Der Optimizer

Klaus Reimers, ORDIX AG

Der Artikel zeigt anhand von Beispielen die Grundzüge der SQL-Optimierung auf und behandelt Parsing, Statistiken („dbms_stats"), Initialisierungsparameter und die Fixierung von Ausführungsplänen so, dass auch Einsteiger dem Thema gut folgen können. Etwas tiefer wird der für viele undurchsichtige Bereich „MBRC" beschrieben.

Der Optimizer ist eine der zentralen Komponenten des gesamten Oracle-Systems. Er entscheidet, in welcher Form ein SQL-Statement intern abgearbeitet wird. Sowohl der Administrator als auch der Entwickler haben einige Möglichkeiten, auf den letztendlich entstehenden Ausführungsplan einzuwirken.

Der Anwender setzt im Client ein SQL-Statement ab, das dann in der SQL-Area abgelegt wird. Die SQL-Area ist dabei eine Komponente im Library Cache des Shared Pool innerhalb der SGA. Das SQL wird geparst, der Ausführungsplan wird entweder erstellt oder als Konserve aus der Datenbank genommen. Anschließend wird das Statement ausgeführt.

Das Parsen

Immer wenn ein SQL-Statement formuliert wird, muss es zunächst geparst werden. Das Parsen besteht immer aus drei Teilschritten:

- Syntaxcheck
- Semantikcheck
- Erstellung des Ausführungsplans

Zunächst erfolgt die Prüfung auf Korrektheit der Anweisung. Im semantischen Check wird das SQL-Statement dann in seine Einzelteile zerlegt. Hier fragt Oracle im Data Dictionary nach, ob die in der Anweisung angesprochenen Tabellen und Spalten real verfügbar sind. Diese Nachfrage erfolgt natürlich auf intern gebildete SQL-Abfragen, den sogenannten „rekursiven SQLs".

Die Hauptarbeit beim Parsen ist die Erstellung des Ausführungsplans. Hier errechnet der Optimizer den vermeintlich optimalen Ausführungsplan. Dazu wer-

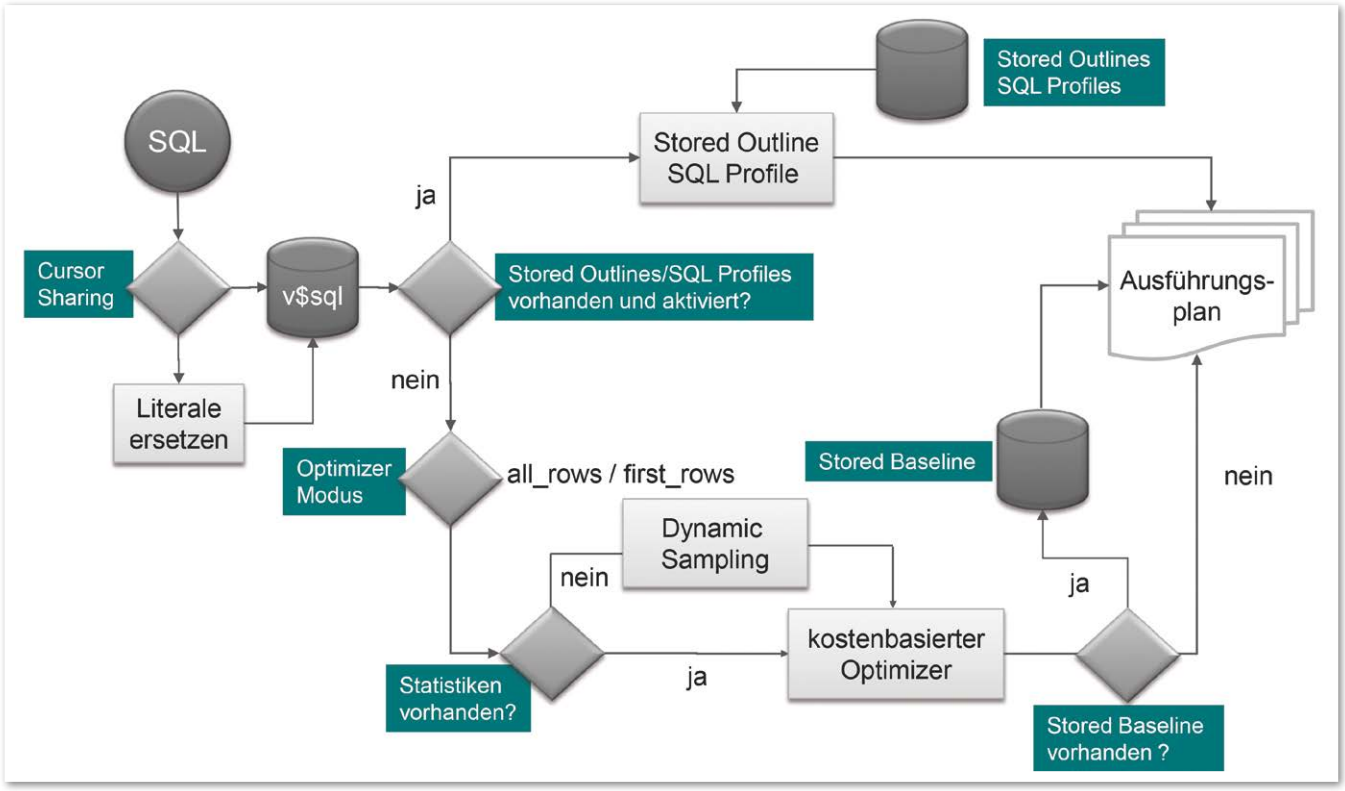

Abbildung 1: Erstellung von Ausführungsplänen - vom SQL zum Ausführungsplan

Statement in der SQL-Area vorliegt. Verständlicherweise ist ein Hard Parse viel aufwändiger als ein Soft Parse. Daher sollten möglichst viele Soft Parsings erfolgen. Dies erreicht man natürlich nur, wenn viele gleiche Statements formuliert werden. *Listing 1* verdeutlicht, dass diese Forderung sicher nicht bei Verwendung von Literalen, sondern nur von Bind-Variablen möglich ist: Daher sollten möglichst viele SQLs mit Bind-Variablen geschrieben werden. Ist dies nicht möglich oder nicht gemacht worden, so kann der Administrator immer noch das sogenannte „Cursor Sharing" aktivieren. Hierdurch werden alle Literale vor dem eigentlichen Parsen in Bind-Variablen verwandelt. Dieses Verfahren kann allerdings einige Nachteile (vor allem bei der Verwendung von Histogrammen) nach sich ziehen.

```
SELECT * FROM MITARBEITER WHERE MITARBEITERNR = 4711;
SELECT * FROM MITARBEITER WHERE MITARBEITERNR = 4712;
SELECT * FROM MITARBEITER WHERE MITARBEITERNR = :manr;
```

Listing 1

Der Optimizer

Der Optimizer generiert den eigentlichen Ausführungsplan. In früheren Versionen von Oracle hat man zwischen dem regelbasierten und dem statistischen Optimizer unterschieden. Der regelbasierte Optimizer (rule based) wird seit Version 10g nicht mehr unterstützt, kann aber nach wie vor intern (vor allem über Hints) verwendet werden. Der Entwickler hatte bei dieser Variante einen starken Einfluss auf die Güte des Plans. Dieser war stabil, da die Generierung auf Basis eines Regelwerks vollzogen wurde und bei gleichem SQL immer der gleiche Plan herauskam.

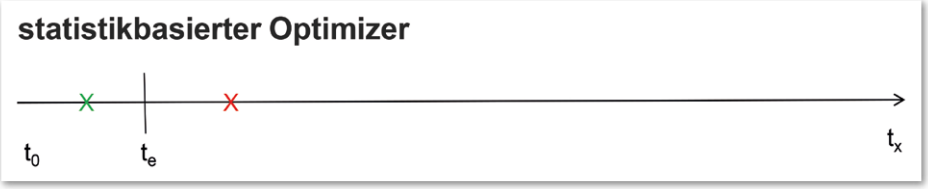

Abbildung 2: Statistikbasierter Optimizer – die Bedenkzeit des Optimizer

Der statistische Optimizer ist heute das Mittel der Wahl. Der Ausführungsplan wird auf Basis zweier Hauptkomponenten erstellt (Statistiken und Parameter). Dieser Optimizer kann in verschiedenen Varianten genutzt werden. Grundsätzlich unterscheidet man zwischen den Einstellungen „FIRST_ROWS" und „ALL_ROWS".

„ALL_ROWS" ist der Standard und optimiert auf eine möglichst schnelle Ausgabe der gesamten Ergebnismenge, während „FIRST_ROWS" einen Plan für eine möglichst schnelle Ausgabe der ersten Sätze erstellt. Grundsätzlich durchdenkt der Optimizer alle theoretisch möglichen Ausführungspläne und entscheidet sich dann für den vermeintlich besten.

Abbildung 2 stellt schematisch den „Denkprozess" des Optimizer dar. Alle theoretisch möglichen Ausführungspläne müssen zwischen „t0" und „tx" evaluiert sein. Jeder Ausführungsplan wird bewertet, das Ergebnis ist eine Zahl (die Kosten). Der Plan mit den niedrigsten Kosten wird verwendet. Es wird also angenommen, dass der günstigste Plan auch der beste ist (Discounter-Prinzip).

Je komplexer ein SQL-Statement ist, desto mehr theoretische Ausführungspläne kann es geben. Da die Parsing-Phase dadurch sehr lange laufen kann, wird nach einer kurzen Zeit („te") abgebrochen und der bis dahin beste (günstigste) Ausführungsplan verwendet. Grundsätzlich kann man sagen: Je komplexer ein SQL formuliert ist, desto höher wird die Gefahr schlechter oder kippender Ausführungspläne, da „te" fix ist. *Abbildung 3* zeigt an einem trivialen Beispiel eines Join über zwei Tabellen, wie viele Ausführungspläne möglich sein können.

statistikbasierter Optimizer (Denkprozess)

SELECT X.b, Y.b FROM X, Y WHERE X.a = Y.a;

- Annahme
  - beide Tabellen haben jeweils ein Index auf der Join-Spalte

Möglichkeit (m)	Anzahl (a)	Gesamt (m*a)
Reihenfolge	2	2
FTS vs Index	4	8
Methode	3	24

- Möglichkeiten bei einem Join über 10 Tabellen?

Abbildung 3: Möglichkeiten bei einem Join über zwei Tabellen

Statistiken

Das PL/SQL-Package „DBMS_STATS" generiert und verwaltet Statistiken für den kostenbasierenden Optimizer. Diese können sowohl im Data Dictionary als auch im Benutzerschema gespeichert sein. Mit diesem Paket können unter anderem folgende Funktionen ausgeführt werden:

- Parallele Generierung von Statistiken
- Erstellung individueller Statistiken
- Erstellen und Löschen von benutzerdefinierten Statistiktabellen
- Austausch von Statistiken zwischen dem Benutzerschema und dem Data Dictionary
- Austausch von Statistiken zwischen Datenbanken

- Sperren von Statistiken
- Wiederherstellung von Statistiken
- Parametrierung der Sammlung von Statistiken

Entscheidend ist hier die konzeptionelle Frage, wann und in welcher Tiefe die Statistiken aktualisiert werden sollen. Grundsätzlich findet man bei den Administratoren zwei verschiedene Vorgehensweisen zum Sammeln von Statistiken. Da neue Statistiken immer die Gefahr von kippenden Ausführungsplänen mit sich bringen, werden auf einigen Datenbanken keine neuen Statistiken erzeugt. Die Pläne sind dann stabil.

Allerdings birgt diese Methode die Gefahr der schleichenden Verschlechterung, da auf veränderte Bedingungen nicht reagiert werden kann. Die meisten Administratoren sammeln daher regelmäßig. Seit Version 9i bietet Oracle hier ein Verfahren an, in dem diejenigen Tabellen mit neuen Statistiken versorgt werden, bei denen mehr als 10 Prozent der Sätze verändert worden sind. Ab 10*g* ist dafür ein automatisierter Task hinterlegt *(siehe Abbildung 4)*.

Seit Version 11*g* kann man individuell je Tabelle die Anforderung an die Art der Statistikerstellung festlegen. Zu diesem Zweck wurden die sogenannten „Preferences" eingeführt:

- *cascade*
  Definiert, ob bei einer neuen Tabellenstatistik auch die an der Tabelle hängenden Indizes mit gepflegt werden sollen
- *degree*
  Grad der Parallelität
- *estimate_percent*
  Definition der Stichprobe in Prozent
- *method_opt*
  Umgang mit Histogrammen
- *no_invalidate*
  Definiert, ob offene Cursor invalidiert werden sollen
- *granularity*
  Umgang mit Partitionen
- *publish*
  Zeitpunkt der Veröffentlichung
- *incremental*
  Globale Statistiken auf partitionierten Tabellen
- *stale_percent*
  Definiert den Grad, wann eine Tabelle als „STALE" definiert wird

Jeder Administrator sollte für jede Datenbank ein klares Konzept haben, wie und in welchen Intervallen Statistiken erzeugt werden.

Parameter

Die Parameterdatei („init.ora"/„spfile") wird grundsätzlich beim Starten der Instanz eingelesen. In der Version 12*c* gibt es 366 offizielle Parameter, die fast alle modifizierbar sind. Allerdings ist bei 115 Parametern die Instanz neu durchzustarten. Etwa 25 bis 30 Parameter haben einen Einfluss auf den „Denkprozess" des Optimizer, *Tabelle 1* zeigt die wichtigsten. Anpassungen an diesen Parametern sollten nicht spontan vorgenommen werden, eine Veränderung muss jeweils gut getestet sein.

NAME	VALUE
optimizer_features_enable	12.1.0.1
optimizer_mode	ALL_ROWS
optimizer_index_cost_adj	100
optimizer_index_caching	0
optimizer_dynamic_sampling	2
optimizer_secure_view_merging	TRUE
optimizer_use_pending_statistics	FALSE
optimizer_capture_sql_plan_baselines	FALSE
optimizer_use_sql_plan_baselines	TRUE
optimizer_use_invisible_indexes	FALSE

Tabelle 1

MBRC

Bei vielen Administratoren gibt es eine Unsicherheit, inwiefern der Parameter „db_

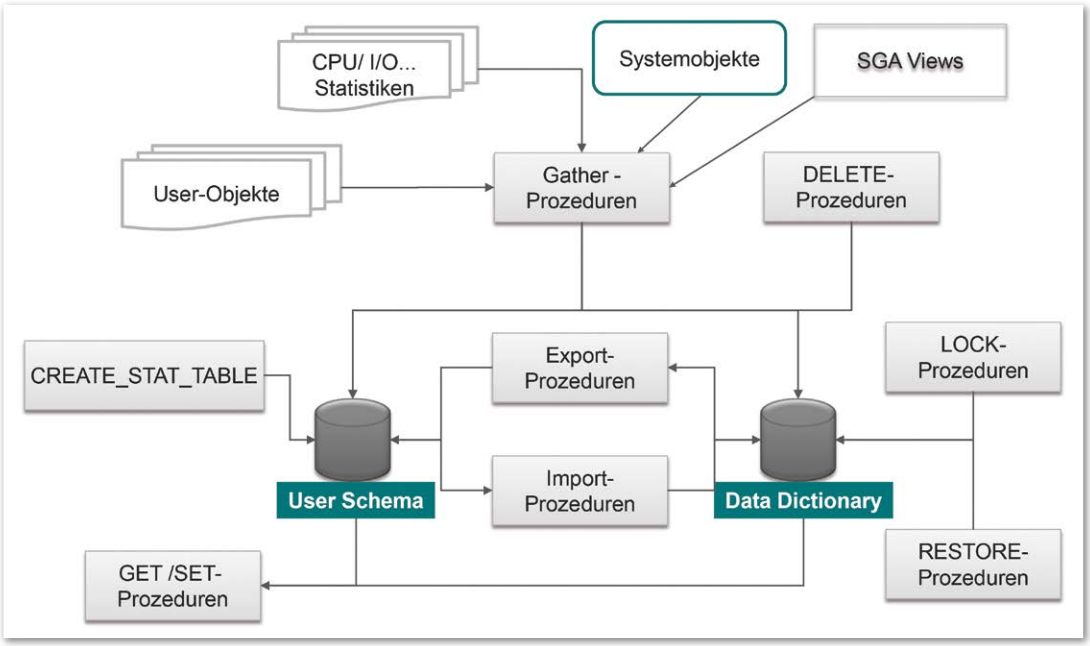

Abbildung 4: Überblick „dbms_stats"

file_multiblock_read_count“ noch wirkt, seit es in den Systemstatistiken den Parameter „MBRC“ gibt. Häufig höre ich die Meinung, dass der Parameter aus der Datei „spfile“ („db_file_multiblock_read_count“) obsolet ist und lediglich der interne Wert aus den Systemstatistiken greift. Hier liegt ein klares Missverständnis vor. Anhand einer kleinen Testreihe kann man die Wirkungsweise verdeutlichen. Grundsätzlich gilt: Je höher „MBRC“ gesetzt ist, desto günstiger wird ein Full Table Scan und desto weniger werden Indizes genutzt *(siehe Abbildung 5)*. Die Testreihe zeigt, dass der Parameter „MBRC“ aus den Systemstatistiken als Input für den Optimizer dient, während der Wert aus „db_file_multiblock_read_count“ bei der realen Durchführung zum Einsatz kommt.

dbfmrc	mbrc	Kosten	Durchführung
1		33924	1
64		6256	64
1	32	6696	1
64	32	6696	64

- Systemstatistiken (mbrc)
  - Kalkulation des Optimizers
- Parameter (db_file_multiblock_read_count)
  - Kalkulation des Optimizers (falls keine Systemstatistiken vorhanden)
  - reale Durchführung

Abbildung 5: Testreihe „MBRC“

Gespeicherte Ausführungspläne

Ist ein SQL-Statement sehr komplex oder ein Ausführungsplan nicht stabil, kann dieser in der Datenbank hart verdrahtet werden. Dafür bietet Oracle drei Möglichkeiten: Stored Outlines, SQL Profiles und SQL Baselines. Diese Varianten sind nicht in allen Oracle-Versionen verfügbar und unterstützt. Zudem ist die Nutzung teilweise kostenpflichtig.

Stored Outlines

Mithilfe der Stored Outlines hat man die Möglichkeit, Ausführungspläne in der Datenbank fest zu speichern. Damit ist gewährleistet, dass man unabhängig vom momentanen Stand der Statistiken immer mit dem gleichen Ausführungsplan arbeitet. Alle Outlines gehören zudem dem User „OUTLN“. Mit Version 10*g* sind die Outlines offiziell aus der Wartung genommen worden – aber intern noch nutzbar.

SQL Profile

Der Optimizer durchdenkt alle theoretisch möglichen Ausführungspläne und errechnet auf Basis der Input-Daten (Parameter und Statistiken) den vermeintlich optimalen Plan. Ist ein SQL nun sehr komplex, so geht die Anzahl der möglichen Ausführungspläne gegen unendlich. Damit die Parsing-Phase nicht unnötig lang wird, bricht der Optimizer den „Denkprozess“ nach einigen Millisekunden ab und es wird der bis zum Abbruch günstigste Plan verwendet. Dieser kann unter Umständen nicht der optimale Plan sein. Im erweiterten Modus (Tuning Mode) durchläuft der Optimizer folgende Tests:

- Statistik-Analyse
- SQL Profiling
- Access-Path-Analyse
- SQL-Structure-Analyse

Der Tuning-Optimizer prüft zunächst, ob Statistiken vorhanden und ob diese Statistiken auch aktuell sind. Stehen keine aktuellen Statistiken zur Verfügung, sammelt der Optimizer diese während der Optimierung. Dies kann dann natürlich länger dauern.

Beim Profiling führt der Optimizer zahlreiche Tests durch. Hierzu gehören beispielsweise die Analyse verschiedener Optimizer-Modi („all_rows“ vs. „first_rows“) sowie das Sampling von Daten. Weiterhin vergleicht der Optimizer unterschiedliche Ausführungspläne bezüglich der Kosten. Akzeptiert der Benutzer das vorgeschlagene Profil, so wird dieses in der Datenbank gespeichert und für spätere Ausführungen genutzt.

SQL Baselines

Bei Verwendung dieser Methode werden die Ausführungspläne in gute und schlechte Pläne unterteilt. Es wird somit garantiert, dass keine als schlecht markierten Pläne mehr ausgeführt werden. Darüber hinaus können Baselines als Fixum erstellt werden. Neue Ausführungspläne werden dann immer an dieser fixierten Basis gemessen und bewertet.

Fazit

Der Optimizer ist eine sehr komplexe Komponente, auf die ein DBA über viele Stellschrauben einwirken kann. Grundsätzlich sollten folgende Verhaltensregeln eingehalten werden:

- Es muss ein Konzept zur Statistikerzeugung vorliegen
- Statistiken müssen aktuell gehalten werden
- Parameter stabilisieren
- Anpassungen intensiv vor Veröffentlichung testen
- Statements nicht zu komplex formulieren

Zum Schluss soll noch eine Lanze für den Optimizer gebrochen werden. Es ist immer wieder erstaunlich, was ihm so alles vor die Füße geworfen wird und wie häufig er daraus einen guten Plan generiert. Die wenigen Fälle, in denen er daneben greift, führen dann aber zu Unmut bei den Anwendern. Der Optimizer ist von Version zu Version stabiler geworden, einzelne Bugs kommen jedoch immer mal vor.

Klaus Reimers
info@ordix.de