具体的な講義の項目

- イントロダクション（本日）
- データベースのモデル
- 関係データベース（関係代数、関係論理）
- 正規化
- 問い合わせ言語
- トランザクション処理
- データベースの内部構造
- データベースの問い合わせ処理
- その他（オブジェクト指向DB、アクティブDB等々）

トランザクション (Transaction)とは

- データベースに要求される事項の一部
  - 複数の利用者がデータベースを共有する
  - 故障や異常が生じた場合にも内容を失わない
- データベース操作をまとめる必要性

ちょっと考えてみよう（7－1）

- データベースの操作をまとめる必要性を、データベースの共有の面と障害回復の面から、自分の言葉で書いてみましょう
  - これから話す内容ですが、この時点で考えておきましょう

トランザクション の性質

- 処理の論理的単位
  - 成功して終了：コミット (Commit)
  - 失敗して終了：ロールバック (Rollback)
    - アボート (Abort) とも呼ばれる
- ACID 性
  - A: 原子性 (Atomicity)
  - C: 一貫性 (Consistency)
  - I: 分離性 (Isolation)
  - D: 持続性 (Durability)

ACID 性（1）

- A: 原子性 (Atomicity)
  - All-or-nothing。あるトランザクション内の操作の結果は、他のトランザクションから全て見えるか、全く見ることができないかのいずれかである。
- C: 一貫性 (Consistency)
  - あるトランザクションによって、データベースの状態は、一貫性制約を満たした状態から、一貫性制約を満たした別の状態に移る。

ACID 性（2）

- I: 分離性 (Isolation)
  - トランザクションが終了するまでは、トランザクション中に変更したデータは他のトランザクションによって見ることが出来ず、トランザクション中に参照したデータは他のトランザクションによって変更することが出来ない。
- D: 持続性 (Durability)
  - 成功して終了したトランザクションの結果は、他のトランザクションによる明示的な変更がない限り、たとえその後システムに障害等が発生しても、永久的に残る。

メモ： 新しいコンセプト BASE トランザクション クラウドKVS向け
(Basically Available, Soft state, Eventually consistent)

ちょっと考えてみよう（7－2）

- 原子性 (Atomicity)が無い場合、どのような問題が発生するか、問題の例を一つ上げてみましょう

SQL によるトランザクション制御

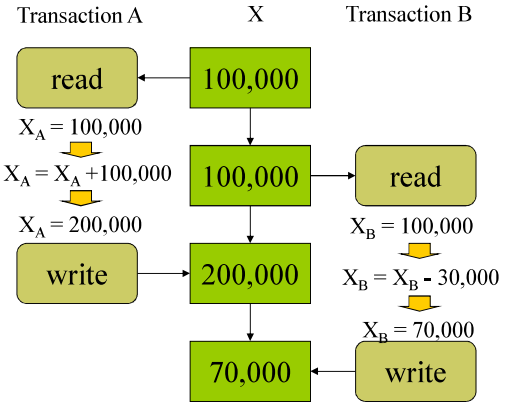
- BEGIN WORK
- COMMIT WORK
  - BEGIN WORKと COMMIT WORKに挟まれた処理の結果は、COMMIT WORKが終了するまで他のトランザクションから見ることが出来ない。
- ROLLBACK WORK
  - BEGIN WORKとROLLBACK WORKに挟まれた処理の結果は、ROLLBACK WORKが終了すると何も残らない。
- DCL: Data Control Language

直列実行性 (Serializability)

- 同時実行制御における正当性の基準
  - 同時に実行したトランザクションの結果が、直列に実行したトランザクションの結果と等しい
  - 直列実行の順番は問題としない

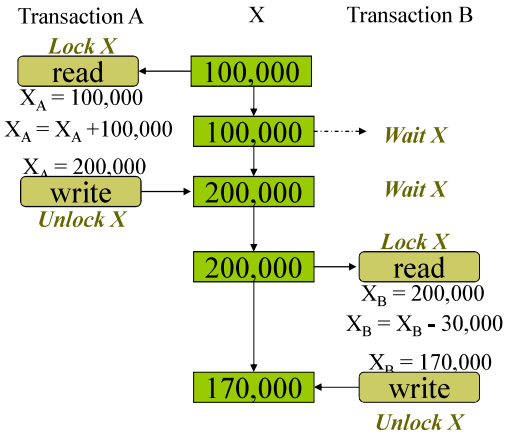
- 変更消失 (Lost Update) 問題
  - 直列実行性が満たされない1つの例
  - トランザクション間で資源を共有
  - 上書きによって先に書かれた内容が消失

Lost Updateの例



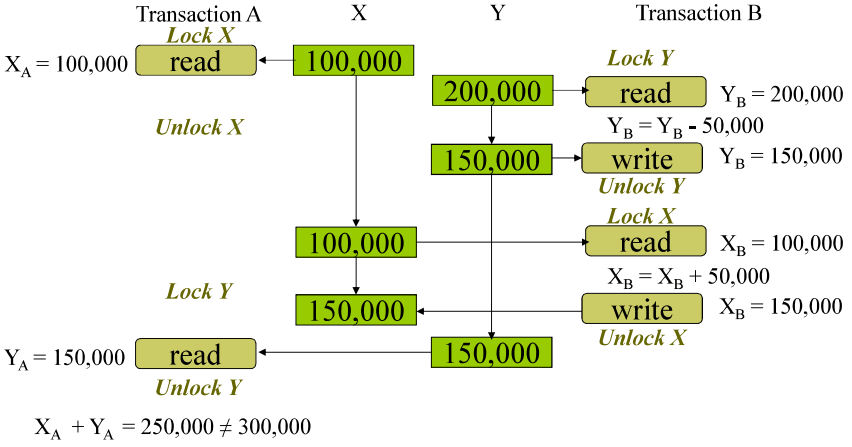
- 本来 A → B でも、B → A でも結果は 170,000 にならなければならない。
- 結果は70,000
- 最初の+100,000が喪失
- ロックの必要性

ロックによる直列実行



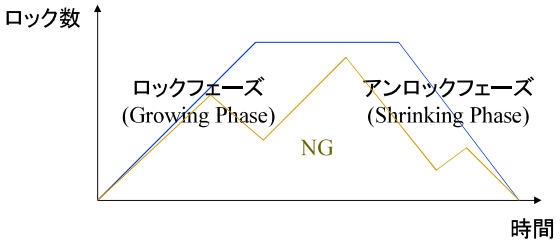
- 誰かが先にロックを取得したら、解放されるまで待たされる
- A → B でも、B → A でも結果は 170,000 で同じ

ロックだけでは一貫性が保てない例:



2 相ロックの必要性

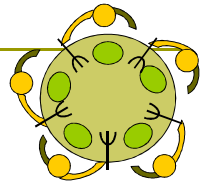
- 2 相ロック(Two-Phase Lock)
  - ロックフェーズとアンロックフェーズを分ることで直列化



ちょっと考えてみよう(7ー3)

- 単に項目をロックするだけでは不十分な理由を整理してみましょう。

デッドロック (Deadlock)

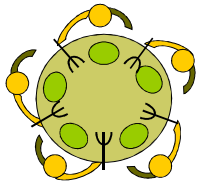


哲学者の食事問題

- 発生
  - お互いが、ロックの解放を待ち合う
    - 処理が前に進まない
- 検出 (Detection)
  - 時間監視:タイムアウトによる検出
    - 複雑なトランザクションでは、実行されないものが出てくる。
  - TWFG (Transaction Wait For Graph) による検出
    - 待ち合わせ関係のグラフ内のループ検出(コスト高)

デッドロック (Deadlock)

- 解消 (Dissolution)
  - いずれかのトランザクションのアボート
    - ロールバック
    - Livelock の可能性
- 回避 (Avoidance)
  - Atomic Lock
    - 全てのロックをアトミックに獲得
  - 優先順位の導入
    - 右側のフォークではなく、フォークに順番づけ



ロックの種類と階層化

- X (eXclusive)
  - 資源を専有することの宣言（排他）
- S (Share)
  - 資源を共有することの宣言

ロックの単位	例	同時実行の程度	制御の効率
小さい	タプル、フィールド	高い	悪い
大きい	リレーション、ページ	低い	良い

ロックの種類と階層化

- IS (Intention Share)
  - そのものが X でロックされること以外を許す
    - 例えば、下位のいくつかの要素を共有で読んでいるだけ
- IX (Intention eXclusive)
  - そのものが S、SIX、X でロックされること以外を許す
    - 例えば、下位のいくつかの要素を排他で専有しており、全体を共有することは許されない
- SIX (Share and Intention eXclusive)
  - 次に IS でロックされることのみを許す
    - 例えば、全体を読みながら、下位の要素を変更している場合、変更していない要素の共有のみを許す

階層ロックの両立関係

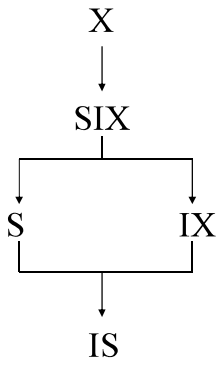
	N	IS	IX	S	SIX	X
N	○	○	○	○	○	○
IS	○	○	○	○	○	×
IX	○	○	○	×	×	×
S	○	○	×	○	×	×
SIX	○	○	×	×	×	×
X	○	×	×	×	×	×

階層ロックの遷移関係

	N	IS	IX	S	SIX	X
N	N	IS	IX	S	SIX	X
IS	IS	IS	IX	S	SIX	X
IX	IX	IX	IX	SIX	SIX	X
S	S	S	SIX	S	SIX	X
SIX	SIX	SIX	SIX	SIX	SIX	X
X	X	X	X	X	X	X

階層ロックの優先度グラフ

- L2 が L1 よりも強いとは、両立関係で L1 のカラムが×である場合には、L2 でも必ず×であることをさす。



ちょっと考えてみよう（7－3）

1. Aさんは、秋葉原支店関係のGDC10が1つ売れたので，在庫数を1減らしたい。一方Bさんは、秋葉原支店関係でのFLC33の販売価格を知りたい。効率良く実行するためには、Aさんのトランザクションと Bさんのトランザクションは何にどのようなロックをかけるべきか。
2. Aさんは、秋葉原支店関係のGDC10が1つ売れたので，在庫数を1減らしたい。一方Bさんは、秋葉原支店関係の販売価格の平均を計算したい。この場合には、Aさんのトランザクションと Bさんのトランザクションは何にどのようなロックをかけるべきか。
3. 1．2．において、SとXロックしかない場合に比べてどのような点が改善がされているか。