INDEXを使ってDBパフォーマンスチューニング

Chowdhury Rumman Rashid

参考 : SQL Performance Explained (Online version)

データベースインデックスとは ？

- インデックスは、データの検索速度を向上させるデータ構造です。
- 本の索引のようなものです。本全体をスキャンするのではなく、トピックを調べて、正しいページに直接ジャンプできます。
- インデックスがない場合、データベースは **フルテーブルスキャン** を実行する必要があり、大きなテーブルでは遅くなります。

インデックスが遅くなる理由

遅いインデックスの2要素：

- 低選択性（例：gender = ‘男’ -> 50%一致）
- 不適切な検索パターン（関数・ワイルドカード・列順序の誤り）

複合インデックス

複合インデックスとは？

- **複合インデックス** は、`(last_name, first_name)`のように複数の列に作成されるインデックスです。
- 列の順序が重要です。インデックスは最初の列でソートし、次に 2番目の列でソートします。

複合インデックスを使用するタイミング

1. **複数の列でフィルタリングする場合** ：`WHERE country = 'USA' AND city = 'New York'`のようなクエリは、`(country, city)`の複合インデックスを使用して行をより速く見つけることができます。
2. **フィルタリングとソートを同時に行う場合** ：`WHERE country = 'USA' ORDER BY city`は、データが国ごとにすでに正しくソートされているため、`(country, city)`のインデックスによって高速化されます。`WHERE country = 'USA'`だけで検索してもこのインデックス使用できますが、 `WHERE city = 'New York'`だけならインデックスを使用できません。
3. **インデックスによって「カバー」される場合** ：`SELECT`句と`WHERE`句がすべてインデックス内の列のみを使用している場合、データベースは主テーブルを調べる必要ありません。これは非常に高速です。

クエリ内の関数とインデックス

- **問題**: WHERE句でインデックス付きの列に関数を使用すると、インデックスが使えなくなる可能性があります。
- **理由**: インデックスは生のソートされた値を格納します。関数が適用されると、データベースは一致を見つけるために各行で計算を行う必要があり、インデックスをバイパスします。これにより、遅い**フルテーブルスキャン** が発生します。
  例:`SELECT * FROM Users WHERE LOWER(email) = 'bob@example.com';`

解決策：

- **クエリを書き換える** ：可能な場合は、インデックス付きの列に関数を使用しないようにします。
- **関数ベースのインデックスを作成する** ：一部のDBでは、
  `CREATE INDEX ON Users (LOWER(email))`のように、関数の結果に対してインデックスを作成できます

JOINクエリ

- JOINは片方のテーブルのキーをもう片方で探す処理なので外部キーや結合キーにインデックスがないと全表スキャンが発生
- ベストプラクティスは主キー（参照される側）と外部キー（参照する側）にインデックス付与
- 例：

```
-- 外部キーにインデックスを作成
CREATE INDEX idx_orders_user_id ON orders (user_id);
SELECT * FROM users JOIN orders ON users.id = orders.user_id;
```

インデックス設計の注意点

- 過剰インデックスしないように注意。1つのINSERT / UPDATE / DELETE で全インデックス更新が必要になる
- 実際に使用するクエリを考えてそれに基づいて設計する
- クエリするときバインド変数を使用する
  - セキュリティ: SQLインジェクション対策
  - 実行計画を再利用でパフォーマンス向上
  - 例: `SELECT * FROM Users WHERE id = :userId`