Faktenbasierte Indexierung – ein Erfahrungsbericht

Lothar Flatz, Diso AG

Das Erstellen einer faktenbasierten Indexierung zählt zu den Königsdisziplinen der IT. Der Artikel zeigt, dass diese sich in verschiedener Hinsicht auszahlt.

Während seiner zehnjährigen Tätigkeit für Oracle Consulting war der Autor mit spannenden Aufgaben betraut. Darunter war die Mitarbeit an der Software zur Auswertung der Messdaten beim CERN oder eben die komplette Neuindexierung einer großen Online-Datenbank. Wenn dabei von „Index" die Rede ist, ist damit der konventionelle „B*"-Index gemeint. Für ein Data Warehouse gelten andere Indexierungsregeln, diese würden allerdings den Rahmen dieses Artikels sprengen.

Kaffeepause mit Folgen

Man saß gemütlich beim Kaffee bei einem Stammkunden, der dem Autor einen Tuning-Auftrag erteilt hatte. Der erfahrene DBA meinte: „Ich wette, mindestens 30 Prozent unserer Indizes sind überflüssig. Die Folge: Unsere Ausführungspläne sind instabil und brauchen Platz. Am liebsten würde ich die Indexierung komplett überprüfen lassen."

Fakt ist: Bei eingekauften Applikationen werden die Indizes in der Regel von den Entwicklern erstellt. Diese erleben das System selten unter Volllast. Zudem ist die Indexierung nie an die Bedürfnisse eines bestimmten Kunden angepasst. Es muss also nach-indexiert werden. Die Grundlage, die Entwickler geschaffen haben, wird aber nicht mehr infrage gestellt. Es kommen lediglich neue Indizes dazu. Das Resultat ist ein unübersichtlicher, dichter Dschungel aus Indizes.

Wie wäre es, einmal alles auf Start zu setzen und von Beginn weg richtig zu machen? Die Fakten dazu sind vorhanden. Das Werkzeug auch, schließlich ist die Oracle-Datenbank eine unerschöpfliche Quelle von Informationen. Über die Informationen, die im Shared Pool und in AWRs gespeichert sind, müsste eine geeignete Indexierung erstellt werden können.

Dieser Gedanke ließ den Autor nicht mehr in Ruhe. Einige Monate später kam die Chance: Einer seiner Kunden hatte genug von halbherzigen Lösungen. Er wollte Ordnung in der Index-Struktur seiner wichtigsten Applikationen. Am Anfang brauchte der Autor seine ganze Überredungskraft, um den Kunden davon zu überzeugen, dass die Indexierung von Grund auf neu gemacht werden muss. Denn er war überzeugt, dass man sich bei Verbesserungen auf Basis der bestehenden Indexierung nur verzetteln würde.

Natürlich hätte man einfach ein großes SQL-Tuning-Set erstellen und dieses durch den SQL-Tuning-Advisor abarbeiten lassen können. Man hätte sich dann allerdings blind auf die Entscheidungen des SQL-Tuning-Advisor verlassen müssen und das Wissen und die Erfahrungen der Entwickler und DBAs nicht verwenden können.

In diesem Zusammenhang die Argumente aus der Fachliteratur: „First, while the automated tools reduce the complexity of the physical design process, it is still nontrivial to identify a representative workload that can be used to drive the physical design in its entirety. Second, automated tools do not consider all factors that impact physical design (e.g., the impact of replication architectures). *[1]*." Neue Schweizer Forschungen weisen darauf hin, dass die automatische Lösung mittels Tools für große Gesamtaufgaben problematisch ist *[2, 3]*.

Man hat sich dann entschlossen, ein halbautomatisches Vorgehen zu wählen. So ließ sich jeder Schritt erklären und nachvollziehen. Jedes Teammitglied konnte seine Gedanken einbringen. Am Schluss stand eine klar begründete Lösung, zu der jeder stehen konnte.

Die Vorbereitung

Zunächst wurde ein Team aus allen Beteiligten gebildet. Dieses bestand aus zwei Vertretern aus dem Entwicklungsteam des Applikationsherstellers, zwei DBAs als Vertreter des Softwarebetreibers und dem Autor als externer Berater für den Datenbank-Hersteller. In zwei Sitzungen wurden das grobe Vorgehen bestimmt und die Grundregeln der Indexierung festgelegt:

- Primär- und Fremdschlüssel bekommen grundsätzlich automatisch einen Index
- Namenskonventionen
- Design, das der Applikation Rechnung trägt
- Art der physischen Speicherung (zum Beispiel Tablespace)

Datensammlungsphase

In dieser Phase wurden möglichst viele Informationen über die Abfragebedingungen auf der Datenbank gespeichert. Dabei war es essenziell, möglichst alle wichtigen Verarbeitungen in die Auswertung mit einzubeziehen – also mussten nicht nur tägliche Verarbeitungen, sondern auch wöchentliche und monatliche Aktivitäten Berücksichtigung finden. Diese zeitaufwändige Phase zog sich über mehrere Monate hin. Die Arbeit wurde von automatischen Sammel-Tools geleistet, das Team hatte damit relativ wenig zu tun. Im

Wesentlichen werden folgende Informationen gesammelt:

- Häufigkeit und die Vergleichsoperatoren, mit denen nach einer bestimmten Spalte gesucht wird
- Kombination von Spalten, nach denen gleichzeitig gesucht wird, und die Häufigkeit, mit der dies geschieht

Dies sind im Wesentlichen die gleichen Informationen, die die Datenbank zur Unterstützung der automatischen Statistik-Generierung in der „sys.col_usage$" sammelt. Ab der Version 11.2.0.2 werden auch Spalten-Kombinationen unterstützt. Das entsprechende Verfahren beschreibt Maria Colgan aus dem Oak-Table-Netz und bis vor Kurzem Produktmanagerin des Optimizers in ihrem Blog *[4]*. In früheren Datenbank-Versionen werden die Spalten-Kombinationen nicht unterstützt und müssen daher aus den anderen Quellen hochgerechnet werden. Als Quellen kommen infrage:

- Die aktuellen Abfragen „Shared Pool". Hier kann man die Suchkriterien ganz einfach aus den Filtern und „Access Predicates" entnehmen.
- Die „top"-Statements aus dem WAR. Hier muss man leider einen „reparse" durchführen, da die Filter- die Access Predicates in der „DBA_HIST_SQL_PLAN"-Tabelle leer sind.

Diese Möglichkeiten sind deshalb wichtig, weil man sich auf den Inhalt der „col_usage$" nicht hundertprozentig verlassen kann. Der Autor hat Datenbanken gesehen, bei denen der Inhalt der „col_usage$" nicht brauchbar war, vermutlich aufgrund von Speichermangel. Es ist klar, dass dann auch die Statistikgenerierung in Mitleidenschaft gezogen wird. Außerdem werden Informationen aus dem Dictionary verwendet :

- Die Selektivität der einzelnen Spalten sowie die der verwendeten Kombinationen
- „Primary Key"- und „Foreign Key"-Constraints

Die Auswertung

In dieser Phase werden die gesammelten Informationen zu einem fertigen Index-Design verdichtet. Dies geschieht über mehrere Etappen. Bei schwierigen Design-Entscheidungen wird immer noch das menschliche Wissen als letzte Instanz hinzugezogen.

Das Vorgehen folgt grob dem Muster des als „Merge and Reduction" bekannten Algorithmus *[3]*. Als Grundlage für die Arbeit dienen die in der Vorphase gefundenen Spalten-Kombinationen. Im Grunde genommen könnte man aus jeder Spalten-Kombination, die im „Suchen" auftaucht, einen Index erzeugen, was aber zu einem Überangebot an Indizes führen würde. Da jeder Index die DML-Operationen verlangsamen kann *[6]*, sollen nur so viele Indizes wie nötig und so wenige wie möglich erstellt werden. In den folgenden Schritten wird daher versucht, die Index-Struktur hinsichtlich Preis/Leistung zu optimieren.

Grund-Indexierung

Meist wird man von einer Basis-Indexierung ausgehen. Primary Keys erhalten in der Regel ohne große Überlegung einen Index. Foreign Keys sollte man im Allgemeinen auch indexieren, und zwar aus folgenden Gründen:

- Da es beim Löschen eines Satzes zu unangenehmen Sperren der abhängigen Sätze kommen kann, sofern ein „Foreign Key"-Constraint existiert *[5]*
- Damit ein „nested loop"-Join in jede bestimmte Richtung möglich ist und somit der Optimizer in der Wahl des besten Zugriffsplans nicht unnötig eingeschränkt wird

Elimination

In dieser Phase nimmt man alle Suchkombinationen, die man ohne große Leistungseinbußen weglassen kann, aus der Betrachtung. Das sind insbesondere:

- Foreign- und Primary-Keys, die bereits im vorigen Schritt indexiert worden sind
- Suchkombinationen, die so wenig selektiv sind, dass sich kein Index lohnt
- Suchkombinationen, die bereits in anderen Suchkombinationen gleichwertig enthalten sind

Dazu einige Beispiele aus dem allgemein bekannten Bereich der Adress- und Personendaten: Ein Index „{Geschlecht}" lohnt sich auf der Tabelle „Person" nicht, da die Suchspalte zu wenig selektiv ist. Ausnahme: Eine Anwendung für Frauenfragen beim Militär, da hier häufig nach einem seltenen Wert gesucht wird. Dieses Beispiel zeigt, dass selbst scheinbar einfache Fälle nicht ohne Nachdenken entschieden werden können. Einmal mehr wird klar, vor welchen Schwierigkeiten eine vollautomatische Indexierung steht. Ein anderes Beispiel: Der Indexkandidat „{Ort}" auf der Tabelle „Adresse" wird eliminiert, wenn es eine andere häufige Suchkombination gibt, zum Beispiel „{Ort, Straße}", in der „{Ort}" bereits enthalten ist. Dies kann man natürlich noch weiter führen: Der dreispaltige Index „{Nachname, Vorname, Alter}" auf der Tabelle „Person" kann drei verschiedene Suchkriterien unterstützen:

- {Nachname, Vorname, Alter}
- {Nachname, Vorname}
- {Nachname}

Die entsprechenden Suchkombinationen können also eliminiert werden. Folgende Suchabfragen hingegen können nicht unterstützt werden und müssen bestehen bleiben:

- {Vorname, Alter}
- {Alter}
- {Vorname}

An diesen Beispielen erkennt man, dass die Wahl der richtigen Reihenfolge der Spalten entscheidend für die Qualität eines flexiblen Index-Designs ist. Eine optimale Indexierung, unabhängig von der Spalten-Reihenfolge, ist mit dem konventionellen „B*"-Index nicht möglich. Wer die ultimative Flexibilität und Performance haben möchte, muss den mehrdimensionalen Index, beispielsweise den der Firma „dimensio", verwenden. Er kann eine beliebige Kombinationen von Suchkriterien parallel auswerten und verknüpfen. Dies erleichtert die optimale Indexierung wesentlich.

Die Synthese

In diesem Schritt versucht man, übrig gebliebene Suchkombinationen zusammenzulegen und dadurch weitere Index-Kandidaten zu eliminieren. So kann man beispielsweise einen „Foreign Key" um

weitere Suchkriterien erweitern. Wenn zum Beispiel in der Tabelle „Umsatz“ häufig nach „{Artikelnummer, Verkaufsdatum}“ gesucht wird, ist es sinnvoll, einen bestehenden „Foreign Key“-Index auf der Artikelnummer um das Feld „Verkaufsdatum“ zu erweitern.

Manchmal kann man auch durch eine leichte Verschlechterung des Index-Designs einen zusätzlichen Index-Kandidaten eliminieren, zum Beispiel bei den Suchkombinationen „{Ort, Nachname, Vorname}“ und „{Ort, Nachname, Geburtsdatum}“. Der Indexkandidat „{Ort, Nachname, Vorname, Geburtsdatum}“ kann die beiden Kombinationen ersetzen. Zwar ist die Suche nach „{Ort, Nachname, Geburtsdatum}“ nicht mehr so optimal unterstützt wie beim spezialisierten Index, jedoch kann die Verschlechterung wahrscheinlich in Kauf genommen werden, weil „Ort“ und „Nachname“ für sich gesehen bereits gute Suchkriterien sind. Man sieht, dass in diesem Schritt die Vernunftgesteuerte Entscheidung des menschlichen Designers besonders wichtig ist.

Tests und Umsetzung

Hat man alle diese Schritte durchlaufen, ist es relativ einfach, aus dem Ergebnis ein „Index create“-Skript zu bilden. Natürlich ist das Ergebnis jetzt ausführlich zu testen. „RAT“ bietet sich dabei als Mittel der Wahl an. Wenn man produktiv geht, sollte man auch Überwachungsmechanismen im Einsatz haben, die schnell in der Lage sind, fehlende Indizes zu finden.

Der Autor hatte ein eigenes Überwachungsskript, das Ausführungspläne finden kann, die durch suboptimale Indexierung entstehen, und entsprechende Verbesserungsvorschläge macht. Natürlich kann man auch den „Oracle Index Advisor“ verwenden, der für den Autor allerdings für diesen spezialisierten Zweck etwas umständlicher in der Handhabung ist. Die neue Index-Struktur zeigte sich in der Produktion erstaunlich stabil. In den ersten Monaten musste nur eine einstellige Anzahl von Indizes nacherzeugt werden.

Fazit

In dem hier geschilderten Fall handelt es sich um eine Applikation, die spezialisierte Consultants der Firma Oracle schon seit Monaten optimiert hatten. Umso erfreulicher war es, dass die Gesamt-Indexierung zusätzlich eine Platzersparnis von 30 Prozent und eine Performance-Verbesserung von ebenfalls 30 Prozent erbrachte.

Ein sehr willkommener Nebeneffekt kam überraschend: die verbesserte Stabilität der Execution-Pläne. Da für einen bestimmten Zweck nur noch ein Index zur Verfügung steht und nicht mehrere, bleiben die Pläne stabiler und verändern sich nicht so leicht in Abhängigkeit von den Werten der Bind-Variablen.

Last but not least: Die neue Indexstruktur mit ihren klaren Regeln und ihrer einheitlichen Namensgebung erleichtert die tägliche Arbeit der DBAs.

Weitere Informationen

[1] Weikum, Moenkeberg, Hasse, Zabback, „Self-tuning Database Technology and Information Services: from Wishful Thinking to Viable Engineering“, Proceedings of the 28th VLDB Conference, Hong Kong, China, 2002

[2] Bruno, N. and Chaudhuri, S. 2007. Physical design refinement: The ‘merge-reduce’ approach. ACM Trans. Database Syst. 32, 4 (Nov. 2007), 28.

[3] Borovica, Alagiannis, Ailamaki. „Automated Physical Designers: What You See in (Not) What You Get“, DBTest’12, May 21, 2012 Scottsdale, AZ, USA.

[4] Colgan M., How do I know what extended statistics are needed for a given workload? blogentry: https://blogs.oracle.com/optimizer/entry/how_do_i_know_what_extended_statistics_are_needed_for_a_given_workload.

[5] Oracle® Database Concepts 12c Release 1 (12.1), Kapitel 9, Locks and Foreign Keys und Kapitel 6, Indexes and Foreign Keys)

[6] Oracle Database 2 Day DBA 12c Release 1 (12.1), Kapitel 8. Managing Indexes

Lothar Flatz
lflatz@diso.ch