stackfuel

SQL Codeübersicht - Kapitel 4

Hier findest du die wichtigsten Befehle der Lektionen mit Beispielcode. Wenn du dir den Output jeder Query noch einmal anschauen willst, kannst du einen Blick in die Notebooks werfen.

Profiling

In MySQL verwendest du das Statement `SET profiling=1;` um das Profiling für Queries zu aktivieren. Du verwendest den Befehl `SHOW PROFILES;` um die aufgezeichneten Profildaten anzuzeigen.

```
# activate profiling
SET profiling = 1;

# execute query
SELECT * FROM customer;

#show execution times
SHOW PROFILES;
```

Du kannst auch zusätzliche Informationen zu einer bestimmten Query abrufen, indem du `SHOW PROFILE FOR QUERY <query_id>;` verwendest, wobei `<query_id>` die ID der spezifischen Query ist, für die du weitere Details sehen möchtest.

```
SHOW PROFILE FOR QUERY 5;
```

Ausführungsplan analysieren

Die Befehle `EXPLAIN`, `EXPLAIN FORMAT=TREE` und `EXPLAIN ANALYZE` sind nützlich, um Engpässe in SQL-Abfragen zu erkennen, die Verwendung von Indizes zu überprüfen und Optimierungsmöglichkeiten zu identifizieren.

Nutze `EXPLAIN` für die Übersicht über den Ausführungsplan.

```
EXPLAIN
SELECT * FROM customer
WHERE last_name = 'Smith';
```

Nutze `EXPLAIN FORMAT=TREE` zur visuellen Darstellung des Ausführungsplans einer Query in einer hierarchischen Struktur, um die Beziehungen zwischen den Schritten besser zu verstehen.

```
EXPLAIN FORMAT=TREE
SELECT * FROM customer
WHERE last_name = 'Smith';
```

stackfuel

Nutze `EXPLAIN ANALYZE` zur detaillierten Analyse des Ausführungsplans einer Query, einschließlich erwarteter und tatsächlicher Ausführungszeiten.

```
EXPLAIN ANALYZE
SELECT * FROM customer
WHERE last_name = 'Smith';
```

Queries optimieren

Bei der manuellen Optimierung geht es darum, die Abfrage selbst anzupassen, um sie schneller und effizienter zu machen. Die manuelle Optimierung erfordert ein gutes Verständnis der Datenbankstruktur und der verwendeten SQL-Techniken. Es gibt keine allgemeingültige Lösung für alle Abfragen, aber ein paar generelle Tipps:

1. **Auswahl der benötigten Spalten:** Wenn du nur bestimmte Spalten aus einer Tabelle benötigst, wähle diese gezielt aus, anstatt alle Spalten abzurufen. Das verringert den Umfang der Abfrage und verbessert die Antwortzeit.

2. **Verwendung von Aggregatfunktionen:** Wenn du aggregierte Informationen benötigst, wie z.B. die Summe oder den Durchschnitt einer Spalte, kannst du Aggregatfunktionen verwenden, um diese Informationen direkt von der Datenbank abzurufen, anstatt sie manuell in deinem Code zu berechnen.

3. **Filtern mit präzisen Bedingungen:** Stelle sicher, dass deine Bedingungen so spezifisch wie möglich sind, um den Umfang der Suche einzuschränken. Verwende geeignete Vergleichsoperatoren und kombiniere Bedingungen effizient, um die Anzahl der zu überprüfenden Datensätze zu minimieren.

4. **Optimierung von Joins und Subqueries:** Wenn du Joins verwendest, um Daten aus verschiedenen Tabellen zu kombinieren, überprüfe die Art des Joins und die Join-Bedingungen. In einigen Fällen kann es besser sein, alternative Join-Typen oder Subqueries zu verwenden, um die Performance zu verbessern. Umgekehrt gilt das auch für Subqueries. Diese werden oft schlecht optimiert und es kann eine Abfrage verbessern, wenn Joins statt Subqueries genutzt werden.

5. **Verwendung von Indizes:** Wie bereits erwähnt, können Indizes den Zugriff auf Daten beschleunigen. Überprüfe, ob geeignete Indizes vorhanden sind und ob sie optimal genutzt werden. Gegebenenfalls kannst du neue Indizes erstellen, um die Leistung zu verbessern.

6. **Überprüfung des Ausführungsplans:** Analysiere den Ausführungsplan der Abfrage, um zu sehen, wie die Datenbank die Abfrage verarbeitet. Das kann dir Aufschluss darüber geben, ob Optimierungen vorgenommen werden können, beispielsweise durch Neuanordnung der Bedingungen oder Veränderung der Join-Reihenfolge.