Die Nacht ist zu kurz: 10 ETL-Performance-Tipps

Dani Schnider
Trivadis AG
Zürich/Glattbrugg, Schweiz

Einleitung

Beim Laden von großen Datenmengen in ein Data Warehouse gelten andere Regeln für die Performanceoptimierung als beispielsweise bei Abfragen in einem OLTP-System. Deshalb ist es wichtig zu verstehen, wie ETL-Prozesse implementiert werden müssen, damit sie in der erforderlichen Zeit ausgeführt werden können. Wenn „die Nacht zu kurz" ist, um alle ETL-Jobs durchzuführen, ist Handlungsbedarf notwendig.

Wenn Sie die Daten in Ihr Data Warehouse mittels SQL-Scripts, PL/SQL-Packages und Views laden, oder wenn Sie ein ETL-Tool verwenden, das in der Lage ist, SQL-Befehle auszuführen, können die nachfolgenden Tipps helfen, effiziente ETL-Jobs zu implementieren oder die Performance von langlaufenden Jobs zu verbessern. Die Liste erhebt keinen Anspruch auf Vollständigkeit, zeigt aber typische Performancemaßnahmen auf, wie sie in vielen Data Warehouses zum Einsatz kommen.

Tipp 1: Mengenbasierte Verarbeitung

Der erste Tipp sollte für jeden ETL-Entwickler offensichtlich sein: Mengenbasierte Operationen in SQL laufen in der Regel schneller ab als zeilenbasierte Ausführungen in prozeduralen Sprachen. Eine INSERT/SELECT-Anweisung oder ein CREATE TABLE AS SELECT ist eine viel bessere Lösung als eine Cursor-Schleife in PL/SQL. Auch bei komplexen Transformationen ist es oft möglich – und empfehlenswert –, eine zeilenbasierte Cursor-Schleife in einen mengenbasierten SQL-Befehl umzuschreiben. Wer mit Oracle Warehouse Builder (OWB) gearbeitet hat, weiß, wie wichtig es war, die Konfiguration der Mappings auf *set-based* umzustellen.

ELT-Tools wie Oracle Data Integrator (ODI) sind in der Lage, SQL-Befehle direkt in der Datenbank auszuführen. Viele ETL-Tools bieten spezielle Funktionen oder Optionen zur Ausführung von Ladeprozessen im mengenbasierten ELT-Modus. Ist dies nicht möglich, unterstützen sie häufig „*Bulk Processing*", d.h. es können mehrere Datensätze in einem Schritt eingefügt werden.

Alle folgenden Tipps basieren auf der Annahme, dass Sie (oder Ihr ETL-Tool) die Ladeprozesse mit SQL-Anweisungen ausführen können.

Tipp 2: Nested Loops vermeiden

Ein *Nested Loops Join* ist eine wunderbare und effiziente Join-Methode für kleine Ergebnismengen in einer OLTP-Anwendung. Wenn Sie wenige Datensätze aus einer großen Tabelle auswählen und diese mit einer anderen Tabelle verknüpfen wollen, sind *Nested Loops* in Kombination mit Indexzugriffen der schnellste Weg, die gewünschten Datensätze zu finden. Aber ETL-Prozesse lesen normalerweise viele (oder alle) Datensätze einer Tabelle und verknüpfen sie mit anderen Tabellen. Für diese Art von Abfragen ist ein *Hash Join* definitiv die bessere Wahl.

Wenn ich den Execution Plan eines ETL-Jobs überprüfe, sehe ich mir zuerst die Join-Methoden an. Enthält der Ausführungsplan *Nested Loops Joins*, ist dies oft ein Indikator für schlechte Performance. Der Optimizer entscheidet sich für diese Join-Methode, wenn die Selektivität klein ist, d.h. wenn nur eine kleine Teilmenge der Daten gelesen wird. Wenn dies wirklich der Fall ist, ist es eine gute Entscheidung, aber oft wird der Optimizer durch unnötige Indizes oder komplexe WHERE-Bedingungen "verwirrt" (siehe Tipps 3 und 4).

Tipp 3: Unnötige Indizes löschen

Viele Data Warehouses sind „überindiziert". Im Gegensatz zu OLTP-Systemen werden Indizes in einem Data Warehouse jedoch viel seltener benötigt. Für viele Datenbankadministratoren und Entwickler ist das Erstellen von Indizes jedoch immer noch der bevorzugte Ansatz, um jedes Performance-Problem zu „lösen": *«Die Abfrage läuft langsam, also lass uns einen zusätzlichen Index erstellen».*

Bei ETL-Prozessen hilft das in der Regel nicht, es erhöht sogar die Ladezeiten. Jeder zusätzliche Index verlangsamt die DML-Performance von INSERT-, UPDATE- oder MERGE-Befehlen und – noch schlimmer – kann dazu führen, dass der Optimizer einen *Nested Loops Join* verwendet (siehe Tipp 2). ETL-Jobs arbeiten auf großen Datenmengen, nicht auf einer kleinen Teilmenge der Daten. Ein Index ist für eine selektive Abfrage konzipiert, die nur einen kleinen Prozentsatz der Daten einer Tabelle zurückgibt.

Deshalb ist es empfehlenswert, in einem Data Warehouse so wenig Indizes wie möglich zu verwenden. Sie sind nur für Primärschlüssel und Unique-Constraints notwendig. In Data Marts können Indizes für eine gute Abfrageleistung nützlich sein. Dies ist typischerweise bei Star Schemas der Fall, in denen Bitmap-Indizes auf den Faktentabellen erstellt werden.

Tipp 4: Funktionen in WHERE-Bedingung vermeiden

Es kann mehrere Gründe für falsche Schätzungen des Query Optimizers geben. Eine häufige Ursache sind Ausdrücke oder Funktionsaufrufe in WHERE-Bedingungen. Wenn eine SQL-Funktion wie UPPER, SUBSTR, TO_CHAR usw. in einer Filterbedingung verwendet wird, ist es für den Optimizer viel schwieriger, die Kardinalität zu schätzen, als wenn nur unveränderte Attribute verwendet werden. Natürlich ist es nicht immer möglich, solche Transformationen zu vermeiden. Aber manchmal ist es einfacher und viel schneller, die Transformationen in einem vorherigen Schritt durchzuführen (z.B. in einer "Cleansing Area" oder in einer temporären Tabelle). Dies hilft, Fehleinschätzungen durch den Optimizer zu vermeiden.

Der schlimmste Fall sind PL/SQL-Funktionsaufrufe. Sie sind für den Optimizer wie eine „Blackbox" und reduzieren die Schätzungen in der Regel dramatisch. Nehmen wir eine Abfrage auf einer Tabelle mit 1 Million Zeilen an. Die Query enthält drei PL/SQL-Funktionsaufrufe in der WHERE-Bedingung, kombiniert mit AND-Operatoren. Der Optimizer geht von einer Selektivität von 1% (0.01) für jede Funktion aus. Die Kardinalität für diese Query wäre *0.01 * 0.01 * 0.01 * 1000000*, also 1 Zeile. Bei einer so kleinen Kardinalität ist es sehr wahrscheinlich, dass der Optimizer einen *Nested Loops Join* wählt.

Tipp 5: Aufgepasst mit OR in WHERE-Bedingungen

Kommt in einer WHERE- oder Join-Bedingung das Schlüsselwort OR vor, lohnt es sich, den Ausführungsplan sehr genau anzusehen. Obwohl der Optimizer in der Lage ist, die Selektivität

mehrerer Bedingungen in Kombination mit einem OR abzuschätzen, wird er teilweise durch komplexe WHERE-Bedingungen „verwirrt“. In vielen Fällen können Bedingungen mit OR durch andere Ausdrücke ersetzt werden. Beispielsweise kann der Ausdruck *attribut = 4711 OR attribut IS NULL* durch *NVL(attribut, 4711) = 4711* ersetzt werden. Das ist zwar auch nicht ideal (siehe Tipp 4), aber oft das kleinere Übel.

In einigen Fällen wandelt der Oracle-Optimizer OR-Bedingungen in *Lateral Inline Views* um. Dies kann unter Umständen nützlich sein, kann aber zu suboptimalen Ausführungsplänen führen. Manchmal ist es sogar noch schneller, eine SQL-Anweisung mit einem OR in der WHERE-Bedingung in zwei separate Anweisungen aufzuteilen. Eine UNION dieser beiden Anweisungen läuft viel schneller als die ursprüngliche Anweisung mit OR.

Tipp 6: Frühzeitige Einschränkung der Datenmenge

Je früher die Datenmenge reduziert werden kann, desto weniger Arbeit muss Oracle leisten, um die entsprechenden Zeilen jeder Tabelle zu lesen und zu verknüpfen. Dies ist die wichtigste Regel für das Performance-Tuning in OLTP-Anwendungen mit selektiven Abfragen. Das gleiche Prinzip kann auch helfen, die Laufzeiten eines ETL-Jobs zu verbessern. Nehmen wir an, wir lesen eine große Tabelle, die mit mehreren kleineren Tabellen verbunden werden muss. In diesem Fall ist es schneller, zuerst alle kleinen Tabellen zu verbinden, bevor die Ergebnismenge mit der hohen Zeilenzahl der großen Tabelle „aufgeblasen“ wird.

In den meisten Fällen ist der Optimizer in der Lage, aus der Kardinalität jeder Tabelle, d.h. der Anzahl der geschätzten Zeilen, die richtige Join-Reihenfolge abzuleiten. Mit *Hash Joins*, unserer bevorzugten Join-Methode in ETL-Jobs, wird immer die kleinere Tabelle als treibende Tabelle eines Joins verwendet. Bei SQL-Anweisungen mit komplexen WHERE-Bedingungen kann es jedoch vorkommen, dass der Optimizer die größte Tabelle zu früh liest. Dieses Problem kann mit einem Hint /*+ *leading* */ oder – eleganter – durch Verwendung einer *Subquery Factoring Clause* gelöst werden, wie im nächsten Abschnitt beschrieben (siehe Tipp 7).

Tipp 7: Verwendung von WITH, um komplexe SQL-Befehle aufzuteilen

Die WITH-Klausel oder *Subquery Factoring Clause*, wie sie von Oracle offiziell genannt wird, erlaubt es, eine umfangreiche SELECT-Anweisung in einzelne Teile zu zerlegen. Dies erleichtert das Lesen des SQL-Befehls, ähnlich wie bei lokalen Prozeduren in einem komplexen PL/SQL-Package. Aber WITH-Klauseln haben noch zusätzliche Vorteile. Da jede *Subquery Factoring Clause* auf alle anderen zuvor definierten WITH-Klauseln verweisen kann, ist es sehr einfach, eine komplexe SQL-Anweisung zu debuggen und die Performance der einzelnen Teile zu testen.

Der Optimizer führt eine *Subquery Factoring Clause* entweder als *Inline View* oder als temporäre Tabelle aus. Insbesondere die Optimierung mittels temporärer Tabelle ist für komplexe ETL-Anweisungen sehr nützlich. Sie ermöglicht die Aufteilung eines komplexen ETL-Jobs in einzelne kleinere Teile, die viel schneller ausgeführt werden können.

Anstatt eine Zwischentabelle zu laden, die als Quelle für den nächsten ETL-Job verwendet wird, können diese Schritte mit einer WITH-Klausel in einer SQL-Anweisung zusammengefasst werden. Und wenn der Optimizer nicht erkennt, dass der schnellste Weg zur Ausführung der Abfrage das Laden einer temporären Tabelle ist, können wir dies mit einem Hint /*+ *materialize* */ in der *Subquery Factoring Clause* erzwingen.

Tipp 8: Parallele Ausführung von SQL-Befehlen

Bei großen Datenmengen ist es sehr empfehlenswert, die ETL-Jobs zu parallelisieren. Oracle ist in der Lage, SQL-Anweisungen mit mehreren parallelen Prozessen auszuführen. Die parallele Ausführung ist standardmäßig für Abfragen und DDL-Befehle aktiviert. Für DML-Anweisungen muss sie mit dem Befehl *ALTER SESSION ENABLE PARALLEL DML* aktiviert werden. Zusätzlich muss die Datenbankkonfiguration und unter Umständen der *Degree of Parallelism (DOP)* der Tabellen angepasst werden.

Aber selbst wenn die Konfiguration für die parallele SQL-Ausführung korrekt eingestellt ist, kann nicht garantiert werden, dass jede SQL-Anweisung parallel ausgeführt wird. Der Optimizer entscheidet für jeden SQL-Befehl – basierend auf der Selektivität – ob er parallel oder seriell ausgeführt wird. Bei Bedarf kann diese Entscheidung vom Optimizer mit den Hints /*+ *parallel* */ oder /*+ *no_parallel* */ beeinflusst werden.

Tipp 9: Verwendung von Direct-Path INSERT

Direct-Path INSERT ist eine effiziente Möglichkeit, große Datenmengen in eine Zieltabelle zu laden. Im Gegensatz zum *Conventional INSERT* ist diese Methode wesentlich schneller, da neue Daten immer in zusätzlichen Tabellenblöcken am Ende der Tabelle (hinter der *High Water Mark*) angefügt werden. Dies vermeidet den Aufwand für die Wiederverwendung von freiem Platz in bestehenden Tabellenblöcken und reduziert UNDO- und REDO-Log-Informationen. Indizes werden nach Abschluss des INSERT-Befehls aktualisiert, nicht für jede eingefügte Zeile einzeln. *Direct-Path INSERT* wird explizit mit dem Hint /*+ *append* */ aktiviert oder implizit für parallele DML-Befehle verwendet (siehe Tipp 8). Ein ähnlicher Hint ist /*+ *append_values* */, der verwendet werden kann, um *Direct-Path INSERT* in Bulk-Insert-Operationen zu erzwingen, z.B. in einer FORALL-Anweisung in PL/SQL.

Auch bei MERGE-Anweisungen kann *Direct-Path INSERT* für das INSERT-Teil verwendet werden. Zu beachten ist dabei, dass in diesem Fall der Hint /*+ *append* */ nach dem Schlüsselwort *MERGE* angegeben werden muss, nicht in der Klausel *WHEN NOT MATCHED THEN INSERT*.

Für *Direct-Path INSERT* gibt es mehrere Einschränkungen. Die wichtigste: Wenn die Zieltabelle aktivierte Foreign Key Constraints enthält, funktioniert *Direct-Path INSERT* nicht. In diesem Fall müssen die Constraints während des Ladens deaktiviert werden. Dies ist einer der Gründe, weshalb in Data Warehouses Constraints häufig mittels *RELY DISABLE NOVALIDATE* definiert werden.

Tipp 10: Statistikberechnung nach dem Laden jeder Tabelle

Der letzte Schritt jedes ETL-Jobs sollte immer das Berechnen der Optimizerstatistiken für die Zieltabelle sein. Dies ist besonders wichtig für Tabellen, die von Grund auf neu geladen werden (z.B. Stage-Tabellen oder Zwischentabellen komplexer ETL-Jobs). Fehlende oder falsche Statistiken können zu schlechten Schätzungen für die nächsten ETL-Schritte führen. Wenn ein ETL-Job mehrere Stage-Tabellen mit veralteten Statistiken verbindet, ist das Risiko eines schlechten Ausführungsplans sehr hoch.

Es gibt mehrere Features in Oracle, um dieses Problem zu lösen, z.B. Dynamic Sampling oder Adaptive Plans (in Oracle 12c). Dennoch ist es ratsam, für jeden Schritt eines komplexen ETL-Laufs aktuelle Statistiken zu haben. Damit diese sofort für den nächsten ETL-Schritt verwendet werden,

sollte der DBMS_STATS-Parameter *no_invalidate => FALSE* angegeben oder als *Preference* definiert werden.

Seit Oracle 12c ist es sehr einfach, Tabellenstatistiken für initial geladene Tabellen zu berechnen. Wenn eine Tabelle mittels TRUNCATE gelöscht und dann mit einem *Direct-Path INSERT* geladen wird, oder wenn sie mit *CREATE TABLE AS SELECT* neu erstellt wird, werden automatisch Statistiken berechnet.

Fazit

Natürlich sind diese zehn Performance-Tipps keine vollständige Referenz für das Performance-Tuning von ETL-Jobs. Aber vielleicht helfen sie, die Ladezeiten einiger ETL-Jobs zu reduzieren. Je mehr Ladeprozesse Sie durch Anwendung dieser Hinweise verbessern, desto mehr werden Sie feststellen, dass die Gründe für schlechte Performance oft mit den gleichen Grundprinzipien zusammenhängen.

Weiterführende Informationen

In meinem Blog Data „Warehousing with Oracle“ (https://danischnider.wordpress.com) beschreibe ich gelegentlich Detailaspekte zum Thema ETL-Performance. In folgenden Blogbeiträge finden Sie weiterführende Informationen der hier beschriebenen Tipps:

- 10 Tips to Improve ETL Performance
  https://danischnider.wordpress.com/2017/07/23/10-tips-to-improve-etl-performance/
- Oracle Initialization Parameters for Data Warehouses
  https://danischnider.wordpress.com/2015/04/29/oracle-initialization-parameters-for-data-warehouses/
- Foreign Key Constraints in an Oracle Data Warehouse
  https://danischnider.wordpress.com/2015/12/01/foreign-key-constraints-in-an-oracle-data-warehouse/
- Avoid dbms_stats.auto_invalidate in ETL jobs
  https://danischnider.wordpress.com/2015/01/06/avoid-dbms_stats-auto_invalidate-in-etl-jobs/
- Online Statistics Gathering in Oracle 12c
  https://danischnider.wordpress.com/2015/12/23/online-statistics-gathering-in-oracle-12c/
- Direct-Path INSERT and NOLOGGING with Oracle 12.2
  https://danischnider.wordpress.com/2018/02/20/direct-path-insert-and-nologging-with-oracle-12-2/
- Using Temporary Tables for Complex Reports
  https://danischnider.wordpress.com/2018/05/15/using-temporary-tables-for-complex-reports/
- 10 Tips to Improve ETL Performance – Revised for ADWC
  https://danischnider.wordpress.com/2018/07/20/10-tips-to-improve-etl-performance-revised-for-adwc/

Kontaktadresse:

Dani Schnider
Trivadis AG
Sägereistrasse 29
CH-8152 Glattbrugg

Telefon: +41 58 459 50 81
Fax: +41 58 459 56 95
E-Mail dani.schnider@trivadis.com
Internet: www.trivadis.com