文章编号:1000 - 582X(2005)05 - 0080 - 06

多数据源数据仓库实体化视图维护与下查*

陈金玉,王启星
(重庆大学 1. 自动化学院;2. 计算机学院,重庆 400030)

摘 要:研究了多源单视图下数据仓库实体化视图联机维护与下查一致性问题,并提出普适性强一致收敛维护算法 M-Glide。算法引入动作列表来控制数据仓库收到的信息顺序,采用版本控制、补偿思想和应答机制来协调源数据库与数据仓库间的数据更新,从而保证了数据仓库视图维护与下查的一致性,并通过一个示例说明了该算法在实际中的具体运用。

关键词:算法 M-Glide;数据仓库;实体化视图;补偿技术; OLAP 查询
中图分类号:TP311 **文献标识码**:A

在数据仓库联机维护技术中,实体化视图(Materialized views)的联机维护是一个关键技术。是指在数据仓库为用户提供服务的同时,当数据库中的原始数据发生改变时,实时地将这种变化反映到数据仓库中,使相应的实体化视图得到及时的刷新。同时,由于数据仓库中存储的只是部分原始数据拷贝和实体化视图集合,它并不能完全满足用户的所有查询。对于某些 OLAP 查询(如下钻等),数据仓库必须通过访问源数据库才能给出最终的查询结果,如何保证这类查询的一致性,也是一个值得研究的课题。

以 Stanford 提出的二层数据仓库体系结构为例,如图 1 所示。每个源对应的监视器(Monitor)负责收集源的变化并通知数据仓库,数据仓库接到通知后,向数据源发出数据查询要求。数据源上的集成器(Integrator)用来接收、转化和集成来自源的数据,在必要时增加信息如时间戳等,并应答数据仓库进行集成存储。可以看出,从源上的数据变化到数据仓库视图数据的修改,要经过通知、查询及应答几个步骤,一旦中间发生信息延迟,使数据仓库收到源变化顺序与各源之间的操作顺序不一致,就会导致数据的不一致。

对于单源单视图维护方面,文献[2 - 5]等进行了充分的研究,其中文献[5]采用版本控制、补偿思想和应答机制来协调数据源与数据仓库间的数据更新,提出了完全一致收敛算法 Glide. 算法保证了仓库视图

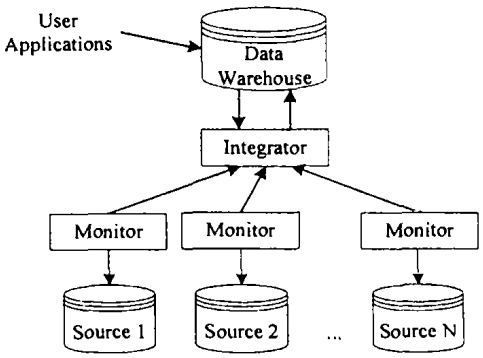

图 1 数据仓库结构模型

维护和 OLAP 查询的一致性,并修正了文献[2 - 3]中存在的一些不足。对于多源实体化视图的维护方面,文献[6 - 7]提出了一个多源补偿算法 Strobe,其主要思想是当发给数据源的查询还没有返回结果就又收到了新的更新消息时,就记录下这些更新事务,并在以后发送一个补偿查询来消除上面的不一致性。为避免算法 Strobe 在维护集合数据组上的开销,文献[8]提出了其强一致收敛改进算法 Strobe *,核心思想是保证 AL 中记载的操作序列与集成器接收的消息的顺序一致。但是,以上 2 个算法都有一个假定,即算法建立在源表存在关键属性,而在实际的数据库环境上,有些表可能没有设定关键字。当以这些表作为数据仓库源表时,显然以上的 2 种维护算法都存在不足,同时这 2 个算法也没有考虑数据下钻问题。如何找到更为一般性

* 收稿日期:2004 - 12 - 08
基金项目:国家教育部博士生基金资助项目(98061117)
作者简介:陈金玉(1969 -),男,福建仙游县人,重庆大学讲师,博士,主要从事数据仓库理论与工程的研究。

的维护算法，是此次研究的重点。

笔者研究的是算子 SPJ(Select-Project-Join)下多数据源单视图的数据仓库视图维护与下查一致性问题，论文在文献[5]的基础上，提出了多源单视图普适性维护算法 M-Glide. 该算法引入一个动作列表来控制数据仓库收到的信息顺序，采用版本控制、补偿思想和应答机制来协调数据源与数据仓库间的数据更新，保证了仓库视图维护和 OLAP 查询的一致性。

1　相关定义

定义 1　现设数据仓库实体化视图定义如下：

$$V = \Pi_{\text{proj}}(\sigma_{\text{cond}}(r_1 \times r_2 \times \cdots \times r_n)),$$

其中 proj 为属性名，cond 为布尔表达式，$r_1, \cdots, r_n$ 为取自不同数据源的关系表。显然，以上定义与 Select, Project 和 Join 的关系代数表达式是等价的。

定义 2　设数据仓库视图如定义 1 所示，若在源库关系 r_i 上增加一元组 r_i^0，则记相应的仓库查询为：

$$Q_i = \Pi_{\text{proj}}(\sigma_{\text{cond}}(r_1 \times \cdots \times r_{i-1} \times + r_i^0 \times r_{i+1} \times \cdots \times r_n)),$$

即在相应的元组上添加符号“+”；若在源库关系 r_j 上删除一元组 r_j^0，则记相应的仓库查询为：

$$Q_j = \Pi_{\text{proj}}(\sigma_{\text{cond}}(r_1 \times \cdots \times r_{j-1} \times - r_j^0 \times r_{j+1} \times \cdots \times r_n)),$$

即在相应的元组上添加符号“-”；且应答结果符号满足以下 2 个规则：

t	$\sigma_{\text{cond}}(t)$	$\prod_{\text{proj}}(t)$
+	+	+
-	-	-

；

t_1	t_2	$t_1 \times t_2$
+	+	+
+	-	-
-	-	+
-	+	-

.

定义 3　设有数据仓库查询表达式为

$$Q = \Pi_{\text{proj}}(\sigma_{\text{cond}}(\tilde{r}_1 \times \tilde{r}_2 \times \cdots \times \tilde{r}_n)),$$

其中 $\tilde{r}_i$ 为源数据库关系 r_i 或为 r_i 的一元组。U_i 为源库关系 r_i 的一个增加(insert)或删除(delete)操作，并记 tuple(U_i) 为相应的元组，当 U_i 为增加操作时，元组符号取“+”，否则为“-”. 修改(*modify*) 相当于先进行删除操作，而后进行增加操作。定义公式 $Q < U_i >$ 为：

1) 当 $\tilde{r}_i$ 为关系 r_i 时：

$$Q < U_i > = \prod_{\text{proj}}(\sigma_{\text{cond}}(\tilde{r}_1 \times \cdots \times \tilde{r}_{i-1} \times \text{tuple}(U_i) \times \tilde{r}_{i+1} \times \cdots \times \tilde{r}_n));$$

2) 当 $\tilde{r}_i$ 为关系 r_i 一元组时：$Q < U_i > = \phi$，特别地，当 $\tilde{r}_i \equiv r_i (i = 1, 2, \cdots, n)$ 时，记 $V < U_i > = Q < U_i >$.

定义 4　数据仓库的每一状态都反映了数据源执行全局串行调度达到的合法状态，并且数据仓库的状态变化顺序与数据源执行全局串行调度的操作顺序一致，则称视图收敛级别为强一致；除满足强一致外，数据源的每一个状态在数据仓库中都有一个与之对应的状态，则称视图收敛级别为完全一致。

定义 5　设源数据库关系 R 有如下的属性$\{att_1, att_2, \cdots, att_n\}$，其中如定义 1 所设的投影属性为$\{att_1, att_2, \cdots, att_k\}$，则称以下关系为关系 R 的扩展模式 R^*：

$\{count_1, count_2, insert, delete, att_1, att_2, \cdots, att_k\}$，

其中 $count_1, count_2$ 属性默认值为 0，insert 属性、delete 属性默认标记为 F.

定义 6　以下给出仓库维护的消息类型

DB_up：DB 端发出的数据变化通知，它包括 3 个参数：insert(r, t)：在关系 r 插入元组 t；delete(r, t)：从关系 r 中删除元组 t；modify(r, t_1, t_2)：先在关系 r 中删除元组 t_1，然后在关系 r 中插入元组 t_2；

DW_qu：DW 端视图维护进程发出的查询通知，它返回的结果用于实体化视图的维护；

DW_olap：DW 端发出的 OLAP 下钻通知，它返回的结果将直接提交给 OLAP 应用；

DB_ans：DB 端发出的 DW_qu 查询后的返回结果；

DB_olap：DB 端发出的完成 DW_olap 查询后的返回结果；

DW_ack：DW 端发出的维护完成确认通知，DB 端在收到这一消息后，将对源库中版本进行切换；

DB_ack：DB 端收到确认通知，并对源库中版本进行切换。

2　多源视图维护算法 M-Glide 研究

应该讲，多源环境下的视图维护和单源单视图的维护是有本质区别的。主要在于，在单源环境下，数据仓库收到查询返回结果和仓库发送出去的查询顺序是一致的，但在多源环境下这种偏序方向将不再存在。

论文在单源单视图维护算法 Glide 的基础上，在数据仓库端加入对事务提交顺序进行控制的方法，并运用到多源视图维护环境下，从而提出相应的多源维护算法 M-Glide 如下：

在每个数据源端

调用文献[5] 相应算法 Glide-DB。

在数据仓库端

初始化：$k_i = 0, i = 1, 2, \cdots, n$; COLLECT = ϕ; $AL = \phi$; $UQS = \phi$

1) DW_up_i

① 接收到 DB 端发来的修改信息 U_i；

② 将相应的修改操作插入队列 AL 的末尾：

$$AL \leftarrow AL + \{U_i\};$$

③ 令查询

$Q_i = V < U_i > + \sum_{Q_j \in UQS} Q_j < U_i >$,

④ $UQS \leftarrow UQS + Q_i$;

⑤ 把查询 Q_i 发给 DB 端,并触发 DB_qu_i 事件。

2) DW_ans_i:

① 收到查询计算值 A_i;

② 用查询计算值 A_i 替换 AL 中相应的修改信息;

③ While (AL 队列中的值为查询计算值,而不是为操作修改信息)

④ {

⑤ $AL \leftarrow AL - \{A_i\}$;

COLLECT = COLLECT + A_i,

⑥ $UQS \leftarrow UQS - Q_i$;

⑦ 在对应的记录源数据关系操作数 k_i 加 1;}

⑧ if UQS = ϕ and AL = ϕ;

⑨ then { $MV \leftarrow MV$ + COLLECT;

⑩ 向 DB 端发送确认 DW_ack 及全部 k_i 的数值;

⑪ COLLECT $\leftarrow \phi$;

⑫ $k_i \leftarrow 0, i = 1,2,\cdots,n$;};

⑬ else do nothing.

3) DW_olap_i

① 接收到用户的 OLAP 查询;

② 记录此时 k_i 的数值,并向 DB 端发送下钻请求 O_i^*,并触发 DB_olap 事件。

其中 AL 为控制查询返回值的提交顺序的一个临时队列。可以看出,算法 M-Glide 与算法 Glide 最大的不同在于算法 M-Glide 要利用队列 AL 来控制提交时机,它保证了计算值提交的顺序和数据仓库收到修改操作的顺序是一致的。

由于算法 M-Glide 在补偿方面与算法 Glide 相似,故算法 M-Glide 是收敛的。又因为在队列 AL 的控制下,部分查询计算值是同时提交到 COLLECT 的,于是其视图的状态与仓库收到的操作集偏序的方向是相同的,即有以下结论:

定理 算法 M-Glide 是强一致收敛的。

3 算法 M-Glide 的一个典型示例

为了更好地了解 M-Glide 算法,现通过一个典型示例来说明算法 M-Glide 在实际应用中是怎样进行维护且如何解决"下钻"查询的一致性问题的。

例 设关系表 r_1,r_2 及 r_3,分别来自不同的源 x, y 和 z, 其关系如下所示:

r_1:

A	B
1	2
3	3

r_2:

B	C
2	3

r_3:

C	D
3	4

设数据仓库实体化视图定义为:

$$V = \Pi_{A,B,C,D}(r_1 \infty r_2 \infty r_3).$$

显然初始视图 $MV = \{[1,2,3,4]\}$. 假设仓库端收到如下顺序的操作:

1) U_1 = Insert(r_3, [3,5]);2) U_2 = Delete(r_1, [1,2]);3) U_3 = Znsert(r_1, [1,3]; 4) U_4 = Delete(r_3,[3,4]).

同时又设在对数据仓库进行实体化视图维护期间启动了一个 OLAP 查询 Q 如下:

Q: select A, B ;

From r_1;

Where B in (select B from V).

不妨假设 OLAP 查询发生在数据仓库收到源数据库发来的查询计算结果 A_1^2 之后。可将 OLAP 查询 Q 分为 2 个按顺序执行的子查询:Q' 和 Q^*,又设 Q' 返回的结果为 α.

Q': select B from V;

Q^*: select A, B from r_1 where B in α.

以下将算法 M-Glide 应用到数据仓库的维护中,并按照事件发生的时间顺序来说明算法 M-Glide 的工作。

1) 初始化源数据库端的扩展模式 R_1^*,R_2^*,R_3^* 如下所示:

R_1^*:

$count_1$	$count_2$	Insert	Delete	A	B
0	0	F	F	1	2

R_2^*:

$count_1$	$count_2$	Insert	Delete	B	C
0	0	F	F	2	3

R_3^*:

$count_1$	$count_2$	Insert	Delete	C	D
0	0	F	F	3	4

2) 在源 z 端进行 insert(r_3,[3,5]) 操作;向 DW 端发送消息 $DB\_up(U_1)$;记录扩展模式 R_1^*,R_3^* 其中模式 R_3^* 发生变化,如下所示:

R_3^*:

$count_1$	$count_2$	Insert	Delete	C	D
0	0	F	F	3	4
0	0	T	F	3	5

3) DW 端收到 $DB\_up(U_1)$,由算法 M-Glide 得到查询 $Q_1 = \Pi_{A,B,C,D}(r_1 \infty r_2 \infty [3,5])$;由多源分解查询函数,向源 y 发送 $DW\_qu(Q_1^1)$,其中 $Q_1^1 = \Pi_{B,C,D}(r_2 \infty [3,5]$;于是得控制队列 $AL = \{U_1\}$,且 $UQS = \{Q_1\}$;

4) 在源 x 端进行 Delete(r_1,[1,2]) 操作;向 DW 端发送消息 $DB\_up(U_2)$;记录扩展模式 R_1^*,R_3^*,其中模式 R_1^* 发生变化,如下所示:

R_1^*:

$count_1$	$count_2$	Insert	Delete	A	B
0	1	F	T	1	2

5) DW 端收到 DB_up(U_2)；由算法 M-Glide 得到查询 Q_2 如下：

$Q_2 = \Pi_{A,B,C,D}(-[1,2] \infty r_2 \infty (r_3 + [3,5]))$

由多源分解查询函数，向源 y 发送 DW_qu(Q_2^1)，其中 Q_2^1 如下所示：

$Q_2^1 = \Pi_{A,B,C}(-[1,2] \infty r_2)$.

又由算法 M-Glide 得

$UQS = \{Q_1, Q_2\}, AL = \{U_1, U_2\}$；

6) 在源 x 端进行 Insert(r_1,[1,3]) 操作；向 DW 端发送消息 DB_up(U_3)；记录扩展模式 R_1^*，R_3^*，其中模式 R_1^* 发生变化，如下所示：

R_1^*：

$count_1$	$count_2$	Insert	Delete	A	B
0	1	F	T	1	2
2	0	T	F	1	3

7) DW 端收到 DB_up(U_3)；由算法 M-Glide 得到查询 Q_3 如下：

$Q_3 = \Pi_{A,B,C,D}([1,3] \infty r_2 \infty (r_3 + [3,5]))$.

由多源分解查询函数，向源 y 发送 DW_qu(Q_3^1)，其中 Q_3^1 如下所示：

$Q_3^1 = \Pi_{A,B,C}([1,3] \infty r_2)$.

又由算法 M-Glide 得

$UQS = \{Q_1, Q_2, Q_3\}, AL = \{U_1, U_2, U_3\}$；

8) DB 端进行 Delete(r_3,[3,4]) 操作；向 DW 端发送消息 DB_up(U_4)；记录扩展模式 R_1^*，R_3^*，其中模式 R_3^* 发生变化，如下所示：

R_3^*：

$count_1$	$count_2$	Insert	Delete	C	D
0	2	F	T	3	4
1	0	T	F	3	5

9) DW 端收到 DB_up(U_4)；由算法 M-Glide 得到查询 Q_4 如下：

$Q_4 = \Pi_{A,B,C,D}((r_1 - [1,2] + [1,3]) \infty r_2 \infty -[3,4]$.

由多源分解查询函数，向源 y 发送 DW_qu(Q_4^1)，其中 Q_4^1 如下所示：

$Q_4^1 = \Pi_{B,C,D}(r_2 \infty -[3,4])$.

又由算法 M-Glide 得

$UQS = \{Q_1, Q_2, Q_3, Q_4\}$,

$AL = \{U_1, U_2, U_3, U_4\}$；

10) 源 y 收到查询 Q_1^1，即收到 DW_qu(Q_1^1)，应用 Glide-DB 算法，计算出 $A_1^1 = \{[2,3,5],[3,3,5]\}$，然后向仓库端发送 DB_ans($A_1^1$)；

11) 源 y 收到 DW_qu(Q_2^1)，应用 Glide-DB 算法，计算出 $A_2^1 = \{-[1, 2, 3]\}$；然后向仓库端发送 DB_ans(A_2^1)；

12) 源 y 端收到 DW_qu(Q_3^1)，应用 Glide-DB 算法，计算出 $A_3^1 = \{[1,3,3]\}$；然后向仓库端发送 DB_ans(A_3^1)；

13) 源 y 端收到 DW_qu(Q_4^1)，应用 Glide-DB 算法，计算出 $A_4^1 = \{-[2,3,4], [3,3,4]\}$；然后向仓库端发送 DB_ans($A_4^1$)；

14) DW 端收到 DB_up(A_1^1)，则由 DW 端向源 x 发出子查询如下：

$Q_1^2 = \Pi_{A,B,C,D}(r_2 \infty ([2,3,5] + [3,3,5]))$；

15) DW 端收到 DB_up(A_2^1)，则由 DW 端向源 z 发出子查询如下：

$Q_2^2 = \Pi_{A,B,C,D}(-[1,2,3] \infty (r_3 + [3,5]))$；

16) DW 端收到 DB_up(A_3^1)，则由 DW 端向源 z 发出子查询如下：

$Q_3^2 = \Pi_{A,B,C,D}([1,3,3] \infty (r_3 + [3,5]))$；

17) DW 端收到 DB_up(A_4^1)，则由 DW 端向源 x 发出子查询如下：

$Q_4^2 = \Pi_{A,B,C,D}((r_1 - [1,2] + [1,3]) \infty (-[3,3,4] - [2,3,4]))$.

假定由于通信的原因，仓库收到自各源点返回值的顺序为：$A_2^2, A_3^2, A_1^2, A_4^2$，类似可以得到，其它相应的顺序也成立；

18) DW 端收到 DB_ans(A_2^2)；由算法 M-Glide 得：$A_2 = \{-[1,2,3,4], -[1,2,3,5]\}$；

$UQS = \{Q_1, Q_3, Q_4\}$，且得控制队列 AL 为：

$AL = \{U_1, \{-[1,2,3,4], -[1,2,3,5]\}, U_3, U_4\}$；

19) W 端收到 DB_ans(A_3^2)；由算法 Glide-DW 得：$A_3 = \{[1,3,3,4],[1,3,3,5]\}$，$UQS = \{Q_1, Q_4\}$，且得控制队列 AL 为：$AL = \{U_1, \{-[1,2,3,4], -[1,2,3,5]\}, \{[1,3,3,4], [1,3,3,5]\}, U_4\}$；

20) DW 端收到 DB_ans(A_1^2)；由算法 Glide-DW 得：$A_1 = \{[1,2,3,5]\}$，$UQS = \{Q_4\}$，且有 $AL = \{\{[1,2,3,5]\}, \{-[1,2,3,4], -[1,2,3,5]\}, \{[1,3,3,4], [1,3,3,5]\}, U_4\}$；

21) 由于此时 AL 中值 U_4 之前都为确定值，由算法 M-Glide，得

① COLLECT = $\{[1,2,3,5], -[1,2,3,4], -[1,2,3,5], [1,3,3,4], [1,3,3,5]\}$；

② AL = $\{U_4\}$；

③ $k_1 = 2; k_3 = 1$；

22) 用户启动 OLAP 查询 Q，首先执行 Q'，由于视

图 V 有:$V = MV + COLLECT = \{[1,3,3,4], [1,3,3,5]\}$,得 $A = \{3\}$ 后,OLAP 向 DB 端发出下钻查询 DW_olap(Q^*);

23) DW 端收到 DB_ans(A_4^1);由算法 Glide-DW 得,

① $A_4 = \{-[1,2,3,4], [1,2,3,4], -[1,3,3,4]\}$;

② $AL = \{-[1,3,3,4]\}$;

③ $UQS = \phi$;

24) 由于 AL 中值为确定值,则有

①COLLECT = $\{[1,2,3,5], -[1,2,3,4], -[1,2,3,5]$,

$[1,3,3,4], [1,3,3,5], -[1,3,3,4]\}$;

② $AL = \phi$;

③ $UQS = \phi$;

④ $k_1 = 2; k_3 = 2$;

⑤ MV ← MV + COLLECT = $\{[1,3,3,5]\}$;

/*(视图维护结果正确)*/

⑥ 向 DB 端发送版本切换消息 DW_ack 及 k_1, k_3 的值;

⑦ $k_1 \leftarrow 0$; $k_3 \leftarrow 0$; COLLECT ← ϕ;

$UQS \leftarrow \phi$;

25) DB 端收到 DW_olap(Q^*),因为此时 $k_1 = 2$; $k_3 = 1$,则由算法 Glide-DB 得,参与计算的关系 r_1 元组为 $\{[1,3]\}$,于是得 $A^* = \{[1,3]\}$;向 DW 发送消息 DB_ans(A^*);

26) DB 端收到消息 DW_ack,进行版本切换,得扩展模式 R_1^*, R_3^* 如下:

R_1^*:

$count_1$	$count_2$	Insert	Delete	A	B
0	0	F	F	1	3

,

R_3^*:

$count_1$	$count_2$	Insert	Delete	C	D
0	0	F	F	3	5

;

27) 在 DW 端收到消息 DB_ans(A^*),将 $A^* = \{[1,3]\}$ 提交给 OLAP 应用。

至此可以看到,算法 M-Glide 不仅保证了 DW 端实体化视图的完全一致维护,而且保证了用户对数据仓库的下钻查询也得到一致的结果。特别地,若在第 20 步之后,对数据库 DB 进行如下的操作:5) insert(r_1, [1,4]); 6) delete(r_1, [1,3]). 此时 DW 端已发出版本切换消息,而 DB 端却尚未收到消息时。经进数据修改后,DB 端扩展模式 R_1^*, R_3^* 如下:

R_1^*:

$count_1$	$count_2$	Insert	Delete	A	B
0	1	F	T	1	2
2	4	T	T	1	3
0	3	T	F	1	4

;

R_3^*:

$count_1$	$count_2$	Insert	Delete	C	D
0	2	F	F	3	4
1	0	T	F	3	5

.

利用算法 Glide-DB 中,对 DB_ack 消息的处理方案,当对数据库 DB 进行版本切换后,其扩展模式 R_1^*, R_3^* 如下所示。

R_1^*:

$count_1$	$count_2$	Insert	Delete	A	B
0	2	F	T	1	3
1	0	T	F	1	4

;

R_3^*:

$count_1$	$count_2$	Insert	Delete	C	D
0	0	F	F	3	5

;

可以看出,版本切换是光滑进行的,即当数据库事务间隔大于数据库与仓库之间的消息交换时间间隔时,就进行 DB 端的版本切换,从而提高了源数据库端 CPU 的利用率,保证了维护的全天候进行。

4 结 语

讨论了在多源单视图下,数据仓库实体化视图的维护与下查一致性问题。引入了一个动作列表概念,以记录数据仓库收到的信息顺序,提出了解决算法 M-Glide,同时还通过一个典型示例说明了该算法在实际中的具体应用。

关于算法 M-Glide 的效率估计,由于篇幅关系,将在另文加以讨论。

参考文献:

[1] HAMMER J, GARCIA-MOLINA H, WIDOM J, et al. The Stanford Data Warehousing Project[J]. IEEE Data Engineering Bulletion, 1995, 18(2): 41 - 48.

[2] ZHUGE Y, GARCIA-MOLINA H, HAMMER J, et al. View Maintenance in a Warehousing Environment[Z]. Proceedings of ACM SIGMOD Conference, San Jose, CA, 1995.

[3] 李子木,孙利民,周兴铭. 数据仓库联机维护中一致性问题的研究[J]. 软件学报,1999,10(8): 812 - 818.

[4] 陈金玉. 数据仓库实体化视图联机一致性维护研究[D]. 重庆:重庆大学计算机学院,2002.

[5] 陈金玉., 曹长修,张邦礼. 数据仓库视图一致性维护与下查研究[J]. 计算机工程与应用,2003,(26): 12 - 17.

[6] ZHUGE Y, GARCIA-MOLINA H, WIENER J L. The Strobe Algorithms for Multi-source Warehouse Consistency [Z]. Proceedings of the Conference on Parallel and Distributed Information Systems, Miami Beach, FL, 1996.

[7] ZHUGE Y, GARCIA-MOLINA H, WIENER J L. Consistency Algorithms for Multi-source Warehouse View Maintenance[J]. Journal of Distributed and Paralled Databases,

1998,(6): 7 - 40.
[8] 王元珍,熊已兴. 多数据源数据仓库的一致性维护算法——Strobe 算法的改进[J]. 计算机工程与应用,2002,(1):211 - 21.

Materialized View Maintenance and Drill - down at the warehouse span multiple sources

CHEN Jin-yu, WANG Qi-xing
(1. College of Automation; 2. College of Computer, Chongqing University, Chongqing 400030, China)

Abstract: The consistency of view maintenance and drill - down at the warehouse span multiple sources is studied. So the wide strongly consistent algorithm M-Glide is introduced, which imposes an action list to record the updating sequel sets, and uses version control and compensating mechanisms, along with acknowledgementmechanisms, to synchronize the data refreshments between data sources and data warehouse. The authors illustrate the application of the algorithm by a typical example.

Key words: algorithm M-Glide; data warehouse; materialized views; compensate techniques; OLAP query

(编辑　吕赛英)

(上接第 67 页)

Expert System for Artificial Intelligence Blast Furnace Smelting Process

SHI Jin-liang, JIA Bi, YU Qun-wei, LAN Shou-bing
(ChongQing Polytechnic College, Chongqing 400050, China)

Abstract: In order to upgrade traditional industry and realize the automation of furnace condition judgement and standardization of operation, an expert system for various furnace conditions and breakdown judgement and operation guidance on blast furnace are developed based on the theory of artificial intelligence and knowledge engineering. The system is fulfilled through C + + Builder and combined with the actual production picture. The system has efficient and convenient man-machine interface by using menu operation method. The testing and operation of actual production data in simulation system indicate that the system is exact and reliable and can provide both the warning of various furnace breakdown and corresponding operation guidance.

Key words: artificial intelligence; expert system; blast furnace smelting

(编辑　陈移峰)