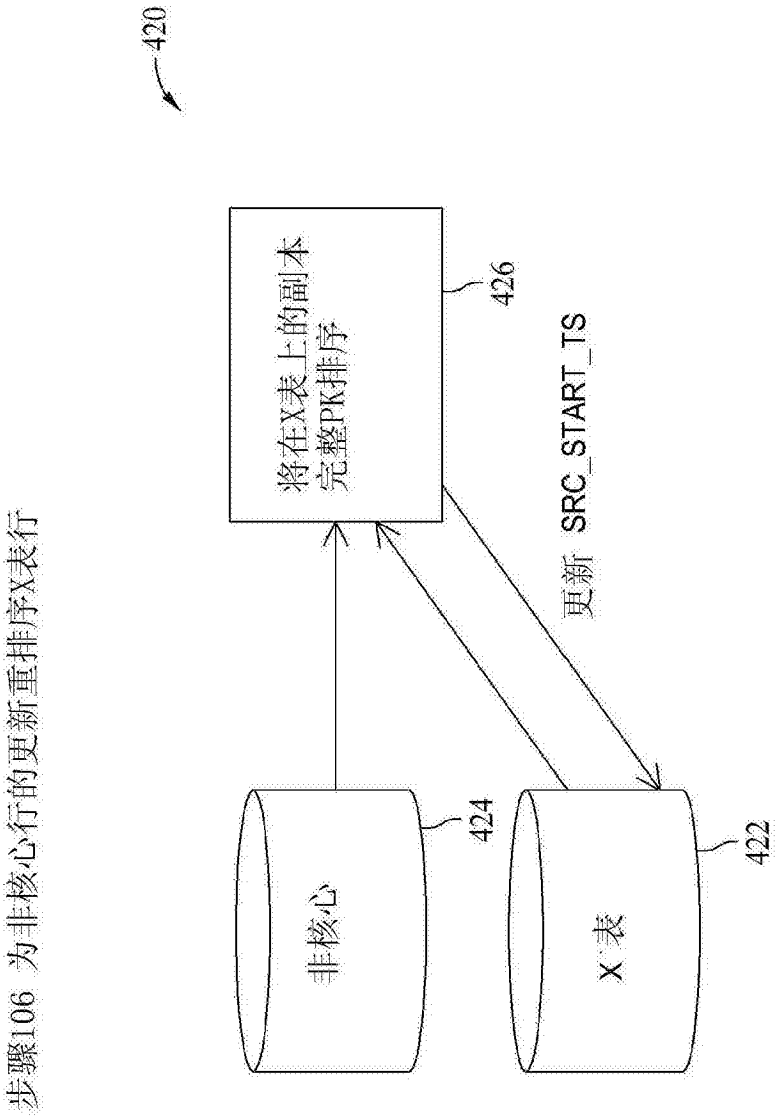

420
步骤106 为非核心行的更新重排序X表行
将在X表上的副本
完整PK排序
426
更新 SRC_START_TS
非核心
424
X 表
422

图12

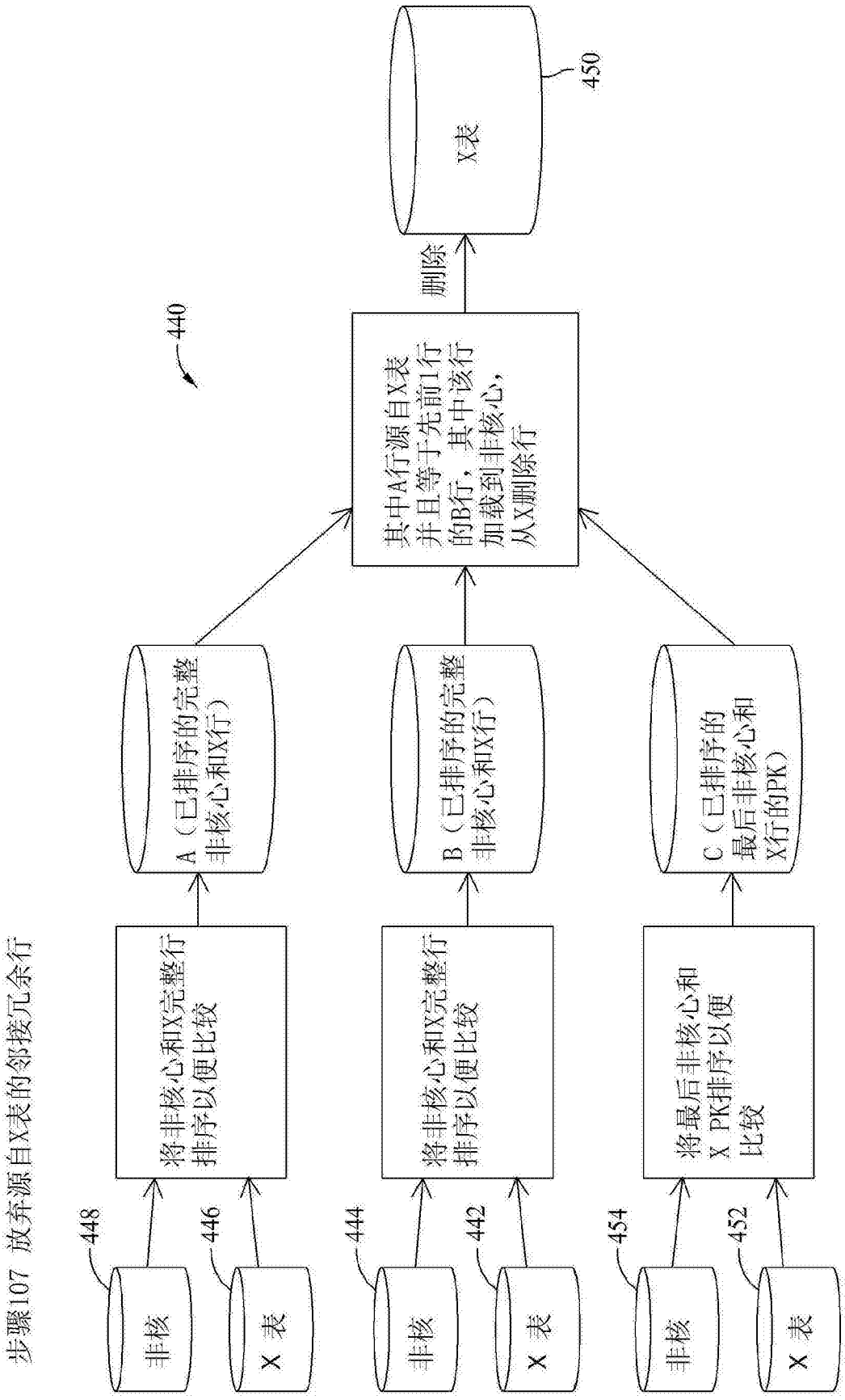

步骤107 放弃源自X表的邻接冗余行
非核
X 表
448
446
将非核心和X完整行排序以便比较
A（已排序的完整非核心和X行）
非核
X 表
444
442
将非核心和X完整行排序以便比较
B（已排序的完整非核心和X行）
非核
X 表
454
452
将最后非核心和X PK排序以便比较
C（已排序的最后非核心和X行的PK）
其中A行源自X表并且等于先前1行的B行，其中该行加载到非核心，从X删除行
440
删除
X表
450

图13

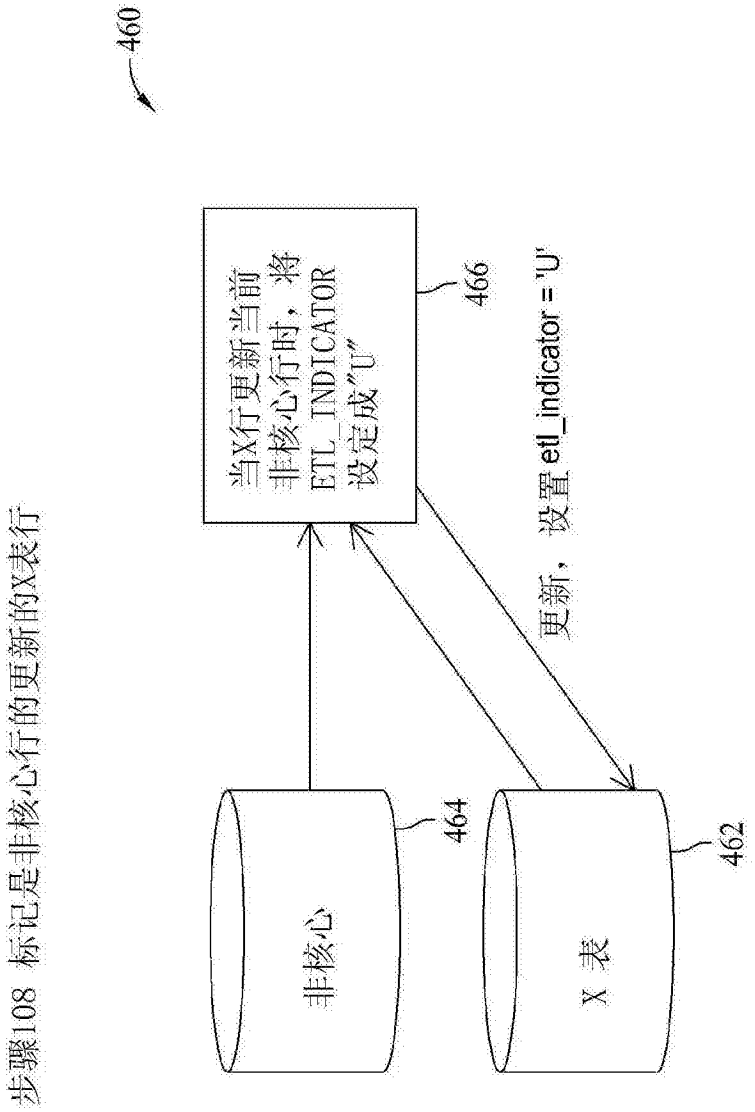

460
步骤108 标记是非核心行的更新的X表行
非核心
464
X 表
462
更新，设置 etl_indicator = 'U'
当X行更新当前
非核心行时，将
ETL_INDICATOR
设定成"U"
466

图14

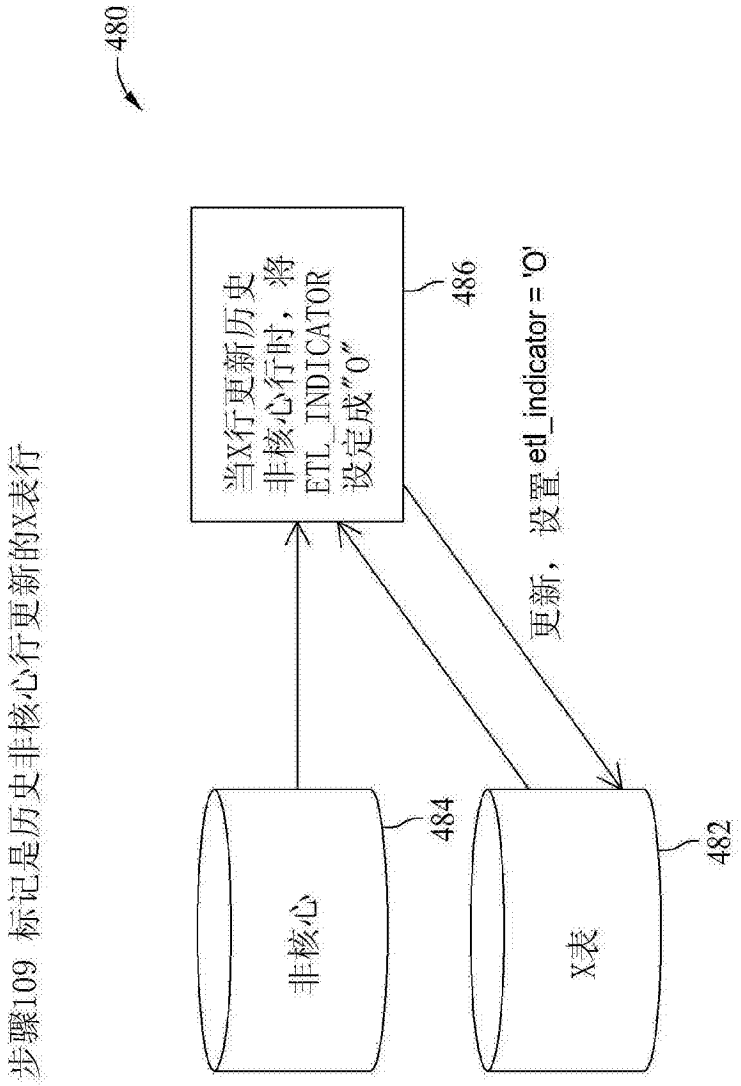

480
步骤109 标记是历史非核心行更新的X表行
当X行更新历史非核心行时，将ETL_INDICATOR设定成"0"
486
更新，设置etl_indicator = 'O'
非核心
484
X表
482

图15

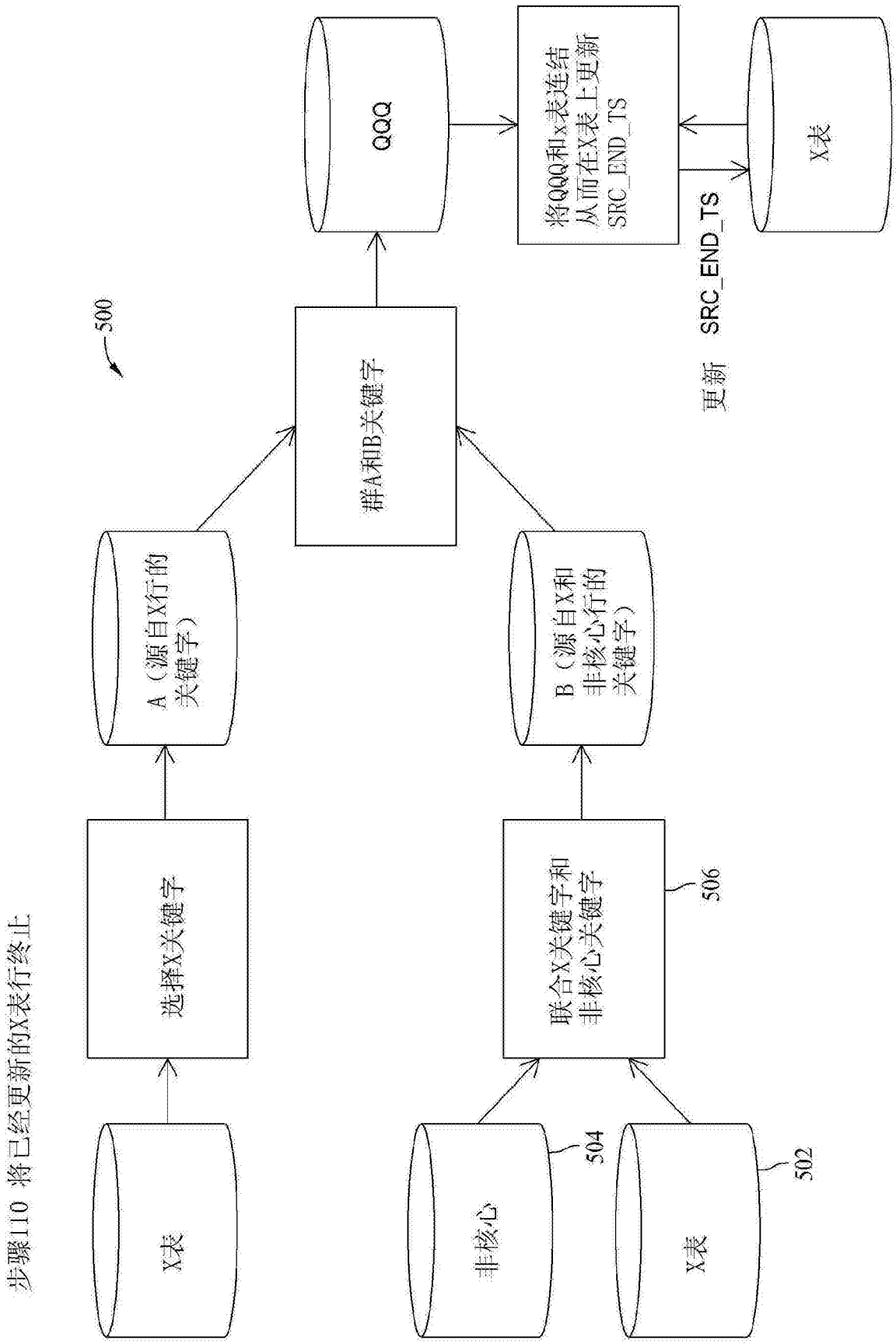

步骤110 将已经更新的X表行终止
X表
选择X关键字
A（源自X行的关键字）
非核心
504
X表
502
联合X关键字和非核心关键字
506
B（源自X和非核心行的关键字）
群A和B关键字
500
QQQ
将QQQ和x表连结从而在X表上更新SRC_END_TS
更新 SRC_END_TS
X表

图16

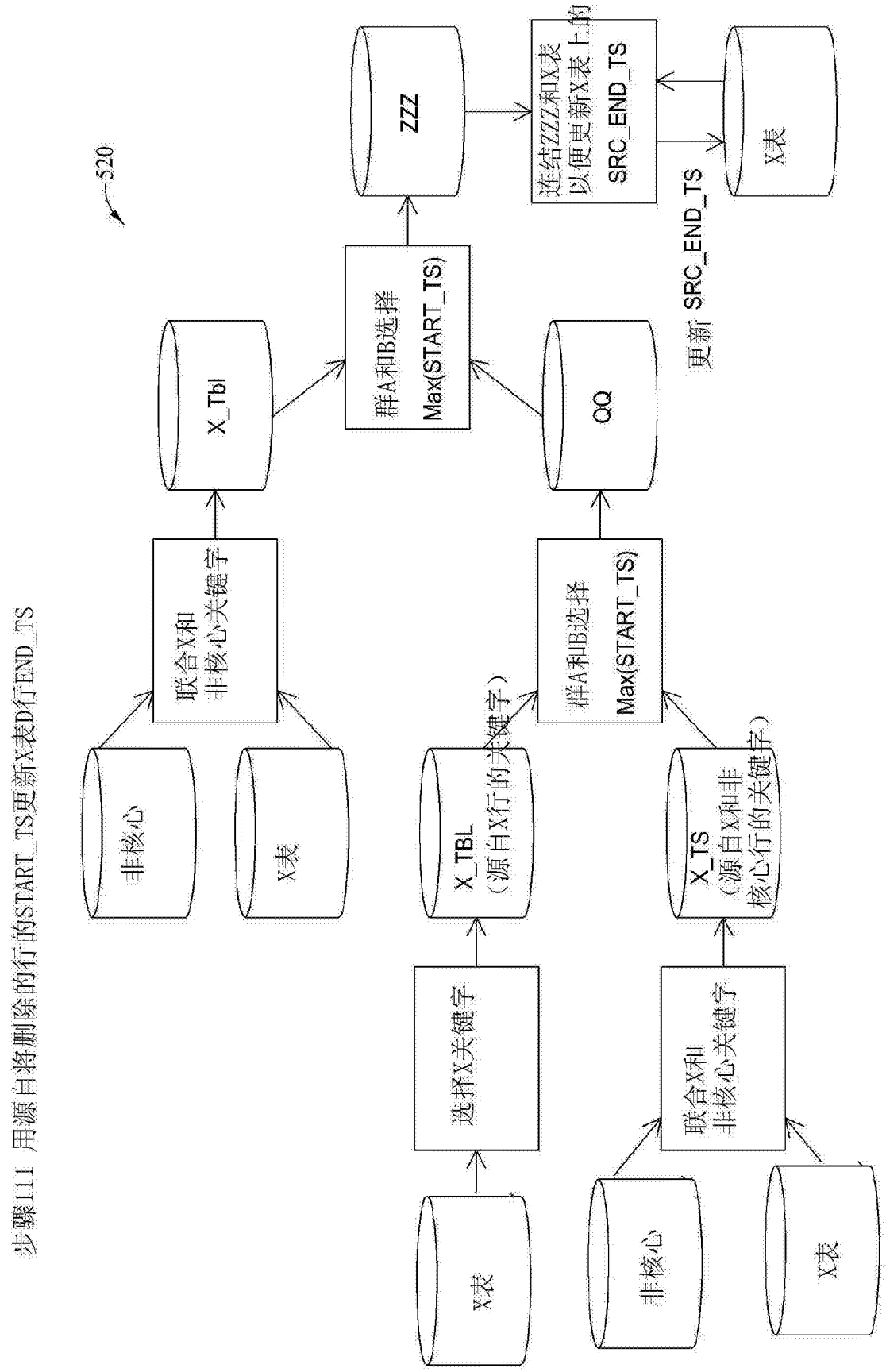

520
步骤111 用源自将删除的行的START_TS更新X表D行END_TS
非核心
X表
联合X和
非核心关键字
X_Tbl
群A和B选择
Max(START_TS)
ZZZ
连结ZZZ和X表
以便更新X表上的
SRC_END_TS
X表
更新 SRC_END_TS
X表
选择X关键字
X_TBL
（源自X行的关键字）
群A和B选择
Max(START_TS)
QQ
非核心
联合X和
非核心关键字
X_TS
（源自X和非
核心行的关键字）
X表

图17

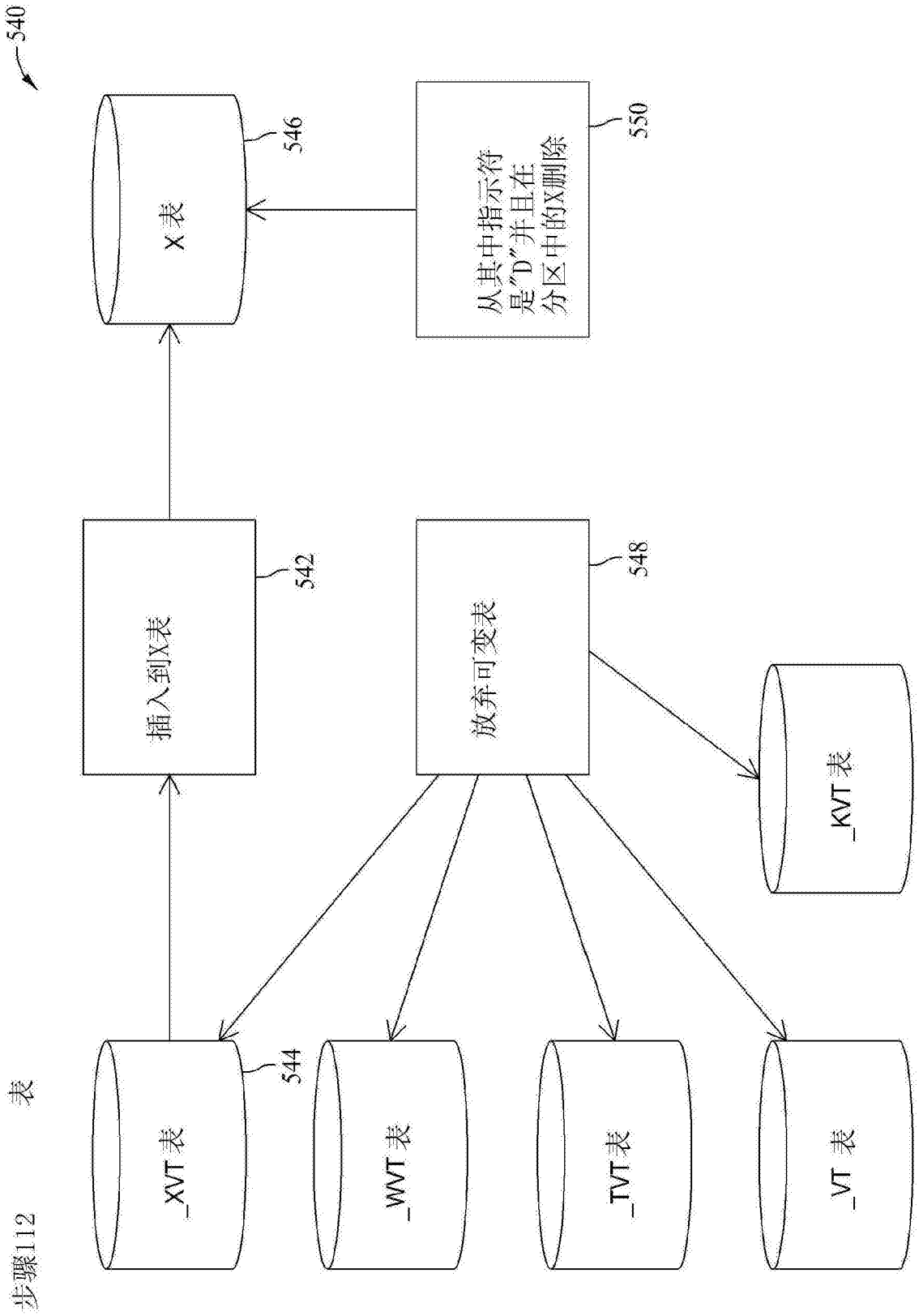

540
X 表
546
从其中指示符
是"D"并且在
分区中的X删除
550
插入到X表
542
放弃可变表
548
_KVT 表
步骤112 表
_XVT 表
544
_WVT 表
_TVT 表
_VT 表

图18

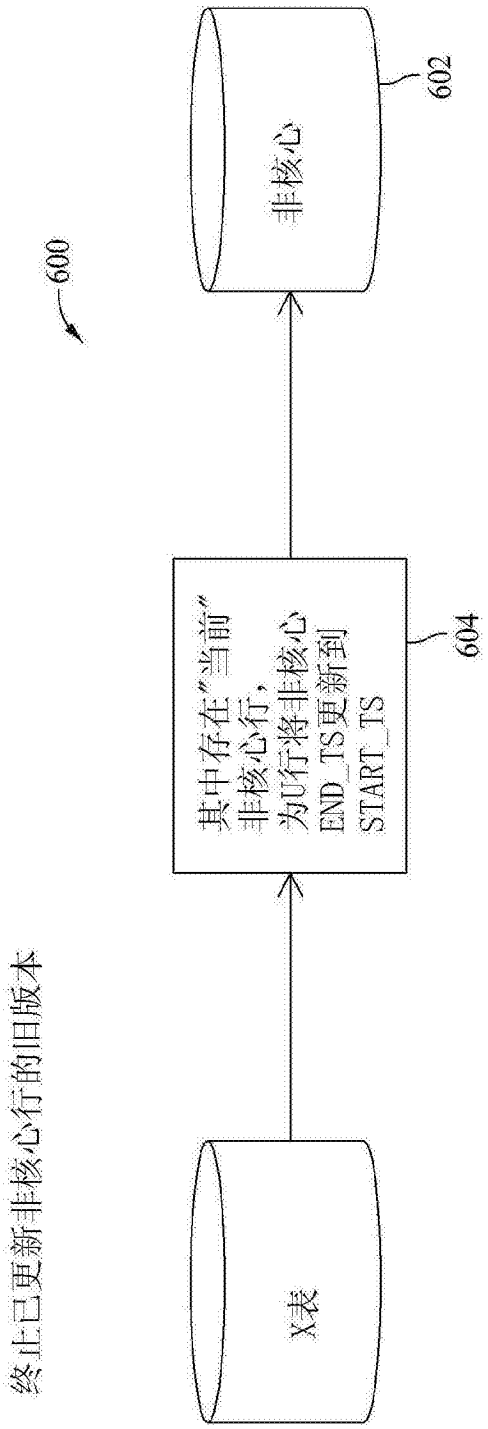

600
非核心
602
其中存在"当前"非核心行，为U行将非核心END_TS更新到START_TS
604
X表
终止已更新非核心行的旧版本

图19

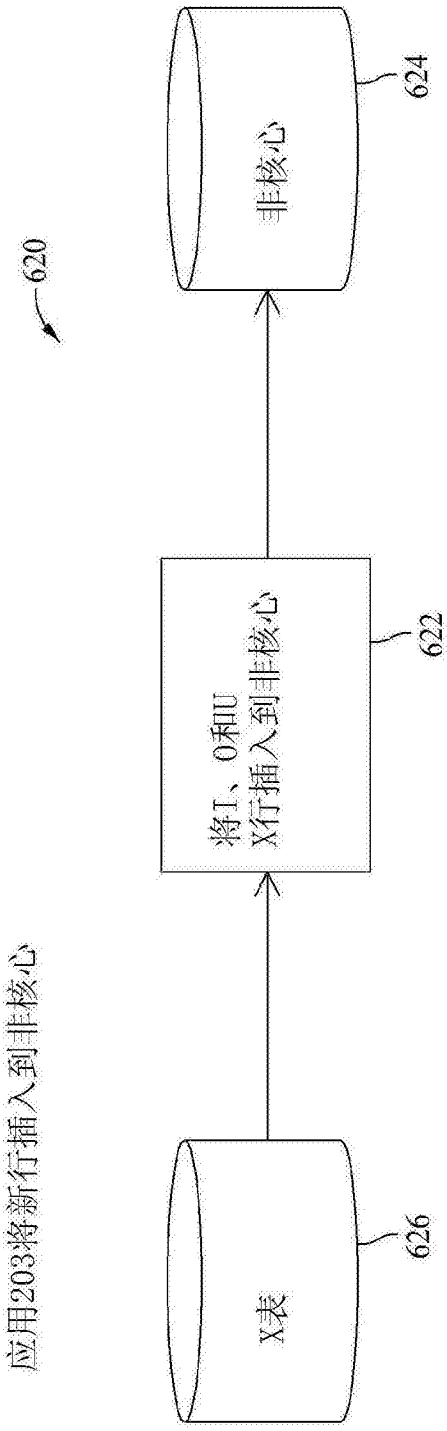

620
非核心
624
将I、0和U X行插入到非核心
622
X表
626
应用203将新行插入到非核心

图20

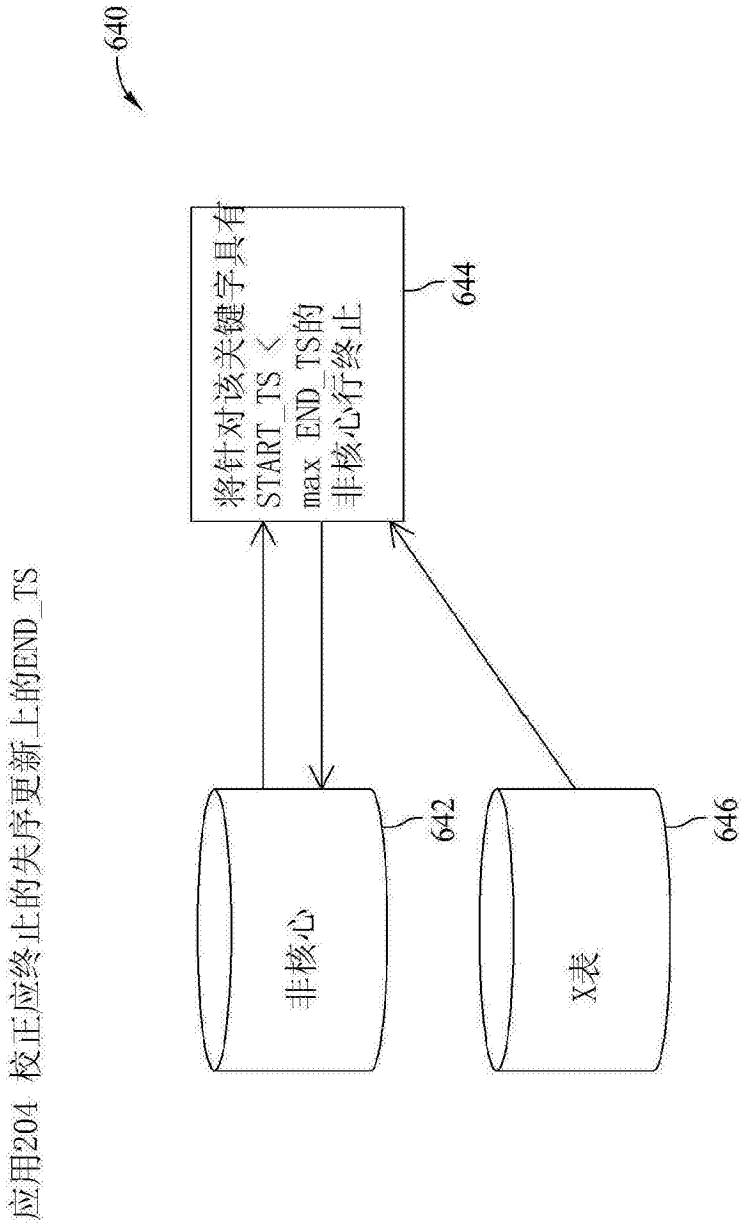

640
应用204 校正应终止的失序更新上的END_TS
将针对该关键字具有START_TS <max END_TS的非核心行终止
644
非核心
642
X表
646

图21

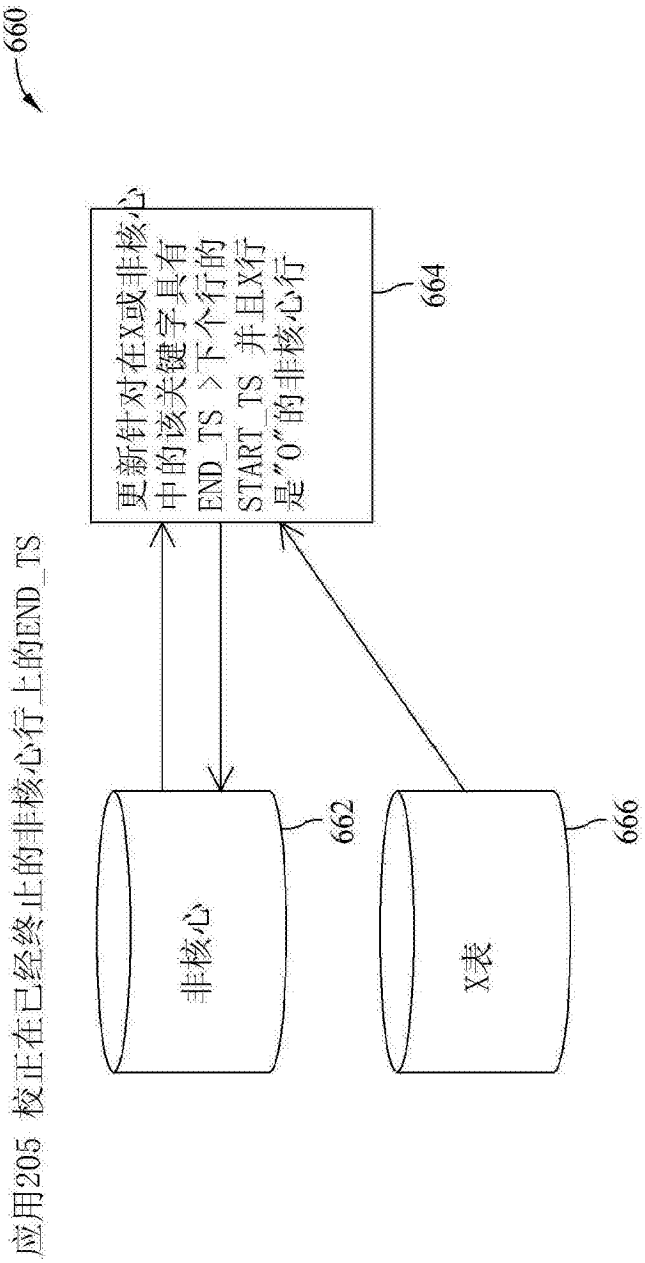

660
应用205 校正在已经终止的非核心行上的END_TS
更新针对在X或非核心中的该关键字具有END_TS >下个行的START_TS 并且X行是"0"的非核心行
664
非核心
662
X表
666

图22

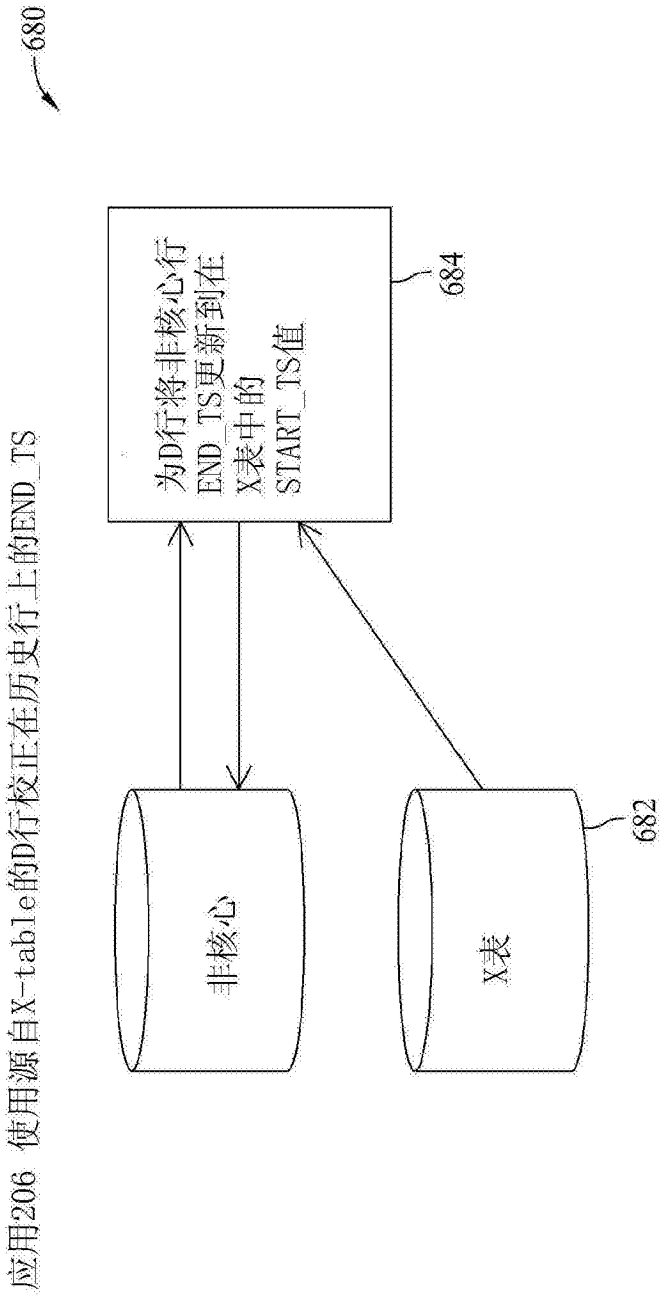

680
应用206 使用源自X-table的D行校正在历史行上的END_TS
为D行将非核心行
END_TS更新到在
X表中的
START_TS值
684
非核心
X表
682

图23

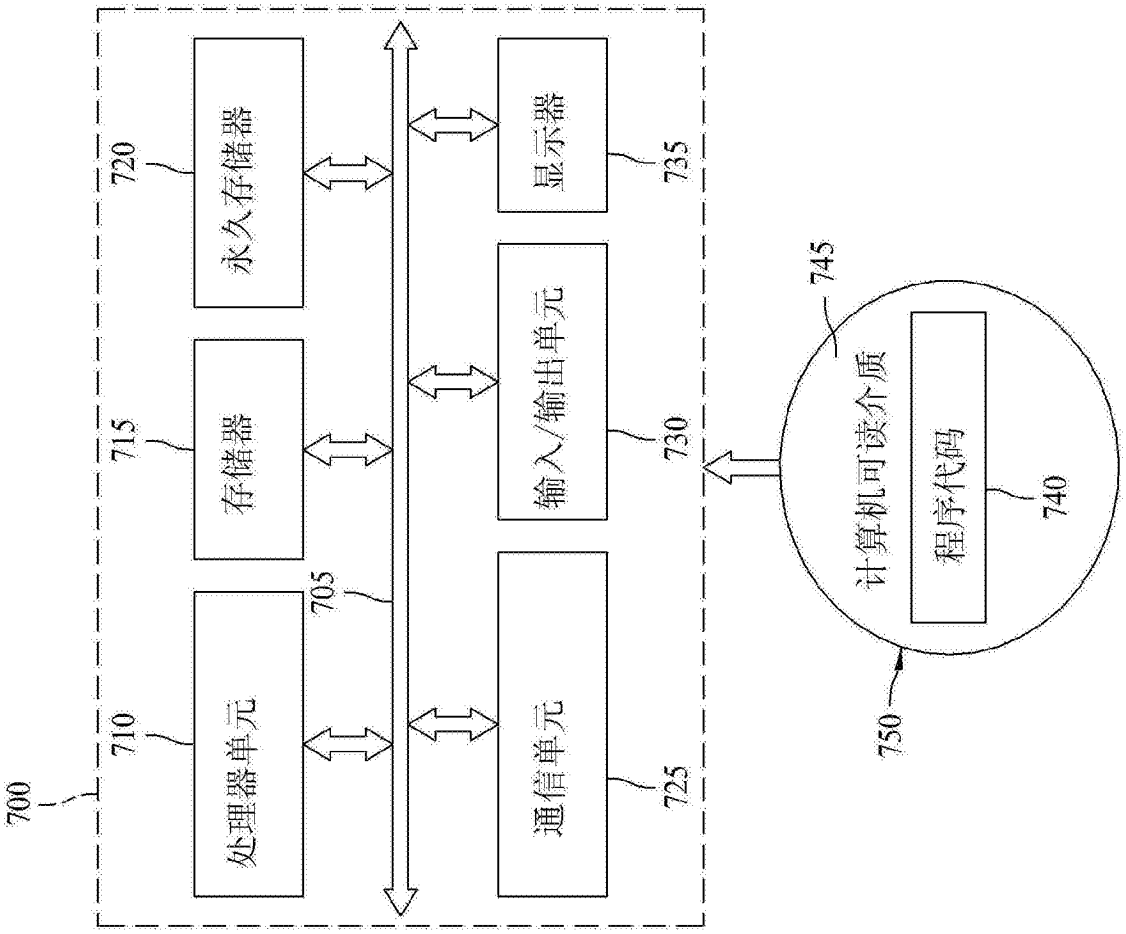

700
710
处理器单元
715
存储器
720
永久存储器
705
725
通信单元
730
输入/输出单元
735
显示器
745
计算机可读介质
740
程序代码
750

图24