Inhalt

- SQL-Optimierung – Grundlagen
- Logische und physische Optimierung
- CBO – Funktionsweise
- Entscheidungsgrundlagen
  - Zugriffspfade
  - Statistiken für den Optimizer
  - Initialisierungsparameter
  - Optimizer Hints
  - Dynamic Sampling
  - Histogramme
  - Planstabilität – Stored Outlines

SQL-Optimierung – Grundlagen

Bestimmungsstücke

- Prinzipien der Plangenerierung
- Kostenfunktionen der verschiedenen Basisoperationen
- Transformationsregeln für SQL-Anweisungen
- Kostenmodell (plattformabhängig!)
- Informationen über die Daten (Statistiken etc.)
- Suchstrategie nach dem effizientesten Plan

Ziel: Abfragen so schnell wie möglich

Prinzipien: Pro Schritt möglichst wenig Tupel

- Selektion so früh wie möglich
- Zusammenfassung von Basisoperationen
- Möglichst wenig Zwischenergebnisse speichern (Pipelining)
- Keine irrelevanten (redundanten) Berechnungen ausführen

Logische Optimierung

Übersetzung der SQL in einen unoptimierten Plan, dabei:

- Sichtexpansion
- Standardisierung (z.B. in Konjunktive Normalform bringen)
- Vereinfachung (Propagierung von Konstanten, Ermittlung unerfüllbarer Prädikate, Idempotenzen)
- Entschachtelung (Auflösung von Unterabfragen)

Algebraische Optimierung

- Entfernen redundanter Operationen
- Verbundreihenfolge (logisch, ohne Kostenabschätzung)
- Vorgruppierungen
- Verschiebung von Selektion und Projektion möglichst weit in Richtung der Blätter des Plans

Physische Optimierung

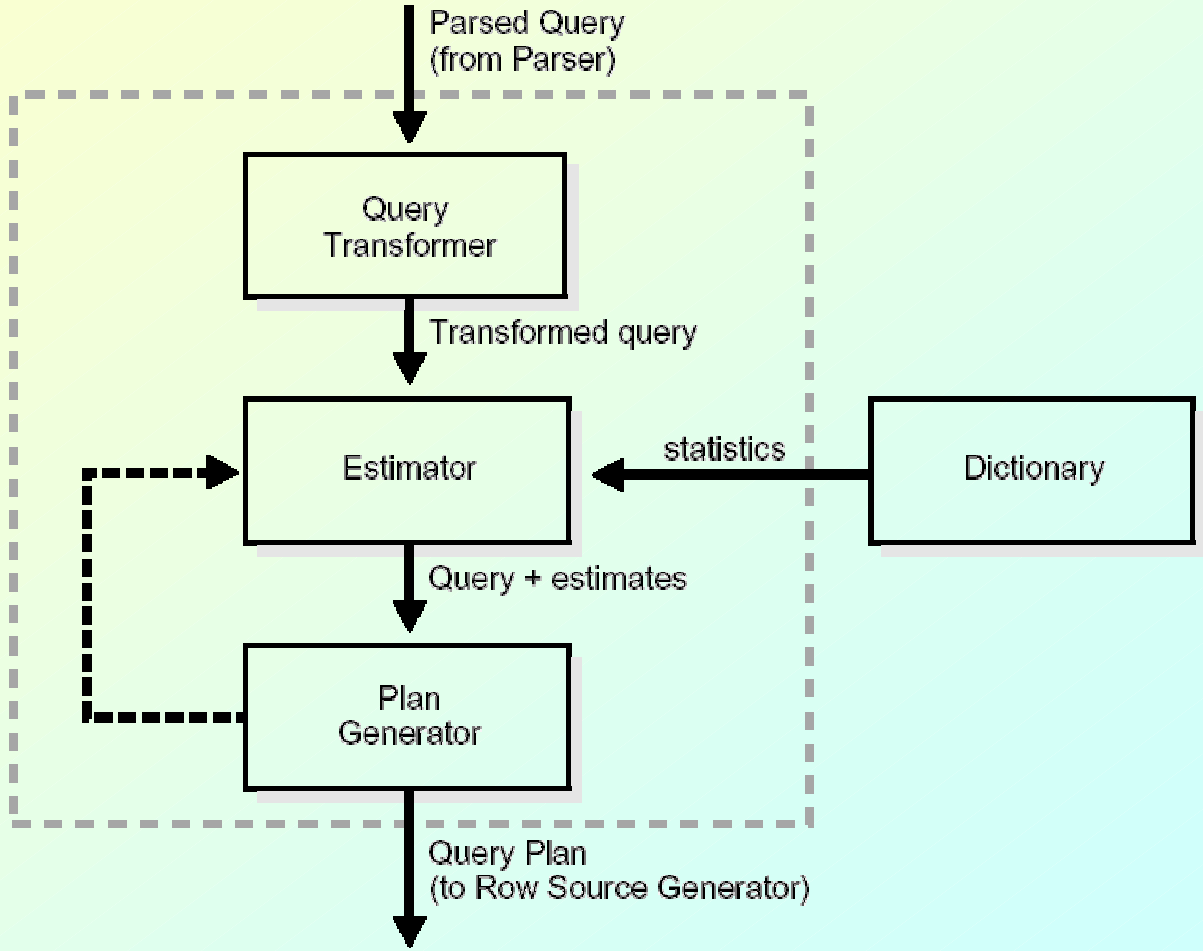
Kostenbasierte Auswahl aus einer Menge von Plänen

- Anwendung der Berechnungsalgorithmen (Transformationsregeln + Kostenfunktionen)
- Einbeziehung von Statistiken über die Daten und ihre Verteilung
- Einsatz eines gewichteten Kostenmodells
- Verwendung von Suchstrategien
- Vermeidung teurer Operationen

Probleme

- Gewicht der Kosten u.U. plattformabhängig
- Deterministische Suchstrategien führen u.U zu aufwendiger Optimierung (Parse-Zeit)
- Heuristische Suchstrategien finden eventuell nicht den optimalen Plan (Nebenminima)

CBO – Funktionsweise



Kostenbasierter Optimizer

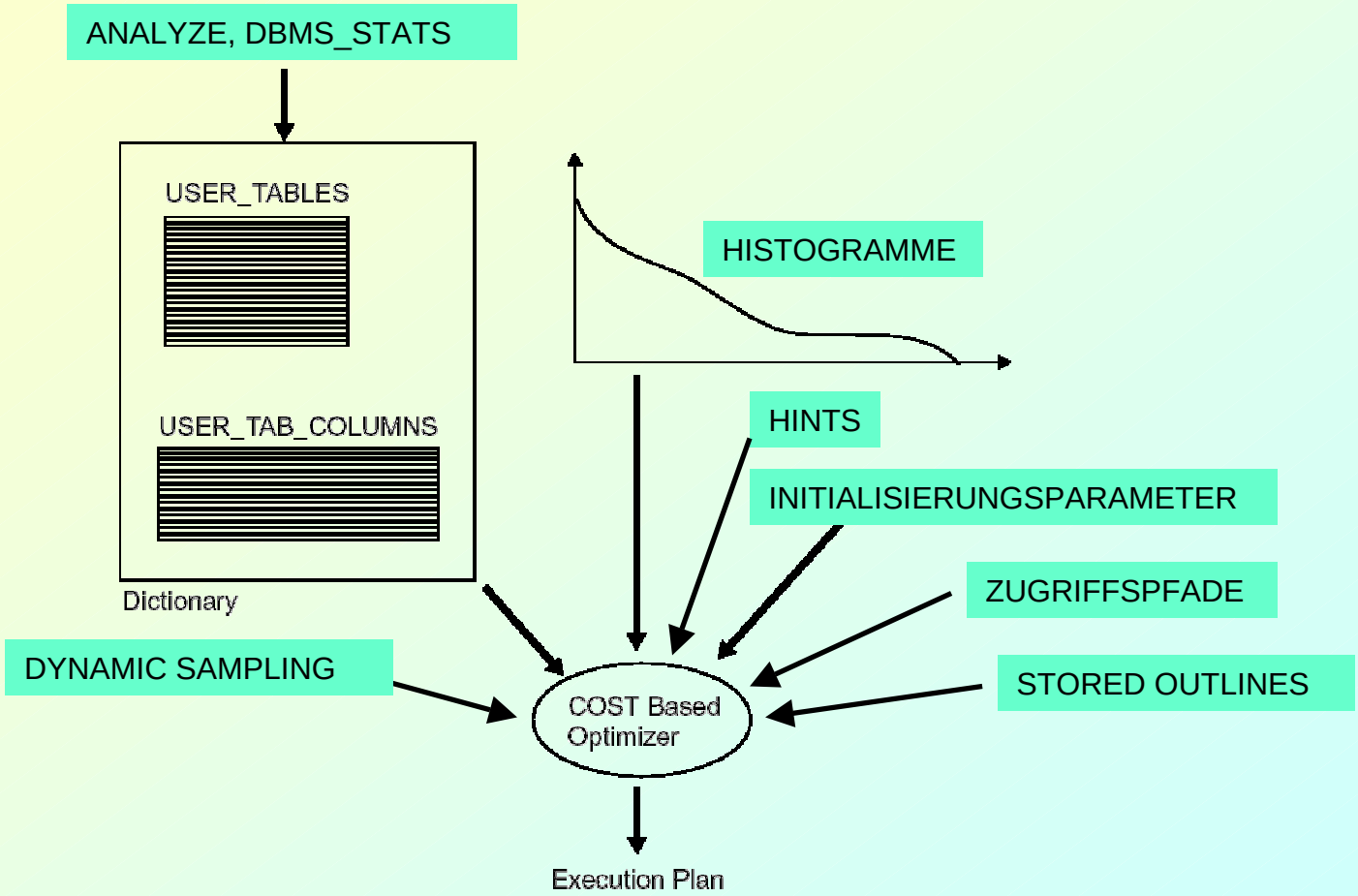
Der CBO

- Anweisungstransformation (z.B. Unterabfragen in Joins)
- Auswahl der Join-Reihenfolge
- Auswahl der Join-Methode
- erzeugt auf der Grundlage der möglichen Zugriffspfade eine Menge von Ausführungsplänen
- schätzt die Selektivität, Kardinalität und Kosten (mutmaßlicher Ressourcenverbrauch) jedes Plans auf der Grundlage von Statistiken. Kostenfaktoren:
  - Anzahl der physischen Lesevorgänge (wichtigster Faktor)
  - CPU-Nutzung
  - Netzwerkverkehr
- vergleicht die Kosten und wählt den günstigsten Plan.

Wichtige Abfragetransformationen

- **IN-Liste**: mehrfaches OR mit Gleichheitszeichen
- **Unterabfragen**: Join
- **LIKE ohne Wildcard**: Gleichheitszeichen
- **a > ANY (b, c)**: a > b OR a > c
- **a > ANY (select b from c)**: EXISTS (select b from c where a > b)
- **a > ALL (b, c)**: a > b AND a > c
- **a BETWEEN b AND c**: a >= b AND a <= c
- **NOT a = (unterabfrage)**: a <> (unterabfrage)
- **Verschiedene Bedingungen mit OR**: UNION ALL
- **DISTINCT, GROUP BY, JOIN**: implizite Sortierung
- **Views:** View Merging oder Predicate Pushing
- **a θ Konstante AND a=b (Transitivität)**: b θ Konstante

Entscheidungsgrundlagen für den CBO



Wichtige Zugriffspfade

- **Full Table Scan**: Alle Blöcke unter der HWM werden geprüft, ob sie Daten enthalten, die der WHERE-Klausel entsprechen
- **Rowid Scan**: Die Rowid enthält Datendatei, Block und Stelle im Block, an der die Zeile zu finden ist.
- **Index Scan**: Liefert die Rowid der gesuchten Zeilen durch
  - **Index Unique Scan**: Rückgabe einer Zeile (=)
  - **Index Range Scan**: Rückgabe eines Zeilenbereichs (<,>,BETWEEN, LIKE, ORDER BY)
  - **Index Skip Scan**: Zusammengesetzter Index wird logisch geteilt. Führende Spalte braucht nicht referenziert zu werden. (Neu in Oracle 9i)
  - **Fast Full Index Scan**: Alle Spalten des Index werden referenziert, kein Tabellenzugriff nötig.
  - **Index Join**: Wenn alle in der Abfrage referenzierten Spalten indiziert sind, ist kein Tabellenzugriff nötig.

Statistiken für den CBO sammeln

ANALYZE: Berechnet oder schätzt Statistik, relativ schlecht für die Performance, wird eventuell in der Zukunft vom CBO nicht mehr unterstützt. (Für Klausel VALIDATE STRUCTURE aber notwendig.)

Syntax:

ANALYZE {INDEX|TABLE|CLUSTER} name {COMPUTE|ESTIMATE|DELETE} STATISTICS [FOR {TABLE|COLUMNS (col_name, ...)|ALL COLUMNS|ALL INDEXED COLUMNS|ALL INDEXES} [SIZE n]] [SAMPLE n {ROWS|PERCENT}];

Views: USER_TABLES, USER_TAB_COLUMNS, USER_TAB_COL_STATISTICS, USER_INDEXES, DBA_HISTOGRAMS

DBMS_STATS: Von Oracle empfohlen, besser für die Performance, Statistiken können exportiert und importiert werden. (Stabilität des Ausführungsplans)

Syntax:

DBMS_STATS.GATHER_DATABASE_STATS()
DBMS_STATS.EXPORT_SCHEMA_STATS()

Anmerkung: Ab Oracle 9i verwendet der CBO auch Systemstatistiken!

Was wird gesammelt?

Tabellen:

- Zeilenzahl
- Anzahl der Blöcke
- Durchschnittliche Zeilenlänge
- Migration und Verkettung

Spalten:

- Anzahl unterschiedlicher Werte (Kardinalität)
- Anzahl der NULL-Werte
- Werteverteilung (Histogramme)

Indizes:

- Anzahl der Blätter (Leaf Blocks)
- Zahl der Ebenen

System:

- I/O-Performance
- CPU-Verwendung

Initialisierungsparameter

Folgende Parameter beeinflussen die Arbeit des CBO (Auswahl):

- ALWAYS_ANTI _JOIN
- ALWAYS_SEMI_JOIN
- CURSOR_SHARING
- DB _FILE_MULTIBLOCK _READ _COUNT
- HASH_AREA _SIZE
- HASH_JOIN _ENABLED
- OPTIMIZER_DYNAMIC_SAMPLING
- OPTIMIZER_FEATURES _ENABLED
- OPTIMIZER_INDEX _CACHING
- OPTIMIZER_INDEX _COST _ADJ
- OPTIMIZER_MAX _PERMUTATIONS
- OPTIMIZER_SEARCH _LIMIT
- SORT_AREA_SIZE
- STAR_TRANSFORMATION_ENABLED
- STATISTICS_LEVEL

Hints für den CBO

Hints

- spezifizieren Ziele für den CBO — FIRST_ROWS
- definieren Zugriffspfade — INDEX
- legen die Join-Reihenfolge fest — ORDERED
- spezifizieren eine Join-Methode — USE_NL
- steuern die Parallelausführung — PARALLEL
- beeinflussen die Query Transformation — NO_MERGE

Syntax

```
{SELECT|UPDATE|DELETE|INSERT} /*+hint */ ...
```

Hints werden ignoriert

- bei anderen als DML-Statements
- bei falscher Schreibweise, richtige im gleichen Statement werden berücksichtigt
- wenn sie sich widersprechen, andere werden verwendet

Dynamische Stichproben

Dynamic Sampling, verfügbar ab Oracle 9i, bedeutet:

Stichproben, mit denen die Statistiken von Segmenten geschätzt werden können, werden zur Laufzeit erstellt.

Wird eingestellt mit dem Parameter

OPTIMIZER_DYNAMIC_SAMPLING = 0 ... 10

Bei 0 ist die Funktion ausgeschaltet, 10 repräsentiert den höchsten Grad dynamischer Stichproben.

Standardwert 1 (10*g* 2) bedeutet, Stichproben werden ausgeführt, wenn

- die Abfrage auf mehrere Tabellen zugreift.
- keine Tabellenstatistiken und kein Index verfügbar ist.
- der Optimizer einen Full Table Scan wählen würde.

Dynamic Sampling empfiehlt sich nur, wenn die Verbesserung des Ausführungsplans den Overhead durch die Stichprobenerhebung deutlich überwiegt.

Histogramme

Problem: Der CBO geht von einer Gleichverteilung der Daten über den Wertebereich aus. Das ist häufig nicht der Fall. Der CBO verschätzt sich dann beim Vergleich der Kosten von Indexzugriff und Full Table Scan (Skewness-Problem).

Lösung: Histogramme

Syntax: ANALYZE TABLE ... FOR COLUMNS

View: DBA_TAB_HISTOGRAMS

Prinzip: Spaltenwerte werden in *Buckets* unterteilt, die in der Höhe ausgeglichen werden. Default sind 75 Buckets.

Beispiel: Verteilung der Werte 1 bis 100 auf 10 Buckets

Daten bezüglich der Werte gleichverteilt

10	20	30	40	50	60	70	80	90	100

ungleich verteilt

5	5	5	5	10	10	20	35	60	100

Planstabilität (Stored Outlines)

Identische Ausführungspläne werden erreicht durch

- Photographisch gleiche SQL
- Hints
- Bindevariablen statt Literale
- Stabilität der Statistiken
- Im Dictionary gespeicherte Ausführungspläne (stored outlines)

Stored Outlines erzeugen:

Init.ora: Parameter CREATE_STORED_OUTLINES = [TRUE|kategorie]

Session: ALTER SESSION SET CREATE_STORED_OUTLINES = [TRUE|kategorie];

Explizit: CREATE OR REPLACE OUTLINE name FOR CATEGORY kategorie ON
SQL-Anweisung;

Stored Outlines verwenden:

Init.ora: Parameter USE_STORED_OUTLINES = [TRUE|kategorie]

Session: ALTER SESSION SET USE_STORED_OUTLINES = [TRUE|kategorie];

Stored Outlines verwalten:

Schema OUTLN, Package OUTLN_PCK, Views DBA_OUTLINES, DBA_OUTLINE_HINTS

Logik der Erstellung von Ausführungsplänen

1. Ist die SQL im Shared Pool? Ja/Nein
   - Ja → **2**
   - Nein → **3**
2. Liegt die gleiche Outlinekategorie vor? Ja/Nein
   - Ja → **5**
   - Nein → **3**
3. Gibt es im Data Dictionary eine übereinstimmende Outline? Ja/Nein
   - Ja → **4**
   - Nein → **6**
4. Die Outline wird integriert und der Plan erstellt. Danach **5**.
5. Der Outline-Plan wird ausgeführt.
6. Normale SQL-Ausführung

Literatur

- R. J. Niemiec: Oracle 9i Performance Tuning: Tips & Techniques. Mc Graw Hill, New York 2003, 826 pp.
- M. Gurry: Oracle SQL Tuning. Pocket Reference. O'Reilly, Sebastopol (CA) 2002, 102 pp.
- K. Floss: Oracle SQL Tuning & CBO Internals. Rampant, Kittrell (NC) 2004, 340 pp.
- G. Powell: Oracle High Performance Tuning for 9i and 10g. Elsevier 2004.
- G. Saake, A. Heuer, K.-U. Sattler: Datenbanken: Implementierungstechniken. mitp, 2. Aufl. Bonn 2005
- T. Härder, E. Rahm: Datenbanksysteme: Konzepte und Techniken der Implementierung. Springer, Berlin Heidelberg New York 2001

Dr. Frank Haney
info@it-haney.de
Tel.: 03641-210224

ORACLE | CERTIFIED PROFESSIONAL