Speed up your Query - Strategien zur Optimierung von SQL-Queries

Ulrike Brenner
click-click IT Solutions e.U.
Wien

Schlüsselworte:

SQL, Performance Tuning

Einleitung

SQL-Queries, obwohl Basis all unserer Applikationen, wird oft wenig Augenmerk geschenkt...bis es zu Performanceproblemen kommt. In diesem Vortrag werde ich anhand von vergleichenden Code-Snippets Möglichkeiten präsentieren, wie wir unsere Abfragen tunen können. So verwenden wir immer wieder mal Sub-Queries, doch wie können wir diese Abfragen optimieren? Wie finden wir am besten alle Datensätze mit maximalem Wert? Und was machen wir, wenn wir jene Mitarbeiter eines Unternehmens suchen, die die 5 höchsten Gehälter bekommen? Für all diese Fragestellungen gibt es immer verschiedenste Lösungsmöglichkeiten, doch welche ist die Beste? Und ist für alle Anforderungen der selbe Lösungsweg der Schnellste? Diese und andere Fragen werde ich in meinem Vortrag behandeln, dabei verschiedene SQL-Funktionen betrachten und miteinander vergleichen. Es ist immer wieder spannend zu sehen, wie kleine Änderungen große Performanceunterschiede ausmachen.

Subquery oder Analytic Function MAX

Gesucht sind alle Mitarbeiter, die in ihrer Abteilung am meisten verdienen. Von diesen Mitarbeitern wollen wir neben dem Namen, das Gehalt und die Abteilung sehen. Als Basis für unsere Abfrage greifen wir auf das HR-Schema zu. Dabei müssen wir beachten, dass es Mitarbeiter geben kann, die keiner Abteilung zugeordnet sind.
Eine mögliche Lösung ist pro Mitarbeiter (pro Zeile in den EMPLOYEES) zu prüfen, ob das Gehalt dem höchsten seiner Abteilung entspricht.

```
SELECT E1.FIRST_NAME
     , E1.LAST_NAME
     , E1.SALARY
     , E1.DEPARTMENT_ID
  FROM HR.EMPLOYEES E1
 WHERE E1.SALARY = ( SELECT MAX(E2.SALARY)
                       FROM HR.EMPLOYEES E2
                      WHERE E2.DEPARTMENT_ID = E1.DEPARTMENT_ID
                        OR (    E2.DEPARTMENT_ID IS NULL
                            AND E1.DEPARTMENT_ID IS NULL
                           )
                   )
/
```

Wenn wir uns den Explain Plan für diese Abfrage anschauen, sehen wir, dass wir 2 Full-Table-Scans haben und Costs von 39.

Um die Performance zu verbessern können wir nun die Subquery folgendermaßen ändern, dass wir anstelle der Oder-Verknüpfung über die DEPARTEMENT_ID ein NVL legen. (Wir wissen, dass die Abteilungsnummer nicht -1 sein kann.)

```
SELECT E1.FIRST_NAME
     , E1.LAST_NAME
     , E1.SALARY
     , E1.DEPARTMENT_ID
  FROM HR.EMPLOYEES E1
 WHERE E1.SALARY = ( SELECT MAX(E2.SALARY)
                        FROM HR.EMPLOYEES E2
                     WHERE NVL(E2.DEPARTMENT_ID,-1) =
                           NVL(E1.DEPARTMENT_ID,-1)
                   )
/
```

Damit haben wir nun bereits die Costs auf 8 reduziert, greifen aber noch immer zwei mal in Form eines Full Table Scans auf die Tabelle EMPOLYEES zu.

Wenn wir nun aber die Analytic Function MAX in einer Inline View zu jedem Mitarbeiter selektieren und dann nur mehr jene ausgeben, deren maximales Gehalt, dem höchsten ihrer Abteilung entspricht, dann haben wir nur mehr einen Zugriff auf die Tabelle EMPLOYEES und die Costs reduzieren sich auf 4.

```
SELECT FIRST_NAME
     , LAST_NAME
     , SALARY
     , DEPARTMENT_ID
 FROM (SELECT FIRST_NAME
            , LAST_NAME
            , SALARY
            , DEPARTMENT_ID
            , MAX(SALARY) OVER (PARTITION BY DEPARTMENT_ID) AS MAX_SAL
         FROM HR.EMPLOYEES
      ) E
 WHERE E.SALARY = E.MAX_SAL
/
```

ORDER BY und ROWNUM oder Analytic Function DENSE_RANK

Gesucht sind diesmal jene Mitarbeiter, die die 5 höchsten Gehälter bekommen, wobei zu beachten ist, dass mehrere Mitarbeiter das selbe verdienen können.

Für die erste Lösung müssen wir die Inline Views verschachteln. Das bedeutet, wir suchen zuerst alle verschiedenen Gehälter, sortieren diese und geben dann die Pseudo-Column ROWNUM dazu.
Nun selektieren wir alle Mitarbeiter die eines der fünf höchsten Gehälter bekommen.

```
SELECT E1.FIRST_NAME
     , E1.LAST_NAME
     , E1.SALARY
     , E1.DEPARTMENT_ID
  FROM ( SELECT SALARY
              , ROWNUM POS
          FROM (SELECT DISTINCT
                       SALARY
                  FROM HR.EMPLOYEES
                 ORDER BY SALARY DESC
                )
        ) E_SAL
     , HR.EMPLOYEES E1
 WHERE E1.SALARY  = E_SAL.SALARY
   AND E_SAL.POS <= 5
 /
```

Nun im Vergleich dazu die analytic Function DENSE_RANK. Neben der leichteren Lesbarkeit haben wir in diesem Fall Costs von 4, im Vergleich zu Costs von 9 bei der ersten Methode.

```
SELECT E_RANK.FIRST_NAME
     , E_RANK.LAST_NAME
     , E_RANK.SALARY
     , E_RANK.DEPARTMENT_ID
  FROM (SELECT DENSE_RANK() OVER (ORDER BY SALARY DESC) EMP_RANK
             , FIRST_NAME
             , LAST_NAME
             , SALARY
             , DEPARTMENT_ID
          FROM HR.EMPLOYEES
       ) E_RANK
 WHERE E_RANK.EMP_RANK <= 5
/
```

NVL oder COALESCE

Die beiden Funktionen sind, falls wir nur 2 Parameter verwenden, sehr ähnlich. Es wird der erste Wert zurückgegeben, der NOT NULL ist. Aber warum sollen wir dann eine, der anderen vorziehen?
Aus Performancegründen wird empfohlen, COALESCE zu verwenden, wenn der zweite Parameter eine Funktion ist, da NVL immer beide Parameter berechnet, COALESE aber nur solange, bis einer der Parameter NOT NULL ist.

Um das zu verifizieren, erzeuge ich eine Funktion, die definitiv so nicht funktioniert.

```
CREATE OR REPLACE FUNCTION Fehlerhafte_Funktion
RETURN VARCHAR2
AS
    vZahl       NUMBER;
BEGIN
    vZAHL := 'abc';
END;
/
```

Nun werde ich die Funktion einmal mit NVL und das zweite Mal mit COALESE verwenden und wir erkennen den Unterschied.

Aus dem selben Grund sollten wir ein CASE einem NVL2 vorziehen.

Funktionen in der WHERE-CLAUSE

Wenn wir in den EMPLOYEES den LAST_NAME auf den Namen 'King' einschränken, erkennen wir, dass der Index EMP_NAME_IX verwendet wird. Was aber, wenn wird nicht wissen, ob der Name groß, klein, mit großem Anfangsbuchstaben o.ä. geschrieben wird. Die gängig Lösung ist, beide Seiten der Einschränkung auf UPPER (oder LOWER) zu konvertieren. Und plötzlich wird der Select extrem langsam. Es gibt nun verschiedene Möglichkeiten dieses Problem zu umgehen: wir legen eine zusätzliche Spalte an, in die immer der Name in Großbuchstaben geschrieben wird (z.B. durch einen DB-Trigger) und indizieren diese Spalte. Oder wir legen einen function based Index an.
Aber auch wenn wir zu einer Zahl „nur“ 0 addieren (bzw. NULL zu einem Character-String konkatenieren) schalten wir damit den Index aus.

Was wir bei allen zusätzlichen Indices bedenken sollten ist, dass diese natürlich beim Erzeugen/Ändern von Datensätzen Zeit kosten.

und implizite Datenkonvertierung

Wir haben erkannt, dass Funktionen unsere Indices ausschalten, außer einen function based Index. Kann es aber auch sein, dass wir gar keine Funktion verwenden und trotzdem wird der Index ausgeschalten?
Ich habe eine einfache Tabelle UB_EMPLOYEES erstellt, mit einem einzigen Feld, der CHAR_EMPLOYEE_ID, mit dem Datentyp VARCHAR2(3). Auf diese Spalte lege ich einen Index.
Bei der Einschränkung CHAR_EMPLOYEE_ID = 101, ändert die Datenbank die Einschränkung auf TO_NUMBER(CHAR_EMPLOYEE_ID) = 101 und wir sehen sofort, dass so der Index ausgeschalten wird. In diesem Fall müssen wir sicher stellen, dass die rechte Seite von der Datenbank als Character-Feld erkannt wird.

Context switch (SQL und PL/SQL)

Wenn eine PL/SQL-Funktion aus einem Select heraus aufgerufen wird, so finden ein so genannter „CONTEXT SWITCH“ statt, da SQL und PL/SQL auf 2 unterschiedlichen „engines“ arbeitet. Im Einzelfall kostet das nicht viel, aber über eine große Datenmenge sind diese Switches nicht zu vernachlässigen.

Dies passiert auch wenn wir „nur“ eine APEX-Funktion aufrufen z.B. v('APP_USER').

Optimizer STATISTICS

Die Datenbank greift auf seine Optimizer Statistiken über die Tabellen und Indices zu um über die Optimale Ausführung eines Selects zu entscheiden. Wenn diese Informationen veraltet sind, und sich z.B. die Datenmenge vollkommen geändert hat, kann der Optimizer nicht den schnellsten Weg finden. Somit macht es Sinn – vor allem, wenn es sukzessive zu Performanceverschlechterungen kommt – die Statistiken neu zu generieren, bzw. zu überprüfen, ob die automatische Erstellung der Statistiken aktiviert ist.

Einige allgemeine Tipps

- Kenne das Datenmodell (nicht nur welche Spalten gejoint werden, sondern auch welche Spalten indiziert sind)
- Kenne die Daten (welche Tabellen haben viele Datensätze)
- Selektiere nur Spalten, die du auch wirklich brauchst
- Verwende nur Tabellen, die du wirklich brauchst
- Schränke so früh wie möglich die Datenmenge ein
- Bei Erweiterungen/Änderungen von bestehenden Selects, überlege, ob es reicht eine Tabelle dazuzujoinen um weitere Spalten auszugeben, oder ob es nicht besser wäre, den ganzen Select zu überarbeiten.
- Bei Views: welche Abfragen greifen auf die Views zu (d.h. macht es Sinn in einer View alles zu selektieren, oder wäre es nicht sinnvoll die Abfragen zu splitten)
- Bei kleinen Tabellen kann ein full table scan schnell sein, trotzdem muss jeder full table scan genau angeschaut werden
- Am schnellsten sind Verknüpfungen mit AND und =
- Minimiere so weit als möglich die Anzahl der Aufrufe einer Tabelle

Kontaktadresse:

Ulrike Brenner
click-click IT Solutions e.U.
Welschgasse 8
A-1230 Wien

Telefon: +43 (0)1 3119425 - 30
E-Mail ulrike.brenner@click-click.at
Internet: www.click-click.at