(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 101176090 B
(45)授权公告日 2010.09.01

(21)申请号 200680008818.1
(22)申请日 2006.02.21
(30)优先权数据
11/084,855 2005.03.18 US
(85)PCT申请进入国家阶段日
2007.09.18
(86)PCT申请的申请数据
PCT/US2006/005702 2006.02.21
(87)PCT申请的公布数据
WO2006/101633 EN 2006.09.28
(73)专利权人 金门软件公司
地址 美国加利福尼亚
(72)发明人 埃里克·伊恩·菲什
斯科特·罗杰·科尔宾
乔尔·谢泼德 乔治·艾伦·皮尔逊
蒂莫西·李·拉思本
(74)专利代理机构 中国国际贸易促进委员会专利商标事务所 11038
代理人 康建峰
(51)Int.Cl.
G06F 17/00(2006.01)
(56)对比文件
US 2002059299 A1,2002.05.16, 全文.
US 2003231647 A1,2003.12.18, 全文.
WO 0217134 A1,2002.02.28, 全文.
US 6449622 B1,2002.09.10, 全文.
US 6321236 B1,2001.11.20, 全文.
US 5884325 A,1999.03.16, 全文.
审查员 吴广平

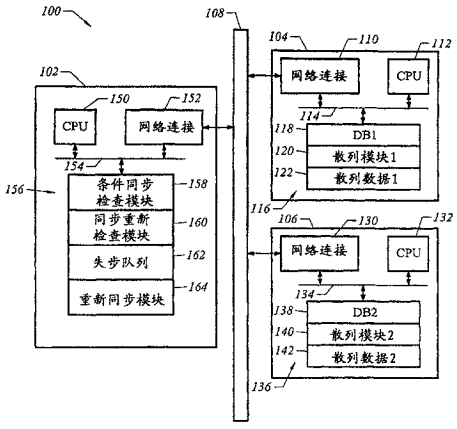
权利要求书 2 页 说明书 29 页 附图 2 页

(54)发明名称

用于识别在冗余数据存储中的异步数据并且将其重新同步的装置和方法

(57)摘要

一种计算机可读介质,包括用于比较数据库的可执行指令。所述可执行指令被配置来识别何时第一数据库的数据段与第二数据库的对应数据段条件失步以在第一时间建立条件失步状态。可执行指令在所述第一时间后允许一段潜伏期,在此期间对于所述第一数据库和所述第二数据库进行改变。可执行指令也在所述潜伏期之后确定是否所述第一数据库的所述数据段与所述第二数据库的所述对应数据段同步。可执行指令也填充重新同步的表格,其由复制机制用于将异步行带回同步。



CN 101176090 B

1. 一种用于比较数据库的方法，包括以下步骤：

识别何时第一数据库的数据段与第二数据库的对应数据段条件地失步，以在第一时间建立条件失步状态；

在所述第一时间后允许一段潜伏期，此期间对于所述第一数据库和所述第二数据库进行改变；并且

在所述潜伏期后确定是否所述第一数据库的所述数据段与所述第二数据库的所述对应数据段同步。

2. 按照权利要求 1 的方法，其中，所述识别的步骤包括对于所述第一数据库的所述数据段和所述第二数据库的所述对应数据段建立公共格式的步骤。

3. 按照权利要求 2 的方法，其中，所述识别的步骤包括在所述第一数据库和所述第二数据库之间建立公共格式的步骤，其解决下述之一：重叠列、重叠行、映射数据和异类数据存储。

4. 按照权利要求 1 的方法，其中，所述识别的步骤包括识别关键数据和非关键数据和对于所述非关键数据执行散列以建立散列数据的步骤。

5. 按照权利要求 4 的方法，其中，所述识别的步骤包括向所述散列数据增加用户指定的非关键数据的步骤。

6. 按照权利要求 4 的方法，还包括以下步骤：将来自所述第一数据库的关键数据与来自所述第二数据库的关键数据相比较，并且将来自所述第一数据库的散列数据与来自所述第二数据库的散列数据相比较，以识别所述条件失步状态。

7. 按照权利要求 1 的方法，其中，所述识别的步骤包括确定是否所述第一数据库的所述数据段包括至少部分地与所述第二数据库的所述对应数据段的对应行或列重叠的行或列的步骤。

8. 按照权利要求 1 的方法，其中，所述识别的步骤包括将所述第一数据库的所述数据段的子数据段并行地与所述第二数据库的对应子数据段相比较的步骤。

9. 按照权利要求 1 的方法，其中，所述识别的步骤包括以下步骤：识别所述第一数据库的一批行，压缩所述批的行，识别非关键数据和包括开始关键字值和结尾关键字值的关键数据，对于所述非关键数据执行散列以建立散列数据，并且向目标发送所述开始关键字值、所述结尾关键字值和所述散列数据。

10. 按照权利要求 9 的方法，还包括以下步骤：处理所述开始关键字值、所述结尾关键字值和所述散列数据以识别在所述第二数据库中的对应数据段，并且执行所述对应数据段的散列以产生第二散列数据。

11. 按照权利要求 10 的方法，还包括使用所述开始关键字值和所述结尾关键字值以在所述第一数据库和所述第二数据库内识别对应于在第一时间条件失步的所述第一数据库和所述第二数据库的数据段的行集的步骤。

12. 按照权利要求 11 的方法，还包括比较所述行集中的各行，以识别条件失步的行的步骤。

13. 按照权利要求 1 的方法，还包括：确定何时所述第一数据库的所述数据段与所述第二数据库的所述对应数据段不同步的步骤；和在所述第一数据库和所述第二数据库的动态操作期间将所述第一数据库的所述数据段与所述第二数据库的所述对应数据段重新同步

的步骤。

14. 按照权利要求 13 的方法，其中，重新同步包括以下步骤：

从所述第一数据库的所述数据段检索值；

锁定所述数据段；以及

在与所述第一数据库相关联的标记表格中存储所述值。

15. 按照权利要求 14 的方法，还包括以下步骤：

使用复制机制来从所述标记表格向所述第二数据库的所述对应数据段传送所述值。

16. 按照权利要求 15 的方法，还包括以下步骤：

使用所述复制机制来将对于所述第一数据库的改变排序。

17. 按照权利要求 13 的方法，其中，所述重新同步的步骤包括访问指定关于失步数据库数据段的信息的失步队列的步骤。

18. 按照权利要求 17 的方法，其中，所述重新同步的步骤包括确定数据方向流的步骤。

19. 按照权利要求 13 的方法，其中，所述重新同步的步骤包括使用复制标记来重新同步的步骤。

20. 按照权利要求 13 的方法，其中，所述重新同步的步骤包括使用源行更新技术来重新同步的步骤。

用于识别在冗余数据存储中的异步数据并且将其重新同步的装置和方法

技术领域

[0001] 本发明一般地涉及电子数据存储。具体上，本发明涉及一种用于识别在两个不同的计算机网络位置（假定保存相同的数据）中的异步数据的技术。而且，本发明涉及将所识别的异步数据重新同步。

背景技术

[0002] 许多企业在多个位置存储相同的电子数据。其原因包括：改善在计算机断电、故障或者灾难的情况下的这个信息的安全和可用性；以及多个实体和应用能够独立地管理相同信息。因此，当数据在一个数据源中改变时，经常需要通过多个可能的机制（包括数据复制）将其拷贝到一个或多个辅助数据源。存在与数据复制相关联的多个挑战。例如，能够独立地证实复制机制准确是重要的。即，确认应当从一个位置向另一个位置拷贝数据确实被拷贝了是重要的。当数据源失步（即异步）时，将它们尽可能有效地重新同步是重要的。需要在以下约束条件的情况下执行这些比较和重新同步行为：数据源中的改变的高活性/速率；动态环境，其中，没有机会关闭应用或者数据库以执行静态比较；以及伴随有在数据源之间的有限带宽的地理分离。

[0003] 鉴于上述情况，非常期望提供一种用于数据比较和重新同步操作的改进的技术。

发明内容

[0004] 本发明包括具有用于比较数据库的可执行指令的计算机可读介质。所述可执行指令被配置来识别何时第一数据库的数据段与第二数据库的对应数据段条件失步，以第一时间建立条件失步状态。可执行指令在所述第一时间后允许一段潜伏期，此期间对于第一数据库和第二数据库进行改变。可执行指令也在所述潜伏期后确定是否第一数据库的数据段与第二数据库的对应数据段同步。

[0005] 本发明提供了一种有效的机制，用于在支持有限带宽和/或在所述连接上传送的数据量很大的所述连接上比较在结构上类似或者不类似的表格。本发明便于随着时间流逝处理数据库的数据段或者子集。本发明也通过使用潜伏期来确定异步状态而有效地适应“在飞行中”的表格数据。

[0006] 本发明提供了一种企业基础结构软件平台，以能够进行高速、大量和在不同环境上的数据移动、数据管理和数据集成。本发明成功地用于银行、金融业务、保健、电缆、电话、公共部门和航空和航天工业。

附图说明

[0007] 通过结合附图的下面的详细说明，可以更全面地理解本发明，其中：

[0008] 图 1 图解了实现本发明的一个实施例的网络。

[0009] 图 2 图解了与本发明的一个实施例相关联的处理操作。

[0010] 类似的附图标号遍及各幅附图指示对应部分。

具体实施方式

[0011] 图 1 图解了按照本发明的一个实施例配置的计算机网络 100。网络 100 包括第一计算机 102、第二计算机 104 和第三计算机 106。计算机 102、104 和 106 通过传输介质 108 连接，所述传输介质可以是任何有线或者无线网络。

[0012] 计算机 104 包括标准计算机部件，其中包括通过总线 114 连接的网络连接电路 110 和中央处理单元 112。存储器也连接到总线 114。所述存储器存储第一数据库 (DB1) 118。散列 (hash) 模块 120 也被存储在存储器 116 中，所述散列模块 120 被配置为一组可执行的指令，用于对于数据库 118 的数据段执行散列操作，如下所述。散列模块 120 产生散列数据 122。所述散列模块 120 向计算机 102 传送散列数据 122。

[0013] 计算机 106 也包括标准计算机部件，其中包括通过总线 134 连接的网络连接电路 130 和中央处理单元 132。存储器 136 存储第二数据库 (DB2) 138。存储器 136 也存储可执行的散列模块 140，可执行的散列模块 140 从在第二数据库 138 中的数据段产生散列数据 142。在第二数据库 138 中的所述数据段对应于在第一数据库 118 中的数据段，在第一数据库 118 中的数据段被处理以产生第一组散列数据 122。散列模块 140 包括可执行指令，用于将第二散列数据 142 传送到计算机 102。因此，计算机 102 接收在计算机 104 上执行的第一散列操作的结果和在计算机 106 上执行的第二散列操作的结果。因为第一数据库 118 和第二数据库 138 的对应部分被处理，因此如果两个数据库数据段的内容相同，则散列数据应当对应。如果数据段不同，则很可能散列不同。注意，散列数据的传送大大地降低了网络通信量，或者便于比较数据库内容的快速技术。

[0014] 计算机 102 包括标准部件中央处理单元 150、网络连接电路 152 和系统总线 154。计算机 102 也具有存储器 156，其存储被配置来执行本发明的操作的可执行程序。在本发明的一个实施例中，存储器 156 存储条件同步检查模块 158。这个模块 158 将第一散列数据 122 与第二散列数据 142 相比较。如果散列数据不等同，则存在条件失步状态。所述条件同步检查模块 158 然后产生暂停以允许在识别条件失步状态后有潜伏期。所述潜伏期允许数据库有可能通过存在的同步机制而同步。在所述潜伏期后，所述散列模块 120 和 140 向计算机 102 提供散列数据 122 和 142，并且同步重新检查模块 160 被调用。这个模块 160 比较最近接收的散列数据 122 和 142。如果散列值相同，则数据库不再被认为是条件失步。另一方面，如果散列结果不一致，则数据库被认为失步。此处，同步重新检查模块 160 向失步队列 162 中装载关于数据库数据段的信息。重新同步模块 164 然后执行各种处理操作来在数据库之间建立同步，其细节在下面说明。

[0015] 图 2 图解了与本发明的一个实施例相关的处理操作。图 2 的第一处理操作是确定是否数据库是条件失步的 200。如上所述，条件同步检查模块 158 可以用于实现这个操作。如果数据库未失步，则处理返回到方框 200。如果数据库条件失步，则调用暂停或者潜伏期 202。其后，确定是否数据库仍然失步 (方框 204)。如果否，则处理返回到方框 200。如果是，则将关于数据库数据段的信息装载到失步队列 206 中。同步重新检查模块 160 可以执行这些操作。重新同步模块 164 然后可以被调用来重新同步在失步队列 208 中的内容。

[0016] 可以按照本发明的实施例来利用图 2 的处理的许多变化。例如，如果正在逐行基

础上执行处理，则开始在方框 200 确定是否在数据库中存在多个数据段或者行。如果是，则确定是否数据库数据段失步。如果是，则暂停 202。如果仍然不同步 204，则将所述数据段装载到队列 206 中。此处，获取下一行，并且处理返回到方框 200。

[0017] 已经说明了与本发明的一个实施例相关的主要操作。现在转向本发明的各个实施例的更详细的说明。应当明白，本发明的所有说明用于公开本发明的本质，因此不应当被理解为将本发明限定到所公开的特定实施例。例如，图 1 的网络配置是示例性的。可以在联网环境中以任何数量的方式来分布所述数据库和可执行模块。重要的是本发明的操作，而不是在哪里执行那些操作。

[0018] 本发明的技术适用于多种数据存储机制。但是，举例而言，在数据库的环境中公开本发明。即，在用于比较和同步在表格内的行和在数据库中的表格的技术的环境中公开本发明。术语行、表格和数据库用于描述用于存储相关信息的一般方法。数据库的一个普通示例是关系数据库。本发明也适用于其他类型的数据库，诸如分层数据库（例如 IBM ® IMS）或者基于关键字的数据库（例如 HP ® Enscribe）。对于这些类型的数据库，行经常被称为记录，并且列被被称为字段。

[0019] 另外，表格也可以被当作可扩展标记语言 (XML) 文档或者甚至 XML 文件中的一部分或者类似的“半结构”的数据，所述“半结构”的数据提供用于描述“行”、“列”和“表格”的一致方式。其也可以包括电子表格和其他文件和文档类型。表格也可以被当作相关数据或者数据子集的逻辑集合，诸如 SQL 视图。

[0020] 表 1 和 2 是分别指定产品和订单的数据库表格的示例。

[0021] 产品表（列是产品编号、产品名称和车轮数量）

[0022]

	产品编号	产品名称	车轮数量
行 1	100	轿车	4
行 2	200	船只	0
行 3	300	卡车	18

[0023] 表 1

[0024] 订单表（列是订单编号、产品编号、价格）

[0025]

	订单编号	产品编号	价格
行 1	1	100	20000
行 2	1	200	15000
行 3	2	200	9000
行 4	5	100	30000

[0026] 表 2

[0027] 在每个表格中的“唯一关键字 (unique key,UK)”是在表格内的一组列,用于唯一地识别在表格内的每行。例如,在所述产品表格内,产品编号是唯一关键字,因为它在表格中从不重复。在所述订单表格中,订单编号不是唯一的,因为它重复(值 1 两次出现)。但是,订单编号和产品编号的组合构成唯一关键字(在 (1,100)、(1,200)、(2,200) 和 (5,100) 中没有重复)。

[0028] 一种访问方法提供了从数据库中的表格检索数据的方式。检索选项包括用于从数据库检索所有行、基于关键字的特定行、特定列和在行和列子集上的计算的方法。示例访问方法包括 SQL、ODBC、ISAM、OracleOCI、被存储的规程和许多其他方法。访问方法也可以被当作从具有与数据库相同的一些特性的文档(诸如 XML 文档或者电子表格)检索数据的方式。

[0029] 在此使用的术语源和目标用于描述冗余数据从一个数据库向另一个数据库或者从一个表格向另一个表格的数据流的方向。经常地,这是预定的单向流。例如,如果正使用复制来从数据库 PRIMARY 向数据库 BACKUP 移动数据,则数据库 PRIMARY 和其中的表格被当作“源”,并且 BACKUP 和其中的表格被当作“目标”。

[0030] 另外,可以以其他方式来指定在源和目标之间的数据流,诸如 :

[0031] - 行的一个特定子集(由唯一关键字定义)被指定为在表格 1 中的“源”和在表格 2 中的“目标”,而行的另一个子集被指定为在表格 2 中的“源”和在表格 1 中的“目标”。

[0032] - 任何一个表格可以修改任何行,然后将那个行拷贝到另一个系统。

[0033] 表格比较涉及两个表格。如果下述条件为真,则同步两个表格(即“同步”):

[0034] - 在一个表格中的所有行可以在另一个表格中被找到,如唯一关键字所识别(对应行)

[0035] - 在每行中的所有列匹配在另一个表中的对应行中的所有列。

[0036] 这个定义涵盖了具有相同结构和行内容的表格。考虑下面的表格。

[0037] PRODUCT1 表格

[0038]

	产品编号	产品名称	车轮数量
行 1	100	轿车	4
行 2	200	船只	0
行 3	300	卡车	18
行 4	400	自行车	2

[0039] 表 3

[0040] PRODUCT2 表格

[0041]

	产品编号	产品名称	车轮数量
行 1	100	轿车	4
行 2	150	摩托车	2
行 3	200	船只	0
行 4	300	卡车	6

[0042] 表 4

[0043] 在这个示例中，所述两个表是失步的：

[0044] 1. 在 PRODUCT1 中的行 1 和 2 匹配在 PRODUCT2 中的行 1 和 3（唯一关键字产品编号在每个表中匹配，并且其他列也匹配）

[0045] 2. 在 PRODUCT2 内没有与 PRODUCT1 中的行 4 对应的表目，因此该行是失步的（不能在 PRODUCT2 中找到产品编号 400）

[0046] 3. 在 PRODUCT1 内没有与 PRODUCT2 中的行 2 对应的表目（找不到产品编号 150）——该行是失步的

[0047] 4. 在两个表格中找到产品编号 300（PRODUCT1 行 3，PRODUCT2 行 4）——但是车轮的数量不同——该行是失步的

[0048] 现在转向“类似结构”或者“重叠”的表格的一些示例。即使在这些类型的表格中数据不完全相同，但是所选择的数据（列和 / 或行）在所述表格之间流动，以保证适当地同步保存重叠的数据。

[0049] 考虑下面的示例，其涉及重叠列的情况。在这个示例中，一些列是相同的，而一些是不同的。

[0050] PROD 表格

[0051]

	产品编号	产品名称	车轮数量	库存
行 1	100	轿车	4	是
行 2	200	船只	0	是

[0052] 表 5

[0053] PROD 表格

[0054]

	产品编号	产品名称	说明	车轮计数
行 1	100	轿车	陆地运输工具	4
行 2	200	船只	适合于水中	0

[0055] 表 6

[0056] 在这个示例中，所述两个表格完全相同，但是可以被认为是同步，因为“重叠”列同步（产品编号和车轮数量 / 车轮计数）。在这些类型的情况下，由表格的用户来定义哪些列重叠，而不是在其中全部内容完全相同的上述的明显情况中那样。

[0057] 现在考虑重叠行。重叠的行集可以被看作同步。

[0058] BASEBALL_TEAMS（棒球队）表格

[0059]

	队名称	区域	胜
行 1	Giants	西部	82
行 2	Yankees	东部	75
行 3	Dodgers	西部	63
行 4	Red Sox	东部	74

[0060] 表 7

[0061] WEST_TEAMS（西部队）表格

[0062]

	队名称	胜	队伍类型
行 1	Giants	82	棒球
行 2	Dodgers	60	棒球
行 3	Lakers	20	篮球

[0063] 表 8

[0064] 在这种情况下，将在 BASEBALL_TEAMS（棒球队）表格中的队伍的子集（其中区域是“西部”的行）与其中队伍类型是“棒球”的在 WEST_TEAMS（西部队）表格中的行相比较。在这个示例中，仅仅一行将失步（(Dodgers,63)，(Dodgers,60)）。从对于 BASEBALL_TEAMS 返回的行集排除行 2 和 4，因为在 WEST_TEAMS 表中没有对应行的信息。类似地，在 WEST_TEAMS 表中的行 3 不预期具有在 BASEBALL_TEAMS 表中的表目，因为其是篮球队。另外，使用列的子集（区域不是 WEST_TEAMS 的一部分，并且队伍类型不是 BASEBALL_TEAMS 的一部分）。注意也可以比较重叠列和重叠行的组合。

[0065] 现在考虑映射数据的情况。下面的示例图解了表示同一事物的不同数据值。当数据从一个系统向另一个流动时，这被称为“数据映射”。为了扩展上述示例，NATIONAL_LEAGUE 表可能看起来如此：

[0066]

	队名称	胜	区域缩写
行 1	Giants	82	W
行 2	Dodgers	60	W

[0067] 表 9

[0068] 在这种情况下，在 ALL_TEAMS 中的区域等同于在 NATIONAL_LEAGUE 中的区域缩写，即使精确的值不同。

[0069] 本发明可以用于异类数据存储。异类数据存储可以不同地存储完全相同的信息。例如：

[0070] ORACLE_ORDER(ORACLE 订单）表格

[0071]

	订单 ID	订单日期
行 1	00100	04AUG2004
行 2	00200	10NOV2003

[0072] 表 10

[0073] SYBASE_ORDER(SYBASE 订单）表格

[0074]

	订单 ID	订单日期
行 1	100.00	2004-08-04
行 2	200.00	2003-11-10

[0075] 表 11

[0076] 在这些表格的每个中，数据的值实际上是相同的，但是被不同地表示。

[0077] 本发明的一个方面是使用行散列技术来比较表格。考虑其中要比较的表格数据驻留在数据库 1(DB1) 和数据库 2(DB2) 中的情况。一个比较客户端 (CC) 进程（例如被实现为条件同步检查模块 158) 建立到两个主服务器 (HS) 进程（例如在计算机 104 和 106 上）的连接，每个表格一个以比较。一个主服务器进程用于检索要比较的每个表格的表格数据。为了获得最大效率，在 HS 和 DB 之间的连接通常在同一系统上或者在很高速的网络上，因为有可能在每个 HS 和 DB 之间流动大量数据。

[0078] HS 进程和 CC 进程可以位于在网络中的任何位置（包括彼此在相同的系统上、一些在相同系统上或者所有都在不同的系统上）。也可以在所述系统之一上的 CC 和 HS 部分运行在同一进程中（作为不同的线程或者在相同线程中）。

[0079] 每个 HS 进程然后从以唯一关键字顺序分类的其对应的表格检索每行。所检索的每行包含一个或多个列的数据（如由数据库本身定义或者经由用户提供的元数据定义）。

根据数据库的类型和其提供的访问方法（示例包括 SQL、ODBC、Oracle OCI、ISAM 等）来以多种方式完成行检索。

[0080] HS 然后将每行的以下数据“封装”到存储缓冲器中。即，HS 封装构成行的唯一关键字的所有列，并且 HS 也封装在行中的剩余列的“行散列”。

[0081] 在此将散列定义为在一组字节上的数值计算，其总是对于所述给定的一组字节（以特定顺序）相同，但是具有很低的概率来对于不同组的字节相同（例如，字节“AB”的组被当作与字节“123”、“BA”、“ABC”的组的任何一个不同）。在这种情况下，所述字节组是每列的数据，其中，每列被转换为对于所述列的数据类型标准的格式。所述散列用于减少需要被传送以便进行两行的比较的数据量。

[0082] 散列计算的一个示例是循环冗余校验 (CRC)。例如，可以在 500 字节的行数据上计算 32 比特的 CRC，对于那行将要传送的数据量从 500 字节减少到 4 字节。在这种情况下，可以比较每行的 4 个字节，以确定是否两行同步，而不是比较 500 字节。这有效地大大减少了要传送的数据的总量，并且成比例地扩大了在给定时段中在给定网络上可以比较的数据量。

[0083] 另外，HS 将单独行信息——唯一关键字外加散列——封装到更大的行块中，所述更大的行块继而在更大的消息中被压缩和通过网络被传送回 CC。压缩技术包括普遍可以获得的算法，诸如 Ziv/Lempel 算法。小行向较大消息的压缩和成块也具有大大减少需要通过网络而传送的数据量，并且成比例地扩大了要比较的数据容量的效果。

[0084] 当其接收到压缩的行散列块时，CC 解压缩所述块，并且将所述消息解块为单独的行。CC 可以对于在独立的处理线程中的每个 HS 进程执行所有的通信和解块行为，以最大化比较的并行化和吞吐量（使得多个处理器同时工作于比较任务）。从 DB1 和 DB2 以唯一关键字的顺序来划分行。

[0085] 在 CC 内的行比较线程然后将来自表 1 的行与在表 2 中的行相比较。表 1 被指定为“源表”(ST)，并且表 2 被指定为“目标表”(TT)。比较进行如下：

[0086] - 依序将失步的行累积到“可能失步队列”(MOOSQ)

[0087] ○存储被比较的行的关键字和散列

[0088] ○存储当执行所述比较时的时间戳

[0089] ○ MOOSQ 可以是存储器队列、文件或者用于当发现行失步时存储行比较数据的等价机制

[0090] - 开始，检索来自 ST 和 TT 的行

[0091] - 所述比较然后执行下面的循环，直到不能从 ST 或者 TT 获得更多的行

[0092] ○当前行被指定为 row(ST) 或者 row(TT)

[0093] ○下一行（跟随当前行）是下一 row(ST) 或者下一 row(TT)

[0094] ○当前行的唯一关键字是 key(ST) 或者 key(TT)

[0095] ○当前行的散列是 hash(ST) 或者 hash(TT)

[0096] ○如果 key(ST) 小于 key(TT) 或者不能从 TT 获得更多的行

[0097] ■作为向 MOOSQ 的“丢失插入”操作，将 row(ST) 排队（参见下面）——这表示为了将表同步，必须将 row(ST) 插入 TT

[0098] ■ row(ST) 变为下一 row(ST)

[0099] ○如果 key(ST) 大于 key(TT) 或者从 ST 不能获得更多的行

[0100] ■作为向 MOOSQ 中的“丢失删除”操作，将 row(TT) 排队——为了将表同步，必须删除 row(TT)

[0101] ■ row(TT) 变为下一 row(TT)

[0102] ○如果 key(ST) 等于 key(TT)

[0103] ■如果 hash(ST) 与 hash(TT) 不同

[0104] ●作为向 MOOSQ 中的“丢失更新”操作，将 row(ST) 排队——为了将表同步，row(ST) 必须替换 row(TT)

[0105] ■如果 hash(ST) 等于 hash(TT)

[0106] ●不进行任何操作，行是同步的

[0107] ■在两种情况下，row(ST) 变为下一 row(ST)，并且 row(TT) 变为下一 row(TT)

[0108] 在所述比较后，MOOSQ 保存对于被发现失步的所有行的关键字。这些关键字的每个可以随后用于从数据库选择全行图像，以产生失步的行的列表。

[0109] 考虑下面的示例，其依赖于表 12 的源表 (ST) 和表 13 的目标表 (TT)。

[0110] 源表 (ST)

[0111]

	产品编号	产品名称	车轮数量	行散列
行 1	100	轿车	4	12345678
行 2	200	船只	0	63482133
行 3	300	卡车	18	34747426
行 4	400	自行车	2	98765342
行 5	500	直升飞机	3	98878972

[0112] 表 12

[0113] 目标表 (TT)

[0114]

	产品编号	产品名称	车轮数量	行散列
行 1	100	轿车	4	12345678
行 2	150	摩托车	2	23823283
行 3	200	船只	0	63482133
行 4	300	卡车	6	49347234

	产品编号	产品名称	车轮数量	行散列
行 5	500	飞机	3	23823932

[0115] 表 13

[0116] 比较步骤

[0117]

	源行关键字	目标行关键字	源行散列	目标行散列	操作类型
1	100	100	12345678	12345678	相等
2	200	150	63482133	23823283	丢失删除
3	200	200	63482133	63482133	相等
4	300	300	34747426	49347234	丢失更新
5	400	500	98765342	23823932	丢失插入
6	500	500	98878972	23823932	丢失更新
7	没有更多	没有更多			（完成）

[0118] 表 14

[0119] 在比较后的 MOOSQ 内容

[0120]

	源行关键字	目标行关键字	源行散列数据	目标行散列	操作类型	比较时间戳
1		150	63482133	23823283	丢失删除	2004-05-04 08:33:32
2	300	300	34747426	49347234	丢失更新	2004-05-04 08:34:10
3	400		98765342		丢失插入	2004-05-04 08:37:58
4	500	500	98878972	23823932	丢失更新	2004-05-04 08:39:00

[0121] 表 15

[0122] 上述的行散列技术可以被应用到前述的重叠表情形，包括：

[0123] - 重叠列情况

[0124] - 重叠行情况

[0125] - 映射数据情况

[0126] - 异类数据存储

[0127] 在重叠列和映射数据情况中，系统的用户提供作为比较过滤器的重叠列列表。作为另一种替代方式，也可以从在表上使用的索引或者在所述表上的视图自动得到所述列列表。在重叠列列表中的每列在每个表中具有相同的信息（对于同步的行），并且将行散列计算限定到仅那些列，滤除不同的列。

[0128] 考虑如上所述的重叠列情况。重叠列列表将为如下：

[0129]

	源表列 (PROD)	目标表列 (PRODUCT)
1	产品编号	产品编号
2	产品名称	产品名称
3	车轮数量	车轮计数

[0130] 表 16

[0131] 注意，即使车轮数量和车轮计数具有不同的名称，但是它们表示相同信息。另外，已经从 PROD 表列表省略了库存项目，因为它不与在 PRODUCT 中的任何列重叠，并且从 PRODUCT 表列表省略说明项目，因为其在 PROD 中没有重叠的列。

[0132] 在重叠的行情况下，系统的用户向被返回用于比较的行集应用过滤器定义。这可以是 SQL 语句 WHERE 子句、用于指定在表格内的物理或者逻辑分区的名称的标识符或者用于在比较中从表格检索和提供行的特定子集的其他类型的机制。

[0133] 使用上述的重叠行示例，下面的过滤器将被应用到：

[0134] -BASEBALL_TEAMS 表格上，仅仅选择其中区域等于“西部”的行

[0135] -WEST_TEAMS 表格上，仅仅选择其中队类型是棒球的行。

[0136] 另外，仅仅队名称和胜列重叠。结果产生的要比较的行集是：

[0137] BASEBALL_TEAMS 表格

[0138]

	队名称	胜
行 1	Giants	82
行 3	Dodgers	63

[0139] 表 17

[0140] （行 2 和 4 被滤除，因为区域不等于“西部”，区域列被滤除，因为其不重叠）

[0141] WEST_TEAMS 表格

[0142]

	队名称	胜
行 1	Giants	82
行 2	Dodgers	60

[0143] 表 18

[0144] （行 3 被滤除，因为队类型不等于“棒球”，队类型列被滤除，因为它不重叠）

[0145] 对于异类数据存储，每列在执行散列计算之前必须被转换到其基本“类型”的标准表示。否则，表示相同事物但是不同地被表示的列数据将导致对应行的不相等的散列值。

[0146] 在如上所述的示例中，编号和日期被转换为标准格式。标准格式可以非常不同，但是需要被定义以最低程度地用于下面的数据类型：

[0147] - 编号

[0148] - 日期

[0149] - 具有尾部空间 (trailing spaces) 的字符列

[0150] 只要所选择的标准被一致地应用到两个表，并且因此对于两个表一致地表示数据，则可以进行精确的散列计算和比较。

[0151] 编号表示的一个示例格式如下：

[0152] - 删节在前的 0

[0153] - 删节小数点右部的尾部的 0

[0154] 日期表示的示例格式是：

[0155] -4 位年，后随 2 位月，后随 2 位日 (YYYYMMDD)

[0156] 使用所述异类数据存储示例，对于 ORACLE_ORDER 和 SYBASE_ORDER 表格标准化数据如下。

[0157]

	订单 ID	订单日期
行 1	100	20040804
行 2	200	20031110

[0158] 表 19

[0159] 在一些情况下，如果所述比较处理被划分为多个部分（其中每个关注一个或多个表分区），则可以更快地处理具有在多个盘驱动器上分布的具有大量行的表（被分区的表）。这可以通过下述方式来进行：通过对于被分区的表建立多个逻辑比较，其中每个负责给定范围的数据。这种方法类似于所述重叠行情况，其中，对于每个比较使用过滤器。例如：

[0160] 源表 (ST)：

[0161]

	最后的名称
分区 1	“A”-“M”
分区 2	“N”-“Z”

[0162] 表 20

[0163] 目标表 (TT)：

[0164]

	最后的名称
分区 1	“A”-“G”
分区 2	“H”-“R”
分区 3	“S”-“Z”

[0165] 表 21

[0166] 可以将在这个表上的比较划分为两个比较：

[0167] -比较 1：从 ST 和 TT 选择行，其中，最后的名称在“A”和“M”之间

[0168] -比较 2：从 ST 和 TT 选择行，其中，最后的名称在“N”和“Z”之间

[0169] 如果同时运行“被划分的比较”，则行范围“A”-“M”和“N”-“Z”的检索会并行发生，因为它们访问不同的盘驱动器。注意，所述分区不必相同，以便实现被划分的比较。

[0170] 行散列技术说明了一种用于确定失步行的有效方法，但是，如果在改变下层表的同时应用行散列技术，则其会产生不确定的结果。这是因为在进行比较的同时下层行值改变的事实，因此，所述比较会将在比较期间实际上变为同步的行识别为失步；用于将表格保持同步的任何下层复制机制都会在源行改变的时刻和该改变作用到目标表的时刻之间引入延迟（潜伏时间）。

[0171] 与本发明相关的一种技术用于补充行散列技术，并且在改变的环境中产生精确的结果。其已知为“确认失步”处理 (COOS)，可以使用同步重新检查模块 160 来实现它。这个处理的目标是以下面的任何格式（但是不限于这些）来披露在一个时段后仍然失步的行：

[0172] -要由用户查看的报告

[0173] -包含会被另一个机制（诸如用于将所述表格在逐行基础上重新同步的程序）处理的失步和可能失步的行的列表的队列或者文件（确认失步队列，或者 COOSQ)

[0174] ○ COOCQ 也可以被用作另一个 COOS 处理的输入，以查看哪些行在另一个时段后或者在应用重新同步步骤后仍然失步。

[0175] 在本发明的一个实施例中，对于在 COOS 步骤中处理的行存在三种可能的状态：

[0176] -“持续失步”——表示自从进行行散列比较以来还没有更新行，因此所述行可以被假定为失步

[0177] -“在飞行中”——表示在行散列步骤中失步的目标行已经被更新，因此通过复制

或者其他机制而已经对于这个行执行了工作，但是还没有确认所述行是同步的

[0178] -“同步”——表示源行经由复制机制而被应用到目标

[0179] 注意，如果下层表连续改变，则即使“同步”的状态也不保证行当前是同步的。但是其确实表示用于将行保持同步的复制或者其他相关联的处理正在进行。

[0180] COOS 处理假定：

[0181] -存在用于将行保持同步的另一个有效机制（例如数据库复制）——一般被称为“复制机制”

[0182] -所述复制机制很可能将在一个表中的改变的应用应用到在特定时间窗口内（已知为“复制潜伏时间”） 的另一个表

[0183] ○通过查看来自所述复制机制的潜伏时间统计程序地或者由用户在没有这样的机制的时手动确定这个潜伏时间在一个实施例中，COOS 处理工作如下：

[0184] -当还有 MOOSQ 记录要处理时，

[0185] ○从 MOOSQ 获得下一个记录

[0186] ○如果失步操作类型 (Optype) 是

[0187] ■“丢失删除”

[0188] ●使用 key(TT) 来从 TT 选择全 row(TT)

[0189] ●如果未发现 row(TT)，则状态变为“同步”，否则，row(TT) 是“持续失步”

[0190] ■“丢失插入”

[0191] ●使用 key(ST) 来从 TT 选择全 row(TT)

[0192] ●如果未发现 row(TT)，则状态变为“持续失步”

[0193] ●如果发现 row(TT)，则计算 hash(TT)

[0194] ○如果 hash(TT) 与 hash(ST) 相同，则状态变为“同步”

[0195] ○如果 hash(TT) 与 hash(ST) 不同，则状态变为“在飞行中”

[0196] ■“丢失更新”

[0197] ●计算 hash(TT)

[0198] ●如果 hash(ST) 等于 hash(TT)，则状态变为“同步”

[0199] ●如果 hash(ST) 不等于 hash(TT)

[0200] ○比较旧的 hash(TT) 与新的 hash(TT)

[0201] ■如果相同，则状态变为“持续失步”

[0202] ■如果不同，则状态变为“飞行中”（如果散列改变，则其表示目标行值改变，并且数据库复制机制——如果有的话——已经更新了行并且有可能正在工作，但是还没有确认所述数据相同）

[0203] ○对于持续失步并且可选地在飞行中的行，

[0204] ■ COOS 可以使用 key(TT) 来查找 row(TT) 的所有对应列值，并且使用 key(ST) 来查找 row(ST) 的所有对应列值

[0205] ●在 MOOSQ 中仅可以获得所述关键字和散列，因此为了看到整个行图像，这是必要步骤

[0206] ●使用诸如 SQL SELECT...WHERE 之类的标准数据库访问方法来进行查找

[0207] 在此，COOSQ 和 / 或 COOS 报告包含或者显示持续失步的所有行的内容，并且如果

期望，则包含或者显示那些在飞行中的行。

[0208] 注意，COOS 处理可以与行散列或者其他初始比较技术同时发生，只要 COOS 处理等待执行每个行确认步骤，直到复制潜伏时间已经超过。例如，如果复制潜伏时间是 60 秒，并且初始比较在 9:30 披露了一个失步行，则不执行那个行的确认步骤，直到 9:31（即使在表中的所有行的初始比较直到 9:33 未完成也是如此）。

[0209] 考虑下面的示例。用户将预期的潜伏时间设置为 60 秒。

[0210] 在行散列比较后的 MOOSQ 内容

[0211]

	源行关键字	目标行关键字	源行散列	目标行散列	操作类型	时间
1		150	63482133	23823283	丢失删除	08:33:32
2	300	300	34747426	49347234	丢失更新	08:34:10
3	400		98765342		丢失插入	08:37:58
4	500	500	98878972	23823932	丢失更新	08:39:00

[0212] 表 22

[0213] 复制行为

[0214]

	源行关键字	目标行关键字	源行散列	目标行散列	行为	时间
1		150		23823283	被应用的删除	08:33:50
2	300	300	34747426	34747426	被应用的更新	08:34:20
3	500	500	98878972	66554443	被应用的更新	08:34:20

[0215] 表 23

[0216] COOS 行为

[0217]

	源行关键字	目标行关键字	时间	操作	结果	新状态
1		150	08:34:20	lookup TT key = 150	找不到记录	同步

	源行关键字	目标行关键字	时间	操作	结果	新状态
2	300	300	08:35:11	lookup TT key = 300	找到记录，新的 hash(TT) 等于 hash(ST)	同步
3	400		08:38:59	lookup TT key = 400	找不到记录	持续失步
4	500	500	08:40:01	lookup TT key = 500	找到记录，新的 hash(TT) 不等于 hash(ST)，但是新的 hash(TT) 不等于旧的 hash(TT)	在飞行中

[0218] 表 24

[0219] COOS 行为的描述

[0220] - 检索 MOOSQ 记录 1

[0221] ○等待直到当前时间 (8:33:33) 大于比较时间外加潜伏时间 (8:33:32+60) 以允许复制应用操作的一个机会

[0222] ○使用目标行关键字值 150 来执行向目标表内的查找

[0223] ○未找到目标记录，因此假设已经应用了删除

[0224] ○将所述操作标注为现在“同步”

[0225] - 检索 MOOSQ 记录 2

[0226] ○等待直到当前时间 (8:35:11) 大于比较时间外加潜伏时间 (8:34:10+60) 以允许复制应用操作的一个机会

[0227] ○使用目标行关键字值 300 来执行向目标表内的查找

[0228] ○找到目标记录；计算在目标表上的一个行散列（新的散列）

[0229] ○将源行的散列与目标行的新散列相比较，它们相同

[0230] ○将所述操作标注为现在“同步”

[0231] - 检索 MOOSQ 记录 3

[0232] ○等待直到当前时间 (8:38:59) 大于比较时间外加潜伏时间 (8:38:58+60) 以允许复制应用操作的一个机会

[0233] ○使用源行关键字值 400 来执行向目标表内的查找

[0234] ○未找到目标记录，因此假设还没有应用丢失插入

[0235] ○将所述操作标注为现在“持续失步”

[0236] - 检索 MOOSQ 记录 4

[0237] ○等待直到当前时间 (8:40:01) 大于比较时间外加潜伏时间 (8:39:00+60) 以允许复制应用操作的一个机会

[0238] ○使用目标行关键字值 500 来执行向目标表内的查找

[0239] ○找到目标记录；计算在目标表上的行散列（新的散列）

[0240] ○将源行的散列与目标行的新散列相比较，它们不同，但是因为自从初始比较起新的散列已经在目标行中被改变，因此一定是自从初始比较起已经改变了所述行，因此，将所述操作标注为“在飞行中”

[0241] 如上所述，可以使用从行散列技术输出的 MOOSQ 来作为输入而执行 COOS 处理技术。COOS 处理使用行关键字、散列和由行散列技术产生的相关信息来确定哪些行有可能失步，然后在可能的复制潜伏时间已经超过时使用该信息来查找详细的行数据。

[0242] 但是，COOS 处理不限于行散列技术输入。对于仅仅在源表（丢失插入）或者目标表（丢失删除）中找到的那些行，COOS 处理可以被应用到提供失步数据的关键字的任何输入。对于在两个表中但是在非关键列中具有不同值的那些行，非关键列的数据不必是行散列——它可以是表示那些列的值的任何内容，其中包括所述列值本身。只要在源和目标上一致地表示数据，则足以确认处理发生。

[0243] 在本发明的一个替代实施例中，使用批散列技术来比较表格。在一个实施例中，为了说明目的，将批散列技术划分为两个阶段（虽然它们可以同时运行）：

[0244] 1. 差批确定

[0245] 2. 行细节获取和比较

[0246] 一旦完成了这些阶段，MOOSQ 存储“可能失步”行集（如同行散列技术）。由此，即使下层的数据库继续改变也可以进行上述的确认失步 (COOS) 步骤以确定保持持续失步的行集。

[0247] 可以将差批确定实现如下。如同行散列技术那样，主机服务器处理来自源和目标表的检索行。在如上所述选择允许异类情况后标准化每行的列数据。另外，能重叠行和列的特征等同地应用到批散列技术——可以从每个表选择特定行和列子集。

[0248] 本技术的目的是将网络通信量比行散列更进一步降低。在批散列中，数据的流发生如下。在每个 HS 处理中，从源和目标表检索的行（理想上是并行的）被装载到行读取缓冲器中：

[0249] - 如同行散列技术那样，以唯一关键字顺序来从源表检索行，

[0250] - 理想上，在 HS1（例如 DB1 118）和 HS2（例如 DB2 138）中并行发生从源和目标表的行检索

[0251] 使用所检索的行，HS1 随后建立和向 HS2 发送多批行信息：

[0252] - 对于“源批尺寸”选择固定编号 N（例如 10 行）

[0253] - 以下面的方式将在 HS1 上的行读取缓冲器中的前 N 个行封装到“批行记录”中：

[0254] ○开始关键字列值（行 1）

[0255] ○结尾关键字列值（行 N）

[0256] ○使用在行散列技术中说明的散列技术（例如 CRC）在 N 行中的所有列（或者所选择的列）上计算的散列数据

[0257] - 将批行记录排队到批散列块中

[0258] - 将此对于行 N+1 到 2N、2N+1 到 3N 等重复

[0259] ○最后的批是特殊情况，并且最终的源批行记录具有被表示为“文件结尾”的结尾关键字值

[0260] - 每当批散列块充满，或者当本地行读取缓冲器充满或者接近充满时，向 HS2 处理发送批散列块

[0261] ○充满应当可配置，并且通常被优化来实现在网络上的数据的最大吞吐量，或者当行读取缓冲器充满或者接近充满

[0262] ○可以压缩块来获得更大的效率

[0263] HS2 读取和比较批行记录与其本身的行（来自目标表）：

[0264] -HS2 以唯一关键字顺序来读取目标表的行数据

[0265] - 从自 HS1 进入的消息解块每个批行记录

[0266] - 对于来自 HS1 的每个批行记录：

[0267] ○ HS2 确定批行比较的边界，即需要与源批行记录匹配的、在 HS2 中的行集

[0268] ■目标批行记录的开始行是从目标表 (TT) 可以获得的第一行

[0269] ■目标批行记录的结尾行是具有小于或者等于源批行记录的最后行的关键字的行

[0270] ■ HS2 也计算在其行集中的所有行上的散列

[0271] ○ HS2 然后将源批行记录的散列与目标批行记录的散列相比较

[0272] ○如果它们不同，则将所述批声明为“差”——这表示在下层行集中的至少一行失步，并且有可能更多；然后建立差批记录，其包括：

[0273] ■开始源和目标批行记录的关键字——开始差批值——的“更小者”

[0274] ■结尾源和目标批行记录的关键字——结尾差批值（注意这必须总是与源批行记录的结尾关键字值相同）——的“更大者”

[0275] ■如果源批行记录指示作为结束关键字值的“文件结尾”，则目标批行记录必须也指示作为其结尾关键字值的“文件结尾”

[0276] ○差批记录被发送（或者缓冲以发送）到 CC 处理

[0277] CC 处理再向可能失步队列 (MOOSQ) 中写入差批记录。所述 MOOSQ 保存失步批的边界。这表示每批至少一行失步，并且可能更多。

[0278] 此点，进行第二步骤以确定失步的精确行集。这个步骤已知被称为“行细节获取”。可以在确定所有的差批后或者在该处理期间执行这个步骤。在确定每个差批后确定失步的行遵循与行散列技术大部分相同的逻辑。所述处理进行如下：

[0279] - 由 CC 来检索差批

[0280] -CC 从源 (HS1) 和目标 (HS2) 请求下层行的细节（每行的列）

[0281] - 每个 HS 处理制定查询以返回在差批记录的范围中的每行；例如，如果差批记录具有带有关键字“B”的开始记录和带有关键字“E”的结尾记录，则将返回在那些关键字值（包括）之间的任何内容（例如 B、C、D、E）。

[0282] ○或者，如果相同的 HS 处理用于批检索步骤和行细节获取步骤中，则 HS 处理可以在批检索期间高速缓存详细的行数据，预期可能需要细节。

[0283] - 向 CC 返回行

[0284] -CC 像在行散列步骤中那样比较每行，并且如果行不匹配，则 CC 向 MOOSQ 输出差行值

[0285] 在此，MOOSQ 包含“可能失步”的行（向在行散列技术中那样）。从那里，如果正在

连续地更新下层的表，则可以在可配置的潜伏期后应用用于“评估连续改变的表的失步条件”的技术，如上所述。

[0286] 结合下面的示例更全面地理解批行技术。考虑：

[0287] 批大小＝3

[0288] 成块＝1个差批记录（为了简化）

[0289] 源表(ST)内容

[0290]

关键字	其他值	批#	批行记录内容	注释
A	1	1		
B	2	1		
C	3	1	A,C,Hash(1,2,3)	批1的结尾（计数＝批大小）
D	4	2		
F	5	2		
G	6	2	D,G,Hash(4,5,6)	批2的结尾（计数＝批大小）
H	7	3	H,EOF,Hash(7)	批3的结尾＝EOF

[0291] 表25

[0292] 目标表(TT)内容

[0293]

关键字	其他值	批#	批行记录内容	注释
A	1	1		＜＝源批1的结尾关键字(C)
C	3	1	A,C,Hash(1,3)	＜＝源批1的结尾关键字——不匹配的批行记录(Hash(1,2,3)不等于Hash(1,3))——输出到BBQ
D	4	2		＞源批1的结尾关键字,＜＝源批2的结尾关键字(G)
F	5	2		＜＝源批2的结尾关键字(G)

关键字	其他值	批 #	批行记录内容	注释
G	6	2	D,G,Hash(4,5,6)	＜＝源批 2 的结尾关键字 (G)——匹配的批行记录（在源、目标中都是 Hash(4,5,6))——好批（抛出）
H	7	3		＞源批 2 的结尾关键字,＜＝源批 3 的结尾关键字 (EOF)
I	8	3	H,EOF,Hash(7,8)	＜＝源批 3 的结尾关键字 (EOF)——不匹配的批行记录 (Hash(7) 不等于 Hash(7,8))——输出到 BBQ

[0294] 表 26

[0295] HS1 从 ST——关键字＝ A，B，C——提取前三个记录，并且计算在那三个记录上的散列 (hash)。所述批行记录包含：

[0296] - 开始关键列值 -A

[0297] - 结尾关键列值 -C

[0298] -hash(1,2,3)

[0299] HS1 向 HS2 发送批行记录。HS2 执行下面的处理：

[0300] - 从 TT（关键字＝ A) 读取第一行，并且将其与来自 HS2 的差批行的结尾关键字值 (C) 相比较—因为其小于或等于结尾关键字 C，所述行被包括在目标的当前批中

[0301] - 从 TT（关键字＝ C)——也包括这个行——读取第二行，因为其小于或者等于结尾关键字 C

[0302] - 从 TT（关键字＝ D) 读取第三行——将其存储以用于下一次的批比较

[0303] - 计算在记录 A、C 上的散列——这与源批行记录散列 (hash(1,3)) 不同，因为在源（关键字＝ B) 中存在的目标批中丢失一行

[0304] - 包含 A、C 的批行记录被输出到差批队列 (BBQ)HS1 然后从在 ST 中的接着的 3 行建立下一个批记录：

[0305] - 开始关键字＝ D

[0306] - 结尾关键字＝ G

[0307] -hash(4,5,6)

[0308] HS1 向 HS2 发送所述批行记录，HS2 执行下面的处理：

[0309] - 使用来自 TT 的剩余行，其未进入前一批（关键字＝ D)

[0310] - 累加行，直到其从源批记录（关键字＝ H) 发现大于结尾关键字的一行；包括直到并且包括在目标批行记录中的 G 的每行

[0311] - 计算目标行散列——hash(4,5,6)——其匹配源批行记录散列，并且确定此批是“好”的

[0312] HS1 建立和向 HS2 发送最后一批。所述批包含下述内容：

[0313] - 开始关键字＝ H

[0314] - 结尾关键字＝ EOF

[0315] -hash(7)

[0316] HS2 包括在其批行记录计算中的所有剩余行，因为来自源批行记录的结尾关键字是 EOF。这些记录不匹配，因为目标批行散列值是 hash(7,8)。这个记录被置入 BBQ 中。

[0317] 在行细节获取步骤中，CC 从 BBQ 检索每个差批记录。CC 从 HS1 和 HS2 的每个检索由开始和结尾关键字值限制的暗示的行集的行细节。一旦已经检索到行，则以所检索的顺序比较它们。

[0318] 比较处理

[0319]

差批记录值（开始关键字，结尾关键字）	源行细节	目标行细节	结果
A，C	A，1	A，1	相等
	B，2	C，3	丢失插入 (B ＜ C)——向 MOOSQ 输出 B，2
	C，3	C，3	相等
H，EOF	H，7	H，7	相等
	EOF	I，8	丢失删除——向 MOOSQ 输出 I，8
	EOF	EOF	完成

[0320] 表 27

[0321] 此处的 MOOSQ 包含可能失步的行。可以在一段潜伏期后执行随后的比较，如在上面的行散列之后所述，以便识别仍然失步（并且通过复制或者类似的机制被使得同步）的行。

[0322] 所述比较技术说明了用于在应当同步但是未同步的两个表中识别行的方法。这些技术也当下层数据正在不断改变时识别如此进行的方式。

[0323] 一旦将两个表识别为失步，则经常要求将它们带回同步。其涉及：

[0324] - 确定应当同步哪些行（或者使用数据驱动规则在逐行基础上或者所有行）——被称为“重新同步行集”

[0325] - 对于在重新同步行集中的每行，确定哪个表是改变信息的真实源（其可以在逐行基础上、数据驱动规则或者在一个方向上流动的所有行）

[0326] - 从源向目标实际移动所述改变的信息，或者触发此移动，使得各行同步

[0327] 在本发明的一个实施例中，将异步队列 (OOSQ) 用作对于重新同步处理的输入。OOSQ 向可能被重新同步的那些行提供包含最少需要的属性，其包括每行关键字和操作类型。许多不同类型的处理可以产生 OOSQ，其中包括如上所述用于产生 MOOSQ 和 COOSQ 队列的比较处理。

[0328] 确定重新同步行集和要同步的数据的方向流是在将表带回同步中的第一步骤。要重新同步的最后行集已知被称为“重新同步队列”(RQ)。这可以采取包括文件和队列的许多形式,但是包含用于影响必要行的重新同步处理所需要的本质信息。RQ 和 OOSQ 可以甚至是相同的文件。如果一旦确定是否和如何应用每个改变就立即进行改变的应用,则 RQ 也可以被当作逻辑概念。

[0329] 可以以下面的方式来进行确定重新同步队列的内容。默认行为是一种手段。在所述比较处理期间,一个表被指定为源,另一个表是指定目标。所述默认行为是从源向目标流动在重新同步行集中的每行。另一个手段是向用户提供行以手动查看和确定。在这种方法中,所述系统从如上所述的 OOSQ 向系统的用户提供数据。如果所述操作是丢失插入,则显示源行 ;对于丢失删除,显示目标行,并且对于丢失更新,显示源和目标行,并且一种选择是加亮失步列以得到用户的注意。

[0330] 当用户查看每行时,他可以确定 1) 是否重新同步该行,2) 在哪个方向上发送行(即确定“真实”源)。注意在已经假定了用于比较的源和目标时,手动查看处理可以在逐案基础上倒转这个关系。系统向 RQ(其可能是 OOSQ 本身或者新的文件 / 存储器队列) 中存储这些设置,以便随后可以同步行。或者,所述系统可以在完成查看后直接执行这些改变,而不存储所述方向信息。

[0331] 用于确定是否应用行和在哪个方向的另一种方法是使用数据驱动规则。在这种情况下,系统提供一种机制,通过其,用户可以指定在 OOSQ 内的行的子集以应用。存在用于实现使能这个功能的语法或者提供提示这个信息的用户界面的许多选项。

[0332] 考虑下面的示例。当同步时,ALL_CITIES_EAST(所有东部城市) 和 ALL_CITIES_WEST(所有西部城市) 应当具有相同的数据。假定 ALL_CITIES_EAST 被指定为源表,并且 ALL_CITIES_WEST 被指定为目标。也假定关于给定城市的数据总是在那个区域的表格的拷贝中发起 (例如关于纽约的数据总是在 ALL_CITIES_EAST 中发起)。

[0333] ALL_CITIES_EAST 表 (指定的源表格)

[0334]

	城市	区域	天气
1	纽约	东部	下雨
2	迈阿密	东部	晴
3	芝加哥	中部	有风

[0335] 表 28

[0336] ALL_CITIES_WEST 表 (指定的目标表格)

[0337]

	城市	区域	天气
1	西雅图	西部	下雨

	城市	区域	天气
2	费城	东部	下雪
3	迈阿密	东部	多云

[0338] 表 29

[0339] OOSQ 内容和期望的结果

[0340]

	城市	区域	OOSQ 操作	期望操作
1	纽约	东部	丢失插入	向 ALL_CITIES_WEST 中插入行
2	费城	东部	丢失删除	从 ALL_CITIES_WEST 删除行
3	迈阿密	东部	丢失更新	替换在 ALL_CITIES_WEST 中的行
4	西雅图	西部	丢失删除	向 ALL_CITIES_EAST 中插入行
5	芝加哥	中部	丢失插入	不做任何事情

[0341] 表 30

[0342] 注意，在除了表格 30 的 #4 和 #5 之外的所有情况下，数据在从指定源向目标的方向上流动。但是，在情况 #4 中，因为西雅图在西部区域，该数据应当被插入 ALL_CITIES_EAST 中，而不是从 ALL_CITIES_WEST 删除行。情况 #5 用于说明不需要在任何方向上应用数据的情况。因此，用于确定方向数据流的自动规则可以如下：

[0343] - 对于在 OOSQ 中的每个记录

[0344] ○如果区域是东部，则改变的方向是从 ALL_CITIES_EAST 向 ALL_CITIES_WEST

[0345] ○如果区域是西部，则改变方向是从 ALL_CITIES_WEST 向 ALL_CITIES_EAST

[0346] ○如果区域是其他，则不在任何方向上应用改变

[0347] 所述系统可以使能涉及关于表格数据、操作类型、比较时间戳和其他信息的组合的布尔逻辑的规则，能够或者禁止行集的重新同步，并且确定那些行的方向。更复杂的规则示例：

[0348] - 如果区域是东部，并且 OOSQ 操作不是丢失删除，或者城市是纽约，则向 ALL_CITIES_WEST 应用改变

[0349] 1 基于规则和手动查看技术的组合

[0350] 所述系统也可以提供扩展规则规范能力的选项以使得能够手动查看和确定未被规则本身确定的行。使用上述示例，规则可以规定：

[0351] - 对于在 OOSQ 中的每个记录

[0352] ○如果区域是东部，则改变的方向是从 ALL_CITIES_EAST 向 ALL_CITIES_WEST

[0353] ○如果区域是西部，则改变方向是从 ALL_CITIES_WEST 向 ALL_CITIES_EAST

[0354] ○如果区域是其他，则进行手动确定

[0355] 也可以通过直接应用技术来重新同步行。在一个实施例中，这个技术包含下述内容：

[0356] -对于每个预期的改变读取 RQ

[0357] -使用列值、关键字信息和操作类型，使用该数据库可以获得的本地方法来更新源或者目标表

[0358] 对于 SQL 数据库，这涉及下面的步骤：

[0359] -对于丢失插入，建立下面的语法：

[0360] ○ 插 入 <table>(<column1>[,<column2>...]) 值 (<columnvalue1>[,<column value2>...])

[0361] ○示例：向 PRODUCT（名称，价格）插入值（“汽车”,32000)

[0362] -对于丢失更新，建立下面的语法：

[0363] ○更新 <table>，设置 <column1> = <column value1>[,<column2> = <column value2>]，其中，<key column1> = <key column1value>[并且 <key column2> = <key column2 value>]

[0364] ○示例：更新 PRODUCT，设置价格= 32000，其中，名称=“汽车”

[0365] -对于丢失删除，建立下面的语法：

[0366] ○从 <table> 删除，其中，<key column1> = <key column1 value>

[0367] [并且 <key column2> = <key column2 value>]

[0368] ○示例：从 PRODUCT 删除名称=“汽车”

[0369] <column 1>...<column n> 是在表中的列的列表，可以在原始比较步骤中获得其值。<column value 1>...<column value n> 是对应的列值。

[0370] <key column 1>...<key column n> 是包括比较中使用的唯一关键字的列的列表。<key value 1>...<key value n> 是对应的值。

[0371] 在一个实施例中，通过下述方式来重新同步行：

[0372] -建立 SQL 语句语法

[0373] -向数据库的 SQL 执行程序发送 SQL 语句

[0374] -或者独立地或者作为一个更大组的操作的一部分来进行每个操作

[0375] 其他数据库支持与 SQL 类似的功能以删除、插入和更新特定行。这些包括各种 ISAM 方法和不同的数据库 API。用于应用行改变的相同的一般方法适用于这些类型的表。用于设置特定列或者字段并且更新这些类型的数据库的方法是公知和可以获得的。

[0376] 在许多公共环境和架构中，两个数据库使用数据库复制方法在任何给定时间保持紧密同步。在这些类型的情况中，当在数据源应用行改变时，其通常通过复制机制而在某个延迟（从毫秒到几天）后被应用到目标。

[0377] 以这种方式应用的数据向重新同步机制带来了挑战。考虑下面的事件集：

[0378]

时间	事件	源关键字,列值	目标关键字,列值
1	行同步	100,汽车	100,汽车
2	源行更新	100,卡车	100,汽车
3	比较检测丢失更新条件	100,卡车	100,汽车
4	通过复制而更新的目标行	100,卡车	100,卡车
5	源行更新	100,飞机	100,卡车
6	通过复制更新的目标行	100,飞机	100,飞机
7	重新同步应用旧源行值	100,飞机	100,卡车

[0379] 表 31

[0380] 在这个示例中,所述比较处理在时间 3 正确地检测失步条件(复制还没有向目标行应用从汽车向卡车的改变)。在时间 4,在时间 2 的源行上的更新最后被应用到目标。随后的行为更新源行值(时间 5),并且复制在时间 6 应用数据的新(正确)版本。

[0381] 但是,在时间 7,重新同步步骤向目标应用过时的改变,错误地以更旧的版本重写最新的记录版本。使得在目标行中的数据不精确,至少直到下一次比较和重新同步发生,或者直到再次更新和复制行而不干扰比较和重新同步行为。

[0382] 本发明的一个方面保证在复制环境中的有序改变应用。因为数据库复制可以引起向给定行的新信息的居间更新,因此所述系统需要协调重新同步行为以便防止异常。用于完成此的必要机制使得复制系统自身检测和应用改变。

[0383] 复制机制典型地以对源数据库执行操作的顺序来检测所述操作,并且在目标数据库上以相同的顺序来应用那些改变。这保证了目标行将(最后,在考虑了复制潜伏时间后)包含精确的信息。

[0384] 一种能有序应用改变的机制是源行更新技术。这样的技术可以被实现如下:

[0385] - 从 RQ 获得失步的行的关键字

[0386] - 使用所述关键字值来将在行中的所有列更新为其本身

[0387] - 这触发数据库将当前的行值记录到数据库事务日志或者类似的机制(诸如由触发器保存的改变日志)中

[0388] - 随后,所述复制系统读取和解译所述改变事件,并且将所述改变复制到目标数据库中

[0389] 考虑下面的示例：

[0390]

时间	事件	源关键字,列值	目标关键字,列值
1	同步的行	100,汽车	100,汽车
2	源行更新	100,卡车	100,汽车
3	比较检测丢失更新条件	100,卡车	100,汽车
4	由复制更新的目标行	100,卡车	100,卡车
5	源行更新	100,飞机	100,卡车
6	重新同步将源行更新到其本身（使得记录当前行值）	100,飞机	100,卡车
7	复制检测在日志中的行值,并且将它们应用到目标	100,飞机	100,飞机

[0391] 表 32

[0392] 注意,与直接应用技术成对比,在此应用行的精确版本,因为复制发现和应用了行的正确的版本。也注意,复制机制必须具有下面的附加能力：

[0393] -重新同步事件总是对于行的更新（将值设置为其本身）;因此,复制机制必须能够检测是否在目标处实际存在行,并且如果在目标处不存在行则将所述行应用为插入

[0394] 本发明的一个实施例使用源行标记技术。这种技术被使用,因为源行更新技术以下面的方式受限：

[0395] -不能通过更新源行来重新建立丢失删除记录——因此,需要另一种机制来建立和处理删除

[0396] -对于行的更新会使得其他非计划中的事件发生,诸如数据库触发器

[0397] -没有容易的方式来在由重新同步机制引起什么行为和由应用引起什么——或者在源端或者在目标通过复制——之间描绘。

[0398] 源行标记技术是这些限制的原因。这种技术也与复制结合工作以保证以适当的顺序来应用改变。所述技术工作如下：

[0399] -(从 RQ) 获得失步的行的关键字

[0400] -如果操作类型是丢失插入或者丢失更新

[0401] ○使用所述关键字值,从源行选择所有的列,同时锁定那行（作为示例,使用 SELECT FOR UPDATE（选择以更新）或者 READ WITH LOCK（以锁定来读取））

[0402] ○至少向 RESYNC_MARKER（重新同步标记）表中记录列值以及表名称和操作类型

[0403] -如果操作是丢失删除

[0404] ○使用所述关键字值，从源行选择所有的列，同时锁定那行（作为示例，使用 SELECT FOR UPDATE（选择以更新）或者 READ WITH LOCK（以锁定来读取））

[0405] ○如果所述行存在，则不进行任何事情

[0406] ○如果所述行不存在，则至少向 RESYNC_MARKER（重新同步标记）表中记录关键字列值以及表名称、操作类型或者当前日期时间

[0407] -进行数据库事务（或者独立地或者以重新同步操作为组）

[0408] -解锁被选择或者以锁定读取的任何行

[0409] 可以使用具有下面的特性的 RESYNC_MARKER（重新同步标记）表来实现本发明：

[0410] -必须将在表格内建立的行记录到数据库事务日志或者等价物（复制改变数据源）——所述事务日志必须与由下层源表格使用的事务日志相同

[0411] -其可以存储不同表结构的数据——其不是与下层源表相同的结构，只要其可以以某种方式来存储源表的列值，诸如将它们串联在一起成为一个大列中

[0412] 所述复制机制的附加要求：

[0413] -其必须跟踪对于 RESYNC_MARKER 表进行的改变

[0414] -其必须知道 RESYNC_MARKER 表的目的，并且与其对于其他表格不同地解译其内容——即其将每个 RESYNC_MARKER 行的内容（其很可能是多个列向一个单列中的封装）翻译为适当的源表格式（将所述单列解封装为期望的多个列）

[0415] -必须能够获得每个 RESYNC_MARKER 行的内容，并且在内部重建对应的行/列值和等价的改变记录

[0416] -其不必处理在 RESYNC_MARKER 表中的删除行

[0417] 在对于特定行进行这个处理后，所述复制机制可以原样将所述数据处理为下层行的通常改变记录。

[0418] 最后，因为仅到复制已经处理了才需要 RESYNC_MARKER 行，因此所述复制机制可以定期地删除不再需要的 RESYNC_MARKER 行。

[0419] 示例：

[0420]

时间	事件	源关键字，列值	目标关键字，列值
1	同步的行	100，汽车	100，汽车
2	源行更新	100，卡车	100，汽车
3	比较检测丢失更新条件	100，卡车	100，汽车
4	由复制更新的目标行	100，卡车	100，卡车
5	源行更新	100，飞机	100，卡车

时间	事件	源关键字，列值	目标关键字，列值
6	重新同步选择（具有锁定的）源行的当前值，然后使用那些值来建立 RESYNC_MARKER 行	100，飞机	100，卡车
7	重新同步提交并解锁 RESYNC_MARKER 行	100，飞机	100，卡车
8	源行更新	100，船只	100，卡车
9	由复制处理的重新同步标记行	100，船只	100，飞机
10	由复制处理的源行更新	100，船只	100，船只

[0421] 表 33

[0422] 注意在这个示例中，在时间 8 的源行更新必须等待对于在时间 6 建立的源行读取释放锁定，以保证在数据库日志中记录 RESYNC_MARKER 行。这保证了将以正确顺序执行在时间 9 和 10 的步骤。

[0423] 上述的讨论描述了一种技术，其中，在表比较披露失步行的情况下，使用复制机制来促进数据库的正在进行的同步以及那些数据库的重新同步。假定复制机制也可以支持如上所述的异类和重叠情况的同步，并且假定所述机制支持上述的源行标记技术，所述机制也可以用于同步异类表和 / 或重叠的行 / 列集。这是因为下述事实：一旦复制系统已经正确地解译源行标记数据，则随后的操作对于复制系统就好像它们产生在下层的源表本身中。因此，已经由复制机制容纳的任何异类或者重叠的数据差别被明显地扩展到组合的重新同步 / 复制机制。

[0424] 所述复制机制也可以用于建立由重新同步处理建立的行的改变的审核跟踪。这允许重新同步机制的用户可以在由重新同步产生的行和由复制或者外部应用产生的行之间描述，

[0425] 由复制机制执行的下面的步骤在用于特定行的目标数据库事务日志中建立审核记录：

[0426] - 获取 RESYNC_MARKER 行

[0427] - 解译行数据，并且将其转换来模仿下层的源行

[0428] - 更新目标行

[0429] - 在相同数据库事务的环境中，在目标数据库中建立 RESYNC_APPLY_MARKER 行（其唯一的目的是将事务识别为已经被重新同步行为引起）

[0430] 一旦在目标数据库事务日志中建立记录，则存在用户访问数据的两种方式：

[0431] 1. 通过查看 RESYNC_APPLY_MARKER 表格本身；

[0432] 2. 通过从目标数据库事务日志挖掘事务数据；因为对 RESYNC_APPLY_MARKER 表格的操作在所述事务中出现，因此可以假定在所述事务中的所有其他操作一定已经通过重新同步行为而产生

[0433] 上述的第二步骤也具有不需要列信息的冗余存储的优点，因为其被包含在事务日志记录中（通过应用该行到目标数据库中的优点）。

[0434] 本发明的一个实施例涉及具有计算机可读介质的计算机存储产品，在所述计算机可读介质上具有计算机代码，用于执行各种计算机实现的操作。所述介质和计算机代码可以是为了本发明的目的而特殊设计和构造的那些，或者它们可以是对于在计算机软件领域中的那些技术人员公知和可以获得的种类。计算机可读介质的示例包括但是不限于：磁介质，诸如硬盘、软盘和磁带；光介质，诸如 CD-ROM 和全息设备；磁光介质，诸如光磁软盘；以及硬件设备，其被特殊配置来存储和执行程序代码，诸如专用集成电路（“ASIC”）、可编程逻辑器件（“PLD”）以及 ROM 和 RAM 器件。计算机代码的示例包括诸如由编译器产生的机器代码与包含由计算机使用解译器执行的更高层代码的文件。例如，可以使用 Java、C++ 或者其他面向对象的编程语言和开发工具来实现本发明的实施例。可以在替代机器可执行的软件指令或者与其组合的硬连线电路中实现本发明的另一个实施例。

[0435] 为了解释，上述的说明使用特定术语来提供对本发明的彻底理解。但是，对于本领域内的技术人员显然，不需要具体细节来实践本发明。因此，本发明的具体实施例的上述说明被提供来用于说明和描述。它们不意欲是穷尽的，或者将本发明限定到所公开的精确形式；显然，基于上述教导，有可能进行许多修改和变化。所述实施例被选择和描述以便最佳地解释本发明的原理及其实际应用，它们由此使得本领域内的技术人员能够最佳地利用本发明，并且具有各种修改的各个实施例适合于所考虑的特定使用。旨在以所附的权利要求及其等价物限定本发明的范围。

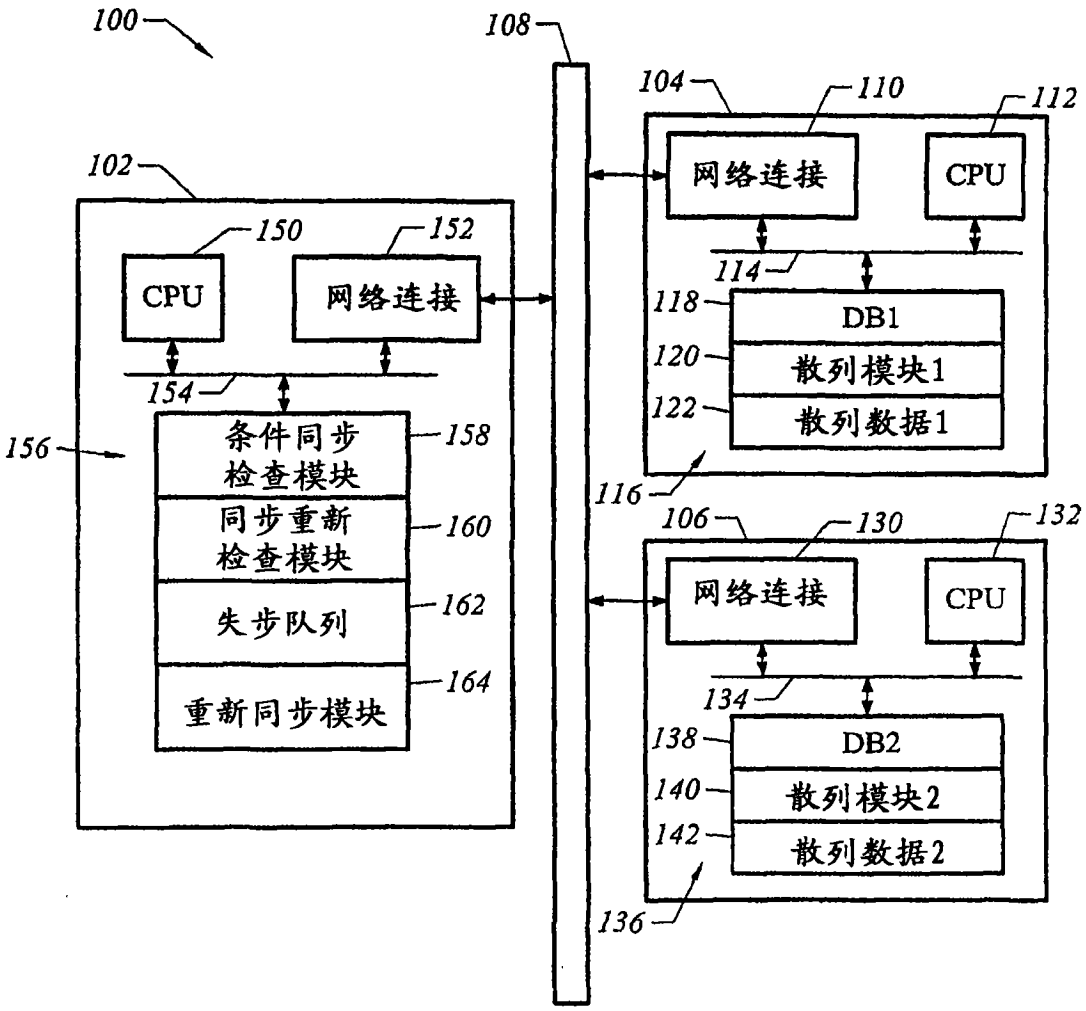

100
108
104
110
112
网络连接
CPU
102
150
152
CPU
网络连接
114
118
DB1
120
散列模块1
122
散列数据1
154
156
条件同步检查模块
158
同步重新检查模块
160
失步队列
162
重新同步模块
164
116
106
130
132
网络连接
CPU
134
138
DB2
140
散列模块2
142
散列数据2
136

图1

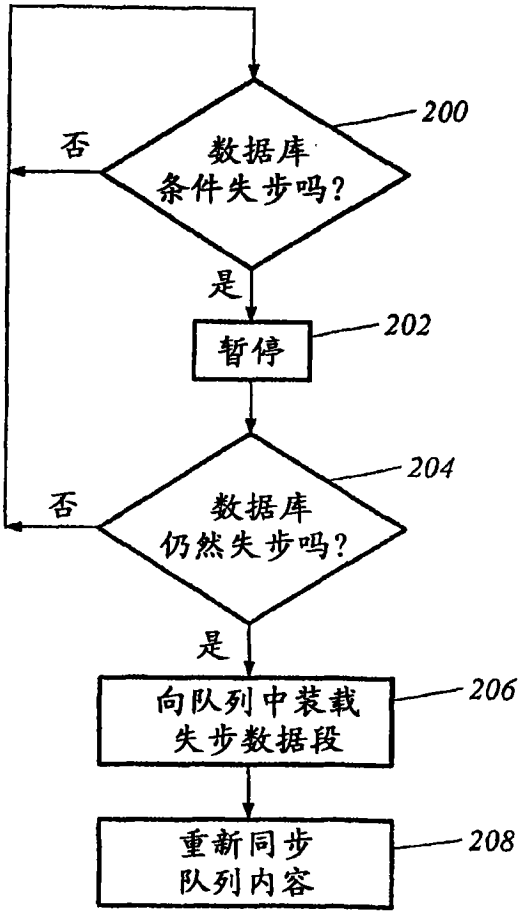

200
数据库
条件失步吗?
否
是
202
暂停
204
数据库
仍然失步吗?
否
是
206
向队列中装载
失步数据段
208
重新同步
队列内容

图2