17_Transactions

2021年5月25日

トランザクションとは

データベースに対する複数の処理を一つにまとめたかたまり(単位)である。データベースはトランザクションの正常な処理を保障しなければならない。

トランザクションはデータベース操作言語SQLまたはC＋＋やJavaといったプログラミング言語によって記述される。また、トランザクションの始まりと終わりは明確に記述される。

代表的な利用例としては、銀行の口座間の送金やクレジットカードの決済システムなどがある。

トランザクションの大きな特徴としては…ACIDという4つの特性を持つ。

ACID特性とは

不可分性(原子性)-Atomicity(ACID)

トランザクションは一つのかたまりであり、全てが実行されずに、途中で実行失敗すると既に行われた実行結果も無効になる。ALL or NOTHING

独立性-Isolation(ACID)

トランザクションは、並列実行の他のトランザクションを含む他の全てのデータベース操作の影響を受けず、一つのトランザクションとして独立している。

永続性(耐久性)-Durability(ACID)

一度、実行されたトランザクションは、システムが落ちようと、その結果が失われることはない。

一貫性-Consistency(ACID)

トランザクションはデータベースの一貫性を保たなければならない。トランザクションの前後でデータベースに矛盾が生じることがあってはならない。

実際のトランザクションの例

少し単純化したトランザクションの記述例

処理内容

- **read(X)**…ディスク上のデータベースから、データXを読み込み、トランザクションが実行されているメインメモリ内で変数Xとして利用可能の状態にする。
- **write(X)**…メインメモリ内の変数Xの値を、データベース内のデータXの値として書き出す。

T_i:
read(A);
$A := A - 50$;
write(A);
read(B);
$B := B + 50$;
write(B).

※実際のデータベースでは、必ずしもwriteという操作の後にすぐにデータベースが更新されるわけではないが、今回はwriteの操作があると即座にデータベースが更新されると考える。

各種ストレージ

揮発性メモリ

揮発性メモリは、データへのアクセスの速度は非常に早いが、電源が供給されていないと、その中身を保つことができない。揮発性メモリには、メインメモリやキャッシュメモリなどがある。

不揮発性メモリ

不揮発性メモリは、揮発性メモリに比べると、データへのアクセス速度は遅いが、電源が落ちても、その中身を保つことができる。不揮発性メモリには、磁気ディスクやフラッシュメモリなどが存在する。

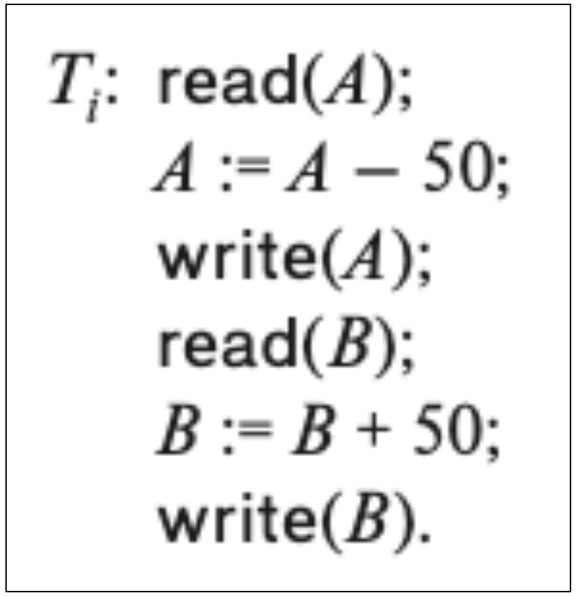
Stable Storage

理論的にデータを失うことはありえない。データの複製を作成し、複数の不揮発性メモリに保存することで、不揮発性メモリが一つ損傷しただけではデータは失われない。

不可分性(Atomicity)

不可分性がないと…

実行失敗した際に、トランザクションが途中までのみ実行され、データベースに齟齬が生じてしまう。



例えば…

write(A)とread(B)の間で 実行が失敗すると、Bへの処理は行われず、ただAの値が減ってしまうだけになってしまう。

そこで不可分性に基づいて…

途中でトランザクションの実行に失敗すると、logファイルのデータをもとに、ロールバックと呼ばれるトランザクション開始前の状態に戻る処理を行う。

logファイルとは…

トランザクションによってデータベースの値が変更されると、まずディスク上のlogファイルに記録される。記録されるのは、トランザクションの識別子、変更されたデータの識別子、変更前と変更後のデータの値である。

トランザクションの状態とその流れ

Active…実行中の状態

Partially committed…最後の実行を終えた状態

Failed…正常に実行されていないことが判明した状態

Aborted…ロールバックし、データベースがトランザクション開始前に戻った状態

Committed…正常に実行が完了した状態

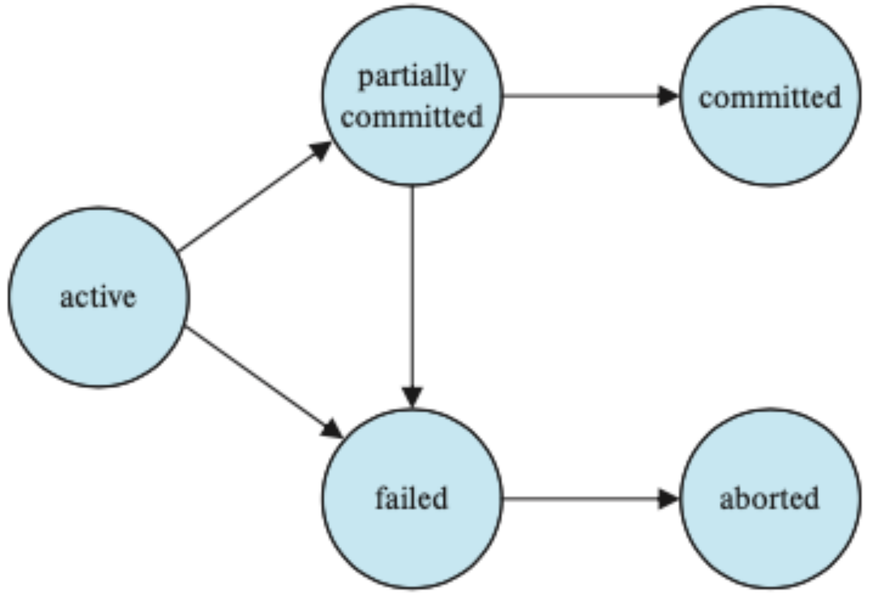


Figure 17.1 State diagram of a transaction.

1. active→partially committed→committed
   トランザクションが正常に実行完了する。
2. active→partially committed→failed→aborted
   トランザクションの処理が正常に行われても、更新されたデータがディスクに書き出される前にエラーが起きると、トランザクションは失敗となる。
3. active→failed→aborted　トランザクション実行中のエラーにより、失敗する。

トランザクションの独立の必要性

複数のトランザクションを互いに干渉させずに実行しようと思うと、トランザクションを一つずつ順番に実行していく逐次処理がシンプルで有効である。複数のトランザクションの並列実行を認めると、複雑さが増してしまう。

しかし、トランザクションの並列実行には大きなメリットが2つある!!

- トランザクションには並列処理が可能な要素があり、一方のトランザクションの処理をCPUで行い、他方のトランザクションはディスクへアクセスしているということが可能である。CPU、ディスクを無駄なく使え、スループットが増える。
- トランザクションには実行時間が長いものがあれば、短いものもある。短いトランザクションが長いトランザクションを待つのは効率が悪い。データベースの異なる部分を扱うトランザクションであれば、並列実行させる方が良い。

トランザクションの逐次処理の例

T1とT2という2つのトランザクション

T_1	T_2
read(A)	
$A := A - 50$	
write(A)	
read(B)	
$B := B + 50$	
write(B)	
commit	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
	$B := B + temp$
	write(B)
	commit

T1→T2

T_1	T_2
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
	$B := B + temp$
	write(B)
	commit
read(A)	
$A := A - 50$	
write(A)	
read(B)	
$B := B + 50$	
write(B)	
commit	

T2→T1

トランザクションが2個であれば逐次処理の順番は2!=2通り、n個であればn！通り

※階乗なので計算量は莫大

トランザクションの並列処理の例

○正しい実行例

T_1	T_2
read(A)	
$A := A - 50$	
write(A)	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
read(B)	
$B := B + 50$	
write(B)	
commit	
	read(B)
	$B := B + temp$
	write(B)
	commit

T1→T2のスケジュールで逐次処理を行った結果と一致

×誤った実行例

T_1	T_2
read(A)	
$A := A - 50$	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
write(A)	
read(B)	
$B := B + 50$	
write(B)	
commit	
	$B := B + temp$
	write(B)
	commit

T1のwrite(A)に上書きされ、T2のwrite(A)の処理は無効にAとB合計に矛盾が生じてしまう。

並列実行の結果は、いずれかの順番で逐次実行した結果と一致する必要がある。