工程数据库中版本管理技术的研究*)

蒋泽军　徐秋元　（西北工业大学）

聂培尧　（山东财政学院）

摘　要

In this paper,a method which is called the "Forward Version Management"is proposed by the author to hold the multiversion in an engineering database. This method can also generate some new database version in terms of the existing historic versioning data stored in an engineering database. Since several design versions can be stored simultaneously in a database,it is possible to hold the historic data and to support the backtracking in a design process. Therefore, it can meet the needs of "trial and error" procedure in designs.

在CAD/CAM系统中，对于一个工程实体的设计，由于性能描述方法的不同，设计方案的差异，以及对性能要求的差别，在设计的各个阶段将会形成同一工程实体的不同设计版本[1,2]。

采用一般的商用数据库系统来管理这些不同的设计版本，就需要建立多个数据库来管理多个设计

因为他必须要对自己将增加的索引实现封锁。当然，他可以利用系统提供的锁管理机制，也可以编写自己所选择的锁。如果自己编写，则在他的锁机制和系统的锁管理机制之间可能存在不可测试的死锁，这一点是要特别注意的，也是非常重要的。最后，还必须掌握系统提供的恢复机制[3]，或自己规定在各个索引页被强制成固定内存时，所需要的排列顺序，因为在事务失败或系统发生故障时，这一顺序可能被用来保护索引的一致性。

本文首先介绍了最简单通用界面的概念及其设计。SCI为扩展WHYMX存贮系统提供了一种有用的形式。其次，概述了WHYMX存贮系统的层次结构。可扩展存贮系统设计最好采用层次结构，而且为了得到最大可扩展性，每层都应该有一个SCI界面。接着，本文描述了WHYMX的存贮系统中存贮结构及它们的操作算法的扩展方法。该方法无论是对用户，还是对数据库管理员来说，使用都极其方便。与其它面向对象的数据库管理系统中所使用的方法相比，该方法显得更加灵活，并且无需对DBMS重新编译。最后，我们探讨了在扩展存贮结构和存贮算法时，应当注意的几个问题。这些问题给用户扩展存贮系统带来了不便和困难，同时这些问题有待于我们以后进一步解决。

*) 国家自然科学基金资助课题。

参考文献

[1] 毛兆余，"WHYMX的存贮管理"，硕士论文，1990.3

[2] 毛兆余等，"WHYMX存贮系统中的对象和文件管理"，《第九届全国数据库学术会议论文集》，上海，1990.9

[3] 徐庆云等，"面向对象的数据库系统的并发控制"，《第九届全国数据库学术会议论文集》，上海，1990.9

版本。这样不但引起了大量的数据冗余，而且也不便于几个版本之间各部分数据的引用和比较。

下面我们将对如何组织和管理多个设计版本的问题加以讨论。

一、现有的版本管理模型简介

要研究多个设计版本的组织和管理，就有必要研究多个设计版本之间的关系。目前版本之间的关系模型[3]有三种：(1) 线性模型（图1(1)），(2) 树状模型（图1(2)），(3)有向无循环图模型（图1(3)）。

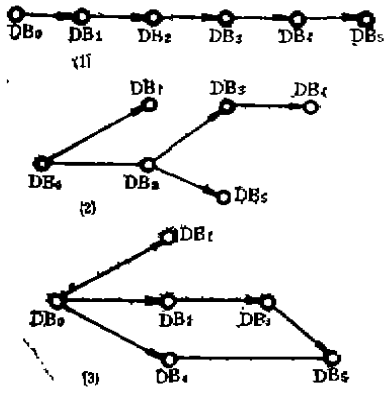

图1 (1)线性版本管理模型； (2) 树状模型；(3)有向无循环图模型。

线性模型是一种最简单的模型，它是以版本出现的先后次序进行排列的。对于一个设计过程的各个阶段所生成的设计版本，线性模型可以很好地加以反映。然而，在反映工程实体的不同设计方案所形成的设计版本时，时间上出现在后的版本就不一定能由紧接在它前面的那个版本得出来。在这种情况下，就必须用树状模型来加以描述。树状模型可以反映出设计中以某一中间版本为基础，选择多种设计方案从而形成多个设计结果的情况。但是树状模型也有它的局限性。对于由多个设计部件合成一完整实体，也就是由多个设计版本融合出一个新版本的情况，只能用有向无循环图模型才能较好地加以描述。有向无循环图模型是一较为完善的模型，它基本上可以反映生成多版本的各种情况。

二、向前版本管理法及形式化描述

无论采用那种模型，数据库的版本均反映了数据库一系列的一致性状态。而事务是保证数据库一致性的最小单位。一般来说，在工程实体的设计过程中，一个事务对应着一个设计步骤，因此我们可以将事务作为数据库版本更新的基本单位。

对于一个网状数据库，我们说，可以将其表示成DB=(O,S)，其中O是实体的集合，S是系值的集合。系值反映了各个实体之间的联系。从实现的角度看，我们可以将一个实体及相应的系值信息记录在同一记录之中。在这种情况下，可以进一步将数据库抽象为记录（record）的集合，即$DB=R=\{r_i | r_i \in R\}$；$i=1,2,\cdots,n$。由此可以看出，一个数据库的版本最终是由该数据库中所有记录的某一特定的版本组成的。因此，在生成数据库的新版本时，可以首先生成数据库中所有记录的新版本，这些所有记录的新版本就构成了数据库的新版本。但是，由于一个事务可能仅仅涉及到数据库中的部分记录，也就是说数据库中还有一部分记录并未更新，所以按上述方法生成数据库的新版本势必产生很大的冗余。

由于数据库的某一版本是由数据库中所有记录的某一版本构成的，为了讨论方便，假设$R^*=\{r_k\}$，$k=1,2,\cdots$，n是数据库中所有可能生成的记录的集合。用$r_{k,0}$（k指第k个记录，0表示零号版本）表示空版本，即表明该记录尚未创建。当某一记录被撤消时，它的新版本也变为空版本。因此可得$DB_i=R^*_i=\{r_{k,i}\}$，$k=1,2,\cdots,n$。其中k表示第k个记录，i表示该记录的第i个版本。

显然DB_0是由所有记录的空版本构成，亦即数据库中尚未创建任何记录。

现在我们来分析图2所示的以数据库版本DB_i为基础，通过事务T_j生成数据库版本DB_j的情况。图中$j \geq i+1$。

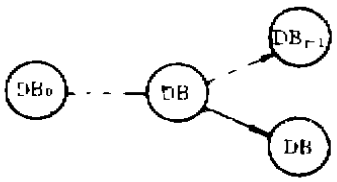

图2 基于版本DB_i生成版本DB_j

令$C_{i,i}=\{r_{k,i} \mid r_{k,i} \in R_i^* \wedge r_{k,j} \in R_j^* \wedge r_{k,i} \triangleq r_{k,j}\}$；$C_{j,i}=\{r_{k,j} \mid r_{k,j} \in R_j^* \wedge r_{k,i} \in R_i^* \wedge r_{k,i} \triangleq r_{k,j}\}$

其中$r_{k,i} \triangleq r_{k,j}$表示记录$r_k$的第$i$个版本与第$j$个版本完全相同。我们记$C_i=\{r_k \mid r_{k,i} \in C_{i,i}\}$；

$C_i'=\{r_k \mid r_{k,i} \in C_{j,i}\}$，显然$C_i=C_i'$。

C_j集合表示在第j个事务T_j运行过程中未被修改的记录的集合，我们定义其为事务T_j对数据库的保留集。$C_{i,i}$是构成DB_i版本，且在保留集C_j中的记录版本的集合。$C_{j,i}$是构成DB_j版本，且在保留集C_j中的记录版本的集合。

令$D_{i,j}=R_i^*-C_{i,j}$

$D_{j,i}=R_j^*-C_{j,i}$

并记$D_j=\{r_k \mid r_{k,i} \in D_{i,j}\}$；

$D_j'=\{r_k \mid r_{k,j} \in D_{j,i}\}$；

显然 $D_j=D_j'$。这里D_j表示在第j个事务T_j运行后，被修改的记录的集合，我们把它定义为事务T_j对数据库的修改集。$D_{i,j}$是构成DB_i版本，且在修改集D_j中的记录版本集合；$D_{j,i}$是构成DB_j版本且在修改集D_j中的记录版本的集合。

由上面的说明可得：

$DB=D_j \cup C_j$，这里$D_j \cap C_j=\phi$；

$DB_i=C_{i,j} \cup D_{i,j}$；

$DB_j=C_{j,i} \cup D_{j,i}$。

由于$C_{i,j}$与$C_{j,i}$中的记录版本是对应相同的，也就是说经事务T_j后，$C_{i,j}$中记录版本的内容未改变。所以，在存贮时不必再存贮新的$C_{j,i}$。此时有$C_{i,i}=C_{i,j}$。

然而，$D_{j,i}$中的记录版本是从$D_{i,j}$中对应的记录版本经事务T_j修改而得到的，所以$D_{i,j}$与$D_{j,i}$正反映了DB_i与DB_j之间的差异，因此应同时保存。综上所述可得：

$DB_i=C_{i,j} \cup D_{i,j}$；

$DB_j=C_{j,i} \cup D_{j,i}=C_{i,j} \cup D_{j,i}$；

$DB_j-DB_i=D_{j,i}$。

所以在事务T_j运行之后，数据库中只需增加存贮$D_{j,i}$中的记录版本即可。也就是说仅D_j中的记录生成新的版本。

这种版本管理法称为向前版本管理法。它只完整存贮原始的数据库版本数据，对后续的数据库版本只存贮它与前驱版本的差。

由于采用了向前版本管理方法，在事务T_j运行以后，当$r_k \in C_j$时，r_k中所有的版本号没有作任何改变。当$r_k \in D_j$时，r_k中将生成一个版本号为j的新的记录版本$r_{k,j}$。所以，对于一个特定的数据库版本DB_i，数据库中的每一个记录r_k不一定都有一个i号记录版本$r_{k,i}$，也就是说构成某一数据库版本DB_i的记录版本号不一定都是i。因此，在存取数据库的某一版本DB_i中的数据时，要判断记录中的哪一个版本是属于DB_i的。

由上面的讨论可知，对于数据库中的任何一个版本DB_i都存在着从DB_0到DB_i的一条或多条路径，这些路径的总和称为设计路径。对于图3中的DB_4、DB_6、DB_8，它们的设计路径分别为：$DB_0 \rightarrow DB_1 \rightarrow DB_4$；

$DB_0 \rightarrow DB_1 \rightarrow DB_2 \rightarrow DB_3 \rightarrow DB_5 \rightarrow DB_6$；

$DB_0 \rightarrow DB_1 \rightarrow DB_4 \rightarrow DB_5 \rightarrow DB_6$；

$DB_0 \rightarrow DB_7 \rightarrow DB_8$。

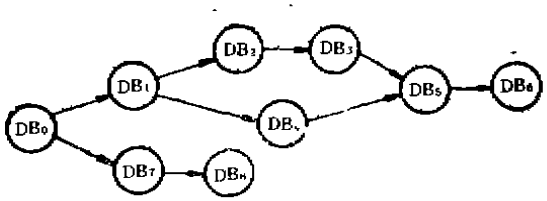

图3 各版本间的设计路径

设计路径上的每一数据库版本称为路径元素，用集合V_i表示DB_i的设计路径上的所有路径元素的版本号的集合。对应于图3分别有：$V_4=\{0, 1, 4\}$；$V_6=\{0, 1, 2, 3, 4, 5, 6\}$及$V_8=\{0, 7, 8\}$。

下面我们分两种情况来确定构成DB_i的记录版本。

(1) 当$r_k\in D_i$时，则由于r_k中有一个记录版本的版本号为i，所以该记录版本号即为构成DB_i的记录版本；

(2) 当$r_k\in C_i$时，r_k中不存在第i号记录版本。设r_k的所有记录版本的版本号构成集合U_k，令$P_k=U_k\cap V_i$，则P_k集合中最大的那个版本号所对应的r_k中的记录版本即为构成DB_i的版本。综上所述可得：

$$DB_i=\{r_{k,j}\mid 若r_k\in D_i，则j=i\vee 若r_k\in C_i，则j\in P_k，且对任意的l\in P_k有l\leqslant j\}$$
$$=\{r_{k,j}\mid j\in P_k\wedge j=\max(P_k)\}。$$

〔例〕设数据库$DB=\{r_1, r_2, r_3, r_4, r_5, r_6\}$且有五个事务$T_1$，$T_2$，$T_3$，$T_4$，$T_5$运行。事务$T_4$将两个已存在的版本$DB_2$和$DB_3$融合成一个新的版本$DB_4$，如图4所示。

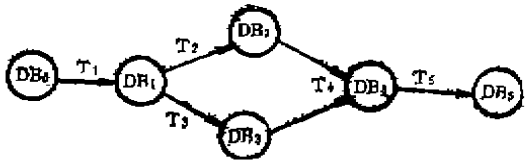

图4 版本DB_5的生成过程

经过事务$T_1\sim T_5$后，数据库DB的记录版本状态为：

DB_0	$r_{1,0}$	$r_{2,0}$	$r_{3,0}$	$r_{4,0}$	$r_{5,0}$	$r_{6,0}$
T_1	$r_{1,1}$	$r_{2,1}$	$r_{3,1}$			
T_2			$r_{3,2}$	$r_{4,2}$	$r_{5,2}$	
T_3	$r_{1,3}$		$r_{3,3}$			$r_{6,3}$
T_4						
T_5				$r_{4,5}$		

并且 $D_1=\{r_1, r_2, r_3\}$，$C_1=\{r_4, r_5, r_6\}$；
$D_2=\{r_3, r_4, r_5\}$，$C_2=\{r_1, r_2, r_6\}$；
$D_3=\{r_1, r_3, r_6\}$，$C_3=\{r_2, r_4, r_5\}$；
$D_4=\phi$，$C_4=\{r_1, r_2, r_3, r_4, r_5, r_6\}$；
$D_5=\{r_4\}$，$C_5=\{r_1, r_2, r_3, r_5, r_6\}$。

如上所示，事务$T_1\sim T_5$运行后，各记录的版本分别为：r_1:$\{r_{1,0}, r_{1,1}, r_{1,3}\}$，$r_2$:$\{r_{2,0}, r_{2,1}\}$，$r_3$:$\{r_{3,0}, r_{3,1}, r_{3,2}, r_{3,3}\}$，$r_4$:$\{r_{4,0}, r_{4,2}, r_{4,5}\}$，$r_5$:$\{r_{5,0}, r_{5,2}\}$，$r_6$:$\{r_{6,0}, r_{6,3}\}$。数据库的各个版本分别为：

$DB_1=\{r_{1,1}, r_{2,1}, r_{3,1}, r_{4,0}, r_{5,0}, r_{6,0}\}$，
$DB_2=\{r_{1,1}, r_{2,1}, r_{3,2}, r_{4,2}, r_{5,2}, r_{6,0}\}$，
$DB_3=\{r_{1,3}, r_{2,1}, r_{3,3}, r_{4,0}, r_{5,0}, r_{6,3}\}$，
$DB_4=\{r_{1,3}, r_{2,1}, r_{3,3}, r_{4,2}, r_{5,2}, r_{6,3}\}$，
$DB_5=\{r_{1,3}, r_{2,1}, r_{3,3}, r_{4,5}, r_{5,2}, r_{6,3}\}$，

上面T_4将两个已存在的版本融合成一个新的版本，不生成任何新的数据，所以$D_4=\phi$。

上面我们讨论了用向前版本管理的方法进行多版本的存贮和管理。这种方法虽然每次仅存贮前后两个版本的差，但是随着版本数目的增多仍将占用很大的存贮空间。因此，在具体实现时，必须考虑如何利用有限的存贮空间，尽可能多地保存重要的设计版本。下面将就此问题进行讨论。

三、有限记录版本法

通过上面的讨论可以看出，为了能保留数据库的所有版本，势必要求每一个记录都可以有任意多个记录版本。然而这在数据库的实现中是不可能的。因为当每一记录的版本数过多时，将占用很大的存贮空间，同时又影响了数据的集聚性，使数据库的性能下降。所以在实现时，对每个记录来说，能保存的版本数是有限的。下面讨论每个记录仅能保留两个不同记录版本的情况。

假设现有一事务T_n运行后，生成了数据库版本DB_n。由于每一个记录仅能保留两个内容不同的记录版本，所以在DB_n生成后，DB_{n-1}可以同时保留，但DB_{n-2}就有可能已被破坏。下面将就两种情况进行讨论。

〔**定理4.1**〕设D_n与D_{n-1}分别是事务T_n及T_{n-1}对数据库的修改集。(1) 当$D_n\cap D_{n-1}=\phi$时，数据库中除了保留有DB_n，DB_{n-1}外，还至少保留有DB_{n-2}版本；(2) 当$D_n\cap D_{n-1}\neq\phi$时，数据库中仅保留有DB_n，DB_{n-1}两个版本。

证明：(1) 对于任意的$r_k\in D_n$，显然$r_k\in$

D_{n-1}。在事务 T_n 运行之前r_k所保留的两个内容不同的记录版本为$r_{k,n-m}$和 $r_{k,n-1}=r_{k,n-2}=\cdots=r_{k,n-m+1}$ $(m\geqslant 3)$。当 T_n运行时，为生成r_k的新版本，将使用$r_{k,n-m}$的空间，因而使原$r_{k,n-m}$丢失。事务T_n运行结束后，r_k所保留的两个内容不同的记录版本分别为$r_{k,n-1}=r_{k,n-2}=\cdots=r_{k,n-m+1}$ $(m\geqslant 3)$ 和$r_{k,n}$，所以构成DB_{n-2}的记录版本没有丢失。

对于任意的$r_k\in D_n$，事务T_n对r_k没有作任何修改，所以r_k中构成DB_{n-2}的记录版本没有丢失。所以，当 $D_n\cap D_{n-1}=\phi$ 时，事务T_n运行之后，数据库的DB_{n-2}版本仍被保留。

(2) 当$D_n\cap D_{n-1}\neq\phi$时，必然存在一记录r_k，$r_k\in D_n\cap D_{n-1}$，则当事务T_{n-1}运行之后，r_k中保留的两个内容不同的记录版本分别为$r_{k,n-1}$，$r_{k,n-2}=\cdots=r_{k,n-m}(m\geqslant 2)$。当事务$T_n$运行时，要生成$r_k$的新的记录版本，所以必须使用$r_{k,n-2}=\cdots=r_{k,n-m}$ $(m\geqslant 2)$的存贮空间。当事务T_n运行之后，记录r_k的两个内容不同的版本号分别为$r_{k,n}$和$r_{k,n-1}$。所以仅能构成DB_n及DB_{n-1}版本，也就是说DB_{n-2}版本已被丢失。□

下面我们给出两个一般性的结论：

推论1 当每一个记录仅能保留两个内容不同的版本时，若存在一个k，$k\in\{1,2,\cdots,n-1\}$使得

$$\bigcup(D_i\cap D_j)=\phi$$
$$i<j$$
$$i=n-k,n-1$$
$$j=n-k+1,n$$

成立，则数据库中可同时保留 DB_n，DB_{n-1}，DB_{n-2}，…，DB_{n-k-1}共$k+2$个数据库版本。

推论2 当每一个记录可以同时保留m个内容不同的记录版本时，若存在一个k，$k\in\{1,2,\cdots,n-m+1\}$使得

$$\bigcup(D_{i1}\cap D_{i2}\cap\cdots\cap D_{im})=\phi$$
$$i_1<i_2<\cdots<i_m$$
$$i_1=n-m-k+2,\ n-m+1$$
$$i_2=n-m-k+3,\ n-m+2$$
$$\vdots$$
$$i_m=n-k+1,\ n$$

成立，则数据库中可同时保留DB_n，DB_{n-1}，…，$DB_{n-m+1-k}$共$k+m$个数据库版本。

通过上面的讨论可以看出，在每一个记录仅能保留有限个记录版本时，只要能合理地安排设计步骤，是能够保留尽可能多的数据库版本的。

由于有限记录版本法能保留从时间上看在DB_n版本之前的若干连续的数据库版本，所以对于线性模型来说是可以使用的。但是，对于树状模型和有向无循环图模型，这种方法就不太适用了，因为它不能有效地保证分支结点版本，如图3中的DB_1版本，不被破坏。

四、关键版本法

对于一数据库的版本DB_n，它的设计路径是由许多数据库版本构成的。而对于这众多的数据库版本，它们的重要性是有区别的。如前面图5所示的DB_3的设计路径是$DB_0\rightarrow DB_1\rightarrow DB_2\rightarrow DB_3$，显然，这一路径中的$DB_1$比$DB_2$重要，因为基于$DB_1$版本可采用两条路径继续进行设计。所以在生成$DB_4$版本之前是不允许丢失$DB_1$版本的。而$DB_2$仅仅是设计过程中的一个中间结果版本，该版本在生成$DB_3$后，即使被破坏了，也不会对整个设计过程产生很大的影响。所以我们根据各数据库版本在整个设计过程中的重要性，将其分为关键版本和非关键版本。在生成数据库的新版本时，对于关键版本是不允许被破坏的，而对于非关键版本则是可以被破坏的。

任何一个数据库的版本是关键版本还是非关键版本，这并不是一成不变的。也就是说，某一版本在某一时刻为关键版本，而在以后的另一时刻可能就不再是关键版本了。如前面图3中，DB_1在生成了DB_2和DB_4之前，它是关键版本，当生成了DB_2和DB_4以后，它就可以不再是关键版本了。根据需要，用户可以将任何一个有效的数据库版本设置成为

关键版本，当然也可以将一关键版本设置为非关键版本。

对于关键版本法又可以分为关键路径法和非关键路径法两种。

(i)关键路径法 关键路径法仅保留设计路径中的关键版本。在生成新的记录版本时，当记录中的某一记录版本不属于某一数据库的关键版本时，记录的这一版本空间就可以用来生成新的记录版本，否则就申请新的版本存贮空间。这种方法的优点是关键版本的数目不受限制，但当关键版本设置过多时，空间的开销仍然很大。

(ii)非关键路径法 在这种方法中，每个记录的最大记录版本数是固定的。也就是说，每个记录的最大允许空间是一定的，在生成记录的新版本时，首先申请存贮空间，当记录的存贮空间已满时，则在该记录所保留的所有版本中，选择一个不属于任何一个数据库关键版本的记录版本，用该记录版本的空间来生成新的记录版本。使用这种方法时，数据库中保留的设计路径中的数据库版本就不一定都是关键版本了。这种方法的优点是可以有效地控制存贮空间的膨胀，但缺点是关键版本的数目受到了限制。

五、设计版本的再组织

保留数据库的多个版本的一个重要目的是为了记录设计过程中的历史数据，以使数据库能回退到以前曾出现过的状态。既然数据库中存贮了大量的历史数据，那么能否充分利用这些历史数据来生成数据库的新版本，而并非是仅仅利用这些历史数据来实现数据库状态的回退呢？利用已有的多个版本来融合出一个新的版本的情况在前面已作了讨论，下面将对这一问题作进一步的讨论。

由前面的讨论可以知道，一个事务T_i的运行结果是为一个修改集D_i中的所有记录生成一个新的记录版本。由于一个设计步骤可能涉及设计对象的多个方面，比如设计一个茶杯，一次事务的运行可能要修改杯盖及杯体的参数，当这一设计步骤完成后，有可能设计者希望能用原来的杯盖和修改后的杯体作为新的设计结果。为了实现这一点，一种方法是使数据库状态回退一步，即重新依据事务T_i运行之前的数据库版本，对杯体进行修改。这样做当然是可以的，但是由于事务T_i运行前后的两个数据库版本都保留了，所以可以通过重新组织这两个版本的数据来实现设计者的要求。也就是将D_i中部分记录的$i-1$号版本作为新的$i+1$号版本，从而构成新的数据库版本DB_{i+1}。我们称这一过程为对相应记录的版本切换。当然，这一切换必须不破坏数据库状态的一致性。

由于一个事务是由若干事务步，也可以说是由一系列DML语句组成的，一条DML语句有时要涉及多个记录，这些记录的记录

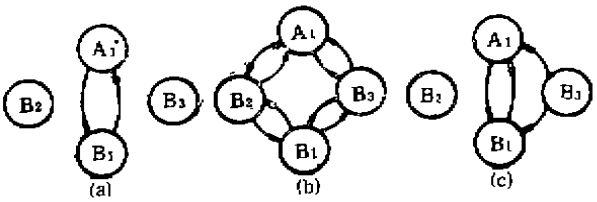

图5 事务T_i运行前后数据库状态

版本必须同时切换，否则将破坏数据库的一致性。

下面我们来考查图5(a)的情形。当事务T_i把B_2、B_3分别插入到B_1的前面及后面时，将产生图5(b)的结果。显然事务T_i是由两条DML的插入语句组成的。第一条插入语句涉及到A_1、B_1、B_2三个记录；第二条插入语句涉及到A_1、B_1、B_3三个记录。事务T_i的修改集$D_i=\{A_1, B_1, B_2, B_3\}$。当事务$T_i$运行以后，如果我们希望不将$B_2$插入到$B_1$之前，直观上看仅需将$D_i$中的$A_1$、$B_1$、及$B_2$的$i-1$号记录版本作为$i+1$号记录版本，从而构成新的数据库版本$DB_{i+1}$即可。也就是对$A_1$、$B_1$及$B_2$进行版本切换。然而，这样做的结果将使数据库的最新版本处于图5(c)所示的状态，这显然是一种不一致的状态。因此有必

要研究对哪些记录的记录版本同时进行切换才能保证数据库的一致性。

定义1 一条DML语句所修改的记录的集合称为一个记录版本的更新基本集，简称基本集，用集合A表示。

定义2 在D_i中为了切换某一记录r_k的版本而必须与其同时进行版本切换才能保持数据库一致的最小集合称为记录版本更新相关集，简称相关集，用集合L(A)表示。

可以按下列算法构造相关集L(A)：

算法 设A_1，A_2，…，A_n是一个事务中由n个DML语句产生的基本集。令：

(1) $L_1(A_i)=A_i$

(2) $L_{k+1}(A_i)=L_k(A_i)\cup\{r_j|r_j\in A_j \wedge \exists r_p(r_p\in A_j \wedge r_p\in L_k(A_i))\}$。

对于由此所构成的集合序列显然有$L_k(A_i)\subseteq L_{k+1}(A_i)$，$(k\geqslant 1)$，而且由于k是一有限正整数，故必存在一个$m$使得

$L_m(A_i)=L_{m+1}(A_i)=\cdots$

令$L(A_i)=L_m(A_i)$ 则$L(A_i)$即为一相关集。

由上面的算法可得，当存在一个$r_k\in A_i\cap A_j$时，则$L(A_i)=L(A_j)$。

定理6.1 设L_{i_1}，L_{i_2}，…，L_{i_p}是修改集D_i中的所有不相等的相关集，则有：

$L_{i_1}\cap L_{i_2}\cap\cdots\cap L_{i_p}=\phi$；

$L_{i_1}\cup L_{i_2}\cup\cdots\cup L_{i_p}=D_i$。

且当对L_{i_j}中的记录同时进行版本切换时不会破坏数据库的一致性。

对于上面讨论的切换不仅仅限于将D_i中的部分记录的$i-1$号记录版本作为$i+1$号记录版本的情况，还可以将任意的D_j中的记录的任何一个有效的记录版本作为一个新的记录版本以构成新的数据库版本。只要在切换时，对相应的相关集中的记录同时进行切换就不会破坏数据库的一致性。

本文对网状数据库中设计版本的存贮和管理，即在数据库中同时保留多个数据库版本的问题进行了讨论，提出了两种具体的实现方法，同时也对如何有效地使用数据库中的历史数据以生成数据库的新版本进行了研究。本文仅仅涉及了工程数据库中版本管理这一方面的内容，工程数据库中其它方面的内容，如事务管理、开发控制、复杂对象的描述等问题将是下一步的研究工作。（本文由聂培尧整理）

主要参考文献

〔1〕 Thomas Neumann, "On representing the design information in a common database", Engineering Design Applications, San Jose, 1983, pp81—88

〔2〕 Jiang Zejun, Nie Peiyao and Xu Qiuyuan, "On Representing the Design Information in Engineering Database", Computer-Aided Design & Graphics, Pergamon Press, 1989, pp401—406

〔3〕 方家骐等，"工程数据库中版本管理问题"，计算机辅助设计与图形学报，№.1，1989

〔4〕 方家骐，"工程数据库的关键和进展"，计算机工程与应用，№.5，1988