

Performance-Prognosen im Test, trotz Datenschutzauflagen



Daniel Stein

DOAG November 2016

Vorstellung

Motivation

Situation heute

Praxisbeispiele

Fazit & Ausblick

- » 31 Jahre
- » 10+ Jahre Erfahrung Java – JDBC-Entwickler
 - » Mitentwicklung eines Software-Repositorys
- » 3+ Jahre Erfahrung Oracle
 - » intensive Beschäftigung mit dem CBO
- » Bindeglied zwischen Entwicklern und DBAs



- » 1905 gegründet
- » Versicherungsverein auf Gegenseitigkeit
- » Größte private Krankenversicherung
- » Mitarbeiter: mehr als 16.500 (davon über 600 in der IT)

Vorstellung

Motivation

Situation heute

Praxisbeispiele

Fazit & Ausblick

- » §3 BDSG (Bundesdatenschutzgesetz)
 - » Forderung nach Anonymisierung von Kundendaten
- » Anonymisierung aller Produktionsdaten bei der Debeka zu aufwändig
- » Teilmengen der Produktionsdaten - anonymisiert - im Testsystem
- » Der Oracle CBO sieht im Testsystem nicht dieselbe Datenverteilung wie in Produktion
- » Objektstatistiken zu Testdaten

Resultat:

- » **andere Daten => andere Pläne => anderes Laufzeitverhalten!**
- » **Kein Zugriff für Entwickler auf Produktionsmaschine!**

Der Anfang und der Beweis: Der größte anzunehmende Test

- » Optimizer Migration 2016
 - » FIRST_ROWS → ALL_ROWS
 - » da einige Schwierigkeiten mit FIRST_ROWS
 - » (https://jonathanlewis.wordpress.com/?s=first_rows)
 - » Pläne zehntausender Statements ändern sich potentiell
 - » Abschätzung der Auswirkung durch Fachentwickler schwierig, da Datenbank ab SQL bisher Blackbox
 - » Wie Aufgabe möglichst automatisieren?

- » mittels PL/SQL Script in DKS_PROD (parallel zum Betrieb):
 - » Abfrage SQL aus V\$SQL
 - » Erzeugen von Plänen (FIRST_ROWS & ALL_ROWS)
 - » Auswählen von Statements mit abweichenden Plänen (Plan Hash)
 - » **Von den Zehntausenden potentiell anderen Plänen nur noch einige Hundert übrig**

- » **Wie sollen Produktions-Pläne im Test ausreichend gut reproduziert werden?**
 - » Optimizer erstellt Pläne auf Grund der Objekt- und Systemstatistiken
 - » Übertragung Statistiken aus Produktion nach Test
 - » Datapump METADATA_ONLY

DKS_STATS

Originaldaten



DKS_PROD

Anonymisierte
Teilmenge



DKS_INT

User DKS_STATS

Individuelle
Testdaten



DKS_ENTW



Batchjob überträgt
Statistiken



Entwickler

DKS = Debeka-Kern-System

» DKS_INT:

- » **ALTER SESSION SET OPTIMIZER_MODE = ALL_ROWS;**
- » **ALTER SESSION SET CURRENT_SCHEMA = DKS_STATS;**

» Erzeugen und Prüfen von Plänen ausgewählter Statements

1. Prüfen der Kardinalität *(reicht für 90% der Fälle)*

“When cardinalities are incorrectly estimated, the optimizer may choose an inefficient query plan. The No. 1, and some might say only, reason for an inefficient plan’s being generated by the optimizer is inaccurate cardinality estimations. I like to say “right cardinality equals right plan; wrong cardinality equals wrong plan.” Tom Kyte

2. Plan-Aufbau

- Analyse Zugriffspfade (Beispiel später)

- » Abgewandeltes „Tuning by Cardinality Feedback“ (Wolfgang Breitling)

Id	Operation	Name	Starts	E-Rows	A-Rows
PLAN_TABLE_OUTPUT					
1	SORT AGGREGATE		1	1	1
* 2	HASH JOIN		1	26753	374K
* 3	HASH JOIN		1	6688	94395
* 4	TABLE ACCESS FULL	TEAMS	1	146	565
5	TABLE ACCESS FULL	MANAGERS	1	2895	2895
6	TABLE ACCESS FULL	MASTER	1	61400	61400

- » Untersuchung E-Rows und A-Rows auf signifikante Abweichungen
- » DKS_STATS:
 - » A-Rows durch Abschätzung Fachentwickler (Beispiel später)

...zu Schade für die Tonne...

- » Optimizer-Migration dank DKS_STATS erfolgreich
- » Unterstützung Fachentwickler bei der täglichen Arbeit, da bisher ...
 - » ...testen der DB-Anwendungen / Skripte nur auf „logische Richtigkeit“
 - » ...Laufzeitverhalten in Test / Entwicklung nicht aussagekräftig für Produktion
- » Günstige alternative Testumgebung
 - » Keine weitere Hardware wie für DKS_PROD nötig
 - » Keine Software / Lizenzkosten

Vorstellung

Motivation

Situation heute

Praxisbeispiele

Fazit & Ausblick

- » **ALTER SESSION SET CURRENT_SCHEMA = DKS_STATS;**
- » Im „Workflow“ Pläne erzeugen
 - » Wie bei Optimizier-Migration vorgestellt

- » Anmelden bei dem Benutzer DKS_STATS zum Einsehen der Statistiktabelle
 - » per SQL
 - » per Reports
 - » Pläne erzeugen

TABLE_NAME	NUM_ROWS	BLOCKS	MB	LAST_ANALYZED
1 KB_VP	8155269	184040	1437,81	01.07.16
2 KB_VPB	12575086	142062	1109,86	26.11.15
3 KB_VN	4686205	132948	1038,66	16.10.16
4 KB_VNB	5811506	68390	534,3	16.10.16
5 KB_VERW	3990116	12760	99,69	01.07.16

TABLE_NAME	NUM_ROWS	BLOCKS	MB	LAST_ANALYZED
1 KB_VP	8155269	184040	1437,81	01.07.16
2 KB_VPB	12575086	142062	1109,86	26.11.15
3 KB_VN	4686205	132948	1038,66	16.10.16
4 KB_VNB	5811506	68390	534,3	16.10.16
5 KB_VERW	3990116	12760	99,69	01.07.16

Spalten Report
 Index-Report

COLUMN_NAME	DATA_TYPE	DATA_LENGTH	DURSCHNITTLÄNGE	NUM_DISTINCT	NUM_NULLS	UNGLEICHVERTEILUNG	NUM_BUCKETS	SAMPLE_SIZE	AVG_ROWS	DENSITY
1 K40_41_K40VERWNR	NUMBER	22	6	4024320	0	NONE	1	4686205	1,16	0,00000024849
2 K41ADAT	NUMBER	22	4	10942	0	NONE	1	4686205	428,28	0,00009139097
3 K41AUSF	NUMBER	22	4	692	0	NONE	1	4686205	6771,97	0,00144508671
4 K41AVNB	NUMBER	22	3	15	0	NONE	1	4686205	312413,67	0,06666666667
5 K41BEG	NUMBER	22	5	10588	0	NONE	1	4686205	442,6	0,00009444654
6 K41BN	NUMBER	22	3	17	0	FREQUENCY	10	5593	329	0,00000010492
7 K41BSL	NUMBER	22	3	16	0	NONE	1	4686205	292887,81	0,0625
8 K41BUMO	NUMBER	22	4	346	0	NONE	1	4686205	13543,95	0,00289017341
9 K41BV	NUMBER	22	3	17	0	FREQUENCY	17	5592	328,94	0,00000010494

Vorstellung

Motivation

Situation heute

Praxisbeispiele

Fazit & Ausblick

INDEX_NAME	COLUMN_NAME	POS	SORT	UNIQUENESS	BLEVEL	LEAF_BLOCKS
1 PK_KB_VN	K41_RECNO	1	ASC	UNIQUE	2	12485
2 UC_KB_VN_AREA_RECNO	K41_AREA_NAME	1	ASC	NONUNIQUE	2	13502
3 UC_KB_VN_AREA_RECNO	K41_RECNO	2	ASC	NONUNIQUE	2	13502
4 UC_KB_VN_K40_41	K41_AREA_NAME	1	ASC	NONUNIQUE	2	17214
5 UC_KB_VN_K40_41	K40_41_K40VERWNR	2	ASC	NONUNIQUE	2	17214
6 UC_KB_VN_K40_41	K41_K40_41_SEQNUM	3	ASC	NONUNIQUE	2	17214

TBL_BLOCKS	ABST_BLOCKS_CLSF	CLSF	ABST_CLSF_ROWS	NUM_ROWS	NUM_DISTINCT	COMPRESSION	LAST_ANALYZED
132948	740614	873562	3812643	4686205	4686205	DISABLED	16.10.16
132948	-4075	128873	4557332	4686205	10	DISABLED	16.10.16
132948	-4075	128873	4557332	4686205	4686205	DISABLED	16.10.16
132948	4193261	4326209	359996	4686205	10	DISABLED	16.10.16
132948	4193261	4326209	359996	4686205	4024320	DISABLED	16.10.16
132948	4193261	4326209	359996	4686205	4686205	DISABLED	16.10.16

- » Kandidat für Komprimierung?
- » Clustering Factor-Analyse
- » Spaltenreihenfolge bei zusammengesetzten Indizes
 - » Index-Verwendung?

Aufspüren inkorrektter Statistiken

```
SELECT *  
FROM KB_VERW  
JOIN KB_VN ON K40_AREA_NAME = K41_AREA_NAME  
           AND K40VERWNR = K40_41_K40VERWNR  
           AND K41VART = 1  
  
...  
WHERE K41_AREA_NAME= '01' -- PARTITIONSSCHLÜSSEL CHAR-COLUMN  
...;
```

- » Fachentwickler A-Rows : >100k Zeilen pro Table Access
- » Entwickler interessiert sich nur für Partition 1

Aufspüren inkorrektter Statistiken

Id	Operation	Name	Rows	Cost (%CPU)	Pstart	Pstop
0	SELECT STATEMENT		1	8 (0)		
1	NESTED LOOPS		1	8 (0)		
2	NESTED LOOPS		1	8 (0)		
3	PARTITION LIST SINGLE		1	4 (0)	1	1
4	TABLE ACCESS BY LOCAL INDEX ROWID	KB_VERW	1	4 (0)	1	1
* 5	INDEX RANGE SCAN	PK_KB_VERW	1	3 (0)	1	1
6	PARTITION LIST SINGLE		1	2 (0)	1	1
* 7	INDEX RANGE SCAN	UC_KB_VN_K40_41	1	2 (0)	1	1
* 8	TABLE ACCESS BY LOCAL INDEX ROWID	KB_VN	1	4 (0)	1	1

» inkorrekte Partitionsstatistiken?

» (Erwartung >100k Rows)

Aufspüren inkorrektener Statistiken

Id	Operation	Name	Rows	Cost (%CPU)	Pstart	Pstop
0	SELECT STATEMENT		114K	38423 (2)		
1	PARTITION LIST ALL		114K	38423 (2)	1	10
* 2	HASH JOIN		114K	38423 (2)		
* 3	TABLE ACCESS FULL	KB_VN	115K	28899 (2)	1	10
4	TABLE ACCESS FULL	KB_VERW	3990K	2865 (3)	1	10

Predicate Information (identified by operation id):

```
2 - access("K40_AREA_NAME"="K41_AREA_NAME" AND "K40VERWNR"="K40_41_K40VERWNR")
3 - filter("K41VART"=1 AND TO_NUMBER("KB_VN"."K41_AREA_NAME")=01)
```

- » `K41_AREA_NAME = 01`
 - » (testweise) um globale Statistiken zu sehen.
- » Hier passen die Kardinalitäten
 - » Schätzung Fachentwickler gar nicht schlecht

Aufspüren inkorrektener Statistiken

Id	Operation	Name	Rows	Cost (%CPU)	Pstart	Pstop
0	SELECT STATEMENT		1	8 (0)		
1	NESTED LOOPS		1	8 (0)		
2	NESTED LOOPS		1	8 (0)		
3	PARTITION LIST SINGLE		1	4 (0)	1	1
4	TABLE ACCESS BY LOCAL INDEX ROWID	KB_VERW	1	4 (0)	1	1
* 5	INDEX RANGE SCAN	PK_KB_VERW	1	3 (0)	1	1
6	PARTITION LIST SINGLE		1	2 (0)	1	1
* 7	INDEX RANGE SCAN	UC_KB_VN_K40_41	1	2 (0)	1	1
* 8	TABLE ACCESS BY LOCAL INDEX ROWID	KB_VN	1	4 (0)	1	1

» **inkorrekte Partitionsstatistiken!**

- » (Erwartung >100k Rows)
- » Nested Loop: für jeden Satz KB_VERW einmal in KB_VN einsteigen
- » außerdem ist eine Kardinalität von 1 meistens verdächtig

Aufspüren inkorrektener Statistiken

Id	Operation	Name	Rows	Cost (%CPU)	Pstart	Pstop
0	SELECT STATEMENT		116K	4672 (2)		
* 1	HASH JOIN		116K	4672 (2)		
2	PART JOIN FILTER CREATE	:BF0000	399K	296 (6)		
3	PARTITION LIST SINGLE		399K	296 (6)	1	1
4	TABLE ACCESS FULL	KB_VERW	399K	296 (6)	1	1
5	PARTITION LIST SINGLE		116K	2736 (2)	KEY(AP)	KEY(AP)
* 6	TABLE ACCESS FULL	KB_VN	116K	2736 (2)	1	1

Predicate Information (identified by operation id):

```

1 - access("K40_AREA_NAME"="K41_AREA_NAME" AND "K40VERWNR"="K40_41_K40VERWNR")
6 - filter("K41VART"=1)

```

- » Untersuchung der Partitionsstatistiken brachte falsche Statistiken zu Tage
- » Hash Join
- » Dieser Plan ist für das vorgestellte Statement effizienter

Ineffizientes Statement

```
SELECT *
FROM KB_VERW
LEFT JOIN KB_VN ON K40_AREA_NAME = K41_AREA_NAME
                AND K40VERWNR = K40_41_K40VERWNR
                AND K41VART = 1
LEFT JOIN KA_VN ON K41VERTRNR = K21VERTRNR
                AND K21_RECNO IN (
                SELECT K21_30_RECNO
                FROM KA_VNS
                WHERE K30SGR = 23
                );
```

Ineffizientes Statement

Id	Operation	Name	Rows	Cost (%CPU)
0	SELECT STATEMENT		3990K	2250M (1)
1	NESTED LOOPS OUTER		3990K	2250M (1)
2	PARTITION LIST ALL		3990K	38891 (2)
* 3	HASH JOIN RIGHT OUTER		3990K	38891 (2)
* 4	TABLE ACCESS FULL	KB_VN	1158K	28765 (2)
5	TABLE ACCESS FULL	KB_VERW	3990K	2865 (3)
6	VIEW		1	564 (1)
7	NESTED LOOPS SEMI		1	564 (1)
* 8	TABLE ACCESS FULL	KA_VN	1	561 (1)
* 9	TABLE ACCESS BY INDEX ROWID	KA_VNS	1117	3 (0)
* 10	INDEX RANGE SCAN	UC_KA_VNS_K21_30	1	2 (0)

- » Driving für Operation Nested Loop
- » Probe für Operation Nested Loop
- » Nested Loop Algo: Für jeden Satz aus der Driving wird die Probe einmal ausgeführt
- » Erwartung (DKS_STATS) : Aufruf View (id 6) ~ 4 Mio Mal


```
SELECT *
FROM KB_VERW
LEFT JOIN KB_VN ON K40_AREA_NAME = K41_AREA_NAME
                AND K40VERWNR = K40_41_K40VERWNR
                AND K41VART = 1
LEFT JOIN (
    SELECT K21VERTRNR
    FROM KA_VN
    JOIN KA_VNS ON K21_RECNO = K21_30_RECNO
    WHERE K30SGR = 23 -- KA_VNS EIN SATZ SGR23 PRO KA VN!
) V0 ON K41VERTRNR = K21VERTRNR;
```

- » Dieses Statement bringt logisch das gleiche Ergebnis
- » Information von Fachentwickler:
 - » Ein Satz mit K30SGR = 23 (KA_VNS) pro Satz aus KA_VN!
(1:1 Beziehung)

Ineffizientes Statement

Id	Operation	Name	Rows	Cost (%CPU)
0	SELECT STATEMENT		3990K	40578 (2)
* 1	HASH JOIN RIGHT OUTER		3990K	40578 (2)
2	VIEW		1117	1656 (1)
* 3	HASH JOIN		1117	1656 (1)
* 4	TABLE ACCESS FULL	KA_VNS	1117	1093 (1)
5	TABLE ACCESS FULL	KA_VN	26374	563 (1)
6	PARTITION LIST ALL		3990K	38891 (2)
* 7	HASH JOIN RIGHT OUTER		3990K	38891 (2)
* 8	TABLE ACCESS FULL	KB_VN	1158K	28765 (2)
9	TABLE ACCESS FULL	KB_VERW	3990K	2865 (3)

Cost vorher:
2250M

- » Driving für Operation Hash Join
- » Probe für Operation Hash Join
- » Hash Join Algo: Baue Hash Table (Driving) und führe Probe einmal daran vorbei
- » Erwartung (DKS_STATS) : Cost und Laufzeit geringer

- » Cost-Abschätzung rekursiver Abfragen... Oracle nimmt eine bestimmte Tiefe an ...
 - » Nicht rekursives Gegenstück war „teurer“, lief aber schneller.
- » Keine Unterstützung durch dynamic sampling
- » Weitere Details im Manuskript

Vorstellung

Motivation

Situation heute

Praxisbeispiele

Fazit und Ausblick

- » In der verfügbaren Zeit konnte nur ein kleiner Ausschnitt gezeigt werden
 - » Weitere Reports und Tool-Integrationen geplant
 - » Automatisierte Prüfung auf Performance-Engpässe in Plänen
 - » DOAG 2017?

- » Statistiken - Reports - und die dahinter verborgene Fachlichkeit sind gemeinsame Gesprächsgrundlage für DBAS und Entwickler

- » „Jonathan Lewis: Cost Based Fundamentals“
 - » *This book is, well, in a word – amazing. If you have ever been baffled or bemused by “why the heck did the optimizer do that – Tom Kyte*

Vielen Dank für Ihre Aufmerksamkeit!

Debeka

Versichern und Bausparen

anders als andere

Daniel Stein, daniel.stein@debeka.de, www.debeka.de