データベースS 講義資料　第13回 同時実行制御

九州工業大学 情報工学部 システム創成情報工学科　講義担当：尾下真樹

1. 同時実行制御

1.1. トランザクション

一般に、データベースシステムでは、多数の処理を並列かつ高速に実行することが望まれる。このとき、一つのまとまった処理のことを**トランザクション**と呼ぶ。トランザクションは、必ず最後まで実行されるか（コミット）、万一トランザクションが実行できないときには実行前の状態に戻るか（アボート）のどちらかで終わるように処理され、トランザクションが途中まで実行された中途半端な状態で終わることはない。このことにより、データベースの整合性が保証される。

ただし、複数のトランザクションが同一のデータを読み書きすると、データの不整合が生じるため、そのようなことが起こらないような工夫が必要になる。基本的には、**計画的な並列処理**を行い、あらかじめトランザクション同士が影響を与えないように同時実行スケジュールを決定するか、**逐次的な並列処理**を行い、トランザクション同士が影響を与えそうになったらトランザクションを一時停止や強制終了させたりするか、どちらかとなる。一般的には、どのような処理が行われるかがあらかじめ全て分かっていることは少ないため、後者の方法で処理を行うことが多い。

1.2. 計画的な並列処理

複数のトランザクションをあるタイミングで並列に実行するとき、同じトランザクションを何らかの順番で直列に実行したときと同一の実行結果になる場合、その並列実行スケジュールは**直列化可能**と言うことができ、並列実行しても問題ないことになる。

2つの実行スケジュールの結果が等しいかどうかを評価するための方法として、**ビュー等価**や**競合等価**がある。ビュー等価では、各トランザクションにおける各データ読み込みについて、どのトランザクションが書き込んだデータを読み込んでいるかどうかを、2つの実行スケジュールにおいて比較し、全ての読み込みについて同一の書き込み結果を読み込んでいれば、等価であると判断する。競合等価では、簡略化した評価方法として、全てのデータについて各トランザクションのアクセス順序のみを判定し、全データでのアクセス順序が同じであれば、等価であると判断する。

ある並列実行スケジュールが与えられたとき、その**先行グラフ**を描くことで、その並列実行スケジュールと競合等価な直列実行スケジュールが存在するかどうかを、簡単に判定することができる。先行グラフは、各データについて、トランザクション A がトランザクションBの書き込みよりも先に、そのデータを読み書きしているときに、トランザクションA→Bのエッジを描く。先行グラフ内に巡回がなければ、必ず、競合等価な直列実行スケジュールが存在する。一方、先行グラフ内に巡回がある場合は、競合等価な直列実行スケジュールは必ず存在しない（厳密には、存在する可能性もあるが、先行グラフからは判断できない）。

1.3. 逐次的な並列処理

逐次的な並列処理では、**ロック**という仕組みが一般に用いられる。ロックとは、トランザクションがあるデータを読み書きするときに、その間に他のトランザクションがそのデータにアクセスできないようにするものである。トランザクションの途中に、トランザクションが影響するデータが書き換えられてしまうと、トランザクションが失敗したときに元の状態に戻すことができなくなってしまうため、そのような事態にならないよう、一度ロックしたデータはトランザクションが終了するまでロックを保持し、トランザクションが完了して初めて全てのデータのロックを解除する。このようなロックの仕方を、**二相ロック**と呼ぶ。

しかし、単純にロックを用いるだけでは、複数のトランザクションがお互いの実行に必要なデータをロックし合い、どのトランザクションも実行できなくなってしまう**デッドロック**と呼ばれる状態になることがある。このような問題を解決するためには、デッドロックを検出して原因となっているトランザクションを一旦強制終了させる（デッドロックの検知）、デッドロックが生じないようまとめてデータをロックする（デッドロックの防止）、デッドロックが発生していても全体が停止することがないように一定ルールに従ってトランザクションを終了させる（デッドロックの回避）、などの方法がある。

このうち、3番目の方法（デッドロックの回避）については、**ウェイト・ダイ方式**、**ワウンド・ウェイト方式**、などのやり方がある。ウェイト・ダイ方式は、あるトランザクションが別のトランザクションが既にロックしているデータにアクセスしようとしたとき、すでにロックしているトランザクションの方が古ければ、新しくロックしようとしたトランザクションを強制終了させる。（この時点ではデッドロックになっているかは分からないが、その可能性があるため、念のために強制終了させる。）一方、ワウンド・ウェイト方式は、あるトランザクションが別のトランザクションが既にロックしているデータにアクセスしようとしたとき、後からロックをしようとしたトランザクションの方が古ければ、最初にロックしようとしたトランザクションを強制終了させる。どちらの方式も、最初に開始されたトランザクションの方が優先されるため、たとえ何度か強制終了されたとしても、しばらく処理を行っているうちに、必ずトランザクションは完了することが保証される。ただし、このような処理方法では、実際にはデッドロックが生じておらず、本来であればトランザクションをアボートする必要がない場合でも、トランザクションがアボートされることがあるため、非効率的になってしまう可能性がある。