データベース・システム　５３－３
（１９８６．５．１９）

マルチバージョン・データベースにおける同時実行制御機構

羽生田　博美，　川上　英，　疋田　定幸
沖電気工業株式会社

更新トランザクションによってアクセスされるデータ項目のバージョン数を２つに制限したマルチバージョン・データベース・モデルと，このモデル上での同時実行制御機構について提案した。

この同時実行制御機構は，通常の単一バージョン・モデルでの２相ロック・プロトコルを拡張することによって得られる。したがって，本稿で提案した同時実行制御機構を持つデータベース・システムは，単一バージョン・モデル上の２相ロック・プロトコルに基づいた既存のデータベース・システムを拡張することによって容易に実現することが可能である。

The Concurrency Control Mechanism in a Multiversion Database (in Japanese)

Hiromi HANIUDA, Suguru KAWAKAMI and Sadayuki HIKITA
OKI Electric Industry Co.Ltd., 4-10-12 Shibaura, Minatoku, Tokyo 108, Japan

A multiversion database model where the number of versions of a data item accessed by update transactions is restricted to only two, and the concurrency control mechanism in this database model are proposed.

This concurrency control mechanism is obtained by extending the ordinary two phase locking protocol in a single version model. Therefore, multiversion database systems with this concurrency control mechanism can be easily implemted by modifying existing database systems based on the two phase locking prtocol in a single version model.

1．はじめに

データベースの同時実行性を向上させる方法の一つとして，マルチバージョンを使用した方法に関して多くの理論的研究が行われ[BERN83，PAPA84]，幾つかのアルゴリズムが提案されてきた[BAYE80a，BAYE80b，BUCK83，CHAN82，DUB082，STER81]。しかし，一般的なマルチバージョン・データベース・モデルに対する多項式時間オンライン・スケジューラの実現は不可能である[PAPA84]。したがって，現実的にはマルチバージョンを用いて同時実行性を向上させるにはマルチバージョン・データベース・モデルになんらかの制限を加えなければならない。この制限には，トランザクションのモデルを制限して，オンライン・スケジューラを使う方法[BUCK83，STER81]，あるいはロックを用いた方法[BAYE80a，CHAN82]がある。データベース・モデルに対してスケジューラによって直列可能性を保証するためには，任意のトランザクションがデータベースに投入された時点で，そのトランザクションが将来行う読み込み操作（read）と書き込み操作（write）の集合をスケジューラが知ることを可能にしなければならない[BUCK83]等の制限が必要である。このような制限は，特にインタラクティブにデータベースにアクセスする要求があるような場合には強い制限であり，現実的ではないと考えられる。また，ロックを用いた方法もマルチバージョンとトランザクションの管理が複雑になる[CHAN82]等の問題があり現実的ではない。そこで我々は，２相ロック機構を持つ単一バージョン・データベースを拡張した現実的なマルチバージョン・データベースの同時実行制御の方法を検討したので提案する。

本論文では，第２章ではマルチバージョン・データベース・モデルとトランザクションの行動について，第３章では同時実行制御方式について，第４章では読み込み専用トランザクションについて述べ，第５章でまとめを行う。

2．マルチバージョン・データベース・モデル

我々の提案するマルチバージョン・データベース・モデルは同時実行制御に影響を与えるバージョンの数を２つに制限したモデルである。本章では，このモデルとその上でのトランザクションについて述べる。

(1) マルチバージョン

データベースは，データ項目から構成され，各データ項目は値の列S<$d_1, d_2, \ldots, d_k$>から構成されている。値の列を構成する各値はd_1，d_2，...の順にそれをwriteしたトランザクションがコミットした時刻順に並べられており，各値をwriteしたトランザクションのタイムスタンプを持っている。

これらの値は，c-バージョン（current version），b-バージョン（before version），またはp-バージョン（past version）と呼ぶ，３種類のクラスに分類される。これを図１を用いて説明する。

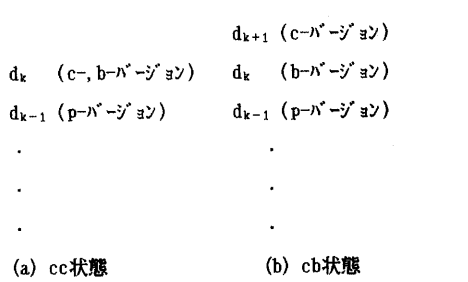


図１　ﾃﾞｰﾀ項目の状態とﾊﾞｰｼﾞｮﾝのｸﾗｽ

c-バージョンは，値の列S中で最新の値である。b-バージョンはデータ項目が更新中でなければc-バージョンと一致している。更新中の場合は，c-バージョンの次に位置する値，すなわちc-バージョンの次に新しい値である。更新のための参照はこのb-バージョン，およびc-バージョンの一方のみに対して行われる。p-バージョンはc-バージョン，b-バージョン以外の値である。

データ項目dのc-バージョンとb-バージョンが一致している時，すなわちd_kがc-バージョンかつb-バージョンである時（図１(a)），データ項目dはcc状態にあると言う。あるトランザクションTが，cc状態にあるデータ項目を更新すると，Tがwriteした値がバージョン列に追加されてc-バージョンd_{k+1}となり，更新前の値であるd_kは，b-バージョンとなる。この時，データ項目dはcb状態に

あるという（図1(b)）。また，cc状態およびcb状態のデータ項目に対するreadは，b-バージョンに対して行われる。

データ項目を更新することにより，新しい値が次々に列Sに追加される。しかし，我々のモデルでは，ある時刻にある更新トランザクションがアクセスできる値はb-バージョンまたはコミットされたc-バージョンの一方だけであり，列Sに新しい値が追加されても，更新トランザクションがアクセスできるバージョンの数を2つに制限しておく必要がある。したがって，新しい値を列Sに追加するには，b-バージョンに対するアクセスを行わせなくする時点が必要である。すなわち，c-バージョンの値をwriteしたトランザクションが終了（システム中から消滅）した後，列Sの最後に位置する値をc-バージョンかつb-バージョンとすることによって，データ項目の状態をcc状態にしなければならない。このバージョンの移り変わりをバージョンアップと呼ぶ。図1において(b)の状態のデータ項目がバージョンアップすると，d_{k+1}がc-,b-バージョンになり，$d_i(i \leq k)$がp-バージョンになる。また，バージョンアップ中のデータ項目をccb状態と呼び，ccb状態のデータ項目に対するreadはc-バージョンに対して行われる。

(2) トランザクション

我々のモデルにおけるトランザクションは，一般のモデルにおけるトランザクションと同様に同時実行制御の単位である。トランザクションTは次の行動をとることができる。

投入： システムに投入されること。システムに投入されたトランザクションは同時実行制御によって行動が制約される。

コミット：Tは以後データ項目にアクセスしないことを宣言すること。コミットしたトランザクションは，ロールバックの対象とはならない。

アボート：Tは以後データ項目にアクセスしないことを宣言すること。Tがwriteしたすべての値はバージョン列から除去される。

終了： すべての処理を終了すること。コミットまたはアボートしたトランザクションのみに許される。終了したトランザクションは，システム中から消滅し，同時実行制御の対象ではなくなる。

ロック： データ項目に対するロック要求。個々のデータ項目毎に行う。

read： データ項目のread

write： データ項目の更新。すなわち，新しい値を書込みデータ項目のバージョン列に追加すること。

システムに投入され，コミットしていないトランザクションを，アクティブ・トランザクションと呼び，コミットしたが，まだ終了していないトランザクションをコミットされたトランザクションと呼ぶ。また，データ項目の更新を行わないトランザクションを読み込み専用トランザクション，更新を行うトランザクションを更新トランザクションと呼ぶ。同時実行制御の対象となるのは，更新トランザクションである。

システムに投入されるトランザクションに，次の規則に従ってタイムスタンプを与えることによって，トランザクション間に時間的な前後関係を与える。

(a) 更新トランザクションにはシステムに投入された時に，システムによってこのトランザクションが投入された時刻がタイムスタンプとして割り当てられる。

(b) 任意の更新トランザクション T_i，T_jに対して $TS(T_i) \neq TS(T_j)$である。ただし，TS(T)はトランザクション Tのタイムスタンプである。

(3) 先行関係

システム中に存在するトランザクション間に先行関係を定義する。データ項目dが列$S\langle d_1, d_2, \ldots, d_k\rangle$から構成されると仮定したとき，dをデータ項目として$T_i \rightarrow_d T_j$，および$T_i \Rightarrow_d T_j$で表される先行関係は次のように定義される。

"$T_i \rightarrow_d T_j$" iff "データ項目dに対し，T_iがd_kをwriteしT_iがd_{k-1}をreadした"

"$T_i \Rightarrow_d T_j$" iff "データ項目dに対し，T_iがwriteしたd_kをT_jがreadした"

システム中に存在するすべてのトランザクションに関して，各トランザクションを節，$\rightarrow_d$，および$\Rightarrow_d$をラベルdを持つエッジとして得られるグラフを先行関係グラフ（以下BF）と呼び，後に述べるようにデータベースの一貫性を保つためには，このBFが非巡回でなければならない。

3．同時実行制御

単一バージョン・モデルにおいては通常の２相ロック・プロトコルによって直列可能性が保証されるが，マルチバージョン・モデルにおいてはこれだけでは不十分であり，トランザクション間の先行関係を考慮して同時実行制御を行わなければならない。我々の同時実行制御機構では，２相ロック・プロトコルと，BFの非巡回性を保障することによってデータベースの一貫性を維持する。すなわち，まずロックによって直列可能性に影響を与えるバージョン数と，制限されたバージョンの各々に対する処理を制限する。これによって，BFは，ロックによって制限を受けないb-バージョンやコミットされたc-バージョンのreadに関係するトランザクションと先行関係だけの縮退されたグラフとなる。次に，この縮退されたBFを維持して，非巡回性をチェックする。

本章では，このロック・プロトコルとBFによる一貫性の維持について述べる。

3．1　ロック・プロトコル

我々のマルチバージョン・モデルでは，直列可能性に影響するバージョンに対する制限はロックによって行われる。したがって，ロック・プロトコルはバージョンアップに対して何をするかを，また各バージョンに対してどのような制限を加えるかを考慮して決定しなければならない。

あるロック・モードは，あるトランザクションTがそのロックを，あるデータ項目dに設定した時，TとT以外のトランザクションT_oに対してどのようなアクセスがdに対して許されるかによって特徴付けられる。また，ロック・モデルにおいては，例えば，更新を許すロックをあるデータ項目dに設定することは，dをreadしてからdを更新するというスケジュールに対応している。したがって，我々のモデルではロックを設定することにより，データ項目の状態が変化すると考えることができ，ロック・モードはロックが許されるデータ項目の状態，ロック後のデータ項目の状態とによっても，特徴付けることができる。

あるトランザクションTがあるデータ項目dをロックする場合に，そのロック・モードとデータ項目dの状態，TおよびT_oに許される処理との関係を表１に示す。

表１　ロック・モードの特徴

		W ロック	R ロック
データの状態	ロック前 ロック後	CC, CB CB	CC, CB, CCB －
許される処理	T T_o	read, write read	read read

－：状態の移行なし

表１におけるreadは，Wモードに関しては，b-バージョンに対するreadである。また，Rに関しては，対象のデータ項目がccb状態ならば（c-バージョンのコミット後のバージョンアップ中であれば），c-バージョンに対するreadであり，さもなければ（対象データ項目がcc/cb状態ならば）b-バージョンのreadである。したがって，WモードはRモードとのみ両立可能であり，Rモードはすべてのモードと両立可能である。更新が行われている，すなわちWロックされているデータ項目のb-バージョンの値をreadさせることにより，WとRモードは両立可能に

なっている。

また，同時実行制御に影響を与えるバージョン数を２つに制限するために，ロックの要求および解除は次の規則に従って行われる。

(a) あるトランザクションがW ロックを設定していたデータ項目の状態は，このトランザクションがコミットする時にccb状態に変えられる。

(b) あるトランザクションが設定したロックは，このトランザクションが終了する時に解除される。

(a)により，トランザクションTがコミットした後の，Tがwriteしたデータ項目のb-バージョンに対する新たなアクセスが禁止される。また，(b)により(a)で禁止されたb-バージョンのreadが再び許されるようになる。すなわち，(a)および(b)を通じてバージョンアップが完了する。

以上のように，直列実行可能性に影響するバージョンをc-バージョンとb-バージョンの２つに制限し，かつ関係するトランザクション数も制限することができる。

３．２　トランザクションの衝突

我々のモデルは，read/writeするデータ項目にはロックを設定することを要求されるので，[PAPA84]における制約モデルの部分集合となっている。したがって，直列性を維持するためには，BFが非巡回である必要がある。ここで，Rロックはb-バージョン，またはコミットされたトランザクションがwriteしたc-バージョンのreadを許しているため，これらのreadによってBF中に閉路が発生する場合があり，上述したロック・プロトコルだけではスケジュールが直列可能でなくなる可能性がある。したがって，BFの非巡回性のチェックをロック・プロトコルとは別に行う必要がある。

BF中の閉路は適当なトランザクションのロールバックによって解消するが，この時次の同時実行制御規則を満たしていなければならない。

(a) BF中に閉路が発生した場合は，閉路を構成する１つ以上のトランザクションをロールバックして閉路を破壊する。

(b) アクティブ・トランザクションのみがロールバック可能である。

(c) (a)によりロールバックを行う場合は，閉路中の最も古いトランザクションをロールバックするようなことはない。

(d) トランザクションTが生成したc-バージョンは，T がコミットされるまで他のトランザクションによってアクセスされることはない。

(a)(d)は単一バージョン・モデルにも共通した一般的規則である。(b)により，コミットしたトランザクションはロールバックの対象外となり，単一バージョン・モデルと同様に利用者にとってコミットが実質的なトランザクションの終了と考えることが可能となる。また，(c)により，特定のトランザクションが常にロールバックされてしまい終了できないという状況を回避することができる。

次に，上記規則を満足するために必要な条件について考える。BF中で，エッジ⇒に対応するreadは，単一バージョンの２相ロック・プロトコルにおけるreadに対応している。したがって，我々のモデルにおいても，２相ロック・プロトコルによって，閉路中のすべてのエッジが⇒（以下エッジのラベルは省略する）であるような閉路は発生しない，したがって，BF中に閉路が発生したならば，あるアクティブ・トランザクションTと，Tとは異なるトランザクションT_i，T_jが存在して，この閉路をCとすれば，（ $T_i \Rightarrow T \rightarrow T_j \in C$ ）または（ $T_i \rightarrow T \rightarrow T_j \in C$ ）である。この時，トランザクションのロールバックに関して最悪の状況は，ロールバックの犠牲となるトランザクションの選択が唯一の場合，すなわちこの閉路中でアクティブ・トランザクションがTだけである場合である。したがって，トランザクション・ロールバックに関する同時実行制御規則を満足するためには，$T_i \rightarrow T_j$なるトランザクションT_i，T_jに対してなんらかの制限を与える必要がある。

$T_i \rightarrow T_j$なる関係は，あるデータ項目dに対して，T_iがdのb-バージョンの値をreadし，T_jがdのc-バージョンの値をwriteした時に発生し，これは我々のモデルでは，R

ロックとWロックが競合する場合である。したがって，我々はロック・モード W，Rの両立性に関して，次の条件を導入する。

(1) T_iによりRロックされているデータ項目dに対してT_jがWロックを要求した場合，$TS(T_i)>TS(T_j)$であれば，このWロックはこのRロックと両立可能であり，さもなければ両立不可能である。
(2) T_jによりWロックされているデータ項目dに対してT_iがRロックを要求した場合，$TS(T_i)>TS(T_j)$であれば，このRロックはこのWロックと両立可能であり，さもなければ両立不可能である。

これらの条件を導入することによって，BF中の閉路を破壊する時には，この閉路を構成するトランザクション中で最も古いトランザクション以外にアクティブ・トランザクションが存在することになる。条件 $TS(T_i)>TS(T_j)$は，T_iがT_jよりも若いことを意味する。このとき，$T_i \rightarrow T_j$は，若いトランザクションが古いトランザクションが更新しているデータ項目の古い値をreadすることを意味する。

3．3　トランザクションの終了

我々のモデルでは，b-バージョンのreadを許すことの代償として，単一バージョンの場合とは異なって，トランザクションはコミットと同時に終了することはできない。ここで，トランザクションTの終了とは，Tに対する後処理等を含む，すべての処理が終了することを意味する。本章ではトランザクションの終了，すなわちトランザクションがいつ終了することができるかについて考える。

読み込み専用トランザクションは同時実行制御に影響を与えないから，無条件に終了することが可能である。したがって，問題はコミットされた更新トランザクション（T_uとする）はどのような条件の下で終了することができるかである。あるトランザクションTに対して，トランザクションの集合BC(T)を次のように，帰納的に定義する。

$BC(T)=\{\ T_i \mid T_i \rightarrow T \text{ or } T_i \Rightarrow T\ \}$，

$BC(T)=\{\ T_i \mid \exists T_j \in BC(T) \text{ and } (\ T_i \rightarrow T_j \text{ or } T_i \Rightarrow T_j\ )\}$

T_uの終了は，T_uがread/writeしたデータ項目に対して先行閉路のチェックを中止することを意味する。したがって，T_uがread/writeしたデータ項目に関して先行閉路が発生しない状態になった時，T_uは終了できる。もし，$BC(T_u)$中にアクティブ・トランザクション（T_iとする）が存在すれば，$T_u \Rightarrow T_i$なる関係が発生する可能性がある。また，任意のトランザクションTに対して，Tがコミットした後にTに関して発生する先行関係は，$T \Rightarrow T_k$なる関係だけである。したがって，次の条件が成立するときT_uは終了することができる。

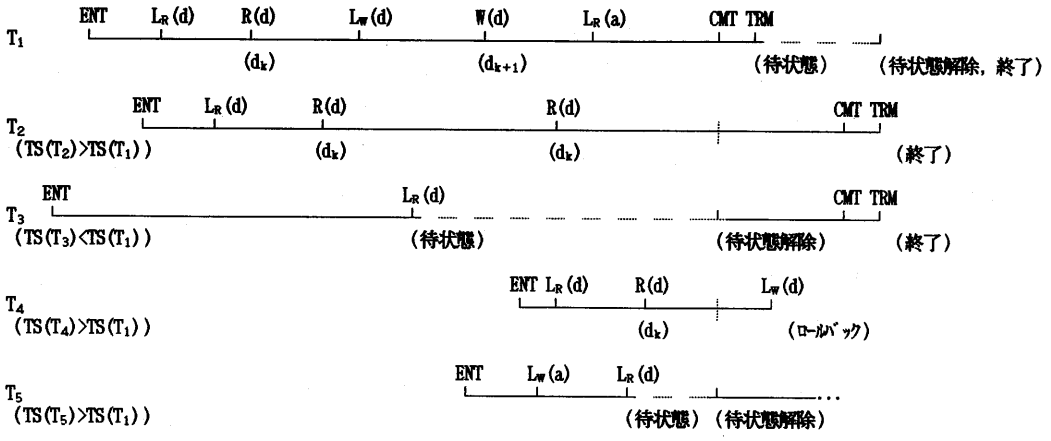
$\forall T \in BC(T_u)$に対してTはコミットされたトランザクションである。

ここで問題は，新たにシステムに投入されるトランザクションが，$BC(T_u)$中のすべてのトランザクションがコミットする前に，この$BC(T_u)$のメンバーとなる場合である。この時，$BC(T_u)$には常にアクティブ・トランザクションが存在することになり，T_uは長時間にわたって終了できないという問題が発生する。この問題は，$BC(T_u)$のメンバーとなり得るトランザクション数を制限することによって解決される。

3．4　同時実行トランザクション例

本章では，前章までに述べた同時実行制御方式を簡単な例を用いて説明する。

データ項目d，aにアクセスする５つのトランザクションを想定する。トランザクションとしては，更新トランザクションT_1，T_2，T_3，T_4が存在し，次のような行動をとる。すなわち，T_1はデータ項目dをreadし，新しい値をwriteして，その後データ項目aをreadして終了する。T_2およびT_3はデータ項目dをreadし，その後終了する。



ENT　：システムへの投入
$L_R(d)$：データ項目dに対するRロック
$L_w(d)$：dに対するWロック
R(d)　：dのread
W(d)　：dのwrite

CMT　：コミット
TRM　：終了
(d_i)　：アクセスされるデータ項目のバージョン
$TS(T_i)$：トランザクションT_iのタイムスタンプ

図2　同時トランザクションの例

T_4はデータ項目dをreadし，その後このdをwriteする。T_5はデータ項目aをwriteし，その後データ項目dをreadする。各トランザクションはT_3，T_1，T_2，T_5，T_4の順にシステムに投入される。これらのトランザクションが行う各要求の順序と，競合状態を図2に示す。

図2では，トランザクションT_1は無条件に終了することはできず，T_1が更新したデータ項目dのb-バージョンをreadした，トランザクションT_2，T_3，T_4が終了またはロールバックされるまで待状態となる。また，トランザクションT_3は，$TS(T_3)<TS(T_1)$であるため，データ項目dのreadが待状態となる。T_4はデータ項目dをwriteすることにより$T_1 \Rightarrow T_4 \rightarrow T_1$となるために，$T_4$のWロック要求$L_w(d)$は認められずロールバックさせられる。また，$T_5$はデータ項目dをreadするにより，$T_1 \rightarrow T_4 \rightarrow T_1$となり，ロック要求$L_w(d)$が待状態となり，$T_1$がコミットすることにより，$T_5$の待状態が解除される。

4．読み込み専用トランザクション

マルチバージョン･データベースの利点として，更新トランザクションの同時実行性の向上の他に，読み込み専用トランザクションに多くのバリエーションを与えることの可能性がある。本章では，読み込み専用トランザクションの実現について考える。

我々のシステムでは，読み込み専用トランザクションはwriteを行わず，かつ同時実行制御に影響を与えないトランザクションとして定義している。したがって，読み込み専用トランザクションの目的は過去の任意の時刻を与えられた時に，その時刻におけるデータベースを可能な限り忠実に利用者に提供することである。読み込み専用トラザクションを，前章までに述べた規則に従わせると，過去のデータベースを読むことができず，単一バージョンにおける読み込み専用トランザクションと同様の機能しか得られない。そこで，このような読み込み専用トランザクションT_rを我々のシステムでは，次のように提供する。

(1) システムが生成された後，システム中にアクティブ・トランザクションが存在しなかった区間を順に，(t_0, t_1)，(t_2, t_3)，．．．，(t_{2i}, t_{2i+1})とし，システムはこれらの区間を記憶する。

(2) ある読み込み専用トランザクションが時刻tにシステムに投入される時，$t_{2i+1} \leqq t$を満足する最大のt_{2i+1}をT_rのタイムスタンプ$TS(T_r)$に割り当てる。

(3) T_rがデータ項目dをreadする時は，T_rはd_{Km}をreadする。ただし，k_mは$\{k \mid TS(d_k) \leqq TS(T_r)\}$のうち最大のkとする。

このようにして定義された読み込み専用トランザクションは同時実行制御に影響を与えることなく一貫したデータベースを得ることができる。

ここで問題は，例えば座席予約システムにおいて，現在の空席数を検索するような，可能な限り最新のデータをreadするという要求に，上記読み込み専用トランザクションは対応できないことである。しかし，このような処理に対しては，前章までに述べた更新トランザクションに対する規則を適用すればよい。すなわち，実際には全くwriteを行わないような更新トランザクションにより目的を達成することができる。

5．おわりに

制限されたマルチバージョン・データベース・モデルと，このモデル上での現実的な同時実行制御機構について述べた。データベース管理において，マルチバージョンの導入による柔軟なシステムの構築は，特にインタラクティブなアクセスの多い環境では重要な意味を持つと考えられる。

一方，現実の単一バージョンのシステムでは２相ロック・プロトコルによる同時実行制御がほとんどである。我々のシステムはこれらのシステムにおいて，(1)ロック・プロトコルの変更，(2)マルチバージョンの導入，という拡張を行うことによって実現することが可能である。ロック・プロトコルの変更は単純な変更であり，またマルチバージョンの導入も，通常のシステムでは障害回復用のコピーが維持されていることから，容易に行うことが可能であると考えられる。したがって，本稿で提案した方式は，マルチバージョン・データベースを実際に提供する上で現実的な方式の１つであると思われる。

参考文献

[BAYE80a] Bayer, R. et al., "A Distributed Concurrency Control in Distributed Systems," Proc. 6th Int. Conf. on Very Large Data Bases, pp.275-284, 1980.

[BAYE80b] Bayer, R. et al., "Parallelism and recovery in database systems," ACM TODS 5-2, pp.139-156, 1980.

[BERN83] Bernstein, P.A. and Goodman, N., "Multiversion Concurrency Control-Theory and Algorithms," ACM TODS 8-4, pp.465-483, 1983.

[BUCK83] Buckley, G.N. an Silberschatz, A., "Obtaining Progressive Protocols for a Simple Multiversion Database Model," Proc. 9th Int. Conf. on Very Large Data Bases, pp.74-80, 1983.

[CHAN82] Chan, A., et al., "The Implementation of An Integrated Concurrency Control and Recovery Scheme," ACM SIGMOD, pp.184-191, 1982.

[DUBO82] DuBourdieu, D.J., "Implementation of Distributed Transactions," Proc. 6th Berkeley Workshop on Distributed Data Management and Computer Networks, pp.81-94, 1982.

[PAPA84] Papadimitriou, C.H. and Kanellakis, P.C., "On Concurrency Control by Multiversions," ACM TODS 9-1, pp.89-99, 1984.

[STER81] Sterns, R.E. and Rozenkrantz, D.J., "Distributed Database Concurrency Control Using Before Values," ACM SIGMOD, pp.74-83, 1981.

[YANN84] Yannakakis, M. "Serializability by Locking," JACM 31-2, pp.227-244. 1984.