【排他制御（同時実行制御）を極める】

はじめに

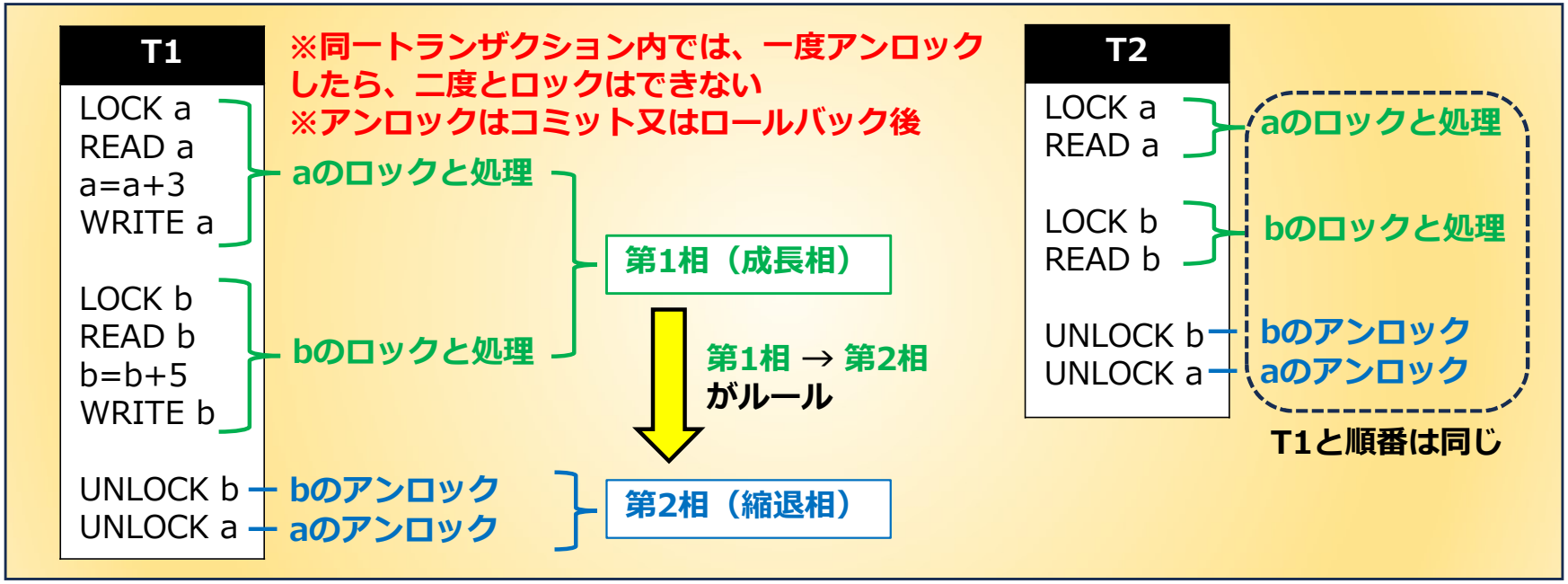
先の投稿のとおり、ACID特性の一貫性及び独立性を実現する機能として**ロック**による**排他制御（同時実行制御）**があります。これ以降の説明は全て、２つのトランザクションT1とT2があり、両者がデータa, bを共有資源としていることを前提とします。まず、以下のとおり、基本的事項について、説明します。

デッドロック	T1がa, bの順でロック、T2がb, aの順でロックする場合、双方とも相手のアンロックを待ち続ける状態
直列化可能性	T1とT2を並列に実行した結果と、完全に独立して順番に実行（T1→T2又はT2→T1）した結果とが等しい場合
競合等価	T1とT2とでデータを読み込む順番が同じ（aの後にb又はbの後にa）場合のT1とT2の関係
競合直列可能性	競合等価の場合の直列化可能性

2相ロック方式

2相ロック方式は、確実に直列化可能性を保証するロック方式です。この「確実に保証」には、直列化可能性を保証できない場合にはトランザクションを中止するなどの措置も含まれています。まず、2相ロック方式とは、下図（2相ロック方式その1）に示すルールがあります。下図のT1とT2は、競合等価なので、自然とロックする順番も同じになります。そのため、T1が処理を開始するとT2は何もできなくなり、その逆も同様であるため、直列化可能性は保証されます。

なお、競合等価でない場合、つまりＴ２のロックの順番がＴ１と逆の場合、デッドロックの可能性がある。

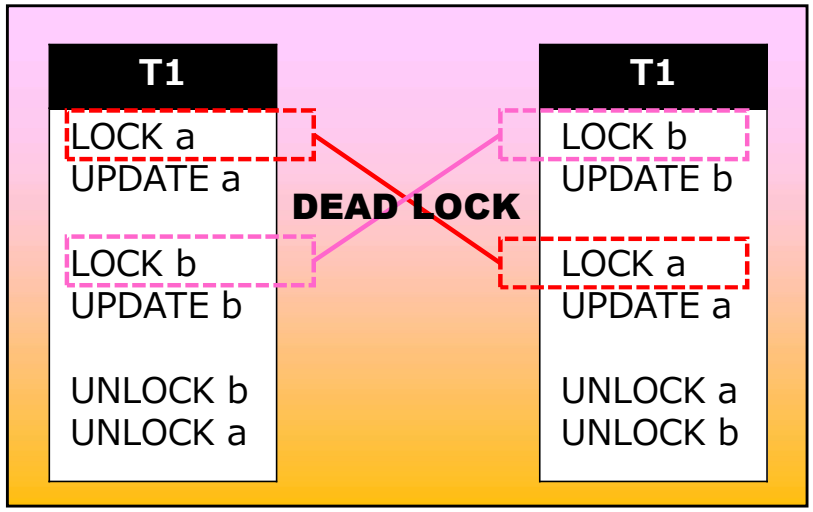


図：2相ロック方式その1

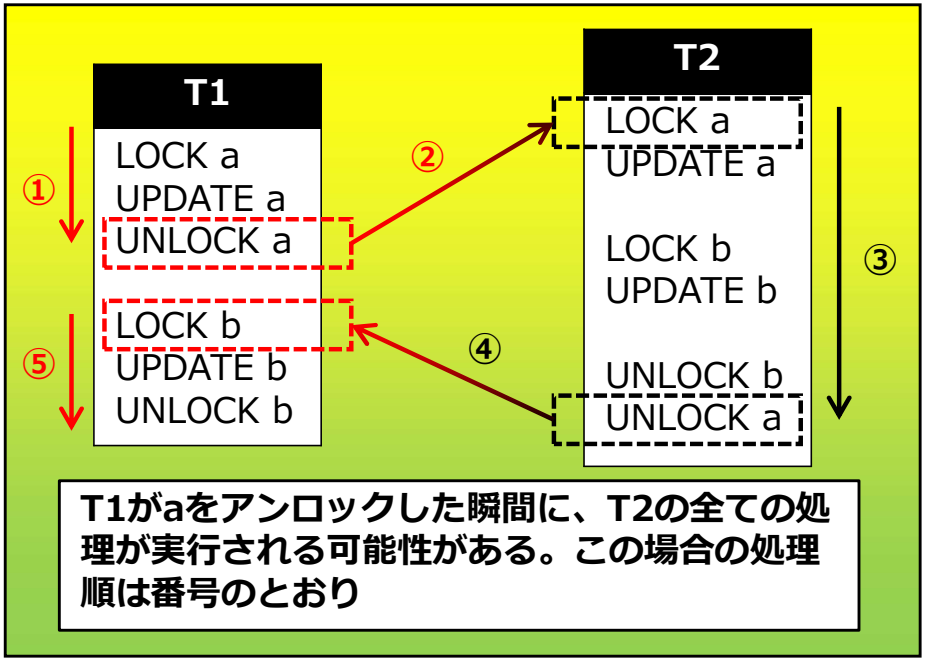
本資料は正確性に欠く部分及び誤字脱字等も多いと思います。そのため、本資料に起因した損害等については、管理人として責任を負いかねますので御了承ください。
© 2025 spine-team.com

しかし、下図（ 2相ロック方式その2）のとおり、競合等価でないために自然にロックする順番が逆の場合には、典型的なデッドロックとなります。この場合、デッドロックを解除するためには、どちらかのトランザクションをアボート（ロールバックにより、そのトランザクションで行われた全ての更新を廃棄）する以外に方法はありません。よって、**デッドロックを回避するためには、競合等価の元、ロックする順番も同じにすることです。**

以上は、T1とT2の両方が2相ロック方式を実行している場合であり、下図（2相ロック方式その３）のとおり、T1が2相ロック方式を実行していない場合は、直列化可能性は保証されません。



図：2相ロック方式その2



図：2相ロック方式その3

本資料は正確性に欠く部分及び誤字脱字等も多いと思います。そのため、本資料に起因した損害等については、管理人として責任を負いかねますので御了承ください。
© 2025 spine-team.com

デッドロックの回避策

ロックによる排他制御におけるデッドロックの回避策は、2相ロック方式であっても、通常のロックを実行する場合であっても、ロックをかける順番を同じにすることが基本です。つまり、ロックを使用しない排他制御であれば、そもそもデッドロックは起こり得ないはずなので、それらの排他制御の概要を下図（その他の排他制御）に示します。

時刻印方式（時刻印アルゴリズム

トランザクション毎に結節点で**時刻印（タイムスタンプ）**を付与し、どこかで競合した場合には、その時の時刻印を比較して、**時刻印**が後のトランザクションをアボートすることで**排他制御**を行っている。ただし、競合の発生をロックの有無で感知しているのか、それとも何らかの方法で監視しているのかについては**調査中**

楽観的制御法

ロックによる排他制御は**悲観的制御法**と呼ばれることから、当然、その逆の、**排他制御**として**楽観的制御法**もある。

楽観的制御法は、更新前イメージを保存してから、ロックなしでデータを更新する。そして、記憶装置に書き出す前に、更新前イメージと記憶装置のデータが同じであること、つまり別のトランザクションに上書きされていなければ、更新データを書き出す。この時のＤＢの更新時刻の管理は、**時刻印（タイムスタンプ）**がよく用いられる。また、別のトランザクションに上書きされていれば、自トランザクションはアボートすることで、**排他制御**を行っている。

多版同時実行制御

多版同時実行制御（MVCC：Multi Version Concurrency Control）の実行プロセスは以下のとおり

①複数のトランザクションが同時に開始された場合、その瞬間のＤＢのスナップショット（初期SS）をとる。
②初期SSに対して、各トランザクションはデータ更新を行い、そのスナップショット（更新SS）をとる。
③トランザクションID等を利用して、初期SSに対して、一番更新時間が古い更新SSで更新し、その更新したデータに2番目に更新時間が古い更新SSで更新し、その作業を最後まで行う。

この方法は、**排他制御特有である更新側の待ち時間もなく**、かつ**トランザクションのやり直し等も不要**になる。そのため、**MVCC**により、ACID特性の一貫性と独立性が簡単に確保できる気がするが、大量のスナップショットの生成によるオーバヘッドやその他の問題もあるらしい。

また、上記①の初期SSを獲得する際に競合が起きれば、ロックによる**排他制御**を行うため、デッドロックの可能性もある。

図：その他の排他制御

本資料は正確性に欠く部分及び誤字脱字等も多いと思います。そのため、本資料に起因した損害等については、管理人として責任を負いかねますので御了承ください。
© 2025 spine-team.com