(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 103168300 B
(45)授权公告日 2016.06.01

(21)申请号 201180038929.8
(22)申请日 2011.08.04
(30)优先权数据
61/371,413 2010.08.06 US
12/973,811 2010.12.20 US
(85)PCT国际申请进入国家阶段日
2013.02.06
(86)PCT国际申请的申请数据
PCT/US2011/046600 2011.08.04
(87)PCT国际申请的公布数据
WO2012/019008 EN 2012.02.09
(73)专利权人 移动解决方案公司
地址 美国加利福尼亚
(72)发明人 阿尼什瓦拉·尼卡
(74)专利代理机构 北京市柳沈律师事务所 11105
代理人 邵亚丽
(51)Int.Cl.
G06F 17/30(2006.01)
(56)对比文件
US 6882993 B1,2005.04.19,
US 2010/0161555 A1,2010.06.24,
CN 1763744 A,2006.04.26,
CN 101127045 A,2008.02.20,
CN 101241503 A,2008.08.13,
审查员 戴诚

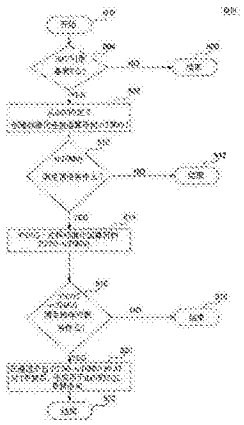
权利要求书1页 说明书27页 附图9页

(54)发明名称

具有外连接的立即实体化视图的增量维护

(57)摘要

本发明涉及在具有外连接的实体化视图的增量维护的关系数据库管理系统中的使用算法的方法和系统。针对实体化外连接视图的类别以及更新操作的性能，该算法实现了下面的目标：将对视图选择列表中的主关键字属性的存在的要求放松至仅关系中的一些(也就是，关系引用为外连接中的保存侧)；放松对于视图定义中使用的一些谓语的空-不兼容性能需求(也就是，谓语引用关系可以由多于一个的外连接的空-导入)；以及对于视图中引用的每个关系，通过对每个视图使用一个更新语句(例如，MERGE,UPDATE,INSERT或DELETE)从而实施外连接视图的维护。该算法允许具有外连接的实体化视图的增量维护的设计和实施被结合入RDBMS。



CN 103168300 B

1.用于创建立即实体化外连接视图的内部表达的方法,其包括:

确定先前定义的立即实体化视图iMV是否引用数据库中的基表;

响应iMV引用基表的确定,建立标准化连接运算符树,其中所述树是基于iMV的定义;

确定所述树是否满足谓语条件;

响应所述树满足谓语条件的确定,执行所述树的保存侧/空-导入侧PSNS注释从而创建被注释的树;

确定所述被注释的树是否满足iMV的选择列表条件;

响应被注释的树满足所述选择列表条件的确定,在来自所述被注释的树的基表更新后,生成用于iMV的更新语句。

2.如权利要求1所述的方法,其中所述数据库是关系数据库。

3.如权利要求2所述的方法,进一步包括:

使用所述被注释的树形成iMV的内部表示;

使所述内部表示与关系数据库管理系统RDBMS的查询优化器整合;以及

其中所述生成进一步包括:

精确地生成用于iMV的一个结构化查询语言SQL的UPDATE语句,从而响应检测到基表被更新而被使用,其中,所述UPDATE语句通过下述内容而放松了由所述iMV定义施加的限制,所述下述内容为:

不要求用于更新非-空-导入元组和空-导入元组的中间临时表;以及

同时计算用于更新所述iMV的非-空-导入元组到空-导入元组。

4.如权利要求2所述的方法,其中所述更新语句是结构化的查询语言SQL语句。

5.如权利要求2所述的方法,其中所述更新语句是MERGE,INSERT,UPDATE或DELETE语句。

6.如权利要求1所述的方法,其中所述iMV的选择列表条件不要求主关键字。

7.如权利要求6所述的方法,其中所述iMV的选择列表条件不要求用于在所述外连接视图中引用的每个关系的主关键字。

8.如权利要求7所述的方法,其中所述iMV的选择列表条件包括具有主关键字的关系的子集。

9.如权利要求8所述的方法,其中所述iMV的选择列表条件包括必须具有主关键字的关系的子集。

10.如权利要求9所述的方法,其中所述关系的子集包括在外连接中引用为保存侧的关系。

11.如权利要求1所述的方法,其中所述基表的更新是基于基表的计算增量。

12.如权利要求11所述的方法,其中基表的增量是基于基表的前映像和基表的后映像的比较而计算。

具有外连接的立即实体化视图的增量维护

[0001] 发明背景

技术领域

[0002] 本发明一般涉及数据库管理系统，以及，更具体地，涉及提供实体化视图的数据库管理系统，实体化视图的增量维护以及特别地涉及含有外连接的实体化视图的维护的系统和方法。

背景技术

[0003] SYBASE™ SQL ANYWHERE™是一种ANSI SQL顺应式的关系数据库管理系统(RDBMS)，其设计成在各种平台上运行，所述平台为从服务器类安装到使用Windows移动操作系统的移动装珞。SQLANYWHERE™是一种自我管理RDBMS，其具有高可靠性、高性能、同步功能、小封装、以及横跨各种32位和64位平台的整个范围的SQL特性。

[0004] 目前可用的产品，比如SQL ANYWHERE™，支持手动实体化视图。只能通过全部重新计算被刷新的手动实体化视图，可由任何复杂的查询定义。然而，一些查询优化器，如SQL ANYWHERE™优化器，在查询优化期间能够仅使用在视图匹配过程中的某类别的实体化视图。虽然SQL ANYWHERE™支持用于群-选择-项目-连接(GROUP-SELECT-PROJECT-JOIN)视图的某实体化视图的增量维护，需要的是更普通类别的可被增量地维护的实体化视图。因此，需要的是用于立即实体化视图(iMVs)的增量维护的方法和系统。进一步需要的是一个RDBMS，其支持扩展类别的立即实体化视图，即具有集合或没有集合的外连接视图。在本发明的一个实施例中，所描述的算法可以在数据库或数据仓库系统中实施，例如但不限于SQLANYWHERE™。

[0005] 外连接查询越来越频繁地用于新系统和外部工具，其中，数据库管理员(DBAs)或经验丰富的数据库开发人员不能直接调整生成的SQL语句。图1示出了这样生成的查询的例子。例如，对于使用外连接视图从语义转换至视图匹配的处理外连接查询，SQLANYWHERE™优化器具有复杂的技术。然后，有必要扩展对具有外连接的实体化视图的增量维护的支持，这可以加快很多使用SQLANYWHERE™ RDBMS的应用。对具有外连接的立即实体化视图的有效支持的目标是多方面的。

[0006] 用于具有外连接的实体化视图的增量维护的一些传统的技术是基于连接析取范式表达。一个连接析取范式编码一个外连接查询作为不同连接的最小联合的一序列。图3中的例子显示了与表1中定义的查询V_1的连接析取范式对应的亲子关系图。这样的增量维护算法包括一系列的步骤：第一步为计算和应用主要增量(非空-导入元组)，然后应用二次增量(空-导入元组)从而删除或插入空-导入增量的一组后续步骤。主要增量被保存并在二次增量的计算中重新使用。为了正确地计算空-导入元组，这种计算可能需要再次访问基数关系。这些传统技术需要一个单独的SQL语句来实现每一个所需要的步骤，它们引发低效率并需要用于解析和执行多个SQL语句的资源。例如，如图3所示的用于视图V_1(在表1中定义)的关系X_2，视图更新算法包括五个步骤：计算和应用主要增量，并计算和应用对应于连接析取

范式关系$R_1R_2T_1T_2$、R_1R_2、$X_1Y_1Y_2$以及X_1的四个二次增量。

[0007] 因此，需要方法和系统，其通过使用用于每一个实体化外连接视图的单一维护更新语句，用于具有外连接的实体化视图的增量维护。

发明内容

[0008] 本发明包括用于优化包括外连接和实体化视图的查询的系统、方法以及计算机程序产品。

[0009] 根据本发明的实施例，对于具有外连接的实体化视图的增量维护，提出保存侧/导入空侧(PSNS)算法。这些算法使得对于每个实体化外连接视图仅产生单个维护更新语句，当更新引用基表时，其将被用于增量地更新实体化视图。因为这是通过RDBMS执行的通用SQL语句，为了达到较好的性能，这允许应用强大的优化同时依次处理更新语句。

[0010] 第二，由于对视图定义施加较少的限制，因为产生的SQL语句同时计算非-空-导入元组和空-导入元组，所以这允许立即实体化视图的扩展类别被支持。

[0011] 第三，既然仅使用一个更新语句，那么在视图更新操作过程中，不需要保存中间临时表。与传统方案相比，这还在性能方面产生了实质性的改进。

[0012] 结合参考附图，下面将详细地描述本发明进一步的特征和优点，以及本发明的各个实施例的结构和操作。应当注意：本发明不限于在此描述的具体实施例。这里介绍的实施例仅为说明发明的目的。基于这里所包括的教导，对于本领域技术人员而言其他实施例也是明显的。

附图说明

[0013] 并入于此并形成说明书一部分的附图描述本发明，以及结合说明书，进一步用于解释本发明的原理并使得相关领域的技术人员能够制备且使用本发明。

[0014] 图1提供了由对象-关系映射系统产生的生成外连接查询的一个例子。

[0015] 图2描述了按照本发明的实施例，用于视图V_1(在表1中定义)的注释PSNS-的标准化连接运算符树。

[0016] 图3说明了与视图V_1(表1中定义)的连接-析取范式对应的亲子关系图。

[0017] 图4-6是阐释了根据本发明的实施例的步骤的流程图，通过该步骤，立即实体化视图(iMVs)被增量地更新。

[0018] 图7提供了一个系统的模块视图，在该系统中可以实施本发明。

[0019] 图8和9是阐释了根据本发明的实施例步骤的流程图，通过该步骤，从用于具有外连接的立即实体化视图(iMVs)的注释PSNS-的标准化连接运算符树而创建生成的更新语句。

[0020] 图10描述了在其中实施本发明的计算机系统的一个示例。

[0021] 将参考附图描述本发明。在图中，一般来说，相同的参考标记指示相同的或功能类似的元件。此外，一般来说，参考标记的最左边的数指示参考标记第一次出现的图。

具体实施方式

[0022] 介绍

[0023] 本发明涉及用于增量地更新具有外连接的立即实体化视图(iMVs)的系统、方法、计算机程序产品实施例以及其组合及子组合。

[0024] 本发明的下面详细描述参考了阐释与本发明一致的示例实施例的附图。在本发明的精神和范围内,其它实施例是可能的,也可以对实施例进行改进。因此,详细的描述并不旨在限制本发明。相反,本发明的范围通过随附的权利要求限定。

[0025] 对本领域的技术人员而言,如下所述,本发明能够以软件、硬件、固件和/或图中阐明的实体的多种不同实施例而实施。具有实现本发明的硬件专用控制的任何实际的软件代码不限制本发明。因此,理解到,给定本说明表达的详细级别,能够进行实施例的改进和变型,进而描述本发明的操作行为。

[0026] 本发明涉及用于增量地更新具有外连接的立即实体化视图(iMVs)的实施算法的系统、方法以及计算机程序产品。

[0027] 根据本发明的一个实施例,该算法可与支持实体化视图的查询优化器整合,例如,但不限于,SQL ANYWHERE™优化器。iMVs的更新可以使用内部生成的触发器获得,一个触发器用于在iMV中引用的每一个关系。内部生成的触发器包含更新语句,考虑到ΔT更新关系代表基表T的已更新的行,更新语句执行对在它们的定义中引用T的iMV的更新。在一个实施例中,更新语句是内部生成的SQL语句,SQL语句通过查询优化器处理和优化,查询优化器正如充分利用存在于任何查询优化器中的查询优化技术的任何其他查询。因此,根据一个实施例,具有外连接的iMV的增量维护能够以类似的方式进行,这可以通过MERGE/INSERT/UPDATE语句实现。相关领域的技术人员会理解可使用其他编程语言,数据库平台和技术来实现这里探讨的逻辑和算法。

[0028] 其次,考虑到使用RDBMS服务器的应用的当前现状,施加至可以被立即维护的外连接类别的限制必须被保持在最低限度。传统的解决方案对选择列表的内容和施加至谓语的空-不相容的性能是非常限制的。通过放松空-不相容的性能限制(即,通过空-相容),在此描述的本发明的实施例改进了传统的方法。

[0029] 第三,外连接更新语句的新一代算法所需要的额外信息有效地被加入至查询优化器使用的外连接查询块的内部表示,所述查询优化器,例如但不限于,SQL ANYWHERE™优化器。

[0030] 这里所描述的算法PSNS()精确地实现了。

[0031] 词汇和术语

[0032] 为了帮助理解下面的讨论,下面定义的目的是说明,而不是限制。

[0033] 连接:连接子句结合来自两个或多个表的记录,例如,但不限于,数据库表。如本文所用,在一个实施例中,数据库可能是关系数据库,内存数据库,文件系统,一批Web服务器文档、文件、或资源,或任何一批电子数据中的一个或多个。

[0034] 外连接:外连接是一种不要求在两个连接表中的每个记录具有匹配记录的连接。连接表保留每个记录,甚至在没有其他匹配记录存在的情况中。外连接可能是左外连接,右外连接,或全外连接,这取决于那个表保存来自(左,右,或两者)的行。

[0035] 关系:关系定义为具有相同属性(列)的一组元组(行)。

[0036] 关系数据库:关系数据库是存储在关系(有时称为表)中的数据集合。关系数据库由IBM的E.F.Codd在1970年发明。

[0037] 结构化查询语言(SQL):SQL代表结构化查询语言。称为SEQUEL(结构化英语查询语言)的最初版本,由IBM在1970年设计。SQL-2008是用于SQL的最新标准,国际标准组织在2008年公布的一份文件中陈述了该标准;参看2008年七月出版的《SQL基础》。

[0038] 模式(T):模式(T)表示关系T的所有属性(列),模式(T)={$A_1,\cdots,A_n$}。

[0039] 元组:针对模式(T)的元组t是对模式(T)的属性名称的值分配,t=($a_1,..,a_n$)。t[A_i]=a_i,表示用于属性A_i的元组t的值。关系T的实例是一组对模式(T)中属性定义的元组,T={$t_1,..,t_k$}。对于就关系$T_1,...,T_n$的模式定义的元组t,符号t[T_i]表示用于关系T_i的属性的元组t的值。符号n(t[T])表示模式(T)中的所有属性在元组t中具有空,同时,nn(t[T])表示模式(T)中的至少一个属性是非空。另外,t=(...,n(T),...,nn(R)…)表示元组t在T也是就针对n(t[T])是空,并且在R,也就是针对nn(t[R])是非空。

[0040] 模式(p):如在此使用,在一个实施例中,谓语p对由模式(p)识别的一组属性定义。符号p($T_1,T_2,...,T_n,$)表示引用关系$T_1,T_2,\cdots,T_n$的一些属性的谓语,也就是,模式$(p) \subseteq U_{i=1,n}$模式(T_i)以及关系(p)={$T_1,T_2,\cdots,T_n$}。施加至元组$t_1 \in T_1, t_2 \in T_2,\cdots, t_n \in T_n$的谓语p具有三个值,即假、真或未知。当被应用时,符号p($t_1,...,t_n$)将表示:当谓语评估为假或未知时,谓语p评估真或($t_1,..,t_n$)<>真。

[0041] 外部联合:如在此使用,在一个实施例中,两个关系的T_1和T_2的术语外部联合由$T_1 \cup T_2$表示,以及它通过以下计算,首先对于(Schema(T_1)∪Schema(T_2))\Schema(T_i)中的属性,用Null填补每个关系T_i,i=1,2的元组,然后计算结果集合的联合。

[0042] 连接,反连接,左,右,和全部外连接:对于两个关系T_1和T_2,连接、反连接、左,右,和全连接分别定义如下:

[0043]
$$R \underset{lo}{\bowtie}_{p(R,T)} T = \{(r,t) | r \in R, t \in T, p(r,t) = True\}$$
$$R \underset{lo}{\triangleright}_{p(R,T)} T = \{r | r \in R, (\nexists t \in T)(p(r,t) = True\}$$
$$R \underset{ro}{\bowtie}_{p(R,T)} T = (R \underset{lo}{\bowtie}_{p(R,T)} T) \cup (R \underset{lo}{\triangleright}_{p(R,T)} T)$$
$$R \underset{fo}{\bowtie}_{p(R,T)} T = T \underset{fo}{\bowtie}_{p(R,T)} R$$
$$R \underset{fo}{\bowtie}_{p(R,T)} T = (R \underset{lo}{\bowtie}_{p(R,T)} T) \cup (R \underset{lo}{\triangleright}_{p(R,T)} T) \cup (T \underset{lo}{\triangleright}_{p(R,T)} R)$$

[0044] 空-导入元组:对于关系T的所有属性都是空-填充或空-扩展的元组,通过T-空-导入元组表示。

[0045] 空-导入元组:对于关系T的至少一个属性不为空的元组,通过T-空-导入元组表示。

[0046] 控制元组:如果它们基于相同模式定义,且t[A]=r[A],$\forall A \in Schema(t)$,其中,r[A]不是空,那么元组t控制元组r。

[0047] 复制元组:t和r是基于相同模式定义的元组。如果$\forall A \in Schema(t)$,t[A]没有区别于r[A],即,它们都是空或它们相等,则t是r的副本。

[0048] 最佳匹配运算符:最佳匹配运算符β定义为β(R)={r|r∈R,通过R中的任何元组,r不受控制或是副本,以及它至少有一个非空值}。应用到一个关系R的最佳匹配运算符消除了由其它元组控制或是其它元组的副本的所有元组,因此β(R)没有两个彼此控制或是副本

的元组。

[0049] 复制消除运算符:δ(R)用于复制消除运算符:δ(R)={r|r∈R,r不是R中任何一个元组的副本}。

[0050] 强谓语:如果属性(p)中的任何属性具有空值,如果谓语p不评估为真,那么谓语p是强的。

[0051] NS-不相容谓语:对于T-空-导入的元组,如果谓语不评估为真,那么谓语p(T_1,..,T_n)基于关系T∈{T_1,...,Tn}是NS-不相容的。NS-不相容谓语和强谓语之间的区别是:要求强谓语p基于模式(p)中的任何属性是空-不相容的。例如,T.XIS NOT DISTINCT FROM R.X[1]AND rowid(T)IS NOT NULL不是一个强谓语,但是根据NS-不相容定义:T.X可是空(NULL),但是谓语仍评估为真(True),其对于关系T是NS-不相容的。在我们的经验中,当其是来自图1的查询的情况时(例如,来自16-17,34-35行的ON谓语),大类别的客户查询不使用强谓语,甚至在ON子句中。对于某关系T_i,当谓语的性能为NS-不相容是很重要的,其通过下面的关系T_i,例如p(T_1,...,$\underline{T_i}$,…,T_n),表示。

[0052] 逻辑运算符树:逻辑运算符树是查询的表示,其中内部节点是二进制连接,同时叶是关系。逻辑运算符树中的每个节点可具有引用来自左子树和右子树的关系的谓语。

[0053] 外连接查询:如在此使用,在一个实施例中,术语外连接查询是包含左和全部外连接(右外连接可以转化为左外连接)以及内部连接的查询。外连接查询可以通过连接运算符树表征,连接运算符树的内部节点是二进制连接,并且叶为关系。

[0054] 空-导入外连接:对于类型左外连接的连接运算符树的内部连接节点,外连接空-导入来自右边的关系。对于一个全外连接节点,外连接在其两侧均空-导入。请注意,在外连接查询中,相同的基础关系可以由多于一个的外连接空-导入。

[0055] 直接外连接:对于通过运算符树表征的外连接查询中的关系T,T的直接外连接是空-导入T的类型左外连接或全外连接的第一个祖先节点。任何关系T能够具有至多一个直接外连接。

[0056] 间接外连接:对于由运算符树代表的外连接查询中的任何关系T,间接外连接是空-导入T但不是T的直接外连接的外连接节点。

[0057] ----------

[0058] [1]IS NOT DISTINCT FROM谓语在ANSI SQL:1999中引入,并且其等同于谓语T.X=R.X OR(T.X IS NULL AND R.X IS NULL)。

[0059] 具有外连接的实体化视图:其定义是一个外连接查询的实体化视图是具有外连接的实体化视图,其具有和不具有集合函数。

[0060] 在一个实施例中,关于在视图定义中使用的谓语,具有外连接的增量地维护的实体化视图被假定为具有下面的性能。外连接J的谓语是仅关于关系的NS-不相容谓语,其具有直接外连接,并且该直接外连接不同于J。施加至一些外连接谓语上的NS-不相容性能确保空-导入涟漪效应:如果关系T是由其直接外连接空-导入的,那么所有其间接外连接还必须空-导入它们的整体空-导入方面。因此,通过具有空-导入的T的外连接查询生成的任何元组将具有也是空-导入的所有相关关系。例如,对于外连接查询$(R \overset{lo}{\bowtie}_{p_1(R,T)} T) \overset{fo}{\bowtie}_{p(T,S)} S$,上

述的条件要求：只有全外连接$\overset{fo}{\bowtie}_{p(\underline{T},S)}$的谓语p(T,S)对于关系T是NS-不相容，关系T是通过其直接左外连接$R \overset{lo}{\bowtie}_{p(R,T)}$ T空-导入。谓语$\overset{lo}{\bowtie}_{p,(R,T)}$不必要是NS-不相容的，因为涟漪效应不适用其他外连接：$\overset{fo}{\bowtie}_{p(\underline{T},S)}$。$\overset{fo}{\bowtie}_{p(T,S)}$将为关系R和T空-导入或均不为两者空-导入，因此其对左外连接$R\overset{lo}{\bowtie}_{p,(R,T)}$ T的结果没有影响

[0061] 在一个实施例中，假定：实体化视图具有唯一索引，该唯一索引具有空值，该空值在可能是空值的属性上无法区分。例如，如果唯一索引关于属性(A,B)定义，两个元组(α,Null)和(α,Null)被认为是等同的且不能同时存在于视图中，这是因为它们违反索引的唯一性。当PSNS算法被描述时，将在稍后讨论对于具有外连接的立即实体化视图的额外需求。

[0062] PSNS保存侧/空导入侧算法

[0063] 对于定义了立即实体化视图V的给定外连接查询，所提出的算法找到基础关系T的空-导入性能的表示，在对关系T进行更新操作后，对于视图V的更新语句，该基础关系T会给出正确公式。主要目标是对视图定义施加尽可能少的限制，并且视图更新语句将十分有效且执行实体化视图V的增量维护。

[0064] 假设：待被更新的关系T是由视图定义[2]中的至少一个外连接空-导入。如果视图V引用一组关系$T_1,T_2,...,T_n$和关系T，对于关系$T_1,...,T_n$和T的实例，V(nn(T))={t\t∈V(T,$T_1,T_2,...,T_n$),nn(t[T])}表示实例$V(T,T_1,T_2,...,T_n)$中的所有元组的集合，实例$V(T,T_1,T_2,...,T_n)$不是空-导入关系T(T-null-supplied)。V(nn(T))可以通过使用最初的视图定义来计算，其中空-导入T的所有外连接被转化为内部或左外连接，使得T不再是空-导入。

[0065] 举例，如果 $V_3=(R \overset{fo}{\bowtie}_{p(R,T)} T) \overset{fo}{\bowtie}_{p(\underline{T},S)} S$, $V_3(nn(T))=(R \overset{ro}{\bowtie}_{p(R,T)} T) \overset{lo}{\bowtie}_{p(\underline{T},S)} S$。V(n(T))是指实例$V(T,T_1,\cdots T_n)$中的所有T-null-supplied元组的集合。

[0066] V中的T-null-supplied元组的最大集合，由$Null(T,V(T,T_1,T_2,...,T_n))$表示为就固定关系$T_1,T_2,...,T_n$和关系T的任何实例计算的视图V的任何实例中的所有可能T-null-supplied元组的集合，关系T的任何实例即$Null(T,T_1,T_2,...,T_n))=\delta(\cup_{\text{any instance of T}}V(n(T)))$。如在此使用，在本发明的实施例中，假设：对于实体化视图V，只有一个关系被更新，并且被视图引用的所有其他关系保持不变。因此，只要有可能，在固定关系没有提到之处使用简化的符号，例如，Null(T,V)=Null(T,V(T1,$T_1,T_2,..,T_n$))，和$V(T,T_1,T_2,..,T_n)=V(T)$。通过使用最初的视图定义得到Null(T,V)，其中关系T由空集(也就是，$Null(T,V)=V(T\rightarrow\varphi)$)取代。在前面的示例中，$\text{Null}(T,V_3)=(R\overset{fo}{\bowtie}_{p(R,T)}\varphi)\overset{fo}{\bowtie}_{p(T,S)} S$。Null(T,V)包含了所有可能的T-null-supplied元组，无关T的内容，T-null-supplied组可以存在于V中。也就是，对于关系T的任何实例，V中的任何T-null-supplied元组存在于Null(T,V)中，只要所有其他基础关系的内容不改变，即$V(n(T))\subseteq Null(T,V)$。在前面的例子中，如

[0067] ----------

[0068] [2]对于不能由外连接空-导入的关系，视图更新语句类似于用于内部连接立即实体化视图的公式。果R={r_0,r_1}，S={s_0}，那么任何可存在于V_3中的T-null-supplied元组是下面元组中的一个：

[0069] Null(T,V)={(r_0,Null,Null),(r_1,Null,Null,(Null,Null,s_0)}。

[0070] 对于具有外连接的实体化视图的增量维护，对于使用集合ΔT对关系T的每个插入或删除操作，待被插入或待被从V(nn(T))中删除的元组是已知的。这用V(nn(ΔT))表示的该集合，可以通过使用在V(nn(ΔT))定义中的ΔT而不是T得到。然而，每个这样的操作可具有与T-null-supplied元组相关的副作用。对于对T的删除操作，从V中删除元组，即，V(nn(ΔT))，可根据新T-null-supplied元组的需要留下视图V。进一步地，在V中插入新元组，即，V(nn(ΔT))，可能会在V中留下一些假的旧元组，这些元组现在有V(nn(ΔT))中的新元组控制。在使用V(nn(ΔT))更新V后，术语“NS-补偿”用于描述在V中插入或删除T-null-supplied元组的操作。

[0071] 目标是设计一个具有以下性能的算法：(1)对于每个元组t∈V(nn(ΔT))，在计算V(nn(ΔT))的同时计算可能的T-null-supplied元组集合，Null(t,T,V)，无需访问视图V。Null(t,T,V)实际上是通过Null(T,V)中的t控制的所有元组的集合。在插入、删除或更新操作后，Null(t,T,V)将被用于NS-补偿；(2)对于每个元组t'∈Null(t,T,V)，决定，使用视图V决定，如果t'将被删除(对于插入以及更新操作)或插入(对于删除或更新操作)V中从而用于V(nn(ΔT))中插入或删除的元组的NS-补偿。(3)没有必要将部分增量保存到临时表，因此单个语句都应该用于使用V(nn(ΔT))的更新操作和使用Null(ΔT,T,V)的NS补偿。

[0072] 对于每个基础关系T，算法PSNS(表4中的算法2)计算集合PSNS(T)(保存侧/空导入侧)，其是成对的子集(ps_i, ns_i)，ps_i，ns_i，$\subseteq Relations(V)$的集合，其中，每个这种子集对精确地描述可从元组t∈V(nn(ΔT))中得到一个T-null-supplied元组，Null(i,t,T,V)。此外，一对(ps_i,ns_i)∈PSNS(T)提供了用于检查使用用于V的NS-补偿的元组Null(i,t,T,V)的需求的条件。集合ns_i包含关系(V)中的所有关系，关系(V)必须与元组Null(i,t,T,V)中的T一起被空-导入。集合ps_i代表了保存在元组Null(i,t,T,V)中的所有关系。当然，当且仅当最初元组t不空-导入ps_i中的关系时，元组Null(i,t,T,V)可以存在。用于在don'tcare(i,T,V)=Relations(V)\(ns_i∪ps_i)中剩余关系的值在Null(i,t,T,V)中不改变。任何元组Null(i,t,T,V)由元组t控制，因此，它们不能同时存在于视图V中。

[0073] 用于集合Null(i,t,T,V)，Null(t,T)，和Null(ΔT,T,V)的形式定义通过以下给出：

[0074] 方程(1)：

[0075] don'tcare(i,T,V)=Relations(V)\(ns_i∪ps_i)。

[0076] Null(i,t,T,V)==(t[don'tcare(i,T,V)],t[ps_i],n(T),n(ns_i))

[0077] Null(t,T,V)=∪$_{(psi,nsi)\in PSNS(T)}$nn(t[psi])Null(i,t,T,V)

[0078] Null(ΔT,T,V)=∪$_{t\in V(nn(\Delta T))}$Null(t,T,V)

[0079] 注意：集合Null(t,T,V)不包含副本或假的元组，因为每对(ps,ns)∈PSNS(T)产生不同的元组，即，Null(t,T,V)=δ(Null(t,T,V))。然而，集合Null(ΔT,T,V)可能包含副本，这是因为ΔT中的两个不同元组可能产生相同的T-null-supplied元组。可以证明，如果V具有PSNS算法所需要的所有性能，上述定义的Null(T,V)通过方程2中的公式得到。换句话说，

所有的T-null-supplied元组可以从视图V的任何实例中计算：

[0080] 方程(2)：

[0081] $$Null(T, V) = V(T \rightarrow \varphi, T_1, T_2, ..., T_n)$$

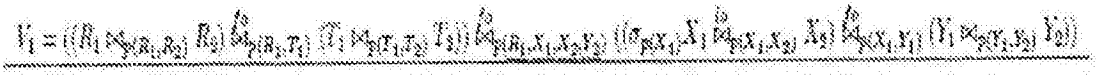
[0082] $$= \delta(\cup_{any\ instance\ of\ T, t \in V(nn(T))} Null(T, V))$$

[0083] 对于举例，考虑表1和图3中定义的视图V_1，使用ΔX_2更新的关系X_2。对于元组$t \in V(nn(\Delta X_2))$，已知的是：$t[X_2]$必须不为空(集合$(V(nn(\Delta X_2))$只包含X_2-null-supplied)：$t=(r_1,r_2,t_1,t_2,x_1,x_2,y_1,y_2)$。集合$PSNS(X_2)$被计算为$\{(\{X_1\},\{R_1,R_2,T_1,T_2,X_2\}),(\{R_1\},\{X_1,X_2,Y_1,Y_2\})\}$。直观地说，第一$(ps_1,ns_1)=(\{X_1\},\{R_1,R_2,T_1,T_2,X_2\})$表示$X_2$-null-supplied元组，其中保存关系$X_1$。当$X_2$在$Null(1,t,X_2,V_1)$中是空-导入时，$X_1$将被保存。以及用于全部外连接$S_0^{fo}$的谓语$p(r_1,x_1,Null,y_2)<>True$，然后，除了$X_2$以外，全部外连接的另一侧也是空-导入。这可以解释为什么$ns_1=\{R_1,R_2,T_1,T_2,X_2\}$正是全部外连接$S_0^{fo}$的另一侧。如果$t[X_1]=x_1$不为空，则具有$y_1$和$y_2$的$Null(1,t,X_2,V_1)=(Null,Null,Null,Null,x_1,Null,y_1,y_2)$保留与元组t相同。$y_1$和$y_2$可以是空或非空值，并且$Null(1,t,X_2,V_1)$的计算独立于其值。类似地，如果$t[R_1]=r_1$不是空，那么$Null(2,t,X_2,V_1)=(r_1,r_2,t_1,t_2,Null,Null,Null,Null)$。$Null(2,t,X_2,V_1)$中保存的关系是$R_1$，并且值$r_2,t_1,t_2$保持不从t改变。然而，当谓语$p(r_1,x_1,Null,y_2)<>True$时，全部外连接$S_0^{fo}$空-导入整个右侧。如果对关系$X_2$的更新操作是插入，如果元组$Null(i,t,X_2,V_1)$存在于V中，它将被删除，因为它是被新元组t控制的假的元组。如果更新操作是删除，元组$Null(i,t,X_2,V_1)$可用于NS-补偿，如果从V中删除$V(nn(\Delta T))$后，在V中没有元组具有值$t[ps_i]$，即，$\nexists t' \in V, t'[ps_i] = t[ps_i]$。

[0084] 表4中描述的算法2计算用于外连接立即实体化视图V中的每个基础关系的PSNS()集合。算法是使用用作输入的标准化连接运算符树，其是查询运算符树的紧凑的表示，其中非嵌套的外连接在相同的子计划中是“扁平的”。在图2中提供了这种标准化的运算符树的例子。设计注释PSNS-的标准化连接运算符树从而支持查询优化器中具有外连接的语句的优化，包括但不限于，SQL ANYWHERE™优化器。标准化连接的主要性能是：其关系通过直接或间接外部外连接都一起被空-导入。例如，在图3中，表示表表达式$((R_1 \bowtie_{p(R_1,R_2)} R_2) \overset{fo}{\bowtie}_{p(R_1,T_1)} (T_1 \bowtie_{p(T_1,T_2)} T2))$的标准化连接$S_1^{fo}$，可以通过外连接$\overset{fo}{\bowtie}_{p(\underline{R_1},\underline{X_1},\underline{X_2},\underline{Y_2})}$被全部空-导入。直观上，作为关系集合的辅助方案代表外连接的原子空-导入方面。在一个实施例中，对于每个辅助方案S，基于用于外连接的谓语，算法PSNS()使用标准化连接运算符树的后序遍历并计算$(ps,ns) \in PSNS(S)$集合。ns是其他关系的集合，如果S是空-导入，则其为空-导入。PS是保存关系的集合，如果S是空-导入，则其被保存。在应用PSNS算法之后，注释PSNS-的标准化连接运算符树包含所有元数据，对于任何关系$T \in Relations(V)$，这些元数据用于产生用于实体化视图V的增量维护的更新语句。图3示出了用于其中$x \in \{lo,fo,j\}$的每个辅助方案S_i^x的最终PSNS集合。在下面的部分中，参照表1-4描述实施例，其中PSNS()集合用于产生用于立即实体化视图的更新语句。

[0085] 表1：用于V_1的注释PSNS-的标准化连接树

[0086]



$PSNS(R_1) = PSNS(R_2) =$	$\{(\{T_1\},(\{R_1, R_2\}, \{X_1, X_2, Y_1, Y_2\}), (X_1, X_2, Y_2,\}, \{R_1, R_2, T_1, T_2\})\}$
$PSNS(T_1) = PSNS(T_2) =$	$\{(\{R_1\}, \{T_1, T_2\})\}$
$PSNS(X_1) =$	$\{(\{Y_1\}, \{R_1, R_2, T_1, T_2, X_1, X_2\}), (\{R_1\}, \{X_1, X_2, Y_1, Y_2\})\}$
(P2) $(X_2) =$	$\{(\{X_1\}, \{R_1, R_2, T_1, T_2, X_2\}), (\{R_1\}, \{X_1, X_2, Y_1, Y_2\})\}$
PSNS $(Y_1) = PSNS(Y_2) =$	$\{(\{X_1\}, \{R_1, R_2, T_1, T_2, Y_1, Y_2\}), (\{R_1\}, \{X_1, X_2, Y_1, Y_2\})\}$
(P2) *MS* –intolent:	$P(\underline{R_1}, \underline{X_1}, \underline{X_2}, \underline{Y_2})$ on R_1, X_1, X_2, Y_2
(P3) *pk* in select list:	$R_1, T_1, X_1, X_2, Y_1, Y_2$
(P4) *nn*(), *n*() in select list:	R_1, T_1, X_1, X_2, Y_1

[0087] 表2：注释PSNS-标准化连接树的例子

[0088]

$V_2 = (R \overset{lo}{\bowtie}_{p(R,T)} T) \overset{lo}{\bowtie}_{p(T,S)} S$

PSNS (R) =	{({S},{R, T})}
PSNS (T) =	{({R}, {T, S}), ({S}, {R, T})}
PSNS (S) =	{({T}, {S})}
(P2) *MS* -intolerant:	p(T, S) on T
(P3) pk	R, S, T
(P4) nn(), n()	R, S, T

$V_3 = (R \overset{fo}{\bowtie}_{p(R,T)} T) \overset{fo}{\bowtie}_{p(T,S)} S$

PSNS (R) =	{({T}, {R})}
PSNS (T) =	{({R}, {T, S}), ({S}, {R, T})}

[0089]

PSNS (S) =	{({T}, {S})}
(P2) $\not\bowtie S$ -intolerant:	p(T, S)
(P3) pk	T, R, S
(P4) nn(), n()	T, R, S

$V_4 = (R_1 \overset{lo}{\bowtie}_{p(R,T,S)} (S \overset{lo}{\bowtie}_{p(T,S)} T)) \propto_{p(R_1, R_2)} R_2$

$PSNS(R_1) = PSNS(R_2) =$	∅
PSNS(T) = PSNS (S) =	{({R_1}, {T, S})}
(P2) $\not\bowtie S$ –intoleratn:	none
(P3) pk	R_1
(P4) nn(), n()	only one of the relations T, S

$V_5 = (R_1 \overset{lo}{\bowtie}_{p(R,S)} (S \overset{lo}{\bowtie}_{p(T,S)} T)$

PSNS (R) =	∅
PSNS (S) =	{({R}, {T, S})}
PSNS (T) =	{({S}, {T})}
(P2) $\not\bowtie S$ –intolerant:	none
(P3) pk	R, S
(P4) nn(), n()	S, T

$V_6 = (R \overset{lo}{\bowtie}_{p(R,S)} (S \overset{lo}{\bowtie}_{p(R,T)} T)$

PSNS (R) =	∅
PSNS (S) =	{({R}, {S})}
PSNS (T) =	{({R}, {T})}
(P2) $\not\bowtie S$ –intolerant:	none
(P3) pk	R
(P4) nn(), n()	S, T

$V_7 = (R \overset{fo}{\bowtie}_{p(R,T)} T) \overset{lo}{\bowtie}_{p(T,S)} (S \overset{lo}{\bowtie}_{p(S,W)} W)$

PSNS (R) =	*{({T}, {R})[don't care = {S, W}]}*
PSNS (T) =	{({R}, {T, S, W})[*don't care* = ∅],

[0090]

	({S},{R, T})[*don't'care* = {W}]}
PSNS (S) =	{({T}, {S, W})[*don't'care* = {R}]}
PSNS (W) =	{({S}, {W}) [*don't'care* = {R, T}]}
(P2) $\mathcal{NS}$ –intolerant:	p(T, S)
(P3) pk	T, R, S
(P4) nn(), n()	T, R, S, W

[0091] 表3：算法1-建立标准化连接树

[0092]

1: 程序：从$\mathcal{T}$中建立一个标准化运算符树

2: 输入：$\mathcal{T}$，节点 n，以及直接父类 $n\mathcal{J}$ = *(Dn$\mathcal{J}$, DRels = {T_1,.....,T_m}, n$\mathcal{J}$s ={n$\mathcal{J}_1$.....n$\mathcal{J}_m$}，p)*

Rels(n$\mathcal{J}$)$\stackrel{def}{=}$ DRels U_i Rels(n$\mathcal{J}_i$)。

3: 所有的语义转换，例如连接消除，外连接至内部连接转换，谓语下推已经施加至运算符树$\mathcal{T}$

4: n$\mathcal{J}_L$ = $\mathcal{N}$ull, n$\mathcal{J}_R$ = $\mathcal{N}$ull, n$\mathcal{J}_F$ = $\mathcal{N}$ull

5: 如果 n 是具有谓语 p 的关系 T: 然后

6: n$\mathcal{J}$ .p = n$\mathcal{J}$ .p^p, n$\mathcal{J}$ .DRels$\cup$ = {T}, T.Dn$\mathcal{J}$ = n$\mathcal{J}$

7: 返回

8: 如果 n 是具有谓语 p 的内部连接: 然后

9: n$\mathcal{J}$ .p = n$\mathcal{J}$ .p $\wedge$ p

10: 调用 n$\mathcal{T}$ ($\mathcal{T}$, n 的左子类，n$\mathcal{J}$)

11: 调用 n$\mathcal{T}$ ($\mathcal{T}$,n 的右子类, n$\mathcal{J}$)

12: 去检查谓语

13: 如果 n 是具有谓语 p 的左外连接: 然后

14: 调用 n$\mathcal{T}$（$\mathcal{T}$，n 的左子类，n$\mathcal{J}$）

15: n$\mathcal{J}_L$ =类型左外连接的新标准化连接

16: n$\mathcal{J}_L$.Dn$\mathcal{J}$ = n$\mathcal{J}$, n$\mathcal{J}$, n$\mathcal{J}$s $\cup$ = {n$\mathcal{J}_L$}, n$\mathcal{J}_L$.p = p

17: 调用 n$\mathcal{T}$ ($\mathcal{T}$,n 的右子类, n$\mathcal{J}_L$)

[0093]

18: 去检查谓语

19: 如果 n 是具有谓语 p 的全部外连接: 然后

20: $n\mathcal{J}_F$ =类型全部外连接的新标准化连接

21: $n\mathcal{J}_F.Dn\mathcal{J} = n\mathcal{J}, n\mathcal{J}, n\mathcal{J}sU = \{n\mathcal{J}_F\}, n\mathcal{J}_F.p = p$

22: $n\mathcal{J}_L$ =类型外连接的新标准化连接

23: $n\mathcal{J}_L.Dn\mathcal{J} = n\mathcal{J}_F$

24: $n\mathcal{J}_R$ =类型外连接的新标准化连接

25: $n\mathcal{J}_R.Dn\mathcal{J} = n\mathcal{J}_F$

26: $n\mathcal{J}_F.n,\mathcal{J}s = \{n\mathcal{J}_L, n\mathcal{J}_R\}$

27: 调用 $n\mathcal{T}$ ($\mathcal{T}$,n 的左子类, $n\mathcal{J}_L$)

28: 调用 $n\mathcal{T}$ ($\mathcal{T}$,n 的右子类, $n\mathcal{J}_R$)

29: 去检查谓语

30: 检查谓语: /*如果 n 不是的直接外连接, (P_1) p 必须引用 n 的左子和右子类, 且 Rels(p) ⊆ Rels($n\mathcal{J}$) (P2) p 必须 T ∈ Rels(p)的 $\mathcal{N}$S-不相容。*/

31: 对于所有 T ∈ Rels(p)): 进行:

要求: 如果 $Dn\mathcal{J}$ (T) ∉ {$n\mathcal{J}$, $n\mathcal{J}_L$, $n\mathcal{J}_R$}, 则 P 必须是 T 的 $\mathcal{N}$S-不相容。

[0094] 表4:算法2-建立注释PSNS-标准化连接树

[0095]

1: 程序: 对于标准化连接，计算 *PSNS()*（保存侧/空-导入侧）集合

2: $n\mathcal{J} = (Dn\mathcal{J}, DRels, n\mathcal{J}s, p) \in nT(V)$

$PSNS(T) \stackrel{def}{=} PSNS(Dn\mathcal{J}(T))$

3: 规则 0: /*如果 n$\mathcal{J}$是元组 t 中的空-导入，然后所有其关系必须是 t 中的空-导入*/

4: PSNS (n$\mathcal{J}$) = {($\mathcal{Q}$, Rels(n$\mathcal{J}$))}

5: 对所有 $n\mathcal{J}' \in n\mathcal{J}s(n\mathcal{J})$进行

[0096]

6: *PSNS* (*nJ*')

7: 对所有 *T* ∈ *Rels(p)*进行

8: 如果 nJ类型是左外连接，然后

9: 规则 1: /* (*T*...) $\ltimes^{lo}_{p(T,...)}$ (nJ) : *T* 在左外连接 nJ的保存侧中。如果 T 为元组 *t* 中的空-导入，然后 nJ必须也是 *t* 中的空-导入，因为谓语 *p* 不能评估为*真*（涟漪效应）: */

10: 如果 T ∈ Rels(nJ)然后

11: 联合 NS (nJ,T)

12: 规则 2: /* ... $\ltimes^{lo}_{p(T,...)}$ (T...): T 位于左外连接 nJ的空-导入侧。如果 nJ是被空-导入的，那么在 Rels（p）\ Rels(nJ)中的关系必须保存。*/

13: 如果 *T* ∈ *Rels(nJ)*然后

14: 对于所有*(ps, ns)* ∈ *PSNS (nJ)*进行

15: *ps* ∪ = *Rels*(p) \ *Rels*(nJ)

16: 如果 nJ类型是全部外连接，然后

17: /* nJs(nJ) = {nJ L,nJ R} */

18: 如果 *T* ∈ *Rels(nJ L)*，然后

19: (nJ 1, nJ 2) = (nJ L, nJ R)

20: 如果 *T* ∈ *Rels(nJ R)*，然后

21: (nJ 1, nJ 2) = (nJ R, nJ L)

22: 规则 3: /*如果 T 通过外连接被空-导入，而不是全部外连接，然后 nJ 2 必须也是空-导入，因为谓语 *p* 不能评估为*真*（涟漪效应）: */

23: 对于 nJ' = DnJ (T)nJ' ≠ nJ nJ' = DnJ (nJ')进行

24: 联合 NS (nJ2, nJ')

25: 规则 4: /*如果 nJ 1是空-导入， nJ 2中的关系必须保存: */

26: 对于 nJ' = DnJ (T)nJ' ≠ nJ nJ' = DnJ (nJ')进行

[0097]

27: ns = {nJ 1}

28: *ps = (Rels(nJ 2) ∩ Rels(p))*

29: *PSNS (nJ') ∪ = {(ps, ns)}*

/ * 检查选择列表：检查对视图$\overline{V} = \pi_{v_1,\ldots,v_k}$ V: (P3)的选择列表的限制，ps_i 集合中的任何关系必须具有主关键字，以及主关键字属性必须在选择列表的表达式（P4）之间。对于 ns_i 集合中的任何标准化连接 *nJ*，谓语 *nn(nJ)*和 *n(nJ)*必须使用选择列表表达式产生。*/

30: 对于所有的 *R* ∈ *Rels(nT ($\overline{V}$), (ps, ns)* ∈ *PSNS (R)*: 进行

31: 对所有 T ∈ ps: 进行

要求：T 具有主关键字 T.pk = {$C_1,\ldots,C_3$} ⊆ {$v_1,\ldots,v_k$} /*谓语 $\wedge_{1<=i<=j}$ ($\overline{V}$.C_i = $\overline{V(\Delta T)}$, C_i)是需要的*/。

32: 对所有的 nJ ∈ ns: 进行

要求：∃*S* ∈ *DRels(nJ)*和 *S.C* = *C* ∈ *{$v_1,\ldots,v_k$}*非空属性/* 谓语 n($\overline{V}$.C)和 nn($\overline{V}$.C): $\overline{V}$.C IS [NOT] NULL 是需要的*/。

[0098] 示例性实施例

[0099] 根据一个实施例，在查询优化过程中被增值地维护或使用的实体化视图，被内部表示成注释PSNS-的标准化连接运算符树(*nJ*)，该注释PSNS-的标准化连接运算符树(*nJ*)能够是由基于成本的查询优化器使用的相同表示，例如但不限于SQL ANYWHERETM优化器。在语义转换和查询分析应用到最初的查询运算符树后，(*nJ*)被建立。视图匹配算法和用于实体化视图增量维护的更新语句的生成均使用代表视图定义的(*nJ*)。

[0100] 在一个实施例中，对于立即实体化视图，由于服务器被启动，注释PSNS-的(*nJ*)在对视图的第一次引用建立。如果关系T被更新，内部生成的触发器被创建，其包括用于引用关系T的任何立即实体化视图的更新语句。该部分描述了一些更新语句，所述更新语句为用于具有外连接的iMVs的这些内部触发器而产生。每个更新语句是个SQL语句，其可能是插入、更新或合并语句。更新语句从实体化视图的注释PSNS-的*nJ*表示生成。当ΔT传递至触发器，在对关系T进行任何更新操作后，完成内部生成的触发器执行。每个生成的更新语句像任何其他SQL查询一样处理，因此，在例如外键限制开发的查询优化器中支持的所有优化、外部和内部连接消除被应用从而寻找有效执行计划。设计生成的算法从而产生正确的更新语句，其可以通过查询优化器有效地优化，例如但不限于，SQL ANYWHERE™优化器。在本发明的示例实施例中，在此所描述的技术在SQL ANYWHERE™12.0中实施。

[0101] 下面的操作部分描述SQL语句，所述SQL语句为具有外连接的iMVs产生。在所有的语句中，派生表V(nn(ΔT))等同于V(nn(ΔT))，但是保留计算V(nn(ΔT))的选择列表以及

Null(ΔT,T,V)中的元组需要所有列。应当注意:Null(ΔT,T,V)中选择列表的复杂表达式不能直接从V(nn(ΔT))中直接计算,因为所需要的列在最终选择列表中的投射出。对于删除和更新操作,通过使用实体化视图必须具有的唯一索引应用V(nn(ΔT))。条件V.ui IS NOT DISTINCT FROM V(nn(ΔT)).ui(请见描述插入操作的表6中第22行,以及描述更新操作的表9中的第24行)包含应用至唯一索引列的所有IS NOT DISTINCT FROM谓语。对于删除和更新操作,在V(nn(ΔT))应用至视图后,完成Null(ΔT,T,V)的计算。通过分别使用嵌入式的DELETE(请见描述删除操作的表8的第18-23行)以及MERG(描述UPDATE操作的表9中的行21-32)

[0102] 插入操作(表5,表6)

[0103] 根据实施例,对于插入操作,在应用V(nn(ΔT))之前,集合Null(ΔT,T,V)的计算可以见到视图数据,因为在插入操作前,假的T-null-supplied元组存在于视图中。因此,与Null(ΔT,T,V)(表6,9-20行)同时,合并语句计算V(nn(ΔV))(表6,7-8行)。为了效率,只有在V∩Null(ΔT,T,V)中发现的元组rowids被传递至通过WHENMATCHED子句处理。在一个实施例中,MERGE语句首先处理通过WHEN MATCHED子句(表6,24行)(这是NS-补偿操作)删除的假元组。其次,通过WHEN NOT MATCHED子句(表6,25-26行),插入来自V(nn(ΔT))的行。下面的表5描述了典型的逻辑以计算用于插入操作的应用Null(i,t,T,V)。根据本发明的一个实施例,下面的表6提供了用于插入操作的典型MERGE SQL语句。

[0104] 表5:对于插入操作,计算并应用Null(i,t,T,V)

1: for all $t \in V(nn(\Delta T))$ do

Ensure: $nn(t[T])$

2: for all i, such that $(ps_i, ns_i) \in PSNS(T), ps_i \neq \emptyset nn(t[ps_i])$ do

3: /*Generate *T-null-supplied tuple* in $Null(i,t,T,V)$ */

4: $\mathcal{N}ull(i, t, T, V) =$

[0105] 5: $= (t.\{Rels(V) \setminus ns_i\}, n(ns_i))$

6: $\overline{Null(i,t,T,V)} =$

7: $= v_1(\mathcal{N}ull(i, t, T, V)), \ldots, v_k(\mathcal{N}ull(i, t, T, V)))$

/* $\overline{Null(i,t,T,V)}. \in \overline{V}$ iff $\overline{V}.ps_i = t[ps_i]$和 $n(\overline{V}.ns_i)$ */

8: if $\overline{Null(i,t,T,V)} \in \overline{V}$ then

9: $\overline{V} = \overline{V} \setminus \{\overline{Null(i,t,T,V)}\}$ // delete $\overline{Null(i,t,T,V)}$

[0106] 表6:用于插入操作的合并语句

1: MERGE INTO $\overline{V}$ USING

2: (WITH $V(nn(\Delta T))$ AS

3: (SELECT*,ΔT.DML_type FROM V($nn(\Delta T)$))

4: SELECT DT.*

[0107]

5: FROM $V(nn(\Delta T))$,LATERAL (/*¬T-*Null-supplied* : $\overline{V(nn(\Delta T))}$TO BE INSERTED*/

6: SELECT $v_1\ V(nn(\Delta T)),\ldots,v_k\ V(nn(\Delta T))$,

7: $V(nn(\Delta T))$.*DML_type* AS *DML_type Null AS d_ rowid*

8: UNION ALL

/* T- *Null-supplied* : *Null (1, t, T, V)* for *NS*-compensation(delete) */

9: SELECT *Null* AS $v_1,\ldots,$ *Null* AS v_k,

10: $V(nn(\Delta T))$. *DML_type* * *(–1)*AS DML_type,rowid($\overline{V}$) *d_rowid*

11:FROM $\overline{V}$ WHERE $\overline{V}$ $.ps_1 = V(nn(\Delta T))\ .ps_1$ AND $n(\overline{V}\ .ns_1)$

12: UNION ALL

13: ...

14: UNION ALL

/* T-*null-supplied* : *Null (m, t, T, V)* for *NS*-compensation(delete) */

[0108]

15: SELECT *Null* AS $v_1,\ldots,$ Null ASv_k,

16: V(nn(ΔT).DML_type * (–1) AS DML_type, rowid($\overline{V}$) d_rowid

17: FROM $\overline{V}$ WHERE $\overline{V}$ $.ps_m = V(nn(\Delta T))\ .ps_m$ AND $n(\overline{V}\ .ns_m)$

18:) AS DT

19:) AS ΔV (ΔT)

20: ON DML_type = -1 AND rowed($\overline{V}$) = $\Delta V\ (\Delta T).\ d_rowid$

21: WHEN MATCHED THEN DELETE

22: WHEN NOT MATCHED AND DML_type = *+1*

23: THEN INSERT

[0109] 删除操作(表7,表8)

[0110] 根据实施例,对于删除操作,在应用V(nn(ΔT))后,集合Null(ΔT,T,V)的计算必须只见到视图数据,使得只有当其对于NS-补偿操作所需求时,元组Null(i,t,T,V)产生。

[0111] 因此，在一个实施例中，INSERT语句首先计算（表8，21行）和运用（表8，18行）V(nn(ΔT))。嵌入式的DELETE语句（表8，18-23行）用于计算和应用V(nn(ΔT))。（嵌入式的更新语句（也被称为select-from-DML）首先被执行，因此，在应用V(nn(ΔT))后，查询的其余部分只见到修改的数据，在我们的情况中，是修改的视图）。当检查用于NS-补偿的条件时（表8，第9行和16行），INSERT语句的剩余部分会见到视图V的修改数据。

[0112] 在DELETE语句（表8中第5至17行）和Null(ΔT,T,V)中的元组被插入视图后（这是NS-补偿操作），计算集合Null(ΔT,T,V)。

[0113] 表7：对于DELETE操作，计算并应用Null(i,t,T,V)

[0114]

1: for all $t \in V(nn(\Delta T))$ do

Ensure: $nn(t[T])$

2: for all I such that $(ps_i, ns_i) \in PSNS(T)$, $ps_i \neq \emptyset$, $ns_i \neq \emptyset$, $nn(t[ps_i])$ do

3: //Generate T-*Null-supplied tuple* in $\mathcal{N}ull(i, t, T, V)$

4: $\mathcal{N}ull(i,t,T,V) =$

5: $= (t[Rels(V) \setminus ns_i], n(ns_i))$

6: $\mathcal{N}ull(i, t, T, V) =$

7: $= (v_1(\mathcal{N}ull(i, t, T, V)), \ldots, v_k(\mathcal{N}ull(i, t, T, V)))$

8: if $\sigma_{\overline{V}.ps_i = t[ps_i]}\ \overline{V} = \emptyset$ then

9: $\overline{V} = \overline{V} \cup \{\overline{\mathcal{N}ull(i,t,T,V)}\}$ //insert $\overline{\mathcal{N}ull(i,t,T,V)}$

[0115] 表8：对于删除（DELETE）操作的插入（INSERT）语句

1: INSERT INTO $\overline{V}$

2: (WITH V(nn(ΔT)) AS

3: (SELECT * FROM $V(nn(\Delta T))$

4: SELECT DISTINCT $\overline{Null(\Delta T, T, V)}.*$

5: FROM $V(nn(\Delta T))$, LATERAL (

[0116] 6: {T-*null-supplied* : $\overline{Null(1,t,T,V)}$ for $\mathcal{N}$S-compensation (insert)}

7: SELECT $v_1(\mathcal{N}ull(l, t, T, V)),\ldots,v_k(\mathcal{N}ull(l, t, T, V))$

8: WHERE nn(V(nn(ΔT)) .ps_1)AND

9: NOT EXISTS (SELECT 1 FROM $\overline{V}$

WHERE $\overline{V}$. $ps_1 = V(nn(\Delta T))$ .ps_1)

10: UNION ALL

11: …

12: UNION ALL

13: {T-*null-supplied* : $\overline{Null(m,t,T,V)}$ for $\mathcal{N}$S-compensation (insert)}

14: SELECT $v_1(\mathcal{N}ull(m, t, T, V)),\ldots,v_k(\mathcal{N}ull(m, t, T, V))$

15: WHERE $nn(V(nn(\Delta T))\ .ps_m)$AND

16: NOT EXISTS(SELECT 1FROM $\overline{V}$

WHERE $\overline{V}$.$ps_m = V(nn(\Delta T))$ .ps_m)

[0117]

17:) AS $\overline{Null(\Delta T,T,V)}$

18: , (SELECT FIRST 1FROM (DELETE $\overline{V}$

19: FROM $V(nn(\Delta T))$, LATERAL (

20: {¬T- $\mathcal{N}$*ull-supplied* : $\overline{V(nn(\Delta T))}$ to be deleted}

21: SELECT v_1 (V(nn(ΔT)),…., v_k(V(nn(ΔT)))) AS $\overline{V(nn(\Delta T))}$

22: WHERE $\overline{V}$.*ui* IS NOT DISTINCT FROM $\overline{V(nn(\Delta T))}$.*ui*

23:)REFERENCING ( ))AS *ZZ*

[0118] 更新操作(表9)

[0119] 依照本发明的一个实施例,为了更新操作,使用MERGE语句。在一个实施例中,与删除操作一样,应用V(nn(ΔT))后,集合Null(ΔT,T.V)的计算必须仅见到视图数据,使得元组Null(i,t,T,V)仅当NS-补偿操作需要时才产生。因此,嵌入式的合并语句(表9,第

21至32行)首先计算(表9,第22行)并应用(表9,第28-30行)V(nn(ΔT))。当用于NS-补偿的条件被检查时(表9,第10行和16行)时,MERGE语句将见到视图V的修改数据。通过WHEN [NOT]MATCHED子句(表9,第35-37行)处理集合Null(ΔT,T,V)。

[0120] 表9:用于更新操作的合并语句

1:MERGE INTO $\bar{V}$ USING

[0121] 2:(WITH V*(nn(ΔT)) AS*

3:(SELECT *, Δ*T.DML_type* FROM *V(nn(ΔT))*)

[0122]

4:SELECT $v_1(DT),....,v_k(DT)$, *DT.DML_type* FROM (

5:SELECT DISTINCT *DT _nulls.* * FROM $V(nn(\Delta T))$, LATERAL (

/* T-*null-supplied* : $\overline{Null\ (1,t,T,V)}$ for *NS*-compensation(insert) */

6:SELECT v_1 ($\mathcal{N}ull\ (1, t, T, V)),....,v_k\ \mathcal{N}ull\ (1, t, T, V)$

7:V$(nn(\Delta T))$ .DML_type * (–1)AS DML_*type* $\mathcal{N}ull$ AS *d_rowid*

8:WHERE $nn(V(nn(\Delta T))\ .ps_1$ AND DML_*type* = +1 AND

9:NOT EXISTS (SELECT 1 FROM $\overline{V}$ WHERE $\overline{V}\ .ps_1 = V(nn(\Delta T))\ .ps_1$)

10: UNION ALL

/* T- $\mathcal{N}ull$-*supplied* : $\overline{Null\ (1,t,T,V)}$ for *NS*-compensation(delete) */

11: SELECT $\mathcal{N}ull$ AS $v_1,...., \mathcal{N}ull$ AS, v_k

12: $V(nn(\Delta T))$ .DML_type * (–1)AS DML_*type,rowid*($\overline{V}$) AS d_*rowid*

13: FROM $\overline{V}$ WHERE DML_*type* = -1 AND

14: $\overline{V}\ .ps_1 = V(nn(\Delta T))\ .ps_1$ AND $n(\overline{V}\ .ns_1)$

15: UNION ALL

16: . . .

17:) AS *DT_nulls*

18:) AS *DT*

19: , (SELECT FIRST 1 FROM (MERGE INTO $\overline{V}$ USING (

20: *select* $v_1(DT),....,v_k(DT)$, *DT.DML_type* FROM (

/* ¬T- $\mathcal{N}ull$-*supplied* : $\overline{V(nn(\Delta T))}$ to be *updated, inserted or deleted* */

21: SELECT * FROM $V(nn(\Delta T))$

22:)AS DT

23:)AS $\overline{V(nn(\Delta T))}$

24: ON $\overline{V}\ .ui$ IS NOT DISTINCT FROM $\overline{V(nn(\Delta T))}\ .ui$

25: WHEN MATCHED AND DML_*type*= +1 THEN

UPDATE

26: WHEN MATCHED AND DML_*type*= *-1* THEN)

DELETE

27: WHEN NOT MATCHED AND DML_*type*= *+1* THEN

INSERT

[0123] 28:)REFERENCING( )) AS ZZ

29:) AS $\overline{Null\ (\Delta T,T,V)}$

30: ON DML_*type* = *-1* AND *rowid* $\overline{V}$ = $\overline{Null\ (1,t,T,V)}$.*d_rowid*

31: WHEN MATCHED THEN DELETE

32. WHEN NOT MATCHED AND DML_*type* = *+1*

33: THEN INSERT

[0124] 立即实体化视图维护方法

[0125] 按照本发明的实施例,图4-6是说明创建立即实体化视图(iMVs)的步骤的流程图。

[0126] 根据本发明的一个实施例,图4是流程图400,其说明更新立即实体化视图(iMV)的步骤。

[0127] 更特别地,根据本发明的一个实施例,流程图400说明了执行基于对基表的更新的iMV更新的步骤。

[0128] 方法从步骤402开始并进行到步骤404,在步骤404,基于基表T,执行更新语句。在本发明的一个实施例中,T是关系数据库中的数据库表。在执行更新语句后,方法进行至步骤406。

[0129] 在步骤406中,ΔT或指示改变基表T的Δ被计算。在一个实施例中,该步骤通过比较在步骤404中被更新的基表T中的被更新的行的之前和之后的图片而被执行。在ΔT被计算后,该方法进行至步骤408。

[0130] 在步骤408中,ΔiMV被计算。根据一个实施例,基于在步骤406中计算的ΔT,ΔiMV被计算。在ΔiMV被计算后,方法进行至步骤410。

[0131] 在步骤410中,iMV视图被更新以及控制传递到方法结束处的步骤412。

[0132] 依照本发明的一个实施例,使用ΔiMV执行该步骤从而反映在步骤406中计算的ΔT变化。

[0133] 依照本发明的一个实施例,图5是阐释了步骤的流程图500,通过该步骤说明为了立即实化视图(iMV)执行触发器的步骤。

[0134] 更特别地,流程图500阐释了步骤,通过该步骤,创建用于更新引用基表的每个iMV的iMV触发器,并执行。

[0135] 方法在步骤502开始并进行到步骤504,在步骤504,基表T被更新。根据本发明的一个示例性实施例,T是关系数据库中的数据库表。在基表T被更新后,方法进行到步骤506。

[0136] 在步骤506中,针对在步骤504中更新的基表T是否存在iMV触发器,进行评估。如果

确定iMV触发器存在，那么控制传递至步骤510。如果确定iMV触发器不存在，那么控制传递至步骤508，在步骤508处，创建iMV触发器。

[0137] 在步骤508中，在创建iMV触发器后，其对应于步骤504中更新的基表T，方法进行至步骤510。

[0138] 在步骤510中，执行iMV触发器从而更新引用基表T的每个iMV，以及控制传递到步骤512，在步骤512处方法结束。

[0139] 根据本发明一个实施例，图6是阐释步骤的流程图600，通过该步骤为立即实体化视图(iMV)保存触发器。

[0140] 更特别地，流程图600阐释了根据本发明的一个实施例的步骤，通过该步骤生成iMV触发器并为基表更新而保存。

[0141] 方法在步骤602开始并进行到步骤604，在步骤604处，所有引用基表T的iMV被识别。根据本发明的一个示例性实施例，T是关系数据库中的数据库表。在所有iMV被识别后，方法进行到步骤606。

[0142] 在步骤606中，对于在步骤604中识别的每个iMV，生成更新语句。在一个实施例中，通过产生MERGE SQL语句从而将ΔiMV合并入在步骤606中识别的iMV。在产生更新语句后，方法进行至步骤608。

[0143] 在步骤608中，为基表T更新保存触发器，iMV(T)，并且控制传递至步骤610。在步骤610处方法结束。

[0144] 图8是阐释根据本发明一个实施例的步骤的流程图800，通过该步骤，在基表T更新后，为iMV产生SQL更新语句。

[0145] 更特别地，根据本发明的一个实施例，流程图800阐释了步骤，通过该步骤，通过来自注释PSNS标准化连接运算符树的ΔT，在更新基表T后，产生用于iMV的SQL更新语句。

[0146] 方法从步骤802开始并进行到步骤804，在步骤804处，确定iMV是否引用基表T。根据本发明的示例性实施例，T是关系数据库中的数据库表。在iMV被识别后，方法进行到步骤806。

[0147] 在步骤804中，鉴于iMV是否引用基表T进行评估。如果确定iMV引用了基表T，控制传递至步骤808。如果确定iMV没有引用基表T，那么控制传递至步骤806，在步骤806处，方法结束。

[0148] 在步骤808中，基于引用基表T的iMV定义，建立标准化连接运算符树 $n\mathcal{T}(\mathrm{iMV})$。在建立标准化连接运算符树 $n\mathcal{T}$ 后，方法进行至步骤810。

[0149] 在步骤810中，鉴于在步骤808中创建的标准化连接运算符树 $n\mathcal{T}(\mathrm{iMV})$ 是否满足谓语条件而进行评估。如果确定标准化连接运算符树满足谓语条件，控制传递至步骤814。如果确定标准化连接运算符树不满足谓语条件，那么控制传递至步骤812，在步骤812处，程序结束。

[0150] 在步骤814中，执行标准化连接运算符树的PSNS注释创建 $\mathrm{PSNS}\text{-}n\mathcal{T}(\mathrm{iMV})$。在标准化运算符连接树被注释后，控制传递至步骤816。

[0151] 在步骤816中，针对在步骤814中注释的标准化连接运算符树 $\mathrm{PSNS}\text{-}n\mathcal{T}(\mathrm{iMV})$ 是

否满足选择列表条件而进行评估。如果确定被注解的标准化连接运算符树满足选择列表条件，那么控制传递至步骤820。如果确定被注解的标准化连接运算符树不满足选择列表条件，那么控制传递至步骤818，在步骤818处，程序结束。

[0152] 在步骤820中，产生用于iMV的SQL更新语句，通过来自PSNS-$n\mathcal{J}$(iMV)的ΔT，在基表T更新后，使用SQL更新语句，并且控制传递至步骤822，在步骤822处，方法结束。

[0153] 图9是阐释根据本发明一个实施例的步骤的流程图900，通过该步骤标准化逻辑连接运算符树。

[0154] 更特别地，流程图900阐释了根据本发明的一个实施例的步骤，通过该步骤，基于关系类型以及用于树的节点连接，逻辑运算符树被标准化。

[0155] 方法在步骤902开始，在步骤902处，运算符树T，连接节点n以及标准化连接nJ被读为输入。在树被读取并解析后，方法进行至步骤904，在步骤904处，进行确定树T中的节点n代表何种类型的操作。在一个实施例中，树T是逻辑运算符树，其为内部节点n是连接和叶，是关系的查询表示。在每个实施例中，逻辑运算符树T中的每个节点具有引用自左子树和右子树中的关系的谓语。

[0156] 在步骤904中，针对节点n是何种类型进行评估。如果n是代表关系的树T的叶，那么控制传递至步骤906，在步骤906处，关系信息传递至步骤908。如果n是代表内部连接的节点，那么控制传递至步骤910，在步骤910处，连接信息传递至步骤912。

[0157] 如果n是代表左外连接的节点，那么控制传递至步骤916，在步骤916处，左外连接信息传递至步骤914。如果n是代表全部外连接的节点，那么控制传递至步骤920，在步骤920处，全部外连接信息传递至步骤918。

[0158] 在步骤908中，基于步骤906中传递的信息，标准化连接nJ随着新节点n而增大。在一个实施例中，用于在该步骤中执行的函数的逻辑表达为：

[0159] nJ.p=nJ.p AND p//增加节点n的谓语至标准化连接nJ

[0160] nJ.ProperRerlations U={T}//关系T是标准化连接nJ的部分

[0161] T.Parent=nJ//关系T具有标准化连接nJ的直接父类部分

[0162] 在执行步骤908的函数后，方法进行至步骤922。

[0163] 在步骤912中，基于步骤910中传递的连接信息，标准化连接nJ随着新节点n而增大。在一个实施例中，用于在该步骤中执行的函数的逻辑表达为：

[0164] nJ.p=nJ.p AND p//增加节点n的谓语至标准化连接nJ

[0165] CALL nT(T,LEFT CHILD n,nJ)//为了建立标准化连接运算符树，横过左子类

[0166] CALL nT(T,RIGHT CHILD n,nJ)//为了建立标准化连接运算符树，横过右子类

[0167] 在执行步骤912的函数后，方法进行至步骤922。

[0168] 在步骤914中，基于步骤916中传递的外连接信息，标准化连接nJ随着新外连接n而增大。在一个实施例中，用于在该步骤中执行的函数的逻辑表达为：

[0169] CALL nT(T,RIGHT CHILD n,nJ)//为了建立标准化连接运算符树，穿过右子类

[0170] nJL=NEW JOIN//创建新标准化连接，其代表外连接n的空-导入侧

[0171] nJ_L.type=LEFT OUTERJOIN

[0172] nJ_L.p=p//ON谓语是新标准化连接nJ_L的谓语

[0173] nJ_L.Parent=nJ//新标准化连接nJ_L具有作为直接父类的输入标准化连接nJ

[0174] nJ.nJoins U={nJ_L}//nJ是新标准化连接nJ_L的直接父类

[0175] CALL nT(T,LEFT CHILD n,nJL)//通过横过外连接n的空-导入侧建立新标准化连接nJ_L

[0176] 在执行步骤914的函数后，方法进行至步骤922。

[0177] 在步骤918中，基于步骤920中传递的全部外连接信息，标准化连接nJ随着新外连接n而增大。在一个实施例中，用于在该步骤中执行的函数的逻辑表达为：

[0178] nJ_F=NEW JOIN//创建类型全部外连接的新标准化连接nJ_F

[0179] nJ_F.type=FULL OUTERJOIN

[0180] nJ_F.Parent=nJ//新标准化连接具有直接父类nJ

[0181] nJ_F.p=p//新标准化连接具有作为谓语的连接n的ON谓语

[0182] nJ.nJoins U={nJ_F}//新标准化连接具有直接父类nJ

[0183] nJ_L=NEW JOIN//创建新标准化连接nJ_L，其代表连接n的左子树

[0184] nJ_L.type=INNER OUTERJOIN

[0185] nJ_L.Parent=nJ_F//新标准化连接nJ_L具有父类nJ_F

[0186] nJ_R=NEW JOIN//创建新标准化连接nJ_R，其代表连接n的右子树

[0187] nJ_R.type=INNER OUTERJOIN

[0188] nJ_R.Parent=nJF//新标准化连接nJ_R具有父类nJ_F

[0189] nJ_F.nJoins U={nJ_L,nJ_R}//标准化连接nJ_F具有两个直接子类nJ_L和nJ_R

[0190] CALL nT(T,LEFT CHILD n,nJ_L)//通过横过全部外连接n的空-导入侧中的一个而建立新标准化连接nJ_L

[0191] CALL nT(T,RIGHT CHILD n,nJ_R)//通过横过全部外连接n的空-导入侧中的一个而建立新标准化连接nJ_R

[0192] 在执行步骤914的函数后，方法进行至步骤922。

[0193] 在步骤922中，对关系(p)中的所有关系T执行函数，此处p是节点n的谓语。在一个实施例中，用于在步骤922中执行的函数的逻辑表达为：

[0194] FOR ALL T IN Relations(p)

[0195] IF T.Parent!=nJ,nJ_F,nJ_L,nJ_R

[0196] THEN p is NS-intolerant

[0197] 在执行922的函数后，方法结束。

[0198] 立即实体化视图系统

[0199] 图7是系统700的例子，在系统700中可以实施上述算法和方法。示例性系统700包括具有数据库706的服务器704。在一个实施例中，服务器704包括关系数据库管理系统(RDBMS)并且数据库706是关系数据库。在系统的实施例中，多个基表T以及立即实体化视图(iMV_s)被存储并在数据库706中维护。虽然在系统700中描述了单个客户702、服务器704以及数据库706，可以理解的是多个客户702和服务器704可以访问多个数据库706。

[0200] 在图7中描述的示例性系统700中，客户702通过服务器704访问驻留在数据库706中的基表以及iMV_s。如相关领域的技术人员所理解的，其他客户702，例如但不限于，移动客户、笔记本电脑和移动电话，可被用于访问服务器704和数据库706。在一个实施例中，数据库706是内存数据库，该内存数据库实施无须存储数据文件或数据，所述数据文件或数据来

自服务器706的磁盘上的多个数据源(未显示)。

[0201] 计算机系统实施的示例

[0202] 本发明的各个方面可以通过软件、硬件、固件或其组合实施。图10说明了计算机系统1000的一个示例,其中本发明或其部分可以作为计算机-可读代码实施。例如,通过图4-6、8和9中的流程图400、500、600、800和900说明该方法,该方法可以在系统1000中实施。而且,例如,图7中描述的系统700的组件可以在系统1000中实施。发明的各个实施例按照该示例计算机系统1000而描述。在阅读该说明书后,对相关领域技术人员而言,如何使用其它计算机系统和/或计算机架构来实施本发明是显而易见的。

[0203] 计算机系统1000包括一个或多个处理器,例如处理器1004。处理器1004可以是特殊用途处理器或通用处理器。处理器1004被连接到通信基础设施1006(例如,总线,或网络)。

[0204] 计算机系统1000还包括主存储器1008,优选随机存取存储器(RAM),并且还可以包括辅助存储器1010。辅助存储器1010可以包括,例如,硬盘驱动器1012,可移动的存储驱动器1014,闪速存储器,记忆棒,和/或任何类似的非易失性的存储机制。可移动存储驱动器1014可以包括软盘驱动器,磁带驱动器,光盘驱动器,闪速存储器等。通过公知的方式,可移动存储驱动器1014从可移动存储单元1018中读取和/或写入到可移动存储单元1018。可移动存储单元1018可以包括软盘,磁带,光盘等,其由可移动存储驱动器1014读写。相关领域的技术人员同样可以理解的,可移动存储单元1018包括计算机可用存储介质,所述计算机可用存储介质具有存储在其中的计算机软件和/或数据。

[0205] 在可选的实施方式中,辅助存储器1010可以包括其他类似的装珞,用于允许计算机程序或者其它指令被装载到计算机系统1000。这种装珞可以包括,例如,可移动存储单元1022和接口1020。这种装珞的例子可包括程序盒式存储器和盒式接口(例如,视频游戏设备中发现的),可移动存储芯片(如EPROM,或者PROM)以及相关的插槽,以及允许软件和数据从可移动存储单元1022传送到计算机系统1000的其它可移动存储单元1022和接口1020。

[0206] 计算机系统1000也可以包括通信接口1024。通信接口1024允许软件和数据在计算机系统1000和外部设备之间进行传输。通信接口1024可包括调制解调器,网络接口(如以太网卡),通信端口,PCMCIA槽和卡等。通过通信接口1024传输的软件和数据是信号的形式,其可以是电子的、电磁的、光学的、或其他能够由通信接口1024接收的信号。通过通信路径1026,这些信号被提供到通信接口1024。通信路径1026携带信号,并且可以使用电线或电缆,光纤,电话线,移动电话链路,RF链路或其他通信信道来实现。

[0207] 计算机系统1000另外包括计算机显示屏1030。根据一个实施例,计算机显示器1030,连同显示器接口1002,可以用于显示用户接口(UI)或在客户端702或服务器704处的数据库管理员(DBA)控制台。

[0208] 在本文档中,术语“计算机程序介质”和“计算机可用介质”一般指代,例如可移动存储单元1018,可移动存储单元1022,和安装在硬盘驱动器1012中的硬盘。通过通信路径1026携带的信号也可以体现本文所描述的逻辑。计算机程序介质和计算机可用介质,也可以涉及存储器,诸如主存储器1008和辅助存储器1010,它可以是存储器半导体(如DRAMs等)。这些计算机程序产品是用于将软件提供给计算机系统1000的装珞。

[0209] 计算机程序(也称为计算机控制逻辑)存储在主存储器1008和/或辅助存储器1010

中。计算机程序也可以通过通信接口1024接收。这样的计算机程序，当被执行时，使计算机系统1000执行如本文中所论述的本发明。特别地，该计算机程序，当被执行时，使处理器1004来实现本发明的方法，如由图4-6、8和9中的流程图400，500，600，800，和900中所示的方法中的步骤。因此，这样的计算机程序代表计算机系统1000的控制器。使用软件实施本发明，该软件可以存储在计算机程序产品中并使用可移动存储驱动器1014，接口1020，硬盘驱动器1012，或者通信接口1024加载到计算机系统1000。

[0210] 本发明还涉及计算机程序产品，其包括存储在任何计算机可用介质上的软件。这样的软件，当在一个或多个数据处理装臵中执行时，使数据处理装臵以如本文所述的方式操作。本发明的实施例采用现在已知的或在将来已知的任何计算机可用或可读介质。计算机可用介质的实施例包括，但不限于，主存储设备(例如，任何类型的随机存取存储器)，辅助存储设备(例如，硬盘驱动器，软盘，CD ROMS，ZIP磁盘，磁带，磁存储设备，光学存储设备，MEMS，纳米技术存储设备等)，和通信介质(例如，有线和无线通信网络，局域网，广域网，内联网等)。

[0211] 结论

[0212] 虽然上面已经描述了本发明的各种实施例，但应当理解，仅作为示例而非限制性地表示这些实施例。相关领域的技术人员可以理解，在不脱离随附权利要求书中所定义的本发明的精神和范围的情况下，可以进行形式和细节上的各种改变。但应当理解，本发明并不限于这些实施例。本发明适用于如在此描述的操作的任何元件。因此，本发明的广度和范围不应该限于任何上述示例性实施例，而是仅应该按照下面的权利要求及其等同物限定。

100

```
SELECT [PR9].[ContactID] AS [ContactID],[PR9].[C1] AS [C1],[PR9].[C2] AS [C2],[PR9].[ContactID1] AS [ContactID1],
       [PR9].[SalesOrderID] AS [SalesOrderID], [PR9].[TotalDue] AS [TotalDue]
FROM
(SELECT [Distinct1].[ContactID] AS [ContactID],1 AS [C1],[PR8].[ContactID] AS [ContactID1],
       [PR6].[SalesOrderID] AS [SalesOrderID],[PR6].[TotalDue] AS [TotalDue],[PR8].[C1] AS [C2]
  FROM (SELECT DISTINCT [EX1].[ContactID] AS [ContactID]
    FROM [Contact] AS [EX1] JOIN [SalesOrderHeader] AS [EX2]
     ON EXISTS (SELECT cast(1 as bit) AS [C1]
       FROM (SELECT cast(1 as bit) AS X ) AS [SingleRowTable]
       LEFT OUTER JOIN (SELECT [EX3].[ContactID] AS [ContactID]
         FROM [Contact] AS [EX3] WHERE [EX2].[ContactID] = [EX3].[ContactID] ) AS [PR1]
           ON cast(1 as bit) = cast(1 as bit)
       LEFT OUTER JOIN (SELECT [EX4].[ContactID] AS [ContactID]
         FROM [Contact] AS [EX4] WHERE [EX2].[ContactID] = [EX4].[ContactID] ) AS [PR2]
           ON cast(1 as bit) = cast(1 as bit)
       WHERE ([EX1].[ContactID] = [PR1].[ContactID])
         OR (([EX1].[ContactID] IS NULL) AND ([PR2].[ContactID] IS NULL)))
  ) AS [Distinct1]
LEFT OUTER JOIN
(SELECT [EX5].[ContactID] AS [ContactID],[EX6].[SalesOrderID] AS [SalesOrderID],[EX6].[TotalDue] AS [TotalDue],1 AS [
    FROM [Contact] AS [EX5]
    JOIN [SalesOrderHeader] AS [EX6]
      ON EXISTS (SELECT cast(1 as bit) AS [C1]
        FROM (SELECT cast(1 as bit) AS X ) AS [SingleRowTable2]
        LEFT OUTER JOIN (SELECT [EX7].[ContactID] AS [ContactID]
          FROM [Contact] AS [EX7] WHERE [EX6].[ContactID] = [EX7].[ContactID] )AS [PR5]
            ON cast(1 as bit) = cast(1 as bit)
        LEFT OUTER JOIN (SELECT [EX8].[ContactID] AS [ContactID]
          FROM [Contact] AS [EX8] WHERE [EX6].[ContactID] = [EX8].[ContactID] )AS [PR6]
            ON cast(1 as bit) = cast(1 as bit)
        WHERE ([EX5].[ContactID] = [PR5].[ContactID])
          OR (([EX5].[ContactID] IS NULL) AND ([PR6].[ContactID] IS NULL))
    ) ) AS [PR8]
    ON ([PR8].[ContactID] = [Distinct1].[ContactID])
      OR (([PR8].[ContactID] IS NULL) AND ([Distinct1].[ContactID] IS NULL))
  ) AS [PR9]
ORDER BY [PR9].[ContactID] ASC, [PR9].[C2] ASC
```

图1

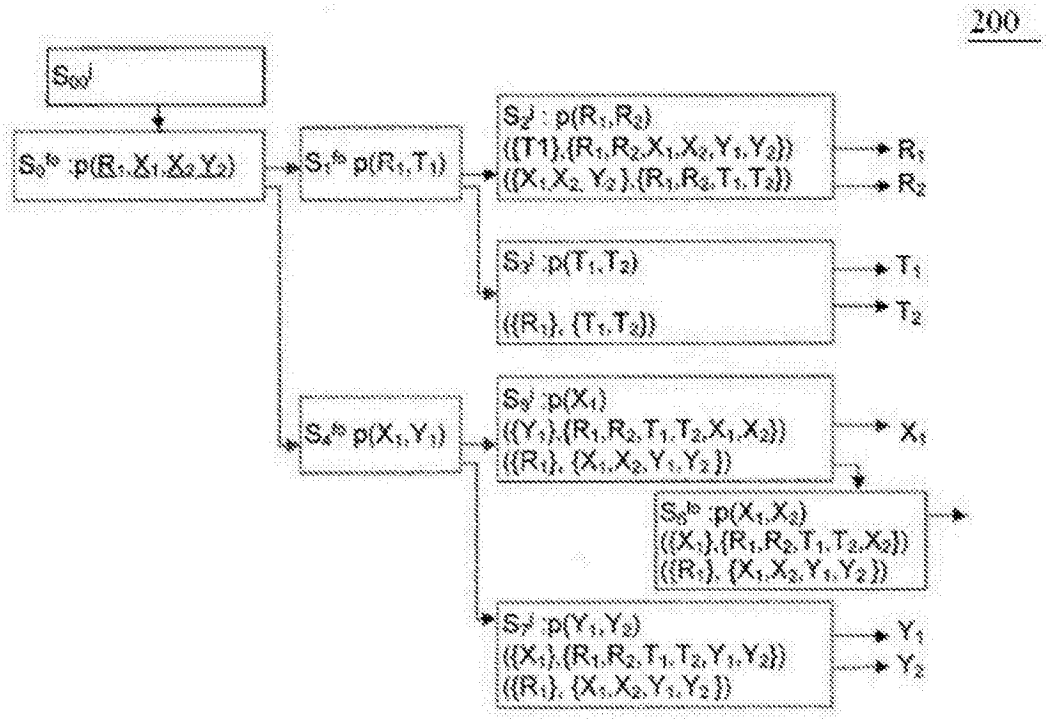

200
S_{00}
S_0 : $p(R_1,R_2,X_1,X_2,Y_1,Y_2)$
S_1 : $p(R_1,T_1)$
S_2 : $p(R_1,R_2)$
$((T1),(R_1,R_2,X_1,X_2,Y_1,Y_2))$
$((X_1,X_2,Y_1),(R_1,R_2,T_1,T_2))$
R_1
R_2
S_3 : $p(T_1,T_2)$
$((R_1),(T_1,T_2))$
T_1
T_2
S_4 : $p(X_1,Y_1)$
S_5 : $p(X_1)$
$((Y_1),(R_1,R_2,T_1,T_2,X_1,X_2))$
$((R_1),(X_1,X_2,Y_1,Y_2))$
X_1
S_6 : $p(X_1,X_2)$
$((X_1),(R_1,R_2,T_1,T_2,X_2))$
$((R_1),(X_1,X_2,Y_1,Y_2))$
S_7 : $p(Y_1,Y_2)$
$((X_1),(R_1,R_2,T_1,T_2,Y_1,Y_2))$
$((R_1),(X_1,X_2,Y_1,Y_2))$
Y_1
Y_2

图2

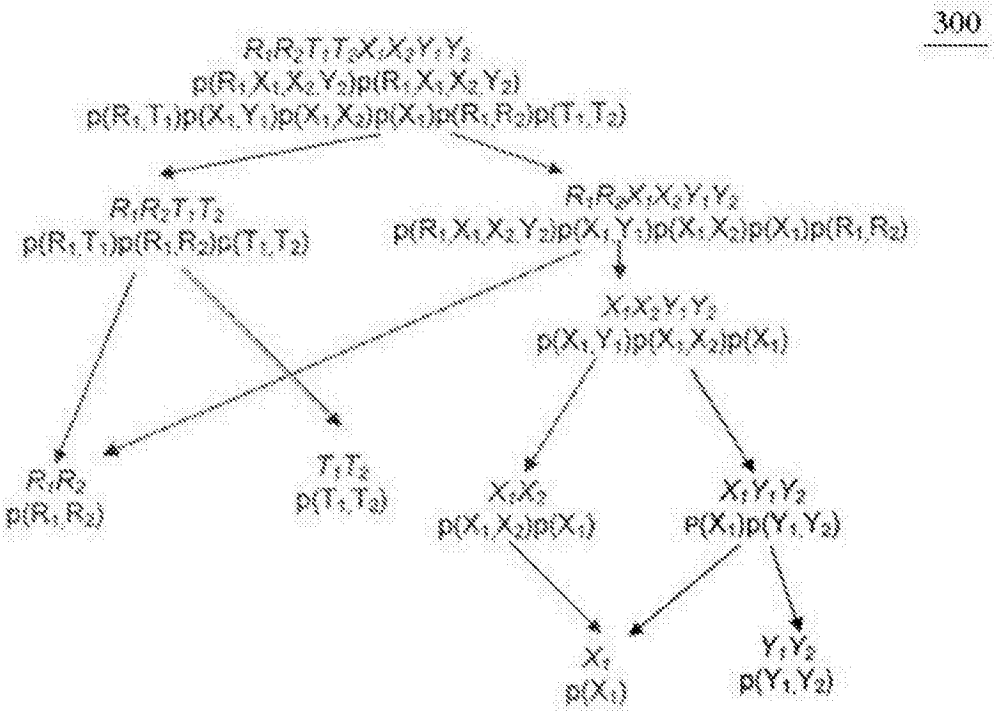

300
$R_1R_2T_1T_2X_1X_2Y_1Y_2$
$p(R_1,X_1,X_2,Y_2)p(R_1,X_1,X_2,Y_2)$
$p(R_1,T_1)p(X_1,Y_1)p(X_1,X_2)p(X_1)p(R_1,R_2)p(T_1,T_2)$
$R_1R_2T_1T_2$
$p(R_1,T_1)p(R_1,R_2)p(T_1,T_2)$
$R_1R_2X_1X_2Y_1Y_2$
$p(R_1,X_1,X_2,Y_2)p(X_1,Y_1)p(X_1,X_2)p(X_1)p(R_1,R_2)$
$X_1X_2Y_1Y_2$
$p(X_1,Y_1)p(X_1,X_2)p(X_1)$
R_1R_2
$p(R_1,R_2)$
T_1T_2
$p(T_1,T_2)$
X_1X_2
$p(X_1,X_2)p(X_1)$
$X_1Y_1Y_2$
$p(X_1)p(Y_1,Y_2)$
X_1
$p(X_1)$
Y_1Y_2
$p(Y_1,Y_2)$

图3

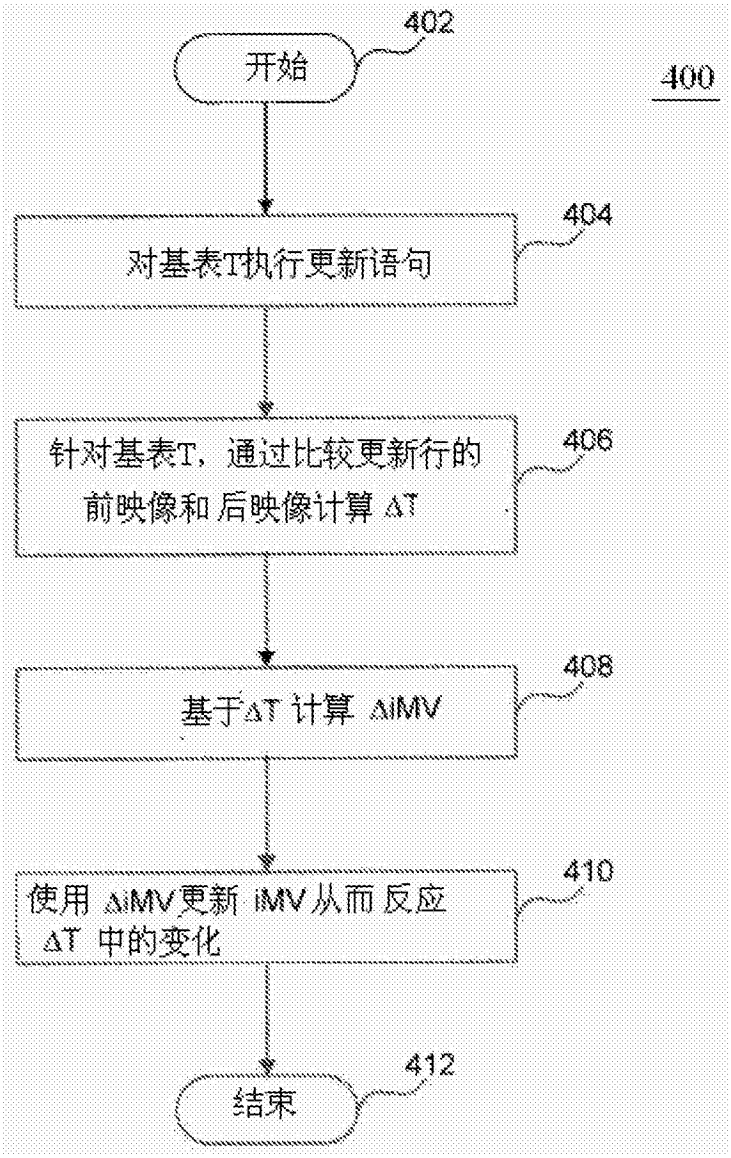

402
开始
400
404
对基表T执行更新语句
406
针对基表T，通过比较更新行的前映像和后映像计算 ΔT
408
基于ΔT 计算 ΔiMV
410
使用 ΔiMV更新 iMV从而反应 ΔT 中的变化
412
结束

图4

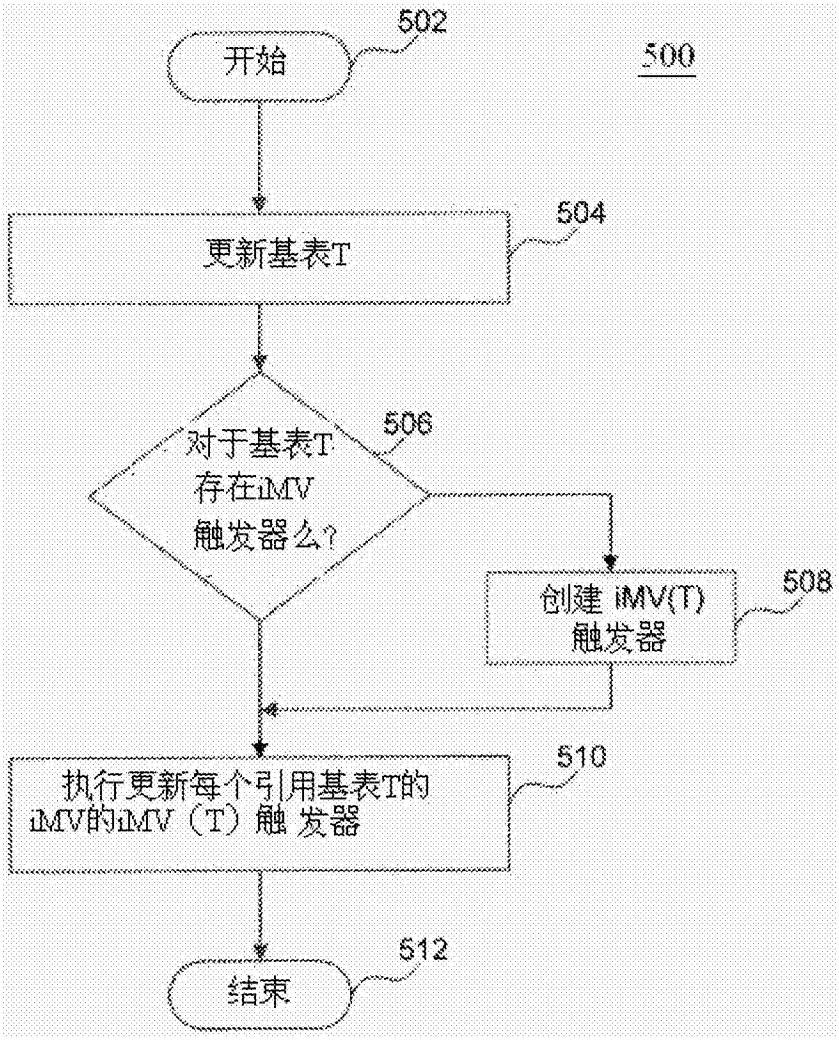

500
502
开始
504
更新基表T
506
对于基表T
存在iMV
触发器么?
508
创建 iMV(T)
触发器
510
执行更新每个引用基表T的
iMV的iMV（T）触发器
512
结束

图5

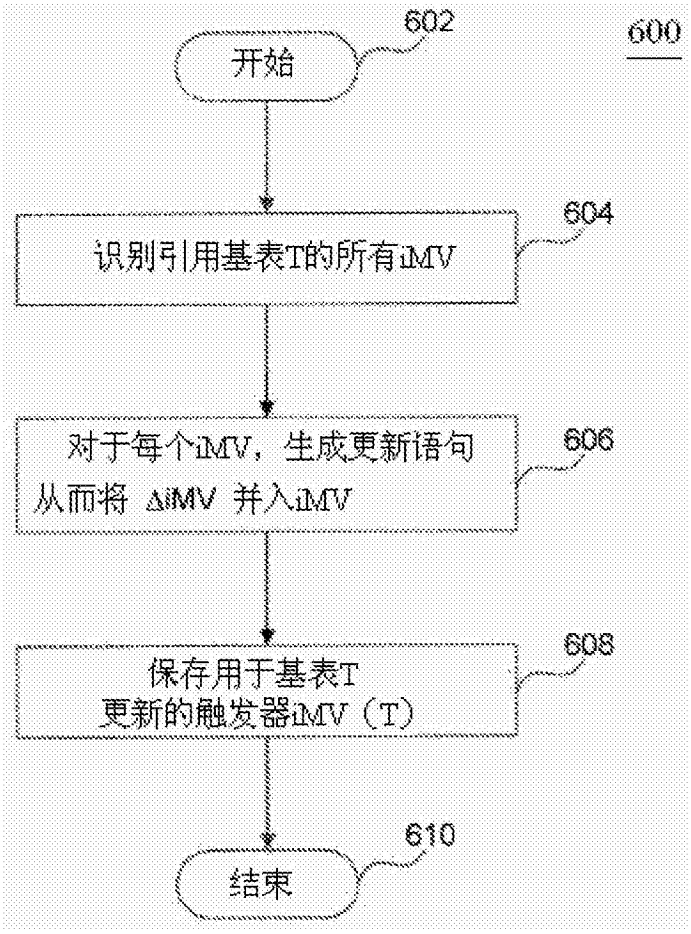

600
602
开始
604
识别引用基表T的所有iMV
606
对于每个iMV，生成更新语句
从而将 ΔiMV 并入iMV
608
保存用于基表T
更新的触发器iMV（T）
610
结束

图6

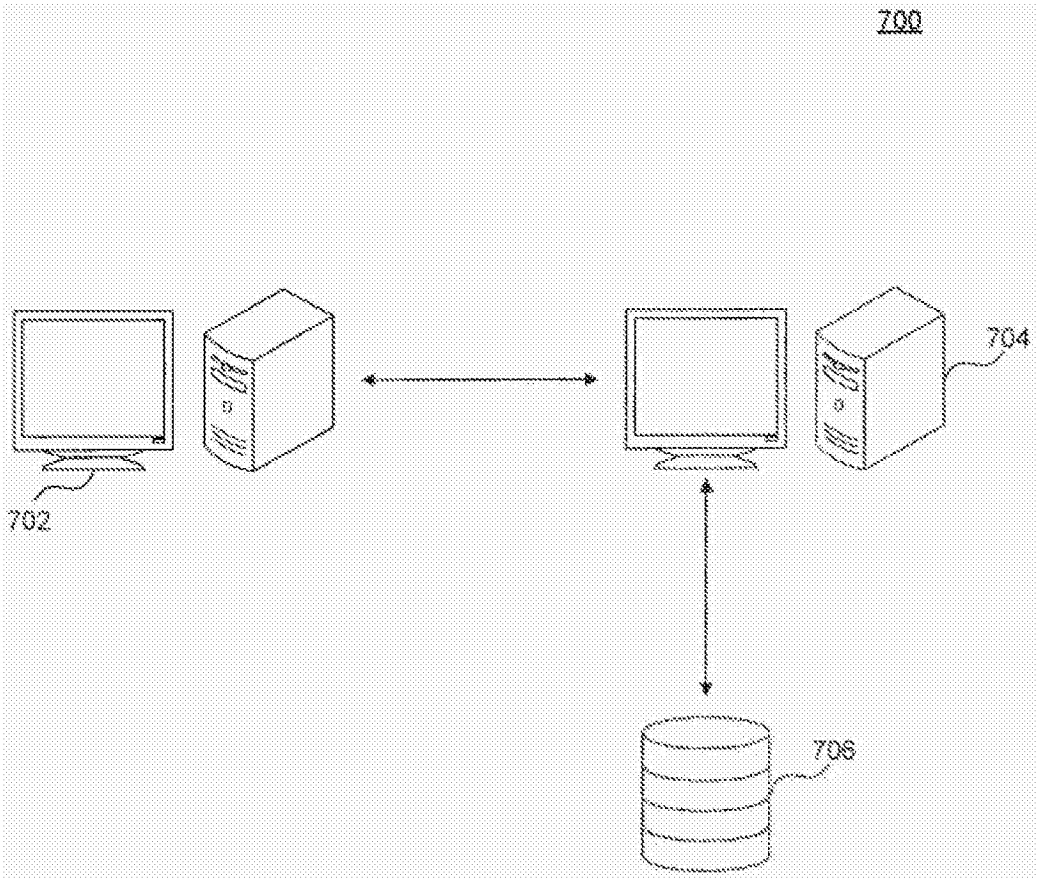


图7

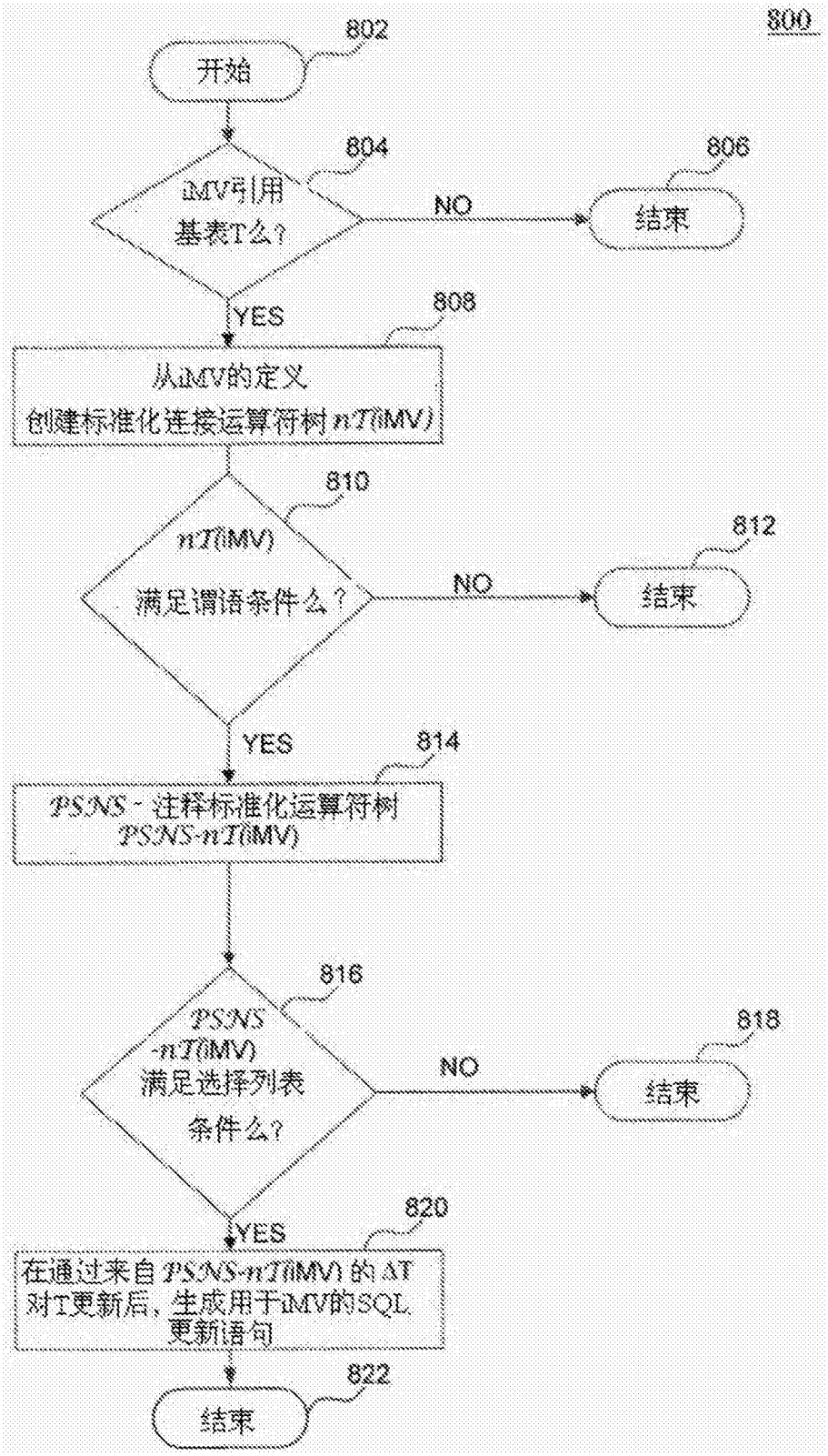

800
802
开始
804
MV引用
基表T么?
NO
806
结束
YES
808
从MV的定义
创建标准化连接运算符树 nT(MV)
810
nT(MV)
满足谓语条件么?
NO
812
结束
YES
814
PSNS - 注释标准化运算符树
PSNS-nT(MV)
816
PSNS
-nT(MV)
满足选择列表
条件么?
NO
818
结束
YES
820
在通过来自 PSNS-nT(MV) 的 AT
对T更新后,生成用于MV的SQL
更新语句
822
结束

图8

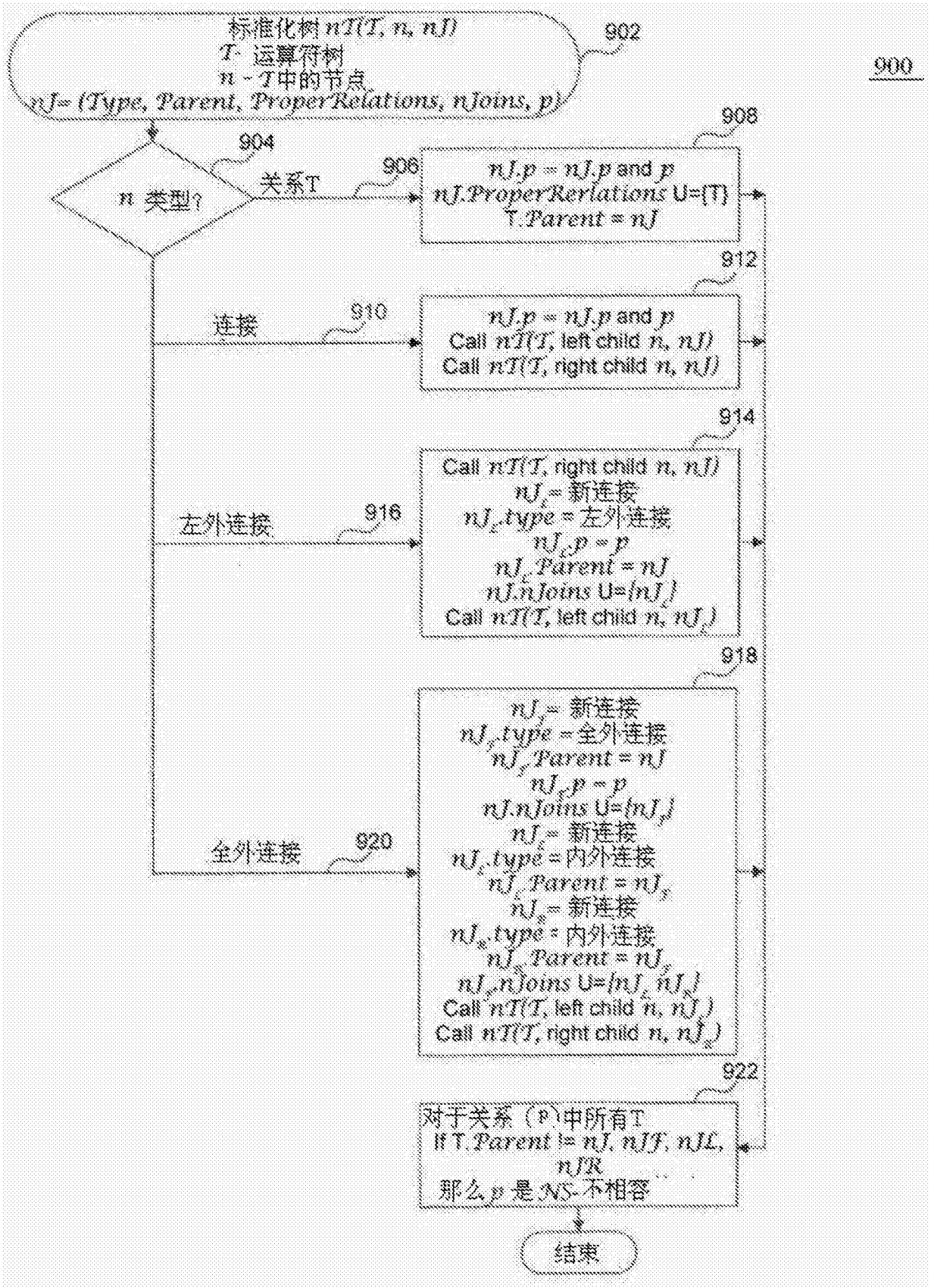

标准化树 nT(T, n, nJ)
T- 运算符树
n - T中的节点
nJ= (Type, Parent, ProperRelations, nJoins, p)
902
900
904
n 类型?
关系T
906
nJ.p = nJ.p and p
nJ.ProperRerlations U={T}
T.Parent = nJ
908
912
连接
910
nJ.p = nJ.p and p
Call nT(T, left child n, nJ)
Call nT(T, right child n, nJ)
914
左外连接
916
Call nT(T, right child n, nJ)
nJc= 新连接
nJc.type = 左外连接
nJc.p = p
nJc.Parent = nJ
nJ.nJoins U={nJc}
Call nT(T, left child n, nJc)
918
全外连接
920
nJc= 新连接
nJc.type = 全外连接
nJc.Parent = nJ
nJc.p = p
nJ.nJoins U={nJc}
nJL= 新连接
nJL.type = 内外连接
nJL.Parent = nJc
nJR= 新连接
nJR.type = 内外连接
nJR.Parent = nJc
nJc.nJoins U={nJL, nJR}
Call nT(T, left child n, nJL)
Call nT(T, right child n, nJR)
922
对于关系（P）中所有T
If T.Parent != nJ, nJF, nJL, nJR
那么p 是NS-不相容
结束

图9

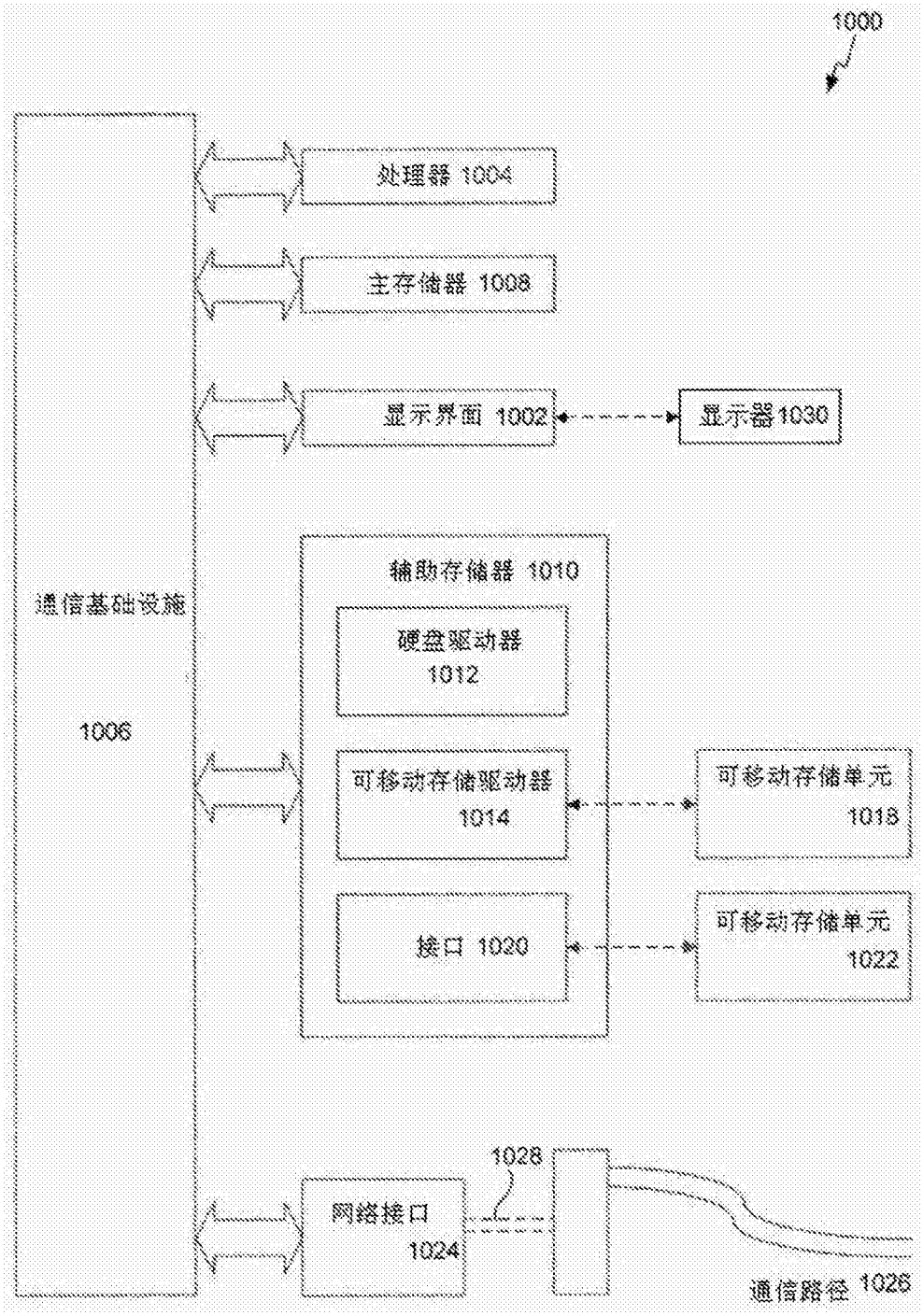

1000
处理器 1004
主存储器 1008
显示界面 1002
显示器1030
辅助存储器 1010
硬盘驱动器
1012
通信基础设施
1006
可移动存储驱动器
1014
可移动存储单元
1018
接口 1020
可移动存储单元
1022
1028
网络接口
1024
通信路径 1026

图10