## 2相ロック：使うときに指定のテーブルをロックし、終わった時に全て解除する

トランザクションを複数同時に行えてしまうと、
片方が操作中でテーブル内の値の変更が反映される前にもう一つのトランザクションがその情報を元にしてさらに操作をおこなってしまうと、両者の操作実行後のデータが正しくなく書き込まれたりするので、データに矛盾が生じてしまう

コミットされてないトランザクションはデータの更新が反映されてない状態。その実行中に他のトランザクションが実行できる場合、更新が反映される前のデータを参照したり、それを前提とした値の更新が行われ、間違った値が書き込まれ、一貫性が保てなくなる

## デッドロック：2つ以上のトランザクションが互いにロックを解除するのを待ち続ける状態。

質問

デッドロックがかかるパターンはDBのテーブルを順番に処理するトランザクションのパターンが複数ある時、複数のユーザーが、別々のトランザクションを行った時に、パターンで利用しているテーブルがお互いにロックしていて身動きが取れず、ずっと待機状態になるというものですか？
→ 正解

1. **複数のトランザクションが同時に実行される**:
   - 複数のユーザーが、異なるトランザクションを実行中である。

2. **トランザクションが異なる順序でリソースをロックする**:
   - 例えば、トランザクションAが**テーブル1→テーブル2**の順でロックを取得し、トランザクションBが**テーブル2→テーブル1**の順でロックを取得しようとする場合、デッドロックが発生する可能性があります。

3. **リソースの排他ロック（Exclusive Lock）がかかる**:
   - 特に更新系（INSERT、UPDATE、DELETE）操作では、データの排他ロックがかかりやすいです。これにより、別のトランザクションが同じリソースにアクセスできなくなります。

4. **リソース解放が行われない待機状態になる**:
   - トランザクションAとトランザクションBがお互いのロックが解放されるのを待ち続けるため、無限に待機状態になります。

---

### 例: デッドロックの具体例
#### 状況
以下のテーブルがあるとします：

- **テーブルA**
- **テーブルB**

2つのトランザクションが次のように動作します：

- **トランザクション1**:
  1. **テーブルA**をロックする。
  2. **テーブルB**をロックしようとする。

- **トランザクション2**:
  1. **テーブルB**をロックする。
  2. **テーブルA**をロックしようとする。

#### 結果
- トランザクション1は、テーブルAをロックした状態でテーブルBのロックが解放されるのを待機します。
- 同時に、トランザクション2は、テーブルBをロックした状態でテーブルAのロックが解放されるのを待機します。
- **双方が待機状態となり、デッドロックが発生します。**

---

### デッドロックを防ぐ方法
1. **リソースアクセスの順序を統一する**:
   - すべてのトランザクションが同じ順序でテーブルや行をロックするように設計します。
   - 例: 「テーブルA → テーブルB」の順序に統一。

2. **トランザクションを短時間で終了させる**:
   - トランザクションをできるだけ短く保つことで、ロックの競合を減らします。

3. **適切なロックの粒度を選択する**:
   - 必要以上に大きなリソース（例: テーブル全体）をロックしないようにします。
   - 例: 行レベルロックを活用。

4. **タイムアウトを設定する**:
   - デッドロック発生時に自動的にタイムアウトを発生させ、トランザクションを中断します。
   - 例: 多くのデータベースは`LOCK_TIMEOUT`の設定が可能。

5. **デッドロック検出機能を活用する**:
   - 多くのデータベース（例: MySQL、PostgreSQL、SQL Server）はデッドロック検出機能を提供しており、デッドロックが発生した場合に片方のトランザクションを強制終了して復旧します。

---

デッドロックの発生する確率が高いもの

### 複数のユーザーと複数のテーブルが絡む場合
特に複数のユーザーが複数のテーブルを異なる順序で操作するトランザクションが複数存在するとき、デッドロックの発生確率が高くなります。トランザクション設計時にリソースのロック順序を明確に定めることが、デッドロックを防ぐ重要なポイントとなります。